04.04.2024 Emir Arda Bayer
22201832-4

Lab 6 – Arbitrary Waveform Generator

Purpose

The purpose of this lab is to get students along with an IP called Clocking Wizard to create arbitrary digital signals of chosen frequency and duty cycles. By making use of the IP users had the opportunity to modify such qualities of waves such as frequency and duty cycle, and create desired waves with manipulated temporal qualities in order to create the necessary signals for using equipment, especially ones that could be used in projects such as using servo motors.

Design

The design was compromised of a single clock_5_mhz module that converted the input 100 MHz signal to 5 MHz using Clocking Wizard IP, it had a single input ‘clocked2initial’ for the internal 100 MHz clock and a single output ‘output_s’ for outputting the desired wave. The architecture included a component created by the IP, and signals regarding registers to hold the new created signals and outputs of the IP’s finite state machine, and a counter to count according to the outputted clock and create desired waves that have desired duty cycles. A process basically checked the rising edge of the outputted clock and created arbitrary pulse waves according to the desired duty cycles and counter values.

Methodology and Results

After creating the inputs and outputs of the system I implemented the IP’s component and process. The wizard takes the input 100 MHz signal, that is initialized to the IP at the setup of the wizard, and outputs a signal with the decided frequency and duty cycle, this time 5 Mhz and an unchanged 50 percent duty cycle. 2 registers hold the outputs of the wizard, one being ‘registermemory’ that carries the 5 MHz signal, and one called ‘locked2’, that basically holds the unused ‘locked’ output of the IP. Later, a process regarding the ‘registermemory’ was created, one beforehand created signal called ‘counter’ counted every cycle of this clock, and its value was used in an if statement in the process. The change of its value to arbitrary chosen values created waves with chosen duty cycles, chosen high and low values for these signals to be later used in various kinds of digital design projects. At each incrementation, the if statements functioned and changed the output signal to low or high, and the values in the if conditions decided the length of high and low times. As the module was complete, a testbench was created to observe the healthy function of the system, and the RTL Schematics were checked. Later a bitstream was generated to use the code on Basys 3. On the constraint file, the pin J2 was used to observe the created signal in analog circuitry, on an oscilloscope using jumpers and a oscilloscope probe. The output analog wave, the physical design, the simulation results and RTL Schematic are in Figures 1-5.

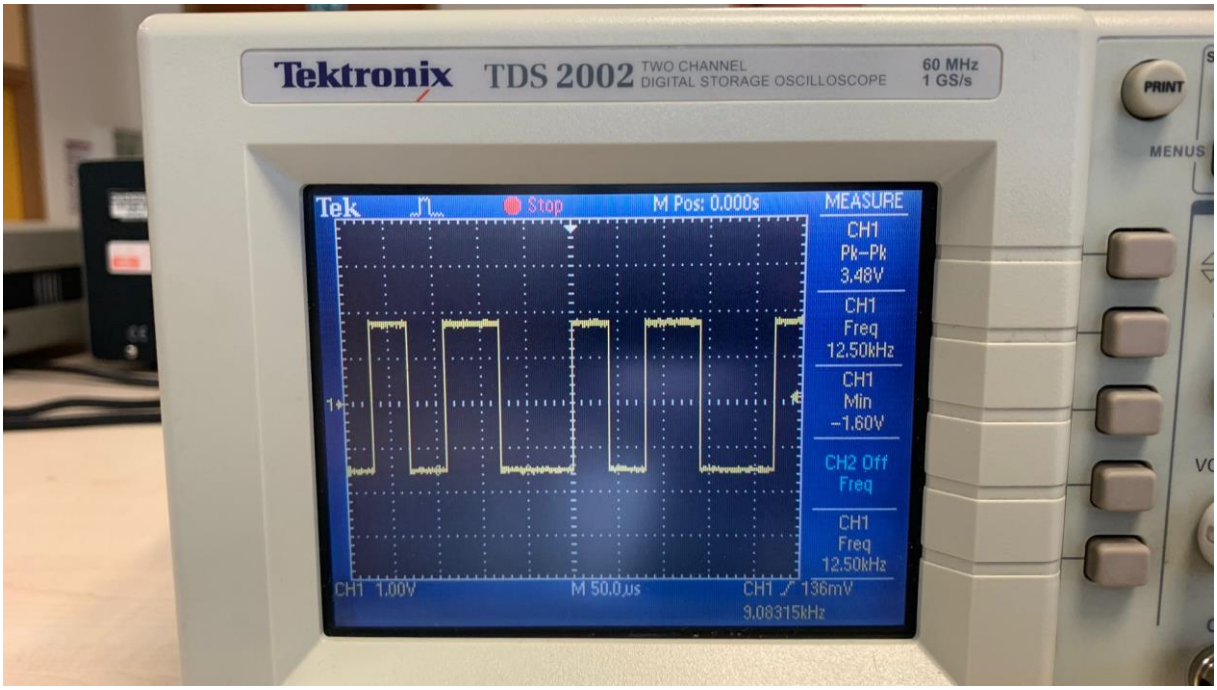

Figure 1: Output Wave

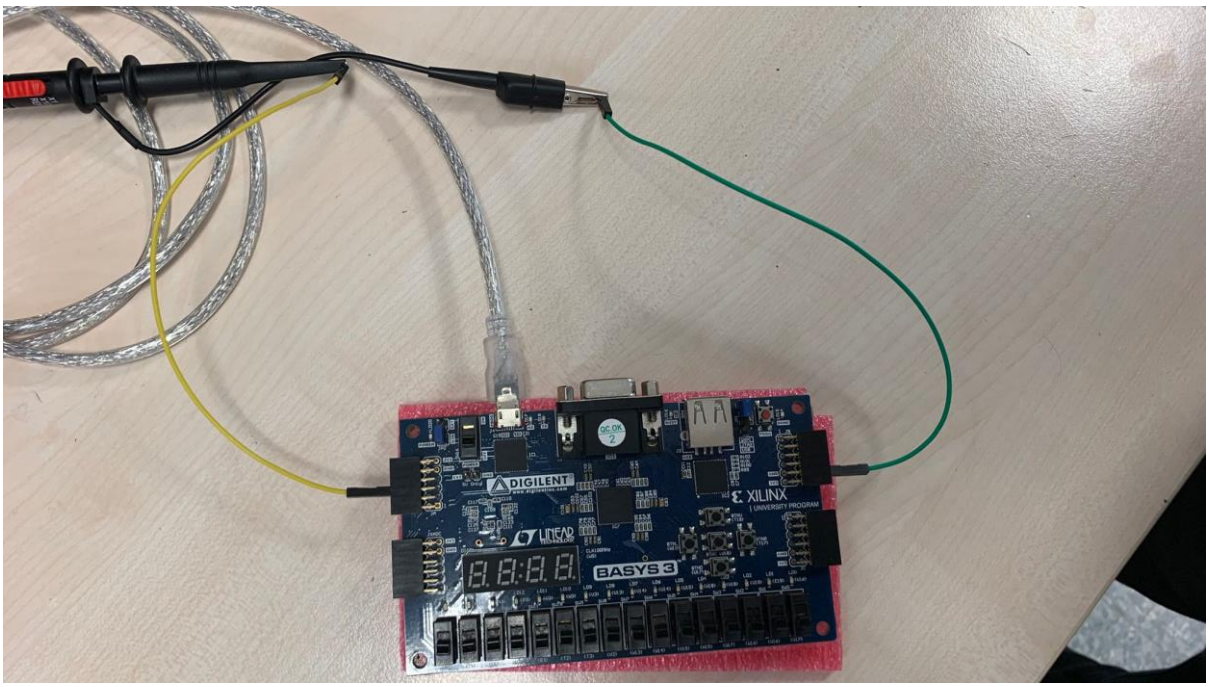

Figure 2: The Implementation on Basys 3 Pins

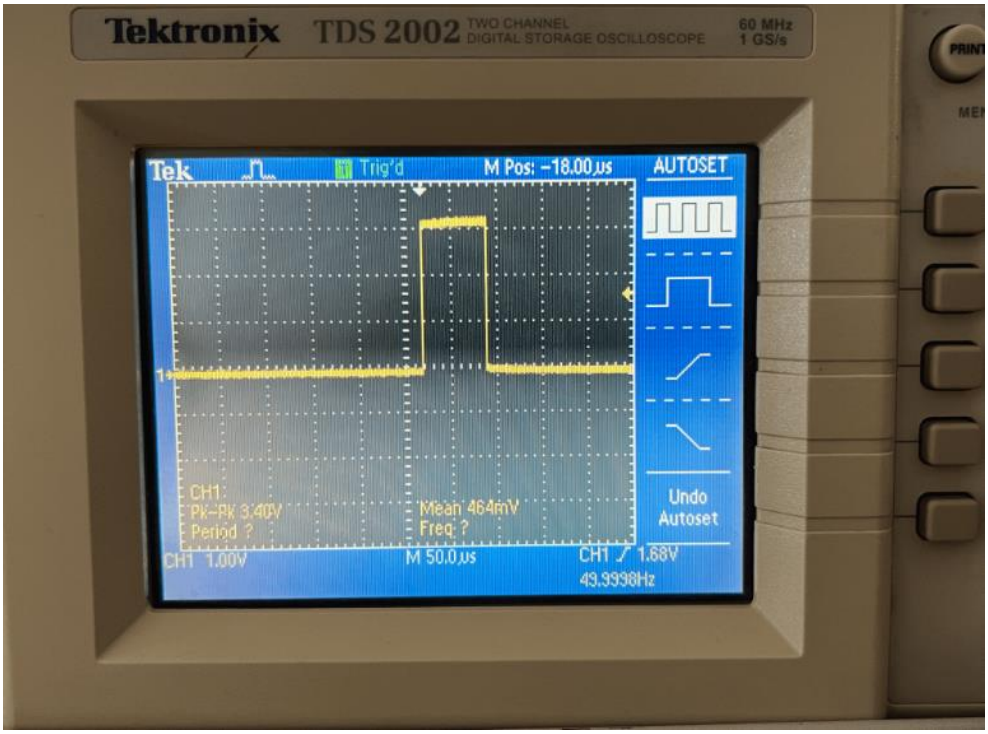

Figure 3: A Single Pulse

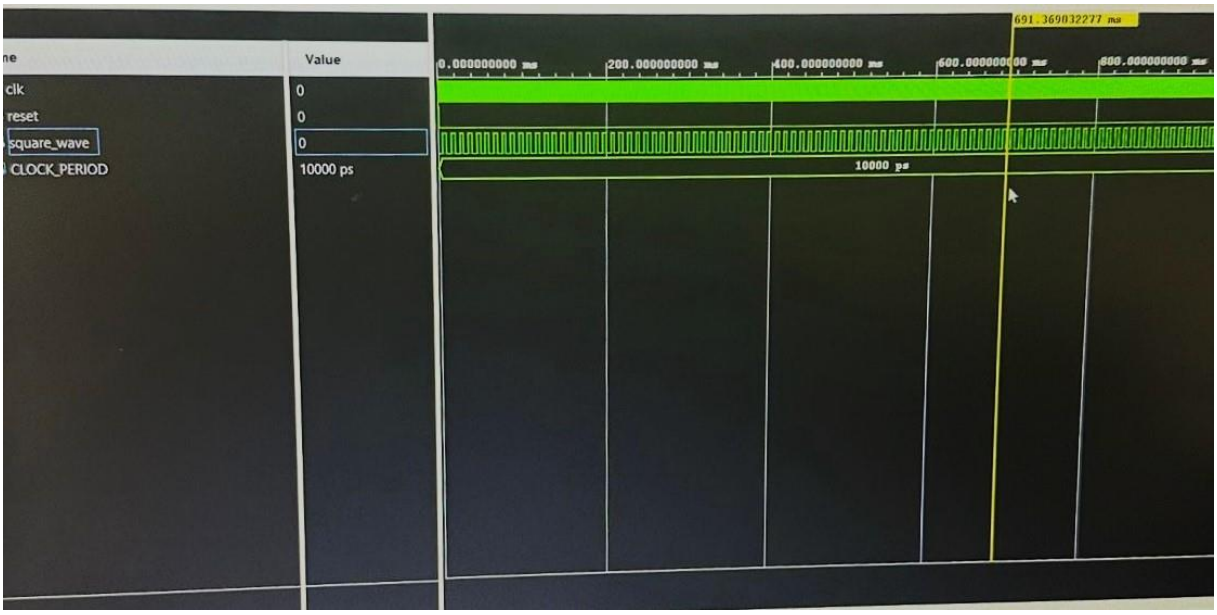

Figure 4: Simulation Results

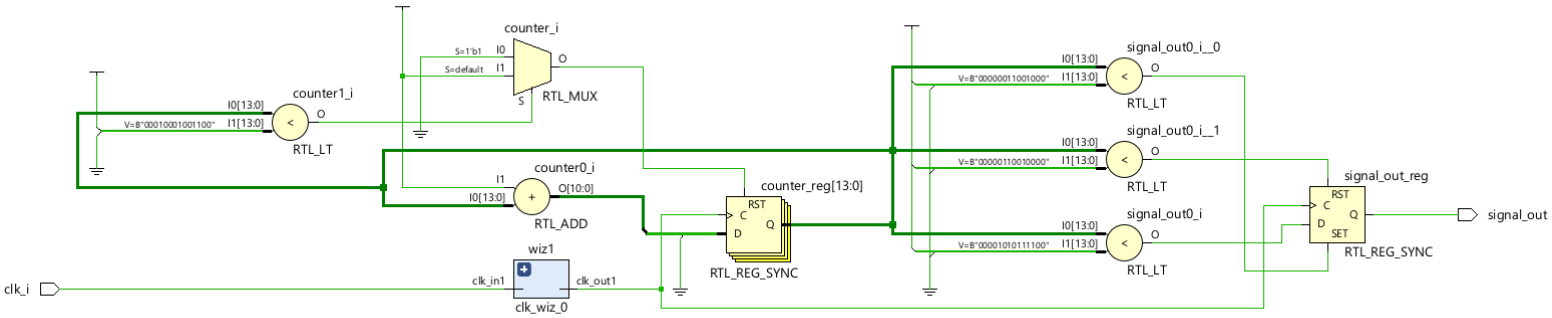

Figure 5: RTL Schematic of the Design

Conclusion

The goal of the sixth lab was to introduce students to IPs, get them along with creating clocks, and processes that depend upon them, like their rising and falling edges, and manipulate clock signals through their frequencies and duty cycles, or individual time length of pulses' high and low values. I had to make online research about the Clocking Wizard IP, about implementing it on my digital design, and had to understand how its effect on design source's architecture, how its existence changes the processes in which clocks are intended to be used. I had to rethink about how internal signals could create to carry the internal signals that are not outputted through the system, and how they could be implemented in the processes. I made mistakes in my test bench and could not observe the outputted final signal of the system, but actually saw the outputs of the IP first, but I later solved this problem by changing the signals and their types, and how they are used in the test bench processes. The sixth lab was useful in terms of understanding how digital signals could be manipulated for further use in electronic circuitry, in possible projects and how electronic components function. I also significantly improved my researching capabilities in this lab.

Appendices

clocked25mhz.vhd

```
library IEEE; -- Importing IEEE standard library

use IEEE.STD_LOGIC_1164.ALL; -- Using standard logic data types

use IEEE.numeric_std.ALL; -- Using numeric standard package
```

```
entity clocked25mhz is -- Entity declaration for the 'clocked25mhz' module
    Port (
        clocked2initial      : in STD_LOGIC; -- Input port for the 100MHz clock
        output_s      : out std_logic -- Output port for the signal
        );
end clocked25mhz; -- End of entity declaration

architecture Behavioral of clocked25mhz is -- Architecture declaration for the 'clocked25mhz' module

--Wizard begin
component clk_wiz_0 -- Component declaration for the clock wizard
port
 (-- Clocked2 input ports
  -- Clocked2 output ports
  clk_out1          : out    std_logic; -- Output port for the clock
  -- Status and control signals
  reset             : in     std_logic; -- Input port for reset signal
  locked2ed            : out    std_logic; -- Output port indicating the lock status
  clocked2initialn1           : in     std_logic -- Input port for the 100MHz clock
 );

end component; -- End of clock wizard component declaration
--Wizard end

signal registermemory : std_logic := '0'; -- Signal to hold the clock output
signal locked2 : std_logic; -- Signal to hold the lock status
signal counter : integer range 0 to 10000 := 0; -- Counter for signal generation
```

```
begin  -- Beginning of architecture implementation

wiz1: clk_wiz_0 -- Instantiation of the clock wizard component

   port map(
          clocked2initialn1  => clocked2initial, -- Mapping input clock port
          clk_out1 => registermemory, -- Mapping clock output port
          reset=>'0', -- Setting reset to low
          locked2ed => locked2 -- Mapping lock status output port
       );

process(registermemory) -- Process to generate output signal based on clock edges
begin
if rising_edge(registermemory) then -- Checking for rising clock edge
       counter <= counter +1; -- Incrementing counter


       output_s<='0'; -- Initializing output signal to '0'


   if counter < 200 then -- Condition to set output signal to '1'
      output_s<='1';
   elsif counter <400 then -- Condition to set output signal to '0'
       output_s<='0';
   elsif counter < 700 then -- Condition to set output signal to '1'
    output_s<='1';
   elsif counter < 1100 then -- Condition to set output signal to '0'
             output_s<='0';
   else
     counter <= 0;  -- Resetting counter when it reaches maximum value
```

```
    end if;
end if;
end process;

end Behavioral; -- End of architecture implementation
```

tb_clocked25mhz.vhd

```
library IEEE; -- Importing IEEE standard library
use IEEE.STD_LOGIC_1164.ALL; -- Using standard logic data types
use IEEE.NUMERIC_STD.ALL; -- Using numeric standard package

entity tb_clocked25mhz is -- Entity declaration for the testbench
end tb_clocked25mhz; -- End of entity declaration

architecture testbench of tb_clocked25mhz is -- Architecture declaration for the testbench

    -- Signals
    signal clocked2initial_tb : std_logic := '0'; -- Signal for the 100MHz clock input
    signal clk_o_tb : std_logic; -- Signal for the 5MHz output clock
    signal clocked2initial, output_s : std_logic; -- Signals for clocked25mhz module inputs and
outputs

    component clocked25mhz -- Component declaration for the clocked25mhz module
        Port (
            clocked2initial : in STD_LOGIC; -- Input port for the 100MHz clock
            output_s : out std_logic -- Output port for the 5MHz clock
        );
```

```
    end component; -- End of clocked25mhz component declaration

begin  -- Beginning of architecture implementation

    -- Instantiating the clocked25mhz module
    uut: entity work.clocked25mhz
        port map (
            clocked2initial => clocked2initial_tb,
            output_s => clk_o_tb
        );

    -- Process for generating the 100MHz clock
    clk_100MHz_process: process
    begin
        clocked2initial_tb <= not clocked2initial_tb; -- Inverting the clock signal
        wait for 5 ns; -- Waiting for 5 ns
    end process clk_100MHz_process; -- End of clock process

end architecture testbench; -- End of architecture implementation
```

pinss.xdc

```
set_property IOSTANDARD LVCMOS33 [get_ports clocked2initial]
set_property IOSTANDARD LVCMOS33 [get_ports output_s]
set_property PACKAGE_PIN W5 [get_ports clocked2initial]
set_property PACKAGE_PIN J2 [get_ports output_s]
```