Report: Arbitrary Waveform Generator

Bilkent University Electrical and Electronics Department

EE102-01 Lab 6

Melih Ege Doğan – 22403329
April 7, 2025

Purpose:

Purpose of this lab is to design a VHDL code that generates arbitrary digital waveform using clock wizard IP and observe it in both testbench and oscilloscope.

Methodology:

This lab focused on generating a precise, arbitrary digital waveform using VHDL on an FPGA. The initial step involved configuring the Clocking Wizard IP core in the Vivado environment. This IP leverages Phase-Locked Loops (PLLs) to synthesize clean, user-defined clock frequencies. For this design, a 100 MHz clock was generated to serve as the main system clock. While the Clocking Wizard was also configured to output a 25 MHz clock, the waveform generation logic exclusively utilized the 100 MHz clock to avoid clock domain crossing issues, as recommended. Following clock generation, VHDL code was written to produce the target waveform. The waveform specification required a cycle consisting of:

1. Logic HIGH for 400 ns
2. Logic LOW for 800 ns
3. Logic HIGH for 160 ns
4. Logic LOW for 240 ns

The total period of this waveform is 1600 ns (400 + 800 + 160 + 240). To implement this timing accurately, a counter was employed within a single VHDL process sensitive to the rising edge of the 25 MHz clock.

Design Specifications:

This VHDL module (main.vhd) implements an arbitrary digital waveform generator. It utilizes the Xilinx Clocking Wizard IP (clk_wiz_0) to generate two clock frequencies (100 MHz and 25 MHz) from a single input clock. The 25 MHz clock is

then used internally as the synchronous clock source for a state machine that produces a specific repeating digital pattern on the output pin o.

Port Name	Direction	Type	Description
clk_in1	Input	std_logic	Primary clock input source for the Clocking Wizard IP (FPGA oscillator). The frequency of this clock determines the wizard's ability to generate the target outputs.
reset	Input	std_logic	Asynchronous reset input for the Clocking Wizard IP. When asserted, it resets the wizard's internal PLL/MMCM.
CLK_100MHZ	Output	std_logic	100 MHz clock signal generated by the Clocking Wizard IP. Not used internally by the waveform generation logic in this architecture.
CLK_25MHZ	Output	std_logic	25 MHz clock signal generated by the Clocking Wizard IP. This clock **is** used as the synchronous clock for the waveform generation logic.

locked	Output	std_logic	Status signal from the Clocking Wizard IP. Indicates when the internal PLL/MMCM has achieved frequency and phase lock, signifying stable clock outputs.
o	Output	std_logic	The generated arbitrary digital waveform output signal.

Results:

The designed arbitrary waveform generator was verified through both simulation using a VHDL testbench and physical measurement using an oscilloscope after implementation on the FPGA. The objective was to generate a repeating waveform with the following sequence based on a 25 MHz clock:160 ns HIGH, 240 ns LOW, 400 ns HIGH, 800 ns LOW, resulting in a total period of 1600 ns.

A testbench was created to drive the clk_in1 signal and monitor the generated outputs. Figure 1 shows a screenshot of the simulation waveform for the output signal o and the driving 25 MHz clock (CLK_25MHZ).

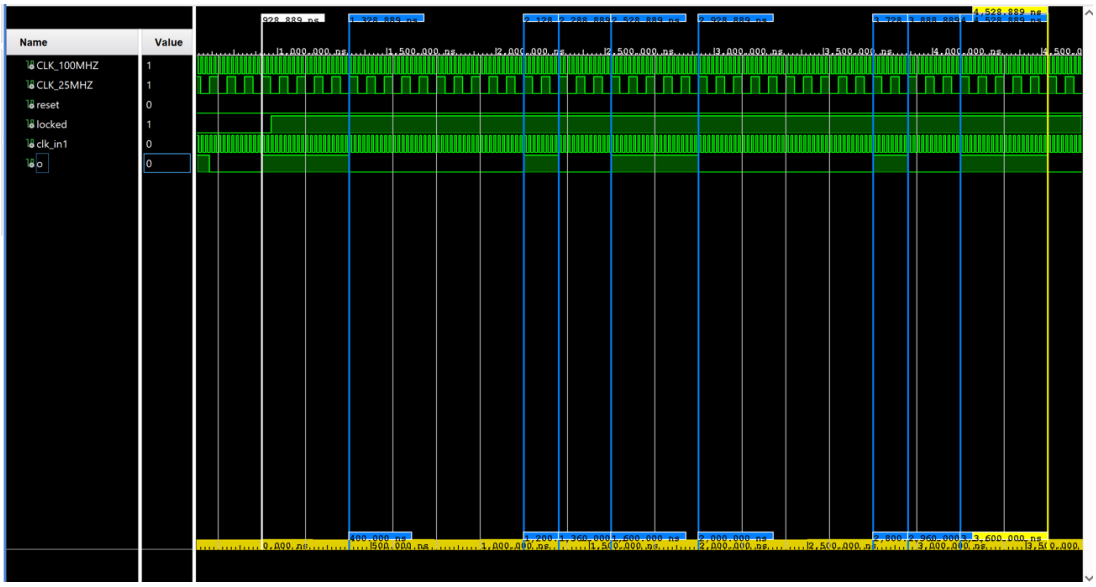

Figure 1: Testbench Simulation.

The simulation clearly shows a repeating digital pattern consistent with the four states defined in the VHDL code. Using the simulation tool's measurement cursors, the following durations were precisely measured for one cycle of the output signal o:

- First Logic HIGH duration: **160 ns**
- First Logic LOW duration: **240 ns**
- Second Logic HIGH duration: **400 ns**
- Second Logic LOW duration: **800 ns**

The design was synthesized, implemented, and downloaded to the Basys3 FPGA board. The output signal o was connected to an oscilloscope. Figure 2 shows the waveform captured by the oscilloscope. As you can see from the picture it is the same waveform with simulation as expected.

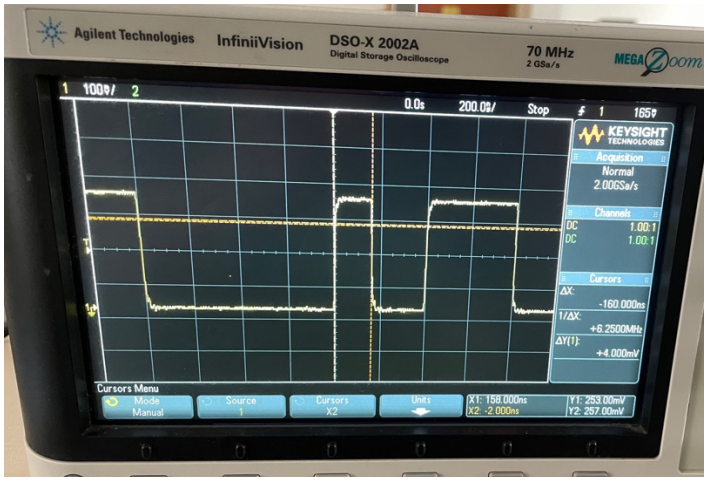

Figure 2: Waveform on Oscilloscope.

Conclusion:

In conclusion, the objective of designing and verifying an arbitrary waveform generator was successfully achieved. The VHDL implementation, leveraging a state machine synchronized to a 25 MHz clock generated by the Xilinx Clocking Wizard IP, proved capable of producing the specified output pattern. Simulation results provided ideal verification, showing exact adherence to the target timings of 160 ns (H), 240 ns (L), 400 ns (H), and 800 ns (L) for a total period of 1600 ns. Subsequent hardware testing and oscilloscope measurements corroborated these findings, displaying the correct waveform shape with measured durations closely aligning with the specifications, considering minor real-world deviations. This lab effectively showcased the process of precise digital timing generation on an FPGA using standard IP cores and synchronous VHDL design practices.

Appendices:

Code 1: main.vhd

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

entity main is
    Port (
        CLK_100MHZ: out std_logic;
        CLK_25MHZ: out std_logic;
        reset: in std_logic;
        locked:out std_logic;
        clk_in1: in std_logic;
        o: out std_logic

    );
end main;

architecture Behavioral of main is

TYPE t is (first,second,third,last);
signal state : t := first;

signal counter : integer := 1;

component clk_wiz_0
port
 (-- Clock in ports
  -- Clock out ports
  CLK_100MHZ          : out    std_logic;
  CLK_25MHZ          : out    std_logic;
  -- Status and control signals
  reset             : in     std_logic;
  locked            : out    std_logic;
  clk_in1           : in     std_logic
 );
end component;

SIGNAL clock: std_logic;
begin
clk25 : clk_wiz_0
   port map (
  -- Clock out ports
   CLK_100MHZ => CLK_100MHZ,
   CLK_25MHZ => clock,
  -- Status and control signals
   reset => reset,
   locked => locked,
   -- Clock in ports
```

```
   clk_in1 => clk_in1
 );
 CLK_25MHZ <= clock;


process(clock)
begin
    if rising_edge(clock) then
        case state is
            when first =>
                if counter = 4 then

                    state <= second;
                    counter <= 1;

                else
                    counter <= counter + 1;
                    o <= '1';
                end if;

            when second =>
                if counter = 6 then
                    state <= third;
                    counter <= 1;

                else
                    counter <= counter + 1;
                    o <= '0';

                end if;
            when third =>
                if counter = 10 then

                    state <= last;
                    counter <= 1;

                else
                counter <= counter + 1;
                    o <= '1';


                end if;
            when last=>
                if counter = 20 then

                    state <= first;
                    counter <= 1;
                else

                    counter <= counter + 1;
                    o <= '0';
```

```
                end if;

        end case;
    end if;

end process;



end Behavioral;
```

Code 2: clock.vhd

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

entity main_tb is
    -
end entity main_tb;

architecture behavior of main_tb is
SIGNAL   CLK_100MHZ: std_logic;
SIGNAL   CLK_25MHZ: std_logic;
SIGNAL   reset: std_logic;
SIGNAL   locked: std_logic;
SIGNAL   clk_in1: std_logic;
SIGNAL   o: std_logic;
begin

dut: entity work.main
 PORT MAP( CLK_100MHZ => CLK_100MHZ, CLK_25MHZ => CLK_25MHZ, reset => reset, locked
=>
locked, clk_in1 => clk_in1, o => o);

clock_process: process
begin
    clk_in1 <= '0';
    wait for 10ns;
    clk_in1 <= '1';
    wait for 10ns;
end process;
```

```
stim_proc: process begin
    reset <= '1';
    wait for 20 ns;
    reset <= '0';
    wait;
end process;
```

```
end architecture behavior;
```

References:

- https://github.com/SemihAkkoc/EEE102
- https://www.youtube.com/watch?v=ngkpvMaNapA&ab_channel=AnasSalahEddin