Lab 6: Arbitrary Waveform Generator

Mert Örnek EEE 102-3 22302052 05.04.2024

Purpose

The purpose of this experiment is to attempt to build a setup that generates arbitrary waveforms. This process includes expanding on our practical and theoretical understanding of how a clock divider works as well as learning how to utilize pre-programed IP's available on Vivado, such as the Clock Wizard IP.

Design Specifications

For the purposes of this experiment, I chose to generate the types of waveforms necessary to utilize an SG90 servo motor. An SG90 servo motor works on the following premise: It receives signals with 20ms periods with varying "high" times to determine the angle towards which it is instructed to turn. These peak values vary between 1ms and 2ms. The values between these angles are converted to angle values between 0 degrees and 180 degrees based on the high time, such as 1ms being 0 degrees, 2ms being 180 degrees, 1.5 ms being 90 degrees and 1.25 ms being 45 degrees.

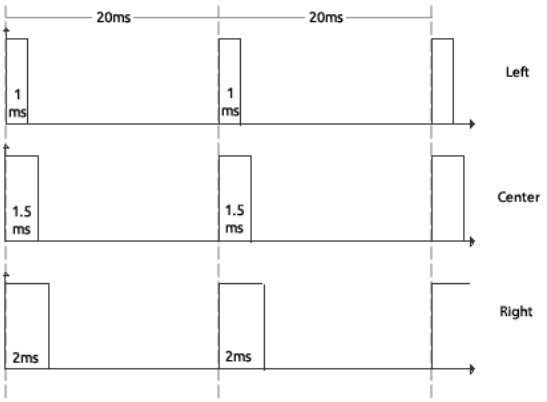

Figure 1: Waveforms used to control an SG90 servo motor.

The setup I intend to implement consists of a top module (top_module.vhd), which is connected to a "Clock Wizard IP" (clk_wiz_0.xci) module and five variations of the signal adjuster (sg90_servo_signal_adjuster.vhd) module. The top module has two inputs and two outputs: The inputs are 3 of the switches(sw) found on Basys 3 as well as the built-in 100MHz clock (clk) , and the outputs are 4 LEDs (LED) to indicate the active switches and waveform and a single JA series PMOD pin (output_port) . The top module uses the least significant index of the switch as an on/off button, in the sense that the theoretical servo maintains a 90 degree angle unless the switch is turned on. When the least significant switch is turned on, the other two remaining switches are used to determine one of four angle values: "00" for 135 degrees right, "10" for 180 degrees right, "01" for 45 degrees left and "11" for 0 degrees left. These rotation angles sequentially convert to the following duty cycles: 7.5%, 8.75%, 10%, 6.25% and 5%. These variations in duty cycles determine the current angle the servo motor turns towards. In this rotation definition, the words "right" and "left" denote the new position of the servo with respect to the center axis of 90 degrees and not the direction in which it is actively turning. The clock input is considered to be the default clock input of the Basys 3 at 100MHz. The LED outputs are directly connected to the corresponding pins situated above their respective switches with the exception of LED (3), which indicates the operational status of the clock output by relaying that signal into that specific LED.

The setup also includes a “Clock Wizard IP” as a submodule. This submodule uses the default clock input of the top module (clk_in1) to generate the same 100MHz signal (clk_out1). The sole purpose of this module was so that a “cleaner” clock signal could be obtained as claimed by the lab manual. Here, it is worth noting that this module was generated in Verilog. This property did not seem to affect its functionality as the submodule of a VHDL file.

The third component of the setup is in the form of the arbitrary signal adjuster (sg90_servo_signal_adjuster.vhd). This module has three inputs and an output. The inputs are the input clock considered to be at 100MHz, a reset button and an integer value between 0 and 200. The integer value serves the purpose of specifying the period of the “high” signal for the arbitrary wave. For instance, an integer value of 150 would translate to a high period of 1.50 ms and a value of 200 would be 2.00 ms. This module was built by using the clock divider constructed for the previous lab as a starting point and modifying it in a way that used the counter to identify both the periods for the “high” and the “low” signals. For this, there is a constant and arbitrary number limit: The constant value is “2000000 – 1”, which signifies the 20ms net period of the generated waveforms, and the arbitrary value is obtained by multiplying the input integer by 1000. If the input reset signal is “1”, the system gives a constant low signal. Else, it will start counting up every rising edge of the input clock. The system gives out a high signal until the arbitrary value is reached, after which it will give a low signal until the counter hits the constant maximum value and resets.

The testbench (top_module_TB.vhd) has no individual inputs or outputs and uses the top module as its sole submodule. It uses a while loop to switch back and forth with 5ns periods between a binary value to simulate the 100MHz internal clock of the Basys 3. The simulation has a net runtime of 250000000ns due to the low frequency of the simulated waveform (50Hz). The switch inputs are each simulated for 40000000ns, sequentially going from “000”, “001”, “101”, “011”, “111”. These circumstances give the simulation a chance to trial each waveform a few times.

The assigned pins of the constraint file are as follows:

Clock: clk: W5

Switches: sw (from index 0 to 2): V17, V16, W16

LEDs: LED (from index 0 to 3): U16, E19, U19, V19

JA1 PMOD pin: output_port: L2

Methodology

The methodology of this Lab session consists of building an arbitrary signal generator in modular fashion and then demonstrating its operational condition through the Vivado simulation feature and implementing it on a Basys3 and connecting it to an oscilloscope. For the lab, any type of arbitrary signal was allowed, and it was recommended that we could generate signals related to our term projects. As a result, I chose to generate the necessary signal outputs to operate an SG90 servo.

Results

The results of this experiment will be observed in two sections: The simulation of the experiment on Vivado and observing the signal it generates on an oscilloscope via Basys 3 implementation.

The simulation of this experiment tested the following switch inputs: “000”, “001”, “101”, “011” and “111”. The least significant digits of these values represent the on/off inputs explained in the design specifications. The other two digits represent the following integer inputs respectively: 150, 175, 200, 125, 100. This means that what we expect to see on the simulation for each of these combinations is a square wave with an end to end period of 20ms and a high period of the respective integer divided by 100, as in: 1.50, 1.75, 2.00, 1.25 and 1.00 respectively. These periods sequentially convert to the following duty cycles: 7.5%, 8.75%, 10%, 6.25% and 5%. The switch inputs effectively alter the duty cycle to control the servo motor’s rotation.

Setup 1: sw: “000”

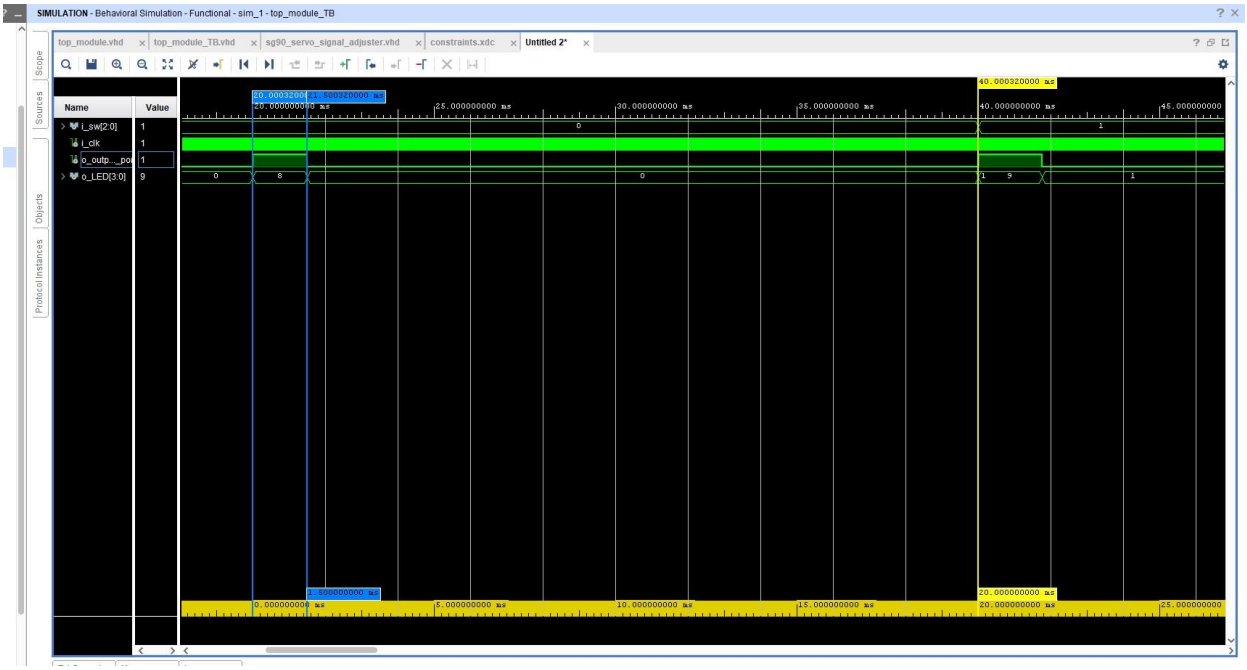

Figure 2.1: input sw: “000”

This figure shows a square wave with a high period of 1.50 ms and a total period of 20ms, as can be observed by the period markers below the screen.

Setup 2: sw: “001”

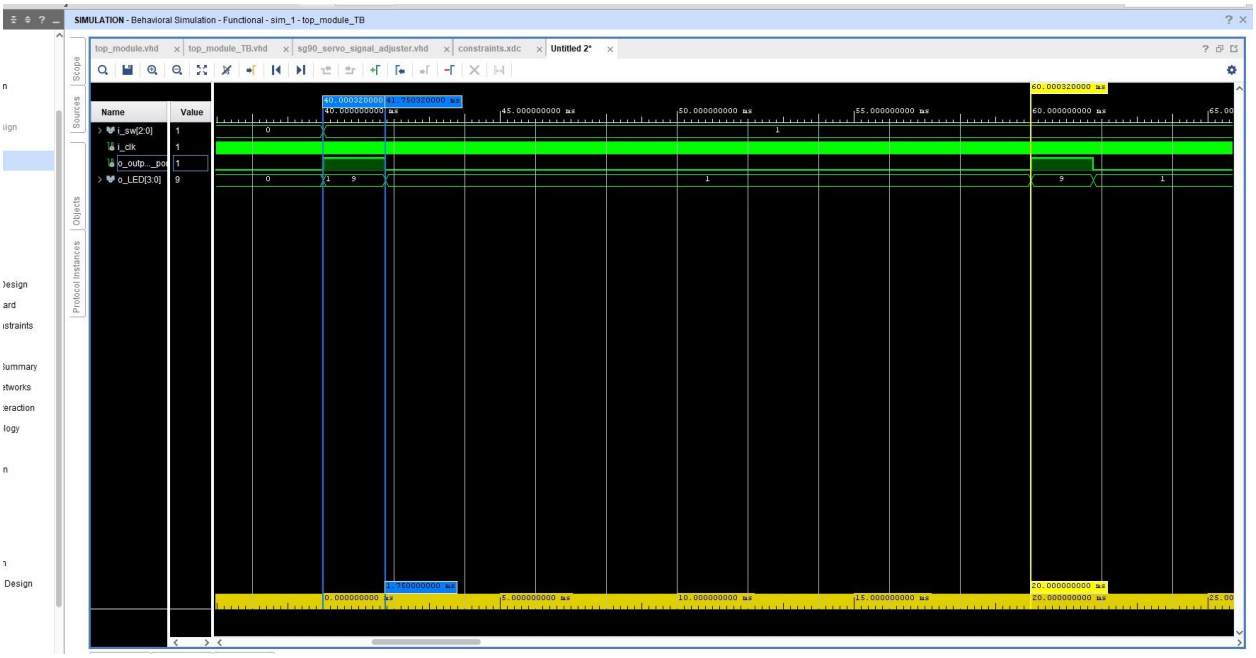

Figure 2.2: input sw: "001"

This figure shows a square wave with a high period of 1.75 ms and a total period of 20ms, as can be observed by the period markers below the screen.

Setup 3: sw: "101"

Figure 2.3: input sw: "101"

This figure shows a square wave with a high period of 2.00 ms and a total period of 20ms, as can be observed by the period markers below the screen.

Setup 4: sw: “011”

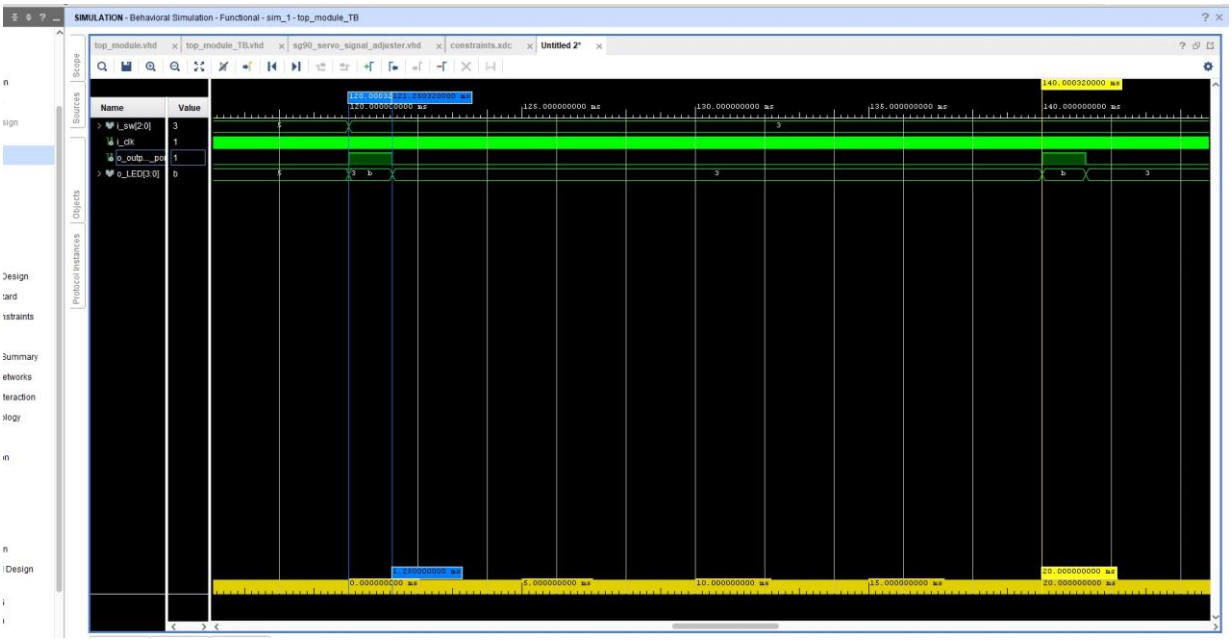

Figure 2.4: input sw: “011”

This figure shows a square wave with a high period of 1.25 ms and a total period of 20ms, as can be observed by the period markers below the screen.

Setup 5: sw: “111”

Figure 2.5: input sw: “111”

This figure shows a square wave with a high period of 1.00 ms and a total period of 20ms, as can be observed by the period markers below the screen.

Overall, these simulation results match the setup we set out to implement in the design specifications. Note the cursors below the simulation results indicating the measured periods of both the period of the high (‘1’) signal as well as the overall period of the waveform.

In order to measure the signal generated via oscilloscope, two jumper wires were connected to the L2 and ground pins of the Basys 3 and connected to the oscilloscope. Each one of the switch inputs was trialed and their high signal periods and frequencies were measured. The results were as follows:

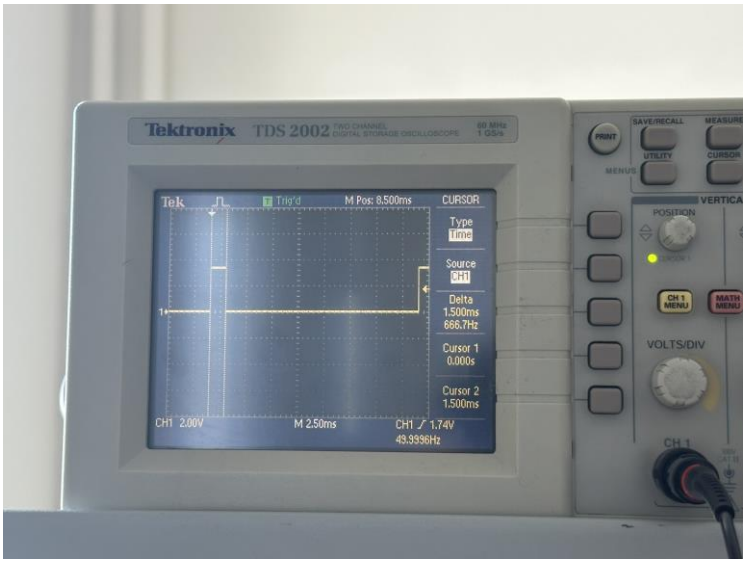

Figure 3.1: sw “000”

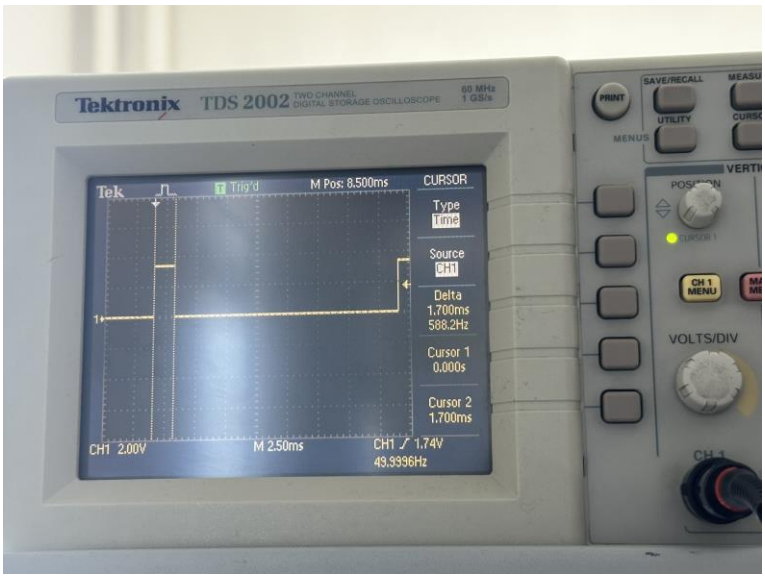

Figure 3.2: sw “001”

Figure 3.3: sw "101"

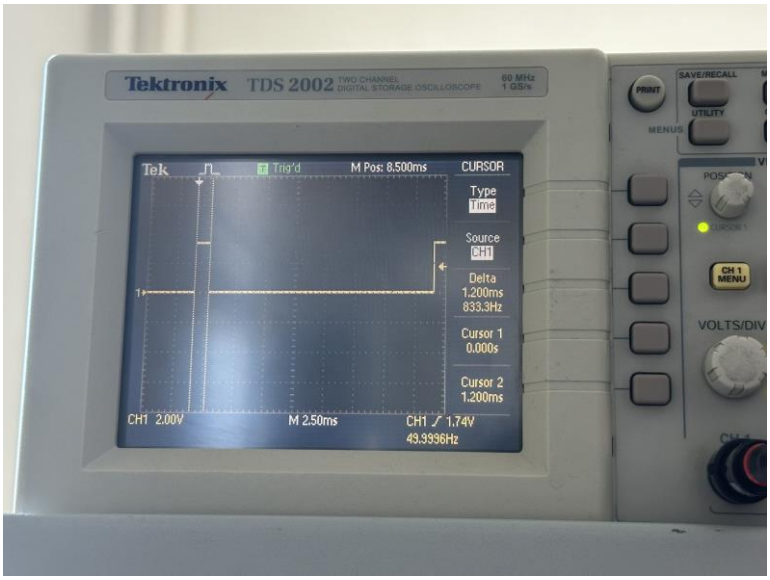

Figure 3.4: sw "011"

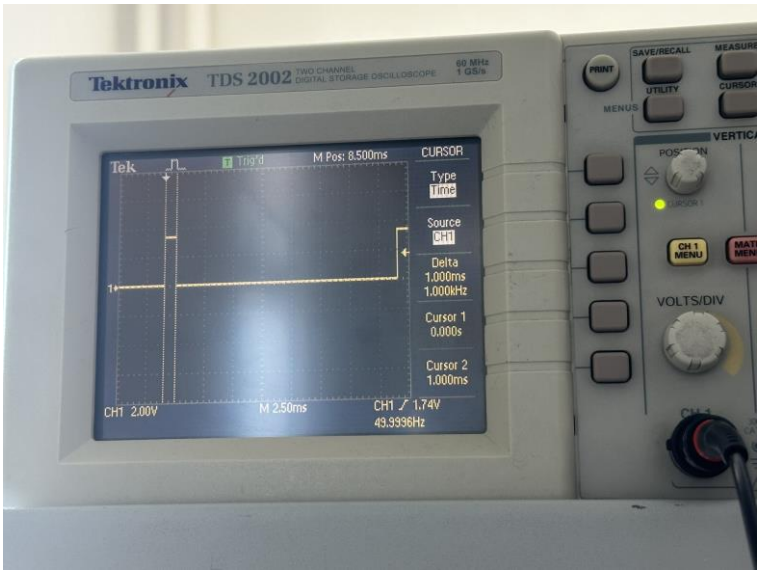

Figure 3.5: sw "111"

Each one of these waves were measured to have a frequency of around 50Hz (20ms period) by the oscilloscope and their respective high signal periods were as follows: 1.5 ms, 1.7 ms, 2.0 ms, 1.2 ms and 1.0 ms respectively. Here, it is worth noting that the cursors on the oscilloscope could not measure further than the 1st decimal digit, meaning that waveforms which were observed to have high signal periods of 1.25ms and 1.75ms were measured to have periods of 1.2ms and 1.7ms respectively. Outside of this marginal error, the periods measured via the oscilloscope replicate the results obtained through the Vivado's simulation feature.

Conclusion

The purpose of this experiment was to learn how to generate arbitrarily chosen waveforms by using VHDL and checking if it worked by using means such as Vivado simulations and measuring the signal output of the Basys 3 via an oscilloscope. The code was implemented in modular fashion via VHDL, and the setup specified in the design specifications was implemented and proven to be performing as intended. Overall, the task criteria of this experiment were met which is why this lab session can be considered as successful. One of the things that I learned in this experiment was how to measure simulation waveforms' periods on VHDL by using markers. Another thing that was new to me was using the PMOD pins on the Basys 3. It was beneficial for me to get accustomed to using these pins as they will be used extensively for the term project. The concept of a "duty cycle" was among the things I learned by this lab session. One thing that proved to be a challenge was generating the simulated waveform as, due to the low frequency of the setup and relatively large number of input combinations, it took a considerable amount of time for Vivado to generate the necessary number of simulation combinations. Outside of that, the experiment did not encounter any other major issues.

Appendices

Code of the Arbitrary Waveform Generator

top_module.vhd :

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

entity top_module is
   Port (sw: in std_logic_vector(2 downto 0);
   clk: in std_logic;
   LED : out std_logic_vector (3 downto 0);
   output_port: out std_logic);
```

```
end top_module;


--sw pins represent the following binary inputs:
--sw(0) represents the online(1)/offline(0) status of the SG90 servo motor       => 1.50 ms in 20ms periods as default
--sw(1 to 2): 00 means 135 degrees to the right                                  => 1.75 ms in 20ms periods
--sw(1 to 2): 01 means all the way to the right                                  => 2.00 ms in 20ms periods
--sw(1 to 2): 10 means 135 degrees to the left                                   => 1.25 ms in 20ms periods
--sw(1 to 2): 11 means all the way to the left                                   => 1.00 ms in 20ms periods

architecture Behavioral of top_module is

component clk_wiz_0 is
 Port(clk_in1: in std_logic;
 reset: in std_logic := '0';
 locked: out std_logic;
 clk_out1: out std_logic);
 end component;



component sg90_servo_signal_adjuster is
   Port ( i_clk : in STD_LOGIC;                    -- assumed to be 100MHz as is std Basys 3 clk
   i_res : in std_logic;                           -- reset
   i_high: in integer range 0 to 200 := 0;         --high period: i_high/100
   o_clk: out std_logic);
end component;

signal s_default: std_logic;
signal s_one: std_logic;
signal s_two: std_logic;
signal s_three: std_logic;
signal s_four: std_logic;
```

```
signal s_resdef: std_logic;

signal s_clkout: std_logic := '0';

signal s_clk_wiz: std_logic;

begin


clk_wiz: clk_wiz_0
   Port Map( clk_in1 => clk,
   clk_out1 => s_clk_wiz);



sig_def: sg90_servo_signal_adjuster          -- will send a single waveform when s(0) = '0' then shut off
      Port Map ( i_clk => s_clk_wiz,
   i_res => s_resdef,
   i_high => 150,
   o_clk => s_default);

sig_one: sg90_servo_signal_adjuster
      Port Map ( i_clk => s_clk_wiz,
   i_res => '0',
   i_high => 175,
   o_clk => s_one);

sig_two: sg90_servo_signal_adjuster
      Port Map ( i_clk => s_clk_wiz,
   i_res => '0',
   i_high => 200,
   o_clk => s_two);
```

```
sig_three: sg90_servo_signal_adjuster
      Port Map ( i_clk => s_clk_wiz,
   i_res => '0',
   i_high => 125,
   o_clk => s_three);

sig_four: sg90_servo_signal_adjuster
      Port Map ( i_clk => s_clk_wiz,
   i_res => '0',
   i_high => 100,
   o_clk => s_four);

process (sw, s_default, s_one, s_two, s_three, s_four)
begin

if sw(0) = '0' then
   s_clkout <= s_default;
   s_resdef <= '0';
else
   s_resdef <= '1';
   if sw(2 downto 1) = "00" then
      s_clkout <= s_one;
   elsif sw(2 downto 1) = "10" then
      s_clkout <= s_two;
   elsif sw(2 downto 1) = "01" then
      s_clkout <= s_three;
   elsif sw(2 downto 1) = "11" then
      s_clkout <= s_four;

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

```
LED (2 downto 0) <= sw;
output_port <= s_clkout;
LED(3) <= s_clkout;

end Behavioral;
```

sg90_servo_signal_adjuster.vhd :

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

entity sg90_servo_signal_adjuster is
   Port ( i_clk : in STD_LOGIC;                -- assumed to be 100MHz as is std Basys 3 clk
   i_res : in std_logic;                       -- reset
   i_high: in integer range 0 to 200 := 0;     --high period: i_high/100
   o_clk: out std_logic);
end sg90_servo_signal_adjuster;

architecture Behavioral of sg90_servo_signal_adjuster is

signal s_counttemp: integer := 0;
signal s_countfixed: integer := 0;  -- 50 Hz
signal s_countselected: integer := 0;

signal s_restart : std_logic := '0';
signal s_highreset : std_logic := '0';

signal s_temp: std_logic := '0';
```

```
begin


s_countselected <= (1000*i_high) ;
s_countfixed <= 2000000 - 1 ;


process(i_clk)
begin
   if (i_res = '1') then
      s_counttemp <= 0;
      s_temp <= '0';
   elsif rising_edge(i_clk) then
      if (s_counttemp = s_countselected) then
         s_temp <= '0';
      elsif (s_counttemp = s_countfixed) then
         s_counttemp <= 0;
         s_temp <= '0';
      elsif (s_counttemp < s_countselected) then
         s_temp <= '1';
      end if;


      if (s_counttemp /= s_countfixed) then
         s_counttemp <= s_counttemp + 1;
      end if;
   end if;
end process;


o_clk <= s_temp;


end Behavioral;
```

top_module_TB.vhd :

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

entity top_module_TB is
end top_module_TB;

architecture Behavioral of top_module_TB is

component top_module is
    Port (sw: in std_logic_vector(2 downto 0);
    clk: in std_logic;
    LED : out std_logic_vector (3 downto 0);
    output_port: out std_logic);
end component;

signal i_sw: std_logic_vector(2 downto 0) := (others => '0');
signal i_clk: std_logic := '0';
signal o_output_port: std_logic := '0';
signal o_LED :std_logic_vector(3 downto 0):= (others => '0');

begin

gen_clk: process
begin
  while now < 990000000 ns loop
     i_clk <= not i_clk;
     wait for 5 ns;
  end loop;
  wait;
end process gen_clk;

top_mod: top_module
port map (sw => i_sw,
```

```
    clk => i_clk,
    LED => o_LED,
    output_port => o_output_port);

stimulus_process: process
begin
    wait for 10ns;
    i_sw <= "000";
    wait for 40000000ns;
    i_sw <= "001";
    wait for 40000000ns;
    i_sw <= "101";
    wait for 40000000ns;
    i_sw <= "011";
    wait for 40000000ns;
    i_sw <= "111";
    wait;
end process stimulus_process;

end Behavioral;
```

>"Clocking Wizard IP" code found on Vivado in Verilog form was utilized in this lab

Works Cited

Ramos, Carlos A. "Servomotor Control With PWM and VHDL." *CodeProject*, 20 Dec. 2012, www.codeproject.com/Articles/513169/Servomotor-Control-with-PWM-and-VHDL.