GHENT UNIVERSITY

Embedded Systems: Hardware Synthesis

# Servo PWM Controller

Ghent University

Joachim Cardoen - 01911655

2020

# Summary

This document provides a step-by-step description of the project assignment for the course of Embedded Systems: Hardware Synthesis. The goal of this project is to establish a design of a PWM controller that is able to control the position of a servo motor. Firstly, the specification of the project and its components is discussed, followed by the actual implementation in VHDL using QuestaSim, consequently the implemented design is analyzed in XILINX Vivado Design Suite. Finally a review of the project is provided in which the implemented design is compared to the prescribed requirements.

# Contents

# Chapter 1

# Project Specification

The goal of this project is create a Servo Controller in VHDL. The specification of the project is split into three parts and can be consulted in full in [2]

1. Design: block diagram;
2. Implementation: necessary functions;
3. Testing and verification;

## 1.1 Design

### 1.1.1 Servo Motor

The servo motor needs to be controller using a PWM signal that indicates its position in the interval of $[-\frac{\pi}{2}, \frac{\pi}{2}]rad$n, where 0x00 equals to $-\frac{\pi}{2}$ and 0xFF to $\frac{\pi}{2}$. The specification states a data resolution of 8 bits, which results in 256 possible positions for the servo motor to be in.

#### PWM Signal

PWM or Pulse Width Modulation in full, is a method to use a percentage to set a certain value for a power application. This percentage is called the duty cycle, this indicates the percentage ($\frac{T_{on}}{T_{total}}$) of a pulse that is considered to be at high logic level. In this project, PWM is used to set the position of a servo motor (albeit theoretically), conforming to the ranges and resolution as stated above. [1]

### 1.1.2 Servo Controller Ports

The servo controller has the following ports:

- Clk: the general system clock (with period T of 20ms);
- Rst: asynchronous reset;
- Servo clk: clock used as a carrier for the PWM servo signal;
- Set: flag that requests an update on the PWM signal;
- Address/Data: shared 8-bit bus holding address of the servo controller and the set value for the servo;

- Done: output signal indicating the servo controller is ready to perform an operation
- PWM signal;

Note: all signals are active high.

### 1.1.3 Timing

**System and servo clocks**

As stated in the previous section, the PWM signal to set a specific position of the servo motor, is built using a carrier signal. The frequency of the PWM signal clock can be calculated as followed:

$$f = (\frac{range}{resolution})^{-1} = (\frac{0.5ms}{256})^{-1} = 512kHz$$

The offset position to obtain a starting time of 1.25 ms can simply be calculated by dividing this value with the clock period of the 512 kHz servo clock.

$$offset = \frac{1.25ms}{2us} = 640$$

After calculating the offset period, the maximum period can be calculated accordingly with the system clock of 20ms, which results in 10240.

**General system timing**

The timing diagram below illustrates the functioning of the servocontroller under normal circumstances.

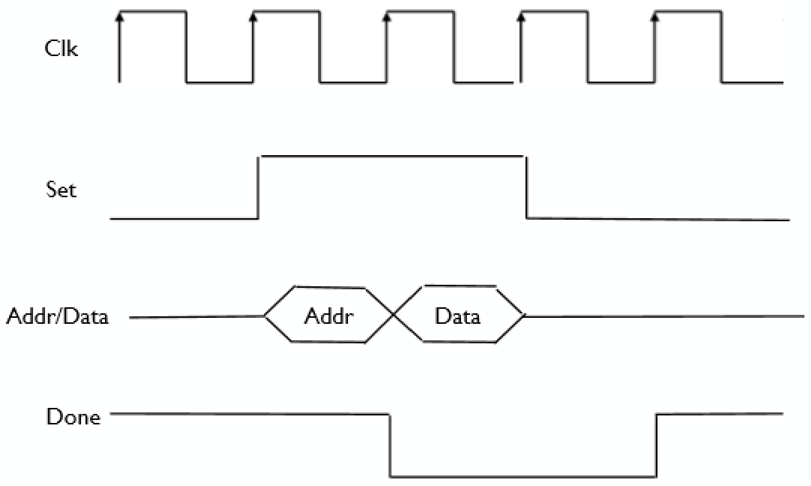


Figure 1.1: Single extract from the timing diagram of the servocontroller

The description of each system clock cycle is provided in the implementation section.

## 1.2 Implementation

The following requirements are set up front:

- Provide asynchronous reset;

- Implement address checking: unicast and broadcast address;
- Necessary timing control flags: done, set, reset;
- Generate PWM signal according to position provided in addrdata line;
- Analysis in Vivado;

## 1.3 Testbench

### 1.3.1 Verification

The verification of the design checks if it implements the necessary requirements. The goal of this project is to provide and assert several test cases that verify the implemented design. The different test cases are listed in the following section.

### 1.3.2 Test cases

The verification test cases are as follows:

1. Verify correct functioning of the outputs done and PWM signal;
2. Verify asynchronous reset;
3. Verify unicast and broadcast address of the PWM controller;
4. Verify value of the PWM signal to the asserted value;

To generate the clocks needed for the testbench, a package has been created, which contains the following code [3]:

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

PACKAGE clocks_pkg IS
    PROCEDURE clock(
        SIGNAL clk : INOUT std_logic;
        CONSTANT period : IN TIME;
        SIGNAL end_simulation : IN BOOLEAN
    );
END PACKAGE clocks_pkg;

PACKAGE BODY clocks_pkg IS
    PROCEDURE clock(SIGNAL clk : INOUT std_logic;
    CONSTANT period : IN TIME; SIGNAL end_simulation : IN BOOLEAN) IS
    BEGIN
        LOOP
            EXIT WHEN end_simulation;
            clk <= NOT clk;
            WAIT FOR period/2;
        END LOOP;
    END PROCEDURE clock;
END PACKAGE BODY clocks_pkg;
```

# Chapter 2

# Project Implementation

## 2.1 Components

After examining the project specification, the following components are deducted:

- General top-level component;
- PWM Counter to build the PWM signal using the servo clock and position;

## 2.2 Top-level PWM Controller

The top-level component handles the actual servo motor. It contains the necessary output and input signals. This component controls the actions, input and output. The description of the entity is as follows:

```
ENTITY PWM_Controller IS
    GENERIC (address : std_logic_vector(7 DOWNTO 0) := "00000001");
    PORT (
        rst : IN std_logic;
        clk : IN std_logic;
        servo_clk : IN std_logic;
        set : IN std_logic;
        done : OUT std_logic;
        addrdata : IN std_logic_vector(7 DOWNTO 0) := (OTHERS => '0');
        pwm : OUT std_logic);
END PWM_Controller;
```

The address of the PWM Controller is provided by using a generic, which contains a default value as to eliminate the need to instantiate it in the testbenches.

### Architecture Overview

The following signals are present in the architecture of the top-level component:

```
    SIGNAL addr_is_read : BOOLEAN := FALSE;
    SIGNAL data_is_read : BOOLEAN := FALSE;
    SIGNAL data_read : std_logic_vector(7 DOWNTO 0);
    SIGNAL addr_correct : INTEGER := 0;
```

- addr _is _read: indicated whether or not the address has been read;
- data_is_read: indicates whether or not the data (ergo position) has been read;
- data_read: signal to hold the position, reset behavior is on position "10000000" (conforms to 1.5ms or 0 rad);
- addr _correct: status flag to indicate if a right or wrong address has been supplied;

The architecture also contains a component for the pwm_counter entity, this results in the following architecture:

```
ARCHITECTURE behavioral OF PWM_Controller IS
    COMPONENT PWM_Counter IS
        PORT (
            rst : IN std_logic;
            servo_clk : IN std_logic;
            position : IN std_logic_vector(7 DOWNTO 0);
            pwm : OUT std_logic);
    END COMPONENT;

    SIGNAL addr_is_read : BOOLEAN := FALSE;
    SIGNAL data_is_read : BOOLEAN := FALSE;
    SIGNAL is_addr_servo : BOOLEAN := FALSE;
    SIGNAL data_read : std_logic_vector(7 DOWNTO 0);
    SIGNAL addr_correct : INTEGER := 0;
BEGIN
    pwm_counter_map : PWM_Counter PORT MAP(
        rst => rst, servo_clk => servo_clk, position => data_read, pwm => pwm
    );

    -- Processes
END ARCHITECTURE;
```

For ease-of-use and integration with XILINX Vivado, the broadcast address has been hardcoded in a package: pwm_pk. Contents of the package:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

PACKAGE pwm_pk IS
    CONSTANT BROADCAST_ADDR : std_logic_vector(7 DOWNTO 0) := "11111111";
    CONSTANT UNICAST_ADDR : std_logic_vector(7 DOWNTO 0) := "00000001";
    -- UNICAST_ADDR is deprecated since using the generic approach
END PACKAGE;
```

As stated above, the control flow is contained inside this component. It checks if the address has been read and matches the address of the specific servo motor or the broadcast address, sets and unsets the required flags during operation. The following process handles this flow:

```
PROCESS (rst, clk) -- start process on change of rst, clk
    BEGIN
        IF (rst = '1') THEN
            addr_is_read <= FALSE;
            data_is_read <= FALSE;
            data_read <= "10000000";
            done <= '1';
            addr_correct <= 0;
        ELSIF rising_edge (clk) THEN
            IF (set = '1') THEN
                -- first clock pulse: read addr
                -- second clock pulse: read data and set done to zero
                IF (addrdata = BROADCAST_ADDR OR addrdata = address)
                AND addr_correct = 0 THEN
                    addr_is_read <= TRUE;
                    addr_correct <= 1;
                    done <= '0';
                ELSE
                    IF addr_correct = 0 THEN
                        addr_correct <= 2;
                    END IF;

                    IF addr_is_read = TRUE AND addr_correct = 1 THEN
                        -- read data
                        data_is_read <= TRUE;
                        data_read <= addrdata;
                        done <= '0';
                    ELSE
                        done <= '1';
                    END IF;
                    IF data_is_read = TRUE THEN
                        done <= '1';
                        addr_correct <= 0;
                    END IF;
                END IF;
            ELSE
                done <= '1';
            END IF;
        END IF;
    END PROCESS;
```

### 2.2.1 PWM Controller Testbench

#### Test case 1: basic behavior

A testbench is created to test the basic behavior of the full cycle of constructing a PWM signal. The description of the visual waveform provided below, divided into clock cycles, is as follows:

- 1st cycle: Initial state where set is L and reset is H, PWM signal stays at zero-position (1.50 ms).
- 2nd cycle: Broadcast/unicast (tested with both) is sent to the controller with set flag logic level high, PWM signal remains at zero-position;
- 3rd cycle: Data (00000000) is sent over the addrdata line, done is being set logic level low to indicate the PWM signal being built;
- 4th cycle: PWM signal is built with duty cycle of 1.25ms (can be seen in the waveform displayed above);
- 5th cycle: Done is on high logic level to indicate that the controller is ready for operation;

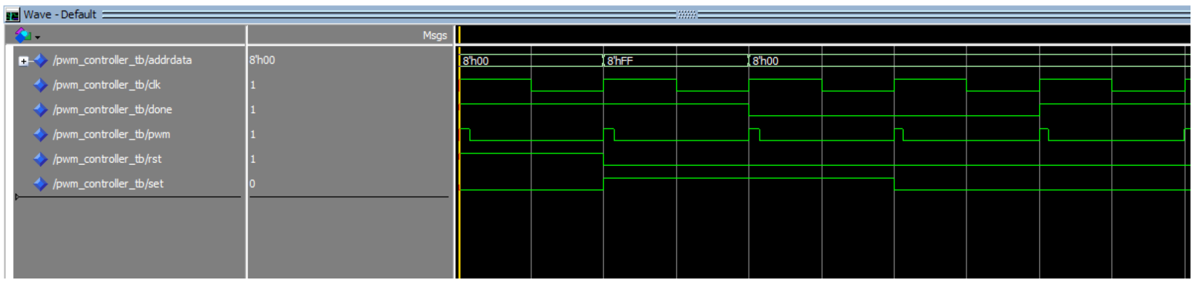


Figure 2.1: Simulation of the top-level component

In this testbench, a test case for a reset after sending the address has been implemented:

```
REPORT "Second check for async reset behavior when sending data";
        set <= '0';
        rst <= '1';
        WAIT UNTIL rising_edge(clk);
        rst <= '0';
        WAIT UNTIL rising_edge(clk);
        set <= '1'; -- SET
        addrdata <= (OTHERS => '1'); -- BROADCAST
        ASSERT done = '1'
        REPORT "Done should remain H if data has not been sent"
        SEVERITY error;
        WAIT UNTIL rising_edge(clk);
        rst <= '1';
        addrdata <= (OTHERS => '0'); -- NOW SEND DATA 00000000
        WAIT UNTIL rising_edge(clk);
        WAIT UNTIL rising_edge(clk);
```

The output waveform shows that after normal functioning with data at 00000000, upon sending a reset signal to the PWM controller, it breaks off the current operation and returns to its zero-position.

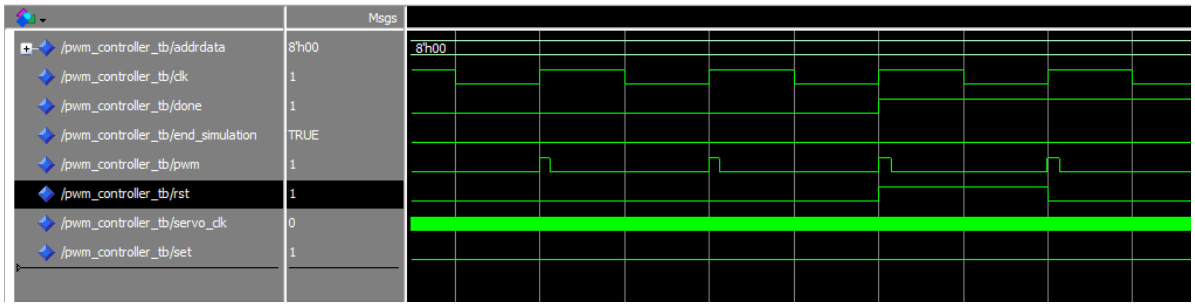

Msgs
/pwm_controller_tb/addrdata 8'h00
/pwm_controller_tb/clk 1
/pwm_controller_tb/done 1
/pwm_controller_tb/end_simulation TRUE
/pwm_controller_tb/pwm 1
/pwm_controller_tb/rst 1
/pwm_controller_tb/servo_clk 0
/pwm_controller_tb/set 1
8'h00

Figure 2.2: Simulation of random asynchronous reset behavior, independent of input values

### Test case 2: address check

To determine whether or not the PWM signal changes when sending a random address, not equal to the controller's address, and a certain position, a separate testbench is created. The waveform output is displayed below.

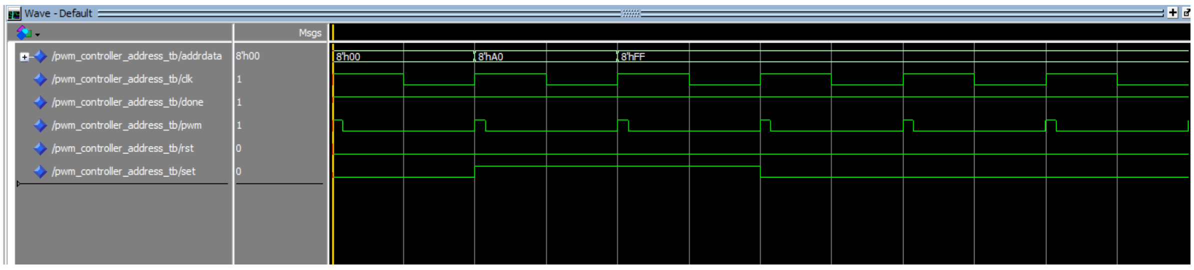
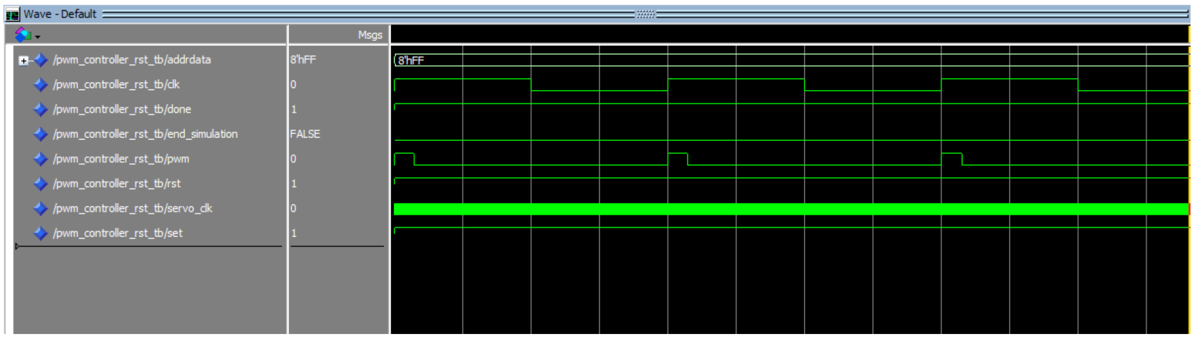

Wave - Default
Msgs
/pwm_controller_address_tb/addrdata 8'h00
/pwm_controller_address_tb/clk 1
/pwm_controller_address_tb/done 1
/pwm_controller_address_tb/pwm 1
/pwm_controller_address_tb/rst 0
/pwm_controller_address_tb/set 0
8'h00
8'hA0
8'hFF

Figure 2.3: Simulation of behavior when address is wrong, the PWM signal does not change.

As can be seen on the waveform, supplied above, the PWM signal does not change after sending a wrong address and a certain position.

### Test case 3: reset behavior

To test the reset functionality, a separate testbench is created that feeds the broadcast address and the maximum position (11111111) to the PWM controller. As expected, the reset does not change the default zero position of the PWM signal (10000000), it remains around 1.5 ms, whatever input is supplied. As the generic address has a default value, it suffices to simply implement a port mapping instead of an additional generic mapping. The simulated waveform is displayed below.



```
/pwm_controller_rst_tb/addrdata        8'hFF
/pwm_controller_rst_tb/clk             0
/pwm_controller_rst_tb/done            1
/pwm_controller_rst_tb/end_simulation  FALSE
/pwm_controller_rst_tb/pwm             0
/pwm_controller_rst_tb/rst             1
/pwm_controller_rst_tb/servo_clk       0
/pwm_controller_rst_tb/set             1
```

Figure 2.4: Simulation of asynchronous reset behavior, independent of input values

## 2.3 PWM Counter

The PWM counter uses the values obtained from calculations in Section 1 to calculate the time of the PWM signal that needs to be at a high digital logic level.

```
    offset_pos <= unsigned("0000" & position) + 640; -- offset
    PROCESS (rst, servo_clk)
    BEGIN
        IF rising_edge(servo_clk) THEN
            IF (counter < 10240) THEN
                counter <= counter + 1;
            ELSE
                counter <= (OTHERS => '0');
            END IF;
        END IF;
    END PROCESS;

    pwm <= '1' WHEN (counter < offset_pos) ELSE
        '0';
```

In the code above, the counter is restarted every 20ms (indicated by maximum count of 10239, starting on 0). After each clock cycle of the system clock, the PWM signal is generated by a simple comparison of the current counter value and the calculated offset position. The architecture of the counter is divided into a sequential process and combinatorial logic to build the PWM signal.

### Architecture Overview

The signals of the behavioral architecture of the PWM Counter entity are:

```
    SIGNAL counter : unsigned(13 DOWNTO 0);
    SIGNAL offset_pos : unsigned(11 DOWNTO 0);
```

The counter holds the current value and the offset position is the position with the predefined and calculated offset of 640. The full architecture is as follows:

```
ARCHITECTURE behavioral OF PWM_Counter IS
    SIGNAL counter : unsigned(13 DOWNTO 0);
    SIGNAL offset_pos : unsigned(11 DOWNTO 0);
BEGIN
    offset_pos <= unsigned("0000" & position) + 640; -- offset
    PROCESS (rst, servo_clk)
    BEGIN
        IF rising_edge(servo_clk) THEN
            IF (counter < 10240) THEN
                counter <= counter + 1;
            ELSE
                counter <= (OTHERS => '0');
            END IF;
        END IF;
    END PROCESS;

    pwm <= '1' WHEN (counter < offset_pos) ELSE '0';

END ARCHITECTURE;
```

### 2.3.1 PWM Counter Testbench

The separated testbench for the PWM counter entity simply checks the timing of the PWM signal for several data points (in steps of 32 bits). This approach allows for an easy and separated approach to testing the PWM signal's conformity.

#### Test case 4: PWM

. The testbench results in the following Waveform from QuestaSim:

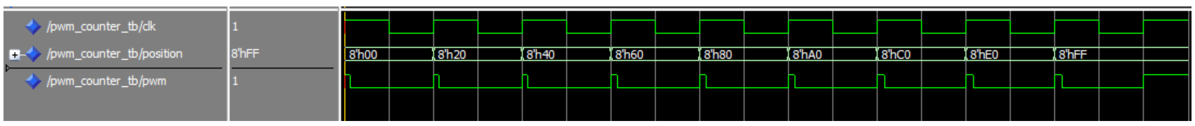


Figure 2.5: Simulation of the PWM counter entity

The code for the process that validates the asserted PWM Ton (time that signal is logic level high) and the actual Ton is as follows:

```
    -- Process to calculate Ton of PWM signal
    test_pwm : PROCESS (pwm)
        VARIABLE time_pwm_rising : TIME;
        VARIABLE time_pwm_falling : TIME;
        VARIABLE time_pwm_diff : TIME;
        VARIABLE current_pwm : INTEGER;
        VARIABLE asserted_time : TIME;
        VARIABLE asserted_time_diff : TIME;
        VARIABLE servo_clock_period : TIME := 1.953125 us;
    BEGIN
```

```
        IF pwm = '1' THEN
            time_pwm_rising := now;
        ELSE
            time_pwm_falling := now;
            time_pwm_diff := time_pwm_falling - time_pwm_rising;
            current_pwm := to_integer(unsigned(position));
            asserted_time := 1.25 ms + (current_pwm * servo_clock_period);
            asserted_time_diff := asserted_time - time_pwm_diff;
            REPORT "Ton of PWM (ms) : " & real'image(real(time_pwm_diff / 1 ns));
            REPORT "Current position " & INTEGER'image(current_pwm);
            REPORT "Time difference between Ton and calculated test time: "
                & TIME'image(asserted_time_diff);
            ASSERT asserted_time_diff < 1 us
            REPORT "Asserted Ton does not equal the output of the PWM Counter"
                SEVERITY error;
        END IF;
    END PROCESS test_pwm;
```

The reported output of this process is provided below.

```
Note: Ton of PWM (ms) : 1.249280e+006
Note: Current position 0
Note: Time difference between Ton and calculated test time: 720 ns
Note: Ton of PWM (ms) : 1.311744e+006
Note: Current position 32
Note: Time difference between Ton and calculated test time: 752 ns
Note: Ton of PWM (ms) : 1.374208e+006
Note: Current position 64
Note: Time difference between Ton and calculated test time: 784 ns
Note: Ton of PWM (ms) : 1.436672e+006
Note: Current position 96
Note: Time difference between Ton and calculated test time: 816 ns
Note: Ton of PWM (ms) : 1.499136e+006
Note: Current position 128
Note: Time difference between Ton and calculated test time: 848 ns
Note: Ton of PWM (ms) : 1.561600e+006
Note: Current position 160
Note: Time difference between Ton and calculated test time: 880 ns
Note: Ton of PWM (ms) : 1.624064e+006
Note: Current position 192
Note: Time difference between Ton and calculated test time: 912 ns
Note: Ton of PWM (ms) : 1.686528e+006
Note: Current position 224
Note: Time difference between Ton and calculated test time: 944 ns
Note: Ton of PWM (ms) : 1.747040e+006
Note: Current position 255
```

```
Note: Time difference between Ton and calculated test time: 975 ns
Note: -- Simulation done --
```

The time difference between the actual Ton (the time range in which the PWM signal is at high logic level) and the asserted time is around 1 us. This might seem as significant when it is compared to the servo clock period of around 2 us, but in comparison to the general system clock with period of 20 ms, the PWM signal is quite accurate.

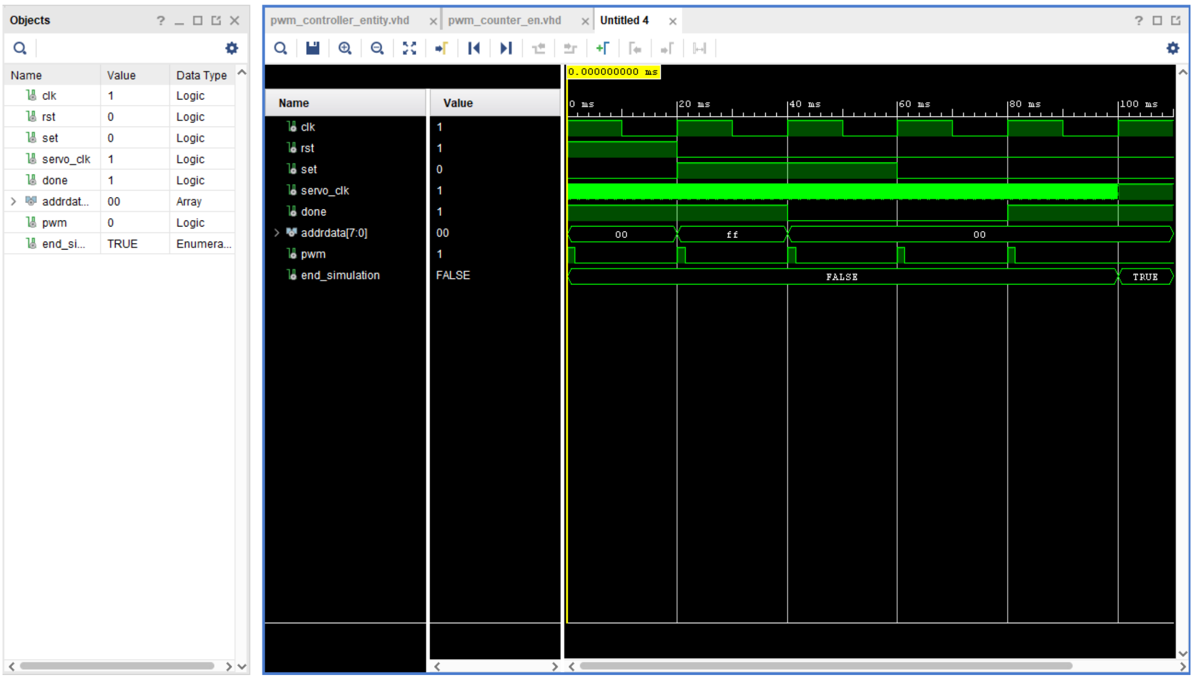
$$\mu_{error} = 850ns = 0.85us = 0.00085ms$$

## 2.4 Vivado Analysis

An additional part of this project is dedicated to featuring the design inside XILINX Vivado Design Suite. The selected board for the Vivado project is the Zynq P1. The different required tasks are briefly discussed in the following sections.

### 2.4.1 Behavioral Simulation

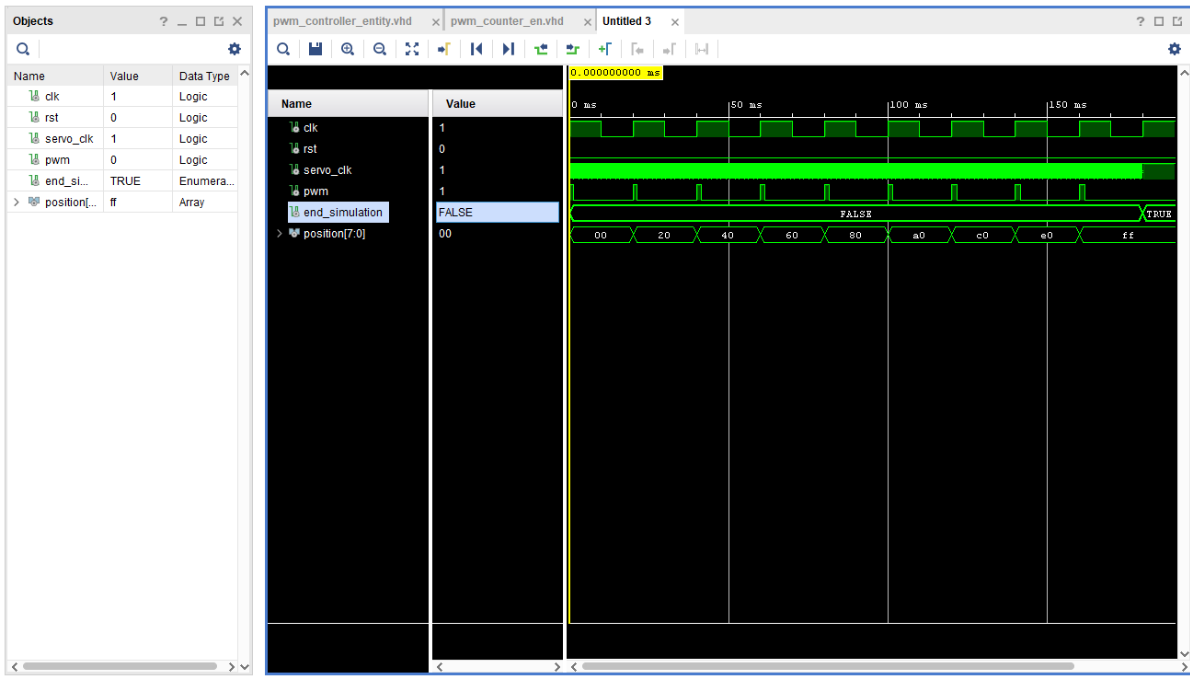
The figures below are the waveform outputs of the different testbenches simulated in XILINX Vivado. As can be seen on the waveforms, these are equal to the output of QuestaSim, which should be no surprise. One thing to note when comparing the different waveform outputs is the timing range, some simulations run for 80 ms, whilst others run for longer or shorter. If not taken into account, these differences can obscure the essence of each simulation and might convey potential errors in the design.



| Name | Value | Data Type |
|---|---|---|
| clk | 1 | Logic |
| rst | 0 | Logic |
| set | 0 | Logic |
| servo_clk | 1 | Logic |
| done | 1 | Logic |
| addrdat... | 00 | Array |
| pwm | 0 | Logic |
| end_si... | TRUE | Enumera... |

| Name | Value |
|---|---|
| clk | 1 |
| rst | 1 |
| set | 0 |
| servo_clk | 1 |
| done | 1 |
| addrdata[7:0] | 00 |
| pwm | 1 |
| end_simulation | FALSE |

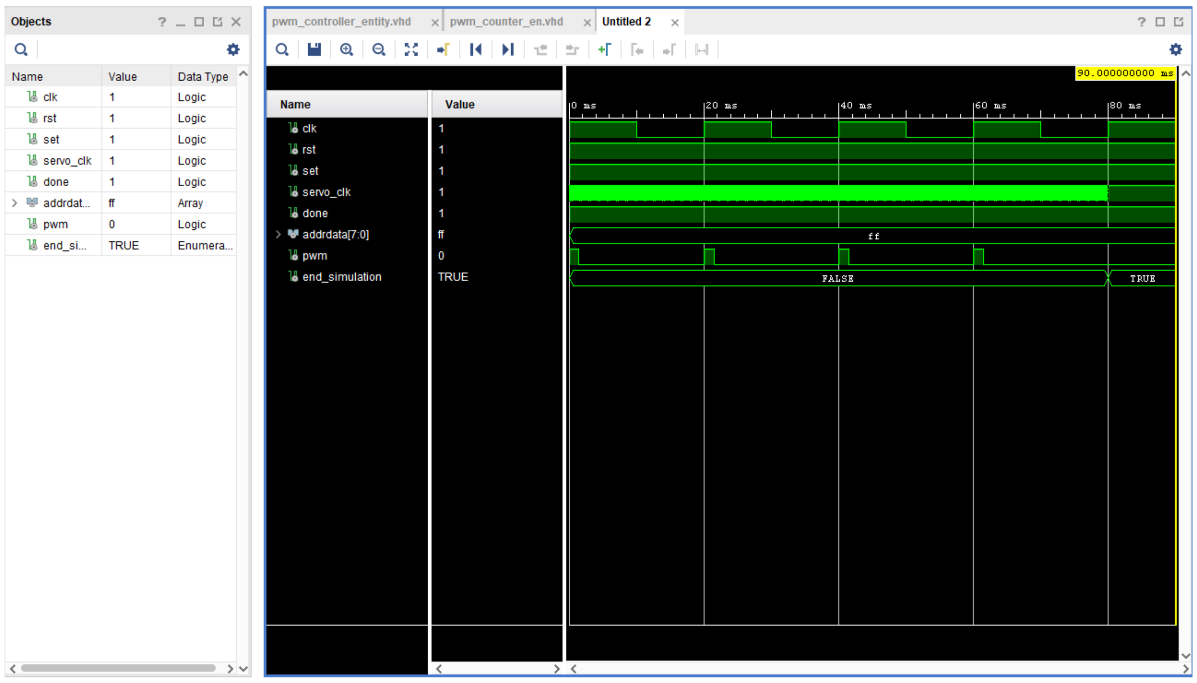
Figure 2.6: Simulation of the default behavior of the controller



| Name | Value | Data Type |
|---|---|---|
| clk | 1 | Logic |
| rst | 0 | Logic |
| servo_clk | 1 | Logic |
| pwm | 0 | Logic |
| end_si... | TRUE | Enumera... |
| position[... | ff | Array |

| Name | Value |
|---|---|
| clk | 1 |
| rst | 0 |
| servo_clk | 1 |
| pwm | 1 |
| end_simulation | FALSE |
| position[7:0] | 00 |

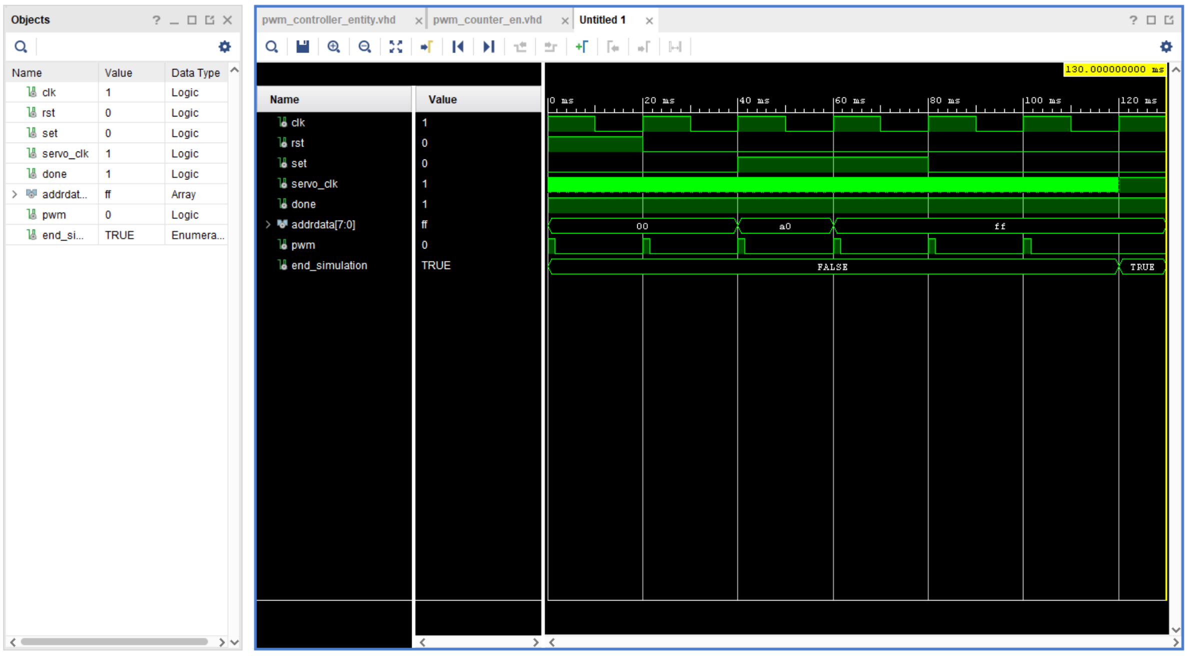
Figure 2.7: Simulation of the counter that constructs the PWM signal



| Name | Value | Data Type |
|---|---|---|
| clk | 1 | Logic |
| rst | 1 | Logic |
| set | 1 | Logic |
| servo_clk | 1 | Logic |
| done | 1 | Logic |
| addrdat... | ff | Array |
| pwm | 0 | Logic |
| end_si... | TRUE | Enumera... |

| Name | Value |
|---|---|
| clk | 1 |
| rst | 1 |
| set | 1 |
| servo_clk | 1 |
| done | 1 |
| addrdata[7:0] | ff |
| pwm | 0 |
| end_simulation | TRUE |

Figure 2.8: Simulation of the reset behavior



Objects

| Name | Value | Data Type |
|---|---|---|
| clk | 1 | Logic |
| rst | 0 | Logic |
| set | 0 | Logic |
| servo_clk | 1 | Logic |
| done | 1 | Logic |
| addrdat... | ff | Array |
| pwm | 0 | Logic |
| end_si... | TRUE | Enumera... |

| Name | Value |
|---|---|
| clk | 1 |
| rst | 0 |
| set | 0 |
| servo_clk | 1 |
| done | 1 |
| addrdata[7:0] | ff |
| pwm | 0 |
| end_simulation | TRUE |

Figure 2.9: Simulation when sending a different address than that of the controller

### 2.4.2 RTL Analysis

After performing the elaborated design task, the schematic obtained is as follows:

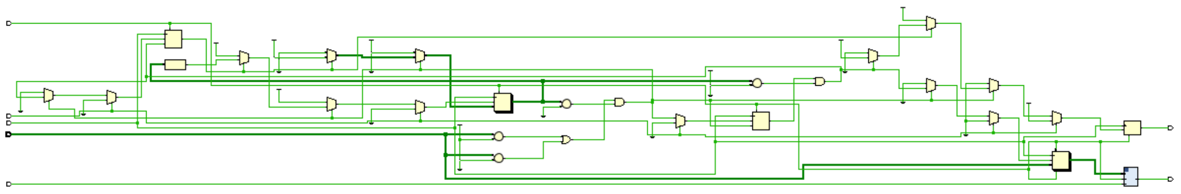


Figure 2.10: RTL Schematic

### 2.4.3 Synthesis

The schematic after synthesis is provided below.

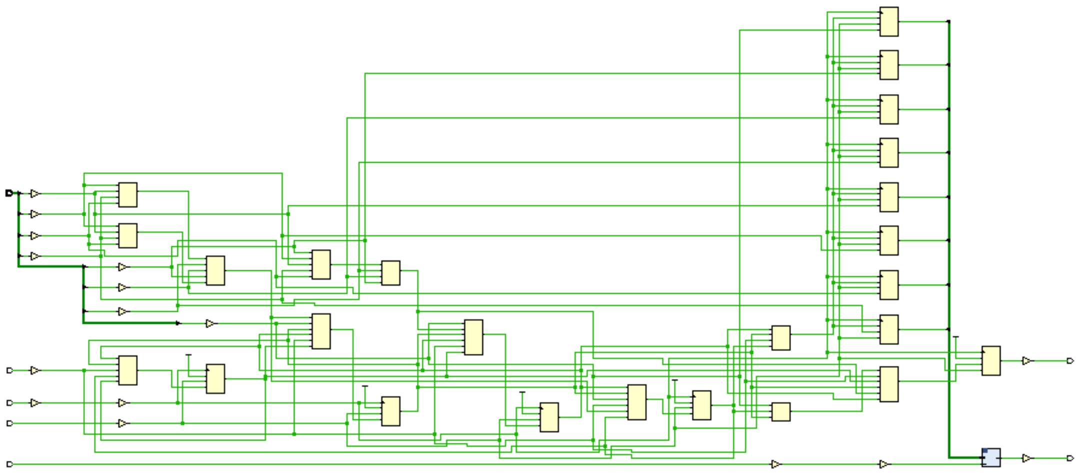
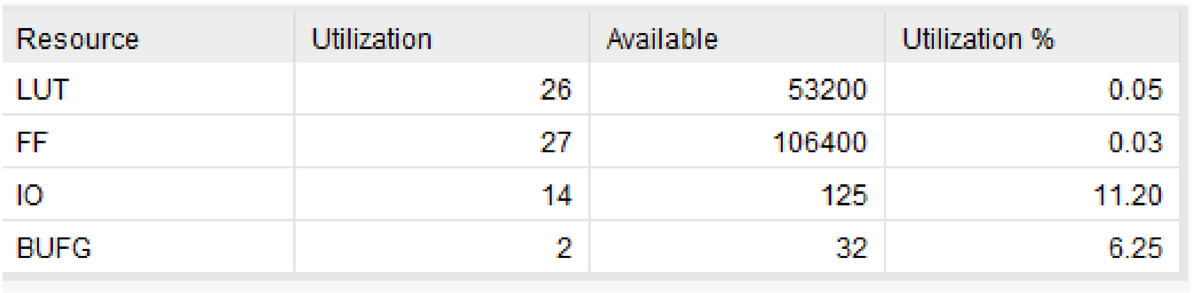


Figure 2.11: Synthesis Schematic

### 2.4.4 Used resources of the FPGA

The table below is the output of the project summary after running the elaborated design, synthesis and implementation tasks. This table shows how many resources are used in the implemented design.



| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 26 | 53200 | 0.05 |
| FF | 27 | 106400 | 0.03 |
| IO | 14 | 125 | 11.20 |
| BUFG | 2 | 32 | 6.25 |

Figure 2.12: Resources used, taken directly from the Project Summary in XILINX Vivado

# Chapter 3

# Review

## 3.1 Functional

During this project several steps are implemented in an iterative manner. Meaning that during the first stage the required components and test cases were abstracted from the requirements. First of all the counter is implemented and also tested with a corresponding testbench. Afterwards, the basic control flow (set-reset) is created using a top-level component, also with a separate testbench. After having tested and verified the correct basic behavior of both components, the more advanced requirements are completed with several different testbenches: address constraint, asynchronous reset behavior, programatic verification of the PWM signal and the random asynchronous reset behavior. Finally the entire source code is analysed using XILINX Vivado. For each testbench, the simulation in QuestaSim is compared to the simulation in Vivado, which results in the same output scenarios in each case. This is an important step to verify the functioning of the implemented design.

## 3.2 Improvements

Some improvements can be made, these can either be functional or structural: some datatypes can be constrained to the exact size required for the design (e.g. the addr _correct flag could be implemented bitwise, as it only requires three possible values, this was omitted for readability). A structural change can be made to the specific architecture of the PWM Controller: instead of working with flags to check the previous state of the controller, a FSM (Finite State Machine) could also have been implemented.

# Chapter 4

# Appendices

## 4.1 clocks-pkg.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

PACKAGE clocks_pkg IS
    PROCEDURE clock(
        SIGNAL clk : INOUT std_logic;
        CONSTANT period : IN TIME;
        SIGNAL end_simulation : IN BOOLEAN
    );
END PACKAGE clocks_pkg;

PACKAGE BODY clocks_pkg IS
    PROCEDURE clock(SIGNAL clk : INOUT std_logic; CONSTANT period : IN TIME;
    SIGNAL end_simulation : IN BOOLEAN) IS
    BEGIN
        LOOP
            EXIT WHEN end_simulation;
            clk <= NOT clk;
            WAIT FOR period/2;
        END LOOP;
    END PROCEDURE clock;
END PACKAGE BODY clocks_pkg;
```

## 4.2 pwm-pk.vhd

```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

PACKAGE pwm_pk IS
    CONSTANT BROADCAST_ADDR : std_logic_vector(7 DOWNTO 0) := "11111111";
    CONSTANT UNICAST_ADDR : std_logic_vector(7 DOWNTO 0) := "00000001";
    -- UNICAST_ADDR is deprecated since using the generic approach
END PACKAGE;
```

## 4.3 pwm-controller-en.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY work;
USE work.pwm_pk.ALL;

ENTITY PWM_Controller IS
    GENERIC (address : std_logic_vector(7 DOWNTO 0) := "00000001");
    PORT (
        rst : IN std_logic;
        clk : IN std_logic;
        servo_clk : IN std_logic;
        set : IN std_logic;
        done : OUT std_logic;
        addrdata : IN std_logic_vector(7 DOWNTO 0) := (OTHERS => '0');
        pwm : OUT std_logic);
END PWM_Controller;

ARCHITECTURE behavioral OF PWM_Controller IS
    COMPONENT PWM_Counter IS
        PORT (
            rst : IN std_logic;
            servo_clk : IN std_logic;
            position : IN std_logic_vector(7 DOWNTO 0);
            pwm : OUT std_logic);
    END COMPONENT;

    SIGNAL addr_is_read : BOOLEAN := FALSE;
    SIGNAL data_is_read : BOOLEAN := FALSE;
    SIGNAL is_addr_servo : BOOLEAN := FALSE;
    SIGNAL data_read : std_logic_vector(7 DOWNTO 0);
    SIGNAL addr_correct : INTEGER := 0;
BEGIN
    pwm_counter_map : PWM_Counter PORT MAP(
        rst => rst, servo_clk => servo_clk, position => data_read, pwm => pwm
    );

    PROCESS (rst, clk) -- start process on change of rst, clk
    BEGIN
        IF (rst = '1') THEN
            addr_is_read <= FALSE;
            data_is_read <= FALSE;
            data_read <= "10000000";
            done <= '1';
            addr_correct <= 0;
        ELSIF rising_edge (clk) THEN
            IF (set = '1') THEN
                -- first clock pulse: read addr
                -- second clock pulse: read data and set done to zero
                IF (addrdata = BROADCAST_ADDR OR addrdata = address)
                AND addr_correct = 0 THEN
                    addr_is_read <= TRUE;
                    addr_correct <= 1;
                    done <= '0';
                ELSE
                    IF addr_correct = 0 THEN
                        addr_correct <= 2;
                    END IF;

                    IF addr_is_read = TRUE AND addr_correct = 1 THEN
                        -- read data
                        data_is_read <= TRUE;
                        data_read <= addrdata;
                        done <= '0';
                    ELSE
                        done <= '1';
                    END IF;
                    IF data_is_read = TRUE THEN
                        done <= '1';
                        addr_correct <= 0;
                    END IF;
                END IF;
            ELSE
                done <= '1';
            END IF;
        END IF;
    END PROCESS;
END ARCHITECTURE;
```

## 4.4 pwm-controller-tb.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.clocks_pkg.ALL;

ENTITY pwm_controller_tb IS
END ENTITY;

ARCHITECTURE test OF pwm_controller_tb IS
    SIGNAL clk : std_logic := '0';
    SIGNAL rst : std_logic;
    SIGNAL set : std_logic;
    SIGNAL servo_clk : std_logic := '0';
    SIGNAL done : std_logic;
    SIGNAL addrdata : std_logic_vector(7 DOWNTO 0) := (OTHERS => '0');
    SIGNAL pwm : std_logic;
    SIGNAL end_simulation : BOOLEAN := FALSE;

    COMPONENT PWM_Counter IS
        PORT (
            rst : IN std_logic;
            servo_clk : IN std_logic;
            position : IN std_logic_vector(7 DOWNTO 0);
            pwm : OUT std_logic);
    END COMPONENT;
    COMPONENT PWM_Controller IS
        PORT (
            rst : IN std_logic;
            clk : IN std_logic;
            servo_clk : IN std_logic;
            set : IN std_logic;
            done : OUT std_logic;
            addrdata : IN std_logic_vector(7 DOWNTO 0);
            pwm : OUT std_logic);
    END COMPONENT;

BEGIN

    UUT : PWM_Controller
    PORT MAP(
        rst => rst,
        clk => clk,
        servo_clk => servo_clk,
        set => set,
        done => done,
        addrdata => addrdata,
        pwm => pwm);

    -- Generate clock signals using clocks_pkg
    clock(servo_clk, 1.953125 us, end_simulation);
    clock(clk, 20 ms, end_simulation);

    stimuli_gen : PROCESS
    BEGIN
        REPORT " -- Simulation start --"
            SEVERITY note;
        -- INITIAL
        WAIT UNTIL rising_edge(clk);
        set <= '0';
        rst <= '1';
        WAIT UNTIL rising_edge(clk);
        rst <= '0';
        set <= '1'; -- SET
        addrdata <= "00000001"; -- BROADCAST
        ASSERT done = '1'
        REPORT "Done should remain H if data has not been sent"
            SEVERITY error;
        WAIT UNTIL rising_edge(clk);
        addrdata <= (OTHERS => '0'); -- NOW SEND DATA 00000000
        WAIT UNTIL rising_edge(clk);
        set <= '0'; -- UNSET
        ASSERT done = '0'
        REPORT "Done should be L when building PWM"
            SEVERITY error;
        WAIT UNTIL rising_edge(clk);

        WAIT UNTIL rising_edge(clk);
        ASSERT done = '1'
        REPORT "Done should be H when PWM is built"
            SEVERITY error;
        REPORT "-- Simulation done --"
            SEVERITY note;
        end_simulation <= true;
        WAIT;
    END PROCESS stimuli_gen;
END ARCHITECTURE test;
```

## 4.5 pwm-controller-rst-tb.vhd

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.clocks_pkg.ALL;

ENTITY pwm_controller_rst_tb IS
END ENTITY;

ARCHITECTURE test OF pwm_controller_rst_tb IS
    SIGNAL clk : std_logic := '0';
    SIGNAL rst : std_logic;
    SIGNAL set : std_logic;
    SIGNAL servo_clk : std_logic := '0';
    SIGNAL done : std_logic;
    SIGNAL addrdata : std_logic_vector(7 DOWNTO 0) := (OTHERS => '0');
    SIGNAL pwm : std_logic;
    SIGNAL end_simulation : BOOLEAN := FALSE;

    COMPONENT PWM_Counter IS
        PORT (
            rst : IN std_logic;
            servo_clk : IN std_logic;
            position : IN std_logic_vector(7 DOWNTO 0);
            pwm : OUT std_logic);
    END COMPONENT;
    COMPONENT PWM_Controller IS
        PORT (
            rst : IN std_logic;
            clk : IN std_logic;
            servo_clk : IN std_logic;
            set : IN std_logic;
            done : OUT std_logic;
            addrdata : IN std_logic_vector(7 DOWNTO 0);
            pwm : OUT std_logic);
    END COMPONENT;

BEGIN

    UUT : PWM_Controller
    PORT MAP(
        rst => rst,
        clk => clk,
        servo_clk => servo_clk,
        set => set,
        done => done,
        addrdata => addrdata,
        pwm => pwm);

    -- Generate clock signals using clocks_pkg
    clock(servo_clk, 1.953125 us, end_simulation);
    clock(clk, 20 ms, end_simulation);

    stimuli_gen : PROCESS
    BEGIN
        REPORT " -- Simulation start --"
            SEVERITY note;
        -- INITIAL
        rst <= '1';
        WAIT UNTIL rising_edge(clk);
        set <= '1'; -- SET
        addrdata <= (OTHERS => '1'); -- BROADCAST
        ASSERT done = '1'
        REPORT "Done should remain H if data has not been sent"
            SEVERITY error;
        WAIT UNTIL rising_edge(clk);
        addrdata <= (OTHERS => '1'); -- Send position max (1.75ms)
        WAIT UNTIL rising_edge(clk);
        WAIT UNTIL rising_edge(clk);
        WAIT UNTIL rising_edge(clk);
        REPORT "-- Simulation done --"
            SEVERITY note;
        end_simulation <= true;
        WAIT;
    END PROCESS stimuli_gen;
END ARCHITECTURE test;
```

## 4.6 pwm-controller-address-tb.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.clocks_pkg.ALL;

ENTITY pwm_controller_address_tb IS
END ENTITY;

ARCHITECTURE test OF pwm_controller_address_tb IS
    SIGNAL clk : std_logic := '0';
    SIGNAL rst : std_logic;
    SIGNAL set : std_logic;
    SIGNAL servo_clk : std_logic := '0';
    SIGNAL done : std_logic;
    SIGNAL addrdata : std_logic_vector(7 DOWNTO 0) := (OTHERS => '0');
    SIGNAL pwm : std_logic;
    SIGNAL end_simulation : BOOLEAN := FALSE;

    COMPONENT PWM_Counter IS
        PORT (
            rst : IN std_logic;
            servo_clk : IN std_logic;
            position : IN std_logic_vector(7 DOWNTO 0);
            pwm : OUT std_logic);
    END COMPONENT;
    COMPONENT PWM_Controller IS
        PORT (
            rst : IN std_logic;
            clk : IN std_logic;
            servo_clk : IN std_logic;
            set : IN std_logic;
            done : OUT std_logic;
            addrdata : IN std_logic_vector(7 DOWNTO 0);
            pwm : OUT std_logic);
    END COMPONENT;

BEGIN

    UUT : PWM_Controller
    PORT MAP(
        rst => rst,
        clk => clk,
        servo_clk => servo_clk,
        set => set,
        done => done,
        addrdata => addrdata,
        pwm => pwm);

    -- Generate clock signals using clocks_pkg
    clock(servo_clk, 1.953125 us, end_simulation);
    clock(clk, 20 ms, end_simulation);

    stimuli_gen : PROCESS
    BEGIN
        REPORT " -- Simulation start --"
            SEVERITY note;
        WAIT UNTIL rising_edge(clk);
        set <= '0';
        rst <= '1';
        WAIT UNTIL rising_edge(clk);
        rst <= '0';
        WAIT UNTIL rising_edge(clk);
        set <= '1';
        addrdata <= "10100000"; -- send random address
        WAIT UNTIL rising_edge(clk);
        addrdata <= (OTHERS => '1'); -- NOW SEND DATA 111111111
        WAIT UNTIL rising_edge(clk);
        set <= '0'; -- UNSET
        WAIT UNTIL rising_edge(clk);
        WAIT UNTIL rising_edge(clk);
        REPORT "-- Simulation done --"
            SEVERITY note;
        end_simulation <= true;
        WAIT;
    END PROCESS stimuli_gen;
END ARCHITECTURE test;
```

## 4.7 pwm-counter-en.vhd

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY PWM_Counter IS
    PORT (
        rst : IN std_logic;
        servo_clk : IN std_logic;
        position : IN std_logic_vector(7 DOWNTO 0);
        pwm : OUT std_logic);
END PWM_Counter;

ARCHITECTURE behavioral OF PWM_Counter IS
    SIGNAL counter : unsigned(13 DOWNTO 0);
    SIGNAL offset_pos : unsigned(11 DOWNTO 0);
BEGIN
    offset_pos <= unsigned("0000" & position) + 640; -- offset
    PROCESS (rst, servo_clk)
    BEGIN
        IF rising_edge(servo_clk) THEN
            IF (counter < 10240) THEN
                counter <= counter + 1;
            ELSE
                counter <= (OTHERS => '0');
            END IF;
        END IF;
    END PROCESS;

    pwm <= '1' WHEN (counter < offset_pos) ELSE
        '0';

END ARCHITECTURE;
```

## 4.8 pwm-counter-tb.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE work.clocks_pkg.ALL;

ENTITY pwm_counter_tb IS
END ENTITY;

ARCHITECTURE test OF pwm_counter_tb IS
    SIGNAL clk : std_logic := '0';
    SIGNAL rst : std_logic := '0';
    SIGNAL servo_clk : std_logic := '0';
    SIGNAL pwm : std_logic;
    SIGNAL end_simulation : BOOLEAN := FALSE;
    SIGNAL position : std_logic_vector(7 DOWNTO 0) := "00000000";

    COMPONENT PWM_Counter IS
        PORT (
            rst : IN std_logic;
            servo_clk : IN std_logic;
            position : IN std_logic_vector(7 DOWNTO 0);
            pwm : OUT std_logic);
    END COMPONENT;

BEGIN
    U1 : PWM_Counter
    PORT MAP(
        rst => rst,
        servo_clk => servo_clk,
        position => position,
        pwm => pwm);

    -- Generate clock signals using clocks_pkg
    clock(servo_clk, 1.953125 us, end_simulation);
    clock(clk, 20 ms, end_simulation);

    stimuli_gen : PROCESS
        VARIABLE count : INTEGER := 0;
    BEGIN
        REPORT " -- Simulation start --"
            SEVERITY note;
        rst <= '1';
        WAIT UNTIL rising_edge(clk);
        rst <= '0';
        WHILE count < 8 LOOP

            WAIT UNTIL rising_edge(clk);
            position <= std_logic_vector(unsigned(position) + 32);
            count := count + 1;
        END LOOP;
        position <= "11111111";
        WAIT UNTIL rising_edge(clk);
        REPORT "-- Simulation done --"
            SEVERITY note;
        end_simulation <= TRUE;
        WAIT;
    END PROCESS stimuli_gen;

    -- Process to calculate Ton of PWM signal
    test_pwm : PROCESS (pwm)
        VARIABLE time_pwm_rising : TIME;
        VARIABLE time_pwm_falling : TIME;
        VARIABLE time_pwm_diff : TIME;
        VARIABLE current_pwm : INTEGER;
        VARIABLE asserted_time : TIME;
        VARIABLE asserted_time_diff : TIME;
        VARIABLE servo_clock_period : TIME := 1.953125 us;
    BEGIN
        IF pwm = '1' THEN
            time_pwm_rising := now;
        ELSE
            time_pwm_falling := now;
            time_pwm_diff := time_pwm_falling - time_pwm_rising;
            current_pwm := to_integer(unsigned(position));
            asserted_time := 1.25 ms + (current_pwm * servo_clock_period);
            asserted_time_diff := asserted_time - time_pwm_diff;
            REPORT "Ton of PWM (ms) : " & real'image(real(time_pwm_diff / 1 ns));
            REPORT "Current position " & INTEGER'image(current_pwm);
            REPORT "Time difference between Ton and calculated test time: "
                & TIME'image(asserted_time_diff);
            ASSERT asserted_time_diff < 1 us
            REPORT "Asserted Ton does not equal the output of the PWM Counter"
                SEVERITY error;
        END IF;
    END PROCESS test_pwm;
END test;
```

# Bibliography

[1] AllAboutCircuits. “Pulse Width Modulation”. In: (). URL: `https://www.allaboutcircuits.com/textbook/semiconductors/chpt-11/pulse-width-modulation/`.

[2] Michael Heyvaert Peter Veelaert Dimitri Van Cauwelaert. “VHDL Design, Testbench: servo controller”. In: (2020).

[3] Steven Redant. *Hardware beschrijven en simuleren in VHDL.*