AES portfolio

By Lilith (freek) bertens

4527747

Contents

[Assignment 1 3](#_Toc184596715)

[1. Problem description 3](#_Toc184596716)

[2. Architecture 3](#_Toc184596717)

[3. Implementation 3](#_Toc184596718)

[Toplevel 3](#_Toc184596719)

[PWM 5](#_Toc184596720)

[MUX 6](#_Toc184596721)

[Explanation 6](#_Toc184596722)

[4. Testbench 7](#_Toc184596723)

[5. Result of verification 9](#_Toc184596724)

[6. RTL 10](#_Toc184596725)

[7. Constraints file 10](#_Toc184596726)

[8. Questions 11](#_Toc184596727)

[9. Conclusion 11](#_Toc184596728)

[Assignment 2 12](#_Toc184596729)

[1. Problem description 12](#_Toc184596730)

[2. Architecture 12](#_Toc184596731)

[3. Implementation 12](#_Toc184596732)

[4. Testbench 13](#_Toc184596733)

[5. Result of verification 16](#_Toc184596734)

[6. RTL 16](#_Toc184596735)

[7. Constraints file 16](#_Toc184596736)

[9. Conclusion 17](#_Toc184596737)

# Assignment 1

PWM with MUX

## 1. Problem description

Make a PWM module which can have its period changed by editing generic values. Make a MUX which will select between different levels of power which are to be sent to the PWM. Make a testbench for both the PWM and MUX added together to see if they work. Confirm the testbench. Add constraints to the general module and check if it works by uploading it to the zybo z10. Add a video showing the board does as it should. All modules should be parametrized so that the max value and number of bits for power can be changed with ease. When reset is on pwm direction should be disconnected. The mux % which should be given from select 0 to select 7 is: 100, 50, 20, 10, -10, -20, -50, 101.

## 2. Architecture

The general architecture used in the code is a toplevel which uses a pwm and mux as its part files. The power output from the mux is then connected to the pwm.  
Pwm outputs to all the outputs in parallel but it counts up the counter in sequence. This is because the counter needs to count up when there is a rising edge which is harder to implement in parallel.

The mux is very simple in design, it is a great number of when else’s stacked one after the other to make a select. It is a very simple component.

## 3. Implementation

### Toplevel

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.numeric\_std.all;

use IEEE.math\_real.all;

entity TopLevel is

Port (

RSTT, CLKT : in std\_logic;

SELT : in std\_logic\_vector(2 downto 0);

PWM\_OUTT, PWM\_DIRECTIONT, PWM\_ERRORT : out std\_logic

);

end TopLevel;

architecture Behavioral of TopLevel is

constant max\_value : integer := 25000;

constant bitnumber : integer := integer(ceil(log2(real(max\_value))));

signal POWERT : std\_logic\_vector(bitnumber downto 0);

signal PWMB : std\_logic;

signal PWMDIRB : std\_logic;

signal PWMERRB : std\_logic;

component PWM

generic (

max\_value : integer;

bitnumber : integer

);

Port (

RST, CLK : in std\_logic := '0';

POWER : in std\_logic\_vector(bitnumber downto 0) := (others => '0'); --the counter can be from 12500 to 25000 (log2(25000) ~= 15) and is signed so one extra bit

PWM\_OUT, PWM\_DIRECTION, PWM\_ERROR : out std\_logic

);

end component;

component MUX

generic (

max\_value : integer;

bitnumber : integer

);

Port (

SEL : in std\_logic\_vector(2 downto 0);

POWER : out std\_logic\_vector(bitnumber downto 0)

);

end component;

begin

PWM1 : PWM

generic map (

max\_value => max\_value,

bitnumber => bitnumber

)

port map (

POWER => POWERT,

RST => RSTT,

CLK => CLKT,

PWM\_OUT => PWM\_OUTT,

PWM\_DIRECTION => PWM\_DIRECTIONT,

PWM\_ERROR => PWM\_ERRORT

);

MUX1 : MUX

generic map (

max\_value => max\_value,

bitnumber => bitnumber

)

port map (

SEL => SELT,

POWER => POWERT

);

end Behavioral;

### PWM

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.numeric\_std.all;

entity PWM is

generic (

max\_value : integer; --according to specification 100us and 200us would be from 12500 to 25000 period

bitnumber : integer --the amount of bits -1 required to describe the max value in power

);

Port (

RST, CLK : in std\_logic;

POWER : in std\_logic\_vector(bitnumber downto 0); --the counter can be from 12500 to 25000 (log2(25000) ~= 15) and is signed so one extra bit

PWM\_OUT, PWM\_DIRECTION, PWM\_ERROR : out std\_logic

);

end PWM;

architecture Behavioral of PWM is

signal Cntr : std\_logic\_vector(bitnumber-1 downto 0);

begin

RisingEdge : process(CLK, RST)

begin

if rising\_edge(CLK) then

if to\_integer(unsigned(Cntr)) >= max\_value or RST = '1' then

Cntr <= (others => '0');

else

Cntr <= std\_logic\_vector(to\_unsigned(to\_integer(unsigned(Cntr)) + 1, bitnumber));

end if;

end if;

end process;

PWM\_OUT <= '1' when (RST = '0' and to\_integer(unsigned(Cntr)) < to\_integer(abs(signed(POWER)))) else '0'; --on when RST is off and counter is smaller then power

PWM\_DIRECTION <= POWER(bitnumber) when RST = '0' else 'Z'; --direction is equal to the first bit of power, when 1 its negative

PWM\_ERROR <= '1' when (to\_integer(abs(signed(POWER))) > max\_value) and RST = '0' else '0'; --there is an error when the power is outside of the max range

end Behavioral;

### MUX

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.numeric\_std.all;

entity MUX is

generic (

max\_value : integer;

bitnumber : integer

);

Port (

SEL : in std\_logic\_vector(2 downto 0);

POWER : out std\_logic\_vector(bitnumber downto 0)

);

end MUX;

architecture Behavioral of MUX is

begin

POWER <= std\_logic\_vector(to\_signed(max\_value, bitnumber+1)) when SEL = "000" else

std\_logic\_vector(to\_signed(max\_value, bitnumber+1)) when SEL = "000" else

std\_logic\_vector(to\_signed(max\_value/2, bitnumber+1)) when SEL = "001" else

std\_logic\_vector(to\_signed(max\_value/5, bitnumber+1)) when SEL = "010" else

std\_logic\_vector(to\_signed(max\_value/10, bitnumber+1)) when SEL = "011" else

std\_logic\_vector(to\_signed(-max\_value/10, bitnumber+1)) when SEL = "100" else

std\_logic\_vector(to\_signed(-max\_value/5, bitnumber+1)) when SEL = "101" else

std\_logic\_vector(to\_signed(-max\_value/2, bitnumber+1)) when SEL = "110" else

std\_logic\_vector(to\_signed(max\_value+1, bitnumber+1)) when SEL = "111" else (others => '0');

end Behavioral;

### Explanation

The implementation is quite minimal. The use of signals, constants and generics are used where they were asked to be used. The constant used in toplevel is to make it easier and not require the engineer to recalculate the number of bits needed every time.

## 4. Testbench

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

use IEEE.math\_real.all;

entity tb\_top\_level\_PWM is

end tb\_top\_level\_PWM;

architecture Behavioral of tb\_top\_level\_PWM is

constant max\_value : integer := 25000;

constant bitnumber : integer := integer(ceil(log2(real(max\_value))));

signal tb\_CLK : STD\_LOGIC := '0';

signal tb\_RST : STD\_LOGIC := '1';

signal tb\_PWM\_OUT : STD\_LOGIC;

signal tb\_PWM\_DIR : STD\_LOGIC;

signal tb\_PWM\_ERR : STD\_LOGIC;

signal tb\_SEL : STD\_LOGIC\_VECTOR (2 downto 0);

component TopLevel

Port (

RSTT, CLKT : in std\_logic;

SELT : in std\_logic\_vector(2 downto 0);

PWM\_OUTT, PWM\_DIRECTIONT, PWM\_ERRORT : out std\_logic

);

end component;

begin

tb\_CLK <= not tb\_CLK after 4ns;

TEST : process begin

tb\_RST <= '1';

wait for 20ns;

tb\_RST <= '0';

wait for 20 ns;

tb\_SEL <= "000";

wait for 1ms; --(20ns \* 20000 cycles)

tb\_SEL <= "001";

wait for 1ms; --(20ns \* 20000 cycles)

tb\_SEL <= "010";

wait for 1ms; --(20ns \* 20000 cycles)

tb\_SEL <= "011";

wait for 1ms; --(20ns \* 20000 cycles)

tb\_SEL <= "100";

wait for 1ms; --(20ns \* 20000 cycles)

tb\_SEL <= "101";

wait for 1ms; --(20ns \* 20000 cycles)

tb\_SEL <= "110";

wait for 1ms; --(20ns \* 20000 cycles)

tb\_SEL <= "111";

wait for 1ms; --(20ns \* 20000 cycles)

tb\_RST <= '1';

wait;

end process;

UUT: TopLevel

Port map (

SELT => tb\_SEL,

CLKT => tb\_CLK,

RSTT => tb\_RST,

PWM\_OUTT => tb\_PWM\_OUT,

PWM\_DIRECTIONT => tb\_PWM\_DIR,

PWM\_ERRORT => tb\_PWM\_ERR

);

end Behavioral;

The testbench includes every possible select and the clock signal. Meaning that all of the possible states are tested. At the end it sets RST to high to test if RST works as it should as well.

## A screenshot of a computer Description automatically generated5. Result of verification

Figure 1: simulation of entire pwm with mux

It passes the verification as we can see that the % aligns with what was asked for the mux. Wel, not exactly. There is a 1 clock cycle gap in the 100% area. This 1 clock cycle is not a mayor problem as we either have issues with 100% or issues with 0% depending on how the code is made. The direction also behaves as it should, so does the pwm error and the reset. Meaning that as far as it can be seen it works as it should. What has not been tested are edge cases and disconnects though. Meaning that it has not been fully confirmed and can act strange in cases which should not happen in the first place.

For the IRL verification see the following video: <https://youtube.com/shorts/7kFivpsEwrM?feature=share>.

## 6. RTL

A close-up of a computer screen

Description automatically generated

Mux:  


PWM:

A computer screen shot of a circuit diagram

Description automatically generated

A screenshot of a computer

Description automatically generated

As can be seen the resources used is quite minimal. The register seen is for the counter which is required.

## 7. Constraints file

##Clock signal

set\_property -dict { PACKAGE\_PIN K17 IOSTANDARD LVCMOS33 } [get\_ports { CLKT }];

##Switches

set\_property -dict { PACKAGE\_PIN G15 IOSTANDARD LVCMOS33 } [get\_ports { SELT[0] }];

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { SELT[1] }];

set\_property -dict { PACKAGE\_PIN W13 IOSTANDARD LVCMOS33 } [get\_ports { SELT[2] }];

##Buttons

set\_property -dict { PACKAGE\_PIN K18 IOSTANDARD LVCMOS33 } [get\_ports { RSTT }];

##LEDs

set\_property -dict { PACKAGE\_PIN M14 IOSTANDARD LVCMOS33 } [get\_ports { PWM\_OUTT }];

set\_property -dict { PACKAGE\_PIN M15 IOSTANDARD LVCMOS33 } [get\_ports { PWM\_ERRORT }];

set\_property -dict { PACKAGE\_PIN G14 IOSTANDARD LVCMOS33 } [get\_ports { PWM\_DIRECTIONT }];

The ports linked were the ones asked for. The select can be set with the switches. Reset is a button and the pwm outputs are sent to the led’s. Of course, the clock must be included so it is linked to the clock in the design.

## 8. Questions

The frequency of the PWM will effect two things. We will assume for the following set of answer that the clock speed is not able to be changed.

Increasing the maximum value and in turn the pwm cycle will mean that you get more precision over the exact voltage you send but the fact that it is a square wave will become more apparent. Meaning that if the cycle is very large it will just be turning off and on instead of continuously turn.

## 9. Conclusion

This was generally very easy, it is not as fool proof as it could possibly be but it works what we need it for. We have made one before and I remember exactly how we were supposed to do it so the only challenge was remembering the syntax.

# Assignment 2

Encoder

## 1. Problem description

The assignment is to make an incremental encoder in vhdl, make it generic so that the bit number of position can be changed and to implement it on the zybo board.

## 2. Architecture

The architecture in use is rather simple, it is one component with one process in it.

## 3. Implementation

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity Encoder is

generic (

PosiBits: integer := 2

);

Port (

CLK, RST, A, B: in std\_logic;

POSITION: out std\_logic\_vector(PosiBits downto 0);--this should be handled as a signed number

ENC\_ERROR: out std\_logic

);

end Encoder;

architecture Behaviour of Encoder is

signal Prevstate: unsigned(1 downto 0);

signal Poss: signed(PosiBits downto 0);

signal AA, BB, AAA, BBB: std\_logic := '0';

begin

process(CLK, RST)

variable state: unsigned(1 downto 0);

variable diff: signed(2 downto 0);

begin

if RST = '1' then

Poss <= (others=>'0');

ENC\_ERROR <= '0';

Prevstate <= BBB & ((not AAA) xor BBB);

elsif rising\_edge(CLK) then

state := BBB & ((not AAA) xor BBB);

diff := signed(('0' & Prevstate) - ('0' & state));

if abs(diff) = 2 then

ENC\_ERROR <= '1';

else

if abs(diff) = 3 then

diff := -(diff rem 2);

end if;

Poss <= Poss + diff;

ENC\_ERROR <= '0';

end if;

AA <= A;

BB <= B;

AAA <= AA;

BBB <= BB;

Prevstate <= state;

end if;

end process;

POSITION <= std\_logic\_vector(Poss);

end architecture Behaviour;

The code which can be seen above loops through all of the states and checks which one the A and B signals are currently in. The state it finds is then used to check in comparison to the last state if this state is ok. If it is ok then the position is either lowered or increased (depending on if it is going CW or CCW). The if else does need to be within the rising edge loop otherwise it will add up or count down every clock signal, which is why RST = ‘1’ is also added as a condition otherwise RST needs to be 1 on a clock rising edge which could go wrong. The AAA and BBB are used to ensure metastability cooperates.

## 4. Testbench

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity Encoder\_tb is

end Encoder\_tb;

architecture Behavioral of Encoder\_tb is

constant PosiBits: integer := 10;

component Encoder

Port (

CLK, RST, A, B: in std\_logic := '0';

POSITION: out std\_logic\_vector(PosiBits downto 0) := (others => '0');--this should be handled as a signed number

ENC\_ERROR: out std\_logic

);

end component;

signal tb\_CLK: std\_logic := '0';

signal tb\_RST: std\_logic := '0';

signal tb\_A: std\_logic := '0';

signal tb\_B: std\_logic := '0';

signal tb\_POSITION: std\_logic\_vector(PosiBits downto 0);

signal tb\_ENC\_ERROR: std\_logic;

begin

UUT: Encoder

port map (

CLK => tb\_CLK,

RST => tb\_RST,

A => tb\_A,

B => tb\_B,

POSITION => tb\_POSITION,

ENC\_ERROR => tb\_ENC\_ERROR

);

tb\_CLK <= not tb\_CLK after 4ns;

Ctete : PROCESS BEGIN

tb\_RST <= '1';

wait for 50 ns;

tb\_RST <= '0';

wait for 2500 ns;

tb\_RST <= '1';

wait for 50 ns;

tb\_RST <= '0';

WAIT;

END PROCESS;

Atest : PROCESS

BEGIN

while now < 1 us loop

tb\_A <= not tb\_A;-- after 20ns;

wait for 40 ns;

end loop;

wait for 20 ns;

while now < 2 us loop

tb\_A <= not tb\_A;

wait for 40 ns;

end loop;

wait for 20 ns;

while now < 3 us loop

tb\_A <= not tb\_A;

wait for 40 ns;

end loop;

wait for 20 ns;

while now < 4 us loop

tb\_A <= not tb\_A;

wait for 40 ns;

end loop;

WAIT;

END PROCESS Atest;

Btest : PROCESS

BEGIN

while now < 1 us loop

tb\_B <= not tb\_B;

wait for 40 ns;

end loop;

while now < 2 us loop

tb\_B <= not tb\_B;

wait for 40 ns;

end loop;

while now < 3 us loop

tb\_B <= not tb\_B;

wait for 40 ns;

end loop;

while now < 4 us loop

tb\_B <= not tb\_B;

wait for 40 ns;

end loop;

WAIT;

END PROCESS Btest;

end Behavioral;

the code above and Figure 2: test result encoder tb it shows that it sends the different combinations of A and B and sends back how it should react to it. It also resets in the middle and lets it negative to show that it can also count backwards.

## 5. Result of verification

A screenshot of a computer

Description automatically generated

Figure 2: test result encoder tb

As we can see it does count up and down like it should. It also gives an error when it should. It does show about the right value, saying exactly what value it should be on is tricky but it should count up 4 times per correct wave. Meaning that if there are 12 pulses of A it should be at about 48. But it is a 51, this can be due to the start and end of the 1 us areas having some extra parts which can be counted as another pulse.

The following video <https://youtu.be/ztNPmg3gCPo> shows a demonstration of the code but with constraints on an actual device. As can be seen it counts up just like it should.

## 6. RTL

A diagram of a circuit

Description automatically generated

A screenshot of a computer

Description automatically generated

## 7. Constraints file

##Clock signal

set\_property -dict { PACKAGE\_PIN K17 IOSTANDARD LVCMOS33 } [get\_ports { CLKT }];

##Switches

set\_property -dict { PACKAGE\_PIN G15 IOSTANDARD LVCMOS33 } [get\_ports { SELT[0] }];

set\_property -dict { PACKAGE\_PIN P15 IOSTANDARD LVCMOS33 } [get\_ports { SELT[1] }];

set\_property -dict { PACKAGE\_PIN W13 IOSTANDARD LVCMOS33 } [get\_ports { SELT[2] }];

##Buttons

set\_property -dict { PACKAGE\_PIN K18 IOSTANDARD LVCMOS33 } [get\_ports { RSTT }];

##LEDs

set\_property -dict { PACKAGE\_PIN M14 IOSTANDARD LVCMOS33 } [get\_ports { PWM\_OUTT }];

set\_property -dict { PACKAGE\_PIN M15 IOSTANDARD LVCMOS33 } [get\_ports { PWM\_ERRORT }];

set\_property -dict { PACKAGE\_PIN G14 IOSTANDARD LVCMOS33 } [get\_ports { PWM\_DIRECTIONT }];

The pins used are exactly as stated in the assignment.

## 9. Conclusion

What didn’t work was vivado. It decided that today was the day that it wanted me to get very angry, because even though all code was 100% correct it still decided that all simulation outputs should be U. This was not true. The solution to getting vivado to cooperate before any electronics broke was to create a new project and add all of the files back to it again. There was no rhyme or reason to this error.

Another problem encountered is the fact that the implementation and simulation did not line up at all. This was fixed by some smaller reorganization and by adding metastability protection.