TP 0x02	Codesign		Bilel
	PWM Generation driven with sawtooth signal		

EX01 :

- Creating vhdl LED controller :

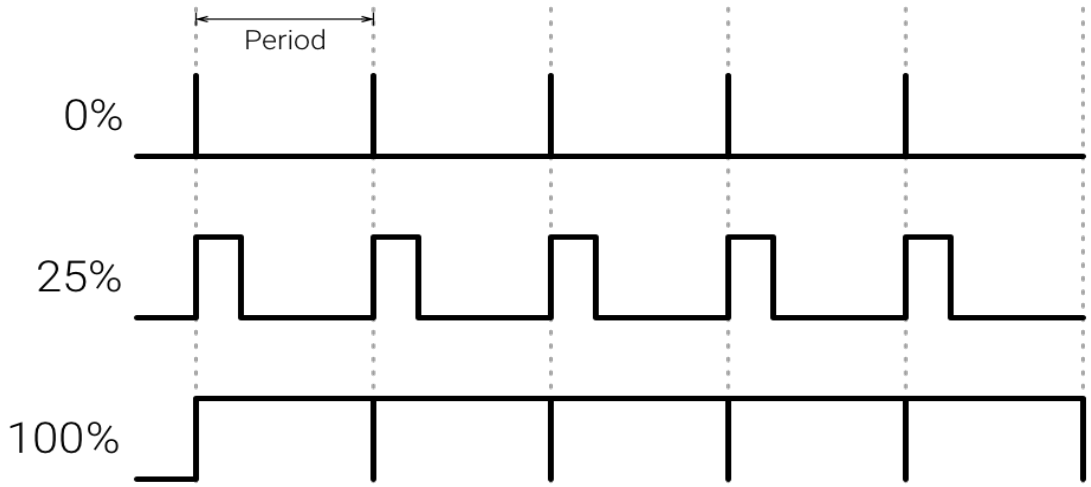
Windows :
Execute vivado using desktop shortcut
Linux :
Open terminal
Source the vivavdo setting64.sh file using the command
source /path to vivado/settings64.sh
Type vivado on terminal window

EX01:

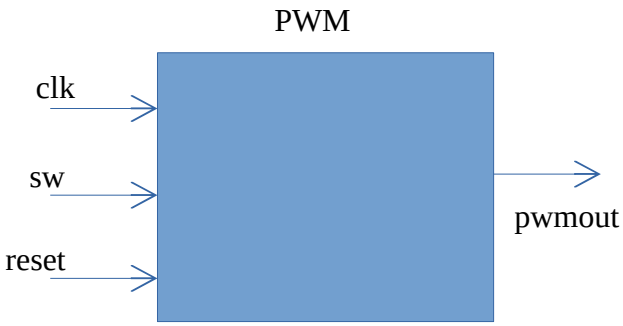
The hardware module that will developed in this exercice supposed to generate a pwm signal that will be routed to the on board LEDS to control their intensity.

About PWM:

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (3.3 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 3.3v controlling the brightness of the LED.



Module ports



clk	Module global clock	input
sw	Duty cycle value	input
reset	Reset signal(active low)	input
pwmout	PWM signal output	output

The clock signal frequency is 100 MHz.
The clock cycle should go between 0 and 100.
The reset signal is active low.

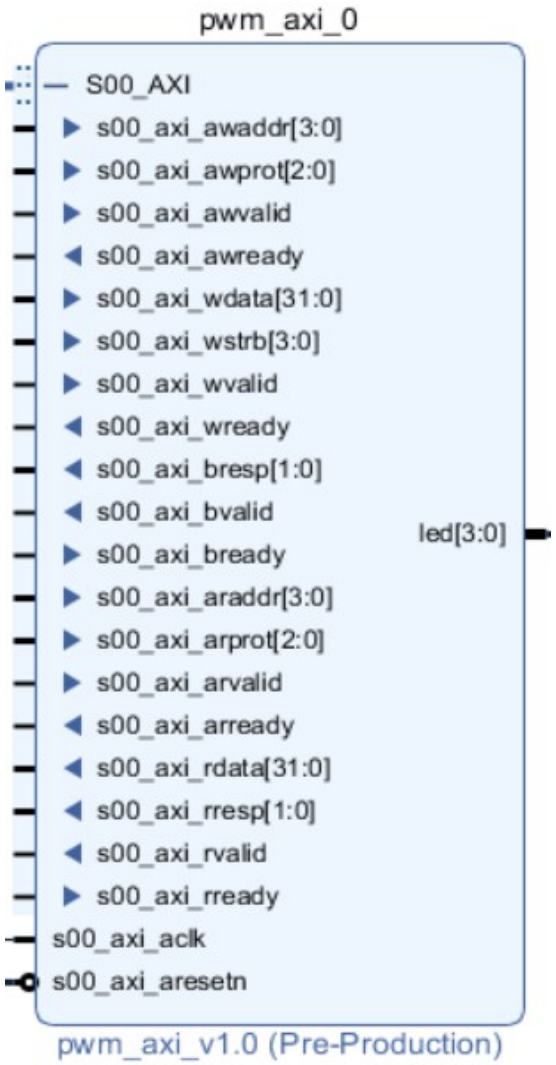
- write the hdl code for the module?
- write a test bench for the module?
- Check the functionality correctness of the module using the generated waves.

EX02:

Creating AXI4 lite IP:

In this exercise we will add an axi lite interface to The previously developed pwm module in the PL of the Zynq device. It will also be connected to the Zynq processor via an AXI interconnect, allowing duty cycle to be changed via a software application.

s00_axi_aclk	Clock signal	input
S00_AXI	AXI Lite interface	port
soo_axi_aresetn	Reset signal	input
led	Leds output port	output



- Using the IP creator tool create an axi interface to the pwm module.(see the ip creator video tutorial)

    on the top design file:
    - look for -- Users to add ports here and add the led output port
    led : out std_logic_vector(3 downto 0);
    - look for -- component declaration and add also the led port to the component
    LED : out std_logic_vector(3 downto 0);
    - add the led port to the port map
    LED => led ,

on the lower level component :

- look for -- Users to add ports here and add the led output port

LED : out std_logic_vector(3 downto 0);

- add the pwm component just before the architecture begin.

```
component pwm is
            port (
                clk: in std_logic;
                SW : in std_logic_vector (7 downto 0);
                reset : in std_logic;
                outpwm : out std_logic
                );

        end component pwm;
```

- look for -- Add user logic here on the generated code file.

- Add 4 instances of the pwm module

```
pwm_instX: pwm
    port map (
        clk => S_AXI_ACLK,
        SW => slv_regX(7 downto 0),
        reset =>S_AXI_ARESETN,
        outpwm => LED(X)
    );
```

each instance should use one of the 4 slv_reg as input for the duty cycle value, and one different bit of the led output port.

- Package the ip using ip creator tool (see packaging ip video tutorial)

- Create a system base design to test your design.

-add the constraints file to your design source(see adding constraints file video tutorial).

- generate the bitstream and export the hardware.

- lunch vivado sdk.

- use the debug interface to validate your hardware design behavior by writing to the module memory addresses. (see writing to memory addresses video tutorial).

- write a software application to assign different intensity values to each one of the 4 leds.

EX03:

in this exercise we will write a software application for our developed module to drive the pwm output signal that goes to the leds.

We are going to develop and axi counter hardware to generate a ramp function that will drive our pwm signal, the ramp signal will change the duty cycle each 1s by incrementing its value.

- write a vhdl module that increment an output signal each clock cycle.

clk	Clock signal	input
reset	Reset signal	input
Count value	Port to output the counting value	output

- comeback to the exercise 02 hardware design lunch the ip creator tool.

- add the counter vhdl file to the project sources.

- adapt the module to the axi lite interface to store the counting value on a slave register.

- test and package the developed ip.
(use the same previous technique to check your module behavior by reading its memory address value).

- add the ip to the previous exercice hardware design.

- run the connection automation.
- write the code that allows to increment a variable value (the variable should go between 0 and 100) each one second by checking the axi counter register value that stores the clock cycles number.
- write the variable value to the duty cycle value when it changes(each register should hold a different value for example the first two holds the actual variable value and the two others howlds 100-variable_value).

EX04:
using the ILA (integrated logic analyzer) tool to debug hardware.

In this exercise we will use the vivado integrated logic analyzer to visualize the generated signal when writing or reading to the developed axi module.

The customizable Integrated Logic Analyzer (ILA) IP core is a logic analyzer core that can be used to monitor the internal signals of a design. The ILA core includes many advanced features of modern logic analyzers, including Boolean trigger equations, and edge transition triggers. Because the ILA core is synchronous to the design being monitored, all design clock constraints that are applied to your design are also applied to the components inside the ILA core.

- add ILA ip to the previous hardware design.
- configure ILA to monitor an axi lite interface (see the ILA video tutorial for the steps).
- use the following code on the vivado sdk.

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include"xil_io.h"
#include"pwm_axi.h"
#include "xil_types.h"

#define reg1 PWM_AXI_S00_AXI_SLV_REG0_OFFSET
#define reg2 PWM_AXI_S00_AXI_SLV_REG1_OFFSET
#define reg3 PWM_AXI_S00_AXI_SLV_REG2_OFFSET
#define reg4 PWM_AXI_S00_AXI_SLV_REG3_OFFSET
#define AXI_PWM XPAR_PWM_AXI_0_S00_AXI_BASEADDR
int main()
{
    init_platform();

    print("Hello World\n\r");
    PWM_AXI_mWriteReg(AXI_PWM, reg1, 256);
    PWM_AXI_mWriteReg(AXI_PWM, reg2, 260);
       PWM_AXI_mWriteReg(AXI_PWM, reg3, 50);
       PWM_AXI_mWriteReg(AXI_PWM, reg4, 768);

    cleanup_platform();
    return 0;
}
```

- does the leds light on when running the program ?
- check the captured signal ? was the axi write transaction initiated ?

- why the leds doesn’t light on? Is it a hardware bug or a software bug ?

- correct the code.