**Lab report3**

**Submitted by: Nabila tabassum**

**Student ID: 40109039**

**Lab: COEN\_317\_FN\_X**

**Objective:**

This lab introduces the logicCORE IP AXI timer. The first portion of the lab introduces AXI timer which is being implemented as a simple timer or counter. The second portion uses timer in capture mode, to implement the storage of current value into an internal register. Lastly, the last portion, uses the pulse width modulation mode of the counter to control the brightness of the LED.

**Implementation:**

In the terminal, plan ahead was launch. Then, new project was created by selection the project name as

Lab3. No sources were specified by selecting VHDL as targeted language. A specification of UCF file for 8 LEDS was copied directly in the

COEN317/Lab3 directory. Afterwards, for the evaluation board, ZYNQ-7 ZC702 board was chosen, for

the project to be created.

Next, an embedded processor project was created with Add source wizard, by selecting add source

under project, then just add embedded source, create sub design, and named the system as module.

Then, the system was designed in XPS, by creating a Base System using the BSB Wizard, afterwards, the

AXI System should be selected by default. Later, Verification of the Zynq Processing System 7 is selected

by Removing the GPIO\_SW and LEDs\_4Bits Peripherals. Lastly, add the AXI General Purpose IO for the

and Change the port settings for axi\_gpio\_for\_\_output\_pins.

The following step is to export the Hardware to SDK by Right-click on “system (system.xmp)” and choose

Create Top HDL. Then, Right-click on “system (system.xmp)” and choose Create Top HDL, Afterwards,

synthesize the VHDL code by clicking on Run Synthesis on the left Panel. When the implementation of

synthesis was successful, then implementation was run. When implementation was successfully done,

Bitstream was generated upon successful completion of it. The bitstream was downloaded to the Zynq

Board by attaching the power cable, the Platform Cable USB II, and the serial cable for the UART.

Moreover, the impact was launched by applying power to the board and the verification of the Platform

Cable USB II status LED is being done by illuminated in green.

Lastly, using SDK application project was created. A new software project was created and then the file

lab2.sdk was copied into it. The program was compiled to build the executable and connection of the

SDK terminal was done to the board. Afterwards, the green connects buttons was connected and

desired connection types and port was selected. The Run the executable file on the board was run by

clicking run configuration.

**Part1:**

**Questions: 1;Use the timer to measure how long (in terms of number of clock cycles) it takes to perform a XGpio\_DiscreteWrite() and compare this to the to the time required to write to the GPIO port using the pointer dereference method. What conclusions can you draw from the two times?**

Ans: When XGpio\_DiscreteWrite() were used, it gave a GPIO port of 43 cycles. While using the pointer deference method, approx. 12 clock cycles were printed. From the above, it is obvious that deference method was faster because it writes the values manually on the AXI timer.

**Question2: If we wish are only interested in using the counter to read the counter register at certain points in time, does it make any difference whether the counter is programmed to operated in Generate mode or Capture mode ? Justify your answer.**

Ans: No certain difference is encountered when programming in generate or capture mode. As both of them, will manually return the clock cycles for time values.

**Part 3:**

Initially, the code works, by storing the values into the 0x26 by updating the values of pwm=1, gent0(external signal) =1, udt0(counter down)=1 for both the timer 0 and timer1. Then it, calculates the value of timer pointer and updates the values for both the timer by enabling the load to 1. Afterwards, it enables the timer by stopping the load, i.e (pwm=1, gent0(external signal)=1, udt0(counter down)=1, ENT( enable timer) = 1, load=0).

**Results and Conclusion:**

The part 3 portion of the lab works as it was expected to. The lab was done successfully, by using LogiCore IP AXI Timer for the PWM mode.

**Appendix:**

#include "stdbool.h"

#include "xparameters.h"

#include "xil\_types.h"

#include "xgpio.h"

#include "xil\_io.h"

#include "xil\_exception.h"

#include "xtmrctr.h"

#include <iostream>

using namespace std;

int main()

{

float DutyCycle;

float TimeCycle;

int xStatus;

static XGpio GPIOInstance\_Ptr;

XTmrCtr TimerInstance\_Ptr;

u32\* Timer\_Ptr = (u32\*)XPAR\_TMRCTR\_0\_BASEADDR;

cout << "#### counter Application Starts ####"<< endl;

if( DutyCycle > TimeCycle) {

cout << "" << endl;

return 1;

}

//Step-1: AXI GPIO Initialization

xStatus = XGpio\_Initialize(&GPIOInstance\_Ptr,XPAR\_AXI\_GPIO\_FOR\_OUTPUT\_DEVICE\_ID);

if(xStatus != XST\_SUCCESS)

{

cout << "GPIO A Initialization FAILED" << endl;

return 1;

}

\*(Timer\_Ptr) = 0x206; // pwm=1, gent0(external signal)=1, udt0(counter down)=1

\*(Timer\_Ptr+4) = 0x206; //

cout <<"enter duty cycle: " << endl;

cin >> DutyCycle;

cout <<"enter period : " << endl;

cin >> TimeCycle;

cout << "duty cycle: "<<DutyCycle<<endl;

cout << "time cycle: "<<TimeCycle<<endl;

// timer1 load

\*(Timer\_Ptr+1) = (TimeCycle\*50000000) -2;

//timer2 load

(Timer\_Ptr+5) = (TimeCycle(DutyCycle/100)\*50000000);

//load value into timer

\*(Timer\_Ptr) = 0x226; // pwm=1, gent0(external signal) =1, udt0(counter down)=1 , LDR=1

\*(Timer\_Ptr+4) = 0x226; // pwm=1, gent0(external signal) =1, udt0(counter down) =1 , LDR=1

// stop load and enable timer

\*(Timer\_Ptr) = 0x286; // pwm=1, gent0(external signal)=1, udt0(counter down)=1, ENT( enable timer) = 1

\*(Timer\_Ptr+4) = 0x286; // pwm=1, gent0(external signal)=1, udt0(counter down)=1, ENT( enable timer) = 1

return 0;

}