LAB 4

Std Name: Nabila Tabassum

Std ID: 40109039

Course: COEN317\_FN\_X.

INTRODUCTION:

This lab introduces the ARM processor for the interrupt structure using the AXI timer. The AXI timer is used to generate a clock cycle for its external interrupt port.

Theory:

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/Lab4 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, just enable the axi\_timer from the

peripheral window. Then, select the ‘Interrupt Port” checkbox in the “Show Ports” of the ‘Standard Filters”. Afterwards, verification of the connection from “L to H: axi\_timer’ should be done. Latly, exist the close the XPS by file -> exist.

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.

Graphical user interface, text, application, email

Description automatically generated

Conclusion:

To conclude, the lab was successful because it interrupts the timer for the ARM processor hardware.

Appendix:

#include <iostream>

using namespace std;

#include "xparameters.h"

#include "xil\_types.h"

#include "xtmrctr.h"

#include "xil\_io.h"

#include "xil\_exception.h"

#include "xscugic.h"

#include <stdio.h>

/\* Instance of the Interrupt Controller \*/

XScuGic InterruptController;

/\* The configuration parameters of the controller \*/

static XScuGic\_Config \*GicConfig;

// Timer Instance

XTmrCtr TimerInstancePtr;

int test = 0;

void Timer\_InterruptHandler(void)

{

//define the timer pointer

unsigned int\* timer\_ptr = (unsigned int\*)XPAR\_AXI\_TIMER\_0\_BASEADDR;

cout << "Value for first:" << \*(timer\_ptr) << " and second: " << \*(timer\_ptr + 4) << endl;

//stop both timers by setting the 8th bit to 0

\*(timer\_ptr) = 0x054;

\*(timer\_ptr + 4) = 0x054;

cout << "Value for first:" << \*(timer\_ptr) << " and second: " << \*(timer\_ptr + 4) << endl;

//read the value from the timer and determine which one called the interrupt

if(\*(timer\_ptr) & (1<<8))

{

cout << "the interrupt has been called by timer\_ptr" << endl;

}

else{ cout << "the interrupt has been called by timer\_ptr + 4" << endl;}

//prompt user to enter a value to begin the timer

char input;

cout << "Press any key to start the timer" << endl;

cin >> input ;

cout << "You pressed "<< input << endl;

cout << "Enabling the timer to start" << endl;

\*(timer\_ptr) = 0x0d4 ;

\*(timer\_ptr + 4) = 0x0d4 ;

}

int SetUpInterruptSystem(XScuGic \*XScuGicInstancePtr)

{

/\*

\* Connect the interrupt controller interrupt handler to the hardware

\* interrupt handling logic in the ARM processor.

\*/

Xil\_ExceptionRegisterHandler(XIL\_EXCEPTION\_ID\_INT,

(Xil\_ExceptionHandler) XScuGic\_InterruptHandler,

XScuGicInstancePtr);

/\*

\* Enable interrupts in the ARM

\*/

Xil\_ExceptionEnable();

return XST\_SUCCESS;

}

int ScuGicInterrupt\_Init(u16 DeviceId,XTmrCtr \*TimerInstancePtr)

{

int Status;

/\*

\* Initialize the interrupt controller driver so that it is ready to

\* use.

\* \*/

GicConfig = XScuGic\_LookupConfig(DeviceId);

if (NULL == GicConfig)

{

return XST\_FAILURE;

}

Status = XScuGic\_CfgInitialize(&InterruptController, GicConfig,

GicConfig->CpuBaseAddress);

if (Status != XST\_SUCCESS)

{

return XST\_FAILURE;

}

/\*

\* Setup the Interrupt System

\* \*/

Status = SetUpInterruptSystem(&InterruptController);

if (Status != XST\_SUCCESS)

{

return XST\_FAILURE;

}

XScuGic\_CPUWriteReg(&InterruptController, XSCUGIC\_EOI\_OFFSET, XPAR\_FABRIC\_AXI\_TIMER\_0\_INTERRUPT\_INTR);

/\*

\* Connect a device driver handler that will be called when an

\* interrupt for the device occurs, the device driver handler performs

\* the specific interrupt processing for the device

\*/

Status = XScuGic\_Connect(&InterruptController,

XPAR\_FABRIC\_AXI\_TIMER\_0\_INTERRUPT\_INTR,

(Xil\_ExceptionHandler)XTmrCtr\_InterruptHandler,

(void \*)TimerInstancePtr);

if (Status != XST\_SUCCESS)

{

return XST\_FAILURE;

}

/\*

\* Enable the interrupt for the device and then cause (simulate) an

\* interrupt so the handlers will be called

\*/

XScuGic\_Enable(&InterruptController, XPAR\_FABRIC\_AXI\_TIMER\_0\_INTERRUPT\_INTR);

return XST\_SUCCESS;

}

int main()

{

cout << "Application starts " << endl;

int xStatus;

//~~~~~~~~~

//Step-1 :AXI Timer Initialization

//~~~~~~~~~

xStatus = XTmrCtr\_Initialize(&TimerInstancePtr, XPAR\_AXI\_TIMER\_0\_DEVICE\_ID);

if(XST\_SUCCESS != xStatus)

{

cout << "TIMER INIT FAILED " << endl;

if(xStatus == XST\_DEVICE\_IS\_STARTED)

{

cout << "TIMER has already started" << endl;

cout << "Please power cycle your board, and re-program the bitstream" << endl;

}

return 1;

}

//~~~~~~~~~

//Step-2 :Set Timer Handler

//~~~~~~~~~

//

// cast second argument to data type XTmrCtr\_Handler since in gcc it gave a warning

// and with g++ for the C++ version it resulted in an error

XTmrCtr\_SetHandler(&TimerInstancePtr, (XTmrCtr\_Handler)Timer\_InterruptHandler, &TimerInstancePtr);

// intialize time pointer with value from xparameters.h file

unsigned int\* timer\_ptr = (unsigned int\* )XPAR\_AXI\_TIMER\_0\_BASEADDR;

//~~~~~~~~~

//Step-3 :load the reset value

//~~~~~~~~~

\*(timer\_ptr+ 1) = 0xf0000000;

\*(timer\_ptr+ 5) = 0xffffff00;

//~~~~~~~~~

//Step-4 : set the timer options

//~~~~~~~~~

// from xparameters.h file #define XPAR\_AXI\_TIMER\_0\_BASEADDR 0x42800000

// Configure timer in generate mode, count up, interrupt enabled

// with autoreload of load register

\*(timer\_ptr) = 0x0f4 ;

\*(timer\_ptr + 4) = 0x0f4 ;

//~~~~~~~~~

//Step-5 : SCUGIC interrupt controller Initialization

//Registration of the Timer ISR

//~~~~~~~~~

xStatus=

ScuGicInterrupt\_Init(XPAR\_PS7\_SCUGIC\_0\_DEVICE\_ID, &TimerInstancePtr);

if(XST\_SUCCESS != xStatus)

{

cout << " :( SCUGIC INIT FAILED )" << endl;

return 1;

}

// Beginning of our main code

//We want to control when the timer starts

char input;

cout << "Press any key to start the timer" << endl;

cin >> input ;

cout << "You pressed "<< input << endl;

cout << "Enabling the timer to start" << endl;

\*(timer\_ptr) = 0x0d4 ; // deassert the load 5 to allow the timer to start counting

\*(timer\_ptr + 4) = 0x0d4 ;

// let timer run forever generating periodic interrupts

while( 1)

{

// // wait forever and let the timer generate periodic interrupts

}

return 0;

}