***Intro to System-on-Chip Design Course***

**LAB 8**

**Interrupt Mechanisms**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc83817560)

[1.1 Lab overview 1](#_Toc83817561)

[1.1.1 Hardware design and implementation: 1](#_Toc83817562)

[1.1.2 Software programming: 1](#_Toc83817563)

[1.1.3 Demonstrate the SoC: 1](#_Toc83817564)

[2 Learning Objectives 1](#_Toc83817565)

[3 Requirements 2](#_Toc83817566)

[4 Project files 2](#_Toc83817567)

[5 Hardware 3](#_Toc83817568)

[5.1 Implementation of the timer interruption mechanism 3](#_Toc83817569)

[5.2 Implementation of the UART interruption mechanism 3](#_Toc83817570)

[5.3 Connect the interrupt to the processor 4](#_Toc83817571)

[6 Software 5](#_Toc83817572)

[6.1 Software tasks 5](#_Toc83817573)

[7 Hardware debugging 6](#_Toc83817574)

[7.1 On-chip debugging 6](#_Toc83817575)

[8 Extension work 6](#_Toc83817576)

[8.1 Extra tasks for this lab: 6](#_Toc83817577)

# Introduction

## Lab overview

In this lab, we will implement an interrupt mechanism for the timer and the UART peripherals. The steps to do this include:

### Hardware design and implementation:

The processor, bus interface, on-chip memory and peripheral hardware are written in Verilog and provided for you, with some modification/additions needed to make it work. The SoC will Implement interrupt mechanisms for the AHB timer and the AHB UART.

### Software programming:

The program targeted at the Cortex-M0 processor contains interrupt service routines for the timer and the UART peripherals.

### Demonstrate the SoC:

* Use the timer interrupt to implement a counter (counting from 0 to 9) and display the value to the VGA display.
* Use the UART interrupt to send characters to a PC or laptop.

# Learning Objectives

* Implement the AHB timer and UART interrupt mechanism at both hardware and software domains by adding appropriate interrupt registers and writing suitable interrupt handler.

# Requirements

This lab requires the following hardware and software:

* **Hardware:**
  + **Diligent BASYS 3** FPGA board connected to computer via **MicroUSB cable.** A constraints file for this board is also provided.
  + **VGA-compliant monitor** and **VGA cable** to connect your board
* **Software**
  + Xilinx Vivado
  + Keil uVision
  + TeraTerm

# Project files

You will need the files from the previous lab (other than **AHBLITE\_SYS.v**) along with the following files which are provided with this Lab:

|  |  |
| --- | --- |
| **File name** | **Description** |
| AHBLITE\_SYS.v | The top-level module |

# Hardware

The hardware components of the SoC include all the peripherals that have been developed in the previous modules.

The AHB timer and the AHB UART will need to be reimplemented with interrupt signals.

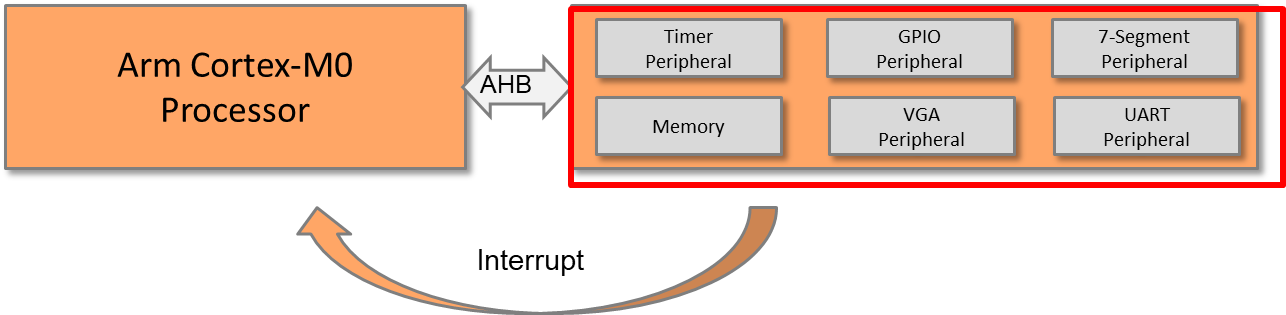


Figure :SoC Peripheral Interrupts

## Implementation of the timer interruption mechanism

An interrupt is generated every time the counter reaches zero.

A clear register needs to be added; this is used to clear the interrupt request once the processor finishes its ISR.

Diagram

Description automatically generated

Figure :Timer Interrupt Signal

TIMER PERIPHERAL REGISTERS

|  |  |  |
| --- | --- | --- |
| **Register** | **Base address** | **Size** |
| Load value | 0x5300\_0000 | 4 bytes |
| Current value | 0x5300\_0004 | 4 bytes |
| Control value | 0x5300\_0008 | 4 bytes |
| Clear register | 0x5300\_000C | 4 bytes |

## Implementation of the UART interruption mechanism

For example, the interrupt can be generated if the receiver FIFO is not empty.

**Diagram

Description automatically generated**

Figure :UART Interrupt Signal

Unlike the timer interruption, we can omit the interrupt cleaning step, since the interrupt request will be automatically cleared after the data is read out from the FIFO.

## Connect the interrupt to the processor

A picture containing icon

Description automatically generated

Figure :Connect Interrupt Signals to the Processor

# Software

## Software tasks

The main code should be written in assembly and should perform the following:

* Initialize the interrupt vector, adding the timer and the UART interrupt vectors
* Reset handler
  + Set the timer interrupt priority to 0x00 (higher).
  + Set the UART interrupt priority to 0x40 (lower).
  + Enable interrupts for the timer and UART.
  + Initialize the timer to generate an interrupt every second.
    - Write the load value register, e.g., 50,000,000
    - Set prescaler, e.g., 1x or 16x.
    - Change the operation mode to load mode.
  + Start the timer.
  + Set up a counting up counter; start from “0” (ASCii=0x30).
* Timer interrupt handler
  + Push registers to the stack (e.g., R1-R4).
  + Clear the timer interrupt request.
  + Increment the counter.
  + Display the counter to the VGA text region.
  + Disable the timer interrupt if the counter reaches 9.
  + Pop the registers from the stack.
* UART interrupt handler
  + Push registers to the stack.
  + Read from the UART (from the keyboard).
  + Write to the UART (to the terminal window).
  + Pop the registers from the stack.

# Hardware debugging

## On-chip debugging

Use an on-chip debugging tool to sample and analyse the signals at run-time. Suggested signals are as follows:

Towards AHB bus:

* HADDR[31:0]
* HWDATA[31:0]
* HRDATA[31:0]
* HWRITE
* HREADY
* HSIZE[2:0]
* HTRANS[1:0]
* HRESP

Towards the peripherals:

* Processor\_IRQ
* Timer\_IRQ
* UART\_IRQ

# Extension work

## Extra tasks for this lab:

* Implement the interrupt mechanism for the GPIO peripheral, e.g., use switch as external interrupts.
* Implement a watchdog and connect it to the non-maskable interrupt (NMI).