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

**LAB 7**

**Timer, GPIO and 7-Segment Peripherals**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc83817578)

[1.1 Lab overview 1](#_Toc83817579)

[1.1.1 Hardware design and implementation: 1](#_Toc83817580)

[1.1.2 Software programming: 1](#_Toc83817581)

[1.1.3 Demonstrate the SoC: 1](#_Toc83817582)

[2 Learning Objectives 1](#_Toc83817583)

[3 Requirements 2](#_Toc83817584)

[4 Project files 2](#_Toc83817585)

[5 Hardware 3](#_Toc83817586)

[**5.1** **Overview of the SoC hardware** 3](#_Toc83817587)

[5.2 Timer Peripheral 3](#_Toc83817588)

[5.2.1 Timer Peripheral Block diagram 3](#_Toc83817589)

[5.3 GPIO Peripheral 4](#_Toc83817590)

[5.3.1 GPIO Peripheral Block diagram 4](#_Toc83817591)

[5.4 7-Segment Display Peripheral 4](#_Toc83817592)

[5.4.1 Peripheral Block diagram 4](#_Toc83817593)

[5.5 Memory map 4](#_Toc83817594)

[6 Software 5](#_Toc83817595)

[6.1 Software tasks 5](#_Toc83817596)

[7 Hardware Debugging 6](#_Toc83817597)

[7.1 On-chip debugging 6](#_Toc83817598)

[8 Extension Work 7](#_Toc83817599)

[8.1 Extra task for this lab: 7](#_Toc83817600)

# Introduction

## Lab overview

In this lab, we implement three AHB peripherals: an internal timer, a general-purpose input output (GPIO), and a 7-segment display. 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 be implemented in an FPGA.

### Software programming:

The program targeted at the Cortex-M0 processor is written in assembly language and will be used to access the three peripherals. The program is provided for you. You will need to compile it in Keil to generate a *code.hex* file which will be copied to the FPGA project directory.

### Demonstrate the SoC:

* Display the timer on the 7-segment display.
* Use GPIO to interface with the switches and LEDs.
* Analyze the behavior of the peripheral using an on-chip hardware debugging tool.

# Learning Objectives

* Implement a simple SoC which consist of Cortex-M0 processor, AHB-Lite bus and AHB peripherals (Program memory and LED, VGA, UART, Timers, GPIOs and 7-Segment) on an FPGA.
* Modify and compile a simple assembly code to initialize the timer.
* Modify and compile a simple assembly code to initialize and start the timer.
* Modify and compile a simple assembly code to display the timer value on the 7-segment display

# 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** |
| AHBTIMER.v | The AHB timer peripheral, including the AHB bus interface |
| prescaler.v | The prescaler is used to scale-down the frequency for the timer. |
| AHBGPIO.v | The AHB GPIO peripheral, including the AHB bus interface |
| AHB7SEGDEC.v | The AHB 7-segment display peripheral, including the AHB bus interface |

# Hardware

* 1. **Overview of the SoC hardware**

The hardware components of the SoC include:

* An Arm Cortex-M0 microprocessor
* An AHB-Lite system bus
* Six AHB peripherals
  + A BRAM memory module
  + A VGA peripheral
  + A UART peripheral
  + An internal timer peripheral
  + A GPIO peripheral
  + A 7-segment display peripheral

Diagram

Description automatically generated

**SoC Peripherals**

## Timer Peripheral

### Timer Peripheral Block diagram

Diagram

Description automatically generated

Figure :Timer Peripheral Block Diagram

­

## GPIO Peripheral

### GPIO Peripheral Block diagram

Diagram

Description automatically generated

Figure :GPIO Peripheral Block Diagram

## 7-Segment Display Peripheral

### Peripheral Block diagram

Diagram

Description automatically generated

Figure :7-Segment Display Peripheral Block Diagram

## Memory map

The default memory map of the peripherals is listed below:

MEMORY MAP OF PERIPHERALS

|  |  |  |  |
| --- | --- | --- | --- |
| **Peripheral** | **Base address** | **End address** | **Size** |
| SRAM | 0x0000\_0000 | 0x00FF\_FFFF | 16MB |
| VGA | 0x5000\_0000 | 0x50FF\_FFFF | 16MB |
| UART | 0x5100\_0000 | 0x51FF\_FFFF | 16MB |
| Timer | 0x5200\_0000 | 0x52FF\_FFFF | 16MB |
| GPIO | 0x5300\_0000 | 0x53FF\_FFFF | 16MB |
| 7-segment display | 0x5400\_0000 | 0x54FF\_FFFF | 16MB |

TIMER PERIPHERAL REGISTERS

|  |  |  |
| --- | --- | --- |
| **Register** | **Base address** | **Size** |
| Load value | 0x5200\_0000 | 4 bytes |
| Current value | 0x5200\_0004 | 4 bytes |
| Control value | 0x5200\_0008 | 4 bytes |

* Load value register: The reset value when the timer reaches zero
* Current value register: The current value of the 32-bit counter
* Control register: Used to start/stop a counter and set the prescaler

GPIO PERIPHERAL REGISTERS

|  |  |  |
| --- | --- | --- |
| **Register** | **Base address** | **Size** |
| Data | 0x5300\_0000 | 4 bytes |
| Direction | 0x5300\_0004 | 4 bytes |

* Data registers
  + Input data: the data read from external devices
  + Output data: the data sent to external devices
* Direction register
  + Controls whether it is a read or write operation

7-SEGMENT DISPLAY PERIPHERAL REGISTERS

|  |  |  |
| --- | --- | --- |
| **Register** | **Base address** | **Size** |
| Digit1 | 0x5400\_0000 | 4 bytes |
| Digit2 | 0x5400\_0004 | 4 bytes |
| Digit3 | 0x5400\_0008 | 4 bytes |
| Digit4 | 0x5400\_000C | 4 bytes |

* Digit 1: the first digit on the 7-segment display
* Digit 2: the second digit on the 7-segment display
* Digit 3: the third digit on the 7-segment display
* Digit 4: the fourth digit on the 7-segment display

# Software

## Software tasks

Create a new Keil project, add the provided assembly file. This time you will need to edit this file to do the following. When done build the program and copy the ***code.hex*** file for the Vivado project.

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

* Initialize the interrupt vector.
* Display a string (e.g., “TEST”) at the console region (same as previous lab).
* Initialize the timer.
  + Write the load value register.
  + Set prescaler, e.g., 1x or 16x.
  + Change the operation mode.
    - Free-run mode: the timer resets on overflow.
    - Load mode: the timer resets when reaching the value in the load register.
  + Start the timer

Then repeat the following:

* Display the timer
  + Read the current timer value.
  + Choose the higher bytes, e.g., 16-bit MSB.
  + Display the value on the 7-segment display using the four digits.
* GPIO test
  + Change the direction of the GPIO to read mode.
  + Read the switch value from the GPIO.
  + Change the direction of the GPIO to write mode.
  + Write the switch value to the LEDs.

# Hardware Debugging

## On-chip debugging

Use an on-chip debugging tool to sample and analyze 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:

* Timer\_Select
* Timer\_Write
* Timer\_Ready
* Timer\_Data
* GPIO\_Select
* GPIO \_Write
* GPIO \_Ready
* GPIO \_Data
* 7-SEG\_Select
* 7-SEG \_Write
* 7-SEG \_Ready
* 7-SEG \_Data

# Extension Work

## Extra task for this lab:

* In the timer peripheral, add more operation modes to the timer, such as compare mode, capture mode, or PWM mode.
* In the GPIO peripheral, add a mask register that can mask out certain bits while writing to the GPIO.
* In the 7-segment display peripheral, add a register to change the display mode, such as display in hex or decimal.