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

**LAB 9**

**Programming an SoC using C Language**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc83817548)

[1.1 Lab overview 1](#_Toc83817549)

[1.1.1 Software programming: 1](#_Toc83817550)

[1.1.2 Demonstrate the SoC: 1](#_Toc83817551)

[2 Learning Objectives 1](#_Toc83817552)

[3 Requirements 2](#_Toc83817553)

[4 Provided files 2](#_Toc83817554)

[5 Software 3](#_Toc83817555)

[5.1 Program procedure 3](#_Toc83817556)

[5.2 An example of the demo: 4](#_Toc83817557)

[6 Extension work 4](#_Toc83817558)

[6.1 Extra tasks for this lab: 4](#_Toc83817559)

# Introduction

## Lab overview

In the previous labs, assembly language was used to program the SoC at low-level, which gave better control over the hardware and helped us understand the hardware mechanism at register transfer level (RTL).

In this lab, C language will be used to program the SoC at a higher level, whereby compilers, software libraries, and tools can be used to hide the complexity of low-level management. This means that we can make more complicated applications in a more efficient way.

Graphical user interface

Description automatically generated with medium confidence

One can always mix C and assembly code in one project, or even in one file.

Hardware implementation has been completed in the previous labs; the following labs will focus on software. The work in this lab includes:

### Software programming:

* Program the Cortex-M0 processor and control the peripherals using C language.

### Demonstrate the SoC:

* Use the timer interrupt to implement a counting up counter (counting from 0 to 9) and display the value to the VGA display (same as previous lab but using C programming).
* Use the UART interrupt to send characters to a PC or laptop (same as previous lab but using C programming).
* Input from 8-bit switch and output to LEDs

# Learning Objectives

* Implement a timer interrupt handler and UART interrupt handler in a high-level language such as C.
* Modify a C program to enable the timer and UART interrupts.
* Modify a C program to set the interrupt priority of the timer and UART interrupts.
* Modify a C program to initialise the timer to generate an interrupt at regular interval.

# 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

# Provided files

You will need the files from the previous labs along with the following files which are provided with this Lab:

|  |  |
| --- | --- |
| **File name** | **Description** |
| cm0dsasm.s | The assembly file that includes interrupt vectors and part of the interrupt service routines |
| main.c | The C code that contains the main program and part of the interrupt service routines |

# Software

## Program procedure

The assembly code in cm0dsasm.s should perform the following:

* Initialize the interrupt vector.
* Define heap and stacks.
* Reset handler.
  + Branch to the main code in main.c.
* Timer handler
  + Push registers (e.g., R1 – R4) to the stack.
  + Branch to the timer interrupt service routine in main.c.
  + Pop registers from the stack.
* UART handler
  + Push registers (e.g., R1 – R4) to the stack.
  + Branch to the UART interrupt service routine in main.c.
  + Pop registers from the stack.

The C code in main.c should perform the following:

* Main program
  + Set the timer interrupt priority to 0x00 (higher).
  + Set the UART interrupt priority to 0x40 (lower).
  + Enable interrupts for the timer and UART.
  + Reset the 7-segment display.
  + 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.
  + Repeat the following:
    - Change the GPIO to read mode.
    - Read from the switches using the GPIO.
    - Change the GPIO to write mode.
    - Write to the LEDs.
* Timer interrupt handler
  + 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.
* UART interrupt handler
  + Read from the UART (from the keyboard).
  + Write to the UART (to the terminal window).

## An example of the demo:

A screenshot of a computer

Description automatically generated with medium confidence

Figure 1:VGA Demo

# Extension work

## Extra tasks for this lab:

* Optimize the code in both operational speed and code density. You can use:
  + Embedded assembly code in C code
  + Embedded C code in assembly code
* Evaluate the coding efficiency between handwritten assembly code and assembly generated by the compiler.