***Intro to SoC Course***

**LAB 10**

**Arm CMSIS and Software Drivers**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc83817641)

[1.1 Lab overview 1](#_Toc83817642)

[1.1.1 Software programming: 1](#_Toc83817643)

[1.1.2 Demonstrate the SoC: 1](#_Toc83817644)

[2 Learning Objectives 1](#_Toc83817645)

[3 Requirements 2](#_Toc83817646)

[4 Provided files 2](#_Toc83817647)

[5 Software 3](#_Toc83817648)

[5.1 Restructure your files 3](#_Toc83817649)

[5.2 Add CMSIS files to your project: 3](#_Toc83817650)

[5.3 Create and use your own software drivers 4](#_Toc83817651)

[5.4 File structure 5](#_Toc83817652)

[5.5 Program procedure 5](#_Toc83817653)

[6 Extension work 6](#_Toc83817654)

[6.1 Extra tasks for this lab: 6](#_Toc83817655)

# Introduction

## Lab overview

In the previous lab, we introduced C language, a high-level language to program the SoC, allowing us to write more complicated applications. In this lab, we will introduce Cortex Microcontroller Software Interface Standard (CMSIS), which is an abstraction layer for the microcontroller, and defines generic tool interfaces. It simplifies software reuse. 

The work in this lab includes:

### Software programming:

* Configure the Cortex-M0 processor using Arm CMSIS.
* Write software drivers for the hardware peripherals.

### Demonstrate the SoC:

* Use the timer interrupt to increment a counter in every second and display the counter on the segment display (using driver functions).
* Use the UART interrupt to send characters to a PC or laptop (using driver functions).
* Input from an 8-bit switch and output to LEDs (using driver functions).

# Learning Objectives

* Add CMSIS files to the Keil project and write custom software drivers for peripherals.
* Modify a C program to initialise the processor and NVIC using CMSIS functions.
* Modify a C program to enable interrupts using CMSIS functions.
* Modify a C program to initialise the timer using timer functions.
* Modify the CMSIS based timer handler driver function to clear interrupt request, increment a counter and update seven segment displays.
* Modify the CMSIS based UART handler driver function to read from and write to the UART.

# 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** |
| core\_cm0.h | CMSIS Cortex-M0 core peripheral access layer header file |
| core\_cmFunc.h | CMSIS Cortex-M core function access header file |
| core\_cmInstr.h | CMSIS Cortex-M core instruction access header file |
| cm0dsasm.s | Includes interrupt vectors and other setup assembly code |
| main.c | Includes the main program and interrupt service routines  |
| EDK\_CM0 | Defines the interrupt numbers and memory map etc. |

# Software

## Restructure your files

To use CMSIS, the files need to be included in your project.

Before adding CMSIS, it would be better to restructure our project repositories. Suggestions are as follows:

1. Open your μvision project directory and create a new folder called ***“Application”*** and move ***“main.c”*** into it. This folder should contain files used for application programming.
2. In the project directory, create a new folder called ***“Device”*** and move ***“cm0dsasm.s”*** into it. This folder should contain files that are specific to the device, e.g., software drivers for peripherals.
3. Reopen the project and remove all groups and files (*See section 3.9 in the Getting Started Guide*).
4. Create two new groups and name them “Application” and “Device” (*See section 3.7 in the Getting Started Guide*).
5. Move ***“main.c”*** and “cm0dsasm.s” to “Application” and “Device,” respectively (*See section 3.4 in the Getting Started Guide*).

## Add CMSIS files to your project:

1. Under the project directory, create a new folder called “Core” that will contain processor-related files.
2. Copy the CMSIS files into the folder; the files include:

|  |  |
| --- | --- |
| **File name** | **Description** |
| core\_cm0.h | CMSIS Cortex-M0 core peripheral access layer header file |
| core\_cmFunc.h | CMSIS Cortex-M core function access header file |
| core\_cmInstr.h | CMSIS Cortex-M core instruction access header file |

1. Under the “Device” folder, create a new header file called “EDK\_CM0.h” (educational development kit); this will be used to define our SoC and its peripherals.
2. In “EDK\_CM0.h,” define the interrupt number, processor core, system memory, and peripheral memory map.
3. Include the path of all header files to your project (Project Configuration 🡪 C/C++ 🡪 Include Paths).



1. Rewrite your program using CMSIS functions.

## Create and use your own software drivers

1. Under the “Device” folder, create a new file named “edk\_driver.c”; this will contain the driver functions for all peripherals.
2. Under the “Device” folder, create another file called “edk\_driver.h”; this will include the names of functions in “edk\_driver.c”.
3. Add the files to your project.
4. Include “edk\_driver.h” in your main program.
5. Write software functions for your peripherals. Suggested functions are as follows:

|  |  |
| --- | --- |
| **File name** | **Description** |
| cm0dsasm.s | Includes interrupt vectors and other setup assembly code |
| main.c | Includes the main program and interrupt service routines  |
| EDK\_CM0 | Defines the interrupt numbers and memory map etc. |

## File structure

Core folder

Device folder

Application folder

core\_cm0.h

cm0dsasm.s

main.c

core\_cmFunc.h

core\_cmInstr.h

EDK\_CM0.h

edk\_driver.c

edk\_driver.h

EXAMPLE FUNCTIONS

|  |  |  |
| --- | --- | --- |
| **Peripheral**  | **Function**  | **Description**  |
| VGA | void VGA\_plot\_pixel (int x, int y, int col); | Plot a pixel in the image region. |
| 7-segment display | void seven\_seg\_write(char dig1, char dig2,char dig3,char dig4); | Write four digits on the 7-segment display. |
| Timer | void timer\_init (int load\_value, int prescale, int mode); | Initialize the timer. |
| void timer\_enable(void); | Enable the timer. |
| void timer\_irq\_clear(void);  | Clear interrupt request from the timer. |
| GPIO | int GPIO\_read(void) | Return with the value read from the input port. |
| void GPIO\_write(int data) | Write a value to the GPIO output. |

## Program procedure

At this stage, the processor and the peripherals should be accessed through functions, rather than through writing/reading registers.

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

* Initialize the interrupt vector.
* Define heap and stack.
* 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
	+ Initialize the processor and the nested vectored interrupt controller (NVIC) using CMSIS functions.
	+ Enable the interrupts using CMSIS functions.
	+ Initialize and start the timer using timer driver functions.
	+ Then repeat the following:
		- Read the value from the switches using GPIO functions.
		- Write the value to the LEDs using GPIO functions.
* Timer interrupt handler
	+ Clear the timer interrupt request using the timer driver function.
	+ Increment the counter.
	+ Display the counter to the 7-segment in decimals using driver functions.
* UART interrupt handler
	+ Read from the UART (from the keyboard) using UART driver functions.
	+ Write to the UART (to the terminal window) using UART driver functions.

# Extension work

## Extra tasks for this lab:

* Explore the other functions provided by CMSIS, e.g., special instructions and control pending status of interrupts.
* Develop more driver functions for your peripherals.