***Embedded SoC Design Course***

**LAB 10 (Project)**

**API and Final Application**

Contents

[1 Introduction 1](#_Toc83817623)

[1.1 Lab overview 1](#_Toc83817624)

[1.1.1 Software programming: 1](#_Toc83817625)

[1.1.2 Demonstrate the SoC: 1](#_Toc83817626)

[2 Learning Objectives 1](#_Toc83817627)

[3 Requirements 2](#_Toc83817628)

[4 Provided files 2](#_Toc83817629)

[5 Software 3](#_Toc83817630)

[5.1 API Development 3](#_Toc83817631)

[5.1.1 Create API file 3](#_Toc83817632)

[5.1.2 Add the retarget file 3](#_Toc83817633)

[5.1.3 File structure 4](#_Toc83817634)

[5.2 Application development 5](#_Toc83817635)

[5.2.1 Application using polling 5](#_Toc83817636)

[5.2.2 Application using interrupt (power saving) 5](#_Toc83817637)

[5.2.3 Example of the demo: 6](#_Toc83817638)

[6 Tasks for this lab 6](#_Toc83817639)

# Introduction

## Lab overview

In previous labs, we used the CMSIS and developed drivers for the peripherals. In this lab, we will develop an API that has more generic and easy-to-use functions. Then, based on the API, we will develop a final game application: Snake.

The work in this lab includes:

### Software programming:

* Using the functions provided by software drivers and CMSIS, develop an API that provides more generic and easy-to-use functions for application development.

### Demonstrate the SoC:

* Develop a final application (such as the Snake game) to demonstrate the SoC
* Use sleep mode to reduce the power consumption of your application

Diagram

Description automatically generated

Figure 1: API and Application Development hierarchy

# Learning Objectives

* Modify reusable and easy to use APIs.
* Develop a single player snake game using C program which implements UART and timer interrupt handlers to control and display/ detect the snake’s coordinates.

# Requirements

This lab requires the following hardware and software:

* **Hardware:**
  + **Diligent** **Nexys A7** 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

## API Development

### Create API file

Under the “Device” folder, create a head file called “API.h” and a C file called “API.c”.

Write the functions in “API.c” and include all the function calls in “API.h”.

Suggested functions are as follows:

|  |  |
| --- | --- |
| **API Functions** | **Description** |
| void SoC\_init(void) | SoC initialization |
| void rectangle(int x1,int y1,int x2,int y2, int color) | Draw a rectangle on the screen. |
| void clear\_screen (void) | Clean up the screen. |
| int read\_switch | Read the value of the 8-bit switches. |
| write\_LED | Write a value to the 8-bit LEDs. |
| void Display\_Int\_Times (void) | Display the number of interrupts that occurred using the 7-segment display. |
| void delay(int value) | Software delay program |
| char random (char min, char max) | A simple random generator based on system tick |

### Add the retarget file

The retarget file allows us to use print library functions such as “printf()”. To add the retarget file:

* Add the “retarget.c” file to the “Device” folder.
* Implement the retarget functions, for example:

|  |  |
| --- | --- |
| **Retarget Functions** | **Description** |
| int KBHIT(void) | Wait for keyboard hit. |
| int fputc(int ch, FILE \*f) | Input characters |
| int fgetc(FILE \*f) | Output characters |
| unsigned char VGAPutc(unsigned char my\_ch) | Output characters to VGA |
| unsigned char UartPutc(unsigned char my\_ch) | Output characters to UART |
| unsigned char UartGetc(void) | Input characters from UART |

Example code:

//define UartPutc

unsigned char UartPutc(unsigned char my\_ch)

{

UART->DATA=my\_ch;

return (my\_ch);

}

//define fputc

int fputc(int ch, FILE \*f) {

return (UARTPutc(ch));

}

//use printf in main.c

printf("HelloWorld");

### File structure

The files can be organized as follows:

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

edk\_api.c

edk\_api.h

retarget.c

## Application development

The following ideas can be used to program the Snake game:

### Application using polling

* Diagram

  Description automatically generatedMain program
  + Initialize the SoC.
  + Initialize the game.
  + Repeat the following:
    - Check if keyboard hits; if yes, then,
      * Update snake direction.
    - Move the snake.
    - Check if it hits the wall; if yes, then,
      * Game over
    - Delay for a short time.

### Application using interrupt (power saving)

* Main program
  + Initialize the SoC.
  + Initialize the game.
  + Enter the sleep-on-exit mode.
* Timer interrupt handler
  + Trigger the snake to move one step.
  + Detect if the target is reached or if the snake hits the wall.
* UART interrupt handler
  + Input the command from the keyboard.
  + Change the direction of the snake.

**­Diagram

Description automatically generated**

**Interrupt-driven Mode**

### Example of the demo:

A picture containing text, monitor, electronics, screen

Description automatically generated

**Demo Example**

# Tasks for this lab

* The main.c is provided in the Application folder, which has snake game implementation. The main.c is incomplete, and you need to complete it to finalize the single-player snake game. In main.c, look for the following comments in the Timer\_ISR() function, where you required to add the code. **(Mandatory)**
  + // TO DO: Generate Target
  + //TO DO: Draw the target
  + //TO DO: Detect if the snake hits itself and end game if it does
  + //TO DO: Detect if the snake hits the boundary

Additionally, optimize your code to reduce the power consumption of your application.

* Implement a two-player snake game. The color of one snake should be red, and the other should be green. The score should be maintained for both the snakes. One point should be added to the score when any snake eats food. On the death of any of the snakes, five points should be added to the score of another snake. If snakes are dying from a head-on collision, no points should be added to either’s score. However, if one snake is dying by colliding with another, then five points need to be added to the other snake’s score. If any snake is dying by colliding with the wall, then five points need to be added to the other snake’s score. The winning player should be displayed with the final score.**(Mandatory)**
* You can implement any one of the following games. You need to define the rules for the game. Make sure to display the rules and relevant information of the game in the text console, and the game should be played in the image console. **(Optional)**
  + TERIS
  + PACMAN
  + TIC-TAC-TOE