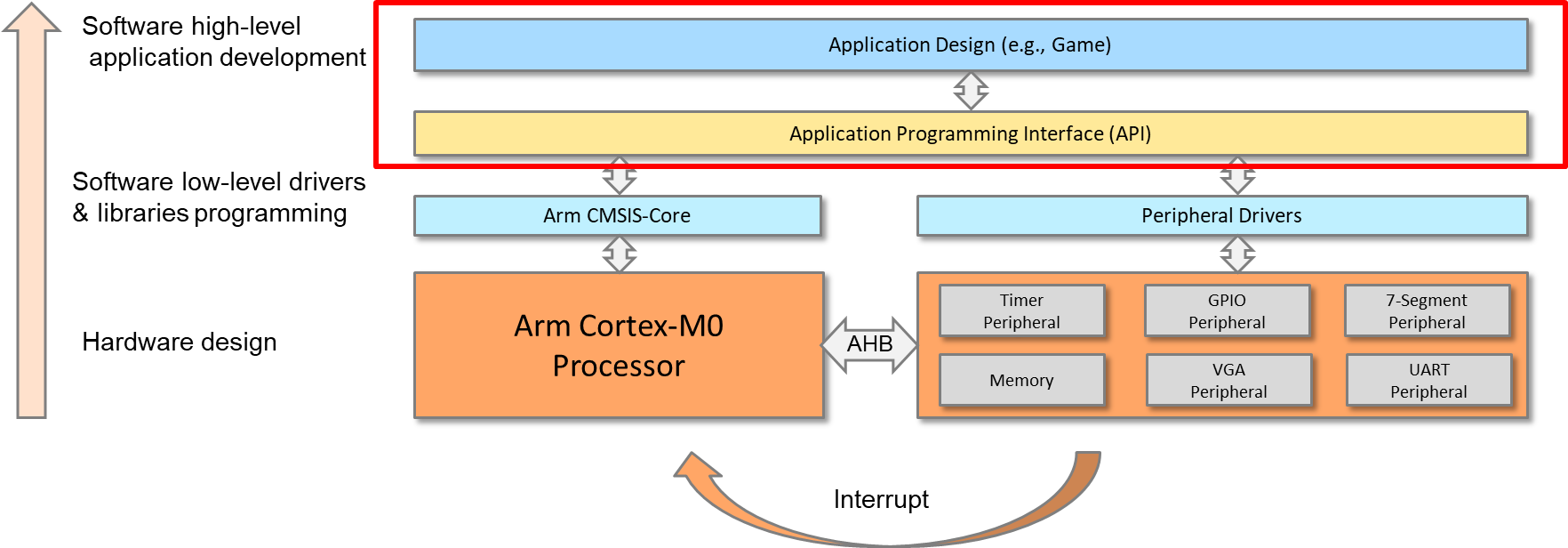
Lab Exercise 11: API and Final Application

# 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.

In this lab, the work 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



**API and Application Development**

# Details

## 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

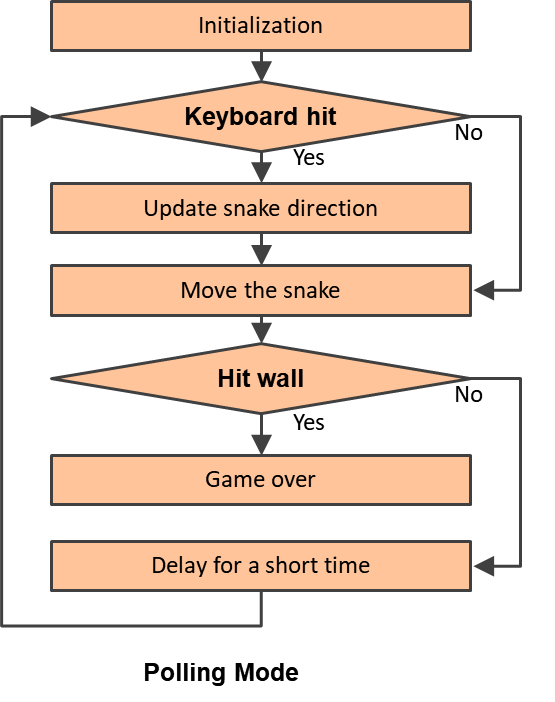
edk\_api.h

retarget.c

### application development

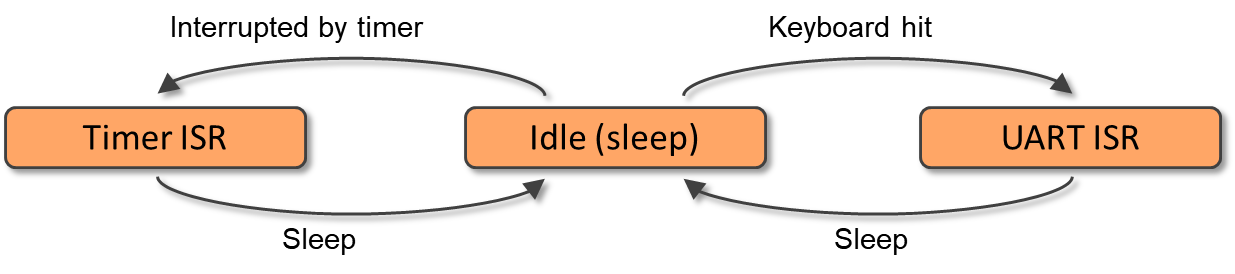
The following ideas can be used to program the Snake game:

#### Application using polling

* Main 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.

**­**

**Interrupt-driven Mode**

#### example of the demo:



**Demo Example**

# extension work

Here are some extra things that you can do:

* Use a sampling energy meter (or other equipment) to measure the run-time power consumption of your game application.
* Optimize your code to reduce power consumption.
* Explore other games, such as TERIS, PACMAN, BREAK, TICTAC, etc.