#cec32xNotes, cec32x_ws_26_WS05_printing_in_binary_format

## WS 5. Discovery Board Programming with CMSIS

### Introduction

In Workshop 3, we used code generated by STM32CubeMX to control an LED and the Joystick. We have also used the HAL GPIO drivers/functions to control the LED and Joystick key. While using the HAL driver is very important to quickly develop a project, understanding the control of the GPIO using CMSIS (via GPIO registers) is very important for improving the understanding of the details of the MCU. In this workshop, we use CMSIS to control these devices directly, as detailed below.

We perform this workshop by modifying/extending the code of Workshop 3. Specifically, we extend the software functionalities of Workshop 3 without using HAL functions for GPIO control. Note that we have only used LD_R and JOY_L in Workshop 3. We will use LD_G (the green user LED) and JOY_R (the right key of Joystick) for added functionalities.

The functionalities of the project are as follows:

- When JOY_L (the left key of Joystick) is pressed, the program turns on LD_R (the red user LED) and turns off LD_G.
- When JOY_R (the right key of Joystick) is pressed, the program turns on LD_G and turns off LD_R.
- When no key is pressed, LD_R and LD_G will toggle in an interval of 100 milliseconds.

Note that you can get started using two approaches:

- Getting started with the working code from Workshop 3. For this approach, you need to work in CubeMX to configure the additional GPIO pins for LD_G and JOY_R.
- Getting started with the working code of Lab 3.

For whatever approach you use, please name the project top folder as `cec320_ws05_GPIO_CMSIS`. Then put your `F412Discovery` or `L476Discovery` folder under it. Also, add your `usr_src` folder under it.

Below, we assume all the GPIO pins are set up appropriately and the base code is generated. If you have questions, please ask.

### Workshop tasks

**Task 1: Building the generated code** (10 points)

Open the project directory using File Explorer and navigate to the ''MDK-ARM'' folder of your specific board. Double-click the ''.uvprojx'' file to open the project in Keil. Press the **Build** button. You should be able to build the

project without error. Of course, the code will do nothing noticeable since we do not have any user code yet. If you cannot build the project without error, please inform the instructor or the TA.

**Task 2: Adding user source files for user task programming** (10 points)

Note that we will put all our special user code in the `cec320_ws05_GPIO_CMSIS/usr_src` folder. Create this folder if you have not done so yet.

Copy `user_tasks.c` and `user_tasks.h` from Workshop 3 to this folder. For your reference, these two files should be, respectively:

The `user_tasks.c` file:

```
#include "usr_tasks.h"

////////////////////////////////////////////////

void USR_Task_RunLoop(void) {
    if (HAL_GPIO_ReadPin(JOY_L_GPIO_Port, JOY_L_Pin)) { // Checking the left key of the Joys
#if defined(STM32L476xx)
        HAL_GPIO_WritePin(LD_R_GPIO_Port, LD_R_Pin, GPIO_PIN_SET);
#elif defined(STM32F412Zx)
        HAL_GPIO_WritePin(LD_R_GPIO_Port, LD_R_Pin, GPIO_PIN_RESET);
#endif
    } else {
        HAL_GPIO_TogglePin(LD_R_GPIO_Port, LD_R_Pin);
        HAL_Delay(200);
    }
}
```

The `user_tasks.h` file:

```
#ifndef __USER_TASKS_H
#define __USER_TASKS_H

#ifdef __cplusplus
extern "C" {
#endif

////////////////////////////////////////////////
#include "main.h"

void USR_Task_RunLoop(void);

////////////////////////////////////////////////
```

```
#ifdef __cplusplus
}
#endif

#endif /* __USER_TASKS_H */
```

Press the **Build** button. You should be able to build the project without error. You should be able to download the code to your board and run the code. The functionality should be the same as that of Workshop 3.

Ask questions if you have any issues about the above contents.

### Task 3: Adding additional functionalities to the project

**Task 3.1. Change the coding style of the main user code and define needed static functions** (20 points)

The coding style of the user code of Workshop 3 is something we see very often. To make the code more readable, we can change the coding style to the following.

```
// Public functions
void USR_Task_RunLoop(void) {
    if (JOY_L_Is_Pressed()) {
        LD_R_On();
        LD_G_Off();
    } else if (JOY_R_Is_Pressed()) {
        LD_G_On();
        LD_R_Off();
    } else {
        LD_R_Toggle();
        HAL_Delay(100);
        LD_G_Toggle();
        HAL_Delay(100);
    }
}
```

We can see that the above code is easily readable and the intended functionalities of the code are very clear.

To use the above code style, we have to define the functions used in the code at the beginning of the file (before `USR_Task_RunLoop()`). These functions can be defined as public functions that can be called by functions in other files. But for this project, we intend to hide them from the other files. Hence, we need to define them as **static** functions, as shown below.

```
// Prototype of static (private) functions
static bool JOY_L_Is_Pressed(void);
static bool JOY_R_Is_Pressed(void);
```

```
static void LD_R_Toggle(void);
static void LD_G_Toggle(void);
```

Please define the prototypes of the other needed static functions in `USR_Task_RunLoop()` accordingly. To use the `bool` type, you need to include `stdbool.h` after `#include "usr_tasks.h"`.

Now, build code and you will found that the code does not build as the above functions are not defined yet.

**Task 3.2. Define the above static functions** (20 points)

Now, we need to define each of the above static functions. For example, `JOY_L_Is_Pressed(void)` can be defined as

```
static bool JOY_L_Is_Pressed(void) {
    return HAL_GPIO_ReadPin(JOY_L_GPIO_Port, JOY_L_Pin);
}
```

Similarly, please define the following static functions using HAL functions.

```
static bool JOY_R_Is_Pressed(void);
static void LD_R_Toggle(void);
static void LD_G_Toggle(void);
```

Note that you need to put these functions at the bottom of the file for readability reasons.

We need to similarly define the LED on/off functions. Recall from Workshop 3 that we use different values in the GPIO output functions to turn on/off the LEDs in the two boards. Here we use a slightly better approach for the definition. Put the following between the definitions of the public and static functions.

```
// Definitions used in static functions
#if defined(STM32L476xx)
    #define LED_ON GPIO_PIN_SET
    #define LED_OFF GPIO_PIN_RESET
#elif defined(STM32F412Zx)
    #define LED_ON GPIO_PIN_RESET
    #define LED_OFF GPIO_PIN_SET
#endif
```

We can see that we can use the following function to turn on the Red LED for any board.

```
static void LD_R_On(void) {
    HAL_GPIO_WritePin(LD_R_GPIO_Port, LD_R_Pin, LED_ON);
}
```

Define the other three LED on/off functions in the same way.

Now, build the project. In the ideal case, you should be able to build without any issue. If you have, there may be something wrong with your own definitions. Debug these issues or ask the instructor/TA.

If there is no issue with your build, you should be able to download the code to your board. See if you have all the required functionalities programmed.

**Task 3.3. Define your own CMSIS functions to replace the HAL functions for the GPIO operations** (30 points)

So far, you have been using HAL functions to operate GPIOs. Now, you need the logic operations we learned in the class to operate on the registers directly so that you can replace the HAL functions with your own implementation on registers. Below is an example of writing a GPIO pin.

```
static void CMS_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin,
                GPIO_PinState PinState) {
    if(PinState != GPIO_PIN_RESET) {
        GPIOx->ODR |= (uint32_t)GPIO_Pin;
    } else {
        GPIOx->ODR &= ~((uint32_t)GPIO_Pin);
    }
}
```

Note that we have used exactly the same parameters as for the `HAL_GPIO_WritePin(...)` function. This is really helpful when you want to swap to a function with a different implementation. Note also that we have used CMS to replace HAL to make it clear that we are using the CMSIS version.

Write the CMS version of the other two HAL functions you have called in your code.

Build the project again with your CMS version functions. You should be able to have the same functionalities as the HAL version.

**Submission of your work**

(10 points)

You are suggested to work with another student so that you can discuss ideas. However, you should submit your own work in a pdf file including the following:

- A screenshot for each of Tasks 1 to 3 showing your work as you see fit.
- Pictures of the working of the project—the working of the LEDs when the left key and right key are pressed separately.