#cec32xNotes, cec32x_ws_18_WS03_getting_started_with_HW_programming.md

WS 3. Getting Started with Hardware Programming

Introduction

In this project, we practice basic programming for the Discovery board(s) based on the code generated by STM32CubeMX. This lab needs a HW kit, which will be borrowed from the lab TAs during the next lab meeting.

In a later workshop/lab, we will learn how to set up STM32CubeMX to generate the code. (Please have STM32CubeMX installed ASAP.)

Workshop tasks

Task 1. Creating the project folder

(10 points)

As shown in cec32x_devtool_20_organization_of_prjt_files.pdf (in the **Dev Tools** section of the **Modules** on the Canvas of CEC 320), we will use a standard file structure to organize the files of projects. This will help to facilitate the porting of the code between the team members and the submission of the work. Please download the `_lib.zip` file under the above file and unzip its contents to the `C:\prjt_arm\_lib` folder. Read the above file carefully to understand the folder structure.

Task 2 Building the provided template code

(10 points)

Download `expl_014_template_for_HW_prjt.zip` in the **Dev Tools** section of the **Modules** on the Canvas of CEC 320. Unzip the file to the `ws03_getting_started_with_HW_programming` folder, which is called the project folder here. Make sure you have the following three folders under this project folder.

- `F412Discovery`
- `L476Discovery`
- `usr_src` (for user source code)

Open this directory using the File Explorer and navigate to the ''MDK-ARM'' folder of your specific board. Double-click the main project management file to open the project in Keil MDK-ARM. Press the **Rebuild** button. You should be able to build the project without error. If you cannot build the project without error, please inform the instructor or the TA.

Task 3. Running the provided template code

(10 points)

Run the built program in the HW kit following the steps in cec32x_devtool_24_running_debugging_a_program_in_Keil_for_HW.pdf. You should be able to see a user LED flashes at about a rate of 5 Hz. If you press the left Joystick key, the LED will be off. If you release the Joystick key, it will flash again.

Task 4. Creating new user functions for user task programming

(50 points total)

In the `usr_src` folder, there two files used to hold the user functions:

- `user_tasks.c` is the source file for C functions.
- `user_tasks.h` is the header file for prototypes of the functions defined in `user_tasks.c`.

In fact, only the prototypes of the global (public) functions need to go to the header file. The static (private) functions can be defined at the beginning in the C source file in a sequential manner to save the definition of the prototypes.

Now, we need to do the following programming.

Task 4a. Creating a global function to flash LD_R three time

(20 points)

Read the provided code carefully and see how the toggle of the LED is implemented. Then, use the same approach to write a global function with the following definition: `void USR_Task_Flash_LD_R_3Times(void)`. Below are requirements and notes for the function:

- This function is defined in the `usr_tasks.c` file, and the prototype should appear in the `usr_tasks.h` file so that it can be called in the `main.c` file, which is generated by STM32CubeMX.
- This function should be called after the initialization of the GPIO, during which LD_R is turned off.
- The LED should **flash** for three time—it will be on for three times, each needs two toggles to go back to the initial state. This means that we need to run the toggle function of the LED for 3 x 2 times.
- You can use the same time of delay between the on/off time as the provided code.
- These is no need to distinguish between the two different HW kits. They work the same way when doing the toggle.

Here are the details for the function definition.

- Use a `for` loop to finish the toggles in the function. You can use `for (int i = 0; i < 3 * xx; i++) {}`, where `xx` is a value you should figure out.
- There is no return so that you can end the function using the closing brace `}`.

This function should be called between `USER CODE BEGIN 2` and `USER CODE BEGIN 2` in `main.c` so that it will be executed only once before the program goes to the infinite `while` loop.

Task 4b. Creating a static function to read the Joystick left key

(10 points)

Read the provided code carefully and see how the reading of the Joystick left key is implemented. Then, use the same approach to write a static function with the following definition: `static bool JOY_L_key_is_pressed(void)`. Below are requirements and notes for the function:

- This is a static function defined in the `usr_tasks.c` file, and the prototype should not appear in the `usr_tasks.h` file.
- This function should be placed at the beginning of the source file so that it can be called by a function defined later in the file.
- The function returns a Boolean value. To use the data type `bool`, you need to include the header file, `stdbool.h` at the beginning of the source file. You can place `#include <stdbool.h>` after `#include "usr_tasks.h"`.
- The `HAL_GPIO_ReadPin` function returns a Boolean variable. You should take advantage of this to return the result of your function.

Task 4c. Creating static functions to turn on and off LD_R

(10 points)

In a similar way as flashing LD_R using a clean and easily understandable standalone function, we can program two functions to turn on and off LD_R. Note that these functions are called in a function defined in the same source file so that we define them as static.

Read the provided code carefully and see how LD_S is turned on and off. Then, use the same approach to write two static functions with the following definitions: `static void Turn_on_LD_R(void)` and `static void Turn_off_LD_R(void)`. Below are requirements and notes for these functions:

- These are static function defined in the `usr_tasks.c` file, and the prototypes should not appear in the `usr_tasks.h` file.
- These functions should be placed at the beginning of the source file so that it can be called by a function defined later in the file. Put them after the definition of the above `static bool JOY_L_key_is_pressed(void)` function.

- These functions should be written in such a manner that it can be used for both the two different HW kits. So, you need to use the C conditional compilation directives `#if`, `#elif`, and `#endif`, as shown in the provided code.

Task 4d. Calling the above static functions

(10 points)

Read the provided code of the `void USR_Task_RunLoop()` function, main user loop task function, and how it is called in the `main.c` file. Note that this loop task function is called in a `while` loop repeatedly to implement various needed user functionalities.

Now, we need to change the code of the user loop task function to implement the following functionalities by calling the above three static functions. **If the Joystick left key is pressed, turn on LD_R for 200 millisecond (by using the delay function); otherwise, turn off LD_R.** Note that you should to have as little other code as possible to improve the speed of the program.

Task 5. Finding out the function prototypes provided in HAL for GPIO operations

(10 points)

List all the prototypes of the **three** HAL functions we have used to read, write, and toggle GPIO pins. These prototypes can be found in the ''stm32y4xx_hal_gpio.h'' file, which can be opened in many ways. One way is to click the + sign before the ''main.c'' function and find the heeded header file in the list.

Submission of your work

(10 points)

Each student needs to submit their own results in two files. The first file is the zipped project. The second file is the pdf file titled as `cec320_ws3_lastname_firstname(or initial).pdf`, which includes the following:

- (Screenshot of) Source code of the entire `usr_tasks.c` file.
- Screenshot of the code snippet in `main.c` regarding the added code between `USER CODE BEGIN 2` and `USER CODE BEGIN 2`.
- Screenshot of the code snippet for the header file where you added the new global function.