Lab 5. State Machine, Macro, and Assembly

1. Tasks of the lab

We have practiced in Lab 3 about how to use CubeMX to set up GPIO pins to read Joystick keys (JOY_C, JOY_L, and JOY_R) and control user LEDs (LD_R and LD_G). We have also practiced a state machine in Workshop 2, where the state of the machine is controlled by a variable saved in the memory, the value of which can be changed when the program is halted. In this lab, we combine the ideas of the above two projects to program a new project using a state machine to control LEDs based on the inputs from Joystick keys. Specifically, during each state, the user LEDs will blink in a certain pattern and the state can transition to a new state according to the inputs from the Joystick keys. This is Task 2 (50 points). We have learned before that we can use a function to perform an operation such as turning on or off an LED or reading a key input. To make the program more readable, we can use macros to give meaningful names to the operations. This is Task 1 (20 points). We also use simple assembly functions to perform GPIO operation—to turn on/off LEDs and read the Joystick keys. This is Task 3 (30 points).

Note that you need to use the `.ioc` file from Lab 3 to generate the base source code for this lab. In the generated `main.c` file, put the following code snippets in the appropriate spaces following the provided comments as a preparation. They are labeled as Code snippet 1 and 2 in the enclosed code file.

```
// Code snippet 1
/* USER CODE BEGIN Includes */
#include <stdbool.h>

/* USER CODE END Includes */

// Code snippet 2
/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/
#ifdef __GNUC__
/* With GCC, small printf (option LD Linker->Libraries->Small printf
   set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

extern uint32_t read_a_bit_of_a_port(__IO uint32_t *pIDR, uint32_t pinMask);
extern void set_a_bit_of_a_port(__IO uint32_t *pODR, uint32_t pinMask);
extern void clear_a_bit_of_a_port(__IO uint32_t *pODR, uint32_t pinMask);

#define LD_R_ON   // Add the appropriate function here
```

```
#define LD_G_ON   // Add the appropriate function here
#define LD_R_OFF  // Add the appropriate function here
#define LD_G_OFF  // Add the appropriate function here

#define JOY_C_IS_PRESSED  // Add the appropriate function here
#define JOY_L_IS_PRESSED  // Add the appropriate function here
#define JOY_R_IS_PRESSED  // Add the appropriate function here

/* USER CODE END PFP */
```

Note that the specific macro definitions are missing in the above code snippet. You need to define them as Task 1 of the lab.

Task 1. Macro definitions

Programming is an iterative process. You write some code to implement certain functionality, then debug those code. When the code is correct, you add more code for additional functionalities, then debug more. This is true for this lab as well. In the beginning, you may want to define the macros using HAL functions to control LEDs and Joystick keys. With these definitions, you can go ahead to program Task 2. When Task 2 is done, you can then proceed to perform Task 3. After that, you can come back to Task 1 to change the macro definitions to the assembly functions you programmed in Task 3.

Task 2. A simple state machine with C

For this lab, there are four states, with the functionalities and transitions defined below:

- State 1. In this state, the two user LEDs blinks alternately (one on and the other off) every `bDelay` ms (defined as 100 in the provided code snippet below), and the program can only transition to State 2 when reading a pressing of the center key (JOY_C).
- State 2. In this state, the two user LEDs are all off, and there are three different choices for transitions now:
    1. go to State 1 if the center key is pressed,
    2. go to State 3 if the left key (JOY_L) is pressed, and
    3. go to State 4 if the right key (JOY_R) is pressed.
- State 3. In this state, only the red LED (LD_R) blinks every `2*bDelay` ms, and the program can transition to two States:
    1. go to State 2 if the center key is pressed and
    2. go to State 4 if the right key is pressed.
- State 4. In this state, only the green LED (LD_G) blinks every `3*bDelay` ms, and the program can also transition to two States:
    1. go to State 2 if the center key is pressed and
    2. go to State 3 if the left key is pressed.

You need to use the following code snippet (Code snippet 3 in the enclosed code file) so that we all use the same variables in each team's code. Put them to the space between `/* USER CODE BEGIN 1 */` and `/* USER CODE END 1 */`.

```
enum progState{State1 = 1, State2, State3, State4};
enum progState currentState = State1;
bool stateChanged = true;
bool ledON = true;
uint32_t bDelay = 100;
```

You need to figure out the meaning of the variables defined above from the context of this lab manual and the provided code.

Now, paste the following code (Code snippet 4 in the enclosed code file) to the space between `/* USER CODE BEGIN 3 */` and `/* USER CODE END 3 */`.

```
if (stateChanged)
    printf("The current state is State%1d.\n\r", currentState);

switch (currentState) {
  case State1:
    if (stateChanged) {
        printf("Press the center key to go to State 2.\n\r");
        stateChanged = false;
    }

    if (JOY_C_IS_PRESSED) {
        currentState = State2;
        stateChanged = true;
        break;
    }

    if (ledON) {
        LD_R_ON;
        LD_G_OFF;
    } else {
        LD_R_OFF;
        LD_G_ON;
    }
    HAL_Delay(bDelay);
    ledON = !ledON;

    break;
  case State2:
    if (stateChanged) {
        // provide your hint to the user and set your "state changed" flag
    }

    if (JOY_C_IS_PRESSED) {
        // provide your code to handle the center Joy key input.
    }
    if (JOY_L_IS_PRESSED) {
        // provide your code to handle the left Joy key input.
    }
    if (JOY_R_IS_PRESSED) {
        // provide your code to handle the right Joy key input.
    }

    LD_R_OFF;
    LD_G_OFF;
```

```
      break;
    case State3:
      if (stateChanged) {
          // your code
      }

      // your state transition code

      if (ledON) {
          LD_R_ON;
      } else {
          LD_R_OFF;
      }

      LD_G_OFF;
      HAL_Delay(2*bDelay);
      ledON = !ledON;

      break;
    case State4:
      if (stateChanged) {
          // your code
      }

      // your state transition code

      // your LED code

      break;
    default:
      printf("The program should not print this line---something is deadly wrong\n\r");
      break;
  }
```

Note that many C statements are missing in the above code snippet. You are supposed to code them to make the state machine work as described.

Note also that there are many macros used in the above code snippet. You should define them using HAL functions to get you started with Task 2. Later, you can replace these HAL functions by assembly functions in Task 3 without changing anything in Task 2.

Task 3. GPIO control with assembly

If you followed the suggestions provided earlier regarding coding and debugging, you have programmed Task 2 using the HAL functions now with the help of Task 1. Now, you need to convert the HAL functions for turning on/off the LEDs as well as the reading of Joystick keys to assembly functions. Follow the examples provided on pages 6 and 7 of notes of Lecture 13 to write your assembly code. For examples of mixed C and assembly programming, see the .c and .s files in demo projects 110_arm_isa and 120_logic_instructions.

2. Lab report

This lab report is special—the results will be used to assess an ABET program outcome. The grading will be following the rubrics in the enclosed pdf file. Please read it carefully before doing your programming.

You need to add appropriate comments to all *user* code (the code not generated by CubeMX). Copy all these user codes together with your comments to your report. Please be reminded that you need to perform your programming as independently from the TA as possible as seeking too much help will lead to unsatisfactory of some Attributes. Note that:

- Task 1 is assessed using Attribute a.
- Task 2 and the structure of the entire code are assessed using Attribute b.
- Task 3 is assessed using Attributes c and d.