Embedded System Lab5 Report

Jiabei Han netid: jh4873

October 2018

1) Introduction

The main objective of this lab is to utilize STM32F746G along with a Diligent PmodKYPD to design a digital lock that recognizes a certain combination of numbers.

2) Theoretical Background

PmodKypd:

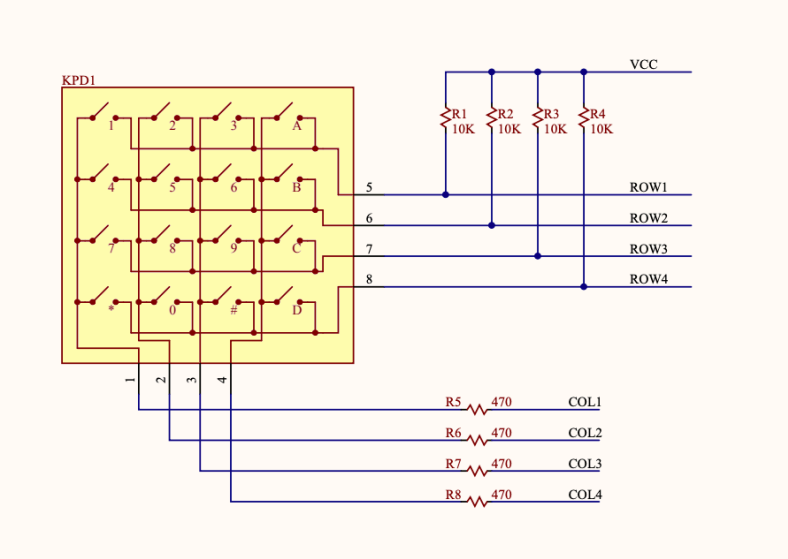

Figure 1. Inner Structure of PmodKYPD

- **Figure 1** briefly shows the inner structure of the keypad ('*' and '#' and '0' should be replaced by '0' and 'E' and 'F'). At default, the readings of both ROWs and COLs are 1(high voltage). When COLs are set to 0, if a key is pressed, the specific row that the key is located at will have the reading 0. Using this basic theory, in order to locate the keys that are pressed, the following procedures need to be performed.
  - COL1, COL2, COL3, COL4 are set to 0 individually in terms.
  - When COL(i) is set to 0, check the reading of ROW1, ROW2, ROW3, ROW4.
  - A reading 0 among these four rows means a button which is located at ROW(j) and COL(i) has been pressed. This way the keypad can detect the button that has been pressed in a very short amount of time.
- Now consider the situation that multiple buttons have been pressed. If two buttons on the same row are pressed simultaneously, the keypad could not detect correspondence readings because reading of ROWi (where 2 keys are pressed) is VCC/2, however ROWi can only recognize VCC or 0.

Connection:

- In order to control the keypad, we need to connect the keypad with STM32F746G. **Figure 2** shows arrangement of all 12 pins on the PmodKYPD. Here, pin11,12 are redundant since only one ground and one VCC is required.

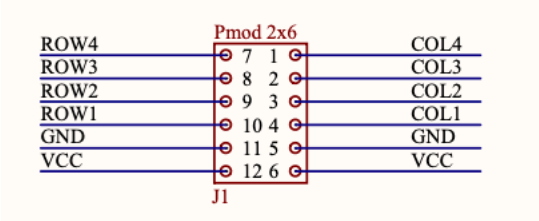

Figure 2. PmodKYPD pins arrangement

- Pin 1 to 10 from the PmodKYPD needs to be connected with the Arduino UNO connector on the STM32F746G board. **Figure 3** shows the schematics of connector. For this lab, only PF6, PF7, PF8, PF9 from GPIOF and PI0, PI1, PI2, PI3 from GPIOI are required. As for this lab, PI0 to PI3 (PI0 is at PA8's position on this schematics) are each connected with pin1 to pin4 as the COLs and PF6 to PF9 are each connected with pin7 to pin10 as the ROWs. VCC is connected with 3.3V on the board.

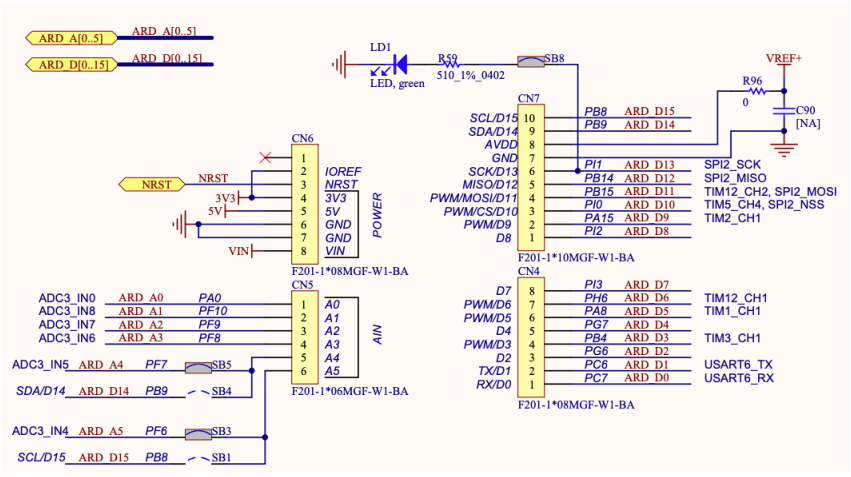

Figure 3. Schematics of Arduino UNO connector

3) Experimental Result

Keypad reading:

- In order to design a digital lock, firstly the reading of the buttons that are pressed on the keypad needs to be transferred first. As mentioned in part 2 of the report, a simple state machine is introduced. **Figure 4** shows state machine diagram.

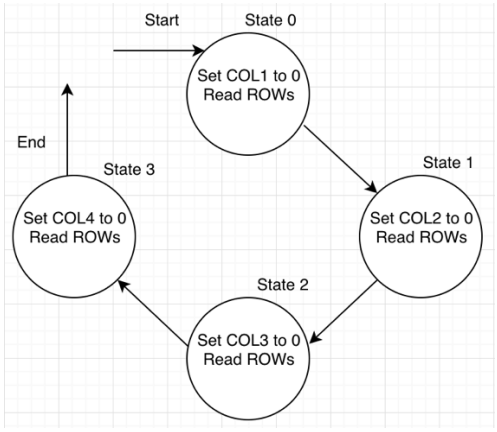

Figure 4. State machine for keypad reading

- The second step is to write a specific function in C that performs the above state machine. Here the

return value OUTPUT is defined as a signed character. The function returns -2 when multiple keys are detected, it returns -1 when no keys are detected. Otherwise it will return the exact value of the key that is pressed, which is in the range from 0 to 15. The following lines shows the C code.

```
signed char read_kypd(void){
    signed char OUTPUT;
    int INPUT,i,j;
    int tab[16] = {1,4,7,0,2,5,8,15,3,6,9,14,10,11,12,13};// reference table
    OUTPUT = -1;                                    // default OUTPUT value
    GPIOI -> BSRR = 0x0F;                           // set COLs to 1
    for(i = 0; i < 4;i++){
        GPIOI -> BSRR = (1<<(16+i));                //set COLi to 0
        INPUT=((GPIOF->IDR&0x3C0)>>6);              //read ROWs from GPIOF
        if (INPUT != 0x0F && OUTPUT != -1){         //multiple keys pressed
                return -2;
            }
        for(j = 0; j < 4; j++){
            if(((INPUT>>j)&0x01)==0){               //a 0 is detected on ROWj
                OUTPUT = tab[i*4 +j];
            }
        }
        GPIOI -> BSRR = (1<<i);                     //set COLi back to 1
    }
    return OUTPUT;
}
```

- For this part of the program, the main problem that was encountered is how to detect the situation when multiple keys are pressed. Here the logic is, when COLi is set to 0, if OUTPUT doesn't equal to -1(one key has been pressed in previous iterations) and the current input doesn't equal to 0x0F(one key in COLi has been pressed), the function should directly return -2.

Digital Lock:

- Before writing C code for the digital lock, a state machine that can recognize a certain combination of digits needs to be finished first. The requirement of the algorithm is that, suppose there are n digits for the password, as long as the last n digits that were pressed are correct, the system will be unlocked. For example, consider the password is {5,3,9,2}, if {5,3,5} has been pressed, the state machine should recognize the third digit '5' and the state machine should stay in the state 1 instead of going back to state 0. **Figure 5** shows the state machine diagram. Different passwords may require different algorithms, password such as {5,3,5,2} that contains one digit more than one time will require more complicated algorithm. There is a general algorithm for this problem, but it will not be discussed in this report. The following state machine diagram and C code only applies with password such as {5,3,9,2} that each digit is unique in the password.

NOTE: OUTPUT in the diagram represents the reading from the keypad.

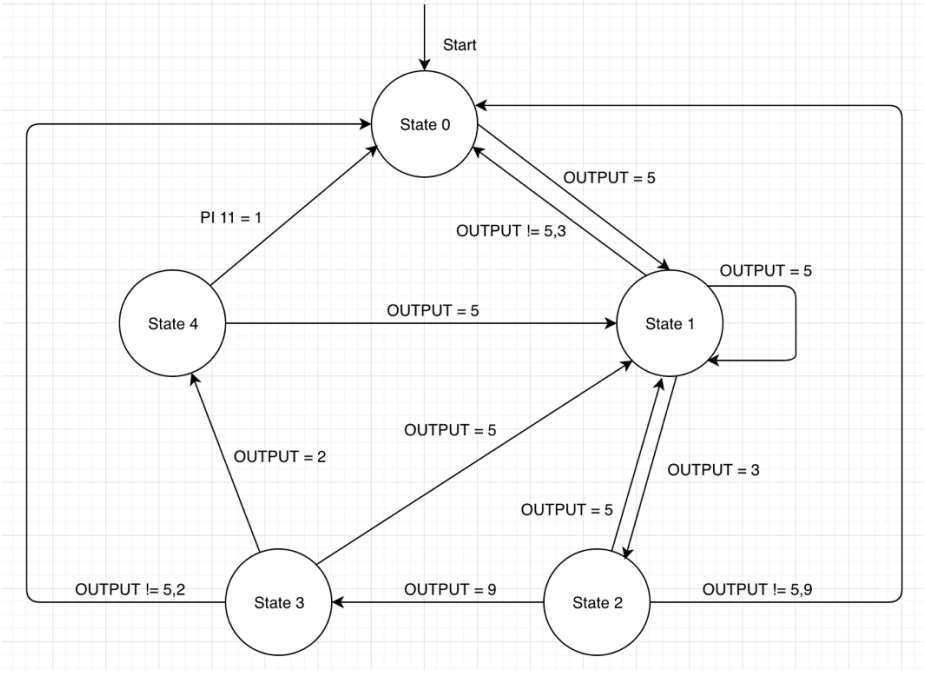

Figure 5. State Machine Diagram for Digital Lock

- Brief explanation for the diagram:
  - State 0: Initial state, no key has been pressed or the password is incorrect.
  - State 1: Last 1 digit of the entire sequence of numbers is correct.
  - State 2: Last 2 digits of the entire sequence of numbers are correct.
  - State 3: Last 3 digits of the entire sequence of numbers are correct.
  - State 4: System unlocked, if reset button (PI11) is pressed, return back to initial state.
- The second step is to write C code in terms of this state diagram. Similar to other state machine in C, there 5 different states in total, each represents a case in switch. However, when actually writing C code, there are several problems that needs to be avoided.
  - In the main function, a while (1) loop contains the entire state machine. When a button is pressed, the main function is constantly reading the keypad. This may cause a strange move. When the system is at state i, the correct number to go to state i+1 is pressed. However, before the button is released, the system immediately travels to state i+1 but the system constantly read the button that hasn't been released. Eventually the system will be in chaos. This problem couldn't be detected in debug mode. The solution for this problem is to introduce a variable prev_key to record the output of keypad during the last iteration. If the output of the keypad equals the value of prev_key, then the system shouldn't enter the state machine during this iteration. The same logic applies with the reset button.
  - When the user releases a button and before the user press the next button, there will be a time period that no keys are pressed. However, the system will constantly read outputs, which is -1. As shown in **Figure 5** the state machine doesn't contain any states that responds to OUTPUT = -1. In order to proceed, the system will not enter the state 0 to state 3 when the reading from the keypad equals -1. The following lines shows the C Code for the state machine.

```c
#include "stm32f7xx_hal.h"              // Keil::Device:STM32Cube HAL:Common
#include "GLCD_Config.h"                // Keil.STM32F746G-Discovery::Board Support:Graphic LCD
#include "Board_GLCD.h"                 // ::Board Support:Graphic LCD
#include <math.h>

static void SystemClock_Config(void);
extern GLCD_FONT GLCD_Font_16x24;

signed char read_kypd(void){

    signed char OUTPUT;
    int INPUT,i,j;
    int tab[16] = {1,4,7,0,2,5,8,15,3,6,9,14,10,11,12,13};// reference table

    OUTPUT = -1;                                // default OUTPUT value
    GPIOI -> BSRR = 0x0F;                       // set COLs to 1
    for(i = 0; i < 4;i++){
        GPIOI -> BSRR = (1<<(16+i));            //set COLi to 0
        INPUT=((GPIOF->IDR&0x3C0)>>6);          //read ROWs from GPIOF

        if (INPUT != 0x0F && OUTPUT != -1){     //multiple keys pressed
                return -2;
            }
        for(j = 0; j < 4; j++){
            if(((INPUT>>j)&0x01)==0){           //a 0 is detected on ROWj
                OUTPUT = tab[i*4 +j];
            }
        }
        GPIOI -> BSRR = (1<<i);                 //set COLi back to 0

    }
    return OUTPUT;

}

int main(void)
{
    int key,prev_key,state,reset,prev_reset;
    int password[4] = {5,3,9,2};    //password
    SystemClock_Config();

    RCC->AHB1ENR |= (1<<5)|(1<<8);  //initialize GPIOI,GPIOF
```

```
    GPIOI->MODER |= (1<<0)|(1<<2)|(1<<4)|(1<<6);//set PI0-PI3 to OUTPUT mode

    GLCD_Initialize();              //initialize GLCD
    GLCD_SetFont(&GLCD_Font_16x24); //set font
    GLCD_SetForegroundColor(GLCD_COLOR_BLACK); //set foreground color
    GLCD_SetBackgroundColor(GLCD_COLOR_BLUE);  //set background color
    GLCD_ClearScreen();
    state,prev_key,reset = 0;
    key = -1;
    prev_reset = 1;

    while (1){
        key = read_kypd();          //read OUTPUT from keypad
        reset = GPIOI->IDR&(1<<11); //read reset from PI11
        if ((key != prev_key)||(reset != prev_reset)){ //check whether there is new reading
            GLCD_ClearScreen();
            prev_key = key;         //update prev_key
            prev_reset = reset;     //update prev_reset
            if ((key != -1)){       //check whether a key is pressed
                //state machine
                switch(state){
                    case 0:{
                        if(key == password[state]){
                            state += 1;
                        }else{
                            state = 0;
                        }
                        break;
                    }
                    case 1:{
                        if(key == password[state]){
                            state += 1;
                        }else if(key == password[state-1]){
                            state = 1;
                        }else{
                            state = 0;
                        }
                        break;
                    }
                    case 2:{
                        if(key == password[state]){
                            state += 1;
```

```
84.                         }else if(key == password[state-2]){
85.                             state = 1;
86.                         }else{
87.                             state = 0;
88.                         }
89.                         break;
90.                     }
91.                     case 3:{
92.                         if(key == password[state]){
93.                             state += 1;
94.                         }else if(key == password[state-3]){
95.                             state = 1;
96.                         }else{
97.                             state = 0;
98.                         }
99.                         break;
100.                     }
101.                 }//end switch
102.             //system is unlocked
103.             }else if(state == 4){
104.                 GLCD_DrawString(GLCD_SIZE_X / 2 - 50, GLCD_SIZE_Y/2, "Syste
    m Unlocked");
105.                 //if reset button is pressed, reset the system
106.                 if (reset == 1){
107.                     state = 0;
108.                 }
109.             }
110.             else{
111.                 continue;
112.             }
113.         }else{
114.             continue;
115.         }
116.     } //end while
117.     return 0;
118. } //end main
119.
120. void SysTick_Handler (void)
121. {
122.     HAL_IncTick();
123. }
124.
125. static void SystemClock_Config(void)
126. {
```

```
127.   RCC_ClkInitTypeDef RCC_ClkInitStruct;
128.   RCC_OscInitTypeDef RCC_OscInitStruct;
129.   HAL_StatusTypeDef ret = HAL_OK;
130.
131.   /* Enable HSE Oscillator and activate PLL with HSE as source */
132.   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
133.   RCC_OscInitStruct.HSEState = RCC_HSE_ON;
134.   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
135.   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
136.   RCC_OscInitStruct.PLL.PLLM = 25;
137.   RCC_OscInitStruct.PLL.PLLN = 432;
138.   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
139.   RCC_OscInitStruct.PLL.PLLQ = 9;
140.
141.   ret = HAL_RCC_OscConfig(&RCC_OscInitStruct);
142.   if(ret != HAL_OK)
143.   {
144.     while(1) { ; }
145.   }
146.
147.   /* Activate the OverDrive to reach the 216 MHz Frequency */
148.   ret = HAL_PWREx_EnableOverDrive();
149.   if(ret != HAL_OK)
150.   {
151.     while(1) { ; }
152.   }
153.
154.   /* Select PLL as system clock source and configure the HCLK, PCLK1 and PC
    LK2 clocks dividers */
155.   RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK
    | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
156.   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
157.   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
158.   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
159.   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
160.
161.   ret = HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7);
162.   if(ret != HAL_OK)
163.   {
164.     while(1) { ; }
165.   }
166. }
```

4) Conclusion

This lab can be categorized into three different parts. The first part is to connect PmodKYPD with STM32F746G in a specific order. The second part is to design a simple state machine that read the output from the keypad. Because of the design of the keypad, each column needs to be read individually. The main problem for this part is to avoid the situation that multiple keys are pressed. The last part is to design a digital lock that can recognize a certain combination of digits. The main challenge for this part is design a state machine that can recognize the last n digits that you pressed. The other challenge while programing is how to present a state machine inside an infinite while loop that constantly reads output from the keypad. Generally speaking, this lab is quite successful, and all the results reached the original expectation.