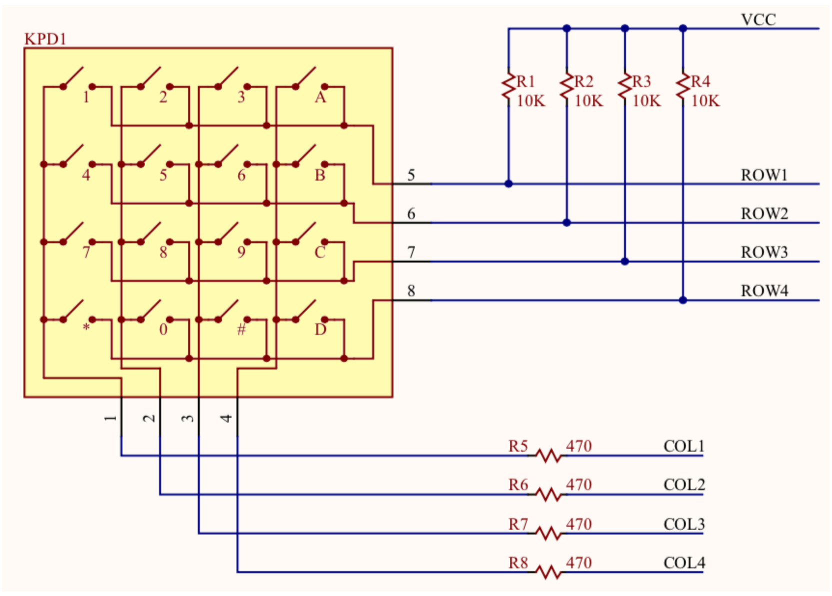
Keypad lab report

Sequential Systems Lab

The end goal of this lab is to make a keypad lock to secure something like a safe. We will simulate the locking and unlocking by displaying on the LCD screen. First we have to wire the keypad (Digilent PmodKYPD) to our board (STM32F746NGHx) using consecutive pins as an easier design choice. We will wire the keypad COL1~4 pins to the board GPIOI0~3 and the keypad COL1~4 to the board GPIOF6~9 pins.

Keypress Program

Next we should read the keypad, and be sure that we are pressing the correct numbers and getting the correct outputs. To make sure we are pressing and reading correct buttons, we make a small program to print out the keys we pressed. The keypad can only read 4 out of the 16 outputs at a time. Whenever we unpower 1 of the GPIOI column pins, we can read the corresponding column from the row GPIOF pins.

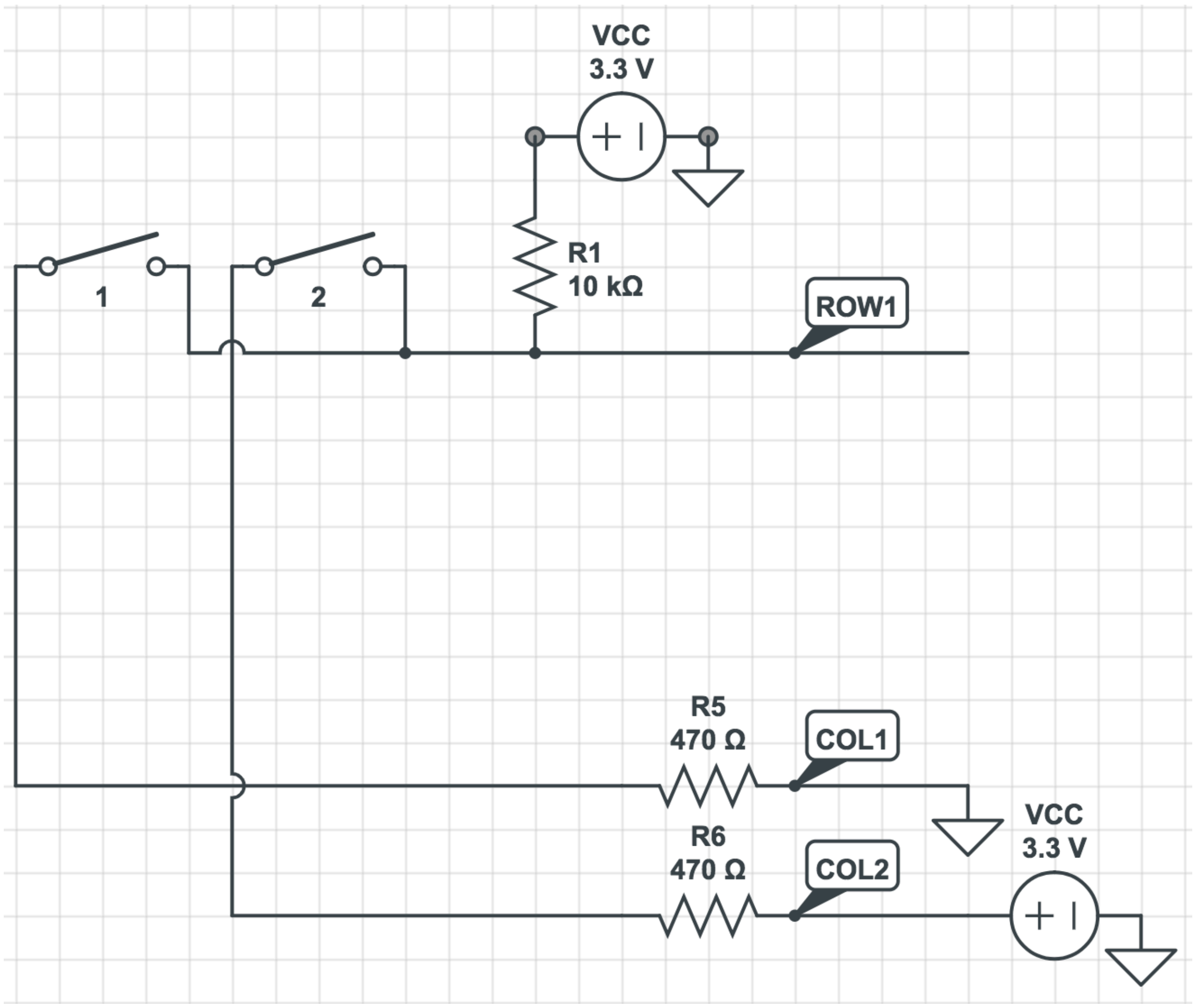


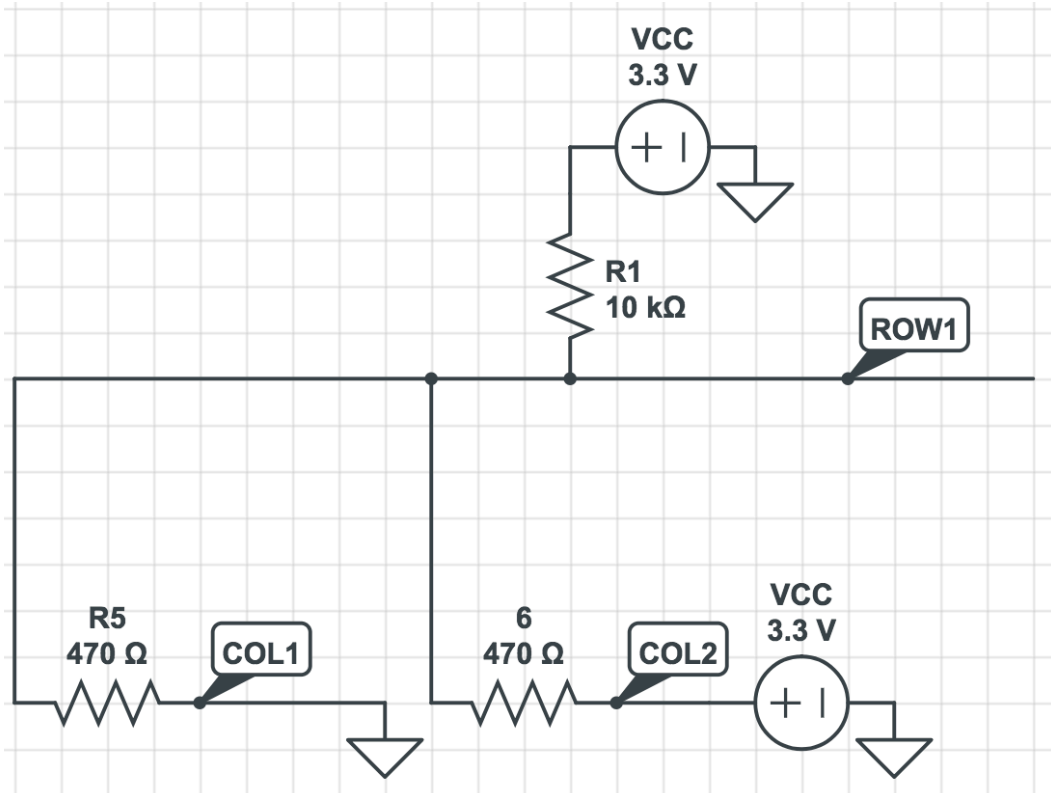
Our function would loop over the columns, and get the output of the rows. After a single iliteration the columns, the function can return our value in the form of a signed char. Negative values `-1` indicates 'no key' is pressed, `-2` indicates 'more than one key is pressed', and the function returns the `char` of whatever key we press normally.

```c
// my_kypd_utils.c
#include "stm32f7xx.h"                  // Device header
#include "my_kypd_utils.h"

schar hexaKeys[4][4] = {
 {'1','2','3','A'},
 {'4','5','6','B'},
 {'7','8','9','C'},
 {'0','F','E','D'}
};

```

```c
schar read_kypd() {
    char dupe = 0;
    schar out = -1;

    GPIOI->BSRR = 0xf;

    // Read every column
    for (char i = 0; i < 4; i++) {
        GPIOI->BSRR |= 1 << (0x10 + i);

        // loop over rows
        for (char j = 6; j < 10; j++) {
            // read input of key of row and column
            if (~GPIOF->IDR & 1 << j) {
                if (dupe)
                    return -2;
                dupe = 1;
                out = hexaKeys[j-6][i];
            }
        }
        GPIOI->BSRR |= 1 << i;
    }

    return out;
}
```

Notice that whenever we press two keys in the same row, we get an output of all 1's from our rows, meaning our function returns `-1` and can't detect any keys on the same row and columns of the two keys pressed. Here are a few simplified images of what is happening.





Notice when keys 1 and 2 are pressed, the power from `COL2` feeds back into our `ROW1` GPIO register, effectively making a voltage divider. Luckily the math checks out, and the voltage is above the threshold where our chip considers it a 1.

Our main program to test out the key functionality is shown below.

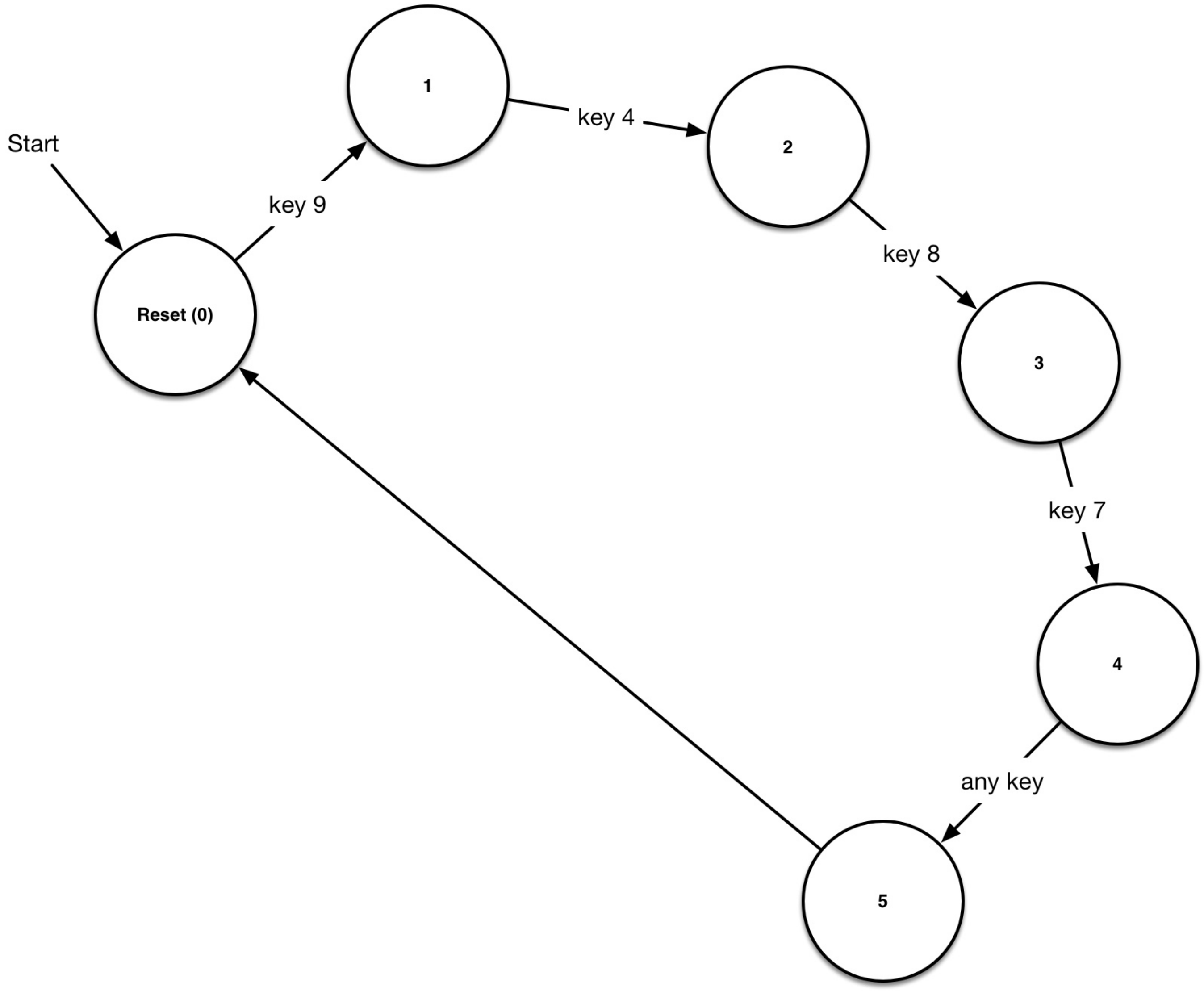
```c
#include "stm32f7xx.h"                  // Device header
#include "Board_GLCD.h"                 // ::Board Support:Graphic LCD
#include "GLCD_Config.h"                // Keil.STM32F746G-Discovery::Board Support:Graphic LCD
#include "my_utils.h"
#include "my_kypd_utils.h"

extern GLCD_FONT GLCD_Font_16x24;

int main() {
    schar c[20] = "                   ";
    schar prev = ' ';

    SystemClock_Config();
    GLCD_Initialize();

    // Initialize GPIOs
    RCC->AHB1ENR |= 1 << F | 1 << I;
    for (int i = 0; i < 4; i++)
        set_register_mode(I, i, 1);

    // Initialize screen LCD
    GLCD_SetBackgroundColor(GLCD_COLOR_BLACK);
    GLCD_SetForegroundColor(GLCD_COLOR_WHITE);
    GLCD_SetFont(&GLCD_Font_16x24);

    GLCD_ClearScreen();

    GLCD_DrawString(50, 50, "You've Pressed:");

    while(1) {
        c[0] = read_kypd();
        if (c[0] != prev) {
            prev = c[0];
            switch (c[0]) {
                case -1:
                    GLCD_DrawString(50, 75, "No keys.           ");
                    break;
                case -2:
                    GLCD_DrawString(50, 75, "More than one key. ");
                    break;
                default:
                    GLCD_DrawString(50, 75, (const char *) c);
            }
        }
    }
}
```

This program outputs directly our key presses to the screen using the built in font, and the program catches the cases for -1 and -2 and displays words instead of trying to print an ascii font that might not exist.

Designing a Lock Proram

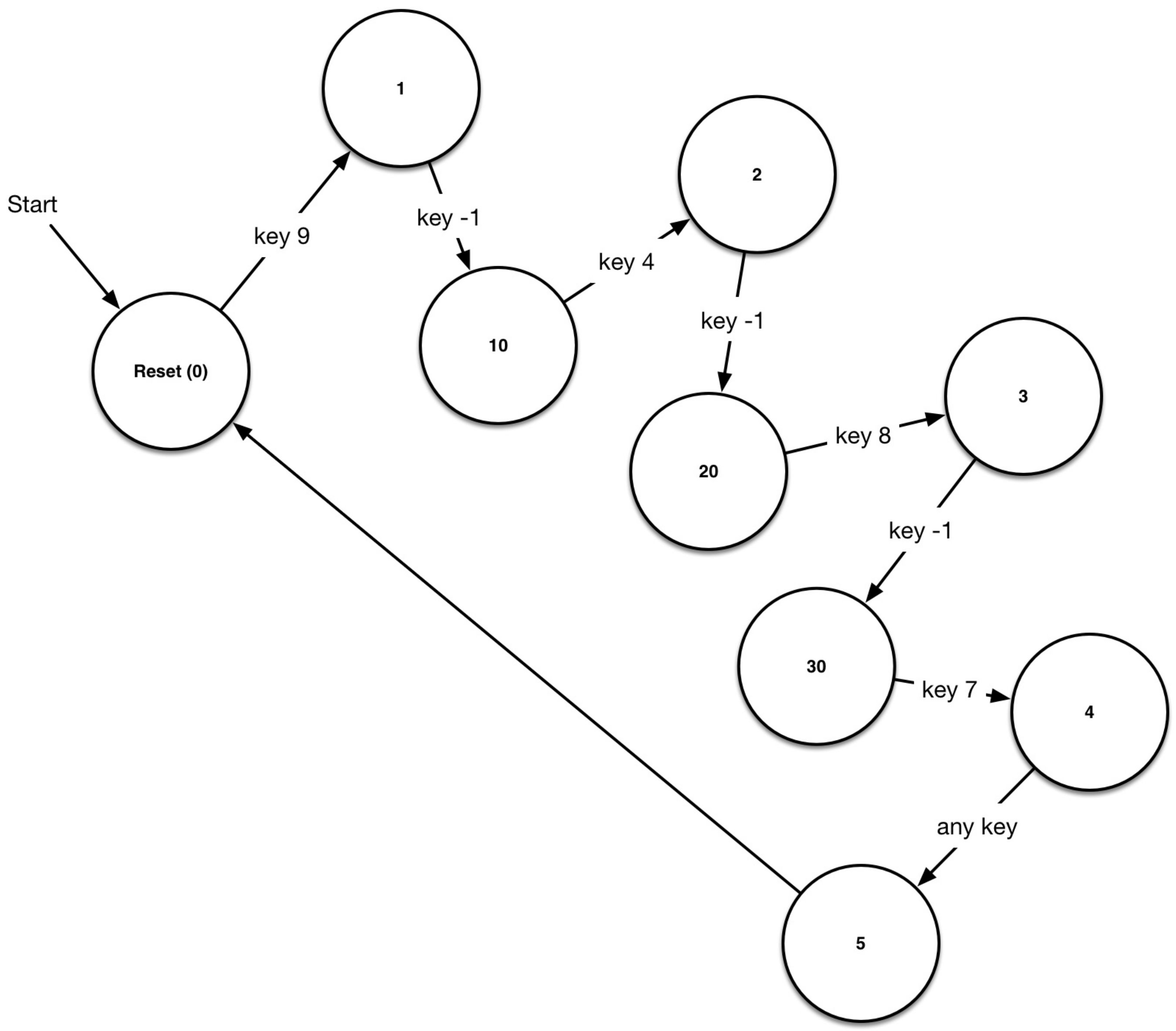
To design a lock program, we need a state machine that represents the functionality we want.

The first attempt at capturing the features were quite interesting.



The state machine functions quite funnily, all of the keys have to be pressed without letting go!

The next attempt at designing a state machine is more sucessful:



Adding the release states, we are able to unlock the keypad lock! But wait, what if someone press every single key 4 times?

```c
while(1) {
    c = read_kypd();
    if (disp_update != c) {
        disp_update = c;
        switch (state) {
            case 0:
                GLCD_DrawString(50, 75, "                   ");
                if (c == '9')
                    state = 1;
                break;
            case 1:
                if (c == '4')
                    state = 2;
                break;
            case 2:
                if (c == '8')
                    state = 3;
                break;
            case 3:
                if (c == '7')
```

```c
                    state = 4;
                break;
            case 4:
                GLCD_DrawString(50, 75, "UNLOCKED           ");
                if (c == '7')
                    state = 0;
                break;
            default:
                GLCD_DrawString(50, 75, "ERR ERR ERR ERR ERR");
        }
    }
}
```

The code says that as long as there is a press of the sequence 9, 4, 8, 7, every press advances the state machine, while the wrong keys stay in the current state. This means we need to capture the case where we press the wrong keys, and return the state back to the beginning of the code pressing sequence.

Finally, the working keypad lock code:

```c
// password_lock.c
#include "stm32f7xx.h"                  // Device header
#include "Board_GLCD.h"                 // ::Board Support:Graphic LCD
#include "GLCD_Config.h"                // Keil.STM32F746G-Discovery::Board Support:Graphic LCD
#include "my_utils.h"
#include "my_kypd_utils.h"

extern GLCD_FONT GLCD_Font_16x24;

int main() {
    schar c, disp_update = ' ';
    char state = 0, lock = 0, lock_update;

    SystemClock_Config();
    GLCD_Initialize();

    // Initialize GPIOs
    RCC->AHB1ENR |= 1 << F | 1 << I;
    for (int i = 0; i < 4; i++)
        set_register_mode(I, i, 1);

    // Initialize screen LCD
    GLCD_SetBackgroundColor(GLCD_COLOR_BLACK);
    GLCD_SetForegroundColor(GLCD_COLOR_WHITE);
    GLCD_SetFont(&GLCD_Font_16x24);

    GLCD_ClearScreen();

    GLCD_DrawString(50, 50, "[Password Lock]");

    while(1) {
        c = read_kypd();
        lock = (GPIOI->IDR & (1 << 11)) >> 11;
        if (disp_update != c || lock_update != lock) {
            disp_update = c;
            lock_update = lock;
            switch (state) {
                case 0:
                    GLCD_DrawString(50, 75, "                   ");
                    if (c == '9')
                        state = 1;
                    else
                        state = 0;
                    break;
                case 10:
                    if (c == '4')
                        state = 2;
                    else
                        state = 0;
                    break;
                case 20:
                    if (c == '8')
                        state = 3;
                    else
                        state = 0;
                    break;
                case 30:
                    if (c == '7')
                        state = 4;
                    else
                        state = 0;
                    break;
                case 4:
                    GLCD_DrawString(50, 75, "UNLOCKED           ");
                    if (lock == 1)
                        state = 0;
                    break;
                case 1:
                    if (c == -1)
                        state = 10;
                    break;
                case 2:
                    if (c == -1)
                        state = 20;
                    break;
                case 3:
                    if (c == -1)
                        state = 30;
                    break;

                default:
                    GLCD_DrawString(50, 75, "ERR ERR ERR ERR ERR");
            }
        }
    }
}
```

Conclusion

In conclusion, our keypad works as expected in the end with a few extra functionalities. Pressing the user button locks the keypad, and the screen refreshes only when there is a user input. We also learned lessions in state machine design, where we have to be quite verbose and accurate in what we want to create. Other things we learned include how the electrical properties of the circuit design purposefully default to a 1 where it defaults to a no key is pressed state. Further work can be done to improve where currently, only half of the activation of the first key '9' is registered as part of the beginning sequence of the passcode.

Full source

Below are the supporting utility files for completeness.

```c
// my_kypd_utils.h
#ifndef MY_KYPD_UTILS_H
# define MY_KYPD_UTILS_H

typedef signed char schar;

schar read_kypd(void);

#endif
```

```c
// my_utils.h
#ifndef MY_UTILS_H
# define MY_UTILS_H

#define A 0
#define B 1
#define C 2
#define D 3
#define E 4
#define F 5
#define G 6
#define H 7
#define I 8
#define J 9
#define K 0xa

typedef unsigned int uint;

void activate_peripheral(uint gpio);

/**
 * gpio name
 * pin  pin of gpio
 * mode 0 for read (input), 1 for write (ouput), 2 for alt function, 3 for analog
 */
void set_register_mode(uint gpio, uint pin, uint mode);

void activate_led(uint gpio, uint pin);

void SysTick_Handler(void);

void SystemClock_Config(void);

#endif
```

```c
// my_utils.c
#include "stm32f7xx.h"                   // Device header
#include "stm32f7xx_hal.h"               // Keil::Device:STM32Cube HAL:Common
#include "my_utils.h"

void activate_peripheral(uint gpio) {
    RCC->AHB1ENR |= 1 << gpio;
}

void set_register_mode(uint gpio, uint pin, uint mode) {
    GPIO_TypeDef  *GPIOx;
    GPIOx = (GPIO_TypeDef *) (GPIOA_BASE + 0x400 * gpio);
    if ((gpio == A && (pin == 15 || pin == 14 || pin == 13)) ||
          (gpio == B && (pin == 4 || pin == 3))) {
        GPIOx->MODER &= ~(0x3 << pin * 2);
    }

    GPIOx->MODER |= mode << pin * 2;
}

void activate_led(uint gpio, uint pin) {
    GPIO_TypeDef  *GPIOx;
    GPIOx = (GPIO_TypeDef *) (GPIOA_BASE + 0x400 * gpio);
    GPIOx->MODER |= 1 << pin * 2;
    GPIOx->ODR   |= 1 << pin;
}

void SysTick_Handler(void) {
    HAL_IncTick();
}

void SystemClock_Config(void) {
    RCC_ClkInitTypeDef RCC_ClkInitStruct;
    RCC_OscInitTypeDef RCC_OscInitStruct;
    HAL_StatusTypeDef ret = HAL_OK;

    /* Enable HSE Oscillator and activate PLL with HSE as source */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLM = 25;
    RCC_OscInitStruct.PLL.PLLN = 432;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 9;

    ret = HAL_RCC_OscConfig(&RCC_OscInitStruct);
    if (ret != HAL_OK) {
        while (1) { ; }
    }

    /* Activate the OverDrive to reach the 216 MHz Frequency */
    ret = HAL_PWREx_EnableOverDrive();
    if (ret != HAL_OK) {
        while (1) { ; }
    }

    /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2 clocks dividers */
    RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_PCLK1 |
                                   RCC_CLOCKTYPE_PCLK2);
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

    ret = HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7);
    if (ret != HAL_OK) {
        while (1) { ; }
    }
}
```