Final Project | MPS Fall 2019

-- A Double-Security System with Password and RFID Verification

Group: Jiahong Ji (Frank), Lizheng Liu (Tiger)

Introduction and Background Information

The security of real properties and data is a hot-spot with a wide range of solutions and discussions in both our physical and cyber world. We anticipate that, the importance of keeping personal or business information secure will rise from the present to the future.

The existing technologies of physical security system include fingerprint verification, face ID recognition and access card. We decide to utilize the resources of STM32F769 to build a system of security, which uses classic and convenient ways in the distinction of users, and to simulate a simple situation verifying of the legitimacy of a person to a system, and to only allow the person entering the correct password to access the system.

We implemented the reading and processing of the data from a number pad and a RFID signal receiver in the process of verifying a legitimate log-in. Several firmware utilities of the STM32F769 microprocessor were utilized in this project: GPIO, Timer, interrupts, and SPI. The number pad is a matrix peripheral read by the digital inputs, and the RFID receiver is a SPI slave device which communicates with the STM32 board unidirectionally. Timer generation was used for the timing characteristics of the security system such that the time of the valid input of the password is limited.

In the development progress, we started with exploring the operation of the keypad, and then the RFID verification, and finally the system integration logics. Intermedium tests were conducted at each of the development steps, and analysis were made to each of these subsystems.

Experimental Procedure, Result, and Analysis

1. Keypad Subsystem

1.1 Experimental Procedure

For the first subsystem, it will be the keypad configuration. The main goal for this part is to configure it to achieve the given function so the user may be able to use it as an external devices to input their password. So, for this part, the keypard we are using is the *WINGONEER 5Pcs 4x4 Universial 16 Key Switch Keypad*, which we purchased it from the amazon. We wrote a different test code for testing its function.

First of all, we need to figure out its internal structure. From the data sheet, we see that the internal connection of the keypad looks like the following:(Figure 1)

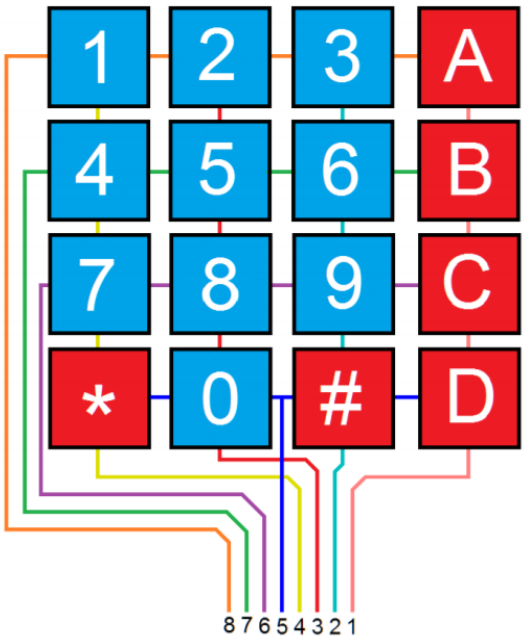

Figure 1: Keypad Datasheet

Basically, it is a Matrix keypads that use a combination of four rows and four columns to provide button states to the host device, typically a microcontroller. Pin 1, 2, 3, and 4 are correspond to the first, second, third, and the fourth column of the keypad, and Pin 5, 6, 7, 8 are corresponding to the firth, second, third and the fourth row of the keypad. If, for example, if “1” is pressed, the first column and the first row will form connection, and the pin 8 and pin 4 will be connected, and vice versa for the other input button.

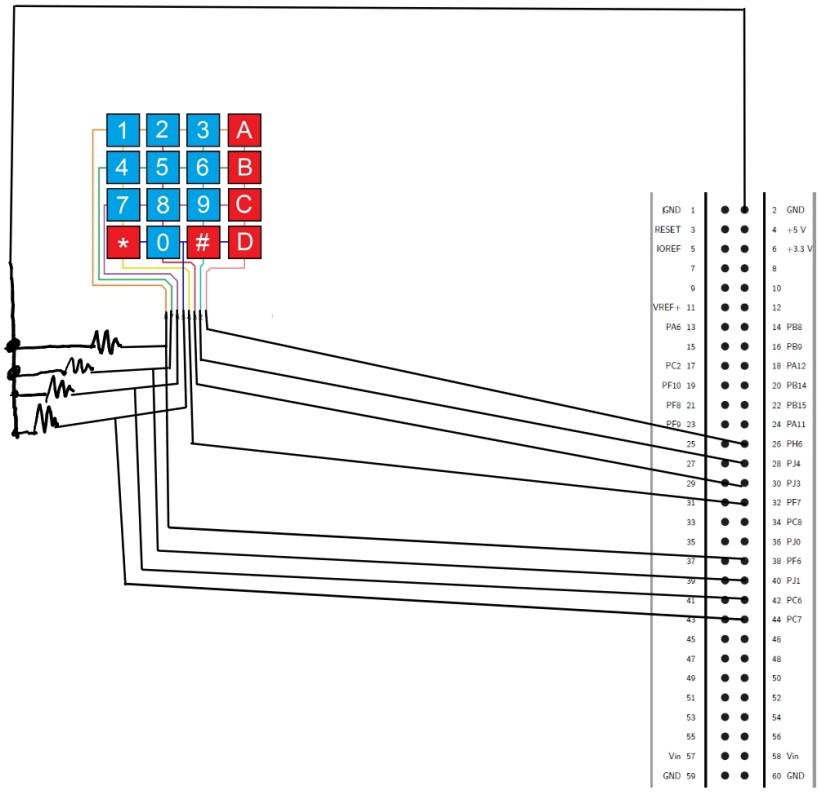

Figure 2: Keypad Hardaware Configuaration

So, for our resigning, we decided to use four to output voltage to the keypad, and use the other for pins to read the inputs. In our initiation function, we set J4, J3, H6, F7 as the GPIO output pins, and F6, J1, C6, C7 as the gpio input pins. We used the GPIO configuration in lab one, and set all of those pins accordingly. Then, since the CPU clock rate is much faster than the speed for an human beings to press a push bottom, we decide to fire each of the four output pins altinavely. And during each time when a output pin is fired, we will detect the whether the four inputs pins successfully read something or not. Also, since the keypad may not return anything that is meaningful but a voltage signal, we also need to manually write down the character on the keyboard and stored it in somewhere. Take the keypad number “5” as the example to illustrate that. From figure one, you can clearly see that “5” is on the second column and second row. So, when the second row was fired, the system will detect whether the four inputs has reading in them or not. Then, of course, the second input pin which is on second column will read something. As the result, the second output pin, which we fired before, will be closed ahead of time. Furthermore, the value 5 will be return. Since rebounding may happened and time it take for a human being to complete one pressing is much longer then the time it take for the CPU to execute the whole loops, *HAL_Delay(500)* will also be used over here to paulse for 500 ms and resolve the rebounding issues.

Since the format of the password is an 4 byte character, the function will loop inside a while(i<4) to make sure we can successfully 4 character string to the Microprocessor.

1.2 Result and Analysis

In generally, we believe the keypad subsystem is kind of an success story. While the keypad is just, to some extent, a bunch of push buttons in series and cannot directly trammit character signal, we successfully turned those into some useful things. When the corresponding buttons are pressed, the Microprocessor will store those characters accordingly. Also, the final version of the keypad testing code is not buggy and easy to use.

1.3 Discussion and Observation

Our initial method for reading the inputs is kind of a failure. We tried to fired the four output pins at the same time, and also tried to read the four inputs at the same.. From our observation, we realize that if we doing so, we may be only able to read the first row accordingly. So, we decided to use fire those four output pins alternatively. Unfortunately, after we made such change, we realize that the input pins may still read the signal unreliable. In some cases, it may read the right value, but in the other cases, it may go wrong. So, we carefully test the individual pins, and we find out the reasons why this may happened is that we thought open circuit may be able to let the input pins read a 0. Nevertheless, when doing the individual pin testing, we found that the noise from the environment may make the open pin read 1 sometimes. So, we changed our circuit slight by do the grounding for those input pins, and add four 1k resistor between the input pins and the group. After we doing so, those inputs pins may steady read 0 when none of the button is pressed.

2. RFID Subsystem

2.1 Experimental Procedure

The RFID receiver RC522 (figure 2.1.1) is compatible with data transmission protocol of I2C and SPI, regarding to the knowledge we have learnt in this course, we decided to use SPI protocol to set up the communication between the STM32F769 microprocessor and the RFID receiver. An open resource library was employed to help with the read and write functions of the RFID receiver.

The SPI configuration of the STM32 as the master of the data transmission was set in the *main.c* file as a independent function *SPI_Init().* SPI channel 2 was utilized to operate the communication. Then the baud rate prescaler was set to 32 to fit the operational frequency of the data transmission of the RFID receiver. The clock signal of the SPI protocol was set to 1 edge clock phase and low clock polarity, so that the data transmission is only sensitive to the first low to high bit of the clock signal. Then the data bit of the SPI communication was set to 8-bit because the data transmitted was the distinction number of the RFID tag, which is an 8-byte series, and each byte transmitted is 8-bit. And the first bit of the data transmitted was

set to the most significant bit, which means that the data were left sided. The chip select pin on the slave device was set to listen to the software setting of the master device.

Figure 2.1.1 RFID RC522 Receiver

Next the *HAL_SPI_Init()* function was called to initiate the SPI configuration, and this function calls the *HAL_SPI_MspInit()* function with the GPIO settings serving the SPI protocol. In the *HAL_SPI_MspInit()* function, the following commands are addressed:

Data Line	GPIO Port	GPIO Pin
SPI CLK	A	12
SPI CS	A	13
SPI MOSI	B	15
SPI MISO	B	14

Table 2.1.1 SPI MspInit Settings

In the main function, **TM_MFRC522_Check()** was called to read the ID and the type of the RFID tag contacting with the RFID receiver repetitively. This function returns the ID of the RFID tag and store it in a variable. The program would print it on the screen and signify the user which account is logged in through the RFID tag. In the RFID check function, a RFID request was made first to ensure the valid RFID tag was detected. Then if the return status by the request function *TM_MFRC522_Request()* was *OK,* RFID anti-collision function and RFID select tag function were called to check that the same RFID tag serial number was read for four consecutive times, so that there was no potential read of two different RFID tags. Then the RFID check function was done, the ID and the type of the tag was returned to a global variable in the main function.

There were a variation of functions in this library, but only necessary ones to our project were chosen.

2.2 Result and Analysis

To test this subsystem of the RFID tag reading, the whole library was utilized but only one block of code was needed in the main function, that was to check the status of the RFID receiver and print the serial number of the tag detected on the terminal. The user can observe the number of the RFID tag to know the account that is logged in. Different RFID tags have different serial numbers read by the RFID receiver, so the user could make distinctions between them.

2.3 Discussion and Observation

When the RFID reading system was firstly tested, there were various problem. The GPIO settings to the RFID receiver was not configured to the correct pins, and the status of the SPI checking function was always returning busy channels, which should be ready channels for the SPI to work. So we reviewed the hardware manual of the STM32 microprocessor and found out that the assigned pins to the SPI GPIO pins were not configured properly. So we changed the pins and the returned value of the SPI functions were normal.

Another problem encountered in this session was the format of the returned string of the RFID check function. The RFID ID returned was an array with four elements. When tested at the first time, the program printed this array as a whole integer, and the displayed value was weird. We observed that different RFID tags had the same number printed on the screen. Then we noticed that we need to print each piece of element in the array. After making this change, the printed RFID serial number had distinctions between different RFID tags.

3. System integration

3.1 Experimental Procedure

After we verify the functionality of the keypad and the RFID reader, we started integrating the whole system. Since the RFID reader is the most vital thing in this project, we decided to use the lab_3 template to assist us completing the integration. In the general, out final codes has 5 files: init.c, main.c, rfid_functions.c, rfid.c, uart.c. For the GPIO pins configuration, we putted it in the init.c file. All of the inputs, outputs, and LED's GPIO initialization were wrote inside the Sys_Init(). Beside that, since the slave device, RFID reader, required lots of codes to accomplish its given function, there were putted in two separate file: rfid_functions.c and rfid.c. rfid.c includes the SPI initiation function and some basic configuration. rfid_function.c include all the required functionality for the SPI slave devices. In the *m*ain.c, it include the keypad reading function, timer initiation function, and the timer interrupt function.

For the hardware configuration, we simply combine the hardware configuration we used for keypad subsystem testing and the RFID subsystem testing, since we deliberately to choose those pins which will not conflict with in each of our subsystem design. The final schematic for our project look like the following

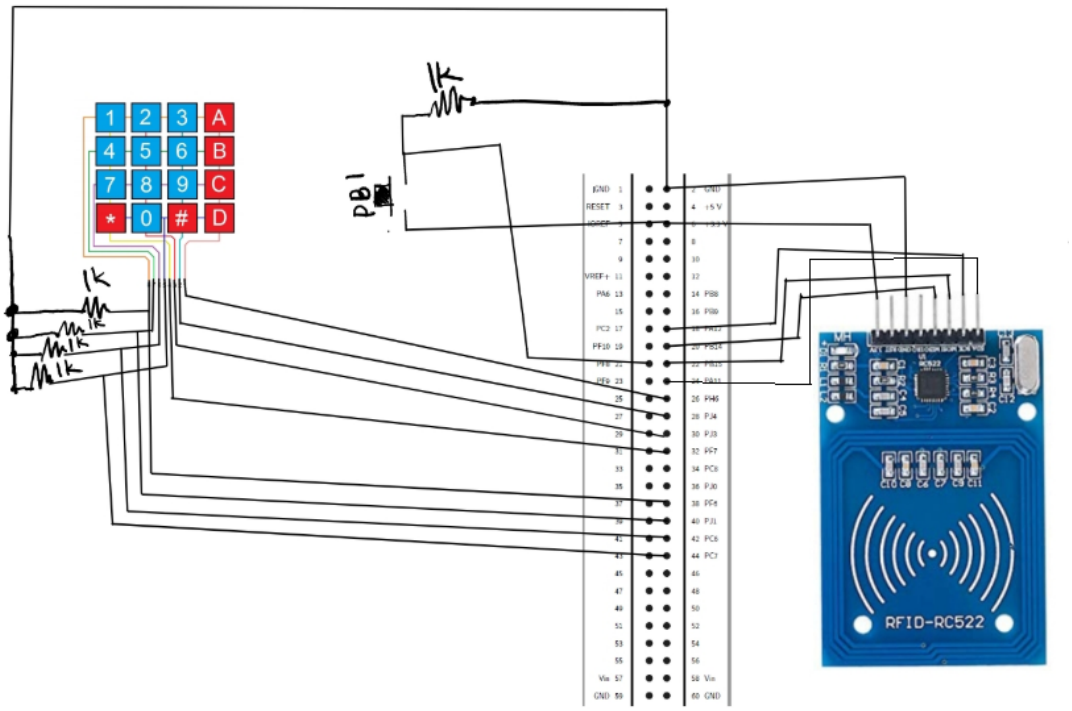

Figure 3.1: System Hardware Configuration

Since, we did not use timer in any of our subsystem, we will set up the timer accordingly. So, we used the basic timer, timer 7, for our project. We initialize its function according. Then we used the following formula to set the timer as one-second counter:

$$\text{Overflow Time} = \texttt{TIM}n\texttt{\_ARR}\left(\frac{\texttt{TIM}n\texttt{\_PSC}+1}{\texttt{TIM}n\texttt{CLK}}\right)$$

Formula 1: Timer fomular

Since we want the one second period, the Prescalor of the timer is setted to 2472, and the period of the timer is setted to 43690. Then, we setted the timer as the interrupt mode, since we need to use it almost in everywhere. In the HAL_TIM_PeriodElapsedCallback(), we will at 1 on the variable counter_1.

So, the illustration above is basically all we need to cover for basic initialization. Then we can move on to the main function in main.c. In the begging, we will called the Sys_Init(); Init_Timer(), SPI_Init(), RFID_RC522_Init() and Init_GPIO() functions to initial all the pins, timer, and the SPI slave device.

Then we will enter while(1) loops. At the begging, will keep reading the status of the RFID by calling the TM_MFRC522_Check() function. Only when the card is read and the function return MI_OK, we will continue to the next part of the code. When the card is read, we will enter the keypad_reading function. Beside for the regular keypad function, we also added the timer within the function. At the begging of the keypad reading function, we will clear the count_1 to zero. And when we are reading inputs from the keypad, the timer interrupt will keep increasing the counter_1 flag. If it reach the value 10, it will break out the whole loop and return a flag called overtime. If this happened this means that the user took too long to enter the password, and it will lose one chance of trying. Also, the red LED on the stm32 board will be lighted up accordingly. For the inputting password loop, it is controlled by an for loop. If the user enter the correct password, it will break out the for loop, light up the Green LED. On the other hand, if the user enter the wrong password for four times, it will set the value frozen, which means this user account should be forzed for a while, to one. As the result, in the next time, the code will not execute the tag reading code. Instead, it will hold for 20s, and the user cannot do anything about that except for waiting. Furthermore, since a switch should also be responsible for the control the whole circuit. So, we add a open-close

detection line within the timer interrupt, since it will go through that function not matter what happened in the main code.

3.2 Result and Analysis

Since we putted lots of efforts on the two individual subsystem, we did not meet too many difficulty during the whole process. After we ressemble them, they just simply work fine. When the RFID is read, it will required the user to input his password, is the password is correct, the account will be online, which is shown by printing out an messenger and lighting up the green LED. If the input is incorrect, or the user takes too long to enter one change, a incorrect messenger will be printed in the terminal. If the user continually enter the wrong password for four times, hie/her account will be locked. An sample output looks like the following:

```
The ID is 6eb5cb61.
Plz enter your password
1s
1s
1s
1s
1s
your inputs is 0000
the password is incorrect, keep entering
1s
1s
1s
1s
your inputs is 0000
the password is incorrect, keep entering
1s
1s
1s
your inputs is 0000
the password is incorrect, keep entering
1s
1s
1s
1s
your inputs is 0000
the password is incorrect, keep entering
1s
The bank system has been frozen due to mutiple incoorect enter
Plz come back later 20s
1s
1s
1s
```

Figure 3.2: Sample output

3.3 Discussion and Observation

During the whole debugging, one of the challenge we faced is that the compiler cannot compile the timer. Since, we are using the template for lab 3, which is mainly dealing with SPI slave devices, the timer functions seems not included in the projects. So, we described the circumstance and tried the find a solution on Google. So, finally, we found out that include timer in the configuration file. After we make such adjustment, the timer can start working.

Another interesting thing is that when we are trying see weather or not we can read the input pin in the timer interrupt function, we found out the doing so may directly hold the whole program. We just simple add the following two line in the HAL_TIM_PeriodElapsedCallback. Luckily, that is exactly what we desired. But what confuse us is that why it can hold the whole program if the switch is off, even through we have not add the Hal_day or while to below it. Our assumption is that this happened because reading the pin may take some time, and the speed for reading that input is much slower than the time of one period of time. As the result, the program is halt since we enter the next interrupt before the reading is completed.

```
if(HAL_GPIO_ReadPin(GPIOF, GPIO_PIN_8))
        printf("and it's on \r\n");fflush(stdout);
```

Conclusion

To conclude our final project, we have achieved digital reading of the input through the GPIO ports, the transmission of data using the SPI protocol, and the timer utilities. To manipulate these functions of the STM32F769 microprocessor, we searched guides in the user's manual, examples and open resources on the internet.

In this project, we utilized multiple programming and circuit techniques in building the functions and main system of our security system, and realized a variety of learning outcomes in the microprocessor systems. At least three laboratories have potential help and similarities to the operation of our experimentation. Besides combining these different approaches studied in this semester, constructing a self-defined meanful system and challenging ourselves are essential outcomes of this project.

We expected to use and develop more softwares and utilities on the microprocessor, and our double-security system is not only showing what we have learnt about the STM32F769 microprocessor, but also endows us more understanding to the possibilities grown on this chip.

Appendices

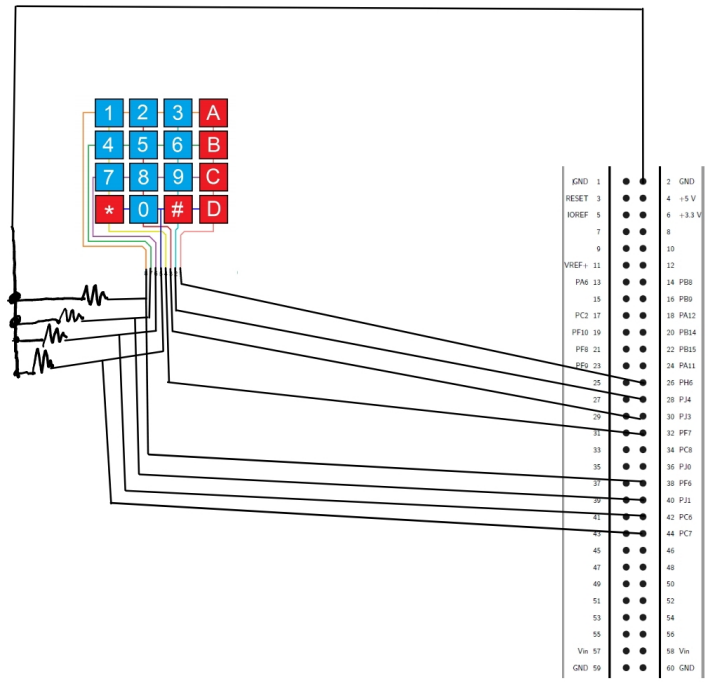

Figure 1: Keypad Hardware Configuration

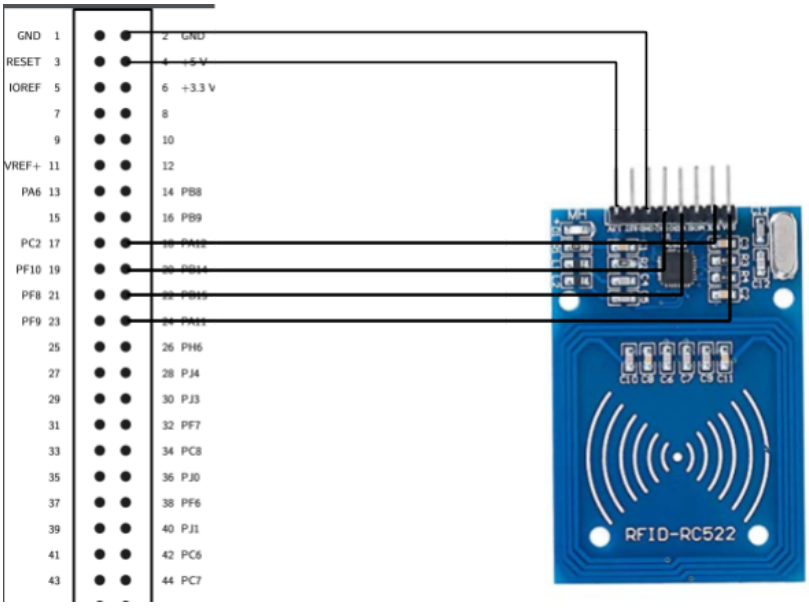

Figure 2: Keypad Hardware Configuration

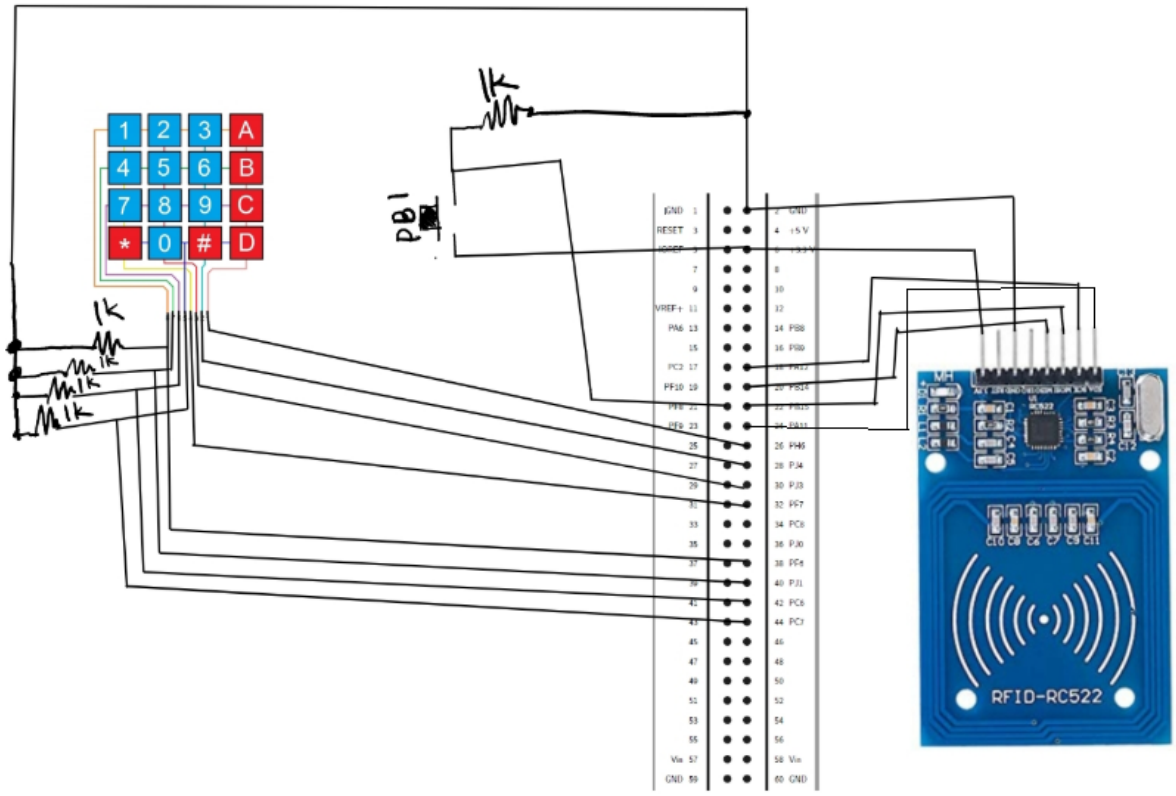

Figure 3: Final System configuration

Reference

1.MPS_Breakout_Board_Mapping RPI. [Online]. Avaliable: https://www.ecse.rpi.edu/courses/F19/ECSE-4790/Documents/MPS_Breakout_Board_Mapping.pdf

2.32f769i_Discovery_Manual STM32. 8, 2018. [Online]. https://www.ecse.rpi.edu/courses/F19/ECSE-4790/Documents/32f769i_Discovery_Manual.pdf

3.STM32_Breakout_Schematic RPI [Online]. Avaliable. https://www.ecse.rpi.edu/courses/F19/ECSE-4790/Documents/STM32_Breakout_Schematic.pdf

4.STM32_HAL_and_LL_Drivers STM32. 8, 2018. [Online] https://www.ecse.rpi.edu/courses/F19/ECSE-4790/Documents/UM1905-stm32f7_HAL_and_LL_Drivers.pdf

6. *Rationale for International Standard - Programming Languages - C*. Erscheinungsort nicht ermittelbar, 2001.

7. RFID Open Source Library, https://github.com/xtrinch/stm32f7-demos, [Access: November 31, 2019]

8.*MFRC522 Standard performance MIFARE and NTAG frontend https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf*

/95