THE UNIVERSITY of TULSA

Electrical and Computer Engineering

Oscillator on an STM32 NUCLEO-F411RE

Design Report

Alec Izett

December 14, 2021

Revision History

Date	Version	Author	Description
25NOV2021	1.0	Izett A.	Create Template, Preemptive Editing, and Import from Project Requirements.
30NOV2021	1.1	Izett A.	Write Mechanical and Electrical System Design.
2DEC2021	1.2	Izett A.	Polish and Edit Mechanical and Electrical System Design. Write Electronic System Design.
8DEC2021	1.3	Izett A.	Fill in Mechanical, Electrical System, and Electronic System Design Writing. Create Figures and Tables for Aforementioned Paragraphs.
11DEC2021	1.4	Izett A.	Added, Formatted, Fixed, and Referenced Figures. Adjusted List of Project Requirements to Reflect the Finished Project. Created "Information on Notes" Table and Listings of Code. Completed Testing Requirements. Created User Guide. Completed Lessons Learned. Imported Source Code. Completed Bill of Materials. Added Bibliography Entry.
13DEC2021	1.5	Izett A.	Added Proof of Testing in Testing Requirements Section. Proof Read, Edited, and Fixed Mistakes.

Customer Approval

Name	**Role**	**Signature of Approval**	**Approval Date**
Alec Izett	Engineer		XXDEC2021
Nathan Hutchins	Customer		XXDEC2021

Contents

List of Figures

List of Tables

Listings

Abstract

This project utilizes the STM F411RE board, an amplifier, and a speaker to produce sound. The F411RE will produce a low amplitude wave at various desired frequencies. This signal will be sent to the amplifier, which will make the signal louder. The speaker then produces this signal to the physical realm. Every note available will be tuned to the western equal temperament tuning system. The notes will come in the form of a square wave, a wave that jumps between high and low with little transition time. The period of this wave may be adjusted to result in desired frequencies. This wave is output to a single pin on the F411RE. This pin will be integrated into an amplifier circuit. The purpose of this project is to apply knowledge of embedded systems to music. A square wave oscillator is one of the foundational building blocks to music production with electronics. The output of this oscillator may be used in a digital audio workstation. Importing the output of this design into such a program would allow the user to apply various effects to the sound. This process is often performed with physical electronics, but modern audio workstations can efficiently perform this task at a much lower price point. Thus, the objective is to create an instrument that may be used in traditional music production.

Nomenclature

F411RE STM32 NUCLEO-F411RE
STM ..... STMicroelectronics (Integrated Device Manufacturer)
C ..... The programming language C

Chapter 1

Introduction

1.1 Introduction

To introduce this project, we must understand the fundamental building blocks of this project. First, the F411RE is a powerful development board built primarily for prototyping. That being said, this board was not developed with music production in mind. STM develops other boards more in line with that design philosophy, and the F411RE is not one of them. Thus, this project presents the challenge of making a board not tailored to music produce music. This can be done by carefully tuning the frequency at which an output port toggles on and off. The frequency this pin toggles may be controlled by a series of button inputs. Each of the notes available will need to be carefully tuned to the correct pitch. The overall output will be amplified to ensure it is audible. The amplifier will come in the form of a simple circuit design that amplifies the incoming voltage signal. This output can then be easily sent to a magnet and coil speaker to be played.

1.2 Objective

This project involves the development of the F411RE board to produce the desired wave. Tuning this wave will involve tying the frequency of the voltage to an onboard timer. The tuning of these waves will heavily lean on the accuracy of the F411RE's onboard timer. Math can be performed on the clock cycle of the board to produce the desired period and thus the desired frequency. This project involves working with sound and sound amplification are topics that have not been touched on in the course work for Embedded Systems in C. Thus, the entirety of the amplification circuit must be researched and developed from the ground up. A portion of the project's research will also be taken up by the development of the program for the F411RE. Forcing a pin to output a tuned wave will be a tedious ordeal, requiring hours of fine-tuning and troubleshooting. The amplifier design outputs a signal directly into the input of a speaker. Thus, little research is required for the speaker portion of this project.

1.3 Customer

This project is primarily for Dr. Hutchins as the final project for ECE-2263, or Embedded Systems in C, at The University of Tulsa. This project also serves as an introduction to combining music production with engineering for the lead engineer of the project, Alec Izett.

1.4 Design Team

1.4.1 Engineering Manager/Lead Engineer

Alec Izett

Chapter 2

Project Requirements

2.1 Hardware Requirements

2.1.1 F411RE

This project will utilize the STM32 F411RE board.

2.1.2 Input

This project will include an input method that allows different notes to be selected by the user.

2.2 Electrical Requirements

2.2.1 Amplifier

This project will have an amplifier circuit which boosts the output signal of the F411RE.

2.2.2 Speaker

This project will be able to produce sound on a speaker.

2.3 Software Requirements

2.3.1 Musical Notes

This project will produce notes that are properly tuned in the equal temperament system.

2.3.2 Note Selection

This project will allow the user to select from a set of notes that are determined in the projects code.

Chapter 3

Hardware Design

3.1 Mechanical Design

As mentioned in section B of this report, we use the STM32 F411RE, LM386-4, a matrix keypad, two 100µF (25V) capacitors, one 100nF (25V) capacitor, one 1mF (25V) capacitor, a donor magnet and coil speaker, various connecting wires, and a USB to micro USB cable.

The circuit starts with the USB to micro USB cable. This cable's purpose is to carry power from a computer to the F411RE board and to upload code from a computer to the F411RE. The F411RE is the brain of the whole operation. It provides the original tuned waveform. The output of the F411RE is then sent to the amplification circuit, the design of which can be seen in greater detail in section 3.3. The amplification circuit is powered by either the 5V or 3.3V source pins from the F411RE; tying the circuit to the 5V source will result in a louder final volume when compared to the 3.3V source. The output of the amplification circuit is then sent to the speaker. I chose to reuse an old wireless speaker I had lying around. This wireless speaker had two magnet and coil speakers inside, so I wired them in series; allowing me to connect the speaker with only two wires instead of four. The wireless speaker already had an enclosure, so I did not need to make one. All that is necessary to recreate a similar speaker setup to mine is to find a speaker that has a positive connection and a negative connection. Preferably this speaker would be in an enclosure. Also, it is preferable to have a mid-range speaker, somewhere around 4 inches (roughly 10 cm) in diameter. A large speaker would be around 12 inches (roughly 30 cm) and would be considered a sub-woofer, a small speaker would be about 1 inch (roughly 2.5 cm) and would be considered a tweeter. We just want to avoid the two extremes. Lastly, we use a matrix keypad to control what notes are being played. This matrix keypad is connected to the F411RE. That all being said, a diagram for how these components are connected together can be seen in Figure 3.1 below.

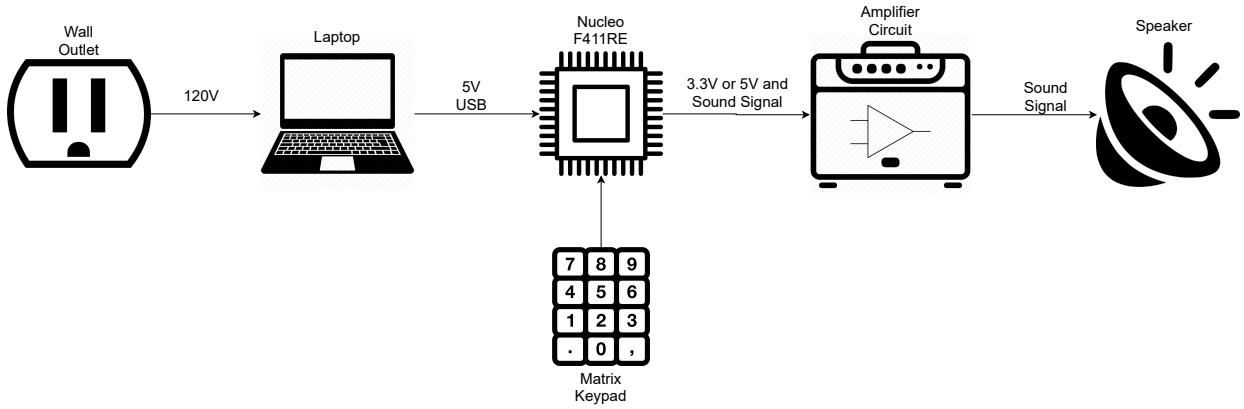

Figure 3.1: Hardware and Power Overview

Also worth mentioning is the hardware functionality of the matrix keypad. The keypad is broken down into 4 row wires and 4 column wires. This means we have 8 pins that we can communicate with that come off of the keypad. If we do the math, that is 16 buttons with only 8 pins to communicate with. The 4x4 keypad works more like an X-Y coordinate system more than an individual button system. From Figure 3.2, we can see number "1" is in the first column and in the first row. This means we can program a function that detects such an occurrence and deciphers it as meaning the character "1". A deeper dive into the programming of the board can be seen in Appendix A.

Figure 3.2: Matrix Keypad

3.2 Electrical System Design

Power for this project comes first from a traditional wall outlet. This powers the laptop, which the F411RE is connected via a USB to micro USB cable. The F411RE then powers the rest of the circuit design. Power from the F411RE is sent to the amplifier circuit which then sends power to the speakers. The power delivery and physical design of the project go hand in hand. Thus, both the power delivery and physical design have been implemented into a single figure, see Figure 3.1.

3.3 Electronic System Design

The main bulk of the electronic system design is taken up by the amplifier circuit design. I sourced my circuit design from a Youtuber named Afrotechmods. I used this video in particular (clickable link). The LM386-4 is the main powerhouse of the amplifier circuit. The chip takes in a signal and boosts it to the amplitude of the source voltage. The signal may then be transmitted to the speakers. The circuit design also includes several capacitors. These capacitors serve the purpose of controlling irregularities in the signals flowing through the design. They allow only the parts of the wave we want to hear to come through. It is worth noting that the amplifier can be powered by any voltage value ranging from 4V to 12V. This project allows the amplifier to be powered by 3.3V or 5V, which is more than enough amplification. The slightly modified circuit design of the amplifier may be seen below in Figure 3.3.

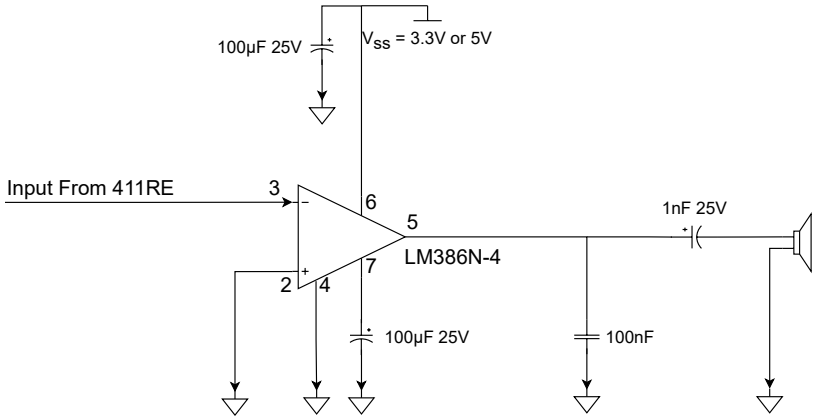

Figure 3.3: LM386N-4 Amplifier Circuit Design

Chapter 4

Software Design

4.1 Matrix Keypad

To implement the matrix keypad in software, we program the rows as GPIO inputs and tie them to an interrupt. We then program all the columns as GPIO outputs, which will read what the GPIO inputs are saying. That being said, physical buttons have an issue known as button debounce, or button bonce. This is when a button will flutter in state for a short period of time, it will change between low and high multiple times for a few milliseconds. To solve this we can wait a few milliseconds between first contact and grabbing the value of a button press. The code for this matrix keypad decoder can be seen in Appendix A as it is quite alot of code to insert here.

4.2 Making Notes and Using the Timer

Every note in this project is in the C Major scale. A breakdown of each note in the program, its associated frequency, and its associated button on the keypad can be seen below in Table 4.1. In order to get a pin to output the correct frequency, we must ground the note in a value that we control. To do this, I tied a variable to the system clock that oscillates at 100MHz. I then carefully timed each note to be a fraction of this system clock, effectively making a controlled division of frequency. After trying various methods of keeping track of the system clock, I found the following method to be the most controllable. I placed a function in the main loop of the code to check if the system timer had changed. If the system timer changed from its previous value, then a temporary variable would add one to its value. The code for this function can be seen below Listing 4.1. It is worth noting that the waveform generator only works in square waves. This is not an issue, as we are working with low-level music generation.

Note on Keypad	Note Name	Notes (Hz)
1	C4	261.63
2	D4	293.66
3	E4	329.63
4	F4	349.23
5	G4	391.00
6	A4	440.00
7	B4	493.88
8	C5	523.25

Table 4.1: Information on Notes

```
1       timerRaw = __HAL_TIM_GET_COUNTER(&htim3);
2       if(timerRaw != timer){
3           timer++;
```

Listing 4.1: Timer Code

4.3 Overall

We can combine these two pieces of code and create a multi-note wave generator. A button is selected on the matrix keypad, and the code subsequently plays a specific note as a result. A flow diagram for this code can be seen below Figure 4.1. A portion of the code for this section may also be seen below in Listing 4.2.

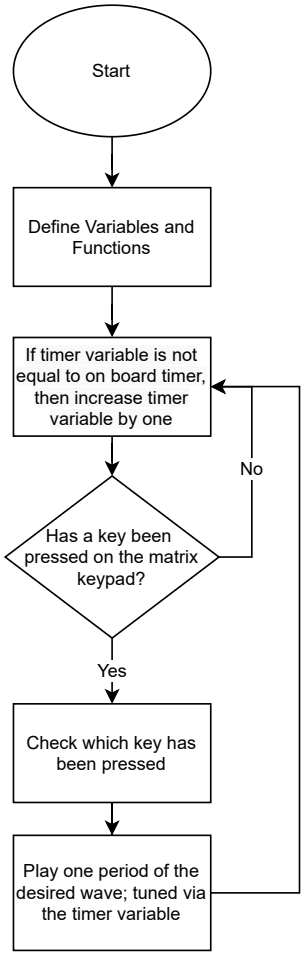

Figure 4.1: Code Flowchart

```
if(keyChar != 0){
    //C4 1 261.63
    if(keyChar == 49){
        if(timer >= 2582){
          timer = 0;
        }
        if(timer <= 1241){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET);
        }else if(timer >= 1241){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);
        }
    }
    //D 2 293.66
    if(keyChar == 50){
        if(timer >= 2300){
          timer = 0;
        }
        if(timer <= 1150){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET);
        }else if(timer >= 1150){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);
        }
    }
```

Listing 4.2: Notes Code

Chapter 5

Testing Requirements

5.1 Hardware Testing Requirements

5.1.1 F411RE

This project will utilize the STM32 F411RE board. To test this requirement, we can simply check if the main board used is the F411RE. A photo of the F411RE being used can be seen in Figure 5.1.

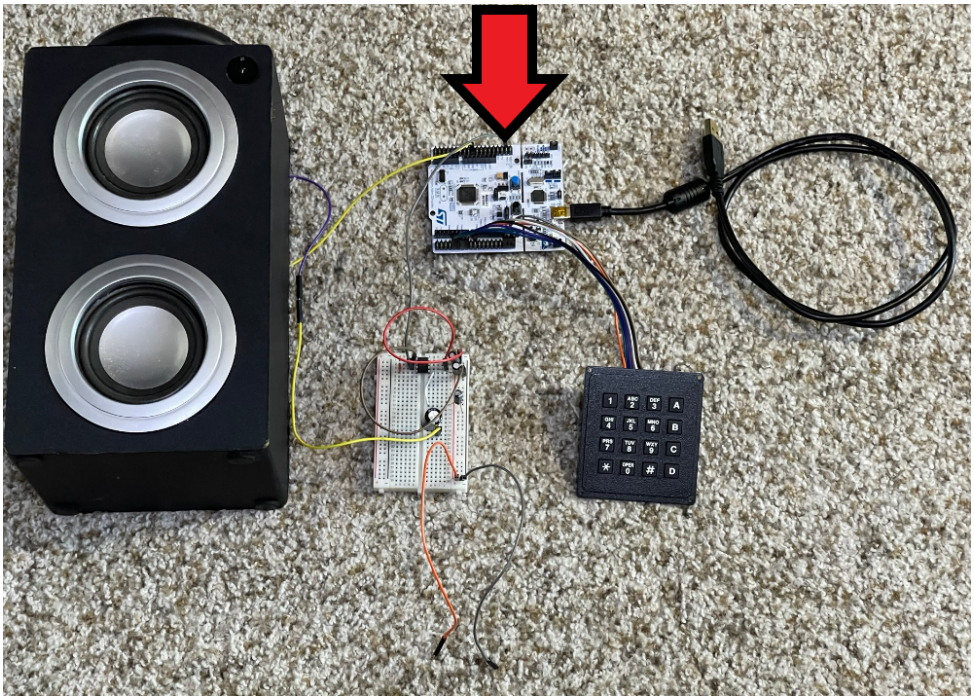

Figure 5.1: F411RE is Used

5.1.2 Input

This project will include an input method that allows different notes to be selected by the user. To test this requirement, we can select several different buttons on the matrix keypad and ensure that each button produces a different note. The best way to show this is via a video available on YouTube (clickable link). (https://www.youtube.com/watch?v=xQuxkakWAjw)

5.2 Electrical Testing Requirements

5.2.1 Amplifier

This project will have an amplifier circuit which boosts the output signal of the F411RE. To test this requirement, we first measure the volume of the unamplified circuit in decibels. Then we can power the amplifier and measure the volume of the circuit in decibels. The amplified sample should be louder than the unamplified sample. See Figure 5.2 to see the decibel readings of the ambient room, the unamplified signal at 5cm away from the speaker, the 3.3V amplified signal at 5cm away from the speaker, and the 5V amplified signal at 5cm away from the speaker.

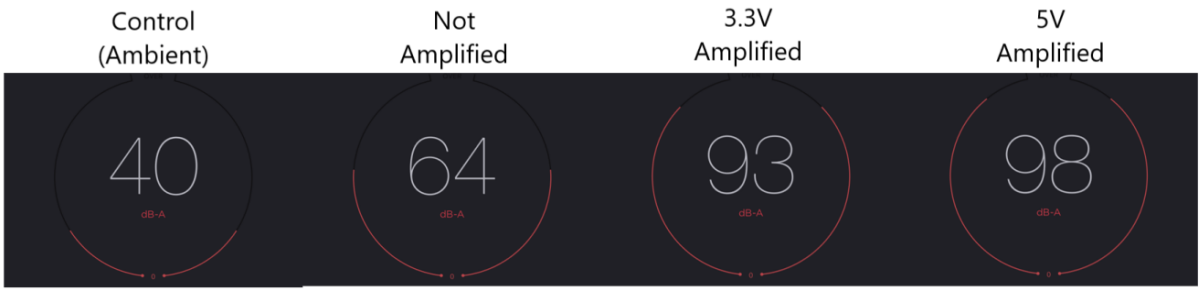

Figure 5.2: Volumes Under Different Circumstances

5.2.2 Speaker

This project will be able to produce sound on a speaker. To test this requirement, we ensure that the speaker is outputting a sound when desired. The best way to show this is via a video available on YouTube (clickable link). (https://www.youtube.com/watch?v=xQuxkakWAjw)

5.3 Software Testing Requirements

5.3.1 Musical Notes

This project will produce notes that are properly tuned in the equal temperament system. To test this requirement, we can place a musical instrument tuner in front of the speaker. We can output various notes and check if the frequency is reasonably close to the frequency we desire for the given note. Below is Figure 5.3 showing that each note is in tune with the use of a mobile phone tuner.

Figure 5.3: Notes' Tuning

5.3.2 Note Selection

This project will allow the user to select from a set of notes that are determined in the projects code. To test this requirement, we can select several different buttons on the matrix keypad and ensure that each button produces a different note. The best way to show this is via a video available on YouTube (clickable link). (https://www.youtube.com/watch?v=xQuxkakWAjw)

Chapter 6

Users Guide

6.1 How to use the F411RE Oscillator

Interacting with the oscillator is as easy as interacting with the matrix keypad. Each note corresponds to a key on the matrix keypad, as seen in Table 4.1. The only other decision the user has to make is how amplified they want the resulting sound to be. This can be changed by not powering the amplifier circuit, by powering the amplifier circuit with 3.3V, or by powering the amplifier circuit with 5V. The higher the voltage, the louder the output will be.

Chapter 7

Lessons Learned

I learned how to build a basic amplifier circuit. Operational amplifiers were discussed heavily in some of my other coursework. That being said, I was never able to apply operational amplifiers to a project until now. I had previously used operational amplifiers in an ideal environment and did not use them for any real purpose. I now know that amplifiers are much more complex than I originally anticipated. For example, capacitors must be used to ensure that clear signals make it from the input to the output of the amplifier.

I learned how to use the Latex text editor! This is something I've wanted to experiment with for quite some time but never took the plunge until now. There are many nuances to the text editor that I still do not fully understand. That being said, I can now say that I have used the software before.

I reinforced the lesson that projects should be tackled in small chunks. For example, I ensured each small portion of code for this project worked before moving on to the next portion of code. This ensured a smooth development process. I was able to solve issues in newly written code when they arose rather than digging through a large amount of previously written code.

I reinforce the lesson that projects do not always go as planned. I initially thought that the raw signal being outputted by the F411RE would be barely audible when put through a speaker. This caused me to add an amplifier circuit to the oscillator design requirements before starting the project. Contrary to my assumption, the unamplified signal is perfectly audible. Thus, the amplified signal is nearly too loud to be reasonably used.

I reinforce the lesson that chipping away at large projects over a period of time is the ideal way to tackle such projects. I wasted no time starting this project, and it paid off. I did not stress over this project, as I knew I had ample time to figure out any issues that I was having.

Appendix A

Source Code

A.1 main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "string.h"
#include "stdlib.h"
#include "stdbool.h"
#include "float.h"
#include "stdio.h"
#include "memory.h"
#include "stdint.h"
#include "stm32f4xx_hal_rcc.h"
#include "stm32f4xx_hal.h"
#include "stm32f4xx_hal_exti.h"
#include "stm32f4xx_hal_flash_ex.h"
#include "stm32f4xx_hal_flash.h"
#include "stm32f4xx_hal_gpio.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
```

```
#define FLASH_PROGRAM_BYTE  ((uint32_t)0x00)
#define FLASH_PROGRAM_HWORD ((uint32_t)0x01)
#define FLASH_PROGRAM_WORD  ((uint32_t)0x02)
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim3;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */
uint32_t timer;
uint32_t timerRaw;

//uint32_t i = 0;

char keyChar = 0;
uint16_t key = 0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */
uint16_t keyPadScan(void);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
```

```
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
//int state = 0;

HAL_TIM_Base_Start(&htim3);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

      timerRaw = __HAL_TIM_GET_COUNTER(&htim3);
      if(timerRaw != timer){
          timer++;
      }


//if((state == 0)&(timer <= 5000)){
//  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET);
//}else if((state == 0)&(timer >= 5000)){
//  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);
//}

if(keyChar != 0){
    //C4 1 261.63
    if(keyChar == 49){
        if(timer >= 2582){
          timer = 0;
        }
        if(timer <= 1241){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET);
        }else if(timer >= 1241){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);
        }
    }
    //D 2 293.66
    if(keyChar == 50){
        if(timer >= 2300){
          timer = 0;
        }
        if(timer <= 1150){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET);
        }else if(timer >= 1150){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);
        }
    }
    //E 3 329.63
    if(keyChar == 51){
        if(timer >= 2052){
          timer = 0;
        }
        if(timer <= 1026){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET);
        }else if(timer >= 1026){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);
        }
    }
    //F 4 349.23
    if(keyChar == 52){
        if(timer >= 1942){
          timer = 0;
        }
        if(timer <= 971){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET);
        }else if(timer >= 971){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);
```

```
        }
    }
    //G 5 391.00
    if(keyChar == 53){
        if(timer >= 1728){
          timer = 0;
        }
        if(timer <= 864){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET);
        }else if(timer >= 864){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);
        }
    }
    //A 6 440.00
    if(keyChar == 54){
        if(timer >= 1530){
          timer = 0;
        }
        if(timer <= 765){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET);
        }else if(timer >= 765){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);
        }
    }
    //B 7 493.88
    if(keyChar == 55){
        if(timer >= 1374){
          timer = 0;
        }
        if(timer <= 687){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET);
        }else if(timer >= 687){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);
        }
    }
    //C5 8 523.25
    if(keyChar == 56){
        if(timer >= 1288){
          timer = 0;
        }
        if(timer <= 644){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_SET);
        }else if(timer >= 644){
            HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);
        }
    }
}


    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
```

```
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 100;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 1;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 65535;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
```

```
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : PC5 PC6 PC7 PC8 */
  GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : PB1 PB13 PB14 PB15 */
  GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : PC12 */
  GPIO_InitStruct.Pin = GPIO_PIN_12;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}

/* USER CODE BEGIN 4 */
uint16_t keyPadScan(void){
    return 0;
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
```

```
455   */
456 void assert_failed(uint8_t *file, uint32_t line)
457 {
458   /* USER CODE BEGIN 6 */
459   /* User can add his own implementation to report the file name and line number,
460      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
461   /* USER CODE END 6 */
462 }
463 #endif /* USE_FULL_ASSERT */
464
465 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Listing A.1: main.c

A.2 Interrupt Code

```
1  /* USER CODE BEGIN Header */
2  /**
3    ******************************************************************************
4    * @file    stm32f4xx_it.c
5    * @brief   Interrupt Service Routines.
6    ******************************************************************************
7    * @attention
8    *
9    * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
10   * All rights reserved.</center></h2>
11   *
12   * This software component is licensed by ST under BSD 3-Clause license,
13   * the "License"; You may not use this file except in compliance with the
14   * License. You may obtain a copy of the License at:
15   *                        opensource.org/licenses/BSD-3-Clause
16   *
17   ******************************************************************************
18   */
19 /* USER CODE END Header */
20
21 /* Includes ------------------------------------------------------------------*/
22 #include "main.h"
23 #include "stm32f4xx_it.h"
24 /* Private includes ----------------------------------------------------------*/
25 /* USER CODE BEGIN Includes */
26 #include "stdbool.h"
27 #include "stdio.h"
28 #include "stm32f4xx_hal_gpio.h"
29 #include "memory.h"
30 /* USER CODE END Includes */
31
32 /* Private typedef -----------------------------------------------------------*/
33 /* USER CODE BEGIN TD */
34
35 /* USER CODE END TD */
36
37 /* Private define ------------------------------------------------------------*/
38 /* USER CODE BEGIN PD */
39 #define FLASH_PROGRAM_BYTE  ((uint32_t)0x00)
40 #define FLASH_PROGRAM_HWORD ((uint32_t)0x01)
41 #define FLASH_PROGRAM_WORD  ((uint32_t)0x02)
42 /* USER CODE END PD */
43
44 /* Private macro -------------------------------------------------------------*/
45 /* USER CODE BEGIN PM */
46
47 /* USER CODE END PM */
48
49 /* Private variables ---------------------------------------------------------*/
50 /* USER CODE BEGIN PV */
51 extern uint32_t timer;
```

```
52
53 extern char keyChar;
54 extern uint16_t key;
55
56 ///Pinout, looking at back of keypad, left to right
57 /// PC8, PC7, PC6, PC5, PB15, PB14, PB13, PB1
58
59 const GPIO_TypeDef* _KEYPAD_COLUMN_GPIO_PORT[] =
60 {
61   GPIOB,
62   GPIOB,
63   GPIOB,
64   GPIOB
65 };
66
67 const uint16_t _KEYPAD_COLUMN_GPIO_PIN[] =
68 {
69   GPIO_PIN_1,
70   GPIO_PIN_13,
71   GPIO_PIN_14,
72   GPIO_PIN_15
73 };
74
75 const GPIO_TypeDef* _KEYPAD_ROW_GPIO_PORT[] =
76 {
77   GPIOC,
78   GPIOC,
79   GPIOC,
80   GPIOC
81 };
82
83 const uint16_t _KEYPAD_ROW_GPIO_PIN[] =
84 {
85   GPIO_PIN_5,
86   GPIO_PIN_6,
87   GPIO_PIN_7,
88   GPIO_PIN_8
89 };
90 /* USER CODE END PV */
91
92 /* Private function prototypes -----------------------------------------------*/
93 /* USER CODE BEGIN PFP */
94 char KeyPadGetChar(uint16_t key);
95 /* USER CODE END PFP */
96
97 /* Private user code ---------------------------------------------------------*/
98 /* USER CODE BEGIN 0 */
99
100 /* USER CODE END 0 */
101
102 /* External variables --------------------------------------------------------*/
103 extern TIM_HandleTypeDef htim3;
104 extern UART_HandleTypeDef huart2;
105 /* USER CODE BEGIN EV */
106
107 /* USER CODE END EV */
108
109 /******************************************************************************/
110 /*           Cortex-M4 Processor Interruption and Exception Handlers          */
111 /******************************************************************************/
112 /**
113   * @brief This function handles Non maskable interrupt.
114   */
115 void NMI_Handler(void)
116 {
117   /* USER CODE BEGIN NonMaskableInt_IRQn 0 */
118
119   /* USER CODE END NonMaskableInt_IRQn 0 */
```

```
120   /* USER CODE BEGIN NonMaskableInt_IRQn 1 */
121   while (1)
122   {
123   }
124   /* USER CODE END NonMaskableInt_IRQn 1 */
125 }
126
127 /**
128   * @brief This function handles Hard fault interrupt.
129   */
130 void HardFault_Handler(void)
131 {
132   /* USER CODE BEGIN HardFault_IRQn 0 */
133
134   /* USER CODE END HardFault_IRQn 0 */
135   while (1)
136   {
137     /* USER CODE BEGIN W1_HardFault_IRQn 0 */
138     /* USER CODE END W1_HardFault_IRQn 0 */
139   }
140 }
141
142 /**
143   * @brief This function handles Memory management fault.
144   */
145 void MemManage_Handler(void)
146 {
147   /* USER CODE BEGIN MemoryManagement_IRQn 0 */
148
149   /* USER CODE END MemoryManagement_IRQn 0 */
150   while (1)
151   {
152     /* USER CODE BEGIN W1_MemoryManagement_IRQn 0 */
153     /* USER CODE END W1_MemoryManagement_IRQn 0 */
154   }
155 }
156
157 /**
158   * @brief This function handles Pre-fetch fault, memory access fault.
159   */
160 void BusFault_Handler(void)
161 {
162   /* USER CODE BEGIN BusFault_IRQn 0 */
163
164   /* USER CODE END BusFault_IRQn 0 */
165   while (1)
166   {
167     /* USER CODE BEGIN W1_BusFault_IRQn 0 */
168     /* USER CODE END W1_BusFault_IRQn 0 */
169   }
170 }
171
172 /**
173   * @brief This function handles Undefined instruction or illegal state.
174   */
175 void UsageFault_Handler(void)
176 {
177   /* USER CODE BEGIN UsageFault_IRQn 0 */
178
179   /* USER CODE END UsageFault_IRQn 0 */
180   while (1)
181   {
182     /* USER CODE BEGIN W1_UsageFault_IRQn 0 */
183     /* USER CODE END W1_UsageFault_IRQn 0 */
184   }
185 }
186
187 /**
```

```
188   * @brief This function handles System service call via SWI instruction.
189   */
190 void SVC_Handler(void)
191 {
192   /* USER CODE BEGIN SVCall_IRQn 0 */
193 
194   /* USER CODE END SVCall_IRQn 0 */
195   /* USER CODE BEGIN SVCall_IRQn 1 */
196 
197   /* USER CODE END SVCall_IRQn 1 */
198 }
199 
200 /**
201   * @brief This function handles Debug monitor.
202   */
203 void DebugMon_Handler(void)
204 {
205   /* USER CODE BEGIN DebugMonitor_IRQn 0 */
206 
207   /* USER CODE END DebugMonitor_IRQn 0 */
208   /* USER CODE BEGIN DebugMonitor_IRQn 1 */
209 
210   /* USER CODE END DebugMonitor_IRQn 1 */
211 }
212 
213 /**
214   * @brief This function handles Pendable request for system service.
215   */
216 void PendSV_Handler(void)
217 {
218   /* USER CODE BEGIN PendSV_IRQn 0 */
219 
220   /* USER CODE END PendSV_IRQn 0 */
221   /* USER CODE BEGIN PendSV_IRQn 1 */
222 
223   /* USER CODE END PendSV_IRQn 1 */
224 }
225 
226 /**
227   * @brief This function handles System tick timer.
228   */
229 void SysTick_Handler(void)
230 {
231   /* USER CODE BEGIN SysTick_IRQn 0 */
232 
233 
234   /* USER CODE END SysTick_IRQn 0 */
235   HAL_IncTick();
236   /* USER CODE BEGIN SysTick_IRQn 1 */
237 //    timer++;
238 
239   /* USER CODE END SysTick_IRQn 1 */
240 }
241 
242 /******************************************************************************/
243 /* STM32F4xx Peripheral Interrupt Handlers                                    */
244 /* Add here the Interrupt Handlers for the used peripherals.                  */
245 /* For the available peripheral interrupt handler names,                      */
246 /* please refer to the startup file (startup_stm32f4xx.s).                    */
247 /******************************************************************************/
248 
249 /**
250   * @brief This function handles EXTI line[9:5] interrupts.
251   */
252 void EXTI9_5_IRQHandler(void)
253 {
254   /* USER CODE BEGIN EXTI9_5_IRQn 0 */
255     key = 0;
```

```
256     for(uint8_t c=0 ; c<4 ; c++){
257         for(uint8_t i=0 ; i<4 ; i++){
258             HAL_GPIO_WritePin((GPIO_TypeDef*)_KEYPAD_COLUMN_GPIO_PORT[i],
    _KEYPAD_COLUMN_GPIO_PIN[i], GPIO_PIN_SET);
259         }
260         HAL_GPIO_WritePin((GPIO_TypeDef*)_KEYPAD_COLUMN_GPIO_PORT[c],
    _KEYPAD_COLUMN_GPIO_PIN[c], GPIO_PIN_RESET);
261         for(uint8_t r=0 ; r<4 ; r++){
262             if(HAL_GPIO_ReadPin((GPIO_TypeDef*)_KEYPAD_ROW_GPIO_PORT[r],
    _KEYPAD_ROW_GPIO_PIN[r]) == GPIO_PIN_RESET){
263                 if(HAL_GPIO_ReadPin((GPIO_TypeDef*)_KEYPAD_ROW_GPIO_PORT[r],
    _KEYPAD_ROW_GPIO_PIN[r]) == GPIO_PIN_RESET){
264                     key |= 1<<c;
265                     key |= 1<<(r+8);
266                     while(HAL_GPIO_ReadPin((GPIO_TypeDef*)_KEYPAD_ROW_GPIO_PORT[r],
    _KEYPAD_ROW_GPIO_PIN[r]) == GPIO_PIN_RESET){
267
268                     }
269                 }
270             }
271         }
272     }
273
274     keyChar = KeyPadGetChar(key);
275     key = 0;
276
277     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15, GPIO_PIN_RESET)
    ;
278   /* USER CODE END EXTI9_5_IRQn 0 */
279   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_5);
280   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_6);
281   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_7);
282   HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_8);
283   /* USER CODE BEGIN EXTI9_5_IRQn 1 */
284
285   /* USER CODE END EXTI9_5_IRQn 1 */
286 }
287
288 /**
289   * @brief This function handles TIM3 global interrupt.
290   */
291 void TIM3_IRQHandler(void)
292 {
293   /* USER CODE BEGIN TIM3_IRQn 0 */
294
295   /* USER CODE END TIM3_IRQn 0 */
296   HAL_TIM_IRQHandler(&htim3);
297   /* USER CODE BEGIN TIM3_IRQn 1 */
298
299   /* USER CODE END TIM3_IRQn 1 */
300 }
301
302 /**
303   * @brief This function handles USART2 global interrupt.
304   */
305 void USART2_IRQHandler(void)
306 {
307   /* USER CODE BEGIN USART2_IRQn 0 */
308
309   /* USER CODE END USART2_IRQn 0 */
310   HAL_UART_IRQHandler(&huart2);
311   /* USER CODE BEGIN USART2_IRQn 1 */
312
313   /* USER CODE END USART2_IRQn 1 */
314 }
315
316 /**
317   * @brief This function handles EXTI line[15:10] interrupts.
```

```
  */
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  /* USER CODE END EXTI15_10_IRQn 1 */
}

/* USER CODE BEGIN 1 */
char KeyPadGetChar(uint16_t key)
{
  switch(key)
  {
    case 0x0000:
      return 0;
        case 0x0101:
            return '1';
        case 0x0201:
            return '2';
        case 0x0401:
            return '3';
        case 0x0801:
            return 'A';
        case 0x0102:
            return '4';
        case 0x0202:
            return '5';
        case 0x0402:
            return '6';
        case 0x0802:
            return 'B';
        case 0x0104:
            return '7';
        case 0x0204:
            return '8';
        case 0x0404:
            return '9';
        case 0x0804:
            return 'C';
        case 0x0108:
            return '.';
        case 0x0208:
            return '0';
        case 0x0408:
            return '#';
        case 0x0808:
            return 'D';

        default:
            return 0;
    }
}
/* USER CODE END 1 */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Listing A.2: Interrupt Code

Appendix B

Bill of Materials

- STM32 NUCLEO-F411RE
- LM386-4 Amplifier
- Matrix Keypad
- Two 100µF (25V) Capacitors
- One 100nF (25V) Capacitor
- One 1mF (25V) Capacitor
- Magnet and Coil Speaker
    1. Optional: Enclosure for Speaker
- Connecting Wires
- USB to Micro USB Cable

Bibliography

YouTube, 17-Jan-2017. [Online]. Available: https://www.youtube.com/watch?v=4ObzEft2R_g. [Accessed: 23-Nov-2021].