GEBZE TECHNICAL UNIVERSITY

ELECTRONICS ENGINEERING

ELEC 334

Microprocessors Lab

LAB #6

Prepared by
1) 1801022037 Ömer Emre POLAT

1. Introduction

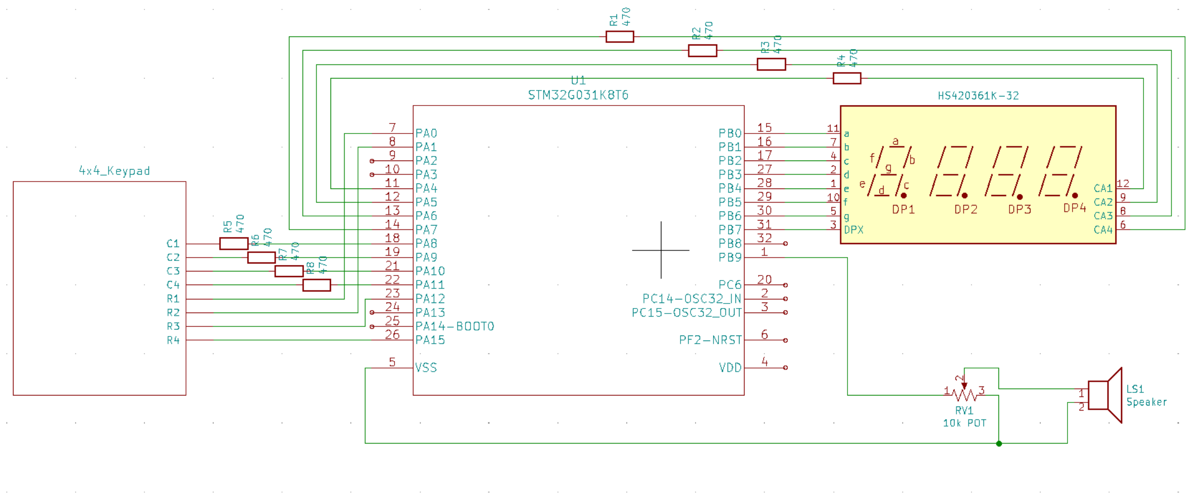
In this lab for the first problem we will create a tone generator using a keypad seven segment display and a speaker. For the second problem we will create a working 6DOF sensor that will communicate with the board using I2C and communicate with the PC using UART.

2. Problem 1 Design

2.1. Flowchart and Block Diagram

First, we will design the board connections and the flowchart for our code for ease of design.

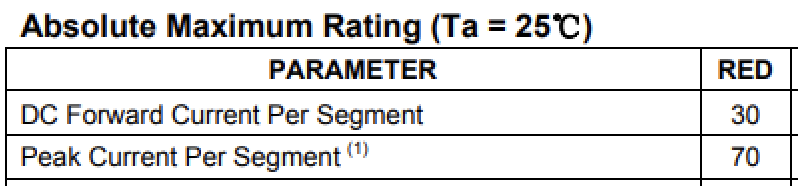
The board connections will be drawn on KiCAD using the model libraries of the components



All the resistors seen in the figure are 470 ohm resistors. There are single resistor sets going to the both keypad and the seven segment display because the lines that do not have a resistor will be connected in series to ones that has a resistor so there is no need to use two sets of resistors.

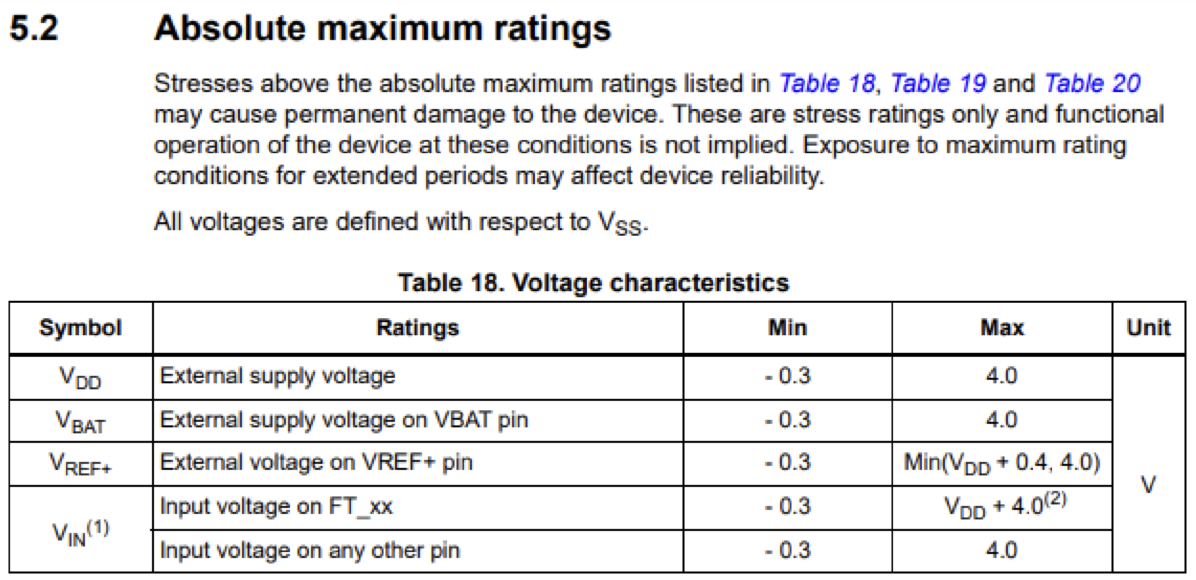
There wasn’t a need to use an amplifier because there is no bandpass filter decreasing the output voltage and the output voltage is 3.3V PWM. This 3.3V PWM signal is lowered into around 1.5-2V using a voltage divider.

The resistance values are calculated using the datasheet of the STM32 board and the HS420361K-32.



Absolute Maximum Rating (Ta = 25℃)

PARAMETER	RED
DC Forward Current Per Segment	30
Peak Current Per Segment [1]	70



5.2 Absolute maximum ratings

Stresses above the absolute maximum ratings listed in *Table 18*, *Table 19* and *Table 20* may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

All voltages are defined with respect to V_{SS}.

Table 18. Voltage characteristics

Symbol	Ratings	Min	Max	Unit
V_{DD}	External supply voltage	- 0.3	4.0	V
V_{BAT}	External supply voltage on VBAT pin	- 0.3	4.0	
V_{REF+}	External voltage on VREF+ pin	- 0.3	Min(V_{DD} + 0.4, 4.0)	
V_{IN}[1]	Input voltage on FT_xx	- 0.3	V_{DD} + 4.0[2]	
	Input voltage on any other pin	- 0.3	4.0	

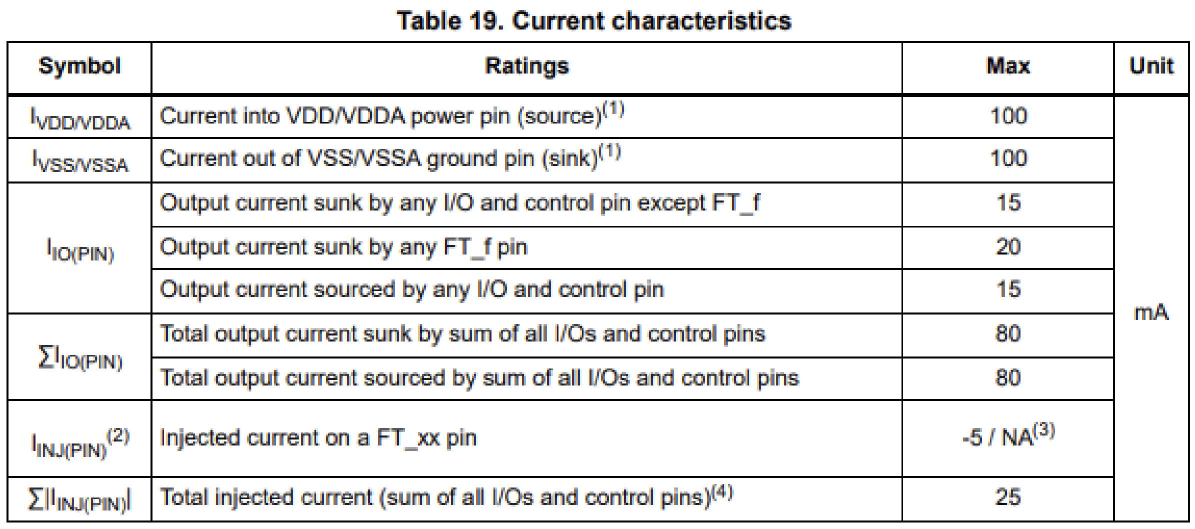


Table 19. Current characteristics

Symbol	Ratings	Max	Unit
$I_{VDD/VDDA}$	Current into VDD/VDDA power pin (source)[1]	100	mA
$I_{VSS/VSSA}$	Current out of VSS/VSSA ground pin (sink)[1]	100	
$I_{IO(PIN)}$	Output current sunk by any I/O and control pin except FT_f	15	
	Output current sunk by any FT_f pin	20	
	Output current sourced by any I/O and control pin	15	
$\Sigma I_{IO(PIN)}$	Total output current sunk by sum of all I/Os and control pins	80	
	Total output current sourced by sum of all I/Os and control pins	80	
$I_{INJ(PIN)}$[2]	Injected current on a FT_xx pin	-5 / NA[3]	
$\Sigma\|I_{INJ(PIN)}\|$	Total injected current (sum of all I/Os and control pins)[4]	25	

$$I_{keypad,ssd} = \frac{V}{R_{total}} = \frac{3.33V}{470} = 7.08mA < 15mA \,\&\, 30mA$$

Now that the absolute maximum DC forward current values are adjusted we can layout the pins.

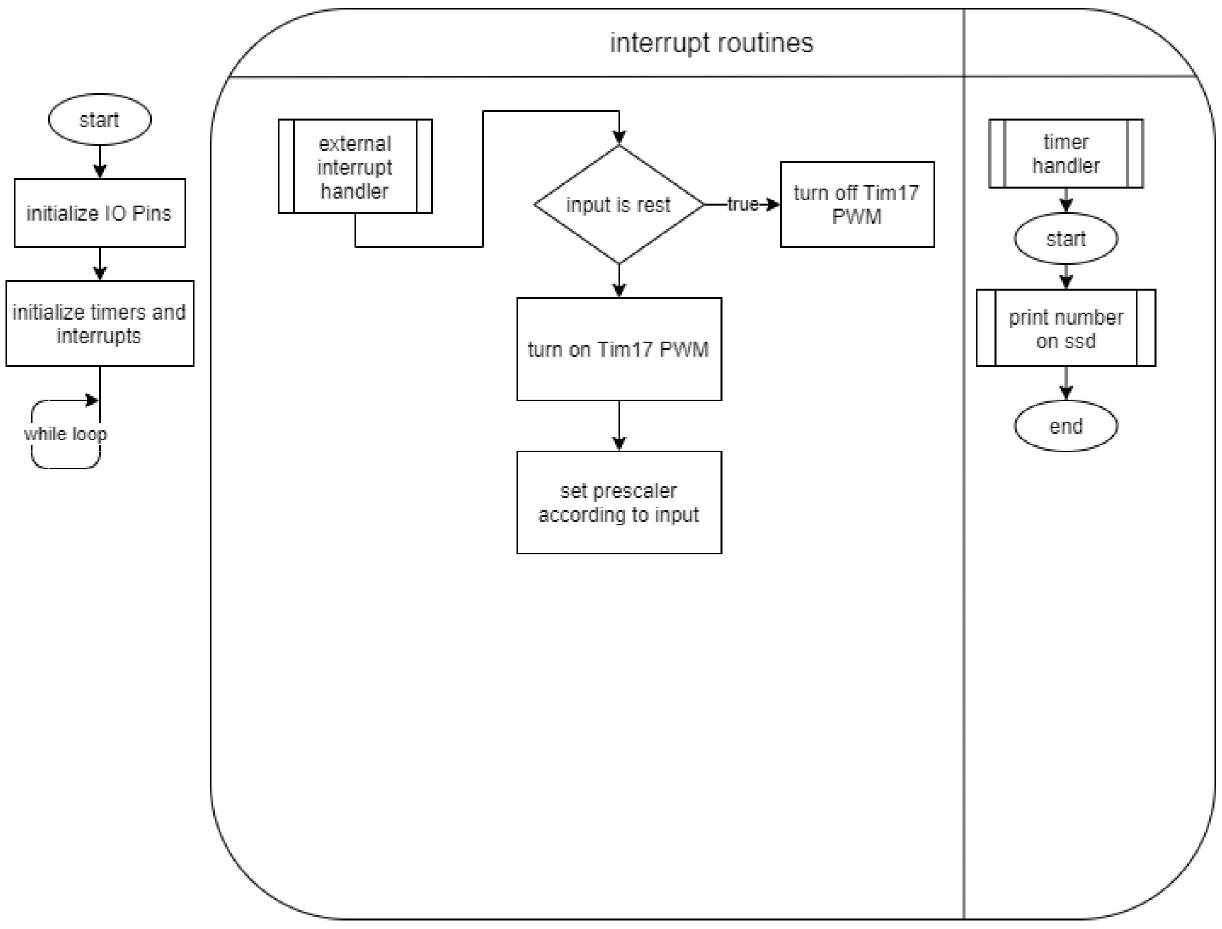
The pin connections are as follows:

PB0 -> A (SSD)   PA0 -> R1 (KP)

PB1 -> B (SSD)	PA1 -> R2 (KP)
....	PA8 -> C1 (KP)
PB6 -> G (SSD)	PA9 -> C2 (KP)
PB7 -> DPX (SSD)	PA10 -> C3 (KP)
PA4 -> D1 (SSD)	PA11 -> C4 (KP)
PA5 -> D2 (SSD)	PA12 -> R3 (KP)
PA6 -> D3 (SSD)	PA15 -> R4 (KP)
PA7 -> D4 (SSD)	PB9 -> POT_1 (POT)

Now that the pins are chosen we can start designing the flowchart and the code.

Code will first set the peripherals and pin input output modes. Then the code will store a number and the keypad input in a global variable. These variables will be modified inside the keypad interrupt and using the input numbers the TIM17 prescaler value will be set to the correct value for the chosen frequency. At last a chosen button input will turn off the PW output.



Now that the flowchart is drawn the full code can be written.

2.2. Keypad Interrupt Handler and TIM17 AF2 PWM Setup

Keypad interrupt handler works by getting the input from kp_get_input function written on project 2. When the input is set to the global variable, interrupt handler will have a switch case which sets the prescaler to the correct value in the TIM17.

```
/* PA8-PA11 */
void EXTI4_15_IRQHandler(void)
{
	kp_get_input(); /* get the input and write it to global input */

	switch(kp_input)
	{
		case '1':
			TIM17->BDTR |= (1U << 15);
			number_1 = 440; /* A4 440Hz */
			break;
		case '2':
			TIM17->BDTR |= (1U << 15);
			number_1 = 494; /* B4 494Hz */
			break;
		case '3':
			TIM17->BDTR |= (1U << 15);
			number_1 = 523; /* C5 523Hz */
			break;
		case '4':
			TIM17->BDTR |= (1U << 15);
			number_1 = 587; /* D5 587Hz */
			break;
		case '5':
			TIM17->BDTR |= (1U << 15);
			number_1 = 659; /* E5 659Hz */
			break;
		case '6':
			TIM17->BDTR |= (1U << 15);
			number_1 = 698; /* F5 698Hz */
			break;
		case '7':
			TIM17->BDTR |= (1U << 15);
			number_1 = 784; /* G5 784Hz */
			break;
		case '8':
			TIM17->BDTR |= (1U << 15);
			number_1 = 880; /* A5 880Hz */
			break;
		case '9':
			TIM17->BDTR |= (1U << 15);
			number_1 = 987; /* B5 987Hz */
			break;
		case '0':
			TIM17->BDTR |= (1U << 15);
			number_1 = 1047; /* C6 1047Hz */
			break;
		case 'A':
			TIM17->BDTR |= (1U << 15);
			number_1 = 1175; /* D6 1175Hz */
```

```
            break;
        case 'B':
            TIM17->BDTR |= (1U << 15);
            number_1 = 1319; /* E6 1319Hz */
            break;
        case 'C':
            TIM17->BDTR |= (1U << 15);
            number_1 = 1397; /* F6 1397Hz */
            break;
        case 'D':
            TIM17->BDTR |= (1U << 15);
            number_1 = 1567; /* G6 1567Hz */
            break;
        case 'F':
            TIM17->BDTR |= (1U << 15);
            number_1 = 1760; /* A6 1760Hz */
            break;
        case 'E':
            TIM17->BDTR &= ~(1U << 15);
            break;
        default:

            break;
    }

    /* set the prescaler for the number_1 frequency */
    TIM17->PSC = ((SystemCoreClock / (long unsigned int) number_1) - 1);

    for(uint32_t i=0; i<400000; i++);

    EXTI->RPR1 |= (15U << 8); /* reset interrupt pending */
}
```

Setting the prescaler is done like this because the auto reload register is set to 1 and the duty cycle is set to %50. The frequencies are taken from the graph below. A 2 octave C-Major scale is used for the keypad inputs. The buttons are set irregularly.

Key number	Helmholtz name	Scientific name	Frequency (Hz) (Equal temperament)
73	a‴	A_6	**1760.000**
72	g♯‴/a♭‴	$G♯_6/A♭_6$	1661.219
71	g‴	G_6	1567.982

Key number	Helmholtz name	Scientific name	Frequency (Hz) (Equal temperament)
70	f♯‴/g♭‴	F♯$_6$/G♭$_6$	1479.978
69	f‴	F$_6$	1396.913
68	e‴	E$_6$	1318.510
67	d♯‴/e♭‴	D♯$_6$/E♭$_6$	1244.508
66	d‴	D$_6$	1174.659
65	c♯‴/d♭‴	C♯$_6$/D♭$_6$	1108.731
64	c‴ 3-line octave	C$_6$ Soprano C (High C)	1046.502
63	b″	B$_5$	987.7666
62	a♯″/b♭″	A♯$_5$/B♭$_5$	932.3275
61	a″	A$_5$	**880.0000**
60	g♯″/a♭″	G♯$_5$/A♭$_5$	830.6094
59	g″	G$_5$	783.9909
58	f♯″/g♭″	F♯$_5$/G♭$_5$	739.9888

Key number	Helmholtz name	Scientific name	Frequency (Hz) (Equal temperament)
57	f″	F_5	698.4565
56	e″	E_5	659.2551
55	d♯″/e♭″	$D\sharp_5/E\flat_5$	622.2540
54	d″	D_5	587.3295
53	c♯″/d♭″	$C\sharp_5/D\flat_5$	554.3653
52	c″ 2-line octave	C_5 Tenor C	523.2511
51	b′	B_4	493.8833
50	a♯′/b♭′	$A\sharp_4/B\flat_4$	466.1638
49	a′	A_4 A440	**440.0000**

Now for the setup of the timer 17 PWM.

```
/* pwm timer 17 AF2 setup */

    //PB9 as AF mode
       GPIOB->MODER &= ~(3U << 2*9);
       GPIOB->MODER |= (2U << 2*9);

       //Choose AF2 from mux
       GPIOB->AFR[1] &= ~(0xFU << 4*1);
       GPIOB->AFR[1] |= (0x2U << 4*1);

       RCC->APBENR2 |= (1U << 18); /* Enable TIM17 clock */

       TIM17->CR1 = 0U; /* resetting control register */
       TIM17->CR1 |= (1U << 7); /* ARPE buffering */
       TIM17->CNT = 0U; /* reset the timer counter */
```

```
      TIM17->PSC = ((SystemCoreClock/RefreshRate) - 1); /* prescaler set to 1600-
1 */
      TIM17->ARR = 1U; /* set the autoreload register for 1 milliseconds */

      TIM17->CR1 |= (1U << 0); /* Enable TIM1 */

      TIM17->CCMR1 |= (6U << 4);

      TIM17->CCMR1 |= (1U << 3);

      TIM17->CCER |= (1U << 0);

      TIM17->CCR1 |= (1U << 0);
```

The timer 17 is set using the same method used as any other timer but the PWM is enabled after setting the CCMR1 (capture/compare mode register), CCER (capture/compare enable register), CCR1 (capture/compare 1 register) and BDTR (break and dead-time register). BDTR is used only for the MOE bit which is main output enable for the TIM 17 PWM. This step only exists in timer 16 and 17.

2.3. Combined Code

After the extra functions are written the combined code can be written with the preceding project 2 codes. The code is written with comments that explain the lines so reading through the code can help to understand.

main.c

```
/* PROBLEM1.C                 */
/* Author: Ömer Emre Polat */
/* Student No: 1801022037   */

#include "bsp.h"
#include "stdlib.h"

int main(void) {

      dp_pos = none;
      BSP_system_init();

    while(1)
    {

    }

    return 0;
}
```

bsp.c

```
/* BSP.C                      */
/* Author: Ömer Emre Polat */
/* Student No: 1801022037   */
```

```c
#include "bsp.h"
#include <math.h>

void TIM1_BRK_UP_TRG_COM_IRQHandler(void)
{
	TIM1->SR &= ~(1U << 0); /* reset status register */
	ssd_print();
}

/* PA8-PA11 */
void EXTI4_15_IRQHandler(void)
{
	kp_get_input(); /* get the input and write it to global input */

	switch(kp_input)
	{
		case '1':
			TIM17->BDTR |= (1U << 15);
			number_1 = 440; /* A4 440Hz */
			break;
		case '2':
			TIM17->BDTR |= (1U << 15);
			number_1 = 494; /* B4 494Hz */
			break;
		case '3':
			TIM17->BDTR |= (1U << 15);
			number_1 = 523; /* C5 523Hz */
			break;
		case '4':
			TIM17->BDTR |= (1U << 15);
			number_1 = 587; /* D5 587Hz */
			break;
		case '5':
			TIM17->BDTR |= (1U << 15);
			number_1 = 659; /* E5 659Hz */
			break;
		case '6':
			TIM17->BDTR |= (1U << 15);
			number_1 = 698; /* F5 698Hz */
			break;
		case '7':
			TIM17->BDTR |= (1U << 15);
			number_1 = 784; /* G5 784Hz */
			break;
		case '8':
			TIM17->BDTR |= (1U << 15);
			number_1 = 880; /* A5 880Hz */
			break;
		case '9':
			TIM17->BDTR |= (1U << 15);
			number_1 = 987; /* B5 987Hz */
			break;
		case '0':
			TIM17->BDTR |= (1U << 15);
			number_1 = 1047; /* C6 1047Hz */
			break;
		case 'A':
			TIM17->BDTR |= (1U << 15);
```

```c
                    number_1 = 1175; /* D6 1175Hz */
                    break;
            case 'B':
                    TIM17->BDTR |= (1U << 15);
                    number_1 = 1319; /* E6 1319Hz */
                    break;
            case 'C':
                    TIM17->BDTR |= (1U << 15);
                    number_1 = 1397; /* F6 1397Hz */
                    break;
            case 'D':
                    TIM17->BDTR |= (1U << 15);
                    number_1 = 1567; /* G6 1567Hz */
                    break;
            case 'F':
                    TIM17->BDTR |= (1U << 15);
                    number_1 = 1760; /* A6 1760Hz */
                    break;
            case 'E':
                    TIM17->BDTR &= ~(1U << 15);
                    break;
            default:

                    break;
        }

        /* set the prescaler for the number_1 frequency */
        TIM17->PSC = ((SystemCoreClock / (long unsigned int) number_1) - 1);

        for(uint32_t i=0; i<333333; i++);

        EXTI->RPR1 |= (15U << 8); /* reset interrupt pending */
}

/*//////////////KEYPAD FUNCTIONS//////////////*/

void kp_get_input(void)
{
        uint8_t c = 0;
        uint8_t r = 0;

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

        if((EXTI->RPR1 >> 8) & (1U))
        {
                c = 0;
        }
        else if ((EXTI->RPR1 >> 9) & (1U))
        {
                c = 1;
        }
        else if ((EXTI->RPR1 >> 10) & (1U))
        {
                c = 2;
        }
        else if ((EXTI->RPR1 >> 11) & (1U))
        {
                c = 3;
        }

        GPIOA->ODR &= ~(1U << 0); /* shut off A0 */
        for(uint32_t i=0; i<ButtonBounceTime; i++);
        if(((GPIOA->IDR >> (8+c)) & (1U)) ^ 1U) /* check if input is not there */
        {
                r = 0;
        }
        GPIOA->ODR |= (1U << 0); /* turn on A0 */

        GPIOA->ODR &= ~(1U << 1); /* shut off A1 */
        for(uint32_t i=0; i<ButtonBounceTime; i++);
        if(((GPIOA->IDR >> (8+c)) & (1U)) ^ 1U) /* check if input is not there */
        {
                r = 1;
        }
        GPIOA->ODR |= (1U << 1); /* turn on A1 */

        GPIOA->ODR &= ~(1U << 12); /* shut off A12 */
        for(uint32_t i=0; i<ButtonBounceTime; i++);
        if(((GPIOA->IDR >> (8+c)) & (1U)) ^ 1U) /* check if input is not there */
        {
                r = 2;
        }
        GPIOA->ODR |= (1U << 12); /* turn on A12 */

        GPIOA->ODR &= ~(1U << 15); /* shut off A15 */
        for(uint32_t i=0; i<ButtonBounceTime; i++);
        if(((GPIOA->IDR >> (8+c)) & (1U)) ^ 1U) /* check if input is not there */
        {
                r = 3;
        }
        GPIOA->ODR |= (1U << 15); /* turn on A15 */

        kp_input = kp_inputs[(c + (4 * r))];
}

/*///////////////BSP FUNCTIONS///////////////*/

void BSP_IWDG_init(void)
{
    IWDG->KR = 0x5555;
    IWDG->PR = 1; // prescaler
    while(IWDG->SR & 0x1); // wait while status update
    IWDG->KR = 0xCCCC;
}

void BSP_IWDG_refresh(void)
{
    IWDG->KR = 0xAAAA;
}

void BSP_system_init()
{
    __disable_irq();
    /* input and output pins setup */

    GPIOB->MODER |= (0x5555U); /* Setup (B0, B7) as output */
    GPIOB->MODER &= ~(0xAAAAU); /* Outputs for seven segments */

    GPIOA->MODER |= (0x55U << 2*4); /* Setup (A4, A7) as output */
    GPIOA->MODER &= ~(0xAAU << 2*4); /* Outputs for digit selections */

    GPIOA->MODER &= ~(255U << 2*8); /* Setup (A8, A11) as input */

    GPIOA->PUPDR &= ~(3U << 2*8); /* A8 Pull down */
    GPIOA->PUPDR |= (2U << 2*8);

    GPIOA->PUPDR &= ~(3U << 2*9); /* A9 Pull down */
    GPIOA->PUPDR |= (2U << 2*9);

    GPIOA->PUPDR &= ~(3U << 2*10); /* A10 Pull down */
    GPIOA->PUPDR |= (2U << 2*10);

    GPIOA->PUPDR &= ~(3U << 2*11); /* A11 Pull down */
    GPIOA->PUPDR |= (2U << 2*11);

    GPIOA->MODER &= ~(3U << 2*0); /* A0 output */
    GPIOA->MODER |= (1U << 2*0);
    GPIOA->ODR |= (1U << 0); /* set A0 to high */

    GPIOA->MODER &= ~(3U << 2*1); /* A1 output */
    GPIOA->MODER |= (1U << 2*1);
    GPIOA->ODR |= (1U << 1); /* set A1 to high */

    GPIOA->MODER &= ~(3U << 2*12); /* A12 output */
    GPIOA->MODER |= (1U << 2*12);
    GPIOA->ODR |= (1U << 12); /* set A12 to high */

    GPIOA->MODER &= ~(3U << 2*15); /* A15 output */
    GPIOA->MODER |= (1U << 2*15);
    GPIOA->ODR |= (1U << 15); /* set A15 to high */

    /* timer and interrupt setup */

      RCC->APBENR2 |= (1U << 11); /* Enable TIM1 clock */

      TIM1->CR1 = 0; /* resetting control register */
      TIM1->CR1 |= (1U << 7); /* ARPE buffering */
      TIM1->CNT = 0; /* reset the timer counter */

      TIM1->PSC = ((SystemCoreClock/RefreshRate) - 1); /* prescaler set to 1600-1
*/
      TIM1->ARR = 1; /* set the autoreload register for 1 milliseconds */

      TIM1->DIER |= (1U << 0); /* update interrupt enable */
      TIM1->CR1 |= (1U << 0); /* Enable TIM1 */

      NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 0U); /* Setting priority for
TIM1 */
      NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn); /* Enabling TIM1 */

    /* pwm timer 17 AF2 setup */
    //PB9 as AF mode
      GPIOB->MODER &= ~(3U << 2*9);
      GPIOB->MODER |= (2U << 2*9);

      //Choose AF2 from mux
      GPIOB->AFR[1] &= ~(0xFU << 4*1);
      GPIOB->AFR[1] |= (0x2U << 4*1);

      RCC->APBENR2 |= (1U << 18); /* Enable TIM17 clock */

      TIM17->CR1 = 0U; /* resetting control register */
      TIM17->CR1 |= (1U << 7); /* ARPE buffering */
      TIM17->CNT = 0U; /* reset the timer counter */

      TIM17->PSC = ((SystemCoreClock/RefreshRate) - 1); /* prescaler set to 1600-
1 */
      TIM17->ARR = 1U; /* set the autoreload register for 1 milliseconds */

      TIM17->CR1 |= (1U << 0); /* Enable TIM1 */

      TIM17->CCMR1 |= (6U << 4);

      TIM17->CCMR1 |= (1U << 3);

      TIM17->CCER |= (1U << 0);

      TIM17->CCR1 |= (1U << 0);

    /* interrupt setup */

    RCC->IOPENR |= (3U << 0);

    EXTI->RTSR1 |= (15U << 8); /* Rising edge selection */
    EXTI->EXTICR[2] &= ~(1U << 8*0); /* 1U to select A8 from mux */
    EXTI->EXTICR[2] &= ~(1U << 8*1); /* 1U to select A9 from mux */
    EXTI->EXTICR[2] &= ~(1U << 8*2); /* 1U to select A10 from mux */
    EXTI->EXTICR[2] &= ~(1U << 8*3); /* 1U to select A11 from mux */
    EXTI->IMR1 |= (15U << 8); /* interrupt mask register */

    NVIC_SetPriority(EXTI4_15_IRQn, 1U); /* Setting priority for EXTI0_1 */
    NVIC_EnableIRQ(EXTI4_15_IRQn); /* Enabling EXTI0_1 */

    //BSP_IWDG_init(); // Watchdog init

    TIM1->SR  =  0x00000000U; /* reset status register */

    EXTI->RPR1 = 0x00000000U; /* reset interrupt pending */


    __enable_irq();
}

/*///////////////SSD FUNCTIONS///////////////*/

void ssd_set_clear(void)
{
      GPIOB->ODR |= (0x7FU);
}
```

```c
void ssd_set(volatile uint32_t digit)
{
	switch(digit)
	{
		case 0:
			GPIOB->ODR &= (0xC0U);
			break;
		case 1:
			GPIOB->ODR &= (0xF9U);
			break;
		case 2:
			GPIOB->ODR &= (0xA4U);
			break;
		case 3:
			GPIOB->ODR &= (0xB0U);
			break;
		case 4:
			GPIOB->ODR &= (0x99U);
			break;
		case 5:
			GPIOB->ODR &= (0x92U);
			break;
		case 6:
			GPIOB->ODR &= (0x82U);
			break;
		case 7:
			GPIOB->ODR &= (0xF8U);
			break;
		case 8:
			GPIOB->ODR &= (0x80U);
			break;
		case 9:
			GPIOB->ODR &= (0x90U);
			break;
	}
}

void ssd_set_overflow(void)
{
	ssd_digit(0);
	GPIOB->ODR &= (0xC0U); /* O */
	ssd_digit(1);
	GPIOB->ODR &= (0xE3U); /* v */
	ssd_digit(2);
	GPIOB->ODR &= (0x8EU); /* F */
	ssd_digit(3);
	GPIOB->ODR &= (0xC7U); /* L */
	ssd_digit_clear();
}

void ssd_digit_clear(void)
{
	GPIOA->ODR &= ~(15U << 4); /* digit clear */
}

void ssd_digit(volatile uint32_t digit) /* 0 = Digit1 */
{
	ssd_digit_clear();
	ssd_set_clear();
	if((uint32_t) dp_pos == (digit + 1))
	{
		GPIOB->ODR &= ~(0x80U);
	}
	else
	{
		GPIOB->ODR |= (0x80U);
	}

	GPIOA->ODR |= (1U << (4 + digit));
}

/* print and refresh functions */

void ssd_print(void)
{
	uint32_t digits[4] = {0};
	int number; /* copy of the number */

	if((number_1 < -999))
	{
		dp_pos = none;
		ssd_set_overflow();
		return;
	}
	if((number_1 > 9999))
	{
		dp_pos = none;
		ssd_set_overflow();
		return;
	}

	if(number_1 < 0) /* negative case */
	{
		if((-1 * number_1) < 10) /* float printing */
		{
			dp_pos = second;
			number = (int) (100 * number_1);
		}
		else if((-1 * number_1) < 100)
		{
			dp_pos = third;
			number = (int) (10 * number_1);
		}
		else if((-1 * number_1) < 1000)
		{
			dp_pos = fourth;
			number = (int) number_1;
		}
		else
		{
			dp_pos = none;
		}
	}
	else /* positive case */
	{
		if(number_1 < 10) /* float printing */
		{
```

```
                    dp_pos = first;
                    number = (int) (1000 * number_1);
            }
            else if(number_1 < 100)
            {
                    dp_pos = second;
                    number = (int) (100 * number_1);
            }
            else if(number_1 < 1000)
            {
                    dp_pos = third;
                    number = (int) (10 * number_1);
            }
            else
            {
                    dp_pos = none;
                    number = (int) number_1;
            }
    }

    if(number < 0) /* negative printing */
    {
            number = -1 * number;

            *(digits+1) = (uint32_t) (number/100); /* hundreds digit extraction
*/
            number -= (int) (*(digits+1) * 100);
            *(digits+2) = (uint32_t) (number/10); /* tens digit extraction */
            number -= (int) (*(digits+2) * 10);
            *(digits+3) = (uint32_t) number; /* ones digit extraction */

            for(uint32_t i=0; i<4; i++)
            {
                    ssd_digit(i);
                    if(i == 0) /* print - sign */
                    {
                            GPIOB->ODR &= (0xBFU);
                    }
                    else
                    {
                            ssd_set(digits[i]);
                    }
            }
            ssd_digit_clear();
    }
    else /* positive printing */
    {
            *(digits+0) = (uint32_t) (number/1000); /* thousands digit
extraction */
            number -= (int) (*(digits+0) * 1000);
            *(digits+1) = (uint32_t) (number/100); /* hundreds digit extraction
*/
            number -= (int) (*(digits+1) * 100);
            *(digits+2) = (uint32_t) (number/10); /* tens digit extraction */
            number -= (int) (*(digits+2) * 10);
            *(digits+3) = (uint32_t) number; /* ones digit extraction */

            for(uint32_t i=0; i<4; i++)
            {
                    ssd_digit(i);
                    ssd_set(digits[i]);
             }
             ssd_digit_clear();
      }
}
```

bsp.h

```
/* BSP.H                    */
/* Author: Ömer Emre Polat */
/* Student No: 1801022037   */

#ifndef BSP_H_
#define BSP_H_

#include "stm32g031xx.h"

#define RefreshRate 10000
#define ButtonBounceTime 400

float number_1;
char kp_input;

typedef enum
{
      none = 0, first = 1, second = 2, third = 3, fourth = 4
}decimal_point_pos;

decimal_point_pos dp_pos;

void BSP_system_init(void);

void BSP_IWDG_init(void);
void BSP_IWDG_refresh(void);

void ssd_print(void);

void ssd_set(volatile uint32_t);
void ssd_set_operation(volatile char);
void ssd_set_overflow(void);
void ssd_set_clear(void);

void ssd_digit(volatile uint32_t);
void ssd_digit_clear(void);

void kp_get_input(void);

#endif
```

Most of the code comes from the preceding project 2 and was explained thoroughly in the project 2 report. The changes done are removing the state machine in the external interrupt handler and removing the extra parts inside the ssd_print function.

2.2 Prices and Parts List

Parts that are used in the problem 1 can be written into the table down below.

	Part Name	**Amount**	**Price**
1	Breadboard	1	7.50TL
2	Jumper Cable (Male-Male)	40	40 piece is around 3.16TL
3	4x4 Keypad	1	5.41TL
4	470 ohm resistors	8	~
5	Seven Segment Display	1	7.99TL
6	STM32G031K8T6 Board	1	102.50TL
7	8R 0.5W 83dB 36x5mm Speaker	1	6.70TL
			TOTAL
			133.26TL

Total price list comes to around 133.26TL excluding the cargo costs. Build takes around 5 to 10 minutes using the block diagram given at the beginning of the report.

2.3 Video and Documenting the Project

Video documentation of the project will be loaded into the youtube. The link is given below.

https://www.youtube.com/watch?v=oKAO_S8MiVA

3. Results and General Comments

The results were as expected. I've learned to design a code implement it to a board and build a project on top of it all. The code had a lot of problems because of the keypad get input is not working consistently. There we're some research done on the tone frequencies since I've had no past experience with musical instruments.

As general comments there were a lot of debugging to see if thing were working properly or not. These debugging sessions could get confusing especially when the onboard written pin names are dramatically different than the software names of the pins. Lastly keypad get input function was not consistent so consistent but when the correct inputs were registered from the keypad the tone generator worked as expected.

4. References

[1]. RM0444 Reference manual
https://www.st.com/resource/en/reference_manual/dm00371828-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[2]. STM32G031K8 Datasheet

https://www.st.com/en/microcontrollers-microprocessors/stm32g031k8.html

[3]. Key frequencies Wikipedia
https://en.wikipedia.org/wiki/Piano_key_frequencies