```c
#include "stm32l476xx.h"
#include "SysClock.h"
#include "LED.h"
#include "Keypad.h"
#include "Beeper.h"

#define DELAY for(int i = 0; i < 10000000; i ++);

int toneFrequencies[16] = {131, 523, 494, 440, 392, 349, 330, 294, 262, 247, 220, 193, 175, 165, 147,
587};

int main(void){

  int running = 1;

  System_Clock_Init(); // Switch System Clock = 80 MHz

  GPIOInitRow();
  GPIOInitCol();

  InitBeeper();

  enum Keys keyPressed;

  while(running == 1){
    keyPressed = scan();
    if(keyPressed == Key_None){
      StopBeep();
    }else{
      Beep(toneFrequencies[keyPressed]);
    }
    DELAY;
  }

}
```

```c
/* Keypad.h - function prototypes and declarations for the c implemetation file
*
*     Copyright 2020 Graeme Judge, Sean Berkvens
*   Change Log:
*       May 3, 2020: Source file created
*/

#include "stm321476xx.h"

//defined values
#define DEBOUNCETIME 3

#define COL1 0x2C
#define COL2 0x2A
#define COL3 0x26
#define COL4 0xE

#define START  0b11110111

#define ROW4 0b1110
#define ROW3 0b1101
#define ROW2 0b1011
#define ROW1 0b0111



enum Keys {
  Key_None = -1,
  Key_1,
  Key_2,
  Key_3,
  Key_A,
  Key_4,
  Key_5,
  Key_6,
  Key_B,
  Key_7,
  Key_8,
  Key_9,
  Key_C,
  Key_Star,
  Key_0,
  Key_Pound,
  Key_D
};

static enum Keys Matrix[4][4] = {{Key_1, Key_2, Key_3, Key_A},
                         {Key_4, Key_5, Key_6, Key_B},
                         {Key_7, Key_8, Key_9, Key_C},
                         {Key_Star, Key_0, Key_Pound, Key_D}};


/*
  void GPIOInit;

  initalizes the gpio ports as needed for the input and the outout for
  the program to function properly

  Input: None
  Output: None
*/
void GPIOInit();

/*
  void GPIOInitRow;

  initalizes the gpio ports as needed for the row outout for
  the program to function properly

  Input: None
  Output: None
*/
```

```
73  void GPIOInitRow(void);
74
75  /*
76    void GPIOInitCol;
77
78    initalizes the gpio ports as needed for the input for
79    the program to function properly
80
81    Input: None
82    Output: None
83  */
84  void GPIOInitCol(void);
85
86  /*
87    void scan;
88
89    scans the keypad and returns the key thats being pressed
90
91    Input: None
92    Output: The pressed key as defined in the Keys struct
93  */
94  enum Keys scan(void);
95
96  /*
97    void getInput;
98
99    Gets the input form the port
100
101   Input: None
102   Output: the lowest 8 bits from the input port
103 */
104 uint8_t getInput(void);
105
106 /*
107   void debouncedKey;
108
109   gets the pressed key value and uses debounce to ensure the correct value
110
111   Input: None
112   Output: the bit pattern of the input  port with debounce
113 */
114 uint8_t debouncedKey(void);
115
```

```c
/* Keypad.c - implemetation of Keypad.h
*
*     Copyright 2020 Graeme Judge, Sean Berkvens
*   Change Log:
*       May 3, 2020: Source file created
*/


#include "Keypad.h"

#define KEYDELAY for(int i =0; i < 8000; i++)

#define DELLY for(int i =0; i < 100000; i++)

uint8_t rows[4] = {ROW1, ROW2, ROW3, ROW4};

void GPIOInit(){
  GPIOInitRow();
  GPIOInitCol();

}

void GPIOInitRow(){
  //Clock init
  RCC -> AHB2ENR |= RCC_AHB2ENR_GPIOEEN;
  //using PE12 to PE15
  for(int i = 12; i < 16; i++){ //for loop to go through all of the pins being used and initializes them
    // GPIO Mode
    //00 input mode
    //01 output mode
    //10 alternate mode
    //11 analog mode
    GPIOE->MODER &= ~(3UL<<(2*i));
    GPIOE->MODER |= (1UL<<(2*i));       //  Output(01)

    //GPIO Speed
    GPIOE->OSPEEDR &= (~(3UL<<(2*i)));
    GPIOE->OSPEEDR |=   (3UL<<(2*i));  // Low speed

    //00 none
    //01 pullup
    //10 pulldown
    //11 reserved
    GPIOE->PUPDR   |= (1UL<<(2*i));  // pull-up

    //GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
    GPIOE->OTYPER |= (1U<<(1*i));       // Push-pull open drain
  }

  //clears the output data register for port E
  GPIOE->ODR &= (uint32_t)0x0000;
}



void GPIOInitCol(){
  //Clock init
  RCC -> AHB2ENR |= RCC_AHB2ENR_GPIOAEN;

  //using PA1 to PA5 excluding 4
  for(int i = 1; i < 4; i++){ //for loop to go through all of the pins being used and initializes them
      // GPIO Mode
      //00 input mode
      //01 output mode
      //10 alternate mode
      //11 analog mode
      GPIOA->MODER &= ~(3UL<<(2*i));

      //00 none
      //01 pullup
      //10 pulldown
      //11 reserved
```

```
73          GPIOA->PUPDR |= (1UL<<(2*i));  // pull-up
74        }
75
76          // GPIO Pin 5
77          //00 input mode
78          //01 output mode
79          //10 alternate mode
80          //11 analog mode
81          GPIOA->MODER &= ~(3UL<<(2*5));
82
83          //00 none
84          //01 pullup
85          //10 pulldown
86          //11 reserved
87          GPIOA->PUPDR |= (1UL<<(2*5));  // pull-up
88    }
89
90
91
92    enum Keys scan(){
93      GPIOE->ODR &= 0b0000 << 12;
94      DELLY;
95      uint8_t button = debouncedKey();
96
97      if((button & 0xFF) == 0x2E){
98        return Key_None;
99      }
100
101     for(int i = 0; i < 4; i++){
102       GPIOE->ODR = (rows[i])<<12;
103       DELLY;
104       button = debouncedKey();
105       if(button != 0x2E){
106         if(button == COL1){
107           return Matrix[i][0];
108         }
109         if(button == COL2){
110           return Matrix[i][1];
111         }
112         if(button == COL3){
113           return Matrix[i][2];
114         }
115         if(button == COL4){
116           return Matrix[i][3];
117         }
118       }
119     }
120   }
121
122   uint8_t getInput(){
123     volatile uint8_t IDR = (GPIOA->IDR & 0x2E);
124     return IDR;
125   }
126
127   uint8_t debouncedKey(){
128     uint8_t read = getInput();
129
130     for(int i = 0; i < DEBOUNCETIME; i++){
131       uint8_t newRead = getInput();
132       if(read != newRead || read == 0x2E){
133         read = 0x2E;
134         return read;
135       }
136       KEYDELAY;
137     }
138     return read;
139   }
140
141
142
143
144
```

```
145
146
147
```

```
1  /* Beeper.h - function prototypes and declarations for the c implemetation file
2  *
3  *     Copyright 2020 Graeme Judge, Sean Berkvens
4  *   Change Log:
5  *       May 3, 2020: Source file created
6  */
7
8  #include "stm32l476xx.h"
9
10 /*
11   void InitBeeper;
12
13   initalizes the gpio ports as needed for the input and the outout for
14   the program to function properly
15
16   Input: None
17   Output: None
18 */
19 void InitBeeper( void );
20
21 /*
22   void Beep;
23
24   Causes the beeper to create a tone at a specific frequency
25
26   Input: a 16bit integer for the frequency to beep at
27   Output: None
28 */
29 void Beep( uint32_t hertz );
30
31 /*
32   void StopBeep;
33
34   Stops the beeper from making noise
35
36   Input: None
37   Output: None
38 */
39 void StopBeep( void );
40
```

```c
/* Beeper.c - implementation of Beeper.h
*
*     Copyright 2020 Graeme Judge, Sean Berkvens
*   Change Log:
*       May 3, 2020: Source file created
*/

#include "utils.h"
#include "Beeper.h"
#define PIN 6

void InitBeeper( void ){
  SET_BITS(RCC->AHB2ENR, RCC_AHB2ENR_GPIOBEN); //clock

  FORCE_BITS(GPIOB->MODER, 3UL << (2*PIN), 2UL << (2*PIN));
  FORCE_BITS(GPIOB->AFR[0], 0xF << (4 * PIN), 2UL << (4 * PIN));

  FORCE_BITS(GPIOB->PUPDR, 3UL << (2*PIN), 0);

  SET_BITS(RCC->APB1ENR1, RCC_APB1ENR1_TIM4EN);//time 4 clock

  CLR_BITS(TIM4-> CR1, TIM_CR1_DIR);  //up counting

  //TIM4->PSC = prescaleValue; --> 1MHz clock is 1us
  TIM4->PSC = 80-1;

  TIM4->ARR = 5000000 - 1; //auto reload every 0.5ms

  CLR_BITS(TIM4->CCMR1, TIM_CCMR1_OC1M);

  TIM4 -> CCR1 = 0; //Start without beeping

  SET_BITS(TIM4->BDTR, TIM_BDTR_MOE);   //output enable

  SET_BITS(TIM4->CCMR1, TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2);   //pwm

  CLR_BITS(TIM4 -> CCER, TIM_CCER_CC1P);        //active high
  SET_BITS(TIM4 -> CCER, TIM_CCER_CC1E);
  //start the counter
  SET_BITS(TIM4->CR1, TIM_CR1_CEN);
}

void Beep( uint32_t hertz ){
  SET_BITS(TIM4->BDTR, TIM_BDTR_MOE); //Turn beeper on
  SET_BITS(TIM4->CR1, TIM_CR1_CEN);

  //math things for the duty cycle
  uint32_t periodInUs = (1.0 / (double)hertz) * 1000000; //get uS period
  uint32_t autoReloadValue = periodInUs * 10 - 1;
  TIM4->ARR = autoReloadValue;
  TIM4 -> CCR1 = (TIM4->ARR + 1) / 2;
}

void StopBeep(){
  CLR_BITS(TIM4->CR1, TIM_CR1_CEN); //turns beeper off
  CLR_BITS(TIM4->BDTR, TIM_BDTR_MOE);
}
```