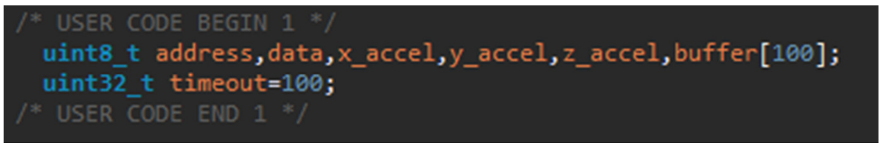
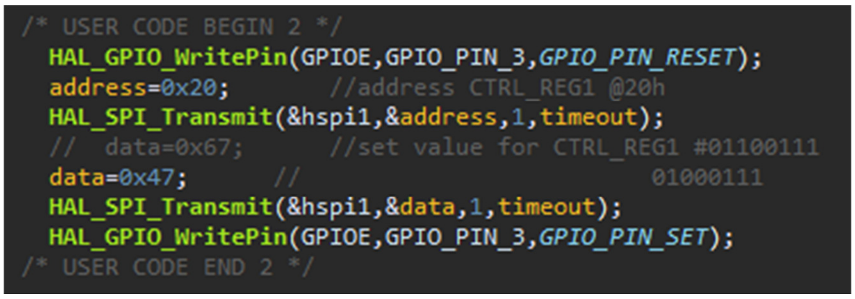
<u>Source code description:</u>

1. Accelerator
   - Variable declaration (code 1)
     i. address => address of register in IC LIS302DL to be adjusted
     ii. data => value to write to register in IC at address
     iii. x_accel => variable to store x value from accelerometer
     iv. y_accel => variable to store y value from accelerometer
     v. z_accel => variable to store z value from accelerometer
     vi. buffer[] => temporary buffer for storing message to be sent to uart
     vii. timeout => variable to store timeout value



```
/* USER CODE BEGIN 1 */
  uint8_t address,data,x_accel,y_accel,z_accel,buffer[100];
  uint32_t timeout=100;
/* USER CODE END 1 */
```

   - Initialization (code 2): describing from top
     i. Set chip select(CS) to choose slave device of SPI protocol by first setting CS(PE3) low
     ii. And then send address of register to modify value in this case the register: CTRL_REG1 (addr=20h)
     iii. To write to register, by passing address to ACC MEM first and then send data to setup such register: (data=67h: 0110_0111) detail in datasheet
     iv. And finally bring the CS pin high again to end SPI protocol sequence

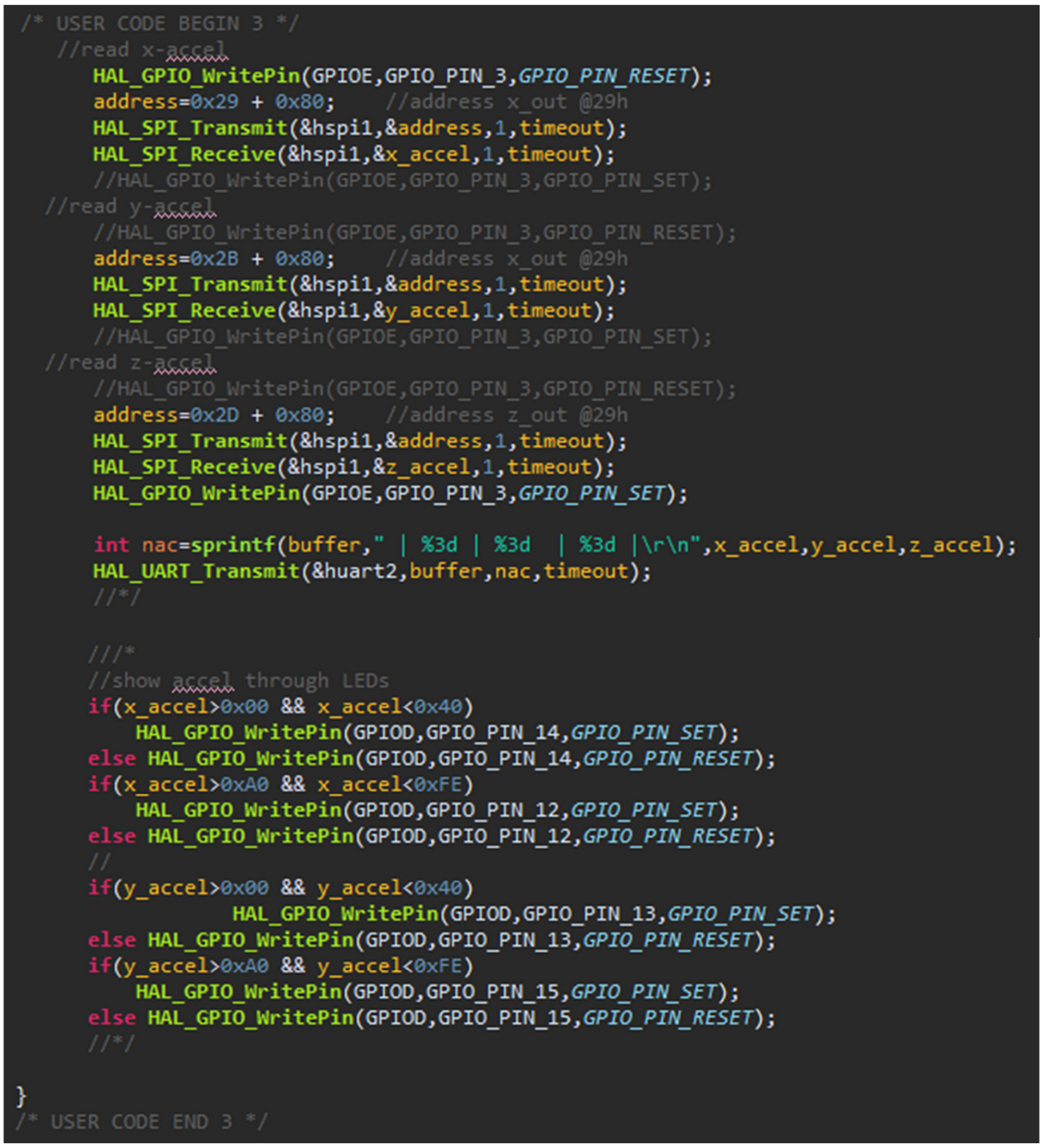


```
/* USER CODE BEGIN 2 */
  HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_RESET);
  address=0x20;        //address CTRL_REG1 @20h
  HAL_SPI_Transmit(&hspi1,&address,1,timeout);
//  data=0x67;         //set value for CTRL_REG1 #01100111
  data=0x47;       //                          01000111
  HAL_SPI_Transmit(&hspi1,&data,1,timeout);
  HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_SET);
/* USER CODE END 2 */
```

   - Looping code (code 3)
     i. To get acceleration value of each axis, by using SPI protocol, same sequence is applied, the only difference is that the address passed is offset by 80h, to tell the destination that the operation will be 'read' from register

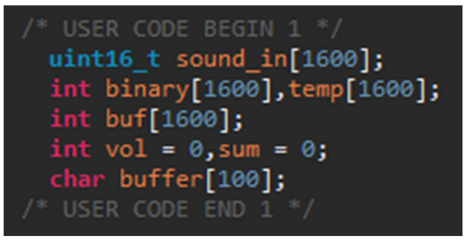
ii. The destination address of each axis is in datasheet

iii. In the case, I show the results in both uart, and LEDs



```
/* USER CODE BEGIN 3 */
   //read x-accel
      HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_RESET);
      address=0x29 + 0x80;    //address x_out @29h
      HAL_SPI_Transmit(&hspi1,&address,1,timeout);
      HAL_SPI_Receive(&hspi1,&x_accel,1,timeout);
      //HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_SET);
  //read y-accel
      //HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_RESET);
      address=0x2B + 0x80;    //address x_out @29h
      HAL_SPI_Transmit(&hspi1,&address,1,timeout);
      HAL_SPI_Receive(&hspi1,&y_accel,1,timeout);
      //HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_SET);
  //read z-accel
      //HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_RESET);
      address=0x2D + 0x80;    //address z_out @29h
      HAL_SPI_Transmit(&hspi1,&address,1,timeout);
      HAL_SPI_Receive(&hspi1,&z_accel,1,timeout);
      HAL_GPIO_WritePin(GPIOE,GPIO_PIN_3,GPIO_PIN_SET);

      int nac=sprintf(buffer," | %3d | %3d  | %3d |\r\n",x_accel,y_accel,z_accel);
      HAL_UART_Transmit(&huart2,buffer,nac,timeout);
      //*/

      ///*
      //show accel through LEDs
      if(x_accel>0x00 && x_accel<0x40)
          HAL_GPIO_WritePin(GPIOD,GPIO_PIN_14,GPIO_PIN_SET);
      else HAL_GPIO_WritePin(GPIOD,GPIO_PIN_14,GPIO_PIN_RESET);
      if(x_accel>0xA0 && x_accel<0xFE)
          HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12,GPIO_PIN_SET);
      else HAL_GPIO_WritePin(GPIOD,GPIO_PIN_12,GPIO_PIN_RESET);
      //
      if(y_accel>0x00 && y_accel<0x40)
                  HAL_GPIO_WritePin(GPIOD,GPIO_PIN_13,GPIO_PIN_SET);
      else HAL_GPIO_WritePin(GPIOD,GPIO_PIN_13,GPIO_PIN_RESET);
      if(y_accel>0xA0 && y_accel<0xFE)
          HAL_GPIO_WritePin(GPIOD,GPIO_PIN_15,GPIO_PIN_SET);
      else HAL_GPIO_WritePin(GPIOD,GPIO_PIN_15,GPIO_PIN_RESET);
      //*/

}
/* USER CODE END 3 */
```

2. Digital microphone
   - Variable declaration (code 1)
      i. sound_in[] => data receiving from I2S protocol
      ii. binary[] => variable to store binary bit converted from sound_in
      iii. temp[] =>
      iv. buf[] => variable to store
      v. vol => variable to store volume of sound/amplitude of the wave
      vi. sum => buffer for storing accumulative sum of sampling sound wave data
      vii. buffer[] => temporary buffer for storing message to be sent to uart

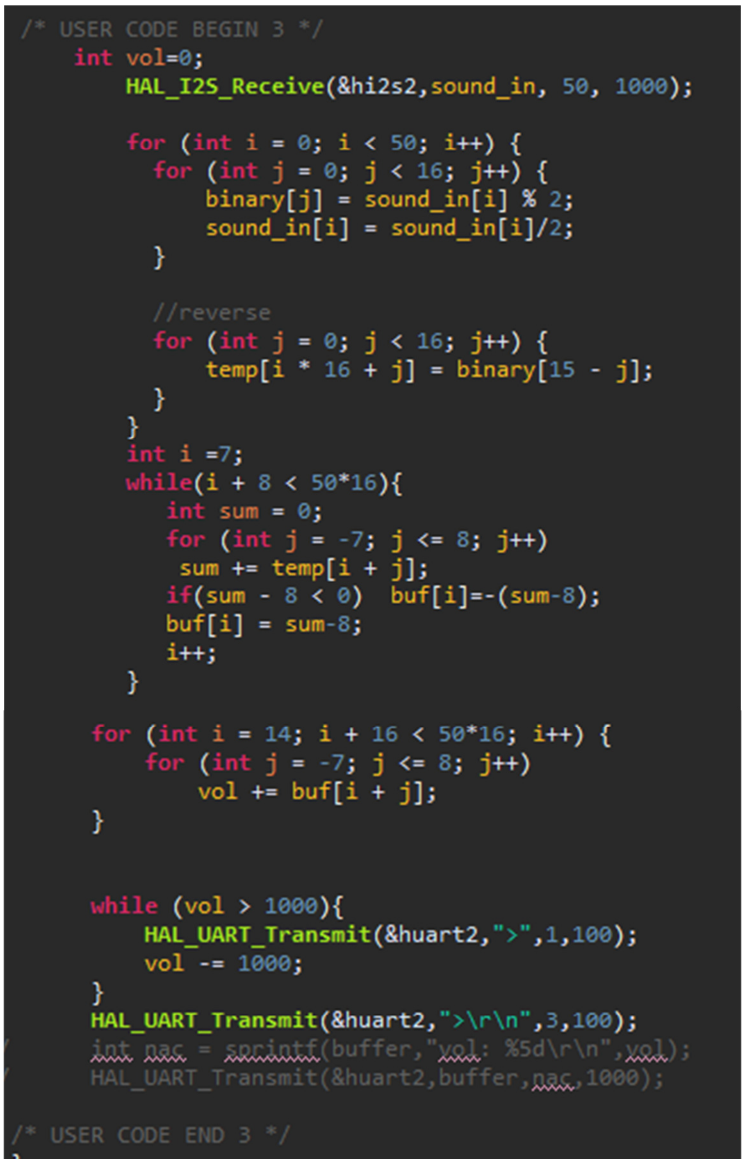


```
/* USER CODE BEGIN 1 */
  uint16_t sound_in[1600];
  int binary[1600],temp[1600];
  int buf[1600];
  int vol = 0,sum = 0;
  char buffer[100];
/* USER CODE END 1 */
```

   - Looping code (code 3)
      i. First, function to receive sound data from microphone via I2S protocol is used. This code set to sample 50 set of such data, 8-bit each, because it is uint16_t.
      ii. The first inner for loop converts sound_in to binary stored in binary[]
      iii. Next loop is to reverse the binary to store the ‘correct binary’ value into temp
      iv. Next while loop calculates accumulative sum of temp and put in variable buf[] with only positive(taking absolute of amplitude: volume in case of sound wave)
      v. Then, vol store buf[] in a period of time to show the intensity of sound in a short instant
      vi. Finally the while loop will display vol in form of bar of sound level(character ‘>’ in this case)

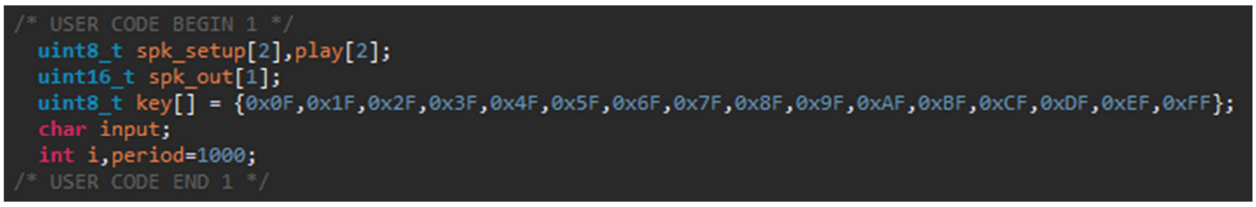
vii. Result will be, for example, like this:

- >>
- >
- >>>>>>>>>
- >>>>>>>>>>>>
- >>>>>>>>>>>
- >>>>>>>>>
- >>>>>>
- >>>
- >>>>>
- >>>
- >>>>>>>>>



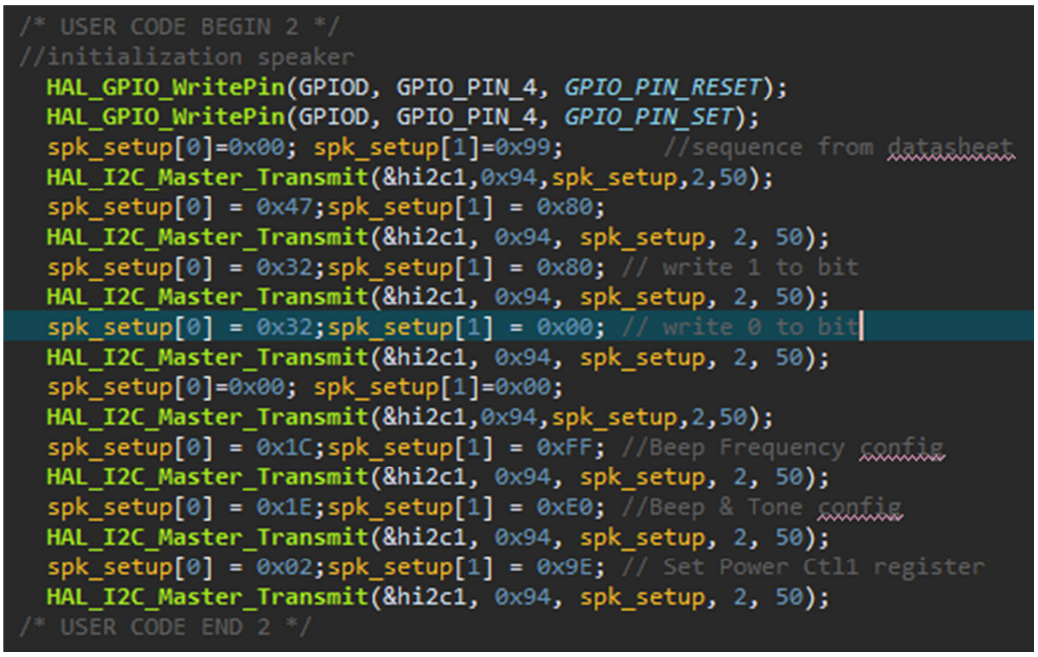
```
/* USER CODE BEGIN 3 */
    int vol=0;
        HAL_I2S_Receive(&hi2s2,sound_in, 50, 1000);

        for (int i = 0; i < 50; i++) {
          for (int j = 0; j < 16; j++) {
              binary[j] = sound_in[i] % 2;
              sound_in[i] = sound_in[i]/2;
          }

          //reverse
          for (int j = 0; j < 16; j++) {
              temp[i * 16 + j] = binary[15 - j];
          }
        }
        int i =7;
        while(i + 8 < 50*16){
           int sum = 0;
           for (int j = -7; j <= 8; j++)
            sum += temp[i + j];
           if(sum - 8 < 0)  buf[i]=-(sum-8);
           buf[i] = sum-8;
           i++;
        }
      for (int i = 14; i + 16 < 50*16; i++) {
          for (int j = -7; j <= 8; j++)
              vol += buf[i + j];
      }


      while (vol > 1000){
          HAL_UART_Transmit(&huart2,">",1,100);
          vol -= 1000;
      }
      HAL_UART_Transmit(&huart2,">\r\n",3,100);
/     int pac = sprintf(buffer,"vol: %5d\r\n",vol);
/     HAL_UART_Transmit(&huart2,buffer,pac,1000);

/* USER CODE END 3 */
```

3. Speaker
   - Variable declaration (code 1)
     1. spk_setup[] => data buffer used to set setup sequence of I2C protocol
     2. play[] => variable to store which frequency to play note
     3. spk_out[] => data buffer used to send to I2S output through speaker
     4. key[] => variable to store music key value
     5. input => variable to store input key value via uart
     6. k,i => variable used to iterate through for loops
     7. period => variable to store period of time each key will be played



```
/* USER CODE BEGIN 1 */
  uint8_t spk_setup[2],play[2];
  uint16_t spk_out[1];
  uint8_t key[] = {0x0F,0x1F,0x2F,0x3F,0x4F,0x5F,0x6F,0x7F,0x8F,0x9F,0xAF,0xBF,0xCF,0xDF,0xEF,0xFF};
  char input;
  int i,period=1000;
/* USER CODE END 1 */
```

   - Initialization (code 2)
     1. Initialization sequence of speaker(detail listed in datasheet)



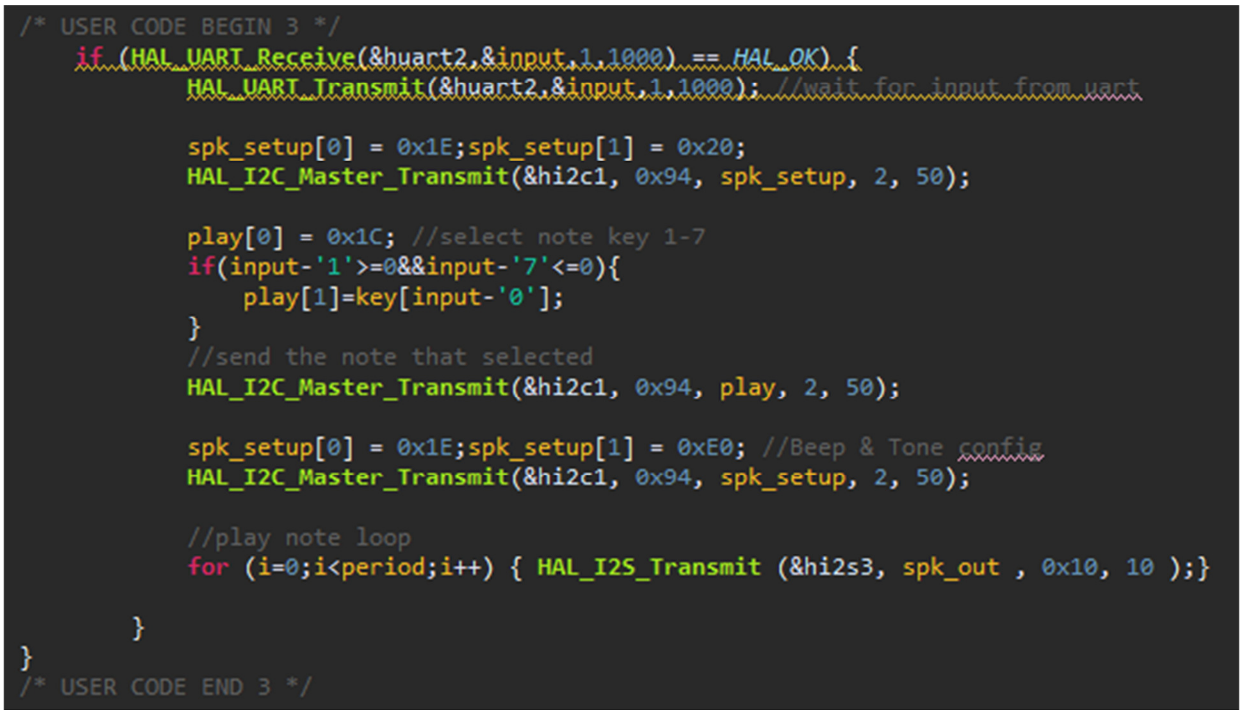
```
/* USER CODE BEGIN 2 */
//initialization speaker
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN_SET);
  spk_setup[0]=0x00; spk_setup[1]=0x99;       //sequence from datasheet
  HAL_I2C_Master_Transmit(&hi2c1,0x94,spk_setup,2,50);
  spk_setup[0] = 0x47;spk_setup[1] = 0x80;
  HAL_I2C_Master_Transmit(&hi2c1, 0x94, spk_setup, 2, 50);
  spk_setup[0] = 0x32;spk_setup[1] = 0x80; // write 1 to bit
  HAL_I2C_Master_Transmit(&hi2c1, 0x94, spk_setup, 2, 50);
  spk_setup[0] = 0x32;spk_setup[1] = 0x00; // write 0 to bit
  HAL_I2C_Master_Transmit(&hi2c1, 0x94, spk_setup, 2, 50);
  spk_setup[0]=0x00; spk_setup[1]=0x00;
  HAL_I2C_Master_Transmit(&hi2c1,0x94,spk_setup,2,50);
  spk_setup[0] = 0x1C;spk_setup[1] = 0xFF; //Beep Frequency config
  HAL_I2C_Master_Transmit(&hi2c1, 0x94, spk_setup, 2, 50);
  spk_setup[0] = 0x1E;spk_setup[1] = 0xE0; //Beep & Tone config
  HAL_I2C_Master_Transmit(&hi2c1, 0x94, spk_setup, 2, 50);
  spk_setup[0] = 0x02;spk_setup[1] = 0x9E; // Set Power Ctl1 register
  HAL_I2C_Master_Transmit(&hi2c1, 0x94, spk_setup, 2, 50);
/* USER CODE END 2 */
```

   - Looping code (code 3)
     1. Starting with receiving input key from uart, which key to be played.
     2. And then setup beep characteristics. (reg#1Eh->write20h)
     3. The address to send message to adjust speaker register is 0x94 (speaker MEM chip number and some setting including power and r/w)

iv. Then to select the right note, by iterating through key[] which collects all notes available. (detail in datasheet), and also print the note in as it is received(echo) via uart.

v. Then send such key to speaker through I2C protocol, and some setting concerning beep characteristics (reg#1Eh->writeE0h)

vi. Finally, output the note via I2S protocol, with some appropriate period, which will determine how long the note will be played.



```
/* USER CODE BEGIN 3 */
    if (HAL_UART_Receive(&huart2,&input,1,1000) == HAL_OK) {
            HAL_UART_Transmit(&huart2,&input,1,1000); //wait for input from uart

            spk_setup[0] = 0x1E;spk_setup[1] = 0x20;
            HAL_I2C_Master_Transmit(&hi2c1, 0x94, spk_setup, 2, 50);

            play[0] = 0x1C; //select note key 1-7
            if(input-'1'>=0&&input-'7'<=0){
                play[1]=key[input-'0'];
            }
            //send the note that selected
            HAL_I2C_Master_Transmit(&hi2c1, 0x94, play, 2, 50);

            spk_setup[0] = 0x1E;spk_setup[1] = 0xE0; //Beep & Tone config
            HAL_I2C_Master_Transmit(&hi2c1, 0x94, spk_setup, 2, 50);

            //play note loop
            for (i=0;i<period;i++) { HAL_I2S_Transmit (&hi2s3, spk_out , 0x10, 10 );}

        }
}
/* USER CODE END 3 */
```