Embedded system design - Final report

Student name: Chi-En Dai (戴麒恩)

NTU student ID: R13942128

Graduate Institute of Communication Engineering - 1st year

***Abstract*—This project involves implementing an Active Noise Cancellation (ANC) system suitable for wall-mounted applications and deploy the system on a resource-limited microcontroller: NUCLEO-64 stm32 F446re. The experimental result show that it can achieve the noise cancellation effect in the sensing area of mems microphone sensing. And from the result of time-frequency analysis, I found that both the fundamental frequency and its harmonics can be cancel after turning on the active mode.**

I. INTRODUCTION

Motivation

In many situations, external noise, such as traffic or construction sounds, penetrates indoor spaces through walls, significantly impacting the quality of life and work efficiency of occupants. Traditional passive noise control measures, such as soundproofing materials, can be effective to some extent but are often costly and less effective at addressing low-frequency noise.

Active Noise Control (ANC), as an advanced technology, can effectively cancel noise by generating sound waves that are out of phase with the noise, making it particularly effective for low-frequency noise suppression. However, most existing ANC systems are designed for open spaces or headphone applications and are not readily applicable to wall-mounted scenarios. Developing an ANC system specifically for walls could effectively reduce noise transmission through wall structures, providing immense value in improving indoor acoustic environments. Therefore, the development of an efficient, reliable, and easy-to-deploy wall-mounted ANC system would play a significant role in supporting future architectural acoustic designs and offer an innovative solution to modern environmental noise challenges.

Another motivation is that the project was constrained to a cyber physical system, it must contain a sensor, an information processing system and an actuator. I was worrying when proposal, since I have no experience in developing control system of a robotic arms or even an autonomous car. But after a lot of survey, I found that I can do something that's correspond to my specialty. While I was major in electrical engineering and have a lot of research experience in signal processing. I decide to do the project about signal. Then after a lot of brainstorming and survey I decide to work with ANC system, by sensing sound wave from environment and process the signal in the microcontroller, then actuated by generating the physical-sound wave to the environment. Which form the loop of a cyber-physical system.

Contribution

The contribution of my work in ANC system development included four parts: analysis and design, algorithm development, hardware & software implementation, experiment design.

1. Analysis and Design:

Analyzing the requirement for a basic ANC system and the controllable parameters and function for a user. Also the hardware components and algorithm for meeting the main function of ANC system. And consider the principle of state-machine to design the system, avoid deadlock or competition in the system.

2. Algorithm Development:

Including the algorithm of DSP filter design, phase reversing and the adaptive adjusting algorithm of phase and band.

3. Hardware & Software Implementation:

Hardware components include: microcontroller, microphone sensor, speaker, amplifier and matrix keyboard. How to implement then together and initializing are an important factor in my whole development process.

Software include the algorithm in 2. and also the pipeline of the main function, operation, state changing.

4. Experiment Design:

How to setup the experiment in real world, and the test case, operating variables. I try two kind of environment setup but in the final I decide one, I will introduce in the following section. Evaluate the

performance by metrics and characteristics from time domain and frequency domain.

II. RELATED WORKS

Paper-1: Active noise control: Open problems and challenges

- Main points:

Virtual sensing techniques:

By mathematic modeling and estimation they simulate a virtual sensor position. And focusing on create the noise cancellation effect in the virtual sensing area.

Active sound quality control:

Through adjusting the spectrogram of residual noise to improve the quality of the residual audio.

- Comparison:

Although their system is more complicated than mine in both modeling algorithm aspect and hardware aspect. But their system is large and expensive due to the computing resource and hardware costs. I thinks that for the design of embedded system, cost is also a important consideration.

Paper-2: Design and Implementation of Real Time Noise Cancellation System based on Spectral Subtraction Method

- Main points:

Implementation:

They handle the ANC problem totally by hardware. With FPGA, the latency can be small to ignore and low-power consumption.

Algorithm:

They perform the main function of ANC system from spectrum domain. The well-known spectral subtraction method. By estimating the magnitude spectrogram, and subtract together.

Voice activity detection:

Integrate with the function of voice activity detection to discriminate the part of speech and noise, improving the accuracy of noise cancellation.

- Comparison:

Their system achieve a good result in both performance and computing efficiency. But the main defect for me is that FPGA is expensive that I can't afford. And the Fourier transform from time-domain to frequency domain take time which might not be a good approach for a STM32 microprocessor. If we adopt the method by spectral subtraction, them it will lead to a long latency and affect the possibility of real-time processing.

Product: NES10-2MK4

- Features:

DSP noise cancellation:

Full adaptive dsp noise cancelling with range of 8 to 40 dB. 8 user-selectable filter levels for customized noise reduction.

User interface:

Three-position function switch: Off, On, DSP.

LED indicator: power on-red, noise cancellation active-green, audio overload-dedicate LED indicator.

Connectivity and Power:

Operates on 10 to 18V DC with 500mA current consumption. With 5W audio output power.

Algorithm:

This product perform the main function of ANC system from spectrum domain. Also spectral subtraction method. By estimating the magnitude spectrogram, and subtract together.

- Comparison:

My system are somehow similar to this product but with a lower hardware specifications. Also different algorithm for active noise cancellation and digital signal processing. Something similar is the use case of this system, both the product and my system can switch to different mode. But as I say, the spectral subtraction method is not suitable for my microcontroller since due to the processor computation speed. And also the power supply is not as large as this product so the speaker is driven by 5V voltage only which limited the power.

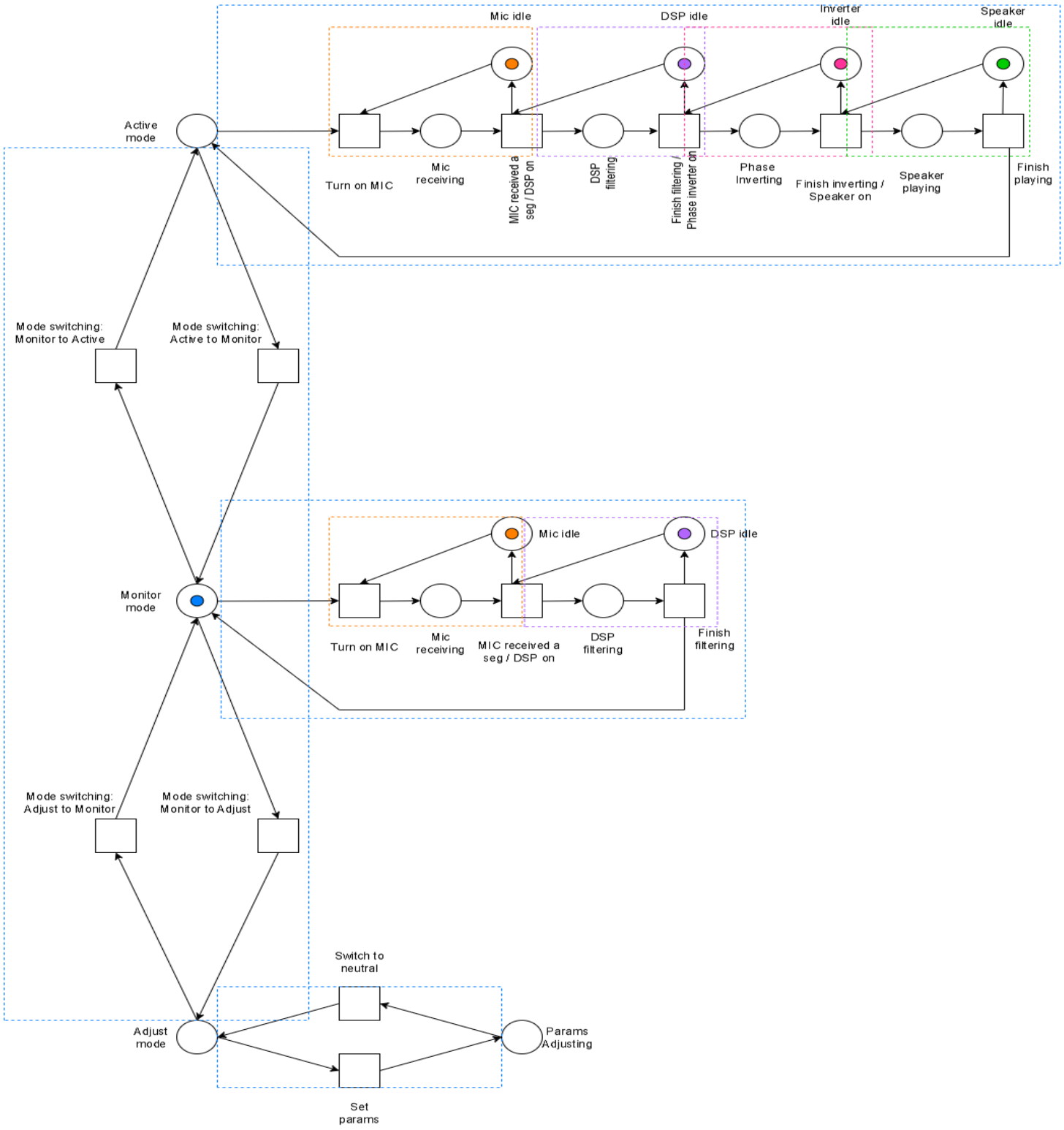

Figure.1 Petri net model of my ANC system

III. SYSTEM SPECIFICATIONS AND DEISIGN

For system specifications and design, I choose Petri net as the modeling technique to show my design. Because the main function of ANC system doesn't have the obvious state changing if using state chart model to express. So I choose Petri net in order to show the transition and event of my system clearly. Especially the token required of every transition. The Petri net is shown in figure.1 on page 3.

Petri net model design explanation:

Please look at the figure.1 in page.3.

1.Tokens:

Blue dots refer to mode and process token.

Orange dots refer to the MIC operating token.

Purple dots refer to the DSP operating token.

Pink dots refer to the phase inverter token.

Green dots refer to the speaker operating token.

2.Loop:

There will be 4 individual loops in my petri net. MIC, DSP, phase inverter and speaker.

3.

The blue block in the petri net is the trace of mode and process token (blue dots). Including: mode switching, params adjusting process in adjust mode, information process in the monitor mode, information process in the active mode (ANC mode).

Thus, every transition in the petri net can be happen if and only if getting the blue token.

4.

For the other token, they are the preset or postset of information process transitions.

5.

Although it seems continuous in the reality, every component works in a discrete time sequence.

6.

Example: To do the transition "turn on the MIC", we have to get both the orange token and blue token.

System specifications:

1. **Input/Output type:** mono-channel audio communicating with microcontroller through I2S protocol

2. **Power supply:** 5V/3.3V

3. **Frequency range:** sound sensor should capture the frequency range from 60Hz to 150Hz

4. **Speaker power:** Speaker power for noise (sound) cancellation should be 4with power supply from MCU

5. **Real-time constraints:** The system should generate sound wave correspond to environment noise in a limited-time

6. Bandpass/Lowpass filter algorithm

7. Adaptive algorithm to adjust phase base on the audio input

8. User can control the parameters like: volume of speaker and frequency band (upper bound and lower bound of a bandpass filter)

IV. SYSTEM IMPLEMENTATION

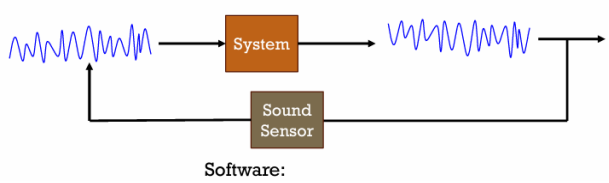

Figure.2 System architecture

Figure 2 show the system architecture. The input sound wave pass through the system for information processing to generate phase-inversed sound wave then actuated by the speaker. The microphone keeps sensing the environmental sound wave. Which forms a loop as the cyber-physical system.

Hardware architecture

1. INMP 441 mems microphone as the sensor. See figure.3 below.

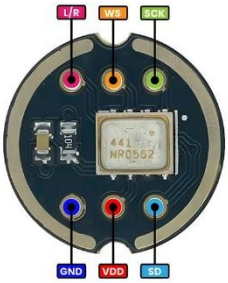

Figure.3 INMP 441 mems microphone sensor

2. NUCLEO-STM32F446 board microcontroller as the information process system. See figure.4 below.

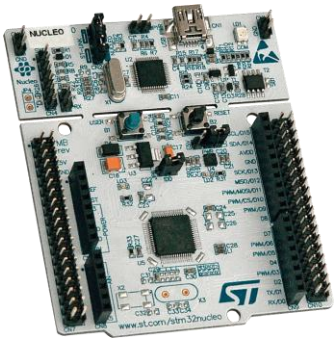

Figure.4 NUCLEO board STM32F446RE

3. MAX98357 speaker amplifier to drive the speaker. See figure.5 below.

Figure.5 MAX98357 I2S mono amplifier

4. Full frequency range speaker (3 ohm, 4W) See figure.6 below.

Figure.6 Speaker with 3 ohm, 4 W

5. 4x4 matrix keypad for user setting. See figure.7 below.

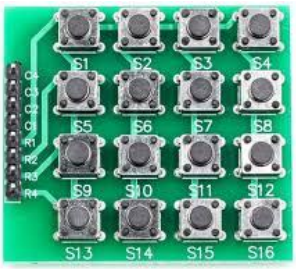

Figure.7 4x4 matrix keypad

Software description

1. **System/Port setup:**

(1) I2S for receive/transmit set to sample rate = 16kHz, data width = half word

(2) UART for transmitting waveform data

(3) Set GPIO Input/Output for 4x4 keypad matrix and set the voltage of input to pull down like the figure.8 shown below:

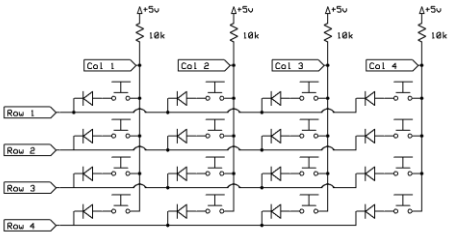

Figure.8 4x4 keypad matrix schematic

2. DSP filter algorithm:

I try with both lowpass and bandpass filter in my code. The design and apply algorithm are shown below.

(1) FIR lowpass filter (with hamming window):

Design:

Given cutoff frequency f_c and filter length N.

→ Estimate ideal impulse response h(n):

The ideal impulse response of lowpass filter is sinc function:

$$h(n) = \begin{cases} \frac{\sin\left(2\pi f_c\left(n-\frac{N-1}{2}\right)\right)}{\pi\left(n-\frac{N-1}{2}\right)}, & n \neq \frac{N-1}{2} \\ 2f_c, & n = \frac{N-1}{2} \end{cases}$$

Where n is index, N is filter length, f_c is cutoff frequency.

→ Define window function w(n):

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right)$$

→ Multiply h(n) with w(n) to get the final impulse response $h_{final}(n)$:

$$h_{\text{final}}(n) = h(n) \cdot w(n)$$

→ Normalize the filter's coefficients:

$$h_{\text{final}}(n) = \sum_{k=0}^{N-1} h_{\text{final}}(k) h_{\text{final}}(n)$$

Apply:

Apply the well-designed filter $h_{final}(n)$ to do convolution with the input signal x(n) to get filter output:

$$y(n) = x(n) * h_{\text{final}}(n) = \sum_{k=0}^{N-1} x(n-k) \cdot h_{\text{final}}(k)$$

(2) FIR bandpass filter:

Design:

Given lower cutoff frequency f_{low} and higher cutoff frequency f_{high} and filter length N

→ Estimate impulse response h(n):

$$h_{\text{ideal}}(n) = h_{\text{lowpass, high}}(n) - h_{\text{lowpass, low}}(n)$$

Where:

$$h_{\text{lowpass, low}}(n) = \begin{cases} \frac{\sin(2\pi f_{\text{low}}(n-\frac{N-1}{2}))}{\pi(n-\frac{N-1}{2})}, & n \neq \frac{N-1}{2} \\ 2f_{\text{low}}, & n = \frac{N-1}{2} \end{cases}$$

$$h_{\text{lowpass, high}}(n) = \begin{cases} \frac{\sin(2\pi f_{\text{high}}(n-\frac{N-1}{2}))}{\pi(n-\frac{N-1}{2})}, & n \neq \frac{N-1}{2} \\ 2f_{\text{high}}, & n = \frac{N-1}{2} \end{cases}$$

Where n is index, N is filter length, f_c is cutoff frequency.

→ Define window function w(n):

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right)$$

→ Multiply h(n) with w(n) to get the final impulse response $h_{final}(n)$:

$$h_{\text{final}}(n) = h(n) \cdot w(n)$$

→ Normalize the filter's coefficients:

$$h_{\text{final}}(n) = \sum_{k=0}^{N-1} h_{\text{final}}(k) h_{\text{final}}(n)$$

Apply:

Apply the well-designed filter $h_{final}(n)$ to do convolution with the input signal x(n) to get filter output:

$$y(n) = x(n) * h_{\text{final}}(n) = \sum_{k=0}^{N-1} x(n-k) \cdot h_{\text{final}}(k)$$

3. Phase reversing and adaptive signal process algorithms:

→ Initialize:

Setting the output signal as the inverse of input signal, with initial phase $\emptyset = 180°$

$$y(n) = -x(n)$$

→ Estimate the residual signal:

$$\text{Residual signal: } r(n) = x(n) + y(n)$$

→ Check the residual signal and adjust phase:

If $r(n) \neq 0$:

$$y(n+1) = -x(n+1)$$

and adjust the phase:

$$y(n+1) = y(n+1) + \alpha x(n+1)$$

else:

$$y(n+1) = y(n)$$

$$y(n+1) = \begin{cases} -y(n) + \alpha \cdot x(n), & \text{if } x(n) \neq 0 \\ y(n), & \text{if } x(n) = 0 \end{cases}$$

→ Repeat the two processes above for every x(n)

4. System parameter adjusting:

→ If s1 button pressed, switch adjust mode

→ Check which button on the matrix be pressed

→ If volume + or – be pressed:

Adjust the volume of speaker

→ If lower cutoff freq + or – be pressed:

Adjust lower cutoff frequency

→ If higher cutoff freq + or – be pressed:

Adjust higher cutoff frequency

V. EVALUATION EXPERIMENT

Experimental setup

My experiment is to simulate an active noise cancellation system for wall/window. The system can cancellate the sound or noise that come behind the wall. The microphone is fixed on the wall, after it received noise and pass the sound through I2S to the information process system which is the STM32 microcontroller. After the estimation on microcontroller, it generates the cancellation wave through the speaker to cancel the sound come behind the wall. The front view of experimental setup is shown as the figure.9 below.

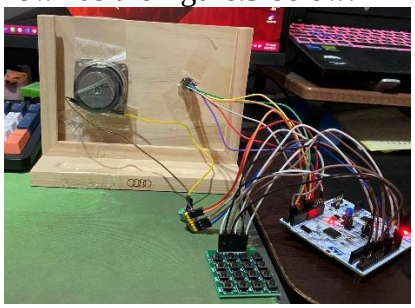

Figure.9 The front view of my experiment environment

And the figure.10 shown below is the top view of my experimental setup.

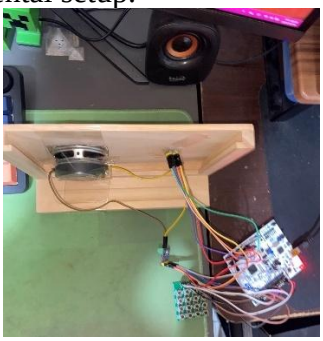

Figure.10 The top view of my experiment environment

Evaluation tools

1. UART serial plotter:

Since the system is suitable for indoor cancellation by cancel the noise passing through the wall. It is impossible to hear the different before/after the active mode turn on if I am in a over field just behind the system themselves. So, for measuring, I use a serial plotter that transmit the data by UART protocol. The serial plotter is opensource, the figure.11 shown below is the interface of it.

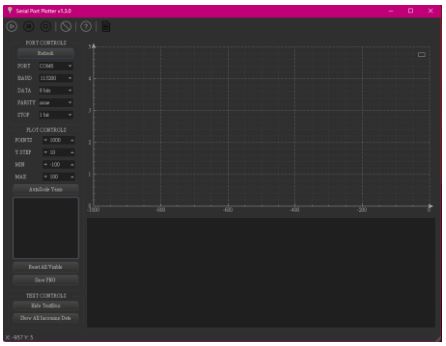

Figure.11 GUI of serial plotter

2. UART transmitter:

In order to listen the difference or the effect of active mode. I additionally write another code to transmit the received sound and save the audio to my computer.

Waveform comparison

1. Frequency fixed, Volume change

Before **After**

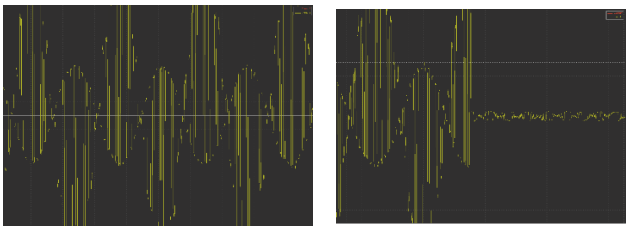

Figure.12 When volume = 30%, Freq=500Hz

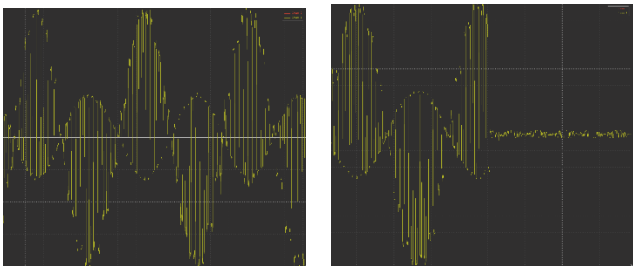

Figure.13 When volume = 50%, Freq=500Hz

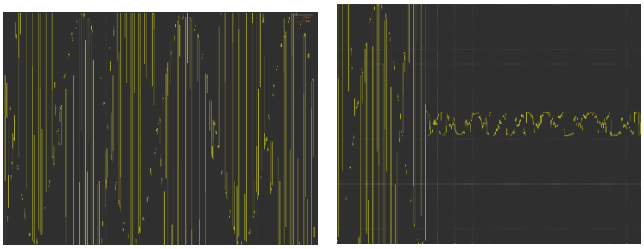

Figure.14 When volume = 70%, Freq=500Hz

2. Frequency change, Volume fixed

Before **After**

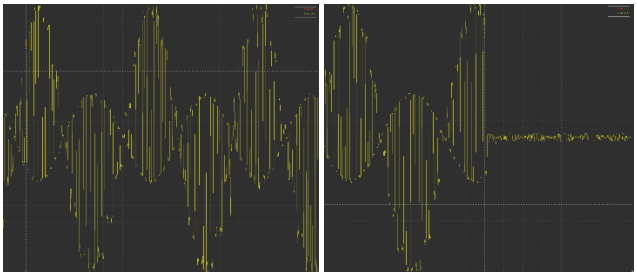

Figure.15 When volume = 50%, Freq=500Hz

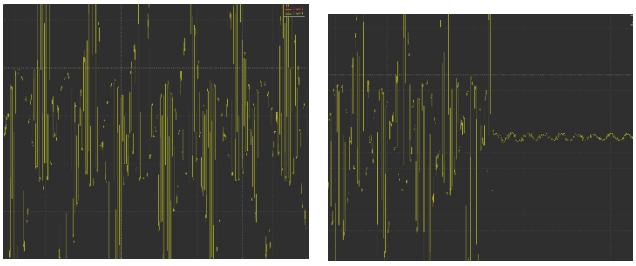

Figure.16 When volume = 50%, Freq=1000Hz

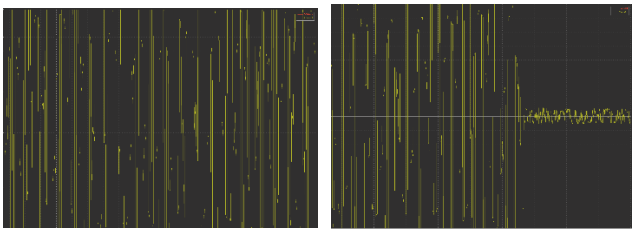

Figure.17 When volume = 50%, Freq=1500Hz

Conclusion:

From the waveform comparison we can see that the system is suitable for the three noises under normal frequency, but it is not good enough for a higher volume when 70% because of the gain of speaker. So, for a louder noise, user should set the gain larger then the system can cancellate the louder noise.

SDR score

	Before (dB)	After (dB)
V=30%, F=500Hz	-3	22
V=50%, F=500Hz	-9	16
V=70%, F=500Hz	-14	10
V=50%, F=1000Hz	-8	15
V=50%, F=1500Hz	-9	13

Conclusion:

Same as the conclusion in the waveform comparison.

Time-Frequency analysis

1. Frequency fixed, Volume change

X-axis is time, Y-axis is frequency and the degree of lightness are the intensity of that T-F bin.

Before **After**

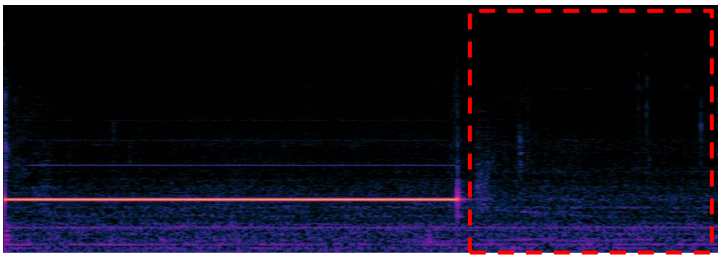

Figure.18 When volume = 30%, Freq=500Hz

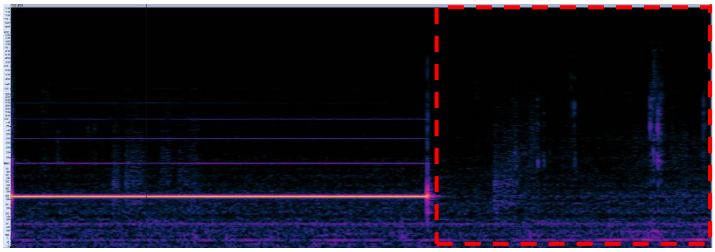

Figure.19 When volume = 50%, Freq=500Hz

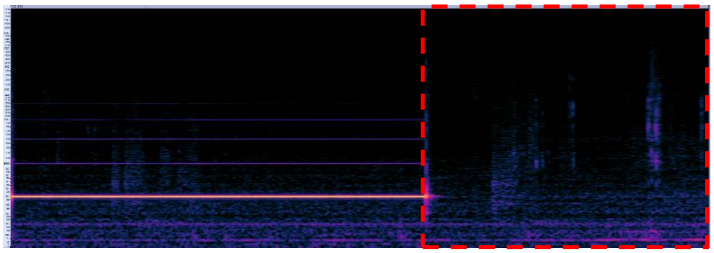

Figure.20 When volume = 70%, Freq=500Hz

2. Frequency change, Volume fixed

Before **After**

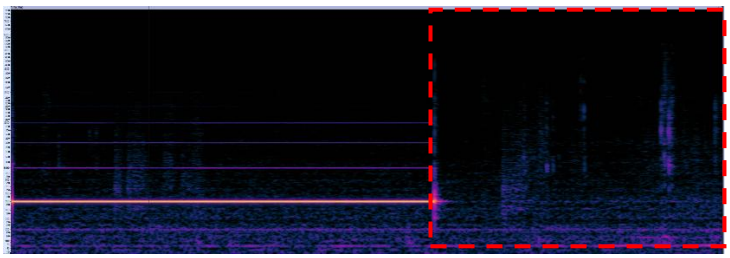

Figure.21 When volume = 50%, Freq=500Hz

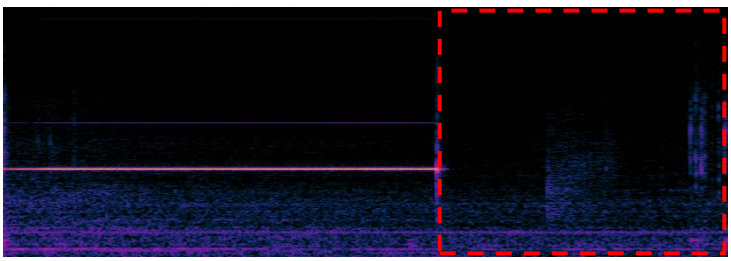

Figure.22 When volume = 50%, Freq=1000Hz

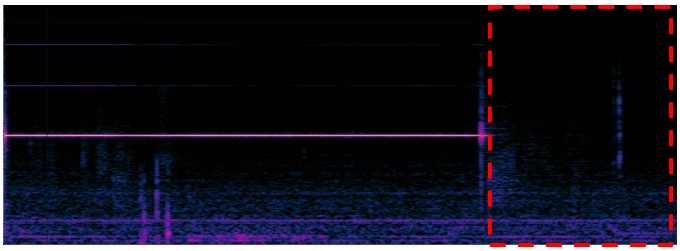

Figure.23 When volume = 50%, Freq=1500Hz

Conclusion:

From the time-frequency analysis, we can see that both the nose and its harmonic are cancelled after turning on the system. But there is still some residual noise in the low frequency area. Probably because of the speaker is not able to well-generate the low frequency sound. Whatever, the intensity of low frequency sound is weak.

VI. COMMENTS

Challenging parts and solutions

1. Acoustic feedback issues

Since my system is to perform noise cancellation, the microphone and speaker should interact together. At the beginning, I set the position of microphone & speaker to be very close and face to face. But it leads to severe acoustic feedback issues sometimes. Which make it unstable for an embedded system.

Solution:

Considering that sound wave transmits faster in solid medium than gas medium. I attached the system to wall so they don't have to be so close enough and can capture the signal from environments.

2. Audio saving problems

When I try to save the audio to my computer it will meet the overload.

Solution:

My solution is to restart the system to free the memory and start one more time. I don't have other solution for this problem so far.

My comments

This final project is somehow challenging and it is not easy. We have to design and deploy the system in reality. For me, as I have the background of electrical engineering. It is not a problem about hardware and circuits, but I still met some physical problems as I say before. However, I thinks that I learn a lot in this course. Following the lectures and homework, I have more concept of my system. About the state changing and operation from the system view. Although my system is not perfect, I am still happy that I finally implement an embedded system from scratch.

REFERENCES

1. S. M. Kuo, K. Kuo, and W.-S. Gan, *"Active noise control: Open problems and challenges,"* in *Proceedings of the 2010 International Conference on Green Circuits and Systems (ICGCS)*, 2010, pp. 1–6. DOI: 10.1109/ICGCS.2010.5543076.
2. A. Firdausi, K. Wirianto, M. Arijal, and T. Adiono, *"Design and Implementation of Real Time Noise Cancellation System based on Spectral Subtraction Method," Procedia Technology*, vol. 11, pp. 607–613, Dec. 2013. DOI: 10.1016/j.protcy.2013.12.287.
3. Micropeta, "STM32F446RE INMP441 I2S Sound," Micropeta, 2024. [Online]. Available: https://www.micropeta.com/video118#google_vignette.
4. ARM Ltd., "CMSIS: Cortex Microcontroller Software Interface Standard," ARM Software, 2024. [Online]. Available: https://arm-software.github.io/CMSIS_6/latest/General/index.html.
5. STMicroelectronics, "STM32CubeG4: STM32Cube MCU Package for STM32G4 Series," GitHub, 2024. [Online]. Available: https://github.com/STMicroelectronics/STM32CubeG4.
6. P. Marwedel, Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things, 4th ed. Springer Nature Switzerland AG, 2021. DOI: 10.1007/978-3-030-60910-8.

Main Code (image are main code text are config)

```c
#include "main.h"
#include "arm_math.h"
#include "stdio.h"
#include <stdbool.h>
#include "string.h"

#define NUM_TAPS 16
float32_t low_cutoff = 0.0f;    //(HZ)  Initial lower cutoff frequency
float32_t LOW_CUTOFF;
float32_t high_cutoff = 5000.0f;  //HZ nitial higher cutoff frequency
float32_t HIGH_CUTOFF;
int VOLUME = 1;
#define BLOCK_SIZE 1

I2S_HandleTypeDef hi2s2;
I2S_HandleTypeDef hi2s3;
DMA_HandleTypeDef hdma_spi2_tx;
DMA_HandleTypeDef hdma_spi3_rx;

UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void PeriphCommonClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2S2_Init(void);
static void MX_I2S3_Init(void);
/* USER CODE BEGIN PFP */
int16_t toReceive;
int16_t toWrite;
arm_fir_instance_f32 S;
float32_t firCoeffs[NUM_TAPS]; // Filter coefficients
float32_t firState[NUM_TAPS + BLOCK_SIZE - 1]; // Filter state
float32_t filter_input;    // filter input
float32_t filter_output;   // filter output
float32_t phase_change = 100.0f;
bool active_mode_select = false;
bool setting_mode_select = false;
bool mode_button_pressed = false;
bool setting_button_pressed = false;
```

```c
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void PeriphCommonClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2S2_Init(void);
static void MX_I2S3_Init(void);
/* USER CODE BEGIN PFP */
int16_t toReceive;
int16_t toWrite;
arm_fir_instance_f32 S;
float32_t firCoeffs[NUM_TAPS]; // Filter coefficients
float32_t firState[NUM_TAPS + BLOCK_SIZE - 1]; // Filter state
float32_t filter_input;    // filter input
float32_t filter_output;   // filter output
float32_t phase_change = 100.0f;
bool active_mode_select = false;
bool setting_mode_select = false;
bool mode_button_pressed = false;
bool setting_button_pressed = false;
/* USER CODE END PFP */

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

bool check_s1_pressed()
{
    static uint32_t last_press_time = 0;
    uint32_t current_time = HAL_GetTick();

    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_6, GPIO_PIN_SET);

    bool pressed = false;
    if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8) == GPIO_PIN_SET)
    {
        if (current_time - last_press_time > 200)
        {
            pressed = true;
            last_press_time = current_time;
        }
    }
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_6, GPIO_PIN_RESET);

    return pressed;
}
```

```c
bool check_specific_button_pressed(int button_id)
{
    int row, col;

    // Map button_id to row and column
    map_button_to_row_col(button_id, &row, &col);

    // Activate the specific row (R1~R4 -> PC6~PC9)
    HAL_GPIO_WritePin(GPIOC, 1 << (row + 6), GPIO_PIN_SET); // Set the corresponding row high

    // Check if the corresponding column (C1~C4 -> PA8~PA11) is high
    bool pressed = HAL_GPIO_ReadPin(GPIOA, 1 << (col + 8)) == GPIO_PIN_SET;

    // Reset the row to low
    HAL_GPIO_WritePin(GPIOC, 1 << (row + 6), GPIO_PIN_RESET);

    return pressed;
}

// Function to map a button ID to row and column
void map_button_to_row_col(int button_id, int *row, int *col)
{
    *row = (button_id - 1) / 4; // Calculate row (0 to 3)
    *col = (button_id - 1) % 4; // Calculate column (0 to 3)
}

void bandpass_fir(float32_t* coeffs, int num_taps, float low_cutoff, float high_cutoff)
{
    int M = num_taps - 1;
    for (int n = 0; n < num_taps; n++)
    {
        if (n == M / 2)
        {
            coeffs[n] = 2 * (high_cutoff - low_cutoff);
        }
        else
        {
            coeffs[n] = (sin(2 * PI * high_cutoff * (n - M / 2)) - sin(2 * PI * low_cutoff * (n - M / 2)))
                        / (PI * (n - M / 2));
            coeffs[n] *= (0.54 - 0.46 * cos(2 * PI * n / M));   // 使用 Hamming 窗
        }
    }
}
```

```c
void lowpass_fir(float32_t* coeffs, int num_taps, float cutoff)
{
    int M = num_taps - 1;
    for (int n = 0; n < num_taps; n++)
    {
        if (n == M / 2)
        {
            coeffs[n] = 2 * cutoff;
        }
        else
        {
            coeffs[n] = sin(2 * PI * cutoff * (n - M / 2)) / (PI * (n - M / 2));
            coeffs[n] *= (0.54 - 0.46 * cos(2 * PI * n / M));  // 使用 Hamming 窗
        }
    }
}

void init_filter()
{
  LOW_CUTOFF = low_cutoff / 16000.0f;
  HIGH_CUTOFF = high_cutoff / 16000.0f;             //If using lowpass filter
    //bandpass_fir(firCoeffs, NUM_TAPS, LOW_CUTOFF, HIGH_CUTOFF);   //If using bandpass filter
  lowpass_fir(firCoeffs, NUM_TAPS, HIGH_CUTOFF);
  arm_fir_init_f32(&S, NUM_TAPS, firCoeffs, firState, BLOCK_SIZE);
}

void adaptive_signal_process(float input_signal[], float output_signal[], int length) {
    float error_signal;
    float current_output = 0;

    for (int i = 0; i < length; i++) {
        error_signal = input_signal[i] + current_output;

        // 輸出下一個值號 (反轉並調整)
        if (fabs(input_signal[i]) > THRESHOLD) {
            current_output = -error_signal * ALPHA;
        } else {
            current_output = current_output;
        }

        output_signal[i] = current_output;
    }
}
```

```c
char buffer[16];
void HAL_I2S_RxCpltCallback(I2S_HandleTypeDef *hi2s)
{
    //Check whether press blue button
    if (HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET) // If pressed
    {
      if (!mode_button_pressed) // If mode_button_pressed = false
            {
        active_mode_select = !active_mode_select; // Mode change
        if (active_mode_select){
          high_cutoff = phase_change;
        }else{
          high_cutoff = 10000.0f;
        }
        init_filter();
        mode_button_pressed = true;      // Button pressed
            }
    }else{
      mode_button_pressed= false;
    }

    //Check S1 button
    if (check_s1_pressed()){
    if (!setting_button_pressed) // Debouncing
    {
      setting_mode_select = !setting_mode_select; // Switch Setting Mode
      setting_button_pressed = true;
    }
  }else{
    setting_button_pressed = false;
  }

    //Set mode or work mode(monitor or active)
    if (setting_mode_select==false){
      //Apply DSP Filter
      filter_input = (float32_t)toReceive; // Transfer to float type
      arm_fir_f32(&S, &filter_input, &filter_output, BLOCK_SIZE);
    //filter_output = toReceive;    // IF no filter apply

      // Mode select
    if (active_mode_select)
    {
      //toWrite = toReceive * (-1);
      toWrite = (int16_t)(filter_output)* -(VOLUME);    //after bandpass filter
      HAL_I2S_Transmit_DMA(&hi2s2, &toWrite, 1);
    }
    }else{
      filter_output = 0;
      // Adjust parameter
    static uint32_t last_adjust_time = 0;
    uint32_t current_time = HAL_GetTick();
    if (current_time - last_adjust_time > 200){    //Adjust every 200 ms
      //Adjusting volume
      if (check_specific_button_pressed(9)){   //Increase volume
        VOLUME++;}
      else if (check_specific_button_pressed(13)){  //Decrease volume
        VOLUME--;}
      //Adjusting frequency band
      else if (check_specific_button_pressed(11)){  // Increase lower cutoff frequency
        low_cutoff += 100.0f;}
      else if (check_specific_button_pressed(15)){  // Decrease lower cutoff frequency
        low_cutoff -= 100.0f;}
      else if (check_specific_button_pressed(12)){  // Increase higher cutoff frequency
        high_cutoff += 100.0f;}
      else if (check_specific_button_pressed(16)){  // Decrease higher cutoff frequency
        high_cutoff -= 100.0f;}
      init_filter();
      last_adjust_time = current_time; // Update last adjustment time
    }
    }
    HAL_I2S_Receive_DMA(hi2s, &toReceive, 1);
}
```

```c
char str[120];
int main(void)
{
  HAL_Init();

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

  /* Configure the peripherals common clocks */
  PeriphCommonClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_USART2_UART_Init();
  MX_I2S2_Init();
  MX_I2S3_Init();
  init_filter();
  /* USER CODE BEGIN 2 */
  HAL_I2S_Receive_DMA(&hi2s3, &toReceive, 1);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    //The code below can plot the signal in serial port plotter
    sprintf(str, "$%d %d;", toWrite, (int16_t)filter_output);    //The signal transmitted to the speaker and the signal pass through the filter
    HAL_UART_Transmit(&huart2, (uint8_t*)str, strlen(str), 10);
    HAL_Delay(1);
  }
}


/**
  * @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_SCALE3);

  /** 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 = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  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_2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

```
void PeriphCommonClock_Config(void)
{
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

  /** Initializes the peripherals clock
  */
  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_I2S_APB1;
  PeriphClkInitStruct.PLLI2S.PLLI2SN = 192;
  PeriphClkInitStruct.PLLI2S.PLLI2SP = RCC_PLLI2SP_DIV2;
  PeriphClkInitStruct.PLLI2S.PLLI2SM = 16;
  PeriphClkInitStruct.PLLI2S.PLLI2SR = 2;
  PeriphClkInitStruct.PLLI2S.PLLI2SQ = 2;
  PeriphClkInitStruct.PLLI2SDivQ = 1;
  PeriphClkInitStruct.I2sApb1ClockSelection = RCC_I2SAPB1CLKSOURCE_PLLI2S;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_I2S2_Init(void)
{
  hi2s2.Instance = SPI2;
  hi2s2.Init.Mode = I2S_MODE_MASTER_TX;
  hi2s2.Init.Standard = I2S_STANDARD_PHILIPS;
  hi2s2.Init.DataFormat = I2S_DATAFORMAT_16B;
  hi2s2.Init.MCLKOutput = I2S_MCLKOUTPUT_DISABLE;
  hi2s2.Init.AudioFreq = I2S_AUDIOFREQ_16K;
  hi2s2.Init.CPOL = I2S_CPOL_LOW;
  hi2s2.Init.ClockSource = I2S_CLOCK_PLL;
  hi2s2.Init.FullDuplexMode = I2S_FULLDUPLEXMODE_DISABLE;
  if (HAL_I2S_Init(&hi2s2) != HAL_OK)
  {
    Error_Handler();
  }
}
static void MX_I2S3_Init(void)
{

  /* USER CODE BEGIN I2S3_Init 0 */

  /* USER CODE END I2S3_Init 0 */

  /* USER CODE BEGIN I2S3_Init 1 */

  /* USER CODE END I2S3_Init 1 */
  hi2s3.Instance = SPI3;
  hi2s3.Init.Mode = I2S_MODE_MASTER_RX;
  hi2s3.Init.Standard = I2S_STANDARD_PHILIPS;
  hi2s3.Init.DataFormat = I2S_DATAFORMAT_16B;
  hi2s3.Init.MCLKOutput = I2S_MCLKOUTPUT_DISABLE;
  hi2s3.Init.AudioFreq = I2S_AUDIOFREQ_16K;
  hi2s3.Init.CPOL = I2S_CPOL_LOW;
  hi2s3.Init.ClockSource = I2S_CLOCK_PLL;
  hi2s3.Init.FullDuplexMode = I2S_FULLDUPLEXMODE_DISABLE;
  if (HAL_I2S_Init(&hi2s3) != HAL_OK)
  {
    Error_Handler();
  }
}
static void MX_USART2_UART_Init(void)
{  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;//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();
  }
}

/**
  * Enable DMA controller clock
```

```
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Stream0_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Stream0_IRQn);
  /* DMA1_Stream4_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Stream4_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Stream4_IRQn);

}
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(GPIOC, GPIO_PIN_6|GPIO_PIN_5|GPIO_PIN_8|GPIO_PIN_9, 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 : PC6 PC7 PC8 PC9 */
  GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_5|GPIO_PIN_8|GPIO_PIN_9;
  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);

  /*Configure GPIO pins : PA8 PA9 PA10 PA11 */
  GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
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
void assert_failed(uint8_t *file, uint32_t line)
{}
#endif /* USE_FULL_ASSERT */
```