THE UNIVERSITY OF DANANG

UNIVERSITY OF SCIENCE AND TECHNOLOGY

Faculty of Advanced Science and Technology

REPORT PROJECT

MICROPROCESSOR AND INTERFACES

TOPIC: THE STM32 COMMUNICATES WITH THE MICROPHONE

Instructor : Nguyen Huynh Nhat Thuong

Mentor : Nguyen Quang Phuong

Class : 21ECE

Group members : Luong Nhu Quynh

Vo Van Buu

Nguyen Thi Tam

Tran Hoang Minh

Da Nang, 06/2024

1. TOPIC OVERVIEW

1.1 Summarize the topic, list the achievements of the topic, describe the functions and operation of the device/system from the user's perspective.

1.1.1. Project Summary

This project focuses on developing a system using the STM32 microcontroller to interface with a microphone, record an audio segment, store the audio, and play it back when needed. The system also has the capability to send the recorded audio data to a computer for playback.

1.1.2. Achieved Results

- Selecting an appropriate microphone module
- Successful Audio Recording: The system can record audio from the microphone and store it in a file .wav in a microSD card
- Using matlab to filter the file wav
- Audio Playback: Users can playback the recorded audio from a microSD card in the computer.

1.1.3. Overall suggestions and solutions

Issues	Solutions
Hardware	STM32F407E Discovery Adafruit I2S MEMS Microphone SPH0648LM4H SDIO microSD card module
Communication Protocols between STM32 and Microphone	I2S, DMA
Communication Protocols between STM32 and SDIO module	SDIO
Digital Signal Processing	Matlab

Table 1: Overall suggestions and solutions

1.1.4. Device Operation from the User's Perspective

- ***Recording:***
  - Users press the record button the system starts recording.
  - Pressing the record button again or after a set duration stops the recording process
- ***Sending Data to the Computer:***
  - Users connect the device to a computer through microSD
  - Users can playback the audio on the computer or save it in the desired file format.

This system is simple yet effective, meeting the basic needs of users for recording, playback, and transmitting audio data.

1.2 Roster and SELF-ASSESSMENT % of members' contributions

Team Member	Assigned Tasks	Contribution (%)
Luong Nhu Quynh	- Microphone interfacing with STM32 via I2S and DMA - Developing the recording functionality	25%
Nguyen Thi Tam	- Interface SD CARD with SDIO in STM32 - Write a .txt file on the SD CARD	25%
Tran Hoang Minh	- Hardware setup - File .wav configuration - Write function to start, write and close the file wav	25%
Vo Van Buu	- Write main() function - Testing and debugging the entire system - File .wav filtering	25%

Table 2: Work Assignment and Self-Evaluation of Contribution Percentage by Team Members

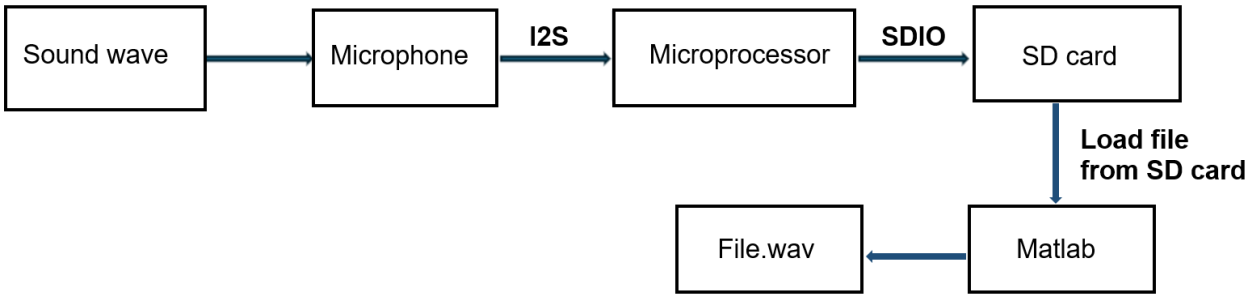
- ***Self-Evaluation of Contribution Percentage***

Each team member has evaluated their own contribution to the project based on the complexity and effort required for their assigned tasks.

2. GENERAL DESIGN

2.1 Hardware block diagram and functional description of each block

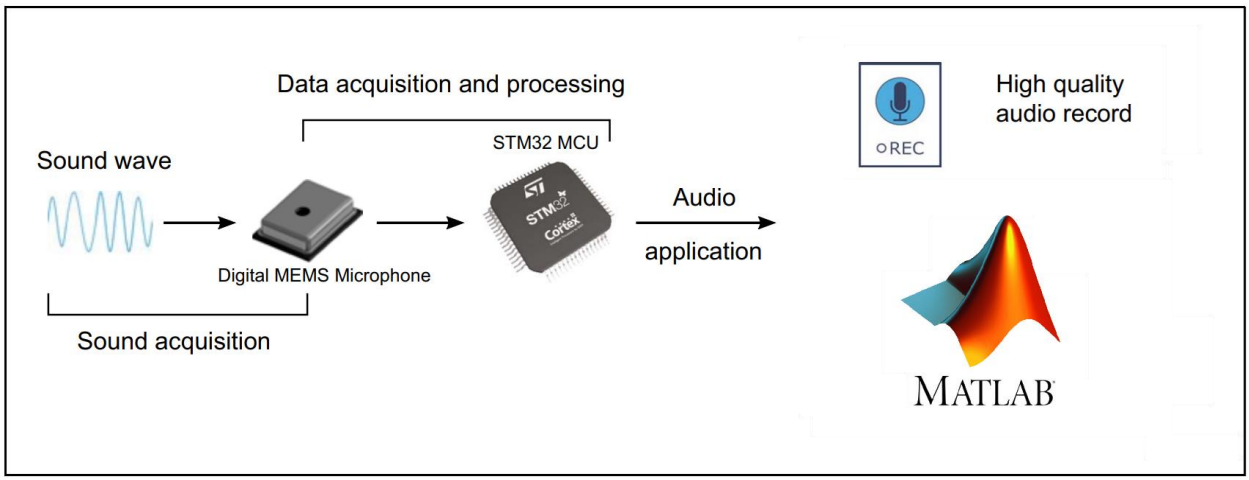
2.1.1 Hardware block diagram:



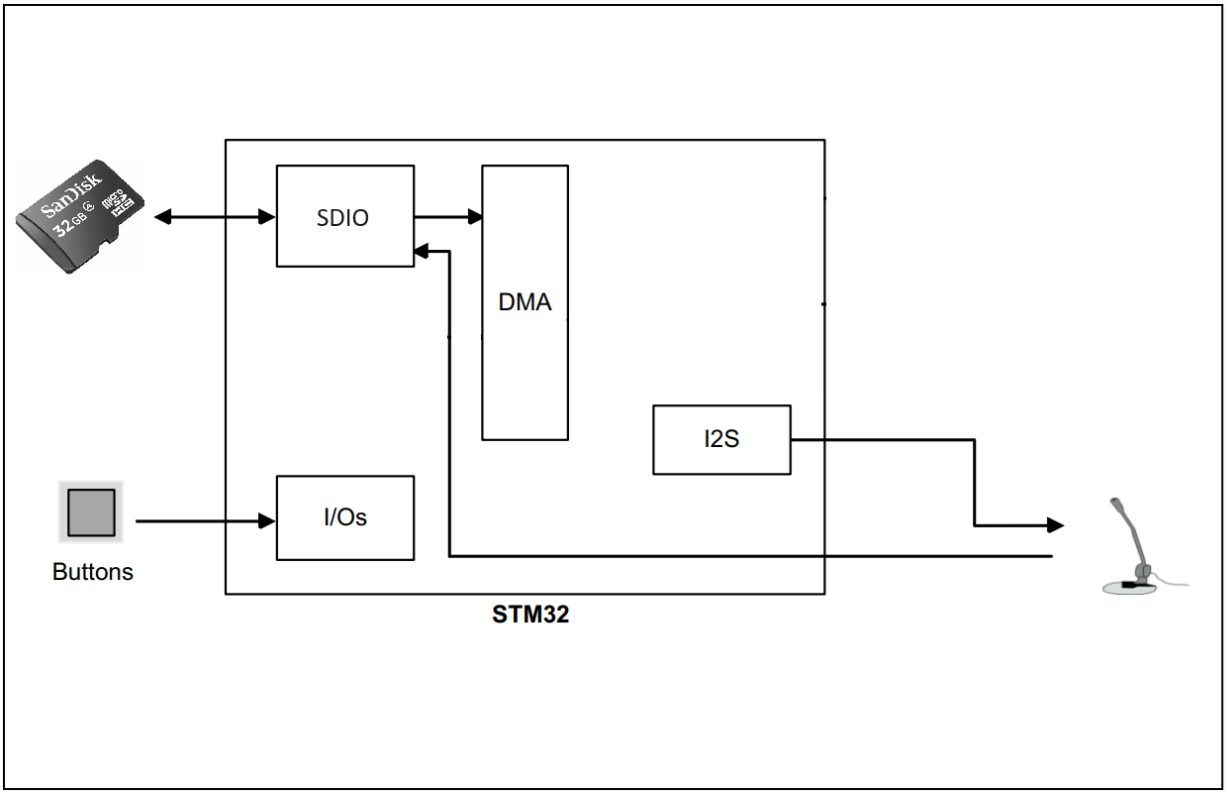
Fingure 1: Hardware block diagram

- Microphone: Receive sound wave signal and transmit to microprocessor
- Microprocessor: Receive data from microphone and write into MicroSD
- MicroSD card module: Store file .wav
- Matlab: Process file .wav

2.1.2 Functional description of each block:



Fingure 2:



Fingure 3:

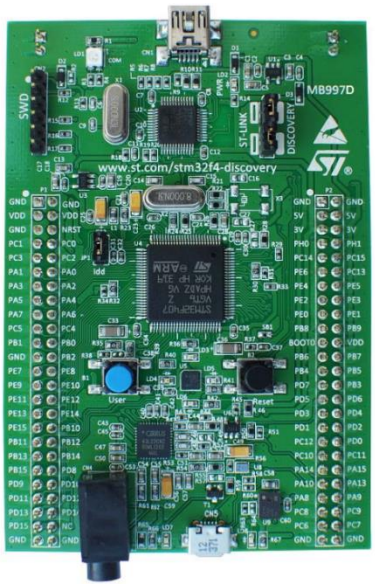
2.2 Pseudocode of the program

```
**Initialize MCU**

Clock_Configuration()
GPIO_Configuration(microphone_pin, SD_card_pins)
ADC_Configuration(microphone_channel, sample_rate)
Initialize_SD_Card()
**Initialize Recording Parameters**

recording_duration (in milliseconds)(optional)
```

```
sample_rate (e.g., 44100 Hz)
bit_depth (e.g., 16 bits)

**Audio Recording Loop**

WHILE(recording_duration not reached)
   CALL Read_ADC_Value(microphone_channel)
   STORE ADC_value in audio_buffer
   INCREMENT recording_elapsed_time
   IF (recording_elapsed_time >= recording_duration)
      BREAK loop

**Store Audio to SD Card**

CALL Open_SD_File(filename, WRITE_MODE)
FOR EACH sample in audio_buffer
   CONVERT sample to desired format (e.g., PCM)
   CALL Write_Byte_to_SD(converted_sample)
CALL Close_SD_File()

**De-initialize Peripherals**

CALL Stop_ADC_Conversion()
CALL Disable_GPIO_Clocks(microphone_pin, SD_card_pins)
```

3. DETAILED DESIGN

3.1 Selection of devices corresponding to blocks and equipment specifications



Fingure 4: STM32 F407E

Name	STM32F407E-DISCOVERY
Microcontroller	STM32F407VGT6
Power supply	3V and 5V

Flash memory	1-Mbyte
RAM	192-Kbyte

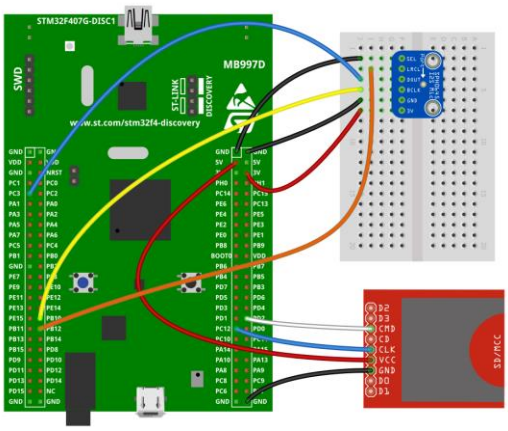
Fingure 5: Microphone SPH0648LM4H

Name	Microphone SPH0648LM4H
Type	MEMS Microphones
Frequency Range	1.024 MHz to 4096 MHz
Operating Supply current	600 uA
Operating Supply Voltage	1.8 V
Output Type	Digital, I2S
Voltage Supply	1.6 V - 3.6 V

Fingure 6: Module SDIO for SD card

Name	Module SDIO for SD card
Supported Interface	SDIO/SPI
Power Supply	3.3V-5V
Number of Pins	8 Pin

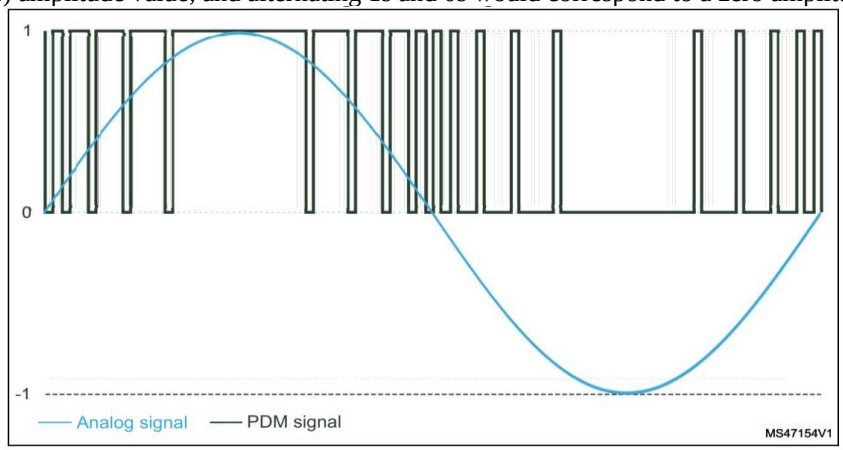
3.2 Diagram of the principle of circuit / wiring connection, table of diagrams of pins of main devices



Fingure 7: Diagram of the principle of circuit

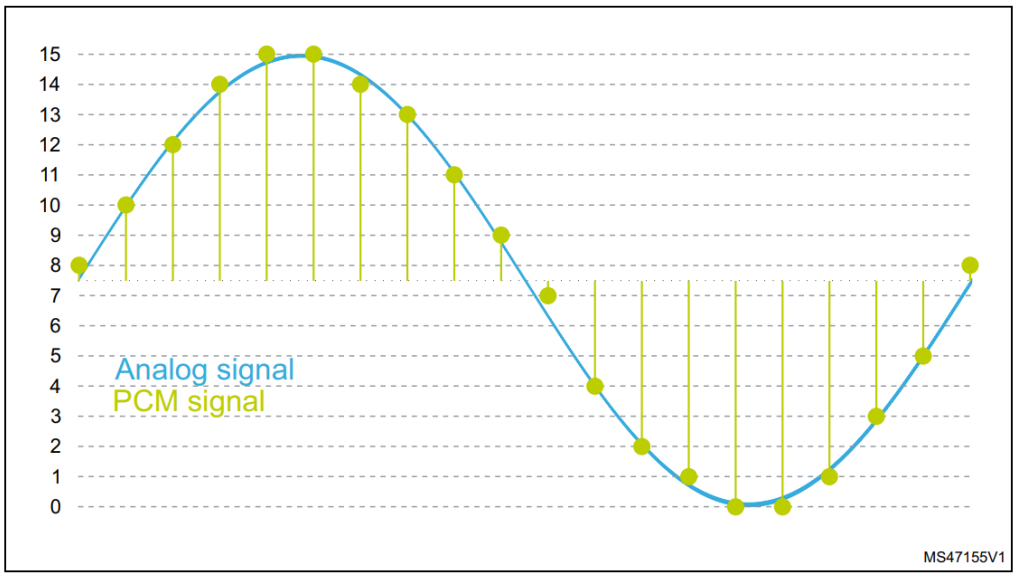
3.3 PDM and PCM

- PDM (Pulse density modulation) is a form of modulation used to represent an analog signal in the digital domain. It is a high frequency stream of 1-bit digital samples. In a PDM signal, the relative density of the pulses corresponds to the analog signal's amplitude. A large cluster of 1s correspond to a high (positive) amplitude value, when a large cluster of 0s would correspond to a low (negative) amplitude value, and alternating 1s and 0s would correspond to a zero amplitude value



- *Fingure 8:*

- In the PCM (Pulse code modulation) signal, specific amplitude values are encoded into pulses. A PCM stream has two basic properties that determine the stream's fidelity to the original analog signal: the sampling rate, the bit depth.The sampling rate is the number of samples of a signal that are taken per second to represent it digitally. The bit depth determines the number of bits of information in each sample.



Fingure 9:

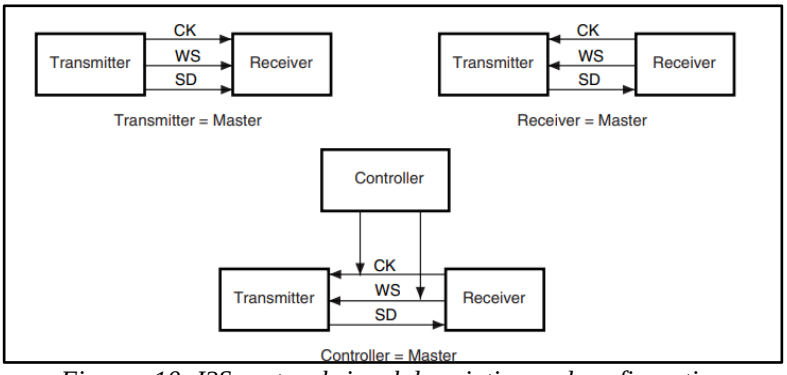
- PDM to PCM conversion: PDM frequency = Audio sampling frequency × decimation factor
- The decimation factor is generally in the range of 48 to 128. The decimation stage is preceded by a low-pass filter to avoid distortion from aliasing.

3.4 I2S overview

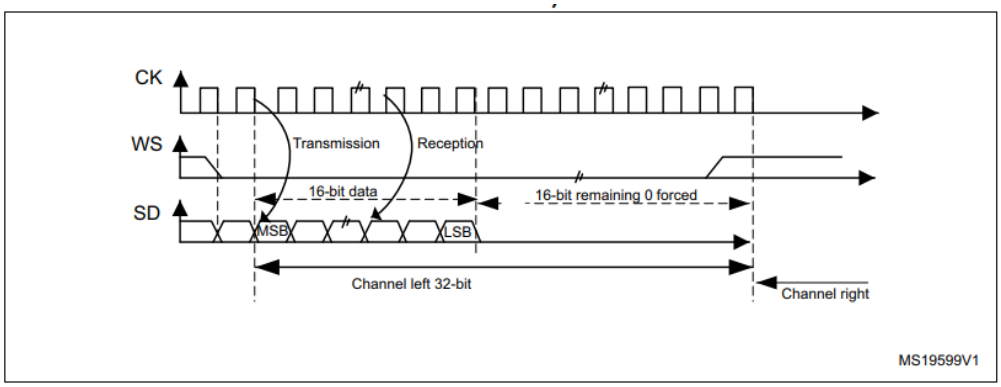
The protocol which is used to transmit digital audio data from one device to another device is known as I2S or Inter-IC Sound protocol. This protocol transmits PCM (pulse-code modulated) audio data from one IC to another within an electronic device. I2S plays a key role in transmitting audio files which are pre-recorded from an MCU to a DAC or amplifier. The I2S protocol is widely used to transfer audio data from a microcontroller/DSP (digital signal processor) to an audio codec, in order to play melodies or to capture sound from a microphone.

3-Wire Connection of I2S:

- SCK (Serial Clock) is the first line of the I2S protocol which is also known as BCLK or bit clock line which is used to obtain the data on a similar cycle. The serial clock frequency is simply defined by using the formula like Frequency = Sample Rate x Bits for each channel x no. of channels.
- WS: In the I2S communication protocol, the WS (word select) is the line which is also known as FS (Frame Select) wire that separates the right or left channel.
- SD (Serial Data) is the last wire where the payload is transmitted within 2 complements. So, it is very significant that the MSB is first transferred, because both the transmitter & receiver may include different word lengths. Thus, the transmitter or the receiver has to recognize how many bits are transmitted.

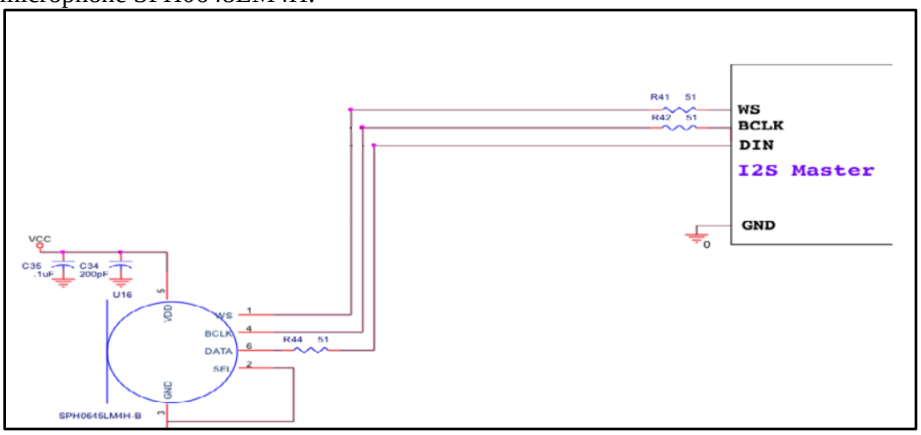


Fingure 10: I2S protocol signal description and configuration



Fingure 11: Writing data frame

Pinouts of microphone SPH0648LM4H:



Fingure 12: Pinouts of microphone SPH0648LM4H

- **3V** - this is the power supply. Technically it can be powered from as low as 1.6V to 3.6V but you'll need to make sure your logic level matches.
- **GND** - power and data ground
- **BCLK** - the bit clock, also known as the data clock - comes from the I2S main to tell the microphone it's time to transmit data.
- **DOUT** - the data output from the mic.
- **LRCLK** - the left/right clock, also known as **WS** (word select), this tells the mic when to start transmitting. When the **LRCLK** is low, the left channel will transmit. When LRCLK is high, the right channel will transmit.
- **SEL** - the channel select pin. By default this pin is low, so that it will transmit on the left channel mono. If you connect this to high logic voltage, the microphone will instantly start transmitting on the right channel.

3.5 DMA controller overview

-Direct memory access (DMA) is used in order to provide high-speed data transfer between peripherals and memory and between memory and memory. Data can be quickly moved by DMA without any CPU action. This keeps CPU resources free for other operations

-In this project, we use Peripheral-to-memory mode, so each time a peripheral request occurs, the stream initiates a transfer from the source to fill the FIFO.And the Increment mode is enabled at half-words, so the address of next transfer is the address of previous one incremented by 2 in DMA_SxCR register.

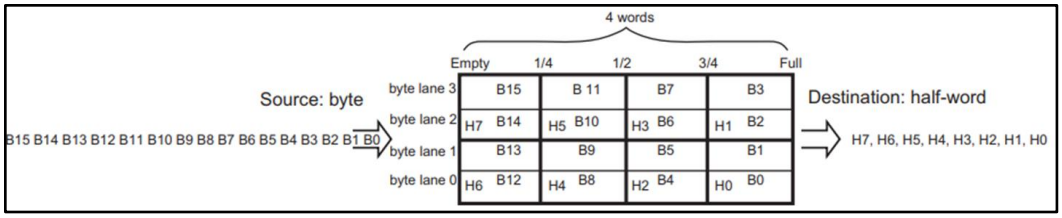


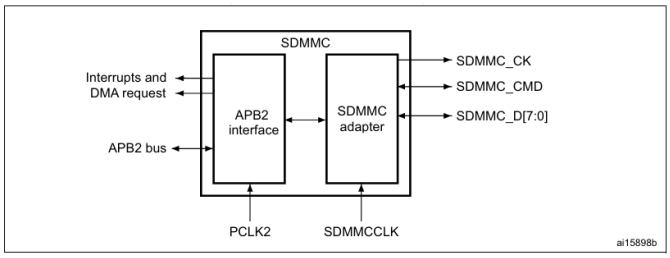
Figure 13: data location in FIFO with half- word mode.

- We also use the circular mode to handle circular buffers and continuous data flows , so number of data items to be transferred is automatically reloaded with the initial value programmed during the stream configuration phase, and the DMA requests continue to be served.

3.6 SDIO interface overview

3.6.1 What is the SDIO communication standard?

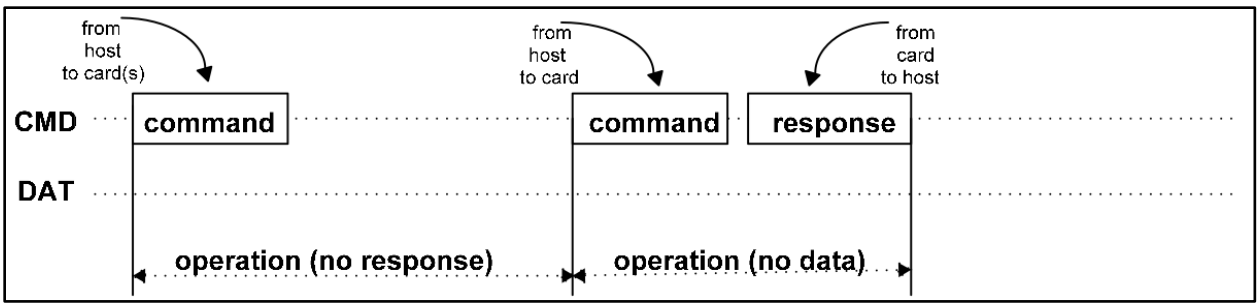
- SDIO is used to communicate between the APB2 Peripheral Bus and MultiMediaCards (MMC), which include SD cards, SDIO, and CE-ATA devices.
- The SDIO consists of two parts:
  - The SDIO adapter block provides all functions specific to the MMC/SD/SD I/O card such as the clock generation unit, command and data transfer.
  - The APB2 interface accesses the SDIO adapter registers and generates interrupt and DMA request signals.



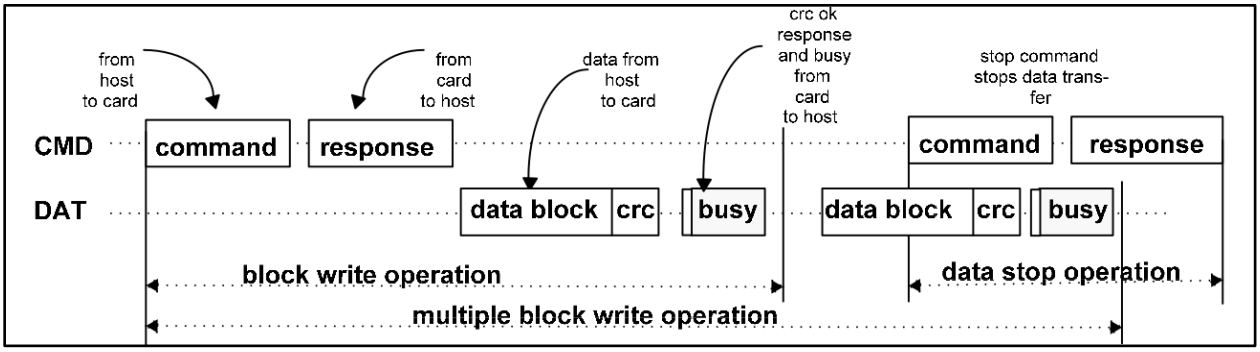
Fingure 13: SDIO block diagram

Pin	Direction	Description
SDIO_CK	Output	MultiMediaCard/SD/SDIO card clock. This pin is the clock from host to card.
SDIO_CMD	Bidirectional	MultiMediaCard/SD/SDIO card command. This pin is the bidirectional command/response signal.
SDIO_D[7:0]	Bidirectional	MultiMediaCard/SD/SDIO card data. These pins are the bidirectional databus.

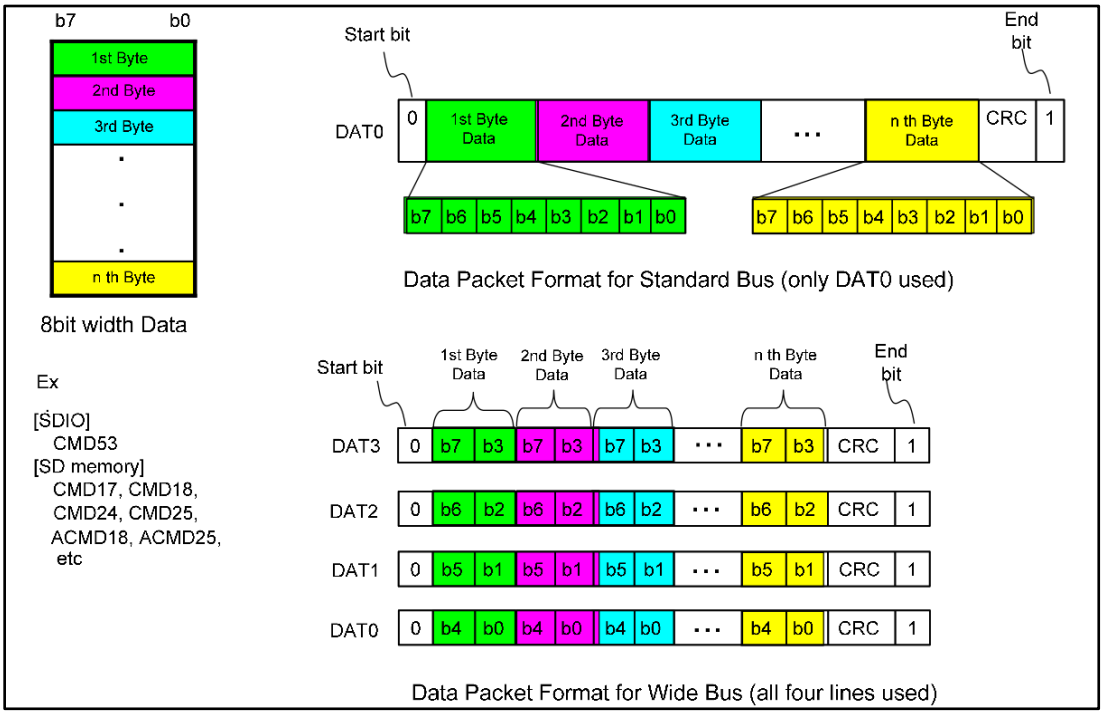
Fingure 14: SDIO I/O definitions



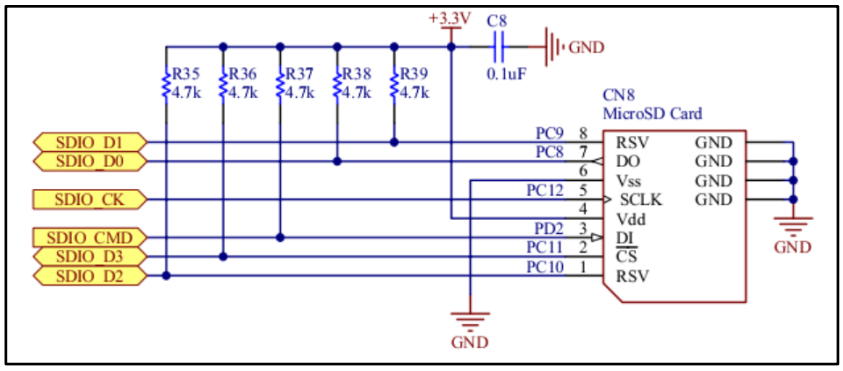
Fingure 15: "no response" and "no data" operations



Fingure 16: Block write operation



Fingure 17: Data packet format – usual data



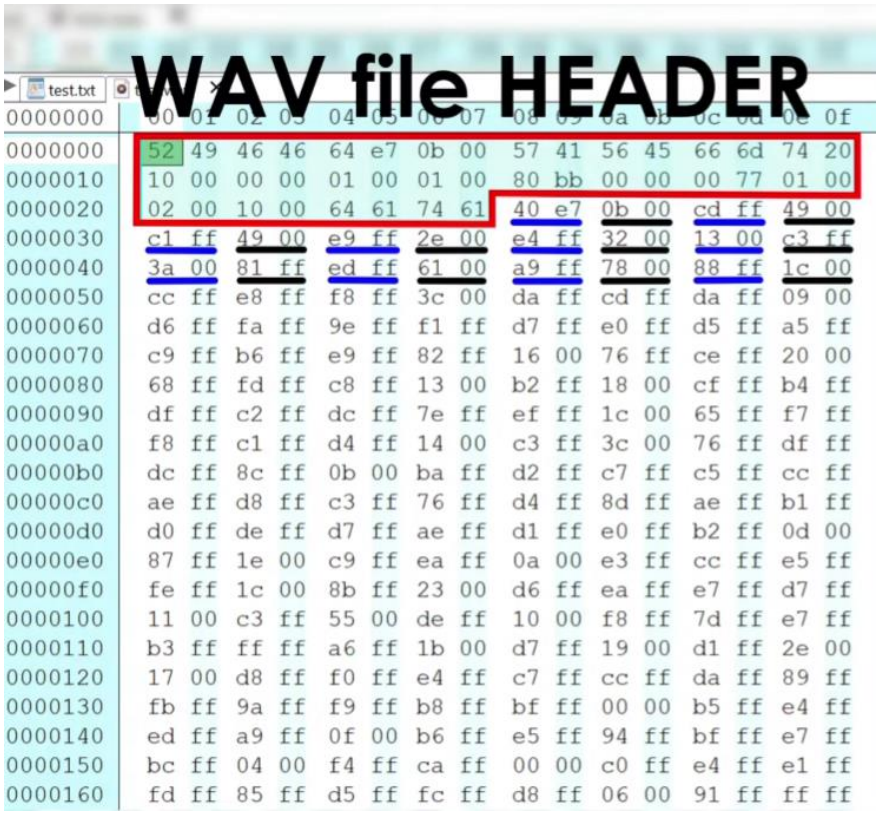
Fingure 18: Connecting SDIO Pins to a Memory Card

3.6.2 FATFS Library

- FatFS is a type of generic FAT filesystem module for embedded systems. FatFS is written in ANSI C and is separated from the I/O layer, making it hardware-independent.
- FatFS resides in the Middleware layer between the hardware and the Application layers in an embedded system. This means it is neither dependent on the hardware nor the software.
- The FATFS commands we commonly use are:
    - f_mount(): Register/Unregister a work area
    - f_open(): Open/Create a file
    - f_close(): Close a file
    - f_read(): Read a file
    - f_write(): Write a file
    - f_lseek(): Move read/write pointer, Expand a file size
    - f_truncate(): Truncate a file size
    - f_sync(): Flush cached data
    - f_opendir(): Open a directory

3.7 File .wav overview

- WAV, known as WAVE (Waveform Audio File Format), is a subset of Microsoft's Resource Interchange File Format (RIFF) specification for storing digital audio files. The format doesn't apply any compression to the bitstream and stores the audio recordings with different sampling rates and bitrates. It has been and is one of the standard formats for audio CDs. Wave files are larger in size as compared to new audio file formats such as MP3 which uses lossy compression to reduce the file size while maintaining the same audio quality.
- The WAVE file format, being a subset of Microsoft's RIFF specification, starts with a file header followed by a sequence of data chunks. A WAVE file has a single "WAVE" chunk which consists of two sub-chunks:
    + a "fmt" chunk - specifies the data format
    + a "data" chunk contains the actual sample data



Fingure 19: Wav file Header

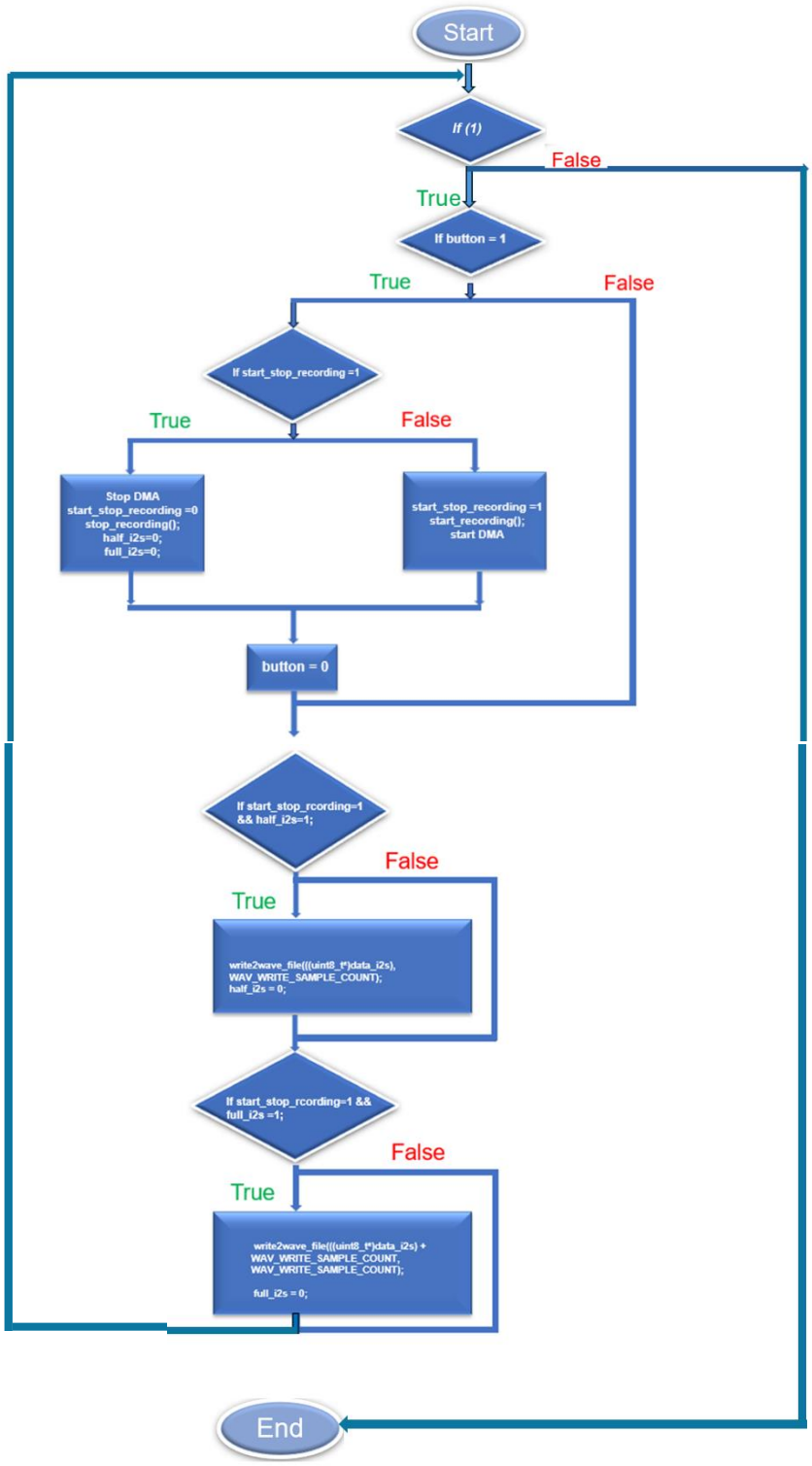
WAV file Header

0 - 3	"RIFF"	0x52, 0x49, 0x46, 0x46
4 - 7	size of the file in bytes	data_section size + 36
8 - 11	"WAVE"	0x57 ,0x41, 0x56, 0x45
12 - 15	"fmt "	0x66, 0x6d, 0x74, 0x20
16 - 19	Length of format data: 16	0x10, 0x00, 0x00, 0x00
20 - 21	type of format, pcm: 1	0x01 0x00
22 - 23	number of channels: 2	0x02 0x00
24 - 27	sample rate: 32 kHz	0x80, 0xbb, 0x00, 0x00
28 - 31	sample rate x bps x channels	0x00, 0xee, 0x02, 0x00
32 - 33	bps x channels: 4	0x04, 0x00
34 - 35	bits per sample: 16	0x10, 0x00
36 - 39	"data"	0x64, 0x61, 0x74, 0x61
40 - 43	size of the data section	data section size

Fingure 20: Wav file header

- Some concepts:
    + Sampling rate: number of samples per second that are taken of a waveform to create a discrete digital signal. The higher the sample rate, the more snapshots you capture of the audio signal. There are common frequencies such as 8000, 12000, 16000, 22050, 44100,... unit is Hz.
    + Bits per sample (or bit depth): When a signal is sampled, it needs to store the sampled audio information in bits. This is where the bit depth comes into place. The bit depth determines how much information can be stored. This is the value used to measure sound fluctuations, the larger the value, the higher the analysis resolution, the more strong sound produced. We often encounter 8-bit and 16-bit types.
    + Number of channels: There are type mono, which means we can only hear it with one speaker; and type stereo can output to two speakers at the same time.

3.8 Flowchart of the main function algorithm and other functions included in the program.



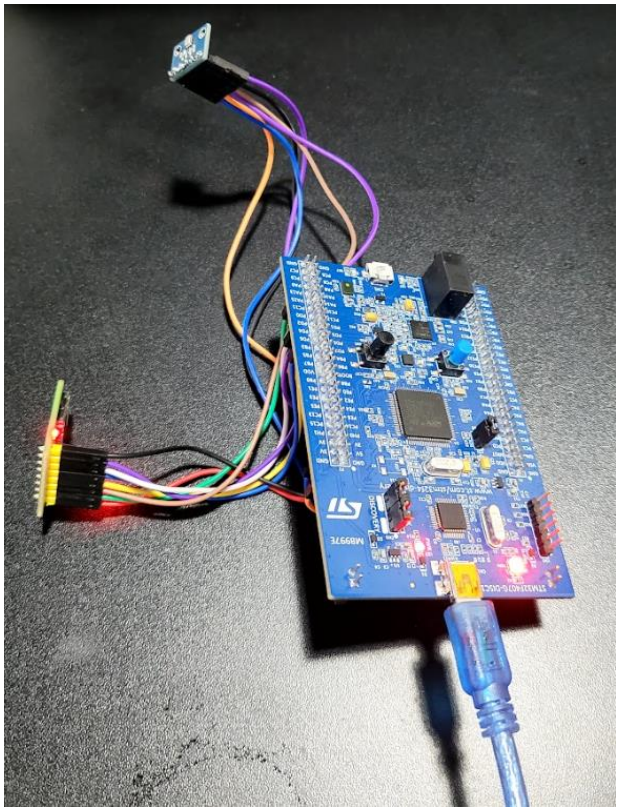
Fingure 21: flowchart

4. CONSTRUCTION

4.1 Configuration parameters of the peripherals of STM32

SDIO_test5.pdf

4.1.1 Model Hardware Result Image



4.1.2 Code the main function and other functions used (except for the generated functions of the HAL library)

Declaration

```
// 0 - 3   -> "RIFF"                                    {0x52, 0x49, 0x46, 0x46}
// 4 - 7   -> size of the file in bytes                 {data_section size + 36}
// 8 - 11  -> File type header, "WAVE"                  {0x57 ,0x41, 0x56, 0x45}
// 12 - 15 -> "fmt "                                    {0x66, 0x6d, 0x74, 0x20}
// 16 - 19 -> Length of format data                 16  {0x10, 0x00, 0x00, 0x00}
// 20 - 21 -> type of format, pcm is                1   {0x01 0x00}
// 22 - 23 -> number of channels                    2   {0x02 0x00}
// 24 - 27 -> sample rate,                          32 kHz  {0x80, 0x7d, 0x00, 0x00}
// 28 - 31 -> sample rate x bps x channels          19200   {0x00, 0xf4, 0x01, 0x00 }
// 32 - 33 -> bps * channels                        4   {0x04, 0x00}
// 34 - 35 -> bits per sample                       16  {0x10, 0x00}
// 36 - 39 -> "data"                                    {0x64, 0x61, 0x74, 0x61}
// 40 - 43 -> size of the data section                  {data section size}
//  data
```

```
static uint8_t wav_file_header[44]={0x52, 0x49, 0x46, 0x46, 0xa4, 0xa9, 0x03,
0x00, 0x57 ,0x41, 0x56, 0x45, 0x66, 0x6d,
            0x74, 0x20, 0x10, 0x00, 0x00, 0x00, 0x01, 0x00, 0x02, 0x00, 0x80,
0x7d, 0x00, 0x00, 0x00, 0xf4, 0x01, 0x00,
            0x04, 0x00, 0x10, 0x00, 0x64, 0x61, 0x74, 0x61, 0x80, 0xa9, 0x03,
0x00};
static FRESULT sd_result;
static FATFS sdCard;
static FIL wavFile;
static uint32_t wav_file_size;
static uint8_t first_time = 0;
int16_t data_i2s[WAV_WRITE_SAMPLE_COUNT];
volatile int16_t sample_i2s;
volatile uint8_t button_flag, start_stop_recording;
volatile uint8_t  half_i2s, full_i2s;
```

start_recording ()

```
void start_recording(uint32_t frequency)
{
	static char file_name[] = "w_000.wav";
	static uint8_t file_counter = 10;
	int file_number_digits = file_counter;
	uint32_t byte_rate = frequency * 2 * 2;
	wav_file_header[24] = (uint8_t)frequency;
	wav_file_header[25] = (uint8_t)(frequency >> 8);
	wav_file_header[26] = (uint8_t)(frequency >> 16);
	wav_file_header[27] = (uint8_t)(frequency >> 24);
	wav_file_header[28] = (uint8_t)byte_rate;
	wav_file_header[29] = (uint8_t)(byte_rate >> 8);
	wav_file_header[30] = (uint8_t)(byte_rate >> 16);
	wav_file_header[31] = (uint8_t)(byte_rate >> 24);
	// defining a wave file name
	file_name[4] = file_number_digits%10 + 48;
	file_number_digits /= 10;
	file_name[3] = file_number_digits%10 + 48;
	file_number_digits /= 10;
	file_name[2] = file_number_digits%10 + 48;
	printf("file name %s \n", file_name);
	file_counter++;
	// creating a file
	sd_result = f_open(&wavFile ,file_name, FA_WRITE|FA_CREATE_ALWAYS);
	if(sd_result != 0)
	{
		printf("error in creating a file: %d \n", sd_result);
		while(1);
	}
	else
	{
		printf("succeeded in opening a file \n");
	}
	wav_file_size = 0;
}
```

write2wave_file()

```
void write2wave_file(uint8_t *data, uint16_t data_size)
{
	uint32_t temp_number;
	printf("w\n");
	if(first_time == 0)
	{
		for(int i = 0; i < 44; i++)
		{
			*(data + i) = wav_file_header[i];
		}
		first_time = 1;
	}
	sd_result wav_file_size += data_size;
= f_write(&wavFile,(void *)data, data_size,(UINT*)&temp_number);
	if(sd_result != 0)
	{
		printf("error in writing to the file: %d \n", sd_result);
		while(1);
	}
	}
```

stop_recording()

```
void stop_recording()
{
	uint16_t temp_number;
	// updating data size sector
	wav_file_size -= 8;
	wav_file_header[4] = (uint8_t)wav_file_size;
	wav_file_header[5] = (uint8_t)(wav_file_size >> 8);
	wav_file_header[6] = (uint8_t)(wav_file_size >> 16);
	wav_file_header[7] = (uint8_t)(wav_file_size >> 24);
	wav_file_size -= 36;
	wav_file_header[40] = (uint8_t)wav_file_size;
	wav_file_header[41] = (uint8_t)(wav_file_size >> 8);
	wav_file_header[42] = (uint8_t)(wav_file_size >> 16);
	wav_file_header[43] = (uint8_t)(wav_file_size >> 24);
	// moving to the beginning of the file to update the file format
	f_lseek(&wavFile, 0);
	f_write(&wavFile,(void *)wav_file_header,
sizeof(wav_file_header),(UINT*)&temp_number);
	if(sd_result != 0)
	{
		printf("error in updating the first sector: %d \n", sd_result);
		while(1);
	}
	f_close(&wavFile);
	first_time = 0;
	printf("closed the file \n");
```

```
}
```

main()

```
int main(void)
{

/* USER CODE BEGIN WHILE */
 while (1)
 {
        if(button_flag)
        {
              if(start_stop_recording)
              {
                    HAL_I2S_DMAStop(&hi2s2);
                    start_stop_recording = 0;
                    stop_recording();
                    half_i2s = 0;
                    full_i2s = 0;
                    printf("stop recording \n");
              }
              else
              {
                    start_stop_recording = 1;
                    start_recording(I2S_AUDIOFREQ_32K);
                    printf("start_recording %d and %d\n", half_i2s,
full_i2s);
                    HAL_I2S_Receive_DMA(&hi2s2, (uint16_t *)data_i2s,
sizeof(data_i2s)/2);
              }
              button_flag = 0;
        }
   /* USER CODE END WHILE */
   /* USER CODE BEGIN 3 */
        if(start_stop_recording == 1 && half_i2s == 1)
              {
                    write2wave_file(((uint8_t*)data_i2s),
WAV_WRITE_SAMPLE_COUNT);
                    half_i2s = 0;
              }
              if(start_stop_recording == 1 && full_i2s == 1)
              {

      write2wave_file(((uint8_t*)data_i2s)+WAV_WRITE_SAMPLE_COUNT,
WAV_WRITE_SAMPLE_COUNT);
                    full_i2s = 0;
              }
 }
}
```

Interrupt

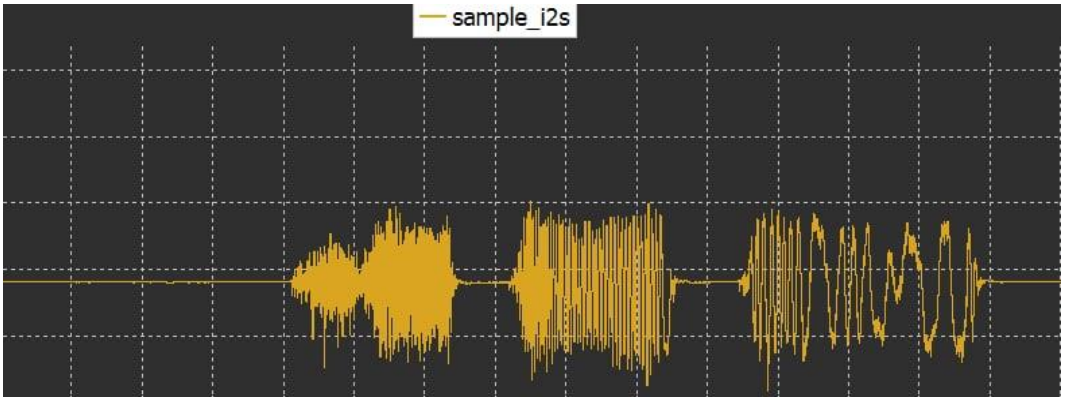
```
int _write(int file, char *ptr, int len)
{
	int DataIdx;
	for (DataIdx = 0; DataIdx < len; DataIdx++)
	{
		ITM_SendChar(*ptr++);
	}
	return len;
}
//    l,r,l,r,l,
void HAL_I2S_RxCpltCallback(I2S_HandleTypeDef *hi2s)
{
	full_i2s = 1;
}
void HAL_I2S_RxHalfCpltCallback(I2S_HandleTypeDef *hi2s)
{
	//sample_i2s = data_i2s[0];
	half_i2s = 1;
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
	if(GPIO_Pin == B1_Pin)
	{
		button_flag = 1;
	}
}
```

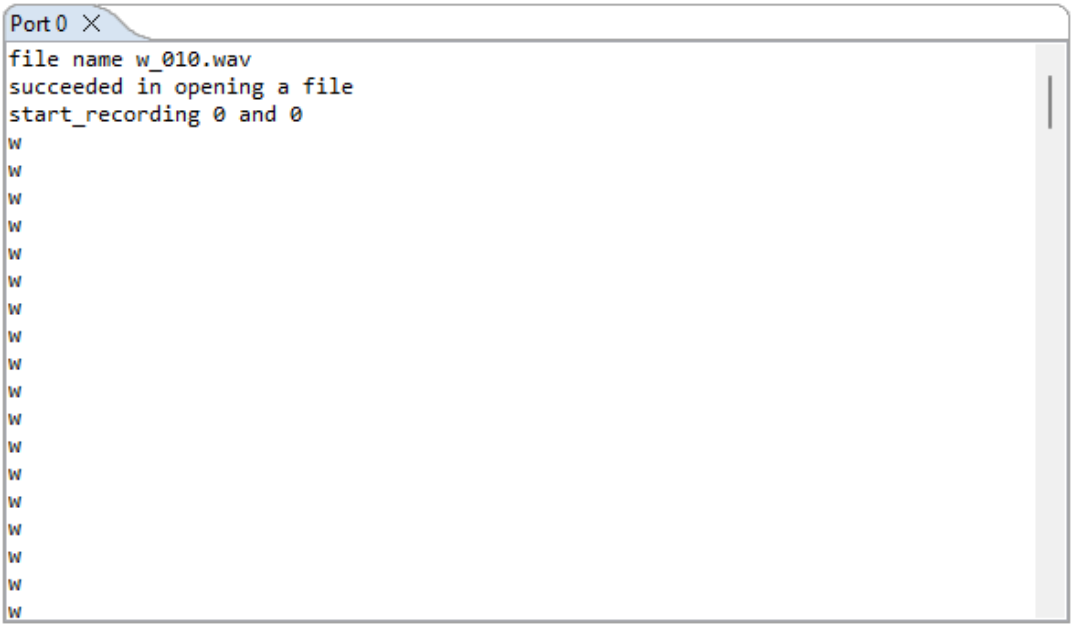
Matlab code:

```
[y_preemphwav, Fs] = audioread('w_012.wav');%Read audio file on matlab
figure
%plot Magnitude of Audio in time domain
    subplot(3,1,1);
    myAudio=y_preemphwav(:,1);
    plot(myAudio);
    ylabel('Magnitude');
    xlabel('Time(ms)');
    title('Time Domain');
%plot Magnitude Spectrum of Audio in frequency domain
    subplot(3,1,2);
    L = length(myAudio);  % Length of the signal
    Y = fft(myAudio);     % Compute the FFT
    P2 = abs(Y );           % Two-sided spectrum P2
    P1 = P2(1:L/2+1);          % Single-sided spectrum P1
    P1(2:end-1) = 2 * P1(2:end-1);  % Compensate for the single side
    f = Fs * (0:(L/2)) / L;    % Frequency vector
    plot(f, P1,'LineWidth',2);
    title('Amplitude Spectrum');
    xlabel('Frequency (Hz)');
    ylabel('|Audio|');
    soundsc(myAudio,Fs);
    pause;
% Filter Audio with IIR filter
```

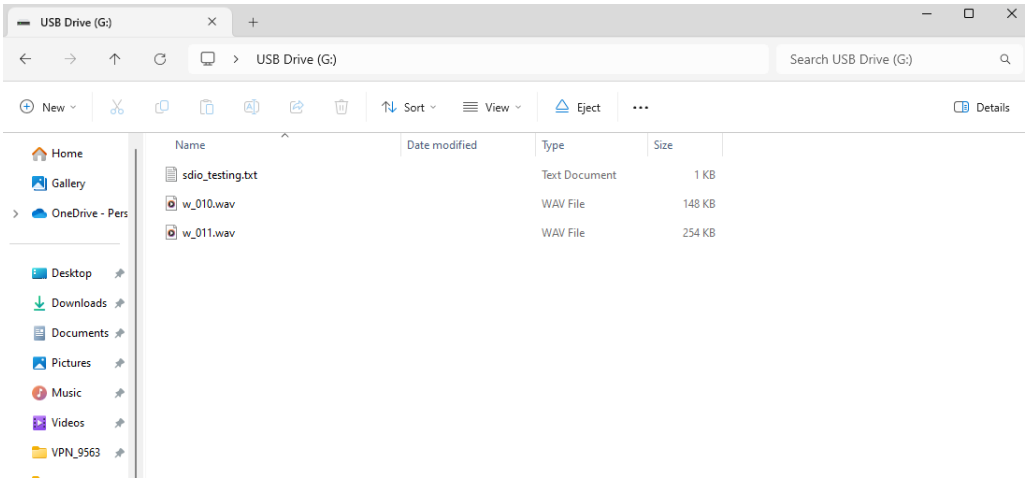
```
    subplot(3,1,3);
    filtered_Audio = filter(b,a,myAudio);
    soundsc(filtered_Audio,Fs)
    L2 = length(filtered_Audio);  % Length of the signal
    Y2 = fft(filtered_Audio);     % Compute the FFT
    P4 = abs(Y2 );           % Two-sided spectrum P4
    P3 = P4(1:L/2+1);          % Single-sided spectrum P3
    P3(2:end-1) = 2 * P3(2:end-1);  % Compensate for the single side
    f2 = Fs * (0:(L2/2)) / L2;    % Frequency vector
    plot(f2, P3,'LineWidth',2);
    title('Amplitude Spectrum After Filter');
    xlabel('Frequency (Hz)');
    ylabel('|Audio|');
    suptitle('Plot Signal Audio');
%Store audio after filter in newfile
    audiowrite('STM32audio.wav',filtered_Audio, Fs);
```

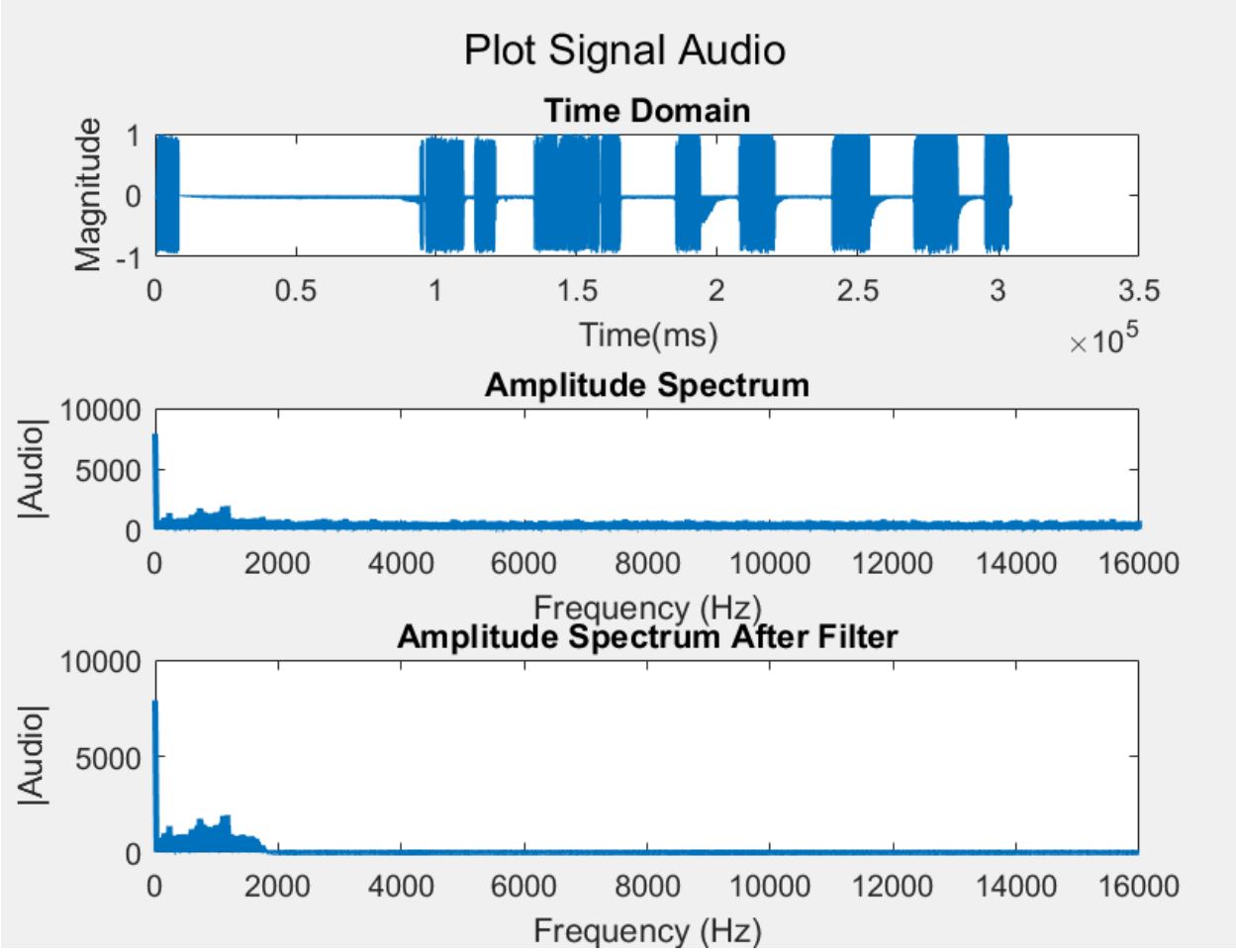
5. TESTING






Port 0
w
w
w
w
w
w
w
w
w
w
w
w
w
w
w
w
w
w
closed the file
stop recording


USB Drive (G:)
USB Drive (G:)
Search USB Drive (G:)
New
Sort
View
Eject
Details
Home
Gallery
OneDrive - Pers
Desktop
Downloads
Documents
Pictures
Music
Videos
VPN_9563
Name
Date modified
Type
Size
sdio_testing.txt
Text Document
1 KB
w_010.wav
WAV File
148 KB
w_011.wav
WAV File
254 KB



=> Comment : Human sound is often in range 250 - 4.000 Hz**[1],** so we can use the filter to remove noise with higher frequencies in order to obtain the clear sound. In this example with our sound, we use the low pass filter to remove noise that has a frequency higher than 1500 Hz, and after filter, the sound is better than raw recorded audio.

5. CONCLUSIONS
   - Audio Recording: The system interfaces with a microphone, enabling the capture of audio segments.
   - Storage: Captured audio is stored on microSD for later retrieval.
   - However, the recorded audio files (.wav format) exhibit noise that requires reduction.
   - Currently, the system relies on post-processing with MATLAB for noise reduction. This approach involves filtering the captured audio signal to remove unwanted noise components.

     => Future work: This research identifies a critical area for future development: on-board noise reduction. The primary objective is to implement noise reduction algorithms directly on the recording device, eliminating the need for post-processing with MATLAB.
6. REFERENCE

   - https://www.st.com/resource/en/application_note/dm00040802-audio-playback-and-recording-using-the-stm32f4discovery-stmicroelectronics.pdf
   - AN4309_V2.book (st.com)
   - https://www.st.com/resource/en/application_note/an5027-interfacing-pdm-digital-microphones-using-stm32-mcus-and-mpus-stmicroelectronics.pdf

- STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs - Reference manual
- STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm®-based 32-bit MCUs - Reference manual
- https://www.st.com/resource/en/reference_manual/rm0090-stm32f405415-stm32f407417-stm32f427437-and-stm32f429439-advanced-armbased-32bit-mcus-stmicroelectronics.pdf
- https://khuenguyencreator.com/lap-trinh-stm32-sdio-doc-ghi-du-lieu-vao-the-nho-sd-card/
- https://community.st.com/t5/stm32-mcus/how-to-create-a-file-system-on-a-sd-card-using-stm32cubeide/ta-p/49830
- https://deepbluembedded.com/stm32-sdio-sd-card-example-fatfs-tutorial/#stm32-sdio-secure-digital-io
- **[1]**https://www.vinmec.com/vi/tin-tuc/thong-tin-suc-khoe/thinh-luc-la-gi/#:~:text=Ti%E1%BA%BFng%20n%C3%B3i%20con%20ng%C6%B0%E1%BB%9Di%20n%E1%BA%B1m%20trong%20v%C3%B9ng%20nh%E1%BA%A1y,n%C3%B3i%20v%E1%BB%ABa%2055%20dB%2C%20n%C3%B3i%20to%2070%20dB%29.

7. CODE

Vi su li