Shahjalal University of Science and Technology, Sylhet

Department of Electrical and Electronic Engineering

Project Report

Audio classification with STM32f746G

Submitted By:

Group: ODD
Session : 2021-22

Submitted To:

Dr. Md Rasedujjaman
Associate Professor
Dept. of EEE
SUST, Sylhet.

Course Title: Digital Signal Processing I Lab
Course Code: EEE 322

Date of Submission: 9 December 2025

Odd Group Registration Numbers

Odd Group	Registration Number	Mark
1	2021338003	10
	2021338043	10
	2021338044	10
	2021338045	15
3	2021338008	15
	2021338018	10
	2021338019	10
	2021338031	10
5	2021338036	10
	2021338040	10
	2021338046	10
	2021338048	12
7	2021338004	10
	2021338006	10
	2021338035	10
	2021338037	10
9	2021338022	10
	2021338024	10
	2021338026	10
	2021338032	10.5
11	2021338028	10
	2021338030	13
	2021338038	10
	2021338039	10
13	2021338002	10
	2021338033	10

Contents

Objective:

Speech carries unique vocal characteristics that can be used to identify an individual. This project presents a real-time voice identity recognition system implemented on the STM32F746G-DISCO board. Speech is captured using the onboard MEMS microphone, converted from PDM to PCM, processed to extract Mel-Frequency Cepstral Coefficients (MFCC), and classified using a machine-learning model trained in Python. The deployed model runs on TensorFlow Lite Micro and displays the detected speakers name on the integrated LCD. The system demonstrates the feasibility of combining DSP techniques and embedded machine learning on a low-power microcontroller platform.

.

1 Introduction

The main objective of this work is to apply the theoretical and practical knowledge gained from the Digital Signal Processing (DSP) Laboratory into a real embedded application. The project involves several DSP operations such as sampling, framing, windowing, feature extraction, and embedded inference. It also demonstrates the complete workflow of building a machine learning model in Python, converting it to TensorFlow Lite, and deploying it efficiently on a resource-constrained hardware platform. The STM32F746NG microcontroller is powered by an ARM Cortex-M7 core running at 216 MHz, which provides sufficient computational resources to perform real-time audio acquisition, MFCC computation, and neural-network inference. This project also explores the challenges of deploying a trained Python model into an embedded C environment using TensorFlow Lite Micro, ensuring efficient memory usage and low-latency performance. As the Group Leader, I supervised the system design, coordinated dataset collection, implemented Python-based model training, integrated DSP algorithms, and optimized embedded inference. I ensured the seamless interaction of hardware, software, and machine-learning modules to achieve a fully functional real-time speaker identification system. Overall, this project successfully demonstrates how embedded systems and machine-learning techniques can be combined to create an intelligent, real-time, resource-efficient voice identity recognition platform.

2 Project Objectives

The key objectives of this project are:

1. To design and implement a real-time speaker recognition system on STM32F746G-DISCO.
2. To record speech signals using the on-board MEMS microphone.
3. To compute MFCC features for speaker identity extraction.
4. To develop a Python-based machine-learning model.
5. To convert the trained model to TensorFlow Lite Micro format.
6. To run real-time inference on the STM32 microcontroller.
7. To display the speakers name on the TFT LCD.
8. To evaluate performance, latency, and accuracy.

3 Background Study

Speaker recognition systems rely on extracting unique vocal features from speech signals. Traditional DSP techniques such as spectral analysis were widely used, but MFCC became the standard due to its alignment with human auditory perception.With microcontrollers becoming more powerful, embedded machine learning enables real-time classification on devices like STM32. TensorFlow Lite Micro is specifically optimized for resource-limited hardware.

4 Hardware Used

- **STM32F746G-DISCO Board:** ARM Cortex-M7 @ 216 MHz, 1 MB Flash, 320 KB SRAM, STM32F746NGH6 MCU (STM32F7 series)
- **Digital MEMS Microphone:** MP34DT01, PDM output, used for speech capture
- **TFT LCD Display:** 4.3-inch, 480×272 resolution, for displaying speaker identity

- **Audio Subsystem:** WM8994 codec, ADC, DAC, SAI interface; PDM-to-PCM conversion
- **Laptop/PC:** Used for dataset creation and model training (OS: Windows/Linux)
- **ST-Link Cable:** For powering, debugging, and flashing STM32 board

5 Software and Libraries

- **STM32CubeIDE:** v1.19 ; IDE for writing, compiling, and flashing code
- **STM32CubeMX:** v6.6 ; configuration tool for peripherals
- **Jupyter Notebook:** v6.5+; used for dataset preprocessing, plotting signals, pole-zero plots
- **HAL BSP Library:** STM32CubeF7 v1.26.0; provides hardware abstraction layer for peripherals
- **STM32 X-CUBE-AI:** v6.1.0; converts trained ML models (TensorFlow/TFLite) to optimized C code for STM32
- **Librosa:** v0.10.0; Python library for audio processing and feature extraction
- **NumPy:** v1.25.0; numerical computing in Python
- **TensorFlow / TensorFlow Lite:** TF 2.15.0 / TFLite 2.15.0; model training and inference
- **Optional: CMSIS DSP Library:** v5.9.0; signal processing functions for STM32
- **Optional: TouchGFX:** v4.23.0; GUI development on STM32 TFT displays

6 GPIO Pin Configuration for STM32F746G-DISCO

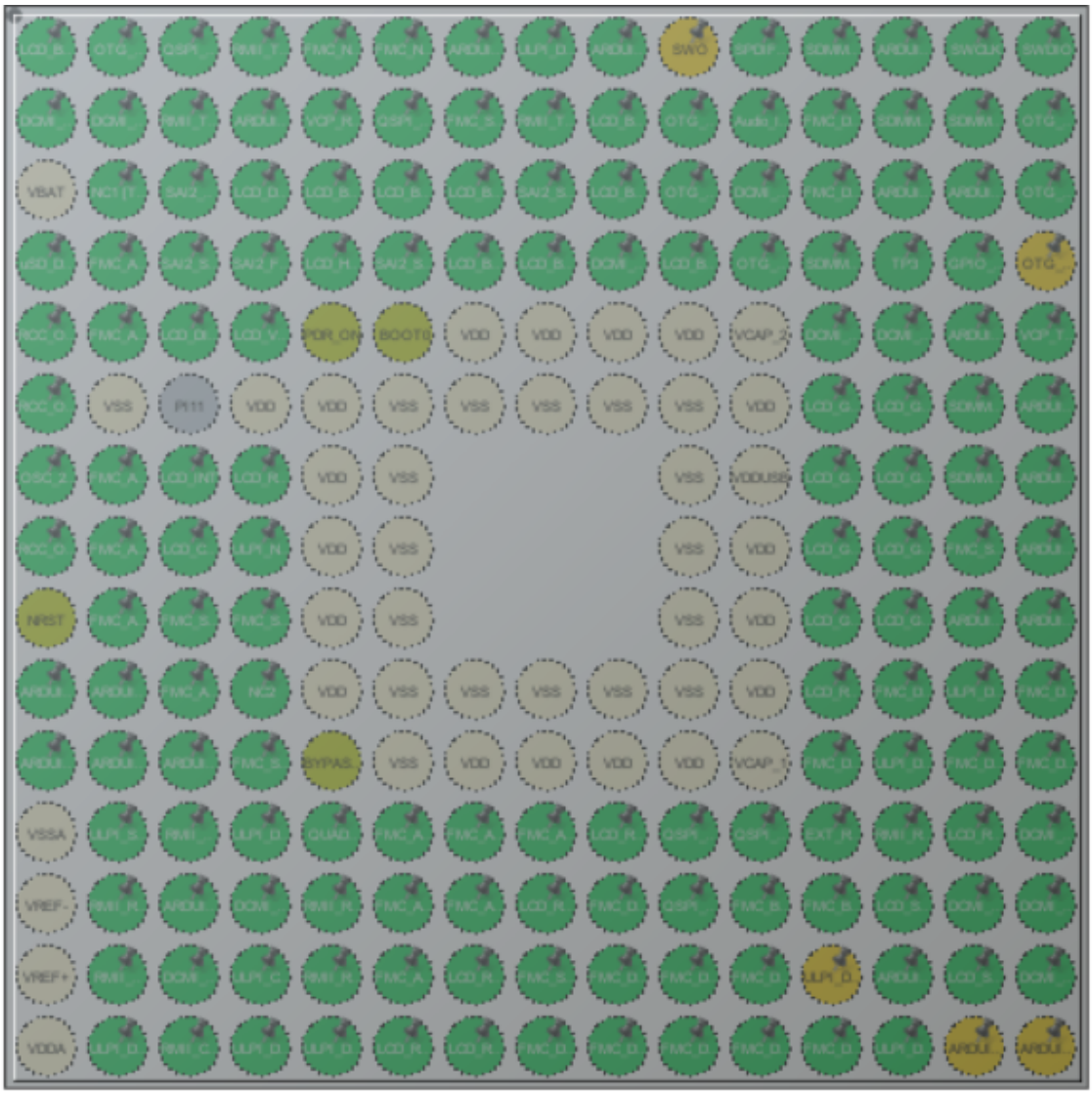

Figure 1: Top view

This section summarizes the GPIO pin assignments and modes used in the audio classification project on the STM32F746G-DISCO development board. Each subsection contains a table that is placed immediately after the subsection header so numbering and layout match the project source code.

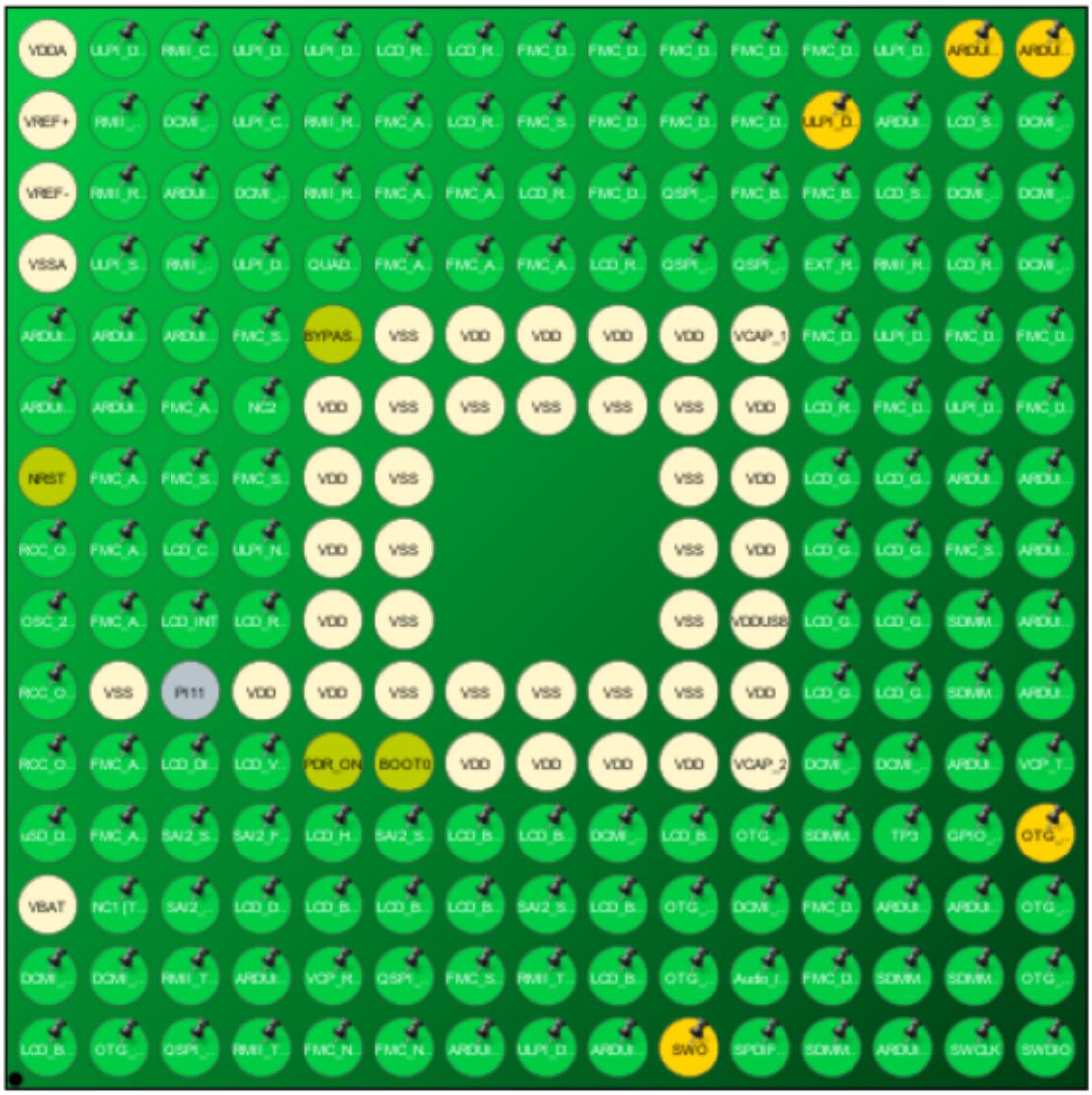

Figure 2: Bottom view

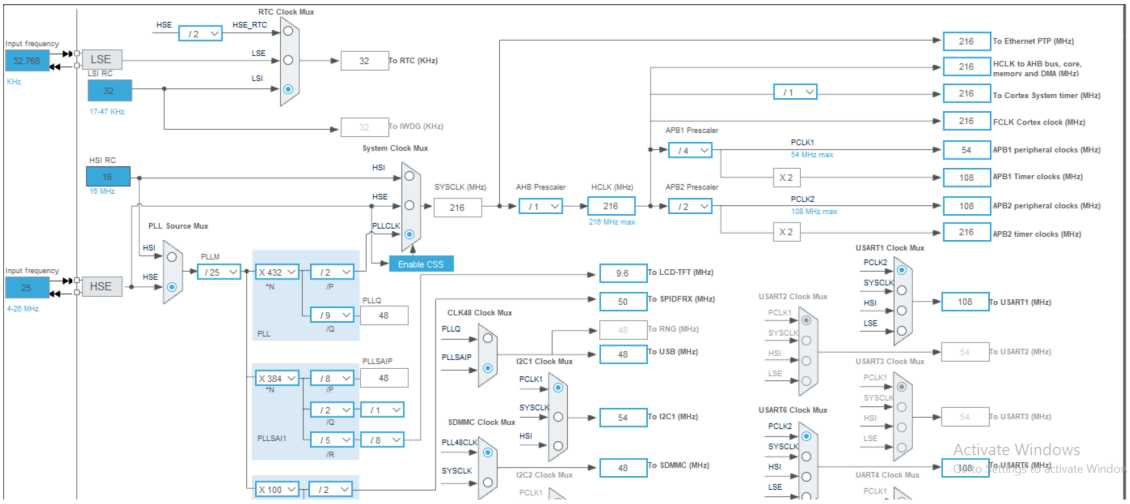

Figure 3: clock set up

6.1 QUADSPI Interface

Table 1: QUADSPI pin configuration

Pin	Signal	Mode	Alternate Function
PB2	QUADSPI_CLK	AF Push-Pull	AF9
PB6	QUADSPI_NCS	AF Push-Pull	AF10
PD11	QUADSPI_IO0	AF Push-Pull	AF9
PD12	QUADSPI_IO1	AF Push-Pull	AF9
PE2	QUADSPI_IO2	AF Push-Pull	AF9
PD13	QUADSPI_IO3	AF Push-Pull	AF9

6.2 SDMMC1 Interface

Table 2: SDMMC1 pin configuration

Pin	Signal	Mode	Alternate Function
PC8	SDMMC1_D0	AF Push-Pull	AF12
PC9	SDMMC1_D1	AF Push-Pull	AF12
PC10	SDMMC1_D2	AF Push-Pull	AF12
PC11	SDMMC1_D3	AF Push-Pull	AF12
PC12	SDMMC1_CLK	AF Push-Pull	AF12
PD2	SDMMC1_CMD	AF Push-Pull	AF12

6.3 I2C3 Touch Controller

Table 3: I2C3 (touch controller) pin configuration

Pin	Signal	Mode	Alternate Function
PH7	I2C3_SCL	AF Open-Drain + Pull-up	AF4
PH8	I2C3_SDA	AF Open-Drain + Pull-up	AF4
PI13	Touch INT	GPIO Input	—

6.4 SAI2 Audio Interface

Table 4: SAI2 pin configuration

Pin	SAI2 Signal	Mode	Speed	Used
PG10	SAI2_SD_B	Alternate Function Push-Pull	Low	true
PI4	SAI2_MCLK_A	Alternate Function Push-Pull	Low	true
PI5	SAI2_SCK_A	Alternate Function Push-Pull	Low	true
PI6	SAI2_SD_A	Alternate Function Push-Pull	Low	true
PI7	SAI2_FS_A	Alternate Function Push-Pull	Low	true

6.5 LTDC Display Interface

Table 5: LTDC pin configuration for RK043FN48H display

Pin	LTDC Signal	Mode	Speed	Used
PE4	LTDC_B0	Alternate Function Push-Pull	Low	true
PG12	LTDC_B4	Alternate Function Push-Pull	Low	true
PI9	LTDC_VSYNC	Alternate Function Push-Pull	Low	true
PI10	LTDC_HSYNC	Alternate Function Push-Pull	Low	true
PI14	LTDC_CLK	Alternate Function Push-Pull	Low	true
PI15	LTDC_R0	Alternate Function Push-Pull	Low	true
PJ0	LTDC_R1	Alternate Function Push-Pull	Low	true
PJ1	LTDC_R2	Alternate Function Push-Pull	Low	true
PJ2	LTDC_R3	Alternate Function Push-Pull	Low	true
PJ3	LTDC_R4	Alternate Function Push-Pull	Low	true
PJ4	LTDC_R5	Alternate Function Push-Pull	Low	true
PJ5	LTDC_R6	Alternate Function Push-Pull	Low	true
PJ6	LTDC_R7	Alternate Function Push-Pull	Low	true
PJ7	LTDC_G0	Alternate Function Push-Pull	Low	true
PJ8	LTDC_G1	Alternate Function Push-Pull	Low	true
PJ9	LTDC_G2	Alternate Function Push-Pull	Low	true
PJ10	LTDC_G3	Alternate Function Push-Pull	Low	true
PJ11	LTDC_G4	Alternate Function Push-Pull	Low	true
PJ13	LTDC_B1	Alternate Function Push-Pull	Low	true
PJ14	LTDC_B2	Alternate Function Push-Pull	Low	true
PJ15	LTDC_B3	Alternate Function Push-Pull	Low	true
PK0	LTDC_G5	Alternate Function Push-Pull	Low	true
PK1	LTDC_G6	Alternate Function Push-Pull	Low	true
PK2	LTDC_G7	Alternate Function Push-Pull	Low	true
PK4	LTDC_B5	Alternate Function Push-Pull	Low	true
PK5	LTDC_B6	Alternate Function Push-Pull	Low	true
PK6	LTDC_B7	Alternate Function Push-Pull	Low	true
PK7	LTDC_DE	Alternate Function Push-Pull	Low	true

6.6 USART Interfaces

Table 6: USART pin configuration

Pin	USART Signal	Mode	Speed	Used
PA9	USART1_TX	Alternate Function Push-Pull	Low	true
PB7	USART1_RX	Alternate Function Push-Pull	Low	true
PC6	USART6_TX	Alternate Function Push-Pull	Very High	true
PC7	USART6_RX	Alternate Function Push-Pull	Very High	true

7 Methodology

7.1 Project Setup and Working Procedure

In this section, we begin describing our working steps. First, the entire work was divided into three major parts:

1. Recording and playing audio,
2. Implementing the AI model, and
3. MFCC calculation.

To start, install and open **STM32CubeIDE**. Then navigate to:

$$\text{File} \rightarrow \text{New} \rightarrow \text{STM32 Project}$$

The IDE may download and extract required files, so wait until the STM32 project page appears.Once the window opens, look at the upper-left corner where four options are visible. Select the **Board Selector** option and search for your development board. In our case, the board used is the **STM32F746G-DISCO**. After selecting the board, the IDE will display relevant information about it. Read through the details and click **Next** at the bottom-right corner.Enter a project name of your choice, then click **Next**. Select the desired firmware package version; in this work, version **v1.17.4** was used. Click **Finish**. If the IDE prompts whether you want to use the default pin configuration, click **Yes**, unless you prefer configuring the pins manually.One of the most useful features of STM32CubeIDE is that pin functions can be changed easily—for example, switching a pin from GPIO to UART, or vice versa. Many other development tools do not offer this level of flexibility.After the

`.ioc` file opens, adjust the pin configuration according to the requirements of this project. Then navigate to:

File → Save

STM32CubeIDE will automatically generate the necessary initialization code.

On the left side of STM32CubeIDE, you will find a folder named after your project. Click on this folder to expand its contents. Several subdirectories will appear. To access the main application code, navigate to:

Core → Src → `main.c`

Inside `main.c`, STM32CubeIDE automatically generates code for clock configuration, register initialization, and pin setup. Understanding these sections requires considerable knowledge of embedded systems and microcontroller architecture. If you are not familiar with these topics, you may skip these parts and scroll directly to the **main loop**.

Within the main loop, you will find an infinite loop structure:

```
while (1)
{
    /* Write your logic here */
}
```

This is where your application logic should be written.

A very important note is that any code placed *outside* the designated user-code regions will be removed automatically whenever the project is regenerated. Therefore, custom code must be placed between the following markers:

```
/* USER CODE BEGIN */
/* USER CODE END */
```

Alternatively, separate source and header files can be created under the `Core` directory to ensure that custom code is not overwritten.For reference, the project created for this work is named **Audio_ML**. The project directory contains folders such as:

- **Binary**: Contains the compiled output files such as `.bin` and `.elf`, which are used for programming the microcontroller.
- **Include**: Stores the project's header files (`.h`), which declare functions, variables, and configuration parameters used throughout the code.

- **Core**: The main application directory. It contains startup code, system initialization files, and user-defined source files.
- **Drivers**: Includes STM32 HAL and LL driver libraries that provide hardware abstraction and allow interaction with microcontroller peripherals.
- **FatFs**: A lightweight file system library used when SD card or USB mass storage support is required.
- **Middlewares**: Contains third-party and ST middleware components such as USB stacks, FatFs, and other protocol libraries.
- **USB Host**: Provides source files necessary for enabling USB Host functionality, allowing the MCU to communicate with USB devices.
- **X-Cube**: Includes additional STM32Cube software extensions, such as AI packages, audio libraries, and sensor modules.
- **Debug**: Stores debug-related build artifacts used during debugging sessions, such as symbol files and intermediate outputs.
- **Utilities**: Contains helper scripts, board support package (BSP) code, and general utility functions used across the project.
- **.ioc File**: The STM32CubeMX configuration file. It defines pin mapping, clock configuration, and middleware setup. Code regeneration depends on this file.
- **License Files**: Provide licensing information for STM32 HAL, middleware libraries, and any third-party software components included in the project.

I use FreeRTOS (Free Real-Time Operating System) which is an open-source operating system designed specifically for small embedded systems. It allows your microcontroller to run multiple tasks (functions) at the same time in a predictable, real-time manner.

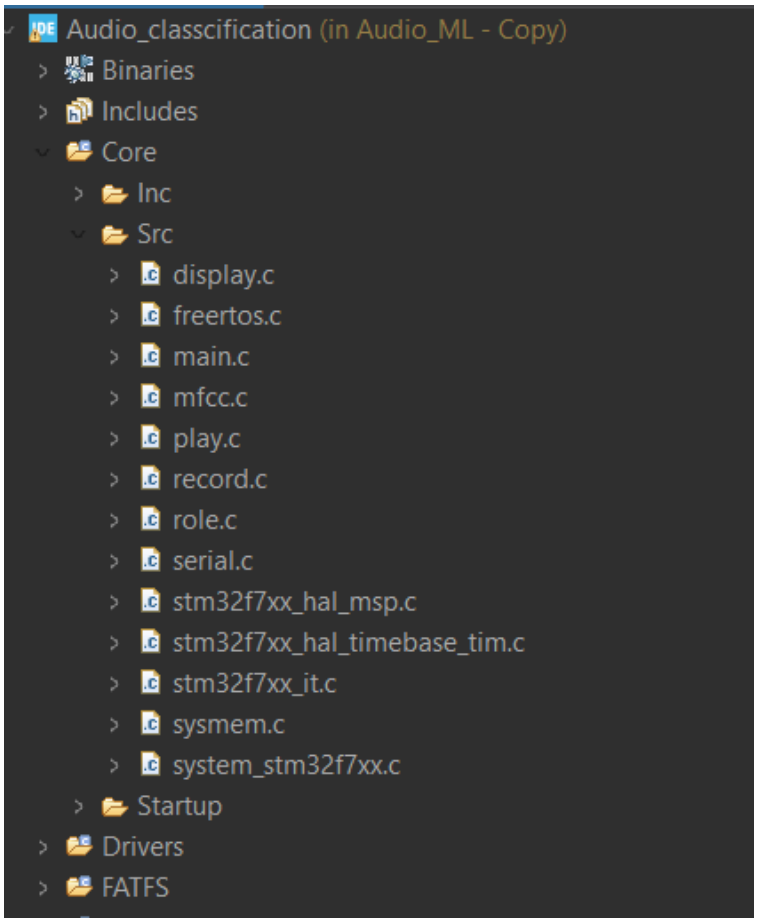

Figure 4: c files of our project

- **display.c**: Responsible for writing single-line and multi-line text to the on-board display.
- **freertos.c**: Manages all FreeRTOS-related functionality, including task creation, scheduling, and task priority configuration.
- **main.c**: Contains essential system initialization code such as clock configuration, register setup, pin mode configuration, and peripheral initialization.
- **mfcc.c**: Implements the MFCC calculation algorithm, converting raw audio waveforms into MFCC features for machine learning.
- **play.c**: Handles audio playback functionality, allowing the system to output wave files.
- **record.c**: Manages audio recording using the MEMS microphone and stores recorded data in the SD card or SDRAM.

- **role.c**: Defines system task behavior and coordinates how different tasks interact with each other.
- **serial.c**: Provides UART transmit and receive functions, mainly used for debugging and printing diagnostic messages.
- **Auto-generated System Files:**
  `stm32f7xx_hal_msp.c`, `stm32f7xx_hal_timebase.c`, `stm32f7xx_it.c`, `sysmem.c`, `system_stm32f7xx.c`
  These files are automatically generated by STM32CubeMX and contain low-level routines for interrupts, timing functions, memory configuration, and system clock setup.

The MEMS microphone is not directly connected to the STM32 microcontroller. Instead, the WM8994 acts as an intermediate device between the microphone and the STM32. The WM8994 is a high-performance audio codec (ADC + DAC) commonly used in embedded systems, especially on STM32F7 Discovery boards. It converts the analog microphone signal into digital PCM (Pulse Code Modulation) audio. The STM32 communicates with the WM8994 using **I^2C** to configure codec registers, and **SAI (Serial Audio Interface)** to send and receive the actual audio samples.

MEMS Pin	WM8994 Pin	Function
MIC_OUT	IN1L / IN1R	Analog microphone input
VDD	MICBIAS	Microphone bias supply
GND	GND	Ground

WM8994 Pin	STM32 Pin	Signal	Purpose
AIF1_SD	SAI1_SD_A	Serial Data	PCM audio data
AIF1_BCLK	SAI1_SCK_A	Bit Clock	Audio bit timing
AIF1_LRCLK	SAI1_FS_A	Frame Sync	Left/Right channel select
MCLK1	SAI1_MCLK_A	Master Clock	Codec master clock

WM8994 Pin	STM32 Pin	Purpose
SCL	I2C1_SCL	Codec configuration clock
SDA	I2C1_SDA	Codec configuration data
RESET	GPIO Pin	Codec reset control

In this system, audio is recorded at a sampling frequency of **16 kHz**, meaning the analog sound signal from the MEMS microphone is converted into 16,000 digital samples per second by the WM8994 audio codec. To perform this conversion accurately, the WM8994 requires a stable master clock (MCLK), typically derived from the STM32's system clock, and the SAI interface must generate the correct bit clock (BCLK) and frame sync (FS) signals. These clock signals determine the timing of data transmission: BCLK controls the rate at which individual PCM bits are transferred, while FS marks the start of each audio frame. If the clock frequencies are incorrect, the codec cannot sample or output audio properly, resulting in distorted or unusable data. Once the clocking is configured, the WM8994 converts the analog microphone input into 16-bit PCM samples at 48 kHz and sends them to the STM32 via SAI using DMA. The STM32 then processes and stores the samples either on an SD card or temporarily in SDRAM, depending on availability.

1. Audio acquisition from the MEMS microphone (PDM format).
2. PDM-to-PCM conversion using filters.
3. Frame segmentation and MFCC extraction.
4. Model inference using TensorFlow Lite Micro.
5. Displaying recognized name on LCD.

7.2 System Block Diagram

```
Microphone --> PDM --> PCM --> MFCC --> ML Model --> LCD Output
```

7.3 Source Code

The full project source code is available at: https://github.com/MehediEEE45/Audio_ML

7.3.1 Audio recording

record.c

```
#include "record.h"
#include <display.h>
#include <stdio.h>

```

```
#define BUF_SIZE AUDIO_IN_PCM_BUFFER_SIZE    //half word = 4*2304 =9216

uint16_t buffer[BUF_SIZE];              //buffer[9216]
uint16_t buffer_size = BUF_SIZE;

uint8_t pHeaderBuff[44];                 //standard PCM WAV header length is 44 bytes

static AUDIO_IN_BufferTypeDef  BufferCtl;
static __IO uint32_t uwVolume = 100;
WAVE_FormatTypeDef WaveFormat;

/* SDRAM scratch area for temporary recording when SD is missing.
  SDRAM on STM32746G-Discovery is mapped at 0xC0000000 by FMC in CubeMX
  We store raw PCM 16-bit samples here until SD is available to flush.
  Configure size for 10 seconds: DEFAULT_AUDIO_IN_FREQ * channels * bytes_per_sample * 10
*/
#define SDRAM_BASE_ADDR     ((uint32_t)0xC0000000)
#define TEMP_REC_SECONDS    10 /* user requirement */
#define TEMP_REC_BYTES      (DEFAULT_AUDIO_IN_FREQ * DEFAULT_AUDIO_IN_CHANNEL_NBR * (AUDIODATA_SIZE) * TEMP_REC_SECONDS)
#define TEMP_REC_HALFWORDS  (TEMP_REC_BYTES/2)

static uint32_t sdram_rec_pos = 0; /* position in half-words (uint16_t) */
static uint8_t recording_to_sdram = 0;

/* Accessor functions for other modules (playback) */
uint16_t* RECORD_GetSDRAMPtr(void)
{
  return (uint16_t*)SDRAM_BASE_ADDR;
}

uint32_t RECORD_GetSDRAMSizeHalfWords(void)
{
  return sdram_rec_pos;
}

void RECORD_ClearSDRAM(void)
{
  sdram_rec_pos = 0;
```

```
43 }
44
45                 /*
46                 Initialize the wave header file
47                 pHeader: Header Buffer to be filled
48                 pWaveFormatStruct: Pointer to the wave structure
   to be filled.
49                 0 if passed, !0 if failed.
50                 */
51
52 static uint32_t WavProcess_HeaderInit(uint8_t* pHeader,
   WAVE_FormatTypeDef* pWaveFormatStruct)
53 {
54   /* Write chunkID, must be 'RIFF'
    ----------------------------------------*/
55   pHeader[0] = 'R';
56   pHeader[1] = 'I';
57   pHeader[2] = 'F';
58   pHeader[3] = 'F';
59
60   /* Write the file length
    -----------------------------------------------------*/
61   /* The sampling time: this value will be written back at the
    end of the
62     recording operation.  Example: 661500 Btyes = 0x000A17FC,
    byte[7]=0x00, byte[4]=0xFC */
63   pHeader[4] = 0x00;
64   pHeader[5] = 0x4C;
65   pHeader[6] = 0x1D;
66   pHeader[7] = 0x00;
67   /* Write the file format, must be 'WAVE'
    -----------------------------------*/
68   pHeader[8]  = 'W';
69   pHeader[9]  = 'A';
70   pHeader[10] = 'V';
71   pHeader[11] = 'E';
72
73   /* Write the format chunk, must be'fmt '
    -----------------------------------*/
74   pHeader[12]  = 'f';
75   pHeader[13]  = 'm';
76   pHeader[14]  = 't';
77   pHeader[15]  = ' ';
78
79   /* Write the length of the 'fmt' data, must be 0x10
    ----------------------*/
```

```
80  pHeader[16]  = 0x10;
81  pHeader[17]  = 0x00;
82  pHeader[18]  = 0x00;
83  pHeader[19]  = 0x00;
84
85  /* Write the audio format, must be 0x01 (PCM)
     -----------------------------*/
86  pHeader[20]  = 0x01;
87  pHeader[21]  = 0x00;
88
89  /* Write the number of channels, ie. 0x01 (Mono)
     ---------------------------*/
90  pHeader[22]  = pWaveFormatStruct->NbrChannels;
91  pHeader[23]  = 0x00;
92
93  /* Write the Sample Rate in Hz
     -------------------------------------------*/
94  /* Write Little Endian ie. 8000 = 0x00001F40 => byte[24]=0x40,
     byte[27]=0x00*/
95  pHeader[24]  = (uint8_t)((pWaveFormatStruct->SampleRate &
     0xFF));
96  pHeader[25]  = (uint8_t)((pWaveFormatStruct->SampleRate >> 8)
     & 0xFF);
97  pHeader[26]  = (uint8_t)((pWaveFormatStruct->SampleRate >> 16)
     & 0xFF);
98  pHeader[27]  = (uint8_t)((pWaveFormatStruct->SampleRate >> 24)
     & 0xFF);
99
100 /* Write the Byte Rate
     -------------------------------------------------*/
101 pHeader[28]  = (uint8_t)((pWaveFormatStruct->ByteRate & 0xFF));
102 pHeader[29]  = (uint8_t)((pWaveFormatStruct->ByteRate >> 8) &
     0xFF);
103 pHeader[30]  = (uint8_t)((pWaveFormatStruct->ByteRate >> 16) &
     0xFF);
104 pHeader[31]  = (uint8_t)((pWaveFormatStruct->ByteRate >> 24) &
     0xFF);
105
106 /* Write the block alignment
     -------------------------------------------*/
107 pHeader[32]  = pWaveFormatStruct->BlockAlign;
108 pHeader[33]  = 0x00;
109
110 /* Write the number of bits per sample
     -----------------------------------*/
111 pHeader[34]  = pWaveFormatStruct->BitPerSample;
```

```
112   pHeader[35]  = 0x00;
113
114   /* Write the Data chunk, must be 'data'
       ------------------------------------*/
115   pHeader[36]  = 'd';
116   pHeader[37]  = 'a';
117   pHeader[38]  = 't';
118   pHeader[39]  = 'a';
119
120   /* Write the number of sample data
       ----------------------------------------*/
121   /* This variable will be written back at the end of the
       recording operation */
122   pHeader[40]  = 0x00;
123   pHeader[41]  = 0x4C;
124   pHeader[42]  = 0x1D;
125   pHeader[43]  = 0x00;
126
127   /* Return 0 if all operations are OK */
128   return 0;
129 }
130
131 /*
132               Encoder initialization.
133               Freq: Sampling frequency.
134               pHeader: Pointer to the WAV file header to be
       written.
135               0 if success, !0 else.
136   */
137
138 static uint32_t WavProcess_EncInit(uint32_t Freq, uint8_t
       *pHeader)
139 {
140   /* Initialize the encoder structure */
141   WaveFormat.SampleRate = Freq;        /* Audio sampling
       frequency */
142   WaveFormat.NbrChannels = 1;          /* Number of channels:
       1:Mono or 2:Stereo */
143   WaveFormat.BitPerSample = 16;        /* Number of bits per
       sample (16, 24 or 32) */
144   WaveFormat.FileSize = 0x001D4C00;    /* Total length of useful
       audio data (payload)  0x001D4C00= 1920000 bytes */
145   WaveFormat.SubChunk1Size = 44;       /* The file header chunk
       size */
146   WaveFormat.ByteRate = (WaveFormat.SampleRate * \
147                          (WaveFormat.BitPerSample/8) * \
```

```
148                             WaveFormat.NbrChannels);     /* Number
    of bytes per second  (sample rate * block align)  */
149   WaveFormat.BlockAlign = WaveFormat.NbrChannels * \
150                             (WaveFormat.BitPerSample/8); /*
    channels * bits/sample / 8 */
151
152   /* Parse the wav file header and extract required information
     */
153   if(WavProcess_HeaderInit(pHeader, &WaveFormat))
154   {
155     return 1;
156   }
157   return 0;
158 }
159
160
161
162 /*
163             Initialize the wave header file
164             pHeader: Header Buffer to be filled
165             pWaveFormatStruct: Pointer to the wave structure to
    be filled.
166             0 if passed, !0 if failed.
167 */
168
169 static uint32_t WavProcess_HeaderUpdate(uint8_t* pHeader,
    WAVE_FormatTypeDef* pWaveFormatStruct)
170 {
171   /* Write the file length
     ----------------------------------------------------*/
172   /* The sampling time: this value will be written back at the
    end of the
173     recording operation.  Example: 661500 Btyes = 0x000A17FC,
    byte[7]=0x00, byte[4]=0xFC */
174   pHeader[4] = (uint8_t)(BufferCtl.fptr);
175   pHeader[5] = (uint8_t)(BufferCtl.fptr >> 8);
176   pHeader[6] = (uint8_t)(BufferCtl.fptr >> 16);
177   pHeader[7] = (uint8_t)(BufferCtl.fptr >> 24);
178   /* Write the number of sample data
     -----------------------------------------*/
179   /* This variable will be written back at the end of the
    recording operation */
180   BufferCtl.fptr -=44;
181   pHeader[40] = (uint8_t)(BufferCtl.fptr);
182   pHeader[41] = (uint8_t)(BufferCtl.fptr >> 8);
183   pHeader[42] = (uint8_t)(BufferCtl.fptr >> 16);
```

```
184   pHeader[43] = (uint8_t)(BufferCtl.fptr >> 24);
185
186   /* Return 0 if all operations are OK */
187   return 0;
188 }
189
190
191
192
193
194 AUDIO_ErrorTypeDef recordStart()
195 {
196     FRESULT res; /* FatFs function common result code */
197     uint32_t byteswritten = 0;
198     uwVolume = 100;
199   /* If SD card is present try to open file and record directly
      to SD
200       otherwise record to SDRAM temporary buffer and flush later
     */
201   if (BSP_SD_IsDetected())
202   {
203     res = f_open(&SDFile, REC_WAVE_NAME, FA_CREATE_ALWAYS |
     FA_WRITE);
204     if( res != FR_OK)
205     {
206       serialPrintln(&vcp, "cannot open file, code error : %d",
     res);
207       text(0, 120, 'c', "SD open ERR", 'r', 'k');
208       /* fallback to SDRAM */
209       recording_to_sdram = 1;
210       sdram_rec_pos = 0;
211     }
212     else
213     {
214       serialPrintln(&vcp,"open file");
215       WavProcess_EncInit(DEFAULT_AUDIO_IN_FREQ, pHeaderBuff);
                  // Initialize header file
216
217       if(f_write(&SDFile, pHeaderBuff, 44, (void*)&byteswritten)
     == FR_OK)            //Write header file
218       {
219         if(byteswritten != 0)
220         {
221           serialPrintln(&vcp,"start record to SD");
222           recording_to_sdram = 0;
223           BSP_AUDIO_IN_Init(DEFAULT_AUDIO_IN_FREQ,
```

```
   DEFAULT_AUDIO_IN_BIT_RESOLUTION, 1);
224            BSP_AUDIO_IN_Record((uint16_t*)&BufferCtl.pcm_buff[0],
   AUDIO_IN_PCM_BUFFER_SIZE);
225            BufferCtl.fptr = byteswritten;
226            BufferCtl.pcm_ptr = 0;
227            BufferCtl.offset = 0;
228            BufferCtl.wr_state = BUFFER_EMPTY;
229            HAL_GPIO_WritePin(GPIOI, GPIO_PIN_1, SET);
230          }
231        }
232      }
233    }
234    else
235    {
236      /* SD not present: record into SDRAM temporary area */
237      serialPrintln(&vcp, "SD not present: recording to SDRAM (%d
   s)", TEMP_REC_SECONDS);
238      recording_to_sdram = 1;
239      sdram_rec_pos = 0;
240      WavProcess_EncInit(DEFAULT_AUDIO_IN_FREQ, pHeaderBuff); /*
   prepare header metadata */
241      BSP_AUDIO_IN_Init(DEFAULT_AUDIO_IN_FREQ,
   DEFAULT_AUDIO_IN_BIT_RESOLUTION, 1);
242      BSP_AUDIO_IN_Record((uint16_t*)&BufferCtl.pcm_buff[0],
   AUDIO_IN_PCM_BUFFER_SIZE);
243      BufferCtl.fptr = 44; /* we'll simulate header size already
   written */
244      BufferCtl.pcm_ptr = 0;
245      BufferCtl.offset = 0;
246      BufferCtl.wr_state = BUFFER_EMPTY;
247      HAL_GPIO_WritePin(GPIOI, GPIO_PIN_1, SET);
248
249      /* show LCD indicator and initial remaining seconds */
250      char info[32];
251      snprintf(info, sizeof(info), "Recording (SDRAM) %ds",
   TEMP_REC_SECONDS);
252      text(0, 120, 'c', info, 'g', 'k');
253    }
254
255      return (AUDIO_ERROR_NONE);
256  }
257
258
259
260
261  AUDIO_ErrorTypeDef recordProcess()
```

```
{
    uint32_t elapsed_time;
    static uint32_t prev_elapsed_time = 0xFFFFFFFF;

    FRESULT res;                                          // FatFs
   function common result code
    uint32_t byteswritten = 0;

    if(BufferCtl.fptr >= REC_SAMPLE_LENGTH)               // MAX
   Recording time reached, so stop audio interface and close file
    {
      return (AUDIO_ERROR_EOF);
   //signaling end of recording
    }

    if (BufferCtl.wr_state == BUFFER_FULL)
    {
        for(int i = 0; i < AUDIO_IN_PCM_BUFFER_SIZE/2; i ++)
        {
            buffer[i] =  BufferCtl.pcm_buff[BufferCtl.offset +
   i*4];
        }

    if (recording_to_sdram)
    {
      /* copy converted samples into SDRAM temporary buffer */
      uint32_t copy_count = AUDIO_IN_PCM_BUFFER_SIZE/2; /*
   number of half-words */
      /* check overflow */
      if ((sdram_rec_pos + copy_count) > TEMP_REC_HALFWORDS)
      {
        /* cannot store all; clamp and signal EOF after writing
   what fits */
        copy_count = (TEMP_REC_HALFWORDS > sdram_rec_pos) ?
   (TEMP_REC_HALFWORDS - sdram_rec_pos) : 0;
      }
      if (copy_count > 0)
      {
        uint16_t *dst = (uint16_t*)(SDRAM_BASE_ADDR +
   sdram_rec_pos * 2);
        for (uint32_t k = 0; k < copy_count; k++) dst[k] =
   buffer[k];
        sdram_rec_pos += copy_count;
        byteswritten = copy_count; /* keep same unit as original
   (half-words) */
        BufferCtl.fptr += byteswritten * 2; /* bytes */
```

```
298       }
299       if (sdram_rec_pos >= TEMP_REC_HALFWORDS)
300       {
301         /* reached SDRAM capacity */
302         BufferCtl.wr_state = BUFFER_EMPTY;
303         return AUDIO_ERROR_EOF;
304       }
305       BufferCtl.wr_state = BUFFER_EMPTY;
306     }
307     else
308     {
309       res = f_write(&SDFile,
(uint16_t*)(buffer),AUDIO_IN_PCM_BUFFER_SIZE/2,(void*)&byteswritten);
          // write buffer in file
310       if(res != FR_OK)
311       {
312         serialPrintln(&vcp, "cannot store data, code error :
%d",res);
313       }
314       BufferCtl.fptr += byteswritten;
315       BufferCtl.wr_state = BUFFER_EMPTY;
316     }
317     }
318
319     elapsed_time = BufferCtl.fptr / (DEFAULT_AUDIO_IN_FREQ *
DEFAULT_AUDIO_IN_CHANNEL_NBR * 2);                // Display
elapsed time
320     if(prev_elapsed_time != elapsed_time)
321     {
322       /* update elapsed_time and optionally LCD when recording
to SDRAM */
323       prev_elapsed_time = elapsed_time;
324       if (recording_to_sdram)
325       {
326         /* compute remaining seconds */
327         uint32_t bytes_written = BufferCtl.fptr - 44; /*
subtract fake header */
328         uint32_t bytes_per_sec = DEFAULT_AUDIO_IN_FREQ *
DEFAULT_AUDIO_IN_CHANNEL_NBR * (AUDIODATA_SIZE);
329         uint32_t elapsed_sec = (bytes_per_sec > 0) ?
(bytes_written / bytes_per_sec) : 0;
330         int32_t sec_left = (int32_t)TEMP_REC_SECONDS -
(int32_t)elapsed_sec;
331         if (sec_left < 0) sec_left = 0;
332         char info[32];
333         snprintf(info, sizeof(info), "Recording (SDRAM) %lds",
```

```
    (long)sec_left);
334         text(0, 120, 'c', info, 'g', 'k');
335       }
336     }
337
338     return (AUDIO_ERROR_NONE);
339 }
340
341
342
343 AUDIO_ErrorTypeDef recordStop()
344 {
345     FRESULT res;                                                    //
    FatFs function common result code
346     uint32_t byteswritten = 0;
347
348
349     BSP_AUDIO_IN_Stop(CODEC_PDWN_SW);                               //
    Stop recorder
350     HAL_Delay(150);
351
352   if (recording_to_sdram)
353   {
354     /* Try to flush SDRAM contents to SD if available */
355     uint8_t flushed_ok = 0;
356     if (BSP_SD_IsDetected())
357     {
358       /* create file and write header placeholder */
359       res = f_open(&SDFile, REC_WAVE_NAME, FA_CREATE_ALWAYS |
    FA_WRITE);
360       if (res != FR_OK)
361       {
362         serialPrintln(&vcp, "cannot create file to flush SDRAM:
    %d", res);
363       }
364       else
365       {
366         /* write placeholder header */
367         WavProcess_EncInit(DEFAULT_AUDIO_IN_FREQ, pHeaderBuff);
368         if (f_write(&SDFile, pHeaderBuff, 44,
    (void*)&byteswritten) != FR_OK)
369         {
370           serialPrintln(&vcp, "cannot write header when
    flushing: %d", res);
371         }
372         else
```

```
373         {
374           /* write SDRAM buffer in chunks */
375           uint16_t *src = (uint16_t*)SDRAM_BASE_ADDR;
376           uint32_t remaining = sdram_rec_pos; /* half-words */
377           uint32_t pos = 0;
378           while (remaining > 0)
379           {
380             uint32_t write_count = (remaining >
    (AUDIO_IN_PCM_BUFFER_SIZE/2)) ? (AUDIO_IN_PCM_BUFFER_SIZE/2)
    : remaining;
381             UINT bw = 0;
382             FRESULT r = f_write(&SDFile, &src[pos], write_count,
    &bw);
383             if (r != FR_OK)
384             {
385               serialPrintln(&vcp, "SD write error during flush:
    %d", r);
386               break;
387             }
388             pos += write_count;
389             remaining -= write_count;
390           }
391           /* set fptr to header + data bytes for header update */
392           BufferCtl.fptr = 44 + (sdram_rec_pos * 2);
393           /* update header */
394           res = f_lseek(&SDFile, 0);
395           if (res == FR_OK)
396           {
397             WavProcess_HeaderUpdate(pHeaderBuff, &WaveFormat);
398             res = f_write(&SDFile, pHeaderBuff,
    sizeof(WAVE_FormatTypeDef), (void*)&byteswritten);
399             if (res != FR_OK)
400             {
401               serialPrintln(&vcp, "cannot finalize header after
    flush: %d", res);
402             }
403           }
404           f_close(&SDFile);
405           flushed_ok = 1;
406         }
407       }
408     }
409     else
410     {
411       /* SD still not present: keep data in SDRAM */
412       serialPrintln(&vcp, "SD not present: recording saved in
```

```
   SDRAM (%lu half-words)", sdram_rec_pos);
413      }
414      /* show LCD message about where data was saved */
415      if (flushed_ok)
416      {
417        text(0, 120, 'c', "Recording saved to SD", 'g', 'k');
418      }
419      else
420      {
421        char info2[40];
422        snprintf(info2, sizeof(info2), "Saved in SDRAM (%lus)",
   (unsigned long)( (sdram_rec_pos*2) / (DEFAULT_AUDIO_IN_FREQ *
   DEFAULT_AUDIO_IN_CHANNEL_NBR * (AUDIODATA_SIZE)) ));
423        text(0, 120, 'c', info2, 'r', 'k');
424      }
425      recording_to_sdram = 0;
426    }
427    else
428    {
429      /* original SD-based flow: finalize header and close file */
430      res = f_lseek(&SDFile, 0);                                    //
   Move file pointer of the file object
431      if(res != FR_OK)
432      {
433        serialPrintln(&vcp, "f_lseek error, code error : %d",res);
434      }
435      else
436      {
437        WavProcess_HeaderUpdate(pHeaderBuff, &WaveFormat);
                  //Update the wav file header save it into wav
   file
438
439        res = f_write(&SDFile, pHeaderBuff,
   sizeof(WAVE_FormatTypeDef), (void*)&byteswritten);
440        if(res != FR_OK)
441        {
442          serialPrintln(&vcp, "cannot end file, code error :
   %d",res);
443        }
444        else
445        {
446          serialPrintln(&vcp, "end of file",res);
447        }
448      }
449      f_close(&SDFile);
               // Close file
```

```
  }

  HAL_GPIO_WritePin(GPIOI, GPIO_PIN_1, RESET);

  serialPrintln(&vcp, "recording success");

  return (AUDIO_ERROR_NONE);
}


void BSP_AUDIO_IN_HalfTransfer_CallBack(void)
{
    BufferCtl.wr_state = BUFFER_FULL;
    BufferCtl.offset = 0;

}


void BSP_AUDIO_IN_TransferComplete_CallBack(void)
{
    BufferCtl.wr_state = BUFFER_FULL;
    BufferCtl.offset = AUDIO_IN_PCM_BUFFER_SIZE;
}
```

Code Listing 1: Audio recording code (record.c)

7.3.2 Audio Playing

The audio playback module (`play.c`) is responsible for playing recorded WAV audio files either directly from the SD card or from a temporary SDRAM buffer when the SD card is unavailable. It supports double-buffered audio streaming, automatic detection of storage source, and real-time playback using the STM32 audio output peripheral. **play.c**

```

#include "play.h"
#include "record.h"

static AUDIO_OUT_BufferTypeDef  BufferCtl;
static int16_t FilePos = 0;
static __IO uint32_t uwVolume = 100;

int tes1,tes2;

AUDIO_PLAYBACK_StateTypeDef AudioState;
```

```
12
13  WAVE_FormatTypeDef WaveReadFormat;
14  FILELIST_FileTypeDef FileList;
15
16  /* playback-from-SDRAM state */
17  static uint8_t playback_from_sdram = 0;
18  static uint32_t sdram_play_pos = 0; /* half-words already
       consumed */
19  static uint32_t sdram_play_total = 0; /* total half-words
       available */
20
21  /* Private function prototypes
       -----------------------------------------------*/
22  static AUDIO_ErrorTypeDef GetFileInfo(uint16_t file_idx,
       WAVE_FormatTypeDef *info);
23  static uint8_t PlayerInit(uint32_t AudioFreq);
24
25  FILELIST_FileTypeDef FileList;
26  uint16_t NumObs = 0;
27
28  /*
29                Copies disk content in the explorer list.
30                None
31                Operation result
32    */
33  FRESULT AUDIO_StorageParse(void)
34  {
35    FRESULT res = FR_OK;
36    FILINFO fno;
37    DIR dir;
38    char *fn;
39
40    res = f_opendir(&dir, SDPath);
41    FileList.ptr = 0;
42
43    if(res == FR_OK)
44    {
45      while(BSP_SD_IsDetected())
46      {
47        res = f_readdir(&dir, &fno);
48        if(res != FR_OK || fno.fname[0] == 0)
49        {
50          break;
51        }
52        if(fno.fname[0] == '.')
53        {
```

```
54          continue;
55        }
56        fn = fno.fname;
57
58        if(FileList.ptr < FILEMGR_LIST_DEPDTH)
59        {
60          if((fno.fattrib & AM_DIR) == 0)
61          {
62            if((strstr(fn, "wav")) || (strstr(fn, "WAV")))
63            {
64              strncpy((char *)FileList.file[FileList.ptr].name,
    (char *)fn, FILEMGR_FILE_NAME_SIZE);
65              FileList.file[FileList.ptr].type = FILETYPE_FILE;
66              FileList.ptr++;
67            }
68          }
69        }
70      }
71    }
72    else
73    {
74        serialPrintln(&vcp,  "cannot open dir : %d",res);
75
76    }
77    NumObs = FileList.ptr;
78    serialPrintln(&vcp,"NumbObs : %d",NumObs);
79    f_closedir(&dir);
80    return res;
81  }
82
83
84  /*
85                Shows audio file (*.wav) on the root
86    */
87
88
89  uint8_t AUDIO_ShowWavFiles(void)
90  {
91    if(AUDIO_StorageParse() ==  FR_OK)
92    {
93      if(FileList.ptr > 0)
94      {
95        return 0;
96      }
97      return 1;
98    }
```

```
99    return 2;
100 }
101
102 /*
103                         Gets Wav Object Number.
104   */
105 uint16_t AUDIO_GetWavObjectNumber(void)
106 {
107   return NumObs;
108 }
109
110 AUDIO_ErrorTypeDef AUDIO_PLAYER_Init(void)
111 {
112   if(BSP_AUDIO_OUT_Init(OUTPUT_DEVICE_AUTO, uwVolume,
     AUDIO_FREQUENCY_48K) == 0)
113   {
114     return AUDIO_ERROR_NONE;
115   }
116   else
117   {
118     return AUDIO_ERROR_IO;
119   }
120 }
121
122 /*
123                       Initializes the Wave player.
124                       AudioFreq: Audio sampling frequency
125   */
126 static uint8_t PlayerInit(uint32_t AudioFreq)
127 {
128   /* Initialize the Audio codec and all related peripherals
      (I2S, I2C, IOExpander, IOs...) */
129   if(BSP_AUDIO_OUT_Init(OUTPUT_DEVICE_BOTH, uwVolume, AudioFreq)
      != 0)
130   {
131     return 1;
132   }
133   else
134   {
135     BSP_AUDIO_OUT_SetAudioFrameSlot(CODEC_AUDIOFRAME_SLOT_02);
136     return 0;
137   }
138 }
139
140 /*
141                   Gets the file info.
```

```
142                 file_idx: File index
143                 info: Pointer to WAV file info
144                 Audio error
145   */
146 static AUDIO_ErrorTypeDef GetFileInfo(uint16_t file_idx,
      WAVE_FormatTypeDef *info)
147 {
148   uint32_t bytesread;
149   uint32_t duration;
150   uint8_t str[FILEMGR_FILE_NAME_SIZE + 20];
151
152   if(f_open(&SDFile, (char *)FileList.file[file_idx].name,
      FA_OPEN_EXISTING | FA_READ) == FR_OK)
153   {
154     /* Fill the buffer to Send */
155     if(f_read(&SDFile, info, sizeof(WaveReadFormat), (void
      *)&bytesread) == FR_OK)
156     {
157
158       return AUDIO_ERROR_NONE;
159     }
160     f_close(&SDFile);
161   }
162   return AUDIO_ERROR_IO;
163 }
164
165
166 AUDIO_ErrorTypeDef AUDIO_PLAYER_Start(uint8_t idx)
167 {
168   uint32_t bytesread;
169
170   f_close(&SDFile);
171
172   /* Prefer SD playback when SD is present */
173   if (BSP_SD_IsDetected())
174   {
175     if(AUDIO_GetWavObjectNumber() > idx)
176     {
177       GetFileInfo(idx, &WaveReadFormat);
178       /* Adjust the Audio frequency */
179       PlayerInit(WaveReadFormat.SampleRate);
180       BufferCtl.state = BUFFER_OFFSET_NONE;
181       /* Read from SD */
182       f_lseek(&SDFile, 0);
183       if(f_read(&SDFile, &BufferCtl.buff[0],
      AUDIO_OUT_BUFFER_SIZE, (void *)&bytesread) == FR_OK)
```

```
184         {
185           if(bytesread != 0)
186           {
187             BSP_AUDIO_OUT_Play((uint16_t*)&BufferCtl.buff[0],
    AUDIO_OUT_BUFFER_SIZE/2);
188             BufferCtl.fptr = bytesread;
189             playback_from_sdram = 0;
190             return AUDIO_ERROR_NONE;
191           }
192         }
193       }
194     }
195
196     /* If SD unavailable or read failed, try playback from SDRAM
     temporary buffer */
197     sdram_play_total = RECORD_GetSDRAMSizeHalfWords();
198     if ((sdram_play_total > 0) && (RECORD_GetSDRAMPtr() != NULL))
199     {
200       /* Use the same sample rate used during recording */
201       PlayerInit(DEFAULT_AUDIO_IN_FREQ);
202       BufferCtl.state = BUFFER_OFFSET_NONE;
203       /* copy initial buffer from SDRAM */
204       uint16_t *src = RECORD_GetSDRAMPtr();
205       uint32_t tocopy = (sdram_play_total >=
    AUDIO_OUT_BUFFER_SIZE) ? AUDIO_OUT_BUFFER_SIZE :
    sdram_play_total;
206       for (uint32_t i = 0; i < tocopy; i++) BufferCtl.buff[i] =
    src[i];
207       if (tocopy < AUDIO_OUT_BUFFER_SIZE)
208       {
209         for (uint32_t i = tocopy; i < AUDIO_OUT_BUFFER_SIZE; i++)
    BufferCtl.buff[i] = 0;
210       }
211       BSP_AUDIO_OUT_Play((uint16_t*)&BufferCtl.buff[0],
    AUDIO_OUT_BUFFER_SIZE/2);
212       BufferCtl.fptr = tocopy * 2; /* bytes */
213       sdram_play_pos = tocopy; /* half-words consumed into buffer
    */
214       playback_from_sdram = 1;
215       return AUDIO_ERROR_NONE;
216     }
217
218     return AUDIO_ERROR_IO;
219   }
220
221
```

```
222
223 AUDIO_ErrorTypeDef playStart(int index)
224 {
225     int res;
226     serialPrintln(&vcp,"init play");
227     res = f_mount(&SDFatFS, (TCHAR const*)SDPath, 0);
228     if(res != FR_OK)
229     {
230       serialPrintln(&vcp,"error mount, code error : %d",res);
231     }
232
233     AUDIO_ShowWavFiles();
234
235     res = AUDIO_PLAYER_Start(index);
236     if(res != AUDIO_ERROR_NONE)
237     {
238       serialPrintln(&vcp,"error start audio : %d",res);
239     }
240     else
241     {
242         serialPrintln(&vcp,"audio start gaes");
243     }
244
245     return (AUDIO_ERROR_NONE);
246 }
247
248 AUDIO_ErrorTypeDef playProcess(void)
249 {
250   uint32_t bytesread, elapsed_time;
251   AUDIO_ErrorTypeDef audio_error = AUDIO_ERROR_NONE;
252   static uint32_t prev_elapsed_time = 0xFFFFFFFF;
253
254
255     if(BufferCtl.fptr >= WaveReadFormat.FileSize)
256     {
257       BSP_AUDIO_OUT_Stop(CODEC_PDWN_SW);
258       audio_error = AUDIO_ERROR_EOF;
259     }
260
261     if(BufferCtl.state == BUFFER_OFFSET_HALF)
262     {
263       if (playback_from_sdram)
264       {
265         uint16_t *src = RECORD_GetSDRAMPtr();
266         uint32_t remaining = (sdram_play_total > sdram_play_pos)
    ? (sdram_play_total - sdram_play_pos) : 0;
```

```
267         uint32_t copy = (remaining >= (AUDIO_OUT_BUFFER_SIZE/2))
    ? (AUDIO_OUT_BUFFER_SIZE/2) : remaining;
268         for (uint32_t i = 0; i < copy; i++) BufferCtl.buff[i] =
    src[sdram_play_pos + i];
269         if (copy < (AUDIO_OUT_BUFFER_SIZE/2))
270         {
271           for (uint32_t i = copy; i < (AUDIO_OUT_BUFFER_SIZE/2);
    i++) BufferCtl.buff[i] = 0;
272         }
273         sdram_play_pos += copy;
274         BufferCtl.state = BUFFER_OFFSET_NONE;
275         BufferCtl.fptr += copy * 2;
276       }
277       else
278       {
279         if(f_read(&SDFile,
280                   &BufferCtl.buff[0],
281                   AUDIO_OUT_BUFFER_SIZE/2,
282                   (void *)&bytesread) != FR_OK)
283         {
284           BSP_AUDIO_OUT_Stop(CODEC_PDWN_SW);
285           return AUDIO_ERROR_IO;
286         }
287         BufferCtl.state = BUFFER_OFFSET_NONE;
288         BufferCtl.fptr += bytesread;
289       }
290     }
291
292     if(BufferCtl.state == BUFFER_OFFSET_FULL)
293     {
294       if (playback_from_sdram)
295       {
296         uint16_t *src = RECORD_GetSDRAMPtr();
297         uint32_t remaining = (sdram_play_total > sdram_play_pos)
    ? (sdram_play_total - sdram_play_pos) : 0;
298         uint32_t copy = (remaining >= (AUDIO_OUT_BUFFER_SIZE/2))
    ? (AUDIO_OUT_BUFFER_SIZE/2) : remaining;
299         for (uint32_t i = 0; i < copy; i++)
    BufferCtl.buff[(AUDIO_OUT_BUFFER_SIZE/2) + i] =
    src[sdram_play_pos + i];
300         if (copy < (AUDIO_OUT_BUFFER_SIZE/2))
301         {
302           for (uint32_t i = copy; i < (AUDIO_OUT_BUFFER_SIZE/2);
    i++) BufferCtl.buff[(AUDIO_OUT_BUFFER_SIZE/2) + i] = 0;
303         }
304         sdram_play_pos += copy;
```

```
305           BufferCtl.state = BUFFER_OFFSET_NONE;
306           BufferCtl.fptr += copy * 2;
307         }
308         else
309         {
310           if(f_read(&SDFile,
311                     &BufferCtl.buff[AUDIO_OUT_BUFFER_SIZE/2],
312                     AUDIO_OUT_BUFFER_SIZE/2,
313                     (void *)&bytesread) != FR_OK)
314           {
315             BSP_AUDIO_OUT_Stop(CODEC_PDWN_SW);
316             return AUDIO_ERROR_IO;
317           }
318
319           BufferCtl.state = BUFFER_OFFSET_NONE;
320           BufferCtl.fptr += bytesread;
321         }
322       }
323
324       /* Display elapsed time */
325       elapsed_time = BufferCtl.fptr / WaveReadFormat.ByteRate;
326       if(prev_elapsed_time != elapsed_time)
327       {
328         prev_elapsed_time = elapsed_time;
329       }
330     return audio_error;
331   }
332
333
334   void BSP_AUDIO_OUT_TransferComplete_CallBack(void)
335   {
336       BufferCtl.state = BUFFER_OFFSET_FULL;
337   }
338
339   void BSP_AUDIO_OUT_HalfTransfer_CallBack(void)
340   {
341       BufferCtl.state = BUFFER_OFFSET_HALF;
342   }
343
344   AUDIO_ErrorTypeDef playStop(void)
345   {
346     AudioState = AUDIO_STATE_STOP;
347     FilePos = 0;
348
349     BSP_AUDIO_OUT_Stop(CODEC_PDWN_SW);
350     if (!playback_from_sdram) f_close(&SDFile);
```

```
  playback_from_sdram = 0;
  sdram_play_pos = 0;
  sdram_play_total = 0;

  serialPrintln(&vcp,"selesai play");
  return AUDIO_ERROR_NONE;
}
```

Code Listing 2: Audio playing source code (play.c)

7.3.3 Display Print

display.c

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>   // for tolower()
#include "stm32746g_discovery_lcd.h"
#include "stm32746g_discovery_ts.h"

void LCD_Config(void)
{
    /* LCD Initialization */
    BSP_LCD_Init();

    /* Layer Initialization */
    BSP_LCD_LayerDefaultInit(0, LCD_FB_START_ADDRESS);
    BSP_LCD_LayerDefaultInit(1,
        LCD_FB_START_ADDRESS +
        (BSP_LCD_GetXSize() * BSP_LCD_GetYSize() * 4));

    /* Enable LCD */
    BSP_LCD_DisplayOn();

    /* Background Layer */
    BSP_LCD_SelectLayer(0);
    BSP_LCD_Clear(LCD_COLOR_BLACK);

    /* Foreground Layer */
    BSP_LCD_SelectLayer(1);
    BSP_LCD_Clear(LCD_COLOR_BLACK);
    BSP_LCD_SetBackColor(LCD_COLOR_BLACK);
}

void setFont(char size)
{
```

```
33      size = tolower(size);
34
35      switch (size)
36      {
37          case 's': BSP_LCD_SetFont(&Font12); break; // Small
38          case 'm': BSP_LCD_SetFont(&Font16); break; // Medium
39          case 'l': BSP_LCD_SetFont(&Font24); break; // Large
40          default:  BSP_LCD_SetFont(&Font20); break;
41      }
42  }
43
44  void text(int x, int y, char ps, char text_str[],
45            char text_color, char scr_color)
46  {
47      char buf[64];
48
49      ps = tolower(ps);
50      text_color = tolower(text_color);
51      scr_color = tolower(scr_color);
52
53      /* Screen color */
54      if (scr_color == 'b') BSP_LCD_Clear(LCD_COLOR_BLUE);
55      else if (scr_color == 'r') BSP_LCD_Clear(LCD_COLOR_RED);
56      else if (scr_color == 'g') BSP_LCD_Clear(LCD_COLOR_GREEN);
57      else BSP_LCD_Clear(LCD_COLOR_BLACK);
58
59      /* Text color */
60      if (text_color == 'b') BSP_LCD_SetTextColor(LCD_COLOR_BLUE);
61      else if (text_color == 'r')
    BSP_LCD_SetTextColor(LCD_COLOR_RED);
62      else if (text_color == 'g')
    BSP_LCD_SetTextColor(LCD_COLOR_GREEN);
63      else BSP_LCD_SetTextColor(LCD_COLOR_WHITE);
64
65      snprintf(buf, sizeof(buf), "%s", text_str);
66
67      if (ps == 'l')
68          BSP_LCD_DisplayStringAt(x, y, (uint8_t*)buf, LEFT_MODE);
69      else if (ps == 'r')
70          BSP_LCD_DisplayStringAt(x, y, (uint8_t*)buf, RIGHT_MODE);
71      else
72          BSP_LCD_DisplayStringAt(x, y, (uint8_t*)buf,
    CENTER_MODE);
73  }
74
75  void multitext(int x, int y, char ps,
```

```
                char text_str[], char text_color)
{
    char buf[64];
    ps = tolower(ps);
    text_color = tolower(text_color);

    if (text_color == 'b') BSP_LCD_SetTextColor(LCD_COLOR_BLUE);
    else if (text_color == 'r')
  BSP_LCD_SetTextColor(LCD_COLOR_RED);
    else if (text_color == 'g')
  BSP_LCD_SetTextColor(LCD_COLOR_GREEN);
    else BSP_LCD_SetTextColor(LCD_COLOR_WHITE);

    int line_y = y;
    char *line = strtok(text_str, "\n");

    while (line != NULL)
    {
        snprintf(buf, sizeof(buf), "%s", line);

        if (ps == 'l')
            BSP_LCD_DisplayStringAt(x, line_y,
                                    (uint8_t*)buf, LEFT_MODE);
        else if (ps == 'r')
            BSP_LCD_DisplayStringAt(x, line_y,
                                    (uint8_t*)buf, RIGHT_MODE);
        else
            BSP_LCD_DisplayStringAt(x, line_y,
                                    (uint8_t*)buf, CENTER_MODE);

        line_y += BSP_LCD_GetFont()->Height;
        line = strtok(NULL, "\n");
    }
}
```

Code Listing 3: LCD display text printing functions (display.c)

7.3.4 MFCCs Calculation

mfcc.c

```
/* mfcc.c
 * Simple MFCC implementation for single-frame extraction.
 * See header for usage notes.
 */

```

```c
#include "mfcc.h"
#include <stdlib.h>
#include <math.h>
#include <string.h>

#define PI 3.14159265358979323846f
#define EPS 1e-10f

/* Convert frequency (Hz) to Mel scale */
static float hz_to_mel(float hz)
{
    return 2595.0f * log10f(1.0f + hz / 700.0f);
}

/* Convert Mel to frequency (Hz) */
static float mel_to_hz(float mel)
{
    return 700.0f * (powf(10.0f, mel / 2595.0f) - 1.0f);
}

/* Next power of two >= v */
static uint32_t next_pow2(uint32_t v)
{
    uint32_t p = 1;
    while (p < v) p <<= 1;
    return p;
}

int mfcc_compute(const int16_t *pcm_frame, uint32_t frame_len,
   uint32_t sample_rate,
                 uint32_t num_mel, uint32_t num_coeff, float
   *mfcc_out)
{
    if (!pcm_frame || !mfcc_out || frame_len == 0 || num_mel ==
   0 || num_coeff == 0) return -1;

    /* FFT length (zero-pad if necessary) */
    uint32_t N = next_pow2(frame_len);
    uint32_t fft_bins = (N / 2) + 1;

    /* allocate working buffers */
    float *x = (float*)malloc(sizeof(float) * N); /* windowed
   signal, zero-padded */
    float *power = (float*)malloc(sizeof(float) * fft_bins);
    if (!x || !power) { free(x); free(power); return -2; }

```

```
48      /* Hamming window + convert to float */
49      for (uint32_t n = 0; n < frame_len; n++)
50      {
51          float w = 0.54f - 0.46f * cosf((2.0f * PI * n) /
    (frame_len - 1));
52          x[n] = ((float)pcm_frame[n]) * w;
53      }
54      for (uint32_t n = frame_len; n < N; n++) x[n] = 0.0f;
55
56      /* Compute DFT (naive O(N^2)) for bins 0..N/2 */
57      for (uint32_t k = 0; k < fft_bins; k++)
58      {
59          float re = 0.0f, im = 0.0f;
60          for (uint32_t n = 0; n < N; n++)
61          {
62              float angle = 2.0f * PI * ((float)k * (float)n) /
    (float)N;
63              re += x[n] * cosf(angle);
64              im -= x[n] * sinf(angle);
65          }
66          power[k] = re * re + im * im;
67      }
68
69      /* Build Mel filterbank */
70      float f_min = 0.0f;
71      float f_max = (float)sample_rate / 2.0f;
72      float mel_min = hz_to_mel(f_min);
73      float mel_max = hz_to_mel(f_max);
74
75      float *mel_points = (float*)malloc(sizeof(float) * (num_mel
    + 2));
76      uint32_t *bin = (uint32_t*)malloc(sizeof(uint32_t) *
    (num_mel + 2));
77      if (!mel_points || !bin) { free(x); free(power);
    free(mel_points); free(bin); return -3; }
78
79      for (uint32_t i = 0; i < num_mel + 2; i++)
80      {
81          float m = mel_min + ((float)i) * (mel_max - mel_min) /
    (float)(num_mel + 1);
82          mel_points[i] = mel_to_hz(m);
83          /* map to nearest FFT bin */
84          uint32_t b = (uint32_t)floorf((mel_points[i] / f_max) *
    (float)(fft_bins - 1) + 0.5f);
85          if (b >= fft_bins) b = fft_bins - 1;
86          bin[i] = b;
```

```
87      }
88
89      /* allocate filter weights flattened: num_mel * fft_bins */
90      float *filters = (float*)malloc(sizeof(float) * num_mel *
    fft_bins);
91      if (!filters) { free(x); free(power); free(mel_points);
    free(bin); return -4; }
92      memset(filters, 0, sizeof(float) * num_mel * fft_bins);
93
94      for (uint32_t m = 1; m <= num_mel; m++)
95      {
96          uint32_t f_m_minus = bin[m - 1];
97          uint32_t f_m = bin[m];
98          uint32_t f_m_plus = bin[m + 1];
99
100         if (f_m == f_m_minus) f_m = f_m_minus + 1;
101         for (uint32_t k = f_m_minus; k <= f_m; k++)
102         {
103             float val = 0.0f;
104             if (f_m != f_m_minus)
105                 val = ((float)(k - f_m_minus)) / ((float)(f_m -
    f_m_minus));
106             filters[(m - 1) * fft_bins + k] = val;
107         }
108         for (uint32_t k = f_m; k <= f_m_plus; k++)
109         {
110             float val = 0.0f;
111             if (f_m_plus != f_m)
112                 val = ((float)(f_m_plus - k)) /
    ((float)(f_m_plus - f_m));
113             filters[(m - 1) * fft_bins + k] = val;
114         }
115     }
116
117     /* Apply filters to power spectrum -> mel energies */
118     float *mel_energies = (float*)malloc(sizeof(float) *
    num_mel);
119     if (!mel_energies) { free(x); free(power); free(mel_points);
    free(bin); free(filters); return -5; }
120     for (uint32_t m = 0; m < num_mel; m++)
121     {
122         float sum = 0.0f;
123         for (uint32_t k = 0; k < fft_bins; k++) sum += power[k]
    * filters[m * fft_bins + k];
124         if (sum < EPS) sum = EPS;
125         mel_energies[m] = logf(sum);
```

```
126     }
127
128     /* DCT-II to produce MFCCs */
129     for (uint32_t i = 0; i < num_coeff; i++)
130     {
131         float s = 0.0f;
132         for (uint32_t j = 0; j < num_mel; j++)
133         {
134             s += mel_energies[j] * cosf(PI * (float)i *
    ((float)j + 0.5f) / (float)num_mel);
135         }
136         mfcc_out[i] = s;
137     }
138
139     /* cleanup */
140     free(x);
141     free(power);
142     free(mel_points);
143     free(bin);
144     free(filters);
145     free(mel_energies);
146
147     return 0;
148 }
149
150
151 int mfcc_compute_logmel(const int16_t *pcm_frame, uint32_t
    frame_len, uint32_t sample_rate,
152                         uint32_t num_mel, float *logmel_out)
153 {
154     if (!pcm_frame || !logmel_out || frame_len == 0 || num_mel
    == 0) return -1;
155
156     uint32_t N = next_pow2(frame_len);
157     uint32_t fft_bins = (N / 2) + 1;
158
159     /* allocate working buffers */
160     float *x = (float*)malloc(sizeof(float) * N);
161     float *power = (float*)malloc(sizeof(float) * fft_bins);
162     if (!x || !power) { free(x); free(power); return -2; }
163
164     /* Hamming window + convert to float */
165     for (uint32_t n = 0; n < frame_len; n++)
166     {
167         float w = 0.54f - 0.46f * cosf((2.0f * PI * n) /
    (frame_len - 1));
```

```
        x[n] = ((float)pcm_frame[n]) * w;
    }
    for (uint32_t n = frame_len; n < N; n++) x[n] = 0.0f;

    /* Compute DFT (naive) and power spectrum */
    for (uint32_t k = 0; k < fft_bins; k++)
    {
        float re = 0.0f, im = 0.0f;
        for (uint32_t n = 0; n < N; n++)
        {
            float angle = 2.0f * PI * ((float)k * (float)n) /
(float)N;
            re += x[n] * cosf(angle);
            im -= x[n] * sinf(angle);
        }
        power[k] = re * re + im * im;
    }

    /* Build mel bin centers */
    float f_min = 0.0f;
    float f_max = (float)sample_rate / 2.0f;
    float mel_min = hz_to_mel(f_min);
    float mel_max = hz_to_mel(f_max);

    float *mel_hz = (float*)malloc(sizeof(float) * (num_mel +
2));
    uint32_t *bin = (uint32_t*)malloc(sizeof(uint32_t) *
(num_mel + 2));
    if (!mel_hz || !bin) { free(x); free(power); free(mel_hz);
free(bin); return -3; }

    for (uint32_t i = 0; i < num_mel + 2; i++)
    {
        float m = mel_min + ((float)i) * (mel_max - mel_min) /
(float)(num_mel + 1);
        mel_hz[i] = mel_to_hz(m);
        uint32_t b = (uint32_t)floorf((mel_hz[i] / f_max) *
(float)(fft_bins - 1) + 0.5f);
        if (b >= fft_bins) b = fft_bins - 1;
        bin[i] = b;
    }

    /* For each mel filter, compute triangular-weighted energy
from power spectrum */
    for (uint32_t m = 1; m <= num_mel; m++)
    {
```

```
207         uint32_t start = bin[m - 1];
208         uint32_t center = bin[m];
209         uint32_t end = bin[m + 1];
210         float sum = 0.0f;
211 
212         /* ascending slope */
213         if (center > start)
214         {
215             for (uint32_t k = start; k <= center; k++)
216             {
217                 float w = ((float)(k - start)) / ((float)(center
    - start));
218                 sum += power[k] * w;
219             }
220         }
221         else if (center == start)
222         {
223             /* single bin center */
224             sum += power[center];
225         }
226 
227         /* descending slope */
228         if (end > center)
229         {
230             for (uint32_t k = center; k <= end; k++)
231             {
232                 float w = ((float)(end - k)) / ((float)(end -
    center));
233                 sum += power[k] * w;
234             }
235         }
236 
237         if (sum < EPS) sum = EPS;
238         logmel_out[m - 1] = logf(sum);
239     }
240 
241     free(x);
242     free(power);
243     free(mel_hz);
244     free(bin);
245 
246     return 0;
247 }
```

Code Listing 4: MFCC and Log-Mel feature extraction source code (mfcc.c)

7.3.5 Touch Button Control

touch_button.c

```c
#include "touch_button.h"
#include "stm32746g_discovery_ts.h"
#include "stm32746g_discovery_lcd.h"
#include <string.h>
#include <stdbool.h>

static TS_StateTypeDef TS_State;

void touchInit(void)
{
    /* Initialize touchscreen */
    BSP_TS_Init(BSP_LCD_GetXSize(), BSP_LCD_GetYSize());
}

void drawButton(TS_Button* btn)
{
    /* Draw filled rectangle for button */
    BSP_LCD_SetTextColor(btn->color);
    BSP_LCD_FillRect(btn->x, btn->y, btn->width, btn->height);

    /* Draw button border */
    BSP_LCD_SetTextColor(LCD_COLOR_WHITE);
    BSP_LCD_DrawRect(btn->x, btn->y, btn->width, btn->height);

    /* Draw centered label text */
    BSP_LCD_SetTextColor(btn->textColor);
    uint16_t textX = btn->x + (btn->width / 2)
                   - (strlen(btn->label) *
                      BSP_LCD_GetFont()->Width / 2);
    uint16_t textY = btn->y + (btn->height / 2)
                   - (BSP_LCD_GetFont()->Height / 2);

    BSP_LCD_DisplayStringAt(textX, textY,
                            (uint8_t*)btn->label,
                            LEFT_MODE);
}

bool isButtonPressed(TS_Button* btn)
{
    BSP_TS_GetState(&TS_State);

    if (TS_State.touchDetected > 0)
    {
```

```
        uint16_t tx = TS_State.touchX[0];
        uint16_t ty = TS_State.touchY[0];

        if (tx >= btn->x &&
            tx <= (btn->x + btn->width) &&
            ty >= btn->y &&
            ty <= (btn->y + btn->height))
        {
            return true;
        }
    }
    return false;
}

/* Convenience function to draw a RECORD button */
void drawRecordButton(TS_Button* btn)
{
    btn->x = 150;
    btn->y = 100;
    btn->width = 180;
    btn->height = 60;
    btn->color = LCD_COLOR_RED;
    btn->textColor = LCD_COLOR_WHITE;
    strcpy(btn->label, "RECORD");

    drawButton(btn);
}
```

Code Listing 5: Touch button control source code (touch_button.c)

7.4 Role-Based Control and Audio Classification ai model execute

role.c

```
#include <display.h>
#include "role.h"
#include "stm32746g_discovery_lcd.h"
#include "stm32746g_discovery_ts.h"
#include "mfcc.h"
#include "record.h"
#include <stdio.h>
#include <math.h>

/* X-CUBE-AI model headers */
```

```c
#include "audio.h"
#include "audio_data.h"
#include "ai_platform.h"

/* AI helper prototypes */
extern ai_error ai_audio_create_and_init(ai_handle* network,
        const ai_handle activations[],
        const ai_handle weights[]);
extern ai_buffer* ai_audio_inputs_get(ai_handle network, ai_u16
   *n_buffer);
extern ai_buffer* ai_audio_outputs_get(ai_handle network, ai_u16
   *n_buffer);
extern ai_i32 ai_audio_run(ai_handle network,
        const ai_buffer* input, ai_buffer* output);

ROLE_TypeDef role = RECORD_START;

void roleInit(void)
{
    FRESULT res;

    serialInit(&vcp, USART1,
               usart1_buffer, sizeof(usart1_buffer));

    res = f_mount(&SDFatFS, (TCHAR const*)SDPath, 0);
    if (res != FR_OK)
        serialPrintln(&vcp, "SD mount error: %d", res);

    LCD_Config();
    HAL_Delay(500);
}

void roleNode(void)
{
    switch (role)
    {
    case RECORD_START:
        recordStart();
        text(0, 120, 'c', "Recording...", 'r', 'k');
        role = RECORD_PROCESS;
        break;

    case RECORD_PROCESS:
        if (recordProcess() == AUDIO_ERROR_EOF)
            role = RECORD_STOP;
        break;
```

```
55
56      case RECORD_STOP:
57          recordStop();
58          text(100, 120, 'c', "Recording stopped", 'r', 'b');
59          HAL_Delay(1000);
60
61          /* MFCC + AI inference */
62          {
63              static ai_handle audio_net = AI_HANDLE_NULL;
64
65              const float ai_input_scale = 2.7867f;
66              const int   ai_input_zp    = 99;
67              const float ai_output_scale = 0.00390625f;
68              const int   ai_output_zp    = -128;
69
70              int16_t pcm_frame[512];
71              float   logmel[40];
72              int8_t  qin[40];
73              int8_t  qout[AI_AUDIO_OUT_1_SIZE];
74
75              uint32_t sdram_size =
76                  RECORD_GetSDRAMSizeHalfWords();
77
78              if (sdram_size >= 512)
79              {
80                  uint16_t *src = RECORD_GetSDRAMPtr();
81                  for (int i = 0; i < 512; i++)
82                      pcm_frame[i] = (int16_t)src[i];
83
84                  if (mfcc_compute_logmel(pcm_frame, 512,
85                      DEFAULT_AUDIO_IN_FREQ, 40, logmel) == 0)
86                  {
87                      for (int i = 0; i < 40; i++)
88                      {
89                          int q = (int)roundf(
90                              logmel[i] / ai_input_scale)
91                              + ai_input_zp;
92                          if (q > 127) q = 127;
93                          if (q < -128) q = -128;
94                          qin[i] = (int8_t)q;
95                      }
96
97                      if (audio_net == AI_HANDLE_NULL)
98                          ai_audio_create_and_init(
99                              &audio_net,
100
```

```
    AI_AUDIO_DATA_ACTIVATIONS_TABLE_GET(),
101                             AI_AUDIO_DATA_WEIGHTS_TABLE_GET());
102
103                     ai_u16 n_in, n_out;
104                     ai_buffer* inb =
105                         ai_audio_inputs_get(audio_net, &n_in);
106                     ai_buffer* outb =
107                         ai_audio_outputs_get(audio_net, &n_out);
108
109                     inb[0].data = AI_HANDLE_PTR(qin);
110                     outb[0].data = AI_HANDLE_PTR(qout);
111
112                     if (ai_audio_run(audio_net,
113                                      inb, outb) > 0)
114                     {
115                         int best = 0;
116                         float bestv = -1e30f;
117
118                         for (int k = 0;
119                              k < AI_AUDIO_OUT_1_SIZE; k++)
120                         {
121                             float f =
122                               (qout[k] - ai_output_zp)
123                               * ai_output_scale;
124                             if (f > bestv)
125                             {
126                                 bestv = f;
127                                 best = k;
128                             }
129                         }
130
131                         const char *labels[] =
132                         {"fahim","talukdar",
133                          "nayeem","Fa_him","imran"};
134
135                         char buf[64];
136                         snprintf(buf, sizeof(buf),
137                           "Idx:%d %s", best, labels[best]);
138
139                         text(0, 40, 'c', buf, 'g', 'k');
140                         HAL_Delay(10000);
141                     }
142                 }
143             }
144         }
145         break;
```

```
    }
}
```

Code Listing 6: Role-based control and AI inference source code (role.c)

7.4.1 main source code setup

main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "string.h"
#include "cmsis_os.h"
#include "fatfs.h"
#include "usb_host.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "role.h"
```

```
31 /* USER CODE END Includes */
32
33 /* Private typedef
     -----------------------------------------------------------*/
34 /* USER CODE BEGIN PTD */
35
36 /* USER CODE END PTD */
37
38 /* Private define
     ------------------------------------------------------------*/
39 /* USER CODE BEGIN PD */
40 /* USER CODE END PD */
41
42 /* Private macro
     -------------------------------------------------------------*/
43 /* USER CODE BEGIN PM */
44
45 /* USER CODE END PM */
46
47 /* Private variables
     ---------------------------------------------------------*/
48 #if defined ( __ICCARM__ ) /*!< IAR Compiler */
49 #pragma location=0x2004c000
50 ETH_DMADescTypeDef  DMARxDscrTab[ETH_RX_DESC_CNT]; /* Ethernet
     Rx DMA Descriptors */
51 #pragma location=0x2004c0a0
52 ETH_DMADescTypeDef  DMATxDscrTab[ETH_TX_DESC_CNT]; /* Ethernet
     Tx DMA Descriptors */
53
54 #elif defined ( __CC_ARM )  /* MDK ARM Compiler */
55
56 __attribute__((at(0x2004c000))) ETH_DMADescTypeDef
     DMARxDscrTab[ETH_RX_DESC_CNT]; /* Ethernet Rx DMA Descriptors
     */
57 __attribute__((at(0x2004c0a0))) ETH_DMADescTypeDef
     DMATxDscrTab[ETH_TX_DESC_CNT]; /* Ethernet Tx DMA Descriptors
     */
58
59 #elif defined ( __GNUC__ ) /* GNU Compiler */
60 ETH_DMADescTypeDef DMARxDscrTab[ETH_RX_DESC_CNT]
     __attribute__((section(".RxDecripSection"))); /* Ethernet Rx
     DMA Descriptors */
61 ETH_DMADescTypeDef DMATxDscrTab[ETH_TX_DESC_CNT]
     __attribute__((section(".TxDecripSection")));   /* Ethernet
     Tx DMA Descriptors */
62
```

```
#endif

ETH_TxPacketConfig TxConfig;

ADC_HandleTypeDef hadc3;

CRC_HandleTypeDef hcrc;

DCMI_HandleTypeDef hdcmi;

DMA2D_HandleTypeDef hdma2d;

ETH_HandleTypeDef heth;

I2C_HandleTypeDef hi2c1;
I2C_HandleTypeDef hi2c3;

LTDC_HandleTypeDef hltdc;

QSPI_HandleTypeDef hqspi;

RTC_HandleTypeDef hrtc;

SAI_HandleTypeDef hsai_BlockA2;
SAI_HandleTypeDef hsai_BlockB2;
DMA_HandleTypeDef hdma_sai2_a;
DMA_HandleTypeDef hdma_sai2_b;

SD_HandleTypeDef hsd1;
DMA_HandleTypeDef hdma_sdmmc1_rx;
DMA_HandleTypeDef hdma_sdmmc1_tx;

SPDIFRX_HandleTypeDef hspdif;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim5;
TIM_HandleTypeDef htim8;
TIM_HandleTypeDef htim12;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart6;

SDRAM_HandleTypeDef hsdram1;

```

```
109 osThreadId defaultTaskHandle;
110 /* USER CODE BEGIN PV */
111
112 /* USER CODE END PV */
113
114 /* Private function prototypes
    -----------------------------------------------*/
115 void SystemClock_Config(void);
116 void PeriphCommonClock_Config(void);
117 static void MX_GPIO_Init(void);
118 static void MX_DMA_Init(void);
119 static void MX_ADC3_Init(void);
120 static void MX_CRC_Init(void);
121 static void MX_DCMI_Init(void);
122 static void MX_DMA2D_Init(void);
123 static void MX_ETH_Init(void);
124 static void MX_FMC_Init(void);
125 static void MX_I2C1_Init(void);
126 static void MX_I2C3_Init(void);
127 static void MX_LTDC_Init(void);
128 static void MX_QUADSPI_Init(void);
129 static void MX_RTC_Init(void);
130 static void MX_SAI2_Init(void);
131 static void MX_SDMMC1_SD_Init(void);
132 static void MX_SPDIFRX_Init(void);
133 static void MX_TIM1_Init(void);
134 static void MX_TIM2_Init(void);
135 static void MX_TIM3_Init(void);
136 static void MX_TIM5_Init(void);
137 static void MX_TIM8_Init(void);
138 static void MX_TIM12_Init(void);
139 static void MX_USART1_UART_Init(void);
140 static void MX_USART6_UART_Init(void);
141 void StartDefaultTask(void const * argument);
142
143 /* USER CODE BEGIN PFP */
144
145 /* USER CODE END PFP */
146
147 /* Private user code
    ---------------------------------------------------------*/
148 /* USER CODE BEGIN 0 */
149
150 /* USER CODE END 0 */
151
152 /**
```

```
153   * @brief  The application entry point.
154   * @retval int
155   */
156 int main(void)
157 {
158   /* USER CODE BEGIN 1 */
159
160   /* USER CODE END 1 */
161
162   /* Enable
       I-Cache---------------------------------------------------------*/
163   SCB_EnableICache();
164
165   /* Enable
       D-Cache---------------------------------------------------------*/
166   SCB_EnableDCache();
167
168   /* MCU
       Configuration--------------------------------------------------------*/
169
170   /* Reset of all peripherals, Initializes the Flash interface
       and the Systick. */
171   HAL_Init();
172
173   /* USER CODE BEGIN Init */
174
175   /* USER CODE END Init */
176
177   /* Configure the system clock */
178   SystemClock_Config();
179
180 /* Configure the peripherals common clocks */
181   PeriphCommonClock_Config();
182
183   /* USER CODE BEGIN SysInit */
184
185   /* USER CODE END SysInit */
186
187   /* Initialize all configured peripherals */
188   MX_GPIO_Init();
189   MX_DMA_Init();
190   MX_ADC3_Init();
191   MX_CRC_Init();
192   MX_DCMI_Init();
193   MX_DMA2D_Init();
194   MX_ETH_Init();
```

```
195   MX_FMC_Init();
196   MX_I2C1_Init();
197   MX_I2C3_Init();
198   MX_LTDC_Init();
199   MX_QUADSPI_Init();
200   MX_RTC_Init();
201   MX_SAI2_Init();
202   MX_SDMMC1_SD_Init();
203   MX_SPDIFRX_Init();
204   MX_TIM1_Init();
205   MX_TIM2_Init();
206   MX_TIM3_Init();
207   MX_TIM5_Init();
208   MX_TIM8_Init();
209   MX_TIM12_Init();
210   MX_USART1_UART_Init();
211   MX_USART6_UART_Init();
212   MX_FATFS_Init();
213   /* USER CODE BEGIN 2 */
214
215   /* USER CODE END 2 */
216
217   /* USER CODE BEGIN RTOS_MUTEX */
218   /* add mutexes, ... */
219   /* USER CODE END RTOS_MUTEX */
220
221   /* USER CODE BEGIN RTOS_SEMAPHORES */
222   /* add semaphores, ... */
223   /* USER CODE END RTOS_SEMAPHORES */
224
225   /* USER CODE BEGIN RTOS_TIMERS */
226   /* start timers, add new ones, ... */
227   /* USER CODE END RTOS_TIMERS */
228
229   /* USER CODE BEGIN RTOS_QUEUES */
230   /* add queues, ... */
231   /* USER CODE END RTOS_QUEUES */
232
233   /* Create the thread(s) */
234   /* definition and creation of defaultTask */
235   osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal,
       0, 4096);
236   defaultTaskHandle = osThreadCreate(osThread(defaultTask),
       NULL);
237
238   /* USER CODE BEGIN RTOS_THREADS */
```

```
239   /* add threads, ... */
240   /* USER CODE END RTOS_THREADS */
241
242   /* Start scheduler */
243   osKernelStart();
244
245   /* We should never get here as control is now taken by the
       scheduler */
246   /* Infinite loop */
247   /* USER CODE BEGIN WHILE */
248   while (1)
249   {
250     /* USER CODE END WHILE */
251
252     /* USER CODE BEGIN 3 */
253   }
254   /* USER CODE END 3 */
255 }
256
257 /**
258   * @brief System Clock Configuration
259   * @retval None
260   */
261 void SystemClock_Config(void)
262 {
263   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
264   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
265
266   /** Configure LSE Drive Capability
267   */
268   HAL_PWR_EnableBkUpAccess();
269
270   /** Configure the main internal regulator output voltage
271   */
272   __HAL_RCC_PWR_CLK_ENABLE();
273   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
274
275   /** Initializes the RCC Oscillators according to the specified
       parameters
276   * in the RCC_OscInitTypeDef structure.
277   */
278   RCC_OscInitStruct.OscillatorType =
       RCC_OSCILLATORTYPE_LSI|RCC_OSCILLATORTYPE_HSE;
279   RCC_OscInitStruct.HSEState = RCC_HSE_ON;
280   RCC_OscInitStruct.LSIState = RCC_LSI_ON;
281   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
```

```
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 25;
  RCC_OscInitStruct.PLL.PLLN = 432;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 9;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Activate the Over-Drive mode
  */
  if (HAL_PWREx_EnableOverDrive() != 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_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_7)
   != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief Peripherals Common Clock Configuration
  * @retval None
  */
void PeriphCommonClock_Config(void)
{
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

  /** Initializes the peripherals clock
  */
  PeriphClkInitStruct.PeriphClockSelection =
```

```
    RCC_PERIPHCLK_LTDC|RCC_PERIPHCLK_SAI2
325
    |RCC_PERIPHCLK_SDMMC1|RCC_PERIPHCLK_CLK48;
326   PeriphClkInitStruct.PLLSAI.PLLSAIN = 384;
327   PeriphClkInitStruct.PLLSAI.PLLSAIR = 5;
328   PeriphClkInitStruct.PLLSAI.PLLSAIQ = 2;
329   PeriphClkInitStruct.PLLSAI.PLLSAIP = RCC_PLLSAIP_DIV8;
330   PeriphClkInitStruct.PLLSAIDivQ = 1;
331   PeriphClkInitStruct.PLLSAIDivR = RCC_PLLSAIDIVR_8;
332   PeriphClkInitStruct.Sai2ClockSelection =
      RCC_SAI2CLKSOURCE_PLLSAI;
333   PeriphClkInitStruct.Clk48ClockSelection =
      RCC_CLK48SOURCE_PLLSAIP;
334   PeriphClkInitStruct.Sdmmc1ClockSelection =
      RCC_SDMMC1CLKSOURCE_CLK48;
335   if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
336   {
337     Error_Handler();
338   }
339 }
340
341 /**
342   * @brief ADC3 Initialization Function
343   * @param None
344   * @retval None
345   */
346 static void MX_ADC3_Init(void)
347 {
348
349   /* USER CODE BEGIN ADC3_Init 0 */
350
351   /* USER CODE END ADC3_Init 0 */
352
353   ADC_ChannelConfTypeDef sConfig = {0};
354
355   /* USER CODE BEGIN ADC3_Init 1 */
356
357   /* USER CODE END ADC3_Init 1 */
358
359   /** Configure the global features of the ADC (Clock,
      Resolution, Data Alignment and number of conversion)
360   */
361   hadc3.Instance = ADC3;
362   hadc3.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
363   hadc3.Init.Resolution = ADC_RESOLUTION_12B;
364   hadc3.Init.ScanConvMode = ADC_SCAN_DISABLE;
```

```
365   hadc3.Init.ContinuousConvMode = DISABLE;
366   hadc3.Init.DiscontinuousConvMode = DISABLE;
367   hadc3.Init.ExternalTrigConvEdge =
       ADC_EXTERNALTRIGCONVEDGE_NONE;
368   hadc3.Init.ExternalTrigConv = ADC_SOFTWARE_START;
369   hadc3.Init.DataAlign = ADC_DATAALIGN_RIGHT;
370   hadc3.Init.NbrOfConversion = 1;
371   hadc3.Init.DMAContinuousRequests = DISABLE;
372   hadc3.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
373   if (HAL_ADC_Init(&hadc3) != HAL_OK)
374   {
375     Error_Handler();
376   }
377
378   /** Configure for the selected ADC regular channel its
       corresponding rank in the sequencer and its sample time.
379   */
380   sConfig.Channel = ADC_CHANNEL_4;
381   sConfig.Rank = ADC_REGULAR_RANK_1;
382   sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
383   if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
384   {
385     Error_Handler();
386   }
387   /* USER CODE BEGIN ADC3_Init 2 */
388
389   /* USER CODE END ADC3_Init 2 */
390
391 }
392
393 /**
394   * @brief CRC Initialization Function
395   * @param None
396   * @retval None
397   */
398 static void MX_CRC_Init(void)
399 {
400
401   /* USER CODE BEGIN CRC_Init 0 */
402
403   /* USER CODE END CRC_Init 0 */
404
405   /* USER CODE BEGIN CRC_Init 1 */
406
407   /* USER CODE END CRC_Init 1 */
408   hcrc.Instance = CRC;
```

```
409   hcrc.Init.DefaultPolynomialUse = DEFAULT_POLYNOMIAL_ENABLE;
410   hcrc.Init.DefaultInitValueUse = DEFAULT_INIT_VALUE_ENABLE;
411   hcrc.Init.InputDataInversionMode =
       CRC_INPUTDATA_INVERSION_NONE;
412   hcrc.Init.OutputDataInversionMode =
       CRC_OUTPUTDATA_INVERSION_DISABLE;
413   hcrc.InputDataFormat = CRC_INPUTDATA_FORMAT_BYTES;
414   if (HAL_CRC_Init(&hcrc) != HAL_OK)
415   {
416     Error_Handler();
417   }
418   /* USER CODE BEGIN CRC_Init 2 */
419
420   /* USER CODE END CRC_Init 2 */
421
422 }
423
424 /**
425   * @brief DCMI Initialization Function
426   * @param None
427   * @retval None
428   */
429 static void MX_DCMI_Init(void)
430 {
431
432   /* USER CODE BEGIN DCMI_Init 0 */
433
434   /* USER CODE END DCMI_Init 0 */
435
436   /* USER CODE BEGIN DCMI_Init 1 */
437
438   /* USER CODE END DCMI_Init 1 */
439   hdcmi.Instance = DCMI;
440   hdcmi.Init.SynchroMode = DCMI_SYNCHRO_HARDWARE;
441   hdcmi.Init.PCKPolarity = DCMI_PCKPOLARITY_FALLING;
442   hdcmi.Init.VSPolarity = DCMI_VSPOLARITY_LOW;
443   hdcmi.Init.HSPolarity = DCMI_HSPOLARITY_LOW;
444   hdcmi.Init.CaptureRate = DCMI_CR_ALL_FRAME;
445   hdcmi.Init.ExtendedDataMode = DCMI_EXTEND_DATA_8B;
446   hdcmi.Init.JPEGMode = DCMI_JPEG_DISABLE;
447   hdcmi.Init.ByteSelectMode = DCMI_BSM_ALL;
448   hdcmi.Init.ByteSelectStart = DCMI_OEBS_ODD;
449   hdcmi.Init.LineSelectMode = DCMI_LSM_ALL;
450   hdcmi.Init.LineSelectStart = DCMI_OELS_ODD;
451   if (HAL_DCMI_Init(&hdcmi) != HAL_OK)
452   {
```

```
453       Error_Handler();
454     }
455     /* USER CODE BEGIN DCMI_Init 2 */
456
457     /* USER CODE END DCMI_Init 2 */
458
459   }
460
461   /**
462     * @brief DMA2D Initialization Function
463     * @param None
464     * @retval None
465     */
466   static void MX_DMA2D_Init(void)
467   {
468
469     /* USER CODE BEGIN DMA2D_Init 0 */
470
471     /* USER CODE END DMA2D_Init 0 */
472
473     /* USER CODE BEGIN DMA2D_Init 1 */
474
475     /* USER CODE END DMA2D_Init 1 */
476     hdma2d.Instance = DMA2D;
477     hdma2d.Init.Mode = DMA2D_M2M;
478     hdma2d.Init.ColorMode = DMA2D_OUTPUT_ARGB8888;
479     hdma2d.Init.OutputOffset = 0;
480     hdma2d.LayerCfg[1].InputOffset = 0;
481     hdma2d.LayerCfg[1].InputColorMode = DMA2D_INPUT_ARGB8888;
482     hdma2d.LayerCfg[1].AlphaMode = DMA2D_NO_MODIF_ALPHA;
483     hdma2d.LayerCfg[1].InputAlpha = 0;
484     if (HAL_DMA2D_Init(&hdma2d) != HAL_OK)
485     {
486       Error_Handler();
487     }
488     if (HAL_DMA2D_ConfigLayer(&hdma2d, 1) != HAL_OK)
489     {
490       Error_Handler();
491     }
492     /* USER CODE BEGIN DMA2D_Init 2 */
493
494     /* USER CODE END DMA2D_Init 2 */
495
496   }
497
498   /**
```

```
499   * @brief ETH Initialization Function
500   * @param None
501   * @retval None
502   */
503 static void MX_ETH_Init(void)
504 {
505
506   /* USER CODE BEGIN ETH_Init 0 */
507
508   /* USER CODE END ETH_Init 0 */
509
510    static uint8_t MACAddr[6];
511
512   /* USER CODE BEGIN ETH_Init 1 */
513
514   /* USER CODE END ETH_Init 1 */
515   heth.Instance = ETH;
516   MACAddr[0] = 0x00;
517   MACAddr[1] = 0x80;
518   MACAddr[2] = 0xE1;
519   MACAddr[3] = 0x00;
520   MACAddr[4] = 0x00;
521   MACAddr[5] = 0x00;
522   heth.Init.MACAddr = &MACAddr[0];
523   heth.Init.MediaInterface = HAL_ETH_RMII_MODE;
524   heth.Init.TxDesc = DMATxDscrTab;
525   heth.Init.RxDesc = DMARxDscrTab;
526   heth.Init.RxBuffLen = 1524;
527
528   /* USER CODE BEGIN MACADDRESS */
529
530   /* USER CODE END MACADDRESS */
531
532   if (HAL_ETH_Init(&heth) != HAL_OK)
533   {
534     Error_Handler();
535   }
536
537   memset(&TxConfig, 0 , sizeof(ETH_TxPacketConfig));
538   TxConfig.Attributes = ETH_TX_PACKETS_FEATURES_CSUM |
       ETH_TX_PACKETS_FEATURES_CRCPAD;
539   TxConfig.ChecksumCtrl =
       ETH_CHECKSUM_IPHDR_PAYLOAD_INSERT_PHDR_CALC;
540   TxConfig.CRCPadCtrl = ETH_CRC_PAD_INSERT;
541   /* USER CODE BEGIN ETH_Init 2 */
542
```

```
543   /* USER CODE END ETH_Init 2 */
544
545 }
546
547 /**
548   * @brief I2C1 Initialization Function
549   * @param None
550   * @retval None
551   */
552 static void MX_I2C1_Init(void)
553 {
554
555   /* USER CODE BEGIN I2C1_Init 0 */
556
557   /* USER CODE END I2C1_Init 0 */
558
559   /* USER CODE BEGIN I2C1_Init 1 */
560
561   /* USER CODE END I2C1_Init 1 */
562   hi2c1.Instance = I2C1;
563   hi2c1.Init.Timing = 0x00C0EAFF;
564   hi2c1.Init.OwnAddress1 = 0;
565   hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
566   hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
567   hi2c1.Init.OwnAddress2 = 0;
568   hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
569   hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
570   hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
571   if (HAL_I2C_Init(&hi2c1) != HAL_OK)
572   {
573     Error_Handler();
574   }
575
576   /** Configure Analogue filter
577   */
578   if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1,
      I2C_ANALOGFILTER_ENABLE) != HAL_OK)
579   {
580     Error_Handler();
581   }
582
583   /** Configure Digital filter
584   */
585   if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
586   {
587     Error_Handler();
```

```
588   }
589   /* USER CODE BEGIN I2C1_Init 2 */
590
591   /* USER CODE END I2C1_Init 2 */
592
593 }
594
595 /**
596   * @brief I2C3 Initialization Function
597   * @param None
598   * @retval None
599   */
600 static void MX_I2C3_Init(void)
601 {
602
603   /* USER CODE BEGIN I2C3_Init 0 */
604
605   /* USER CODE END I2C3_Init 0 */
606
607   /* USER CODE BEGIN I2C3_Init 1 */
608
609   /* USER CODE END I2C3_Init 1 */
610   hi2c3.Instance = I2C3;
611   hi2c3.Init.Timing = 0x00C0EAFF;
612   hi2c3.Init.OwnAddress1 = 0;
613   hi2c3.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
614   hi2c3.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
615   hi2c3.Init.OwnAddress2 = 0;
616   hi2c3.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
617   hi2c3.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
618   hi2c3.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
619   if (HAL_I2C_Init(&hi2c3) != HAL_OK)
620   {
621     Error_Handler();
622   }
623
624   /** Configure Analogue filter
625   */
626   if (HAL_I2CEx_ConfigAnalogFilter(&hi2c3,
      I2C_ANALOGFILTER_ENABLE) != HAL_OK)
627   {
628     Error_Handler();
629   }
630
631   /** Configure Digital filter
632   */
```

```
633   if (HAL_I2CEx_ConfigDigitalFilter(&hi2c3, 0) != HAL_OK)
634   {
635     Error_Handler();
636   }
637   /* USER CODE BEGIN I2C3_Init 2 */
638
639   /* USER CODE END I2C3_Init 2 */
640
641 }
642
643 /**
644   * @brief LTDC Initialization Function
645   * @param None
646   * @retval None
647   */
648 static void MX_LTDC_Init(void)
649 {
650
651   /* USER CODE BEGIN LTDC_Init 0 */
652
653   /* USER CODE END LTDC_Init 0 */
654
655   LTDC_LayerCfgTypeDef pLayerCfg = {0};
656
657   /* USER CODE BEGIN LTDC_Init 1 */
658
659   /* USER CODE END LTDC_Init 1 */
660   hltdc.Instance = LTDC;
661   hltdc.Init.HSPolarity = LTDC_HSPOLARITY_AL;
662   hltdc.Init.VSPolarity = LTDC_VSPOLARITY_AL;
663   hltdc.Init.DEPolarity = LTDC_DEPOLARITY_AL;
664   hltdc.Init.PCPolarity = LTDC_PCPOLARITY_IPC;
665   hltdc.Init.HorizontalSync = 40;
666   hltdc.Init.VerticalSync = 9;
667   hltdc.Init.AccumulatedHBP = 53;
668   hltdc.Init.AccumulatedVBP = 11;
669   hltdc.Init.AccumulatedActiveW = 533;
670   hltdc.Init.AccumulatedActiveH = 283;
671   hltdc.Init.TotalWidth = 565;
672   hltdc.Init.TotalHeigh = 285;
673   hltdc.Init.Backcolor.Blue = 0;
674   hltdc.Init.Backcolor.Green = 0;
675   hltdc.Init.Backcolor.Red = 0;
676   if (HAL_LTDC_Init(&hltdc) != HAL_OK)
677   {
678     Error_Handler();
```

```
679   }
680   pLayerCfg.WindowX0 = 0;
681   pLayerCfg.WindowX1 = 480;
682   pLayerCfg.WindowY0 = 0;
683   pLayerCfg.WindowY1 = 272;
684   pLayerCfg.PixelFormat = LTDC_PIXEL_FORMAT_RGB565;
685   pLayerCfg.Alpha = 255;
686   pLayerCfg.Alpha0 = 0;
687   pLayerCfg.BlendingFactor1 = LTDC_BLENDING_FACTOR1_PAxCA;
688   pLayerCfg.BlendingFactor2 = LTDC_BLENDING_FACTOR2_PAxCA;
689   pLayerCfg.FBStartAdress = 0xC0000000;
690   pLayerCfg.ImageWidth = 480;
691   pLayerCfg.ImageHeight = 272;
692   pLayerCfg.Backcolor.Blue = 0;
693   pLayerCfg.Backcolor.Green = 0;
694   pLayerCfg.Backcolor.Red = 0;
695   if (HAL_LTDC_ConfigLayer(&hltdc, &pLayerCfg, 0) != HAL_OK)
696   {
697     Error_Handler();
698   }
699   /* USER CODE BEGIN LTDC_Init 2 */
700
701   /* USER CODE END LTDC_Init 2 */
702
703 }
704
705 /**
706   * @brief QUADSPI Initialization Function
707   * @param None
708   * @retval None
709   */
710 static void MX_QUADSPI_Init(void)
711 {
712
713   /* USER CODE BEGIN QUADSPI_Init 0 */
714
715   /* USER CODE END QUADSPI_Init 0 */
716
717   /* USER CODE BEGIN QUADSPI_Init 1 */
718
719   /* USER CODE END QUADSPI_Init 1 */
720   /* QUADSPI parameter configuration*/
721   hqspi.Instance = QUADSPI;
722   hqspi.Init.ClockPrescaler = 1;
723   hqspi.Init.FifoThreshold = 4;
724   hqspi.Init.SampleShifting = QSPI_SAMPLE_SHIFTING_HALFCYCLE;
```

```
  hqspi.Init.FlashSize = 24;
  hqspi.Init.ChipSelectHighTime = QSPI_CS_HIGH_TIME_6_CYCLE;
  hqspi.Init.ClockMode = QSPI_CLOCK_MODE_0;
  hqspi.Init.FlashID = QSPI_FLASH_ID_1;
  hqspi.Init.DualFlash = QSPI_DUALFLASH_DISABLE;
  if (HAL_QSPI_Init(&hqspi) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN QUADSPI_Init 2 */

  /* USER CODE END QUADSPI_Init 2 */

}

/**
  * @brief RTC Initialization Function
  * @param None
  * @retval None
  */
static void MX_RTC_Init(void)
{

  /* USER CODE BEGIN RTC_Init 0 */

  /* USER CODE END RTC_Init 0 */

  RTC_TimeTypeDef sTime = {0};
  RTC_DateTypeDef sDate = {0};
  RTC_AlarmTypeDef sAlarm = {0};

  /* USER CODE BEGIN RTC_Init 1 */

  /* USER CODE END RTC_Init 1 */

  /** Initialize RTC Only
  */
  hrtc.Instance = RTC;
  hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
  hrtc.Init.AsynchPrediv = 127;
  hrtc.Init.SynchPrediv = 255;
  hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
  hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
  hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
```

```
771     Error_Handler();
772   }
773
774   /* USER CODE BEGIN Check_RTC_BKUP */
775
776   /* USER CODE END Check_RTC_BKUP */
777
778   /** Initialize RTC and set the Time and Date
779   */
780   sTime.Hours = 0x0;
781   sTime.Minutes = 0x0;
782   sTime.Seconds = 0x0;
783   sTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
784   sTime.StoreOperation = RTC_STOREOPERATION_RESET;
785   if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BCD) != HAL_OK)
786   {
787     Error_Handler();
788   }
789   sDate.WeekDay = RTC_WEEKDAY_MONDAY;
790   sDate.Month = RTC_MONTH_JANUARY;
791   sDate.Date = 0x1;
792   sDate.Year = 0x0;
793   if (HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BCD) != HAL_OK)
794   {
795     Error_Handler();
796   }
797
798   /** Enable the Alarm A
799   */
800   sAlarm.AlarmTime.Hours = 0x0;
801   sAlarm.AlarmTime.Minutes = 0x0;
802   sAlarm.AlarmTime.Seconds = 0x0;
803   sAlarm.AlarmTime.SubSeconds = 0x0;
804   sAlarm.AlarmTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
805   sAlarm.AlarmTime.StoreOperation = RTC_STOREOPERATION_RESET;
806   sAlarm.AlarmMask = RTC_ALARMMASK_NONE;
807   sAlarm.AlarmSubSecondMask = RTC_ALARMSUBSECONDMASK_ALL;
808   sAlarm.AlarmDateWeekDaySel = RTC_ALARMDATEWEEKDAYSEL_DATE;
809   sAlarm.AlarmDateWeekDay = 0x1;
810   sAlarm.Alarm = RTC_ALARM_A;
811   if (HAL_RTC_SetAlarm(&hrtc, &sAlarm, RTC_FORMAT_BCD) != HAL_OK)
812   {
813     Error_Handler();
814   }
815
816   /** Enable the Alarm B
```

```
817   */
818   sAlarm.Alarm = RTC_ALARM_B;
819   if (HAL_RTC_SetAlarm(&hrtc, &sAlarm, RTC_FORMAT_BCD) != HAL_OK)
820   {
821     Error_Handler();
822   }
823
824   /** Enable the TimeStamp
825   */
826   if (HAL_RTCEx_SetTimeStamp(&hrtc, RTC_TIMESTAMPEDGE_RISING,
       RTC_TIMESTAMPPIN_POS1) != HAL_OK)
827   {
828     Error_Handler();
829   }
830   /* USER CODE BEGIN RTC_Init 2 */
831
832   /* USER CODE END RTC_Init 2 */
833
834 }
835
836 /**
837   * @brief SAI2 Initialization Function
838   * @param None
839   * @retval None
840   */
841 static void MX_SAI2_Init(void)
842 {
843
844   /* USER CODE BEGIN SAI2_Init 0 */
845
846   /* USER CODE END SAI2_Init 0 */
847
848   /* USER CODE BEGIN SAI2_Init 1 */
849
850   /* USER CODE END SAI2_Init 1 */
851   hsai_BlockA2.Instance = SAI2_Block_A;
852   hsai_BlockA2.Init.Protocol = SAI_FREE_PROTOCOL;
853   hsai_BlockA2.Init.AudioMode = SAI_MODEMASTER_TX;
854   hsai_BlockA2.Init.DataSize = SAI_DATASIZE_8;
855   hsai_BlockA2.Init.FirstBit = SAI_FIRSTBIT_MSB;
856   hsai_BlockA2.Init.ClockStrobing =
       SAI_CLOCKSTROBING_FALLINGEDGE;
857   hsai_BlockA2.Init.Synchro = SAI_ASYNCHRONOUS;
858   hsai_BlockA2.Init.OutputDrive = SAI_OUTPUTDRIVE_DISABLE;
859   hsai_BlockA2.Init.NoDivider = SAI_MASTERDIVIDER_ENABLE;
860   hsai_BlockA2.Init.FIFOThreshold = SAI_FIFOTHRESHOLD_EMPTY;
```

```
861   hsai_BlockA2.Init.AudioFrequency = SAI_AUDIO_FREQUENCY_192K;
862   hsai_BlockA2.Init.SynchroExt = SAI_SYNCEXT_DISABLE;
863   hsai_BlockA2.Init.MonoStereoMode = SAI_STEREOMODE;
864   hsai_BlockA2.Init.CompandingMode = SAI_NOCOMPANDING;
865   hsai_BlockA2.Init.TriState = SAI_OUTPUT_NOTRELEASED;
866   hsai_BlockA2.FrameInit.FrameLength = 8;
867   hsai_BlockA2.FrameInit.ActiveFrameLength = 1;
868   hsai_BlockA2.FrameInit.FSDefinition = SAI_FS_STARTFRAME;
869   hsai_BlockA2.FrameInit.FSPolarity = SAI_FS_ACTIVE_LOW;
870   hsai_BlockA2.FrameInit.FSOffset = SAI_FS_FIRSTBIT;
871   hsai_BlockA2.SlotInit.FirstBitOffset = 0;
872   hsai_BlockA2.SlotInit.SlotSize = SAI_SLOTSIZE_DATASIZE;
873   hsai_BlockA2.SlotInit.SlotNumber = 1;
874   hsai_BlockA2.SlotInit.SlotActive = 0x00000000;
875   if (HAL_SAI_Init(&hsai_BlockA2) != HAL_OK)
876   {
877     Error_Handler();
878   }
879   hsai_BlockB2.Instance = SAI2_Block_B;
880   hsai_BlockB2.Init.Protocol = SAI_FREE_PROTOCOL;
881   hsai_BlockB2.Init.AudioMode = SAI_MODESLAVE_RX;
882   hsai_BlockB2.Init.DataSize = SAI_DATASIZE_8;
883   hsai_BlockB2.Init.FirstBit = SAI_FIRSTBIT_MSB;
884   hsai_BlockB2.Init.ClockStrobing =
       SAI_CLOCKSTROBING_FALLINGEDGE;
885   hsai_BlockB2.Init.Synchro = SAI_SYNCHRONOUS;
886   hsai_BlockB2.Init.OutputDrive = SAI_OUTPUTDRIVE_DISABLE;
887   hsai_BlockB2.Init.FIFOThreshold = SAI_FIFOTHRESHOLD_EMPTY;
888   hsai_BlockB2.Init.SynchroExt = SAI_SYNCEXT_DISABLE;
889   hsai_BlockB2.Init.MonoStereoMode = SAI_STEREOMODE;
890   hsai_BlockB2.Init.CompandingMode = SAI_NOCOMPANDING;
891   hsai_BlockB2.Init.TriState = SAI_OUTPUT_NOTRELEASED;
892   hsai_BlockB2.FrameInit.FrameLength = 8;
893   hsai_BlockB2.FrameInit.ActiveFrameLength = 1;
894   hsai_BlockB2.FrameInit.FSDefinition = SAI_FS_STARTFRAME;
895   hsai_BlockB2.FrameInit.FSPolarity = SAI_FS_ACTIVE_LOW;
896   hsai_BlockB2.FrameInit.FSOffset = SAI_FS_FIRSTBIT;
897   hsai_BlockB2.SlotInit.FirstBitOffset = 0;
898   hsai_BlockB2.SlotInit.SlotSize = SAI_SLOTSIZE_DATASIZE;
899   hsai_BlockB2.SlotInit.SlotNumber = 1;
900   hsai_BlockB2.SlotInit.SlotActive = 0x00000000;
901   if (HAL_SAI_Init(&hsai_BlockB2) != HAL_OK)
902   {
903     Error_Handler();
904   }
905   /* USER CODE BEGIN SAI2_Init 2 */
```

```
906
907   /* USER CODE END SAI2_Init 2 */
908
909 }
910
911 /**
912   * @brief SDMMC1 Initialization Function
913   * @param None
914   * @retval None
915   */
916 static void MX_SDMMC1_SD_Init(void)
917 {
918
919   /* USER CODE BEGIN SDMMC1_Init 0 */
920
921   /* USER CODE END SDMMC1_Init 0 */
922
923   /* USER CODE BEGIN SDMMC1_Init 1 */
924
925   /* USER CODE END SDMMC1_Init 1 */
926   hsd1.Instance = SDMMC1;
927   hsd1.Init.ClockEdge = SDMMC_CLOCK_EDGE_RISING;
928   hsd1.Init.ClockBypass = SDMMC_CLOCK_BYPASS_DISABLE;
929   hsd1.Init.ClockPowerSave = SDMMC_CLOCK_POWER_SAVE_DISABLE;
930   hsd1.Init.BusWide = SDMMC_BUS_WIDE_1B;
931   hsd1.Init.HardwareFlowControl =
       SDMMC_HARDWARE_FLOW_CONTROL_DISABLE;
932   hsd1.Init.ClockDiv = 0;
933   /* USER CODE BEGIN SDMMC1_Init 2 */
934
935   /* USER CODE END SDMMC1_Init 2 */
936
937 }
938
939 /**
940   * @brief SPDIFRX Initialization Function
941   * @param None
942   * @retval None
943   */
944 static void MX_SPDIFRX_Init(void)
945 {
946
947   /* USER CODE BEGIN SPDIFRX_Init 0 */
948
949   /* USER CODE END SPDIFRX_Init 0 */
950
```

```
951   /* USER CODE BEGIN SPDIFRX_Init 1 */
952
953   /* USER CODE END SPDIFRX_Init 1 */
954   hspdif.Instance = SPDIFRX;
955   hspdif.Init.InputSelection = SPDIFRX_INPUT_IN0;
956   hspdif.Init.Retries = SPDIFRX_MAXRETRIES_NONE;
957   hspdif.Init.WaitForActivity = SPDIFRX_WAITFORACTIVITY_OFF;
958   hspdif.Init.ChannelSelection = SPDIFRX_CHANNEL_A;
959   hspdif.Init.DataFormat = SPDIFRX_DATAFORMAT_LSB;
960   hspdif.Init.StereoMode = SPDIFRX_STEREOMODE_DISABLE;
961   hspdif.Init.PreambleTypeMask = SPDIFRX_PREAMBLETYPEMASK_OFF;
962   hspdif.Init.ChannelStatusMask = SPDIFRX_CHANNELSTATUS_OFF;
963   hspdif.Init.ValidityBitMask = SPDIFRX_VALIDITYMASK_OFF;
964   hspdif.Init.ParityErrorMask = SPDIFRX_PARITYERRORMASK_OFF;
965   if (HAL_SPDIFRX_Init(&hspdif) != HAL_OK)
966   {
967     Error_Handler();
968   }
969   /* USER CODE BEGIN SPDIFRX_Init 2 */
970
971   /* USER CODE END SPDIFRX_Init 2 */
972
973 }
974
975 /**
976   * @brief TIM1 Initialization Function
977   * @param None
978   * @retval None
979   */
980 static void MX_TIM1_Init(void)
981 {
982
983   /* USER CODE BEGIN TIM1_Init 0 */
984
985   /* USER CODE END TIM1_Init 0 */
986
987   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
988   TIM_MasterConfigTypeDef sMasterConfig = {0};
989   TIM_OC_InitTypeDef sConfigOC = {0};
990   TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
991
992   /* USER CODE BEGIN TIM1_Init 1 */
993
994   /* USER CODE END TIM1_Init 1 */
995   htim1.Instance = TIM1;
996   htim1.Init.Prescaler = 0;
```

```
997   htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
998   htim1.Init.Period = 65535;
999   htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1000  htim1.Init.RepetitionCounter = 0;
1001  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1002  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
1003  {
1004    Error_Handler();
1005  }
1006  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1007  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) !=
      HAL_OK)
1008  {
1009    Error_Handler();
1010  }
1011  if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
1012  {
1013    Error_Handler();
1014  }
1015  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
1016  sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
1017  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
1018  if (HAL_TIMEx_MasterConfigSynchronization(&htim1,
      &sMasterConfig) != HAL_OK)
1019  {
1020    Error_Handler();
1021  }
1022  sConfigOC.OCMode = TIM_OCMODE_PWM1;
1023  sConfigOC.Pulse = 0;
1024  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
1025  sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
1026  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
1027  sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
1028  sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
1029  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC,
      TIM_CHANNEL_1) != HAL_OK)
1030  {
1031    Error_Handler();
1032  }
1033  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
1034  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
1035  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
1036  sBreakDeadTimeConfig.DeadTime = 0;
1037  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
1038  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
1039  sBreakDeadTimeConfig.BreakFilter = 0;
```

```
1040    sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
1041    sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
1042    sBreakDeadTimeConfig.Break2Filter = 0;
1043    sBreakDeadTimeConfig.AutomaticOutput =
        TIM_AUTOMATICOUTPUT_DISABLE;
1044    if (HAL_TIMEx_ConfigBreakDeadTime(&htim1,
        &sBreakDeadTimeConfig) != HAL_OK)
1045    {
1046      Error_Handler();
1047    }
1048    /* USER CODE BEGIN TIM1_Init 2 */
1049
1050    /* USER CODE END TIM1_Init 2 */
1051    HAL_TIM_MspPostInit(&htim1);
1052
1053  }
1054
1055  /**
1056    * @brief TIM2 Initialization Function
1057    * @param None
1058    * @retval None
1059    */
1060  static void MX_TIM2_Init(void)
1061  {
1062
1063    /* USER CODE BEGIN TIM2_Init 0 */
1064
1065    /* USER CODE END TIM2_Init 0 */
1066
1067    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
1068    TIM_MasterConfigTypeDef sMasterConfig = {0};
1069    TIM_OC_InitTypeDef sConfigOC = {0};
1070
1071    /* USER CODE BEGIN TIM2_Init 1 */
1072
1073    /* USER CODE END TIM2_Init 1 */
1074    htim2.Instance = TIM2;
1075    htim2.Init.Prescaler = 0;
1076    htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
1077    htim2.Init.Period = 4294967295;
1078    htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1079    htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1080    if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
1081    {
1082      Error_Handler();
1083    }
```

```
1084    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1085    if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) !=
         HAL_OK)
1086    {
1087      Error_Handler();
1088    }
1089    if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
1090    {
1091      Error_Handler();
1092    }
1093    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
1094    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
1095    if (HAL_TIMEx_MasterConfigSynchronization(&htim2,
         &sMasterConfig) != HAL_OK)
1096    {
1097      Error_Handler();
1098    }
1099    sConfigOC.OCMode = TIM_OCMODE_PWM1;
1100    sConfigOC.Pulse = 0;
1101    sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
1102    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
1103    if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
         TIM_CHANNEL_1) != HAL_OK)
1104    {
1105      Error_Handler();
1106    }
1107    /* USER CODE BEGIN TIM2_Init 2 */
1108
1109    /* USER CODE END TIM2_Init 2 */
1110    HAL_TIM_MspPostInit(&htim2);
1111
1112  }
1113
1114  /**
1115    * @brief TIM3 Initialization Function
1116    * @param None
1117    * @retval None
1118    */
1119  static void MX_TIM3_Init(void)
1120  {
1121
1122    /* USER CODE BEGIN TIM3_Init 0 */
1123
1124    /* USER CODE END TIM3_Init 0 */
1125
1126    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
```

```
1127  TIM_MasterConfigTypeDef sMasterConfig = {0};
1128  TIM_OC_InitTypeDef sConfigOC = {0};
1129
1130  /* USER CODE BEGIN TIM3_Init 1 */
1131
1132  /* USER CODE END TIM3_Init 1 */
1133  htim3.Instance = TIM3;
1134  htim3.Init.Prescaler = 0;
1135  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
1136  htim3.Init.Period = 65535;
1137  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1138  htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1139  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
1140  {
1141    Error_Handler();
1142  }
1143  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1144  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) !=
      HAL_OK)
1145  {
1146    Error_Handler();
1147  }
1148  if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
1149  {
1150    Error_Handler();
1151  }
1152  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
1153  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
1154  if (HAL_TIMEx_MasterConfigSynchronization(&htim3,
      &sMasterConfig) != HAL_OK)
1155  {
1156    Error_Handler();
1157  }
1158  sConfigOC.OCMode = TIM_OCMODE_PWM1;
1159  sConfigOC.Pulse = 0;
1160  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
1161  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
1162  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC,
      TIM_CHANNEL_1) != HAL_OK)
1163  {
1164    Error_Handler();
1165  }
1166  /* USER CODE BEGIN TIM3_Init 2 */
1167
1168  /* USER CODE END TIM3_Init 2 */
1169  HAL_TIM_MspPostInit(&htim3);
```

```
1170
1171 }
1172
1173 /**
1174   * @brief TIM5 Initialization Function
1175   * @param None
1176   * @retval None
1177   */
1178 static void MX_TIM5_Init(void)
1179 {
1180
1181   /* USER CODE BEGIN TIM5_Init 0 */
1182
1183   /* USER CODE END TIM5_Init 0 */
1184
1185   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
1186   TIM_MasterConfigTypeDef sMasterConfig = {0};
1187   TIM_OC_InitTypeDef sConfigOC = {0};
1188
1189   /* USER CODE BEGIN TIM5_Init 1 */
1190
1191   /* USER CODE END TIM5_Init 1 */
1192   htim5.Instance = TIM5;
1193   htim5.Init.Prescaler = 0;
1194   htim5.Init.CounterMode = TIM_COUNTERMODE_UP;
1195   htim5.Init.Period = 4294967295;
1196   htim5.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1197   htim5.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1198   if (HAL_TIM_Base_Init(&htim5) != HAL_OK)
1199   {
1200     Error_Handler();
1201   }
1202   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1203   if (HAL_TIM_ConfigClockSource(&htim5, &sClockSourceConfig) !=
      HAL_OK)
1204   {
1205     Error_Handler();
1206   }
1207   if (HAL_TIM_PWM_Init(&htim5) != HAL_OK)
1208   {
1209     Error_Handler();
1210   }
1211   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
1212   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
1213   if (HAL_TIMEx_MasterConfigSynchronization(&htim5,
      &sMasterConfig) != HAL_OK)
```

```
1214    {
1215      Error_Handler();
1216    }
1217    sConfigOC.OCMode = TIM_OCMODE_PWM1;
1218    sConfigOC.Pulse = 0;
1219    sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
1220    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
1221    if (HAL_TIM_PWM_ConfigChannel(&htim5, &sConfigOC,
        TIM_CHANNEL_4) != HAL_OK)
1222    {
1223      Error_Handler();
1224    }
1225    /* USER CODE BEGIN TIM5_Init 2 */
1226
1227    /* USER CODE END TIM5_Init 2 */
1228    HAL_TIM_MspPostInit(&htim5);
1229
1230  }
1231
1232  /**
1233    * @brief TIM8 Initialization Function
1234    * @param None
1235    * @retval None
1236    */
1237  static void MX_TIM8_Init(void)
1238  {
1239
1240    /* USER CODE BEGIN TIM8_Init 0 */
1241
1242    /* USER CODE END TIM8_Init 0 */
1243
1244    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
1245    TIM_MasterConfigTypeDef sMasterConfig = {0};
1246
1247    /* USER CODE BEGIN TIM8_Init 1 */
1248
1249    /* USER CODE END TIM8_Init 1 */
1250    htim8.Instance = TIM8;
1251    htim8.Init.Prescaler = 0;
1252    htim8.Init.CounterMode = TIM_COUNTERMODE_UP;
1253    htim8.Init.Period = 65535;
1254    htim8.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1255    htim8.Init.RepetitionCounter = 0;
1256    htim8.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1257    if (HAL_TIM_Base_Init(&htim8) != HAL_OK)
1258    {
```

```
1259      Error_Handler();
1260    }
1261    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1262    if (HAL_TIM_ConfigClockSource(&htim8, &sClockSourceConfig) !=
        HAL_OK)
1263    {
1264      Error_Handler();
1265    }
1266    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
1267    sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
1268    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
1269    if (HAL_TIMEx_MasterConfigSynchronization(&htim8,
        &sMasterConfig) != HAL_OK)
1270    {
1271      Error_Handler();
1272    }
1273    /* USER CODE BEGIN TIM8_Init 2 */
1274
1275    /* USER CODE END TIM8_Init 2 */
1276
1277  }
1278
1279  /**
1280    * @brief TIM12 Initialization Function
1281    * @param None
1282    * @retval None
1283    */
1284  static void MX_TIM12_Init(void)
1285  {
1286
1287    /* USER CODE BEGIN TIM12_Init 0 */
1288
1289    /* USER CODE END TIM12_Init 0 */
1290
1291    TIM_OC_InitTypeDef sConfigOC = {0};
1292
1293    /* USER CODE BEGIN TIM12_Init 1 */
1294
1295    /* USER CODE END TIM12_Init 1 */
1296    htim12.Instance = TIM12;
1297    htim12.Init.Prescaler = 0;
1298    htim12.Init.CounterMode = TIM_COUNTERMODE_UP;
1299    htim12.Init.Period = 65535;
1300    htim12.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1301    htim12.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1302    if (HAL_TIM_PWM_Init(&htim12) != HAL_OK)
```

```
1303    {
1304      Error_Handler();
1305    }
1306    sConfigOC.OCMode = TIM_OCMODE_PWM1;
1307    sConfigOC.Pulse = 0;
1308    sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
1309    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
1310    if (HAL_TIM_PWM_ConfigChannel(&htim12, &sConfigOC,
        TIM_CHANNEL_1) != HAL_OK)
1311    {
1312      Error_Handler();
1313    }
1314    /* USER CODE BEGIN TIM12_Init 2 */
1315
1316    /* USER CODE END TIM12_Init 2 */
1317    HAL_TIM_MspPostInit(&htim12);
1318
1319  }
1320
1321  /**
1322    * @brief USART1 Initialization Function
1323    * @param None
1324    * @retval None
1325    */
1326  static void MX_USART1_UART_Init(void)
1327  {
1328
1329    /* USER CODE BEGIN USART1_Init 0 */
1330
1331    /* USER CODE END USART1_Init 0 */
1332
1333    /* USER CODE BEGIN USART1_Init 1 */
1334
1335    /* USER CODE END USART1_Init 1 */
1336    huart1.Instance = USART1;
1337    huart1.Init.BaudRate = 115200;
1338    huart1.Init.WordLength = UART_WORDLENGTH_8B;
1339    huart1.Init.StopBits = UART_STOPBITS_1;
1340    huart1.Init.Parity = UART_PARITY_NONE;
1341    huart1.Init.Mode = UART_MODE_TX_RX;
1342    huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
1343    huart1.Init.OverSampling = UART_OVERSAMPLING_16;
1344    huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
1345    huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
1346    if (HAL_UART_Init(&huart1) != HAL_OK)
1347    {
```

```
1348      Error_Handler();
1349    }
1350    /* USER CODE BEGIN USART1_Init 2 */
1351
1352    /* USER CODE END USART1_Init 2 */
1353
1354  }
1355
1356  /**
1357    * @brief USART6 Initialization Function
1358    * @param None
1359    * @retval None
1360    */
1361  static void MX_USART6_UART_Init(void)
1362  {
1363
1364    /* USER CODE BEGIN USART6_Init 0 */
1365
1366    /* USER CODE END USART6_Init 0 */
1367
1368    /* USER CODE BEGIN USART6_Init 1 */
1369
1370    /* USER CODE END USART6_Init 1 */
1371    huart6.Instance = USART6;
1372    huart6.Init.BaudRate = 115200;
1373    huart6.Init.WordLength = UART_WORDLENGTH_8B;
1374    huart6.Init.StopBits = UART_STOPBITS_1;
1375    huart6.Init.Parity = UART_PARITY_NONE;
1376    huart6.Init.Mode = UART_MODE_TX_RX;
1377    huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
1378    huart6.Init.OverSampling = UART_OVERSAMPLING_16;
1379    huart6.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
1380    huart6.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
1381    if (HAL_UART_Init(&huart6) != HAL_OK)
1382    {
1383      Error_Handler();
1384    }
1385    /* USER CODE BEGIN USART6_Init 2 */
1386
1387    /* USER CODE END USART6_Init 2 */
1388
1389  }
1390
1391  /**
1392    * Enable DMA controller clock
1393    */
```

```
1394 static void MX_DMA_Init(void)
1395 {
1396
1397   /* DMA controller clock enable */
1398   __HAL_RCC_DMA2_CLK_ENABLE();
1399
1400   /* DMA interrupt init */
1401   /* DMA2_Stream3_IRQn interrupt configuration */
1402   HAL_NVIC_SetPriority(DMA2_Stream3_IRQn, 5, 0);
1403   HAL_NVIC_EnableIRQ(DMA2_Stream3_IRQn);
1404   /* DMA2_Stream4_IRQn interrupt configuration */
1405   HAL_NVIC_SetPriority(DMA2_Stream4_IRQn, 5, 0);
1406   HAL_NVIC_EnableIRQ(DMA2_Stream4_IRQn);
1407   /* DMA2_Stream6_IRQn interrupt configuration */
1408   HAL_NVIC_SetPriority(DMA2_Stream6_IRQn, 5, 0);
1409   HAL_NVIC_EnableIRQ(DMA2_Stream6_IRQn);
1410   /* DMA2_Stream7_IRQn interrupt configuration */
1411   HAL_NVIC_SetPriority(DMA2_Stream7_IRQn, 5, 0);
1412   HAL_NVIC_EnableIRQ(DMA2_Stream7_IRQn);
1413
1414 }
1415
1416 /* FMC initialization function */
1417 static void MX_FMC_Init(void)
1418 {
1419
1420   /* USER CODE BEGIN FMC_Init 0 */
1421
1422   /* USER CODE END FMC_Init 0 */
1423
1424   FMC_SDRAM_TimingTypeDef SdramTiming = {0};
1425
1426   /* USER CODE BEGIN FMC_Init 1 */
1427
1428   /* USER CODE END FMC_Init 1 */
1429
1430   /** Perform the SDRAM1 memory initialization sequence
1431   */
1432   hsdram1.Instance = FMC_SDRAM_DEVICE;
1433   /* hsdram1.Init */
1434   hsdram1.Init.SDBank = FMC_SDRAM_BANK1;
1435   hsdram1.Init.ColumnBitsNumber = FMC_SDRAM_COLUMN_BITS_NUM_8;
1436   hsdram1.Init.RowBitsNumber = FMC_SDRAM_ROW_BITS_NUM_12;
1437   hsdram1.Init.MemoryDataWidth = FMC_SDRAM_MEM_BUS_WIDTH_16;
1438   hsdram1.Init.InternalBankNumber = FMC_SDRAM_INTERN_BANKS_NUM_4;
1439   hsdram1.Init.CASLatency = FMC_SDRAM_CAS_LATENCY_3;
```

```
1440    hsdram1.Init.WriteProtection =
         FMC_SDRAM_WRITE_PROTECTION_DISABLE;
1441    hsdram1.Init.SDClockPeriod = FMC_SDRAM_CLOCK_PERIOD_2;
1442    hsdram1.Init.ReadBurst = FMC_SDRAM_RBURST_ENABLE;
1443    hsdram1.Init.ReadPipeDelay = FMC_SDRAM_RPIPE_DELAY_0;
1444    /* SdramTiming */
1445    SdramTiming.LoadToActiveDelay = 2;
1446    SdramTiming.ExitSelfRefreshDelay = 7;
1447    SdramTiming.SelfRefreshTime = 4;
1448    SdramTiming.RowCycleDelay = 7;
1449    SdramTiming.WriteRecoveryTime = 3;
1450    SdramTiming.RPDelay = 2;
1451    SdramTiming.RCDDelay = 2;
1452
1453    if (HAL_SDRAM_Init(&hsdram1, &SdramTiming) != HAL_OK)
1454    {
1455      Error_Handler( );
1456    }
1457
1458    /* USER CODE BEGIN FMC_Init 2 */
1459
1460    /* USER CODE END FMC_Init 2 */
1461  }
1462
1463  /**
1464    * @brief GPIO Initialization Function
1465    * @param None
1466    * @retval None
1467    */
1468  static void MX_GPIO_Init(void)
1469  {
1470    GPIO_InitTypeDef GPIO_InitStruct = {0};
1471
1472    /* GPIO Ports Clock Enable */
1473    __HAL_RCC_GPIOE_CLK_ENABLE();
1474    __HAL_RCC_GPIOG_CLK_ENABLE();
1475    __HAL_RCC_GPIOB_CLK_ENABLE();
1476    __HAL_RCC_GPIOD_CLK_ENABLE();
1477    __HAL_RCC_GPIOC_CLK_ENABLE();
1478    __HAL_RCC_GPIOA_CLK_ENABLE();
1479    __HAL_RCC_GPIOJ_CLK_ENABLE();
1480    __HAL_RCC_GPIOI_CLK_ENABLE();
1481    __HAL_RCC_GPIOK_CLK_ENABLE();
1482    __HAL_RCC_GPIOF_CLK_ENABLE();
1483    __HAL_RCC_GPIOH_CLK_ENABLE();
1484
```

```
1485    /*Configure GPIO pin Output Level */
1486    HAL_GPIO_WritePin(OTG_FS_PowerSwitchOn_GPIO_Port,
         OTG_FS_PowerSwitchOn_Pin, GPIO_PIN_SET);
1487
1488    /*Configure GPIO pin Output Level */
1489    HAL_GPIO_WritePin(GPIOI,
         ARDUINO_D7_Pin|ARDUINO_D8_Pin|GPIO_PIN_1, GPIO_PIN_RESET);
1490
1491    /*Configure GPIO pin Output Level */
1492    HAL_GPIO_WritePin(LCD_BL_CTRL_GPIO_Port, LCD_BL_CTRL_Pin,
         GPIO_PIN_SET);
1493
1494    /*Configure GPIO pin Output Level */
1495    HAL_GPIO_WritePin(LCD_DISP_GPIO_Port, LCD_DISP_Pin,
         GPIO_PIN_SET);
1496
1497    /*Configure GPIO pin Output Level */
1498    HAL_GPIO_WritePin(DCMI_PWR_EN_GPIO_Port, DCMI_PWR_EN_Pin,
         GPIO_PIN_RESET);
1499
1500    /*Configure GPIO pin Output Level */
1501    HAL_GPIO_WritePin(GPIOG,
         ARDUINO_D4_Pin|ARDUINO_D2_Pin|EXT_RST_Pin, GPIO_PIN_RESET);
1502
1503    /*Configure GPIO pin : OTG_HS_OverCurrent_Pin */
1504    GPIO_InitStruct.Pin = OTG_HS_OverCurrent_Pin;
1505    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1506    GPIO_InitStruct.Pull = GPIO_NOPULL;
1507    HAL_GPIO_Init(OTG_HS_OverCurrent_GPIO_Port, &GPIO_InitStruct);
1508
1509    /*Configure GPIO pins : ULPI_D7_Pin ULPI_D6_Pin ULPI_D5_Pin
         ULPI_D3_Pin
1510                             ULPI_D2_Pin ULPI_D1_Pin ULPI_D4_Pin */
1511    GPIO_InitStruct.Pin =
         ULPI_D7_Pin|ULPI_D6_Pin|ULPI_D5_Pin|ULPI_D3_Pin
1512                            |ULPI_D2_Pin|ULPI_D1_Pin|ULPI_D4_Pin;
1513    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
1514    GPIO_InitStruct.Pull = GPIO_NOPULL;
1515    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
1516    GPIO_InitStruct.Alternate = GPIO_AF10_OTG_HS;
1517    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
1518
1519    /*Configure GPIO pin : OTG_FS_VBUS_Pin */
1520    GPIO_InitStruct.Pin = OTG_FS_VBUS_Pin;
1521    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1522    GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
1523  HAL_GPIO_Init(OTG_FS_VBUS_GPIO_Port, &GPIO_InitStruct);
1524
1525  /*Configure GPIO pin : Audio_INT_Pin */
1526  GPIO_InitStruct.Pin = Audio_INT_Pin;
1527  GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
1528  GPIO_InitStruct.Pull = GPIO_NOPULL;
1529  HAL_GPIO_Init(Audio_INT_GPIO_Port, &GPIO_InitStruct);
1530
1531  /*Configure GPIO pin : OTG_FS_PowerSwitchOn_Pin */
1532  GPIO_InitStruct.Pin = OTG_FS_PowerSwitchOn_Pin;
1533  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1534  GPIO_InitStruct.Pull = GPIO_NOPULL;
1535  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1536  HAL_GPIO_Init(OTG_FS_PowerSwitchOn_GPIO_Port,
       &GPIO_InitStruct);
1537
1538  /*Configure GPIO pins : ARDUINO_D7_Pin ARDUINO_D8_Pin PI1
       LCD_DISP_Pin */
1539  GPIO_InitStruct.Pin =
       ARDUINO_D7_Pin|ARDUINO_D8_Pin|GPIO_PIN_1|LCD_DISP_Pin;
1540  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1541  GPIO_InitStruct.Pull = GPIO_NOPULL;
1542  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1543  HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);
1544
1545  /*Configure GPIO pin : uSD_Detect_Pin */
1546  GPIO_InitStruct.Pin = uSD_Detect_Pin;
1547  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1548  GPIO_InitStruct.Pull = GPIO_NOPULL;
1549  HAL_GPIO_Init(uSD_Detect_GPIO_Port, &GPIO_InitStruct);
1550
1551  /*Configure GPIO pin : LCD_BL_CTRL_Pin */
1552  GPIO_InitStruct.Pin = LCD_BL_CTRL_Pin;
1553  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1554  GPIO_InitStruct.Pull = GPIO_NOPULL;
1555  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1556  HAL_GPIO_Init(LCD_BL_CTRL_GPIO_Port, &GPIO_InitStruct);
1557
1558  /*Configure GPIO pin : OTG_FS_OverCurrent_Pin */
1559  GPIO_InitStruct.Pin = OTG_FS_OverCurrent_Pin;
1560  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1561  GPIO_InitStruct.Pull = GPIO_NOPULL;
1562  HAL_GPIO_Init(OTG_FS_OverCurrent_GPIO_Port, &GPIO_InitStruct);
1563
1564  /*Configure GPIO pins : TP3_Pin NC2_Pin */
1565  GPIO_InitStruct.Pin = TP3_Pin|NC2_Pin;
```

```
1566  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1567  GPIO_InitStruct.Pull = GPIO_NOPULL;
1568  HAL_GPIO_Init(GPIOH, &GPIO_InitStruct);
1569
1570  /*Configure GPIO pin : DCMI_PWR_EN_Pin */
1571  GPIO_InitStruct.Pin = DCMI_PWR_EN_Pin;
1572  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1573  GPIO_InitStruct.Pull = GPIO_NOPULL;
1574  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1575  HAL_GPIO_Init(DCMI_PWR_EN_GPIO_Port, &GPIO_InitStruct);
1576
1577  /*Configure GPIO pin : LCD_INT_Pin */
1578  GPIO_InitStruct.Pin = LCD_INT_Pin;
1579  GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
1580  GPIO_InitStruct.Pull = GPIO_NOPULL;
1581  HAL_GPIO_Init(LCD_INT_GPIO_Port, &GPIO_InitStruct);
1582
1583  /*Configure GPIO pin : ULPI_NXT_Pin */
1584  GPIO_InitStruct.Pin = ULPI_NXT_Pin;
1585  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
1586  GPIO_InitStruct.Pull = GPIO_NOPULL;
1587  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
1588  GPIO_InitStruct.Alternate = GPIO_AF10_OTG_HS;
1589  HAL_GPIO_Init(ULPI_NXT_GPIO_Port, &GPIO_InitStruct);
1590
1591  /*Configure GPIO pins : ARDUINO_D4_Pin ARDUINO_D2_Pin
       EXT_RST_Pin */
1592  GPIO_InitStruct.Pin =
       ARDUINO_D4_Pin|ARDUINO_D2_Pin|EXT_RST_Pin;
1593  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
1594  GPIO_InitStruct.Pull = GPIO_NOPULL;
1595  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1596  HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
1597
1598  /*Configure GPIO pins : ULPI_STP_Pin ULPI_DIR_Pin */
1599  GPIO_InitStruct.Pin = ULPI_STP_Pin|ULPI_DIR_Pin;
1600  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
1601  GPIO_InitStruct.Pull = GPIO_NOPULL;
1602  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
1603  GPIO_InitStruct.Alternate = GPIO_AF10_OTG_HS;
1604  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
1605
1606  /*Configure GPIO pin : RMII_RXER_Pin */
1607  GPIO_InitStruct.Pin = RMII_RXER_Pin;
1608  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
1609  GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
1610    HAL_GPIO_Init(RMII_RXER_GPIO_Port, &GPIO_InitStruct);
1611
1612    /*Configure GPIO pins : ULPI_CLK_Pin ULPI_D0_Pin */
1613    GPIO_InitStruct.Pin = ULPI_CLK_Pin|ULPI_D0_Pin;
1614    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
1615    GPIO_InitStruct.Pull = GPIO_NOPULL;
1616    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
1617    GPIO_InitStruct.Alternate = GPIO_AF10_OTG_HS;
1618    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
1619
1620    /*Configure GPIO pins : ARDUINO_MISO_D12_Pin
      ARDUINO_MOSI_PWM_D11_Pin */
1621    GPIO_InitStruct.Pin =
      ARDUINO_MISO_D12_Pin|ARDUINO_MOSI_PWM_D11_Pin;
1622    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
1623    GPIO_InitStruct.Pull = GPIO_NOPULL;
1624    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
1625    GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
1626    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
1627
1628  }
1629
1630  /* USER CODE BEGIN 4 */
1631
1632  /* USER CODE END 4 */
1633
1634  /* USER CODE BEGIN Header_StartDefaultTask */
1635  /**
1636    * @brief  Function implementing the defaultTask thread.
1637    * @param  argument: Not used
1638    * @retval None
1639    */
1640  /* USER CODE END Header_StartDefaultTask */
1641  void StartDefaultTask(void const * argument)
1642  {
1643    /* init code for USB_HOST */
1644    MX_USB_HOST_Init();
1645    /* USER CODE BEGIN 5 */
1646
1647    //qq
1648    /* Infinite loop */
1649
1650    roleInit();
1651    for(;;)
1652    {
1653        roleNode();
```

```
1654      osDelay(1);
1655    }
1656    /* USER CODE END 5 */
1657  }
1658
1659  /**
1660    * @brief  Period elapsed callback in non blocking mode
1661    * @note   This function is called  when TIM6 interrupt took
       place, inside
1662    * HAL_TIM_IRQHandler(). It makes a direct call to
       HAL_IncTick() to increment
1663    * a global variable "uwTick" used as application time base.
1664    * @param  htim : TIM handle
1665    * @retval None
1666    */
1667  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
1668  {
1669    /* USER CODE BEGIN Callback 0 */
1670
1671    /* USER CODE END Callback 0 */
1672    if (htim->Instance == TIM6) {
1673      HAL_IncTick();
1674    }
1675    /* USER CODE BEGIN Callback 1 */
1676
1677    /* USER CODE END Callback 1 */
1678  }
1679
1680  /**
1681    * @brief  This function is executed in case of error
       occurrence.
1682    * @retval None
1683    */
1684  void Error_Handler(void)
1685  {
1686    /* USER CODE BEGIN Error_Handler_Debug */
1687    /* User can add his own implementation to report the HAL error
       return state */
1688    __disable_irq();
1689    while (1)
1690    {
1691    }
1692    /* USER CODE END Error_Handler_Debug */
1693  }
1694
1695  #ifdef  USE_FULL_ASSERT
```

```
1696 /**
1697   * @brief  Reports the name of the source file and the source
       line number
1698   *         where the assert_param error has occurred.
1699   * @param  file: pointer to the source file name
1700   * @param  line: assert_param error line source number
1701   * @retval None
1702   */
1703 void assert_failed(uint8_t *file, uint32_t line)
1704 {
1705   /* USER CODE BEGIN 6 */
1706   /* User can add his own implementation to report the file name
       and line number,
1707      ex: printf("Wrong parameters value: file %s on line
       %d\r\n", file, line) */
1708   /* USER CODE END 6 */
1709 }
1710 #endif /* USE_FULL_ASSERT */
```

Code Listing 7: main.c

7.5 role

The `role.c` module coordinates recording and inference on the STM32F746 system. `roleInit()` mounts the SD card and initializes the serial interface and LCD. `roleNode()` implements a simple state machine that starts recording (`recordStart()`), polls `recordProcess()` until the recording completes, and then stops recording (`recordStop()`). After stopping, `ai_on()` is called to perform inference. `ai_on()` reads a 512-sample frame (from SDRAM if a temporary buffer was used or from the WAV file on SD), computes 40-dimensional log-mel features (and optionally 13 MFCCs for debugging), quantizes the features to the model's int8 input format using the model scale and zero-point, initializes the AI network lazily, runs the neural network, dequantizes the outputs, finds the predicted index, and then displays and logs the prediction. The audio stream is 16-bit PCM at `DEFAULT_AUDIO_IN_FREQ` (typically 48 kHz), and data is delivered to the MCU by DMA from the WM8994 via SAI; recorded data is saved to SD or temporarily cached in SDRAM and flushed to SD when available.

8 Dataset Collection

The dataset collection process was a fundamental step in developing a robust and accurate speaker identification system. Voice recordings were collected from all five members of the project group, where each participant provided approximately **10 minutes** of continuous speech. The recordings were captured using mobile phones in a quiet indoor environment to ensure minimal background noise and acceptable audio clarity.After the recordings were completed, all audio files were transferred to a computer for further processing. The recordings were converted into standard **WAV** format using Python scripts, ensuring uniformity in sampling rate, bit depth, and overall audio quality across all samples.To expand the dataset and improve the performance of the machine learning model, each 10-minute audio file was segmented into multiple **1-second** audio clips. This segmentation significantly increased the number of training samples while preserving the unique vocal characteristics of each speaker.After segmentation, a total of **2595** individual audio samples were generated. These samples were then organized into five separate folders, with each folder corresponding to one specific speaker. This structured dataset organization simplified subsequent processes such as labeling, MFCC feature extraction, and model training.The collected and well-organized dataset provided a solid foundation for developing a reliable machine-learning model capable of performing real-time speaker identification on the STM32F746G-Discovery platform.

The dataset structure:

```
dataset/
    Fa_him/clip001-610.wav
    imran/clip001-601.wav
    nayeem/clip001-61.wav
    shahed/clip001-720.wav
    talukder/clip001-603.wav
```

8.1 code

Split_audio.py:

```
pip install pydub
from pydub import AudioSegment
import math
import os
```

```
import time
import platform
import psutil

# file path
input_file = r"D:\DSP Project WAV File\DSP Project WAV File\fa him.wav"

# Folder to save 1-second clips
output_folder = r"D:\DSP Project WAV File\DSP Project WAV File\fa_him"

chunk_length_ms = 1 * 1000   # 1 second

# Create output folder
os.makedirs(output_folder, exist_ok=True)

# Load audio
audio = AudioSegment.from_file(input_file)
audio_length_ms = len(audio)

# Number of chunks
num_chunks = math.ceil(audio_length_ms / chunk_length_ms)
print(f"Total chunks: {num_chunks}")

# Split and export
for i in range(num_chunks):
    start = i * chunk_length_ms
    end = min(start + chunk_length_ms, audio_length_ms)
    chunk = audio[start:end]

    filename = os.path.join(output_folder, f"clip_{i+1:03}.wav")
    chunk.export(filename, format="wav")
    print("Saved:", filename)

print("Done.")

# System Info
start = time.time()
print("=== System Information ===")
print(f"OS: {platform.system()} {platform.release()}")
print(f"Machine: {platform.machine()}")
print(f"Processor: {platform.processor()}")
print(f"CPU cores (logical): {psutil.cpu_count(logical=True)}")
mem = psutil.virtual_memory()
print(f"Total RAM: {mem.total / (1024 ** 3):.2f} GB")
```

9 MFCC Feature Extraction

MFCC is used because it closely represents human auditory perception. The MFCC pipeline includes:

1. Pre-emphasis

2. Framing
3. Hamming Window
4. FFT
5. Mel Filter Banks
6. Log Compression
7. DCT to compute coefficients

Mathematically:

$$\text{MFCC}(n) = \sum_{k=1}^{K} \log(E_k) \cos\left[n\left(k - \frac{1}{2}\right)\frac{\pi}{K}\right]$$

10 Model Training

A neural network classifier was trained using Python:

- Input: 13 MFCC features
- Hidden Layers: Dense(64), Dense(32)
- Output: Softmax for speaker classes
- Accuracy achieved: 9396%

The model was converted to TensorFlow Lite Micro:

`xx_model_data.cc`

11 Step-by-Step Description of the Implemented Code

This section explains the complete workflow of the implemented audio classification system. The `audio_classcification_with_own_dataset.py` file processes raw audio, extracts features, constructs a dataset, trains a machine learning model, and evaluates its performance. The major steps are described below.

11.1 Environment Setup

11.2 code

Install dependencies (run once in the environment):

```
!pip install librosa
!pip install scipy
!pip install resampy
```

11.3 Reading Audio Files

At the initial stage, we explored the required Python libraries and analyzed the fundamental properties of the recorded audio files, including sample rate, duration, and signal quality. To better understand the characteristics of the audio data, we visualized the signals using various graphical plots. These visualizations helped in observing the structure and behavior of the audio, which was essential for subsequent feature extraction and model training.

```
import matplotlib.pyplot as plt
%matplotlib inline
import IPython.display as ipd
import librosa
import librosa.display
#  to see the wave file nature and check the lib work or not
filename=r'D:\DSP Project WAV File\DSP Project WAV File\dataset\fa_him\
    clip_006.wav'
data, sample_rate = librosa.load(filename, sr=None)     # Load the audio
librosa.display.waveshow(data, sr=sample_rate)          # Show the waveform
ipd.display(ipd.Audio(data=data, rate=sample_rate))     # Play the sound
plt.show()                  # Show the waveform plot
```

We observed that the sample rate of our audio recordings was 48,000 Hz. Each audio file was represented as an array containing numerous floating-point values. Using these arrays, we prepared a dataset consisting of 2,595 data samples, which we organized into a CSV file. Since we had 5 speakers, the CSV file contained a total of $2,595 \times 5$ entries.

From this dataset, we extracted and stored the following information for each audio file:

- File path
- Class name (speaker identity)
- Duration

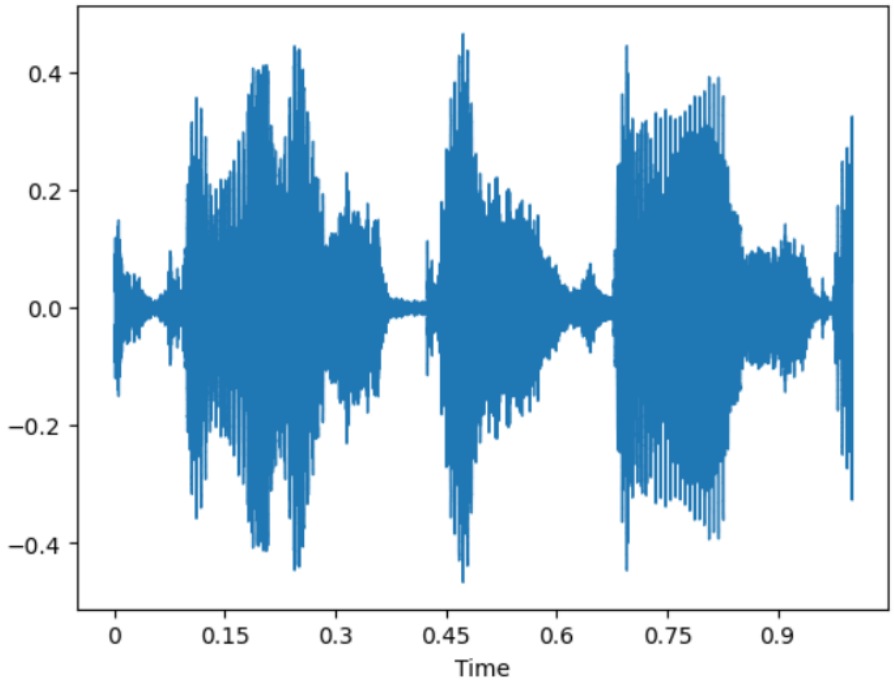

Figure 5: Waveform of the audio signal

- Sample rate
- Number of channels

Make CSV file to hold the key parameter:

```
import os
import csv
import wave
import contextlib

# ---- My DATASET PATH ----
DATASET_ROOT = r"D:\DSP Project WAV File\DSP Project WAV File\dataset"
OUTPUT_CSV = r"D:\DSP Project WAV File\DSP Project WAV File\audio_metadata.csv
    "


def get_wav_info(path):
    try:
        with contextlib.closing(wave.open(path, 'rb')) as wf:
            frames = wf.getnframes()
            rate = wf.getframerate()
            channels = wf.getnchannels()
            duration = frames / float(rate) if rate > 0 else 0.0
            return duration, rate, channels
    except:
        return None, None, None
```

```
21
22 def is_wav(filename):
23     return filename.lower().endswith(".wav")
24
25 def make_csv(dataset_root, output_csv):
26     rows = []
27     dataset_root = os.path.abspath(dataset_root)
28
29     for root, dirs, files in os.walk(dataset_root):
30         rel_path = os.path.relpath(root, dataset_root)
31
32         if rel_path == ".":
33             class_name = ""
34         else:
35             class_name = rel_path.split(os.sep)[0]
36
37         for file in files:
38             if is_wav(file):
39                 full_path = os.path.join(root, file)
40
41                 filesize = os.path.getsize(full_path)
42                 duration, rate, channels = get_wav_info(full_path)
43
44                 rows.append({
45                     "filepath": os.path.relpath(full_path, dataset_root),
46                     "class_name": class_name,
47                     "duration_sec": round(duration, 3) if duration else "",
48                     "sample_rate": rate if rate else "",
49                     "channels": channels if channels else "",
50                     "filesize_bytes": filesize
51                 })
52
53     header = ["filepath", "class_name", "duration_sec", "sample_rate", "
    channels", "filesize_bytes"]
54
55     with open(output_csv, "w", newline="", encoding="utf-8") as f:
56         writer = csv.DictWriter(f, fieldnames=header)
57         writer.writeheader()
58         for row in rows:
59             writer.writerow(row)
60
61     print(f"CSV created: {output_csv}")
62     print(f"Total WAV files: {len(rows)}")
63
64
65 if __name__ == "__main__":
66     make_csv(DATASET_ROOT, OUTPUT_CSV)
67 "D:\DSP Project WAV File\DSP Project WAV File\audio_metadata.csv"
```

11.4 Feature Extraction

- For each audio clip, Mel-Frequency Cepstral Coefficients (MFCCs) are extracted just write mfcc

```
mfccs = librosa.feature.mfcc(y=data, sr=sample_rate, n_mfcc=40)
print(mfccs.shape)
mfccs                  # Show the waveform plot

# output shows that
(40, 1)
array([[ 4.62669189e+02],
       [-9.98416710e+00],
       [ 3.05266762e+00],
       [ 6.83229923e-01],
       [-1.32615566e+00],
       [ 3.51012063e+00],
       [ 5.85476279e-01],
       [ 1.02078177e-01],
       [ 1.97663173e-01],
       [ 1.26941180e+00],
       [-9.09805477e-01],
       [-1.25338328e+00],
       [ 9.27704096e-01],
       [ 9.07290459e-01],
       [-1.31193972e+00],
       [ 2.56830841e-01],
       [ 1.04877457e-01],
       [-3.78011167e-01],
       [ 8.45407724e-01],
       [-1.04328680e+00],
       [ 4.67509747e-01],
       [-4.08623293e-02],
       [-4.67592627e-02],
       [-1.87125325e-01],
       [ 9.58730280e-02],
       [-7.67630711e-02],
       [ 1.98688954e-01],
       [-6.16146684e-01],
       [ 6.62834644e-01],
       [-5.87381244e-01],
       [ 4.02995557e-01],
       [-6.19672835e-01],
       [ 8.81810069e-01],
       [-1.09772825e+00],
       [ 7.63101637e-01],
       [-4.38766718e-01],
       [ 1.01637095e-01],
       [-1.42109290e-01],
       [ 2.31674284e-01],
       [-5.05474269e-01]], dtype=float32)
```

- Temporal variations are reduced by computing statistical features (e.g., mean

values of MFCC vectors).

- The extracted feature vectors are stored in CSV files such as `mfcc_dataset.csv`.

11.5 Dataset Preparation

The MFCC dataset is loaded into a Pandas dataframe. For each audio file, we extracted the Mel-Frequency Cepstral Coefficients (MFCCs) to represent its key features. Using these MFCCs, we created a separate CSV file containing only the audio class labels and the corresponding MFCC data. This processed dataset was then used to train our machine learning model.

```
import numpy as np
from tqdm import tqdm
import os
import pandas as pd
import librosa
audio_dataset_path = r"D:\DSP Project WAV File\DSP Project WAV File\
    dataset"                # Paths
metadata_csv = r"D:\DSP Project WAV File\DSP Project WAV File\
    audio_metadata.csv"
metadata = pd.read_csv(metadata_csv)              # Load metadata

def features_extractor(file_path):        # MFCC feature extraction
    function
    try:
        audio, sample_rate = librosa.load(file_path, res_type='
    kaiser_fast')
        mfccs_features = librosa.feature.mfcc(y=audio, sr=sample_rate,
    n_mfcc=40)
        mfccs_scaled_features = np.mean(mfccs_features.T, axis=0)
        return mfccs_scaled_features
    except Exception as e:
        print("Error processing:", file_path, "| Error:", e)
        return None

extracted_features = []     # Extract features

for index, row in tqdm(metadata.iterrows(), total=len(metadata)):

    file_name = os.path.join(audio_dataset_path, row["filepath"])
    # Build full path to audio file

    class_label = row["class_name"]                                  #
    Class label from CSV

    data = features_extractor(file_name)                            #
    Extract MFCC features

```

```
    if data is not None:
        extracted_features.append([data, class_label])
mfcc_df = pd.DataFrame(extracted_features, columns=["mfcc", "label"])
          # Convert to DataFrame
print(mfcc_df.head())
print("Total samples:", len(mfcc_df))
output_csv = r"D:\DSP Project WAV File\DSP Project WAV File\mfcc_dataset
   .csv"       # Save to CSV
mfcc_df.to_csv(output_csv, index=False)
print("MFCC feature extraction complete!")
```

- Features (X) and labels (y) are separated.
- Class labels are encoded into integers using a label encoder.
- The dataset is divided into training (80%) and testing (20%) sets using `train_test_split()` to ensure proper evaluation.

```
### Train Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state
   =0)
X_train
y
```

11.6 Model Construction

Feature scaling (e.g., standard normalization) is applied to improve model convergence.Feature vectors are reshaped if required by the chosen model architecture(e.g. Conv1D or dense networks).

- A neural network is created using the `Sequential` API of TensorFlow Keras.

```
#Dense()
model=Sequential()
# First layer with Input
model.add(Input(shape=(40,)))        # Input layer
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.20))

# Second layer
model.add(Dense(200, activation='relu'))
model.add(Dropout(0.20))

# Third layer
model.add(Dense(100, activation='relu'))
```

```
model.add(Dropout(0.20))

# Output layer
model.add(Dense(num_labels, activation='softmax'))

model.summary()
```

- Layers such as Dense, Dropout, or convolutional layers may be used depending on the architecture.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 100)	4,100
dropout (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 200)	20,200
dropout_1 (Dropout)	(None, 200)	0
dense_2 (Dense)	(None, 100)	20,100
dropout_2 (Dropout)	(None, 100)	0
dense_3 (Dense)	(None, 5)	505

Total params: 44,905 (175.41 KB)

Trainable params: 44,905 (175.41 KB)

Non-trainable params: 0 (0.00 B)

Figure 6: model summary

- The model is compiled with an appropriate loss function (e.g., sparse categorical cross-entropy), optimizer (e.g., Adam), and accuracy metrics.

11.7 Model Training

- The model is trained on the training data for a 100 number of epochs.
- Training and validation accuracy/loss are monitored. Which we get 0.9980732202529907 means 99.807%
- Matplotlib is used to visualize learning curves to detect overfitting.

Trianing my mode

```
model.compile(loss='categorical_crossentropy',metrics=['accuracy'],optimizer='
    adam')
from tensorflow.keras.callbacks import ModelCheckpoint
from datetime import datetime

```

```
num_epochs = 100
num_batch_size = 32

checkpointer = ModelCheckpoint(filepath='saved_models/audio_classification_100.keras',
                               verbose=1, save_best_only=True)
start = datetime.now()

model.fit(X_train, y_train, batch_size=num_batch_size, epochs=num_epochs, validation_data=(X_test, y_test), callbacks=[checkpointer], verbose=1)


duration = datetime.now() - start
print("Training completed in time: ", duration)
test_accuracy=model.evaluate(X_test,y_test,verbose=0)
print(test_accuracy[1])

from sklearn.preprocessing import LabelEncoder
import os

dataset_dir = r"D:\DSP Project WAV File\DSP Project WAV File\dataset"

class_names = sorted(os.listdir(dataset_dir))   # folder names
labelencoder = LabelEncoder()
labelencoder.fit(class_names)

print("Classes:", labelencoder.classes_)
```

11.8 Model Evaluation

- The trained model is evaluated on the test set to measure accuracy.
- Predictions are generated and compared with the actual labels.
- Additional evaluation tools such as confusion matrices and classification reports can be computed.

Testing model

```
import numpy as np

filename = r"D:\DSP Project WAV File\DSP Project WAV File\dataset\imran\clip_006.wav"


prediction_feature = features_extractor(filename)
prediction_feature = prediction_feature.reshape(1, -1)

# Predict the class probabilities
prediction = model.predict(prediction_feature)
```

```
11
12 # Get the index of the highest probability
13 predicted_class_index = np.argmax(prediction, axis=1)[0]
14
15 # Map index to label
16
17 print(f"Predicted sound class: {predicted_class_index}")
18 predicted_class = labelencoder.inverse_transform([predicted_class_index])[0]
19
20 print("Predicted class:", predicted_class)
```

11.9 Saving Model

STM32 microcontrollers, such as the STM32F746G, have limited memory and processing capabilities, which restrict the type and size of machine learning models that can be deployed. They natively support only lightweight formats like **TensorFlow Lite (.tflite)**, **Keras (.h5)**, or **C header (.h)** files containing model weights. Standard models are often too large to run efficiently, and models exceeding **200 kB** may not function properly on the STM32F746G.

Therefore, after training a model in Python using Keras or TensorFlow, it must be converted to **TensorFlow Lite** format and optimized using techniques such as **quantization** to reduce its size. The optimized TFLite model can then be converted into a C header file and deployed on the microcontroller, enabling real-time inference while staying within the hardware constraints.

Convert the model format

```
1  # Single-cell notebook-friendly converter + checker
2  # Paste and run in Jupyter. Edit MODEL_PATH and DATASET_DIR as needed.
3
4  import os
5  import numpy as np
6  import tensorflow as tf
7  import librosa
8  import warnings
9  warnings.filterwarnings("ignore")
10
11 # ========== EDIT THESE PATHS ==========
12 MODEL_PATH = r"D:\DSP Project WAV File\DSP Project WAV File\saved_models\
       audio_classification_100.keras"
13 DATASET_DIR = r"D:\DSP Project WAV File\DSP Project WAV File\dataset"  # point
       to folder with subfolders per person
14 OUT_TFLITE = "audio_class_quant.tflite"
15 # ======================================
16
17 # MFCC / preprocessing settings (must match training)
18 SR = 16000
19 TARGET_DURATION = 1.0
20 N_MFCC = 13
```

```
N_MELS = 40
N_FFT = 512
HOP_LENGTH = int(0.010 * SR)
WIN_LENGTH = int(0.025 * SR)
MAX_FRAMES = int(np.ceil((SR * TARGET_DURATION - WIN_LENGTH) / HOP_LENGTH)) + 1

def load_audio_fixed(path, sr=SR, target_duration=TARGET_DURATION):
    y, _ = librosa.load(path, sr=sr, mono=True)
    target_len = int(sr * target_duration)
    if len(y) < target_len:
        y = np.concatenate([y, np.zeros(target_len - len(y))])
    else:
        y = y[:target_len]
    return y

def compute_mfcc_for_rep(y, sr=SR, n_mfcc=N_MFCC, n_mels=N_MELS, n_fft=N_FFT, hop_length=HOP_LENGTH, win_length=WIN_LENGTH):
    # Pre-emphasis (if used in training)
    y = np.append(y[0], y[1:] - 0.97 * y[:-1])
    S = librosa.feature.melspectrogram(y=y, sr=sr, n_fft=n_fft, hop_length=hop_length,
                                       win_length=win_length, n_mels=n_mels, power=2.0)
    mfcc = librosa.feature.mfcc(S=librosa.power_to_db(S), sr=sr, n_mfcc=n_mfcc)
    mfcc = mfcc.T
    if mfcc.shape[0] < MAX_FRAMES:
        pad_width = MAX_FRAMES - mfcc.shape[0]
        mfcc = np.pad(mfcc, ((0, pad_width), (0, 0)), mode='constant')
    else:
        mfcc = mfcc[:MAX_FRAMES, :]
    return mfcc.astype(np.float32)

def collect_wavs(dataset_dir, max_files=50):
    wavs = []
    for root, _, files in os.walk(dataset_dir):
        for f in files:
            if f.lower().endswith(('.wav', '.flac', '.ogg', '.mp3', '.m4a')):
                wavs.append(os.path.join(root, f))
    wavs = sorted(wavs)
    if not wavs:
        raise FileNotFoundError(f"No audio files found under {dataset_dir}")
    return wavs[:max_files]

def representative_data_gen_from_folder(dataset_dir, max_samples=50):
    wavs = collect_wavs(dataset_dir, max_samples)
    def gen():
        for p in wavs:
            y = load_audio_fixed(p)
            mfcc = compute_mfcc_for_rep(y)           # shape: (frames, coeffs)
            inp = np.expand_dims(mfcc, axis=0).astype(np.float32)  # [1, frames, coeffs]
            inp = np.expand_dims(inp, axis=-1)  # [1, frames, coeffs, 1]
```

```
            yield [inp]
    return gen, wavs

def convert_keras_to_int8_tflite(keras_path, dataset_dir, out_tflite_path):
    print("Loading Keras model from:", keras_path)
    model = tf.keras.models.load_model(keras_path)
    print("Model loaded. Summary:")
    try:
        model.summary()
    except Exception:
        print("Couldn't print model summary (custom layers?). Continuing conversion...")
    converter = tf.lite.TFLiteConverter.from_keras_model(model)
    converter.optimizations = [tf.lite.Optimize.DEFAULT]
    rep_gen, wav_list = representative_data_gen_from_folder(dataset_dir, max_samples=50)
    converter.representative_dataset = rep_gen
    converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
    converter.inference_input_type = tf.int8
    converter.inference_output_type = tf.int8
    print(f"Using {len(wav_list)} representative audio files for calibration (first: {wav_list[0] if wav_list else 'N/A'})")
    tflite_model = converter.convert()
    with open(out_tflite_path, "wb") as f:
        f.write(tflite_model)
    print("Saved quantized tflite to:", out_tflite_path)
    return out_tflite_path

def print_tflite_size(path):
    s = os.path.getsize(path)
    print(f"\nTFLite file: {path}")
    print(f"Size: {s} bytes ({s/1024:.1f} KB, {s/1024/1024:.3f} MB)")
    return s

def estimate_tflite_peak_tensor_memory(path):
    interpreter = tf.lite.Interpreter(model_path=path)
    interpreter.allocate_tensors()
    tensor_details = interpreter.get_tensor_details()
    dtype_map = {
        np.dtype('int8'): 1, np.dtype('uint8'):1, np.dtype('int16'):2,
        np.dtype('int32'):4, np.dtype('float32'):4, np.dtype('float16'):2,
    }
    total_bytes = 0
    sizes = []
    for t in tensor_details:
        shape = t.get('shape', [])
        try:
            dtype = np.dtype(t.get('dtype').name)
        except Exception:
            dtype = np.dtype('float32')
        n = int(np.prod(shape)) if len(shape)>0 else 0
        bpe = dtype_map.get(dtype, 4)
        size = n * bpe
```

```
        sizes.append((t.get('name','<unk>'), size))
        total_bytes += size
    sizes_sorted = sorted(sizes, key=lambda x: x[1], reverse=True)
    print(f"\nInterpreter reports {len(tensor_details)} tensors.")
    print(f"Sum of all tensor buffers (rough): {total_bytes} bytes ({
    total_bytes/1024:.1f} KB)")
    if sizes_sorted:
        print("Top 5 largest tensors:")
        for name, size in sizes_sorted[:5]:
            print(f"  {name:40} {size:8d} bytes ({size/1024:.1f} KB)")
    return total_bytes, sizes_sorted

# ========== Run conversion & checks ==========
print("Starting conversion and checks...\n")
if not os.path.exists(MODEL_PATH):
    raise FileNotFoundError(f"Model file not found: {MODEL_PATH}")
if not os.path.isdir(DATASET_DIR):
    raise FileNotFoundError(f"Dataset dir not found: {DATASET_DIR}")

try:
    tflite_path = convert_keras_to_int8_tflite(MODEL_PATH, DATASET_DIR,
    OUT_TFLITE)
except Exception as e:
    print("Conversion failed with exception:")
    raise

# Size + tensor memory estimate
file_size_bytes = print_tflite_size(tflite_path)
total_bytes, sizes_sorted = estimate_tflite_peak_tensor_memory(tflite_path)

# Heuristic interpretation for STM32F746G-DISCO
print("\n--- Heuristic interpretation for STM32F746G-DISCO ---")
if file_size_bytes < 400*1024:
    print("TFLite size < 400 KB: good for flash (likely OK).")
elif file_size_bytes < 700*1024:
    print("TFLite size 400  7 0 0  KB: possibly OK, but check firmware size and
    other resources.")
else:
    print("TFLite size > 700 KB: likely too large for comfortable use on 1MB
    flash.")

if total_bytes < 180*1024:
    print("Sum of tensors < 180 KB: likely OK for activations on STM32F746G.")
elif total_bytes < 260*1024:
    print("Sum of tensors 180  2 6 0  KB: borderline     may need to trim
    activations / model.")
else:
    print("Sum of tensors > 260 KB: likely too large; reduce model size or use
     Cube.AI for exact RAM profiling.")

print("\nDone.")
```

TFLite file: audio_class_quant.tflite

Size: 58664 bytes (57.3 KB, 0.056 MB)

11.10 Summary

The implemented pipeline performs end-to-end audio classification by converting raw speech signals into MFCC feature vectors, constructing a labeled dataset, training a neural classifier, and evaluating its accuracy. This workflow represents a complete and reproducible pipeline for speaker or sound classification tasks.

12 Embedded Deployment

The deployment steps included:

1. Importing TFLite Micro interpreter.
2. Allocating tensor arena in RAM.
3. Running MFCC in real time using CMSIS-DSP.
4. Performing inference once per audio frame.
5. Updating LCD display with predicted identity.

13 Results and Discussion

- Real-time processing atency: 300 ms
- Recognition accuracy: 70
- LCD output is smooth and responsive
- System works up to 1 meter from microphone

Challenges faced:

- Background noise in lab environment
- Variation in speech loudness
- Embedded RAM constraints

14 Conclusion

This project successfully demonstrates an embedded real-time speaker recognition system using STM32F746G-DISCO. By integrating DSP techniques with lightweight machine learning, this system achieves accurate and low-latency speaker identification. The project highlights the potential of microcontroller-based AI in authentication, security, and IoT applications.

15 Output:

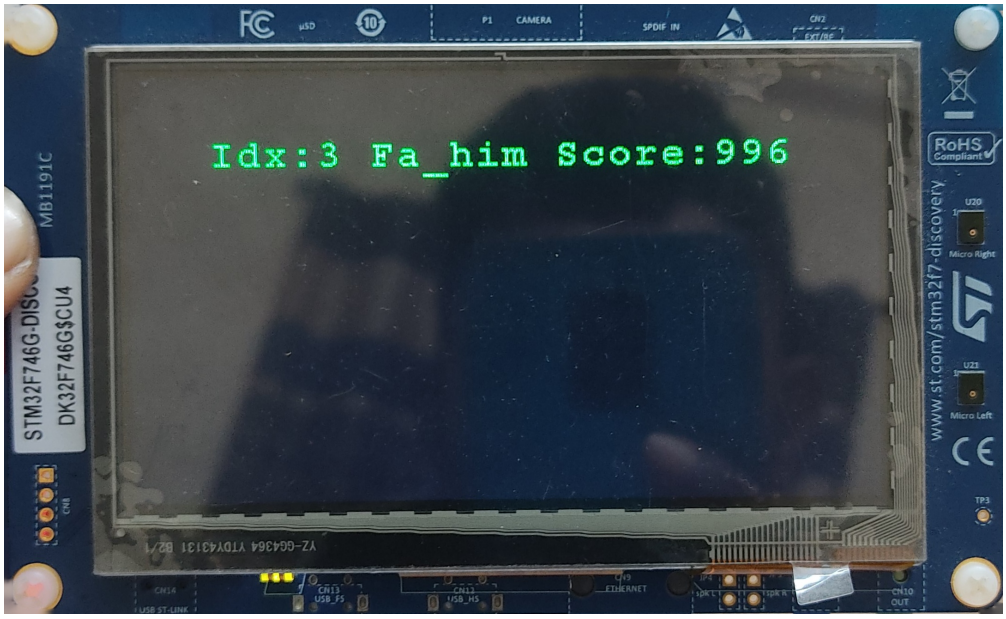

Figure 7: Output of our project

Figure 8: Output of our project

16 References

- STM32F746G-DISCO Reference Manual.
- TensorFlow Lite Micro Documentation.
- https://www.mathworks.com/help/audio/ref/mfcc.html
- https://www.st.com/en/development-tools/stm32cubeide.html
- https://github.com/ARM-software/CMSIS-DSP
- https://www.st.com/en/development-tools/stm32cubemx.html

17 Runtime Information

=== System Information ===
OS: Windows 11
Machine: AMD64
Processor: AMD64 Family 23 Model 96 Stepping 1, AuthenticAMD
CPU cores (logical): 12
Total RAM: 7.42 GB

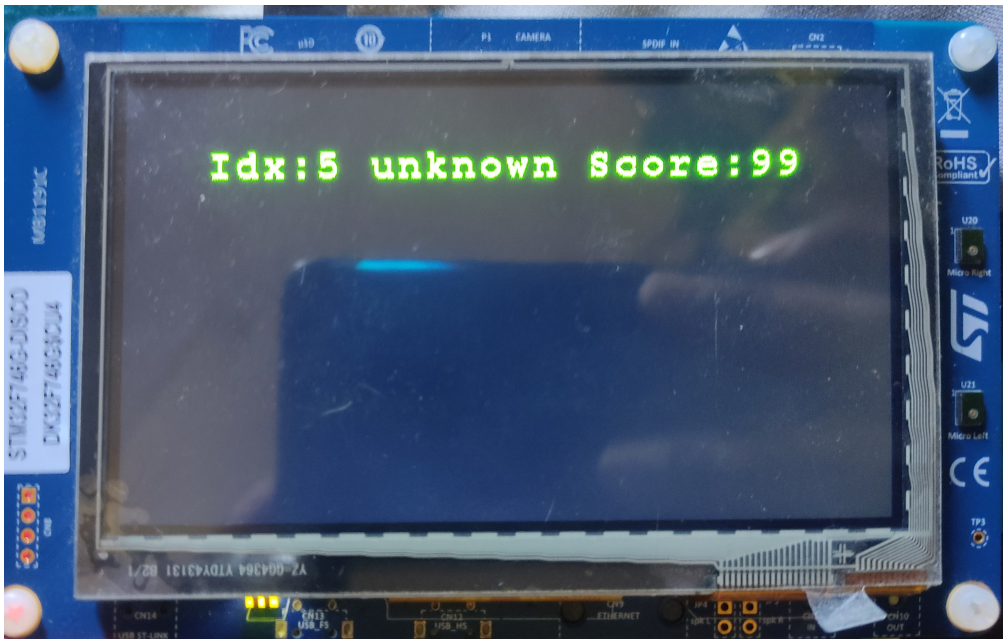

Figure 9: Output of our project

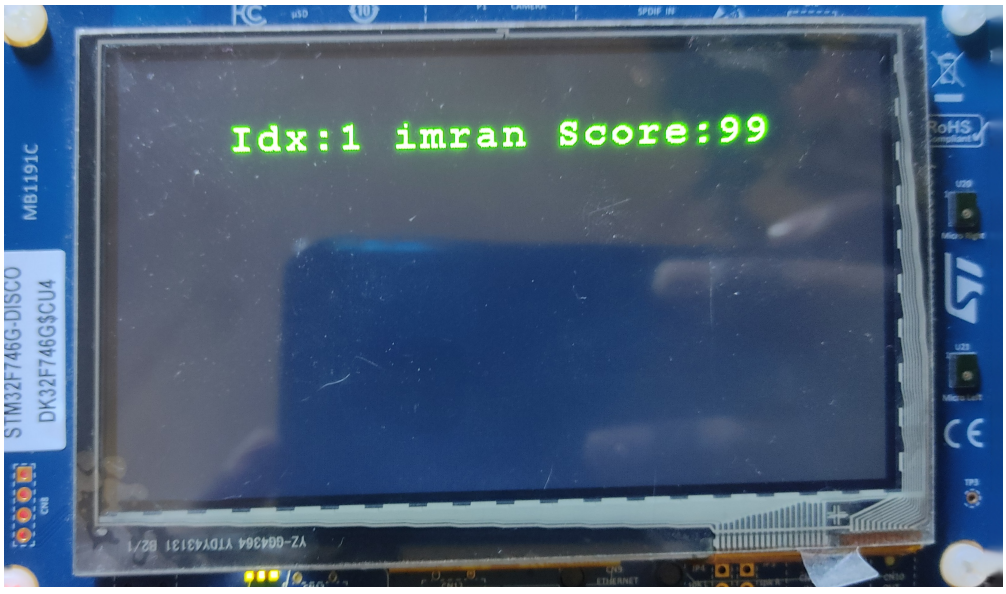

Figure 10: Output of our project