

I M A (Intelligent Magic Audio)

Generated by Doxygen 1.11.0



<b>1 Data Structure Index</b>	<b>1</b>
1.1 Data Structures	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Data Structure Documentation</b>	<b>5</b>
3.1 File Struct Reference	5
3.1.1 Detailed Description	5
3.2 FileManager Struct Reference	5
3.2.1 Detailed Description	6
3.3 FontDef Struct Reference	6
3.4 SSD1306_t Struct Reference	7
3.5 wav_header_t Struct Reference	7
3.5.1 Detailed Description	7
3.6 WavPlayer Struct Reference	8
3.6.1 Detailed Description	8
<b>4 File Documentation</b>	<b>9</b>
4.1 f401_display_encoder_fader_test/Core/Inc/display.h File Reference	9
4.1.1 Detailed Description	9
4.1.2 Function Documentation	10
4.1.2.1 displayStrings()	10
4.1.2.2 drawFaderProzent()	10
4.1.2.3 renderSelectedFile()	11
4.2 display.h	11
4.3 f401_display_encoder_fader_test/Core/Inc/filemanager.h File Reference	12
4.3.1 Detailed Description	13
4.3.2 Macro Definition Documentation	13
4.3.2.1 EXISTS	13
4.3.2.2 MAX_CLASSES	13
4.3.2.3 MAX_FILES	13
4.3.3 Function Documentation	13
4.3.3.1 addFile()	13
4.3.3.2 cursorDown()	14
4.3.3.3 cursorUp()	14
4.3.3.4 initializeFileManager()	14
4.3.3.5 safeCurrentFileName()	15
4.3.3.6 selectFile()	15
4.3.3.7 setCursor()	16
4.4 filemanager.h	16
4.5 fonts.h	17
4.6 f401_display_encoder_fader_test/Core/Inc/interface.h File Reference	17

4.6.1 Detailed Description	19
4.6.2 Macro Definition Documentation	19
4.6.2.1 THREASHOLD	19
4.6.3 Function Documentation	19
4.6.3.1 compareADCValues()	19
4.6.3.2 HAL_ADC_ConvCpltCallback()	20
4.6.3.3 HAL_GPIO_EXTI_Callback()	20
4.6.3.4 HAL_TIM_PeriodElapsedCallback()	21
4.6.3.5 listFiles()	21
4.6.3.6 resetShownFiles()	22
4.6.3.7 screenInit()	22
4.6.3.8 sortFiles()	22
4.6.3.9 updateScreen()	23
4.6.4 Variable Documentation	23
4.6.4.1 adcBuffer	23
4.6.4.2 adcDmaFlag	23
4.6.4.3 fileNameesSDCard	23
4.6.4.4 fm	24
4.6.4.5 fno	24
4.7 interface.h	24
4.8 ssd1306.h	25
4.9 f401_display_encoder_fader_test/Core/Src/display.c File Reference	26
4.9.1 Detailed Description	27
4.9.2 Macro Definition Documentation	27
4.9.2.1 DISPLAY_WIDTH	27
4.9.3 Function Documentation	27
4.9.3.1 displayStrings()	27
4.9.3.2 drawFaderProzent()	28
4.9.3.3 renderSelectedFile()	28
4.9.3.4 scrollDown()	29
4.9.3.5 scrollUp()	29
4.10 f401_display_encoder_fader_test/Core/Src/filemanager.c File Reference	30
4.10.1 Detailed Description	30
4.10.2 Function Documentation	30
4.10.2.1 addFile()	30
4.10.2.2 cursorDown()	31
4.10.2.3 cursorUp()	31
4.10.2.4 initializeFileManager()	31
4.10.2.5 safeCurrentFileName()	32
4.10.2.6 selectFile()	32
4.10.2.7 setCursor()	33
4.11 f401_display_encoder_fader_test/Core/Src/interface.c File Reference	33

4.11.1 Detailed Description	34
4.11.2 Function Documentation	35
4.11.2.1 compareADCValues()	35
4.11.2.2 HAL_ADC_ConvCpltCallback()	35
4.11.2.3 HAL_GPIO_EXTI_Callback()	36
4.11.2.4 HAL_TIM_PeriodElapsedCallback()	36
4.11.2.5 listFiles()	37
4.11.2.6 resetShownFiles()	37
4.11.2.7 screenInit()	37
4.11.2.8 sortFiles()	38
4.11.2.9 updateScreen()	38
4.11.3 Variable Documentation	39
4.11.3.1 rotary_enc_count	39
4.12 f401_sd_card_audio_codec_test/Core/Inc/audio.h File Reference	39
4.12.1 Detailed Description	40
4.12.2 Function Documentation	40
4.12.2.1 fillHalfBufferFromSD()	40
4.12.2.2 generateSineWave()	40
4.12.2.3 HAL_I2S_TxHalfCpltCallback()	41
4.12.2.4 setPitchFactor()	41
4.13 audio.h	41
4.14 f401_sd_card_audio_codec_test/Core/Inc/audioPreprocessor.h File Reference	42
4.14.1 Detailed Description	43
4.14.2 Function Documentation	43
4.14.2.1 downsample_Block()	43
4.14.2.2 downsample_to_1024_samples()	43
4.14.2.3 get_number_subsamples()	44
4.14.2.4 initFilter()	44
4.14.2.5 resampleFile()	44
4.14.3 Variable Documentation	45
4.14.3.1 filter_taps	45
4.15 audioPreprocessor.h	45
4.16 cpu_time.h	46
4.17 f401_sd_card_audio_codec_test/Core/Inc/lowpass_16kFilter.h File Reference	46
4.17.1 Detailed Description	46
4.17.2 Variable Documentation	47
4.17.2.1 filter_taps	47
4.18 lowpass_16kFilter.h	47
4.19 f401_display_encoder_fader_test/Core/Inc/main.h File Reference	47
4.19.1 Detailed Description	48
4.19.2 Function Documentation	48
4.19.2.1 Error_Handler()	48

4.20 main.h . . . . .	49
4.21 f401_sd_card_audio_codec_test/Core/Inc/main.h File Reference . . . . .	50
4.21.1 Detailed Description . . . . .	50
4.21.2 Function Documentation . . . . .	50
4.21.2.1 Error_Handler() . . . . .	50
4.22 main.h . . . . .	51
4.23 neuronal_network/esp_cubeai_test/Core/Inc/main.h File Reference . . . . .	52
4.23.1 Detailed Description . . . . .	53
4.23.2 Function Documentation . . . . .	53
4.23.2.1 Error_Handler() . . . . .	53
4.24 main.h . . . . .	53
4.25 util.h . . . . .	54
4.26 util.h . . . . .	54
4.27 f401_sd_card_audio_codec_test/Core/Inc/wavPlayer.h File Reference . . . . .	55
4.27.1 Detailed Description . . . . .	56
4.27.2 Typedef Documentation . . . . .	56
4.27.2.1 wav_header_t . . . . .	56
4.27.2.2 WavPlayer . . . . .	56
4.27.3 Function Documentation . . . . .	56
4.27.3.1 checkWav() . . . . .	56
4.27.3.2 initPlayer() . . . . .	56
4.27.3.3 playButtonHandler() . . . . .	57
4.27.3.4 populateWavHeader() . . . . .	57
4.27.3.5 wavLoad() . . . . .	57
4.27.3.6 wavPlay() . . . . .	58
4.27.3.7 wavPlayPitched() . . . . .	58
4.28 wavPlayer.h . . . . .	59
4.29 f401_sd_card_audio_codec_test/Core/Src/audio.c File Reference . . . . .	59
4.29.1 Detailed Description . . . . .	60
4.29.2 Function Documentation . . . . .	60
4.29.2.1 fillHalfBufferFromSD() . . . . .	60
4.29.2.2 generateSineWave() . . . . .	61
4.29.2.3 HAL_I2S_TxCpltCallback() . . . . .	61
4.29.2.4 HAL_I2S_TxHalfCpltCallback() . . . . .	61
4.29.2.5 setPitchFactor() . . . . .	61
4.30 f401_sd_card_audio_codec_test/Core/Src/audioPreprocessor.c File Reference . . . . .	62
4.30.1 Detailed Description . . . . .	62
4.30.2 Function Documentation . . . . .	62
4.30.2.1 downsample_Block() . . . . .	62
4.30.2.2 downsample_to_1024_samples() . . . . .	63
4.30.2.3 get_number_subsamples() . . . . .	63
4.30.2.4 initFilter() . . . . .	64

4.30.2.5 resampleFile()	64
4.31 f401_sd_card_audio_codec_test/Core/Src/cpu_time.c File Reference	64
4.31.1 Detailed Description	65
4.31.2 Function Documentation	65
4.31.2.1 CyclesToMicroseconds()	65
4.31.2.2 CyclesToMilliseconds()	65
4.31.2.3 EnableDWT()	66
4.31.2.4 GetMeasuredCycles()	66
4.31.2.5 StartCycleMeasurement()	66
4.31.2.6 StopCycleMeasurement()	66
4.32 f401_sd_card_audio_codec_test/Core/Src/lowpass_16kFilter.c File Reference	67
4.32.1 Detailed Description	67
4.32.2 Variable Documentation	67
4.32.2.1 filter_taps	67
4.33 f401_display_encoder_fader_test/Core/Src/main.c File Reference	67
4.33.1 Detailed Description	69
4.33.2 Function Documentation	69
4.33.2.1 Error_Handler()	69
4.33.2.2 main()	69
4.33.2.3 SystemClock_Config()	69
4.33.2.4 writeStringToFile()	70
4.33.3 Variable Documentation	70
4.33.3.1 adcBuffer	70
4.33.3.2 adcDmaFlag	70
4.33.3.3 fileNameSdCard	71
4.33.3.4 fm	71
4.33.3.5 fno	71
4.33.3.6 lfn	71
4.34 f401_sd_card_audio_codec_test/Core/Src/main.c File Reference	71
4.34.1 Detailed Description	72
4.34.2 Function Documentation	72
4.34.2.1 Error_Handler()	72
4.34.2.2 main()	72
4.34.2.3 SystemClock_Config()	73
4.35 neuronal_network/esp_cubeai_test/Core/Src/main.c File Reference	73
4.35.1 Detailed Description	74
4.35.2 Function Documentation	74
4.35.2.1 Error_Handler()	74
4.35.2.2 main()	74
4.35.2.3 SystemClock_Config()	74
4.36 f401_sd_card_audio_codec_test/Core/Src/util.c File Reference	75
4.36.1 Detailed Description	75

4.36.2 Function Documentation . . . . .	75
4.36.2.1 uart_printf() . . . . .	75
4.37 f401_sd_card_audio_codec_test/Core/Src/wavPlayer.c File Reference . . . . .	76
4.37.1 Detailed Description . . . . .	76
4.37.2 Function Documentation . . . . .	76
4.37.2.1 checkWav() . . . . .	76
4.37.2.2 initPlayer() . . . . .	77
4.37.2.3 playButtonHandler() . . . . .	77
4.37.2.4 populateWavHeader() . . . . .	77
4.37.2.5 wavLoad() . . . . .	78
4.37.2.6 wavPlay() . . . . .	78
4.37.2.7 wavPlayPitched() . . . . .	78
4.38 audio_classification.h . . . . .	79
4.39 audio_classification.h . . . . .	80
4.40 neuronal_network/esp_cubeai_for_integration/Core/Src/audio_classification.c File Reference . . . . .	80
4.40.1 Detailed Description . . . . .	82
4.40.2 Function Documentation . . . . .	82
4.40.2.1 calculate_spectrogram_column() . . . . .	82
4.40.2.2 calculate_total_classification_result() . . . . .	82
4.40.2.3 classify_file() . . . . .	83
4.40.2.4 de_init_nn() . . . . .	83
4.40.2.5 init_nn() . . . . .	83
4.40.2.6 run_nn_classification() . . . . .	84
4.40.2.7 spectrogram_generation_init() . . . . .	84
4.40.2.8 spectrogram_power_to_db() . . . . .	84
4.40.2.9 store_classification_result() . . . . .	85
4.40.3 Variable Documentation . . . . .	85
4.40.3.1 aiOutDataLabels . . . . .	85
4.41 neuronal_network/esp_cubeai_test/Core/Src/audio_classification.c File Reference . . . . .	86
4.41.1 Detailed Description . . . . .	87
4.41.2 Function Documentation . . . . .	87
4.41.2.1 frame_subsamples() . . . . .	87
4.41.2.2 init_nn() . . . . .	87
4.41.2.3 PowerToDb() . . . . .	88
4.41.2.4 preprocess_frame() . . . . .	88
4.41.2.5 Preprocessing_Init() . . . . .	88
4.41.2.6 process_subsample() . . . . .	89
4.41.2.7 run_nn_classification() . . . . .	90
4.41.2.8 test_call_classification() . . . . .	90
4.42 featurescaler.h . . . . .	90
4.43 test_audio_data.h . . . . .	91
4.44 utils.h . . . . .	91







# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">File</a>	Structure representing a file with its name and classification classes . . . . .	5
<a href="#">FileManager</a>	Structure to manage the files and their display on the LCD . . . . .	5
<a href="#">FontDef</a>	. . . . .	6
<a href="#">SSD1306_t</a>	. . . . .	7
<a href="#">wav_header_t</a>	Represents the header of a WAV file . . . . .	7
<a href="#">WavPlayer</a>	Represents a WAV audio player with playback control and state management . . . . .	8



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

f401_display_encoder_fader_test/Core/Inc/display.h	9
: Header file for display functions and graphics rendering	
f401_display_encoder_fader_test/Core/Inc/filemanager.h	12
: Header file for file management functions and structures	
f401_display_encoder_fader_test/Core/Inc/fonts.h	17
f401_display_encoder_fader_test/Core/Inc/interface.h	17
: Header file for interface functions and global definitions	
f401_display_encoder_fader_test/Core/Inc/main.h	47
: Header for main.c file. This file contains the common defines of the application	
f401_display_encoder_fader_test/Core/Inc/ssd1306.h	25
f401_display_encoder_fader_test/Core/Inc/util.h	54
f401_display_encoder_fader_test/Core/Src/display.c	26
: Display handling functions for OLED screen	
f401_display_encoder_fader_test/Core/Src/filemanager.c	30
: File management functions for handling files and cursor operations	
f401_display_encoder_fader_test/Core/Src/interface.c	33
: Interface functions for handling user inputs, file management, and display updates	
f401_display_encoder_fader_test/Core/Src/main.c	67
: Main program body	
f401_sd_card_audio_codec_test/Core/Inc/audio.h	39
Header file for audio playback and processing functions	
f401_sd_card_audio_codec_test/Core/Inc/audioPreprocessor.h	42
Header file for the audio preprocessing DSP-Operations	
f401_sd_card_audio_codec_test/Core/Inc/cpu_time.h	46
f401_sd_card_audio_codec_test/Core/Inc/lowpass_16kFilter.h	46
Header file for the low-pass FIR filter used for decimation	
f401_sd_card_audio_codec_test/Core/Inc/main.h	50
: Header for main.c file. This file contains the common defines of the application	
f401_sd_card_audio_codec_test/Core/Inc/util.h	54
f401_sd_card_audio_codec_test/Core/Inc/wavPlayer.h	55
Header file for WAV audio player functionality	
f401_sd_card_audio_codec_test/Core/Src/audio.c	59
Audio playback and processing functions	
f401_sd_card_audio_codec_test/Core/Src/audioPreprocessor.c	62
Audio preprocessing functions for FIR decimation and resampling	

f401_sd_card_audio_codec_test/Core/Src/ <a href="#">cpu_time.c</a>	
Functions for cycle counting and time measurement using DWT on STM32 . . . . .	64
f401_sd_card_audio_codec_test/Core/Src/ <a href="#">lowpass_16kFilter.c</a>	
Contains the filter coefficients for a 16 kHz low-pass FIR filter . . . . .	67
f401_sd_card_audio_codec_test/Core/Src/ <a href="#">main.c</a>	
: Main program body . . . . .	71
f401_sd_card_audio_codec_test/Core/Src/ <a href="#">util.c</a>	
Utility functions for the project . . . . .	75
f401_sd_card_audio_codec_test/Core/Src/ <a href="#">wavPlayer.c</a>	
Implementation of a WAV player using FATFS and I2S with DMA . . . . .	76
neuronal_network/esp_cubeai_for_integration/Core/Inc/ <a href="#">audio_classification.h</a>	79
neuronal_network/esp_cubeai_for_integration/Core/Src/ <a href="#">audio_classification.c</a>	
Audio classification implementation for embedded systems via stm32cube.ai (for integration into the system). Everything is untested! . . . . .	80
neuronal_network/esp_cubeai_test/Core/Inc/ <a href="#">audio_classification.h</a>	80
neuronal_network/esp_cubeai_test/Core/Inc/ <a href="#">featurescaler.h</a>	90
neuronal_network/esp_cubeai_test/Core/Inc/ <a href="#">main.h</a>	
: Header for main.c file. This file contains the common defines of the application . . . . .	52
neuronal_network/esp_cubeai_test/Core/Inc/ <a href="#">test_audio_data.h</a>	91
neuronal_network/esp_cubeai_test/Core/Inc/ <a href="#">utils.h</a>	91
neuronal_network/esp_cubeai_test/Core/Src/ <a href="#">audio_classification.c</a>	
Audio classification implementation for embedded systems via stm32cube.ai (for testing purposes) . . . . .	86
neuronal_network/esp_cubeai_test/Core/Src/ <a href="#">main.c</a>	
: Main program body . . . . .	73

## Chapter 3

# Data Structure Documentation

### 3.1 File Struct Reference

Structure representing a file with its name and classification classes.

```
#include <filemanager.h>
```

#### Data Fields

- char **filename** [MAX\_FILENAME\_LENGTH]  
*The name of the file.*
- float **classes** [MAX\_CLASSES]  
*The classification classes for the file.*

#### 3.1.1 Detailed Description

Structure representing a file with its name and classification classes.

This structure stores information about a file, including its name and its classification classes.

The documentation for this struct was generated from the following file:

- f401\_display\_encoder\_fader\_test/Core/Inc/[filemanager.h](#)

### 3.2 FileManager Struct Reference

Structure to manage the files and their display on the LCD.

```
#include <filemanager.h>
```

## Data Fields

- **File files** [[MAX\\_FILES](#)]  
*Array to store all files that have been classified.*
- **File shownFiles** [[MAX\\_FILES](#)]  
*Array to store files that match the fader settings.*
- **char current\_cursor\_filename** [25]  
*The filename of the selected by the user.*
- **int num\_files**  
*The total number of files.*
- **int num\_matched\_files**  
*The number of files that match the fader settings.*
- **int cursor\_index**  
*The index of the current cursor position.*
- **int current\_file\_index**  
*The index position of the selected file.*
- **float fader\_Class** [[MAX\\_CLASSES](#)]  
*The settings from fader.*

### 3.2.1 Detailed Description

Structure to manage the files and their display on the LCD.

This structure is used to manage files, including storing all files, filtering files based on fader settings, and keeping track of the current selection.

The documentation for this struct was generated from the following file:

- [f401\\_display\\_encoder\\_fader\\_test/Core/Inc/filemanager.h](#)

## 3.3 FontDef Struct Reference

### Data Fields

- **const uint8\_t FontWidth**
- **uint8\_t FontHeight**
- **const uint16\_t \* data**

The documentation for this struct was generated from the following file:

- [f401\\_display\\_encoder\\_fader\\_test/Core/Inc/fonts.h](#)



## 3.4 SSD1306\_t Struct Reference

### Data Fields

- uint16\_t **CurrentX**
- uint16\_t **CurrentY**
- uint8\_t **Inverted**
- uint8\_t **Initialized**

The documentation for this struct was generated from the following file:

- f401\_display\_encoder\_fader\_test/Core/Inc/ssd1306.h

## 3.5 wav\_header\_t Struct Reference

Represents the header of a WAV file.

```
#include <wavPlayer.h>
```

### Data Fields

- uint32\_t **ChunkID**
- uint32\_t **ChunkSize**
- uint32\_t **Format**
- uint32\_t **Subchunk1ID**
- uint32\_t **Subchunk1Size**
- uint16\_t **AudioFormat**
- uint16\_t **NumChannels**
- uint32\_t **SampleRate**
- uint32\_t **ByteRate**
- uint16\_t **BlockAlign**
- uint16\_t **BitsPerSample**
- uint32\_t **Subchunk2ID**
- uint32\_t **Subchunk2Size**

### 3.5.1 Detailed Description

Represents the header of a WAV file.

This structure defines the layout of the WAV file header, which includes information about the RIFF chunk, format details, and the data chunk. It provides essential metadata for interpreting the WAV file's audio data.

The documentation for this struct was generated from the following file:

- f401\_sd\_card\_audio\_codec\_test/Core/Inc/[wavPlayer.h](#)

## 3.6 WavPlayer Struct Reference

Represents a WAV audio player with playback control and state management.

```
#include <wavPlayer.h>
```

### Data Fields

- volatile bool **restartPlayback**
- volatile bool **playbackActive**
- FIL \* **file**
- [wav\\_header\\_t](#) \* **wavHeader**
- uint32\_t **headerSize**
- float **pitchFactor**
- bool **pitchChanged**

### 3.6.1 Detailed Description

Represents a WAV audio player with playback control and state management.

This structure contains information and control flags related to WAV audio playback, including file handling, playback status, and pitch adjustment.

The documentation for this struct was generated from the following file:

- f401\_sd\_card\_audio\_codec\_test/Core/Inc/[wavPlayer.h](#)

# Chapter 4

## File Documentation

### 4.1 f401\_display\_encoder\_fader\_test/Core/Inc/display.h File Reference

: Header file for display functions and graphics rendering

```
#include <math.h>
#include <stdlib.h>
#include "ssd1306.h"
#include "fonts.h"
```

#### Functions

- void [displayStrings](#) (I2C\_HandleTypeDef \*hi2c1, char \*\*strings, uint8\_t numStrings, uint8\_t cursor\_index)  
*Displays a list of strings on the OLED screen with cursor highlighting.*
- void [renderSelectedFile](#) (I2C\_HandleTypeDef \*hi2c1, const char \*filename)  
*Renders the selected file name on the OLED screen.*
- void **drawWaveform** (I2C\_HandleTypeDef \*hi2c1, int16\_t \*samples, uint32\_t numSamples)
- void **drawLine** (int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, uint16\_t color)
- void [drawFaderProzent](#) (I2C\_HandleTypeDef \*hi2c1, const char \*filename, int multiplikator)  
*Draws the fader percentage on the OLED screen at a specified vertical position.*

#### 4.1.1 Detailed Description

: Header file for display functions and graphics rendering

##### Attention

This file contains the function prototypes for managing and rendering graphics on the display. It includes functions for:

- Displaying a list of strings on the screen ([displayStrings](#)).
- Rendering the currently selected file name ([renderSelectedFile](#)).
- Drawing waveforms based on sample data ([drawWaveform](#)).
- Drawing lines on the display ([drawLine](#)).
- Displaying fader percentage values ([drawFaderProzent](#)).

These functions interact with the SSD1306 display using I2C communication and are crucial for visual representation of data and user interface elements.

## 4.1.2 Function Documentation

### 4.1.2.1 displayStrings()

```
void displayStrings (
    I2C_HandleTypeDef * hi2c1,
    char ** strings,
    uint8_t numStrings,
    uint8_t cursor_index)
```

Displays a list of strings on the OLED screen with cursor highlighting.

This function renders a list of strings on the OLED display, highlighting the current selection with a cursor. It also manages the display of strings based on the current cursor position and adjusts the visible portion of the list as needed.

- Clears the screen before drawing new content.
- Calculates the visible range of strings based on the `cursor_index` and ensures that the cursor is within the visible range.
- Draws each string on the display, with the current selection highlighted by a cursor.
- Draws a border around the list section of the screen.

#### Parameters

in	<i>hi2c1</i>	Pointer to the I2C handle used for communication with the OLED display.
in	<i>strings</i>	Array of string pointers to be displayed.
in	<i>numStrings</i>	Total number of strings in the array.
in	<i>cursor_index</i>	Index of the currently selected string, which will be highlighted on the display.

#### Note

Ensure that the `strings` array is correctly populated and `numStrings` reflects the actual number of valid strings. The display dimensions and font sizes should be configured to match the display hardware.

### 4.1.2.2 drawFaderProzent()

```
void drawFaderProzent (
    I2C_HandleTypeDef * hi2c1,
    const char * prozent,
    int multiplikator)
```

Draws the fader percentage on the OLED screen at a specified vertical position.

This function updates the display to show the fader percentage in a designated area of the screen. It first clears the region where the fader percentage will be displayed, and then writes the percentage at a vertical position determined by the `multiplikator`.

The `multiplikator` parameter is used to calculate the vertical offset from a base position on the screen. This allows for the display of multiple fader percentages at different vertical positions.

## Parameters

in	<i>hi2c1</i>	Pointer to the I2C handle used for communication with the OLED display.
in	<i>prozent</i>	Pointer to a null-terminated string representing the fader percentage to be displayed.
in	<i>multiplikator</i>	Integer value used to adjust the vertical position on the screen, affecting where the percentage is drawn.

## Note

Ensure that the `prozent` string is properly null-terminated and that the display dimensions and font settings are configured correctly to match the hardware. The `multiplikator` should be set appropriately to ensure that the displayed text does not overlap with other screen elements.

## 4.1.2.3 renderSelectedFile()

```
void renderSelectedFile (  
    I2C_HandleTypeDef * hi2c1,  
    const char * filename)
```

Renders the selected file name on the OLED screen.

This function updates the display to show the currently selected file name in a dedicated section of the screen. It first clears the area designated for displaying the selected file name, and then writes the file name at the appropriate position.

## Parameters

in	<i>hi2c1</i>	Pointer to the I2C handle used for communication with the OLED display.
in	<i>filename</i>	Pointer to a null-terminated string representing the name of the selected file to be displayed.

## Note

Ensure that the `filename` is properly null-terminated and that the display dimensions and font settings are configured correctly to match the hardware.

## 4.2 display.h

[Go to the documentation of this file.](#)

```
00001  
00023 #include <math.h>  
00024 #include <stdlib.h>  
00025 #include "ssd1306.h"  
00026 #include "fonts.h"  
00027  
00028 void displayStrings(I2C_HandleTypeDef *hi2c1, char** strings, uint8_t numStrings, uint8_t  
    cursor_index);  
00029 void renderSelectedFile(I2C_HandleTypeDef *hi2c1, const char *filename);  
00030 void drawWaveform(I2C_HandleTypeDef *hi2c1, int16_t *samples, uint32_t numSamples);  
00031 void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint16_t color);  
00032 void drawFaderProzent(I2C_HandleTypeDef *hi2c1, const char *filename, int multiplikator);
```

## 4.3 f401\_display\_encoder\_fader\_test/Core/Inc/filemanager.h File Reference

: Header file for file management functions and structures

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

### Data Structures

- struct [File](#)  
*Structure representing a file with its name and classification classes.*
- struct [FileManager](#)  
*Structure to manage the files and their display on the LCD.*

### Macros

- #define [MAX\\_FILES](#) 500  
*Maximum number of files and filename length.*
- #define [MAX\\_FILENAME\\_LENGTH](#) 30
- #define [MAX\\_CLASSES](#) 4  
*Number of classes for file classification.*
- #define [EXISTS](#) 0  
*[File](#) existence status codes.*
- #define [NOT\\_EXISTS](#) -1

### Functions

- void [initializeFileManager](#) ([FileManager](#) \*fm)  
*Initializes default values for the [FileManager](#).*
- void [addFile](#) ([FileManager](#) \*fm, const char \*filename, const float \*classes)  
*Adds a file to the [FileManager](#).*
- void [selectFile](#) ([FileManager](#) \*fm)  
*Sets the current file index to the cursor position.*
- void [cursorUp](#) ([FileManager](#) \*fm)  
*Decrements the cursor index in the [FileManager](#).*
- void [cursorDown](#) ([FileManager](#) \*fm)  
*Increments the cursor index in the [FileManager](#).*
- void [safeCurrentFileName](#) ([FileManager](#) \*fm)  
*Saves the filename at the current cursor index.*
- void [setCursor](#) ([FileManager](#) \*fm)  
*Sets the cursor position based on the currently displayed files in the [FileManager](#).*

### 4.3.1 Detailed Description

: Header file for file management functions and structures

Attention

This file contains the definitions for managing files within the file management system. It includes:

- Constants defining limits and statuses for files.
- Structures for storing file information and managing file operations.
- Function prototypes for initializing the file manager, adding files, handling file selection, and managing cursor operations.

The [FileManager](#) structure handles the storage and management of files, including their classification and display. This file serves as an interface for interacting with the file management system, providing functions to manipulate file lists and update file selections.

### 4.3.2 Macro Definition Documentation

#### 4.3.2.1 EXISTS

```
#define EXISTS 0
```

[File](#) existence status codes.

These constants are used to indicate whether a file exists or not.

#### 4.3.2.2 MAX\_CLASSES

```
#define MAX_CLASSES 4
```

Number of classes for file classification.

This constant defines the number of classes used in the sorting algorithm to categorize files.

#### 4.3.2.3 MAX\_FILES

```
#define MAX_FILES 500
```

Maximum number of files and filename length.

These constants define the maximum number of files that can be handled and the maximum length of a filename.

### 4.3.3 Function Documentation

#### 4.3.3.1 addFile()

```
void addFile (  
    FileManager * fm,  
    const char * filename,  
    const float * classes)
```

Adds a file to the [FileManager](#).

This function adds a file to the `files[]` array within the [FileManager](#) structure. It associates the given file name with its corresponding class values and updates the file manager accordingly.

**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure to which the file will be added.
in	<i>filename</i>	The name of the file to be added to the <a href="#">FileManager</a> .
in	<i>classes</i>	Array of float values representing the file's class percentages.

**Note**

Ensure that the [FileManager](#) has sufficient capacity in the `files[]` array to accommodate new files.

**4.3.3.2 cursorDown()**

```
void cursorDown (
    FileManager * fm)
```

Increments the cursor index in the [FileManager](#).

This function increments the `cursor_index` within the [FileManager](#) structure to move the cursor visually lower on the LCD display.

**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure whose <code>cursor_index</code> is to be incremented.
----	-----------	--

**4.3.3.3 cursorUp()**

```
void cursorUp (
    FileManager * fm)
```

Decrements the cursor index in the [FileManager](#).

This function decrements the `cursor_index` within the [FileManager](#) structure to move the cursor visually higher on the LCD display.

**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure whose <code>cursor_index</code> is to be decremented.
----	-----------	--

**4.3.3.4 initializeFileManager()**

```
void initializeFileManager (
    FileManager * fm)
```

Initializes default values for the [FileManager](#).

This function sets default values for the given [FileManager](#) structure. It initializes various fields to prepare the file manager for use.



**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure to be initialized.
----	-----------	---

**4.3.3.5 safeCurrentFileName()**

```
void safeCurrentFileName (  
    FileManager * fm)
```

Saves the filename at the current cursor index.

This function copies the filename of the file currently pointed to by `cursor_index` in the [FileManager](#) structure to `current_cursor_filename`. It updates the `current_cursor_filename` with the filename from the `shownFiles` array at the position indicated by `cursor_index`.

**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure from which the filename is retrieved and stored.
----	-----------	---

**Note**

Ensure that `cursor_index` is within the valid range of indices in the `shownFiles` array to avoid out-of-bounds access. The `current_cursor_filename` should have sufficient space allocated to store the filename.

**4.3.3.6 selectFile()**

```
void selectFile (  
    FileManager * fm)
```

Sets the current file index to the cursor position.

This function updates the `current_file_index` in the [FileManager](#) to the current position of the `cursor_index`. This allows the file at the cursor position to be marked as the selected file, which can then be processed or displayed as needed.

**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure in which the <code>current_file_index</code> will be set to the value of <code>cursor_index</code> .
----	-----------	---

#### 4.3.3.7 setCursor()

```
void setCursor (
    FileManager * fm)
```

Sets the cursor position based on the currently displayed files in the [FileManager](#).

This function adjusts the `cursor_index` in the [FileManager](#) based on the currently displayed files and their relevance to the fader settings. The behavior of the function is as follows:

- If no files match the fader settings (`num_matched_files` is 0):
  - The cursor is set to the start (index 0).
- If the `current_cursor_filename` matches one of the filenames in `shownFiles`:
  - The cursor is positioned at the index of the matching file in the `shownFiles` array.
- If the `current_cursor_filename` no longer exists in `shownFiles`:
  - The `current_cursor_filename` is updated to the filename at the position indicated by `current_file_index`.
- If `cursor_index` is larger than the number of matched files:
  - The `cursor_index` is set to the last position in the list.

#### Parameters

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure used to adjust the cursor position.
----	-----------	--

## 4.4 filemanager.h

[Go to the documentation of this file.](#)

```
00001
00024 #ifndef FILE_MANAGER_H
00025 #define FILE_MANAGER_H
00026
00027 #include <stdio.h>
00028 #include <stdlib.h>
00029 #include <string.h>
00030
00036 #define MAX_FILES 500
00037 #define MAX_FILENAME_LENGTH 30
00038
00044 #define MAX_CLASSES 4
00045
00051 #define EXISTS 0
00052 #define NOT_EXISTS -1
00053
00060 typedef struct {
00061     char filename[MAX_FILENAME_LENGTH];
00062     float classes[MAX_CLASSES];
00063 } File;
00064
00072 typedef struct {
00073     File files[MAX_FILES];
00074     File shownFiles[MAX_FILES];
00075     char current_cursor_filename[25];
00076     int num_files;
00077     int num_matched_files;
00078     int cursor_index;
00079     int current_file_index;
00080     float fader_Class[MAX_CLASSES];
00081 } FileManager;
00082
```

```

00083 void initializeFileManager(FileManager *fm);
00084 void addFile(FileManager *fm, const char *filename, const float *classes);
00085 void selectFile(FileManager *fm);
00086 void cursorUp(FileManager *fm);
00087 void cursorDown(FileManager *fm);
00088 void safeCurrentFileName(FileManager *fm);
00089 void setCursor(FileManager *fm);
00090
00091
00092
00093 #endif /* FILE_MANAGER_H */

```

## 4.5 fonts.h

```

00001 #ifndef _FONTS_H
00002 #define _FONTS_H
00003
00004 #include <stdint.h>
00005
00006 //
00007 // Structure used to define fonts
00008 //
00009 typedef struct {
00010     const uint8_t FontWidth;    /* Font width in pixels */
00011     uint8_t FontHeight;        /* Font height in pixels */
00012     const uint16_t *data;       /* Pointer to data font data array */
00013 } FontDef;
00014
00015 //
00016 // Export the 3 available fonts
00017 //
00018 extern FontDef Font_7x10;
00019 extern FontDef Font_11x18;
00020 extern FontDef Font_16x26;
00021
00022 #endif // _FONTS_H

```

## 4.6 f401\_display\_encoder\_fader\_test/Core/Inc/interface.h File Reference

: Header file for interface functions and global definitions

```

#include <stdlib.h>
#include <stdbool.h>
#include "ssd1306.h"
#include "fonts.h"
#include "display.h"
#include "filemanager.h"
#include "main.h"
#include "ff.h"
#include "util.h"

```

### Macros

- #define **THREASHOLD** 0.1f  
*Definitions for thresholds, existence checks, comparison results, and smoothing height.*
- #define **EXISTS** 0  
*Value indicating existence.*
- #define **NOT\_EXISTS** -1  
*Value indicating non-existence.*
- #define **SAME** 0  
*Value indicating comparison result is the same.*
- #define **NOT\_SAME** -1  
*Value indicating comparison result is different.*
- #define **SMOOTHING\_HEIGHT** 30000  
*Height value used for smoothing calculations.*

## Functions

- void [HAL\\_ADC\\_ConvCpltCallback](#) (ADC\_HandleTypeDef \*hadc)  
*Callback function for the ADC conversion.*
- void [HAL\\_GPIO\\_EXTI\\_Callback](#) (uint16\_t GPIO\_Pin)  
*Callback function that responds to encoder signals.*
- void [HAL\\_TIM\\_PeriodElapsedCallback](#) (TIM\_HandleTypeDef \*htim)  
*Callback function triggered when a timer expires.*
- void [screenInit](#) (char(\*fileNames)[MAX\_FILENAME\_LENGTH])  
*Initializes the screen and sets up the file manager with random class percentages.*
- void [updateScreen](#) (void)  
*Updates the display with filenames, the currently selected file, and fader settings.*
- void [sortFiles](#) (void)  
*Sorts filenames for display based on fader settings.*
- void [compareADCValues](#) (void)  
*Checks if fader settings have changed and updates accordingly.*
- void [resetShownFiles](#) (void)  
*Resets the `shownFiles` array and updates the number of matched files.*
- void [writeFileManagerOnSD](#) (void)
- void [readFileManagerFromSD](#) (void)
- void [listFiles](#) (const char \*path)  
*Lists all filenames with a .wav extension from the specified directory on the SD card.*

## Variables

- ADC\_HandleTypeDef **hadc1**
- DMA\_HandleTypeDef **hdma\_adc1**
- I2C\_HandleTypeDef **hi2c1**
- TIM\_HandleTypeDef **htim3**
- TIM\_HandleTypeDef **htim5**
- [FileManager](#) **fm**  
*FileManager instances for managing and testing files.*
- [FileManager](#) **fmCopy**  
*Copy of FileManager for testing purposes.*
- FATFS **FatFs**  
*FATFS handle for the file system.*
- FILINFO **fno**  
*FATFS file system objects and handles.*
- DIR **dir**  
*Directory handle for directory operations.*
- FRESULT **fres**  
*Result of FATFS operations.*
- FIL **file**  
*File object for FATFS.*
- uint32\_t [adcBuffer](#) [NUM\_CHANNELS]  
*Buffer for ADC polling.*
- char [fileNamesSDCard](#) [MAX\_FILES][MAX\_FILENAME\_LENGTH]  
*List of filenames on the SD card with .wav extension.*
- int **fileCount**  
*Counter for the number of .wav files found on the SD card.*
- bool [adcDmaFlag](#)

*Flags for managing time intervals and DMA operations.*

- bool **updateScreenFlag**

*Flag to manage the time interval for display updates.*

- bool **sortFilesFlag**

*Flag to regroup the shown file names.*

## 4.6.1 Detailed Description

: Header file for interface functions and global definitions

### Attention

This file contains function prototypes and global definitions for managing user inputs, file operations, and display updates. It includes:

- Function prototypes for handling ADC conversions, GPIO interrupts, and timer callbacks.
- Functions for initializing the display, updating the screen, sorting files, comparing ADC values, resetting displayed files, and managing file storage on SD.
- Global variables and external references related to ADC, I2C, timer, file management, and display operations.

These functions and definitions are crucial for interfacing between the user inputs, display rendering, and file management system, ensuring smooth operation of the application.

## 4.6.2 Macro Definition Documentation

### 4.6.2.1 THREASHOLD

```
#define THREASHOLD 0.1f
```

Definitions for thresholds, existence checks, comparison results, and smoothing height.

These macros are used for setting thresholds, checking existence, comparing results, and defining the height for smoothing calculations. Threshold value for class comparison

## 4.6.3 Function Documentation

### 4.6.3.1 compareADCValues()

```
void compareADCValues (
    void )
```

Checks if fader settings have changed and updates accordingly.

This function compares the current fader settings with previous values to determine if any changes have occurred. If changes are detected, it performs the following actions:

- Compares the current and past fader settings.
- Updates the past fader settings with the current values if a change is detected.
- Sets a flag to indicate that the file list needs to be sorted.
- Saves the current filename and clears the `shownFiles` array if a change is detected.

### Note

This function uses global variables for fader settings and file management.

#### 4.6.3.2 HAL\_ADC\_ConvCpltCallback()

```
void HAL_ADC_ConvCpltCallback (
    ADC_HandleTypeDef * hadc)
```

Callback function for the ADC conversion.

This function is called when the ADC conversion is complete. It performs the following actions:

- Calculates the smooth values for all channels of the ADC.
- Computes the average values for display and comparison operations.
  - These average values are used for displaying on the screen and for the sorting algorithm's comparison operations.
- Initializes the character array that will be shown on the display.

##### Parameters

in	<i>hadc</i>	Pointer to the ADC handle structure.
----	-------------	--------------------------------------

#### 4.6.3.3 HAL\_GPIO\_EXTI\_Callback()

```
void HAL_GPIO_EXTI_Callback (
    uint16_t GPIO_Pin)
```

Callback function that responds to encoder signals.

This function handles encoder signals by adjusting the encoder count and cursor position on the display. It also debounces the encoder switch using Timer5 and a debounce flag.

- If A is High and B is High:
  - Decrement the encoder count and increment the cursor position on the display.
- If A is High and B is Low:
  - Increment the encoder count and decrement the cursor position on the display.
- If the switch is pushed:
  - Select the file and debounce the encoder switch using Timer5 and the debounce flag.

##### Parameters

in	<i>GPIO_pin</i>	The GPIO pin signals from the MCU related to the encoder and switch.
----	-----------------	--

#### 4.6.3.4 HAL\_TIM\_PeriodElapsedCallback()

```
void HAL_TIM_PeriodElapsedCallback (
    TIM_HandleTypeDef * htim)
```

Callback function triggered when a timer expires.

This function handles timer expirations by performing the following actions:

For Timer5 (tim5):

- Resets the debounce flag after Timer5 has elapsed.
- Stops Timer5.

For Timer3 (tim3):

- Sets flags to indicate DMA completion and screen update.
- Sets a new timer for the next refresh interval.

##### Parameters

in	<i>htim</i>	Pointer to the TIM_HandleTypeDef structure for the current timer.
----	-------------	---

#### 4.6.3.5 listFiles()

```
void listFiles (
    const char * path)
```

Lists all filenames with a .wav extension from the specified directory on the SD card.

This function opens the directory specified by the `path` parameter, reads through its contents, and stores filenames with a .wav extension in the `fileNamesSDCard` array. It performs the following actions:

- Opens the directory at the specified path.
- Iterates through directory entries until reaching the maximum file count or encountering an end-of-directory condition.
- Checks if each entry is a file (not a directory) and if it has a .wav extension.
- Stores valid filenames in the `fileNamesSDCard` array.
- Closes the directory after reading all entries.

##### Parameters

in	<i>path</i>	The directory path on the SD card to search for files.
----	-------------	--

##### Note

This function assumes that `fileNamesSDCard` and `fileCount` are global variables and that `MAX_FILES` and `MAX_FILENAME_LENGTH` are defined constants.

#### 4.6.3.6 resetShownFiles()

```
void resetShownFiles (
    void )
```

Resets the `shownFiles` array and updates the number of matched files.

This function sets all entries in the `shownFiles` array to zero and resets the count of matched files.

- Sets the memory area of `fm.shownFiles` to zero using `memset`.
- Resets the `num_matched_files` field to 0.

#### Note

This function is used to clear the list of files currently displayed on the screen.

#### 4.6.3.7 screenInit()

```
void screenInit (
    char(*) fileName[MAX_FILENAME_LENGTH])
```

Initializes the screen and sets up the file manager with random class percentages.

This function initializes the LCD-Display, fills it with black, and updates the screen. It also initializes the file manager with file names and assigns random class percentages for system tests.

#### Parameters

in	<i>fileName</i>	Array of file names to be added to the file manager.
----	-----------------	--

#### 4.6.3.8 sortFiles()

```
void sortFiles (
    void )
```

Sorts filenames for display based on fader settings.

This function sorts filenames by comparing file classes with the fader settings. It performs the following actions:

- Compares file classes against the fader settings with a threshold of 0.1.
- Compares the newly added filenames with the existing ones.
- Adds files from the existing file list to the `shownFiles` in the file manager.

#### Parameters

in	<i>void</i>	
----	-------------	--



#### 4.6.3.9 updateScreen()

```
void updateScreen (
    void )
```

Updates the display with filenames, the currently selected file, and fader settings.

This function updates the display by performing the following tasks:

- Extracts filenames from the file manager's `shownFiles` array.
- Sets the current filename to be displayed based on the initialization state.
- Displays all filenames in the `shownFiles` array.
- Display the currently selected filename.
- Shows the fader settings on the display.

#### Note

The function relies on global variables and external functions to interact with the display.

### 4.6.4 Variable Documentation

#### 4.6.4.1 adcBuffer

```
uint32_t adcBuffer[NUM_CHANNELS] [extern]
```

Buffer for ADC polling.

This buffer stores the values read from the ADC channels. Buffer for storing ADC channel values.

#### 4.6.4.2 adcDmaFlag

```
bool adcDmaFlag [extern]
```

Flags for managing time intervals and DMA operations.

These flags are used to manage the intervals at which DMA operations and display updates occur. Flag to manage the time interval for ADC DMA operations.

#### 4.6.4.3 fileNameSdCard

```
char fileNameSdCard[MAX_FILES][MAX_FILENAME_LENGTH] [extern]
```

List of filenames on the SD card with .wav extension.

This array stores the filenames of all .wav files found on the SD card. Array of filenames on the SD card with .wav extension.

#### 4.6.4.4 fm

```
FileManager fm [extern]
```

[FileManager](#) instances for managing and testing files.

These variables are used to manage files, including the main [FileManager](#) and a copy for testing purposes. Main [FileManager](#) instance.

#### 4.6.4.5 fno

```
FILINFO fno [extern]
```

FATFS file system objects and handles.

These variables are used for managing file operations and directory access with the FATFS library. FileInfo handle for displaying filenames.

## 4.7 interface.h

[Go to the documentation of this file.](#)

```
00001
00021 #include <stdlib.h>
00022 #include <stdbool.h>
00023 #include "ssd1306.h"
00024 #include "fonts.h"
00025 #include "display.h"
00026 #include "filemanager.h"
00027 #include "main.h"
00028 #include "ff.h"
00029 #include "util.h"
00030
00038 #define THREASHOLD          0.1f
00039
00040 #define EXISTS              0
00041
00042 #define NOT_EXISTS         -1
00043
00044 #define SAME               0
00045
00046 #define NOT_SAME          -1
00047
00048 #define SMOOTHING_HEIGHT   30000
00049
00050
00051 //extern parameters from main
00052 //-----
00053
00054 extern ADC_HandleTypeDef hadc1;
00055 extern DMA_HandleTypeDef hdma_adc1;
00056 extern I2C_HandleTypeDef hi2c1;
00057 extern TIM_HandleTypeDef htim3;
00058 extern TIM_HandleTypeDef htim5;
00059
00060 extern FileManager fm;
00061 extern FileManager fmCopy;
00062
00063 extern FATFS FatFs;
00064 extern FILINFO fno;
00065 extern DIR dir;
00066 extern FRESULT fres;
00067 extern FIL file;
00068
00069
00070 extern uint32_t adcBuffer[NUM_CHANNELS];
00071 extern char fileNameSSDCard[MAX_FILES][MAX_FILENAME_LENGTH];
00072 extern int fileCount;
00073
00074 extern bool adcDmaFlag;
00075 extern bool updateScreenFlag;
```

```

00076 extern bool sortFilesFlag;
00077
00078 //-----
00079
00080 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc);
00081 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
00082 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);
00083
00084 void screenInit(char (*fileNames) [MAX_FILENAME_LENGTH]);
00085 void updateScreen(void);
00086 void sortFiles(void);
00087 void compareADCValues(void);
00088 void resetShownFiles(void);
00089 void writeFileManagerOnSD(void);
00090 void readFileManagerFromSD(void);
00091 void listFiles(const char *path);
00092

```

## 4.8 ssd1306.h

```

00001
00015 #ifndef _SSD1306_H
00016 #define _SSD1306_H
00017
00018 #include "stm32f4xx_hal.h"
00019 #include "fonts.h"
00020
00021 // I2c address
00022 #ifndef SSD1306_I2C_ADDR
00023 #define SSD1306_I2C_ADDR 0x78
00024 #endif // SSD1306_I2C_ADDR
00025
00026 // SSD1306 width in pixels
00027 #ifndef SSD1306_WIDTH
00028 #define SSD1306_WIDTH 128
00029 #endif // SSD1306_WIDTH
00030
00031 // SSD1306 LCD height in pixels
00032 #ifndef SSD1306_HEIGHT
00033 #define SSD1306_HEIGHT 128
00034 #endif // SSD1306_HEIGHT
00035
00036 #ifndef SSD1306_COM_LR_REMAP
00037 #define SSD1306_COM_LR_REMAP 0
00038 #endif // SSD1306_COM_LR_REMAP
00039
00040 #ifndef SSD1306_COM_ALTERNATIVE_PIN_CONFIG
00041 #define SSD1306_COM_ALTERNATIVE_PIN_CONFIG 1
00042 #endif // SSD1306_COM_ALTERNATIVE_PIN_CONFIG
00043
00044
00045 #define SH110X_MEMORYMODE 0x20
00046 #define SH110X_COLUMNADDR 0x21
00047 #define SH110X_PAGEADDR 0x22
00048 #define SH110X_SETCONTRAST 0x81
00049 #define SH110X_CHARGE_PUMP 0x8D
00050 #define SH110X_SEGREMAP 0xA0
00051 #define SH110X_DISPLAYALLON_RESUME 0xA4
00052 #define SH110X_DISPLAYALLON 0xA5
00053 #define SH110X_NORMALDISPLAY 0xA6
00054 #define SH110X_INVERTDISPLAY 0xA7
00055 #define SH110X_SETMULTIPLEX 0xA8
00056 #define SH110X_DCD 0xAD
00057 #define SH110X_DISPLAYOFF 0xAE
00058 #define SH110X_DISPLAYON 0xAF
00059 #define SH110X_SETPAGEADDR
00060 0xB0
00061
00062 #define SH110X_COMSCANINC 0xC0
00063 #define SH110X_COMSCANDEC 0xC8
00064 #define SH110X_SETDISPLAYOFFSET 0xD3
00065 #define SH110X_SETDISPLAYCLOCKDIV 0xD5
00066 #define SH110X_SETPRECHARGE 0xD9
00067 #define SH110X_SETCOMPINS 0xDA
00068 #define SH110X_SETVCOMDETECT 0xDB
00069 #define SH110X_SETDISPSTARTLINE
00070 0xDC
00071
00072
00073 #define SH110X_SETLOWCOLUMN 0x00
00074 #define SH110X_SETHIGHCOLUMN 0x10
00075 #define SH110X_SETSTARTLINE 0x40
00076
00077 //
00078 // Enumeration for screen colors

```

```

00079 //
00080 typedef enum {
00081     Black = 0x00,    // Black color, no pixel
00082     White = 0x01,    // Pixel is set. Color depends on LCD
00083 } SSD1306_COLOR;
00084
00085 //
00086 // Struct to store transformations
00087 //
00088 typedef struct {
00089     uint16_t CurrentX;
00090     uint16_t CurrentY;
00091     uint8_t Inverted;
00092     uint8_t Initialized;
00093 } SSD1306_t;
00094
00095 //
00096 // Function definitions
00097 //
00098
00099 uint8_t ssd1306_Init(I2C_HandleTypeDef *hi2c);
00100 void ssd1306_UpdateScreen(I2C_HandleTypeDef *hi2c);
00101 void ssd1306_Fill(SSD1306_COLOR color);
00102 void ssd1306_DrawPixel(uint8_t x, uint8_t y, SSD1306_COLOR color);
00103 char ssd1306_WriteChar(char ch, FontDef Font, SSD1306_COLOR color);
00104 char ssd1306_WriteString(const char* str, FontDef Font, SSD1306_COLOR color);
00105 void ssd1306_SetCursor(uint8_t x, uint8_t y);
00106 void ssd1306_InvertColors(void);
00107
00108 #endif // _SSD1306_H

```

## 4.9 f401\_display\_encoder\_fader\_test/Core/Src/display.c File Reference

: Display handling functions for OLED screen

```

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

```

### Macros

- #define **DISPLAY\_WIDTH** 128  
*Definitions for display dimensions and global variables for managing the visible portion of the list.*
- #define **DISPLAY\_HEIGHT** 128  
*Height of the display in pixels.*
- #define **LIST\_SECTION\_HEIGHT** 81  
*Height of the section that lists files in pixels.*
- #define **SELECTED\_FILE\_HEIGHT** 10  
*Height of the section that displays the selected file in pixels.*
- #define **BORDER\_WIDTH** 1  
*Width of the border around display sections in pixels.*
- #define **LINE\_HEIGHT** 10  
*Height of each line of text in pixels.*
- #define **CURSOR** '>'  
*Character used to indicate the cursor position.*
- #define **WAVEFORM\_TOP** 0  
*Top position for waveform display.*
- #define **WAVEFORM\_BOTTOM** (**LIST\_SECTION\_HEIGHT** - **BORDER\_WIDTH**)  
*Bottom position for waveform display.*
- #define **SAMPLE\_SKIP** 2  
*Number of samples to skip when drawing the waveform.*

## Functions

- void `displayStrings` (I2C\_HandleTypeDef \*hi2c1, char \*\*strings, uint8\_t numStrings, uint8\_t cursor\_index)  
*Displays a list of strings on the OLED screen with cursor highlighting.*
- void `scrollUp` ()  
*Scrolls the list up by one line.*
- void `scrollDown` (uint8\_t numStrings)  
*Scrolls the list down by one line.*
- void `renderSelectedFile` (I2C\_HandleTypeDef \*hi2c1, const char \*filename)  
*Renders the selected file name on the OLED screen.*
- void `drawLine` (int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1, uint16\_t color)
- void `drawFaderProzent` (I2C\_HandleTypeDef \*hi2c1, const char \*prozent, int multiplikator)  
*Draws the fader percentage on the OLED screen at a specified vertical position.*

### 4.9.1 Detailed Description

: Display handling functions for OLED screen

#### Attention

This file contains the definitions and functions for managing the OLED display, including displaying lists, handling cursor positions, and drawing various elements on the screen.

### 4.9.2 Macro Definition Documentation

#### 4.9.2.1 DISPLAY\_WIDTH

```
#define DISPLAY_WIDTH 128
```

Definitions for display dimensions and global variables for managing the visible portion of the list.

These definitions and variables control the layout and display sections of the OLED screen, as well as manage the portion of the file list that is visible on the screen. Width of the display in pixels

### 4.9.3 Function Documentation

#### 4.9.3.1 displayStrings()

```
void displayStrings (
    I2C_HandleTypeDef * hi2c1,
    char ** strings,
    uint8_t numStrings,
    uint8_t cursor_index)
```

Displays a list of strings on the OLED screen with cursor highlighting.

This function renders a list of strings on the OLED display, highlighting the current selection with a cursor. It also manages the display of strings based on the current cursor position and adjusts the visible portion of the list as needed.

- Clears the screen before drawing new content.
- Calculates the visible range of strings based on the `cursor_index` and ensures that the cursor is within the visible range.
- Draws each string on the display, with the current selection highlighted by a cursor.
- Draws a border around the list section of the screen.

**Parameters**

in	<i>hi2c1</i>	Pointer to the I2C handle used for communication with the OLED display.
in	<i>strings</i>	Array of string pointers to be displayed.
in	<i>numStrings</i>	Total number of strings in the array.
in	<i>cursor_index</i>	Index of the currently selected string, which will be highlighted on the display.

**Note**

Ensure that the `strings` array is correctly populated and `numStrings` reflects the actual number of valid strings. The display dimensions and font sizes should be configured to match the display hardware.

**4.9.3.2 drawFaderProzent()**

```
void drawFaderProzent (
    I2C_HandleTypeDef * hi2c1,
    const char * prozent,
    int multiplikator)
```

Draws the fader percentage on the OLED screen at a specified vertical position.

This function updates the display to show the fader percentage in a designated area of the screen. It first clears the region where the fader percentage will be displayed, and then writes the percentage at a vertical position determined by the `multiplikator`.

The `multiplikator` parameter is used to calculate the vertical offset from a base position on the screen. This allows for the display of multiple fader percentages at different vertical positions.

**Parameters**

in	<i>hi2c1</i>	Pointer to the I2C handle used for communication with the OLED display.
in	<i>prozent</i>	Pointer to a null-terminated string representing the fader percentage to be displayed.
in	<i>multiplikator</i>	Integer value used to adjust the vertical position on the screen, affecting where the percentage is drawn.

**Note**

Ensure that the `prozent` string is properly null-terminated and that the display dimensions and font settings are configured correctly to match the hardware. The `multiplikator` should be set appropriately to ensure that the displayed text does not overlap with other screen elements.

**4.9.3.3 renderSelectedFile()**

```
void renderSelectedFile (
    I2C_HandleTypeDef * hi2c1,
    const char * filename)
```

Renders the selected file name on the OLED screen.

This function updates the display to show the currently selected file name in a dedicated section of the screen. It first clears the area designated for displaying the selected file name, and then writes the file name at the appropriate position.

**Parameters**

in	<i>hi2c1</i>	Pointer to the I2C handle used for communication with the OLED display.
in	<i>filename</i>	Pointer to a null-terminated string representing the name of the selected file to be displayed.

**Note**

Ensure that the `filename` is properly null-terminated and that the display dimensions and font settings are configured correctly to match the hardware.

**4.9.3.4 scrollDown()**

```
void scrollDown (  
    uint8_t numStrings)
```

Scrolls the list down by one line.

This function updates the `first_visible_index` to scroll the displayed list downward by one line.

It ensures that the `first_visible_index` does not exceed the number of available strings (`numStrings`), preventing it from scrolling beyond the end of the list.

**Parameters**

in	<i>numStrings</i>	Total number of strings available in the list.
----	-------------------	--

**Note**

If scrolling down would cause the `first_visible_index` to go beyond the last item in the list, the function does nothing to avoid invalid index access.

**4.9.3.5 scrollUp()**

```
void scrollUp ()
```

Scrolls the list up by one line.

This function updates the `first_visible_index` to scroll the displayed list upward by one line. It ensures that the `first_visible_index` does not go below zero, which would be out of bounds for the visible portion of the list.

**Note**

If `first_visible_index` is already at the top of the list (i.e., 0), the function does nothing to avoid invalid index access.

## 4.10 f401\_display\_encoder\_fader\_test/Core/Src/filemanager.c File Reference

: [File](#) management functions for handling files and cursor operations

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "filemanager.h"
#include "display.h"
```

### Functions

- void [initializeFileManager](#) ([FileManager](#) \*fm)  
*Initializes default values for the [FileManager](#).*
- void [addFile](#) ([FileManager](#) \*fm, const char \*filename, const float \*classes)  
*Adds a file to the [FileManager](#).*
- void [cursorUp](#) ([FileManager](#) \*fm)  
*Decrements the cursor index in the [FileManager](#).*
- void [cursorDown](#) ([FileManager](#) \*fm)  
*Increments the cursor index in the [FileManager](#).*
- void [safeCurrentFileName](#) ([FileManager](#) \*fm)  
*Saves the filename at the current cursor index.*
- void [setCursor](#) ([FileManager](#) \*fm)  
*Sets the cursor position based on the currently displayed files in the [FileManager](#).*
- void [selectFile](#) ([FileManager](#) \*fm)  
*Sets the current file index to the cursor position.*

### 4.10.1 Detailed Description

: [File](#) management functions for handling files and cursor operations

#### Attention

This file contains the definitions and functions for managing files within the [FileManager](#) structure, including adding files, updating cursor positions, and handling file selection. It serves as the core for file management operations.

### 4.10.2 Function Documentation

#### 4.10.2.1 addFile()

```
void addFile (
    FileManager * fm,
    const char * filename,
    const float * classes)
```

Adds a file to the [FileManager](#).

This function adds a file to the `files[]` array within the [FileManager](#) structure. It associates the given file name with its corresponding class values and updates the file manager accordingly.



**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure to which the file will be added.
in	<i>filename</i>	The name of the file to be added to the <a href="#">FileManager</a> .
in	<i>classes</i>	Array of float values representing the file's class percentages.

**Note**

Ensure that the [FileManager](#) has sufficient capacity in the `files[]` array to accommodate new files.

**4.10.2.2 cursorDown()**

```
void cursorDown (  
    FileManager * fm)
```

Increments the cursor index in the [FileManager](#).

This function increments the `cursor_index` within the [FileManager](#) structure to move the cursor visually lower on the LCD display.

**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure whose <code>cursor_index</code> is to be incremented.
----	-----------	--

**4.10.2.3 cursorUp()**

```
void cursorUp (  
    FileManager * fm)
```

Decrements the cursor index in the [FileManager](#).

This function decrements the `cursor_index` within the [FileManager](#) structure to move the cursor visually higher on the LCD display.

**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure whose <code>cursor_index</code> is to be decremented.
----	-----------	--

**4.10.2.4 initializeFileManager()**

```
void initializeFileManager (  
    FileManager * fm)
```

Initializes default values for the [FileManager](#).

This function sets default values for the given [FileManager](#) structure. It initializes various fields to prepare the file manager for use.

**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure to be initialized.
----	-----------	---

**4.10.2.5 safeCurrentFileName()**

```
void safeCurrentFileName (
    FileManager * fm)
```

Saves the filename at the current cursor index.

This function copies the filename of the file currently pointed to by `cursor_index` in the [FileManager](#) structure to `current_cursor_filename`. It updates the `current_cursor_filename` with the filename from the `shownFiles` array at the position indicated by `cursor_index`.

**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure from which the filename is retrieved and stored.
----	-----------	---

**Note**

Ensure that `cursor_index` is within the valid range of indices in the `shownFiles` array to avoid out-of-bounds access. The `current_cursor_filename` should have sufficient space allocated to store the filename.

**4.10.2.6 selectFile()**

```
void selectFile (
    FileManager * fm)
```

Sets the current file index to the cursor position.

This function updates the `current_file_index` in the [FileManager](#) to the current position of the `cursor_index`. This allows the file at the cursor position to be marked as the selected file, which can then be processed or displayed as needed.

**Parameters**

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure in which the <code>current_file_index</code> will be set to the value of <code>cursor_index</code> .
----	-----------	---

#### 4.10.2.7 setCursor()

```
void setCursor (
    FileManager * fm)
```

Sets the cursor position based on the currently displayed files in the [FileManager](#).

This function adjusts the `cursor_index` in the [FileManager](#) based on the currently displayed files and their relevance to the fader settings. The behavior of the function is as follows:

- If no files match the fader settings (`num_matched_files` is 0):
  - The cursor is set to the start (index 0).
- If the `current_cursor_filename` matches one of the filenames in `shownFiles`:
  - The cursor is positioned at the index of the matching file in the `shownFiles` array.
- If the `current_cursor_filename` no longer exists in `shownFiles`:
  - The `current_cursor_filename` is updated to the filename at the position indicated by `current_file_index`.
- If `cursor_index` is larger than the number of matched files:
  - The `cursor_index` is set to the last position in the list.

##### Parameters

in	<i>fm</i>	Pointer to the <a href="#">FileManager</a> structure used to adjust the cursor position.
----	-----------	--

## 4.11 f401\_display\_encoder\_fader\_test/Core/Src/interface.c File Reference

: Interface functions for handling user inputs, file management, and display updates

```
#include "interface.h"
```

### Functions

- void [HAL\\_GPIO\\_EXTI\\_Callback](#) (uint16\_t GPIO\_Pin)  
*Callback function that responds to encoder signals.*
- void [HAL\\_TIM\\_PeriodElapsedCallback](#) (TIM\_HandleTypeDef \*htim)  
*Callback function triggered when a timer expires.*
- void [HAL\\_ADC\\_ConvCpltCallback](#) (ADC\_HandleTypeDef \*hadc)  
*Callback function for the ADC conversion.*
- void [screenInit](#) (char(\*fileNames)[MAX\_FILENAME\_LENGTH])  
*Initializes the screen and sets up the file manager with random class percentages.*
- void [sortFiles](#) (void)  
*Sorts filenames for display based on fader settings.*
- void [updateScreen](#) (void)

- *Updates the display with filenames, the currently selected file, and fader settings.*
- void `compareADCValues` (void)
- *Checks if fader settings have changed and updates accordingly.*
- void `resetShownFiles` (void)
- *Resets the `shownFiles` array and updates the number of matched files.*
- void `listFiles` (const char \*path)
- *Lists all filenames with a .wav extension from the specified directory on the SD card.*
- void `writeFileManagerOnSD` (void)
- void `readFileManagerFromSD` (void)

## Variables

- uint32\_t `rotary_enc_count` = 0
- *Global variables for debugging, debouncing, initialization, and managing fader settings.*
- bool `switch_push_button` = false
- *Boolean to verify if the encoder switch is pushed.*
- bool `debounce` = false
- *Flag to debounce the encoder.*
- bool `init` = false
- *Flag to check if initialization is done.*
- bool `sortFilesFlag` = false
- *Flag to regroup the shown file names.*
- uint16\_t `cnt`
- *Counter to smooth the fader values.*
- uint32\_t `smoothValue` [NUM\_CHANNELS] = {0}
- *Array to store smoothed values of the ADC buffer outputs.*
- uint8\_t `pastClassPercentADC` [NUM\_CHANNELS] = {0}
- *Array to cache the previous fader settings.*
- uint8\_t `currentClassPercentADC` [NUM\_CHANNELS] = {0}
- *Array to store the current fader settings in percentage.*
- char `faderProzent` [2][50]
- *Array to hold fader percentage strings for display on the LCD.*

### 4.11.1 Detailed Description

: Interface functions for handling user inputs, file management, and display updates

#### Attention

This file contains functions for handling user inputs through encoders and switches, updating the display, and managing file operations. Key functionalities include:

- Handling encoder signals and debouncing.
- Updating screen display based on file names and fader settings.
- Sorting files based on fader values.
- Managing file lists and file selection.
- Interacting with the ADC for smooth value calculations.

This file is essential for the interaction between the user interface and the file management system, ensuring that the display is updated with relevant information and user inputs are processed correctly.

## 4.11.2 Function Documentation

### 4.11.2.1 compareADCValues()

```
void compareADCValues (  
    void )
```

Checks if fader settings have changed and updates accordingly.

This function compares the current fader settings with previous values to determine if any changes have occurred. If changes are detected, it performs the following actions:

- Compares the current and past fader settings.
- Updates the past fader settings with the current values if a change is detected.
- Sets a flag to indicate that the file list needs to be sorted.
- Saves the current filename and clears the `shownFiles` array if a change is detected.

#### Note

This function uses global variables for fader settings and file management.

### 4.11.2.2 HAL\_ADC\_ConvCpltCallback()

```
void HAL_ADC_ConvCpltCallback (  
    ADC_HandleTypeDef * hadc)
```

Callback function for the ADC conversion.

This function is called when the ADC conversion is complete. It performs the following actions:

- Calculates the smooth values for all channels of the ADC.
- Computes the average values for display and comparison operations.
  - These average values are used for displaying on the screen and for the sorting algorithm's comparison operations.
- Initializes the character array that will be shown on the display.

#### Parameters

in	<i>hadc</i>	Pointer to the ADC handle structure.
----	-------------	--------------------------------------

#### 4.11.2.3 HAL\_GPIO\_EXTI\_Callback()

```
void HAL_GPIO_EXTI_Callback (
    uint16_t GPIO_Pin)
```

Callback function that responds to encoder signals.

This function handles encoder signals by adjusting the encoder count and cursor position on the display. It also debounces the encoder switch using Timer5 and a debounce flag.

- If A is High and B is High:
  - Decrement the encoder count and increment the cursor position on the display.
- If A is High and B is Low:
  - Increment the encoder count and decrement the cursor position on the display.
- If the switch is pushed:
  - Select the file and debounce the encoder switch using Timer5 and the debounce flag.

##### Parameters

in	<i>GPIO_pin</i>	The GPIO pin signals from the MCU related to the encoder and switch.
----	-----------------	--

#### 4.11.2.4 HAL\_TIM\_PeriodElapsedCallback()

```
void HAL_TIM_PeriodElapsedCallback (
    TIM_HandleTypeDef * htim)
```

Callback function triggered when a timer expires.

This function handles timer expirations by performing the following actions:

For Timer5 (tim5):

- Resets the debounce flag after Timer5 has elapsed.
- Stops Timer5.

For Timer3 (tim3):

- Sets flags to indicate DMA completion and screen update.
- Sets a new timer for the next refresh interval.

##### Parameters

in	<i>htim</i>	Pointer to the TIM_HandleTypeDef structure for the current timer.
----	-------------	---

#### 4.11.2.5 listFiles()

```
void listFiles (
    const char * path)
```

Lists all filenames with a .wav extension from the specified directory on the SD card.

This function opens the directory specified by the `path` parameter, reads through its contents, and stores filenames with a .wav extension in the `fileNamesSDCard` array. It performs the following actions:

- Opens the directory at the specified path.
- Iterates through directory entries until reaching the maximum file count or encountering an end-of-directory condition.
- Checks if each entry is a file (not a directory) and if it has a .wav extension.
- Stores valid filenames in the `fileNamesSDCard` array.
- Closes the directory after reading all entries.

##### Parameters

in	<i>path</i>	The directory path on the SD card to search for files.
----	-------------	--

##### Note

This function assumes that `fileNamesSDCard` and `fileCount` are global variables and that `MAX_FILES` and `MAX_FILENAME_LENGTH` are defined constants.

#### 4.11.2.6 resetShownFiles()

```
void resetShownFiles (
    void )
```

Resets the `shownFiles` array and updates the number of matched files.

This function sets all entries in the `shownFiles` array to zero and resets the count of matched files.

- Sets the memory area of `fm.shownFiles` to zero using `memset`.
- Resets the `num_matched_files` field to 0.

##### Note

This function is used to clear the list of files currently displayed on the screen.

#### 4.11.2.7 screenInit()

```
void screenInit (
    char(*) fileNames[MAX_FILENAME_LENGTH])
```

Initializes the screen and sets up the file manager with random class percentages.

This function initializes the LCD-Display, fills it with black, and updates the screen. It also initializes the file manager with file names and assigns random class percentages for system tests.

**Parameters**

in	<i>fileNames</i>	Array of file names to be added to the file manager.
----	------------------	--

**4.11.2.8 sortFiles()**

```
void sortFiles (  
    void )
```

Sorts filenames for display based on fader settings.

This function sorts filenames by comparing file classes with the fader settings. It performs the following actions:

- Compares file classes against the fader settings with a threshold of 0.1.
- Compares the newly added filenames with the existing ones.
- Adds files from the existing file list to the `shownFiles` in the file manager.

**Parameters**

in	<i>void</i>	
----	-------------	--

**4.11.2.9 updateScreen()**

```
void updateScreen (  
    void )
```

Updates the display with filenames, the currently selected file, and fader settings.

This function updates the display by performing the following tasks:

- Extracts filenames from the file manager's `shownFiles` array.
- Sets the current filename to be displayed based on the initialization state.
- Displays all filenames in the `shownFiles` array.
- Display the currently selected filename.
- Shows the fader settings on the display.

**Note**

The function relies on global variables and external functions to interact with the display.



### 4.11.3 Variable Documentation

#### 4.11.3.1 rotary\_enc\_count

```
uint32_t rotary_enc_count = 0
```

Global variables for debugging, debouncing, initialization, and managing fader settings.

These variables are used for various purposes throughout the program, including debugging, debouncing encoder inputs, initializing states, sorting file names, smoothing ADC values, and displaying fader settings on the LCD. Count variable for debugging purposes.

## 4.12 f401\_sd\_card\_audio\_codec\_test/Core/Inc/audio.h File Reference

Header file for audio playback and processing functions.

```
#include <stdbool.h>
#include <math.h>
#include "main.h"
#include "fatfs.h"
#include "stm32f4xx_hal_i2s_ex.h"
#include "wavPlayer.h"
```

### Macros

- **#define BUFFER\_SIZE 256**  
*Size of the audio buffer.*
- **#define HALF\_BUFFER\_SIZE (BUFFER\_SIZE / 2)**  
*Size of half of the audio buffer.*
- **#define BUFFER\_SIZE\_INPUT 4096**  
*Size of the input buffer for reading from the file.*

### Functions

- void **HAL\_I2S\_TxCpltCallback** (I2S\_HandleTypeDef \*hi2s)  
*Called when the first half of the I2S buffer is transmitted.*
- void **HAL\_I2SS\_TxCpltCallback** (I2S\_HandleTypeDef \*hi2s)
- uint16\_t **fillHalfBufferFromSD** (struct WavPlayer \*player, bool pitched)  
*Fills half of the buffer with audio samples from an SD file.*
- void **generateSineWave** (double frequency)  
*Populates half of the DAC buffer with a sine wave using a lookup table.*
- void **initSineTable** ()
- void **setPitchFactor** (float newPitchFactor)  
*Sets a new pitch factor for audio playback.*
- int16\_t **interpolate** (int16\_t \*buffer, float index)

## Variables

- I2S\_HandleTypeDef **hi2s2**
- int16\_t **dacData** [BUFFER\_SIZE]
- int16\_t **fileReadBuf** [BUFFER\_SIZE\_INPUT]
- bool **dma\_dataReady**
- volatile int16\_t \* **outBufPtr**
- volatile float **pitchFactor**
- volatile bool **pitchChanged**

### 4.12.1 Detailed Description

Header file for audio playback and processing functions.

### 4.12.2 Function Documentation

#### 4.12.2.1 fillHalfBufferFromSD()

```
uint16_t fillHalfBufferFromSD (  
    WavPlayer * player,  
    bool pitched)
```

Fills half of the buffer with audio samples from an SD file.

The function reads audio samples from an SD card file into a buffer. If the `pitched` parameter is true, it adjusts the number of samples to account for pitch shifting. It ensures the number of samples is even and reads the data into `fileReadBuf`.

#### Parameters

<i>player</i>	Pointer to a <a href="#">WavPlayer</a> structure containing file information and pitch factor.
<i>pitched</i>	Flag indicating whether pitch shifting should be applied.

#### Returns

The number of bytes actually read from the file.

#### 4.12.2.2 generateSineWave()

```
void generateSineWave (  
    double frequency)
```

Populates half of the DAC buffer with a sine wave using a lookup table.

The function generates a sine wave based on the provided frequency. It uses a lookup table to fill the DAC buffer with sine values and manages the phase index for the waveform. It waits for the DMA to be ready before updating the buffer.

## Parameters

<i>frequency</i>	Desired frequency of the sine wave.
------------------	-------------------------------------

< Retains the index between function calls

#### 4.12.2.3 HAL\_I2S\_TxHalfCpltCallback()

```
void HAL_I2S_TxHalfCpltCallback (
    I2S_HandleTypeDef * hi2s)
```

Called when the first half of the I2S buffer is transmitted.

Updates the buffer pointer to the start of `dacData` and sets the DMA data ready flag.

## Parameters

<i>hi2s</i>	Pointer to the I2S handle structure.
-------------	--------------------------------------

#### 4.12.2.4 setPitchFactor()

```
void setPitchFactor (
    float newPitchFactor)
```

Sets a new pitch factor for audio playback.

Updates the global pitch factor and marks it as changed. Interrupts are temporarily disabled to ensure thread safety during the update.

## Parameters

<i>newPitchFactor</i>	The new pitch factor to set.
-----------------------	------------------------------

## 4.13 audio.h

[Go to the documentation of this file.](#)

```
00001
00006 #ifndef INC_AUDIO_H_
00007 #define INC_AUDIO_H_
00008
00009 #include <stdbool.h>
00010 #include <math.h>
00011 #include "main.h"
00012 #include "fatfs.h"
00013 #include "stm32f4xx_hal_i2s_ex.h"
00014 #include "wavPlayer.h"
00015
00016 #ifndef M_PI
00017 #define M_PI 3.14159265358979323846
00018 #endif
00019
00023 #define BUFFER_SIZE 256
00024
00028 #define HALF_BUFFER_SIZE (BUFFER_SIZE / 2)
00029
```

```

00033 #define BUFFER_SIZE_INPUT 4096
00034
00035 extern I2S_HandleTypeDef hi2s2;
00036 extern int16_t dacData[BUFFER_SIZE];
00037 extern int16_t fileReadBuf[BUFFER_SIZE_INPUT]; // actually for one halfBuffer, now is = BUFFER_SIZE,
        to allow 2x playback speed
00038 extern bool dma_dataReady;
00039 extern volatile int16_t *outBufPtr;
00040
00041 extern volatile float pitchFactor;
00042 extern volatile bool pitchChanged;
00043
00044 struct WavPlayer;
00045
00046 // Callbacks for DMA Complete
00047 void HAL_I2S_TxHalfCpltCallback(I2S_HandleTypeDef *hi2s);
00048 void HAL_I2S_TxCpltCallback(I2S_HandleTypeDef *hi2s);
00049
00050 // fills half of the buffer with next chunk of data from file
00051 // returns the number of bytes read
00052 uint16_t fillHalfBufferFromSD(struct WavPlayer *player, bool pitched);
00053
00054 // Test Functions
00055 void generateSineWave(double frequency);
00056 void initSineTable();
00057
00058 // Pitch functions
00059 void setPitchFactor(float newPitchFactor);
00060
00061 int16_t interpolate(int16_t *buffer, float index);
00062
00063 #endif /* INC_AUDIO_H_ */

```

## 4.14 f401\_sd\_card\_audio\_codec\_test/Core/Inc/audioPreprocessor.h File Reference

Header file for the audio preprocessing DSP-Operations.

```

#include <stdint.h>
#include "arm_math.h"
#include "fatfs.h"
#include "lowpass_16kFilter.h"
#include "wavPlayer.h"

```

### Macros

- **#define DECIMATION\_FACTOR\_ROUNDED 3**  
*Decimation factor rounded up from 2.8 to 3.0.*
- **#define NUM\_CHANNELS 2**
- **#define NUM\_SAMPLES\_CHUNK\_OUT 1024**  
*Number of samples per chunk for output processing.*
- **#define NUM\_SAMPLES\_CHUNK\_IN ((NUM\_SAMPLES\_CHUNK\_OUT \* DECIMATION\_FACTOR\_ROUNDED \* NUM\_CHANNELS + 1) & ~1)**  
*Number of samples per chunk for input processing, adjusted to be even for stereo.*
- **#define RESAMPLE\_CHUNK\_IN\_SIZE (NUM\_SAMPLES\_CHUNK\_IN \* sizeof(int16\_t))**  
*Byte size of the input chunk for resampling.*
- **#define RESAMPLE\_CHUNK\_OUT\_SIZE (NUM\_SAMPLES\_CHUNK\_OUT \* sizeof(int16\_t))**  
*Byte size of the output chunk for resampling.*
- **#define FRAME\_SIZE 1024**  
*Size of the window frame for processing.*
- **#define HOP\_SIZE 512**  
*Overlap size between consecutive frames.*
- **#define STEP\_SIZE (FRAME\_SIZE - HOP\_SIZE)**  
*Step size between frames, calculated as the difference between frame size and overlap size.*

## Functions

- arm\_status [initFilter](#) ()  
*Initializes the FIR decimation filter.*
- uint16\_t [resampleFile](#) (FIL \*inFil, FIL \*outFil, [wav\\_header\\_t](#) \*waveHeaderInput, [wav\\_header\\_t](#) \*waveHeaderResampled)  
*Resamples audio data from an input file and writes to an output file.*
- void [downsample\\_Block](#) (int16\_t \*src, int16\_t \*dest)  
*Performs downsampling and FIR filtering on a block of audio data.*
- uint16\_t [get\\_number\\_subsamples](#) (FIL \*file)  
*Calculates the number of subsamples based on the WAV file data.*
- uint32\_t [downsample\\_to\\_1024\\_samples](#) (FIL \*file, int16\_t outChunk[[NUM\\_SAMPLES\\_CHUNK\\_OUT](#)])  
*Downsamples a chunk of audio data to 1024 samples.*

## Variables

- float32\_t [filter\\_taps](#) [[NUM\\_TAPS](#)]  
*Array of filter coefficients for the FIR filter.*

### 4.14.1 Detailed Description

Header file for the audio preprocessing DSP-Operations.

### 4.14.2 Function Documentation

#### 4.14.2.1 downsample\_Block()

```
void downsample_Block (
    int16_t * src,
    int16_t * dest)
```

Performs downsampling and FIR filtering on a block of audio data.

Converts a block of 16-bit integer audio samples to floating-point format, applies FIR filtering and decimation, and then converts the filtered samples back to 16-bit integer format. The processed data is written to the destination buffer.

#### Parameters

<i>src</i>	Pointer to the source buffer containing 16-bit audio samples to be processed.
<i>dest</i>	Pointer to the destination buffer where the downsampled 16-bit audio samples will be written.

#### 4.14.2.2 downsample\_to\_1024\_samples()

```
uint32_t downsample_to_1024_samples (
    FIL * file,
    int16_t outChunk[NUM\_SAMPLES\_CHUNK\_OUT])
```

Downsamples a chunk of audio data to 1024 samples.

Reads a chunk of audio data from the specified file, converts stereo samples to mono, and then performs downsampling. The processed samples are written to the `outChunk` buffer. The function processes data in blocks and returns the number of downsampled samples. `NUM_SAMPLES_CHUNK_OUT * DECIMATION_FACTOR_ROUNDROUNDED = NUM_SAMPLES_CHUNK_IN`

**Parameters**

<i>file</i>	Pointer to the input file ( <code>FIL</code> structure) containing audio data.
<i>outChunk</i>	Array to store the downsampled audio samples.

**Returns**

The actual number of samples written to the `outChunk` buffer.

**4.14.2.3 get\_number\_subsamples()**

```
uint16_t get_number_subsamples (
    FIL * file)
```

Calculates the number of subsamples based on the WAV file data.

Determines the number of mono subsamples in a WAV file after applying downsampling. It calculates the total number of samples based on the file's audio data size, decimation factor, and header information, then computes the number of frames using the `HOP_SIZE` and `STEP_SIZE`.

**Parameters**

<i>file</i>	Pointer to the WAV file ( <code>FIL</code> structure) from which the header information is read.
-------------	--

**Returns**

The number of frames after downsampling, as a `uint16_t`.

**4.14.2.4 initFilter()**

```
arm_status initFilter ()
```

Initializes the FIR decimation filter.

Configures the FIR decimation filter using the ARM CMSIS DSP library. Sets up the filter structure with the specified number of taps, decimation factor, filter coefficients, state buffer, and block size.

**Returns**

Status of the filter initialization (`arm_status`).

**4.14.2.5 resampleFile()**

```
uint16_t resampleFile (
    FIL * inFil,
    FIL * outFil,
    wav_header_t * waveHeaderInput,
    wav_header_t * waveHeaderResampled)
```

Resamples audio data from an input file and writes to an output file.

Reads chunks of audio data from the input file, performs downsampling and resampling, and writes the processed data to the output file. Updates the WAV header of the output file to reflect the new format after resampling.

## Parameters

<i>inFil</i>	Pointer to the input file (FIL structure) containing original audio data.
<i>outFil</i>	Pointer to the output file (FIL structure) to write resampled audio data.
<i>waveHeaderInput</i>	Pointer to the WAV header of the input file.
<i>waveHeaderResampled</i>	Pointer to the WAV header to be written to the output file.

## Returns

The number of bytes written to the resampled file.

### 4.14.3 Variable Documentation

#### 4.14.3.1 filter\_taps

```
float32_t filter_taps[NUM_TAPS] [extern]
```

Array of filter coefficients for the FIR filter.

Holds the coefficients used in the FIR filter. The size of the array is determined by NUM\_TAPS, which specifies the number of filter taps. These coefficients are used for filtering audio or signal data.

## 4.15 audioPreprocessor.h

[Go to the documentation of this file.](#)

```
00001
00006 #ifndef RESAMPLE_H
00007 #define RESAMPLE_H
00008
00009 #include <stdint.h>
00010 #include "arm_math.h"
00011 #include "fatfs.h"
00012 #include "lowpass_16kFilter.h"
00013 #include "wavPlayer.h"
00014
00015 // Constants
00019 #define DECIMATION_FACTOR_ROUNDED 3 // Rounded up from 44.1kHz / 16kHz = 2.75625
00020 #define NUM_CHANNELS 2
00021
00025 #define NUM_SAMPLES_CHUNK_OUT 1024
00029 #define NUM_SAMPLES_CHUNK_IN ((NUM_SAMPLES_CHUNK_OUT * DECIMATION_FACTOR_ROUNDED * NUM_CHANNELS + 1) &
~1)
00033 #define RESAMPLE_CHUNK_IN_SIZE (NUM_SAMPLES_CHUNK_IN * sizeof(int16_t))
00037 #define RESAMPLE_CHUNK_OUT_SIZE (NUM_SAMPLES_CHUNK_OUT * sizeof(int16_t))
00041 #define FRAME_SIZE 1024
00045 #define HOP_SIZE 512
00049 #define STEP_SIZE (FRAME_SIZE - HOP_SIZE)
00050 // External filter coefficients (defined in your filter file)
00051 extern float32_t filter_taps[NUM_TAPS];
00052
00053 // Function declarations
00054 arm_status initFilter();
00055 uint16_t resampleFile(FIL *inFil, FIL *outFil, wav_header_t *waveHeaderInput, wav_header_t
*waveHeaderResampled);
00056 void downsample_Block(int16_t *src, int16_t *dest);
00057 uint16_t get_number_subsamples(FIL *file);
00058 uint32_t downsample_to_1024_samples(FIL *file, int16_t outChunk[NUM_SAMPLES_CHUNK_OUT]);
00059
00060 #endif // RESAMPLE_H
```

## 4.16 cpu\_time.h

```

00001 #ifndef CPU_TIME_H
00002 #define CPU_TIME_H
00003
00004 #include <stdint.h>
00005
00006
00007
00008 // enable the DWT cycle counter
00009 void EnableDWT(void);
00010
00011 // Functions to start and stop cycle measurement
00012 void StartCycleMeasurement(void);
00013 void StopCycleMeasurement(void);
00014
00015 // Function to get the measured cycle count
00016 uint32_t GetMeasuredCycles(void);
00017
00018 // Function to convert cycles to milliseconds
00019 uint32_t CyclesToMilliseconds(uint32_t cycles);
00020
00021 // Function to convert cycles to microseconds
00022 uint32_t CyclesToMicroseconds(uint32_t cycles);
00023
00024 #endif

```

## 4.17 f401\_sd\_card\_audio\_codec\_test/Core/Inc/lowpass\_16kFilter.h File Reference

Header file for the low-pass FIR filter used for decimation.

```
#include "arm_math.h"
```

### Macros

- **#define NUM\_TAPS 57**  
*Number of taps in the FIR filter.*
- **#define BLOCK\_SIZE 33**  
*Size of each block for processing, must be a multiple of the decimation factor.*

### Variables

- float32\_t **filter\_taps** [NUM\_TAPS]  
*Array of filter coefficients for the FIR filter.*

### 4.17.1 Detailed Description

Header file for the low-pass FIR filter used for decimation.

Defines the number of taps and block size for the FIR filter. Includes the external filter coefficients and necessary header files for ARM CMSIS-DSP functions.



## 4.17.2 Variable Documentation

### 4.17.2.1 filter\_taps

```
float32_t filter_taps[NUM_TAPS] [extern]
```

Array of filter coefficients for the FIR filter.

Holds the coefficients used in the FIR filter. The size of the array is determined by NUM\_TAPS, which specifies the number of filter taps. These coefficients are used for filtering audio or signal data.

## 4.18 lowpass\_16kFilter.h

[Go to the documentation of this file.](#)

```
00001
00008 #ifndef LOWPASS_16KFILTER_H_
00009 #define LOWPASS_16KFILTER_H_
00010
00011 #include "arm_math.h"
00012
00013
00017 #define NUM_TAPS 57
00018
00022 #define BLOCK_SIZE 33
00023
00024 extern float32_t filter_taps[NUM_TAPS];
00025
00026 #endif
```

## 4.19 f401\_display\_encoder\_fader\_test/Core/Inc/main.h File Reference

: Header for main.c file. This file contains the common defines of the application.

```
#include "stm32f4xx_hal.h"
```

### Macros

- #define **B1\_Pin** GPIO\_PIN\_13
- #define **B1\_GPIO\_Port** GPIOC
- #define **B1\_EXTI\_IRQn** EXTI15\_10\_IRQn
- #define **FADER\_IN5\_Pin** GPIO\_PIN\_0
- #define **FADER\_IN5\_GPIO\_Port** GPIOC
- #define **enc\_a\_clk\_in1\_Pin** GPIO\_PIN\_0
- #define **enc\_a\_clk\_in1\_GPIO\_Port** GPIOA
- #define **enc\_a\_clk\_in1\_EXTI\_IRQn** EXTI0\_IRQn
- #define **enc\_b\_dt\_in2\_Pin** GPIO\_PIN\_1
- #define **enc\_b\_dt\_in2\_GPIO\_Port** GPIOA
- #define **USART\_TX\_Pin** GPIO\_PIN\_2
- #define **USART\_TX\_GPIO\_Port** GPIOA
- #define **USART\_RX\_Pin** GPIO\_PIN\_3
- #define **USART\_RX\_GPIO\_Port** GPIOA
- #define **enc\_switch\_in3\_Pin** GPIO\_PIN\_4
- #define **enc\_switch\_in3\_GPIO\_Port** GPIOA

- `#define enc_switch_in3_EXTI_IRQn EXTI4_IRQn`
- `#define LD2_Pin GPIO_PIN_5`
- `#define LD2_GPIO_Port GPIOA`
- `#define FADER_IN1_Pin GPIO_PIN_6`
- `#define FADER_IN1_GPIO_Port GPIOA`
- `#define FADER_IN2_Pin GPIO_PIN_7`
- `#define FADER_IN2_GPIO_Port GPIOA`
- `#define FADER_IN3_Pin GPIO_PIN_0`
- `#define FADER_IN3_GPIO_Port GPIOB`
- `#define FADER_IN4_Pin GPIO_PIN_1`
- `#define FADER_IN4_GPIO_Port GPIOB`
- `#define TMS_Pin GPIO_PIN_13`
- `#define TMS_GPIO_Port GPIOA`
- `#define TCK_Pin GPIO_PIN_14`
- `#define TCK_GPIO_Port GPIOA`
- `#define SWO_Pin GPIO_PIN_3`
- `#define SWO_GPIO_Port GPIOB`
- `#define NUM_CHANNELS 5`

## Functions

- void [Error\\_Handler](#) (void)

*This function is executed in case of error occurrence.*

### 4.19.1 Detailed Description

: Header for main.c file. This file contains the common defines of the application.

#### Attention

Copyright (c) 2024 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.

### 4.19.2 Function Documentation

#### 4.19.2.1 Error\_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

#### Return values

None	
------	--

## 4.20 main.h

[Go to the documentation of this file.](#)

```

00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Define to prevent recursive inclusion -----*/
00022 #ifndef __MAIN_H
00023 #define __MAIN_H
00024
00025 #ifdef __cplusplus
00026 extern "C" {
00027 #endif
00028
00029 /* Includes -----*/
00030 #include "stm32f4xx_hal.h"
00031
00032 /* Private includes -----*/
00033 /* USER CODE BEGIN Includes */
00034
00035 /* USER CODE END Includes */
00036
00037 /* Exported types -----*/
00038 /* USER CODE BEGIN ET */
00039
00040 /* USER CODE END ET */
00041
00042 /* Exported constants -----*/
00043 /* USER CODE BEGIN EC */
00044
00045 /* USER CODE END EC */
00046
00047 /* Exported macro -----*/
00048 /* USER CODE BEGIN EM */
00049
00050 /* USER CODE END EM */
00051
00052 /* Exported functions prototypes -----*/
00053 void Error_Handler(void);
00054
00055 /* USER CODE BEGIN EFP */
00056
00057 /* USER CODE END EFP */
00058
00059 /* Private defines -----*/
00060 #define B1_Pin GPIO_PIN_13
00061 #define B1_GPIO_Port GPIOC
00062 #define B1_EXTI_IRQn EXTI15_10_IRQn
00063 #define FADER_IN5_Pin GPIO_PIN_0
00064 #define FADER_IN5_GPIO_Port GPIOC
00065 #define enc_a_clk_in1_Pin GPIO_PIN_0
00066 #define enc_a_clk_in1_GPIO_Port GPIOA
00067 #define enc_a_clk_in1_EXTI_IRQn EXTI0_IRQn
00068 #define enc_b_dt_in2_Pin GPIO_PIN_1
00069 #define enc_b_dt_in2_GPIO_Port GPIOA
00070 #define USART_TX_Pin GPIO_PIN_2
00071 #define USART_TX_GPIO_Port GPIOA
00072 #define USART_RX_Pin GPIO_PIN_3
00073 #define USART_RX_GPIO_Port GPIOA
00074 #define enc_switch_in3_Pin GPIO_PIN_4
00075 #define enc_switch_in3_GPIO_Port GPIOA
00076 #define enc_switch_in3_EXTI_IRQn EXTI4_IRQn
00077 #define LD2_Pin GPIO_PIN_5
00078 #define LD2_GPIO_Port GPIOA
00079 #define FADER_IN1_Pin GPIO_PIN_6
00080 #define FADER_IN1_GPIO_Port GPIOA
00081 #define FADER_IN2_Pin GPIO_PIN_7
00082 #define FADER_IN2_GPIO_Port GPIOA
00083 #define FADER_IN3_Pin GPIO_PIN_0
00084 #define FADER_IN3_GPIO_Port GPIOB
00085 #define FADER_IN4_Pin GPIO_PIN_1
00086 #define FADER_IN4_GPIO_Port GPIOB
00087 #define TMS_Pin GPIO_PIN_13
00088 #define TMS_GPIO_Port GPIOA
00089 #define TCK_Pin GPIO_PIN_14
00090 #define TCK_GPIO_Port GPIOA
00091 #define SWO_Pin GPIO_PIN_3
00092 #define SWO_GPIO_Port GPIOB
00093
00094 /* USER CODE BEGIN Private defines */
00095 #define NUM_CHANNELS 5
00096 /* USER CODE END Private defines */
00097
00098 #ifdef __cplusplus
00099 }
```

```
00100 #endif
00101
00102 #endif /* __MAIN_H */
```

## 4.21 f401\_sd\_card\_audio\_codec\_test/Core/Inc/main.h File Reference

: Header for main.c file. This file contains the common defines of the application.

```
#include "stm32f4xx_hal.h"
```

### Macros

- #define **B1\_Pin** GPIO\_PIN\_13
- #define **B1\_GPIO\_Port** GPIOC
- #define **B1\_EXTI\_IRQn** EXTI15\_10\_IRQn
- #define **USART\_TX\_Pin** GPIO\_PIN\_2
- #define **USART\_TX\_GPIO\_Port** GPIOA
- #define **USART\_RX\_Pin** GPIO\_PIN\_3
- #define **USART\_RX\_GPIO\_Port** GPIOA
- #define **SD\_CS\_Pin** GPIO\_PIN\_1
- #define **SD\_CS\_GPIO\_Port** GPIOB
- #define **TMS\_Pin** GPIO\_PIN\_13
- #define **TMS\_GPIO\_Port** GPIOA
- #define **TCK\_Pin** GPIO\_PIN\_14
- #define **TCK\_GPIO\_Port** GPIOA
- #define **SWO\_Pin** GPIO\_PIN\_3
- #define **SWO\_GPIO\_Port** GPIOB
- #define **SD\_SPI\_HANDLE** hspi2

### Functions

- void [Error\\_Handler](#) (void)  
*This function is executed in case of error occurrence.*

### 4.21.1 Detailed Description

: Header for main.c file. This file contains the common defines of the application.

#### Attention

Copyright (c) 2024 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.

### 4.21.2 Function Documentation

#### 4.21.2.1 Error\_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

## Return values

None	
------	--

## 4.22 main.h

[Go to the documentation of this file.](#)

```

00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Define to prevent recursive inclusion -----*/
00022 #ifndef __MAIN_H
00023 #define __MAIN_H
00024
00025 #ifdef __cplusplus
00026 extern "C" {
00027 #endif
00028
00029 /* Includes -----*/
00030 #include "stm32f4xx_hal.h"
00031
00032 /* Private includes -----*/
00033 /* USER CODE BEGIN Includes */
00034
00035 /* USER CODE END Includes */
00036
00037 /* Exported types -----*/
00038 /* USER CODE BEGIN ET */
00039
00040 /* USER CODE END ET */
00041
00042 /* Exported constants -----*/
00043 /* USER CODE BEGIN EC */
00044
00045 /* USER CODE END EC */
00046
00047 /* Exported macro -----*/
00048 /* USER CODE BEGIN EM */
00049
00050 /* USER CODE END EM */
00051
00052 /* Exported functions prototypes -----*/
00053 void Error_Handler(void);
00054
00055 /* USER CODE BEGIN EFP */
00056
00057 /* USER CODE END EFP */
00058
00059 /* Private defines -----*/
00060 #define B1_Pin GPIO_PIN_13
00061 #define B1_GPIO_Port GPIOC
00062 #define B1_EXTI_IRQn EXTI15_10_IRQn
00063 #define USART_TX_Pin GPIO_PIN_2
00064 #define USART_TX_GPIO_Port GPIOA
00065 #define USART_RX_Pin GPIO_PIN_3
00066 #define USART_RX_GPIO_Port GPIOA
00067 #define SD_CS_Pin GPIO_PIN_1
00068 #define SD_CS_GPIO_Port GPIOB
00069 #define TMS_Pin GPIO_PIN_13
00070 #define TMS_GPIO_Port GPIOA
00071 #define TCK_Pin GPIO_PIN_14
00072 #define TCK_GPIO_Port GPIOA
00073 #define SWO_Pin GPIO_PIN_3
00074 #define SWO_GPIO_Port GPIOB
00075
00076 /* USER CODE BEGIN Private defines */
00077 #define SD_SPI_HANDLE hspi2
00078 /* USER CODE END Private defines */
00079
00080 #ifdef __cplusplus
00081 }
00082 #endif
00083
00084 #endif /* __MAIN_H */

```

## 4.23 neuronal\_network/esp\_cubeai\_test/Core/Inc/main.h File Reference

: Header for main.c file. This file contains the common defines of the application.

```
#include "stm32f7xx_hal.h"
#include <arm_math.h>
```

### Macros

- #define **USER\_Btn\_Pin** GPIO\_PIN\_13
- #define **USER\_Btn\_GPIO\_Port** GPIOC
- #define **MCO\_Pin** GPIO\_PIN\_0
- #define **MCO\_GPIO\_Port** GPIOH
- #define **LD1\_Pin** GPIO\_PIN\_0
- #define **LD1\_GPIO\_Port** GPIOB
- #define **LD3\_Pin** GPIO\_PIN\_14
- #define **LD3\_GPIO\_Port** GPIOB
- #define **STLK\_RX\_Pin** GPIO\_PIN\_8
- #define **STLK\_RX\_GPIO\_Port** GPIOD
- #define **STLK\_TX\_Pin** GPIO\_PIN\_9
- #define **STLK\_TX\_GPIO\_Port** GPIOD
- #define **USB\_PowerSwitchOn\_Pin** GPIO\_PIN\_6
- #define **USB\_PowerSwitchOn\_GPIO\_Port** GPIOG
- #define **USB\_OverCurrent\_Pin** GPIO\_PIN\_7
- #define **USB\_OverCurrent\_GPIO\_Port** GPIOG
- #define **USB\_SOF\_Pin** GPIO\_PIN\_8
- #define **USB\_SOF\_GPIO\_Port** GPIOA
- #define **USB\_VBUS\_Pin** GPIO\_PIN\_9
- #define **USB\_VBUS\_GPIO\_Port** GPIOA
- #define **USB\_ID\_Pin** GPIO\_PIN\_10
- #define **USB\_ID\_GPIO\_Port** GPIOA
- #define **USB\_DM\_Pin** GPIO\_PIN\_11
- #define **USB\_DM\_GPIO\_Port** GPIOA
- #define **USB\_DP\_Pin** GPIO\_PIN\_12
- #define **USB\_DP\_GPIO\_Port** GPIOA
- #define **TMS\_Pin** GPIO\_PIN\_13
- #define **TMS\_GPIO\_Port** GPIOA
- #define **TCK\_Pin** GPIO\_PIN\_14
- #define **TCK\_GPIO\_Port** GPIOA
- #define **SWO\_Pin** GPIO\_PIN\_3
- #define **SWO\_GPIO\_Port** GPIOB
- #define **LD2\_Pin** GPIO\_PIN\_7
- #define **LD2\_GPIO\_Port** GPIOB

### Functions

- void [Error\\_Handler](#) (void)  
*This function is executed in case of error occurrence.*

### 4.23.1 Detailed Description

: Header for main.c file. This file contains the common defines of the application.

#### Attention

Copyright (c) 2024 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.

### 4.23.2 Function Documentation

#### 4.23.2.1 Error\_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

#### Return values

None	
------	--

## 4.24 main.h

[Go to the documentation of this file.](#)

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Define to prevent recursive inclusion -----*/
00022 #ifndef __MAIN_H
00023 #define __MAIN_H
00024
00025 #ifdef __cplusplus
00026 extern "C" {
00027 #endif
00028
00029 /* Includes -----*/
00030 #include "stm32f7xx_hal.h"
00031
00032 /* Private includes -----*/
00033 /* USER CODE BEGIN Includes */
00034 #include <arm_math.h>
00035 /* USER CODE END Includes */
00036
00037 /* Exported types -----*/
00038 /* USER CODE BEGIN ET */
00039
00040 /* USER CODE END ET */
00041
00042 /* Exported constants -----*/
00043 /* USER CODE BEGIN EC */
00044
00045 /* USER CODE END EC */
00046
00047 /* Exported macro -----*/
00048 /* USER CODE BEGIN EM */
00049
00050 /* USER CODE END EM */
```

```

00051
00052 /* Exported functions prototypes -----*/
00053 void Error_Handler(void);
00054
00055 /* USER CODE BEGIN EFP */
00056
00057 /* USER CODE END EFP */
00058
00059 /* Private defines -----*/
00060 #define USER_Btn_Pin GPIO_PIN_13
00061 #define USER_Btn_GPIO_Port GPIOC
00062 #define MCO_Pin GPIO_PIN_0
00063 #define MCO_GPIO_Port GPIOH
00064 #define LD1_Pin GPIO_PIN_0
00065 #define LD1_GPIO_Port GPIOB
00066 #define LD3_Pin GPIO_PIN_14
00067 #define LD3_GPIO_Port GPIOB
00068 #define STLK_RX_Pin GPIO_PIN_8
00069 #define STLK_RX_GPIO_Port GPIOD
00070 #define STLK_TX_Pin GPIO_PIN_9
00071 #define STLK_TX_GPIO_Port GPIOD
00072 #define USB_PowerSwitchOn_Pin GPIO_PIN_6
00073 #define USB_PowerSwitchOn_GPIO_Port GPIOG
00074 #define USB_OverCurrent_Pin GPIO_PIN_7
00075 #define USB_OverCurrent_GPIO_Port GPIOG
00076 #define USB_SOF_Pin GPIO_PIN_8
00077 #define USB_SOF_GPIO_Port GPIOA
00078 #define USB_VBUS_Pin GPIO_PIN_9
00079 #define USB_VBUS_GPIO_Port GPIOA
00080 #define USB_ID_Pin GPIO_PIN_10
00081 #define USB_ID_GPIO_Port GPIOA
00082 #define USB_DM_Pin GPIO_PIN_11
00083 #define USB_DM_GPIO_Port GPIOA
00084 #define USB_DP_Pin GPIO_PIN_12
00085 #define USB_DP_GPIO_Port GPIOA
00086 #define TMS_Pin GPIO_PIN_13
00087 #define TMS_GPIO_Port GPIOA
00088 #define TCK_Pin GPIO_PIN_14
00089 #define TCK_GPIO_Port GPIOA
00090 #define SWO_Pin GPIO_PIN_3
00091 #define SWO_GPIO_Port GPIOB
00092 #define LD2_Pin GPIO_PIN_7
00093 #define LD2_GPIO_Port GPIOB
00094
00095 /* USER CODE BEGIN Private defines */
00096
00097 /* USER CODE END Private defines */
00098
00099 #ifdef __cplusplus
00100 }
00101 #endif
00102
00103 #endif /* __MAIN_H */

```

## 4.25 util.h

```

00001 #ifndef UTIL_H
00002 #define UTIL_H
00003
00004 #include "main.h"
00005 #include <stdarg.h> //for va_list var arg functions
00006 #include <stdio.h>
00007 #include <string.h>
00008 // #include "stm32f4xx_hal.h"
00009
00010 extern UART_HandleTypeDef huart2;
00011
00012 void uart_printf(const char *fmt, ...);
00013
00014 #endif

```

## 4.26 util.h

```

00001 #ifndef UTIL_H
00002 #define UTIL_H
00003
00004 #include "main.h"
00005 #include <stdarg.h> //for va_list var arg functions
00006 #include <stdio.h>

```



```
00007 #include <string.h>
00008 // #include "stm32f4xx_hal.h"
00009
00010 extern UART_HandleTypeDef huart2;
00011
00012 void uart_printf(const char *fmt, ...);
00013
00014 #endif
```

## 4.27 f401\_sd\_card\_audio\_codec\_test/Core/Inc/wavPlayer.h File Reference

Header file for WAV audio player functionality.

```
#include <stdint.h>
#include "fatfs.h"
#include "audio.h"
```

### Data Structures

- struct [wav\\_header\\_t](#)  
*Represents the header of a WAV file.*
- struct [WavPlayer](#)  
*Represents a WAV audio player with playback control and state management.*

### Typedefs

- typedef struct wav\_header\_t [wav\\_header\\_t](#)  
*Represents the header of a WAV file.*
- typedef struct WavPlayer [WavPlayer](#)  
*Represents a WAV audio player with playback control and state management.*

### Functions

- void [initPlayer](#) ([WavPlayer](#) \*player, FIL \*file, [wav\\_header\\_t](#) \*wavHeader)  
*Initializes a WAV player with specified file and header.*
- FRESULT [wavLoad](#) ([WavPlayer](#) \*player, const char \*filename)  
*Loads a WAV file and validates its header format.*
- uint32\_t [populateWavHeader](#) (FIL \*file, [wav\\_header\\_t](#) \*wavHeader)  
*Reads and populates the WAV header from a file.*
- uint8\_t [checkWav](#) ([WavPlayer](#) \*player)  
*Validates the WAV file header for correctness.*
- uint8\_t [wavPlay](#) ([WavPlayer](#) \*player)  
*Plays the WAV file from the given [WavPlayer](#).*
- uint8\_t [wavPlayPitched](#) ([WavPlayer](#) \*player)  
*Plays a WAV file with pitch shifting applied.*
- void [playButtonHandler](#) ([WavPlayer](#) \*player)  
*Handles the play button action for audio playback.*
- void [wavStop](#) (void)

### 4.27.1 Detailed Description

Header file for WAV audio player functionality.

Defines the structures and function prototypes for handling WAV file playback.

### 4.27.2 Typedef Documentation

#### 4.27.2.1 wav\_header\_t

```
typedef struct wav_header_t wav_header_t
```

Represents the header of a WAV file.

This structure defines the layout of the WAV file header, which includes information about the RIFF chunk, format details, and the data chunk. It provides essential metadata for interpreting the WAV file's audio data.

#### 4.27.2.2 WavPlayer

```
typedef struct WavPlayer WavPlayer
```

Represents a WAV audio player with playback control and state management.

This structure contains information and control flags related to WAV audio playback, including file handling, playback status, and pitch adjustment.

### 4.27.3 Function Documentation

#### 4.27.3.1 checkWav()

```
uint8_t checkWav (
    WavPlayer * player)
```

Validates the WAV file header for correctness.

This function reads the WAV header from the [WavPlayer](#) structure and checks for validity based on key header fields. It verifies the RIFF header and essential "fmt " subchunk to ensure the WAV file conforms to expected formats.

##### Parameters

<i>player</i>	Pointer to the <a href="#">WavPlayer</a> structure containing the WAV file and header.
---------------	--

##### Returns

1 if the WAV header is valid, 0 otherwise.

#### 4.27.3.2 initPlayer()

```
void initPlayer (
    WavPlayer * player,
    FILE * file,
    wav_header_t * wavHeader)
```

Initializes a WAV player with specified file and header.

Sets up the [WavPlayer](#) structure with the given file pointer and WAV header. Initializes playback control flags, header size, and pitch factor to default values.

## Parameters

<i>player</i>	Pointer to the <a href="#">WavPlayer</a> structure to initialize.
<i>file</i>	Pointer to the <code>FIL</code> object representing the WAV file.
<i>wavHeader</i>	Pointer to the <a href="#">wav_header_t</a> structure containing the WAV file header.

### 4.27.3.3 playButtonHandler()

```
void playButtonHandler (  
    WavPlayer * player)
```

Handles the play button action for audio playback.

This function toggles the playback state of the [WavPlayer](#). If playback is not active, it sets the `playbackActive` flag to `true`. If playback is already active, it sets the `restartPlayback` flag to `true` to restart playback.

## Parameters

<i>player</i>	Pointer to the <a href="#">WavPlayer</a> structure managing the audio playback.
---------------	---

### 4.27.3.4 populateWavHeader()

```
uint32_t populateWavHeader (  
    FIL * file,  
    wav\_header\_t * wavHeader)
```

Reads and populates the WAV header from a file.

This function reads the WAV file header and fills the provided [wav\\_header\\_t](#) structure. It continues reading chunks until it finds the "fmt " subchunk and returns the total size of the WAV header.

## Parameters

<i>file</i>	Pointer to the <code>FIL</code> object representing the WAV file.
<i>wavHeader</i>	Pointer to the <a href="#">wav_header_t</a> structure to be populated with header data.

## Returns

Size of the WAV header in bytes, or 0 on error.

### 4.27.3.5 wavLoad()

```
FRESULT wavLoad (  
    WavPlayer * player,  
    const char * filename)
```

Loads a WAV file and validates its header format.

This function opens a WAV file using FATFS, populates the [wav\\_header\\_t](#) structure in the [WavPlayer](#), and checks if the file format is correct. It returns the result of the file opening operation.

**Parameters**

<i>player</i>	Pointer to the <a href="#">WavPlayer</a> structure that will be used to manage the WAV file.
<i>filename</i>	Name of the WAV file to be loaded.

**Returns**

FR\_OK if the file is successfully opened and format is valid, or an error code from FATFS otherwise.

**4.27.3.6 wavPlay()**

```
uint8_t wavPlay (  
    WavPlayer * player)
```

Plays the WAV file from the given [WavPlayer](#).

This function initializes the DMA stream for I2S transmission, reads data from the WAV file, and plays it through the DAC. It handles playback restarting and manages the remaining bytes of audio data. The function also resets the playback state upon completion.

**Parameters**

<i>player</i>	Pointer to the <a href="#">WavPlayer</a> structure containing the WAV file and header.
---------------	--

**Returns**

0 on successful playback completion. Non-zero return values are reserved for error handling.

**4.27.3.7 wavPlayPitched()**

```
uint8_t wavPlayPitched (  
    WavPlayer * player)
```

Plays a WAV file with pitch shifting applied.

This function plays a WAV file with pitch adjustment using DMA for I2S transmission. It reads audio data from the file, applies pitch shifting by adjusting the playback speed, and handles playback restarting. The function updates the pitch factor dynamically if changed.

**Parameters**

<i>player</i>	Pointer to the <a href="#">WavPlayer</a> structure managing the WAV file and playback.
---------------	--

**Returns**

0 on successful playback completion. Non-zero return values are reserved for error handling.

## 4.28 wavPlayer.h

[Go to the documentation of this file.](#)

```

00001
00008 #ifndef INC_WAVPLAYER_H_
00009 #define INC_WAVPLAYER_H_
00010
00011 #include <stdint.h>
00012 #include "fatfs.h"
00013 #include "audio.h"
00014
00022 typedef struct wav_header_t {
00023     // RIFF Header
00024     uint32_t ChunkID;           // Contains "RIFF"
00025     uint32_t ChunkSize;         // Size of the wav portion of the file, which follows the first 8 bytes.
00026     File size - 8
00027     uint32_t Format;           // Contains "WAVE"
00028
00028     // Format Header
00029     uint32_t Subchunk1ID;       // Contains "fmt " (includes trailing space)
00030     uint32_t Subchunk1Size;     // Should be 16 for PCM
00031     uint16_t AudioFormat;       // Should be 1 for PCM. 3 for IEEE Float
00032     uint16_t NumChannels;
00033     uint32_t SampleRate;
00034     uint32_t ByteRate;          // Number of bytes per second. sample_rate * num_channels * Bytes Per
00035     Sample
00036     uint16_t BlockAlign;        // num_channels * Bytes Per Sample
00037     uint16_t BitsPerSample;     // Number of bits per sample
00038
00038     // Data
00039     uint32_t Subchunk2ID;       // Contains "data"
00040     uint32_t Subchunk2Size;     // Number of bytes in data. Number of samples * num_channels * sample
00041     byte size
00042     // uint8_t bytes[];         // Remainder of wave file is bytes
00043 } wav_header_t;
00044
00050 typedef struct WavPlayer {
00051     volatile bool restartPlayback;
00052     volatile bool playbackActive;
00053     FIL *file;
00054     wav_header_t *wavHeader;
00055     uint32_t headerSize;
00056     float pitchFactor;
00057     bool pitchChanged;
00058 } WavPlayer;
00059
00060 void initPlayer(WavPlayer *player, FIL *file, wav_header_t *wavHeader);
00061
00062 FRESULT wavLoad(WavPlayer *player, const char *filename);
00063
00064 uint32_t populateWavHeader(FIL *file, wav_header_t *wavHeader);
00065 uint8_t checkWav(WavPlayer *player);
00066
00067 uint8_t wavPlay(WavPlayer *player);
00068 uint8_t wavPlayPitched(WavPlayer *player);
00069
00070 void playButtonHandler(WavPlayer *player);
00071
00072 void wavStop(void);
00073
00074 #endif /* INC_WAVPLAYER_H_ */

```

## 4.29 f401\_sd\_card\_audio\_codec\_test/Core/Src/audio.c File Reference

Audio playback and processing functions.

```
#include "audio.h"
```

### Macros

- #define SINE\_TABLE\_SIZE 256

## Functions

- void **initSineTable** ()
- void **HAL\_I2S\_TxHalfCpltCallback** (I2S\_HandleTypeDef \*hi2s)  
*Called when the first half of the I2S buffer is transmitted.*
- void **HAL\_I2S\_TxCpltCallback** (I2S\_HandleTypeDef \*hi2s)  
*Called when the entire I2S buffer is transmitted.*
- void **generateSineWave** (double frequency)  
*Populates half of the DAC buffer with a sine wave using a lookup table.*
- uint16\_t **fillHalfBufferFromSD** (WavPlayer \*player, bool pitched)  
*Fills half of the buffer with audio samples from an SD file.*
- void **setPitchFactor** (float newPitchFactor)  
*Sets a new pitch factor for audio playback.*

## Variables

- bool **dma\_dataReady** = false
- int16\_t **dacData** [BUFFER\_SIZE]
- int16\_t **fileReadBuf** [BUFFER\_SIZE\_INPUT]
- volatile int16\_t \* **outBufPtr** = &dacData[0]
- volatile bool **pitchChanged** = false
- int16\_t **sineTable** [SINE\_TABLE\_SIZE]

### 4.29.1 Detailed Description

Audio playback and processing functions.

Implements functions for audio playback using DMA and I2S, including sine wave generation, buffer management, and pitch adjustment. Also provides a precomputed sine wave table for testing the Codec.

### 4.29.2 Function Documentation

#### 4.29.2.1 fillHalfBufferFromSD()

```
uint16_t fillHalfBufferFromSD (
    WavPlayer * player,
    bool pitched)
```

Fills half of the buffer with audio samples from an SD file.

The function reads audio samples from an SD card file into a buffer. If the `pitched` parameter is true, it adjusts the number of samples to account for pitch shifting. It ensures the number of samples is even and reads the data into `fileReadBuf`.

#### Parameters

<i>player</i>	Pointer to a <a href="#">WavPlayer</a> structure containing file information and pitch factor.
<i>pitched</i>	Flag indicating whether pitch shifting should be applied.

#### Returns

The number of bytes actually read from the file.

#### 4.29.2.2 generateSineWave()

```
void generateSineWave (
    double frequency)
```

Populates half of the DAC buffer with a sine wave using a lookup table.

The function generates a sine wave based on the provided frequency. It uses a lookup table to fill the DAC buffer with sine values and manages the phase index for the waveform. It waits for the DMA to be ready before updating the buffer.

##### Parameters

<i>frequency</i>	Desired frequency of the sine wave.
------------------	-------------------------------------

< Retains the index between function calls

#### 4.29.2.3 HAL\_I2S\_TxCpltCallback()

```
void HAL_I2S_TxCpltCallback (
    I2S_HandleTypeDef * hi2s)
```

Called when the entire I2S buffer is transmitted.

Updates the buffer pointer to the second half of `dacData` and sets the DMA data ready flag.

##### Parameters

<i>hi2s</i>	Pointer to the I2S handle structure.
-------------	--------------------------------------

#### 4.29.2.4 HAL\_I2S\_TxHalfCpltCallback()

```
void HAL_I2S_TxHalfCpltCallback (
    I2S_HandleTypeDef * hi2s)
```

Called when the first half of the I2S buffer is transmitted.

Updates the buffer pointer to the start of `dacData` and sets the DMA data ready flag.

##### Parameters

<i>hi2s</i>	Pointer to the I2S handle structure.
-------------	--------------------------------------

#### 4.29.2.5 setPitchFactor()

```
void setPitchFactor (
    float newPitchFactor)
```

Sets a new pitch factor for audio playback.

Updates the global pitch factor and marks it as changed. Interrupts are temporarily disabled to ensure thread safety during the update.

## Parameters

<i>newPitchFactor</i>	The new pitch factor to set.
-----------------------	------------------------------

## 4.30 f401\_sd\_card\_audio\_codec\_test/Core/Src/audioPreprocessor.c File Reference

Audio preprocessing functions for FIR decimation and resampling.

```
#include "audioPreprocessor.h"
```

### Functions

- uint32\_t **DWT\_GetCycleCount** (void)
- arm\_status **initFilter** ()  
*Initializes the FIR decimation filter.*
- uint16\_t **resampleFile** (FIL \*inFil, FIL \*outFil, wav\_header\_t \*waveHeaderInput, wav\_header\_t \*waveHeaderResampled)  
*Resamples audio data from an input file and writes to an output file.*
- void **downsample\_Block** (int16\_t \*src, int16\_t \*dest)  
*Performs downsampling and FIR filtering on a block of audio data.*
- uint16\_t **get\_number\_subsamples** (FIL \*file)  
*Calculates the number of subsamples based on the WAV file data.*
- uint32\_t **downsample\_to\_1024\_samples** (FIL \*file, int16\_t outChunk[NUM\_SAMPLES\_CHUNK\_OUT])  
*Downsamples a chunk of audio data to 1024 samples.*

### Variables

- arm\_fir\_decimate\_instance\_f32 **FIR\_F32\_Struct**

### 4.30.1 Detailed Description

Audio preprocessing functions for FIR decimation and resampling.

Contains functions for initializing FIR decimation filters, resampling audio files, and performing block downsampling of audio data for the neuronal net input. Utilizes the ARM CMSIS DSP library for efficient signal processing.

### 4.30.2 Function Documentation

#### 4.30.2.1 downsample\_Block()

```
void downsample_Block (
    int16_t * src,
    int16_t * dest)
```

Performs downsampling and FIR filtering on a block of audio data.

Converts a block of 16-bit integer audio samples to floating-point format, applies FIR filtering and decimation, and then converts the filtered samples back to 16-bit integer format. The processed data is written to the destination buffer.



**Parameters**

<i>src</i>	Pointer to the source buffer containing 16-bit audio samples to be processed.
<i>dest</i>	Pointer to the destination buffer where the downsampled 16-bit audio samples will be written.

**4.30.2.2 downsample\_to\_1024\_samples()**

```
uint32_t downsample_to_1024_samples (
    FIL * file,
    int16_t outChunk[NUM_SAMPLES_CHUNK_OUT])
```

Downsamples a chunk of audio data to 1024 samples.

Reads a chunk of audio data from the specified file, converts stereo samples to mono, and then performs down-sampling. The processed samples are written to the `outChunk` buffer. The function processes data in blocks and returns the number of downsampled samples.  $\text{NUM\_SAMPLES\_CHUNK\_OUT} * \text{DECIMATION\_FACTOR\_ROUNDED} = \text{NUM\_SAMPLES\_CHUNK\_IN}$

**Parameters**

<i>file</i>	Pointer to the input file ( <code>FIL</code> structure) containing audio data.
<i>outChunk</i>	Array to store the downsampled audio samples.

**Returns**

The actual number of samples written to the `outChunk` buffer.

**4.30.2.3 get\_number\_subsamples()**

```
uint16_t get_number_subsamples (
    FIL * file)
```

Calculates the number of subsamples based on the WAV file data.

Determines the number of mono subsamples in a WAV file after applying downsampling. It calculates the total number of samples based on the file's audio data size, decimation factor, and header information, then computes the number of frames using the `HOP_SIZE` and `STEP_SIZE`.

**Parameters**

<i>file</i>	Pointer to the WAV file ( <code>FIL</code> structure) from which the header information is read.
-------------	--

**Returns**

The number of frames after downsampling, as a `uint16_t`.

#### 4.30.2.4 `initFilter()`

```
arm_status initFilter ()
```

Initializes the FIR decimation filter.

Configures the FIR decimation filter using the ARM CMSIS DSP library. Sets up the filter structure with the specified number of taps, decimation factor, filter coefficients, state buffer, and block size.

##### Returns

Status of the filter initialization (`arm_status`).

#### 4.30.2.5 `resampleFile()`

```
uint16_t resampleFile (
    FIL * inFil,
    FIL * outFil,
    wav_header_t * waveHeaderInput,
    wav_header_t * waveHeaderResampled)
```

Resamples audio data from an input file and writes to an output file.

Reads chunks of audio data from the input file, performs downsampling and resampling, and writes the processed data to the output file. Updates the WAV header of the output file to reflect the new format after resampling.

##### Parameters

<i>inFil</i>	Pointer to the input file ( <code>FIL</code> structure) containing original audio data.
<i>outFil</i>	Pointer to the output file ( <code>FIL</code> structure) to write resampled audio data.
<i>waveHeaderInput</i>	Pointer to the WAV header of the input file.
<i>waveHeaderResampled</i>	Pointer to the WAV header to be written to the output file.

##### Returns

The number of bytes written to the resampled file.

## 4.31 `f401_sd_card_audio_codec_test/Core/Src/cpu_time.c` File Reference

Functions for cycle counting and time measurement using DWT on STM32.

```
#include "cpu_time.h"
#include "stm32f4xx_hal.h"
```

## Functions

- void [EnableDWT](#) (void)  
*Configures and enables the Data Watchpoint and Trace (DWT) unit.*
- void [StartCycleMeasurement](#) (void)  
*Starts cycle measurement.*
- void [StopCycleMeasurement](#) (void)  
*Stops cycle measurement.*
- uint32\_t [GetMeasuredCycles](#) (void)  
*Retrieves the measured number of cycles.*
- uint32\_t [CyclesToMilliseconds](#) (uint32\_t cycles)  
*Converts cycle count to milliseconds.*
- uint32\_t [CyclesToMicroseconds](#) (uint32\_t cycles)  
*Converts cycle count to microseconds.*

## Variables

- volatile unsigned int \* [DWT\\_CYCCNT](#) = (volatile unsigned int \*)0xE0001004
- volatile unsigned int \* [DWT\\_CONTROL](#) = (volatile unsigned int \*)0xE0001000
- volatile unsigned int \* [DWT\\_LAR](#) = (volatile unsigned int \*)0xE0001FB0
- volatile unsigned int \* [SCB\\_DEMCR](#) = (volatile unsigned int \*)0xE000EDFC

### 4.31.1 Detailed Description

Functions for cycle counting and time measurement using DWT on STM32.

This file provides functions to configure and use the Data Watchpoint and Trace (DWT) unit on STM32 microcontrollers for precise cycle counting.

### 4.31.2 Function Documentation

#### 4.31.2.1 CyclesToMicroseconds()

```
uint32_t CyclesToMicroseconds (
    uint32_t cycles)
```

Converts cycle count to microseconds.

Converts a given number of cycles to microseconds based on the system clock frequency.

#### Parameters

<i>cycles</i>	The number of cycles to convert.
---------------	----------------------------------

#### Returns

The equivalent time in microseconds.

#### 4.31.2.2 CyclesToMilliseconds()

```
uint32_t CyclesToMilliseconds (
    uint32_t cycles)
```

Converts cycle count to milliseconds.

Converts a given number of cycles to milliseconds based on the system clock frequency.

**Parameters**

<i>cycles</i>	The number of cycles to convert.
---------------	----------------------------------

**Returns**

The equivalent time in milliseconds.

**4.31.2.3 EnableDWT()**

```
void EnableDWT (  
    void )
```

Configures and enables the Data Watchpoint and Trace (DWT) unit.

Enables DWT and ITM, unlocks DWT registers, resets the cycle counter, and enables the cycle counter.

**4.31.2.4 GetMeasuredCycles()**

```
uint32_t GetMeasuredCycles (  
    void )
```

Retrieves the measured number of cycles.

Calculates the difference between `end_cycles` and `start_cycles` to return the number of cycles elapsed.

**Returns**

The number of cycles measured.

**4.31.2.5 StartCycleMeasurement()**

```
void StartCycleMeasurement (  
    void )
```

Starts cycle measurement.

Records the current cycle count from the DWT cycle counter to `start_cycles`.

**4.31.2.6 StopCycleMeasurement()**

```
void StopCycleMeasurement (  
    void )
```

Stops cycle measurement.

Records the current cycle count from the DWT cycle counter to `end_cycles`.

## 4.32 f401\_sd\_card\_audio\_codec\_test/Core/Src/lowpass\_16kFilter.c File Reference

Contains the filter coefficients for a 16 kHz low-pass FIR filter.

```
#include "lowpass_16kFilter.h"
```

### Variables

- float `filter_taps` [NUM\_TAPS]  
*Array of filter coefficients for the FIR filter.*

### 4.32.1 Detailed Description

Contains the filter coefficients for a 16 kHz low-pass FIR filter.

This file defines an array of filter coefficients used in a finite impulse response (FIR) filter. The filter is designed for low-pass filtering with a cutoff frequency suitable for 16 kHz sampled audio or signal data.

### 4.32.2 Variable Documentation

#### 4.32.2.1 filter\_taps

```
float filter_taps[NUM_TAPS]
```

Array of filter coefficients for the FIR filter.

Holds the coefficients used in the FIR filter. The size of the array is determined by NUM\_TAPS, which specifies the number of filter taps. These coefficients are used for filtering audio or signal data.

## 4.33 f401\_display\_encoder\_fader\_test/Core/Src/main.c File Reference

: Main program body

```
#include "main.h"  
#include "fatfs.h"  
#include <stdlib.h>  
#include <stdbool.h>  
#include "ssd1306.h"  
#include "fonts.h"  
#include "display.h"  
#include "filemanager.h"  
#include "interface.h"  
#include "util.h"
```

## Functions

- void [SystemClock\\_Config](#) (void)  
*System Clock Configuration.*
- void [writeStringToFile](#) (const char \*filename, const char \*str)  
*Writes a string to a file on the SD card.*
- int [main](#) (void)  
*The application entry point.*
- void [Error\\_Handler](#) (void)  
*This function is executed in case of error occurrence.*

## Variables

- ADC\_HandleTypeDef **hadc1**
- DMA\_HandleTypeDef **hdma\_adc1**
- I2C\_HandleTypeDef **hi2c1**
- SD\_HandleTypeDef **hsd**
- DMA\_HandleTypeDef **hdma\_sdio\_rx**
- DMA\_HandleTypeDef **hdma\_sdio\_tx**
- TIM\_HandleTypeDef **htim3**
- TIM\_HandleTypeDef **htim5**
- UART\_HandleTypeDef **huart2**
- [FileManager](#) **fm**  
*[FileManager](#) instances for managing and testing files.*
- [FileManager](#) **fmCopy**  
*Copy of [FileManager](#) for testing purposes.*
- char **fn** [MAX\_FILENAME\_LENGTH]  
*Buffer for the filename string.*
- FILINFO **fno**  
*FATFS file system objects and handles.*
- FATFS **FatFs**  
*FATFS handle for the file system.*
- FIL **file**  
*[File](#) object for FATFS.*
- UINT **bytesWritten**  
*Number of bytes written in file operations.*
- FRESULT **fres**  
*Result of FATFS operations.*
- DIR **dir**  
*Directory handle for directory operations.*
- char **fileNamesSDCard** [MAX\_FILES][MAX\_FILENAME\_LENGTH]  
*List of filenames on the SD card with .wav extension.*
- int **fileCount** = 0  
*Counter for the number of .wav files found on the SD card.*
- bool **adcDmaFlag** = false  
*Flags for managing time intervals and DMA operations.*
- bool **updateScreenFlag** = false  
*Flag to manage the time interval for display updates.*
- uint32\_t **adcBuffer** [NUM\_CHANNELS] = {0}  
*Buffer for ADC polling.*

### 4.33.1 Detailed Description

: Main program body

Attention

Copyright (c) 2024 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.

### 4.33.2 Function Documentation

#### 4.33.2.1 Error\_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

Return values

None	
------	--

#### 4.33.2.2 main()

```
int main (
    void )
```

The application entry point.

Return values

int	
-----	--

#### 4.33.2.3 SystemClock\_Config()

```
void SystemClock_Config (
    void )
```

System Clock Configuration.

Return values

None	
------	--

Configure the main internal regulator output voltage

Initializes the RCC Oscillators according to the specified parameters in the RCC\_OscInitTypeDef structure.

Initializes the CPU, AHB and APB buses clocks

#### 4.33.2.4 writeStringToFile()

```
void writeStringToFile (
    const char * filename,
    const char * str)
```

Writes a string to a file on the SD card.

This function opens a file for writing and writes a specified string to it. If the file does not exist, it will be created. The function performs the following actions:

- Opens the file specified by `filename` in write mode, creating it if it does not already exist.
- Writes the provided string to the file.
- Checks for errors during the file operations and reports them.
- Closes the file after writing is completed.

##### Parameters

in	<i>filename</i>	The name of the file to write to. (Note: This parameter is not used in the current implementation.)
in	<i>str</i>	The string to be written to the file.

##### Note

In the current implementation, the `filename` parameter is not used, and the file name is hardcoded as `"test.txt"`. Modify the file opening mode and filename if you need to use the provided `filename` parameter.

### 4.33.3 Variable Documentation

#### 4.33.3.1 adcBuffer

```
uint32_t adcBuffer[NUM_CHANNELS] = {0}
```

Buffer for ADC polling.

This buffer stores the values read from the ADC channels. Buffer for storing ADC channel values.

#### 4.33.3.2 adcDmaFlag

```
bool adcDmaFlag = false
```

Flags for managing time intervals and DMA operations.

These flags are used to manage the intervals at which DMA operations and display updates occur. Flag to manage the time interval for ADC DMA operations.



#### 4.33.3.3 fileNamesSDCard

```
char fileNamesSDCard[MAX_FILES][MAX_FILENAME_LENGTH]
```

List of filenames on the SD card with .wav extension.

This array stores the filenames of all .wav files found on the SD card. Array of filenames on the SD card with .wav extension.

#### 4.33.3.4 fm

```
FileManager fm
```

[FileManager](#) instances for managing and testing files.

These variables are used to manage files, including the main [FileManager](#) and a copy for testing purposes. Main [FileManager](#) instance.

#### 4.33.3.5 fno

```
FILINFO fno
```

FATFS file system objects and handles.

These variables are used for managing file operations and directory access with the FATFS library. FileInfo handle for displaying filenames.

#### 4.33.3.6 lfn

```
char lfn[MAX_FILENAME_LENGTH]
```

Buffer for the filename string.

This buffer stores filenames with a maximum length defined by MAX\_FILENAME\_LENGTH. Buffer for storing a filename string.

## 4.34 f401\_sd\_card\_audio\_codec\_test/Core/Src/main.c File Reference

: Main program body

```
#include "main.h"
#include "fatfs.h"
#include <stdio.h>
#include <string.h>
#include "math.h"
#include "audio.h"
#include "wavPlayer.h"
#include "cpu_time.h"
#include "util.h"
#include "audioPreprocessor.h"
```

## Functions

- void [SystemClock\\_Config](#) (void)  
*System Clock Configuration.*
- int [main](#) (void)  
*The application entry point.*
- void **HAL\_GPIO\_EXTI\_Callback** (uint16\_t GPIO\_Pin)
- void [Error\\_Handler](#) (void)  
*This function is executed in case of error occurrence.*

## Variables

- I2S\_HandleTypeDef **hi2s2**
- DMA\_HandleTypeDef **hdma\_spi2\_tx**
- SD\_HandleTypeDef **hsd**
- DMA\_HandleTypeDef **hdma\_sdio\_rx**
- DMA\_HandleTypeDef **hdma\_sdio\_tx**
- UART\_HandleTypeDef **huart2**
- [WavPlayer](#) **player**

### 4.34.1 Detailed Description

: Main program body

Attention

© Copyright (c) 2020 STMicroelectronics. All rights reserved.

This software component is licensed by ST under BSD 3-Clause license, the "License"; You may not use this file except in compliance with the License. You may obtain a copy of the License at: [opensource.org/licenses/BSD-3-Clause](https://opensource.org/licenses/BSD-3-Clause)

### 4.34.2 Function Documentation

#### 4.34.2.1 Error\_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

Return values

<i>None</i>	
-------------	--

#### 4.34.2.2 main()

```
int main (
    void )
```

The application entry point.

## Return values

<i>int</i>	
------------	--

**4.34.2.3 SystemClock\_Config()**

```
void SystemClock_Config (
    void )
```

System Clock Configuration.

## Return values

<i>None</i>	
-------------	--

Configure the main internal regulator output voltage

Initializes the RCC Oscillators according to the specified parameters in the RCC\_OscInitTypeDef structure.

Initializes the CPU, AHB and APB buses clocks

**4.35 neuronal\_network/esp\_cubeai\_test/Core/Src/main.c File Reference**

: Main program body

```
#include "main.h"
#include <stdio.h>
#include <stdlib.h>
#include "audio_classification.h"
#include "test_audio_data.h"
#include "utils.h"
```

**Functions**

- void [SystemClock\\_Config](#) (void)  
*System Clock Configuration.*
- int [main](#) (void)  
*The application entry point.*
- void [Error\\_Handler](#) (void)  
*This function is executed in case of error occurrence.*

**Variables**

- CRC\_HandleTypeDef **hcrc**
- TIM\_HandleTypeDef **htim2**
- TIM\_HandleTypeDef **htim3**
- UART\_HandleTypeDef **huart3**
- PCD\_HandleTypeDef **hpcd\_USB\_OTG\_FS**

### 4.35.1 Detailed Description

: Main program body

Attention

Copyright (c) 2024 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.

### 4.35.2 Function Documentation

#### 4.35.2.1 Error\_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

Return values

<i>None</i>	
-------------	--

#### 4.35.2.2 main()

```
int main (
    void )
```

The application entry point.

Return values

<i>int</i>	
------------	--

#### 4.35.2.3 SystemClock\_Config()

```
void SystemClock_Config (
    void )
```

System Clock Configuration.

Return values

<i>None</i>	
-------------	--

Configure LSE Drive Capability

Configure the main internal regulator output voltage

Initializes the RCC Oscillators according to the specified parameters in the RCC\_OscInitTypeDef structure.

Activate the Over-Drive mode

Initializes the CPU, AHB and APB buses clocks

## 4.36 f401\_sd\_card\_audio\_codec\_test/Core/Src/util.c File Reference

Utility functions for the project.

```
#include "util.h"
```

### Functions

- void `uart_printf` (const char \*fmt,...)  
*Formats and sends a string over UART.*

### 4.36.1 Detailed Description

Utility functions for the project.

This file contains utility functions used across the project. Currently, it includes a function to format and send strings over UART.

Functions provided:

- void `uart_printf(const char *fmt, ...)`: Formats and sends a string over UART.

The `uart_printf` function:

- Uses `vsnprintf` to format a string.
- Uses `HAL_UART_Transmit` to send the formatted string over UART2.

### 4.36.2 Function Documentation

#### 4.36.2.1 `uart_printf()`

```
void uart_printf (  
    const char * fmt,  
    ...)
```

Formats and sends a string over UART.

Uses `vsnprintf` to format the string and `HAL_UART_Transmit` to send it over UART2.

#### Parameters

<i>fmt</i>	Format string.
...	Arguments for formatting.

## 4.37 f401\_sd\_card\_audio\_codec\_test/Core/Src/wavPlayer.c File Reference

Implementation of a WAV player using FATFS and I2S with DMA.

```
#include "fatfs.h"
#include "wavPlayer.h"
#include "string.h"
```

### Functions

- void [initPlayer](#) ([WavPlayer](#) \*player, FIL \*file, [wav\\_header\\_t](#) \*wavHeader)  
*Initializes a WAV player with specified file and header.*
- uint32\_t [populateWavHeader](#) (FIL \*file, [wav\\_header\\_t](#) \*wavHeader)  
*Reads and populates the WAV header from a file.*
- uint8\_t [checkWav](#) ([WavPlayer](#) \*player)  
*Validates the WAV file header for correctness.*
- void [playButtonHandler](#) ([WavPlayer](#) \*player)  
*Handles the play button action for audio playback.*
- FRESULT [wavLoad](#) ([WavPlayer](#) \*player, const char \*filename)  
*Loads a WAV file and validates its header format.*
- uint8\_t [wavPlay](#) ([WavPlayer](#) \*player)  
*Plays the WAV file from the given [WavPlayer](#).*
- uint8\_t [wavPlayPitched](#) ([WavPlayer](#) \*player)  
*Plays a WAV file with pitch shifting applied.*

### 4.37.1 Detailed Description

Implementation of a WAV player using FATFS and I2S with DMA.

This file provides functions to:

- Initialize the WAV player.
- Read and validate WAV file headers.
- Handle playback control via button actions.
- Load and play WAV files, with optional pitch shifting.

### 4.37.2 Function Documentation

#### 4.37.2.1 [checkWav\(\)](#)

```
uint8_t checkWav (  
    WavPlayer * player)
```

Validates the WAV file header for correctness.

This function reads the WAV header from the [WavPlayer](#) structure and checks for validity based on key header fields. It verifies the RIFF header and essential "fmt " subchunk to ensure the WAV file conforms to expected formats.

## Parameters

<i>player</i>	Pointer to the <a href="#">WavPlayer</a> structure containing the WAV file and header.
---------------	--

## Returns

1 if the WAV header is valid, 0 otherwise.

#### 4.37.2.2 initPlayer()

```
void initPlayer (  
    WavPlayer * player,  
    FILE * file,  
    wav\_header\_t * wavHeader)
```

Initializes a WAV player with specified file and header.

Sets up the [WavPlayer](#) structure with the given file pointer and WAV header. Initializes playback control flags, header size, and pitch factor to default values.

## Parameters

<i>player</i>	Pointer to the <a href="#">WavPlayer</a> structure to initialize.
<i>file</i>	Pointer to the FILE object representing the WAV file.
<i>wavHeader</i>	Pointer to the <a href="#">wav_header_t</a> structure containing the WAV file header.

#### 4.37.2.3 playButtonHandler()

```
void playButtonHandler (  
    WavPlayer * player)
```

Handles the play button action for audio playback.

This function toggles the playback state of the [WavPlayer](#). If playback is not active, it sets the `playbackActive` flag to true. If playback is already active, it sets the `restartPlayback` flag to true to restart playback.

## Parameters

<i>player</i>	Pointer to the <a href="#">WavPlayer</a> structure managing the audio playback.
---------------	---

#### 4.37.2.4 populateWavHeader()

```
uint32_t populateWavHeader (  
    FILE * file,  
    wav\_header\_t * wavHeader)
```

Reads and populates the WAV header from a file.

This function reads the WAV file header and fills the provided [wav\\_header\\_t](#) structure. It continues reading chunks until it finds the "fmt" subchunk and returns the total size of the WAV header.

**Parameters**

<i>file</i>	Pointer to the <code>FIL</code> object representing the WAV file.
<i>wavHeader</i>	Pointer to the <code>wav_header_t</code> structure to be populated with header data.

**Returns**

Size of the WAV header in bytes, or 0 on error.

**4.37.2.5 wavLoad()**

```
FRESULT wavLoad (
    WavPlayer * player,
    const char * filename)
```

Loads a WAV file and validates its header format.

This function opens a WAV file using FATFS, populates the `wav_header_t` structure in the `WavPlayer`, and checks if the file format is correct. It returns the result of the file opening operation.

**Parameters**

<i>player</i>	Pointer to the <code>WavPlayer</code> structure that will be used to manage the WAV file.
<i>filename</i>	Name of the WAV file to be loaded.

**Returns**

`FR_OK` if the file is successfully opened and format is valid, or an error code from FATFS otherwise.

**4.37.2.6 wavPlay()**

```
uint8_t wavPlay (
    WavPlayer * player)
```

Plays the WAV file from the given `WavPlayer`.

This function initializes the DMA stream for I2S transmission, reads data from the WAV file, and plays it through the DAC. It handles playback restarting and manages the remaining bytes of audio data. The function also resets the playback state upon completion.

**Parameters**

<i>player</i>	Pointer to the <code>WavPlayer</code> structure containing the WAV file and header.
---------------	---

**Returns**

0 on successful playback completion. Non-zero return values are reserved for error handling.

**4.37.2.7 wavPlayPitched()**

```
uint8_t wavPlayPitched (
    WavPlayer * player)
```

Plays a WAV file with pitch shifting applied.

This function plays a WAV file with pitch adjustment using DMA for I2S transmission. It reads audio data from the file, applies pitch shifting by adjusting the playback speed, and handles playback restarting. The function updates the pitch factor dynamically if changed.



## Parameters

<i>player</i>	Pointer to the <a href="#">WavPlayer</a> structure managing the WAV file and playback.
---------------	--

## Returns

0 on successful playback completion. Non-zero return values are reserved for error handling.

## 4.38 audio\_classification.h

```

00001 /*
00002  * audio_classification.h
00003  *
00004  * Created on: Jun 20, 2024
00005  * Author: leon
00006  */
00007
00008 #ifndef AUDIO_CLASSIFICATION_H_
00009 #define AUDIO_CLASSIFICATION_H_
00010
00011 /* includes */
00012 #include "feature_extraction.h"
00013 #include "featurescaler.h"
00014 #include "network_l.h"
00015 #include "network_l_data.h"
00016 #include "network_l_config.h"
00017 #include "network_l_data_params.h"
00018
00019 /* defines */
00020 // define spectrogram dimensions
00021 #define NUM_MEL_BANDS 30
00022 #define SPECTROGRAM_ROWS NUM_MEL_BANDS
00023 #define SPECTROGRAM_COLS 32
00024
00025 // define data preprocessing variables
00026 #define TARGET_SAMPLE_RATE 16000
00027 #define FRAME_LENGTH 1024
00028 #define HOP_LENGTH 512
00029 #define N_SAMPLES_PER_SUBSAMPLE 16896
00030 #define N_FRAMES_PER_SUBSAMPLE SPECTROGRAM_COLS
00031
00032
00033 /* function definitions */
00034 int init_nn(void);
00035 int de_init_nn(void);
00036 int run_nn_classification(float* pSpectrogram, float* classification_result);
00037 void spectrogram_generation_init(void);
00038 void classify_file(FILE *fil, char* file_name)
00039 void calculate_spectrogram_column(float* frame, int col_index);
00040 void calculate_total_classification_result(float* total_file_classification_result, float
classification_results_subsamples[][AI_NETWORK_1_OUT_1_SIZE], int number_subsamples_in_file);
00041 void spectrogram_power_to_db(float32_t *spectrogram);
00042 void store_classification_result(FILE *fil, char* file_name, float* classification_result)
00043
00044 // stm32cube.ai related variables
00045 extern ai_handle network;
00046 extern ai_u8 activations[AI_NETWORK_1_DATA_ACTIVATIONS_SIZE];
00047 extern ai_buffer * ai_input;
00048 extern ai_buffer * ai_output;
00049
00050 // own input and output variables for nn classification
00051 extern float spectrogram[AI_NETWORK_1_IN_1_SIZE];
00052 extern float aiOutData[AI_NETWORK_1_OUT_1_SIZE];
00053
00054 // variable for mapping aiOutData indices to the actual labels
00055 extern char aiOutDataLabels[AI_NETWORK_1_OUT_1_SIZE][10];
00056
00057
00058 // variables for spectrogram generation parameters
00059 static arm_rfft_fast_instance_f32 S_Rfft;
00060 static SpectrogramTypeDef S_Spectr;
00061 static MelSpectrogramTypeDef S_MelSpectr;
00062 static MelFilterTypeDef S_MelFilter;
00063
00064 #endif

```

## 4.39 audio\_classification.h

```

00001 /*
00002  * audio_classification.h
00003  *
00004  * Created on: Jun 20, 2024
00005  * Author: leon
00006  */
00007
00008 #ifndef AUDIO_CLASSIFICATION_H_
00009 #define AUDIO_CLASSIFICATION_H_
00010
00011 #include "feature_extraction.h"
00012 #include "network_1.h"
00013 #include "network_1_data.h"
00014 #include "network_1_config.h"
00015 #include "network_1_data_params.h"
00016
00017 /* defines */
00018 // define spectrogram dimensions
00019 #define NUM_MEL_BANDS 30
00020 #define SPECTROGRAM_ROWS NUM_MEL_BANDS
00021 #define SPECTROGRAM_COLS 32
00022
00023 // define data preprocessing variables
00024 #define TARGET_SAMPLE_RATE 16000
00025 #define FRAME_LENGTH 1024
00026 #define HOP_LENGTH 512
00027 #define N_SAMPLES_PER_SUBSAMPLE 16896
00028 #define N_FRAMES_PER_SUBSAMPLE SPECTROGRAM_COLS
00029
00030 /* function definitions */
00031 int init_nn(void);
00032 int run_nn_classification(float* pSpectrogram, float* classification_result);
00033 void test_call_classification(void);
00034 void Preprocessing_Init(void);
00035 int resample_audio(const short* input_data, int input_length, int input_sample_rate, float32_t*
    output_data, int* output_length);
00036 void frame_subsamples(float32_t* subsample);
00037 void preprocess_frame(float* frame);
00038 void process_subsample(float* pSpectrogram);
00039 void PowerToDb(float32_t *spectrogram);
00040
00041 // stm32cube.ai related variables
00042 extern ai_handle network;
00043 extern ai_u8 activations[AI_NETWORK_1_DATA_ACTIVATIONS_SIZE];
00044 extern ai_buffer * ai_input;
00045 extern ai_buffer * ai_output;
00046
00047 // own input and output variables for nn classification
00048 extern ai_float spectrogram[AI_NETWORK_1_IN_1_SIZE];
00049 extern ai_float aiOutData[AI_NETWORK_1_OUT_1_SIZE];
00050
00051 // counter variable for current column number of the mel spectrogram
00052 extern uint32_t spectrogram_col_index;
00053
00054 // variables for spectrogram generation parameters
00055 static arm_rfft_fast_instance_f32 S_Rfft;
00056 static SpectrogramTypeDef S_Spectr;
00057 static MelSpectrogramTypeDef S_MelSpectr;
00058 static MelFilterTypeDef S_MelFilter;
00059
00060 #endif

```

## 4.40 neuronal\_network/esp\_cubeai\_for\_integration/Core/Src/audio\_↵ classification.c File Reference

Audio classification implementation for embedded systems via stm32cube.ai (for integration into the system). Everything is untested!

```

#include "audio_classification.h"
#include <math.h>
#include "feature_extraction.h"
#include "network_1.h"
#include "network_1_data.h"
#include "network_1_config.h"

```

```
#include "network_1_data_params.h"
```

## Functions

- int `init_nn` ()  
*Initializes the neural network for audio classification.*
- int `de_init_nn` ()  
*De-initializes the neural network and disables necessary hardware configurations.*
- int `run_nn_classification` (ai\_float \*pSpectrogram, ai\_float \*classification\_result)  
*Runs the neural network to classify audio data based on the provided spectrogram.*
- void `spectrogram_generation_init` (void)  
*Initializes the mel-spectrogram calculation lib which is part of the audio preprocessing.*
- void `classify_file` (FIL \*fil, char \*file\_name)  
*Classifies audio data from a single file using a neural network.*
- void `calculate_spectrogram_column` (float \*pFrame, int col\_index)  
*Generates a Mel-scaled spectrogram column from a frame (1024 samples) and inserts it into a spectrogram array.*
- void `spectrogram_power_to_db` (float \*pSpectrogram)  
*Converts power values in a spectrogram to decibels (dB).*
- void `calculate_total_classification_result` (float \*total\_file\_classification\_result, float classification\_results\_↔ subsamples[][AI\_NETWORK\_1\_OUT\_1\_SIZE], int number\_subsamples\_in\_file)  
*Aggregates classification results from subsamples into a total result.*
- void `store_classification_result` (FIL \*fil, char \*file\_name, float \*classification\_result)  
*Stores the classification results into struct saved to a file.*

## Variables

- ai\_handle **network**  
*Handle to the neural network (stm32cube.ai) instance used for audio classification.*
- ai\_u8 **activations** [AI\_NETWORK\_1\_DATA\_ACTIVATIONS\_SIZE]  
*Buffer for storing activation data required by stm32cube.ai.*
- ai\_buffer \* **ai\_input**  
*Pointer to the input buffer of the stm32cube.ai.*
- ai\_buffer \* **ai\_output**  
*Pointer to the output buffer of stm32cube.ai, where classification results are stored.*
- ai\_float **spectrogram** [AI\_NETWORK\_1\_IN\_1\_SIZE]  
*Array to hold the input spectrogram data that feeds into the neural network.*
- float **aiOutData** [AI\_NETWORK\_1\_OUT\_1\_SIZE] = {0.0, 0.0, 0.0, 0.0, 0.0}  
*Output data from the neural network, containing classification results.*
- char **aiOutDataLabels** [AI\_NETWORK\_1\_OUT\_1\_SIZE][10]  
*Labels for the classification outputs of the neural network.*
- float32\_t **mel\_spectrogram\_column\_buffer** [SPECTROGRAM\_ROWS]  
*Buffer to store a single column of the Mel spectrogram during preprocessing.*

### 4.40.1 Detailed Description

Audio classification implementation for embedded systems via stm32cube.ai (for integration into the system). Everything is untested!

This file includes the complete integration of audio classification functions tailored for an embedded environment using STM32 hardware. It covers the initialization and de-initialization of neural network components, the generation and processing of spectrograms from audio data, classification of these spectrograms, and the handling of classification results. The code supports operations on audio data that involve feature scaling, neural network inference, and post-processing to interpret the network's output.

Functions in this file demonstrate the full pipeline from reading raw audio data, converting it into Mel-spectrogram columns, classifying these spectrograms using a pre-trained neural network, and storing the results.

#### Date

June 20, 2024

#### Author

Leon Braungardt

### 4.40.2 Function Documentation

#### 4.40.2.1 calculate\_spectrogram\_column()

```
void calculate_spectrogram_column (  
    float * pFrame,  
    int col_index)
```

Generates a Mel-scaled spectrogram column from a frame (1024 samples) and inserts it into a spectrogram array.

#### Parameters

<i>float*</i>	pFrame Pointer to the audio frame.
<i>int</i>	col_index Index at which the spectrogram column will be inserted into the main spectrogram array.

#### 4.40.2.2 calculate\_total\_classification\_result()

```
void calculate_total_classification_result (  
    float * total_file_classification_result,  
    float classification_results_subsamples[] [AI_NETWORK_1_OUT_1_SIZE],  
    int number_subsamples_in_file)
```

Aggregates classification results from subsamples into a total result.

This function calculates the total classification results by summing and then averaging the results of individual subsamples.

## Parameters

<i>float*</i>	total_file_classification_result Array to store the final averaged classification results.
<i>float</i>	classification_results_subsamples[][AI_NETWORK_1_OUT_1_SIZE] Two-dimensional array containing classification results for each subsample.
<i>int</i>	number_subsamples_in_file The total number of subsamples processed, used for averaging.

## 4.40.2.3 classify\_file()

```
void classify_file (
    FIL * fil,
    char * file_name)
```

Classifies audio data from a single file using a neural network.

This function processes an audio file and splits it into audiosubsamples. It handles reading data from the file, generating spectrogram columns, converting them to dB, running neural network classifications, and aggregating the results. The entire classification result for the file is stored afterwards.

## Parameters

<i>FIL</i>	*fil Pointer to the file on the SD card.
<i>char*</i>	file_name Name of the file being classified, used for storing results.

## 4.40.2.4 de\_init\_nn()

```
int de_init_nn (
    void )
```

De-initializes the neural network and disables necessary hardware configurations.

This function is responsible for safely shutting down the neural network instance by destroying it and deallocating any associated resources. Additionally, it disables the CRC (Cyclic Redundancy Check) clock used by the STM32↔Cube.AI library.

## Returns

int Returns 0 on successful de-initialization or -1 if an error occurs during the destruction of the neural network.

## 4.40.2.5 init\_nn()

```
int init_nn (
    void )
```

Initializes the neural network for audio classification.

This function sets up the neural network by creating an instance of the model and initializing it with the required activation buffers. It retrieves the necessary input and output buffers for processing the audio data. Error handling is incorporated to manage potential initialization failures.

## Returns

int Returns 0 on successful initialization, or -1 if an error occurs during the network creation or initialization.

## Note

Driven from [https://wiki.st.com/stm32mcu/wiki/AI:How\\_to\\_perform\\_motion\\_sensing\\_on\\_STM32L4\\_IoTnode#Create\\_an\\_STM32Cube-AI\\_application\\_using\\_X-CUBE-AI](https://wiki.st.com/stm32mcu/wiki/AI:How_to_perform_motion_sensing_on_STM32L4_IoTnode#Create_an_STM32Cube-AI_application_using_X-CUBE-AI)

#### 4.40.2.6 run\_nn\_classification()

```
int run_nn_classification (
    ai_float * pSpectrogram,
    ai_float * classification_result)
```

Runs the neural network to classify audio data based on the provided spectrogram.

This function takes a pointer to a spectrogram array, processes it by normalizing with pre-calculated mean and standard deviation values, and then feeds it into the neural network. The output from the neural network is stored in the provided classification\_result array.

##### Parameters

<i>ai_float*</i>	pSpectrogram Pointer to the input spectrogram array.
<i>ai_float*</i>	classification_result Pointer to the array where the neural network's output (classification results) will be stored.

##### Returns

int Returns 0 on successful classification, or -1 if an error occurs during the network run or if the network handle is null.

#### 4.40.2.7 spectrogram\_generation\_init()

```
void spectrogram_generation_init (
    void )
```

Initializes the mel-spectrogram calculation lib which is part of the audio preprocessing.

This function sets up the components needed for generating Mel spectrograms from audio data.

##### Note

Derived from the FP-AI-SENSING feature pack: <https://www.st.com/en/embedded-software/fp-ai-sensing.html>

#### 4.40.2.8 spectrogram\_power\_to\_db()

```
void spectrogram_power_to_db (
    float * pSpectrogram)
```

Converts power values in a spectrogram to decibels (dB).

This function scans the spectrogram for the maximum power value to use as a reference for converting all power values to dB scale. It handles special cases where the maximum power is zero (silence) by setting all dB values to -80 dB to avoid division by zero. Values below -80 dB or resulting in NaN are also set to -80 dB to ensure numerical stability.

#### Parameters

<i>float</i>	*pSpectrogram Pointer to the spectrogram data array.
--------------	--

#### 4.40.2.9 store\_classification\_result()

```
void store_classification_result (
    FIL * fil,
    char * file_name,
    float * classification_result)
```

Stores the classification results into struct saved to a file.

This function is intended to write the classification results for an audiosample to the file containing the struct with all classification results as an object.

#### Parameters

<i>FIL</i>	*fil Pointer to the file structure representing the file containing the struct (with all classification results) as an object.
<i>char*</i>	file_name Name of the file where the classification results will be stored.
<i>float*</i>	classification_result Array containing the neural network's classification results to be stored.

#### Note

Unfortunately, this function was not implemented due to the full system integration never happening because the project reached its time limit.

### 4.40.3 Variable Documentation

#### 4.40.3.1 aiOutDataLabels

```
char aiOutDataLabels[AI_NETWORK_1_OUT_1_SIZE][10]
```

#### Initial value:

```
= {
    "bass",
    "pitched",
    "sustained",
    "rhythmic",
    "melodic"
}
```

Labels for the classification outputs of the neural network.

This array stores string labels corresponding to the classification categories of the neural network outputs.

## 4.41 neuronal\_network/esp\_cubeai\_test/Core/Src/audio\_classification.c File Reference

Audio classification implementation for embedded systems via stm32cube.ai (for testing purposes).

```
#include "audio_classification.h"
#include "featurescaler.h"
#include <math.h>
#include "feature_extraction.h"
#include "network_1.h"
#include "network_1_data.h"
#include "network_1_config.h"
#include "network_1_data_params.h"
#include "utils.h"
```

### Functions

- int `init_nn` ()  
*Initializes the neural network for audio classification.*
- int `run_nn_classification` (ai\_float \*pSpectrogram, ai\_float \*classification\_result)  
*Runs the neural network to classify audio data based on the provided spectrogram.*
- void `test_call_classification` ()  
*Tests the classification process by simulating the generation and processing of a spectrogram without actually requiring input data. Was used for previous tests.*
- void `Preprocessing_Init` (void)  
*Initializes the mel-spectrogram calculation lib which is part of the audio preprocessing.*
- void `frame_subsamples` (float32\_t \*subsample)  
*Splits an audiosubsample into overlapping frames ready for further processing.*
- void `preprocess_frame` (float \*pFrame)  
*Processes a single audio frame and incorporates it into the spectrogram.*
- void `process_subsample` (float \*pSpectrogram)  
*Processes a complete spectrogram, runs classification, and measures performance.*
- void `PowerToDb` (float \*pSpectrogram)  
*Converts power values in a spectrogram to decibels (dB).*

### Variables

- ai\_handle **network**  
*Handle to the neural network (stm32cube.ai) instance used for audio classification.*
- ai\_u8 **activations** [AI\_NETWORK\_1\_DATA\_ACTIVATIONS\_SIZE]  
*Buffer for storing activation data required by stm32cube.ai.*
- ai\_buffer \* **ai\_input**  
*Pointer to the input buffer of the stm32cube.ai.*
- ai\_buffer \* **ai\_output**  
*Pointer to the output buffer of stm32cube.ai, where classification results are stored.*
- ai\_float **spectrogram** [AI\_NETWORK\_1\_IN\_1\_SIZE]  
*Array to hold the input spectrogram data that feeds into the neural network.*
- float **aiOutData** [AI\_NETWORK\_1\_OUT\_1\_SIZE] = {0.0, 0.0, 0.0, 0.0, 0.0}  
*Output data from the neural network, containing classification results.*
- uint32\_t **spectrogram\_col\_index** = 0  
*Index to track the current column in the spectrogram buffer during data preprocessing.*
- float32\_t **mel\_spectrogram\_column\_buffer** [SPECTROGRAM\_ROWS]



### 4.41.1 Detailed Description

Audio classification implementation for embedded systems via stm32cube.ai (for testing purposes).

This file implements the methods and algorithms required for audio classification using neural networks on embedded systems via stm32cube.ai. It includes initialization and execution of a neural network, preprocessing of audio data to generate spectrograms, and the handling of neural network output.

#### Date

June 20, 2024

#### Author

Leon Braungardt

### 4.41.2 Function Documentation

#### 4.41.2.1 frame\_subsamples()

```
void frame_subsamples (
    float32_t * subsample)
```

Splits an audiosubsample into overlapping frames ready for further processing.

This function segments an audiosubsample into frames 1024 samples with a 512 sample overlap, preparing them for feature extraction (spectrogram calculation). Each frame is extracted based on the hop length and frame length definitions, with zero-padding applied where necessary. The frames are then passed to the `preprocess_frame` function for following processing steps.

#### Parameters

<code>float32_t*</code>	subsample Pointer to the array containing the audio subsample data.
-------------------------	---

#### 4.41.2.2 init\_nn()

```
int init_nn (
    void )
```

Initializes the neural network for audio classification.

This function sets up the neural network by creating an instance of the model and initializing it with the required activation buffers. It retrieves the necessary input and output buffers for processing the audio data. Error handling is incorporated to manage potential initialization failures.

#### Returns

int Returns 0 on successful initialization, or -1 if an error occurs during the network creation or initialization.

#### Note

Driven from [https://wiki.st.com/stm32mcu/wiki/AI:How\\_to\\_perform\\_motion\\_sensing\\_on\\_STM32L4\\_IoTnode#Create\\_an\\_STM32Cube-AI\\_application\\_using\\_X-CUBE-AI](https://wiki.st.com/stm32mcu/wiki/AI:How_to_perform_motion_sensing_on_STM32L4_IoTnode#Create_an_STM32Cube-AI_application_using_X-CUBE-AI)

#### 4.41.2.3 PowerToDb()

```
void PowerToDb (
    float * pSpectrogram)
```

Converts power values in a spectrogram to decibels (dB).

This function scans the spectrogram for the maximum power value to use as a reference for converting all power values to dB scale. It handles special cases where the maximum power is zero (silence) by setting all dB values to -80 dB to avoid division by zero. Values below -80 dB or resulting in NaN are also set to -80 dB to ensure numerical stability.

##### Parameters

<i>float</i>	*pSpectrogram Pointer to the spectrogram data array.
--------------	--

#### 4.41.2.4 preprocess\_frame()

```
void preprocess_frame (
    float * pFrame)
```

Processes a single audio frame and incorporates it into the spectrogram.

This function takes an audio frame, computes its Mel-scaled spectrogram column, and appends this column to the ongoing spectrogram array. Once the spectrogram is filled with the number of columns determined by SPECTROGRAM\_COLS, it triggers the calculation of the spectrogram.

##### Parameters

<i>float*</i>	pFrame Pointer to the array containing the audio frame data, which is 1024 samples in length.
---------------	---

##### Note

This function updates the global `spectrogram_col_index` to manage the position within the spectrogram array. When the spectrogram array is full, it calls `process_subsample` to handle the filled spectrogram and resets the index for new data.

#### 4.41.2.5 Preprocessing\_Init()

```
void Preprocessing_Init (
    void )
```

Initializes the mel-spectrogram calculation lib which is part of the audio preprocessing.

This function sets up the components needed for generating Mel spectrograms from audio data.

##### Note

Derived from the FP-AI-SENSING feature pack: <https://www.st.com/en/embedded-software/fp-ai-sensing.html>

#### 4.41.2.6 process\_subsample()

```
void process_subsample (  
    float * pSpectrogram)
```

Processes a complete spectrogram, runs classification, and measures performance.

This function is responsible for converting a power spectrogram into decibel scale, running the neural network to classify the audio data, and measuring the processing time in CPU cycles and milliseconds. It performs a series of operations: conversion of spectrogram values with the predefined scaler (imitating the standardization), neural network classification and performance tracking. The classification results are then contained in aiOutData.

**Parameters**

<i>float*</i>	pSpectrogram Pointer to the array holding the complete spectrogram data that will be processed and classified.
---------------	--

**4.41.2.7 run\_nn\_classification()**

```
int run_nn_classification (
    ai_float * pSpectrogram,
    ai_float * classification_result)
```

Runs the neural network to classify audio data based on the provided spectrogram.

This function takes a pointer to a spectrogram array, processes it by normalizing with pre-calculated mean and standard deviation values, and then feeds it into the neural network. The output from the neural network is stored in the provided classification\_result array.

**Parameters**

<i>ai_float*</i>	pSpectrogram Pointer to the input spectrogram array.
<i>ai_float*</i>	classification_result Pointer to the array where the neural network's output (classification results) will be stored.

**Returns**

int Returns 0 on successful classification, or -1 if an error occurs during the network run or if the network handle is null.

**4.41.2.8 test\_call\_classification()**

```
void test_call_classification (
    void )
```

Tests the classification process by simulating the generation and processing of a spectrogram without actually requiring input data. Was used for previous tests.

**Note**

The loop fixed at 32 iterations represents a placeholder for actual Mel spectrogram calculation based on audio input.

**4.42 featurescaler.h**

```
00001 /*
00002  * featurescaler.h
00003  *
00004  * Created on: May 16, 2024
00005  * Author: leon
00006  */
00007
00008 #ifndef APP_FEATURESCALER_H_
00009 #define APP_FEATURESCALER_H_
00010 #include "arm_math.h"
00011
00012 extern const float feature_scaler_mean[960];
00013 extern const float feature_scaler_std[960];
00014
00015
00016 #endif /* APP_FEATURESCALER_H_ */
```

## 4.43 test\_audio\_data.h

```
00001 /*
00002  * test_audio_data.c
00003  *
00004  * Created on: Jun 2, 2024
00005  * Author: leon
00006  */
00007
00008 #ifndef INC_TEST_AUDIO_DATA_C_
00009 #define INC_TEST_AUDIO_DATA_C_
00010
00011 extern const int16_t audio_data[];
00012 extern const int audio_data_length;
00013 extern const int audio_sample_rate;
00014
00015 #endif /* INC_TEST_AUDIO_DATA_C_ */
```

## 4.44 utils.h

```
00001 /*
00002  * utils.h
00003  *
00004  * Created on: Jul 3, 2024
00005  * Author: leon
00006  */
00007
00008 #ifndef INC_UTILS_H_
00009 #define INC_UTILS_H_
00010
00011 #include <stdint.h>
00012
00013
00014
00015 // enable the DWT cycle counter
00016 void EnableDWT(void);
00017
00018 // Functions to start and stop cycle measurement
00019 void StartCycleMeasurement(void);
00020 void StopCycleMeasurement(void);
00021
00022 // Function to get the measured cycle count
00023 uint32_t GetMeasuredCycles(void);
00024
00025 // Function to convert cycles to milliseconds
00026 uint32_t CyclesToMilliseconds(uint32_t cycles);
00027
00028 // Function to convert cycles to microseconds
00029 uint32_t CyclesToMicroseconds(uint32_t cycles);
00030
00031 #endif
```



# Index

- adcBuffer
  - interface.h, 23
  - main.c, 70
- adcDmaFlag
  - interface.h, 23
  - main.c, 70
- addFile
  - filemanager.c, 30
  - filemanager.h, 13
- aiOutDataLabels
  - audio\_classification.c, 85
- audio.c
  - fillHalfBufferFromSD, 60
  - generateSineWave, 60
  - HAL\_I2S\_TxCpltCallback, 61
  - HAL\_I2S\_TxHalfCpltCallback, 61
  - setPitchFactor, 61
- audio.h
  - fillHalfBufferFromSD, 40
  - generateSineWave, 40
  - HAL\_I2S\_TxHalfCpltCallback, 41
  - setPitchFactor, 41
- audio\_classification.c
  - aiOutDataLabels, 85
  - calculate\_spectrogram\_column, 82
  - calculate\_total\_classification\_result, 82
  - classify\_file, 83
  - de\_init\_nn, 83
  - frame\_subsamples, 87
  - init\_nn, 83, 87
  - PowerToDb, 87
  - preprocess\_frame, 88
  - Preprocessing\_Init, 88
  - process\_subsample, 88
  - run\_nn\_classification, 83, 90
  - spectrogram\_generation\_init, 84
  - spectrogram\_power\_to\_db, 84
  - store\_classification\_result, 85
  - test\_call\_classification, 90
- audioPreprocessor.c
  - downsample\_Block, 62
  - downsample\_to\_1024\_samples, 63
  - get\_number\_subsamples, 63
  - initFilter, 63
  - resampleFile, 64
- audioPreprocessor.h
  - downsample\_Block, 43
  - downsample\_to\_1024\_samples, 43
  - filter\_taps, 45
  - get\_number\_subsamples, 44
  - initFilter, 44
  - resampleFile, 44
- calculate\_spectrogram\_column
  - audio\_classification.c, 82
- calculate\_total\_classification\_result
  - audio\_classification.c, 82
- checkWav
  - wavPlayer.c, 76
  - wavPlayer.h, 56
- classify\_file
  - audio\_classification.c, 83
- compareADCValues
  - interface.c, 35
  - interface.h, 19
- cpu\_time.c
  - CyclesToMicroseconds, 65
  - CyclesToMilliseconds, 65
  - EnableDWT, 66
  - GetMeasuredCycles, 66
  - StartCycleMeasurement, 66
  - StopCycleMeasurement, 66
- cursorDown
  - filemanager.c, 31
  - filemanager.h, 14
- cursorUp
  - filemanager.c, 31
  - filemanager.h, 14
- CyclesToMicroseconds
  - cpu\_time.c, 65
- CyclesToMilliseconds
  - cpu\_time.c, 65
- de\_init\_nn
  - audio\_classification.c, 83
- display.c
  - DISPLAY\_WIDTH, 27
  - displayStrings, 27
  - drawFaderProzent, 28
  - renderSelectedFile, 28
  - scrollDown, 29
  - scrollUp, 29
- display.h
  - displayStrings, 10
  - drawFaderProzent, 10
  - renderSelectedFile, 11
- DISPLAY\_WIDTH
  - display.c, 27
- displayStrings

- display.c, [27](#)
- display.h, [10](#)
- downsample\_Block
  - audioPreprocessor.c, [62](#)
  - audioPreprocessor.h, [43](#)
- downsample\_to\_1024\_samples
  - audioPreprocessor.c, [63](#)
  - audioPreprocessor.h, [43](#)
- drawFaderProzent
  - display.c, [28](#)
  - display.h, [10](#)
- EnableDWT
  - cpu\_time.c, [66](#)
- Error\_Handler
  - main.c, [69](#), [72](#), [74](#)
  - main.h, [48](#), [50](#), [53](#)
- EXISTS
  - filemanager.h, [13](#)
- f401\_display\_encoder\_fader\_test/Core/Inc/display.h, [9](#), [11](#)
- f401\_display\_encoder\_fader\_test/Core/Inc/filemanager.h, [12](#), [16](#)
- f401\_display\_encoder\_fader\_test/Core/Inc/fonts.h, [17](#)
- f401\_display\_encoder\_fader\_test/Core/Inc/interface.h, [17](#), [24](#)
- f401\_display\_encoder\_fader\_test/Core/Inc/main.h, [47](#), [49](#)
- f401\_display\_encoder\_fader\_test/Core/Inc/ssd1306.h, [25](#)
- f401\_display\_encoder\_fader\_test/Core/Inc/util.h, [54](#)
- f401\_display\_encoder\_fader\_test/Core/Src/display.c, [26](#)
- f401\_display\_encoder\_fader\_test/Core/Src/filemanager.c, [30](#)
- f401\_display\_encoder\_fader\_test/Core/Src/interface.c, [33](#)
- f401\_display\_encoder\_fader\_test/Core/Src/main.c, [67](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Inc/audio.h, [39](#), [41](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Inc/audioPreprocessor.h, [42](#), [45](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Inc/cpu\_time.h, [46](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Inc/lowpass\_16kFilter.h, [46](#), [47](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Inc/main.h, [50](#), [51](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Inc/util.h, [54](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Inc/wavPlayer.h, [55](#), [59](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Src/audio.c, [59](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Src/audioPreprocessor.c, [62](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Src/cpu\_time.c, [64](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Src/lowpass\_16kFilter.c, [67](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Src/main.c, [71](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Src/util.c, [75](#)
- f401\_sd\_card\_audio\_codec\_test/Core/Src/wavPlayer.c, [76](#)
- File, [5](#)
- FileManager, [5](#)
- filemanager.c
  - addFile, [30](#)
  - cursorDown, [31](#)
  - cursorUp, [31](#)
  - initializeFileManager, [31](#)
  - safeCurrentFileName, [32](#)
  - selectFile, [32](#)
  - setCursor, [32](#)
- filemanager.h
  - addFile, [13](#)
  - cursorDown, [14](#)
  - cursorUp, [14](#)
  - EXISTS, [13](#)
  - initializeFileManager, [14](#)
  - MAX\_CLASSES, [13](#)
  - MAX\_FILES, [13](#)
  - safeCurrentFileName, [15](#)
  - selectFile, [15](#)
  - setCursor, [15](#)
- fileNamesSDCard
  - interface.h, [23](#)
  - main.c, [70](#)
- fillHalfBufferFromSD
  - audio.c, [60](#)
  - audio.h, [40](#)
- filter\_taps
  - audioPreprocessor.h, [45](#)
  - lowpass\_16kFilter.c, [67](#)
  - lowpass\_16kFilter.h, [47](#)
- fm
  - interface.h, [23](#)
  - main.c, [71](#)
- fno
  - interface.h, [24](#)
  - main.c, [71](#)
- FontDef, [6](#)
- frame\_subsamples
  - audio\_classification.c, [87](#)
- generateSineWave
  - audio.c, [60](#)
  - audio.h, [40](#)
- get\_number\_subsamples
  - audioPreprocessor.c, [63](#)
  - audioPreprocessor.h, [44](#)
- GetMeasuredCycles
  - cpu\_time.c, [66](#)
- HAL\_ADC\_ConvCpltCallback
  - interface.c, [35](#)
  - interface.h, [19](#)
- HAL\_GPIO\_EXTI\_Callback
  - interface.c, [35](#)
  - interface.h, [20](#)



- HAL\_I2S\_TxCpltCallback
  - audio.c, [61](#)
- HAL\_I2S\_TxHalfCpltCallback
  - audio.c, [61](#)
  - audio.h, [41](#)
- HAL\_TIM\_PeriodElapsedCallback
  - interface.c, [36](#)
  - interface.h, [20](#)
- init\_nn
  - audio\_classification.c, [83](#), [87](#)
- initFilter
  - audioPreprocessor.c, [63](#)
  - audioPreprocessor.h, [44](#)
- initializeFileManager
  - filemanager.c, [31](#)
  - filemanager.h, [14](#)
- initPlayer
  - wavPlayer.c, [77](#)
  - wavPlayer.h, [56](#)
- interface.c
  - compareADCValues, [35](#)
  - HAL\_ADC\_ConvCpltCallback, [35](#)
  - HAL\_GPIO\_EXTI\_Callback, [35](#)
  - HAL\_TIM\_PeriodElapsedCallback, [36](#)
  - listFiles, [36](#)
  - resetShownFiles, [37](#)
  - rotary\_enc\_count, [39](#)
  - screenInit, [37](#)
  - sortFiles, [38](#)
  - updateScreen, [38](#)
- interface.h
  - adcBuffer, [23](#)
  - adcDmaFlag, [23](#)
  - compareADCValues, [19](#)
  - fileNamesSDCard, [23](#)
  - fm, [23](#)
  - fno, [24](#)
  - HAL\_ADC\_ConvCpltCallback, [19](#)
  - HAL\_GPIO\_EXTI\_Callback, [20](#)
  - HAL\_TIM\_PeriodElapsedCallback, [20](#)
  - listFiles, [21](#)
  - resetShownFiles, [21](#)
  - screenInit, [22](#)
  - sortFiles, [22](#)
  - THREASHOLD, [19](#)
  - updateScreen, [22](#)
- lfn
  - main.c, [71](#)
- listFiles
  - interface.c, [36](#)
  - interface.h, [21](#)
- lowpass\_16kFilter.c
  - filter\_taps, [67](#)
- lowpass\_16kFilter.h
  - filter\_taps, [47](#)
- main
  - main.c, [69](#), [72](#), [74](#)
- main.c
  - adcBuffer, [70](#)
  - adcDmaFlag, [70](#)
  - Error\_Handler, [69](#), [72](#), [74](#)
  - fileNamesSDCard, [70](#)
  - fm, [71](#)
  - fno, [71](#)
  - lfn, [71](#)
  - main, [69](#), [72](#), [74](#)
  - SystemClock\_Config, [69](#), [73](#), [74](#)
  - writeStringToFile, [69](#)
- main.h
  - Error\_Handler, [48](#), [50](#), [53](#)
- MAX\_CLASSES
  - filemanager.h, [13](#)
- MAX\_FILES
  - filemanager.h, [13](#)
- neural\_network/esp\_cubeai\_for\_integration/Core/Inc/audio\_classification.h, [79](#)
- neural\_network/esp\_cubeai\_for\_integration/Core/Src/audio\_classification.c, [80](#)
- neural\_network/esp\_cubeai\_test/Core/Inc/audio\_classification.h, [80](#)
- neural\_network/esp\_cubeai\_test/Core/Inc/featurescaler.h, [90](#)
- neural\_network/esp\_cubeai\_test/Core/Inc/main.h, [52](#), [53](#)
- neural\_network/esp\_cubeai\_test/Core/Inc/test\_audio\_data.h, [91](#)
- neural\_network/esp\_cubeai\_test/Core/Inc/Utils.h, [91](#)
- neural\_network/esp\_cubeai\_test/Core/Src/audio\_classification.c, [86](#)
- neural\_network/esp\_cubeai\_test/Core/Src/main.c, [73](#)
- playButtonHandler
  - wavPlayer.c, [77](#)
  - wavPlayer.h, [57](#)
- populateWavHeader
  - wavPlayer.c, [77](#)
  - wavPlayer.h, [57](#)
- PowerToDb
  - audio\_classification.c, [87](#)
- preprocess\_frame
  - audio\_classification.c, [88](#)
- Preprocessing\_Init
  - audio\_classification.c, [88](#)
- process\_subsample
  - audio\_classification.c, [88](#)
- renderSelectedFile
  - display.c, [28](#)
  - display.h, [11](#)
- resampleFile
  - audioPreprocessor.c, [64](#)
  - audioPreprocessor.h, [44](#)
- resetShownFiles
  - interface.c, [37](#)

- interface.h, 21
- rotary\_enc\_count
  - interface.c, 39
- run\_nn\_classification
  - audio\_classification.c, 83, 90
- safeCurrentFileName
  - filemanager.c, 32
  - filemanager.h, 15
- screenInit
  - interface.c, 37
  - interface.h, 22
- scrollDown
  - display.c, 29
- scrollUp
  - display.c, 29
- selectFile
  - filemanager.c, 32
  - filemanager.h, 15
- setCursor
  - filemanager.c, 32
  - filemanager.h, 15
- setPitchFactor
  - audio.c, 61
  - audio.h, 41
- sortFiles
  - interface.c, 38
  - interface.h, 22
- spectrogram\_generation\_init
  - audio\_classification.c, 84
- spectrogram\_power\_to\_db
  - audio\_classification.c, 84
- SSD1306\_t, 7
- StartCycleMeasurement
  - cpu\_time.c, 66
- StopCycleMeasurement
  - cpu\_time.c, 66
- store\_classification\_result
  - audio\_classification.c, 85
- SystemClock\_Config
  - main.c, 69, 73, 74
- test\_call\_classification
  - audio\_classification.c, 90
- THREASHOLD
  - interface.h, 19
- uart\_printf
  - util.c, 75
- updateScreen
  - interface.c, 38
  - interface.h, 22
- util.c
  - uart\_printf, 75
- wav\_header\_t, 7
  - wavPlayer.h, 56
- wavLoad
  - wavPlayer.c, 78
- wavPlayer.h, 57
- wavPlay
  - wavPlayer.c, 78
  - wavPlayer.h, 58
- WavPlayer, 8
  - wavPlayer.h, 56
- wavPlayer.c
  - checkWav, 76
  - initPlayer, 77
  - playButtonHandler, 77
  - populateWavHeader, 77
  - wavLoad, 78
  - wavPlay, 78
  - wavPlayPitched, 78
- wavPlayer.h
  - checkWav, 56
  - initPlayer, 56
  - playButtonHandler, 57
  - populateWavHeader, 57
  - wav\_header\_t, 56
  - wavLoad, 57
  - wavPlay, 58
  - WavPlayer, 56
  - wavPlayPitched, 58
- wavPlayPitched
  - wavPlayer.c, 78
  - wavPlayer.h, 58
- writeStringToFile
  - main.c, 69