I M A (Intelligent Magic Audio)

Generated by Doxygen 1.11.0

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

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	*hi2c1*	Pointer to the I2C handle used for communication with the OLED display.
in	*strings*	Array of string pointers to be displayed.
in	*numStrings*	Total number of strings in the array.
in	*cursor_index*	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	*hi2c1*	Pointer to the I2C handle used for communication with the OLED display.
in	*prozent*	Pointer to a null-terminated string representing the fader percentage to be displayed.
in	*multiplikator*	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	*hi2c1*	Pointer to the I2C handle used for communication with the OLED display.
in	*filename*	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.

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

void displayStrings(I2C_HandleTypeDef *hi2c1, char** strings, uint8_t numStrings, uint8_t
      cursor_index);
void renderSelectedFile(I2C_HandleTypeDef *hi2c1, const char *filename);
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);
```

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	*fm*	Pointer to the `FileManager` structure to which the file will be added.
in	*filename*	The name of the file to be added to the `FileManager`.
in	*classes*	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	*fm*	Pointer to the `FileManager` structure whose `cursor_index` 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	*fm*	Pointer to the `FileManager` structure whose `cursor_index` 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	*fm*	Pointer to the `FileManager` 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	*fm*	Pointer to the `FileManager` 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	*fm*	Pointer to the `FileManager` structure in which the `current_file_index` will be set to the value of `cursor_index`.

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	*fm*	Pointer to the `FileManager` structure used to adjust the cursor position.

4.4 filemanager.h

Go to the documentation of this file.

```
#ifndef FILE_MANAGER_H
#define FILE_MANAGER_H

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

#define MAX_FILES 500
#define MAX_FILENAME_LENGTH 30

#define MAX_CLASSES 4

#define EXISTS 0
#define NOT_EXISTS -1

typedef struct {
    char filename[MAX_FILENAME_LENGTH];
    float classes[MAX_CLASSES];
} File;

typedef struct {
    File files[MAX_FILES];
    File shownFiles[MAX_FILES];
    char current_cursor_filename[25];
    int num_files;
    int num_matched_files;
    int cursor_index;
    int current_file_index;
    float fader_Class[MAX_CLASSES];
} FileManager;

```

```
void initializeFileManager(FileManager *fm);
void addFile(FileManager *fm, const char *filename, const float *classes);
void selectFile(FileManager *fm);
void cursorUp(FileManager *fm);
void cursorDown(FileManager *fm);
void safeCurrentFileName(FileManager *fm);
void setCursor(FileManager *fm);



#endif /* FILE_MANAGER_H */
```

4.5 fonts.h

```
#ifndef _FONTS_H
#define _FONTS_H

#include <stdint.h>

//
//  Structure used to define fonts
//
typedef struct {
    const uint8_t FontWidth;    /* Font width in pixels */
    uint8_t FontHeight;         /* Font height in pixels */
    const uint16_t *data;       /* Pointer to data font data array */
} FontDef;

//
//  Export the 3 available fonts
//
extern FontDef Font_7x10;
extern FontDef Font_11x18;
extern FontDef Font_16x26;

#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. Threashold 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	*hadc*	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	*GPIO_pin*	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`	*htim*	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`	*path*	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(*) 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	*fileNames*	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	*void*	

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 fileNamesSDCard

```
char fileNamesSDCard[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.

```

#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"

#define THREASHOLD               0.1f

#define EXISTS                  0

#define NOT_EXISTS              -1

#define SAME                    0

#define NOT_SAME                -1

#define SMOOTHING_HEIGHT        30000


//extern parameters from main
//-------------------------------------------------------------------------

extern ADC_HandleTypeDef hadc1;
extern DMA_HandleTypeDef hdma_adc1;
extern I2C_HandleTypeDef hi2c1;
extern TIM_HandleTypeDef htim3;
extern TIM_HandleTypeDef htim5;

extern FileManager fm;
extern FileManager fmCopy;

extern FATFS FatFs;
extern FILINFO fno;
extern DIR dir;
extern FRESULT fres;
extern FIL file;


extern uint32_t adcBuffer[NUM_CHANNELS];
extern char fileNamesSDCard[MAX_FILES][MAX_FILENAME_LENGTH];
extern int fileCount;

extern bool adcDmaFlag;
extern bool updateScreenFlag;
extern bool sortFilesFlag;

//------------------------------------------------------------------------------

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc);
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);

void screenInit(char (*fileNames)[MAX_FILENAME_LENGTH]);
void updateScreen(void);
void sortFiles(void);
void compareADCValues(void);
void resetShownFiles(void);
void writeFileManagerOnSD(void);
void readFileManagerFromSD(void);
void listFiles(const char *path);

```

4.8 ssd1306.h

```

#ifndef _SSD1306_H
#define _SSD1306_H

#include "stm32f4xx_hal.h"
#include "fonts.h"

// I2c address
#ifndef SSD1306_I2C_ADDR
#define SSD1306_I2C_ADDR        0x78
#endif // SSD1306_I2C_ADDR

// SSD1306 width in pixels
#ifndef SSD1306_WIDTH
#define SSD1306_WIDTH           128
#endif // SSD1306_WIDTH

// SSD1306 LCD height in pixels
#ifndef SSD1306_HEIGHT
#define SSD1306_HEIGHT          128
#endif // SSD1306_HEIGHT

#ifndef SSD1306_COM_LR_REMAP
#define SSD1306_COM_LR_REMAP    0
#endif // SSD1306_COM_LR_REMAP

#ifndef SSD1306_COM_ALTERNATIVE_PIN_CONFIG
#define SSD1306_COM_ALTERNATIVE_PIN_CONFIG    1
#endif // SSD1306_COM_ALTERNATIVE_PIN_CONFIG


#define SH110X_MEMORYMODE 0x20
#define SH110X_COLUMNADDR 0x21
#define SH110X_PAGEADDR 0x22
#define SH110X_SETCONTRAST 0x81
#define SH110X_CHARGEPUMP 0x8D
#define SH110X_SEGREMAP 0xA0
#define SH110X_DISPLAYALLON_RESUME 0xA4
#define SH110X_DISPLAYALLON 0xA5
#define SH110X_NORMALDISPLAY 0xA6
#define SH110X_INVERTDISPLAY 0xA7
#define SH110X_SETMULTIPLEX 0xA8
#define SH110X_DCDC 0xAD
#define SH110X_DISPLAYOFF 0xAE
#define SH110X_DISPLAYON 0xAF
#define SH110X_SETPAGEADDR                                                     \
  0xB0
#define SH110X_COMSCANINC 0xC0
#define SH110X_COMSCANDEC 0xC8
#define SH110X_SETDISPLAYOFFSET 0xD3
#define SH110X_SETDISPLAYCLOCKDIV 0xD5
#define SH110X_SETPRECHARGE 0xD9
#define SH110X_SETCOMPINS 0xDA
#define SH110X_SETVCOMDETECT 0xDB
#define SH110X_SETDISPSTARTLINE                                                \
  0xDC

#define SH110X_SETLOWCOLUMN 0x00
#define SH110X_SETHIGHCOLUMN 0x10
#define SH110X_SETSTARTLINE 0x40

//
//  Enumeration for screen colors
```

```
//
typedef enum {
    Black = 0x00,   // Black color, no pixel
    White = 0x01,   // Pixel is set. Color depends on LCD
} SSD1306_COLOR;

//
//  Struct to store transformations
//
typedef struct {
    uint16_t CurrentX;
    uint16_t CurrentY;
    uint8_t Inverted;
    uint8_t Initialized;
} SSD1306_t;

//
//  Function definitions
//

uint8_t ssd1306_Init(I2C_HandleTypeDef *hi2c);
void ssd1306_UpdateScreen(I2C_HandleTypeDef *hi2c);
void ssd1306_Fill(SSD1306_COLOR color);
void ssd1306_DrawPixel(uint8_t x, uint8_t y, SSD1306_COLOR color);
char ssd1306_WriteChar(char ch, FontDef Font, SSD1306_COLOR color);
char ssd1306_WriteString(const char* str, FontDef Font, SSD1306_COLOR color);
void ssd1306_SetCursor(uint8_t x, uint8_t y);
void ssd1306_InvertColors(void);

#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	*hi2c1*	Pointer to the I2C handle used for communication with the OLED display.
in	*strings*	Array of string pointers to be displayed.
in	*numStrings*	Total number of strings in the array.
in	*cursor_index*	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	*hi2c1*	Pointer to the I2C handle used for communication with the OLED display.
in	*prozent*	Pointer to a null-terminated string representing the fader percentage to be displayed.
in	*multiplikator*	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	*hi2c1*	Pointer to the I2C handle used for communication with the OLED display.
in	*filename*	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	*numStrings*	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	*fm*	Pointer to the `FileManager` structure to which the file will be added.
in	*filename*	The name of the file to be added to the `FileManager`.
in	*classes*	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	*fm*	Pointer to the `FileManager` structure whose `cursor_index` 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	*fm*	Pointer to the `FileManager` structure whose `cursor_index` 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	*fm*	Pointer to the FileManager 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	*fm*	Pointer to the FileManager 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	*fm*	Pointer to the FileManager structure in which the `current_file_index` will be set to the value of `cursor_index`.

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	*fm*	Pointer to the `FileManager` 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	*hadc*	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	*GPIO_pin*	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	*htim*	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	*path*	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`	*fileNames*	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`	*void*	

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_TxHalfCpltCallback (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

player	Pointer to a `WavPlayer` structure containing file information and pitch factor.
pitched	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

frequency	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

hi2s	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

newPitchFactor	The new pitch factor to set.

4.13 audio.h

Go to the documentation of this file.

```

#ifndef INC_AUDIO_H_
#define INC_AUDIO_H_

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

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

#define BUFFER_SIZE 256

#define HALF_BUFFER_SIZE (BUFFER_SIZE / 2)

```

```
#define BUFFER_SIZE_INPUT 4096

extern I2S_HandleTypeDef hi2s2;
extern int16_t dacData[BUFFER_SIZE];
extern int16_t fileReadBuf[BUFFER_SIZE_INPUT]; // actually for one halfBuffer, now is = BUFFER_SIZE,
to allow 2x playback speed
extern bool dma_dataReady;
extern volatile int16_t *outBufPtr;

extern volatile float pitchFactor;
extern volatile bool pitchChanged;

struct WavPlayer;

// Callbacks for DMA Complete
void HAL_I2S_TxHalfCpltCallback(I2S_HandleTypeDef *hi2s);
void HAL_I2SS_TxCpltCallback(I2S_HandleTypeDef *hi2s);

// fills half of the buffer with next chunk of data from file
// returns the number of bytes read
uint16_t fillHalfBufferFromSD(struct WavPlayer *player, bool pitched);

// Test Functions
void generateSineWave(double frequency);
void initSineTable();

// Pitch functions
void setPitchFactor(float newPitchFactor);

int16_t interpolate(int16_t *buffer, float index);

#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 ∗wave←HeaderResampled)

  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

src	Pointer to the source buffer containing 16-bit audio samples to be processed.
dest	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_←ROUNDED = NUM_SAMPLES_CHUNK_IN

Parameters

file	Pointer to the input file (`FIL` structure) containing audio data.
outChunk	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

file	Pointer to the WAV file (`FIL` 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

inFil	Pointer to the input file (`FIL` structure) containing original audio data.
outFil	Pointer to the output file (`FIL` structure) to write resampled audio data.
waveHeaderInput	Pointer to the WAV header of the input file.
waveHeaderResampled	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.

```

#ifndef RESAMPLE_H
#define RESAMPLE_H

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

// Constants
#define DECIMATION_FACTOR_ROUNDED 3 // Rounded up from 44.1kHz / 16kHz = 2.75625
#define NUM_CHANNELS 2

#define NUM_SAMPLES_CHUNK_OUT 1024
#define NUM_SAMPLES_CHUNK_IN ((NUM_SAMPLES_CHUNK_OUT * DECIMATION_FACTOR_ROUNDED * NUM_CHANNELS + 1) & ~1)
#define RESAMPLE_CHUNK_IN_SIZE (NUM_SAMPLES_CHUNK_IN * sizeof(int16_t))
#define RESAMPLE_CHUNK_OUT_SIZE (NUM_SAMPLES_CHUNK_OUT * sizeof(int16_t))
#define FRAME_SIZE 1024
#define HOP_SIZE 512
#define STEP_SIZE (FRAME_SIZE - HOP_SIZE)
// External filter coefficients (defined in your filter file)
extern float32_t filter_taps[NUM_TAPS];

// Function declarations
arm_status initFilter();
uint16_t resampleFile(FIL *inFil, FIL *outFil, wav_header_t *waveHeaderInput, wav_header_t *waveHeaderResampled);
void downsample_Block(int16_t *src, int16_t *dest);
uint16_t get_number_subsamples(FIL *file);
uint32_t downsample_to_1024_samples(FIL *file, int16_t outChunk[NUM_SAMPLES_CHUNK_OUT]);

#endif // RESAMPLE_H
```

4.16 cpu_time.h

```
#ifndef CPU_TIME_H
#define CPU_TIME_H

#include <stdint.h>



// enable the DWT cycle counter
void EnableDWT(void);

// Functions to start and stop cycle measurement
void StartCycleMeasurement(void);
void StopCycleMeasurement(void);

// Function to get the measured cycle count
uint32_t GetMeasuredCycles(void);

// Function to convert cycles to milliseconds
uint32_t CyclesToMilliseconds(uint32_t cycles);

// Function to convert cycles to microseconds
uint32_t CyclesToMicroseconds(uint32_t cycles);

#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.

```

#ifndef LOWPASS_16KFILTER_H_
#define LOWPASS_16KFILTER_H_

#include "arm_math.h"


#define NUM_TAPS 57

#define BLOCK_SIZE 33

extern float32_t filter_taps[NUM_TAPS];

#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

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.

```c
/* USER CODE BEGIN Header */
/* USER CODE END Header */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
#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

/* USER CODE BEGIN Private defines */
#define NUM_CHANNELS 5
/* USER CODE END Private defines */

#ifdef __cplusplus
}
```

```
#endif

#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

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.

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

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
#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

/* USER CODE BEGIN Private defines */
#define SD_SPI_HANDLE hspi2
/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#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

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.

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

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f7xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <arm_math.h>
/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
#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

/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

4.25 util.h

```
#ifndef UTIL_H
#define UTIL_H

#include "main.h"
#include <stdarg.h> //for va_list var arg functions
#include <stdio.h>
#include <string.h>
// #include "stm32f4xx_hal.h"

extern UART_HandleTypeDef huart2;

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

#endif
```

4.26 util.h

```
#ifndef UTIL_H
#define UTIL_H

#include "main.h"
#include <stdarg.h> //for va_list var arg functions
#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

player	Pointer to the `WavPlayer` 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,
            FIL * 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

player	Pointer to the `WavPlayer` structure to initialize.
file	Pointer to the `FIL` object representing the WAV file.
wavHeader	Pointer to the `wav_header_t` 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 `playback↩Active` flag to `true`. If playback is already active, it sets the `restartPlayback` flag to `true` to restart playback.

Parameters

player	Pointer to the `WavPlayer` 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

file	Pointer to the `FIL` object representing the WAV file.
wavHeader	Pointer to the `wav_header_t` 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

player	Pointer to the `WavPlayer` structure that will be used to manage the WAV file.
filename	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

player	Pointer to the `WavPlayer` 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

player	Pointer to the `WavPlayer` 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.

```
#ifndef INC_WAVPLAYER_H_
#define INC_WAVPLAYER_H_

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

typedef struct wav_header_t {
    // RIFF Header
    uint32_t ChunkID;          // Contains "RIFF"
    uint32_t ChunkSize;        // Size of the wav portion of the file, which follows the first 8 bytes.
File size - 8
    uint32_t Format;           // Contains "WAVE"

    // Format Header
    uint32_t Subchunk1ID;        // Contains "fmt " (includes trailing space)
    uint32_t Subchunk1Size;      // Should be 16 for PCM
    uint16_t AudioFormat;        // Should be 1 for PCM. 3 for IEEE Float
    uint16_t NumChannels ;
    uint32_t SampleRate;
    uint32_t ByteRate;         // Number of bytes per second. sample_rate * num_channels * Bytes Per
Sample
    uint16_t BlockAlign;       // num_channels * Bytes Per Sample
    uint16_t BitsPerSample;      // Number of bits per sample

    // Data
    uint32_t Subchunk2ID;        // Contains "data"
    uint32_t Subchunk2Size ;     // Number of bytes in data. Number of samples * num_channels * sample
byte size
    // uint8_t bytes[];        // Remainder of wave file is bytes
} wav_header_t;

typedef struct WavPlayer {
    volatile bool restartPlayback;
    volatile bool playbackActive;
    FIL *file;
    wav_header_t *wavHeader;
    uint32_t headerSize;
    float pitchFactor;
    bool pitchChanged;
} WavPlayer;

void initPlayer(WavPlayer *player, FIL *file, wav_header_t *wavHeader);

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

uint32_t populateWavHeader(FIL *file, wav_header_t *wavHeader);
uint8_t checkWav(WavPlayer *player);

uint8_t wavPlay(WavPlayer *player);
uint8_t wavPlayPitched(WavPlayer *player);

void playButtonHandler(WavPlayer *player);

void wavStop(void);

#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

player	Pointer to a `WavPlayer` structure containing file information and pitch factor.
pitched	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

frequency	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

hi2s	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

hi2s	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

newPitchFactor	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 ∗wave←HeaderResampled)

  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

src	Pointer to the source buffer containing 16-bit audio samples to be processed.
dest	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 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_↩ ROUNDED = NUM_SAMPLES_CHUNK_IN

Parameters

file	Pointer to the input file (`FIL` structure) containing audio data.
outChunk	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

file	Pointer to the WAV file (`FIL` 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

inFil	Pointer to the input file (`FIL` structure) containing original audio data.
outFil	Pointer to the output file (`FIL` structure) to write resampled audio data.
waveHeaderInput	Pointer to the WAV header of the input file.
waveHeaderResampled	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

cycles	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

cycles	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 lfn [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

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	*filename*	The name of the file to write to. (Note: This parameter is not used in the current implementation.)
in	*str*	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

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

None	

4.34.2.2 main()

```
int main (
            void )
```

The application entry point.

Return values

int	

4.34.2.3 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

4.35 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

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

None	

4.35.2.2 main()

```
int main (
            void )
```

The application entry point.

Return values

int	

4.35.2.3 SystemClock_Config()

```
void SystemClock_Config (
            void )
```

System Clock Configuration.

Return values

None	

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

fmt	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

player	Pointer to the `WavPlayer` 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,
            FIL * 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

player	Pointer to the `WavPlayer` structure to initialize.
file	Pointer to the `FIL` object representing the WAV file.
wavHeader	Pointer to the `wav_header_t` 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 `playback←Active` flag to `true`. If playback is already active, it sets the `restartPlayback` flag to `true` to restart playback.

Parameters

player	Pointer to the `WavPlayer` structure managing the audio playback.

4.37.2.4 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

file	Pointer to the `FIL` object representing the WAV file.
wavHeader	Pointer to the `wav_header_t` 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

player	Pointer to the `WavPlayer` structure that will be used to manage the WAV file.
filename	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

player	Pointer to the `WavPlayer` 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

player	Pointer to the `WavPlayer` 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

```
/*
 * audio_classification.h
 *
 *  Created on: Jun 20, 2024
 *      Author: leon
 */

#ifndef AUDIO_CLASSIFICATION_H_
#define AUDIO_CLASSIFICATION_H_

/* includes */
#include "feature_extraction.h"
#include "featurescaler.h"
#include "network_1.h"
#include "network_1_data.h"
#include "network_1_config.h"
#include "network_1_data_params.h"

/* defines */
// define spectrogram dimensions
#define NUM_MEL_BANDS 30
#define SPECTROGRAM_ROWS NUM_MEL_BANDS
#define SPECTROGRAM_COLS 32

// define data preprocessing variables
#define TARGET_SAMPLE_RATE 16000
#define FRAME_LENGTH 1024
#define HOP_LENGTH 512
#define N_SAMPLES_PER_SUBSAMPLE 16896
#define N_FRAMES_PER_SUBSAMPLE SPECTROGRAM_COLS


/* function definitions */
int init_nn(void);
int de_init_nn(void);
int run_nn_classification(float* pSpectrogram, float* classification_result);
void spectrogram_generation_init(void);
void classify_file(FIL *fil, char* file_name)
void calculate_spectrogram_column(float* frame, int col_index);
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);
void spectrogram_power_to_db(float32_t *spectrogram);
void store_classification_result(FIL *fil, char* file_name, float* classification_result)

// stm32cube.ai related variables
extern ai_handle network;
extern ai_u8 activations[AI_NETWORK_1_DATA_ACTIVATIONS_SIZE];
extern ai_buffer * ai_input;
extern ai_buffer * ai_output;

// own input and output variables for nn classification
extern float spectrogram[AI_NETWORK_1_IN_1_SIZE];
extern float aiOutData[AI_NETWORK_1_OUT_1_SIZE];

// variable for mapping aiOutData indices to the actual labels
extern char aiOutDataLabels[AI_NETWORK_1_OUT_1_SIZE][10];


// variables for spectrogram generation parameters
static arm_rfft_fast_instance_f32 S_Rfft;
static SpectrogramTypeDef         S_Spectr;
static MelSpectrogramTypeDef      S_MelSpectr;
static MelFilterTypeDef           S_MelFilter;

#endif
```

4.39 audio_classification.h

```
/*
 * audio_classification.h
 *
 *  Created on: Jun 20, 2024
 *      Author: leon
 */

#ifndef AUDIO_CLASSIFICATION_H_
#define AUDIO_CLASSIFICATION_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"

/* defines */
// define spectrogram dimensions
#define NUM_MEL_BANDS 30
#define SPECTROGRAM_ROWS NUM_MEL_BANDS
#define SPECTROGRAM_COLS 32

// define data preprocessing variables
#define TARGET_SAMPLE_RATE 16000
#define FRAME_LENGTH 1024
#define HOP_LENGTH 512
#define N_SAMPLES_PER_SUBSAMPLE 16896
#define N_FRAMES_PER_SUBSAMPLE SPECTROGRAM_COLS

/* function definitions */
int init_nn(void);
int run_nn_classification(float* pSpectrogram, float* classification_result);
void test_call_classification(void);
void Preprocessing_Init(void);
int resample_audio(const short* input_data, int input_length, int input_sample_rate, float32_t*
output_data, int* output_length);
void frame_subsamples(float32_t* subsample);
void preprocess_frame(float* frame);
void process_subsample(float* pSpectrogram);
void PowerToDb(float32_t *spectrogram);

// stm32cube.ai related variables
extern ai_handle network;
extern ai_u8 activations[AI_NETWORK_1_DATA_ACTIVATIONS_SIZE];
extern ai_buffer * ai_input;
extern ai_buffer * ai_output;

// own input and output variables for nn classification
extern ai_float spectrogram[AI_NETWORK_1_IN_1_SIZE];
extern ai_float aiOutData[AI_NETWORK_1_OUT_1_SIZE];

// counter variable for current column number of the mel spectrogram
extern uint32_t spectrogram_col_index;

// variables for spectrogram generation parameters
static arm_rfft_fast_instance_f32 S_Rfft;
static SpectrogramTypeDef         S_Spectr;
static MelSpectrogramTypeDef      S_MelSpectr;
static MelFilterTypeDef           S_MelFilter;

#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

float∗	pFrame Pointer to the audio frame.
int	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

float∗	total_file_classification_result Array to store the final averaged classification results.
float	classification_results_subsamples[][AI_NETWORK_1_OUT_1_SIZE] Two-dimensional array containing classification results for each subsample.
int	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

FIL	∗fil Pointer to the file on the SD card.
char∗	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

Drived 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

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

ai_float∗	pSpectrogram Pointer to the input spectrogram array.
ai_float∗	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-sens
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

float	∗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

FIL	∗fil Pointer to the file structure representing the file containing the struct (with all classification results) as an object.
char∗	file_name Name of the file where the classification results will be stored.
float∗	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

float32↩ _t∗	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

Drived 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`

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

float	*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

*float**	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-sens
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

float∗	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

ai_float∗	pSpectrogram Pointer to the input spectrogram array.
ai_float∗	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

```
/*
 * featurescaler.h
 *
 *  Created on: May 16, 2024
 *      Author: leon
 */

#ifndef APP_FEATURESCALER_H_
#define APP_FEATURESCALER_H_
#include "arm_math.h"

extern const float feature_scaler_mean[960];
extern const float feature_scaler_std[960];


#endif /* APP_FEATURESCALER_H_ */
```

4.43 test_audio_data.h

```
/*
 * test_audio_data.c
 *
 *  Created on: Jun 2, 2024
 *      Author: leon
 */

#ifndef INC_TEST_AUDIO_DATA_C_
#define INC_TEST_AUDIO_DATA_C_

extern const int16_t audio_data[];
extern const int audio_data_length;
extern const int audio_sample_rate;

#endif /* INC_TEST_AUDIO_DATA_C_ */
```

4.44 utils.h

```
/*
 * utils.h
 *
 *  Created on: Jul 3, 2024
 *      Author: leon
 */

#ifndef INC_UTILS_H_
#define INC_UTILS_H_

#include <stdint.h>



// enable the DWT cycle counter
void EnableDWT(void);

// Functions to start and stop cycle measurement
void StartCycleMeasurement(void);
void StopCycleMeasurement(void);

// Function to get the measured cycle count
uint32_t GetMeasuredCycles(void);

// Function to convert cycles to milliseconds
uint32_t CyclesToMilliseconds(uint32_t cycles);

// Function to convert cycles to microseconds
uint32_t CyclesToMicroseconds(uint32_t cycles);

#endif
```

Index