An

Project Report

On

Audio Player Using Microcontroller

Submitted in

Partial fulfilment of

PG Diploma in Embedded Systems and Design

By

Mr. Tejas Sunil Suryawanshi
Mr. Pratik Nemane
Mr. Pranav Patil
Mrs. Darshana Joshi
Mrs. Dipashree Patil

At

SUNBEAM INFOTECH PRIVATE LIMITED , HINJEWADI

CIRTIFICATE

This is to certify that Project

Audio Player Using Microcontroller

Has been Successfully submitted by

Mr. Tejas Sunil Suryawanshi

In Partial fulfilment of **PG Diploma in Embedded Systems and Design** been Satisfactorily carried as per prescribed by **CDAC ACTS, PUNE.**

Place: Hinjewadi Date:

Authorized Signature

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who supported and contributed to the successful completion of the "Audio Player Using Microcontroller" project. Without their assistance, guidance, and encouragement, this project would not have been possible.

First and foremost, I would like to thank Professor Mr. Nilesh Ghule, our project supervisor and the Technical Director at SUNBEAM Infotech Private Limited, for their invaluable guidance, expert advice, and unwavering support throughout the project. Your insights and mentorship were instrumental in shaping the project's direction.

I extend my heartfelt thanks to my Project mates who collaborated tirelessly throughout the project. Your dedication, hard work, and technical expertise were crucial in overcoming challenges and achieving our goals.

I would like to acknowledge Mr. Devendra Dhande, our course coordinator and all Lab mentors at SUNBEAM Infotech Private Limited, for there generous support and guidance, which significantly contributed to the project's success.

I would also like to thank SUNBEAM Infotech Private Limited for providing an environment conducive to learning and innovation. The resources and facilities offered by the institute played a crucial role in the development and testing phases of this project.

Thank you to everyone who played a part in this project. Your contributions have been invaluable, and I am deeply grateful for your assistance.

Date: Tejas Sunil Suryawanshi

ABSTRACT

With the advancement in semiconductor technology, scope for development of embedded systems has increased manifolds. New processors with improved computing capabilities and low power consumption have further accelerated the developments in embedded domain. Consumers are looking for affordable multimedia devices with high performance and durability making embedded developers to think creatively and use all resources at hand to meet the desired user specifications.

This is one such attempt by designing an Audio Player to play the wave audio files from a USB flash drive in the same format.

INDEX

List of Figure

INTRODUCTION

1.1 Introduction

STM32 microcontrollers offer a large number of serial and parallel communication peripherals which can be interfaced with all kinds of electronic components. Audio player using STM32F407 Discovery board interfaced to a USB device and using on chip DAC.

Music is stored on an USB device connected to STM32F407 board on I2S in the form of WAV files. Music is one of the best ways to relieve pressure in stressful modern society life. An audio file is a type of file which stores different sounds which were recorded.

These audio files store the digital form of these sounds and they are converted into Analog format to play them. there are 3 categories of audio files formats in real world. The purpose of this project is to develop a player which can play the mainstream file format. Which can be controlled by mobile application. Meanwhile, this mobile application can play, pause and repeat with play button and next button according to sets requirement as well as set up songs.

1.2 Overview Of Project

An Audio Player which uses STM32F407DISC1 discovery board which takes Wave Audio files from the connected USB drive and play it with the help of DMA controller and I2S protocols. The audio is controlled (by having pause, resume, next, previous and stop features) with the help of a user switch which is on the board.

Playing wave audio files which are the basic high quality audio format because these are also uncompressed files, the quality of the output is nowhere compromised. Thus, if one is seeking to play music with a single device, the device thus developed may come very handy.

With the improvements in processor technology and development of software design environments for the same, designing of an embedded system has not remained a very tedious task as it was a decade ago. Processors such ARM which have very good computing performances are becoming more popular among embedded developers.

LITRETURE SURVEY

2.1 Purpose of Project:

The Audio Player is the only hardware which plays the audio from a recorded file. These provide excitement, comfort, well-being and reduce depression to its listeners. In general, these players should be designed with some specifications like

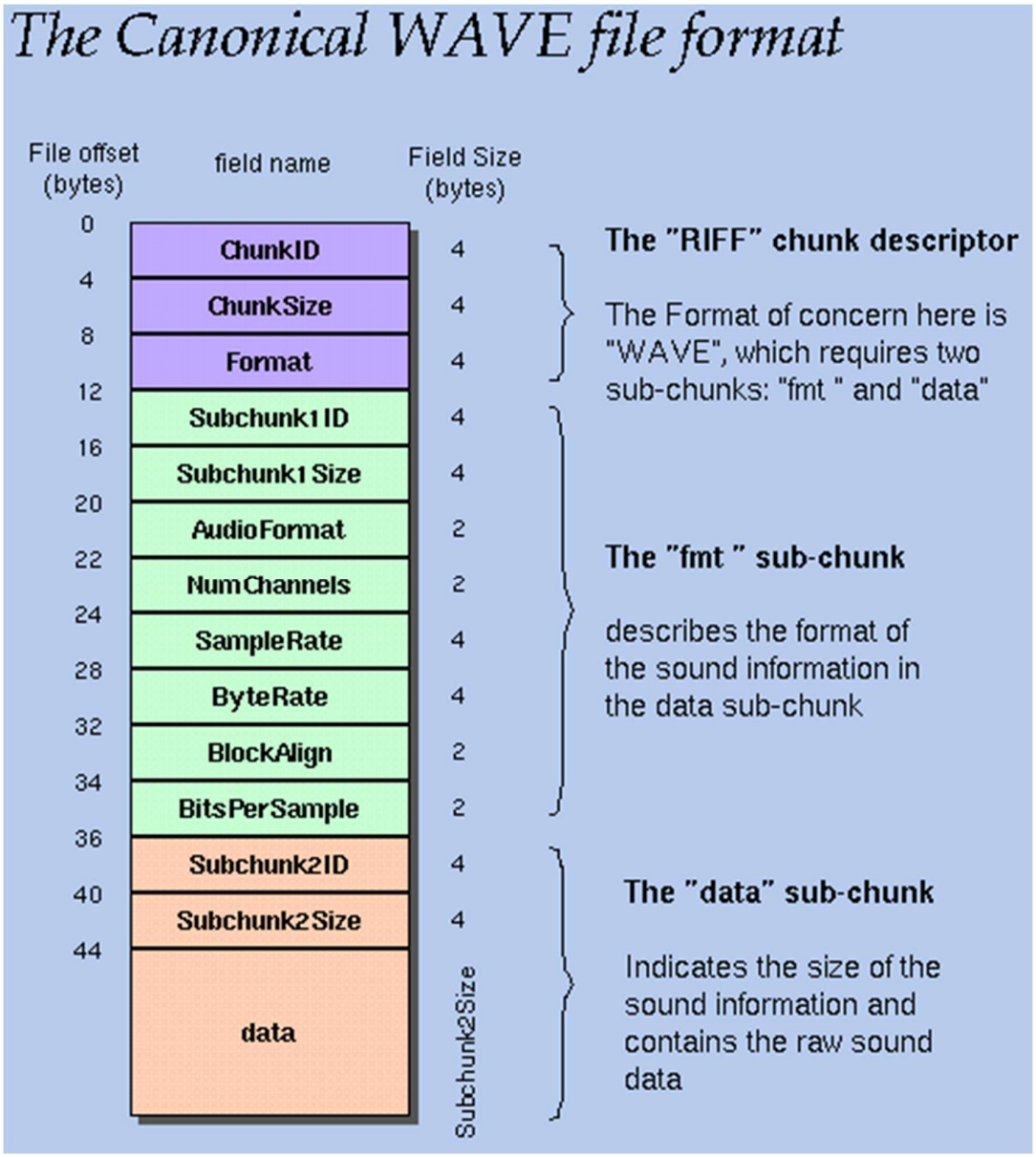
The clarity of the audio output should be more with minimal noise so that humans can understand the sounds. The output shouldn't affect the medical conditions of a person at the maximum volume (especially for heart patients). It should cancel the external noise while playing an audio with minimal volume. There should be no data loss in the system. Sometimes high-quality audio is required to match the exact sound in the real world. This is mostly followed by audio players used for professional audio needs. Recently everyone is focusing on wireless hardware and mostly they want to control each and every hardware. To develop an application-based Audio player, which works according to the control given by command icons provided in application. With the help of Stm32F407 board, for communication between USB drive and speaker and use audio files in .wav format.

- **.WAV Format :**

The WAV file is an instance of a Resource Interchange File Format (RIFF) defined by IBM and Microsoft. The RIFF format acts as a wrapper for various audio coding formats.

Though a WAV file can contain compressed audio, the most common WAV audio format is uncompressed audio in the linear pulse-code modulation (LPCM) format. LPCM is also the standard audio coding format for audio CDs, which store two-channel LPCM audio sampled at 44.1 kHz with 16 bits per sample. Since LPCM is uncompressed and retains all of the samples of an audio track, professional users or audio experts may use the WAV format with LPCM audio for maximum audio quality. WAV files can also be edited and manipulated with relative ease using software.

On Microsoft Windows, the WAV format supports compressed audio using the Audio Compression Manager (ACM). Any ACM codec can be used to compress a WAV file. The user interface (UI) for Audio Compression Manager may be accessed through various programs that use it, including Sound Recorder in some versions of Windows.



2.1 .wav format

- **Fatfs:**

The family of FAT file systems is supported by almost all operating systems for personal computers, including all versions of Windows and MS-DOS/PC DOS, OS/2, and DR-DOS. (PC DOS is an OEM version of MS-DOS, MS-DOS was originally based on SCP's 86-DOS. DR-DOS was based on Digital Research's Concurrent DOS, a successor of CP/M-86.) The FAT file systems are therefore well-suited as a universal exchange format between computers and devices of most any type and age.

The FAT file system traces its roots back to an (incompatible) 8-bit FAT precursor in Standalone Disk BASIC and the short-lived MDOS/MIDAS project.

Over the years, the file system has been expanded from FAT12 to FAT16 and FAT32. Various features have been added to the file system including subdirectories, codepage support, extended attributes, and long filenames. Third parties such as Digital Research have incorporated optional support for deletion tracking, and volume/directory/file-based multi-user security schemes to support file and directory passwords and permissions such as read/write/execute/delete access rights. Most of these extensions are not supported by Windows.

The FAT12 and FAT16 file systems had a limit on the number of entries in the root directory of the file system and had restrictions on the maximum size of FAT-formatted disks or partitions.

FAT32 addresses the limitations in FAT12 and FAT16, except for the file size limit of close to 4 GB, but it remains limited compared to NTFS. FAT12, FAT16 and FAT32 also have a limit of eight characters for the file name, and three characters for the extension (such as .exe). This is commonly referred to as the 8.3 filename limit. VFAT, an optional extension to FAT12, FAT16 and FAT32, introduced in Windows 95 and Windows NT 3.5, allowed long file names (LFN) to be stored in the FAT file system in a backwards compatible fashion.

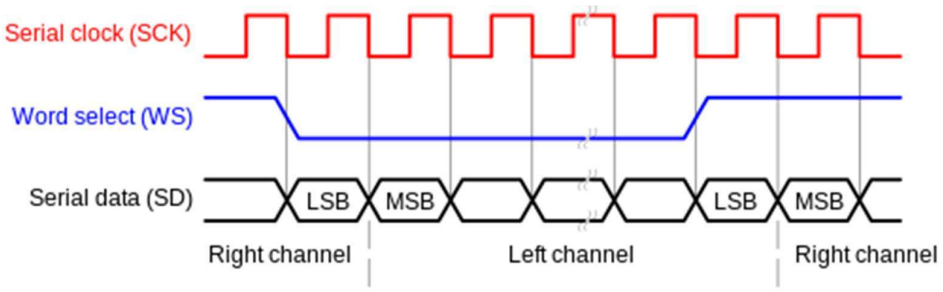
Features:

- DOS/Windows Compatible FAT/exFAT Filesystem.
- Platform Independent. Easy to port.
- Very Small Footprint for Program Code and Work Area.
- Various Configuration Options to Support for:
- Long File Name in ANSI/OEM or Unicode.
- exFAT Filesystem, 64-bit LBA and GPT for Huge Storages.
- Thread Safe for RTOS.
- Multiple Volumes. (Physical Drives and Partitions).
- We have used FAT32 file system in this project.

- **I²S (Inter-IC Sound):**

pronounced "eye-squared-ess"[citation needed]), is an electrical serial bus interface standard used for connecting digital audio devices together. It is used to communicate PCM audio data between integrated circuits in an electronic device. The I²S bus separates clock and serial data signals, resulting in simpler receivers than those required for asynchronous communications systems that need to recover the clock from the data stream. Alternatively I²S is spelled I2S (pronounced eye-two-ess) or IIS (pronounced eye-eye-ess). Despite the similar name, I²S is unrelated to the bidirectional I²C (IIC) bus.

This standard was introduced in 1986 by Philips Semiconductor (now NXP Semiconductors) and was first revised June 5, 1996. The standard was last revised on February 17, 2022 and updated terms master and slave to controller and target.



2.2Timing diagram of I²S

The I²S protocol outlines one specific type of PCM digital audio communication with defined parameters outlined in the Philips specification.

The bus consists of at least three lines:

- Bit clock line

Officially "continuous serial clock (SCK)". Typically written "bit clock (BCLK)".

- Word clock line

Officially "word select (WS)". Typically called "left-right clock (LRCLK)" or "frame sync (FS)".

0 = Left channel, 1 = Right channel

- At least one multiplexed data line Officially "serial data (SD)" but can be called SDATA, SDIN, SDOUT, DACDAT, ADCDAT, etc.

It may also include the following lines:

Master clock (typically 256 x LRCLK)

This is not part of the I2S standard, but is commonly included for synchronizing the internal operation of the analog/digital converters.

A multiplexed data line for upload

The bit clock pulses once for each discrete bit of data on the data lines. The bit clock frequency is the product of the sample rate, the number of bits per channel and the number of channels. So, for example, CD Audio with a sample frequency of 44.1 kHz, with 16 bits of precision and two channels (stereo) has a bit clock frequency of:

44.1 kHz × 16 × 2 = 1.4112 MHz

The word select clock lets the device know whether channel 0 or channel 1 is currently being sent, because I²S allows two channels to be sent on the same data line. It is a 50% duty-cycle signal that has the same frequency as the sample frequency. For stereo material, the I²S specification states that left audio is transmitted on the low cycle of the word select clock and the right channel is transmitted on the high cycle. It is typically synchronized to the falling edge of the serial clock, as the data is latched on the rising edge. The word select clock changes one bit clock period before the MSB is transmitted. This enables, for example, the receiver to store the previous word and clear the input for the next.

Data is signed, encoded as two's complement with the MSB (most significant bit) first. This allows the number of bits per frame to be

arbitrary, with no negotiation required between transmitter and receiver.

- **I2C (Inter-Integrated Circuit):**

It is a bus interface connection protocol incorporated into devices for serial communication. It was originally designed by Philips Semiconductor in 1982. Recently, it is a widely used protocol for short-distance communication. It is also known as Two Wired Interface(TWI).

Working of I2C Communication Protocol :
It uses only 2 bi-directional open-drain lines for data communication called SDA and SCL. Both these lines are pulled high.

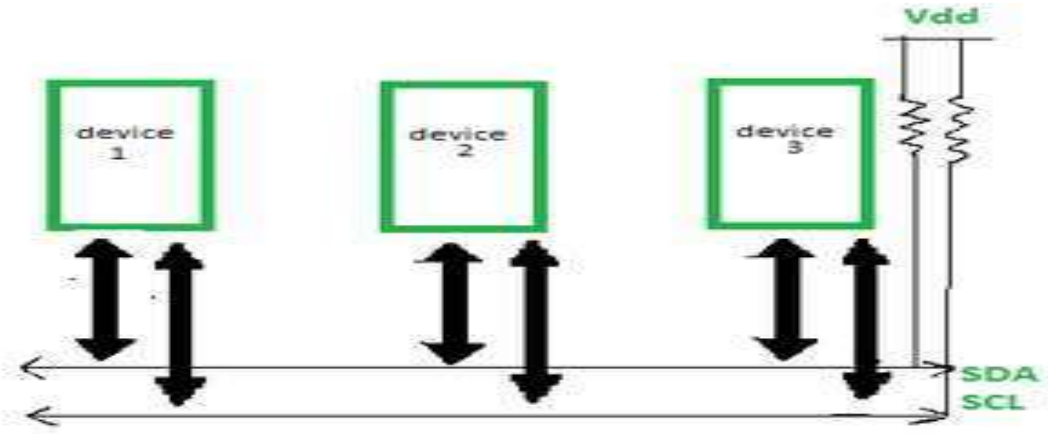
Serial Data (SDA) – Transfer of data takes place through this pin.
Serial Clock (SCL) – It carries the clock signal.

I2C operates in 2 modes –

Master mode

Slave mode

Each data bit transferred on SDA line is synchronized by a high to the low pulse of each clock on the SCL line.



2.3 General circuit of I2C

According to I2C protocols, the data line can not change when the clock line is high, it can change only when the clock line is low. The 2 lines

are open drain, hence a pull-up resistor is required so that the lines are high since the devices on the I2C bus are active low. The data is transmitted in the form of packets which comprises 9 bits. The sequence of these bits are –

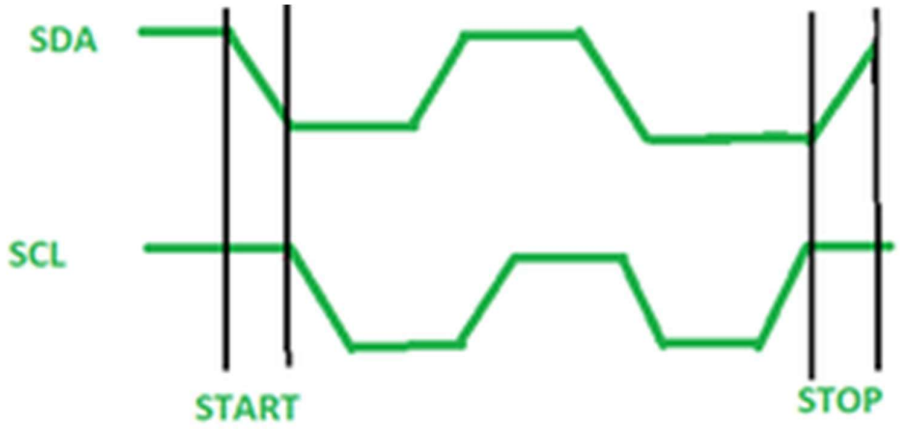
Start Condition – 1 bit

Slave Address – 8 bit

Acknowledge – 1 bit

Start and Stop Conditions :

START and STOP can be generated by keeping the SCL line high and changing the level of SDA. To generate START condition the SDA is changed from high to low while keeping the SCL high. To generate STOP condition SDA goes from low to high while keeping the SCL high, as shown in the figure below.



2.4Start and Stop Condition

Repeated Start Condition :

Between each start and stop condition pair, the bus is considered as busy and no master can take control of the bus. If the master tries to initiate a new transfer and does not want to release the bus before starting the new transfer, it issues a new START condition. It is called a REPEATED START condition.

Read/Write Bit :

A high Read/Write bit indicates that the master is sending the data to the slave, whereas a low Read/Write bit indicates that the master is receiving data from the slave.

ACK/NACK Bit :

After every data frame, follows an ACK/NACK bit. If the data frame is received successfully then ACK bit is sent to the sender by the receiver.

Addressing :

The address frame is the first frame after the start bit. The address of the slave with which the master wants to communicate is sent by the master to every slave connected with it. The slave then compares its own address with this address and sends ACK.

I2C Packet Format :

In the I2C communication protocol, the data is transmitted in the form of packets. These packets are 9 bits long, out of which the first 8 bits are put in SDA line and the 9th bit is reserved for ACK/NACK i.e. Acknowledge or Not Acknowledge by the receiver.

START condition plus address packet plus one more data packet plus STOP condition collectively form a complete Data transfer.

Features of I2C Communication Protocol :

Half-duplex Communication Protocol – Bi-directional communication is possible but not simultaneously.

Synchronous Communication – The data is transferred in the form of frames or blocks.

Can be configured in a multi-master configuration.

Clock Stretching – The clock is stretched when the slave device is not ready to accept more data by holding the SCL line low, hence disabling the master to raise the clock line. Master will not be able to raise the clock line because

the wires are AND wired and wait until the slave releases the SCL line to show it is ready to transfer next bit.

Arbitration –
I2C protocol supports multi-master bus system but more than one bus can not be used simultaneously. The SDA and SCL are monitored by the masters. If the SDA is found high when it was supposed to be low it will be inferred that another master is active and hence it stops the transfer of data.

Serial transmission –
I2C uses serial transmission for transmission of data.

Used for low-speed communication.

Advantages :

Can be configured in multi-master mode.

Complexity is reduced because it uses only 2 bi-directional lines (unlike SPI Communication).

Cost-efficient.

It uses ACK/NACK feature due to which it has improved error handling capabilities.

Limitations :

Slower speed.

Half-duplex communication is used in the I2C communication protocol.

For our Project we used I2C protocol to control DAC chip.

- **Direct memory access (DMA)** :

Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems to access main system memory independently of the central processing unit (CPU).

Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write

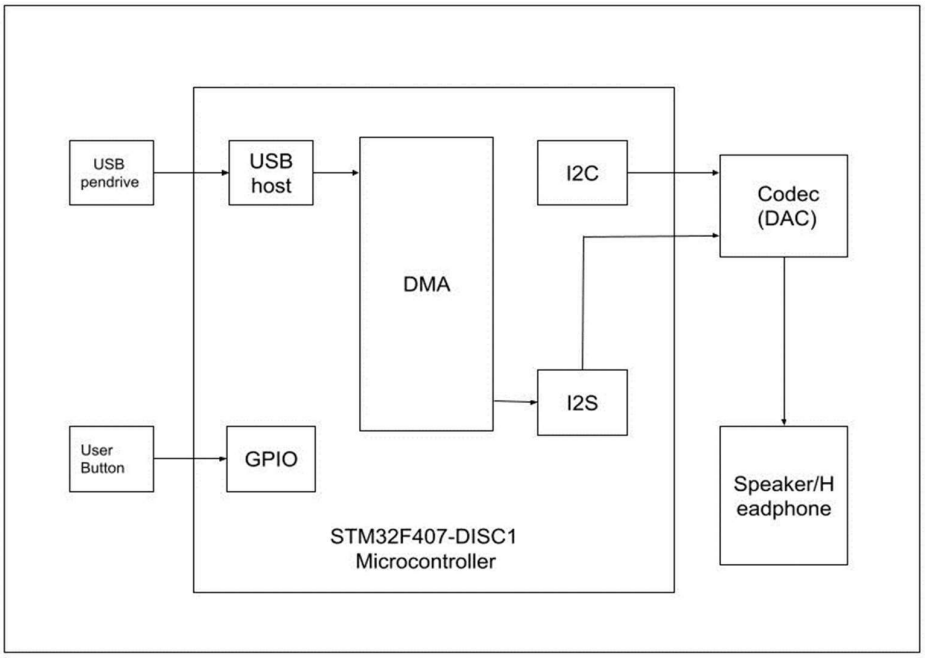
operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller (DMAC) when the operation is done. This feature is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in some multi-core processors. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without DMA channels. Similarly, a processing circuitry inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

DMA can also be used for "memory to memory" copying or moving of data within memory. DMA can offload expensive memory operations, such as large copies or scatter-gather operations, from the CPU to a dedicated DMA engine. An implementation example is the I/O Acceleration Technology. DMA is of interest in network-on-chip and in-memory computing architectures.

In our Project We used DMA to directly access data from our external USB flash drive.

PROJECT DEVELOPMENT

3.1 Block Diagram:



3.1 Block Diagram

Major blocks of this project is shown above in Block diagram

The major blocks of our projects are

1. I2C protocol:

   This is used to control the DAC chip CS43L22 that is give on STM32F407DISC1 Board it is used to control DAC operations like start DAC, stop DAC, etc

2. I2S protocol:

   This Protocol is used for connecting the audio devices together it process the Audio data in .wav and convert it to Digital form this DATA is then Given To the DAC to convert it in to Anaglog signal.

3.DAC:

The main work of converting the audio data given by I2S to the Analog signal is done by this unit the so that it can fed to speaker or headphones .IC CS43L22 is the DAC provided by STM on their Discovery Board is being used for this.

3. DMA:

It is used to Map the memory from USB flash drive where the audio file is saved directly to the RAM so that CPU can focus on other important tasks rather than getting stuck on data transfer only.

4. USB HOST FS:

It is enabled to use the USB OTG functionality available on Board by enabling this we can use the USB connector for mass storage class.

5. GPIO:

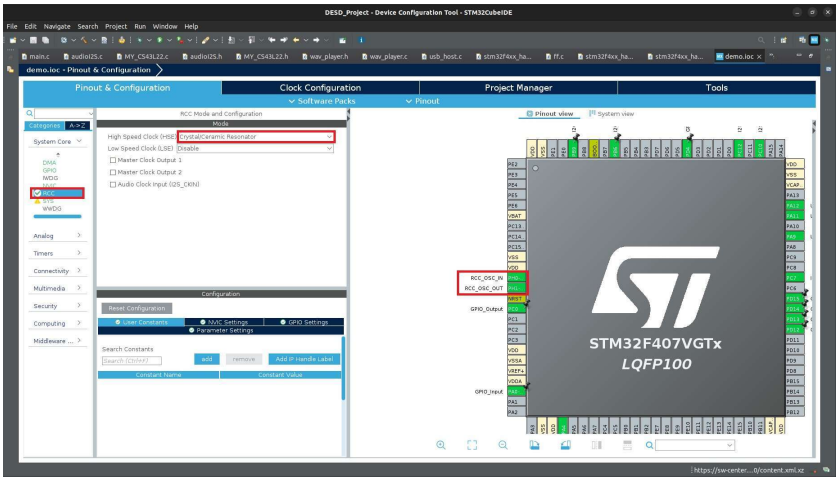
The GPIO button present on the pin no 0 of PORT A is used as a GPIO INPUT to take input from used.

We are connecting USB drive to STM32 Discovery Board (STM32F407G-DISC1) and the module is powered up by connecting it to the power supply of 5v. Connect the speaker to AUX port of the board using an AUX cable. Connect a Pen-drive to STM32 board's micro USB port directly (if the drive has a micro USB end) or use a Converter if it doesn't has it. Make sure there are audio files with .WAV format. STM32 has a CS43L22 chip which acts as a DAC and receives data either in Analog mode or I2S mode. In addition, this chip requires I2C protocol to control the chip operations

3.2 Configurations

- **RCC:**

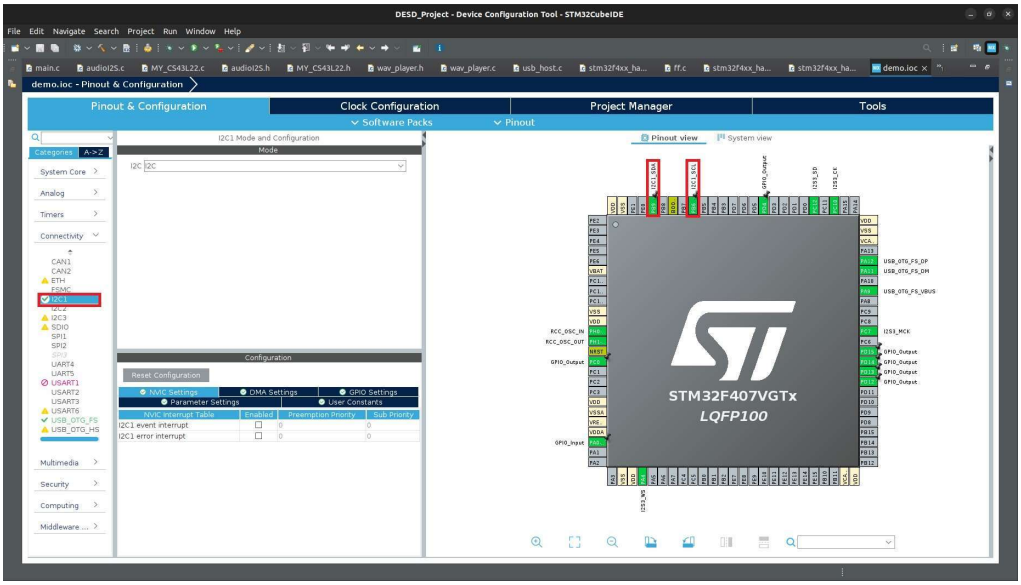
  we have selected external clock for this application as per the schematic for audio given by STM



3.2 RCC configuration

- I2C:

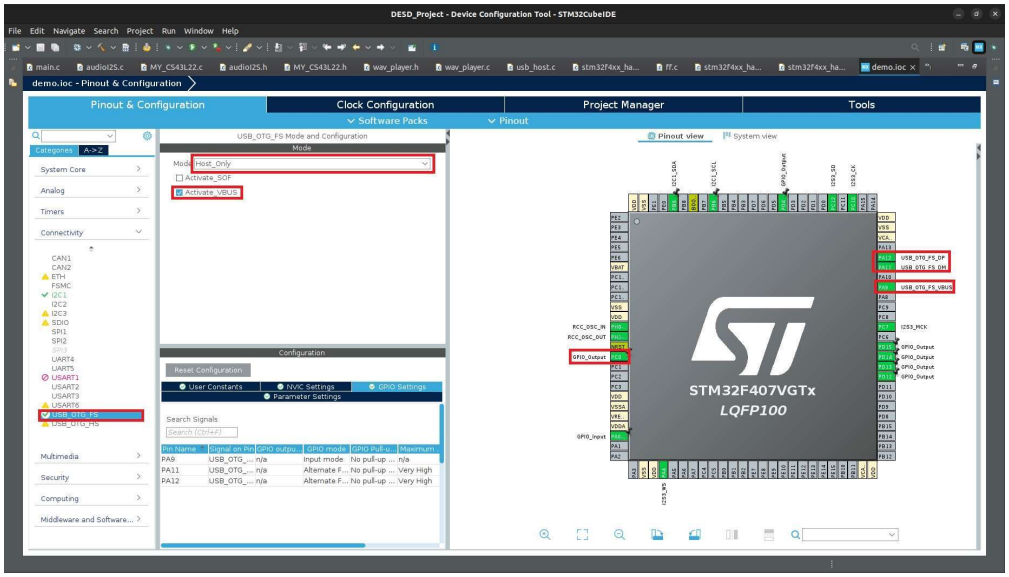
  We have selected the I2C1



3.3 I2C configuration

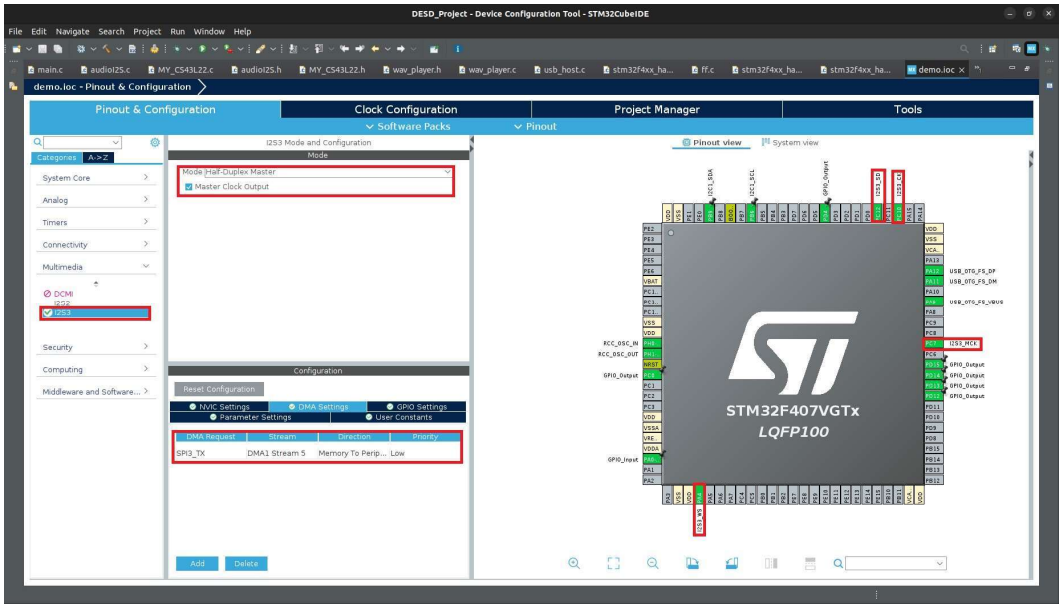
- USB OTG FS:

  We have used USB OTG FS in HOST ONLY mode with Active VBUS as per Audio Schematic of STM



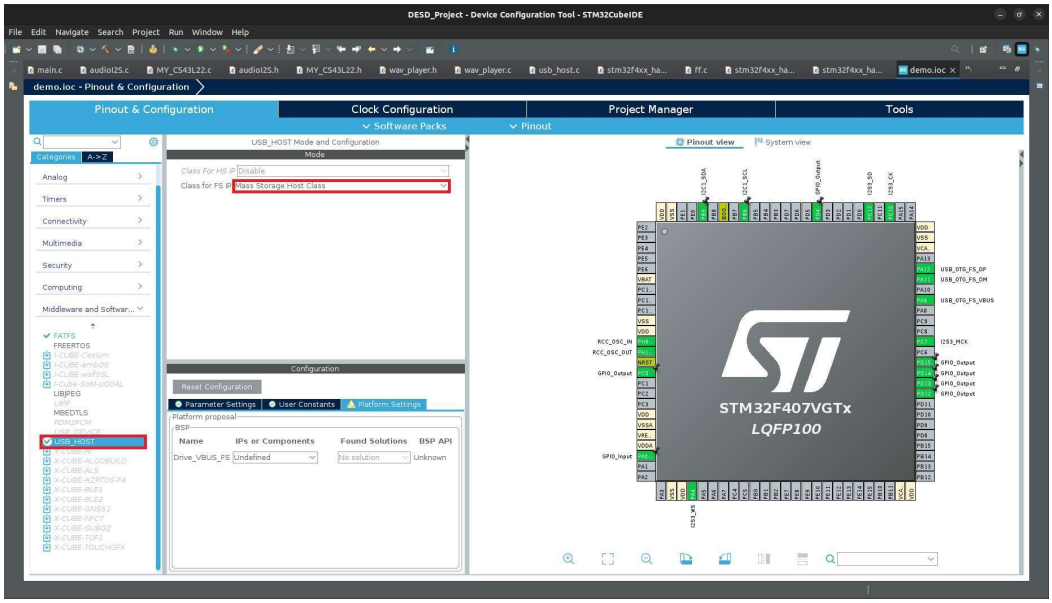
3.4 USB HOST FS

- I2S:
  We used I2S with HALF DUPLEX MASTER mode with DMA.



3.5 I2S configuration

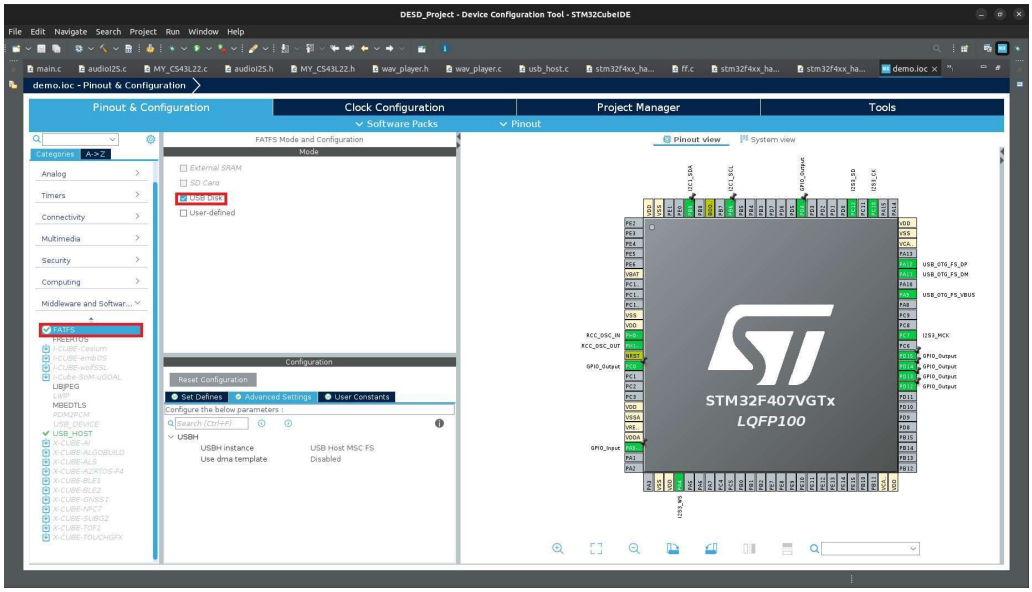
- USB HOST:
  We selected USB Host class as mass storage class.



3.6 USB HOST configuration

- FATFS:

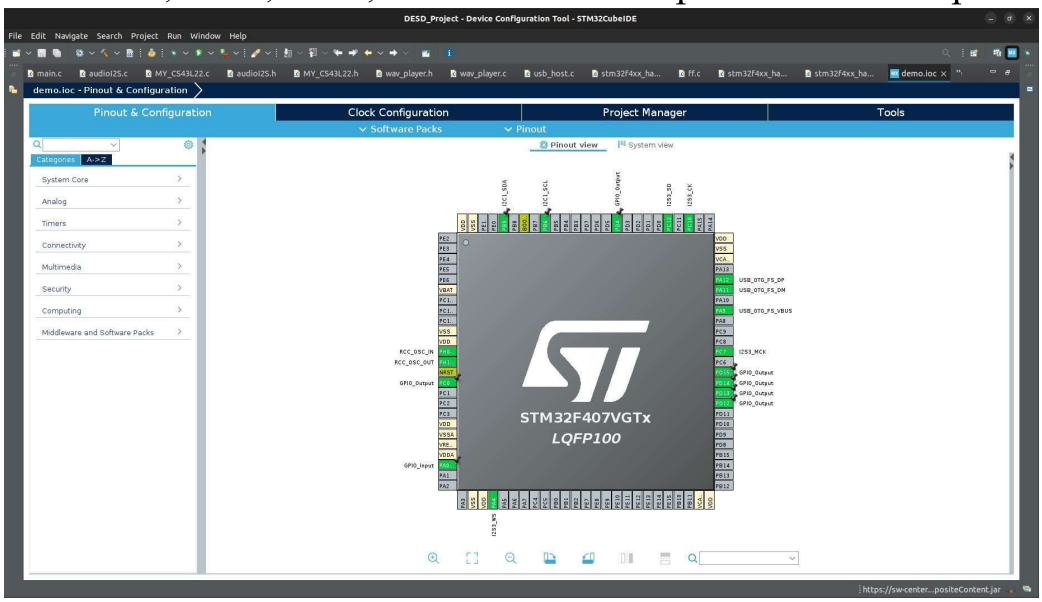
We have selected USB HOST mode in FATFS.



3.7 FATFS configuration

- GPIO:

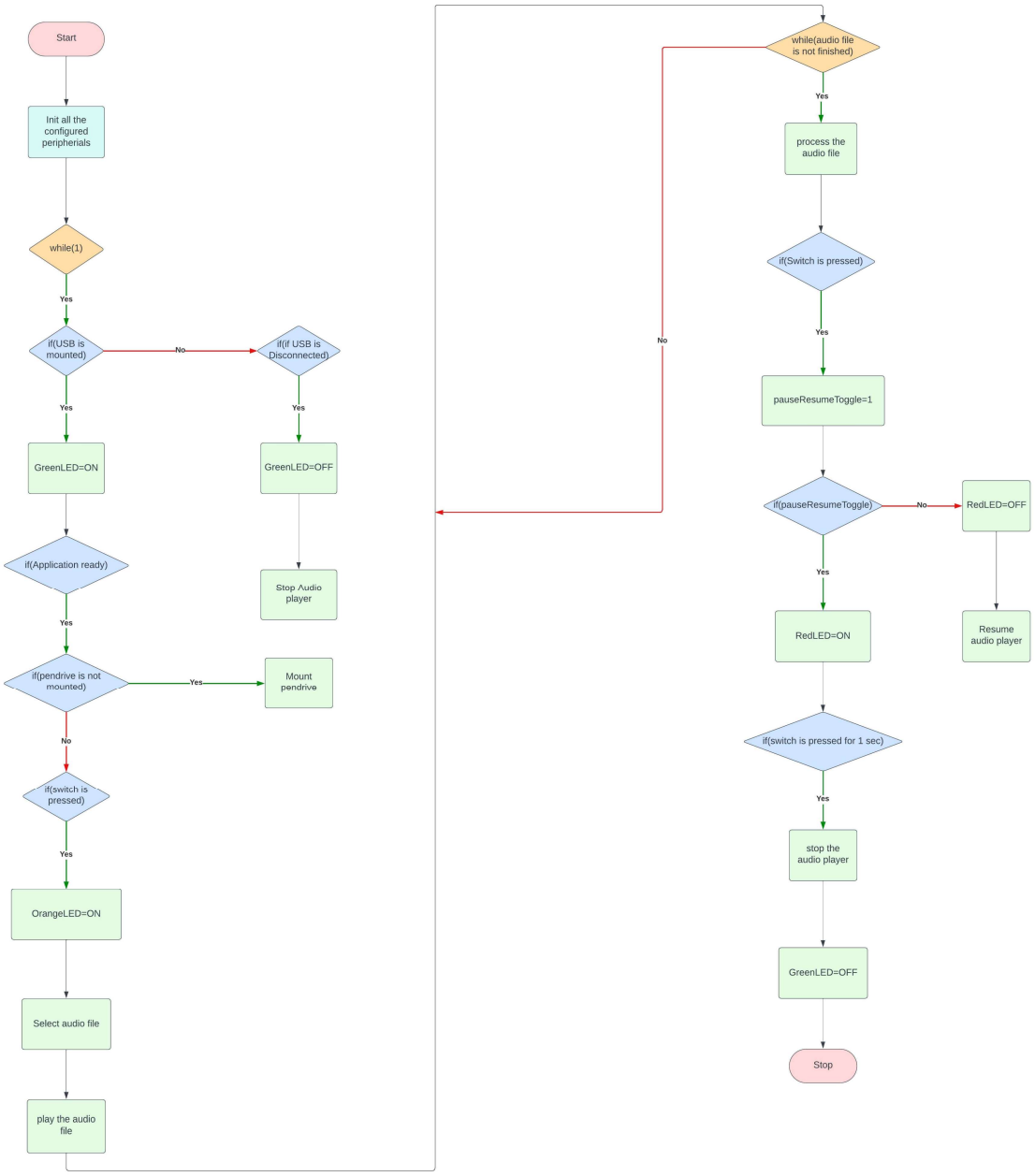
We have selected PA0 as GPIO INPUT this is User switch pin and PD12,PD13,PD14,PD15 as GPIO output this are LED pin.



3.8 GPIO configuration

3.3 Flowchart:

This is the functional flowchart of the code.



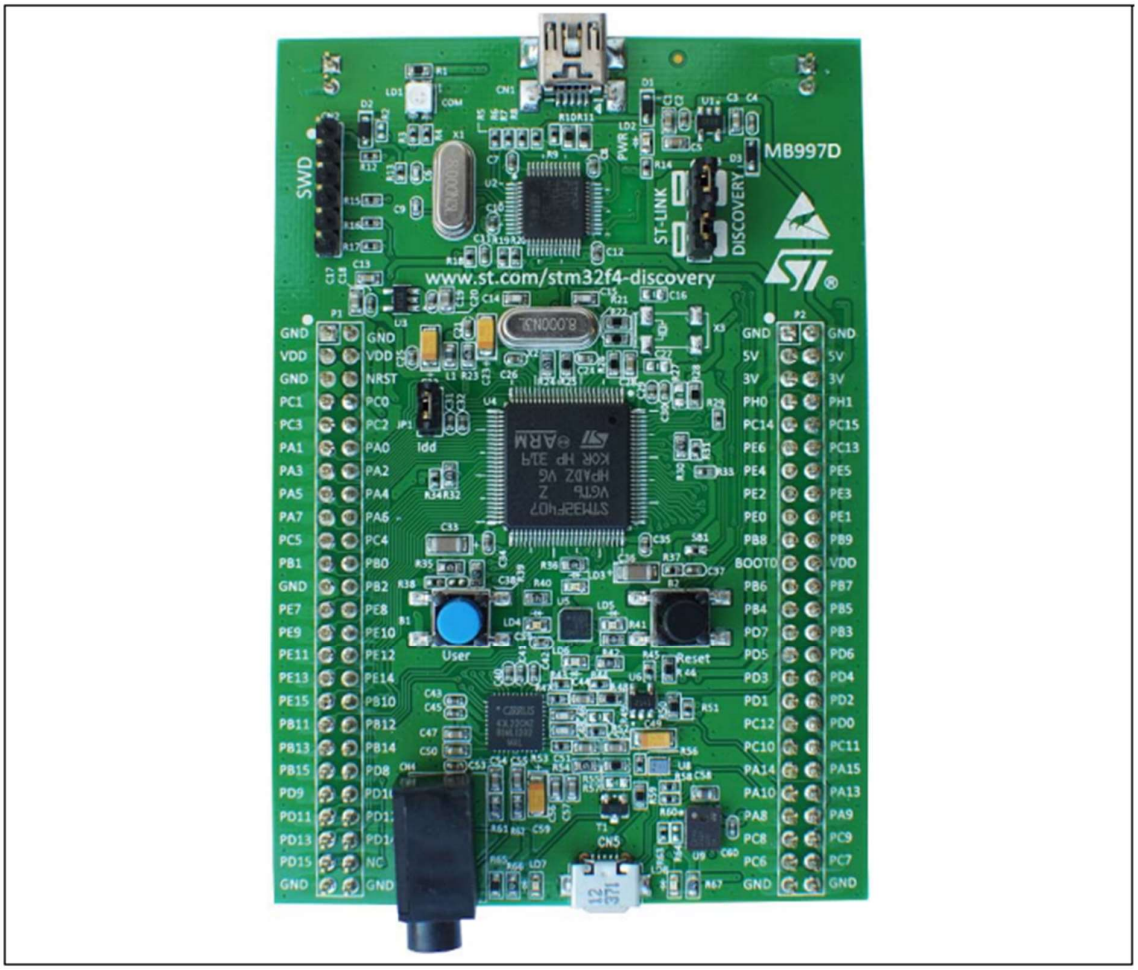
3.9 Flowchart

3.4 WORKING:

The STM32F4 Discovery Board has inbuilt USB port where an external USB drive is connected which stores .wav audio format. The DMA Controller gets data from USB drive to DMA buffer. The DMA & Timer to periodically trigger the DMA unit so that it moves a sample data point from the lookup table stored in memory to the DAC output. Each time the timer trigger event occurs, the DMA transfers a data point from the table to the DAC output and increments the memory pointer to move the next data point in the next trigger event. Further data get transfer to DAC using I2S protocol, where it converts digital data to Analog signal. Finally, the speaker receives this Analog signal from AUX port. For controlling part, the STM32F4 board is having I2C peripheral. As soon as we press a user switch on the STM32 board the .wave file transferred to the circuit based on this state of the Audio Player gets changed and then it performs the respective operation, after that we can listen the song.

HARDWRE DESCRIPTION

4.1 Hardware required:



4.1 STM32F407- Discovery 1

- Features:

The STM32F4DISCOVERY offers the following features:

• STM32F407VGT6 microcontroller featuring 32-bit Arm®(a) Cortex®-M4 with FPU core, 1-Mbyte Flash memory, 192-Kbyte RAM in an LQFP100 package

• USB OTG FS

• ST MEMS 3-axis accelerometer

• ST-MEMS audio sensor omni-directional digital microphone

• Audio DAC with integrated class D speaker driver

• User and reset push-buttons

• Eight LEDs: – LD1 (red/green) for USB communication – LD2 (red) for 3.3 V power on – Four user LEDs, LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue) – Two USB OTG LEDs, LD7 (green) VBUS and LD8 (red) over-current

• Board connectors: – USB with Micro-AB – Stereo headphone output jack – 2.54 mm pitch extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing

• Flexible power-supply options: ST-LINK, USB VBUS, or external sources

• External application power supply: 3 V and 5 V

• Comprehensive free software including a variety of examples, part of STM32CubeF4 MCU Package, or STSW-STM32068 for using legacy standard libraries

• On-board ST-LINK/V2-A debugger/programmer with USB re-enumeration capability: mass storage, Virtual COM port, and debug port

• Support of a wide choice of Integrated Development Environments (IDEs) including IAR Embedded Workbench®, MDK-ARM, and STM32CubeIDE

Why chose STM32F407 DISC1?

The STM32F4DISCOVERY Discovery kit allows users to easily develop applications with the STM32F407VG high-performance microcontroller with the Arm® Cortex®-M4 32-bitcore. It includes everything required either for beginners or experienced users to get started quickly.

As It has all the necessary things that are needed with a USB OTG and a 3.5 mm Audio jack builtin to it.

Moreover we also have a good hands on experience with this board.

Specifications of STM32 discovery board:-

Specs:

GPIO
82

Operating Temperature	
Max	Min
-40	105

Operating Frequency	
Max operating frequency	168 Mhz

Voltage	
Standard Operating Voltage	1.8V(min) - 3.6V(max)

Current consumption			
Run Mode		Sleep Mode	
All peripherals enabled	All peripherals disable	All peripherals enabled	All peripherals disabled
93mA	46	59	12

Memory:-

RAM	Flash
192+4KB SRAM	1 MB

Commuication protocols:

SPI	3
I2C	3
CAN	2
UART	2
USART	4
I2S	2
USB host FS(12MB/s)	1(USB 2.0)
USB host HS(480MB/s)	1(USB 2.0)

4.2 CS43L22 DAC:

The CS43L22 is a highly integrated, low power stereo DAC which has 2 output channels for stereo and mono interface and Class D speaker amplifiers. This chip is internally available in STM32 discovery board. The CS43L22 offers many features suitable for low power, portable system applications. It has about 30 registers in which are used to controls the chip for audio transmission.

FEATURES:

′ 98 dB Dynamic Range (A-wtd) ′ 88 dB THD+N ′ Headphone Amplifier - GND Centered

• No DC-Blocking Capacitors Required • Integrated Negative Voltage Regulator

• 2 x 23 mW into Stereo 16 Ω @ 1.8 V

• 2 x 44 mW into Stereo 16 Ω @ 2.5V ′ Stereo Analog Input Passthrough Architecture

• Analog Input Mixing

• Analog Passthrough with Volume Control ′ Digital Signal Processing Engine

• Bass & Treble Tone Control, De-Emphasis

• PCM Input w/Independent Vol Control • Master Digital Volume Control and Limiter

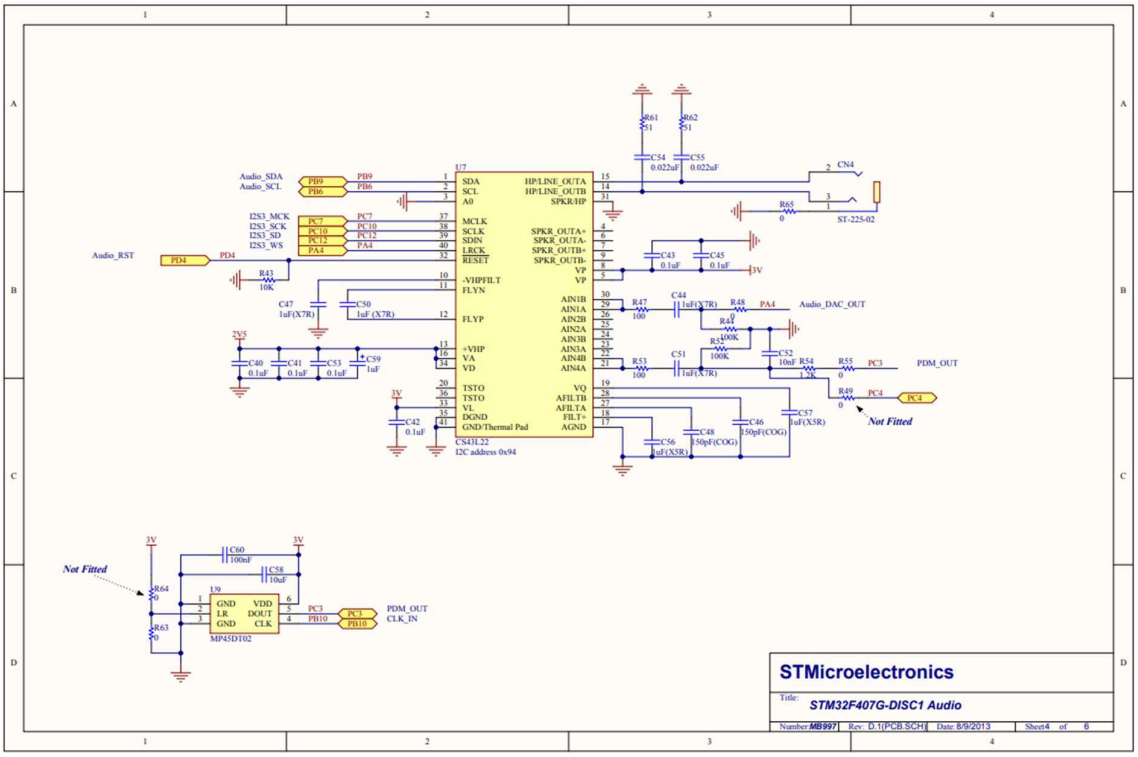
• Soft-Ramp & Zero-Cross Transitions ′ Programmable Peak-Detect and Limiter ′ Beep Generator w/Full Tone Control

• Tone Selections Across Two Octaves

• Separate Volume Control

• Programmable On and Off Time Intervals

• Continuous, Periodic, One-Shot Beep Selections



4.2 STM32F407-DISC1 Audio schematic

SOFTWARE DESCRIPTION

5.1STM32CubeIDE:

an all-in-one multi-OS development tool, which is part of the STM32Cube software ecosystem.

STM32CubeIDE is an advanced C/C++ development platform with peripheral configuration, code generation, code compilation, and debug features for STM32 microcontrollers and microprocessors. It is based on the Eclipse®/CDT™ framework and GCC toolchain for the development, and GDB for the debugging. It allows the integration of the hundreds of existing plugins that complete the features of the Eclipse® IDE.

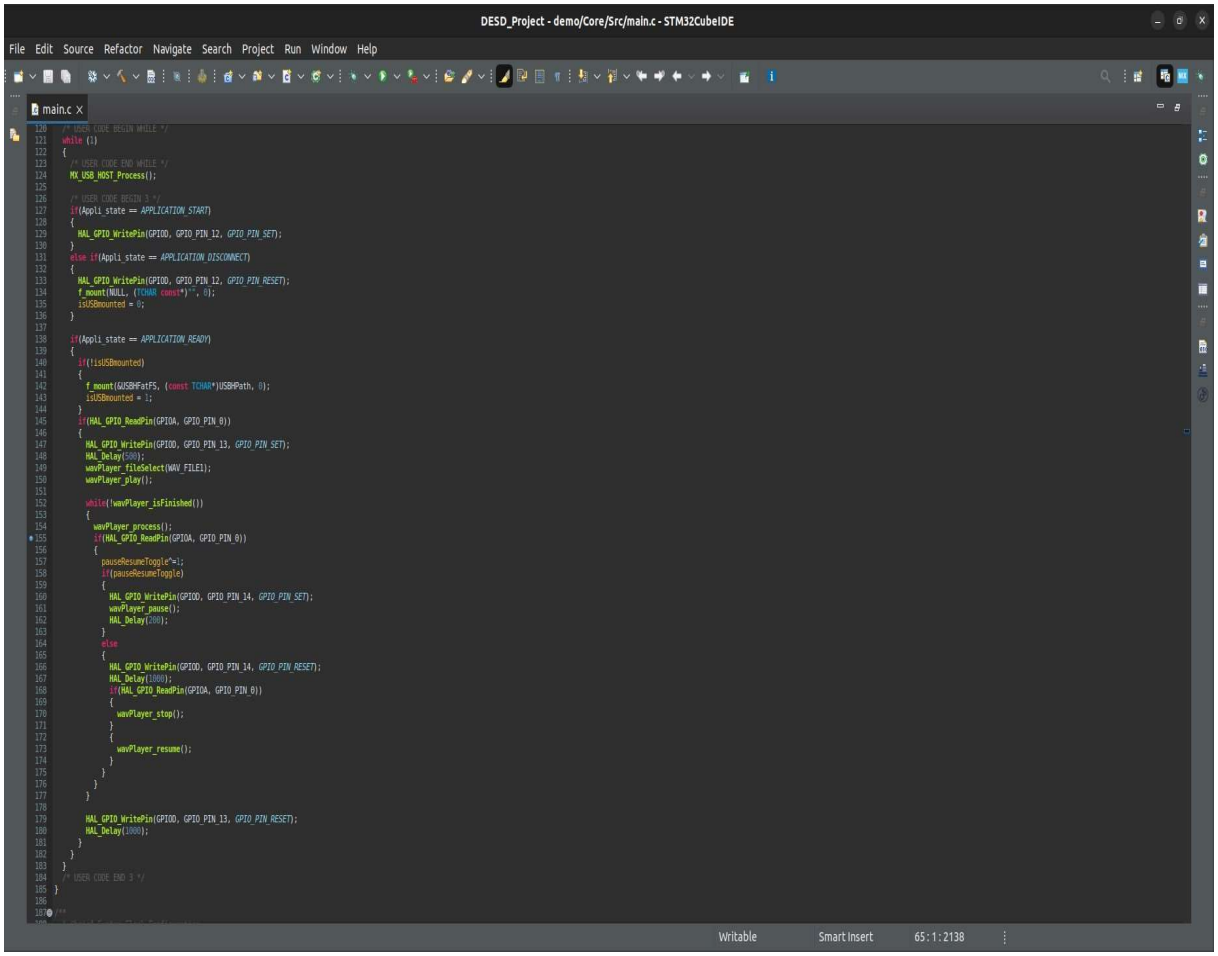
STM32CubeIDE integrates STM32 configuration and project creation functionalities from STM32CubeMX to offer all-in-one tool experience and save installation and development time. After the selection of an empty STM32 MCU or MPU, or preconfigured microcontroller or microprocessor from the selection of a board or the selection of an example, the project is created and initialization code generated. At any time during the development, the user can return to the initialization and configuration of the peripherals or middleware and regenerate the initialization code with no impact on the user code.

STM32CubeIDE includes build and stack analyzers that provide the user with useful information about project status and memory requirements.

STM32CubeIDE also includes standard and advanced debugging features including views of CPU core registers, memories, and peripheral registers, as well as live variable watch, Serial Wire Viewer interface, or fault analyzer

SOURCE CODE EXPLAINATION

6.1 Source code



6.1 source code

In this code we have implemented the functionality of our audio player

Lets see each function individually

1.FILE select function:

Here we write a function which will take wavHeader from wav file and open and start reading the file and configure its wav sampling rate.

```
bool wavPlayer_fileSelect(const char* filePath)
{
  WAV_HeaderTypeDef wavHeader;
  UINT readBytes = 0;
  //Open WAV file
  if(f_open(&wavFile, filePath, FA_READ) != FR_OK)
  {
    return false;
  }
  //Read WAV file Header
  f_read(&wavFile, &wavHeader, sizeof(wavHeader), &readBytes);
  //Get audio data size
  fileLength = wavHeader.FileSize;
  //Play the WAV file with frequency specified in header
  samplingFreq = wavHeader.SampleRate;
  return true;
}
```

6.2 file select function

2.Play function:

In this function we initialize audio I2S with sampling rate and make the fpos of file at the start of the file also we check for remaining size of the file in this function at last we have called audio I2S play function which will internally call CS43 inti and start functions to play the audio file.

```
 */
void wavPlayer_play(void)
{
  isFinished = false;
  //Initialise I2S Audio Sampling settings
  audioI2S_init(samplingFreq);
  //Read Audio data from USB Disk
  f_lseek(&wavFile, 0);
  f_read (&wavFile, &audioBuffer[0], AUDIO_BUFFER_SIZE, &playerReadBytes);
  audioRemainSize = fileLength - playerReadBytes;
  //Start playing the WAV
  audioI2S_play((uint16_t *)&audioBuffer[0], AUDIO_BUFFER_SIZE);
}
```

6.3 Play function

3.Audio process function:

It is switch case used to take the audio sample in the buffer for our application we selected fullbuffer which is of 4096 kb.

```
void wavPlayer_process(void)
{
  switch(playerControlSM)
  {
  case PLAYER_CONTROL_Idle:
    break;

  case PLAYER_CONTROL_HalfBuffer:
    playerReadBytes = 0;
    playerControlSM = PLAYER_CONTROL_Idle;
    f_read (&wavFile, &audioBuffer[0], AUDIO_BUFFER_SIZE/2, &playerReadBytes);
    if(audioRemainSize > (AUDIO_BUFFER_SIZE / 2))
    {
      audioRemainSize -= playerReadBytes;
    }
    else
    {
      audioRemainSize = 0;
      playerControlSM = PLAYER_CONTROL_EndOfFile;
    }
    break;

  case PLAYER_CONTROL_FullBuffer:
    playerReadBytes = 0;
    playerControlSM = PLAYER_CONTROL_Idle;
    f_read (&wavFile, &audioBuffer[AUDIO_BUFFER_SIZE/2], AUDIO_BUFFER_SIZE/2, &playerReadBytes);
    if(audioRemainSize > (AUDIO_BUFFER_SIZE / 2))
    {
      audioRemainSize -= playerReadBytes;
    }
    else
    {
      audioRemainSize = 0;
      playerControlSM = PLAYER_CONTROL_EndOfFile;
    }
    break;

  case PLAYER_CONTROL_EndOfFile:
    f_close(&wavFile);
    wavPlayer_reset();
    isFinished = true;
    playerControlSM = PLAYER_CONTROL_Idle;
    break;
  }
}
```

6.4 Process function

4. Stop function:

This function will call the audio I2S stop function which will stop the audio player.

```
void wavPlayer_stop(void)
{
  audioI2S_stop();
  isFinished = true;
}
```

6.5 stop function

5. Pause and resume function:

this function internally makes call to audio I2S pause and resume function which internally stops the DAC conversion and save the remaining file size at the time of pause and resume it from that file pos.

```
void wavPlayer_pause(void)
{
  audioI2S_pause();
}
void wavPlayer_resume(void)
{
  audioI2S_resume();
}
```

6.6 pause and resume function

6. is Finished function:

This function is used to check if the file is finished or not.

```
bool wavPlayer_isFinished(void)
{
  return isFinished;
}
```

6.7 is finished function

RESULT

7.1. Hardware Circuit

As you can see in the below figure, we have connected a SANDISK Pen-Drive, with the help of a OTG to the usb port on STM32 board. Since it is an audio player there is no change in the circuit that can be shown using image to listen to the audio output we have used a JBL headphone connected to the AUX port of STM32 Board.

FUTURE SCOPE

• Create a LIFI based audio transmission between audio player circuit and speaker using LASER lights.

• Upgrade the communication between mobile and hardware by replacing Bluetooth with WIFI and controlling it with the help of cloud.
• Add the volume feature too.

• Upgrade the system in such a way that it can accept multiple file formats.

• We can interface this by gesture control sensor and play audio by using using gestures, moving them right, left, up and down for next, previous, volume up and volume down

CONCLUSION

Through the development of Audio Payer on STM32F4 and Android platform, we get a clear understanding of overall process of the system. The core part of the Audio Player is mainly composed of main interface, file browsing and song listing. This project performs the basic function of Audio Player: play, pause, next, previous, shuffle, etc. which is controlled through an Android System Itself. As we are using Bluetooth device with the application the processing has become so handy that we can even perform the commands through voice as well.

REFERANCES

1. stm32f407_user_manual
2. stm32f407vg_disc1_user_manual.
3. stm32f407vg-data-sheet
4. stm32f407vg_disc1_schematic
5. arm_cm4_tech_ref_manual
6. arm_cm4_tech_ref_manual
7. stm32f407vg_disc1_schematic
8. stm32f407vg-data-sheet
9. an3997-audio-playback-and-recording-using-the-stm32f4discovery-stmicroelectronics-1
10. ARM Developers Referance Module