```
#include <SD.h>
#include <SPIFFS.h>
#include <esp_log.h>
#include <freertos/task.h>
#include <DACOutput.h>
#define MINIMP3_IMPLEMENTATION
#define MINIMP3_ONLY_MP3
#define MINIMP3_NO_STDIO
#include "minimp3.h"

#define MP3_MAX_VOLUME 4096
#define MP3_SPIFFS_FILE_NAME "/intern.mp3"
#define MP3_MAX_INTERNAL_FILE_SIZE 2097152 // 2 * 1024 * 1024

// Played Audio has to have a Constant Bitrate of 320kbps, Mono,
// with a sample rate of 48000Hz, and is played with a speed multiplier of 0.5
// (So to have it play, with the correct speed, you have to apply a speed and pitch multiplier of 2.0)

class MP3
{
    private:
    static const int BUFFER_SIZE = 8192;

    static bool LoadToIPFS(std::string sourceFileName)
    {
        Serial.println(("Loading " + sourceFileName + " to SPIFFS...").c_str());
        File sourceFile = SD.open(sourceFileName.c_str());
        Serial.print("Filesize: ");
        Serial.println(sourceFile.size());

        if(sourceFile.size() > MP3_MAX_INTERNAL_FILE_SIZE)
        {
            Serial.println("File too big!");
            return false;
        }

        SPIFFS.remove(MP3_SPIFFS_FILE_NAME);
        File destFile = SPIFFS.open(MP3_SPIFFS_FILE_NAME, FILE_WRITE);

        static uint8_t buf[1024];
        while(sourceFile.read(buf, 1024))
        {
            destFile.write(buf, 1024);
        }

        sourceFile.close();
        destFile.close();

        Serial.println("Loading finished!");

        return true;
    }

    public:
    static std::string mp3File;
    static int VOLUME;
    static TaskHandle_t mp3TaskHandle;

    static void play_task(void *param)
    {
        Serial.println("MP3 PLAY TASK");
        Output *output = new DACOutput();
        // setup for the mp3 decoded
        short *pcm = (short *)malloc(sizeof(short) * MINIMP3_MAX_SAMPLES_PER_FRAME);
        uint8_t *input_buf = (uint8_t *)malloc(BUFFER_SIZE);
        if (!pcm)
        {
            ESP_LOGE("MP3", "Failed to allocate pcm memory");
        }
        if (!input_buf)
        {
            ESP_LOGE("MP3", "Failed to allocate input_buf memory");
        }

        Serial.println("MP3 STARTING");

        while (true)
        {
            // mp3 decoder state
            mp3dec_t mp3d = {};
            mp3dec_init(&mp3d);
            mp3dec_frame_info_t info = {};
            // keep track of how much data we have buffered, need to read and decoded
            int to_read = BUFFER_SIZE;
            int buffered = 0;
            int decoded = 0;
            bool is_output_started = false;

            FILE *fp;

            fp = fopen(("/sd" + mp3File).c_str(), "r");

            if (!fp)
            {
                ESP_LOGE("MP3", "Failed to open file");
                fclose(fp);
                continue;
            }
            while (1)
            {
                auto adc_value = float(VOLUME) / 4096.0f;
                output->set_volume(adc_value * adc_value);
                // read in the data that is needed to top up the buffer
                size_t n = fread(input_buf + buffered, 1, to_read, fp);
                // feed the watchdog
                vTaskDelay(pdMS_TO_TICKS(1));
                // ESP_LOGI("main", "Read %d bytes\n", n);
                buffered += n;
                if (buffered == 0)
                {
                    // we've reached the end of the file and processed all the buffered data
                    output->stop();
                    is_output_started = false;
                    break;
                }
                // decode the next frame
                int samples = mp3dec_decode_frame(&mp3d, input_buf, buffered, pcm, &info);
                // we've processed this may bytes from teh buffered data
                buffered -= info.frame_bytes;
                // shift the remaining data to the front of the buffer
                memmove(input_buf, input_buf + info.frame_bytes, buffered);
                // we need to top up the buffer from the file
                to_read = info.frame_bytes;
                if (samples > 0)
                {
                    // if we haven't started the output yet we can do it now as we now know the sample rate and number of channels
                    if (!is_output_started)
                    {
                        output->start(info.hz);
                        is_output_started = true;
                    }
                    // if we've decoded a frame of mono samples convert it to stereo by duplicating the left channel
                    // we can do this in place as our samples buffer has enough space
                    if (info.channels == 1)
                    {
                        for (int i = samples - 1; i >= 0; i--)
                        {
                            pcm[i * 2] = pcm[i];
                            pcm[i * 2 - 1] = pcm[i];
                        }
                    }
                    // write the decoded samples to the I2S output
                    output->write(pcm, samples);
                    // keep track of how many samples we've decoded
                    decoded += samples;
                }
```

```
             //ESP_LOGI("main", "decoded %d samples\n", decoded);
            }
            ESP_LOGI("mp3", "Finished\n");
            fclose(fp);
            break;
        }
        vTaskDelete(NULL);
    }
    static void Play()
    {
        mp3TaskHandle = NULL;
        xTaskCreatePinnedToCore(play_task, "mp3Task", 32768, NULL, 1, &mp3TaskHandle, 0);
    }
    static void Stop()
    {
        vTaskDelete(mp3TaskHandle);
    }
};
```