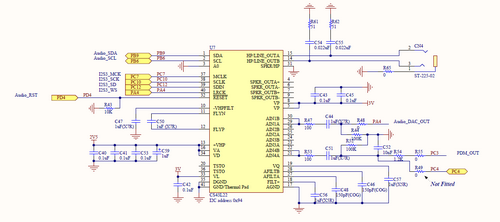
**I2S AUDIO**

Today we are starting to learn a new interface for us - it's an I2S bus, something that is similar to I2C, but sharpened directly to the transfer of a digital audio stream. The description and diagrams of the protocol of this bus can be found on page 894 STM32F4 Reference manual. Well, briefly, this bus is also synchronous, and synchronization is provided not only for each transmitted bit, as for I2C, but also for the channel. That is, a separate synchronization wire is used so that when all the bits are transmitted (they can be 8, 16, 24) of one channel, it is in state 1, and when all bits of the other channel are transmitted, it is 0. This condition ensures the inadvertent exchange of channels among themselves due to the distortion of the flow signal.

We will create our project from one of the previous projects USB\_HOST\_MSC\_FATFS, since we will work with USB Flash Drive, because we need to take this digital audio from somewhere to study the conversion of digital audio. It was decided to take it from WAV files located on this medium. The project we will name by the name of the bus I2S\_AUDIO. Since we will still connect the USB Flash Drive to the Discovery card, we need to copy it for the WAV file with the sound: Track1.wav and Track2.wav. The sampling frequency of these files can be any, but preferably not more than 48 kHz.

Let's see the connection of the audio chip in the Discovery board (click on the image to enlarge the image)

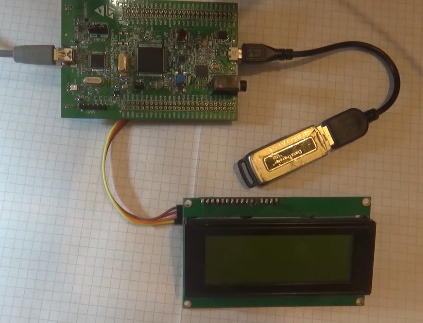
[](http://narodstream.ru/wp-content/uploads/2016/11/image00_1189.png)

This chip is called CS43L22. Its main characteristic is the supported sampling rate. This is from 4 kHz to 96 kHz. Let's open its datasheet. There are four types of I2S protocol. We use the very first standard. The only thing from the datasheet is unclear why the MCLK contact is needed. The following description of this contact is given in datasheet:

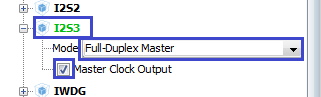
MCLK - Master Clock (Input) - Clock source for the delta-sigma modulators.

By browsing through several forums, I found that this is the third type of synchronization - sampling.

That's how we have everything connected to the board

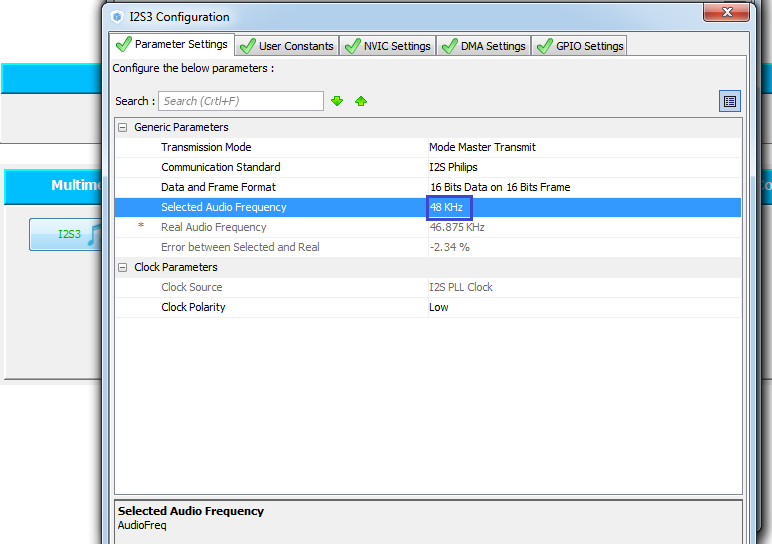


Open our project in MS Cube and make the necessary addition of certain settings. Turn first the bus itself I2S



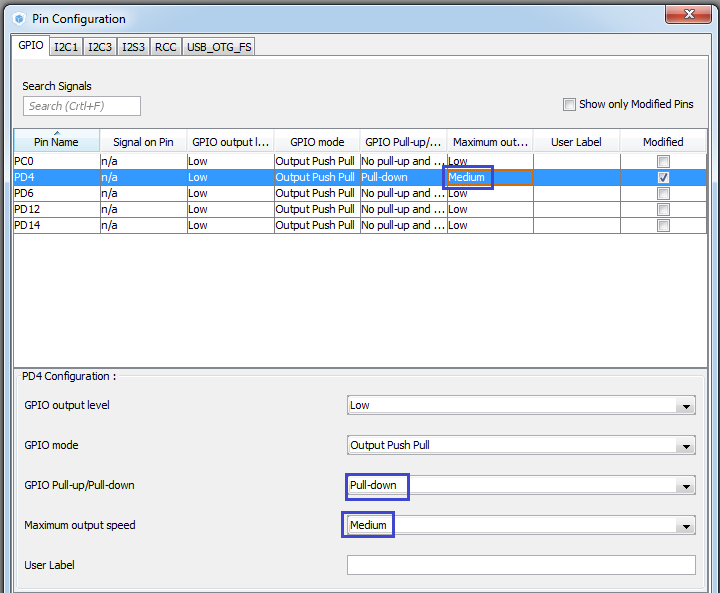
No legs are redefined. Let's leave it as it is. It is on these legs that the Audio DAC chip is connected.

We will also make some adjustments to the I2S settings

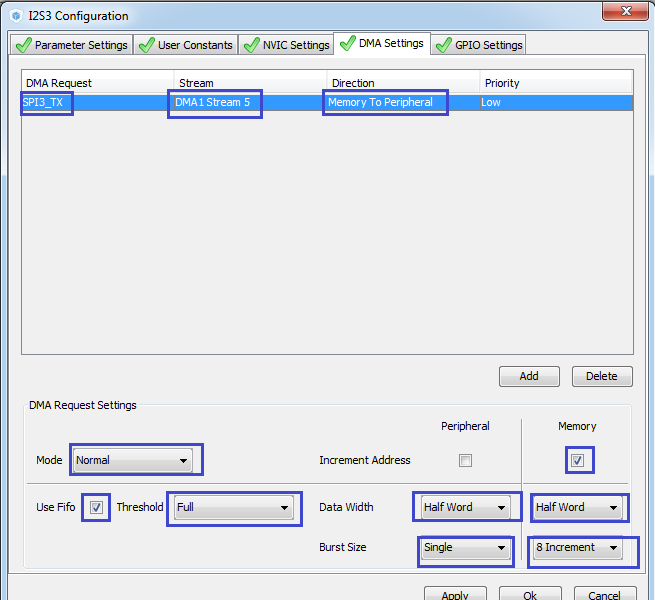


For the work of the RESET foot of the microcircuit, we will still need to turn on the output and adjust this port foot (PD4)





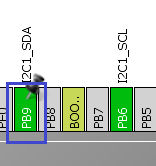
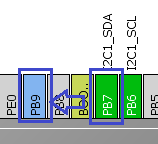
Also add and configure DMA on the I2S bus



And since the chip management occurs in contrast to the main audio stream already on the I2C bus, it is necessary to include also this bus



Only I2C3 we give under the symbol display, and I2C1 we will need for the audio-chip. Moreover, the PB7 foot will need to be redefined on PB9. I think everyone already knows how to do this. First, reset it, and then redefine it



Now the display adapter will be connected to the other ports of the ports:

SCL-PA8

SDA - PC9.

Generate the project for the Keil environment, configure the programmer for auto-cutting, add the lcd.c file to the project tree and compile the project.

Create and add new files to the project that are designed to work with audio, audioplay.c and audioplay.h as follows:

audioplay.c:

#include "audioplay.h"

#include "fatfs.h"

#include "usb\_host.h"

#include "lcd.h"

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

audioplay.h:

#ifndef AUDIOPLAY\_H\_

#define AUDIOPLAY\_H\_

#ifdef \_\_cplusplus

 extern "C" {

#endif

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

#include "stm32f4xx\_hal.h"

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

#define I2S3 SPI3

#ifdef \_\_cplusplus

}

#endif

#endif / \* AUDIOPLAY\_H\_ \* /

Also add the connection of our library to the main module main.h

#include "lcd.h"

**#include "audioplay.h"**

Due to the fact that the LCD is now on the 3rd I2C bus, we'll make some changes to the lcd.c file:

extern I2C\_HandleTypeDef hi2c **3**;

...

HAL\_I2C\_Master\_Transmit (& hi2c **3**, (uint16\_t) 0x4E, buf, 1,1000);

Remove this from main ()

~~char str [100];~~

  / \* USER CODE END 1 \* /

Also, in main.c, the FileReadWrite function is fixed in MenuProcess and cleaned up completely of its body:

/ \* USER CODE BEGIN 0 \* /

void **MenuProcess**(void)

{

}

Accordingly, in an infinite cycle, there will also be certain changes

  while (1)

  {

  / \* USER CODE END WHILE \* /

    MX\_USB\_HOST\_Process ();

  / \* USER CODE BEGIN 3 \* /

**MenuProcess**();

  }

  / \* USER CODE END 3 \* /

Add some variables and macros to the header file audioplay.h.

#define I2S3 SPI3

**typedef enum**

**{**

**BUFFER\_OFFSET\_NONE = 0,**

**BUFFER\_OFFSET\_HALF,**

**BUFFER\_OFFSET\_FULL,**

**} BUFFER\_StateTypeDef;**

**typedef struct**

**{**

**uint32\_t ChunkID; / \* 0 \* /**

**uint32\_t FileSize; /\* 4 \*/**

**uint32\_t FileFormat; /\* 8 \*/**

**uint32\_t SubChunk1ID; /\* 12 \*/**

**uint32\_t SubChunk1Size; /\* 16 \*/**

**uint16\_t AudioFormat; /\* 20 \*/**

**uint16\_t NbrChannels; / \* 22 \* ​​/**

**uint32\_t SampleRate; / \* 24 \* /**

**uint32\_t ByteRate; / \* 28 \* /**

**uint16\_t BlockAlign; / \* 32 \* /**

**uint16\_t BitPerSample; / \* 34 \* /**

**uint32\_t SubChunk2ID; / \* 36 \* /**

**uint32\_t SubChunk2Size; / \* 40 \* /**

**} WAVE\_FormatTypeDef;**

**/ \* Audio State Structure \* /**

**typedef enum {**

**AUDIO\_IDLE = 0,**

**AUDIO\_WAIT,**

**AUDIO\_EXPLORE,**

**AUDIO\_PLAYBACK,**

**AUDIO\_IN,**

**} AUDIO\_State;**

**/ \* Audio Demo State Machine Structure \* /**

**typedef struct \_StateMachine {**

**\_\_IO AUDIO\_State state;**

**\_\_IO uint8\_t select;**

**} AUDIO\_StateMachine;**

**// ----------------**

And that is not all. Also add some more variables, structures and macros to the file audioplay.c

#include "lcd.h"

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

#define I2S\_STANDARD I2S\_STANDARD\_PHILIPS

/ \* Audio status definition \* /

#define AUDIO\_OK 0

#define AUDIO\_ERROR 1

#define AUDIO\_TIMEOUT 2

/ \* Position in the audio play buffer \* /

\_\_IO BUFFER\_StateTypeDef buffer\_offset = BUFFER\_OFFSET\_NONE;

/ \* Codec output DEVICE \* /

#define OUTPUT\_DEVICE\_SPEAKER 1

#define OUTPUT\_DEVICE\_HEADPHONE 2

#define OUTPUT\_DEVICE\_BOTH 3

#define OUTPUT\_DEVICE\_AUTO 4

/ \* MUTE commands \* /

#define AUDIO\_MUTE\_ON 1

#define AUDIO\_MUTE\_OFF 0

/ \* Defines for the Audio playing process \* /

#define PAUSE\_STATUS ((uint32\_t) 0x00) / \* Audio Player in Pause Status \* /

#define RESUME\_STATUS ((uint32\_t) 0x01) / \* Audio Player in Resume Status \* /

#define IDLE\_STATUS ((uint32\_t) 0x02) / \* Audio Player in Idle Status \* /

#define AUDIO\_RESET\_GPIO\_CLK\_ENABLE () \_\_GPIOD\_CLK\_ENABLE ()

#define AUDIO\_RESET\_PIN GPIO\_PIN\_4

#define AUDIO\_RESET\_GPIO GPIOD

#define VOLUME\_CONVERT (Volume) (((Volume)> 100)? 100: ((uint8\_t) (((Volume) \* 255) / 100)))

#define CODEC\_STANDARD 0x04

/ \* Variables used in normal mode to manage audio file during DMA transfer \* /

uint32\_t AudioTotalSize = 0xFFFF; / \* This variable holds the total size of the audio file \* /

int32\_t AudioRemSize = 0xFFFF; / \* This variable holds the remaining data in the audio file \* /

uint16\_t \* CurrentPos; / \* This variable holds the current position of the audio pointer \* /

static uint8\_t Is\_cs43l22\_Stop = 1;

\_\_IO uint32\_t PauseResumeStatus = IDLE\_STATUS;

#define CS43L22\_REG\_MISC\_CTL 0x0E

#define AUDIO\_I2C\_ADDRESS 0x94

#define CS43L22\_CHIPID\_ADDR 0x01

#define CS43L22\_ID 0xE0

#define CS43L22\_ID\_MASK 0xF8

#define AUDIO\_BUFFER\_SIZE 0X8000

#define DMA\_MAX\_SZE 0xFFFF

#define DMA\_MAX (\_X\_) (((\_X\_) <= DMA\_MAX\_SZE)? (\_X \_): DMA\_MAX\_SZE)

#define AUDIODATA\_SIZE 2 / \* 16-bits audio data size \* /

extern FIL WavFile;

const uint32\_t I2SFreq [8] = {8000, 11025, 16000, 22050, 32000, 44100, 48000, 96000};

const uint32\_t I2SPLLN [8] = {256, 429, 213, 429, 426, 271, 258, 344};

const uint32\_t I2SPLLR [8] = {5, 4, 4, 4, 4, 6, 3, 1};

volatile uint8\_t OutputDev = 0;

static uint32\_t WaveDataLength = 0;

extern I2C\_HandleTypeDef hi2c1;

extern I2S\_HandleTypeDef hi2s3;

uint8\_t Audio\_Buffer [AUDIO\_BUFFER\_SIZE];

extern WAVE\_FormatTypeDef \* waveformat;

AUDIO\_StateMachine Audio;

uint32\_t samplerate;

extern ApplicationTypeDef Appli\_state;

char str2 [20];

char str3 [20];

uint32\_t offsetpos;

uint32\_t cnt;

uint32\_t dur; // variable for the remaining time of playing the file

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

That's how much we all need. Of course, all this is not in vain, it is done for our own convenience. Well, that does not seem to be wrong extern without the main addition of a variable, add it to main.c, as well as a few more variables, replacing the existing ones, since we got here

/ \* USER CODE BEGIN PV \* /

/ \* Private variables ------------------- \* /

FATFS USBDISKFatFs;

FIL WavFile;

extern char USBH\_Path [4]; / \* USBH logical drive path \* /

extern ApplicationTypeDef Appli\_state;

extern AUDIO\_StateMachine Audio;

WAVE\_FormatTypeDef \* waveformat = NULL;

uint32\_t WaveDataLength = 0;

char str [100];

char FileName [100] = {0};

uint8\_t info [44];

/ \* USER CODE END PV \* /

Well, finally, we can proceed with the basic code. First, write the error function in the file audioplay.c.We use this function constantly

**void Error (void)**

**{**

**HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_14, GPIO\_PIN\_SET);**

**}**

**// ------------------**

Let's write the initialization function

**void AudioPlay\_Init (uint32\_t AudioFreq)**

**{**

**samplerate = AudioFreq;**

**\_\_IO uint8\_t volume = 70;**

**}**

**// ----------------- // -----------------**

Also we will create a prototype for this function, since it will be useful to us outside

} AUDIO\_StateMachine;

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

**void AudioPlay\_Init (uint32\_t AudioFreq);**

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

In the [**next part of the**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-2/) lesson we will write the function of extracting information from the sound file about the parameters of this file and outputting these parameters to the display.

**Lesson 46**

**Part 2**

# ****I2S AUDIO****

In the [**previous part of the**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-1/) lesson we created and configured the project in Cube MX, created the application framework, added all the macros and global variables that we will need in the process of writing the code.   
Also we started to write some functions of the audio library.

Create another initialization function and add some variables to it as well

**uint8\_t AudioOut\_Init (uint16\_t OutputDevice, uint8\_t Volume, uint32\_t AudioFreq)**

**{**

**uint8\_t ret = AUDIO\_ERROR;**

**uint32\_t deviceid = 0x00;**

**RCC\_PeriphCLKInitTypeDef rccclkinit;**

**uint8\_t index = 0, freqindex = 0xFF;**

**}**

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

void AudioPlay\_Init (uint32\_t AudioFreq)

Call this function in the AudioPlay\_Init function

  \_\_IO uint8\_t volume = 70;

**if (AudioOut\_Init (OUTPUT\_DEVICE\_AUTO, volume, samplerate)! = 0)**

**{**

**Error ();**

**}**

}

In main.c in the MenuProcess function we create some code framework

void MenuProcess (void)

{

**if (Appli\_state == APPLICATION\_READY)**

**{**

**switch (Audio.state)**

**{**

**case AUDIO\_IDLE:**

**break;**

**case AUDIO\_WAIT:**

**break;**

**case AUDIO\_EXPLORE:**

**break;**

**case AUDIO\_PLAYBACK:**

**break;**

**case AUDIO\_IN:**

**break;**

**default:**

**break;**

**}**

**}**

}

Let's begin by filling the given frame with the code:

            case AUDIO\_IDLE:

**Audio.state = AUDIO\_WAIT;**

**LCD\_SetPos (0, 0);**

**LCD\_String ("Stm32 Audio Player");**

                                break;

In the usb\_host.c file in the USBH\_UserProcess function, we will correct the code to a more standard form

static void USBH\_UserProcess (USBH\_HandleTypeDef \* phost, uint8\_t id)

{

  / \* USER CODE BEGIN 2 \* /

  switch (id)

  {

  case HOST\_USER\_SELECT\_CONFIGURATION:

  break;

  case HOST\_USER\_DISCONNECTION:

**Appli\_state = APPLICATION\_DISCONNECT;**

  break;

  case HOST\_USER\_CLASS\_ACTIVE:

  Appli\_state = **APPLICATION\_READY**;

  break;

  case HOST\_USER\_CONNECTION:

**Appli\_state = APPLICATION\_START**;

  break;

  default:

  break;

  }

  / \* USER CODE END 2 \* /

}

In main.c we add one more function

/ \* USER CODE BEGIN 0 \* /

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

**void GetFileInfo (void)**

**{**

**if (f\_open (& WavFile, FileName, FA\_OPEN\_EXISTING | FA\_READ) == FR\_OK)**

**{**

**uint32\_t duration;**

**uint32\_t bytesread;**

**if (f\_read (& WavFile, info, 44, (void \*) & bytesread) == FR\_OK)**

**{**

**waveformat = (WAVE\_FormatTypeDef \*) info;**

**sprintf ((char \*) str, "% d", (int) (waveformat-> SampleRate));**

**LCD\_SetPos (0, 2);**

**LCD\_String (str);**

**sprintf ((char \*) str, "% d", (int) (waveformat-> NbrChannels));**

**LCD\_String (str);**

**duration = waveformat-> FileSize / waveformat-> ByteRate;**

**sprintf ((char \*) str, "% d% 02d:% 02d", (int) (waveformat-> FileSize / 1024), (int) (duration / 60), (int) (duration% 60));**

**LCD\_String (str);**

**}**

**f\_close (& WavFile);**

**}**

**}**

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

Also add some code in the main function main ()

        LCD\_Clear ();

**HAL\_Delay (500);**

**sprintf (FileName, "Track1.wav");**

  / \* USER CODE END 2 \* /

Continue to write the code for the MenuProcess function

                        case AUDIO\_WAIT:

**LCD\_SetPos (0, 1);**

**if ((f\_mount (& USBDISKFatFs, (TCHAR const \*) USBH\_Path, 0)! = FR\_OK))**

**{**

**LCD\_String ("Can not Init FatFs!");**

**}**

**else**

**{**

**LCD\_String ("FatFs Initialized!");**

**}**

**f\_close (& WavFile);**

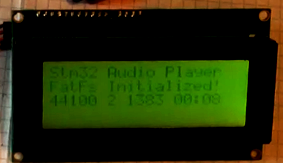
**GetFileInfo ();**

**AudioPlay\_Init (waveformat-> SampleRate);**

**Audio.state = AUDIO\_PLAYBACK;**

                                break;

Let's try to collect the code and flash the controller to see if we have opened the file



We see on the display that the file has opened, can see its sampling frequency, the length in bytes and seconds.

In the [**next part of the**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-3/) lesson we will continue the initialization of the Audio DAC chip and write the function of reading the ID of this chip via the I2C bus.

**Lesson 46**

**Part 3**

# ****I2S AUDIO****

In the [**previous part of the**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-2/) lesson we wrote a function to extract information from a sound file about the parameters of this file and display these parameters on the display.

Let's write some more code in the function AudioOut\_Init file audioplay.c

  uint8\_t index = 0, freqindex = 0xFF;

**for (index = 0; index <8; index ++)**

**{**

**if (I2SFreq [index] == AudioFreq)**

**{**

**freqindex = index;**

**}**

**}**

**/ \* Enable PLLI2S clock \* /**

**HAL\_RCCEx\_GetPeriphCLKConfig (& rccclkinit);**

**/ \* PLLI2S\_VCO Input = HSE\_VALUE / PLL\_M = 1 Mhz \* /**

**if ((freqindex & 0x7) == 0)**

**{**

**/ \* I2S clock config**

**PLLI2S\_VCO = f (VCO clock) = f (PLLI2S clock input) X (PLLI2SN / PLLM)**

**I2SCLK = f (PLLI2S clock output) = f (VCO clock) / PLLI2SR \* /**

**rccclkinit.PeriphClockSelection = RCC\_PERIPHCLK\_I2S;**

**rccclkinit.PLLI2S.PLLI2SN = I2SPLLN [freqindex];**

**rccclkinit.PLLI2S.PLLI2SR = I2SPLLR [freqindex];**

**HAL\_RCCEx\_PeriphCLKConfig (& rccclkinit);**

**}**

**else**

**{**

**/ \* I2S clock config**

**PLLI2S\_VCO = f (VCO clock) = f (PLLI2S clock input) X (PLLI2SN / PLLM)**

**I2SCLK = f (PLLI2S clock output) = f (VCO clock) / PLLI2SR \* /**

**rccclkinit.PeriphClockSelection = RCC\_PERIPHCLK\_I2S;**

**rccclkinit.PLLI2S.PLLI2SN = 258;**

**rccclkinit.PLLI2S.PLLI2SR = 3;**

**HAL\_RCCEx\_PeriphCLKConfig (& rccclkinit);**

**}**

**HAL\_GPIO\_WritePin (AUDIO\_RESET\_GPIO, AUDIO\_RESET\_PIN, GPIO\_PIN\_RESET);**

**/ \* Wait for a delay to insure registers erasing \* /**

**HAL\_Delay (5);**

**/ \* Power on the codec \* /**

**HAL\_GPIO\_WritePin (AUDIO\_RESET\_GPIO, AUDIO\_RESET\_PIN, GPIO\_PIN\_SET);**

**/ \* Wait for a delay to insure registers erasing \* /**

**HAL\_Delay (5);**

}

In this code, there is basically a re-initialization of the bus depending on the sampling frequency.

Add to this file a function for reading the chip ID

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

**uint32\_t cs43l22\_ReadID (uint16\_t DeviceAddr)**

**{**

**HAL\_StatusTypeDef status = HAL\_OK;**

**uint8\_t value = 0;**

**status = HAL\_I2C\_Mem\_Read (& hi2c1, DeviceAddr, (uint16\_t) CS43L22\_CHIPID\_ADDR, I2C\_MEMADD\_SIZE\_8BIT, & value, 1.0 × 1000);**

**if (status == HAL\_OK)**

**{**

**value = (value & CS43L22\_ID\_MASK);**

**return ((uint32\_t) value);**

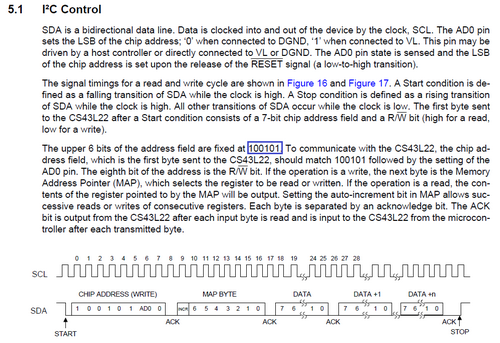
**}**

**return 0;**

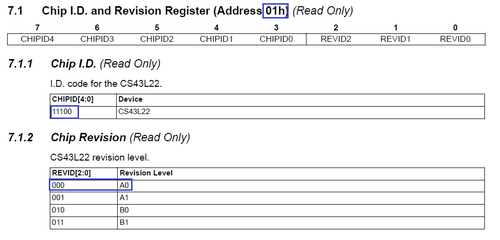
**}**

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

And here we are trying to read the chip ID. First, in order to count it, we must somehow communicate with microsecurity. And for this we need an I2C address. Find it in datashit on page 33 (click on the image to increase the size)

[](http://narodstream.ru/wp-content/uploads/2016/11/image11_0998.png)

In general, at this address, adding two zeros to it, we will work with the I2C bus of the chip. This ID is stored at 0x01. In the following figure, we will see this, and also which ID we should consider (click on the image to enlarge the size)

[](http://narodstream.ru/wp-content/uploads/2016/11/image09_1009.png)

Call the function cs43l22\_ReadID in the AudioOut\_Init function

  HAL\_Delay (5);

**deviceid = cs43l22\_ReadID (AUDIO\_I2C\_ADDRESS);**

**if ((deviceid & CS43L22\_ID\_MASK) == CS43L22\_ID)**

**{**

**ret = AUDIO\_OK;**

**}**

**return ret;**

}

Let's collect the code, we'll tell the controller and if everything is OK on the display, the file is read, ID is read and if the red LED is not lit, we can safely be sure that we are working with this chip and that the files are read and recognized by us.

In the [**next part of the**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-4/) lesson, we will continue to write the initialization of the Audio DAC chip. We will put some settings in certain registers, and also write the function of setting the volume of channels.

**Lesson 46**

**Part 4**

# ****I2S AUDIO****

In the [**previous part of the**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-3/) lesson we continued to write the initialization of the Audio DAC chip and wrote the function of reading the identifier of this chip via the I2C bus.

Continue on to initialization. To do this, add another function to the file audioplay.c

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

**uint32\_t cs43l22\_Init (uint16\_t DeviceAddr, uint16\_t OutputDevice, uint8\_t Volume, uint32\_t AudioFreq)**

**{**

**uint32\_t counter = 0;**

**HAL\_GPIO\_WritePin (AUDIO\_RESET\_GPIO, AUDIO\_RESET\_PIN, GPIO\_PIN\_RESET);**

**/ \* Wait for a delay to insure registers erasing \* /**

**HAL\_Delay (5);**

**/ \* Power on the codec \* /**

**HAL\_GPIO\_WritePin (AUDIO\_RESET\_GPIO, AUDIO\_RESET\_PIN, GPIO\_PIN\_SET);**

**/ \* Wait for a delay to insure registers erasing \* /**

**HAL\_Delay (5);**

**return counter;**

**}**

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

Call it in the function AudioOut\_Init

  if ((deviceid & CS43L22\_ID\_MASK) == CS43L22\_ID)

  {

    ret = AUDIO\_OK;

  }

**if (ret == AUDIO\_OK)**

**{**

**cs43l22\_Init (AUDIO\_I2C\_ADDRESS, OutputDevice, Volume, AudioFreq);**

**}**

**return ret;**

One more feature. The function of writing in the register of the chip of a certain value in the form of a byte

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

**static uint8\_t CODEC\_IO\_Write (uint8\_t Addr, uint8\_t Reg, uint8\_t Value)**

**{**

**HAL\_StatusTypeDef status = HAL\_OK;**

**uint32\_t result = 0;**

**status = HAL\_I2C\_Mem\_Write (& hi2c1, Addr, (uint16\_t) Reg, I2C\_MEMADD\_SIZE\_8BIT, & Value, 1, 0x1000);**

**if (status! = HAL\_OK)**

**{**

**Error ();**

**return 1;**

**}**

**return result;**

**}**

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

Continue to write cs43l22\_Init

  HAL\_Delay (5);

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x02, 0x01);**

**/ \* Save Output device for mute ON / OFF procedure \* /**

**switch (OutputDevice)**

**{**

**case OUTPUT\_DEVICE\_SPEAKER:**

**OutputDev = 0xFA;**

**break;**

**case OUTPUT\_DEVICE\_HEADPHONE:**

**OutputDev = 0xAF;**

**break;**

**case OUTPUT\_DEVICE\_BOTH:**

**OutputDev = 0xAA;**

**break;**

**case OUTPUT\_DEVICE\_AUTO:**

**OutputDev = 0x05;**

**break;**

**default:**

**OutputDev = 0x05;**

**break;**

**}**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x04, OutputDev);**

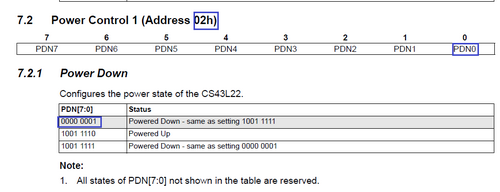
**/ \* Clock configuration: Auto detection \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x05, 0x81);**

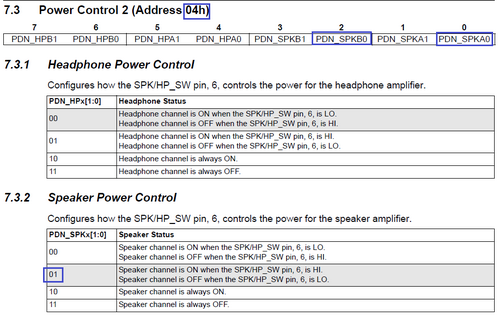
**/ \* Set the Slave Mode and the audio Standard \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x06, CODEC\_STANDARD);**

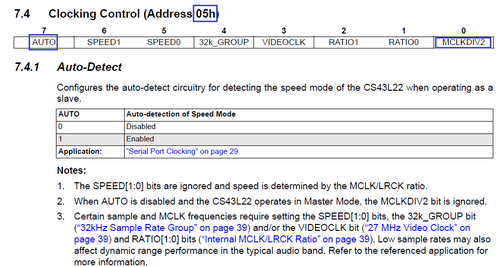
Here we first turned off the DAC (click on the image to enlarge the size)

[](http://narodstream.ru/wp-content/uploads/2016/11/image10_1009.png)

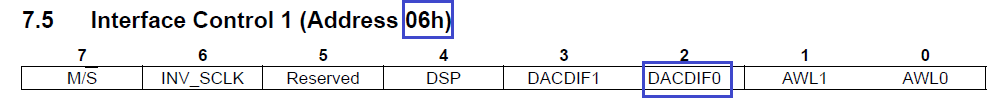
Further the operating mode of the DAC. At us OUTPUT\_DEVICE\_AUTO, that is at us bits will be included the following (are marked by dark blue squares)   (click on a picture to increase the size)

[](http://narodstream.ru/wp-content/uploads/2016/11/image12_0981.png)

Next, configure the timing of the DAC  (click on the image to increase the size)

[](http://narodstream.ru/wp-content/uploads/2016/11/image13_1005.png)

Decipher all the settings of this register will not, everything is written further down in the data. Then configure the interface :



This setting of the register reads as follows:

Serial Port Clocks: Slave (input ONLY)

SCLK Polarity: Not Inverted

DSP Mode: Disabled

DAC Interface Format: I²S, up to 24-bit data

Audio Word Length: 24-bit data

We will write one more function (for setting the volume)

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

**uint32\_t cs43l22\_SetVolume (uint16\_t DeviceAddr, uint8\_t Volume)**

**{**

**uint32\_t counter = 0;**

**uint8\_t convertedvol = VOLUME\_CONVERT (Volume);**

**if (Volume> 0xE6)**

**{**

**/ \* Set the Master volume \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x20, convertedvol - 0xE7);**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x21, convertedvol - 0xE7);**

**}**

**else**

**{**

**/ \* Set the Master volume \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x20, convertedvol + 0x19);**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x21, convertedvol + 0x19);**

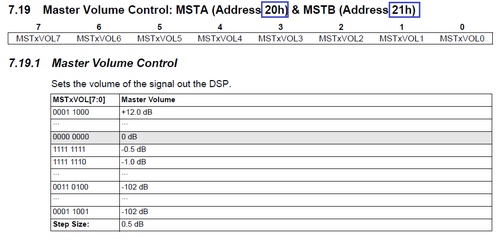
**}**

**return counter;**

**}**

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

When setting the volume, use the following registers (per channel by register) (click on the picture to enlarge the size) :

[](http://narodstream.ru/wp-content/uploads/2016/11/image15_0995.png)

In the [**next part of**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-5/) our lesson we will continue and almost finish writing the initialization of the Audio DAC and write some more important functions for working with this chip.

**Lesson 46**

**Part 5**

# ****I2S AUDIO****

In the [**previous part of the**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-4/) lesson, we continued to write the initialization of the Audio DAC chip. We put some settings in certain registers, and also wrote the function of setting the volume of channels.

We continue the function cs43l22\_Init

  counter + = CODEC\_IO\_Write (DeviceAddr, 0x06, CODEC\_STANDARD);

**/ \* Set the Master volume \* /**

**counter + = cs43l22\_SetVolume (DeviceAddr, Volume);**

**/ \* If the Speaker is enabled, set the Mono mode and volume attenuation level \* /**

**if (OutputDevice! = OUTPUT\_DEVICE\_HEADPHONE)**

**{**

**/ \* Set the Speaker Mono mode \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x0F, 0x06);**

**/ \* Set the Speaker attenuation level \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x24, 0x00);**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x25, 0x00);**

**}**

**/ \* Disable the analog soft ramp \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x0A, 0x00);**

**/ \* Disable the digital soft ramp \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x0E, 0x04);**

**/ \* Disable the limiter attack level \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x27, 0x00);**

**/ \* Adjust Bass and Treble levels \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x1F, 0x0F);**

**/ \* Adjust PCM volume level \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x1A, 0x0A);**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x1B, 0x0A);**

  return counter;

Here, in principle, we are all about the comments.

That's how much we have already written the code, and to the topic of our lesson - we have not touched the I2S bus yet. Well, finally this hour has come! Let's write a new function for initializing this bus.

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

static void I2S3\_Init (uint32\_t AudioFreq)

{

        hi2s3.Instance = I2S3;

         / \* Disable I2S block \* /

  \_\_HAL\_I2S\_DISABLE (& hi2s3);

  hi2s3.Init.AudioFreq = AudioFreq;

  hi2s3.Init.Standard = I2S\_STANDARD;

        HAL\_I2S\_DeInit (& hi2s3);

  if (HAL\_I2S\_Init (& hi2s3)! = HAL\_OK)

        {

                        Error ();

        }

}

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

And call it in the function cs43l22\_Init

  counter + = CODEC\_IO\_Write (DeviceAddr, 0x1B, 0x0A);

**I2S3\_Init (AudioFreq);**

  return counter;

Once again, we'll compile the code, we'll run through the controller and make sure that we have no errors and that the red LED does not light up.

Well, we can say that we have completed the initialization.

Now slowly begin writing the sending of the audio stream.

Add another new function to start playing a sound file

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

**void AudioPlay\_Start (uint32\_t AudioFreq)**

**{**

**offsetpos = 44;**

**cnt = 0;**

**UINT bytesread = 0;**

**AudioTotalSize = waveformat-> FileSize;**

**/ \* Get Data from USB Flash Disk \* /**

**WaveDataLength = waveformat-> FileSize;**

**f\_lseek (& WavFile, 0);**

**f\_read (& WavFile, & Audio\_Buffer [0], AUDIO\_BUFFER\_SIZE, & bytesread);**

**AudioRemSize = WaveDataLength - bytesread;**

**CurrentPos = 0;**

**}**

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

Add a prototype to this function in the header file audioplay.h

void AudioPlay\_Init (uint32\_t AudioFreq);

**void AudioPlay\_Start (uint32\_t AudioFreq);**

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

And we call it in main.c in the function, adding there also some more code

                        case AUDIO\_PLAYBACK:

**WaveDataLength = waveformat-> FileSize;**

**if (f\_open (& WavFile, FileName, FA\_OPEN\_EXISTING | FA\_READ) == FR\_OK)**

**{**

**AudioPlay\_Start (waveformat-> SampleRate);**

**f\_close (& WavFile);**

**}**

**LCD\_Clear ();**

**sprintf (FileName, "Track2.wav");**

**Audio.state = AUDIO\_IDLE;**

                                break;

In the file audioplay.c add another function

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

**uint32\_t cs43l22\_SetMute (uint16\_t DeviceAddr, uint32\_t Cmd)**

**{**

**uint32\_t counter = 0;**

**/ \* Set the Mute mode \* /**

**if (Cmd == AUDIO\_MUTE\_ON)**

**{**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x04, 0xFF);**

**}**

**else / \* AUDIO\_MUTE\_OFF Disable the Mute \* /**

**{**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x04, OutputDev);**

**}**

**return counter;**

**}**

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

Here we again work with the fourth register. If we decide to turn off the sound, then we transfer units to all of its bits, thereby disabling all channels, and if we turn on somehow, then we pass zeros to the corresponding bits.

In the [**next part of**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-6/) our lesson, we'll continue to write the sound playback function, we 'll write a stop playback function.

**Lesson 46**

**Part 6**

# ****I2S AUDIO****

In the [**previous part of the**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-5/) lesson we continued and almost finished writing the initialization of the Audio DAC and wrote some more important functions for working with this chip.

We write the following function

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

**uint32\_t cs43l22\_Play (uint16\_t DeviceAddr)**

**{**

**uint32\_t counter = 0;**

**if (Is\_cs43l22\_Stop == 1)**

**{**

**/ \* Enable the digital soft ramp \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, CS43L22\_REG\_MISC\_CTL, 0x06);**

**/ \* Enable Output device \* /**

**counter + = cs43l22\_SetMute (DeviceAddr, AUDIO\_MUTE\_OFF);**

**/ \* Power on the Codec \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x02, 0x9E);**

**Is\_cs43l22\_Stop = 0;**

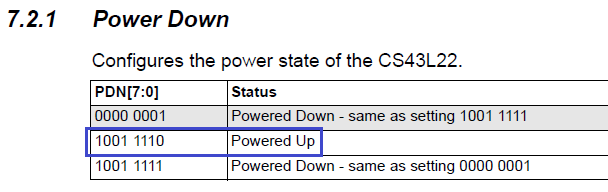
**}**

**return counter;**

**}**

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

Here we first pass the six to register 0x0E, thereby disabling the software ramp. I, unfortunately, have not yet understood what it is and why, but it is necessary to do so. Then we turn off the muting of the sound by the function just considered above. Then, in the 2 register of the DAC, we enter the value 0x9E. With this register, we already worked at the very beginning. There we disabled the DAC so that we did not initiate initialization in its on state. With the value 0x9E we already include the DAC.



Continue to write the code function AudioPlay\_Start

  CurrentPos = 0;

**if (cs43l22\_Play (AUDIO\_I2C\_ADDRESS)! = 0)**

**{**

**Error ();**

**}**

**else**

**{**

**HAL\_I2S\_Transmit\_DMA (& hi2s3, (uint16\_t \*) & Audio\_Buffer [0], DMA\_MAX (AUDIO\_BUFFER\_SIZE / AUDIODATA\_SIZE));**

**}**

**/ \* Check if the device is connected. \* /**

**while ((AudioRemSize! = 0) && (Appli\_state! = APPLICATION\_IDLE))**

**{**

**if (buffer\_offset == BUFFER\_OFFSET\_HALF)**

**{**

**f\_read (& WavFile, & Audio\_Buffer [0], AUDIO\_BUFFER\_SIZE / 2, (void \*) & bytesread);**

**buffer\_offset = BUFFER\_OFFSET\_NONE;**

**AudioRemSize - = bytesread;**

**dur = (waveformat-> FileSize-AudioRemSize) / waveformat-> ByteRate;**

**sprintf ((char \*) str2, "% 02d:% 02d", (int) (dur / 60), (int) (dur% 60));**

**dur = AudioRemSize / waveformat-> ByteRate;**

**sprintf ((char \*) str3, "% 02d:% 02d", (int) (dur / 60), (int) (dur% 60));**

**}**

**if (buffer\_offset == BUFFER\_OFFSET\_FULL)**

**{**

**f\_read (& WavFile, & Audio\_Buffer [AUDIO\_BUFFER\_SIZE / 2], AUDIO\_BUFFER\_SIZE / 2, (void \*) & bytesread);**

**buffer\_offset = BUFFER\_OFFSET\_NONE;**

**AudioRemSize - = bytesread;**

**}**

**if (AudioRemSize <0)**

**{**

**AudioRemSize = 0;**

**dur = 0;**

**sprintf ((char \*) str3, "% 02d:% 02d", (int) (dur / 60), (int) (dur% 60));**

**}**

**LCD\_SetPos (0, 3);**

**LCD\_String (str2);**

**LCD\_String (str3);**

**}**

**HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_12, GPIO\_PIN\_SET);**

Let's write here such function:

**// -----------------**

**uint32\_t cs43l22\_Stop (uint16\_t DeviceAddr)**

**{**

**uint32\_t counter = 0;**

**/ \* Mute the output first \* /**

**counter + = cs43l22\_SetMute (DeviceAddr, AUDIO\_MUTE\_ON);**

**/ \* Power down the DAC and the speaker (PMDAC and PMSPK bits) \* /**

**counter + = CODEC\_IO\_Write (DeviceAddr, 0x02, 0x9F);**

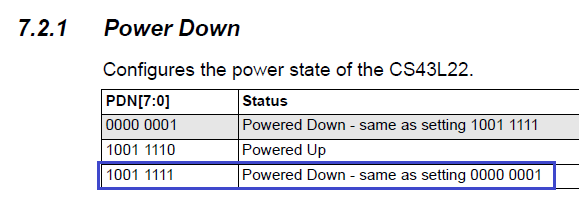
**Is\_cs43l22\_Stop = 1;**

**return counter;**

**}**

**// ------------------**

Here, respectively, we first mute the sound, and then turn off the DAC



Now this:

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

**void AudioPlay\_Stop (void)**

**{**

**HAL\_Delay (1);**

**HAL\_I2S\_DMAStop (& hi2s3);**

**HAL\_Delay (1);**

**cs43l22\_Stop (AUDIO\_I2C\_ADDRESS);**

**}**

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

Call it in AudioPlay\_Start

        HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_12, GPIO\_PIN\_SET);

**AudioPlay\_Stop ();**

}

In the [**next part of**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-7/) this lesson, we will correct the mistakes of the last part of the lesson, write some more functions, and finally finish the lesson.

**Lesson 46**

**Part 7**

# ****I2S AUDIO****

In the [**previous part of the**](http://narodstream.ru/stm-urok-46-i2s-audio-chast-6/) lesson we wrote a function for playing sound, and also wrote a stop playback function.

Let's write two more functions for processing interrupts from I2S to the same file audioplay.c

**void AudioPlay\_HalfTransfer\_CallBack (void)**

**{**

**buffer\_offset = BUFFER\_OFFSET\_HALF;**

**}**

**void AudioPlay\_TransferComplete\_CallBack (void)**

**{**

**buffer\_offset = BUFFER\_OFFSET\_FULL;**

**if (! Is\_cs43l22\_Stop)**

**{**

**HAL\_I2S\_Transmit\_DMA (& hi2s3, (uint16\_t \*) & Audio\_Buffer [0], AUDIO\_BUFFER\_SIZE / 2);**

**}**

**else**

**{**

**cs43l22\_SetVolume (AUDIO\_I2C\_ADDRESS, 0);**

**}**

**}**

Well, since these are not official handlers, it is necessary here somehow to process the processing

For this, first, for them we will add prototypes to the header file audioplay.h

void AudioPlay\_Start (uint32\_t AudioFreq);

**void AudioPlay\_TransferComplete\_CallBack (void);**

**void AudioPlay\_HalfTransfer\_CallBack (void);**

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

Let's write a call to two interrupt functions from I2S in main.c and thus redirect the handler to its functions

/ \* USER CODE BEGIN 4 \* /

**void HAL\_I2S\_TxCpltCallback (I2S\_HandleTypeDef \* hi2s3)**

**{**

**if (hi2s3-> Instance == I2S3)**

**{**

**/ \* Call the user function which will manage directly transfer complete \* /**

**AudioPlay\_TransferComplete\_CallBack ();**

**}**

**}**

**// -------------------------**

**void HAL\_I2S\_TxHalfCpltCallback (I2S\_HandleTypeDef \* hi2s)**

**{**

**if (hi2s-> Instance == I2S3)**

**{**

**/ \* Manage the remaining file size and new address offset: This function should**

**be coded by user (its prototype is already declared in stm32f4\_discovery\_audio.h) \* /**

**AudioPlay\_HalfTransfer\_CallBack ();**

**}**

**}**

/ \* USER CODE END 4 \* /

Check, stitch, should all play (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2016/11/image20-6.png)