Riscy STM32WC55 Project

Chapter 001

Nov 2022

Richard Payne

Contents

[0.0 Reference 2](#_Toc119740021)

[0.1 Weblink 2](#_Toc119740022)

[1.0 Journal 2022 3](#_Toc119740023)

[1.1 16/Oct/22: What is codec? 3](#_Toc119740024)

[1.2 23/Oct/22: STM32 Audio project review 4](#_Toc119740025)

[1.2.1 Objective: Audio Player (WAV) 4](#_Toc119740026)

[1.2.2 Links 4](#_Toc119740027)

[1.2.3 Part 1 Review 4](#_Toc119740028)

[1.2.4 Part 1: Overview of IMA: ADPCM (STM32: AN4453) 5](#_Toc119740029)

[1.2.5 Part 1: Why use Dual PWM Output? 6](#_Toc119740030)

[1.2.6 Part 1: Timer Configuration 7](#_Toc119740031)

[1.2.7 Part 2: Converting WAV file to IMA and then C 7](#_Toc119740032)

[1.2.8 Part 2b: Adding files to the project 8](#_Toc119740033)

[1.2.9 Part 2c: Review Timer: STM32G071RB…TIM3 for reference 9](#_Toc119740034)

[1.2.10 Part 2c: Review Timer: STM32WB55RG…TIM2 for this project 9](#_Toc119740035)

[1.2.11 Configuration of TIM2 10](#_Toc119740036)

[1.2.12 Part 2: linker scripts: Added myAudioFiles to the list 11](#_Toc119740037)

[1.2.13 Part 2: Initial Coding works 12](#_Toc119740038)

[1.2.14 Part 3: Focus on interrupt section 13](#_Toc119740039)

[1.2.15 Result: The PWM worked!!! 14](#_Toc119740040)

[1.3 Audio Driver: TS4990 15](#_Toc119740041)

[1.4 Audio Driver: TS4990 15](#_Toc119740042)

[1.5 16/Oct/22: STM32WB55 Nucleo Pack Review 16](#_Toc119740043)

[1.5.1 STM32WB overview 16](#_Toc119740044)

[1.5.2 Video Tutorial (not good) 17](#_Toc119740045)

[1.6 16/Oct/22: Audio Project: PWM to Audio Method (no DAC) 18](#_Toc119740046)

[1.6.1 Useful tutorial 18](#_Toc119740047)

[1.7 15/Oct/22: J-LINK LITE: It working fine so we could test it with STM32 device 19](#_Toc119740048)

# Reference

## Weblink

# Journal 2022

## 19/Nov/22: Newark Order: Nucleo-H723ZG (144 Pin) for deeper ethernet experiments

* We use this op-amp for the minion project. TS4990 is unavailable.

Graphical user interface, text, application

Description automatically generated

## 16/Oct/22: What is codec?

* Need I2C and ISC interface

## 23/Oct/22: STM32 Audio project review

### Objective: Audio Player (WAV)

1. Review the project from the link below and see how this apply to STM32WB55 project kits
2. Is PWM timer sufficient to generate PWM O/P
3. STM32WB55 do not have DAC so we use Timer/PWM instead.
4. Leave BLE out for now.

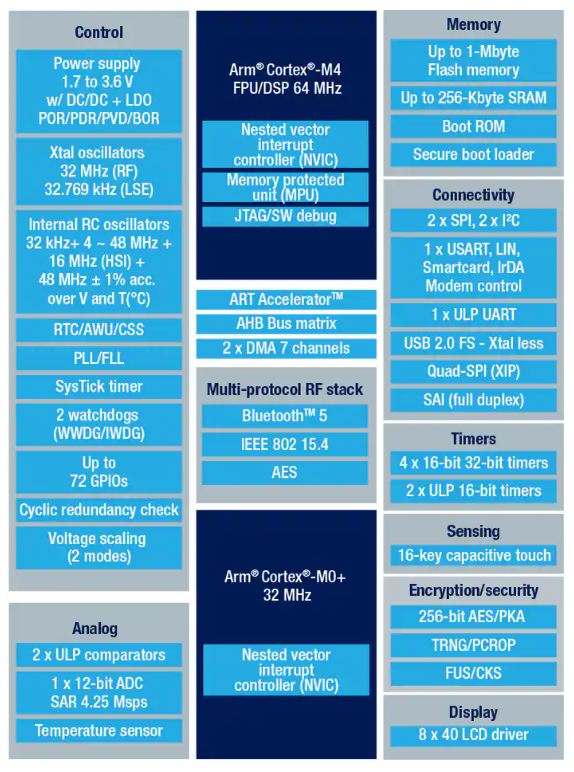
### Links

<https://community.st.com/s/article/how-to-play-audio-files-using-stm32-part-1>

<https://community.st.com/s/article/how-to-play-audio-files-using-stm32-part-2>

<https://community.st.com/s/article/how-to-play-audio-files-using-stm3232-part-3>

### Part 1 Review



1. There are a few ways to play audio files with the STM32, in this article we’ll cover 2 very simple methods: using a low pass filter with the PWM output and using a DAC output. On both cases an audio amplifier and a microphone / speaker will be needed to properly output the audio.
2. STM32WB55 do not have DAC so we use Timer/PWM instead.
3. The use G series STM32: [NUCLEO-G071RB](https://www.st.com/en/evaluation-tools/nucleo-g071rb.html)
4. They use dual 8 bit TIM3 (L and H) to achieve 16 bits

Diagram

Description automatically generated

1. For STM32WB55, we use Timer 2 so we can use this device for the project.

### Part 1: Overview of IMA: ADPCM (STM32: AN4453)

Graphical user interface, text

Description automatically generated with medium confidence

ADPCM = Adaptive differential pulse coded modulation

IMA = Interactive multimedia association

**Overview**

ADPCM is a fixed length codeword audio codec which reduces redundant information from audio waveforms. The information core is separated from the correlated waveform samples by encoding differences between the current samples and the predicted ones. As the correlation between consecutive audio samples is generally high, this method is reasonably effective and preserves good audio quality.

The IMA ADPCM codec is quite popular on PC platforms and a wide range of audio software with different capabilities to process and encode audio data is available.

**What is Parser? : Parser break down data into pieces in accordance to standard protocol**

A parser is **a program that is part of the compiler**, and parsing is part of the compiling process. Parsing happens during the analysis stage of compilation. In parsing, code is taken from the preprocessor, broken into smaller pieces and analyzed so other software can understand it.

**WAV File Notes**

On the Microsoft Windows® platform, a WAVEform (WAV) audio data container is often used to store linear PCM data. The WAV can also be used to store IMA ADPCM audio data.

To use the WAV for storing audio bitstream, a WAV parser has to be implemented in the microcontroller decoding firmware.

This is to unpack raw data so that they can be decompressed. The WAV parser increases the complexity and size of the application and

does not bring many additional benefits when we target basic sound quality.

Fortunately, software is available which can store coded audio bitstreams directly as raw data. An example includes the Sound eXchange (SOX) command-line application for audio manipulation which is distributed under a GNU general-purpose license. SOX is able to:

* + resample input audio data to any target frequency.;
  + encode such data in IMA ADPCM format;
  + save the output bitstream as unformatted raw data

Various input file formats can be used, including PCM WAV, MP3, MP4, OGG, FLAC, and many others. See the SOX documentation for further details.

**Detail about ADPCM**

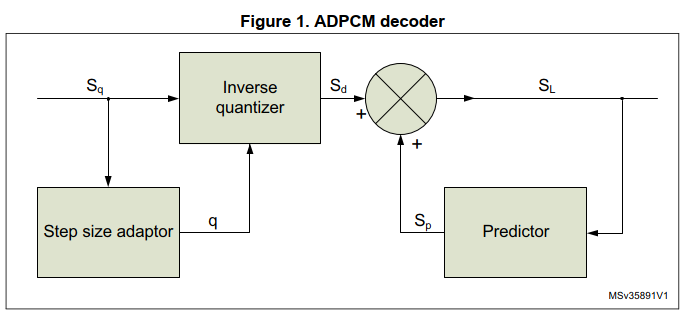
The IMA:ADPCM codec is one such alternative solution. This codec is widely used across different computer platforms, for example in Microsoft® Sound Recorder or Apple® QuickTime®. It was originally offered by Intel/DVI® as an open standard for use by the IMA.

The reference algorithms and recommended formats were initially developed by the digital audio technical working group (DATWG) and refined by the digital audio focus group (DAFG) of the IMA. These groups are no longer active.

The IMA DATWG reference algorithm is less complex than the G.726 algorithm. The number of encoding/decoding CPU cycles needed is reduced by using fixed prediction and by replacing complex floating point mathematical operations by look-up tables.

**ADPCM: Decoder**

* Details are quite technical, refer to STM32: AN4453



**ADPCM: Compression**

* Again details is covered by STM32: AN4453

**Storing data via In Application Programming (IAP)**

* See AN2606 (STM32 microcontroller system memory boot mode) for more details.

**Storing data via link audio data arrays (which we use here)**

* Another way to store audio data to the internal memory is to link audio data arrays with the compiled code. To include raw data in a project, a binary representation of the data has to be converted to ASCII format. Any advanced HEX editor is able to export data in this format which is then readable by ANSI C compilers.

**Hexa Editor**

Graphical user interface, application, table, Excel

Description automatically generated

The output of this operation is a C source file with one array:

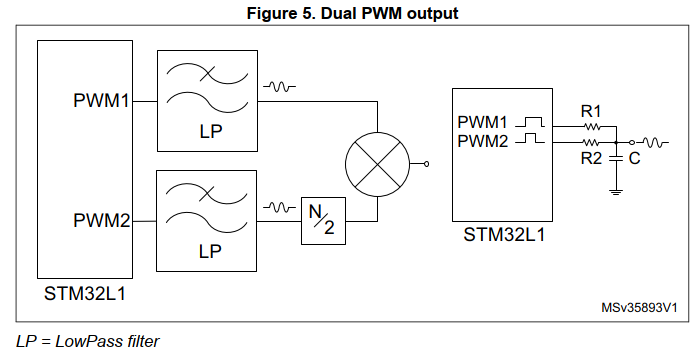
unsigned char rawData[10340] = { ...

This declaration has to be modified, using compiler directives, into an array of constants in the program memory. This is done for the COSMIC compiler as follows:

const unsigned char rawData[10340] = { ...

### Part 1: Why use Dual PWM Output?

* The sampling freq is 15625Hz with 16MHz system clock so we limited to 8 bit resolution from 16 bit data
* The last 8 bit is dropped for single PWM Output.
* But if we use dual PWM, the top 8 bit goes to PWM1 and bottom 8 bit goes to PWM2
* The PWM1 and PWM2 is paired together with product device with weighting factor of 1/2N.
  + This is done by two resistor can caps, see below.
* This in theory recovered to 16 bit resolution.



* RP: Note. In STM32, we suggest to set 16MHz to PWM from default system clock or modify to 16MHz just for audio project
  + Later on we revert back to suitable clock for BLE operation.
* RP: Note. Maybe we don’t need 2nd PWM2 for short speech reply, 8 bit might be sufficient for toy speaker.

relationship between the resistor values, where R2 is obtained by dividing R1 by 256:

* RP: We need audio amp to drive speaker from R1/R2/C circuit
  + R1 = 100K then R2 = 100K/256 = 390R.
  + R1 = 470K then R2 = 1K84 or 1K8.

The actual values used for this demo were: **R1 = 8.2KOhm, R2 = 33Ohm C = 10nF,** but the demo was also tested using only the PWM1 (no R2 connection), which provided a good output audio overall as well.

Audio Amp: STEVAL-CCA037VI based on TS4990IDT. Let find other parts that are ex-stock.

### Part 1: Timer Configuration

The STM32G0 can go up to 64MHz and this application will use the PWM frequency at 125KHz, so we can have each audio sample outputted 8 times to ensure a stable quality audio level, thus rendering the actual audio output frequency as a 15.625KHz – a common sampled audio frequency.  
  
The reason behind these numbers is to ensure the 16bit resolution in an exact amount. Here are the known parameters and how they reflect on the numbers we need:

  
  
   
  
With these simple equations, you can adjust the time and numbers for your own MCU, as much as needed.

### Part 2: Converting WAV file to IMA and then C

We be using compression via adpcm.h/c so WAV need to be converted to IMA and then to C.

Based on STM32 AN4453: Important: Set Sample Freq = 15625Hz.

**STEP 1:** You will need to manually copy the adpcm.c and adpcm.h from the STSW-STM32022 pack into your STM32CubeIDE project: Core/Src and Core/Inc folders.

**STEP 2:** Install sox-14.4.2-win32, this is command based app, so CMD box is required, but went to PowerShell instead.

Using PowerShell (easier than CMD): need to use statement **.\sox.exe** rather than sox.exe as in CMD.

PS E:\014\_WorkSTM32\032\_AudioPlayback\100\_Software\sox-14.4.2-win32> **.\sox.exe "E:\014\_WorkSTM32\032\_AudioPlayback\101\_WavStuff\laugh.wav" -r 15625 "E:\014\_WorkSTM32\032\_AudioPlayback\101\_WavStuff\laugh.ima" gain -12**

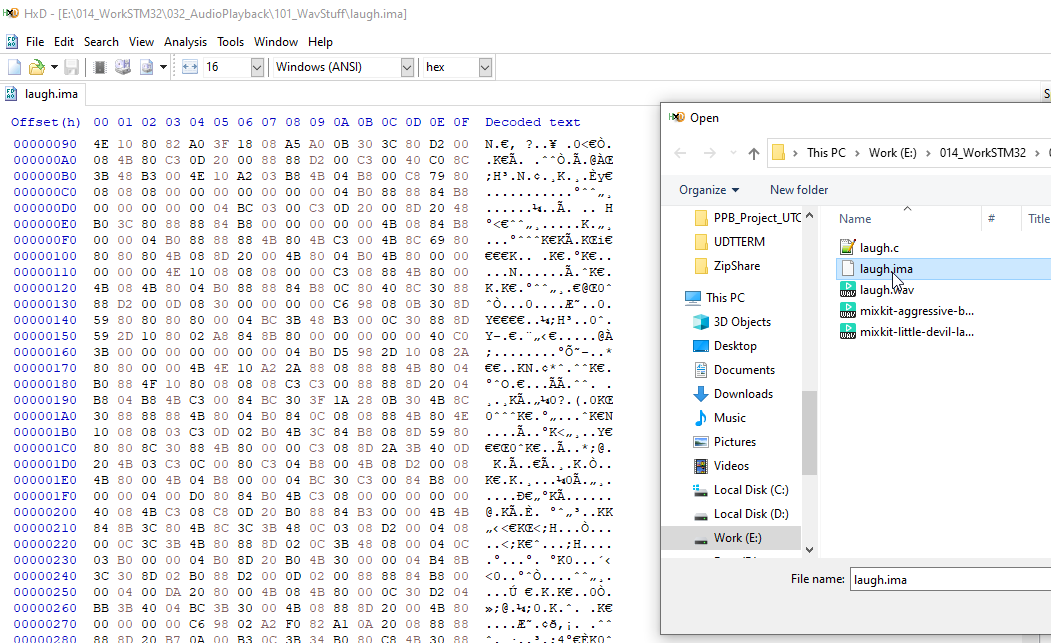
E:\014\_WorkSTM32\032\_AudioPlayback\100\_Software\sox-14.4.2-win32\sox.exe **WARN formats: ima can't encode stereo; setting channels to 1**

**PS** E:\014\_WorkSTM32\032\_AudioPlayback\100\_Software\sox-14.4.2-win32>

NB: Not sure why but the CD in CMD window is not working correctly so I use powershell instead.

**Step 3:** Use HxD (really cool!) to convert the IMA file into C code in array fashion

* HxD is already in the laptop, never use it but now it used!, very easy and wow what a cool feature that is!
* Open file and click ima
* Then export to C file
* That it

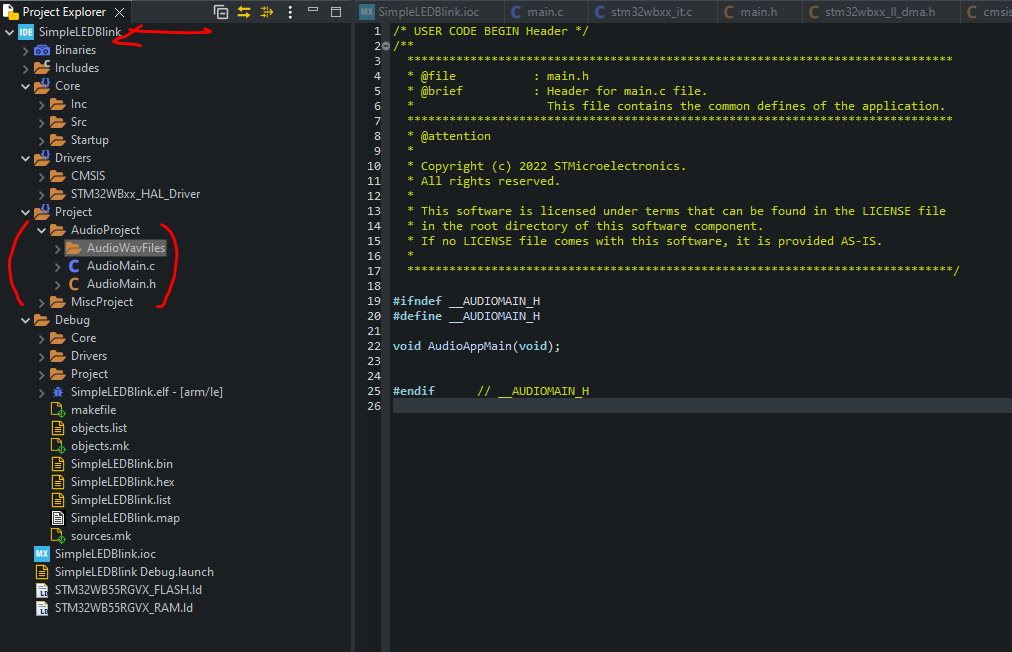


Text

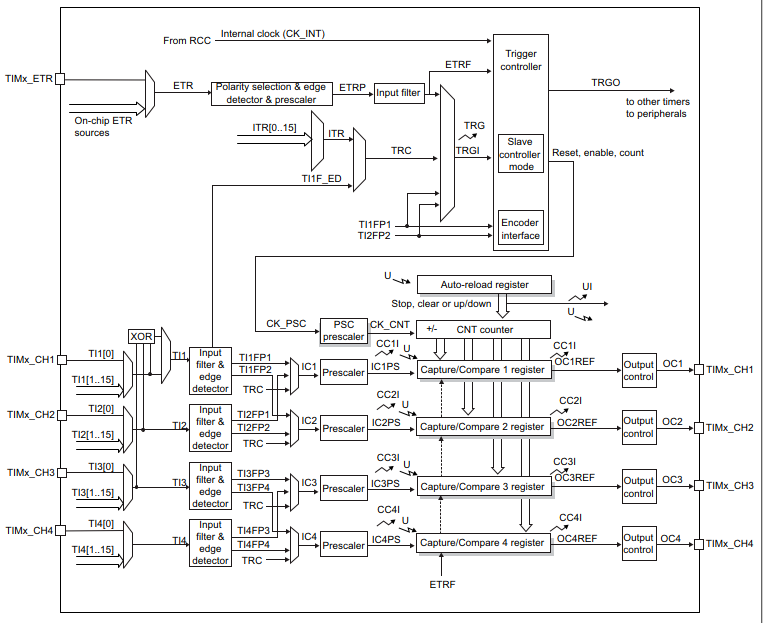
Description automatically generated

### Part 2b: Adding files to the project

Restructured the new project, will rename to AudioProject Workspace later on.



### Part 2c: Review Timer: STM32G071RB…TIM3 for reference



### Part 2c: Review Timer: STM32WB55RG…TIM2 for this project

* Difference is the on-chip ETR sources which is missing, let hope it not used.

Diagram, schematic

Description automatically generated

The STM32G0 can go up to 64MHz and this application will use the PWM frequency at 125KHz, so we can have each audio sample outputted 8 times to ensure a stable quality audio level, thus rendering the actual audio output frequency as a 15.625KHz – a common sampled audio frequency.  
  
The reason behind these numbers is to ensure the 16bit resolution in an exact amount. Here are the known parameters and how they reflect on the numbers we need:

  
  
   
  
With these simple equations, you can adjust the time and numbers for your own MCU, as much as needed.

Based on STM32WB55RG with default clock configuration, the system clock is 32MHz.

### Configuration of TIM2

This link cover all details how to setup the Timer 2 via CubeMX

* <https://community.st.com/s/article/how-to-play-audio-files-using-stm32-part-1>
* Since we operating 32MHz so prescaler is set to 0 (Only set to 1 if system clock upgraded to 64MHz)
* PA0 and PA1 output is defined.
* NVIC Interrupt is used.

Graphical user interface, text, application

Description automatically generated

* The guide suggest to use LL library
  + Decided to limit this to TIM2 only, while the rest are HAL

Graphical user interface, text

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

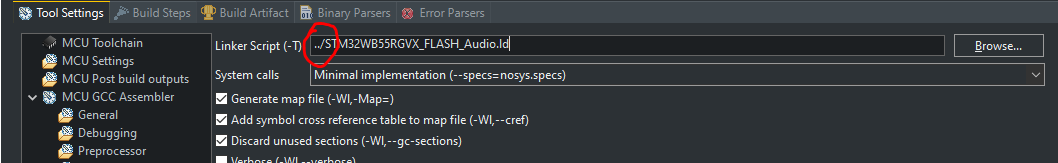
### Part 2: linker scripts: Added myAudioFiles to the list

* Copy and rename to <STM32WB55RGVX\_FLASH\_Audio.ld>

Text

Description automatically generated

* Update linker section, note the special trick for relative file address with ../



### Part 2: Initial Coding works

As per Part 2 guide

**void** AudioAppMain(**void**)

{

//zSYSTICK\_Obj \*zstobj = (zSYSTICK\_Obj \*)zSystickHandle;

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

STM32\_Debug\_Freeze\_Peripheral();

STM32\_Systick1mSec\_Init();

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

//HAL\_IWDG\_Refresh(&hiwdg); // Done by systick loop.

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

//SandBoxStuff();

LoadAudioFiles();

// capture/compare registers (CC1 PWM duty 50%)

TIM2->CCR1 = DEFAULT\_STARTUP\_VAL;

TIM2->CCR2 = DEFAULT\_STARTUP\_VAL;

LL\_TIM\_EnableIT\_UPDATE(TIM2);

TIM2->CCER |= TIM\_CCER\_CC2E | TIM\_CCER\_CC1E;

LL\_TIM\_EnableCounter(TIM2);

**while** (1)

{

// if(!HAL\_GPIO\_ReadPin(BT1\_GPIO\_Port,BT1\_Pin))

// {

// while(!HAL\_GPIO\_ReadPin(BT1\_GPIO\_Port,BT1\_Pin));

**if**(AudioFileToPlay>=1)

{

AudioFileToPlay = 0;

}**else**

{

AudioFileToPlay++;

}

/\* Disable the TIM3 Interrupt \*/

NVIC\_EnableIRQ(TIM2\_IRQn);

// stop the timer

LL\_TIM\_EnableCounter(TIM2);

// }

HAL\_Delay(1000);

}

}

And also update the audio file as per guide.

Below has been extensively upgraded.

/\* Define to prevent recursive inclusion -------------------------------------\*/

**#ifndef** \_\_ADPCM\_H

**#define** \_\_ADPCM\_H

**#define** SAMPLE\_RATE\_DIV 4 // 16000000/256/4 = 15625 Hz

**#define** ADPCMDATA\_DIM 7755

**#define** ADPCMD\_LAUGH\_PI 13322 // Filename: laugh.c

**#define** NUMBER\_OF\_AUDIO\_FILES 1

/\* Includes ------------------------------------------------------------------\*/

**#include** "stm32wbxx.h"

**#include** "main.h"

/\* Exported types ------------------------------------------------------------\*/

**typedef** **union**

{

uint8\_t uBytes[2];

uint16\_t uShort;

} tTwoByte;

**typedef** **struct**

{

uint32\_t AudioFiles[NUMBER\_OF\_AUDIO\_FILES];

uint32\_t AudioSize[NUMBER\_OF\_AUDIO\_FILES];

} AudioElement;

/\* Exported constants --------------------------------------------------------\*/

**extern** **const** **unsigned** **char** Audio\_Laugh[ADPCMD\_LAUGH\_PI];

/\* Exported macro ------------------------------------------------------------\*/

**#define** NELEMS(x) (**sizeof**(x) / **sizeof**((x)[0]))

/\* Exported functions ------------------------------------------------------- \*/

uint8\_t ADPCM\_Encode(int32\_t sample);

int16\_t ADPCM\_Decode(uint8\_t code);

**#endif** /\* \_\_ADPCM\_H\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (C) COPYRIGHT 2009 STMicroelectronics \*\*\*\*\*END OF FILE\*\*\*\*/

No complier error.

### Part 3: Focus on interrupt section

In the interrupt service routine, more specifically in the TIM2 periodic interrupt, we must execute a few actions as well:

1. Include the adpcm header
2. Check if the audio file to be played has changed or not
3. Clear the interrupt flag
4. Decode the audio file
5. Repeat the same position of the audio file 8 times
6. Output the decoded value in the PWM or DAC
7. Once the 8 repetitions are made, move to the next decoded position in the vector
8. If the audio file was fully played, stop the TIM3 interrupt to stop the audio output

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* STM32WBxx Peripheral Interrupt Handlers \*/

/\* Add here the Interrupt Handlers for the used peripherals. \*/

/\* For the available peripheral interrupt handler names, \*/

/\* please refer to the startup file (startup\_stm32wbxx.s). \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* USER CODE BEGIN PV \*/

tTwoByte newSample;

**extern** AudioElement AudioFile;

**extern** uint8\_t AudioFileToPlay;

/\* USER CODE END PV \*/

/\*\*

\* @brief This function handles TIM2 global interrupt.

\*/

**void** TIM2\_IRQHandler(**void**)

{

/\* USER CODE BEGIN TIM2\_IRQn 0 \*/

uint8\_t adpcmSample;

**static** uint16\_t pcmSample;

**static** uint8\_t nibble = 1;

**static** uint8\_t repetition = 0;

**static** uint16\_t sample\_position = 0;

**static** **unsigned** **char** \*RawAudio;

**static** uint8\_t PrevAudioFileToPlay = 0xFF;

**if**(PrevAudioFileToPlay != AudioFileToPlay)

{

PrevAudioFileToPlay = AudioFileToPlay;

nibble = 1;

repetition = 0;

sample\_position = 0;

RawAudio = (**unsigned** **char** \*)AudioFile.AudioFiles[AudioFileToPlay];

}

**if** (LL\_TIM\_IsActiveFlag\_UPDATE(TIM2))

{

LL\_TIM\_ClearFlag\_UPDATE(TIM2);

**if** ((repetition==0) & (sample\_position < AudioFile.AudioSize[AudioFileToPlay]))

{ // new sample is generated

repetition = 7; // reinitialize repetition down counter

**if** (nibble)

{ // first 4 bits of the ADPCM byte decoded

adpcmSample = (uint8\_t)(RawAudio[sample\_position] >> 4);

}

**else**

{ // last 4 bits of the ADPCM byte decoded

adpcmSample = (uint8\_t)(RawAudio[sample\_position] & 0x0F);

sample\_position++ ;

}

nibble = (uint8\_t)(!nibble);/\* indicator inverted mean next interrupt will handle

the second part of the byte. \*/

pcmSample = ADPCM\_Decode(adpcmSample);

// update sample

newSample.uShort = (uint16\_t)32768 + pcmSample;

TIM2->CCR2 = newSample.uBytes[0]; //LSB

TIM2->CCR1 = newSample.uBytes[1]; //MSB

}

**else** **if** (sample\_position < AudioFile.AudioSize[AudioFileToPlay])

{ // repetition 7 more times of the PWM period before new sample, total of times the same value is repeated = 8

repetition--;

// reload Timer with the actual sample value

newSample.uShort = (uint16\_t)32768 + pcmSample;

TIM2->CCR2 = newSample.uBytes[0]; //LSB

TIM2->CCR1 = newSample.uBytes[1]; //MSB

}

**else**

{ // end of the audio clip

/\* Disable the TIM3 Interrupt \*/

NVIC\_DisableIRQ(TIM2\_IRQn);

// stop the timer

LL\_TIM\_DisableCounter(TIM2);

}

}

**return**;

/\* USER CODE END TIM2\_IRQn 0 \*/

/\* USER CODE BEGIN TIM2\_IRQn 1 \*/

/\* USER CODE END TIM2\_IRQn 1 \*/

}

### Result: The PWM worked!!!

Based on PWM waveform, it looked good, but not yet tested.

Adding in the LPF

Diagram

Description automatically generated

relationship between the resistor values, where R2 is obtained by dividing R1 by 256:

The actual values used for this demo were: **R1 = 8.2KOhm, R2 = 33Ohm C = 10nF,** but the demo was also tested using only the PWM1 (no R2 connection), which provided a good output audio overall as well.

The actual implementation allowing for component constraints

R1 = 82K

R2 = 330R

C = 1nF

This seem to work okay, but the audio amp need 3dB cutoff. The C seem to be edge filter.

The sample rate is 125KHz / 8 = 15K625Hz, so this is 3fB cut off

the PWM frequency at 125KHz, so we can have each audio sample outputted 8 times to ensure a stable quality audio level, thus rendering the actual audio output frequency as a 15.625KHz – a common sampled audio frequency.

## Audio Driver: TS4990

Diagram, engineering drawing

Description automatically generated

Diagram, schematic

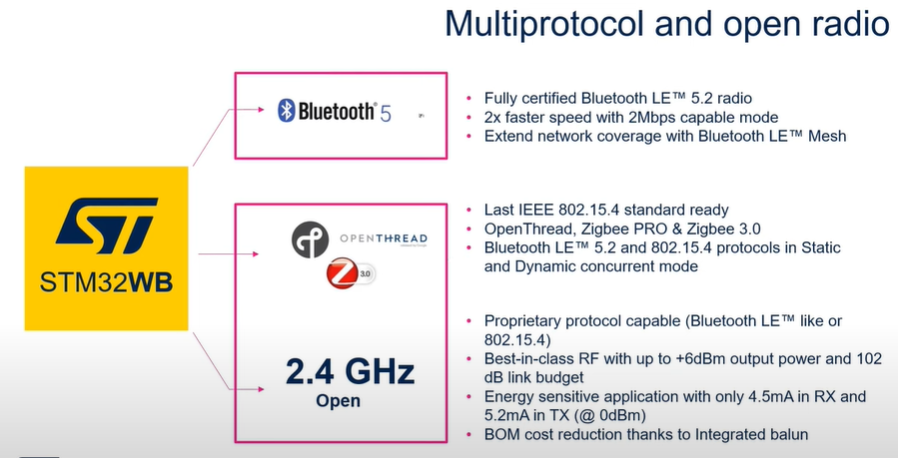
Description automatically generated

## Audio Driver: TS4990

## 16/Oct/22: STM32WB55 Nucleo Pack Review

### STM32WB overview

<https://www.youtube.com/watch?v=7_wBH7mTQ2g&list=PLnMKNibPkDnG9JRe2fbOOpVpWY7E4WbJ->



Timeline

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

### Video Tutorial (not good)

<https://www.youtube.com/watch?v=AcrbUOhApd0>

NB: COM8 for the USB Device



Graphical user interface, application

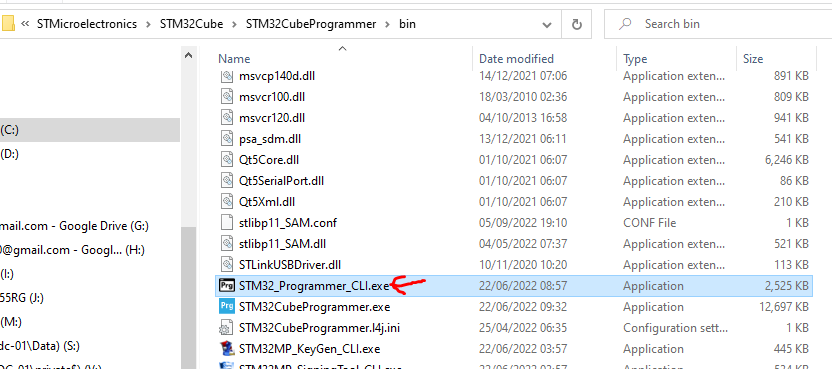
Description automatically generated

How do we program both MCU, it really easy, use STM32Programmer\_CLI.exe (below is STM32CubeProgrammer.exe)

Graphical user interface, application

Description automatically generated

Found STM32Programmer\_CLI.exe



[C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin](file:///C:\Program%20Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin)

## 16/Oct/22: Audio Project: PWM to Audio Method (no DAC)

### Useful tutorial

<https://community.st.com/s/feed/0D73W000001hvMi?fromEmail=1&s1oid=00Db0000000YtG6&s1nid=0DB0X000000DYbd&s1uid=0053W000002o7IP&s1ext=0&emkind=chatterCommentNotification&emtm=1664524031194>

* Hello here the links to the 3 parts:

[Part 1](https://community.st.com/s/article/how-to-play-audio-files-using-stm32-part-1)

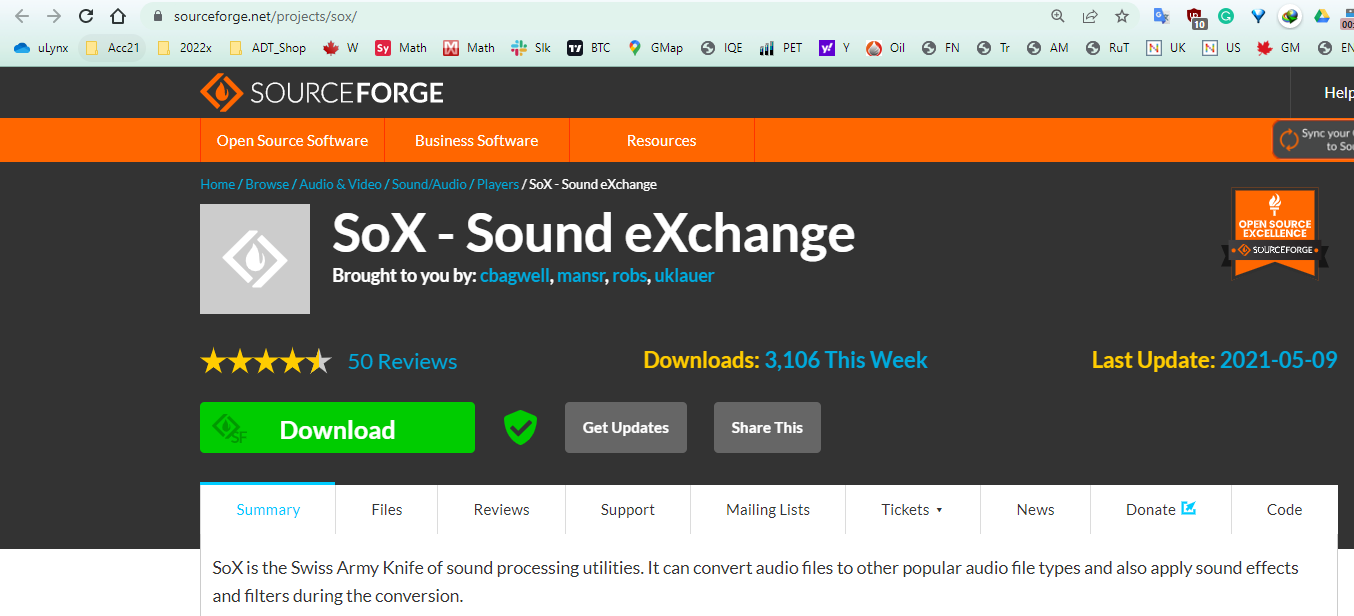
[Part 2](https://community.st.com/s/article/how-to-play-audio-files-using-stm32-part-2)

[Part 3](https://community.st.com/s/article/how-to-play-audio-files-using-stm3232-part-3)

<https://community.st.com/s/article/how-to-play-audio-files-using-stm32-part-1>

<https://community.st.com/s/article/how-to-play-audio-files-using-stm32-part-2>

<https://community.st.com/s/article/how-to-play-audio-files-using-stm3232-part-3>



## 15/Oct/22: J-LINK LITE: It working fine so we could test it with STM32 device

Found this in the box, can we use this ?

* Not, sure it supplied by a vendor not sure what restriction in use.
* It seem to be working fine
* May need SW to update firmware if allowed.

<https://www.segger.com/products/debug-probes/j-link/models/j-link-lite/overview/>

<https://www.segger.com/products/debug-probes/j-link/models/j-link-lite/j-link-lite-cortex-m-5v/>

After installing J-LINK Software pack, it automatically updated the firmware from 2014 to 2018.

