AN2931
Application note

Implementing the ADPCM algorithm in high-density STM32F103xx microcontrollers

Introduction

This application note describes the ADPCM audio firmware codec, and provides a demonstration firmware showing how to play ADPCM files using the high-density STM32F103xx I^2S feature with an external DAC.

This application note is based on “AN2739: How to use the high-density STM32F103xx microcontroller to play audio files with an external I^2S audio codec”. The features linked to the SPI, I^2S and the external DAC will be skipped here because they are already explained in AN2739. It is therefore highly recommended to refer to AN2739 to fully understand this application note.

Contents

List of tables

List of figures

1 ADPCM algorithm

1.1 General overview

Adaptive differential pulse-code modulation, or simply ADPCM, is an audio algorithm for waveform coding, which consists in predicting the current signal value from previous values, and transmitting only the difference between the real and the predicted value.
In plain pulse-code modulation (PCM), the real or actual signal value is transmitted.

The advantage of ADPCM is that the difference between the predicted signal value and the actual signal value is usually quite small, which means it can be represented using fewer bits than the corresponding PCM value.

Depending on the desired quality and compression ratio, the differential signal is quantized using 4 (2 bit), 8 (3 bit), 16 (4 bit) or 32 (5 bit) levels.

Many implementations of the ADPCM algorithm exist. They differ by the quantization and the prediction patterns.

In this application note, we provide a 4-bit quantization algorithm, developed by the Interactive Multimedia Association (IMA), IMA ADPCM.

IMA ADPCM was chosen for several reasons:

- it can operate at different sampling rates between 8 kHz and 44.10 kHz
- it guarantees good quality with low CPU usage and memory size requirements
- it has widespread implementations, for instance in the Windows and MAC operating systems

The IMA ADPCM algorithm is fully described in a document published by the IMA Digital Audio Focus and Technical Working Groups: *“Recommended Practices for Enhancing Digital Audio Compatibility in Multimedia Systems” revision 3.*

1.2 ADPCM algorithm implementation

The IMA ADPCM algorithm provided in this application note is used to encode audio files that have the following specification:

- Audio format: PCM
- Audio sample size: 16 bits
- Channels: 1 (mono)
- Audio sample rate: from 8 kHz to 44.1 kHz

Each 16-bit PCM sample is encoded into a 4-bit ADPCM sample, which gives a compression rate equal to ¼.

The implementation of the IMA ADPCM algorithm consists in two functions, one that codes and the other that decodes the audio samples.

The ADPCM firmware is composed of two files:

a) *adpcm.c*: it contains the source code of the two ADPCM functions that perform the coding and decoding.

b) *adpcm.h*: it is the header file of *adpmc.c*. It should be included in the files where the ADPCM functions are called.

1.3 ADPCM algorithm functions

Table 1 describes the ADPCM functions.

Table 1. ADPCM algorithm functions

Function name	Description
`ADPCM_Encode`	Encodes a 16-bit PCM sample into a 4-bit ADPCM sample.
`ADPCM_Decode`	Decodes a 4-bit ADPCM sample into a 16-bit PCM sample.

1.3.1 `ADPCM_Encode` function

Table 2 describes the `ADPCM_Encode` function.

Table 2. `ADPCM_Encode` function

Function name	**`ADPCM_Encode`**
Prototype	`uint8_t ADPCM_Encode(int16_t sample);`
Behavior description	Encodes a 16-bit PCM sample into 4-bit ADPCM sample
Input parameter	Sample: a 16-bit PCM sample
Output parameter	None
Return parameter	The encoded ADPCM 4-bit sample loaded into a byte

The `ADPCM_Encode` function returns a byte containing the 4-bit ADPCM sample. The software has to store every two ADPCM samples into one byte to save memory space.

Example:

```
//Input: pcm_sample1 and pcm_sample2; two 16-bit PCM samples.
//Output: adpcm_byte; two 4-bit ADPCM samples stored into one byte.

uint8_t code;
/* Encode the first 16-bit sample */
code = ADPCM_Encode(pcm_sample1);
/* Store the first 4-bit sample */
adpcm_byte = code;
/* Encode the second 16-bit sample */
code = ADPCM_Encode(pcm_sample2);
/* Store the second 4-bit sample */
adpcm_byte |= (code << 4);/* The adpcm_byte contains two 4-bit ADPCM
samples */
```

1.3.2 `ADPCM_Decode` function

Table 3 describes the `ADPCM_Decode` function.

Table 3. **`ADPCM_Decode` function**

Function name	`ADPCM_Decode`
Prototype	`int16_t ADPCM_Decode(uint8_t code);`
Behavior description	Decodes a 4-bit ADPCM sample into a 16-bit PCM sample
Input parameter	Code: an 8-bit data item whose 4 LSBs contain the encoded ADPCM sample
Output parameter	None
Return parameter	A 16-bit PCM sample

The input of the `ADPCM_Decode` function is a byte that contains the 4-bit ADPCM sample. The software has to extract the 4-bit ADPCM data and store them into a byte before calling the `ADPCM_Decode` function.

Example:

```
//Input: adpcm_byte; two 4-bit ADPCM samples stored into one byte.
//Output: pcm_sample1 and pcm_sample2; two 16-bit PCM samples.

uint8_t code;
/* Extract the first ADPCM 4-bit sample */
code = (adpcm_byte & 0x0F);
/* Decode the first ADPCM sample */
pcm_sample1 = ADPCM_Decode(code);
/* Extract the second ADPCM sample */
code = (adpcm_byte >> 4);
/* Decode the second ADPCM sample */
pcm_sample2 = ADPCM_Decode(code);
```

2 Implementation example

2.1 Description

You can run the provided example on the STM3210E-EVAL board. It is a typical audio application that consists in:

- first encoding a PCM file into the ADPCM format using a PC software (This software is called **muse.exe**, and was developed by ST. It implements the same ADPCM algorithm encoder as the one that runs on STM32F103xx performance line devices. **muse.exe** is included in the zipped software package that comes with this application note: please refer to *Section 2.2* for how to use this tool.)
- then loading the encoded file into a Flash memory
- and finally decoding the file and playing it using the high-density STM32F103xx microcontroller.

On the STM3210E-EVAL board, the 128 Mbit NOR Flash memory stores the ADPCM file and the firmware decoder that runs on STM32F103xx devices. It then decodes the stream and, with the I^{2}S external DAC, plays back the audio samples.

The hardware environment is based on the application note “AN2739: How to use the high-density STM32F103xx microcontroller to play audio files with an external I^{2}S audio codec”, so please refer to that application note for more information.

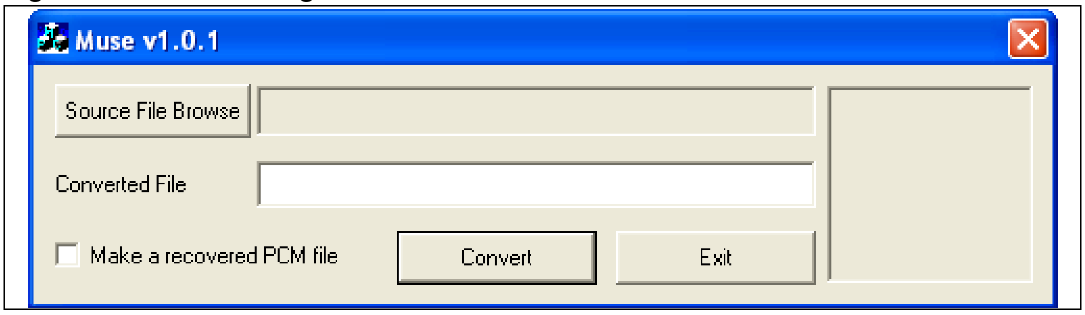
To run this demo, three steps are required:

1. generating the ADPCM file
2. loading the ADPCM file into the NOR Flash memory
3. playing the ADPCM file

2.2 ADPCM file generation

The application note package contains a PC software called Muse to encode PCM files into the ADPCM format.

Figure 1. ADPCM file generator



If you run the *Muse.exe* program and press the **Source File Browse** button, the **File Selection** dialog appears, prompting you to select a WAV source file. As mentioned before, the WAV file should meet the following requirements:

- Audio format: PCM
- Audio sample size: 16 bits
- Channels: 1 (mono)
- Audio sample rate: from 8 kHz to 44.1 kHz

If your WAV file does not satisfy these conditions, you must previously convert it using an audio converter tool, like *sndrec32.exe*, which is the Microsoft® default program for audio records.

Once you have selected the file to be converted, the **New File Selection** dialog appears, prompting you to select the save location of your ADPCM converted file.

You can retrieve the PCM format of the encoded audio file by ticking the **Make a recovered PCM file** check box. In this case, the generated ADPCM file is decoded and formatted into a WAV file. This WAV file is called *Recover.wav* and it is generated in the same directory as your WAV source file. In this way you can listen to both the original WAV file and the recovered one to compare the sound quality.

2.3 ADPCM file loading

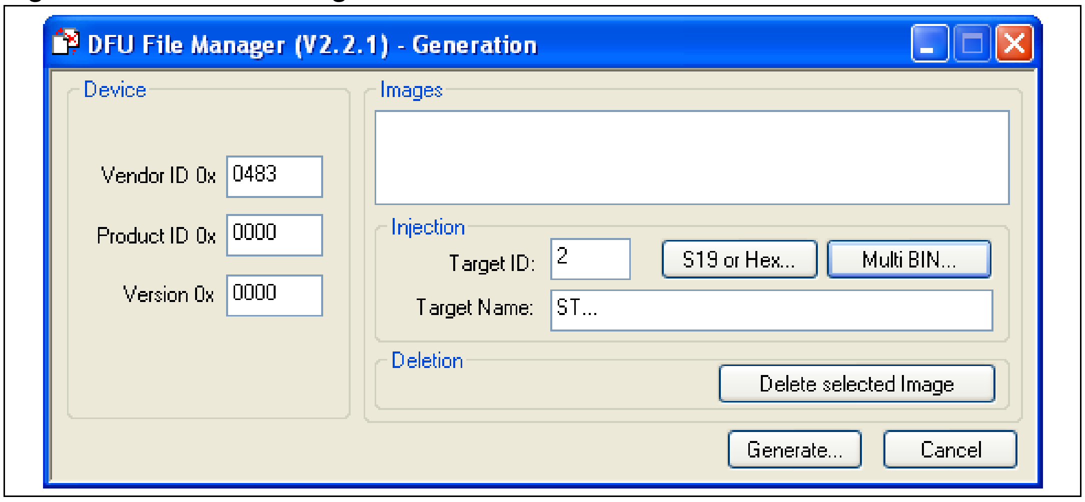
Use the DfuSe software to load the ADPCM file into the NOR Flash memory. To do so, two steps are necessary:

1. generating the .dfu image
2. loading the .dfu image into the NOR Flash memory

Generating the .dfu image

To generate the .dfu image, you have to use the DFU File Manager software provided within the DfuSe package. For that, open the DFU File Manager program and choose the DFU file generation option. The window shown in *Figure 2* opens. You have to select 2 for the target ID, meaning that the file is loaded into the NOR Flash memory. Then, click on the **Multi BIN...** button to open the source browser and select your ADPCM file.

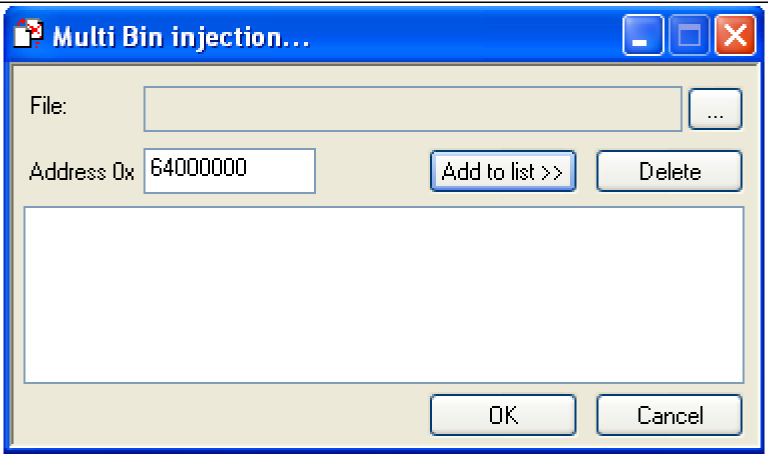
Figure 2. DFU File Manager



Vendor ID 0x 0483
Product ID 0x 0000
Version 0x 0000
Target ID: 2
Target Name: ST...

The NOR Flash memory is mapped on the FSMC interface from address 0x6400 0000, which is the start address of the ADPCM file storage area. So, you have to set the address of the .dfu image to 0x6400 0000 (see *Figure 3*). Finally, click on **Add to list** and then **Generate** to finish the DFU file generation.

Figure 3. DFU File Manager source browser



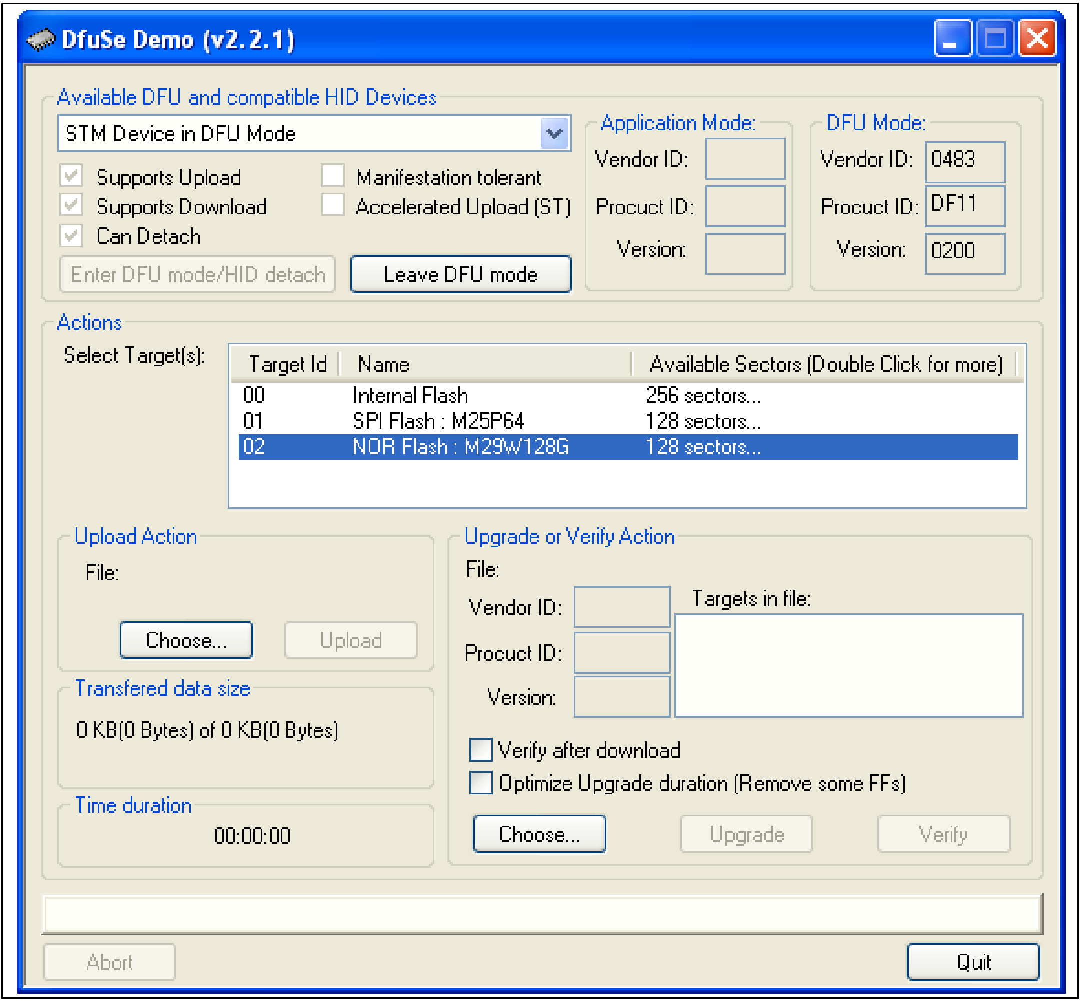
Address 0x 64000000

Loading the DFU image

To load the DFU image into the NOR Flash memory you have to:

1. connect the STM3210E-EVAL board to a PC using a USB cable
2. load and run the device firmware upgrade (DFU) demonstration on the STM3210E-EVAL board. You can find this firmware in the STM3210xxx USB development kit
3. start the DfuSe software on the PC (see *Figure 4*)

Figure 4. DfuSe software



STM Device in DFU Mode

- [✓] Supports Upload
- [✓] Supports Download
- [✓] Can Detach
- [ ] Manifestation tolerant
- [ ] Accelerated Upload (ST)

	Application Mode	DFU Mode
Vendor ID:		0483
Procuct ID:		DF11
Version:		0200

Select Target(s):

Target Id	Name	Available Sectors (Double Click for more)
00	Internal Flash	256 sectors...
01	SPI Flash : M25P64	128 sectors...
02	NOR Flash : M29W128G	128 sectors...

Upload Action — File:

Transfered data size: 0 KB(0 Bytes) of 0 KB(0 Bytes)

Time duration: 00:00:00

Upgrade or Verify Action — File:
Vendor ID:
Procuct ID:
Version:
Targets in file:

- [ ] Verify after download
- [ ] Optimize Upgrade duration (Remove some FFs)

Select the NOR Flash memory as target, then click on the **Choose** button located in the **Upgrade or Verify Action** area to select your ADPCM DFU image. Finally, click on **Upgrade** to perform the download.

2.4 ADPCM file playing

At this point, you have succeeded in loading the ADPCM file into the NOR Flash memory. It is now time to start the playback demo. Go to the application note package and open the *ADPCM_AN* folder. Open the folder project, select the suitable project workspace and open it in your toolchain. Build all the source files, load the project image and then run the program.

The LCD on the STM3210E-EVAL board will display the audio commands while the joystick and the key buttons will be used to control the demo. The allowed audio commands are: play, pause, stop, forward, rewind and volume setting.

You can select the audio output interface, which can be the speaker provided on the STM3210E-EVAL board, or headphones. Better audio quality is achieved with headphones. You can connect them to the STM3210E-EVAL board via the audio jack.

Note that to rewind and forward through the audio file, you have to use the joystick buttons as follows:

- push the joystick to the right to forward
- push the joystick to the left to rewind

3 Conclusion

This application note describes how to implement the IMA ADPCM firmware audio codec dedicated to STM32F103xx devices. It also provides an example of use that consists in decoding an ADPCM-encoded audio stream stored in a Flash memory, and playing back the decoded samples.

The high-density STM32F103xx was used in this application note because it features an I^2S interface that allows communication with the external DAC available on the STM3210E-EVAL evaluation board.
It is however possible to run this ADPCM firmware codec and play audio files with other STM32F10xxx devices that do not feature an I^2S peripheral, by using the PWM output.
In this case, you simply need to update the provided decoding example for the STM3210B-EVAL evaluation board, and use the PWM output for audio playback.

4 Revision history

Table 4. Document revision history

Date	Revision	Changes
04-Mar-2009	1	Initial release.
30-Apr-2009	2	In code, `u8` and `s16` updated to `uint8_t` and `int16_t`, respectively.