# AN4453
# Application note

# Implementing the ADPCM algorithm in STM32L1xx microcontrollers

## Introduction

This application note describes a simple and easy-to-use solution to facilitate design and development of products with an audio output.

Audio data usually are very demanding in terms of memory storage, the needed memory space can be reduced by compressing the audio data via a specified compression method.

Processing these data is a complex task, requiring powerful computing machines such as DSP or audio chip, while standard microcontrollers do not offer either a large memory or a high computing performance.

The software solution (STSW-STM32022) offered in this application note reconstructs audio signals from compressed samples. A simple audio codec based on an adaptive differential pulse coded modulation (ADPCM) algorithm is used, taking advantage of the powerful ARM® Cortex®-M3 core.

The digital samples are then converted into an analog signal by some external or on-chip digital-to-analog converter (DAC). There are a lot of applications where the high fidelity audio output is not a priority and/or the use of an external DAC is not suitable for cost-effective solutions or for low-power applications. To reduce costs and to offer a correct quality audio, an ideal solution is to generate a signal using pulse width modulation (PWM).

An example where firmware and application hardware design are described is included in this application note, to enable easy porting of the offered solution to the final application.

# Contents


**1 ADPCM codec . . . . . 5**

1.1 Adaptive differential pulse coded modulation (ADPCM) . . . . . 5

1.2 Interactive multimedia association (IMA) ADPCM . . . . . 5

1.3 IMA ADPCM decompression . . . . . 6

1.4 IMA ADPCM compression . . . . . 7

**2 Audio data encoding and storing . . . . . 8**

2.1 Internal Flash memory used for audio data storing . . . . . 8

**3 STM32L1 audio output hardware overview . . . . . 10**

**4 Conclusion . . . . . 12**

**5 References . . . . . 13**

**6 Revision history . . . . . 14**

# List of tables

Table 1. Document revision history . . . . . 14

# List of figures

Figure 1. ADPCM decoder . . . . . 6
Figure 2. ADPCM encoder . . . . . 7
Figure 3. Hexa editor export to C source code . . . . . 9
Figure 4. Single PWM output . . . . . 10
Figure 5. Dual PWM output . . . . . 11

# 1 ADPCM codec

## 1.1 Adaptive differential pulse coded modulation (ADPCM)

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 ADPCM codec, which is based on coding waveforms in time domains, is much less complex than codecs based on voice perception (vocoders) that operate in frequency domains.

The ADPCM algorithm has been developed for speech coding. It is implemented in several steps and is used in telephone systems like ITU-T G.726 (covering CCITT G.721 and G.723 standards). G.726 and similar telecommunication codecs offer several advanced features which are necessary for successful transmission and streaming. Such features include synchronous coding adjustment, tone detection to carry data modem signals, adapted speed control and recovery from transmission error conditions. These features lead to additional algorithm complexity which either requires the use of a digital signal processor (DSP) or lowering the encoding/decoding sample rate. Fortunately, such features do not bring many additional benefits to the microcontroller for coding/decoding the audio or speech signals. Less complex alternatives can be found.

## 1.2 Interactive multimedia association (IMA) ADPCM

The IMA[1] 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. The implementation of the ADPCM codec presented in this application note is compatible with the IMA reference algorithm published by the IMA DATWG/DAFG in the “Recommended Practices for Enhancing Digital Audio Compatibility in Multimedia Systems”, revision 3.00 (see *References*).

1. IMA is a computer/audio/video industry trade association that has worked to promote multimedia application development.

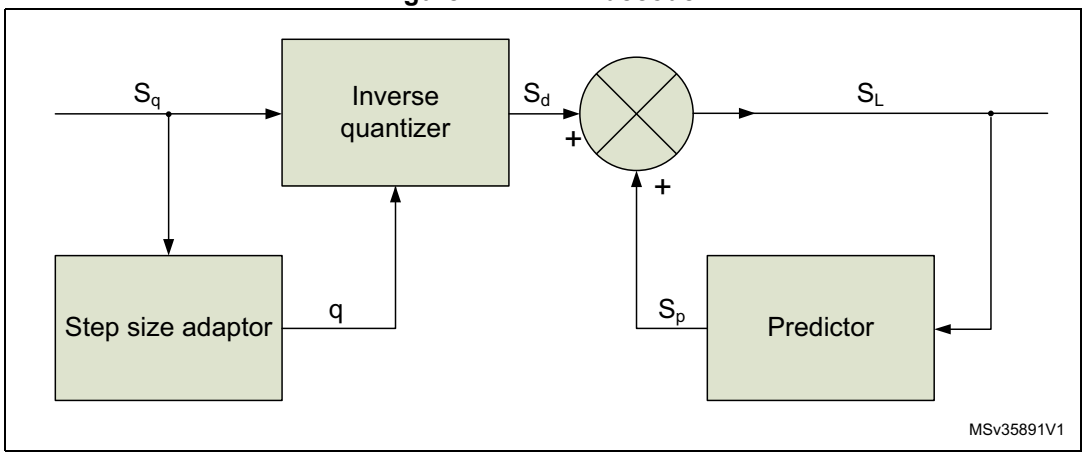
## 1.3 IMA ADPCM decompression

The IMA ADPCM decompression algorithm compresses samples which are stored as 4-bit signed two's complement values. The result is decompressed samples in linear 16-bit two's complement format.

The input sample ($S_q$) is dequantized using an inverse quantizer with an adapted step (q) to obtain a difference ($S_d$). To reduce quantization errors, one half of the step size is added to $S_d$. The resulting decompressed linear sample is the sum of $S_d$ and the predicted sample ($S_p$). As $S_p$ is a simple zero order hold function, $S_d$ is simply added to the previous output value. The step size is adapted according to $S_q$. The decoder preconditions are that $S_p$ is cleared and that the quantization step is set to smallest one.

This flow is sketched in *Figure 1*.

**Figure 1. ADPCM decoder**

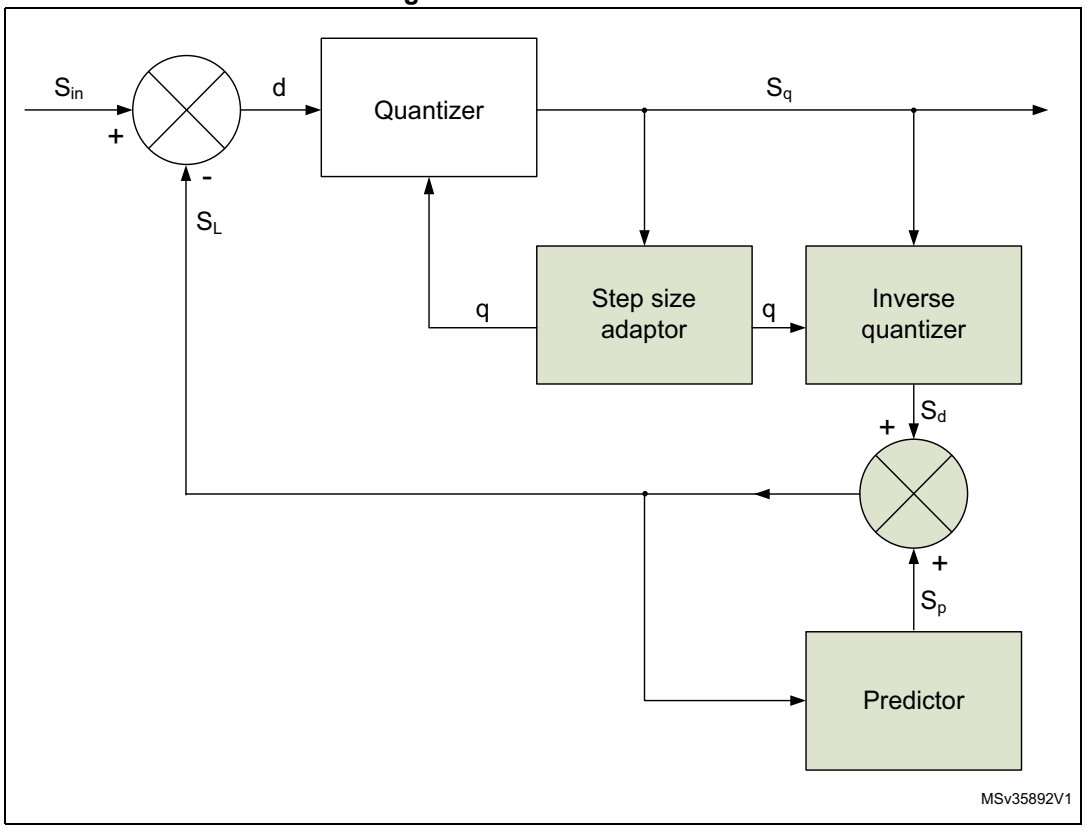


## 1.4 IMA ADPCM compression

The IMA ADPCM compression algorithm encoder calculates a difference (d) between a 16-bit input linear sample ($S_{in}$) and a predicted linear sample ($S_L$). The difference is quantized to a 4-bit compressed output sample ($S_q$) by using an adapted quantization step (q). This step is the same one as that used by the decoder.

*Figure 2* provides a flowchart of the ADPCM encoder. It also shows the decoder building blocks with the functionality described in *Section 1.3* (see shaded components). As the encoder features some blocks identical to those of the decoder and produces identical results, there is no need to store prediction information. The quantized difference is the only core information about the coded samples that needs to be stored or transmitted.

**Figure 2. ADPCM encoder**



# 2 Audio data encoding and storing

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. The problematic part of encoded file storage is the file format of the ADPCM bitstream as it is not standardized in [1]. On microcontrollers, raw data without any formatting is preferred to obtain the easiest manipulation with the coded audio file.

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.

The sample command below shows the input parameters used to resample data to:

- target frequency 15625 Hz
- reduce volume by -12 dB
- compress data by using the IMA ADPCM codec.

```
sox inputfilename -r 15625 outputfilename.ima gain -12
```

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

## 2.1 Internal Flash memory used for audio data storing

The internal memory can be used to store short audio waveforms. One 16-bit PCM sample takes half a byte when compressed by the IMA ADPCM codec. The memory requirement can be evaluated using *Equation 1*.

**Equation 1**

$$\text{Memory} = \text{samplerate} \times \text{length} / 2$$

For example, a 5 seconds ADPCM bitstream with an 8 kHz sample rate uses 20 KBytes of memory.

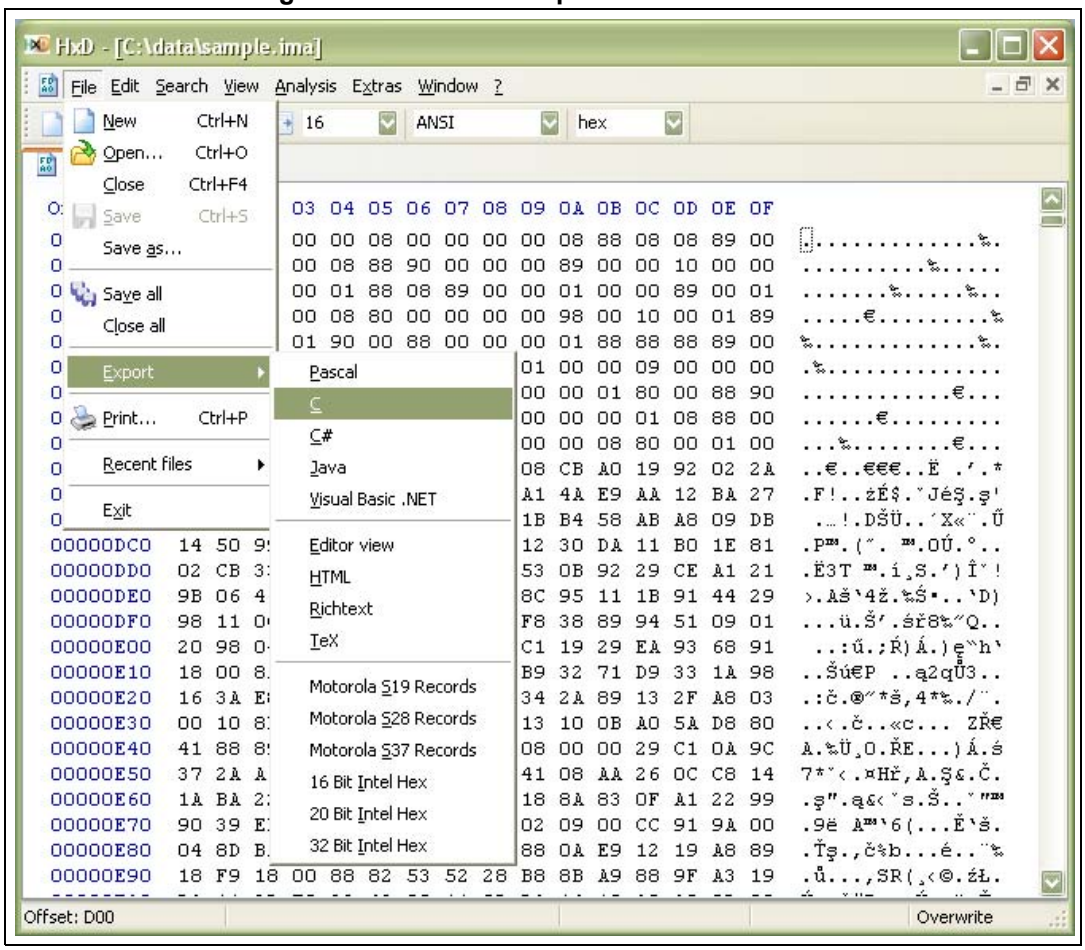
The audio data can be saved to the internal memory by using “in-application programming” (IAP) or the bootloader. See AN2606 (STM32 microcontroller system memory boot mode) for more details.

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. The TxD hexa editor (see *References*) is an easy-to-use software, with a free license, no restrictions for commercial use, and which can be freely distributed.

The processing of audio data, which has to be included in a project, consists of loading binary data into the editor and exporting it as a C source code, as shown in *Figure 3*.

**Figure 3. Hexa editor export to C source code**




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:

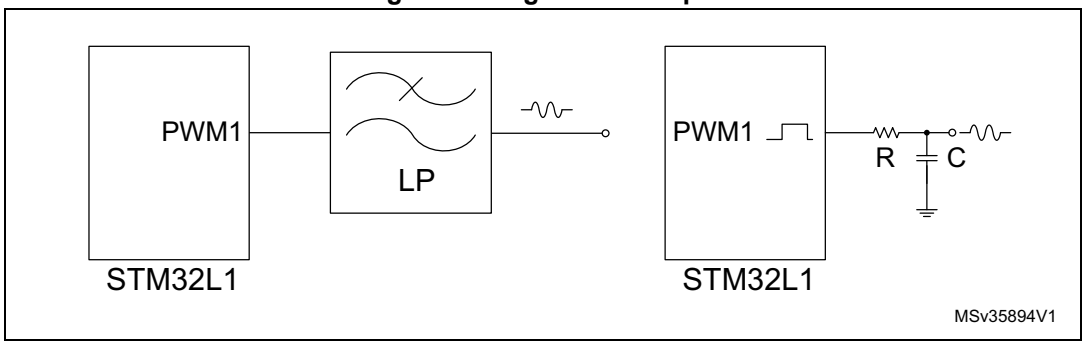
```
const unsigned char rawData[10340] = { ...
```

For more details, see the firmware project associated with this application note on www.st.com/mcu.

# 3 STM32L1 audio output hardware overview

The hardware solution benefits from the STM32L1 16-bit timer with pulse width modulation (PWM) output which can operate in center-aligned mode. Only a few external components are needed to build an application with audio output, for example, a reconstruction filter is needed to reduce the high frequency noise in the PWM output frequency spectrum (*Figure 4*).

**Figure 4. Single PWM output**




*Note: LP = LowPass filter*

The reconstruction filter should be set up as a low pass filter with a cutoff frequency close to half that of the sampling frequency ($f_{sampling}$). In the current example, the PWM frequency is four times higher than the sampling frequency. This is to keep an adequate margin in the frequency band for elimination of high frequency noise components by the reconstruction filter. The higher the band stop attenuation (steepest frequency characteristic transition), the lower the output noise, and consequently, the higher the resolution.

The achievable resolution (N) of the PWM signal is given by the frequency of the PWM signal (fPWM) and the system clock frequency of 16 MHz which is given by the high-speed internal oscillator (fHSI). This relation is expressed by *Equation 2*.

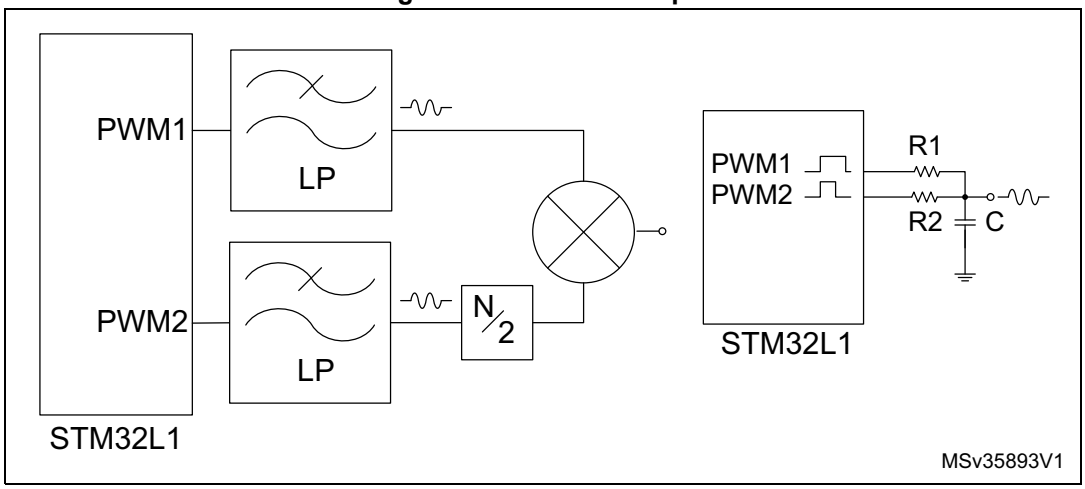
**Equation 2**

$$2^N = \frac{f_{HSI}}{4f_{sampling}}$$

By evaluating *Equation 2*, where we have a sampling frequency of 15625 Hz and a 16-MHz system clock, we can only get an 8-bit resolution. Due to the 16-bit output samples, the lower byte could be lost. This can be audible with higher quality speakers and a good reconstruction filter.

To avoid such reduction, a second PWM2 output is used to generate a signal corresponding to the lower byte. This signal is multiplied by an adequate weighting factor (1/2N) and subsequently added to the signal corresponding to the higher byte PWM1. Theoretically, this allows the 16-bit resolution to be reached. This principle is described in *Figure 5*.

**Figure 5. Dual PWM output**




*Note: LP = LowPass filter*

The simplest form of the reconstruction filter is the RC low pass filter. Concerning the choice of the RC couple, it is not possible to provide an exact number for it. It is closely linked to the impedance of the speaker/headphones connected to the device.

The value of R2 is obtain by dividing R1 by 256. To obtain an effective signal reconstruction, the lowest cutoff frequency must be around the sampling frequency.

If some more efficient filtering is required in the application to reduce noise, you must move to second order analog filters, or active filtering using an operational amplifier. However, the price and complexity of the final application will also be increased.

# 4 Conclusion

This application note offers an easy-to-use application solution for STM32L1MCUs with good quality audio/speech output. To effectively reduce memory space which allows an audio bitstream to be stored, an IMA ADPCM codec is implemented. Short audio clips can be stored directly in the internal program memory. To store long length audio clips, an external memory is needed.

The STM32L1 audio output solution was developed with the aim to limit external components. This solution benefits from the STM32L1 timer with dual PWM output to achieve 16-bit resolution.

# 5 References

IMA Digital Audio Focus and Technical Working Group, Recommended Practices for Enhancing Digital Audio Compatibility in Multimedia Systems, revision 3.00, October 21, 1992.

Maël Hörz, HxD, revision 1.7.7.0 - hexa editor.

# 6 Revision history

Table 1. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 11-Feb-2015 | 1 | Initial release |
| 01-Apr-2015 | 2 | Updated software RPN on Cover page |