Application Report
SLAA361A – July 2007 – Revised August 2018

Speech and Sound Compression and Decompression With MSP430™ MCUs

MSP430 Applications

ABSTRACT

This document describes an IMA adaptive differential pulse code modulation (ADPCM) compression and decompression algorithm and the steps to use the ADPCM library on MSP430™ microcontrollers (MCUs). This document describes the use of the ADPCM library for two voice recorder examples that use the signal-chain-on-chip feature of the MSP430 MCUs.

Source code and sample files are available for download from http://www.ti.com/lit/zip/slaa361.

Contents

List of Figures

Trademarks

MSP430 is a trademark of Texas Instruments.
IAR Embedded Workbench is a registered trademark of IAR Systems.
All other trademarks are the property of their respective owners.

1 Introduction

Sound recorders are implemented with relative ease on microcontrollers (MCUs) that have an integrated analog-to-digital converter (ADC). A microphone followed by an amplifier captures the sound, which is fed to the analog input of the ADC. The recorded sound can then be stored in memory (flash or RAM). A button press can trigger the MCU to play the recorded sound. To play the sound, software moves the stored data to a digital-to-analog converter (DAC) followed by an audio power amplifier.

Such a sound recorder can be easily realized using the MSP430 MCUs. The MSP430 MCU takes advantage of the integrated peripherals to form an on-chip analog signal chain. In addition, the CPU of the MSP430 is powerful enough to perform compression of the recorded sound.

This application report and the accompanying code uses the MSP430FG4618 microcontroller and the MSP430F169 microcontroller for demonstration purposes. The concepts and code in this application report apply to any MSP430 MCU that has at least one ADC input and a DAC.

2 Compression and Decompression Algorithms

The simplest way to make a voice recorder is to store the analog-to-digital conversion results (for example, 12-bit samples) directly in the flash memory. Most of the time, the audio data does not use the complete ADC range, which means that redundant information is stored in the flash memory. Compression algorithms remove this redundant information and reduce the data that must be stored.

Adaptive differential pulse code modulation (ADPCM) is a compression algorithm. Various ADPCM algorithms exist, all of which use differential codes and adaptation of the step-size of the quantizer. Section 2.1 describes differential pulse code modulation (DPCM), and Section 2.2 describes the IMA ADPCM algorithm, which is used in the associated code.

2.1 *Differential Pulse Code Modulation (DPCM)*

DPCM encodes the analog audio input signal based on the difference between the current sample and the previous sample. Figure 1 shows a DPCM encoder and decoder block diagram. In this example, the signal difference, d(n), is determined using a signal estimate, Se(n), rather than the previous input. This method makes sure that the encoder uses the same information available to the decoder. If the true previous input sample were used by the encoder, an accumulation of quantization errors could occur. This accumulation would lead to a drift of the reconstructed signal from the original input signal. Use of the signal estimate in Figure 1 prevents drift from the original input signal in the reconstructed signal, Sr(n). The reconstructed signal, Sr(n), is the input to the predictor, which determines the next signal estimate, Se(n+1).

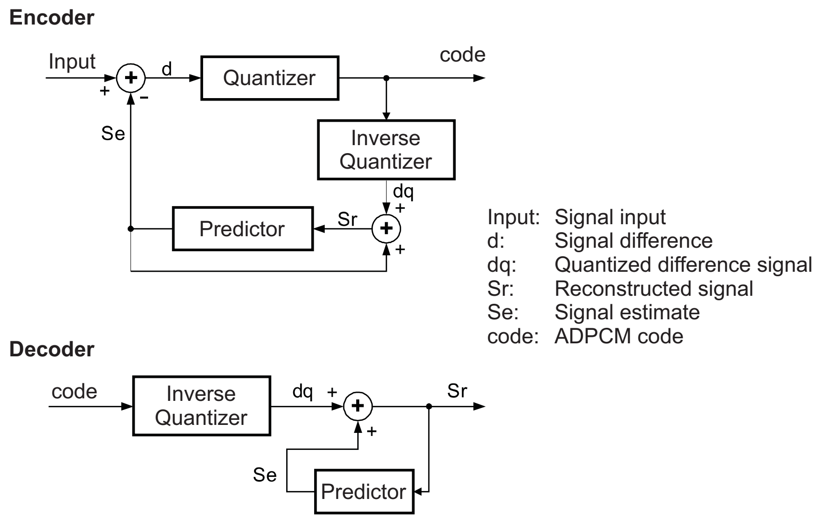
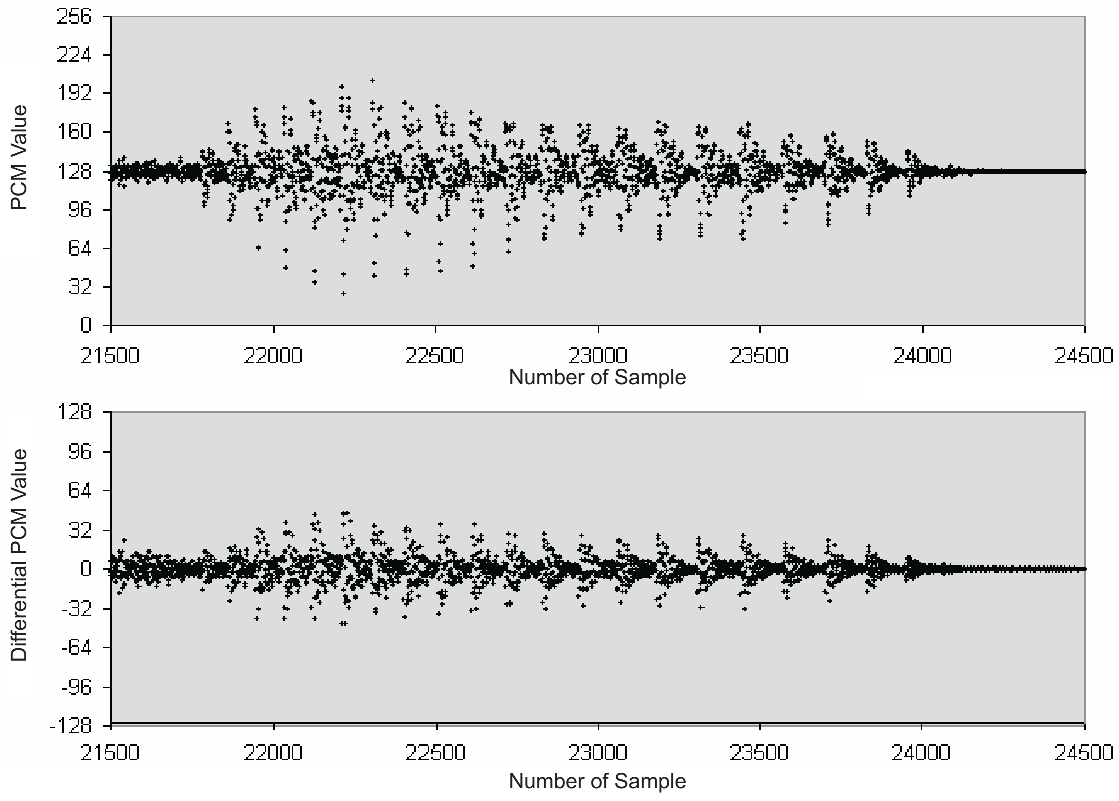


Figure 1. DPCM Encoder and Decoder Block Diagram

Figure 2 shows a small part of a recorded audio stream. Analog audio input samples (PCM values) and the differences between successive samples (DPCM values) are compared in the two diagrams.

The range of the PCM values is between 26 and 203, with a delta of 177 steps. The encoded DPCM values are within a range of –44 to 46, with a delta of 90 steps. Despite a quantizer step size of one, this DPCM encoding already shows a compression of the input data. The range of the encoded DPCM values could be further decreased by selecting a higher quantizer step size.

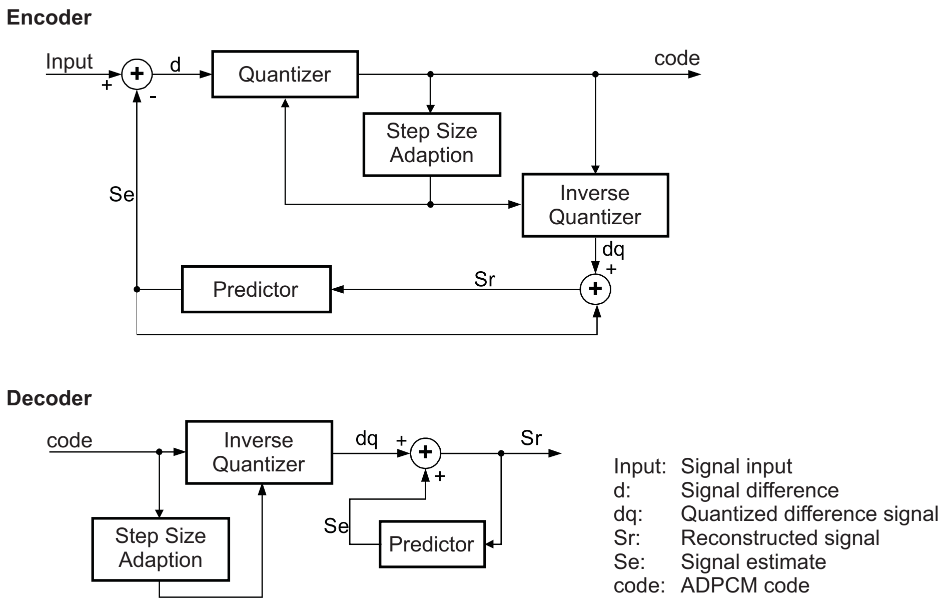


NOTE: Differential PCM Step Size = 1

Figure 2. Comparison of 8-Bit Audio Data and Difference of Successive Samples

2.2 Adaptive Differential Pulse Code Modulation (ADPCM)

ADPCM is a variant of DPCM that varies the quantization step size. Amplitude variations of speech input signals are seen between different speakers or between voiced and unvoiced segments of the speech input signal. The adaptation of the quantizer step size takes place every sample and ensures equal encoding efficiency for both low and high input signal amplitudes. Figure 3 shows the modified DPCM block diagram including the step-size adaptation.



Input:	Signal input
d:	Signal difference
dq:	Quantized difference signal
Sr:	Reconstructed signal
Se:	Signal estimate
code:	ADPCM code

Figure 3. ADPCM Encoder and Decoder Block Diagram

The ADPCM encoder calculates the signal estimate, (Se), by decoding the ADPCM code. This means that the decoder is part of an ADPCM encoder. Hence, the encoded audio data stream can only be replayed using the decoder. This means that the decoder must track the encoder.

The initial encoder and decoder signal estimate level, as well as the step-size adaptation level, must be defined before starting encoding or decoding. Otherwise, the encoded or decoded value could exceed the scale.

3 Signal Chain on Chip in MSP430 MCUs

The MSP430 microcontrollers offers a variety of on-chip peripherals. To complete a signal-chain-on-chip solution, the MSP430 MCU must have at least one analog input to the ADC and a DAC. Section 3.1 and Section 3.2 introduce solutions with two different MSP430 MCUs.

3.1 *MSP430F169 Signal-Chain-on-Chip Solution*

The MSP430F169 has an integrated 12-bit SAR ADC, a hardware multiplier module that allows efficient realization of digital filters, and an integrated 12-bit DAC module. Figure 4 shows a signal chain circuit in the MSP430F169.

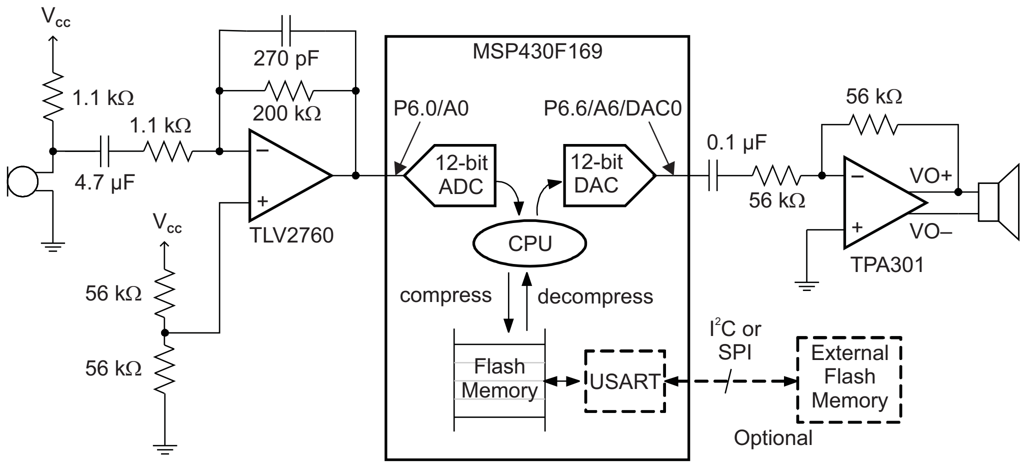


Figure 4. Signal-Chain-on-Chip Solution With MSP430F169

This configuration also works with an external serial flash memory, which could be used for storage of the audio data. It is possible to connect an external flash memory through the I^{2}C or SPI interface of the MSP430 MCU. The CPU loading could be greatly reduced by using the MSP430F169 DMA module for automatic transfers of received data to RAM.

3.2 MSP430FG4618 Signal-Chain-on-Chip Solution

Another signal-chain-on-chip solution is possible with an MSP430FG4618. While the MSP430F169 has 60KB of integrated flash memory, the MSP430FG4618 has 116KB of flash memory. Another advantage of the MSP430FG4618 is the integrated operational amplifier module. These operational amplifiers can be used to amplify the microphone input and the analog output of the DAC. Figure 5 shows the signal chain of the MSP430FG4618 solution. This is the configuration of the MSP430FG4618/F2013 Experimenter Board. This evaluation board can be used with the associated code example.

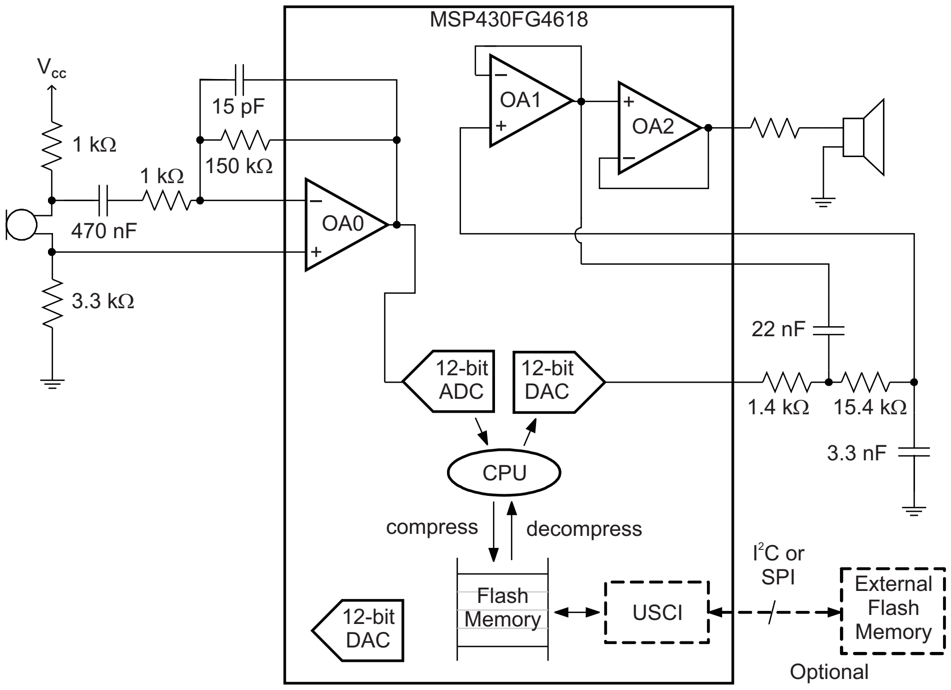


Figure 5. Signal-Chain-on-Chip Solution Using MSP430FG4618

The output signal of the microphone is small and must be amplified. The MSP430 operational amplifiers can be used in different operating modes. Using the amplifier in PGA mode allows a maximum amplification of 15, which is not enough for the microphone amplifier. Additional gain is provided using external components, and operational amplifier OA0 in Figure 5 is used in general-purpose amplifier mode. There are eight performance settings for the amplifiers to allow performance (gain-bandwidth product and slew rate) to be traded with current consumption. The example code uses high-performance mode (fast mode) for all amplifiers (OA0, OA1, and OA2).

For more information about the operational amplifier, refer to the MSP430FG4618/F2013 Experimenter Board User's Guide.

The universal serial communication interface (USCI) could be used to store the audio data in an external flash memory. The connection to the external memory could be done either through an I^{2}C bus or and SPI bus.

4 Performance on the MSP430 MCUs

The associated code includes "*.wav" file examples that show the quality of a decoded ADPCM data. Play these example files on media player software on the PC to compare them and hear the quality that is possible with an ADPCM compression algorithm. To improve the quality, increase the audio sample rate and optimize the audio sample size (resolution).

4.1 Using the Associated Code

The associated code includes two software projects. These two versions are based on the descriptions in Section 3, and both use the IMA ADPCM algorithm.

The use of the ADPCM functions is simple. First, ADPCM.h must be included in the application code. This header file declares the ADPCM functions of the ADPCM.c file. Before each sequence of recording or replaying audio data, the ADPCM_Init() function must be called. This function defines the start values for the signal estimate (Se) and the step size pointer that is used for the quantizer step size adaptation. The encoder and decoder are synchronized by these settings. Encoding is realized by calling the ADPCM_Encoder(int value) function, and playback is realized by calling ADPCM_Decoder() for each audio sample. The following code segment shows how this could be done.

```
#include "ADPCM.h"

void main(void)
{   // initialization of application software

    while(1)   // Main Loop
    {   // application software
        if (P1IN & 0x01)
           record();
        if (P1IN & 0x02)
           play();
     }
}

void record(void)
{   // initialization for recording (ADC, timer, amplifier, ...)
     ADPCM_Init();   // this is done before recording is started

     // start recording
}

void play(void)
{   // initialization for recording (DAC, timer, amplifier, ...)
     ADPCM_Init();   // this is done before playback is started

     // start playback
}
```

The following measurements of the ADPCM function execution times were made with IAR Embedded Workbench® KickStart version 3.42A. The default optimization settings were used for the measurements.

ADPCM_Encoder() function call requires between 114 and 126 cycles.

ADPCM_Decoder() function call requires between 99 and 109 cycles.

These measurements include only the compression and decompression algorithms. Algorithms to record and play require additional code.

5 References

1. MSP430x4xx Family User's Guide
2. MSP430F15x, MSP430F16x, MSP430F161x Mixed-Signal Microcontrollers
3. MSP430FG461x, MSP430CG461x Mixed-Signal Microcontrollers
4. A Low-Cost 12-Bit Speech CODEC Design Using the MSP430F13x
5. MSP430FG4618/F2013 Experimenter Board User's Guide

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from July 6, 2007 to August 24, 2018	Page
• Editorial and format changes throughout document	1
• Added the paragraph that begins "This application report and the accompanying code..." in Section 1, *Introduction*	2
• Updated Section 5, *References*	8