Embedded Systems Design 4840

MANIC

Music All Night In Columbia

Design Document

Team:

Prakash G.S

pg2132@columbia.edu

Devyani Gupta

dg2168@columbia.edu

Vijayarka Nandikonda

vn2107@columbia.edu

Contents

1. Introduction

The original goal of our project was to design an MP3 player. However, we have revised this goal towards implement an ADPCM player owing to time constraints and numerous problems faced with respect to implementation. In addition, we also endeavor to implement a PCM player for 8 and 16 bit audio formats.

What is Adaptive Differential Pulse Code Modulation?

ADPCM is an audio coding technique that is widely used throughout the telecommunications industry. It works by calculating the difference between two consecutive samples in standard pulse code modulation (PCM) and codes the error of the 'predicted' next sample increment (from the previous sample increment) to the true sample increment.
It is a lossy compression technique that achieves a compression ratio of 4:1. However, it is popular since is returns a high quality signal with very little processing power required for fast decoding.
There are primarily 2 different industry formats for ADPCM:

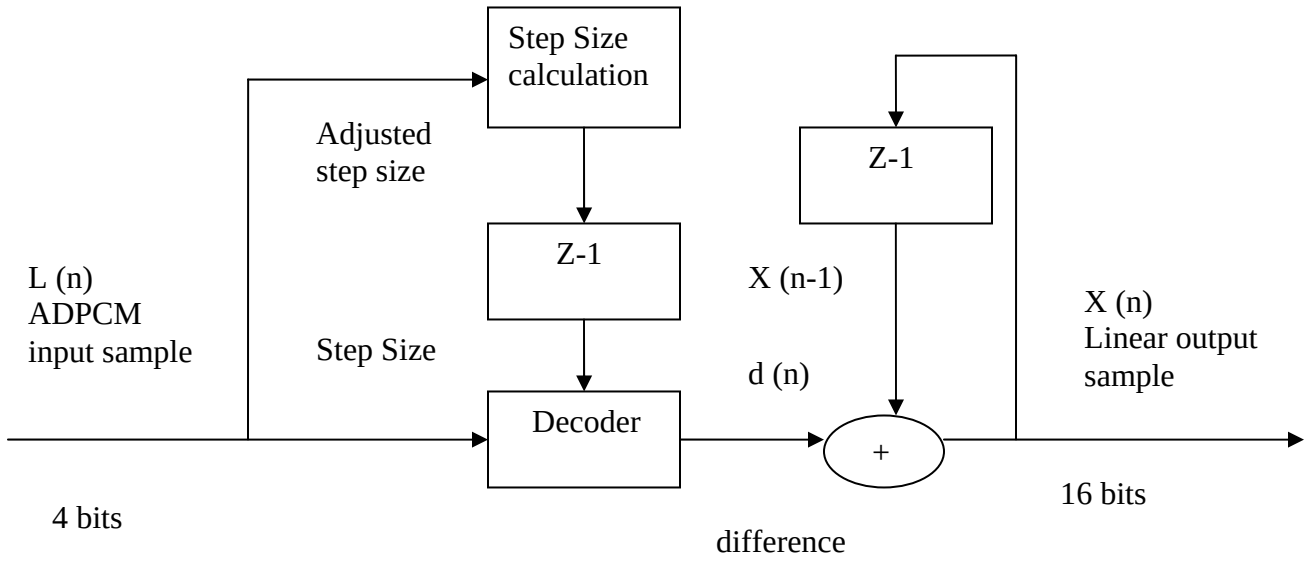
1. IMA/DVI ADPCM
2. Microsoft ADPCM

Our implementation is centered on the IMA format.

2. ADPCM Decoding

2.1 Algorithm

The diagram shows the steps involved in ADPCM decoding.



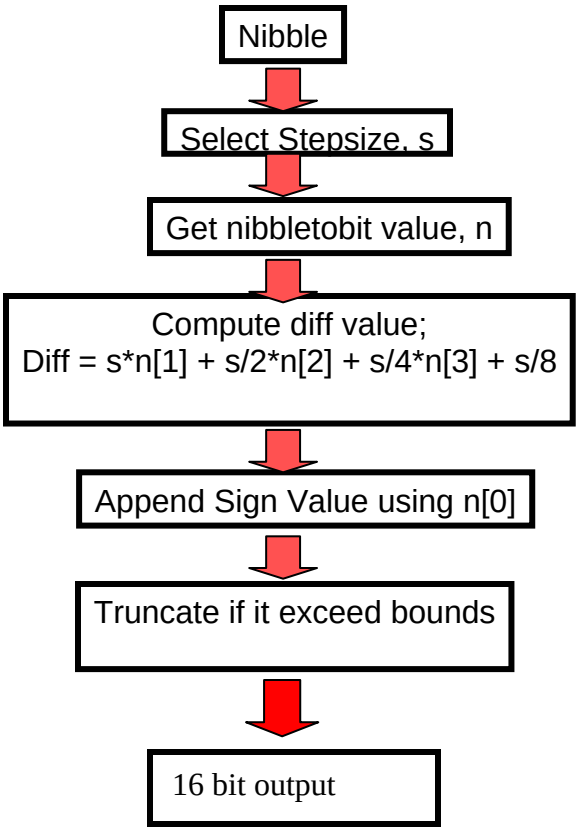
ADPCM decoding is comprised of the following steps:

Step-Size calculation:
The step-size is basically a coding scale for the ADPCM. It varies dynamically to accommodate the differences between small and large samples. The step size initially starts off at a preconfigured value. This value is then readjusted/predicted for the next sample, depending on the sample received. This process is called ***step-size adjustment***.

Decoding:
The decoding for each 4 bits then happens by using the current sample and the step-size. The difference of the decoded output and the previous sample is then taken. This difference yields a 16-bit linear PCM sample.

2.2 Implementation

The following flow chart defines our implementation. We maintain 3 arrays to store stepsize values, nibbletobit values and sign value. We also have an array to store the value of the last 128 samples.



ADPCM decoding sequence

2.3 Calculation of Step Size

For both the encoding and decoding process, the ADPCM algorithm adjusts the quantizer stepsize based on the most recent ADPCM value. The step size for the next sample, n+l, is calculated with the following equation:
ss(n+1) = ss(n) * 1.1M(L(n))

This equation can be implemented efficiently as a two-stage lookup table. First the magnitude of the ADPCM code is used as an index to look up an adjustment factor as shown in *Table 1*. Then that adjustment factor is used to move an index pointer in *Table 2*. The index pointer then points to the new step size. Values greater than 3 will increase the step size. Values less than 4 decrease the step size.

Table 1. `M(L(n)) Values

L(n)	Value M(L(n))
1111	0111 +8
1110	0110 +6
1101	0101 +4
1100	0100 +2
1011	0011 -1
1010	0010 -1
1001	0001 -1
1000	0000 -1

Table 2. Calculated Step Sizes

No.	StepSize	No.	StepSize	No.	StepSize	No.	StepSize
1	16	13	50	25	157	37	494
2	17	14	55	26	173	38	544
3	19	15	60	27	190	39	598
4	21	16	66	28	209	40	658
5	23	17	73	29	230	41	724
6	25	18	80	30	253	42	796
7	28	19	88	31	279	43	876
8	31	20	97	32	307	44	963
9	34	21	107	33	337	45	1060
10	37	22	118	34	371	46	1166
11	41	23	130	35	408	47	1282
12	45	24	143	36	449	48	1411
49	1552						

This method of adapting the scale factor with changes in the waveform is optimized for voice signals, not square waves or other non-sinusoidal waveforms.

3. Design Alternatives

ADPCM implementation can be done in

- Software (in a programming language) like C or
- In hardware (in a hardware-description language) like VHDL.

We explored both alternatives, finally settling on the software approach in C. This was due to the following reasons:

1. The computations involved in the decoding were not of a time-consuming nature. So implementing in hardware would have given no substantial benefit.

   MicroBlaze runs at a clock frequency of 50 MHz, and our input ADPCM file is of 12 kHz. So we have 50M/12k = 4000 cycles for processing each sample. This gives us ample time to implement the algorithm in software.

2. There are no floating-point computations in the procedure.

MicroBlaze cannot do floating point computations. If there were floating point computations in the process, we would have had no choice but to do it in VHDL.

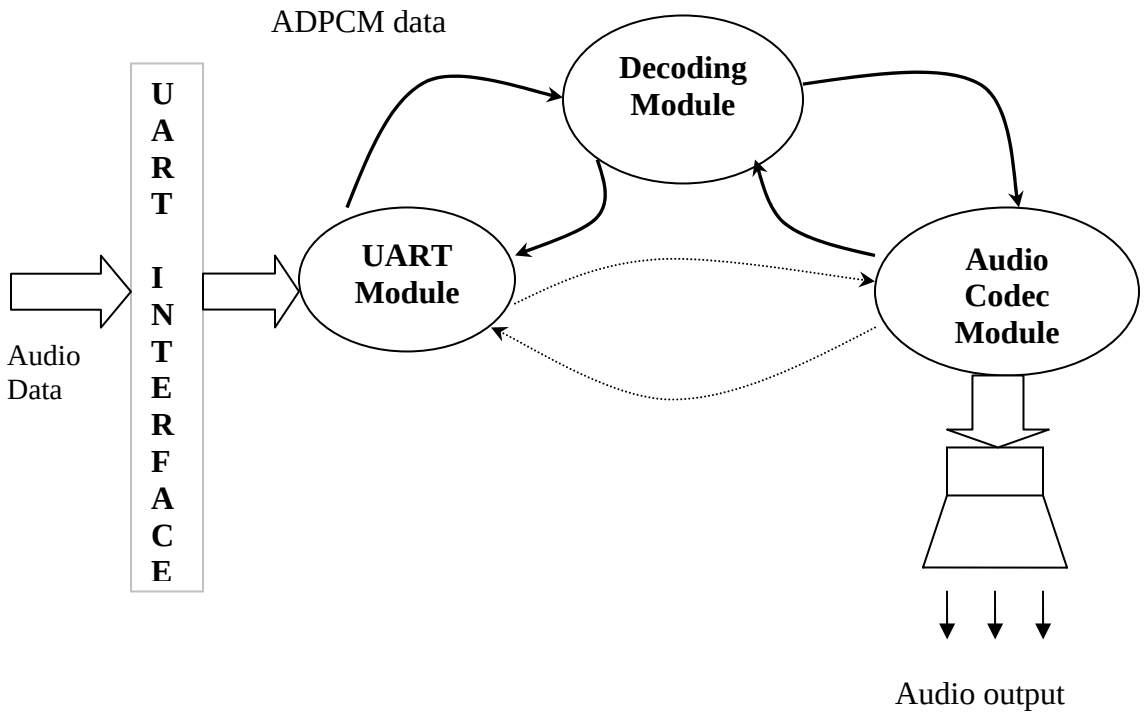
3. The procedure involves few simple multiplication operations (step-size x 1-bit) and these can be optimized in software into simple conditional statements with additions.

   Timing is the most crucial thing in high speed I/O. Code optimization reduced the time taken to process the samples, thus obeying the time constraints.

4. Final Design

Our design essentially composes of 3 modules, namely the UART module, the Audio Codec module and the Decoding module.

The diagram below shows interaction among the various modules.



UART Module:

The UART module receives data from the UART and buffers for the audio processing module. It also plays a role in flow-control, sending start and stop patterns to the UART program on the PC for alternately starting and stopping data.

Audio Processing Module:

The audio processing module processes data from the UART buffer and calls the ADPCM conversion routine to convert ADPCM data. In case of PCM data, this module converts mono to stereo and buffers it in the audio codec buffer.

Audio Codec Module:

The audio codec module streams data from the audio codec buffer into the audio codec.