Dialogic ADPCM Algorithm

Abstract

This application note describes the implementation of Adaptive Differential Pulse Code Modulation (ADPCM) as used in Dialogic Voice Processing Applications.
The following topics are covered.

- File format for voice data files.
- ADPCM encoding algorithm.
- ADPCM decoding algorithm.
- Step size determination.
- Initial and reset conditions.

00-1366-001

Warranty Disclaimer

Dialogic expressly disclaims all warranties with regard to this application note, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Dialogic does not warrant, guarantee, or make any representation regarding the use or the results of the use of this application note in terms of correctness, accuracy, or reliability. The contents of this application note are subject to change without notice. Dialogic will publish updates and revisions of this document as needed. This document supersedes all previous versions.

Limitation of Liability

You agree that Dialogic shall not be liable to you under this agreement for any damages, including without limitation any lost profits or lost savings, or any consequential, incidental, or punitive damages arising out of the use or inability to use this application note and related documents, or for any claim by another party. You agree to hold Dialogic harmless for all claims and damages arising from any third party as a result of their use or inability to use any product that you develop based on this application note and the products and/or services documented herein.

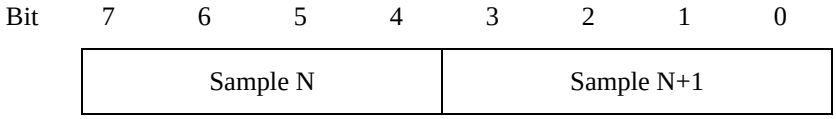
VOX file format specification:

VOX files are flat binary files containing digitized voice data samples. Each byte contains two samples. There is a direct relationship between any positional offset within the file and time, expressed in the following formula:

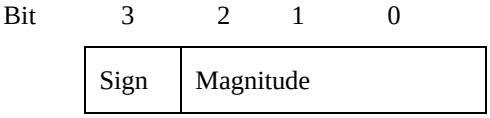
```
T(i) = 2i * I/SR
```

where: T(i) is the time offset in seconds from the beginning of the file of byte number "i" within the file. SR is the sampling rate in samples per second.

The encoding within each byte is as follows:



The encoding within each sample is Adaptive Differential Pulse Code Modulation (ADPCM). This is a differential coding scheme in which each sample approximates the difference between the present input value and the previous one. The weighting of the magnitude portion of the difference is adaptive (non-linear). That is, it can change after each sample.



Sign: Positive (0) or negative (1) sample.

Magnitude: Change (0 to 7) from previous sample.

ADPCM Encoding

Figure 1 shows a block diagram of the ADPCM encoding process. A linear input sample X(n) is compared to the previous estimate of that input X(n-l). The difference, d(n), along with the present step size, ss(n), are presented to the encoder logic. This logic, described below, produces an ADPCM output sample. This output sample is also used to update the step size calculation ss(n+l), and is presented to the decoder to compute the linear estimate of the input sample.

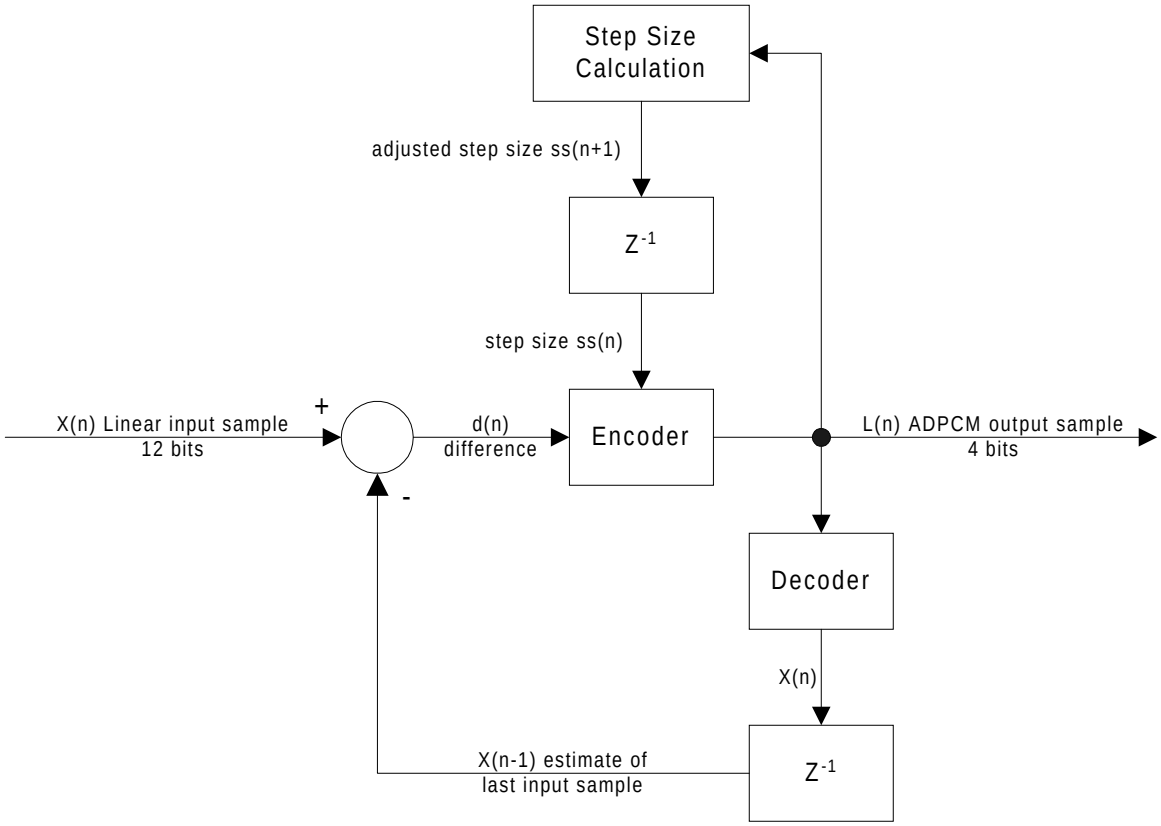


Figure 1

The encoder accepts the differential value, d(n), from the comparator and the step size, and calculates a 4-bit ADPCM code. The following is a representation of this calculation in pseudocode.

```
let B3 = B2 = B1 = B0 = 0
if (d(n) < 0)
    then B3 = 1
d(n) = ABS(d(n))
if (d(n) >= ss(n))
    then B2 = 1 and d(n) = d(n) - ss(n)
if (d(n) >= ss(n) / 2)
    then B1 = 1 and d(n) = d(n) - ss(n) / 2
if (d(n) >= ss(n) / 4)
    then B0 = 1
L(n) = (10002 * B3) + (1002 * B2) + (102 * B1) + B0
```

Note: For the calculation of ss(n), see "Calculation of Step Size."

ADPCM Decoding

Figure 2 shows a block diagram of the ADPCM decoding process. An ADPCM sample is presented to the decoder. The decoder computes the difference between the previous linear output estimate and the anticipated one. This difference is added to the previous estimate to produce the linear output estimate. The input ADPCM sample is also presented to the step size calculator to compute the step size estimate.

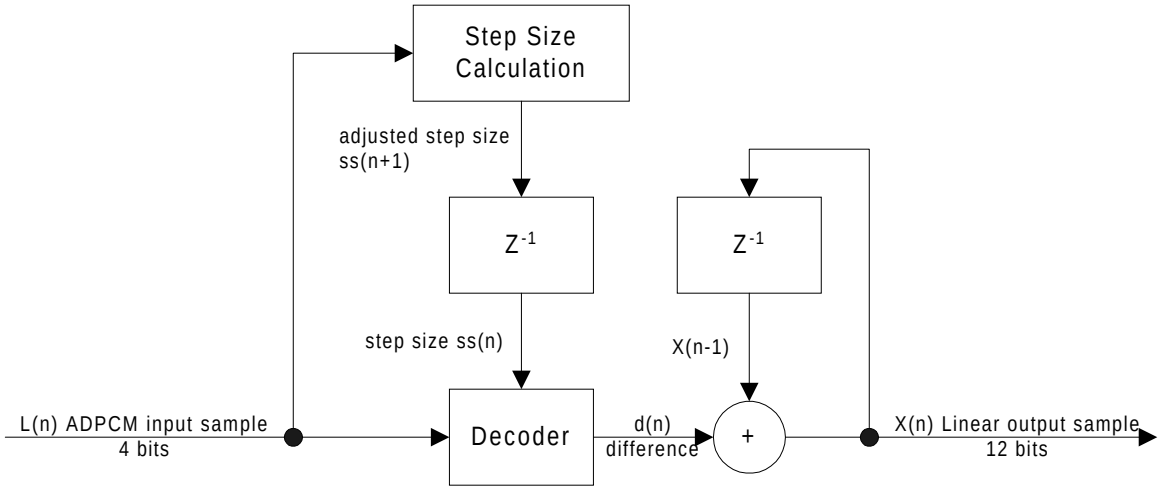


Figure 2

The decoder accepts ADPCM code values, L(n), and step size values. It calculates a reproduced differential value, and accumulates an estimated waveform value, X. Here is a pseudocode algorithm:

```
d(n) = (ss(n)*B2)+(ss(n)/2*B1)+(ss(n)/4*B0)+(ss(n)/8)
if (B3 = 1)
    then d(n) = d(n) * (-1)
X(n) = X(n-1) + d(n)
```

Note: For the calculation of ss(n), see "Calculation of Step Size."

Calculation of Step Size

For both the encoding and decoding process, the ADPCM algorithm adjusts the quantizer step size 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 or 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.	Step Size	No.	Step Size	No.	Step Size	No.	Step Size
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.

Initial Conditions

When the ADPCM algorithm is reset, the step size ss(n) is set to the minimum value (16) and the estimated waveform value X is set to zero (half scale). Playback of 48 samples (24 bytes) of plus and minus zero (1000_2 and 0000_2) will reset the algorithm. Twenty-four bytes of 08 Hex or 80 Hex will satisfy this requirement. It is necessary to alternate positive and negative zero values because the encoding formula always adds 1/8 of the quantization size. If all values were positive or negative, a DC component would be added that would create a false reference level.