Mercury Computer Systems, Inc.

CVSD Decoder Component

FM3TR Waveform Reference Implementation

SDR Forum Contract

March 23, 2007

Revision 1.0

Table of Contents

1 Component Name

CVSD Decoder

2 Component Processing Summary

The continuously variable slope delta (CVSD) algorithm is a lossy audio encoding scheme that uses one bit per sample to compress the amount of data transmitted over a digital channel. Based upon delta modulation, CVSD continuously changes its slope to track to the input signal. If the reference value (CVSD output) is greater than the input value, a binary zero is recorded, and delta is subtracted from the reference; otherwise a binary one is recorded and delta is added. The past *N* bits are stored in a register. If all *N* bits are either zeros or ones, the value for delta is doubled; otherwise delta is halved. For this implementation, *N* was chosen to be three. Usually the value of delta has practical restrictions. The decoder simply reverses this process based upon the incoming bit stream. The resulting decoded waveform is usually low-pass filtered to remove high-frequency noise due to the lossy encoding process. This results in greater audio quality than conventional fixed-delta encoders, particularly at higher sample rates.

3 Where used

The CVSD decoder component is used in all encoded audio waveforms, located before the AudioPlayback component.

4 Data Input and Output Ports

The CVSD decoder has one uses and one provides data ports. The input data port (CVSDDecoderIn) accepts a sequence of signed octets (one byte for each encoded “bit”) for decoding. After decoding, the component pushes a sequence of signed short integers to the output port (CVSDDecoderOut). The output sequence is the same length as the input.

5 Control Interfaces

The CVSD decoder inherits the control interfaces from CF::Resource.

6 Component SCA Properties

Aside from `DLL_ENTRY_POINT`, the CVSD decoder contains no additional properties.

7 Component Attributes/Key Variables

Below is a list of several key variables to the CVSD encoding and decoding algorithms with a brief description of their purpose.

`DA_MAX`	Maximum level for the “digital-to-analog” converter.

`AD_MAX`	Maximum level for the “analog-to-digital” converter.
`CVSD_STEP_MIN`	Minimum step size for CVSD encoder/decoder delta value.
`CVSD_STEP_MAX`	Maximum step size for CVSD encoder/decoder delta value.
`CVSD_DELTA_TC`	Low-pass filter time constant for delta (???)
`CVSD_REF_TC`	Low-pass filter (integrator) time constant for input reference signal.
`m_coCoef`	Low-pass audio filter coefficients
`FILTER_LENGTH`	Length of low-pass audio filter.
`m_iOldbits`	CVSD codec state for last *N* bits observed

8 Processing Details

The CVSD audio decoder algorithm effectively executes as the reverse of the encoder; the received bits are used to adjust the reference level and delta value. The encoding/decoding process adds a significant amount of high-frequency out-of-band noise, most of which is removed with a low-pass audio filter.

8.1 Method: Decode()

The *Decode()* method uses the received bits to adjust both the output level (stored in `m_vfCVSDData`) and delta value. The last *N* (3) received bits are stored in a variable and the value of delta is adjusted accordingly. If the past *N* bits are either all zeros or ones, the value of delta is increased asymptotically towards `CVSD_STEP_MAX`, otherwise, its values is decreased asymptotically towards `CVSD_STEP_MIN`. The output value is adjusted by delta; a binary ‘1’ subtracts delta from the current reference while a binary ‘0’ adds. The resulting sample is stored in an output buffer (`m_vfCVSDData`).

8.2 Method: LowPassFilter()

The encoding/decoding process adds a significant amount of high-frequency out-of-band noise, most of which is removed with a low-pass audio filter. This implementation uses a 51-tap finite impulse response filter that removes a significant amount of energy that is outside of the human speech frequency band to improve audio clarity. The filter taps are stored in the array `m_coCoeff` and are convolved with the output buffer in the *LowPassFilter()* method before pushing to the AudioPlayback component.