SPIRIT DSP
Embedded Voice Experience

SPIRIT Loudness control User's Guide

Version 1.0

October, 2010

Copyright Information

Read this first

The document describes Loudness control software interface, contains flow-charts and testing procedures description. Document contains examples of the algorithm integration and usage.

To read this document you are not required to have any special knowledge. However, prior experience with C-programming is desirable.

Loudness control is an algorithm for a distortion compensation which is caused by linear signal scaling (amplification).

The document includes Loudness control algorithm overview, recommendations on the algorithm applications, description of the interface and testing procedure.

Contents

1. Introduction

The purpose of this document is to describe API, integration process and test procedures of Loudness control software.

1.1. Product overview

The loudness control software modifies signal before amplification in order to preserve perceptual signal loudness. Modification is performed by 2nd order low-pass shelving filter which parameters depended on chosen amplification gain. The algorithm operates independently on the "frame by frame" basis. The frame length is a user-defined parameter.

Specification of the loudness control software product is presented in Table **Table 1**.

Algorithm	Serial connection of shelving (1st order) and peaking (2nd order) filters.
Channels number	1,2
Frame size	Arbitrary
Algorithmic delay	No extra delay, except framing delay.
Sampling rate	8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100, 48000 Hz
Gain range	–36 to +36 dB
Input and output signal format	Interleaved, linear 16-bit PCM, little-endian format.

Table 1 Specifications

For more information about other SPIRIT audio processing technologies visit www.spiritDSP.com.

SPIRIT reserves the right to make changes to its products to improve the products' technical characteristics without any notice. Customers are advised to obtain the latest version of relevant information to verify that the data is up-to-date before placing the orders.

SPIRIT warrants performance of its products to current specifications in accordance with SPIRIT's standard warranty. Testing and other quality control techniques are utilized to the extent deemed necessary to support this warranty.

1.2. Related products

The loudness control can be efficiently used in conjunction with other products of SPIRIT:

- MPEG-4 AAC Decoder
- MPEG-1 Layer 3 Codec
- Dynamic Range Control
- Sample Rate Converter
- Automatic Gain Control

1.3. Document Overview

This document is organized as follows.

Section 2 explains a functional description of the software.

Section 3 describes software integration flow, descriptions of structures and interface functions.

2. Functional Description

This section describes loudness control algorithm and provides several recommendations on its usage.

2.1. Algorithm overview

In fact, human does not perceive two sounds with equal magnitude as equal loudness signal. For example pure tone of 3 kHz perceives much louder that 1 kHz tone. The loudness level can be approximately calculated for each particular frequency[†], the unit of measuring is called *phon* (Ph). It is followed from definition that two pure tones with equal Ph level perceive as equal loudness signals. Generally, phon is not only a function of frequency, but also it depended on magnitude (SPL).

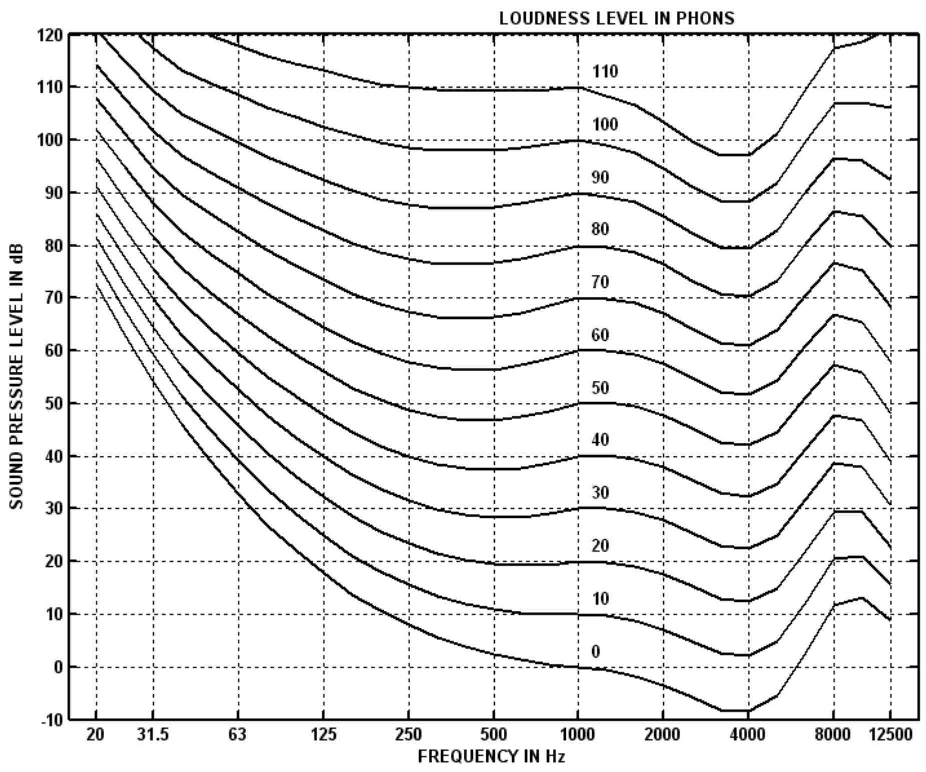

Figure 1. Fletcher-Munson equal loudness contour

Now, consider the equal loudness contour dependency on SPL. More precisely, let references SPL is 40 dB and test signals are two pure tones *f0* and *f1* with magnitude level corresponding to 40 Ph level; listener can't determine the difference in their loudness. Amplify test vectors with a given gain. They becomes at a different loudness level, it means they perceived as different volume signals. This difference can be expressed in phons:

$$\Delta Ph_{f0,f1} = Ph(A_{f1}\_dB + gain\_dB, f_1) - Ph(A_{f0}\_dB + gain\_dB, f_0) \neq 0$$

To make f_1 with amplitude $A_{f1}\_dB + gain\_dB$ as loud as f_0 with amplitude $A_{f0}\_dB + gain\_dB$ we must amplify by:

$$\Delta\ B = GetEqualLoudnessMagnitude(Ph(A_{f0}\_dB + gain\_dB, f_0), f_1) - A_{f1}\_dB + gain\_dB)$$

[†] See ISO 226:1987 (E), standardized frequency range [20, 12500] Hz

Assumes f_0 is fixed at 1kHz, the correction which should be introduced after linear scaling to move signal to a $Ph(A_{f0}\_dB + gain\_dB)$ level is shown on Figure 2.

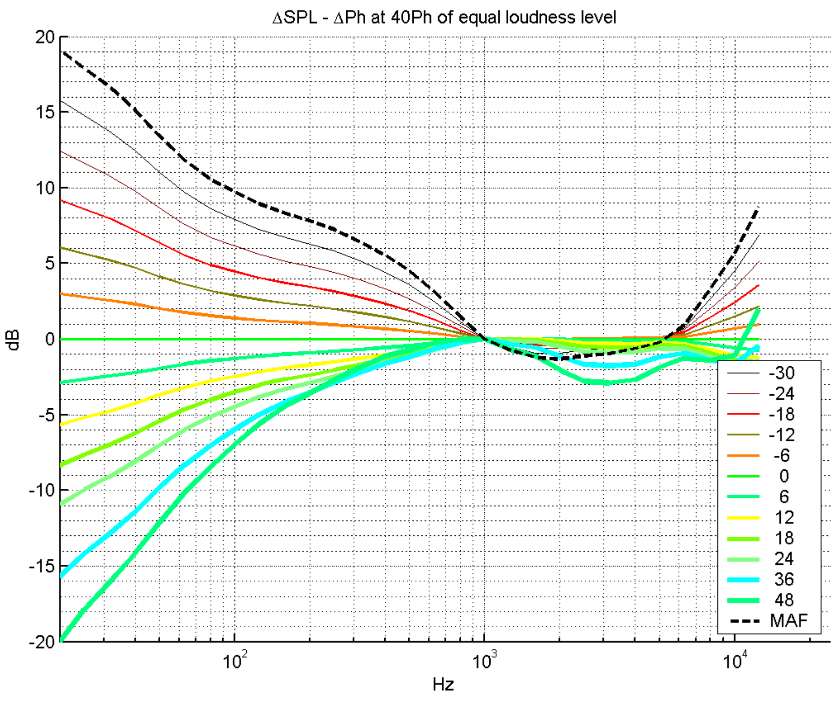

Figure 2. Loudness level correction curves with 6 Ph step

For a practical use the set of loudness curves is approximated by a number of IIR filters. One 2nd order shelf-filter for each gain in range --36:36:6 dB is used. Intermediate curves approximated as a linear combination of neighbor filters.

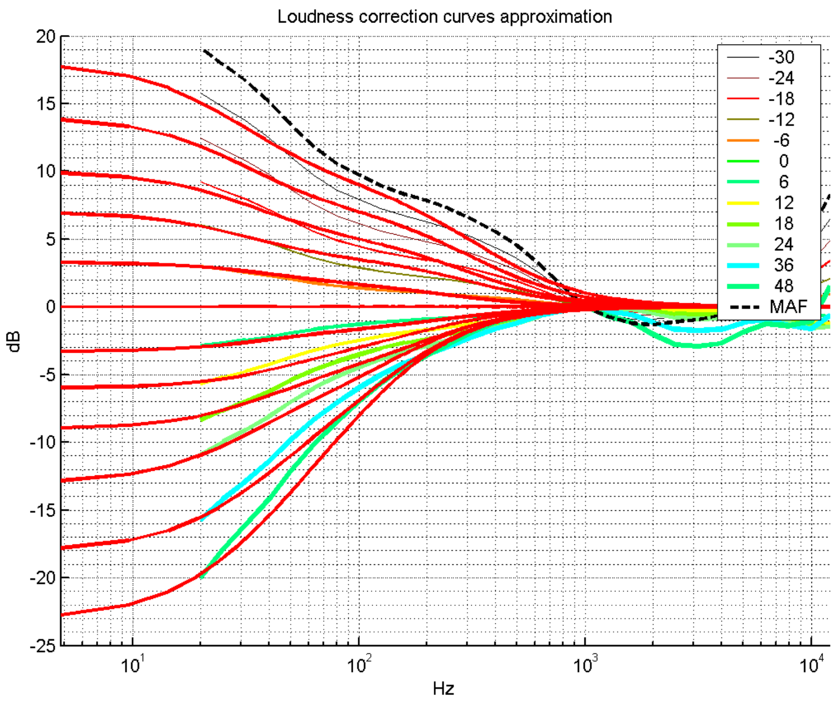

Figure 3. Loudness Correction Curves Approximation

The algorithm processing scheme is depicted below.

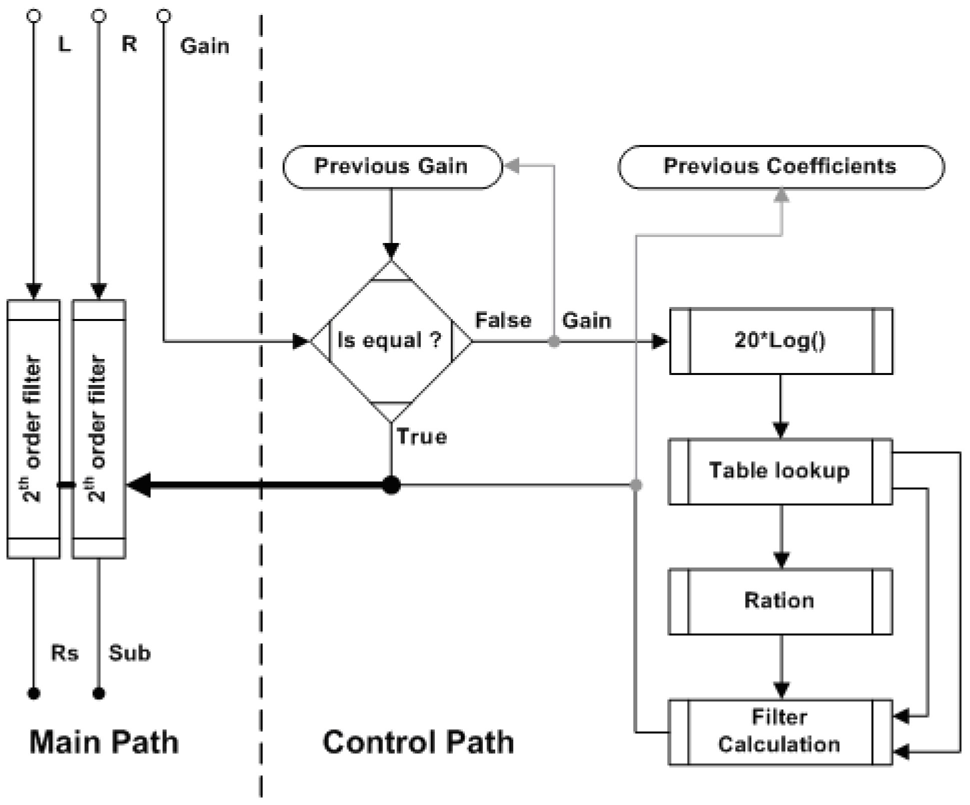

Figure 4. Block diagram of the loudness control algorithm

2.2. Features and recommendations on loudness control usage

The loudness control modifies signal loudness according to output gain value that is why this module should be placed right before the amplifier. Also, default amplification must be defined (system_unity_gain). If amplifier gain is equal to *system_unity_gain,* than loudness control module consider output level as 'normal' or 'reference' and does not modify signal.

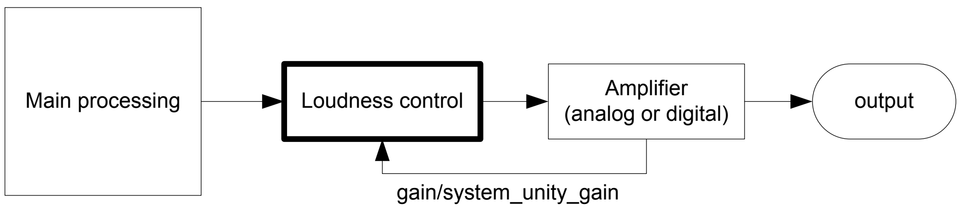

Figure 5. Loudness control placement

By default, acceptable gain range is [-36, +36] dB. This range may be too wide for most of devices. One can reduce this range just feeding scaled gain value to loudness control unit. For example, the multiplication of input gain by 2 constricts acceptable gain range to [-30, 30] dB interval.

3. API description

This section describes software implementation and its interface.

3.1. Integration flow

In order to integrate the algorithm into a user's framework, the user should:

- Initialize the software using *SpiritLdCtrl_Init()* function.
- Set run-time parameters using *SpiritLdCtrl_SetPrms()* function.
- Feed each new block of input data to *SpiritLdCtrl_Apply ()* function.

3.2. Predefined constants

SPIRIT_LDCTRL_PERSIST_SIZE_IN_BYTES	Persistent memory size in bytes.
SPIRIT_LDCTRL_MAX_CH	Maximal number of bands.
SPIRIT_LDCTRL_GAIN_Q_BITS	Maximal number of bands.
SPIRIT_LDCTRL_GAIN_MAX	Maximal gain value
SPIRIT_LDCTRL_GAIN_MIN	Minimal gain value

3.3. TSpiritLdCtrl

Syntax

```
typedef void TSpiritLdCtrl;
```

Remarks

Alias of the persistent memory buffer. Only a pointer on this type has sense; all API functions expect it points to a 4 bytes aligned memory region.

3.4. TSpiritLdCtrl_Prms

Syntax

```
typedef struct {
    int gainQ8;
} TSpiritLdCtrl_Prms;
```

Members

gainQ8	Amplifier gain in Q8 format in range *[SPIRIT_LDCTRL_GAIN_MIN, SPIRIT_LDCTRL_GAIN_MAX]*

Remarks

This structure carries run-time software parameters. Note that unity gain value in Q8 format is equal to *(1<< SPIRIT_LDCTRL_GAIN_Q_BITS).*

3.5. SpiritLdCtrl_Init ()

Syntax

```
  int SpiritLdCtrl_Init (
            const TSpiritLdCtrl *ld,
            unsigned long sampleRateHz
  );
```

Parameters

Id	Pointer to persistent memory buffer.
sampleRateHz	Sampling rate in Hz. Must be positive integer number.

Return value

Error code.

Remarks

This function initializes loudness control persistent memory buffer and resets filters delay.

3.6. SpiritLdCtrl_Reset ()

Syntax

```
  int SpiritLdCtrl_Reset (
            const TSpiritLdCtrl *ld
  );
```

Parameters

Id	Pointer to initialized persistent memory buffer.

Return value

Error code.

Remarks

This function resets loudness control state:

- Set filter delay to zero

- Setup unity gain value (setup filter corresponding to unity gain)

3.7. SpiritLdCtrl_SetPrms ()

Syntax

```
int SpiritLdCtrl_SetPrms(
          TSpiritLdCtrl *ld,
          const TSpiritLdCtrl_Prms *prms
);
```

Parameters

ld	Pointer to initialized persistent memory buffer.
prms	Initialization parameters.

Return value

Error code.

Remarks

This function can be invoked in run-time. It is recommended to receive current parameters first and then setup new. The proper call sequence is the following:

```
TSpiritLdCtrl_Prms prms;

SpiritLdCtrl_SetPrms(obj, &prms);  // get current parameters

prms.xxx = xxx;                   // modify
prms.yyy = yyy;

SpiritLdCtrl_GetPrms(obj, &prms);  // set new parameters
```

Note, no error returned in gain value falls out of the range *[SPIRIT_LDCTRL_GAIN_MIN, SPIRIT_LDCTRL_GAIN_MAX]*. Instead, gain is limited to fit the range.

3.8. SpiritLdCtrl_GetPrms ()

Syntax

```
int SpiritLdCtrl_GetPrms (
          const TSpiritLdCtrl *ld,
          TSpiritLdCtrl_Prms *prms
);
```

Parameters

ld	Pointer to initialized persistent memory buffer.
prms	Loudness control parameters storage.

Return value

Error code.

Remarks

This function returns actual loudness control parameters.

3.9. SpiritLdCtrl_Apply ()

Syntax

```
int SpiritLdCtrl_Apply (
          TSpiritLdCtrl *ld,
          int nChannels,
          short *pcm,
          int nSamplesPerCh,
          void *scratch,

);
```

Parameters

ld	Pointer to initialized persistent memory buffer.
nChannels	Channel number in range [1, SPIRIT_LDCTRL_MAX_CH].
pcm	Input/output PCM buffer.
nSamplesPerCh	Number of input frame samples per channel.
scratch	Pointer to scratch memory buffer.

Return value

Error code.

Remarks

This function applies shelving filter to input PCM data based on initialization parameters.

Note, input gain only defines particular low-pass filter, but not applied to a signal. Gain transition (filter switching) performed softly. Full transition executed with a speed of 40dB/sec. For example, gain change from 0 db to 20 dB (gain x1 to x10) would take .5 second. The software requires two times more computational power during gain transition period because of two filters operate simultaneously.

3.10. Error codes

Enumerated name	Description
`SPIRIT_LDCTRL_S_OK`	Operation completed successfully.
`SPIRIT_LDCTRL_E_INVALIDARG`	Bad arguments for a function.
`SPIRIT_LDCTRL_E_SAMPLERATE`	Bad sample rate.

Enumerated name	Description
SPIRIT_LDCTRL_E_CH	Bad number of channels.
SPIRIT_LDCTRL_E_BADALIGN_PERSIST	Bad persistent memory alignment.
SPIRIT_LDCTRL_E_BADALIGN_SCRATCH	Bad scratch memory alignment.

3.11. Application example

This is an example of loudness control usage; it can be linked with loudness control library and run.

```
#include <stdio.h>
#include "spiritLdCtrl.h"

static long ld[SPIRIT_LDCTRL_PERSIST_SIZE_IN_BYTES/4];

#define PCM_BUFFER_SIZE_IN_SAMPLES 256
static short bufInOut[PCM_BUFFER_SIZE_IN_SAMPLES];

void Process_SingleFile (
    const   char    *szInputName,
    const   char    *szOutputName
)
{
    FILE *pFileIn = NULL, *pFileOut = NULL;
    unsigned long  lHz = 48000;
    unsigned int   nCh = 2;

    pFileIn  = fopen(szInputName,  "rb");
    pFileOut = fopen(szOutputName, "wb");

    // Initialize
    SpiritLdCtrl_Init(ld, lHz);

    // Setup parameters
    {
        TSpiritLdCtrl_Prms prms;
        SpiritLdCtrl_GetPrms(ld, &prms);
        prms.gainQ8 = 20<<8;
        SpiritLdCtrl_SetPrms(ld, &prms);
    }

    while(1) {

        int nSamples = fread(bufInOut, sizeof(short), PCM_BUFFER_SIZE_IN_SAMPLES, pFileIn);
        int nSamplesPerCh = nSamples >> (nCh==2 ?1:0);

        if(nSamplesPerCh == 0) {
            break;
        }

        // Run loudness control
        SpiritLdCtrl_Apply(ld, nCh, bufInOut, nSamplesPerCh);

        // Write output data
        fwrite(bufInOut, sizeof(short), nSamplesPerCh*nCh, pFileOut);
    }

    fclose(pFileIn);
    fclose(pFileOut);
}
```