VS1053B / VS1063A EQUALIZER

VSMPG “VLSI Solution Audio Decoder”

Project Code:
Project Name: VSMPG

All information in this document is provided as-is without warranty. Features are subject to change without notice.

Revision History			
Rev.	**Date**	**Author**	**Description**
1.11	2013-12-19	HH	Added Chapter 7, Frequency Responses. Software is the same as in v1.10.
1.10	2013-10-25	HH	Added 7-band equalizer designer, Chapter 4.
1.01	2013-10-24	HH	Corrected error in Example 2 in Chapter 3.1.
1.00	2013-10-23	HH	Initial release.

Contents

List of Figures

1 Introduction

This document is an instruction manual on how to use the VS1053b / VS1063a equalizer.

Chapter 2 describes the application. Chapter 3 show how to set it up and run it.

The filters can be used in two different ways: either by using a 7-Band Equalizer Designer as shown in Chapter 4, or by designing filters manually as shown in Chapter 5.

Support functions needed to use the application are shown in Chapter 6.

Chapter 7 presents the filter frequency responses.

Chapter 8 tells how to load plugin code to a VS10XX chip.

Finally, Chapter 9 contains VLSI Solution's contact information.

2 The VS1053b / VS1063a Equalizer

The *VS1053b / VS1063a Equalizer* application allows the VS10x3 IC to act as a multi-channel equalizer.

When running the application, other functions of VS10x3 is not available.

Some key features of the equalizer are:

- Runs at 48 kHz.
- Audio delay approximately 3 ms.
- Upto 22 / 28 equalization filters available for VS1053b and VS1063a, respectively. Note: A stereo channel uses two filters.
- Alternatively, a 7-Band Equalizer Designer can be used.
- Multi-IC binary, compatible with both VS1053b and VS1063a.
- The application sets VS10x3 to its maximum in-spec clock rate.

The VS1053b / VS1063a Equalizer is available as an application, downloadable at *http://www.vlsi.fi/en/support/software/vs10xxapplications.html* .

2.1 Limitations and Requirements

- For the sample rate to be exactly 48 kHz, the crystal needs to be 12.288 MHz. With other clocks, the internal sample rate is $f_s = \frac{48 \times c}{12.288}$ where c is the crystal frequency in MHz.

3 Running VS1053b / VS1063a Equalizer

There are several registers that may be read by the user, presented in Chapter 3.1.

The equalizer filters can be set in two ways.

Chapter 4 shows how to set the equalized using the automatic 7-Band Equalizer Designer. The designer uses upto 26 filter channels to create an equalizer that would have the best available audio quality and interband rejection. This is the recommended method for users without a deep signal processing or equalizer background.

Chapter 5 explains how to set individual filters manually. Manual setting is only recommended for users that want to realize a filter system that cannot be created with the 7-Band Equalizer Designer.

3.1 VS1053b / VS1063a Equalizer Read Registers

After starting the equalizer, the following registers can be read by the user:

Register	Bits	Description
SCI_AICTRL0	15:8 7:0	Always 0. Current internal buffer fill state (debug).
SCI_AICTRL1	15:1 0	Always 0. 0 = Enough CPU, 1 = internal overflow. Register may be cleared by the user.
SCI_AICTRL1	15:0	Not used.
SCI_AICTRL3	15:8 7:0	Maximum number of filters available with this IC. Current number of filters used.

4 Using the 7-Band Equalizer Designer

When the 7-Band Equalizer Designer is used, the memory locations that the application needs to write to are as follows:

Function	X Address
64 Hz amplification in dB*256, signed two's complement	0x1800
160 Hz amplification in dB*256, signed two's complement	0x1801
400 Hz amplification in dB*256, signed two's complement	0x1802
Set to 0	0x1803
1000 Hz amplification in dB*256, signed two's complement	0x1804
2500 Hz amplification in dB*256, signed two's complement	0x1805
6250 Hz amplification in dB*256, signed two's complement	0x1806
Set to 0	0x1807
15600 Hz amplification in dB*256, signed two's complement	0x1808
Set to 0	0x1809
Set to 0	0x180A
Set to 8 (activates designer)	0x180B

The user first sets the requested amplifications (recommended range -36...+18 dB), then activates the designer. The designer will create upto 26 filters to best implement the 7-band equalizer as well as possible.

When the 7-Band Equalizer Designer is used, any previous filters designed manually (Chapter 5) will be removed. The filters the designer has created will be located in memory locations 19...31.

Example:
To set a 7-band filter which greatly emphasizes bass and treble, the following code could be used:

```
//                      64   160   400  1000  2500  6250 15600 Hz
u_int16 dBTab[7] = {12.0,  5.0,  1.0, -1.5, -3.0,  2.0,  6.0};

Set7Chan(dBTab);
```

The implementation for Set7Chan() is provided in Chapter 6.1.

5 Setting Filters Manually

There are 32 filter memory locations available to the user. When the application is started, all memory location are cleared. The memory locations are located in data memory as follows:

Memory Location	X Address
0	0x1800
1	0x1804
2	0x1808
3	0x180c
...	...
31	0x187c

Each filter memory location contains the following fields which may be set by the user:

Offset	Bits	Description
0	15:0	Filter center frequency (Hz). Recommended range is 20...20000 Hz.
1	15:0	Filter amplification (dB * 256, signed two's complement). Recommended range is -36...+18 dB.
2	15:0	Filter Q factor (Q * 256, unsigned). Recommended range is 0.1...10.
3 3 3 3 3	15:4 3 2 1 0	Unused Reserved, set to 0. 1 = Apply filter to left channel. 1 = Apply filter to right channel. 1 = Update filter. This needs to be clear before new values can be written to, and must be set by the user. See examples below.

Example 1:
To set a 12.5 dB lifting filter at 2000 Hz with a Q factor of 2 to both channels at filter memory location 3, do the following:

```
//        mem    Hz    dB    Q  l  r
SetFilter(  3, 2000, 12.5, 2.0, 1, 1);
```

Example 2:
To turn the previous filter off, do the following:

```
//        mem  Hz   dB    Q  l  r
SetFilter(  3,  0, 0.0, 0.0, 0, 0); // Only last two zeroes are significant
```

The implementation for SetFilter() is provided in Chapter 6.2.

6 Support Functions

This chapter provides implementations for functions used in this document.

6.1 Function Set7Chan() Implementation

```
#define SCI_WRAMADDR 7
#define SCI_WRAM 6

// Parameter is a 7-long vector of dB values for
// 64, 160, 400, 1000, 2500, 6250, and 15600 Hz
void Set7Chan(const double *dB) {
  // Check that a previous filter update is not going on,
  // and wait if necessary.
  do {
    WriteSci(SCI_WRAMADDR, 0x180B);
  } while (ReadSci(SCI_WRAM) & 8);

  // Set new equalizer values
  WriteSci(SCI_WRAMADDR, 0x1800);
  WriteSci(SCI_WRAM, (s_int16)(dB[0]*256.0));
  WriteSci(SCI_WRAM, (s_int16)(dB[1]*256.0));
  WriteSci(SCI_WRAM, (s_int16)(dB[2]*256.0));
  WriteSci(SCI_WRAM, 0);
  WriteSci(SCI_WRAM, (s_int16)(dB[3]*256.0));
  WriteSci(SCI_WRAM, (s_int16)(dB[4]*256.0));
  WriteSci(SCI_WRAM, (s_int16)(dB[5]*256.0));
  WriteSci(SCI_WRAM, 0);
  WriteSci(SCI_WRAM, (s_int16)(dB[6]*256.0));
  WriteSci(SCI_WRAM, 0);
  WriteSci(SCI_WRAM, 0);
  WriteSci(SCI_WRAM, 8);  // Activate filter designer
}
```

6.2 Function SetFilter() Implementation

```
#define SCI_WRAMADDR 7
#define SCI_WRAM 6

// Sets one manual filter.
void SetFilter(u_int16 memoryLocation, u_int16 freqHz, double dB, double q,
               s_int16 left, s_int16 right) {
  // Check that a previous filter update is not going on,
  // and wait if necessary.
  do {
    WriteSci(SCI_WRAMADDR, 0x1800 + memoryLocation*4 + 3);
  } while (ReadSci(SCI_WRAM) & 1);

  // Do our filter update
  WriteSci(SCI_WRAMADDR, 0x1800 + memoryLocation*4);
  WriteSci(SCI_WRAM, freqHz);
  WriteSci(SCI_WRAM, (s_int16)(dB*256.0));
  WriteSci(SCI_WRAM, (u_int16)( q*256.0));
  WriteSci(SCI_WRAM, (left?4:0)|(right?2:0)|1); // Left, right, update
}
```

6.3 Function WriteSci() Implementation

```
// This pseudo-code shows how to write to VS10xx's SCI
// (serial command interface) SPI bus. You need to write
// your own implementation for your own microcontroller.

void WriteSci(u_int16 addr, u_int16 data) {
  AssertSpiChipSelect();
  WriteSpiByte(2);         // 2 = write, 3 = read
  WriteSpiByte(addr);      // SCI register number
  WriteSpiByte(data>>8);   // 8 MSb's of data
  WriteSpiByte(data&0xff); // 8 LSb's of data
  DeassertSpiChipSelect();
}
```

6.3.1 Loading and Starting the Code

To load and start the VS1053b WAV PCM Recorder, do the following steps:

1. Start up VS1053b/VS1063a in a normal fashion. There is no need to set the clock register. Do not set SCI_BASS (2) to anything else than the default 0.
2. Disable any potential user application by setting SCI_AIADDR (10) to 0.
3. Load the application (Chapter 8).
4. Activate the application by writing 0x34 to register SCI_AIADDR (10).
5. Wait until DREQ pin goes high.
6. No filters are on by default, so now you will hear unprocessed sound.
7. Start setting filters.

7 Frequency Responses

This Chapter shows example frequency responses for the filters designed either with the 7-Band Equalizer Designer (Chapter 7.1), or manually (Chapter 7.2).

All examples have been generated by running the equalizer software on the VSDSP processor from a digital input to a digital output.

7.1 Frequency Responses: 7-Band Equalizer

All of the figures in this Chapter are for the VS1063.

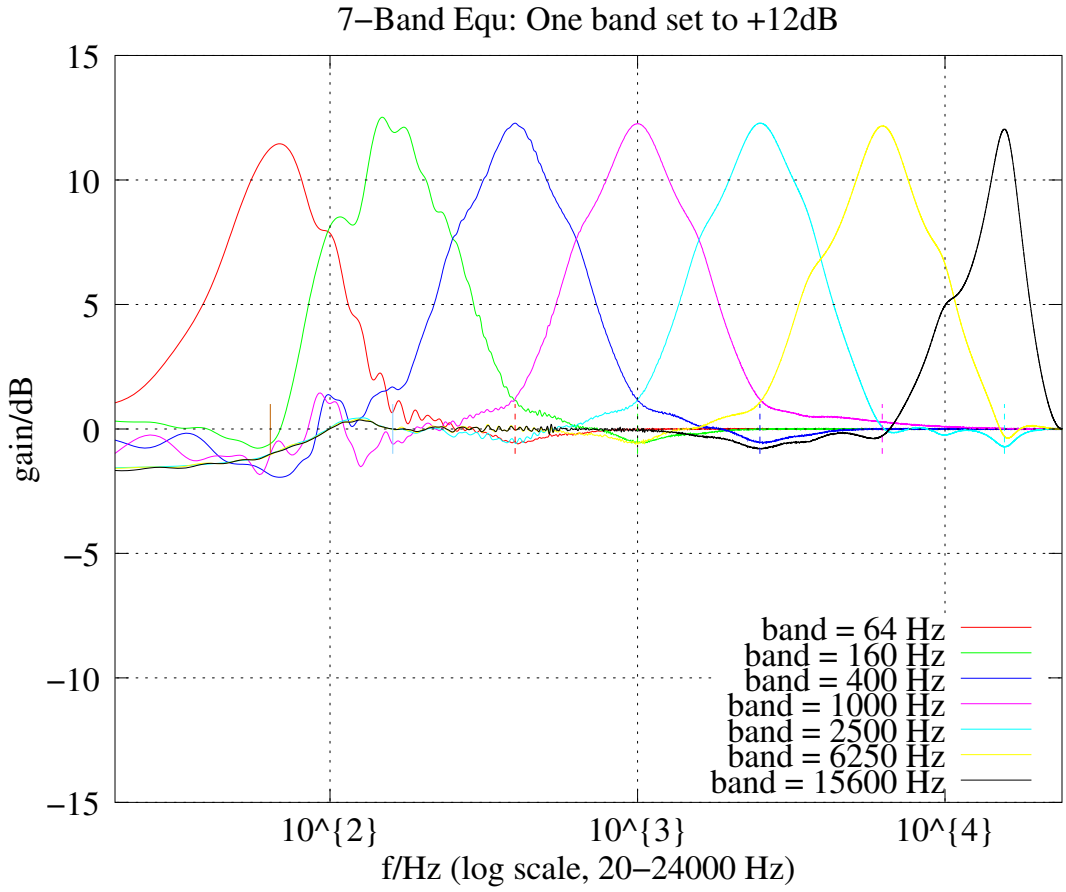

Figure 1: 7-Band Equalizer: One band set to +12 dB, others at 0 dB.

Figure **??** shows the frequency responses for the 7-band equalizer when each of the seven bands are set to +12 dB, while all others are set to 0 dB.

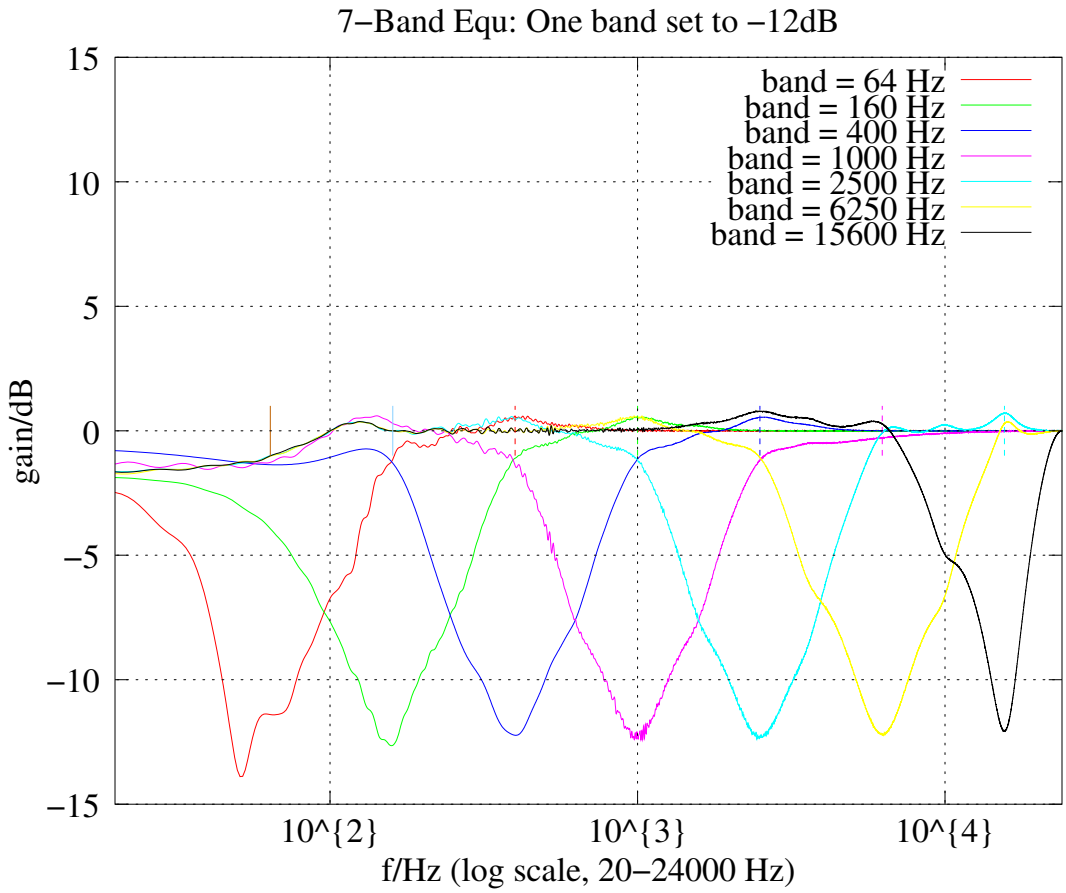

Figure 2: 7-Band Equalizer: One band set to -12 dB, others at 0 dB.

Figure 2 shows the frequency responses for the 7-band equalizer when each of the seven bands are set to -12 dB, while all others are set to 0 dB.

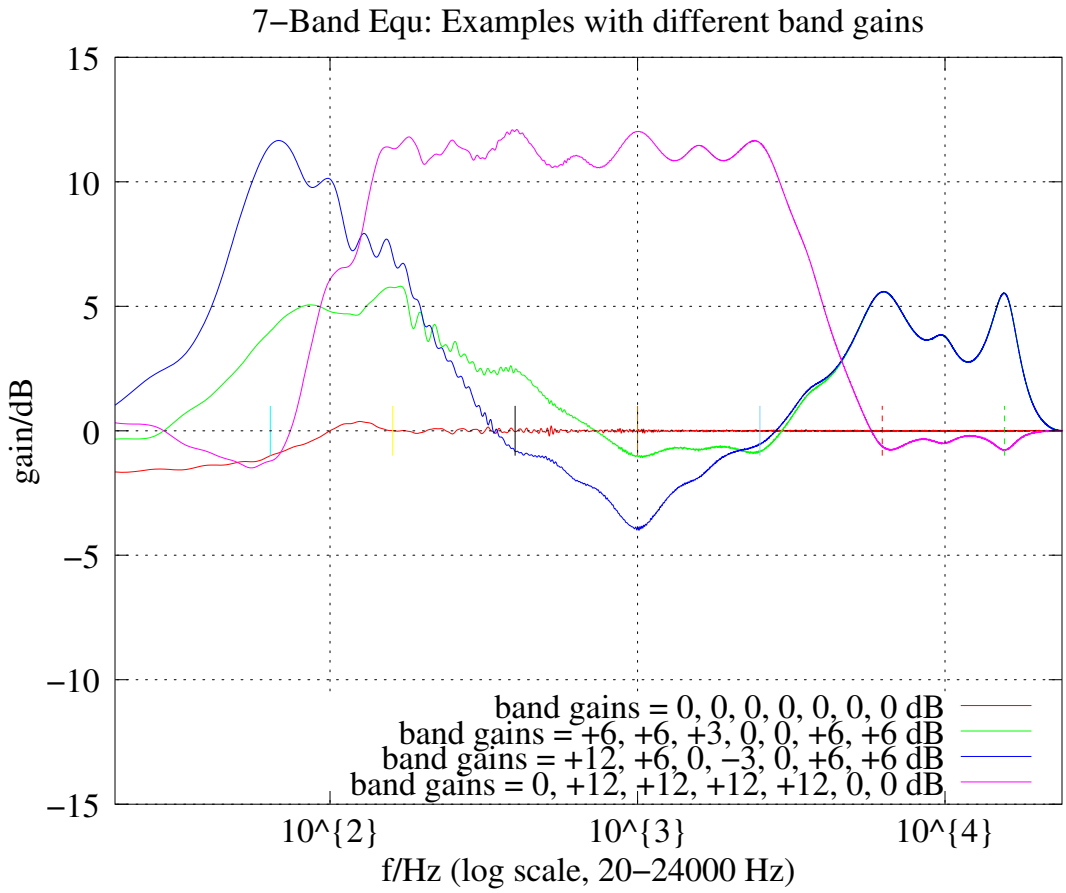

Figure 3: 7-Band Equalizer: Examples with different band gains

fig:f7b3

Figure **??** shows the frequency responses for the 7-band equalizer with four different settings. The first setting is an “all-neutral” setting with all bands set to 0 dB.

The two next ones are different “loudness” curves.

The last example shows how the 7-band filter uses additional filters to smooth its frequency response: while four adjacent bands are set to +12 dB, you can see seven peaks in the frequency response: the four main bands, as well as an extra band between each of them.

7.2 Frequency Responses: Single Filters

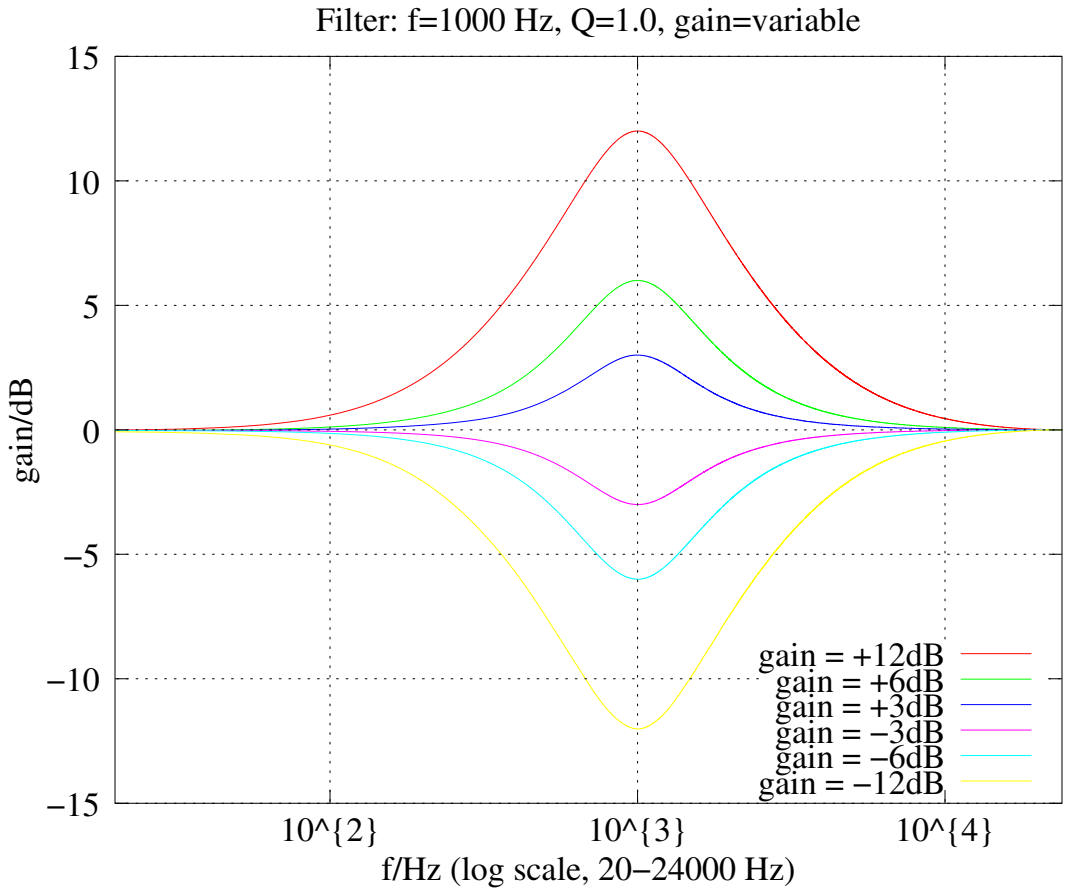

Figure 4: Single filter: Effect of gain.

Figure 4 shows how gain affects a single filter set to 1 kHz, Q=1.0.

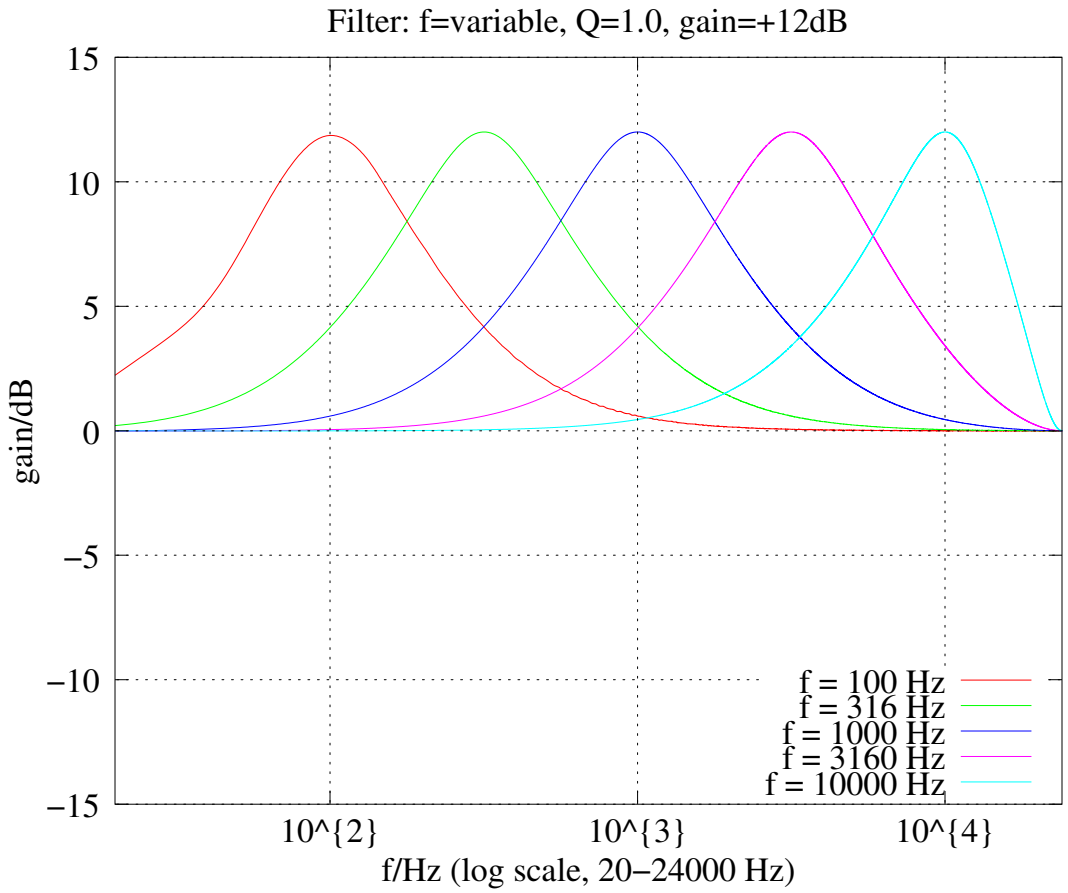

Figure 5: Single filter: Effect of center frequency.

Figure 5 shows how center frequency affects a single filter set to Q=1.0, gain=+12 dB.

Note that filter frequency responses are symmetrical on a logarithmic frequency scale. A slight exception to this are the filters that are close to zero frequency or fs/2 (24000 Hz).

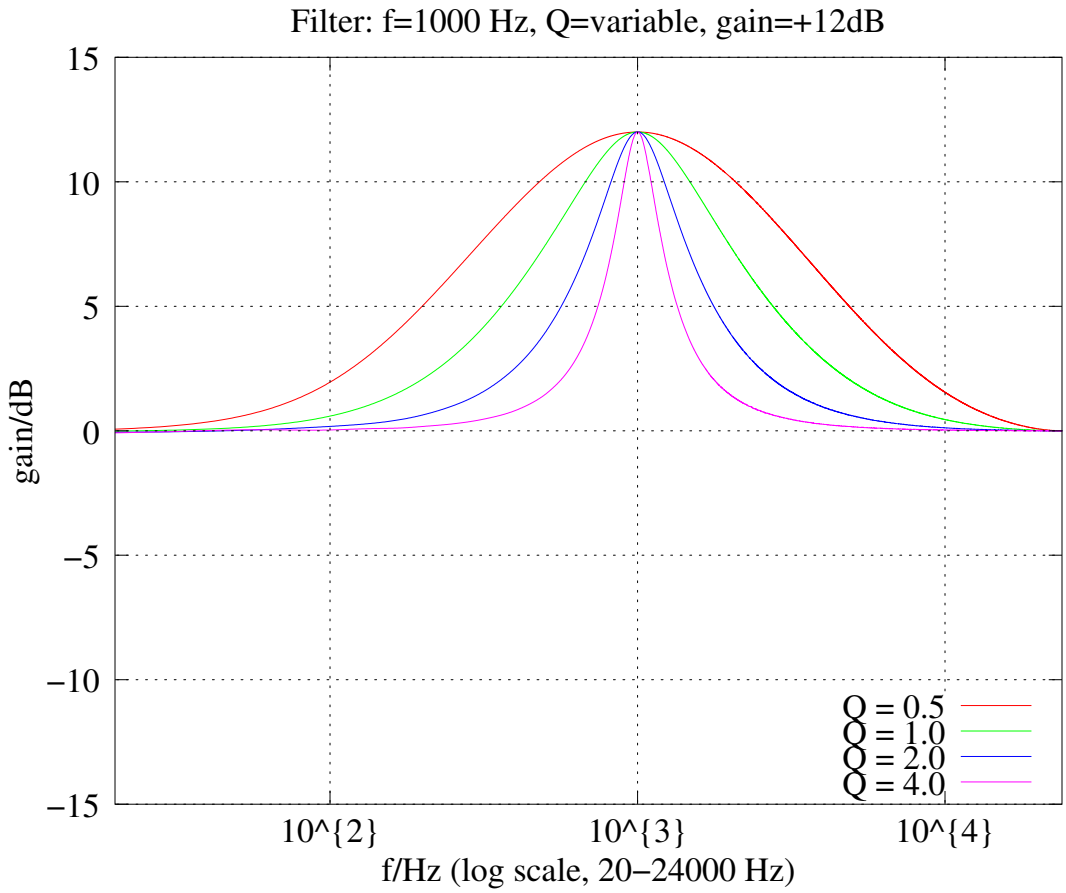

Figure 6: Single filter: Effect of Q factor.

Figure 6 shows how Q factor affects a single filter set to 1 kHz, gain=+12 dB.

8 How to Load a Plugin

A plugin file (.plg) contains a data file that contains one unsigned 16-bit vector called plugin. The file is in an interleaved and RLE compressed format. An example of a plugin vector is:

```
const unsigned short plugin[10] = { /* Compressed plugin */
  0x0007, 0x0001, 0x8260,
  0x0006, 0x0002, 0x1234, 0x5678,
  0x0006, 0x8004, 0xabcd,
};
```

The vector is decoded as follows:

1. Read register address number `addr` and repeat number `n`.
2. If `n` & 0x8000U, write the next word `n` times to register `addr`.
3. Else write next `n` words to register `addr`.
4. Continue until table has been exhausted.

The example vector first tells to write 0x8260 to register 7. Then write 2 words, 0x1234 and 0x5678, to register 6. Finally, write 0xabcd 4 times to register 6.

Assuming the vector is in vector `plugin[]`, a full decoder in C language is provided below:

```
void WriteVS10xxRegister(unsigned short addr, unsigned short value);

void LoadUserCode(void) {
  int i = 0;

  while (i<sizeof(plugin)/sizeof(plugin[0])) {
    unsigned short addr, n, val;
    addr = plugin[i++];
    n = plugin[i++];
    if (n & 0x8000U) { /* RLE run, replicate n samples */
      n &= 0x7FFF;
      val = plugin[i++];
      while (n--) {
        WriteVS10xxRegister(addr, val);
      }
    } else {           /* Copy run, copy n samples */
      while (n--) {
        val = plugin[i++];
        WriteVS10xxRegister(addr, val);
      }
    }
  }
}
```

9 Contact Information

VLSI Solution Oy
Entrance G, 2nd floor
Hermiankatu 8
FI-33720 Tampere
FINLAND

Fax: +358-3-3140-8288
Phone: +358-3-3140-8200
Commercial e-mail: sales@vlsi.fi
URL: http://www.vlsi.fi/

For technical support or suggestions regarding this document, please participate at
http://www.vsdsp-forum.com/
For confidential technical discussions, contact
support@vlsi.fi