VS1003 ADC SPECTRUM ANALYZER

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**
0.10	2012-06-01	PO&HH	Initial version.

Contents

1 VS1003 ADC Spectrum Analyzer

This patch provides upto 23 spectrum band analyzers for implementing a graphical spectrum analyzer display on the microcontroller. The analyzer works for the microphone or line input.

The patch code uses the user X and Y memory areas of VS10xx, so you can't use other patch codes that use these areas at the same time. When you upload the code, it will be automatically started.

Hardware or software reset will deactivate the patch. You must reload the patch at each hardware reset. You must either set the frequencies or reload the patch at each software reset.

Chip	File	Description
VS1003b	vs1003adcSpecAna.plg vs1003adcSpecAna.cmd	Plugin load file Old style command load file

The default number of bands is 14. The default frequencies of the bands are 50, 79, 126, 200, 317, 504, 800, 1270, 2016, 3200, 5080, 8063, 12800, and 20319 Hz. The actual frequencies differ from these slightly, depending among other things on the sampling frequency. Note that the highest bands may not have any information if the sampling frequency is less than 44100 Hz.

The spectrum analyzes sets the processor to 4×XTALI clock speed, and runs audio at 48 kHz. Do *not* write to registers SCI_CLOCKF and SCI_AUDATA, and do *not* read from SCI_AIADDR after starting the analyzer. (You can do this again after exiting the analyzer with a software reset.)

Before you load the plugin, select between line and microphone input by setting or clearing SCI_MODE (reg 0) bit SM_LINE_IN (bit 14), respectively.

1.1 Communication

Address VS10x3	Address VS1011	Field	Usage
0x1801	0x1381	rate	Sample rate
0x1802	0x1382	bands	Bands currently used
0x1804..0x181a	0x1384..0x139a	results	First..last band
0x1868..0x187e	0x13e8..0x13fe	frequencies	First..last band

1.2 Reading Results

You can read the results from X-RAM addresses 0x1804..0x181a, and the number of used bands from 0x1802. Unused bands are kept at zero, thus you can also skip reading of the bands variable and just read a predefined number of bands.

A suitable update period is from 5 to 20 times per second.

```
#define BASE 0x1800 /* 0x1380 for VS1011 */
    WriteSciReg(SCI_WRAMADDR, BASE+2);
    /* If VS1011b, one dummy ReadSciReg(SCI_AICTRL3); here*/
    bands = ReadSciReg(SCI_WRAM); /* If VS1011b, use SCI_AICTRL3 */
    WriteSciReg(SCI_WRAMADDR, BASE+4);
    /* If VS1011b, one dummy ReadSciReg(SCI_AICTRL3); here*/
    for (i=0;i<bands;i++) {
        int val = ReadSciReg(SCI_WRAM); /* If VS1011b, use SCI_AICTRL3 */
        /* current value in bits  5..0, normally 0..31
           peak value    in bits 11..6, normally 0..31 */
    }
```

Six bits are reserved for the current result and a peak value. The values are from zero to 31 in 3 dB steps.

1.3 Setting Bands

You can also change the frequency band center frequencies and the number of bands to best suit your needs. The bands are in XRAM 0x1868..0x187e in ascending order. If not all 23 bands are used, end the list with 25000. To activate the new band selections, write 0 to the sample rate field (0x1801).

```
#define BASE 0x1800 /* 0x1380 for VS1011 */
    int bands = 14;
    static const short frequency[] = {
        50, 79, 126, 200, 317, 504, 800, 1270, 2016, 3200,
        5080, 8063, 12800, 20319
    };
    /* send new frequencies */
    WriteSciReg(SCI_WRAMADDR, BASE+0x68);
    for (i=0;i<bands;i++)
        WriteSciReg(SCI_WRAM, frequency[i]);
    if (i < 23)
        WriteSciReg(SCI_WRAM, 25000);
    /* activate */
    WriteSciReg(SCI_WRAMADDR, BASE+1);
    WriteSciReg(SCI_WRAM, 0);
```

1.4 Reading Frequencies

If you want to read the actual center frequency for each analyzer band, you can use the following code:

```
#define BASE 0x1800 /* 0x1380 for VS1011 */
    WriteSciReg(SCI_WRAMADDR, BASE+1);
    /* If VS1011b, one dummy ReadSciReg(SCI_AICTRL3); here*/
    rate  = ReadSciReg(SCI_WRAM); /* If VS1011b, use SCI_AICTRL3 */
    bands = ReadSciReg(SCI_WRAM); /* If VS1011b, use SCI_AICTRL3 */

    for (i=0;i<bands;i++) {
        int a;
        WriteSciReg(SCI_WRAMADDR, BASE+0x1c+3*i);
    /* If VS1011b, one dummy ReadSciReg(SCI_AICTRL3); here*/
        a = ReadSciReg(SCI_WRAM); /* If VS1011b, use SCI_AICTRL3 */
        freq[i] = (long)rate * a >> 11;

        printf("%2d %3d %5dHz\n", i, a, freq[i]);
    }
```

Example output:

```
 0   2    43Hz
 1   3    64Hz
 2   5   107Hz
 3   9   193Hz
 4  14   301Hz
 5  22   473Hz
 6  36   775Hz
 7  56  1205Hz
 8  88  1894Hz
 9 128  2756Hz
10 224  4823Hz
11 352  7579Hz
12 576 12403Hz
13 928 19982Hz
```

Note that the actual center frequencies can change when sample rate changes.

2 How to Load a Plugin

A plugin file (.plg) contains a data file that contains one unsigned 16-bit array called plugin. The file is in an interleaved and RLE compressed format. Plugins can be easily combined by using preprocessor #include command and the SKIP_PLUGIN_VARNAME define. An example of a plugin array 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 array has been exhausted.

The example array 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 array is in `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);
      }
    }
  }
}
```

3 How to Use Old Loading Tables

Each patch contains two arrays: `atab` and `dtab`. `dtab` contains the data words to write, and `atab` gives the SCI registers to write the data values into. For example:

```
const unsigned char atab[] = { /* Register addresses */
    7, 6, 6, 6, 6
};
const unsigned short dtab[] = { /* Data to write */
    0x8260, 0x0030, 0x0717, 0xb080, 0x3c17
};
```

These arrays tell to write 0x8260 to SCI_WRAMADDR (register 7), then 0x0030, 0x0717, 0xb080, and 0x3c17 to SCI_WRAM (register 6). This sequence writes two 32-bit instruction words to instruction RAM starting from address 0x260. It is also possible to write 16-bit words to X and Y RAM. The following code loads the patch code into VS10xx memory.

```
/* A prototype for a function that writes to SCI */
void WriteVS10xxRegister(unsigned char sciReg, unsigned short data);

void LoadUserCode(void) {
  int i;
  for (i=0;i<sizeof(dtab)/sizeof(dtab[0]);i++) {
    WriteVS10xxRegister(atab[i]/*SCI register*/, dtab[i]/*data word*/);
  }
}
```

Patch code tables use mainly these two registers to apply patches, but they may also contain other SCI registers, especially SCI_AIADDR (10), which is the application code hook.

If different patch codes do not use overlapping memory areas, you can concatenate the data from separate patch arrays into one pair of `atab` and `dtab` arrays, and load them with a single `LoadUserCode()`.

4 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
Email: sales@vlsi.fi
URL: http://www.vlsi.fi/

For technical questions or suggestions regarding this application, please contact support@vlsi.fi.