VS1053B EQ5

VSMPG "VLSI Solution Audio Decoder"

Project Code:	VS1053
Project Name:	Support

Revision History

Rev.	Date	Author	Description
1.0	2014-02-25	PO	Now works with WAV too.
0.9	2012-08-08	PO	Initial version.

Contents

List of Figures

1 Description

EQ5 implementation for VS1053B, compatible with vs1053b Patches Package.

Chip	File	Description
VS1053B	vs1053b-eq5.plg	compressed plugin
VS1053B	vs1053b-eq5.c	old array format

This plugin uses the application address. Write 0xbc0 to SCI_AIADDR to activate. This plugin uses memory areas: IRAM 0xbc0..0xef1, XRAM 0x35cb..0x36ff, and YRAM 0xfb3c..0xfcad.

Both old loading tables and the new compressed plugin format is available. The new plugin format is recommended, because it saves data space and future plugins, patches, and application will be using the new format.

The VS1053b Patches package (without FLAC decoder) and the VS1053 AD Mixer do work with this package, as long as you load and start those first.

The VS1053 PCM Mixer and the VS1053b Patches package with FLAC decoder do not work with this plugin. (Instruction memory overlaps.)

See the documentation of the configuration values from vs1063 datasheet. Note that the configuration structure is at 0x3567 with this plugin (instead of at 0x1e12 in vs1063).

Example configuration:

Reg	Value	Description
SCI_VOL	0x1010	Set volume, -8 dB allows upto 8 dB gain for each band
SCI_AIADDR	0x0bc0	Start the EQ5 plugin, can also be the last write
SCI_WRAMADDR	0x3567	Start of the EQ5 parameters
SCI_WRAM	0	eq5Dummy – Not used
SCI_WRAM	6	eq5Level1 – Bass level in 1/2 dB steps
SCI_WRAM	70	eq5Freq1 – Bass/Mid-Bass cutoff in Hz
SCI_WRAM	0	eq5Level2 – Mid-Bass level in 1/2 dB steps
SCI_WRAM	300	eq5Freq2 – Mid-Bass/Mid cutoff in Hz
SCI_WRAM	0	eq5Level3 – Mid level in 1/2 dB steps
SCI_WRAM	3000	eq5Freq3 – Mid/Mid-High cutoff in Hz
SCI_WRAM	5	eq5Level4 – Mid-High level in 1/2 dB steps
SCI_WRAM	8000	eq5Freq4 – Mid-High/Treble cutoff in Hz
SCI_WRAM	9	eq5Level5 – Treble level in 1/2 dB steps
SCI_WRAM	1	eq5Update – setting to 1 updates the settings
SCI_WRAMADDR	0x1e09	Address of parametric_x.playmode
SCI_WRAM	0x20	Enable EQ5 from playmode

Also note that SCI_BASS should be 0, because bass and treble controls override EQ5.

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. 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()`.