VS1053B IMA ENCODING FIX

VSMPG “VLSI Solution Audio Decoder”

Project Code:
Project Name: VSMPG

Revision History

Rev.	Date	Author	Description
1.1	2017-11-17	PO	Encoding fix.
1.0	2008-05-23	PO	Initial version.

1 Description

The new VS1053 audio path provides a low-delay monitoring of mono and stereo analog-to-digital conversion through DAC. It also allows a wide range of sample rates for the encoded data. However, the transfer of encoded data never starts in VS1053B. This problem can be corrected with a small patch code.

Chip	**File**	**IRAM**	**Description**
VS1053B	imafix.c	0x50 .. 0x70	old atab+dtab array format
VS1053B	imafix.plg	0x50 .. 0x70	new compressed plugin file

This patch rewrites the SRC interrupt vector and restarts the firmare, so the patch should be loaded after the encoding parameters have been set in AICTRL0 through AICTRL3, and the IMA ADPCM bit has been set in SCI_MODE (without setting the software reset bit).

1. Set SM_ADPCM bit in SCI_MODE, and set or clear the SM_LINE bit depending on your input. Do not set the software reset bit!
2. Set other encoding parameters in AICTRL0 through AICTRL3.
3. Load this patch. The encoding mode will start automatically.
4. Transfer data through SCI_HDAT1 and SCI_HDAT0.

There are two versions: one with the old loading tables, and one with the new compressed plugin format. The new compressed plugin format is recommended, because it saves data space and future plugins, patches, and application will be using the new format.

2 How to Load a Plugin

A plugin file (.plg) contains one unsigned 16-bit array called plugin. The file is in an interleaved and RLE-compressed format. The imafix plugin array is:

```
#ifndef SKIP_PLUGIN_VARNAME
const unsigned short plugin[] = { /* Compressed plugin */
#endif
  0x0007,0x0001, /*copy 1*/
  0x8050,
  0x0006,0x0042, /*copy 66*/
  0x0000,0x1790,0xf400,0x5400,0x0000,0x0a10,0xf400,0x5600,
  0xb080,0x0024,0x0007,0x9257,0x3f00,0x0024,0x0030,0x0297,
  0x3f00,0x0024,0x0000,0x004d,0x0014,0x958f,0x0000,0x1b4e,
  0x280f,0xe100,0x0006,0x2016,0x2a00,0x17ce,0x3e12,0xb817,
  0x3e14,0xf812,0x3e01,0xb811,0x0007,0x9717,0x0020,0xffd2,
  0x0030,0x11d1,0x3111,0x8024,0x3704,0xc024,0x3b81,0x8024,
  0x3101,0x8024,0x3b81,0x8024,0x3f04,0xc024,0x2808,0x4800,
  0x36f1,0x9811,0x2814,0x9c91,0x0000,0x004d,0x2814,0x9940,
  0x003f,0x0013,
  0x000a,0x0001, /*copy 1*/
  0x0050,
#define PLUGIN_SIZE 74
#ifndef SKIP_PLUGIN_VARNAME
};
#endif
```

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 imafix plugin tells to write 0x8050 to register 7. Then write 66 (0x42) words to register 6, starting with values 0x0000 and 0x1790. Finally, write 0x50 to register 0x0a.

A generic plugin loader 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()`.