VS1033C MPEG2.0 Layer 2 Patch

VSMPG "VLSI Solution Audio Decoder"

Project Code:
Project Name: VSMPG

Revision History

Rev.	**Date**	**Author**	**Description**
1.2	2008-08-25	PO	Better decoding accuracy for VS1011e
1.1	2007-03-14	PO	Better decoding accuracy for VS1033c
1.0	2007-03-02	PO	Initial version

1 Description

VS1011E and VS1033C support also MPEG layer I and II decoding if you enable it with the SCIMB_ALLOW_LAYER12 bit in the mode register. However, layer II decoding does not handle the halfrate extension (24000 Hz, 22050 Hz, and 16000 Hz sample rates) files correctly. The frame size is calculated incorrectly and the files just do not play.

This patch corrects the halfrate bug.

In addition, the VS1033C and VS1011E versions of the patch further increase the layer II decoding accuracy in two of the most important cases.

Chip	**File**	**IRAM Location**
VS1033C	layer2patch.c	0x260 .. 0x2bd
VS1033C	layer2patch.plg	0x260 .. 0x2bd
VS1033B	layer2patch33b.c	0x260 .. 0x278
VS1033B	layer2patch33b.plg	0x260 .. 0x278
VS1011E	layer2patch11e.c	0x1e0 .. 0x242
VS1011E	layer2patch11e.plg	0x1e0 .. 0x242

Hardware or software reset will deactivate the patch. You must reload the patch after each hardware and software reset.

This patch uses the application address to start automatically (the last entry in the patch tables writes to SCI_AIADDR), but does not use it afterwards. So, you must load any patch that actually uses the application address after this patch or it will be deactivated.

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()`.