VS1063A SDIENCODER

"VLSI Solution Audio Decoder/Encoder"

Project Code: VS1063
Project Name: Support

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

Revision History			
Rev.	**Date**	**Author**	**Description**
0.9	2013-06-07	POj	Updated the mp3 encoder fix.
0.81	2011-10-28	POj	Now has the mp3 encoder fix activated.
0.8	2011-10-24	POj	Initial version.

Contents

Table of Contents **2**

1 VS1063a SDI Encoder **3**
1.1 Description 3
1.2 Files 3
1.3 Additional Information 3

2 Usage **4**
2.1 How to Encode WAV to Other Formats 4
2.2 How to Encode Raw PCM to Other Formats 4
2.3 Known Shortcomings 5

3 How to Load a Plugin **6**

4 How to Use Old Loading Tables **7**

List of Figures

1 VS1063a SDI Encoder

1.1 Description

VS1063 encodes data from the microphone or line input, but sometimes you want to encode audio that is already in digital format. This application is created for you. This application receives a WAV file from SDI and encodes it with any of the available encoding formats to be read through SCI.

Several WAV input formats are supported in mono and stereo: 8-bit and 16-bit PCM, μlaw, Alaw, G.722 ADPCM, and IMA ADPCM.

The Huffman code patch for the VS1063a MP3 encoder is included. (See VS1063 Patches package for a detailed explanation.)

See VS1063 datasheet Section 10.7 "Audio Encoding" for more information about how to read data through SCI and how to fix the RIFF WAV file size fields.

1.2 Files

Chip	File	IRAM	Description
VS1063A	vs1063a-sdiencoder.c	0x50..0x79d	old array format
VS1063A	vs1063a-sdiencoder.plg	0x50..0x79d	compressed plugin
VS1063A	vs1063a-sdiencoder.cmd	0x50..0x79d	legacy command format

The application only works with VS1063A.

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.

Unless otherwise specified, this patch is **not** compatible with other vs1063a plugins. You need to give a software reset and load the right code when you switch between different plugins.

1.3 Additional Information

If you find a bug, a curious feature, or are unsure how to use VS1063, let us know.

VSDSP Forum is at **http://www.vsdsp-forum.com/** .

Support e-mail address is **support@vlsi.fi** .

2 Usage

2.1 How to Encode WAV to Other Formats

1. Send the application from .plg or .c file.
2. Write the encoding format to SCI_AICTRL3.
3. Write encoding parameters to SCI_WRAMADDR (for MP3 and Ogg Vorbis).
4. Start encoding by writing 0x50 to SCI_AIADDR. Wait until DREQ rises.
5. Then send the WAV file to SDI.
   - Send data to SDI 32 bytes at a time whenever DREQ is high.
   - SCI_HDAT1 becomes non-zero when encoded data is available. Then read the indicated number of encoded words from SCI_HDAT0.
6. End encoding in the normal way by setting SM_CANCEL in SCI_MODE. Send zeros to SDI, and read the remainder of the encoded data.
7. Give software reset to return to normal decoding mode.

2.2 How to Encode Raw PCM to Other Formats

1. Send the application from .plg or .c file.
2. Write the sample rate to SCI_AICTRL0.
3. Write the encoding format and channel configuration to SCI_AICTRL3.
4. Write encoding parameters to SCI_WRAMADDR (for MP3 and Ogg Vorbis).
5. Start encoding by writing 0x50 to SCI_AIADDR. Wait until DREQ rises.
6. Then send `"RAW "` to select 16-bit little-endian, or `"RAWB"` to select 16-bit big-endian PCM audio data. Then send the audio data to SDI.
   - Send data to SDI 32 bytes at a time whenever DREQ is high.
   - SCI_HDAT1 becomes non-zero when encoded data is available. Then read the indicated number of encoded words from SCI_HDAT0.
7. Send 2048 zeros to flush the SDI FIFO, then end encoding in the normal way by setting SM_CANCEL in SCI_MODE. Send zeros to SDI, and read the remainder of the encoded data. (See VS1063 datasheet for the full end procedure.)
8. Give software reset to return to normal decoding mode.

AICTRL3 Bits Supported		
Bits	**Symbol**	**Description**
0..3	Channel mode	use 0 for stereo, 2 for mono (for RAW modes)
10	AICTRL3_NORIFF	When set does not create WAV header.
12	AICTRL3_ENABLEUART	UART mode (not tested)

See how to use the UART mode from VS1063 datasheet.

2.3 Known Shortcomings

The end of encoding is not exact. This is caused because of the audio frame format of mp3 and Ogg Vorbis and also because of the way encoding is ended. If you send all data to SDI and set SM_CANCEL, some data may be left unencoded. If you send all data, then send 2048 zeros to flush SDI buffer before setting SM_CANCEL, you may get some extra silence at the end of the encoded file.

The WAV file file size or data size fields are not currently used. In a future version the WAV data size could be used to determine when to end encoding whenever `parametric_x.resync` is set to 0.

3 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);
      }
    }
  }
}
```

4 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.