VS1053B AD MONITOR WITH EQ AND PITCH SHIFTER

VSMPG “VLSI Solution Audio Decoder”

Project Code: VS1053
Project Name: Support

Revision History			
Rev.	**Date**	**Author**	**Description**
1.2	2020-01-16	PO	Added Pitch Shifter
1.1	2011-06-09	PO	First release version

Contents

List of Figures

1 Description

Sometimes you would like to route the signal at the analog inputs of vs1053b to the analog outputs. You can do this by starting the encoding mode, but the encoding mode doesn't use the Bass or Treble controls.

This small application reads data from the ADC, optionally performs Bass and Treble control, and plays the result from the DAC.

AICTRL0, 1, 2, and 3 are used like in the normal vs1053b encoding mode. Supported rates are: 8000 Hz, 12000 Hz, 16000 Hz, 24000 Hz, and 48000 Hz. Generally you would select 48000 Hz, or 24000 Hz with pitch shifter. XTALI must be 12.288 MHz for the exact rates.

CLOCKF should be set to for example 0x6000 (for 3.0x clock) or higher.

- Configure SCI_CLOCKF.
- If other applications, plugins, or patches are loaded, give software reset.
- Configure AICTRL0..3
  - AICTRL0 for samplerate, must be exactly one of the supported rates.
  - AICTRL1 = 0 for AGC, other values for fixed gain.
  - AICTRL2 to set AGC maximum amplification, e.g. 0x1000.
  - AICTRL3 = 2 for left channel mono, 3 for right channel mono, 0 or 1 for stereo modes, or (special to this plugin) use 256 or 257 to convert stereo inputs into mono.
- Clear the SM_LINE bit from the MODE register, if you want to select MIC.
- Optionally write SCI_BASS to configure bass and treble controls.
- Load the application, it starts automatically.

The code does not set the ADPCM bit in the mode register, and does not provide any data transfer. SDI data is ignored.

You can change SCI_VOL, SCI_BASS, EarSpeaker, AGC settings, and ADC mode during the running of the plugin. To change the samplerate you need to restart.

When the patch is loaded, it starts automatically and restarts the system (you should wait for DREQ to rise).

This works with the regular version of vs1053b Spectrum Analyzer, but not with other applications, plugins, or patches. Load the spectrum analyzer after loading and starting the AD Monitor. Spectrum analyzer works on the ADC signal, not the pitch-shifted signal, and is unaffected by bass and treble controls.

1.1 Pitch Shifting

This application also allows to pitch shift the signal. AICTRL0 will be set to 0x4000 after the application has started. After this AICTRL0 controls pitch shifting - 0x4000 is the neutral setting.

SCI_AICTRL0 sets the pitch to use. Register value 0x4000 (16384) is equal to 1.0, i.e. no change. Values above 1.0 raise the pitch, and values below 1.0 lower the pitch. The intended range is $0.7071 \ldots 1.3348$, but if more CPU power is available, $0.5 \ldots 1.9999$ can be used.

Pitch Shift configuration can be changed at any time, but there may be some artefacts when the ratio changes between < 1.0, 1.0, and > 1.0.

The following table lists pitch shifter control values for the intended one octave range. Other ratios between 0.7 and 1.34 can also be used. AICTRL0 $= +16384 \times ratio$.

Pitch Shifter Control		
Ratio	**AICTRL0**	**Description**
0.70711	11585	-6 Down six semitones
0.74915	12274	-5 Down five semitones
0.79370	13004	-4 Down four semitones
0.84089	13777	-3 Down three semitones
0.89089	14595	-2 Down two semitones
0.94400	15466	-1 Down one semitone
1.0000	16384	+0 Neutral position, no extra CPU needed
1.0595	17359	+1 Up one semitone
1.1250	18432	+2 Up two semitones
1.1890	19481	+3 Up three semitones
1.2599	20642	+4 Up four semitones
1.3348	21869	+5 Up five semitones

If 48000 Hz sample rate or higher is used, pitch shifting up would exceed the maximum sample rate of the DAC (48000 Hz with 12.288 MHz clock). Use 24000 Hz rate if you want to pitch shift up.

1.2 Examples

An example configuration:

```
SCI_CLOCKF = 0x8000
SCI_MODE   = 0x0c00
SCI_AICTRL0 = 0x5dc0
SCI_AICTRL1 = 0x0000
SCI_AICTRL2 = 0x1000
SCI_AICTRL3 = 0x0002
SCI_BASS = 0x0000
SCI_VOL  = 0x1010
```

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()`.