# VS10XX PITCH SHIFTER

## VSMPG “VLSI Solution Audio Decoder”

Project Code:
Project Name: VSMPG

| Revision History | | | |
|---|---|---|---|
| **Rev.** | **Date** | **Author** | **Description** |
| 1.31 | 2012-04-05 | HH | Added .plg format data. Software is unchanged. |
| 1.3 | 2008-06-12 | PO | VS1033d version added, uninitialized variable fixed. |
| 1.2 | 2007-11-09 | PO | VS1053b version added |
| 1.1 | 2006-10-11 | PO | VS1033c version added, combined AICTRL0 and AICTRL1 |
| 1.0 | 2005-09-28 | PO | Initial version |

# 1 Pitch Shifter / Tempo Change

In special applications you want to change the pitch of a song without changing the tempo (pitch shifter), or you want to change the tempo without changing the pitch. You can not do pitch shifting by simply changing the sample rate, because the tempo would also change. Special algorithm is needed.

This application code implements both the pitch shifter and tempo changer.

- **This application is highly chip-specific. It only works on the exact firmware versions mentioned.**
- Other patches can not be used at the same time.
  - Uses: user hook, application address, dac interrupt, user I-, X-, and Y-RAM
- Also, PlusV decoding should not be enabled. (SCIMB_PLUSV should be 0.)
- 3.6 MHz more CPU is required when pitch shifter is active.
  If tempo is faster than normal, more CPU is needed for decoding as well.
  - VS1002D requires higher clock (e.g. $2 \times 14.318$ MHz, also higher VDD) **or** restrictions to maximum bitrate (e.g. max. 128 kbit/s) **or** restrictions to maximum sample rate (e.g. max. 32 kHz)
  - In other chips you can use higher PLL setting (e.g. $3.5\times$)
- Best audio quality is achieved with the 960-word pitch buffer.
- For MIDI songs, the best quality can be achieved using MIDI master tune control. Exact details vary depending on the chip version.

| Chip | File | Features |
|---|---|---|
| VS1002D | `ps1002tab.c` | 480-word pitch buffer |
| VS1003B | `ps1003tab.c` | 960-word pitch buffer |
| VS1033C | `ps1033ctab.c` | 960-word pitch buffer |
| VS1033D | `ps1033dtab.c` | 960-word pitch buffer |
| VS1053B | `ps1053btab.c` | 960-word pitch buffer |

Perform normal configuration of SCI_VOL, SCI_CLOCKF, and other registers. Then upload the code, set the configuration in SCI_AICTRL0, then start the code by writing 0x30 to SCI_AIADDR (0x50 for VS1053). Configuration can be changed at any time, but there may be some artefacts when the ratio changes between $< 1.0$, $1.0$, and $> 1.0$.

| SCI registers | | |
|---|---|---|
| **Reg** | **Abbrev** | **Description** |
| **0xA** | **AIADDR** | **Application private** |
| **0xC** | **AICTRL0** | **Pitch / Tempo Control** |

If 44100 Hz sample rate or higher is used, pitch shifting by a higher ratio than 1.088 would exceed the maximum sample rate of the DAC (48000 Hz with 12.288 MHz clock). SCI_AIADDR is used by the application to down-sample rates above 44100 Hz. Down-sampling is not done when pitch/tempo is 1.0. The user should not change the SCI_AIADDR value when the pitch shifter is active.

SCI_AICTRL0 sets the pitch or tempo to use. Register value 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. Intended range is $0.7071 \ldots 1.3348$, but if more CPU power is available, $0.5 \ldots 1.9999$ can be used.

The sign of SCI_AICTRL0 selects the operation mode. If the register value is negative, tempo is changed without changing pitch. Otherwise tempo is kept the same, but pitch is changed.

When tempo change is active, values above 1.0 slow down the playback, and values below 1.0 speed up the playback. Note that if you speed up the playback, decoding will require more CPU.

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 |

The following table lists some tempo control values. Other ratios can also be used, but remember that more decoding power is required when tempo is faster. AICTRL0 $= -16384/ratio$.

| Tempo Control | | |
|---|---|---|
| **Ratio** | **AICTRL0** | **Description** |
| 1.5 | -10923 | 50% faster, decoding needs 50% more CPU |
| 1.25 | -13107 | 25% faster, decoding needs 25% more CPU |
| 1.1 | -14894 | 10% faster, decoding needs 10% more CPU |
| 1.0 | -16384 | Normal speed, no extra CPU needed |
| 0.9 | -18204 | 10% slower |
| 0.8 | -20480 | 20% slower |

# 2 How to Load a Plugin

## 2.1 How to Load a .PLG File

A plugin file (.plg) contains a data file that contains one unsigned 16-bit vector called plugin. The file is in an interleaved and RLE compressed format. An example of a plugin vector 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 table has been exhausted.

The example vector 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 vector is in vector `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);
      }
    }
  }
}
```