VS1002d/VS1003 8kHz System

VSMPG "VLSI Solution Audio Decoder"

Project Code:
Project Name: VSMPG

Revision History			
Rev.	**Date**	**Author**	**Description**
0.6	2017-02-24	PO	Recompile, some saturation operations added
0.5	2005-02-14	PO	More detailed, SCI timing
0.1	2005-01-18	PO	Initial version

Table of Contents

1 System Overview

In addition to DAC and earphone driver, the VS1002d and VS1003 contain a microphone input that can be used in two-way communications in mobile phones or general head-set applications. However, there is a fundamental problem: other than 12.288 MHz input clocks can not generate exactly 8 kHz sample rates to neither DAC nor ADC.

1.1 12 MHz, 13 MHz

This VS1002d/VS1003 8kHz system is designed to implement exactly 8 kHz sample rates for both DAC and ADC with 12 MHz and 13 MHz crystals (or 24 MHz and 26 MHz without the clock doubler with VS1002d). Some other clocks are also permitted.

The system block diagram is shown in Figure 1.1. Three volume settings control the earphone amplitude (main volume), microphone feedback (monitor volume), and microphone gain (mic volume). Microphone volume can also select automatic gain control and the maximum gain to be used can be selected. Limiting the maximum gain helps to keep noise levels down when there is no signal.

Loopback from microphone to earphone is implemented with minimal delay.

Optionally, it is possible to use IMA ADPCM encoding for the input and output datastream to limit the needed data bandwidth. The block size used is 32 bytes, which contains 57 IMA-ADPCM-encoded samples.

Figure 1.1: VS1002d/VS1003 8kHz System, 12/13 MHz

Figure 1.2: VS1002d/VS1003 8kHz System, 12.288 MHz

1.2 12.288 MHz

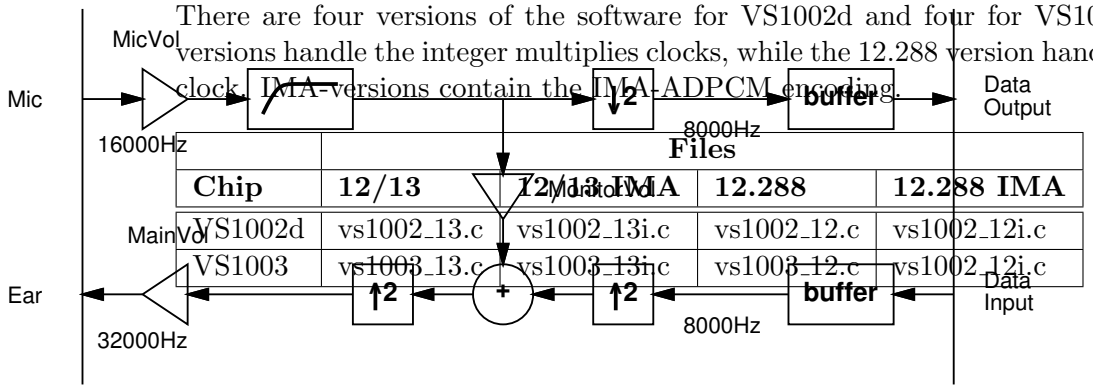
Even when the 12.288 MHz clock is used, the DAC samplerate can not be generated exactly although the ADC rate is correct. In addition, directly using a sample rate of 8000 Hz would generate performance problems with ADC because of aliasing, and the sound delay in the feedback to the earphone would increase.

Thus, a separate program handles the 12.288 MHz (or 24.576 MHz without clock doubler with VS1002d) system. The system block diagram is shown in Figure 1.2.

Both systems manipulate the DAC frequency to get exact output frequency.

1.3 Files

There are four versions of the software for VS1002d and four for VS1003. The 12/13 versions handle the integer multiplies clocks, while the 12.288 version handles the nominal clock. IMA-versions contain the IMA-ADPCM encoding.



Files				
Chip	**12/13**	**12/13 IMA**	**12.288**	**12.288 IMA**
VS1002d	vs1002_13.c	vs1002_13i.c	vs1002_12.c	vs1002_12i.c
VS1003	vs1003_13.c	vs1003_13i.c	vs1003_12.c	vs1002_12i.c

2 Usage

2.1 Initialization

The input clock must be set in register SCI_CLOCKF like in normal mode. Notice that VS1002d and VS1003 use SCI_CLOCKF differently. The application code is loaded to VS1002d / VS1003 using the SCI_WRAMADDR and SCI_WRAM registers and started by writing 0x0030 to SCI_AIADDR. After this the system can be controlled through SCI registers.

The supported input clocks for 12/13 MHz code are 11 MHz, 12 MHz, 13 MHz, 14 MHz, 15 MHz, and 16 MHz for both VS1002d and VS1003, and 22 MHz, 24 MHz, 26 MHz, 28 MHz, 30 MHz, and 32 MHz for VS1002d. If none of these are detected, 12 MHz is assumed.

The supported input clock for 12.288 MHz code is 12.288 MHz for both VS1002d and VS1003, and also 24.576 MHz for VS1002d.

With VS1002d the clock doubler will be enabled for clocks below 20 MHz, VS1003 is always run in 2.0× mode.

Normal mode is restored by setting the software reset bit in the SCI_MODE register.

The following code example shows how to load the patch code to VS1002d/VS1003 memory and how to activate it.

```
#include "vs10xx.h"
#include "vs1002_13.c"
void LoadUserCode(void) {
  int i;
  for (i=0;i<CODE_SIZE;i++) {
    WriteVS10xxRegister(atab[i], dtab[i]);
  }
}

void ActivateUserCode(void) {
  WriteVS10xxRegister(0x0a, 0x0030);
}
```

Note that this user application takes full control over the whole system. If any other user applications or patches are active, you should perform software reset (and set SCI_CLOCKF) before loading and activating the 8 kHz system code.

2.2 Control

The following SCI registers control the behavior of the 8 kHz system. When the 8 kHz system software has been started, no other SCI registers should be used.

Register	Controls	Range	Explanation
MODE	Mode		Software Reset detected
CLOCKF	Clock Setting		Defines the external clock
AICTRL0	Monitor Volume	0..32000	Feedback gain (linear)
AICTRL1	Mic Volume	0..65535U	Mic gain upto 64×, 0=autogain
AICTRL2	Mic Max Volume	1..65535U	Max gain the autogain can use
AICTRL3	Main Volume	0..32767	Earphone volume (linear)
AUDATA	Input Data		Data in (8 kHz)
DECODE_TIME	Input Fullness		Samples waiting to be played
HDAT1	Data Count		Waiting data count
HDAT0	Output Data		Waiting data

2.2.1 SCI_MODE

The SCI_MODE register only uses the software reset bit (SM_RESET). When this bit is set by the user, the 8 kHz system will return control to the normal firmware code by performing a software reset.

2.2.2 SCI_CLOCKF

The input clock frequency is read from this register. The normal VS1002d or VS1003 SCI_CLOCKF configuration should be used.

The supported input clocks for 12/13 MHz code are 11 MHz, 12 MHz, 13 MHz, 14 MHz, 15 MHz, and 16 MHz for both VS1002d and VS1003, and 22 MHz, 24 MHz, 26 MHz, 28 MHz, 30 MHz, and 32 MHz for VS1002d. If none of these are detected, 12 MHz is assumed.

The supported input clock for 12.288 MHz code is 12.288 MHz for both VS1002d and VS1003, and also 24.576 MHz for VS1002d.

With VS1002d the clock doubler will be enabled for clocks below 20 MHz, VS1003 is always run in 2.0× mode.

2.2.3 SCI_AICTRL0

SCI_AICTRL0 is monitor volume, which controls the feedback gain. Valid values are from zero (muted) to 32000 (approximately 1×). The default value after startup is 32000.

2.2.4 SCI_AICTRL1

SCI_AICTRL1 is mic volume, which controls the microphone gain. Valid values are from zero (muted) through 1024 (1× gain) to 65535 (64×, i.e. 36 dB). If the value is 0, automatic gain control is used and SCI_AICTRL2 limits the maximum gain. The default value after startup is 0.

2.2.5 SCI_AICTRL2

If SCI_AICTRL1 is 0, SCI_AICTRL2 gives the maximum gain the automatic gain control can use. Valid values are from zero through 1024 (1× gain) to 65535 (64×, i.e. 36 dB). The default value after startup is 32768 (32×) for VS1002 and 16384 (16×) for VS1003. Do not use SCI_AICTRL2 for muting, use SCI_AICTRL1 instead.

2.2.6 SCI_AICTRL3

SCI_AICTRL3 is main volume, which controls the earphone gain. Valid values are from zero (muted) to 32767. The default value after startup is 32767.

2.2.7 SCI_AUDATA

Input data should be sent in 125 μs intervals (i.e. 8 kHz). A circular buffer allows you to send upto 255 samples beforehand. If the system runs out of data (you send samples too slowly), a zero-sample is inserted. If you send samples too fast, they are accumulated until the input data buffer overruns. If you do not have a 8 kHz timebase, you can use SCI_DECODE_TIME to see how many samples are waiting to be played.

2.2.8 SCI_DECODE_TIME

SCI_DECODE_TIME is the input buffer fullness indicator. If you have an exact 8 kHz timebase and are sending samples at the correct frequency, the number of samples in the input buffer should remain fairly low.

If you have a need to resynchronize, abstain from sending new samples until the input buffer is empty.

When IMA-ADPCM is in use, the SCI_DECODE_TIME contains the number of input samples after IMA-ADPCM decoding. Thus it stays zero until you have sent a full IMA-ADPCM block (32 bytes), and then jumps to 57.

2.2.9 SCI_HDAT0

Output data is read from the SCI_HDAT0 register. The number of available samples is in SCI_HDAT1. Also the DREQ pin reflects the state of the output data buffer. If there is no data to read, DREQ stays low, otherwise DREQ is high. If you read too much data (or too fast), the previous sample value is returned. If you read too little data (or too slowly), it is accumulated until the output buffer overruns.

Because the software updates the contents of SCI_HDAT0, you should read the register with enough wait in-between (7.5 μs). Otherwise the state of DREQ, and contents of both SCI_HDAT1 and SCI_HDAT0 may not be correct.

2.2.10 SCI_HDAT1

SCI_HDAT1 contains the number of words waiting in the output buffer.

When IMA-ADPCM is in use, SCI_HDAT1 stays zero until 57 samples have been gathered and IMA-ADPCM-encoded. Then it jumps to 16 (32 bytes).

2.3 SCI Timing

Because the software must react to SCI register reads to update the register contents, there must be at least a 7.5 μs delay between reads/writes.

2.4 IMA ADPCM

In normal PCM operation 16-bit 8 kHz samples are read and written through the SCI interface. When the input and output buffer fullnesses are kept low, the audio delays are minimal.

It is possible to reduce the needed data bandwidth by approximately 4 by using IMA-ADPCM encoding with a 32-byte block size (57 samples). However, when IMA-ADPCM is used, the audio delay is increased because of the block encoding and decoding by at least 57 samples, i.e. $57/8000\ Hz = 7.125\ ms$. Because the data transfer is not instantaneous, in practice the delay is closer to 2*57 samples, i.e. $14.25ms$.

If word-synchronization is lost, the 8 kHz system automatically re-synchronizes to the IMA blocks. However, both the normal PCM and IMA modes do not perform any byte-synchronization. This must be correctly handled by the transfer system.

2.5 Example Usage

The following example depends on synchronization given by VS1002d when reading samples, and depends on its peer when writing samples.

```
#include "vs1002_13.c"
  WriteVS10xxRegister(SCI_CLOCKF, 0x8000U|(13000/2));
  LoadUserCode();
  ActivateUserCode();
  while (1) {
    short inData;
    if (ReadVS10xxRegister(SCI_HDAT1)) {
      /* Read/WriteVS10xxRegister() is slower than
         7.5us, thus no delay needed */
      SendData(ReadVS10xxRegister(SCI_HDAT0)); /* 8kHz */
    }
    if (ReceiveData(&inData)) {
      WriteVS10xxRegister(SCI_AUDATA, inData); /* 8kHz */
    }
  }
```

The following example uses only VS1002d synchronization when reading data from file and writing recorded data to file.

```
#include "vs1002_13.c"
  WriteVS10xxRegister(SCI_CLOCKF, 0x8000U|(13000/2));
  LoadUserCode();
  ActivateUserCode();
  while (1) {
    if (ReadVS10xxRegister(SCI_HDAT1)) {
      /* Read/WriteVS10xxRegister() is slower than
         7.5us, thus no delay needed */
      WriteFile(ReadVS10xxRegister(SCI_HDAT0));
    }
    if (ReadVS10xxRegister(SCI_DECODE_TIME) < 10) {
      WriteVS10xxRegister(SCI_AUDATA, ReadFile());
    }
  }
```