APPLICATION NOTE

VS1053 audio i/o

Revision History			
Rev.	**Date**	**Author**	**Description**
1.0	2008-06-03	ToV	Initial version
1.01	2008-09-23	ToV	mem_desc updated

VS1053 Introduction

VS1053 is an efficient audio codec chip with many audio decoders already in program ROM. Stereo line-in and stereo DAC together with 16 KiB (4096 words) of program RAM make this chip a highly versatile DSP platform for other audio solutions as well.

This document presents how to use audio input and audio output of the processor. There is a c-code example included. It demonstrates how to implement an audio processing algorithm on VS1053 evaluation board.

Getting started - Hardware

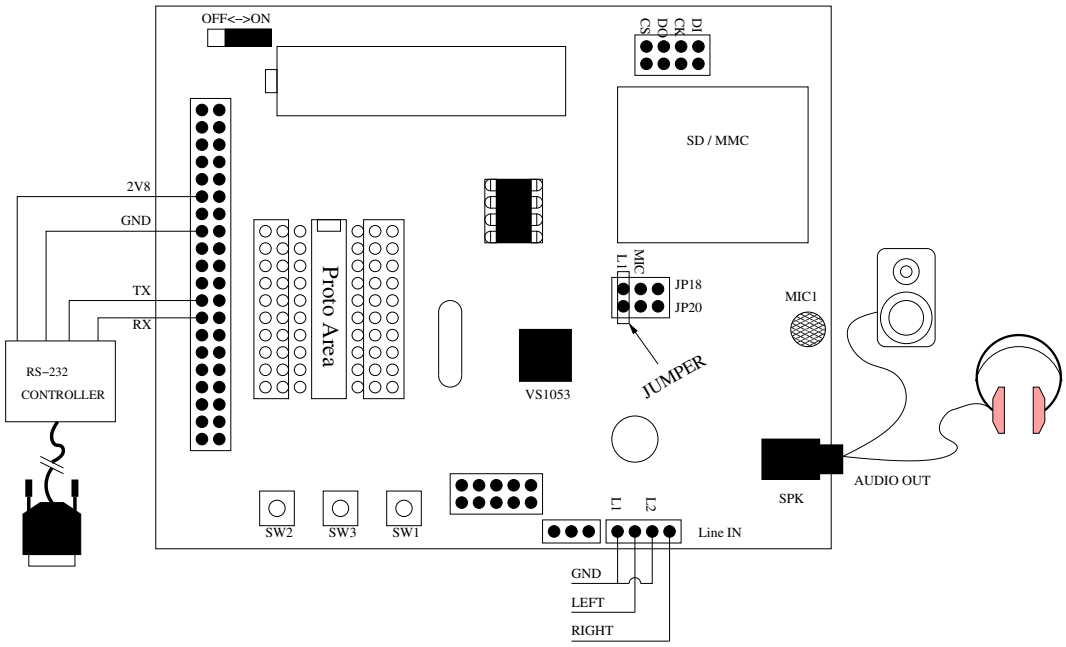

Figure 1: VS1053 proto board connections

For implementing own algorithms on VS1053 the prototyping board and programming tools are required.

The prototyping board has a serial port interface for communication with PC. There are also stereo line-in pins and a headspeaker connector on the board. To select a line-in connector instead of microphone input as a left channel analog input, the jumper between `JP18/JP20` must be placed on the `L1` position. The right channel line-in pin is always in line-in mode. The line-in pins are marked `Line IN` on the board.

Getting started - Software

For compiling and linking the code, some software tools are needed. The package vskit120.zip should contain all essential software. The software package can be downloaded from VLSI Solution's www-site:

http://www.vlsi.fi/fileadmin/software/VS10XX/vskit120.zip

The contents of the vskit120 should be as follows:

```
vskit120 -
         |
          -CHANGES.txt
         |
          -README.txt
         |
          -UNPACK.txt
         |
          -bin--
         |      |
         |     make.exe
         |     mkabs.exe
         |     vcc.exe
         |     vs3emu.exe
         |     vsa.exe
         |     .  .  .
         |
          -config
         |      |
         |     hw_desc
         |     mem_desc
         |     mem_desc.sim
         |
          -libc16_v4
         |      |
         |     assert.h
         |     ctype.h
         |       .  .  .
         |
          -libc16
         |
          -pdf--
         |      |
         |     vsdsp2_um.pdf
         |     vs_tools.pdf
         |
          -src--
               |
              hw_desc
              Makefile
              mem_desc
              .  .  .
```

There are two configuration files needed for compiling and linking code: memory description file `mem_desc` and hardware configuration file `hw_desc`.

The mem_desc file defines the memory addresses which can be used for the code (instruction and data RAM). If there is an error in memory description file, the linker may try to put code for example to the program or data ROM area. This results as a code execution failure. The following `mem_desc` file is recommended to be used:

```
MEMORY {
page 0:
        #Program RAM
        modu_int: origin = 0x23, length = 1
        ram_prog:  origin = 0x0050, length = 0x0fd0
page 1:
        #Data X RAM
        data_x:   origin =  0x0800, length = 0x1000
page 2:
        #Data Y RAM
        data_y:   origin =  0x1000, length = 0x0800
        extra_y:  origin = 0xeb00, length = 0x1500
}
```

The configuration file `hw_desc` defines configuration parameters for the VS1053 i.e. data and program word length in bits. The correct `hw_desc` file should be in src directory in vskit120 package.

All essential binaries (programs) are in `bin` directory of the vskit120. It would be good idea to put the `bin` directory to the Windows PATH. The directory `libc16_v4` is VSDSP programming library for DSP core 4. This directory should be in the working directory.

Audio input

VS1053 has stereo line-in. The right channel is always in line-in mode. The left channel is default as a microphone input but can be turned to line-in mode.

To turn the left channel to the mic-in mode the SCI_MODE register value is modified as follows

```
USEX(SCI_MODE) &= ~(1<<SCIMB_LINE); /*left=mic-in, right=line-in*/
```

To turn the left channel to the line-in mode the SCI_MODE register value is modified as follows

```
USEX(SCI_MODE) |= (1<<SCIMB_LINE); /*left=mic-in, right=line-in*/
```

Default mode for the left channel input is line-in.

AD sample rate is adjusted by writing to the DECIM_CONTROL register. Sample rates 192 kHz, 96 kHz, 48 kHz, 24 kHz can be selected. Use defines DECIM_FACTOR192K, DECIM_FACTOR96K, DECIM_FACTOR48K, DECIM_FACTOR24K according to desired sample rate.

```
USEX(DECIM_CONTROL) = DECIM_ENABLE | DECIM_FACTOR48K; /*AD 48 kHz*/
```

In addition, the AD clock speed can be halved resulting in previous sample rates to be divided by two. This makes also 12 kHz AD sample rate possible.

```
USEX(SCI_STATUS) |= (1<<SCIST_AD_CLOCK); /*AD clock 6MHz -> 3MHz */
```

The AD interrupt controller uses `stream_wr_pointer` to write input data to `stream_buffer`. It is 1024 words long 16-bit buffer. The ROM function `MyGetCPairs()` can be used to read input samples:

```
MyGetCPairs(lineInWrPtr, 2); //Read left and right input sample
```

The first parameter of the funtion is a pointer to the array where to write the samples. The second parameter is a number of samples to be read.

The ROM function `StreamDiff()` returns the number of samples in `stream_buffer`. As many 16-bit samples can be read from `stream_buffer`. If the samples are not read fast enough, the buffer overflows and returns to empty state. Highest possible value is 1023 (buffer full).

Audio output

VS1053 has 18-bit stereo DAC. DAC samplerate can be adjusted by ROM function:

```
SetHardware(2, 48000U); // Stereo, 48 kHz
```

Parameters for the function are number of output channels (1 or 2) and output sample rate in Hz. Fine tuning for the DAC sample rate can be done with a 32-bit number. The lower 16 bits are written to the register `FREQCTLL` and higher 4 bits to the register `FREQCTLH`.

The sample rate can be counted from FREQCTL with the following formula:

$$f_s = (FREQCTLH * 65536 + FREQCTLL) * 2^{27} * XTALI \quad (1)$$

When input clock `XTALI` is 12.288 MHz, adding 1 to FREQCTLL increases the sample rate by ≈0.0916 Hz.

```
u_int32 baseFctl = (USEX(FREQCTLH)<<16) | USEX(FREQCTLL);
baseFctl += 10000; //increase DAC sample rate by 915.54 Hz
USEX(FREQCTLH) = (u_int16)(baseFctl >> 16);
USEX(FREQCTLL) = (u_int16)baseFctl;
```

For writing audio samples to DAC buffer `audio_buffer`, the ROM function `WmaStereoCopy()` is used:

```
WmaStereoCopy(audioSamp, 1); //Write one stereo sample pair
```

The first parameter is a pointer to the samples to be written. The second parameter is a number of stereo sample pairs to be written. To check, how many samples there are in `audio_buffer` ready for DAC, the ROM function `AudioBufFill()` can be used. The function returns the number of samples in audio buffer.

Note: WmaStereoCopy() uses sample pairs, but AudioBufFill() samples!

Note: Audio buffer size of VS1053 is 4096 samples!

Compiler, assembler, linker and emulator

Note: Files c.s and rom1053.txt can be downloaded from www.vlsi.fi

There are two additional object files required for linking `c.o` and `rom1053.o`. The file `c.o` contains a jump to the funtion *main()*. The jump code is in fixed address 0x50 where the VS1053 starts the execution of the code. The file `c.s` contains also function `mymodu_int`.

The file `c.s` is compiled to the object file `c.o` with the following command:

```
vsa.exe -o c.o c.s
```

The file `rom1053.o` contains ROM symbols of VS1053 chip. The ROM symbols are generated to the object file as follows:

```
mkabs.exe -o rom1053.o -f rom1053.txt
```

The c-code itself is compiled with VLSI c-compiler vcc.exe

```
vcc.exe -P130 -g -O -fsmall-code -Ilibc16_v4
                                -I. audio.o audio.c
```

The warning 130 is promoted into an error (-P130). The option -g adds symbol and line number information, -O means optimization of the code, -fsmall-code means 16-bit code space (65536 address limit), -I means included directories where the compiler searches included header files (VLSI 16-bit library and working directory included). After this compiling there should be object file audio.o in the current working directory. For further information about VLSI c-compiler options refer to the manual `vs_tools.pdf` chapters 2 and 10. The document is in the vskit120 package.

Object files are linked into the executable binary code for the VS1053 by VLSI linker vslink.exe

```
vslink.exe -m mem_desc -k c.o audio.o rom1053.o
                    -o audio.bin -L ./ -Llibc16_v4/ -lc
```

The object files are linked with emulation link library `libc.a` (-lc). The current working directory and directory libc16_v4 are included to the library search path. The memory description file (-m mem_desc) is used for linking. For further information of vslink.exe refer to the chapters 5 and 10 in `vs_tools.pdf`.

After linking there should be a binary file `audio.bin` in working directory.
To emulate the binary code on VS1053 prototyping board, connect the serial cable to the board, turn on the power and type the following command:

```
vs3emu.exe -p 1 -chip vs1002b -s 9600 -e 0x50 -x 12288
                 -ts 115200 -l audio.bin -m mem_desc
```

The first parameter is a serial port number of the PC (-p 1). The VLSI chip type for VS1053 is vs1002b (-chip vs1002b). Serial port start speed is set to 9600 bits per second (-s) and target speed is set to 115200 bps (-ts). The emulation start address (where the processor starts to execute code) is 0x50 (-e). The chrystal speed is 12.288 MHz (-x 12288). The binary file `audio.bin` is loaded (-l) and memory map mem_desc is used (-m).

Emulation is started with emulator command “e”. Debug prints can be put inside the c-code to debug the code. The VS1053 prototyping board should be reseted before the emulator loads the code into the VS1053. Otherwise the emulator cannot connect to the chip.

Now the audio fed into line-in should be heard from speaker connector.

Easier way - Makefile

To ease the programming of VS1053 prototyping board, the Makefile can be used. The Makefile is a text file that is input for make.exe program. In Makefile there are rules how to build a binary file for VS1053. When the Makefile is configured right, the only command the user must give is simply "make". There are dependencies on the Makefile so, that every time the programmer changes i.e. source code and saves the c-file, the make.exe notices this change and recompiles the source code and links the binary. The Makefile for doing all the commands teached in previous chapter would be something like this:

Example Makefile

```
VCC = vcc.exe
VCCFLAGS = -P130 -g -O -fsmall-code
INCDIR = -Ilibc16_v4 -I.
LIBDIR = -Llibc16_v4/ -L./
OBJS = c.o audio.o  rom1053b.o

all: audio.bin

rom1053b.o: rom1053b.txt
        mkabs -o $@ -f $<

.c.o:
        $(VCC) $(VCCFLAGS) $(INCDIR) -o $@ $<

.s.o:
        vsa -D ASM $(INCDIR) -o $@ $<
c.o: c.s

audio.o: audio.c

audio.bin: $(OBJS)
        vslink -m mem_desc -k $(OBJS) -o $@ $(LIBDIR) -lc

clean:
        del -f *.o *.coff *~ \#* audio.a
```

In the beginning of the `Makefile` there are variable initializations. When the variable is used it is preceded by dollar sign and the variable name is bracketed.

Bare make command does whatever is after "all:". In this case the target is `audio.bin` file. There can be more than one target. The target to be build can be given as a

parameter for make.exe. For example "make rom1053b.o" builds only the object file `rom1053b.o`. The rule for building `.o` files from `.c` files is

```
$(VCC) $(VCCFLAGS) $(INCDIR) -o $@ $<
```

which means

```
vcc.exe -P130 -g -O -fsmall-code -Ilibc16_v4 -I. -o *.o *.c
```

With "make clean" all temporary files can be deleted in the current working directory.

Example c-code

Our c-code example uses 48 kHz input (decimator) sample rate and 48 kHz DAC sample rate. Audio stream is read to the `lineInBuf` ring buffer. In while loop the processing mode can be selected. There is either direct loopback from AD to DA or batch conversion of zeroing lower byte of 16-bit samples.

```
/*=======================================================*/
/*==================== audio.c ==========================*/
/*=======================================================*/

//#include <stdio.h>
#include <string.h>
//#include <stdlib.h>
#include <vstypes.h>

#define BLOCKSIZE 64 //Number of stereo samples processed in one loop round

#define SCI_MODE           0xC000
#define SCI_STATUS         0xC001
#define SCI_CLOCKF         0xC003
#define SCI_AICTRL0        0xC00C
#define SCI_AICTRL1        0xC00D
#define SCI_AICTRL2        0xC00E
#define SCI_AICTRL3        0xC00F
#define DECIM_FACTOR24K         6 // 24 kHz f256
#define DECIM_FACTOR48K         4 // 48 kHz f128
#define DECIM_FACTOR96K         2 // 96 kHz f64
#define DECIM_FACTOR192K        0 //192 kHz f32
#define DECIM_ENABLE            1
#define DECIM_CONTROL      0xc042
#define SCIST_APDOWN1           2 /* Analog internal */
#define SCIST_APDOWN2           3 /* Driver */
#define SCIMB_LINE             14
#define INT_ENABLE         0xC01A /* RW */
#define INT_EN_DAC              0
#define INT_EN_MODU             3
#define AUDIO_BUFFER_SZ      4096
```

```
#define STREAM_BUFFER_SZ     1024

auto void SetClockFromSpi(void);
auto void SetHardware(register s_int16 channels, register u_int16 rate);
auto s_int16 WmaStereoCopy(s_int16 *buf, s_int16 cnt);
auto void MyGetCPairs(register __i0 __y u_int16 *p, register __a0 s_int16 n);
extern s_int16 (*applAddr)(s_int16 register __i0 **d,
                           s_int16 register __a1 mode,
                           s_int16 register __a0 n);
void Disable(void);        /* NOT auto */
void Enable(void);        /* NOT auto */
extern u_int16 __y uartByteSpeed;
auto s_int16 StreamDiff(void);
auto s_int16 AudioBufFill(void);
extern s_int16 __y audio_buffer[];
volatile extern s_int16 __y * audio_wr_pointer;
volatile extern s_int16 __y * audio_rd_pointer;
extern u_int16 stream_buffer[];
volatile extern u_int16 __x * __y stream_wr_pointer;
volatile extern u_int16 __x * __y stream_rd_pointer;

/*
  Called before HALT, i.e.  when there is nothing else to do.
  Either there is not enough data in stream buffer, or
  the audio buffer is full.
*/
void UserHook(void) {
}

void main(void) {
  s_int16 audioSamp[2*BLOCKSIZE];
  s_int16 __y lineInBuf[2*BLOCKSIZE];

  s_int16 i = 0;
  Disable(); //Don't allow interrupts yet

    /* zero applAddr so we are not called again */
    applAddr = 0;

    /* must disable analog powerdown if it isn't already!  */
    USEX(SCI_STATUS) &= ~((1<<SCIST_APDOWN1) | (1<<SCIST_APDOWN2));


    /*=========================================*/
    /*= Turn left analog input to line-in mode =*/
    /*=========================================*/
#if 1
    USEX(SCI_MODE) |= (1<<SCIMB_LINE); /*left=line-in, right=line-in*/
#endif

    /*=================================================*/
    /*= Set uart speed for serial port communication =*/
    /*=================================================*/
    uartByteSpeed = 11520U;
```

```
    /*======================*/
    /*= Set DSP core clock =*/
    /*======================*/
    USEX(SCI_CLOCKF) = 0x6000; /* 3.0x 12.288MHz */
    //USEX(SCI_CLOCKF) = 0x8000; /* 3.5x 12.288MHz */
    //USEX(SCI_CLOCKF) = 0xa000; /* 4.0x 12.288MHz */
    //USEX(SCI_CLOCKF) = 0xc000; /* 4.5x 12.288MHz */
    //USEX(SCI_CLOCKF) = 0xe000; /* 5.0x 12.288MHz */
    SetClockFromSpi();

    USEX(SCI_AICTRL1) = 1024; /*autogain==0, 1x==1024, 0.5x==512 etc.*/
    USEX(SCI_AICTRL3) = 0x0004; /*Joint stereo (common gain).  Linear pcm*/

    //adjust AD sample rate
    USEX(DECIM_CONTROL) = DECIM_ENABLE | DECIM_FACTOR48K; /*AD sample rate is 48 kHz*/

#define HALVE_AD_CLOCK 0 //Makes 12 kHz AD sample rate possible (24 kHz/2)
#if HALVE_AD_CLOCK
    USEX(SCI_STATUS) |= (1<<SCIST_AD_CLOCK); /*Halves AD clock 6MHz -> 3MHz */
#endif

    SetHardware(2, 48000U); /*DA sample rate in Hz*/

    /* 'Empty' stream buffer*/
    stream_rd_pointer = stream_wr_pointer = (u_int16 __x *)&stream_buffer[0];
    USEX(INT_ENABLE) |= (1<<INT_EN_MODU)/*AD*/ | (1<<INT_EN_DAC);

    /* Set audio_buffer read and write pointers */
    audio_rd_pointer = audio_buffer;
    audio_wr_pointer = audio_buffer + 2*BLOCKSIZE;

    //Write zero to audio buffer.  Prevent unwanted samples to be played.
    memsetY(audio_buffer, 0, AUDIO_BUFFER_SZ);

    Enable(); //allow interrupts (DAC, AD,...)

    while (1) { //Main loop
#if 1
    USEX(SCI_MODE) |= (1<<SCIMB_LINE); /*left=line-in, right=line-in*/
#endif
      if(StreamDiff() > BLOCKSIZE*2) {
        /*Read input samples*/
        MyGetCPairs((__y u_int16 *)lineInBuf, 2*BLOCKSIZE);
#if 0 //do some sample processing
        {
          int i;
          __y s_int16 *lp = lineInBuf;
          s_int16 *sp = audioSamp;
          for(i=0;i<BLOCKSIZE*2;i++) {
            *sp++ = *lp++ & 0xff00U; //turn LSB to zero
          }
        }
#else //just copy samples (loopback)
        /*Copy samples from __y buffer to __x buffer*/
        memcpyYX(audioSamp, lineInBuf, sizeof(audioSamp));
```

```
#endif

        /*Write ouput sample pairs*/
        WmaStereoCopy(audioSamp, BLOCKSIZE);
      }

    }
}
```