Connecting VS1053/VS1063 to a Linux host

This document explains how to connect a VLSI VS1053 or VS1063 DSP to a Linux host. The VLSI VS1053/VS1063 Simple DSP Board and the Atmel AT91SAM9260 microcontroller are used in the examples to demonstrate how a system can be built that plays music.

The target audience is the software engineer who is familiar with embedded systems. Experience with (or a strong drive to learn) embedded Linux is assumed in this document.

Disclaimer

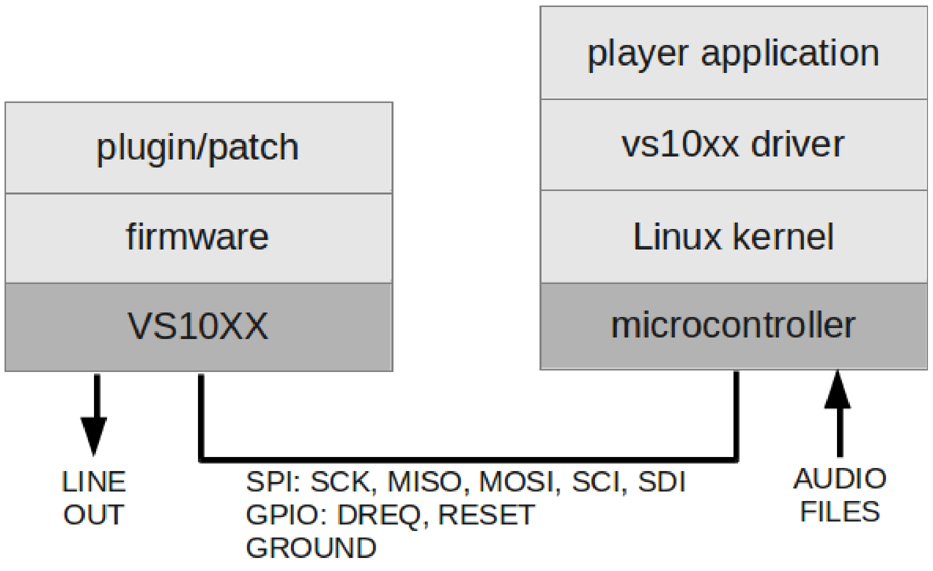
THE INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. THIS INFORMATION MUST ONLY BE APPLIED AFTER CAREFUL EVALUATION AND INTERPRETATION. MODIFICATION OF THIS INFORMATION IS VERY LIKELY NECESSARY TO SUIT CIRCUMSTANCES.

THE READER MUST UNDERSTAND THAT WORKING WITH ELECTRONICS AND HARDWARE COULD LEAD TO DAMAGE OF ANY KIND TO (BUT NOT LIMITED TO) HARDWARE, ENVIRONMENT, OR HUMANS. IT IS THE READER'S RESPONSIBILITY TO TAKE CARE OF A SAFE AND HEALTHY WORK ENVIRONMENT.

THE AUTHOR IS NOT RESPONSIBLE FOR ANY ERRORS IN THIS INFORMATION AND/OR ANY CONSEQUENCES OF APPLICATION OF THIS INFORMATION. THE READER MUST UNDERSTAND THAT THIS INFORMATION IS TO BE USED AT OWN RISK.

1 Prerequisites

The figure below outlines the components in the system that is described in this document.

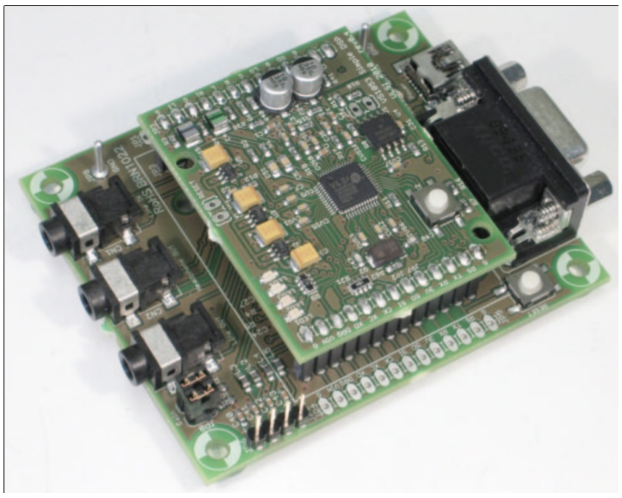


VS10XX and Microcontroller are hardware components. The other items are software components.

LINE OUT is the connection between the VS10XX DSP and an amplifier. AUDIO FILES represents any method to transfer audio into the system, e.g. MP3 from a sdcard or an Internet radio stream. The connection between V10XX and Microcontroller consist of a SPI bus, 2 x GPIO for DREQ and RESET and a common signal ground.

The components are described in more detail in the following sections.

1.1 VLSI Simple DSP board with VS1053 or VS1063



This is the VLSI Simple DSP board, mounted to a host connector board. The Simple DSP board can be used stand-alone. For development/prototyping it can conveniently be connected to a host connector board that provides connectors for line-in/out, headphones and power input.

Header pins can be soldered to the host connector board so that jumper wire can be used for the connections to a microcontroller board

Power (5V) can be applied via an USB cable, or from another source through the header pins. I/O signal levels are 3V3.

1.2 Microcontroller board capable of running Linux

There are many microcontrollers boards that are capable of running Linux. Search for example for 'ARM9' in Google Images to see the variety available.

The vs10xx driver was developed and tested for an ARM9 microcontroller (Atmel AT91SAM9260, 200Mhz). It is portable to other architectures but that is out scope of this document. When choosing a board, verify that it:

- has a microcontroller with MMU (e.g. ARM9 and up)
- exports a SPI bus with 2 free chip selects
- exports 2 free GPIO's, of which one supports interrupts
- has support for Linux and a good manual

1.3 Linux kernel

Kernel version 2.6.33.9 was used for development of the driver. Other versions could be used, but these were not tested. The Linux kernel can be downloaded from:

http://www.kernel.org

1.4 vs10xx kernel driver

The vs10xx kernel driver can be downloaded from sourceforge:

http://sourceforge.net/projects/vs10xx

The vs10xx driver is released under the GPLv2 license:

http://www.gnu.org/licenses/gpl-2.0.html

1.5 Toolchain for building an embedded environment

Some boards come with configured kernels and root file system. A toolchain is needed to build a kernel, driver and userspace tooling. Crosstools and buildroot for example are free tools that can be used.

http://crosstool-ng.org

http://buildroot.uclibc.org

Spend some time reading the documentation on these sites to become familiar with toolchains, cross compilers and root file systems.

1.6 Documentation

Read the documentation that is available for the VLSI Simple DSP board at:

http://www.vlsi.fi

Read the documentation that is available for the microcontroller and the microcontroller board to which the VLSI Simple DSP board is to be connected. The datasheet of the AT91SAM9260 microcontroller that is used as example in this document is available at:

http://www.atmel.com/Images/doc6221.pdf

Pages 25-27 provide, together with the manual of the microcontroller board provides the information that is needed to determine the connections in the table in section 2.

2 Connecting the hardware

It is assumed that the microcontroller board has been configured for development purpose and connected to a PC, i.e.:

– a debug console is available (e.g. through RS232, USB)

– a software load mechanism is in place (e.g. through serial, tftp, sdcard)

– access to the boot loader is available

The VLSI Simple DSP board could be connected to the microcontroller as shown in the table below. It lists the pins for the ATMEL AT91SAM9260 microcontroller. The manual of the microcontroller board lists which pins of the microcontroller are exported to (e.g.) header pins.

The vs10xx in this HOWTO is connected to SPI bus #1 with chip select 0 used for XCS and chip select 1 to XDCS. The AT91SAM9260 has two SPI busses with four chip selects each. Up to four vs10xx chips can be attached. The manual of the microcontroller board lists which SPI interface and GPIO pins are free to use. Note that the GPIO pin connected to DREQ should support interrupts. The vs10xx supports polling mode, but this has serious performance impact.

Simple DSP board (slave)	**Function**	**Microcontroller board (master)**
G	Ground	Connect to ground of the microcontroller board.
VIN	5V to Simple DSP board	n.c
DREQ	Data request	PB17 (GPIO)
XD (XDCS)	SDI bus select	PC5 (SPI1_NPCS1)
XC (XCS)	SCI bus select	PB3 (SPI1_NPCS0)
SC (SCLK)	SPI clock	PB2 (SPI1_SPCK)
SI	SPI data master out slave in	PB1 (SPI1_MOSI)
SO	SPI data master in slave out	PB0 (SPI1_MISO)
3V	3V3 from Simple DSP board	n.c.
RX	Console RX	n.c
TX	Console TX	n.c
RS	Reset	PB16 (GPIO)
G	Ground	Connect to ground of the microcontroller board.

The Simple DSP board can be powered though USB if it is mounted on the host connector board. If USB power is not applied, then an alternative (+5V) source must be used. Some microcontroller boards supply +5V for external use.

The connections between Simple DSP board and microcontroller board can be made with jumper wire if both boards are equipped with header pins. Beware however that long wires easily pick up interference during high speed SPI transfers. Use short cables (<10cm).

3 Building the software

3.1 Linux kernel

The chosen SPI/GPIO configuration must be added to the board support section of the Linux kernel. The vs10xx driver contains a patch that adds this support to the kernel for the example AT91SAM9260 microcontroller. It contains the following code section in *linux-2.6.33.9/arch/arm/mach-at91/board-sam9260-ek.c*:

```
/*
 * VS10XX SPI Devices
 */

struct vs10xx_board_info {
       int device_id;
       int gpio_reset;
       int gpio_dreq;
};

struct vs10xx_board_info vs10xx_device_0 = {
       .device_id  = 0,             // = /dev/vs10xx-0
       .gpio_reset = AT91_PIN_PB16, // = GPIO 80
       .gpio_dreq  = AT91_PIN_PB17, // = GPIO 81
};

#if 0
struct vs10xx_board_info vs10xx_device_1 = {
       .device_id  = 1,             // = /dev/vs10xx-1
       .gpio_reset = AT91_PIN_PB18, // = GPIO 82
       .gpio_dreq  = AT91_PIN_PB19, // = GPIO 83
};
#endif

static struct spi_board_info ek_spi_devices[] = {

       {      /* device 0 on SPI bus 1, cs 0 */
              .modalias      = "vs10xx-ctrl",
              .bus_num       = 1,
              .chip_select   = 0,
              .platform_data = &vs10xx_device_0,
              .max_speed_hz   = 1 * 1000 * 1000,
       },
       {      /* device 0 on SPI bus 1, cs 1 */
              .modalias      = "vs10xx-data",
              .bus_num       = 1,
              .chip_select   = 1,
              .platform_data = &vs10xx_device_0,
              .max_speed_hz   = 12 * 1000 * 1000,
       },
#if 0
       {      /* device 1 on SPI bus 1, cs 2 */
              .modalias      = "vs10xx-ctrl",
              .bus_num       = 1,
              .chip_select   = 2,
              .platform_data = &vs10xx_device_1,
              .max_speed_hz   = 1 * 1000 * 1000,
       },
       {      /* device 1 on SPI bus 1, cs 3 */
              .modalias      = "vs10xx-data",
              .bus_num       = 1,
              .chip_select   = 3,
              .platform_data = &vs10xx_device_1,
              .max_speed_hz   = 12 * 1000 * 1000,
       },
#endif
};
```

The data structure *vs10xx_device_0* describes the device id and GPIO pins that are to be used. Two SPI devices are defined in the data structure *ek_spi_devices*, these represent the SCI and SDI channels of the vs10xx. Both devices are best chosen on the same bus (e.g. bus 1, cs0, cs1). The communication speed can be chosen differently for SCI and SDI channels (SCI needs low speed at startup to set clockf). The *.platform_data* member of the data structure links the vs10xx device with GPIO definitions to the SPI devices. The vs10xx driver recognizes this configuration as one vs10xx chip.

This SPI configuration must be registered at the Linux kernel, this is also done in the board support section with the following code:

```
static void __init ek_board_init(void) {

        ...
        at91_add_device_spi(ek_spi_devices, ARRAY_SIZE(ek_spi_devices));
        ...
}
```

Build the kernel as usual (select the correct board when configuring the kernel) and copy the resulting kernel to the target.

That's it. An oscilloscope is useful to verify that all signals are configured correctly.

3.2 vs10xx driver

The vs10xx driver can be built out of tree. Two variables must be changed in the top level Makefile before building the driver:

```
export KROOT = /path/to/linux-2.6.33.9
export CROSS_COMPILE = /path/to//toolchain/bin/arm-unknown-linux-uclibcgnueabi-
```

These variables must point to the location where the (patched) linux kernel is build and to the location of the toolchain. Use 'make' to build the driver:

```
user@Desktop /tmp/vs10xx $ make
cd module && make  --no-print-directory C=2 all
  CHECK   /tmp/vs10xx/module/vs10xx_iocomm.c
  CC [M]  /tmp/vs10xx/module/vs10xx_iocomm.o
  CHECK   /tmp/vs10xx/module/vs10xx_queue.c
  CC [M]  /tmp/vs10xx/module/vs10xx_queue.o
  CHECK   /tmp/vs10xx/module/vs10xx_device.c
  CC [M]  /tmp/vs10xx/module/vs10xx_device.o
  CHECK   /tmp/vs10xx/module/vs10xx_main.c
  CC [M]  /tmp/vs10xx/module/vs10xx_main.o
  LD [M]  /tmp/vs10xx/module/vs10xx.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /tmp/vs10xx/module/vs10xx.mod.o
  LD [M]  /tmp/vs10xx/module/vs10xx.ko
cd ioctl && make  --no-print-directory C=2 all
/tlchn/bin/arm-unknown-linux-uclibcgnueabi-gcc -O2 -Wall -pedantic ioctl.c -o ioctl
```

The resulting kernel module *vs10xx.ko* and *ioctl* tool can be copied to the rootfs of the target.

3.3 Root file system

Building and configuration of a root file system is a topic of its own. It can be facilitated by tools like:

- buildroot (http://http://buildroot.uclibc.org/)
- openwrt (https://openwrt.org/)
- openembedded (http://http://www.openembedded.org/)
- or alike.

A minimal root file system contains busybox and a file/directory skeleton.

4 Operation

It is assumed in this section that the kernel, the root file system and the driver have been built and loaded onto the target system.

4.1 Initialization

The vs10xx driver can be inserted as follows:

```
root@board:~$ insmod vs10xx.ko
```

It provides the following messages in the syslog:

```
vs10xx_spi_ctrl_probe: id:0 ctrl spi:1.0 gpio_reset:80 gpio_dreq:81
vs10xx_spi_data_probe: id:0 data spi:1.1 gpio_reset:80 gpio_dreq:81
vs10xx_io_init: id:0 gpio_reset:80 gpio_dreq:81 irq_dreq:81
vs10xx_queue_alloc: id:0 allocated 2048/2048 buffers (73728 bytes)
vs10xx_device_reset: id:0 found vs1063 device
vs10xx_device_plugin: id:0 plugin loaded (4512/4512 bytes)
vs10xx_init: id:0 name:vs10xx-0 maj:253 min:0
```

The term “id:0” matches the device_id that was defined in the board support section. Similar, the gpio's are reported. The driver reports which vs10xx chip was detected (VS1053 and VS1063 are supported). It also loads the main patch on startup.

A character device is created (permissions were changed by mdev):

```
admin@board:~$ ls -al /dev/vs10xx-*
crw-rw-rw-    1 root     root      253,   0 Jan 20 12:00 /dev/vs10xx-0
```

Multiple VS10XX devices will show up as */dev/vs10xx-1*, */dev/vs10xx-2*, and so on.

An entry in sysfs is created:

```
admin@board:~$ ls -al /sys/class/vs10xx
drwxr-xr-x    2 root     root             0 Jan 20 12:00 .
drwxr-xr-x   19 root     root             0 Jan 20 12:00 ..
lrwxrwxrwx    1 root     root             0 Jan 20 12:00 vs10xx-0

admin@board:~$ ls -al /sys/class/vs10xx/vs10xx-0/
drwxr-xr-x    2 root     root             0 Jan 20 12:00 .
drwxr-xr-x    3 root     root             0 Jan 20 12:00 ..
-r--r--r--    1 root     root          4096 Jan 20 12:00 dev
--w--w--w-    1 root     root          4096 Jan 20 12:00 reset
-r--r--r--    1 root     root          4096 Jan 20 12:00 status
lrwxrwxrwx    1 root     root             0 Jan 20 12:00 subsystem
-rw-rw-rw-    1 root     root          4096 Jan 20 12:00 test
-rw-r--r--    1 root     root          4096 Jan 20 12:00 uevent
```

The sysfs entry *reset* can be used to instruct the driver to reset the board:

```
admin@board:~$ echo 1 > reset
```

to list the *status*:

```
admin@board:~$ cat status
xversion: 6
playstat: P
strmtype: 6
dreq/rdy: 0
underrun: 0
```

or to perform a *test* (must reset after running a test):

```
admin@board:~$ echo memtest > test
admin@board:~$ echo sinetest > test
admin@board:~$ echo 1 > reset
```

Module parameters are available to change the behavior of the driver. The driver should work fine using the defaults, but the following can be configured when necessary:

```
debug          sets debug information level (0..3)
queuelen       sets internal buffer size (x32 bytes)
irqmode        sets irq mode (0=disable, 1=enable)
skipdreq       sets dreq honoring (0=ignore, 1=honor)
hwreset        sets reset method (0=sw only, 1=hw+sw)
clockf         sets initial clockf value
plugin         sets plugin to load (0=none, 1=built-in)
wclose         sets wait on close time in msecs (0=do not wait)
```

The tool *ioctl* that is bundled with the driver operates on the vs10xx devices. It can be used to change SCI registers:

```
admin@board:~$ ioctl --help
Usage: ioctl [options] [command] [parms ...]

Options:
  -h --help
  -d --device  device

Commands:      Parameters:
  reset
  getscireg    regno
  setscireg    regno msb lsb
  getclockf
  setclockf    mul add clk
  getvolume
  setvolume    left right
  gettone
  settone      tb tl bb bl
  getinfo
```

For example, to set/get the volume:

```
admin@board:~$ ioctl --device /dev/vs10xx-0 setvolume 200 200
admin@board:~$ ioctl --device /dev/vs10xx-0 getvolume
volume: left:200 rght:200
```

4.2 Playing music

Playing music through the vs10xx can be done as follows:

```
admin@board:~$ cat test.mp3 > /dev/vs10xx-0
admin@board:~$ cat test.flac > /dev/vs10xx-0
```

All formats that the VS1053/VS1063 support can be played this way. Music should start playing immediately. The status of the driver shows the play status through sysfs *status* entry in the playstat field (P=playing, S=stopped, F=finishing).

Since */dev/vs10xx-n* is a character device, any tool can be used that dumps output to a file. For example, the tool *mpg123* can be used to play streaming audio from a radio station on the Internet as follows:

```
admin@board:/tmp# mpg123 -q -w /dev/vs10xx-0 http://icecast.omroep.nl:80/3fm-bb-mp3
```

The streaming audio will be decoded to WAV by *mpg123* and dumped to the */dev/vs10xx-0* device. This is of course inefficient, other tools are available or can be created to dump a stream directly to the device.

4.3 Performance

On a AT91SAM9260 200MHz board, playing MP3 takes 1-5% CPU load. FLAC takes 8-10% and WAV takes 15-20%. The latter two are more resource hungry because of their higher bit rate, the relatively small buffer size of the VS1053/VS1063 and the fact that the driver transfers data in chunks of 32 bytes. Resource usage will roughly scale with the number of VS1053/VS1063 devices connected.