

VS1063 STANDALONE PLAYER

VSMPG “VLSI Solution Audio Decoder”

Project Code:
Project Name: VSMPG

All information in this document is provided as-is without warranty. Features are subject to change without notice.

Revision History			
Rev.	Date	Author	Description
1.22	2012-06-11	HH	Documentation update.
1.21	2011-11-15	PO	Card change detected, cleanup of code etc.
1.20	2011-10-03	PO	Integrated the mp3 encoding patch.
1.19	2011-09-16	PO	Recorder version, VS1063-specific.
1.18	2009-10-27	PO	VS1053-specific version.

Contents

VS1063 Standalone Player Front Page	1
Table of Contents	2
1 VS1063 Standalone Player	3
2 Boot EEPROM and MMC	5
3 Player with Three-Button UI	7
3.1 Five-Button User Interface	8
3.2 Boot Images	9
3.3 Power-on Defaults	9
4 Standalone Recorder	10
5 SCI-Controlled Player	13
5.1 UART Control	17
5.2 Reading the 8.3-character Filename	18
5.3 Bypass Mode	19
6 Schematics	20
7 Playing Order	21
8 Document Version Changes	23
9 Contact Information	24

List of Figures

1 VS1063 Prototyping Board	3
2 SPI-Boot and MMC connection	5
3 Five-button interface connection	8
4 Standalone Recorder configured for mono microphone input	10
5 Standalone Recorder configured for stereo line input (cable not included)	11
6 SCI connection	13
7 Example of shared access	19
8 VS1053 Standalone Player Schematics	20
9 Play order with subdirectories	21
10 Play order with nested subdirectories	22

1 VS1063 Standalone Player

All information in this document is provided as-is without warranty. Features are subject to change without notice.

The SPI bootloader that is available in VS10XX chips can be used to add new features to the system. Patch codes and new codecs can be automatically loaded from SPI EEPROM at startup. One interesting application is a single-chip standalone player / recorder.

The standalone player application uses MMC/SD directly connected to VS1063 using the same GPIO pins that are used to download the player software from the boot EEPROM.

The increased instruction RAM of 4096 words (16 kilobytes) in VS1053 and VS1063 is used for MMC communication routines, handling of the FAT and FAT32 filesystems, upto a five-button user interface, and recording features.

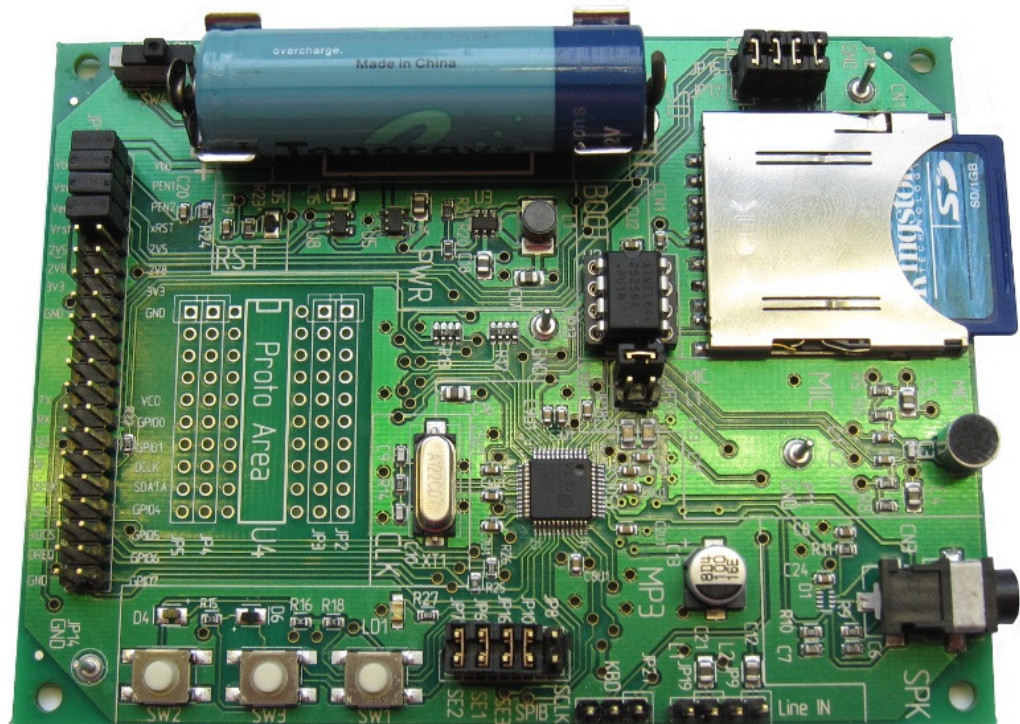


Figure 1: VS1063 Prototyping Board

The VS1063 standalone player/recorder has been developed to run on the VS1063 Prototyping Board. By default VLSI Solution's web store ships the board as player / recorder, recording mono sound from microphone, saved to card as 128 kbps mp3.

Note: 32 kB EEPROM 25LC256 is required because the recorder takes more than 8 kB.

Standalone Features:

- **No microcontroller is required**, boots from SPI EEPROM (25LC256).
- Low-power operation
- Uses MMC/SD/SDHC for storage. Hot-removal and insertion of card is supported. (But do not do that during recording!)
- Supports FAT and FAT32 filesystems, **including subdirectories** (upto 16 levels). FAT12 is partially supported: subdirectories or fragmented files are not allowed.
- Automatically starts playing from the first file after power-on.
- Power-on defaults are configurable.
- VS1063A transfer speed 4.8 Mbit/s (3.5×12.288 MHz clock).
- High transfer speed supports even 48 kHz 16-bit stereo WAV files.
- Default 3-button interface allows pause/play, song selection, volume control, and start of recording.
- Optional five-button interface allows pause/play, shuffle play and loudness toggle, song selection, volume control, and recording.
- Recording in MP3, Ogg Vorbis, or several WAV formats by recompilation of source code.
- LED for user interface feedback

With Optional Microcontroller:

- External microcontroller can control the player through SCI or UART.
- Code can be loaded through SCI by a microcontroller to eliminate SPI EEPROM.

2 Boot EEPROM and MMC

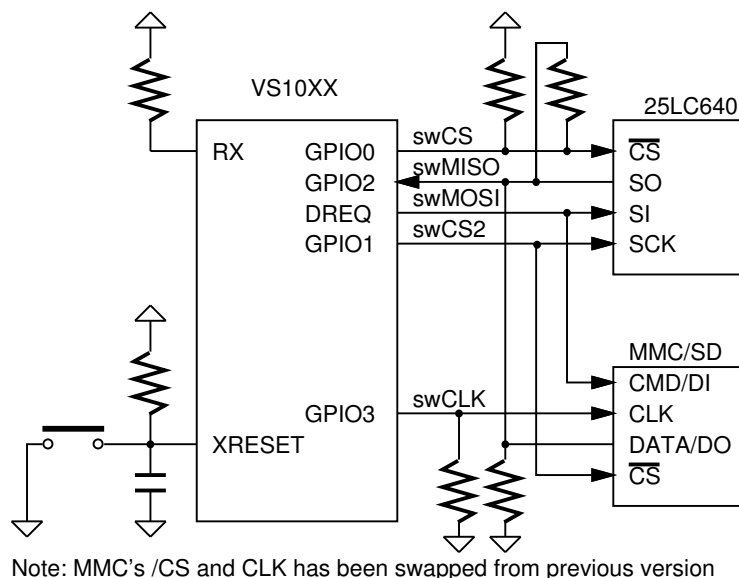


Figure 2: SPI-Boot and MMC connection

The standalone player software is loaded from SPI eeprom at power-up or reset when GPIO0 is tied high with a pull-up resistor. The memory has to be an SPI Bus Serial EEPROM with 16-bit addresses. Currently the standalone player without recording features takes less than 8 kB. The recorder code requires more than 11 kB, thus a 32 kB SPI EEPROM (25LC256) is recommended.

SPI boot and MMC/SD usage redefines the following pins:

Pin	SPI Boot	Other
GPIO0	swCS (EEPROM XCS)	100 k Ω pull-up resistor
GPIO1	swCS2 (MMC XCS)	Also used as SPI clock during boot
DREQ	swMOSI	
GPIO2	swMISO	100 k Ω between xSPI & swMISO, 680 k Ω to GND
GPIO3	swCLK (MMC CLK)	Data clock for MMC, 10 M Ω to GND

Pull-down resistors on GPIO2 and GPIO3 keep the MMC CLK and DATA in valid states on powerup.

The SPI EEPROM boot is used for the button-controlled standalone player. The code for the SCI-controlled player can be uploaded through the SCI instead of using an SPI EEPROM.

Defective or partially defective MMC cards can drive the CMD (DI) pin until they get the first clock. This interferes with the SPI boot if MMC's drive capability is higher than VS10xx's. So, **if you have powerup problems when MMC is inserted, you need something like a 330 Ω resistor between swMOSI (DREQ) and MMC's CMD/DI pin.** Normally this resistor is not required.

Because the SPI EEPROM and MMC share pins, it is crucial that MMC does not drive the pins while VS10xx is booting. MMC boots up in mmc-mode, which does not care about the chip select input, but listens to the CMD/DI pin. MMC-mode commands are protected with cyclic redundancy check codes (CRC's). Previously it was assumed that when no valid command appears in the CMD pin, the MMC does not do anything. However, it seems that some MMC's react even to commands with invalid CRC's, which messes up the SPI boot.

The only way to cure this problem was to change how the MMC is connected. The minimum changes were achieved by swapping MMC's chip select and clock inputs. This way MMC does not get clocked during the SPI boot and the system should work with all MMC's. Because the swap only occurred on the MMC pins, the SPI EEPROM connection is unchanged!

3 Player with Three-Button UI

The source code has a default preprocessor define COMPAT_KEYS in standalone.h. This allows the old three-button interface of the prototyping board to be used as-is. The DREQ to SCLK jumper (JP8) is not required with VS1063.

The three-button interface provides the most needed controls.

Button	Short Keypress	Long Keypress
SW1	Next song	Volume up
SW2	Previous song	Volume down
SW3	Pause/Play	Play mode: Toggle loudness Pause mode: Toggle shuffle play

A LED connected to DREQ can be used for indicating system activity. In play mode a long blink of the LED indicates loudness ON, in pause mode a long blink indicates shuffle play ON. Otherwise the LED shows MMC/SD activity. In pause mode the LED lights up dimly.

Notice that SCI and SDI can not be used to transfer data simultaneously with the button interface.

3.1 Five-Button User Interface

With VS1063 the number of buttons can be extended. VS1063 can read some dedicated pins directly, so four buttons can be connected to SI, xDCS, xCS, and SCLK. The fifth key is connected to GPIO4. All buttons can also be read independently so simultaneous presses of several keys can be detected.

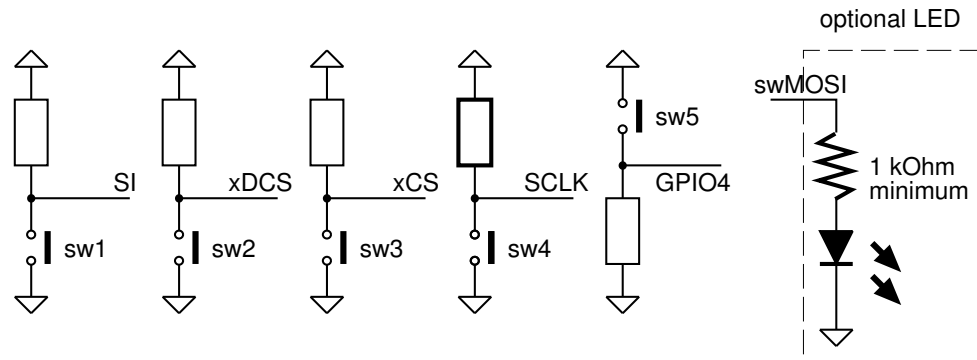


Figure 3: Five-button interface connection

GPIO5, GPIO6 and GPIO7 are also free to be used for extra buttons or LEDs. Note: GPIO6 is used in recorder to select between mono and stereo. Using GPIO4 to GPIO7 as key scan outputs, it is also possible to connect a 4×4 matrix keyboard.

SW1 and SW2 on the prototyping board can be used for SI and xDCS keys without changes. SCLK jumper (JP8) and SE3 jumper (JP16) should be removed. The prototyping board also contains a pull-up resistor for xCS and pull-down resistors for GPIO's, so only the SCLK pull-up and the SW3, SW4, and SW5 button switches need to be added.

3.2 Boot Images

The SPI EEPROM boot images can be found from the `code/` subdirectory. **Note that this application is highly chip-specific. It only works on the exact firmware versions mentioned.**

Chip	File	Features
VS1063A	player1063abut.bin	Three-button interface

3.3 Power-on Defaults

Default values are loaded from SPI EEPROM at power-on reset.

The input clock is assumed to be 12.288 MHz. If you want to use a different crystal, the SCI_CLOCKF value can be found from byte offsets 10 and 11 in the boot image. The default value is 0x8000 (3.5×12.288 MHz) for VS1063. You can reduce the power consumption a bit by lowering the default clock and allowing the clock add (see chip datasheet for details). Recording mode uses $4.5 \times$ clock.

Volume (SCI_VOL) default value is in byte offsets 26 and 27. Loudness default is in byte offsets 32 and 33 (treble and bass controls, respectively). The bass control value should be odd to make the loudness indicator LED blink work. SCI_BASS default value is in byte offsets 8 and 9.

If you want the loudness ON by default, replace bytes 8 and 9 in the image with the same values you use as the loudness default in offsets 32 and 33.

Offset	Register	Default	Meaning
8, 9	SCI_BASS	0x0000	Bass enhancer control at power-up
10, 11	SCI_CLOCKF	0x8000	Clock control
26, 27	SCI_VOL	0x2020	Power-up volume, left and right channel
28, 29	SCI_AICTRL0	0	Song number to play at power-up
32, 33	SCI_AICTRL2	0x33d9	Treble and bass control for loudness
34, 35	SCI_AICTRL3	0	Play mode & Miscellaneous configuration

4 Standalone Recorder

The Standalone Recorder makes use of the VS1063A microphone input or stereo line inputs. In addition to playing files from MMC/SD, audio from the microphone or line inputs can be written to MMC/SD in one of the VS1063A encoding formats.

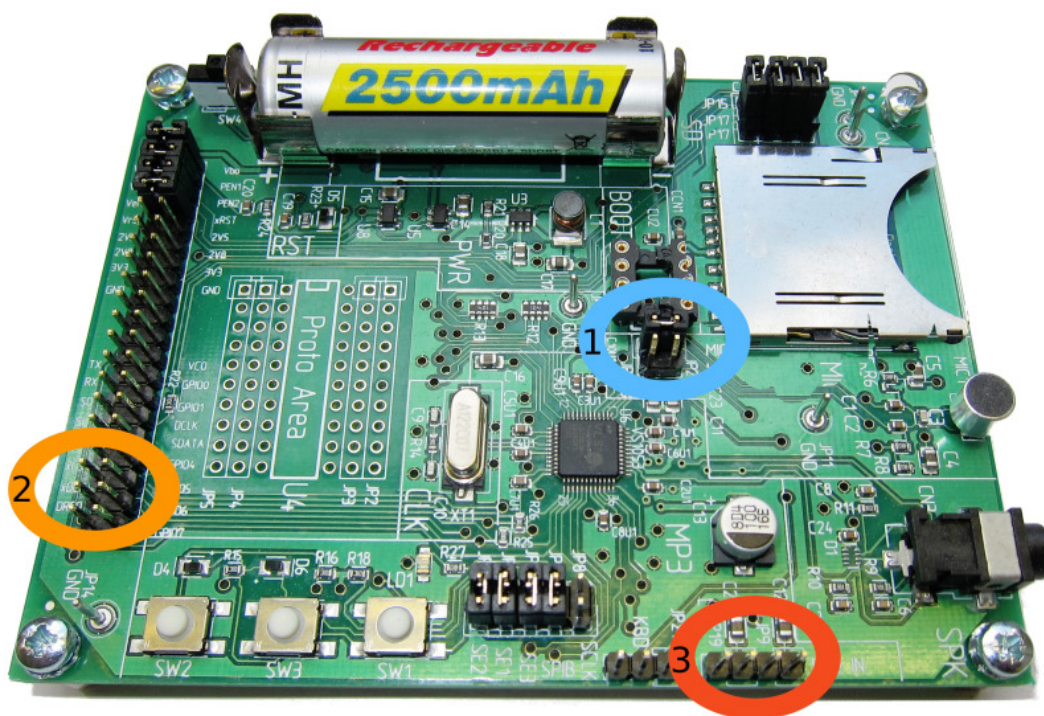


Figure 4: Standalone Recorder configured for mono microphone input

In Figure 4 the prototyping board is configured for mono 48 kHz mono 128 kbit/s MP3 recording from the microphone input. Automatic gain control with a maximum gain of $8\times$ (18 dB) is used. Microphone gain and autogain max can be edited from the boot image. To record, do the following:

- 1: Set the horizontal Mic/L1 jumper to the top position: MIC.
- 2: Remove the jumper between DREQ and GPIO6.
- 3: Don't connect anything to the Line IN pins.



Figure 5: Standalone Recorder configured for stereo line input (cable not included)

In Figure 5 the prototyping board is configured for 48 kHz stereo 160 kbit/s MP3 recording from the line input with a fixed $1.0\times$ gain. To record, do the following:

- 1: Set the horizontal Mic/L1 jumper to the bottom position: L1.
- 2: Insert a jumper between DREQ and GPIO6.
- 3: Connect line level inputs into Line IN pins.

Pin order from left to right is: Line in Left, Left GND, Line in Right, Right GND.

The recording format, sample rate, channel format and bitrate, as well as gain settings can be changed from standalone.c .

The recording mode automatically locates the free space on the MMC/SD, allocates a directory entry from the root directory, and also extends the directory if needed. Extending directory works in FAT32 only, FAT16 just fails if the root directory is full. The maximum recording time is determined by the available contiguous space.

Button	Short Keypress	Long Keypress
SW1	Next song	Volume up
SW2	Previous song	Volume down
SW3	Pause/Play	Start recording

Start of recording will take a few seconds, depending on the speed and size of the MMC/SD. Recording stops when any button is pressed shortly, or the available space becomes full. The maximum filesize created is 2'147'483'136 bytes.

Do not turn off power when recording is active or you risk corrupting the MMC. Return to play mode first.

The recording file entry is created only when recording is ended. If you turn off the unit during recording, the current recording will be lost.

The loopback audio level from ADC to DAC is slightly lowered in recording mode to prevent audio feedback.

The SPI EEPROM boot images can be found from the `code/` subdirectory. **Note that this application is highly chip-specific. It only works on the exact firmware versions mentioned.**

Chip	File	Features
VS1063A	recorder1063a.bin	Player/recorder

Power-on Defaults

Almost the same power-on defaults that the standalone player uses are available in the standalone recorder. Loudness can not be toggled, thus the loudness default in `SCI_AICTRL2` is not used, but instead, the maximum gain of the recording mode can be set using bytes in file offsets 18 and 19.

This value can be used to limit the automatic gain control of the IMA ADPCM recording. Reducing the maximum gain limits the audible noise when there is no sound.

Offset	Register	Default	Meaning
8, 9	SCI_BASS	0x0000	Bass enhancer control at power-up
10, 11	SCI_CLOCKF	0x8000	Clock control
16, 17	-	0	Record gain, 0=AGC, 512=0.5×, 1536=1.5×, etc.
18, 19	-	0x2000	Max gain, 65535=64×, 16384=16×, etc.
26, 27	SCI_VOL	0x2020	Power-up volume, left and right channel
28, 29	SCI_AICTRL0	0	Song number to play at power-up
32, 33	SCI_AICTRL2	-	Not used
34, 35	SCI_AICTRL3	0	Play mode & Miscellaneous configuration

5 SCI-Controlled Player

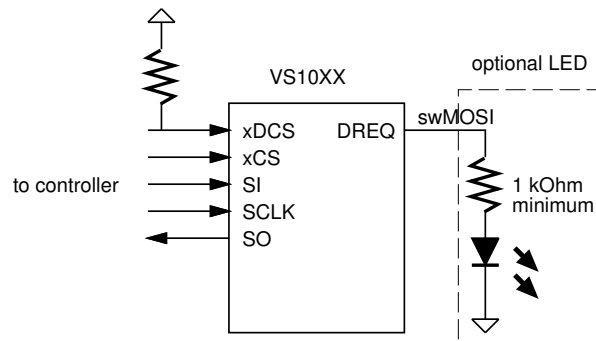


Figure 6: SCI connection

If the button interface is not used, the player can be controlled through the serial control interface (SCI). In this mode xCS, SI, SO, and SCLK are connected to the host controller's SPI bus. SCLK jumper (JP8) and SE3 jumper (JP16) should be removed. (JP6 and JP7 can also be removed.) xDCS should have a pull-up resistor, or if no other devices share the same SPI bus, the SHARED_MODE can be set in the SCI_MODE register instead.

Normally the code is loaded through SCI by the microcontroller. In this case the boot EEPROM can be eliminated, and the pull-up resistor in GPIO0 can be changed into a pull-down resistor.

Because the SCI/SDI connection is available, the VS10XX chip can be used also normally in slave mode. When standalone playing from MMC/SD is wanted, the code is loaded and started through SCI. Software or hardware reset returns the chip to slave mode.

The application loading tables for the microcontroller are available in the `code/` subdirectory. To start the application after uploading the code, write 0x50 to SCI_AIADDR (SCI register 10). Before starting the code, you should initialize SCI_CLOCKF and SCI_VOL.

Chip	File	Features
VS1063A	player1063ascii.c	SCI+UART control, watchdog

All non-application SCI registers can be used normally, except that SM_SDINEW must be kept at '1' to enable GPIO2 and GPIO3. SCI_CLOCKF must be set by the user, preferably before starting the code.

SCI_AIADDR, SCI_AICTRL0, SCI_AICTRL1, SCI_AICTRL2, and SCI_AICTRL3 are used by the player.

SCI registers		
Reg	Abbrev	Description
0x0	MODE	Mode control, SM_SDINEW=1
0x1	STATUS	Status of VS10xx
0x2	BASS	Built-in bass/treble control
0x3	CLOCKF	Clock freq + multiplier
0x4	DECODE_TIME	Decode time in seconds
0x5	AUDATA	Misc. audio data
0x6	WRAM	RAM write/read
0x7	WRAMADDR	Base address for RAM write/read
0x8	HDATA0	Stream header data 0
0x9	HDATA1	Stream header data 1
0xA	AIADDR	Player private, do not change
0xB	VOL	Volume control
0xC	AICTRL0	Current song number / Song change
0xD	AICTRL1	Number of songs on MMC
0xF	AICTRL3	Play mode

The currently playing song can be read from SCI_AICTRL0. In normal play mode the value is incremented when a file ends, and the next file is played. When the last file has been played, SCI_AICTRL0 becomes zero and playing restarts from the first file.

Write 0x8000 + song number to SCI_AICTRL0 to jump to another song. The high bit will be cleared when the song change is detected. The pause mode (CTRL3_PAUSE_ON), file ready (CTRL3_FILE_READY), and paused at end (CTRL3_AT_END) bits are automatically cleared. If the song number is too large, playing restarts from the first file. If you write to SCI_AICTRL0 before starting the code, you can directly write the song number of the first song to play.

SCI_AICTRL1 contains the number of songs (files) found from the MMC card. You can disable this feature (CTRL3_NO_NUMFILES) to speed up the start of playback. In this case AICTRL1 will contain 0x7fff after MMC/SD has been successfully initialized.

You can use SCI_WRAMADDR and SCI_WRAM to both write and read memory.

SCI_AICTRL3 bits		
Name	Bit	Description
CTRL3_AT_END	6	if PLAY_MODE=3, 1=pause at end of file
CTRL3_NO_NUMFILES	5	0=normal, 1=do not count the number of files
CTRL3_PAUSE_ON	4	0=normal, 1=pause ON
CTRL3_FILE_READY	3	1=file found
CTRL3_PLAY_MODE_MASK	2:1	0=normal, 1=loop song, 2=pause before play, 3=pause after play
CTRL3_RANDOM_PLAY	0	0=normal, 1=shuffle play

AICTRL3 should be set to the desired play mode by the user before starting the code. If it is changed during play, care must be taken.

If the lowest bit of SCI_AICTRL3 is 1, a random song is selected each time a new song starts. The shuffle play goes through all files in random order, then goes through the files in a different order. It can play a file twice in a row when new random order is initiated.

The play mode mask bits can be used to change the default play behaviour. In *normal* mode the files are played one after another. In *loop song* mode the playing file is repeated until a new file is selected. CTRL3_FILE_READY will be set to indicate a file was found and playing has started, but it will not be automatically cleared.

Pause before play mode will first locate the file, then go to pause mode. CTRL3_PAUSE_ON will get set to indicate pause mode, CTRL3_FILE_READY will be set to indicate a file was found. When the user has read the file ready indicator, he should reset the file ready bit. The user must also reset the CTRL3_PAUSE_ON bit to start playing.

One use for the *pause before play* mode is scanning the file names.

Pause after play mode will play files normally, but will go to pause mode and set the CTRL3_AT_END bit right after finishing a file. AICTRL0 will be increased to point to the next file (or the number of files if the song played was the last file), but this file is not yet ready to play. CTRL3_PAUSE_ON will get set to indicate pause mode. The user must reset the CTRL3_PAUSE_ON bit to move on to locate the next file, or select a new file by writing 0x8000 + song number to AICTRL0. CTRL3_PAUSE_ON, CTRL3_FILE_READY, and CTRL3_AT_END bits are automatically cleared when new file is selected through AICTRL0.

Pause after play and *loop mode* are only checked when the file has been fully read. *Pause before play* is checked after the file has been located, but before the actual playing starts. Take this into account if you want to change playing mode while files are playing.

You can speed up the start of playback by setting CTRL3_NO_NUMFILES. In this case the number of files on the card is not calculated. In this mode AICTRL1 will contain 0x7fff after MMC/SD has been successfully initialized. This affects the working of the shuffle mode, but the bit is useful if you implement random or shuffle play on the microcontroller. You probably want to determine the number of files on the card once to make it possible to jump from the first file to the last.

If your microcontroller does not have enough memory for the code loading tables, the SCI-controlled version can also be loaded from SPI-EEPROM. Then the SCI register default values are also loaded from EEPROM. You can change the power-on defaults in the same way than in the standalone player version.

Chip	File	Features
VS1063A	player1063ascii.bin	SCI+UART control, watchdog

If you want to use the chip in normal slave mode also with the SPI EEPROM, change the GPIO0 pull-up resistor into a pull-down resistor. This prevents automatic boot after reset, and the chip stays in normal slave mode.

To start the SCI-controlled standalone player, write 0xC017 to SCI_WRAMADDR, then 0x0001, 0x0000, and 0x0001 to SCI_WRAM. This sets GPIO0 to output a '1'. Then give a software reset. The chip now detects GPIO0 high, and performs boot from SPI EEPROM.

To return to slave mode either give a hardware reset, or write 0xC017 to SCI_WRAMADDR, then 0x0000 to SCI_WRAM, and give a software reset.

5.1 UART Control

The SCI-Controlled Player (and recorder) also supports limited control through UART at 9600bps data rate (8 data bits, no parity, 1 stop bit). When UART control is used, the code is loaded from SPI EEPROM and SCI connection is not needed.

Loading the code through UART is possible, but complicated, so that will not be available unless there is serious demand.

UART is just an alternative way to write SCI registers. You send 4-byte commands to write to SCI registers, or any register in the range of 0xc000..0xc07f. The first three bytes send 2, 7, and 7 bits of a 16-bit data value and have the most significant bit cleared. The last byte has the most significant bit set, and the register number is in the low 7 bits.

```
/* putchar() sends a 8-bit value through UART */
void WriteRegThroughUart(unsigned short value, unsigned short reg) {
    putchar((value>>14) & 127);
    putchar((value>>7) & 127);
    putchar(value & 127);
    putchar(reg | 0x80);
}
```

Example: to select song #10, send bytes 0x02, 0x00, 0x0a, 0x8c. The value to write is $(0x02 \ll 14) \mid (0x00 \ll 7) \mid 0x0a = 0x800a$, and it will be written to 0xc00c, i.e. to SCI_AICTRL0. As can be seen from the SCI control documentation, this will prompt the player to end the playing of current song and start playing song #10.

To set volume: send 0x00, 0x30, 0x10, 0x8b to set SCI_VOL to 0x1010.

If you are using some other play mode than normal play mode, or pause mode is on, you have to adapt the writes accordingly.

Byte/TX	Status
0x65 'e'	song/recording ended
0x70 'p'	song paused
0x63 'c'	play continued
0x6c 'l'	song looped
0x72 'r'	recording started

Some status information is also returned. 'e' is returned after a song or recording ends. Loop mode sends 'l' for every restart of the song. 'p' is sent when pause mode is entered, and 'c' when playing continues. This information is sent also when you use SCI control.

5.2 Reading the 8.3-character Filename

When a file has been selected, the MSDOS short filename (8+3 characters) can be read from VS10xx memory. The filename is in Y memory at addresses 0x1800..0x1805. The first character is in the most-significant bits of the first word.

The following pseudocode tries to locate a file named "SONG.MP3". If it is found, it is played continuously in a loop.

```
#define MKWORD(a,b) (((int)(unsigned char)(a)<<8)|(unsigned char)(b))
int song = 0;
WriteMp3Reg(SCI_AICTRL3, (2<<1)); /* pause before play mode */
WriteMp3Reg(SCI_AICTRL0, 0x8000+song); /* select song */
while (1) {
    if (ReadMp3Reg(SCI_AICTRL3) & (1<<3)) { /* file ready */
        unsigned short ch[6], name[6] = {
            MKWORD('S','O'), MKWORD('N','G'), MKWORD(' ',' '),
            MKWORD(' ',' '), MKWORD('M','P'), MKWORD('3','\0')};
        int i;

        WriteMp3Reg(SCI_WRAMADDR, 0x5800);
        for (i=0; i < 6; i++) { /* read filename */
            ch[i] = ReadMp3Reg(SCI_WRAM); /* first 2 chars */
            printf("%c%c", ch[i]>>8, ch[i]);
        }
        ch[5] &= 0xff00; /* mask away unused bits */
        printf("\n");
        if (!memcmp(ch, name)) { /* compare filenames */
            break; /* filename matched, leave loop */
        } else {
            /* the right file not found!! */
            if (++song == ReadMp3Reg(SCI_AICTRL1)) {
                /* The requested file was not on the card! */
            } else {
                /* clear file ready, keep pause on, pause before play mode */
                WriteMp3Reg(SCI_AICTRL3, (1<<4)|(2<<1));
                WriteMp3Reg(SCI_AICTRL0, 0x8000+song); /* select next song */
            }
        }
    }
}
/* SONG.MP3 file number is now in the variable 'song' */
/* clear file ready and pause, select loop song mode */
WriteMp3Reg(SCI_AICTRL3, (1<<1));
```

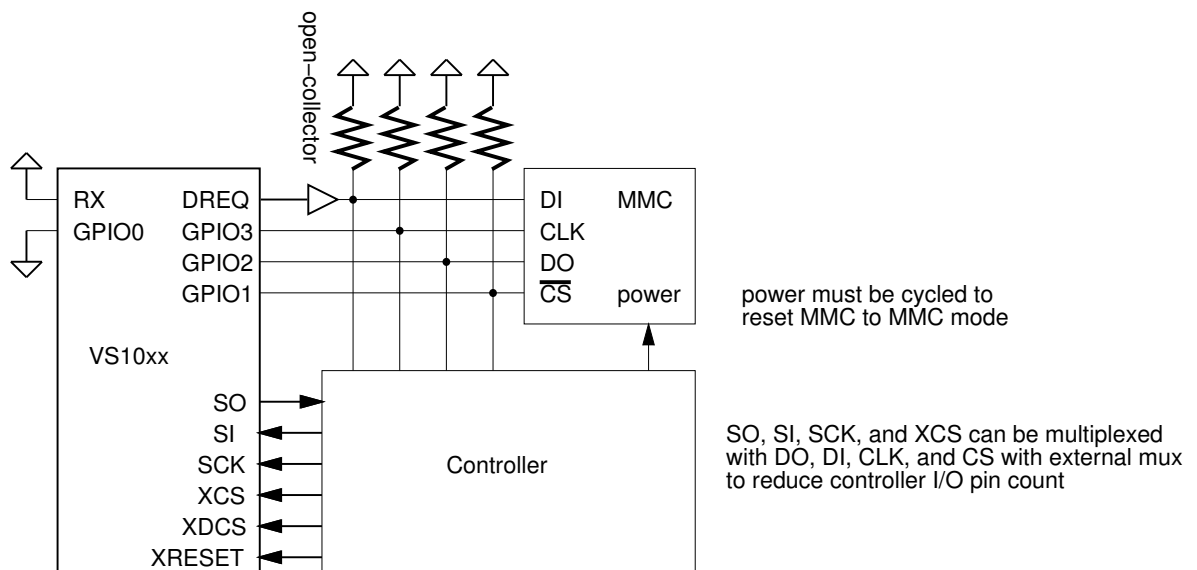
5.3 Bypass Mode

VS10xx can be disconnected from MMC to allow direct microcontroller access. A good way to disconnect VS10xx from MMC is keeping GPIO0 low when reset is deasserted (software reset can also be used). This bypasses the SPI-boot, leaving GPIO pins as inputs. SM_SDINew must be '1', this is the default in VS1053/VS1063. DREQ rises when normal firmware is ready. In this case an open-collector driver is used to connect DREQ and the controller's I/O pin to MMC's DI-pin.

Because this bypass mode is actually the normal firmware operation mode, the controller can use VS10xx through SCI and SDI normally, for example for audio cues while accessing the MMC. The controller can upload the SCI-controlled standalone player through SCI and start it whenever it wants.

Because the MMC can not be returned to MMC mode without power cycling, the controller needs a way to power off the MMC.

Concept connection diagram for SCI-controlled standalone player when code is loaded through SCI.



To start playing:

- 1) Cycle MMC power to reset it to default state
- 2) Reset VS10xx – DREQ will rise when boot complete
- 3) Upload the code from controller to VS10xx through SCI
- 4) Start the code, VS10xx accesses the MMC
- 5) The player can be controlled through SCI commands

Note: controller pins connected to MMC must be high-impedance state

To access MMC from controller:

- 1) hardware (XRESET) or software-reset (through SCI) VS10xx
- 2) DREQ rises when boot complete, GPIO's remain high-impedance
- 3) Cycle MMC power to reset it to default state
- 4) Access MMC with controller in either MMC or SPI mode

Figure 7: Example of shared access

6 Schematics

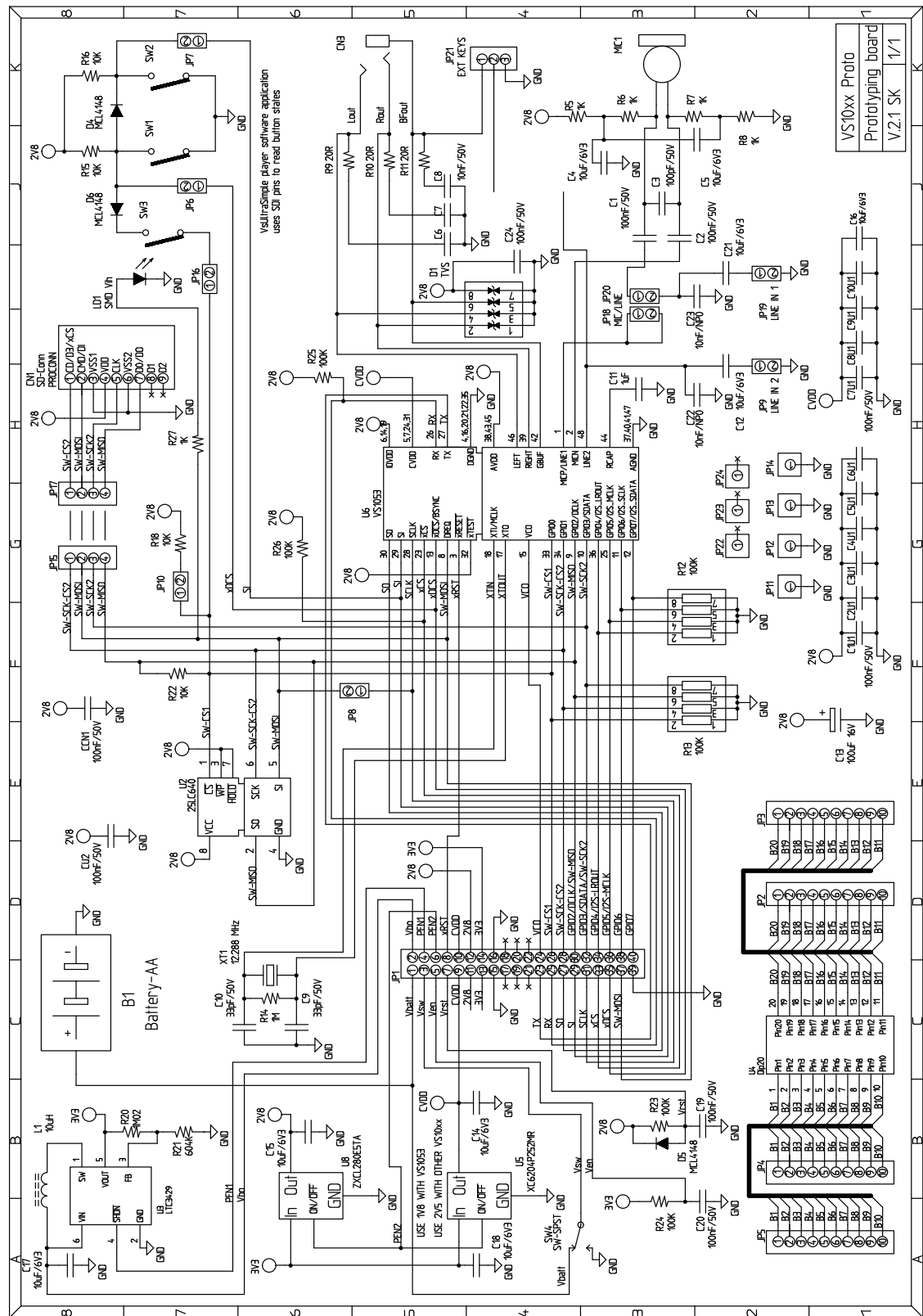


Figure 8: VS1053 Standalone Player Schematics

7 Playing Order

The playing order of files is not the same order as how they appear in Windows' file browser. The file browser sorts the entries by name and puts directories before files. It can also sort the entries by type, size or date. The standalone player does not have the resources to do that. Instead, the player handles the files and directories in the order they appear in the card's filesystem structures.

Since the 1.02 version, if the filename suffix does not match any of the valid ones for the specific chip, the file is ignored.

Normally the order of files and directories in a FAT filesystem is the order they were created. If files are deleted and new files added, this is no longer true. Also, if you copy multiple files at once, the order of those files can be anything. So, if you want a specific play order: 1) only copy files into an empty card, 2) copy files one at a time in the order you like them played.

There are also programs like LFNSORT that can reorder FAT16/FAT32 entries by different criteria. See "<http://www8.pair.com/dmurdoch/programs/lfnsort.htm>".

The following picture shows the order in which the player processes files. First DIR1 and then DIR2 has been created into an empty card, then `third.jpg` is copied, DIR3 is created and the rest of the files have been copied. `song.mid` was copied before `start.wav`, and `example.mp3` was copied before `song.mp3` because they appear in their directories first.

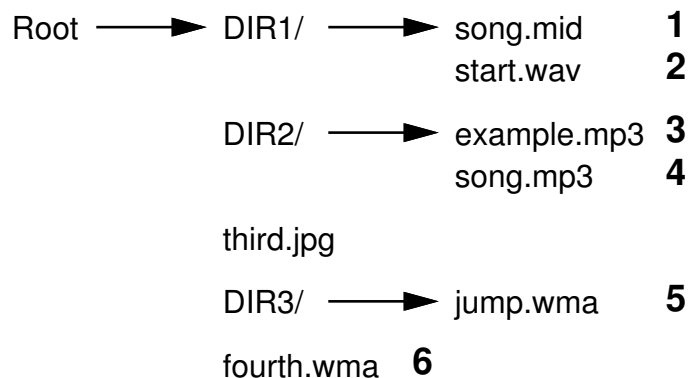


Figure 9: Play order with subdirectories

Because DIR1 appears first, all files in it are processed first, in the order they are located inside DIR1, then files in DIR2. Because `third.jpg` appears in the root directory before DIR3, it is next but ignored because the suffix does not match a supported file type, then files in DIR3, and finally the last root directory file `fourth.wma`.

If DIR2 is now moved inside DIR3, the playing order changes as follows.

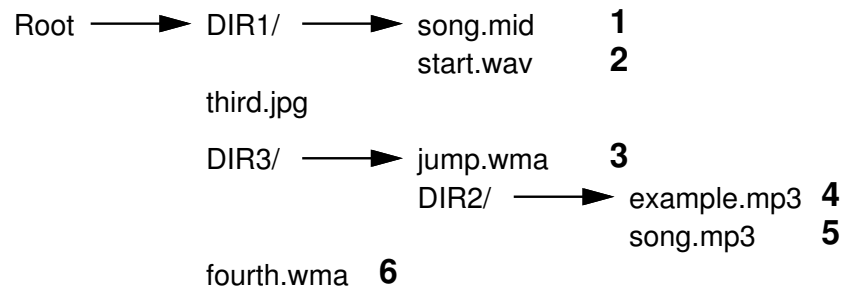


Figure 10: Play order with nested subdirectories

8 Document Version Changes

Version 1.22, 2012-06-11

- Enhanced Chapter 4, *Standalone Recorder*, and along with it Figures 4 and 5.
- Firmware has not been updated.

Version 1.21, 2011-11-15

- MP3 quantizer problem (13-16 kHz) fixed (see vs1063a-patches package)
- Free space cached correctly (has card change detection)

Version 1.20, 2011-10-03

- Integrated the MP3 encoding patch (see vs1063a-patches package)

Version 1.19, 2011-09-23

- First release with more comments about file saving added to the source code.
- VS1063 version with MP3 recording.
- Schematics updated to v2.1 (full) version.
- Various cleanups to code.

Version 1.18, 2009-10-27

- Filename read example changed to use SCI_WRAM (SCI_AICTRL2 with VS1002 only).

9 Contact Information

VLSI Solution Oy
Entrance G, 2nd floor
Hermiankatu 8
FI-33720 Tampere
FINLAND

Fax: +358-3-3140-8288
Phone: +358-3-3140-8200
Email: sales@vlsi.fi
URL: <http://www.vlsi.fi/>

For technical questions or suggestions regarding this application, please contact
support@vlsi.fi.