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
1.52	2018-10-26	PO	An optimization was incompatible with some cards. Workaround for FAT in an EFI partition (with USE_EFI). Do not use clock added with UART. DASH support.
1.5	2018-10-15	PO	Key control and UART control now possible together in the recorder.
1.4	2018-04-18	PO	SCI control always available (if you don't use the pins for buttons).
1.3	2017-04-01	PO	AICTRL3 bits have changed to allow SCI-controlled recorder. Better UART control.
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

List of Figures

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 IC's 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.

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 instruction RAM of 4096 words (20 kilobytes) in VS1063 is used for MMC/SD communication routines, handling of the FAT and FAT32 filesystems, upto a five-button user interface, and recording features.

Note: you need 32 kB EEPROM 25LC256. 8 kB EEPROM is not large enough for the standalone recorder.

Standalone Features:

- **No microcontroller is required**, boots from SPI EEPROM (25LC256).
- Uses MMC/SD/SDHC for storage. Hot-removal and insertion of card is supported.
- Supports FAT and FAT32 filesystems, **including subdirectories** (upto 16 levels). FAT12 is partially supported: subdirectories or fragmented files are not allowed. Recording to a FAT12 disk is not possible.
- Can use FAT with and without MBR on the disk, and FAT inside an EFI partition.
- Automatically starts playing from the first file after power-on.
- Power-on defaults are configurable.
- Three-button interface allows pause/play, shuffle play and loudness toggle, song selection, and volume control. Also supports UART and SCI control.
- Recording is possible in the recorder version.
- Code can be loaded through SCI by a microcontroller to eliminate SPI EEPROM.
- Available as a VSIDE solution and can thus be customized.

The vs1063 standalone player and the vs1063 standalone player/recorder have been developed to run on the VS1063 Protyping 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.

2 VSIDE Solution - Player and Player/Recorder

This VSIDE solution contains projects for a generic music player (**vs1063-standalone** with `standalone.c`) and a player/recorder (**vs1063-recorder** with `recorder.c`). Both are configured using pre-processor definitions in the `standalone.h` header file. In addition the solution includes a standalone link library project (**vs1063-standalone-lib**), which contains routines shared by the player and player/recorder projects.

Not all options you can choose from `standalone.h` can be used together for two main reasons: either the options are mutually exclusive by nature, or the combination of options may take too much memory.

2.1 Player

The Standalone Player `standalone.c` implements a SD / μSD card player with basic functions.

2.2 Player/Recorder

The Standalone Recorder `recorder.c` 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.

The default version (LINE_IN_ENABLE set to 6) checks GPIO6 to determine the encoding source (mic / line) and chooses the matching endcoding parameters. If I2S_ENABLE is set, GPIO6 is used for I2S (the jumper needs to be removed) and recording defaults to microphone input.

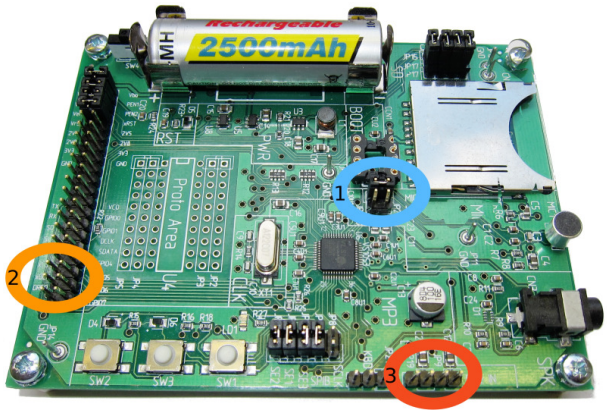

Figure 1: Standalone Recorder configured for mono microphone input

In Figure 1 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× (18 dB) is used. Microphone gain and autogain max can be edited from the boot image. To record from microphone, 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 2: Standalone Recorder configured for stereo line input (cable not included)

In Figure 2 the prototyping board is configured for 48 kHz stereo 160 kbit/s MP3 recording from the line input with a fixed 1.0× 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.

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.

Recording to new SD/uSD cards require a lot of current. If you use a rechargable battery it may not be able to provide enough peak current. If you are having issues with recording, supply external 2.8V power to the 2V8 pin of the prototyping board.

2.3 SCI Control

Control through the Serial Control Interface (SCI) is available in both the player and in the player/recorder as long as the SCI pins are not used for key control.

Controlling the player through SCI registers is the preferred control method when you already load the code from the microcontroller through SCI. It can still be used even when the vs1063a loads its code from the SPI EEPROM.

All non-application SCI registers can be used normally, except SM_SDINEW must be kept at '1' to enable GPIO2 and GPIO3. If the code is loaded through SCI, SCI_CLOCKF should be set by the user, preferably before starting the code. (This might vary depending on the VSIDE solution.)

SCI_AIADDR, SCI_AICTRL0, SCI_AICTRL1, SCI_AICTRL2, and SCI_AICTRL3 are used by the player, see below for their functions.

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	Samplerate and channels
0x6	WRAM	RAM write/read
0x7	WRAMADDR	Base address for RAM write/read
0x8	HDAT0	Stream header data 0
0x9	HDAT1	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**
0xE	**AICTRL2**	**Loudness setting / Recorder private**
0xF	**AICTRL3**	**Play mode**

AICTRL0

The currently playing song can be read from SCI_AICTRL0. In normal continuous 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 define which song you want played first.

AICTRL1

SCI_AICTRL1 contains the number of songs (playable files) found from the 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.

AICTRL2

In the player SCI_AICTRL2 holds the loudness value. SCI_BASS will be exclusive-ored with this value when loudness is toggled. The lowest bit should be 1 for the loudness indication to work correctly. In the player/recorder SCI_AICTRL2 is used only as the AGC maximum for the recording mode.

AICTRL3

SCI_AICTRL3 controls play mode, random play and other miscellaneous functions. AICTRL3 should be set to the desired play mode by the user before starting the code. If AICTRL3 is changed during play, note that the various play modes can put the player into pause mode.

SCI_AICTRL3 bits		
Name	**Bit**	**Description**
CTLR3_RECORD_ON	12	0=play, 1=record ON
CTLR3_PAUSE_ON	11	0=normal, 1=pause ON
CTRL3_BY_NAME	7	0=normal, 1=locate file by name
CTLR3_AT_END	6	if PLAY_MODE=3, 1=paused at end of file
CTLR3_NO_NUMFILES	5	0=normal, 1=do not count the number of files
CTLR3_ITERATE	4	internal - for directory listing
CTLR3_FILE_READY	3	1=file found
CTLR3_PLAY_MODE_MASK	2:1	0=normal, 1=loop song, 2=pause before play, 3=pause after play
CTLR3_RANDOM_PLAY	0	0=normal, 1=shuffle play

If CTRL3_RANDOM_PLAY is 1, a random song is selected each time a new song starts. if SHUFFLE_PLAY is enabled, the player goes through all files in random order, then goes through the files in a different order.

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 locates the file, then goes 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 plays files normally, but goes to pause mode and sets 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 is 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 a 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.

AICTRL3 should be set to the desired play mode by the user before starting the standalone application code. If the play mode is changed during play, care must be taken to switch modes in the correct order.

Open by Name

You can open specific files by using the CTRL3_BY_NAME bit.

You should first set pause mode bit CTRL3_PAUSE_ON and the open-by-name bit CTRL3_BY_NAME in AICTRL3, then write the 8.3-character filename into memory, then write 0xffff to AICTRL0 to select the song. After a file has been located you can check the file name to see if the file was located or not. You can also check SCI_AICTRL0: if it is non-zero, the file has been located, otherwise you have to check the file name to be certain.

To write the file name, first write 0x5800 to SCI_WRAMADDR, then the 6 words of the file name to SCI_WRAM.

The MSDOS 8.3-character filename does not include the point, so instead of sending "00000002.MP3" you need to send "00000002MP3\0", i.e. without the . and pad with a zero. Characters need to be upper case.

Recorder

SCI registers are used in the same way with the Player/Recorder as with the Player. SCI_AICTRL3 has one extra bit CTRL3_RECORD_ON to start recording mode.

When CTRL3_RECORD_ON is set to '1' the recording is started.

To end recording, clear the CTRL3_RECORD_ON bit of SCI_AICTRL3. Alternatively, write 0x8001 to SCI_AICTRL0. After the recording file is closed and the directory entry created, vs1063a will reset itself, which restores all SCI registers to their default values. If the code was loaded from SPI memory, the restart will happen automatically. If you loaded the code through SCI, monitor AICTRL3 or AICTRL0. When you see AICTRL3

(or AICTRL0 respectively) reset to 0, you can restart the recorder by first writing the appropriate SCI register values and then writing 0x50 to SCI_AIADDR.

During the recording mode SCI_AICTRL3 and other AICTRL registers are used by the encoder itself. However, RECORD_ON and PAUSE_ON are compatible with the player mode so that you can control the start and end of recording properly and you can use pause in both modes in the same way.

2.4 Keys

The user interface can be controlled by buttons. There are three different arrangements. Define the appropriate preprocessor definition in standalone.h .

- COMPAT_KEYS - three buttons, compatible with VS10xx prototyping board, not possible with SCI / SDI control.
- DIRECT_KEYS - when you need more than 3 buttons. Not possible with SCI / SDI control.
- SCI_KEYS - SCI/SDI pins cannot be used with SCI control, so GPIO4 to GPIO6 are used instead.

If none of the key options are defined, the key interface is not used.
With COMPAT_KEYS and DIRECT_KEYS the SCI and SDI can't be used.

2.4.1 COMPAT_KEYS

VS1011 and vs1003 do not have any spare GPIO's to connect keys, so the VS10xx Prototyping Board takes advantage of the SDI interface to connect three keys. COMPAT_KEYS is provided to be compatible with the Prototyping Board.

If you are using the Prototyping Board, check that the COMPAT_KEYS pre-processor definition is active in `standalone.h`. This allows the three-button interface of the vs10xx prototyping board to be used as-is. The DREQ to SCLK jumper (JP8) is not required with VS1063. SCI control cannot be used with COMPAT_KEYS active, because they use the same pins.

The three-button interface provides the most needed controls for the player. The keys function slightly differently in the recorder, see Section 2.4.4.

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

Note that SCI and SDI can not be used to trasfer data simultaneously with the COMPAT_KEYS button interface. (See SCI_KEYS.)

2.4.2 DIRECT_KEYS

VS1063 can read some dedicated pins directly, so four buttons can be connected to SI, xDCS, xCS, and SCLK.

This arrangement is used when DIRECT_KEYS is defined in `standalone.h` .

All buttons can be read independently so simultaneous presses of several keys can be detected if required.

If you don't use the I2S output, GPIO4, GPIO5, GPIO6, and GPIO7 are also free to be used for extra buttons or LEDs.

It is also possible to connect a 4×4 matrix keyboard by sing GPIO4 to GPIO7 as key scan outputs.

SW1 and SW2 on the prototyping board can be used for SI and xDCS keys without changes. SCLK (JP8) and SE3 jumper (JP16) should be removed. The prototyping board 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.

By default only the same 3 buttons are used.

Button	Short Keypress	Long Keypress
SW1 XDCS	Next song	Volume up
SW2 SI	Previous song	Volume down
SW3 XCS	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.

Note that SCI and SDI can not be used to trasfer data simultaneously with the DIRECT_KEYS button interface. (See SCI_KEYS.)

2.4.3 SCI_KEYS

If you want to use both the SCI control and keys at the same time, buttons cannot be connected to the SCI or SDI pins.

When you want the use both SCI control and keys, SCI_KEYS should be defined in `standalone.h` and buttons are connected to GPIO4 through GPIO6.

In this mode xCS, SI, SO, and SCLK are connected to the host controller's SPI bus. 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.

Button	Short Keypress	Long Keypress
GPIO5	Previous song	Volume down
GPIO4	Pause/Play	Toggle Shuffle
GPIO6	Next song	Volume up

Note: SCI_KEYS and ENABLE_I2S are mutually exclusive.

2.4.4 Player/Recorder Keys

The key mapping is slightly different in the Recorder. A long press of SW3 (pause/play) starts the recording mode and a short press toggles record pause during the recording mode.

Start of recording will take a few seconds, depending on the speed and size of the MMC/SD. Recording can be paused and continued by a short press of SW3. Recoding ends when SW1 or SW2 are pressed shortly or when the available space becomes full. The file is only created when the recording is ended through one of these methods. A file is not created if the unit is turned off or given a reset. The maximum filesize created is 2147483136 bytes, which gives 12 hours 25 seconds of recording time at 24 kHz.

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

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

The recorder can also be used with SCI control and UART control if enabled by the define UART_BUFFERED.

If UART_BUFFERED is enabled, the recorder automatically disables LOOP_MODE, random/shuffle play, and SKIP_ID3V2. CTRL3_NO_NUMFILES is also not available. If keys are used with UART, only the date is set with the "S" UART command.

2.5 UART Control

The Player and Player/Recorder also support control through UART at 115200 bps data rate (8 data bits, no parity, 1 stop bit) when compiled with UART_BUFFERED enabled in standalone.h . You can change the UART speed from `standalone.c` and `recorder.c`.

When UART control is used, the code is usually loaded from SPI EEPROM and SCI connection is not needed. The code can also be loaded and controlled through SCI and UART output used for visual feedback or logging.

Loading the code through UART is possible, but complicated, so code loading through UART is not detailed here.

2.5.1 UART Commands

Both the player and player/recorder work in one main mode - the playing mode, and another - the filemode, where the software waits for a command line before playing the next file. In addition, the player/recorder has a recording mode with slightly different commands.

During playing mode most commands are single-byte commands.

When `USE_PRINTABLE_OUTPUT` is defined, extra status information is sent to UART in human-readable format. Values are also printed in decimal.

Playing mode commands

- '+' volume up, responds with a byte indicating the new volume value
- '-' volume down, responds with a byte indicating the new volume value
- 'C' cancel play, in effect replay the same file from the beginning (unless file play mode or pause-before-play mode is active)
- '.' next file
- ',' previous file, or if played 5 seconds or more, the same file from the beginning
- 'l' toggles loop play mode. Returns 'l' for loop mode on, '>' for loop mode off.
- 'r' selects random play (shuffle) mode.
- 'e' toggle EarSpeaker headphone processing. (if enabled by USE_EARSPEAKER)
- 'c' selects continuous play mode, clears random play mode.
- 'f' selects filemode.
- '=' pause play
- '>' continue play (set 1x play speed)
- '≫' (0xbb) play faster (2x, 3x, 4x...)
- '?' request information, will respond with decode time and a value indicating file read position 0..255 (start..end). You can calculate a percentage by dividing the value by 2.56 .

Some additional playing mode commands of the recorder:

- 'R' Start recording without a specific name.
- 'E' End recording, create file.
- 'C' Cancel recording, do not create file.

In file mode the player waits for the next command after playing each file. Each command is a line delimited by the newline character (ASCII 10, '\n', ctrl-J). Whenever a command line is entered, the MMC/SD card communication is tested to detect SD card removal.

File mode commands

- "?" List the available commands. (if USE_PRINTABLE_OUTPUT)
- "c" Switch to continuous play mode.
- "L" List all files. Unless USE_PRINTABLE_OUTPUT is set, list is produced in raw data, the whole 32-byte directory entry of FAT is displayed for each file.
- "P<filenameogg>" Play the named file. The name should be a 8.3-character filename without the '.'. For example "PFILENAMEOGG".
- "p<number>" Play the name by number. The number is from 0 to the number of files minus 1.

Some additional filemode commands of the recorder:

- "S<created0> <created1>" Set recording date (and time) by 16-bit values. See recorder.c and FAT filesystem documentation for the values.
- "R" Start recording without a specific name.
- "R<filename>" Start recording into a file with specific name (8.3 format). The suffix is automatically determined by the encoding format.
- "D<filename>" Delete file by name. (if DELETE_FILE option defined)

2.6 Option Definitions

ENABLE_I2S

ENABLE_I2S enables the I2S output and the I2S output pins. The output format is 2-channel 16-bit 48 kHz.

If ENABLE_I2S is not set, LINE_IN_ENABLE is GPIO6 (see `recorder.c`).
If ENABLE_I2S is set, recording source is the microphone input.

UART_BUFFERED, USE_PRINTABLE_OUTPUT

UART_BUFFERED enables interrupt-driven UART receive, UART control, and status messages are sent to UART. If also USE_PRINTABLE_OUTPUT is defined, most UART output prints values in printable decimal notation. Otherwise values are printed as raw UART bytes of the appropriate size (1 byte, 2 bytes - MSB first, or 4 bytes - MSB first).

Note that USE_PRINTABLE_OUTPUT takes more code space, so you may not be able to use it with some combinations of other options.

With UART_BUFFERED you should use a fixed clock (the default CLOCKF_VAL is 5.0x). If you choose to allow automatic clock add (for WMA and AAC) by changing CLOCKF_VAL, the code tries to keep UART speed constant, but you may still miss UART bytes, and it takes a bit of instruction memory.

SHUFFLE_PLAY

If SHUFFLE_PLAY is active, random play uses a shuffle function instead of totally random sequence. Shuffle plays all files once before choosing another random order to play the files. A file cannot be played twice in a row (unless there is only one playable file).

SHUFFLE_PLAY is disabled automatically in the recorder when UART_BUFFERED is active.

SKIP_ID3V2

Some files have very large ID3v2 tags in them. It may take quite a while for the player to go through them trying to find some audio data. With SKIP_ID3V2 the tag is skipped and the correct point to start decoding is found immediately.

SKIP_ID3V2 is disabled automatically in the recorder when UART_BUFFERED is active.

ENABLE_HIGHREF

ENABLE_HIGHREF sets the analog reference voltage to 1.65V. This gives the output a little bit more swing and the input allows a larger voltage range. AVDD should be 3.3 V or higher with ENABLE_HIGHREF.

SD_POWER_ENABLE

If you power the SD card from its own regulator, you can choose which GPIO pin controls the regulator with SD_POWER_ENABLE.

Cycling the power of the SD card is the only way to reset it. If you have issues with card insertion, being able to cycle the power may make the player more robust.

START_IN_FILEMODE

The default mode is continuous play mode. Set START_IN_FILEMODE if you want to start in filemode.

Note that after the player/recorder finishes (or cancels) recording, the vs1063a will get a hardware reset, which returns the player to the initial continuous play/filemode setting.

NO_ENCODE_MP3

Affects recorder only - If you don't need to encode in MP3 format, you can save considerable amount of instruction memory by setting the `NO_MP3_FORMAT` define. (An easy way to gain space during debugging.)

USE_READ_MULTIPLE_BLOCK_ONE

There are some μSD cards that do not really work 100% with SINGLE_READ_BLOCK. The default version thus contains a workaround that always uses multiple read block, even when reading single sectors. However, very old MMC cards do not support read multiple block. If you want to support old cards instead of these new misbahaving ones, comment out the `USE_READ_MULTIPLE_BLOCK_ONE` define in `standalone.c` / `recorder.c`, and also change the #if 0 to #if 1 in `asm-readdisk.s` .

USE_DASH_IN_PLAYER, USE_DASH_IN_RECORDER

These allow AAC files in the MPEG-DASH (Dynamic Adaptive Streaming over HTTP) container to play correctly. The former enables DASH support in the player, the latter in the recorder.

USE_EARSPEAKER

If also `UART_BUFFERED` adds the 'e' command to toggle EarSpeaker processing.

USE_EFI

Sometimes a FAT partition may be hiding inside an EFI partition. This option (default on) enables the support for EFI partitions.

3 Application Loading

3.1 SPI Boot and MMC/SD

The software is loaded from SPI eeprom at power-up or reset if GPIO0 is pulled high with a pull-up resistor. The memory has to be an SPI EEPROM with a 16-bit or 24-bit address. The player code currently requires around 11 kB, thus at least 16 kB SPI EEPROM is recommended. The recorder code currently requires over 16 kB, thus at least 32 kB SPI EEPROM is recommended.

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

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

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

Some 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. **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). It seems that some MMC's react even to commands with invalid CRC's, which messes up the SPI boot.

To fix this issue MMC's chip select and clock inputs were swapped. This way MMC does not get clocked during the SPI boot and the system should work with all MMC's. The swap only occurs on the MMC pins, the SPI EEPROM connection must remain unchanged!

Boot Images

The SPI EEPROM boot images are created by the post-build step into the VSIDE solution directory.

Chip	File	Features
VS1063A	`player.img`	Player
VS1063A	`recorder.img`	Recorder

How to program in VSIDE:

1. Power down the prototyping board, connect GND, TX, RX.
2. Remove SPIB jumper (JP10).
3. Power up the prototyping board.
4. In VSIDE, choose the player or recorder as active project and build.
5. In VSIDE, choose Project → Prommer/Flasher Utility, press Next, Next, Start.
6. VSIDE uploads the programmer to the vs1063, then runs it to program the image file. If you see “Verify OK!”, the programming finished successfully.
7. Insert the SPIB jumper. Power up the prototyping board.

Power-on Defaults

Power-on Defaults can be changed from `standalone.h`, `c2.s`, or by adding register writes at the beginning of MyMain() in `standalone.c` or `recorder.c` .

The input clock is assumed to be 12.288 MHz. If you want to use a different crystal, the SCI_CLOCKF value should be changed. The default value is 0x9000 (3.5×+1.5× 12.288 MHz) for VS1063a, defined by CLOCKF_VAL define in `standalone.h`.

Code Loaded through SCI

When controlling the player through SCI, also the code is normally 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.

When 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 compressed plugins of the application for the microcontroller are created by the post-build step into the VSIDE solution directory. To start the application after uploading the code, write 0x50 to SCI_AIADDR (SCI register 10). Before starting the code, you may want to initialize SCI_CLOCKF and SCI_VOL. Other registers are initialized by the plugin or by the player code. Defaults can be changed from `standalone.h`, `c2.s`, or by adding register writes at the beginning of MyMain() in `standalone.c` or `recorder.c` .

Chip	File	Features
VS1063A	`player.plg`	player, compressed plugin
VS1063A	`recorder.plg`	recorder, compressed plugin

If your microcontroller does not have enough memory for the code loading tables, the player or recorder can also be loaded from SPI-EEPROM.

4 SCI Features

4.1 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 Schematics

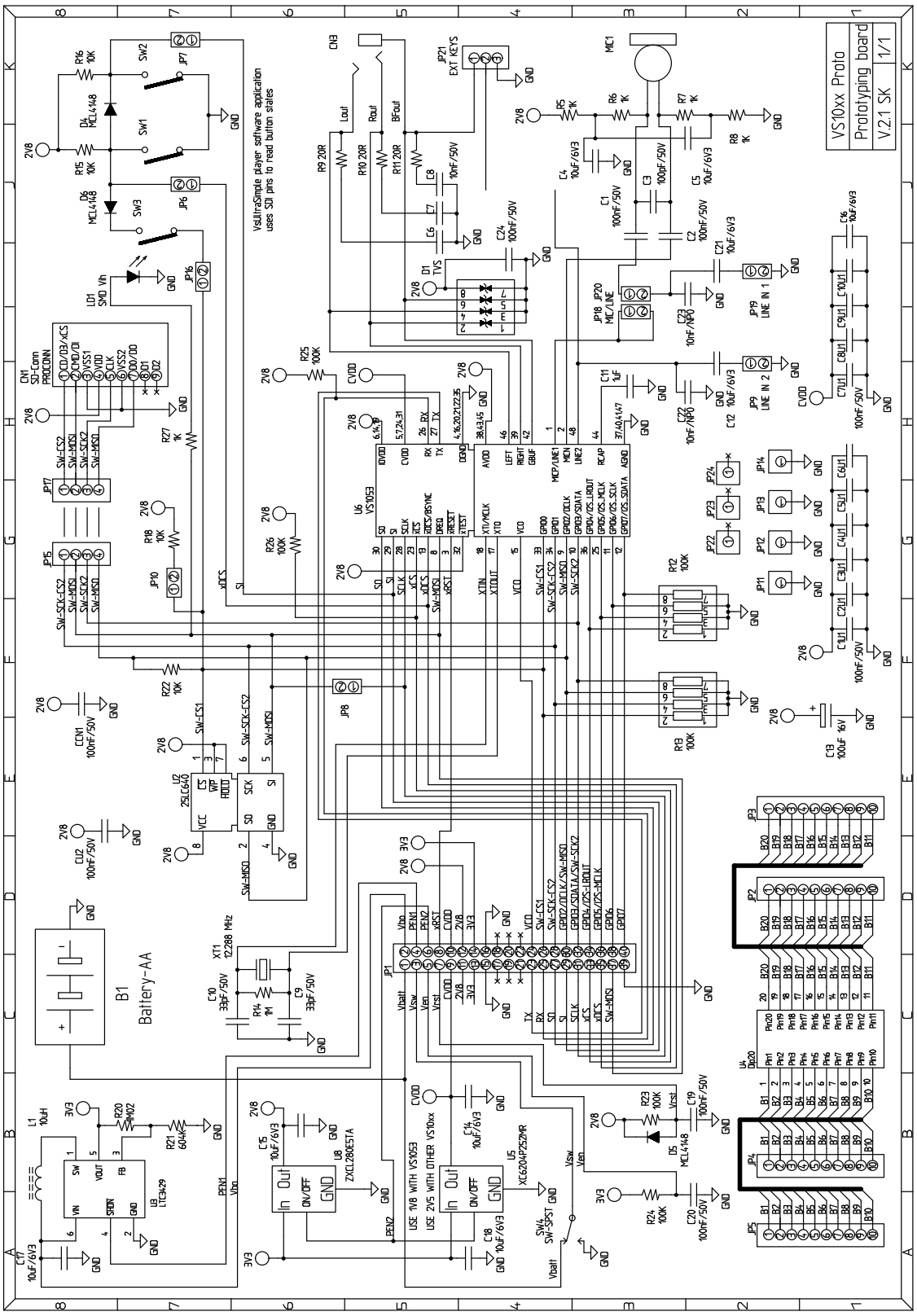

Figure 3: VS1063 Standalone Player Schematics

6 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.

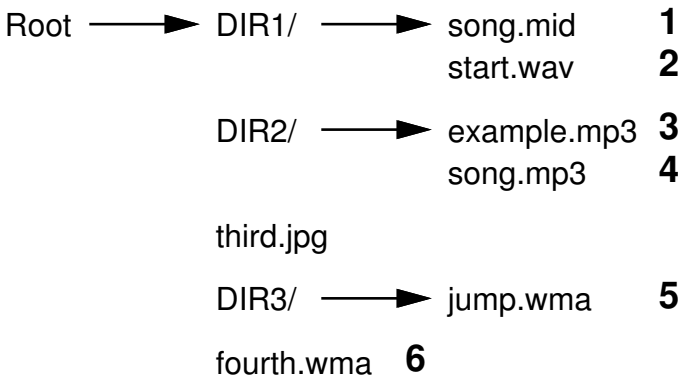

Figure 4: Play order with subdirectories

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.

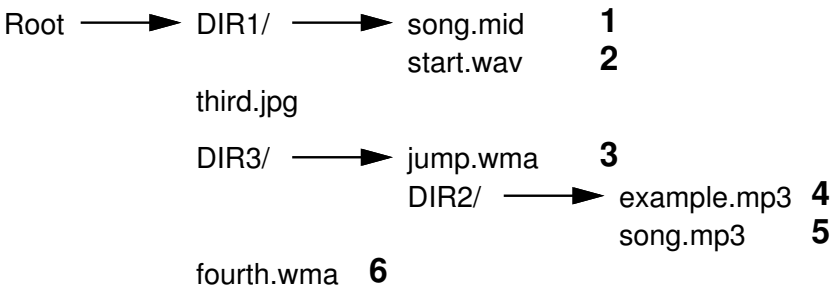

Figure 5: Play order with nested subdirectories

7 Document Version Changes

Version 1.5, 2018-10-09

- Optimization, now UART_BUFFERED and COMPAT_KEYS (and the other KEYS) are possible at the same time.

Version 1.4, 2018-04-26

- SCI control now always available.
- Player updated.
- Reformatted document.

Version 1.22, 2012-06-11

- Enhanced *Standalone Recorder*, and along with it Figures 1 and 2.
- 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).

8 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.