VS1053 STANDALONE PLAYER

VSMPG “VLSI Solution Audio Decoder”

Project Code: VS1053
Project Name: Support

Revision History			
Rev.	**Date**	**Author**	**Description**
1.30	2016-06-29	PO	Better UART control, VSIDE solution. (WiP)
1.20	2016-06-29	PO	New play loop for easier control.
1.19	2010-09-23	PO	Recorder version.
1.18	2009-10-27	PO	VS1053-specific version.

Contents

List of Figures

1 VS1053 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 VS1011E, VS1003B, VS1053B, and VS1103B 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 VS1053 using the same GPIO pins that are used to download the player software from the boot EEPROM.

The increased instruction RAM of 4096 words (20 kilobytes) in VS1053 is used for MMC 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. The default 8 kB EEPROM is not large enough for the standalone recorder, and neither for the standalone player with FLAC support.

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.
- 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.
- VS1053B transfer speed 4.8 Mbit/s (3.5×12.288 MHz clock).
- High transfer speed supports even 48 kHz 16-bit stereo WAV files.
- Three-button interface allows pause/play, shuffle play and loudness toggle, song selection, and volume control. Recording is possible in the recorder version.
- LED for user interface feedback

With Optional Microcontroller:

- External microcontroller can control the player through SCI or UART.
- Bypass mode allows MMC to be accessed also directly by the microcontroller.
- Code can be loaded through SCI by a microcontroller to eliminate SPI EEPROM.

2 VSIDE Solution - Player and Player/Recorder

This VSIDE solution contains a project for both a generic music player and a player/recorder that runs in the vs1053b chip itself. In addition there is a standalone library project, which contains a lot of routines shared by the player and player/recorder projects.

To increase the readability of the code - at least slightly - there are ow separate source files for the player (`standalone.c`) and the player/recorder (`recorder.c`). Both are configured using pre-processor definitions in the `standalone.h` header file.

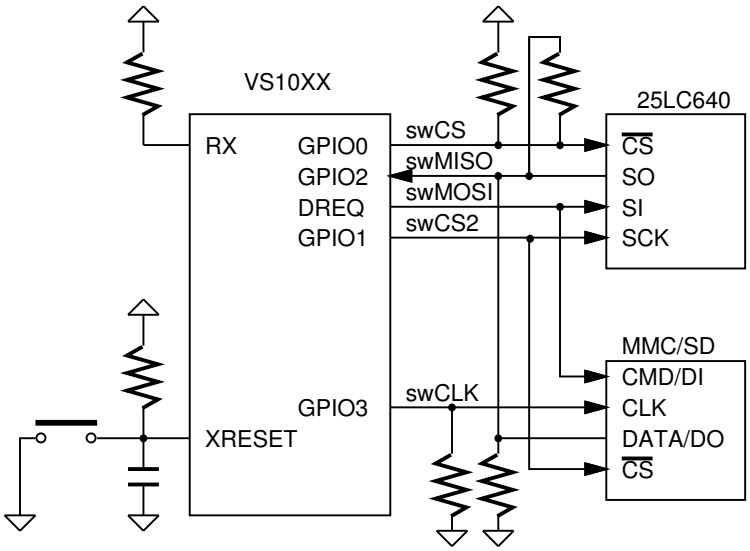
Note that not all options can be used together for two main reasons: either the options are alternative and thus mutually exclusive by nature, or an option may take too much instruction memory to fit into memory with some other option.

The options can be divided into a few classes:

- Control options (SCI, UART, keys)
- UART input and output type (human-readable, raw binary)
- Generic options
- Special options

2.1 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.



Note: MMC's /CS and CLK has been swapped from previous version

Figure 1: SPI-Boot and MMC/SD connection

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!

2.1.1 Boot Images

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

Chip	File	Features
VS1053B	`player.img`	Player
VS1053B	`recorder.img`	Recorder

Power-on Defaults

Power-on Defaults can be changed from `c2.s` or by adding register writes at the beginning of MyMain() is `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 0x8000 (3.5× 12.288 MHz) for VS1053b.

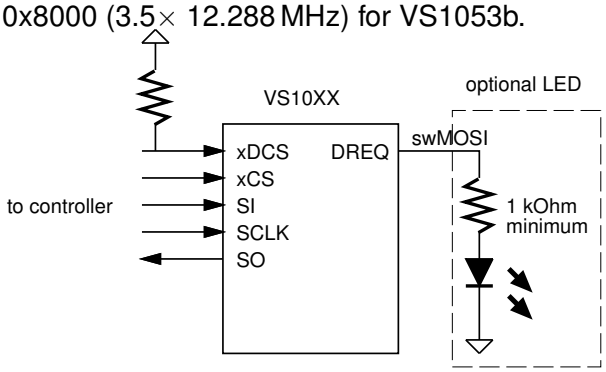


Figure 2: SCI connection

The code can also be loaded through the serial control inteface (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. GPIO1 should also have a pull-down resistor to prevent booting into real-time MIDI mode.

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.

Note that the connection from SCLK to DREQ is not used with vs1053b. VS1053b can read the buttons directly without using SCLK / SDATA.

Code Loaded through SCI

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. GPIO1 should also have a pull-down resistor to prevent booting into real-time MIDI mode.

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 and compressed plugins 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. Other registers are initialized by the loading tables. You can change the defaults by modifying the loading tables.

Chip	File	Features
VS1053B	`player.plg`	player, compressed plugin
VS1053B	`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.

2.2 SCI Control

Controlling the player by modifying the SCI registers through the serial control interface is the preferred control method when you load the code from the microcontroller through SCI. It can also be used if the vs1053b loads its code from the SPI EEPROM.

All non-application SCI registers can be used normally, except that 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.

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**	**-**
0xF	**AICTRL3**	**Play mode**

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 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.

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.

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**
CTRL3_BY_NAME	8	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_PAUSE_ON	4	0=normal, 1=pause ON
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. 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 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 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 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 (SCI_AICTRL2 for VS1103b) 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.

Open by Name

Since the 1.18 version, 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.

2.3 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.
- 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.

2.3.1 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`.

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

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.)

2.3.2 DIRECT_KEYS

VS1053 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

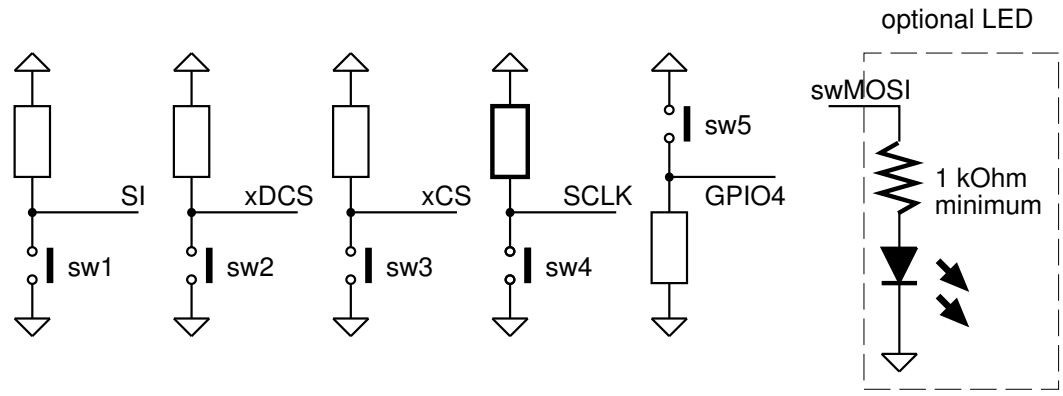


Figure 3: Five-button interface connection

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.)

2.3.3 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.

With SCI control, 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

2.4 UART Control

The Player and Recorder also support control through UART at 9600bps data rate (8 data bits, no parity, 1 stop bit) when compiled with UART_BUFFERED enabled in standalone.h .

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.

2.4.1 UART Commands

Both the player and player/recorder work in one main mode - the playing mode, and the filemode, where the software waits for a command 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.

- `'+'` volume up, responds with a byte indicating the new volume value
- `'-'` volume down, responds with a byte indicating the new volume value
- "XV\n" sets volume attenuation to X (decimal) in 0.5dB steps, e.g. "24V\n" sets -12dB.
  Volume commands respond with a byte indicating the new volume value.
- `'C'` cancel play, in effect replay the same file from the beginning (unless pause-before-play mode is active)
- `'.'` next file
- `','` previous file, or if played 5 seconds or more, the same file from the beginning
- "XP\n" play file X (decimal), e.g. "11P\n" plays the 12th file.
  Responds with "done\n" at the end of the current file, "files" + two bytes + "\n" for the total number of files available, then "play FILENAME\n".

- `'r'` selects random play (shuffle) mode.
- `'c'` selects continuous play mode, clears random play mode.
- `'f'` selects file play mode (pause before play)
- `'='` 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 .

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 it tested to detect SD card removal.

File mode commands

- `"?"` List the available commands. (if USE_PRINTABLE_INPUT)
- `"c"` Switch to continuous play mode.
- `"L"` List all files. 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> <created2>"` Set recording time and date by 3 16-bit values. See FAT documentation for the values.
- `"R"` Start recording without a specific name.
- `"R<filename>"` Start recording into a file with specific name (8.3 format).
- `"D<filename>"` Delete file by name. (if DELETE_FILE option defined)

Player/Recorder

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

SCI_AICTRL3 bits		
Name	**Bit**	**Description**
CTRL3_UPDATE_VOL	15	'1' = update volume (for UART control)
CTRL3_BY_NAME	8	'1' = locate file by name
CTRL3_RECORD_ON	7	'1' = start recording, '0' = end recording
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_PAUSE_ON	4	0=normal, 1=pause ON
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

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 to switch modes in the correct order.

When CTRL3_RECORD_ON is set to '1' the recording is started. Recording will end when the CTRL3_RECORD_ON is cleared, or when the available space is used up. After recording playback will start from the first song.

Note that during the recording mode SCI_AICTRL3 is used by the encoder itself and the only supported function is to clear CTRL3_RECORD_ON to stop recording.

3 Player and Player/Recorder

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

The Standalone Recorder makes use of the VS1053b microphone input. In addition to playing files from MMC/SD, audio from the microphone can be written to MMC/SD in mono linear 16-bit format. By default the sample rate is 24000 Hz. It can be changed with the `RECORD_FS` define from `standalone.h` .

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 (if EXTEND_DIRECTORY is defined). Directory extension works in FAT32 only, FAT16 just fails if the root directory is full. The maximum recording time is determined by the available contiguous space.

The key mapping is slightly different in the Recorder. A long press of SW3 will start the recording mode and a short press will toggle record pause during the recording mode.

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 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 UART control and SCI control if enabled.

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));
```

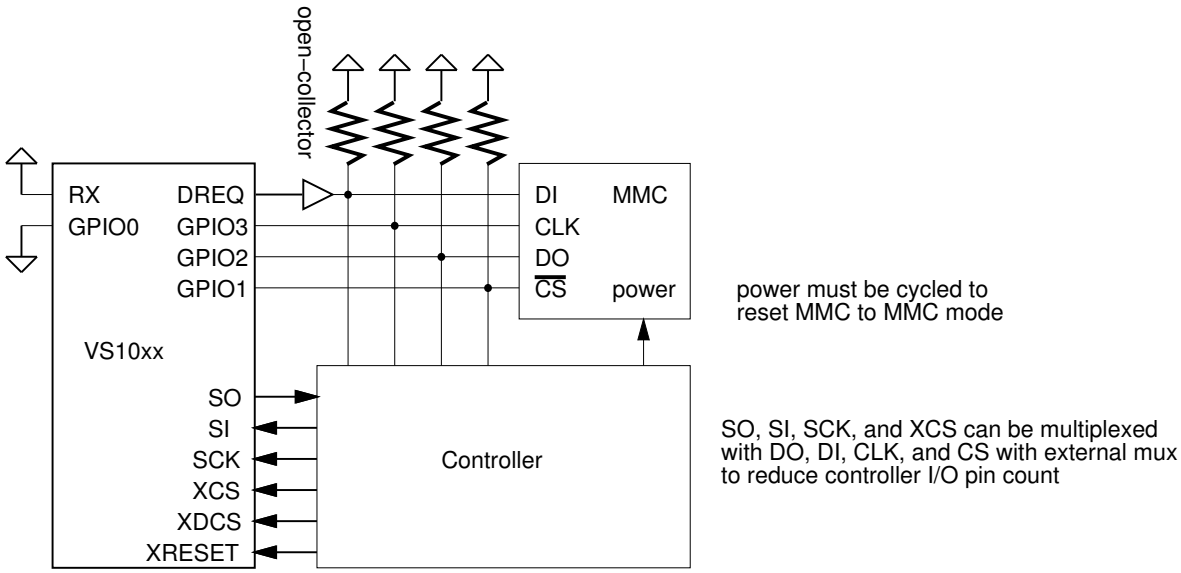
4.2 Bypass Mode

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. 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 though 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 4: Example of shared access

5 Example Implementation

The standalone player was implemented using the VS10xx prototyping board.

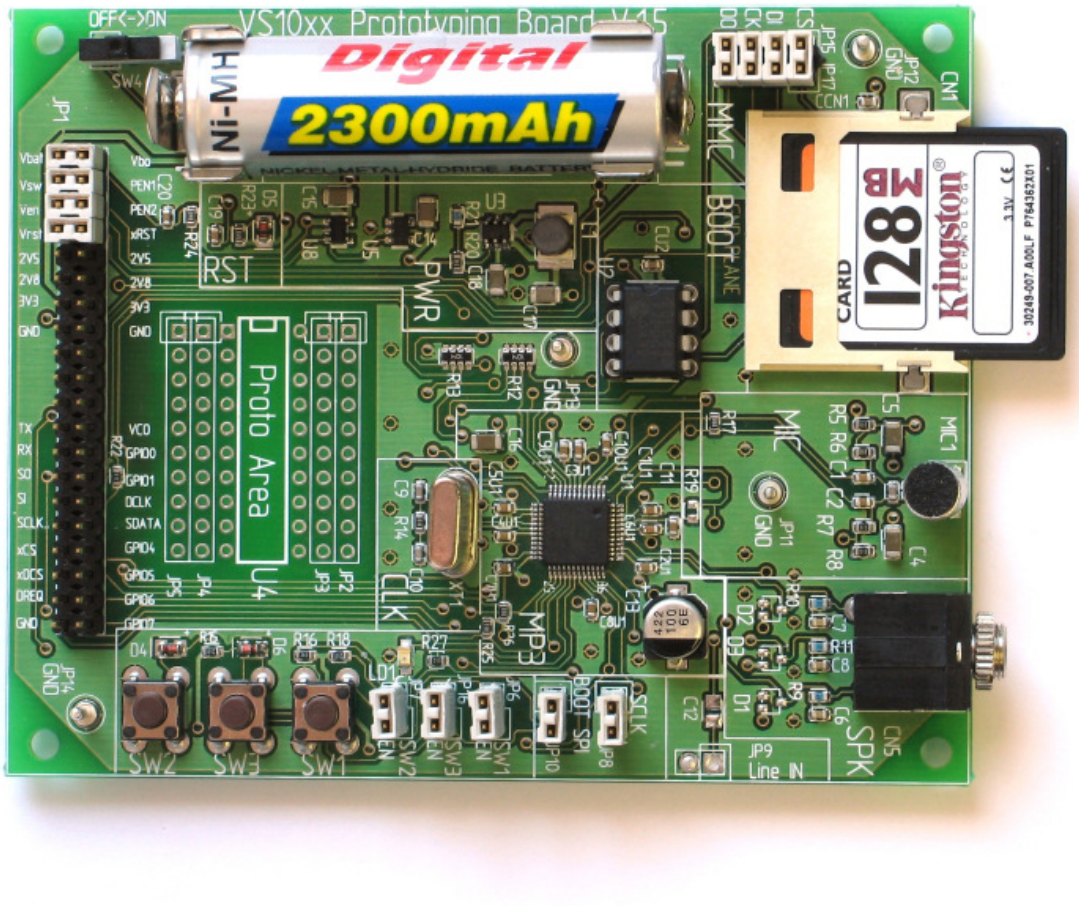
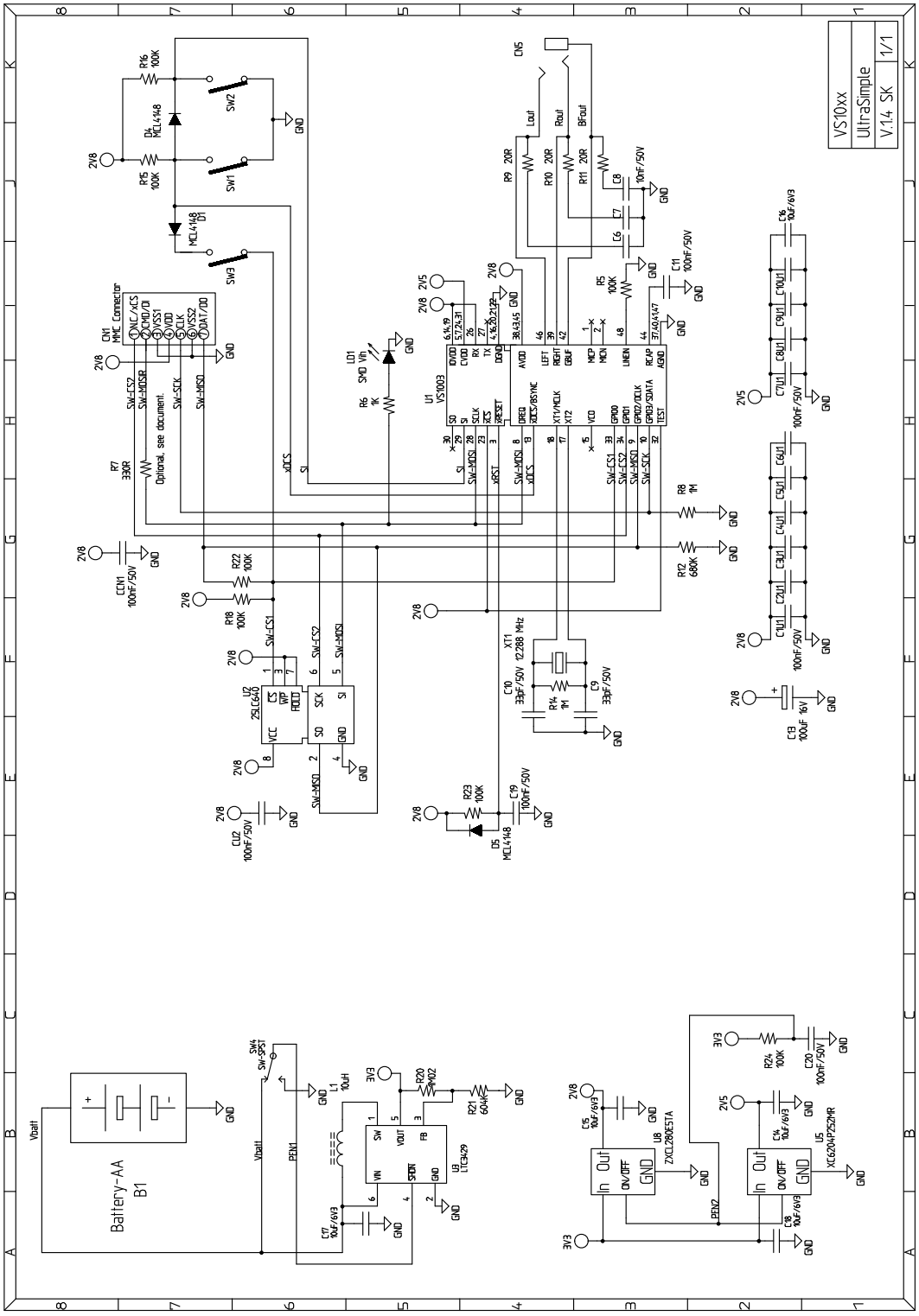


Figure 5: Standalone Player in Prototyping Board

The following example schematics contains a simple implementation for VS1003B. Power generation and player logic are separated. **Note: the schematics is a stripped-down version of the Prototyping Board. Use the attached schematics only as a basis for your own designs and refer to the Prototyping Board schematics when you work with the Prototyping Board.**



Note: **MMC's /CS and CLK of the SD card are swapped.** Optional resistor fixes problems with some MMC's (chapter2.1). See also Figure 3.

6 Document Version Changes

6.1 Version 1.20, 2016-06-29

- Rewritten the play loop so decoders can (mostly) exit nicely and the main control can do things more easily.
- Separated the recorder to a different source file recorder.c .
- Uses buffered human-readable UART control and output.

6.2 Version 1.19, 2010-09-23

- First release with more comments about file saving added to the source code.

6.3 Version 1.18, 2009-10-27

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

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.

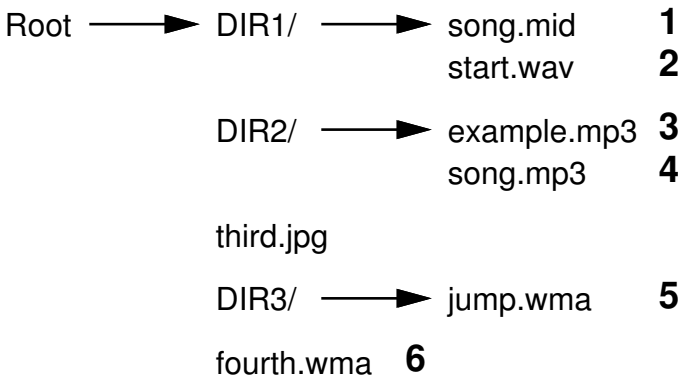


Figure 6: Play Order with subdirectories

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.

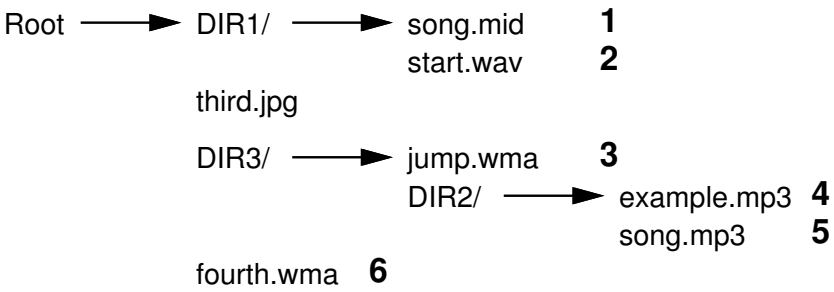


Figure 7: Play Order with nested subdirectories