VS1053 STANDALONE PLAYER

VSMPG “VLSI Solution Audio Decoder”

Project Code:	VS1053
Project Name:	Support

Revision History

Rev.	**Date**	**Author**	**Description**
1.19	2011-12-05	PO	SCI-only Player version for VSIDE.
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 VS1053B 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.

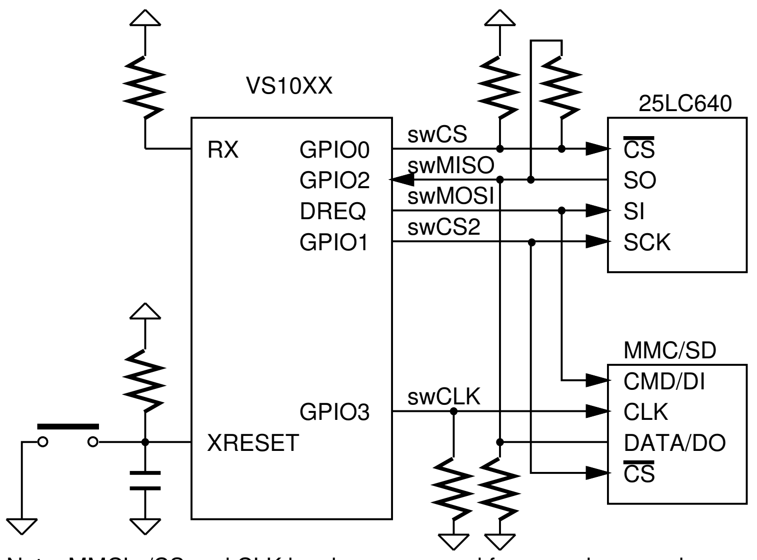
The increased instruction RAM of 4096 words (20 kilobytes) in VS1053 is used for MMC communication routines, handling of the FAT and FAT32 filesystems, upto a five-button user interface, and optional FLAC playback.

Note: you may need 32 kB EEPROM 25LC256. The default 8 kB EEPROM is not large enough for for the standalone player with FLAC support.

SCI-Controlled Standalone Player Features:

- Microcontroller loads code into VS1053.
- Microcontroller controls the player through serial control interface (SCI).
- 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 starting/power-on.
- Power-on defaults are configurable or set by microcontroller.
- 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.
- Bypass mode allows MMC to be accessed also directly by the microcontroller.
- Code can also be loaded from SPI EEPROM.

2 Boot EEPROM and MMC



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

Figure 1: SPI-Boot and MMC connection

Normally the SCI-controlled standalone player is loaded by the microcontroller through SCI. It can also be 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.

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.

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 use SPI boot and **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 SCI-Controlled Player

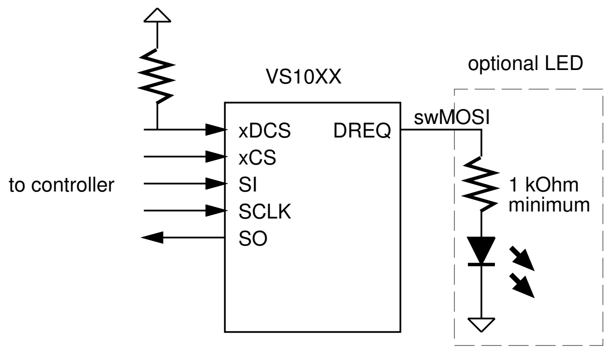


Figure 2: SCI connection

The SCI-controlled standalone player is controlled through the serial control interface (SCI). 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.

Normally the code is loaded through SCI by the microcontroller. In this case the boot EEPROM is 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 plugin files and old 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, and optionally AICTRL3 to set a suitable playing mode.

Chip	**File**	**Features**
VS1053B	`player1053bsci.c`	SCI+UART control
VS1053B	`sciplayer.plg`	SCI+UART control

With VSIDE the plugin file `sciplayer.plg` is created into the project directory.

Note: to be able to create the plugin, copy the included `coff2allboot.exe` to your VSIDE\BIN directory.

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	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 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_I2S_ENABLE	9	Enable I2S output, VS1053 only
CTRL3_BY_NAME	8	’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

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 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.

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.

With VS1053 you can use CTRL3_I2S_ENABLE to activate the I2S output. GPIO4 to 7 are then configured as I2S output pins, MCLK output is enabled, and 48 kHz output rate is selected (with 12.288 MHz XTALI).

SCI-Controlled Player with SPI Boot

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**
VS1053B	`player1053bsci.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. Have a pull-down also in GPIO1 to prevent boot into real-time MIDI 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.

4 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 'opens' a song and reads the file name, the it is played continuously in a loop.

```
#define MKWORD(a,b) (((int)(unsigned char)(a)<<8)|(unsigned char)(b))
  int song = 5;
  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];

      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");
    }
  }
  /* clear file ready and pause, select loop song mode */
  WriteMp3Reg(SCI_AICTRL3, (1<<1));
```

5 VSIDE Solution/Project

The SCI-Controlled Standalone Player is now delivered as a VSIDE solution/project. Decompress the source code packet `vs1053-standalone-sci-vside.zip` into your solution tree, click the solution file to start VSIDE and you are ready to compile, modify, and create your own additions.

The easiest way to configure the player features is editing `standalone.h` and commenting out or restoring various `#define`s.

The following defines are useful in the SCI-controlled player.

#define USE_FLAC

When active, enables FLAC (free lossless audio codec) decoding. The FLAC decoder will be linked into the software from `flac/libflac_min.a`. This is the same FLAC decoder version than in vs1063a ROM. The vs1053b patches package has slightly better version which calculates and checks also data CRC instead of just header CRC. Note that with FLAC support the executable and plugin file will be much larger.

#define AAC_PATCH

Enables 2 patches for AAC decoding. Note: may not be upto date with vs1053b-patches package.

#define SCI_KEYS

Normally SCI control handles the whole user interface. With SCI_KEYS you get both SCI control and keys. See which GPIO's are used for keys from standalone.c. Keys are assumed to be pulled high in idle state and grounded when pressed. The polarity of keys can be changed by defining KEYS_IDLE_LOW.

#define SHUFFLE_PLAY

Enables shuffle play instead of random play. Shuffle plays all files once before using a new play order.

#define SKIP_ID3V2

Jumps quickly over ID3v2 tags to allow playback to start as soon as possible. This is useful when mp3 files have embedded album art.

#define NO_WMA

Reject WMA files. Thus WMA end-product license is not needed.

#define NO_AAC

Reject AAC files. Thus AAC end-product license is not needed.

#define NO_MIDI

Reject MIDI files.

#define ENABLE_I2S

Enables I2S output at 48kHz rate. Can not be used together with SCI_KEYS.

#define ENABLE_HIGHREF

Enables 1.65 V reference voltage (normally 1.2 V), which allows better SNR/THD with higher output levels. Use only if AVDD is at least 3.3 V.

6 Example Implementation

The standalone player was implemented using the VS10xx prototyping board.

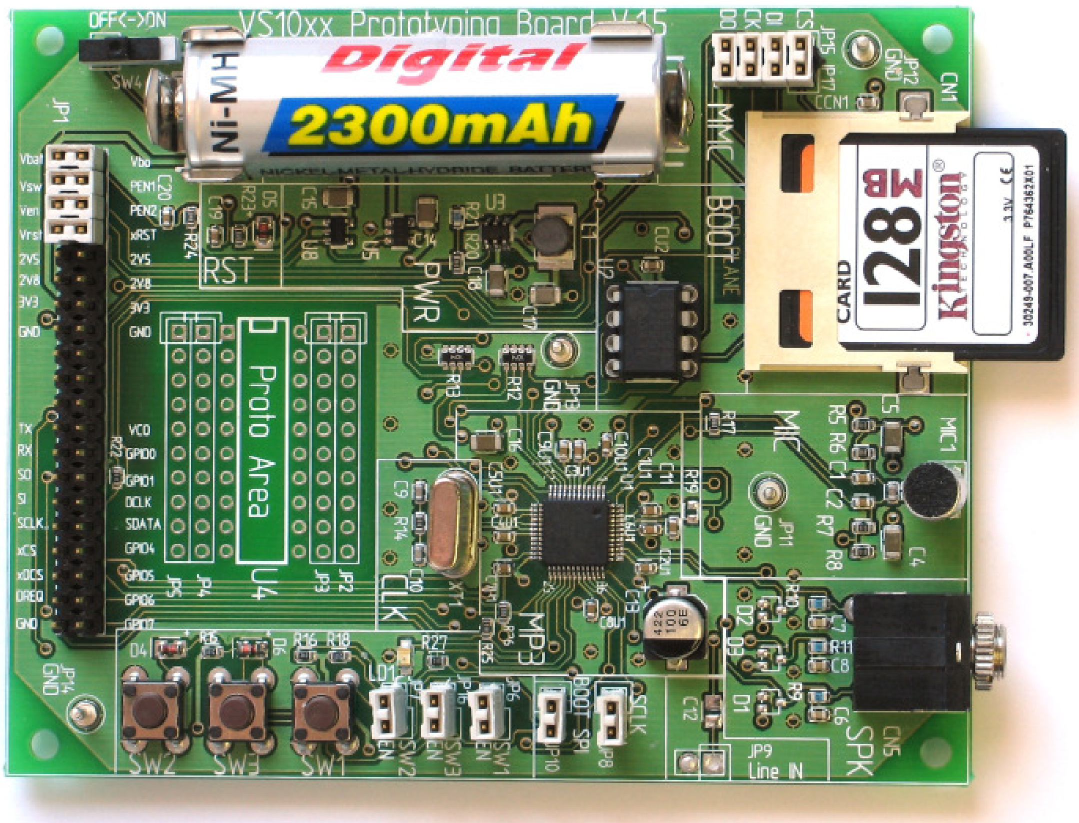
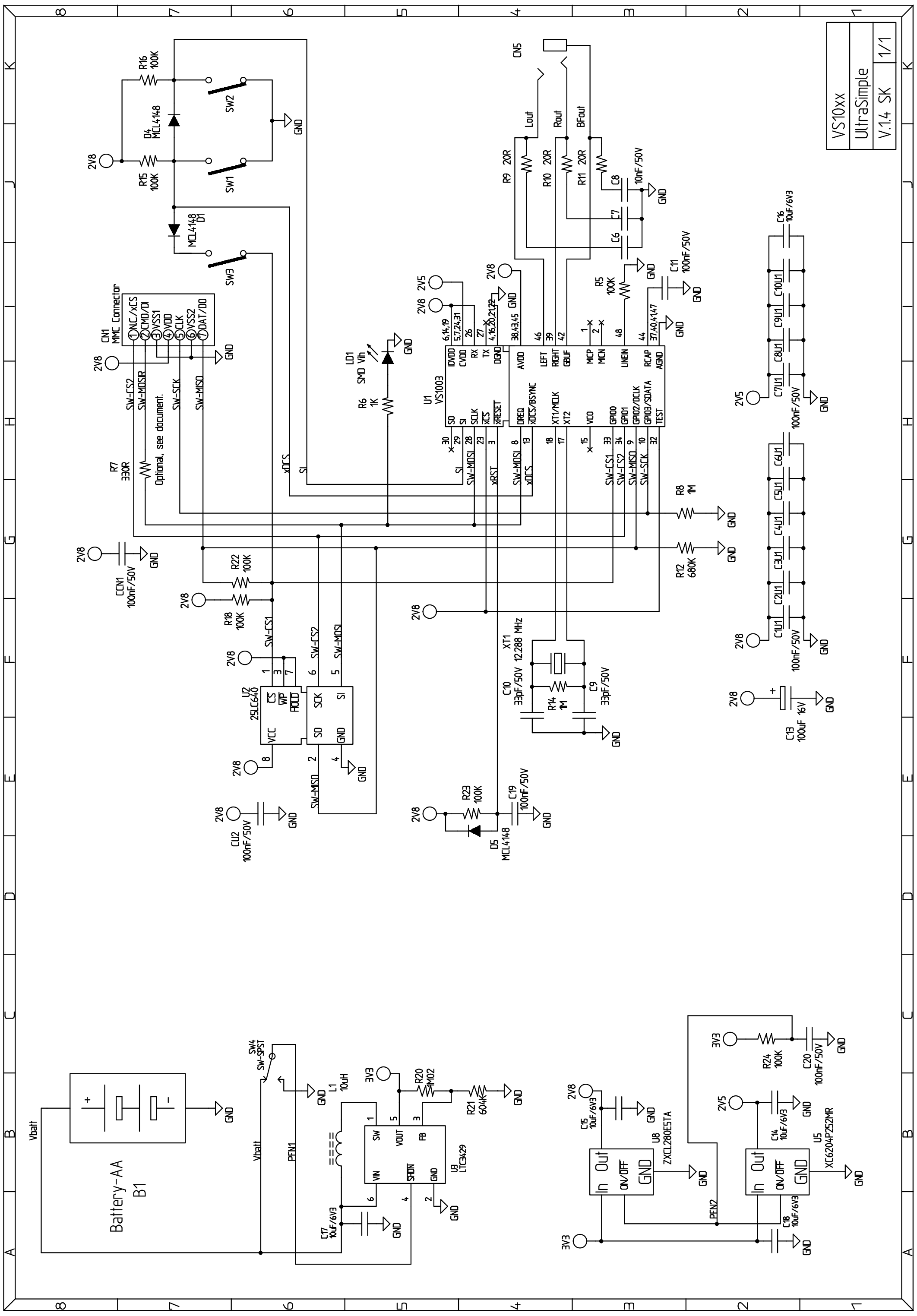


Figure 3: Standalone Player in Prototyping Board

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: **This is 3-button player version. Parts need to be omitted for SCI-controlled version.** See Figure 2.

7 Document Version Changes

7.1 Version 1.19, 2011-12-05

- Stripped version for VSIDE. Only supports SCI-controller player.
- Note: the schematics has not yet been updated.

8 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.

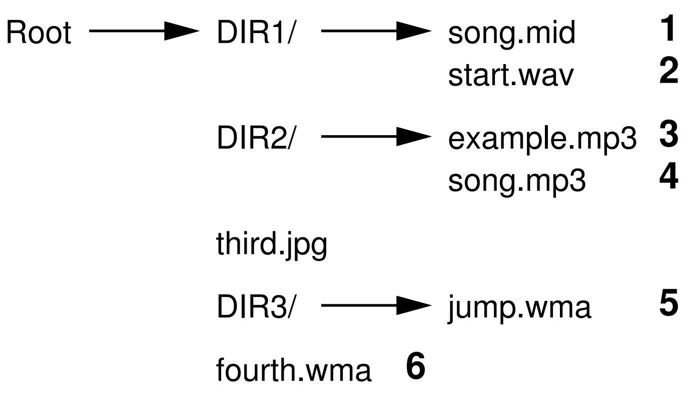


Figure 4: Play Order with subdirectories

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.

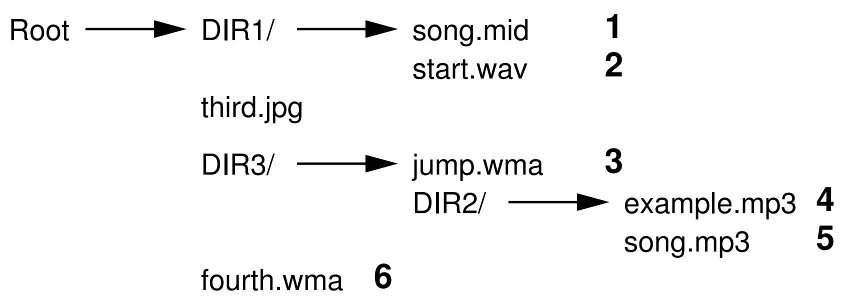


Figure 5: Play Order with nested subdirectories

9 How to Load a Plugin

A plugin file (.plg) contains a data file that contains one unsigned 16-bit array called plugin. The file is in an interleaved and RLE compressed format. An example of a plugin array is:

```
const unsigned short plugin[10] = { /* Compressed plugin */
  0x0007, 0x0001, 0x8260,
  0x0006, 0x0002, 0x1234, 0x5678,
  0x0006, 0x8004, 0xabcd,
};
```

The vector is decoded as follows:

1. Read register address number `addr` and repeat number `n`.
2. If `(n & 0x8000U)`, write the next word `n` times to register `addr`.
3. Else write next `n` words to register `addr`.
4. Continue until array has been exhausted.

The example array first tells to write 0x8260 to register 7. Then write 2 words, 0x1234 and 0x5678, to register 6. Finally, write 0xabcd 4 times to register 6.

Assuming the array is in `plugin[]`, a full decoder in C language is provided below:

```
void WriteVS10xxRegister(unsigned short addr, unsigned short value);

void LoadUserCode(void) {
  int i = 0;

  while (i<sizeof(plugin)/sizeof(plugin[0])) {
    unsigned short addr, n, val;
    addr = plugin[i++];
    n = plugin[i++];
    if (n & 0x8000U) { /* RLE run, replicate n samples */
      n &= 0x7FFF;
      val = plugin[i++];
      while (n--) {
        WriteVS10xxRegister(addr, val);
      }
    } else {           /* Copy run, copy n samples */
      while (n--) {
        val = plugin[i++];
        WriteVS10xxRegister(addr, val);
      }
    }
  }
}
```