VS1053 STANDALONE PLAYER

VSMPG “VLSI Solution Audio Decoder”

Project Code: VS1053
Project Name: Support

Revision History			
Rev.	**Date**	**Author**	**Description**
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.

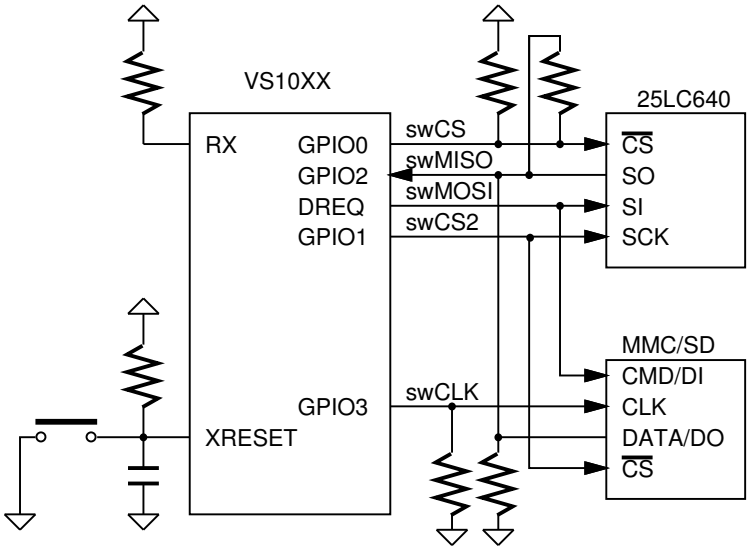
Standalone Features:

- **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.
- Five-button interface allows pause/play, shuffle play and loudness toggle, song selection, and volume control. Also recording is possible.
- 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 Boot EEPROM and MMC



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

Figure 1: 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. The player code currently requires almost 5 kB, thus at least 8 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.

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 Five-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 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

The number of buttons can also be extended. VS1053 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 be read independently so simultaneous presses of several keys can be detected.

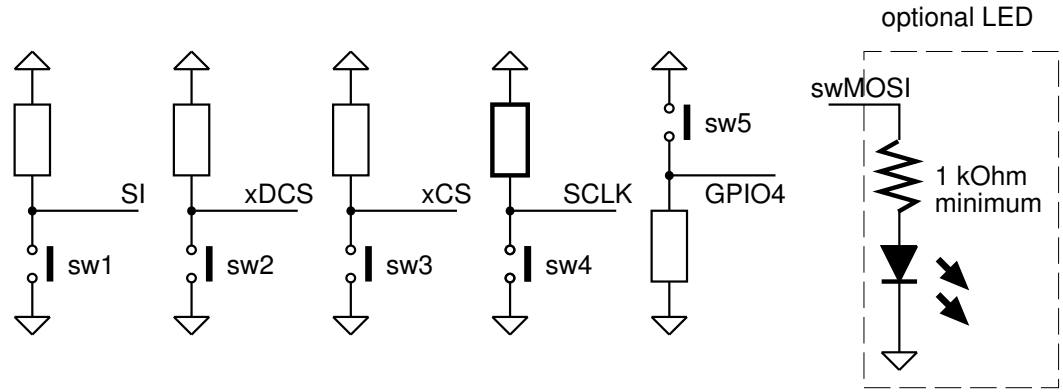


Figure 2: Five-button interface connection

GPIO5, GPIO6, and GPIO7 are also free to be used for extra buttons or LEDs. Us-

ing 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.

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 trasfer data simultaneously with the button interface.

3.1 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
VS1053B	`player1053bbut.bin`	Three-button interface, watchdog
VS1053B	`recorder1053bbut.bin`	Three-button interface, watchdog

3.2 Power-on Defaults

Default values are loaded from SPI EEPROM at power-on reset. Before the MMC/SD card is first accessed after power-on, approximately 22 ms delay is executed. The startup delay time can be changed from the boot image. The middle bytes in the string 0x00 0x12 0x34 0x0e contain the default value 0x1234 (22 ms). This value can be changed between 0x0000 (0 ms) and 0x3fff (80 ms). Do not change the 0x00 and 0x0e bytes.

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 values is 0x8000 (3.5× 12.288 MHz) for VS1053b. You can reduce the power consumption a bit by lowering the default clock and allowing the clock add (see chip datasheet for details).

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	0x9800	Clock control (for VS1003B/33C 0xa000)
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

Note: Standalone Recorder is work-in-progress. Features are subject to change without notice.

The Standalone Recorder makes use of the VS1053b microphone input. In addition to playing files from MMC/SD, sound from the microphone can be written to MMC/SD in mono linear 16-bit format. By default the sample rate is 24000 Hz. See the RECORD_FS define in 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. This 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 SW3 is pressed shortly, or the available space becomes full. 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 from ADC to DAC is 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
VS1053B	`recorder1053b.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	0x9800	Clock control (for VS1003B/33C 0x9000)
16, 17	-	0	Record gain, 0=AGC, 512=0.5×, 1536=1.5×, etc.
18, 19	-	0xffff	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

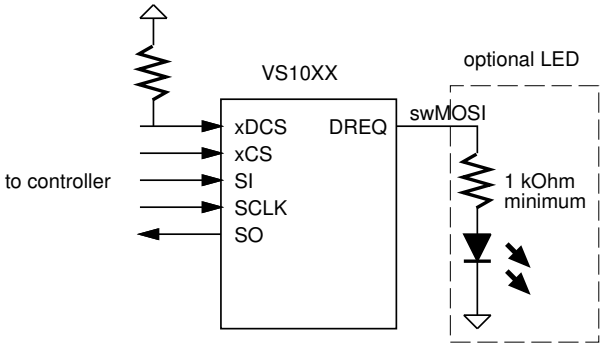


Figure 3: SCI connection

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. 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. 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 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
VS1053B	`player1053bsci.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	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**	**Number of songs on MMC (VS1103b)**
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_UPDATE_VOL	15	'1' = update volume (for UART control)
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 (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.

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.

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.

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

Example: to select song #10, send bytes 0x02, 0x00, 0x0a, 0x8c. The value to write is (0x02<<14) | (0x00<<7) | 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. This sets the volume register to the new value, but does not yet calculate new volume in all VS10xx chips because the write did not cause an SCI interrupt (this is also why writing to SCI_WRAMADDR and SCI_WRAM with UART control does not write to memory). However, VS1053B uses the new SCI_VOL value automatically.

To force volume calculation in chips that need it, send 0x02, 0x00, 0x00, 0x8f to set the volume update flag (CTRL3_UPDATE_VOL) in SCI_AICTRL3. 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));
```

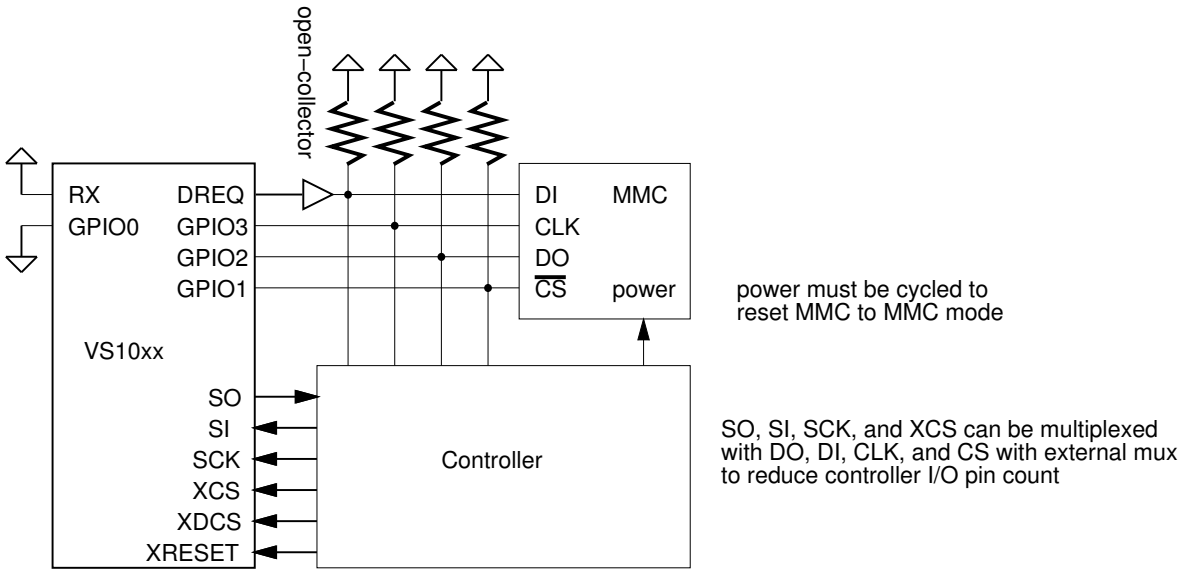
5.3 Bypass Mode

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. 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

6 SCI-Controlled Recorder

If the button interface is not used, the recorder can be controlled through the serial control interface (SCI). See chapter 5.

The SCI-Controlled Recorder is still experimental and user feedback is appreciated.

Code Loaded through SCI

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). Other registers are initialized by the loading tables. You can change the defaults by modifying the loading tables.

Chip	File	Features
VS1053B	`recorder1053bsci.c`	SCI control, watchdog

SPI EEPROM

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

Chip	File	Features
VS1053B	`recorder1053bsci.bin`	SCI control, watchdog

SCI Control

SCI registers are used in the same way as with the SCI-controlled Player. SCI_AICTRL3 has one extra bit to start recording mode. Do not set CTRL3_NO_NUMFILES, or the VSRECORD.WAV is not located and recording will not be possible.

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.

See the documentation of the common bits from the SCI-Controlled Player chapter.

When CTRL3_RECORD_ON is set to '1' and VSRECORD.WAV has been located on the card, the recording is started. Recording will end when the end of VSRECORD.WAV has been reached. You can also end recording by clearing CTRL3_RECORD_ON. After recording playback will start from the first song.

UART Control

The SCI-Controlled Recorder also supports UART control. See section 5.1 on how to use it.

In the recorder new song can be selected when pause mode is active, but not while recording is active.

Also note that the recorder does not have shuffle play mode.

7 Example Implementation

The standalone player was implemented using the VS10xx prototyping board.

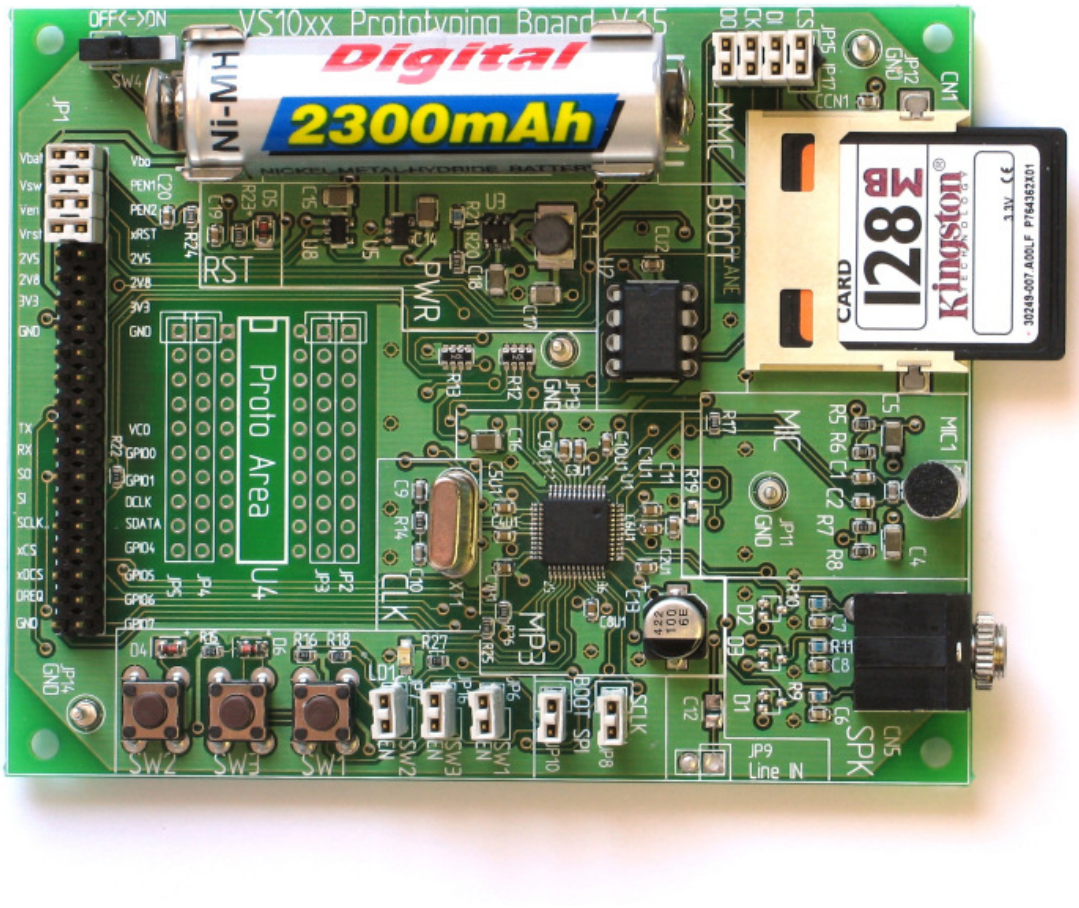
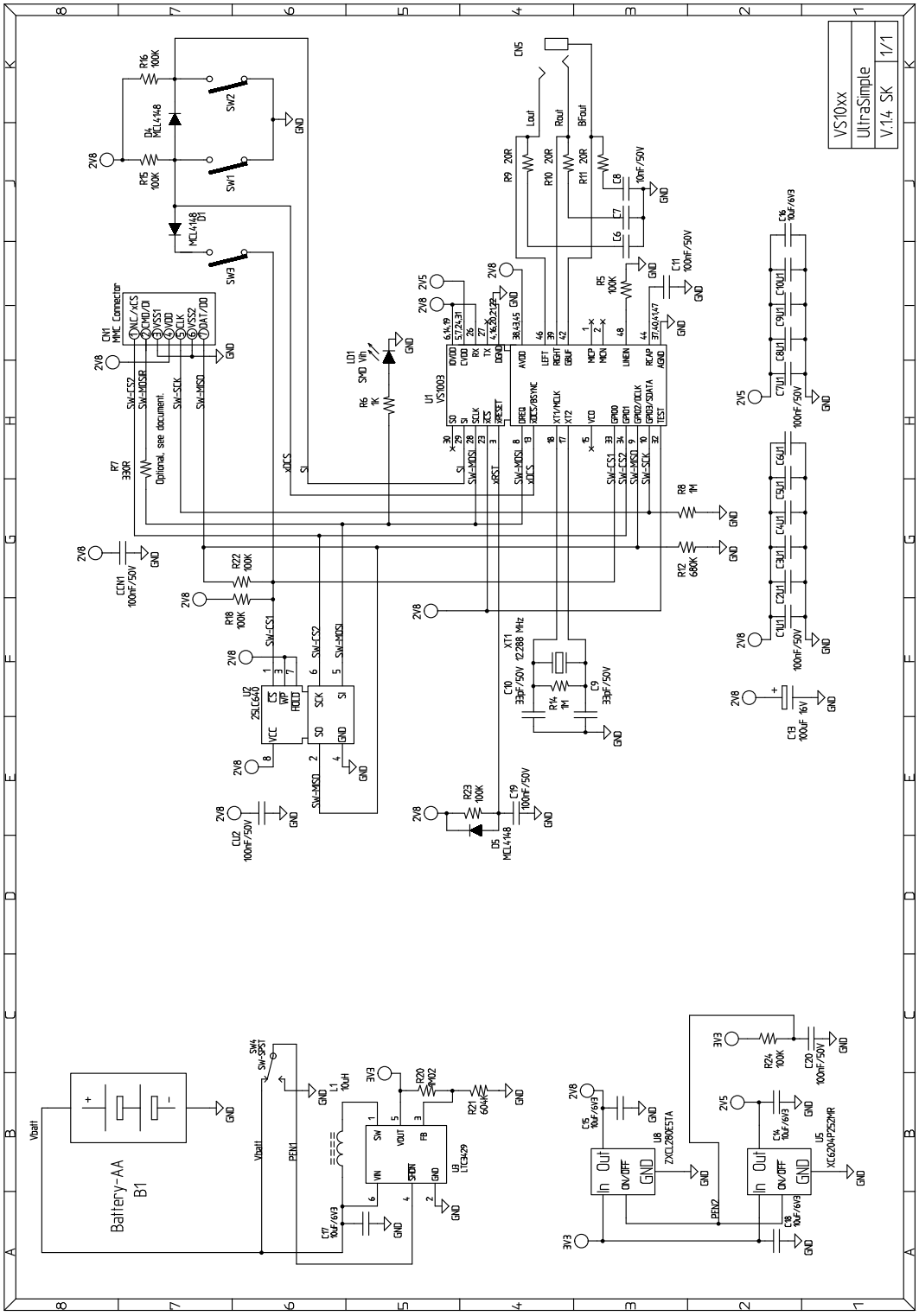


Figure 5: Standalone Player in Prototyping Board

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 swapped.** Optional resistor fixes problems with some MMC's (chapter2). See also Figure 2.

Battery life was tested with the prototyping board:

Test conditions:

- VS1002D
- Kingston 128 MB MultiMediaCard
- Beyerdynamic DT 131 headphones connected all of the time
- a single 2300 mAh AA-sized NiMH 1.2V Rechargeable Battery
- 128 kbps MP3 song (`utopia-free-sample.mp3`) on autorepeat
- default volume (-16dB)
- no LED

Test result: 25 hours 27 minutes of play time.

8 Document Version Changes

8.1 Version 1.19, 2010-09-23

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

8.2 Version 1.18, 2009-10-27

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

9 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.

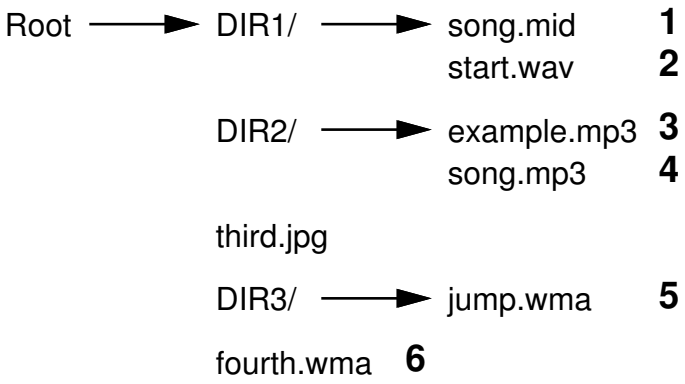


Figure 6: Play Order with subdirectories

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.

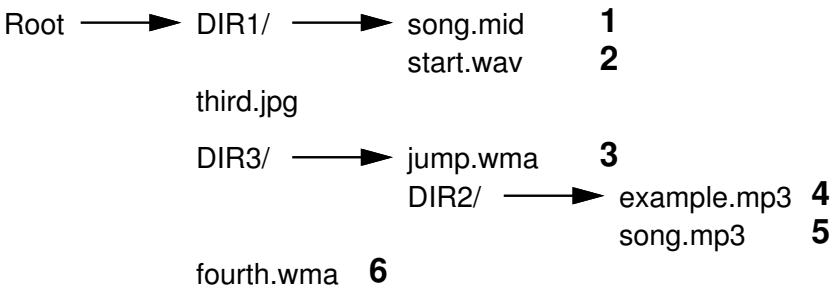


Figure 7: Play Order with nested subdirectories