VS10xx Standalone Player

VSMPG "VLSI Solution Audio Decoder"

Project Code:
Project Name: VSMPG

Revision History			
Rev.	**Date**	**Author**	**Description**
1.13	2007-01-23	PO	Drivers kept inactive until MMC found. SCI-version fixes. VS1033c version added.
1.12	2006-07-13	PO	Recorder updates. MMC communication changed.
1.11	2006-05-08	PO	More fragments allowed. SCI-version fixes.
1.10	2006-03-10	PO	Recorder bug fix, samplerate changed to 8000 Hz.
1.10pre	2006-01-13	PO	First Recorder Implementation.
1.02	2005-11-17	PO	File type detection from filename suffix. Pulldown resistor recommended for GPIO3.
1.01	2005-11-07	PO	Minor changes, see section 7 for details.
1.00	2005-09-30	PO	**SCHEMATICS CHANGED!** See also chapter 2 and section 2.1
0.95	2005-09-16	PO	SCI-version fixes, transfer routine fixes, skips non-audio files, startup delay
0.94	2005-08-24	PO	8.3-char filename in SCI version, loudness defaults, subdirectory support, partial FAT12 support fix for MMC-related powerup problem (chapter 2)
0.92	2005-07-13	PO	Version for VS1011E added, schematics fixed
0.91	2005-06-22	PO	SCI-controlled version upgrades (more to come)
0.9	2005-06-07	PO	Fixed user hook init for SCI-version and some MMC powerup problems
0.8	2005-05-27	PO	Fixed third button connection in the concept pic
0.7	2005-05-18	PO	Third button and optional LED added
0.6	2005-05-12	PO	Prototyping board picture added
0.5	2005-05-09	PO	Initial version

1 VS10xx 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, VS1002D, VS1003B , and VS1033C 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 directly connected to VS10xx using the same GPIO pins that are used to download the player software from the boot EEPROM.

The instruction RAM of 1280 words (5 kilobytes) is used for MMC communication routines, read-only handling of the FAT and FAT32 filesystems and a simple three-button user interface.

Standalone Features:

- **No microcontroller is required**, boots from SPI EEPROM (25LC640).
- Low-power operation
- Uses MMC for storage. Hot-removal and insertion of MMC is supported.
- Supports FAT and FAT32 filesystems, **including subdirectories** (upto 16 levels). FAT12 is partially supported: only unfragmented files are allowed for FAT12.
- Automatically plays the MMC contents from the first file after power-on.
- Power-on defaults configurable.
- VS1011E/VS1002D transfer speed 4.1 Mbit/s (24.576 or 2×12.288 MHz clock[1]).
- VS1003B/VS1033C transfer speed 4.8 Mbit/s (3.5×12.288 MHz clock).
- High transfer speed supports even 48 kHz 16-bit stereo WAV files.
- Optional three-button interface allows pause/play, random play and loudness toggle, song selection, and volume control.
- Optional LED for user interface feedback

[1] Because MMC communication takes some CPU time, 320 kbit/sec MP3 files need higher than 12.288 MHz clock in VS1011E/VS1002D if both bass enhancer and treble control are active.

With Optional Microcontroller:

- Optional external microcontroller can control the player through SCI.
- 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

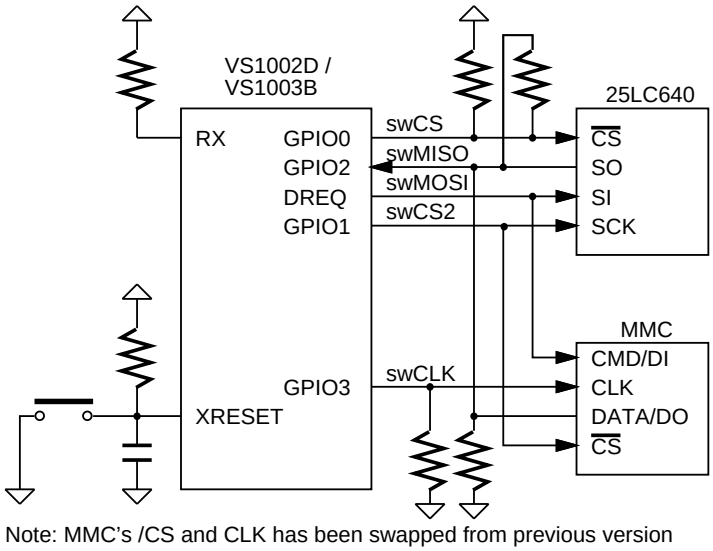


Figure 2.1: SPI-Boot and MMC connection

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 redefines the following pins (**note: MMC pin assignment has changed**):

Pin	SPI Boot	Other
GPIO0	swCS (EEPROM CS)	100 kΩ pull-up resistor
GPIO1	**swCS2 (MMC CS)**	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 a 330 Ω resistor between swMOSI (DREQ) and MMC's CMD/DI pin.**

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 is to change how the MMC is connected. The minimum changes are 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. But notice that the swap only occurs on the MMC pins, do not change the SPI EEPROM connection!

2.1 Fixing Existing VS10xx Prototyping Board 1.5

Since the 1.00 version of the standalone player MMC's /CS and CLK have been swapped. This change has no effect elsewhere in the design. However, to be able to use 1.00 or later player version, you need to use the new pin assignments.

Fixing the VS10xx Prototyping Board 1.5 is easy (if it has not been done for you already). Remove the CS and CK jumpers from JP15 / JP17 (pins 1 and 3). Then connect pin 1 of JP15 to pin 3 of JP17, and pin 1 of JP17 to pin3 of JP15. And that's all.

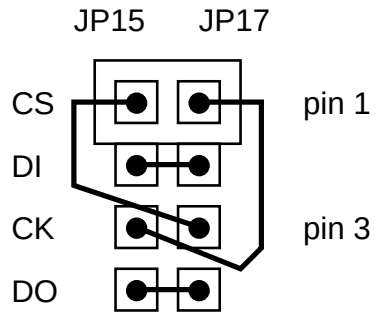


Figure 2.2: VS10xx Prototyping Board 1.5 Fix

3 Player with Three-Button UI

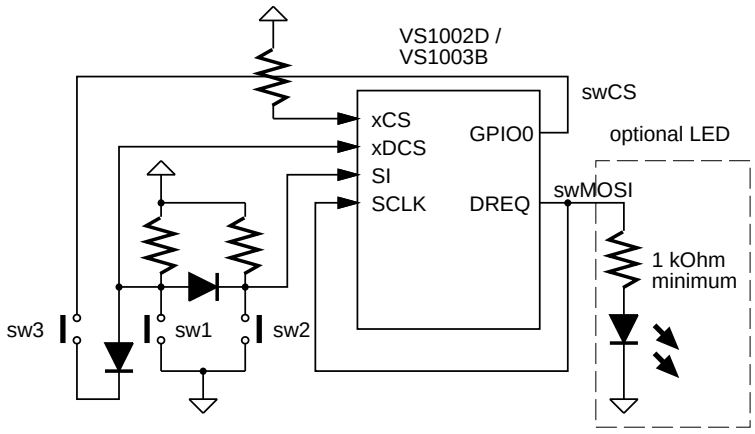


Figure 3.1: Three-button interface connection

A three-button interface is implemented with two diodes and two resistors. Only one button is detected simultaneously. If both SW1 and SW2 are pressed, only the other one (SW2) is detected. SW3 is only read if neither SW1 nor SW2 are pressed.

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 random play

Very little changes to the user interface are possible, because of the very restricted instruction RAM availability.

An optional LED 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 random play ON. Otherwise the LED shows MMC activity. In pause mode the LED lights up dimly.

Notice that SCI and SDI can not be used simultaneously with the three-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.** Note that to be able to use 1.00 or later player version, you need to use the new MMC pin assignments (see chapter 2 and section 2.1).

For VS1003B you can also select a version that does not play WMA files. If you use that version in your product, a WMA license should not be required.

Chip	File	Features
VS1011E	`player1011ebut.bin`	Three-button interface
VS1002D	`player1002but.bin`	Three-button interface, watchdog
VS1003B	`player1003but.bin`	Three-button interface, watchdog
VS1003B	`player1003nwbut.bin`	Three-button interface, watchdog, No WMA
VS1033C	`player1033cbut.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 are 0x9800 (2× 12.288 MHz) for VS1011E and VS1002D, and 0xa000 (3.5× 12.288 MHz) for VS1003B and VS1033C. You can reduce the power consumption a bit by using 0x8800 (3.0×..3.5× 12.288 MHz).

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 VS1002D, VS1003B and VS1033C microphone input. In addition to playing files from MMC, sound from the microphone can be IMA-ADPCM-encoded and written to MMC. By default the sample rate is 8000 Hz.

The recording always writes to the same file (VSRECORD.WAV) and the file must be initially provided by the user with a specific 512-byte header (see **mkrecord**). The samplerate of the VSRECORD.WAV file must match the samplerate used by the recorder, so remember this if you change the samplerate in the recorder boot image. (For code space reasons the sample rate can not be read from the WAV file.) The file size determines the maximum recording time.

Button	Short Keypress	Long Keypress
SW1	Next song	Volume up
SW2	Previous song	Volume down
SW3	Pause/Play (not in VS1002)	Start recording

Recording will only start if VSRECORD.WAV exists. Recording stops when the recording file is full or when any of the buttons are pressed shortly. **Do not turn off power when recording is active or you risk corrupting the MMC. Return to play mode first.**

Because of the instruction memory constraints, the user interface in the Standalone Recorder is simplified and some other features have been removed.

- no extra startup delay
- no Pause/Play in VS1002 version
- no random play
- no loudness selection (preset value is available)
- previous song selected also when more than 5 secs played
- only root directory supported, no subdirectory support
- no filename suffix check and no timeout if file is not playable
- LED does not indicate operating mode

VS1003B gives better recording quality than VS1002D, because it has higher microphone gain, but it has a longer delay in recorded sound monitoring. VS1033C tries to keep the loopback delay shorter.

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.** Note that to be able to use 1.00 or later player version, you need to use the new MMC pin assignments (see chapter 2 and section 2.1).

Chip	File	Features
VS1002D	`recorder1002.bin`	Player/recorder
VS1003B	`recorder1003.bin`	Player/recorder
VS1033C	`recorder1033c.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
56, 57		0x0300	Default rate 8000 Hz (0x0480 for VS1003)
63		0x04	Recording mode: see below

Sample rate can also be changed from the boot image. Note, however, that you have to have the same sample rate in VSRECORD.WAV.

Recording Sample Rate Selection		
Rate /Hz	**VS1002**	**VS1003**
8000	0x0300	0x0480
11076	-	0x0340
12000	-	0x0300
16000	-	0x0240
24000	-	0x0180

Recording Mode	
0x04	Mic selected, no high-pass
0x0c	Mic selected, high-pass (use for 8000 Hz only)
0x14	Line input selected, no high-pass
0x1c	Line input selected, high-pass (use for 8000 Hz only)

mkrecord

The `mkrecord` program generates a `VSRECORD.WAV` file that can be used with the standalone recorder. The default size is two megabytes, i.e. 517 seconds. You can set the file size in either bytes or in seconds.

For example the following commands create a twelve-minute `VSRECORD.WAV` into `H:` .

```
C:\ > H:
H:\ > mkrecord -t 12:00
720 seconds
Rate:  8000 Hz, Size:  2919936, Byte rate:  2027
Number of samples 5759020, Duration 719.88 seconds
Created VSRECORD.WAV
```

The source code and windows executable are available as `code/mkrecord.c` and `code/mkrecord.exe` respectively.

`mkrecord.c`:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

unsigned char header[512] = { /* RIFF WAV header */
    0x52,0x49,0x46,0x46,0xF8,0x8F,0x06,0x00,0x57,0x41,0x56,0x45,
    0x66,0x6D,0x74,0x20,0x14,0x00,0x00,0x00,0x11,0x00,0x01,0x00,
    0x80,0x3E,0x00,0x00,0xD7,0x0F,0x00,0x00,0x00,0x01,0x04,0x00,
    0x02,0x00,0xF9,0x01,0x66,0x61,0x63,0x74,0xC8,0x01,0x00,0x00,
    0x1E,0xEE,0x0C,0x00  /* the rest are zeros */
};

long GetSeconds(char *s, char **sc) {
    long secs = 0;
    while (1) {
        secs += strtol(s, sc, 10);
        if (**sc != ':')
            return secs;
        /* if it was minutes (or hours even..) */
        secs *= 60;
        s = *sc + 1;
    }
}

void FixVal(unsigned char *s, unsigned long val) {
    s[0] = (unsigned char)val;
    s[1] = (unsigned char)(val>>8);
    s[2] = (unsigned char)(val>>16);
    s[3] = (unsigned char)(val>>24);
}
```

```
int main(int argc, char *argv[]) {
    unsigned long fs = 8000, size = 2048*1024, dataSize, numOfSamples, tim = 0;
    int n;
    FILE *fp;
    for (n = 1; n < argc; n++) {
        char *err = NULL;
        if (!strcmp(argv[n], "-f")) {
            fs = strtol(argv[++n], &err, 0);
        } else if (!strcmp(argv[n], "-s")) {
            size = strtoul(argv[++n], &err, 0) & ~511L;
        } else if (!strcmp(argv[n], "-t")) {
            tim = GetSeconds(argv[++n], &err);
        } else {
            fprintf(stderr, "Usage: %s [-f <rate>] [-s <size>] [-t <time>]\n",
                    argv[0]);
            fprintf(stderr, "Example: %s -t 5:30\n", argv[0]);
            return EXIT_FAILURE;
        }
        if (err && *err)
            fprintf(stderr, "Invalid number '%s'\n", argv[n]);
    }
    if (tim) {
        fprintf(stderr, "%ld seconds\n", tim);
        size = ((unsigned long)(tim * (fs * 256/505.0)) + 511) & ~511L;
    }
    dataSize = size - 512;
    numOfSamples = dataSize / 256 * 505;

    fprintf(stderr, "Rate: %ld Hz, Size: %lu, Byte rate: %ld\n",
            fs, size, fs*128/505);
    fprintf(stderr, "Number of samples %lu, Duration %6.2f seconds\n",
            numOfSamples, (double)numOfSamples/fs);

    FixVal(header+4, size-8); /* chunk size = file size - 8 */
    FixVal(header+24, fs);    /* sample rate */
    FixVal(header+28, fs*256/505); /* byte rate */
    FixVal(header+48, numOfSamples); /* number of samples */
    FixVal(header+504, 0x61746164); /* "data" in little-endian format */
    FixVal(header+508, dataSize);   /* data size = file size - 512 */

    if ((fp = fopen("VSRECORD.WAV", "wb"))) {
        unsigned char zero[512] = {0};
        fwrite(header, 512, 1, fp);

        if (fseek(fp, dataSize, SEEK_SET) == 0) {
            fwrite(zero, 512, 1, fp); /* Seek successful, write last block */
        } else {
            while ((size -= 512) > 0) /* Failed, write as much as possible */
                fwrite(zero, 512, 1, fp);
        }
        fclose(fp);
        fprintf(stderr, "Created VSRECORD.WAV\n");
        return EXIT_SUCCESS;
    }
    fprintf(stderr, "Could not open VSRECORD.WAV for writing!\n");
    return EXIT_FAILURE;
}
```

5 SCI-Controlled Version

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 and xDCS should have a pull-up resistor.

The code is loaded through SCI by the microcontroller. The application loading tables for the microcontroller are available in the `code/` subdirectory. To start the application after uploading the code, write 0x30 to SCI_AIADDR (SCI register 10). Before starting the code, you should initialize SCI_CLOCKF and SCI_VOL.

Chip	File	Features
VS1011B	`player1011bsci.c`	SCI control
VS1011E	`player1011esci.c`	SCI control
VS1002D	`player1002sci.c`	SCI control, watchdog
VS1003B	`player1003sci.c`	SCI control, watchdog
VS1003B	`player1003nwsci.c`	SCI control, watchdog, No WMA
VS1033C	`player1033csci.c`	SCI control, watchdog

All non-application SCI registers can be used normally, except that SM_SDINEW must be set to '1' to enable GPIO2 and GPIO3 (default for VS1002 / VS1003 / VS1033C). 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**	**Memory read in SCI version**
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. 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 use SCI_AICTRL2 to read VS10XX memory. First write the address (0..0x3fff for X memory, 0x4000..0x7fff for Y memory) to SCI_WRAMADDR, then read from SCI_AICTRL2. The first read will return carbage, the second one returns the value from the address you specified. Each read increments the internal address so you can get data from consequtive addresses with consequtive reads from SCI_AICTRL2.

In VS1003B, VS1033C and VS1011E you can use SCI_WRAMADDR and SCI_WRAM to both write and read memory and you don't need to use SCI_AICTRL2.

SCI_AICTRL3 bits		
Name	**Bit**	**Description**
CLTR3_PAUSE_ON	4	0=normal, 1=pause ON
CLTR3_FILE_READY	3	if PLAY_MODE=2, 1=file found
CLTR3_PLAY_MODE_MASK	2:1	0=normal, 1=loop song, 2=pause before play
CLTR3_RANDOM_PLAY	0	0=normal, 1=random play

AICTRL3 should be set to the desired play mode by the user before starting the code.

If the lowest bit of SCI_AICTRL3 is 1, a random song is selected each time a new song starts. Note that this is random play, not shuffle play. Shuffle plays all songs in random order, while random play can select the same song multiple times.

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.

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.

The SCI-controlled version can also be loaded from SPI-EEPROM. You can change the power-on defaults in the same way than in the standalone player version.

Chip	File	Features
VS1011E	`player1011esci.bin`	SCI control
VS1002D	`player1002sci.bin`	SCI control, watchdog
VS1003B	`player1003sci.bin`	SCI control, watchdog
VS1003B	`player1003nwsci.bin`	SCI control, watchdog, No WMA
VS1033C	`player1033csci.bin`	SCI control, watchdog

5.1 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 0x780..0x785 (VS1011B, VS1011E, VS1002D) or 0x1800..0x1805 (VS1003B, VS1033C). 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, 0x4780); /* 0x5800 for VS1003B/VS1033C */
      ReadMp3Reg(SCI_AICTRL2); /* dummy read */
      for (i=0; i < 6; i++) {  /* read filename */
        ch[i] = ReadMp3Reg(SCI_AICTRL2); /* first 2 characters */
        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));
```

5.2 Bypass Mode

VS10xx can be disconnected from MMC to allow direct microcontroller access by keeping it in reset. This leaves GPIO pins as inputs, but connecting microcontroller to MMC's DI needs an OR between DREQ and controller's I/O pin. This is because DREQ is always an output pin.

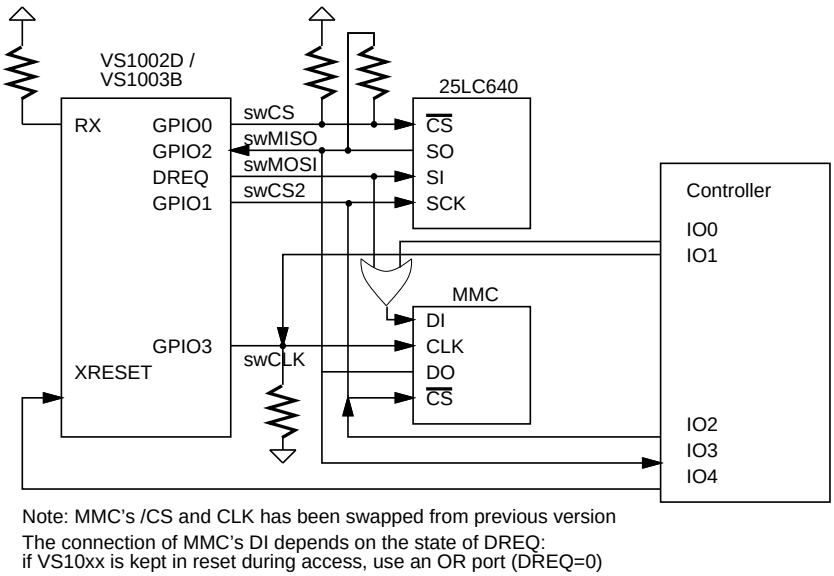


Figure 5.1: Example of shared access

Another way to disconnect VS10xx from MMC is keeping GPIO0 low when reset is deasserted. This bypasses the SPI-boot, leaving GPIO pins as inputs. SM_SDINEW must be '1' (the default in VS1002/VS1003). DREQ rises when normal firmware is ready. In this case an AND between DREQ and the controller's I/O pin is required.

Because the latter 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.

6 Example Implementation

The standalone player was implemented using the VS10xx prototyping board.

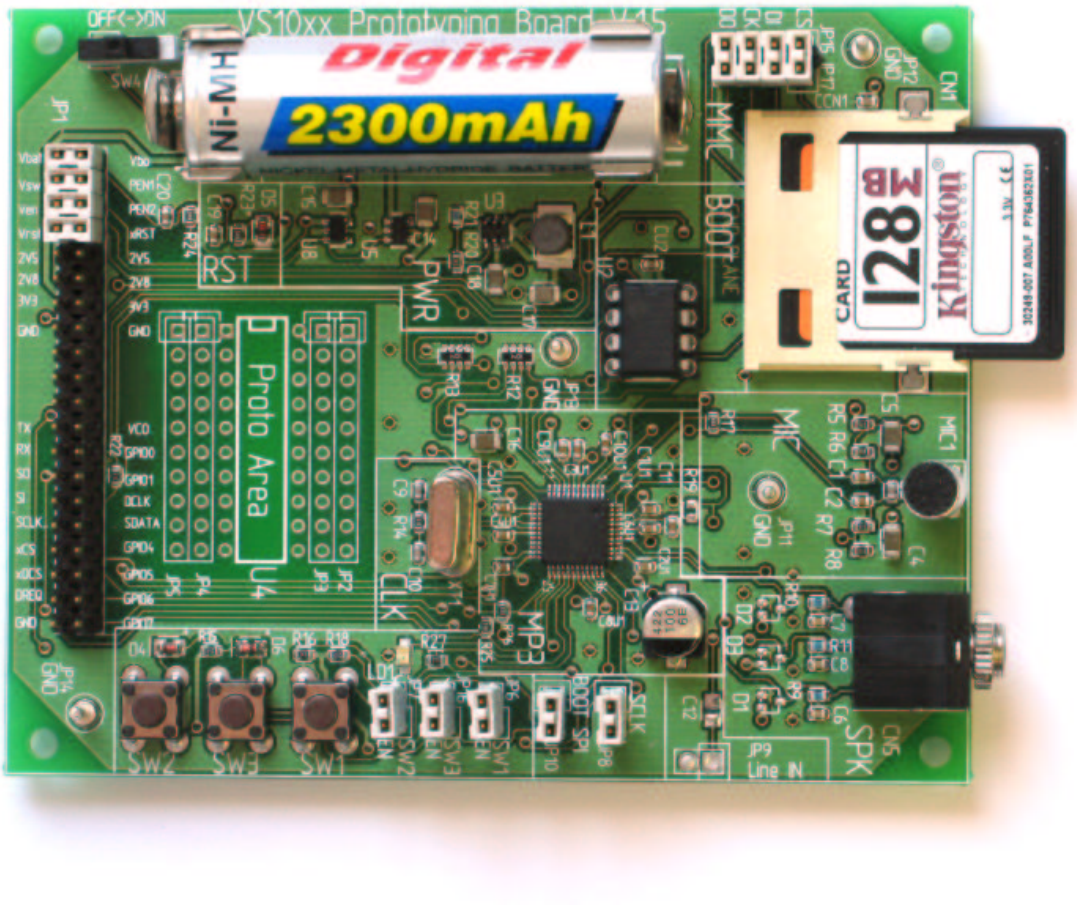
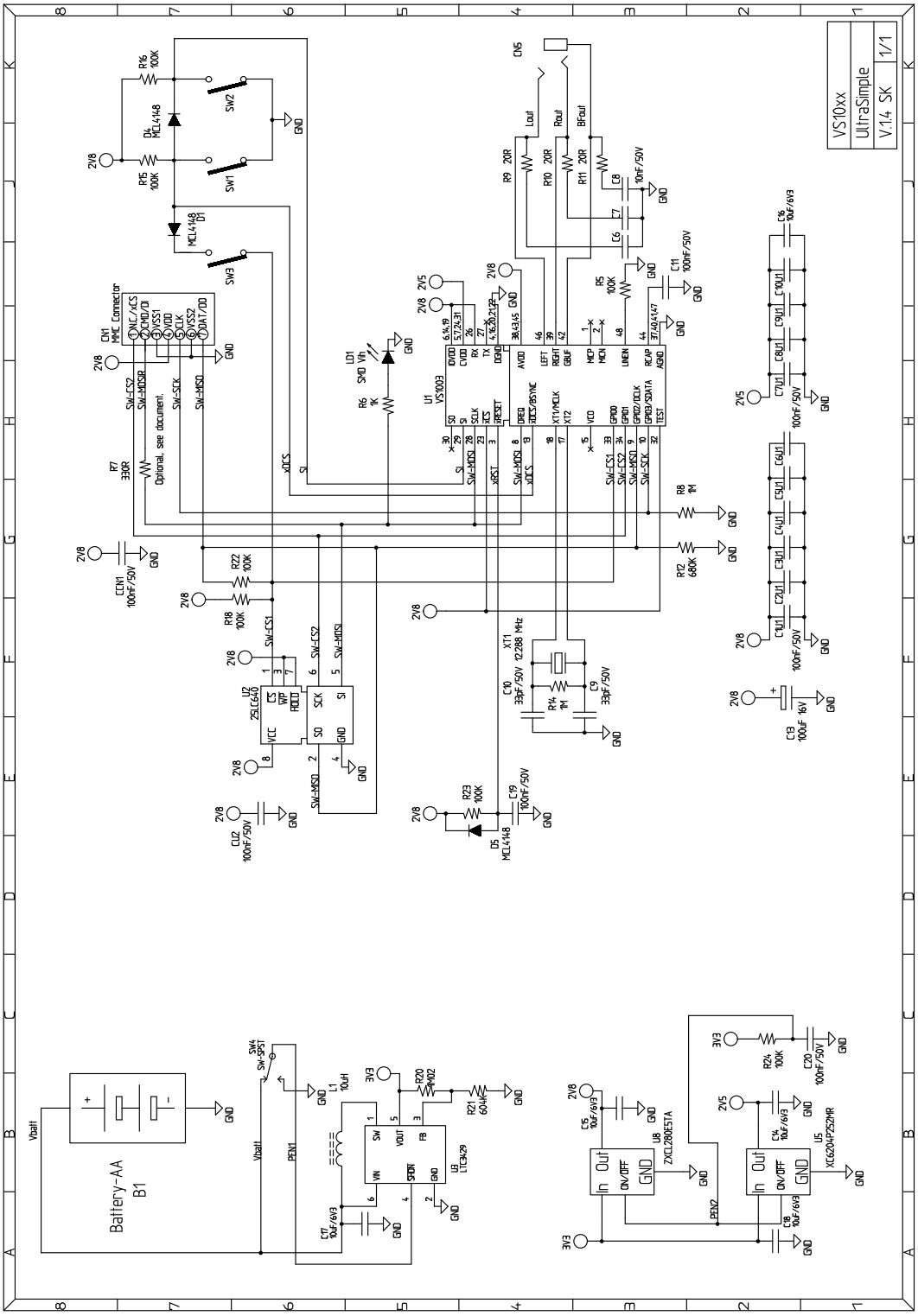


Figure 6.1: Standalone Player in Prototyping Board

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.**

VS1002D version differs by having no separate IOVDD and CVDD, and no line input. VS1011E has no microphone input either.



Note: **MMC's /CS and CLK swapped.** Optional resistor fixes problems with some MMC's (chapter2). See also Figure 3.1.

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.

7 Document Version Changes

7.1 Version 1.13, 2007-01-23

- Analog drivers are kept inactive until a MMC/SD card is found.
- SCI-controlled version now skips ID3V2 tags.
- SCI-controlled version increments decode time between repeats when in looped play mode. Thus the decode time increases even for short (less than one second) WAV files.
- VS1033c version added.

7.2 Version 1.12, 2006-07-13

- MMC communication changed. Now supports some problem MMC's.
- mkrecord.c changed to handle large WAV's better (unsigned size calculation).
- Sample rate and mic/line selection can now be configured from the boot image for the Standalone Recorder.
- Standalone Recorder now has pause in VS1003B version.
- The maximum playable/recordable filesize increased from around 544 MB to over 1 GB. (Filesize was limited in 1.11 when more fragments were supported.)

7.3 Version 1.11, 2006-05-08

- Standalone Recorder auto gain maximum set to 64× (0xffff). It can be changed from the EEPROM image.
- SCI-controlled VS1002D version song change fixed (bug introduced in 1.02).
- SCI_AICTRL1 is zero while the directory is being processed.
- Large FAT32 directories tend to be very fragmented. Now handles maximum of 35 fragments (was 16).
- Pull-down resistors added to GPIO2 and GPIO3.

7.4 Version 1.10, 2006-03-10

- Standalone Recorder has less player features to make recording possible.
- Standalone Recorder sample rate set to 8000 Hz.
- VS1011B and VS1023 versions removed
- Added SPI-EEPROM boot images for SCI-controlled version.
- A bug that could trash the directory fixed in Standalone Recorder.

7.5 Version 1.02, 2005-11-17

- Now detects the file type from 8.3-character filename suffix. Only matching files are processed.
  - VS1011E and VS1002D: .MP3 .WAV
  - VS1003B: .MP3 .WAV .MID .WMA .WMV .ASF
  - VS1033C: .MP3 .WAV .MID .WMA .WMV .ASF .AAC .MP4 .M4A

  If the file name suffix matches but still nothing decodable is found from the first 400 kB, the file is skipped.

7.6 Version 1.01, 2005-11-07

- Song numbering with IMA-ADPCM playback fixed - only VS1002D was affected.
- Watchdog was not activated in the SCI-controlled version 1.00. Note that for SCI-controlled version you need to reactivate the player after watchdog reset.
- Added chapter 8 to explain the playing order of files.

7.7 Version 1.00, 2005-09-30

- **SCHEMATICS CHANGED!**
  - MMC's CLK and /CS have been swapped. Everything else is the same.
  - MMC no longer interferes with SPI boot on powerup.
  - The swap can be done in existing boards with two wires in JP15 and JP17. See section 2.1.
- VS1011E/VS1002D transfer speed increased to 4.1 Mbit/s
- Support for FAT12/FAT16 and FAT32 disks that have no partition block.
- Random play can now select all of the supported 32768 files.

7.8 Version 0.95, 2005-09-16

- Power-on startup delay (default 22 ms) added to reduce power consumption spike.
- If nothing playable is found in the first 300 kB of a file, the rest is skipped. This takes approximately 3 seconds per file.
- SCI-controlled version fixes:
  - Initialization of SCI registers left to user (BASS,CLOCKF,VOL)
  - SCI control fixed: SCI interrupts were only enabled in vs1011 version
- Transfer routine fixes:
  - Internal timing changed to support more MMC/SD cards
  - Transfer speed reduced to 4.78 Mbit/s in VS1003B/VS1033 versions

7.9 Version 0.94, 2005-08-24

- FAT 8.3-character filename readable in SCI-controlled version (see section 5.1)
- Subdirectories are now supported for FAT32 and FAT16 filesystems
  - Upto 16 levels of subdirectories can be used
  - Files and subdirectories are played in the order they appear in the filesystem structures
- Partial FAT12 support:
  - Unfragmented files in FAT12 root directory are supported

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.

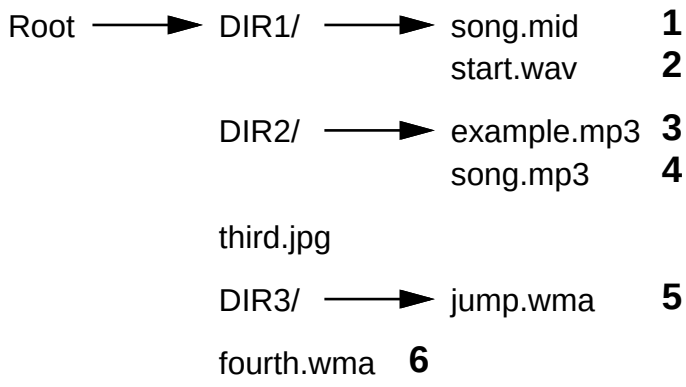


Figure 8.1: Play Order with subdirectories

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.

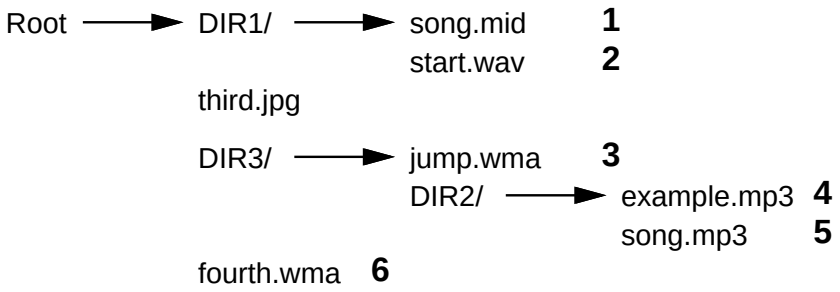


Figure 8.2: Play Order with nested subdirectories