VS10XX REAL-TIME MIDI INPUT

VSMPG “VLSI Solution Audio Decoder”

Project Code:
Project Name: VSMPG

All information in this document is provided as-is without warranty. Features are subject to change without notice.

Revision History			
Rev.	**Date**	**Author**	**Description**
1.1b	2015-09-18	PO	Raises DREQ after starting. RTMIDI UART speed fixed.
1.0	2013-02-26	PO	UART FIFO size increased. Added vs1053b version.
0.9	2009-02-11	PO	Uninitialized register variable fixed.
0.8	2006-11-20	PO	Initial version

1 VS10xx Real-Time Midi Input

All information in this document is provided as-is without warranty. Features are subject to change without notice.

With a small addition to the prototyping board / standalone player board, it can be used as a standalone real-time MIDI synthesizer, that takes its input from a generic MIDI keyboard. The following circuit implements a MIDI IN connection that can be used with all standard MIDI equipments that has a MIDI OUT connection.

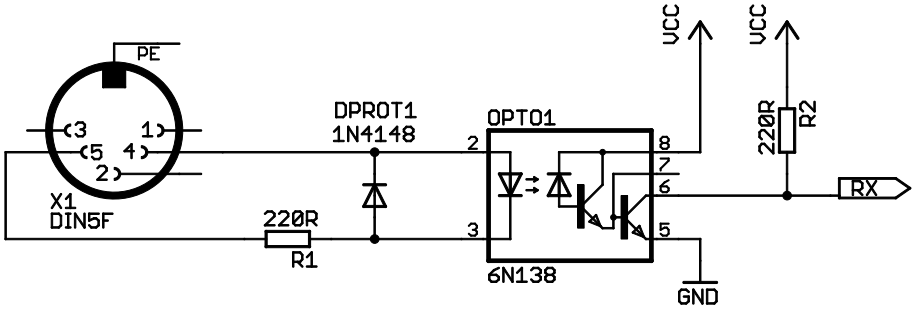

Figure 1: MIDI IN

This is very convenient way to compose MIDI music intended to be played with the VS1003B or VS1033C. Optimizing and fine-tuning the music to the specific chip will become faster and easier.

In addition to the handling of real-time MIDI IN from UART (31250 bps), the RT-Midi code also handles real-time input from the serial data interface (SDI). Both UART and SDI inputs can be used with the same code, but they should not be used simultaneously.

When SDI is used, 0x00 must be sent first, then the actual MIDI byte. This guarantees that the midi command goes straight to processing and does not get stuck waiting in the 16-bit SDI receive register.

In VS1053b and VS1103b the real-time MIDI mode is available in the firmware ROM, and can be started by having GPIO pins in specific states at reset or by a small patch code.

However, the vs1053b MIDI parser does not know how to parse sysex messages. Also, the UART buffer is too small, so if there are a lot of notes playing, it can overflow. So, we're including a version for vs1053b that knows how to skip sysex messages and has a longer UART buffer.

1.1 Boot Images

The real-time midi input 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 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.**

For RT-MIDI with VS1053b and VS1103b also consult their respective datasheets.

Chip	File	Features
VS1003B	`rtmidi1003b.bin`	Version for VS1003B
VS1033C	`rtmidi1033c.bin`	Version for VS1033C
VS1053B	`rtmidi1053b.bin`	Version for VS1053B

The input clock is assumed to be 12.288 MHz. The SCI_CLOCKF value is 0xc000 (4.0× 12.288 MHz, 4.5× for vs1053b). Volume (SCI_VOL) default value is 0x0303, i.e. -1.5 dB.

1.2 Loading Through SCI

The software can also be loaded through SCI in the same way as all patch codes. The application loading tables and plugin files are available in the `code/` subdirectory. After the code is loaded, it is started by writing 0x30 to SCI_AIADDR (0x50 for vs1053b). The code clears SCI_AIADDR automatically after start. To return to normal decoding mode give a software reset.

Chip	File	Features
VS1003B	`rtmidi1003b.plg`	Plugin format for VS1003B
VS1033C	`rtmidi1033c.plg`	Plugin format for VS1033C
VS1053B	`rtmidi1053b.plg`	Plugin format for VS1053B
VS1003B	`rtmidi1003b.c`	Loading table format for VS1003B
VS1033C	`rtmidi1033c.c`	Loading table format for VS1033C
VS1053B	`rtmidi1053b.c`	Loading table format for VS1053B

2 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);
      }
    }
  }
}
```

3 How to Use Old Loading Tables

Each patch contains two arrays: `atab` and `dtab`. `dtab` contains the data words to write, and `atab` gives the SCI registers to write the data values into. For example:

```
const unsigned char atab[] = { /* Register addresses */
    7, 6, 6, 6, 6
};
const unsigned short dtab[] = { /* Data to write */
    0x8260, 0x0030, 0x0717, 0xb080, 0x3c17
};
```

These arrays tell to write 0x8260 to SCI_WRAMADDR (register 7), then 0x0030, 0x0717, 0xb080, and 0x3c17 to SCI_WRAM (register 6). This sequence writes two 32-bit instruction words to instruction RAM starting from address 0x260. It is also possible to write 16-bit words to X and Y RAM. The following code loads the patch code into VS10xx memory.

```
/* A prototype for a function that writes to SCI */
void WriteVS10xxRegister(unsigned char sciReg, unsigned short data);

void LoadUserCode(void) {
  int i;
  for (i=0;i<sizeof(dtab)/sizeof(dtab[0]);i++) {
    WriteVS10xxRegister(atab[i]/*SCI register*/, dtab[i]/*data word*/);
  }
}
```

Patch code tables use mainly these two registers to apply patches, but they may also contain other SCI registers, especially SCI_AIADDR (10), which is the application code hook.

If different patch codes do not use overlapping memory areas, you can concatenate the data from separate patch arrays into one pair of `atab` and `dtab` arrays, and load them with a single `LoadUserCode()`.

4 Document Version Changes

Version 1.1b, 2015-09-18

- Now raises DREQ after starting.
- The UART rate for RTMIDI was wrong (9600bps), it is now the correct 31250bps.

Version 1.0, 2013-02-26

- UART FIFO size increased.
- Added the vs1053b version.

Version 0.9, 2009-02-11

- Fixed uninitialized variable from the state machine. Now loading through SCI works also.
- Added the application in plugin format.
- Added “How to load a plugin” and “How to use old loading tables” chapters.

Version 0.8, 2006-11-20

- First version
  - VS1003B and VS1033C versions.