# VS1003B DTMF Examples

VSMPG "VLSI Solution Audio Decoder"

Project Code:
Project Name: VSMPG

| Revision History | | | |
|---|---|---|---|
| **Rev.** | **Date** | **Author** | **Description** |
| 0.5 | 2009-02-02 | PO | Initial version |

# 1 VS1003B Code Examples

This package contains a working setup for user code development for VS10xx. In addition you need VSKIT and a perl interpreter. Read the tools manual from the VSKIT distribution for general VSDSP coding tips, and tool usage documentation.

Included:

- Makefile - project makefile for command file
- mem_desc.vs1003 - VS1003B memory map for linking
- rom1003.txt - absolute address definitions for VS1003B
- vs1003.h - the most useful definitions and function prototypes
- c.s - ASM stub and hookup code, ADC interrupt stub
- mycodec.c - C code for startup, idle hook, and user codec
- coff2cmd.pl - converts a COFF executable into command file
- cmdtotab.pl - converts a command file into C arrays

Generated:

- mycodec.bin - application executable
- mycodec.cmd - application loading command script
- mycodec.plg - application plugin tables
- mycodectab.c - application loading tables as C code

This DTMF example generates versions for vs1011e, vs1002d, vs1003b, vs1033c, vs1033d, and vs1053b.

## 1.1 Application Hook

The application hook code receives all samples that are going to the audio buffer. This way the samples can be easily manipulated regardless what the actual audio decoder generates. In this example sine signals are added on top of the original decoded signal.

The application hook is set using the SCI_AIADDR register. In this example c.s contains a jump instruction in a fixed address, so by writing 0x30 (0x50 for VS1053) to SCI_AIADDR will enable calling our ApplAddr() function, and writing 0 to SCI_AIADDR will disable it.

The application address routine will also be called for other things than audio data, so the mode must be checked. For example when SCI_AIADDR is written (to enable the hook), the application address is called with APPL_RESET.

```
s_int16 ApplAddr(register __i0 s_int16 **d, register __a1 s_int16 mode,
                 register __a0 s_int16 n) {
    if (mode == APPL_RESET) {
        aictrl0 = 0;
        aictrl1 = 0;
        rate = 0;
        memset(sines, 0, sizeof(sines));
    }
    if (mode == APPL_AUDIO) {
    }
    return n;
}
```

In the example APPL_RESET performs a couple of initializations.

## 1.2 Activation and Deactivation

1. Load the plugin after each harware or software reset. (See chapter 2 or 3.)
2. Set appropriate parameters to AICTRL registers.
3. Activate the plugin by writing 0x30 to SCI_AIADDR. (Use 0x50 for VS1053.)

The default compile generates a separate sine to left and right channels.

The frequencies are controlled by AICTRL0 and AICTRL1. The volume of the sines are controlled with AICTRL2. The original signal volume can be controlled using AICTRL3. Both volume controls are linear, so that 0xffff is maximum volume, 0x8000 is -6dB, 0x4000 is -12dB, etc.

For example to get a 2000Hz tone to both channels with -24dB level, set AICTRL0 = 2000, AICTRL1 = 2000, AICTRL2 = 0x1000, AICTRL3 = 0xffff.

You can also change the parameters during sine generation.

To deactivate: write 0 to SCI_AICTRL.

To reactivate: write 0x30 (or 0x50 for VS1053) to SCI_AIADDR.

If you need more accurate frequency control than 1 Hz, change the PH_SCALE define in addsine.s and recompile. The scaling will change by $2^{PH\_SCALE}$, so for example by setting PH_SCALE to 2 you have 0.25 Hz resolution.

# 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()`.