Digital Audio (PCM) Sound Effects Software for Embedded Micro-controller Applications

by Michael J Bauer

Overview

Appealing sound effects are generally too complex to be generated in real time by the target micro-controller of an embedded system. Therefore, sound effects may be synthesized by a purpose-built application and stored as a sequence of PCM samples in a data file. Saved sound effects can be loaded into flash memory in the target embedded system for fast access by an audio player function. The storage medium is typically a "serial data flash" (SDF) memory device interfaced to the MCU via SPI. However, the micro-controller's internal flash program memory can also hold sound FX data.

PCM sound effects data files can also be produced from standard audio file formats such as WAV, MP3, etc, using "Audacity" -- a free open-source audio editor application. There is a plethora of sound effects available for download on the internet. Audacity can also be used to record sounds using a microphone, CD player, sound synthesizer or musical instrument, etc. Once loaded into Audacity, audio data can be re-sampled at the required sample rate and then exported in "RAW" data format (stripped of header information) compatible with the target embedded player.

The software described in this article comprises a simple sound FX synthesizer application, an audio player function and various utilities to transfer sound FX data files between the SDF memory device and a removable mass-storage device, e.g. a USB flash drive. The software was originally developed for Microchip's PIC32MX (32-bit) micro-controller family, but could be easily customized to suit others.

The hardware platform used by the author for development and testing of the sound synthesis software and real-time play-back routine is a PIC32-MX460 development board from Olimex (www.olimex.com). The MCU is a PIC32MX460F512L clocked at 80MHz. The board has a monophonic audio output circuit driven from a PWM pin on the MCU. The output amplifier can drive low-impedance headphones or a small speaker directly.

The Olimex board does not include a serial data-flash memory, but a suitable device can be easily added (see appendix at the end of the document). The board has provision for a micro-SD card which could be used instead of the SPI data-flash device, but this would require substantial modifications to the firmware. Also, depending on whatever file system might be implemented for the micro-SD card, the read access speed may or may not be fast enough to support the audio playback function. Microchip's MDD (Memory Disk Drive) file system supports only one mass-storage medium, which could be either the USB MSD host (USB flash drive) or the micro-SD card, but not both together.

The software includes a code module "sound_synth.c" which provides functions for the synthesis of sound effects, plus a separate code module "pcm_audio.c" providing functions for the playback of sound clips. Another code module "data_flash_fs.c" implements a file system for handling data stored in the SDF memory. The application provides a command-line user-interface (CLI), accessible via a serial port connected to a host PC acting as a console terminal.

Real-time Audio Player software

The code module "pcm-audio.c" provides functions for playback of PCM sound effects from raw sample data stored in external SDF files or in MCU program flash memory.

PCM audio data files are comprised of 16-bit signed integer "samples" with values in the range −32000 to +32000. However, the player function is limited to 12-bit (signed)

sample values in the range −2000 to +2000. The audio dynamic range obtainable is therefore about 2000:1 (= 66dB) which is more than acceptable for this application.

The playback sample rate is fixed at 20ks/s (samples per second), giving a usable audio bandwidth of about 8kHz, also adequate for this application. The sample rate and resolution were chosen to suit the PIC32-MX460 processor clock speed: 80MHz.

For a PWM duty resolution of 1/4000, the Timer period is (4000 / 80M) = 50us, giving the 20kHz sample rate. The Timer is configured to generate a periodic interrupt at this rate. The processor overhead to service this IRQ is about 1% when there is no sound being generated, rising to about 20% when playback is active, i.e. when sample values are being read from external SPI flash memory (at 20MHz SPI clock rate).

Audio signal generation is realized by an Output Compare module (OCx) on the PIC32 which generates a PWM output signal. The audio output signal amplitude is proportional to the PWM duty. Because the analog audio signal is unipolar, i.e. its "zero" level is biased to VDD / 2, bipolar sample values read from the sound clip file must be offset by 2000 counts to give a PWM duty in the range 0 to 4000. Thus, the PWM duty will be 50% for a (bipolar analog) sample value of zero.

The Timer interrupt (50us interval) drives the PWM output, updating the PWM duty (OCx duty register) as each sample is fetched from the sound clip file.

The PWM (OCx) output pin can drive a class-D "amplifier" directly, i.e. half-bridge push-pull or full-bridge differential switch. In the case of push-pull (half-bridge) output, the speaker must be AC coupled to the output. A second-order LC low-pass filter with a cut-off frequency of about 6kHz is recommended to be inserted between the switch output and the speaker to attenuate the 20kHz PWM carrier signal.

Maximum volume can be set using the amplifier supply voltage, or a series resistor. The playback volume level is also settable in the software, of course.

Converting Audio Files to RAW format using Audacity

To convert a standard WAV file for use with the embedded audio player, first open the WAV file in Audacity. Set the Project sample rate to 20,000Hz. From the Track menu, select Re-sample and set the sample rate to 20,000 and the format to 16-bit signed (integer). If the input file is stereo, be sure to change settings to mix the stereo tracks down to a single mono track when exporting. Execute the re-sample function.

From the File menu, select Export Audio. Click Options in the Save dialogue box. Set the export file type to "Other uncompressed formats" and select RAW (no header data). Execute the Export function. Copy the RAW output file to a USB flash drive. The file name must be in FAT32 short form (up to 8 characters plus the ".RAW" extension.)

Insert the USB drive into the board's USB host port and use the console command:

```
import -b <filename.raw> <SDF filename>
```

This will copy the file to the board's SDF memory. The SDF filename can be up to 19 chars (no spaces) and should end with ".pcm" (optionally) to indicate the file type.

Sound clips can be tested using the command:

```
play <SDF fileID#>
```

The `<SDF fileID#>` number is obtained using the command:

```
sdf -l
```

… to list files in the SDF memory. Type "sdf" without any arg's to see all options.

Sound Synthesizer Software

The sound synthesizer application is driven by a command-line user interface (CLI) which communicates on the "RS232" serial port (UART2). A cheap USB/Serial cable can be used to connect the Olimex board to a PC. A terminal emulator app is required to be installed on the host PC. PuTTY-Tel is highly recommended (and it's free to download). Set it up for serial terminal mode at 57600 baud.

The console CLI command "synth" allows short audio clips of up to 5 seconds in duration to be synthesized. (The maximum clip duration can be easily changed, if required.) Generated sample data are stored in a buffer as 16-bit signed integer "samples". The audio player function uses sample values in the range −2000 to +2000 (at a sample rate of 20kHz). The audio dynamic range obtainable is therefore 2000:1 (= 66dB) which is more than acceptable for this sort of application.

The synthesizer generates a sequence of audio samples in a temporary buffer implemented as a file in external SPI flash memory. Completed sound effects can be saved to a file on a USB drive inserted into the USB host port on the board. There is also a command to dump the output buffer as a C source code definition (initialized array of 16-bit integers), for target applications which will have the sound clip data stored in the PIC32 on-chip program flash memory.

The software-synthesized sound is characterized by a set of user-supplied parameters modeling traditional analog sound synthesizers. The basic model, Patch #1 – "QAM" illustrated below, provides two "oscillators" each covering a frequency range of 0.01Hz ~ 5000Hz feeding into a "ring modulator" (4-quadrant multiplier) followed by an amplitude "envelope shaper" (with variable "hold-off", "attack", "sustain", and "release" times). Envelope parameters are in units of milliseconds (up to 5000).

The two oscillators have a choice of output waveform (sine, triangle, square and saw-tooth) with variable output level (bipolar, 0 to ±1.0) and DC offset (0 to ±1.0). If only one oscillator is required, the output level of the other oscillator is set to zero and its offset parameter is set to full-scale (1.0).

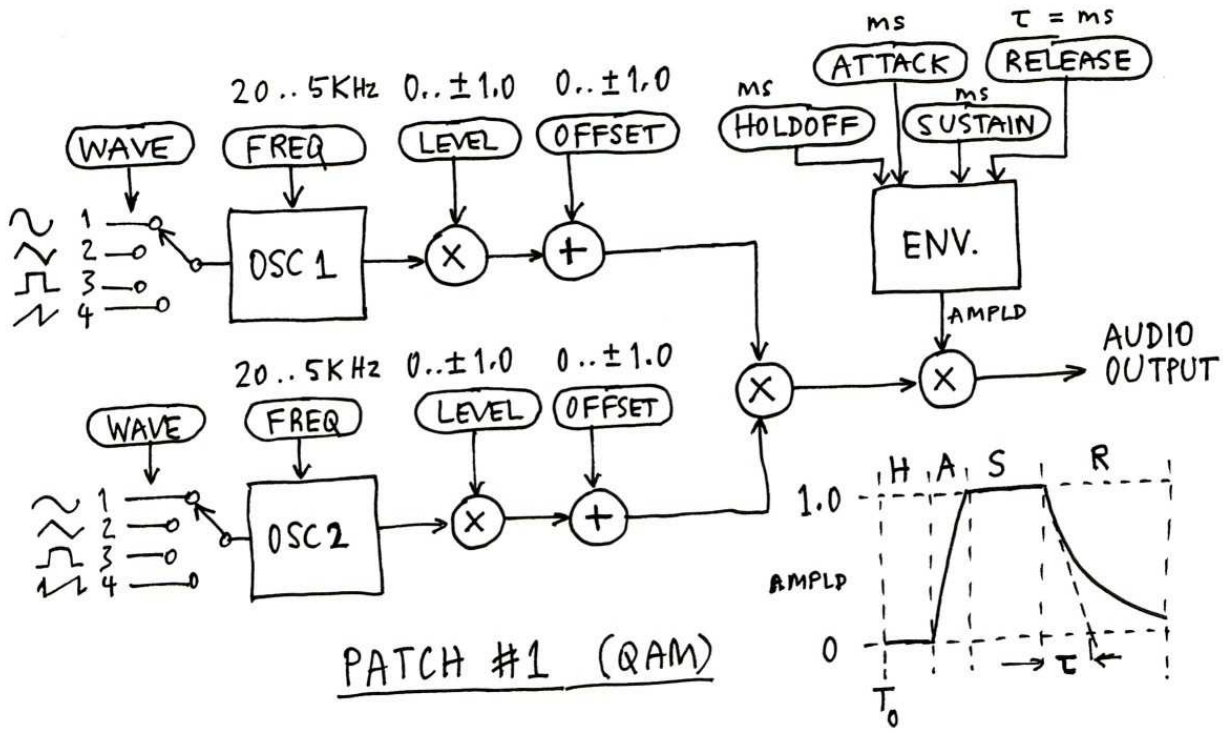
Note that signals in the synthesizer model are floating-point numbers, normalized to a full-scale magnitude of one (i.e. 1.0). The full-scale signal value (1.0) represents an actual PCM sample value of 30,000 in the output file. The audio player function scales the 16-bit sample values to 12-bit values to suit the PWM duty resolution.

The envelope shaper (ENV) is simply a pulse generator with variable rise (attack) time and fall (release) time. The peak amplitude of the envelope signal is 1.0V.

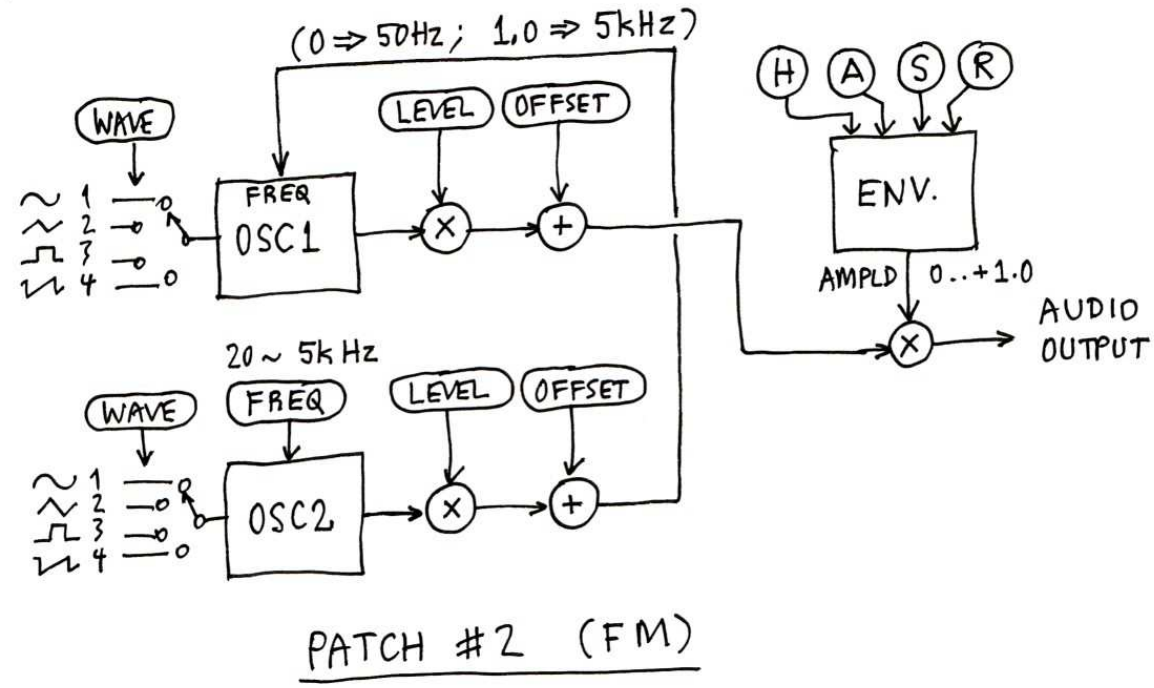
The envelope shaper "release" parameter is actually a time-constant representing the rate of exponential decay of the audio output amplitude. As a general rule, allow at least 5 times the time-constant value for the signal to decay to near zero. For example, if the required "release" time is 500ms, set the RELEASE parameter value to 100ms or less.

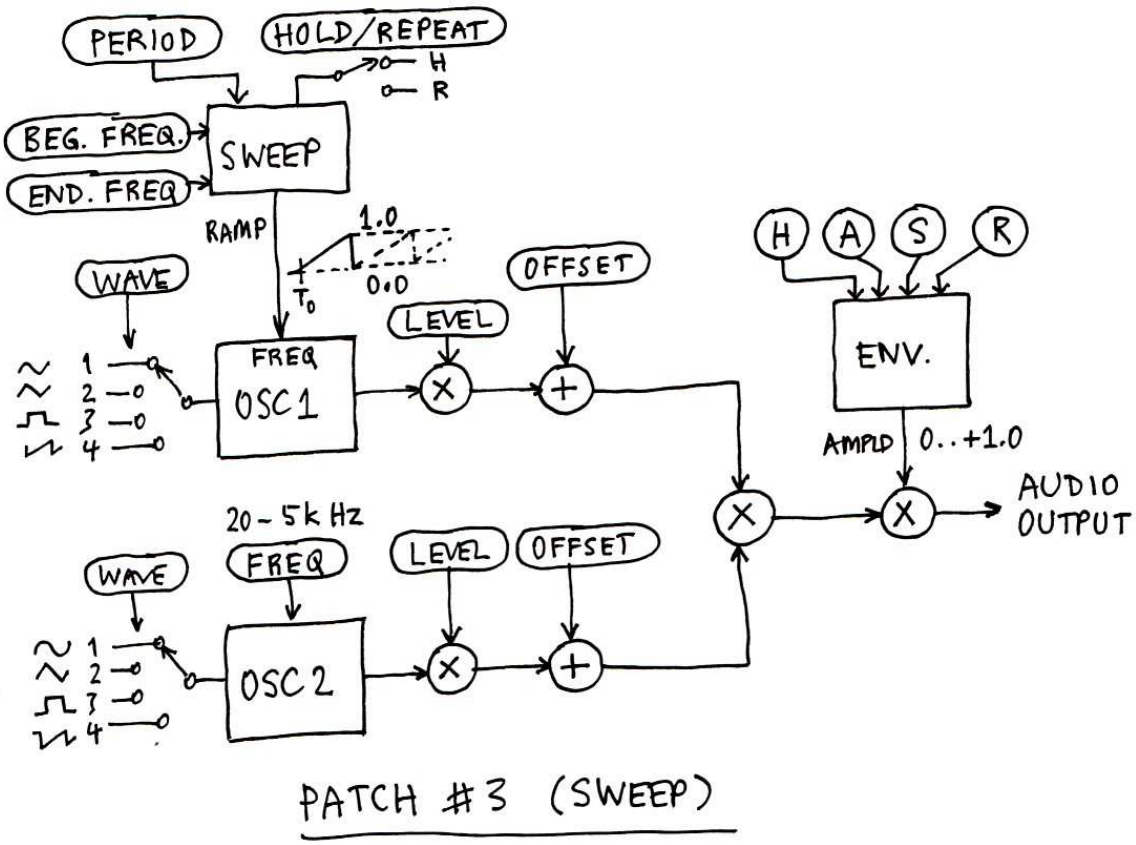
The peak amplitude can be sustained for a specified time, called the "sustain" time, before the "release" phase of the envelope begins. In addition, the "attack" can be delayed by a specified time, called the "hold-off" time. The purpose of the hold-off parameter will be explained later when "multi-layering" of sounds is described.

The oscillators can be "patched" so that OSC1 frequency is controlled (modulated) by the output level of OSC2. This is Patch #2 – "FM" (illustrated below). The FM topology can produce a range of "clangorous" (bell-like) tones, among other "interesting" effects. It takes some experimentation to discover the full gamut of sounds which can be produced by this method.



Frequency Modulation is obtained using "Patch #2" (illustrated below), i.e. set the PATCH parameter to 2. The LEVEL and OFFSET parameters of OSC2 are used to set the modulation depth and median frequency (resp.).





A third patch option, Patch #3 -- "SWEEP" illustrated above, provides a "sweep generator" (ramp waveform) to control the frequency of OSC1. The frequency at the start of the ramp and frequency at the end of the ramp are settable parameters, as is the time interval of the sweep/ramp. What happens at the end of the sweep is determined by a Boolean parameter, "sweep hold". If SWEEP-HOLD is set to '1' (True), the frequency of OSC1 at the end of the sweep is held for the remainder of the duration of the clip. Otherwise, if SWEEP-HOLD is set to '0' (False), the sweep ramp will repeat over the duration of the clip.

The synthesizer allows "multi-layering" of sounds, i.e. mixing of two or more "layers" of sample data, to create more complex sound effects. Moreover, layers can be "time-shifted" relative to each other by delaying the envelope "attack". This is the intended purpose of the "hold-off" parameter. For example, to create a door-bell effect, two layers can be generated, one for the "ding" and one for the "dong". A short delay between the start of the "ding" and start of the "dong" is achieved by setting an appropriate value for the "hold-off" time of the "dong" layer (e.g. 200ms).

Software Synthesizer Operation

The synthesizer is operated with a command-line user interface. Various sound effects can be produced according to the values of user-settable parameters.

Sample data held in the SDF output buffer can be written to a file on a USB drive inserted in the USB host port. Output files can be in RAW format (compatible with Audacity raw import), or in Intel Hex (INHX32) file format. The application implements a simple file system for managing data files in the SDF memory.

Another output option allows buffered sample data to be output in C source code syntax for the definitoin of an initialized array of (short) integers. This would be appropriate for very short audio clips to be stored in MCU on-chip program flash memory.

Current layer parameter values are listed using the "param" command (alias “set”) as shown below.

```
> param
duration   =     1000    |  patch      =        1
osc1wave   =        1    |  osc2wave   =        1
osc1freq   =     1000    |  osc2freq   =  500.000
osc1level  =    1.000    |  osc2level  =    0.000
osc1ofst   =    0.000    |  osc2ofst   =    1.000
holdoff    =        0    |  attack     =       10
sustain    =      900    |  release    =       20
sweepbegf  =      200    |  sweependf  =     4000
sweeptime  =     1000    |  sweephold  =        1
volume     =       70    |
```

The units and ranges of synth parameters are as follows:

Parameter	Unit	Range / options
Duration	ms	1 .. 5000 (overall clip duration)
Patch	--	1 = QAM, 2 = FM, 3 = SWEEP
Osc1wave	--	1 = sine, 2 = triangle, 3 = square, 4 = sawtooth
Osc1freq	Hz	0.01 .. 5000
Osc1level	V	0 .. ±1.0
Osc1ofst	V	0 .. ±1.0
Osc2wave	--	1 = sine, 2 = triangle, 3 = square, 4 = sawtooth
Osc2freq	Hz	0.01 .. 5000
Osc2level	V	0 .. ±1.0
Osc2ofst	V	0 .. ±1.0
Holdoff	ms	0 .. 5000
Attack	ms	0 .. 5000
Sustain	ms	0 .. 5000
Release	ms	0 .. 5000
Sweepbegf	Hz	20 .. 5000
Sweependf	Hz	20 .. 5000
Sweeptime	ms	0 .. 5000
Sweephold	--	0 = repeat sweep, 1 = hold end freq.
Volume	%	0 .. 100 (square-law curve)

Parameter values may be set using either of these command aliases:

```
param name [=] value

set name [=] value
```

… where 'name' is the same as listed by the 'param' command (above). Parameter names are not case-sensitive when entered into the command line. The equality symbol is optional. If you try to enter a value which is out of range, the software will complain and leave the current value as-is.

After setting parameters for the desired sound effect, the command "synth -g" is used to generate sample data and store it in a temporary buffer called the "layer buffer". The command "synth -t" may be issued to test the sound in the buffer, i.e. to play it directly from the layer buffer. Before the sound in the layer buffer can be written to a file, it should be merged into another buffer called the "output buffer".

To produce multi-layered sound effects, an output buffer (file) is defined on the SDF memory. The command "synth –m" merges the layer buffer (produced by the command "synth –g") with the output buffer. Optionally, a new "layer" can then be created and merged into the output buffer. The merge process can be repeated any number of times until the desired multi-layer sound effect has been produced.

The command "synth -g" generates a new sound "layer" using the current synth parameter values. The merge formula is a simple addition followed by a "normalization" process which scales (i.e. multiplies) all samples in the buffer by a constant value such that the maximum sample amplitude is nearly full-scale (allowing a little "headroom").

An "undo" command "synth –u" is provided in case a merged sound is not what was desired. There is only one level of "undo", i.e. only the single most recent merge operation can be reversed.

When you are satisfied with a particular sound effect and wish to save it for use in an application, the output buffer can be saved to a raw PCM data file on the USB drive, or it may be output as a text file (C source code format) for inclusion in a firmware compilation.

To save the sound clip to a RAW file on the USB drive, use the command:

```
synth –s [clip_name]
```

… where <clip_name> can be up to 19 characters, except no colons or spaces allowed. Otherwise, use general file naming conventions. The RAW output file name is generated automatically using a random number. If supplied, the clip name is written to another file with the same name as the RAW output file, but with extension ".SID" (Sound ID). The SID file is a text file containing the clip name, clip length, layer parameters, etc.

To get a summary of "synth" command options, type the command with no arguments, as follows:

```
> synth
Usage:  synth  <option>  [arg]
<option>----------------------------------------
  -l :  List layer parameters (verbose)
  -n :  New sound, current layer = 1
  -g :  Generate sample data in current layer
  -m :  Merge current layer into output buffer
  -u :  Undo last merge
  -t :  Test sound in current layer
  -p :  Play merged sound in output buffer
  -s :  Save clip to USB drive as RAW file...
```

```
          <arg> = clip_name (optional)
          MDD file-name generated automatically
  -x :  Save clip to USB drive as HEX file...
          <arg> = clip_name (up to 19 chars),
          MDD file-name generated automatically
  -d :  Dump sound clip buffer as C array def'n
          <arg> = clip_name (optional)
  -v :  View values for patch, osc#wave, etc.
-------------------------------------------------
```

There is a command "sdf" provided for the purpose of managing files on the SDF memory device. Type "sdf" (with no arg's) to get this summary:

```
> sdf
Usage:  sdf  <opt> [arg2] [arg3]
<opt> =
 -f : Format (erase) SDF memory, arg2 = '+++'
 -l : List SDF directory (file index)
 -e : Export bin file from SDF memory to USB drive,
        arg2 = SDF fid#, arg3 = MDD filename (8.3)
 -i : Import bin file from USB drive to SDF memory,
        arg2 = MDD filename (8.3), arg3 = SDF filename
 -t : Export Text from file on SDF to USB drive,
        stop at EOT ($FF), args as in <-e> option.
 -x : Import Hex file on USB drive to SDF bin file,
        arg2 = MDD filename, [arg3 = SDF filename]
 -h : Hide file, arg2 = SDF fid#
 -m : Make file, arg2 = name, arg3 = size (sectors)
 -k : CheckK SDF file system integrity
```

The command "sdf -i" is used to import a RAW (binary) sound FX file from MDD media (USB drive) to SDF memory. The SDF filename can be up to 19 chars (no spaces) and should end with ".pcm" (optionally) to indicate the file type.

The command "sdf -x" is used to import a HEX format sound FX file from MDD media (USB drive) to SDF memory, converted to binary format. The source file must be in Intel Hex (INHX32) format. If the hex file contains a title record with the name of the sound clip, it is optional to supply "arg3" (clip name) in the command line. The clip name found in the title record, if any, will be used to name the destination file in SDF memory. If "arg3" is supplied, it will be used as the SDF file name; the title record will be ignored.

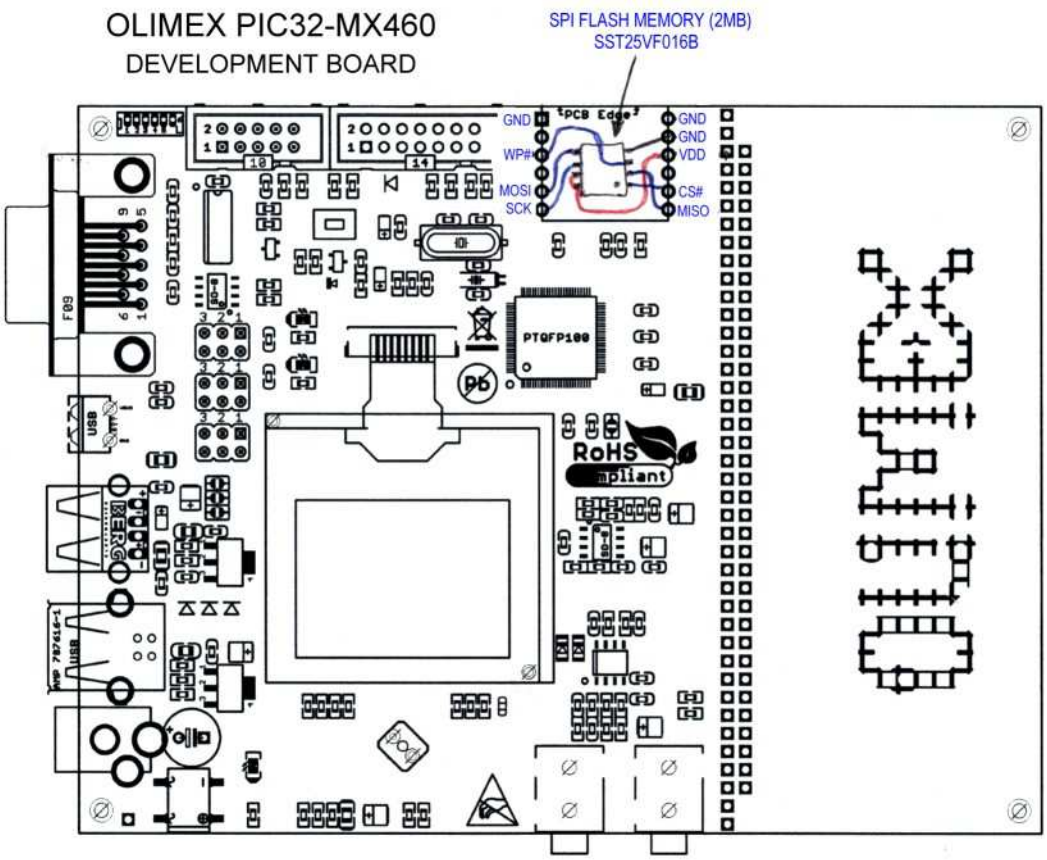
After importing a saved sound clip file from the USB drive to SDF memory, the clip can be played back using the command "`play <file_ID>`"… where `<file_ID>` is a number identifying the file. (Note: The "play" command is not meant to be used to play sounds stored in the temporary synth buffers, because in general the sound clip duration will be less than the buffer size, i.e. less than the maximum defined length.)

Use the command "`sdf -l`" to list files on the SDF memory, with their ID numbers.

Use the command "`ls -l`" to list files on the MDD media device (USB drive). Hex sound FX files with a clip name (title record) will be listed with their clip name alongside.

Appendix: Olimex board modification, SPI Flash Memory

Add an SPI flash memory device SST25VF016B (8-lead SOIC) in place of the RF module. Stick the chip to the board with hobby glue or double-sided tape and solder hookup wires as shown in the picture below. (If you want to use the RF module, place the flash memory device elsewhere on the board and use spare I/O pins for CS# and WP#. The firmware will need to be modified accordingly.)



Hardware Real-Time Clock Chip

Microchip's MDD media FAT32 file system uses the system time and date to "time-stamp" files whenever they are created or modified. The synthesizer application provides a command "time" to set the system time and date. To avoid having to set the time and date every time the board is powered up, a battery-backed real-time clock-calendar device (RTCC) can be interfaced to the MCU via an I2C bus. Suitable RTCC modules are available from Olimex (https://www.olimex.com/Products/Modules/Time/) and many other suppliers (e.g. www.futurlec.com or www.microchip.com).

The software checks for presence of a hardware RTCC on power-up and, if it finds one, automatically synchronizes the system time and date to the RTCC. The software is configurable to support either MCP79410 or ISL1208 type devices. (MCP79410 is recommended. The pre-built firmware (hex file) in the distribution package supports the MCP79410 device. The firmware can be easily modified to suit other RTCC devices.)