Introduction

The Audio Engine post-processing on STM32F4xx - Sampling rate conversion SRC441 library user manual describes the software interface and requirements of the module into a main program and provides a rough understanding of the underlying algorithm.

The SRC441 library is used to convert the sampling frequency from 44.1 kHz to 48 kHz.

The SRC441 library is part of the STM32-AUDIO100A firmware package.
## Contents

1. **Module overview** ................................................. 6
   1.1 Algorithm function ......................................... 6
   1.2 Module configuration ....................................... 6
   1.3 Resource summary .......................................... 6

2. **Module Interfaces** ........................................... 8
   2.1 APIs .......................................................... 8
      2.1.1 src441_reset function ................................ 8
      2.1.2 src441_setParam function ......................... 8
      2.1.3 src441_getParam function ......................... 9
      2.1.4 src441_setConfig function ...................... 9
      2.1.5 src441_getConfig function ...................... 9
      2.1.6 src441_process function ......................... 10
   2.2 External definitions and types .......................... 10
      2.2.1 Input and output buffers .......................... 10
      2.2.2 Returned error values .............................. 11
   2.3 Static parameters structure ............................ 11
   2.4 Dynamic parameters structure ........................... 12

3. **Algorithm high level view** ............................... 13
   3.1 Processing steps .......................................... 13
   3.2 Data formats ............................................... 13
   3.3 Performance measurements ................................ 14
      3.3.1 SINAD measurements ................................ 14
      3.3.2 Frequency response measurements ............... 15

4. **System requirements and hardware setup** ............... 16
   4.1 Recommendations for optimal setup .................... 16
      4.1.1 Memory allocation ................................... 16
      4.1.2 Module API calls .................................... 16
      4.1.3 Module integration summary ...................... 17

5. **How to tune and run the application** .................. 19
6 Revision history ........................................... 20
List of tables

Table 1. Resource summary ......................................................... 7
Table 2. src441_reset ................................................................. 8
Table 3. src441_setParam ............................................................ 8
Table 4. src441_getParam ............................................................ 9
Table 5. src441_setConfig .......................................................... 9
Table 6. src441_getConfig .......................................................... 10
Table 7. src441_process ............................................................. 10
Table 8. Input and output buffers .................................................. 11
Table 9. Returned error values ...................................................... 11
Table 10. Static parameters structure ........................................... 11
Table 11. Dynamic parameters structure ....................................... 12
Table 12. SINAD values ............................................................. 14
Table 13. Frequency response values .......................................... 15
Table 14. Document revision history ............................................ 20
List of figures

Figure 1. SRC441 module ................................................................. 13
Figure 2. API call procedure .......................................................... 17
1 Module overview

1.1 Algorithm function

The SRC441 module provides functions to handle the sampling rate conversion of mono and stereo signals from 44.1 kHz to 48 kHz.

*Note:* Conversions with other ratios are handled by another module named SRC236.

1.2 Module configuration

The SRC441 module supports mono and stereo interleaved with either 16-bit or 32-bit I/O data, with an input frame size of 147 samples (corresponding to 3.33 ms). Processing is then looped three times to resample exactly 10 ms of data.

Four versions of the module are available:

- **lib_src441_m4.a:** Standard configuration for low-MIPS, good quality requirements and 16-bit input/output buffers.
- **lib_src441_HQ_m4.a:** reserved for high quality needs (consumes more MIPS and memory as well), with 16-bit input/output buffers.
- **lib_src441_32b_m4.a:** Standard configuration for low-MIPS, good quality requirements and 32-bit input/output buffers.
- **lib_src441_HQ_32b_m4.a:** reserved for high quality needs (consumes more MIPS and memory as well), with 32-bit input/output buffers.

1.3 Resource summary

*Table 1* contains the module requirements for the Flash, stack and RAM memories, and frequency (MHz).

The required core frequency (MHz) is estimated using EWARM v6.50 profiler, while in parenthesis values have been measured on real hardware, running on STM32F407IG chipset.
## Table 1. Resource summary

<table>
<thead>
<tr>
<th>Version</th>
<th>SRC441</th>
<th>Flash code (.text)</th>
<th>Flash data (.rodata)</th>
<th>Stack</th>
<th>Static RAM</th>
<th>Dynamic RAM</th>
<th>Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Standard with 10 ms buffers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>measured on hardware</td>
</tr>
<tr>
<td>Standard</td>
<td>44.1 to 48</td>
<td>3184 Bytes</td>
<td>8 Bytes</td>
<td>232 Bytes</td>
<td>3228 Bytes</td>
<td>Stereo: 13.4 MHz (14 MHz) Mono: 9.3 MHz (9.9 MHz)</td>
<td></td>
</tr>
<tr>
<td>High quality</td>
<td></td>
<td>3878 Bytes</td>
<td></td>
<td>360 Bytes</td>
<td>3676 Bytes</td>
<td>Stereo: 18.8 MHz (20.7 MHz) Mono: 13.4 MHz (15.3 MHz)</td>
<td></td>
</tr>
<tr>
<td>Standard 32-bit I/O</td>
<td>3204 Bytes</td>
<td></td>
<td></td>
<td>232 Bytes</td>
<td>3228 Bytes</td>
<td>Stereo: 13.1 MHz (13.7 MHz) Mono: 9.2 MHz (9.8 MHz)</td>
<td></td>
</tr>
<tr>
<td>High quality 32-bit I/O</td>
<td>3894 Bytes</td>
<td></td>
<td></td>
<td>360 Bytes</td>
<td>3676 Bytes</td>
<td>Stereo: 18.5 MHz (20.4 MHz) Mono: 13.3 MHz (15.2 MHz)</td>
<td></td>
</tr>
</tbody>
</table>
Module Interfaces

Two files are needed to integrate the SRC441 module: `lib_src441_XXX_m4.a` and the `src441_glo.h` header file which contains all definitions and structures to be exported to the framework.

Note: The `audio_fw_glo.h` file is a generic header file common to all audio modules; it must be included in the audio framework.

2.1 APIs

Six generic functions have a software interface to the main program. They allow the developer to initialize, reset, set and get parameters.

2.1.1 src441_reset function

This procedure initializes the static memory of the SRC441 module, and initializes static and dynamic parameters with default values.

```c
int32_t src441_reset(void *static_mem_ptr, void *dynamic_mem_ptr);
```

Table 2. src441_reset

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Input</td>
<td>dynamic_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal dynamic memory</td>
</tr>
<tr>
<td>Returned value</td>
<td>int32_t</td>
<td>Error value</td>
<td></td>
</tr>
</tbody>
</table>

This routine must be called at least once at initialization time, when the real time processing has not started.

2.1.2 src441_setParam function

This procedure writes module static parameters from the main framework to the module’s internal memory. It can be called after the reset routine and before the start of the real time processing. It handles the static parameters, i.e. the parameters with values which cannot be changed during the module processing.

```c
int32_t src441_setParam(src441_static_param_t *input_static_param_ptr, void *static_mem_ptr);
```

Table 3. src441_setParam

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_static_param_ptr</td>
<td>src441_static_param_t*</td>
<td>Pointer to static parameters structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Returned value</td>
<td>int32_t</td>
<td>Error value</td>
<td></td>
</tr>
</tbody>
</table>
Note: There is currently no static parameter, so no reason to call this routine in this module version.

2.1.3 src441_getParam function

This procedure gets the module static parameters from the module internal memory to the main framework. It can be called after the reset routine and before the start of the real time processing. It handles the static parameters, i.e. the parameters with values which cannot be changed during the module processing.

```
int32_t src441_setParam(src441_static_param_t *input_static_param_ptr, void *static_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_static_param_ptr</td>
<td>src441_static_param_t*</td>
<td>Pointer to static parameters structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Returned value</td>
<td>-</td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>

2.1.4 src441_setConfig function

This procedure sets the module dynamic parameters from the main framework to the module internal memory. It can be called at any time during processing (after reset and setParam routines).

```
int32_t src441_setConfig(src441_dynamic_param_t *input_dynamic_param_ptr, void *static_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_dynamic_param_ptr</td>
<td>src441_dynamic_param_t*</td>
<td>Pointer to dynamic parameters structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Returned value</td>
<td>-</td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>

Note: There is currently no dynamic parameter, so no reason to call this routine in this module version.

2.1.5 src441_getConfig function

This procedure gets module dynamic parameters from the internal static memory to the main framework. It can be called at any time during processing (after the reset and setParam routines).

```
int32_t src441_getConfig(src441_dynamic_param_t *input_dynamic_param_ptr, void *static_mem_ptr);
```
2.1.6 src441_process function

This procedure is the module’s main processing routine. It should be called at any time, to process each frame.

```c
int32_t src441_process(buffer_t *input_buffer, buffer_t *output_buffer, void *static_mem_ptr);
```

2.2 External definitions and types

2.2.1 Input and output buffers

The SRC441 library is using extended I/O buffers which contain, in addition to the samples, some useful information on the stream such as the number of channels, the number of bytes per sample, and the interleaving mode.

An I/O buffer structure type, as described below, must be followed and filled in by the main framework before each call to the processing routine:

```c
typedef struct {
    int32_t nb_channels;
    int32_t nb_bytes_per_Sample;
    void *data_ptr;
    int32_t buffer_size;
    int32_t mode;
} buffer_t;
```
Table 8. Input and output buffers

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb_channels</td>
<td>int32_t</td>
<td>Number of channels in data: 1 for mono, 2 for stereo</td>
</tr>
<tr>
<td>nb_bytes_per_Sample</td>
<td>int32_t</td>
<td>16-bit = 2, 24-bit = 3, 32-bit = 4</td>
</tr>
<tr>
<td>data_ptr</td>
<td>void *</td>
<td>Pointer to data buffer (must be allocated by the main framework)</td>
</tr>
<tr>
<td>buffer_size</td>
<td>int32_t</td>
<td>Number of samples per channel in the data buffer</td>
</tr>
<tr>
<td>mode</td>
<td>int32_t</td>
<td>In case of stereo stream, left and right channels can be interleaved. 0 = not interleaved, 1 = interleaved.</td>
</tr>
</tbody>
</table>

2.2.2 Returned error values

Possible returned error values are described below:

Table 9. Returned error values

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC441_ERROR_NONE</td>
<td>0</td>
<td>OK - no error detected</td>
</tr>
<tr>
<td>SRC441_UNSUPPORTED_MODE</td>
<td>-1</td>
<td>If input data is not interleaved</td>
</tr>
<tr>
<td>SRC441_BAD_FRAME_SIZE</td>
<td>-2</td>
<td>If the number of input samples is not 147</td>
</tr>
<tr>
<td>SRC441_WRONG_NBBYTES_PER_SAMPLES</td>
<td>-3</td>
<td>Input data is not a 16-bit value</td>
</tr>
<tr>
<td>SRC441_UNSUPPORTED_NB_CHANNELS</td>
<td>-4</td>
<td>Input data is not stereo</td>
</tr>
<tr>
<td>SRC441_UNSUPPORTED_INPLACE_PROCESSING</td>
<td>-5</td>
<td>If input and output buffers are not different</td>
</tr>
<tr>
<td>SRC441_BAD_HW</td>
<td>-6</td>
<td>May happen if the library is not used with the right hardware.</td>
</tr>
</tbody>
</table>

2.3 Static parameters structure

There is no static parameter to be used. For commonality reasons, the static parameter structure contains a dummy field.

```c
struct src441_static_param {
    int32_t empty;
};

typedef struct src441_static_param src441_static_param_t;
```

Table 10. Static parameters structure

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>int32_t</td>
<td>Dummy field - just required to have a non-empty structure</td>
</tr>
</tbody>
</table>
2.4 Dynamic parameters structure

There is no dynamic parameter to be used. For commonality reasons, the dynamic parameter structure contains a dummy field.

```c
struct src441_dynamic_param {
    int32_t empty;
};
typedef struct src441_dynamic_param src441_dynamic_param_t;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>int32_t</td>
<td>Dummy field - just required to have a non-empty structure</td>
</tr>
</tbody>
</table>
3 Algorithm high level view

3.1 Processing steps

The SRC441 module is a re-sampler based on a two-stage polyphase filter. This implementation has been Flash-optimized for Cortex® M4 core using two polyphase filters; it has been MIPS-optimized using M4 SIMD instruction set.

Figure 1 shows an example of re-sampling from 44.1 kHz to 48 kHz with a scheduling of 3.33 ms.

3.2 Data formats

The module supports fixed point data, in Q15 or Q31 format, with a mono or stereo interleaved pattern.

The input buffer size is fixed at 147 samples and will generate 160 output samples per frame.
3.3 Performance measurements

3.3.1 SINAD measurements

Quality measurement is done on a 16-bit input signal with 16-bit I/O library versions, and on a 32-bit input signal (derived from a 24-bit input signal) with 32-bit I/O library versions.

THDN (Total Harmonic Distortion + Noise) corresponds to the reverse of SINAD (Signal to Noise And Distortion ratio) in case of a pure frequency tone input. The measurements below estimate that the SRC quality follows the AES 17-1998 (r2004) recommendations, by injecting a sine-wave, filtering the output with a standard notch filter (quality factor Q = 5) and the following computing ratio:

$$\text{SINAD} = 1/\text{THDN} = \frac{\text{Power of Input}}{\text{Power of notched filtered output}}$$

Table 12 summarizes the SINAD values in dB.

<table>
<thead>
<tr>
<th>Tests</th>
<th>Amp (dBFS)</th>
<th>-0.1</th>
<th>-1</th>
<th>-1</th>
<th>-1</th>
<th>-1</th>
<th>-1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freq (Hz)</td>
<td></td>
<td>997</td>
<td>40</td>
<td>160</td>
<td>640</td>
<td>1280</td>
<td>2560</td>
<td>5120</td>
</tr>
<tr>
<td>Standard, 44.1 to 48 kHz</td>
<td></td>
<td>95.2</td>
<td>94.2</td>
<td>94.3</td>
<td>94.1</td>
<td>94.0</td>
<td>94.1</td>
<td>94.5</td>
</tr>
<tr>
<td>High quality</td>
<td></td>
<td>95.4</td>
<td>94.4</td>
<td>94.2</td>
<td>94.0</td>
<td>94.1</td>
<td>94.5</td>
<td>94.8</td>
</tr>
<tr>
<td>Standard, 32 bits</td>
<td></td>
<td>107.1</td>
<td>107.2</td>
<td>107.2</td>
<td>107.1</td>
<td>107.2</td>
<td>108</td>
<td>108.7</td>
</tr>
<tr>
<td>High quality, 32 bits</td>
<td></td>
<td>131</td>
<td>132.4</td>
<td>132.4</td>
<td>131.5</td>
<td>131.1</td>
<td>132.7</td>
<td>132.3</td>
</tr>
</tbody>
</table>

Note: No windowing is applied in the measurements above, and A-Law usage shows a gain of about 2-3 dBs between 600 Hz and 8 kHz, and a loss of several dBs outside this range due to the A-Law shape that will bring the signal to analyze closer to the noise floor.
### 3.3.2 Frequency response measurements

The frequency response analysis gives information on in-band ripple, frequency cut at -1 and -3 dB and filter group delay (the filters used have a linear phase). Table 13 summarizes the data with a standard or a high quality version.

**Table 13. Frequency response values**

<table>
<thead>
<tr>
<th>Tests</th>
<th>Freq (Hz)</th>
<th>Max. ripple (dB)</th>
<th>Min. ripple (dB)</th>
<th>Frequency cut at -1 dB</th>
<th>Frequency cut at -3 dB</th>
<th>Filter group delay (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>44.1 to 48 kHz</td>
<td>0.1</td>
<td>-0.07</td>
<td>16200</td>
<td>17300</td>
<td>0.3</td>
</tr>
<tr>
<td>High quality</td>
<td>44.1 to 48 kHz</td>
<td>0.13</td>
<td>-0.05</td>
<td>17400</td>
<td>18400</td>
<td>0.43</td>
</tr>
<tr>
<td>Standard, 32 bits</td>
<td></td>
<td>0.1</td>
<td>-0.07</td>
<td>16200</td>
<td>17300</td>
<td>0.3</td>
</tr>
<tr>
<td>High quality, 32 bits</td>
<td></td>
<td>0.13</td>
<td>-0.05</td>
<td>17400</td>
<td>18400</td>
<td>0.43</td>
</tr>
</tbody>
</table>
4 System requirements and hardware setup

The SRC441 library is built to run on a Cortex M4 core without FPU usage. It can be integrated and run on STM32F4 family devices. There is no other hardware dependency.

4.1 Recommendations for optimal setup

The sampling rate conversion algorithm should be placed quite early in the audio chain, for instance just after the audio decoder in order to get all the audio streams at the same sampling frequency. If needed, streams can be mixed now, or a post-processing can be applied. Samples are then played on the audio DAC.

Follow the steps below to integrate the SRC441 in an audio framework.

4.1.1 Memory allocation

First of all, all the memory used by the module must be allocated.

There is neither static nor dynamic parameter in the SRC441 module, thus no memory to allocate to get and set static or dynamic parameters.

Next, static and dynamic memory required by the SRC441 module must be allocated by the framework. Structures are hidden to the audio framework, but their sizes are exported as a constant in the src441_glo.h file. The memory allocation can be done as written below:

```c
/* SRC441 memory structure memory allocation */
void *static_mem_ptr = malloc(src441_static_mem_size);
void *dynamic_mem_ptr = malloc(src441_dynamic_mem_size);
```

dynamic_mem_ptr pointer is a parameter of src441_reset() routine, while static_mem_ptr pointer is a parameter of all SRC441 exported APIs.

It is then necessary to allocate the memory for input and output audio buffers.

4.1.2 Module API calls

Once the memory as been allocated, the src441_reset() routine must be called to initialize the SRC441 module static memory.

- Since there is no static parameter in the current version, src441_setParam() and src441_getParam() are not used.
- Since there is no dynamic parameter in the current version, src441_setConfig() and src441_getConfig() are not used.

Now that the hardware is configured and the SRC441 module initialized and configured, the run time process can start.

At each new frame, the input buffer structure fields must be filled in as in the example below, as well as the data address for the output buffer structure:

```c
/* SRC441 input buffer configuration */
input_buffer_t.data_ptr = input_buffer_ptr;
input_buffer_t.buffer_size = 147;
input_buffer_t.mode = INTERLEAVED;
input_buffer_t.nb_bytes_per_Sample = 2;
```
input_buffer_t.nb_channels = 2;
output_buffer_t.data_ptr = output_buffer_ptr;

And the src441_process() routine can be called if a sampling rate conversion is needed:
/* SRC441 processing call */
src441_error = SRC441_ERROR_NONE;
If (input_sampling_frequency != 48000) {
src441_error = src441_process(&input_buffer_t, &output_buffer_t,
static_mem_ptr);
}
Else {
// Recopy input buffer to output buffer...
}
If (src441_error != SRC441_ERROR_NONE) {
// Error management...
}

4.1.3 Module integration summary

Figure 2. API call procedure
1. As explained above, SRC441 static and dynamic memories have to be allocated, as well as the input and output buffer.
2. Once the memory has been allocated, the call to src441_reset() function will initialize the internal variables.
3. The audio stream is read from the proper interface, and the input_buffer structure has to be filled in according to the stream characteristics (number of channels, sample rate, interleaving and data pointer). The output buffer structure has to be set as well.
4. A call to the process will re-sample the stream in the output buffer.
5. The output audio stream can now be written in the proper interface.
6. Once the processing loop is over, the allocated memory has to be freed.
5 How to tune and run the application

There is no tuning available for the SRC441 module via the audio framework.

The only available choice is to link lib_src441_m4.a library, lib_src441_HQ_m4.a, lib_src441_32b_m4.a or lib_src441_HQ_32b_m4.a library version, all of them being associated to the src441_glo.h header file.

Once the module has been integrated into the audio framework which will play samples at 48 kHz, launch the Audio player and choose a .WAV or .MP3 file with an output sampling frequency of 44.1 or 48 kHz. The file will be decoded, re-sampled and played at 48 kHz without returning any error message.
6 Revision history

Table 14. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-Jun-2013</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>26-Aug-2013</td>
<td>2</td>
<td>Changed all “stereo” occurrences into “mono and stereo”. Added “mono” values to Table 1: Resource summary.</td>
</tr>
<tr>
<td>28-Nov-2014</td>
<td>3</td>
<td>Classification changed from ST Restricted to public. Replaced the reference STSW-STM32APP by STM32-AUDIO100A.</td>
</tr>
</tbody>
</table>