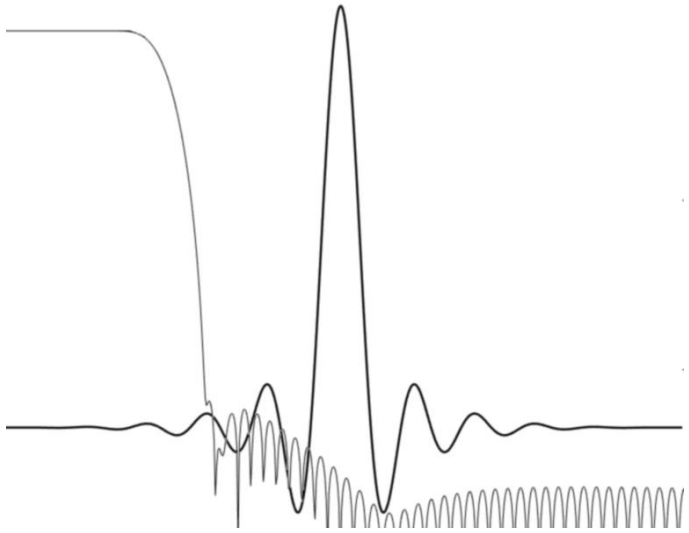
LOW-COMPLEXITY ARBITRARY SAMPLE-RATE CONVERTER



Speech and audio signal processing
for embedded markets

Overview

This program implements an adjustable interpolator used for the sampling rate conversion (SRC).

The code is ported to Cortex-M4 / M7 and Cortex-A with or without NEON.

Several configurations are precomputed and can be prepared depending on the memory size constraints of your device.

The bit-exact demonstration files and executable (BATCH_SRC.BAT) are located at :
http://firmware-developments.com/WEB/P6x/SSRC_M4/DEMO/

Programming interfaces

The program works from a pre-computed coefficient-set corresponding to a low-pass filter. A Matlab/Octave program generates the coefficients from different parameters: computation load, sharpness of the filter, resampling accuracy, memory size constraints.

The public APIs of the programs are:

1. Returning the amount of required memory from input parameters : input and output sampling rates, samples format.
2. Create an instance of the sample-rate converter and initializing it
3. Process one instance, taking an input mono audio buffer and returning the new samples in an output buffer with the number of interpolated output samples.

Details

The focus of this program is run as fast as possible. To do so, the computation is done with one single filtering step.

Sampling-rate accuracy. The SRC is computing the output samples from a polyphase FIR (finite impulse response) filter. The number of phase in the filter depends on the least common multiples between the input and output frequencies. For example, going from 16kHz to 48kHz means 3 phases because 16kHz x 3 = 48kHz. But going from 11.025kHz to 32kHz means 1280 phases because 11.025kHz x (1280/441) = 32kHz.

If your sampling frequencies are not corresponding to the number of phases of the filter, the program will arrange to find the closest approximation. For example, a filter shape of 12 phases used to go from 44.1kHz to 48kHz cannot use the ideal ratio (160/147) but will use (12/11) instead, resulting in 0.2% sampling error. (44.1kHz x (12/11) = 48.1kHz).

Complexity. The minimum complexity in the polyphase filtering is the minimum number of taps in the FIR to process one sample. Usually this number is 24, but can be set at different values (from 8 to 32 for example) depending on the computation capabilities of the processor and the shape of the filter.

Memory. The flash memory consumption mainly comes from the filter coefficients. The size is (4 bytes) x (number of phases) x (minimum number of taps). For example with 12 phases and 24 taps the size of table of coefficients is 1152 bytes.

The RAM memory is (minimum number of taps x (1 + (high sampling rate / low sampling rate)).

The number of samples in the output buffer is equal to (input buffer size) x (output sampling rate / input sampling rate). When this number is not an integer the number of output samples vary from one process call to the other.

Algorithm complexity numbers

The shape of the pre-computed low-pass filter is used to create a polyphase FIR. The number of taps (NFIR in the table below) is arranged to be a multiple of 4 to be compatible with NEON vector operations. In the example below the minimum FIR length is set to 24 and the number of taps is proportional to the interpolation or decimation ratio. The table gives the number of millions of multiply-accumulate (MAC) operations per second. For example the interpolation from 8kHz to 16kHz takes 0.384 Million MAC/s.

The critical loop of the program consists in a dot-product operation, the speed of which depends on the micro-architecture of the processor.

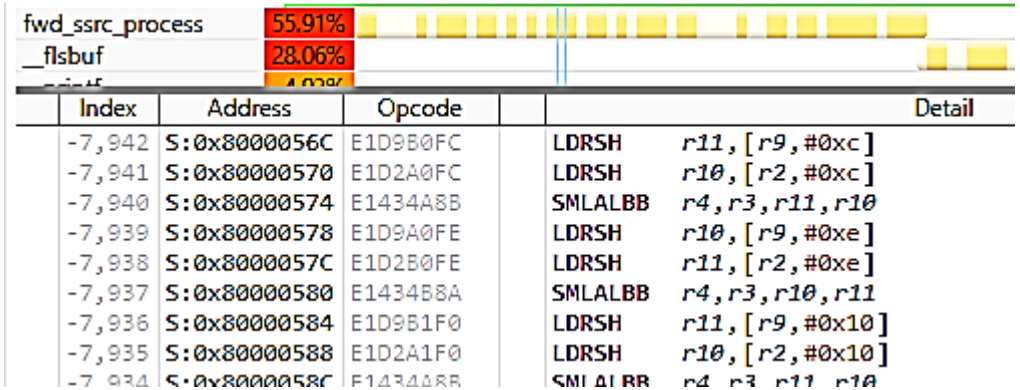
```
 8000 => 16000 NFIR= 24 L= 1280 / M=  640   MMAC/s= 0.3840
 8000 => 22050 NFIR= 24 L= 1323 / M=  480   MMAC/s= 0.5292
 8000 => 24000 NFIR= 24 L= 1281 / M=  427   MMAC/s= 0.5760
 8000 => 32000 NFIR= 24 L= 1280 / M=  320   MMAC/s= 0.7680
 8000 => 44100 NFIR= 24 L= 1323 / M=  240   MMAC/s= 1.0584
 8000 => 48000 NFIR= 28 L= 1278 / M=  213   MMAC/s= 1.3440
16000 =>  8000 NFIR= 48 L=  640 / M= 1280   MMAC/s= 0.3840
16000 => 22050 NFIR= 24 L= 1323 / M=  960   MMAC/s= 0.5292
16000 => 24000 NFIR= 24 L= 1281 / M=  854   MMAC/s= 0.5760
16000 => 32000 NFIR= 24 L= 1280 / M=  640   MMAC/s= 0.7680
16000 => 44100 NFIR= 24 L= 1323 / M=  480   MMAC/s= 1.0584
16000 => 48000 NFIR= 24 L= 1281 / M=  427   MMAC/s= 1.1520
22050 =>  8000 NFIR= 64 L=  480 / M= 1323   MMAC/s= 0.5120
22050 => 16000 NFIR= 32 L=  960 / M= 1323   MMAC/s= 0.5120
22050 => 24000 NFIR= 24 L= 1280 / M= 1176   MMAC/s= 0.5760
22050 => 32000 NFIR= 24 L= 1280 / M=  882   MMAC/s= 0.7680
22050 => 44100 NFIR= 24 L= 1280 / M=  640   MMAC/s= 1.0584
22050 => 48000 NFIR= 24 L= 1280 / M=  588   MMAC/s= 1.1520
```

```
32000 =>  8000 NFIR= 96 L=  320 / M= 1280   MMAC/s= 0.7680
32000 => 16000 NFIR= 48 L=  640 / M= 1280   MMAC/s= 0.7680
32000 => 22050 NFIR= 36 L=  882 / M= 1280   MMAC/s= 0.7938
32000 => 24000 NFIR= 32 L=  960 / M= 1280   MMAC/s= 0.7680
32000 => 44100 NFIR= 24 L= 1323 / M=  960   MMAC/s= 1.0584
32000 => 48000 NFIR= 24 L= 1281 / M=  854   MMAC/s= 1.1520
44100 =>  8000 NFIR=128 L=  240 / M= 1323   MMAC/s= 1.0240
44100 => 16000 NFIR= 64 L=  480 / M= 1323   MMAC/s= 1.0240
44100 => 22050 NFIR= 48 L=  640 / M= 1280   MMAC/s= 1.0584
44100 => 24000 NFIR= 44 L=  720 / M= 1323   MMAC/s= 1.0560
44100 => 32000 NFIR= 32 L=  960 / M= 1323   MMAC/s= 1.0240
44100 => 48000 NFIR= 24 L= 1280 / M= 1176   MMAC/s= 1.1520
48000 =>  8000 NFIR=148 L=  213 / M= 1278   MMAC/s= 1.1840
48000 => 16000 NFIR= 72 L=  427 / M= 1281   MMAC/s= 1.1520
48000 => 22050 NFIR= 56 L=  588 / M= 1280   MMAC/s= 1.2348
48000 => 24000 NFIR= 48 L=  640 / M= 1280   MMAC/s= 1.1520
48000 => 32000 NFIR= 36 L=  854 / M= 1281   MMAC/s= 1.1520
48000 => 44100 NFIR= 28 L= 1176 / M= 1280   MMAC/s= 1.2348
```

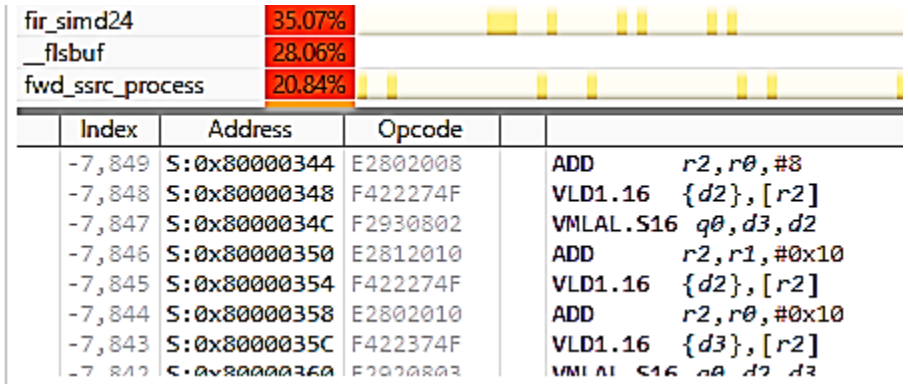
CPU load simulation

The speed simulation was made on DS-5 (v5.27) on the Cortex-A7-FVP model emulating a system clocked at 133MHz with caches enabled. The samples are processed by packets of 16 samples for a conversion from 16kHz to 44.1kHz. The number of generated output samples is : (160k input samples) x (44.1/16 ratio) = 441k samples. The compiler used is armcc with command line: armcc --cpu=Cortex-A7 -O3 --vectorize –g --md –c

Results : 330 system ticks (1ms) are used for the scalar version (99cycles/sample) and 210 ticks for the SIMD version (63cycles/sample)
Numbers are twice lower using the aarch64 model for Cortex-A57, with about the same speed improvement ratio for SIMD.



Scalar filtering



Filtering with SIMD instructions

THDN performances – floating-point 32bits

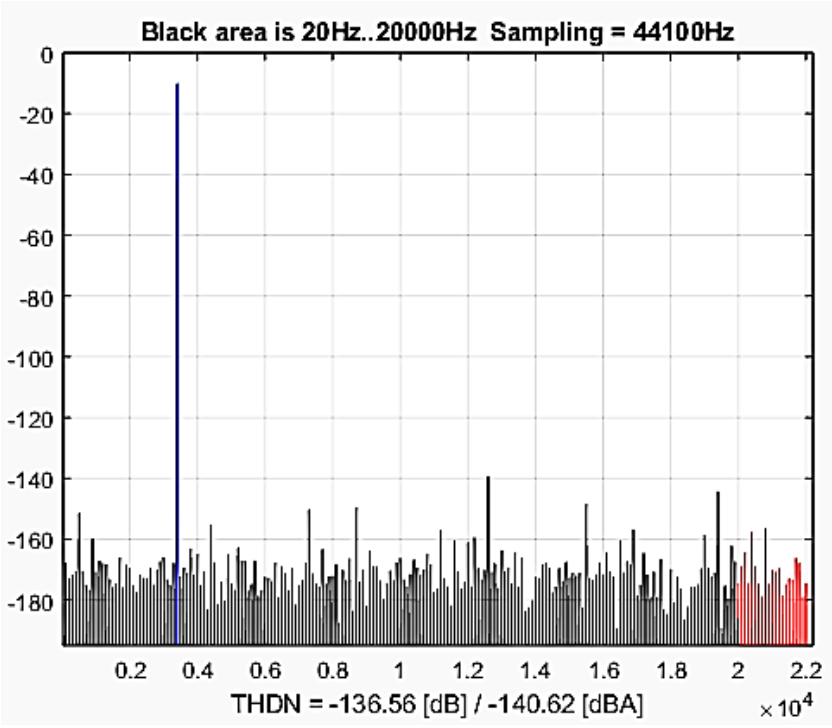
A sine wave (frequency 251Hz and 3400Hz) is resampled.
The computation of the A-weighted THD+N gives numbers larger than 130dB.
The minimum filter length was set to 28 taps.

Sine wave -10dB at 251Hz (THD+N results in dBA)

Fs out=>	8000Hz	16000Hz	22050Hz	24000Hz	32000Hz	44100Hz	48000Hz
8000Hz		150	133	134	134	134	137
16000Hz	147		138	134	151	143	139
22050Hz	143	141		135	138	152	142
24000Hz	145	148	139		136	143	153
32000Hz	144	147	143	142		136	142
44100Hz	143	143	148	143	144		135
48000Hz	143	146	143	148	150	138	

Sine wave -10dB at 3400Hz (THD+N results in dBA)

Fs out=>	8000Hz	16000Hz	22050Hz	24000Hz	32000Hz	44100Hz	48000Hz
8000Hz							
16000Hz			126	138	138	128	137
22050Hz		140		127	132	135	135
24000Hz		141	132		132	144	139
32000Hz		149	143	143		140	137
44100Hz		145	148	145	143		133
48000Hz		147	143	148	145	138	



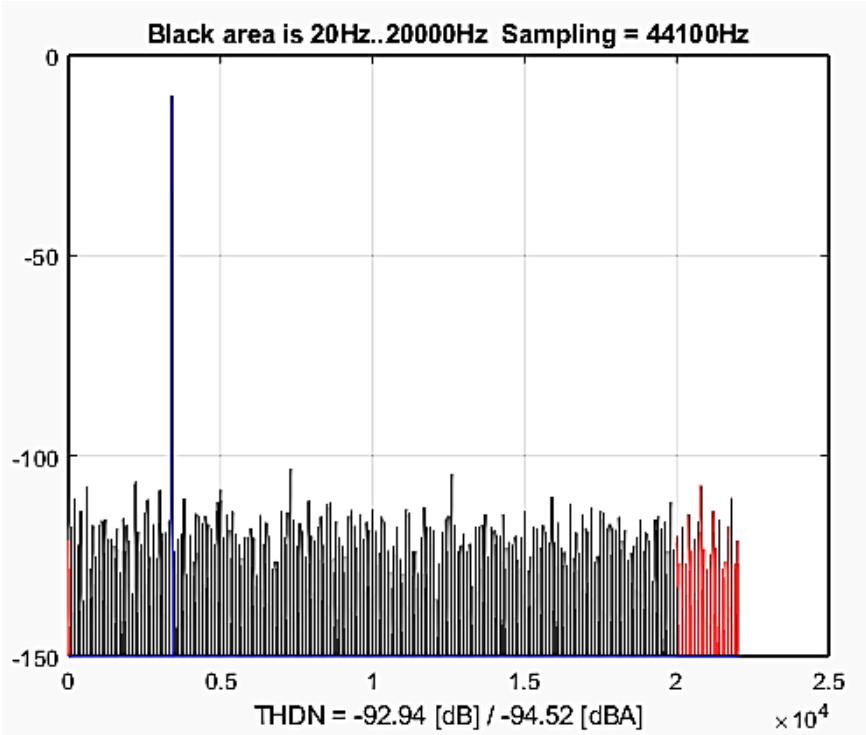
THDN performances – fixed-point 16bits

A sine wave (frequency 251Hz and 3400Hz) at -10dB FS is resampled at several frequencies.
The computation of the A-weighted THD+N gives numbers in the 95dBA range.
The minimum filter length was set to 24 taps.

Fs out=>	8000Hz	16000Hz	22050Hz	24000Hz	32000Hz	44100Hz	48000Hz
8000Hz		95	93	95	94	94	95
16000Hz	97		93	95	96	95	96
22050Hz	91	93		94	94	97	95
24000Hz	97	96	93		94	95	97
32000Hz	97	96	92	94		95	97
44100Hz	90	92	97	93	94		95
48000Hz	98	97	92	97	97	95	

Sine wave -10dB at 3400Hz (THD+N results in dBA)

Fs out=>	8000Hz	16000Hz	22050Hz	24000Hz	32000Hz	44100Hz	48000Hz
8000Hz							
16000Hz			94	96	95	95	97
22050Hz		93		93	94	95	95
24000Hz		93	93		94	94	96
32000Hz		99	93	94		96	97
44100Hz		91	97	93	94		95
48000Hz		97	91	97	93	95	



Spectral flatness

The computation do not introduce ripples in the output samples.
The two plots on the left are the spectrum shape of the of the 24-taps filter.
The right-plot is the wave and spectrogram of the fixed-point Q15 processing result.
In this example, the spectral flatness at 1dB is guaranteed up to 19500Hz.