Atmel's DSPLib re-sampling algorithm (version 2)

Following is a brief description of the frequency re-sampling algorithm used in the DSP library. It is aimed for anybody so no specifics digital signal processing knowledge is required to understand this presentation.

Algorithm

The principle is quite simple, it consists in 2 main stages, increasing the sampling rate by an integer value (**L**), this action is also called **interpolation**, and then reducing it by another integer value (**M**), also known as **decimation**.
This process is also known as **band limited interpolation**.

L and M calculation

L and M are 2 integers that are calculated by getting the **GCD** (Greatest Common Divisor) of the input (**FSin**) and the output (**FSout**) sampling frequencies. The number resulting will divide Fsin and Fsout to respectively give M and L.

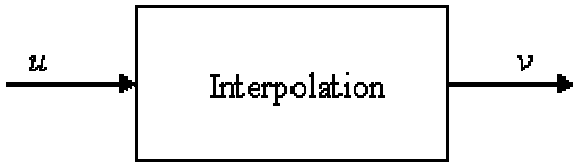
$$M = \frac{Fs_{in}}{gcd(Fs_{in}, Fs_{out})}, L = \frac{Fs_{out}}{gcd(Fs_{in}, Fs_{out})}$$

Then the following formula can be applied:

FSout = FSin * L / M

Interpolation

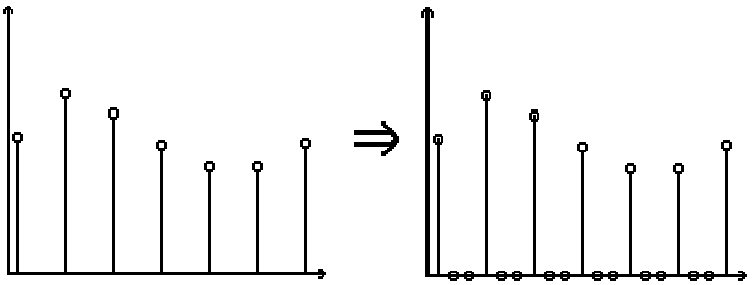
This process increases the frequency sampling rate of the input signal by an integer factor. The factor used at this stage of the process by the re-sampling algorithm will be the pre-calculated "interpolation factor" **L**. Which means, if we consider this process as a black box with 1 input (**u**) and 1 output (**v**), the output signal sampling frequency (Fs(v)) will be equals to the input signal sampling frequency (Fs(u)) multiplied by L.



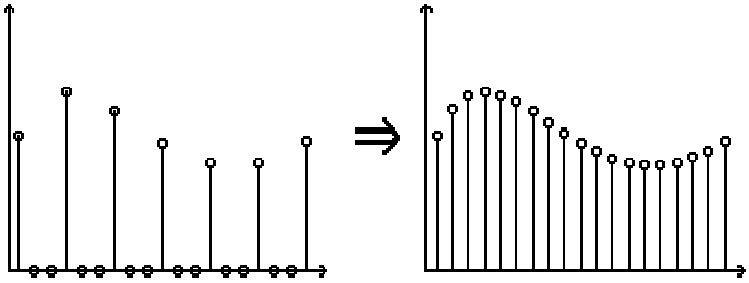
$$Fs(v) = Fs(u).L$$

The following describes the algorithm used to implement the interpolation. The method consists in two parts:

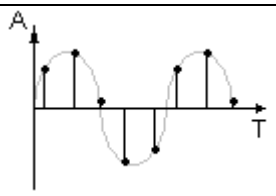
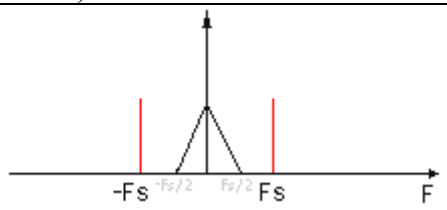
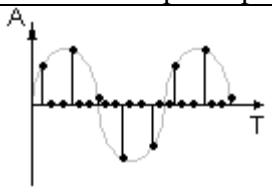
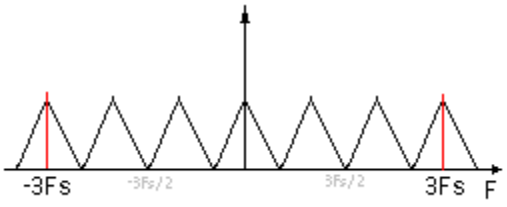
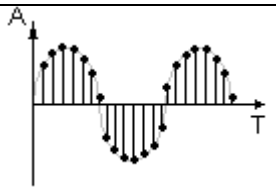
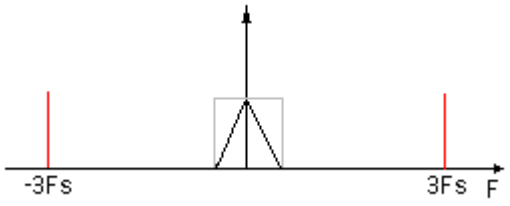
1. Extending the signal by filling "blank spaces" with zeros, in order to obtain a signal with the desired sampling rate. This is called **up-sampling**.



2. **Low-pass filtering** the signal to “remove zeros” introduced during the previous stage.



Here is a view of the process (time domain Vs frequency domain):

Time domain	Frequency domain
Input signal (Fs = FSin)	
	
Up-sampling with L = 3 – Fs becomes L * Fs	
	
Low-pass filtering (Fc = Fs/2)	
	

Filtering

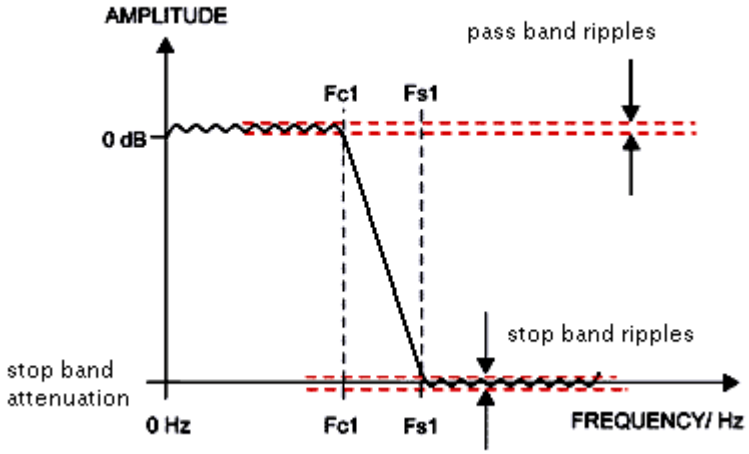
According to the diagram above, the filter must be a low-pass filter with a cut-off frequency equal to **min(FSin, FSout) / 2**. The frequency sample of this filter is **Fs * L**.

How to choose the filter?

The frequency response of the system is defined by the filter itself. Therefore this part is very important to ensure the best characteristics for your system.

There are many different ways to generate FIR filter coefficients, and therefore there characteristics will be different from one method to the other. But it is always a trade off between the filter resources and its quality. The larger the filter is (defined by its order), the more memory and processor resources it will consume but the best quality it will have.

Following is the frequency response of a low-pass FIR filter:



Note that the cut-off frequency should be between Fc1 and Fs1. By choosing Fs1 equals to the cut-off frequency, you will ensure the lowest transition band aliasing but the pass band of the filter will be reduced.

In order to estimate the right filter order to be used according to what the system requires, some formula exists, please refer to one of the following:
- "**ωp**" and "**ωc**" are respectively the Fc1 and Fs1 in radian. In other word, it is equal to 2 * π * F / Fs.
- "**δp**" is the pass band ripples amplitude.
- "**δs**" is the stop band attenuation.

3. Kaiser's Formula

$$N \cong \frac{-20\log_{10}(\sqrt{\delta_p \delta_s})}{14.6(\omega_s - \omega_p)/2\pi}$$

4. Hermann-Rabiner Chan's Formula

$$N \cong \frac{D_\infty(\delta_p,\delta_s) - F(\delta_p,\delta_s)[(\omega_s-\omega_p)/2\pi]^2}{(\omega_s-\omega_p)/2\pi}$$

where:

$$D_\infty(\delta_p,\delta_s) = [a_1(\log_{10}\delta_p)^2 + a_2(\log_{10}\delta_p) + a_3]\log_{10}\delta_s + [a_4(\log_{10}\delta_p)^2 + a_5(\log_{10}\delta_p) + a_6]$$

and

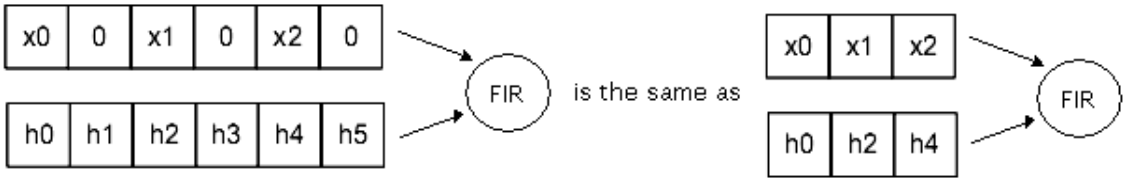
$$F(\delta_p,\delta_s) = b_1 + b_2[\log_{10}\delta_p - \log_{10}\delta_s]$$

5. Fred Harris' Formula

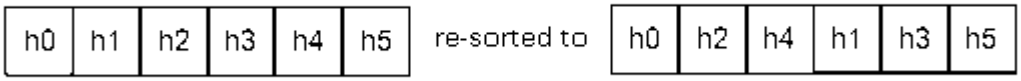
$$N \cong \frac{\delta_s(dB)}{22(\omega_s - \omega_p)/2\pi}$$

Polyphase filter
In this algorithm to improve efficiency, the filter is used as a **polyphase** FIR filter. Polyphase filters are basically normal filters processed in different pieces. The goal here

is to have one piece of the data with the non-zero values of the signal and the rest with the zeros. The result will be then the same as processing only the non-zeros samples from the input signal:



Filter coefficients arrangement
In order to get the most efficiency out of this method, the algorithm also re-sorts the filter coefficients in the following way:



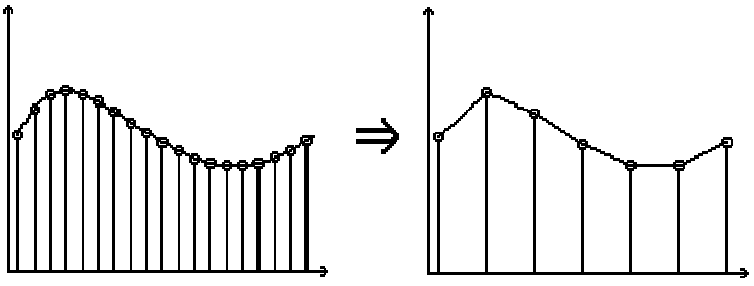
Filter coefficient normalization
If enabled, the algorithm also normalizes the coefficients to ensure that the output result will never go above 1 in amplitude. This ensures that the value will never overflowed which is very useful when fixed-point format is used.

Decimation (frequency down-sampling)

This process is much simpler than the interpolation.
It consists in removing samples in order to keep the same signal wave form but with a lower sampling rate.
Therefore, to obtain the desired output sampling frequency, the signal has to be down sampled by **M** (decimation factor).

$$Fs(v) = Fs(u) / M$$

Every M samples are kept from the input signal and all the others are simply removed.



Conclusion

By processing these 2 main stages, the signal is re-sampled by a factor equals to **L**/**M**. Therefore, the smaller the 2 frequencies have their GCD (Greatest Common Divisor), the more memory it will need (to store the FIR filter coefficients).
This method is one of the most used in digital signal processing systems. It will generate a clean signal and evaluate at best the waveform of the output signal.

About the API

Filter coefficients generation methods

The API allows the user to use two different methods to feed the internal FIR filter with its coefficients:

1. The **dynamic** method – which uses an internal algorithm to design on-the-fly FIR coefficients. It is the easiest way to proceed, since the algorithm will choose by itself all the FIR characteristics. (Note that the user can also define some custom characteristics if needed, see the automated Doxygen documentation of the code for more information). The algorithm will then need higher requirement in RAM memory than the other method since the coefficients will be stored in RAM.
2. The **fixed** method – it uses pre-calculated filter coefficients. The user can then design its filter according to its needs. This solution gives the most flexibility. The filter coefficients should be stored in FLASH memory.

By using the dynamic method, the user can choose by adding to the filter a window. The window is also automatically generated by the algorithm and any windows from the library can be used.

"Order" Vs "Filter order"

Note, by using this API, the user must define an order for the re-sampling module. This order is NOT the order of the filter order to be used but defines it indirectly. The actual filter order used is a multiplication from this order with the interpolation factor:

<filter order> = <order> * <interpolation factor>

Normalization

Normalization of the filter coefficients can only be done in dynamic mode. By doing so, the algorithm will make sure the amplitude of a re-sampled signal can never go above 1. The resulting signal is in practical case around half the size of the input signal, but this depends on the filter used.

Frequency response

The frequency response of the re-sampling algorithm is directly related to the filter used. To know what should the response look like; an easy way is to look at the frequency response of the filter. Note that the order of the filter is the order of the re-sampling algorithm multiplied by the interpolation factor. Therefore, at equal re-sampling order, re-sampling a stream from 32 KHz to 64 KHz will have a worse frequency response than re-sampling a stream from 32 KHz to 44.1 KHz (because the filter order will be much lower).

Resources

Memory

Following is the memory footprint of the algorithm:

- Algorithm **context** (includes all the internal data used by the algorithm):
  (sizeof(dsp_resampling_t) + <input buffer size in bytes>) * <number of channels>
  =

  ~ (50 + <input buffer size in bytes>) * <number of channels> bytes
- A buffer to store **filter coefficients**:

  2 * <order> * <interpolation factor> bytes
- Code size:

  ~ 5 Kbytes

If the fixed method is used:

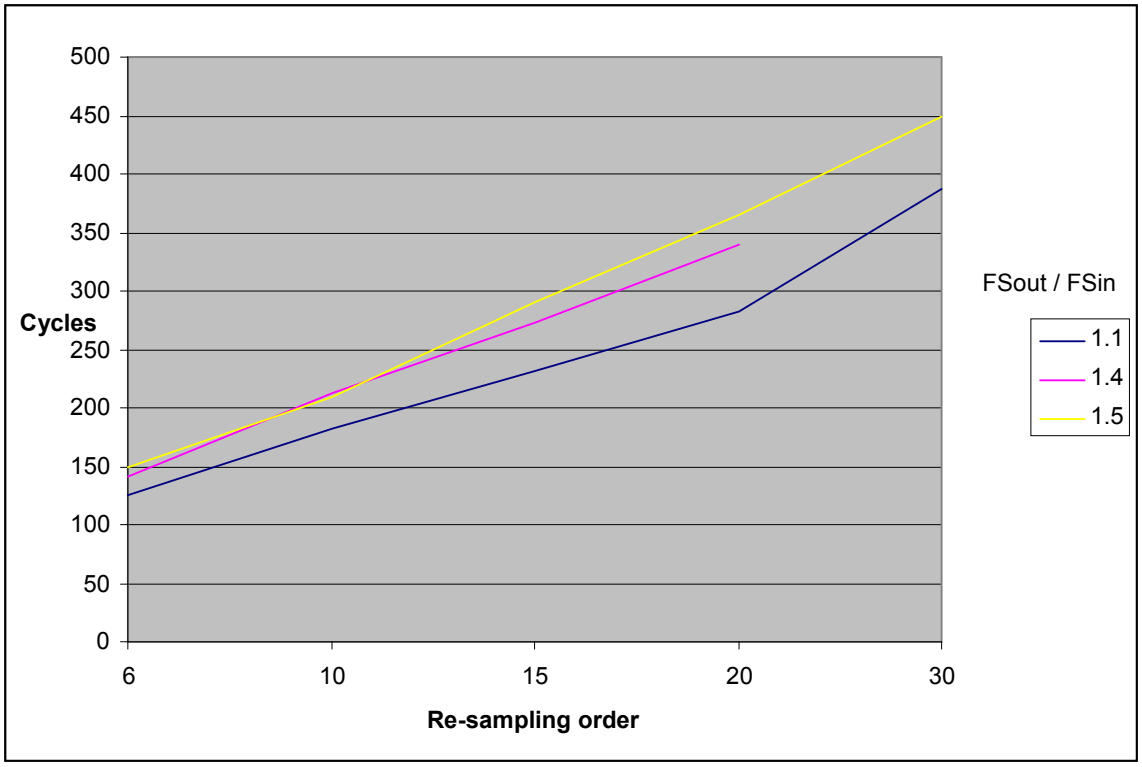
RAM	ROM
50 * <# of channels> + <input buffer size in bytes> * <# of channels> bytes	5 Kbytes + 2 * <order> * <interpolation factor> bytes

If the dynamic method is used:

RAM	ROM
50 * <# of channels> + <input buffer size in bytes> * <# of channels> + 2 * <order> * <interpolation factor> bytes	5 Kbytes

Cycles

The number of cycles is related to the re-sampling order and the ratio between the input sampling frequency and the output sampling frequency.
The following graph shows the number of cycles to process one sample amongst different frequency re-sampling ratios:

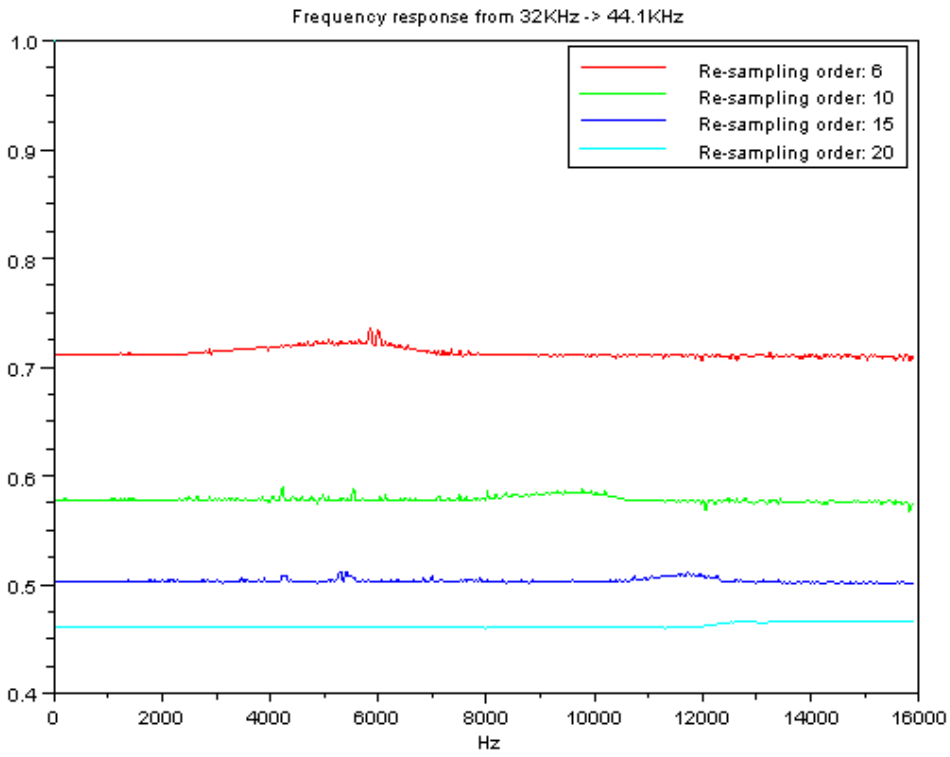


Examples

Re-sampling algorithm used with a **256-byte input buffer** for a stereo audio input stream (**2 channels**). (**Fixed** method is used with the pre-calculated filter coefficient generated with Scilab: windowed FIR filter + Hann window).

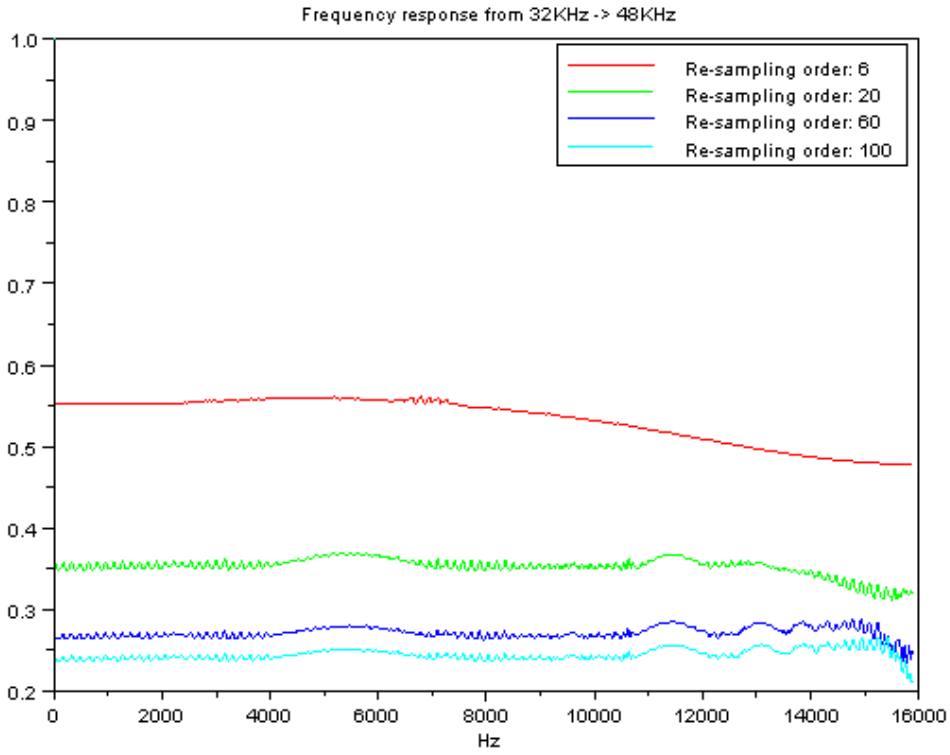
Frequency re-sampling from 32 KHz to 44.1 KHz

Order	**6**	**10**	**15**	**20**
Interpolation factor	441	441	441	441
Filter order	2646	4410	6615	8820
RAM (bytes) [1]	612	612	612	612
ROM (bytes) [1]	10K	14K	18K	23K
# cycles / samples [2]	141	213	279	340



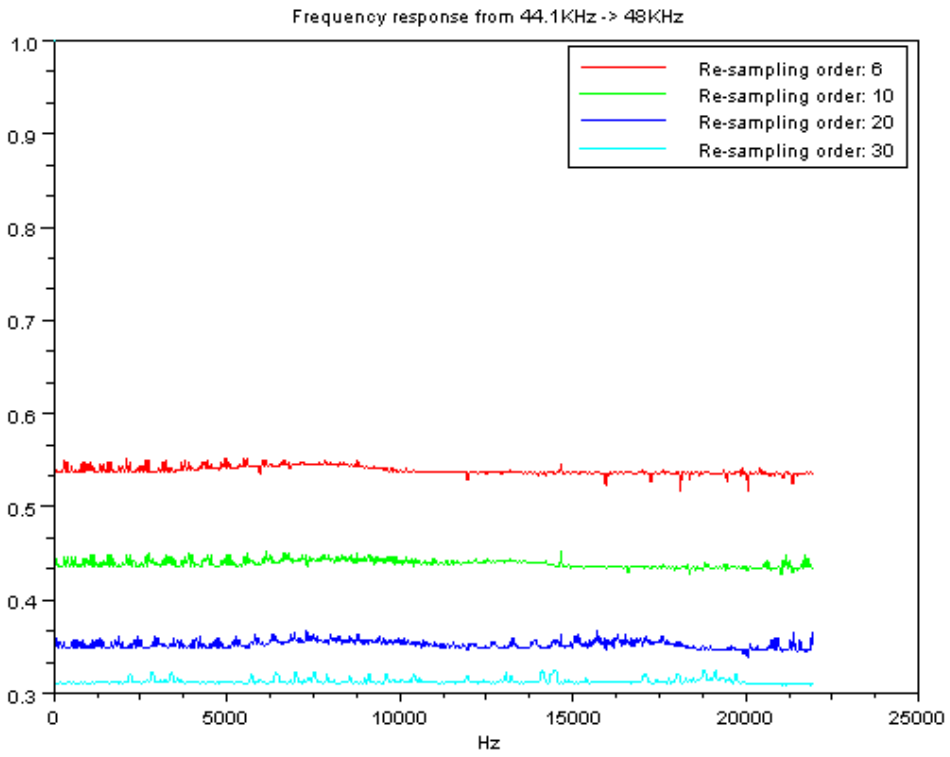
Frequency re-sampling from 32 KHz to 48 KHz

Order	**6**	**20**	**60**	**100**
Interpolation factor	3	3	3	3
Filter order	18	60	180	300
RAM (bytes) [1]	612	612	612	612
ROM (bytes) [1]	5K	5K	6K	6K
# cycles / samples [2]	149	365	928	1476



Frequency re-sampling from 44.1 KHz to 48 KHz

Order	**6**	**10**	**20**	**30**
Interpolation factor	160	160	160	160
Filter order	960	1600	3200	4800
RAM (bytes) [1]	612	612	612	612
ROM (bytes) [1]	7K	8K	12K	15K
# cycles / samples [2]	125	182	283	388



(1) The RAM and ROM memory footprints include the code size of the algorithm and its context as well as the filter coefficients size. Therefore, if the user chooses to re-sample different frequency rates, only the size of the filter coefficients will be added to the count of the memory footprint. See the *Memory* chapter on page 7 for more details.

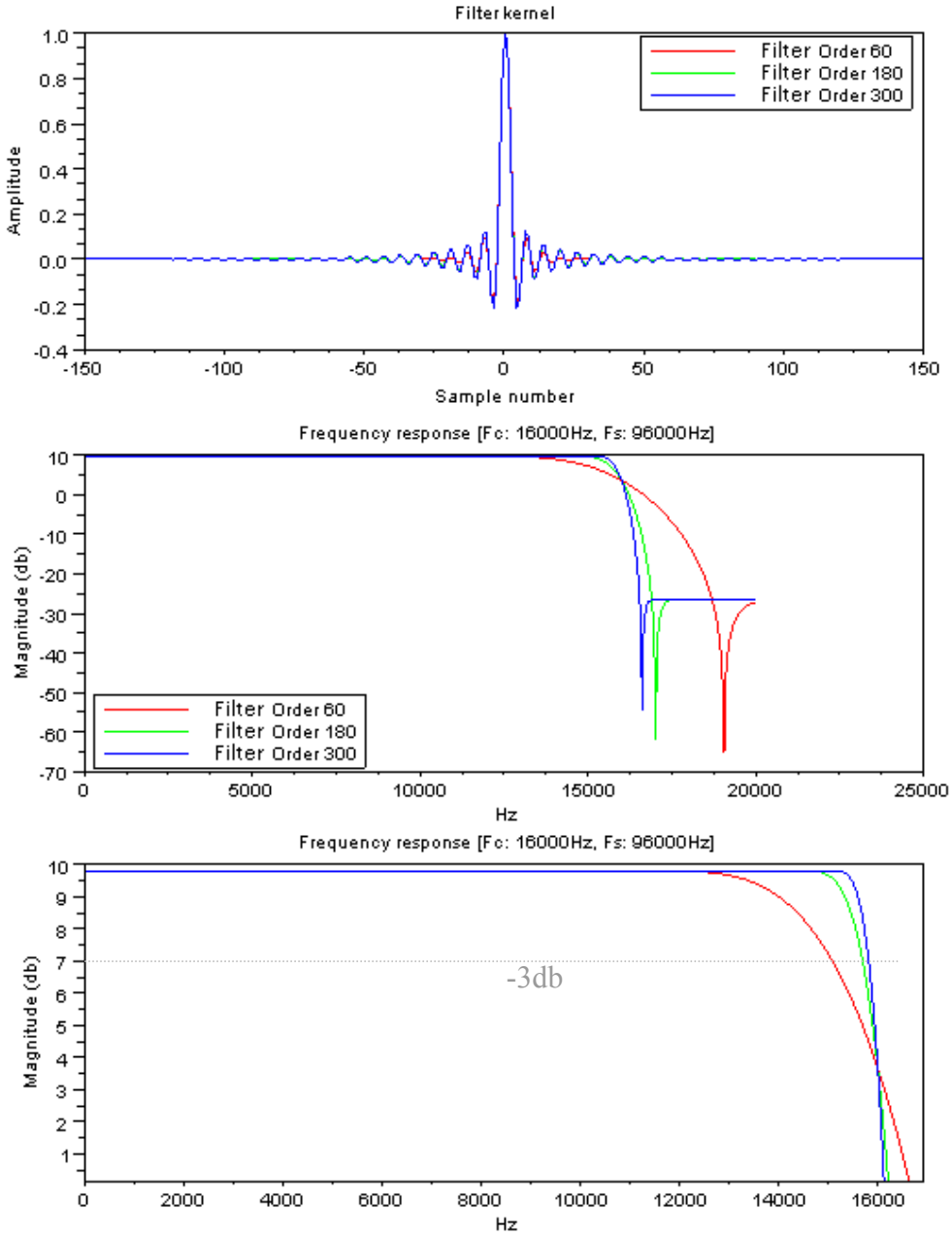
(2) The number of cycles is for 1 mono (1 channel) sample. This measurement has been done with IAR for 32-bit AVR v3.20 using high speed optimization.

Characterization of the algorithm from simulation

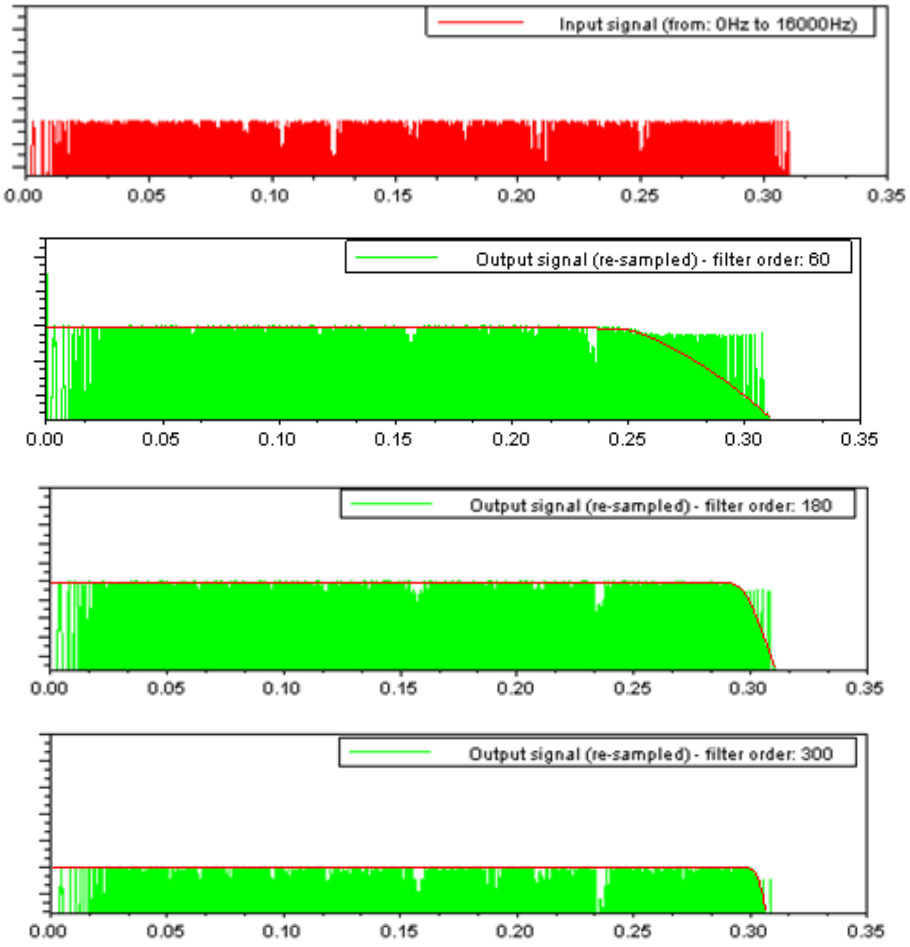
Frequency re-sampling from 32 KHz to 48 KHz (dynamic method)

Using the DSPLib to auto-generate the filter coefficients using the windowed sinc method:

- Filter characteristics:



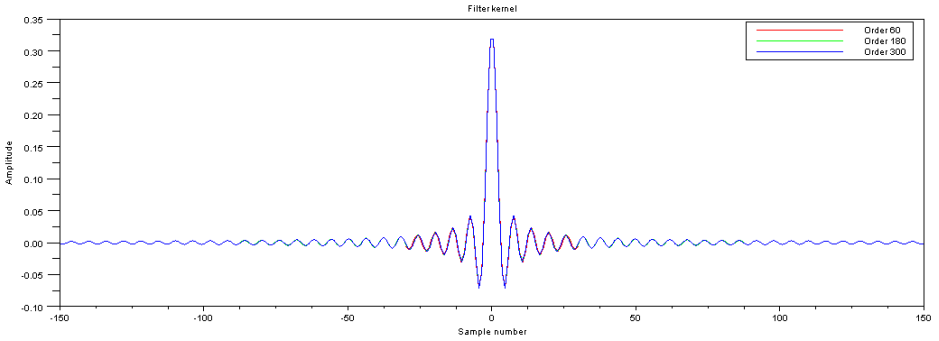
- Re-sampling algorithm - time domain (input signal is a linear frequency sweep from 0 to 16 KHz):

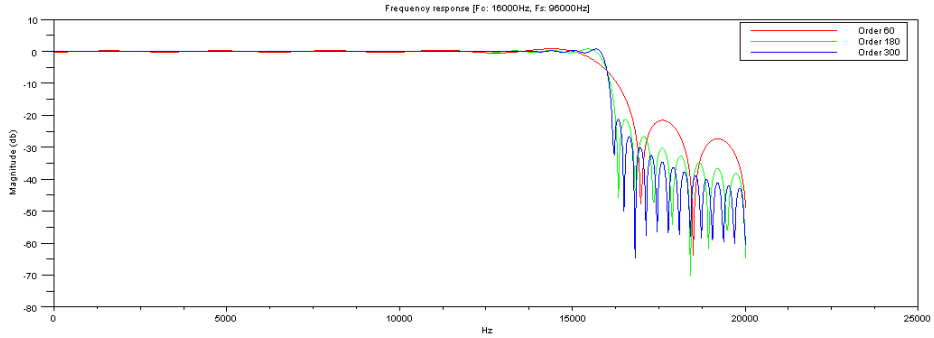


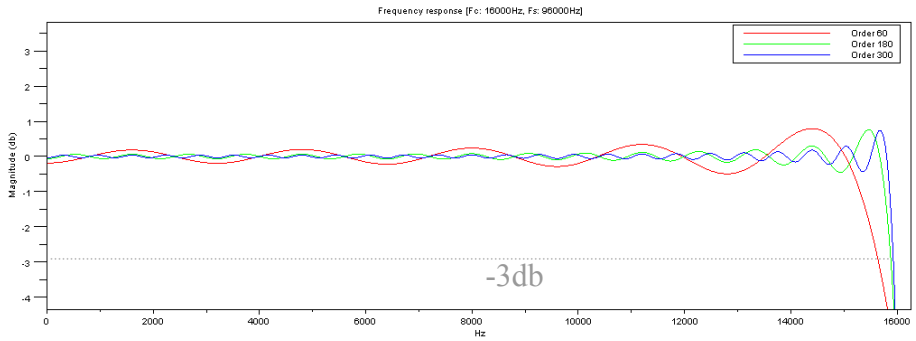
Frequency re-sampling from 32 KHz to 48 KHz (fixed method)

Using pre-generated coefficients for the filter (rectangular window):

- Filter characteristics:

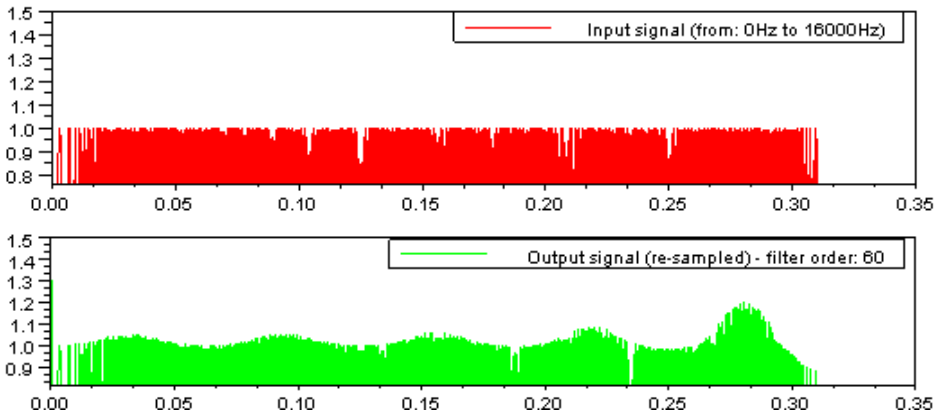


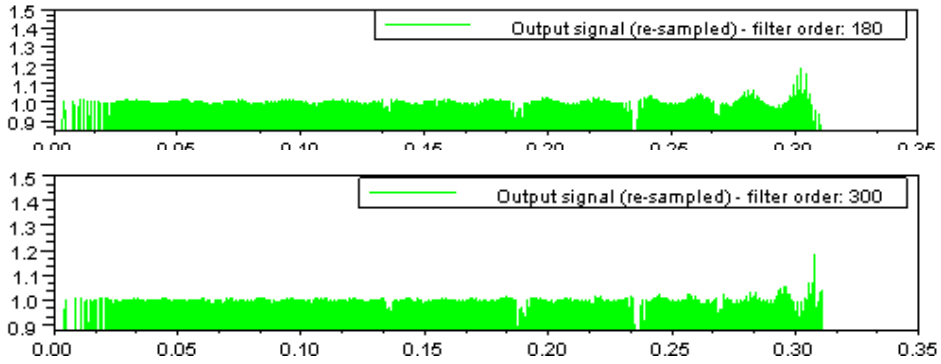




Note: The "Order" represents the filter order here.

- Re-sampling algorithm - time domain (input signal is a linear frequency sweep from 0 to 16 KHz):

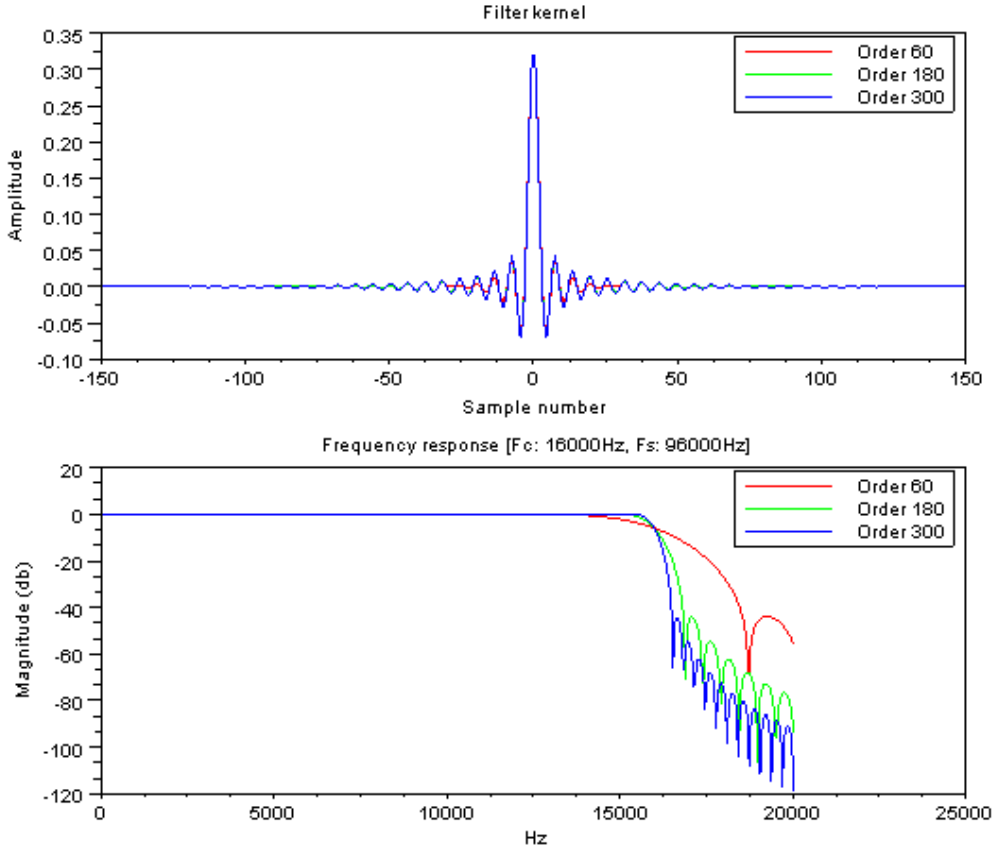


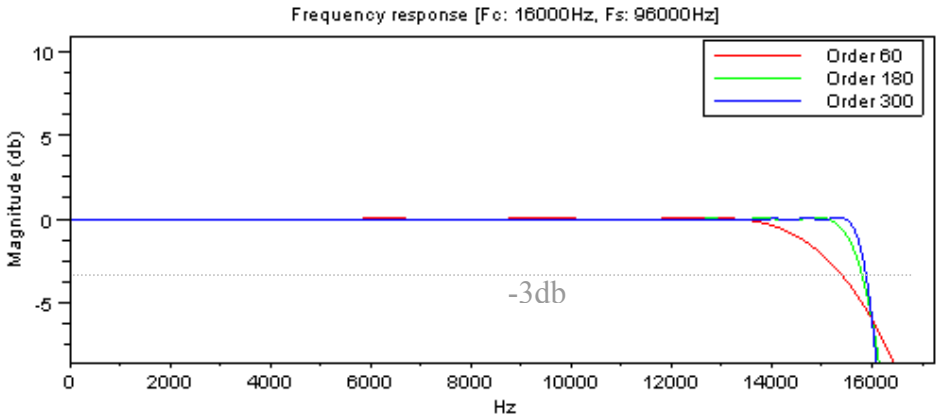


Frequency re-sampling from 32 KHz to 48 KHz (fixed method + Hann window)

Using pre-generated coefficients for the filter (**Hann window**):

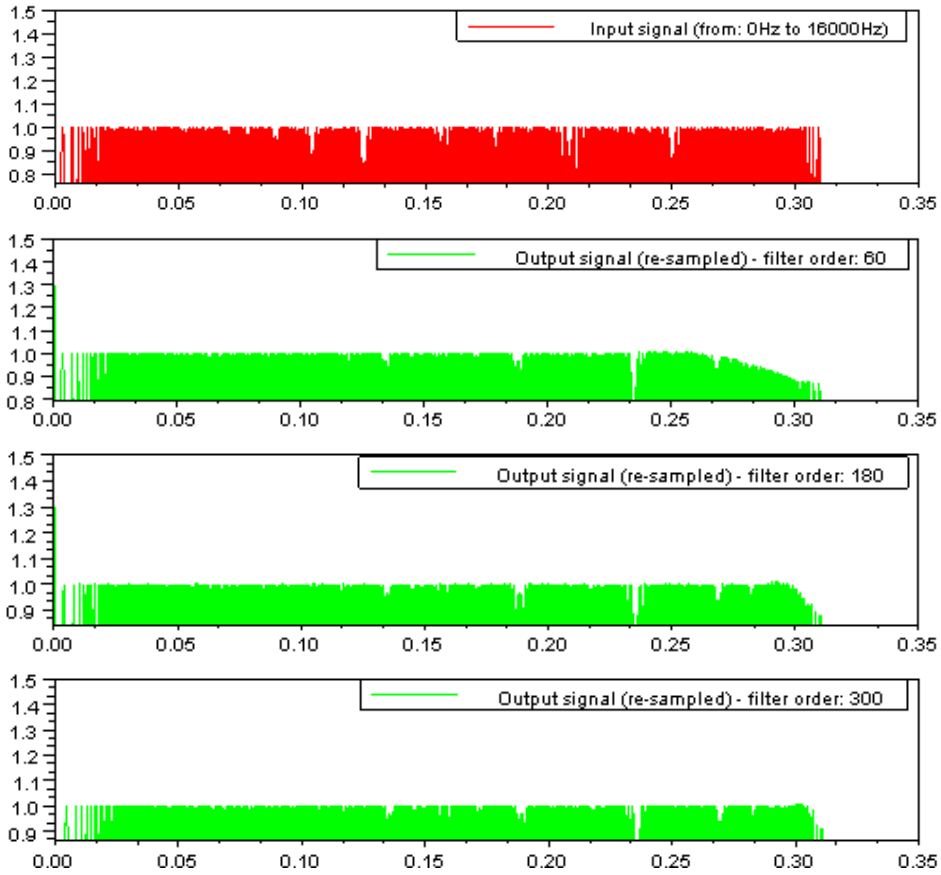
- Filter characteristics:





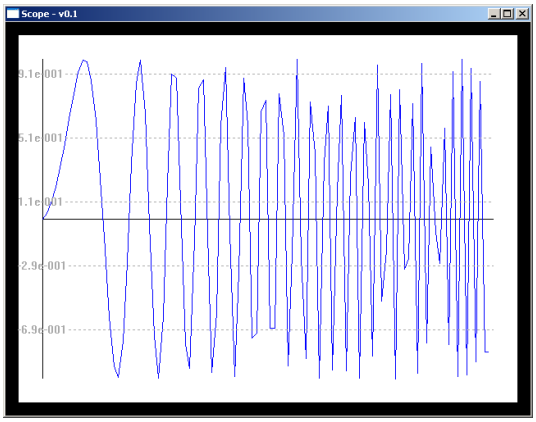
Note: The "Order" represents the filter order here.

- Re-sampling algorithm - time domain (input signal is a linear frequency sweep from 0 to 16 KHz):



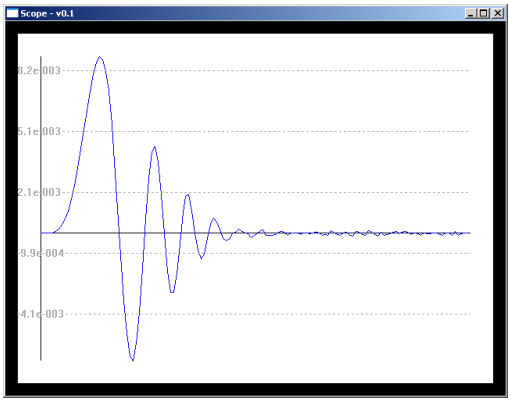
Improvements from version 1 to version 2

Re-sampling from 32 KHz to 48 KHz

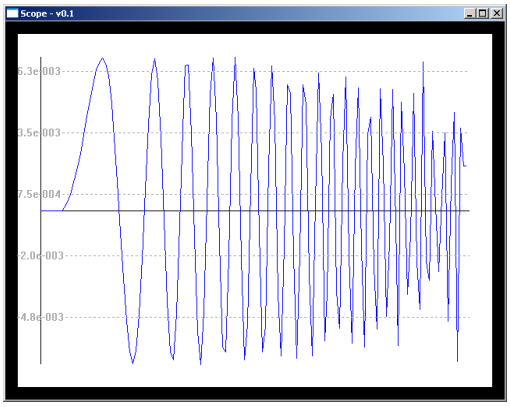


Input signal
(frequency sweep from 0 to 16 KHz)

->

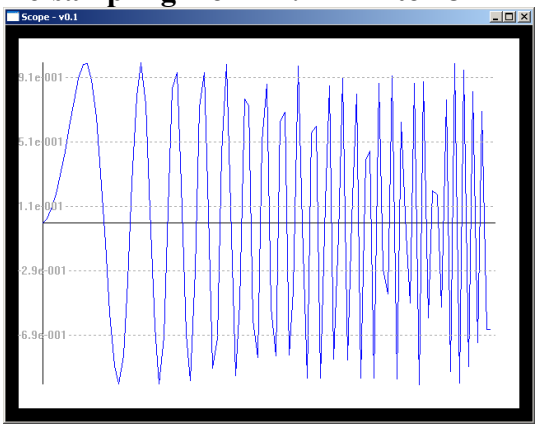


Version 1



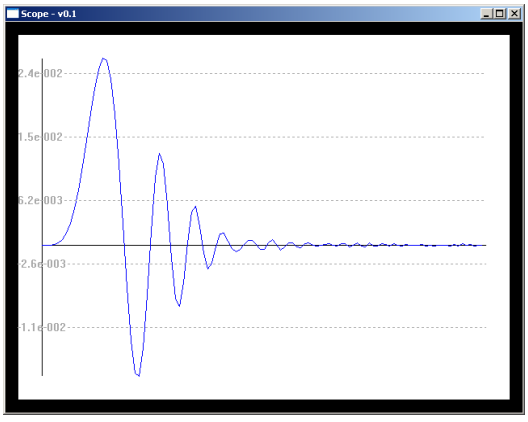
Version 2 (dynamic method)

Re-sampling from 44.1 KHz to 48 KHz



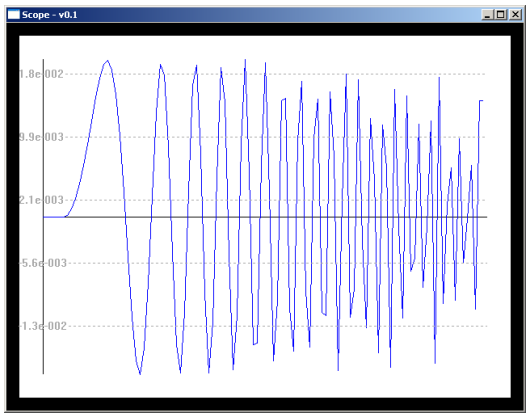
Input signal

->



Version 1

(frequency sweep from 0 to 20 KHz)



Version 2 (dynamic method)