lib_src: Sample rate conversion

Publication Date: 2026/2/3
Document Number: XM-010383-UG v2.7.1

XMOS

IN THIS DOCUMENT

1 Introduction

1.1 lib_src components

`lib_src` provides both synchronous and asynchronous audio sample rate conversion functions.

`lib_src` includes the following components:

- HiFi quality multi-rate sample rate conversion:
  - Synchronous Sample Rate Converter (SSRC) function
  - Asynchronous Sample Rate Converter (ASRC) function
- HiFi quality fixed factor sample rate conversion:

- Synchronous factor of 3 downsample function (`src_ds3`)
- Synchronous factor of 3 oversample function (`src_os3`)

- Voice quality fixed factor sample rate conversion:
  - Synchronous factor of 3 downsample function (`src_ds3_voice`)
  - Synchronous factor of 3 oversample function (`src_us3_voice`)

- Voice quality fixed factor sample rate conversion optimised for XS3:
  - Synchronous factor of 3 downsample function (`src_ff3_96t_ds`)
  - Synchronous factor of 3 oversample function (`src_ff3_96t_us`)
  - Synchronous factor of 3/2 downsample function (`src_rat_2_3_96t_ds`)
  - Synchronous factor of 3/2 oversample function (`src_rat_3_2_96t_us`)

- Integration support:
  - Asynchronous FIFO with controller for use with ASRC

The component listing above includes three different component options that support fixed factor of 3 up/downsampling. To order to choose which one to use follow these steps:

1. If HiFi quality (130 dB SNR) up/downsampling is required, use `src_ds3` or `src_os3`.
2. If voice quality (65 dB SNR) is required running on *xcore-200*, use `src_ds3_voice` or `src_us3_voice`.
3. If voice quality (75 dB SNR) is required running *xcore-ai*, use `src_ff3_96t_ds` or `src_ff3_96t_us`.

1.2 Using lib_src

`lib_src` is intended to be used with the XCommon CMake , the *XMOS* application build and dependency management system.

To use this library, include `lib_src` in the application's `APP_DEPENDENT_MODULES` list, for example:

```
set(APP_DEPENDENT_MODULES "lib_src")
```

Applications should then include the `src.h` header file.

2 HiFi quality multi-rate SRC

Both SSRC and ASRC functions are accessed via standard function calls, making them accessible from C or XC. Both SSRC and ASRC functions are passed an external state structure which provides re-entrancy. The functions may be called in-line with other signal processing or placed in a thread on it's own to provide guaranteed performance. By placing the calls to SRC functions on separate threads, multiple instances can be processed concurrently.

The API is designed to be simple and intuitive with just two public functions per sample rate converter type.

2.1 Initialisation

All public ASRC and SSRC functions are declared within the `src.h` header:

```
#include "src.h"
```

There are a number of arrays of structures that must be declared from the application which contain the buffers between the FIR stages, state and adapted coefficients (ASRC only). There must be one element of each structure declared for each channel handled by the SRC instance. The structures are then all linked into a single control structure, allowing a single reference to be passed each time a call to the SRC is made.

For SSRC, the following state structures are required:

```
//State of SSRC module
ssrc_state_t     ssrc_state[SSRC_CHANNELS_PER_INSTANCE];
//Buffers between processing stages
int              ssrc_stack[SSRC_CHANNELS_PER_INSTANCE][SSRC_STACK_LENGTH_MULT * SSRC_N_IN_SAMPLES];
//SSRC Control structure
ssrc_ctrl_t      ssrc_ctrl[SSRC_CHANNELS_PER_INSTANCE];
```

For ASRC, the following state structures are required. Note that only one instance of the filter coefficients need be declared because these are shared amongst channels within the instance:

```
//ASRC state
asrc_state_t       asrc_state[ASRC_CHANNELS_PER_INSTANCE];
int                asrc_stack[ASRC_CHANNELS_PER_INSTANCE][ASRC_STACK_LENGTH_MULT * ASRC_N_IN_SAMPLES];
//Control structure
asrc_ctrl_t        asrc_ctrl[ASRC_CHANNELS_PER_INSTANCE];
//Adaptive filter coefficients
asrc_adfir_coefs_t asrc_adfir_coefs;
```

> **Note**
>
> `lib_src` expects `SSRC_N_CHANNELS`/`ASRC_N_CHANNELS` and `SSRC_N_IN_SAMPLES`/`ASRC_N_IN_SAMPLES` to be defined by the user. It will define `SSRC_STACK_LENGTH_MULT`/`ASRC_STACK_LENGTH_MULT` based on these values. `SSRC_N_CHANNELS`/`ASRC_N_CHANNELS` values are expected to match the `n_in_samples` parameter to the initialisation functions.

There is an initialisation call which sets up the variables within the structures associated with the SRC instance and clears the inter-stage buffers. Initialisation ensures the correct selection, ordering and configuration of the filtering stages, be they decimators, interpolators or pass-through blocks. This initialisation call contains arguments defining selected input and output nominal sample rates as well as settings for the sample rate converter:

`ssrc_init()`

The initialisation call is similar for ASRC:

asrc_init()

The input block size must be a power of 2 and is function of the `n_in_samples` and `n_channels_per_instance` arguments - the total number of input samples expected for each processing call is `n_in_samples * n_channels_per_instance`.

2.2 Processing

Following initialisation, the processing API is called for each block of input samples which will then produce a block of output samples as shown in Fig. 1.

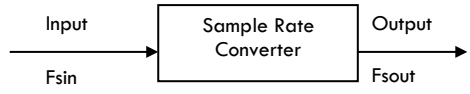


Fig. 1: SRC Operation

The logic is designed so that the final filtering stage always receives a sample to process. The sample rate converters have been designed to handle a maximum decimation of factor four from the first two stages. This architecture requires a minimum input block size of 4 to operate.

The processing function call takes the the input and output buffers and a reference to the control structure as parameters. In the case of ASRC a fractional frequency ratio parameter must also be supplied. The processing functions are as follows:

ssrc_process() asrc_process()

The SRC processing call always returns a whole number of output samples produced by the sample rate conversion. Depending on the sample ratios selected, this number may be between zero and `(n_in_samples * n_channels_per_instance * SRC_N_OUT_IN_RATIO_MAX)`. Where `SRC_N_OUT_IN_RATIO_MAX` is the maximum number of output samples for a single input sample. For example, if the input frequency is 44.1 kHz and the output rate is 192 kHz then a sample rate conversion of one sample input may produce up to 5 output samples.

The fractional number of samples produced to be carried to the next operation is stored inside the control structure, and additional whole samples are added during subsequent calls to the sample rate converter as necessary.

For example, a sample rate conversion from 44.1 kHz to 48 kHz with a input block size of 4 will produce a 4 sample result with a 5 sample result approximately every third call.

Each SRC processing call returns the integer number of samples produced during the sample rate conversion.

The SSRC is synchronous in nature and assumes that the ratio is equal to the nominal sample rate ratio. For example, to convert from 44.1 kHz to 48 kHz, it is assumed that the sample clocks of the input and output stream are derived from the same master-clock and have an exact ratio of 147:160.

If the sample clocks are derived from separate master-clocks, different oscillators for example, or are not synchronous (for example are derived from each other using a fractional PLL), ASRC must be used rather than SSRC.

2.3 Buffer formats

The format of the sample buffers sent and received from each SRC instance is time domain interleaved. How this looks in practice depends on the number of channels and SRC instances. Three examples are shown below, each showing `n_in_samples = 4`. The ordering of sample indicies is 0 representing the oldest sample and `n - 1`, where n is the buffer size, representing the newest sample.

In the case where two channels are handled by a single SRC instance Fig. 2 shows that the samples are interleaved into a single buffer of size 8.

Fig. 2: Buffer Format for Single Stereo SRC instance

Where a single audio channel is mapped to a single instance, the buffers are simply an array of samples starting with the oldest sample and ending with the newest sample as shown in Fig. 3.

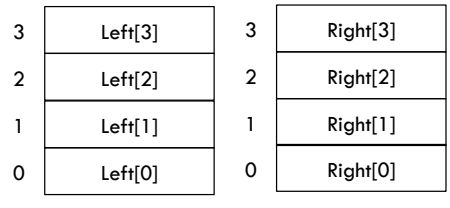


Fig. 3: Buffer Format for Dual Mono SRC instances

In the case where four channels are processed by two instances, channels 0 & 1 are processed by SRC instance 0 and channels 2 & 3 are processed by SRC instance 1 as shown in Fig. 4. For each instance, four pairs of samples are passed into the SRC processing function and n pairs of samples are returned, where n depends on the input and output sample rate ratio.

In addition to the above arguments the `asrc_process()` call also requires an unsigned Q4.60 fixed point ratio value specifying the actual input to output ratio for the next calculated block of samples. This allows the input and output rates to be fully asynchronous by allowing rate changes on each call to the ASRC. The converter dynamically computes

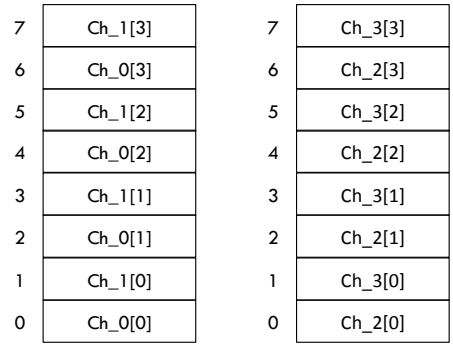


Fig. 4: Buffer Format for Dual Stereo SRC instances (4 channels total)

coefficients using a spline interpolation within the last filter stage. It is up to the callee to maintain the input and output sample rate ratio difference.

Further details about these function arguments are contained here: *SSRC API*.

2.4 Performance and resource utilisation

2.4.1 Audio performance

The performance of the SSRC library is as follows:

- THD+N (1 kHz, 0 dBFs): better than -130 dB, depending on the accuracy of the ratio estimation
- SNR: 140 dB (or better). Note that when dither is not used, SNR is infinite as output from a zero input signal is zero.

To see frequency plots illustrating the noise floor with respect to a sample rate converted tone refer to the *Performance characterisation* section of this document.

2.4.2 SSRC resource utilisation

The SSRC algorithm runs a series of cascaded FIR filters to perform the rate conversion. This includes interpolation, decimation and bandwidth limiting filters with a final polyphase FIR filter. The last stage supports the rational rate change of 147:160 or 160:147 allowing conversion between 44.1 kHz family of sample rates to the 48 kHz family of sample rates.

Tip

Table 1 shows the worst case MHz consumption at a given sample rate using the minimum block size of 4 input samples with dithering disabled. The MHz requirement can be reduced by around 8-12%, depending on sample rate, by increasing the input block size to 16. It is not usefully reduced by increasing block size beyond 16.

Tip

Table 1 is timed on xcore-200. When using xcore.ai the performance requirement is roughly halved due to VPU optimisations.

Table 1: SSRC Processor Usage per Channel (MHz) for xcore-200

		Output rate					
		44.1 kHz	48 kHz	88.2 kHz	96 kHz	176.4 kHz	192 kHz
Input rate	44.1 kHz	1 MHz	23 MHz	16 MHz	26 MHz	26 MHz	46 MHz
	48 kHz	26 MHz	1 MHz	28 MHz	17 MHz	48 MHz	29 MHz
	88.2 kHz	18 MHz	43 MHz	1 MHz	46 MHz	32 MHz	53 MHz
	96 kHz	48 MHz	20 MHz	52 MHz	2 MHz	56 MHz	35 MHz
	176.4 kHz	33 MHz	61 MHz	37 MHz	67 MHz	3 MHz	76 MHz
	192 kHz	66 MHz	36 MHz	70 MHz	40 MHz	80 MHz	4 MHz

2.4.3 ASRC performance

The performance of the ASRC library is as follows:

- THD+N: (1 kHz, 0 dBFs): better than -130 dB
- SNR: 135 dB (or better). Note that when dither is not used, SNR is infinite as output from a zero input signal is zero.

To see frequency plots illustrating the noise floor with respect to a sample rate converted tone refer to the *Performance characterisation* section of this document.

2.4.4 ASRC latency / group delay

The ASRC algorithm runs a series of cascaded FIR filters to perform the rate conversion including a final adaptive filter to handle the varying rate change between the input and the output. The latency or group delay through the filter depends on the input rate and output rate and the input block size. Table 2 quantifies the amount of latency in milliseconds seen from the filtering. For input block sizes of greater than four please add *1000 * (INPUT_BLOCK_SIZE - 4) / INPUT_SAMPLE_RATE* milliseconds to the numbers in the table.

Table 2: ASRC total filter delay by sample rate in milliseconds for input block size of four

		Output rate					
		44.1 kHz	48 kHz	88.2 kHz	96 kHz	176.4 kHz	192 kHz
Input rate	44.1 kHz	0.907	0.899	0.896	0.899	0.896	0.894
	48 kHz	0.910	0.833	0.830	0.823	0.825	0.823
	88.2 kHz	0.907	0.908	0.454	0.450	0.448	0.450
	96 kHz	0.931	0.833	0.455	0.417	0.415	0.411
	176.4 kHz	0.998	0.996	0.317	0.319	0.159	0.157
	192 kHz	0.998	0.917	0.295	0.292	0.148	0.146

ASRC resource utilisation The ASRC algorithm runs a series of cascaded FIR filters to perform the rate conversion. The final filter is different because it uses adaptive coefficients to handle the varying rate change between the input and the output. The adaptive coefficients must be computed for each output sample period, but can be shared amongst all channels within the ASRC instance. Consequently, the MHz usage of the ASRC is expressed as two tables; Table 3 quantifies the MHz required for the first channel with adaptive coefficients calculation and Table 4 specifies the MHz required for filtering of each additional channel processed by the ASRC instance.

Tip

The below tables show the worst case MHz consumption per sample, using the minimum block size of 4 input samples. The MHz requirement can be reduced by around 8-12% by increasing the input block size to 16.

Tip

Typically some performance headroom is needed for buffering (especially if the system is sample orientated rather than block orientated) and inter-task communication.

Tip

Table 3 is timed on xcore-200. When using xcore.ai the performance requirement is roughly halved due to VPU optimisations.

Table 3: ASRC Processor Usage (MHz) for the First Channel in the ASRC Instance for xcore-200

		Output rate					
		44.1 kHz	48 kHz	88.2 kHz	96 kHz	176.4 kHz	192 kHz
Input rate	44.1 kHz	29 MHz	30 MHz	40 MHz	42 MHz	62 MHz	66 MHz
	48 kHz	33 MHz	32 MHz	42 MHz	43 MHz	63 MHz	66 MHz
	88.2 kHz	47 MHz	50 MHz	58 MHz	61 MHz	80 MHz	85 MHz
	96 kHz	55 MHz	51 MHz	67 MHz	64 MHz	84 MHz	87 MHz
	176.4 kHz	60 MHz	66 MHz	76 MHz	81 MHz	105 MHz	106 MHz
	192 kHz	69 MHz	66 MHz	82 MHz	82 MHz	109 MHz	115 MHz

Caution

Configurations requiring more than 100 MHz may not be able run in real time on a single logical core. The performance limit for a single core on a 500 MHz xcore-200 device is 100 MHz (500/5) however an xcore-ai device running at 600 MHz can provide 120 MHz logical cores.

Tip

Table 4 is timed on xcore-200. When using xcore.ai the performance requirement is roughly halved due to VPU optimisations.

Table 4: ASRC Processor Usage (MHz) for Subsequent Channels in the ASRC Instance

		Output rate					
		44.1 kHz	48 kHz	88.2 kHz	96 kHz	176.4 kHz	192 kHz
Input rate	44.1 kHz	28 MHz	28 MHz	32 MHz	30 MHz	40 MHz	40 MHz
	48 kHz	39 MHz	31 MHz	33 MHz	36 MHz	40 MHz	45 MHz
	88.2 kHz	51 MHz	49 MHz	57 MHz	55 MHz	65 MHz	60 MHz
	96 kHz	51 MHz	56 MHz	57 MHz	62 MHz	66 MHz	71 MHz
	176.4 kHz	60 MHz	66 MHz	76 MHz	79 MHz	92 MHz	91 MHz
	192 kHz	69 MHz	66 MHz	76 MHz	82 MHz	90 MHz	100 MHz

2.5 Implementation detail

The SSRC and ASRC implementations are closely related to each other and share the majority of the system building blocks. The key difference between them is that SSRC uses fixed polyphase 160:147 and 147:160 final rate change filters whereas the ASRC uses an adaptive polyphase filter. The ASRC adaptive polyphase coefficients are computed for every sample using second order spline based interpolation.

2.5.1 SSRC structure

The SSRC algorithm is based on three cascaded FIR filter stages (F1, F2 and F3). These stages are configured differently depending on rate change and only part of them is used in certain cases. Fig. 5 shows an overall view of the SSRC algorithm:

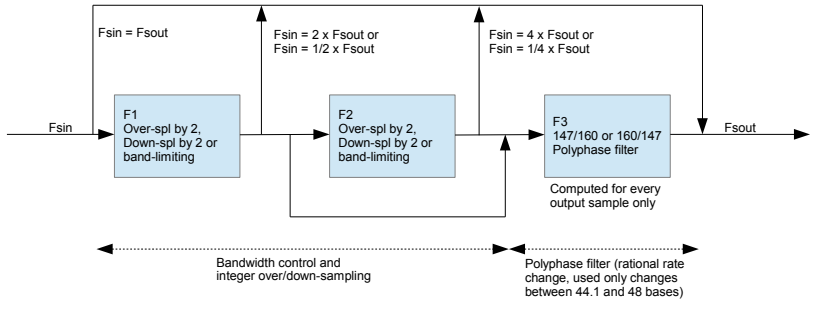


Fig. 5: SSRC Algorithm Structure

The SSRC algorithm is implemented as a two stage structure:

- The bandwidth control stage which includes filters F1 and F2 is responsible for limiting the bandwidth of the input signal and for providing integer rate Sample Rate Conver-

sion. It is also used for signal conditioning in the case of rational non-integer Sample Rate Conversion.

- The polyphase filter stage which converts between the 44.1 kHz and the 48 kHz families of sample rates.

2.5.2 ASRC structure

Similar to the SSRC, the ASRC algorithm is based on three cascaded FIR filters (F1, F2 and F3). These are configured differently depending on rate change and F2 is not used in certain rate changes. Fig. 6 shows an overall view of the ASRC algorithm:

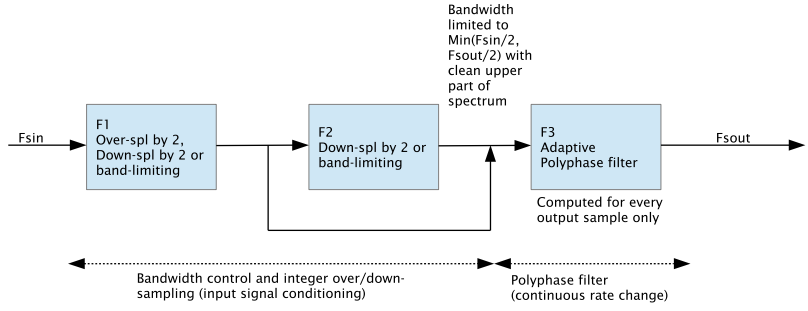


Fig. 6: ASRC Algorithm Structure

The ASRC algorithm is implemented as a two stage structure:

- The bandwidth control stage includes filters F1 and F2 which are responsible for limiting the bandwidth of the input signal and for providing integer rate sample rate conversion to condition the input signal for the adaptive polyphase stage (F3).
- The polyphase filter stage consists of the adaptive polyphase filter F3, which effectively provides the asynchronous connection between the input and output clock domains.

2.5.3 SRC filter list

A complete list of the filters supported by the SRC library, both SSRC and ASRC, is shown in Table 5. The filters are implemented in C within the `FilterDefs.c` function and the coefficients can be found in the `/FilterData` folder. The particular combination of filters cascaded together for a given sample rate change is specified in `ssrc.c` and `asrc.c`.

Table 5: SRC Filter Specifications

Filter	Fs (norm)	Pass-band	Stop-band	Rip-ple	Atten-uation	Taps	Notes
BL	2	0.454	0.546	0.01 dB	155 dB	144	Down-sampler by two, steep
BL9644	2	0.417	0.501	0.01 dB	155 dB	160	Low-pass filter, steep for 96 to 44.1
BL8848	2	0.494	0.594	0.01 dB	155 dB	144	Low-pass, steep for 88.2 to 48
BLF	2	0.41	0.546	0.01 dB	155 dB	96	Low-pass at half band
BL19288	2	0.365	0.501	0.01 dB	155 dB	96	Low pass, steep for 192 to 88.2
BL17696	2	0.455	0.594	0.01 dB	155 dB	96	Low-pass, steep for 176.4 to 96
UP	2	0.454	0.546	0.01 dB	155 dB	144	Over sample by 2, steep
UP4844	2	0.417	0.501	0.01 dB	155 dB	160	Over sample by 2, steep for 48 to 44.1
UPF	2	0.41	0.546	0.01 dB	155 dB	96	Over sample by 2, steep for 176.4 to 192
UP192176	2	0.365	0.501	0.01 dB	155 dB	96	Over sample by 2, steep for 192 to 176.4
DS	4	0.57	1.39	0.01 dB	160 dB	32	Down sample by 2, re-laxed
OS	2	0.57	1.39	0.01 dB	160 dB	32	Over sample by 2, re-laxed
HS294	284	0.55	1.39	0.01 dB	155 dB	2352	Polyphase 147/160 rate change
HS320	320	0.55	1.40	0.01 dB	151 dB	2560	Polyphase 160/147 rate change
AD-FIR	256	0.45	1.45	0.012 dB	170 dB	1920	Adaptive polyphase prototype filter

2.6 File structure and overview

All source files for the SSRC and ASRC are located within the `multirate_hifi` subdirectory.

- src_mrhf_ssrc_wrapper.c / src_mrhf_ssrc_wrapper.h

  These wrapper files provide a simplified public API to the SSRC initialization and processing functions.

- src_mrhf_asrc_wrapper.c / src_mrhf_asrc_wrapper.h

  These wrapper files provide a simplified public API to the ASRC initialization and processing functions.

- src_mrhf_ssrc.c / src_mrhf_ssrc.h

  These files contain the core of the SSRC algorithm. They set up the correct filtering chains depending on rate change and apply in the processing calls. The table sFilter-

sIDs declared in SSRC.c contains definitions of the filter chains for all supported rate changes. The files also integrate the code for the optional dithering function.

- src_mrhf_asrc.c / src_mrhf_asrc.h

  These files contain the core of the ASRC algorithm. They setup the correct filtering chains depending on rate change and apply them for the corresponding processing calls. Note that filters F1, F2 and dithering are implemented using a block based approach similar to SSRC. The adaptive polyphase filter (ADFIR) is implemented on a sample by sample basis. These files also contain functions to compute the adaptive polyphase filter coefficients.

- src_mrhf_fir.c / src_mrhf_fir.h

  These files provide Finite Impulse Response (FIR) filtering setup, with calls to the assembler-optimized inner loops. They provide functions for handling down-sampling by 2, synchronous or over-sampling by 2 FIRs. They also provides functions for handling polyphase filters used for rational ratio rate change in the SSRC and adaptive FIR filters used in the asynchronous section of the ASRC.

- src_mrhf_filter_defs.c / src_mrhf_filter_defs.h

  These files define the size and coefficient sources for all the filters used by the SRC algorithms.

- /FilterData directory (various files)

  This directory contains the pre-computed coefficients for all of the fixed FIR filters. The numbers are stored as signed Q1.31 format and are directly included in the source of FilterDefs.c. Both the .dat files used by the C compiler and the .sfp ScopeFIR (http://iowegian.com/scopefir/) design source files, used to originally create the filters, are included.

- src_mrhf_fir_inner_loop_asm.S / src_mrhf_fir_inner_loop_asm.h

  Inner loop for the standard FIR function optimized for double-word load and store, 32 bit * 32 bit -> 64 bit MACC and saturation instructions. Even and odd sample long word alignment versions are provided.

- src_mrhf_fir_os_inner_loop_asm.S / scr_mrhf_fir_os_inner_loop_asm.h

  Inner loop for the oversampling FIR function optimized for double-word load and store, 32 bit * 32 bit -> 64 bit MACC and saturation instructions. Both (long word) even and odd sample input versions are provided.

- src_mrhf_spline_coeff_gen_inner_loop_asm.S / src_mrhf_spline_coeff_gen_inner_loop_asm.h

  Inner loop for generating the spline interpolated coefficients. This assembler function is optimized for double-word load and store, 32 bit * 32 bit -> 64 bit MACC and saturation instructions.

- src_mrhf_adfir_inner_loop_asm.S / src_mrhf_adfir_inner_loop_asm.h

  Inner loop for the adaptive FIR function using the previously computed spline interpolated coefficients. It is optimized for double-word load and store, 32 bit * 32 bit -> 64 bit MACC and saturation instructions. Both (long word) even and odd sample input versions are provided.

- src_mrhf_int_arithmetic.c / src_mrhf_int_arithmetic.h

  These files contain simulation implementations of XMOS ISA specific assembler instructions. These are only used for dithering functions, and may be eliminated during future optimizations.

2.7 SSRC API

void **ssrc_init**(
const fs_code_t sr_in, const fs_code_t sr_out, ssrc_ctrl_t ssrc_ctrl[], const unsigned n_channels_per_instance, const unsigned n_in_samples, const dither_flag_t dither_on_off,
)

initializes synchronous sample rate conversion instance.

Parameters

- **sr_in** – Nominal sample rate code of input stream
- **sr_out** – Nominal sample rate code of output stream
- **ssrc_ctrl** – Reference to array of SSRC control stuctures
- **n_channels_per_instance** – Number of channels handled by this instance of SSRC
- **n_in_samples** – Number of input samples per SSRC call
- **dither_on_off** – Dither to 24b on/off

unsigned **ssrc_process**(int in_buff[], int out_buff[], ssrc_ctrl_t ssrc_ctrl[])

Perform synchronous sample rate conversion processing on block of input samples using previously initialized settings.

Parameters

- **in_buff** – Reference to input sample buffer array
- **out_buff** – Reference to output sample buffer array
- **ssrc_ctrl** – Reference to array of SSRC control stuctures

Returns

The number of output samples produced by the SRC operation

2.8 ASRC API

uint64_t **asrc_init**(
const fs_code_t sr_in, const fs_code_t sr_out, asrc_ctrl_t asrc_ctrl[], const unsigned n_channels_per_instance, const unsigned n_in_samples, const dither_flag_t dither_on_off,
)

initializes asynchronous sample rate conversion instance.

Parameters

- **sr_in** – Nominal sample rate code of input stream
- **sr_out** – Nominal sample rate code of output stream
- **asrc_ctrl** – Reference to array of ASRC control structures
- **n_channels_per_instance** – Number of channels handled by this instance of SSRC
- **n_in_samples** – Number of input samples per SSRC call
- **dither_on_off** – Dither to 24b on/off

Returns

The nominal sample rate ratio of in to out in Q4.60 format

unsigned **asrc_process**(
int in_buff[], int out_buff[], uint64_t fs_ratio, asrc_ctrl_t asrc_ctrl[],
)

Perform asynchronous sample rate conversion processing on block of input samples using previously initialized settings.

Parameters

- **in_buff** – Reference to input sample buffer array
- **out_buff** – Reference to output sample buffer array
- **fs_ratio** – Fixed point ratio of in/out sample rates in Q4.60 format
- **asrc_ctrl** – Reference to array of ASRC control structures

Returns

The number of output samples produced by the SRC operation.

3 HiFi quality fixed factor of 3 SRC

The SRC library includes synchronous sample rate conversion functions to downsample (decimate) and oversample (upsample or interpolate) by a fixed factor of 3.

These components offer a high quality conversion with an SNR of 130 dB.

In each case, the processing is carried out each time a single output sample is required. In the case of the decimator, three input samples are passed to the filter with a resulting one sample output on calling the processing function. The interpolator produces an output sample each time the processing function is called but will require a single sample to be pushed into the filter every third cycle. All samples use Q1.31 format (left justified signed 32b integer).

Both sample rate converters are based on a 144 tap FIR filter with two sets of coefficients available, depending on application requirements:

- `firos3_b_144.dat` / `firds3_b_144.dat` - These filters have 20 dB of attenuation at the Nyquist frequency and a higher cutoff frequency
- `firos3_144.dat` / `firds3_144.dat` - These filters have 60 dB of attenuation at the Nyquist frequency but trade this off with a lower cutoff frequency

The default setting is to use the coefficients that provide 60 dB of attenuation at the Nyquist frequency.

The filter coefficients may be selected by adjusting the line:

```
#define   FIROS3_COEFS_FILE
```

and:

```
#define   FIRDS3_COEFS_FILE
```

in the files `src_ff3_os3.h` (API for oversampling) and `src_ff3_ds3.h` (API for downsampling) respectively.

The OS3 processing takes up to 153 core cycles to compute a sample which translates to 1.53 μs at 100 MHz or 2.448 μs at 62.5 MHz core speed. This permits up to 8 channels of 16 kHz -> 48 kHz sample rate conversion in a single 62.5MHz core.

The DS3 processing takes up to 389 core cycles to compute a sample which translates to 3.89 μs at 100 MHz or 6.224 μs at 62.5 MHz core speed. This permits up to 9 channels of 48 kHz -> 16 kHz sample rate conversion in a single 62.5MHz core.

Both downsample and oversample functions return `ERROR` or `NO_ERROR` status codes as defined in the return code enums listed below. The only way these functions can error is if the passed *delay_base* structure member is uninitialized (NULL).

The downsampling functions return the following error codes

```
FIRDS3_NO_ERROR
FIRDS3_ERROR
```

The upsampling functions return the following error codes

```
FIROS3_NO_ERROR
FIROS3_ERROR
```


Tip

There are three different component options that support fixed factor of 3 up/down-sampling. To help choose which one to use follow these steps: If HiFi quality (130 dB SNR) up/downsampling is required, use ds3 or os3. If voice quality (65 dB SNR) is required running on xcore-200, use ds3_voice or us3_voice. If voice quality (75 dB SNR) is required running xcore-ai, use ff3_96t_ds or ff3_96t_us

3.1 Shared API items

enum **src_ff3_return_code_t**

Fixed factor of 3 return codes

This type describes the possible error status states from calls to the DS3 and OS3 API.

Values:

enumerator **SRC_FF3_NO_ERROR**

enumerator **SRC_FF3_ERROR**

3.2 HiFi quality DS3 API

struct **src_ds3_ctrl_t**

Downsample by 3 control structure

src_ff3_return_code_t **src_ds3_init**(*src_ds3_ctrl_t* *src_ds3_ctrl)

This function initializes the decimate by 3 function for a given instance

Parameters

- **src_ds3_ctrl** – DS3 control structure

Returns

SRC_FF3_NO_ERROR on success, SRC_FF3_ERROR on failure

src_ff3_return_code_t **src_ds3_sync**(*src_ds3_ctrl_t* *src_ds3_ctrl)

This function clears the decimate by 3 delay line for a given instance

Parameters

- **src_ds3_ctrl** – DS3 control structure

Returns

SRC_FF3_NO_ERROR on success, SRC_FF3_ERROR on failure

src_ff3_return_code_t **src_ds3_proc**(*src_ds3_ctrl_t* *src_ds3_ctrl)

This function performs the decimation on three input samples and outputs one sample. The input and output buffers are pointed to by members of the src_ds3_ctrl structure

Parameters

- **src_ds3_ctrl** – DS3 control structure

Returns

SRC_FF3_NO_ERROR on success, SRC_FF3_ERROR on failure

3.3 HiFi quality OS3 API

struct **src_os3_ctrl_t**

Oversample by 3 control structure

src_ff3_return_code_t **src_os3_init**(*src_os3_ctrl_t* *src_os3_ctrl)

This function initializes the oversample by 3 function for a given instance

Parameters

- **src_os3_ctrl** – OS3 control structure

Returns

SRC_FF3_NO_ERROR on success, SRC_FF3_ERROR on failure

src_ff3_return_code_t **src_os3_sync**(*src_os3_ctrl_t* *src_os3_ctrl)

This function clears the oversample by 3 delay line for a given instance

Parameters

- **src_os3_ctrl** – OS3 control structure

Returns

SRC_FF3_NO_ERROR on success, SRC_FF3_ERROR on failure

src_ff3_return_code_t **src_os3_input**(*src_os3_ctrl_t* *src_os3_ctrl)

This function pushes a single input sample into the filter. It should be called three times for each FIROS3_proc call

Parameters

- **src_os3_ctrl** – OS3 control structure

Returns

SRC_FF3_NO_ERROR on success, SRC_FF3_ERROR on failure

src_ff3_return_code_t **src_os3_proc**(*src_os3_ctrl_t* *src_os3_ctrl)

This function performs the oversampling by 3 and outputs one sample. The input and output buffers are pointed to by members of the src_os3_ctrl structure

Parameters

- **src_os3_ctrl** – OS3 control structure

Returns

SRC_FF3_NO_ERROR on success, SRC_FF3_ERROR on failure

4 Voice quality fixed factor of 3 SRC

A pair of SRC components supporting upconversion and downconversion by a factor of 3 are provided that are suitable for voice applications. They provide voice quality SNR (around 60 dB) and use a 72 tap Remez FIR filter and are optimized for the XS2 instruction set.

Warning

These SRC components have been deprecated. For new designs using `xcore-ai`, use the XS3 optimised components which provide both better performance and use approximately half of the MIPS. See *ff3_voice_vpu_hdr*

4.1 Voice quality DS3 API

int64_t **src_ds3_voice_add_sample**(
    int64_t sum, int32_t data[], const int32_t coefs[], int32_t sample,
)

This function performs the first two iterations of the downsampling process

Parameters

- **sum** – Partially accumulated value returned during previous cycle
- **data** – Data delay line
- **coefs** – FIR filter coefficients
- **sample** – The newest sample

Returns

Partially accumulated value, passed as `sum` parameter next cycle

int64_t **src_ds3_voice_add_final_sample**(
    int64_t sum, int32_t data[], const int32_t coefs[], int32_t sample,
)

This function performs the final iteration of the downsampling process

Parameters

- **sum** – Partially accumulated value returned during previous cycle
- **data** – Data delay line
- **coefs** – FIR filter coefficients
- **sample** – The newest sample

Returns

The decimated sample

4.2 Voice quality US3 API

int32_t **src_us3_voice_input_sample**(
    int32_t data[], const int32_t coefs[], int32_t sample,
)

This function performs the initial iteration of the upsampling process

Parameters

- **data** – Data delay line
- **coefs** – FIR filter coefficients
- **sample** – The newest sample

Returns

A decimated sample

int32_t **src_us3_voice_get_next_sample**(int32_t data[], const int32_t coefs[])

This function performs the final two iterations of the upsampling process

Parameters

- **data** – Data delay line
- **coefs** – FIR filter coefficients

Returns

A decimated sample

5 Voice quality fixed factor of 3 and 3/2 SRC optimised for XS3

A set of SRC components are provided which are optimised for the Vector Processing Unit (VPU) and are suitable for voice applications. They cannot be run on XS2 based devices.

The fixed factor of 3 SRC components are designed for conversion between 48 kHz to 16 kHz and the fixed factor of 3/2 are designed for conversion between 48 kHz and 32 kHz.

They have been designed for voice applications and, in particular, conformance to the *Microsoft Teams* v5 specification.

Warning

Synchronous fixed factor of 3 and 3/2 downsample and oversample functions for voice applications optimised for the XS3 Vector Processing Unit currently overflow rather than saturate in cases where a full scale input causes a perturbation above full scale at the output. To avoid this scenario, ensure that the input amplitude is always 3.5 dB below full scale.

5.1 Fixed factor of 3 VPU implementation

The filters use less than half of the cycles of the previous fixed factor of 3 functions but at the same time offer a much improved filter response thanks to an increased filter length of 96 taps (compared with 72 taps) and use of a Kaiser window with a beta of 4.0. The filter specification is shown in Table 6.

Table 6: Fixed Factor of 3 Voice VPU SRC characteristics

Filter	CPU cycles	Pass-band	Stop-band	Ripple	Attenuation	Taps
src_ff3_96t_ds	104	0.475	0.525	0.01 dB	70 dB min	96
src_ff3_96t_us	85	0.475	0.525	0.01 dB	70 dB min	96

The fixed factor of 3 components produce three samples for each call passing one sample in the case of upsampling and produce a single sample for each call passing three samples in the case of downsampling. All input and output samples are signed 32 bit integers. The filter characteristics are shown in Fig. 7 and Fig. 8.

5.2 Voice quality DS3 VPU API

static inline void **src_ff3_96t_ds**(
    int32_t samp_in[3], int32_t samp_out[1], const int32_t coefs_ff3[3][32], int32_t state_ds[3][32],
)

Performs VPU-optimised 96 taps polyphase fixed-factor-of-3 downsampling.

Parameters

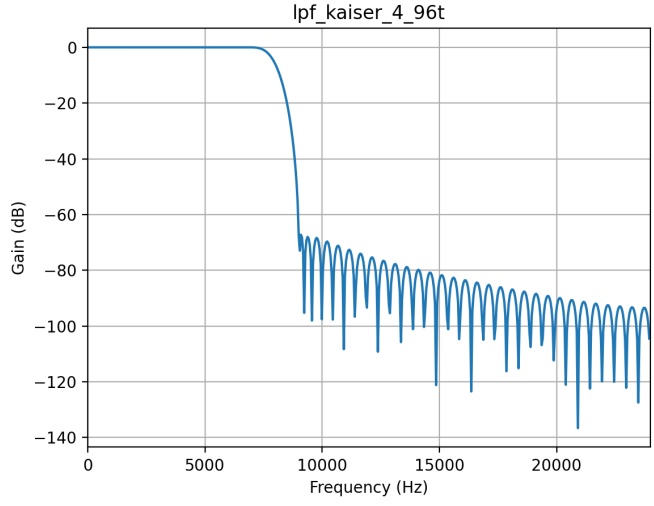


Fig. 7: Fixed Factor of 3 Voice VPU SRC filter response

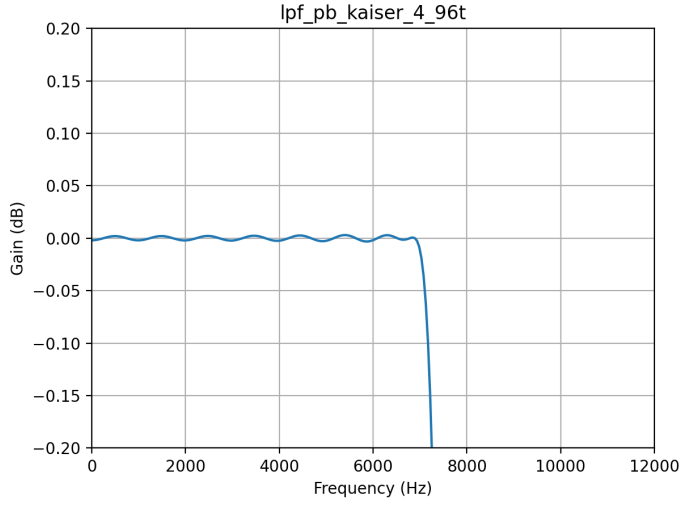


Fig. 8: Fixed Factor of 3 Voice VPU SRC passband ripple

- **samp_in** – Values to be downsampled
- **samp_out** – Downsampled output
- **coefs_ff3** – Three-phase FIR coefficients array with [3][32] dimensions
- **state_ds** – Three-phase FIR state array with [3][32] dimensions

5.3 Voice quality US3 VPU API

static inline void **src_ff3_96t_us**(
    int32_t samp_in[1], int32_t samp_out[3], const int32_t coefs_ff3[3][32], int32_t state_us[32],
)

Performs VPU-optimised 96 taps polyphase fixed-factor-of-3 upsampling.

> **Note**
>
> samp_in and samp_out have to be different memory locations

Parameters

- **samp_in** – Value to be upsampled
- **samp_out** – Upsampled output
- **coefs_ff3** – Three-phase FIR coefficients array with [3][32] dimensions
- **state_us** – FIR state array with 32 elements in it

5.4 Fixed factor of 3/2 VPU implementation

The fixed factor of 3/2 VPU sample rate converts use a rational factor polyphase architecture to achieve the non-integer rate ratio. Downsampling takes two phases while upsampling takes three. The filters have been designed with a Kaiser window with a beta of 3.2. The filter specification is shown in Table 7.

Table 7: Fixed Factor of 3/2 Voice VPU SRC characteristics

Filter	CPU cycles	Pass-band	Stop-band	Ripple	Attenuation	Taps
src_rat_2_3_96t_ds	112	0.46875	0.53125	0.03 dB	70 dB	96
src_rat_3_2_96t_us	95	0.46875	0.53125	0.03 dB	70 dB	96

The fixed factor of 3/2 components produce three samples for each call passing two samples in the case of upsampling and produce two samples for each call passing three samples in the case of downsampling. All input and output samples are signed 32 bit integers. The filter characteristics are shown in Fig. 9 and Fig. 10.

5.5 Voice quality DS3/2 API

static inline void **src_rat_2_3_96t_ds**(
    int32_t samp_in[3], int32_t samp_out[2], const int32_t coefs_ds[2][48], int32_t state_ds[48],
)

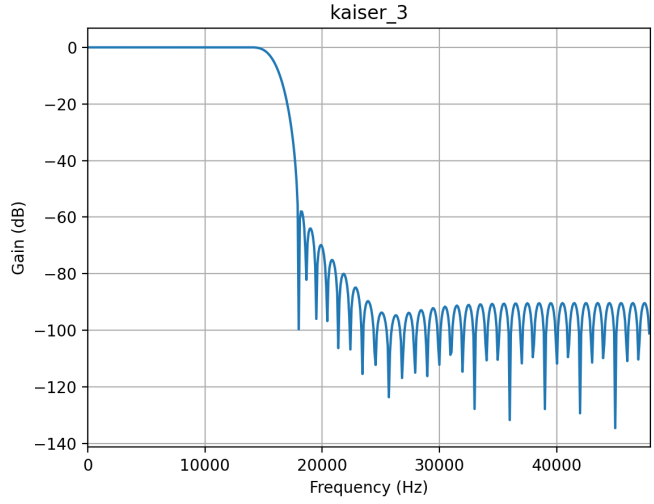


Fig. 9: Fixed Factor of 3/2 Voice VPU SRC filter response

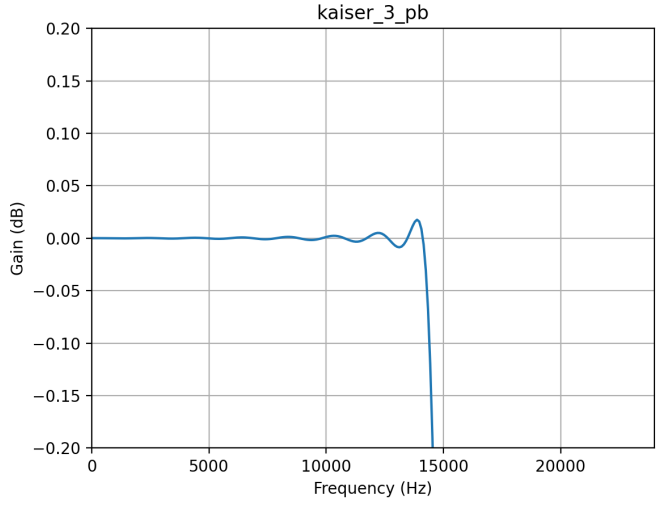


Fig. 10: Fixed Factor of 3/2 Voice VPU SRC passband ripple

Performs VPU-optimised 96 taps polyphase rational factor 2/3 downsampling.

Parameters

- **samp_in** – Values to be downsampled
- **samp_out** – Downsampled output
- **coefs_ds** – Two-phase FIR coefficients array with [2][48] dimensions
- **state_ds** – FIR state array with 48 elements in it

5.6 Voice quality US3/2 API

static inline void **src_rat_3_2_96t_us**(
int32_t samp_in[2], int32_t samp_out[3], const int32_t coefs_us[3][32], int32_t state_us[32],
)

Performs VPU-optimised 96 taps polyphase rational factor 3/2 upsampling.

> **Note**
>
> samp_in and samp_out have to be different memory locations

Parameters

- **samp_in** – Values to be upsampled
- **samp_out** – Upsampled output
- **coefs_us** – Three-phase FIR coefficients array with [3][32] dimensions
- **state_us** – FIR state array with 32 elements in it

6 Asynchronous FIFO

An Asynchronous FIFO is a non-blocking data structure in which elements gets pushed in on one side and pulled out on the other side. It is primarily designed to be used with the ASRC to help build practical systems. The keys to this component are:

1. The non-blocking nature of the interfaces on both sides.

2. An underlying assumption that software on both sides rate-matches their requests.

The asynchronous FIFO has a PID control inside it that can be used to control the rate of either the producer or the consumer.

Two typical use cases are shown in *Use cases for the asynchronous FIFO*. In the first use case there is an Asynchronous Sample Rate Converter (ASRC) in front of the FIFO. The task of this ASRC is to dynamically introduce or remove samples in order to match the rate of producer and consumer. In the second use case there is a PLL (either hardware or software) that is used to match the rate of the producer and consumer.

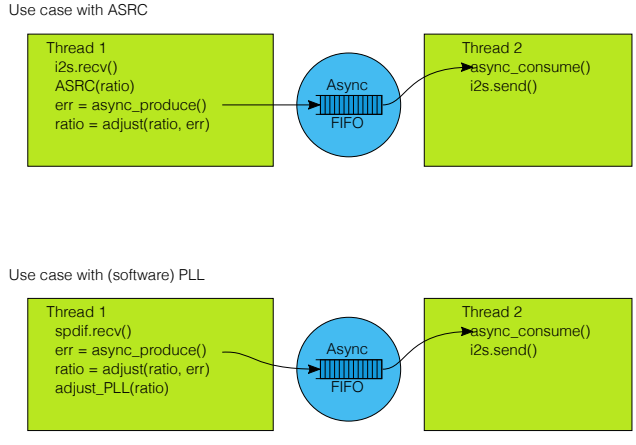


Fig. 11: Use cases for the asynchronous FIFO

In order to use the asynchronous FIFO one needs at least two threads that are located on the same tile. A producer thread (on the left), and a consumer thread (on the right). These threads are free-running relative to each other, and the FIFO transports data from the producer to the consumer. Free-running means that the threads can simultaneously access the FIFO without being able to observe a change in timing.

The FIFO has a fixed length, set on creation, and the control algorithm inside the FIFO tries and keep the FIFO half-full at all times. When the producer is slower than the consumer the FIFO will drain a bit until the rates match again, and when the producer is faster than the consumer the FIFO will grow until the rates match again. In order to ensure that the FIFO stays half full, the control algorithm will always slightly overshoot on a change in relative rates. Note that the FIFO is unaware whether it is the producer that is too fast, or the consumer that is too slow. It does not attribute blame for a rate-mismatch. The FIFO just observes the mismatch.

6.1 Using the asynchronous FIFO

An Asynchronous FIFO is allocated as an array of double-word integers:

```
int64_t array[ASYNCHRONOUS_FIFO_INT64_ELEMENTS(ENTRIES, SAMPLE_SIZE)];
```

The `ASYNCHRONOUS_FIFO_INT64_ELEMENTS()` macro calculates the number of double words required for the FIFO given the number of entries in the FIFO, and the number of words that each sample occupies. For example, when transferring stereo Audio through a fifo with 40 elements one would use `ASYNCHRONOUS_FIFO_INT64_ELEMENTS(40, 2)`. Note that the two elements are not interchangeable. The number 40 is the total number of elements in the FIFO, in this case the FIFO will be started half-full, so the first 20 elements read will be zeroes, after which the produced data will appear on the consumer side.

The number of elements in the FIFO is a trade-off that the system designer makes. As the FIFO will always aim to be half-full, a large number of elements will introduce a high latency in the system and occupy a large amount of memory. A short FIFO will contribute little latency but may easily overflow and underflow. More on this in *Design parameters*.

The Asynchronous FIFO has the following functions to control the FIFO:

- `asynchronous_fifo_init()` initialises the FIFO structure. It needs to know the number of integers that comprise a single sample, the maximum length that has been allocated for the FIFO.
- `asynchronous_fifo_exit()` uninitialises the FIFO structure.
- `asynchronous_fifo_producer_put()` puts N samples into the FIFO. It needs a timestamp that is related to when sample N-1 was obtained.
- `asynchronous_fifo_consumer_get()` gets one sample from the FIFO. It must be given a timestamp related to when this (or the previous) sample is (was) output. It returns 0 if the pulled samples are valid.

All timestamps are measured in 100 MHz ticks.

The `asynchronous_fifo_producer_put()` function returns the current rate-error observed between the producer and consumer. The rate-error is typically a number close to one, eg, 1.00001231 or 0.99995442, and for convenience the function returns epsilon, where `epsilon = rate - 1`. That is, it would return the values 0.00001231 or -0.00004558. This epsilon is represented in a signed fixed point value Q32.32. Hence, given an ideal rate the estimated rate is calculated as:

```
est_rate = ideal_rate + ((epsilon * (int64_t) ideal_rate) >> 32)
```

in 32-bit precision or for 64-bit precision:

```
est_rate = (((int64_t)ideal_rate) << 32) + epsilon * (int64_t) ideal_rate
```

Where `ideal_rate` is the expected value that would make producer and consumer match if they had no error and `epsilon` is the value returned by `asynchronous_fifo_producer_put()`. The number used for `ideal_rate` may be a PLL setting, or an ASRC ratio value. Note that the above maths can be executed in a single multiply-accumulate instruction on XCORE.

It is important to note that the `ideal_rate` is never changed; the estimated rate is a linear function combining the error and the ideal rate. Internally the Asynchronous FIFO accumulates the errors so that the epsilon returned will eventually stabilise.

The *ASRC Task* section provides an example of the integration of the FIFO with an ASRC.

6.2 Design parameters

There are three degrees of freedom in this system:

- The length of the FIFO
- The time constant of the loop filter
- The jitter characteristics of the two clocks that can be sustained.

If a long FIFO length is chosen, operation will be guaranteed but a large delay (i.e. latency) between input-signal and output-signal is introduced. If a short time constant for the loop-filter is chosen, the adjustments of the ASRC will be audible as harmonic distortion. If only small changes between the clocks is permitted, then a long time constant on the loop filter can be used with along with a short FIFO.

The value of a third parameter must match the choice of the first two; given the jitter characteristics and the time-constant the FIFO length follows. Alternatively, given the jitter characteristics and the FIFO length the maximum time constant for the loop-filter follows.

6.2.1 Practical FIFO sizing for ASRC usage

Typically for most ASRC connected systems, the hardest case for the control loop is to stabilise at startup when the peak PPM difference is first seen. This results in a FIFO depth excursion from the half full state until the control loop has zeroed the error and the FIFO level has settled back to half full. It is not typical to see a large change in PPM difference during operation of practical systems; only small drifts due to voltage and temperature changes but a system always has a startup condition which needs to be accommodated.

The FIFO size must be at least twice the peak expected perturbation to account for either a positive or negative PPM difference. Should the FIFO underflow or overflow due to insufficient depth it will reset and wait to be filled to half and attempt to close the loop again.

A typical FIFO depth plot at startup for a 500 PPM deviation is shown in Fig. 12. Note that the plot appears to be thick line because the ASRC produces on average four samples at a time whereas the FIFO is emptied one sample at a time. This “lumpiness” in the FIFO fill level means the real-time FIFO depth plot looks like a sawtooth waveform when zoomed in.

The size of the FIFO required depends on:

- The nominal output rate of the ASRC. This defines how quickly the FIFO fills. Higher rates require a larger FIFO.
- The PPM deviation from normal. This defines the maximum deviation of the nominal sample rates and the peak perturbation from half full. The PPM range of the input and output clocks must be added together. For example if a source can vary by up to +500 PPM and the sink can vary by -500 PPM then the system must account for a 1000 PPM worst-case clock rate difference.
- The input block size multiplied by the maximum upsample ratio. This defines the “lumpiness” of the real-time FIFO level and needs to be taken account of to fully buffer the block being written. This needs to be supported in both positive and negative PPM cases.

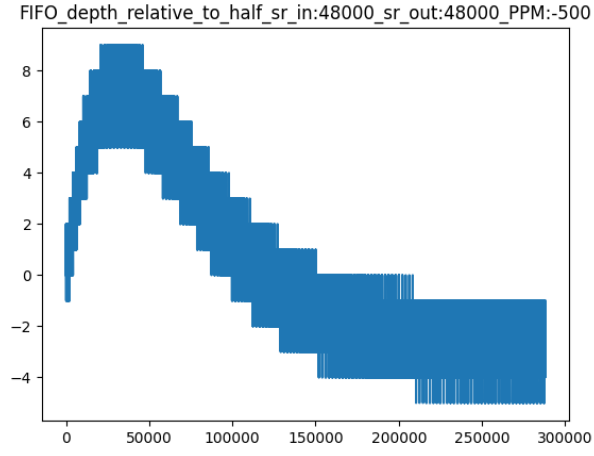


Fig. 12: Peak FIFO excursion at startup for a 500 PPM deviation at 48 kHz output rate.

Using the default constants for the loop filter (settings are conservative resulting in convergence time of around four seconds for a large step change in rate) and using the default (and minimum) input block size of four the FIFO should be sized to *at least*:

```
FIFO_LEN = (OUTPUT_RATE * PPM / 16000000) + (2 * SRC_N_IN_SAMPLES x SRC_N_OUT_IN_RATIO_MAX)
```

A sensible choice is to round up `FIFO_LEN` to the nearest multiple of 2 to ensure it is symmetrical.

A few examples follow for an ASRC input block size of four. Note that the additional latency/group delay added to the system will nominally be half of FIFO depth divided by the output rate:

Table 8: Example minimum FIFO length setting

Input Sample Rate	Output Sample Rate	Peak PPM difference	Minimum FIFO length
48000	48000	250	16
48000	48000	500	24
48000	48000	1000	38
48000	48000	2000	68
48000	96000	500	46
48000	192000	500	96
192000	48000	500	20

Note

The above settings are for the case when the timestamps are accurately measured. A time stamp relative offset between input and output values may require longer FIFO lengths since this may result in a FIFO nominal fill level away from half full.

Note

Larger input block sizes will require longer FIFO lengths. Scaling the above number by around 1.5 for a block size of eight and 3.0 for a block size of 16 will help reduce the chance of a FIFO overflow or underflow during a frequency step change.

It is recommended to test a system to the maximum PPM tolerance across all supported sample rates to verify the chosen FIFO setting, especially if the goal is to minimise the latency by reducing the FIFO size, otherwise a conservative FIFO size setting may be applied at the cost of additional latency.

6.3 Controller settings

The asynchronous FIFO includes a Proportional–integral–derivative (PID) based controller.

The PID constants can be set in two ways:

- When used with an ASRC they can be set based on input and output sample rates to a value that stabilises a 375 ppm change in approximately four seconds at 48,000 Hz.
- When used in other situations one can provide ones own Kp and Ki values. Both are represented as 32-bit integers, and a typical value for Ki is 422 (at 48 KHz, smaller for higher frequencies), and a typical value for Kp is 28,000,000 (for X kHz to X KHz; higher when the input frequency goes up, smaller when the output frequency goes up).

6.4 API

enum **asynchronous_fifo_get_return_t_**

Return code for *asynchronous_fifo_consumer_get()*

Values:

enumerator **ASYNCH_FIFO_OK**

enumerator **ASYNCH_FIFO_UNDERFLOW**

enumerator **ASYNCH_FIFO_IN_RESET**

typedef struct *asynchronous_fifo_t_* **asynchronous_fifo_t**

Data structure that holds the status of an asynchronous FIFO

typedef enum *asynchronous_fifo_get_return_t_*
asynchronous_fifo_get_return_t

Return code for *asynchronous_fifo_consumer_get()*

void **asynchronous_fifo_init**(
asynchronous_fifo_t *state, int channel_count, int max_fifo_depth,
)

Function that must be called to initialise the asynchronous FIFO. The **state** argument should be an int64_t array of **ASYNCHRONOUS_FIFO_INT64_ELEMENTS** elements that is cast to **asynchronous_fifo_t***.

That pointer should also be used for all other operations, including operations both the consumer and producer sides.

After initialising, you must initialise the PID by calling one of *asynchronous_fifo_init_PID_fs_codes()* or *asynchronous_fifo_init_PID_raw()*

Parameters

- **state** – Asynchronous FIFO to be initialised
- **channel_count** – Number of audio channels
- **max_fifo_depth** – Length of the FIFO, delay when stable will be max_fifo_depth/2

void **asynchronous_fifo_init_PID_fs_codes**(
asynchronous_fifo_t *state, int fs_input, int fs_output,
)

Function that that initialises the PID of a FIFO. Either this function or *asynchronous_fifo_init_PID_raw()* should be called. This function uses frequency codes as defined in the ASRC for a quick default setup, the raw function allows full control

Parameters

- **state** – Asynchronous FIFO to be initialised
- **fs_input** – Input FS ratio, used to pick appropriate Kp, and Ki. Must be a number less than 6.
- **fs_output** – Input FS ratio, used to pick appropriate Kp, Ki, ideal phase. Must be a number less than 6.

void **asynchronous_fifo_init_PID_raw**(
asynchronous_fifo_t *state, int Kp, int Ki, int ticks_between_samples,
)

Function that that initialises the PID of a FIFO. Either this function or *asynchronous_fifo_init_PID_raw()* should be called. This function uses frequency codes as defined in the ASRC for a quick default setup, the raw function allows full control.

This function may be called at any time by the producer in order to alter the PID and midpoint settings. It does not reset the error; one of the asynchronous_fifo_init_reset_producer() or asynchronous_fifo_init_reset_consumer() functions should be called for that.

Parameters

- **state** – Asynchronous FIFO to be initialised
- **Kp** – Proportional constant for the FIFO. This gets multiplied by the differential error measured in ticks (typically -2..2) and added to the ratio_error. A typical value is 30,000,000 - 60,000,000.
- **Ki** – Integral constant for the FIFO. This gets multiplied by the phase error measured in ticks (typically -20,000 - 20,000) and added to the ratio_error. A typical value is 200 - 300.
- **ticks_between_samples** – The number of ticks between samples is used to estimate the expected phase error halfway down the FIFO.

void **asynchronous_fifo_reset_producer**(*asynchronous_fifo_t* *state)

Function that that resets the FIFO from the producer side. Either this function should be called on the producing side, or `asynchronous_fifo_reset_consumer` should be called on the consumer side. In both cases the whole FIFO will be reset back

Parameters

- **state** – Asynchronous FIFO to be initialised

void **asynchronous_fifo_reset_consumer**(*asynchronous_fifo_t* *state)

Function that that resets the FIFO from the consumer side. Either this function should be called on the consuming side, or *asynchronous_fifo_reset_producer()* should be called on the producer side. In both cases the whole FIFO will be reset back

Parameters

- **state** – Asynchronous FIFO to be initialised

void **asynchronous_fifo_exit**(*asynchronous_fifo_t* *state)

Function that must be called to deinitalise the asynchronous FIFO

Parameters

- **state** – ASRC structure to be de-initialised

int32_t **asynchronous_fifo_producer_put**(
asynchronous_fifo_t *state, int32_t *samples, int n, int32_t timestamp,
)

Function that provides the next samples to the asynchronous FIFO.

This function and *asynchronous_fifo_consumer_get()* function both need a timestamp, which is the time that the last sample was input (this function) or output (*asynchronous_fifo_consumer_get()*). The asynchronous FIFO will hand the samples across from producer to consumer through an elastic queue, and run a PID algorithm to calculate the best way to equalise the input clock relative to the output clock. Therefore, the timestamps have to be measured on either the same clock or two very similar clocks. It is probably fine to use the reference clocks on two tiles, provided the tiles came out of reset at more or less the same time. Using the clocks from two different chips would require the two chips to share an oscillator, and for them to come out of reset simultaneously.

The output is filtered and should be applied directly as a correction factor eg, multiplied into an ASRC ratio, or multiplied into a PLL timing.

Parameters

- **state** – ASRC structure to push the sample into
- **samples** – The sample values.
- **n** – The number of samples
- **timestamp** – The number of ticks when this sample was input.

Returns

The current estimate of the mismatch of input and output frequencies. This is represented as a 32-bit signed number. Zero means no mismatch, a value less than zero means that the producer is faster than the consumer, a value greater than zero means that the producer is slower than the consumer. The value should be

scaled by 2**-32. That is, the current best approximation for consumer_speed/producer_speed is 1 + (return_value * 2**-32)

asynchronous_fifo_get_return_t **asynchronous_fifo_consumer_get**(
asynchronous_fifo_t *state, int32_t *samples, int32_t timestamp,
)

Function that gets an output sample from the asynchronous FIFO

Function that implements the consumer interface. Control communication happens through two variables: reset and sample_data_valid. These shall only be set as the last action, as they signify to the production side that the datastructure can now be read on that side.

Note that the samples are filled in regardless of whether the FIFO is operating or not; the consumer will repeatedly get the same sample if the producer fails. The producer side is reset exactly once on reset. If this is a problem then please use the return flag (0 = OK) to handle.

Parameters

- **state** – ASRC structure to read a sample out off.
- **samples** – The array where the frame with output samples will be stored.
- **timestamp** – A timestamp taken at the time that the last sample was output. See `asynchronous_fifo_produce` for requirements.

Returns

The FIFO status and whether the samples are valid or not

ASYNCHRONOUS_FIFO_INT64_ELEMENTS(N, C)

macro that calculates the number of int64_t to be allocated for the fifo for a FIFO of N elements and C channels

struct **asynchronous_fifo_t_**

#include <asynchronous_fifo.h> Data structure that holds the status of an asynchronous FIFO

int **asrc_timestamp_interpolation**(
int timestamp, asrc_ctrl_t *asrc_ctrl, int ideal_freq,
)

Function that interpolates a timestamp for a sample generated by the ASRC. Given a measured timestamp for the sample going into the ASRC, the asrc control structure, and the expected output frequency, this function returns a timestamp for when the last sample was produced by the ASRC.

Parameters

- **timestamp** – Value of the reference clock taken when the last sample fed into the ASRC was sampled.
- **asrc_ctrl** – ASRC control block
- **ideal_freq** – Expected base frequency to which the ASRC is operating; eg, 48000 or 44100

6.5 Implementation detail

This section details the inner workings of the FIFO and is intended only for advanced users who wish to understand the operation in more detail.

6.5.1 Measurements for the PID

The asynchronous FIFO uses the phase difference as the input for a PID controller. The phase difference is shown in *Measurement of the phase difference*. It is defined as the time difference between a sample when it entered the queue and left the queue. Unlike traditional phase differences that are measured in radians and where the maximum phase difference is *+/- pi* , the phase difference is measured as a time difference, and thereby allow the phase to be off by more than half a sample.

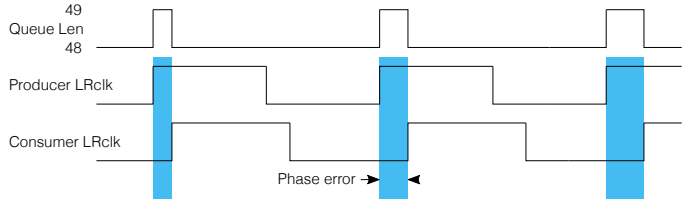


Fig. 13: Measurement of the phase difference

In a stable situation, it is desirable that the FIFO is half-full, it follows that the desired phase difference is half the maximum length of the FIFO multiplied by the sample rate. For example, for a FIFO of 10 elements the ideal fill level is 5, and at 48 kHz the ideal phase error is 5 x 2.0833 us = 10.4166 us. If the output is running slightly too fast then sample X will enter the FIFO just after X-N/2 leaves the FIFO; if the output is running slightly too slow than sample X will enter the FIFO just before X-N/2 leaves the FIFO.

The *phase-error* is defined as the difference between the ideal phase-difference and the measured phase difference. Say that the queue has filled up badly and stores 9 items, then the phase difference will account for the 4 extra items in the FIFO, causing a phase difference 18.75 us rather than the desired 10.4166 us, producing a phase error of between 8.33 us. The phase difference is notionally a continuous value (a time stamp) in practice it is measured with the reference clock which has a 10 ns granularity. However, that is of far higher granularity than whole samples (2083 times better at a 48 KHz sample rate)

It is worth noting that the phase difference itself is an integral value; it is the number of samples since the beginning of time that the ASRC is out by. The goal of the rate converter is to make the phase difference stable (ie, it does not move between subsequent samples), and zero (ie, the FIFO is exactly mid level). Hence, the differential of the phase error can be seen as a proportional error, and the phase error itself as an integral error.

6.5.2 Implementation of asynchronicity

The FIFO straddles two threads; this is essential as the two threads operate on different rates. Hence, the FIFO is a shared-memory element between those two threads. A read-pointer (managed by the consuming thread) and a write-pointer (managed by the producing thread) are maintained independently. The read-pointer and write-pointer are normally N/2 elements apart.

During normal operation the Incoming and outgoing traffic are rate-matched, and the read-pointer and write-pointer will be on opposite ends of the circular buffer.

There are three situations where operation may be abnormal:

- Where the consumer is no longer consuming samples
- Where the producer is no longer producing samples

- Where a larger than expected change in the sample rates has caused the loop filter to require more than N/2 spaces away from the mid-point.

Detecting these cases requires the calculation of the modulo difference between the write-pointer and read-pointer; if that difference is close to zero the FIFO is about to underflow; if it is close to N the FIFO is about to overflow. The notion "close to" is used since the read- and write-pointer are updated independently by different threads, so the pointer may be one less than anticipated, and an update may be missed (i.e. a race condition). Underflow is detected by the thread on the output side, overflow is detected by the thread on the input side. Differentiating overflow/underflow from too large a change in the sample rate may be hard and not necessary if they are all treated in the same way.

The employed method is to use two flags; `RESET` and `DO_NOT_PRODUCE` that are owned by the consumer and producer sides respectively.

- The `RESET` flag is set by the consumer if it spots an underflow condition. Once `RESET` is set, the consumer will no longer advance the FIFO, return the same sample on each call, and wait for `RESET` to clear. Only the consumer may set `RESET`, only the producer may clear `RESET`.
- The `DO_NOT_PRODUCE` flag is set by the producer if it spots on overflow condition. Once set, the producer will no longer advance the FIFO, and wait for the consumer to set the `RESET` flag once it has come to an underflow (which must happen as the producer has stopped producing), at which point a third action is met.
- If the producer spots `RESET` being high, it resets the FIFO state except for the read-pointer; it leaves that as it is maintained by the consumer. Instead, it sets the write pointer to be at the other side of the buffer. Once the state is reset it will clear `DO_NOT_PRODUCE` and finally `RESET`, whereupon all should start running again.

6.5.3 Communication and reset protocol summary

In the thread on the producer side a `put()` operation performs the following:

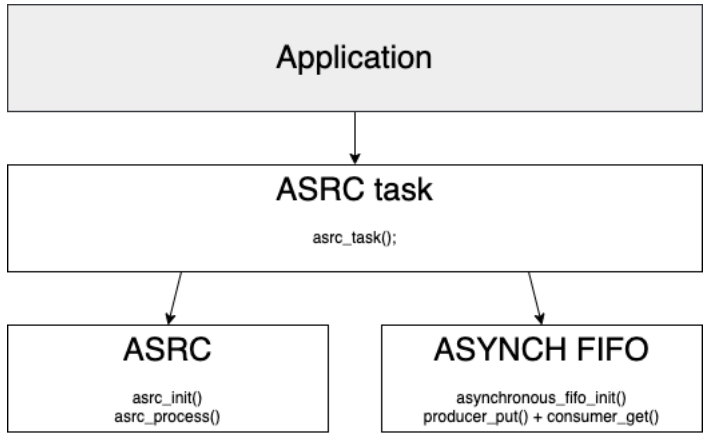
- If the `RESET` flag is set:
  1. Set the write-pointer to half-way from the read-pointer
  2. Set `fs_ratio` to 1
  3. Clear the phase error and reset all other PID state.
  4. Clear the `DO_NOT_PRODUCE` flag
  5. Clear the `RESET` flag (this is the last step, unlocking the consumer when it is safe to do so)
- else if there is no room left in the FIFO to store all samples:
  1. Set the `DO_NOT_PRODUCE` flag
- else if the `DO_NOT_PRODUCE` flag is not set:
  1. Copy `N` frames into the FIFO
  2. Increase the write-pointer
  3. Obtain a timestamp that was queued by the consumer
  4. Calculate the phase-error and the difference with the previous phase error
  5. Update the PID using the difference as the proportional error and the phase-error as the integral error.

In the thread on the consumer side a `get()` operation performs the following:

- Copy the sample at the read-pointer into the buffer provided by the consumer
- If the RESET flag is clear and there is at least one sample in the FIFO:
  1. Record the timestamp in the time-stamp queue
  2. Increase the read-pointer.
- else if the RESET flag is clear:
  1. Set the RESET flag.

7 ASRC Task

The ASRC library provides a function call that operates on blocks of samples whereas typical *XMOS* audio IO libraries provide streaming audio one sample at a time. The ASRC task wraps up the core ASRC function with all of the other lower level APIs (eg. FIFO) and required sample change and initialisation logic. It provides a simple-to-use and generic ASRC conversion block suitable for integration into practical designs. It is fully re-entrant permitting multiple instances within a project supporting multiple (or bi-directional) sample rates and audio clock domain bridges.



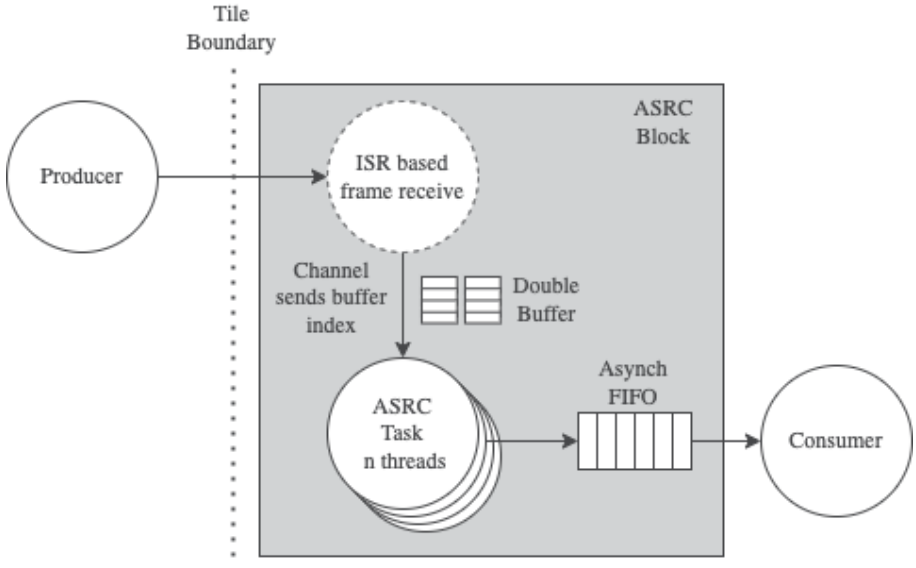
7.1 Operation

The ASRC task handles bridging between two asynchronous audio sources. It has an input side and output side. The input samples are provided over a channel allowing the source to be placed on a different *xcore* tile if needed. The output side sample interface is via an asynchronous FIFO meaning the consumer must reside on the same *xcore* tile as the ASRC. The ASRC task uses a minimum of one thread but may be configured to use many depending on processing requirements.

Both input and output interfaces must specify the nominal sample rate required and additionally the input must specify a channel count. The output channel count will be set to the same as the input channel count automatically once the ASRC has automatically configured itself. A timestamp indicating the time of the production of the last input sample and the consumption of the first output sample must also be supplied which allows the ASRC FIFO to calculate the rate and phase difference. Each time either the input or output nominal sample rate or the channel count changes the ASRC subsystem automatically re-configures itself and restarts with the new settings.

The ASRC Task supports the following nominal sample rates for input and output:

- 44.1 kHz
- 48 kHz
- 88.2 kHz
- 96 kHz
- 176.4 kHz
- 192 kHz



Because the required compute for multi-channel systems may exceed the performance limit of a single thread, the ASRC subsystem is able to make use of multiple threads in parallel to achieve the required conversion within the sample time period. It uses a dynamic fork and join architecture to share the ASRC workload across multiple threads each time a batch of samples is processed. The threads must all reside on the same tile as the ASRC task due to them sharing input and output buffers. The workload and buffer partitioning is dynamically computed by the ASRC task at stream startup and is constrained by the user at compile time to set maximum limits of both channel count and worker threads.

The number of threads that are required depends on the required channel count and sample rates required. Higher sample rates require more MIPS. The amount of thread MHz (and consequently how many threads) required can be *roughly* calculated using the following formulae:

- Total thread MHz required for *xcore.ai* systems = 0.15 * Max channel count * (Max SR input kHz + Max SR output kHz)
- Total thread MHz required for *xcore-200* systems = 0.3 * Max channel count * (Max SR input kHz + Max SR output kHz)

The difference between the performance requirement between the two architectures is due to xcore.ai supporting a Vector Processing Unit (VPU) which allows acceleration of the internal filters used by the ASRC. For example:

- A two channel system supporting up to 192kHz input and output will require about (0.15 * (192 + 192) * 2) ~= 115 thread MHz. This means a single thread (assuming no more than 5 active threads on an *xcore.ai* device with a 600MHz clock) will likely be capable of handling this stream.
- An eight channel system consisting of either 44.1kHz or 48kHz input with maximum output rate of 192kHz will require about (0.15 * (48 + 192) * 8) ~= 288 thread MHz. This can adequately be provided by four threads (assuming up to 8 active threads on an *xcore.ai* device with a 600MHz clock).

In reality the amount of thread MHz needed will be lower than the above formulae suggest since subsequent ASRC channels after the first can share some of the calculations. This results in about at 10% performance requirement reduction per additional channel per worker thread. Increasing the input frame size in the ASRC task may also reduce the MHz requirement a few % at the cost of larger buffers and a slight latency increase.

Warning

Exceeding the processing time available by specifying a channel count, input/output rates, number of worker threads or device clock speed may result in at best choppy audio or a blocked ASRC task if the overrun is persistent.

It is strongly recommended that the system is tested for the desired channel count and input and output sample rates. An optional timing calculation and check is provided in the ASRC to allow characterisation at run-time which can be found in the *asrc_task.c* source code.

The low level ASRC processing function call API accepts a minimum input frame size of four whereas most XMOS audio interfaces provide a single sample period frame. The ASRC subsystem integrates a serial to block back to serial conversion to support this. The input side works by stealing cycles from the ASRC using an interrupt and notifies the main ASRC loop using a single channel end when a complete frame of double buffered is available to process. The ASRC output side is handled by the asynchronous FIFO which supports a block *put* with single sample *get* and thus provides de-serialisation intrinsically.

7.2 Latency characterisation

The latency shown by ASRC Task depends on many factors:

- Input sample rate (dynamically variable)
- Output sample rate (dynamically variable)
- ASRC filter stages latency (fixed based on input and output sample rates)
- FIFO sizing (statically or dynamically variable by user)
- ASRC sample processing block size (default of 4 which is the minimum for the ASRC and recommended for most applications)

The input and output sample rate are defined by the application and are not negotiable. The ASRC filters have fixed group delay according to the input and output rates. The underlying filter delay can be found in *ASRC latency characterisation section* and typically dominates the total delay.

The ASRC sample block processing size is nominally 4 but can be increased to 8, 16 or 32 to slightly reduce the MIPS required to run the processing but will incur extra delay. The case for the block size of 4 is already accounted for in the ASRC filter stage figures.

FIFO sizing is the major variable which the user has control over. The FIFO size is configurable and is a trade-off between PPM lock range required, output sample rate and desired latency. It is also partly governed by the maximum upsample ratio since, when upsampling, multiple samples are produced for a single input sample and hence the FIFO needs to be larger to accommodate a whole block.

See the *Practical FIFO sizing* section for more details.

The total delay can be calculated as follows:

```
GROUP_DELAY = ((INPUT_BLOCK_SIZE - 4) / INPUT_SAMPLE_RATE) + ASRC_FILTER_DELAY + (OUTPUT_FIFO_LENGTH / OUTPUT_
↪SAMPLE_RATE / 2)
```

7.3 API & usage

The ASRC Task consists of a task to which various data structures must be declared and passed:

- A pointer to instance of the *asrc_in_out_t* structure which contains buffers, stream information and ASRC task state.
- A pointer to the FIFO used at the output side of the ASRC task.
- The length of the FIFO passed in above.

In addition the following two functions may be declared in a user *C* file (note that *XC* does not handle function pointers):

- The callback function from ASRC task which receives samples over a channel from the producer.
- A callback initialisation function which registers the callback function into the *asrc_in_out_t* struct

If these are not defined, then a default receive implementation will be used which is matched with the **`send_asrc_input_samples_default()`** function on the user's producer side. This should be sufficient for typical usage.

> **Note**
>
> **`ASRC task`** must have **`asrc_task_config.h`** defined in the user application which sets various static settings for the ASRC. See *ASRC task API* for details or reference AN02003: SPDIF/ADAT/I2S Receive to I2S Slave Bridge with ASRC as an example.

An example of calling the ASRC task form and **XC** main function is provided below. Note use of *unsafe* permitting the compiler to allow shared memory structures to be accessed by more than one thread:

```
chan c_producer;

// FIFO and ASRC I/O declaration. Unsafe to allow producer and consumer to access it from XC
#define FIFO_LENGTH     40 // Example only. Depends on rates and PPM - see docs
int64_t array[ASYNCHRONOUS_FIFO_INT64_ELEMENTS(FIFO_LENGTH, MAX_ASRC_CHANNELS_TOTAL)];

unsafe{
    // IO struct for ASRC must be passed to both asrc_proc and consumer
    asrc_in_out_t asrc_io = {{{0}}};
    asrc_in_out_t * unsafe asrc_io_ptr = &asrc_io;
    asynchronous_fifo_t * unsafe fifo = (asynchronous_fifo_t *)array;
    setup_asrc_io_custom_callback(asrc_io_ptr); // Optional user rx function

    par
    {
        producer(c_producer);
        asrc_task(c_producer, asrc_io_ptr, fifo, FIFO_LENGTH);
        consumer(asrc_io_ptr, fifo);

    }
} // unsafe region
```

An example of the user-defined *C* function for receiving the input samples is shown below along with the user callback registration function. The *receive_asrc_input_samples()* function must be as short as possible because it steals cycles from the ASRC task operation. Because this function is not called until the first channel word is received from the producer, the *chanend_in_word()* operations will happen straight away and not block as long as the producer immediately produces all required samples.

```
// Default implementation of receive (called from ASRC) which receives samples and config over a channel. This
↪is overridable.
ASRC_TASK_ISR_CALLBACK_ATTR
unsigned receive_asrc_input_samples_cb_default(chanend_t c_asrc_input, asrc_in_out_t *asrc_io, unsigned *new_
↪input_rate){
    static unsigned asrc_in_counter = 0;

    // Get format and timing data from channel
    *new_input_rate = chanend_in_word(c_asrc_input);
    asrc_io->input_timestamp[asrc_io->input_write_idx] = chanend_in_word(c_asrc_input);
    asrc_io->input_channel_count = chanend_in_word(c_asrc_input);

    // Pack into array properly LRLRLRLR for 2ch or 123412341234 for 4ch etc.
    for(int i = 0; i < asrc_io->input_channel_count; i++){
        int idx = i + asrc_io->input_channel_count * asrc_in_counter;
        asrc_io->input_samples[asrc_io->input_write_idx][idx] = chanend_in_word(c_asrc_input);
    }

    if(++asrc_in_counter == SRC_N_IN_SAMPLES){
        asrc_in_counter = 0;
    }

    return asrc_in_counter;
}
// END ASRC_TASK_ISR_CALLBACK_ATTR
```

Note that the producing side of the above transaction must match the channel protocol. For this example, the producer must send the following items across the channel in order:

- The nominal input sample rate.
- The input time stamp of the last sample received.
- The input channel count of the current frame.
- The samples from 0..n.

Because a *streaming* channel is used the back-pressure on the producer side will be very low because the channel outputs will be buffered and the receive callback will always respond to the received words.

This callback function helps bridge between *sample based* systems and the block-based nature of the underlying ASRC functions without consuming an extra thread.

The API for ASRC task is shown below:

typedef unsigned (***asrc_task_produce_isr_cb_t***)(chanend_t c_asrc_input, *asrc_in_out_t* *asrc_io, unsigned *new_input_rate)

Type definition of the callback function if a user version is required. May only be used from "C" (XC does not support function pointers).

void **asrc_task**(
chanend c_asrc_input, *asrc_in_out_t* *asrc_io, *asynchronous_fifo_t* *fifo, unsigned fifo_length,
)

Main ASRC processor task. Runs forever waiting on new samples from the producer. Spawns up to MAX_ASRC_THREADS during ASRC processing.

Parameters

- **c_asrc_input** – The channel end used to connect the producer to the ASRC task.
- **asrc_io** – A pointer to the structure used for holding ASRC IO and state.
- **fifo** – A pointer to the FIFO used for outputting samples from the ASRC task to the consumer.
- **fifo_length** – The length (depth) of the output FIFO. This is multiplied by channel count internally.

int **pull_samples**(
asrc_in_out_t *asrc_io, *asynchronous_fifo_t* *fifo, int32_t *samples, uint32_t output_frequency, int32_t consume_timestamp,
)

Helper function called by consumer to provide ASRC output samples. Samples are populated in the *samples array and the user must provide the current nominal output frequency and a timestamp of when the last samples were consumed from the 100 MHz ref clock

Parameters

- **asrc_io** – A pointer to the structure used for holding ASRC IO and state.
- **fifo** – A pointer to the FIFO used for outputting samples from the ASRC task to the consumer.
- **samples** – A pointer to a whole output frame (all channels in a single sample period) to populate.
- **output_frequency** – The nominal output frequency. Used for detecting a sample rate change.
- **consume_timestamp** – The timestamp of the first consumed sample.

void **reset_asrc_fifo_consumer**(*asynchronous_fifo_t* *fifo)

Helper function called by consumer to reset the FIFO. Resets to half full and clears the contents to zero.

Parameters

- **fifo** – A pointer to the FIFO used for outputting samples from the ASRC task to the consumer.

void **init_asrc_io_callback**(
asrc_in_out_t *asrc_io, *asrc_task_produce_isr_cb_t* asrc_rx_fp,
)

Prototype that can optionally be defined by the user to initialise the function pointer for the ASRC receive produced samples ISR. If this is not called then receive_asrc_input_samples_cb_default() is used and the you may call *send_asrc_input_samples_default()* from the application to send samples to the ASRC task.

Must be called before running *asrc_task()*

Parameters

- **asrc_io** – A pointer to the structure used for holding ASRC IO and state.
- **asrc_rx_fp** – A pointer to the user asrc_receive_samples function. NOTE - This MUST be decorated by ASRC_TASK_ISR_CALLBACK_ATTR to allow proper

stack calculation by the compiler. See receive_asrc_input_samples_cb_default() in asrc_task.c for an example of how to do this.

void **send_asrc_input_samples_default**(
chanend c_asrc_input, unsigned input_frequency, int32_t input_timestamp, unsigned input_channel_count, int32_t *input_samples,
)

If the *init_asrc_io_callback()* function is not called then a default implementation of the ASRC receive will be used. This send function (called by the user producer side) mirrors the receive and can be used to push samples into the ASRC.

Parameters

- **c_asrc_input** – The chan end on the application producer side connecting to the ASRC task.
- **input_frequency** – The sample rate of the input stream (44100, 48000, ...).
- **input_timestamp** – The ref clock timestamp of latest received input sample.
- **input_channel_count** – The number of input audio channels (1, 2, 3 ...).
- **input_samples** – A pointer to the input samples array (channel 0, 1, ...).

ASRC_TASK_ISR_CALLBACK_ATTR

Decorator for user's ASRC producer receive callback. Must be used to allow stack usage calculation.

MAX_ASRC_CHANNELS_TOTAL

Maximum number of audio channels in total. Used for buffer sizing and FIFO sizing (statically defined).

MAX_ASRC_THREADS

Maximum number of threads to be spawned by ASRC task. Used for buffer sizing and FIFO sizing (statically defined).

SRC_N_IN_SAMPLES

Block size of input to the low level asrc_process function. Must be a power of 2 and minimum value is 4, maximum is 16. Used for buffer sizing and FIFO sizing (statically defined).

SRC_N_OUT_IN_RATIO_MAX

Max ratio between samples out:in per processing step (44.1->192 is worst case). Used for buffer sizing and FIFO sizing (statically defined).

SRC_DITHER_SETTING

Enables or disables quantisation of output with dithering to 24b.

struct **asrc_in_out_t**

#include <asrc_task.h>

8 Performance characterisation

The FFT plots in this section provide a visual guide to the performance of the SSRC, ASRC, DS3 and OS3 sample rate converters. Test signals were created allowing analysis of the sample rate converter output across different scenarios.

Two input signals were played through a stereo sample rate converter across a range of input and output sample rates. For Channel 0, a single pure tone was generated ensuring its frequency was well within the overall nyquist rate. For Channel 1, multiple tones spaced logarithmically were generated with the spacing most dense at higher frequencies.

The resulting frequency plot output clearly shows the noise floor relative to the sample rate converted injected tone(s). The plots are annotated with an estimate of the Signal to Noise Ratio (SNR) as well as Total Harmonic Distortion (THD).

For the case of the ASRC, in addition to the nominal input frequency of 0 PPM deviation, the +/-100 PPM frequency deviation cases are also shown.

8.1 Pure Tone FFT SRC plots across sample rate combinations

8.1.1 Frequency error: 0.999900Hz

Output Fs : 16,000Hz

- No SRC available for this scenario.

Output Fs : 32,000Hz

- No SRC available for this scenario.

Output Fs : 44,100Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

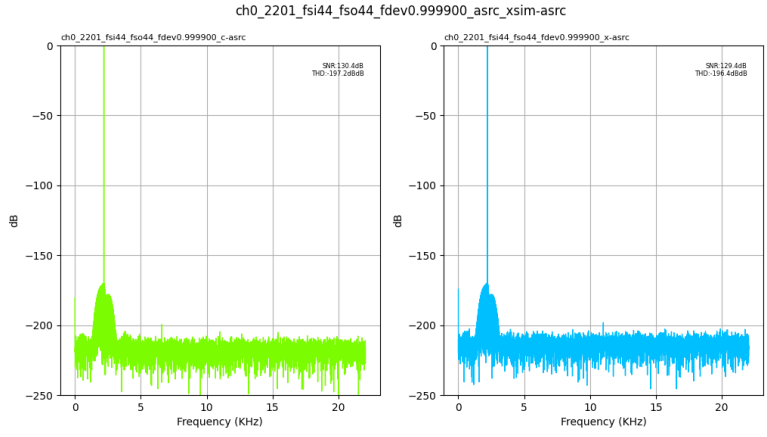


Fig. 14: Input Fs: 44,100Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

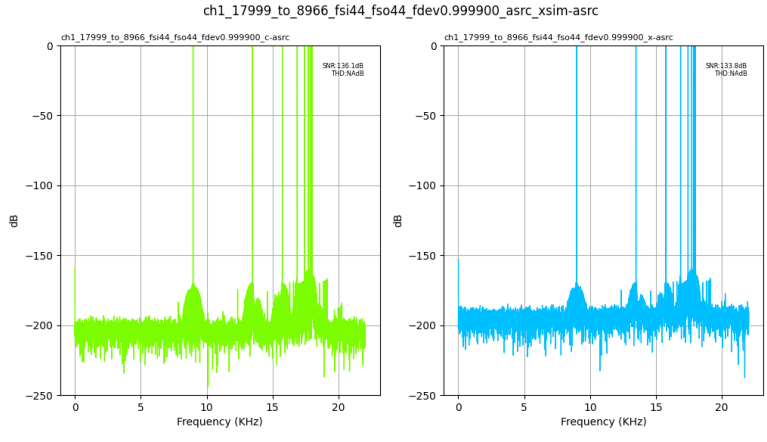


Fig. 15: Input Fs: 44,100Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

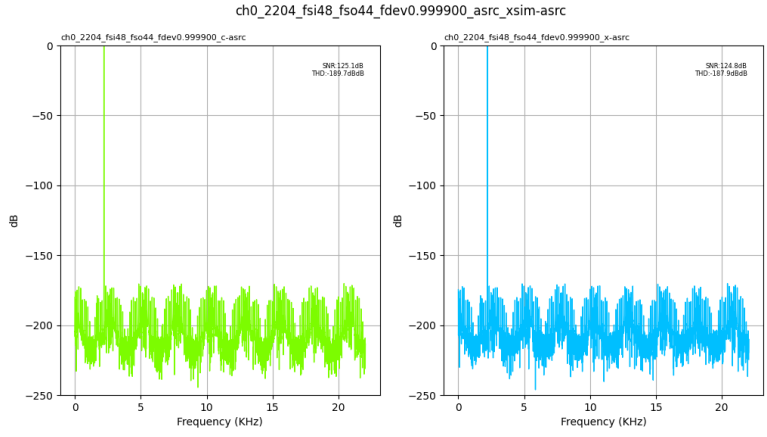


Fig. 16: Input Fs: 48,000Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

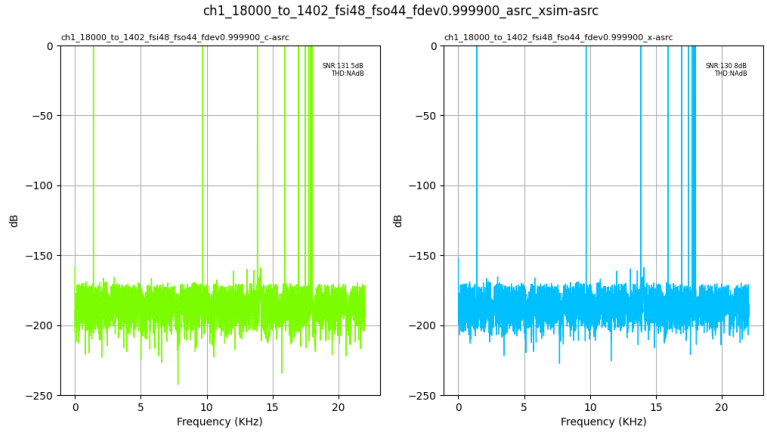


Fig. 17: Input Fs: 48,000Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

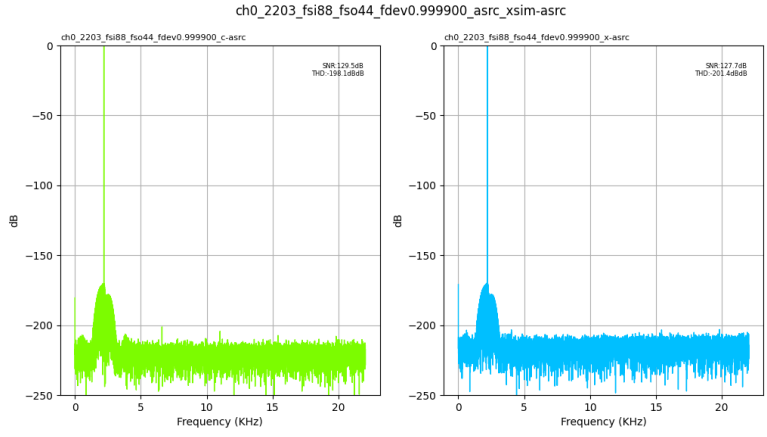


Fig. 18: Input Fs: 88,200Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

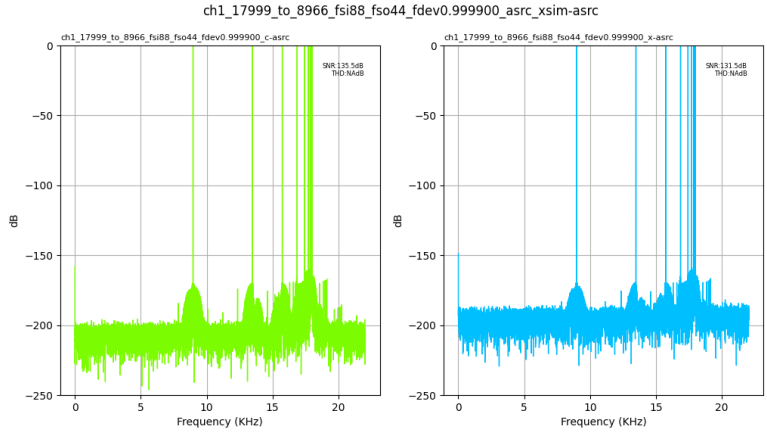


Fig. 19: Input Fs: 88,200Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

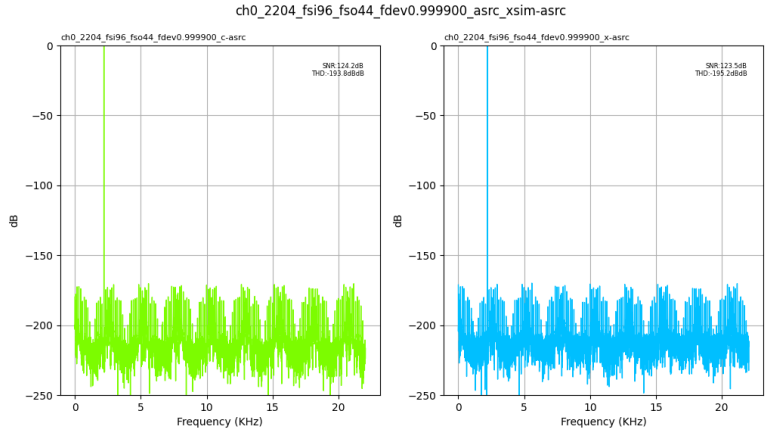


Fig. 20: Input Fs: 96,000Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

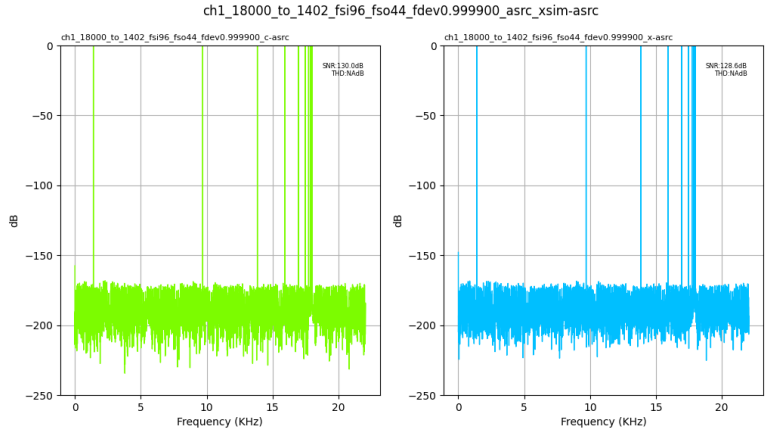


Fig. 21: Input Fs: 96,000Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

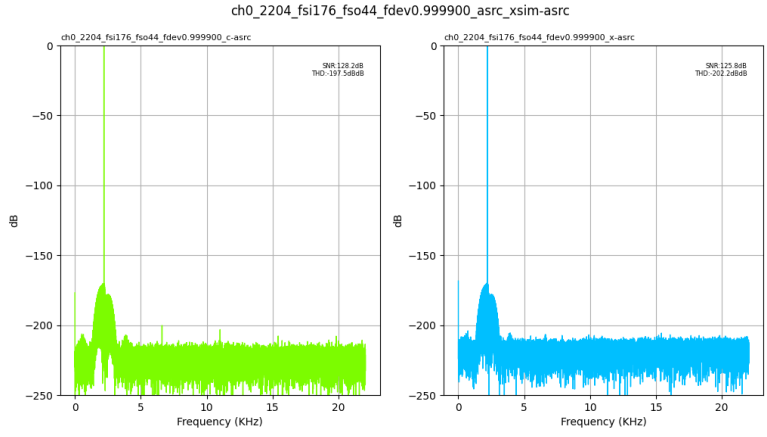


Fig. 22: Input Fs: 176,400Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

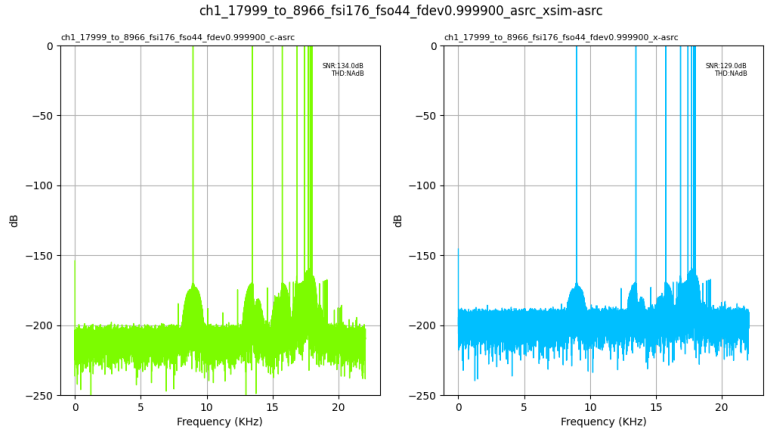


Fig. 23: Input Fs: 176,400Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

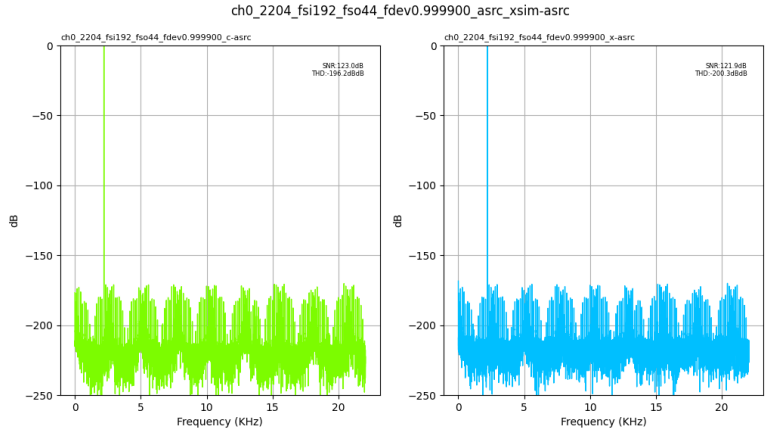


Fig. 24: Input Fs: 192,000Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

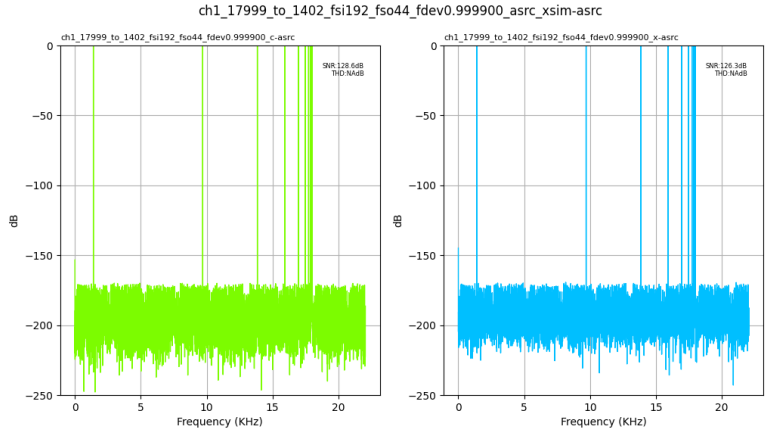


Fig. 25: Input Fs: 192,000Hz, Output Fs: 44,100Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

Output Fs : 48,000Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

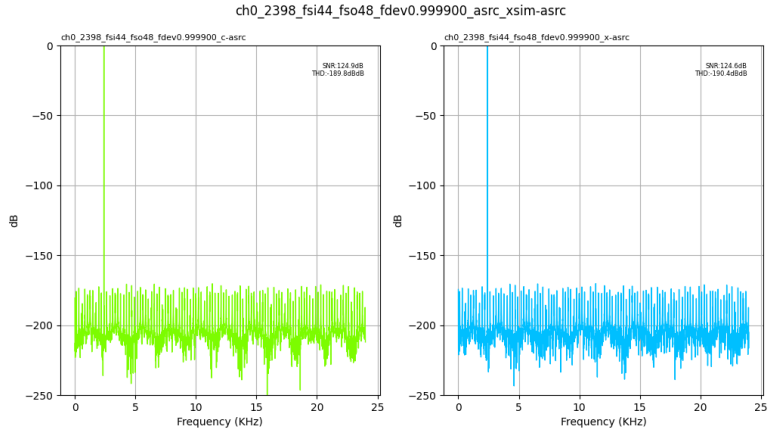


Fig. 26: Input Fs: 44,100Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

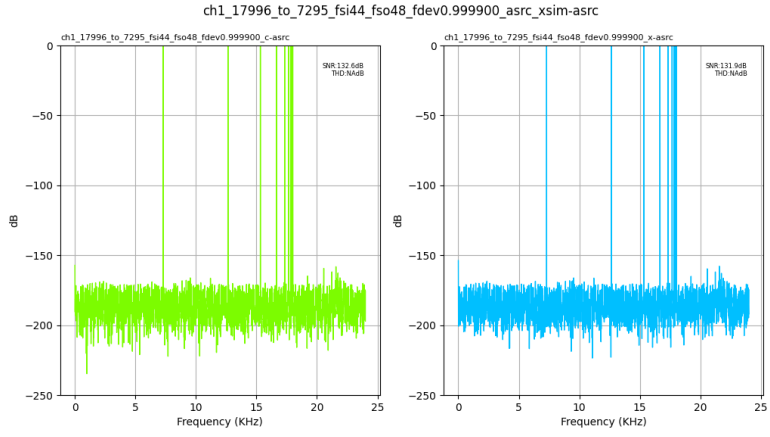


Fig. 27: Input Fs: 44,100Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

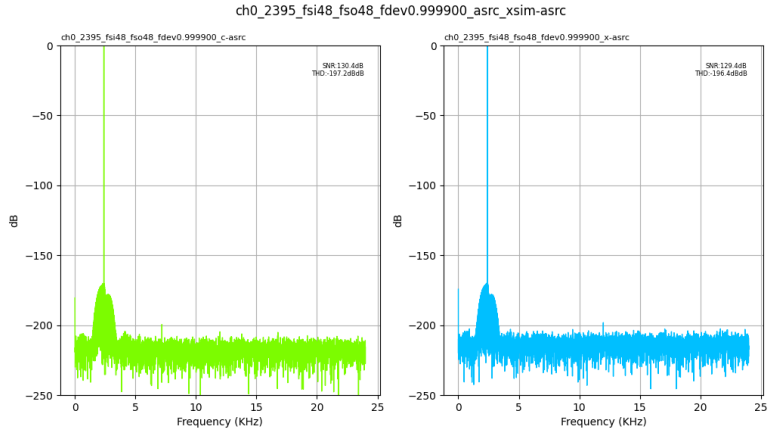


Fig. 28: Input Fs: 48,000Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

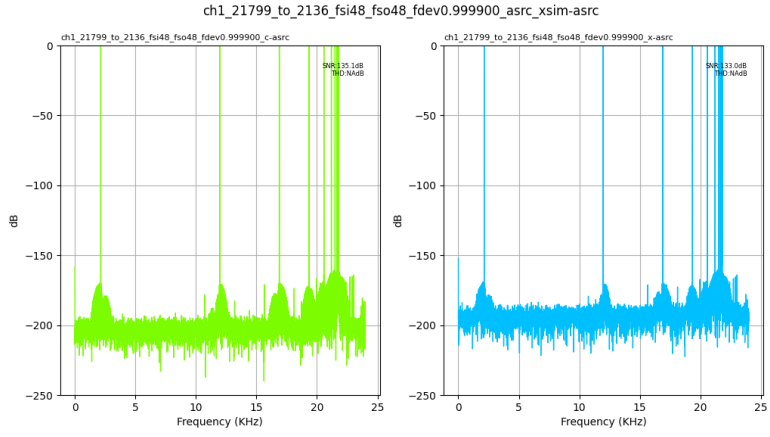


Fig. 29: Input Fs: 48,000Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

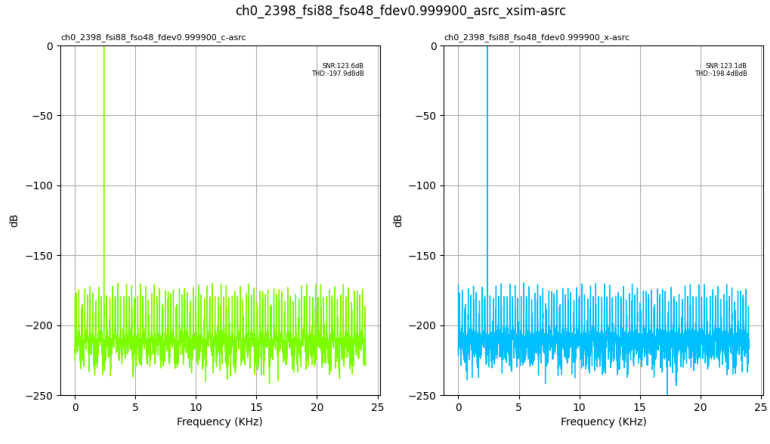


Fig. 30: Input Fs: 88,200Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

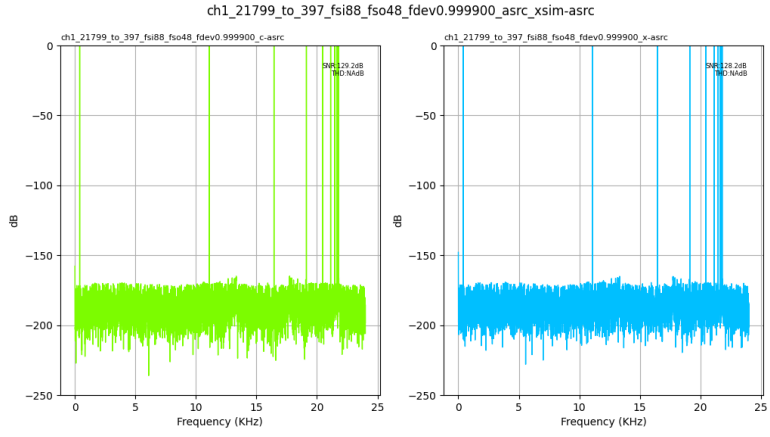


Fig. 31: Input Fs: 88,200Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

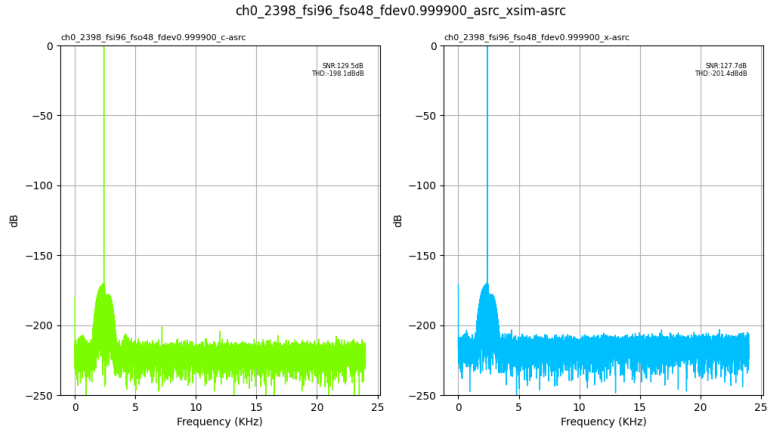


Fig. 32: Input Fs: 96,000Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

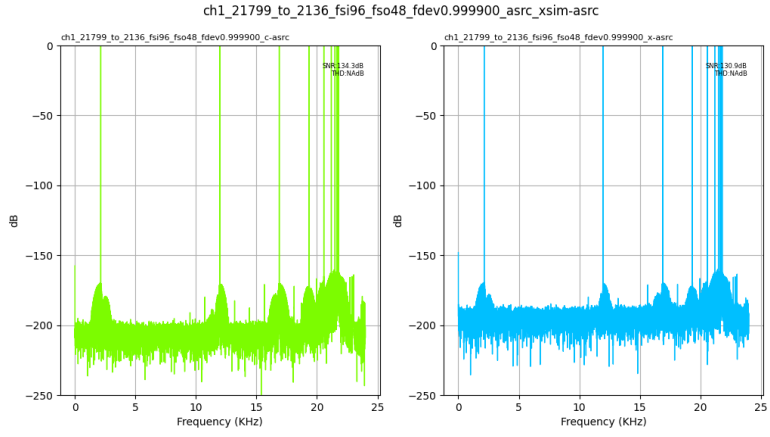


Fig. 33: Input Fs: 96,000Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

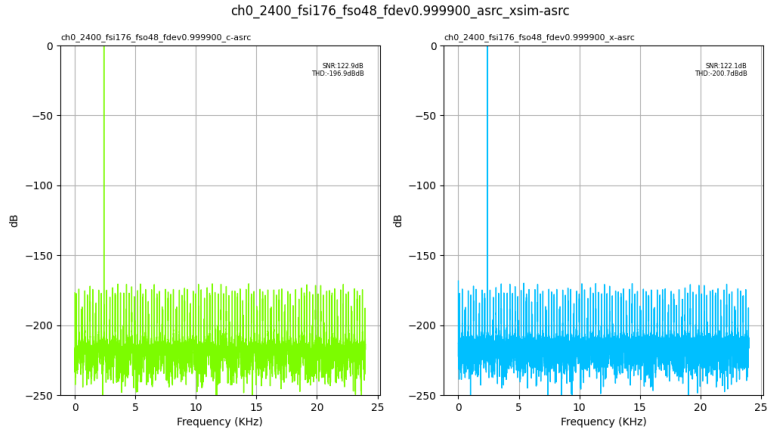


Fig. 34: Input Fs: 176,400Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

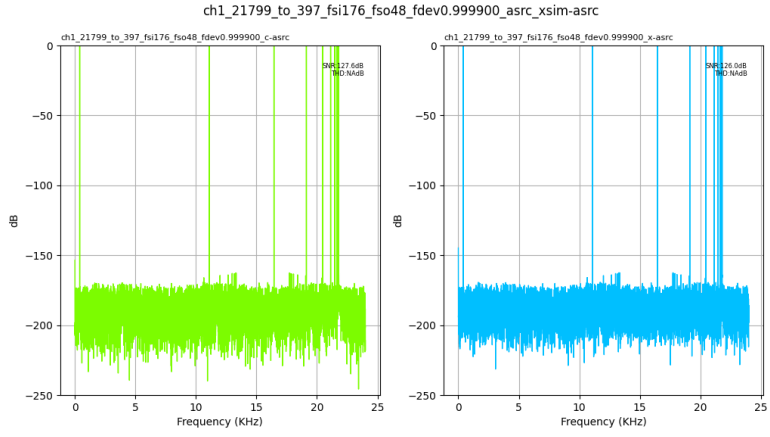


Fig. 35: Input Fs: 176,400Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

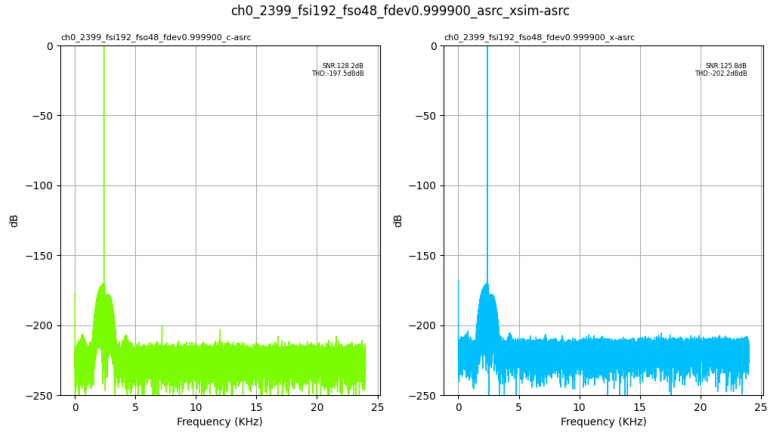


Fig. 36: Input Fs: 192,000Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

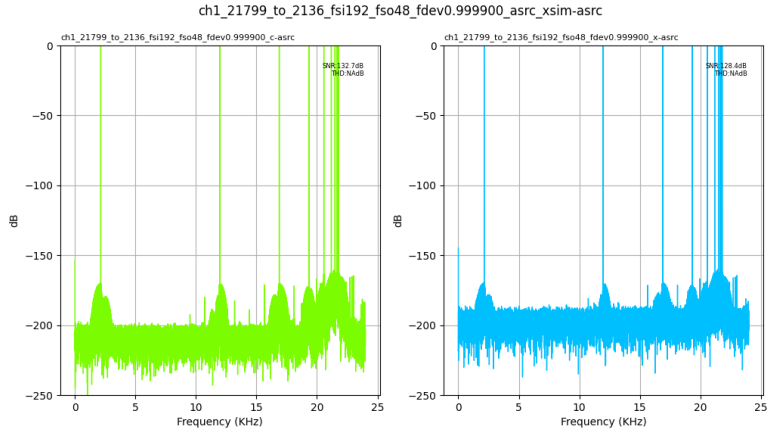


Fig. 37: Input Fs: 192,000Hz, Output Fs: 48,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

Output Fs : 88,200Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

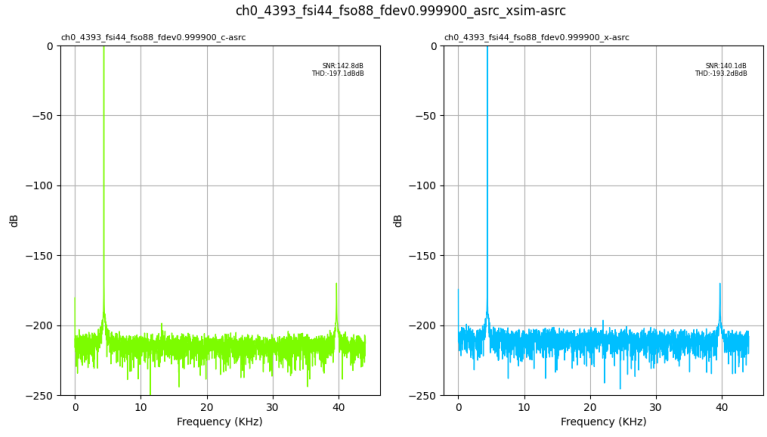


Fig. 38: Input Fs: 44,100Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

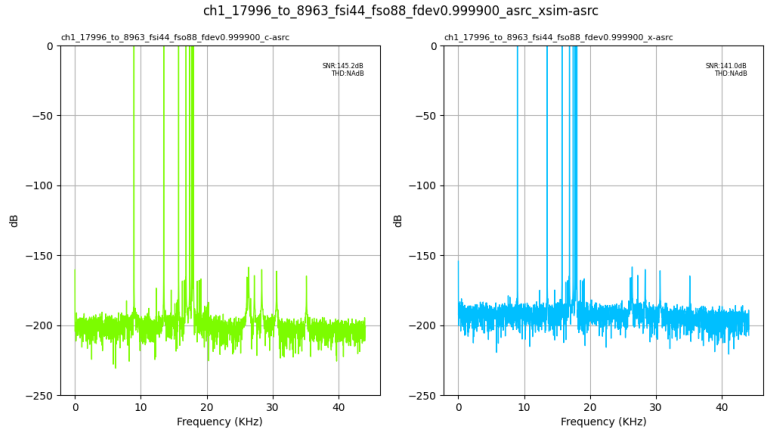


Fig. 39: Input Fs: 44,100Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

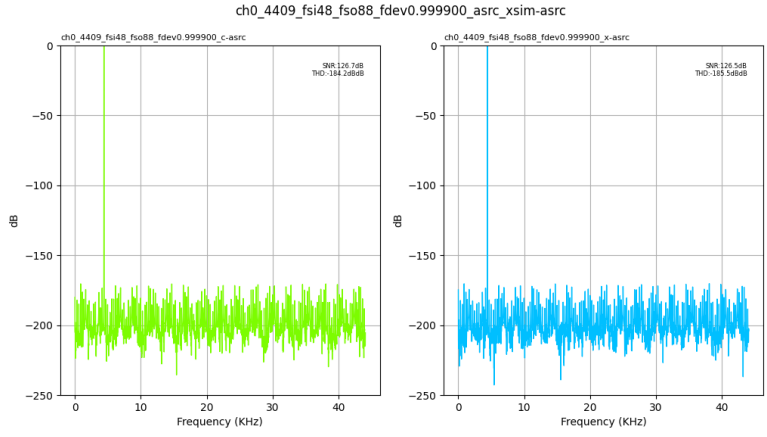


Fig. 40: Input Fs: 48,000Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

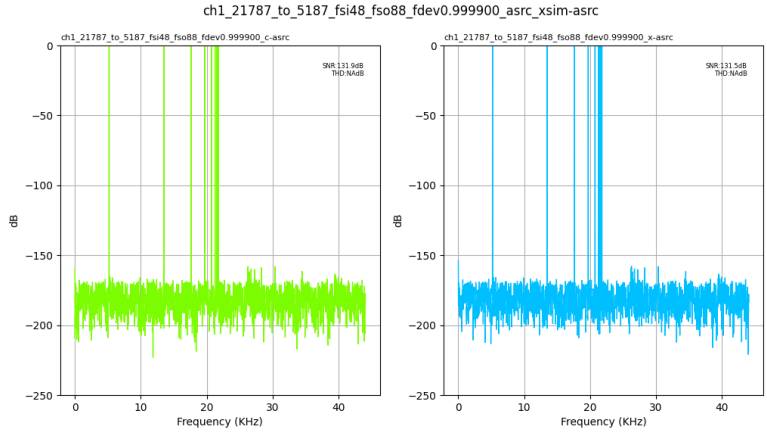


Fig. 41: Input Fs: 48,000Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

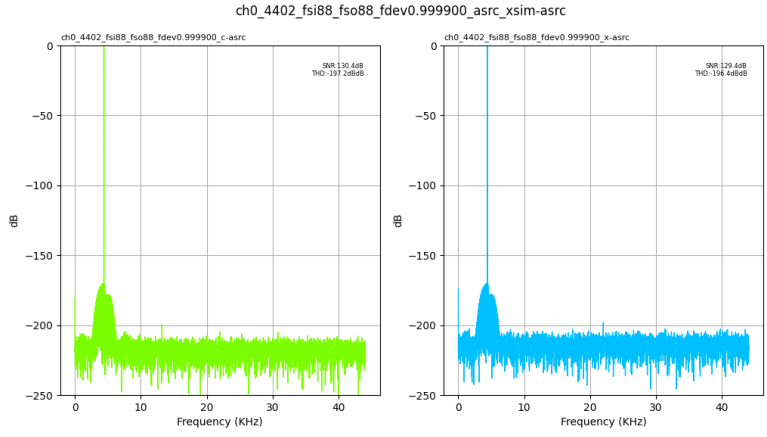


Fig. 42: Input Fs: 88,200Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

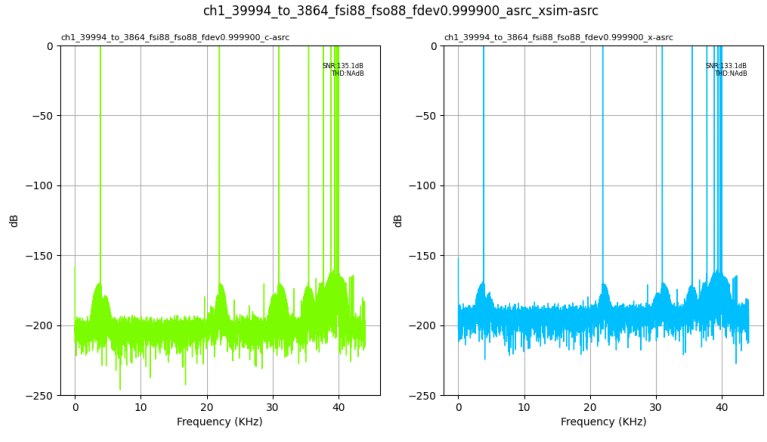


Fig. 43: Input Fs: 88,200Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

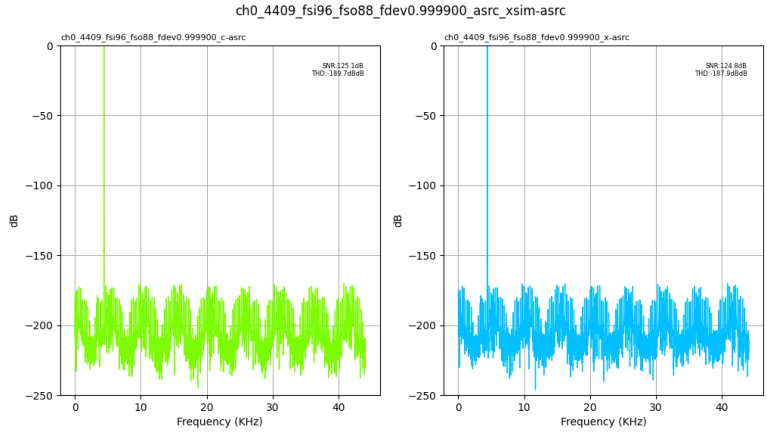


Fig. 44: Input Fs: 96,000Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

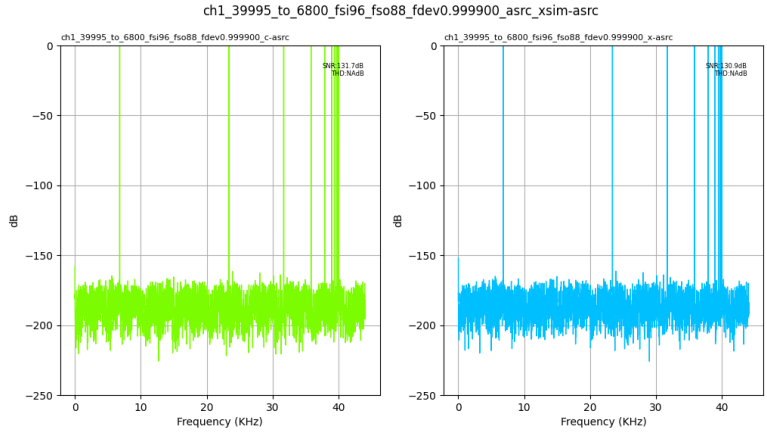


Fig. 45: Input Fs: 96,000Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

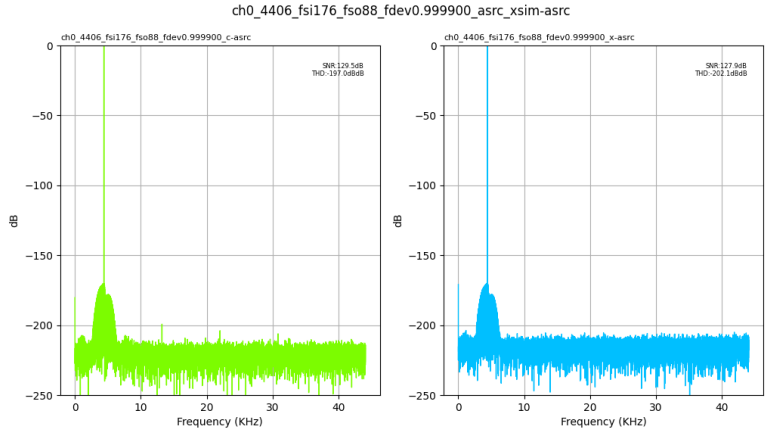


Fig. 46: Input Fs: 176,400Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

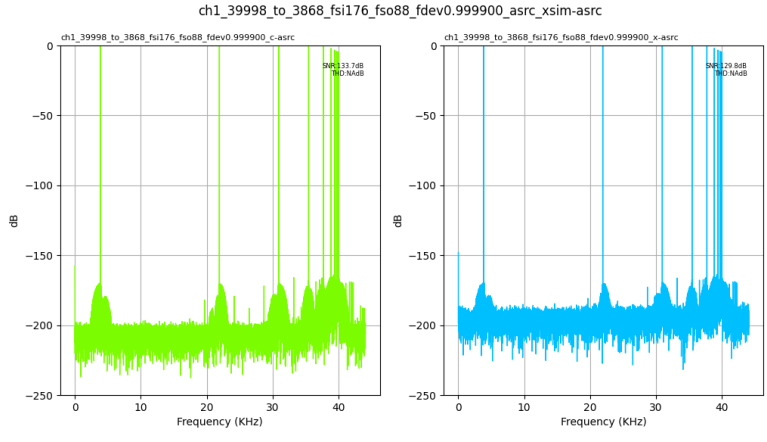


Fig. 47: Input Fs: 176,400Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

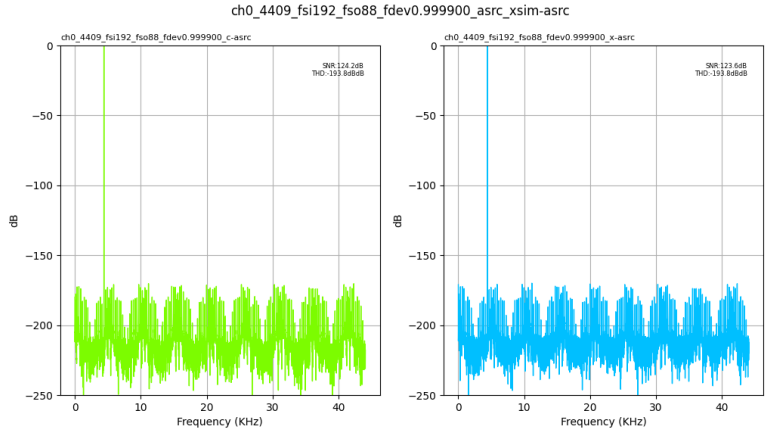


Fig. 48: Input Fs: 192,000Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

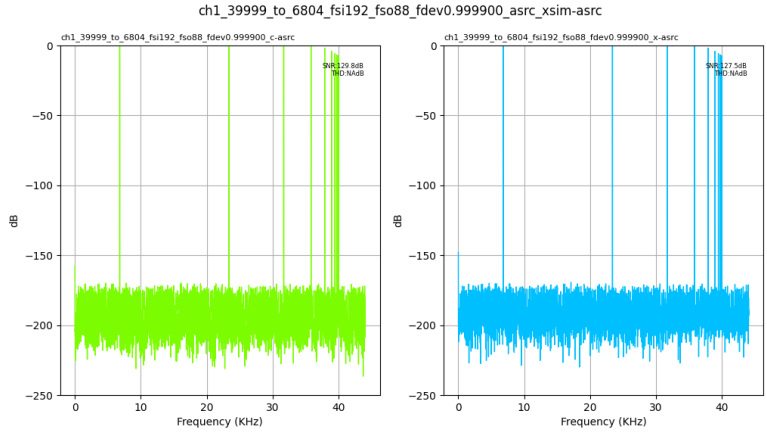


Fig. 49: Input Fs: 192,000Hz, Output Fs: 88,200Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

Output Fs : 96,000Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

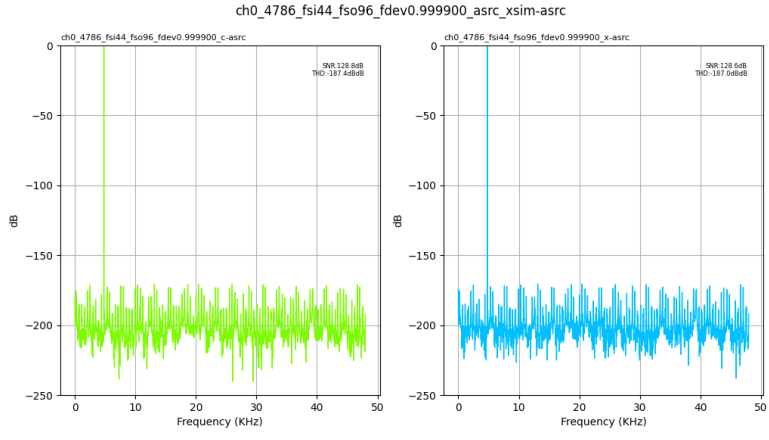


Fig. 50: Input Fs: 44,100Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

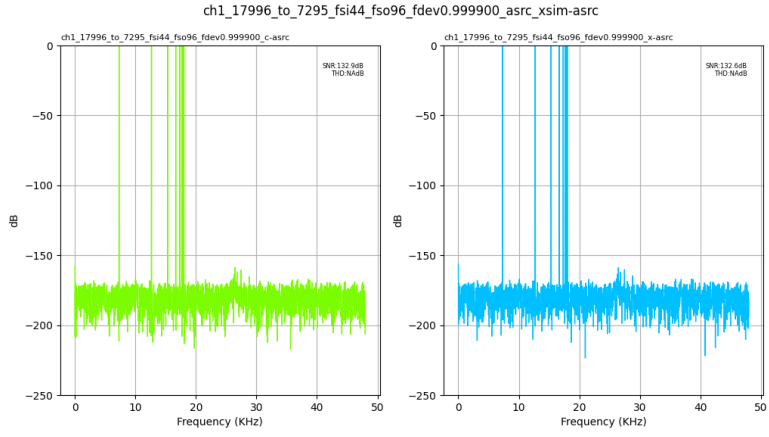


Fig. 51: Input Fs: 44,100Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

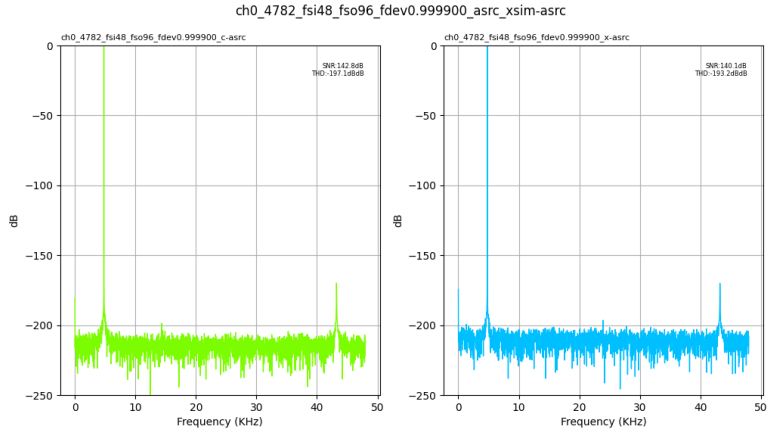


Fig. 52: Input Fs: 48,000Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

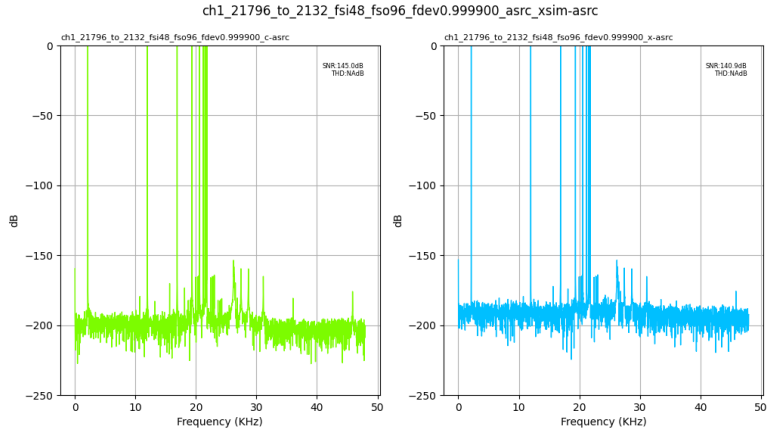


Fig. 53: Input Fs: 48,000Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

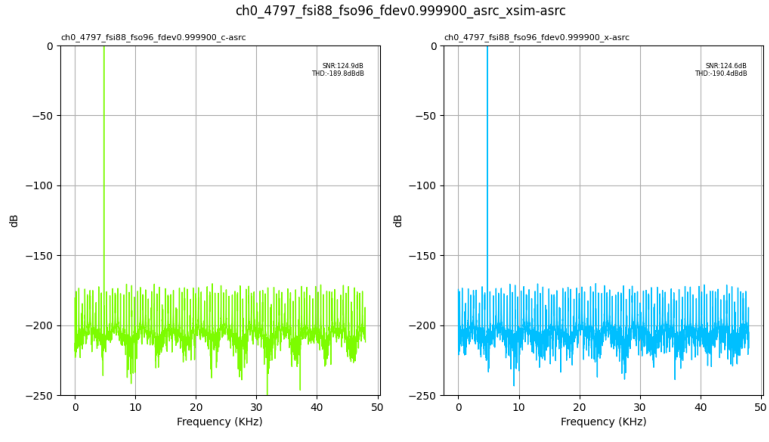


Fig. 54: Input Fs: 88,200Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

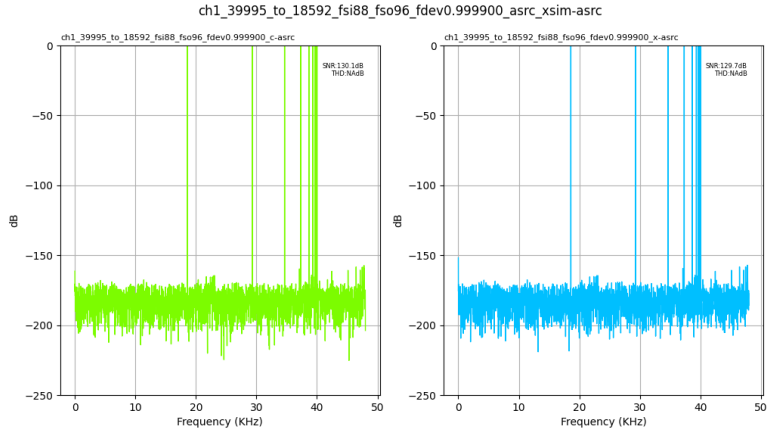


Fig. 55: Input Fs: 88,200Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

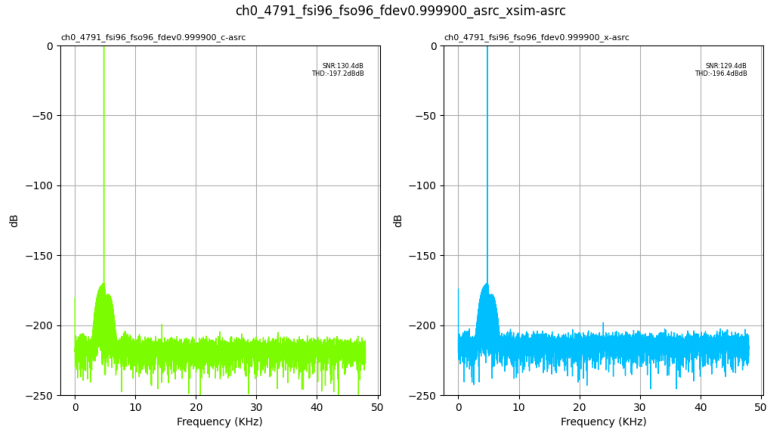


Fig. 56: Input Fs: 96,000Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

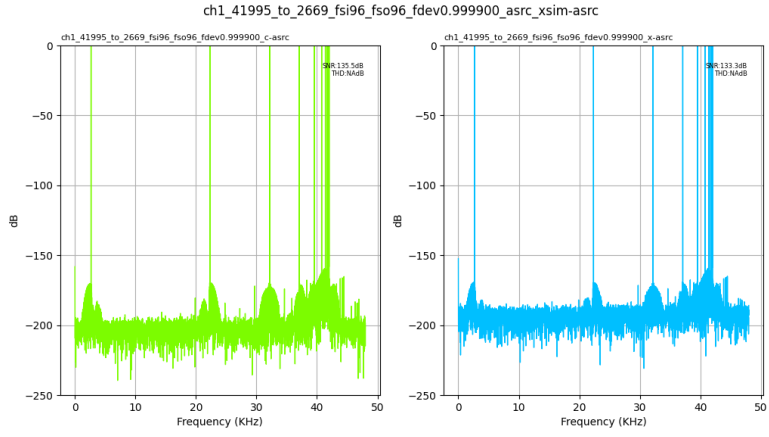


Fig. 57: Input Fs: 96,000Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

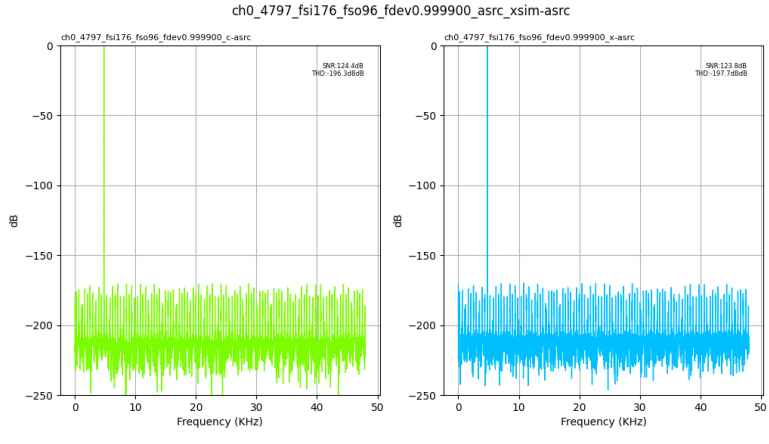


Fig. 58: Input Fs: 176,400Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

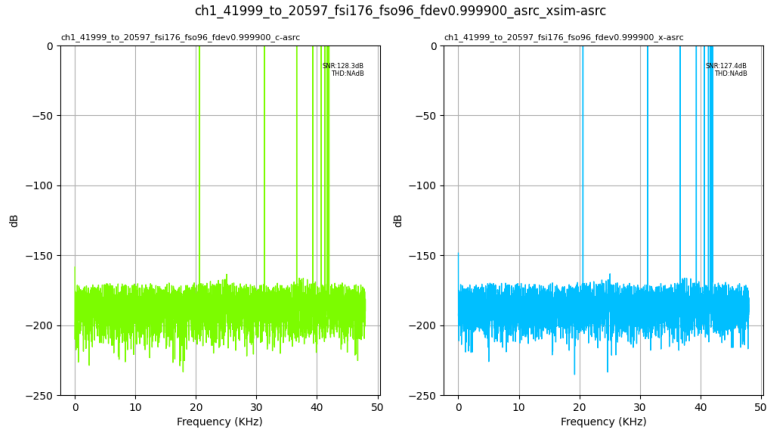


Fig. 59: Input Fs: 176,400Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

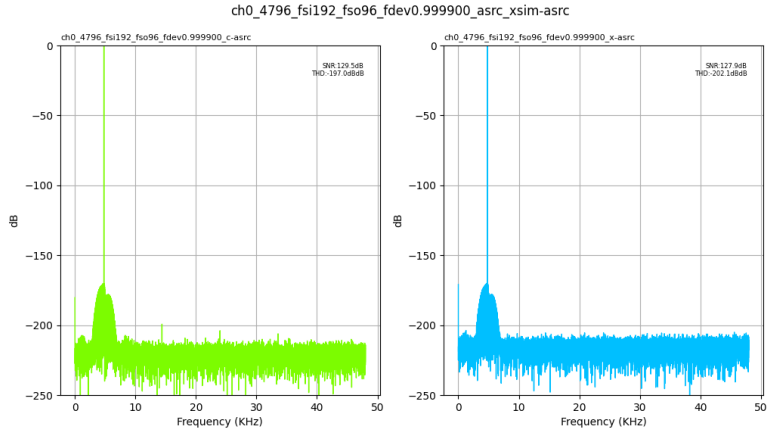


Fig. 60: Input Fs: 192,000Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

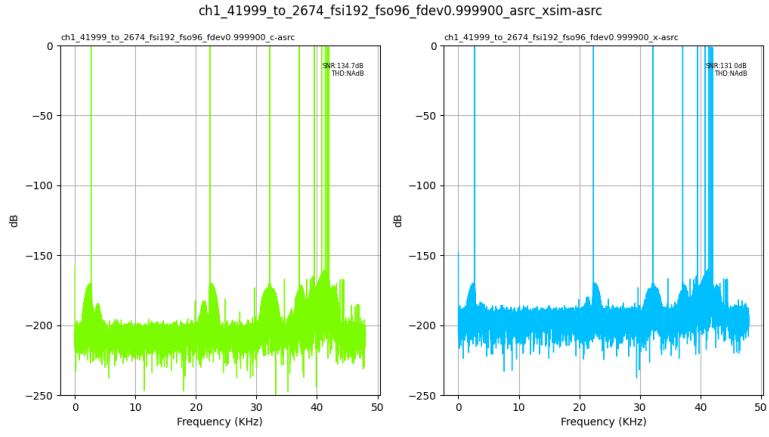


Fig. 61: Input Fs: 192,000Hz, Output Fs: 96,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

Output Fs : 176,400Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

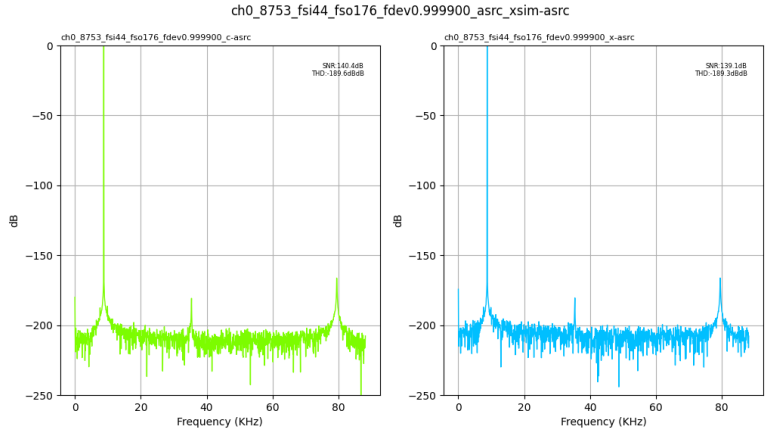


Fig. 62: Input Fs: 44,100Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

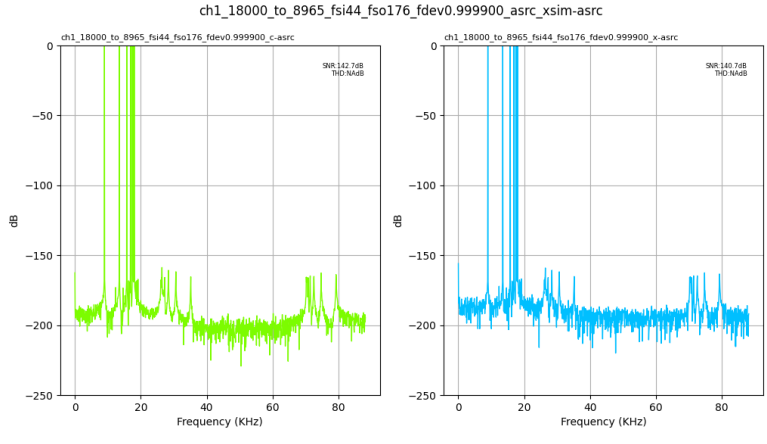


Fig. 63: Input Fs: 44,100Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

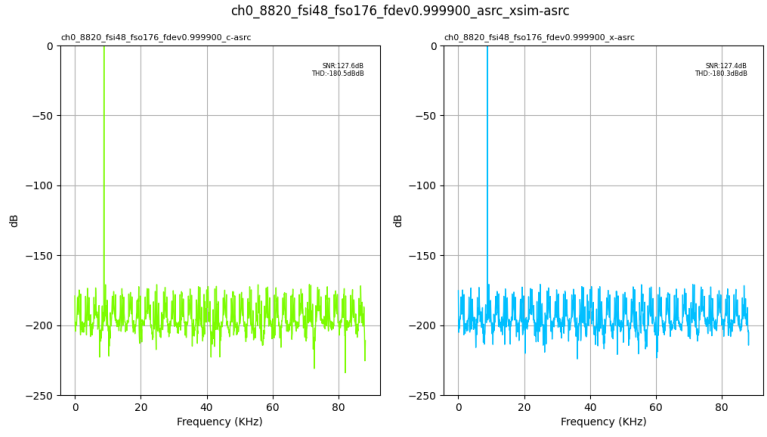


Fig. 64: Input Fs: 48,000Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

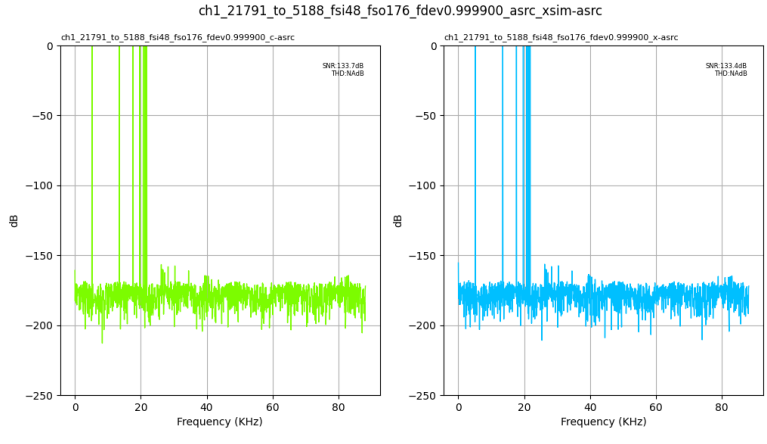


Fig. 65: Input Fs: 48,000Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

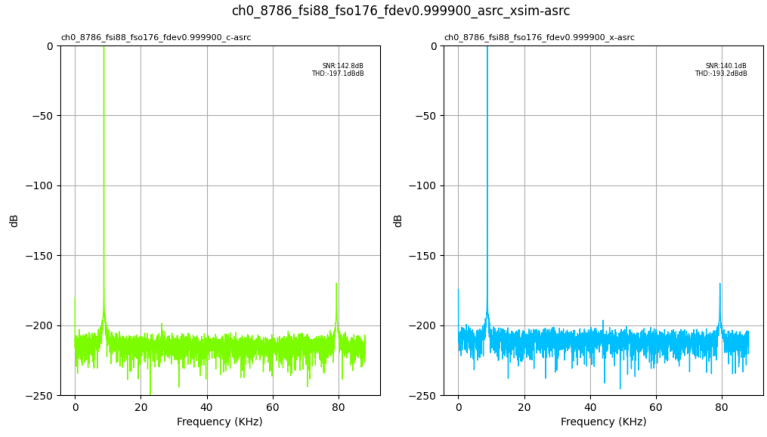


Fig. 66: Input Fs: 88,200Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

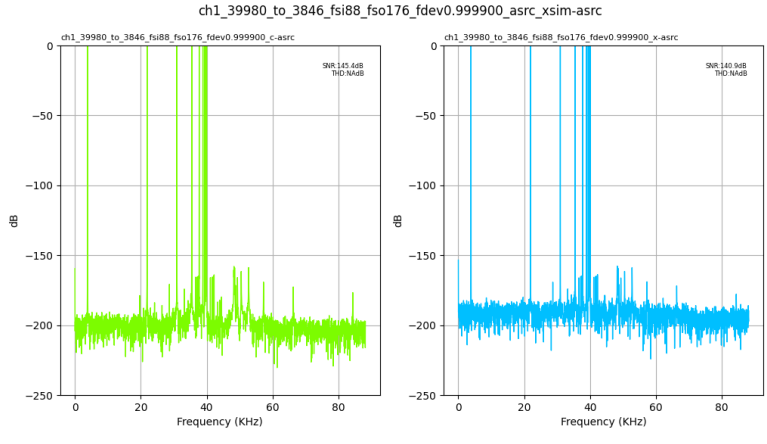


Fig. 67: Input Fs: 88,200Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

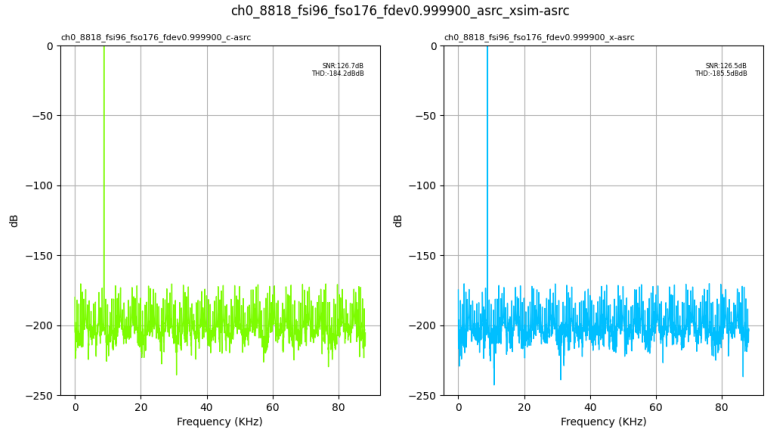


Fig. 68: Input Fs: 96,000Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

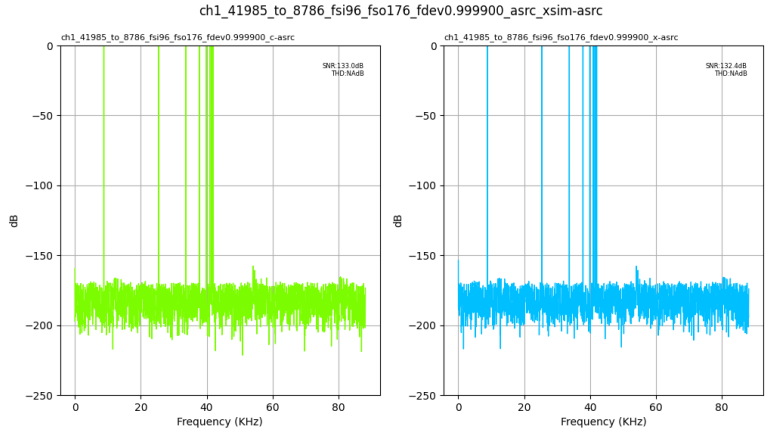


Fig. 69: Input Fs: 96,000Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

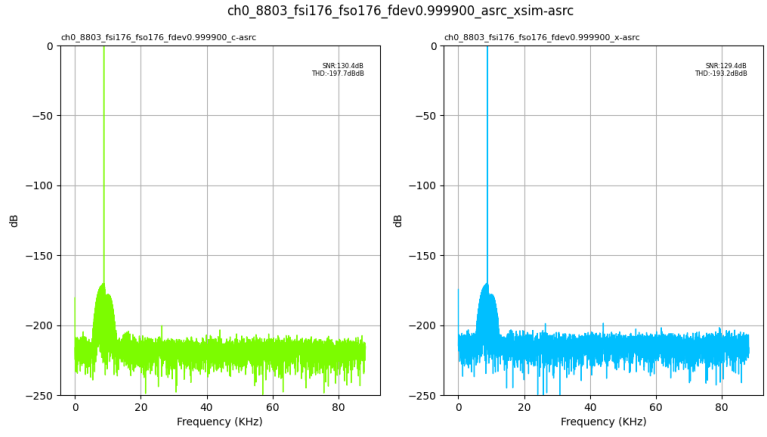


Fig. 70: Input Fs: 176,400Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

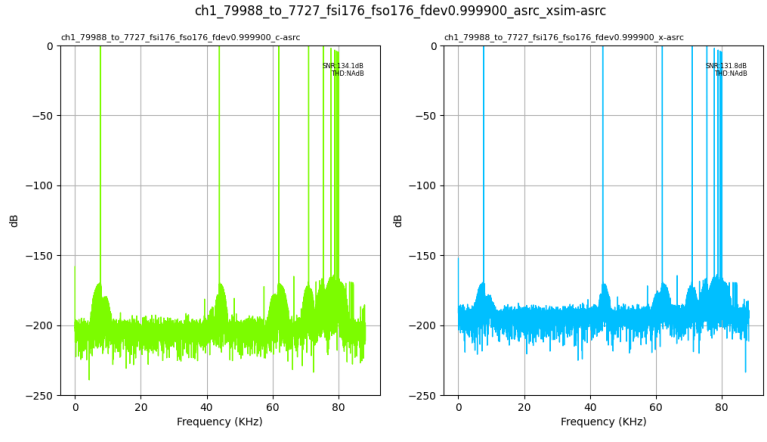


Fig. 71: Input Fs: 176,400Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

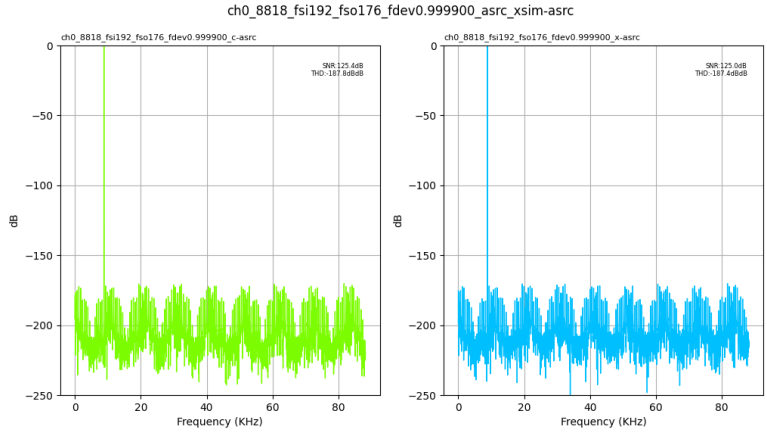


Fig. 72: Input Fs: 192,000Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

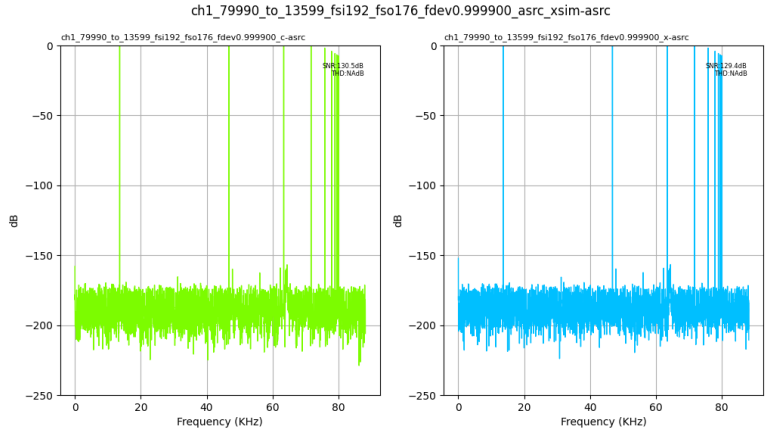


Fig. 73: Input Fs: 192,000Hz, Output Fs: 176,400Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

Output Fs : 192,000Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

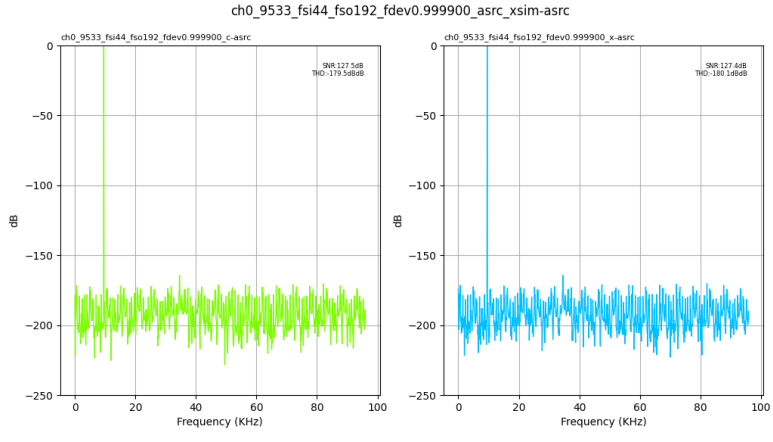


Fig. 74: Input Fs: 44,100Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

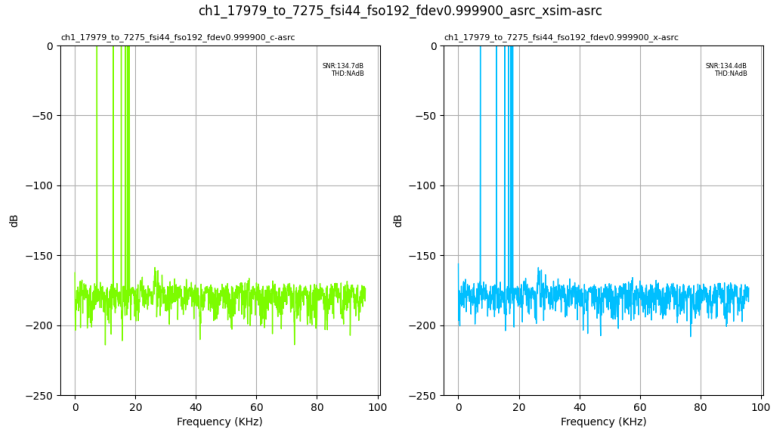


Fig. 75: Input Fs: 44,100Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

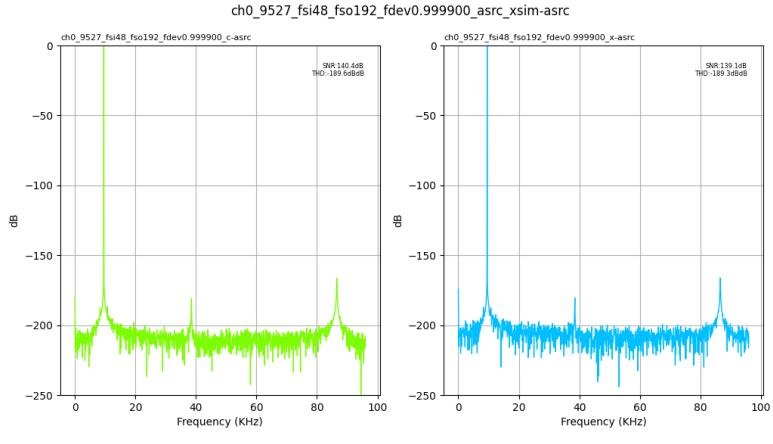


Fig. 76: Input Fs: 48,000Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

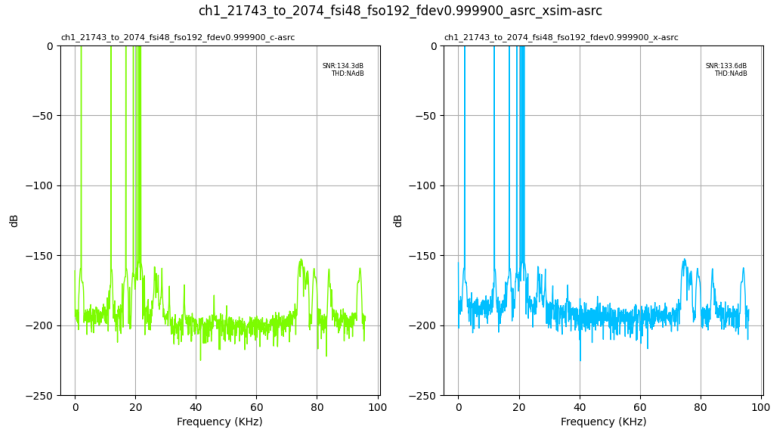


Fig. 77: Input Fs: 48,000Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

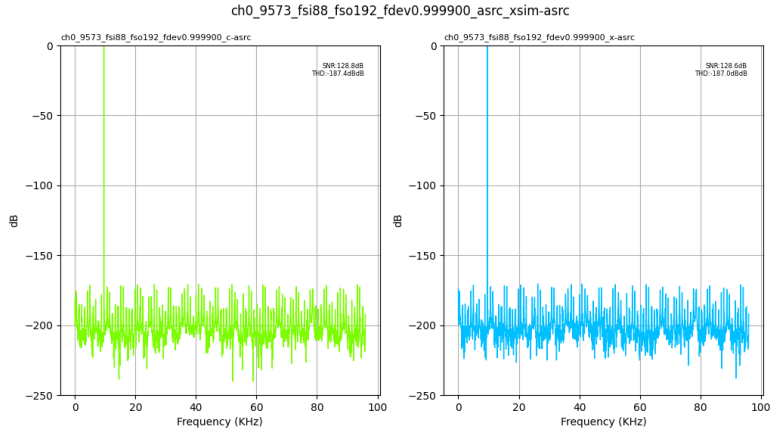


Fig. 78: Input Fs: 88,200Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

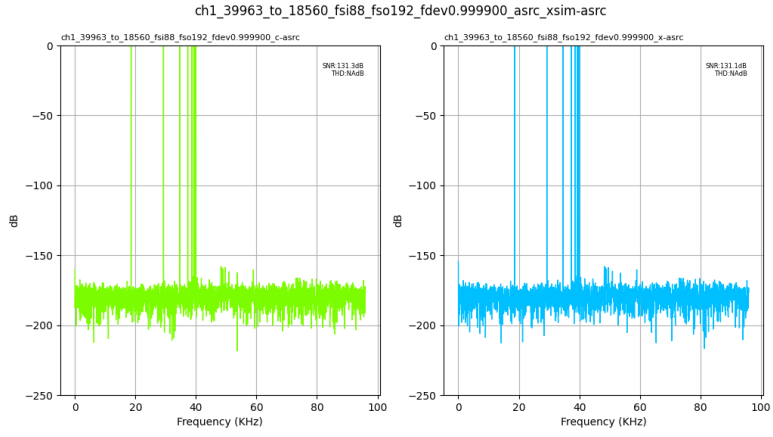


Fig. 79: Input Fs: 88,200Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

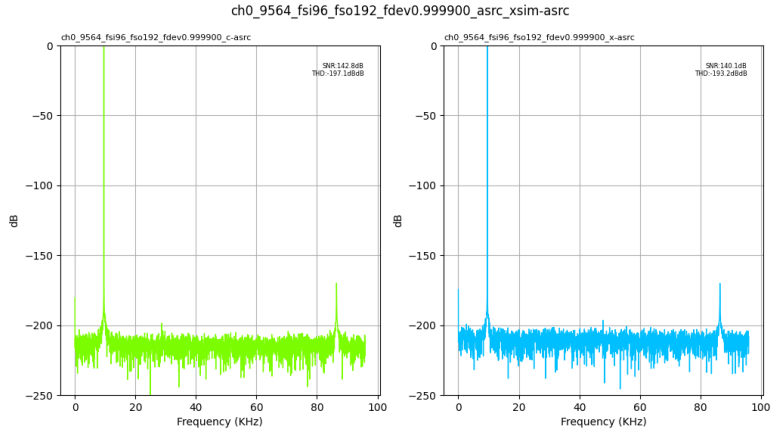


Fig. 80: Input Fs: 96,000Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

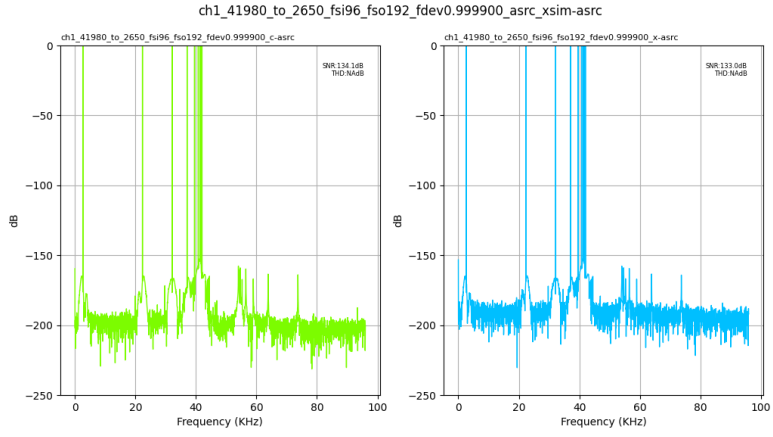


Fig. 81: Input Fs: 96,000Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

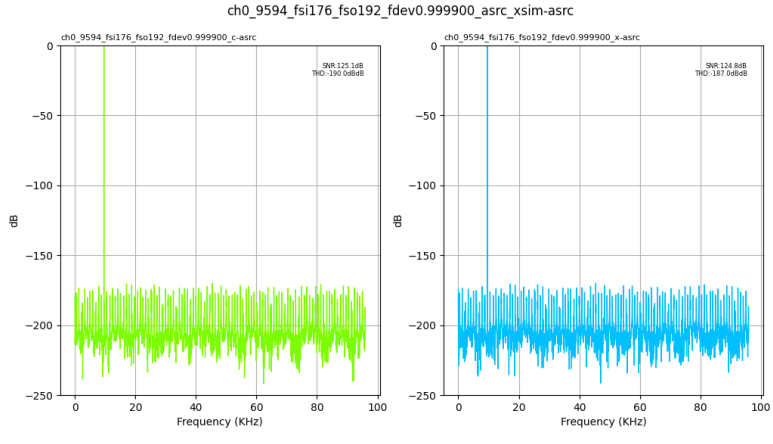


Fig. 82: Input Fs: 176,400Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

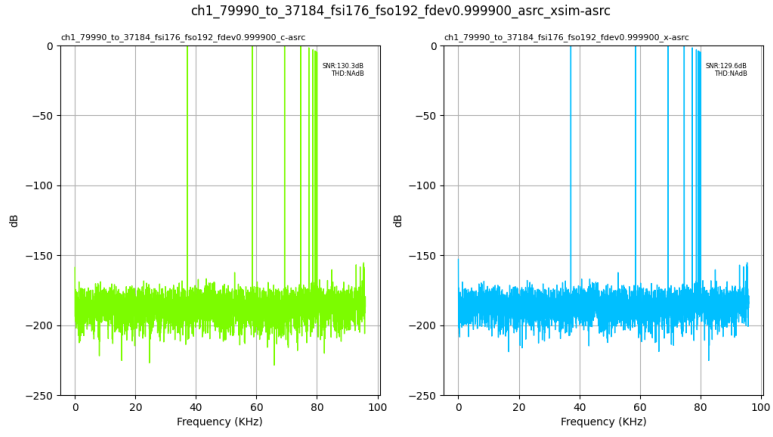


Fig. 83: Input Fs: 176,400Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

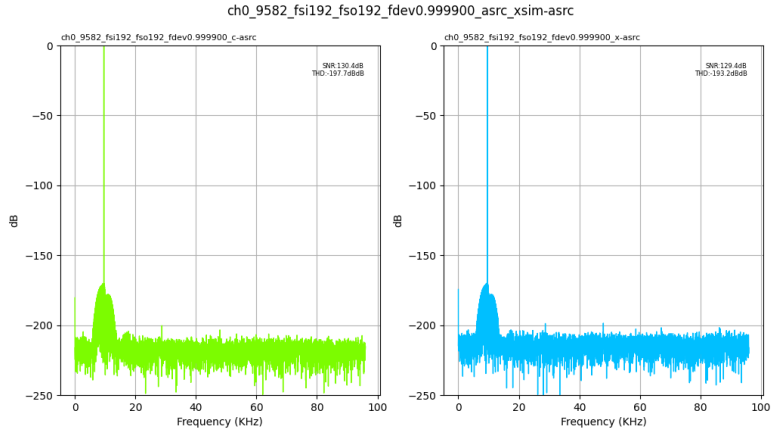


Fig. 84: Input Fs: 192,000Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

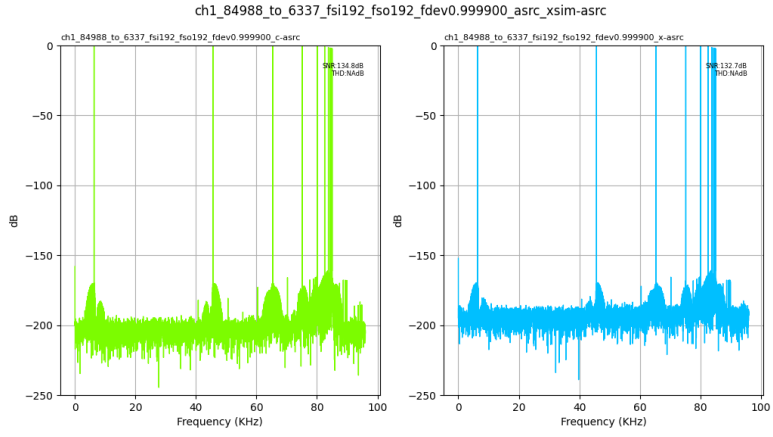


Fig. 85: Input Fs: 192,000Hz, Output Fs: 192,000Hz, Fs error: 0.999900, Results for: asrc, xsim-asrc

8.1.2 Frequency error: 1.000000Hz

Output Fs : 16,000Hz

- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*

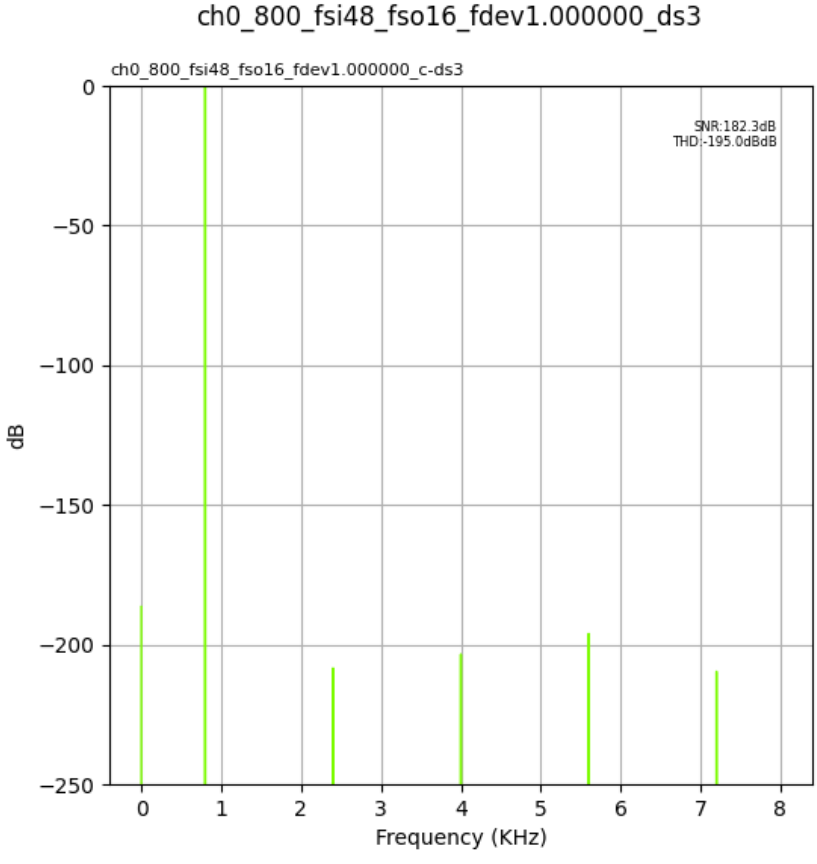


Fig. 86: Input Fs: 48,000Hz, Output Fs: 16,000Hz, Fs error: 1.000000, Results for: ds3

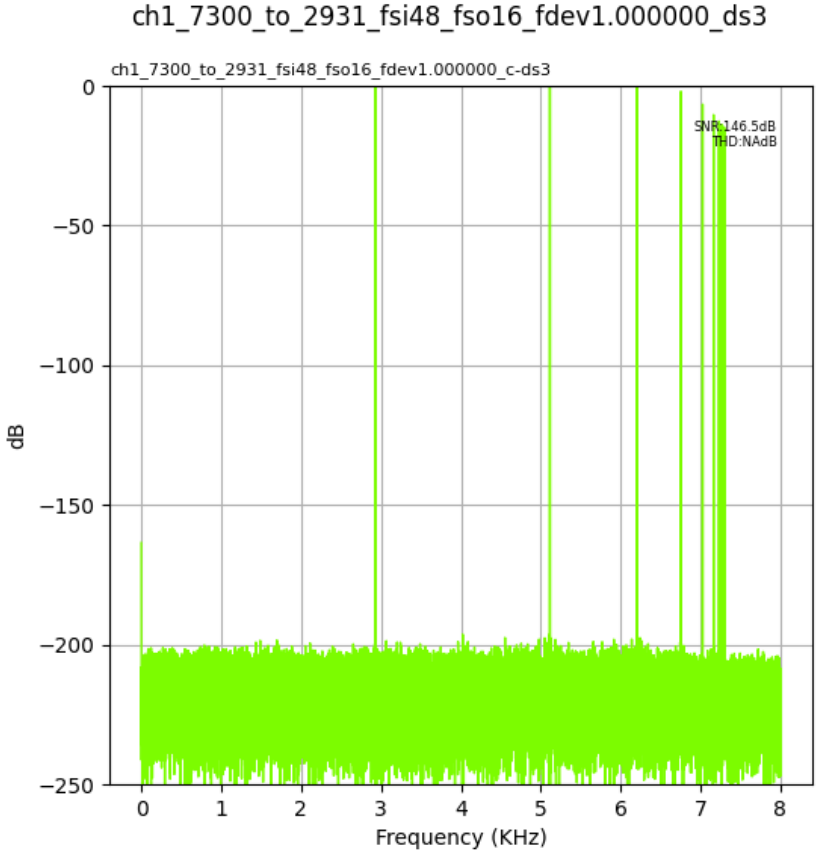


Fig. 87: Input Fs: 48,000Hz, Output Fs: 16,000Hz, Fs error: 1.000000, Results for: ds3

Output Fs : 32,000Hz

- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*

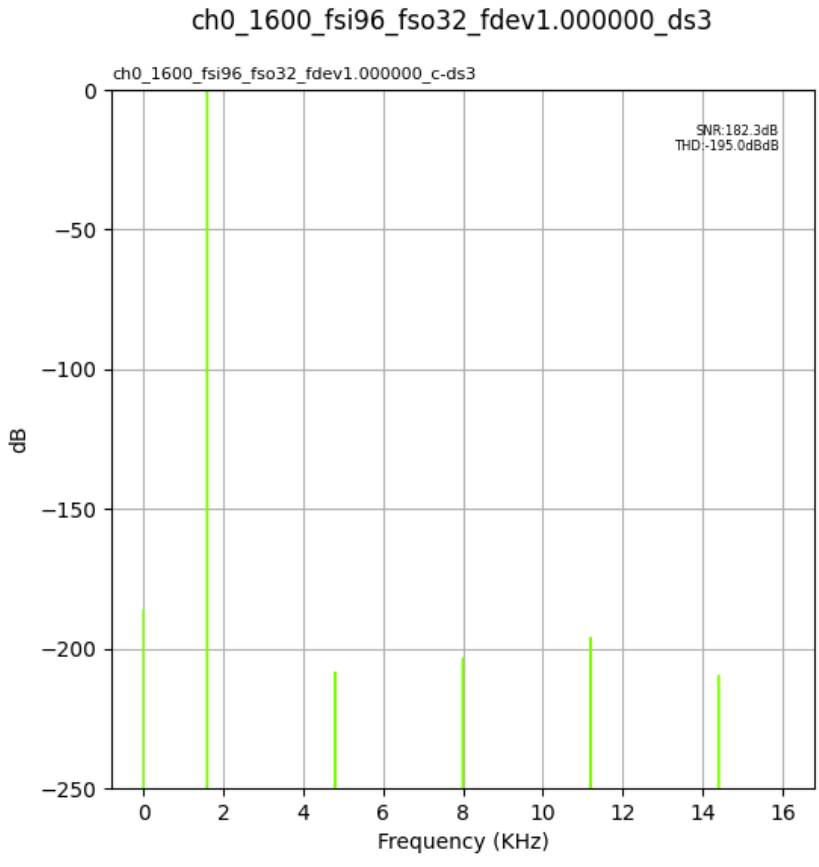


Fig. 88: Input Fs: 96,000Hz, Output Fs: 32,000Hz, Fs error: 1.000000, Results for: ds3

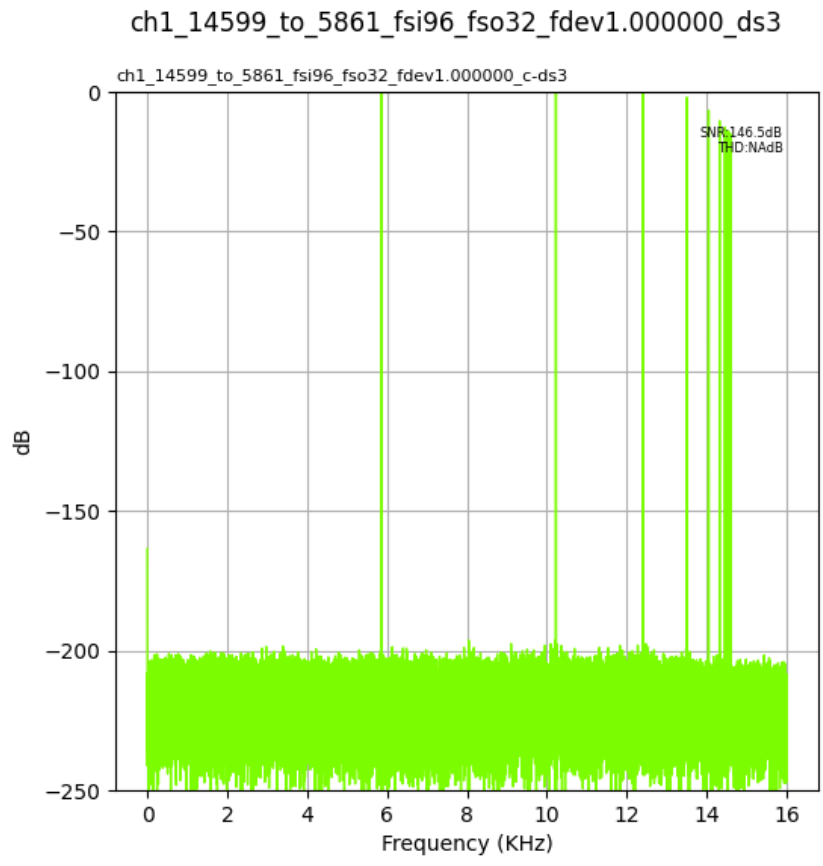


Fig. 89: Input Fs: 96,000Hz, Output Fs: 32,000Hz, Fs error: 1.000000, Results for: ds3

Output Fs : 44,100Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*

- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

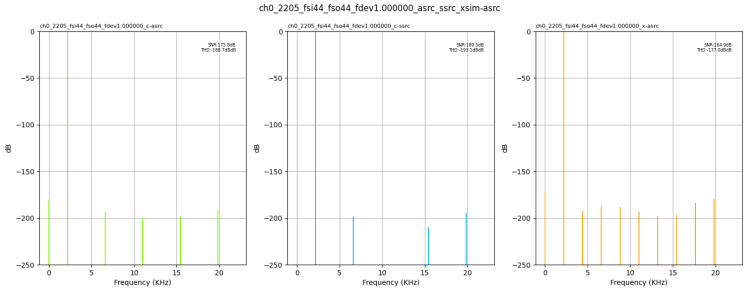


Fig. 90: Input Fs: 44,100Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

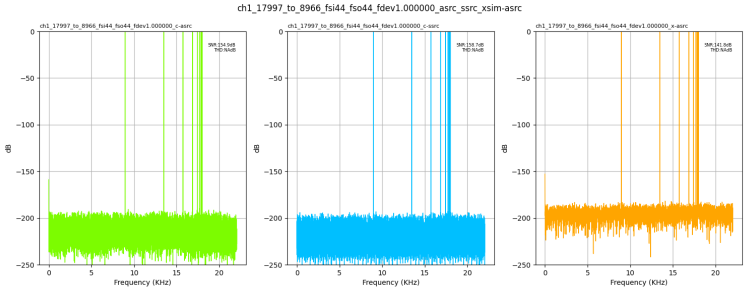


Fig. 91: Input Fs: 44,100Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

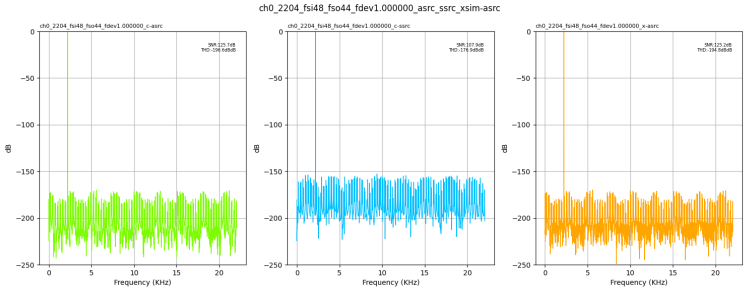


Fig. 92: Input Fs: 48,000Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

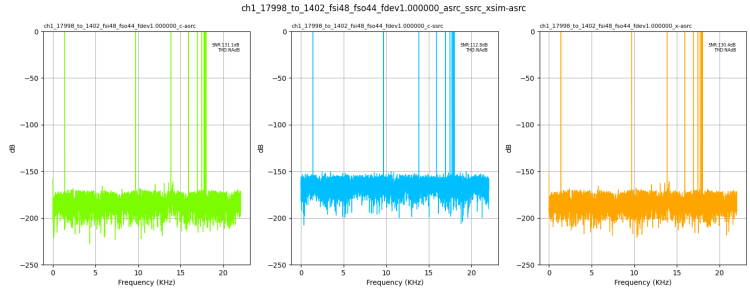


Fig. 93: Input Fs: 48,000Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

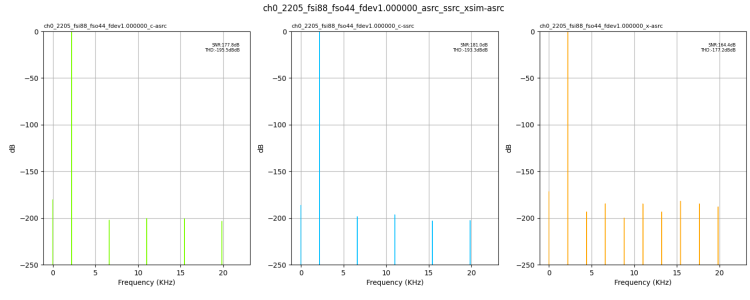


Fig. 94: Input Fs: 88,200Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

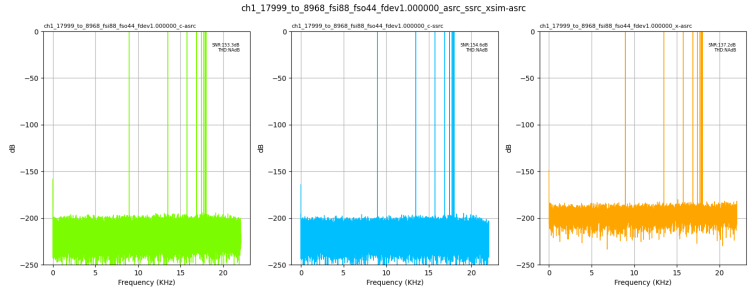


Fig. 95: Input Fs: 88,200Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

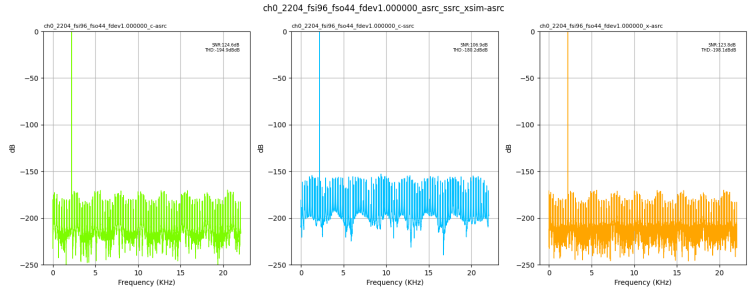


Fig. 96: Input Fs: 96,000Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

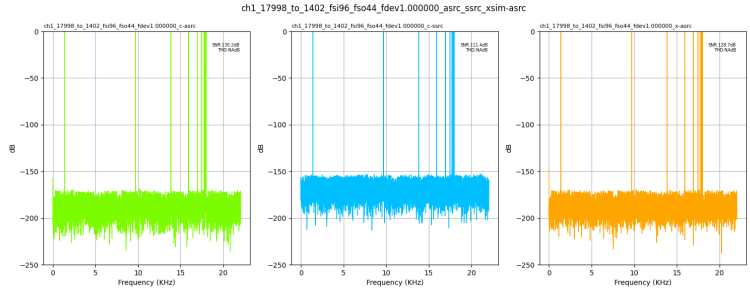


Fig. 97: Input Fs: 96,000Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

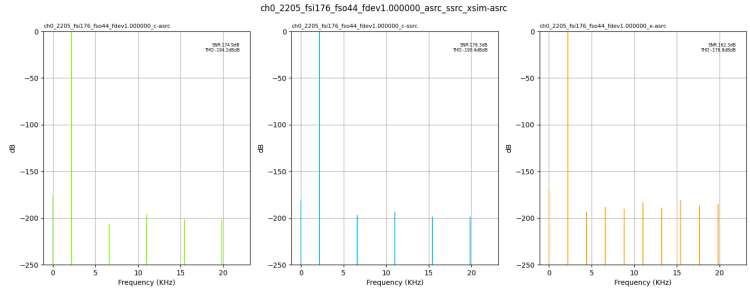


Fig. 98: Input Fs: 176,400Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

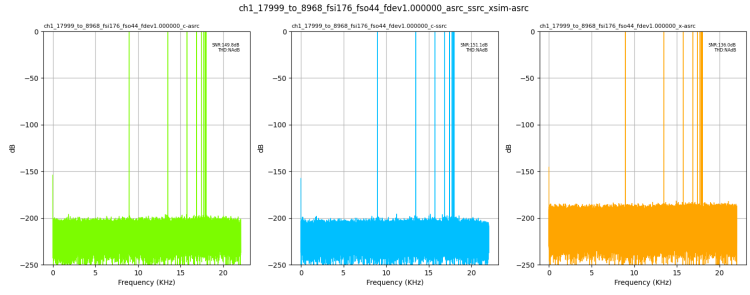


Fig. 99: Input Fs: 176,400Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

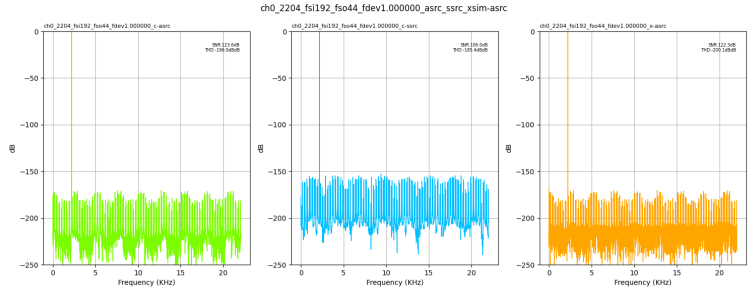


Fig. 100: Input Fs: 192,000Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

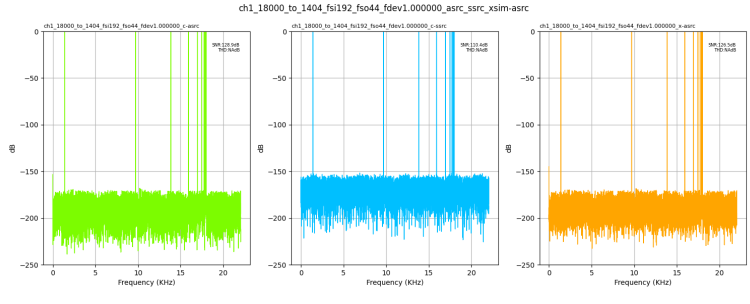


Fig. 101: Input Fs: 192,000Hz, Output Fs: 44,100Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

Output Fs : 48,000Hz

- *Input Fs: 16,000Hz, channel 0*
- *Input Fs: 16,000Hz, channel 1*
- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

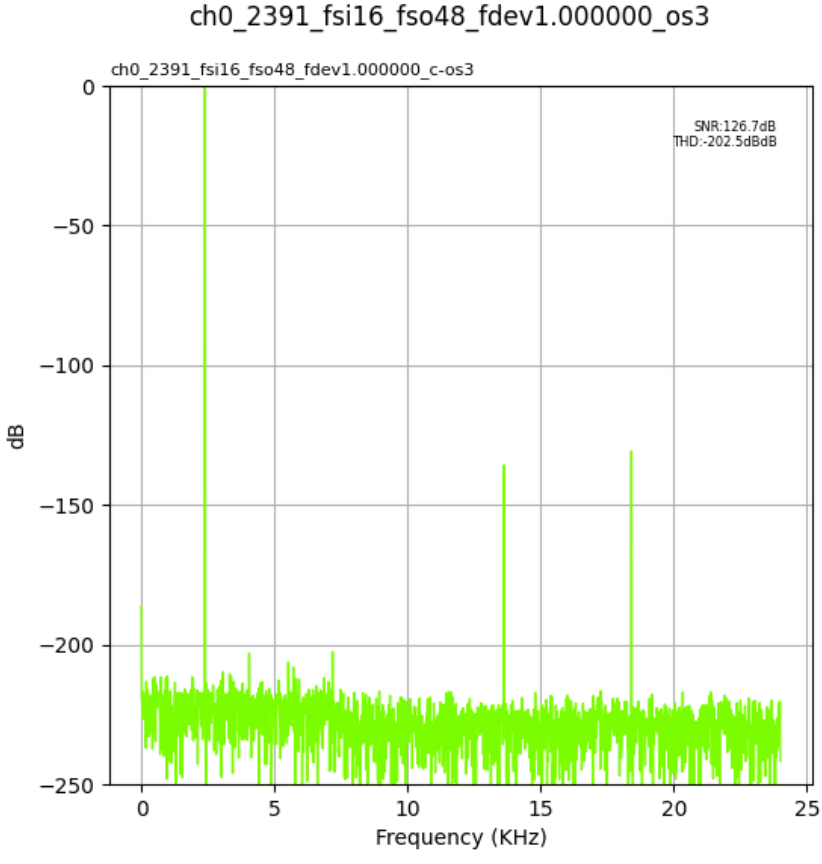


Fig. 102: Input Fs: 16,000Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: os3

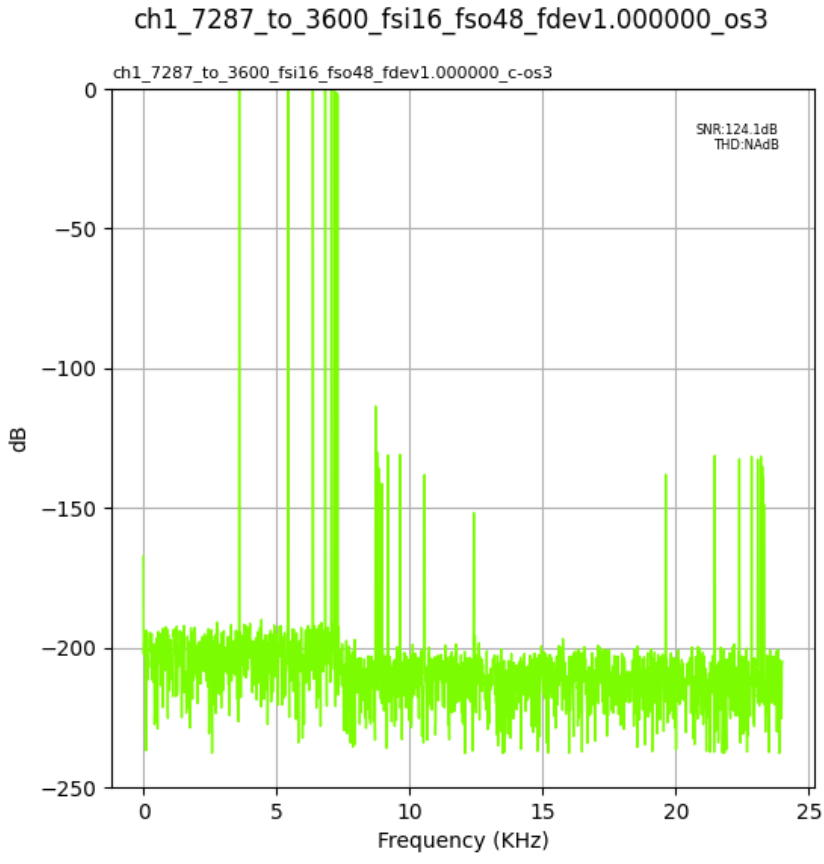


Fig. 103: Input Fs: 16,000Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: os3

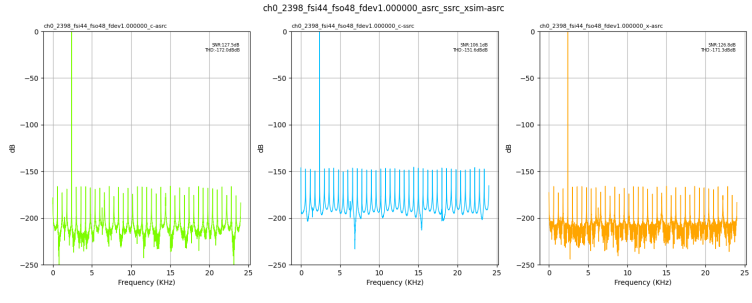


Fig. 104: Input Fs: 44,100Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

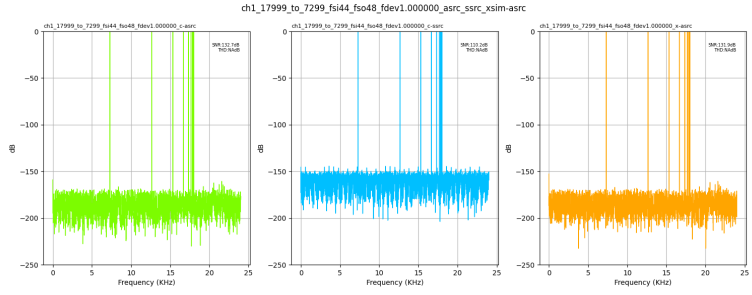


Fig. 105: Input Fs: 44,100Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

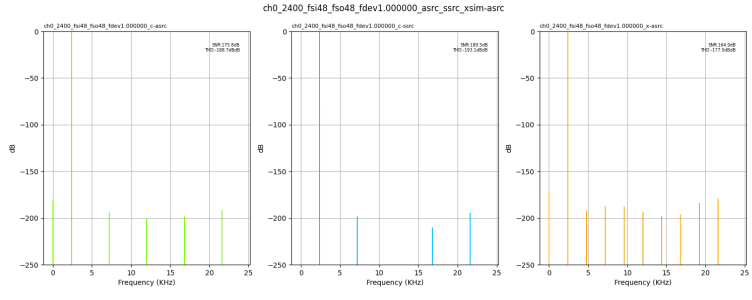


Fig. 106: Input Fs: 48,000Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

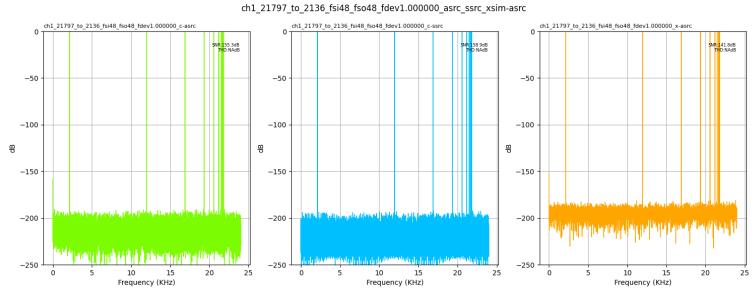


Fig. 107: Input Fs: 48,000Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

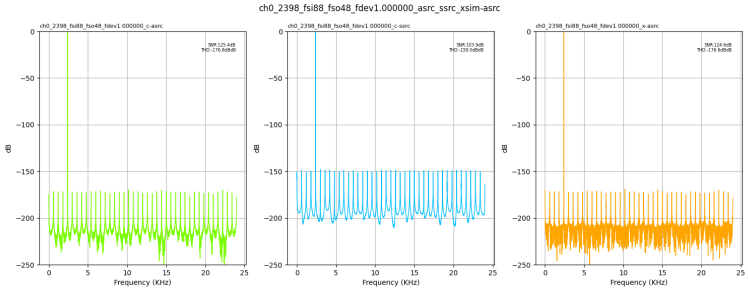


Fig. 108: Input Fs: 88,200Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

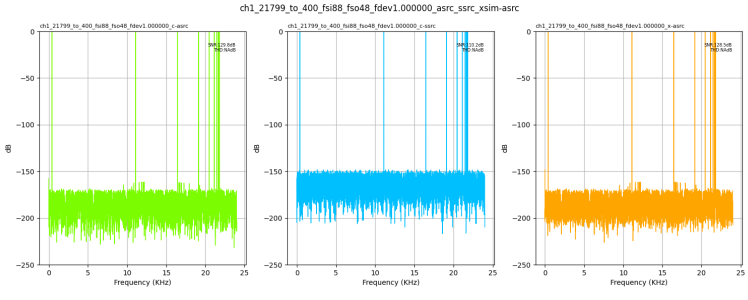


Fig. 109: Input Fs: 88,200Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

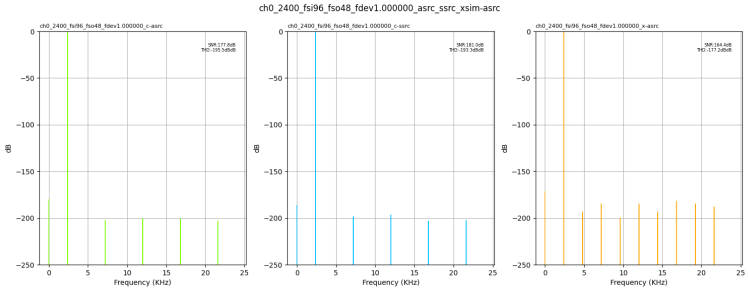


Fig. 110: Input Fs: 96,000Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

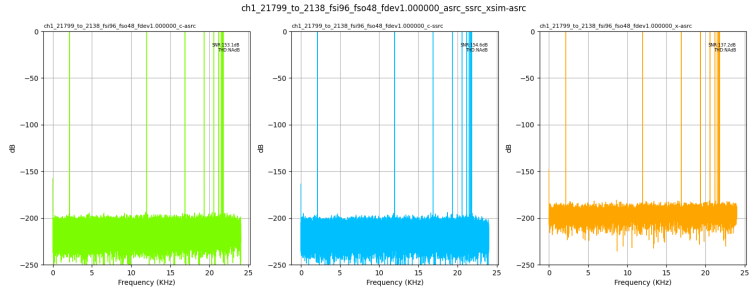


Fig. 111: Input Fs: 96,000Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

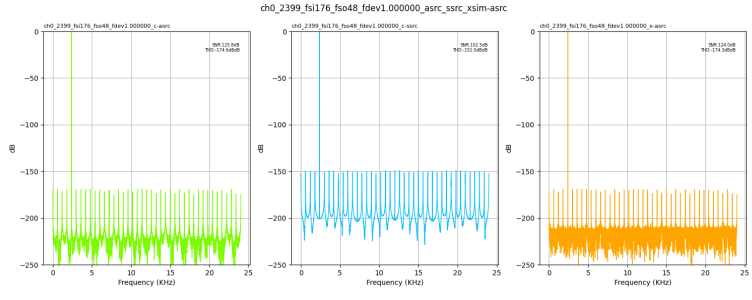


Fig. 112: Input Fs: 176,400Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

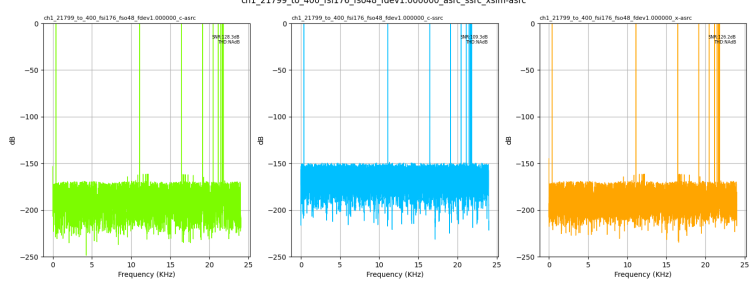


Fig. 113: Input Fs: 176,400Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

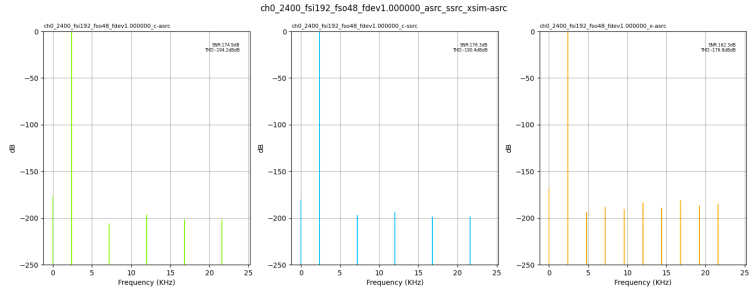


Fig. 114: Input Fs: 192,000Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

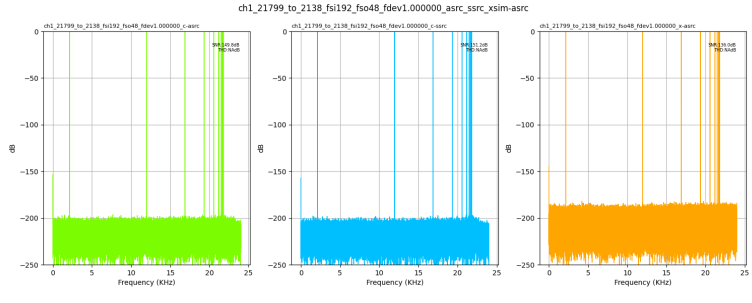


Fig. 115: Input Fs: 192,000Hz, Output Fs: 48,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

Output Fs : 88,200Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

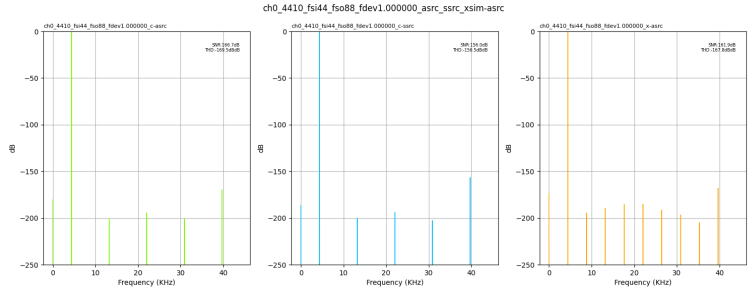


Fig. 116: Input Fs: 44,100Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

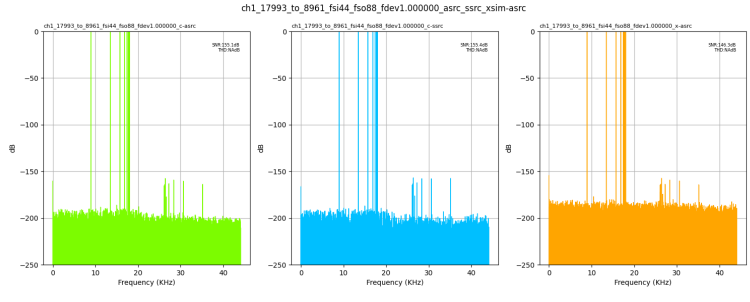


Fig. 117: Input Fs: 44,100Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

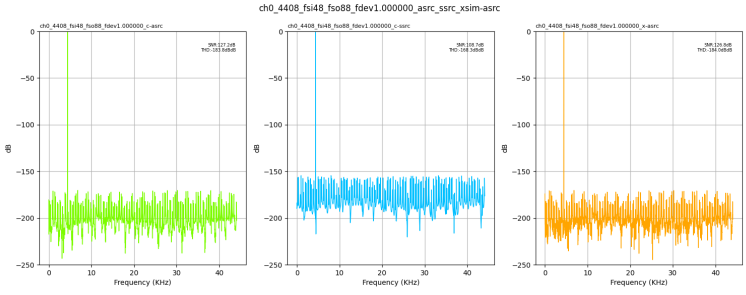


Fig. 118: Input Fs: 48,000Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

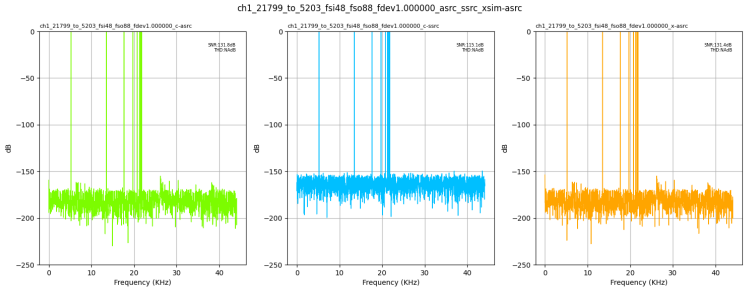


Fig. 119: Input Fs: 48,000Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

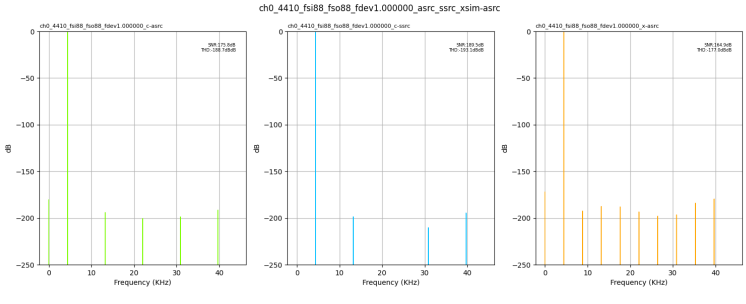


Fig. 120: Input Fs: 88,200Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

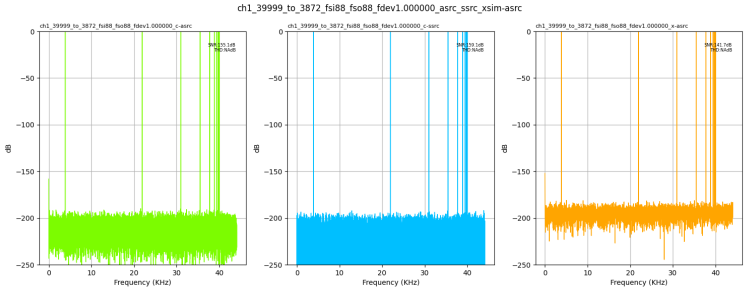


Fig. 121: Input Fs: 88,200Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

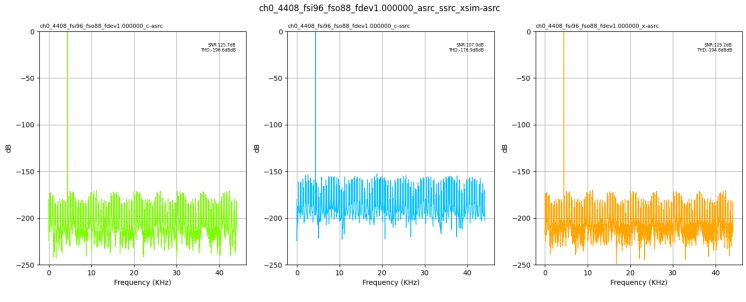


Fig. 122: Input Fs: 96,000Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

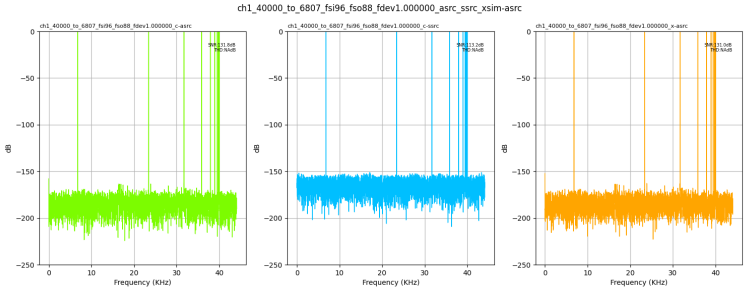


Fig. 123: Input Fs: 96,000Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

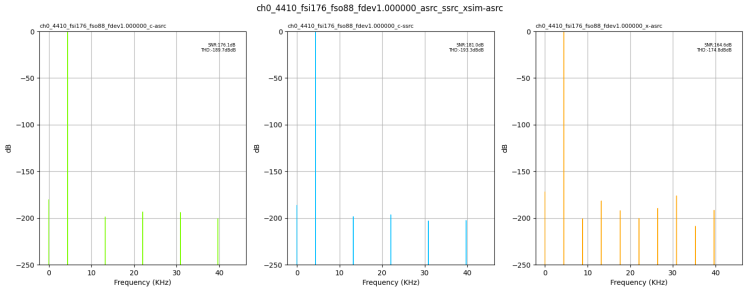


Fig. 124: Input Fs: 176,400Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

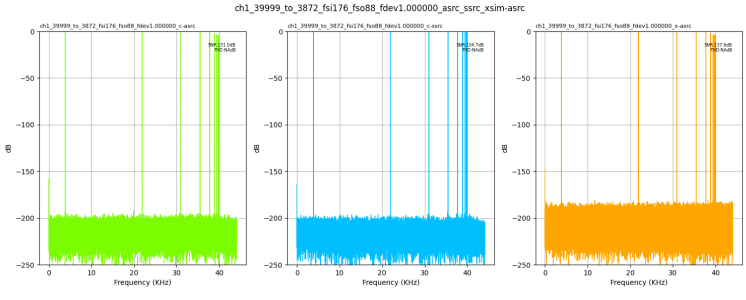


Fig. 125: Input Fs: 176,400Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

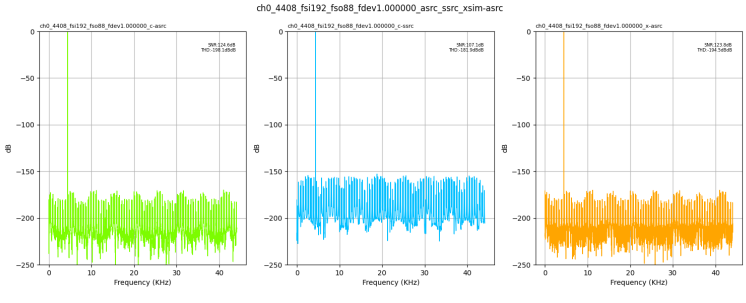


Fig. 126: Input Fs: 192,000Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

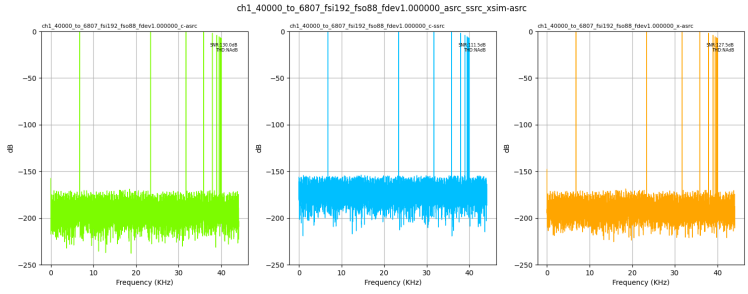


Fig. 127: Input Fs: 192,000Hz, Output Fs: 88,200Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

Output Fs : 96,000Hz

- *Input Fs: 32,000Hz, channel 0*
- *Input Fs: 32,000Hz, channel 1*
- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

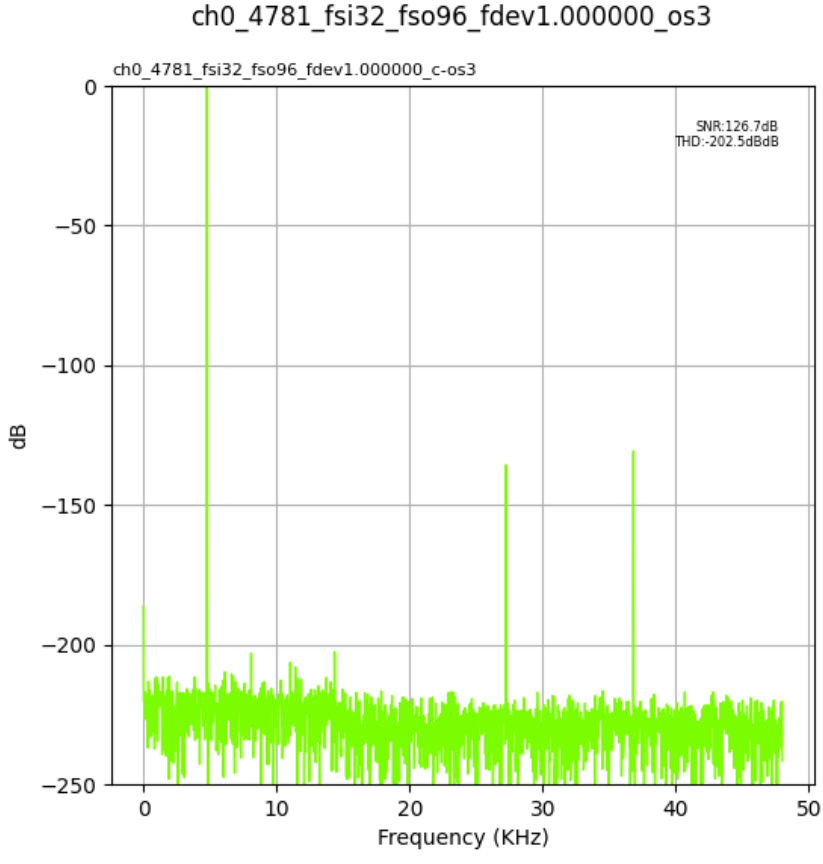


Fig. 128: Input Fs: 32,000Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: os3

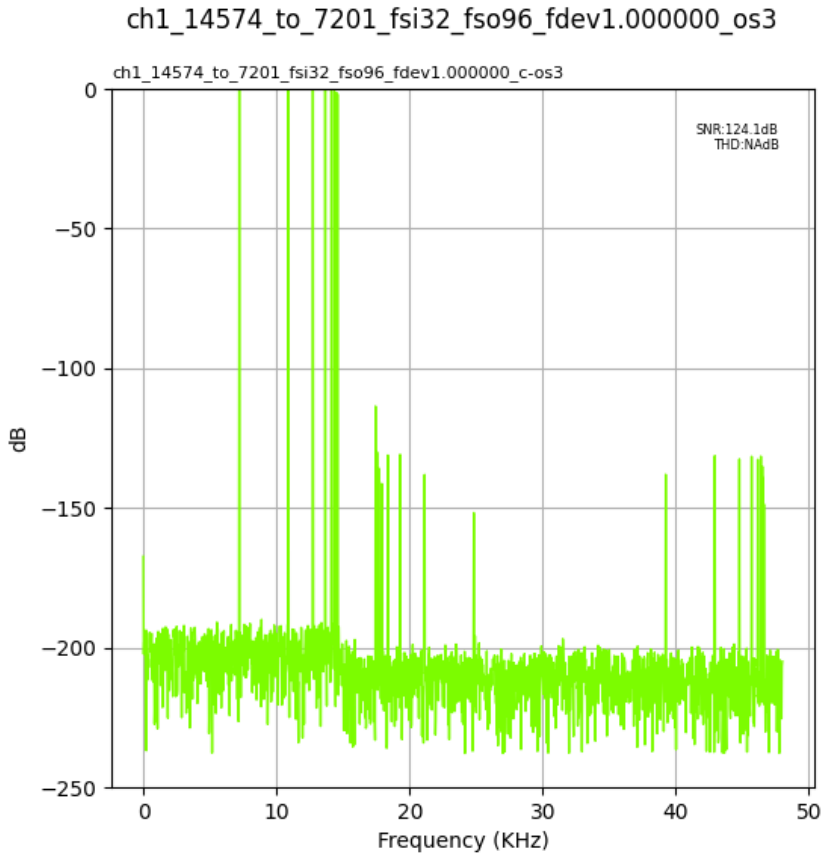


Fig. 129: Input Fs: 32,000Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: os3

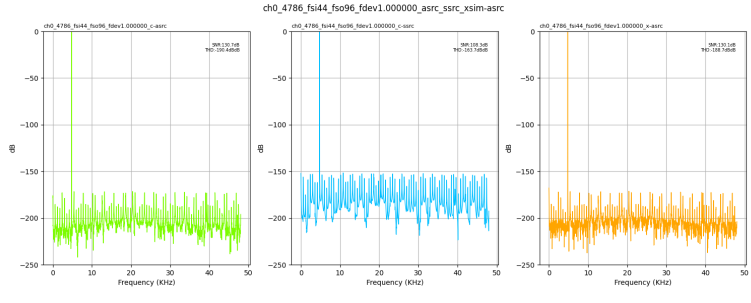


Fig. 130: Input Fs: 44,100Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

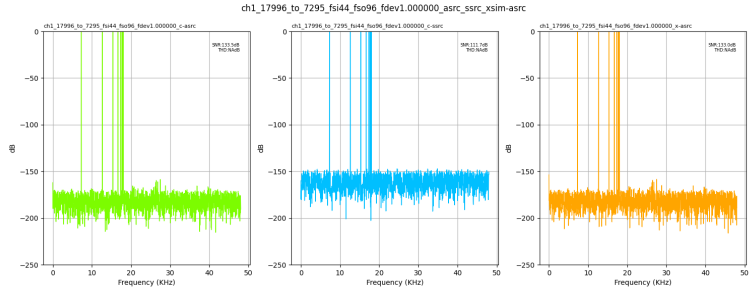


Fig. 131: Input Fs: 44,100Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

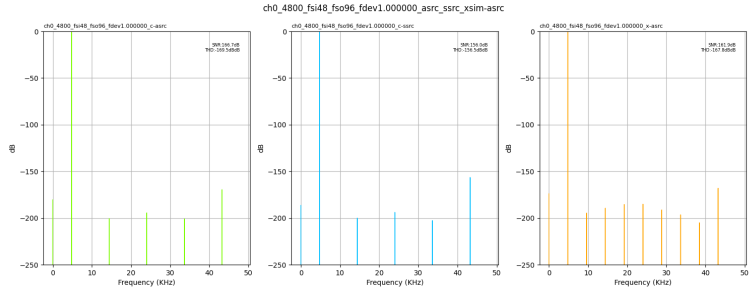


Fig. 132: Input Fs: 48,000Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

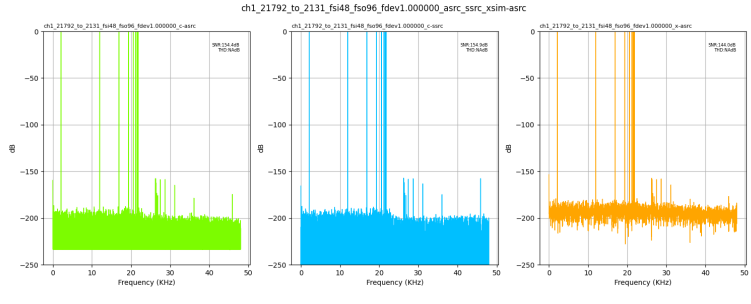


Fig. 133: Input Fs: 48,000Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

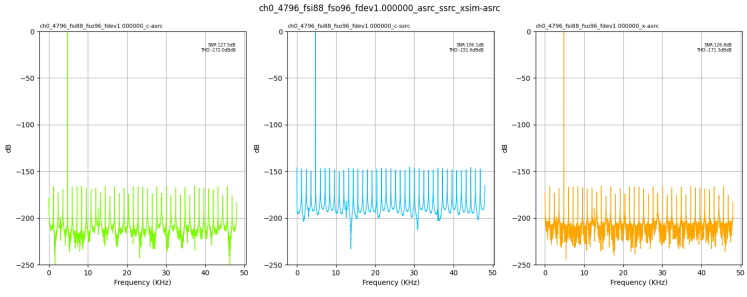


Fig. 134: Input Fs: 88,200Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

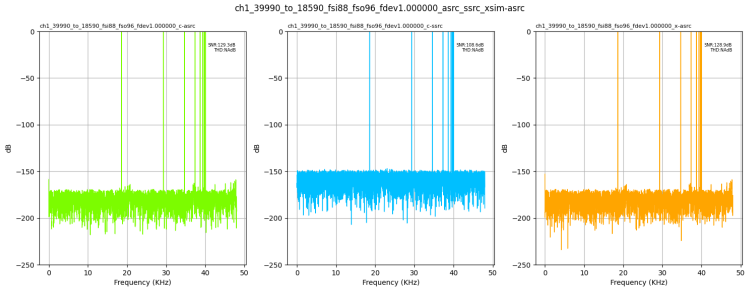


Fig. 135: Input Fs: 88,200Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

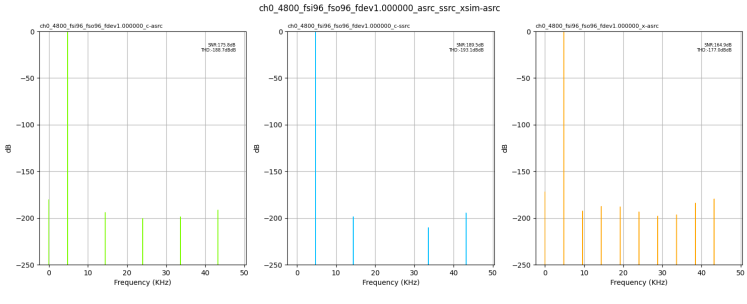


Fig. 136: Input Fs: 96,000Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

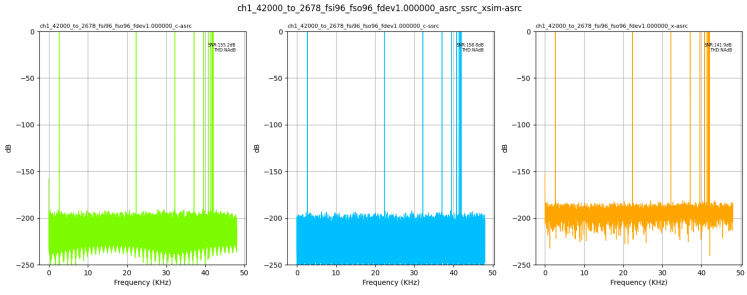


Fig. 137: Input Fs: 96,000Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

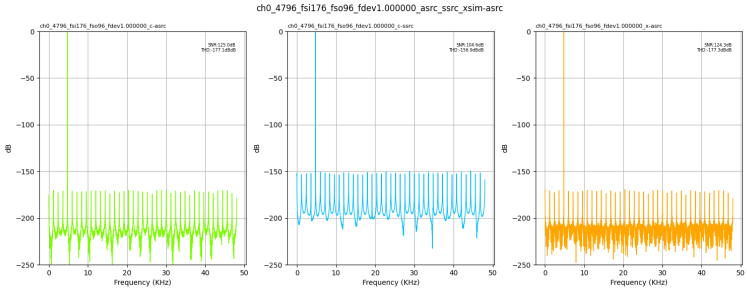


Fig. 138: Input Fs: 176,400Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

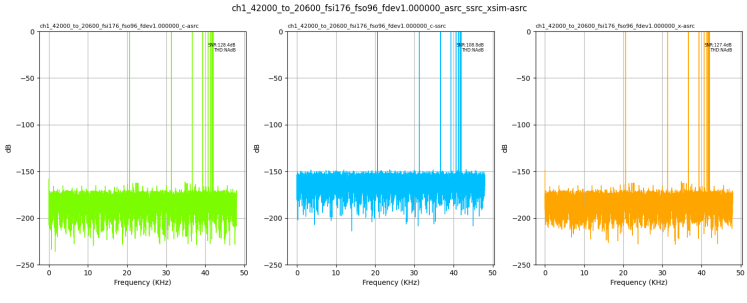


Fig. 139: Input Fs: 176,400Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

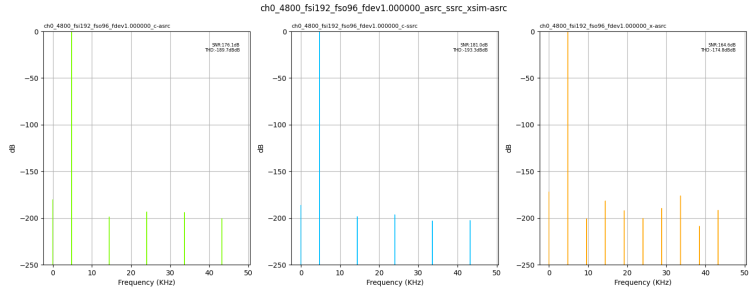


Fig. 140: Input Fs: 192,000Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

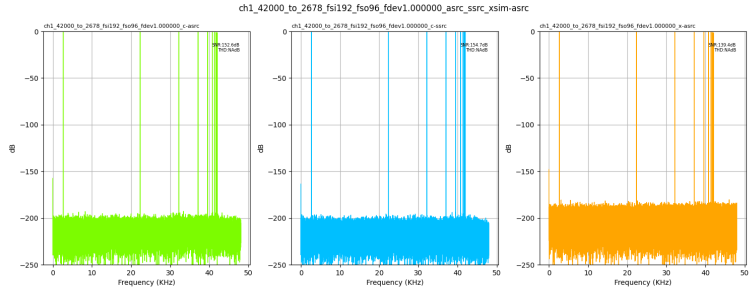


Fig. 141: Input Fs: 192,000Hz, Output Fs: 96,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

Output Fs : 176,400Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

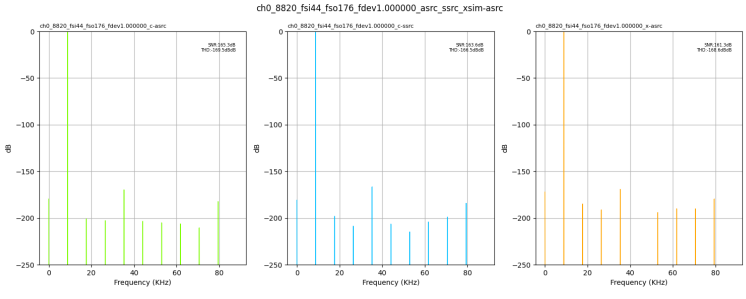


Fig. 142: Input Fs: 44,100Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

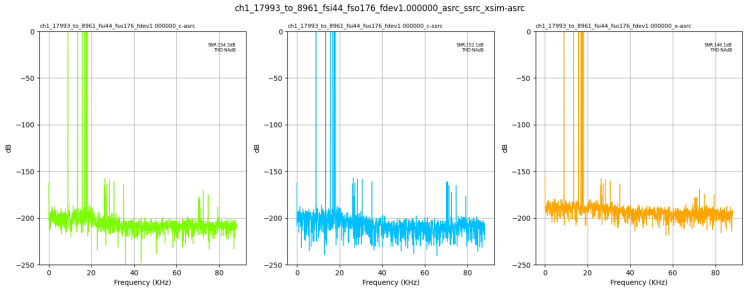


Fig. 143: Input Fs: 44,100Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

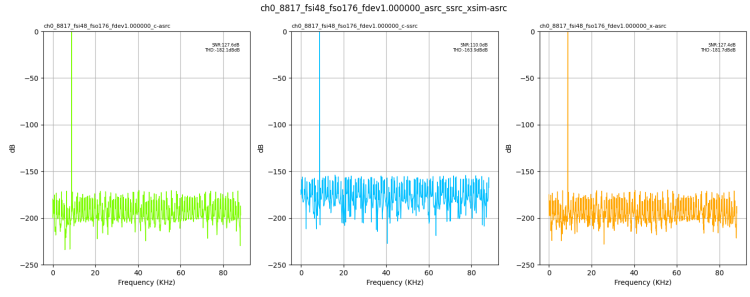


Fig. 144: Input Fs: 48,000Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

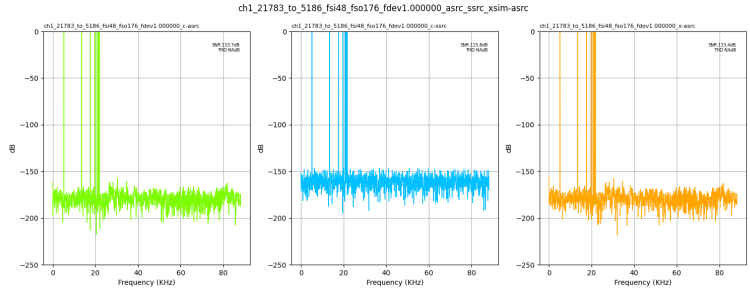


Fig. 145: Input Fs: 48,000Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

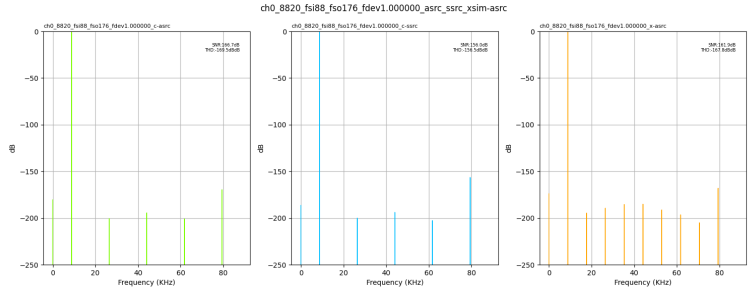


Fig. 146: Input Fs: 88,200Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

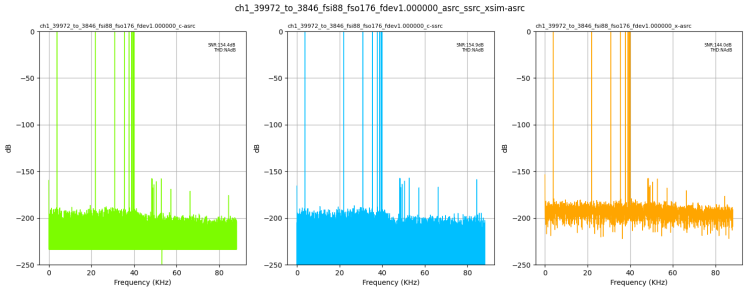


Fig. 147: Input Fs: 88,200Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

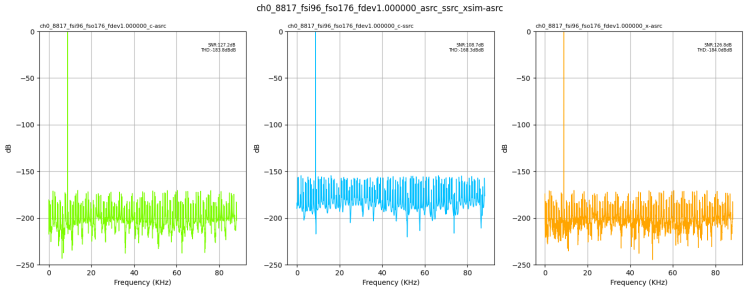


Fig. 148: Input Fs: 96,000Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

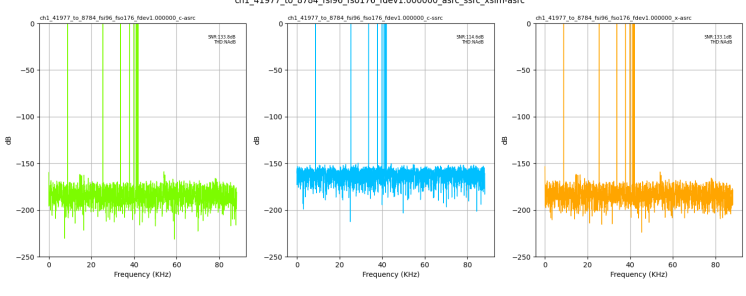


Fig. 149: Input Fs: 96,000Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

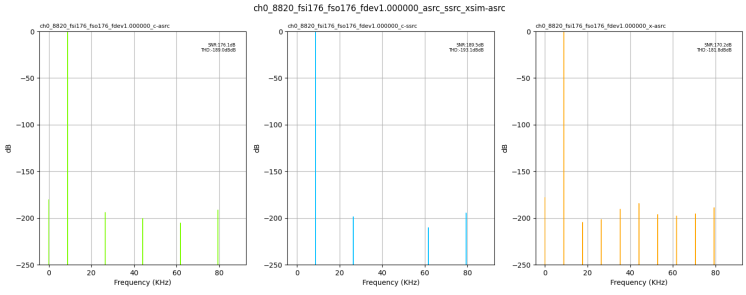


Fig. 150: Input Fs: 176,400Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

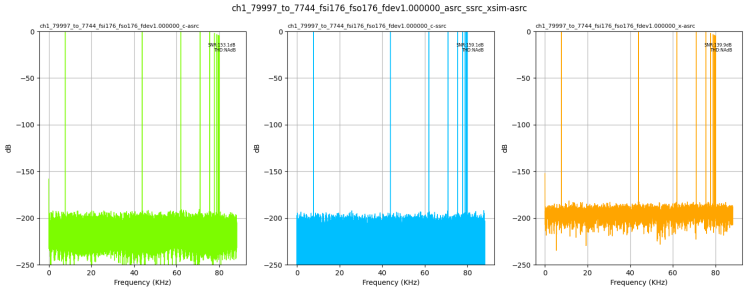


Fig. 151: Input Fs: 176,400Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

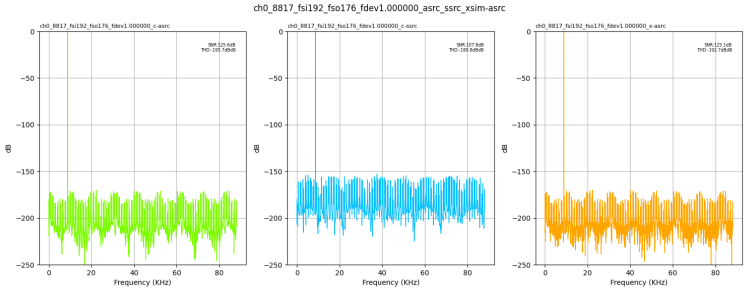


Fig. 152: Input Fs: 192,000Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

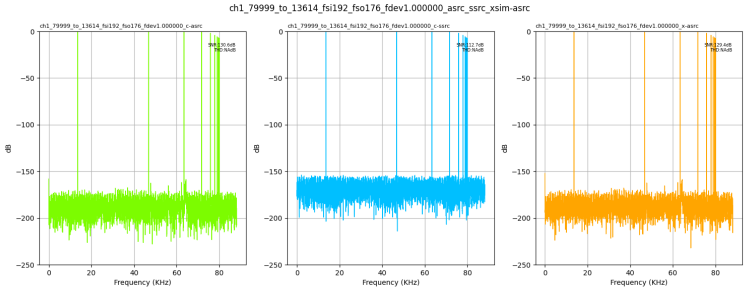


Fig. 153: Input Fs: 192,000Hz, Output Fs: 176,400Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

Output Fs : 192,000Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

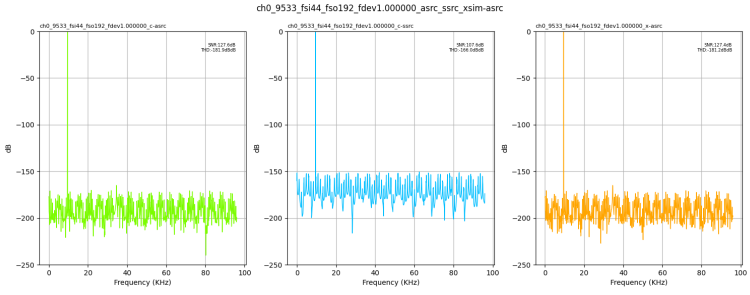


Fig. 154: Input Fs: 44,100Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

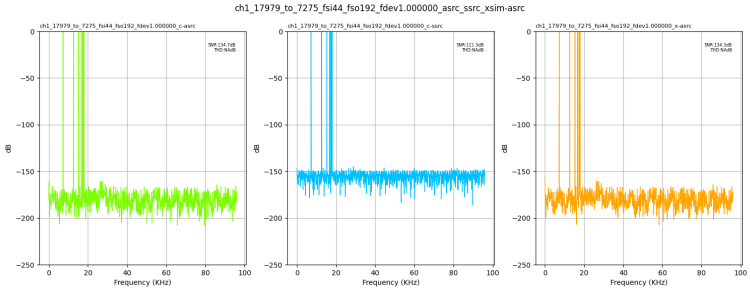


Fig. 155: Input Fs: 44,100Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

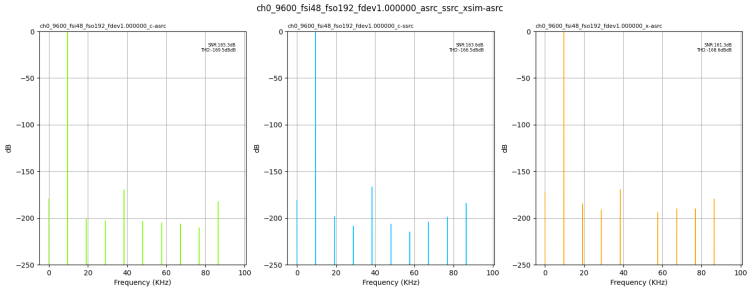


Fig. 156: Input Fs: 48,000Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

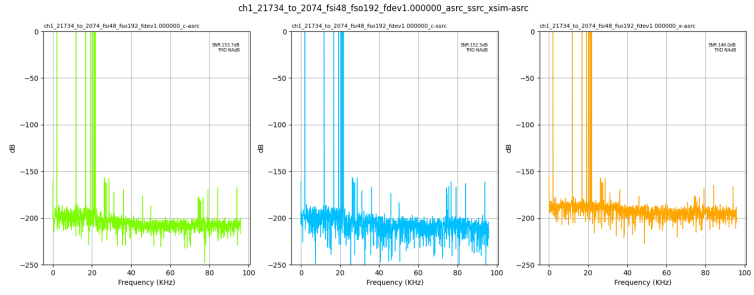


Fig. 157: Input Fs: 48,000Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

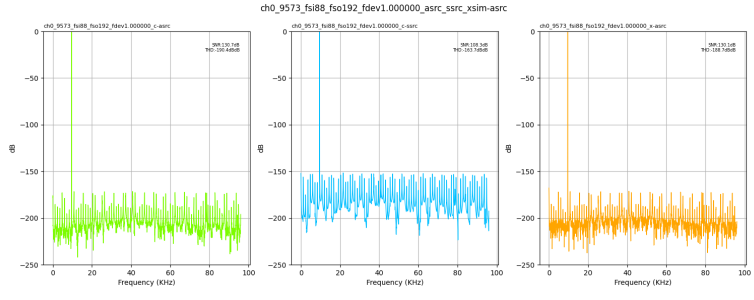


Fig. 158: Input Fs: 88,200Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

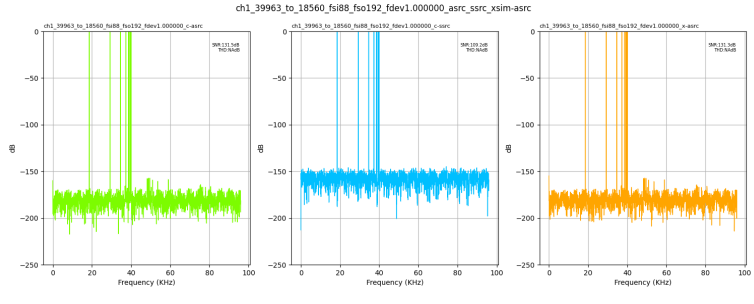


Fig. 159: Input Fs: 88,200Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

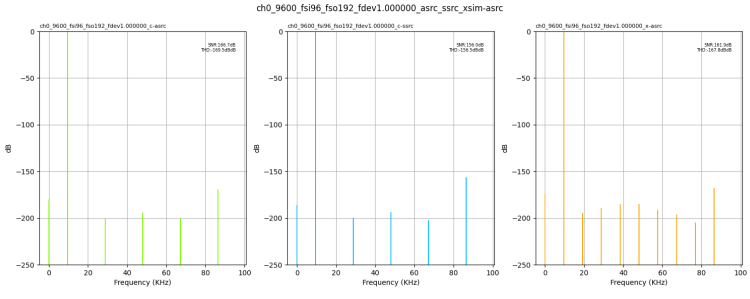


Fig. 160: Input Fs: 96,000Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

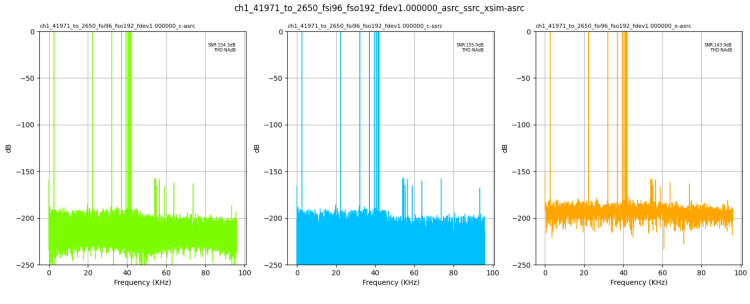


Fig. 161: Input Fs: 96,000Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

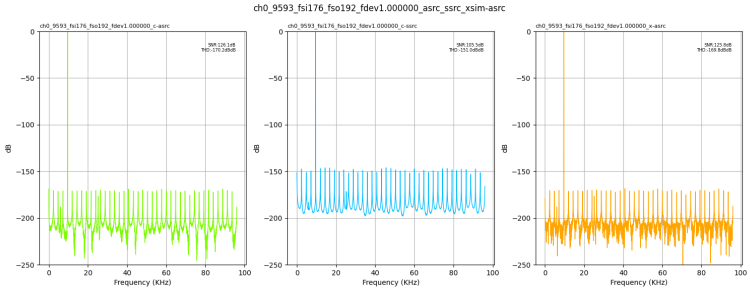


Fig. 162: Input Fs: 176,400Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

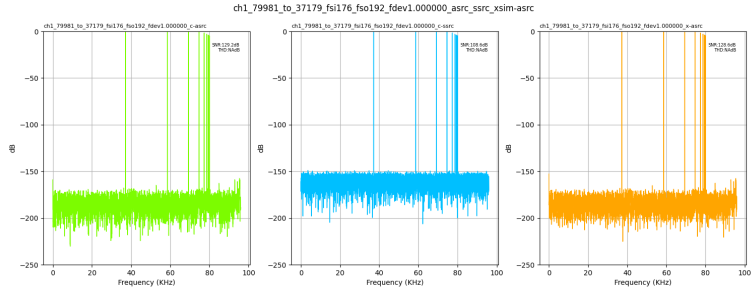


Fig. 163: Input Fs: 176,400Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

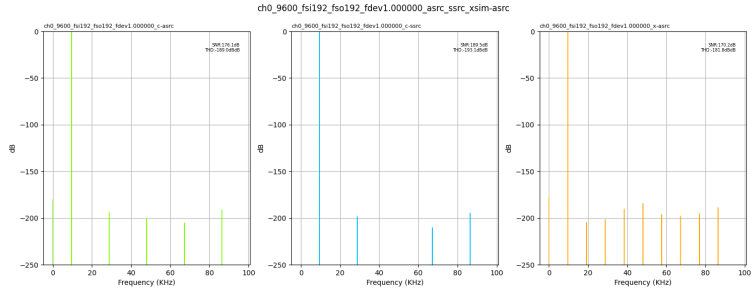


Fig. 164: Input Fs: 192,000Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

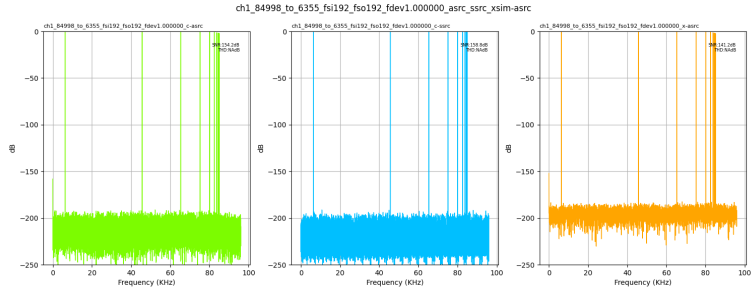


Fig. 165: Input Fs: 192,000Hz, Output Fs: 192,000Hz, Fs error: 1.000000, Results for: asrc, ssrc, xsim-asrc

8.1.3 Frequency error: 1.000100Hz

Output Fs : 16,000Hz

- No SRC available for this scenario.

Output Fs : 32,000Hz

- No SRC available for this scenario.

Output Fs : 44,100Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

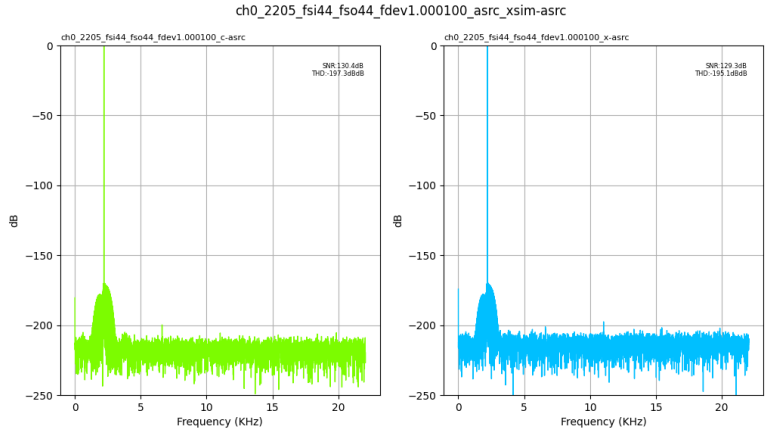


Fig. 166: Input Fs: 44,100Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

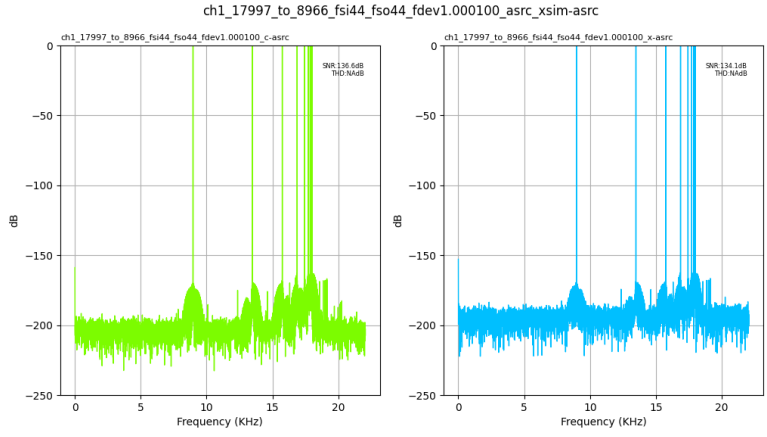


Fig. 167: Input Fs: 44,100Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

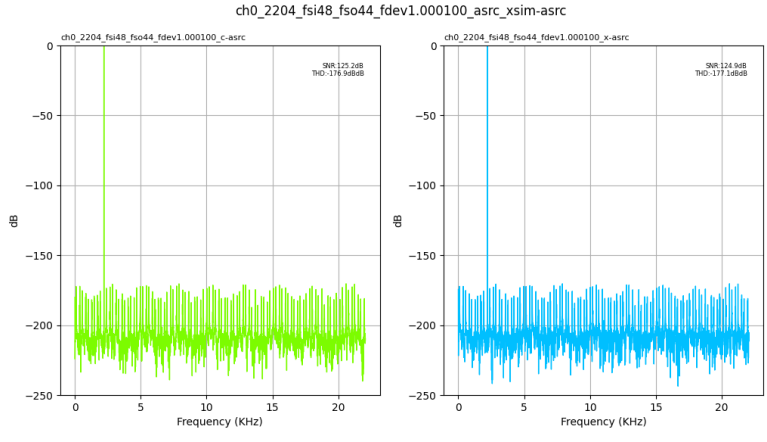


Fig. 168: Input Fs: 48,000Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

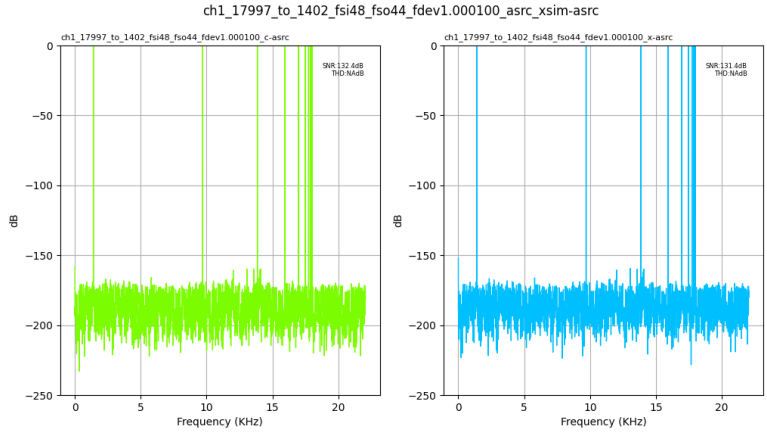


Fig. 169: Input Fs: 48,000Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

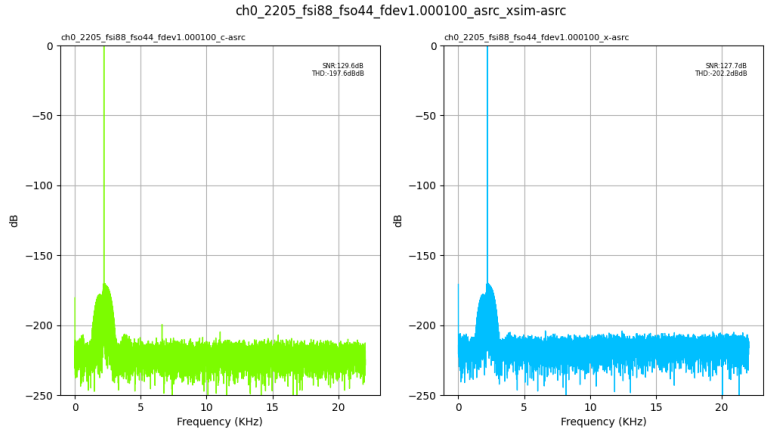


Fig. 170: Input Fs: 88,200Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

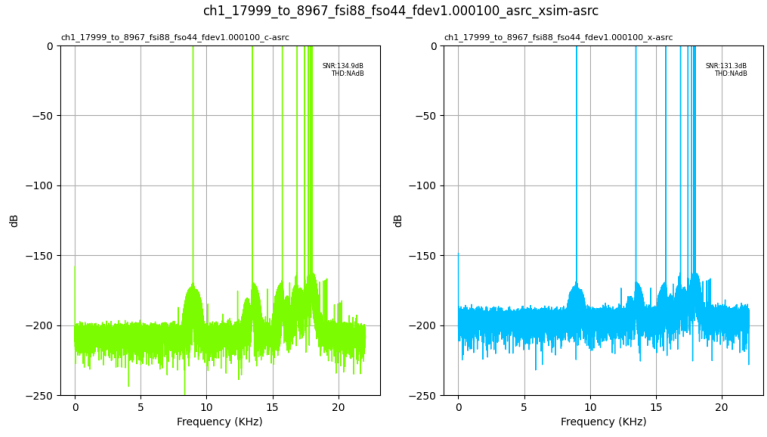


Fig. 171: Input Fs: 88,200Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

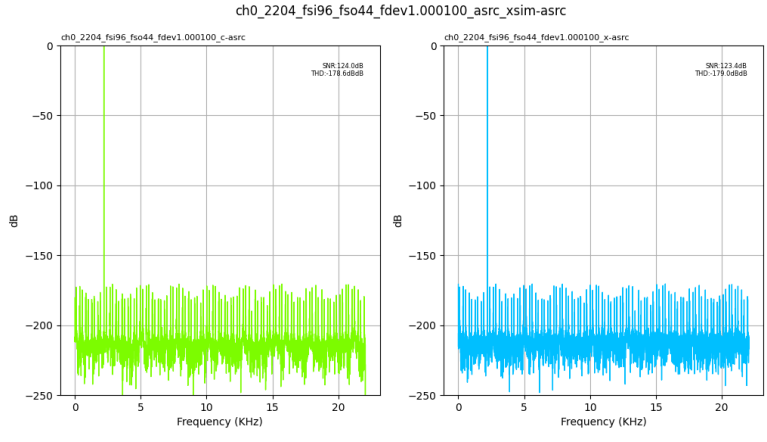


Fig. 172: Input Fs: 96,000Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

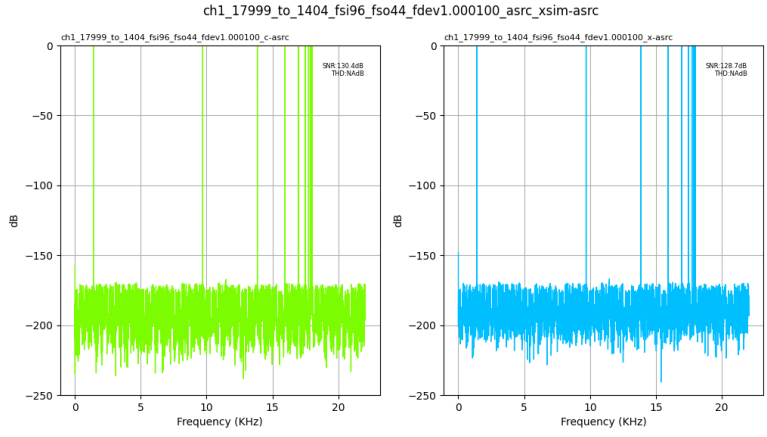


Fig. 173: Input Fs: 96,000Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

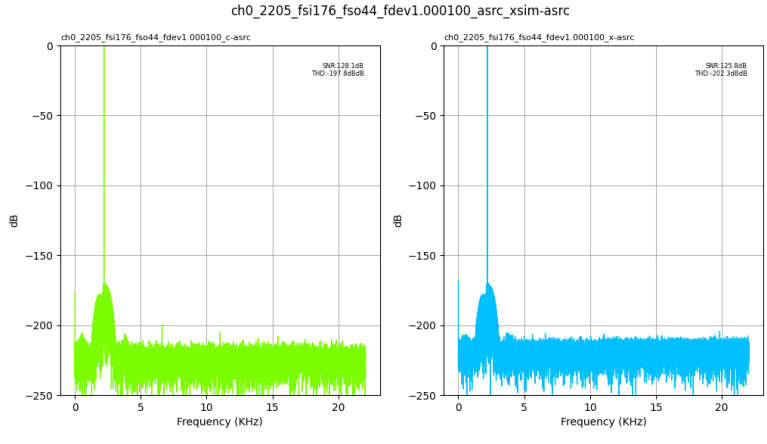


Fig. 174: Input Fs: 176,400Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

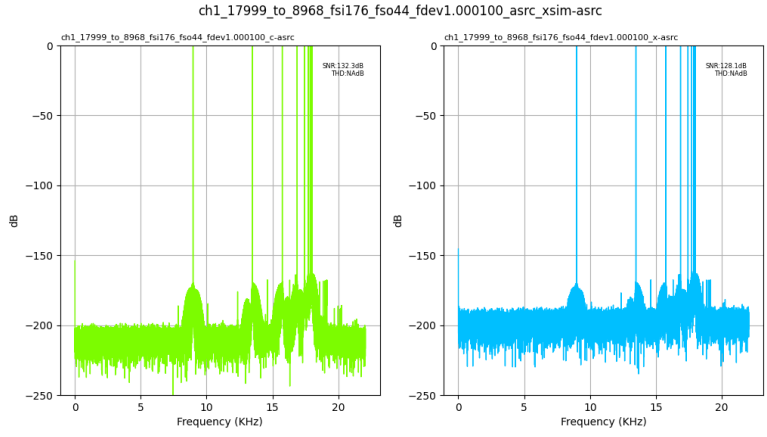


Fig. 175: Input Fs: 176,400Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

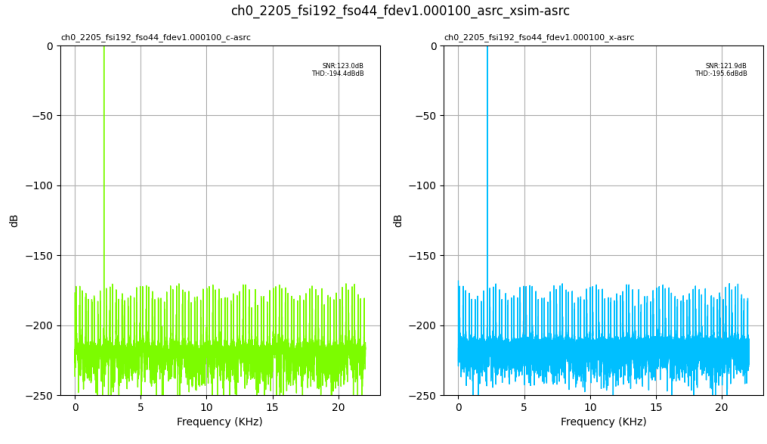


Fig. 176: Input Fs: 192,000Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

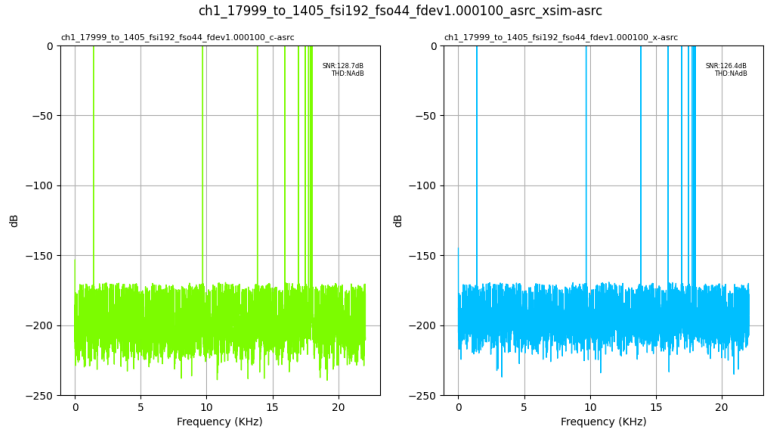


Fig. 177: Input Fs: 192,000Hz, Output Fs: 44,100Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

Output Fs : 48,000Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

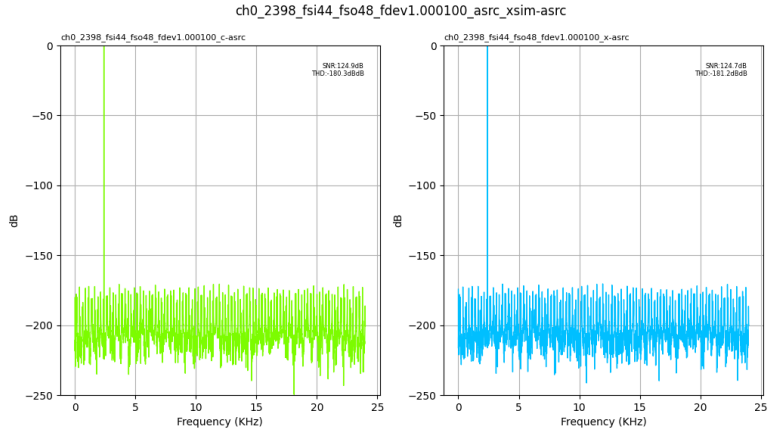


Fig. 178: Input Fs: 44,100Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

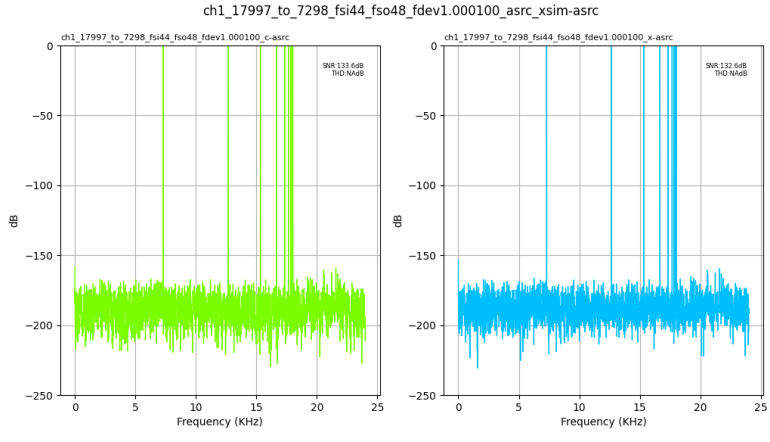


Fig. 179: Input Fs: 44,100Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

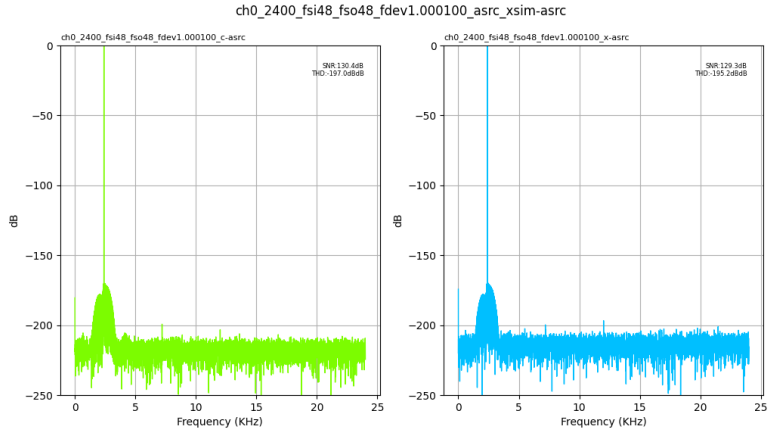


Fig. 180: Input Fs: 48,000Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

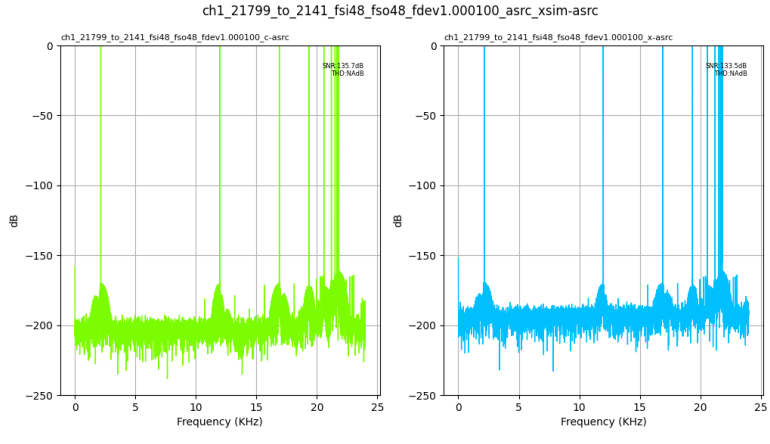


Fig. 181: Input Fs: 48,000Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

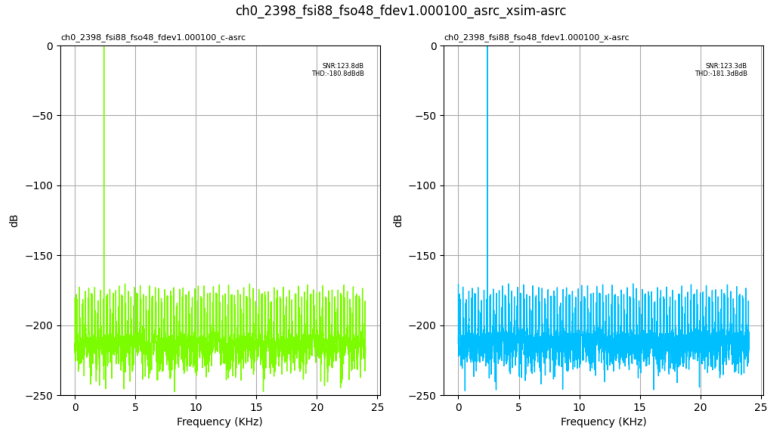


Fig. 182: Input Fs: 88,200Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

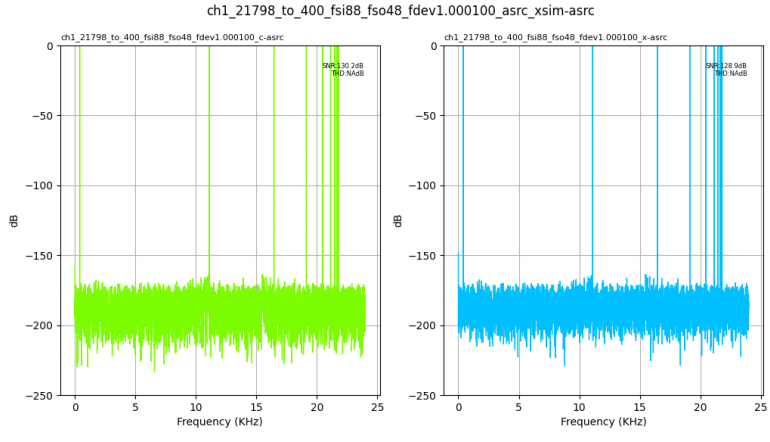


Fig. 183: Input Fs: 88,200Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

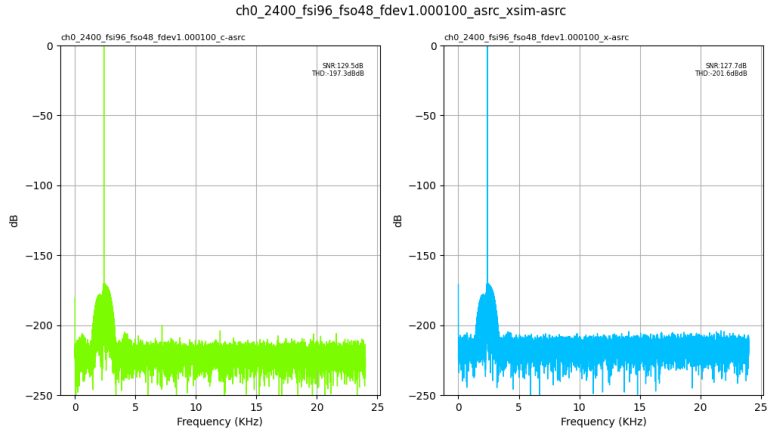


Fig. 184: Input Fs: 96,000Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

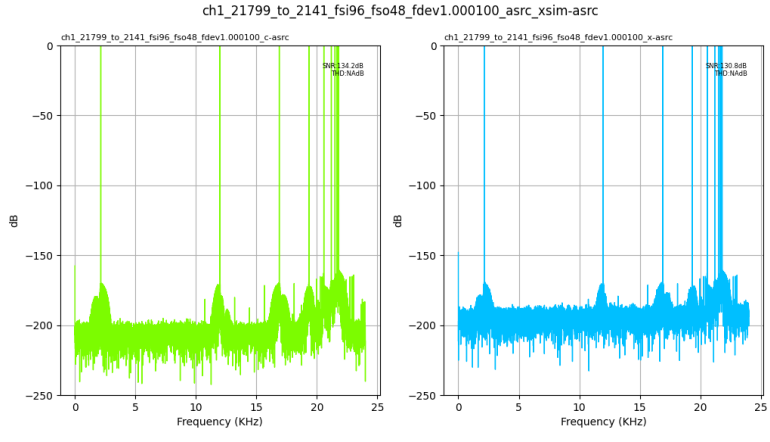


Fig. 185: Input Fs: 96,000Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

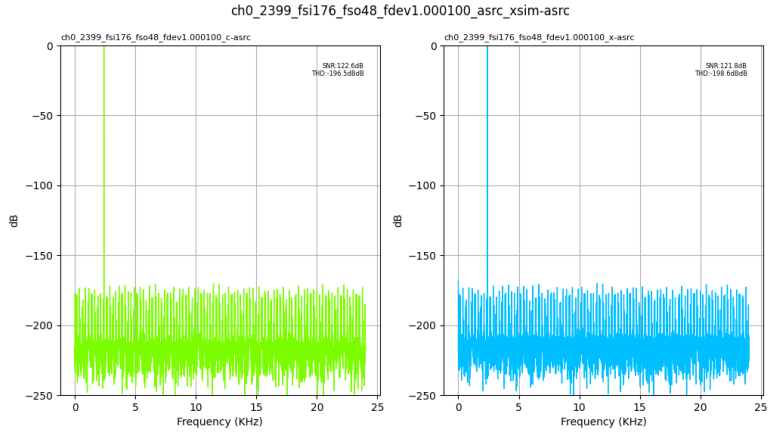


Fig. 186: Input Fs: 176,400Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

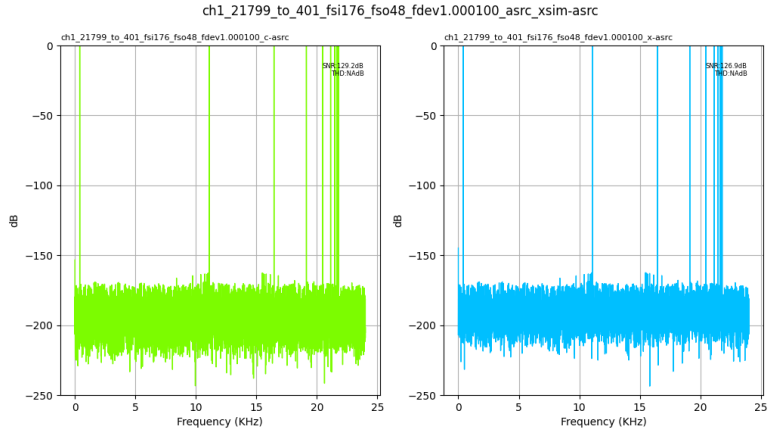


Fig. 187: Input Fs: 176,400Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

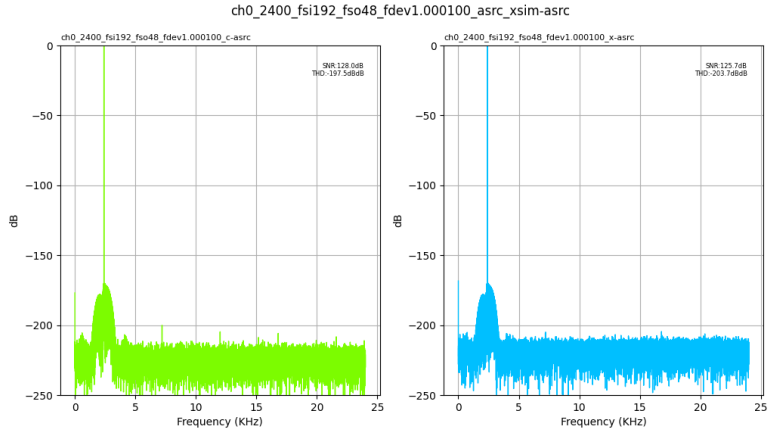


Fig. 188: Input Fs: 192,000Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

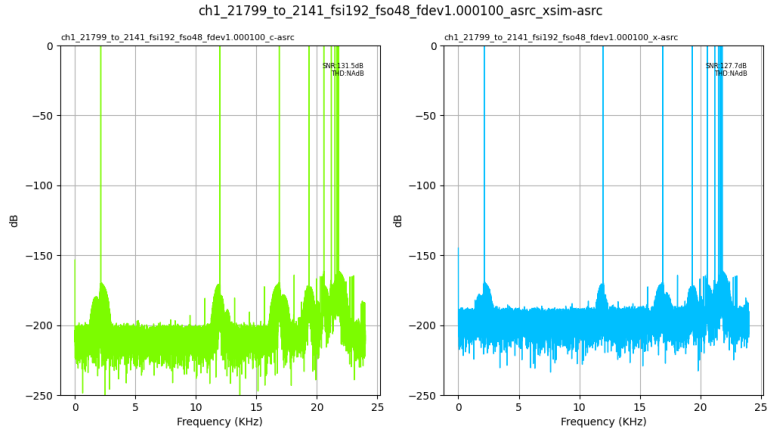


Fig. 189: Input Fs: 192,000Hz, Output Fs: 48,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

Output Fs : 88,200Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

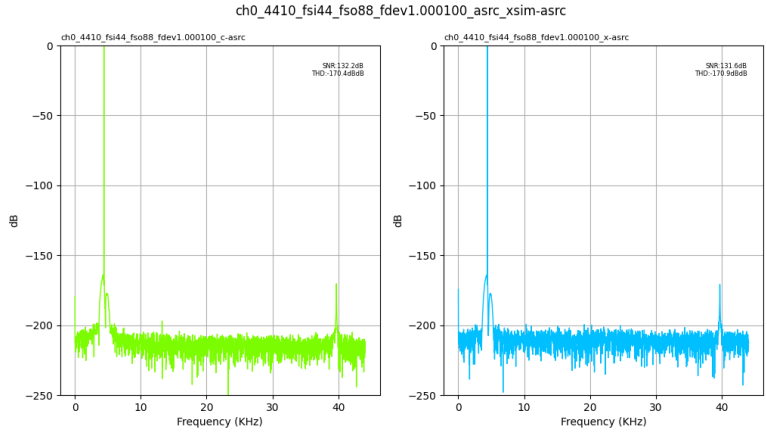


Fig. 190: Input Fs: 44,100Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

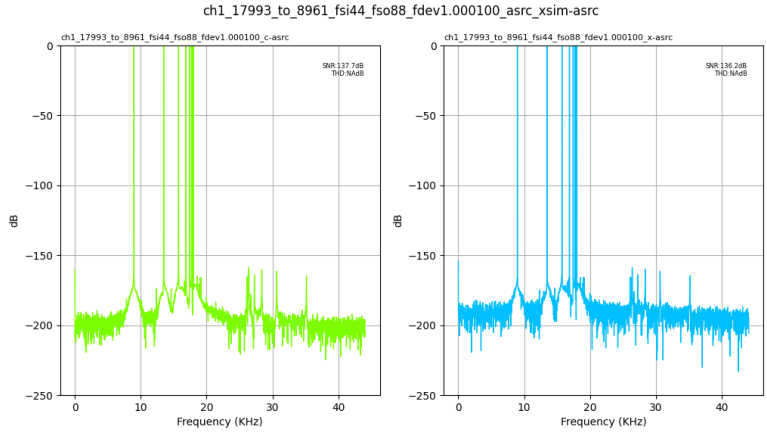


Fig. 191: Input Fs: 44,100Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

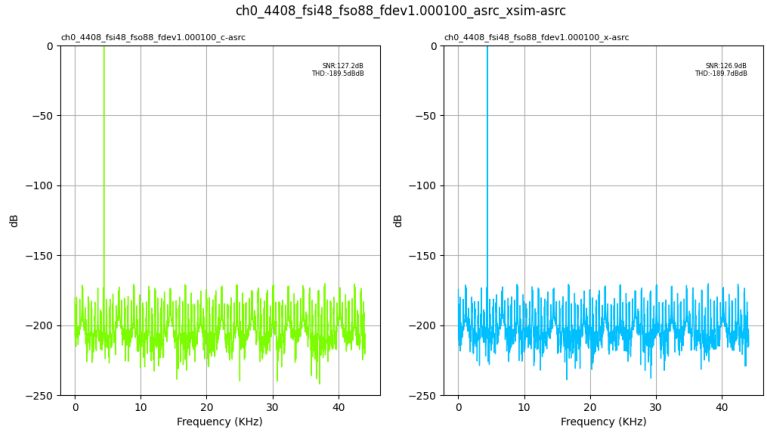


Fig. 192: Input Fs: 48,000Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

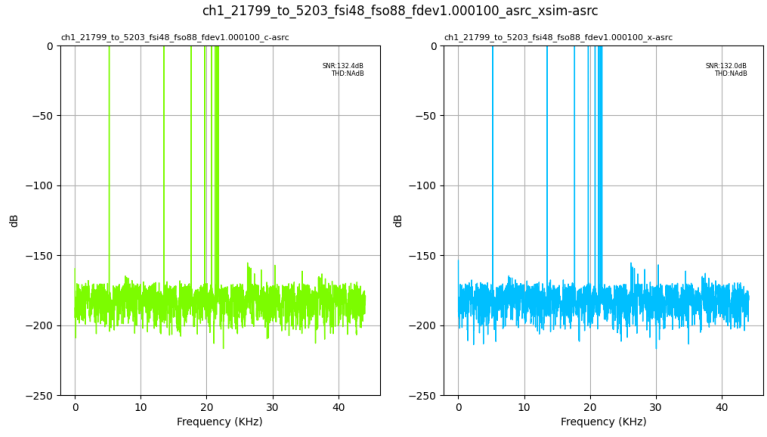


Fig. 193: Input Fs: 48,000Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

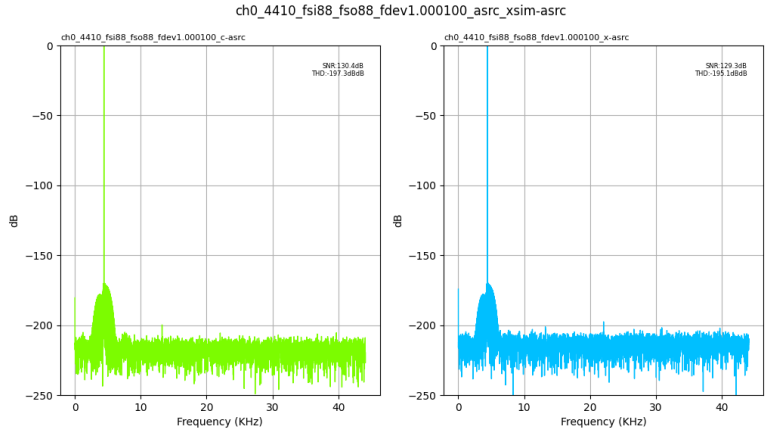


Fig. 194: Input Fs: 88,200Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

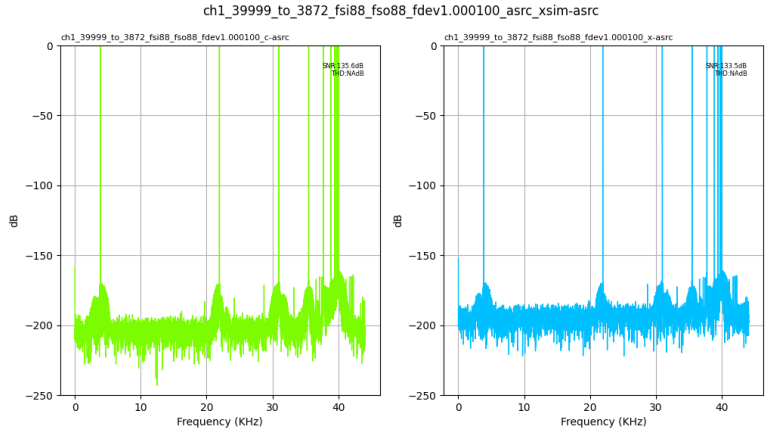


Fig. 195: Input Fs: 88,200Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

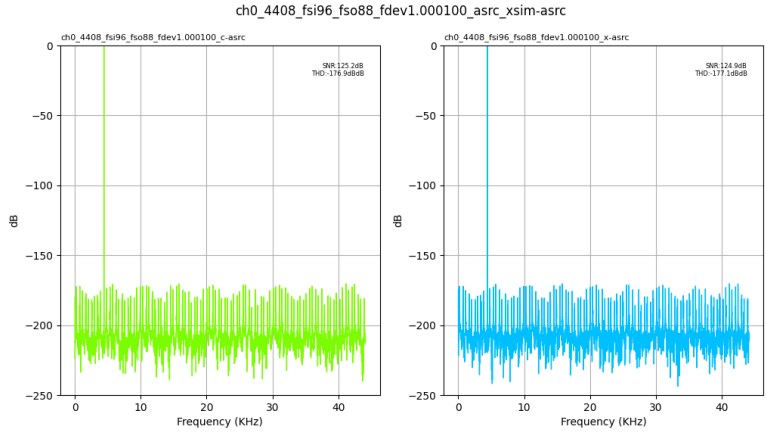


Fig. 196: Input Fs: 96,000Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

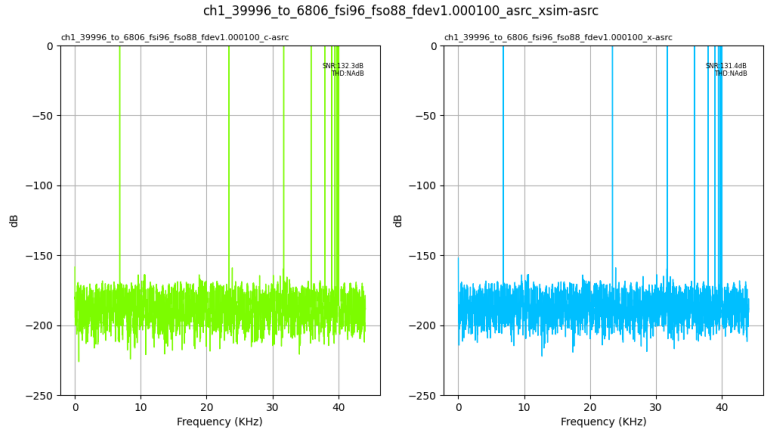


Fig. 197: Input Fs: 96,000Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

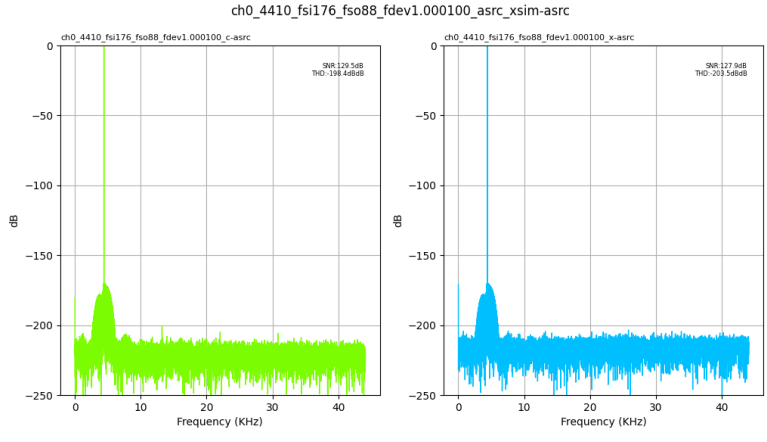


Fig. 198: Input Fs: 176,400Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

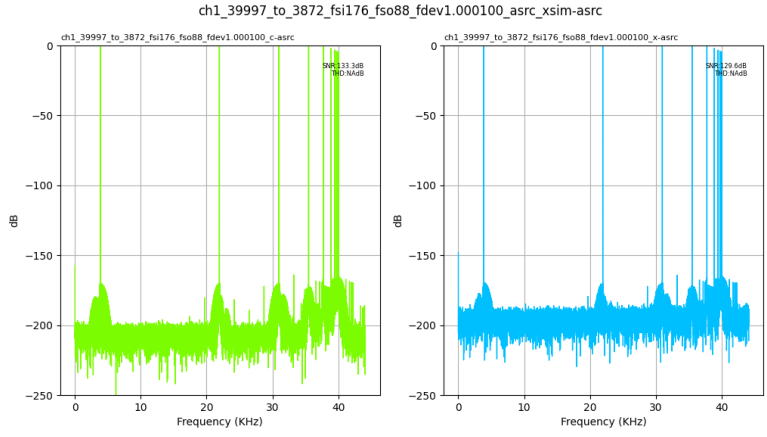


Fig. 199: Input Fs: 176,400Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

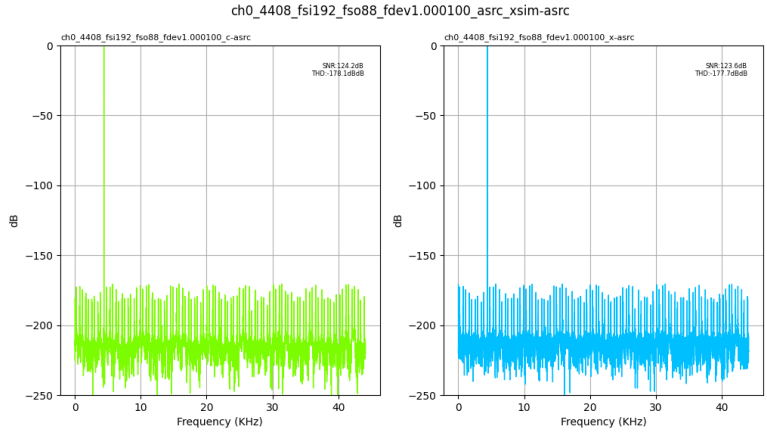


Fig. 200: Input Fs: 192,000Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

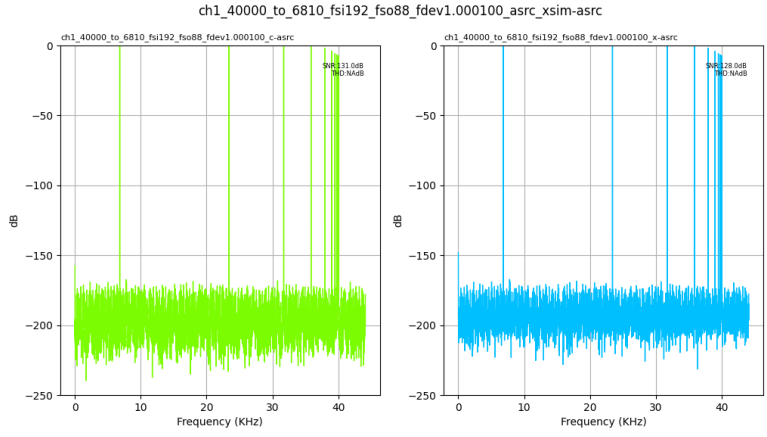


Fig. 201: Input Fs: 192,000Hz, Output Fs: 88,200Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

Output Fs : 96,000Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

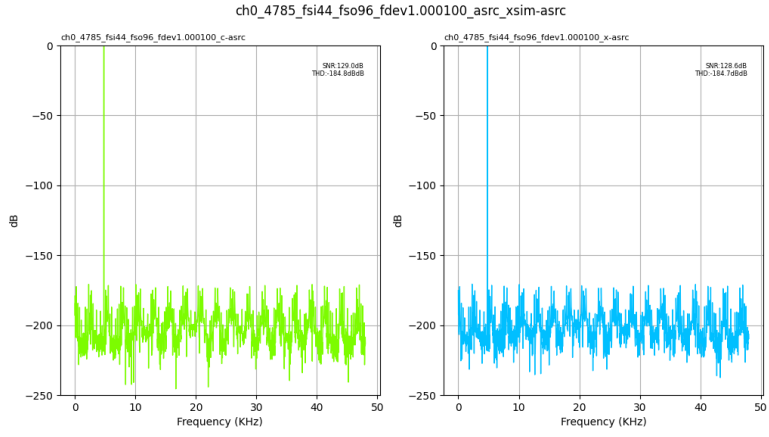


Fig. 202: Input Fs: 44,100Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

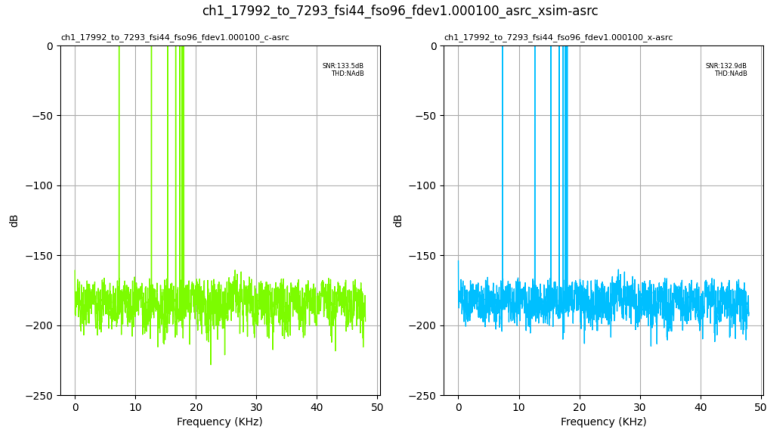


Fig. 203: Input Fs: 44,100Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

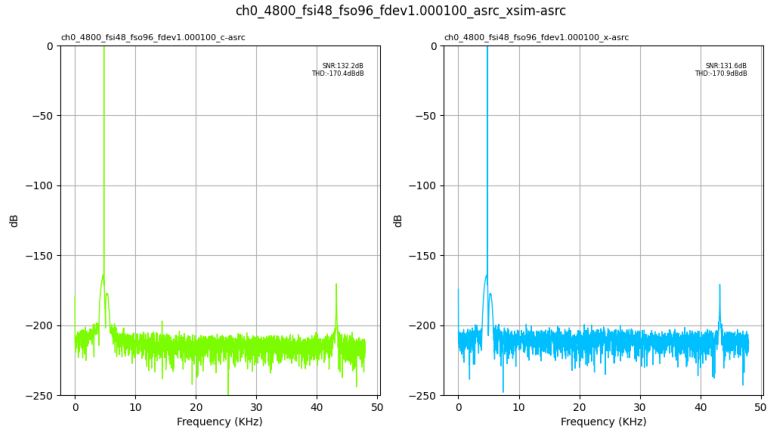


Fig. 204: Input Fs: 48,000Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

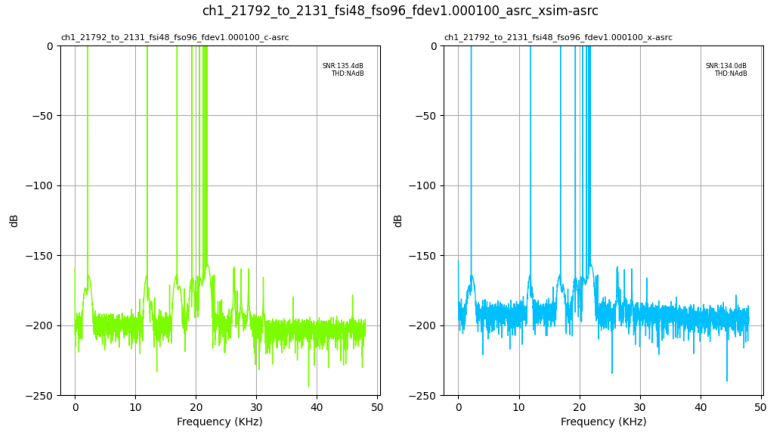


Fig. 205: Input Fs: 48,000Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

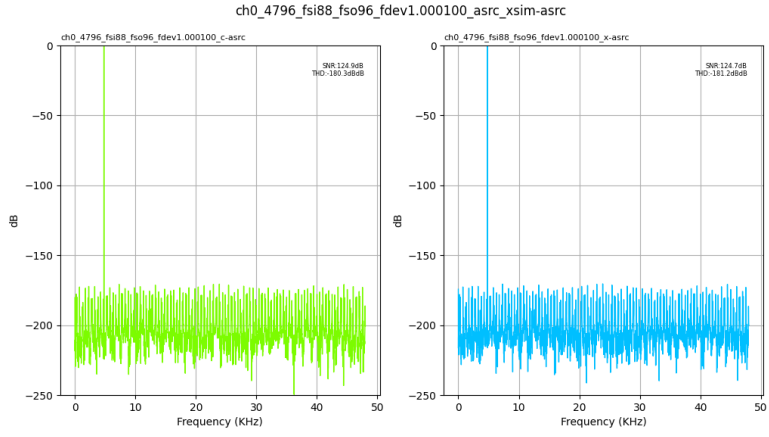


Fig. 206: Input Fs: 88,200Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

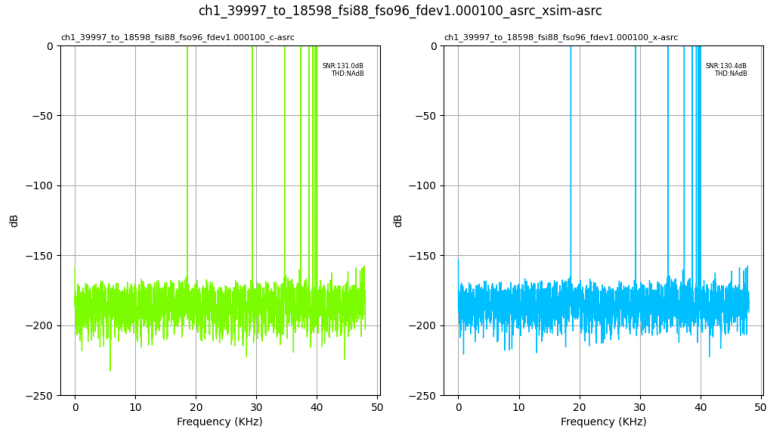


Fig. 207: Input Fs: 88,200Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

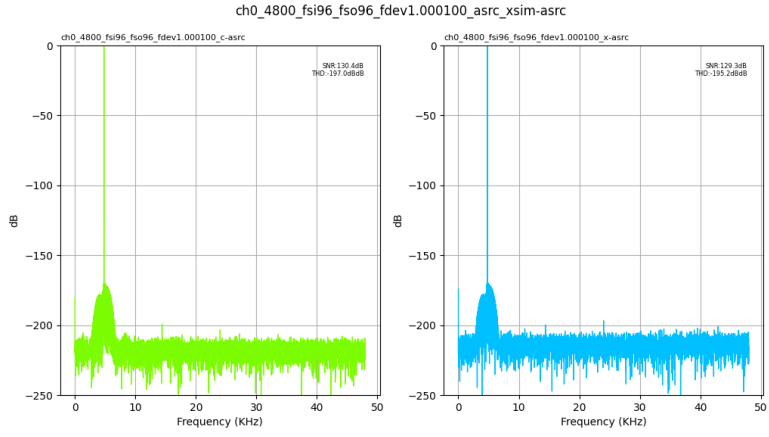


Fig. 208: Input Fs: 96,000Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

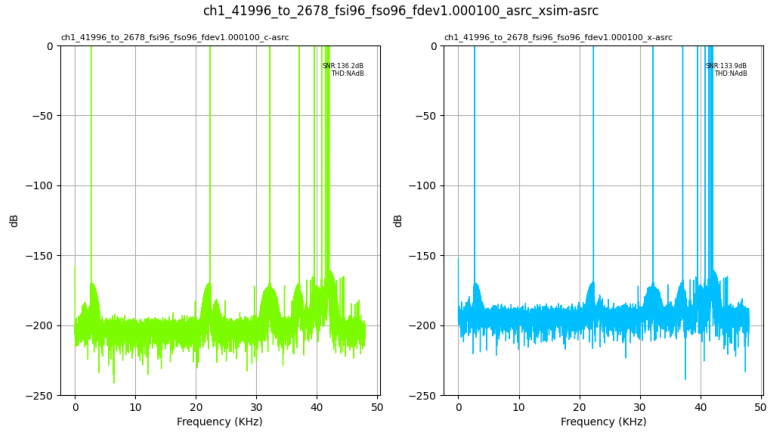


Fig. 209: Input Fs: 96,000Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

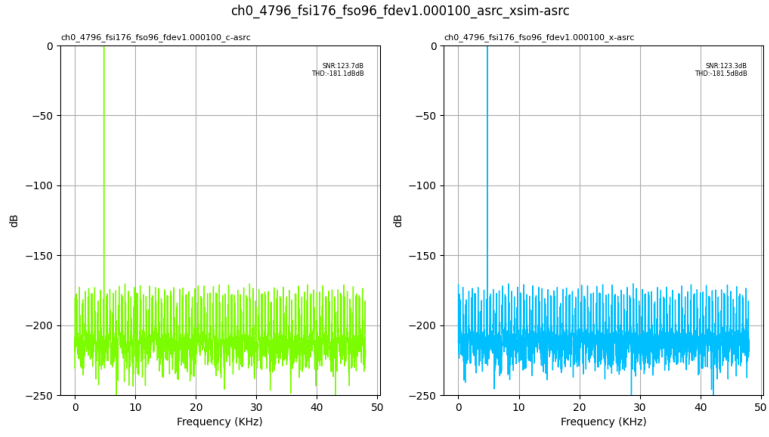


Fig. 210: Input Fs: 176,400Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

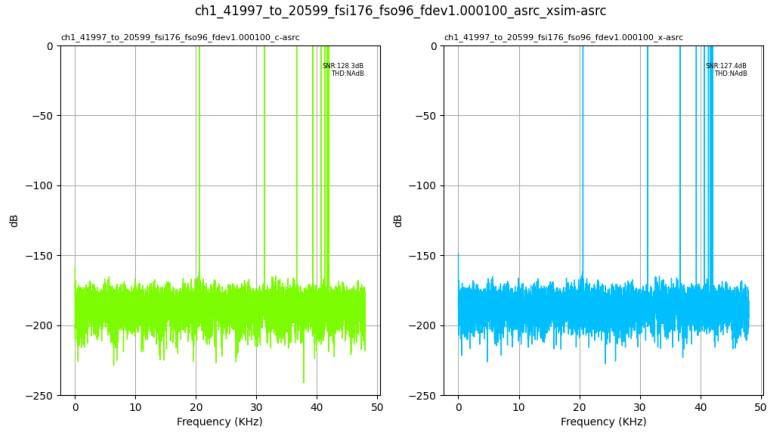


Fig. 211: Input Fs: 176,400Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

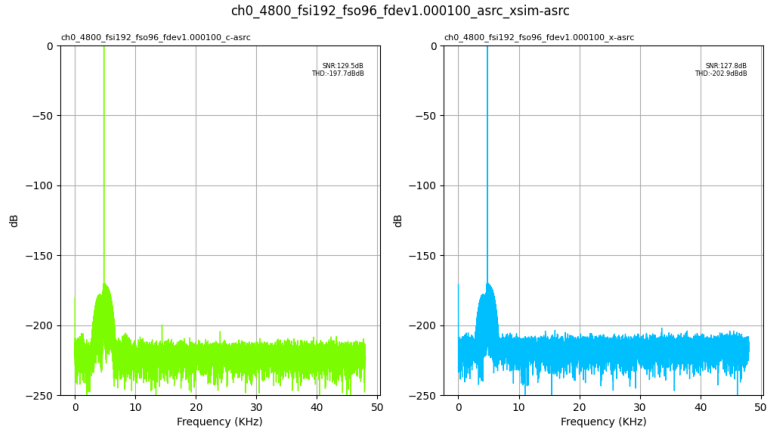


Fig. 212: Input Fs: 192,000Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

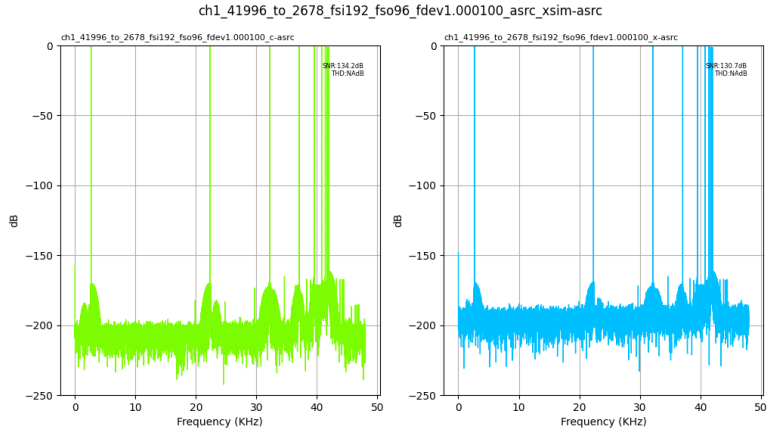


Fig. 213: Input Fs: 192,000Hz, Output Fs: 96,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

Output Fs : 176,400Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

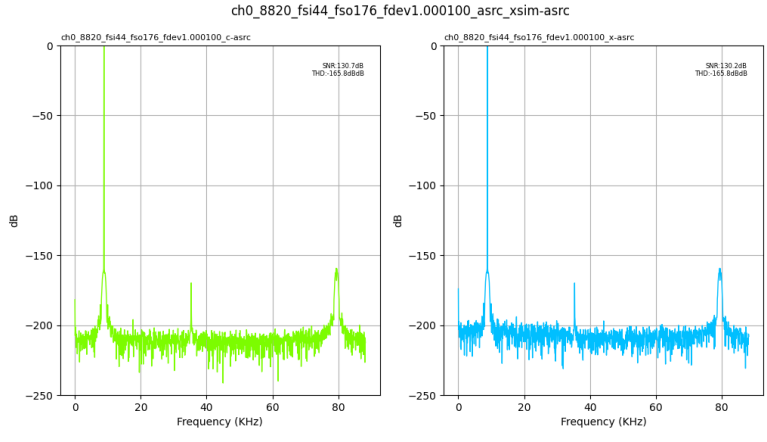


Fig. 214: Input Fs: 44,100Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

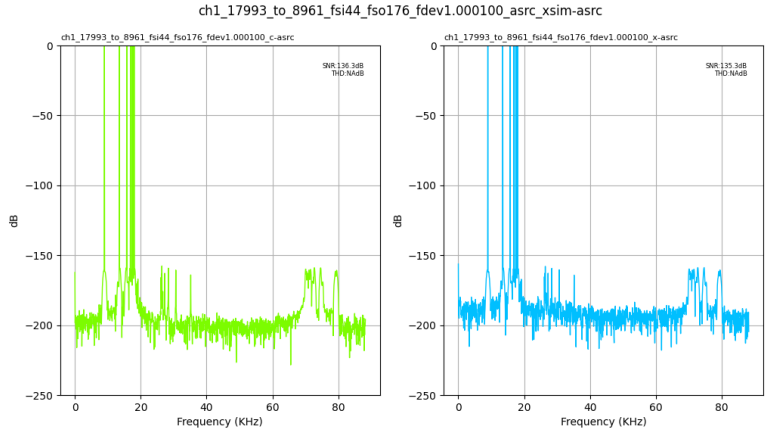


Fig. 215: Input Fs: 44,100Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

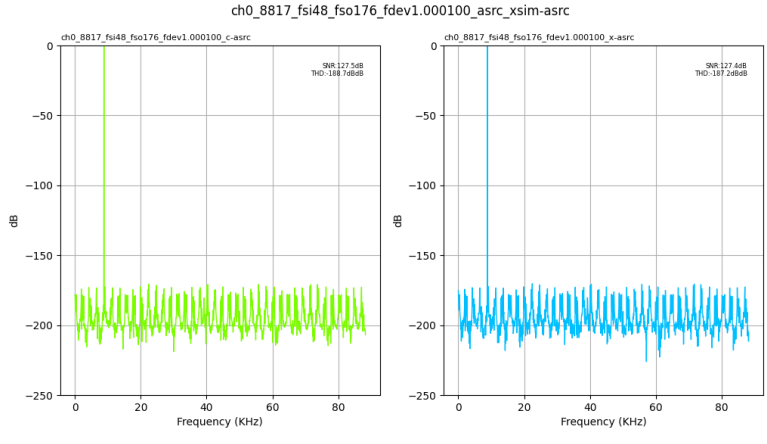


Fig. 216: Input Fs: 48,000Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

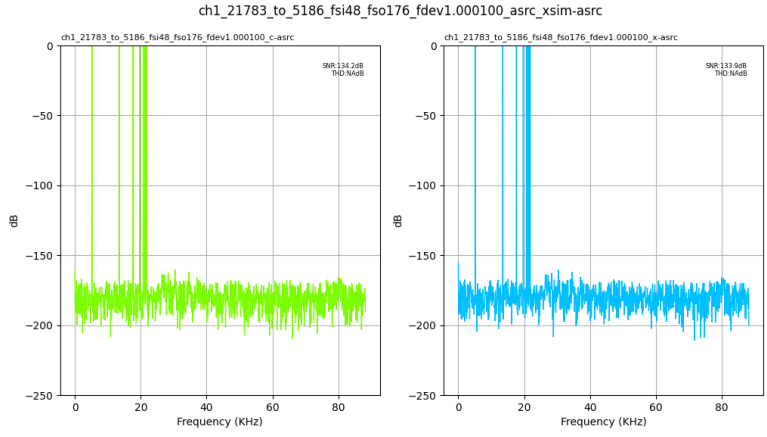


Fig. 217: Input Fs: 48,000Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

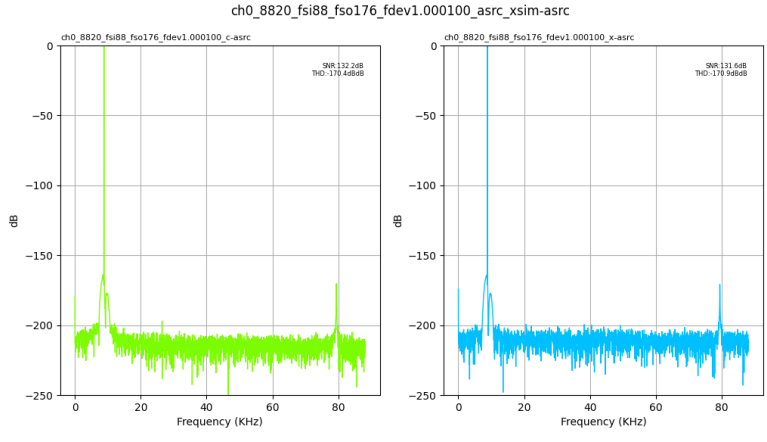


Fig. 218: Input Fs: 88,200Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

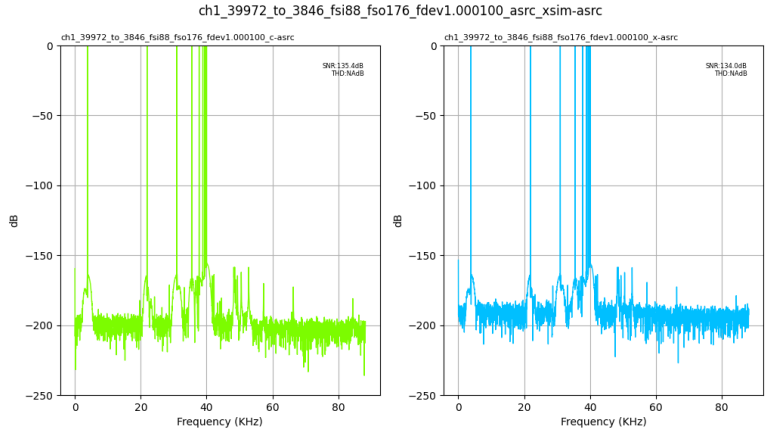


Fig. 219: Input Fs: 88,200Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

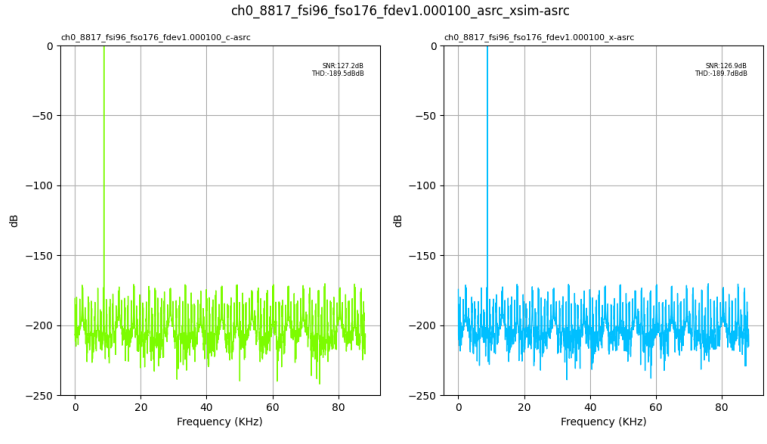


Fig. 220: Input Fs: 96,000Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

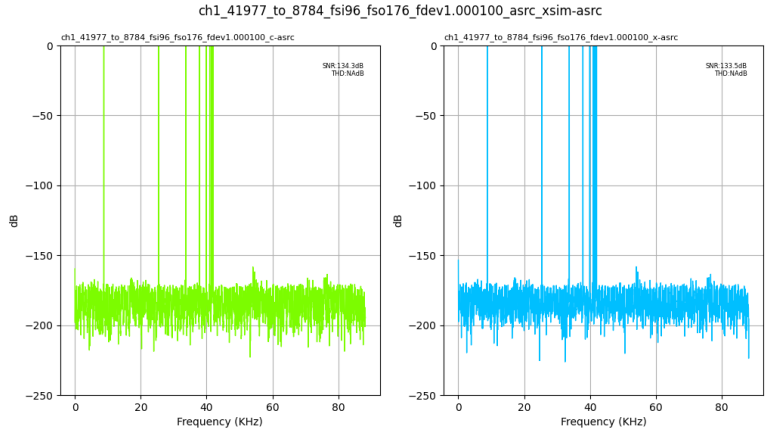


Fig. 221: Input Fs: 96,000Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

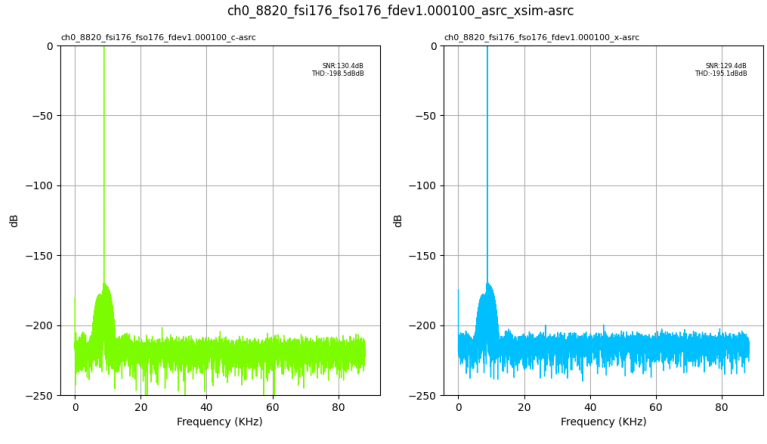


Fig. 222: Input Fs: 176,400Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

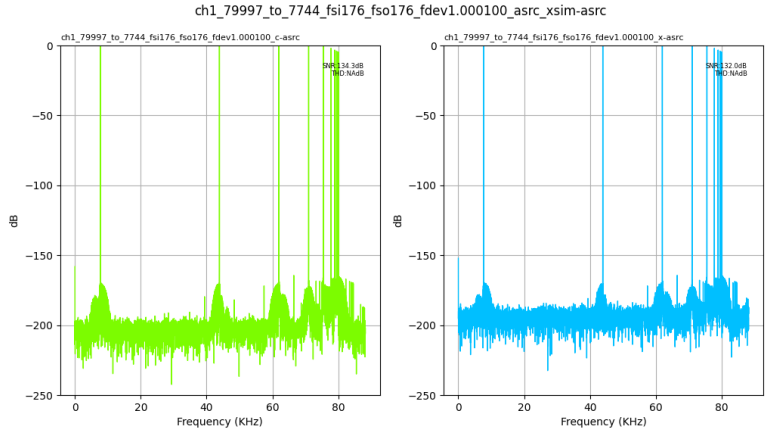


Fig. 223: Input Fs: 176,400Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

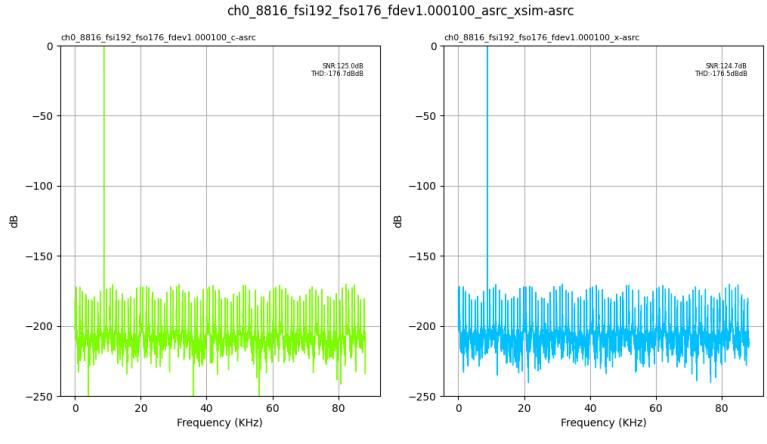


Fig. 224: Input Fs: 192,000Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

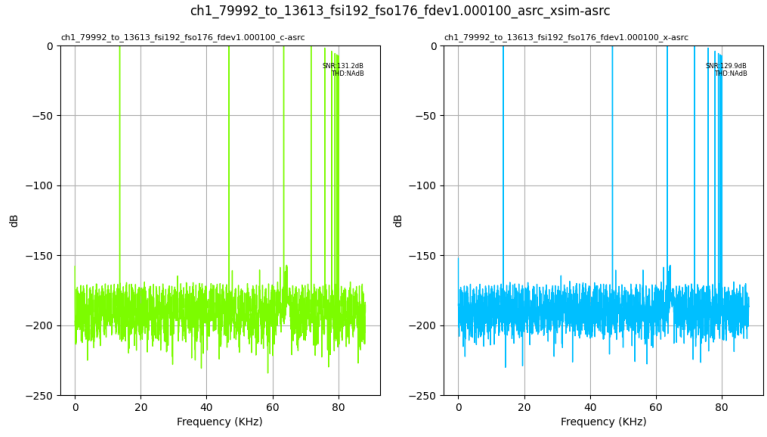


Fig. 225: Input Fs: 192,000Hz, Output Fs: 176,400Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

Output Fs : 192,000Hz

- *Input Fs: 44,100Hz, channel 0*
- *Input Fs: 44,100Hz, channel 1*
- *Input Fs: 48,000Hz, channel 0*
- *Input Fs: 48,000Hz, channel 1*
- *Input Fs: 88,200Hz, channel 0*
- *Input Fs: 88,200Hz, channel 1*
- *Input Fs: 96,000Hz, channel 0*
- *Input Fs: 96,000Hz, channel 1*
- *Input Fs: 176,400Hz, channel 0*
- *Input Fs: 176,400Hz, channel 1*
- *Input Fs: 192,000Hz, channel 0*
- *Input Fs: 192,000Hz, channel 1*

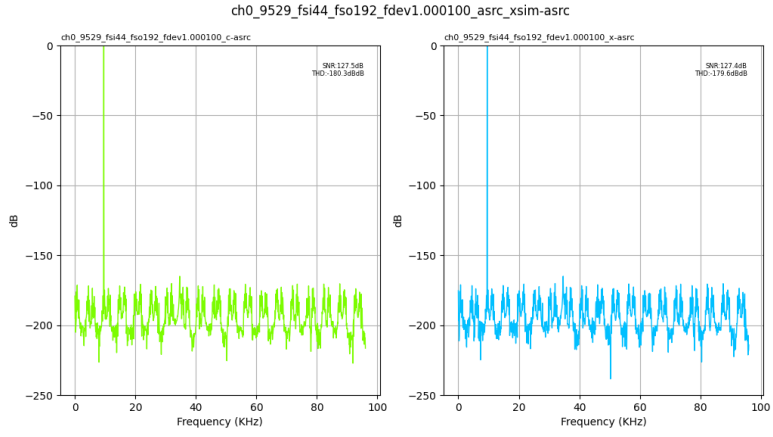


Fig. 226: Input Fs: 44,100Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

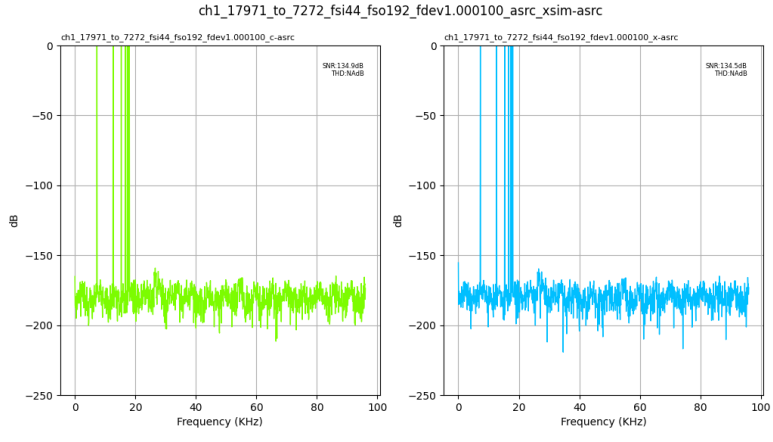


Fig. 227: Input Fs: 44,100Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

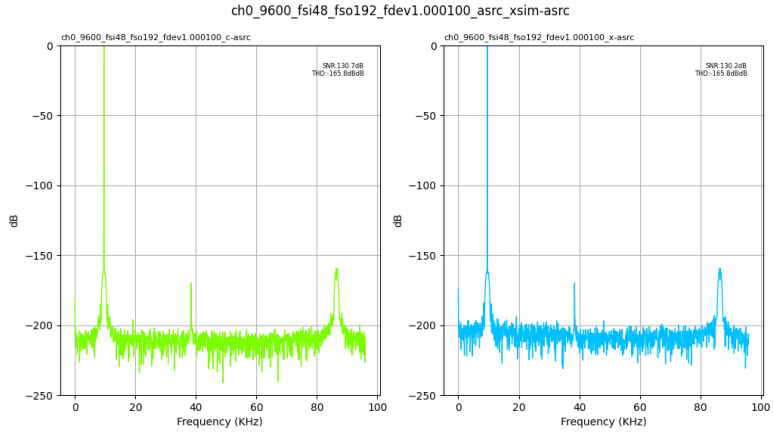


Fig. 228: Input Fs: 48,000Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

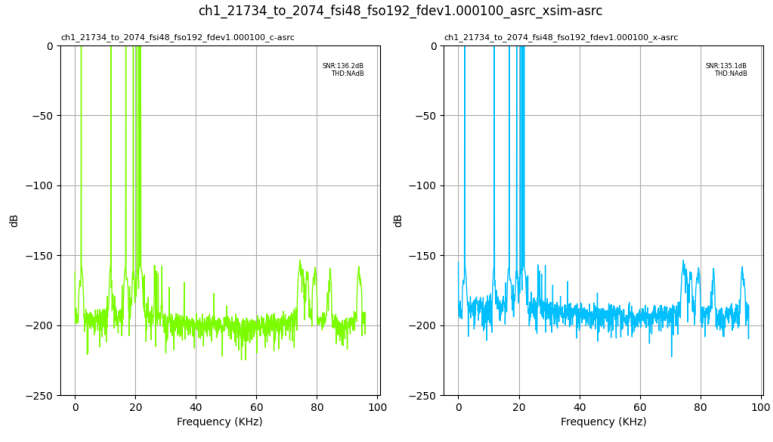


Fig. 229: Input Fs: 48,000Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

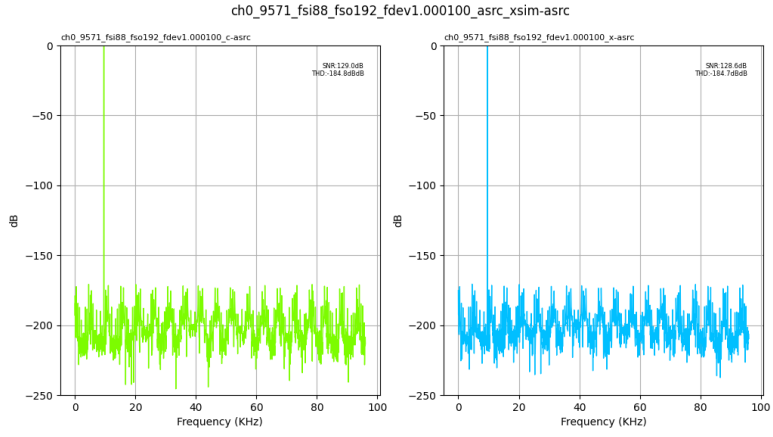


Fig. 230: Input Fs: 88,200Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

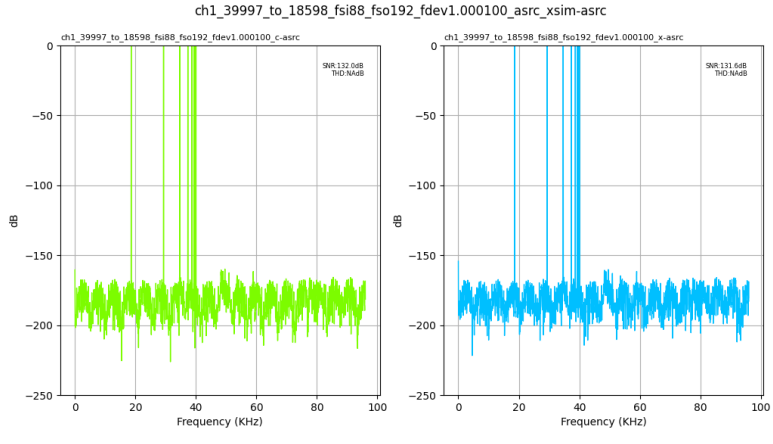


Fig. 231: Input Fs: 88,200Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

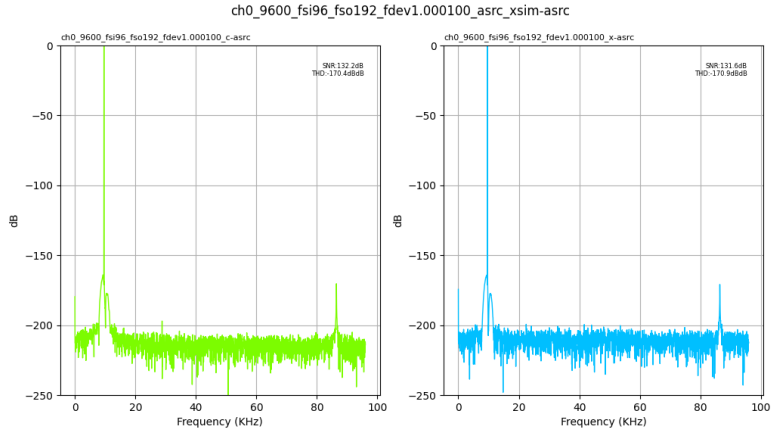


Fig. 232: Input Fs: 96,000Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

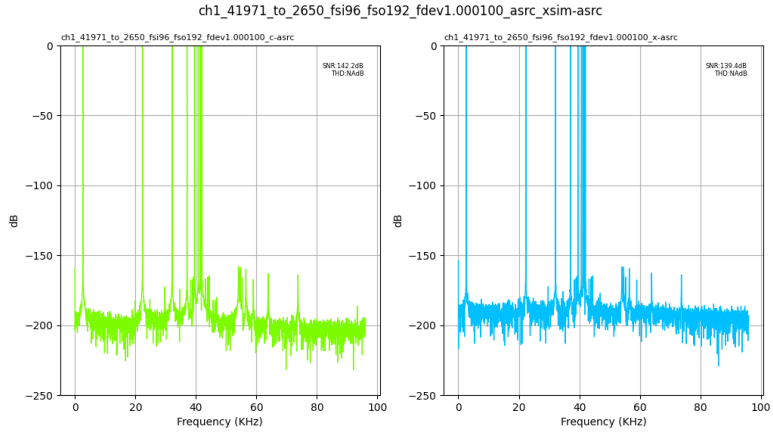


Fig. 233: Input Fs: 96,000Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

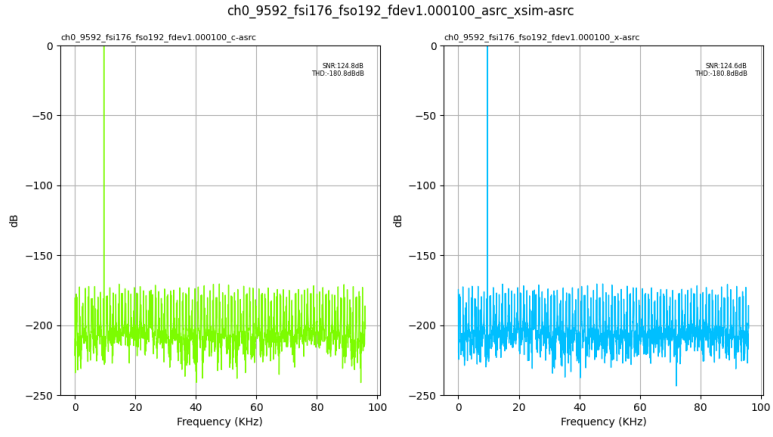


Fig. 234: Input Fs: 176,400Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

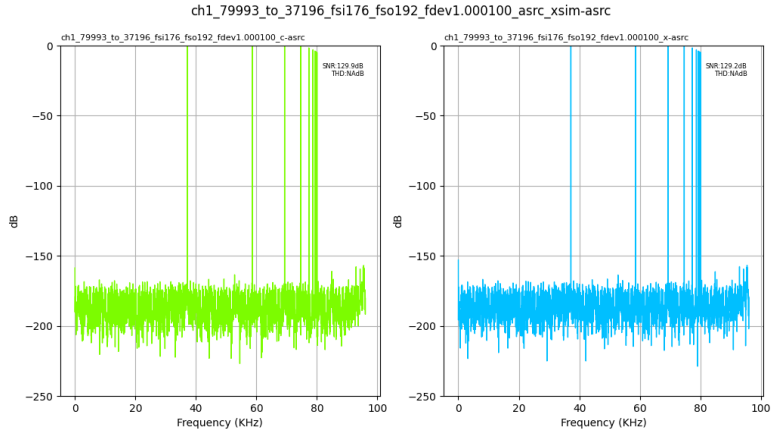


Fig. 235: Input Fs: 176,400Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

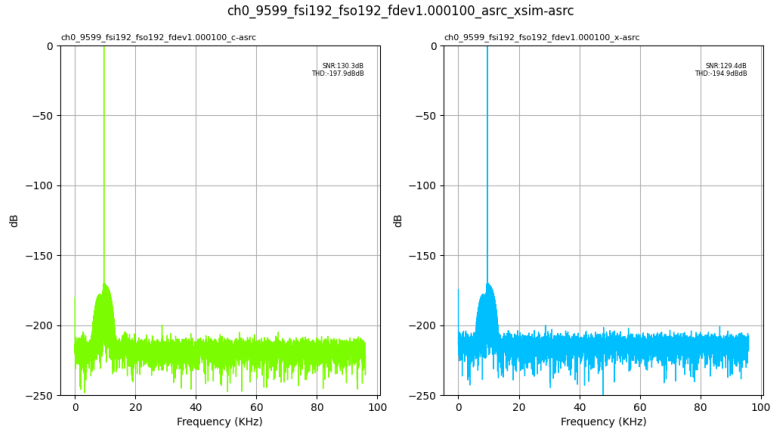


Fig. 236: Input Fs: 192,000Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

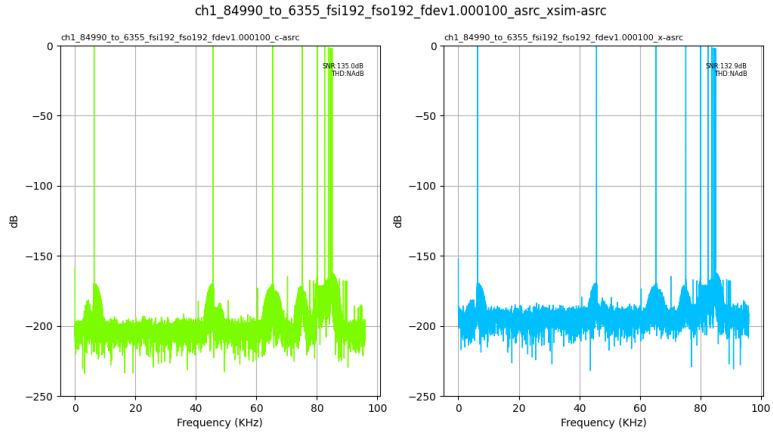


Fig. 237: Input Fs: 192,000Hz, Output Fs: 192,000Hz, Fs error: 1.000100, Results for: asrc, xsim-asrc

8.2 Summary table

Table 9: Data table

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
asrc	44100	44100	0.9999	0	2201.030	130.4	-197.2dB
xsim-asrc	44100	44100	0.9999	0	2201.030	129.4	-196.4dB
asrc	44100	44100	0.9999	1	8967.323-18000.810	136.1	NA
xsim-asrc	44100	44100	0.9999	1	8967.323-18000.810	133.8	NA
asrc	48000	44100	0.9999	0	2204.613	125.1	-189.7dB
xsim-asrc	48000	44100	0.9999	0	2204.613	124.8	-187.9dB
asrc	48000	44100	0.9999	1	1402.198-18001.635	131.5	NA
xsim-asrc	48000	44100	0.9999	1	1402.198-18001.635	130.8	NA
asrc	88200	44100	0.9999	0	2203.236	129.5	-198.1dB
xsim-asrc	88200	44100	0.9999	0	2203.236	127.7	-201.4dB
asrc	88200	44100	0.9999	1	8967.323-18000.810	135.5	NA
xsim-asrc	88200	44100	0.9999	1	8967.323-18000.810	131.5	NA
asrc	96000	44100	0.9999	0	2204.613	124.2	-193.8dB
xsim-asrc	96000	44100	0.9999	0	2204.613	123.5	-195.2dB
asrc	96000	44100	0.9999	1	1402.198-18001.635	130.0	NA
xsim-asrc	96000	44100	0.9999	1	1402.198-18001.635	128.6	NA
asrc	176400	44100	0.9999	0	2204.338	128.2	-197.5dB
xsim-asrc	176400	44100	0.9999	0	2204.338	125.8	-202.2dB
asrc	176400	44100	0.9999	1	8967.323-18000.810	134.0	NA
xsim-asrc	176400	44100	0.9999	1	8967.323-18000.810	129.0	NA
asrc	192000	44100	0.9999	0	2204.562	123.0	-196.2dB
xsim-asrc	192000	44100	0.9999	0	2204.562	121.9	-200.3dB
asrc	192000	44100	0.9999	1	1402.166-18001.221	128.6	NA
xsim-asrc	192000	44100	0.9999	1	1402.166-18001.221	126.3	NA
asrc	44100	48000	0.9999	0	2398.672	124.9	-189.8dB

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
xsim-asrc	44100	48000	0.9999	0	2398.672	124.6	-190.4dB
asrc	44100	48000	0.9999	1	7295.308-17997.881	132.6	NA
xsim-asrc	44100	48000	0.9999	1	7295.308-17997.881	131.9	NA
asrc	48000	48000	0.9999	0	2395.679	130.4	-197.2dB
xsim-asrc	48000	48000	0.9999	0	2395.679	129.4	-196.4dB
asrc	48000	48000	0.9999	1	2136.427-21801.160	135.1	NA
xsim-asrc	48000	48000	0.9999	1	2136.427-21801.160	133.0	NA
asrc	88200	48000	0.9999	0	2398.542	123.6	-197.9dB
xsim-asrc	88200	48000	0.9999	0	2398.542	123.1	-198.4dB
asrc	88200	48000	0.9999	1	397.144-21801.124	129.2	NA
xsim-asrc	88200	48000	0.9999	1	397.144-21801.124	128.2	NA
asrc	96000	48000	0.9999	0	2398.080	129.5	-198.1dB
xsim-asrc	96000	48000	0.9999	0	2398.080	127.7	-201.4dB
asrc	96000	48000	0.9999	1	2136.427-21801.160	134.3	NA
xsim-asrc	96000	48000	0.9999	1	2136.427-21801.160	130.9	NA
asrc	176400	48000	0.9999	0	2399.848	122.9	-196.9dB
xsim-asrc	176400	48000	0.9999	0	2399.848	122.1	-200.7dB
asrc	176400	48000	0.9999	1	397.144-21801.124	127.6	NA
xsim-asrc	176400	48000	0.9999	1	397.144-21801.124	126.0	NA
asrc	192000	48000	0.9999	0	2399.280	128.2	-197.5dB
xsim-asrc	192000	48000	0.9999	0	2399.280	125.8	-202.2dB
asrc	192000	48000	0.9999	1	2136.427-21801.160	132.7	NA
xsim-asrc	192000	48000	0.9999	1	2136.427-21801.160	128.4	NA
asrc	44100	88200	0.9999	0	4393.678	142.8	-197.1dB
xsim-asrc	44100	88200	0.9999	0	4393.678	140.1	-193.2dB
asrc	44100	88200	0.9999	1	8963.809-17998.199	145.2	NA

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
xsim-asrc	44100	88200	0.9999	1	8963.809-17998.199	141.0	NA
asrc	48000	88200	0.9999	0	4409.630	126.7	-184.2dB
xsim-asrc	48000	88200	0.9999	0	4409.630	126.5	-185.5dB
asrc	48000	88200	0.9999	1	5187.801-21788.762	131.9	NA
xsim-asrc	48000	88200	0.9999	1	5187.801-21788.762	131.5	NA
asrc	88200	88200	0.9999	0	4402.060	130.4	-197.2dB
xsim-asrc	88200	88200	0.9999	0	4402.060	129.4	-196.4dB
asrc	88200	88200	0.9999	1	3863.933-39997.879	135.1	NA
xsim-asrc	88200	88200	0.9999	1	3863.933-39997.879	133.1	NA
asrc	96000	88200	0.9999	0	4409.225	125.1	-189.7dB
xsim-asrc	96000	88200	0.9999	0	4409.225	124.8	-187.9dB
asrc	96000	88200	0.9999	1	6800.257-39999.130	131.7	NA
xsim-asrc	96000	88200	0.9999	1	6800.257-39999.130	130.9	NA
asrc	176400	88200	0.9999	0	4406.471	129.5	-197.0dB
xsim-asrc	176400	88200	0.9999	0	4406.471	127.9	-202.1dB
asrc	176400	88200	0.9999	1	3868.344-40002.290	133.7	NA
xsim-asrc	176400	88200	0.9999	1	3868.344-40002.290	129.8	NA
asrc	192000	88200	0.9999	0	4409.225	124.2	-193.8dB
xsim-asrc	192000	88200	0.9999	0	4409.225	123.6	-193.8dB
asrc	192000	88200	0.9999	1	6804.310-40003.183	129.8	NA
xsim-asrc	192000	88200	0.9999	1	6804.310-40003.183	127.5	NA
asrc	44100	96000	0.9999	0	4786.893	128.8	-187.4dB
xsim-asrc	44100	96000	0.9999	0	4786.893	128.6	-187.0dB
asrc	44100	96000	0.9999	1	7295.308-17997.881	132.9	NA
xsim-asrc	44100	96000	0.9999	1	7295.308-17997.881	132.6	NA
asrc	48000	96000	0.9999	0	4782.235	142.8	-197.1dB
xsim-asrc	48000	96000	0.9999	0	4782.235	140.1	-193.2dB

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
asrc	48000	96000	0.9999	1	2131.840-21798.539	145.0	NA
xsim-asrc	48000	96000	0.9999	1	2131.840-21798.539	140.9	NA
asrc	88200	96000	0.9999	0	4797.345	124.9	-189.8dB
xsim-asrc	88200	96000	0.9999	0	4797.345	124.6	-190.4dB
asrc	88200	96000	0.9999	1	18593.629-39998.775	130.1	NA
xsim-asrc	88200	96000	0.9999	1	18593.629-39998.775	129.7	NA
asrc	96000	96000	0.9999	0	4791.358	130.4	-197.2dB
xsim-asrc	96000	96000	0.9999	0	4791.358	129.4	-196.4dB
asrc	96000	96000	0.9999	1	2669.334-41998.799	135.5	NA
xsim-asrc	96000	96000	0.9999	1	2669.334-41998.799	133.3	NA
asrc	176400	96000	0.9999	0	4797.083	124.4	-196.3dB
xsim-asrc	176400	96000	0.9999	0	4797.083	123.8	-197.7dB
asrc	176400	96000	0.9999	1	20599.241-42003.221	128.3	NA
xsim-asrc	176400	96000	0.9999	1	20599.241-42003.221	127.4	NA
asrc	192000	96000	0.9999	0	4796.159	129.5	-197.0dB
xsim-asrc	192000	96000	0.9999	0	4796.159	127.9	-202.1dB
asrc	192000	96000	0.9999	1	2674.135-42003.600	134.7	NA
xsim-asrc	192000	96000	0.9999	1	2674.135-42003.600	131.0	NA
asrc	44100	176400	0.9999	0	8753.817	140.4	-189.6dB
xsim-asrc	44100	176400	0.9999	0	8753.817	139.1	-189.3dB
asrc	44100	176400	0.9999	1	8965.602-18001.800	142.7	NA
xsim-asrc	44100	176400	0.9999	1	8965.602-18001.800	140.7	NA
asrc	48000	176400	0.9999	0	8820.882	127.6	-180.5dB
xsim-asrc	48000	176400	0.9999	0	8820.882	127.4	-180.3dB
asrc	48000	176400	0.9999	1	5188.754-21792.768	133.7	NA
xsim-asrc	48000	176400	0.9999	1	5188.754-21792.768	133.4	NA
asrc	88200	176400	0.9999	0	8787.356	142.8	-197.1dB

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
xsim-asrc	88200	176400	0.9999	0	8787.356	140.1	-193.2dB
asrc	88200	176400	0.9999	1	3846.674-39984.234	145.4	NA
xsim-asrc	88200	176400	0.9999	1	3846.674-39984.234	140.9	NA
asrc	96000	176400	0.9999	0	8819.261	126.7	-184.2dB
xsim-asrc	96000	176400	0.9999	0	8819.261	126.5	-185.5dB
asrc	96000	176400	0.9999	1	8786.837-41988.761	133.0	NA
xsim-asrc	96000	176400	0.9999	1	8786.837-41988.761	132.4	NA
asrc	176400	176400	0.9999	0	8804.121	130.4	-197.7dB
xsim-asrc	176400	176400	0.9999	0	8804.121	129.4	-193.2dB
asrc	176400	176400	0.9999	1	7727.865-79995.758	134.1	NA
xsim-asrc	176400	176400	0.9999	1	7727.865-79995.758	131.8	NA
asrc	192000	176400	0.9999	0	8818.451	125.4	-187.8dB
xsim-asrc	192000	176400	0.9999	0	8818.451	125.0	-187.4dB
asrc	192000	176400	0.9999	1	13600.515-79998.260	130.5	NA
xsim-asrc	192000	176400	0.9999	1	13600.515-79998.260	129.4	NA
asrc	44100	192000	0.9999	0	9534.054	127.5	-179.5dB
xsim-asrc	44100	192000	0.9999	0	9534.054	127.4	-180.1dB
asrc	44100	192000	0.9999	1	7275.989-17980.892	134.7	NA
xsim-asrc	44100	192000	0.9999	1	7275.989-17980.892	134.4	NA
asrc	48000	192000	0.9999	0	9527.964	140.4	-189.6dB
xsim-asrc	48000	192000	0.9999	0	9527.964	139.1	-189.3dB
asrc	48000	192000	0.9999	1	2074.637-21745.272	134.3	NA
xsim-asrc	48000	192000	0.9999	1	2074.637-21745.272	133.6	NA
asrc	88200	192000	0.9999	0	9573.786	128.8	-187.4dB
xsim-asrc	88200	192000	0.9999	0	9573.786	128.6	-187.0dB
asrc	88200	192000	0.9999	1	18562.274-39967.419	131.3	NA

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
xsim-asrc	88200	192000	0.9999	1	18562.274-39967.419	131.1	NA
asrc	96000	192000	0.9999	0	9564.469	142.8	-197.1dB
xsim-asrc	96000	192000	0.9999	0	9564.469	140.1	-193.2dB
asrc	96000	192000	0.9999	1	2650.395-41983.794	134.1	NA
xsim-asrc	96000	192000	0.9999	1	2650.395-41983.794	133.0	NA
asrc	176400	192000	0.9999	0	9594.689	125.1	-190.0dB
xsim-asrc	176400	192000	0.9999	0	9594.689	124.8	-187.0dB
asrc	176400	192000	0.9999	1	37187.259-79997.549	130.3	NA
xsim-asrc	176400	192000	0.9999	1	37187.259-79997.549	129.6	NA
asrc	192000	192000	0.9999	0	9582.716	130.4	-197.7dB
xsim-asrc	192000	192000	0.9999	0	9582.716	129.4	-193.2dB
asrc	192000	192000	0.9999	1	6337.267-84996.198	134.8	NA
xsim-asrc	192000	192000	0.9999	1	6337.267-84996.198	132.7	NA
ds3	48000	16000	1.0	0	800.000	182.3	-195.0dB
ds3	48000	16000	1.0	1	2930.667-7299.733	146.5	NA
ds3	96000	32000	1.0	0	1600.000	182.3	-195.0dB
ds3	96000	32000	1.0	1	5861.333-14599.467	146.5	NA
asrc	44100	44100	1.0	0	2205.000	175.8	-188.7dB
ssrc	44100	44100	1.0	0	2205.000	189.5	-193.1dB
xsim-asrc	44100	44100	1.0	0	2205.000	164.9	-177.0dB
asrc	44100	44100	1.0	1	8965.530-17997.210	154.9	NA
ssrc	44100	44100	1.0	1	8965.530-17997.210	158.7	NA
xsim-asrc	44100	44100	1.0	1	8965.530-17997.210	141.8	NA
asrc	48000	44100	1.0	0	2204.190	125.7	-196.6dB
ssrc	48000	44100	1.0	0	2204.190	107.9	-176.9dB
xsim-asrc	48000	44100	1.0	0	2204.190	125.2	-194.8dB
asrc	48000	44100	1.0	1	1401.929-17998.181	131.1	NA

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
ssrc	48000	44100	1.0	1	1401.929-17998.181	112.8	NA
xsim-asrc	48000	44100	1.0	1	1401.929-17998.181	130.4	NA
asrc	88200	44100	1.0	0	2205.000	177.8	-195.5dB
ssrc	88200	44100	1.0	0	2205.000	181.0	-193.3dB
xsim-asrc	88200	44100	1.0	0	2205.000	164.4	-177.2dB
asrc	88200	44100	1.0	1	8967.735-17999.415	153.3	NA
ssrc	88200	44100	1.0	1	8967.735-17999.415	154.6	NA
xsim-asrc	88200	44100	1.0	1	8967.735-17999.415	137.2	NA
asrc	96000	44100	1.0	0	2204.190	124.6	-194.9dB
ssrc	96000	44100	1.0	0	2204.190	106.9	-180.2dB
xsim-asrc	96000	44100	1.0	0	2204.190	123.8	-198.1dB
asrc	96000	44100	1.0	1	1401.929-17998.181	130.2	NA
ssrc	96000	44100	1.0	1	1401.929-17998.181	111.4	NA
xsim-asrc	96000	44100	1.0	1	1401.929-17998.181	128.7	NA
asrc	176400	44100	1.0	0	2205.000	174.9	-194.2dB
ssrc	176400	44100	1.0	0	2205.000	176.3	-190.4dB
xsim-asrc	176400	44100	1.0	0	2205.000	162.3	-176.8dB
asrc	176400	44100	1.0	1	8967.735-17999.415	149.8	NA
ssrc	176400	44100	1.0	1	8967.735-17999.415	151.1	NA
xsim-asrc	176400	44100	1.0	1	8967.735-17999.415	136.0	NA
asrc	192000	44100	1.0	0	2204.139	123.6	-196.9dB
ssrc	192000	44100	1.0	0	2204.139	106.0	-185.4dB
xsim-asrc	192000	44100	1.0	0	2204.139	122.3	-200.1dB
asrc	192000	44100	1.0	1	1403.923-17999.793	128.9	NA
ssrc	192000	44100	1.0	1	1403.923-17999.793	110.4	NA
xsim-asrc	192000	44100	1.0	1	1403.923-17999.793	126.5	NA
os3	16000	48000	1.0	0	2390.639	126.7	-202.5dB

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
os3	16000	48000	1.0	1	3600.360-7287.129	124.1	NA
asrc	44100	48000	1.0	0	2398.171	127.5	-172.0dB
ssrc	44100	48000	1.0	0	2398.171	106.1	-151.6dB
xsim-asrc	44100	48000	1.0	0	2398.171	126.8	-171.3dB
asrc	44100	48000	1.0	1	7299.009-17999.347	132.7	NA
ssrc	44100	48000	1.0	1	7299.009-17999.347	110.2	NA
xsim-asrc	44100	48000	1.0	1	7299.009-17999.347	131.9	NA
asrc	48000	48000	1.0	0	2400.000	175.8	-188.7dB
ssrc	48000	48000	1.0	0	2400.000	189.5	-193.1dB
xsim-asrc	48000	48000	1.0	0	2400.000	164.9	-177.0dB
asrc	48000	48000	1.0	1	2136.000-21796.800	155.3	NA
ssrc	48000	48000	1.0	1	2136.000-21796.800	158.9	NA
xsim-asrc	48000	48000	1.0	1	2136.000-21796.800	141.8	NA
asrc	88200	48000	1.0	0	2398.041	125.4	-176.8dB
ssrc	88200	48000	1.0	0	2398.041	103.9	-159.0dB
xsim-asrc	88200	48000	1.0	0	2398.041	124.6	-176.6dB
asrc	88200	48000	1.0	1	399.673-21799.184	129.8	NA
ssrc	88200	48000	1.0	1	399.673-21799.184	110.2	NA
xsim-asrc	88200	48000	1.0	1	399.673-21799.184	128.5	NA
asrc	96000	48000	1.0	0	2400.000	177.8	-195.5dB
ssrc	96000	48000	1.0	0	2400.000	181.0	-193.3dB
xsim-asrc	96000	48000	1.0	0	2400.000	164.4	-177.2dB
asrc	96000	48000	1.0	1	2138.400-21799.200	153.1	NA
ssrc	96000	48000	1.0	1	2138.400-21799.200	154.6	NA
xsim-asrc	96000	48000	1.0	1	2138.400-21799.200	137.2	NA
asrc	176400	48000	1.0	0	2399.347	125.8	-174.6dB
ssrc	176400	48000	1.0	0	2399.347	102.5	-151.0dB
xsim-asrc	176400	48000	1.0	0	2399.347	124.0	-174.3dB

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
asrc	176400	48000	1.0	1	399.673-21799.184	128.3	NA
ssrc	176400	48000	1.0	1	399.673-21799.184	109.3	NA
xsim-asrc	176400	48000	1.0	1	399.673-21799.184	126.2	NA
asrc	192000	48000	1.0	0	2400.000	174.9	-194.2dB
ssrc	192000	48000	1.0	0	2400.000	176.3	-190.4dB
xsim-asrc	192000	48000	1.0	0	2400.000	162.3	-176.8dB
asrc	192000	48000	1.0	1	2138.400-21799.200	149.8	NA
ssrc	192000	48000	1.0	1	2138.400-21799.200	151.2	NA
xsim-asrc	192000	48000	1.0	1	2138.400-21799.200	136.0	NA
asrc	44100	88200	1.0	0	4410.000	166.7	-169.5dB
ssrc	44100	88200	1.0	0	4410.000	156.0	-156.5dB
xsim-asrc	44100	88200	1.0	0	4410.000	161.9	-167.8dB
asrc	44100	88200	1.0	1	8961.120-17992.800	155.1	NA
ssrc	44100	88200	1.0	1	8961.120-17992.800	155.4	NA
xsim-asrc	44100	88200	1.0	1	8961.120-17992.800	146.3	NA
asrc	48000	88200	1.0	0	4408.379	127.2	-183.8dB
ssrc	48000	88200	1.0	0	4408.379	108.7	-168.3dB
xsim-asrc	48000	88200	1.0	0	4408.379	126.8	-184.0dB
asrc	48000	88200	1.0	1	5202.536-21798.787	131.8	NA
ssrc	48000	88200	1.0	1	5202.536-21798.787	115.1	NA
xsim-asrc	48000	88200	1.0	1	5202.536-21798.787	131.4	NA
asrc	88200	88200	1.0	0	4410.000	175.8	-188.7dB
ssrc	88200	88200	1.0	0	4410.000	189.5	-193.1dB
xsim-asrc	88200	88200	1.0	0	4410.000	164.9	-177.0dB
asrc	88200	88200	1.0	1	3871.980-39998.700	155.1	NA
ssrc	88200	88200	1.0	1	3871.980-39998.700	159.1	NA
xsim-asrc	88200	88200	1.0	1	3871.980-39998.700	141.7	NA

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
asrc	96000	88200	1.0	0	4408.379	125.7	-196.6dB
ssrc	96000	88200	1.0	0	4408.379	107.9	-176.9dB
xsim-asrc	96000	88200	1.0	0	4408.379	125.2	-194.8dB
asrc	96000	88200	1.0	1	6807.056-39999.559	131.8	NA
ssrc	96000	88200	1.0	1	6807.056-39999.559	113.2	NA
xsim-asrc	96000	88200	1.0	1	6807.056-39999.559	131.0	NA
asrc	176400	88200	1.0	0	4410.000	176.1	-189.7dB
ssrc	176400	88200	1.0	0	4410.000	181.0	-193.3dB
xsim-asrc	176400	88200	1.0	0	4410.000	164.6	-174.8dB
asrc	176400	88200	1.0	1	3871.980-39998.700	151.0	NA
ssrc	176400	88200	1.0	1	3871.980-39998.700	154.7	NA
xsim-asrc	176400	88200	1.0	1	3871.980-39998.700	137.6	NA
asrc	192000	88200	1.0	0	4408.379	124.6	-198.1dB
ssrc	192000	88200	1.0	0	4408.379	107.1	-181.9dB
xsim-asrc	192000	88200	1.0	0	4408.379	123.8	-194.5dB
asrc	192000	88200	1.0	1	6807.056-39999.559	130.0	NA
ssrc	192000	88200	1.0	1	6807.056-39999.559	111.5	NA
xsim-asrc	192000	88200	1.0	1	6807.056-39999.559	127.5	NA
os3	32000	96000	1.0	0	4781.278	126.7	-202.5dB
os3	32000	96000	1.0	1	7200.720-14574.257	124.1	NA
asrc	44100	96000	1.0	0	4786.414	130.7	-190.4dB
ssrc	44100	96000	1.0	0	4786.414	108.3	-163.7dB
xsim-asrc	44100	96000	1.0	0	4786.414	130.1	-188.7dB
asrc	44100	96000	1.0	1	7294.579-17996.081	133.5	NA
ssrc	44100	96000	1.0	1	7294.579-17996.081	111.7	NA
xsim-asrc	44100	96000	1.0	1	7294.579-17996.081	133.0	NA
asrc	48000	96000	1.0	0	4800.000	166.7	-169.5dB
ssrc	48000	96000	1.0	0	4800.000	156.0	-156.5dB

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
xsim-asrc	48000	96000	1.0	0	4800.000	161.9	-167.8dB
asrc	48000	96000	1.0	1	2131.200-21792.000	154.4	NA
ssrc	48000	96000	1.0	1	2131.200-21792.000	154.9	NA
xsim-asrc	48000	96000	1.0	1	2131.200-21792.000	144.0	NA
asrc	88200	96000	1.0	0	4796.343	127.5	-172.0dB
ssrc	88200	96000	1.0	0	4796.343	106.1	-151.6dB
xsim-asrc	88200	96000	1.0	0	4796.343	126.8	-171.3dB
asrc	88200	96000	1.0	1	18589.746-39990.421	129.3	NA
ssrc	88200	96000	1.0	1	18589.746-39990.421	108.6	NA
xsim-asrc	88200	96000	1.0	1	18589.746-39990.421	128.9	NA
asrc	96000	96000	1.0	0	4800.000	175.8	-188.7dB
ssrc	96000	96000	1.0	0	4800.000	189.5	-193.1dB
xsim-asrc	96000	96000	1.0	0	4800.000	164.9	-177.0dB
asrc	96000	96000	1.0	1	2678.400-42000.000	155.2	NA
ssrc	96000	96000	1.0	1	2678.400-42000.000	158.8	NA
xsim-asrc	96000	96000	1.0	1	2678.400-42000.000	141.9	NA
asrc	176400	96000	1.0	0	4796.082	125.0	-177.1dB
ssrc	176400	96000	1.0	0	4796.082	104.6	-156.9dB
xsim-asrc	176400	96000	1.0	0	4796.082	124.3	-177.3dB
asrc	176400	96000	1.0	1	20600.163-41999.673	128.4	NA
ssrc	176400	96000	1.0	1	20600.163-41999.673	108.8	NA
xsim-asrc	176400	96000	1.0	1	20600.163-41999.673	127.4	NA
asrc	192000	96000	1.0	0	4800.000	176.1	-189.7dB
ssrc	192000	96000	1.0	0	4800.000	181.0	-193.3dB
xsim-asrc	192000	96000	1.0	0	4800.000	164.6	-174.8dB
asrc	192000	96000	1.0	1	2678.400-42000.000	152.6	NA
ssrc	192000	96000	1.0	1	2678.400-42000.000	154.7	NA

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
xsim-asrc	192000	96000	1.0	1	2678.400-42000.000	139.4	NA
asrc	44100	176400	1.0	0	8820.000	165.3	-169.5dB
ssrc	44100	176400	1.0	0	8820.000	163.6	-166.5dB
xsim-asrc	44100	176400	1.0	0	8820.000	161.3	-168.6dB
asrc	44100	176400	1.0	1	8961.120-17992.800	154.3	NA
ssrc	44100	176400	1.0	1	8961.120-17992.800	152.1	NA
xsim-asrc	44100	176400	1.0	1	8961.120-17992.800	146.1	NA
asrc	48000	176400	1.0	0	8816.759	127.6	-182.1dB
ssrc	48000	176400	1.0	0	8816.759	110.0	-163.9dB
xsim-asrc	48000	176400	1.0	0	8816.759	127.4	-181.7dB
asrc	48000	176400	1.0	1	5186.329-21782.580	133.7	NA
ssrc	48000	176400	1.0	1	5186.329-21782.580	115.8	NA
xsim-asrc	48000	176400	1.0	1	5186.329-21782.580	133.4	NA
asrc	88200	176400	1.0	0	8820.000	166.7	-169.5dB
ssrc	88200	176400	1.0	0	8820.000	156.0	-156.5dB
xsim-asrc	88200	176400	1.0	0	8820.000	161.9	-167.8dB
asrc	88200	176400	1.0	1	3845.520-39972.240	154.4	NA
ssrc	88200	176400	1.0	1	3845.520-39972.240	154.9	NA
xsim-asrc	88200	176400	1.0	1	3845.520-39972.240	144.0	NA
asrc	96000	176400	1.0	0	8816.759	127.2	-183.8dB
ssrc	96000	176400	1.0	0	8816.759	108.7	-168.3dB
xsim-asrc	96000	176400	1.0	0	8816.759	126.8	-184.0dB
asrc	96000	176400	1.0	1	8784.344-41976.847	133.8	NA
ssrc	96000	176400	1.0	1	8784.344-41976.847	114.6	NA
xsim-asrc	96000	176400	1.0	1	8784.344-41976.847	133.1	NA
asrc	176400	176400	1.0	0	8820.000	176.1	-189.0dB
ssrc	176400	176400	1.0	0	8820.000	189.5	-193.1dB
xsim-asrc	176400	176400	1.0	0	8820.000	170.2	-181.8dB

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
asrc	176400	176400	1.0	1	7743.960-79997.400	153.1	NA
ssrc	176400	176400	1.0	1	7743.960-79997.400	159.1	NA
xsim-asrc	176400	176400	1.0	1	7743.960-79997.400	139.9	NA
asrc	192000	176400	1.0	0	8816.759	125.6	-195.7dB
ssrc	192000	176400	1.0	0	8816.759	107.8	-180.8dB
xsim-asrc	192000	176400	1.0	0	8816.759	125.1	-192.7dB
asrc	192000	176400	1.0	1	13614.112-79999.118	130.6	NA
ssrc	192000	176400	1.0	1	13614.112-79999.118	112.7	NA
xsim-asrc	192000	176400	1.0	1	13614.112-79999.118	129.4	NA
asrc	44100	192000	1.0	0	9533.101	127.6	-181.9dB
ssrc	44100	192000	1.0	0	9533.101	107.6	-166.0dB
xsim-asrc	44100	192000	1.0	0	9533.101	127.4	-181.2dB
asrc	44100	192000	1.0	1	7275.261-17979.094	134.7	NA
ssrc	44100	192000	1.0	1	7275.261-17979.094	111.3	NA
xsim-asrc	44100	192000	1.0	1	7275.261-17979.094	134.3	NA
asrc	48000	192000	1.0	0	9600.000	165.3	-169.5dB
ssrc	48000	192000	1.0	0	9600.000	163.6	-166.5dB
xsim-asrc	48000	192000	1.0	0	9600.000	161.3	-168.6dB
asrc	48000	192000	1.0	1	2073.600-21734.400	153.7	NA
ssrc	48000	192000	1.0	1	2073.600-21734.400	152.5	NA
xsim-asrc	48000	192000	1.0	1	2073.600-21734.400	146.0	NA
asrc	88200	192000	1.0	0	9572.828	130.7	-190.4dB
ssrc	88200	192000	1.0	0	9572.828	108.3	-163.7dB
xsim-asrc	88200	192000	1.0	0	9572.828	130.1	-188.7dB
asrc	88200	192000	1.0	1	18560.418-39963.423	131.5	NA
ssrc	88200	192000	1.0	1	18560.418-39963.423	109.2	NA
xsim-asrc	88200	192000	1.0	1	18560.418-39963.423	131.3	NA

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
asrc	96000	192000	1.0	0	9600.000	166.7	-169.5dB
ssrc	96000	192000	1.0	0	9600.000	156.0	-156.5dB
xsim-asrc	96000	192000	1.0	0	9600.000	161.9	-167.8dB
asrc	96000	192000	1.0	1	2649.600-41971.200	154.3	NA
ssrc	96000	192000	1.0	1	2649.600-41971.200	155.0	NA
xsim-asrc	96000	192000	1.0	1	2649.600-41971.200	143.9	NA
asrc	176400	192000	1.0	0	9592.685	126.1	-170.2dB
ssrc	176400	192000	1.0	0	9592.685	105.5	-151.0dB
xsim-asrc	176400	192000	1.0	0	9592.685	125.8	-169.8dB
asrc	176400	192000	1.0	1	37179.493-79980.842	129.2	NA
ssrc	176400	192000	1.0	1	37179.493-79980.842	108.6	NA
xsim-asrc	176400	192000	1.0	1	37179.493-79980.842	128.6	NA
asrc	192000	192000	1.0	0	9600.000	176.1	-189.0dB
ssrc	192000	192000	1.0	0	9600.000	189.5	-193.1dB
xsim-asrc	192000	192000	1.0	0	9600.000	170.2	-181.8dB
asrc	192000	192000	1.0	1	6355.200-84998.400	154.2	NA
ssrc	192000	192000	1.0	1	6355.200-84998.400	158.8	NA
xsim-asrc	192000	192000	1.0	1	6355.200-84998.400	141.2	NA
asrc	44100	44100	1.0001	0	2204.780	130.4	-197.3dB
xsim-asrc	44100	44100	1.0001	0	2204.780	129.3	-195.1dB
asrc	44100	44100	1.0001	1	8964.634-17995.410	136.6	NA
xsim-asrc	44100	44100	1.0001	1	8964.634-17995.410	134.1	NA
asrc	48000	44100	1.0001	0	2203.767	125.2	-176.9dB
xsim-asrc	48000	44100	1.0001	0	2203.767	124.9	-177.1dB
asrc	48000	44100	1.0001	1	1401.660-17994.728	132.4	NA
xsim-asrc	48000	44100	1.0001	1	1401.660-17994.728	131.4	NA
asrc	88200	44100	1.0001	0	2204.669	129.6	-197.6dB

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
xsim-asrc	88200	44100	1.0001	0	2204.669	127.7	-202.2dB
asrc	88200	44100	1.0001	1	8966.390-17996.715	134.9	NA
xsim-asrc	88200	44100	1.0001	1	8966.390-17996.715	131.3	NA
asrc	96000	44100	1.0001	0	2203.767	124.0	-178.6dB
xsim-asrc	96000	44100	1.0001	0	2203.767	123.4	-179.0dB
asrc	96000	44100	1.0001	1	1403.686-17996.753	130.4	NA
xsim-asrc	96000	44100	1.0001	1	1403.686-17996.753	128.7	NA
asrc	176400	44100	1.0001	0	2204.614	128.1	-197.8dB
xsim-asrc	176400	44100	1.0001	0	2204.614	125.8	-202.3dB
asrc	176400	44100	1.0001	1	8967.268-17997.368	132.3	NA
xsim-asrc	176400	44100	1.0001	1	8967.268-17997.368	128.1	NA
asrc	192000	44100	1.0001	0	2204.729	123.0	-194.4dB
xsim-asrc	192000	44100	1.0001	0	2204.729	121.9	-195.6dB
asrc	192000	44100	1.0001	1	1404.666-17997.353	128.7	NA
xsim-asrc	192000	44100	1.0001	1	1404.666-17997.353	126.4	NA
asrc	44100	48000	1.0001	0	2397.671	124.9	-180.3dB
xsim-asrc	44100	48000	1.0001	0	2397.671	124.7	-181.2dB
asrc	44100	48000	1.0001	1	7297.485-17995.588	133.6	NA
xsim-asrc	44100	48000	1.0001	1	7297.485-17995.588	132.6	NA
asrc	48000	48000	1.0001	0	2399.520	130.4	-197.0dB
xsim-asrc	48000	48000	1.0001	0	2399.520	129.3	-195.2dB
asrc	48000	48000	1.0001	1	2140.372-21797.240	135.7	NA
xsim-asrc	48000	48000	1.0001	1	2140.372-21797.240	133.5	NA
asrc	88200	48000	1.0001	0	2397.671	123.8	-180.8dB
xsim-asrc	88200	48000	1.0001	0	2397.671	123.3	-181.3dB
asrc	88200	48000	1.0001	1	399.612-21795.818	130.2	NA

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
xsim-asrc	88200	48000	1.0001	1	399.612-21795.818	128.9	NA
asrc	96000	48000	1.0001	0	2399.520	129.5	-197.3dB
xsim-asrc	96000	48000	1.0001	0	2399.520	127.7	-201.6dB
asrc	96000	48000	1.0001	1	2140.372-21797.240	134.2	NA
xsim-asrc	96000	48000	1.0001	1	2140.372-21797.240	130.8	NA
asrc	176400	48000	1.0001	0	2398.911	122.6	-196.5dB
xsim-asrc	176400	48000	1.0001	0	2398.911	121.8	-198.6dB
asrc	176400	48000	1.0001	1	400.907-21796.531	129.2	NA
xsim-asrc	176400	48000	1.0001	1	400.907-21796.531	126.9	NA
asrc	192000	48000	1.0001	0	2399.520	128.0	-197.5dB
xsim-asrc	192000	48000	1.0001	0	2399.520	125.7	-203.7dB
asrc	192000	48000	1.0001	1	2140.372-21797.240	131.5	NA
xsim-asrc	192000	48000	1.0001	1	2140.372-21797.240	127.7	NA
asrc	44100	88200	1.0001	0	4409.559	132.2	-170.4dB
xsim-asrc	44100	88200	1.0001	0	4409.559	131.6	-170.9dB
asrc	44100	88200	1.0001	1	8960.224-17991.001	137.7	NA
xsim-asrc	44100	88200	1.0001	1	8960.224-17991.001	136.2	NA
asrc	48000	88200	1.0001	0	4407.938	127.2	-189.5dB
xsim-asrc	48000	88200	1.0001	0	4407.938	126.9	-189.7dB
asrc	48000	88200	1.0001	1	5202.016-21796.608	132.4	NA
xsim-asrc	48000	88200	1.0001	1	5202.016-21796.608	132.0	NA
asrc	88200	88200	1.0001	0	4409.559	130.4	-197.3dB
xsim-asrc	88200	88200	1.0001	0	4409.559	129.3	-195.1dB
asrc	88200	88200	1.0001	1	3871.593-39994.701	135.6	NA
xsim-asrc	88200	88200	1.0001	1	3871.593-39994.701	133.5	NA
asrc	96000	88200	1.0001	0	4407.534	125.2	-176.9dB
xsim-asrc	96000	88200	1.0001	0	4407.534	124.9	-177.1dB

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
asrc	96000	88200	1.0001	1	6805.750-39991.885	132.3	NA
xsim-asrc	96000	88200	1.0001	1	6805.750-39991.885	131.4	NA
asrc	176400	88200	1.0001	0	4409.339	129.5	-198.4dB
xsim-asrc	176400	88200	1.0001	0	4409.339	127.9	-203.5dB
asrc	176400	88200	1.0001	1	3871.399-39992.701	133.3	NA
xsim-asrc	176400	88200	1.0001	1	3871.399-39992.701	129.6	NA
asrc	192000	88200	1.0001	0	4407.534	124.2	-178.1dB
xsim-asrc	192000	88200	1.0001	0	4407.534	123.6	-177.7dB
asrc	192000	88200	1.0001	1	6809.801-39995.936	131.0	NA
xsim-asrc	192000	88200	1.0001	1	6809.801-39995.936	128.0	NA
asrc	44100	96000	1.0001	0	4784.894	129.0	-184.8dB
xsim-asrc	44100	96000	1.0001	0	4784.894	128.6	-184.7dB
asrc	44100	96000	1.0001	1	7292.262-17990.365	133.5	NA
xsim-asrc	44100	96000	1.0001	1	7292.262-17990.365	132.9	NA
asrc	48000	96000	1.0001	0	4799.520	132.2	-170.4dB
xsim-asrc	48000	96000	1.0001	0	4799.520	131.6	-170.9dB
asrc	48000	96000	1.0001	1	2130.987-21789.821	135.4	NA
xsim-asrc	48000	96000	1.0001	1	2130.987-21789.821	134.0	NA
asrc	88200	96000	1.0001	0	4795.341	124.9	-180.3dB
xsim-asrc	88200	96000	1.0001	0	4795.341	124.7	-181.2dB
asrc	88200	96000	1.0001	1	18596.312-39992.518	131.0	NA
xsim-asrc	88200	96000	1.0001	1	18596.312-39992.518	130.4	NA
asrc	96000	96000	1.0001	0	4799.040	130.4	-197.0dB
xsim-asrc	96000	96000	1.0001	0	4799.040	129.3	-195.2dB
asrc	96000	96000	1.0001	1	2677.864-41991.601	136.2	NA
xsim-asrc	96000	96000	1.0001	1	2677.864-41991.601	133.9	NA
asrc	176400	96000	1.0001	0	4795.341	123.7	-181.1dB

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
xsim-asrc	176400	96000	1.0001	0	4795.341	123.3	-181.5dB
asrc	176400	96000	1.0001	1	20596.983-41993.189	128.3	NA
xsim-asrc	176400	96000	1.0001	1	20596.983-41993.189	127.4	NA
asrc	192000	96000	1.0001	0	4799.040	129.5	-197.7dB
xsim-asrc	192000	96000	1.0001	0	4799.040	127.8	-202.9dB
asrc	192000	96000	1.0001	1	2677.864-41991.601	134.2	NA
xsim-asrc	192000	96000	1.0001	1	2677.864-41991.601	130.7	NA
asrc	44100	176400	1.0001	0	8819.118	130.7	-165.8dB
xsim-asrc	44100	176400	1.0001	0	8819.118	130.2	-165.8dB
asrc	44100	176400	1.0001	1	8960.224-17991.001	136.3	NA
xsim-asrc	44100	176400	1.0001	1	8960.224-17991.001	135.3	NA
asrc	48000	176400	1.0001	0	8815.877	127.5	-188.7dB
xsim-asrc	48000	176400	1.0001	0	8815.877	127.4	-187.2dB
asrc	48000	176400	1.0001	1	5185.810-21780.402	134.2	NA
xsim-asrc	48000	176400	1.0001	1	5185.810-21780.402	133.9	NA
asrc	88200	176400	1.0001	0	8819.118	132.2	-170.4dB
xsim-asrc	88200	176400	1.0001	0	8819.118	131.6	-170.9dB
asrc	88200	176400	1.0001	1	3845.135-39968.243	135.4	NA
xsim-asrc	88200	176400	1.0001	1	3845.135-39968.243	134.0	NA
asrc	96000	176400	1.0001	0	8815.877	127.2	-189.5dB
xsim-asrc	96000	176400	1.0001	0	8815.877	126.9	-189.7dB
asrc	96000	176400	1.0001	1	8783.466-41972.649	134.3	NA
xsim-asrc	96000	176400	1.0001	1	8783.466-41972.649	133.5	NA
asrc	176400	176400	1.0001	0	8819.118	130.4	-198.5dB
xsim-asrc	176400	176400	1.0001	0	8819.118	129.4	-195.1dB
asrc	176400	176400	1.0001	1	7743.186-79989.401	134.3	NA

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
xsim-asrc	176400	176400	1.0001	1	7743.186-79989.401	132.0	NA
asrc	192000	176400	1.0001	0	8815.067	125.0	-176.7dB
xsim-asrc	192000	176400	1.0001	0	8815.067	124.7	-176.5dB
asrc	192000	176400	1.0001	1	13611.501-79983.770	131.2	NA
xsim-asrc	192000	176400	1.0001	1	13611.501-79983.770	129.9	NA
asrc	44100	192000	1.0001	0	9527.998	127.5	-180.3dB
xsim-asrc	44100	192000	1.0001	0	9527.998	127.4	-179.6dB
asrc	44100	192000	1.0001	1	7271.367-17969.470	134.9	NA
xsim-asrc	44100	192000	1.0001	1	7271.367-17969.470	134.5	NA
asrc	48000	192000	1.0001	0	9599.040	130.7	-165.8dB
xsim-asrc	48000	192000	1.0001	0	9599.040	130.2	-165.8dB
asrc	48000	192000	1.0001	1	2073.393-21732.227	136.2	NA
xsim-asrc	48000	192000	1.0001	1	2073.393-21732.227	135.1	NA
asrc	88200	192000	1.0001	0	9569.787	129.0	-184.8dB
xsim-asrc	88200	192000	1.0001	0	9569.787	128.6	-184.7dB
asrc	88200	192000	1.0001	1	18596.312-39992.518	132.0	NA
xsim-asrc	88200	192000	1.0001	1	18596.312-39992.518	131.6	NA
asrc	96000	192000	1.0001	0	9599.040	132.2	-170.4dB
xsim-asrc	96000	192000	1.0001	0	9599.040	131.6	-170.9dB
asrc	96000	192000	1.0001	1	2649.335-41967.003	142.2	NA
xsim-asrc	96000	192000	1.0001	1	2649.335-41967.003	139.4	NA
asrc	176400	192000	1.0001	0	9590.682	124.8	-180.8dB
xsim-asrc	176400	192000	1.0001	0	9590.682	124.6	-180.8dB
asrc	176400	192000	1.0001	1	37192.624-79985.036	129.9	NA
xsim-asrc	176400	192000	1.0001	1	37192.624-79985.036	129.2	NA
asrc	192000	192000	1.0001	0	9598.080	130.3	-197.9dB
xsim-asrc	192000	192000	1.0001	0	9598.080	129.4	-194.9dB

continues on next page

Table 9 – continued from previous page

source	ipRate(Hz)	opRate(Hz)	fDev	ch	signals(Hz)	SNR(dB)	THD(dB)
asrc	192000	192000	1.0001	1	6353.929-84981.403	135.0	NA
xsim-asrc	192000	192000	1.0001	1	6353.929-84981.403	132.9	NA