AI-assisted Design and Synthesis of Continuous-Time ADCs

Internal Report#2
AIR-CHIP Project (PID2022-138078OB-I00)

Author
Francisco Gómez Pulido

Distribution List
Gustavo Liñán-Cembrano
Jose M. de la Rosa

Date: February 19, 2025

Contents

1 Schreirer's toolbox

Date: Febrary 04, 2025

1.1 Introduction

Schreirer's Delta-Sigma Toolbox is a collection of MATLAB functions designed to model and simulate $\Sigma\Delta$Ms. It enables engineers and designers to explore different architectures, evaluate their performance, and optimize their design before hardware implementation.

Key features of the toolbox include:

- Design of noise transfer functions (NTF) for various orders and topologies.
- Spectral analysis to assess quantization noise shaping.
- Time-domain simulation of sigma-delta modulators.
- Evaluation of signal-to-noise ratio (SNR) and dynamic range.
- Design tools for ADC and DAC converters.

1.2 Basic functions of Schreirer's Delta-Sigma Toolbox

The Delta-Sigma Toolbox includes several key functions that help design, analyze, and simulate Sigma-Delta Modulators. Below are some of the most important functions and their purposes:

1.2.1 `synthesizeNTF`: Noise Transfer Function (NTF) design

```
ntf = synthesizeNTF(order, OSR, opt, H_inf, f0)
```

where:

- `order`: order of the $\Sigma\Delta$M. Higher orders provide better noise shaping but may introduce instability.
- `OSR`: oversampling ratio (how much the sampling rate exceeds the Nyquist rate).
- `opt`: flag used to set the placement of the zeros of NFT (0 not optimized, 1 optimized, 2 optimized with at least one zero at band-center, 3 optimized zeros).
- `H_inf`: maximum NTF gain.
- `f0`: center frequency for bandpass modulators, being `f0=0` (default) for LP-$\Sigma\Delta$Ms (low-pass).

Purpose: generates a Noise Transfer Function (NTF) that defines how quantization noise is shaped in a $\Sigma\Delta$M. The generated NTF is required for further analysis and implementation.

Example: `ntf = synthesizeNTF(order=3, osr=64, opt=0, H_inf=1.5, f0=0)`

1.2.2 `realizeNTF`: Convert NTF to a practical structure

```
[a, g, b, c] = realizeNTF(nft, form, stf)
```

where:

- `a, g, b, c`: coefficient matrix needed for further analysis.
  - `a`: $[1 \times n]$ vector corresponding to the feedback/feedforward coefficients from/to the quantizier (n is the modulator order).
  - `g`: $[1 \times n/2]$ vector of resonator coefficients.
  - `b`: $[1 \times (n+1)]$ vector of feed-in coefficients from the modulator input to the each integrator input.
  - `c`: $[1 \times n]$ vector of integrator weights.

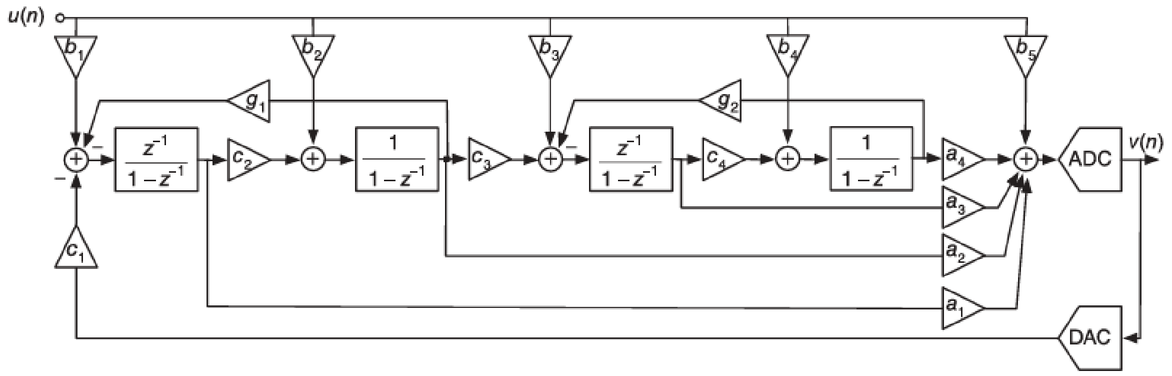


Figure 1: Block diagram of CRFF $\Sigma\Delta$M with the coefficients used in Schreirer's toolbox.

- `ntf`: the Noise Transfer Function obtained from `synthesizeNTF`.
- `form`: defines the implementation type (`CRFB`, `CRFF`, `CIFB`, `CIFF`).
- **stf**: Signal Transfer Function.

Purpose: converts a NTF into a realizable modulator structure that can be implemented in hardware or simulated in MATLAB.

```
Example: [a, g, b, c] = realizeNTF(ntf, form='CRFB', stf=1)
```

1.2.3 `stuffABCD`: Create state-space matrix

```
ABCD = stuffABCD(a, g, b, c, form)
```

where:

- `ABCD`: contains a state-space representation of the $\Sigma\Delta$M loop filter used in the toolbox. The state-space equations used to update the loop-filter state, $x(n)$, and to compute the output of the loop filter, $y(n)$, are given by:

$$x(n+1) = A \cdot x(n) + B \cdot \begin{bmatrix} u(n) \\ v(n) \end{bmatrix} \tag{1}$$

$$y(n) = C \cdot x(n) + D \cdot \begin{bmatrix} u(n) \\ v(n) \end{bmatrix} \tag{2}$$

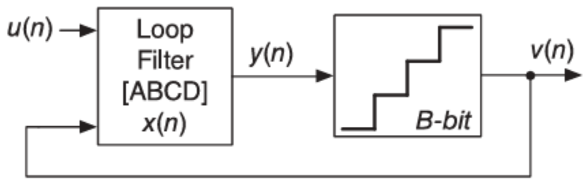


Figure 2: State-space representation of a ΣΔM.

- `a, g, b, c`: coefficients from `realizeNTF`.
- `form`: defines the implementation type (`CRFB`, `CRFF`, `CIFB`, `CIFF`).

Purpose: takes the parameters from `realizeNTF` and generates an ABCD state-space matrix, which represents the modulator in a standard format for control system analysis.

Example: `ABCD = stuffABCD(a, g, b, c, form='CRFB')`

1.2.4 `mapABCD`: Convert state-space model to a different structure

`[a, g, b, c] = mapABCD(ABCD, form)`

where:

- `a, g, b, c`: coefficients.
- `ABCD`: contains a state-space representation of the ΣΔM loop filter used in the toolbox.
- `form`: defines the implementation type (`CRFB`, `CRFF`, `CIFB`, `CIFF`).

Purpose: the opposite transformation of `stuffABCD`, gives the matrix of loop-filter coefficients. Converts state-space models into different modulator structures for hardware or simulation.

Example: `[a, g, b, c] = mapABCD(ABCD, form='CRFB')`

1.2.5 `scaleABCD`: Scale state-space model for stability

`[ABCDs, umax] = scaleABCD(ABCD, nlev, f, xlim, ymax, umax, N)`

where:

- `nlev`: number of quantizer levels.
- `f`: input frequency (normalized to f_s).
- `xlim`: limit set on the states.
- `ymax`: a threshold used to keep the ΣΔM stable so that if the quantizier input exceeds this value, stability is not guaranteed.
- `umax`: maximum allowable input amplitude. It is calculated if not supplied.

Purpose: Adjusts the ABCD state-space model so that the modulator operates within a stable range and doesn't saturate.

Example: `[ABCDs, umax, S] = scaleABCD(ABCD, nlev=2, f=0, xlim=1, ymax=nlev+5, umax, N=1e5, N0=10)`

1.2.6 Example: a 4th-order CRFF LP/BPΣΔM

```
%% Synthesis of NTF (synthesizeNTF)
close all; clear all;
OSR = 50; % Oversampling ratio
L = 4; % Loop-filter order
H_inf = 2.048; % OBG (can be different for each notch)
opt = 2; % Optimum placement of NTF zeros
nlev = 5; % Number of levels in embedded quantizer
tun = 0; % Notch frequency parameter used in synthesizeNTF
notch = 12; % Notch frequency

for i=(1:48) % Synthesize NTF for 100 notch frequencies
    H(i) = synthesizeNTF(L, OSR, opt, H_inf, 0.01*tun);
    stf(i) = H(i); % Set STF with the same poles as NTF
    stf(i).z = [];
    tun = i;
end


%% Loop-filter realization in CRFF form (realizeNTF & stuffABCD)
p = notch + 1;
H = H(p);
form = 'CRFF';
[a, g, b, c] = realizeNTF(H, form);
ABCD = stuffABCD(a, g, b, c, form);


%% Compute and plot NTF, STF (calculateTF, evalTF)
[ntf, stf] = calculateTF(ABCD);
f = linspace(0, 0.48, 1000);
z = exp(2i*pi*f);
ntf_dB = dbv(evalTF(ntf, z));
stf_dB = dbv(evalTF(stf, z));

figure(1);
plot(f, ntf_dB, 'b', f, stf_dB, 'm', 'LineWidth', 1)
xlabel('Normalized frequency');
ylabel('Magnitude');
legend('Noise Transfer Function', 'Signal Transfer Function')


%% Scale loop-filter coefficients (scaleABCD, mapABCD)
% Limits on the states extracted from simulations
x = [0.85 0.75 1.4 1.45]; % Limit on the states (depends on notch)
Np = 1e4;
[ABCDs, umax] = scaleABCD(ABCD, nlev, 0.01*(p-1), x, nlev+5, [], Np);
[as, gs, bs, cs] = mapABCD(ABCDs, form);


```

```
%% Simulation (simulateDSM)
fs = 100e6;
fi = 0.01 * (p-1) * fs;
Ts = 1 / fs;
N = 65536;
i = 1;
sigma = 0.005;
u = 0.1 * sin(2*pi*fi/(fs)*[0:N-1]);
v = simulateDSM(u, ABCDs, nlev);
spec = fft(v.*ds_hann(N))/(N/4);

figure(2);
plot(linspace(0,OSR,N/2+1), dbv(spec(1:N/2+1)));


%% Map loop-filter coefficient onto feasible SC implementation
R = 100; % Rounding Factor
a2 = round([as].*R)./R;
g2 = round([gs].*R)./R;
b2 = round([bs].*R)./R;
c2 = round([cs].*R)./R;

asn_f(p,:) = a2; asn = a2;
gsn_f(p,:) = g2; gsn = g2;
bsn_f(p,:) = b2; bsn = b2;
csn_f(p,:) = c2; csn = c2;
```

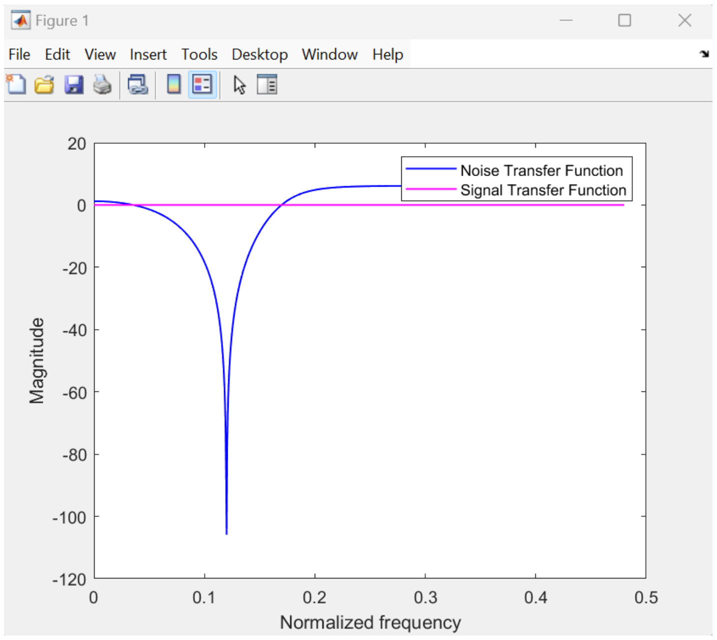


Figure 3: Noise Transfer Function (NTF) and Signal Transfer Function (STF) of a ΣΔM. The NTF, in blue, exhibits a deep notch at a specific normalized frequency, indicating strong noise suppression in that band. Meanwhile, the STF, in magenta, remains relatively flat, suggesting that the signal is preserved across frequencies. This behavior confirms that the modulator effectively reduces quantization noise in the desired band while maintaining the integrity of the input signal.

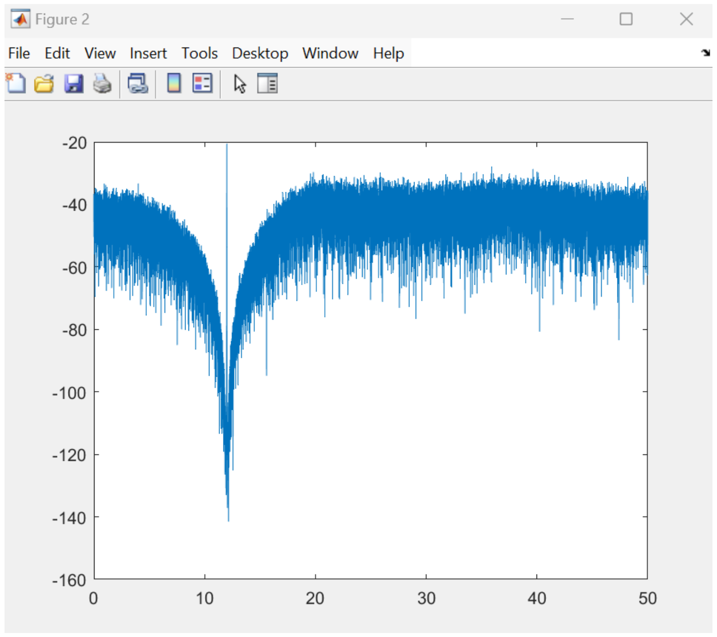
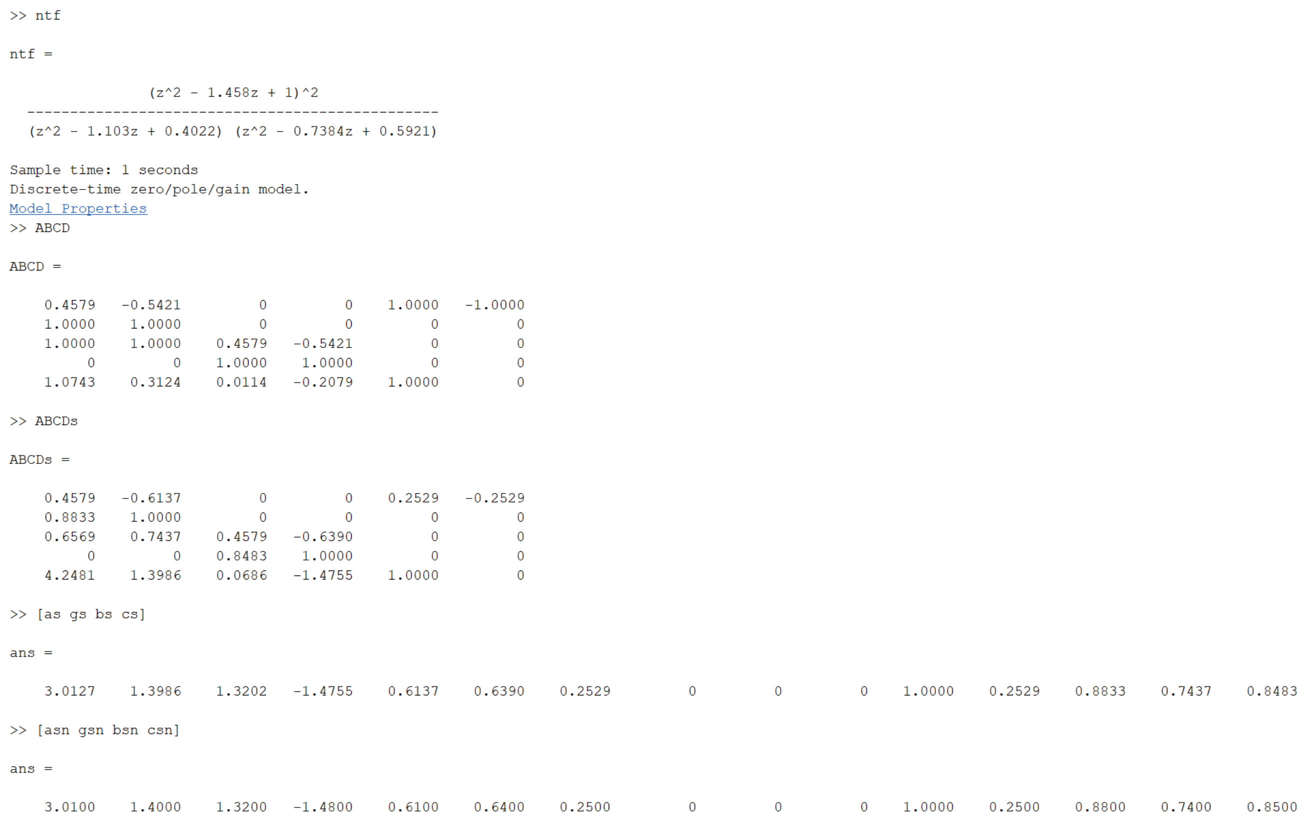


Figure 4: Output spectrum of the ΣΔM with a notch frequency at approximately index 12. Significant attenuation is observed at this frequency, indicating that the modulator is effectively suppressing noise in that band. However, outside the notch, the quantization noise is relatively high, which is characteristic of high-order modulators with noise shaping outside the useful band. The overall shape of the spectrum confirms that the modulator is functioning correctly, fulfilling its role of reducing noise in the band of interest.



```
>> ntf

ntf =

                (z^2 - 1.458z + 1)^2
  -------------------------------------------------
  (z^2 - 1.103z + 0.4022) (z^2 - 0.7384z + 0.5921)

Sample time: 1 seconds
Discrete-time zero/pole/gain model.
Model Properties
>> ABCD

ABCD =

    0.4579   -0.5421         0         0    1.0000   -1.0000
    1.0000    1.0000         0         0         0         0
    1.0000    1.0000    0.4579   -0.5421         0         0
         0         0    1.0000    1.0000         0         0
    1.0743    0.3124    0.0114   -0.2079    1.0000         0

>> ABCDs

ABCDs =

    0.4579   -0.6137         0         0    0.2529   -0.2529
    0.8833    1.0000         0         0         0         0
    0.6569    0.7437    0.4579   -0.6390         0         0
         0         0    0.8483    1.0000         0         0
    4.2481    1.3986    0.0686   -1.4755    1.0000         0

>> [as gs bs cs]

ans =

    3.0127    1.3986    1.3202   -1.4755    0.6137    0.6390    0.2529         0         0         0    1.0000    0.2529    0.8833    0.7437    0.8483

>> [asn gsn bsn csn]

ans =

    3.0100    1.4000    1.3200   -1.4800    0.6100    0.6400    0.2500         0         0         0    1.0000    0.2500    0.8800    0.7400    0.8500
```

Figure 5: The displayed results show the Noise Transfer Function (NTF) of a ΣΔM, along with its ABCD state-space representation before and after scaling. The NTF confirms the expected noise shaping behavior, with a notch at a specific frequency. The ABCD and ABCDs matrices represent the system's state-space coefficients before and after applying scaling adjustments to optimize dynamic range and stability. The quantized coefficients ([asn, gsn, bsn, csn]) demonstrate slight rounding variations due to finite precision implementation, which is crucial for practical circuit realization while maintaining the desired filtering characteristics.

1.3 Optimization-based high-level synthesis of ΣΔMs

The aim of this section is to explain how the optimization process can be automatized. The metodology presented is focused on the use of SIMSIDES.

1.3.1 Combining behavioral simulation and optimization

This subsection explains how behavioral simulations are combined with optimization algorithms to enhance the design process.

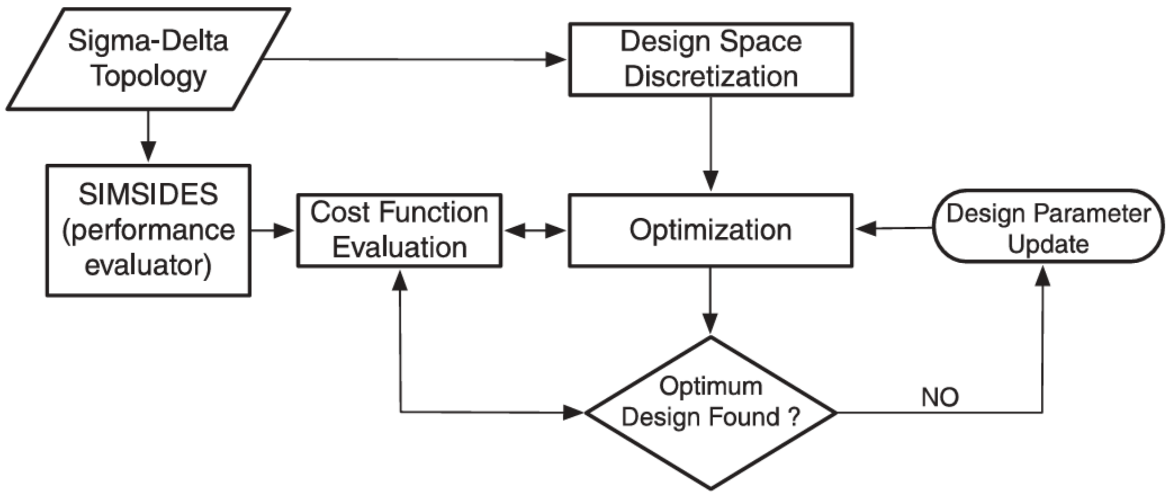


Figure 6: Flow diagram of the optimization-based synthesis of ΣΔMs. The starting point is the modulator topology, which can be synthesized using Schreirer's toolbox. Here, the design parameters are the building-block specifications (the nonideals parameters used to model the main error mechanisms that affect the ΣΔM's performance and define the electrical specifications of its subcircuits: integrators, comparators, DAC elements...). Considering arbitrary initial conditions, a set of perturbations of the design parameters is generated by the optimizer. With the new desing parameters, appropiate simulations are carried out to evaluate the modulator performance and the process is repeated in an iterative way until the cost function (based on the performance metrics) is optimized.

SIMSIDES can be combined with any of the optimization methods provided with MATLAB.

1.3.2 Using simulated annealing as optimization engine

Simulated Annealing (SA): Probabilistic optimization algorithm inspired by the physical process of annealing in metallurgy, where a material is heated and then slowly cooled to reduce defects and reach a stable, low-energy state.

This method is particularly useful in ΣΔM design because it can efficiently explore complex, multi-dimensional design spaces while avoiding local minima, leading to globally optimal solutions. This section describes how Simulated Annealing (SA) is applied to the high-level synthesis of ΣΔMs, detailing its principles, algorithmic steps, advantages, and practical implementation considerations.

Simulated annealing algorithm:

1. **Initialize the system** with an initial design solution (a set of ΣΔM parameters), define an **initial temperature** T_0 (which controls the probability of accepting worse solutions) and set the **cooling schedule**, which gradually reduces T over iterations.

2. For each iteration, a **new candidate solution** is generated by making small **random modifications** to the current design parameters. The **cost function** (representing modulator performance) is evaluated for both the current and new solutions.

3. **Acceptance criteria**: if the new solution improves the cost function, it is **automatically accepted**. If it is worse, it may be **accepted with probability** $P = e^{-\frac{\triangle E}{T}}$, where $\triangle E$ is the difference in cost function values and T is the current temperature. This probabilistic acceptance allows the algorithm to escape local minima.

4. **Cooling process**: the temperature T is gradually reduced following a cooling schedule (for example, exponential decay: $T = \alpha T$, with α close to 1). As T decreases, the probability of accepting worse solutions reduces, leading to convergence.

5. **Termination condition**: The algorithm stops when T reaches a predefined threshold or when no further improvements are observed over multiple iterations.

The steps we need to follow is:

1. Define parameters in a MATLAB script. This script sets the values of all model parameters, except those used as design variables in the optimization. Ideal values are given for those design parameters not considered in the optimization.

2. Define the ranges of the design variables, the targets and the solver numerical methods (simulated annealing, called FRIDGE) in SIMSIDES.

3. Set the design objectives and targets (for example: SNR, IBN, THD...). Optionally, a set of design constraints (optionally weighted to give priority).

4. Finally, all the information provided by the user is generated by SIMSIDES in the form of a text file. The file is used as the input netlist by the optimizer.

1.3.3 Combining SIMSIDES with MATLAB optimizers

An updated version of SIMSIDES includes an optimization menu that guides designers through the main steps to set up and run an optimization. Designers can choose different ways to automate the high-level synthesis of a given ΣΔM, the behavioral model of which has been built using SIMSIDES. Two options:

1. Launch the SIMULINK Design Optimization tool, from designers can define the optimization problem (design variables, ranges, objectives, constraints...), select the optimization algorithm, run the optimization and analyze the results.

2. In SIMSIDES the optimization menu allows designers to customize their optimization problem by defining all the required pieces of information: the name of the SIMSIDES model, the MATLAB script and the simulation parameters, the number of design variables, and the optimization method.

1.4 Lifting method and hardware acceleration to optimize CT-ΣΔMs

Getting a precise simulation involves increasing CPU time. For instance, using SIMSIDES, a high-order (third-order, fourth-order) CT-ΣΔM considering all circuit errors, typically requires a few seconds to run a simulation. If thousands simulations are carried out in an optimization-based synthesis procedure, the CPU time might increase up to several hours.

In order to overcome these limitations, authors have proposed the ***lifting method*** to accelerate the simulation of CT-ΣΔMs by several orders of magnitude.

For ΣΔMs, the lifting method is used to enhance the loop filter structure, reducing redundant computations and making it more hardware-friendly. This method enables a more compact realization of the filter coefficients and efficient numerical representation, which is crucial for FPGA (Fiel Programmable Gate Array) implementations.

1.4.1 Hardware emulation of CT-ΣΔMs on an FPGA

FPGAs usually include the possibility of emulating some other useful ΣΔM building blocks, so that, in principle, an arbitrary CT-ΣΔM topology can be emulated in hardware.

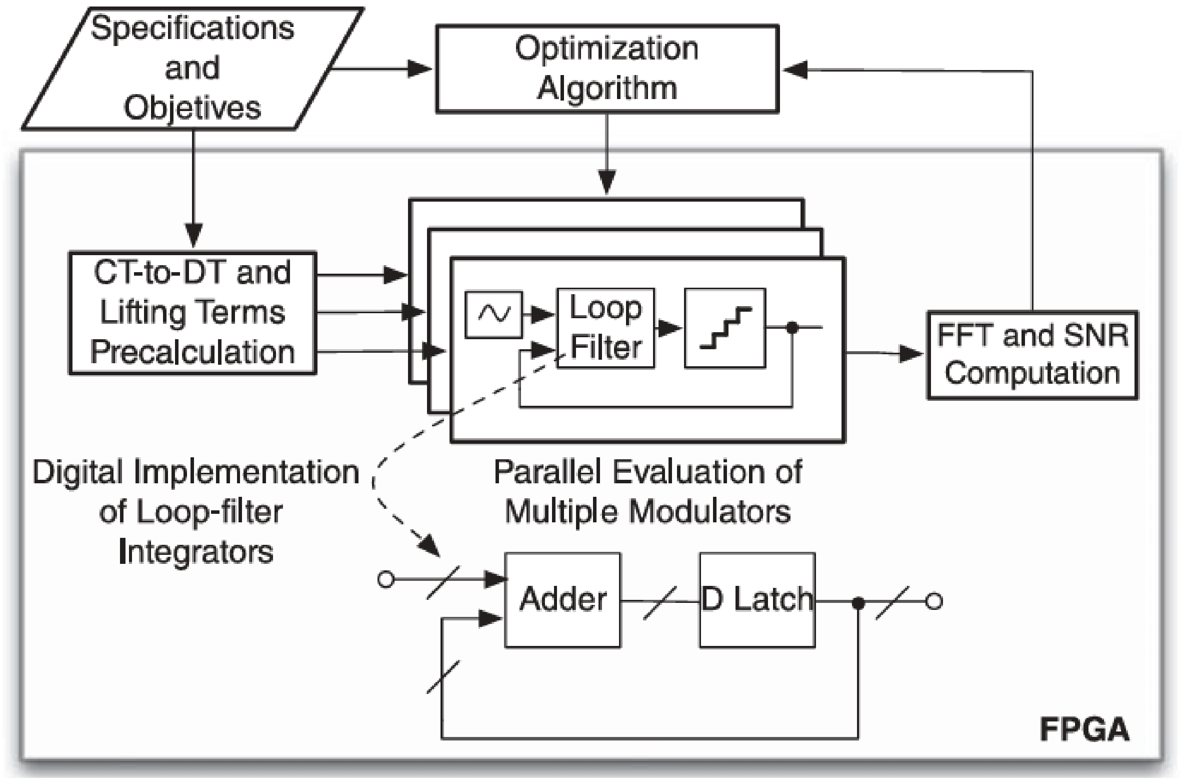

Specifications and Objetives
Optimization Algorithm
CT-to-DT and Lifting Terms Precalculation
Loop Filter
FFT and SNR Computation
Digital Implementation of Loop-filter Integrators
Parallel Evaluation of Multiple Modulators
Adder
D Latch
FPGA

Figure 7: Emulation of CT-ΣΔMs on FPGAs for synthesis purposes.

1.4.2 GPU-accelerated computing of CT-ΣΔMs

Following the same philosophy, another powerful approach to the high-level synthesis of CT-ΣΔMs were proposed, based on fast simulation on a GPU, instead of on a CPU. The method allowed ten or even hundreds of thousands of simulations to be evaluated per second, thus exploring a whole design space of millions of design in less than a minute.

2 SIMSIDES

Date: Febrary 05, 2025

To run SIMSIDES, we need to type in a MATLTAB console:

```
addpath('path_to_SIMSIDES')
```

```
addpath(genpath('path_to_SIMSIDES'))
```

```
simsides
```

Then, the main menu will be opended. The interface looks like:

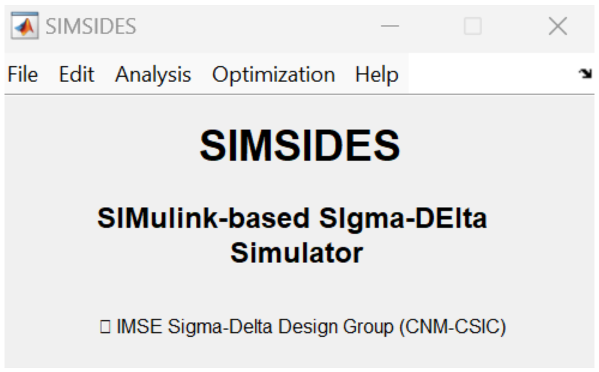
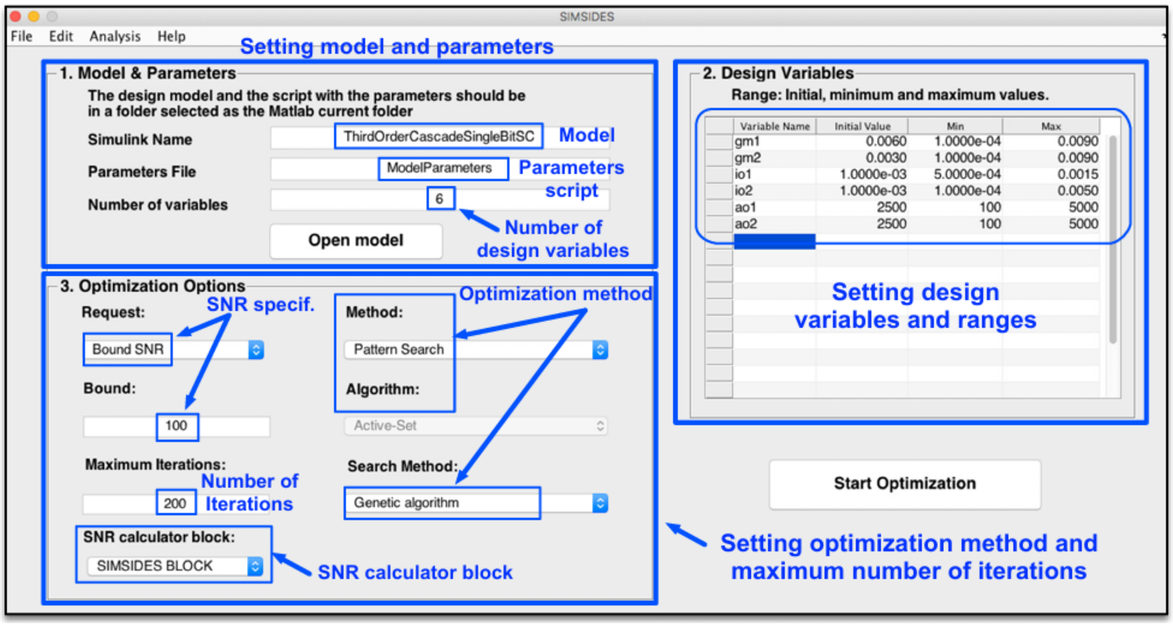

SIMSIDES
File Edit Analysis Optimization Help
SIMSIDES
SIMulink-based SIgma-DElta Simulator
IMSE Sigma-Delta Design Group (CNM-CSIC)

Figure 8: SIMSIDES' main menu.

In case of any doubt, look for SIMSIDES' manual.

2.1 Optimization interface

In the main menu, go to `Optimization` → `Simulink Design Optimization` → `Response Optimization Script`. Fill:



1. Model & Parameters

The design model and the script with the parameters should be in a folder selected as the Matlab current folder

Simulink Name: ThirdOrderCascadeSingleBitSC

Parameters File: ModelParameters

Number of variables: 6

2. Design Variables

Range: Initial, minimum and maximum values.

Variable Name	Initial Value	Min	Max
gm1	0.0060	1.0000e-04	0.0090
gm2	0.0030	1.0000e-04	0.0090
io1	1.0000e-03	5.0000e-04	0.0015
io2	1.0000e-03	1.0000e-04	0.0050
ao1	2500	100	5000
ao2	2500	100	5000

3. Optimization Options

Request: Bound SNR

Bound: 100

Maximum Iterations: 200

SNR calculator block: SIMSIDES BLOCK

Method: Pattern Search

Algorithm: Active-Set

Search Method: Genetic algorithm

Figure 9: User interface for setting optimization.

Take into consideration:

- **Simulink name**: the model must be defined in the same directory (or subdirectory). The used in the example is:

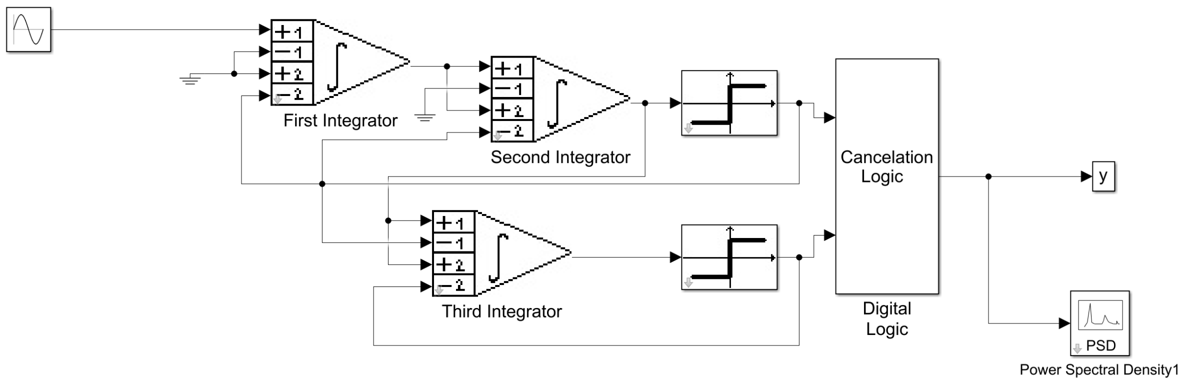


Figure 10: Example of simulink file: Third-order Cascade Single-bit SC.

- **Parameters file**: the parameters script must be defined in the same directory (or subdirectory). The used in the example is:

```
        %% SDM parameters:
        % Sampling Frequency(fs), Input Frequency (fi), Sampling Time (Ts)
        % OverSampling Ratio (OSR=M); Number of points (N)
        fs=5.12e6; fi=5e3; Ts=1/fs; M=128; N=65536; Ain = 0.5;
        % Model parameters
        kt=0.026*1.6e-19; % Boltzmann constant

```

```matlab

%% First Integrator's parameters
Cint1=24e-12; % integration capacitor For gain=1
Cs11=6e-12; % sampling capacitor (branch 1)
Cs21=6e-12; % sampling capacitor (branch 2)
innoise1=0; % rms value of the input equivalent noise
ao1=2.5e+03; % open-loop OTA DC gain
gm1=10e-03; % transconductance
io1=5.0e-03; % maximum OTA output current
ron1=60; % sampling switch-on resistance


%% Second- and Third- Integrators
Cint2=3e-12;
Cs12=1.5e-12;
Cs22=1.5e-12;
innoise2=0;
ao2=2.5e+03;
gm2=10e-03;
io2=5.0e-03;
ron2=650;


%% Common integrator parameters
temp=175; % temperature
osp=2.7; % output swing
cnl1=0; % capacitor first-order non-linear coef.
cnl2=25e-6; % capacitor second-order non-linear coef.
avnl1=0; % DC gain first-order non-linear coef.
avnl2=15e-2; % DC gain second-order non-linear coef.
avnl3=0; % DC gain third-order non-linear coef.
avnl4=0; % DC gain fourth-order non-linear coef.
cpar1=0.6e-12; % parasitic (opamp) input capacitance
cpar2=0.6e-12;
cload=2.28e-12; % opamp (intrinsic) load capacitance
% Comparators
vref=2; % DAC reference voltage
hys=30e-3; % comparator hysteresis


%% Ideal
% £ao2 = 1.946401e+03       2.5e+03
% £ao1 = 1.742154e+03       2.5e+03
% £gm1 = 5.736949e-03       10e-03
% £gm2 = 1.186684e-04       10e-03
% £io1 = 1.304425e-03       5.0e-03
% £io2 = 2.775390e-03       5.0e-03
```

2.2 Tutorial example: using SIMSIDES to model and analyze $\Sigma\Delta$Ms

Date: Febrary 06, 2025

In this section we illustrate the use of SIMSIDES to analyze the main features of a $\Sigma\Delta$M. The modulator under study is a third-order cascade 2-1 DT-$\Sigma\Delta$M with a single-bit quantization in both stages.

2.2.1 Creating the cascade 2-1 ΣΔM block diagram in SIMSIDES

The purpose of this section is to build this architecture:

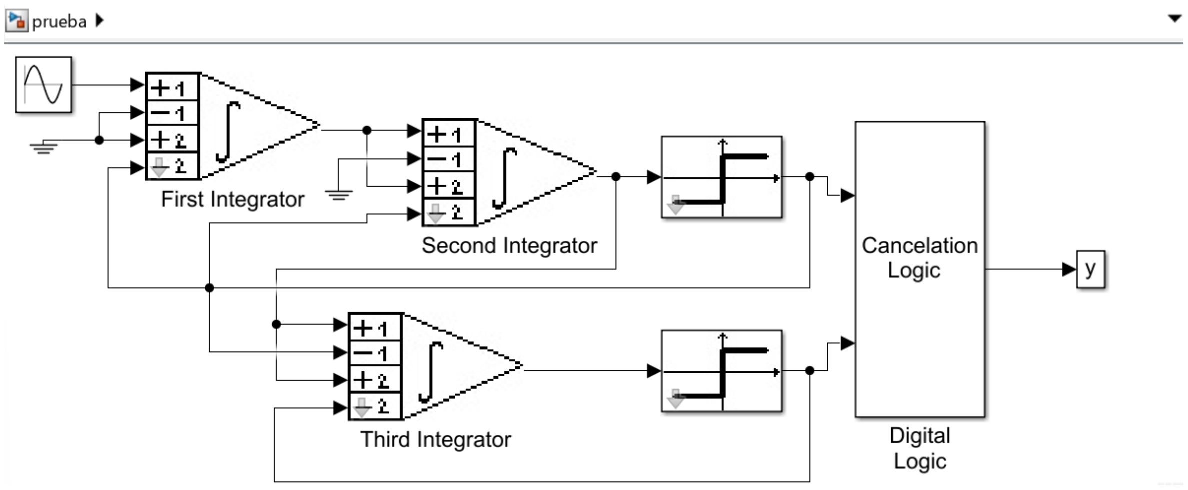


Figure 11: SIMSIDES block diagram of the ΣΔM architecture.

1. From SIMSIDES' main menu: `File` → `New Architecture`. Introduce the name of the architecture.
2. To include the integrators and comparators, go to SIMSIDES' main menu, `Edit` → `Add block`, and select the model libraries. Then, drag and drop the objects. Note: you can also open each library from the design interface (`Open` and select the required library).
3. Add remaining blocks from SIMULINK model library. `Edit` → `Simulink Library`, and drag and drop the required objects. In my case, I don't know why but I can't find these libraries, so the solution is to type in the MATLAB console (the name of my architecture was `prueba.slx`):
```
open_system('prueba')
add_block('simulink/Sources/Sine Wave', 'prueba/Sine Wave')
add_block('simulink/Sources/Ground', 'prueba/Ground')
add_block('simulink/Discrete/Unit Delay', 'prueba/Unit Delay')
add_block('simulink/Discrete/Discrete Filter', 'prueba/Discrete Filter')
add_block('simulink/Sinks/To Workspace', 'prueba/To Workspace')
```
4. If needed, we can create new blocks. To do it, in the designer interface go to `Modeling` → `Create Subsystem` (or `New` → `Subsystem`). We can rename it and create inside the logic of the block.

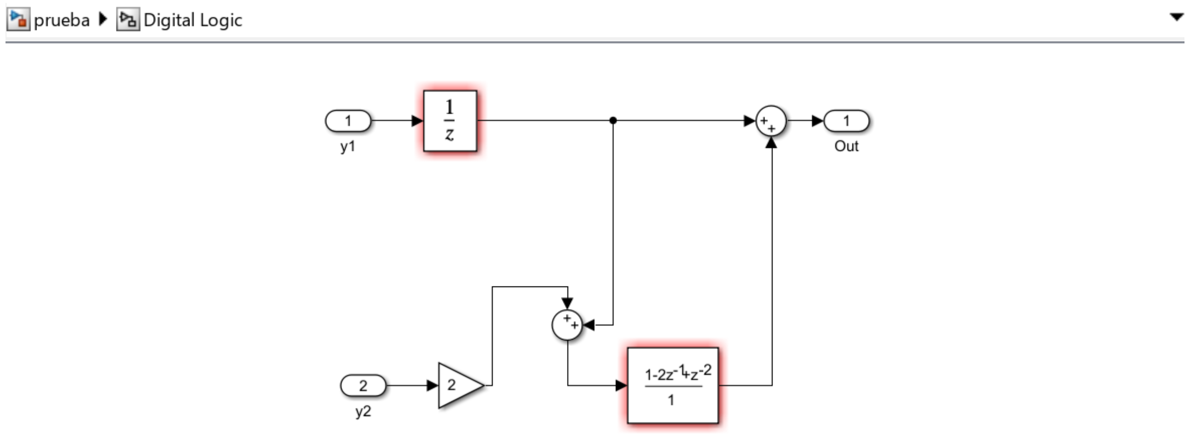


Figure 12: Create subsystem.

5. Connect properly all the blocks included.

If we want to use a imported architecture, we can select it from SIMSIDES' menu (`File` → `Open Architecture`).

2.2.2 Setting model parameters

The modulator parameters and model parameters required to simulate the block diagram can be either set up in the MATLAB console or saved in an M-file. In this case, the file is the shown in the previous section.

In addition to these model parameters, simulation parameters must be set up to run a simulation. Go to `Model Settings` (`Configuration Parameters`):

- `Solver` → `Simulation time` → `Start time: 0.0`, `Stop time: (N-1)*Ts`.
- `Solver` → `Solver selection` → `Type: Variable-step`.
- `Solver` → `Solver details` → `Max step size: auto`.

2.2.3 Computing the output spectrum

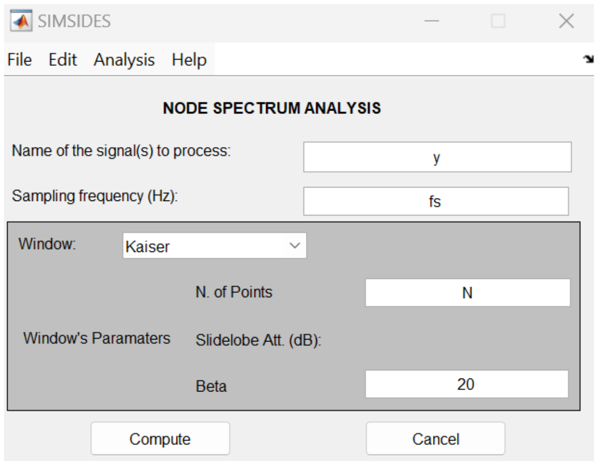
Once we have the parameters file, we must type in the command window:

```
run('modulator_params.m')
```

We can see that all parameters has been defined in the workspace.

Then, we can simulate the modulator from the SIMULINK menu: `Simulate` → `Run`.

When the simulation finishes, in SIMSIDES' menu go to `Analysis` → `Node Spectrum Analysis`. Then define the parameters requested:



NODE SPECTRUM ANALYSIS

Name of the signal(s) to process: y

Sampling frequency (Hz): fs

Window: Kaiser

Window's Paramaters

N. of Points: N

Slidelobe Att. (dB):

Beta: 20

Figure 13: Setting parameters in Node spectrum analysis in SIMSIDES.

After that, we click on `Plot` and the output spectrum is displayed:

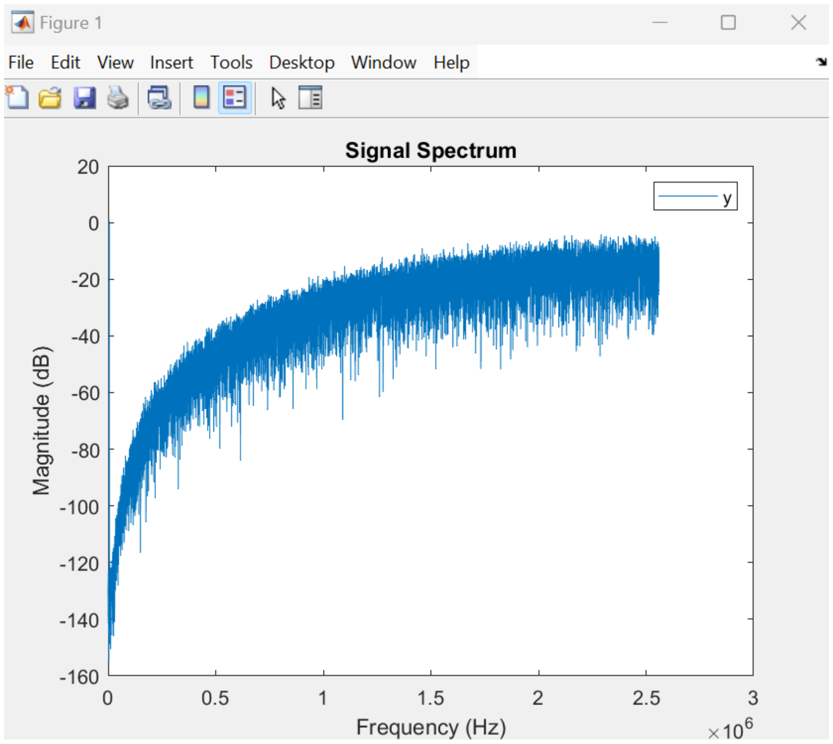
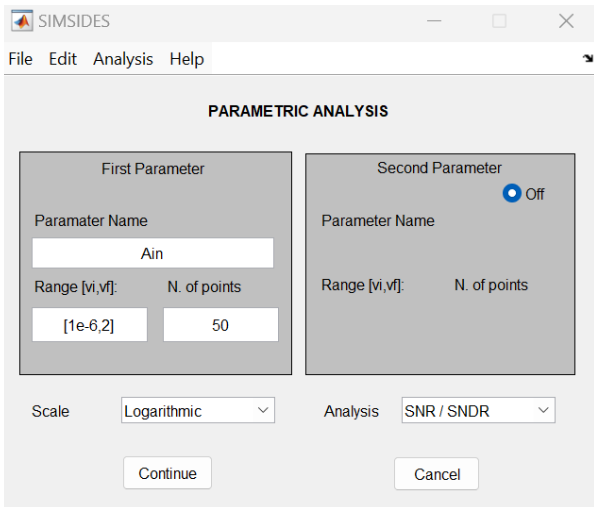


Figure 14: Signal spectrum.

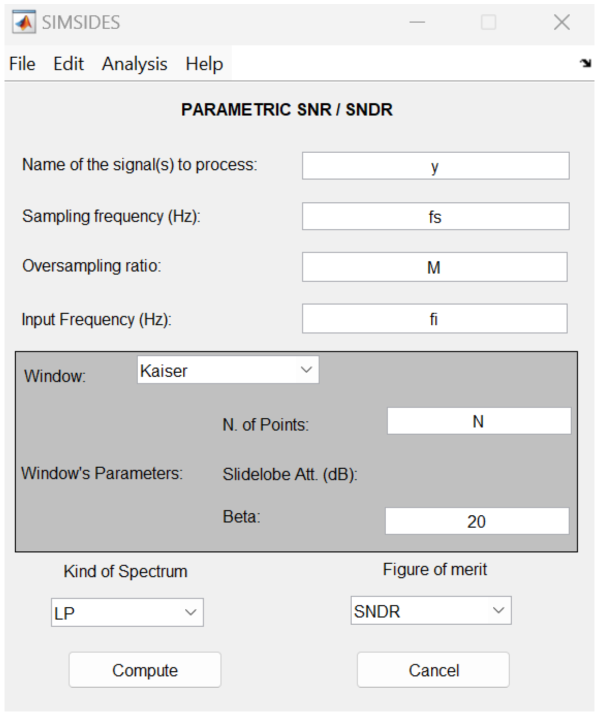
2.2.4 SNR versus input amplitude level

SIMSIDES' menu → `Analysis` → `Parametric Analysis`:



Paramater Name: Ain
Range [vi,vf]: [1e-6,2]
N. of points: 50
Parameter Name:
Range [vi,vf]:
N. of points
Scale: Logarithmic
Analysis: SNR / SNDR

Figure 15: Setting parameters in Parametric analysis in SIMSIDES (1).



Name of the signal(s) to process: y
Sampling frequency (Hz): fs
Oversampling ratio: M
Input Frequency (Hz): fi
Window: Kaiser
Window's Parameters:
N. of Points: N
Slidelobe Att. (dB):
Beta: 20
Kind of Spectrum: LP
Figure of merit: SNDR

Figure 16: Setting parameters in Parametric analysis in SIMSIDES (2).

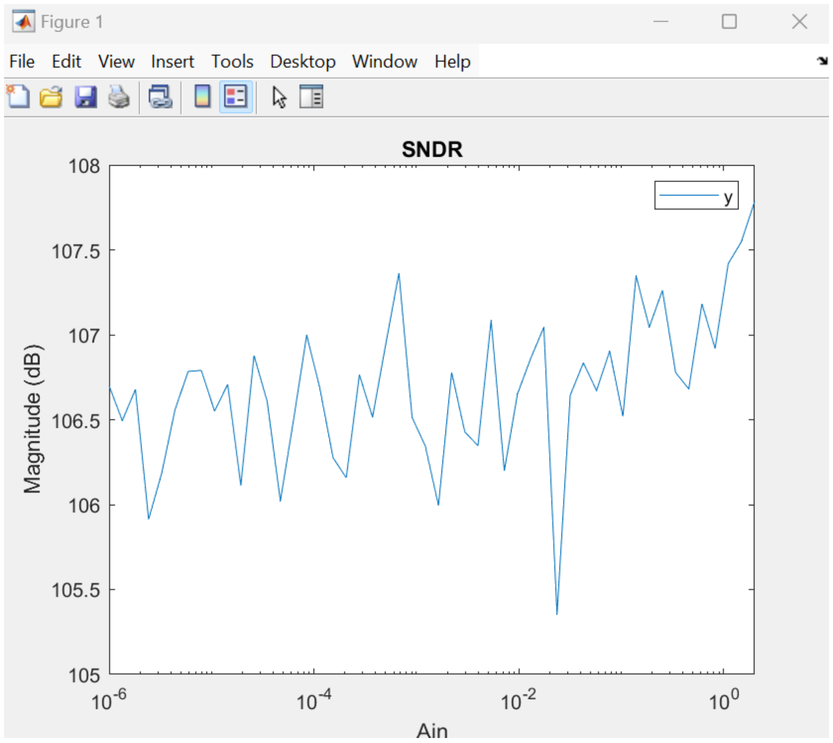
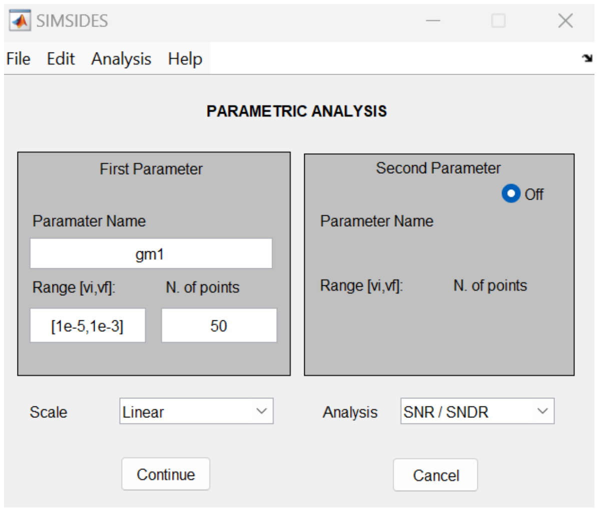


Figure 17: SNDR vs input amplitude level of the ΣΔM.

2.2.5 Parametric analysis considering only one parameter

SIMSIDES' menu → `Analysis` → `Parametric Analysis`:



First Parameter

Paramater Name: gm1

Range [vi,vf]: [1e-5,1e-3] N. of points: 50

Second Parameter [✓ Off]

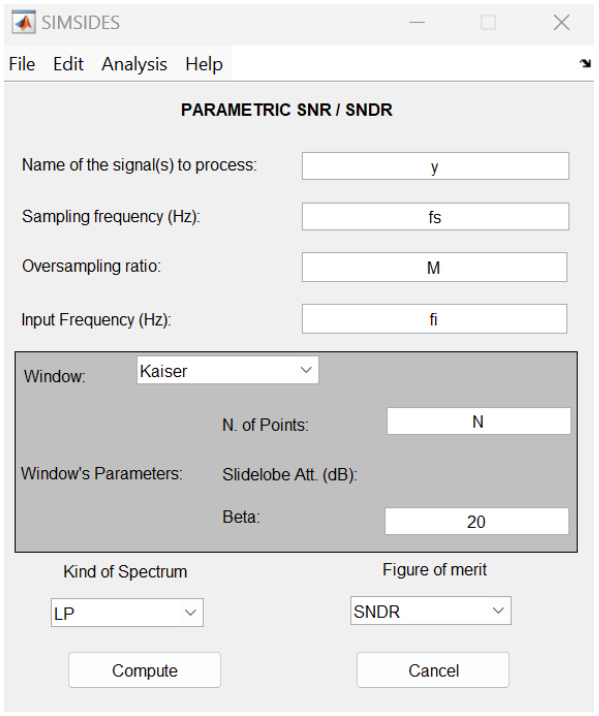
Parameter Name

Range [vi,vf]: N. of points

Scale: Linear

Analysis: SNR / SNDR

Figure 18: Setting parameters in Parametric analysis in SIMSIDES (3).



PARAMETRIC SNR / SNDR

Name of the signal(s) to process: y

Sampling frequency (Hz): fs

Oversampling ratio: M

Input Frequency (Hz): fi

Window: Kaiser

Window's Parameters:

N. of Points: N

Slidelobe Att. (dB):

Beta: 20

Kind of Spectrum: LP

Figure of merit: SNDR

Figure 19: Setting parameters in Parametric analysis in SIMSIDES (4).

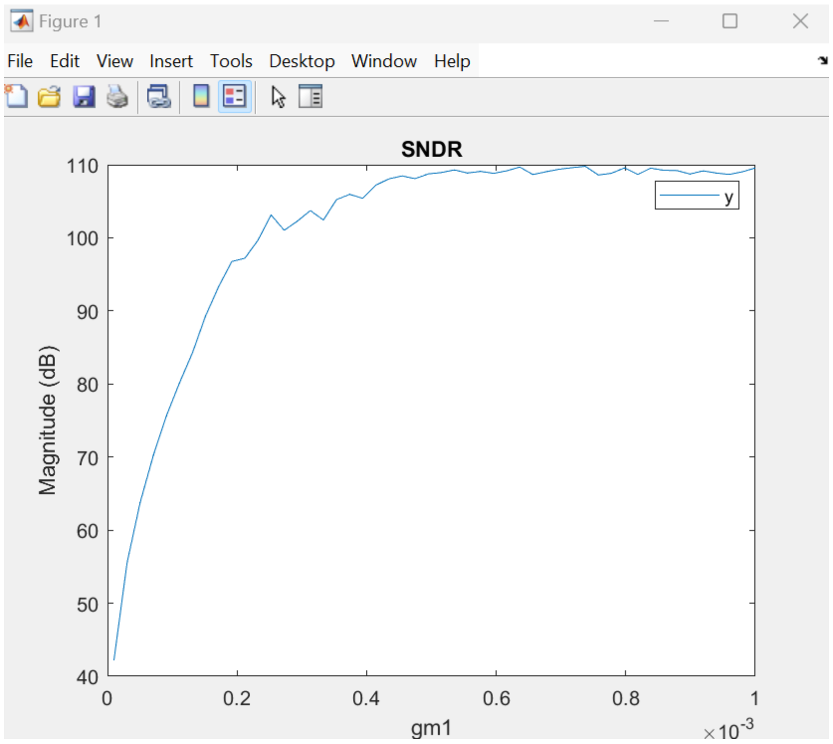
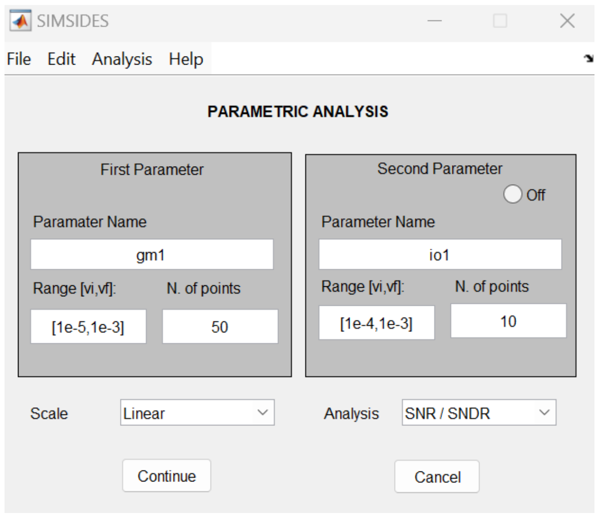


Figure 20: SNDR vs gm1 (g_m).

2.2.6 Parametric analysis considering only one parameter

SIMSIDES' menu → `Analysis` → `Parametric Analysis`:



PARAMETRIC ANALYSIS

First Parameter

Paramater Name: gm1

Range [vi,vf]: [1e-5,1e-3]

N. of points: 50

Second Parameter

Off

Parameter Name: io1

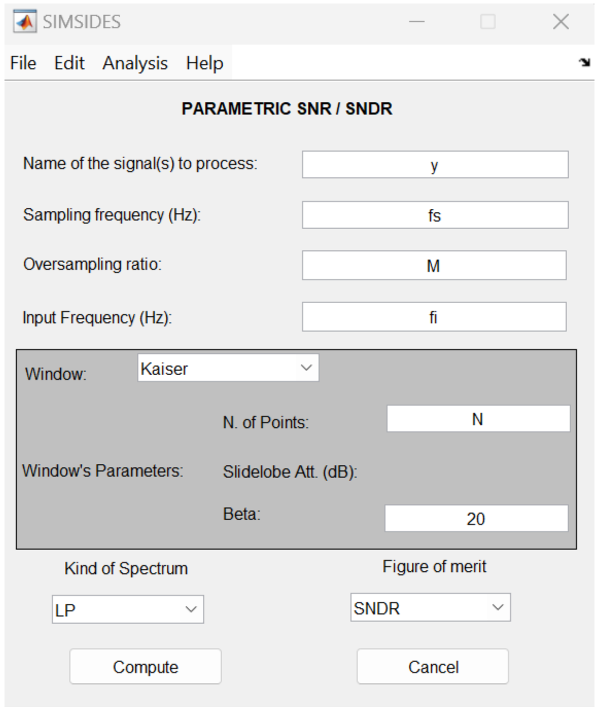
Range [vi,vf]: [1e-4,1e-3]

N. of points: 10

Scale: Linear

Analysis: SNR / SNDR

Figure 21: Setting parameters in Parametric analysis in SIMSIDES (5).



PARAMETRIC SNR / SNDR

Name of the signal(s) to process: y

Sampling frequency (Hz): fs

Oversampling ratio: M

Input Frequency (Hz): fi

Window: Kaiser

N. of Points: N

Window's Parameters:

Slidelobe Att. (dB):

Beta: 20

Kind of Spectrum: LP

Figure of merit: SNDR

Figure 22: Setting parameters in Parametric analysis in SIMSIDES (6).

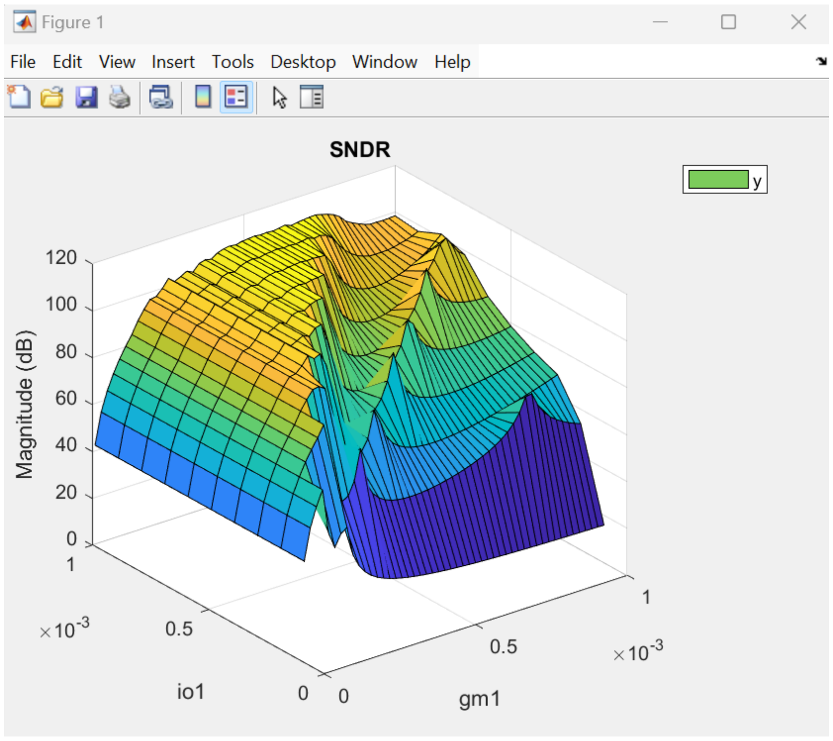


Figure 23: SNDR vs gm1, io1 (g_m and I_{o1}).

2.3 Analyzing ΣΔMs in SIMSIDES

As shown in the previous section, simulation output data can be processed in SIMULINK using the `Analysis` menu in SIMSIDES. It includes the following submenus:

- `Node Spectrum Analysis`: computes and plots the FFT magnitude spectrum of a given signal.
- `Integrated Power Noise`: calculates and plots the IBN within a given band signal bandwidth.
- `SNR/SNDR`: computes the SNR and/or SNDR within the band of interest, considering both LP- and BP-ΣΔMs.
- `Harmonic Distorsion`: computes dynamic harmonic distortion figures, like THD and intermodulation distortion figures.
- `Histograms`: represents histograms and analyzes the input/output swing in ΣΔM building blocks.
- `INL/DNL`: calculates static harmonic distorsion.
- `MTPR`: computes multi-tone power ratio (MTPR).
- `Parametric Analysis`: allows to simulate the impact of a given model parameter on the performance of ΣΔMs.
- `Monte Carlo Analysis`: does Monte Carlo simulations.

Some of them are shown in the previous section. For more information, search into the SIMSIDES' manual, in which there are explications about all methods.