CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Behavioral modeling of low-frequency noise in switched-capacitor circuits using Python
Authors	Kalogiros, Spyridon;Salgado, Gerardo;McCarthy, Kevin;O'Connell, Ivan
Publication date	2022-08-05
Original Citation	Kalogiros, S., Salgado, G., McCarthy, K. and O'Connell, I. (2022) 'Behavioral modeling of low-frequency noise in switched-capacitor circuits using Python', 2022 20th IEEE Interregional NEWCAS Conference (NEWCAS), Quebec City, QC, Canada, 19-22 June, pp. 446-449. doi: 10.1109/NEWCAS52662.2022.9842076
Type of publication	Conference item
Link to publisher's version	10.1109/NEWCAS52662.2022.9842076
Rights	© 2022, European Union. Published by IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Download date	2026-11-06 15:31:18
Item downloaded from	https://hdl.handle.net/10468/13772

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Behavioral Modeling of Low-Frequency Noise in Switched-Capacitor Circuits Using Python

Spyridon Kalogiros[1,2], Gerardo Salgado[1,2], Kevin McCarthy[2] and Ivan O'Connell[1,2]
[1]*Microelectronic Circuits Centre Ireland, Tyndall National Institute, Cork, Ireland*
[2]*University College Cork, Department of Electrical and Electronic Engineering, Cork, Ireland*
email: spyridon.kalogiros@umail.ucc.ie

***Abstract*—In precision circuits validating the performance in the presence of low-frequency noise is particularly challenging especially at transistor level, as long simulations are required to observe the low frequency performance. However, running such system-level simulations is rarely practical at transistor level as these simulations can take days to weeks to complete. This work presents a high-level model in Python for generating low-frequency noise which can be used for validating the low-frequency performance of a design in a timely manner. Simulation times can be reduced from days to minutes, enabling designers to achieve a high-level simulation coverage. With Python and NumPy this can be achieved using open-source software tools at no cost.**

Keywords—Python, System-Level Design, Behavioral Modeling, Flicker Noise, Switched-Capacitor Integrators

I. Introduction

In the design of precision switched capacitor circuits it is essential to simulate and verify the resultant performance in the presence of low-frequency noise sources such as flicker or *1/f* noise. To accurately simulate low frequency noise in the kHz range requires simulations ms long, whereas to simulate low frequency in the 10's Hz region requires simulations that run 100 ms. However, for circuits operating from a MHz clock this requires 100000 or more clock cycles and this increases with increasing clock frequency. While transistor-level simulations do provide the necessary accuracy and precision, running system simulations of these durations results in simulation times of days and weeks [1]. This makes it extremely difficult to achieve a reasonable level of simulation coverage in a reasonable timeframe. Hence, many designs are often sent for fabrication without having a satisfactory level of simulation coverage in relation to their performance in the presence of *1/f* noise, which often results in unnecessary redesigns. To address this, MATLAB and Simulink have gained broad adoption at the system level [2]. The Schreier Toolbox [3] is widely adopted at the system level in the design and verification of Delta-Sigma Modulator architectures. However, MATLAB is a commercial tool, which means that not every designer has access to it, especially outside an academic environment. Python, which has recently become one of the most popular programming languages [4], is being widely adopted in the test and measurement space due its robustness and vast repository of packages [5]. This has seen it being adopted as the language of choice in the area of Analog IC Design, one such example is the Berkley Analog Generator [6]. The absence of any license fees is an important aspect in democratising analogue IC Design and enabling designers to undertake rapid low-cost system design and verification.

This paper presents a high-level noise model for a Switched-Capacitor Integrator in Python, which enables the designer to accurately model low frequency *1/f* noise for the first time in a timely manner. This paper is organized as follows. Section II presents an algorithmic analysis in Python of low-frequency noise modeling on a single-ended Switched-Capacitor Integrator, pointing out the main low-frequency component, the flicker (*1/f*) noise. Section III introduces modeling in Python of a Delta-Sigma Modulator, example which contains Switched-Capacitor Integrators in the loop filter, where low-frequency noise is also present. Finally, Section IV summarizes the main points of this work.

II. Modeling of Flicker Noise Corner In Switched-Capacitor Integrators

Switched-Capacitor Integrators, as illustrated in Fig. 1(a) below, are the key building block in many discrete time precision circuits and in particular Delta-Sigma ADCs. From a noise perspective, there are the wideband noise sources which include the sampled thermal noise and the amplifier thermal noise, as shown in Fig. 1(b). These are often lumped together and treated as $\sqrt{\frac{kT}{C}}$ noise [7]. In addition, there is shaped flicker noise resulting from the trapping/detrapping phenomena with the MOS transistors, which also needs to be considered. However, to date, this is typically modelled as an offset within the operational amplifier at the system level.

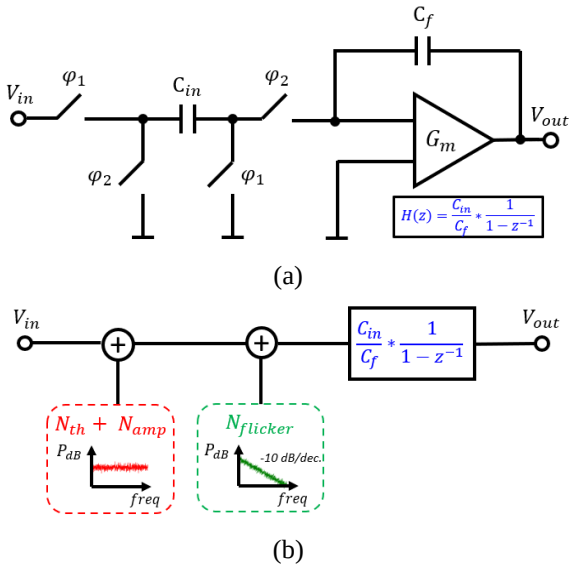


Figure 1: (a) Single-ended Switched-Capacitor Integrator (b) Noise equivalent circuit

A. Thermal Gaussian Noise Generation

The sampled thermal noise can be modelled using a random Gaussian (or normal) distribution, with a standard deviation of $\sqrt{\frac{kT}{C_{in}}}$ and a mean of zero. The NumPy Python package has such a function built in, *normal(mean, std_dev, N)* [8], which takes 3 parameters and returns an array of length *N*, with a mean of *mean* and a standard deviation of *std_dev*. Hence, a differential Switched-Capacitor Integrator can be modelled as having an input referred sampled thermal

noise standard deviation of $\sigma = \sqrt{\frac{4kT}{C_{in}}}$, where C_{in} is the sampling capacitor. To build an equivalent single-ended model of the switched capacitor integrator in Python requires that the added thermal noise voltage is divided by 2 to maintain the same Signal to Noise Ratio as the differential circuit. Hence, the added thermal noise is now modelled as:

$$sigma_Nthermal = \sqrt{\overline{N_{th}}^2} = \sqrt{\frac{4kT}{C_{in}}}/2 \simeq \sqrt{\frac{kT}{C_{in}}} \quad (1)$$

The first part of the proposed Python code, to generate an array of random white thermal noise, *Nthermal*, is shown in Fig. 2 below, for a 1 pF sampling capacitor:

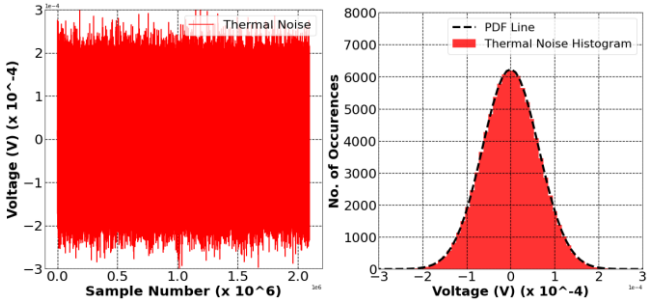
```
from numpy        import sqrt, log10, zeros
from numpy        import pi, sin, round
from numpy.random import normal
from numpy.fft    import fft, fftfreq

N = 2**21     # Array size and No. of frequency bins
k = 1.38e-23  # Boltzmann's constant
T = 300       # Absolute Temperature (°K)
C = 1e-12     # Sampling Capacitor 1 pF

mean_Nthermal  = 0
sigma_Nthermal = round(sqrt(k*T/C), 6)
Nthermal = normal(mean_Nthermal, sigma_Nthermal, N)
```

Figure 2 Proposed Python script (part 1): Thermal noise generation

The resulting array of created thermal noise is plotted in Fig. 3(a). A histogram of the array of the generated values is shown in Fig. 3(b) confirming the normal distribution shape.



(a) *(b)*

Figure 3 (a) Gaussian thermal noise [Nth] (b) Histogram

B. Shaped Low-Frequency Noise Generation

As discussed in the previous section, the *normal* NumPy function produces a normal distribution which results in white noise, with a flat frequency spectrum. However, to approach the -10 dB/dec. of flicker noise roll-off, it is required that the added noise is shaped and that the Power Spectral Density (PSD) varies with frequency and is no longer flat. To achieve this, it's necessary to filter the random noise, such that it's shaped. One such filter to achieve this is the Low-Pass Filter proposed in [9] and given by:

$$H(z) = \frac{a}{b - cz^{-1}} \quad (2)$$

which can be rewritten as:

$$\frac{Nrandom}{Nflicker} = \frac{a}{b - cz^{-1}} \quad (3)$$

where *Nrandom* is the random white noise and *Nflicker* is the resultant shaped low-frequency noise. This can be further rewritten, in time domain, as:

$$Nflicker[i] = \frac{1}{b}\left(aN_{random}[i] + cN_{flicker}[i-1]\right) \quad (4)$$

making it a lot easier to implement the filter in time domain in Python. The coefficients *a*, *b* and *c* and are all set to 1. However, the resultant magnitude of the noise is determined by the magnitude of the random input noise, N_{random}. Sampled thermal noise is always expressed in terms of the total integrated noise in µV. Low-frequency flicker noise is typically expressed in terms of a noise corner, below which the shaped *1/f* noise dominates and above which the white thermal noise dominates. Therefore, the standard deviation, *sigma_Nrandom*, of N_{random}, the unfiltered random noise used to realise the *1/f* noise, is given by:

$$sigma_N_{random} = \frac{sigma_Nthermal}{scaling_factor} \quad (5)$$

The *scaling_factor* is used to scale *sigma_Nthermal* to achieve the desired corner frequency and can be shown as:

$$scaling_factor = \frac{F_{Nyq}}{2 * F_{corner}} \quad (6)$$

where F_{corner} is the desired *1/f* noise corner frequency and F_{Nyq} is the Nyquist $\frac{F_s}{2}$ frequency. The denominator of equation (6) is formed in a way to further improve the normalization and the proper placing of the corner frequency. For instance, a selected F_{Nyq} of 10 MHz and *Fcorner* of 5 kHz create a scaling factor equal to 10 MHz / 10 kHz, or equivalently, 1000. This, in turn, will divide the *sigma_Nrandom* at a value equal to *sigma_Nthermal*/1000, or equivalently, from 64 µV to 64 nV, changing the spectral crossing point of both *Nrandom* and *Nthermal FFTs* to the desired one. The second part of Python code, to realize flicker noise, is shown in Fig. 4 below:

```
Nflicker        = zeros(N)
Fs              = 20e6
fft_freq        = fftfreq(N, 1/Fs)
Fnyq            = max(fft_freq)
Fcorner         = 5000
scaling_factor  = round(Fnyq/(2*Fcorner))
sigma_Nrandom   = sigma_Nthermal/scaling_factor
Nrandom         = normal(0.0, sigma_Nrandom, N)
a               = 1
b               = 1
c               = 1

for i in range(1, N):
      Nflicker[i] = (a*Nrandom[i]+c*Nflicker[i-1])/b
```

Figure 4 Proposed Python script (part 2): Flicker noise generation

The code shown in Fig. 4, results in the $N_{flicker}$ noise being generated from the converted input noise using the transfer function of (2) and it is illustrated in Fig. 5 below:

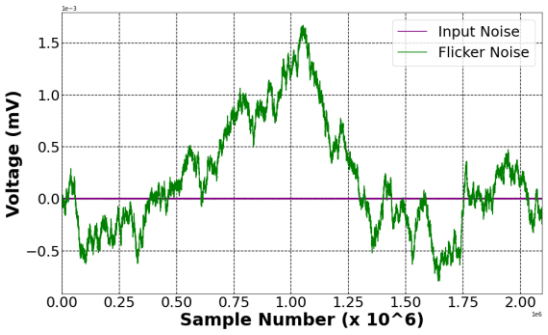
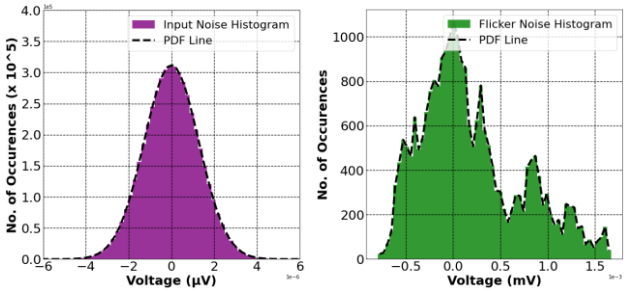


Figure 5 Random input and generated flicker noise



(a) (b)

Figure 6 (a) Input noise histogram (b) Generated flicker noise histogram

Fig. 6 above clearly shows that the generated flicker noise is not Gaussian.

Illustrated in Fig. 7 below are the FFTs of the thermal and flicker noise, on the same plot. The FFT of the random input noise which is converted to flicker is also plotted. The figure shows clearly that, firstly, the random input & thermal noise FFTs are flat across the range of frequencies as expected, and secondly, the desired flicker noise corner frequency (here in this example equal to 5 kHz) is modelled properly due to the equations (5) and (6) that force the average of flicker noise FFT to cross the average of thermal noise FFT at this spectral point:

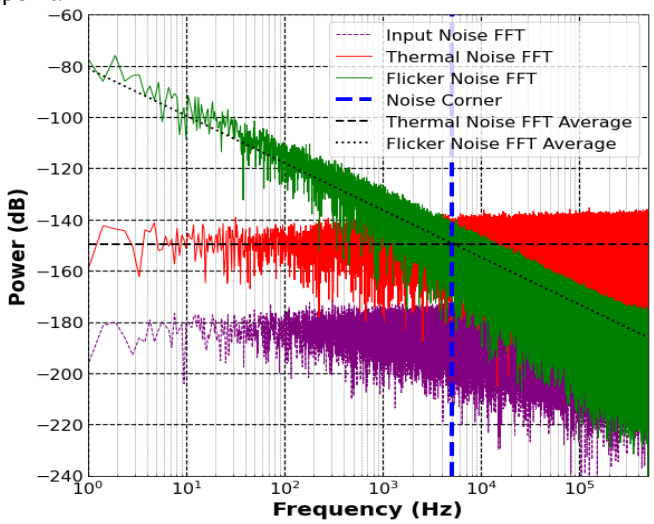


Figure 7 FFTs of the generated thermal, input and flicker noise

III. MODELING OF FLICKER NOISE CORNER IN DELTA-SIGMA MODULATORS

To simulate the robustness of the proposed Python model, a Second-Order Delta-Sigma Modulator is modelled in Python based on the modulator structure that is proposed in [10], since Switched-Capacitor Integrators are broadly used to implement the loop filter. A single-bit version of the Delta-Sigma Modulator from [10], which is used in the present analysis, is illustrated in Fig. 8 below:

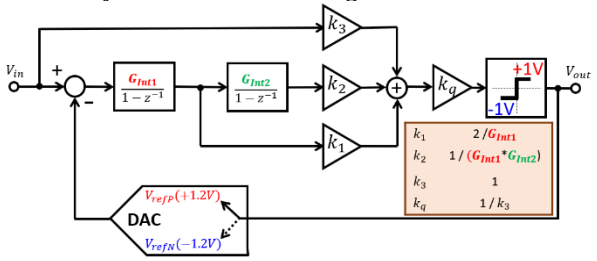


Figure 8 Wideband Second-Order Delta-Sigma Modulator

The architecture contains the loop filter coefficients k_1, k_2 the feedforward coefficient k_3, the quantizer gain k_q, and the dynamic range scaling methodology for proper loop stability is also shown. A Delta-Sigma Modulator like the one above can be transferred into a Python environment by using the differential equations that are forming it [11]. Starting from the Switched-Capacitor Integrators, their generic transfer function in z domain is illustrated on Fig. 1. In time domain, integrator 1 and 2 equations can be expressed from the modulator of Fig. 8 as follows:

$$Int1[n] = G_{Int1}(V_{in}[n] - V_{out}[n-1]) + Int1[n-1] \quad (7)$$

$$Int2[n] = G_{Int2}(Int1[n]) + Int2[n-1] \quad (8)$$

A third important differential equation is the input of the ADC unit, the quantizer, which can be expressed below:

$$Q[i] = k_q * (k1 * Int1[i] + k2 * Int2[i] + k3 * Vin[i]) \quad (9)$$

To implement the modulator in Python, the Python code of Fig. 9 below is added as a third part (after Fig. 4), injecting also the flicker and thermal noise quantities:

```
ampl                = 0.7
VrefP               = 1.2
VrefN               =-1.2
Ncycles             = 2**7
BW                  = (Ncycles/N)*Fs
Tin                 = 1/BW
Ts                  = 1/Fs
samples_per_period  = round(Tin/Ts)
Vin                 = zeros(N)
diff_in             = zeros(N)
Int1                = zeros(N)
Int2                = zeros(N)
comp_in             = zeros(N)
comp_out            = zeros(N)
dac_out             = zeros(N)

Gint1 = 1.0; Gint2 = 1.0

k1 = 2/Gint1; k2 = 1/(Gint1*Gint2); k3 = 1; kq = 1/k3

for i in range(N):

  Vin[i] = ampl*sin(2*pi/samples_per_period*i)

  if (i>=1):
    diff_in[i] = Nthermal[i] + Nflicker[i] + Vin[i] -
dac_out[i-1]
    Int1[i]    = Gint1*diff_in[i] + Int1[i-1]
    Int2[i]    = Gint2*Int1[i] + Int2[i-1]
    comp_in[i] = kq*(k1*Int1[i] + k2*Int2[i] +
k3*Vin[i])

    if (comp_in[i]>0):
      comp_out[i] = 1.0
    else:
      comp_out[i] =-1.0

    if (comp_out[i]==-1.0):
      dac_out[i] = VrefN
    else:
      dac_out[i] = VrefP

freq_Xaxis   = fft_freq[:N//2]
comp_out_fft = fft(comp_out[:N])/N
Nflicker_fft = fft(Nflicker[:N])/N
Nthermal_fft = fft(Nthermal[:N])/N
comp_out_dB  = 20*log10(abs((comp_out_fft[:N//2])))
Nflicker_dB  = 20*log10(abs((Nflicker_fft[:N//2])))
Nthermal_dB  = 20*log10(abs((Nthermal_fft[:N//2])))
```

Figure 9 Proposed Python script (part 3): Second-Order Feed-Forward Delta-Sigma Modulator implementation

The first group of lines consist the initialization of the necessary parameters. The first step is the generation of an input sine wave of a 0.7 V amplitude and a frequency around the 1 kHz range, a band that is affected by the low-frequency flicker noise. The input signal is generated with a formula that places its frequency directly on a frequency bin to avoid spectral leakage on the FFT plot, and that is the following:

$$BW = \left(\frac{Ncycles}{N}\right) * Fs \quad (10)$$

where $Ncycles$ is the number of periods of the signal, N the total number of frequency bins and Fs the sampling frequency.

A sampling frequency of 20 MHz is introduced (from Fig. 4) and a total number N of frequency bins equal to 2^{21} is selected (from Fig. 2), a number that in transistor-level designs takes significant amount time to simulate. The code of Fig. 9 continues with the calculation of the input signal's period *Tin*, the sampling period *Ts*, the number of samples that are taken per one input signal's period and the initialization of the modulator's main signal arrays. Finally, the Delta-Sigma Modulator's closed-loop system is simulated with all the differential equations in a *for* loop which is executed for N times.

Illustrated in Fig. 10 below is the FFT of the modulator output in the presence of both thermal and flicker, using a 20 MHz clock. It is also showing the standard 40 dB/dec. noise shaping associated with a second order modulator. The presence of the thermal noise is clearly evident by the flat spectrum between 1 kHz to 30 kHz, whereas the *1/f* noise is clearly evident below 1 kHz, causing the rising noise floor. In total, three main regions of interest are shown on the output FFT, the *1/f* noise, the flat thermal noise and the quantization noise shaping. The same flicker noise corner of 5 kHz is also simulated in this example:

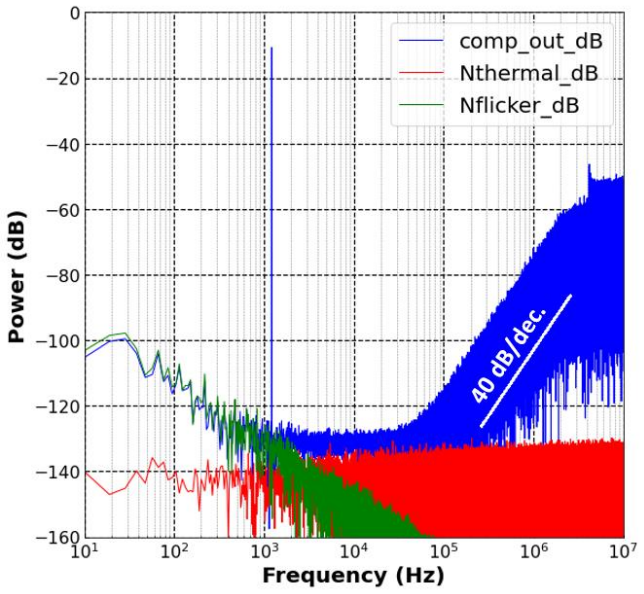


Figure 10 Modulator output FFT with FFTs of flicker and thermal noise

IV. Conclusions

The low-frequency nature of flicker noise requires that hundreds of thousands of clock cycles are required to observe the impact at low frequencies, which is rarely feasible at transistor level. The open-source nature of Python eliminates any license constraints associated with commercial tools, while also helping to democratizing the design of CMOS designs. This enables designers to add low-frequency noise to their design and ultimately observe how the noise propagates through the system. The approach of using equation (2) to shape the white noise, results in a 20 dB/dec. roll-off compared to the 10 dB/dec. resulting from flicker noise. This deviation is deemed acceptable, as it over-estimates the impact of flicker noise at lower frequencies, ensuring that a systems robustness is fully tested and validated. This increases designers' confidence that their design will achieve the system-level specifications.

Acknowledgements

This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) and is co-funded under the European Regional Development Fund under Grant Number 13/RC/2077

References

[1] E. Chang, N. Narevsky, K. Settaluri and E. Alon, "BAG: A Process-Portable Framework for Generator-based AMS Circuit Design," *2019 IEEE Custom Integrated Circuits Conference (CICC)*, 2019, pp. 1-20

[2] Gerardo Molina Salgado, Daniel O'Hare, Ivan O'Connell, "Recent Advances and Trends in Noise Shaping SAR ADCs", *Circuits and Systems II: Express Briefs IEEE Transactions on*, vol. 68, no. 2, pp. 545-549, 2021

[3] https://www.mathworks.com/matlabcentral/fileexchange/19-delta-sigma-toolbox

[4] https://statisticstimes.com/tech/top-computer-languages.php

[5] E. Alon, K. Asanović, J. Bachrach and B. Nikolić, "Invited: Open-Source EDA Tools and IP, A View from the Trenches," 2019 56th ACM/IEEE Design Automation Conference (DAC), 2019, pp. 1-3

[6] J. Han, W. Bae, E. Chang, Z. Wang, B. Nikolić and E. Alon, "LAYGO: A Template-and-Grid-Based Layout Generation Engine for Advanced CMOS Technologies," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 3, pp. 1012-1022, Mar. 20121

[7] R. Schreier, J. Silva, J. Steensgaard and G. C. Temes, "Design-oriented estimation of thermal noise in switched-capacitor circuits," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 52, no. 11, pp. 2358-2368, Nov 2005

[8] https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html

[9] G. M. Salgado, D. O'Hare and I. O'Connell, "Modeling and Analysis of Error Feedback Noise-Shaping SAR ADCs," *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9180995.

[10] J. Silva, U. Moon, J. Steensgaard and G.C. Temes, "Wideband Low-Distortion Delta-Sigma ADC topology", ELECTRONICS LETTERS 7th June 2001 Vol. 37 No. 12, pp. 737-738

[11] Shanthi Pavan, Richard Schreier, Gabor C. Temes, Understanding Delta-Sigma Data Converters", Second Edition, IEEE Press, Wiley, 2017.