AN AUTOMATIC ARCHITECTURE GENERATOR FOR SIGMA-DELTA MODULATORS CONSIDERING COMPONENT NON-IDEALITIES

by

Ömer Yetik

B.S., in Electrical and Electronics Engineering, Boğaziçi University, 2005

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University
2007

AN AUTOMATIC ARCHITECTURE GENERATOR FOR SIGMA-DELTA MODULATORS CONSIDERING COMPONENT NON-IDEALITIES

APPROVED BY:

Prof. Dr. Günhan Dündar .....
(Thesis Supervisor)

Asst. Prof. Dr. Şenol Mutlu .....

Prof. Dr. A. C. Cem Say .....

DATE OF APPROVAL: 17.08.2007

You'll always stay right by my side...

ACKNOWLEDGEMENTS

First of all I would like to express my gratitudes to my supervisor Prof. Günhan Dündar for his endless support and encouragement throughout my thesis.

Then, I would like to thank to my committee members Asst. Prof. Şenol Mutlu and Prof. A. C. Cem Say for their kindness and their contributions to my thesis.

I would like to thank to my research partner and lifelong friend Muharrem Orkun Sağlamdemir for his contributions to my thesis and for his friendship.

Also I would like to thank to my friends and collagues Cumhur Ozan Yalçın, Yücel Altuğ, and Nafiz Polat Ayerden for their support and innovative contributions. Throughout my life, I will always need them.

I am very greatful to Selçuk Talay and all the members of BETA for their help and support.

And, I would like to express my special thanks to my family for their endless patience, support and everything. Without them nothing would be possible.

Also I would like to thank to my friends Işıl Burcu Barla, Selen Özgür and Gönenç Seçil Tarakçıoğlu for their friendship and their support both throughout the thesis and at my presentation.

Last but not the least, I would like to extend a final "thank you" to the legend, Metallica, for their great songs and "never-ending" support that accompanied me through sleepless nights.

ABSTRACT

AN AUTOMATIC ARCHITECTURE GENERATOR FOR SIGMA-DELTA MODULATORS CONSIDERING COMPONENT NON-IDEALITIES

In this thesis, a tool generated in MATLAB environment for automatic modeling and architecture synthesis of sigma-delta modulators is introduced. The tool is capable of generating the parametric signal and noise transfer functions of a sigma-delta modulator of any order automatically. After generating the transfer functions, it optimizes both the signal and noise transfer functions of the system simultaneously in such a way to realize a desired frequency response.

Most important of all, this tool is capable of taking component non-idealities into account and optimizing the coefficients so as to compensate the effects of non-idealities on the system. The component non-idealities taken into consideration consist of the integrator non-idealities such as the switched capacitor mismatches, integrator leakage due to finite DC gain of the op-amp, etc. since they can directly be mapped to the transfer functions of the system.

The tool uses some criteria in generating the architectures such as minimization of the number of signal paths of the architecture in order to obtain minimum possible complexity, avoiding of single closed loops without a delay and forcing all the coefficients to be real numbers. Also, another important aspect of the tool is that it spans the whole solution space according to these criteria and returns a set of several parametric solutions.

ÖZET

SİGMA-DELTA KİPLEYİCİLER İÇİN ELEMAN İDEALSİZLİKLERİNİ DE HESABA KATAN BİR OTOMATİK MİMARİ GELİŞTİRİCİ

Bu tezde, sigma-delta kipleyiciler için MATLAB ortamında geliştirilmiş bir otomatik modelleme ve yapı sentezleme aracı tanıtılmaktadır. Bu araç, her hangi bir dereceden bir sigma-delta kipleyicinin parametrik sinyal ve gürültü transfer fonksiyonlarını otomatik olarak geliştirebilmektedir. Transfer fonksiyonlarını geliştirdikten sonra, sistemin hem sinyal hem de gürültü transfer fonksiyonlarını, istenen belirli bir frekans cevabını gerçekleyebilecek şekilde aynı anda optimize edebilmektedir.

Herşeyden önemlisi, bu araç, eleman idealsizliklerini hesaba katabilme ve mimari katsayılarını bu idealsizliklerin sistem üzerindeki etkilerini yok edebilecek şekilde optimize edebilme yetisine sahiptir. Hesaba katılan eleman idealsizlikleri, doğrudan sistemin transfer fonksiyonlarına aktarılabiliyor olmalarından ötürü, anahtar kondansatör uyumsuzlukları, işlevsel yükseltecin sonlu DC kazancına bağlı inetgrator kaçağı, vb. gibi integrator idealsizliklerinden oluşmaktadır.

Araç, yapıları oluştururken mümkün olan en az karmaşıklığı sağlayabilmek için yapıdaki sinyal yollarının sayısının en aza indirilmesi, gecikmesiz kapalı döngülerin ortaya çıkmasının engellenmesi, bulunan katsayıların reel olması gibi bir takım kriterler kullanmaktadır. Ayrıca aracın en önemli yanlarından birisi de bahsedilen kriterlere göre tüm çözüm uzayını tarıyor ve pek çok parametrik çözüm kümesi döndürüyor olmasıdır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv
ABSTRACT v
ÖZET vi
LIST OF FIGURES ix
LIST OF TABLES xi
LIST OF SYMBOLS/ABBREVIATIONS xii
1. INTRODUCTION 1
1.1. Background and Problem Definition 1
1.2. Outline of the Thesis 4
2. OVERSAMPLING SD A/D CONVERTERS 5
2.1. Basic Concepts 5
2.1.1. Oversampling and Quantization Noise 6
2.1.2. SD Modulator 7
2.1.3. Commonly Used Figures of Merit 9
2.1.3.1. Signal-to-Noise Ratio (SNR) 9
2.1.3.2. Dynamic Range (DR) 9
2.1.3.3. Effective Resolution (B) 10
2.2. SD Modulator Architectures 10
2.2.1. Second Order SD Modulator 11
2.2.2. Single-loop High-order Modulators 12
2.2.2.1. Stability Considerations 13
2.2.3. High-order SD Modulator Architectures 13
2.2.3.1. Single-loop Modulators 14
2.2.3.2. Cascade Modulators 14
2.2.4. Multi-bit Quantization SD Modulators 15
3. SD MODULATOR NON-IDEALITIES 17
3.1. Clock Jitter 17
3.2. Integrator Noise 18
3.2.1. Switch Thermal Noise 18

3.2.2. Op-amp Noise 19
3.3. Integrator Non-idealities 20
3.3.1. Finite DC Gain 20
3.3.2. Bandwidth and Slew-rate 20
3.3.3. Saturation 21
4. THE AUTOMATIC ARCHITECTURE GENERATOR 22
4.1. Generation of the Parametric (Symbolic) Transfer Functions 22
4.2. Generation of All Possible SD Modulator Architectures 26
4.3. Inclusion of the Non-idealities 29
4.3.1. Extraction of Non-ideality Parameters from a SC Integrator . . 29
4.3.2. Design of a High-Performance Operational Amplifier 31
5. APPLICATION OF THE AUTOMATIC ARCHITECTURE GENERATOR 33
5.1. Generation of Standard Second Order Architectures Ignoring the Component Non-idealities 33
5.2. Generation of Standard Second Order Architectures Including Component Non-idealities 35
5.2.1. Ideal Op-amp Case 36
5.2.2. High-Performance Op-amp Case 36
5.2.3. Low-Performance Op-amp Case 37
5.2.4. Comparison of the Results 38
5.3. Further Examples on Application of the Tool 40
5.3.1. Realizing a Band-Pass STF and a Notch NTF 40
5.3.2. Realizing a Delay-less STF ($STF = 1$) and Standard Second Order NTF 41
5.3.3. Application of the Tool on a Third Order System 43
6. CONCLUSIONS AND FUTURE WORK 46
APPENDIX A: THE SOURCE CODES 50
REFERENCES 65

LIST OF FIGURES

Figure 1.1. Application range of several modulator architectures in the resolution-speed plane 2

Figure 2.1. Block diagram of a SD A/D converter 5

Figure 2.2. Basic structure of an SD modulator and a quantizer 7

Figure 2.3. A first order SD modulator 9

Figure 2.4. A second order SD modulator 11

Figure 2.5. Noise transfer functions of a first and second order SD modulator as a function of the frequency 12

Figure 2.6. L-th order Lee-Sodini SD Modulator 14

Figure 2.7. Block diagram of a cascade SD modulator 15

Figure 2.8. Summary of SD modulator architectures 16

Figure 3.1. A differential SC integrator 18

Figure 4.1. The generic standard second order SD modulator architecture comprising all possible feedback and feedforward paths 23

Figure 4.2. The block level netlist of the generic second order architecture . . 24

Figure 4.3. The flowchart of the automatic architecture generator tool 28

Figure 4.4. A fully differential folded cascode op-amp 31

Figure 5.1. The standard second order SD modulator 34

Figure 5.2. An unconventional second order SD modulator architecture 35

Figure 5.3. The PSD plots for standard second order architectures 35

Figure 5.4. The PSD plots for the low-performance op-amp case 38

Figure 5.5. A third order SD modulator architecture comprising all possible feedbac and feedforward paths 43

LIST OF TABLES

Table 5.1. Architectures Found for the Standard Second Order Response . . . 34

Table 5.2. SNR values in dB for Different Simulation Scenarios 39

Table 5.3. Architectures Found for the BP STF and Notch NTF 41

Table 5.4. Architectures Found for Delayless STF and High-pass NTF 42

Table 5.5. Architectures Found for the Third Order System 44

LIST OF SYMBOLS/ABBREVIATIONS

C_f	Feedback capacitor
C_f	Sampling capacitor
e_T	Switch thermal noise voltage
f_d	Nyquist frequency
f_in	Input signal frequency
f_s	Sampling frequency
M	Oversampling ratio
$S_E(f)$	Noise power spectral density
P_Q	In-band noise power
V_{cm}	Common-mode voltage
V_{in}	Input voltage
V_n	Total RMS noise voltage referred to op-amp input
v_o	Output voltage
α	Integrator leakage
δ	Error in the sampling time instance / Sampling time uncertainty
Δ	Quantizer stepsize
$\Delta\tau$	Standard deviation of the sampling time uncertainity
Φ_1	Clock phase 1
Φ_2	Clock phase 2
τ	Integrator time constant
A/D	Analog-to-digital
B	Effective resolution
BP	Band-pass
BW	Bandwidth
CAD	Computer aided design
D/A	Digital-to-analog

DAC	Digital-to-analog converter
DR	Dynamic range
DSP	Digital signal processor/processing
GBW	Gain-bandwidth product
NTF	Noise transfer function
OSR	Oversampling ratio
PSD	Power spectral density
SC	Switched-capacitor
SD	Sigma-delta
SNR	Signal-to-noise ratio
SR	Slew-rate
STF	Signal transfer function
TSNR	Signal-to-(noise distortion) ratio

1. INTRODUCTION

1.1. Background and Problem Definition

Although real world signals are analog, it is often desirable to convert them into the digital domain using an A/D converter because digital signals are much more efficient than analog signals in terms of both data transmission and storage. Intricate processing of the signal may also necessitate analog to digital conversion since such processing is only feasible in the digital domain using either conventional digital computers or special purpose DSPs. Signal processing in the digital domain is also extremely useful in such areas as biomedical applications, providing the needed accuracy for tasks such as ultrasound imaging [1]

A/D conversion techniques may roughly be divided into two groups:

- Nyquist-rate (or conventional) converters,
- Oversampling converters.

Oversampling methods are widely used in A/D and D/A conversion since they avoid many difficulties encountered with conventional methods. Some attributes of conventional converters such as the use of analog filters, the need for high precision analog circuits, and vulnerability to noise and interference make their circuits difficult to implement but, their usage of relatively low sampling rates is still an advantage [2].

Among various oversampling A/D conversion techniques available in the literature, sigma-delta (SD) conversion technique is becoming more and more popular. SD converters operate with redundant temporal data, obtained by using oversampling with low-resolution quantizers, and apply signal processing techniques to combine these temporal data which increases the effective resolution. This temporal parallelism is the basis of robustness of the SD converters, which have widened their range from instrumentation to communication as shown in Figure 1.1. The thermal noise limit position

in Figure 1.1 corresponds to the state-of-the-art CMOS data converters reported at the *IEEE International Solid-State Circuits Conference* (ISSCC'96) [3] and in the *IEEE Journal of Solid-State Circuits* [4]. The use of SD based A/D converters for data con-

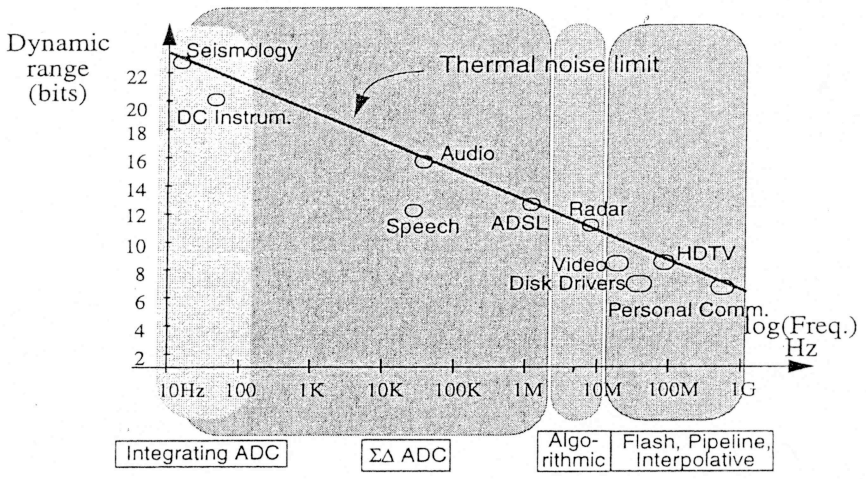

Figure 1.1. Application range of several modulator architectures in the resolution-speed plane

version is very attractive, since it uses basic blocks and requires no sample and hold. The sampling rate employed is much faster than the highest frequency in the message signal itself. The high accuracy conversion is achieved as a result of the modulator operating as a self-adaptive, fast limit cycling system [1, 5]. Besides the advantages of this technique, there are some difficulties in designing SD A/D converters. One of the major difficulties is the determination of the appropriate SD structure, which provides the required performance. Since SD A/D converters contain large number of connections between building blocks (quantizer(s), integrator(s), DAC) there exist more than one structure satisfying the desired performance specifications for a required application and generally the design procedure gets extremely complicated. In order to decrease the complexity of the design procedure, automation tools have been developed [4, 6, 7]. Although all of these approaches have been successful in addressing some aspects of the SD converter design problem, none have been able to solve the architecture selection problem completely.

A standard SD modulator system utilizes several signal feedforward and signal feedback paths. For every such signal path there exists an associated coefficient, which

is the path gain. These coefficients are all related to each other and a small change in one of them may cause dramatic changes in the operation, response, and performance of the overall modulator. On the other hand, including or removing any of these paths corresponds to a different modulator topology. Another important issue is that there are many non-idealities in the SD modulator system in real-life, which should also be taken into account in the design process. These non-idealities may include the clock-jitter, kT/C noise, op-amp noise, and integrator non-idealities such as the finite DC gain (leakage), the switched capacitor mismatches, slew-rate limitation of the op-amp, dc offset of the comparator, etc. [8].

Hence, in designing an SD modulator, the challenge is not only the selection of a modulator topology from a large set of possibilities but also the optimization of the topology parameters, the coefficients, in such a way to satisfy the system specifications without ignoring the mentioned non-idealities. The design flow should include three basic steps:

- determination of all possible SD modulator topologies of any order, which is capable of realizing a desired response,
- selection of the optimum topology from these possibilities,
- calculation of the topology coefficients, which satisfy the system specifications and still remain in considerable limits.

As obvious from the above discussion, the optimum topology for a specific application can only be constructed by employing a design automation tool. Some versatile tools were previously developed in the MATLAB environment as a solution to this design automation problem such as [9] and [10].

This thesis proposes a new design automation tool generated in the MATLAB environment. The tool works independently of the modulator order and finds all possible SD modulator topologies satisfying a desired system response with minimum number of signal paths which in turn leads to minimum complexity. Compared to similar work in this area, this tool has several advantages. First of all, it has a stand-alone tool

which is a symbolic analyzer which works in a SPICE-like fashion; that is, it takes a netlist of an SD modulator architecture of any order and any complexity in block level as the input, determines the input-output relation for each block in z-domain and generates an equation for each node of the architecture in terms of symbolic variables. The designer may utilize this analyzer to evaluate various design alternatives at the block level. It is not only independent of the order of the architecture, but also independent of the actual blocks, whereby the user may add new functional blocks to the tool. Another important advantage of the tool proposed is that, it models the integrator non-idealities which constitute most of the non-idealities of the SD modulator architecture. The tool also optimizes the coefficients not for STF or NTF only, but for both of them simultaneously and it provides the whole solution space satisfying a desired response [11, 12].

1.2. Outline of the Thesis

The next chapter deals with the theoretical aspects of the oversampling SD A/D converters. Basic concepts about SD modulation such as quantization, oversampling, etc. are described. Various different SD modulator architectures from single-loop to cascaded stages, single-bit to multi-bit systems are explained.

In Chapter 3, the SD modulator non-idealities are discussed. Their effects on system performance are investigated.

In Chapter 4, a new automatic architecture generator tool designed in the MATLAB environment is introduced. The operation principles of the tool are described in detail.

Chapter 5 focuses on the application of the automatic architecture generator tool. Various examples are given for the application of the tool and the results are discussed.

Finally, Chapter 6 concludes the thesis.

2. OVERSAMPLING SD A/D CONVERTERS

This chapter focuses on oversampling SD A/D converters. It starts with the description of basic concepts, continues with the oversampling theory, quantization and various SD modulator architectures, from single-loop to cascade higher-order architectures. A much more detailed version of the following may be found in [4].

2.1. Basic Concepts

The block diagram of an oversampling SD A/D converter is shown in Figure 2.1. The following SD A/D converter contains:

- **an anti-aliasing filter,** to eliminate the spectral components with a frequency higher than the Nyquist rate. A simple passive first order filter is enough here, thanks to oversampling.
- **a modulator,** to sample and quantize the signal. Also the modulator shapes the power spectral density (PSD) of the inherent quantization error in such a way that most of its power is carried out of the signal band. The output of the modulator is usually coded into a reduced number of bits at the sampling rate [4].
- **a decimator,** to filter all the spectral components out of the signal band - most of which is the quantization error power - and to decimate the data to reduce the sampling frequency down to the Nyquist frequency so that the signal becomes coded in large number of bits at the output.

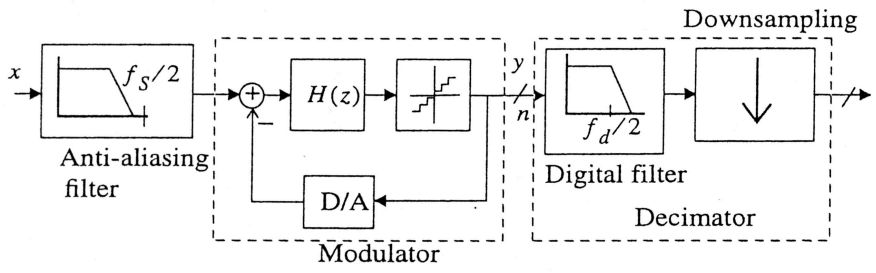

Figure 2.1. Block diagram of a SD A/D converter [4]

The design of the decimator and the anti-aliasing filter are easy compared to the modulator since the former is a well-structured digital block that can be designed by the help of CAD tools and the latter simplified up to a simple RC low-pass filter by the help of oversampling [13] Associated with the modulator there are several error mechanisms including the inherent quantization noise and several circuit non-idealities due to component imperfections which degrade the performance of the system.

2.1.1. Oversampling and Quantization Noise

The quantizer in the SD modulator may be represented mathematically by a non-linear function as

$$y = g_q i + e, \tag{2.1}$$

where y is the output and i is the input of the quantizer; g_q is the slope of the line intersecting the code steps (or the quantizer gain) and e is the quantization error. If the quantizer input is limited to the range $[i_{min}, i_{max}]$, the quantizer error is bounded in the range $[-\Delta/2, \Delta/2]$ where Δ is the difference between consecutive quantizer levels or the stepsize.

If the input to the quantizer is assumed to be varying randomly from sample to sample in the given input range $[i_{min}, i_{max}]$, and the number of levels of the quantizer is large enough, it can be shown [14] that the quantizer error distributes uniformly in the range $[-\Delta/2, \Delta/2]$ and has a constant power spectral density as white noise. That is the reason why the quantization error is usually called *quantization noise*.

If the total quantization noise power is denoted by $\sigma^2(e)$ and if it is uniformly distributed over the frequency range $[-f_s/2, f_s/2]$ where f_s is the sampling frequency, then the PSD of the quantization noise is

$$S_E(f) = \frac{\sigma^2(e)}{f_s} = \frac{\Delta^2}{12 f_s}. \tag{2.2}$$

The whole noise power sits in the signal band if the sampling frequency f_s is the same as the Nyquist frequency and it equals $\Delta^2/12$. But if the sampling frequency is different than the Nyquist frequency, the in-band quantization noise power is given by

$$P_Q = \frac{\Delta^2 f_d}{12 f_s} = \frac{\Delta^2}{12M}, \tag{2.3}$$

where f_d is the Nyquist frequency and f_d/f_s, denoted by M, is the OSR. The in-band noise power is inversely proportional to the OSR, and as suggested by Equation (2.3), there occurs a $3dB/octave$ reduction in the in-band quantization noise power for every increment in the OSR . The above issues are discussed in more detail in [4].

2.1.2. SD Modulator

A basic SD modulator scheme is shown in Figure 2.2. For the quantizer, the signal $e(t)$ is assumed to be uniformly distributed in the range $[-\Delta/2, \Delta/2]$ and its PSD is given by Equation (2.2). The validity of these assumptions are proven in [15] for time-variant modulator inputs. The modulator in Figure 2.2 is a 2-input 1-output

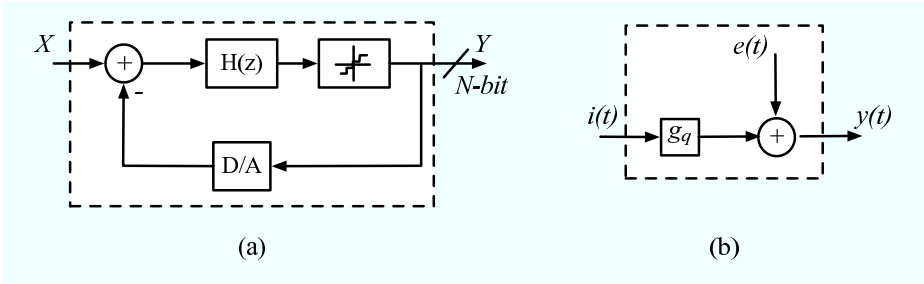

Figure 2.2. (a) Basic structure of the SD modulator. (b) Quantizer model. [4]

system that can be represented in z-domain by

$$Y(z) = STF(z)X(z) + NTF(z)E(z), \tag{2.4}$$

where $Y(z)$ is the z-transform of the output, $X(z)$, $E(z)$ are the z-transforms of the input and quantization noise signals; and $STF(z)$, $NTF(z)$ are the transfer functions

of these signals respectively. $STF(z)$ and $STF(z)$ are mathematically defined as

$$STF(z) = \left.\frac{Y(z)}{X(z)}\right|_{E(z)=0} \quad \text{and} \quad NTF(z) = \left.\frac{Y(z)}{E(z)}\right|_{X(z)=0}. \tag{2.5}$$

The transfer functions $STF(z)$ and $NTF(z)$ depend on the modulator architecture but generally, in order to have a working modulator the following conditions should be imposed [4]:

$$\begin{array}{rcl} |STF(z)| & = & cte \\ NTF(z) & \rightarrow & 0 \end{array} \quad \text{for} \quad z \rightarrow 1 \tag{2.6}$$

The governing equation for the SD modulator scheme shown in Figure 2.2 can be written as

$$Y(z) = \frac{H(z)}{1+H(z)}X(z) + \frac{1}{1+H(z)}E(z). \tag{2.7}$$

This equation, combined with the conditions given in Equation (2.6) results in a discrete time filter $H(z)$ which is expressed as

$$H(z) = \frac{z^{-1}}{1-z^{-1}}. \tag{2.8}$$

Substituting this result in (2.7) leads to an output in the form,

$$Y(z) = z^{-1}X(z) + (1-z^{-1})E(z), \tag{2.9}$$

which means that the output is a delayed version of the input signal combined with the *shaped* quantization noise where the shaping function is the $NTF(z) = (1-z^{-1})$. Since the given NTF is first order, the modulator in Figure 2.2 is said to be *a first order modulator*. Another representation of the same modulator is given in Figure 2.3 where both the signal path and the feedback path has the associated coefficients or gain factors g_1 and $g_1{}'$ respectively.

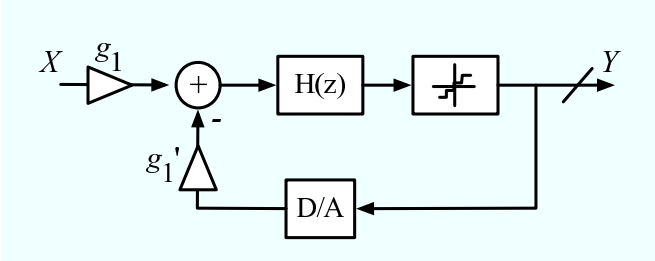

Figure 2.3. A first order SD modulator

2.1.3. Commonly Used Figures of Merit

Commonly used figures of merit for the performance estimation of oversampling converters may be summarized as follows:

2.1.3.1. Signal-to-Noise Ratio (SNR). If quantization noise is assumed to be the only noise source of the system, the SNR, which is only dependent on the input signal amplitude in this case, is given as

$$SNR(dB) = 10\log_{10}\left(\frac{A^2/2}{P_Q}\right), \tag{2.10}$$

where A stands for the amplitude of the input sinusoid, $A^2/2$ is the output power at the frequency of the input sinusoid and P_Q is the in-band noise power given by Equation (2.3). According to Equation (2.10), SNR increases monotonously with the input amplitude but, beyond a certain input level, the input at the quantizer exceeds the range $[i_{min}, i_{max}]$ given in Section 2.1.1 and at a definite input level, a sharp decrease occurs in the SNR vs. *input amplitude* curve.

It has to be noted that the assumption of quantization noise as the only noise source is generally very optimistic and there are other noise sources due to circuit level non-idealities in reality. For this reason, it is more convenient to use the signal-to-(noise distortion) ratio, $TSNR$ [4].

2.1.3.2. Dynamic Range (DR). The dynamic range is the ratio of the output power for an input with the full-scale range amplitude and that for an input at the same

frequency with the previous one but with an amplitude which results in a 0 dB SNR. If the quantizer is a single-bit one, the full-scale range amplitude corresponds to $\Delta/2$. The output power for an input resulting in a 0 dB SNR is obviously P_Q from Equation (2.10). So for that case, the DR can be given as [4]

$$DR(dB) = 10\log_{10}\left[\frac{(\Delta/2)^2}{2P_Q}\right]. \tag{2.11}$$

2.1.3.3. Effective Resolution (B). The effective resolution of an SD modulator may be defined in terms of its DR as [4]

$$B(bit) = \frac{DR(dB) - 1.76}{6.02}, \tag{2.12}$$

which states that a 3 dB increase in DR results in a 0.5 bit increase in effective resolution.

2.2. SD Modulator Architectures

Up to now, several different SD modulator architectures have been proposed. The main objective in designing new SD architectures is to provide the maximum possible in-band quantization noise reduction. This reduction makes it possible to obtain a given resolution with a lower oversampling ratio, and in turn results in a lower power-to-speed ratio for the SD modulator. There are two basic ways of in-band noise power reduction:

i. Increasing the order of the NTF(z) by increasing that of the discrete time filter H(z), in order to achieve a better cancellation of the quantization noise.
ii. Increasing the number of bits of the internal quantizer, since increased quantizer resolution results in an increase in the effective resolution of the modulator. Because the stepsize Δ and in turn the PSD of quantization error is reduced by this way.

2.2.1. Second Order SD Modulator

If the number of integrators used in the SD modulator architecture is two, then the modulator is called a *'second order SD modulator'* which is shown in Figure 2.4. The order of NTF in this case is two also. The stability of this loop is guaranteed if

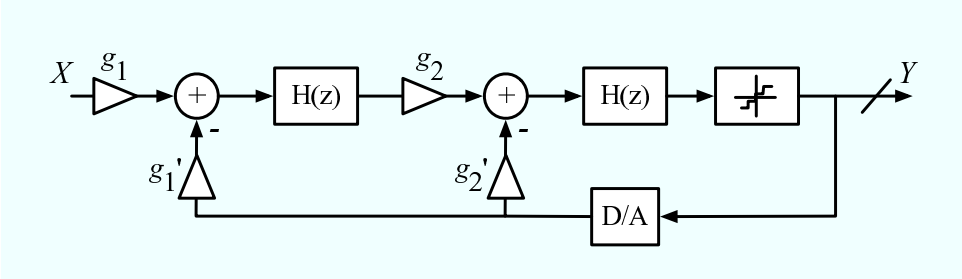

Figure 2.4. A second order SD modulator

$g_2{}' = 2g_1g_2$ [16]. Also imposing $g_1{}' = g_1$ results in the following output:

$$Y(z) = z^{-2}X(z) + (1 - z^{-1})^2 E(z) \tag{2.13}$$

Compared to the output of a first order system given in Equation (2.9), the $NTF(z)$ is now a second order filter which provides a better noise cancellation and decreases the in-band noise power. Also the extra feedback path has has a very important function in decreasing the correlation between the input and quantization error signals.

The PSD of quantization noise for the second order case is

$$S_Q(f) = S_E(f)\left[16\sin^4\left(\pi\frac{f}{f_s}\right)\right], \tag{2.14}$$

where $S_E(f)$ is as given in Equation (2.2). Furthermore, the in-band-noise power for the second order SD modulator is

$$P_Q \cong \frac{\Delta^2}{12}\frac{\pi^4}{5M^5}, \tag{2.15}$$

where M is the OSR. (For a detailed analysis of the PSD and in-band noise power for this case, please refer to [4].) As can be deduced from Equation (2.14) and Equa-

tion (2.15), the DR and the effective resolution is higher for the second order modulator compared with the first order one.

The difference between $NTF(z)$ of the first and the second order system is shown in Figure 2.5. For the second order system, the PSD is reduced in the low frequency regions and it is increased in the high frequency region which means the noise is carried out of the signal band more efficiently and the in-band noise power is decreased.

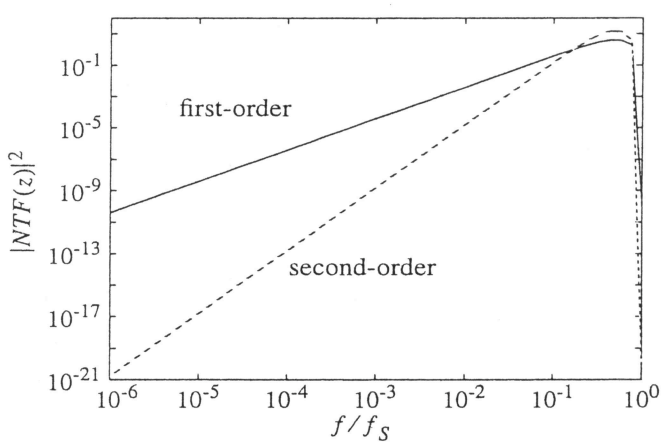

Figure 2.5. Noise transfer functions of a first and second order SD modulator as a function of the frequency [4]

Although the second order modulator has such advantages over the first order one, it is more prone to instability. An increase in the loop gain beyond a definite value, or an extra delay in the loop may cause the modulator become instable.

2.2.2. Single-loop High-order Modulators

Higher-order SD modulator architectures can easily be obtained by increasing the number of integrators before the quantizer. For the generalized case of L integrators, which means a modulator of order L, the generalized z-domain output can be expressed as

$$Y(z) = z^{-L}X(z) + (1 - z^{-1})^{L}E(z), \tag{2.16}$$

and the generalized expressions for the quantization noise PSD and the in-band noise power become

$$S_Q(f) = S_E(f)\left[2^{2L}\sin^{2L}\left(\pi\frac{f}{f_s}\right)\right], \tag{2.17}$$

and

$$P_Q \cong \frac{\Delta^2}{12}\frac{\pi^{2L}}{(2L+1)M^{2L+1}}. \tag{2.18}$$

2.2.2.1. Stability Considerations. As the order L of an SD modulator increases, its tendency to instability increases also. In order an SD modulator to be called *'stable'* the outputs of the integrators of the system should be bounded in a definite range for bounded inputs independent of the initial conditions. It can be mathematically proven that first order and second order systems are stable for every input in the ranges $(-\Delta/2, \Delta/2)$ and $(-0.9\Delta/2, 0.9\Delta/2)$ respectively [16].

But unfortunately, for systems of order $L > 2$, it is impossible to derive a stability condition mathematically. It is only figured out that the stability depends on the architecture coefficients (path gains) and the initial conditions. So in order to obtain stable higher order modulators, the coefficients should be selected properly and the out-of-band quantization noise should be reduced to such a level to ensure stability. Also the integrator outputs may be bounded in a definite range by using limiters. Another way is globally resetting the integrators in case of an unstable operation which can be made possible by usage of comparators and local feedback loops [17].

2.2.3. High-order SD Modulator Architectures

As explained in Section 2.2.2.1, high-order modulators suffer from the problem of instability. Various ways have been discovered to overcome the instability problem associated with these high-order modulators.

2.2.3.1. Single-loop Modulators. Lee and Sodini have proposed a solution [18] to the instability problem of high-order modulators which makes it possible to generate $NTF(z)$ with multiple poles and zeros. The architecture is shown in Figure 2.6. The $STF(z)$ and $NTF(z)$ of this architecture is as follows:

$$STF(z) = \frac{\sum_{i=0}^{L} A_i (z-1)^{L-i}}{z\left[(z-1)^L - \sum_{i=1}^{L} B_i (z-1)^{L-i}\right] + \sum_{i=0}^{L} A_i (z-1)^{L-i}} \tag{2.19}$$

$$NTF(z) = \frac{(z-1)^L - \sum_{i=1}^{L} B_i (z-1)^{L-i}}{z\left[(z-1)^L - \sum_{i=1}^{L} B_i (z-1)^{L-i}\right] + \sum_{i=0}^{L} A_i (z-1)^{L-i}} \tag{2.20}$$

By adjusting the coefficients A_i and B_i, the location of zeros and poles may be defined and the in-band quantization noise power may be minimized.

2.2.3.2. Cascade Modulators. Higher order SD modulator architectures may be obtained by cascading zeroth, first and second order modulators whose stability is guaranteed by proper design. The resulting architecture is the *"cascade"* architecture which

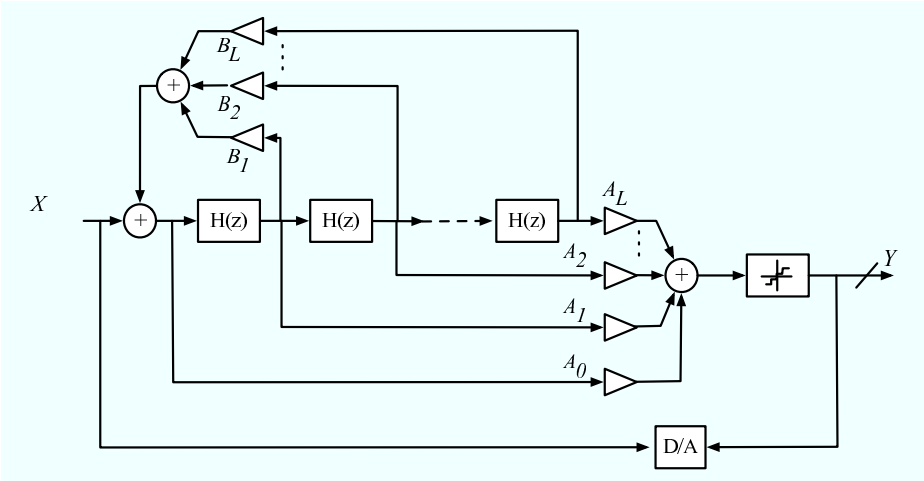

Figure 2.6. L-th order Lee-Sodini SD Modulator

is also known as *multi-stage* or *MASH* architecture.

In the cascade architecture, the output of one stage is the input of the other as shown in Figure 2.7 and by this way, the quantization noise is shaped in each section further and further. After that, the resulting noise is cancelled in a *digital cancellation block*. So the order of the $NTF(z)$ is equal to the total number of integrators in the architecture. For a general case of a cascaded modulator of L-stages (0th, 1st or 2nd

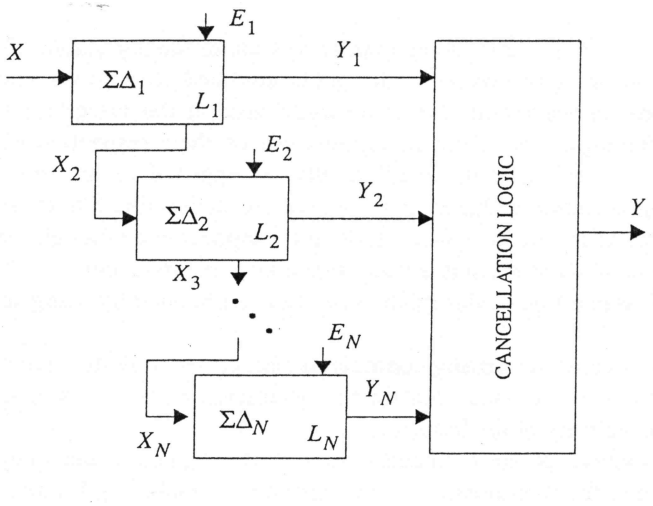

Figure 2.7. Block diagram of a cascade SD modulator [4]

order) the general z-domain output can be expressed as;

$$Y(z) = STF(z)X(z) + NTF_1(z)E_1(z) + NTF_2(z)E_2(z) + \cdots + NTF_L(z)E_L(z) \quad (2.21)$$

where $NTF_L(z)$ and $E_L(z)$ denotes the NTF and quantization noise of the Lth stage respectively.

2.2.4. Multi-bit Quantization SD Modulators

Increasing the number of bits of the quantizer in the SD loop increases the resolution as noted in the beginning of Section 2.2. Another advantage is that, since for a multi-bit quantizer it is easy to guess the saturation of the integrator, the multi-bit system is less prone to instability. Also obviously, increasing the number of quantization levels results in a better linear approximation at the output of the quantizer to

the signal at its input.

A summary of the advantages and disadvantages of the architectures mentioned in this text is given in Figure 2.8 for comparison purposes.

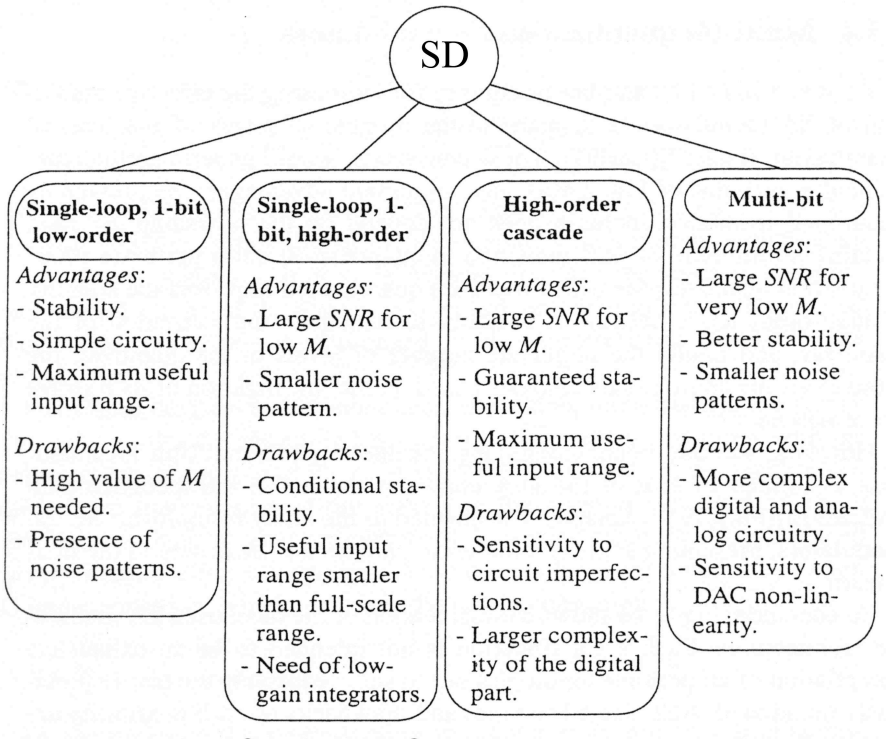

Figure 2.8. Summary of SD modulator architectures [4]

3. SD MODULATOR NON-IDEALITIES

In a standard switched capacitor (SC) SD modulator loop there is an inherent non-linearity due to the non-idealities of the components of the system. This chapter will summarize these component non-idealities that effect the modulator behavior.

3.1. Clock Jitter

The operation of an SC SD modulator depends on complete charge transfers during each of the clock phases. After the analog signal has been sampled, the system turns into a sampled-data system and the variations of the clock periods do not have a direct effect on the system. So it is enough to compute the effect of clock jitter on the sampling process in a SD modulator only. Here it can easily be deduced that, the effect of clock jitter on the operation of an SD modulator is independent of the modulator architecture.

Sampling jitter causes a non-uniform sampling and increases the total noise power in the quantizer output. The error introduced when a sinusoidal signal with amplitude A and frequency f_{in} is sampled at an instant which is in error by an amount δ is given by [8]

$$x(t+\delta) - x(t) = 2\pi f_{in}\delta A\cos(2\pi f_{in}t) = \delta\frac{d}{dt}x(t) \tag{3.1}$$

where $x(t)$ is the analog input signal.

If the sampling time uncertainty δ is assumed to be a *Gaussian process* with the standard deviation of $\Delta\tau$, the resulting jitter error has a uniform PSD over the range $[0, f_s/2]$ with a total power of $\frac{(2\pi f_{in}\Delta\tau A)^2}{2}$.

3.2. Integrator Noise

The thermal noise associated to the sampling switches and the intrinsic noise of the operational amplifier are among the most important noise sources affecting the operation of an SC SD modulator. There are three noise power sources adding up to the total noise power of the SD modulator system. They are,

- quantization noise
- switch thermal noise
- op-amp noise.

The noise performance of the system is mainly determined by the switches and the op-amp of the first stage, since the low-frequency gain of the first stage is large compared to the next stages.

3.2.1. Switch Thermal Noise

Thermal noise is caused by random fluctuation of carriers due to thermal energy and is present even at equilibrium. Thermal noise has a white spectrum and wide band limited only by the time constant of the switched capacitors or the bandwidths of op-amps. Therefore it must be taken into account for both the switches and the op-amps.

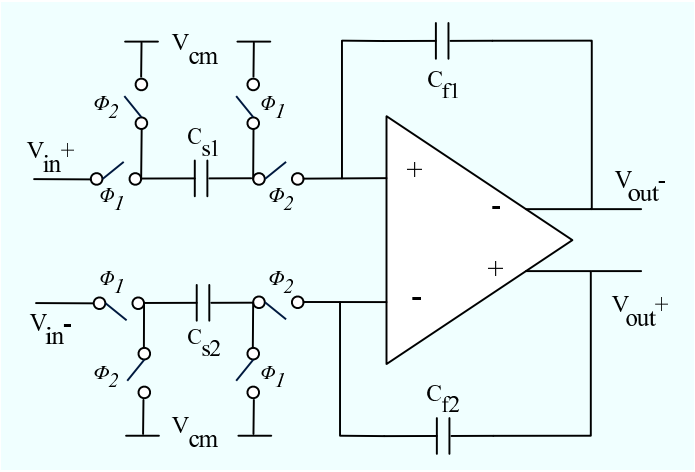

Figure 3.1. A differential SC integrator

A differential SC integrator is shown in Figure 3.1. The sampling capacitors C_{s1} and C_{s2} are in series with switches with the *on-resistance* R_{on}. These switches periodically open and close sampling a noise voltage on the capacitors. Taking $C_{s1} = C_{s2} = C_s$, the total noise power can be evaluated by [13]

$$e_T^2 = \int_0^\infty \frac{4kTR_{on}}{1+(2\pi f R_{on}C_s)^2} df = \frac{kT}{C_s}, \tag{3.2}$$

where k is the Boltzman constant, T is the absolute temperature and the resistance is modeled with a series noise source with power $4kTR_{on}\Delta f$ and e_T is the switch thermal noise voltage. This voltage is superimposed to the input voltage $x(t)$ as follows:

$$y(t) = [x(t) + e_T(t)]b = \left[x(t) + \sqrt[2]{\frac{kT}{bC_f}}n(t)\right] b, \tag{3.3}$$

where $n(t)$ denotes a Gaussian random process with unity standard deviation, b is the integrator gain and C_f is the feedback capacitance assuming $C_{f1} = C_{f2} = C_f$. Since the integrator is differential, both of the sampling capacitors will contribute to the switch thermal noise. Also typically, there are also the switched feedback capacitors at both inputs of the integrators carrying the feedback signals coming from the DAC. These will also contribute to the thermal noise power.

3.2.2. Op-amp Noise

The flicker noise $(1/f)$, the wide-band thermal noise and the DC offset contribute to the total RMS noise voltage referred to the input of the op-amp. If this total RMS noise voltage is denoted by V_n, then the total noise power due to this source, which is V_n^2, can be calculated by simulation in clock phase Φ_2, adding all the noise contributions referred to the op-amp input and integrating the result over the whole frequency range [8].

3.3. Integrator Non-idealities

The transfer function of an ideal integrator is given in Equation (2.8). However, real analog implementations of the integrator deviate from this ideal response due to several non-ideal effects. The integrator non-idealities constitute an integral part of the SC SD modulator performance degradation since they result in incomplete charge transfer.

3.3.1. Finite DC Gain

The DC gain of an integrator is ideally infinite. However, this DC gain is finite in practice due to circuit implementation constraints. The physical meaning of the gain limitation is charge leakage. This leakage means that not all of the previous output but only a fraction α of it is fed to the input of the integrator in the next cycle. So the non-ideal integrator transfer function becomes

$$H(z)_{non-ideal} = \frac{z^{-1}}{1 - \alpha z^{-1}}. \tag{3.4}$$

This limited gain in low frequencies increases the in-band noise [8].

3.3.2. Bandwidth and Slew-rate

The finite BW and SR limitations are related to each other and can be interpreted as a non-linear gain. The output voltage of the integrator given in Figure 3.1, at the nth integration period is

$$v_o(t) = v_o(nT - T) + \alpha V_s \left(1 - e^{-t/\tau}\right) \ , \quad nT - \frac{T}{2} < t < nT \tag{3.5}$$

where $V_s = V_{in}(nT - \frac{T}{2})$, α is the integrator leakage, $\tau = 1/(2\pi GBW)$ is the time constant of the integrator and GBW is the gain-bandwidth product.The slope of this

curve reaches its maximum at $t = 0$ and that slope is

$$\frac{d}{dt}v_o(t)\Big|_{max} = \alpha\frac{V_s}{\tau}. \tag{3.6}$$

There are two separate cases then, if the slope value calculated by Equation (3.6) is lower than the op-amp SR value there is no slewing else the op-amp is said to be slewing.

3.3.3. Saturation

The dynamic of signals is a major concern for SD modulators. The saturation levels of the op-amp, directly changes the dynamic of signals at the intermediate nodes in the SD modulator architecture. Because, the saturation levels of the op-amp limit the maximum achievable voltage level at the output of the integrators and this limitation may cause dramatic errors in the A/D conversion process, especially for systems with multi-bit quantizers. Hence, the saturation of the op-amp has a very important effect on the system performance and should also be taken into account in designing SD modulators.

The next chapter will introduce a new design automation tool, an automatic architecture generator, for SD modulators which takes into account most of the modulator non-idealities and generates all possible modulator architectures satisfying a desired frequency response.

4. THE AUTOMATIC ARCHITECTURE GENERATOR

The automatic architecture generator is a new design automation tool for SD modulators generated in the MATLAB environment. This tool works independently of the modulator order and finds all possible SD modulator topologies satisfying a desired system response with the minimum number of signal paths which in turn leads to minimum complexity. The operation of the tool may be divided into two basic parts: (1) Generation of the symbolic transfer functions (STF and NTF) for any given topology of any order and with any complexity, (2) Generation of all possible topologies with minimum number of signal paths that is, minimum complexity. These two parts will be explained in detail in the following sections.

4.1. Generation of the Parametric (Symbolic) Transfer Functions

This part of the tool is a symbolic analyzer for SD modulators. It works in a SPICE-like fashion; that is, it takes a netlist of an SD modulator architecture of any order and any complexity in block level as the input, determines the input-output relation for each block in z-domain and generates an equation for each node of the architecture in terms of symbolic variables. Then the user is able to find the transfer function from any node to any node by just writing the ratio of one node to one another. Several blocks are defined in this tool which can be summarized as follows:

- *INTEGRATOR:* An integrator block with the z-domain transfer function of $\frac{1}{1-z^{-1}}$. It has no delay.
- *INTEGRATORD:* An integrator block with the z-domain transfer function of $\frac{z^{-1}}{1-z^{-1}}$. It has one unit delay, for which the capital letter D stands.
- *NONIDEAL_INTEGRATOR:* An integrator block with the z-domain transfer function of $\frac{b}{1-cz^{-1}}$, where the parameters b and c stand for the integrator non-idealities that will be explained in Section 4.3. It has no delay.
- *NONIDEAL_INTEGRATORD:* An integrator block with the z-domain transfer function of $\frac{bz^{-1}}{1-cz^{-1}}$ where the parameters b and c again stand for the integrator

non-idealities. It has one unit delay, for which the capital letter D stands.

- *ADDER:* This is a dynamic block, that is, it works independently of the number of inputs to it. It can add or subtract any number of signals simultaneously if the nodes are defined properly as positive (denoting an addition) or negative (denoting a subtraction) in the netlist.
- *GAIN:* This is a one-input one-output block used to model the SD modulator architecture coefficients or path gains. Its output is equal to its input multiplied by a constant term.

Apart from these there are also some reserved terms used in defining the netlist of the architecture. These are: (1)IN, and (2)$NOISE$ defining the input signal, and the quantization noise signal nodes of the architecture. At this point, the 1-bit quantizer in the SD modulator architecture is assumed to be an ideal unity gain element, which just adds a noise signal into the system. For this reason it is defined as an adder in the netlist and no extra element has been defined for the quantizer. The D/A converter is also considered as ideal and is just a unity gain element. A generic second order SD

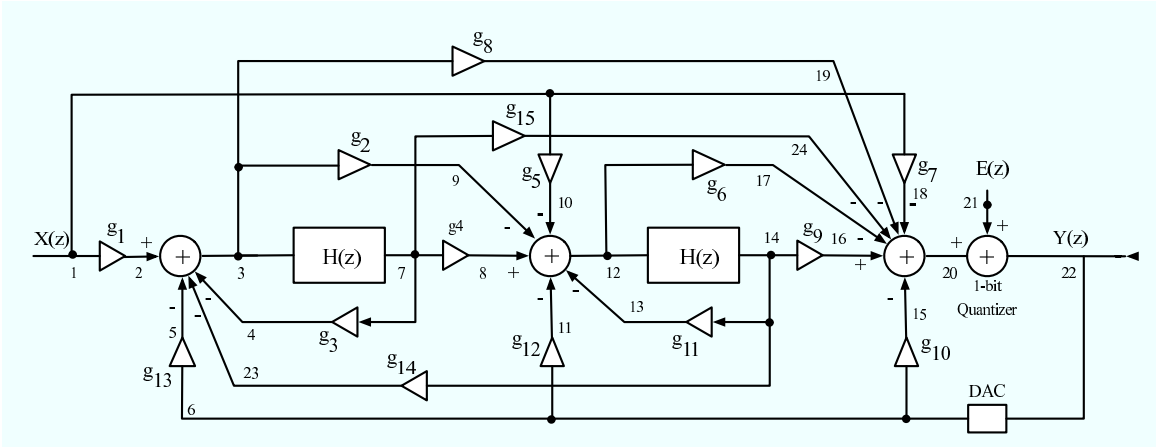

Figure 4.1. The generic standard second order SD modulator architecture comprising all possible feedback and feedforward paths

modulator architecture is shown in Figure 4.1. It comprises all the possible feedback and feedforward paths from g_1 up to g_{15}. Assuming the integrators to be ideal, the architecture given in Figure 4.1 can be defined in block level by the netlist shown in Figure 4.2. The tool gets this netlist as the input and generates symbolic expressions for each node of the architecture of Figure 4.1. For example after processing line 17

```
1   IN 1                    13  GAIN12 6 11
2   GAIN1 1 2               14  GAIN13 6 5
3   GAIN2 3 9               15  GAIN14 14 23
4   GAIN3 7 4               16  GAIN15 7 24
5   GAIN4 7 8               17  ADDER 2 -5 -4 -23 3
6   GAIN5 1 10              18  ADDER 8 -9 -10 -11 -13 12
7   GAIN6 12 17             19  ADDER 16 -17 -18 -19 -15 -24 20
8   GAIN7 1 18              20  ADDER 20 21 22
9   GAIN8 3 19              21  DAC 22 6
10  GAIN9 14 16             22  INTEGRATORD 3 7
11  GAIN10 6 15             23  INTEGRATORD 12 14
12  GAIN11 14 13            24  NOISE 21
```

Figure 4.2. The block level netlist for the architecture shown in Figure 4.1

in the netlist, the tool generates the equation

$$x_3 = x_2 - x_5 - x_4 - x_{23}, \tag{4.1}$$

and line 22 results in the equation

$$x_7 = \frac{z^{-1}}{1 - z^{-1}} x_3 \tag{4.2}$$

etc., each x_i denoting the symbolic expression associated with node i of the architecture. Finally, after processing all the lines of the netlist, the symbolic equations can be expressed in terms of the path gains g_1 to g_{15}.

After calculations of all symbolic expressions, the STF and the NTF in terms of path gains can be generated by inputting manually the proper fractional expressions, which are

$$STF = \left.\frac{x_{22}}{x_1}\right|_{NOISE=0} \quad and \quad NTF = \left.\frac{x_{22}}{x_{21}}\right|_{IN=0} \tag{4.3}$$

for the specific architecture of Figure 4.1, where x_{22} is the output node, x_1 is the signal input node and x_{21} is the noise input node. The resulting STF and NTF in terms of

path gains for the given architecture are as follows:

$$STF(z) = \frac{n_{2_{STF}} z^2 + n_{1_{STF}} z + n_{0_{STF}}}{d_{2_{STF}} z^2 + d_{1_{STF}} z + d_{0_{STF}}} \tag{4.4}$$

$$NTF(z) = \frac{n_{2_{NTF}} z^2 + n_{1_{NTF}} z + n_{0_{NTF}}}{d_{2_{NTF}} z^2 + d_{1_{NTF}} z + d_{0_{NTF}}} \tag{4.5}$$

where

$$\begin{aligned}
n_{2_{STF}} &= -g_7 + g_6\, g_5 + g_1\, g_6\, g_2 - g_1\, g_8 \\
n_{1_{STF}} &= -g_1\, g_9\, g_2 + g_{14}\, g_2\, g_7 - g_1\, g_8\, g_{11} - g_1\, g_{15} \\
&\quad -g_{14}\, g_5\, g_8 - g_7\, g_1\, 1 + 2\, g_1\, g_8 - g_3\, g_7 - g_1\, g_6\, g_4 \\
&\quad +g_3\, g_6\, g_5 - g_9\, g_5 + 2\, g_7 - 2\, g_1\, g_6\, g_2 - 2\, g_6\, g_5 \\
n_{0_{STF}} &= -g_7 + g_3\, g_7 + g_1\, g_{15} - g_1\, g_8 + g_9\, g_5 + g_7\, g_{11} \\
&\quad -g_3\, g_6\, g_5 + g_6\, g_5 + g_1\, g_6\, g_2 + g_1\, g_6\, g_4 + g_1\, g_9\, g_4 \\
&\quad +g_1\, g_8\, g_{11} + g_1\, g_9\, g_2 + g_{14}\, g_5\, g_8 - g_{14}\, g_5\, g_{15} \\
&\quad -g_{14}\, g_4\, g_7 - g_1\, g_{15}\, g_{11} - g_3\, g_7\, g_{11} - g_3\, g_9\, g_5 - g_{14}\, g_2\, g_7 \\
n_{2_{NTF}} &= 1 \\
n_{1_{NTF}} &= -2 + g_3 + g_{11} - g_{14}\, g_2 \\
n_{0_{NTF}} &= 1 - g_{11} + g_{14}\, g_2 - g_3 + g_{14}\, g_4 + g_{11}\, g_3 \\
d_{2_{STF}} = d_{2_{NTF}} &= -g_8\, g_{13} + g_{10} + g_6\, g_{13}\, g_2 - g_6\, g_{12} + 1 \\
d_{1_{STF}} = d_{1_{NTF}} &= -2 + g_3 + g_{11} - g_9\, g_{13}\, g_2 + g_{10}\, g_3 + g_9\, g_{12} \\
&\quad +g_{10}\, g_{11} - g_{14}\, g_2 - g_8\, g_{11}\, g_{13} - g_{14}\, g_{10}\, g_2 + 2\, g_8\, g_{13} \\
&\quad -2\, g_{10} - g_{15}\, g_{13} + 2\, g_6\, g_{12} - g_6\, g_3\, g_{12} + g_{14}\, g_8\, g_{12} \\
&\quad -g_6\, g_{13}\, g_4 - 2\, g_6\, g_{13}\, g_2 \\
d_{0_{STF}} = d_{0_{NTF}} &= 1 - g_3 + g_{10} - g_{11} + g_6\, g_{13}\, g_2 + g_{14}\, g_2 + g_8\, g_{11}\, g_{13} \\
&\quad -g_{15}\, g_{11}\, g_{13} + g_{14}\, g_{15}\, g_{12} + g_{14}\, g_{10}\, g_2 + g_{14}\, g_4 \\
&\quad +g_9\, g_3\, g_{12} + g_{10}\, g_{11}\, g_3 + g_{11}\, g_3 + g_9\, g_{13}\, g_4 + g_9\, g_{13}\, g_2 \\
&\quad +g_6\, g_3\, g_{12} - g_{14}\, g_8\, g_{12} + g_6\, g_{13}\, g_4 + g_{14}\, g_{10}\, g_4 + g_{15}\, g_{13} \\
&\quad -g_9\, g_{12} - g_6\, g_{12} - g_{10}\, g_{11} - g_{10}\, g_3 - g_8\, g_{13}
\end{aligned} \tag{4.6}$$

Before closing this section, it has to be mentioned that this part of the tool can also be used as a stand-alone tool which is a symbolic analyzer. The designer may utilize this analyzer to evaluate various design alternatives at the block level. It is not only

independent of the order of the architecture, but also independent of the actual blocks, whereby the user may add new functional blocks to the tool. This symbolic analyzer is a really powerful tool and nothing similar to it has been proposed up to now.

4.2. Generation of All Possible SD Modulator Architectures

This part of the tool is invoked after the symbolic STF and NTF are generated as described above. The user inputs two numeric transfer functions one describing the desired STF, and the other describing the desired NTF. These numeric transfer functions are the responses to be realized by the symbolic transfer functions such as those given in Equation (4.4) and Equation (4.5) for the generic architecture of Figure. 4.1. In order to achieve this, the coefficients of the symbolic STF and NTF given in Equation (4.6) are matched with those of the numeric STF and NTF respectively. Here it should be reminded that, the poles of both STF and NTF should be at the same locations in the frequency domain; that is, the denominators of both STF and NTF are identical to each other as can be seen from Equation (4.6).

If the numeric transfer functions to be realized are in the form;

$$STF(z) = \frac{p_2 z^2 + p_1 z + p_0}{q_2 z^2 + q_1 z + q_0} \tag{4.7}$$

$$NTF(z) = \frac{r_2 z^2 + r_1 z + r_0}{q_2 z^2 + q_1 z + q_0} \tag{4.8}$$

where p_i, q_i, r_i are all real numbers, then the tool generates the following set of equations in terms of fifteen path gain parameters from g_1 to g_{15}:

$$\begin{aligned} n_{2_{STF}} &= p_2, \quad n_{2_{NTF}} = r_2, \quad d_{2_{STF}} = d_{2_{NTF}} = q_2 \\ n_{1_{STF}} &= p_1, \quad n_{1_{NTF}} = r_1, \quad d_{1_{STF}} = d_{1_{NTF}} = q_1 \\ n_{0_{STF}} &= p_0, \quad n_{0_{NTF}} = r_0, \quad d_{0_{STF}} = d_{0_{NTF}} = q_0 \end{aligned} \tag{4.9}$$

Then, this set of equations, which consist of nine equations for this given specific example, will be solved simultaneously to generate a set of solutions. Each element of this set corresponds to a set of path gains from g_1 to g_{15} which in turn corresponds to

a different SD modulator architecture.

Here it is obvious that the number of equations, which is nine for this specific case, is less than the number of variables, which is fifteen. On one hand, this excess of variables increases the complexity of the problem but, on the other hand it introduces a great deal of freedom to the designer in generating many different topologies all realizing the same desired response.

In solving this system of equations, the tool uses some criteria such as minimization of the number of signal paths of the architecture in order to obtain minimum possible complexity, avoiding of single closed loops without a delay and forcing all the coefficients to be real numbers. Taking these criteria into account, different combinations of coefficients are assigned as zeros, which means that those paths are removed from the generic topology. At the beginning, the number of coefficients to be assigned as zeros is equal to the amount by which the number of coefficients exceeds the number of equations. The more is the number of different combinations, the higher is the degree of freedom and, the more is the number of different topologies. Then the resulting set of equations are solved by invoking MATLAB's symbolic toolbox, which is constructed on the MAPLE kernel.

After the solution set is obtained, the tool runs some checks on the solution set. Firstly it checks whether there exists any duplicated solutions or not. If yes, it eliminates them. After that it checks the solutions to ensure that all of them are real numbers, if not, it eliminates the imaginary ones. Finally, the tool returns a set of parametric solutions for the coefficients in which the values of coefficients are defined in terms of a few other coefficients. The rest is just as straightforward as assigning some values to the parameters in the final solution set. These values should be selected carefully for ease of implementation such as ± 1, ± 0.5, etc.

Here it has to be noted that this tool has some important advantages with respect to other SD modulator design automation tools such as the commonly known SD Toolbox DELSIG [19]. DELSIG only finds solutions for 4 basic architectures,

which are cascade-of-resonators-feedback form, cascade-of-resonators-feedforward form, cascade-of integrators-feedback form, and cascade-of-integrators-feedforward form. On the other hand, this tool has no such limitation. It can find different topologies for every modulator architecture defined with a netlist. Occurrence of both several feed-forward and several feedback paths is allowed in this tool. Also as mentioned earlier, the STF and NTF are strongly dependent on each other since the poles of both transfer functions are the same. Hence, these two transfer functions should be optimized simultaneously; since it is not only STF or NTF but both of them that determines the system behavior. This tool is capable of doing this, which is one of its major advantages over [10]. A flowchart summarizing the operation of the tool is given in Figure 4.3.

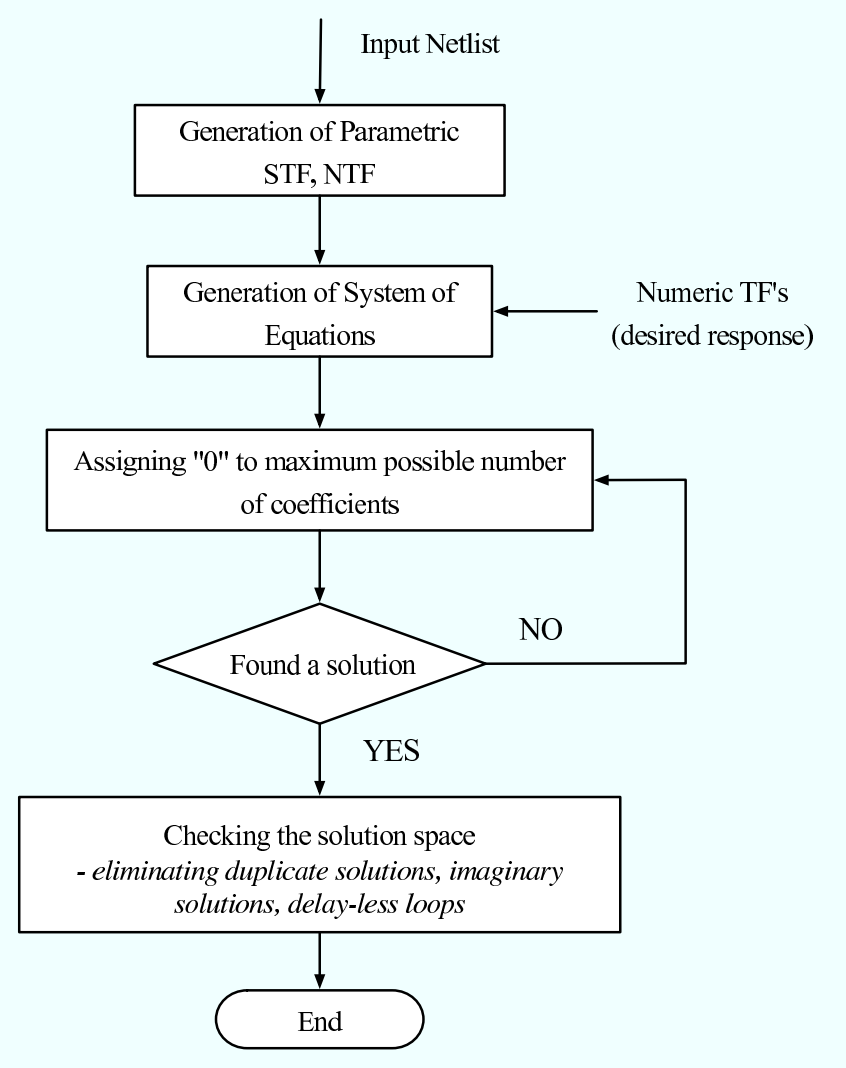

Figure 4.3. The flowchart of the automatic architecture generator tool

4.3. Inclusion of the Non-idealities

As described in Chapter 3, the component non-idealities in the SC implementation of SD modulator architectures have a great impact on the system performance. Some of the non-idealities described in Chapter 3 can be directly mapped to the transfer functions of the system as an extra parameter or an extra noise source and some of them cannot. Among those that can be directly mapped to the transfer function, the integrator non-idealities constitute the most important part and the following analysis will focus on them.

4.3.1. Extraction of Non-ideality Parameters from a SC Integrator

A SC integrator has been given in Figure 3.1. According to the models proposed by Robertini and Guggenbühl in [20], all the integrator non-idealities - except the SR and saturation limitations which cannot be mapped directly to the transfer function - can be modeled by the following non-ideal integrator transfer function with reference to Figure 3.1 and with the assumptions $C_{f1} = C_{f2} = C_f$, $C_{s1} = C_{s2} = C_s$:

$$H(z)_{non-ideal} = \frac{C_s}{C_f} \frac{(1-\delta_1)(1-\delta_2)z^{-1}}{(1-(1-\delta_3)z^{-1})(1-\delta_1\delta_2)z^{-1}} \tag{4.10}$$

where,

C_s : *sampling capacitor*
C_f : *feedback capacitor*
δ_1 : *charging error*
δ_2 : *charge transfer error*
δ_3 : *finite gain error*

The exact expression for the non-ideal integrator is rather complex as can be seen; but for the sake of simplicity, the higher order terms may be ignored and the transfer

function in Equation (4.10) can be estimated with the following one:

$$H(z)_{non-ideal} = \frac{bz^{-1}}{1 - cz^{-1}} \tag{4.11}$$

where the parameters b and c are used to model most of the integrator non-idealities mentioned before.

These integrator non-idealities are included in the tool as described in Section 4.1 via the NONIDEAL_INTEGRATOR and NONIDEAL_INTEGRATORD blocks. The user may include these non-ideal effects in the architecture by just replacing the lines 22 and 23 in the netlist shown in Figure 4.2 properly.

The numeric values of the parameters b and c will obviously be needed for the tool to be run for any non-ideal architecture. These values may be extracted easily from an implemented SC integrator circuit by simulation. For a non-ideal integrator for which the relation between the output $Y_{int}(z)$ and the input $X_{int}(z)$ is given by Equation (4.11), the following analysis can be carried on:

$$Y_{int}(z) = \frac{bz^{-1}}{1 - cz^{-1}} X_{int}(z), \tag{4.12}$$

$$Y_{int}(z) - cz^{-1}Y_{int}(z) = bz^{-1}X(z) \tag{4.13}$$

Taking the inverse z-tansform of the last expression and rearranging, we end up with

$$y[n] = cy[n-1] + bx[n-1], \tag{4.14}$$

which means the current output sample of the integrator is the sum of c times its previous output sample and b times its previous input sample. So by running a simulation for the SC integrator, and taking sufficient number of samples, the values of parameters b and c can easily be estimated. A folded-cascode op-amp has been designed to be used for implementation purposes of SD modulators and the parameters b and c for an SC integrator implemented by using this op-amp have been extracted by applying

the technique described above. These issues will be discussed in the next section.

4.3.2. Design of a High-Performance Operational Amplifier

In order to test the performance of the tool proposed here in a standard second order system including the discussed integrator non-idealities, several behavioral simulations have been carried out. However, to be more realistic in choice of parameters, a high-performance fully differential folded cascode op-amp has been designed in Mentor Graphics AMS Design Architect environment initially. Figure 4.4 shows the schematic diagram of the designed op-amp. The specifications of this op-amp, with a load capacitance of 4 pF at each output are as follows: Gain = $76.5dB$, BW = $200MHz$, $3-dB$ cut-off = $55kHz$, SR = $182,4\ V/\mu s$. The output DC level of the op-amp is $1.472V$ and the bias voltages are $V_{bias1} = 2.1V$, $V_{bias2} = 1.83V$.

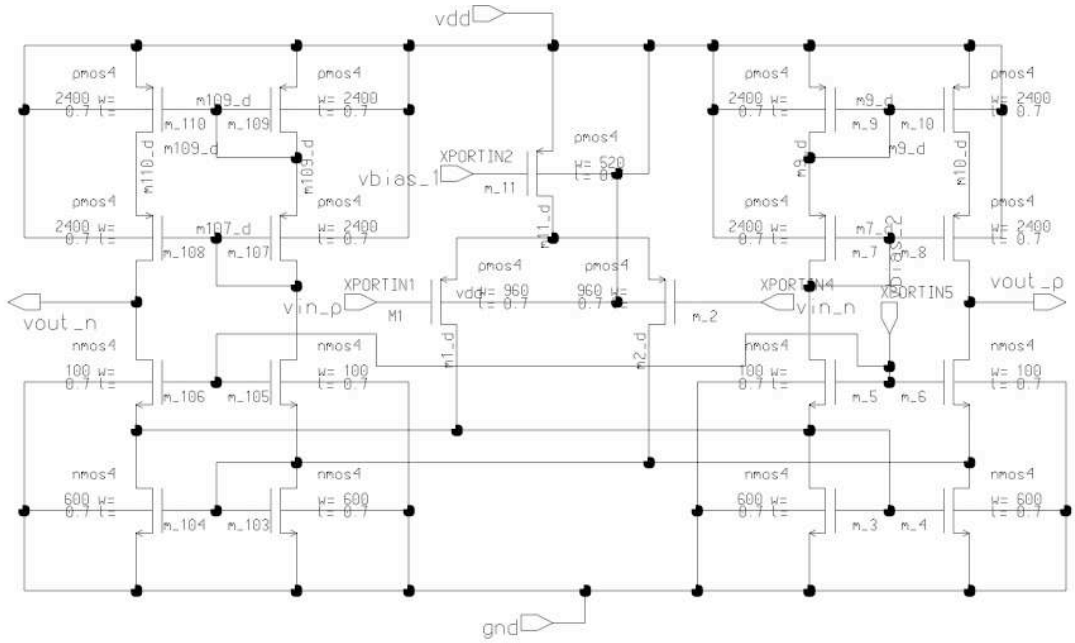

Figure 4.4. A fully differential folded cascode op-amp

The op-amp has been designed as folded cascode, in order to have a high gain by using only a single stage, and consequently to avoid any need for frequency compensation, which decreases SR, and in turn the speed of the op-amp. Further information about folded cascode configuration may be found in [21].

By using this op-amp the switched capacitor integrator shown in Figure 3.1 has been implemented by selecting $C_{s1} = C_{s2} = 1pF$ and $C_{f1} = C_{f2} = 2pF$. This integrator has been simulated by giving a differential input voltage of $V_{in} = 100mV$. The integrator non-ideality parameters have been extracted as $b = 0.4994$ and $c = 0.9996$ for this op-amp from this simulation. For the case of selected capacitor values, the ideal values of the parameters b and c are 0.5 and 1. So the calculated values denote that this op-amp may be considered as a very high-performance op-amp since the deviation of these parameters from their ideal values are very small.

As mentioned earlier, the component non-idealities have great impact in the performance of SD modulators. A reliable design automation tool has to take these non-idealities into account and generate solutions so as to compensate for the degrading effects of them. This tool has the capability of doing this, as will be demonstrated in Chapter 5, and this is one of its major advantages over the previous work in this area.

The program codes for the automatic architecture generator may be found in Appendix A. Chapter 5 will focus on the application of the tool by demonstrating it on various examples.

5. APPLICATION OF THE AUTOMATIC ARCHITECTURE GENERATOR

This chapter will focus on the application of the tool described in Chapter 4. Firstly, the application of the tool on a standard second order system ignoring component non-idealities will be discussed. Then the case with including the non-idealities will be demonstrated by examples for three different cases: (1) An ideal op-amp case, (2) a high-performance op-amp case, (3) a low-performance op-amp case.

5.1. Generation of Standard Second Order Architectures Ignoring the Component Non-idealities

For this case, the tool has been run by inputting exactly the netlist shown in Figure 4.2. The integrators are assumed to have the ideal transfer function given in Eq (2.8). The numeric transfer functions desired to be realized were;

$$\begin{aligned} STF(z) &= z^{-2} \\ NTF(z) &= (1-z^{-1})^2 \end{aligned} \tag{5.1}$$

which define the standard second order system response, as $STF(z)$ being an all-pass filter with two unit delays and $NTF(z)$ being a second order high-pass shaping function.

The tool has generated 70 different architectures for this case. Out of these 70 architectures, five are given as examples in Table 5.1. The solution in the first column of Table 5.1 is the standard second order SD modulator architecture shown in Figure 5.1. This can be verified by making $g_9 = g_{13} = 1$.

In fact a rather more important result is observed in the third column. The architecture proposed in the third column is extremely interesting in the sense that the

Table 5.1. Architectures Found for the Standard Second Order Response

Coeffs.	Different Solutions				
	1	2	3	4	5
g_1	g_{13}	$\frac{1}{g_4 g_9}$	0	$\frac{4}{g_8}$	g_1
g_2	0	0	0	0	0
g_3	0	0	-0.5	0.5	0.5
g_4	$\frac{1}{g_9 g_{13}}$	g_4	g_4	$\frac{1}{4g_{14}}$	$\frac{1}{g_1 g_9}$
g_5	0	$\frac{2}{g_9}$	$-\frac{4}{g_6}$	0	0
g_6	0	0	g_6	0	0
g_7	0	0	-4	-4	0
g_8	0	0	0	g_8	0
g_9	g_9	g_9	$-\frac{g_6}{2}$	0	g_9
g_{10}	0	0	0	0	0
g_{11}	0	0	0.5	-0.5	-0.5
g_{12}	$\frac{2}{g_9}$	0	0	$\frac{2}{g_8 g_{14}}$	$\frac{2}{g_9}$
g_{13}	g_{13}	$\frac{1}{g_9 g_{14}}$	$-\frac{2}{g_4 g_6}$	0	0
g_{14}	0	0	$\frac{1}{4g_4}$	g_{14}	$\frac{g_1 g_9}{4}$
g_{15}	0	$-2g_4 g_9$	0	$\frac{g_8}{2}$	0

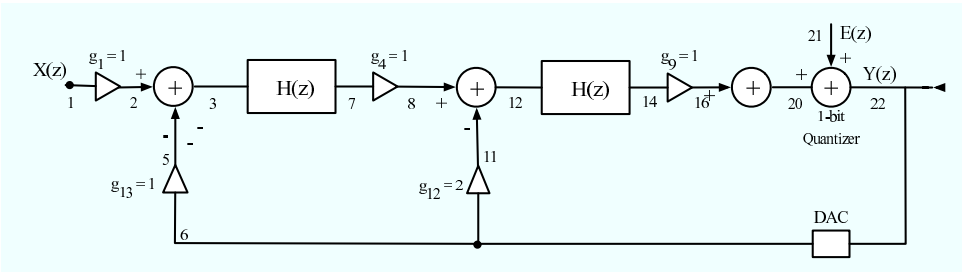

Figure 5.1. The standard second order SD modulator

input signal is not directly fed to the first integrator but given to the second one in the loop (see Figure 5.2). By making $g_3 = -0.5,\ g_4 = 1,\ g_5 = -2,\ g_6 = 2,\ g_7 = -4,\ g_9 =$

$-1,\ g_{11} = 0.5,\ g_{13} = -1,\ g_{14} = 0.25$ we obtain a very interesting topology which still can realize the desired frequency responses. The architectures shown in Figure 5.1 and

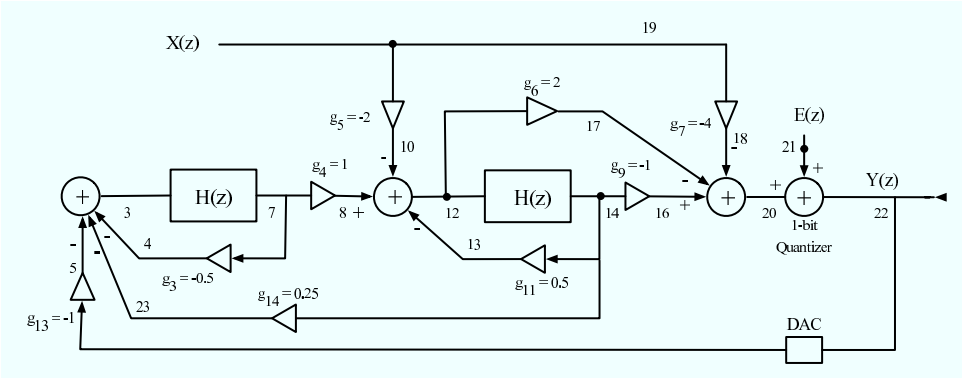

Figure 5.2. The solution proposed in column 3 of Table 5.1

5.2 have been implemented and simulated in MATLAB Simulink and the PSD plots for these architectures are shown in Figure 5.3. As can be seen from the PSD plots, these two architectures have exactly the same behavior. The SNR values are approximately 80 dB.

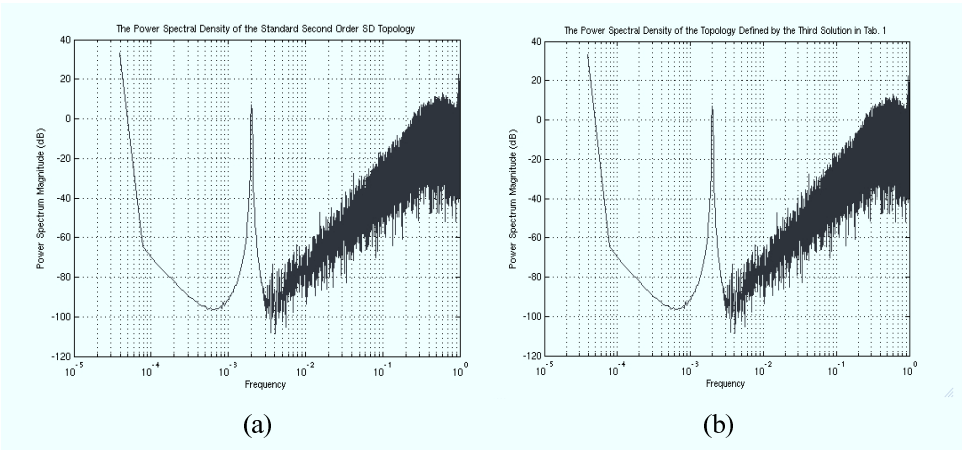

Figure 5.3. The PSD plot for the solution given in (a) Figure 5.1, (b) Figure 5.2

5.2. Generation of Standard Second Order Architectures Including Component Non-idealities

In this case, the tool has been applied to the same generic second order SD modulator architecture shown in Figure 4.1, but the integrators have been changed to non-ideal. That is, the lines 22 and 23 in the netlist in Figure 4.2 have been changed

properly to define the integrators as non-ideal and this modified netlist has been given to the tool as input. Three different combinations of the non-ideality parameters b and c have been used for these simulations: (1) $b = 0.5,\ c = 1$, corresponding to an ideal op-amp (or integrator) case, (2) $b = 0.4994,\ c = 0.9994$, corresponding a high-performance op-amp case and, (3) $b = 0.496,\ c = 0.98$, corresponding to a low-performance op-amp case. The desired response to be realized was again the same standard second order SD modulator response given in Equation (5.1).

5.2.1. Ideal Op-amp Case

Here the non-ideality parameters have been used as $b = 0.5$ and $c = 1$ taking $C_f = 1pF$ and $C_s = 0.5pF$ for the SC integrator of Figure 3.1. The tool found 135 different architectures satisfying the desired response. One simple solution was $g_1 = 1,\ g_2 = 0,\ g_3 = -0,\ g_4 = 2,\ g_5 = 0,\ g_6 = 0,\ g_7 = 0,\ g_8 = 0,\ g_9 = 2,\ g_{10} = 0,\ g_{11} = 0,\ g_{12} = 2,\ g_{13} = 1,\ g_{14} = 0,\ g_{15} = 0$. This topology has been implemented and simulated in MATLAB Simulink and the SNR value has been found to be 101.5 dB.

5.2.2. High-Performance Op-amp Case

Here the non-ideality parameters have been used as $b = 0.4994$ and $c = 0.9996$ which have been extracted from the folded-cascode op-amp shown in Figure 4.4. The tool has found 84 different architectures satisfying the desired response. One of the solutions was $g_1 = 1,\ g_2 = 0,\ g_3 = -0.0008,\ g_4 = 2.0048,\ g_5 = 0,\ g_6 = 0,\ g_7 = 0,\ g_8 = 0,\ g_9 = 2,\ g_{10} = 0,\ g_{11} = -0.0008,\ g_{12} = 2.0024,\ g_{13} = 1,\ g_{14} = 0,\ g_{15} = 0$. The coefficients differ from the ideal case just slightly.

The main difference is the two local positive feedback paths, which the tool puts around the integrators in order to compensate for the leakage non-ideality. The most important thing observed here was that; this op-amp has performed very close to the ideal case even with the ideal op-amp case coefficients given in Section 5.2.1, with an SNR of 96.89 dB. Simulations for two more cases have been carried on with this op-

amp, one with the coefficients calculated by the tool but the local feedback coefficients (g_3 and g_{11}) set to zero, and one with all of the calculated coefficients included including the local feedback ones. The SNR values have been found as 98.7 dB and 100.8 dB respectively, which means that if the op-amp is a high-performance one, neglecting the calculated local feedback paths does not cause much degradation in the modulator performance.

5.2.3. Low-Performance Op-amp Case

It has to be noted that, the power consumption of the op-amp given in Figure 4.4 is rather high in order to obtain a high SR. There may exist some applications in which power consumption may be a limiting factor and it may be necessary to use an op-amp with low power consumption. For such an op-amp, the effects of integrator non-idealities become more significant and there exists a higher deviation in the values of the parameters b and c compared to their ideal values. Such a case has been simulated by using $b = 0.496$ and $c = 0.98$ which corresponds to a low-performance op-amp.

The tool has found 82 different topologies satisfying the desired frequency response in this case. One of the solutions was $g_1 = 1,\ g_2 = 0,\ g_3 = -0.0403,\ g_4 = 2.0324,\ g_5 = 0,\ g_6 = 0,\ g_7 = 0,\ g_8 = 0,\ g_9 = 2,\ g_{10} = 0,\ g_{11} = -0.0403,\ g_{12} = 2.01613,\ g_{13} = 1,\ g_{14} = 0,\ g_{15} = 0$. The deviation from the ideal op-amp case is higher now, compared to the high-performance op-amp case. The tool again puts two local positive feedback paths around the integrators to compensate for the leakage non-ideality. Three different behavioral simulations have been carried on for this low-performance op-amp case; one with the ideal op-amp coefficients given in Section 5.2.1, one with the calculated coefficients by the tool but the positive local feedback coefficients (g_3 and $g_1 1$) set to zero, and finally one with all of the calculated coefficients including the local feedback ones. These simulations resulted in SNR values of 74 dB, 84.14 dB and 100.74 dB respectively. The PSD plots for the first case (the low-performance op-amp and the ideal op-amp coefficients) and the last case (the low-performance op-amp and all of the non-ideal coefficients calculated by the tool) is given in Figure 5.4.

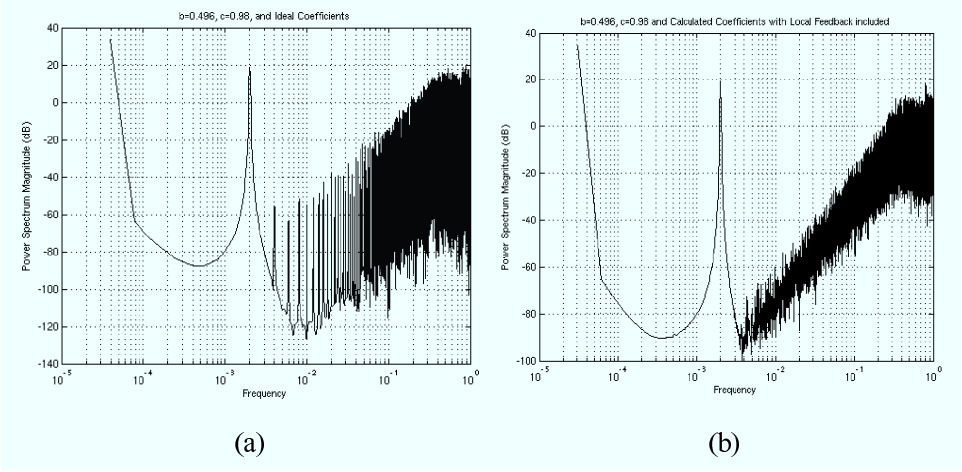

Figure 5.4. The PSD plot for the case with the low-performance op-amp and (a) the ideal op-amp case coefficients (b) the non-ideal coefficients calculated by the tool including the local feedback coefficients g_3 and g_{11}

5.2.4. Comparison of the Results

Several important results may be obtained from the simulation data presented in the previous sections. If the SNR values in Table 5.2 are analyzed, it can be seen that the SNR value for the case of high-performance op-amp ($b = 0.9449$, $c = 0.9996$) with the ideal op-amp coefficient values is very close to the SNR value for the case of an ideal op-amp and the ideal coefficient set. This means that if the op-amp is designed in such a good way that the variations of b and c from their ideal values are very small; the SD modulator works pretty fine with the ideal system coefficients and no extra paths are necessary to obtain a satisfactory system performance.

Rather more interesting results are obtained with the low-performance op-amp. In Figure 5.4, the PSD plot for an SD modulator with a low-performance op-amp (b=0.496, c=0.98) and ideal op-amp coefficient set is shown. It can easily be seen that the response is much worse than that of a standard second order SD modulator. The SNR value for this case (74 dB) is obviously much lower than that of the ideal op-amp case, which is 101.5 dB. These results give an estimate about how much the system response may degrade due to the component non-idealities.

On the other hand, the PSD plot of the SD modulator with the low-performance

Table 5.2. SNR values in dB for Different Simulation Scenarios

Parameter sets	**Coefficient Sets**		
	Ideal Op-amp Case Coeffs.	Calculated Non-ideal Coeffs. Ignoring Local Feedback	Calculated Non-ideal Coeffs. Including Local Feedback
$b = 0.5$ $c = 1$	101.5	-	-
$b = 0.4994$ $c = 0.9996$	96.89	98.7	100.8
$b = 0.496$ $c = 0.98$	74	84.14	100.74

op-amp and with all the coefficients calculated by the automatic architecture generator tool (including the local feedback ones also) is shown in Figure 5.4. As can be seen, the PSD plot for this case is very close to that of an ideal second order system shown in Figure 5.3a. The SNR value in this case is very close to the ideal value of 101.5 dB. This means that the tool is very successful in modeling the non-idealities and proposing architectures to compensate for their effects on the system performance.

These results are very important because since the tool is so successful in compensating for the effects of system non-idealities, it loosens the performance constraints of the analog components to be used in designing SD modulators. Hence, it can easily be possible to obtain very high-performance modulator architectures utilizing rather low-performance analog components.

Another important issue to be questioned here was what would happen if the extra coefficients proposed by the tool (g_3 and g_{11} in our case) were omitted since their values are rather small compared to the other coefficients. The results show that, for the high-performance op-amp case, omitting them does not make such a big difference, the SNR decreases to 98.7 dB from 100.8 dB. However, the situation is different for

the case of the low-performance op-amp. Omitting the extra coefficients decreases the SNR from 100.74 dB to 84.14 dB, which is a big loss.

5.3. Further Examples on Application of the Tool

The tool is not limited to realize a specific desired frequency response only. Although all the examples up to now were about realizing the frequency response defined in Equation (5.1), the tool is capable of realizing any other response as long as the desired numeric STF and NTF are theoretically realizable and the location of poles of both transfer functions are the same. Further examples will be demonstrated in the following sections.

5.3.1. Realizing a Band-Pass STF and a Notch NTF

The following transfer functions define a BP STF and a notch NTF response:

$$STF(z) = \frac{z^2 - 1}{z^2 + 0.8} \tag{5.2}$$

$$NTF(z) = \frac{z^2 + 1}{z^2 + 0.8} \tag{5.3}$$

Such an STF allows only a very narrow frequency region to pass and suppresses the rest. Such an NTF on the other hand, suppresses the noise at a very narrow frequency range corresponding to the range of signal frequency allowed by the STF, and allows the rest of the noise signal to pass. Such a response may be desired to be realized in specific telecommunication applications such as in designing a narrow-band receiver. When the automatic architecture generator tool is invoked to realize the response above by the generic architecture given in Figure 4.1, the number of different architectures generated is 35. Five out of these 35 solutions are given in Table 5.3 as example.

Table 5.3. Architectures Found for the BP STF and Notch NTF

Coeffs.	Different Solutions				
	1	2	3	4	5
g_1	0	$-\frac{2}{g_4 g_9}$	$10g_{13}$	0	g_1
g_2	0	0	0	$-\frac{2}{g_{14}}$	$\frac{2}{g_1 g_9}$
g_3	2	0	2	0	0
g_4	$\frac{2}{g_{14}}$	g_4	g_4	$\frac{2}{g_{14}}$	$-\frac{2}{g_1 g_9}$
g_5	$-\frac{1}{g_8 g_{14}}$	0	0	$\frac{2}{g_{14} g_{15}}$	0
g_6	0	0	0	0	0
g_7	-1	-1	-1	-1	-1
g_8	g_8	0	0	0	0
g_9	$-g_8 g_{14}$	g_9	$-\frac{0.2}{g_4 g_{13}}$	0	g_9
g_{10}	0	0	0	0	0
g_{11}	0	2	0	0	0
g_{12}	$-\frac{0.1}{g_8 g_{14}}$	0	0	$-\frac{0.2}{g_{14} g_{15}}$	0
g_{13}	0	$-\frac{0.2}{g_4 g_9}$	g_{13}	0	$0.1g_1$
g_{14}	g_{14}	$\frac{2}{g_4}$	$\frac{2}{g_4}$	g_{14}	$-g_1 g_9$
g_{15}	0	0	0	g_{15}	$-\frac{2}{g_1}$

5.3.2. Realizing a Delay-less STF ($STF = 1$) and Standard Second Order NTF

In some cases it may be desired to send the signal directly to the output without a delay. This may provide an advantage in minimizing the power dissipation of the architectures since the output is subtracted from the input via the feedback path from the DAC to the input of the first integrator (See Figure 4.1). This subtraction decreases the amplitude of the signal at the integrator input and may in turn decrease the power

Table 5.4. Architectures Found for Delayless STF and High-pass NTF

Coeffs.	Different Solutions				
	1	2	3	4	5
g_1	0	$\frac{1}{g_4 g_9}$	$\frac{1}{g_4 g_9}$	$-\frac{2}{g_4 g_6}$	$\frac{1}{g_4 g_9}$
g_2	0	0	0	0	g_2
g_3	$\frac{1}{2}$	0	0	g_3	0
g_4	g_4	g_4	g_4	g_4	g_4
g_5	$-\frac{2}{g_9}$	$-\frac{2}{g_9}$	0	0	$-\frac{g_2}{g_4 g_9}$
g_6	0	0	$-\frac{2}{g_9}$	g_6	$-2g_9$
g_7	-1	-1	-1	-1	-1
g_8	0	0	0	0	0
g_9	g_9	g_9	g_9	$-\frac{1}{2}g_6$	g_9
g_{10}	0	0	0	0	0
g_{11}	$-\frac{1}{2}$	0	0	$-g_3$	0
g_{12}	$\frac{2}{g_9}$	$\frac{2}{g_9}$	0	0	$\frac{g_2}{g_4 g_9}$
g_{13}	0	$\frac{1}{g_4 g_9}$	$\frac{1}{g_4 g_9}$	$-\frac{2}{g_4 g_6}$	$\frac{1}{g_4 g_9}$
g_{14}	g_{14}	0	0	$\frac{g_3^2}{g_4}$	0
g_{15}	0	0	0	0	0

dissipation. For this case the input netlist was still the one shown in Figure 4.2 and the tool has been invoked to realize the response given in Equation (5.4) and Equation 5.5.

$$STF(z) = 1 \tag{5.4}$$

$$NTF(z) = (1 - z^{-1})^2 \tag{5.5}$$

The tool has generated 40 different architectures for this case and five out of these 40 are given in Table 5.4.

5.3.3. Application of the Tool on a Third Order System

As mentioned earlier in the text, the automatic architecture generator tool is independent of the modulator order. A single-loop third order SD modulator architecture comprising all possible feedback and feedforward paths is shown in Figure 5.5. This architecture has been defined as a netlist in a similar manner to the one shown in Figure 4.2 and the architecture generator tool has been invoked by inputting that netlist in order to realize the following standard third order SD modulator response:

$$STF(z) = z^{-3} \tag{5.6}$$

$$NTF(z) = (1 - z^{-1})^3 \tag{5.7}$$

The tool has found 3107 different architectures satisfying the response given in Equa-

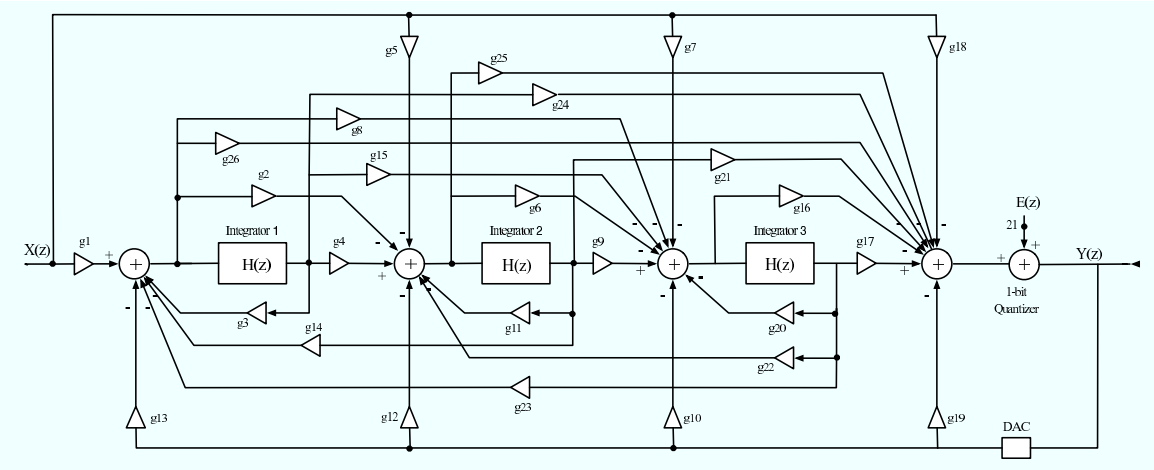

Figure 5.5. A third order SD modulator architecture comprising all possible feedbac and feedforward paths

tion (5.6) and Equation (5.7). Five out of these 3107 solutions have been given in Table 5.5.

Table 5.5. Architectures Found for the Third Order System

Coeffs.	Different Solutions				
	1	2	3	4	5
g_1	$\frac{1}{g_{24}}$	$\frac{1}{3}\frac{g_7}{g_4 g_9}$	g_{13}	g_{13}	0
g_2	0	0	0	0	0
g_3	0	0	0	0	$-\frac{1}{3}$
g_4	g_4	g_4	g_4	$\frac{1}{3}\frac{g_5}{g_{13}}$	g_4
g_5	0	0	0	g_5	0
g_6	0	0	0	0	0
g_7	g_7	g_7	0	0	$-\frac{1}{g_4 g_{21} g_{23}}$
g_8	0	0	0	0	0
g_9	0	g_9	g_9	g_9	0
g_{10}	$-g_7$	0	$\frac{3}{g_{17}}$	$\frac{1}{3}(3 + g_5 g_{21}) g_5 g_9$	0
g_{11}	0	0	0	0	0
g_{12}	0	$\frac{g_7}{g_9}$	$3 g_4 g_{13}$	0	$-\frac{3}{g_{21}}$
g_{13}	$-\frac{2}{g_{24}}$	$\frac{1}{3}\frac{g_7}{g_4 g_9}$	g_{13}	g_{13}	$-\frac{3}{g_4 g_{21}}$
g_{14}	0	0	0	0	0
g_{15}	0	0	0	$-\frac{(9+g_5 g_{21}) g_5 g_9}{9 g_{13}}$	$-\frac{1}{9 g_{23}}$
g_{16}	0	0	0	0	0
g_{17}	$-\frac{1}{g_7}$	$\frac{3}{g_7}$	g_{17}	$\frac{3}{g_5 g_9}$	0
g_{18}	0	0	0	0	0
g_{19}	0	0	0	0	0
g_{20}	0	0	0	0	$\frac{1}{3}$

Continued on next page

Coeffs.	Different Solutions				
	1	2	3	4	5
g_{21}	$\frac{g_{24}}{g_4}$	0	0	g_{21}	g_{21}
g_{22}	0	0	0	0	0
g_{23}	$-\frac{1}{g_7 g_{24}}$	0	0	0	g_{23}
g_{24}	g_{24}	$-\frac{9}{g_4 g_7 g_9}$	0	$\frac{g_5 g_{21}}{g_{13}}$	0
g_{25}	0	0	0	0	0
g_{26}	0	0	0	0	0

In this chapter, the automatic architecture generator tool has been demonstrated. Several examples have been given about the application of the tool. Most of the examples were about the application of the tool on second order systems but it has to be reminded that, as explained in Section 2.2.3, it is possible to construct higher order modulators by cascading several first and second order stages. Hence this tool can easily be used in the design of cascaded higher-order modulators, since each stage of a cascaded higher-order modulator can be generated by applying this tool. Generation of single-loop high-order modulators by applying this tool may not be feasible since the amount of time required may increase dramatically. In fact, implementing high-order modulators in single-loop fashion is not logical since the stability condition for this type of modulators cannot be derived mathematically as described in Section 2.2.2.1.

6. CONCLUSIONS AND FUTURE WORK

The use of SD based A/D converters for the primary data conversion is very attractive, since it uses basic blocks and requires no sample and hold. Besides the advantages of this technique, there are some difficulties in designing SD A/D converters. One of the major difficulties is the determination of the appropriate SD structure, which provides the required performance. Since SD A/D converters contain large number of connections between building blocks (quantizer(s), integrator(s), DAC) there are many structures satisfying the desired performance specifications for a required application and generally the design procedure gets extremely complicated. In order to decrease the complexity of the design procedure, automation tools have been developed.

A standard SD modulator system utilizes several signal feedforward and signal feedback paths. For every such signal path, there exists an associated coefficient, which is the path gain. These coefficients are all related to each other and a small change in one of them may cause dramatic changes in the operation, response, and performance of the overall modulator. On the other hand, including or removing any of these paths corresponds to a different modulator topology. Another important issue is that there are many non-idealities in the SD modulator system in real-life, which should also be taken into account in the design process. These non-idealities may include the clock-jitter, kT/C noise, op-amp noise, and integrator non-idealities such as the finite DC gain (leakage), the switched capacitor mismatches, slew-rate limitation of the op-amp, dc offset of the comparator, etc.

So, in designing an SD modulator, the challenge is not only the selection of a modulator topology from a large set of possibilities, but also the optimization of the topology parameters the coefficients in such a way to satisfy the system specifications without ignoring of mentioned non-idealities.

In this thesis, a tool created in the MATLAB environment for automated design of SD modulator architectures has been proposed. The tool starts working on a generic

SD modulator architecture as shown in Figure 4.1. There is no limit other than time on the order or the complexity of this generic architecture. The parametric STF and NTF are calculated at first for the generic architecture automatically. These symbolic transfer functions are matched to a set of numerical transfer functions describing a definite, desired response to be realized by the generic architecture of interest. A set of equations is created in this manner. Since the number of coefficients is always greater than the number of equations, there is a high degree of freedom in the system, which makes it possible to have a set of parametric solutions for the architecture coefficients. The tool makes use of this degree of freedom and finds all the possible SD modulator topologies satisfying the desired frequency responses. It even finds some architectures, which look unconventional at first glance but have been proven to be working through behavioral simulation. The tool uses some criteria in this process such as minimization of the number of signal paths in the architecture, and avoiding the occurrences of closed signal loops that has no delay.

Although there are several other tools proposed for the solution of the design automation problem of the SD modulators, this tool has many advantages over the others. First of all, it has a symbolic analyzer which works in a SPICE-like fashion; that is, it takes a netlist of an SD modulator architecture of any order and any complexity in block level as the input, determines the input-output relation for each block in z-domain and generates an equation for each node of the architecture in terms of symbolic variables. This symbolic analyzer may be used as a stand-alone tool and the designer may utilize this analyzer to evaluate various design alternatives at the block level. It is not only independent of the order of the architecture, but also independent of the actual blocks, whereby the user may add new functional blocks to the tool. Another advantage is that, the tool optimizes both STF and NTF simultaneously, not only one at a time. Also it spans all the solution space according to the criteria mentioned above and returns parametric solutions, not only a few strict numeric solutions.

Most important of all, this tool has the capability of taking the component non-idealities into account and optimizing the coefficients such a way to compensate the undesired effects of these non-idealities. Most of the component non-idealities can be

mapped directly to the transfer function of the system and most of them are related to the switched capacitor integrator in the SD modulator system. For this reason, the tool focuses on modeling the integrator non-idealities as mentioned earlier.

Despite all its advantages described up to now, of course the tool has some weaknesses which may be defined as open problems and future work in this area. The major problem is the architecture selection problem. The tool proposed here generates many architectures realizing a definite frequency response but it gives no idea about which one of these architectures is the best in terms of some performance criteria such as power dissipation, sensitivity, area, etc. In fact this architecture selection problem is the subject of another thesis which has been progressing simultaneously with this one. Furthermore, the architecture generation and selection problem may be defined as a non-linear discrete optimization problem if well-defined cost functions can be constructed for the performance metrics (power dissipation, sensitivity, area, etc.).

Another questionable part of this work is the strict definition of the desired response with the numeric z-domain transfer function as given in Equation (4.7) and Equation (4.8). As another approach, the desired frequency response may be defined with looser constraints, for example by some ranges for band frequencies and some ranges for the magnitude responses corresponding to the defined frequency bands with a given amount of tolerance. In such a case, many different numeric transfer functions may be generated in the given tolerance boundaries. Then the generated architectures and the desired frequency response may be optimized simultaneously to give the best solution. Maybe one numeric transfer function, after including the non-idealities, will result in much simpler architectures and much lower power dissipation will be obtained in the presence of system non-idealities.

Also the assignment of zeros to maximum number of coefficients, as shown in Figure 4.3, is done somewhat randomly. A better and more intelligent algorithm may be proposed to minimize the number of signal paths in the architecture.

Finally; considering all the advantages and disadvantages discussed above, it can be said that the approach of the tool proposed in this work to the problem of design automation and architecture generation for SD modulators, has many innovative parts from all aspects. Also the results shown in the preceding sections prove that the tool works very well and is quite successful in proposing a new solution to the architecture generation problem of SD modulators.

APPENDIX A: THE SOURCE CODES

The whole automatic architecture generator consists of the following functions and scripts written in MATLAB environment.

1. **symbolic_tfs.m:** The symbolic transfer functions are generated via this script.

```
[equation, gains, var] = tf_gen2('generic_2nd_omer_15gain.txt');
syms z E IN;
nodes = solve(equation,var);
var_array = split(var, ',');
var_array(1);
for m = 1 : length(var_array)
    eval(['syms ' var_array{m}])
    eval([var_array{m} '= nodes.' var_array{m};]);
end
```

It invokes the "tf_gen2.m" function which is as follows:

```
function [total_equation, no_of_gains, variables] = tf_gen(scriptname)
syms z IN OUT E b c;
fid = fopen(scriptname);
i = 1;
no_of_gains = 1;
no_of_equations = 0;
variables = [];
while (feof(fid) ~= 1)
tline = fgetl(fid);
[T,R] = strtok(tline);
if length(T) == length('IN')
if T == 'IN'
[component(i).in1, component(i).dummy] = strtok(R);
```

```
equation(i).out = ['x' component(i).in1 '= IN'];
node = ['x' component(i).in1];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
total_equation = equation(i).out;
no_of_equations = no_of_equations + 1;
end
end
if length(T) == length('NOISE')
if T == 'NOISE'
[component(i).in1, component(i).dummy] = strtok(R);
equation(i).out = ['x' component(i).in1 '= E'];
node = ['x' component(i).in1];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
total_equation = [total_equation ',' equation(i).out];
no_of_equations = no_of_equations + 1;
end
end
if length(T) == length('ADDER')
if T == 'ADDER'
node_array = split(R, ' ')
equation(i).out = ['x' char(node_array(length(node_array))) '=']
node = ['x' char(node_array(length(node_array)))]
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
for j = 1:length(node_array)-1
if str2num(char(node_array(j)))< 0
equation(i).out = [equation(i).out '-x'
```

```
num2str(abs(str2num(char(node_array(j)))))]
node = ['x' num2str(abs(str2num(char(node_array(j)))))]
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
else
equation(i).out = [equation(i).out '+x' char(node_array(j))]
node = ['x' num2str(abs(str2num(char(node_array(j)))))]
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
end
end
total_equation = [total_equation ',' equation(i).out];
no_of_equations = no_of_equations + 1;
end
end
if length(T) == length('QUANTIZER')
if T == 'QUANTIZER'
[component(i).in1, component(i).out] = strtok(R);
[component(i).out, component(i).dummy] = strtok(component(i).out);
equation(i).out = ['x' component(i).out '=' 'x' component(i).in1];
node = ['x' component(i).in1];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
node = ['x' component(i).out];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
total_equation = [total_equation ',' equation(i).out];
no_of_equations = no_of_equations + 1;
```

```
end
end
if length(T) == length('GAINXX')
if T(1:4) == 'GAIN'
index=T(5:6);
[component(i).in1, component(i).out] = strtok(R);
[component(i).out, component(i).dummy] = strtok(component(i).out);
equation(i).out = ['x' component(i).out '=' 'g' num2str(index) '*'
'x' component(i).in1];
node = ['x' component(i).in1];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
node = ['x' component(i).out];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
total_equation = [total_equation ',' equation(i).out];
no_of_equations = no_of_equations + 1;
no_of_gains = no_of_gains + 1;
end
end
if length(T) == length('GAINX')
if T(1:4) == 'GAIN'
index=T(5);
[component(i).in1, component(i).out] = strtok(R);
[component(i).out, component(i).dummy] = strtok(component(i).out);
equation(i).out = ['x' component(i).out '=' 'g' num2str(index) '*'
'x' component(i).in1];
node = ['x' component(i).in1];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
```

```
end
node = ['x' component(i).out];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
total_equation = [total_equation ',' equation(i).out];
no_of_equations = no_of_equations + 1;
no_of_gains = no_of_gains + 1;
end
end
if length(T) == length('DAC')
if T == 'DAC'
[component(i).in1, component(i).out] = strtok(R);
[component(i).out, component(i).dummy] = strtok(component(i).out);
equation(i).out = ['x' component(i).out '=' 'x' component(i).in1];
node = ['x' component(i).in1];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
node = ['x' component(i).out];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
total_equation = [total_equation ',' equation(i).out];
no_of_equations = no_of_equations + 1;
end
end
if length(T) == length('INTEGRATORD')
if T == 'INTEGRATORD'
[component(i).in1, component(i).out] = strtok(R);
[component(i).out, component(i).dummy] =
strtok(component(i).out);
```

```
equation(i).out = ['x' component(i).out '=
x' component(i).in1 '/(z-1)'];
node = ['x' component(i).in1];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
node = ['x' component(i).out];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
total_equation = [total_equation ',' equation(i).out];
no_of_equations = no_of_equations + 1;
end
end
if length(T) == length('INTEGRATOR')
if T == 'INTEGRATOR'
[component(i).in1, component(i).out] = strtok(R);
[component(i).out, component(i).dummy] =
strtok(component(i).out);
equation(i).out = ['x' component(i).out '=
x' component(i).in1 '*z/(z-1)'];
node = ['x' component(i).in1];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
node = ['x' component(i).out];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
total_equation = [total_equation ',' equation(i).out];
no_of_equations = no_of_equations + 1;
end
```

```
end
if length(T) == length('NONIDEAL_INTEGRATORD')
if T == 'NONIDEAL_INTEGRATORD'
[component(i).in1, component(i).out] = strtok(R);
[component(i).out, component(i).dummy] =
strtok(component(i).out);
equation(i).out = ['x' component(i).out '=
x' component(i).in1 '*b/(z-c)'];
node = ['x' component(i).in1];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
node = ['x' component(i).out];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
total_equation = [total_equation ',' equation(i).out];
no_of_equations = no_of_equations + 1;
end
end
if length(T) == length('NONIDEAL_INTEGRATOR')
if T == 'NONIDEAL_INTEGRATOR'
[component(i).in1, component(i).out] = strtok(R);
[component(i).out, component(i).dummy] =
strtok(component(i).out);
equation(i).out = ['x' component(i).out '=
x' component(i).in1 '*(b*z)/(z-c)'];
node = ['x' component(i).in1];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
node = ['x' component(i).out];
```

```
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
total_equation = [total_equation ',' equation(i).out];
no_of_equations = no_of_equations + 1;
end
end
if length(T) == length('OUT')
if T == 'OUT'
[component(i).out, component(i).dummy] = strtok(R);
eval(['syms x' component(i).out]);
equation(i).out = ['x' component(i).out '= OUT'];
node = ['x' component(i).out];
if isempty(strfind(variables, node)) == 1
variables = [variables ',' node]
end
total_equation = [total_equation ',' equation(i).out];
no_of_equations = no_of_equations + 1;
end
end
i = i + 1;
end
variables = variables(2:length(variables))
end
```

2. **equgen.m:** The matching of symbolic transfer functions to the numeric transfer fucntions are done by this function. Then the bunch of equations to be solved are created.

```
function denklem = equgen(tf, response_tf)

syms IN E z b c
denklem = [];
```

```
tf = collect(simplify(tf));

[tf_num tf_den] = numden(tf);
[response_tf_num response_tf_den] =
tfdata(response_tf, 'v')

[c_tf_num t_tf_num] = coeffs(collect(tf_num), z)
[c_tf_num_sort, t_tf_num_sort] =
coeffsort(c_tf_num, t_tf_num, z)

[c_tf_den t_tf_den] = coeffs(collect(tf_den), z)
[c_tf_den_sort, t_tf_den_sort] =
coeffsort(c_tf_den, t_tf_den, z)

if t_tf_num_sort(length(t_tf_num_sort)) == 'z'
tf_num_order = 1;
else
tf_num_order_char = char(t_tf_num_sort(1))
tf_num_order_char =
tf_num_order_char(3:length(tf_num_order_char));
tf_num_order = str2num(tf_num_order_char)
end

if t_tf_den_sort(length(t_tf_den_sort)) == 'z'
tf_den_order = 1;
else
tf_den_order_char = char(t_tf_den_sort(1))
tf_den_order_char =
tf_den_order_char(3:length(tf_den_order_char));
tf_den_order = str2num(tf_den_order_char)
end
```

```
diff = tf_num_order - length(response_tf_num) + 1
diff2 = tf_den_order - length(response_tf_den) + 1

if diff < 0
display('This response cannot be realized by this topology!')
return;
elseif diff > 0
response_tf_num = fliplr(response_tf_num)
for i = 1 : diff
response_tf_num = [response_tf_num, 0];
end
response_tf_num = fliplr(response_tf_num)
end

if diff2 < 0
display('This response cannot be realized by this topology!')
return;
elseif diff2 > 0
response_tf_den = fliplr(response_tf_den)
for i = 1 : diff2
response_tf_den = [response_tf_den, 0];
end
response_tf_den = fliplr(response_tf_den)
end

for j = 0 : tf_num_order
if t_tf_num_sort(tf_num_order + 1 - j) ~= z^j
t_tf_num_sort(length(t_tf_num_sort) + 1) = z^j
c_tf_num_sort(length(t_tf_num_sort) + 1) = sym('0')
end
[c_tf_num_sort, t_tf_num_sort] =
```

```
coeffsort(c_tf_num_sort, t_tf_num_sort, z);
end

for j = 0 : tf_den_order
if t_tf_den_sort(tf_den_order +1 - j) ~= z^j
t_tf_den_sort(length(t_tf_den_sort) + 1) = z^j;
c_tf_den_sort(length(t_tf_den_sort) + 1) = sym('0');
end
[c_tf_den_sort, t_tf_den_sort]
= coeffsort(c_tf_den_sort, t_tf_den_sort, z);
end

for i = 1 : length(t_tf_num_sort)
denk = [char(c_tf_num_sort(i)) '=' num2str(response_tf_num(i))];
denklem = [denklem ',' denk];
end

for i = 1 : length(t_tf_den_sort)
denk = [char(c_tf_den_sort(i)) '=' num2str(response_tf_den(i))];
denklem = [denklem ',' denk];
end

denklem = denklem(2:length(denklem))

end
```

It uses the "coeffsort.m" [22] to sort the polynomial coefficinets here.

```
function [SortCoefVctr,SortSymPwrVctr] =
coeffsort(CoefVctr,SymPwrVctr,SortSym)
if isempty(SortSym)
fprintf(1,'\n\nError coeffsort  No sort symbol provided\n\n');
return
```

```
end

LenSymPwrVctr = length(SymPwrVctr);
SymPwrIdx = [];
if LenSymPwrVctr > 1
for k1 = 1:LenSymPwrVctr
SymPwrIdx(k1) = find(SymPwrVctr == SortSym^(k1-1));
end
for k1 = 1:LenSymPwrVctr
SortCoefVctr(k1)   = CoefVctr(SymPwrIdx(k1));
SortSymPwrVctr(k1) = SymPwrVctr(SymPwrIdx(k1));
end
SortCoefVctr   = fliplr(SortCoefVctr);
SortSymPwrVctr = fliplr(SortSymPwrVctr);
elseif LenSymPwrVctr == 1
SortCoefVctr   = CoefVctr;
SortSymPwrVctr = 1;
elseif (LenSymPwrVctr < 1)
| (isempty(CoefVctr)) | (isempty(SymPwrVctr))
fprintf(1,'\n\nError coeffsort  Empty argument vectors\n\n');
end

if isempty(SymPwrIdx)
SortCoefVctr   = CoefVctr*SymPwrVctr;
SortSymPwrVctr = 1;
end
return;
```

3. **generic.m:** The initial set of parametric. solutions are found by this script. It is invoked by the "generate_topology" script.

```
var = 'g1, g2, g3, g4, g5, g6, g7, g9, g11, g12, g13, g14, g15';
var_ = split(var, ',');
```

```
combos = nchoosek(var_, 4);
k = 1;
h = sym('0');
for i = 1 : length(combos)
eval(['denk.a' num2str(i) '=
subs(denklem2 , combos(' num2str(i) ',:), {0, 0, 0, 0});'])
end

for i = 1 : length(combos)

eval(['temp = char(denk.a' num2str(i) '(1));']);

for j = 2 : length(denk.a1)
temp = [temp ',' eval(['char(denk.a' num2str(i)
'(' num2str(j) '));'])];
end
temp;
t = solve(temp,var);
if isempty(t) ~= 1
for n = 1 : length(combos(1, :))
for index = 1 : length(t.g1)
eval(['t.' combos{i,n} '(' num2str(index) ')= h;']);
end
end
eval(['cozum15.a' num2str(k) '= t;']);
k = k + 1
else
continue
end
end
```

4. **generate_topology.m:** The "generic.m" script is invoked by this script and the initial set of solutions are obtained. Then the checks against delayless loops,

imaginary solutions, etc. are applied here

```
denklem2 = subs(denklem2, 'g8', 0);
denklem2 = subs(denklem2, 'g10', 0);

%denklem2 = subs(denklem2, c, 0.9996)
%denklem2 = subs(denklem2, b, 0.49942)

generic;
omer = omer15;
eliminate_rows;
omer_real =
subs(red_sol, var_, {10,10,10,10,10,10,10,10,10,10,10,10,10});

you = 1;
for i = 1 : length(omer_real)
if isreal(omer_real(i, :)) == 1
omer_son(you, :) = red_sol(i, :);
you = you + 1;
end
end
```

It also invokes the "eliminate_rows.m" script here.

```
red_sol(1,:) = omer(1,:)
for i = 2 : length(omer)
k = 1
siz = size(red_sol);
for j = 1 : siz(1)
if ifzero(omer(i,:), red_sol(j, :)) == 1
k = 0;
```

```
end
end
if k == 1
red_sol = [red_sol; omer(i,:)]
end
end
```

REFERENCES

1. Aziz, P. M., H. V. Sorensen, and J. V. D. Spiegel, "An overview of sigma-delta converters", *IEEE Signal Processing Magazine*, pp. 61-84, January 1996.

2. Candy, J. C. and G. C. Temes, *Oversampling Delta-Sigma Data Converters*, IEEE Press, New York, 1992.

3. Goodenough, F., "Analog Technologies of All Varieties Dominate ISSCC", *Electronic Design*, Vol. 44, pp. 96-111. February, 1996.

4. Medeiro, F., B. Perez-Verdu and A. Rodriguez-Vazquez, *Top-Down Design of High-Performance Sigma-Delta Modulators*, Kluwer Academic Publishers, Netherlands, 1999.

5. G. C. Temes and J. C. Candy: *A Tutorial Discussion of The Oversampling Method for A/D and D/A Conversion*, The IEEE Publications, 1990.

6. Williams, L., "Midas-a functional simulator for mixed digital and analog sampled data systems", *In proceedings IEEE International Symposium on Circuits and Systems*, pp. 2148-2151. San Diego, USA. May, 1992.

7. Francken, K. and G. E. Gielen, "A high-level simulation and synthesis environment for delta sigma modulators", *IEEE Transactions on Computer-Aided Design*, pp.1049-1061. August, 2003.

8. Brigati, S., F.Francesconi, P. Malcovati, D. Tonietto, A. Baschirotto and F. Maloberti, "Modeling Sigma-Delta Modulator Non-Idealities in Simulink", *In proceedings of International Symposium on Circuits and Systems*, 1999.

9. Ruiz-Amaya, J., J. Rosa, F. Medeiro, F. Fernandez, R. Rio, B. Perez-Verdu, and A. Rodriguez-Vazquez, "An optimization based tool for the high-level synthesis of

discrete-time and continuous-time SD modulators in the Matlab/Simulink environment", *In proceedings of IEEE International Symposium on Circuits and Systems*, pp. 97-100. Vancouver, Canada. May, 2004.

10. Tang, H., and A. Doboli: High-level synthesis of SD modulator topologies optimized for complexity, sensitivity, and power consumption, *IEEE Transactions on Computer-Aided Design*, pp 597 607. March, 2006.

11. Yetik, Ö., O. Saglamdemir, S. Talay and G. Dundar, "A Coefficient Optimization and Architecture Selection Tool for SD Modulators in MATLAB", *Design, Automation & Test in Europe Conference & Exhibition, 2007 (DATE '07)*, pp.1-6. April, 2007

12. Saglamdemir, O., Ö. Yetik, S. Talay, and G. Dündar, "A coefficient optimization and architecture selection tool for SD modulators considering component nonidealities", *In Proceedings of the 17th Great Lakes Symposium on Great Lakes Symposium on VLSI (GLSVLSI '07)*, ACM Press, New York, NY, pp. 423-428. March, 2007)

13. Norsworthy, S. R., R. Schereier and G. C. Temes (Editors), *"Delta Sigma Data Converters: Theory, Design and Simulation"*, IEEE Press, New York, 1997.

14. Bennet, W., "Spectra of Quantized Signals", *Bell Syst. Tech. J.*, Vol. 27, pp. 446-472, July 1948.

15. Gray, M. R., "Quantization Noise Spectra", *IEEE Transactions on Information Theory*, Vol. 36, pp. 1220-1244, November 1990.

16. Candy, J. C., "A Use of Double Integration in Sigma-Delta Modulation". *IEEE Transactions on Communications*, Vol. 33, pp. 249-258, March 1985.

17. Moussavi, S. M., and B. H. Leung, "High-Order Single-Stage Single-Bit Oversamling A/D Converter Stabilized with Local Feedback Loops", *IEEE Transactions on*

Circuits and Systems, Vol. 41, pp. 19-25, January 1994.

18. Lee, W. L., and C. G. Sodini, "A Topology for Higher Order Interpolative Coders", *In Proc. of IEEE International Symposium on Circuits and Systems*, pp. 459-462, 1987.

19. Schereier, R, "The Delta-Sigma Toolbox 7.1 (Online)", *http://www.mathworks.com/matlabcentral/fileexchange*, December 2004.

20. Robertini, A. and W. Guggenbühl, Errors in SC Circuits Derived from Linearly Modeled Amplifiers and Switches, *IEEE Transactions on Circuits and Systems*, Vol. 39, No. 2, February 1992.

21. Sedra, A. S. and K. C. Smith: *"Microelectronic Circuits* 4_{th} *Edition"*, Oxford University Press, New York, 1998.

22. Mathworks Software Inc. "MATLAB File Exchange", *http://www.mathworks.com/matlabcentral/fileexhange*