The Effects of Excess Loop Delay in Continuous-Time Sigma-Delta Modulators

by

Hyunjoo Jenny Lee

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2005

Author ....
Department of Electrical Engineering and Computer Science
August 15, 2005

Certified by ....
William Yang
VI-A Company
Thesis Supervisor

Certified by ....
Hae-Sung Lee
M.I.T.
~~Thesis Su~~pervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

The Effects of Excess Loop Delay in Continuous-Time Sigma-Delta Modulators

by

Hyunjoo Jenny Lee

Submitted to the Department of Electrical Engineering and Computer Science
on August 15, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Continuous-time sigma-delta (CT-$\Sigma\Delta$) modulators have recently received great attention in the academia as well as in the industry. Despite the improved understanding of the operation of CT-$\Sigma\Delta$ modulators, the problem due to excess loop delay that arises from timing mismatch and parasitic delay still remains unsolved. Thus, the thesis investigates the effects of the excess loop delay. In specific, the sensitivity of various CT-$\Sigma\Delta$ topologies to the excess loop delay is explored by converting the CT modulators to its DT equivalents and realizing loop filters in state-space representations in MATLAB ©.

Thesis Supervisor: William Yang
Title: VI-A Company

Thesis Supervisor: Hae-Sung Lee
Title: M.I.T.

Acknowledgments

I would like to thank my thesis supervisors, William Yang and Professor Hae-Sueng Lee for their patience and guidance for this thesis to come to a conclusion. I would also like to thank Anne Hunter, Lisa Bella, my friends, and family for their kind and continuous encouragement throughout my stays at M.I.T.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Oversampling sigma-delta ($\Sigma\Delta$) converters have become very popular during the last decade because they overcome some of inherent problems of conventional Nyquist-rate converters (Fig. 1-1). Conventional converters favorably sample at the lowest sampling frequency, but inconveniently require highly accurate analog circuitry. These characteristics have encumbered design and implementation of high-resolution, medium-to-low speed ADCs.

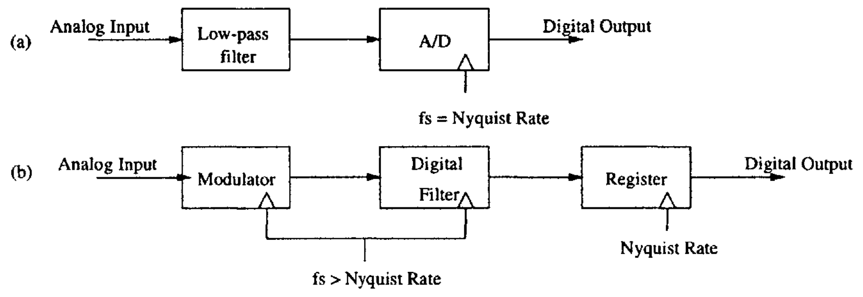


Figure 1-1: Block diagram of (a) conventional (b) oversampling A/D converters [14].

In contrast, oversampling converters relax the requirements on analog circuitry, such as matching tolerances and anti-aliasing specifications. This benefit is gained at the tolerable expense of higher sampling frequency and more stringent digital signal processing (DSP). In fact, implementing stricter DSP circuitry is more realizable in fine-line VLSI technology than implementing highly accurate and precise analog circuitry. Thus, oversampling converters provide a better solution for mixed-signal integration [9].

Sigma-delta converters are oversampling converters that incorporate sigma-delta modulation. By oversampling the input signal, applying coarse quantization and shaping the quantization noise spectrum, sigma-delta converters can produce high resolution in a relatively small bandwidth [3]. For this reason, sigma-delta converters are now common implementation of medium-to-low speed, high-resolution ADCs [9].

To this date, the majority of sigma-delta ($\Sigma\Delta$) converters have been implemented as discrete-time (DT) circuits, such as switched-capacitor circuits. Hence, researchers and industry experts have obtained a fair amount of insights on DT-converters and solved major technical difficulties in implementation. On the contrary, continuous-time (CT) sigma-delta converters are still fairly new to the industry.

Prior to expanding the research to CT-$\Sigma\Delta$ converter, there have been various opinions on such issues as which type is superior and what is the most suitable criterion to compare the two. Nevertheless, CT-$\Sigma\Delta$ converters provide enough advantages over DT-$\Sigma\Delta$ converters to motivate further researches. Difficulties or problems associated with CT-$\Sigma\Delta$ should be challenged such that CT-$\Sigma\Delta$ converters eventually provide competitive or even better performance measures, such as SNR or dynamic range, than other existing converters.

With this motivation, this thesis examines the excess loop delay problem of CT-$\Sigma\Delta$ converters. The primary objective is to understand sensitivity of circuit to the loop delay in system-level such that after some insights are obtained into the problem, solutions to increase the tolerance to the loop delay can be understood or proposed.

An efficient methodology to simulate excess loop delay in CT-$\Sigma\Delta$ modulators for different topologies is devised and implemented. The results of the implementation demonstrate some insights to the effects of the delay in different topologies.

Chapter 2

Technical Background

This chapter summarizes background knowledge useful in understanding CT-$\Sigma\Delta$ converters.

2.1 Operational Description of $\Sigma\Delta$ Modulator

$\Sigma\Delta$ modulator is composed of three important components as illustrated in Figure 2-1(a): loop filter, clocked quantizer and feedback DAC. The overall behavior of such modulator can be qualitatively understood through a linear analysis. The highly non-linear and noisy quantizer, however, makes such analysis difficult. As a remedy, the quantizer is modelled as a linear component and the quantization error, e, is assumed to be highly-uncorrelated or even independent of the input signal, x, as shown in Figure 2-1(b).

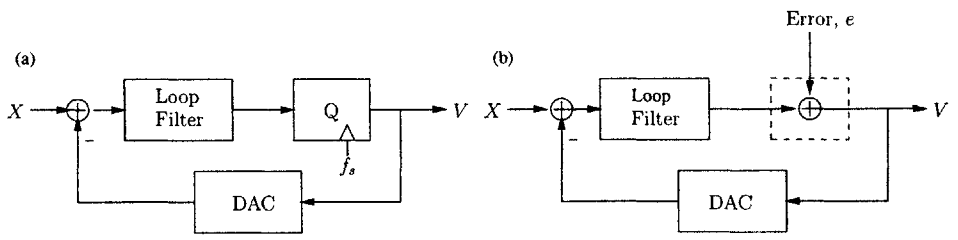


Figure 2-1: Block diagram of (a) basic structure (b) linear model of the $\Sigma\Delta$ modulator [8].

Based on the block diagram in Figure 2 (b), the input-output relationship can be written as shown in Equation 2.1.

$$\begin{aligned} V(s,z) &= \frac{H(s,z)}{1+H(s,z)} \cdot X(s,z) + \frac{1}{1+H(s,z)} \cdot E(s,z) \\ &= \text{STF(s,z)} \cdot X(s,z) + \text{NTF(s,z)} \cdot E(s,z) \end{aligned} \tag{2.1}$$

where STF and NTF are respectively signal transfer function and noise transfer function. When the $|H| \gg 1$, E is greatly attenuated and $V \approx X$. In other words, loop filter, H, shapes the quantization noise away from the frequency band of interest, while passing the input spectra almost unchanged when the gain of H is large [3].

In summary, the analog input signal is modulated into a digital word sequence whose spectrum approximates that of the analog input well in a narrow frequency range, but is noisy outside the frequency range [8].

2.2 CT-ΣΔ and DT-ΣΔ Converters

Figure 2-2 illustrates that the key difference between DT and CT-ΣΔ modulators is the sampling instants; CT-ΣΔ modulator samples signal at the quantizer while DT-ΣΔ modulator samples at the input. This difference ascribes to the major advantages and disadvantages of two systems summarized in Table 2.1.

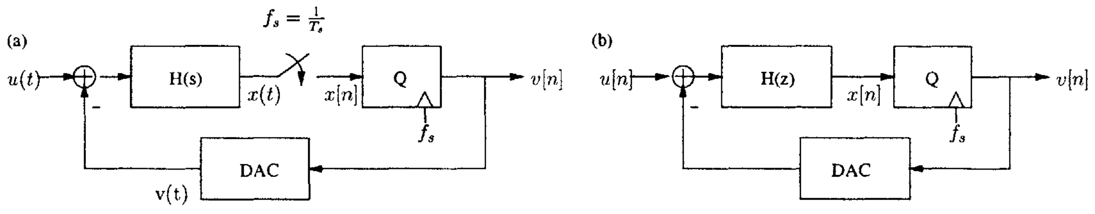


Figure 2-2: Block diagram of (a) CT-ΣΔ (b) DT-ΣΔ Modulators

To illustrate how sampling instant plays an important role, consider the effects of input noise in both CT and DT circuitry. When the bandwidth of the noise spectrum at sampling point is greater than half of the sampling frequency, noise-folding (or more commonly aliasing) occurs at the sampling instant.

To prevent such noise-folding, DT-$\Sigma\Delta$ modulator requires separate anti-aliasing filter prior to the input. In contrast, CT-$\Sigma\Delta$ modulator may not require separate anti-aliasing filter because the integrator prior to sampling can act as an implicit anti-aliasing filter. Integration of the input signal over one clock period can be considered as a convolution of the input with a rectangular pulse. In other words, the input spectrum is multiplied by a *sinc* function in frequency domain. The sinc function conveniently nulls signals at multiples of sampling frequencies, $k \cdot f_s$ and attenuates signals near $k \cdot f_s$, which otherwise would alias. The anti-aliasing property arises because the sampling happens after the integrator.

Table 2.1: Summary and comparisons of DT and CT-$\Sigma\Delta$ Converters

Type	**Discrete-Time**		**Continuous-Time**	
	Low-pass	Band-pass	Low-pass	Band-pass
Implementation	Switched-capacitor filters		Op-amp-RC, Gm-c or LC filters	
	Accumulator	Resonators	Integrators	Resonators
Advantages(+) **Drawbacks(−)**	+ Accurate Transfer Functions + High Linearity − Requires Anti-Aliasing		− Moderately Accurate Transfer Functions − Moderate Linearity + High-speed + Implicit Anti-aliasing Filter	
Non-idealities	− Mismatch in capacitor ratios implementing DACs − Clock feed-through from switches		− Excess loop delay − Clock jitter in the feedback loop − Sensitive to 1/f noise − Mismatch in current source	

2.3 DAC Pulses

In CT-$\Sigma\Delta$ modulators, excess loop delay shifts the edges of DAC pulses which in turn affects the overall signal and noise transfer function of the modulator. Thus, it is helpful to know the transfer functions of various DAC pulse types and shapes.

The three common DAC pulses: non-return-to-zero (NRZ), return-to-zero (RZ) and half-delayed returned-to-zero (HRZ) are depicted in Figure 2-3 and their respective transfer functions are stated [8].

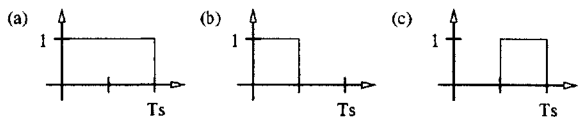


Figure 2-3: Common DAC pulse types: (a) NRZ(s) $= \frac{1-e^{sT_s}}{s}$ (b) RZ(s) $= \frac{1-e^{sT_s/2}}{s}$ (c) HRZ(s) $= e^{-sT_s/2}\frac{1-e^{-sT_s/2}}{s}$

Using superposition to incorporate delays and impulse-invariant transformation to convert to discrete domain, the effects of excess loop delay to the overall transfer function of the modulators can be studied.

2.4 State-Space Representation

Physical systems can be represented in a variety of forms, such as differential equations, transfer functions, linear graphs, and state-space descriptions. For circuit systems, transfer function representations are widely used to explore frequency-domain characteristics and stability of the systems. For multiple-input, multiple-output (MIMO) systems, however, state-space representations are more appropriate.

State-space representation describes a system in terms of states, $\mathbf{x}$, where transformations between states are summarized in matrices. A general linear continuous system with feedback and feedforward paths shown in Figure 2-4 can be described by Equation 2.2.

The relationship between inputs and internal states is summarized in matrix $\mathbf{A}$ and $\mathbf{B}$, while the relationship between the internal states and outputs is summarized in

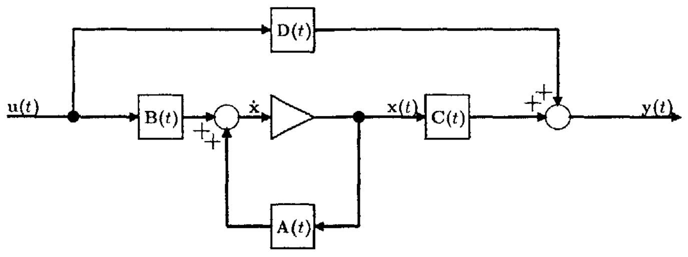


Figure 2-4: State space representation of continuous-time linear system [4].

matrix **C** and **D**. The clear advantage of matrix representation is the extendability in size to accommodate multiple inputs and outputs of the system.

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t) \tag{2.2a}$$

$$\mathbf{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t) \tag{2.2b}$$

Chapter 3

Problem Statement and Review of Literature

This chapter depicts the excess loop delay problem addressed in this thesis, justifies the importance of the problem, and presents a brief review of major ideas presented up to date to compensate for the degrading effects of excess loop delay.

3.1 Problem Statement: Excess Loop Delay

The one of major concerns of CT-$\Sigma\Delta$ modulators is excess loop delay [8]. Excess loop delay refers to the nonzero delay between the quantizer clock edge and the edge of the DAC pulse. Ideally, DAC pulse should respond immediately to the quantizer clock edge, but due to nonzero gate delays and transistor switching time, there is a finite delay in the feedback.

The timing error, excess loop delay, exists both in DT and CT-$\Sigma\Delta$ modulators. However, the errors are only problematic in CT-$\Sigma\Delta$ modulator because the timing errors are continuously accumulated at the integrator through the feedback DAC. In DT-$\Sigma\Delta$ modulators, on the other hand, clock jitter introduces noise on the sampled input, while excess loop delay is irrelevant.

The behavior of an ideal 1-bit DAC in the feedback path is normally modelled as a zeroth order hold function as shown in Equation 3.1a. The nonidealities can be modelled as a delay, d, and a first order linear system with a time constant, τ, as shown in the step response of nonideal DAC in Figure 3.1. At circuit-level, d is attributed to the reaction time between input and output while τ is attributed to finite rise and fall time of the response.

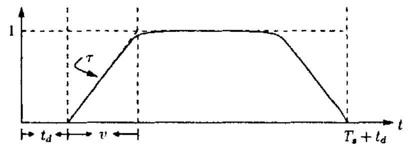


Figure 3-1: Step response of non-ideal DAC

$$DAC_{ideal} = \frac{1 - e^{T_s s}}{s} \tag{3.1a}$$

$$DAC_{nonideal} = DAC_{ideal} \frac{e^{-t_d s}}{(1 + \tau s)} \tag{3.1b}$$

This excess loop delay is problematic because the output current of DAC is continuously integrated at the CT loop filter. In specific, if not accounted into the original design, this unpredictable delay can modify the input-output relationship of the DAC as shown in Equation 3.1b, increasing the order of the feedback open loop by one [8]. Added order and feedback delay can move poles of NTF out of unity circle, causing instability. In addition, the delay degrades the performance of the modulator, such as SNR. Since zeros of NTF are fixed at the unity circle, SNR does not significantly worsen with small delays, but after a certain delay value, SNR can degrade significantly [12].

The existing remedies, such as coefficient tuning and adding an extra delay in the feedback [8], resolve the problem to some extent, but lack in insights to the nature of the problem. Answering the three questions stated in Section 3.1 will help to enhance the performance, ease the design process, and increase current understanding of CT-$\Sigma\Delta$ modulators.

Thus, this thesis focuses on the effects of the excess loop delay in CT-$\Sigma\Delta$ modulator with the following research questions in mind.

I. What are the tolerable loop delay values in various architectures of CT-$\Sigma\Delta$ modulators?

II. Why certain type or architecture of CT-$\Sigma\Delta$ modulators is more susceptible or robust to the loop delay?

III. What are the ways to make CT-$\Sigma\Delta$ modulators more tolerable to the loop delay?

3.2 Review of Compensation Methods

This section briefly summarizes the major ideas suggested up to date to compensate the effects of excess loop delay.

3.2.1 DAC Pulse Selection Approach

Excess loop delay pushes the falling edge of the DAC pulse beyond clock period, T_s, as shown in Figure 3.1, which causes the realized loop transfer function to be different from the desired function. To prevent the end of the DAC pulse from exceeding T_s, [1] first suggested using a RZ DAC pulse instead of a NRZ DAC pulse illustrated in Section 2.3. Using a RZ DAC pulse forces the output of DAC to go to zero at each integration cycle. Thus, as long as the time delay, t_d, is smaller than 0.5, the effects of excess loop delay are completely compensated. Using a RZ pulse, however, increases the power consumption and DAC jitter sensitivity, and decreases the speed of the DAC circuitry because the output level of DAC must return to zero at every integration cycle.

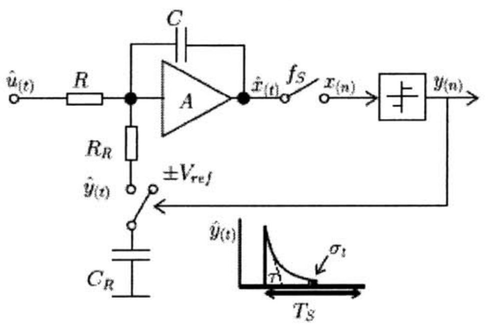
3.2.2 Coefficient Tuning Approach

Excess loop delay increases the modulator order by one, resulting in one more numerator coefficients in NTF. [7] and [8] demonstrated that tuning this extra coefficients in feedback can result in a better match between the desired NTF and the realized

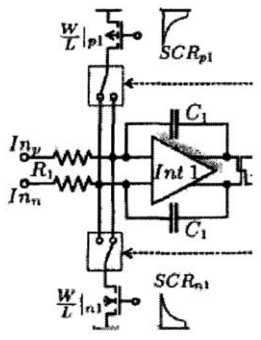
NTF after implementation. In addition, feedback coefficient tuning can alleviate other nonidealities in the DAC output, such as finite rise and fall times. Technical details regarding tuning are explained in detail in [7]. To find a match, however, t_d must be known to certain accuracy while t_d cannot be measured prior to circuit design and implementation.

3.2.3 SCR/SCR-I Feedback Techniques

[13] proposed using a sloping feedback DAC pulse as shown in Figure 3-2(a) to reduce the sensitivity of CT-$\Sigma\Delta$ modulators to DAC nonidealities. By discharging capacitor, C_R, over resistor R_R, an exponentially decaying feedback pulse with $\tau = R_R C_R$ is added to the integrator input rather than a square pulse. Thus, the charge error due to the time-delayed jitter is much smaller as shown in 3-2(a). This switched-capacitor-resistor (SCR) feedback technique does not require prior knowledge of t_d value and does not impose fast slewing of the CT integrator as the maximum feedback current is limited.



(a) With SCR Feedback (P8)



(b) With SCR-I Feedback (P3)

Figure 3-2: $\Sigma\Delta$ modulator with different feedback circuitries

[12] proposed SCR-I technique, which is a modification to SCR technique, to remove the discharging process at the integrator input. SCR-I technique implements a current mode feedback controlled by exponentially decaying control voltage as shown in Figure 3-2(b), achieving the similar affect as the SCR technique. As long as the total

current within each clock period is transferred at the end of each clock period, there is no error due to the excess loop delay in both techniques.

Chapter 4

MATLAB Simulation

This chapter presents the main frame work of this thesis. Section 4.1 discusses various methods for simulating CT-$\Sigma\Delta$ modulators, from which the hold equivalence method approach is discussed in detail in Section 4.2. Section 4.3 introduces the CT-$\Sigma\Delta$ model used in this thesis and Section 4.4 explains in detail how feedback delay is simulated in different loop topologies of CT-$\Sigma\Delta$ modulators. Results of this implementation are illustrated and analyzed in Chapter 5.

4.1 Simulating CT-$\Sigma\Delta$ Modulators

While frequency-domain simulation alone is adequate for conventional data converters, time-domain simulation is often required to derive meaningful performance metrics, such as FFT to calculate SNR, for $\Sigma\Delta$ modulators. In addition, modelling and simulating CT modulators is often more difficult than DT circuits, because CT-$\Sigma\Delta$ circuits require smaller time steps to accurately model the behavior. Many publications have suggested and compared various approaches to simulating CT-$\Sigma\Delta$ modulators, which are summarized in Table 4.1.

In specific, Cherry et all demonstrated in [5] that simulating $\Sigma\Delta$ modulator using analog tools is extremely slow and inefficient. Thus, a practical approach is to use either a behavioral or DT-equivalence model with a DT simulation tool. Many available tools

Methods	Transistor Level	Intermediate Level	Behavioral Level	DT Equivalence
Speed	Slowest	Moderate	Fast	Fastest
Accuracy	Highest	High	High	Low
Other Properties	Easy to include nonlinearity	Useful if transient analysis is not excessive	Requires deep knowledge	
Available Tools	SPICE Opal	ELDO SABER SWITCAP-2	MATLAB TOSCA ASIDES	MATLAB

Table 4.1: Comparison of different approaches to CT-$\Sigma\Delta$ simulation [5], [10], [17], [2], and [11].

listed in Table 4.1, however, are not open to public, waiting to be commercialized. The only $\Sigma\Delta$ tool available to public is Delta-Sigma toolbox [15] in MATLAB©, which provides various functions to do a high-level design and simulation of DT-$\Sigma\Delta$ modulators. To utilize this readily available tool, this thesis employs the DT-equivalence model to synthesize and simulate delay effects on CT-$\Sigma\Delta$ modulators.

There are three renowned methods of finding an equivalent DT system: numerical integration with bilinear transformation, pole-zero mapping, and hold equivalents. Due to the fundamental differences in the nature of CT and DT signals, there is no exact equivalence between two systems. Thus, finding an equivalent DT system is interpreted as finding a discrete transfer function which has approximately the same characteristics over a range of frequency as continuous transfer function [6]. All three methods described in the following subsections, except for numerical integration with forward rectangular rule, guarantees a stable discrete system.

4.1.1 Numerical Integration Approach

Numerical integration approach approximates $H(z)$ by finding a difference equation whose solution is an approximation of a differential equation derived from $H(s)$. The general procedures are summarized below:

1. Represent the given filter transfer function $H(s)$ as a differential equation.
2. Solve the differential equation in terms of time.
3. Choose how the incremental area term is approximated with a fixed time step. Three simple and common ways to approximate the area within one time step are forward rectangular rule, backward rectangular rule, and bilinear rule.
4. Substitute the time step chosen in Procedure 3 to solution of differential equation to find a difference equation.
5. Convert the difference equation to $H(z)$

Using numerical integration method is easier and simpler when systems are expressed in state-space representation. If a CT system is expressed in state-space representation as shown in Equation 4.1,

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}(t) + \mathbf{B}e(t) \qquad (4.1a) \qquad \mathbf{w}(k+1) = \mathbf{\Phi}\mathbf{w}(k) + \mathbf{\Gamma}e(k) \quad (4.2a)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}e(t) \qquad (4.1b) \qquad u(k) = \mathbf{H}\mathbf{w}(k) + \mathbf{J}e(k) \quad (4.2b)$$

then a discrete equivalent system with sampling period T can be conveniently described by Equation 4.2. $\mathbf{\Phi}$, $\mathbf{\Gamma}$, $\mathbf{H}$, and $\mathbf{J}$ can be expressed only in terms of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, $\mathbf{I}$, and T as shown in Table 4.2, where $\mathbf{I}$ is an identity matrix.

Schreier used numerical integration methods with transfer function representations in [16] to derive discrete equivalents for low-order CT-$\Sigma\Delta$ modulators. For high-order modulators with multiple feedback paths, however, required calculations become too complicated, making numerical integration not suitable for simulation of various topologies.

	Forward	Backward	Bilinear
Φ	$\mathrm{I} + \mathrm{A}T$	$(\mathrm{I} - \mathrm{A}T)^{-1}\mathrm{B}T$	$(\mathrm{I} + \frac{\mathrm{A}T}{2})(\mathrm{I} - \frac{\mathrm{A}T}{2})^{-1}$
Γ	$\mathrm{B}T$	$(\mathrm{I} - \mathrm{A}T)^{-1}$	$(\mathrm{I} - \frac{\mathrm{A}T}{2})^{-1}\mathrm{B}\sqrt{T}$
H		$\mathrm{C}(\mathrm{I} - \mathrm{A}T)^{-1}$	$\sqrt{T}\mathrm{C}(\mathrm{I} - \frac{\mathrm{A}T}{2})^{-1}$
J	D	$\mathrm{D} + \mathrm{C}(\mathrm{I} - \mathrm{A}T)^{-1}\mathrm{B}T$	$\mathrm{D} + \mathrm{C}(\mathrm{I} - \frac{\mathrm{A}T}{2})^{-1}\mathrm{B}\frac{T}{2}$

Table 4.2: Definition of Φ, Γ, H, and J

4.1.2 Pole-Zero Mapping Approach

Pole-zero mapping exploits the relationship between s and z planes, $z = e^{sT}$, where T is sampling period. General rules for mapping poles and zeros to a discrete equivalent are described below:

1. All poles, s_p, of H(s) $\longrightarrow z = e^{s_pT}$
2. All **finite** zeros, s_z, of H(s) $\longrightarrow z = e^{s_zT}$
3. Zeros of H(s) at $s = \infty \longrightarrow z = -1$
4. Match the gain such that $|H(s)|_{s=0} = |H(z)|_{z=1}$

Pole-zero mapping method is the easiest and most effective method among the three methods. Cherry used a variation of this method in [14] for synthesis of CT-$\Sigma\Delta$ modulators with bilinear transformation. This method, however, is not optimal to simulate a feedback delay, because delay affects the overall transfer function and poles and zeros must be numerically solved for each delay value. Since poles and zeros are roots of polynomials in transfer function, the relationship between delay in different topologies and changes in pole-zero values is as difficult as finding a general rule in solving high-order polynomials.

4.1.3 Hold Equivalents Approach

Unlike the two previous approaches, hold equivalents approach finds its solution in time-domain. Since a continuous system represented in Figure 4-1(a) is mapped to a discrete system with a continuous input and a discrete output, discrete equivalent system can be approximated by placing a sample and hold (S/H) at the input as shown in Figure 4-1(b). The goal is, then, to design a $H(z)$ with an input consisting of samples of $u(t)$ and an output, $\hat{y}(k)$, that approximates $y(t)$.

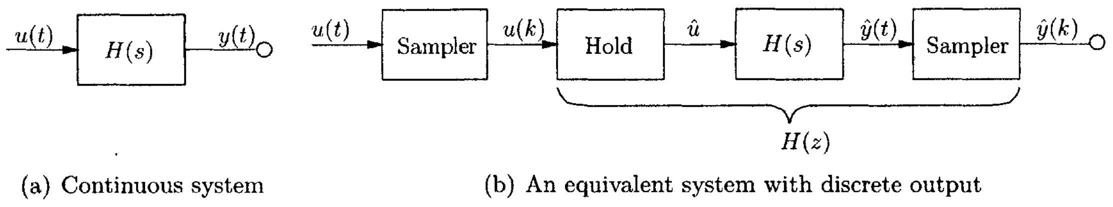


Figure 4-1: System construction for hold equivalents [6]

There are many variations in approximating the continuous signal, $\hat{u}$, from the sampled input, $u(k)$, starting from zero-order-hold (ZOH), first-order-hold (FOH), to higher-order holds. The ZOH and FOH equivalents to H(s) is shown in Equation 4.3 and 4.4 respectively.

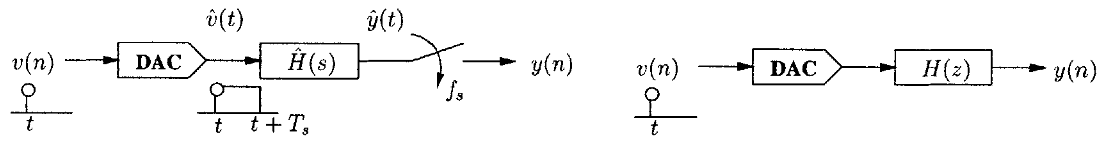
$$H_{zero}(z) = (1 - z^{-1})\mathcal{Z}\{\frac{H(s)}{s}\} \quad (4.3) \qquad H_{tri}(z) = \frac{(z-1)^2}{T_Z}\mathcal{Z}\{\frac{H(s)}{s^2}\} \quad (4.4)$$

Hold equivalent concept has been proven to be very useful in transforming between CT and DT for $\Sigma\Delta$ modulators due to the presence of quantizer in the loop. Schreier and Cherry both made use of this concept to simulate CT-$\Sigma\Delta$ modulators using DT simulation tools in [16] and [14] respectively.

4.2 Hold Equivalent Approach for CT-$\Sigma\Delta$ Modulators with Delays.

The goal of simulation is to explore excess loop delay effects on different loop topologies of CT-$\Sigma\Delta$ modulators through a DT simulation tool. Thus, the chosen method to find a DT equivalent should support easy manipulation of loop parameters of various loop topologies. While numerical integration approach becomes too complex in calculation and thus is impractical to simulate high-order modulators, pole-zero mapping approach is only effective if poles and zeros of continuous system are known. In contrast, hold equivalent focuses on S/H which is conveniently modelled by the quantizer and allows various topologies to be realized if expressed in state-space representations. Therefore, this thesis employs hold equivalent approach to simulate CT-$\Sigma\Delta$ modulators with delays.

Consider CT and DT-$\Sigma\Delta$ closed loop system in Figure 2-2. Sample and hold concept in Section 4.1.3 can be applied around the quantizer input and output, where extrapolating between samples is now determined by a DAC pulse. Cutting the loops around the quantizer results in open loop system illustrated in Figure 4-2, where input is nulled such that impulse response of only the linear portion can be considered for equivalence [14].

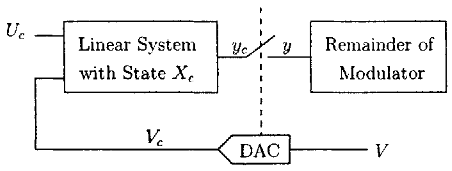


(a) CT (b) DT

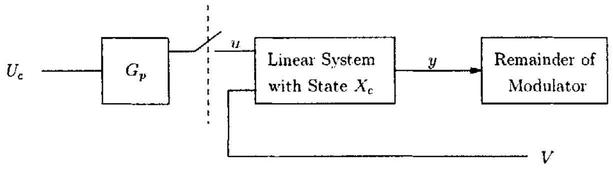
Figure 4-2: Open loop $\Sigma\Delta$ modulators

The two systems are equivalent if outputs of quantizer are equivalent at the sampling instance, which requires inputs to quantizer to be the same at sampling instants. For inputs to be the same, $y(n) = \hat{y}(t)|_{t=nT_s}$, impulse response of the open-loop systems must be the same at sampling times. Thus, by comparing the impulse responses, equivalent NTF of CT and DT-$\Sigma\Delta$ modulators can be found.

If the input is to be included, the linear open-loop system becomes multiple input system as shown in Figure 4-3(a). In addition, CT input in the equivalent DT modulator is pre-filtered to provide DT equivalent input as shown in Figure 4-3(b). In addition,



(a) A modulator with CT front end.



(b) An equivalent DT modulator with a prefilter on the input

Figure 4-3: Block diagram of discrete equivalents of CT-ΣΔ modulator.

for multiple I/O systems, it is much easier to work with state-space representations, hence the mapping from H(s) to state-space system with state X_c.

4.3 CT-ΣΔ Model

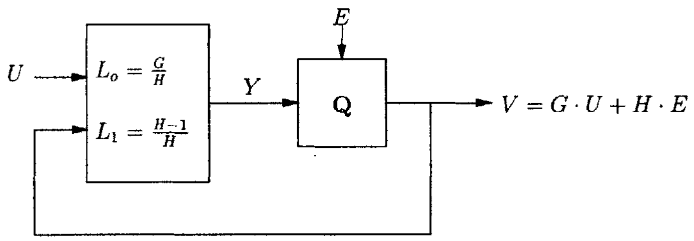


Figure 4-4: General block diagram of a single-quantizer DT-ΣΔ modulator [16].

All ΣΔ modulators with single-quantizer loops can be described by a linear model illustrated in Figure 4-4. This model is particularly useful for finding the DT equivalent of a CT modulator, because input and feedback input pass through two loop filters, L_0 and L_1, independently. These loop filters can also be expressed as functions of loop parameters for different topologies.

4.4 Methodology

The goal is to simulate feedback delays in different loop topologies of CT-ΣΔ modulators. The flow chart of the methodology employed in this thesis for such simulation is illustrated in Figure 4-5.

The first step is to synthesize an ideal modulator. Since $\Sigma\Delta$ modulators can be described by NTF and STF as explained in Section 2.2, NTF of a low-pass modulator with a specific OSR and order is synthesized through synthesizeNTF function provided in Delta-Sigma toolbox. This function returns a DT NTF for a stable $\Sigma\Delta$ modulator with maximum out-of-band gain of H_inf.

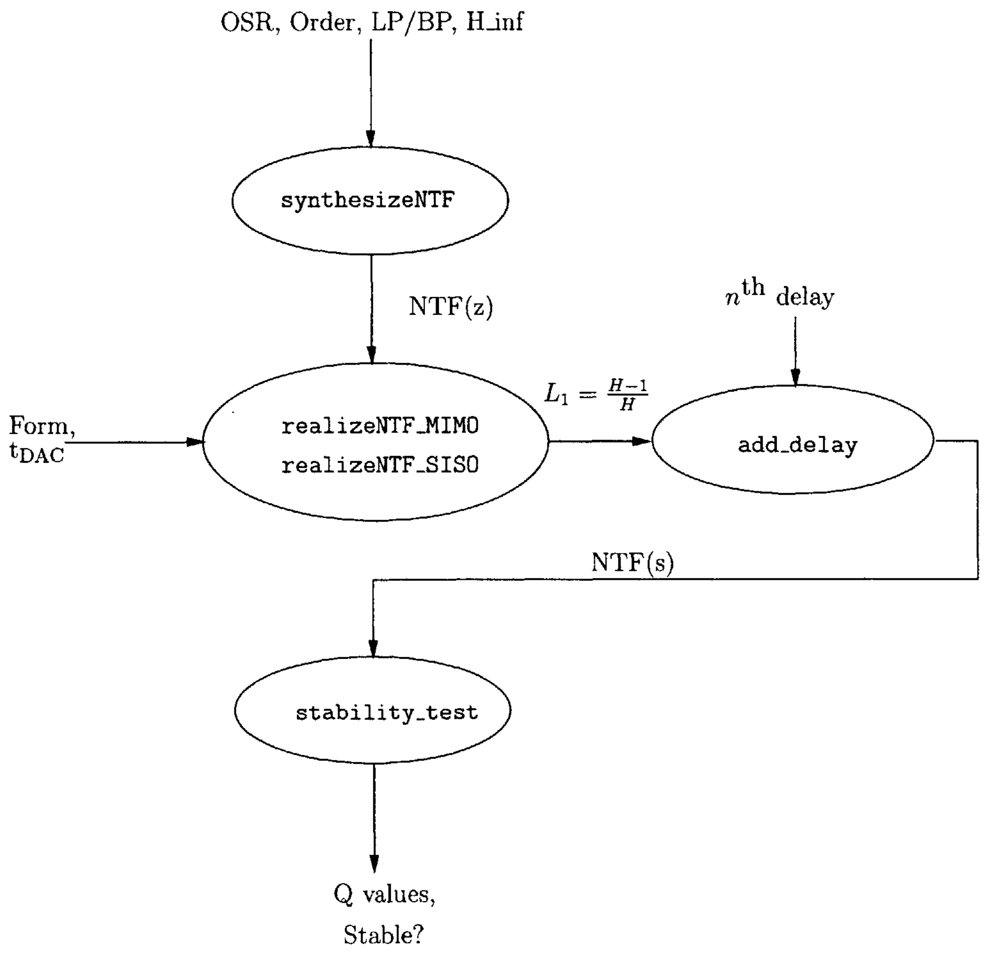


Figure 4-5: Flow diagram of overall design to simulate delay in CT-$\Sigma\Delta$ modulator

Then, the second stage is to realize this modulator into different loop topologies, where gain coefficients of feedback and feedforward paths are computed. RealizeNTF_MIMO and realizeNTF_SISO take DT-NTF, a topology, and a timing for a DAC pulse as inputs, and return a CT loop transfer function, $L_1(s)$. $L_1(s)$ is returned instead of a NTF because $L_1(s)$ describes relationship between output of the modulator and the feedback portion of input, so that delays can be added to either multiple feedback paths (realizeNTF_MIMO) or a single overall feedback path (realizeNTF_SISO). In addition, $L_1(s)$ is expressed in terms of loop coefficients, reflecting the specific topology of the modulator.

Various delay values are, then, applied to feedback inputs of $L_1(s)$ in add_delay. add_delay also rewires the system such that the output returns a NTF instead of $L_1(s)$ so that the new system with delays can be compared to the ideal system.

The new NTF with excess loop delay is, then, tested in the last stage for its stability by calculating Q values. The experimental Q value for which the system goes unstable is around 5. Power spectral density of the NTF with a half-scale sine-wave input is also plotted to demonstrate the feasibility of this methodology to measure degradation in performance.

4.5 Implementation

This section explains the implementation of new functions which do not exist in Delta-Sigma toolbox: realizeNTF_MIMO/realizeNTF_SISO, add_delay, and test_stability. Without loss of generality, sampling period, T_s is set to 1 for all implementations.

4.5.1 realizeNTF_MIMO/realizeNTF_SISO

Implementation of RealizeNTF_MIMO and realizeNTF_SISO is first explained for a general modulator topology. Then, an MIMO implementation is described in detail for distributed feedback topology. For other topologies implemented, only block diagrams, loop transfer functions, and state-space representations are stated.

First, impulse response of the DT NTF is computed with an ideal clock pulse. Then, state-space matrices, **A**, **B**, **C**, and **D** are defined for each topology with gain coefficients of 1. Then, pulse response is computed for this continuous system by subtracting impulse response at the end time of DAC pulse, t_2, from the one at the beginning of DAC pulse, t_1. By comparing this pulse response to the impulse response computed for DT NTF, coefficients for feedback and feedforward paths are determined. Justification for this methods is discussed in [16].

If there is an extra state due to t_2 extending beyond one DAC pulse, an extra state or direct feedforward is added to the system through matrix **D** manipulation. Lastly, coefficients for matrices **B** and **D** are scaled for input such that the STF magnitude at zero frequency is 1.

Distributed Feedback

Distributed feedback topology is used to stabilize higher-order modulators, where fraction of the output is applied at each integrate state as shown in Figure 4-6. Feedback compensation adds zeros to NTF as shown in Equation 4.5c.

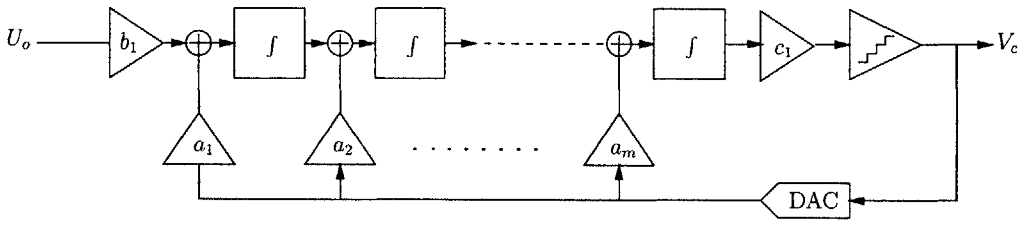


Figure 4-6: Block diagram of CT-$\Sigma\Delta$ modulator with weighted distributed feedbacks

$$L_0(s) = \frac{b_1}{s^m} \tag{4.5a}$$

$$L_1(z) = \frac{a_1}{(z-1)^m} + \frac{a_2}{(z-1)^{m-1}} + \cdots + \frac{a_m}{(z-1)} \tag{4.5b}$$

$$NTF(z) = \frac{1}{1-L_1(z)} = \frac{1}{1-\sum_{i=1}^{m}\frac{a_i}{(z-1)^{m+1-i}}} \tag{4.5c}$$

$$STF = L_oNTF = \frac{b_1}{s^m - \sum_{i=1}^{m}\frac{a_i s^m}{(z-1)^{m+1-i}}} \tag{4.5d}$$

Assuming a linear system, the feedback paths can be opened up as shown in Figure 4-7, creating a system where feedback paths are considered as inputs. With this implementation, delays can be added to a single feedback path. The general formulation for state-space representations shown in Section 2.3 is applied to the MIMO $L_1(s)$ system in Figure 4-7 to result in Equation 4.6.

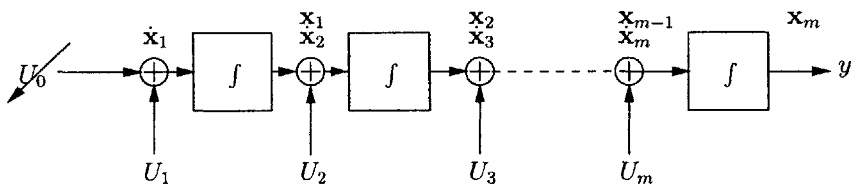


Figure 4-7: Block diagram of L_1 transfer function with weighted distributed feedbacks topology with multiple inputs.

$$
\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \vdots \\ \dot{x}_m \end{bmatrix} = \begin{bmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & & \ddots & & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & \cdots & \cdots & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ \vdots \\ U_m \end{bmatrix}
$$

$$
\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} 0 & \cdots & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} 0 & 0 & \cdots & \cdots & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_m \end{bmatrix} \tag{4.6}
$$

Distributed Feedback with Local Feedback

Resonators, a local feedback around two integrators, are preferred for higher-order modulators to spread the gain more evenly in signal band. Adding a small negative-feedback term moves the open-loop poles away from dc along the unit circle [R2], where these poles correspond to closed-loop zeros of NTF. Thus, the frequencies, where there is an infinite noise attenuation, are shifted away from DC to finite positive frequencies.

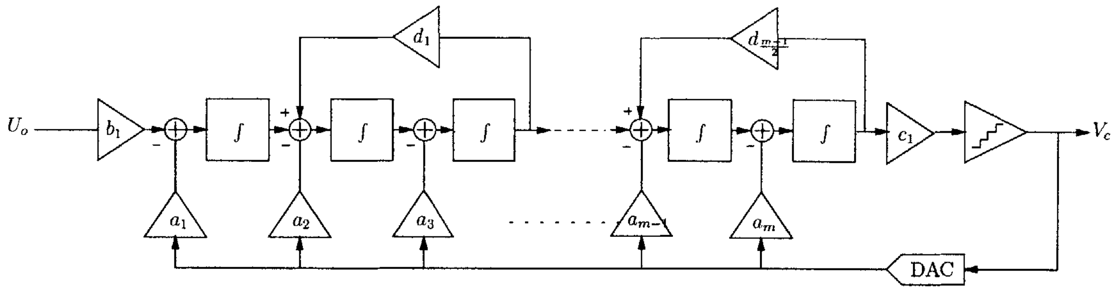


Figure 4-8: Block diagram of CT-ΣΔ modulator with feedback compensation with local feedback

To realize a hybrid system with distributed feedback or distributed feedforward with local feedback, the linearity of the model in 4.3 is fully exploited. Loop coefficients for distributed feedback or feedforward paths are first computed as explained in the previous section. Then, the local feedback coefficients are found by equating the desired NTF and the NTF realized into distributed feedback or feedforward topology. Thus, for topologies with local feedback, only the block diagrams are stated in the following sections.

Distributed Feedforward

$$L_0(s) = \frac{a_1}{s} + \frac{a_2}{s^2} + \cdots + \frac{a_m}{s^m} = \sum_{i=1}^{m} \frac{a_i}{s^i} \tag{4.7a}$$

$$L_1(z) = -\sum_{i=1}^{m} \frac{a_i}{(z-1)^i} \tag{4.7b}$$

$$NTF(z) = \frac{1}{1 - L_1(z)} = \frac{1}{1 + \sum_{i=1}^{m} \frac{a_i}{(z-1)^i}} \tag{4.7c}$$

$$STF = L_0 NTF = \frac{\sum_{i=1}^{m} \frac{a_i}{s^i}}{1 + \sum_{i=1}^{m} \frac{a_i}{(z-1)^i}}$$

(4.7d)

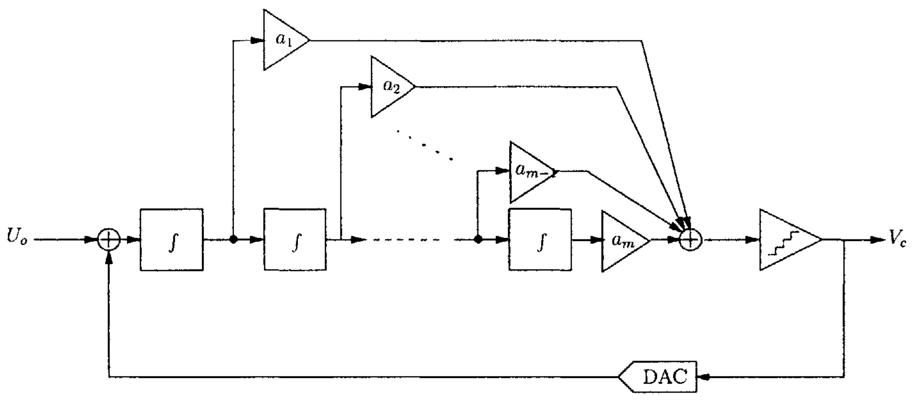


Figure 4-9: Block diagram of CT-$\Sigma\Delta$ modulator with distributed feedforward

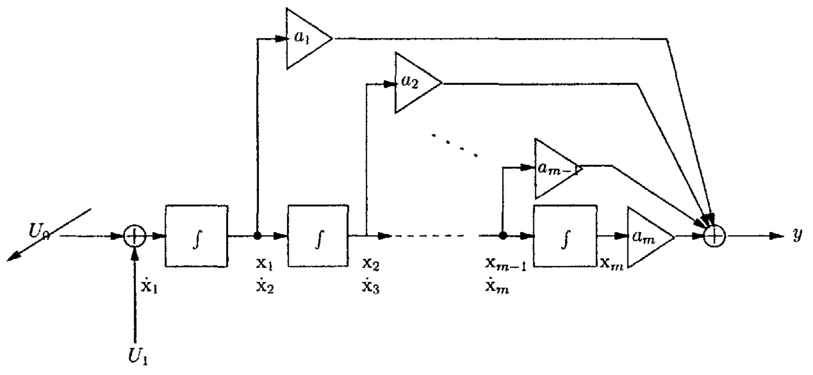


Figure 4-10: Block diagram of L_1 transfer function with weighted distributed feed-forward summation topology

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \vdots \\ \dot{x}_m \end{bmatrix} = \begin{bmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ 1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & & \ddots & & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ \vdots \\ U_m \end{bmatrix}$$

$$\begin{bmatrix} y \end{bmatrix} = \begin{bmatrix} C_{c1} & C_{c2} & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} U_1 \end{bmatrix} \tag{4.8}$$

Distributed Feedforward with Local Feedback

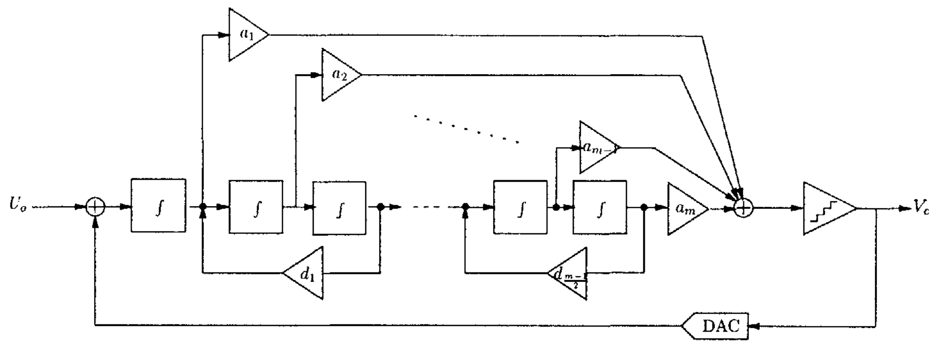


Figure 4-11: Block diagram of CT-ΣΔ modulator with feedforward compensation with local feedback

Distributed Feedback with Distributed Feedforward Inputs

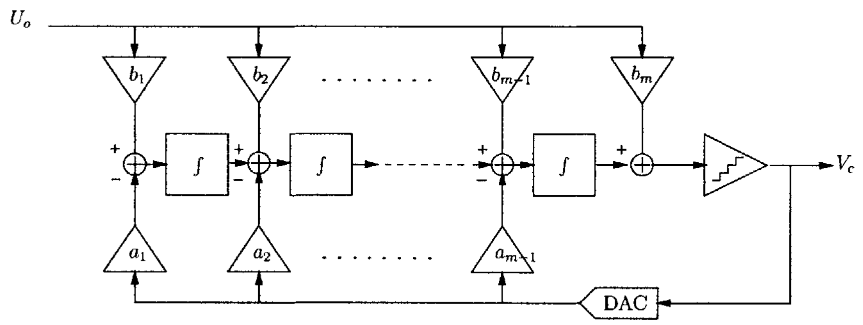


Figure 4-12: Block diagram of CT-ΣΔ modulator with weighted distributed input and distributed feedback

$$L_0(s) = \frac{b_1}{s^m} + \frac{b_2}{s^{m-1}} + \frac{b_3}{s^{m-2}} + \cdots + \frac{b_m}{s^1} \tag{4.9a}$$

$$-L_1(z) = \frac{a_1}{(z-1)^m} + \frac{a_2}{(z-1)^{m-1}} + \frac{a_3}{(z-1)^{m-2}} + \cdots\cdots \tag{4.9b}$$

$$NTF(z) = \frac{1}{1-L_1(z)} = \frac{1}{1-\sum_{i=1}^{m}\frac{a_i}{(z-1)^{m+1-i}}} \tag{4.9c}$$

$$STF(s) = \frac{\sum_{i=1}^{m}\frac{b_i}{s^{m+1-i}}}{1-\sum_{i=1}^{m}\frac{a_i}{(z-1)^{m+1-i}}} \tag{4.9d}$$

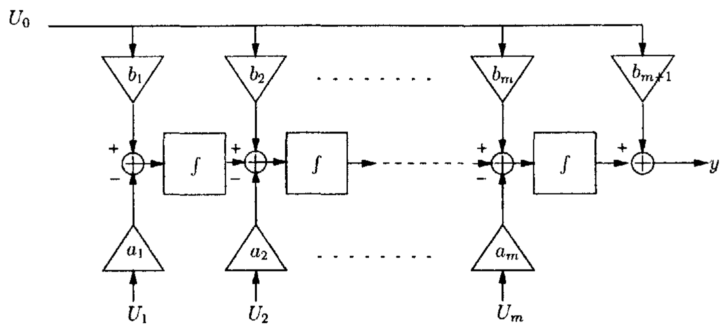


Figure 4-13: Block diagram of L_1 transfer function with weighted distributed input and distributed feedback

4.5.2 add_delay

First, various delay values, $\Delta t \in [0, 1]$, are applied to a set of feedback paths by setting the InputDelay parameter available in MATLAB ©. The resulting $L_1(s)$ with delay is converted back to DT using zero-hold equivalent method available via c2d function. Using the relationship expressed in Equation 4.10 between L_1 and NTF, $L_1(z)$ is converted back to $NTF(z)$ such that characteristics of NTF with delays can be compared to that of ideal NTF.

$$L_1 = \frac{H-1}{H} \iff H = \frac{1}{1-L_1} \tag{4.10}$$

4.5.3 stability_test

Quality factor (Q) is a measure of sensitivity of pole locations to perturbation. Q value for each system is found by observing phase and magnitude of each pole and choosing the largest value as stated in Equation 4.11.

$$Q = \max \left| \frac{\angle p}{1-|p|} \right| \quad \text{for} \quad \forall p \in \text{Poles of NTF(z)} \tag{4.11}$$

Power spectral density (PSD) of the NTF with excess loop delay is found for a half-scale sine-wave input, using simulateDSM function available in Delta-Sigma toolbox.

Chapter 5

Simulation Results and Discussion

This chapter validates the design methodology introduced in Chapter 4 in Section 5.1. Then, simulation results and discussion are organized by topology in the subsequent sections.

5.1 Validation

The methodology and implementation is validated by comparing output spectrum of DT NTF to that of NTF realized in different topologies with and without delay. For the comparison without any excess loop delay, the output spectrum of NTF realized into different loop topologies should be the same to that of DT NTF.

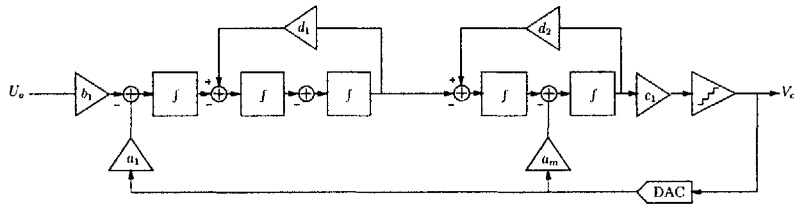


Figure 5-1: 5th order hybrid topology use for verification

To verify, a fifth-order modulator with OSR of 32 is synthesized and realized into three different topologies, distributed feedback (FB), distributed feedforward (FF), and a hybrid topology with local feedback (HB), as shown in Figure 5-1. The resulting output spectrums are illustrated in Figure 5-3. For all topologies without delay, the spectrum is the same as the ideal DT NTF shown in Figure 5-2. Figure 5-3 demonstrates that when delays are added to overall feedback paths with time increment of 0.1, the modulator for all topologies goes unstable for $\triangle t > 0.3$ as shown in Figure 5-3.

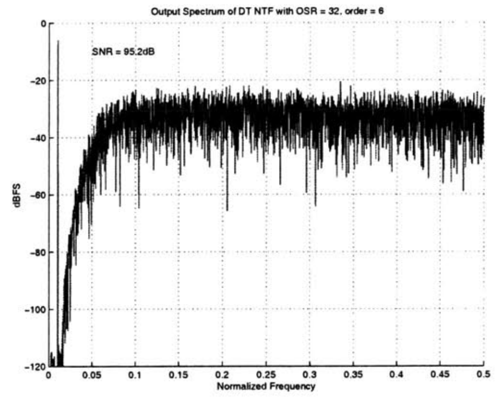


Figure 5-2: Output spectrums of ideal DT NTF

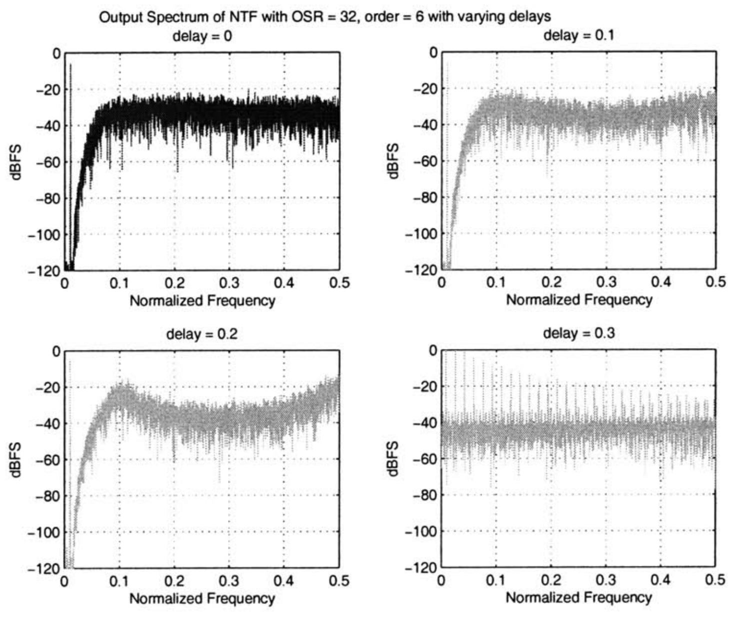


Figure 5-3: Output spectrums of DT NTF with that realized into three different topologies, FF, FB, and HB, with excess loop delay.

5.2 Distributed Feedback

Distributed feedback (FB) topology is often used for high-order stabilization. The disadvantage of such topology is that the outputs of the integrators contain both filtered quantization noise and low-frequency component equal to the input signals [3]. To accommodate the signal component, the gain of the input integrators are much lower than that in other topologies. Thus, noise and distortion of the backend integrators are not suppressed by a high gain of previous integrator stages.

To test the sensitivity of delay in FB topology, various order modulators with OSR of 32 are synthesized and delays are applied to a single feedback path to each modulator. The results are illustrated in Figure 5-4, where k_i corresponds to a_i path in Figure 4-6.

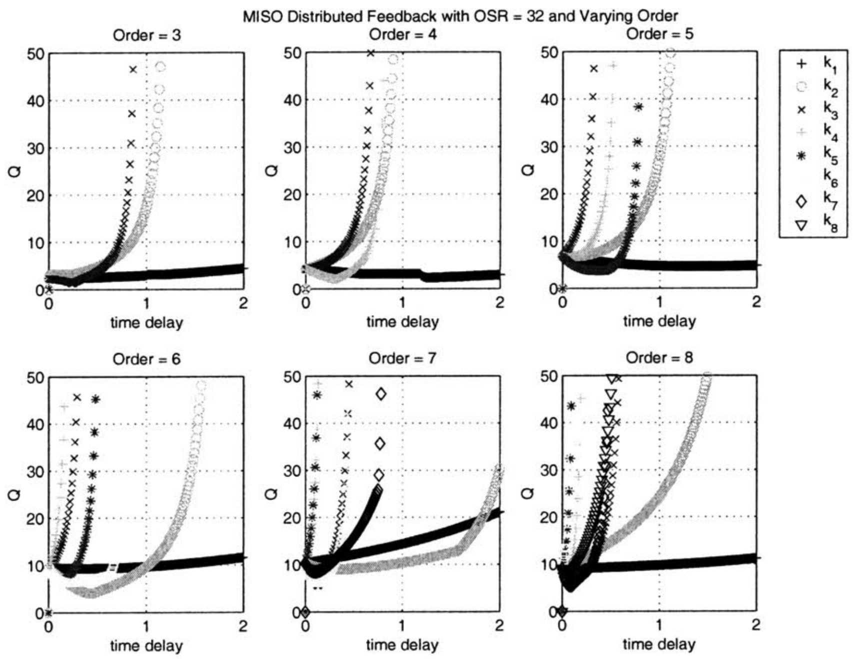


Figure 5-4: Sensitivity of a FB topology of varying order modulators with OSR = 32 to excess loop delay

The first two feedback paths near the input are the least sensitive to the delays, while the middle and last paths near the output respond more to even smaller delay values.

These results are reasonable since the effects of the delay are suppressed more when it passes more numbers of integrators.

To confirm these results, a fifth-order modulator with OSR of 32 is synthesized, where delay of 0.5 is added to the first feedback path (near input) and to the last feedback path (near output). The PSD of both cases are plotted in Figure 5-5. PSD of a modulator with delay added at the first feedback path shows better performance. The SNR of the latter is worse by 1.9 and the peak is higher by 5db compared to that of the former.

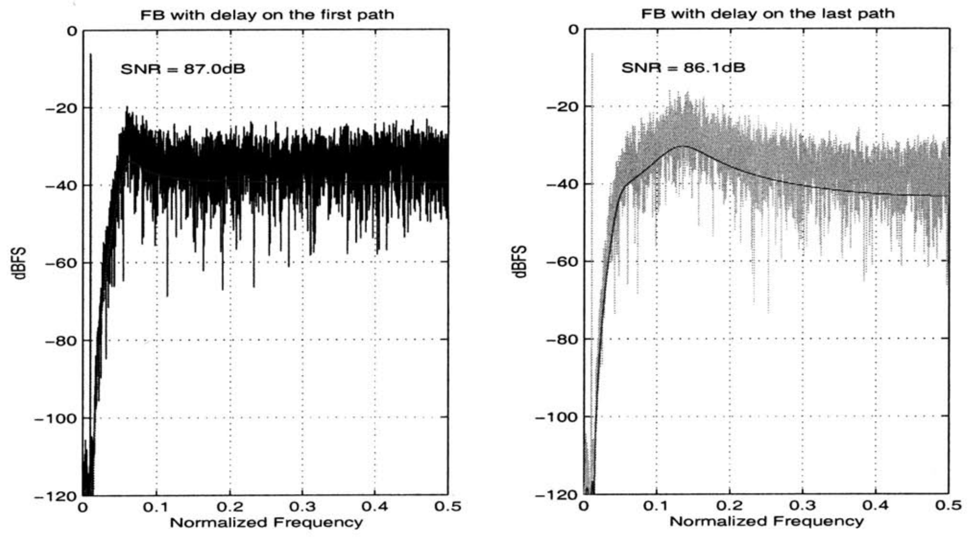


Figure 5-5: PSD of a fifth-order modulator with OSR = 32 with excess loop delay of 0.5

5.3 Distributed Feedforward

Distributed feedforward (FF) topology compensates for the high-order stabilization similar to FB topology. The major difference, however, is that adding a feedforward path introduces a zero to the STF. This creates peaking at a certain frequency, reducing the maximum stable input level at that frequency due to the gain of the peaking [3].

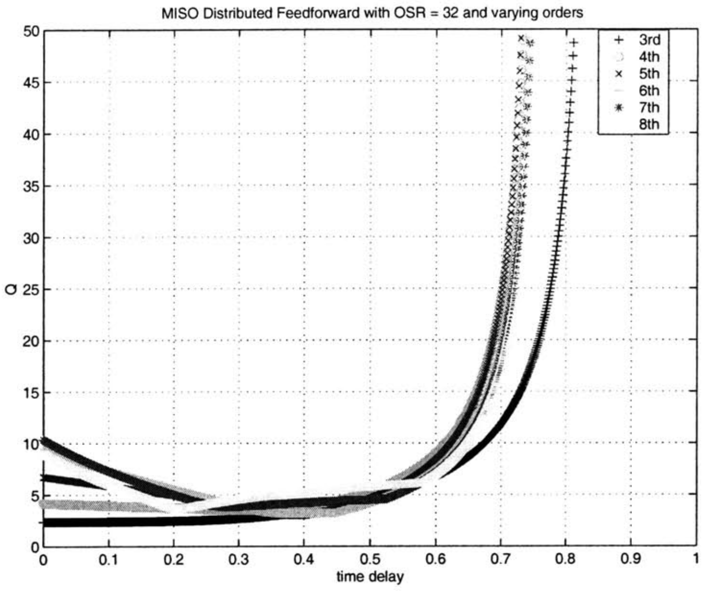


Figure 5-6: Sensitivity of a FF topology of varying order modulators with $OSR = 32$ to excess loop delay

Since modulators become unstable for quality factor greater than $5 \sim 10$, low-order modulators, such as 3rd and 4th, can endure an excess loop delay of 0.4, while high-order modulators tend to be unstable even with a slight excess loop delay near zero. This stability issue is generally solved by using local feedbacks around two integrators, which will be discussed in detail in Section 5.4.

5.4 Distributed Feedforward with Local Feedback

Local feedback around two integrator stages are often used when designing $CT - \Sigma\Delta$ modulators. As mentioned briefly in Section 4.5.1, this resonator with transfer function shown in Equation 5.1 shifts the DC gain of the integrators to a finite positive frequency [3].

$$H(s) = \frac{w_u s}{s^2 + w_u^2} \tag{5.1}$$

Thus, FF topology with local feedback should be less sensitive to the excess loop delay compared to that without local feedback.

5.5 Distributed Feedback with Distributed Feedforward Inputs

Distributed feedback with distributed feedforward inputs (FBFFI) topology allows a certain degree of independence in specifying the NTF and STF [14]. In specific, zeros of STF(s) stated in Equation 4.9a can be placed to cancel some of the poles to allow slower roll-off at STF. This topology, however, has the same NTF and $L_1(s)$ as the FB topology and thus sensitivity to excess loop delay should be similar to the FB topology.

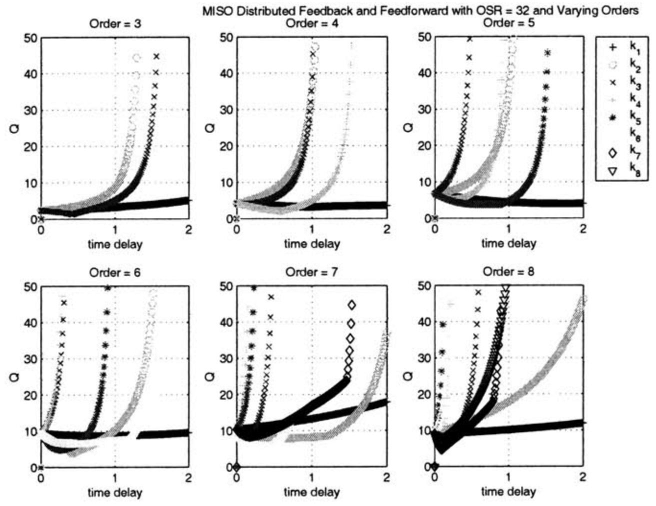


Figure 5-7: Sensitivity of a FBFFI topology of varying order modulators with OSR = 32 to excess loop delay

Chapter 6

Conclusion

Excess loop delay effects are more severe for feedback path near the output and thus possible solution to compensate for the delay effect is to apply SCR/SCR-I feedback techniques introduced in Section 3.2.3 to the last feedback path rather than the first one.

This research work also demonstrated a working methodology to simulate excess loop delay for CT-$\Sigma\Delta$ modulators in different loop topologies. Unlike many $\Sigma\Delta$ tools in academia which are not available to the public, this methodology is implemented using a Delta-Sigma toolbox which is available to public through MATLAB ©.

Appendix A

RealizeNTF_MIMO.m

```
function [L1_c, n_extra, x]= realizeNTF_MIMO(ntf, form, tdac)

% A variation of realizeNTF_ct for lowpass only:
%Assuming perfect clock pulses -- !!!
%   Input:
%       NTF in DT
%       tdac = [rising_edge falling_edge]
%   Output: A matrix containing transfer functions of
%               - MISO (Feedback Form)
%               - SIMO (Feedforward Form)
%           such that varying delays can be applied to different feedbacks.
%
% Created 09/13/2004
% Strong Reference to William Yang's excess_loop_delay.m

% Input Sanity Check:
% Extract appropriate information about NTF

order = length(ntf.z{:}); f0 = 0;
```

```
% Sample NTF impulse response: ---- To scale coefficients later ...

n_imp = ceil(2*order + max(tdac) + 1);
y = impL1(ntf, n_imp); %Impulse response of comparator for NTF ...

% Compute the time step for CT System
t1 = tdac(1) ; t2 = tdac(2);
[n1 d1] = rat(t1-floor(t1)); %delay from clk_edge to DAC pulse ...
[n2 d2] = rat(t2-floor(t2));
dt = 1/lcm(d1,d2); % maximum time scale to divide to hit both edges ...

sample_points = 1:1/dt:(n_imp+1)/dt; %time scale ...

% Number of direct feedback and extra states ...
% Pulses greater than one pulse length are considered
% as a direct feedforward
% adding extra states to the system ...

%FROM R.Schreier's paper -->
%p328 explains how comparator sees only sampled instance

n_direct = ceil(t2) - 1;
if ceil(t2 - floor(t1)) > 1 %Pulse is larger than one time pulse....
    n_extra = n_direct - 1;
else
    n_extra = n_direct; % Pulse is less than one time pulse....
end

fprintf('extra =%d, direct = %d \n', n_extra, n_direct);
```

```
% Define SS parameters for MISO or SIMO...
%% Calculate the step response of each coefficient ...
%%  1) Define initial topological coefficients....
%%      ys : matrix containing step responses of inputs thruogh
%%          the basic backbone.
%%      i.e. ys = [step(u1) step(u2) step(u3) ...]

% resonator feedbacks.
A = diag(ones(order-1,1), -1); i=1; while (i<=order)
    if abs(angle(ntf.z{:}(i))) < 1e-6  %integrator
        i = i + 1;
    elseif abs(angle(ntf.z{:}(i)) + angle(ntf.z{:}(i+1))) <1e-6
        A(i, i+1) = -angle(ntf.z{:}(i)).^2;
        i = i + 2;
    else
    error('non-DC ntf zeros must be arranged as complex conjugate pairs');
    end
end

switch form
case 'FB' %MISO
  %  A = diag(ones(order-1,1), -1);
    C = [zeros(1, order-1), 1];
    B = eye(order);
    D = zeros(1, order);
    ys = squeeze(step(ss(A, B, C, D), 0:dt:n_imp-t1+dt/2));
case 'FF' %SIMO
  %  order_o = order; %original order
   % order = order+n_extra; %order increased due to extra states...
  % A = diag(ones(order-1,1), -1);
    C = eye(order);
    B = [-1; zeros(order-1,1)];
    D = zeros(order,1);    % not include direct feedback.
    ys = squeeze(step(ss(A, B, C, D),0:dt:n_imp-t1+dt/2));

case 'FBFF'
% first calculate the step response of all the feedback paths
% with feedforward structure in, but wihout Multiple output???

    B = eye(order);
    C = ones(1,order);
    D = zeros(1,order);
    ys = squeeze(step(ss(A,B,C,D), 0:dt:n_imp-t1+dt/2));

otherwise
    error('%s error. "%s" is not supported', mfilename, form);
end

%% 2) Calculate pulse reponse by subtraction ....
%%     a) add t1/dt or t2/dt amount of zeros to represent edge->t1 and
%%        edge->t2 responses.
%%     b) y1, y2 = ys values at samples points...
%%     c) pulse response = step response(y2) - step response(y1)

y_1 = [zeros(round(t1/dt), order); ys]; y_1 = y_1(sample_points,
:); y_2 = [zeros(round(t2/dt), order); ys]; y_2 =
y_2(sample_points, :); yy = y_1 - y_2;

%% 3) Endow yy with n_extra+1 extra impulses
```

```
if n_extra ~= 0
    y_right = padb([zeros(1, n_extra); eye(n_extra)], n_imp+1);
    yy = [yy(:,1:end) y_right(:, end:-1:1)];
else
    yy = [yy(:,1:end)];
end

%% yy -> Pulse response, y -> Impulse response ...............
x = yy\y; if norm(yy*x-y)>1e-4
    warning('Pulse response fit is poor.');
end

%% D value:
switch form case {'FB', 'FBFF'}
    %%%%Check redundancy!!! of D with previous switch statement...
    Dc1 = [0];  %---for u_o
    if (n_direct)
        Dc2 = [zeros(1, order) x(order+1:end)'];
    else
        Dc2 = [zeros(1, order)];
    end
case 'FF'
   % Dc1 = zeros(order,1);
    %%D = zeros(order,2+n_extra);
    %%D(1, 3:end) = (x(order+1:end));
    Dc1 = [0];
    if (n_direct)
        Dc2 = [0 x(order+1:end)'];
    else
```

```
        Dc2 = [0];
    end

otherwise
    error('%s error. "%s" is not supported', mfilename, form);
end
D = [Dc1 Dc2];

%% A and B Values
switch form case {'FB','FBFF'}
    Bc2 = [diag(x(1:order)) zeros(order, size(x(order+1:end), 1))];
case 'FF'
    Bc2 = [[-1; zeros(order-1,1)] zeros(order, n_extra)];
    C = x(1:order)';
        %Bc2 = [[-1; zeros(order-1,1)] zeros(order, n_extra)];
    %Cc = diag(x(1:order));
otherwise
    error('%s error. "%s" is not supported', mfilename, form);
end

%% Scale Bc1 for unity STF magnitude at f0
Bc1 = [1; zeros(order-1,1)];
L0c = zpk(ss(A,Bc1,C,Dc1));  %--- for uo
G0 = abs(evalTFP(L0c,ntf,f0));
B = [Bc1/G0 Bc2];

% remove elements less than lmin.
lmin = 1e-9; A=A.*(abs(A)>lmin); B=B.*(abs(B)>lmin);
C=C.*(abs(C)>lmin); D=D.*(abs(D)>lmin); L1_c = ss(A, B, C, D);
```

Appendix B

RealizeNTF_SISO.m

```
function [L1_c, n_extra, x]= realizeNTF_SISO(ntf, form, tdac)

% A variation of realizeNTF_ct for lowpass only:
% Assuming perfect clock pulses -- !!!
%   Input:
%       NTF in DT
%       tdac = [rising_edge falling_edge]
%   Output: A matrix containing transfer functions of
%           such that varying delays can be applied
%           to different feedbacks.
% FROM WILLIAM_YANG's EXCESS_LOOP_DELAY.m

% Input Sanity Check: Extract appropriate information about NTF

order = length(ntf.z{:}); f0 = 0;
% Sample NTF impulse response: ---- To scale coefficients later ...

n_imp = ceil(2*order + max(tdac) + 1);
y = impL1(ntf, n_imp); %Impulse response of comparator for NTF ...
```

```
% Compute the time step for CT System
t1 = tdac(1) ; t2 = tdac(2);
[n1 d1] = rat(t1-floor(t1)); %delay from clk_edge to DAC pulse ...
[n2 d2] = rat(t2-floor(t2));
dt = 1/lcm(d1,d2); % maximum time scale to divide to hit both edges ...

sample_points = 1:1/dt:(n_imp+1)/dt; %time scale ...

% Number of direct feedback and extra states ...
% Pulses greater than one pulse length are considered
% as a direct feedforward
% adding extra states to the system ...
% FROM R.Schreier's paper
%--> p328 explains how comparator sees only sampled instance.

n_direct = ceil(t2) - 1;
if ceil(t2 - floor(t1)) > 1 %Pulse is larger than one time pulse....
    n_extra = n_direct - 1;
else
    n_extra = n_direct; % Pulse is less than one time pulse....
end

fprintf('extra =%d, direct = %d \n', n_extra, n_direct);

% Define SS parameters for MISO or SIMO...

%% Calculate the step response of each coefficient ...
%%  1) Define initial topological coefficients....
%%      ys : matrix containing step responses of inputs
%%          thruogh the basic backbone.
%%      i.e. ys = [step(u1) step(u2) step(u3) ...]

% resonator feedbacks.
A = diag(ones(order-1,1), -1); i=1; while (i<=order)
    if abs(angle(ntf.z{:}(i))) < 1e-6  %integrator
        i = i + 1;
    elseif abs(angle(ntf.z{:}(i)) + angle(ntf.z{:}(i+1))) <1e-6
        A(i, i+1) = -angle(ntf.z{:}(i)).^2;
        i = i + 2;
    else
    error('non-DC ntf zeros must be arranged as complex conjugate pairs');
    end
end

switch form
case 'FB' %MISO
  %  A = diag(ones(order-1,1), -1);
    C = [zeros(1, order-1), 1];
    B = eye(order);
    D = zeros(1, order);
    ys = squeeze(step(ss(A, B, C, D), 0:dt:n_imp-t1+dt/2));
case 'FF' %SIMO
    C = eye(order);
    B = [-1; zeros(order-1,1)];
    D = zeros(order,1);    % not include direct feedback.
    ys = squeeze(step(ss(A, B, C, D),0:dt:n_imp-t1+dt/2));

case 'FBFF'
% first calculate the step response of all the feedback paths
% with feedforward structure in, but wihout Multiple output???
```

```
        B = eye(order);
        C = ones(1,order);
        D = zeros(1,order);
        ys = squeeze(step(ss(A,B,C,D), 0:dt:n_imp-t1+dt/2));
    case 'HB'
        if (order < 3)
            error('The hybrid structure requires loop order >=3.');
        end
        % first calculate the step response of the two feedback paths
        C = [zeros(1,order-1) 1];
        B = zeros(order,2); B(1) = 1; B(end) = 1;
        D = zeros(1,2);     % not include direct feedback.
        ys(:,1:2) = squeeze(step(ss(A, B, C, D),0:dt:n_imp-t1+dt/2));
        % Calculate the step response of the internal FF paths.
        % Assuming outer FB be 1. Will scale according to the actual FB coeffcient.
        B = [1; zeros(order-1,1)];
        D = [0];   % Cc doesn't change.
        for i = 1:order-2,
            A(order,i) = 1;
            ys(:,i+2) = squeeze(step(ss(A, B, C, D),0:dt:n_imp-t1+dt/2))-ys(:,1);
            A(order,i) = 0;
        end
    case 'HB2'
        if (order < 3) error('The hybrid structure requires loop order >=3.');
        end
        % first calculate the step response of the feedback paths
        C = [zeros(1,order-1) 1];
        B = eye(order); B(:,2) = [];
        D = zeros(1,order-1);    % not include direct feedback.
    ys(:,1:order-1) = squeeze(step(ss(A, B, C, D),0:dt:n_imp-t1+dt/2));
    % Calculate the step response of the internal FF path.
    % Assuming outer FB be 1. Will scale according
    % to the actual FB coeffcient.
    B = [1; zeros(order-1,1)];
    D = [0];   % Cc doesn't change.
    A(3,1) = 1;
    ys(:,order) = squeeze(step(ss(A, B, C, D),0:dt:n_imp-t1+dt/2))-ys(:,1);
    A(3,1) = 0;
case 'HB3'
    if (order < 4) error('HB3 requires loop order >=4.');
    end
    % first calculate the step response of the feedback paths
    C = [zeros(1,order-1) 1];
    B = eye(order); B(:,2:3) = [];
    D = zeros(1,order-2);    % not include direct feedback.
    ys(:,1:order-2) = squeeze(step(ss(A, B, C, D),0:dt:n_imp-t1+dt/2));
    % Calculate the step response of the internal FF path.
    % Assuming outer FB be 1.
    % Will scale according to the actual FB coeffcient.
    B = [1; zeros(order-1,1)];
    D = [0];   % Cc doesn't change.
    for i = 1:2,
        A(4,i) = 1;
        ys(:,order-2+i) = squeeze(step(ss(A, B, C, D),0:dt:n_imp-t1+dt/2))
                          -ys(:,1);
        A(4,i) = 0;
    end
otherwise
    error('%s error. "%s" is not supported', mfilename, form);
```

```
end


%% 2) Calculate pulse reponse by subtraction ....
%%     a) add t1/dt or t2/dt amount of zeros to represent edge->t1
%%      and edge->t2 responses.
%%     b) y1, y2 = ys values at samples points...
%%     c) pulse response = step response(y2) - step response(y1)

y_1 = [zeros(round(t1/dt), order); ys]; y_1 = y_1(sample_points,
:); y_2 = [zeros(round(t2/dt), order); ys]; y_2 =
y_2(sample_points, :); yy = y_1 - y_2;

%% 3) Endow yy with n_extra+1 extra impulses
if n_extra ~= 0
    y_right = padb([zeros(1, n_extra); eye(n_extra)], n_imp+1);
    yy = [yy(:,1:end) y_right(:, end:-1:1)];
else
    yy = [yy(:,1:end)];
end

%% yy -> Pulse response, y -> Impulse response ................
x = yy\y; if norm(yy*x-y)>1e-4
    warning('Pulse response fit is poor.');
end

%% D value:
switch form case {'FB', 'FBFF'}
    %%%%Check redundancy!!! of D with previous switch statement...
    Dc1 = [0];  %---for u_o
    if (n_direct)
```

```
  %       Dc2 = [x(order+1:end)' zeros(1,n_direct)];    7/19/2005
          Dc2 = [zeros(1, n_direct) x(order+1:end)'];
    else
        Dc2 = [0];
    end
case 'FF'
   % Dc1 = zeros(order,1);
    %%D = zeros(order,2+n_extra);
    %%D(1, 3:end) = (x(order+1:end));
    Dc1 = [0];
    if (n_direct)
        Dc2 = [0 x(order+1:end)'];
    else
        Dc2 = [0];
    end
case {'HB', 'HB2', 'HB3'}
    Dc1 = [0];
    if (n_direct)
        Dc2 = [0 x(order+1:end)'];
    else
        Dc2 = [0];
    end

otherwise
    error('%s error. "%s" is not supported', mfilename, form);
end

D = [Dc1 Dc2];
%% A and B Values
switch form case {'FB','FBFF'}
```

```
    Bc2 = [x(1:order) zeros(order, size(x(order+1:end), 1))];
case 'FF'
    Bc2 = [[-1; zeros(order-1,1)] zeros(order, n_extra)];
    C = x(1:order)';
        %Bc2 = [[-1; zeros(order-1,1)] zeros(order, n_extra)];
    %Cc = diag(x(1:order));
case 'HB'
    Bc2 = [[x(1); zeros(order-2,1); x(2)] zeros(order, n_extra)];
    A(order,1:order-2) = x(3:order)'/x(1);
case 'HB2'
    Bc2 = [[x(1); 0; x(2:order-1)] zeros(order, n_extra)];
    A(3,1) = x(order)'/x(1);
case 'HB3'
    Bc2 = [[x(1); 0; 0; x(2:order-2)] zeros(order, n_extra)];
    A(4,1:2) = x(order-1:order)'/x(1);
otherwise
    error('%s error. "%s" is not supported', mfilename, form);
end

%% Scale Bc1 for unity STF magnitude at f0
Bc1 = [1; zeros(order-1,1)];
L0c = zpk(ss(A,Bc1,C,Dc1));  %--- for uo
G0 = abs(evalTFP(L0c,ntf,f0));  % product of L0c * ntf at frequency f0 ...?!?
B = [Bc1/G0 Bc2];

% remove elements less than lmin.
lmin = 1e-9; A=A.*(abs(A)>lmin); B=B.*(abs(B)>lmin);
C=C.*(abs(C)>lmin); D=D.*(abs(D)>lmin);

L1_c = ss(A, B, C, D);
```

Appendix C

add_delay.m

```
function [L1array, delta] = add_delay(Hdummy, L1, tdac, n, order,
form, n_extra)

%For FB for now...
%
% n = nth feedback loop that the delay is applied..
%     in L-> R Direction.
% Sweep each feed-ins by dt....
%
% L1array{row, col} contains transfer fuction in zpk format for
% delay = delta(col) applied to nth (i.e. row-th) feed-ins.

% Extracting relavation information ...
[n_output, n_input] = size(L1); t1 = tdac(1); t2 = tdac(2);

% Sweeping time steps ....
dt = 0.01;
%dt = 0.002;

% Set the delay:
```

```
input_delay = zeros(1, n_input); delta = 0:dt:1;

% Compute H with delay, delta, and store it in Hzpk.
% Should have length(delta)
for i = 1:length(delta)
    Hc = L1;

    input_delay(n+1) = t1 + delta(i);
%-- because feedback starts at u2 not at u1 (u1 = actual input)

    if(n_input+1-n_extra) == (n+1),
        error('%s Error: Feedback delay length has problem', mfilename);
    end

    input_delay((n_input+1-n_extra):n_input) = 1;
    d{n}{i} = [sprintf('input delay array is ')
                sprintf('%d ', input_delay)];

    %    if form ~= 'FB'
    %        input_delay(n+2) = 0.5;
    %    end

    set(Hc, 'InputDelay', input_delay);  %??? [0 delta 0.5]  why 0.5??
    Hd = c2d(Hc, 1, 'zoh');

    % Converting L1(z) to H(z)

    Hsplit = ss([0], [0], zeros(n_input-1, 1), ones(n_input-1, 1), 1);
    Hcombine = series(Hsplit, Hd, [1:n_input-1], [2:n_input]);
    %     switch form
    %     case 'FB'
    Hfeedback = 1;
    Hntf = feedback(Hfeedback, Hcombine, [1], [1], 1);
    Hzpk = zpk(Hntf);



    Hdummy{n,i} = Hzpk;
end

L1array = Hdummy;
% %%%%TESTING ADD_DELAY %%%%%
% N = {[1 -1 ],[1 2], [-1 1]};
% D = {[1 1],[1 4 5], [1 2 3]};
%
% N1 = {[1 -1];[-1 1];[2 1]};
% D1 = {[1 1]; [1 2 3];[1 3]};
%
% L1 = tf(N,D); H1 = tf(N1,D1);
% Hss = ss(L1); Hss1 = ss(H1);
% new_L1 = add_delay(Hss, 1); new_H1 = add_delay(Hss1, 1);
% new_L1.inputdelay
% new_L1.outputdelay
% new_H1.inputdelay
% new_H1.outputdelay
```

Appendix D

test_bench.m

```
%function test_bench()

% To test 'linear delay' model
% Parameters needed:
%       -order, OSR, opt, Hinf, f0, tdac, form

clear; figure(3); clf; figure(4); clf;

%test_var;
%OSRs = [32 48 64 96];
%orders = [3 4 5 6 7];
%H_infs = [1.9 1.9 1.9 1.9 1.9];
%nlevs = [5 17 18 20 14 12 2];

OSRs = [32]; orders = [3 4 5 6 7 8];
%orders = [3];
H_infs = [1.9]; nlevs = [5]; labels = {};

for z = 1:length(orders)
```

```
fprintf('order =%d \n', orders(z));
%z = 1;

clear Q t Hc H;

order = orders(z);
OSR = OSRs(1);
H_inf = H_infs(1);
nlev = nlevs(1);

tdac = [1 2];
form = 'FB';
opt = 1;
f0 = 0 ;
DR  = 100;
% Synthesize NTF

ntf_dt = synthesizeNTF(order, OSR, opt, H_inf, f0);

%% Check validitiy of resulting NTF:

dr = -dbv(rmsGain(ntf_dt,f0-0.5/OSR, f0+0.5/OSR))
    +dbp(OSR)+dbv(nlev-1)+1.76;
dr_print = sprintf('Dynamic Range Low: Only %d', dr);
if (dr < DR) warning(dr_print); end

% Realize NTF in CT for a given form (MISO or SIMO output)

[Hc, n_extra] = realizeNTF_MIMO(ntf_dt, form, tdac);
%Hc = realizeNTF_ct(ntf_dt, form, tdac, f0)
```

```
% Apply varying delays to the feedbacks
% H contains tf ...
H = {};

[n_output, n_input] = size(Hc);

switch form
case {'FB', 'FBFF'}
    n_in = order;
case 'FF'
    n_in = 1;
otherwise
    error('%s error. "%s" is not supported', mfilename, form);
end

for i = 1:n_in
    [H, delta] = add_delay(H, Hc, tdac, i, order, form, n_extra);
end
%[H, delta] = add_delay(H, Hc, tdac, 1, order, form);

for j = 1:size(H,1)
    k = 1;
    for i = 1:size(H,2)
        ntf = H{j,i};
        q = max(abs(angle(ntf.p{:}))./(1-abs(ntf.p{:})));
        [mag{j,i}, pha{j,i}] = bode(ntf);

%            if ((min(q) <0 | max(q) > 200)) break;
        if ((min(q) < 0)) break;
        else
            Q(j,k) = max(q);
            t(j,k) = delta(i);
            k= k+1;
        end

    end
end
%%%% Visualization:
colors = {'yh' 'mp' 'c>' 'r<' 'g^' 'bv' 'kd'
        'ys' 'm*' 'c+' 'rx' 'go' 'b+'};
while length(colors) < order
    colors = {colors{1:end} colors{1:order-end}};
end

switch form
case {'FB', 'FBFF'}
    figure(3);
    subplot(2, 3, z);



    for i = 1:size(t,1)
        %for  i=1:1
        plot(t(i,:), Q(i,:), [colors{end+1-i}]);
        hold on;
        axis([0 1 0 1000]);
        title(sprintf('Order = %d', order));
        xlabel('time delay');
        ylabel('Q');
        grid on;
        end

    case 'FF'
        figure(4);
        plot(t(1,:), Q(1,:),[colors{end+1-z}]);
        hold on;
        axis([0 1 0 50]);
        labels{z} = sprintf('%dth', order);
        xlabel('time delay');
        ylabel('Q');
        grid on;

    otherwise
        error('%s error. "%s" is not supported', mfilename, form);
    end


end % for z=1:length(orders)

switch form case {'FB', 'FBFF'}
    labels = {};
    for i = 1:max(orders)
        labels{i} = sprintf('k_%d', i);
    end
    legend(labels);
case 'FF'
    legend(labels);
otherwise
    error('%s error. "%s" is not supported', mfilename, form);
end
```

Appendix E

validity_test.m

```
%Demonstrate that my method is correct by showing
%SPD of output of non-ideal SDM.

[ntf_delays, deltas] = test_bench_SISO; close all;
%OSRs = [32];
%orders = [3 4 5 6 7 8];
%H_infs = [1.9];
%nlevs = [5];

%Declare appropriate variables:
OSR = 32; order = 5; H_inf = 1.9; nlevs=5; opt = 1; f0=0; DR =
100; tdac = [1 2]; R = OSR;

%Discrete Time NTF without any delay:

ntf_dt = synthesizeNTF(order, OSR, opt, H_inf, f0); [f, y] =
SPDplot(R, ntf_dt);

%Plot w/o delay:
fig; title('Output Spectrum of NTF with OSR = 32, order = 6 and
zero delay');

subplot(2,2,1); plot(f, y, 'b'); hold on; axis([0 0.5 -120 0]);
grid on; xlabel('Normalized Frequency') ylabel('dBFS') title('DT
NTF');

%NTF with delay added....
FB_H = ntf_delays{1}{1}; FF_H = ntf_delays{2}{1}; FBFF_H =
ntf_delays{3}{1};

[f_fbd, y_fbd] = SPDplot(R, FB_H); [f_ffd, y_ffd] = SPDplot(R,
FF_H); [f_fbffd, y_fbffd] = SPDplot(R, FBFF_H);

%Plot w/ delay:
subplot(2,2,2)
plot(f_fbd, y_fbd, 'r'); hold on; axis([0 0.5 -120
0]); grid on; xlabel('Normalized Frequency') ylabel('dBFS')
title('FB');

%Plot w/ delay:
subplot(2,2,3)
plot(f_ffd, y_ffd, 'r'); hold on; axis([0 0.5 -120
0]); grid on; xlabel('Normalized Frequency') ylabel('dBFS')
title('FF');

%Plot w/ delay:
subplot(2,2,4)
plot(f_fbffd, y_fbffd, 'r'); hold on; axis([0 0.5
-120 0]); grid on; xlabel('Normalized Frequency') ylabel('dBFS')
title('HB');
```

Appendix F

SPDplot.m

```
function [f, y] = SPDplot(R, ntf_dt);

N=8192; fB = ceil(N/(2*R)); ftest=floor(2/3*fB);
u = 0.5*sin(2*pi*ftest/N*[0:N-1]);  % half-scale sine-wave input
v = simulateDSM(u,ntf_dt, 5);

f = linspace(0,0.5,N/2+1); spec = fft(v.*hann(N))/(N/4);

y = dbv(spec(1:N/2+1));
```

Bibliography

[1] R. W. Adams. Design and implementation of an audio 18-bit analog-to-digital converter using oversampling techniques. *Audio Eng. Soc.*, 34:153–166, March 1986.

[2] B.E. Boser, K.-P. Karmann, H. Martin, and B.A. Wooley. Simulating and testing oversampled analog-to-digital converters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7:668 - 674, June 1988.

[3] Lucien Breems and Johan H. Huijsing. *Continuous-Time Delta-Sigma Modulation For A/D Conversion In Radio Receivers.* Kluwer Academic Publishers, 2001.

[4] William L. Brogan. *Modern Control Theory.* Quantum Publishers, INC., 1974.

[5] J.A Cherry and W.M Snelgrove. Approaches to simulating continuous-time delta sigma modulators. *Proceedings of the 1998 IEEE International Symposium on Circuit and Systems*, 1:587–590, June 1998.

[6] Gene F. Franklin, J. David Powell, and Michael Workman. *Digital Control of Dynamic Systems.* Addison Wesley Longman, Inc., 1998.

[7] W. Gao, O. Shoaei, and W.M. Snelgrove. Excess loop delay effects in continuous-time delta-sigma modulators and the compensation solution. *Proceedings of 1997 IEEE International Symposium on Circuits and Systems*, 1:9–12, June 1997.

[8] W.Martin Snelgrove James A. Cherry. *Continuous-Time Delta-Sigma Modulators For High-Speed A/D Conversion.* Kluwer Academic Publishers, 2000.

[9] J. F. Jensen, G. Raghavan, A. E. Cosand, and R. H. Walden. A 3.2 hz second-order delta-sigma modulator implemented in ing hat technology. *IEEE J. Solid-State Circuits*, 30:119–1127, October 1995.

[10] Fernando Medeiro, Angel Perez-Verdu, and Angel Rodriguez-Vazquez. *Top-down design of high-performance sigma-delta modulators.* Kluwer Academic Publisher, 1999.

[11] J. Moreno-Reina, J.M. de la Rosa, F. Medeiro, R. Romay, R. del Rio, B. Perez-Verdu, and A. Rodriguez-Vazquez. A simulink-based approach for fast and precise simulation of switched-capacitor, switched-current and continuous-time /spl sigma//spl delta/ modulators. *Proceedings of the 2003 International Symposium on Circuits and Systems*, 4:IV–620–623, May 2003.

[12] M. Ortmanns, F. Gerfers, and Y. Manoli. A continuous-time sigma-delta modulator with switched capacitor controlled current mode feedback. *Proceedings of the 29th European Solid-State Circuits Conference*, 16:249–252, September 2003.

[13] Maurits Ortmanns and Yiannos Manoli. A continuous-time sigma-delta modulator with reduced jitter sensitivity. *Proc. ESSCirC*, 1:287–290, January 2002.

[14] G. C. Temes S. R. Norsworthy, R. Schreier. *Delta-Sigma Data Converters.* IEEE Press, 1997.

[15] Richard Schreier. Delsig toolbox. Online, January 2000. http://www.mathworks.com/matlabcentral/fileexchange/.

[16] Richard Schreier and Bo Zhang. Delta-sigma modulators employing continuous-time circuitry. *IEEE Transactions on circuits and systems - I: Fundamental Theory and Applications*, 43:n/a, April 1996.

[17] C.M. Wolff and L.R. Carley. Simulation of δ-σ modulators using behavioral models. *Proceedings of the 1998 IEEE International Symposium on Circuit and Systems*, 1:376–379, May 1990.

3233-63