Feasibility of a 16bit, 3MSPS Multibit per Stage Pipeline ADC using Digital Calibration

by

Matthew Louis Courcy

Submitted to the Department Of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

February 4, 1998

Signature of Author ....

Department of Electrical Engineering and Computer Science
February 4, 1998

Certified by.....

2/4/98

Professor Hae-Seung Lee
Thesis Supervisor

Accepted by.....

Arthur C. Smith
Chairman, Department Committee on Graduate Students

Feasibility of a 16bit, 3MSPS Multibit per Stage Pipeline ADC using Digital Calibration

by
Matthew Courcy

Submitted to the

Department of Electrical Engineering and Computer Science

February 4, 1998,

In Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and Master of Engineering in Electircal Engineering and Computer Science

ABSTRACT

This work examines the feasibility of using a multibit per stage pipeline ADC to achieve high resolution and linearity. The proposed architecture uses simple digital calibration of the first stage to achieve high linearity with low digital complexity. By keeping the digital complexity low, the ADC escapes the effects of coupled digital noise present in current pipeline converters at this resolution.

A 16 bit 3 Ms/s pipeline ADC architecture using digital calibration is considered in response to market needs. Background is given on ideal nyquist rate ADCs and some associated performance metrics. Problems due to process parameters and physical non-idealities are discussed. Methods to correct these problems are presented. An overview of pipeline architecture trade-offs is given, and an architecture, based on a current 14 bit ADC, is chosen for the proposed ADC. Circuit modifications necessary to realize the new 16 bit ADC are discussed in detail. Final analysis of a simulated ADC is given, and issues pertaining to final design are outlined.

Thesis Supervisor: Hae-Seung Lee
Title: Professor, Department of Electrical Engineering and Computer Science

Acknowledgments

I would like to recognize Larry Singer, Todd Brooks, Dave Robertson, Katsu Nakamura, Mike Timko, Stacy Ho, Steve Harston, and Donald Paterson along with the rest of the High Speed Converter Group at Analog Devices Inc. Without their expertise and help, I would never have been able to complete this undertaking. I would especially like to thank my supervisor, Larry Singer, who put up with hours of questioning, and persevered through my detailed solutions of the least intuitive problems. I am not sure I would be writing this without him. I'd also like to recognize Proffessor Hae-Seung Lee at MIT for keeping my sanity in check during my time at Analog and during my thesis preparation.

I would like to thank my parents for their love and support in all aspects of my education, including this thesis. I would like to thank my sisters, Lori and Tracey, and my uncle Gill for believing in me through all of my hardships. I'd also like to recognize Kate. Without her support, I would have never been able to write this document.

This research was supported by Analog Devices, Inc.

Table of Contents

Table of Figures

Chapter 1

Introduction

1.1 Motivation

During the past ten years, the field of high-speed Analog-to-Digital conversion has grown extensively. Medical and Imaging technologies have presented a need for Analog-to-Digital Converters (ADCs) with sampling rates on the order of megasamples to tens of megasamples per second (Ms/s). The earliest high speed data converters were Flash ADCs. Flash type ADCs were limited to resolutions no greater than 8 bits. As the ADC market pushed towards higher resolution ADCs, two-step and pipeline architectures promised higher resolution while covering a smaller area and consuming less power. The new architectures did not operate quite as fast as their Flash counterparts, but in the 10Ms/s market they won new customers. The promise of these new designs lead to the flooding of the 10 to 12 bit ADC markets. The market stalled at 12 bits and the market began to focus more on lower power consumption and higher sampling rates. ADC resolution plateaued as a result of the new focus.

Another primary direction of ADC design in the past years has been toward the monolithic implementation of mixed-signal integrated circuits. The coexistence of analog and digital circuitry on the same substrate required that both analog and digital components be manufactured using the same process technology. CMOS (Complementary MOS) has proven to be the least expensive process with which to implement both analog and dig-

ital circuitry. A hybrid process named BiCMOS (Bipolar and CMOS) has been used to implement higher frequency analog circuits while maintaining the digital capability of CMOS. Due to the extra processing steps required, BiCMOS is more expensive to implement than CMOS, making CMOS the technology of choice. By using these well defined processes, many analog electronics firms are currently competing to produce the state of the art in the 10 to 12 bit high speed ADC market. The current state of the art in the 10 bit range is a 100MS/s CMOS ADC consuming 1.1W of power.[1] In the 12 bit market, a 50Ms/s BiCMOS (Bipolar and CMOS) ADC consuming 535mW of power marks the state of the art. [2]

The market for high-speed, high-resolution ADCs has just started to push beyond the 12 bit level. The period of stagnation in resolution enhancement has been due to the inability of companies to produce pipeline ADCs that possessed sufficient linearity at resolutions greater than 12bits. Non-linearity is caused by the inherent inaccuracy of today's ADC manufacturing technology. In particular, the mismatch of transistors, resistors, and capacitors, cause the greatest linearity problems in pipeline ADCs. A few circuit techniques and measurement methods can be used to minimize the effects of this mismatch. Architecture changes alone also allow greater than 12bit resolution to be achieved. Using a multi-bit per stage pipeline architecture, Analog Devices Inc. has reported a 14 bit pipeline ADC running at 10MS/s. [3] This ADC has high resolution but lacks sufficient linearity. The use of high-speed sigma-delta architectures has also allowed high-speed oversampling ADCs to press beyond the 12 bit level. Analog Devices, Inc. has reported a 16 bit Sigma-Delta based ADC using oversampling techniques to perform 16 bit linear ADC functions. The ADC marks the state of the art in high-speed, high-resolution dynamic ADCs. [4] Many high resolution pipeline ADCs use calibration as method of increasing ADC linearity. Analog Devices has reported a 14 bit 2.5MS/s BiCMOS factory calibrated pipeline ADC. [5] The report stated that the ADC consumed 500mW of power. In comparison to other linear 14bit ADCs of the time, it was relatively power-efficient. In an even higher resolution arena, 16 bit, National Semiconductor has produced a 16bit 1MS/s digitally self-calibrated ADC, representing the state of the art in high-resolution nyquist rate ADCs. [6] Calibration in this ADC is responsible for adding a large amount of digital complexity to the ADC core. The switching noise associated with the digital cir-

cuitry ends up coupling through the substrate and corrupting the analog signal path. Consequently, the ADC does not deliver true 16 bit noise performance.

The motivation of this thesis is to examine the feasibility of a 16bit ADC running at 3MS/s based on the 14bit converter architecture presented in [3]. This ADC employs digital calibration to increase linearity; yet the calibration is kept simple in order to diminish the effects of digital noise coupling. With the use of digital calibration, the ADC is specified to achieve one quarter LSB DNL at the target resolution of 16 bits. The ADC alone is also be specified to generate no more than one third LSB of input referred RMS noise.

1.2 Organization

This remainder of this thesis will be organized in the following manner. Chapter 2 discusses the evolution Flash ADCs to pipeline ADCs, and common linearity metrics used in describing Nyquist rate converters. Chapter 3 discusses problems that arise in pipeline ADCs as a result of processing errors and physical non-idealities. Methods for correcting these errors will also be discussed at part of chapter 3. Chapter 4 discusses the trade-offs of modern pipeline architectures, and makes an argument for the use of multibit per stage ADCs in high resolution applications. Chapter 5 introduces the new ADC architecture and discusses several modifications made to the existing AD9243 which result in the proposed 16 bit pipeline. Finally, chapter 6 reviews the results of a behavioral simulation on the 16 bit ADC architecture and briefly outlines issues of the future design process that must be solved before release.

Chapter 2

Evolution of the High-Speed Pipeline ADC

2.1 Introduction

Over the years ADCs have been developed with a wide range of resolutions and speeds. Within the ADC family exists a sub-family of converters whose input sample rates match their output data rates. These converters are generally able to sample input frequencies up to half the sampling rate, and are called nyquist rate converters. Unlike oversampled ADCs, nyquist rate converters do not rely on increased clock speed to increase linearity or decrease noise. Nyquist rate operation allows this family of ADCs to convert much higher frequency signals than their oversampling counterparts. This fact makes nyquist rate converters the clear choice for high- speed applications. The fastest nyquist rate converters were Flash ADCs. Flash ADCs had a number of drawbacks. An evolution towards new high speed converter architectures occurred in response to the Flash ADC's drawbacks. Along one path, ADCs evolved from the low-resolution high speed Flash ADC to the higher resolution pipeline ADC. This chapter gives an overview of the evolutionary path and what drawbacks and advantages were gained along the way.

2.2 Non-linearity Metrics for Nyquist Rate Converters

Before discussing the events that lead to the conception of the modern pipeline converter, a discussion of certain performance metrics for nyquist rate ADCs must be given. Linearity, in any ADC, is very important. An ideal ADC has a linear staircase transfer function mapping distinct sections of the analog input range to individual digital output

codes. The ideal N-bit ADC transfer function looks like a staircase with 2^N steps, each of which has equal run. Designers try to emulate the ideal transfer function when designing linear ADCs, but errors will still exist. Linearity metrics are used to describe these errors. The two linearity metrics commonly used in describing nyquist rate converters are differential non-linearity (DNL) and integral non-linearity (INL). Together these two metrics can be used to illustrate an ADC's non-linear properties.

The first linearity metric discussed is differential non-linearity. DNL is defined as the deviation of each digital output code width from the ideal. [7] Figure 2.1 shows an ideal ADC transfer function segment and a similar segment with DNL. Figure 2.1 also displays plots of 'output code vs. DNL,' a visual representation of the metric. The ideal digital code width is given by,

$$Ideal = \frac{FullScale}{2^N} \qquad \text{(Eq. 2.1)}$$

where FullScale is the ideal fullscale voltage range of the converter, and N is the resolution of the ADC in bits. DNL values of +1/4 LSB and -1/4 LSB correspond to digital code widths of 5/4 times and 3/4 times the ideal respectively. Specifications often state that a converter's DNL falls between -1/2 and +1/2 LSB. This means that the step widths range between 1/2 to 1 1/2 times the ideal respectively. The term, 'N-bit DNL' corresponds to the system having DNL no greater than ±1 LSB at N-bit resolution. The same can be said for the term 'N-bit linearity' but in a more general sense. DNL surpassing +1 LSB denotes a code width more that twice the ideal. Non-monotonicities or "extra codes" are indicated in DNL plots by two or more consecutive codes with greater than +1 LSB DNL. At the other extreme, -1 LSB DNL indicates that a zero width code exists, a code which is 'missing.' 'Extra' and 'missing' codes and some of their causes are discussed in more detail in chapter 3.

Code width variations inevitably causes the slope of the ADC transfer function tovary across the input range. Slope variation causes the ADC transfer function to waver around a straight line lying between the lowest and highest output codes. The difference between the non-linear transfer function and the ideal straight line ADC transfer function is referred to as the Integral non-linearity (INL) of the system. [7] Figure 2.2 shows the

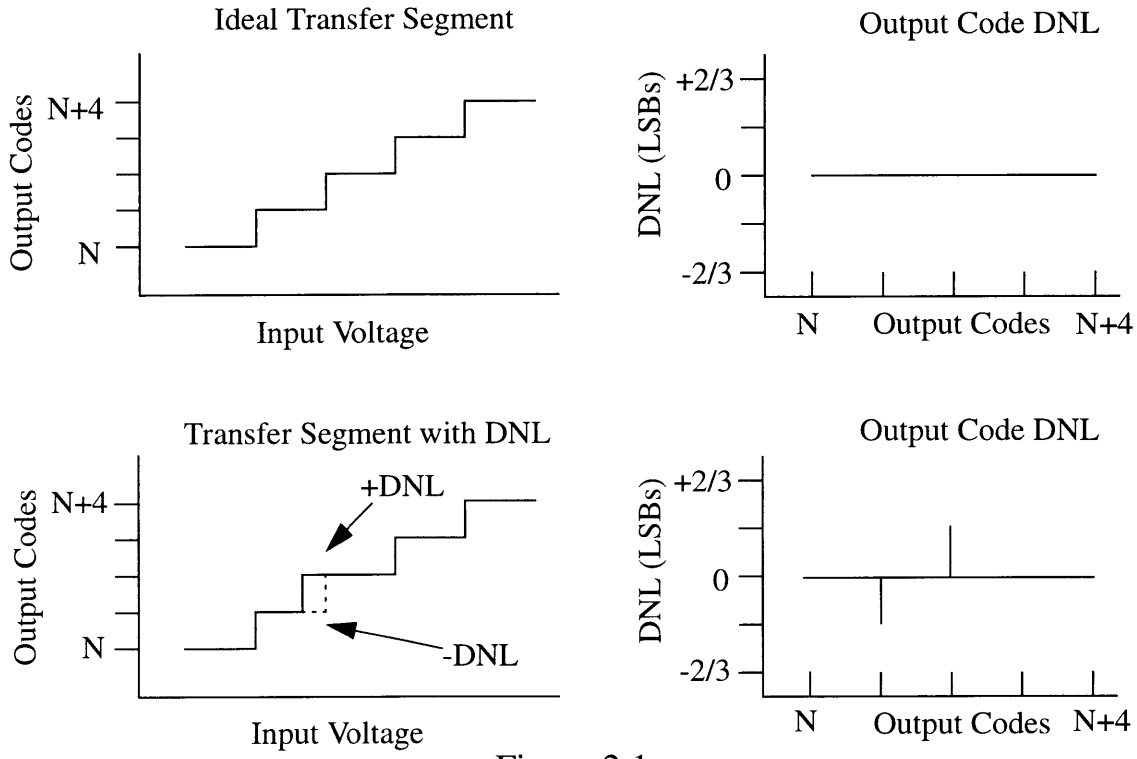


Figure 2.1

transfer function of an ADC with INL and a "code vs. INL plot", the graphical representation of INL. The largest factor affecting ADC INL is circuit non-linearity. These factors will be discussed more in-depth in chapter 3. Now that INL and DNL have been presented, the evolution from the Flash ADC to the Pipeline ADC can be discussed.

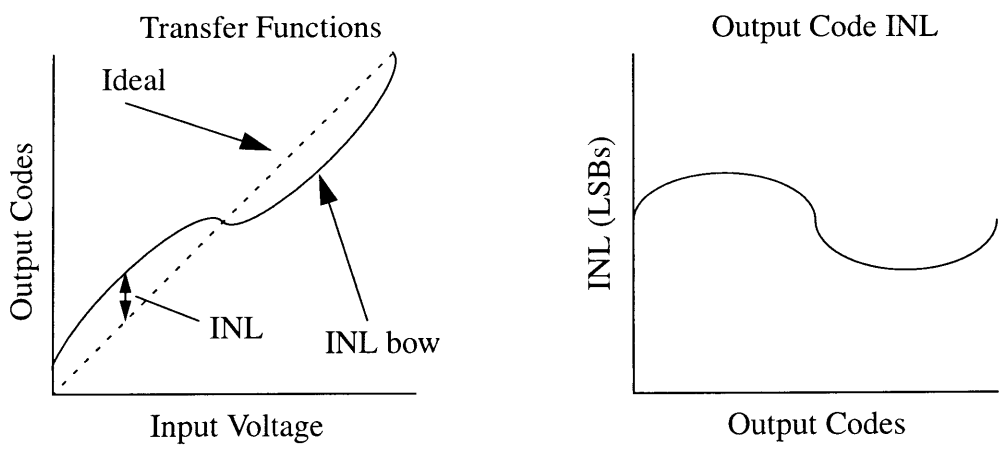


Figure 2.2

2.3 The Flash ADC

One of the fastest nyquist rate ADCs is the Flash ADC. The Flash ADC is based on the principal of equally partitioning the input range into 2^N distinct adjacent segments, determining in which segment the input sample exists, and mapping that segment to a distinct code in the digital domain. This is all done during one clock cycle. Carrying out this principal in practice requires a large amount of processing power. The Flash ADC employs a highly parallel architecture to supply processing power and throughput. A Flash ADC, shown in Figure 2.3, consists of an array of 2^N-1 comparators making decisions at 2^N-1 equally distributed analog voltages. The Flash input signal is routed to the input of each comparator and decision levels are supplied by a string of 2^N resistors. The output of

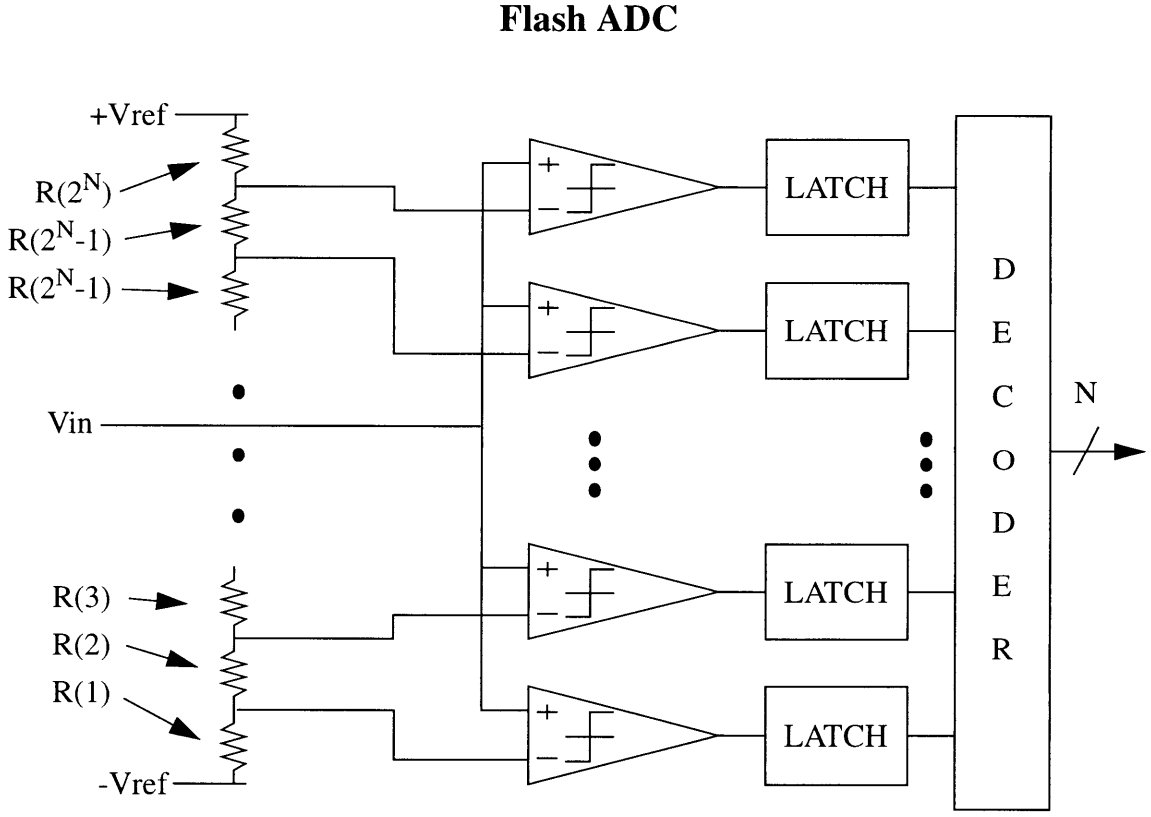


Figure 2.3

the 2^N-1 comparators are latched once every clock cycle and the latch output lines are mapped to a digital code via the decoder. Flash ADC conversion speed is dependent on a

few factors. The comparator reaction time, the time it takes a change at the input to change the output, limits ADC accuracy and speed. For fast comparators this reaction time is quite small. Latching speed and the decoder propagation delay also limit clocking speed. These delays are typically on the order of a few nanoseconds. Due to these small time delays, the Flash architecture is a good choice for very fast applications.

The Flash architecture does have a number of drawbacks. For each new bit of resolution added to the ADC, the total number of comparators doubles. As a result, the ADC size and power dissipation vary exponentially with resolution. Consequently, Flash ADC resolution has not surpassed 8 bits without size and power problems. In practice, the Flash architecture is also prone to non-linearity problems at high resolutions. In reality, resistors never match exactly. Mismatch between string resistors causes non-linearity in the Flash ADC decision spacing, leading directly to system INL. In addition, each comparator in the array has a random input offset voltage. Offset voltages shift the comparator decision points about randomly adding to system DNL. Low resolution Flash ADCs are less prone to these non-linearity errors because decision spacing is typically much larger than the comparator offsets or mismatch errors. The next generation of high speed ADC based on this Flash architecture, the two-step ADC, was conceived to solve some of the problems that plagued the Flash ADC.

2.4 The Two-Step ADC

A method of reducing the number of comparators in an ADC is to break the quantization operation into two parts. If N/2 bits are returned by an initial coarse quantization, and N/2 bits are then returned in a second fine quantization, the number of necessary comparators is reduced from 2^N to $2^{N/2+1}$. A digital output code is generated by adding the fine quantization to the coarse quantization. The process just described is the principle of the two-step architecture. Figure 2.4 shows a block diagram of a two-step ADC architecture. During the first step, the coarse Flash ADC quantizes the input sample to N bits. The digital output is input to a reconstruction DAC. The DAC output is then subtracted from the

original input sample. This value is then multiplied by a gain of 2^N to produce a residue which spans the second step's input range. Residue is sometimes plotted in relation to input voltage as shown by Figure 2.5. In the second step, the residue is quantized by the fine Flash ADC to M bits. The fine M bits and coarse N bits are then added as shown in Figure 2.6 to produce an M+N bit code. More resolution can be attained in a two-step architecture than in a conventional Flash due to the relaxed requirements on comparator array size. A two-step ADC with a resolution of (4N-2) bits can be realized with the same number of comparators necessary to construct a 2N bit Flash ADC. The ability to increase resolution with little increase in power and area makes the two-step a better choice for higher resolution applications.

The two-step architecture has its own drawbacks. In practice, two-step ADCs are often implemented as switched-capacitor circuits. The residue gain element situated between the two stages is implemented using an operational amplifier. In a switched capacitor ADC, the residue amplifier needs time to settle to its final output value. The settling time of the amplifier is governed by its closed loop gain and is typically much slower than digital propagation delay. Consequently, a two-step ADC has a slower conversion rate than the Flash ADC. Also, in addition to the non-linearity introduced by the Flash ADC in each of the two steps, errors incurred by the reconstruction DAC, adder, and gain-ADC in each of the two steps, errors incurred by the reconstruction DAC, adder, and gain element cause DNL and INL. In a two step architecture, DNL and INL are typically smaller than in the Flash ADC case. This is true because the Flash ADC in each step has low resolution and subsequently low INL and DNL error while the DAC and gain error are smaller than high resolution Flash error when output referred. Chapter 3 and 4 shed more light on the effects of DAC and gain error on DNL and INL for two-step and pipeline ADCs. Even though resolution can be increased and both power and area can be decreased as a result of the two-step architecture, a maximum resolution still exists when the Flash converters and residue amplifiers become too large. By further reducing the resolution of each step and increasing the number of steps taken during the conversion, it is possible to alleviate the resolution limitation.

Two-Step Architecture

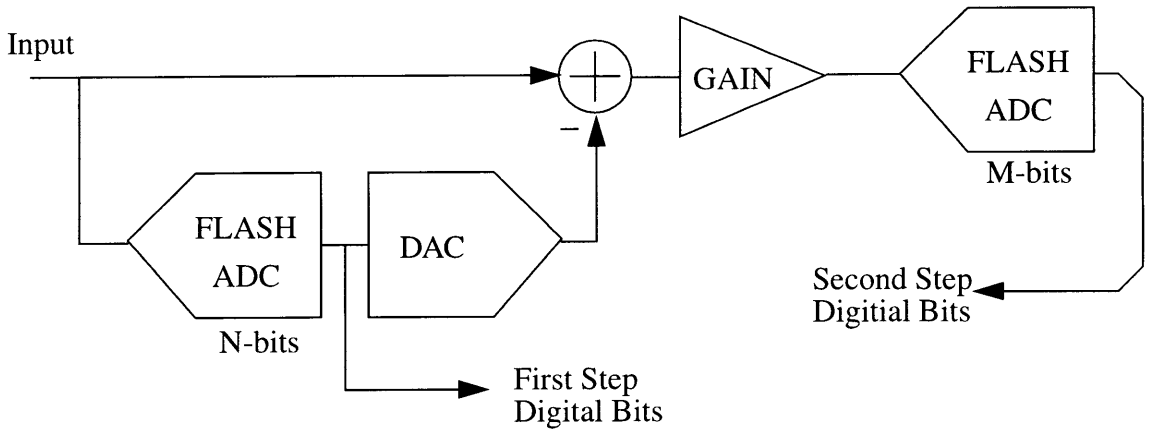


Figure 2.4

Step One Residue

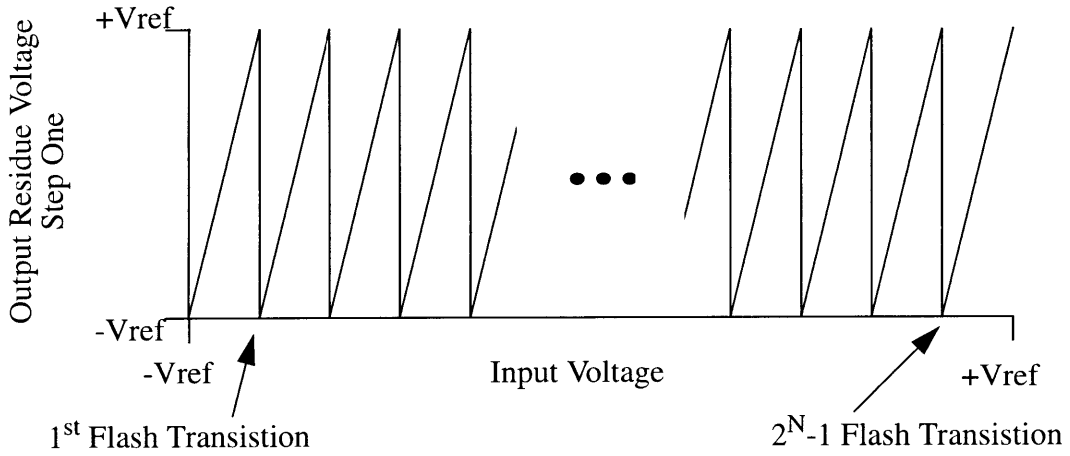


Figure 2.5

Two-Step Bit Additions

Assuming each step has six bit resolution...

	AAAAAA	(Bits returned by step one)
+	BBBBBB	(Bits returned by step two)
	AAAAAABBBBBB	(Digital output code)

Figure 2.6

2.5 The Pipeline ADC

The pipeline ADC is a natural generalization of the two-step ADC in which M-1 stages and a final Flash perform the M-step data conversion. The first M-1 stages have the form shown in Figure 2.7. As in the two-step ADC, each stage comes complete with a Flash, a reconstruction DAC, analog adder, and a residue scaler. The M^{th} stage of the structure is commonly a Flash ADC resolving the finest bits of the system. As shown in Figure 2.8, the bits from each stage are added together to form a digital output code. The pipeline converter is labeled "pipeline" because its operation allows for the high throughput common to digital pipeline circuits. Each stage possesses a sample and hold amplifier (SHA) used to sample the output voltage of the previous stage. As a result, each stage of the pipeline samples an input voltage, completes a residue amplification, has its output sampled by the following stage, and is then ready to sample the next input voltage all in one clock cycle. As a result, the pipeline ADC has the ability to take a sample on every clock cycle, the residues of which are gated down the pipeline one after another. Maximum throughput and Nyquist rate conversion are achieved by the pipeline ADC.

The pipeline has a number of advantages over the two-step ADC. The resolution of the ADC can be increased by increasing the number of steps used to do each conversion. Increased resolution can be achieved without a large number of comparators. Consequently, power and area can be reduced to the point where amplifier power dissipation and size exceeds that saved by reducing the Flash comparator array sizes. With decreased resolution in each stage comes a similar reduction in residue amplifier closed loop gain. Residue gain reduction in each pipeline stage allows for shorter settling time and increased speed. The pipeline ADCs sampling speed is hence limited by the slowest settling residue amplifier in the pipeline. Ideally, a pipeline ADC optimized for high-speed, low power dissipation, and small area would have a large number of stages each with one bit resolution. In practice, converter linearity and resolution are limited by errors occurring in the Flash ADCs, reconstruction DACs, and residue amplifiers. In practice, optimal designs may not have single bit per stage structures. The issue of optimal pipeline architecture is discussed in chapter 4, as pipeline architecture trade-offs are more closely examined.

Pipeline Stage Architechture

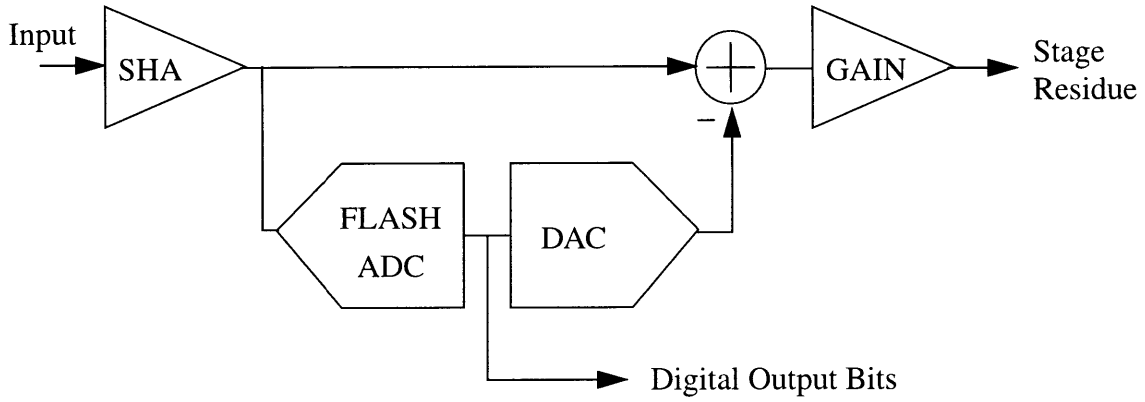


Figure 2.7

Pipeline Bit Additions

Assuming a four stage pipeline with each stage having four bit resolution

	AAAA	(Stage1 Bits)
	.BBBB	(Stage2 Bits)
	3CCCC	(Stage3 Bits)
+	:DDDD	(Stage4 Bits)
	AAAABBBBCCCCDDDD	(Digital output code)

Figure 2.8

As the evolution from Flash ADCs to pipeline ADCs progressed, optimizations were made for speed, power consumption, area, and resolution. Through the generations, higher resolution conversion has become a possibility. In reality, process and noise related errors cause problems in all sections of the pipeline architecture limiting resolutions to ten to twelve bits. As a consequence, the source and effects of these errors must be understood if they are to be corrected. The next chapter will shed more light of process related error and simple circuit techniques used to alleviate the problems they cause. Chapter 5 will give a description of noise effects found in switched capacitor pipeline ADCs.

Chapter 3

Correction Techniques for Physical and Process Related Errors

3.1 Introduction

Due to the inherent inaccuracy of modern semiconductor processing techniques, the ideal converters presented in chapter 2 can not be realistically implemented on silicon substrates. Errors generated by the processing of circuit components such as capacitors, resistors, and MOS transistors affect the signal path of the pipeline ADC directly. These effects corrupt the proper operation of the pipeline ADC, sometimes fatally. In order to perform proper ADC functionality it is very important to minimize or correct process errors. A discussion of the causes and effects of these errors is important in understanding how they might be corrected. This chapter will give insight into process related error and techniques for suppressing the effects.

3.2 Integrated Capacitor Errors

Capacitive elements in modern CMOS electronics are implemented in parallel plate configurations. Capacitors are commonly made of metal, polysilicon, or substrate-silicon. Integrated capacitors give circuit designers the impedance modules necessary to

build continuous-time filters as well as the charge storage ability necessary for switched-capacitor circuitry. Capacitors in high-precision switched-capacitor circuits need to match. Most linear circuits including continuous time filters and switched-capacitor ADCs require linear capacitor behavior as well. Capacitor mismatch and non-linearity currently cause problems in the production of high-accuracy, linear switched-capacitor ADCs.

Mismatch is dependent upon the method of processing integrated capacitors. The plates of any given capacitor are either deposited on the surface of an integrated circuit or diffused into the silicon substrate. The accuracy of plate dimensions is highly dependent on the accuracy of the photolithographic process used. Photolithographic processes usually cause dimensional defects on the order of 0.1um. This means capacitor lengths are widths can vary by ~.1um causing capacitor value fluctuation. Because the process is photographic, a certain amount of graininess occurs in the development of the photoresist. This graininess leads to edge roughness which causes dimensional deformation. [16] Dimensional variation can be seen in Figure 3.1. Alignment between masking steps can cause inconsistencies in the relative positioning of capacitor plates. This error can usually be averted by using a larger bottom plate in each capacitor. Small errors do still exist. An example of alignment error is displayed in figure 3.2. Uneven dielectric thickness is yet

Dimensional Lithography Error (Exaggerated)

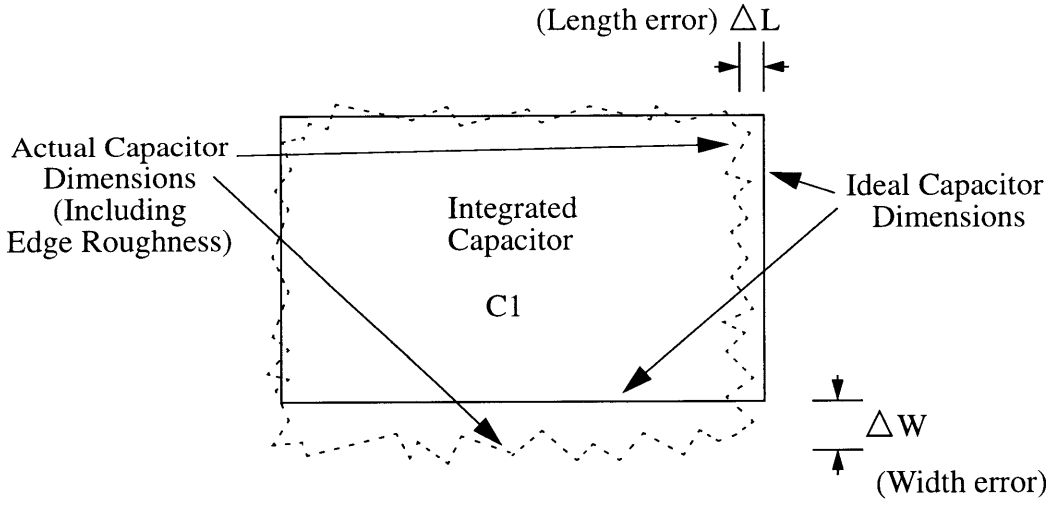
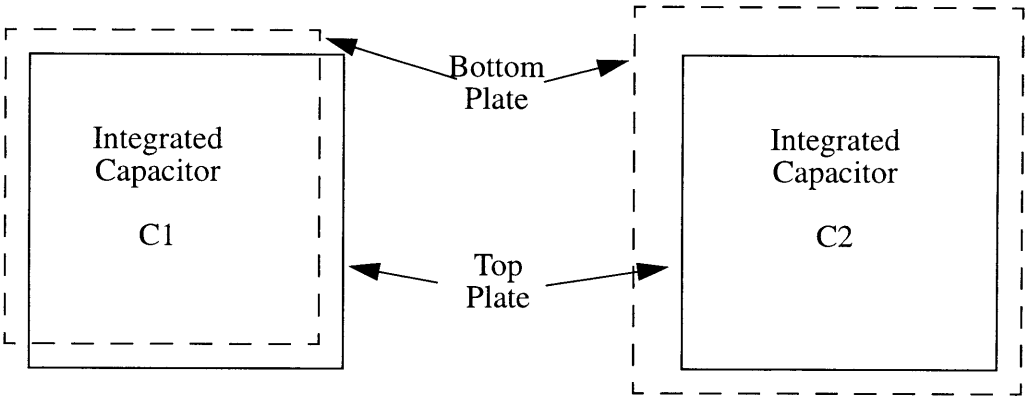


Figure 3.1

another error contributing to capacitor mismatch. Silicon-dioxide is the main dielectric



Nominally C1 and C2 have equal values. The mis-allignment of C1 causes its value to decrease, while the mis-allignment of C2 causes no appreciable different from its nominal value.

Figure 3.2

used in silicon processes. If the thickness of the oxide varies across the capacitor, the effective capacitance will vary accordingly. Common centroid layout techniques have been employed to combat this error, but as in the case of alignment, small errors still exist. An example of common-centroid correction is shown in Figure 3.3. In common centroid layout, components a placed across chip gradients and cross interconnected in order to average gradient effects and reduce the overall error. Because width and length deviations and edge roughness effects each have a fixed stochastic variance, the change for small area capacitors is a larger percentage of the total value than that of large area capacitors. Due to the compact size of low value capacitor arrays, oxide thickness gradients yield little error to mismatch; hence small capacitor mismatch is dominated by edge roughness and dimensional deviations. On the other hand, large value capacitor arrays take up much more area. Oxide gradients cause larger changes in value from capacitor to capacitor and the averaging properties of the common-centroid technique become weak producing adverse effects of matching. As mentioned above, large capacitor do not suffer that much from edge roughness and dimensional change; hence oxide gradients tend to dominate capacitor mismatch for large capacitor arrays. [15] Each of these process related errors causes problems for circuit designers.

Several groups have conducted studies on the capacitor mismatching properties of the processes which they use. Statistical data on capacitor mismatch vs. capacitor size is

used by designers who wish to predict how well their final integrated products will perform. A study done by Tuinhout, Elzinga, Brugman, and Postma gives results on a example double-poly process. Results of this study show that the RMS mismatch ($\Delta C/C$) of 50umX50um capacitors is approximately 0.015% while for 200umX200um capacitors, it is 0.005%. [8] Feasibility analysis of high-precision ADC circuit performance is easier to perform given measured mismatch data.

Common-Centroid Layout

The thickness gradient is represented by the shading across the capacitors. The lightest shading is the thinnest oxide, while the heaviest shading respresents the thickest oxide.

Capacitors in common centroid configuration match better because the average of two capacitors across the gradient matches better than just single capacitors.

For example if:

C1A = 2pF, C1B=1pF
C2A= 1.5pF, C2A=1.5pF

$\frac{C1A+C1B}{C2A+C2B} = 1$ while $\frac{C1A}{C2A} = 0.67$.

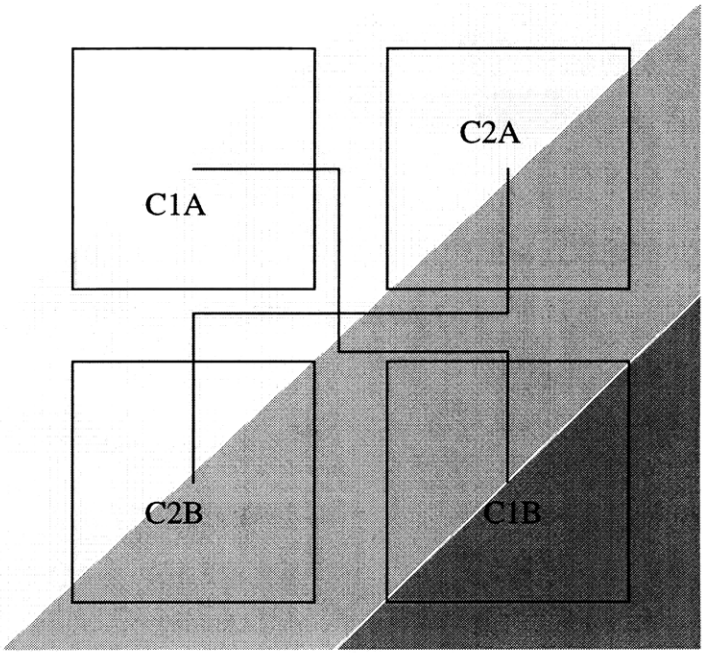


Figure 3.3

Capacitors also exhibit non-linear behavior. Capacitor non-linearity has caused numerous problems in high-precision linear and switched-capacitor circuitry. Capacitor non-linearity, unlike mismatch, is based on physical properties of the device. Non-linearity causes harmonic distortion and spurious behavior in systems manipulating dynamic signals. In most communication applications, distortion is intolerable.

Capacitors can be fabricated from polysilicon, substrate silicon, or metal. Each material exhibits a different degree of non-linearity. Capacitors constructed from polysili-

con and substrate silicon plates exhibit non-linear characteristics. Capacitor non-linearity is caused by voltage dependent charge induction on the silicon plates of integrated capacitors. Semiconductors deplete areas of charge at the dielectric-plate boundaries. The capacitance of the structure is non-linear because the voltage at the semiconductor-dielectric boundary varies non-linearly with applied voltage. This non-linearity is commonly referred to as a voltage coefficient. [9] The extent of the capacitor's non-linearity is dependent on the doping concentration of the silicon plate. Depletion regions are typically wider and cause higher non-linearity in more lightly doped silicon. In practice, it is possible to more heavily dope the substrate plate of these capacitors to flatten the capacitor non-linearity. Unfortunately, many processing techniques do not employ this extra doping step. Consequently, circuits employing capacitors made with substrate silicon plates have bad non-linear behavior. The non-linear effect in degenerately doping polysilicon is much smaller than that for lightly doped substrate silicon. [9] For this reason, degenerately doped polysilicon-oxide-polysilicon capacitor structures have become commonplace in linear switched capacitor systems. The effects of capacitor voltage coefficients on the ADC are discussed later in this chapter.

Capacitors made with metal plates exhibit more linear behavior. Charge induction occurs differently on metal capacitor plates. Unlike semiconductor plates, metal plates accumulate charge at the metal-dielectric boundary. No appreciable non-linearity in voltage from contact to dielectric occurs, making metal-metal capacitors linear. Proper controls over capacitor fabrication are necessary to implement these capacitors.

Parasitic capacitance also causes problems. Non-linear parasitic capacitance is common to both poly-poly and metal-metal capacitors. These capacitors sit atop an oxide dielectric with a substrate beneath, creating a parasitic MOS structure. Although capacitance from top plate to bottom plate is linear, the parasitic capacitance existing between bottom plate and substrate is still non-linear. In many circuits, non-linear parasitics cause minimal circuit non-linearity if voltages are carefully applied to the appropriate capacitor plates. In switched-capacitor circuits, bottom-plate sampling is used to minimize the non-linear parasitic error. By driving both input and reference voltages onto the bottom plate of an input capacitor, the effects of non-linear parasitic charge build-up can be decoupled

from the output of a switched-capacitor circuit block. The end of this chapter deals with the effects arising from capacitor mismatch and non-linearity. Methods for correcting the effects of mismatch and non-linearity are also discussed.

3.3 Integrated Resistor Errors

Resistors are used for multiple applications. Resistor strings are used in Flash ADCs to set the decision voltages for the Flash comparator array. Integrated resistors are also important when constructing continuous time active filters. Resistors suffer from the same lithographic problems that capacitors do resulting in resistor mismatch.

Resistors on silicon integrated circuits are commonly implemented with polysilicon lying on a thick field oxide. With the market pushing for lower power applications, large resistor values must be implemented for low power consumption. The value of a polysilicon resistor is given by,

$$R = \frac{(SR)L}{W} \tag{3.1}$$

where W, L, and SR are width, length, and sheet resistance respectively. Polysilicon sheet resistance typically ranges from 20-80 ohms/square. From the above equation it is apparent that minimizing the width and maximizing the length of the resistor will maximize resistance while minimizing chip area. Narrow devices are susceptible to lithographic sensitivity. For high value polysilicon resistors, small changes in width cause large changes in value. Silicon thickness, a contributing factor to sheet resistance, is not well controlled but in resistor matching situations silicon thickness error can be minimized using common centroid type techniques.

Resistors do have non-linear parasitic capacitance. As with integrated capacitors, polysilicon resistors form MOS capacitive structures with the substrate. The parasitic capacitance is non-linear and can cause problems in dynamic systems. The implementation considered in this thesis is immune to this capacitance. Resistors are used as DC components converting DC currents into Flash decision levels. The effects of parasitic capacitance are not present at DC and the analysis of non-linear effects is unnecessary.

The end of this chapter covers the effects of resistor mismatch on switched-capacitor pipeline ADCs. Methods for correcting the associated errors are introduced and discussed.

3.4 MOSFET Errors

As the primary active devices in CMOS integrated circuits, MOSFETs are the most commonly used integrated devices today. In switched-capacitor circuits, MOSFETs are used for more purposes than just amplification. Because MOSFETs make good switching devices, input switches, sampling switches, and charge routing switches are implemented using MOSFETs. Although used for a wide variety of applications in integrated circuit design, MOSFETs introduce a whole set of new problems to the circuit designer.

Much like integrated capacitors, MOSFET transistors suffer from width and length variations. Variations lead to changes in amplifier gain and input offset voltage. MOSFET transistors also suffer from mismatch of transistor threshold voltages. Threshold voltage difference adds directly to the input offset voltage caused by dimensional errors mentioned. The total input offset for a differential MOS input stage with current load shown in figure 3.4 is given by,

$$V_{OS} = \Delta V_{t(1-2)} + \Delta V_{t(3-4)}\left(\frac{g_{m3}}{g_{m1}}\right) + \frac{(V_{GS}-V_t)_{(1-2)}}{2}\left(\frac{\frac{-\Delta W}{L_{(1-2)}}}{\frac{W}{L_{(1-2)}}} - \frac{\frac{\Delta W}{L_{(3-4)}}}{\frac{W}{L_{(3-4)}}}\right) \tag{3.2}$$

and is typically on the order of tens of millivolts.[14] Gain is typically over-engineered and gain fluctuations in MOS transistors cause little concern. Offset voltages of millivolts can be potentially hazardous pipeline ADCs. Input offset is typically the worst effect of transistor mismatch. Offset affects are present in both residue amplifiers and Flash comparators.

New problems associated with MOSFETS appear when the devices are used as charge transmission switches. Charge injection from the MOSFET channels and voltage dependent channel resistance present problems in switched-capacitor signal acquisition circuits. Figure 3.5 shows a passive sample-and-hold structure consisting of a single tran-

Actively Loaded Differential Input Pair

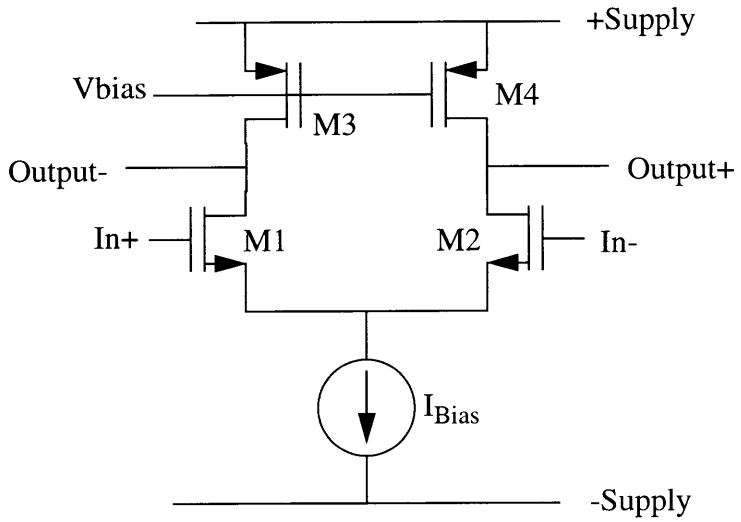


Figure 3.4

sistor and capacitor. When the switch is closed, the capacitor tracks the input voltage. When the switch is opened, the capacitor voltage is sampled. If the switch is opened quickly, approximately half the transistor channel charge flows freely onto the capacitor. The act of adding transistor channel charge to the sampling capacitor is referred to as charge injection. Injected charge causes the final capacitor voltage to be offset from the ideal sample voltage. A fixed offset in this case would not cause a problem. The problem in this case occurs because the charge injection onto the capacitor varies with the input voltage. Because Vin is attached to the switch source, the variation of gate-source voltage contributes a linear variation while the source-bulk voltage contributes a non-linear variation to the charge injection. In active switched capacitor systems, techniques are used to minimize the effects of the non-linear charge injection contributed by the input transmission switch. The charge injection above assumes that the switch opens very quickly at even intervals. If this is not the case, the switch timing can cause errors. If the switch closes slowly, the voltage on the capacitor may no be the same as the input voltage at the sampling instant. Also, if the sampling instant does not come at equal intervals, an effect called clock-jitter produces deformations in the sampled signal.

Simple Sampling Structure

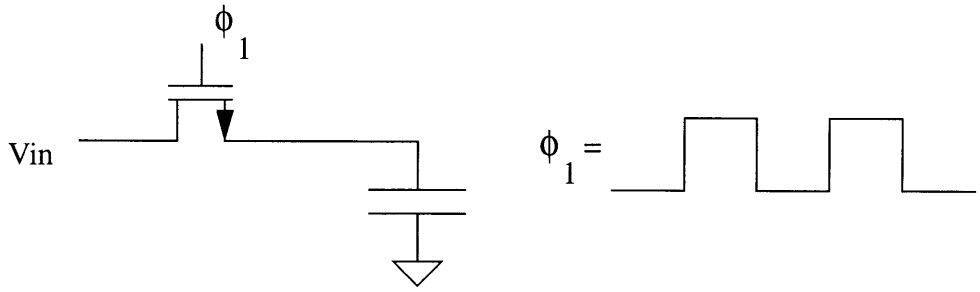


Figure 3.5

Dynamically driven MOSFET input switches produce sampling problems. The structure in Figure 3.5 has a finite bandwidth with a pole at,

$$p_1 = \frac{1}{R_{ON}C} \tag{3.3}$$

where R_{ON} and C are the channel resistance and capacitor value respectively. Pole, p_1, affects the magnitude of the input signal at high-frequencies. R_{ON} is dependent on the gate to source voltage of the transistor. As a result, the pole of the system is amplitude dependent causing this system, at high frequencies, to exhibit behavior similar to harmonic distortion. Large complementary switches usually remedy the problem. The pole frequency of the system can be increased by using a large switch. A complementary switch employs source and drain coupled NMOS and PMOS devices to reduce channel resistance variation. As a result, the system pole is no longer as amplitude dependent, and system distortion is reduced.

As described, physical non-ideality and lithographic error cause a number of problems in integrated circuit design. Linear and precision, low-offset circuits can be compromised greatly by these errors. As a precision linear circuit, the pipeline ADC is often compromised significantly by these errors. Methods of error cancellation must be employed in order to reap the benefits of the pipeline ADC.

3.5 Solving the Amplifier Offset Problem

As mentioned in the section on MOSFET error, transistor mismatch in the input

stage of a differential operational amplifier causes a large offset to appear between its input terminals. Amplifier input offsets manifest themselves at the outputs of active sampling structures. The effects of offset vary based on how the application employs the amplifier. Two amplifier applications used in pipeline ADCs, the residue amplifier and the Flash comparator, are considered below.

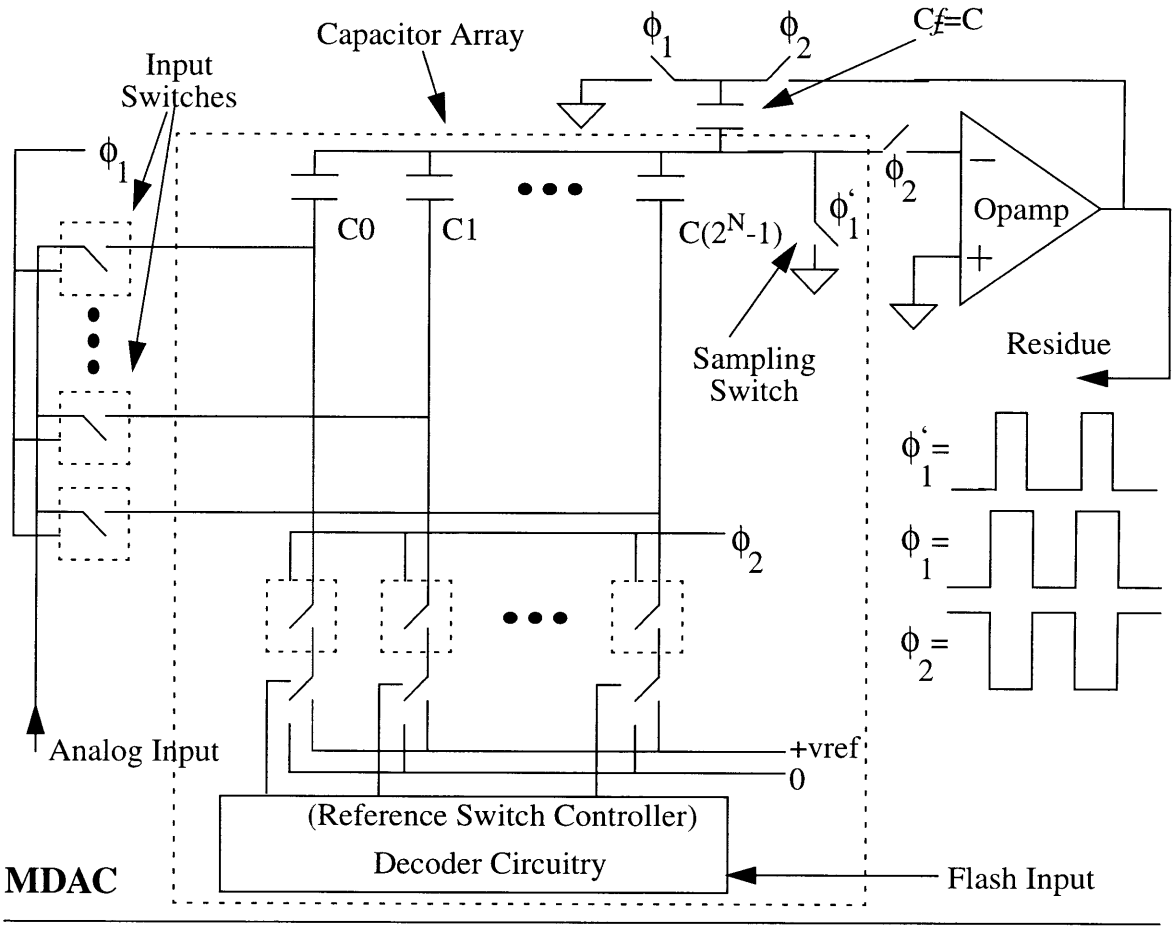
3.5.1 Offset Correction of the Residue Amplifier

The first amplifier application considered is the residue amplifier. In chapter two, a general description of pipeline ADC operation was presented. In switched-capacitor pipeline ADCs, the reconstruction DAC, SHA, addition block, and residue scaler described in chapter 2, can be constructed with a single amplifier, a capacitor array, and a switching network. This circuit is often called an MDAC (Multiplying DAC). The MDAC circuit is displayed in figure 3.6. The MDAC operates in the following manner. During the first phase of operation, the sampling phase, the input switches are closed allowing the input signal to be sampled onto the each of the 2^N capacitors in the input array. Another capacitor, the feedback capacitor, is sampled to ground. The sampling switch to ground is opened just prior to the end of phase one, injecting charge onto the capacitors in the array. The input switches are then opened at the end of phase one. After the sampling switch opens, the input capacitors have no DC path to ground and no charge is injected from the input switches. During phase two, the amplify phase, the feedback capacitor is attached between the inverting input and the output terminal of the amplifier. The capacitors in the array are attached between either a positive or negative reference and the inverting amplifier input. The positive or negative reference attachments are determined by the Flash's quantization of the input sample. Charge is redistributed through the system during phase two forming a residue. The ideal residue plot of the system is also shown in figure 3.6. Note that the first stage Flash ADC has 2^N decision levels unlike the Flash presented in chapter two. Given an input voltage ranging from ground to $+V_{ref}$, the residue of the MDAC in Figure 3.6 is given by,

$$V_{OUT} = 2^N V_{IN} - iV_{ref} + \frac{Q_{inj}}{C} + 2^N V_{OS} \tag{3.4}$$

where V_{OUT} is the residue, V_{IN} is the input sample, *i* is the quantization returned by the Flash, Q_{inj} is the charge injected by the sampling switch, C is the value of the feedback

Capacitive MDAC Sampling to Ground and Ideal Residue



RESIDUE

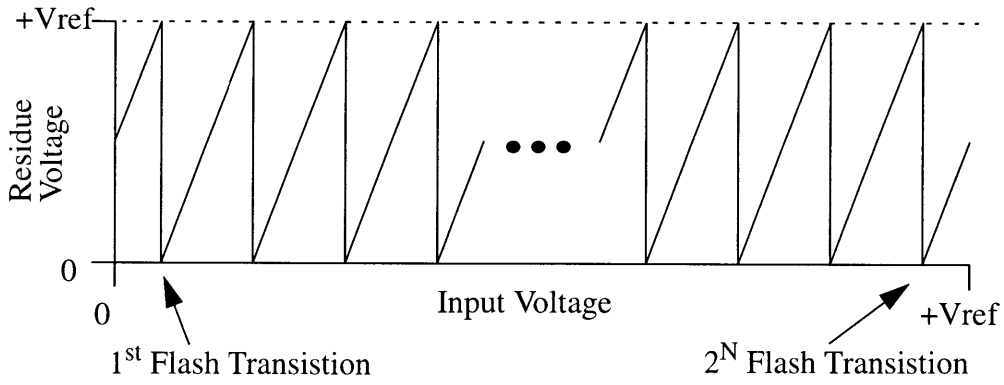


Figure 3.6

capacitor and each array capacitor, and V_{OS} is the amplifier input offset. Although the backend ADC has been designed with a small amount of extra correction range beyond the fullscale input range, if the offset in (3.4) is large enough, it could push the residue of the stage outside this correction range. The affected residue can be seen in Figure 3.7. This mismatch in ranges leads to problems. When the offset affects shown in Figure 3.7 actually occur, for either positive or negative offset, the residue lying outside the next stage's input causes an over-range condition. In an over-range condition, all the outlying residue is quantized to the highest or lowest backend ADC code dependent on which reference point was over-ranged. Ideally, a residue spans the fullscale of the backend exactly. Assuming all of the array capacitors are matched and gain error does not exist, when one reference is over-ranged, the other reference is under-ranged. When an under-range condition occurs, the quantization codes not spanned by the previous stage's residue are never reached. These codes are referred to as "missing codes." In the overall transfer function of the ADC, over-range conditions show up as horizontal segments, and "missing codes" show up as vertical discontinuities. These discontinuities can be seen in Figure 3.7. If residue amplifier offsets can be reduced so that residues no longer fall outside of backend correction range, then the linearity of the transfer function can be preserved. The only result of these small offsets will be an input referred ADC offset. A method known as offset cancellation can be used to reduce input referred converter offsets drastically.

By making some simple changes to the MDAC in Figure 3.6, the MDAC shown in Figure 3.8 can constructed. This new MDAC reduces input offset effects at the MDAC output. This technique is called offset cancellation. The circuit in Figure 3.8 works in the following manner. During phase one, phi1, the input voltage is sampled onto the bottom plates of the 2^N capacitors in the input array while ground is sampled onto the bottom plate of the feedback capacitor. In the new scheme, the residue amplifier is in unity feedback causing the top plate of each capacitor to track the amplifier's input offset voltage. Just prior to the end of phase one, the sampling switch opens injecting charge at the amplifier's summing node. The input switches then open at the end of phase one. During phase two, phi2, all connections are made as they were in the previous implementation. The new

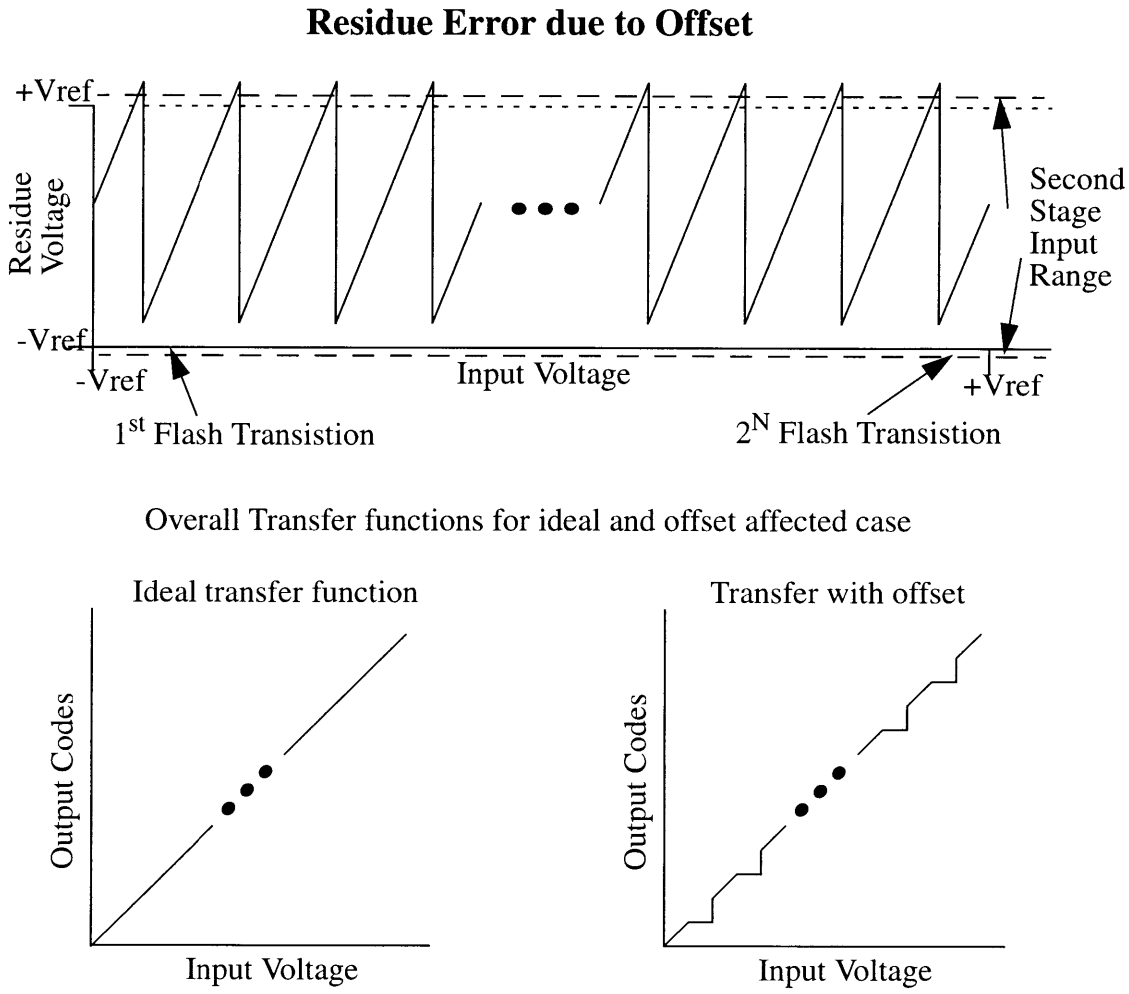


Figure 3.7

output residue is given by,

$$V_{OUT} = 2^N V_{IN} - iV_{ref} + \frac{Q_{inj}}{C} + \frac{1}{A+1} V_{OS} \tag{3.5}$$

where V_{OUT}, V_{IN}, Q_{inj}, C, and V_{OS} are the same as for (3.4) and A is the open loop DC gain of the residue amplifier. The effect of offset has been decreased greatly, and the only residual effect is very small. On the other hand, the charge injection offset remains. By making the MDAC fully differential as shown in figure 3.9, charge injection can be reduced. In a differential configuration, the same amount of charge is injected to both summing nodes if the two sampling switches match reasonably well. As long as the amplifier provides adequate common-mode rejection, a common trait to most differential amplifiers,

Offset Cancelling MDAC and Associated Timing

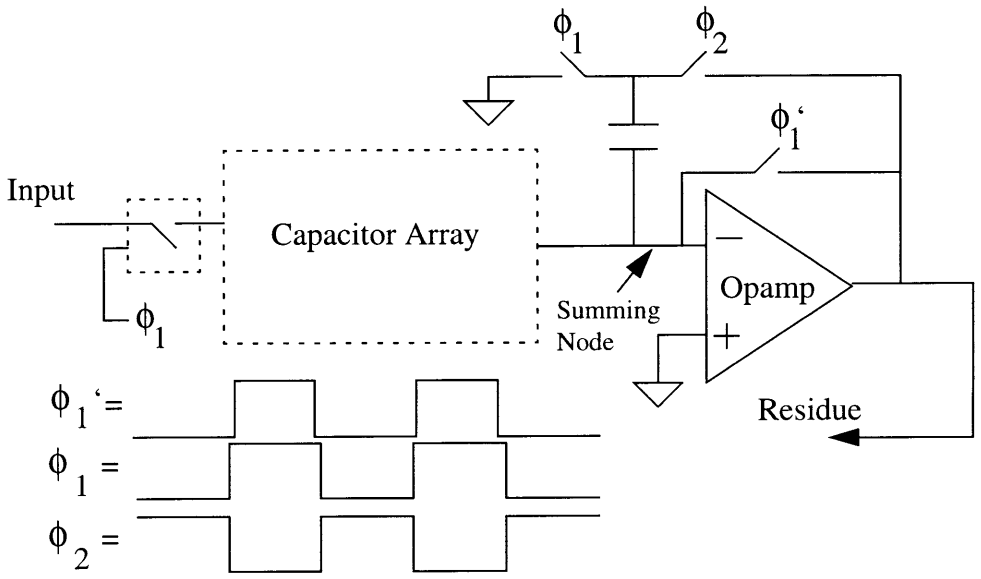


Figure 3.8

the effects of charge injection are cancelled. Given a differential input where each side swings between +Vref and -Vref, the new stage residue is given by,

$$V_{OUT} = 2^N V_{IN} - iV_{ref} + (2^N - i)V_{ref} + \frac{Q_{inj1} - Q_{inj2}}{C} + \frac{1}{A+1}V_{OS} \qquad (3.6)$$

where Q_{inj1} and Q_{inj2} match as well as the two sampling switches.

Equation (3.6) shows that the effects of input offset and sampling switch charge injection have been effectively cancelled. Small offsets still exist, but only cause problems in very high resolution ADCs.

3.5.2 Correcting Offset Error in Flash Comparators

Flash ADCs quantize the input sample taken in each pipeline stage. Much like the Flash ADC architecture presented in chapter 2, the pipeline ADC Flash is comprised of a resistor string which sets threshold voltages and an array of parallel switched capacitor comparators. As also mentioned in chapter 2, the Flash ADC is not immune to the problems of amplifier input offset. The following section deals with correction of Flash ADC errors.

Offset Cancelling Differential MDAC and Associated Timing

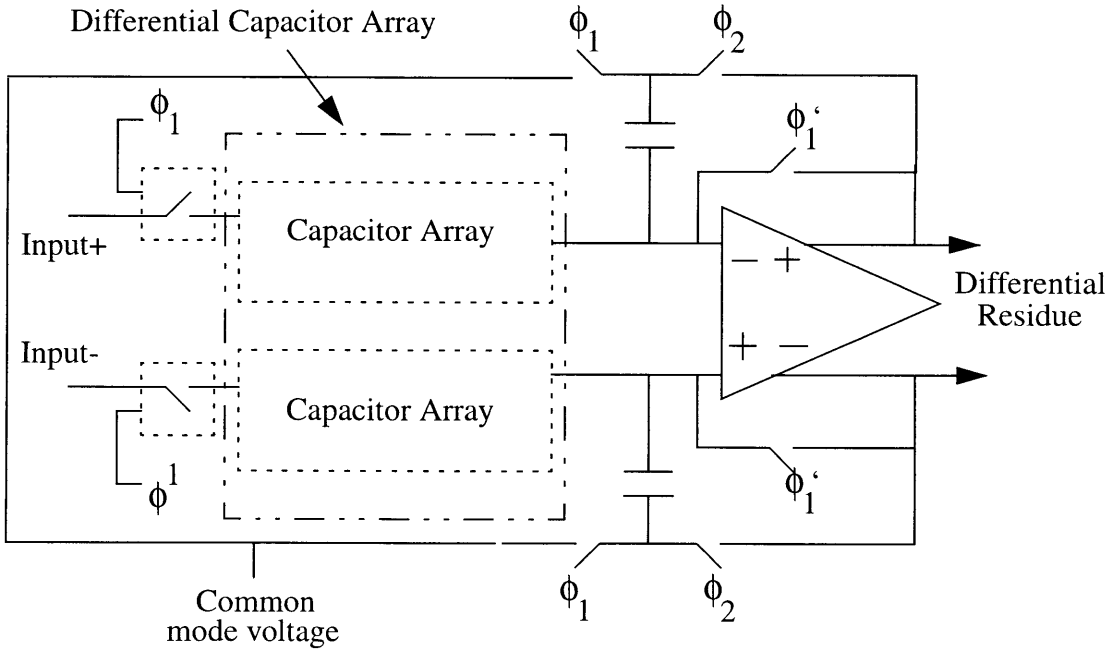


Figure 3.9

Figure 3.10 shows a differential switched capacitor comparator, consisting of two input capacitors, a switched capacitor network, a high gain open loop amplifier, and a latch. The comparator operates in the following manner. During phase one, phi1, a differential voltage representing the comparator decision level is sampled across the two input capacitors. The top plates are attached to the common-mode voltage of the system by a set of sampling switches. Prior to the end of phase 1, the sampling switches open injecting charge onto the two input capacitors. The effect of this charge injection is reduced due to the differential structure of the circuit. At the end of phase one, the decision level switches are opened and the input switches are closed. During phase two, phi2, the comparator input tracks the input voltage minus the decision level, and the output gives the corresponding decision of the comparator. Just before the end of phase two, the latch acquires the comparator output decision. When an offset exists between the inputs of the amplifier, the decision level of the comparator is shifted by $-V_{off}$. As mentioned in chapter 2, comparator decision levels can also vary from their ideal values due to resistor mismatch. Comparator offsets and resistor mismatch cause DNL and consequently INL in each Flash ADC.

Decision level deviation has serious consequences in pipeline ADCs. Even spacing of decision levels is necessary so that the residue of the associated stage spans exactly the next stage input range. Uneven threshold voltage spacing leads to over-range and under-range conditions as seen in Figure 3.11. One method of correcting this problem is to add more input range to the next pipeline stage. The input range of the following stage can be effectively increased by decreasing

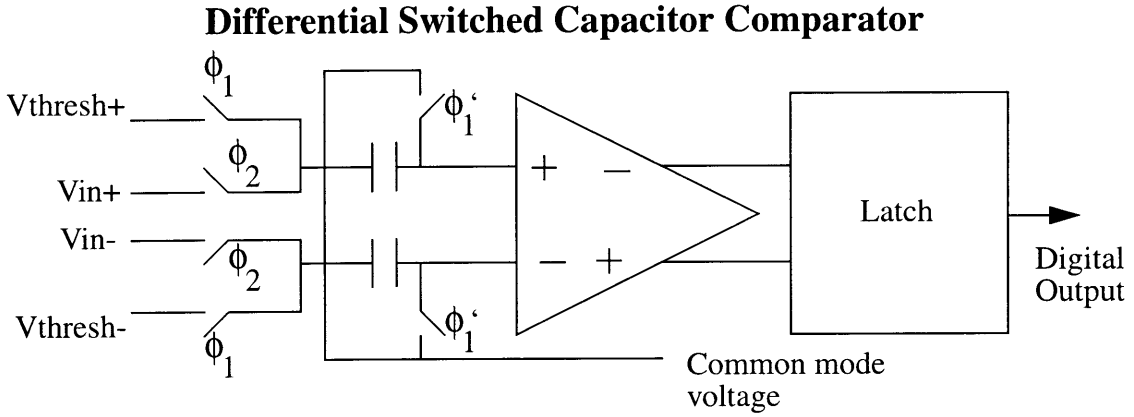
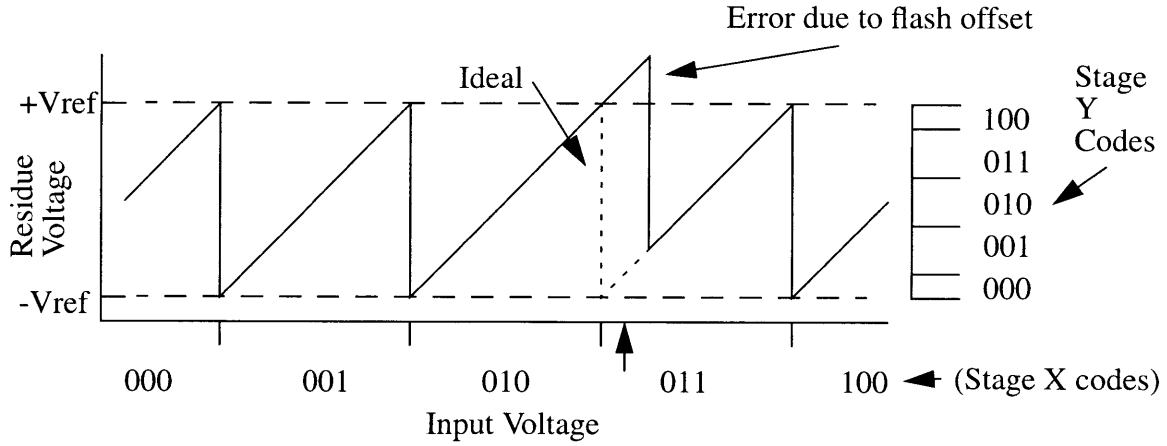


Figure 3.10

the closed loop gain of the current stage. With extra range, over-range conditions do not occur unless large circuit errors are present. The system can now "tolerate" decision error of the Flash ADC. Another result is that the resolution of each stage is reduced by one bit. This bit is used for "Digital Error Correction."

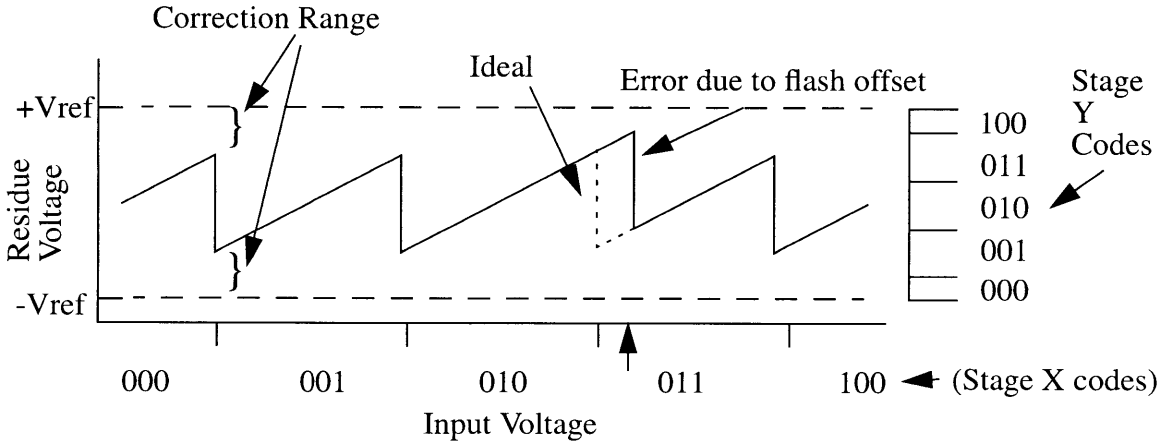
A pictorial view of digital error correction is shown in Figure 3.11. Stage X incorrectly quantizes an input signal to a value that is too low. The incorrect quantization shifts stage X's residue into stage Y's extended input range. Stage Y quantizes the residue as well as measuring the quantization error made by Stage X. The bits returned by stage Y are then added to the stage X bit in an overlapping manner to form a corrected digital output code free from decision level error. Digital error correction is a method of "tolerating" Flash error and residue amplifier offset error. For instance in a five bit stage with a differential input range of five volts, ADC threshold points are 156.25 mV apart. With a zero offset residue amplifier, a five bit pipeline ADC stage can tolerate ±78 mV of Flash error before over-ranging. Stages which sample continuous time signals typically need more error tolerance due tracking differences between the MDAC and Flash. By reducing the

Pictorial View of Digital Error Correction



In the picture above for the non-error corrected two-stage configuration, the arrow pointing to the bottom line represente the input voltage sample. Each of the two stages has two bit resolution. The input sample is quantized incorrectly due to the flash offset error, and the residue over-ranges the following stage. The difference between the ideal code and what iscalculated is given by,

Ideal: 01100 + 00001 = 01101 With Error: 01000 + 00100 = 01100



In the second picute, the gain of the first stage has been reduced so that the system possesses some correction range. Now if digitial error correction coding is done on both ideal and errored residues, the results are given by,

Ideal: 0110 + 0001 = 0111 With Error : 0100 + 0011 = 0111

which are similar unlike the original case.

Figure 3.11

offsets of each comparator, more tolerance can be opened up for continuous time sampling. Offset cancelled sampling comparators are employed to reduce comparator offset

error.

Figure 3.12 shows a differential offset cancelled sampling comparator. It consists of two amplifiers with small gain, a set of input capacitors, a set of offset capacitors between the two amplifiers, a switching network, and a latch. During phase one, phi1, a differential decision level voltage is sampled onto the input capacitors. The difference between the output referred first stage offset and second stage input offset is sampled onto the offset sampling capacitors. Prior to the end of phase one the four sampling switches are opened injecting charge onto the input and offset capacitors. Once again the charge injection effects are reduced. At the end of phase one the decision switches are opened. At this moment the output is equal to the second stage offset. The new input referred offset is given by,

$$Voff = \frac{1}{A_1(A_2+1)}V_{OS2} + \frac{\Delta Q_{inj1}}{C_1} + \frac{\Delta Q_{inj2}}{(A_1)C_2} \tag{3.7}$$

where A_1 and A_2 are the amplifier gains and V_{OS2} is the second amplifier's offset. Equation (3.7) does not have V_{OS1} as a term because this offset is stored and cancelled by the capacitors C_2. During the second phase of operation the input voltage is applied across the input capacitors, and the latch returns the appropriate state. Now that comparator decision level is accurately represented, the resistor string mismatch and charge injection now constitute the bulk of the Flash error. Resistor string error is typically quite small.

Differential Offset Cancelled Sampling Comparator

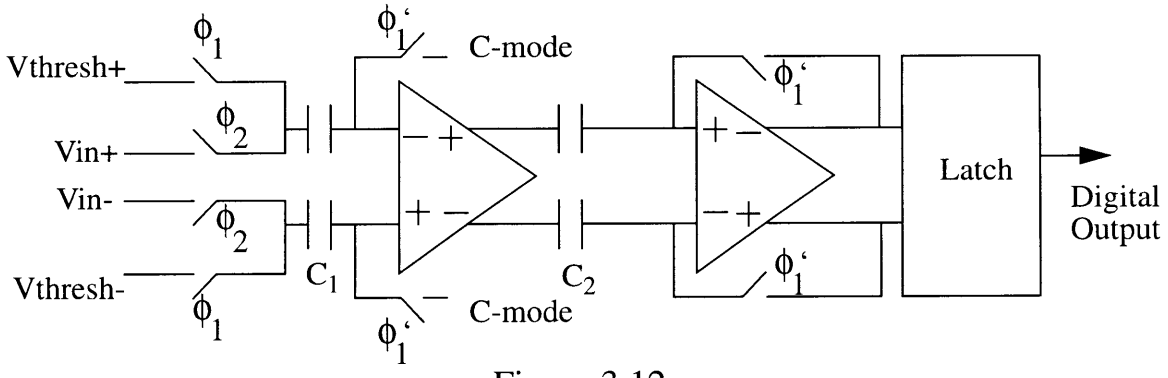


Figure 3.12

3.6 Capacitor Error Effects and Calibration

The effects due to capacitor mismatch are not as easy to remedy as were amplifier offset effects. Simple circuit techniques are not sufficient to correct the errors that capacitor mismatch introduces to high resolution pipeline ADCs. Capacitive elements lie in the main analog path of the pipeline ADC, corrupting analog signals directly. Unlike comparator decision and resistor string error, capacitor error can not be represented digitally until a complete conversion cycle has been completed, rendering a digital error correction type technique ineffective. Capacitor mismatch currently places the largest constraint on pipeline ADC linearity.

The capacitor array, as shown previously by Figure 3.6, is the main charge transfer block in the pipeline ADC. With an ideal set of capacitors, an MDAC will have a residue which spans the exact input range of the next pipeline stage. As mentioned in the discussion of Flash ADC errors, the distance between successive DAC transitions (vertical discontinuity in the residue plot) in any given stage is governed by the Flash ADC. On the other hand, the size of each DAC transition is based on the ratio of the array capacitor associated with that transition and the feedback capacitor of the MDAC. The gain of the MDAC stage is also affected by capacitor mismatch error. If each array capacitor was given an error a_m, then (3.6) would become,

$$V_{OUT} = \left(2^N + \sum_{m=1}^{2^N} a_m\right)V_{IN} - \left(i + \sum_{n=1}^{i} a_n\right)V_{ref} + \left(2^N - i + \sum_{l=i+1}^{2^N} a_l\right)V_{ref} + \frac{Q_{inj1} - Q_{inj2}}{C} + \frac{1}{A+1}V_{OS} \tag{3.8}$$

The residue has been corrupted and it is important to know how this error affects the digital output of the ADC.

Transition height variations and gain errors can be seen directly from the residue of the associated pipeline stage. DAC transition size is governed by gain error and random mismatch. Gain error adds uniform error to each of the DAC transitions made in the MDAC. Capacitor mismatch within the capacitor array adds random size variations from

one transition to the next. The effects of gain error and combined gain and mismatch error can be seen in Figure 3.13. Capacitor errors cause over-range and under-range conditions in ADCs which are not digitally error corrected. By using digital error correction, true over-range conditions will disappear. The specialized case of digitally error corrected (DEC) pipeline stages is discussed from this point on. An under-range condition for a DEC stage can now be defined as a DAC transition smaller than 1/2 the next stage input range. Under-range conditions will cause "missing code" vertical discontinuity in the ADC transfer function. In a case when capacitor error is very small, all the back-end codes are reached. The highest and lowest codes are not spanned completely by the residue in question and the result at the output is a "narrow code." A "narrow code" appears as a

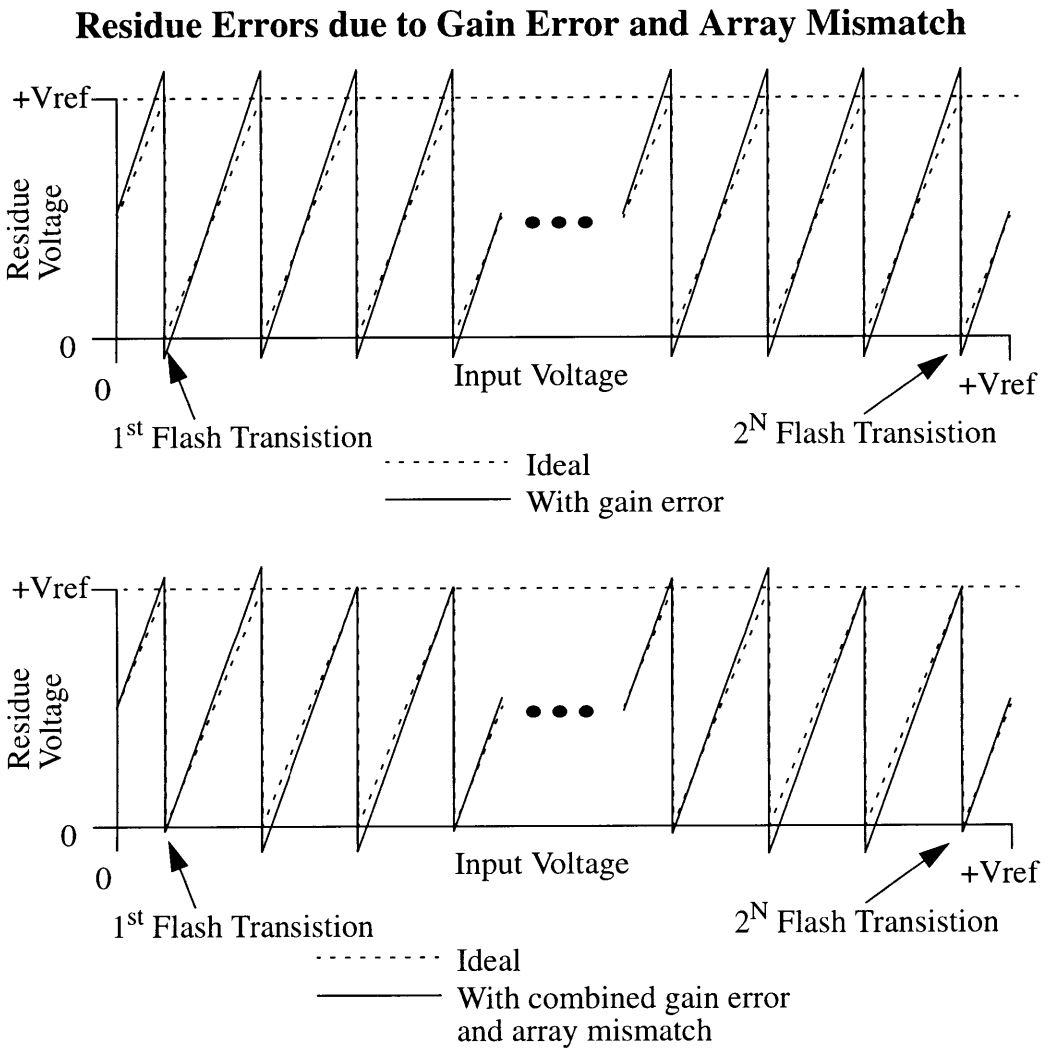
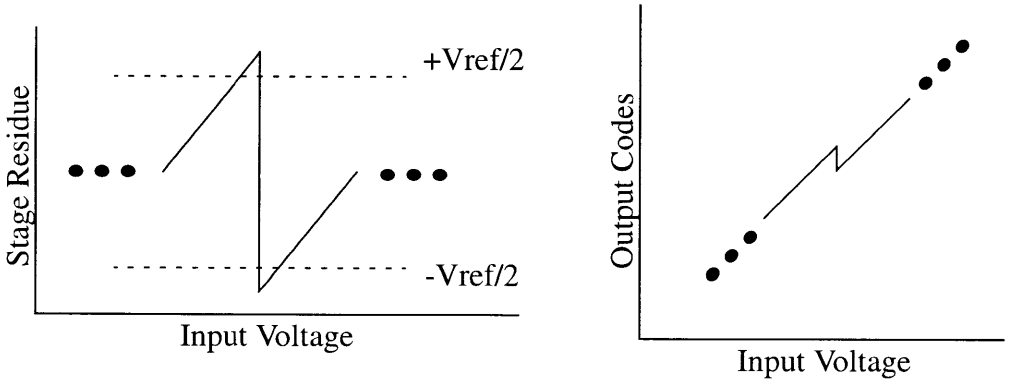


Figure 3.13

DNL point lying between -1 LSB and 0. When using digital error correction, an over-range condition can be defined as a DAC transition larger than 1/2 the next stage's input range. The digital error correction technique allows more than one way to arrive at the same output code. When DEC is combined with an over-range condition, some back-end ADC codes are reached more than once and "extra code" non-monotonicity results. When the errors are very small, the highest and lowest back-end ADC codes are spanned more than an ideal amount by the stage residue causing "wide codes" to appear. Wide codes show up as DNL points lying between 0 and +1 LSB. The effects of under-ranging and over-range can be seen in Figure 3.14. Missing codes and non-monotonicity are not acceptable in ADC transfer functions because they compromise the ADC's linearity. Wide code and narrow code widths must be kept close to the ideal step width in order to preserve linearity.

Over-range and Under-range Effects on Residue and Transfer Function

Residue and Transfer function effect for "Over-range"



Residue and Transfer function effect for "Under-range"

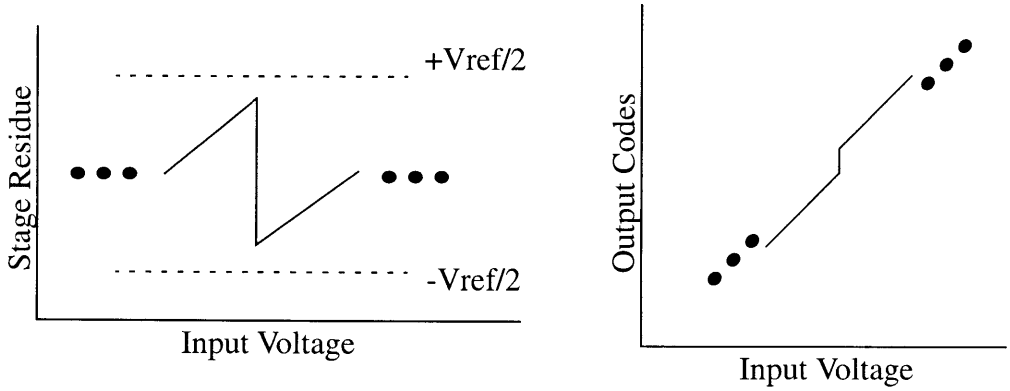


Figure 3.14

For lower resolution multibit/stage ADCs (under 12 bits) with less stringent constraints on linearity, capacitor matching levels are adequate and no correction is necessary. It is important to solve the problems associated with capacitor mismatch when designing ADCs of the 14 and 16 bit variety having reasonable linearity.

3.6.1 Calibration

When calibrating a part, a set of measurements are taken and later used to correct the errors of the system. For precision components, calibration can be done in a number of ways. Early calibration of pipeline ADC circuits was inherently analog. Techniques involved either analog circuit design tricks or the direct alteration of physical components. In a 14 bit pipeline ADC designed by Mercer at Analog Devices Inc., the two types of calibration techniques are employed together. This current-mode ADC employes laser-trimmable thin-film resistors as DAC elements. Matching within the DACs is achieved by trimming the resistors. Residual gain error associated with the DACs is corrected by scaling the input ranges of the pipeline stages. [5] Although an accurate method of matching precision components, laser-wafer trimming is very slow and expensive. In pipeline ADC's it is also possible for calibration to be done by measuring digital errors produced in the analog front-end. These digital errors can simply be added to or subtracted from the ADC's raw digital output. By doing this, ADC linearity can be corrected to a high-degree of accuracy.

In 1992, Karanicolas approached the problem of digital calibration using a very simple circuit. Figure 3.15 shows a single bit per stage module and its ideal residue plot. Karanicolas proposed that reducing the amplifier closed loop gain, shrinking the output residue range, and creating a buffer zone for error absorption could be used to eliminate over-range conditions. This proposed buffer zone can be seen in Figure 3.16. Using offset cancellation techniques in both Flash and MDAC blocks, Karanicolas kept offset errors to a minimum. Residual offsets, charge injection effects, and mismatch error were then absorbed into the buffer zone. The stage gain was made small enough so that all errors fell within the range. This architecture allowed only under-range conditions. Under-range conditions could be corrected digitally.

Single Bit Pipeline Stage and Residue

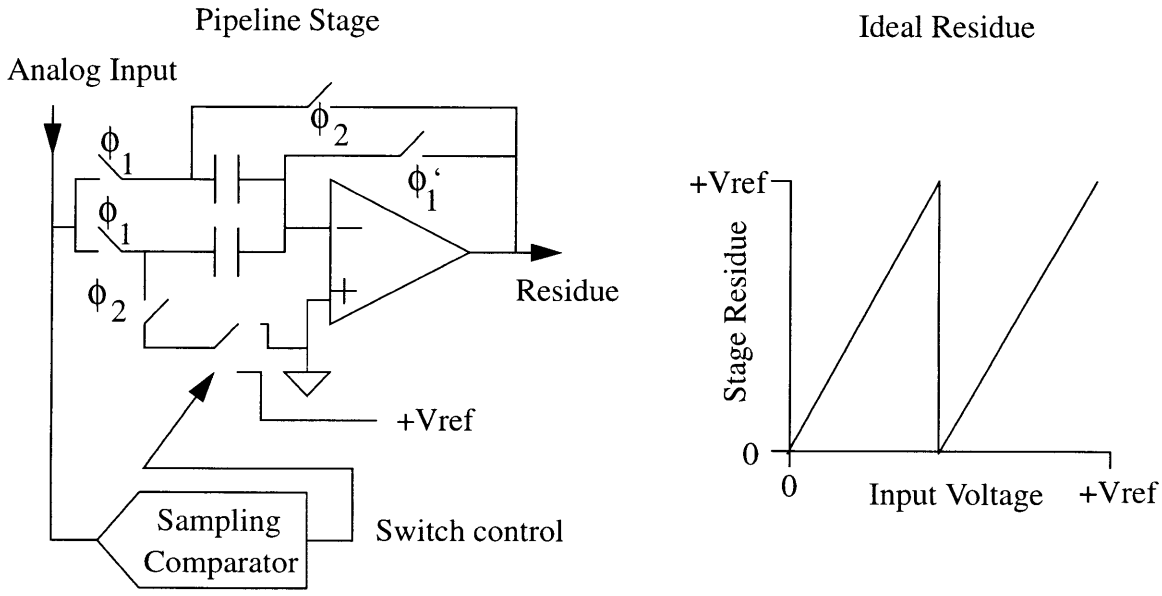


Figure 3.15

In a pipeline ADC utilizing this approach, only a certain number of stages are designed with gain reduction. Upon calibration, the final gain reduced stage is examined first. The size of the DAC step is measured using the back-end ADC. The back-end is either reasonably accurate in the case of the first measurement, or a calibrated pipeline ADC itself. Calibration is performed by forcing the stage under calibration into certain modes. During phase one of each mode, the mid range input voltage is sampled onto the input capacitors. During phase two of mode one, the DAC transition is not applied. In mode one, the output residue is equal to point A as seen in Figure 3.16. During phase two of mode two, the DAC transition is applied. In mode two, the output residue is equal to point B. Points A and B are quantized by the back-end ADC. Their values are stored in memory as X1 and X2 respectively. The digital algorithm used on the calibrated stage during normal mode operation and front-end calibration is given by,

$$C = \begin{cases} Y & \text{Bit=0} \\ Y + (X1 - X2) & \text{Bit=1} \end{cases} \tag{3.9}$$

where C is the digital code produced by this stage and the back-end, and Y is the back-end

quantization of the stage's residue. The algorithm matches the top quantization level of the first residue section with the bottom quantization of the second residue section. By matching these codes, the algorithm corrects the DNL error associated with the uncalibrated transition. The freshly calibrated stage and back-end stages now form a reasonably accurate ADC for calibrating the upper pipeline stages. This calibration scheme is continued stage by stage until the first stage of the pipeline is calibrated. Karanicolas was able to achieve DNL of less than ±1/4 LSB at 15 bits with his digital calibration scheme. [10]

Residue of the Karanicolas Single Bit Stage

Gain reduction leads to buffer zones at top and bottom.

This residue plot is not to scale and buffer zones are typically much smaller.

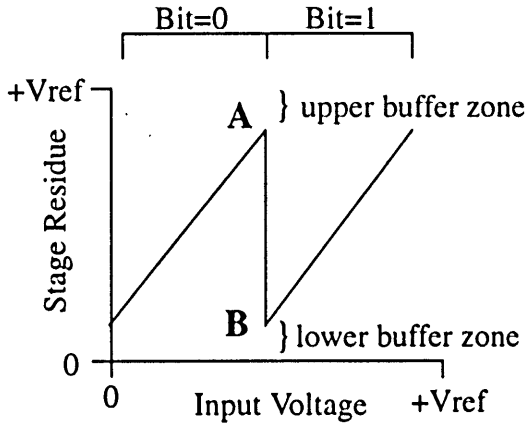


Figure 3.16

The calibration scheme mentioned above does have drawbacks. Pipeline ADCs often have a power-of-two signal gain from input to output. Each calibrated stage in a Karanicolas converter must have reduced gain in order to provide a sufficient buffer zone. Gain reduction makes it necessary for extra stages to be implemented in order to provide enough signal gain through the ADC. Extra stages take up extra area and burn more power. The Karanicolas converter usually runs more than ten stages with reduced gain causing a non-power-of-two ADC gain and a larger than normal gain error. An on-board multiplier is used to correct the converter gain error. Digital multipliers are noisy due to the large amount of digital switching they do. This noise couples to the analog circuitry through the substrate. Added noise to the analog signal path degrades an ADC's noise performance. Attempting to limit coupled noise, National Semiconductor designed new clocking schemes on their 16 bit single-bit per stage pipeline ADC. [6]

The only INL which is corrected by the Karanicolas calibration algorithm is that produced by the random DNL present in the transfer function. As the converter DNL is reduced, so will the associated INL be reduced. Unfortunately, random DNL is not the only cause of system INL and DNL correction methods do little to correct for non-DNL related INL error. These other sources of INL manifest themselves as non-linearities in the analog signal path of the pipeline ADC. In switched capacitor systems, capacitor voltage coefficients, non-linear amplifier gain, and non-linear charge injection contribute to the non-linearity of the ADC transfer function.

Capacitors, as mentioned in section 3.2, possess voltage coefficients which are directly related to their physical parameters. In [9], McCreary measured the voltage coefficients of a number of differently doped silicon-silicon capacitors. These measurements showed that more highly doped capacitor plates (N_D=1.5x10^{20}/cm^3) gave relatively small, linear voltage coefficients (~8ppm/V linear) while more lightly doped silicon (N_D=9x10^{18}/cm^3) gave relatively large, non-linear voltage coefficients (~250ppm/V average over curve). McCreary's plots can be found in Figure 3.17. With voltage coefficients

McCreary's Plots of Voltage Coefficients

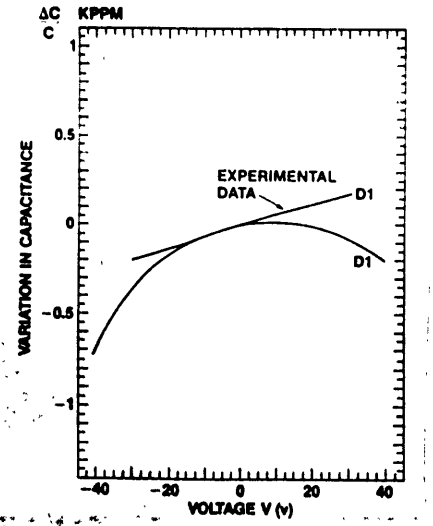


Fig. 8. Calculated and measured *CV* curves for a poly-to-Si capacitor *D1* with N_d(poly) = 1.5 × 10^{20}/cm^3 and N_d(Si) = 1.1 × 10^{20}/cm^3.

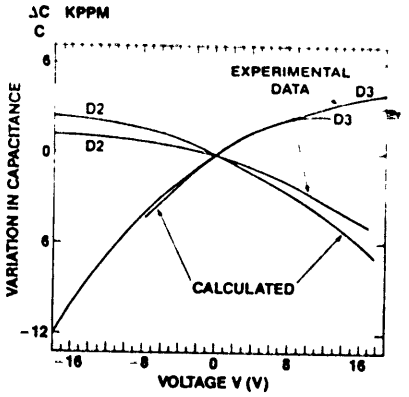


Fig. 9. Calculated and measured *CV* curves for poly-to-Si capacitor *D2* with N_d(poly) = 9 × 10^{18}/cm^3 and N_d(Si) = 1.65 × 10^{20}/cm^3, and curves for poly-to-Si capacitor *D3* with N_d(poly) = 1.2 × 10^{20}/cm^3 and N_d(Si) = 5.5 × 10^{18}/cm^3.

Figure 3.17

present in ADC array capacitors, with an increase in voltage, an increased amount of charge would be redistributed in each stage as the input voltage increased causing the out-

put residue to change. The overall ADC transfer function would take on the shape of the non-linearity in the voltage coefficient. In a fully differential pipeline ADC with common-mode biased array capacitors, McCreary's voltage coefficient curves would produce an INL plot shown in Figure 3.18. One way to resolve the INL problems due to voltage coefficients is to use more linear capacitors when manufacturing these circuits. Highly doped double-poly capacitors or metal-metal capacitors would be best to reduce converter INL.

Reside amplifier gain non-linearity can also be a problem to system INL. Each amplifier must have a linear transfer function across it's entire output range. Frequently open-loop gain will degrade non-linearly toward the upper and lower ends of this output range due to output stage transistors operating near the triode region. On the other hand, if the amplifier gain is designed to be more than accurate at the limits of the output swing while having higher gain in the middle of the range, the non-linearity will not contribute more than a fraction of an LSB to the converter INL.

INL Due to Voltage Coefficients in Figure 3.17

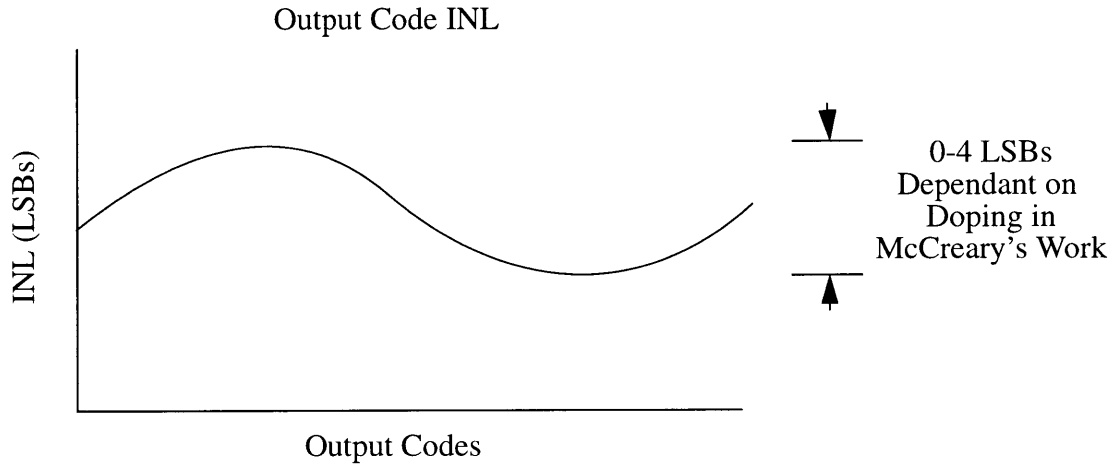


Figure 3.18

Charge injection may also be a problem in high resolution pipeline ADCs. As mentioned earlier in chapter 3, the input switch of a sampling structure adds a non-linear voltage dependent charge injection to the input capacitor. In the sampling structure shown in Figure 3.9 the input switch contributes little, but the little it does contribute is non-linear. The sampling switch may also cause problems. The charge dumped by this sampling

switch could possibly be affected by the non-linear impedance of the input switch. This non-linear contribution is not yet known, but if it is necessary, it could be decreased by decreasing the sample switch size or the input switch sizes.

INL is commonly corrected by matching codes. Highly linear external calibration DACs are used to input an analog signal into a pipeline ADC. The calibrated output code of the ADC is then compared to the DAC input code and the discrepancy in stored on chip in a ROM. These discrepancies are then corrected as codes reach the output of the ADC during normal operation. [6] This method of INL calibration is performed using external circuitry and a large internal memory and tends to increase die area. The proposed ADC currently deals only with DNL calibration. The calibration algorithm is discussed in more detail in chapter 4.

For multibit per stage pipeline ADCs, digital calibration schemes have been proposed. Karanicolas generalizes his calibration scheme to multibit bit per stage pipelines in his thesis; yet gain reduction still exists. Digital calibration must be preformed without the need for gain correction in order to better noise performance. Seung-Hoon Lee presents a simple calibration technique for binary-weighted two-step ADCs. [11] A different form of calibration, based on the ideas of both Karanicolas and Lee is presented in chapter 4. This calibration method along with the other error correction techniques described above are used to create a high-resolution pipeline ADC. The next chapter presents the architectural background of the proposed ADC and how the architecture can be used to achieve good linearity.

Chapter 4

Architecture and Multibit per Stage Calibration

4.1 Introduction

Achieving high linearity in pipeline ADCs is very important. Reducing the complexity of any calibration on chip is equally important. The first step toward achieving these goals is selection of an appropriate architecture. Architecture should provide solutions to the two issues above as well as the issues of low-power dissipation, noise-performance, ability to correct Flash and offset errors, and ability to perform all ADC functionality. This chapter deals with the process of selecting an architecture for the proposed 16 bit pipeline ADC.

4.2 Power management, Noise, and Error Correction in Pipeline Architectures

Whether the issue is lengthening battery life or low temperature operation, most ADCs today are designed to dissipate minimal power. ADCs which utilize circuit components which draw less current, lower power amplifiers, and a smaller number of comparators will naturally dissipate minimal power. Single bit per stage pipeline ADCs require only a power of two multiplication and on comparator per stage. Residue amplifiers in

these ADCs often draw much less current than the higher gain-bandwidth amplifiers implemented in multibit per stage pipeline ADCs. The single comparator per stage also dissipates much less power than larger multibit comparator arrays found in multibit per stage ADCs. There is on the other hand a trade-off. Multibit per stage ADCs implement much fewer residue amplifiers and the extra comparators used burn relatively little power in comparison to the residue amplifiers. For low speed applications, multibit and single-bit per stage ADCs can be design to dissipate similar amounts of power although the single-bit per stage ADC will typically win in the speed vs. power arena.

Good ADC noise performance is necessary for achieving maximum dynamic range in specific applications. In single bit per stage ADCs, the converter noise is generated by a SHA and the first few stages of the pipeline. As will be shown in chapter 5, increasing the input capacitance of a stage will decrease the ADC noise levels. To increase the noise performance of a single bit per stage ADC, the input capacitances of the first few stages must be kept large. These capacitor sizes impose a power restriction on the first few residue amplifiers of the ADC and increase the power necessary to run the ADC properly. In a multibit per stage ADC, the noise performance is governed by the SHA and first stage primarily. Large capacitors are still needed on the SHA and first stage to ensure noise performance, but smaller input capacitors can be used on the second and subsequent stages of the ADC. The power constraints on the residue amplifiers in these ADCs are hence loosened and become more lax than those of the single-bit per stage ADC. As a result, power can be reduced with better noise performance in multibit per stage ADCs.

One of the key features of digital error correction in multibit per stage ADCs is its ability to tolerate residue amplifier offsets and Flash decision errors reducing the frequency of over-range conditions. These errors occur in both multibit and single-bit per stage pipelines. Single bit per stage ADCs, on the other hand, can not tolerate these errors without specialized design considerations. Multibit per stage ADCs employing digital error correction require fewer design considerations when it comes to total error correction than do single bit per stage ADCs. This advantage along with its noise performance capabilities and low-power possibilities makes the multibit per stage architecture the choice for the proposed converter.

4.3 Previous 16 bit Architectures

Achieving 16 bit resolution and linearity at high-speed is no easy feat. A number of architectures have developed in order to solve the problem. As mentioned above, in 1996, National Semiconductor has developed a 20 stage pipeline using the Karanicolas calibration approach. A number of front end stages are calibrated and good linearity is obtained. Designed using a single bit per stage architecture, the part operates with both low-power consumption (200mW) and high speed (1Ms/s). As previously mentioned, the analog section of National's part suffers from digital noise pollution.[6]

Analog Devices Inc., has developed a 16 bit ADC combining a Sigma-Delta front-end modulator with a back-end multibit per stage pipeline ADC. [4] The Sigma-Delta front end oversamples the input signal, performs a multibit quantization, and shapes the quantization noise. The quantization noise is then measured by a twelve bit back-end pipeline ADC and subtracted from the front end output to achieve 16 bit resolution. The converter is highly linear over an input signal bandwidth of 1.25MHz and has excellent dynamic performance. As a dynamic system, the Sigma-Delta configuration is not useful for converting individual samples, hence the part can not perform all useful ADC functionality. Datel has also come out with other hybrid architectures useful in 16 bit applications. These ADC's currently mark the best the market has to offer in fast, 16 bit ADCs.

4.4 Achieving High Resolution without Calibration

Linearity in a pipeline ADC is governed by the stages that produce the most influential analog signal path errors. Each ADC stage adds a certain power-of-two signal gain to the input. Consequently, when output referred, errors occurring toward the front of the pipeline have the most signal gain applied to them. The first few pipeline stages typically cause the largest linearity problems. The number of ADC pipeline stages along with their respective resolutions, in the end, govern the linearity of the ADC.

As mentioned in chapter 3, capacitor mismatch in MDAC arrays adds error directly to the analog signal path. Capacitor mismatch in a pipeline ADC is commonly measured in bits of accuracy. If capacitor mismatch is said to be "ten bit good", the follow-

ing statement is true: Any DAC transition error due to capacitor mismatch in a pipeline MDAC will not generate a "missing" or "extra" code in the output transfer function unless the MDAC's residue is quantized to greater than ten bit accuracy by an ideal back-end ADC. A non-linearity arises when an incorrectly sized DAC transition is subtracted in any pipeline stage. If the error of a single array capacitor, C_i, is given as a_i, the i^{th} DAC transition of any N-bit MDAC is given by,

$$Dacheight = (1 + a_i)\frac{C_i}{C_f} \tag{4.1}$$

where C_i is the ideal input capacitance, and C_f is the feedback capacitor. This is simply the ratio of the i^{th} array capacitor to the feedback capacitor. This means that residue error due to capacitor mismatch is not dependent on stage resolution. For similar capacitor matching, the DAC transition heights in a 4 bit stage will produce no more error at the output of its MDAC than the DAC transition heights in a 1 bit stage will at its output. For example, if a 5 bit stage and a 1 bit stage, both with "ten bit good" capacitor matching, are each attached to ideal ten bit back-ends, neither ADC produces "missing" or "extra" codes. The ADC with the 5 bit first stage provides four bits better resolution and DNL than the ADC with the 1 bit first stage. If digital error correction is also employed, the 5 bit first stage case yields fourteen bit resolution and linearity. In comparison, the one bit stage can yield at most eleven bit resolution and linearity.

In the example above, an ideal back-end converter is used. Ideal back-end converters do not exist. Similarly, if the linearity of a back-end ADC can be kept high, the front-end ADC stage will still contribute the largest non-linearity to the system. Additional non-linearity can be kept small by using a multibit per stage back-end. When referred to the input of the back-end, the back-end non-linearity is much smaller than the front end contribution. In this case, the first stage will be responsible for the largest ADC DNL. For example, the DNL produced by the four bit first stage of a ten bit back-end ADC can be no more than one eighth of a converter LSB if the capacitor matching is ten bit good. Conversely, if the back-end were constructed out of single bit stages, the first stage would add one half converter LSB non-linearity, while the second would add one quarter LSB, etc. The ability to achieve higher resolution without calibration is a distinct advantage of

multibit per stage pipelines. This advantage is exploited in the design of the proposed ADC.

The AD924x, the core on which the proposed ADC is based, was designed with a multibit per stage architecture. A back-end ADC comprised of two four bit pipeline stages and a four bit Flash is used to provide resolution of ten bits with DNL of no more than one quarter LSB. A five bit stage precedes the back-end increasing the total resolution to fourteen bits. The first stage array capacitors were sized for noise performance and better than "ten bit good" matching. The overall converter is a 5-4-4-4 structure giving fourteen bit resolution and DNL of no more than one LSB. This architecture is illustrated in Figure 4.1. Figure 4.2 shows a computer simulation of the DNL and INL of a fourteen bit pipeline ADC with the 5-4-4-4 structure. The performance of the AD924x architecture currently marks the practical resolution and linearity limit achievable with non-calibrated pipeline ADCs. [3]

In theory, higher resolution converters could be produced by increasing the resolution of the front-end stage. Pipeline stages converting more than 5 bits are impractical power and area hungry structures. Another possibility for increasing resolution would be to use higher than "ten bit good" matched capacitors in the first stage, and increase the back-end ADC resolution. Matching capacitors to this level physically is difficult and expensive. If a back-end with high enough

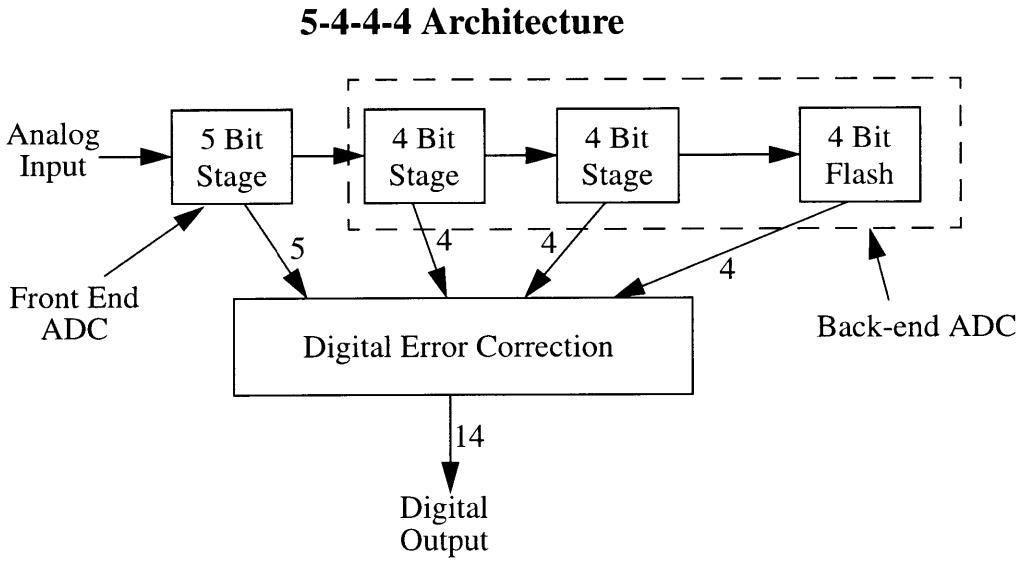


Figure 4.1

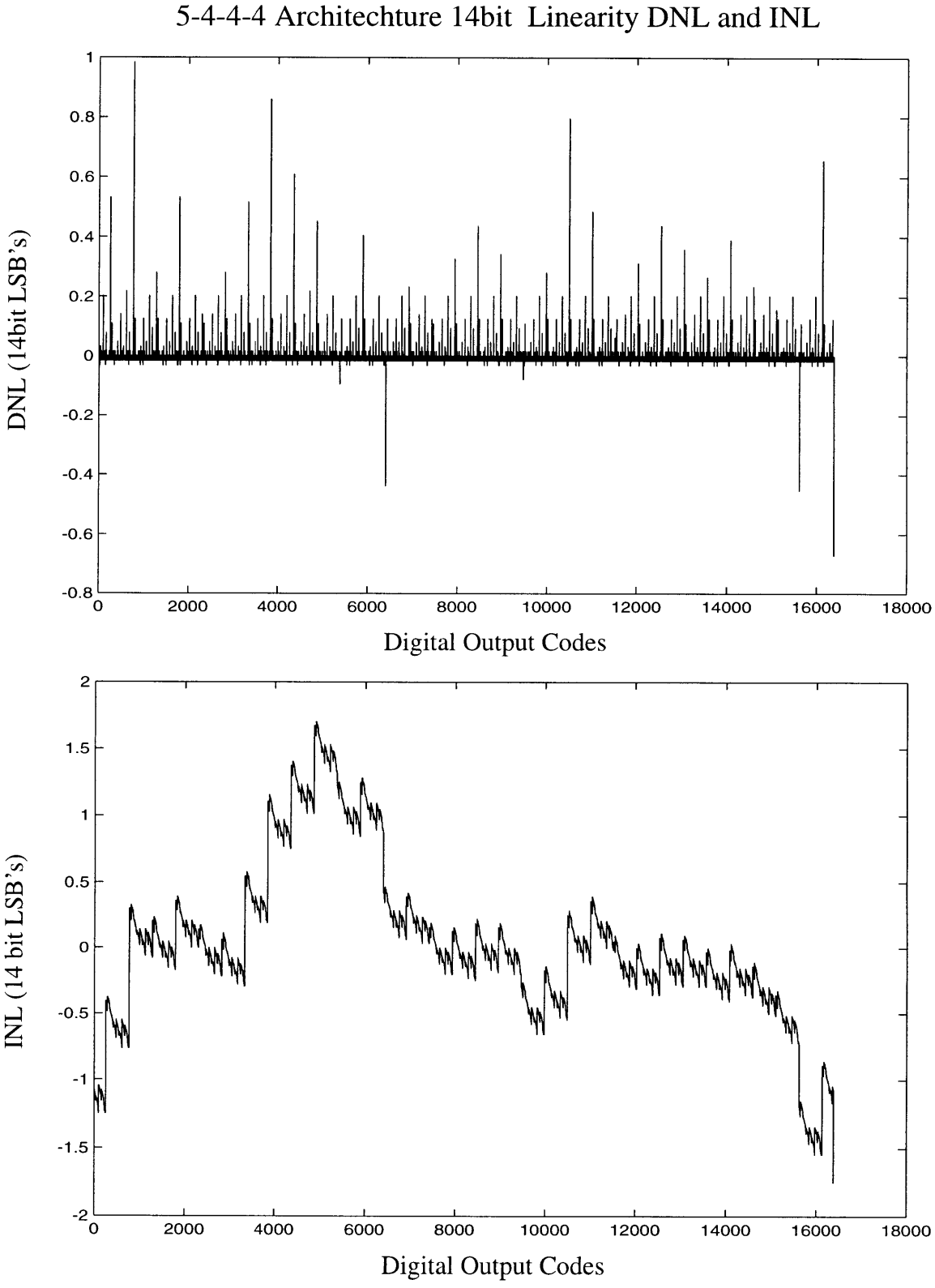


Figure 4.2

resolution and linearity is used, digital calibration of the front-end can be used to achieve

high linearity.

4.5 Achieving Linearity by Digitally Calibrating a Multibit per Stage ADC

Using the general concept behind digital calibration stated in chapter 3, the calibration of a multibit stage can be done in a straightforward manner. Much like the calibration of stages of single bit stages, calibration of multibit stages can be done by repeating the Karanicolas measurement process for each of the 2^N DAC transitions. Unlike the Karanicolas approach, gain reduction is unnecessary in digitally error corrected pipeline ADCs. The correction range produced by the digital error correction technique supplies the necessary buffer zone required for Karanicolas type calibration.

In "Code-Error Calibration Techniques For Two-Step Flash Ana-log-Digital Converters," by Seung-Hoon Lee, et. al., a method of measuring DAC transition sizes and storing only the mismatch error is introduced. By subtracting the error terms from the digital output of the ADC, Lee's algorithm reduced DNL to a sufficient level. Lee conducted the first calibration of a 12 bit Two-Step ADC. By exploiting the DNL symmetry of a binary weighted capacitor array as seen in Figure 4.3, Lee needed only $2^{(N-1)}$ memory locations to correct errors in an N-bit stage. [12] This calibration theory can be adapted to unweighted capacitor arrays by using measurement methods employed by Karanicolas and error extraction methods used by Lee.

The linearity specification for a pipeline ADC governs the number of stages requiring calibration. If the back-end of an ADC is linear enough to produce no "missing" or "extra" codes, then only the front-end ADC requires calibration. Linearity specifications are typically stringent. Specified linearity of no more than one quarter LSB DNL is commonplace. An N bit ADC can be produced with no more than 1/4 LSB DNL when an (N+2) bit linear ADC is designed and the two LSBs of the digital output code are truncated. Using this concept, design of a 16 bit ADC with 18 bit linearity requires that an 18 bit linear ADC be developed.

To minimize digital complexity it is necessary to realize an 18 bit linear ADC with

Binary Weighted Capacitor Array

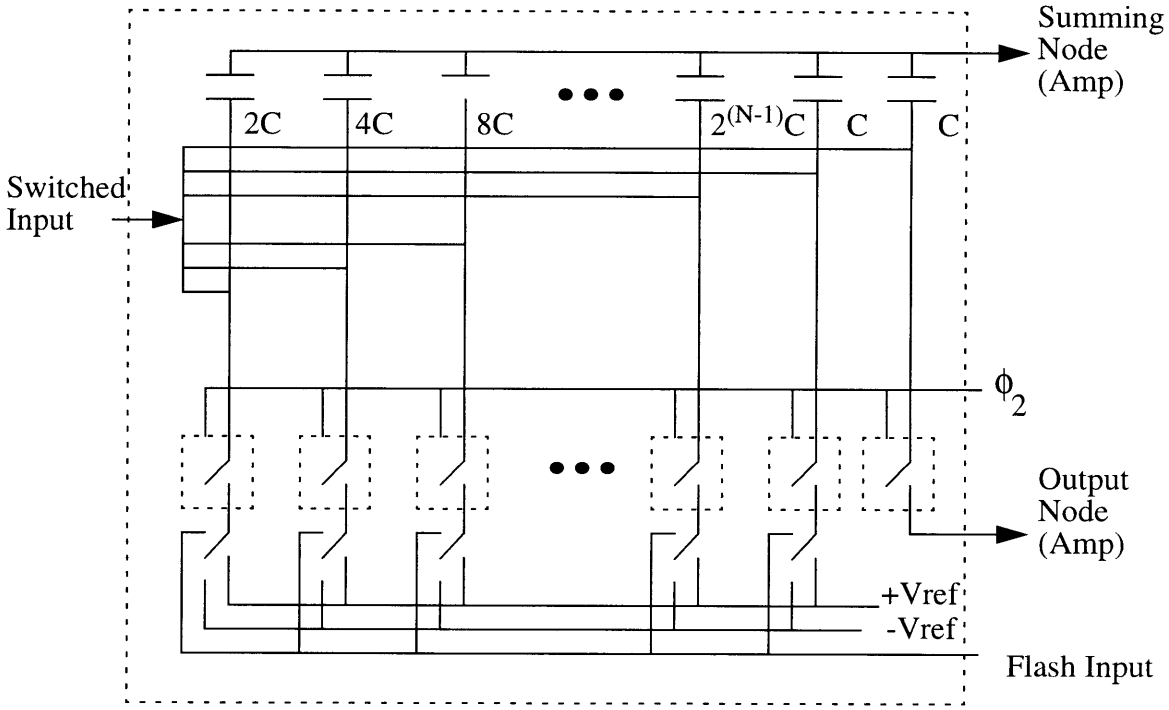


Figure 4.3

single stage calibration. ADC resolution of 18 bits can be achieved by preceding a fourteen bit linear ADC with a calibrated five bit stage. The linearity of the resulting ADC is based on the back-end ADC and the effective calibrated matching of the first stage capacitors. If the error is not measured to 14bit accuracy, the ADC linearity will drop to 16-17 bits. Due to the back-end linearity, measurement accuracy is guaranteed to be 14 bit good. With measurements made correctly, the first stage produces worst case DNL no larger than one quarter of a 16 bit LSB. In this architecture, the first stage is not the only stage to produce non-linearity error. Due to its non-linearity, the back-end produces worst case DNL of one quarter LSB at 16 bits resolution. The 5-5-4-4-4 architecture with a single calibrated first stage can theoretically produce 18 bit DNL performance. This is the architecture of the proposed ADC.

Instead of the two residue segments used by Karanicolas, there exist 2^N+1 in the

new converter. Figure 4.4 shows the 33 different residue segments of the new first stage. Measurements are made by forcing the stage into certain modes of operation. The residue corresponding to A_i, in Figure 4.4, is produced by the following procedure. A voltage equal to the first Flash decision level is sampled onto the capacitor array during the sample phase of the MDAC. During the amplify phase, all the switches are set to the low reference. Residue, A_i, is referred to as the base case in chapter 5. The residue corresponding to B_i is produced by the following procedure. During sample phase, the same decision voltage is sampled onto the capacitor array. During the amplify phase all switches are forced to the low reference except the one attached to C_i which is forced to the high reference. A_i and B_i are quantized using the back-end. Subtracting the quantized values yields,

$$X_i = A_i - B_i \qquad 0>i>33, \quad A_i = A_1, \quad X_0=0 \tag{4.2}$$

These measurements are done one at a time for each and every DAC step, and each X_i is

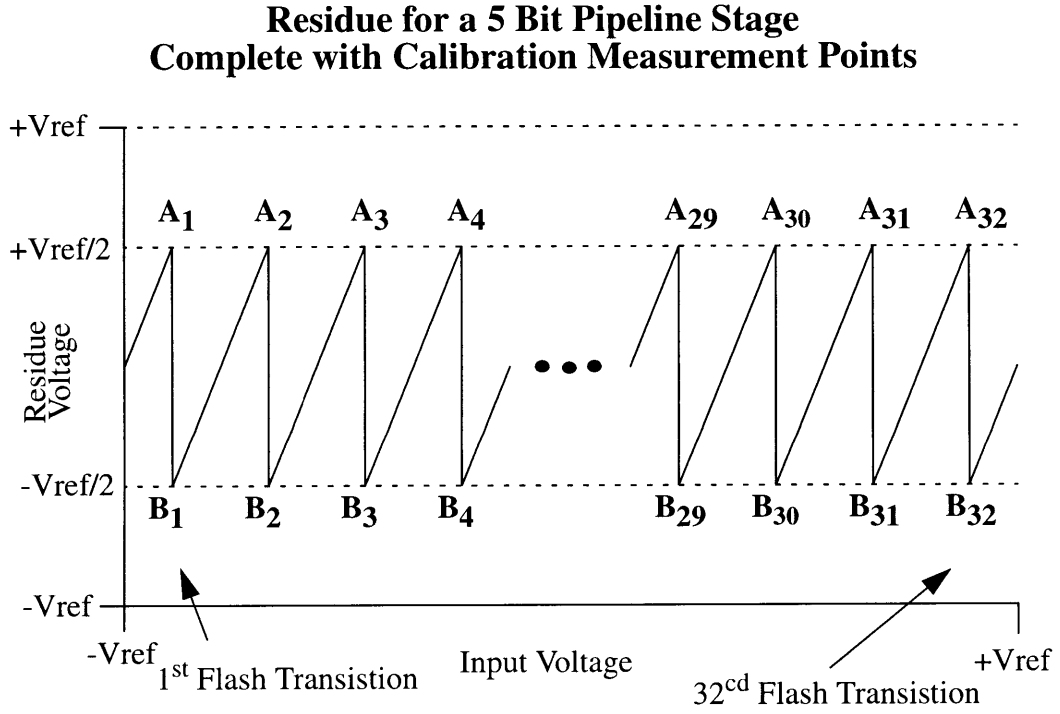


Figure 4.4

stored in memory. Using a Karanicolas type methodology, the output digital code would

now be given by,

$$C = \begin{cases} Y & \text{Segment 0} \\ Y + \sum_{m=1}^{i} X_m & \text{Segment } i \quad 0<i<33 \end{cases} \tag{4.3}$$

where C is once again the code produced by the stage and back-end, Y is the back-end quantization of the stage residue, and i is the stage's Flash code. On the other hand, it can be shown by,

$$Error_i = X_i - LSBF \tag{4.4}$$

where LSBF is the digital representation of a first stage's LSB, that the error due to capacitor mismatch is a small component of the measured value. The other component, LSBF, can be returned by the converter itself. This means that a wide digital code like X_i can be narrowed by subtracting the ideal step size represented by LSBF. The new calibration algorithm is given by,

$$C = \begin{cases} Z & \text{Segment 0} \\ Z + \sum_{m=1}^{i} Error_m & \text{Segment } i \quad 0<i<33 \end{cases} \tag{4.5}$$

where Z is the raw digital output of the overall converter. The new algorithm uses narrower digital memory locations and adds errors directly to the raw output of the ADC. By doing this, the complexity of the integrated digital circuitry is reduced. Chapter 5 presents the digital circuit that performs these functions.

The measurements mentioned above could be done differently, and this difference is worth mentioning. Measurements could be done by sampling 32 different voltages, each consistent with the 32 Flash transitions of the first stage, and measuring the step sizes at these points individually. This new method requires an external DAC or additional internal analog amplifier circuitry to perform calibration. This drawback is offset by the fact that non-linearities in the input capacitor voltage coefficient could possibly effect the accuracy of the measurements made by the original method. Without major non-linearities occurring due to capacitor voltage coefficients, the original method of measurement will be

accurate. In the future, it may be necessary to involve an external DAC in the calibration of the ADC to minimize errors cause by internal non-linearity.

4.6 Simulated Linearity and New Discoveries

Before implementing the proposed architecture, extensive computer simulation was done to assure that the back-end ADC would provide good enough measurement capability. A simulation program was written in the C language is shown in Appendix A. A model of an N-bit pipeline ADC stage including the effects of input offset, Flash ADC error, and capacitor mismatch was developed. Digital error correction was applied by reducing the closed loop gain of each stage by half and overlap adding the Flash outputs. First a worst case simulation of the 5-4-4-4 backend ADC was performed. Plots of DNL and INL were displayed previously in Figure 4.2. With DNL of less than 1 LSB, the linearity of the backend is good enough to ensure 18 bit overall ADC linearity. Next, a simulation was run for worst case capacitor matching, input noise effects on calibration, residue amplifier input offset voltage, and Flash error in a 16 bit ADC. DNL and INL plots were extracted from the simulation results. The simulated INL and DNL plots for an uncalibrated 16 bit ADC with a 5-5-4-4-4 architecture are displayed in Figure 4.5. Figure 4.6 shows simulated INL and DNL for the same converter after digital calibration is performed. Note that after digital calibration, DNL meets the linearity specification and INL is greatly reduced.

Besides proving that specified linearity can be achieved, the simulation led to a few other discoveries. After adding random noise to the inputs of each pipeline stage in the simulation, it was discovered that the noise effected the accuracy of the measurements in the calibration phase. The effects of the noise on measurements can be reduced by one half for each four times averaging done to the measurements. With a simulated 1 LSB RMS input referred ADC noise, averaging by 1024 times reduced the effective RMS noise effecting the measurements by 1/32. Effective input referred noise of 1/32 LSB RMS was not large enough to cause error in the calibration measurements. Averaging by 1024 times was employed to reduce the noise effects. As mentioned in chapter 3, residue amplifier offset causes total ADC offset. Offset due to the calibration algorithm also added to the

total ADC offset. The transfer offset was corrected by measuring it after calibration and subtracting its digital value from each calibrated ADC output. Finally, a gain error was found in the ADC transfer function. A distribution of ADC gain error for the 5-5-4-4-4 architecture is illustrated in Figure 4.7. The gain error had a mean of 0 and standard deviation of 0.22%. This error size caused no alarm during simulation and consequently no gain error correction has been designed into the proposed converter.

By exploiting the linearity of multibit per stage ADCs, an architecture, namely the 5-5-4-4-4 structure, has been chosen to fulfill the linearity specification of the proposed ADC. In contrast to the Karanicolas approach, 16 bit accuracy can be attained with a high degree of linearity while calibrating only one pipeline stage. In theory, this ADC architecture can produce the desired linearity and performance. In practice, issues pertaining to speed, accuracy, noise performance, and digital functionality must be addressed before a final product can be released. Chapter 5 discusses practical aspects of the proposed ADC while addressing the issues mentioned above.

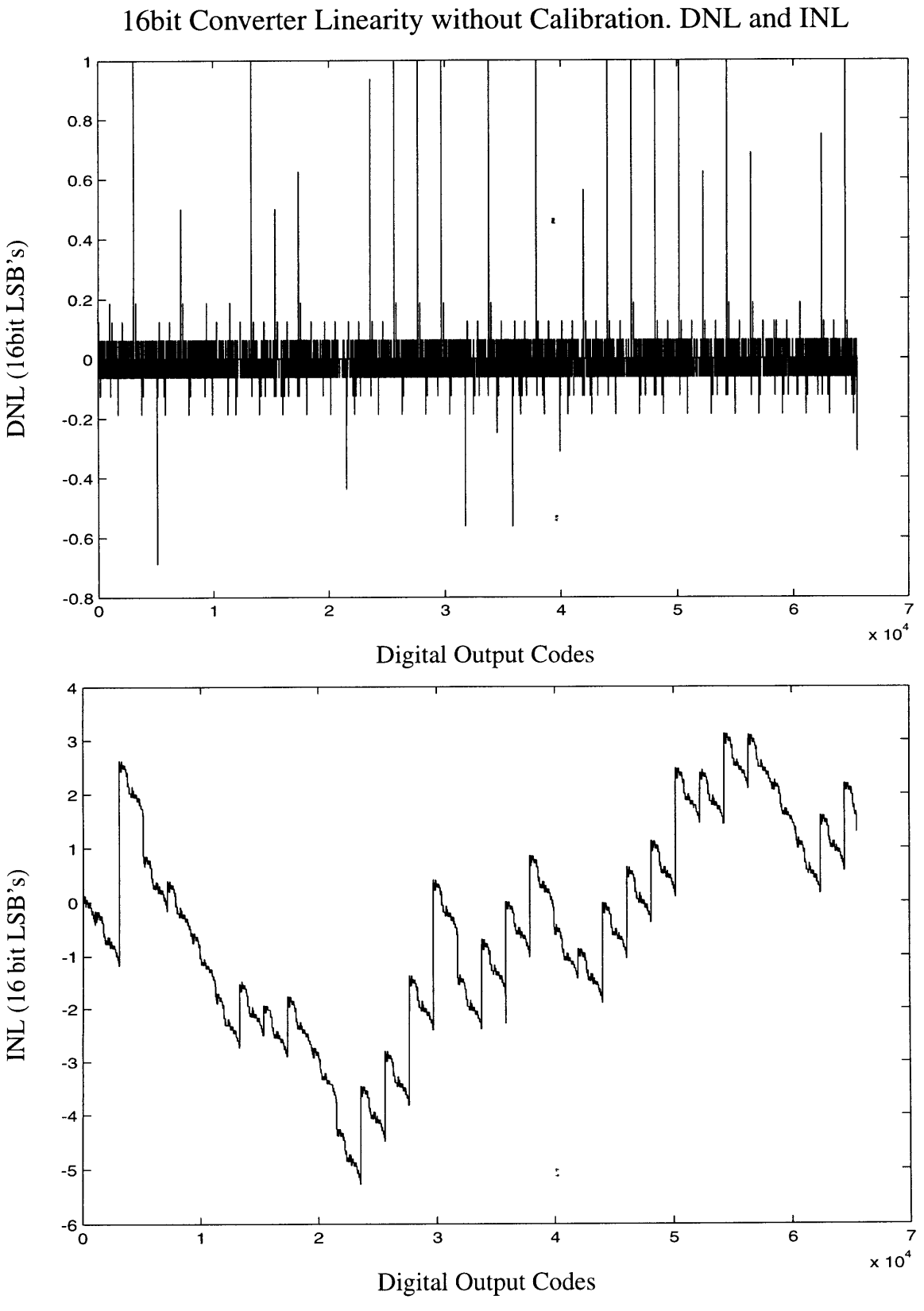

16bit Converter Linearity without Calibration. DNL and INL
DNL (16bit LSB's)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
1
2
3
4
5
6
7
x 10^4
Digital Output Codes
INL (16 bit LSB's)
4
3
2
1
0
-1
-2
-3
-4
-5
-6
0
1
2
3
4
5
6
7
x 10^4
Digital Output Codes

Figure 4.5

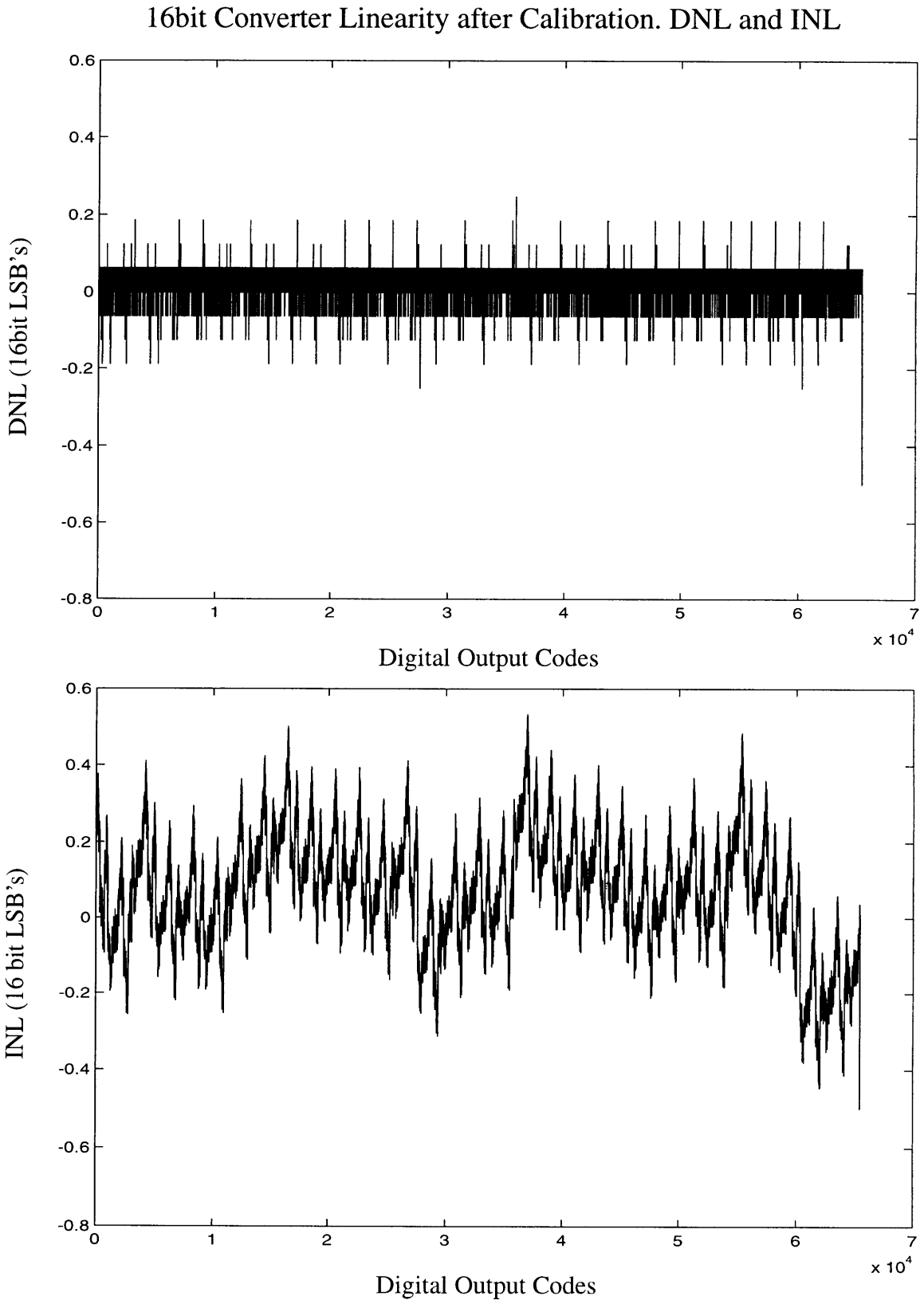

16bit Converter Linearity after Calibration. DNL and INL
DNL (16bit LSB's)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
1
2
3
4
5
6
7
x 10^4
Digital Output Codes
INL (16 bit LSB's)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
0
1
2
3
4
5
6
7
x 10^4
Digital Output Codes

Figure 4.6

Gain Error Distribution

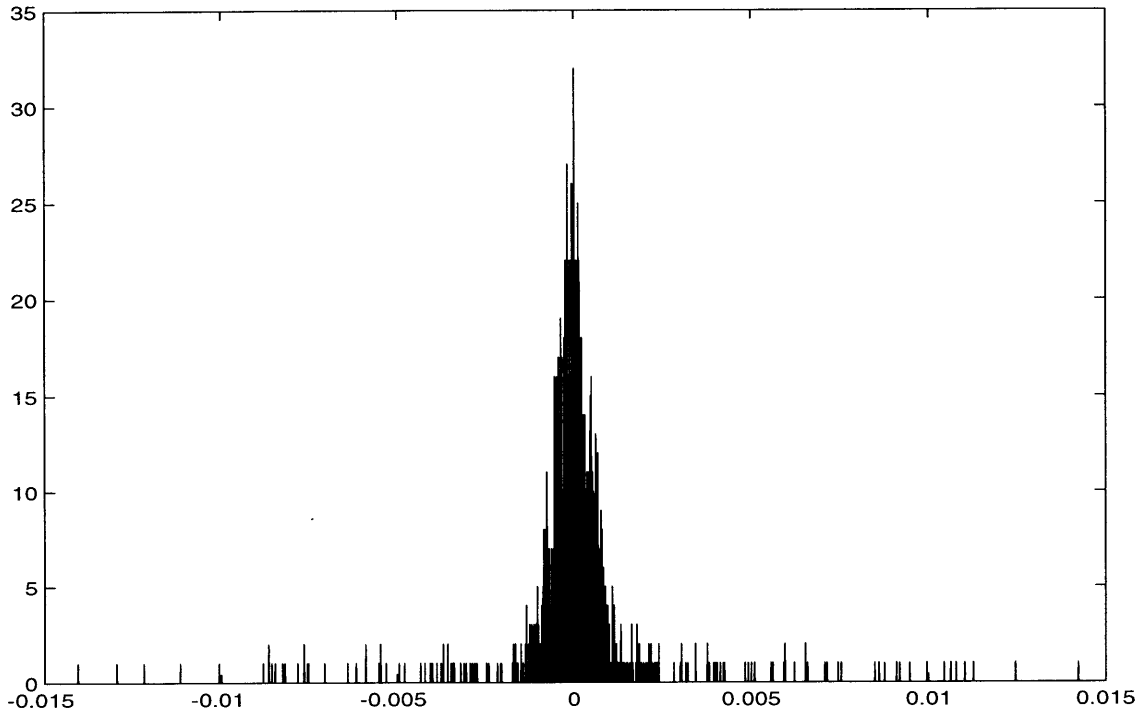


Figure 4.7

Chapter 5

Circuit Design of the Proposed ADC

5.1 Introduction

Now that the theory of operation, error correction, and architecture modification has been presented, circuit level design for the proposed ADC can be discussed. Circuit levels changes are very important to the overall operation of the ADC. The proposed ADC architecture change calls for a new first stage. The first stage must be designed to provide sufficient accuracy and speed to operate within the ADC specifications. Circuit changes must be made to guarantee that the ADC provides sufficient noise performance. For linearity purposes, a digital overhaul must also be performed in order to supply the on-chip functionality required by the calibration algorithm. This chapter deals with each of these considerations individually.

5.2 Base Architecture

The proposed ADC is based in part on the AD9243, 3Ms/s 14 bit pipeline ADC. The AD9243 is a switched capacitor mixed signal circuit with a block diagram shown in Figure 5.1. The ADC is comprised of analog and digital sections. The analog section includes a SHA, three MDAC stages, four Flash ADCs, a voltage reference, and a set of

output buffers. The digital components include a clock generator, MDAC switch decoding circuitry in each MDAC & Flash pair, encoding and delay circuitry for Flash outputs, and digital error correction logic. Together these subsystems work as a 14 bit linear entity.

5-4-4-4 AD924x Core Analog-Digital Converter

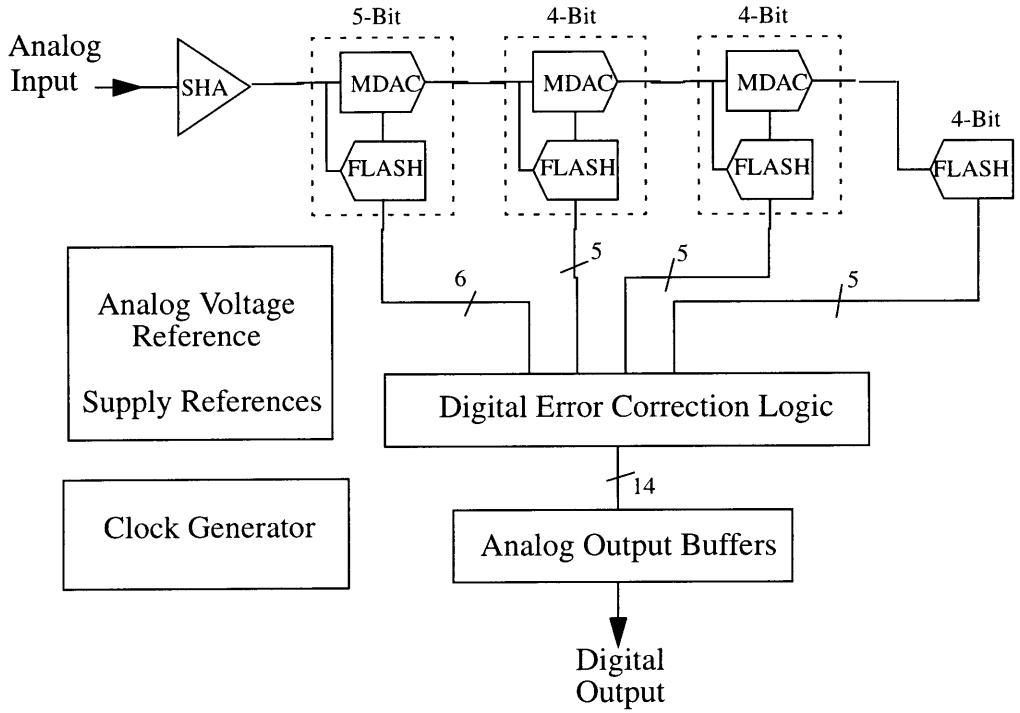


Figure 5.1

The AD9243 operation is as follows. The SHA tracks a continuous time signal on one the first of two clock phases, samples the input just prior to the end of this phase, and transfers the input sample value to its output during the second clock phase. As long as it properly settles to the correct value, the SHA output can be sampled by the first pipeline stage prior to the end of the second clock phase. A differential SHA along with the ADC clocking scheme is shown in Figure 5.2. Figure 5.3 illustrates the first stage of the pipeline, the next stage to be considered. The first stage works in the following manner. During the second clock phase, the MDAC and first Flash track the SHA output. By this time the Flash has already sampled its decision levels. Prior to the end of the second clock phase, the MDAC samples the SHA output and the Flash latches on the decisions made. During the first phase of the next clock cycle, the MDAC uses the latched Flash output to settle to a residue value. This residue value is ready for sampling by the next stage just prior to the

end of the current clock phase. This process is repeated by the successive stages of the pipeline on alternating clock cycles. As a result, the latency of each stage is one half clock cycle. The bit outputs by each Flash are delayed by different amounts so that they may be added together on the same clock phase. A digital error corrected code is produced in the correction logic. The code is then adjusted to eliminate code over-range and is buffered to the output pins. The latency from input to output is on the order of 3 clock cycles and presents no appreciable delay.

Differential Sample and Hold Amplifier and Timing

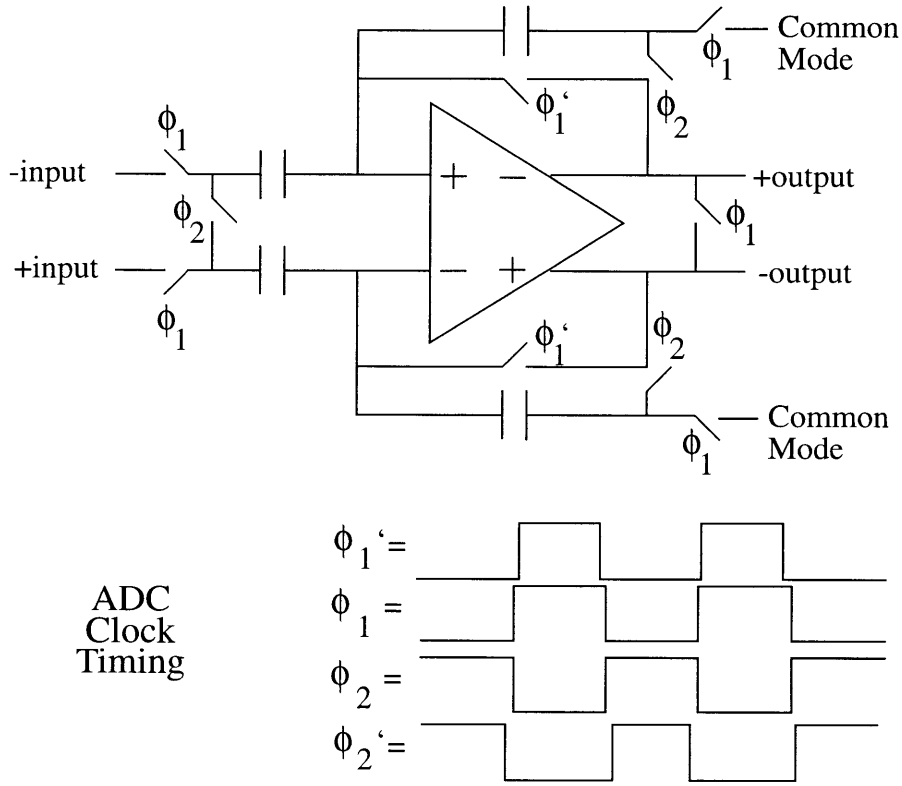


Figure 5.2

Other components are necessary in the overview of the AD9243. For instance, the on-board voltage reference and reference amplifier provide two reference voltages, 1.25V and 3.75V, representing the single-ended input range of each stage. Implemented with a fully differential architecture, the AD9243 has a 5Vp-p differential input range. As a result, one LSB at 14 bits is 305uV. Another component, the clock generator, takes an outside master clock and generates the non-overlapping clock phases shown in Figure 5.2. Finally, between each Flash ADC and MDAC exists logic to provide communication.

Encoders in the Flash create thermometer code from the outputs of the Flash latches. This thermometer code is used by the MDAC. In the MDAC a decoder is used to operate the reference switches during the MDAC amplify phase.

Flash and MDAC pair showing circuit timing

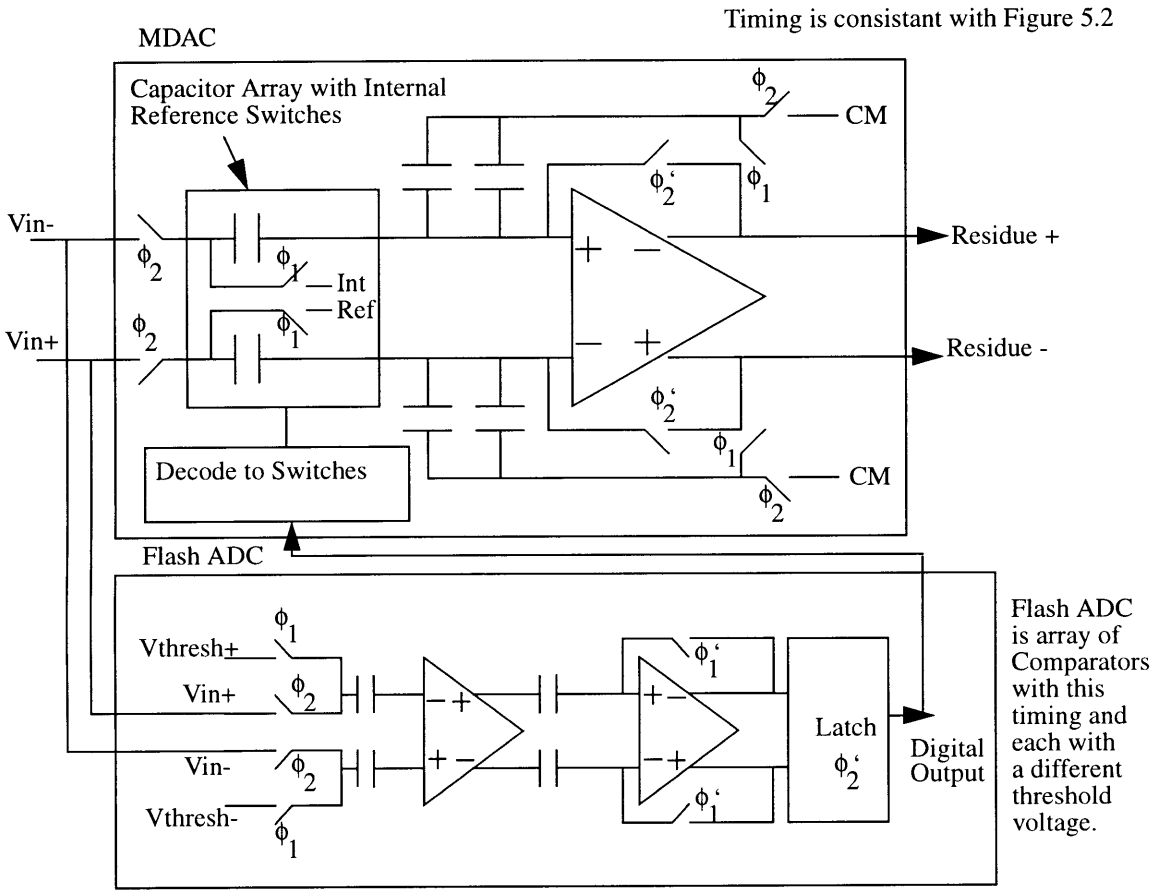


Figure 5.3

Certain performance measures are useful when it comes to adapting the ADC for higher resolution. The INL and DNL of the AD9243 limit the INL and DNL of a new converter. Knowledge of these metrics is important. The AD9243 operates with max INL of 2 LSBs and max DNL of 1 LSB at fourteen bit resolution. The noise performance of the AD9243 is also important in designing noise performance into the new converter. Input referred RMS noise in the AD9243 is135uV, or 44% of an LSB. Since the new converter must operate at 3Ms/s, the speed of this ADC is also important. The AD9243 converter

runs comfortably at a speed of 3Ms/s. Consequently, extension to 16 bits at this speed requires no changes to the back-end ADC itself.

5.3 The Proposed ADC and Its Structure

Realization of the proposed 16 bit ADC actually requires minimal adaptation of the AD9243. Simple circuitry changes and trade-offs are made so that each of the new ADC specifications is met. The architecture of the 16 bit ADC is displayed in Figure 5.4. The addition of a 5 bit MDAC and Flash ADC pair, the elimination of the SHA, and the overhaul of the part's digital circuitry constitute the only major changes made to the AD9243. Conversion to a 5-5-4-4-4 architecture requires the addition of a new front-end stage. A SHA free circuit implementation increases the ADC's noise performance. Also, a digital circuit overhaul is necessary to supply off-chip calibration and on-chip correction capability. All the other circuit components are left alone. The latency of the new ADC is similar to that for the AD9243. With the new first pipeline stage taking the place of the SHA from the AD9243, the timing of the back-end is left unchanged. Of the adaptations presented above, the new first pipeline stage amplifier shall be considered first.

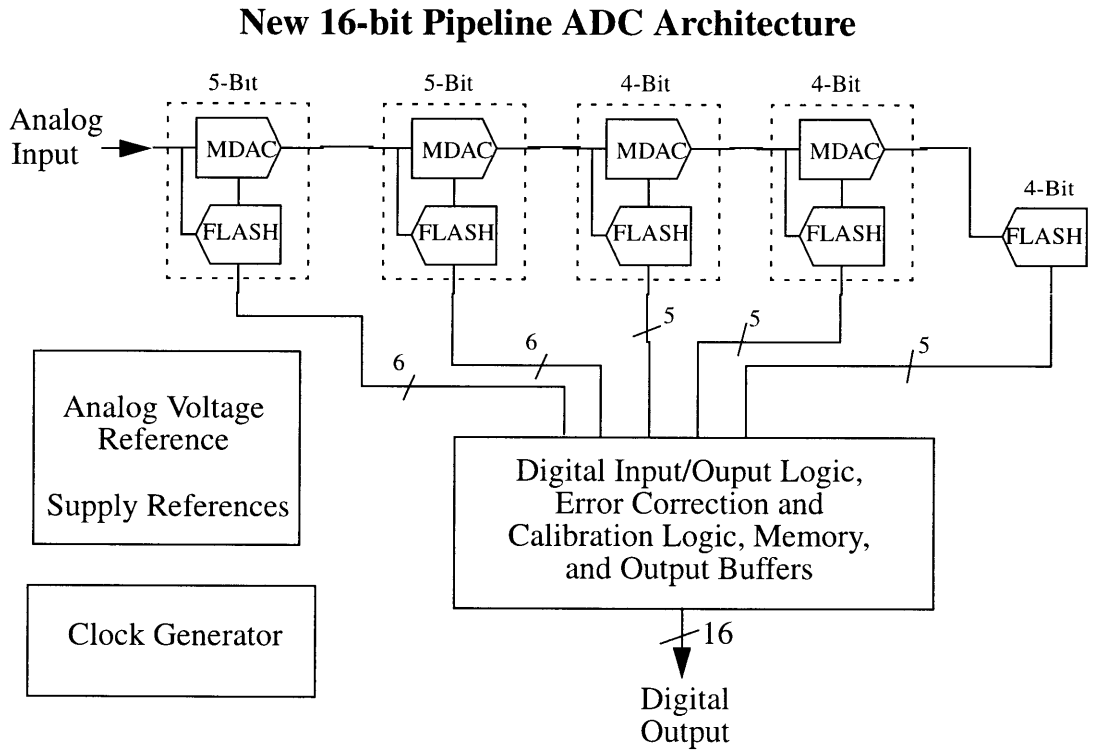


Figure 5.4

5.4 The First Pipeline Stage

The first stage in every pipeline must meet a set of stringent specifications in order that the entire pipeline work. For the proposed first stage these specifications are as follows. The first stage must be able to provide 5 bit resolution over its input range. Both the Flash and MDAC capacitor array must be size accordingly. With a first stage resolution of 5 bits, the residue amplifier must have a closed loop gain of sixteen. The residue amplifier must also be able to settle the same load capacitance as the AD9243 SHA in the same amount of time. Settling must occur to sixteen bit accuracy so that calibration measurements will be correct and normal mode operation will be accurate. Aside from the residue amplifier and Flash, the first stage capacitor array and switching network need to be modified so that the entire MDAC can be forced into calibration states.

5.4.1 The Flash ADC

The addition of a Flash ADC is very simple in this case. A duplicate of the Flash ADC used in the first stage of the AD9243 can be used as the new first stage Flash ADC. The Flash returns the 5 bits required in the resolution specification. The Flash ADC is constructed from an array of 32 offset cancelled sampling comparators. As a result, the Flash provides the increased error correction needed for continuous time sampling. With the new dynamic sampling scheme, sampling agreement must be made between the first MDAC and first Flash ADC. In chapter 6, this issue will be discussed as a future design consideration.

5.4.2 The Residue amplifier

Unlike the Flash ADC, the addition of the residue amplifier is not simple. An appropriate amplifier architecture must be found to provide all of the specifications required above. The first stage amplifier must have high DC gain for settling accuracy. The application also requires high bandwidth in order to quickly settle a sizable load capacitance under high closed loop gain conditions. An adapted version of the AD9243 first stage residue amplifier is used to do the job.

The AD9243 first stage amplifier is a Miller compensated two-stage amplifier. The amplifier is used in two distinct modes of operation. These modes of operation are displayed in Figure 5.5. The two-stage amplifier itself is not unity gain stable. As a result, the first and second stages are decoupled during the sampling phase. Tied in unity feedback, the first amplifier stage performs the offset cancellation sample function. During the MDAC amplify phase the two stages are again coupled. With a higher closed loop gain during amplify phase, the residue amplifier is stable and performs the correct residue amplification. With some modification this amplifier can be used in the first stage of the new ADC.

First Amplifier Modes of Operation

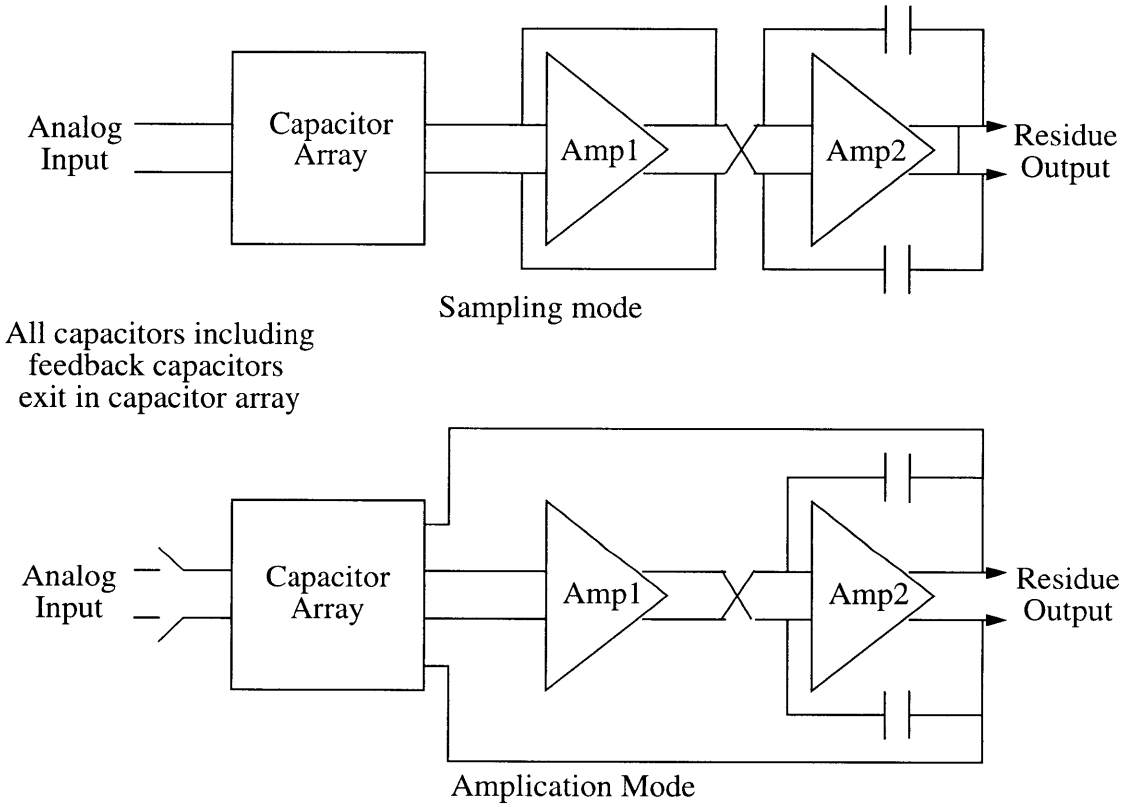


Figure 5.5

Understanding the specifications of the AD9243 first stage residue amplifier is necessary when adapting for use in the new first stage. The old amplifier settles a 5pF load capacitance in 166ns. The amplifier also has enough DC gain to settle a ten-bit back-end to approximately 14 bit accuracy. The new op-amp must settle a load capacitance of approximate 15.5pF. The amplifier DC gain must also be increased to provide at least 16

bit settling accuracy. This accuracy can be related directly to DC gain by,

$$GainError = \frac{1}{1 + DCgain} \tag{5.1}$$

16 bit settling accuracy can be translated to a gain error of 0.00153%. The smallest gain providing such gain error is 65400. As a result, DC gain should be designed to be greater than this value. Taking into account these facts, a new residue amplifier can be designed.

First of all, considering the large difference in load capacitance, it is necessary to make the new amplifier have a higher drive capability. The new load capacitance is more than three times the load seen by the old amplifier. In order to keep the second order system pole of this amplifier at the same place as it was in the old amplifier design, the second stage transconductance must be scaled up by three. The increase in second stage transconductance increases the second stage input parasitics causing pole movement toward lower frequencies. By scaling the compensation capacitors up by three, the second order pole placement can be preserved. An increase in compensation now requires a three times scaling of the first amplifier stage transconductance. In effect the total amplifier is scaled by three. As a scaled version of the old residue amplifier, in both size and current, the new amplifier can settle the required load capacitance. Settling the load to the appropriate accuracy is also necessary.

To better the settling accuracy of the first stage residue amplifier, more DC gain must be added. Both amplifier stages already possess single cascode transistors to achieve high output impedance. The addition of a second cascode transistor would further increase the DC gain of the entire amplifier. It would also reduce the output headroom of each stage which is already critically small. Both DC gain and offset cancellation performance can be manipulated by changing the first stage amplifier gain. As a result, modification is done on the complementary cascoded first stage amplifier shown in Figure 5.6a. In the old first stage amplifier transistors M1 and M2 set bias voltages for the cascode transistors. By using these transistors to set the common-mode output voltages for a pair of differential gain enhancement amplifiers, the DC gain of the residue amplifier can be increased while the headroom at the output nodes stays the same. The new amplifier first stage is shown in Figure 5.6b. Also, after increasing the first stage DC gain, cancellation of the first stage input offset voltage becomes much better.

After the gain enhancement was added to the amplifier, settling time and accuracy were tested. Originally DC gain fell well above the specified value and settling was lightning fast. It was found that the settling speed was more dependent on the slewing than on actual linear settling. This slewing is directly related to the slew capabilities of the new

Amplifier First Stage - Before and After

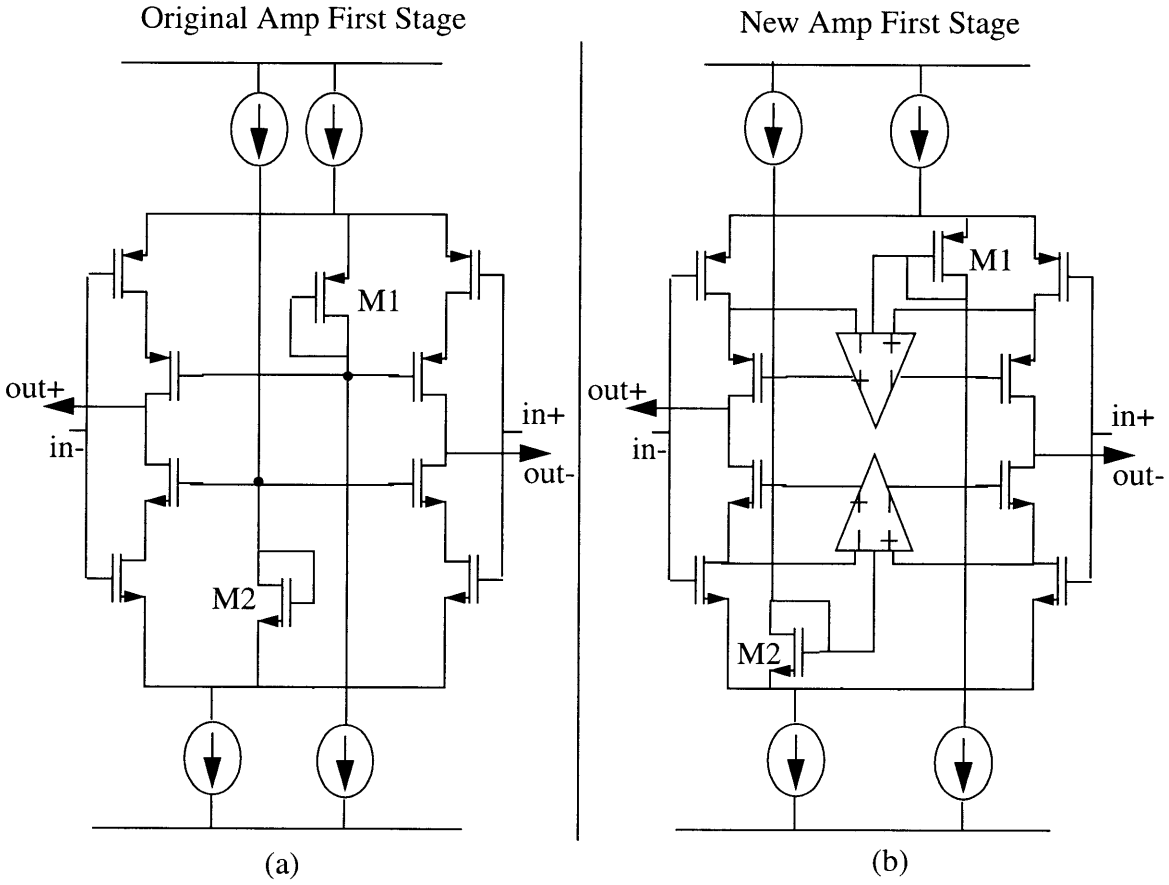


Figure 5.6

gain enhancement amplifiers. Speed and accuracy were well within specifications and could be degraded without a loss of performance in the overall ADC. A series of optimizations was done so that the amplifier would slow down, become less accurate, and burn less power. These optimizations included reducing the physical size of all transistors by 2/3, reducing the bias current and power of the amplifier by 2/3, and reducing the value of the compensation capacitors by 2/3 to preserve bandwidth. After the optimizations were made the final amplifier design still met the necessary specifications. Gain and phase for the final amplifier design is given in Figure 5.7. Figure 5.7 takes into account the closed loop gain of 16 and the worst case conditions for these plots. The amplifier has a DC gain of

105dB, 177000. It is stable with a closed loop phase margin of >70°. Figure 5.8 shows the worst case settling over process conditions and input voltages for the amplifier. Note the slewing in the settling function as the amplifier settles in 100ns. The initial power dissipated by the entire first stage residue amplifier was approximately 15mW. After the modifications, the amplifier power rose to 45mW.

Changes made to the rest of the first stage circuitry are tied closely to the calibration scheme that will be used. These modifications are dealt with when the calibration circuitry is introduced.

5.5 Minimizing Noise

In modern electronics it is very important to keep noise levels as low as possible to prevent noise from masking very small input signals. In ADCs, tolerance to noise depends directly on the resolution of the ADC. The minimum noise level that can be attained in an ADC is referred to as the quantization noise level. RMS quantization noise is given by,

$$\sigma = \frac{LSB}{\sqrt{12}} \tag{5.2}$$

where LSB is the LSB voltage of the system. When the ADC random noise component is smaller than the quantization noise, the quantization noise dominates. If the RMS input referred random noise component is kept under 1/3 LSB, the increase in ADC noise will be less than 4dB. This noise level gives relatively good performance and is used as a target for the proposed ADC.

Analog noise is due to a number of different factors. Amplifier noise contributes to the overall ADC noise component. Amplifier noise is typically tough to calculate by hand, so an intuitive view shall be given. During each phase, sample and amplify, the first stage amplifier contributes noise components to the ADC. During sampling mode, the unity feedback first amplifier stage amplifies its input noise onto the capacitor array. This noise is composed of thermal and flicker noise components. Flicker noise is dominant at low frequencies. In circuits with high sampling rates using offset cancellation, the flicker noise component is effectively cancelled with the DC input offset of the amplifier. [13] Thermal

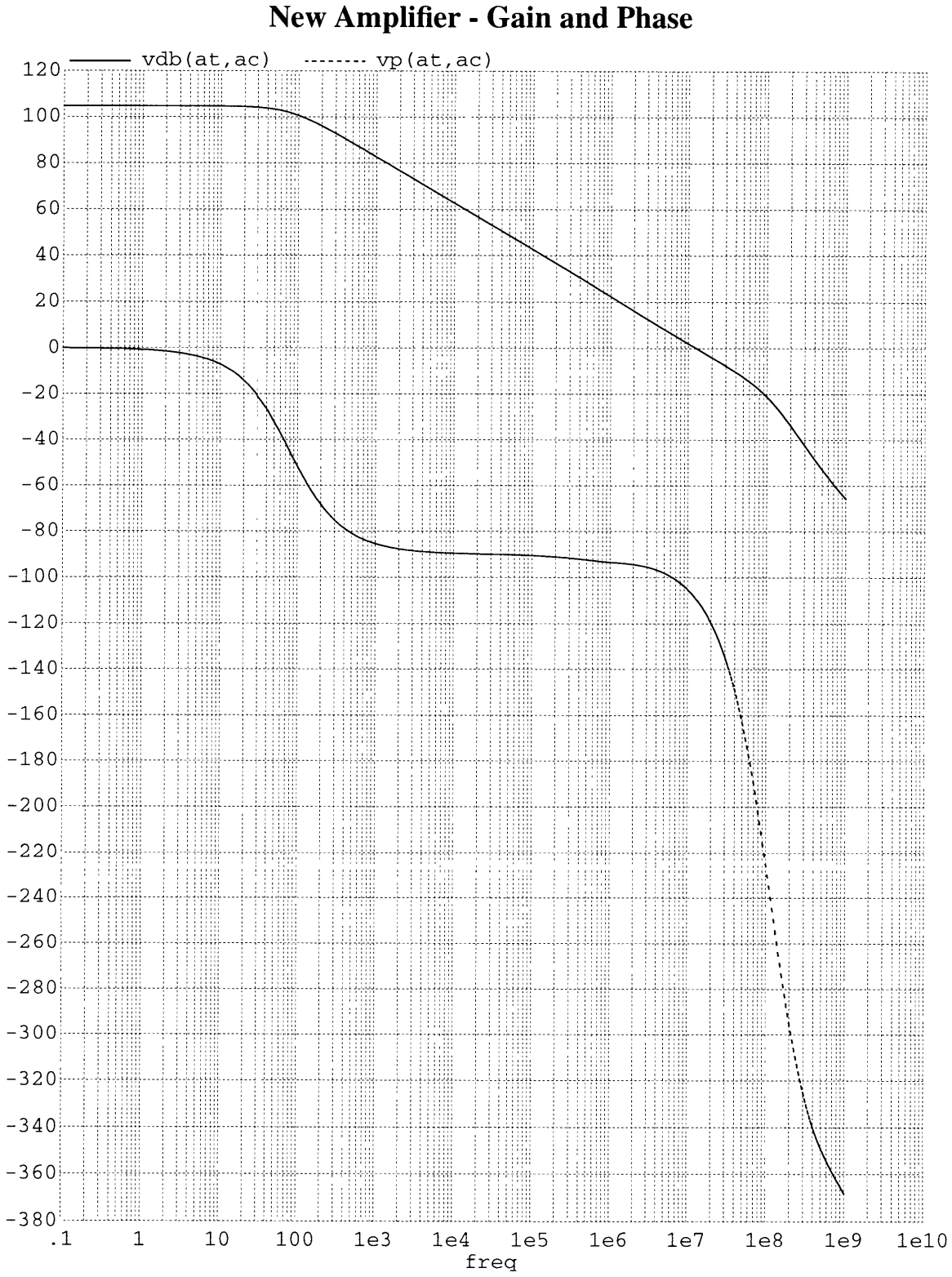

New Amplifier - Gain and Phase
vdb(at,ac)
vp(at,ac)
120
100
80
60
40
20
0
-20
-40
-60
-80
-100
-120
-140
-160
-180
-200
-220
-240
-260
-280
-300
-320
-340
-360
-380
.1
1
10
100
1e3
1e4
1e5
1e6
1e7
1e8
1e9
1e10
freq

Figure 5.7a

View of Transition frequency and Phase Margin

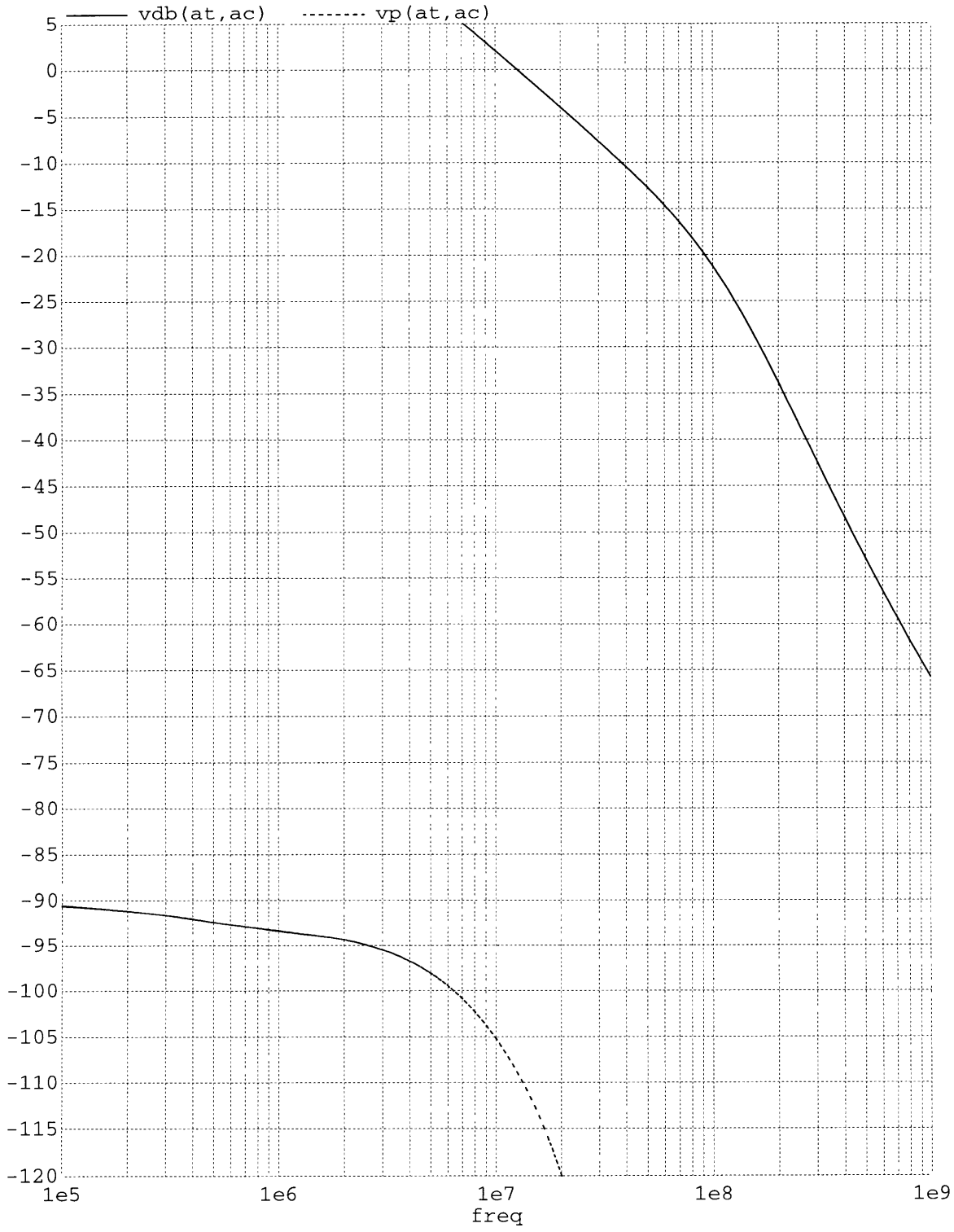


Figure 5.7b

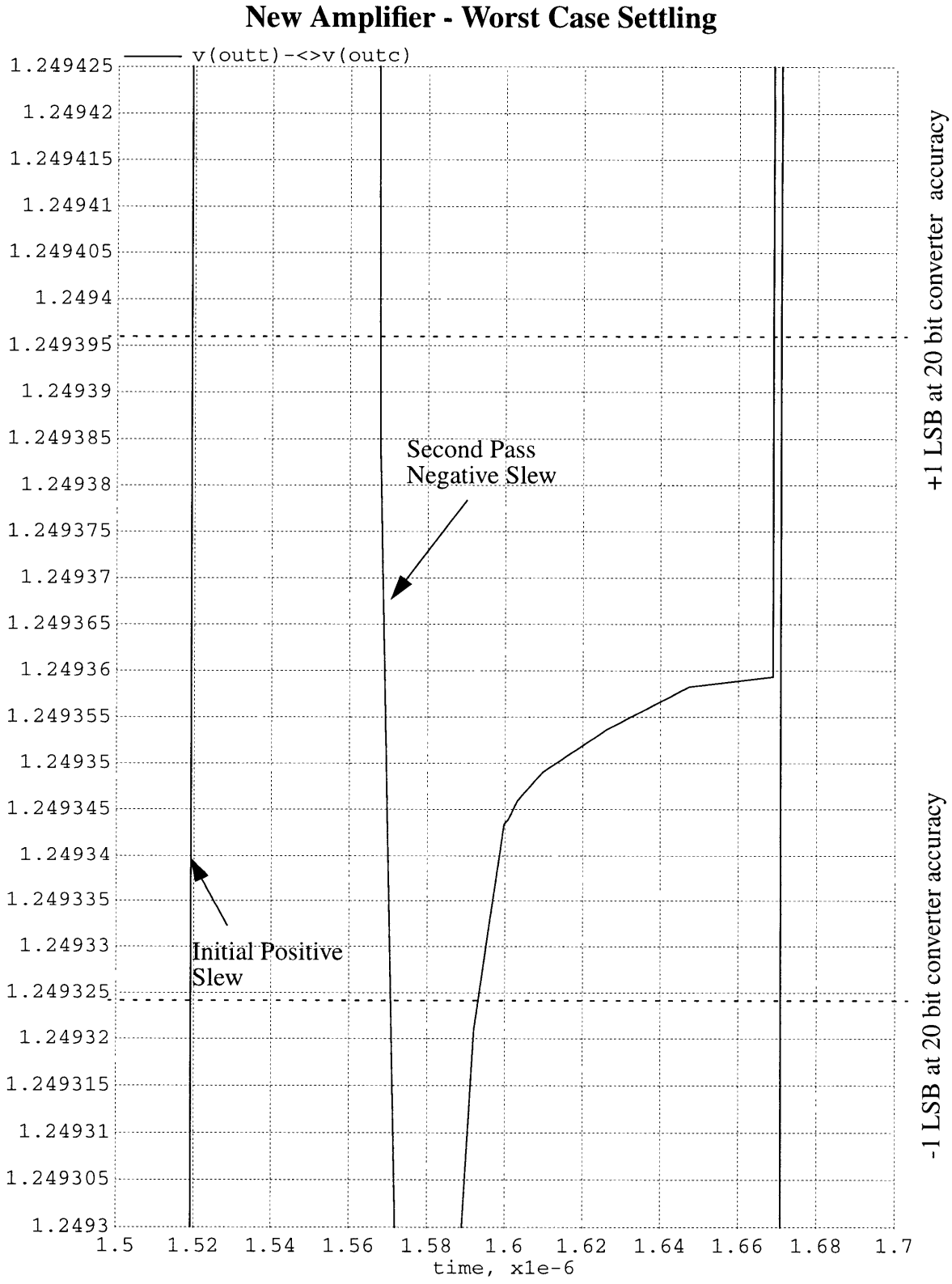

New Amplifier - Worst Case Settling
v(outt)-<>v(outc)
1.249425
1.24942
1.249415
1.24941
1.249405
1.2494
1.249395
1.24939
1.249385
1.24938
1.249375
1.24937
1.249365
1.24936
1.249355
1.24935
1.249345
1.24934
1.249335
1.24933
1.249325
1.24932
1.249315
1.24931
1.249305
1.2493
1.5
1.52
1.54
1.56
1.58
1.6
1.62
1.64
1.66
1.68
1.7
time, x1e-6
Second Pass
Negative Slew
Initial Positive
Slew
+1 LSB at 20 bit converter accuracy
-1 LSB at 20 bit converter accuracy

Figure 5.8

noise is wideband and adds significantly more noise power at the amplifier summing node. Wideband noise added by drive amplifiers is also caught on the input capacitors of a pipeline stage. The capacitor array and input switches, on the other hand, form an RC low-pass filter which filters and reduces the power of this input amplifier noise. During the amplification phase, the amplifier's input noise is multiplied by its closed loop gain and sampled by the next pipeline stage. This noise contribution is quite large. KT/C noise, or switch noise, is the another noise component in switched-capacitor circuits. Produced by the input switches, switch noise is trapped on the capacitor when the sampling switch opens. The RMS value of this noise for a differential sampling structure is given by,

$$\sigma = \sqrt{\frac{2KT}{C}} \tag{5.3}$$

where C in the MDAC input capacitance. Noise contributes are made on both phases. During the sampling phase, switch noise is contributed by the input switches of the MDAC. During amplification, the reference switches will add to the total noise but the amplifier will dominate the second phase contribution. Noise in switched-capacitor circuits can be reduced by increasing the value of C in Eq. 5.3. Circuit changes in the proposed ADC for noise performance shall now be discussed.

The new converter has the same input range as the AD9243. Its LSB size is 76uV. The noise from the SHA and first MDAC of the AD9243 is 130uV RMS. From the Eq. 5.3, to lower the noise to 26uV RMS, the capacitors in the circuit would have to be increased 25 times. The AD9243 SHA has 4pF input capacitors. The new ADC would have 100pF input capacitors. This capacitance is ludicrous in high-speed applications. The SHA would also have to drive 25 times its original load. This notion is impractical and the proposed ADC does not include a SHA. With a SHA, the input referred noise is dependent on the SHA and first stage. The second stage noise, when input referred, is reduced by the closed loop gain of the first stage and is almost negligible. Without a SHA, the first stage becomes the only dominant noise contributor in the system. This means by increasing the first stage array capacitor sizes, the noise performance can be increased.

Without the presence of a SHA, the first ADC stage no longer samples a held voltage. In the new configuration, the first MDAC and Flash ADC pair sample a continuous time input signal. This input signal is driven through the input switches by an external

buffer amplifier. During the sampling instant the sample switches of all the Flash comparators as well as the sample switch of the MDAC open. The input switches then open as usual. A problem may occur if all the sample switches do not open at the same time. As mentioned earlier, this issue will be discussed in Chapter 6.

SPICE simulations were used to measure the noise contributed by the first two stages of the new ADC. Each of the array capacitors in the first stage was increased from 0.2pF to 0.6pF. Taking into account the back plate and input switch parasitic capacitances of the first MDAC, the input capacitance of the first MDAC rose to approximately 40pF. All other capacitances in the ADC were left alone. SPICE simulations returned promising numbers. During the first MDAC's sampling phase, the input referred noise of the system was 1.37×10^{-7} V^2. During the second MDAC's sampling phase, input referred noise registered at 4.78×10^{-8} V^2. Together, the two phases contributed 26.8uV RMS input referred noise to the ADC. Noise associated with the third and following stages contributed very little to the input referred amount so this noise was not considered. The 26.8uV RMS ADC generated noise measured above just meets the target specification of 1/3 LSB.

5.6 Design for Digital Calibration

In chapter 3, the theory of digital calibration was discussed. Implementation must now be performed. Since most of the calibration is done in the digital domain, a redesign of the AD9243's digital circuitry is required for calibration to be possible. Functionality must also be added to the first stage MDAC so that calibration measurements can be made. In this design the amount of digital complexity must be kept low. The calibration must also guarantee that the ADC's performance doesn't vary much between uses. Whether calibration is performed on chip or off, all of these issues must be addressed.

Two different methods of digital calibration, self-calibration and factory calibration, can be used to calibrate the new ADC. In self-calibration, the ADC calibrates itself every time it powers up or at specific times when it is not being used. Self-calibration requires the ADC to have an on-board calibration controller. The controller increases the digital complexity of the ADC contrary to the requirement of digital simplicity. With self-calibration, DNL can change between successive calibrations. The other form of digital

calibration, factory calibration, is done by external equipment at the manufacturing site. An on-chip calibration controller is no longer needed, and digital simplicity can be attained. In factory calibration, error coefficients are stored in a non-volatile on-board memory and do not change after the initial write sequence. As a result, DNL does not fluctuate and more stringent specifications can be applied. Factory calibration meets both specifications for digital simplicity and strict specification and for this reason is used on the proposed ADC.

The proposed digital circuit overhaul must provide the following:

- Non-volatile memory for storage of error coefficients and offset value.
- Mathematical operators to pre-add error coefficients. This pre-addition is analogous to the summation term of the calibration algorithm shown in Eq. 4.5.
- Mathematical functionality so that the offset correction and pre-added error coefficients can be added to the raw digital output during normal mode operation.
- Ability to force first MDAC into calibration states.
- A path for uncorrupted ADC measurement information to reach the ADC output during calibration.
- Provide input/output interface circuitry for external DSP communication during calibration.

A digital block designed specifically to meet these requirements is shown in Figure 5.9. The block is comprised of delays, error-correction logic, a bank of electronically writable fuses, a bank of pre-adders, a selector, two adders, input/output logic, and a binary to single line-select encoder. These units along with an external DSP and some first stage MDAC modifications are all that is needed to perform a factory calibration.

The delay/correction logic can be seen in the upper right corner of Figure 5.9. The delay block makes sure that each of the Flash output words reach the correction logic at the same time. The correction logic adds the delayed words together. The last two bits of each word are overlapped with first two bits of the next in order to perform digital error correction. The output of this block is the raw ADC output code. In each Flash the output

words are binarily encoded. When added, the encoded words produce a raw output code which ranges from 0 to 262143. The flash encoding for the new ADC is shown in Figure 5.10. After digital error correction, the raw ADC code is ready to be offset corrected and DNL corrected.

Error coefficients calculated during calibration are stored in a bank of fuses. The bank of fuses constitutes the non-volatile memory in the system and can be seen at the far left in Figure 5.9. Another set of 11 fuses referred to as the offset fuse in Figure 5.9 is used

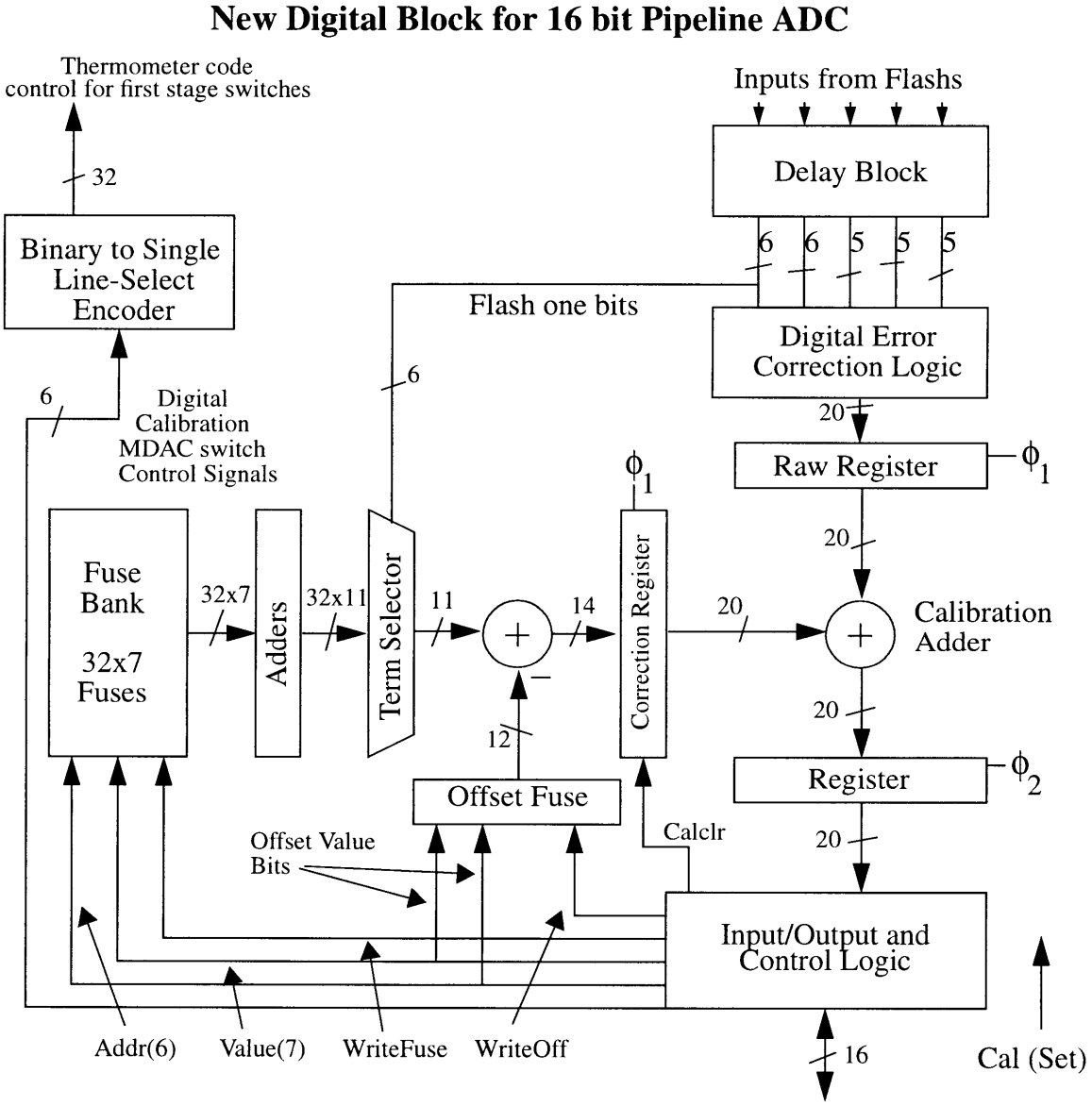


Figure 5.9

to store the offset correction term. Given the attainable capacitor matching, Initial C pro-

gram simulations proved that correction coefficients could be represented by 7 bit or narrower two's-complement words. As a result, the fuse bank is composed of an array of 32 words, each represented by 7 fuses. The fuse bank has 14 inputs. Of these 14 inputs, 7 are used to supply the block with a value, 6 are used to supply an address, and one is used to write the value to the addressed memory word. The offset fuse has 14 inputs of which 13 are used to input a value and one carries the write signal. The inputs are fed directly from the input/output logic and are controlled by an external DSP during calibration. At the output of the fuse bank is a pre-adder block. It is composed of 32 adders, the output of each given by,

$$Correctionterm(i) = \sum_{j=1}^{i} Error(j) \qquad Correctionterm(0) = 0 \qquad (5.4)$$

where Error(j) is stored at address j of the fuse bank. The outputs from the adder, each 11 bit words, are fed into a 33 to 1 selector. The bits used to select the appropriate correction term are fed directly from the output of the delay block and correspond to the first stage Flash ADC output. By using the Flash output bits directly, the appropriate selection term is always correctly selected. The ADC offset term is subtracted from the selected correction term and stored in the correction register. The correction register, during calibration, can be zeroed so that raw measurement data passing through the calibration adder is not corrupted on its way to the ADC output. During normal mode operation, the raw ADC code and appropriate correction term combine in the calibration adder to form a calibrated output code. This output code is fed into the input/output block. The correction adder and calibration adder both perform two's-complement addition and are sized to avoid overrange. Now that the digital error correction, calibration correction, and offset correction circuitry has been described, the focus can change to circuitry active during calibration.

In the first MDAC, a number of circuit changes are needed so that calibration measurements can be performed. First of all, the external DSP must be able to control the reference switches of the first MDAC individually. Each MDAC in the AD9243 has decoding circuitry that translates Flash thermometer code into reference switch signals used by the MDAC. By multiplexing 32 DSP controlled input lines with the Flash thermometer code input lines, the MDAC allows control from either the Flash or external DSP. The binary to

ROM Coding and Summation Examples

First Rom Coding	Second Rom Coding
011110	101110
011101	101101
011100	101100
011011	101011
011010	101010
011001	101001
011000	101000
010111	100111
010110	100110
010101	100101
010100	100100
010011	100011
010010	100010
010001	100001
010000	100000
001111	011111
► 001110	► 011110
001101	011101
001100	011100
001011	011011
001010	011010
001001	011001
001000	011000
000111	010111
000110	010110
000101	010101
000100	010100
000011	010011
000010	010010
000001	010001
000000	010000
111111	001111
111110	001110

Top number in each row is highest Flash ADC quantization level, while bottom is lowest. The Arrow points to the center Flash quantization level.

Third Rom Coding	Fourth Rom Coding	Fifth Rom Coding
10110	10110	11000
10101	10101	10111
10100	10100	10110
10011	10011	10101
10010	10010	10100
10001	10001	10011
10000	10000	10010
01111	01111	10001
► 01110	► 01110	► 10000
01101	01101	01111
01100	01100	01110
01011	01011	01101
01010	01010	01100
01001	01001	01011
01000	01000	01010
00111	00111	01001
00110	00110	01000

Examples of addition:

Center Quantization:

```
  001110
   011110
    01110
     01110
+    10000
-------------
 0100000000000000000
```

High Quantization:

```
  011110
   011110
    01110
     01110
+    01111
-------------
 0111111111111111111
```

Low Quantization:

```
  111110
   011110
    01110
     01110
+    10000
-------------
 0000000000000000000
```

Figure 5.10

line-select decoder in the Figure 5.9 is used to decode a binary input into the selection of a single output line. With the help of the decoder and multiplexor, the DSP, during the cali-

bration mode, has free control to select any one reference switch in the first MDAC. The DSP now has the ability to calibrate the ADC. The multiplexing scheme is illustrated in Figure 5.11.

Sampling during calibration is also important. During calibration, the MDAC samples the negative reference because this can be done with a simple attachment. If the negative supply is sampled and then a DAC transition is tripped during calibration, the first stage residue might over-range the backend converter causing an error in that measure-

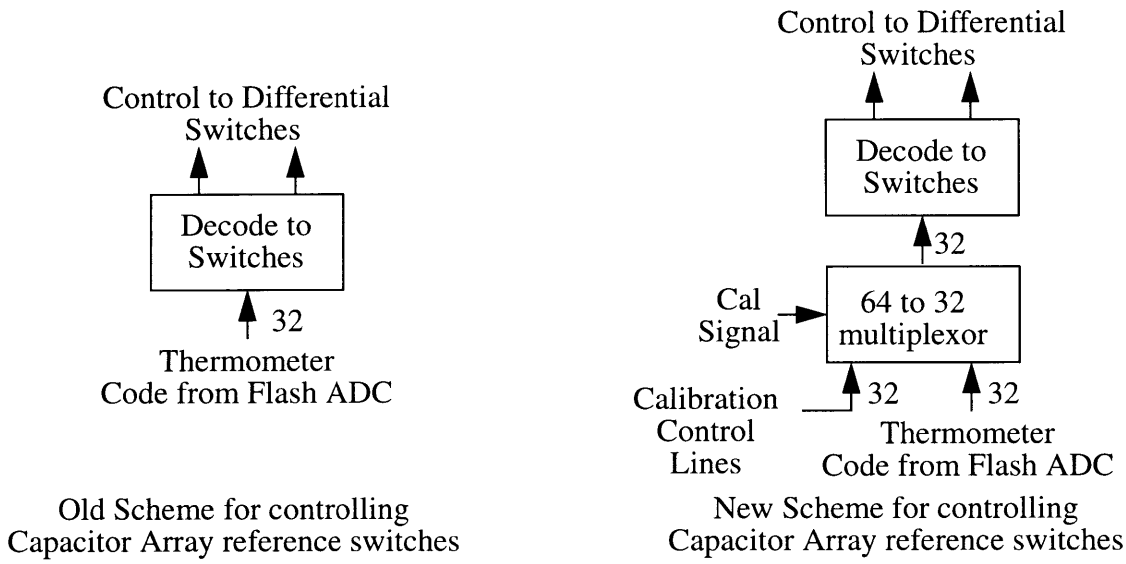


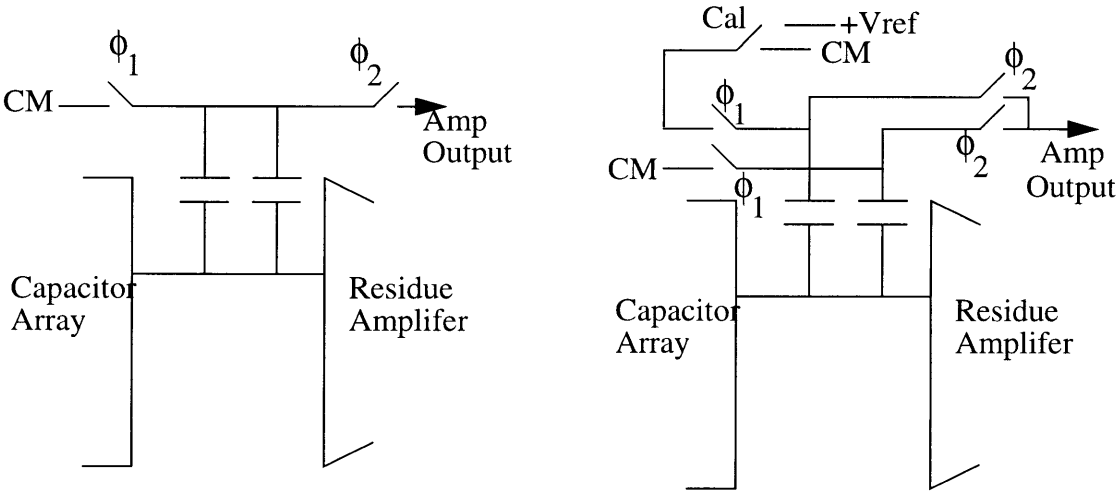
Figure 5.11

ment. This over-range is not acceptable and can cause major problems in the calibration. As mentioned in chapter 3, the stage samples the first Flash decision level during calibration avoiding over-range in the back-end ADC. During normal mode sampling, two separate feedback capacitors are sampled to ground. If one of the two feedback capacitors samples the positive reference, the residue of the MDAC is the same as if the first Flash decision level had been sampled normally. With the addition of a few switches, the MDAC's residue, during calibration, comes out to the correct voltage. The new feedback capacitor configuration can be seen in Figure 5.12. With the new modifications described above, the DSP can now calibrate the ADC without over-ranging the measuring ADC. On the other hand, the DSP does not have direct control over any of the internal digital blocks.

It actually enters commands and data though the 16 output pins of the ADC. The input/output logic is responsible for supplying the correct signals to the internal digital blocks.

The input/output block handles two distinct modes of operation. During normal mode operation, the input/output logic truncates the 18 bit output of the calibration adder to 16 bits. The logic also makes sure that the truncated code falls within the allowable digital output range (0-65535). During the calibration mode, the input/output logic acts as and interface between the external calibration DSP and the internal digital blocks. The input/output logic takes DSP commands and translates them to internal controls. During calibra-

Old and New Schemes for Feedback Capacitor Sampling



Old attachment for two feedback capacitors

New attachment with cal signal controling one feedback capacitor's sampled value

(For more MDAC detail look at Figure 5.3)

Figure 5.12

tion, the ADC 16 pin output is broken down into 13 data pins, Lsbs and Addr, and three control pins, CLK, OFF, and INOUT. One extra pin, CAL, is used to set the calibration mode. The translation table for the input/output block can be seen in Figure 5.13. This shows how the data pins can be switched between data inputs or outputs dependent on the control pin values. Together, the input/output logic and DSP perform the calibration steps outlined in Figure 5.14.

The digital circuit overhaul described in this chapter allows digital calibration to be done while keeping digital simplicity. The simplicity of this circuit has an advantage over

the digital multiplication. In the new proposed digital block, the adder blocks do a majority of the digital switching. Adder switching occurs on the positive edge of each clock phase and stops within a few nanoseconds. The settling time of each residue amplifier is 166ns. Noise coupled from these digital switching structures only occurs during first 10ns to 20ns of the amplifier settling time.As a result, each residue amplifier has plenty of time to settle after the noise component has disappeared. In each stage, the sampling instant occurs before the positive edge of the next clock phase in what is called the ADC quiet time. By sampling during the quiet time, the ADC can escape the effects of digitally coupled noise.

The changes to the AD9243 described in this chapter make it possible for linearity and resolution to be increased to the 18 and 16 bit levels respectively. These changes alone are good enough to ensure that the specifications set for the proposed converter are met. The ADC can run at 3Ms/s due to the new first stage's settling capability. Having a SHA free design, the new ADC generates only 1/3 LSB RMS input referred noise. A digital overhaul has also been done so that calibration can be performed. It must now be proven that this circuitry works in a complete converter configuration. Chapter 6 outlines simulation results for a calibrated behavioral model of the ADC. Chapter 6 also outlines some future design considerations.

Input/Output Translation Table

CAL SIGNAL HIGH

INOUT	CLK	OFF	
0	0	0	Output Measurement for Capacitor = Addr Addr -> Input, Lsbs -> Output MDAC Cal signal high, clear corr. reg. No write signals
0	0	1	Output Offset Value MDAC Cal Signal low, do not clear corr. reg Addr -> Output, Lsbs -> Output No Write Signals
0	1	0	Output Measurement for Base Case, no caps Addr -> Input, Lsbs -> Output MDAC Cal signal high, clear corr. reg. No Write Signals
0	1	1	Does Nothing
1	0	0	Input 7 bit value for writing to fuse=Addr Addr -> Input, Lsbs -> Input MDAC Cal signal high, clear corr. reg. No Write Signals
1	0	1	Input 13 bit value for writing to offset fuse Addr -> Input, Lsbs -> Input MDAC Cal signal low, do not clear corr. reg. No Write Signals
1	1	0	Write 7 bit value to fuse=Addr Addr -> Input, Lsbs -> Input MDAC Cal signal high, clear corr. reg Write Fuse, Do not Write offset
1	1	1	Write 13 bit value to offset fuse Addr -> Input, Lsbs -> Input MDAC Cal signal low, Do not clear corr. reg. Write Offset, Do not write fuse.
CAL SIGNAL LOW			Correction Register signal low MDAC Cal signal low

Addr = output bits 8-13 Addr -> Input means Addr bit are input lines
Lsbs = output bits 0-7 Lsbs-> Output means Lsbs bits are output lines

DSP has control of Lsbs, Addr, OFF, INOUT, CAL, CLK

Figure 5.13

Calibration Flow Chart of external factory DSP

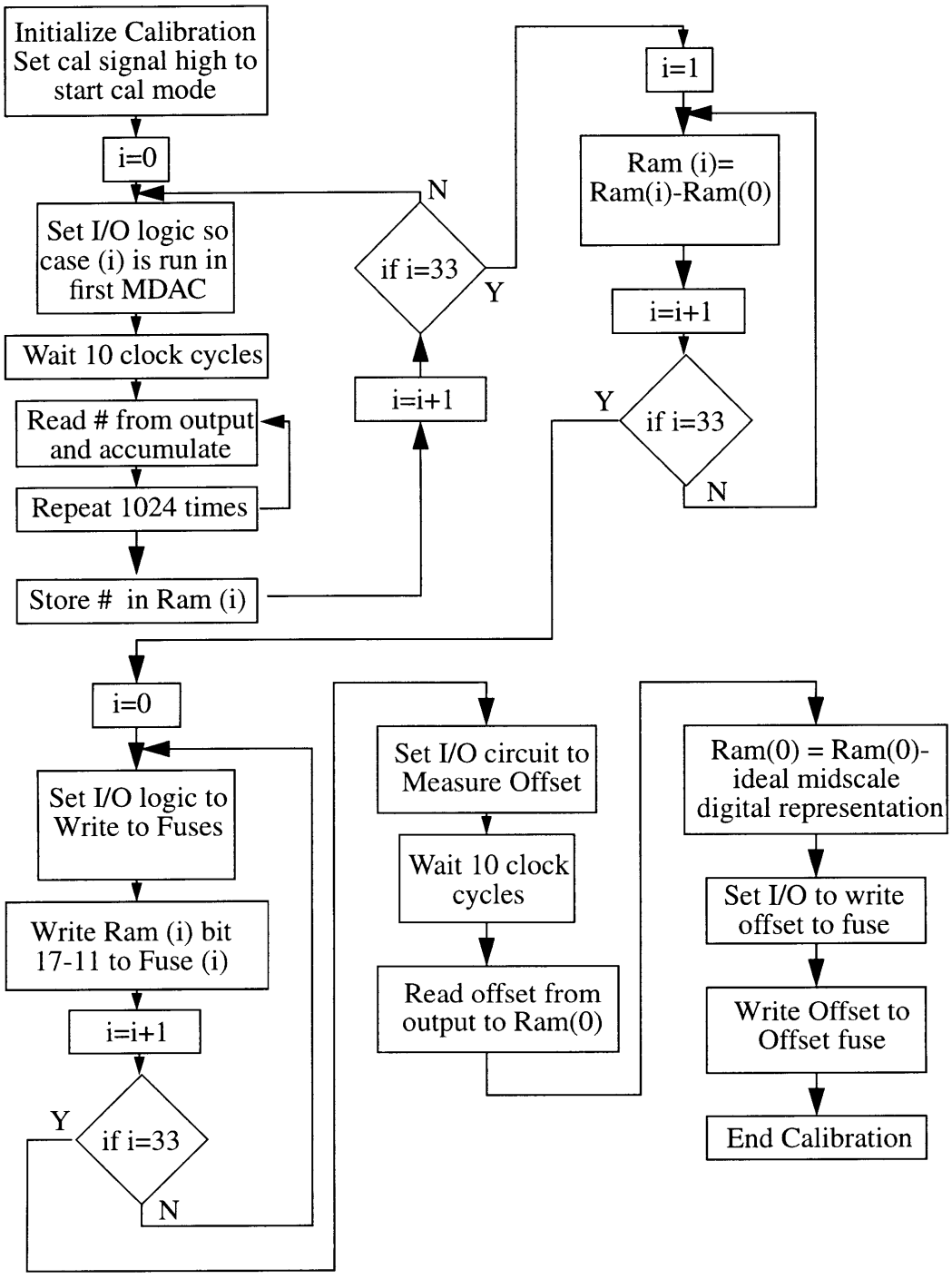


Figure 5.14

Chapter 6

Calibration Results and Conclusion

6.1 Simulation Results

Using a computer simulator, a behavioral model of the proposed ADC was constructed. This model was used to prove that calibration could truly be performed with the proposed digital circuitry in chapter 5. Four MDAC models provided the MDAC functionality for the behavioral model. Each model also represented MDAC capacitor error, offset error, and gain error. Five Flash models were used as quantization functions. The Flash models provided decision level error capability also. A block housing a Verilog representation of the new digital block was added to the ADC. This Verilog representation can be found in Appendix B. A clocking scheme analogous to a real time clocking scheme was implemented for the timing of the behavioral model. A DSP was designed using Verilog and attached to the ADC output lines. During the simulation, errors were assigned to the MDACs and Flash ADCs. A full calibration ensued and error coefficients were written to the on board memory. After the calibration steps, the DSP was detached from the ADC, and an analog simulation was performed. The analog simulation entailed ramping the analog input voltage from reference to reference while the ADC sampled 2^{20} times. The output codes were then analyzed and INL and DNL plots were extracted from the resulting transfer function. Figure 6.1 shows the ADC DNL and INL before the calibration was per-

formed. Figure 6.2 shows the ADC DNL and INL after calibration. In the proposed ADC INL and DNL were improved drastically. The DNL fit within the set specification, and the simulation proved that the new digital circuitry would work as planned.

6.2 Future Outlook

Even though the specifications set in this work have been met, a good amount of design must be done before the proposed ADC is ever released. The following are a number of considerations that should be taken into account as follow-ups to this work:

- Capacitor Non-linearity - Even though non-linearity is not taken into account in the design of this ADC, a 16 bit ADC could be a useful tool for studying the effects of capacitor non-linearity and other circuit non-linearity in the pipeline ADC. Specifically the resolution at which the non-linearity requires correction could be determined using the first silicon of this ADC. Assuming McCreary's work, [9], is valid for silicon capacitors, the highly doped characteristics in the left plot of Figure 3.17 would yield a gain error of 0.002% with little INL. The characteristics of the right plot would yield a gain error on the order of 0.0625% with estimated INL not larger than 2 LSBs. Using more heavily doped capacitor plates would definitely be necessary for reducing converter non-linearity.
- Clock Skew Between MDAC and Flash - The first MDAC and Flash sample a continuous time input signal. The Flash and MDAC should both sample at the same time. Due to clock skew, the Flash ADC could be responsible to making decisions on a different input sample than the MDAC has acquired. If this error is too large, then the ADC will over-range. The worst case timing error can be found be assuming the circuit can tolerate 78mV of Flash error and then finding what the fastest time the input signal can sweep this voltage. The maximum sweep rate of a signal oscillating at half the Nyquist rate is given by,

$$Rate = Amplitude \times 2 \times pi \times 1.5Mhz \tag{6.1}$$

In the proposed converter the amplitude is 5V and the subsequent sweep rate is $4.7\text{x}10^7$ volts/sec. Given this sweep rate, and the fact that the total error tolerance is

78mV, if the sampling instants of the MDAC and Flash differ by 78mV/4.7x10^7 volts/sec, or 1.65nS, then trouble will occur. Therefore the skew in the sampling instant between the first MDAC and the first Flash must be minimized. This can be done using careful layout techniques to minimize clock skew.

- Input Drive - The input capacitance of the proposed ADC is nearly 40pF. For high speed applications this is a hefty load capacitance for an input buffer to drive. A powerful enough input buffer must be supplied in order to drive the ADC, or noise performance can be sacrificed for increased drive capability.
- Input Noise - The noise described in chapter 5 is generated by the ADC stages only. The on-board reference buffer and external input buffer will also contribute noise to the system. In order to keep noise performance good, quiet buffers must be designed to perform these two functions.
- First Stage residue amplifier - The first stage residue amplifier has been optimized for settling performance. The issue of how well the amplifier can track its input remains. The amplifier might need to be modified so that overall ADC distortion can be kept to a minimum.

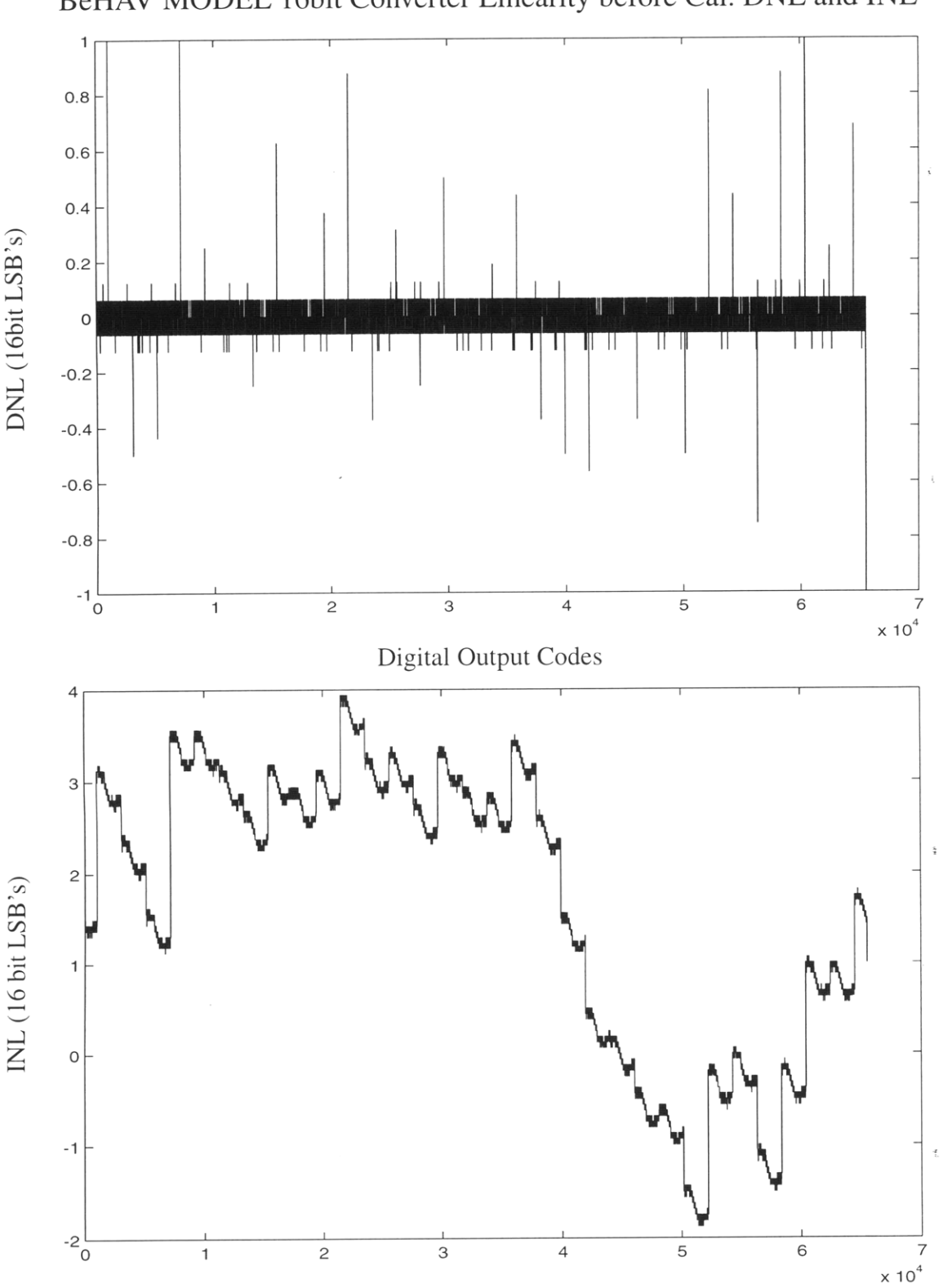

BeHAV MODEL 16bit Converter Linearity before Cal. DNL and INL
DNL (16bit LSB's)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
7
x 10^4
Digital Output Codes
INL (16 bit LSB's)
4
3
2
1
0
-1
-2
0
1
2
3
4
5
6
7
x 10^4
Digital Output Codes

Figure 6.1

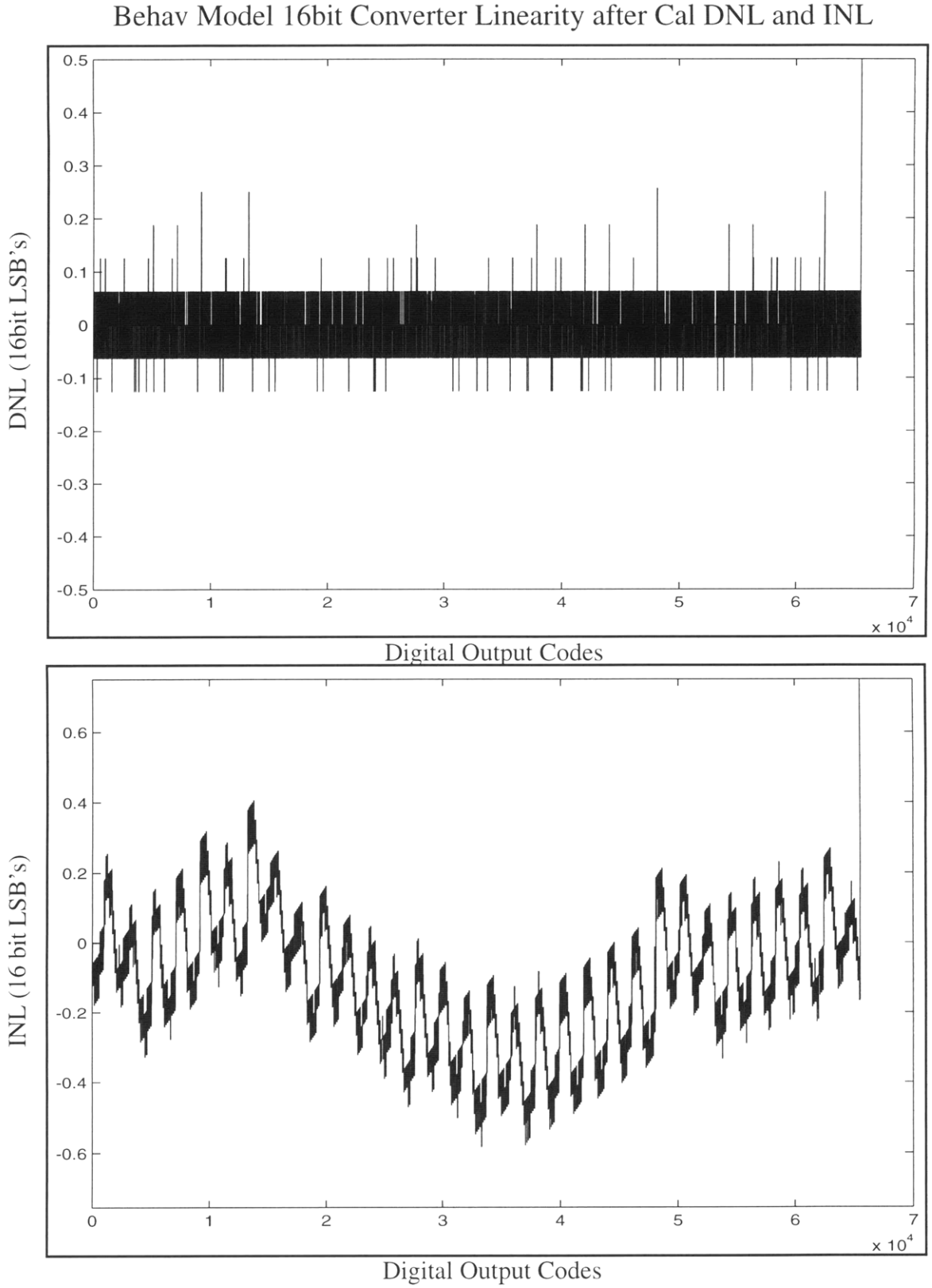

Behav Model 16bit Converter Linearity after Cal DNL and INL
DNL (16bit LSB's)
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0
1
2
3
4
5
6
7
x 10^4
Digital Output Codes
INL (16 bit LSB's)
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
0
1
2
3
4
5
6
7
x 10^4
Digital Output Codes

Figure 6.2

Appendix A

C-language Simulator for INL and DNL

The following code simulates a calibrated 5-5-4-4-4 architecture with 10 bit capacitor matching offset error, gain error, and decision spacing error. The program returns files corresponding to INL and DNL for the calibrated system.

i) Random number generator

```
/
************************************************************************
This file, generator.h contains a guassian random number generator  */
Author:Matt Courcy, Todd Brooks
WorkPlace:Analog Devices
Name:Generator.h (Random Number Generator)
************************************************************************
*/

double RANDO1(rand)
int *rand;
{
 int rand2;
 double random1;

 do{
    *rand=*rand*1103515245 + 12345;
    rand2=(unsigned int)(*rand/65536)%32768;
    random1=(double)rand2/32768.0;
   } while((random1 <=0.0) || (random1 >=1.0));
  return(random1);
}

/* GENERATE AN ARRAY OF RANDOM DATA RETURNED IN random_data. number
DESIGNATES THE NUMBER OF ELEMENTS IN THE ARRAY, AND
randseed IS THE RANDOM NUMBER SEED. THE FUNCTION RETURNS
A RANDOM INTEGER TO ITS CALL LINE, THIS CAN BE USED AS THE
RANDOM NUMBER SEED FOR THE NEXT CALL TO THIS FUNCTION. */
```

```
int generator(number, rand_seed, random_data)
int number;
int rand_seed;
double *random_data;
{
 double random1, x1, y1, z, yy1, xx1, s, l;
 double num_sigma;
 int *rand, index;

 num_sigma=12.0;

 rand=(int *) malloc(sizeof(int));

 *rand=rand_seed;
        for(index=0;  index<number; index++)
         {
          do {
              do {
                  random1=RANDO1(rand);
                  x1=random1;
                  random1=RANDO1(rand);
                  y1=2*random1-1.0;
                  xx1=x1*x1;
                  yy1=y1*y1;
                  s=xx1+yy1;
                 } while (s>1.0);
               random1=RANDO1(rand);
               l=sqrt(-2.0*log(random1))/s;
               z=(xx1-yy1)*l;
              } while (z>num_sigma || z<-num_sigma);
            *(random_data+index)=z;
         }
 return(*rand);
}
```

ii) File modeling ADC stage "stage_gen_small.h"

```
/* CALIBRATABLE RESIDUE STAGE */

void res_genc(vina, vinb, cmlevel, idealcap, topfeedcap, topsamplecap,
botfeedcap, botsamplecap, ampgain, stageoffset, residuea, residueb, dac-
code, reftop, refbot, forcedac, forcecode,order,feednumber,decrand,cal)
double vina;
double vinb;
double cmlevel;
double idealcap;
double *topfeedcap;
double *topsamplecap;
double *botfeedcap;
double *botsamplecap;
double ampgain;
double stageoffset;
double *residuea;
```

```
double *residueb;
int *daccode;
double reftop;
double refbot;
int forcedac;
int forcecode;
int order;
int feednumber;
double *decrand;
int cal;
{
 double chargetop, chargetop2;     /* Top half differential charge */
 double chargebot, chargebot2;     /* Bottom half differential charge */
 double resa;                      /* Top half differential residue */
 double resb;                     /* Bottom half differential residue */
 double vintotal;                  /* Difference between differential
                                      inputs */
 double id, low, hig;
 int i;
 double res;                     /* difference between output residues */
 double scalefactor;               /* ampgain effect on residue */
 int bits;
 double divfac;
 double topcap, botcap;

 bits=(int)ldexp(1.0,order);
 chargetop=0.0;
 chargebot=0.0;
 chargetop2=0.0;
 chargebot2=0.0;

 for(i=0; i<bits; i++)
  {
   chargetop=chargetop+(*(topsamplecap+i))*(cmlevel-vina);
   chargebot=chargebot+(*(botsamplecap+i))*(cmlevel-vinb);
  }

 vintotal=vina-vinb;
 for(i=0; i<(bits+1); i++)
  {
   id=(double)i;
   divfac=(double)(bits);
   low=((2*id/divfac)-(1.0/divfac)-1.0+((0.125/divfac)*(*(decrand+i))));
                          hig=((2*id/divfac)+(1.0/divfac)-1.0+((0.125/div-
fac)*(*(decrand+i+1))));
   low=low*(reftop-refbot);
   hig=hig*(reftop-refbot);

   if((vintotal<hig) && (vintotal>=low))
    *daccode=i;
  }

 if(forcedac)
  {
```

```
  *daccode=forcecode;
  }
if(cal==1)
 {
  for(i=0; i<(bits); i++)
  {
   if(i!=((*daccode)-1))
    {
     chargetop2=chargetop2+(*(topsamplecap+i))*(cmlevel-refbot);
     chargebot2=chargebot2+(*(botsamplecap+i))*(cmlevel-reftop);
    }
   else
    {
     chargetop2=chargetop2+(*(topsamplecap+i))*(cmlevel-reftop);
     chargebot2=chargebot2+(*(botsamplecap+i))*(cmlevel-refbot);
    }
  }
 }
else
 {
  for(i=0; i<(bits); i++)
  {
   if(i>=*daccode)
    {
     chargetop2=chargetop2+(*(topsamplecap+i))*(cmlevel-refbot);
     chargebot2=chargebot2+(*(botsamplecap+i))*(cmlevel-reftop);
    }
   else
    {
     chargetop2=chargetop2+(*(topsamplecap+i))*(cmlevel-reftop);
     chargebot2=chargebot2+(*(botsamplecap+i))*(cmlevel-refbot);
    }
  }
}
 topcap=0.0;
 botcap=0.0;

 for(i=0; i<feednumber; i++)
  {
   topcap=topcap+(*(topfeedcap+i));
   botcap=botcap+(*(botfeedcap+i));
  }

 resa=cmlevel-((chargetop-chargetop2)/(topcap));
 resb=cmlevel-((chargebot-chargebot2)/(botcap));

 scalefactor=(ampgain/(1+ampgain));

 res=scalefactor*(resa-resb);

 *residuea=cmlevel+res/2.0+stageoffset/2;
 *residueb=cmlevel-res/2.0-stageoffset/2;
}
```

iii) Modeling program including calibration and DNL and INL run "modeling.c"

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "generator.h"
#include "stage_gen_small.h"

#define NUMRUNS 1
#define INL 1
#define DNL 1

main(argc ,argv)
int argc;
char **argv;
{
 FILE *inldnlfile;

 double noiset[45000], noiseb[45000],
        noiset1[45000], noiseb1[45000],
        noiset2[45000], noiseb2[45000],
        noiset3[45000], noiseb3[45000],
        noiset4[45000],noiseb4[45000],
        topfc[10], botfc[10],
        topsc[116], botsc[116],
        ampgain[5], decideh[116],
        decidel[5], soff[5],
        reftop[5], refbot[5],
        inl[70000], dnl[70000], ifl,
        vina, vinb, resa, resb, total, ave, id, divfac,high,inh,inlow;
  int        randseed,  randseed2,  measurement[64],  correctionterm[64],
orgrand,
        index, i, index1, correct, daccode, number, number1, num1, num2,
        offset, numt, averaging;


 if(argc<3)
  exit(0);
  randseed=atoi(argv[1]);
  randseed2=atoi(argv[2]);

/* OPEN OUTPUT FILE */

 if((inldnlfile=fopen("inldnlfile","w+"))==NULL)
  {
   printf("Can't Open: inldnlfile\n");
   exit(0);
  }

/* ASSIGN ERRORS TO STAGES VARIABLES*/

  orgrand=randseed;
  randseed=generator(10,randseed,&topfc[0]);
  randseed=generator(10,randseed,&botfc[0]);
  randseed=generator(116,randseed,&topsc[0]);
```

```
 randseed=generator(116,randseed,&botsc[0]);
 randseed=generator(5,randseed,&ampgain[0]);
 randseed=generator(116,randseed,&decideh[0]);
 randseed=generator(5,randseed,&decidel[0]);
 randseed=generator(5,randseed,&soff[0]);
 randseed=generator(5,randseed,&reftop[0]);
 randseed=generator(5,randseed,&refbot[0]);

 for(index=0; index<64; index++)
  {
   correctionterm[index]=0;
   for(i=0; i<2; i++)
    measurement[index]=0;
  }

if(orgrand==-1)
 {
  for(index=0; index<10; index++)
   {
    topfc[index]=1.0;
    botfc[index]=1.0;
   }
  for(index=0; index<116; index++)
   {
    topsc[index]=1.0;
    botsc[index]=1.0;
    decideh[index]=0.0;
   }
  for(index=0; index<5; index++)
   {
    ampgain[index]=500000000.0;
    decidel[index]=0.0;
    soff[index]=0.0;
    reftop[index]=3.00;
    refbot[index]=2.00;
   }
 }
else
 {
  for(index=0; index<10; index++)
   {
    topfc[index]=1.0+(1.0/(1024.0))*topfc[index];
    botfc[index]=1.0+(1.0/(1024.0))*botfc[index];
   }
  for(index=0; index<64; index++)
   {
    if(index>31)
     {
      topsc[index]=1.0+(1.0/(1024.0))*topsc[index];
      botsc[index]=1.0+(1.0/(1024.0))*botsc[index];
      decideh[index]=0.0+0.005*decideh[index];
     }
    else
     {
```

```
      topsc[index]=1.0+(1.0/(1024.0))*topsc[index];
      botsc[index]=1.0+(1.0/(1024.0))*botsc[index];
      decideh[index]=0.0+0.005*decideh[index];
     }
    }
   for(index=64; index<116; index++)
    {
     topsc[index]=1.0+(1.0/(512.0))*topsc[index];
     botsc[index]=1.0+(1.0/(512.0))*botsc[index];
     decideh[index]=0.0+0.005*decideh[index];
    }

   for(index=0; index<5; index++)
    {
     ampgain[index]=50000.0+500.0*ampgain[index];
     decidel[index]=0.0+0.005*decidel[index];
     soff[index]=0.010*soff[index];
     reftop[index]=3.00;
     refbot[index]=2.00;
    }
  }

/* RUN CALIBRATION PROCESS WITH "averaging" TIMES AVERAGING FOR NOISE */

 correctionterm[0]=0;

 numt=0;
 averaging=1024;

 for(index=0; index<32; index++)
  {
  randseed2=generator(4096,randseed2,&noiset[0]);
  randseed2=generator(4096,randseed2,&noiseb[0]);
  randseed2=generator(4096,randseed2,&noiset1[0]);
  randseed2=generator(4096,randseed2,&noiseb1[0]);
  randseed2=generator(4096,randseed2,&noiset2[0]);
  randseed2=generator(4096,randseed2,&noiseb2[0]);
  randseed2=generator(4096,randseed2,&noiset3[0]);
  randseed2=generator(4096,randseed2,&noiseb3[0]);
  randseed2=generator(4096,randseed2,&noiset4[0]);
  randseed2=generator(4096,randseed2,&noiseb4[0]);

  for(index1=0; index1<averaging; index1++)
  {
   noiset[index1]=noiset[index1]*0.000024;
   noiseb[index1]=noiseb[index1]*0.000052;
   noiset1[index1]=noiset1[index1]*0.000052;
   noiseb1[index1]=noiseb1[index1]*0.000052;
   noiset2[index1]=noiset2[index1]*0.000052;
   noiseb2[index1]=noiseb2[index1]*0.000052;
   noiset3[index1]=noiset3[index1]*0.000052;
   noiseb3[index1]=noiseb3[index1]*0.000052;
   noiset4[index1]=noiset4[index1]*0.000052;
   noiseb4[index1]=noiseb4[index1]*0.000052;
```

```
    id=(double)index;
    divfac=32.0;
    high=((1.0/divfac)-1.0);
    inh=(0.5*high)+2.5;
    inlow=(-0.5*high)+2.5;

    res_genc(inh+noiset[index1],inlow,2.5,1.0,&topfc[0],&topsc[0],
             &botfc[0],&botsc[0],ampgain[0],soff[0],&resa,
             &resb,&daccode,reftop[0],refbot[0],1,0,5,2,&decideh[0],1);

    vina=resa;
    vinb=resb;
    number=0;
    number1=0;

    res_genc(vina+noiset1[index1],vinb,2.5,1.0,&topfc[2],&topsc[32],
             &botfc[2],&botsc[32],ampgain[1],soff[1],&resa,
             &resb,&daccode,reftop[1],refbot[1],0,0,5,2,&decideh[32],0);

    vina=resa;
    vinb=resb;
    number=daccode-2;
    number=number*8;

    res_genc(vina+noiset2[index1],vinb,2.5,1.0,&topfc[4],&topsc[64],
             &botfc[4],&botsc[64],ampgain[2],soff[2],&resa,&resb,
             &daccode,reftop[2],refbot[2],0,0,4,2,&decideh[64],0);

    vina=resa;
    vinb=resb;
    number=number+daccode+6;
    number=number*8;
    res_genc(vina+noiset3[index1],vinb,2.5,1.0,&topfc[6],
             &topsc[80],&botfc[6],&botsc[80],ampgain[3],soff[3],
                  &resa,&resb,&daccode,reftop[3],refbot[3],0,0,4,2,&deci-
deh[80],0);

    vina=resa;
    vinb=resb;
    number=number+daccode+6;
    number=number*8;
    res_genc(vina+noiset4[index1],vinb,2.5,1.0,&topfc[8],
             &topsc[96],&botfc[8],&botsc[96],ampgain[4],soff[4],&resa,
             &resb,&daccode,reftop[4],refbot[4],0,0,4,2,&decideh[96],0);

    vina=resa;
    vinb=resb;
    number=number+daccode+8;
    num1=number%128;

    res_genc(inh+noiseb[index1],inlow,2.5,1.0,&topfc[0],&topsc[0],
             &botfc[0],&botsc[0],ampgain[0],soff[0],&resa,&resb,
             &daccode,reftop[0],refbot[0],1,(index+1),5,2,&decideh[0],1);
```

```
   vina=resa;
   vinb=resb;
   number=0;
   number1=0;
   res_genc(vina+noiseb1[index1],vinb,2.5,1.0,&topfc[2],
            &topsc[32],&botfc[2],&botsc[32],ampgain[1],soff[1],&resa,
            &resb,&daccode,reftop[1],refbot[1],0,0,5,2,&decideh[32],0);

   vina=resa;
   vinb=resb;
   number=daccode-2;
   number=number*8;

   res_genc(vina+noiseb2[index1],vinb,2.5,1.0,&topfc[4],&topsc[64],
           &botfc[4],&botsc[64],ampgain[2],soff[2],&resa,&resb,&daccode,
            reftop[2],refbot[2],0,0,4,2,&decideh[64],0);

   vina=resa;
   vinb=resb;
   number=number+daccode+6;
   number=number*8;
   res_genc(vina+noiseb3[index1],vinb,2.5,1.0,&topfc[6],&topsc[80],
           &botfc[6],&botsc[80],ampgain[3],soff[3],&resa,&resb,&daccode,
            reftop[3],refbot[3],0,0,4,2,&decideh[80],0);

   vina=resa;
   vinb=resb;
   number=number+daccode+6;
   number=number*8;
   res_genc(vina+noiseb4[index1],vinb,2.5,1.0,&topfc[8],&topsc[96],
           &botfc[8],&botsc[96],ampgain[4],soff[4],&resa,&resb,&daccode,
            reftop[4],refbot[4],0,0,4,2,&decideh[96],0);

   vina=resa;
   vinb=resb;
   number=number+daccode+8;
   num2=number%128;
   numt=numt+((num1-num2));
  }

   measurement[index]=numt;
   correctionterm[index+1]=numt/averaging;
 }


/* OFFSET CALIBRATION */

   number=0;
   correct=0;

   res_genc(2.5,2.5,2.5,1.0,&topfc[0],&topsc[0],&botfc[0],&botsc[0],
    ampgain[0],soff[0],&resa,&resb,&daccode,reftop[0],refbot[0],
    0,0,5,2,&decideh[0],0);
```

```
    vina=resa;
    vinb=resb;
    number=number+daccode-1;
    number=number*16;

    correct=correct-correctionterm[daccode];
    res_genc(vina,vinb,2.5,1.0,&topfc[2],&topsc[32],&botfc[2],&botsc[32],
     ampgain[1],soff[1],&resa,&resb,&daccode,reftop[1],refbot[1],
     0,0,5,2,&decideh[32],0);

    vina=resa;
    vinb=resb;
    number=number+daccode-2;
    number=number*8;

    res_genc(vina,vinb,2.5,1.0,&topfc[4],&topsc[64],&botfc[4],&botsc[64],
     ampgain[2],soff[2],&resa,&resb,&daccode,reftop[2],refbot[2],
     0,0,4,2,&decideh[64],0);

    vina=resa;
    vinb=resb;
    number=number+daccode+6;
    number=number*8;

    res_genc(vina,vinb,2.5,1.0,&topfc[6],&topsc[80],&botfc[6],&botsc[80],
     ampgain[3],soff[3],&resa,&resb,&daccode,reftop[3],refbot[3],
     0,0,4,2,&decideh[80],0);

    vina=resa;
    vinb=resb;
    number=number+daccode+6;
    number=number*8;

    res_genc(vina,vinb,2.5,1.0,&topfc[8],&topsc[96],&botfc[8],&botsc[96],
     ampgain[4],soff[4],&resa,&resb,&daccode,reftop[4],refbot[4],
     0,0,4,2,&decideh[96],0);

    vina=resa;
    vinb=resb;
    number=number+daccode+8;
    offset=(number*averaging-correct);
    offset=offset-131072*averaging;

/* RUN CALCULATION OF LINEARITY */

 for(index=0; index<70000; index++)
  {
   dnl[index]=0.0;
   inl[index]=0.0;
  }

 for(i=0; i<1048576; i++)
  {
   ifl=((double)i)+0.5;
```

```
   vina=2.00+1.0*ifl/1048576.0;
   vinb=3.00-1.0*ifl/1048576.0;

   number=0;
   correct=0;

   res_genc(vina,vinb,2.5,1.0,&topfc[0],&topsc[0],&botfc[0],&botsc[0],
    ampgain[0],soff[0],&resa,&resb,&daccode,reftop[0],refbot[0],
    0,0,5,2,&decideh[0],0);

   vina=resa;
   vinb=resb;
   number=number+daccode-1;
   number=number*16;
   correct=correct-correctionterm[daccode];

   res_genc(vina,vinb,2.5,1.0,&topfc[2],&topsc[32],&botfc[2],&botsc[32],
    ampgain[1],soff[1],&resa,&resb,&daccode,reftop[1],refbot[1],
    0,0,5,2,&decideh[32],0);

   vina=resa;
   vinb=resb;
   number=number+daccode-2;
   number=number*8;

   res_genc(vina,vinb,2.5,1.0,&topfc[4],&topsc[64],&botfc[4],&botsc[64],
     ampgain[2],soff[2],&resa,&resb,&daccode,reftop[2],refbot[2],
    0,0,4,2,&decideh[64],0);

   vina=resa;
   vinb=resb;
   number=number+daccode+6;
   number=number*8;

   res_genc(vina,vinb,2.5,1.0,&topfc[6],&topsc[80],&botfc[6],&botsc[80],
    ampgain[3],soff[3],&resa,&resb,&daccode,reftop[3],refbot[3],
    0,0,4,2,&decideh[80],0);

   vina=resa;
   vinb=resb;
   number=number+daccode+6;
   number=number*8;

   res_genc(vina,vinb,2.5,1.0,&topfc[8],&topsc[96],&botfc[8],&botsc[96],
    ampgain[4],soff[4],&resa,&resb,&daccode,reftop[4],refbot[4],
    0,0,4,2,&decideh[96],0);

   vina=resa;
   vinb=resb;
   number=number+daccode+8;
   number1=(number*averaging-correct-offset)/(averaging*4);
  }

/* STORE DNL AND INL INTO OUTSIDE FILE NAME inldnlfile */
```

```
 number=0;
 total=0.0;
 index=0;

 for(i=(0+0+1); i<(65536); i++)
  {
   if(dnl[i]==0.0)
    index++;
   total=total+dnl[i];
  }

 ave=total/(65536.0-2.0-(double)0-(double)0-(double)index);
 inl[0]=0.0;

 for(i=(0+0+1); i<(65536); i++)
  {

   dnl[i]=dnl[i]-ave;


   if((i==0) && ((dnl[i]/ave)!=-1.0))
    {
     inl[i]=0+dnl[i];
     }
   else if((dnl[i]/ave)!=-1.0)
    {
     inl[i]=inl[i-1]+dnl[i];
    }

   if((DNL) && (NUMRUNS==1) && ((dnl[i]/ave)!=-1.0))
    fprintf(inldnlfile,"%f ",dnl[i]/ave);
   if((INL) && (NUMRUNS==1) && ((dnl[i]/ave)!=-1.0))
    fprintf(inldnlfile,"%f",inl[i]/ave);
   if(((DNL) || (INL)) && (NUMRUNS==1) && ((dnl[i]/ave)!=-1.0))
    fprintf(inldnlfile,"\n");
  }

fclose(inldnlfile);
}
```

Appendix B

Verilog Code Representation of New Digital Circuitry

This verilog code represents the entire digital circuit described in chapter 5. It does not include the external DSP algorithm, but includes the interface logic that is used by the ADC to communicate with the DSP. This file was used in the behavioral simulation leading to the results in chapter 6.

```
`timescale 1 ns / 100 ps
module digital_block
(fuse0,fuse1,fuse2,fuse3,fuse4,fuse5,fuse6,fuse7,fuse8,fuse9,fuse10,fus
e11,fuse12,fuse13,fuse14,fuse15,fuse16,fuse17,fuse18,fuse19,fuse20,fuse
21,fuse22,fuse23,fuse24,fuse25,fuse26,fuse27,fuse28,fuse29,fuse30,fuse3
1,flash0,flash1,flash2,flash3,flash4,phase1,phase2,databus,cal,calmda-
cout,lsbsout,addrout,offsetin,wroff,wrfuse);

input [6:0] fuse0;
input [6:0] fuse1;
input [6:0] fuse2;
input [6:0] fuse3;
input [6:0] fuse4;
input [6:0] fuse5;
input [6:0] fuse6;
input [6:0] fuse7;
input [6:0] fuse8;
input [6:0] fuse9;
input [6:0] fuse10;
input [6:0] fuse11;
input [6:0] fuse12;
input [6:0] fuse13;
input [6:0] fuse14;
input [6:0] fuse15;
input [6:0] fuse16;
input [6:0] fuse17;    /* 32 Input fuse registers */
input [6:0] fuse18;    /* 7 bits per fuse */
input [6:0] fuse19;
input [6:0] fuse20;
```

```
input [6:0] fuse21;
input [6:0] fuse22;
input [6:0] fuse23;
input [6:0] fuse24;
input [6:0] fuse25;
input [6:0] fuse26;
input [6:0] fuse27;
input [6:0] fuse28;
input [6:0] fuse29;
input [6:0] fuse30;
input [6:0] fuse31;
input [5:0] flash0;
input [5:0] flash1;
input [4:0] flash2;
input [4:0] flash3;
input [4:0] flash4;

input phase1;
input phase2;
inout [15:0] databus;   /* Bidirectional databus 16bits */
input cal;              /* Calibration pin */
input [12:0] offsetin;  /* Input from offset fuse */
output calmdacout;      /* Calibration Signal to Hook to first Mdac */
reg calmdacout;        /* During this signal samples are taken from the
*/
                       /* references and amplification is done by
*/
                      /* calibration controlled caps otherwise by flash
*/

output [6:0] lsbsout;   /* Output 7 lsbs of dataline used from load-
ing    */
                      /* fuses and offset fuse                        */

output [5:0] addrout;   /* Used to address fuse bank and to select
*/
                        /* capacitor to measure.  */

output wrfuse;          /* Write signal to fuse bank */
output wroff;           /* Write signal to offset fuse */
reg wrfuse;
reg wroff;
reg [6:0] lsbsout;
reg [5:0] addrout;

wire [5:0] selectline;  /* wire representing selection of correction fac-
tor */
wire [19:0] raw_in;     /* Output of correction logic = raw digital */
wire [6:0] lsbline;     /* interconnecting lsb bus */
wire [5:0] addrline;    /* interconnecting address bus */
wire wrfuseline;        /* interconnecting fuse write line */
wire [10:0] muxoutline;       /* interconnecting mux output line */
wire wroffline;         /* interconnecting offset write line */
wire [19:0] off_addoutline;   /* main_sum output line */
```

```
wire clrreg;            /* correction register clear signal only asserted
*/

/* clear during (cal AND not offset) */

always @(lsbline)
begin
 lsbsout <= lsbline;  /* When lsbline changes set output equal to new
value */
end

always @(addrline)
begin
 addrout <= addrline; /* When address line changes set output to new
value */
end

always @(clrreg)
begin
 calmdacout <= clrreg;/* When clrreg changes set calmdacout to new value
*/
/* clrreg high, sample from references amp from addressing */
/* clrreg low, sample cmlevel amp from flash */
/* clrreg low during offset calibration and normal usage */
end

always @(wrfuseline)
begin
 wrfuse <= wrfuseline;  /* When wrfuseline changes set output to new
value */
end

always @(wroffline)
begin
 wroff <= wroffline;    /* When wroffline changes set output to new value
*/
end

correction_module
F(flash0,flash1,flash2,flash3,flash4,phase1,phase2,raw_in,
selectline);
add_mux A
(fuse0,fuse1,fuse2,fuse3,fuse4,fuse5,fuse6,fuse7,fuse8,fuse9,fuse10,
fuse11,fuse12,fuse13,fuse14,fuse15,fuse16,fuse17,fuse18,fuse19,fuse20,f
use21,
fuse22,fuse23,fuse24,fuse25,fuse26,fuse27,fuse28,fuse29,fuse30,fuse31,
muxoutline,selectline);

/* Instantiate correction_module */

main_sum B (offsetin,muxoutline,phase1,phase2,raw_in,clrreg,
off_addoutline);

/* Instantiate main_sum */
```

```
output_logic C (cal,databus,off_addoutline,addrline,lsbline,wroffline,
wrfuseline,clrreg);

/* Instantiate output_logic */

endmodule


module
correction_module(flash0,flash1,flash2,flash3,flash4,phase1,phase2,out,
flash0_bits);

input [5:0] flash0;
input [5:0] flash1;
input [4:0] flash2;
input [4:0] flash3;
input [4:0] flash4;
input phase1;
input phase2;
output [19:0] out;       /* Output number : Raw digital code */

output [5:0] flash0_bits;
    /* Bits from first stage, used to select correction factor from
mux.*/

reg [19:0] out;
reg [5:0] flash0_bits;

reg [5:0] f011;
reg [5:0] f012;
reg [5:0] f013;
reg [5:0] f014;   /* Delay registers */
reg [5:0] f111;
reg [5:0] f112;
reg [5:0] f113;
reg [4:0] f211;
reg [4:0] f212;
reg [4:0] f311;

always @(phase2 or f011 or f013 or f112 or flash1 or f211 or flash3)
begin
 if(phase2)
 begin
  f012 <= f011;
  f014 <= f013;
  f111 <= flash1;  /* This block loads the appropriate delay */
  f113 <= f112;    /* registers for phase2 */
  f212 <= f211;
  f311 <= flash3;
 end
end

always @(phase1 or flash0 or flash2 or f012 or f111)
```

```
begin
 if(phase1)
 begin
  f011 <= flash0;
  f013 <= f012;    /* This block loads the appropriate delay */
  f112 <= f111;    /* registers for phase1 */
  f211 <= flash2;
 end
end

/* The following block adds the flash values together with the appropri-
ate */
/* overlap to produce an error corrected output signal */

always @(f014 or f113 or f212 or f311 or flash4 or phase1)
begin
 if(phase1)
  begin
   out <=
{1'b0,f014,13'b0}+{5'b0,f113,9'b0}+{9'b0,f212,6'b0}+{12'b0,f311,3'b0}+
{15'b0,flash4};
  end
 flash0_bits <= f014;    /* Assign selection line output */
end

endmodule

module main_sum (offsetin,mux_in,phase1,phase2,raw_in,regclr,sum_out);

input [12:0] offsetin;
input [10:0] mux_in;
input phase1;
input phase2;
input [19:0] raw_in;
input regclr;
output [19:0] sum_out;
reg [19:0] sum_out;

wire [12:0] offset;
reg [13:0] reg1;

/* Compliment offset for easy subtraction */

assign offset = ((~offsetin)+1);

/* This block simulates an adder and register */
/* If regclr is high, the the output of the   */
/* block is 0, but is regclr is low, then the */
/* the output is then mux_in-offsetin which   */
/* is the necessary function for calculating  */
/* an individual correction term      */
/* The register loads on phase1 while the     */
/* clear signal is asychronous      */
```

```
always @(regclr or offset or mux_in or phase1)
begin
 if(regclr)
  reg1 <= 14'b00000000000000;
 else if(phase1)
       reg1 <= {mux_in[10],mux_in[10],mux_in[10],mux_in}+{offset[12],
offset};
end

/* This block adds the correction factor to   */
/* raw digital number. The register here is   */
/* loaded of phase2      */

always @(reg1 or phase2 or raw_in)
begin
 if(phase2)
  sum_out <=
{reg1[13],reg1[13],reg1[13],reg1[13],reg1[13],reg1[13],reg1} + raw_in;
end
endmodule

module add_mux
(fuse0,fuse1,fuse2,fuse3,fuse4,fuse5,fuse6,fuse7,fuse8,fuse9,fuse10,fus
e11,fuse12,fuse13,fuse14,fuse15,fuse16,fuse17,fuse18,fuse19,fuse20,fuse
21,fuse22,fuse23,fuse24,fuse25,fuse26,fuse27,fuse28,fuse29,fuse30,fuse3
1,mux_out,select);

input [6:0] fuse0;
input [6:0] fuse1;
input [6:0] fuse2;
input [6:0] fuse3;
input [6:0] fuse4;
input [6:0] fuse5;
input [6:0] fuse6;
input [6:0] fuse7;
input [6:0] fuse8;
input [6:0] fuse9;
input [6:0] fuse10;
input [6:0] fuse11;
input [6:0] fuse12;
input [6:0] fuse13;       /* 32 input fuses 7 bits each */
input [6:0] fuse14;
input [6:0] fuse15;
input [6:0] fuse16;
input [6:0] fuse17;
input [6:0] fuse18;
input [6:0] fuse19;
input [6:0] fuse20;
input [6:0] fuse21;
input [6:0] fuse22;
input [6:0] fuse23;
input [6:0] fuse24;
input [6:0] fuse25;
input [6:0] fuse26;
```

```
input [6:0] fuse27;
input [6:0] fuse28;
input [6:0] fuse29;
input [6:0] fuse30;
input [6:0] fuse31;
input [5:0] select;

output [10:0] mux_out;      /* Single mux output line */
reg [10:0] mux_out;

wire [6:0] addout0;
wire [7:0] addout1;
wire [8:0] addout2;
wire [8:0] addout3;
wire [8:0] addout4;
wire [8:0] addout5;
wire [8:0] addout6;
wire [9:0] addout7;
wire [9:0] addout8;
wire [9:0] addout9;
wire [9:0] addout10;
wire [9:0] addout11;
wire [9:0] addout12;        /* Output of 32 adders */
wire [9:0] addout13;        /* Used for summing 32 input fuses */
wire [9:0] addout14;
wire [10:0] addout15;
wire [10:0] addout16;
wire [10:0] addout17;
wire [10:0] addout18;
wire [10:0] addout19;
wire [10:0] addout20;
wire [10:0] addout21;
wire [10:0] addout22;
wire [10:0] addout23;
wire [10:0] addout24;
wire [10:0] addout25;
wire [10:0] addout26;
wire [10:0] addout27;
wire [10:0] addout28;
wire [10:0] addout29;
wire [10:0] addout30;
wire [10:0] addout31;

/
**************************************************************************
*/
/*This following block accumulates the values of the fuses */
/*adder0-fuse0, adder1=adder0+fuse1, adder2=adder1+fuse2 etc. */
/
**************************************************************************
*/

assign addout0=fuse0;
assign addout1={addout0[6],addout0}+{fuse1[6],fuse1};
```

```
assign addout2={addout1[7],addout1}+{fuse2[6],fuse2[6],fuse2};
assign addout3=addout2+{fuse3[6],fuse3[6],fuse3};
assign addout4=addout3+{fuse4[6],fuse4[6],fuse4};
assign addout5=addout4+{fuse5[6],fuse5[6],fuse5};
assign addout6=addout5+{fuse6[6],fuse6[6],fuse6};
assign addout7={addout6[8],addout6}+{fuse7[6],fuse7[6],fuse7[6],fuse7};
assign addout8=addout7+{fuse8[6],fuse8[6],fuse8[6],fuse8};
assign addout9=addout8+{fuse9[6],fuse9[6],fuse9[6],fuse9};
assign addout10=addout9+{fuse10[6],fuse10[6],fuse10[6],fuse10};
assign addout11=addout10+{fuse11[6],fuse11[6],fuse11[6],fuse11};
assign addout12=addout11+{fuse12[6],fuse12[6],fuse12[6],fuse12};
assign addout13=addout12+{fuse13[6],fuse13[6],fuse13[6],fuse13};
assign addout14=addout13+{fuse14[6],fuse14[6],fuse14[6],fuse14};
assign addout15={addout14[9],addout14}+{fuse15[6],fuse15[6],fuse15[6],
                fuse15[6],fuse15};
assign
addout16=addout15+{fuse16[6],fuse16[6],fuse16[6],fuse16[6],fuse16};
assign
addout17=addout16+{fuse17[6],fuse17[6],fuse17[6],fuse17[6],fuse17};
assign
addout18=addout17+{fuse18[6],fuse18[6],fuse18[6],fuse18[6],fuse18};
assign
addout19=addout18+{fuse19[6],fuse19[6],fuse19[6],fuse19[6],fuse19};
assign
addout20=addout19+{fuse20[6],fuse20[6],fuse20[6],fuse20[6],fuse20};
assign
addout21=addout20+{fuse21[6],fuse21[6],fuse21[6],fuse21[6],fuse21};
assign
addout22=addout21+{fuse22[6],fuse22[6],fuse22[6],fuse22[6],fuse22};
assign
addout23=addout22+{fuse23[6],fuse23[6],fuse23[6],fuse23[6],fuse23};
assign
addout24=addout23+{fuse24[6],fuse24[6],fuse24[6],fuse24[6],fuse24};
assign
addout25=addout24+{fuse25[6],fuse25[6],fuse25[6],fuse25[6],fuse25};
assign
addout26=addout25+{fuse26[6],fuse26[6],fuse26[6],fuse26[6],fuse26};
assign
addout27=addout26+{fuse27[6],fuse27[6],fuse27[6],fuse27[6],fuse27};
assign
addout28=addout27+{fuse28[6],fuse28[6],fuse28[6],fuse28[6],fuse28};
assign
addout29=addout28+{fuse29[6],fuse29[6],fuse29[6],fuse29[6],fuse29};
assign
addout30=addout29+{fuse30[6],fuse30[6],fuse30[6],fuse30[6],fuse30};
assign
addout31=addout30+{fuse31[6],fuse31[6],fuse31[6],fuse31[6],fuse31};

/
**************************************************************************
*/
/*This following block preforms the function of multiplexor upon*/
/*the 32 inputs to the block. The output is selected using the */
/*select signal.*/
```

```
/
**************************************************************************
*/

always @(addout0 or addout1 or addout2 or addout3 or addout4 or addout5
or addout6 or addout7 or addout8 or addout9 or addout10 or addout11 or
addout12 or addout13 or addout14 or addout15 or addout16 or addout17 or
addout18 or addout19 or addout20 or addout21 or addout22 or addout23 or
addout24 or addout25 or addout26 or addout27 or addout28 or addout29 or
addout30 or addout31 or select)
begin
 case (select)
  0 : mux_out <= {addout0[6],addout0[6],addout0[6],addout0[6],addout0};
  1 : mux_out <= {addout1[7],addout1[7],addout1[7],addout1};
  2 : mux_out <= {addout2[8],addout2[8],addout2};
  3 : mux_out <= {addout3[8],addout3[8],addout3};
  4 : mux_out <= {addout4[8],addout4[8],addout4};
  5 : mux_out <= {addout5[8],addout5[8],addout5};
  6 : mux_out <= {addout6[8],addout6[8],addout6};
  7 : mux_out <= {addout7[9],addout7};
  8 : mux_out <= {addout8[9],addout8};
  9 : mux_out <= {addout9[9],addout9};
  10 : mux_out <= {addout10[9],addout10};
  11 : mux_out <= {addout11[9],addout11};
  12 : mux_out <= {addout12[9],addout12};
  13 : mux_out <= {addout13[9],addout13};
  14 : mux_out <= {addout14[9],addout14};
  15 : mux_out <= addout15;
  16 : mux_out <= addout16;
  17 : mux_out <= addout17;
  18 : mux_out <= addout18;
  19 : mux_out <= addout19;
  20 : mux_out <= addout20;
  21 : mux_out <= addout21;
  22 : mux_out <= addout22;
  23 : mux_out <= addout23;
  24 : mux_out <= addout24;
  25 : mux_out <= addout25;
  26 : mux_out <= addout26;
  27 : mux_out <= addout27;
  28 : mux_out <= addout28;
  29 : mux_out <= addout29;
  30 : mux_out <= addout30;
  31 : mux_out <= addout31;
  default : mux_out <= 11'b00000000000;
 endcase
end

endmodule

module output_logic (cal,data,word_in,addr,lsbs,wroff,wrfuse,clrreg);

input cal;
inout [15:0] data;
```

```
input [19:0] word_in;
output [5:0] addr;
output [6:0] lsbs;
output wroff;
output wrfuse;
output clrreg;
reg clrreg;
reg [15:0] data_reg;
wire [15:0] data = data_reg;
reg [5:0] addr;
reg [6:0] lsbs;
reg wroff;
reg wrfuse;
wire offset;
wire io;
wire clk;

/* This assign block assigns all control signals from the bidirectional
bus */
/* These control signals are to be used when cal is high */

assign offset = data[15];
assign clk = data[14];
assign io = data[13];


/* This block is used to assign the write and clear signals for the dig-
ital */
/* These signals write fuses, offsets, or clear the correction regis-
ter     */
/* Based on the control signals */

always @(offset or clk or io or cal)
begin
 wroff <= (io & clk & offset & cal);
 wrfuse <= (io & clk & ~offset & cal);
 clrreg <= (cal & ~offset);
end

/* This block is manipulation of the output bus */
/* If cal is low then normal operation happens  */
/* The output is a 16bit number which is limited */
/* between all 1's and all 0's, if too small all */
/* 0`s is the condition, and if too high all 1's */
/* is the condition. */

/* In cal mode, if io is high then the bottom 13 */
/* pins (addr,lsbs) is set by external source and */
/* read into the registers lsbs and addr         */
/* lsbs and addr are directed to the offset fuse */
/* and the fuse bank. Addr addresses the proper fuse */
/* in the fuse bank while addr is the msbs of the offset */
/* fuse. When clk is asserted, depending whether offset  */
/* is high or low, a fuse is written. i.e. */
```

```
/* offset = low on clk fuse "addr" is written with the value "lsbs" */
/* offset = high on clk offset fuse is written with the value "addr,lsbs"
*/
/* In cal mode if offset is high and io is low   */
/* bottom 13 pins are used as outputs to         */
/* so an external circuit can manipulate measurements */
/* If cal mode and offset is low and io is low   */
/* Then lsbs represent a measurement output and addr */
/* is an input address. The input address is used to */
/* select the cap to be measured, and the lsbs */
/* correspond to the measurement made on that cap */

always @(cal or data[12:0] or word_in or io or offset or clk)
begin
 if(cal)
  begin
   if(io)
    begin
     data_reg <= 16'bz;
     if(offset)
      begin
       addr <= data[12:7];
       lsbs <= data[6:0];
      end
     else
      begin
       addr <= {1'b0,data[12:8]};
       lsbs <= data[6:0];
      end
    end
   else
    begin
     if(offset)
      begin
       data_reg <= {3'bz,word_in[12:0]};
       addr <= 6'bz;
       lsbs <= 7'bz;
      end
     else
      begin
       lsbs <= 8'bz;
       if(clk)
        addr <= {1'b0,data[12:8]};
       else
        addr <= {1'b0,data[12:8]}+1;
       data_reg <= {8'bz,word_in[7:0]};
      end
    end
  end
 else
  begin
   if(word_in[19:17]==2)
    data_reg <= 16'b1111111111111111;
   else if(word_in[19:17]==3)
```

```
    data_reg <= 16'b0000000000000000;
   else
    data_reg <= word_in[17:2];
   lsbs <= 7'bz;
   addr <= 6'bz;
  end
end

endmodule
```

Bibliography

[1] K.Y. Kim, N.Kusayanagi, A.A. Abidi, "A 10b 100MS/s CMOS A/D Converter," IEEE Journal of Solid State Circuits, Vol.32, no. 3, pp. 302-311, March 1997.

[2] F.Murden and R. Gosser, "A 12b 50MS/s twostage A/D Converter," in International Solid State Circuits Conference, San Francisco, Ca., 1995, pp. 278-279

[3] L. Singer and T.Brooks, "A 14-bit 10-MHz Calibration Free CMOS Pipelined A/D Converter," 1996 Symposium on VLSI Circuits, Honolulu HAWAII, 1996, pp. 94-95.

[4] T. Brooks, D. Robertson, D. Kelly, A. Del Muro, S. Harston, "A 16bit SIGMA DELTA Pipeline ADC with 2.5MHz output Data Rate," in International Solid State Circuits Conference, Salon, 1997, pp. 208-209.

[5] D. A. Mercer, "A 14-b 2.5MSPS Pipelined ADC with on Chip EPROM," IEEE Journal of Solid State Circuits, vol. 31, no. 1, pp. 70-76, Jan. 1996.

[6] M. Mayes and S. W. Chin, "A 200mW, 1Msample/s, 16-b Pipelined A/D Converter with On-Chip 32-b Microcontroller," IEEE Journal of Solid State Circuits, vol. 31, no. 12, pp. 1862-1872.

[7] B. Razavi, *Data Conversion System Design*, IEEE Press, 1995.

[8] H. P. Tuinhout, H. Elzinga, J. T. Brugman, F. Postma, "The Floating Gate Measurement Technique for Characterization of Capacitor Matching" IEEE Transactions on Semiconductor Manufacturing, Vol. 9, No. 1, February 1996.

[9] J. L. McCreary, "Matching properties, and voltage and temperature dependence of MOS capacitors," IEEE Journal of Solid State Circuits, vol. SC-16, pp. 608-616, Dec 1981.

[10] A. Karanicolas, H. S. Lee, and K. Bacrania, "15-b 1-Msample/s digital self-calibrated pipelined ADC," IEEE Journal of Solid State Circuits, vol. 28, no. 12, pp.1207-1215, Dec. 1993.

[11] Seung-Hoon Lee, "Code-Error Calibration Techniques for Two-Step Flash Analog-to-Digital Converters," Ph.D. Dissertation, University of Illinois, Urbana, IL.

[12] S. H. Lee and B. S. Song, "Digital-Domain Calibration of Multi-Step Analog-to-Digital Converters", IEEE Transactions of Circuits and Systems, vol. 27, no. 12, pp. 1679-1688, Dec. 1992.

[13] S. Ho "Design of a 10-bit 10Ms/s Pipelined A/D Converter," Master's Thesis, Department of Computer Science and Electrical Engineering, Massachusetts Intitute of Technology, Cambridge, MA.

[14] P. R. Gray and R. G. Meyer, *Analysis and DEisng of Analog Integrated Circuits*, Wiley 1984.

[15] D. J. Allstot and W. C. Black, Jr. "Technological Deisng Considerations for Monolithic MOS Switched-Capacitor Filtering Systems" (Proceedings of the IEEE, Auguest 1983)

[16] Gregorian, *Analog MOS Integrated Circuits for Signal Processing*, Wiley 1986.