Pipelined Multi-step Interpolating A/D Converter

by

Edmond Patrick Coady

38

Pipelined Multi-step Interpolating A/D Converter

by

Edmond Patrick Coady

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

Master of Science

and

Bachelor of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May, 1995

Signature [illegible] ....
Department of Electrical Engineering and Computer Science
May 12, 1995

Certified by [illegible] ....
Professor James K. Roberge
Department of Electrical Engineering and Computer Science
Thesis Advisor

Certified by [illegible] ....
Katsufumi Nakamura
Senior Design Engineer
Thesis Advisor

Pipelined Multi-step Interpolating A/D Converter

by

Edmond Patrick Coady

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 1995 in partial fulfillment of the requirements for the degrees of
Master of Science
and
Bachelor of Science
in Electrical Engineering and Computer Science.

Abstract

This work investigates the feasibility of an interpolating approach to pipelined multi-step analog-to-digital conversion. The proposed architecture aims to ease the severe constraints imposed on amplifier design in standard pipelined multi-step architectures. By easing the amplifier performance criteria, circuits for low voltage supply and low power operation are feasible.

The design of a 10 bit, 20MSPS ADC is presented to illustrate the benefits of the pipelined multi-step interpolating converter. The chosen architecture is a 4-residue interpolator with a 5-stage pipeline. The first 4 stages resolve 3 bits each and the last stage resolves 2 bits. The circuit functions are implemented in 0.6μm CMOS using primarily switched-capacitor techniques. By taking advantage the reduced gain requirements of the amplifiers, the circuitry can operate on a 2.5V supply. The analog circuitry dissipates a simulated 65mW while achieving <1/2 LSB DNL performance at 10-bits.

Thesis Supervisor: Katsufumi Nakamura
Title: Senior Design Engineer

Thesis Supervisor: James K. Roberge
Title: Professor of Electrical Engineering

Acknowledgments

I would like to thank the Video and Image Processing group at Analog Devices for serving as patient and willing mentors. It was the VIP group's excitement about analog IC design which spurred my own interests. Todd Brooks, Stacy Ho, Chris Mangelsdorf and Larry Singer deserve special thanks for their valuable help and suggestions. I would like to especially thank Katsu Nakamura for providing the inspiration for this research. This work would not have been possible without his creativity and encouragement. Also, thanks to Professor Jim Roberge for providing valuable advice on my thesis draft. Finally, all my gratitude to my Mother and Father who ultimately made this all possible.

This research was supported by Analog Devices, Inc.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Many techniques have been developed to perform analog-to-digital conversion. None of these available techniques are suitable for all applications. Depending on speed, accuracy, power requirements, and other considerations, various architectures present specific advantages. This research concentrates on an architecture which is suitable for moderate resolution (8 to 12 bits) and samples at rates beyond 20 mega-samples per second (MSPS). Converters in this realm are typically needed for video applications at about 20MSPS or ultrasound/professional video applications at sampling rates of 40MSPS.

Most ADC architectures are similar in the respect that they require comparison, amplification and/or digital-to-analog conversion. The performance required from these circuit blocks depends largely on the selected A/D converter architecture. In CMOS technologies, most high-speed / moderate resolution converters are based on a multi-step approach. These architectures typically utilize inter-stage amplification cells that require extreme accuracy. As supply voltages drop, the design of high-gain, wide-bandwidth amplifiers can become extremely difficult, if not altogether impossible. This research explores an alternative architecture which aims to relax the demanding performance specifications on amplifiers. In addition to reducing the performance demands on the amplifiers, it is important to minimize the overall power dissipation. Based on a 0.6μm double-poly double-metal (DPDM) process, the feasibility of a 10 bit 20MSPS converter is investigated. The design demonstrates achievable performance with 2.5V analog and digital supplies, while dissipating less than 100mW.

1.2 Organization

Following the introduction, Chapter 2 provides an overview of many of the available ADC architectures. The chapter is also intended to provide insight into the constraints which ultimately limit the viability of various architectures. Chapter 2 makes clear the various advantages each architecture presents. This background is useful because some of the desirable aspects of the various architectures are pulled into the architecture that is ultimately presented.

Chapter 3 discusses the idea of multi-step interpolation. The particular advantages of this architecture are analyzed. Specifics of implementation are discussed which ultimately lead to the selection of a viable architecture. This chapter serves as the motivation for the analysis done in later chapters.

The specific implementation of interpolation using CMOS technology is presented in Chapter 4. The advantages of switched-capacitor approaches to interpolation in CMOS are discussed. The effects of non-idealities on circuit performance are also analyzed in some depth.

Chapter 5 illustrates the CMOS transistor level design of the amplifier topology used in the switched-capacitor interpolation presented in Chapter 4. The reasons for selecting the particularly amplifier topology are discussed and justified. The circuits' simulated settling results are presented to demonstrate its viability.

In Chapter 6, all of the pieces from the previous chapters are pulled together into a design example. Implementation specific issues and design issues are discussed. The ADC achieves simulated performance of 10 bits at 2.5V supplies. The linearity performance is simulated using a top-level behavioral model (given in appendix A).

Chapter 7 summarizes the conclusions of this work, and provides suggestions for further research.

Chapter 2

Nyquist-Rate A/D Architectures

2.1 Introduction

There are several architectures suitable for high-speed, moderate-resolution analog-to-digital conversion. The particular data converter application will dictate the process technology and architecture chosen. Further fine-tuning of the architecture will take place based on the process selection. It is clear that there is no one optimal architecture for all applications. All of the architectures have specific performance advantages which must be weighed and traded off based on the desired performance goals.

This chapter surveys three classes of data converters. A flash converter is presented first and is the most common and familiar class of A/D converters. Interpolating flash converters are discussed next, and are an extension of the standard flash converter. Both the flash and interpolating flash are characterized as one-step converters. A class of converters known as multi-step converters will be illustrated using a pipelined multi-step architecture. This overview of some available architectures will highlight the advantages and drawbacks of each class of converter. The chapter will begin, however, with a description of some of the performance metrics commonly used in the characterization of A/D converters.

2.2 Performance Metrics

Several performance specifications are useful in determining the relative merit of competing analog-to-digital converter architectures. The static and dynamic error definitions [1] below are not a complete listing, but a sufficient sampling to describe important performance characteristics of architectures discussed in this chapter.

• Differential nonlinearity (DNL) - Defined as the deviation of actual code width from the ideal code width. Usually measured in least significant bits (LSBs).

• Integral nonlinearity (INL) - Defined as the deviation of the converter transfer function from the ideal straight line. INL is typically measured with respect to the "best fit" straight line so as not to capture overall converter gain errors which are not considered nonlinearities. Again this is measured in LSBs.

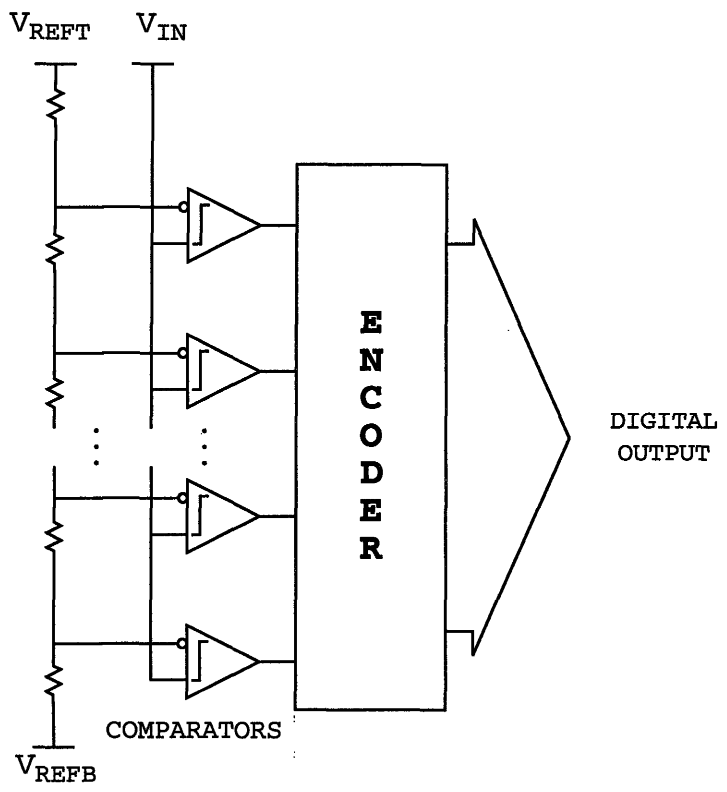

Figure 2.1. Flash architecture.

• Offset - Defined as the amount (in LSBs) the converter transfer function is shifted from the ideal converter transfer function. Offset is usually not a particularly important specification because software or hardware following the converter can easily correct for converter offset.

• Gain error - The variation of the slope of the converter transfer function from the ideal. Gain error is typically specified as a percentage variation from the ideal. Again, this is usually not a particularly important specification because it is easily correctable with software or hardware calibration.

• Signal-to-noise ratio (SNR) - Defined as the ratio of the power of a full-scale sinewave to the power of the noise of the converter. The SNR is limited in an ideal converter by the quantization noise floor (given by $SNR = 6.02N + 1.8dB$ for a N-bit converter).

• Total harmonic distortion (THD) - The ratio of the power of all harmonics to the power of the fundamental signal component.

2.3 Flash Architecture

Flash (or fully parallel) architectures take full advantage of parallelism to realize extremely high conversion rates. A block diagram of a flash converter is shown in Figure 2.1.

For a n-bit resolution converter, a flash converter requires 2^n-1 comparators. Based on this requirement, the power and area of this architectures grows exponentially with the resolution of the converter. This architecture results in an extremely large input capacitance which can create the need for a high-performance buffer amplifier at the input of the converter. For these reasons, flash converters are generally restricted to less than 10-bit applications. Also, timing differences between comparators can also create conversion errors (for high frequency input signals) which are remedied by the addition of a high-performance S/H amplifier.

Flash architectures are particularly undesirable in CMOS technologies. MOS transistors exhibit poor matching properties and limited transconductance. Because comparator offset translates directly into DNL errors, auto-zeroed comparators are required. These preamplifiers can require considerable design effort and consume additional power and area. Auto-zeroed preamplifiers tend to be slower than open loop preamplifiers that can be used in a bipolar process. For these reasons, high-speed flash converters are typically designed for bipolar processes (taking advantage of the excellent matching properties and high-speeds attainable with bipolar devices). Using bipolar technology, 8-bit performance has been obtained at 250MSPS [2] and 500MSPS [3]. The need for integrated CMOS systems with data converters has driven several high-speed CMOS flash designs (e.g. [4],[5], and [6]), in spite of the disadvantages of this technology.

2.4 Interpolating Flash Architecture

The interpolating flash converter is a direct modification of the flash converter. A section of an interpolating flash with 4x interpolation is illustrated in Figure 2.2. Like the flash converter, this is a one-step architecture. Figure 2.3 shows the transfer functions of two preamplifiers in an interpolating flash. These signals are known as "residues," because they are the residual of the input subtracted from the quantization level. Between these transfer functions or "residues" are the interpolated signals. The latches following the pre-amplifiers and their interpolated residues have decision thresholds at 0. This architecture will produce the familiar "thermometer" code output that is seen in flash converters.

2.4.1 Offset Sensitivity

Interpolating converters are inherently desensitized to amplifier offset errors. Both

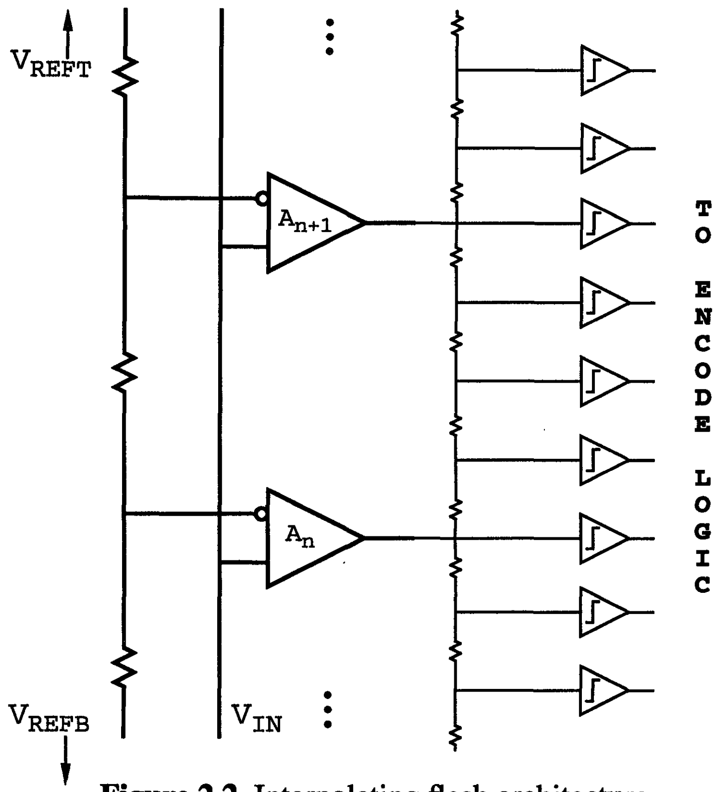

Figure 2.2. Interpolating flash architecture.

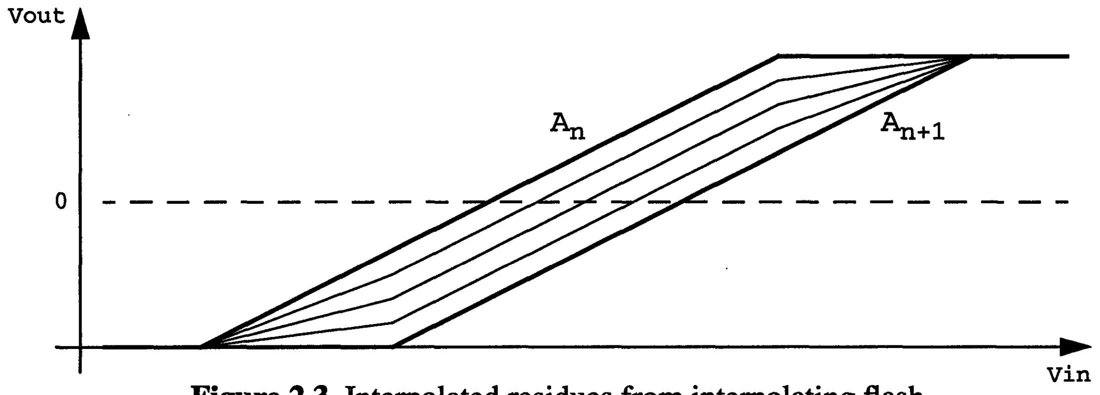

Figure 2.3. Interpolated residues from interpolating flash.

capacitive [7] and resistive implementations [8] have taken advantage of this offset insensitivity. Resistive interpolation provides an added benefit linearizing the amplifiers by causing “error currents” to flow (from the preamplifiers to the resistive ladder) to further correct amplifier offset beyond the benefit of interpolation [9].

Figure 2.4 illustrates the mechanism by which this architecture is desensitized to preamplifier offset. The dark solid lines represent 4 preamplifier transfer functions (amplifier saturation is left out for simplicity). In this figure, amplifier A2 has an input-

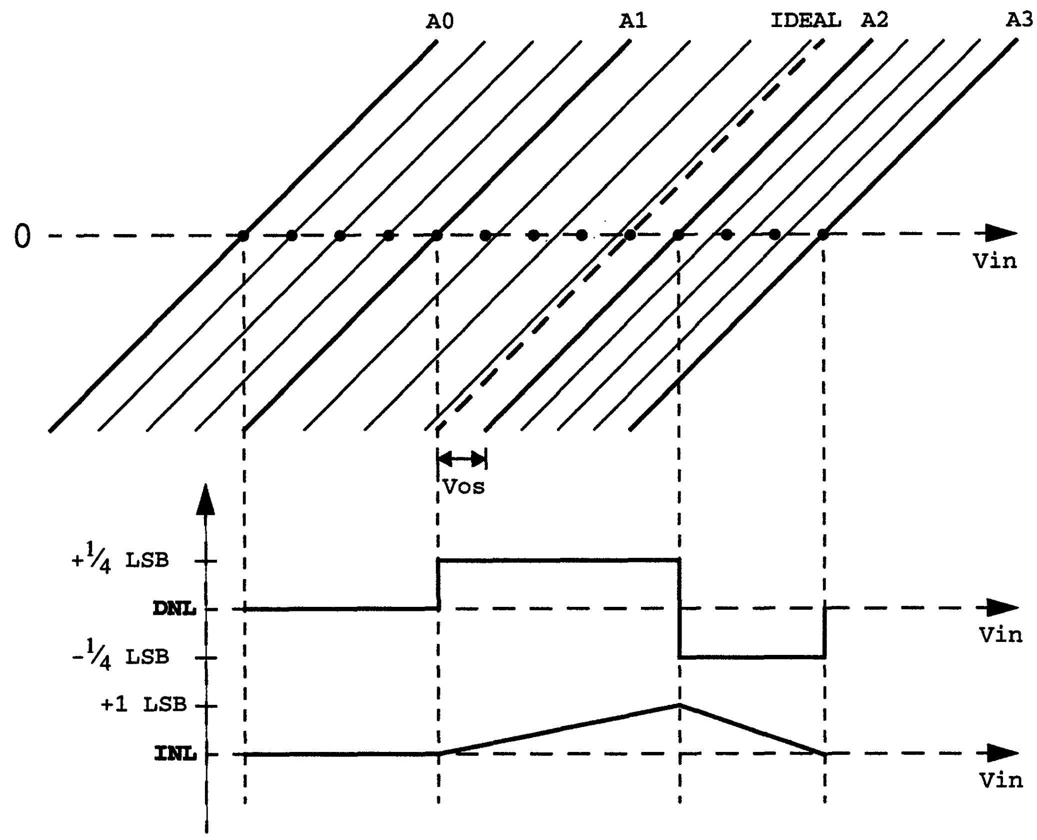

Figure 2.4. Linearity errors caused by pre-amplifier offset.

referred offset of +1 LSB. The dashed transfer function represents the ideal transfer function of amplifier A2 and the dots on the horizontal axis represent the ideal location of the residue "zero-crossings." The location of zero-crossings is the important feature, because their location determines the location of the latch transition. In this example, an 1 LSB offset error creates banks of codes that are ¼ LSB wide and ¼ LSB narrow. In a flash converter, 1 LSB of offset creates a 1 LSB DNL error (a missing code). From Figure 2.4 it can be seen that the offset's effect is spread evenly among the interpolated signals. If this example were extended to 8x interpolation, the maximum DNL error would be further reduced to ⅛ LSB. This architecture does not improve the INL performance over that of a flash converter.

Practically, the maximum interpolation factor is limited by the linear range of the preamplifier. Accurate interpolation will no longer be achieved as one of the amplifiers nears its saturation region. This requirement limits the preamplifier gain that can be realized in this implementation. If a gain is chosen to be too large, the adjacent amplifiers

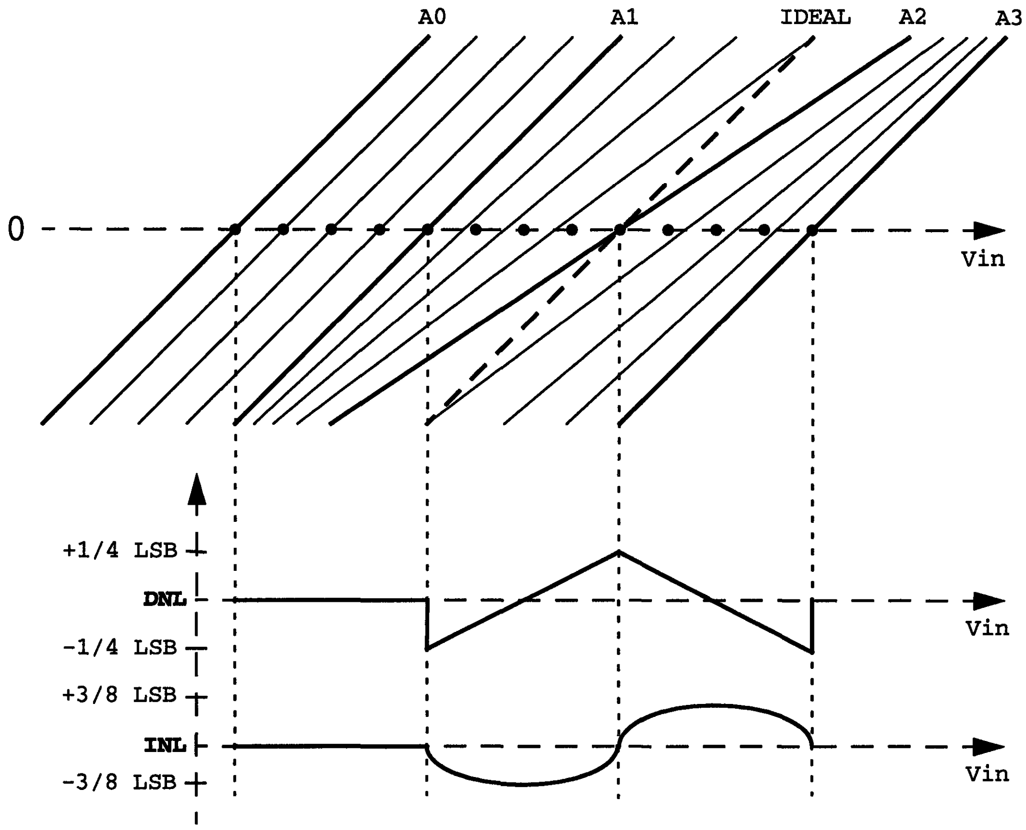

Figure 2.5. Linearity errors caused by pre-amplifier gain mismatch.

will not have sufficient overlap of their linear ranges to perform accurate interpolation. The ultimate benefit that can be obtained through interpolation is limited by the size of the amplifiers linear output range.

2.4.2 Gain Mismatch Sensitivity

Figure 2.5 illustrates the effect of a gain mismatch between adjacent amplifiers in an interpolating flash. This particular example illustrates a -30% gain error in amplifier A2. Even for this drastic mismatch, the worst-case INL and DNL performance is fairly reasonable. The INL sensitivity to gain mismatch increases linearly with number of quantization steps in-between the residues. The DNL performance is fairly independent of the number of quantization steps being interpolated.

The nonlinearity performance is worse than that in a standard flash converter, where gain mismatches do not affect the linearity of the converter. The interpolating architecture is more insensitive to amplifier offsets that the flash converter. Interpolation essentially trades off gain mismatch sensitivity for offset sensitivity.

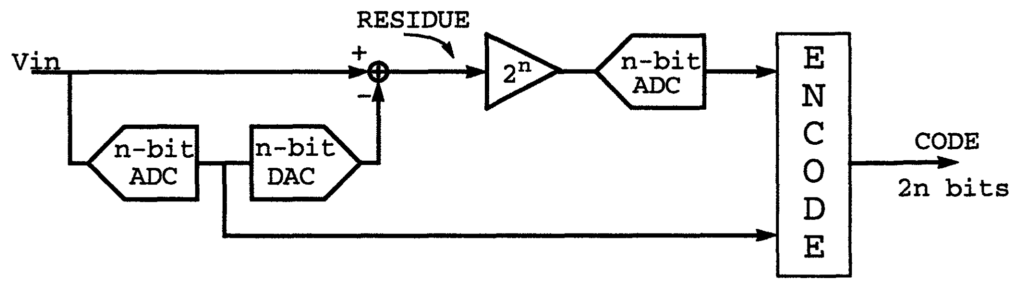

Figure 2.6. Two-step A/D architecture.

2.4.3 Summary

Interpolating flash converters present a number of advantages over a standard flash architecture. The sensitivity to preamplifier offset is greatly reduced. In CMOS technologies, this is particularly important. The inherent device matching in CMOS is inferior to bipolar technologies. Interpolating flash converters also significantly reduce the number of preamplifiers connected to the converter input; therefore, substantially reducing the input capacitance of the converter. There is also the additional benefit of reducing the area and possibly the power consumption of the converter. Even with these advantages, it is probably not feasible to extend this technique beyond 10 bits of resolution. At these resolutions, the required 2^n-1 comparators is still extremely cumbersome to implement. Interpolating techniques have been implemented successfully using bipolar technology to obtain 10-bit 300MSPS performance [8].

2.5 Multi-step Pipelined Architectures

The obvious drawback of fully parallel or flash converters is that the number of circuit components grows exponentially with converter resolution. Multi-step converters break the conversion into a number of lower resolution conversions. Figure 2.6 shows a two-step architecture. For a 2n-bit converter the number of comparators is reduced to approximately 2^n (as opposed to 2^{2n} for a full flash converter). A two-step architecture is a specific example of a general class of converters known as multi-step converters.

The two-step converter's operation is fairly straightforward. First a “coarse” conversion is made which best approximates the input signal. A D/A converter reproduces the analog signal corresponding to this conversion and subtracts it from the signal path. This “residue” is amplified to fill the full-scale range of the second or “fine” conversion.

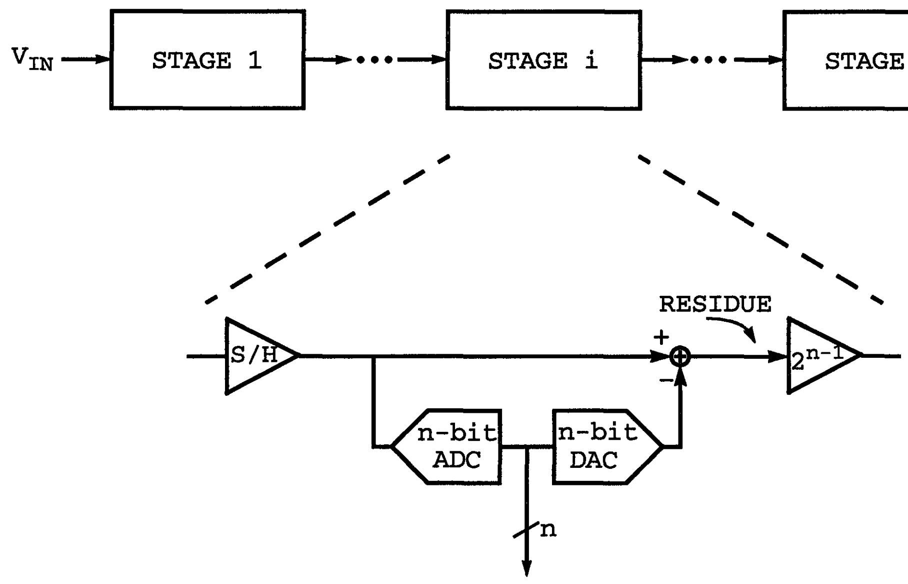

Figure 2.7. Multi-step pipelined architecture.

These two results are added digitally to get the output code. Often the coarse and fine conversions have a 1 bit of overlap to allow for errors in the coarse quantization (overlap range is discussed in more detail in section 2.5.2).

There are a number of important characteristics of multi-step converters. In general, the n-bit quantizers need only be accurate (linear) to n-bits. The D/A converters and subtraction circuitry must be accurate to the remaining accuracy of the converter. The interstage gain must have gain accuracy better than the remaining resolution of the converter. The constraints on the subtraction and interstage gain circuitry usually dictates that a high-gain amplifier be used in a feedback configuration. The presence of a closed-loop amplifier in the signal path typically limits the overall conversion rate.

2.5.1 Pipelining

Figure 2.7 shows a general multi-step architecture. This illustrates that the two-step architecture can be extended to an arbitrary number of stages. The limiting example is a 1-bit per stage architecture which requires only 1 comparator per bit of resolution desired (e.g. [10]).

The addition of a sample-and-hold amplifier allows the multi-step architecture to be pipelined to increase the throughput rate of the converter. The sample-and-hold must have linearity and offset performance that is commensurate with the accuracy of the remaining

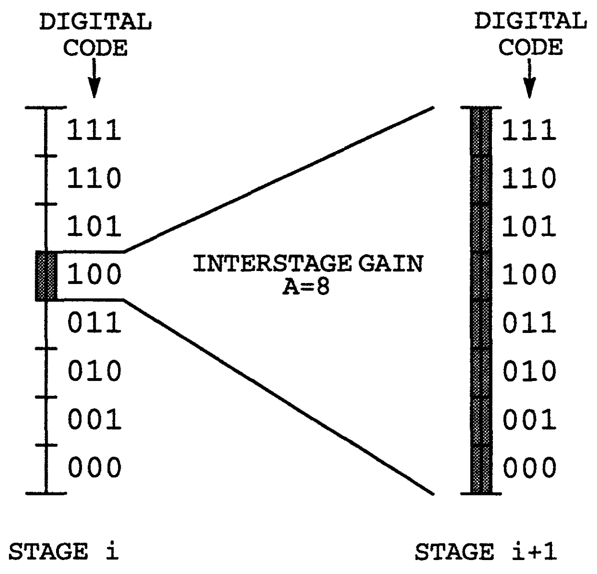

Figure 2.8. Interstage coding (no error correction).

stages. Again, the subtraction circuitry and interstage gain must also meet accuracy requirements equal to the remaining bits in the converter.

Multi-stage architectures have the added benefit of reducing the required accuracy in later stages of the converter. All errors are reduced by the preceding interstage gain when referred to input. By taking advantage of reduced accuracy requirements, power and area savings can be realized in the amplifiers in the later stages. This feature strongly influences the choice of how many bits to quantize in the first stage.

A number of converters have been realized utilizing a pipelined multi-step approach. Using a BiCMOS process, 10-bit 20MSPS performances has been achieved [11]. Similar performance has been realized in 0.9µm CMOS technology [12]. Lower power CMOS converters with similar performance have since been developed [13].

2.5.2 Overlap Range

Digital error correction is almost universally used in multi-step architectures. This technique reduces the accuracy requirement of each stage's quantizers to the number of bits being resolved in that stage. Digital error correction is accomplished by providing over- and under-range in the next stage [15]. This extra range is designed to be sufficient to correct errors made by the preceding quantizer.

Figure 2.8 illustrates interstage coding of a converter with no overlap range (i.e. no digital error correction). If the interstage A/D, D/A and subtracter are ideal, one

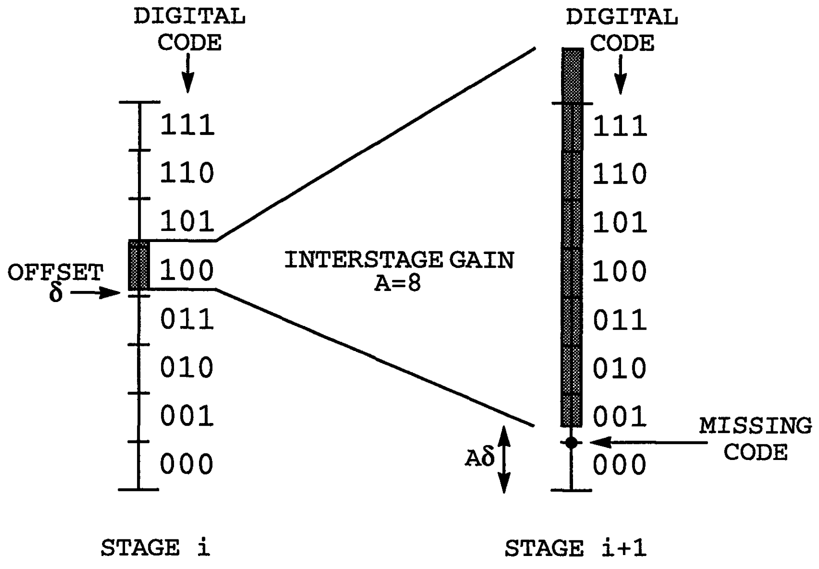

Figure 2.9. Interstage coding with quantizer offset error (no error correction).

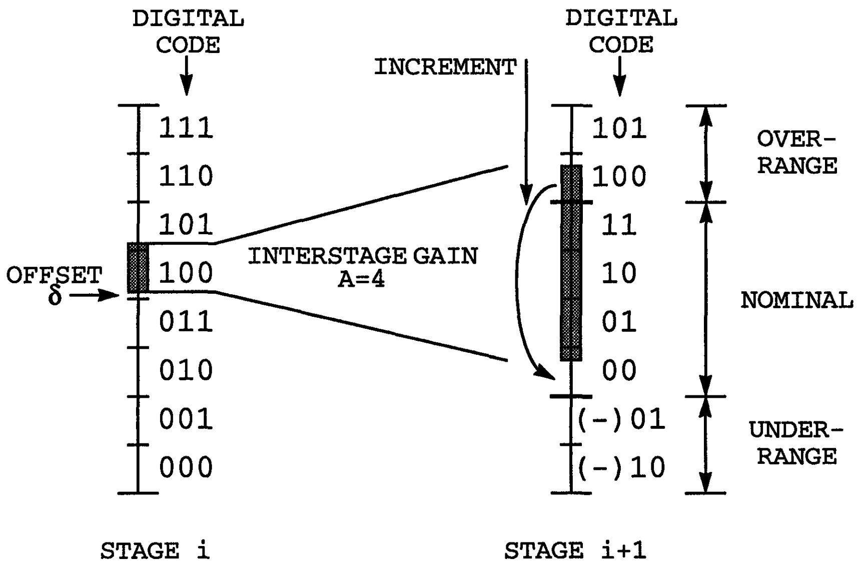

Figure 2.10. Interstage coding with quantizer offset (with error correction).

quantization step of the first stage is amplified to exactly fill the full-scale range of the next stage. Figure 2.9 illustrates the same interstage coding with offset added to 2 of the quantization levels. Because of errors in the first stage A/D converter, the signal range now falls outside of the next stage's full-scale input range. As illustrated, this results in a region of overflow and a region of missing codes. By adding over- and under-range, the converter will be desensitized to offset errors in the quantizers.

```
   100              100
+    100         -     01
--------         --------
   10100            01111

   (a)              (b)
```

Figure 2.11. Digital correction arithmetic for **(a)** over-range and **(b)** under-range.

Figure 2.10 shows one possible implementation of error-correction using over- and under-range. Because the interstage gain has been reduced by half, the converter can now tolerate ½ LSB errors in the quantizer. For signals that fall in the over-range, an overlapped add is performed to increment the previous stage's result by 1 LSB. Signals that fall in the under-range subtract 1 LSB from the previous stage. Figure 2.11 illustrates the digital arithmetic that must be done to perform the error correction. As mentioned above, this technique greatly eases the constraints imposed on the interstage quantizers.

2.5.3 Summary

Pipelined multi-step converters offer several advantages over standard flash converters. By breaking the conversion process into many steps, the area and power of the converter can be significantly reduced. The input capacitance of the converter is reduced as well. These factors make 10-bit and higher resolution converters feasible in CMOS. Also, by pipelining the converter, the throughput rate can be increased significantly.

Multi-step converters are not without their drawbacks. The requirement of having accurate gain and subtraction circuits makes design difficult. As mentioned above, these circuits are generally implemented using closed-loop techniques. These closed loop amplifiers impose a speed bottleneck on the conversion process. A multiplying digital-to-analog converter (MDAC) approach in CMOS has been used to combine the subtraction, gain and D/A functions [14]. The problem is reduced to one amplifier design per stage by using a MDAC-based architecture. For high-speed conversion, the performance demands on the amplifiers are extreme. They need extremely high gain for accurate MDAC operation, and excellent settling performance. For conversion speed, these amplifiers present the ultimate limit to performance.

2.6 Summary

This chapter has served as a brief overview of several popular A/D conversion techniques. It is clear from this examination that accurate subtraction and gain are the limiting factors in multi-step architectures. Flash-type architectures do not suffer from these design constraints; however, flash architectures are not efficient from a power or area standpoint. For applications that do not require ultra-high conversion rates, but do need moderate resolution (10b, <50MSPS), a flash architecture is probably not an appropriate choice.

The features of each of the above architectures which make them desirable are:

- No preamplifier gain-matching sensitivity in flash converters.
- Excellent conversion rate of full-parallel implementation.
- Reduced offset sensitivity of interpolating flash converters.
- No closed-loop amplifiers in interpolating architectures.
- Reduced power and area of multi-step approaches.
- Increased conversion rate gained through pipelining.

The approach of this research is to realize an architecture that takes advantage of the best features of each of these architectures.

Chapter 3

Multi-step Interpolation

3.1 Introduction

Pipelined multi-step architectures discussed in Chapter 2 have achieved excellent performance in CMOS [13]. These converters can be characterized as "single residue" architectures. Their accuracy is determined, in part, by how well the reference range of the each stage matches full-scale range of the single residue. The accuracy of the interstage gain in a "standard" pipelined multi-step converter determines how well these ranges will match. A small error in the interstage gain will cause the full-scale range of the residue to mismatch the full-scale range of the next stage. Because of required closed-loop gain accuracy, these architectures require extremely high-gain, wide-bandwidth amplifiers in order to obtain good conversion accuracy and speed.

The problem created by interstage gain error is illustrated in Figure 3.1 In this example, the interstage gain is too low. Because the gain is too low, the residue never reaches the full-scale range of the second stage. This example illustrates the first stage ADC transitioning before full-scale is reached in the second stage. Narrow or possibly missing codes are possible at the first stage ADC transitions.

3.2 Multi-residue Architectures

Interstage gain dependency would be alleviated if the reference for the next stage were passed through a unity gain amplifier with a percentage gain-error identical to the interstage gain error. The reference range of the stage would then match the full-scale range of the residue. Unfortunately, creating a matching gain error in a reference buffer amplifier is not a feasible approach in CMOS (this approach has been used at the 8b level in a bipolar technology [16]). An alternative, but equivalent, approach is to have the residue carry the reference information. This approach would force the reference and signal information to pass through the same interstage gain. Any gain error would affect both the reference and signal path identically. Multi-residue interpolating converters provide this advantage.

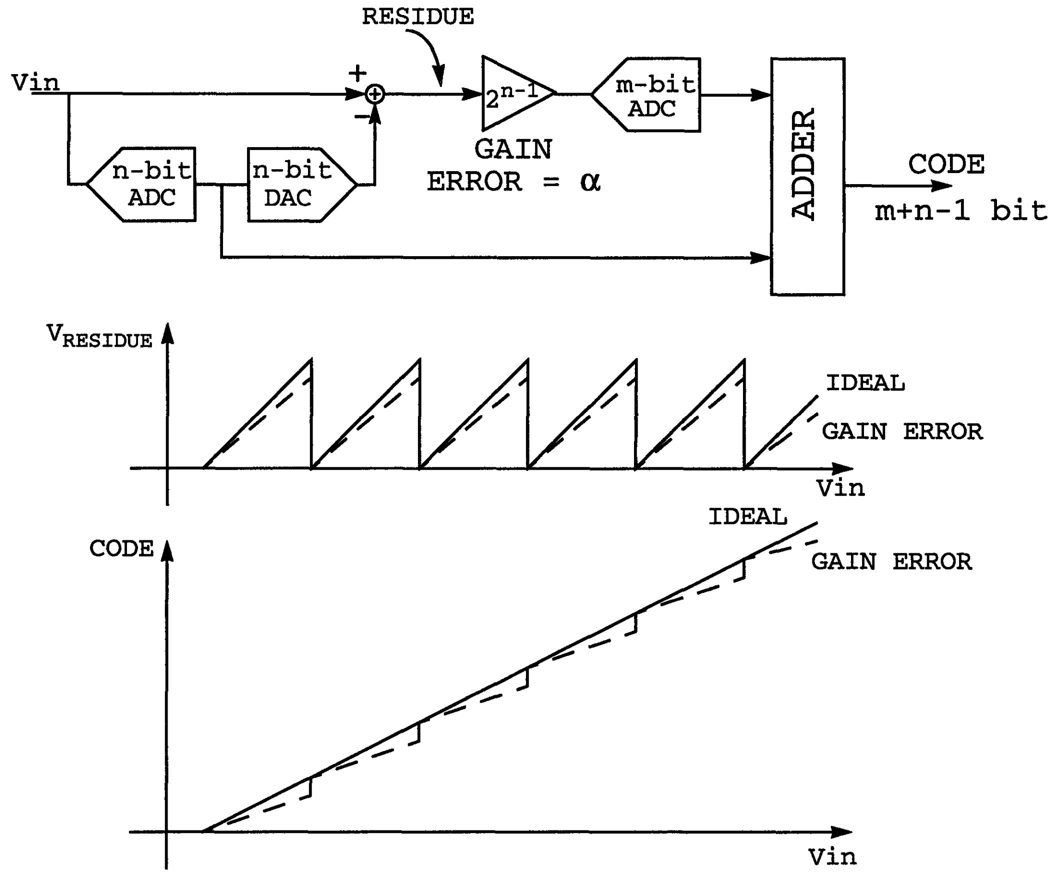

Figure 3.1. Multi-step converter with interstage gain error.

3.3 Two-residue Architecture

A two-residue architecture has been shown to drastically reduce the gain-bandwidth requirements of interstage amplifiers [17]. A conceptual diagram of this architecture is shown in Figure 3.2. A first residue is defined by the difference between the analog input and the closest quantization level. The second residue is the difference between the analog input and the second closest quantization level. The interstage amplifiers are switched so that input voltage occurs between their inverting inputs. This requirement guarantees that bottom amplifier's output (A1) will be positive and the top amplifier's output (A2) will be negative. By placing an interpolating ladder at the outputs of these two amplifiers and resolving where the "zero-crossing" of the interpolated residues occurs, the value of the input is quantized. As illustrated in Figure 3.2, this interpolation process can be continued with more interstage amplifiers and interpolating ladders to resolve the signal with increasing resolution. In each stage, the location of the zero-crossing is quantized. These

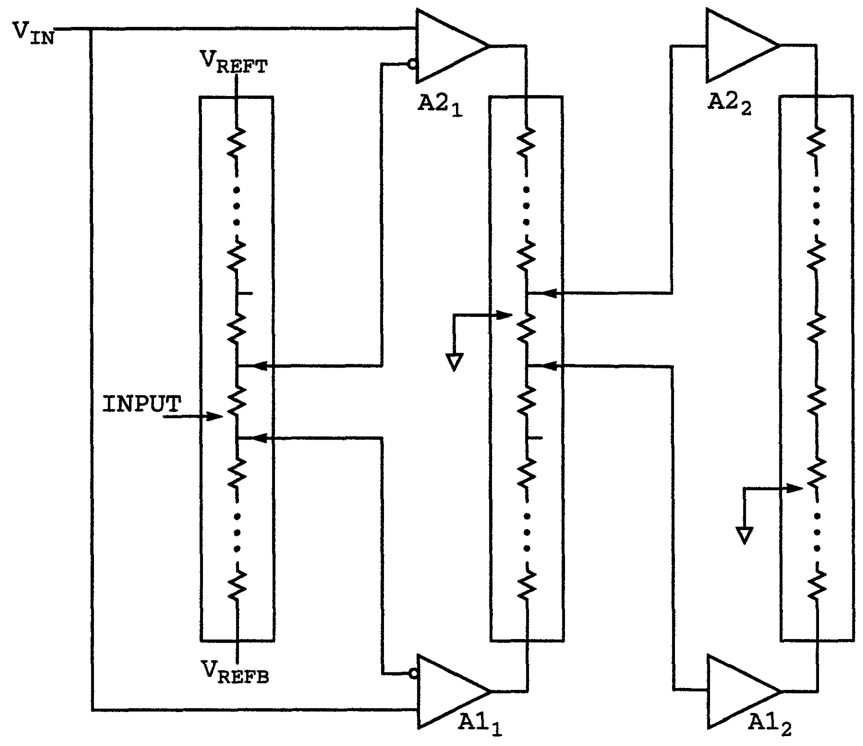

Figure 3.2. Multi-step two-residue architecture.

results are added digitally at the end of the conversion process to produce the output code. This digital addition is implemented in a similar fashion to a standard multi-step architecture described in Section 2.5. The results of the first stage are weighted to have the most significance, and each preceding stage is weighted with less significance.

3.3.1 Gain Matching Sensitivity

In this architecture, the difference between the two residues carries a scaled value of the quantization step, so conceptually the residues self-define their reference range. This desensitizes the architecture to variations in the residue amplifiers' gain. Figure 3.3 illustrates that the "zero-crossings" occur in identical locations regardless of the interstage gain. The position of the zero-crossings determines the location of the code transitions.

The residues are conceptually identical to those seen in the interpolating flash architecture Section 2.4). The difference arises from the fact that the residues are "folded" by the switching action along the taps of the interpolating ladders. This folding is what reduces the number of required comparators [18]. In essence, each comparator is used several times over the conversion range.

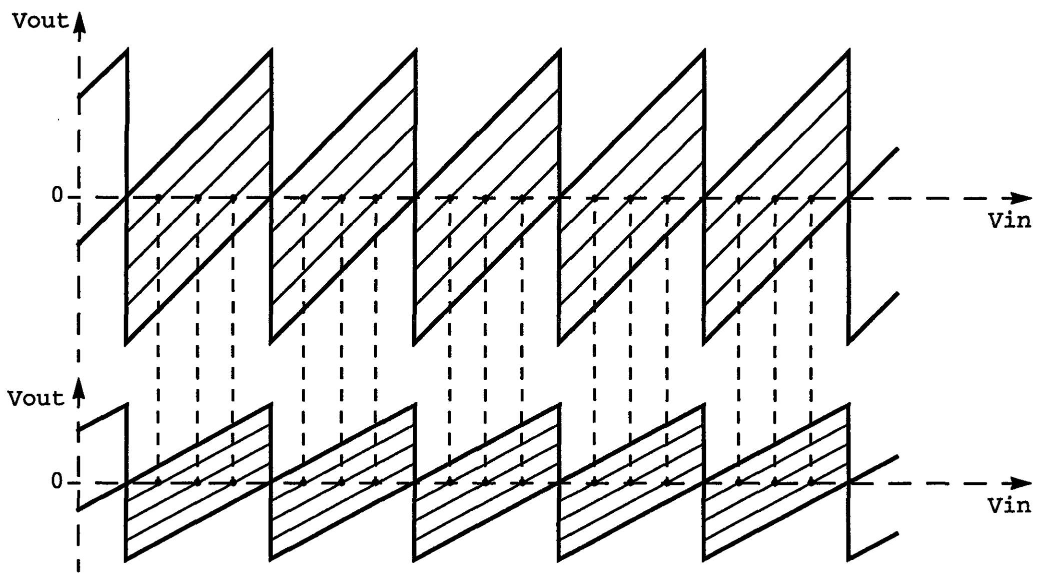

Figure 3.3. Two-residue architecture with varied interstage gain.

The gain matching between the two amplifiers is the important factor in determining the DNL performance of the converter. Interestingly, if both amplifiers have linear settling (i.e. no slewing), then complete settling isn't required in order to obtain converter linearity [17]. First order settling can be represented by (3.1).

$$V_o(t) = A \cdot V_i(1 - e^{-t/\tau}) \tag{3.1}$$

The exponential term is a constant factor if both amplifiers are sampled simultaneously and have identical time-constants. As mentioned above, the nominal value of the gain is not particularly important, just the gain matching. This concept holds for multiple order settling as well, as long as it is linear. This property is in stark contrast to standard multi-step architectures, where complete settling is paramount to the performance of the converter. This feature significantly eases constraints on the amplifier design (typically the most demanding specification of a multi-step architecture).

As illustrated in the discussion about the interpolating flash (Section 2.4), interpolating architectures are fairly insensitive to gain mismatches. Figure 2.5 illustrated a -30% gain mismatch that only resulted in $\frac{1}{4}$ LSB DNL errors. The maximum DNL error due to a gain mismatch is given by (3.2), where N is number of quantization levels between the residues and α is the gain error. The maximum INL due to a gain mismatch is given by (3.3).

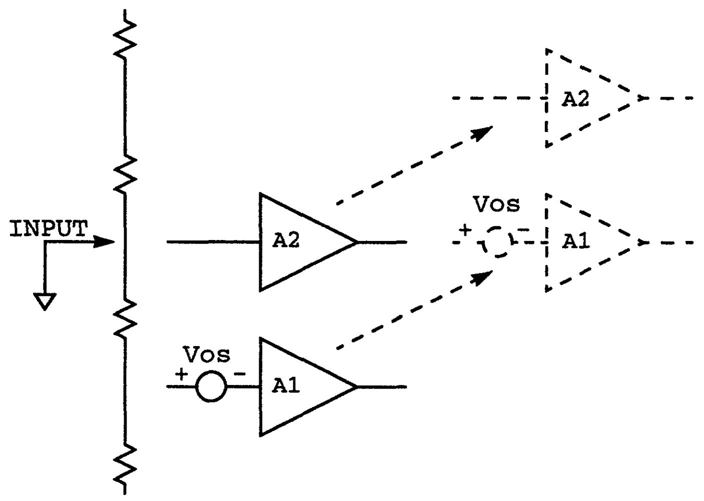

Figure 3.4. Two-residue architecture with sliding switching.

$$DNL \cong (N-1)\left(\frac{\alpha-1}{(\alpha-1)+N}\right) \tag{3.2}$$

$$INL \cong \frac{N}{2}\left(\frac{1}{0.5(\alpha-1)+1}-1\right) \tag{3.3}$$

3.3.2 Offset Sensitivity

Interpolating multi-step converters also have much less sensitivity to amplifier offset errors. Again, this is a general property of interpolating converters which was demonstrated for an interpolating flash in Section 2.4 (Figure 2.4). The maximum INL error is equal to the value of the offset; However, the worst case DNL is given by (3.4). Intuitively, (3.4) shows that offset errors are distributed evenly as DNL among each quantization level between the two residues.

$$DNL \cong \frac{V_{off(LSBs)}}{N} \tag{3.4}$$

3.3.3 DNL Degradation Due to Switching

The two-residue architecture illustrated in Figure 3.4 uses "sliding" switching to implement residue folding. In a sliding implementation, the residue amplifiers slide up/down one tap on the interpolation ladder when a comparator transition occurs. This approach severely degrades the DNL performance of the converter, and negates the offset insensitivity inherent in interpolating architectures. Also, since each amplifier must have a switch attached to each interpolation tap, a sliding architecture requires extensive wiring.

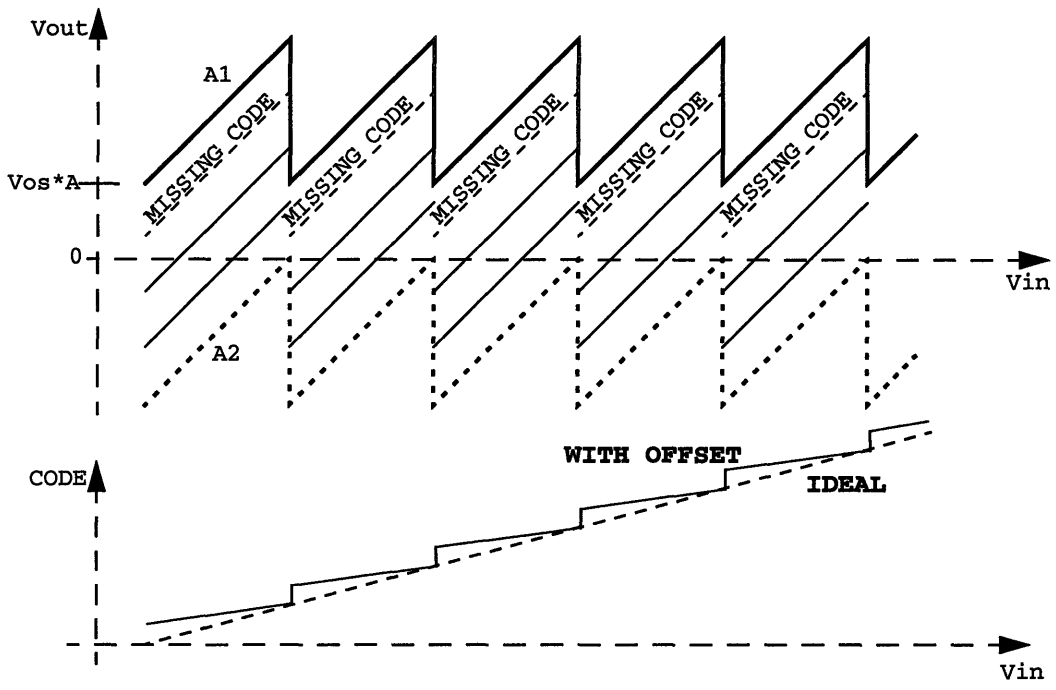

Figure 3.5. Amplifier residues for a sliding switching implementation.

Figure 3.4 illustrates the problem that occurs when a sliding switching implementation is used. Assume the input signal shown is centered on the threshold of amplifier A2. If the signal is incremented slightly, the associated comparator will trip and the amplifiers will slide up. Now the input is at an amplifier with an offset voltage. For an infinitesimal change in the input voltage, there is a step change in the output residue voltage (equal to the offset voltage multiplied by the amplifier gain). This corresponds to a DNL spike.

The residues seen at the outputs of amplifiers A1 and A2 for a ramp input are shown in Figure 3.5. The discontinuities are created by the folding action of the amplifiers being switched up and down along the interpolation ladder taps. This again illustrates the problem with a sliding implementation of switching. The offset in amplifier A1 shifts its transfer function up. The offset illustrated in Figure 3.5 is sufficient to pull one of the interpolated signals completely above the horizontal axis. This interpolated residue never crosses zero, so the corresponding code is missing altogether.

3.3.4 Leap Frog Switching

Figure 3.6 illustrates the “leap-frog” method of switching. The signal remains centered on the same amplifier threshold before and after the comparator transition. Figure 3.7 shows the amplifier residues for a leap-frog implementation. These residues are folded at

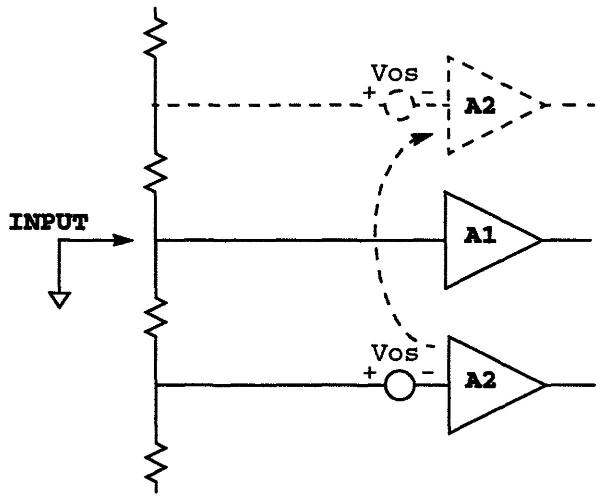

Figure 3.6. Two-residue interpolation with leap-from switching.

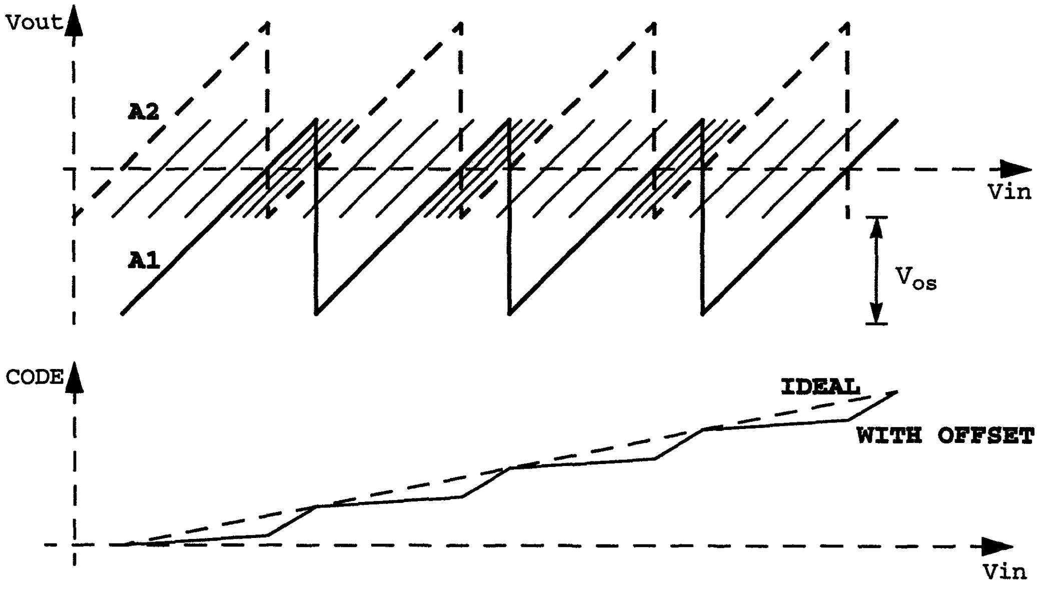

Figure 3.7. Amplifier residues for leap-frog switching implementation.

half the frequency as those in a sliding implementation. This characteristic makes sense because for a ramp input, each amplifier is moved half as often (as opposed to sliding both amplifiers up at each comparator transition). Interpolated signals are illustrated to show that no missing codes occur even for this drastic offset in amplifier A2. In fact, this architecture is guaranteed to have no missing codes as long as the amplifier offsets are smaller than the quantization steps on the interpolation ladder.

Some complications arise in the implementation of leap-frog switching in 2-residue architectures with comparator offsets. The presence of comparators with different offsets than the amplifiers creates the need for an overlap range to implement digital error

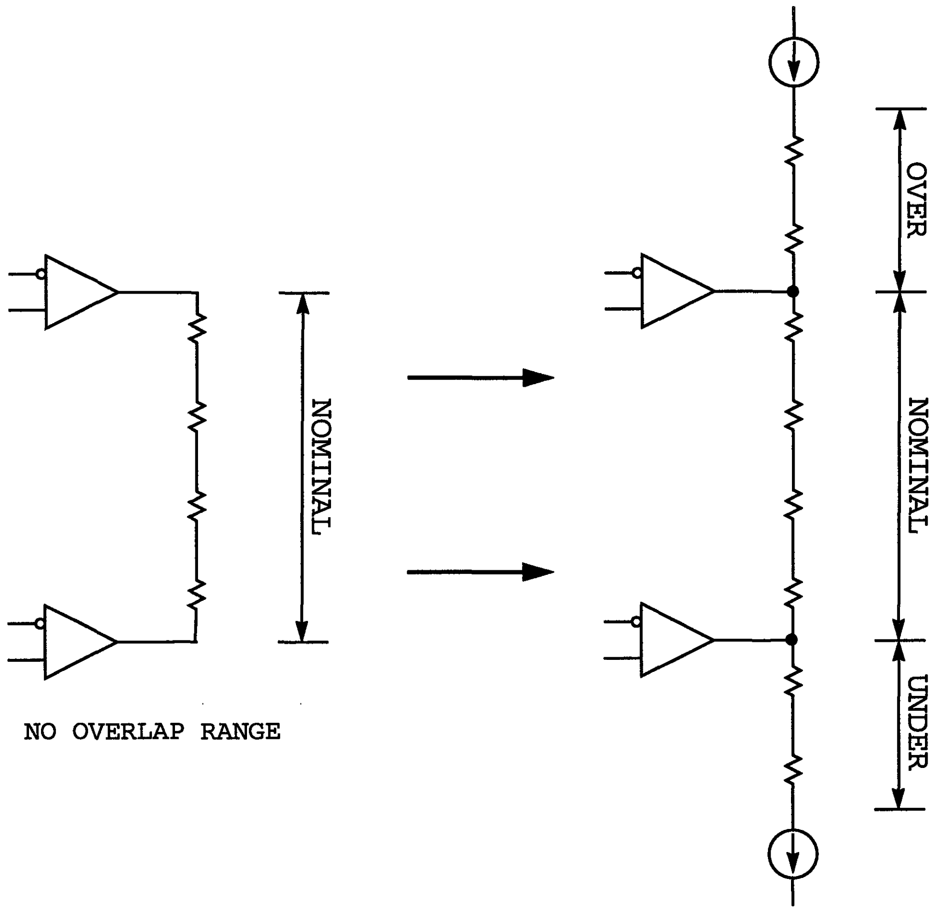

Figure 3.8. Artificially generated over- and under-range for 2-residue architecture.

correction. The presence of an overlap range complicates the above analysis, and ultimately encourages the design to 3- and 4-residue architectures.

3.3.5 Overlap Range

Over- and under-range must be created artificially in the 2-residue architecture [17]. Conceptually, over- and under-range is implemented by extending the interpolating ladders above and below the residue amplifiers and biasing these legs so that the quantization steps are appropriately sized. Figure 3.8 illustrates an artificially generated overlap range scheme using current sources to bias the over- and under-range correctly. It is unlikely that these artificially created over- and under-range quantization steps will match the nominal range steps exactly. When there is a comparator offset, the signal will occupy the over- or under-range. When a comparator transitions and brings the signal back into the nominal range, the mismatch between the replicated range and the nominal range will cause converter DNL. This problem can be remedied by using more than 2

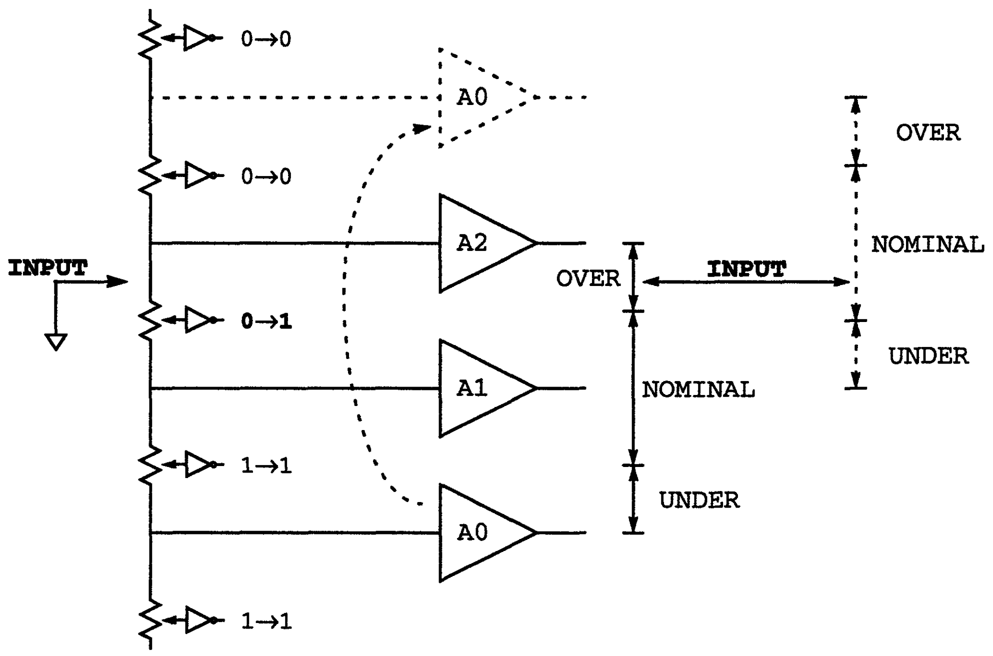

Figure 3.9. 3-residue interpolation with leap-frog switching.

residues. With 3 residues, the over- and under-range no longer need to be created artificially [19].

Figure 3.9 illustrates a 3-residue architecture with leap-frog switching. For this architecture, the comparator thresholds are set in-between the amplifier thresholds (as opposed to at the amplifier thresholds for a 2-residue architecture). An input which is centered on a comparator transition remains "between" the same two amplifiers and utilizes the identical interpolation network regardless of which direction the comparator flips. For an infinitesimal change in the input which flips a comparator, the signal path remains identical before and after the comparator transition. The combination of "leap-frog" switching and the presence of 3-residues makes this possible. Having 3 residues eliminates the need to create the overlap range artificially. As shown in figure 3.9, the over-range before a comparator transition is identical to the nominal range after the comparator transition. There is now no mismatch between the two ranges as was seen when they are artificially created.

Figure 3.10 illustrates the problem that occurs with a sliding approach to switching for a 3-residue architecture. A signal in the over-range before a comparator transition is mapped into the nominal range after a comparator transition. Because the over-range and

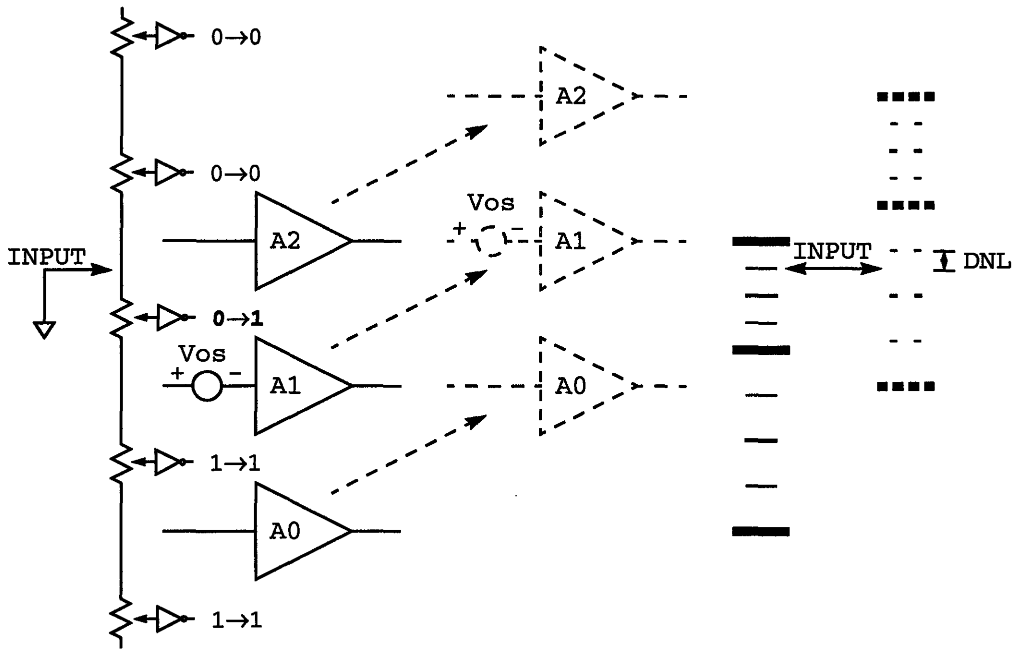

Figure 3.10. Three-residue architecture with sliding switching.

nominal range are created by different pairs of amplifiers (with different offsets) the two ranges do not match. A signal that has its interpolated zero-crossing at the 1/4 interpolation point before switching should have its zero-crossing at the same point after switching. In Figure 3.10, the zero-crossing points do not line up and a DNL error results. Leap-frogging ensures that zero-crossing occurs between the same pair of amplifiers before and after a comparator transitions. Note that Matsuzawa in [19] does not utilize leap-frog switching; Instead, a "sliding" implementation is utilized which is feasible because of the superior offset matching of bipolar transistors. Better DNL performance might have been achieved by utilizing a leap-frog switching scheme to implement the folding.

Error correction for this architecture can be accomplished in an identical manner as seen in pipelined multi-step architectures. The digital coding for the over- and under-range can be implemented as shown in Figure 2.10. Again, the overlap range eases the constraints on the comparators as mentioned in the error correction discussion in Chapter 2.

3.4 Four-residue Architecture

A four-residue architecture is a direct extension of a three-residue architecture. Figure

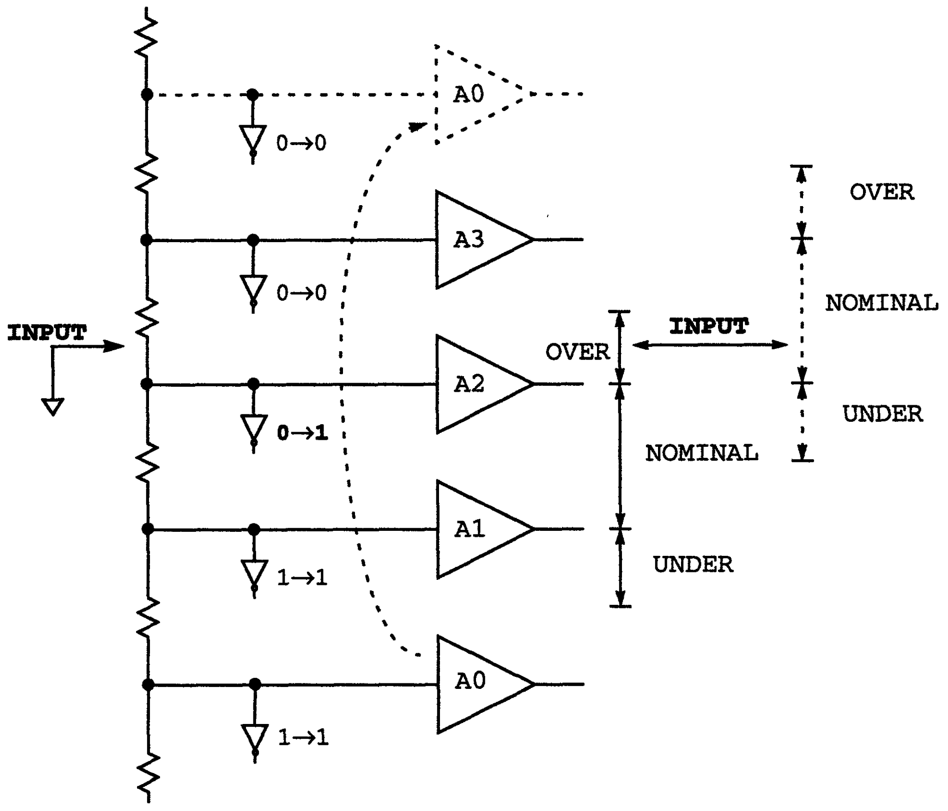

Figure 3.11. 4-residue interpolation with leap-frog switching.

3.11 illustrates a four-residue architecture with leap-frogged switching. The middle two amplifiers generate the nominal input range. The outer two amplifiers generate the under- and over-range. It is apparent from Figure 3.11 that some of the signal range is wasted, as compared to a 3-residue implementation. The wasted signal range above and below the over- and under-range amounts to ¼ of the total signal range. The wasted range indicates that one of the 4 amplifiers should be removed - which returns us to a 3-residue implementation. The reasons for using a 4-residue implementation are discussed in Section 3.4.2, and are based on implementation issues.

The benefit of leap-frog switching can be seen in Figure 3.11. Due to a comparator offset, the signal is occupying the over-range (between amplifiers A2 and A3). Assuming the signal is at the comparator transition, if it is incremented slightly then amplifier A0 will leap-frog above amplifier A3. The input signal remains between amplifiers A2 and A3 before and after the leap-frog switching occurs. Leap-frog switching provides the property where the signal path remains identical before and after the comparator

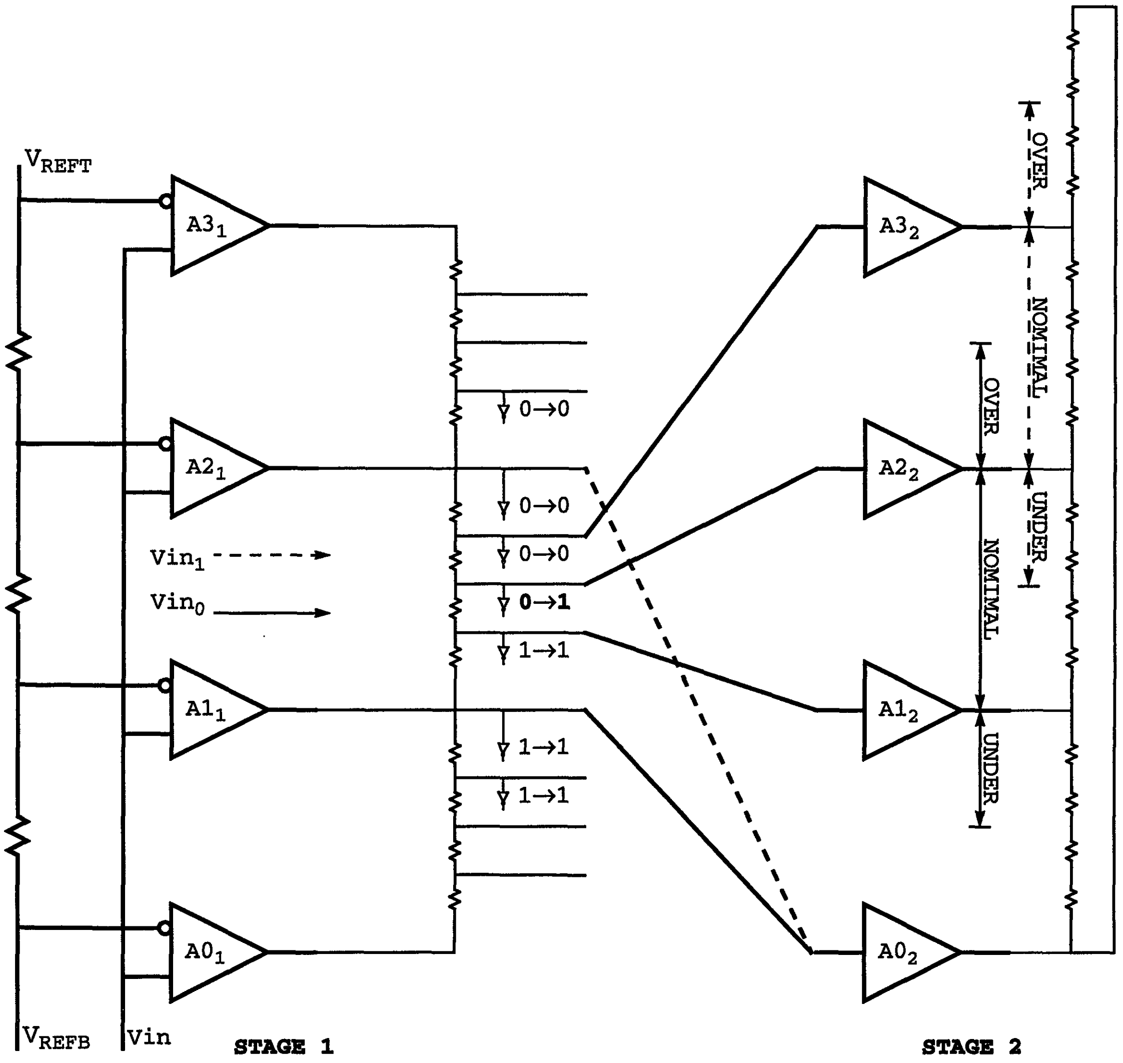

Figure 3.12. Four-residue architecture with leap-frog switching.

transition. As we have seen earlier, a sliding approach changes the amplifiers which "surround" the input signal (and also changes the interpolation ladder).

3.4.1 4-Residue Example

The first stage of a four-residue architecture is shown in Figure 3.12. Amplifiers $A0_1$-$A3_1$ create the first four residues by subtracting the input from four equally spaced quantization steps. These amplifiers are followed by a resistive interpolation ladder which is used to interpolate the location of the zero-crossing more finely. (Again, the location of the residues' interpolated zero-crossing indicates the value of the input.) Comparators are placed at the taps of the interpolation ladder. The comparators produce the familiar

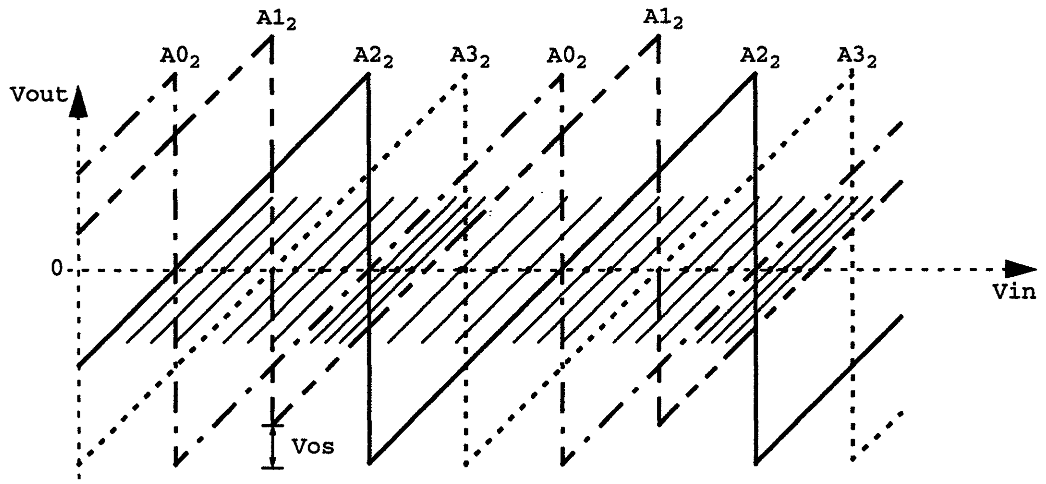

Figure 3.13. Transfer functions of second stage amplifiers in 4-residue architecture.

"thermometer" code which indicates the location of the zero crossing by a $1 \rightarrow 0$ transition.

Based on the comparators' thermometer code, the next stage is switched to the interpolation taps so the zero crossing occurs in the next stage's nominal range. The next stage amplifies the four residues and performs interpolation to resolve the value of the input signal more finely. In Figure 3.12, the nominal range occurs between $A1_2$-$A2_2$ for Vin_0. For Vin_1, the nominal range now occurs between amplifiers $A2_2$-$A3_2$. With a leap-frog architecture, the nominal range is not constrained to occur between any two particular amplifiers. It is for this reason that the interpolation ladder must be circular in the second stage. This is in contrast to a sliding approach to switching, where the nominal range always occurs between the same two amplifiers.

As Figure 3.12 shows, leap-frog switching is used to implement signal folding. Figure 3.13 illustrates the transfer functions seen at the outputs of amplifiers $A0_2$-$A3_2$. Because of the leap-frog switching implementation, these signals are folded at ¼ of the frequency than would be seen in a sliding implementation of switching. The folding frequency can understood by realizing that for a ramp input, each amplifier is only leap-frogged upwards one out of every four comparator transitions.

Figure 3.13 also illustrates the effect of a positive offset in amplifier $A1_2$. Amplifier $A1_2$'s transfer function is shifted upwards by an amount equal to its offset. By inspecting the interpolated crossing, it is can be seen that the leap-frog switching does not produce

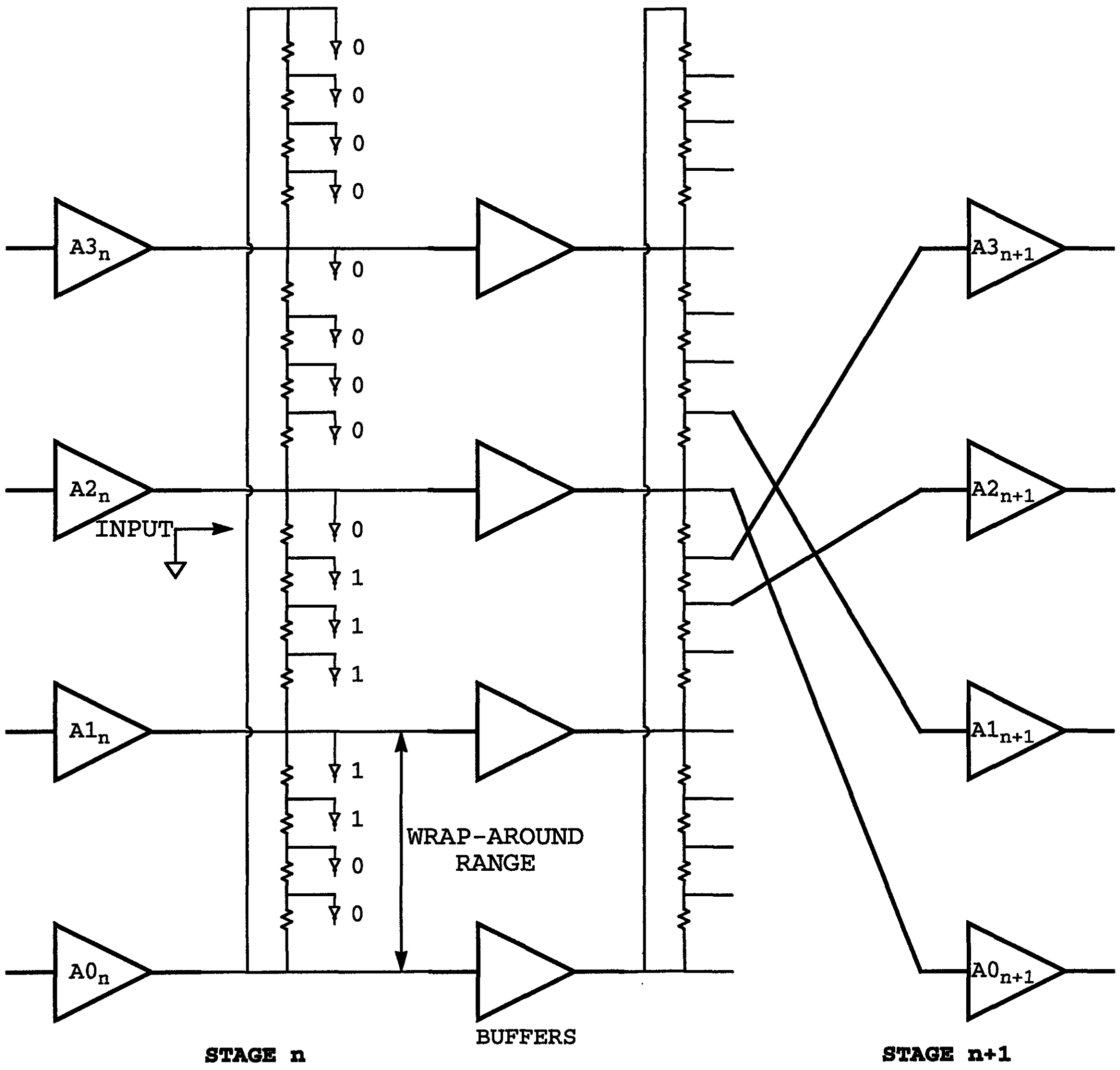

Figure 3.14. 3-bit interpolation stage with leap-frog switching.

DNL spikes. The interpolated signals shown have the same properties as those seen in the interpolating flash (Section 2.4). An offset error creates banks of uniformly wide/narrow codes. Offsets do not create DNL "spikes" at the switching transitions of the first stage flash. A sliding approach to switching would have introduced such spikes.

Figure 3.14 shows a more general implementation of a four residue stage. This architecture is closest to the actual implementation that will be presented in Chapter 6. Each stage has two banks of 4 amplifiers. Each bank of amplifiers drives its own interpolation ladder. The first amplifier bank drives the interpolation ladder for the quantizer. The second bank of amplifiers drives the interpolation ladder for the next stage.

The primary purpose of the second bank of amplifiers is to provide a pipeline delay. The delay is used so that the next stage can switch into place after the quantization takes place. By having two separate interpolation ladders, the number of required amplifiers per stage is reduced. If only one ladder were used, the second amplifier bank would require 16 buffer amplifiers to buffer the interpolation taps. Also, since the actual implementation uses a capacitive ladder, the quantizer and the amplifiers can not share the same ladder.

As mentioned earlier, all of the interpolation ladders must be circular due to the leap-frogged switching. The nominal range is not constrained to occur between any two specific amplifiers. Because of uncertainty about the nominal range's location, the interpolated zero-crossing can occur at any point along the circular ladder. This architecture requires 16 comparators per 3-bit stage instead of the usual 7 comparators. Note also, the comparator decisions in Figure 3.14 indicate two zero-crossings. One of these crossings corresponds to the zero-crossing we are interested in, the other is due fact that the interpolation ladder must wrap around to the "bottom" amplifier. The location of the "wrap-around" moves as the amplifiers are leap-frogged. The zero-crossing we are interested in is indicated by the $1 \rightarrow 0$ transition as we move upwards along the ladder.

3.4.2 Leap-frog Switching in 4-residue 3-bit Stage

Notice that with the leap-frog implementation in Figure 3.12, each amplifier is always connected to the same interpolation ratio. As drawn in Figure 3.12, amplifier $A0_2$ is always connected to the "no interpolation" point, amplifier $A1_2$ is always connected to the $^3/_4$ interpolation point, etc. This property is unique to architectures that have the same number of residue amplifiers as the number of interpolation points. This property only holds when the architecture uses leap-frog switching. This feature of a 4-residue 3-bit stage is useful particularly from an implementation standpoint.

Instead of attaching the output buffers permanently to the interpolation network (as shown in Figure 3.14), the input amplifiers of the next stage each have their own interpolation network. Essentially, each input amplifier has its own interpolation network, which remains unchanged. In capacitive switched-capacitor interpolation, this greatly simplifies circuit design.

3.5 Summary

Pipelined multi-step interpolation has the benefits of standard pipelined multi-step architectures. The number of required comparators are reduced drastically. The accuracy requirements of the comparators can be reduced through the implementation of an analog overlap region. By pipelining the architecture, the throughput can be increased dramatically.

By using a multi-residue interpolating architecture, several additional benefits are gained. Most importantly, the gain requirements of the amplifiers are reduced significantly over the requirements of amplifiers in single-residue architecture. As seen in this chapter, the interpolating architecture is not very sensitive to inter-stage gain errors. Also, offsets in this architecture do not create significant DNL errors when leap-frogged switching is used. These characteristics ease the constraints on the amplifier design.

The drawback of these architectures is that many more amplifiers are required. In a standard MDAC multi-step converter, only 1 MDAC is required per stage. For a pipelined multi-step architecture with 4-residues, 8 amplifiers are required per stage. The accuracy requirements of these amplifiers is greatly reduced, but this is still a large number.

From a preliminary investigation, it appears that these architectures have several potential benefits. The most important of which is the ability to achieve excellent converter DNL performance with lower gain-bandwidth amplifiers.

Chapter 4

Switched Capacitor Interpolation

4.1 Introduction

Up to this point, interpolation has been demonstrated exclusively using resistors. Although extremely simple to implement, a resistive ladder is not always the best option. The current supplied to interpolation ladder from the amplifier creates an error in the amplifier's output (due to the amplifiers' finite output impedance). In some implementations, this amplifier error actually helps matters by partially correcting amplifier offsets [9]. In a multi-step converter with a circular resistive ladder (as demonstrated in Section 3.4.1), these errors are not desirable. As the amplifiers leap-frog, the amount of ladder current each amplifier is supplying varies. Assuming the amplifiers have a finite output impedance, the leap-frogged switching will create a signal dependant error-voltage. A solution is to design the amplifiers to have an extremely low output impedance. This approach is a costly solution, especially in CMOS due to the poor transconductance of the devices. Switched-capacitor circuits alleviate this concern in CMOS circuits by taking advantage of the infinite DC impedance of capacitors.

The pipelined multi-step interpolating converter needs several switched capacitor functions. First, the architecture requires a cell that implements interpolation. Also, each stage in the interpolating architecture has a nominal gain of 4 (in a 3 bit per stage architecture). Because each stage is pipelined, it is necessary to break each stage into two gain stages. From a gain-bandwidth perspective it is desirable to break the stage into two gain of 2 stages. However, from a circuit standpoint it turns out that it is desirable to break the architecture into a gain of 4 stage followed by a gain of 1 stage. The gain of 4 stage will perform interpolation in addition to amplification. The gain of 1 stage only serves as a unity-gain pipeline delay.

The multi-step interpolating converter also requires an interpolating quantizer. Typically, a resistive ladder is used in a multi-step architecture. As mentioned above, we would like to avoid having a finite DC impedance attached to the outputs of the residue amplifiers. In this chapter, a capacitive alternative to a resistive ladder is discussed.

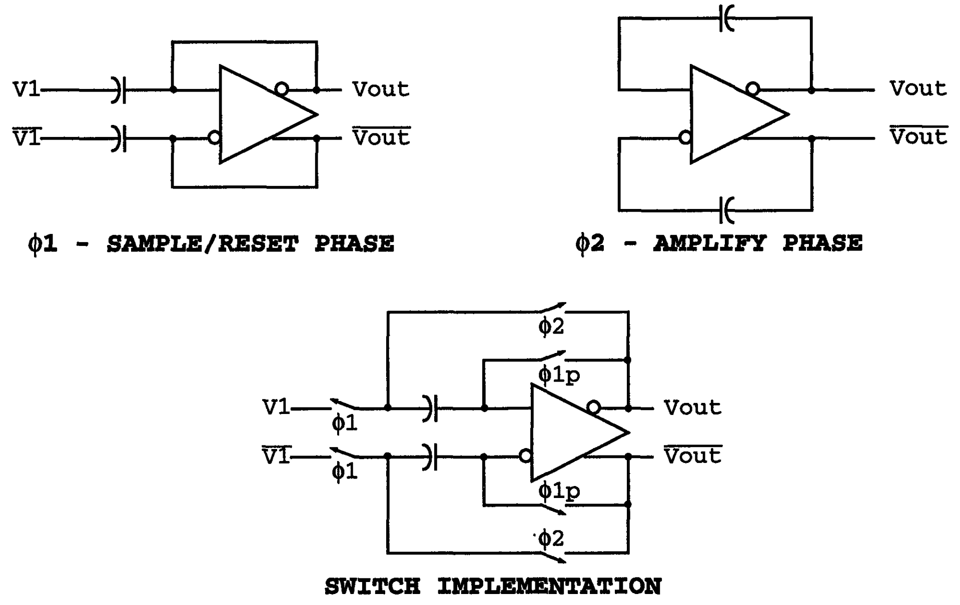

Figure 4.1. Unity-gain switched capacitor amplifier.

It is worth mentioning at this point that the switched-capacitor circuits discussed in this chapter are all fully differential. Until now, all of the converter examples have been demonstrated using a single-ended signal path. The single-ended implementation is much easier to present and retains many of the important features of the differential implementation. The actual converter implementation discussed in Chapter 6 has a fully differential signal path.

4.2 Switched Capacitor Unity-Gain Cell

The operation of a unity-gain switched capacitor cell is a natural place to begin the explanation of the switched capacitor circuits. Figure 4.1 shows a unity-gain switched capacitor buffer. This circuit is clocked using a standard 2-phase non-overlapped clocking scheme.

On $\phi 1$, the amplifier resets and samples the input signal. In addition to acquiring the input signal on the bottom plates (the curved plates in the schematic) of the capacitors, the amplifier also auto-zeros itself by storing the amplifier offset on the top plates (the straight plates in the schematic) of the sampling capacitors. The offset is stored on the top plates by placing the amplifier in a unity-gain feedback configuration. In unity-gain feedback, the amplifier balances its input pair, which causes the amplifier offset to appear at the inputs (assuming infinite amplifier gain). The input is sampled on a pre-edge of $\phi 1$ by opening the unity-gain connection, this captures a charge on the capacitors equal to the input minus the amplifier offset voltage. Because of the almost infinite input impedance seen at the input to a CMOS amplifier, the charge on the sampling capacitors does not decay significantly.

The input signal appears at the output during the second clock phase by flipping the two unit capacitors around the amplifier in a feedback configuration. The voltage across the capacitors remains identical, because the stored charge is the same. As a result, the output voltage during $\phi 2$ is equal to the input voltage at $\phi 1$. This 2-phase operation makes this a viable unity-gain buffer to implement an analog pipeline delay.

4.2.1 Finite Amplifier Gain Effects - Offset Cancellation

In reality, the CMOS operational amplifier has finite gain, and this influences the performance of the unity-gain buffer illustrated above. Because of finite amplifier gain, when the amplifier is connected in unity-gain during reset mode, the exact value of the offset does not appear at the inputs. The voltage that actually appears at the input to the amplifier is given by (4.1). The exact value of the amplifier offset is not stored on the top plates of the sampling capacitors. Because of this, the amplifier offset is not nulled exactly. The actual input-referred offset is given by (4.2).

$$V_{input} = -\left(\frac{A}{1+A}\right)V_{os} \tag{4.1}$$

$$V_{offset} = -\frac{V_{os}}{1+A} \tag{4.2}$$

4.2.2 Finite Amplifier Gain Effects - Gain Error

Ideally, the gain of this switched-capacitor amplifier configuration should be 1, but due to the finite amplifier gain it is actually slightly less. For a given differential output, there must be a slight voltage difference at the input to the amplifier. This difference is equal to the output voltage divided by the open-loop amplifier gain. The amplifier outputs must provide this extra voltage difference at the inputs and this degrades the closed-loop gain of this configuration slightly. The actual gain of the amplifier is easily derived from Black's feedback expression and is given by (4.3).

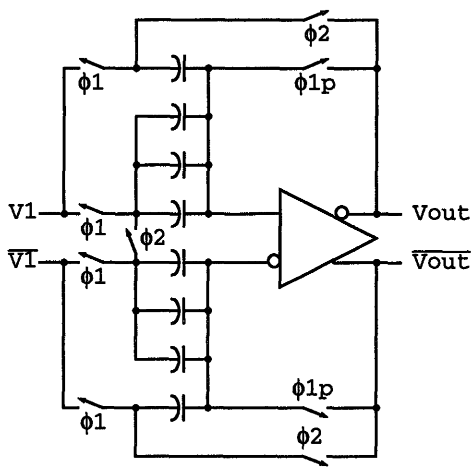

Figure 4.2. Gain of 4 switched-capacitor amplifier.

$$\frac{V_{out}}{V_{in}} = \frac{A}{1+A} \tag{4.3}$$

4.2.3 Finite Amplifier Gain Effects - Parasitic Capacitance

The presence of parasitic capacitance at the input to the amplifier degrades the gain accuracy even further. As before, a non-zero output voltage implies that the input terminals must have a voltage across them. A parasitic capacitance at the amplifier input must be charged to this voltage. This parasitic charge is supplied by the sampling capacitors and acts to decrease the voltage stored on the sampling capacitors. Therefore, the parasitic capacitance reduces the closed-loop gain. Expression (4.4) gives the gain expression with the addition of a parasitic capacitance (this reduces to (4.3) by setting C_p to zero).

$$\frac{V_{out}}{V_{in}} = \frac{A}{(1+A) + \frac{C_p}{C}} \tag{4.4}$$

4.3 Switched Capacitor Gain of 4

Figure 4.2 illustrates the switched capacitor implementation of a gain of 4. All of the capacitors are of equal unit size. The operation of this circuit is similar to the unity-gain amplifier illustrated above. On the first clock phase the input is sampled on each of the 4 pairs of sampling capacitors. During this phase, the amplifier is configured in unity-gain in order to null the offset. As before, the input voltage minus the amplifier offset is sampled on all of the sampling capacitors by opening the unity-gain feedback switch (on $\phi 1p$). The charge on the sampling capacitors is now trapped by the large input impedance of the amplifier inputs. During the second clock phase, one pair of the unit capacitors is "flipped" around the amplifier in feedback. The bottom plates of the other 3 pairs of capacitors are shorted differentially (a differential ground). By differentially shorting these capacitors, their differential charge corresponding to the input voltage is forced onto the feedback capacitors. The charge on the feedback capacitors is now equal to 4 times the charge that was initially stored on each sampling capacitor. The output during the second clock phase must be 4 times the input voltage.

4.3.1 Finite Amplifier Gain Effects

The gain of 4 switched capacitor amplifier is subject to the same finite amplifier gain effects that created limitations in the unity-gain amplifier. Again, the due to the finite amplifier gain, the exact value of the offset is not stored on the top plates of the sampling capacitors. The referred to input offset expression for the switched capacitor gain of four circuit is given by (4.5). As before, the finite amplifier gain also limits the overall gain accuracy of the amplifier. Expression (4.6) gives the actual gain realized using a finite gain CMOS operational amplifier. Finally, the presence of parasitic capacitance at the input of the operational amplifier also degrades the gain performance. Equation (4.7) gives the degradation in gain performance (again, this reduces to (4.6) by setting C_p to zero).

$$V_{offset} = -\frac{V_{os}}{1+A} \tag{4.5}$$

$$\frac{V_{out}}{V_{in}} = \frac{4}{1+\frac{4}{A}} \tag{4.6}$$

$$\frac{V_{out}}{V_{in}} = \frac{4A}{(4+A)+\frac{C_p}{C}} \tag{4.7}$$

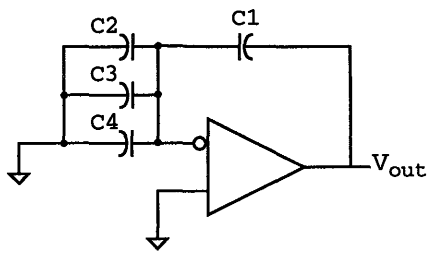

Figure 4.3. Single-ended switched capacitor amplifier in amplify phase.

4.3.2 Capacitor Matching

The unity-gain buffer was insensitive to the capacitor matching of the 2 sampling capacitors. The gain of 4 architecture is sensitive to the matching between the capacitor elements. It is simpler to analyze the effects of capacitor mismatch using a single-ended architecture. Figure 4.3 shows a single-ended switched capacitor gain of 4 amplifier during amplify phase. With an ideal operational amplifier, the gain of this structure is given by (4.8). A mismatch in any of the capacitors creates a gain error. From (4.8) it is obvious that this structure is most sensitive to capacitor C1 (the feedback capacitor). For a capacitor mismatch in C1 of ΔC the change in gain is given by (4.9).

$$\frac{V_{out}}{V_{in}} = \frac{C1 + C2 + C3 + C4}{C1} \tag{4.8}$$

$$\frac{\Delta G}{G} \cong -\frac{3}{4}\frac{\Delta C}{C} \tag{4.9}$$

4.4 Switched Capacitor Interpolation

Switched capacitor interpolation can be implemented by making simple modifications to the gain of 4 circuit shown in Figure 4.2. Circuits that interpolate $\frac{1}{4}$, $\frac{1}{2}$ and $\frac{3}{4}$ points are shown in Figure 4.4. These circuits are essentially identical to the switched capacitor gain of 4 circuit discussed earlier. To realize interpolation, some fraction of the input capacitors are charged to a different input voltage. As expected, these circuits also realize a gain of 4 in addition to performing interpolation.

The $\frac{1}{2}$ point interpolation is accomplished by sampling V1 on 2 pairs of the sampling capacitors and sampling V2 on the other two pairs of input capacitors. The total charge on the capacitors is given by expression (4.10). As before, the charge from the other 3 pairs of

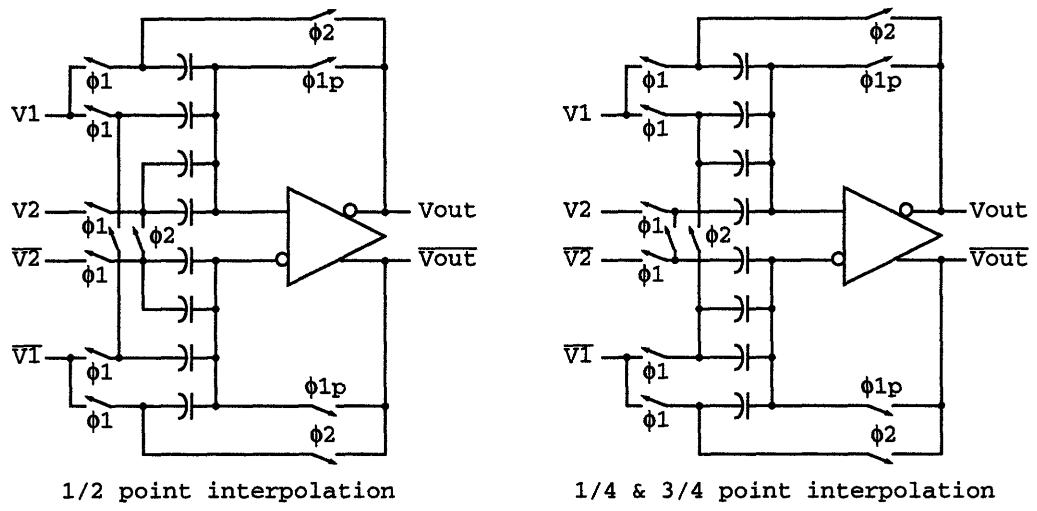

Figure 4.4. Switched capacitor interpolation ($\frac{1}{4}$, $\frac{1}{2}$ and $\frac{3}{4}$).

sampling capacitors is transferred to the feedback capacitors during the amplify phase. Therefore, the charge on the feedback capacitor corresponds to the voltage given by (4.11). This configuration implements the desired function of interpolating the $\frac{1}{2}$ point of the two input signals and amplifying the result by 4.

$$Q_{total} = C(2 \cdot V1 + 2 \cdot V2) \tag{4.10}$$

$$V_{out} = 2 \cdot V1 + 2 \cdot V2 \tag{4.11}$$

The $\frac{1}{4}$ and $\frac{3}{4}$ point interpolation is performed using a similar method to the one used to accomplish $\frac{1}{2}$ point interpolation. Note that the only difference between $\frac{1}{4}$ and $\frac{3}{4}$ point interpolation is the order which the inputs are applied to the circuit. This simplification is allowed because the $\frac{1}{4}$ and $\frac{3}{4}$ points are symmetric with respect to the $\frac{1}{2}$ point. For $\frac{1}{4}$ point interpolation, V1 is sampled on 1 pair of the input capacitors and V2 is sampled on the other 3 pairs of sampling capacitors. As before, all of the charge is transferred to the feedback capacitor during the amplify phase of operation. Depending on the order V1 and V2 are connected to the input to the circuit, the function in equation (4.12) or (4.13) will be performed.

$$V_{out} = 1 \cdot V1 + 3 \cdot V2 \tag{4.12}$$

$$V_{out} = 3 \cdot V1 + 1 \cdot V2 \tag{4.13}$$

The sources of error in the interpolating gain of 4 circuit are identical to those seen in the switched capacitor gain of four circuit. The offset is given by (4.5), the gain error due to finite operational amplifier gain is given by equation (4.6) and the gain error due to the presence of parasitic input capacitance is given by (4.7).

4.4.1 Capacitor Matching

The capacitor matching ultimately determines the accuracy of the interpolation. By analyzing the circuit from a single-ended perspective, the interpolation functions are given by equations (4.14) and (4.15), where C1-C4 are the four sampling capacitors. An error of ΔC in any of these capacitors gives a interpolation error which is approximately $^{\Delta C}/_{C}$. The interpolation accuracy is not dependant on the open-loop gain of the operational amplifier. The finite open-loop gain of the amplifier only contributes to an error in the gain of the interpolation cell, not an error in the interpolation.

$$point_{(1/4)} = \frac{C1}{C1 + C2 + C3 + C4} \tag{4.14}$$

$$point_{(1/2)} = \frac{C1 + C2}{C1 + C2 + C3 + C4} \tag{4.15}$$

4.5 Switched Capacitor Quantization

As mentioned in the introduction, it is also important to have a method of performing interpolation for the input to a quantizer. Again, capacitive interpolation is necessary because a resistive interpolation ladder loads the residue amplifiers in a multi-step interpolation architecture. The technique for performing switched capacitor interpolation for a comparator is extremely similar to the techniques used to perform interpolated amplification.

Figure 4.5 illustrates an "interpolating comparator" which performs $\frac{1}{4}$ point interpolation. On the first clock phase (shown in figure), the inputs are sampled on the input capacitors. The sampling is performed by grounding the top plate of all of the capacitors and applying the input signal to the bottom plates of the capacitors. In this example, V1 is sampled on 3 unit capacitors and V2 is sampled on 1 unit capacitor. On the

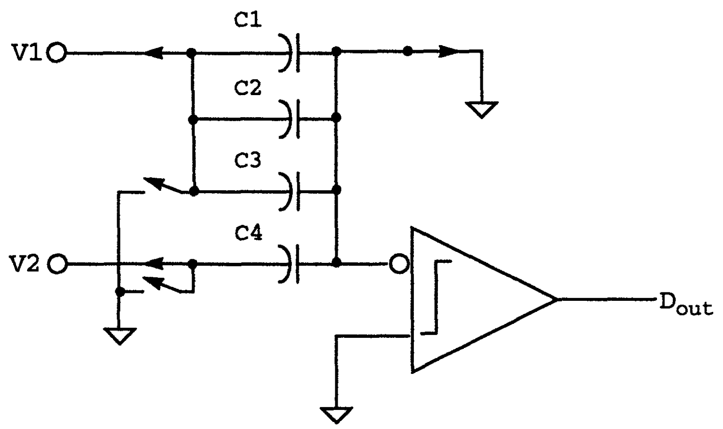

Figure 4.5. Interpolating comparator at ¼ point.

second clock phase, the ground connection is removed from the top plates on a pre-edge. Next, the bottom plates of all of the capacitors are grounded. All of the capacitor charge is forced to redistribute evenly among the four capacitors. The voltage at all of the top plates after this redistribution is given by (4.16). It is clear that this arrangement provides the desired interpolation function.

$$V_{top} = -\frac{V1\,(C1 + C2 + C3) + V2\,(C4)}{C1 + C2 + C3 + C4} = -\frac{3 \cdot V1 + 1 \cdot V2}{4} \qquad (4.16)$$

The ½ point can be generated by using 2 unit sampling capacitors. In this case, V1 would be sampled on one of the unit capacitors and V2 on the other. When the bottom plates are shorted, the charge is forced to redistribute. The voltage at the top plates is given by (4.17).

$$V_{top} = -\frac{V1\,(C1) + V2\,(C2)}{C1 + C2} = -\frac{V1 + V2}{2} \qquad (4.17)$$

The interpolation circuit was shown using a single-ended implementation for simplicity. The differential implementation is shown in figure 4.6. Instead of grounding the bottom-plates, they are differentially shorted (a differential ground). The top plates are connected to the common-mode level during the sampling phase.

4.5.1 Parasitic Input Capacitance

The presence of a parasitic input capacitance does not alter the interpolation accuracy of the circuit. A capacitance at the input to the comparator only acts as a capacitive attenuator. The ratio of the total sampling capacitance to the sampling capacitance plus the

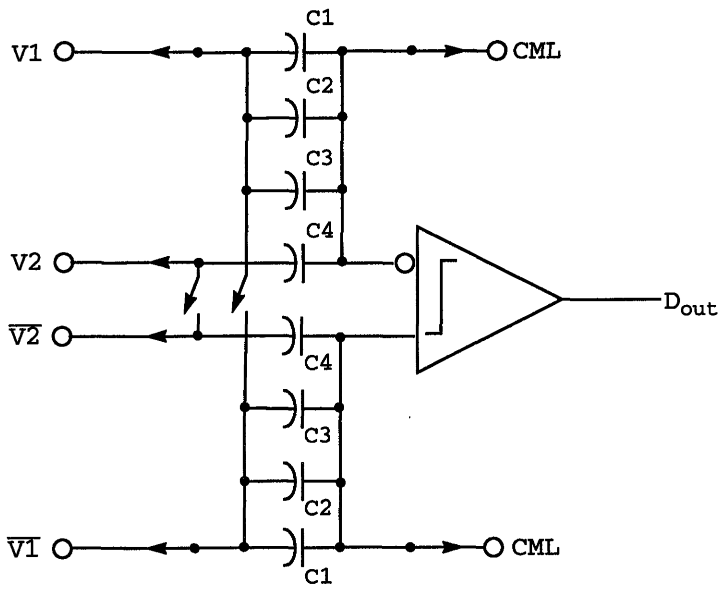

Figure 4.6. Differential implementation of interpolating comparator at ¼ point.

parasitic input capacitance determines the attenuation factor. An attenuation from the input to the interpolator to the comparator will increase the input-referred offset of the interpolating comparator. For most designs this effect will not be particularly important.

4.5.2 Capacitor Mismatch

The effect of capacitor mismatch is similar to the effect seen in the interpolating amplifiers. As shown earlier, expressions (4.14) and (4.15) give the interpolation ratio. It is clear that a ΔC mismatch will create an error on the order of $^{\Delta C}/_{C}$ at the input to the comparator. Typically, this is not an important source of error. The comparator offset will be the dominant source of error in this circuit.

4.6 Conclusion

Using switched capacitor techniques, it is possible to realize accurate interpolation. The ability to use capacitors in lieu of a resistive approach is desirable in CMOS. The design of the amplifiers will be dictated by AC settling performance as opposed to DC drive characteristics.

The sources of accuracy limitation in these circuits are due to finite amplifier gain, amplifier offset voltage, parasitic capacitance and capacitor mismatch. The interpolating ADC architecture is largely insensitive to errors in nominal gain and amplifier offset.

From this standpoint, the amplifier offset and capacitor matching accuracy are not extremely important. The accuracy of the interpolation ratio is not terribly important in the multi-step interpolating architecture either. The interpolation ratio accuracy is limited by the capacitor matching, which is fairly good in CMOS technology [20]. Component matching does not effect the DNL performance of an interpolating ADC significantly, however poor matching can seriously degrade the converter INL performance.

Finally, it is important to note that these capacitive interpolation circuits require an exponential growth in circuit area to realize more interpolation points. In addition to the number of capacitors increasing exponentially, the number of amplifiers increases linearly. A resistive approach, on the other hand, has a ladder that grows linearly and the number of amplifiers remains constant. This trade-off ultimately limits the number of desirable levels of interpolation using a CMOS switched-capacitor approach.

Chapter 5

Low-Voltage / High-Speed Amplifier Design

5.1 General Considerations

Because of the migration to smaller geometry CMOS processes, many system supply voltages have dropped to 3.3V and below. Lower supplies are used in part to protect the CMOS transistors from catastrophic and non-catastrophic failures. More benign effects such as impact ionization can degrade circuit performance. More serious issues such as device breakdown can permanently damage a part. Lowering the supply voltage also has the benefit of reducing power dissipation; however, the loss of dynamic range often encountered by lowering the supply voltage can potentially negate this benefit in analog circuitry.

The switched capacitor circuits discussed in Chapter 4 require high-speed amplifiers to realize fast settling times. Lower supply voltages make the design of high-gain / high-bandwidth amplifiers difficult. It is difficult to add gain-boosting cascode transistors without sacrificing the output dynamic range of the amplifier. As discussed in Chapter 3, the multi-step interpolating architecture does not require the extreme gain accuracy needed in typical multi-step architectures. The decreased gain requirements of the amplifier allow the implementation of a single-stage gain architecture, with a minimum of stacked cascode devices.

5.2 Cascode Amplifier

The cascode amplifier is illustrated in Figure 5.1 is chosen as the amplifier architecture. Cascode devices are only used on the input pair of the amplifier. This configuration allows the input pair to have minimum channel length to maximize the g_m. The output impedance of the input pair is boosted by the addition of the cascode. The cascode has the added benefit of eliminating Miller multiplication of the C_{gd} of the input pair. We can save some dynamic range by not cascoding the pMOS current source loads. Instead, pMOS load devices can be designed to have a longer channel length to lower their g_{ds}.

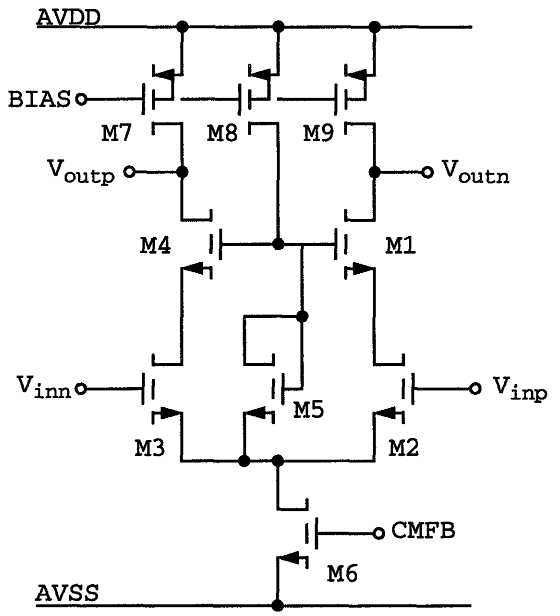

Figure 5.1. Cascode gain stage.

5.2.1 Circuit Description

The cascode devices are biased using a nMOS diode (M5) connected to the sources of the input pair in order to track the input common-mode. By sizing this device, and scaling its drain current, the cascode bias voltage can be set. The bias point is chosen so that the MOS input pair will remain in saturation regardless of reasonable process, temperature and supply variations (this is typically guaranteed by providing 250mV of V_{DS} above V_{DSAT}). The g_m of the bias diode must also be sufficiently large so that the bias node has fairly quick transient settling.

The DC gain of this amplifier is given approximately by expression (5.1). The cascode raises the effective output impedance of the input pair by a factor of g_{m1}/g_{ds1}. Because the gain is essentially determined by a g_m/g_{ds} ratio, increasing the current in the pair actually reduces the gain of the stage. This effect occurs because a MOS g_m increases with the root of the current, as seen in (5.2). However, the output impedance falls linearly with increasing input current, as seen in (5.3). The net effect is that amplifier's gain falls as the root of a bias current increase.

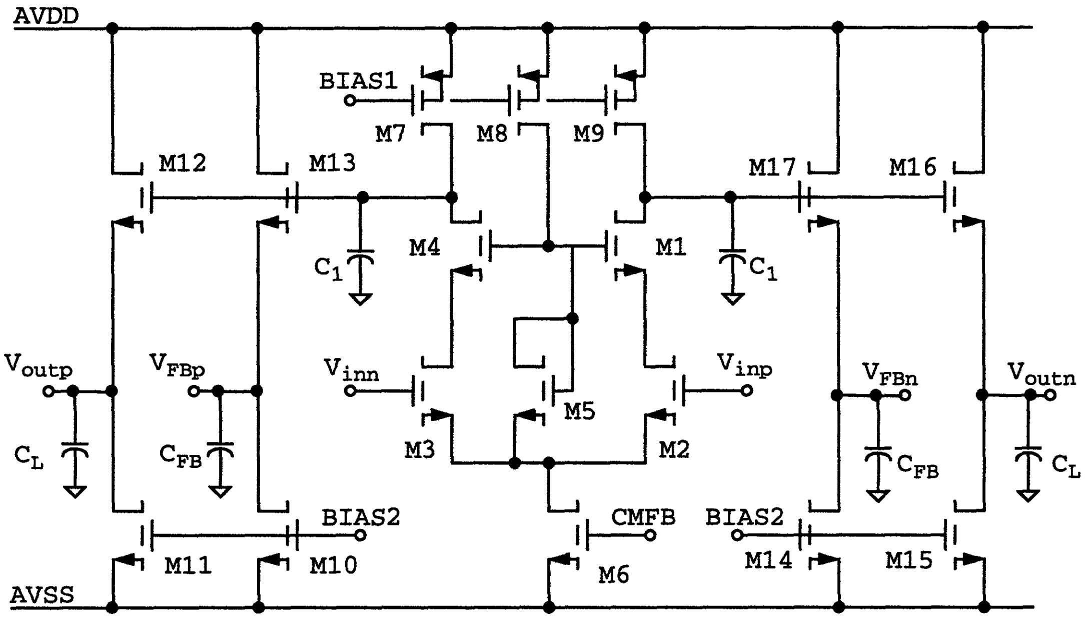

Figure 5.2. Telescopic cascode with dual-follower output stage.

$$A_v \cong -\frac{g_{m2}}{\left(\frac{g_{ds1}g_{ds2}}{g_{m1}} + g_{ds9}\right)} \tag{5.1}$$

$$g_m = \sqrt{2\mu C_{ox}\frac{W}{L}I_D} \tag{5.2}$$

$$g_{ds} = \lambda I_D \tag{5.3}$$

The tail current device (M6) is used to control the common-mode output voltage of the amplifier. By sensing the common-mode voltage, a control signal can be used to modulate the gate of M6 to set an appropriate common-mode level.

The circuit is auto-zeroed by connecting the gates of the input devices to the drains of the cascode devices (V_{g2} to V_{d1} and V_{g3} to V_{d4}). Auto-zeroing serves to null the offset of the amplifier during the reset / auto-zero phase of the switched capacitor circuit. The two cascode devices have plenty of headroom to remain in saturation during the auto-zero phase.

5.2.2 Dual-Follower Output Stage

Figure 5.2 shows the complete amplifier with the output stage. An output-buffered approach was chosen over a single-stage transconductance amplifier due to the capacitive loading the amplifier must drive. Typically, the capacitive output load can be used in a transconductance amplifier to stabilize the amplifier by setting the location of the dominant pole. In the multi-step interpolating architecture, the capacitive load can vary with the input signal by +100%/-50%. This variance in load capacitance creates a large fluctuation in the circuit bandwidth. When the capacitive loading is small, the bandwidth may be too large, creating a stability problem. When the capacitive loading is large, the amplifier will be extremely stable, but the settling will be much too slow.

By using a single gain stage with a source-follower, the load capacitance determines the non-dominant pole. The non-dominant pole's location determines the phase margin of the amplifier, but has much less effect on the settling time. By designing the amplifier to be stable under worst case conditions, the amplifier will be stable at all other corners of operation. Equation (5.4) gives the loop transmission of the amplifier when it is connected in an unity gain configuration. The crossover frequency (ω_c) of this expression is given by (5.6). The crossover frequency is defined as the frequency where the magnitude if the loop transmission falls to unity. In order to maintain a comfortable phase marge the non-dominant pole must be set sufficiently far from the crossover frequency. This is guaranteed by meeting the requirement in (5.7). Adding a feedback network or parasitic input capacitance to the amplifier will act to attenuate the loop-transmission (5.4), this will lower the cross-over frequency of the amplifier (and increase the phase margin).

$$LT(s) = \frac{g_{m2} r_o}{(r_o C_1 s + 1)} \frac{1}{\left(\frac{C_{FB}}{g_{m17}} s + 1\right)} \tag{5.4}$$

$$\frac{1}{r_o} = \frac{g_{ds1} g_{ds2}}{g_{m1}} + g_{ds9} \tag{5.5}$$

$$\omega_c \cong \frac{g_{m2}}{C_1} \tag{5.6}$$

$$\frac{g_{m2}}{C_1}\frac{C_{FB}}{g_{m17}} \ll 1 \quad (5.7)$$

This amplifier topology has the added benefit of having a fairly constant phase margin over process, temperature and bias variations. The location of the crossover frequency and the non-dominant pole are both determined by the nMOS g_m and the poly-poly capacitance of the CMOS process. Any variations will affect the location of the crossover frequency and the non-dominant pole identically. The relative distance between non-dominant pole and crossover will remain constant, so the phase margin will remain constant to first order. This advantage is realized by purposely adding poly-poly compensation capacitance (C_1 in figure 5.2). If the gate capacitance of the source-followers is used to compensate the circuit, the non-dominant pole will no longer track the crossover frequency over process, temperature and bias variations. The non-dominant pole and crossover tracking is lost because the poly-poly capacitance will not track gate capacitance variations.

Each of the amplifier's outputs has a pair of matching source followers. Because the non-dominant pole is set by the source-follower g_m and the load capacitance, it is desirable to minimize the capacitance loading the source follower. One of the followers will be used to drive the feedback capacitor in the switched capacitor circuitry. The other follower will drive the load capacitance of the next stage. Typically, the feedback capacitance is a factor of 10 smaller than the total load capacitance. Incidently, a gain of 4 interpolation circuit was chosen because it minimizes the number of unit feedback capacitors (a gain of 2 interpolation circuit requires 2 unit feedback capacitors as opposed to 1 unit feedback capacitor required for the gain of 4 interpolation circuit). Because the loop is not closed around followers M16 and M12 (which are driving the large load capacitance), they do not contribute a pole to the circuit operation. This configuration allows the source-follower to have a much lower g_m, which ultimately saves power and area. Practically, the amount of capacitance at the sources of M17 and M13 is dominated by device parasitics. The device parasitics ultimately set the upper-bound on the device size that can be used for the followers.

5.2.3 Matching Considerations

The dual source-follower configuration introduces a much larger offset sensitivity to device matching. During the amplify phase, the feedback capacitor is connected to a different source-follower than the output is taken from. The offset of the output source-follower is not nulled by the feedback. This means that we are relying on the "open-loop" matching of the two source followers. For two followers operating at identical current, the offset seen at their sources is given by (5.8). The contribution from V_T mismatch can be reduced by increasing the overall device area (the WL product). The matching is improved further by making W as large as possible. Increasing the channel length (L) does not provide as much benefit because the V_{gs} - V_T terms grows as the square root of any increase in L. Any gains made by reducing $\Delta L/L$ are partially negated by the increasing V_{gs} - V_T term. Note that in a typical design, the mismatch of the source-follower devices dominate the matching performance.

$$V_{os} = \Delta V_T + \frac{1}{2}(V_{gs} - V_T)\left(\frac{\Delta W}{W} + \frac{\Delta L}{L}\right) \tag{5.8}$$

There is also an offset term due to any mismatch in the current sources biasing the followers. The current mismatch is given by (5.9). The offset voltage due to the current mismatch is given by (5.10). The sizing strategy is slightly different for the currents sources to minimize their current offset. It is desirable to use long channel devices. Not only do longer channels reduce the $\Delta L/L$ term, but they increase the V_{gs} - V_T term in (5.9). Again, it is desirable to increase the device area (WL product) to reduce the V_T mismatch.

$$\frac{\Delta I}{I} = \frac{\Delta W}{W} + \frac{\Delta L}{L + \Delta L} + \frac{\Delta V_T}{(V_{gs} - V_T)} \tag{5.9}$$

$$V_{os} = \Delta I \cdot \sqrt{2\mu C_{ox}\frac{W_{sf}}{L_{sf}}I} \tag{5.10}$$

5.2.4 Compensation

If the circuit is not designed for unity gain amplification, it will need additional compensation during its auto-zero phase. During the auto-zero phase, the amplifier is operating in a unity-gain configuration. If the amplifier is designed for a gain of 4 during

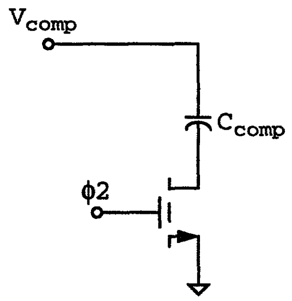

Figure 5.3. Auto-zero phase compensation.

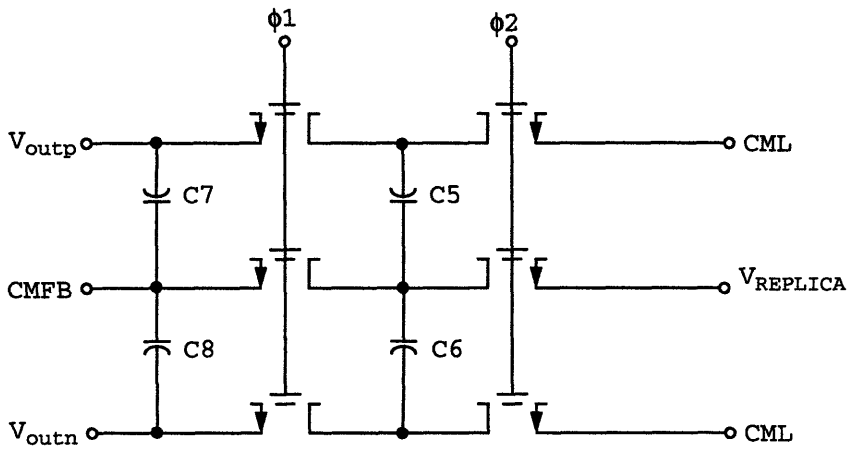

Figure 5.4. Switched capacitor CMFB circuit.

the amplify phase, the bandwidth will be increased by approximately a factor of 4 during the auto-zero phase. This large bandwidth will typically create a stability problem due to parasitic poles present in the circuit.

To alleviate this problem extra compensation capacitance is added to the drains of M1 and M4 during the auto-zero phase. The capacitance can be switched in using a nMOS switch as shown in Figure 5.3. An appropriate amount of added capacitance will lower the crossover frequency of the circuit. The crossover frequency can be adjusted so that a comfortable phase margin is attained during the auto-zero phase of operation.

5.3 Switched Capacitor CMFB

As mentioned in Section 5.2.1, the gate of M6 (Figure 5.2) provides a common-mode control for the circuit. Figure 5.4 illustrates the common-mode feedback circuitry. The

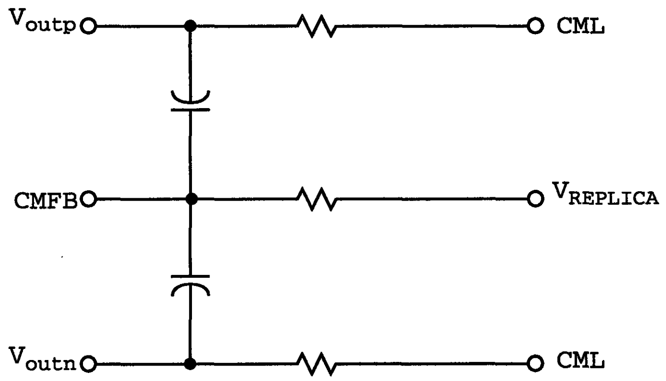

Figure 5.5. Equivalent CMFB circuit using switched capacitor resistors.

nMOS switches are clocked on non-overlapping clock phases. Capacitors C5 and C6 can actually be thought of as switched capacitor resistors as shown in Figure 5.5. By this analogy, the DC path to the CMFB node is biased to $V_{replica}$. Voltage $V_{replica}$ is set so that M6 (in Figure 5.2) is biased to sink a current equal to the three pMOS current sources (M7, M8, and M9). Capacitors C7 and C8 form the AC path which controls the gate of M6 (in Figure 5.2). If the common-mode level of the output is too high (higher than CML), the gate of M6 will be raised which will lower the common-mode level of the outputs. This negative feedback acts to stabilize the common-mode level of the outputs.

5.3.1 CMFB Settling

The crossover frequency of the CMFB loop is given by (5.11). The capacitive attenuation term is due to the capacitive divider formed by the gate of M6 and common-mode feedback capacitors. The g_m of M6 is chosen in order to set the crossover frequency of the loop. The crossover frequency is targeted based on settling speed and stability requirements. The size of M6 also controls the loop gain of the common-mode feedback.

$$\omega_c \cong \left(\frac{C_7 + C_8}{C_7 + C_8 + C_{gs6}}\right)\frac{g_{m6}}{C_1} \tag{5.11}$$

5.4 Simulated Performance

The performance of this amplifier topology was simulated using a gain of 4 switched capacitor circuit. These simulation results are equally applicable to the gain of 4 interpolation circuits discussed in Chapter 4. Table 5.1 summarizes the conditions the

Plot	Bias Current	Temperature	Process Corner
1	+40%	70°C	slow p/nMOS - large C
2	+40%	0°C	fast p/nMOS - small C
3	+40%	0°C	fast p/nMOS - large C
4	-40%	70°C	slow p/nMOS - large C
5	nominal	27°C	nominal

Table 5.1. Key to amplifier settling plots.

amplifier settling was simulated under. The circuit was simulated using 2.5V analog and digital supplies. At nominal bias the circuit draws 740μA from the analog supply. With 2.5V supplies, the amplifier dissipates ~1.8mW of power. The process corners in Table 5.1 are selected to demonstrate the robustness of the amplifier over wide variations of process, temperature and bias current. The bias current variation described in Table 5.1 is the variation from the nominal bias current of 740μA. Finally, the amplifier is designed to have $1V_{p\text{-}p}$ differential output drive capability.

Figure 5.6 illustrates the settling for a 1/2 FS step input. The amplifier settles comfortably to 0.15% in 20ns over all process, temperature and supply current variations (the solid vertical bars are separated by ~20ns). Note that percentage settling to the final value is important, not settling to a defined absolute voltage (this feature is specific to the interpolating architecture the amplifiers are being designed for). Plot 4 corresponds to the slowest settling time, due to fact it is simulated at the slow corner at -30% bias current and high temperature. Plot 2 has the fastest settling time and is simulated at the fast process corner at +40% supply current and low temperature. Note that although the settling times vary, the phase margin corresponding to each response is nearly identical. This result was expected by using an amplifier with the crossover frequency and the non-dominant pole set by the nMOS g_m and the poly-poly capacitance. As predicted, the non-dominant pole tracks variations in the crossover, resulting in a fairly constant phase margin.

Figure 5.7 shows the reset mode settling of the gain of 4 amplifier. Again, over all corners, the amplifier resets comfortably to $<{}^{1}\!/_{10}$ LSB in 20nS. Note that in the first stage of a multi-step converter with an 1V full-scale, 1 LSB corresponds to ~1mV. As mentioned in Section 5.2.4, during the reset phase, additional compensation capacitance

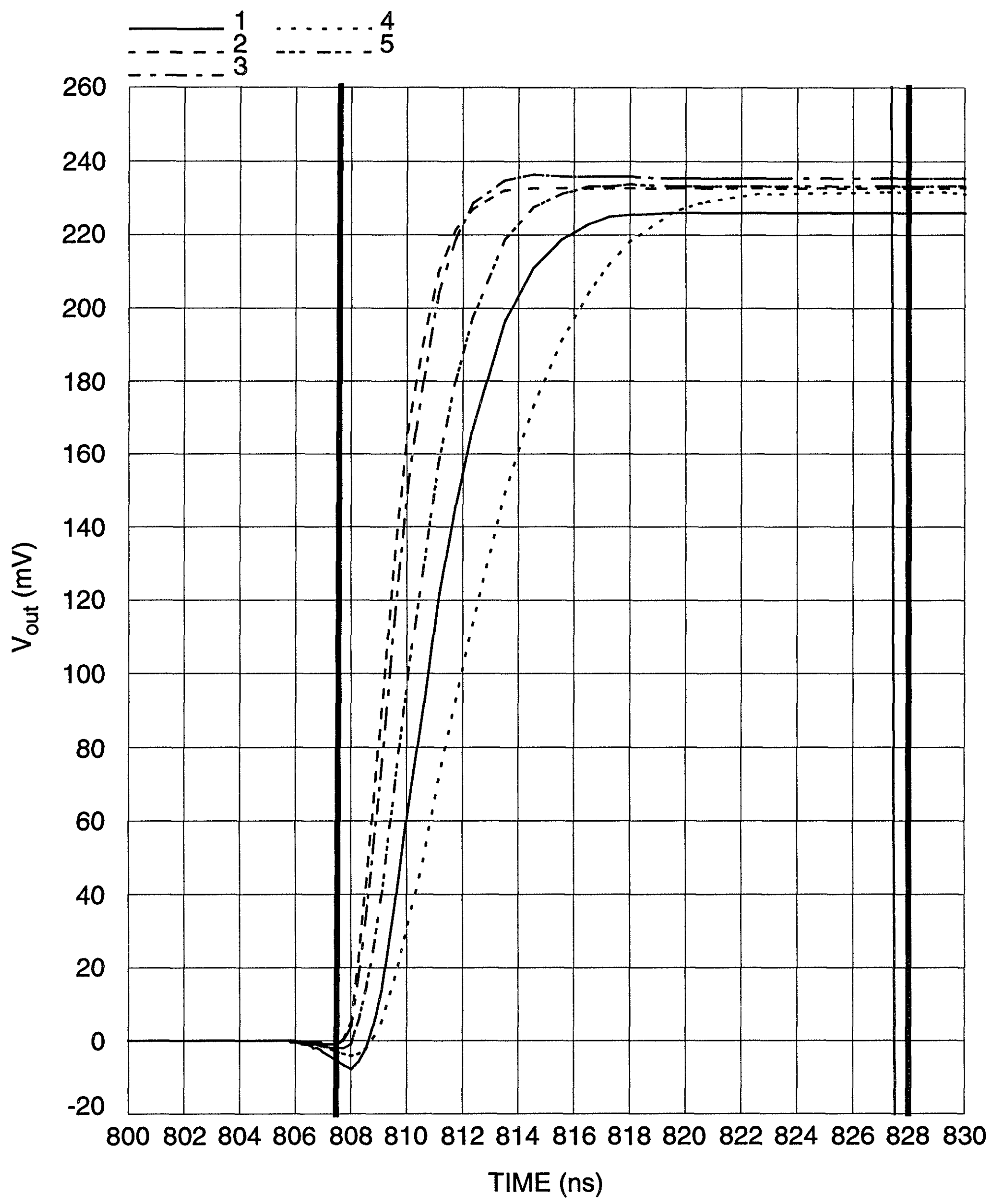

Figure 5.6. Gain of 4 amplifier settling over process, temperature and bias variation.

must be added to maintain a comfortable phase margin. As can be seen in Figures 5.6 and 5.7, this technique is effective and produces well behaved settling in both amplify and reset phases.

The performance of the amplifier configured for a closed-loop gain of 4 is summarized in Table 5.2. As discussed earlier, the phase margin remains fairly constant over a wide range of process, temperature and bias levels. Based on the simulated slew rate, this

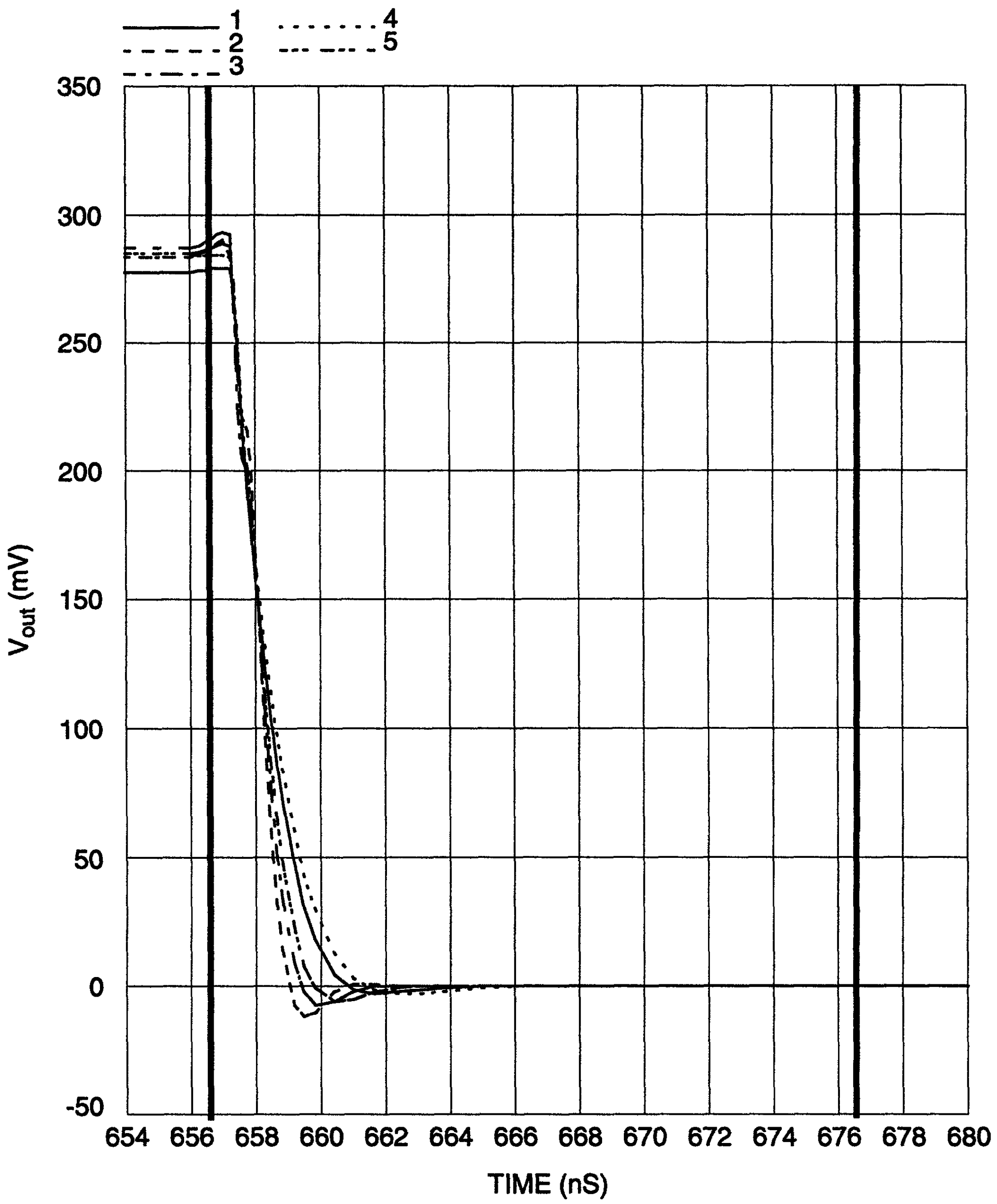

Figure 5.7. Reset mode settling over process, temperature and bias variation.

amplifier should have fairly linear settling for a full-scale step response. Linear settling is expected because the derivative of a full-scale step response is equal to V_{FS}/τ when evaluated at *time* = 0 (this is the initial transient slope). For these amplifiers, the initial transient slope is fairly close to the slew rate performance of the amplifiers. Finally, the output dynamic range safely meets the design goal of $1.0V_{p\text{-}p}$ over process corners.

	Nominal	Fast Corner	Slow Corner
open-loop amplifier gain	39.5dB	41.3dB	35.7dB
loop gain ($\frac{1}{4}$ feedback)	22.8dB	23.4dB	19.7dB
crossover frequency	70MHz	120MHz	32MHz
phase margin	83°	82°	86°
settling time (0.15%)	12.5ns	6.5ns	20.1ns
slew rate	180V/μs	270V/μs	110V/μs
output swing range	$1.7V_{p\text{-}p}$	$1.3V_{p\text{-}p}$	$1.4V_{p\text{-}p}$
power supply voltage	2.5V	2.5V	2.5V
power consumption	1.84mW	2.58mW	1.31mW
temperature	27°C	0°C	70°C
process corner	nominal	fast p/nMOS small C	slow p/nMOS large C

Table 5.2. Summary of amplifier performance.

5.5 Conclusion

Using a single-gain stage, it is possible to achieve the gain performance required in the multi-step interpolating architecture. By adding a source-follower output stage, the architecture's settling characteristic is made independent of the load capacitance. By modifying this further to a matched dual-follower output stage, the requirements on the output impedance of the follower are reduced greatly. This results in a significant power and area savings. The amplifier power is kept under 2mW, but the amplifier still gives the desired settling performance when used in 20MHz clock-rate switched capacitor amplifiers and interpolators.

The drawback of the chosen architecture is the increased offset added by relying on the open-loop matching of the source-followers. The interpolating multi-step architecture has a low sensitivity to amplifier offset. Trading off some offset performance for large power savings is a reasonable decision based on the constraints of this architecture.

Chapter 6

A/D Converter Design

6.1 Overview

Using the building blocks described in the previous chapters, this chapter outlines the design of a 10-bit, 20MSPS pipelined multi-step interpolating ADC. This design is intended for implementation in CMOS with 2.5V analog and digital supply voltages. The circuit blocks discussed in Chapters 4 and 5 are all designed for 2.5V operation. This chapter will begin with a top-level description of the architecture. The top-level description will be followed by a closer look at the individual blocks. Using predicted performance from the individual circuit blocks, a top-level behavioral simulation is performed. The top-level simulation is intended to predict the linearity performance of the converter.

This converter is targeted for video applications, where DNL performance is emphasized over INL performance. Also, the ability to design at low supply voltages with small power dissipation is extremely important for portable applications. The chosen architecture presents advantages that make it desirable for low voltage design and possibly low power design.

6.2 ADC Architecture

The top-level block diagram of the 10-bit pipelined multi-step converter is shown in Figure 6.1. This is a 3-3-3-3-2 architecture (3 bits per stage in the first four stages and 2 bits in the last stage). This multi-step pipeline is analogous to a standard pipelined multi-step converter (as discussed in Section 2.5). The primary difference is that the residues in this architecture self-define their reference range. This difference is due to the interpolating techniques that are employed.

There are only a few important circuit blocks in Figure 6.1. The input block converts the input signal into a four-residue representation. The signal path is handled by 4 interpolation blocks along the top of the figure. The signal path is represented by the four

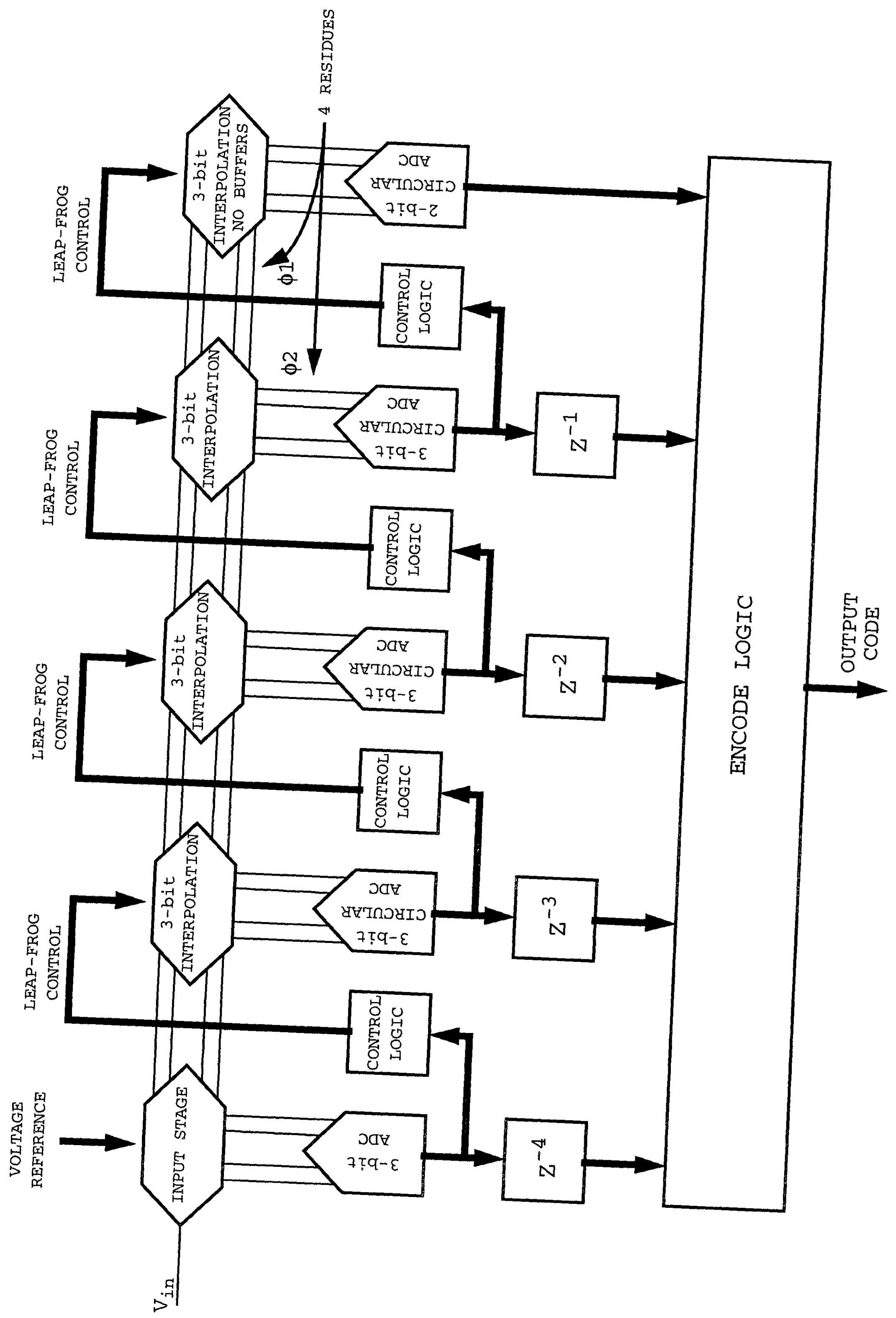

Figure 6.1. Top-level block diagram of 10b pipelined multi-step interpolating ADC.

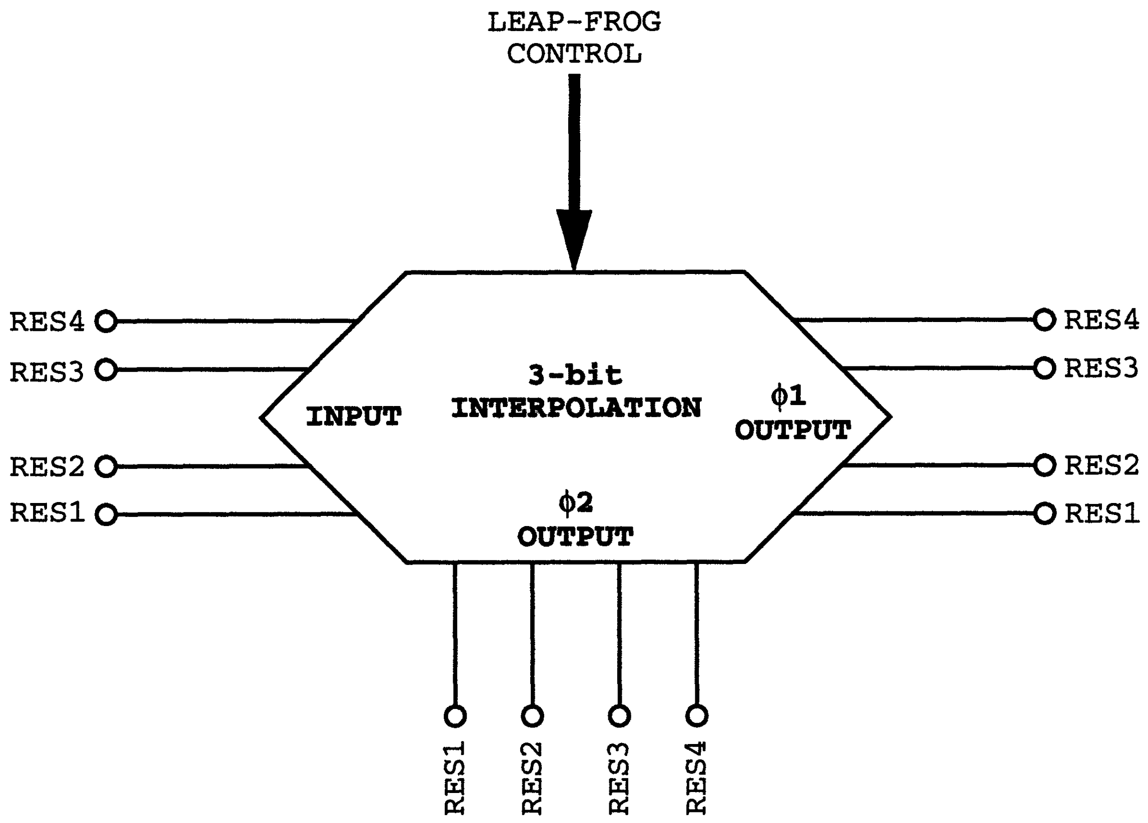

Figure 6.2. Description of 3-bit interpolator input/output terminal.

residues propagated from block to block. Figure 6.2 clarifies the input/output terminals on the 3-bit interpolating blocks.

As shown in Figure 6.2, each interpolation block has two sets of 4-residue outputs. The residues that are sent to the inter-stage quantizer are available on φ2 of the system clock. The 4-residue signal sent to the next interpolation stage is available on φ1 of the system clock. This two phase operation in necessary so that the next interpolation stage has the previous stage's quantization result in time to switch its interpolation network into place. Although all of the figures are drawn with a single-ended implementation, the actual converter is implemented using a fully differential signal path.

6.2.1 Input Stage

The input stage to the converter is shown in Figure 6.3. The primary purpose of this stage is to generate the four residues that will be used for subsequent interpolation. The reference ladder provides equally spaced tap voltages which are brought to the inverting inputs of the residue amplifiers. The input range only extends from halfway between input

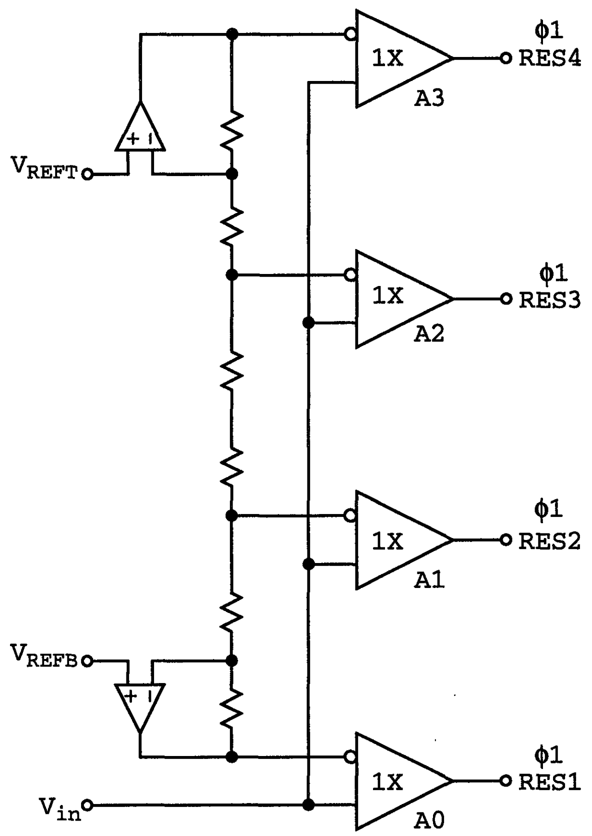

Figure 6.3. Input stage for 4-residue interpolating converter.

taps for amplifiers A0 and A1 to halfway between amplifiers A2 and A3. The reference servo amplifiers bias the ladder so that V_{REFT} and V_{REFB} represent the full-scale input range. The reference servo is not required in the actual implementation. The actual implementation is fully differential, so the required voltages can be generated easily using a differential voltage ladder.

The first stage is also connected to a 3-bit flash converter. This flash performs the first or "coarse" conversion. As can be seen from Figure 6.1, the result from this flash conversion is used for two purposes. First, the result is passed to a bank of delay elements which time-aligns each stage's converted result. Time-alignment is necessary because the pipelined architecture is operating on several data samples simultaneously. Also, the result from the first stage flash is passed to the logic that will control the leap-frog switching in the next stage's interpolation.

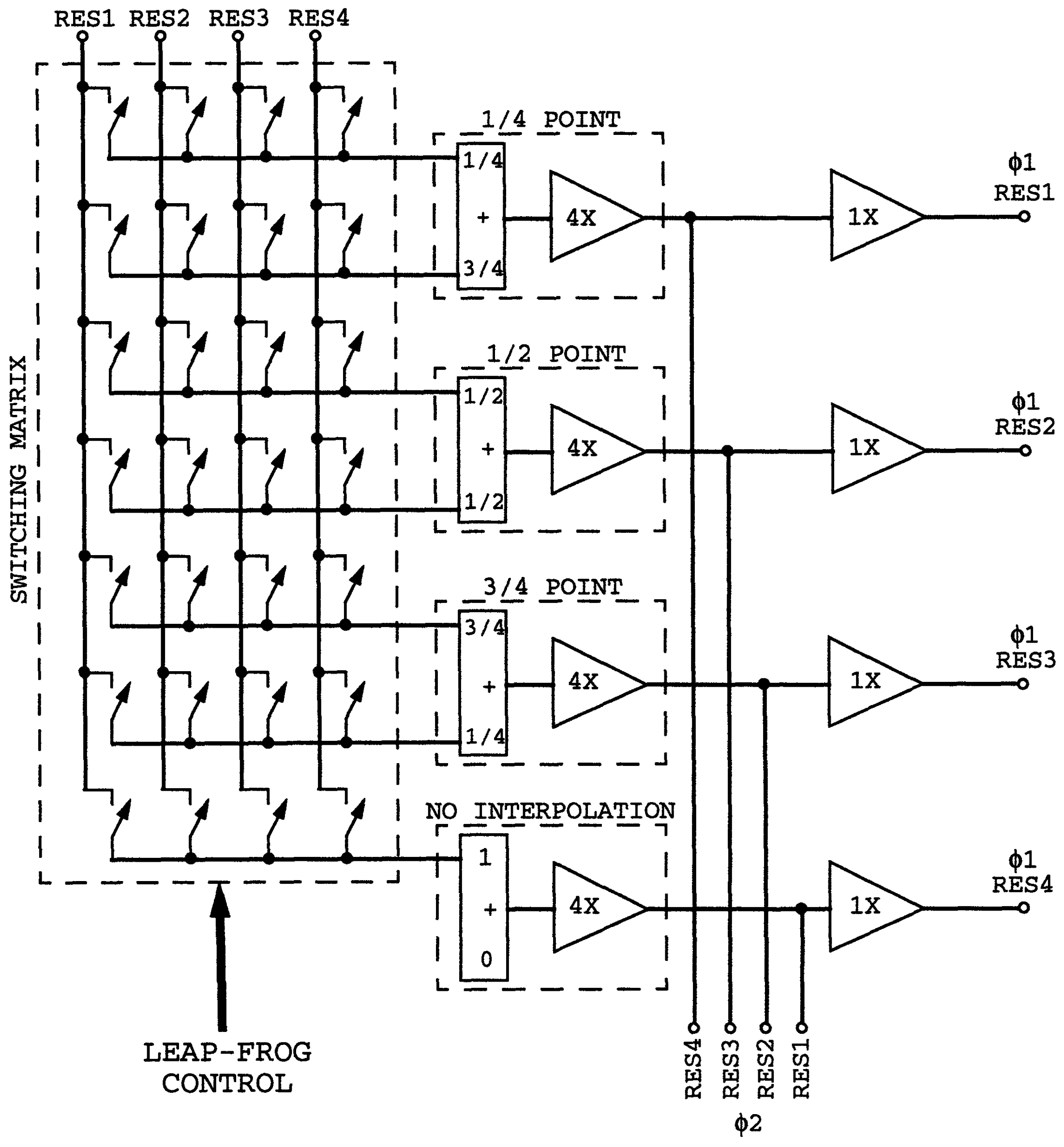

Figure 6.4. 3-bit interpolation stage.

6.2.2 3-bit Interpolation Stage

A functional diagram of the 3-bit interpolation stage is shown in Figure 6.4. Using the result of the previous stage's conversion, this stage will switch its interpolating residue amplifiers to the appropriate residue taps from the previous stage. As mentioned in Section 3.4.2, in a leap-frogged 4-residue system with 4 levels of interpolation, the interpolation network can be permanently connected to each input amplifiers. In other words, each amplifier always performs the same interpolation function. Each amplifier is switched to the two residues that it will be interpolating between. The previous stage informs the leap-frog control logic where the interpolated zero crossing occurs. Based on this information, the switching matrix is controlled so that this residues are interpolated and amplified in the region of the zero-crossing.

Notice that the stage illustrated in Figure 6.4 has its outputs available on both clock phases because of the addition of the unity gain buffer. The $\phi 2$ residue outputs are used by the interpolating quantizer. The $\phi 1$ residue outputs are used to drive the interpolating amplifiers in the next stage. This two phase operation is necessary to allow the inter-stage interpolating quantizer time to make its decision. The inter-stage quantizer needs the residue outputs one phase early so that it can switch the leap-frog switching for the next stage. The next stage must be switched into place before the unity gain buffer amplifiers go into their amplify phase. By utilizing this two-phase timing the architecture can be pipelined to increase the throughput of the converter.

6.2.3 Circular Quantizer

As mentioned above, the residue outputs on $\phi 2$ are used to drive the circular interpolating quantizer. The function of this quantizer is to resolve (to 3 bits) the location of the interpolated zero-crossing. Conceptually, this function is illustrated in Figure 6.5. Because leap-frog switching is utilized (discussion in Chapter 3), the nominal, under- and over-range are not constrained to occur between any two particular amplifiers. Because of the uncertain location of the nominal range, a circular quantizer is required that can locate a zero-crossing between any two amplifiers. In the example shown in Figure 6.5, the zero-crossing is indicated by the $1 \rightarrow 0$ transition as you move upwards along the "thermometer" code. The $0 \rightarrow 1$ transition also indicates a zero-crossing, but this zero-crossing occurs wherever the quantization ladder wraps back around to the "bottom." The location of this wrap-around changes as the interpolating residue amplifiers are leap-frogged up and down.

The implementation of the circular interpolating quantizer is shown in Figure 6.6. Each comparator has its own interpolation network associated with it. (The circuit implementation of the interpolating comparator is discussed in Section 4.5.) The residue inputs to this circular quantizers are taken from the $\phi 2$ outputs of the interpolation stage. As mentioned earlier, this allows time for the result of this quantization to be used to

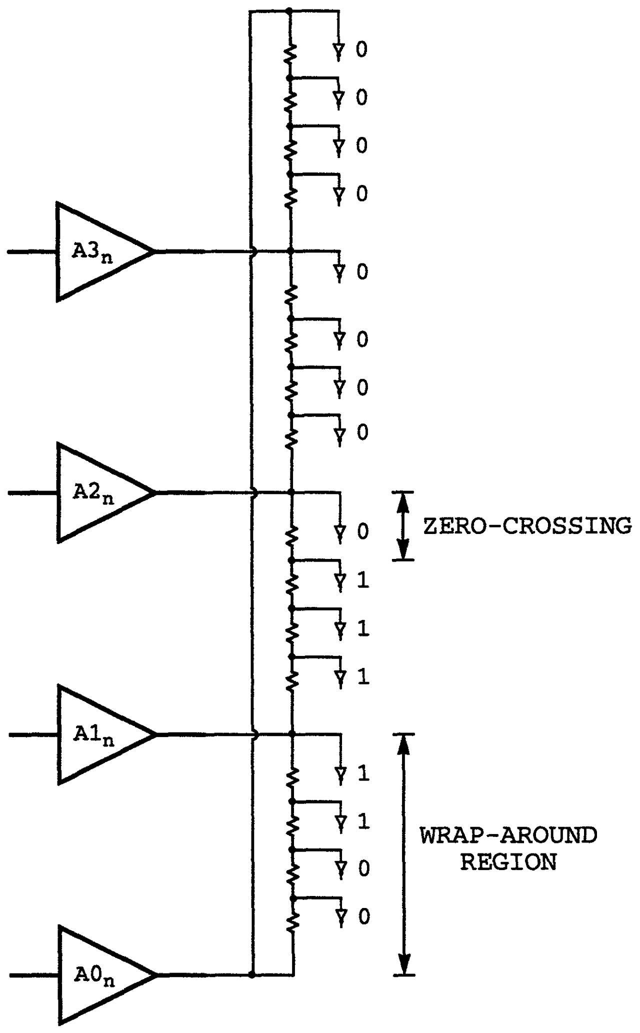

Figure 6.5. Conceptual diagram of 3-bit circular ADC.

switch the next interpolating stage into place. The outputs from this quantizer are also sent to the digital encode logic.

6.2.4 Time Alignment

As shown in Figure 6.1, the outputs from each stage's quantizer are passed though a delay stage. Because this converter is pipelined, it is necessary to time-align the outputs of each quantizer. This is performed so that each stage's quantized result for a given input signal will reach the decode logic on the same clock phase.

6.3 Simulated Performance

The primary circuit effects that are liable to degrade the performance of this converter are the capacitor matching, residue amplifier offset, residue amplifier settling and comparator

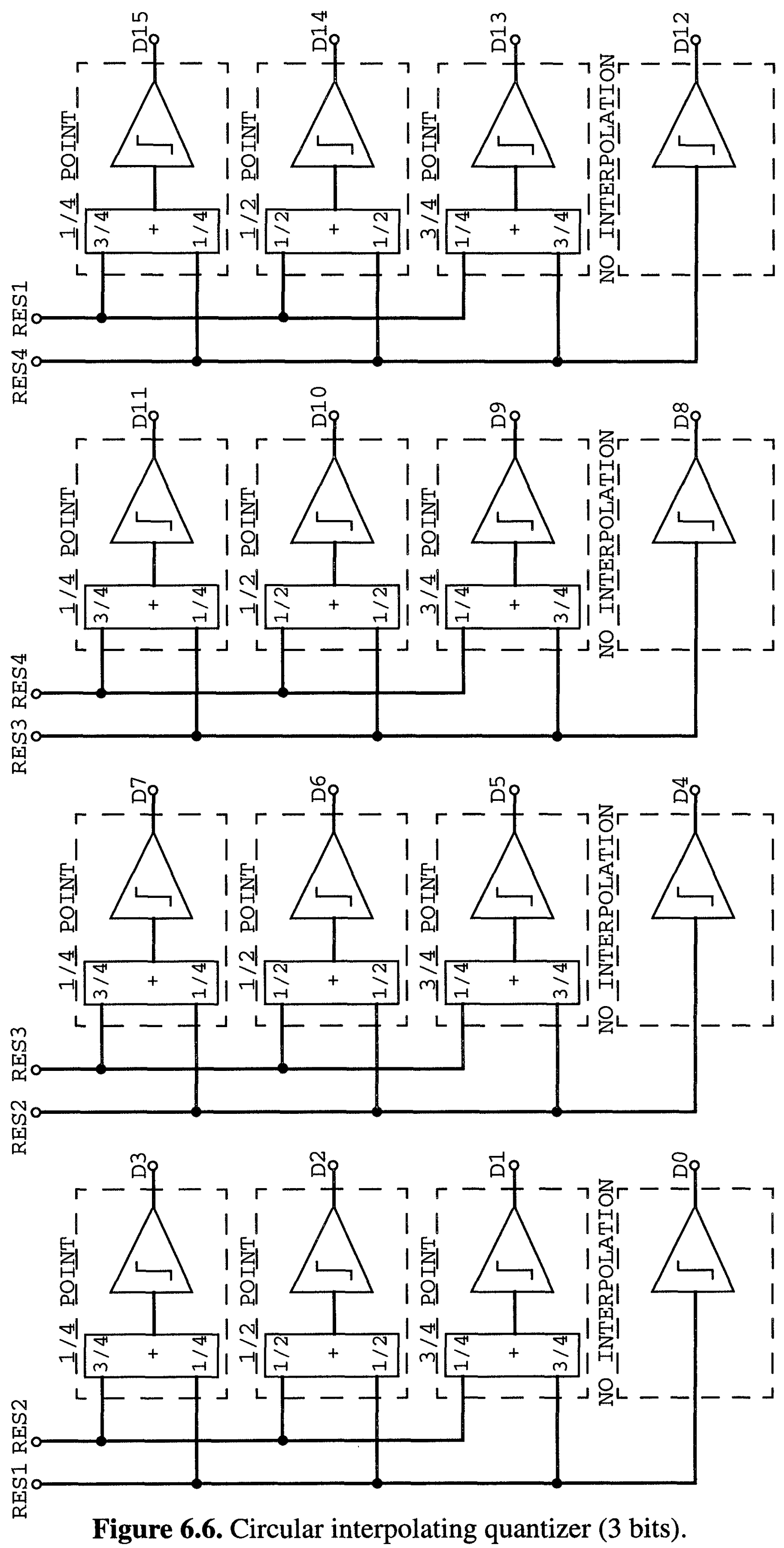

Figure 6.6. Circular interpolating quantizer (3 bits).

offset. The capacitor mismatch contributes to create gain mismatch between inter-stage amplifiers. Gain mismatch creates INL and DNL as discussed in Section 3.3.1. Capacitor mismatch also creates an interpolation error. Interpolation errors can also degrade converter INL and DNL. Amplifier offset contributes to both the DNL and INL of the converter as discussed in Section 3.3.2. Comparator offset is typically corrected for in multi-step converters by the addition of an overlap range. Amplifier settling errors appear as a gain error and affect the converter performance in a similar manner to a gain error.

In Chapter 5, the amplifier settling was tested at 2.5V supplies and settled to 0.15% in 20ns over bias, supply and temperature variations. Based on calculations, the amplifier settling will contribute negligible errors to the architecture. As mentioned in Section 3.3.1, if the settling of the amplifiers are matched then they do not contribute to converter error. With 0.15% settling we are guaranteed $\frac{1}{10}$ LSB performance at 10-bits even with mismatched settling. Based on these results, settling errors were left out of the behavioral simulation.

Table 6.1 gives the performance expectations of the offset and capacitor matching in a

Parameter	σ
comparator offset	<10mV
amplifier offset	<10mV
capacitor matching	<0.1%

Table 6.1. CMOS process characteristics.

typical CMOS process. Using the predicted distribution of these values, a top-level behavioral model was constructed. According to a normal distribution, with the given 1-sigma value, component values are assigned to each capacitor, amplifier and comparator in the converter. The behavioral model performs a "DC sweep" of the converter's transfer function. The sweep speed is set such that each code will be "hit" 100 times nominally. Based on a code hit density analysis, the width of each code can be calculated. The code widths correspond to the simulated DNL performance of the converter. The integrated DNL plot gives the INL performance of the converter. Figure 6.7 shows the DNL and INL

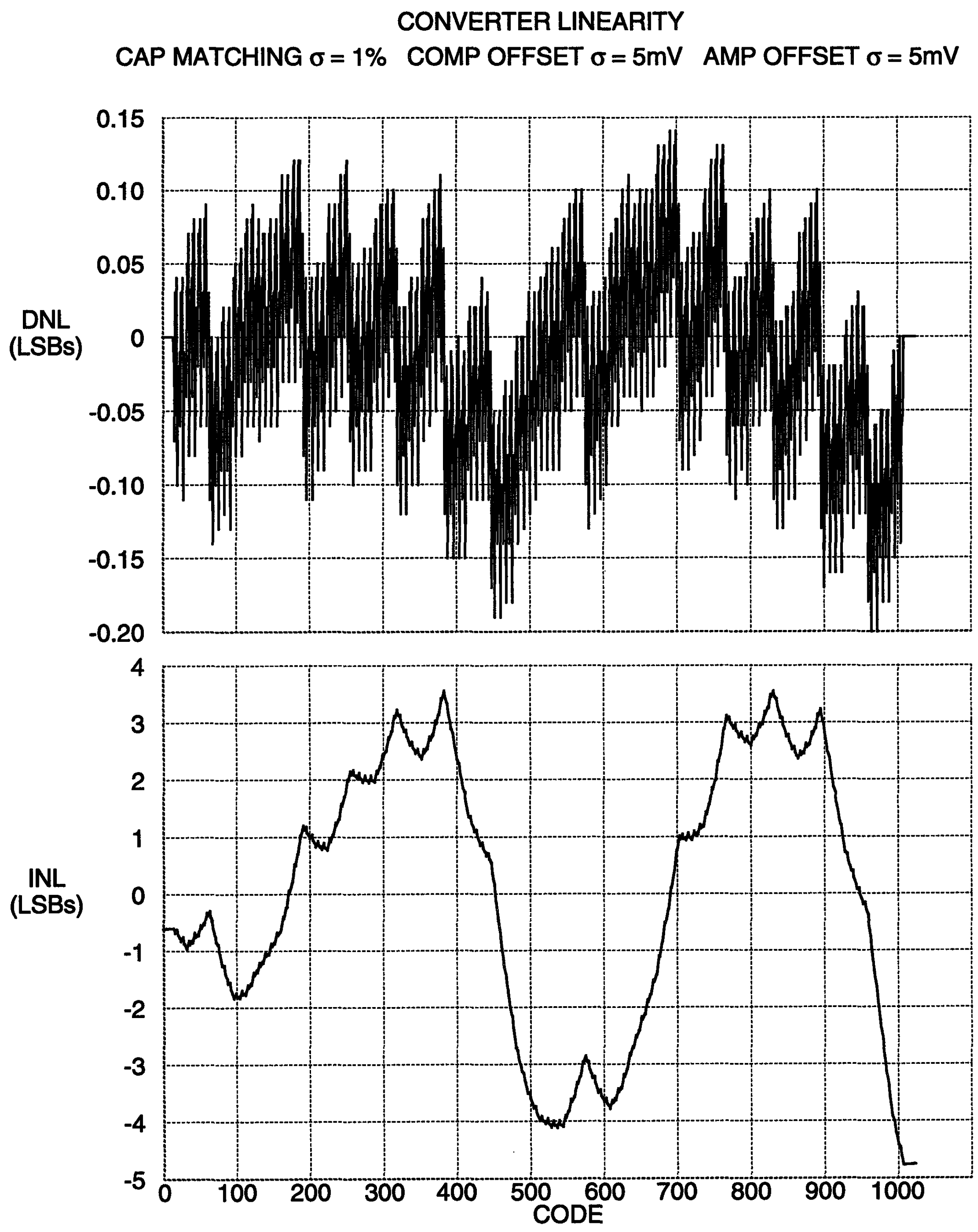

Figure 6.7. Behavioral simulation of converter linearity (DNL and INL).

plots for a randomly generated converter. This plot is fairly typical of those seen after performing several simulations with the given distribution of component values.

The DNL performance shown in Figure 6.7 is within the design goals. The simulated INL performance would be unacceptable for many applications (such as communications)

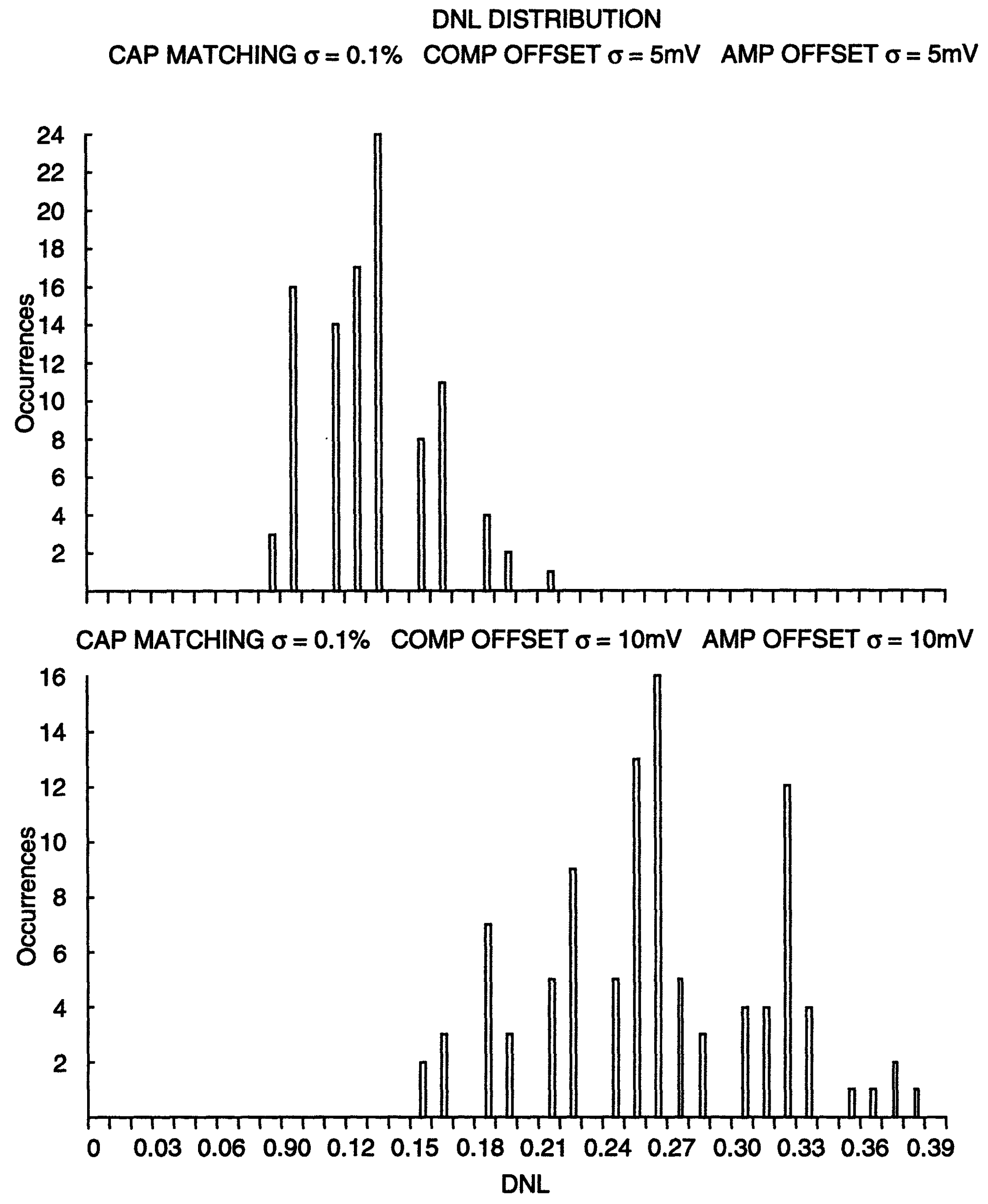

Figure 6.8. Monte-Carlo analysis of worst case linearity error (DNL).

that require good distortion performance. The primary effect which causes the INL degradation is the poor offset performance of the residue amplifiers. By using an amplifier topology with better offset performance the INL could be significantly improved. As an added benefit, the DNL performance would be improved further.

Figure 6.8 was generated by doing 2 sets of monte-carlo analysis using different component mismatch distributions. For each histogram, 100 converters were generated based on the labeled 1-sigma values. The DNL of each converter was extracted. From each converter, the code with the worst DNL was selected and logged. Even for fairly dramatic component errors, the DNL performance remains well within $\frac{1}{2}$ LSB at 10-bits. The limiting factor in the DNL performance is the amplifier offset performance. A factor of 2 improvement in the amplifier offset results in a corresponding factor of 2 improvement in the DNL performance.

6.4 Conclusion

The 10-bit implementation of a pipelined multi-step interpolating converter appears promising from a number of aspects. The most notable feature is the ability to use low-gain amplifiers (~30-40dB) and still maintain excellent DNL performance. The amplifiers are able to operate at 2.5V analog and digital supplies, and do not encounter headroom problems over temperature, bias and process variation. Also, with extremely poor offset performance from the amplifiers, $\frac{1}{2}$ LSB DNL performance is attainable. The offset performance of the amplifiers is the effect which most limits the linearity performance of the converter.

The converter operates at 20MSPS with supplies down to 2.5V. Table 6.2 gives the

	#	**POWER (2.5V)**
Amplifiers	32	1.84mW
Comparators	63	100μW
TOTAL		65mW

Table 6.2. Power accounting for 20MSPS ADC.

power accounting for the analog portion of the proposed converter. The expected power dissipation of the analog portion of the converter is 65mW at nominal bias conditions. The excellent DNL performance and operation at low voltage supplies indicates strong potential for portable video applications.

Chapter 7

Conclusion and Suggestions

7.1 Conclusion

Pipelined multi-step converters in CMOS have achieved 10-bit performance at 20MSPS [12] and [13]. Many of these designs rely on high-gain / wide-bandwidth amplifiers to achieve accurate interstage gain performance. These amplifiers typically require a large design effort and consume a fair amount of power. As portable video and other consumer applications require lower supply voltages, it is anticipated that a standard approach to multi-step conversion (such as a MDAC approach which is used in CMOS technologies) will become cumbersome, if not impossible to implement. As this research has presented, through the use of an alternative architecture it is possible to lower the performance requirements on amplifiers.

A multi-residue approach to conversion has been shown in this research to ease amplifier requirements in converter design. This work has demonstrated that the required circuits can be designed in the constraints of a 2.5V system supply. The amplifiers only require 30-40dB of gain performance which can be easily obtained in a single gain stage. This research shows that the required amplifiers are able to settle to the required accuracy in 20ns without using excessive power.

In design, when the demands on one circuit block are reduced, typically the demands on other circuit components are increased to offset this. In the interpolating multi-residue approach this does not seem to be the case. As well as reduced amplifier performance, the architecture can tolerate moderate component matching. The accuracy requirements of the comparators are the same as those in a standard CMOS pipelined converter. The reduced accuracy of the amplifiers and component matching degrades the INL performance, but has little effect on DNL.

The DNL performance of this converter remains excellent for moderate component matching and single-stage amplifiers. The INL performance is degraded moderately due to the component non-idealities. As discussed in Chapter 6, amplifier offset accounts for the majority of linearity loss. This work has emphasized DNL performance over INL performance. The DNL performance is a much more important than INL for video applications, where display units have limited linearity to begin with.

In order to demonstrate the advantages of the pipelined multi-step interpolating converter, a potential circuit implementation was presented. Using a combination of circuit simulation and top-level behavioral simulation, the feasibility of this architecture was demonstrated. With predicted component matching, this implementation is expected to achieve $1/2$ LSB DNL performance at 10-bits. The INL performance is not quite as outstanding and appears to be ±4 LSB typically. The INL is degraded primarily because of offset in the residue amplifiers. Any improvements here would significantly improve the INL (and as a side benefit, further improve the already excellent DNL). The suggested converter implementation appears to achieve 10-bit 20MSPS performance at 2.5V supply voltages. The analog circuitry is expected to dissipate 65mW.

A concern with a new architecture is the ease of implementation. The switching involved in the implementation of this architecture is extensive; however, it is not significantly more complicated than the switching used in MDAC-based pipelined architectures. The digital encoding for this architecture is also different than a standard MDAC approach. These digital issues should not present a significant impediment to architecture implementation.

7.2 Future Work

In the course of the research, many ideas have surfaced which may merit further research:

- Amplifier offset creates the largest source of performance degradation in this architecture. In the presented implementation, the dual-follower output stage is dominant source of offset. The architecture relies on the open-loop matching of MOS source-followers. By implementing this architecture in BiCMOS, bipolar devices could be used for the matched dual-followers. Because the expected V_{BE} matching of bipolar devices is typically better than 1mV, this could potentially improve the INL and DNL performance by a factor of 10. In addition, the bipolar device can achieve the required g_m with half the current. This savings would cut the power consumption of the amplifiers by roughly half - and by extension almost halve the power of the converter.

- Another method of reducing the amplifier offset is to use an amplifier without the dual-follower. A single-stage transconductance amplifier could be implemented (e.g. folded cascode). To keep the amplifier stable, dummy capacitors would have to switched to the outputs of the amplifier whenever interpolating capacitors are switched off the output.

• As seen in [17], if the settling in the residue amplifiers matches, they do not need to settle completely. An interesting experiment to run would be to cut the current by half to slow the settling. The performance of the converter should remain largely unchanged.

• Trading off resolution for speed in this architecture should be relatively straightforward. Investigation of extremely fast, lower resolution (8 bits) converters would be worthwhile.

Appendix A

Behavioral Simulation of Multi-Step Interpolation Architecture

A.1 Introduction

The following C code was written to test the architecture's sensitivity to component mismatches. Each run of the simulation assigns a random value to all components in the architecture. The values are based on a normal distribution with sigma values chosen before compilation. The behavioral simulation of the converter is run with a ramp input. The ramp "speed" is chosen that that each code is hit 100 times nominally. Based on a code hit density analysis, DNL and INL data is generated for each code. This text file can be imported into most data analysis programs to be viewed.

A.2 Behavioral Model

```
/************************************************************/
/** Multi-step Interpolating ADC - Behavioral Simulation   **/
/**                                                        **/
/** ACTION: simulates ramp input, with 100 hit/code.       **/
/**         based on code hit density analysis, DNL and    **/
/**         INL are computed and written to 'dnl.dat' and  **/
/**         'inl.dat' respectively.                        **/
/**                                                        **/
/**         component values are assigned according to a   **/
/**         normal distribution with sigma assigned        **/
/**         by user.                                       **/
/**                                                        **/
/************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#define STAGES 5
#define ARCH 33332       /* 3-3-3-3-2 architecture - do not alter */
#define MAX_LEVELS 16
#define SIG_LAD 0.01     /* ladder resistor sigma - mean = 1      */
#define SIG_COMPOFF 0.01 /* comparator offset sigma - volts       */
#define SIG_AMPOFF 0.005 /* amplifier offset sigma - volts        */
#define SIG_CAP 0.01     /* capacitor mismatch sigma - delta_C/C  */
#define REFT 1           /* input range 0-1V                      */
#define REFB 0

/** CONSTANTS FOR UNIFORM DEVIATE **/
#define M1 259200
#define IA1 7141
```

```
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349
/** END CONSTANTS FOR UNIFORM DEVIATE **/

double normal(double, double);
float gasdev(int *);
float ran1(int *);
void initialize(int *, double *,double [][], double [][], double [][], double [][], \
       double [][], double *, double [][], double [][]);
int convert(double, int *, double [][], double [][], double [][], double [][], \
       double [][], double [], double [][], double [][]);
void captaps(int, int, double *, double *, double [][]);
void interpol(int, int, double *, double [][]);
void resistaps(double *, double *, double);
void bin2dnl(double *);
int decode(int *, int *);
int adjust(int, int *, int);
void simout(double *);
double amp1(double, double, double);
double amp2(double, double, double);
int flash(int, int, double [][], double *);

main()
{
    int bits[STAGES],code;
    double amp1off[STAGES][4];
    double amp2off[STAGES][4];
    double compoff[STAGES][MAX_LEVELS];
    double gain1[STAGES][4];
    double gain2[STAGES][4];
    double ladder[MAX_LEVELS];
    double caps1[STAGES][MAX_LEVELS*MAX_LEVELS/4];
    double caps2[STAGES][MAX_LEVELS];
    double bins[1024];
    double vin;

    initialize(bits,bins,amp1off,amp2off,compoff,gain1,gain2,ladder,caps1,caps2);
    for (vin = -0.5; vin < 0.5; vin = vin + 1.0/102400.0) {
        code = convert(vin,bits,amp1off,amp2off,compoff,gain1,gain2,ladder,caps1,caps2);
        if ((code >= 0) && (code < 1024))
            ++bins[code];
    }
    bin2dnl(bins);
    simout(bins);
}

int convert(double vin, int *bits, double amp1off[STAGES][4], \
       double amp2off[STAGES][4], \
       double compoff[STAGES][MAX_LEVELS], \
       double gain1[STAGES][4], \
       double gain2[STAGES][4], \
       double ladder[MAX_LEVELS], \
        double caps1[STAGES][MAX_LEVELS*MAX_LEVELS/4], \
        double caps2[STAGES][MAX_LEVELS])

{
    double res[4];
    double taps[MAX_LEVELS];
    int code[STAGES],stage,result;

    resistaps(taps,ladder,vin);
    res[0] = amp2(taps[0],gain2[0][0],amp2off[0][0]);
    res[1] = amp2(taps[4],gain2[0][1],amp2off[0][1]);
    res[2] = amp2(taps[8],gain2[0][2],amp2off[0][2]);
    res[3] = amp2(taps[12],gain2[0][3],amp2off[0][3]);
    code[0] = flash(bits[0],0,compoff,taps); /* put catch in for code out of range */
    for (stage = 1; stage < STAGES; stage++) {
        interpol(code[stage-1],stage,res,caps2);
        res[0] = amp1(res[0],gain1[stage][0],amp1off[stage][0]);
        res[1] = amp1(res[1],gain1[stage][1],amp1off[stage][1]);
        res[2] = amp1(res[2],gain1[stage][2],amp1off[stage][2]);
        res[3] = amp1(res[3],gain1[stage][3],amp1off[stage][3]);
        captaps(stage,bits[stage],res,taps,caps1);
        code[stage] = flash(bits[stage],stage,compoff,taps);
        res[0] = amp2(res[0],gain2[stage][0],amp2off[stage][0]);
        res[1] = amp2(res[1],gain2[stage][1],amp2off[stage][1]);
        res[2] = amp2(res[2],gain2[stage][2],amp2off[stage][2]);
        res[3] = amp2(res[3],gain2[stage][3],amp2off[stage][3]);
    }
    result = decode(bits,code);

    return result;
}

int decode(int *bits, int *code)
{
    int loop,weight,result;

    result = 0;
    weight = 1024;
    weight = weight / ldexp(1,bits[0]);
    result = result + (code[0] - 2)*weight;
    for (loop = 1; loop < STAGES; ++loop) {
        weight = weight / ldexp(1,bits[loop]-1);
        result = result + \
            (adjust(loop,code,bits[loop])-ldexp(1,bits[loop]-2))*weight;
    }

    return result;
}

int adjust(int stage, int *code, int bits)
{
    int levels,shift,leap,current;

    levels = ldexp(1,bits+1);
    shift = ldexp(1,bits-2);
    leap = ldexp(1,bits-1);
    current = code[stage];
    if ((code[stage-1] % 4) == 0) current = (current - shift + levels) % levels;
    if ((code[stage-1] % 4) == 1) current = (current - shift - leap + levels) % levels;
    if ((code[stage-1] % 4) == 2) current = (current - shift - 2*leap + levels) % levels;
    if ((code[stage-1] % 4) == 3) current = (current - shift - 3*leap + levels) % levels;

    return current;
}

void captaps(int stage, int bits, double *res, double *taps, \
        double caps1[STAGES][MAX_LEVELS*MAX_LEVELS/4])
{
    int loop,point;
    int intfactor,intfactorsq,c1loop,c2loop;
    double a[MAX_LEVELS*MAX_LEVELS/4];
    double b[MAX_LEVELS*MAX_LEVELS/4];
    double c1,c2;

    intfactor = ldexp(1,bits-1);
    intfactorsq = intfactor*intfactor;
    for (loop = 0; loop < 4; ++loop) {
        for (point = 0; point < intfactor; ++point) {
            c1 = 0;
            c2 = 0;
            for (c1loop = 0; c1loop < point; ++c1loop)
                c1 = c1 + caps1[stage][loop*intfactorsq+point*intfactor+c1loop];
            for (c2loop = c1loop; c2loop < intfactor; ++c2loop)
                c2 = c2 + caps1[stage][loop*intfactorsq+point*intfactor+c2loop];
            taps[loop*intfactor+point] = (c2*res[loop]+c1*res[(loop+1)%4])/(c1+c2);
        }
    }
}

void resistaps(double *taps, double *ladder, double vin)
{
    int loop;
    double ladtot,cumsum;
    double rtaps[MAX_LEVELS];

    ladtot = 0;
    cumsum = 0;
    for (loop = 0; loop < 16; ++loop)
        ladtot = ladtot + ladder[loop];
    for (loop = 0; loop < 16; ++loop) {
        cumsum = cumsum + ladder[loop];
        rtaps[loop] = cumsum/ladtot*(REFT-REFB)+REFB;
    }
    taps[0] = vin+rtaps[1]-rtaps[13];
    taps[1] = vin+rtaps[2]-rtaps[12];
    taps[2] = vin+rtaps[3]-rtaps[11];
    taps[3] = vin+rtaps[4]-rtaps[10];
    taps[4] = vin+rtaps[5]-rtaps[9];
    taps[5] = vin+rtaps[6]-rtaps[8];
    taps[6] = vin+rtaps[7]-rtaps[7];
    taps[7] = vin+rtaps[8]-rtaps[6];
    taps[8] = vin+rtaps[9]-rtaps[5];
    taps[9] = vin+rtaps[10]-rtaps[4];
    taps[10] = vin+rtaps[11]-rtaps[3];
    taps[11] = vin+rtaps[12]-rtaps[2];
    taps[12] = vin+rtaps[13]-rtaps[1];
    taps[13] = 1e6;
    taps[14] = 1e6;
    taps[15] = 1e6;
}

void interpol(int code, int stage, double *res, double caps2[STAGES][MAX_LEVELS])
{
    double res0,res1,res2,res3;

    switch (code) {
        case 0 :
            res1 = res[0];
            res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[0] + \
                caps2[stage][11]*res[1])/(caps2[stage][8]+caps2[stage][9] + \
                caps2[stage][10]+caps2[stage][11]);
            res3 = ((caps2[stage][12]+caps2[stage][13])*res[0] + \
                (caps2[stage][14]+caps2[stage][15])*res[1]) / \
                (caps2[stage][12]+caps2[stage][13] + \
                caps2[stage][14]+caps2[stage][15]);
            res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[0] + \
                caps2[stage][3]*res[3])/(caps2[stage][0]+caps2[stage][1] + \
                caps2[stage][2]+caps2[stage][3]);
            break;
        case 1 :
            res1 = res[0];
            res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[0] + \
                caps2[stage][11]*res[1])/(caps2[stage][8]+caps2[stage][9] + \
                caps2[stage][10]+caps2[stage][11]);
            res3 = ((caps2[stage][12]+caps2[stage][13])*res[0] + \
                (caps2[stage][14]+caps2[stage][15])*res[1]) / \
                (caps2[stage][12]+caps2[stage][13] + \
                caps2[stage][14]+caps2[stage][15]);
            res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[1] + \
                caps2[stage][3]*res[0])/(caps2[stage][0]+caps2[stage][1] + \
                caps2[stage][2]+caps2[stage][3]);
            break;
        case 2 :
            res1 = res[1];
            res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[0] + \
                caps2[stage][11]*res[1])/(caps2[stage][8]+caps2[stage][9] + \
                caps2[stage][10]+caps2[stage][11]);
            res3 = ((caps2[stage][12]+caps2[stage][13])*res[0] + \
                (caps2[stage][14]+caps2[stage][15])*res[1]) / \
                (caps2[stage][12]+caps2[stage][13] + \
                caps2[stage][14]+caps2[stage][15]);
            res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[1] + \
                caps2[stage][3]*res[0])/(caps2[stage][0]+caps2[stage][1] + \
                caps2[stage][2]+caps2[stage][3]);
            break;
        case 3 :
            res1 = res[1];
            res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[1] + \
                caps2[stage][11]*res[2])/(caps2[stage][8]+caps2[stage][9] + \
                caps2[stage][10]+caps2[stage][11]);
            res3 = ((caps2[stage][12]+caps2[stage][13])*res[0] + \
                (caps2[stage][14]+caps2[stage][15])*res[1]) / \
                (caps2[stage][12]+caps2[stage][13] + \
                caps2[stage][14]+caps2[stage][15]);
            res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[1] + \
                caps2[stage][3]*res[0])/(caps2[stage][0]+caps2[stage][1] + \
                caps2[stage][2]+caps2[stage][3]);
        break;
    case 4 :
        res1 = res[1];
        res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[1] + \
            caps2[stage][11]*res[2])/(caps2[stage][8]+caps2[stage][9] + \
            caps2[stage][10]+caps2[stage][11]);
        res3 = ((caps2[stage][12]+caps2[stage][13])*res[1] + \
            (caps2[stage][14]+caps2[stage][15])*res[2]) / \
            (caps2[stage][12]+caps2[stage][13] + \
            caps2[stage][14]+caps2[stage][15]);
        res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[1] + \
            caps2[stage][3]*res[0])/(caps2[stage][0]+caps2[stage][1] + \
            caps2[stage][2]+caps2[stage][3]);
        break;
    case 5 :
        res1 = res[1];
        res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[1] + \
            caps2[stage][11]*res[2])/(caps2[stage][8]+caps2[stage][9] + \
            caps2[stage][10]+caps2[stage][11]);
        res3 = ((caps2[stage][12]+caps2[stage][13])*res[1] + \
            (caps2[stage][14]+caps2[stage][15])*res[2]) / \
            (caps2[stage][12]+caps2[stage][13] + \
            caps2[stage][14]+caps2[stage][15]);
        res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[2] + \
            caps2[stage][3]*res[1])/(caps2[stage][0]+caps2[stage][1] + \
            caps2[stage][2]+caps2[stage][3]);
        break;
    case 6 :
        res1 = res[2];
        res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[1] + \
            caps2[stage][11]*res[2])/(caps2[stage][8]+caps2[stage][9] + \
            caps2[stage][10]+caps2[stage][11]);
        res3 = ((caps2[stage][12]+caps2[stage][13])*res[1] + \
            (caps2[stage][14]+caps2[stage][15])*res[2]) / \
            (caps2[stage][12]+caps2[stage][13] + \
            caps2[stage][14]+caps2[stage][15]);
        res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[2] + \
            caps2[stage][3]*res[1])/(caps2[stage][0]+caps2[stage][1] + \
            caps2[stage][2]+caps2[stage][3]);
        break;
    case 7 :
        res1 = res[2];
        res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[2] + \
            caps2[stage][11]*res[3])/(caps2[stage][8]+caps2[stage][9] + \
            caps2[stage][10]+caps2[stage][11]);
        res3 = ((caps2[stage][12]+caps2[stage][13])*res[1] + \
            (caps2[stage][14]+caps2[stage][15])*res[2]) / \
            (caps2[stage][12]+caps2[stage][13] + \
            caps2[stage][14]+caps2[stage][15]);
        res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[2] + \
            caps2[stage][3]*res[1])/(caps2[stage][0]+caps2[stage][1] + \
            caps2[stage][2]+caps2[stage][3]);
        break;
    case 8 :
        res1 = res[2];
        res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[2] + \
            caps2[stage][11]*res[3])/(caps2[stage][8]+caps2[stage][9] + \
            caps2[stage][10]+caps2[stage][11]);
        res3 = ((caps2[stage][12]+caps2[stage][13])*res[2] + \
            (caps2[stage][14]+caps2[stage][15])*res[3]) / \
            (caps2[stage][12]+caps2[stage][13] + \
            caps2[stage][14]+caps2[stage][15]);
        res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[2] + \
            caps2[stage][3]*res[1])/(caps2[stage][0]+caps2[stage][1] + \
            caps2[stage][2]+caps2[stage][3]);
        break;
    case 9 :
        res1 = res[2];
        res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[2] + \
            caps2[stage][11]*res[3])/(caps2[stage][8]+caps2[stage][9] + \
            caps2[stage][10]+caps2[stage][11]);
        res3 = ((caps2[stage][12]+caps2[stage][13])*res[2] + \
            (caps2[stage][14]+caps2[stage][15])*res[3]) / \
            (caps2[stage][12]+caps2[stage][13] + \
            caps2[stage][14]+caps2[stage][15]);
        res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[3] + \
            caps2[stage][3]*res[2])/(caps2[stage][0]+caps2[stage][1] + \
            caps2[stage][2]+caps2[stage][3]);
        break;
    case 10 :
        res1 = res[3];
        res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[2] + \
            caps2[stage][11]*res[3])/(caps2[stage][8]+caps2[stage][9] + \
            caps2[stage][10]+caps2[stage][11]);
        res3 = ((caps2[stage][12]+caps2[stage][13])*res[2] + \
            (caps2[stage][14]+caps2[stage][15])*res[3]) / \
            (caps2[stage][12]+caps2[stage][13] + \
            caps2[stage][14]+caps2[stage][15]);
        res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[3] + \
            caps2[stage][3]*res[2])/(caps2[stage][0]+caps2[stage][1] + \
            caps2[stage][2]+caps2[stage][3]);
        break;
    case 11 :
        res1 = res[3];
        res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[3] + \
            caps2[stage][11]*res[0])/(caps2[stage][8]+caps2[stage][9] + \
            caps2[stage][10]+caps2[stage][11]);
        res3 = ((caps2[stage][12]+caps2[stage][13])*res[2] + \
            (caps2[stage][14]+caps2[stage][15])*res[3]) / \
            (caps2[stage][12]+caps2[stage][13] + \
            caps2[stage][14]+caps2[stage][15]);
        res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[3] + \
            caps2[stage][3]*res[2])/(caps2[stage][0]+caps2[stage][1] + \
            caps2[stage][2]+caps2[stage][3]);
        break;
    case 12 :
        res1 = res[3];
        res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[3] + \
            caps2[stage][11]*res[0])/(caps2[stage][8]+caps2[stage][9] + \
            caps2[stage][10]+caps2[stage][11]);
        res3 = ((caps2[stage][12]+caps2[stage][13])*res[3] + \
            (caps2[stage][14]+caps2[stage][15])*res[0]) / \
            (caps2[stage][12]+caps2[stage][13] + \
            caps2[stage][14]+caps2[stage][15]);
        res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[3] + \
            caps2[stage][3]*res[2])/(caps2[stage][0]+caps2[stage][1] + \
            caps2[stage][2]+caps2[stage][3]);
        break;
        case 13 :
            res1 = res[3];
            res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[3] + \
                caps2[stage][11]*res[0])/(caps2[stage][8]+caps2[stage][9] + \
                caps2[stage][10]+caps2[stage][11]);
            res3 = ((caps2[stage][12]+caps2[stage][13])*res[3] + \
                (caps2[stage][14]+caps2[stage][15])*res[0]) / \
                (caps2[stage][12]+caps2[stage][13] + \
                caps2[stage][14]+caps2[stage][15]);
            res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[0] + \
                caps2[stage][3]*res[3])/(caps2[stage][0]+caps2[stage][1] + \
                caps2[stage][2]+caps2[stage][3]);
            break;
        case 14 :
            res1 = res[0];
            res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[3] + \
                caps2[stage][11]*res[0])/(caps2[stage][8]+caps2[stage][9] + \
                caps2[stage][10]+caps2[stage][11]);
            res3 = ((caps2[stage][12]+caps2[stage][13])*res[3] + \
                (caps2[stage][14]+caps2[stage][15])*res[0]) / \
                (caps2[stage][12]+caps2[stage][13] + \
                caps2[stage][14]+caps2[stage][15]);
            res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[0] + \
                caps2[stage][3]*res[3])/(caps2[stage][0]+caps2[stage][1] + \
                caps2[stage][2]+caps2[stage][3]);
            break;
        case 15 :
            res1 = res[0];
            res2 = ((caps2[stage][8]+caps2[stage][9]+caps2[stage][10])*res[0] + \
                caps2[stage][11]*res[1])/(caps2[stage][8]+caps2[stage][9] + \
                caps2[stage][10]+caps2[stage][11]);
            res3 = ((caps2[stage][12]+caps2[stage][13])*res[3] + \
                (caps2[stage][14]+caps2[stage][15])*res[0]) / \
                (caps2[stage][12]+caps2[stage][13] + \
                caps2[stage][14]+caps2[stage][15]);
            res0 = ((caps2[stage][0]+caps2[stage][1]+caps2[stage][2])*res[0] + \
                caps2[stage][3]*res[3])/(caps2[stage][0]+caps2[stage][1] + \
                caps2[stage][2]+caps2[stage][3]);
            break;
        default : printf("\nERROR: code out of range.\n");
    }
    res[0] = res0;
    res[1] = res1;
    res[2] = res2;
    res[3] = res3;
}

int flash(int bits, int stage, double compoff[STAGES][MAX_LEVELS], double *taps)
{
    int code,loop;

    code = -1;
    for (loop = 0; loop < ldexp(1,bits+1)-1; ++loop) {
        if ((taps[loop] <= compoff[stage][loop]) && \
            (taps[loop+1] > compoff[stage][loop+1]) && (code == -1))
            code = loop;
    }
    if (code == -1)
        code = loop;
    return code;
}

double amp1(double ain, double gain, double offset)
{
    double result;

    result = (ain + offset) * gain;

    return result;
}

double amp2(double ain, double gain, double offset)
{
    double result;

    result = (ain + offset) * gain;

    return result;
}


void initialize(int *bits, double *bins, double amp1off[STAGES][4], \
        double amp2off[STAGES][4], \
        double compoff[STAGES][MAX_LEVELS], \
        double gain1[STAGES][4], \
        double gain2[STAGES][4], \
        double ladder[MAX_LEVELS], \
        double caps1[STAGES][MAX_LEVELS*MAX_LEVELS/4], \
        double caps2[STAGES][MAX_LEVELS])
{
    int loop,arch,index,levels,inner,idum;
    time_t now;

    time(&now);
    arch = ARCH;
    index = 0;
    idum = -now;
    gasdev(&idum);
    for (loop = STAGES; loop > 0; --loop) {
        bits[index] = arch/pow(10,loop - 1);
        arch -= bits[index] * pow(10,loop - 1);
        ++index;
    }
    for (loop = 0; loop < STAGES; ++loop) {
        levels = ldexp(1,bits[loop]+1);
        for (inner = 0; inner < 4; ++inner) {
            gain1[loop][inner] = 4 * normal(1,SIG_GAIN);
            gain2[loop][inner] = normal(1,SIG_GAIN);
            amp1off[loop][inner] = normal(0,SIG_AMPOFF)/gain1[loop][inner];
            amp2off[loop][inner] = normal(0,SIG_AMPOFF)/gain2[loop][inner];
        }
        for (inner = 0; inner < levels; ++inner)
            compoff[loop][inner] = normal(0,SIG_COMPOFF);
        for (inner = 0; inner < levels * levels / 4; ++inner)
            caps1[loop][inner] = normal(1,SIG_CAP);
        for (inner = 0; inner < MAX_LEVELS; ++inner)
            caps2[loop][inner] = normal(1,SIG_CAP);
    }
    for (loop = 0; loop < 1024; ++loop)
        bins[loop] = 0;
    for (loop = 0; loop < ldexp(1,bits[0]+1); ++loop)
        ladder[loop] = normal(1,SIG_LAD);
}

double normal(double mean, double sigma)
{
    int idum;
    double result;

    result = (gasdev(&idum) * sigma) + mean;

    return result;
}

float gasdev(int *idum)
{
    static int iset = 0;
    static float gset;
    float fac,r,v1,v2;

    if (iset == 0) {
        do {
            v1 = 2.0*ran1(idum)-1.0;
            v2 = 2.0*ran1(idum)-1.0;
            r = v1*v1+v2*v2;
        } while (r >= 1.0);
        fac = sqrt(-2.0*log(r)/r);
        gset = v1*fac;
        iset = 1;
        return v2*fac;
    } else {
        iset = 0;
        return gset;
    }
}

float ran1(int *idum)
{
    static long ix1,ix2,ix3;
    static float r[98];
    float temp;
    static int iff=0;
    int j;

    if (*idum < 0 || iff == 0) {
        iff = 1;
        ix1 = (IC1 - (*idum)) % M1;
        ix1 = (IA1*ix1+IC1) % M1;
        ix2 = ix1 % M2;
        ix1 = (IA1*ix1+IC1) % M1;
        ix3 = ix1 % M3;
        for (j = 1; j <= 97; j++) {
            ix1 = (IA1 * ix1 + IC1) % M1;
            ix2 = (IA2 * ix2 + IC2) % M2;
            r[j] = (ix1 +  ix2*RM2) * RM1;
        }
        *idum = 1;
    }
    ix1 = (IA1*ix1 + IC1) % M1;
    ix2 = (IA2*ix2 + IC2) % M2;
    ix3 = (IA3*ix3 + IC3) % M3;
    j = 1 + ((97*ix3) / M3);
    if (j > 97 || j < 1) printf("RAN1: This cannot happen.\n");
    temp = r[j];
    r[j] = (ix1+ix2*RM2)*RM1;
    return temp;
}

void bin2dnl(double *bins)
{
    int loop;

    for (loop = 0; loop < 1024; ++loop)
        bins[loop] = bins[loop]/100 - 1.0;
}

void simout(double *bins)
{
    FILE *fp1, *fp2;
    int loop;
    double cumsum;

    fp1 = fopen("dnl.dat","w");
    fp2 = fopen("inl.dat","w");


    cumsum = 0;
    for (loop = 0; loop < 1024; ++loop) {
        if ((loop < 15) || (loop > 1007))
            bins[loop] = 0;
        cumsum += bins[loop];
        fprintf(fp1,"%f\n",bins[loop]);
        fprintf(fp2,"%f\n",cumsum);
    }

    fclose(fp1);
    fclose(fp2);
}
```

References

[1] D. H. Sheingold, *Analog-Digital Conversion Handbook*, Prentice-Hall 1986.

[2] B. Peetz et al., "An 8b 250MHz A/D converter," *International Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 136-137, Feb. 1986.

[3] Y. Gendai et al., "An 8b 500MHz ADC," *International Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 172-173, Feb. 1991.

[4] A. Yukawa, "A CMOS 8-bit high-speed A/D converter IC," *IEEE J. Solid-State Circuits*, vol. SC-20, pp. 775-779, June 1985.

[5] T. Tsukada et al., "CMOS 8b 25MHz flash ADC," *International Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 34-35, Feb. 1985.

[6] T. Kumamoto et al., "An 8-bit high-speed CMOS A/D converter," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 6, Dec. 1986, pp. 976-982.

[7] K. Kusumoto et al., "A 10b 20MHz 30mW pipelined interpolating ADC," *IEEE J. Solid-State Circuit*s, vol. 28, no. 12, pp. 1200-1206, Dec. 1993.

[8] H. Kimura et al., "A 10b 300MHz interpolated-parallel A/D converter," *IEEE J. Solid-State Circuits*, vol. 28, no. 4, pp. 438-446, April 1993.

[9] K. Kattmann and J. Barrow, "A technique for reducing differential non-linearity errors in flash A/D converters," *International Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 1991, pp. 170-171.

[10] B. S. Song et al., "A 12-bit 1-Msample/s capacitor error-averaging pipelined A/D converter," *IEEE J. Solid-State Circuits*, vol. SC-23, pp. 1324-1333, Dec. 1988.

[11] D. Robertson et al., "A wideband 10-bit, 20Msps pipelined ADC using current-mode signals," *International Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 160-161, Feb. 1990.

[12] S. Lewis et al., "A 10-b 20-Msample/s analog-to-digital converter," *IEEE J. Solid-State Circuits*, vol. 27, no. 3, pp. 351-358.

[13] Chris Mangelsdorf, et al., "Design for testability in digitally-corrected ADCs," *International Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 70-71, Feb. 1993.

[14] B. S. Song et al., "A 10b 15MHz CMOS recycling two-step A/D converter," IEEE J. Solid-State Circuits, vol. 25, no. 12, pp. 1328-1338, Dec. 1990.

[15] S. Lewis and P. R. Gray, "A pipelined 5-Msample/s 9-bit analog-to-digital converter," *IEEE J. Solid-State Circuits*, vol. SC-22, no. 6, pp. 954-961, Dec. 1987.

[16] C. W. Moreland, "An 8b 150MSample/s serial ADC," *International Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 272-273, Feb. 1995.

[17] C. Mangelsdorf et al., "A two-residue architecture for multistage ADCs," *International Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 64-65, Feb. 1993.

[18] R. van de Grift and M. van der Veen, "An 8b 50Mhz video ADC with folding and interpolation techniques," *International Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 94-95, Feb. 1987.

[19] A. Matsuzawa et al. "A 10b 30MHz two-step parallel BiCMOS ADC with internal S/H," *International Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 162-163, Feb. 1990.

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

[illegible]