Split Cyclic Analog to Digital Converter Using A Nonlinear Gain Stage

by

Hattie Spetla

A Thesis
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Master of Science
in
Electrical and Computer Engineering
by

____

September 2009

APPROVED:

____

Professor John A. McNeill, Major Advisor

____

Professor Stephen J. Bitar

____

Professor Donald R. Brown

Abstract

Previous implementations of digital background calibration for cyclic ADCs have required linear amplifier behavior in the gain stage for accurate correction. Correction is digital decoding of ADC outputs to determine the original ADC input. Permitting nonlinearity in the gain stage of the ADC allows for less demanding amplifier design requirements, reducing power and size. However this requires a method of determining the value of this variable gain during digital correction. Look up tables (LUTs,) are an effective and efficient method of compensating for analog circuit imperfections. The LUT correction and calibration method discussed in this work has been simulated using Cadence integrated circuit simulation ADC specifications and MATLAB.

Acknowledgements

I would first like to thank Professor John McNeill, who with clear explanations and impeccable blackboards has been a source inspiration since my undergraduate days. I have learned an immense amount while in NECAMSID and am fortunate to have him as an advisor. His guidance has both led me in the right direction while still requiring me to find it myself, a careful balance.

Thank you also to my Committee members Professor Stephen J. Bitar and Professor Donald R. Brown for their input and insight.

Next I would like to thank Texas Instruments for providing funding for my research. This has been an amazing opportunity.

Thanks is also due to my colleagues Chris David, Chilann Ka Yan Chan, Cody Brenneman, Tsai Chen, Sam Beam, Ed Oliveira, Michael Irace and Michael Leferman. The support, criticisms, and occasional snacks they provided have been invaluable.

Lastly, a special thank you to my mother Janis for the countless ways she has been there for me, especially for the endless encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Analog to digital converters, (ADCs) are an essential part of many electrical systems. Today, the trend is for circuits to decrease in size and power consumption. This allows for increased use in mobile and other low power applications.

Open loop gain stages in nanoscale CMOS are of interest as they allow for much smaller circuits, reducing power consumption and increasing speed. Open loop gain stages are less dependent on analog precision. Accuracy is one of the most demanding aspects of analog design[4]. Digital circuits can be used to compensate for more relaxed design requirements in analog circuit designs. The advantage to this is that digital circuity is less 'expensive,' in terms of both power and size, than analog portions of the system[1]. Several recent works have shown an interest in allowing nonlinearity into amplifier portions of ADCs [2] [4] [13] [14]. Because the requirements on design of the analog portion goes down, the size can decrease and the analog portion can be made more easily reducing cost.

This work builds on previous work with split cyclic ADCs. By introducing nonlinearity into the gain stage of the residue amplifier, analog design requirements are further reduced. Previous cyclic ADC techniques have depended on a linear gain stage for accurate correction and calibration[6].

This project has worked concurrently with an IC design group. The design group has

IC Specifications	
Maximum Size	1 mm2
Process Type	0.18 μm
Resolution	12 Bits
Throughput	1 Msps
Test Time	Less than 1 sec
Other Specifications	Fully Differential

Table 1.1: IC Specifications

simulated and is in the process of layout for a cyclic IC. The research from this project helped shape the design of the amplifier and rest of the circuit. The research on the operation of the circuit has also been instrumental in developing the correction and calibration information [15]. The IC development was sponsored by the New England Center for Analog and Mixed Signal Design(NECAMSID).

In addition to correction, calibration of an ADC with a nonlinear gain stage was attempted using the ‘Split’ technique. Split ADC architecture has been shown to be an effective method for calibrating cyclic ADCs[6]. The split architecture has many advantages, one such advantage is continuous calibration that adapts to environmental changes in the behavior of the ADC including temperature. A second advantage to split calibration is it is a background calibration technique, meaning it’s performed without altering the input of the ADC. It also requires a low number of conversions for calibration [9].

1.2 Terminology

The term correction, when referring to the ADC used in this work, is the process of determining the output code of the ADC. The correction process takes the outputs of the comparators, which will be discussed further in Section 2, and creates a final ADC output code which reflects the original ADC input. Calibration refers to the process of improving the accuracy of the ‘corrected’ ADC output.

1.3 Overview

This thesis discusses correction and digital background calibration of a split cyclic ADC. Previous cyclic ADC implementations have depended on a linear gain stage for effective calibration and correction. This project however allows for further imperfection in the analog portion of the circuit, in the form of amplifier nonlinearity, and can correct and calibrate even with non-linearity in the gain stage.

To accomplish correction and digital background calibration, a look up table (LUT), is used. The LUT stores points which can then be interpolated to determine the behavior of the amplifier, which is used to determine the ADC output in ADC correction. Calibration of this LUT based stored amplifier behavior was attempted using a split cyclic architecture. Split architecture involves comparing the outputs of two ADCs to determine errors in the outputs. Ideally, the outputs would only agree when correct, otherwise errors in the stored amplifier behavior model would lead them to create different output codes.

1.4 Chapter Organization

Chapter 2, is a general background discussion of cyclic analog to digital converters, the ADC used in this method of correction and calibration. Previous work with cyclic analog to digital converters is covered. This section focuses especially on the introduction of nonlinearity into the gain stage of cyclic ADC. There is an explanation of nonlinearity in differential amplifiers, and suggestions, including advantages and disadvantages, for methods of modeling this behavior.

The next section will cover the look up table (LUT) approach to correction in the ADC. It discusses the output correction algorithm and explanation of residue modes. It contains results from simulation of a LUT approach to correction and compares it to the previous work.

The third chapter is a discussion of calibration of cyclic ADCs. Initially, calibration was performed with a known error. After this was accomplished, the split ADC implementation was used to calibrate the LUTs. The fourth section covers the the split architecture. It discusses the complexity involved in implementing calibration in a LUT based cyclic ADC.

This is followed by a discussion of conclusions and proposed future work. Future work includes suggestions for the improvement of the calibration, and implementation.

Chapter 2

Cyclic Analog to Digital Converters

2.1 Introduction

Cyclic ADCs, also known as Algorithmic converters, are Nyquist rate ADCs. They operate at near the Nyquist frequency, as opposed to oversampling converters. As such the bandwidth can be approximated as the sampling frequency, f_n over two [3].

$$Bandwidth = \frac{f_n}{2} \tag{2.1}$$

A basic overview of cyclic ADCs is given in Section 2.2.

One of the advantages of this type of ADC is that calibration depends primarily on the gain stage of the circuit. Although previous work has shown that calibration of the cyclic ADC is possible, that work did not correct for nonlinearity in the gain stage. The advantages of allowing nonlinearity in the gain stage will be discussed further in Section 2.4. Modeling and correction of the ADC increases considerably in complexity, however, as soon as nonlinearity is introduced. Two approaches discussed in this paper are covered in Section 2.5.1, a polynomial approach, and Section 2.5.2, a LUT approach.

2.2 Conceptual Overview

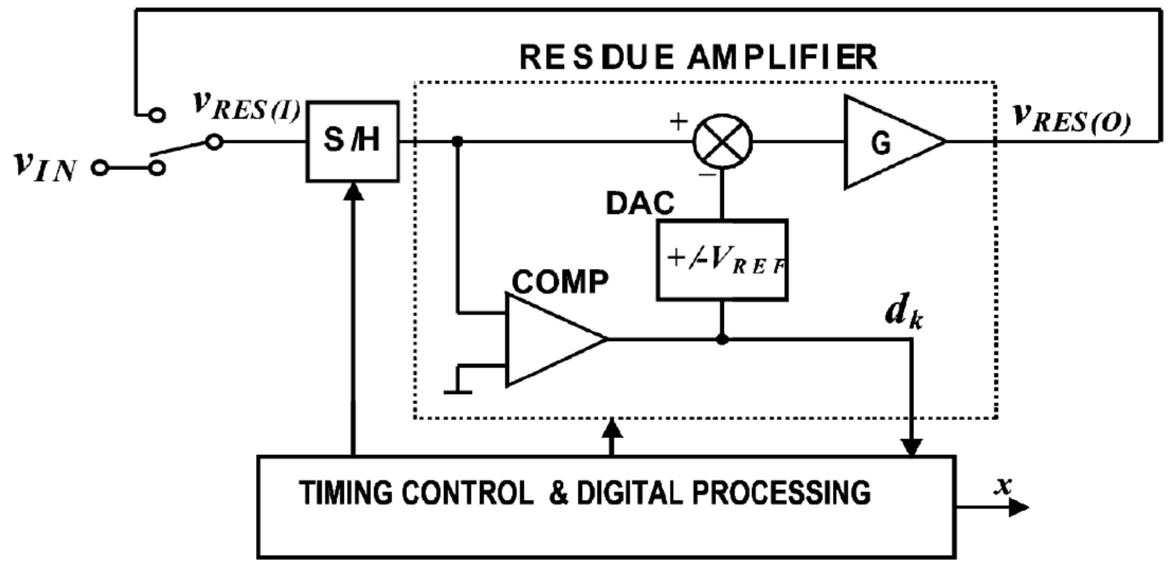

Figure 2.1: Cyclic Converter Block Diagram [6]

Figure 2.1 is a block diagram illustrating the operation of a cyclic ADC. Initially, as is shown in Figure 2.1, the input switch is set to an input voltage, V_{IN}. This V_{IN} is sampled at the sample and hold (S/H) block. Based on V_{IN}, the comparators generate a digital decision output d. A digital to analog converter(DAC), adds or subtracts a reference voltage from the input based on the comparator decision. Then this new value V_{RES} is amplified by a gain G. The input switch is moved to V_{RES} sending it to the S/H for the next cycle. This process is then repeated a number of cycles, until the desired resolution is obtained.

The cyclic ADC acts like a negative feedback loop. Because the DAC subtracts an appropriate amount from each of the V_{RES} values, the gain must be kept less than two. If the gain is increased to more than two, the V_{RES} would eventually increase out of the range of the ADC.

Figure 2.2 is a simplified graphical representation of the behavior of a 4 cycle cyclic converter. This ADC has a gain of close to 2 and an input range of around +/-1V for V_{REF}. The first cycle takes the 0.62V V_{IN}. The comparators make a decision of +1. This leads the DAC to subtract 0.5V, which is the comparator decision multiplied by $\frac{V_{REF}}{2}$. This sets V now to be 0.12V. Then this value is multiplied by the gain of near 2. This gives a $V_{RES(OUT)}$ gain graphically approximated to be 0.3V.

The second stage takes the 0.3V as the new S/H input. The comparator makes another comparator decision of +1. The DAC again subtracts 0.5V, creating a V_{RES} of near -0.2V. This is then multiplied by the gain stage and results in a $V_{RES(OUT)}$ of -0.46V. The next cycle takes this -0.46V and again makes a comparator decision, this time of -1, subtracts -0.5V using the DAC, and outputs a new V_{OUT} of 0.1V. With an input of $V_{IN} = 0.62$ the ADC output would be $1, 1, -1, 1$.

The cyclic ADC in this simulation has 5 possible output decisions. The decisions used are case dependent on the residue mode used, which can be varied, as discussed in Section 2.3.

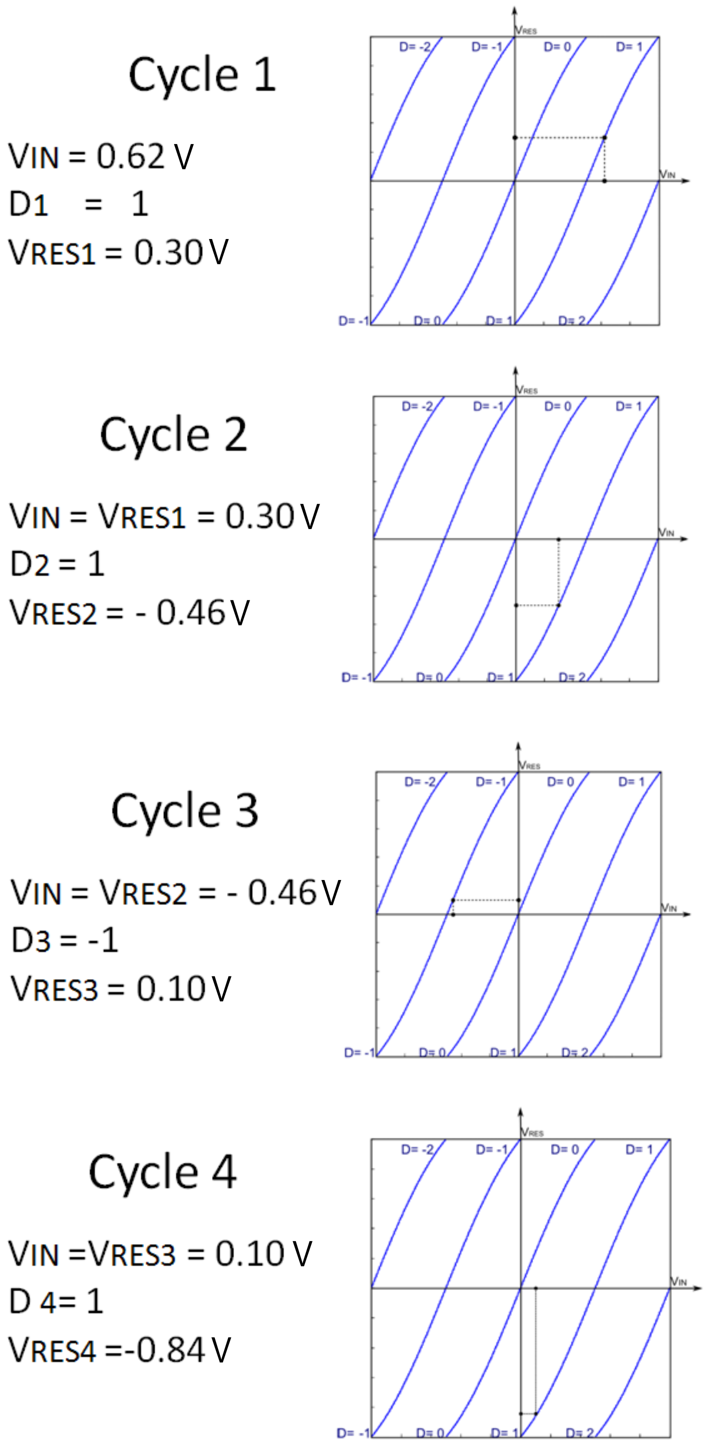

Figure 2.2: Graphic Representation of ADC Behavior

2.3 ADC Simulation

The ADC was simulated in MATLAB for analysis. Equation 2.2 is the basic formula for calculating V_{RESOUT}.

$$V_{RESOUT} = G(V_{RESIN} - D(\frac{V_{REF}}{2})) \tag{2.2}$$

A gain value G is multiplied by the value of $V_{RESIN} - D(\frac{V_{REF}}{2})$. For simulation G was stored as a 100,000 point vector, with a 15 significant figure accuracy, and spline interpolation was used to calculate gain for voltage points in between the points on the vector.

With such a high number of points for the curve, error due to interpolation is minimized. This vector was produced using Cadence simulation of an amplifier portion of a cyclic ADC currently under design. The amplifier was designed to have a maximum gain of 1.9. This G was also stored as 5 separate gain curves to further improve the ability of the simulation to replicate the behavior of the real world model, which might have different gains based on different input ranges. This gain could be edited curve by curve.

The ADC under design has a V_{REF}, and input range, of $+/-$ 0.68V. Anything over this will swing outside the range of the amplifier. Inputs to the simulation were set as a vector of input voltages, as would be seen at the S/H portion of the circuit. The advantage of using a vector is increased speed when processing in MATLAB. This was beneficial in that it allowed the cycling of a large number of voltages vectors at once for correction and calibration.

The V_{RESIN} vector was then used to create 2 new vectors. These vectors were a decision vector, d and a V_{RESOUT} vector.

For the first cycle V_{RESIN} is the input voltage. For subsequent cycles V_{RESIN} is the V_{RESOUT} from the previous cycle, which was continuously stored for analysis.

The minimum gain calculated in the amplifier curve used was around 1.6. This is a 15 percent deviation from the maximum and linear gain, which from simulation is actually 1.9093, very close to the goal of 1.9. With a gain of 1.6 the resolution achieved per ADC conversion set is lower than 2^N, N being the number of cycles. Because the gain is variable, the bits resolution may be slightly higher than 2 to the lowest gain, but a minimum gain of 1.6 gives a resolution of around 12,000 levels, or near 2^{14}.

$$LSB = \frac{2 * V_{REF}}{G^{20}} \tag{2.3}$$

This gives us a least significant bit (LSB) value of the total range, $1.36V$ over the total number of bits, 1.6^{20}, an LSB equal to $112\mu V$. The ADC resolution is much lower than would likely be actualized, but the high level of nonlinearity gives a good representation of the performance of the correction algorithm in worst case scenarios. This is also better than the goal of a 12 bit from the IC design. Based on a 12 bit system the LSB would be $332\mu V$.

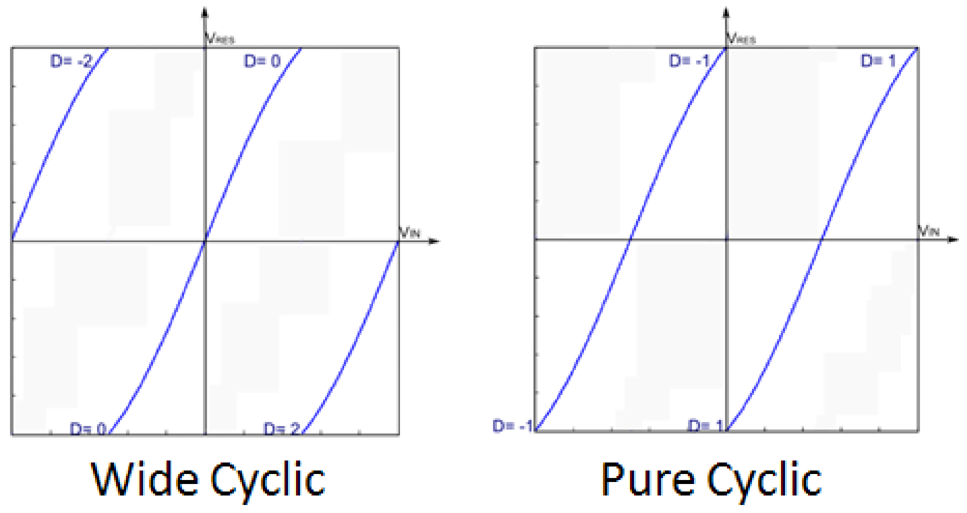

Figure 2.3: Residue Modes (a) Wide Zero (b) Pure Cyclic

The choice of which gain curve to use is controlled in the ADC. There are two possible gain curve sets, referred to as residue modes, and these are shown in Figure 2.3. Wide zero residue mode sets the decisions to -2, 0, and 2. The pure cyclic residue mode uses only the -1 and 1 decisions. In the simulation residue mode could be set to be random by input point, random by cycle, or consistently wide zero or pure cyclic.

2.4 Nonlinearity In Amplifier Circuits

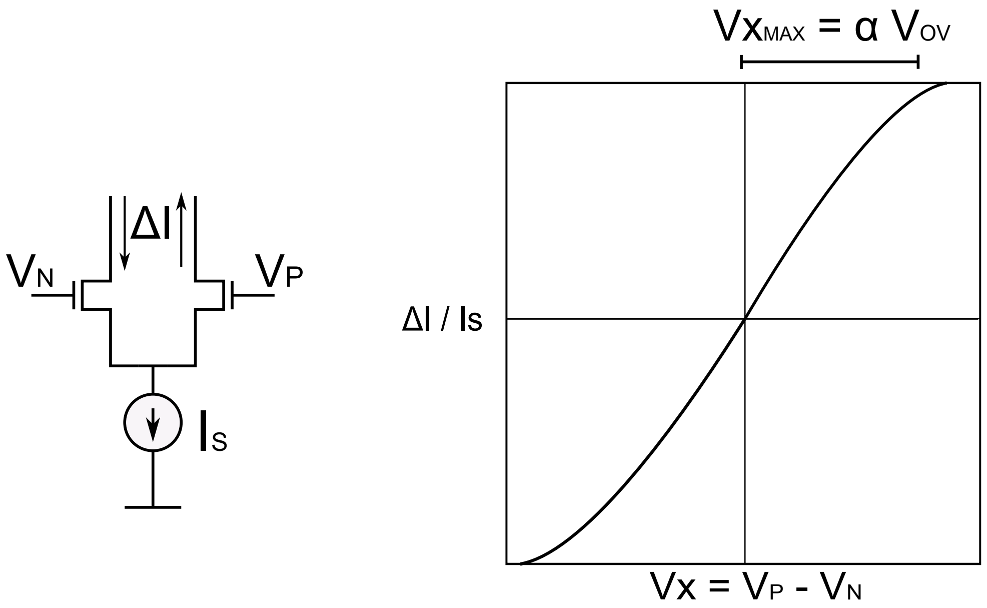

Figure 2.4: Differential Pair V-I Characteristics

Figure 2.4 shows the general model for a differential pair amplifier and its V-I characteristics. Distortion can be modeled as shown in Equation 2.4,

$$\frac{\Delta I}{I_{SS}} = \frac{V_X}{V_{OV}} + \frac{1}{4}\frac{\Delta\beta}{\beta}(\frac{V_x}{V_{OV}})^2 - \frac{1}{8}\frac{\Delta\beta}{\beta}(\frac{V_x}{V_{OV}})^3 ... \tag{2.4}$$

where $\frac{\Delta\beta}{\beta}$ is the mismatch of the two transistors and V_{OV} is the overdrive voltage [11].

$$V_{OV} = V_{GS} - V_{TH} \tag{2.5}$$

The maximum higher order nonlinearity is determined by the input swing of the amplifier which is shown in Equation 2.6

$$\alpha = \frac{V_{xMAX}}{V_{OV}} \tag{2.6}$$

To reduce the nonlinearity, either a large V_{OV} must be chosen, or a small V_{xMAX}, reducing the input range of the amplifier. V_{xMAX} in this case is controlled by the V_{REF} as shown in equation 2.7. This means the main way to increase the linearity of the circuit is to raise the V_{OV} voltage. However this results in a power penalty and may not be possible depending on the IC design process used [11].

$$V_{xMAX} = \frac{V_{REF}}{G} \tag{2.7}$$

Another obvious advantage to open loop amplifiers is the reduction in the number of transistors and the complexity of the circuit. Because there are more components in a closed loop system, there are also more noise sources, decreasing amplifier efficiency. The attainable bandwidth is also increased because the poles in the feedback of a closed loop system become a stability problem. As a larger range is permitted in the operation of the amplifier, headroom for the amplifier is increased as well [11].

Power consumption is inversely related to accuracy in amplifiers. Any reduction that can be made in the amplifier accuracy, without compromising the resolution of the ADC will reduce power consumption of the ADC[14].

2.5 Amplifier Modeling in Digital Correction

Digital correction is the analysis of the digital outputs to determine the original analog input to the system. In a cyclic ADC this is done using knowledge of the decision set, the V_{REF}, and the amplifier behavior.

$$V_{IN} = \frac{V_{OUT}}{G} + D(\frac{V_{REF}}{2}) \tag{2.8}$$

For the linear amplifier, a single value can be stored for gain, a constant G and used for the complete range of input voltage values. A non-linear gain stage however, requires a method for determining the gain to a high precision for a large number of voltages. In simulation it may be feasible to use tens of thousands of points to recreate the behavior of the amplifier. However, in digital implementation this is not practical.

Previous implementations of cyclic ADCs have required linear amplifier behavior for proper correction. Allowing the introduction of nonlinearity in the gain stage of the amplifier allows for less demanding amplifier design requirements, but increases the complexity of

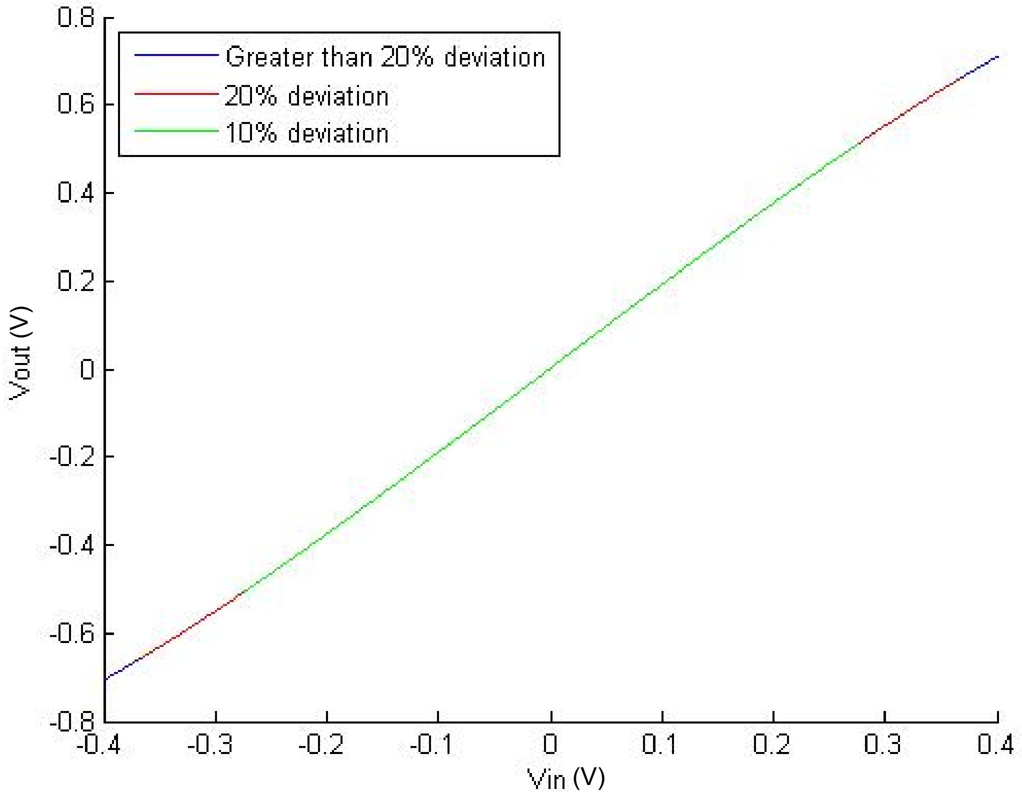

Figure 2.5: Nonlinear Amplifier Behavior

the digital portion of the circuit. Figure 2.5 depicts the Cadence simulation output of the amplifier under design. As can be seen in the figure, for a V_{REF} of $0.68V$, giving an amplifier range of $0.34V$, the deviation from maximum gain is between 10 and 20 percent.

Two approaches for implementing this variable gain value in the correction stage of the ADC were considered. One was the use of a polynomial representing the gain curve. This is complicated, requiring hardware intensive and time consuming calculation, especially during calibration. The second method was the storage of a set number of representative points on a look up table, LUT, and interpolation of these points. The effectiveness of varying LUT sizes and three different methods of interpolation were considered. As will be shown, high accuracy can be obtained with a limited number of LUT points and simple linear interpolation, eliminating the need for both complex calculations and large memory

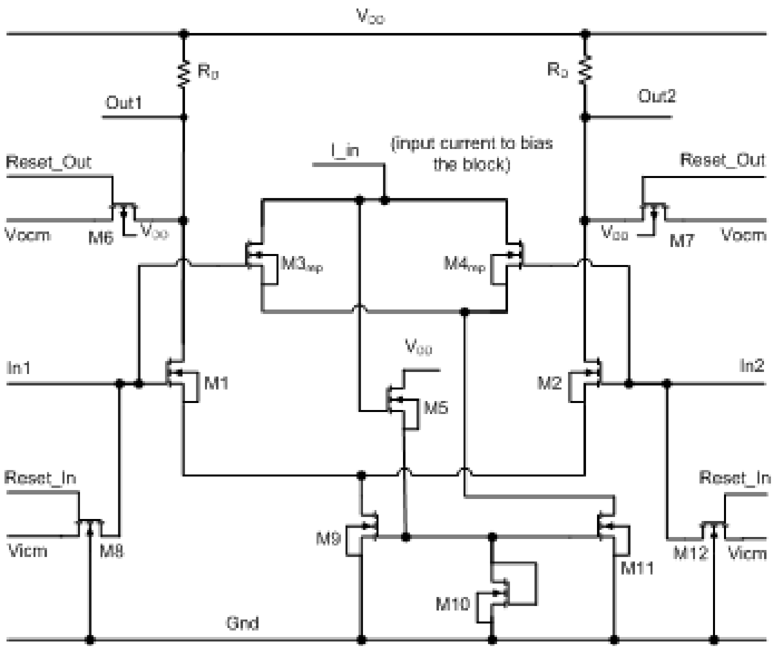

Figure 2.6: Cadence Amplifier Schematic

requirements. The use of a LUT also allows for a simple yet effective calibration of the stored digital data.

2.5.1 Polynomial Approach

One way to approximate the nonlinear behavior is to use a polynomial. Input vectors are plugged into the polynomial, giving an output value.

A third order nonlinearity model is represented in Equation 2.9 [12].

$$V_{RESOUT} = a_1 V_{RESIN} + a_2 V_{RESIN}^2 + a_3 V_{RESIN}^3 \tag{2.9}$$

If the amplifier behavior is irregular it is difficult to use a polynomial to represent it. A LUT would provide more freedom in representing amplifier behavior. It is possible for a large change at one end of the curve to have repercussions at the other one while using a polynomial. Using a polynomial with an order higher than three is impractical [12].

2.5.2 Look Up Table Approach to Correction

The second method considered was the use of a LUT and interpolation. A table of values approximates the behavior of the amplifier as these values take into account both gain and offset error. Interpolation can then be used to determine the amplifier output for any value within the input range. Figure 2.7 and Table 2.1 show the ideal 2^5 point LUT created from the amplifier behavior simulated in Cadence. This amplifier was specifically designed for a cyclic ADC IC layout.

The complete range of the amplifier is not used for all points. Occasionally, as will be discussed in a future section, the V_{RES} value leaves the expected range of the amplifier because of errors introduce by an initial guess. This will only happen at early stages in the correction, because the errors introduced have very little weight in the final output error.

2.6 Polynomial and LUT Results

Figure 2.8 shows error in calculating a VRES value when using a 2^5 point LUT and linear interpolation versus using a third order polynomial. The polynomial was calculated to a high level of accuracy using MATLAB. Plotted is the difference between the 100000 point simulation output, and points calculated using LUT and polynomial. In red are the points calculated using the LUT. Close to the LUT points the accuracy is high, further from the points the accuracy decreases. Accuracy is much lower for the polynomial. At only 2^5 points the accuracy of the LUT is much higher than that of the 3rd degree polynomial.

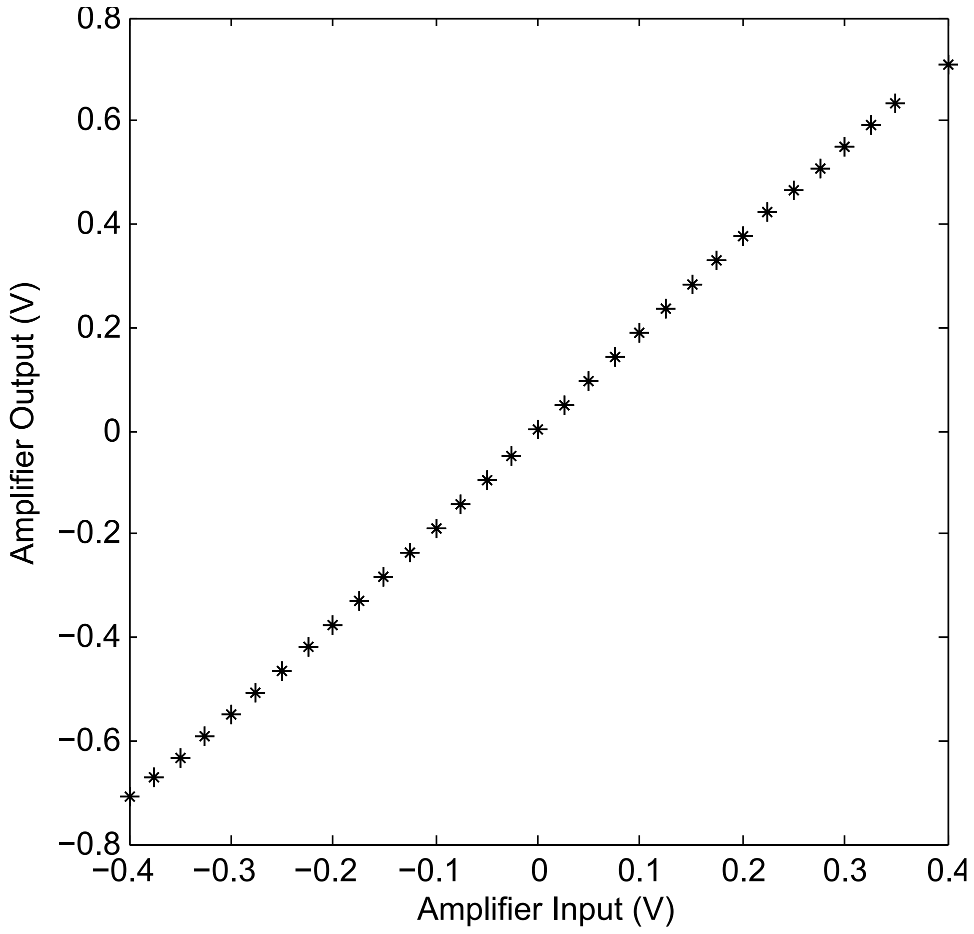

Figure 2.7: 2^5 point LUT

V_{IN}	V_{OUT}
-0.4000	-0.7075
-0.3750	-0.6710
-0.3500	-0.6326
-0.3250	-0.5926
-0.3000	-0.5513
-0.2750	-0.5088
-0.2500	-0.4653
-0.2249	-0.4209
-0.1999	-0.3758
-0.1749	-0.3301
-0.1499	-0.2838
-0.1249	-0.2371
-0.0999	-0.1900
-0.0749	-0.1427
-0.0499	-0.0952
-0.0249	-0.0475
0.0001	0.0002
0.0251	0.0480
0.0501	0.0956
0.0752	0.1432
0.1002	0.1905
0.1252	0.2376
0.1502	0.2843
0.1752	0.3305
0.2002	0.3763
0.2252	0.4214
0.2502	0.4657
0.2752	0.5092
0.3002	0.5517
0.3252	0.5930
0.3502	0.6330
0.4000	0.7075

Table 2.1: 2^5 LUT Points

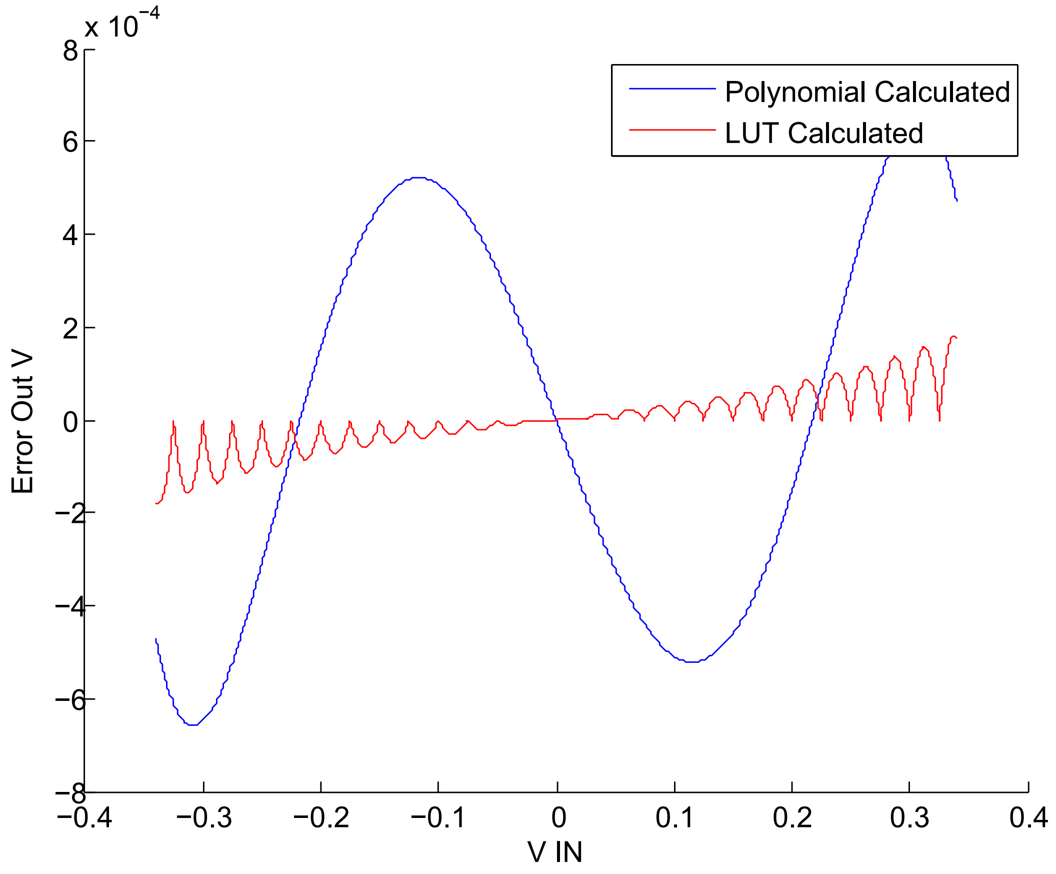

Figure 2.8: Error in LUT Vs Third Order Polynomial

Chapter 3

LUT Approach to Correction

3.1 Correction Algorithm

Correction takes the D outputs of the ADC and uses them to determine the original input. Figure 3.1 depicts the correction that goes on to determine the original voltage input of the ADC. In this case based on an initial guess of 0 it finds the corresponding place on the gain curve for that decision and the corresponding previous V_{RESOUT}. Although the final V_{RESOUT} value is unknown during correction because the weight in the initial phase of correction is minimal, error introduced because of an incorrect guess is insignificant.

The complex gain can be modeled as a function.

$$V_{RESOUT} = f(\frac{V_{RESIN}}{V_{REF}}, D) \tag{3.1}$$

When modeled like this, the operation of the ADC becomes as shown in Equation 3.2

$$x = f(f(f(V_{IN}, D_1), D_2), D_3) \tag{3.2}$$

Correction works by using the inverse of the gain function to calculate the output code.

$$x = f^{-1}(D_1, f^{-1}(D_2, f^{-1}(D_3, V_{RESOUT}))) \tag{3.3}$$

The correction algorithm works in a similar fashion to the ADC simulation. The complexity lies in the gain portion. Interpolation of a LUT is used in place of gain. The LUT

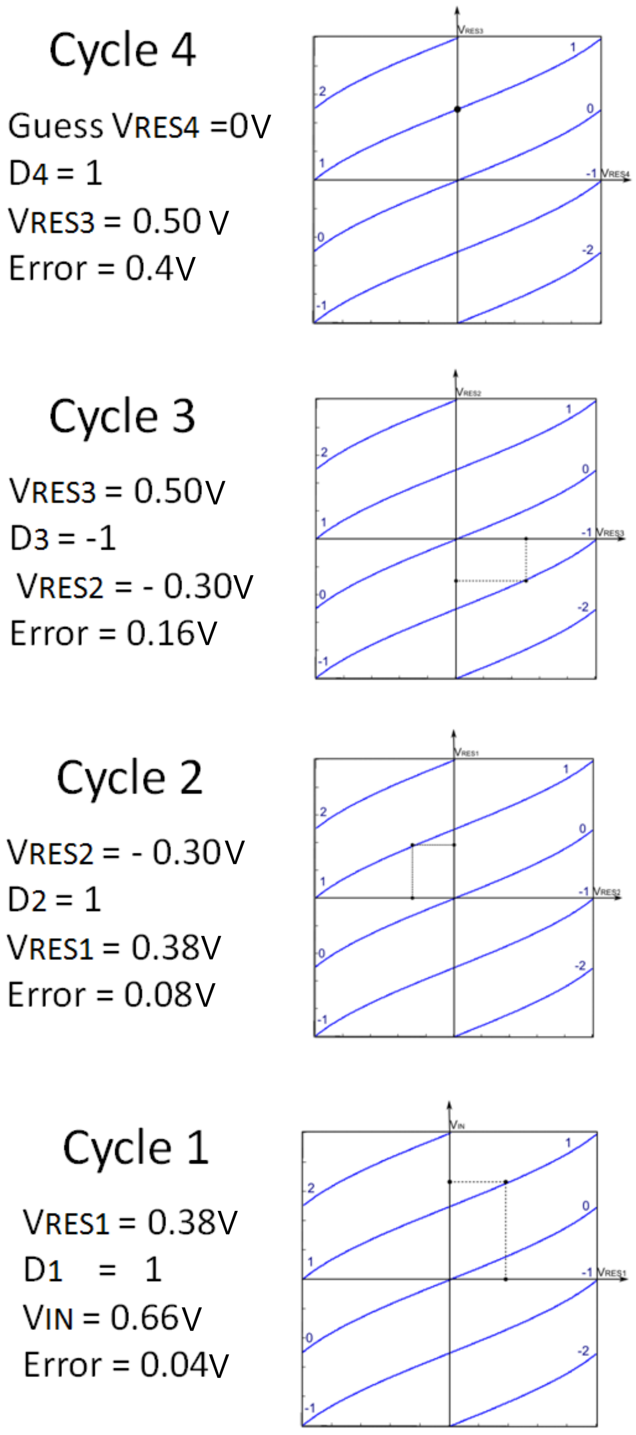

Figure 3.1: Correction of ADC Output

has a limited number of points, the sizing of which is discussed in the next section.

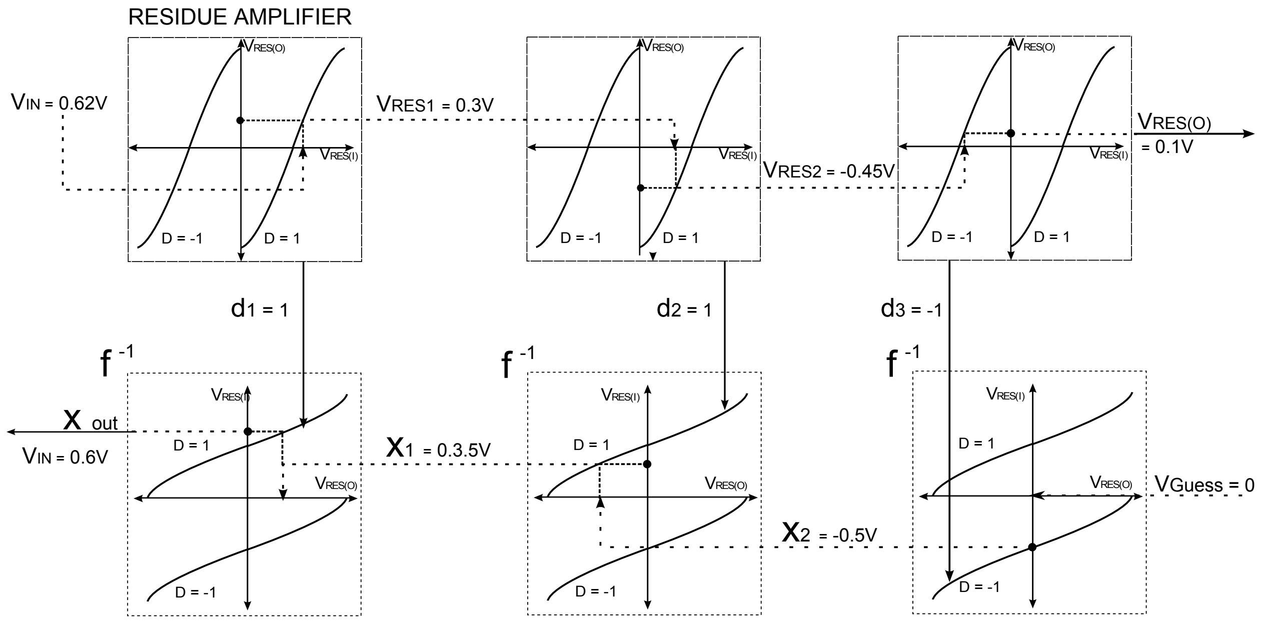

Figure 3.2: ADC Behavior and Correction of ADC Output

3.2 LUT Sizing Considerations

When initially considering the size of the LUT, attention was paid mostly to the difference between LUT sizes and types of interpolation. Analysis was done between interpolated curve values and the vector created in Cadence simulation. As the project progressed, the number of cycles was determined to be the major factor in sizing the LUT.

3.2.1 Choice of Interpolation Method

Two types of interpolation were analyzed, linear and simple polynomial. Equation 3.4 is simple linear interpolation and Equation 3.5 is four point polynomial interpolation[7].

$$Y = Y_1 + (X_1 + X_2)(\frac{Y_2 - Y_1}{X_2 - X_1}) \tag{3.4}$$

$$Y = Y_1 \frac{(X - X_2)(X - X_3)(X - X_4)}{(X_1 - X_2)(X_1 - X_3)(X_1 - X_4)} + ... + Y_4 \frac{(X - X_1)(X - X_2)(X - X_3)}{(X_4 - X_1)(X_4 - X_2)(X_4 - X_3)} \tag{3.5}$$

For both of these methods of interpolation, analysis was run between each, for varying non-linearity and LUT size. This was to give a good idea of the nonlinearity that would be tolerated by the correction algorithm, and be reproduced with minimal distortion using the LUT method.

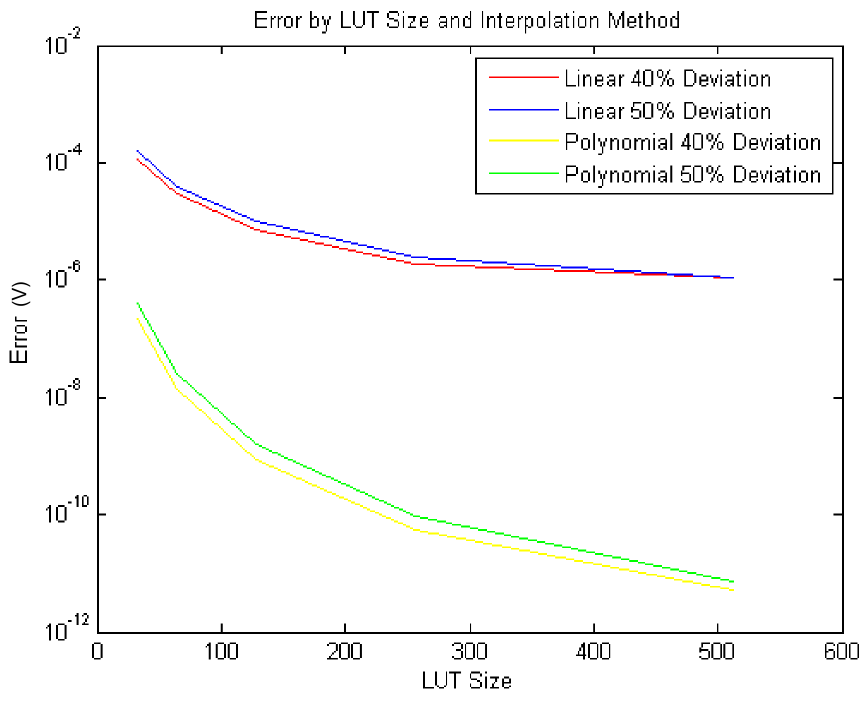

Figure 3.3: Error by LUT Size and Interpolation Method

3.2.2 Cycle Limit Of Accuracy

Figure 3.4 shows the the relationship between the number of cycles for each input to the ADC, the LUT size and the output RMS error. For this graph, an input range of -.65V to .65V, with a step size of 5mV was run through the decoder for various LUT sizes. RMS error was calculated for the outputs over this range. The RMS, or quadratic mean of the

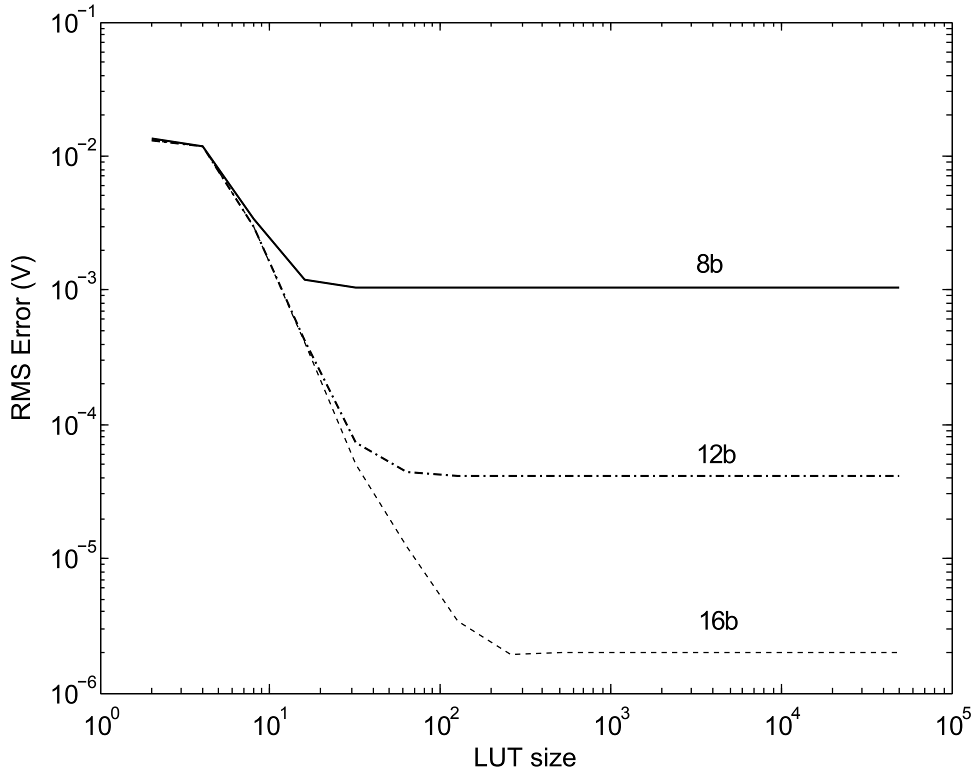

Figure 3.4: Number of Cycles Effect on Output Accuracy

error, can be calculated as can be seen in Equation 3.6. This analysis of the error can take both positive and negative values over the input range [8].

$$Y = \sqrt{\frac{Err_1^2 + Err_2^2 ... Err_n^2}{n}} \tag{3.6}$$

The decoder was using linear interpolation to determine the input value and all wide zero residue decisions, to prevent the error caused by leaving the LUT range, which is discussed in Section 3.2.3.

As can be seen, after a certain point, the size of the LUT no longer has any affect on the final output error. The number of cycles has a much larger roll in the RMS output error than the size of the LUT. Knowing the number of cycles, a maximum LUT size can be determined, beyond which addition points are no longer beneficial.

3.2.3 Residue Mode Limitations

A major problem that occurs during the decode stage is the occurrence of 'Not a Number'(NaN) values in the simulation. This occurs when the final V_{RESOUT} guess and the cycle of decisions leads the V_{RES} value to leave the range of the LUT. One case in which this is possible is when the initial guess is 0 and the last three output decisions are 1, -1, -2.

Decoder cycle 1 takes the -2 D value and outputs a V_{RESOUT} of $-0.68V$. Decoder cycle 2 takes the -1 D and the V_{RESOUT} of $-0.68V$ and calculates a new V_{RESOUT}. If the gain is anything under 2, the resultant V_{RESOUT} will be greater than the V_{REF} value. This is outside of the stored digital amplifier range creating a condition where the digital portion of the circuit cannot calculate a new V_{RESOUT} value.

$$V_{IN} = \frac{V_{OUT}}{G} + D(\frac{V_{REF}}{2}) \tag{3.7}$$

$$V_{OUT} = \frac{0}{<2} + (-2)(\frac{0.68V}{2}) \tag{3.8}$$

$$V_{OUT} = \frac{-.68V}{1.79} + (-1)(\frac{0.68V}{2}) \tag{3.9}$$

In this case, the V_{RES} values quickly reach a value of $-0.7212V$, which is outside the range of the LUT, leading the simulation to return an error value of NaN.

Another issue with the nonlinear gain is a variable correction coefficient. Certain input values will receive greater correction. The simulation gain curve ranges from 1.9 to 1.7. This is, however, only a rough idea of what the actual gain will be at the output. It is likely that the actual circuit fabricated will have a different curve. Simulations were run with a gain curve of 2.5 to 1.3. Although this is an extreme case it illustrates the issues that arise due to variations in gain. With a poor gain curve the ADC can get stuck in low gain areas as can be seen in Figure 3.5. This situation was obtained using the gain curve with extreme nonlinearity.

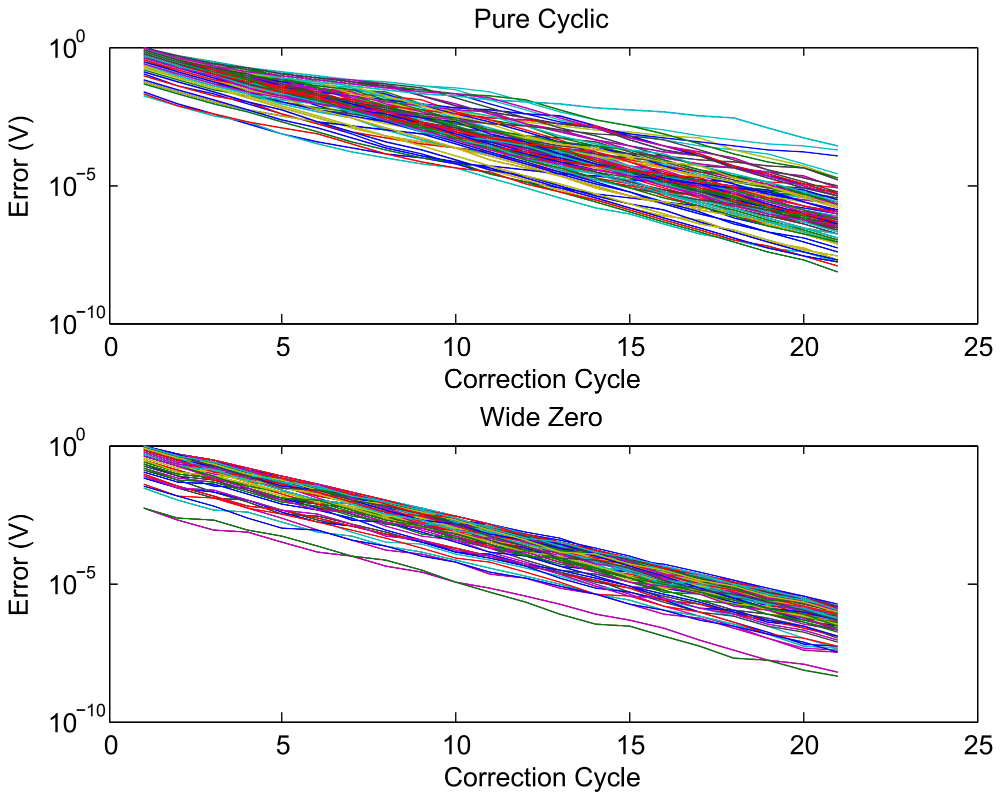

Figure 3.5: Residue Mode and Correction Error with High Nonlinearity

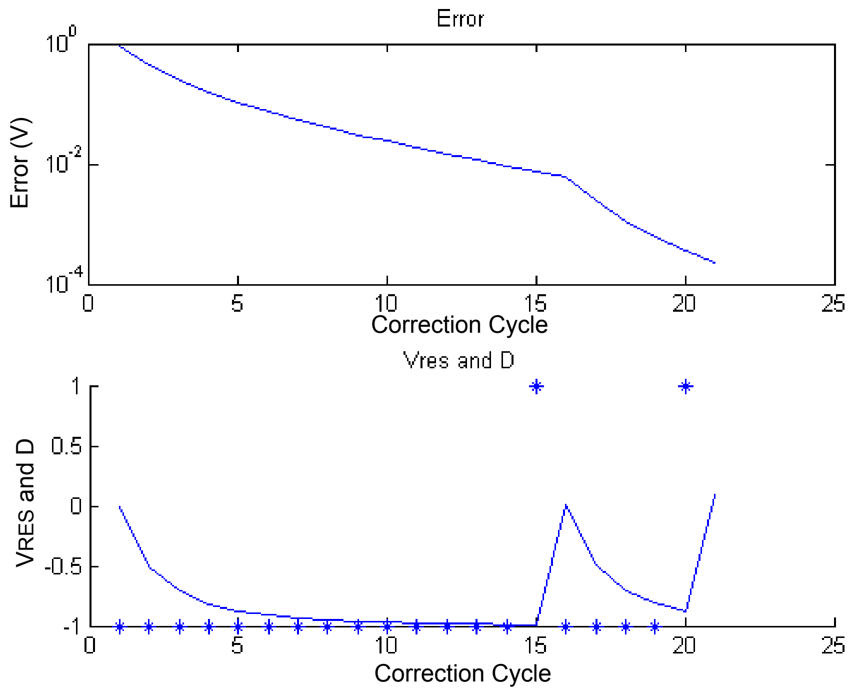

Figure 3.6: Residue Mode and Correction Error with High Nonlinearity

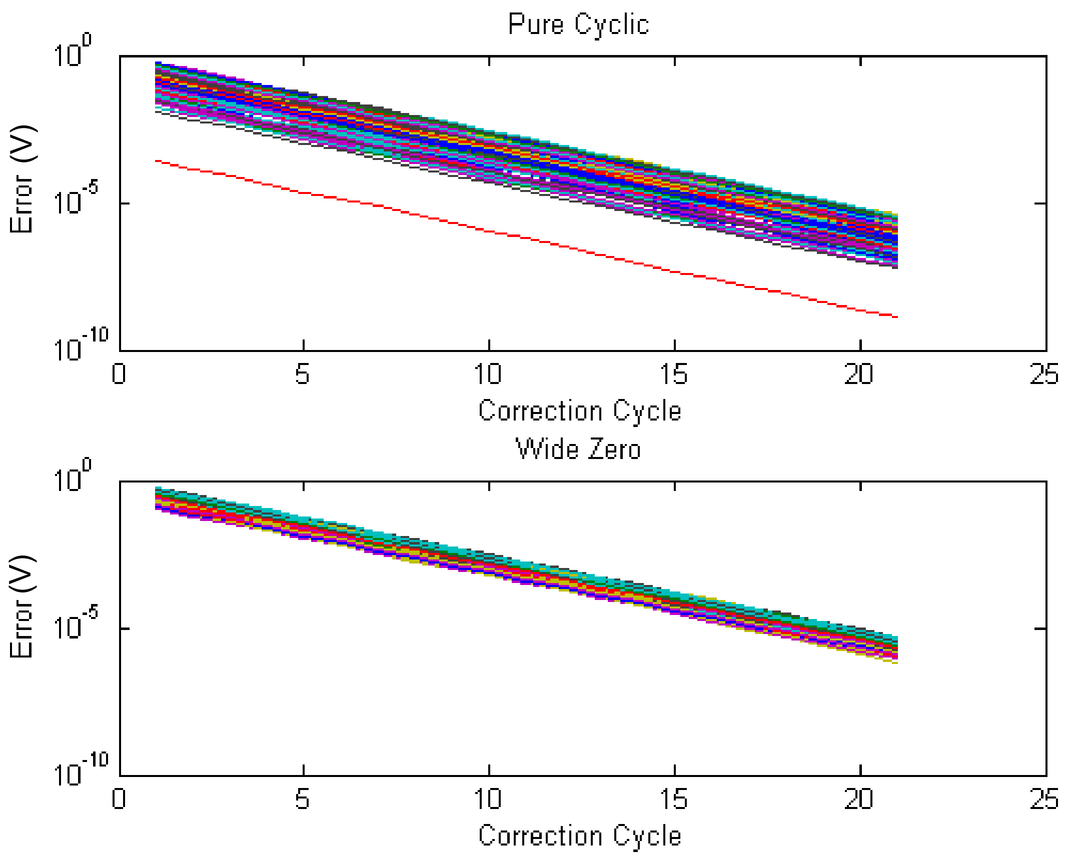

Figure 3.7: Residue Mode and Correction Error with Simulated Gain

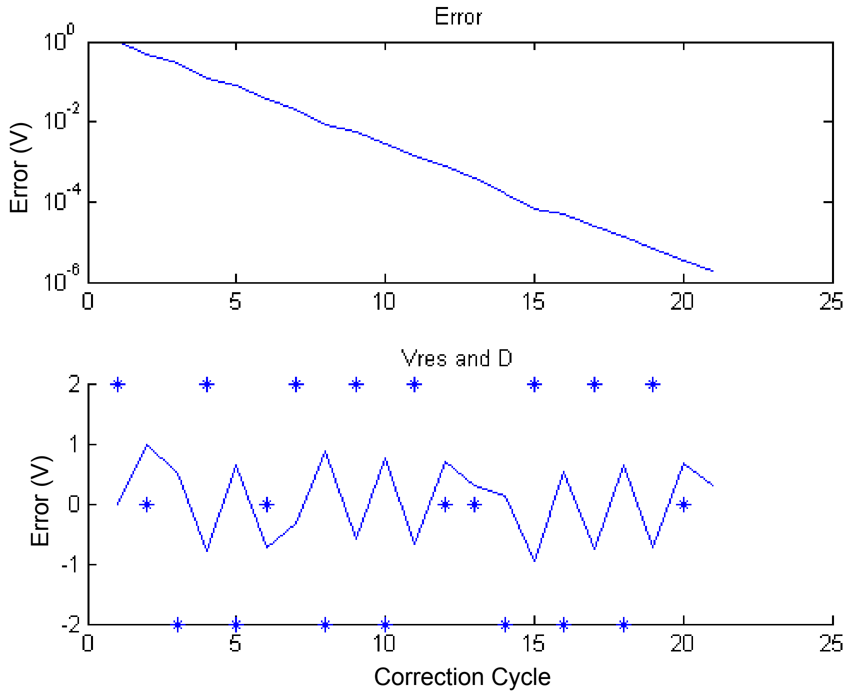

Figure 3.8: Residue Mode and Correction Error with Low Nonlinearity

3.2.4 Solutions

One way to reduce the affect of the lower gain portions of the ADC amplifier is to shuffle residue modes, selecting it randomly each cycle. This prevents the system from getting stuck in the lower gain stages.

As can be seen, reducing the amount of nonlinearity in the gain stage can also reduce the effect of low gain on the correction. It is possible in a practical implementation, as seen in Section 3.3, to not need to shuffle the residue modes and still obtain a comparably low output error.

3.3 LUT Correction Results

Figure 3.9 is seen the result of the LUT based correction compared to a linear approximation. With a LUT size of 2^5 the difference between correction performed with a linear approximation and the correction done by the nonlinear look-up table is a factor of 1000 times different. As is apparent, LUT correction of a cyclic ADC with nonlinear gain is possible with a relatively small LUT size to a high degree of accuracy. The LUT was linearly interpolated, and the input voltage was a ramp from -0.65 V to 0.65 V.

Figure 3.10 shows the difference between a LUT of size 2^6 and 2^8. These were done assuming a LSB based on a 12 bit ADC, $332\mu V$.

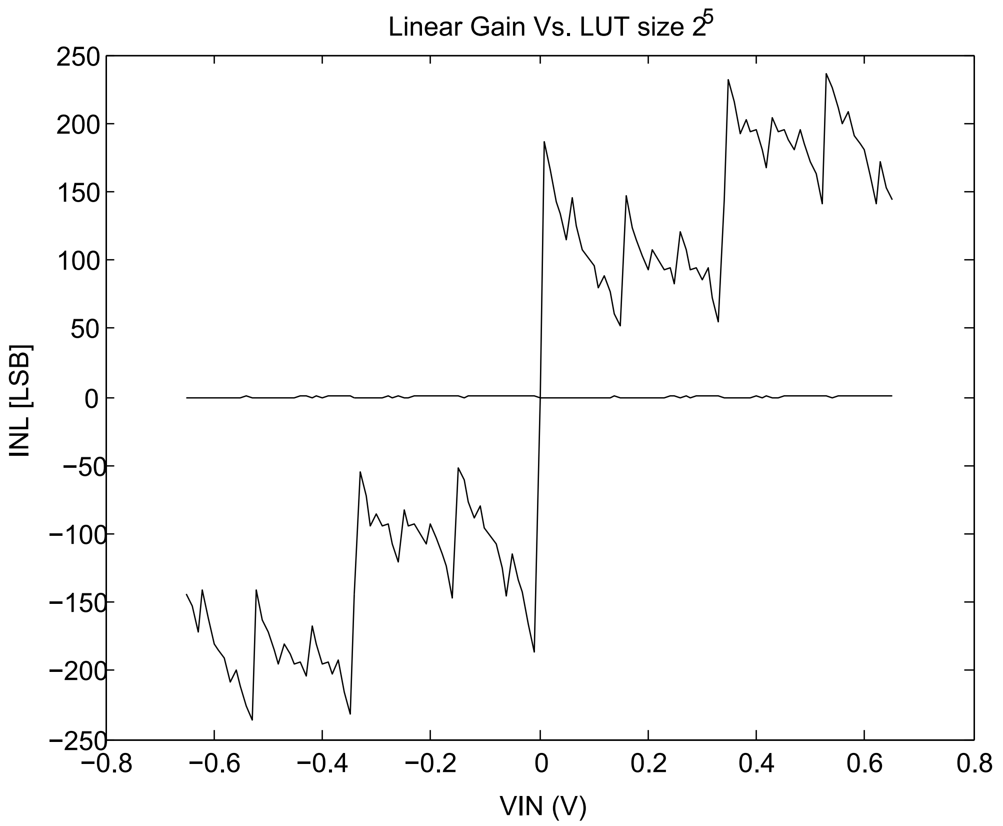

Figure 3.9: INL Linear Correction vs LUT Approach

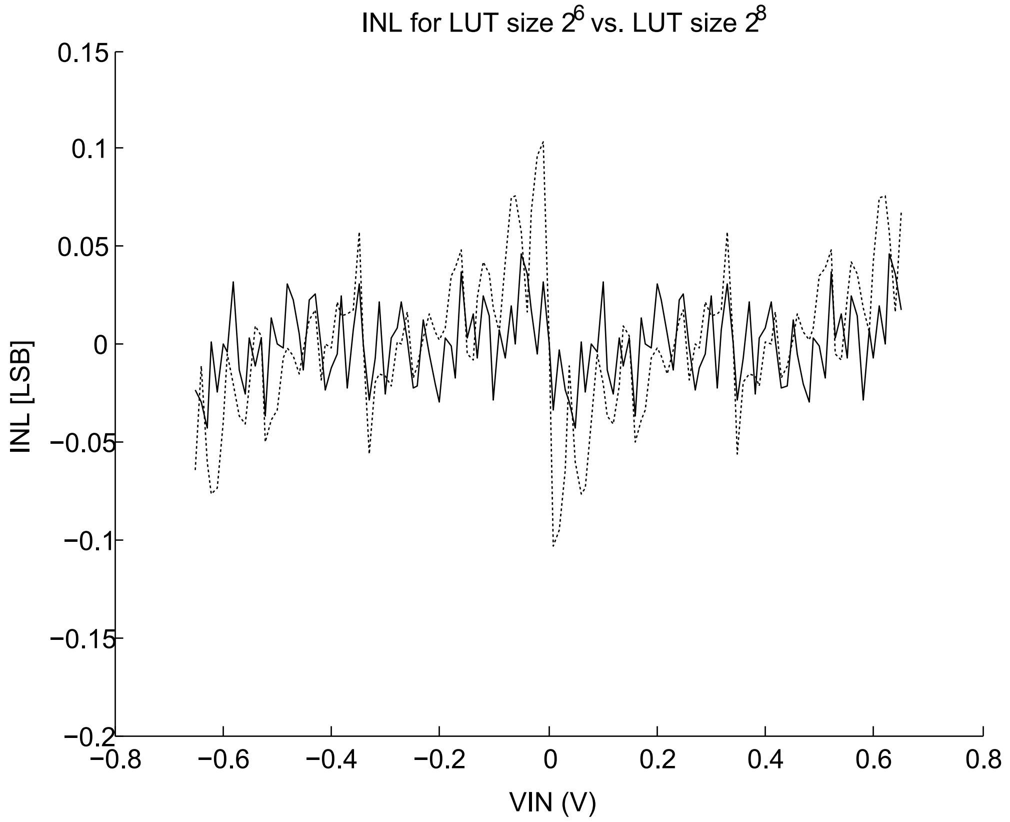

Figure 3.10: INL LUT Sizing

Chapter 4

Calibration

4.1 Overview

The second goal of this work was continuous background calibration of the nonlinear gain LUTs. Background calibration signifies that calibration occurs without disrupting the ADC input signal. Continuous calibration allows for the correction of changes to the amplifier behavior caused by environmental factors such as temperature without taking the converter off line. Previously, correction was performed on a single gain factor. With the LUT method, this is not possible. Gain varies from section to section of the amplifier behavior and is independent of the gain in other portions.

Calibration of the LUT depends firstly on determining whether or not output error is directly reflective of the error in the LUT and a correlation can be drawn. The correlation could be used to make changes to the LUTs, making the output of the ADC more accurate. This section discusses the first step in performing the ADC calibration, which was determining how error in the LUTs manifested in the output of the ADC.

The first part of this chapter discusses the propagation of error in the ADC from the LUT cycle by cycle. The chapter then covers corrections done to the ADC using a converter with a known output error. This output error is used to correct the LUT proving that calibration of the LUT is possible with knowledge of the output error.

4.2 Propagation of Error

Figure 4.1 is the output error of the cyclic ADC when the LUT of a single cycle is incorrect. As is shown, LUT error in the final cycle has the most weight on the output error of the ADC. This was predicted earlier in the report and it allows an initial guess of 0 for the correction input. The error introduced by an incorrect guess is minimal. Calibration can focus on the last few cycles of the ADC.

Ideally, calibration could involve more than the final cycle of the ADC and be used to correct the LUT. To determine the weight earlier cycles would have on the final output error, the simulation was run with incorrect LUTs for each cycle. The final output error was then plotted against the error in previous cycles to determine what, if any, correlation was to be drawn between final error and error in the earlier cycles. Using the polyfit function, a gain of 0.6 was determined to be the error from the previous cycle and then the second to last error is 0.3.

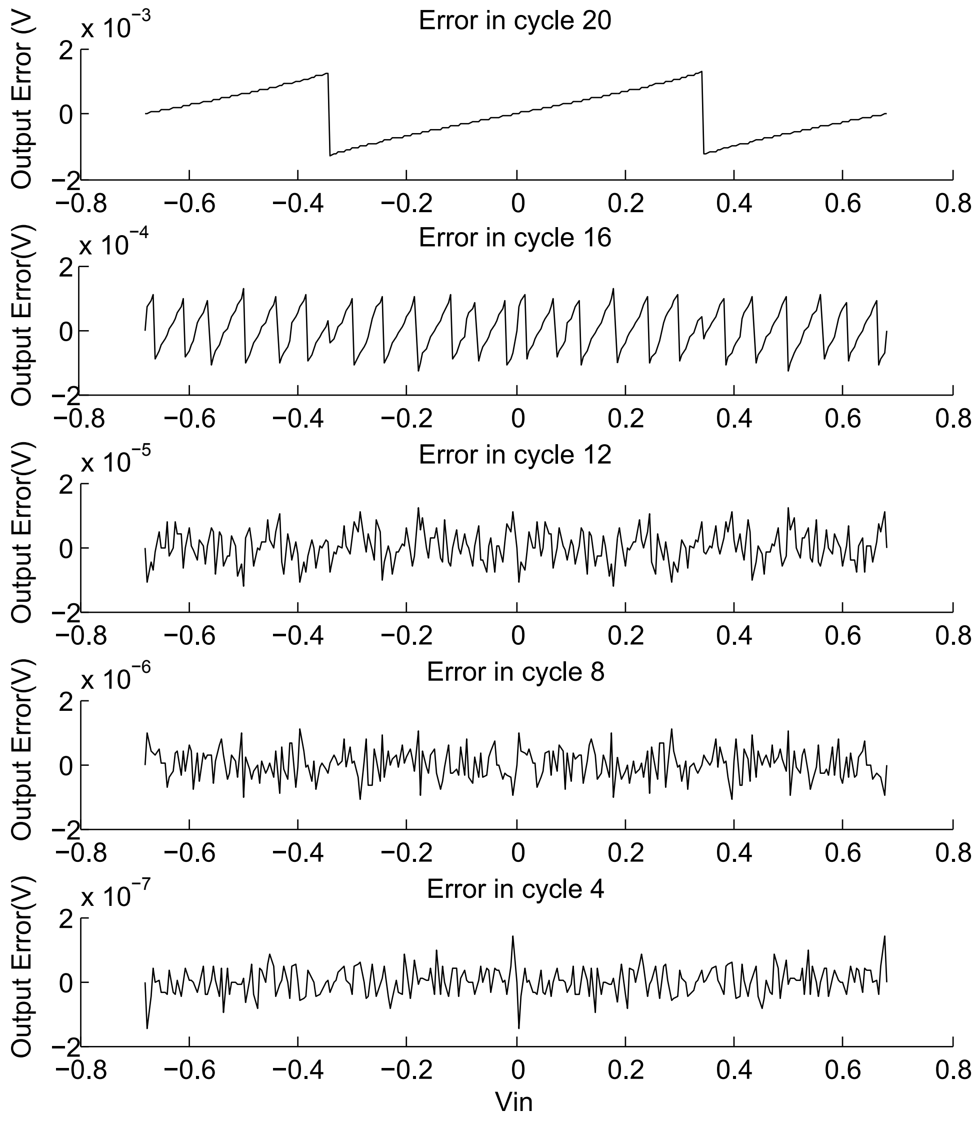

Figure 4.1: LUT Error in Single Cycle

4.3 Calibration with Known Error

To verify that the calibration technique is functioning, a simulation was run using an ideal case for calibration. This involved running two separate ADCs. The first ADC had an error introduced into its LUT. The second ADC had an ideal LUT, meaning that the LUT was taken directly from points on the simulated amplifier curve.

The ADC was then run with a set of input points. After the ADC had output it's first corrected vector of V_{OUT}, the output error was calculated by taking the difference between the two ADC outputs. This information, along with the decision outputs of the ADC, was used to calibrate the incorrect LUT.

$$LUT_{NEW} = LUT_{OLD} + \frac{Er_{SUM}}{Er_{Count}} * \mu \tag{4.1}$$

LUT_{NEW} is the 'calibrated' LUT point. This is equal to the old LUT point plus the sum of all error for outputs using the LUT point, divided by the total number of times the LUT point is used for a set of input values. This is then multiplied by a μ, which initially was the correlation factor of 0.6 determined in Section 4.2.

The results of this calibration can be seen in Figures 4.2 and 4.3. Calibration with a known error, using Equation 4.1 is significantly lower than 1 LSB. This indicates that errors at the output of the correction algorithm of the ADC can be used to calibrate points on the LUT table.

4.4 Calibration of DAC and Comparator Errors

Calibration of the gain in the LUT can also improve the accuracy of the comparators. This would be seen as an offset in the stored LUT values. If the comparators trigger at the wrong point and the DAC subtracts the wrong value, the 0 level in the LUT can be changed.

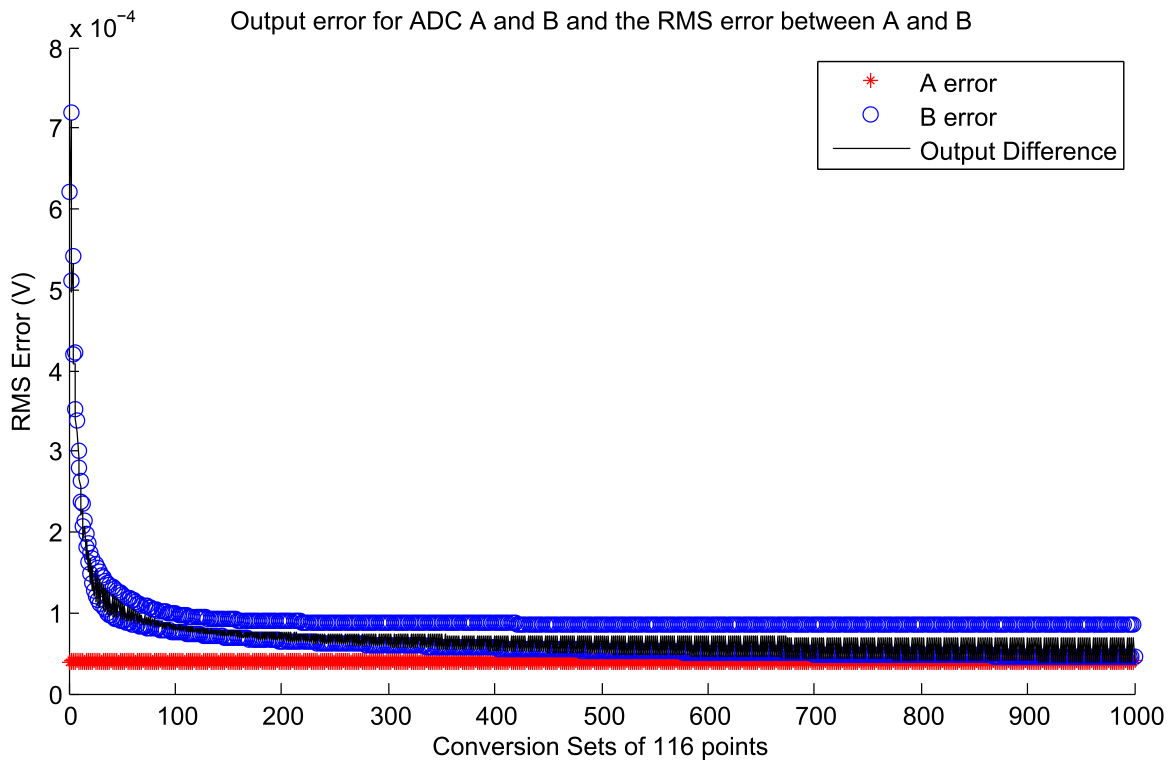

Figure 4.2: Ideal Sine wave Calibration with μ 0.6

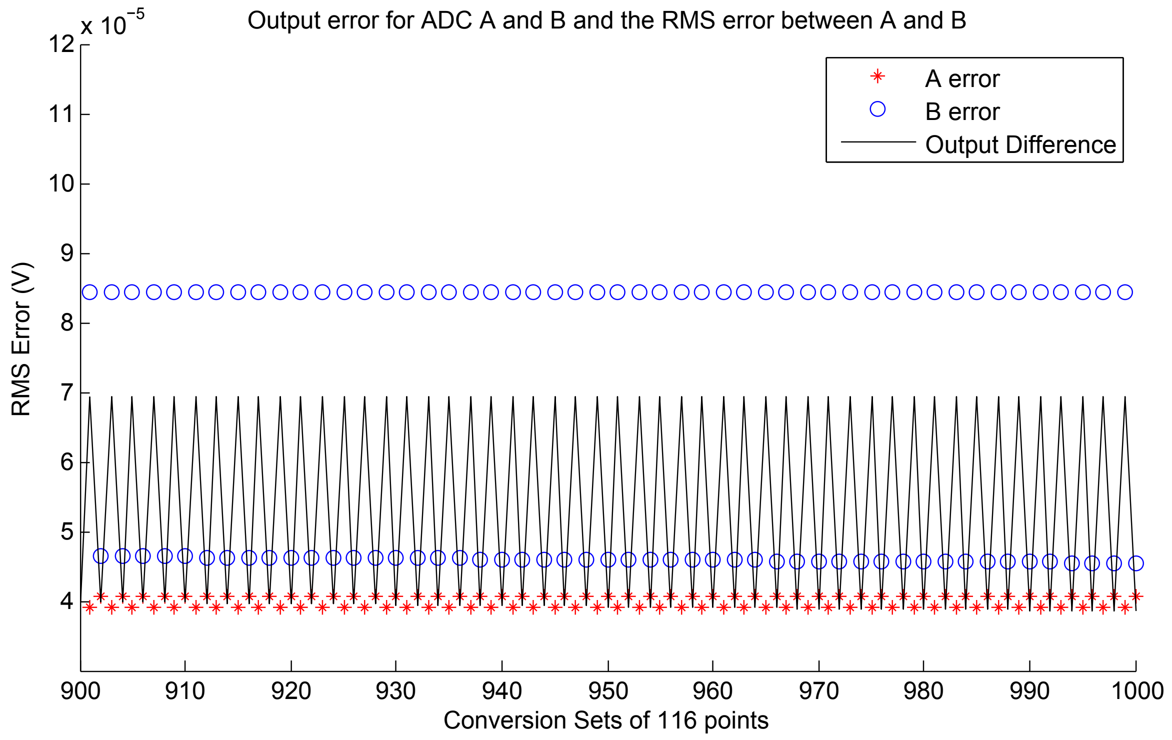

Figure 4.3: Ideal Sine Wave Calibration with High μ 0.6

Chapter 5

Split Cyclic

5.1 Split Cyclic Overview

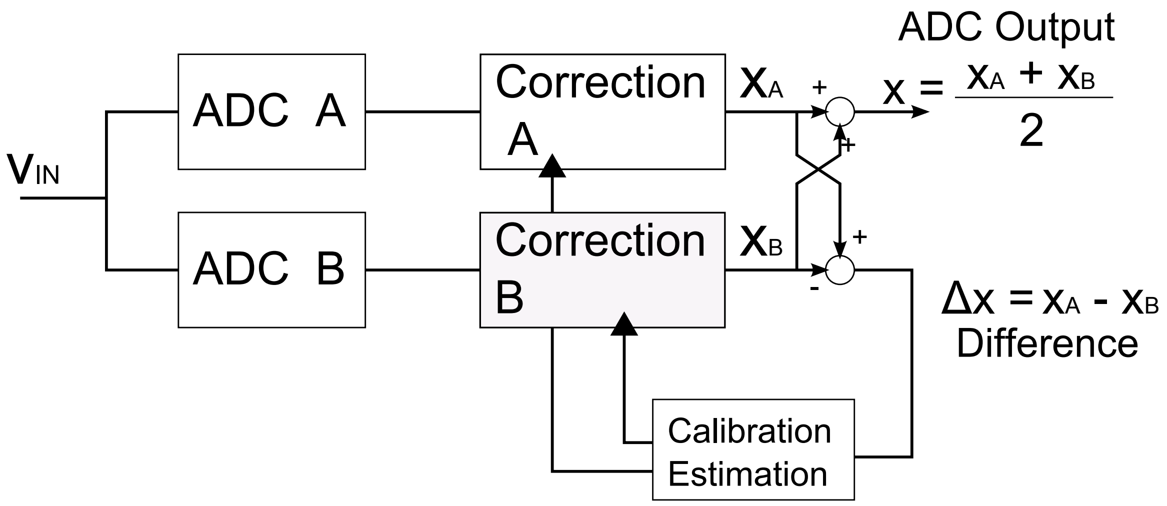

Figure 5.1: Split ADC Architecture

Split ADC is a method for performing background calibration that has been successfully implemented in cyclic ADCs with linear gain stages.

The split cyclic works by taking the difference between two ADCs with the same input [10]. The two ADCs are set to have two different residue modes, so that the output decisions are different. These output decisions, after going though correction, should generate identical output codes, as the two ADCs have the same input. A block diagram of this

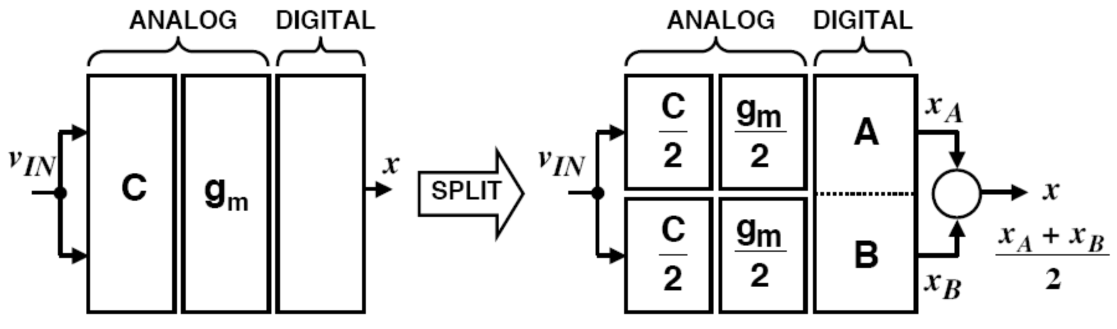

Figure 5.2: Split ADC Compared to Traditional ADC

process is shown in Figure 5.1.

However, if there is any error in the correction process, the outputs will not be the same. The output error can be used to guess what the error in the LUTs in the ADC correction are. This is shown in the previous section where the output error is correlated to the error in the LUT in the last few ADC cycles.

The worst case scenario for the split cyclic ADC calibration is to have identical output errors. To prevent this from happening the residue modes of each ADC are set to be different from each other. To show that this would prevent the output Δx from being identical for the same output table error, the ADC was simulated with a V_{IN} from $-0.65V$ to $0.65V$. ADC A was run using a wide zero residue mode and ADC B was run using the pure cyclic mode. The output difference is shown in Figure 5.3. Even with both LUTs containing an identical error, the output difference is still significant, as shown in Figure 5.3 and can be used to correct the LUTs.

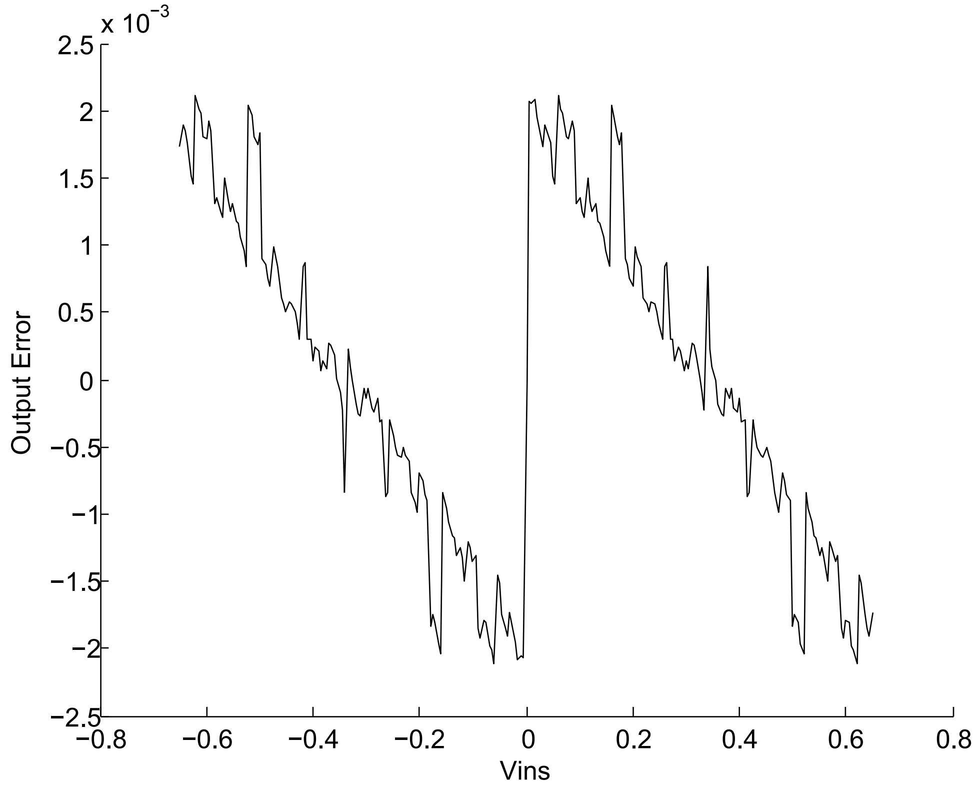

Figure 5.3: Output error with identical LUT

It would be possible for the ADC correction blocks to generate a similar error. To do this their LUTs must be different, as Figure 5.3 proves an identical LUT generates a significant output error. Swapping back and forth between the two residue modes should correct for having identical output errors, but different LUT errors.

5.2 Split Cyclic Implementation

For the split cyclic calibration, two ADCs were run with identical inputs but two different residue modes, as shown in Figure 5.1. The LUTs had identical errors. Initially they were run with V_{IN} as sets of 100 of the same DC voltage. Equation 4.1 was used with μ of 0.6 for the 20th cycle correction and 0.3 for the 19th cycle, as discussed in the previous section. The residue modes switched for each ADC with each iteration of 100 input points.

The split cyclic calibration was almost identical to the calibration with a known error, but with both ADC LUTs containing error and being adjusted.

Figure 5.4 shows the results of calibration. After around 20 cycles, or 2000 sample points calibration is achieved for the DC input. The RMS error reaches a steady $28\mu V$. This is sufficiently accurate to be below 1 LSB, , $332\mu V.$.

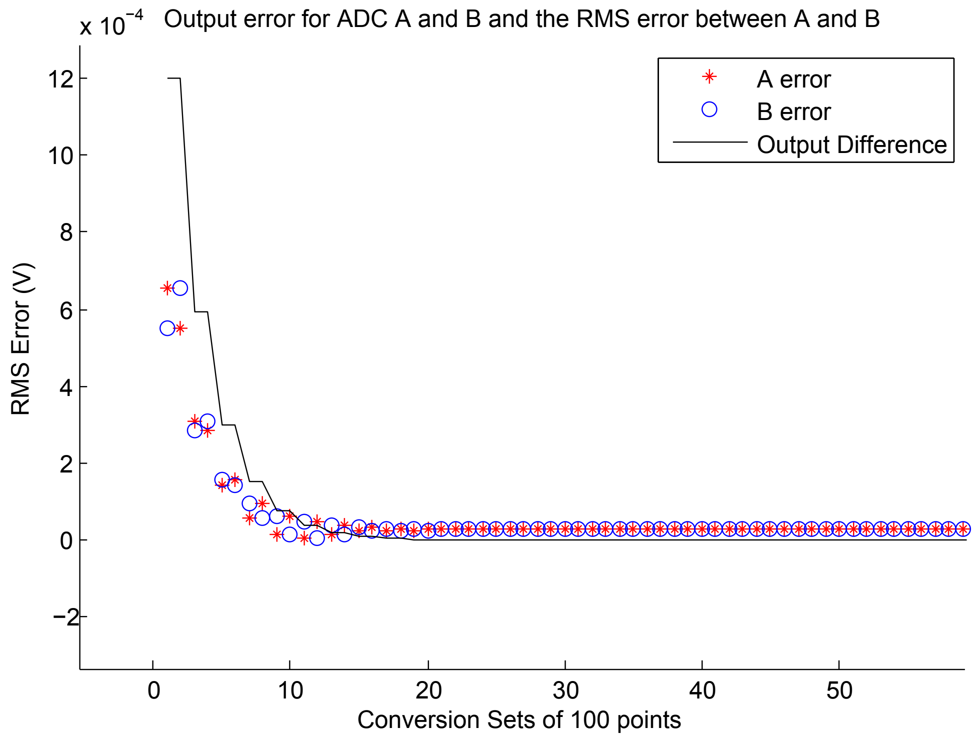

Figure 5.4: DC Calibration

However, when the same technique was applied to a sample of a sine wave, the results were poor.

One problem with this might be overshoot. Because the change factor for each of the points in the LUT is relatively high, there might be overshoot in determining the correct value for the point [5].

However, even if the μ factor is reduced significantly as seen in Figure 5.6, in this case to 0.001, the outputs are still failing to reach a suitable level of calibration.

Another issue may be that the two ADCs have 2 conversion points. The first point is the desired result of a correct ADC output, the second point may be an incorrect value. To try and solve this problem, two new residue modes were introduced and switched in randomly,

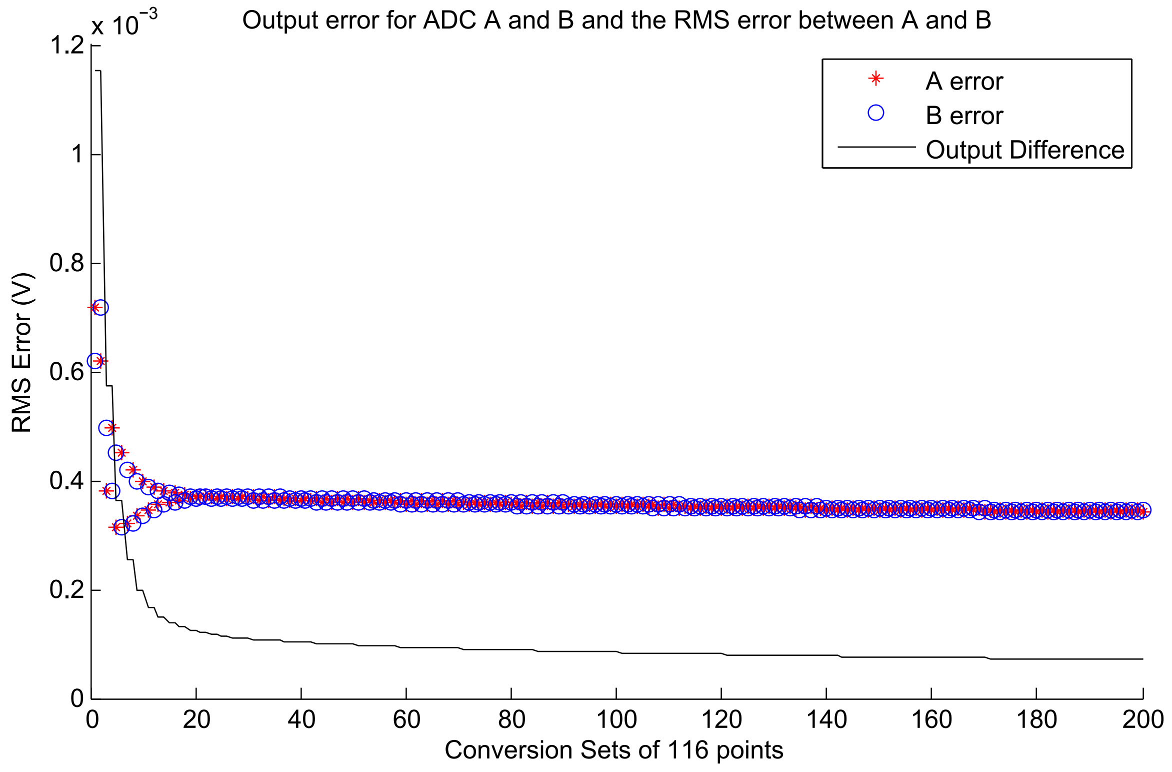

Figure 5.5: Sine Wave Split Calibration

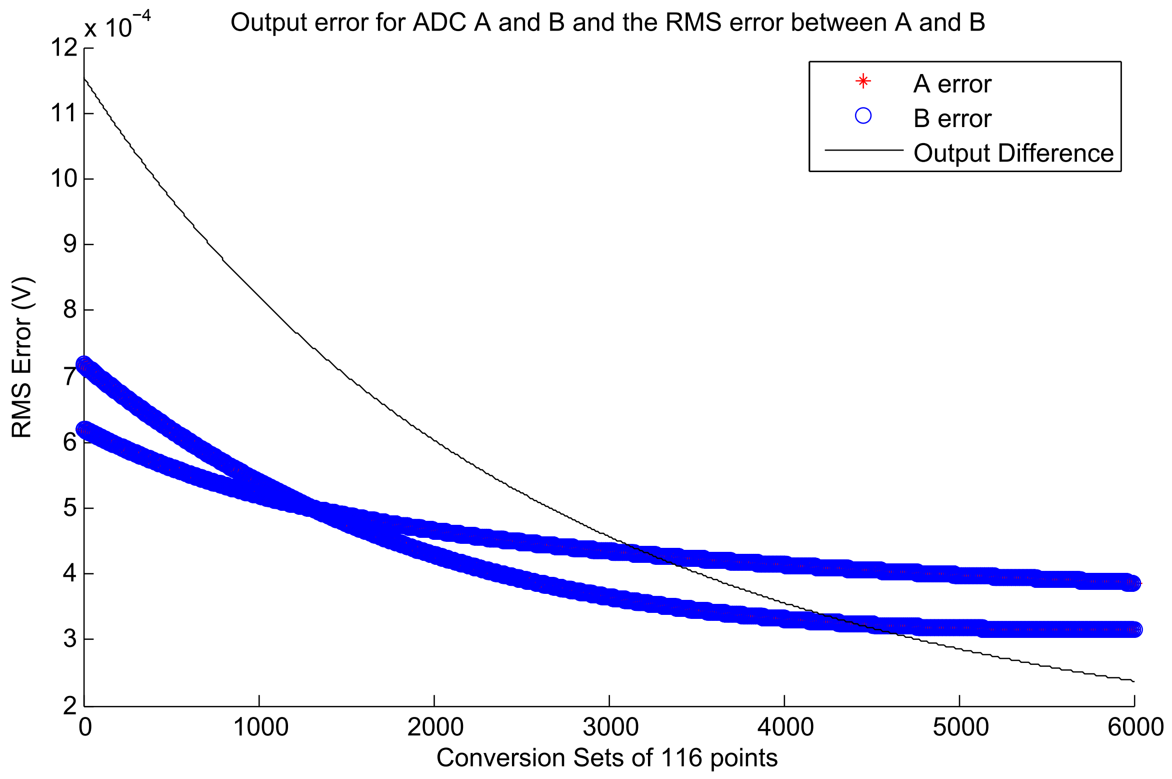

Figure 5.6: Sine Wave Calibration with Low μ

high cyclic and low cyclic. These can be seen in Figure 5.7.

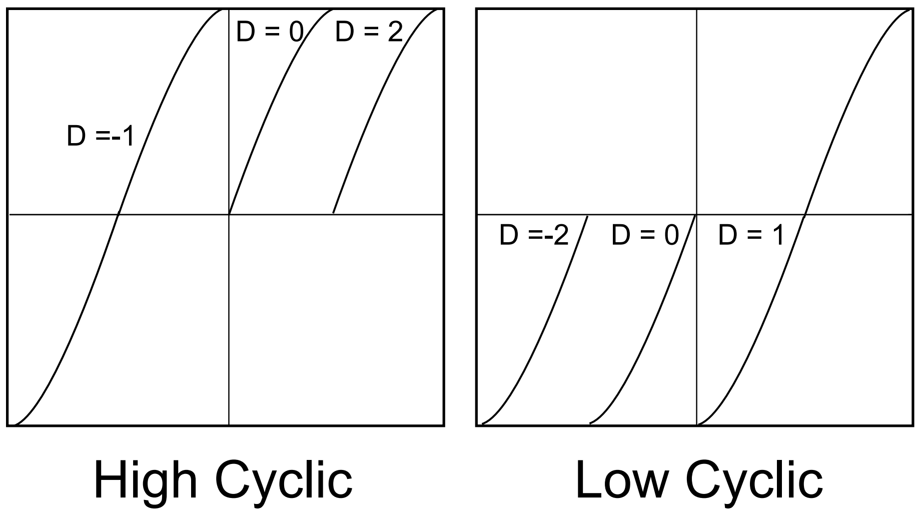

Figure 5.7: Residue Modes (a) High Cyclic (b) Low Cyclic

However, the same issue occurs. The outputs do not reach sufficient levels of calibration, as is shown in Figure 5.8. Looking at the LUT errors in Figure 5.9, the LUT error is much higher in those areas where the gain is much closer to linearity. Weighting from the surrounding LUT points may be useful to even out the final output error and correcting for this. It may increase the correction done at points furthest from the center, where error is the highest.

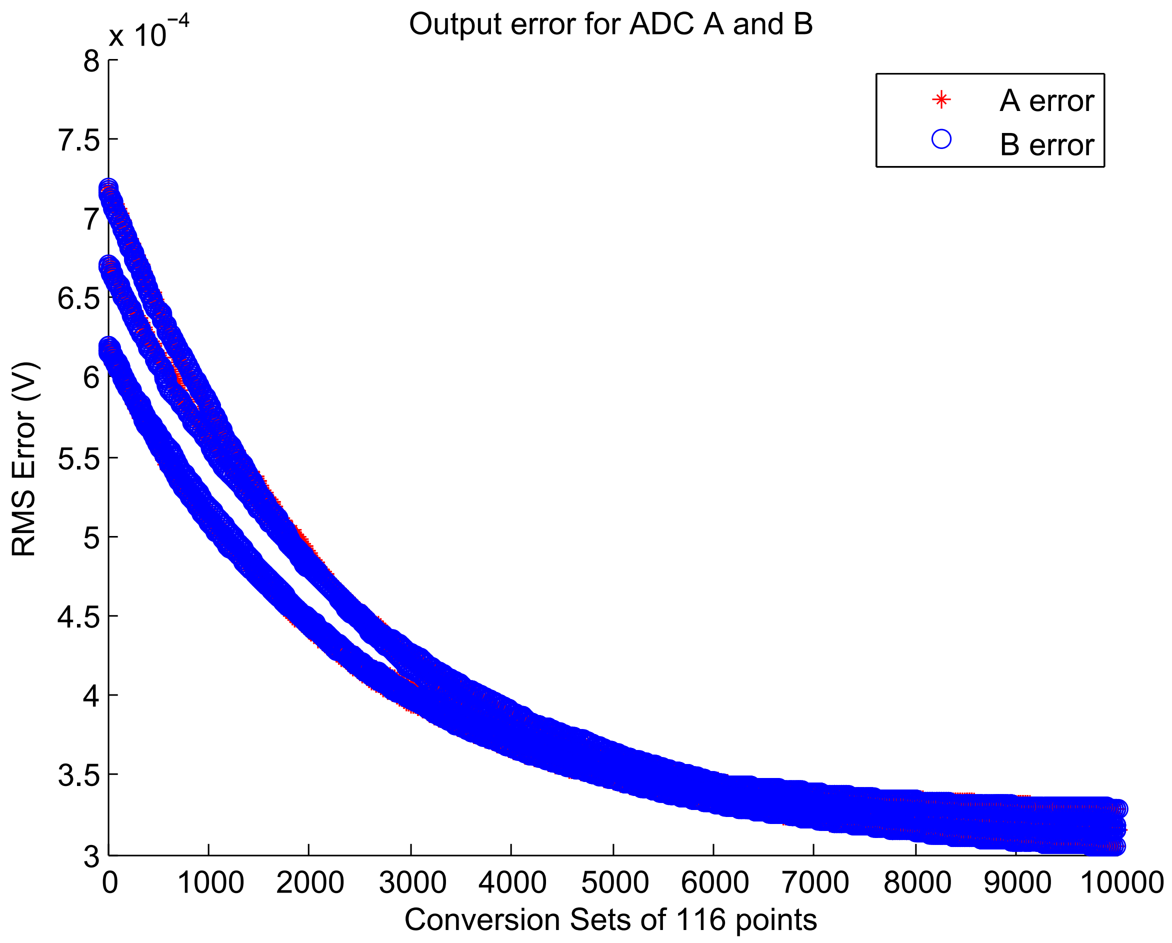

Figure 5.8: Calibration Using 4 Residue Modes

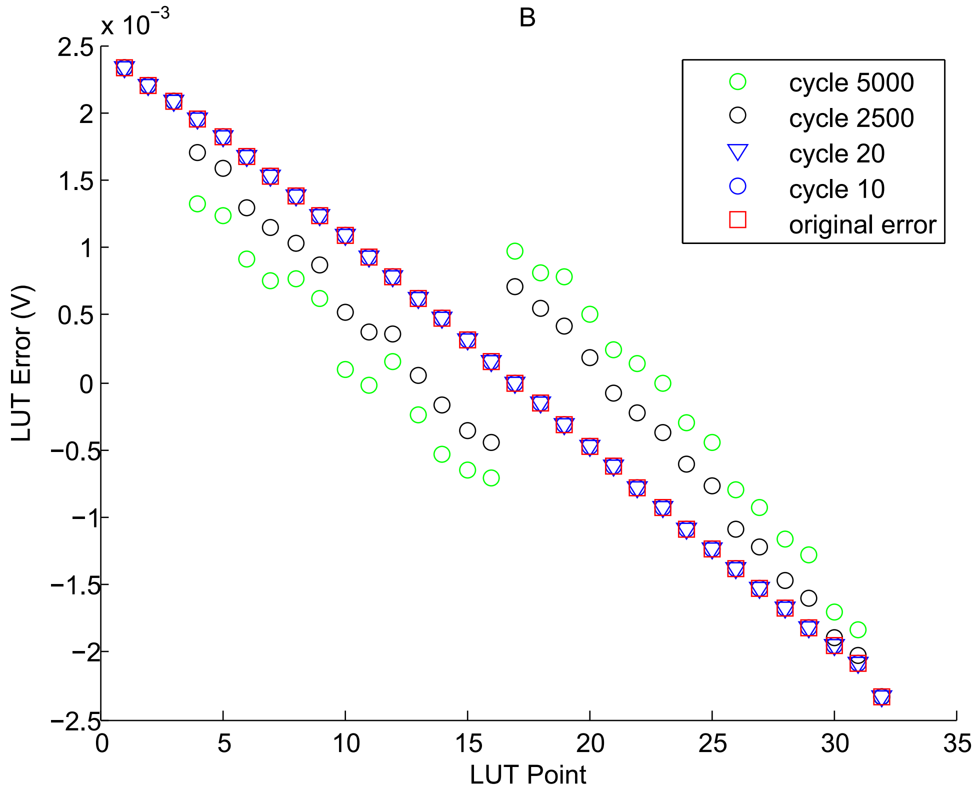

Figure 5.9: LUT Error after Iterations

Chapter 6

Conclusions and Future Work

6.1 Conclusions

LUTs are an effective way to compensate for nonlinearity in cyclic ADCs. A LUT of size 2^5, using linear interpolation can effectively correct an 12 bit cyclic ADC. The complexity of calculations is relatively low with linear interpolation and can be implemented with minimal digital logic. In addition to the use of minimal digital circuitry, the LUT method of correction has the advantage of allowing a considerable leniency in the design of the amplifier, as show with idealized calibration done with an amplifier of over 20 percent nonlinearity.

The error at the output can be tracked down to specific LUT points verifying calibration is possible. The split cyclic method is advantageous because it is performed continuously and without disruption of the ADC output. It is shown that even if the ADC LUTs have the same error, there will still be a correction factor at the output.

Although calibration has not been performed to 12 bit accuracy, the framework has been created to further pursue the split ADC architecture.

6.2 Future Work

Future work will involve the implementation of this algorithm using digital logic to perform correction and calibration. An IC is in development using the cyclic ADC parameters, including the amplifier specifications as discussed in this paper. An FPGA would be a rapid method for calibration and corrections. The LUTs may be small enough for on chip memory use. This would allow for a small, low power, portable ADC that could be used for high resolution applications.

6.2.1 Weighting

Several methods could be implemented to improve the calibration technique. One method that could involve weighting the errors used at each of the LUT points. The point before and after the interpolated point, A and B in Figure 6.1, are known. Both of the points are not used equally, however, in calculating the final output value. The point which is closer to the interpolated voltage should be altered to a greater degree than the other point depending on the output error. This error weight was determined by how close the point used was to each of the two points.

$$Weight_A = \frac{V_{IN} - B}{A - B} \tag{6.1}$$

$$Weight_B = 1 - \frac{VIN_{IN} - B}{A - B} \tag{6.2}$$

In addition to just the individual point weighting, calculating the number of times the LUT point is used in comparison can give a better idea of whether the error calculated for it is representative of an actual error in the system. For example, if a point is not used, in the current manifestation of the calibration procedure, it's error is calculated as 0. However, this may not be the case. If a point adjacent to it is used frequently and the error for that point is high, it is likely that the error in the unused point is also related to the error in the heavily used point and should be adjusted accordingly.

6.2.2 Use of Previous Cycle Residue

Because the digital portion of the circuit is also calculating V_{RES} for all of the cycles up until the output of the ADC, more accurate information about the cycles leading up the final cycle can be obtained by calculating the output error at previous cycles. This information

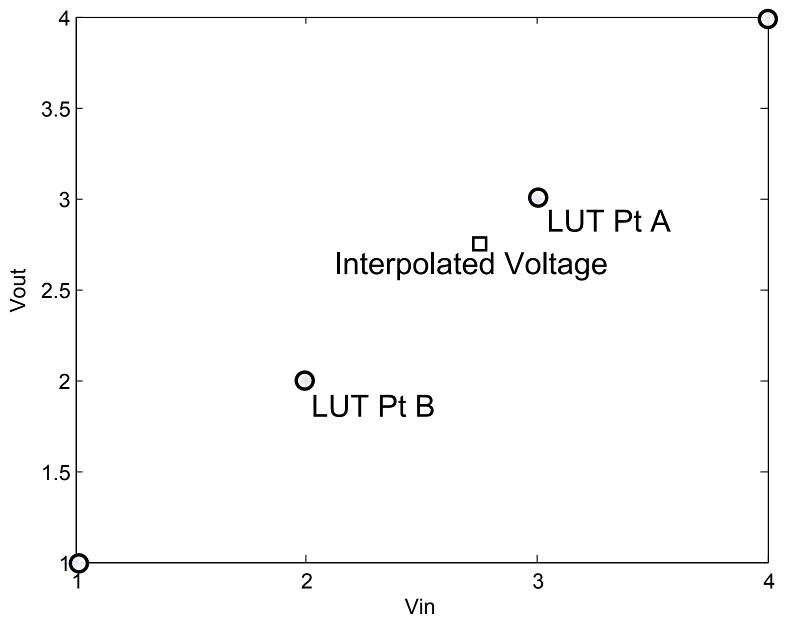

Figure 6.1: Weighting

could be used to correct earlier cycles in the ADC. Though it would be preferable not to have to store another layer of information from the output of the ADC, and perform another set of calculations to determine the output error for previous cycles, the information could lead to better correction of the ADC LUTs.

Appendix A

Top Level MATLAB Simulation

```
%  Nonlin Split Cyclic ADC Top Level
clear
% VARIABLES
    % dcrange     scaled linear input voltage vector, used for LUT
      and ADC
    % nonlin      nonlinear amplifier curve from MQP simulation

    % vins        vector of input voltages
    % ncycles     number of cycles for adc

    % vref        reference voltage
    %     Comparator Values
    % vhi         high decision level (wide zero residue)
    % vmid        mid decision level  (pure cyclic)
    % vlo         low decision level  (wide zero)
    %
% loads simulation data     -.34 to .34 DC input range
%                           -0.6167 to 0.6167 amplifier output

load M:\matlabADC\mqpgroupdata\err\shants
dcrange= 2*dcrange;  % fixes simulation data that is not
   differential
```

```
%%
sima(1,:) = nonlin;   % sim is the 'real' amp behavior from
    simulation
sima(2,:) = nonlin;   % ADC A
sima(3,:) = nonlin;
sima(4,:) = nonlin;
sima(5,:) = nonlin;

simb(1,:) = nonlin;   % sim is the 'real' amp behavior from
    simulation
simb(2,:) = nonlin;   % ADC B
simb(3,:) = nonlin;
simb(4,:) = nonlin;
simb(5,:) = nonlin;

%  "Pure Cyclic"
%       /|   /
%      / |  /
%     /  | /
%    /   |/
%  LUT2 : LUT4

%  "Wide zero"
%     / |   /|
%    /  |  / |
%       | /  | /
%       |/   |/
%  LUT1:LUT3: LUT5

%% create look-up tables for correction
n = .33;   % n is gain error
```

```
lut_create %% sets up LUTs and error
           % LUTa(1,:,:), LUTb(1,:,:), LUTi(1,:,:)
%lut_createideal  % LUT a is ideal, or LUT without error

%% Set ADC [cycle number,] modes, residues and Vins
vins = sin(1:.1:4*pi)* .5760;
%vins = (rand([1,1000])*1.20) − .6;
%vins = .57 * ones(1,100);% dc @ 90% − 100
%vins = zeros(1, 100);
%vins=(−.65:.005:.65);  % set input to  −.65 to .65 slope (within
   vref)

%%%%% COMPARATOR THRESHOLDS
vref = 0.68;
vhi=+vref/2;
vmid=0;
vlo=-vref/2;

%%%%% ADC Settings
ncycles = 20;

%%RESIDUE MODE
% Residue mode changes BY CYCLE (cyclenl2)
%chooser (1,:) = randsample(0:1, ncycles, 1);
%%% set RANDOM 'chooser' for wz or pc
chooser (1,:) = zeros(1,ncycles);
%%% 1 ALL Are PC, 0 all are WZ NON RANDOM
%chooser(1,20) = 1;  % prevents some of NaN from vres leaving amp
   range
%chooser(1,19) = 1;
%chooser(1,18) = 0;
```

```
%% SPLIT ADC SIM
%%%%% inputs
%%%     sima            A amplifier from simulation
%%%     simb            B amplifier from simulation
%%%     vins            input voltages
%%%     vref
%%%     ncycles
%%%     chooser
%%%
%%%%% outputs
%%%     compouta        comparator outputs (D) from A
%%%     compoutb        comparator outputs (D) from B
%%%     vresiduesa      vresidues from A
%%%     vresiduesb      vresidues from B

splitADCsim %%%% uses pure cyclic and WZ res modes

    compoutalrg(:,:,1) = compouta;    % alternates res mode
    compoutblrg(:,:,1) = compoutb;

    compoutalrg(:,:,2) = compoutb;
    compoutblrg(:,:,2) = compouta;

splitADCsim3  %%%% uses high and low res modes

    compoutalrg(:,:,3) = compouta;    % alternates res mode
    compoutblrg(:,:,3) = compoutb;

    compoutalrg(:,:,4) = compoutb;
    compoutblrg(:,:,4) = compouta;

%% Split Decoder
%%%%% decoder
```

```
%%%%% inputs
%%%     LUTa
%%%     LUTb
%%%     compouta
%%%     compoutb
%%%     m            %times decoded/ LUTs to use
%%%
%%%%% outputs
%%%     decoderresiduesa
%%%     decoderresiduesb

m = 1;    %% Decode time! First time!

%%% decoder need 2—D matrix for compout
%%% but compout is 3—D. squeeze fixes this.
    compouta = squeeze(compoutalrg(:,:,1));
    compoutb = squeeze(compoutblrg(:,:,2));

    decoder


%% Adjust A
%%%%% Calibrated LUTS
%%%%%%
%%%%% calibration
%%%%% inputs
%%%     LUTa (m)
%%%     LUTb (m)
%%%     decoderresiduesa
%%%     decoderresiduesb
%%%     mu     — calibration factor
%%%
%%%%% outputs
%%%     LUTa (m+1)
%%%     LUTb (m+1)
%%%

mu1 = .001;    %% correction factor
calibration
%%calibrationideal    %%% when using 1 idea LUT

    actualerrora(:,m) = squeeze(decoderresiduesa(21,:,m)) - vins;
    rmsa(m)    = norm(actualerrora(:,m))/sqrt(length(actualerrora(:,m
       )));
    actualerrorb(:,m) = squeeze(decoderresiduesb(21,:,m)) - vins;
    rmsb(m)    = norm(actualerrorb(:,m))/sqrt(length(actualerrorb(:,m
       )));
    splitdif = (squeeze(decoderresiduesa(21,:,m)) - squeeze(
       decoderresiduesb(21,:,m)));
    rmssplit(m) = norm(splitdif)/sqrt(length(splitdif));

%%
%%% iterations is number of correction cycles to go through
iterations = 4*5000;

%%%preallocate for speed
decoderresiduesa = (zeros(21,size(vins,2),iterations));
decoderresiduesb = (zeros(21,size(vins,2),iterations));

%%
for m = 2: iterations

    %%% this sets the residue mode to change in A every 1000
       iterations
    %%% and in B every 100 iterations
    compouta = squeeze(compoutalrg(:,:,round((mod(m/1000,3))+1)));
    compoutb = squeeze(compoutblrg(:,:,round((mod(m/100,3))+1)));
    %%% this sets the residue mode to change in A and B randomly
    %compouta = squeeze(compoutalrg(:,:,randi(4,1,1)));
    %compoutb = squeeze(compoutblrg(:,:,randi(4,1,1)));

   decoder
   calibration
 %%%calibrationideal

    %%% splitdif is the difference between the output of the 2 ADCs
    splitdif = (squeeze(decoderresiduesa(21,:,m)) - squeeze(
       decoderresiduesb(21,:,m)));
    hold on

    actualerrora(:,m) = squeeze(decoderresiduesa(21,:,m)) - vins;
    rmsa(m)    = norm(actualerrora(:,m))/sqrt(length(actualerrora(:,m
       )));
    actualerrorb(:,m) = squeeze(decoderresiduesb(21,:,m)) - vins;
    rmsb(m)    = norm(actualerrorb(:,m))/sqrt(length(actualerrorb(:,m
       )));

    rmssplit(m) = norm(splitdif)/sqrt(length(splitdif));

end;
```

Appendix B

LUT Creation

```
%%LUT_EDIT



l = 5;     %%% 2^l is LUT size
dclut                  = dcrange(1:ceil(size(dcrange,1)/(2^l)):end);
%selects points from DC input
nonlinlutideal         = nonlin(1:ceil((size(nonlin,1)/(2^l))):end);
% selects points from amp output
dclut(end)             = dcrange(end);
% ensures that LUT covers range by taking last pt.
nonlinlutideal(end)   = nonlin(end);

% create idea LUTS
LUTi(1,:,1) = nonlinlutideal;
LUTi(2,:,1) = nonlinlutideal;
LUTi(3,:,1) = nonlinlutideal;
LUTi(4,:,1) = nonlinlutideal;
LUTi(5,:,1) = nonlinlutideal;

%%% Error introduced into LUT
nonlinlut = nonlinlutideal .* 1.01^n;
%nonlinlut = nonlinlutideal - .005*n;
```

```
%nonlinlut = (1.8 + .001 * n)* dclut;  %%% sets up lut as constant
    gain
%nonlinlut = (1.8 + .001 * n)* dclut;  %%% sets up lut as constant
    gain
LUTa(1,:,1) = nonlinlut;
LUTa(2,:,1) = nonlinlut;
LUTa(3,:,1) = nonlinlut;
LUTa(4,:,1) = nonlinlut;
LUTa(5,:,1) = nonlinlut;
%  LUTa(1,1,:) = nonlinlutideal;
%  LUTa(1,2,:) = nonlinlutideal;
%  LUTa(1,3,:) = nonlinlutideal;
%  LUTa(1,4,:) = nonlinlutideal;
%  LUTa(1,5,:) = nonlinlutideal;


LUTb(1,:,1) = nonlinlut;
LUTb(2,:,1) = nonlinlut;
LUTb(3,:,1) = nonlinlut;
LUTb(4,:,1) = nonlinlut;
LUTb(5,:,1) = nonlinlut;

%  "Pure Cyclic"
%      /|   /
%     / |  /
%    /  | /
%   /   |/
%  LUT2 : LUT4

%  "Wide zero"
%    / |   /|
%   /  |  / |
%        | /   | /
%        |/    |/
%  LUT1:LUT3: LUT5
```

Appendix C

ADC Simulation

```
%% Split ADC Sim

%%%%% inputs
%%%     sima            A amplifier
%%%     simb            B amplifier
%%%     vins            input voltages
%%%     vref
%%%     ncycles
%%%     chooser
%%%
%%%%% outputs
%%%     compouta        comparator outputs (D) from A
%%%     compoutb        comparator outputs (D) from B
%%%     vresiduesa      vresidues from A
%%%     vresiduesb      vresidues from B
%%%

%% ADC Simulation A
% Set variables for a
sim = sima;          % sets up A amplifier
chooser = chooser;   % sets up resmode
```

```
%%preallocate
vresidues = zeros(ncycles, size(vins,2));
compout   = zeros(ncycles, size(vins,2));

%%% Residue Mode variable by cycle
for k = 1:ncycles
    if  k <1.5
        cyclechsr = chooser(k);  %%determines which residue mode
        cyclein = vins;          %% sets vres in  to vins
        cyclen_2;                %% calcs vresidues and D
        vresidues(1,:)= vins;    %% stores
    else
        cyclechsr = chooser(k);
        cyclein = vresidues(k,:);  %% sets vres in to previous res
           out
        cyclen_2
    end
        vresidues(k+1,:)= vouts(1,:);
        compout(k,:)= vouts(2,:);
end

vresiduesa = vresidues;
compouta = compout;

%% ADC Simulation B

% Set variables for B
sim     = simb;      % sets up B amplifier
chooser = ~chooser;  % sets up resmode as not A resmode

%%preallocate
vresidues = zeros(ncycles, size(vins,2));
compout   = zeros(ncycles, size(vins,2));
```

```

%%% Residue Mode variable by cycle
for k = 1:ncycles
    if  k <1.5
        cyclechsr = chooser(k);  %%determines which residue mode
        cyclein = vins;          %% sets vres in  to vins
        cyclen_2;                %% calcs vresidues and D
        vresidues(1,:)= vins;    %% stores
    else
        cyclechsr = chooser(k);
        cyclein = vresidues(k,:);  %% sets vres in to previous res
           out
        cyclen_2
    end
        vresidues(k+1,:)= vouts(1,:);
        compout(k,:)= vouts(2,:);
end

vresiduesb  = vresidues;
compoutb    = compout;
```

Appendix D

V_{RES} and D calculation Pure Cyclic and Wide Zero Residue Modes

```
%% Cyclen — Generates Vres and D

%%%%%%%%% inputs
%%% dcrange   scaled linear −0.5 to 0.5 input voltage vector
%%% nonlin    scaled representative nonlinear amplifier curve
%%% vins      vector of input voltages
%%% vref      reference voltage
%%% vhi       high decision level (wide zero residue)
%%% vmid      mid decision level  (pure cyclic)
%%% vlo       low decision level  (wide zero)
%
%%% INTERNAL VARIABLES
%%% lodec,hidec               Comparator descisions
%%% compoutwz, compoutpc      Comparator outputs (−2, −1, 0, 1, 2)
%%% vreswz,vrespc             Residue voltages after cyclic
   operation
%
%%%%%%%% outputs
```

```
%%% vouts       4XN Vector with PC and WZ residue voltages in rows 1
    and 2
%%%             and PC and WZ comparator decisions in rows 3 and 4

%%preallocate/clear
compoutwz = zeros(1, size(vins,2));
compoutpc = zeros(1, size(vins,2));
vres(:,:) = zeros(5, size(vins,2));

if chooser(k)== 0
    % Wide Zero
    cyclein<vlo ;lodecwz = ans;  %% should use simulation curve 1 [
       sim1]
    cyclein>vhi ;hidecwz = ans;  %% should use sim5
    middecwz = ~lodecwz&~hidecwz;                    %% else sim3

    compoutwz = 2*(hidecwz - lodecwz);  %% makes values +-2

    % Calculate Residue
    % WZ
    vres(1,:) = interp1( dcrange ,sim(1,:), (cyclein - compoutwz.*(
       vref/2)), 'spline') .* lodecwz;
    vres(3,:) = interp1( dcrange ,sim(3,:), (cyclein - compoutwz.*(
       vref/2)), 'spline') .* middecwz;
    vres(5,:) = interp1( dcrange ,sim(5,:), (cyclein - compoutwz.*(
       vref/2)), 'spline') .* hidecwz;

else
    % Pure Cyclic
    cyclein<vmid;lowdecpc=ans;  %% Should use sim2
    cyclein>vmid;hidecpc=ans;   %% Should use sim4

    compoutpc = hidecpc-lowdecpc; %%
    % PC
    vres(2,:) = interp1( dcrange ,sim(2,:), (cyclein - compoutpc.*(
       vref/2)), 'spline').* lowdecpc;
    vres(4,:) = interp1( dcrange ,sim(4,:), (cyclein - compoutpc.*(
       vref/2)), 'spline').* hidecpc;

end

vouts =[ vres(1,:)+ vres(2,:)+ vres(3,:)+ vres(4,:)+ vres(5,:) ;
   compoutpc + compoutwz];
```

Appendix E

V_{RES} and D calculation High and Low Residue Modes

```
%%%%%%%%%%%%%%%%%%%%%%% VARIABLES
%
% INPUT ARGUMENTS
% dcrange   scaled linear -0.5 to 0.5 input voltage vector
% nonlin    scaled representative nonlinear amplifier curve
% vins      vector of input voltages
% vref      reference voltage
% vhi       high decision level (wide zero residue)
% vmid      mid decision level  (pure cyclic)
% vlo       low decision level  (wide zero)
%
% INTERNAL VARIABLES
% lodec,hidec                Comparator descisions
% compoutwz, compoutpc       Comparator outputs (-2, -1, 0, 1, 2)
% vreswz,vrespc              Residue voltages after cyclic operation
%
% OUTPUTS
% vouts 4XN Vector with PC and WZ residue voltages in rows 1 and 2
% and PC and WZ comparator decisions in rows 3 and 4
```

```
%
%

%%preallocate/clear
compouthi = zeros(1, size(vins,2));
compoutlo = zeros(1, size(vins,2));
vres(:,:) = zeros(5, size(vins,2));

if chooser(k)== 0
    % HIGH
    cyclein<vmid ;lodechi = ans;  %% should use simulation curve 2 [
       sim1]
    cyclein>vhi  ;hidechi2 = ans;  %% should use sim5
    middechi = ~lodechi&~hidechi2;                  %% else sim3

    compouthi = 2*hidechi2 - lodechi;  %%

    % Calculate Residue
    % HIGH
    vres(2,:) = interp1( dcrange ,sim(1,:), (cyclein - compouthi.*(
       vref/2)), 'spline') .* lodechi;
    vres(3,:) = interp1( dcrange ,sim(3,:), (cyclein - compouthi.*(
       vref/2)), 'spline') .* middechi;
    vres(4,:) = interp1( dcrange ,sim(5,:), (cyclein - compouthi.*(
       vref/2)), 'spline') .* hidechi2;

else
    % LOW
    %%%%%%%cyclein<vmid ;lowdecpc=ans;  %% Should use sim2   0
    cyclein<vlo  ; lowdec2lo=ans;  %% Should use sim1  -2
    cyclein>vmid ; hideclo=ans;   %% Should use sim4     1
    middeclo = ~lowdec2lo&~hideclo;   %% Should use sim2   0
    compoutlo = hideclo-2*lowdec2lo; %%

    % LOW
    vres(2,:) = interp1( dcrange ,sim(2,:), (cyclein - compoutlo.*(
       vref/2)), 'spline').* lowdec2lo;
    vres(3,:) = interp1( dcrange ,sim(2,:), (cyclein - compoutlo.*(
       vref/2)), 'spline').* middeclo;
    vres(4,:) = interp1( dcrange ,sim(4,:), (cyclein - compoutlo.*(
       vref/2)), 'spline').* hideclo;

end

vouts =[ vres(1,:)+ vres(2,:)+ vres(3,:)+ vres(4,:)+ vres(5,:) ;
   compouthi + compoutlo];
```

Appendix F

Decoder(Correction) Top Level

```
%% Decoder top level
%%%%% inputs
%%%     LUTa
%%%     LUTb
%%%     compouta
%%%     compoutb
%%%     m           %times decoded/ LUTs to use
%%%
%%%%% outputs
%%%     decoderresiduesa
%%%     decoderresiduesb
%%%

LUT = squeeze(LUTa(:,:,m));
%%% squeeze removes 'singletons' and sets lut to right size matrix
compout = compouta;
decodesingle
decoderresiduesa(:,:,m) = decoderresidues;

% Set up LUTb and compoutb
LUT = squeeze(LUTb(:,:,m));
%%% squeeze removes 'singletons' and sets lut to right size matrix
```

```
compout = compoutb;
decodesingle
decoderresiduesb(:,:,m) = decoderresidues;
```

Appendix G

Decoder(Correction)

```
%% Single ADC Decode(Correction)
% INPUTS
  % COMPOUT — comparator output from ADC
  % LUT     — ADC LUT
% OUPUTS
  % decoderresidues — final residue is corrected ADC output x

%%%%Preallocate and Clear variables
decoderresidues = zeros(ncycles + 1, size(vins,2));
decoded = zeros(1, size(vins,2));

%%%%%%% Use 0s for initial guess
resguess = zeros(size( compout(ncycles,:)));

% decode
for k = 1:ncycles
    decoded = zeros(1, size(vins,2));

if  k <1.5           %% using initial guess

    %% Determines which LUT to use or given vins/residues
         dec1 = find(compout(ncycles +1-k,:) == -2);
```

```
            dec2 = find(compout(ncycles +1-k,:) == -1);
            dec3 = find(compout(ncycles +1-k,:) == 0);
            dec4 = find(compout(ncycles +1-k,:) == 1);
            dec5 = find(compout(ncycles +1-k,:) == 2);

        % if compout(ncycles,:) == -2    %% use LUT1
    decoded(1, dec1)= -2.*(vref/2) + interp1(LUT(1,:), dclut,
        resguess(dec1), 'linear');

        % elseif compout(ncycles,:) == 0 %% USE LUT3
    decoded(1, dec3)= 0*(vref/2) + interp1(LUT(3,:), dclut, resguess
        (dec3), 'linear');

        % elseif compout(ncycles,:) == 2
    decoded(1, dec5)= 2*(vref/2) + interp1(LUT(5,:), dclut, resguess
        (dec5), 'linear');

        % elseif compout(ncycles,:) == -1
    decoded(1, dec2)= -1*(vref/2) + interp1(LUT(2,:), dclut,
        resguess(dec2), 'linear');

        % elseif compout(ncycles,:) == 1
    decoded(1, dec4)= 1*(vref/2) + interp1(LUT(4,:), dclut, resguess
        (dec4), 'linear');

    decoderresidues(1,:)= resguess; %% sets initial vres to guess

else    %% using previous residue voltage
            dec1 = find(compout(21-k,:) == -2);
            dec2 = find(compout(21-k,:) == -1);
            dec3 = find(compout(21-k,:) == 0);
            dec4 = find(compout(21-k,:) == 1);
            dec5 = find(compout(21-k,:) == 2);
        %%WZ
        decoded(1, dec1)= -2*(vref/2) + interp1(LUT(1,:), dclut ,
           decoderresidues(k,dec1), 'linear');
        decoded(1, dec3)= 0*(vref/2) + interp1(LUT(3,:), dclut  ,
           decoderresidues(k,dec3), 'linear');
        decoded(1, dec5)= 2*(vref/2) + interp1(LUT(5,:), dclut  ,
           decoderresidues(k,dec5), 'linear');

        %% PC
        decoded(1, dec4)= 1*(vref/2) + interp1(LUT(4,:),  dclut  ,
           decoderresidues(k,dec4), 'linear');
        decoded(1, dec2)= -1*(vref/2) + interp1(LUT(2,:), dclut  ,
           decoderresidues(k,dec2), 'linear');

    end

       decoderresidues(k+1,:)= decoded; %sets residue to
          interpolated values

end
```

Appendix H

Calibration Top Level

```
%% Calibration
%%%%% Calibrated LUTS
%%%%%%
%%%%% inputs
%%%     LUTa (m)
%%%     LUTb (m)
%%%     decoderresiduesa
%%%     decoderresiduesb
%%%
%%%%% outputs
%%%     LUTa (m+1)
%%%     LUTb (m+1)
%%%


%%%Adjust LutA
LUT = squeeze(LUTa(:,:,m));
compout = compouta;
decoderresidues = squeeze(decoderresiduesa(:,:,m));
splitdif = (squeeze(decoderresiduesa(21,:,m)) - squeeze(
    decoderresiduesb(21,:,m)));
LUT_correct8_7  %% determines average error per point and changes
    LUT
```

```
LUTa(:,:,m+1) = LUT; % sets LUTa to adjusted values

%%% Adjust LutB
LUT = squeeze(LUTb(:,:,m));
compout = compoutb;
decoderresidues = squeeze(decoderresiduesb(:,:,m));
splitdif = (squeeze(decoderresiduesb(21,:,m)) - squeeze(
    decoderresiduesa(21,:,m)));
LUT_correct8_7      %% determines average error per point and
    changes LUT
LUTb(:,:,m+1) = LUT;  % sets LUTb to adjusted values
```

Appendix I

LUT Calibration

```
%% PREALLOCATE
lut1errstore=zeros(3, size(LUT(1,:),2)); %% stores cumulative error
    for each pt on lut
lut2errstore=zeros(3, size(LUT(1,:),2));
lut3errstore=zeros(3, size(LUT(1,:),2));
lut4errstore=zeros(3, size(LUT(1,:),2));
lut5errstore=zeros(3, size(LUT(1,:),2));

lut1errct= zeros(3, size(LUT(1,:),2));   %% counts # of times LUT pt
     is used
lut2errct= zeros(3, size(LUT(1,:),2));
lut3errct= zeros(3, size(LUT(1,:),2));
lut4errct= zeros(3, size(LUT(1,:),2));
lut5errct= zeros(3, size(LUT(1,:),2));

lutposition = zeros(size(vins,2),3);    %%LUT POINT for refpoint

weight = zeros(3, size(LUT(1,:),2),5);

clear dec1 dec2 dec3 dec4 dec5

%% Accumulated Error
```

```
%  refpoint — where APPROX on LUT the error is for vres(out)~
    decoderreidues
%  error VOLTAGE for 3 cycles
 refpoint(1,:) = decoderresidues(21, :) - compout(1, :).*(vref/2);
 refpoint(2,:) = decoderresidues(20, :) - compout(2, :).*(vref/2);
 refpoint(3,:) = decoderresidues(19, :) - compout(3, :).*(vref/2);

% refpointa(1,:) = decoderresidues(21, :) — compout(1, :).*(vref/2);
% refpointb(1,:) = decoderresidues(20, :) — compout(2, :).*(vref/2);
% refpointc(1,:) = decoderresidues(19, :) — compout(3, :).*(vref/2);

%%
for k = 1:3  % number of cycles back to track error
    % resdif — difference between the ideal LUT output and the error
        LUT
    % output
    % resdif(k,:) = decoderresidues2(21, :) — decoderresiduesi(21,
       :);

         dec1 = find(compout(k,:) == -2);   %% Determines which LUT
            to use
         dec2 = find(compout(k,:) == -1);   %% for vins/residues
         dec3 = find(compout(k,:) == 0);
         dec4 = find(compout(k,:) == 1);    %% sum of 5 dec should =
             size(vins)
         dec5 = find(compout(k,:) == 2);

    for i = dec1  %% for pts using LUT1
        if find(refpoint(k,i)<= dclut, 1) %%% 'if' corrects for '
           Subscripted assignment dimension mismatch.'
                                          %%%  in lutposition(i,k)
                                             when find returns empty
```

```
                                        matrix
lutposition(i,k) = find(refpoint(k,i)<= dclut , 1); %
   find point on DC input of LUT


%          *  dclut(lutposition(i,k))   (B)
%         /
%        X − refpoint
%       /
%      /
%    *  dclut(lutposition(i,k)−1)  (A)
%


% weight(A) = weightprev. + (1−(refpoint − A)/ ( B − A))
% weight(B) = weightprev. + ((refpoint − A)/ ( B − A))



w  = (refpoint(k,i)- dclut(lutposition(i,k)-1))/(dclut(
   lutposition(i,k)) - dclut(lutposition(i,k)-1));     %
    weights
w2 = 1 - (refpoint(k,i)- dclut(lutposition(i,k)-1))/(
   dclut(lutposition(i,k)) - dclut(lutposition(i,k)-1));
    % weights

weight(k,lutposition(i,k),1) = weight(k,lutposition(i,k)
   ,1) + w;
weight(k,lutposition(i,k)-1,1) = weight(k,lutposition(i,
   k)-1,1) + w2;

%lut1errstore(k,lutposition(i,k))    =  lut1errstore(k,
   lutposition(i,k)) +  w * splitdif(1,i);  % cumulative
    error
```

```
            %lut1errstore(k,lutposition(i,k)−1)  =  lut1errstore(k,
               lutposition(i,k)−1) +  w2* splitdif(1,i);  %

             lut1errstore(k,lutposition(i,k))    =  lut1errstore(k,
                lutposition(i,k)) +  splitdif(1,i);  % cumulative
                error
             lut1errstore(k,lutposition(i,k)-1)  =  lut1errstore(k,
                lutposition(i,k)-1) +  splitdif(1,i);  %

             lut1errct(k,lutposition(i,k))= lut1errct(k,lutposition(
                i,k)) + 1;
             lut1errct(k,lutposition(i,k)-1)= lut1errct(k,
                lutposition(i,k)-1) + 1; % counts number of times
                LUT pt is used
        end
    end

    for i = dec2
        if find(refpoint(k,i)<= dclut , 1)
            lutposition(i,k) = find(refpoint(k,i)<= dclut , 1); %
               find spot on DC input of LUT

            w  = (refpoint(k,i)- dclut(lutposition(i,k)-1))/(dclut(
               lutposition(i,k)) - dclut(lutposition(i,k)-1));
            w2 = 1 - (refpoint(k,i)- dclut(lutposition(i,k)-1))/(
               dclut(lutposition(i,k)) - dclut(lutposition(i,k)-1));

            weight(k,lutposition(i,k),2) = weight(k,lutposition(i,k)
               ,2) + w;
            weight(k,lutposition(i,k)-1,2) = weight(k,lutposition(i,
               k)-1,2) + w2;
            lut2errstore(k,lutposition(i,k))   = lut2errstore(k,lutposition(i,k)) +  splitdif(1,i); % cumulative error
            lut2errstore(k,lutposition(i,k)-1) = lut2errstore(k,lutposition(i,k)-1) + splitdif(1,i);  %

             %lut2errstore(k,lutposition(i,k))    = lut2errstore(k,lutposition(i,k)) + w* splitdif(1,i);  % cumulative error
             %lut2errstore(k,lutposition(i,k)−1)  = lut2errstore(k,lutposition(i,k)−1) + w2* splitdif(1,i);  %

             lut2errct(k,lutposition(i,k))= lut2errct(k,lutposition(i,k)) + 1;
            lut2errct(k,lutposition(i,k)-1)= lut2errct(k,lutposition(i,k)-1) + 1;
        end
    end

    for i = dec3
        if find(refpoint(k,i)<= dclut , 1)
            lutposition(i, k) = find(refpoint(k,i)<= dclut , 1); % find spot on DC input of LUT

            w  = (refpoint(k,i)- dclut(lutposition(i,k)-1))/(dclut(lutposition(i,k)) - dclut(lutposition(i,k)-1));
            w2 = 1 - (refpoint(k,i)- dclut(lutposition(i,k)-1))/(dclut(lutposition(i,k)) - dclut(lutposition(i,k)-1));

              weight(k,lutposition(i,k),3) = weight(k,lutposition(i,k),3) + w;
            weight(k,lutposition(i,k)-1,3) = weight(k,lutposition(i,k)-1,3) + w2;
            lut3errstore(k,lutposition(i,k))    =  lut3errstore(k,lutposition(i,k)) +  splitdif(1,i);  % cumulative error
            lut3errstore(k,lutposition(i,k)-1)  =  lut3errstore(k,lutposition(i,k)-1) +  splitdif(1,i);

            %(refpoint(k,i)- dclut(lutposition(i,k)-1))/(dclut(lutposition(i,k)) - dclut(lutposition(i,k)-1))
            % lut3errstore(k,lutposition(i,k))    =  lut3errstore(k,lutposition(i,k)) +  w*splitdif(1,i);  % cumulative error
            % lut3errstore(k,lutposition(i,k)-1)  =  lut3errstore(k,lutposition(i,k)-1) + w2* splitdif(1,i);
            lut3errct(k,lutposition(i,k))= lut3errct(k,lutposition(i,k)) + 1;
           lut3errct(k,lutposition(i,k)-1)= lut3errct(k,lutposition(i,k)-1) + 1;
        end
    end

    for i = dec4
        if find(refpoint(k,i)<= dclut, 1)
            lutposition(i, k) = find(refpoint(k,i)<= dclut, 1); % find spot on DC input of LUT

            w  = (refpoint(k,i)- dclut(lutposition(i,k)-1))/(dclut(lutposition(i,k)) - dclut(lutposition(i,k)-1));
            w2 = 1 - (refpoint(k,i)- dclut(lutposition(i,k)-1))/(dclut(lutposition(i,k)) - dclut(lutposition(i,k)-1));

            weight(k,lutposition(i,k),4) = weight(k,lutposition(i,k),4) + w;
            weight(k,lutposition(i,k)-1,4) = weight(k,lutposition(i,
               k)-1,4) + w2;

            lut4errstore(k,lutposition(i,k))    =  lut4errstore(k,
               lutposition(i,k)) +  splitdif(1,i);  % cumulative
               error
            lut4errstore(k,lutposition(i,k)-1)  =  lut4errstore(k,
               lutposition(i,k)-1) +   splitdif(1,i);  %

            % lut4errstore(k,lutposition(i,k))    =  lut4errstore(k,
               lutposition(i,k)) + w* splitdif(1,i);  % cumulative
               error
            % lut4errstore(k,lutposition(i,k)-1)  =  lut4errstore(k,
               lutposition(i,k)-1) + w2*  splitdif(1,i);  %

            lut4errct(k,lutposition(i,k))= lut4errct(k,lutposition(i
               ,k)) + 1;
            lut4errct(k,lutposition(i,k)-1)= lut4errct(k,lutposition
               (i,k)-1) + 1;
        end
    end

    for i = dec5
        if find(refpoint(k,i)<= dclut, 1)
            lutposition(i, k) = find(refpoint(k,i)<= dclut, 1); %
               find spot on DC input of LUT

            w  = (refpoint(k,i)- dclut(lutposition(i,k)-1))/(dclut(
               lutposition(i,k)) - dclut(lutposition(i,k)-1));
            w2 = 1 - (refpoint(k,i)- dclut(lutposition(i,k)-1))/(
               dclut(lutposition(i,k)) - dclut(lutposition(i,k)-1));
            weight(k,lutposition(i,k),5) = weight(k,lutposition(i,k)
               ,5) + w;
            weight(k,lutposition(i,k)-1,5) = weight(k,lutposition(i,
               k)-1,5) + w2;

            lut5errstore(k,lutposition(i,k))    =  lut5errstore(k,
               lutposition(i,k)) +   splitdif(1,i);  % cumulative
               error
            lut5errstore(k,lutposition(i,k)-1)  =  lut5errstore(k,
               lutposition(i,k)-1) +  splitdif(1,i);  %

            %lut5errstore(k,lutposition(i,k))    =  lut5errstore(k,
               lutposition(i,k))   + w* splitdif(1,i);  % cumulative
                error
            %lut5errstore(k,lutposition(i,k)-1)  =  lut5errstore(k,
               lutposition(i,k)-1) + w2* splitdif(1,i);  %

            lut5errct(k,lutposition(i,k))= lut5errct(k,lutposition(i
               ,k)) + 1;
            lut5errct(k,lutposition(i,k)-1)= lut5errct(k,lutposition
               (i,k)-1) + 1;
        end
    end

end


%% LUT Adjustment!!!!

k=1; % Adjust final cycle weight

for i = 1:size(LUT(1,:),2)
```

```
    if lut1errct(k,i) %% corrects for 'Subscripted assignment
       dimension mismatch.'
    LUT(1,i) = LUT(1,i) +  (lut1errstore(k,i)/lut1errct(k,i)) .* mu1
       ;  % corrects LUT pt
    end
    if lut2errct(k,i)
    LUT(2,i) = LUT(2,i) +  (lut2errstore(k,i)/lut2errct(k,i)) .* mu1
       ;
    end
    if lut3errct(k,i)
    LUT(3,i) = LUT(3,i) +  (lut3errstore(k,i)/lut3errct(k,i)) .* mu1
       ;
    end
    if lut4errct(k,i)
    LUT(4,i) = LUT(4,i) +  (lut4errstore(k,i)/lut4errct(k,i)) .* mu1
       ;
    end
    if lut5errct(k,i)
    LUT(5,i) = LUT(5,i) +  (lut5errstore(k,i)/lut5errct(k,i)) .* mu1
       ;
    end
end

LUTfirst = LUT;

k=2; % Adjust second to last cycle weight

for i = 1:size(LUT(1,:),2)
    if lut1errct(k,i)
    LUT(1,i) = LUT(1,i) +  (lut1errstore(k,i)/lut1errct(k,i)) .* (
       mu1/10);
    end
    if lut2errct(k,i)
    LUT(2,i) = LUT(2,i) +  (lut2errstore(k,i)/lut2errct(k,i)) .* (
       mu1/10);
    end
    if lut3errct(k,i)
    LUT(3,i) = LUT(3,i) +  (lut3errstore(k,i)/lut3errct(k,i)) .* (
       mu1/10);
    end
    if lut4errct(k,i)
    LUT(4,i) = LUT(4,i) +  (lut4errstore(k,i)/lut4errct(k,i)) .* (
       mu1/10);
    end
    if lut5errct(k,i)
    LUT(5,i) = LUT(5,i) +  (lut5errstore(k,i)/lut5errct(k,i)) .* (
       mu1/10);
    end
end

LUTsecond = LUT;
```

Bibliography

[1] A. J. Annema, *Analog circuit performance and process scaling*, IEEE Custom Intergrated Circuits Conference, 1999, pp. 301–304.

[2] P. J. Hurst C. R. Grace and S. H. Lewis, *A 12-bit 80-msample/s pipelined adc with bootstrapped digital calibration*, IEEE Journal of Solid-State Circuits **40** (2005), 1038 – 1046.

[3] M. Demler, *High-speed analog-to-digital conversion*, Academic Press, Inc., 1991.

[4] C. Tsang et al., *Background adc calibration in digital domain*, IEEE Custom Intergrated Circuits Conference, 2008, pp. 301–304.

[5] W. Press et al., *Numerical recipes*, Cambridge University Press, 1990.

[6] M. Coln J. McNeill and B.Larivee, *Split adc architecture for deterministic digital background calibration of a 16-bit 1-ms/s adc*, IEEE Journal of Solid-State Circuits **40** (2005), no. 12, 2437–2445.

[7] L. Komzsik, *Approximation techniques for engineers*, Taylor and Francis Group, 2007.

[8] H. Kwakernaak and R. Sivan, *Modern signal and systems*, Prentice Hall, 1991.

[9] J.A. McNeill, M.C.W. Coln, D.R. Brown, and B.J. Larivee, *Digital background-calibration algorithm for split adc architecture*, Circuits and Systems I: Regular Papers, IEEE Transactions on **56** (2009), no. 2, 294–306.

[10] J.A. McNeill, C. David, M. Coln, and R. Croughwell, *split adc calibration for all-digital correction of time-interleaved adc errors*, Circuits and Systems II: Express Briefs, IEEE Transactions on **56** (2009), no. 5, 344–348.

[11] B. Murmann, *A 12b 75ms/s pipelined adc using open-loop residue amplification*, 2003, Berkleley Wireless Research Seminar.

[12] ———, *Digital calibration for low-power high-performance a/d conversion*, Ph.D. thesis, University of California, Berkeley, Fall 2003.

[13] B. Murmann and B. E. Boser, *A 12b 75ms/s pipelined adc using open-loop residue amplification*, ISSCC Dig. Tech. Papers, Feb. 2003.

[14] A. Panigada and I. Galton, *Digital background correction of harmonic distortion in pipelined adcs*, IEEE Journal of Solid-State Circuits, May 2005.

[15] G. Rrudho S. Orchanian and A. Soares Jr., *Fundamental blocks for a cyclic analog-to-digital converter*, Worcester Polytechnic Institute MQP, 2009.