2015-04-30

Lookup-Table-Based Background Linearization for VCO-Based ADCs

Long Pham
Worcester Polytechnic Institute

Lookup-Table-Based Background Linearization for VCO-Based ADCs

by

Long Pham

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

Long

May 2015

APPROVED:

[illegible]

Professor John A. McNeill, Major Thesis Advisor

[illegible]

Professor Thomas Eisenbarth, ECE Department, WPI

Abstract

Scaling of CMOS to nanometer dimensions has enabled dramatic improvement in digital power efficiency, with lower V_{DD} supply voltage and decreased power consumption for logic functions. However, most traditionally prevalent ADC architectures are not well suited to the lower V_{DD} environment. The improvement in time resolution enabled by increased digital speeds naturally drives design toward time-domain architectures such as voltage-controlled-oscillator (VCO) based ADCs. The major obstacle in the VCO-based technique is linearizing the VCO voltage-to-frequency characteristic. Achieving signal-to-noise (SNR) performance better than -40dB requires some form of calibration, which can be realized by analog or digital techniques, or some combination [1, 2, 9]. A further challenge is implementing calibration without degrading energy efficiency performance. This thesis project discusses a complete design of a 10 bit three stage ring VCO-based ADC. A lookup-table (LUT) digital correction technique enabled by the "Split ADC" calibration approach is presented suitable for linearization of the ADC. An improvement in the calibration algorithm compared to [1, 2] is introduced to ensures LUT continuity. Measured results for a 10 bit 48.8-kSps ADC show INL improvement of $\approx 10X$ after calibration convergence.

Acknowledgements

I would like to express the deepest appreciation to my thesis advisor, Professor John McNeill, for giving me the opportunity to work with him. He has been the most inspiring and motivating advisor I have had in my life. His understanding, wisdom, guidance and encouragement has pushed me further than I thought I could go in this project.

I would also like to thank Jianping Gong, PhD student in the NECAMSID lab, for discussing the project with me and for making my experience at the lab more fun and exciting. Also to Sulin Li who supported my idea and is continuing to further explore the topic of my project.

Thanks also to Robert Boisse for teaching me how to do the surface mount soldering. Without his help, the printed circuit board would have not been completed.

Last but not least, I want to thank my family, my girl friend and my roommates for helping me survive all the stress in my college life and not letting me give up.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Goals and Motivation

The analog-to-digital converter (ADC) plays an important role as a bridge between the inherently analog world and ever-increasing digital processing world. Ultra-low-power ADCs are needed in systems constrained by battery power or scavenged energy limits in applications such as wireless communication, autonomously powered sensing and monitoring nodes, or implanted biomedical devices for assistive technology. The ADC energy efficiency expressed by the fJ/step figure-of-merit is a critical design system driver in power-constrained applications. When pushing for increasing data rates there is a corresponding increase in the demands of bandwidth and power dissipation [4].

The advance in CMOS technologies has dramatically improved the performance of general purpose processors and digital signal processors. However, most traditionally prevalent ADC architectures have not been able to utilize the process scaling to the same extent, suffering from reduced voltage headroom and reduced analog gain. The improvement in time resolution enabled by increased digital speeds naturally drives design toward time-domain architectures such as voltage-controled-oscillator (VCO) based ADCs. A major difficulty with this approach is that ADC linearity depends directly on the linearity of the VCO voltage-to-frequency control characteristic, which is in general poorly controlled.

Efforts have been made to improve performance of VCO-based ADC. One could be to place the VCO-based qunatizer within a continuous-time $\Sigma\Delta$ loops where the high loop gain suppressed non-linearity and phase noise, such as [5, 6, 7, 8, 9]. This approach, however, still required op-amp based integrators or additional DACs and was thus very analog in nature. Another would be to use a highly linear current controlled oscillator (ICO) architecture as shown in [8]. However without the support of a $\Sigma\Delta$ modulator and a feedback DAC, the linearity of the VCO can only be improved to maximum 7.4 bits.

This thesis project describes a complete design of a 10 bit VCO-based ADC including an implementation of a lookup table (LUT) linearization technique suitable for that ADC. A three stage ring VCO is constructed and the digital output of the ADC can be obtained by measuring the VCO output clock frequency. The "Split ADC' approach [1, 2, 3, 11] is applied to realize continuous digital background calibration. All the calibration and correction process is performed entirely in the digital domain by a field-programmable gate array (FPGA) board.

1.2 Organization

This thesis is organized as follows: Chapter 2 provides the background on ADC characteristics and nonideal behavior. The VCO-based ADC is also investigated, followed by different VCO architectures. Chapter 3 reviews the background calibration and correction technique in details, as well as introduces the improved algorithm to preserve LUT continuity. Chapter 4 presents the hardware implementation in a print circuit board (PCB) including the ring VCOs and frequency dividers. Chapter 5 discusses the FPGA implementation of the calibration technique. All the inputs, outputs and timing configurations of each digital blocks are explained. The result measurements of the ADC are provided in Section 6. Finally, Chapter 7 concludes the research work presented here and provides possible paths for future investigation.

Chapter 2

Background

This section firstly provides background information on analog to digital converters including ideal characteristics and performance metrics. It then goes on to discuss the VCO-based ADC architecture. Finally, different VCO architectures are investigated in the last subsection.

2.1 Analog to Digital Converter (ADC)

Analog to digital converters (ADCs) translate analog quantities, which are the inherent characteristic of "real world" signals, to digital language used in various applications including computing, information processing and control systems. The relationship between inputs and outputs of ADCs is shown in Figure 2.1. The ADC takes an analog voltage as an input and returns a unique group of digital levels, or binary codes, corresponding to each analog level.

Before designing an ADC, it is necessary to understand the performances and specifications of the ADC. The following subsections firstly describe the ADC transfer function, followed by discussions on different sources of error of ADCs.

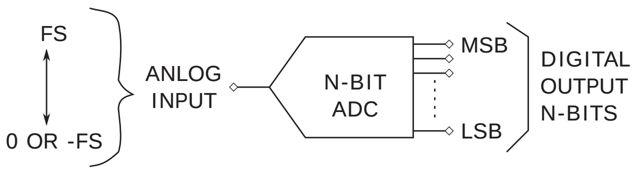


Figure 2.1: ADC Input and Output Definitions

2.1.1 Ideal ADC Characteristics

During the conversion process, the analog input signal is quantized to a digital value. The resolution of an analog to digital converter describes how many quantization levels the ADC can represent. Since the output of an ADC is in binary format, the resolution is given in powers of 2. For example, a 10-bit A/D converter can represent an analog signal using 2^{10} or 1024 quantization levels. Figure 2.2 shows the ideal transfer characteristics of an ADC. It is important to note that while the analog voltage is continuous, the digital output is quantized to certain levels.

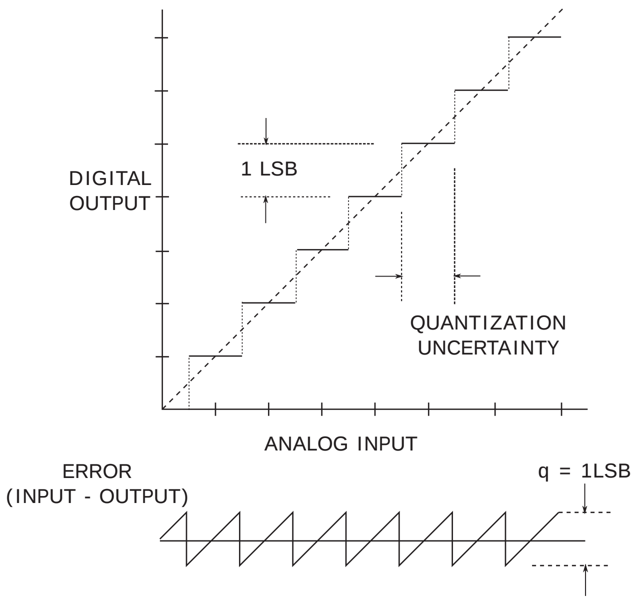


Figure 2.2: Ideal ADC Transfer Function and Quantization Noise

The only error mechanism present in an ideal ADC is quantization. This error arises because the analog input signal may assume any value within the input range of the ADC while the output data is a sequence of finite precision samples [4]. As Figure 2.2 shown, the quantization error for any signals that spans more than few LSBs can be modeled as a sawtooth waveform [12]. The maximum error an ideal converter makes when digitizing a signal is $0.5V_{LSB}$. Assuming the quantization noise is uniformly distributed over the range $\pm 0.5V_{LSB}$, the root-mean-square quantization error can be calculated as:

$$\text{(RMS)}V_q \approx \frac{V_{LSB}}{\sqrt{12}} \tag{2.1}$$

For a N bit fullscale analog input sinewave, $V_{in} = 2^{N-1}V_{LSB}\sin(2\pi ft)$, swinging from $-2^{N-1}V_{LSB}$ to $2^{N-1}V_{LSB}$, the RMS value is expressed as:

$$\text{(RMS)}V_{FS} = \frac{2^{N-1}V_{LSB}}{\sqrt{2}} \tag{2.2}$$

Therefore, the signal-to-noise ratio for an ideal N-bit converter is

$$\text{SNR} = 20\log_{10}\left[\frac{(RMS)V_{FS}}{(RMS)V_q}\right] = 6.02N + 1.76dB \tag{2.3}$$

It is important to note that, Equation 2.3 assumes that the quantization noises and the input signal are uncorrelated and are both measured over the full Nyquist bandwidth. In the case that the actual signal occupies in a smaller bandwidth, a correction factor must be included when calculating the signal-to-noise-ratio [12].

2.1.2 Static Errors

There are four possible sources of DC errors associated with ADCs including offset error, gain error, differential nonlinearity (DNL) and integral nonlinearity (INL). Unlike the inevitable quantization noise, these error sources only occurs in non-ideal

ADCs. As described in Figure 2.2, the dashed straight line joining the midpoints of all the “steps” often refered as the “code centers”, plays an important role in determining the static errors of the ADCs. This ideal transfer function of an ADC might be expressed as a straight line $y = Ax + b$, where y is the digital code, x is the analog input voltage, A and b are two constants. The gain and offset errors are defined as the deviations between the actual gain A and offset b and the ideal values, respectively. However, since “real world” ADC characteristic might not be a straight line, the two types of linearity error are used when investigating the ADC performance.

Figure 2.3 shows linearity errors of ADCs. The integral nonlinearity (INL) is defined as maximum deviation of the actual transfer characteristic of the converter from a straight line. This straight line can be either a best straight line which is drawn so as to minimize these deviations or it can be a line drawn between the end points of the transfer function once the gain and offset errors have been nullified. Differential nonlinearity (DNL) is the difference between the step size of an ADC’s output and the ideal step size. The DNL and INL are usually measured in terms of least significant bit (LSB). In the case where DNL equals to -1 or $+1$, the ADC is nonmonotonic or has missing codes [12].

There are many possible methods to test the linearity performances of an ADC. One could be to directly measure the the code transitions of the analog input voltage while observing the digital outputs. However, this method requires a large amount of measurements and only works well if ADC’s input referred noise is less than 1 LSB [12]. Another approach is the back-to-back static test which captures the error waveform by comparing the analog input to the digital output through a feedback path. The main difficulty with this approach is that it requires an additional digital to analog converter which must have an accuracy significantly greater than the ADC

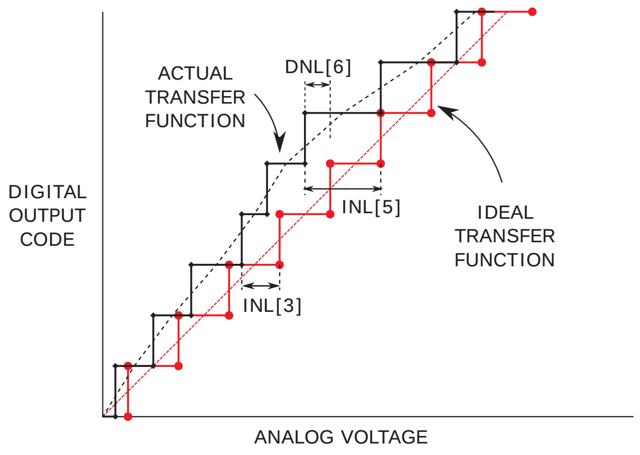


Figure 2.3: Nonlinear Static Errors in Nonideal ADCs

under test [12]. The servo-loop code transition test and computer-based servo-loop test lend themselves to automated measurements, either ATE systems or in PC-based controller. These method's complexity goes beyond the scope of this project.

Histogram (code density) test with linear ramp input is the most suitable method in testing the linearity performance of the ADC for this project. It involves collecting a large number of digitized samples over a period of time for a well-defined input signal which is usually a low frequency fullscale triangular waveform. The number of occurrences, $h(n)$, are then recorded for each code bin. Ideally, the number of hits in each bins are equal and can be calculated based on the total number of output samples M and total number of bins $2^N - 1$.

$$h(n) = \frac{M}{2^N - 2} \tag{2.4}$$

If measured histogram indicates the actual number of hits in a bin is $h(n)_{actual}$, then the DNL can be calculated as:

$$DNL(n) = \frac{h(n)_{actual}}{h(n)_{theoretical}} - 1 \tag{2.5}$$

Since the INL measures the nonlinearity of the overall transfer function, it is simply the cumulative sum of the DNL.

$$INL(n) = \sum_{i=0}^{n} DNL(n) \tag{2.6}$$

Figure 2.4 shows an example of a histogram test of a four bit ADC. Note that, the histogram test alone does not imply monotonicity in an ADC. Additionally, in order to eliminate the linearity due to the input voltage source, the linear input ramp used in the histogram test must have greater precision compared to the ADC under test [12].

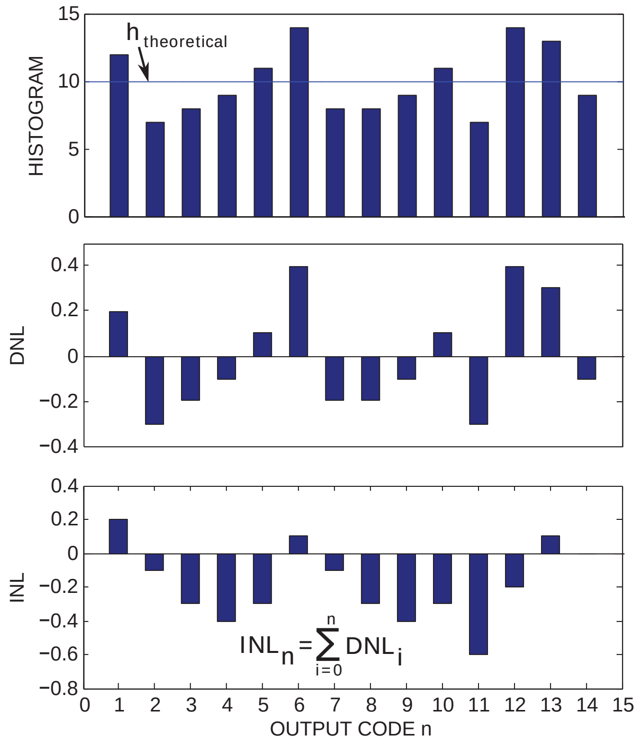


Figure 2.4: Histogram testing of a 4 bit ADC

2.1.3 Dynamic Errors

The fact that DNL and INL meet the system requirements does not implies that the DAC will perform well for AC input signals. There are several ways to characterize the dynamic performance of an ADC. An FFT analysis is often used to measure the AC distortion of the signal. From that, three important parameters are defined including Signal-to-Noise-and-Distortion ratio (SNDR), Signal-to-Noise ratio (SNR), spurious-free dynamic range (SFDR), and effective number of bits (ENOB).

SINAD is defined as the ratio of the RMS signal amplitude to the mean value of the root-sum-squares of all other noise components. SNR is similar to SINAD except that it does not include the harmonic content which occurs at multiples of signal frequency. Therefore, SNR can reveal the noise floor, which ideally only includes the quantization noise. As discussed in Section 2.1.1 the ideal SNR is directly related to the resolution of the ADC. In a similar manner, the effective number of bits is defined as:

$$ENOB = \frac{SINAD - 1.76dB}{6.02} \tag{2.7}$$

It should be noted that SINAD and ENOB are functions of the input signal frequency. As frequency increases toward the Nyquist limit, SINAD decreases; so does ENOB.

Another significant specification for an ADC is the Spurious Free Dynamic Range (SFDR). SFDR is the ratio of the RMS value of an input sine wave to the RMS value of the largest spur observed in the frequency domain. Figure 2.5 shows an example of SFDR specification for an ADC. As the figure illustrates, the SFDR could be calculated either based on the amplitude of the carrier signal (dBc) or with respect to the full scale amplitude (dBFS).

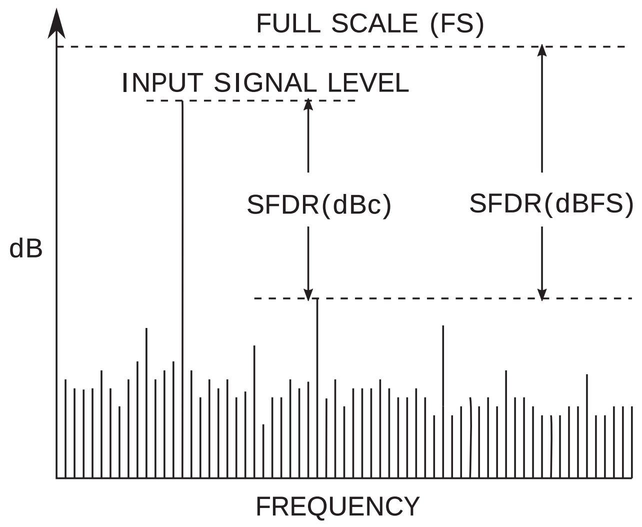


Figure 2.5: Spurious Free Dynamic Range

2.2 VCO-Based ADCs

There are a wide variety of different ADC architectures available depending on the requirements of the application. They can range from high-speed, low resolution flash converters to the high-resolution, low-speed oversampled noise-shaping sigma-delta converters. Scaling of CMOS to nanometer dimensions has enabled dramatic improvement in digital power efficiency, with lower V_{DD} supply voltage and decreased power consumption for logic functions. This drives ADCs toward Voltage-Controlled-Oscillator-Based ADC which takes advantage of the high speed performance and low power consumption of logic circuits. This section firstly describes structure of a VCO-based ADC, then discusses the properties, followed by the nonideality associated with it.

2.2.1 VCO-Based-ADC Architecture

A VCO-based ADC is a time-based architecture that converts an analog input to frequency which is then quantized to digital output by other digital circuitry. Figure

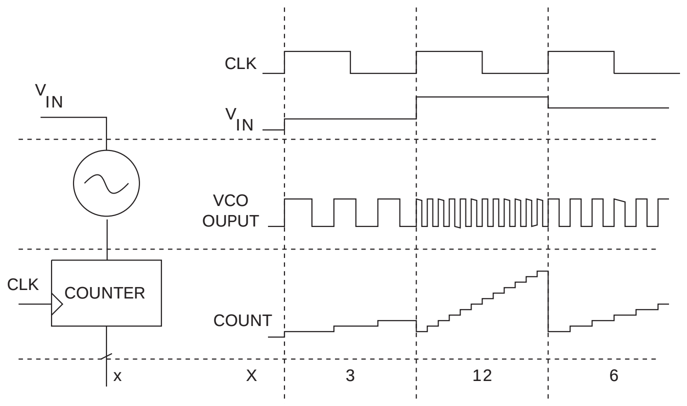


Figure 2.6: Simplified VCO-Based ADC

2.6 shows the VCO-based approach in its simplest form: the ADC input V_{IN} is the control input to the VCO; the output of the VCO is a digital clock. The digital output x of the ADC can be obtained by measuring the clock frequency in the digital domain, for example by a counter recording the number n of VCO clock phase edges in a given period of time. At the end of every clock period the output of the counter is sampled and then reset to zero.

In order to increase the resolution of the VCO-based ADC, multiple phase outputs of the VCO are used together as shown in Figure 2.7 [10, 17]. In this structure, the output of each stage is an oscillating waveform which has frequency given by:

$$f_{VCO} = K_{VCO} V_{IN} \tag{2.8}$$

where K_{VCO} is the slope of the voltage-to-frequency characteristic of the VCO, and V_{IN} is the analog voltage. The digital output of the ADC is resulted from summing the number of phase transitions n in a fixed period T_{CONV} of every stage. Therefore, for an N-stage ring VCO, the digital output is:

$$n = 2N f_{VCO} T_{CONV} = (2N K_{VCO} T_{CONV}) V_{IN} \tag{2.9}$$

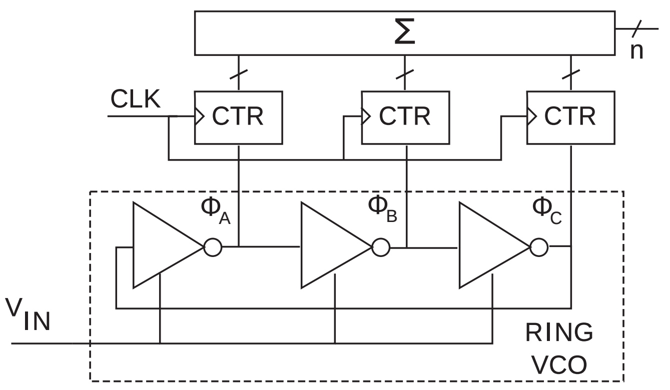


Figure 2.7: Multi-phase VCO-Based ADC Architecture

As Equation 2.9 indicates, ideally, the digital output is directly proportional to analog input voltage, which is the desired characteristic of an ADC. For an N-phase VCO-based ADC, the resolution of the ADC is higher by a factor of 2N compared to that of a single-phase converter. The multi-phase approach does require a modest increase in complexity, power consumption and chip area [17].

2.2.2 VCO-Based ADC Properties

The VCO-based ADC has several important properties making it suitable for high speed and low power consumption signal processing applications. The first property is that quantization noise is first-order noise-shaped [10]. The quantization error of the VCO-based ADC is not as straight forward as conventional ADC architectures. Since the number of cycles of each the clock signal is counted, the phases are quantized by 2π. If both the rising edges and the falling edges of the oscillated waveform are counted, the phases are quantized by π. A signal waveform capturing the phase of the VCO output is shown in Figure 2.8. Since the residual phase (quantization error $\phi_q[n-1]$) of the previous sampling period inherently becomes the initial phase $\phi_i[n]$ of the next period, the output of a N-phase VCO-based ADC can be calculated by:

$$y[n] = \frac{N}{2\pi}(\phi_x[n] + \phi_q[n-1] - \phi_q[n]) \tag{2.10}$$

where $\phi_x[n]$ is the VCO phase change due to analog inpute. Taking the Z-transform of Equation 2.10 gives:

$$Y(z) = \frac{N}{2\pi}(\Phi_x(z) + \Phi_q(z)(z^{-1} - 1)) \tag{2.11}$$

As Equation 2.11 indicates, the quantization phase noise $\phi[n]$ "sees"' a high pass filter transfer function $(z^{-1} - 1)$. Therefore, the VCO-based ADC is first-order noise-shaped.

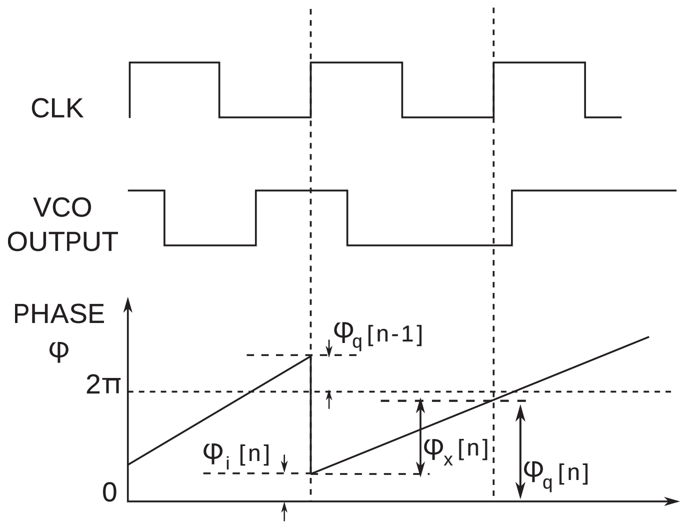


Figure 2.8: Quantization of phase in VCO-based ADC

Another property that makes VCO-based ADC superior to other architectures is that the output of a ring VCO is digital in nature. The clock signal toggles between two discrete levels, either V_{DD} or GND. This property makes the VCO a great converter building block which takes advantage of the high performance nanometer CMOS technology without worrying about the decreasing power supply. In addition, since the amplitude of the VCO output does not play an essential role in the quantization process, the architecture greatly reduces the need of additional analog circuitries such as buffers and amplifiers.

2.2.3 VCO-Based ADC Nonideality

Despite of these attractive properties discussed in Section 2.2.2, implementing VCO-based ADC still faces a critical challenge, since the voltage-to-frequency tuning curve of the VCO is usually nonlinear. This translates directly to the nonlinearity of the ADC, which highly degrades both static and AC performance of the ADC. Mitigating this effects of the VCO characteristics becomes the main subject of this project.

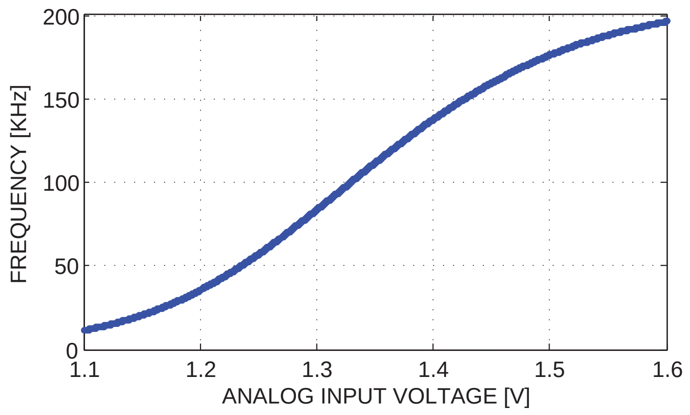


Figure 2.9: A 3-stage VCO voltage-to-frequency characteristic

Figure 2.9 captures the voltage-to-frequency characteristic of a three stage current starved VCO. The figure shows that the V-f curve of the VCO is nonlinear with an offset and a varied slope $K_{VCO} = \partial f_{VCO}/\partial V_{in}$. Although the offset of the VCO can easily be corrected by subtracting a fixed amount to the digital output code, some form of calibration to correct the nonlinear slope K_{VCO} is required in order to achieve a SNR better than $40dB$ [1, 2, 9].

2.3 VCO architectures

This section briefly discusses some of the available VCO architectures. One of the oscillator topologies is the LC oscillator. Although this topology out-performs ring oscillators in terms of phase noise [14, 17], it is analog in nature and requires area-consuming passive elements; therefore, it is not considered further in this project.

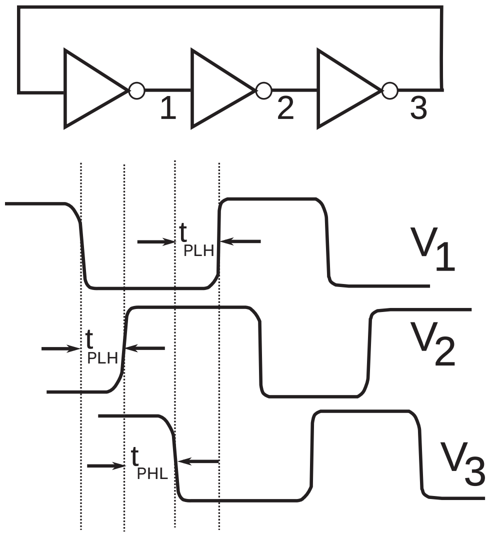


Figure 2.10: Waveforms of a three stage single ended ring VCO

Another common type of oscillator is the ring oscillator. The simplest ring oscillator can be formed by an odd number of inverters connected in a closed loop with positive feedback as shown in Figure 2.10. The delay time between the 50% points of the input and output are labeled t_{PLH} and t_{PHL} depending on whether the output logic level is changing from high to low or from low to high. If the total number of stages is n and all of them have the same delay, the oscillation frequency is then given by:

$$f = \frac{1}{n(t_{PHL} + t_{PLH})} \tag{2.12}$$

The number of stages n in a ring oscillator is determined by various requirements, including speed, power consumption, and noise immunity. Usually, there must be an odd number of inversions in the loop so that the circuit does not latch up. However, the differential implementation of the ring oscillator can utilize an even number of stages by simply configuring one stage such that it does not invert [14].

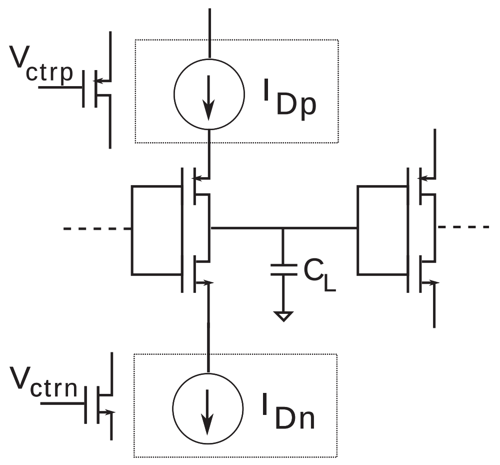


Figure 2.11: Simplified Current Starved VCO

As shown in Equation 2.12 the oscillation frequency depends on the total delay of each stage. Thus, to vary the frequency, the delay time can be adjusted. There are several ways to control this delay. The first method is to control the current drive strength charging and discharging the load of each inverter. This topology is referred as the current-starved VCO. Figure 2.11 shows a simplified view of a single stage of the current-starved VCO. The two additional MOSFETs, controlled by the input voltage V_{CTR}, are used to limit the drain currents to the inverter; in other word, the inverter is starved for current. The total sum of the output capacitor of the first stage and the input capacitor of the second stage can be modeled as a load capacitor C_L. The time it takes to charge and discharge C_L depends on the current I_{Dp} and I_{Dn}, respectively. These time delays are the same as the propagation delays t_{PHL} and t_{PLH} in Equation 2.12. Therefore, by controlling either the current I_{D1} or I_{D2} the frequency can be changed.

Another method is to vary the propagation delay by varying the capacitive load C_L. This variation in load capacitance can be realized by one ore more voltage dependent capacitors called varactor diodes. A reverse-bias *pn* junction can serve as a varactor which has capacitance of:

$$C_{var} = \frac{C_0}{(1 - V/\Phi_B)^m} \tag{2.13}$$

where C_0 is a zero bias capacitance, V is the applied voltage, Φ_B is the built-in voltage of the junction, and m is a value typically between 0.3 and 0.5 [15]. Adding a varactor diode increases the load capacitance, therefore, directly affect the tuning range of the VCO. Additionally, the nonlinear relationship between controlled voltage and the varactor diode capacitance is also translated into the nonlinearity of the VCO.

There are many other methods to implement a VCO including the source-coupled VCO [13] and delay interpolation VCO [14]. Circuit-level techniques can be used to improve uncalibrated VCO linearity, easing requirements on digital calibration and allowing smaller LUT size. However, since designing a VCO is not the main focus of this project, these techniques are not covered in details. Additionally, as the calibration technique operates entirely in the digital domain, its applicability is not limited by the specific VCO circuit architecture [1].

Chapter 3

Background Calibration and Correction Technique

A significant challenge to the VCO-based ADC architecture is to mitigate the effect of the nonlinear V-f characteristic of the VCO. The main purpose of this section is to discuss a calibration method to improve the linearity of the VCO-based ADC. First, the lookup-table with linear interpolation method is discussed. The next section investigates the “Split ADC” background digital calibration approach. Finally, the calibration and correction technique is summarized and the functional block diagram, as how the technique is implemented, is presented.

3.1 Lookup-Table Linearity Correction

An ideal VCO-based ADC has the digital output code n proportional to the analog input V_{IN}. However, as discussed by Section 2.2.3, the real relationship between the VCO count n and the input V_{IN} is usually nonlinear. In order to correct the nonideal output code, the proposed digital correction technique utilizes a lookup-table (LUT).

This LUT provides an additional transfer function between the "uncorrected" count and the final desired digital output code. Mathematically, this transfer function is the inverse of the VCO characteristic; thus it can cancel the nonlinearity of the VCO. This can be explained in Figure 3.1.

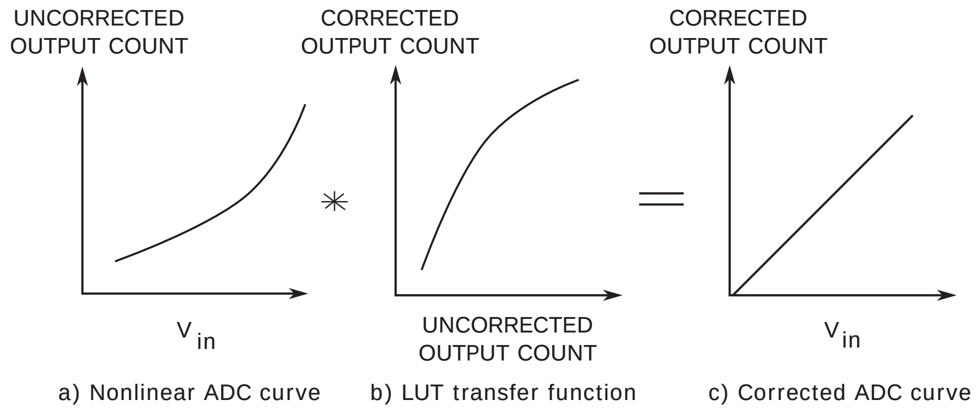


Figure 3.1: Transfer functions of nonlinear ADC with LUT correction

Since LUT is a discrete point by point mapping implementation of a transfer characteristic, it requires 2^N entries to fully cover the whole range of the N bit ADC uncorrected output. In order to reduce complexity of the digital implementation, a combination of the LUT approach and linear interpolation is used [1, 2]. Figure 3.2 shows definition of the LUT: The uncorrected counter output n is divided into an upper and a lower group of bits, thereby segmenting the ADC transfer characteristic. The size of the upper MSB word n_U, U bits long, determines the maximum number of points M in the LUT: $M \leq 2^U + 1$. The MSBs n_U is served as index to a lookup table which holds correction coefficients a_{n_U}. Within each segment, the value of the LSB word n_L is used to linearly interpolate between adjunct a_{n_U} and a_{n_U+1} values in the LUT. Since the two adjunct LUT entries are separated by 2^L on the n axis, the corrected output code x can be calculated as:

$$x = a_{n_U} + \underbrace{\frac{n_L}{2^L}}_{y}(a_{n_U+1} - a_{n_U}) \tag{3.1}$$

The LUT can be implemented using digital registers and multiplexers. One multiplication is required for the linear interpolation. The fraction $1/2^L$ can be realized simply by an L bit shift in radix point. Since the lengths of the MSB word and LSB word determine the LUT length and spacing, they also affect the linearity of the final digital output. The number of points in the LUT needed for adequate correction is determined by the desired ADC accuracy and the nonlinearity of the VCO V-f characteristic. More points in the LUT provide a better linearity of the digital output but require more complex digital circuitry, and consume more power [1, 2].

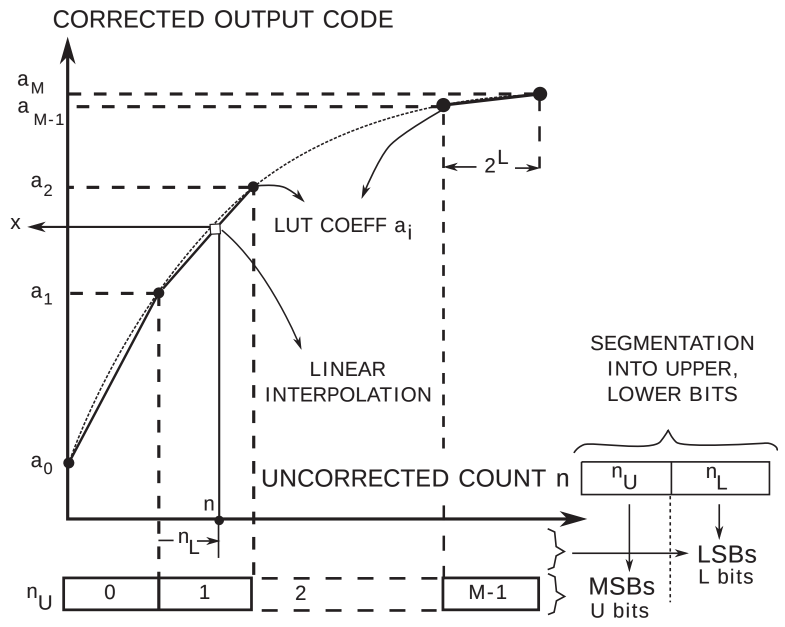


Figure 3.2: Lookup table with linear interpolation digital correction

Another aspect that must be considered when implementing the LUT is the "redundancy factor". Due to the discrete nature of the input count, there are numbers that cannot come out of the interpolation. An example of this is shown in Figure 3.3. At the region where the slope of the LUT transfer function dx/dn is greater than 1, as the input count increases by 1 the corresponding output code might increase by more than 1, therefore skipping over some possible values, leading to missing codes in the ADC. To ensure every output x can be reached by at least

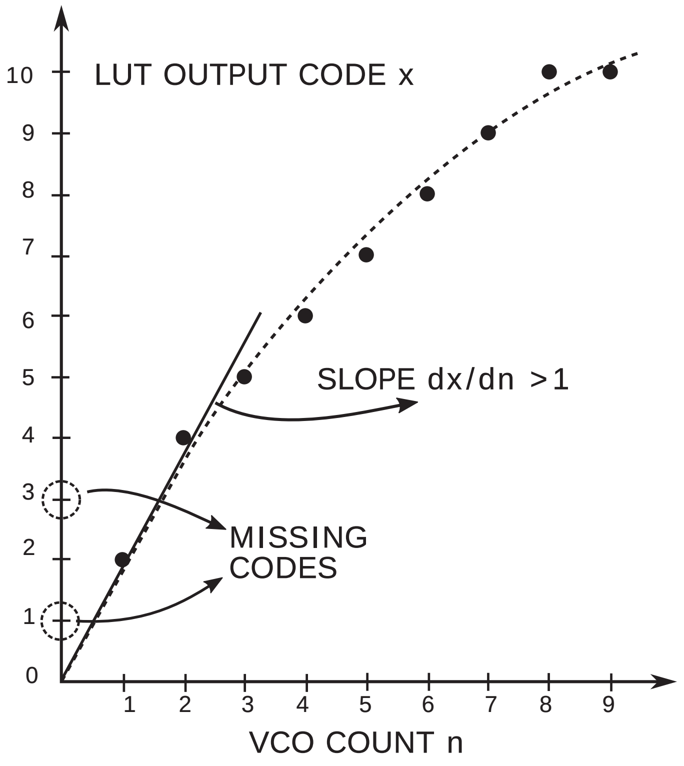


Figure 3.3: Missing codes in LUT implementation

one input count n, the slope dx/dn must always be less than unity [18]. For an N-bit converter, the ultimate output code x will have 2^N possible values, zero to $2^N - 1$. In order to ensure the slope to be less than unity, the total possible output counts of the VCO must be increased to $R(2^N - 1)$, where R is a redundancy factor. For an ideal ADC, the slope is always 1, therefore, no redundancy would be required, and $R = 1$.

There are several ways to implement the redundancy factor in VCO-based ADC. The first one would be to initialize the LUT so that total output range is reduced by a factor of R compared to the input range. For example, as discussed previously, the distance between two adjunct LUT entries in the n domain is 2^L where L is the LSB word length. The redundancy factor R could be realized by initialize the LUT such that the difference between two values in the adjunct LUT locations is $2^L/R$.

Another simpler approach is to shift the radix point of the digital output code. For instance, the redundancy factor of 4 can be implemented by a left shift in the radix point by 2 bits. Although, this method only works if R is a power of 2, it is easy to be implemented in digital domain and will be utilized in this project.

3.2 Dithered Split-ADC Calibration Concept

The task of the calibration procedure is to determine the a_i coefficients in the LUT used for linear interpolation as shown in Figure 3.2. One option would be to take the ADC offline, sweep the input linearity over the entire signal range and determine the proper coefficients a_i, as shown in [10]. Disadvantages of this approach include the need to take the ADC offline and develop the known input signal. Another drawback is that the VCO characteristic could be a function of temperature. As the VCO characteristic changes, it requires different set of coefficients a_i to properly correct the digital output. Therefore, offline calibration technique cannot mitigate the nonlinearity problem if the ADC characteristic differs over time. This section discusses the proposed approach using “Split ADC” architecture presented in [1, 2] to realize the background calibration with no need for an accurately known input signal.

Figure 3.4 shows the split ADC concept as implemented in this project. The

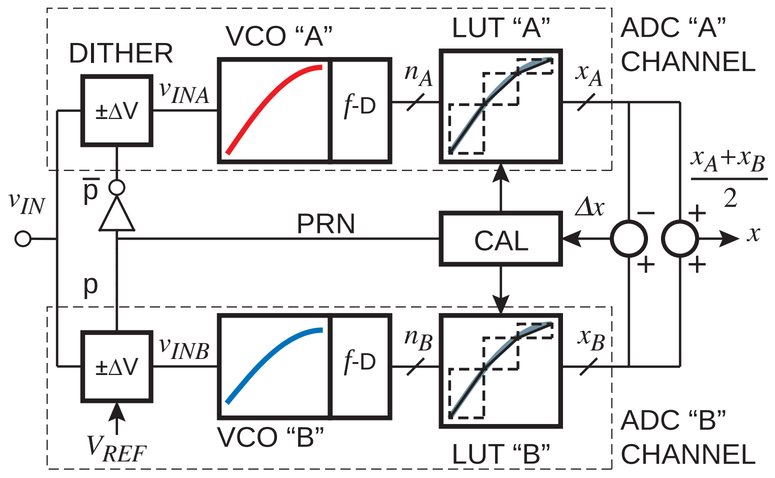


Figure 3.4: Dithered Split ADC system block diagram

signal path is split into two channels, each producing individual output codes x_A and x_B. A dither signal $\pm\Delta V$ is added to the input voltage V_{IN} so that the inputs to each channels are:

$$V_{INA} = V_{IN} - p\Delta V \tag{3.2}$$

$$V_{INB} = V_{IN} + p\Delta V \tag{3.3}$$

in which $p = \pm 1$ is chosen on a pseudo-random basis for each conversion. The best estimate for ADC output code x is the average of the x_A and x_B outputs:

$$x = \frac{x_A + x_B}{2} \tag{3.4}$$

Since, the dither is added to one channel, and subtracted from the other, its effect can be eliminated when averaging the digital output codes. The difference Δx of the two output codes x_A and x_B can be used to calibrate the two LUTs transparently to converter operation in the output code signal path:

$$\Delta x = x_B - x_A \tag{3.5}$$

For more intuitive understanding of the algorithm, the next subsections consider the local difference between the “A” and “B” characteristics, followed by calibration of the slope.

3.2.1 ADC Characteristic Alignment

In general the two VCOs have different characteristics, requiring two separate LUTs for each of the “A” and “B” ADCs, as shown in Figure 3.5. Since the VCO characteristics are different, even with identical input voltages the uncorrected output count n_A and n_B will generally be different. If the two LUTs had been correctly calibrated, the x_A and x_B outputs would have been equal. If the LUTs are not calibrated correctly, the LUT output x_A and x_B would have been different. The nonzero difference $\Delta x = x_B - x_A$ indicates a need to adjust the LUTs to bring the LUTs and the ADC into calibration. Since there is no reason to prefer one VCO

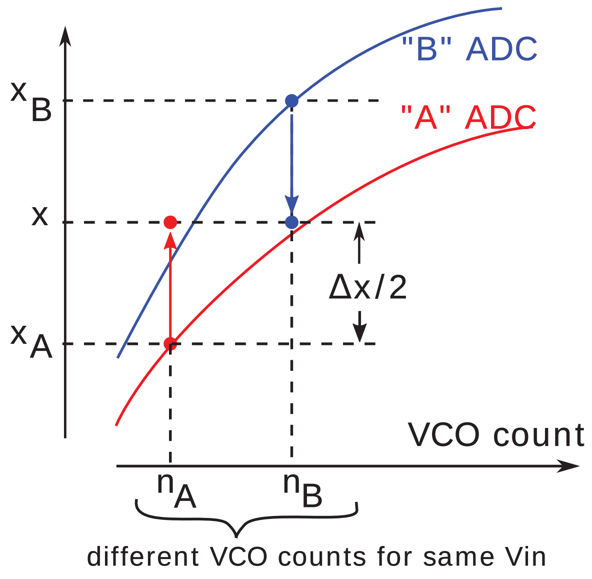


Figure 3.5: Split ADC characteristic alignment

over the other, half of the total difference Δx is assigned to each LUT to produce the same output code of:

$$x = x_B - \frac{\Delta x}{2} = x_A + \frac{\Delta x}{2} = \frac{x_B + x_A}{2} \tag{3.6}$$

For each conversion, one LUT location is updated such that the output codes are the same for identical input voltages V_{IN} regardless of the disagreement in the two V_{IN}-to-n characteristics. Therefore, over many conversions across the input signal range, repeating this LUT adjustment process will eventually bring the ADC characteristics into agreement. However, the main difficulty of this approach is that while point-by-point agreement among the conversions can be achieved, the linearity of the ADC is not guaranteed. In other words, calibration can end up with two agreeing but equally erroneous nonlinear characteristics as shown in Figure 3.6. This difficulty is addressed in Section 3.2.2 below.

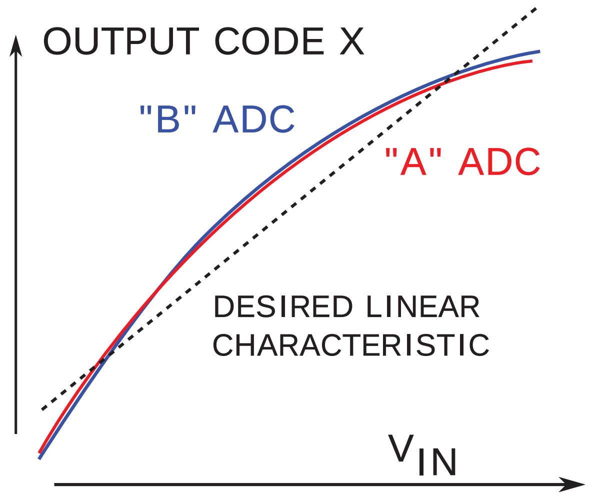


Figure 3.6: Split ADC Characteristic alignment with two agreeing but equally non-linear ADC characteristics

3.2.2 Slope Calibration

The characteristic alignment technique discussed above does not have any visibility to the linearity of the two channels. To enable correction of linearity errors, the dither function is added to the input as shown in the system block diagram of Figure 3.4. The inputs are offset by a known dither value $\pm\Delta V$ representing a known code excursion $\pm D$ at the ADC output. However, since the two ADC characteristics are different, the actual codes corresponding to ΔV also differs from the ideal value. Therefore, the general input voltages and output codes of the two channels can be expressed by:

$$V_{INA} = V_{IN} - p\Delta V \rightarrow x_A = x - pd_a \tag{3.7}$$

$$V_{INB} = V_{IN} + p\Delta V \rightarrow x_B = x + pd_b \tag{3.8}$$

where p is either +1 or −1. As Equation 3.7 and Equation 3.8 indicate, not only the point-by-point agreement but also the slopes of the VCO characteristics are now under consideration. The difference of the two output codes x_{outA} and x_{outB} can be calculated as:

$$\Delta x = (x - x) + p(d_a + d_b) = p(d_a + d_b) \tag{3.9}$$

Figure 3.7 shows the idea of slope calibration. Ideally, when the two ADCs are identical and linear, d_a and d_b would be equal to D. Therefore $\Delta x_{(ideal)} = p(D+D) = 2pD$. The fact that Δx is different from the desired value $\pm 2D$ indicates that the slope of the “A” and “B” characteristics need to be adjusted. The mathematical derivation of how the LUT entries would be updated is discussed in Section 3.2.3 below.

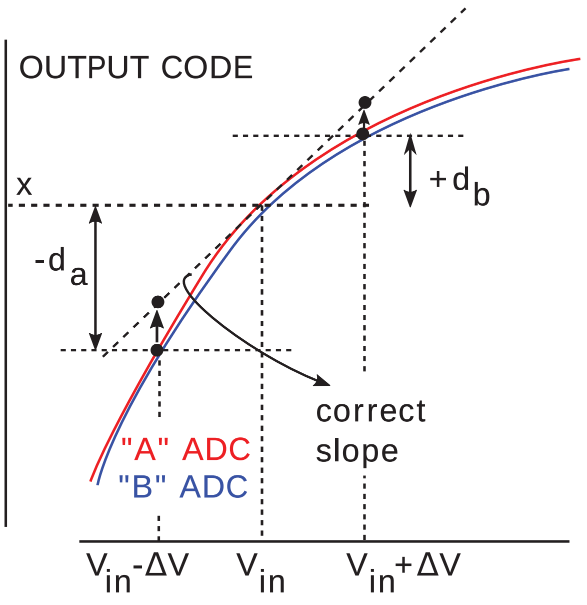


Figure 3.7: Slope Calibration

3.2.3 Error Estimation

This section investigates the mathematical derivation of the error estimation and LUT calibration process. Firstly, as discussed in the previous sections, generally the two LUTs have different coefficients denoted by a_i and b_i corresponding to ADC "A" and ADC "B". Therefore, for each LUT Equation 3.1 can be rewritten as:

$$\hat{x}_A = (1 - y_A)\hat{a}_{n_{UA}} + y_A\hat{a}_{n_{UA}+1} \tag{3.10}$$

$$\hat{x}_B = (1 - y_B)\hat{b}_{n_{UB}} + y_B\hat{b}_{n_{UB}+1} \tag{3.11}$$

The "hat" above each parameter in the equations denotes the difference between the actual value, that is needed to be calibrated, and the desired correct value. The LUT entries can be redefined as the sum of the correct value and an error term ε.

$$\hat{a}_{n_{UA}} = a_{n_{UA}} + \varepsilon_{n_{UA}} \tag{3.12}$$

$$\hat{b}_{n_{UA}} = b_{n_{UA}} + \varepsilon_{n_{UB}} \tag{3.13}$$

Substitute (3.12) and (3.13) into (3.10) and (3.11) gives:

$$\begin{aligned}\hat{x}_A &= (1-y_A)(a_{n_{UA}} + \varepsilon_{n_{UA}}) + y_A(a_{n_{UA}+1} + \varepsilon_{n_{UA}+1}) \\ &= \underbrace{[(1-y_A)a_{n_{UA}} + y_A a_{n_{UA}+1}]}_{x_A} + (1-y_A)\varepsilon_{n_{UA}} + y_A\varepsilon_{n_{UA}+1}\end{aligned}$$

and:

$$\begin{aligned}\hat{x}_B &= (1-y_B)(a_{n_{UB}} + \varepsilon_{n_{UB}}) + y_B(b_{n_{UB}+1} + \varepsilon_{n_{UB}+1}) \\ &= \underbrace{[(1-y_B)b_{n_{UB}} + y_B b_{n_{UB}+1}]}_{x_B} + (1-y_B)\varepsilon_{n_{UB}} + y_B\varepsilon_{n_{UB}+1}\end{aligned}$$

Given that the difference of the two corrected digital outputs, $x_B - x_A$, should be $2pD$ as mentioned in Section 3.2.2, taking the difference of the estimates results in:

$$\hat{x_B} - \hat{x_A} - 2pD = (1-y_B)\varepsilon_{n_{UB}} + y_B\varepsilon_{n_{UB}+1} - (1-y_A)\varepsilon_{n_{UA}} - y_A\varepsilon_{n_{UA}+1} \tag{3.14}$$

Equation 3.14 captures the contribution of each LUT entry error to the variation of $\Delta x = x_B - x_A$ from its ideal value $2pD$. If all the error terms ε in (3.14) are zero, then the left hand side must be equal to zero, indicating the correct offset and slope calibration of the ADC characteristics.

Mathematically, four conversions are needed to solve for four unknown errors in a specific LUT entry in (3.14). Since there are many LUTs errors to determine, an ensemble of K (≈ 1000) conversions is accumulated. A matrix representation of these results are described by:

$$\begin{bmatrix} \mathbf{Y_A} & \mathbf{Y_B} \end{bmatrix} \cdot \begin{bmatrix} -\mathbf{e_A} \\ \mathbf{e_B} \end{bmatrix} = \mathbf{\Delta x} - 2D\mathbf{p} \tag{3.15}$$

$\mathbf{Y_A}$ and $\mathbf{Y_B}$ are $K \times M$ matrices containing coefficients y_A and y_B in (3.14); K is the ensemble size and M is the length of the LUT. An example of $\mathbf{Y_A}$ is shown in (3.16). Row i^{th} of the matrix represents error weight coefficients for the i^{th} conversion in the ensemble; and column k^{th} contains coefficients corresponding to k^{th} entry in the LUT. Since in every conversion, only two of the LUT locations n_U and $n_U + 1$ would be hit, there are only two nonzero terms,$1 - y$ and y, in each row.

$$\mathbf{Y_A} = \begin{bmatrix} 0 & (1-y_{A1}) & y_{A1} & 0 & \cdots \\ 0 & 0 & (1-y_{A2}) & y_{A2} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \\ (1-y_{AK}) & y_{AK} & \cdots & 0 & \cdots \end{bmatrix} \tag{3.16}$$

$\mathbf{e_A}$ and $\mathbf{e_B}$ in (3.15) are $M \times 1$ column vector of the LUT errors to be determined:

$$\mathbf{e_A} = \begin{bmatrix} \varepsilon_{0A} \\ \varepsilon_{1A} \\ \vdots \\ \varepsilon_{MA} \end{bmatrix}, \qquad \mathbf{e_B} = \begin{bmatrix} \varepsilon_{0B} \\ \varepsilon_{1B} \\ \vdots \\ \varepsilon_{MB} \end{bmatrix} \tag{3.17}$$

$\mathbf{\Delta x}$ and $2D\mathbf{p}$ are $K \times 1$ column vectors of the actual $\hat{x}_B - \hat{x}_A$ differences and the ideal values $x_B - x_A = 2pD$, respectively in each conversion.

$$\mathbf{\Delta x} - 2D\mathbf{p} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_K \end{bmatrix} - 2D \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_K \end{bmatrix} \tag{3.18}$$

Ideally, the LUT error vectors $\mathbf{e_A}$ and $\mathbf{e_B}$ could be determined by solving Equation 3.15, then would be subtracted from the a_i and b_i in the LUT to get correct coefficients. However, there are difficulties associated with this approach. Firstly, since the total number of LUT locations ($2M$) is much fewer than the number of conversions K in an ensemble, the system is considered as an "overdetermined" systems. In other words, the matrix $\mathbf{Y} = \begin{bmatrix} \mathbf{Y_A} & \mathbf{Y_B} \end{bmatrix}$ in Equation 3.15 has more rows than columns; therefore, the system of equations has more equations than unknowns. According to [20], overdetermined system is usually inconsistent and does not have a unique solution, especially when there is a lot of uncertainty and randomness in the system. Secondly, reducing the number of conversions K to $2M$ makes $\mathbf{Y}$ to be a square matrix, but does not guarantee it to be full rank; and unique solution does not exist. Particularly, when there are fewer conversions, it is very likely that the input signal does not hit all the LUT locations in one ensemble, resulting in "zero" columns in $\mathbf{Y}$. Finally, even if the solution of (3.15) exists, solving for $\mathbf{e_A}$ and $\mathbf{e_B}$ requires a very complicated left inverse matrix operation which is too computationally intensive and could not be realized by a simple digital circuitry. Due to all of these difficulties, another approach is investigated in the next section to estimate the error of the LUT coefficients.

3.2.4 Iterative Matrix Solution

The main challenge to the calibration process addressed in Section 3.2.3 includes the ability to solve for an exact solution of Equation 3.15. To simplify the digital hardware, an iterative procedure is used to avoid matrix inversion [3]. Instead of solving for exact error terms, an LMS-style estimation method is adopted. The procedure begins by firstly multiply both side of Equation 3.15 with the transpose of $\mathbf{Y}$:

$$\left(\mathbf{Y^T} \cdot \mathbf{Y}\right) \cdot \mathbf{e} = \mathbf{Y^T} \cdot (\mathbf{\Delta x} - 2D\mathbf{p}) \tag{3.19}$$

For simplicity, assume (unrealistically) that the $\left(\mathbf{Y}^{\mathbf{T}} \cdot \mathbf{Y}\right)$ matrix is equal to the identity matrix

$$\left(\mathbf{Y}^{\mathbf{T}} \cdot \mathbf{Y}\right) = \mathbf{I} \tag{3.20}$$

Substitute (3.20) into (3.19) gives an "over simplified" solution to the error matrix:

$$\mathbf{e} = \begin{bmatrix} -\mathbf{e_A} \\ \mathbf{e_B} \end{bmatrix} = \mathbf{Y^T} \cdot (\mathbf{\Delta x} - 2D\mathbf{p}) \tag{3.21}$$

The corrected LUT entries would be calculated by subtracting the error terms from the incorrect LUT coefficients. However, since (3.21) relies on the unrealistic assumption (3.20), a least mean square (LMS) method is adopted by subtracting a small portion μ of the estimated errors $\mathbf{e}$ as follow:

$$a_i^{new} = a_i^{old} - \mu\varepsilon_{iA} \tag{3.22}$$

$$b_i^{new} = b_i^{old} - \mu\varepsilon_{iB} \tag{3.23}$$

Examination of (3.21) and (3.16) shows that the large $\mathbf{Y}$ matrix need not be stored, since the information required for the $\varepsilon_{\mathbf{A}}$ and $\varepsilon_{\mathbf{B}}$ estimation can be accumulated on a conversion-by-conversion basis. Every conversion, if a LUT location is hit, the error for that entry will be accumulated and then finally subtracted from the actual LUT coefficients at the end of the ensemble. The way each error is accumulated in every conversion can be described by:

$$\varepsilon_{iA}{}^{(new)} = \varepsilon_{iA}{}^{(old)} - (1 - y_{iA})(\Delta x_i - 2p_i D) \tag{3.24}$$

$$\varepsilon_{(i+1)A}{}^{(new)} = \varepsilon_{(i+1)A}{}^{(old)} - y_{iA}(\Delta x_i - 2p_i D) \tag{3.25}$$

$$\varepsilon_{iB}{}^{(new)} = \varepsilon_{iB}{}^{(old)} + (1 - y_{iB})(\Delta x_i - 2p_i D) \tag{3.26}$$

$$\varepsilon_{(i+1)B}{}^{(new)} = \varepsilon_{(i+1)B}{}^{(old)} + y_{iB}(\Delta x_i - 2p_i D) \tag{3.27}$$

The energy cost of implementing the algorithm is modest; only two additional multiplications per conversion are required.and the multiplier is already available as a resource since it is required for the linear interpolation. As the four equations indicate, all four error terms are proportional to $\Delta x_i - 2p_iD$. The more Δx differs from its ideal value, the larger the error terms. Additionally, the "distance" between the actual output code to the LUT coefficient also plays an important role in updating the LUT. It is related to the percentage of the contribution of each LUT entry to the error of the output. Intuitively, this makes senses, since the closer the value to a LUT location, the more effect that LUT entry has on the output code. Therefore the error of the output code is more likely due to the error of that LUT entry compared to the others. This can be seen in Figure 3.8. One extreme example of this is when y_{iA} equals zero; all of the output error is due to LUT coefficient a_i and no information on the error of a_{i+1} is observed.

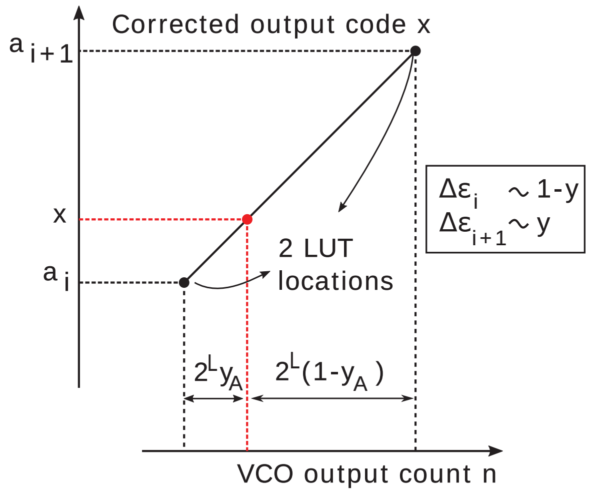


Figure 3.8: Error estimation depends on distance of the LUT locations to the input count

A key advantage of the LMS approach is that the error estimates need not be accurate; all that is required is that they be zero-bias and (on average) steer the convergence of each LUT entry in the correct direction [3]. Secondly, development of the split ADC approach relied on the A and B inputs differing by a known ΔV dither. In practice, noise will cause an additional difference, leading to inaccuracy in error estimation even if the error terms had been solved exactly. By averaging information over many ensembles of conversions, the LMS approach averages out the effect of noise in determining calibration parameters [1].

3.2.5 Limited Signal Range and "Stitching"' Estimation

As mentioned previously in Section 3.2.3, for each conversion, only the two LUT locations, which are hit, are updated. Therefore, regions of the LUT not covered by the input signal are not calibrated. If the signal activity histogram changes to access previously unused LUT entirely, ADC error may increase dramatically until the background calibration loop can converge for the newly used portions of the LUT.

Consider the example shown in Figure 3.9. Suppose the error estimation process begins with an initial distribution of linearly spaced LUT entries a_i as shown in the plot at the top of the figure. Also shown as the solid gray line in the figure is the correct characteristic for error estimation process to converge to. Since the LUT entries are incorrect the error estimation process would result in nonzero values, which will be used in the LMS loop to drive the errors toward zero.

In the middle of Figure 3.9 is a histogram of input voltage distribution during one ensemble of K conversions. Due to the limited signal range, only LUT segments 2-5 are used; the input voltage never reaches the range corresponding to segments 1, 6, or 7.

Below the histogram in Figure 3.9 is a plot showing open circles to represent the results of the error estimation process for this ensemble of data. Nonzero error estimates are correctly returned for locations a_1-a_5. But since LUT locations a_0, a_6,

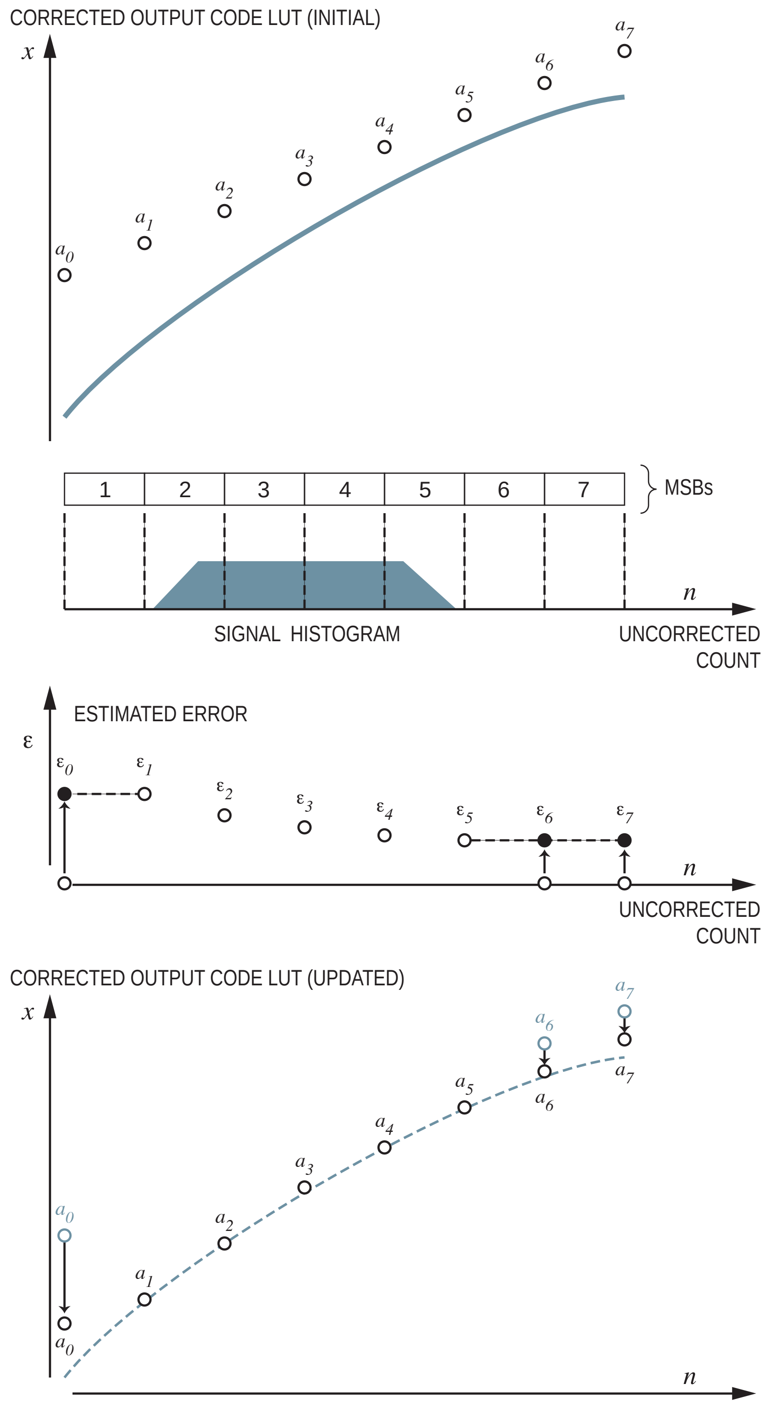


Figure 3.9: Example for portions of input range not covered by signal.

and a_7 are never used, estimate for these locations is zero.

At the bottom of Figure 3.9 is a plot of the updated lookup table. As the LMS process updates the a_i values, locations a_1-a_5 are corrected since the input signal range allows proper estimation of errors ε_1-ε_5. But since the error estimates are zero for a_0, a_6, and a_7, they will not change even as a_1-a_5 are corrected. The result, with gray circles shown for a_0, a_6, and a_7 at the bottom of figure 3.9, is a LUT with discontinuities and potentially even nonomontonicity. If the input signal histogram shifts, the ADC will make large errors until the calibration loop converges with the new information.

The solution implemented in this work is based on the ability to distinguish between the cases of $\varepsilon = 0$ due to a true zero error, and $\varepsilon = 0$ due to an LUT location not being used. During each ensemble of conversions, the calibration algorithm keeps track of whether an LUT location has been used or not. After calculation of the ε_i values, but before the LMS updating, the algorithm checks for unused LUT locations. When the algorithm reaches an unused LUT location, the $\varepsilon = 0$ in that location is replaced with the nearest valid estimate from an LUT location that was used. This substitution is represented with the arrows and solid circles in the plot of ε in Figure 3.9. The result, shown with the open circles for a_0, a_6, and a_7 at the bottom of Figure 3.9, preserves continuity of the LUT and reduces ADC errors when the input signal range histogram changes. This "stitching" of the LUT does not completely eliminate ADC errors. Since there is no signal in the unused LUT locations no information as to the correctness of those values is determined. However this technique does preserve continuity of the lookup table at the boundary between the used and unused portions of the signal range.

3.2.6 Offset Consideration

As noted earlier, this calibration approach provides no information on offset. The error estimation block only sees the difference between the two channels, thus has no vision on the absolute offset of the ADC. Since the LMS loop is a perfect numerical integrator, any systematic offset errors in the estimation process would accumulate indefinitely, causing numerical overflow. To prevent this numerical problem, the value of one location in the "A" table is fixed and all other error estimates are referenced to that location to prevent a global drift in offset of the lookup table entries.

If absolute offset accuracy is required, one of the "A" or "B" converters can be taken off line for one conversion to sample a known DC voltage and provide an absolute offset reference for the ADC. When this is done the averaging the two channel is suspended for that one conversion and the ADC output is determined only by the output of the other "A" or "B" converter that was not taken offline.

3.3 Calibration Algorithm Summary

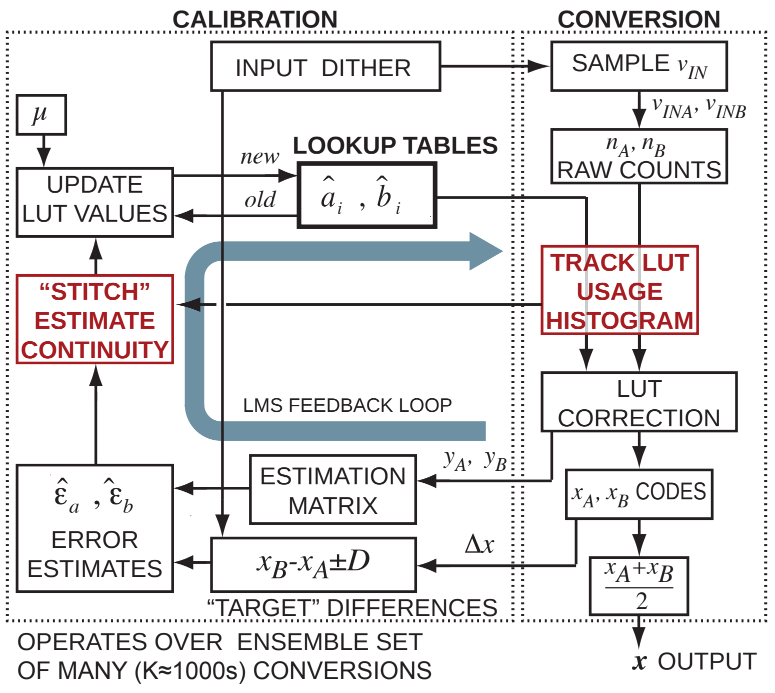


Figure 3.10: Calibration algorithm flow chart

Figure 3.10 summarizes the LUT calibration and correction technique implemented in this project. While the right hand side of the figure captures the operations that occur every conversion, the left hand side includes all calculations that are done for every ensemble (K conversions).

Initially, the analog input voltage is preprocessed by adding a known dither voltage ΔV to one channel and subtracting ΔV to the other. The sign of the dither is determined by a pseudo-random sequence $p = \pm 1$. The analog voltage is then converted in to digital outputs n_A, n_B by counting the number of phase transitions of the VCO output clock signals in a given period of time. Generally, the two VCOs are not identical and both nonlinear; therefore two LUTs are implemented to correct the output counts. The final digital code is obtained and the effect of dither is eliminated by averaging the two resulted outputs of the two channels x_A and x_B.

In every conversion, there are at most two adjunct LUT coefficients used for each channel. These locations are then recorded by the "TRACK LUT USAGE HISTOGRAM" block in Figure 3.10. Additionally, all parameters needed to estimate the error using Equation 3.15 are also calculated, including y_A, y_B, and Δx. The error terms ε_A and ε_B of all the used LUT locations are then accumulated over K conversions. Finally, before the LUTs are updated by subtracting a small amount $\mu\varepsilon$, the error of the unused LUT coefficients are managed by the "STITCH EXTIMATE CONTINUITY" block to preserve the continuity of the LUT.

Chapter 4

Analog Circuit and PCB Implementation

This chapter describes the Printed Circuit Board (PCB) design of the VCO-based ADC. The choice of the VCO architecture is discussed in the first section. In order to reduce the frequency of the signal processed by the FPGA board, a frequency divider was built; the details are discussed in the next section. Finally, the overall block diagram of the PCB is summarized in the last section.

4.1 Ring VCO

The purpose of this section is to discuss a VCO topology which can provides an oscillating waveform swinging from $0V$ to $1.8V$ and has wide frequency range of about 100MHz. Although the linearity of the VCO directly affects the performance of the ADC, the digital calibration technique applicability is not limited by the VCO architecture [1, 2]. Due to the time constraint, this project does not concentrate on a complete VCO design in transistor level. Instead, different circuits were built at the discrete level, and then tested to determined the most suitable implementation of the VCO for the ADC specifications.

Most of these implementations were based on the 74AUC1G00 NAND gate. There is no specific reason for using this chip other than the availability of the component in the lab. It should be noted that the fundamental building block of a VCO, an inverter, can be easily implemented using a NAND gate by connecting the two input of the gate as shown in Figure 4.1. Three NAND gates are then connected together in a feedback loop to form a ring oscillator. The frequency at which the circuit oscillates depends on the propagation delay of each stage as discussed in Section 2.3

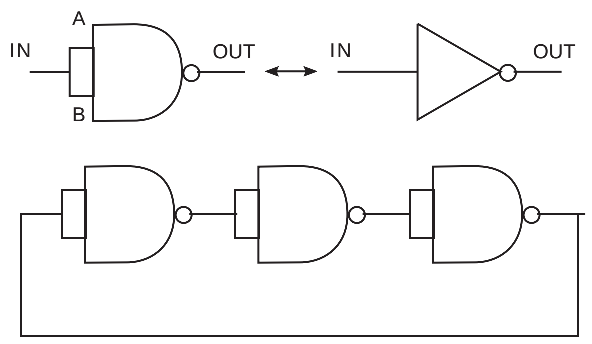


Figure 4.1: Simple ring oscillator implemented by NAND gate

There are several ways to control the frequency of the oscillator. One could be to use the current starved VCO architecture by using a MOSFET to control to current provided to the inverter as shown in Figure 4.2. However, one difficulty with this approach is that the power supply V_{DD} and the input voltage range V_{IN} needs to be high enough so that transistor does not crash in to saturation region. More importantly, experiments show that the frequency to V_{IN} characteristic of this structure is not well behaved. The output waveform has relatively large ripples, causing uncertainty in the phase transition measurements. This might be caused by the internal structure of the NAND gate which cannot be controlled. Therefore another structure is considered for this project.

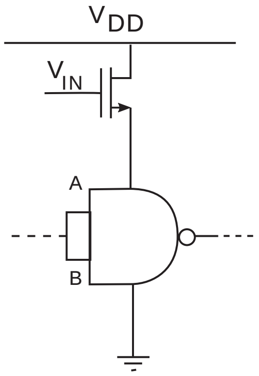


Figure 4.2: Using MOSFET to control the delay of the inverter

Shown in Figure 4.3 is the schematic of the VCO used in this project. The ring oscillator is implemented in the same manner as Figure 4.1. The speed of the oscillator is controlled directly by varying the supply voltage of these inverters. $R_S = 50\Omega$ is the protecting resistor to limit the input current. An emitter follower is added to buffer the input voltage V_{IN}, preventing the two channels to be coupled together.

One important factor that needs to be considered when designing the emitter follower is that as the current drawn by the ring oscillator changes, the bias voltage V_{BE} also changes exponentially. In order to avoid the nonlinear effect introduced by the emitter follower, it is desired that the collector current of transistor I_C should be large enough compared to the maximum current drawn by the ring oscillator I_{RING}. Measurement shows $I_{RING(max)} \approx 5mA$. If I_C is about $3X$ larger than I_{RING} then the resistor R_E can be determined by:

$$R_E = \frac{V_E}{I_C - I_{RING}} \approx \frac{2V}{15mA - 5mA} = 200\Omega \tag{4.1}$$

The input voltage V_{IN} is translated directly to the supply voltage of the ring

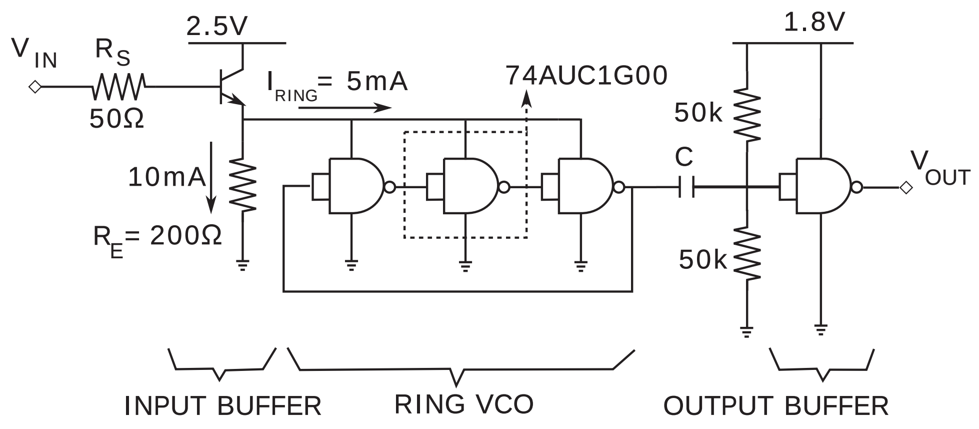


Figure 4.3: Ring VCO schematics

oscillator by: $V_{RING} \approx V_{IN} - 0.7V$. Therefore, not only the frequency but also the amplitude of the oscillated output waveform depend on the input voltage V_{IN}. In order to keeps the amplitude of the output clock signal approximately at $1.8V$, one more inverter is required as shown in Figure 4.3. This inverter is powered by a constant $1.8V$ supply voltage providing an output swing from 0 to $1.8V$. A simple resistor bias circuit is needed to bias the input of the inverter at around $0.9V$. The output of the ring oscillator is AC coupled to the output stage by the capacitor $C = 1000pF$. Two VCOs in Figure 4.3 were built; the sample waveform of the

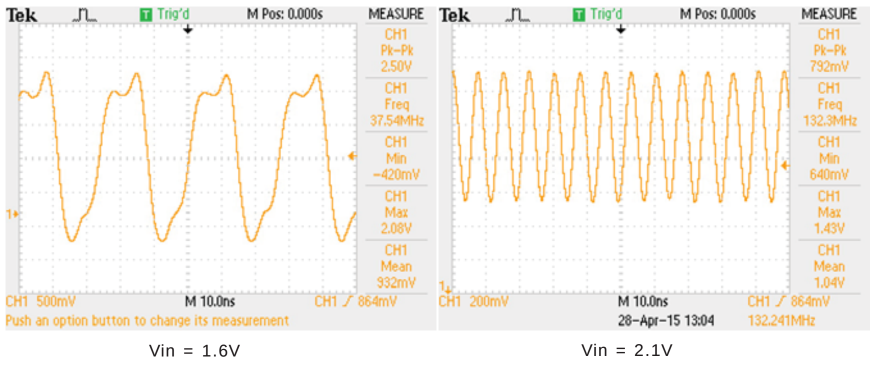


Figure 4.4: VCO output waveform for 1.6V and 2.1V input voltage

output is shown in Figure 4.4 and the characteristics were measured as shown in Figure 4.5. The analog input voltage ranges from $1.6V$ to $2.1V$ and the output

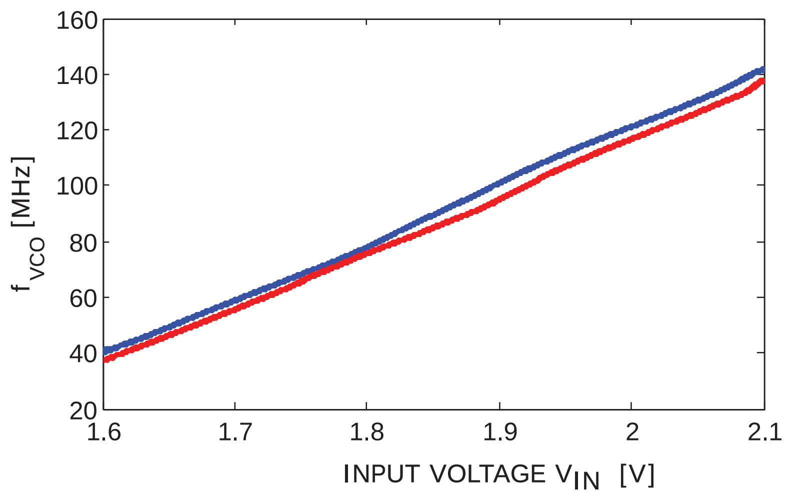


Figure 4.5: VCO V-to-f characteristic

frequency ranges from 40 MHz to about 140 MHz. It can be seen in the figure that the two VCOs have similar but not identical characteristics. Both of them are nonlinear; but the slopes df/dV only vary within about ±20% of its average value which is given by:

$$K_{VCO(AVG)} = \frac{\Delta f}{\Delta V} \approx \frac{14\text{MHz} - 40\text{MHz}}{2.1V - 1.6V} = 200 \left[\frac{\text{MHz}}{V}\right] \tag{4.2}$$

4.2 Frequency Divider

Although the digital calibration can be implemented solely using the FPGA board, a divider is built in order to reduce the frequency of the waveforms processed by the FPGA. This frequency divider can be considered as a part of the counter. In other words, the 12 bits counter is split into two 6 bit counters: one is constructed in the PCB, the other is implemented by the FPGA . The PCB schematic of the frequency divider is shown in Figure 4.6. As there is no correlation between the VCO output clock signal and the reset signal provided by the FPGA board, the ripple counter structure is used instead of the synchronous one.

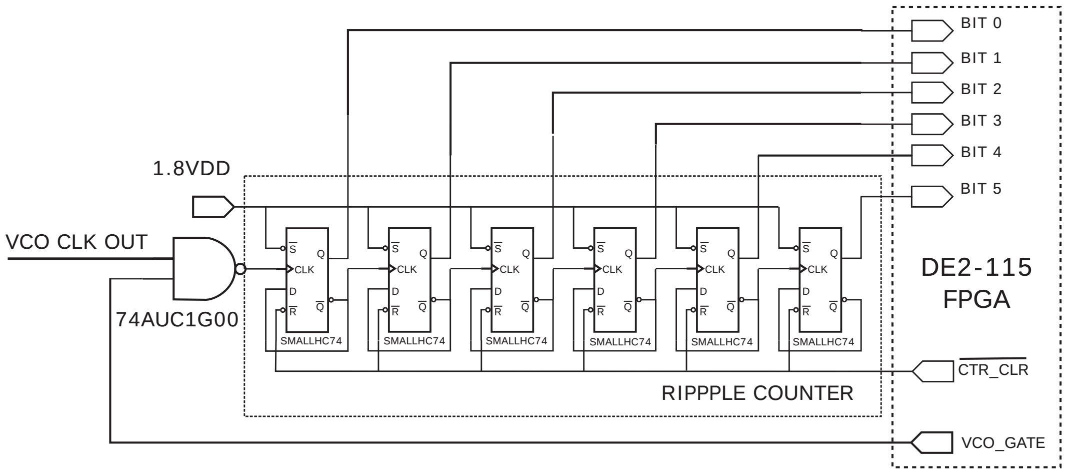


Figure 4.6: Ripple counter circuit schematic

The circuit uses six dual positive edge triggered D-type flip-flops 74HC74. All six bit output signals of the ripple counter become the inputs to the FPGA board. The FPGA only needs to count the most significant bit signal "BIT 5" of which the frequency is 64 times smaller than that of the VCO output f_{VCO}.

It should be noted that, in addition to the low enable reset signal $\overline{\text{CTR\_CLR}}$ the counter also requires a gate signal VCO_GATE. When VCO_GATE = 1, the NAND gate inverts the VCO output clock signal; when VCO_GATE = 0 the output of the NAND gate is always high. Therefore, no phase transition of the VCO output would be counted during the time when VCO_GATE is high. This sets up a "break" time window for the FPGA to grab the data and reset the counter. The actual effective conversion time is only the period when VCO_GATE is low.

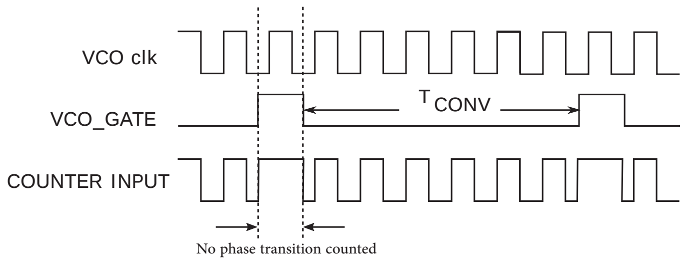


Figure 4.7: Timing diagram of VCO_GATE signal

4.3 PCB Design Summary

Figure 4.8 shows the complete block diagram implementation of the PCB for this project. The input voltage V_{IN} is split into two channels. The PRN signal represents a pseudo-random sequence of either 1 ($V_{DD} = 1.8V$) or 0 (GND). The dither size can be controlled by the resistor R_x as follow:

$$2\Delta V = \frac{V_{DD} R_s}{(R_x + R_s)} \Rightarrow \Delta V = \frac{V_{DD} R_s}{2(R_x + R_s)} \tag{4.3}$$

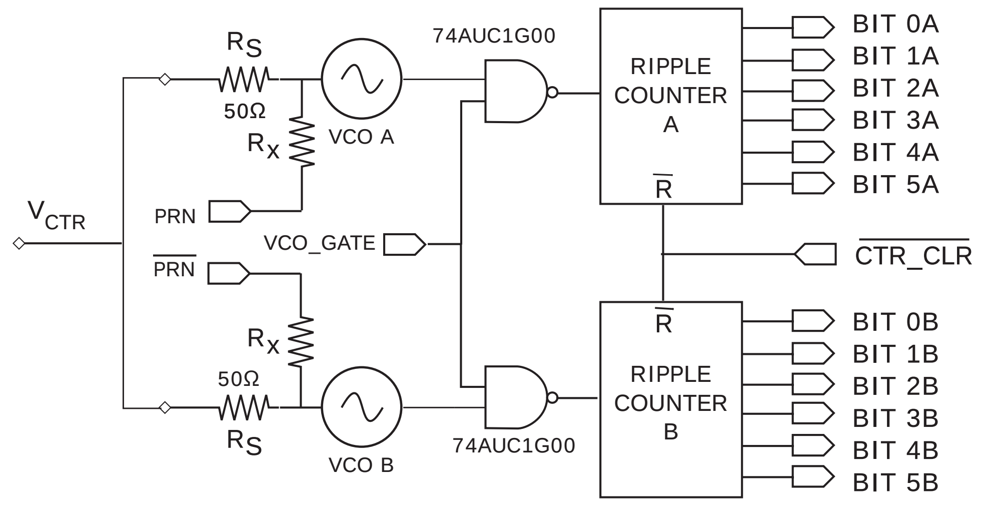


Figure 4.8: PCB block diagram

The analog voltages are then converted to frequency domain using the two VCOs of which the V-to-f characteristics are shown in Figure 4.5. The two 774AUC1G00 NAND gates are then used to generate a break time window for the counter to grab the data and to reset. Two 6 bit ripple counters are constructed using six 74HC74 D-flip flops. The reset signal $\overline{\text{CTR\_CLR}}$, gate signal VCO_GATE, and dither PRN are 0-1.8V digital signals provided by the FPGA board. The first five least significant bits of the two ripple counters are sampled by the FPGA, while the sixth bit signals are then counted to get the last seven most significant bits of the VCO counts n. The detailed implementation of FPGA board are documented Chapter 5

Chapter 5

FPGA Implementation

The main focus of this project is to implement the calibration algorithm in an FPGA board. The board used in this project is DE2-115; and the algorithm is implemented in Altera - Quartus II 13.0. Although the whole calibration technique can be coded solely using Verilog, it is easier to implement each block separately using the schematic feature in Quartus II.

5.1 Top Level Block Diagram Design

This section describes a top design structure of the FPGA implementation for this project. The Quartus II CAD system supports a schematic design method in which the users draws a graphical diagram of the circuit. Each function can be implemented using one block diagram which then can be described by either a verilog source code file or a schematic design file. There are totally 6 blocks in the top level design:

- CLOCK GENERATOR: generate all the clock signals controlling the timing operation of other blocks and provide reset and gate signals for the PCB

- PRN GENERATOR: generate pseudo-random sequence p for the dither ΔV
- COUNTER A: sample 5 LSB signals and count 7 MSBs of VCO A output
- COUNTER B: sample 5 LSB signals and count 7 MSBs of VCO B output
- CALIBRATION: perform the LUT correction and background calibration.
- MEMORY CONTROLLER: control the SRAM to store measured data

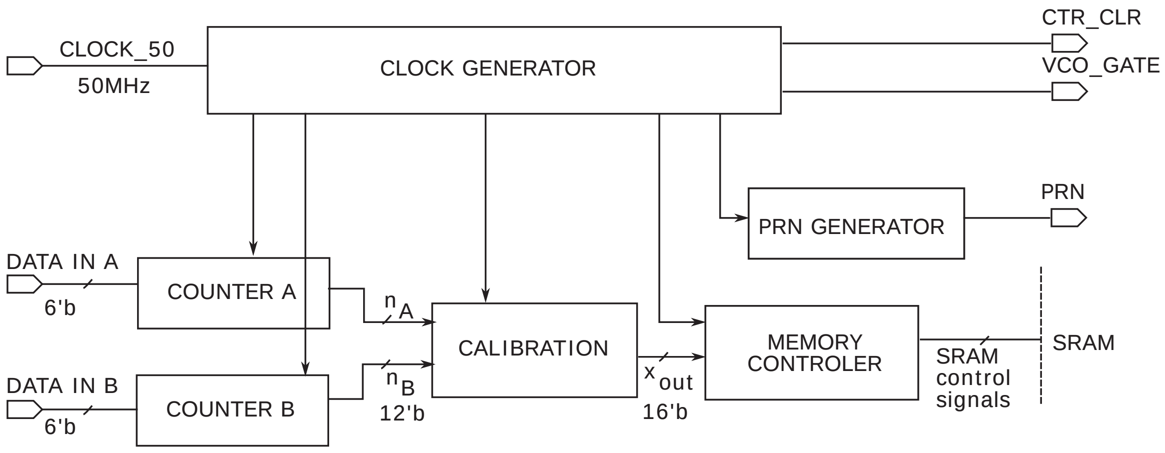


Figure 5.1: Simplified top level block diagram for FPGA implementation

Figure 5.1 captures the simplified block diagram of the FPGA implementation. All the clock signals are generated based on the 50MHz clock signal "CLOCK_50" produced by an oscillator of the DE2-115 board. The two COUNTER blocks output 12 bit numbers n_A and n_B which are then corrected by the CALIBRATION block. The corrected output x has 16 bit and is written in the SRAM memory by the MEMORY CONTROLLER block.

The DE2-115 FPGA board interfaces with the designed PCB through a 40-pin expansion header. The voltage level of the I/O pins on the header is adjusted to 1.8V by the JP6 jumper. Figure 5.2 captures the inputs and outputs connected to each pin of the GPIO. The FPGA takes 12 output signals of the two frequency dividers, implemented on the PBC, as the inputs and provides the reset, gate clock signals and the dither sequence p and $\bar{p}$. The DE2-115 Control Panel is used to performs all data transfers between the FPGA board and a host computer via the USB Blaster link. Readers are encouraged to refer to the DE2-115 user manual for more information.

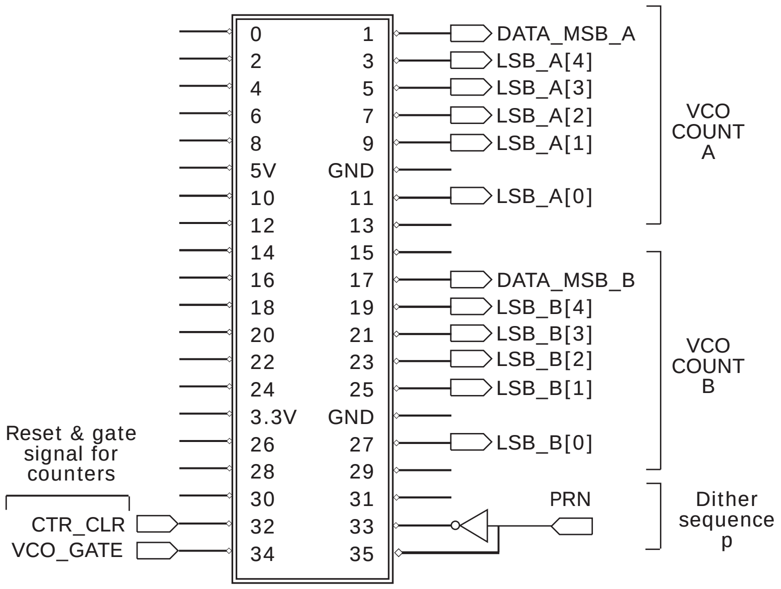


Figure 5.2: GPIO input, output configuration

5.2 Clock Signal Generator

The first parameter that needs to be determined when implementing a clock generator is the sampling frequency f_s. For this project, since the frequency range of the VCO output is around 100MHz, in order to realize a 10 bit ADC with a redundancy factor $R = 2$ the sampling frequency is required to be:

$$f_s = \frac{\Delta f_{VCO}}{2^N R} \approx \frac{100\text{MHz}}{2 \cdot 2^{10}} \approx 48.8\text{MHz} \tag{5.1}$$

Figure 5.3 shows the CLOCK GENERATOR block. There are a total of 7 output

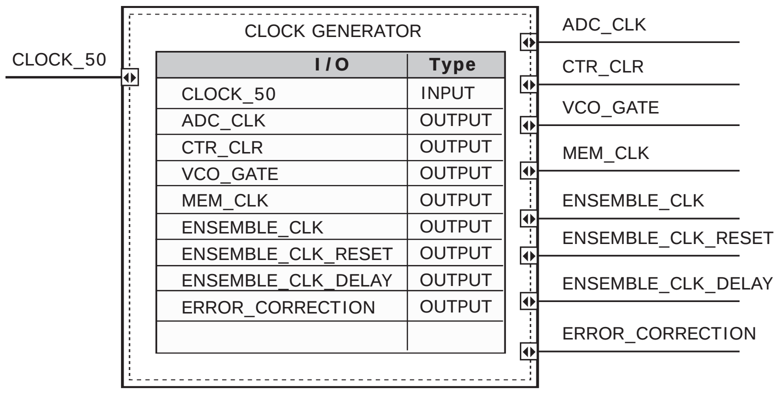


Figure 5.3: GPIO input, output configuration

signals produced which are listed below:

- ADC_CLK: 48.8kHz main conversion clock of the ADC
- MEM_CLK: 48.8kHz clock to control the MEMORY_COTROLLER block
- CTR_CLR: 48.8kHz reset signal to reset the counters
- VCO_GATE: 48.8kHz gate window to block the VCO oscillated signal, "freezing" the counter outputs while the ADC samples data.
- ENSEMBLE_CLK: 48.8kHz/K clock to sample all the LUT error terms every K conversions
- STITCH_CLK: 48.8kHz/K clock to perform the "stitching" error estimation for all unused LUT locations
- ENSEMBLE_CLK_DELAY: 48.8kHz/K clock to updates the LUT every K conversions

All the clock signals are generated based on the 50MHz reference clock of the FPGA. The simplest way to divide a clock frequency is to count the number of rising edge of the reference clock. The maximum count of the counter determines the frequency of the output signal:

$$f_{out} = \frac{f_{REF}}{\text{Max count} + 1} = \frac{50\text{MHz}}{\text{Max count} + 1} \tag{5.2}$$

In this project, a 10 bit counter with maximum output count of 1023 is used to realize the frequency of 48.8 kHz. To get the desired clock signals, the output of the counter is then compared to a reference count value. This value determines the relative delay of the output clocks. Figure 5.4 shows the timing diagram of all the input and output signals to the CLOCK GENERATOR block.

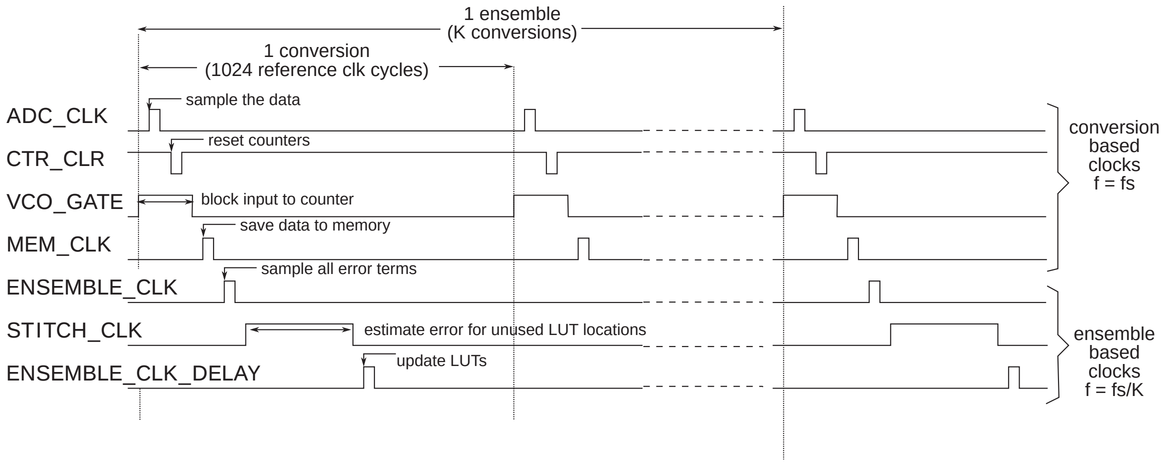


Figure 5.4: GPIO input, output configuration

When implementing a complex clock network, eventually the design will introduce physical gates in the clock paths to control the downstream clocks. This is referred as "clock gating". These gates could introduce significant delay and cause large clock skew and lead to setup and hold-time violations [16]. Although the FPGA has a 50MHz low skew reference clock, it is not flexible enough to represent the clocking needs of a sophisticated design. When the global clock lines cannot naturally accommodate these physical gates, the place and route tools will be forced to use other on-chip routing resources for the clock networks with inserted gates, usually resulting in large clock skews between different paths to destination registers. Additionally, using the detecting the rising edge of the clocks shown in Figure 5.4 is difficult. The main reason is because these clocks have very small duty cycles of less than 1%. Therefore, they might have relatively high noise and jitter. An unwanted ripple in the waveform my be miss-detected as a rising edge, resulting in inconsistency in the design.

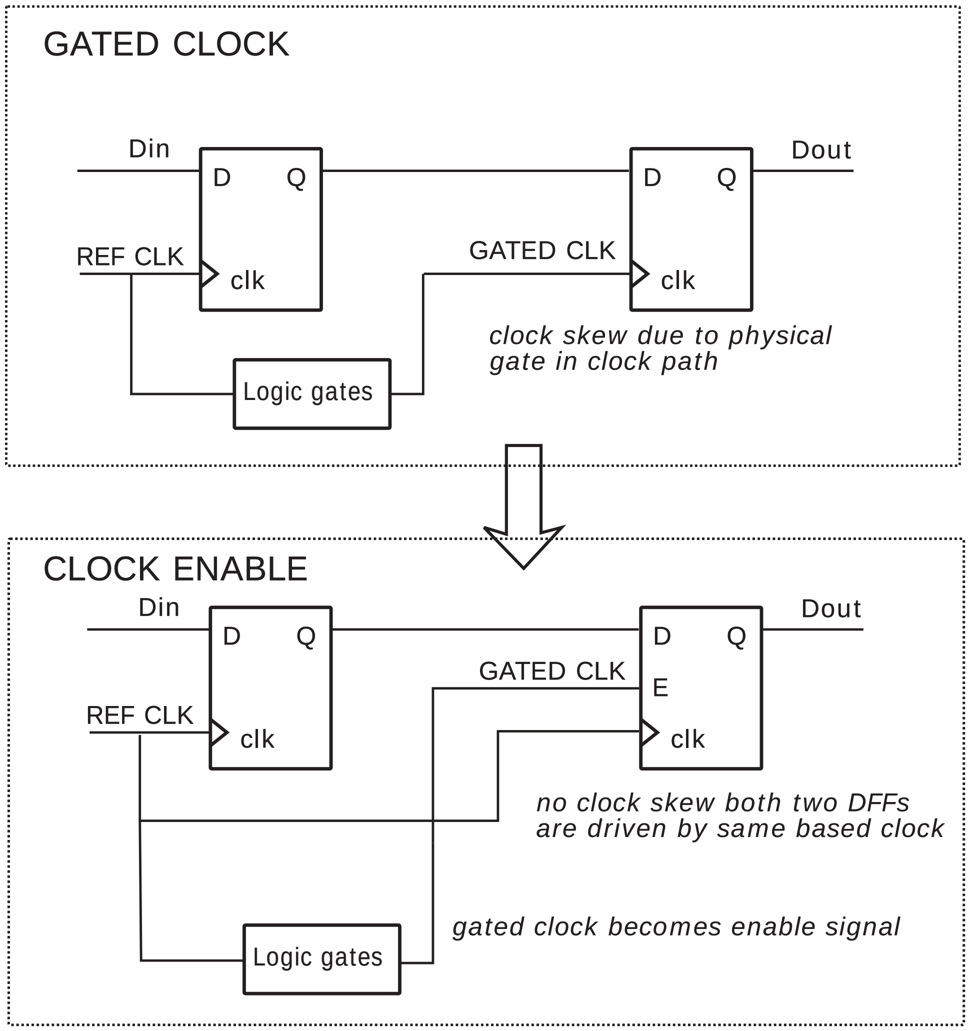

GATED CLOCK
Din
D
Q
REF CLK
clk
GATED CLK
D
Q
Dout
clk
clock skew due to physical gate in clock path
Logic gates
CLOCK ENABLE
Din
D
Q
REF CLK
clk
GATED CLK
D
Q
E
clk
Dout
no clock skew both two DFFs are driven by same based clock
gated clock becomes enable signal
Logic gates

Figure 5.5: Converting gated clocks to clock enables to eliminate clock skews

To overcome the two problems above, the "clock enable" method, illustrated in Figure 5.5, is applied in this project [16]. The gated clock is used as an enable signal of the flip flops. When the gated clock is to be switch "on", the sequential elements will be enabled and when the clock is to be switched "off", the sequential elements will be disabled. All the operations are performed at the rising edge of the reference clock, therefore, eliminating the large clock skews. Additionally, since the circuit functions when the the gated clock is high rather than at the rising edge of the gate clocks, it also solve the rising edge miss-detection problem mentioned earlier.

5.3 Dither Generator

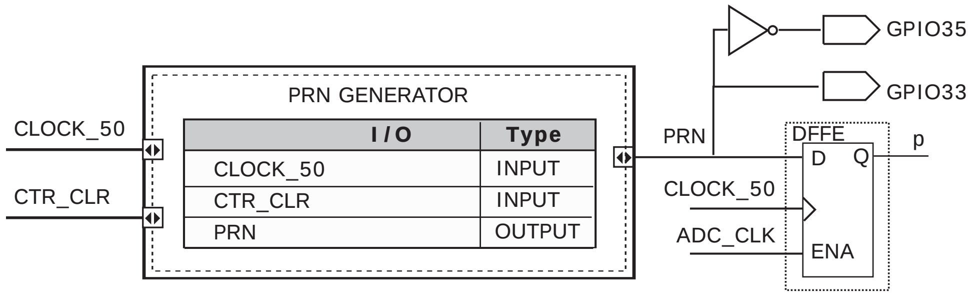


Figure 5.6: PRN GENERATOR block

Figure 5.6 shows the PRN GENERATOR block. One challenge in implementing the PRN GENERATOR block is that the pseudo-random sequence is used for both analog input and digital calibration. For each conversion, the dither is added at the beginning and remains unchanged during the conversion time period T_{CONV}, while the digital calculation corresponding to that conversion is performed in the next period, after the output counts of the counters are sampled. In Figure 5.6, PRN is the pseudo-random sequence that is connected to the GPIO output pin for the analog side of the ADC; whilse p is one period delayed version of the PRN needed for digital calculation. Two clock signals are used: CTR_CLR and ADC_CLK to implement these two pseudo-random signals. The PRN GENERATOR block generates an output PRN at every falling edge of the CTR_CLR signal. This output is then sampled using a D-flip flop, which is enabled by the ADC_CLK clock, to get p. The idea of this can be explained by the timing diagram shown in Figure 5.7

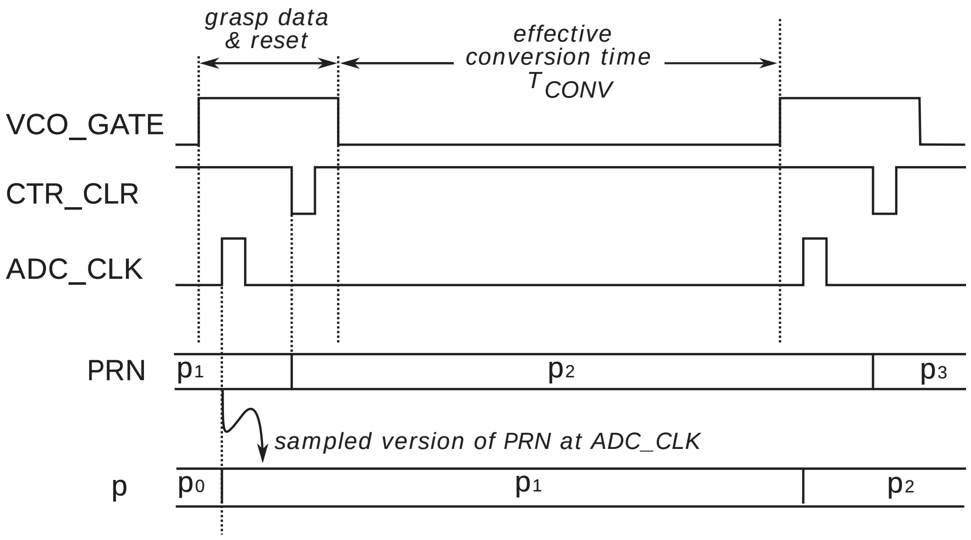


Figure 5.7: Pseudo-random signal timing diagram

The simplest pseudo-random bit sequence generator is the feedback shift register [19]. A shift register of length m bits is clocked by the CTR_CLR signal. The reference clock is utilized to convert the gated-clock to enable clock as discussed earlier in Section 5.2. An exclusive-OR gate generates the serial input signal from the exclusive-OR combination of the n^{th} bit and the last m^{th} bit of the shift register. The output of this circuit goes though a set of states and eventually repeating itself after K clock cycles. The length K of this sequence can be maximized and controlled by changing the feedback tap n and the length of the shift register m. For this project K is chosen to be equal to the ensemble sized of 1024. The corresponding values for m and n are 10 and 7, respectively [19]. Figure 5.8 shows the digital circuit implementation in FPGA of the feedback shift register. When all the outputs of the flip flops are low, the circuit is trapped at this state and the output is always 0. In order to avoid this problem, an extra OR combination of all 10 bits is used as the input preset signal to the flip flops. When all 10 bits are 0, the flip flop outputs are reset to 1's.

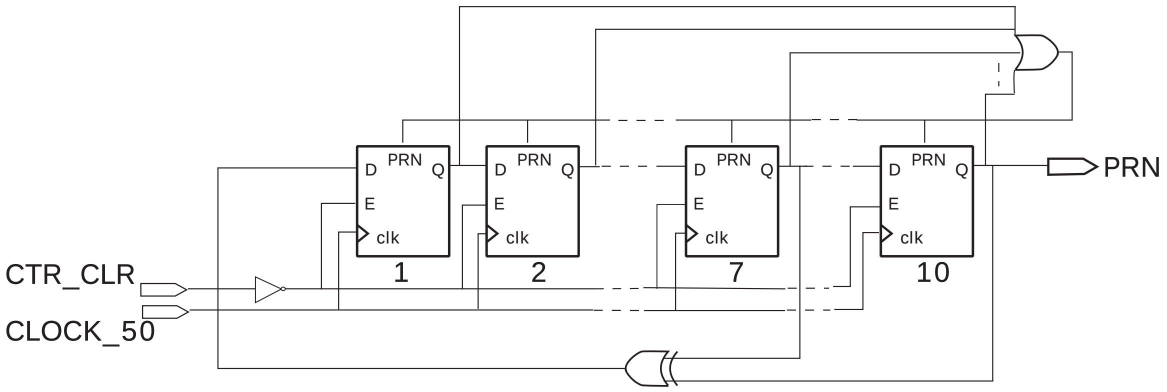


Figure 5.8: Pseudo-random signal timing diagram

5.4 Counter

As discussed in Section 4.2 the first 6 LSBs of the VCO output count n are counted by a frequency divider implemented in the PCB. Therefore, the FPGA only need to take care of the 6 MSBs. Figure 5.9 shows the COUNTER block with input and output definitions. The same rippled counter structure shown in Figure 4.6 is implemented in FPGA for both channels. The counters are reset when the CTR_CLR signal is low. All 12 bits of the output count are then sampled by the ADC_CLK clock signal and stored in a two 12 bit registers n_A and n_B. The LSB[4..0] and MSB_DATA signals are extracted from the 6 bit output of the PCB counters connected to the FPGA board via the GPIO port.

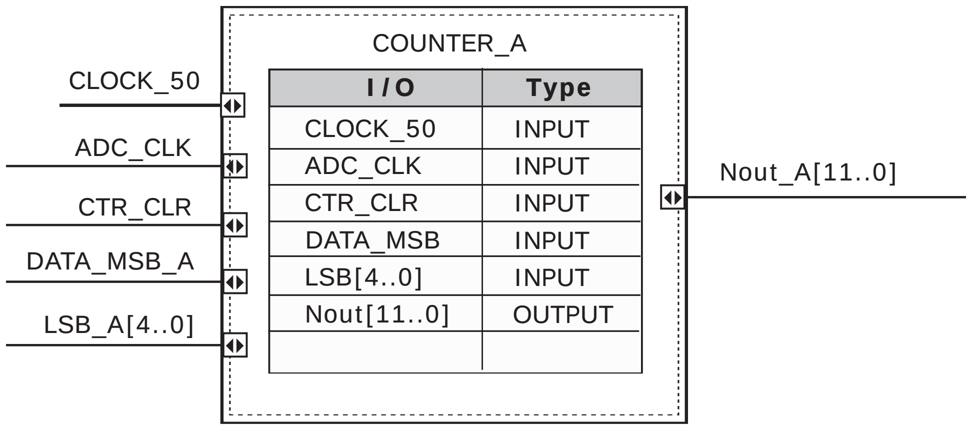


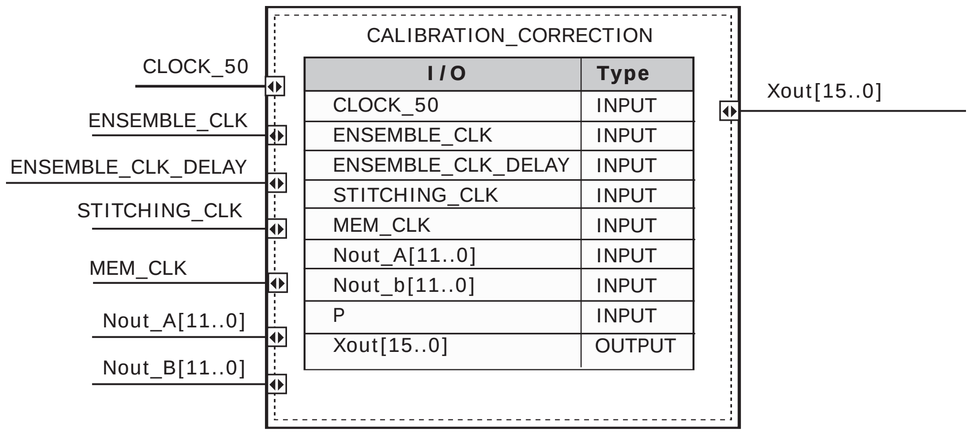
Figure 5.9: Counter Block

5.5 Calibration and Correction Block

After the clock output of the VCO is processed by the frequency divider and the COUNTER block, the raw output count is then corrected by a LUT which is calibrated digitally in background. These functions are performed by the CALIBRATION block shown in Figure 5.10. The main two input data of the block is the two raw output count $Nout\_A$ and $Nout\_B$ of the two counters which are sampled by the ADC_CLK clock at 48.8 KHz. All the other inputs are the clock signals controlling the timing of the correction and calibration process. The corrected 16 bit output X_out is then saved in the memory SRAM every conversion. In order to understand the operation as well as the timing of the block, this section firstly describe the LUT and error matrix implementation in FPGA. After that all the calculations which occur every conversion is discussed, followed by those which occur in updating calibration parameters after every ensemble set of K conversions.

5.5.1 LUT and Error Matrix Implementation

There are several ways of implementing a LUT in Verilog code. One could be to use a simple *case* statement which is synthesized to purely combinational digital logic



I / O	Type
CLOCK_50	INPUT
ENSEMBLE_CLK	INPUT
ENSEMBLE_CLK_DELAY	INPUT
STITCHING_CLK	INPUT
MEM_CLK	INPUT
Nout_A[11..0]	INPUT
Nout_b[11..0]	INPUT
P	INPUT
Xout[15..0]	OUTPUT

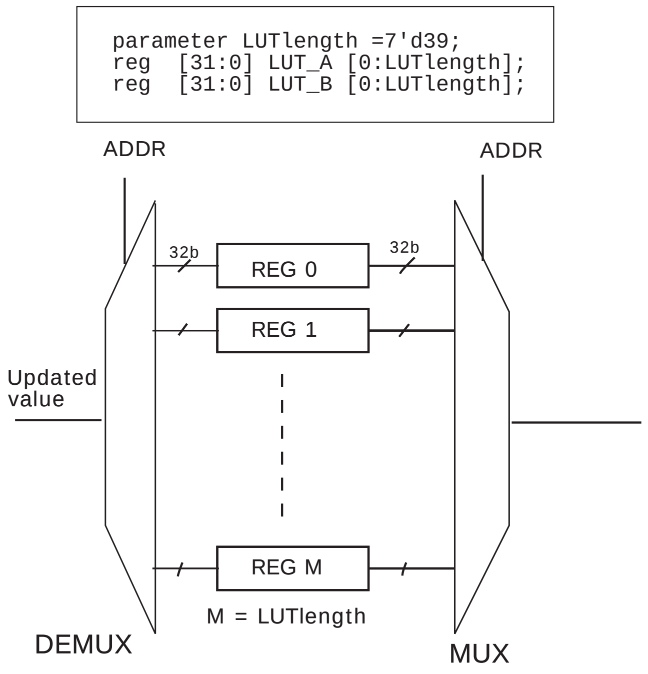
Figure 5.10: Calibration and Correction Block

elements [16]. The difficulty with this approach is that the LUT coefficients are fixed and cannot be updated easily. Another method which is more appropriate for this project is to implement the LUT as a memory where the LUT values are stored in registers. These values can be referred by a multiplexer (MUX) and updated via a demultiplexer (DEMUX). The example Verilog code for the LUT and the synthesized digital circuit diagram is shown in Figure 5.11.

The same structure is used to implement the matrix errors ε. In order to avoid the rounding error due to all digital calculations, precisions of all LUT and error matrices are set to be 32 bits. To estimate the error matrices, three error terms are needed for each channels.

- **rawea**, **raweb**: The raw error matrices that is updated every conversion.
- **ea**, **eb**: The sampled version of **rawea** and **raweb** updated every ensemble when the ENSEMBLE_CLK signal is high.
- **eavalid**, **ebvalid**: The final error matrix that includes the “stitching” algorithm to preserve the LUT continuity

While, simply, all the error matrices are set to zero, several important factors



```
parameter LUTlength =7'd39;
reg  [31:0] LUT_A [0:LUTlength];
reg  [31:0] LUT_B [0:LUTlength];
```

Figure 5.11: Code and synthesized digital circuit for LUT implementation

need to be considered when initializing the LUT. Intuitively, the LUT contains all the output $x = a_i$ corresponding to the uncorrected count $n = \mathrm{MSB} \cdot 2^L$, which has all the LSBs equal to zero. The most straightforward way to initialize the LUT is to set the output equal to the input code; and the MSB of the input code is equal to the LUT location i. However, since the uncorrected count n does not start from zero but from a minimum value n_{min}, there would be a unused LUT region from location 0 to location MSB_{min}. Therefore, to eliminate this redundant region, the address to the LUT is defined as

$$\mathrm{LUTlocation} = i = \mathrm{MSB} - \mathrm{MSB}_{min} + 1 \tag{5.3}$$

where MSB_{min} is the minimum MSB value that the uncorrected code n ever has for a full-scale input signal. Since the LUT has 20 more precision bits compared to the uncorrected count, the LUT coefficients are initialized as:

$$a_i = i \cdot 2^L * 2^{20} \tag{5.4}$$

Note that the 2^{20} factor does not imply a scale factor of 2^{20} but indicates the 20 precision bits of the LUT coefficients, as the radix point of the final output codes x will be shifted to the correct position using MATLAB. The next section describes the detailed calculations for the LUT correction and calibration block.

5.5.2 Conversion Based Calculations

This section describes all the calculations that occurs every conversion in the calibration and correction block. These include the linear interpolation and error estimation. The linear interpolation is performed solely by combinational logic. In other words, no clock signal is needed to control the timing of these operation. Figure 5.12 shows a synthesized circuit diagram of the correction block. Firstly, the MSB and LSB are recorded based on the count n. The MSB is then used as the control signal to the MUX to determine the LUT coeffients a_i and a_{i+1}. Then the linear interpolation in Equation 3.1 is performed using a multiplier and a shift register. The same structure is used for B channel. After that the final output code

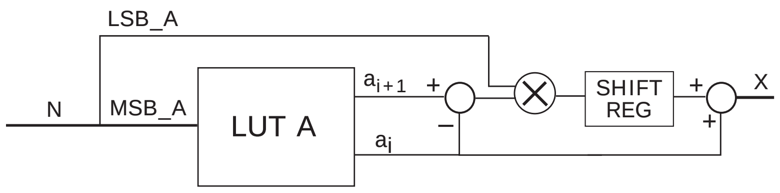


Figure 5.12: Synthesized digital circuit for the correction block

is calculated by taking the average of the two output x_A and x_B. The error term *deltax* is also extracted as:

$$deltax = x_B - x_A - 2pD \tag{5.5}$$

This equation can be realized by a digital circuit as shown in Figure 5.13. The two values 2D and -2D are two parameters that are initially stored in two registers. The pseudo-random signal p is served as the control signal to a multiplexer to indicate the correct sign of $\pm 2D$. The calculated value *deltax* is then utilized to in the error estimation process which is discussed next.

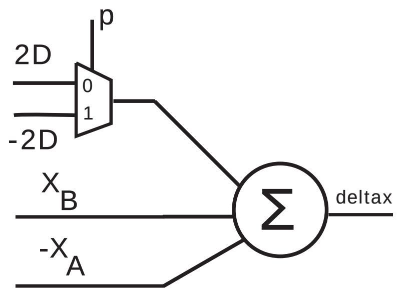


Figure 5.13: Synthesized digital circuit for the correction block

Since the input n is clocked by the ADC_CLK signal, the output x and *deltax* is expected to change its value after a time delay compared to the ADC_CLK. Therefore, all the next operations to estimate the error are timed by the clock MEM_CLK which is a delayed version of the ADC_CLK. When the MEM_CLK signal is high, the following calculations are performed:

- Update the **rawea** and **raweb** matrices by:

$$rawea[i] \leftarrow rawea[i] - deltax(2^L - \text{LSB\_A}) \tag{5.6}$$

$$raweb[i+1] \leftarrow raweb[i+1] + deltax\text{LSB\_A} \tag{5.7}$$

$$raweb[j] \leftarrow raweb[j] - deltax(2^L - \text{LSB\_B}) \tag{5.8}$$

$$raweb[j+1] \leftarrow raweb[j+1] + deltax\text{LSB\_B} \tag{5.9}$$

- Track the used LUT location by:

$$hit\_A[i] = 1'b1 \tag{5.10}$$

$$hit\_A[i+1] = 1'b1 \tag{5.11}$$

$$hit\_B[j] = 1'b1 \tag{5.12}$$

$$hit\_B[j+1] = 1'b1 \tag{5.13}$$

The two arrays *hit_A* and *hit_B* track which LUT locations have been hit during one ensemble. Any zero entry in the arrays indicates an unused LUT location. These arrays will be utilized to estimate the error terms of the unused LUT region to preserve the LUT continuity.

5.5.3 Ensemble Based Calculations

In order to understand all the calculations that occur every ensemble, a timing diagram of all the clock signals is first investigated. There are totally 4 clock signals controlling these ensemble based operations as shown in Figure 5.14

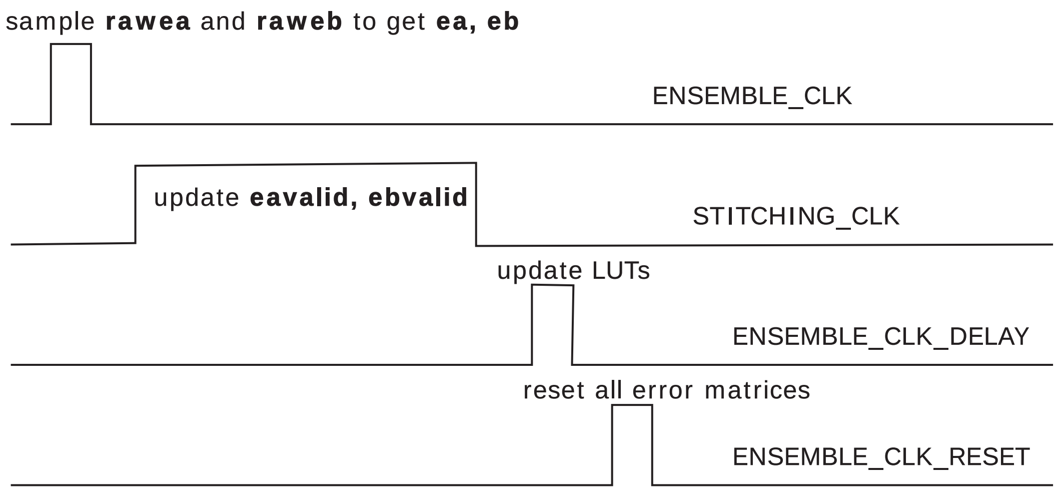


Figure 5.14: Ensemble based clock signals

After K conversions, the **rawea** and **raweb** error matrices are then sampled by a set of D-flip flops enabled by the ENSEMBLE_CLK clock signal to get **ea** and **eb**. Before the LUTs are updated at the rising edge of ENSEMBLE_CLK_DELAY, the "stitching" algorithm presented in Section 3.2.5 is applied to estimate the error terms for unused LUT locations. The results are stored in the **eavalid** and **ebvalid**. Finally, all the error matrices needed for the calibration is reset by the ENSEMBLE_CLK_RESET clock.

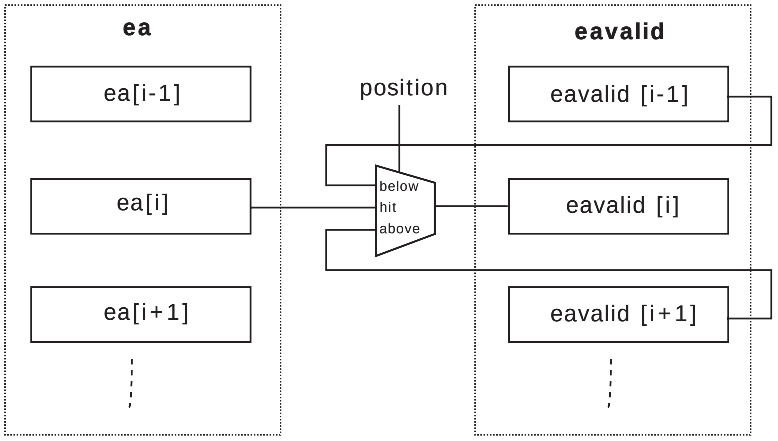


Figure 5.15: Block diagram to realize the "stitching" algorithm

To realize the "stitching" algorithm, the structure shown in Figure 5.15 is utilized. A 3:1 multiplexer is used for each entry of the error matrix. The idea of this approach is that, if the LUT location i is hit, then the corresponding estimated error term **ea**$[i]$ is valid and is copied to the **eavalid** matrix. On the other hand, if the i^{th} entry is not used and is above the region covered by the signal range, no information is gathered for that location; therefore **ea**$[i]$ equals 0. In this case, the value stored in the right below location $(i-1)^{th}$of the **eavalid** matrix is copied to the i^{th} entry. It is not necessary that the $(i-1)^{th}$ location would be covered by the input signal. However, after several clock cycles, the valid error term of the top used LUT location will be copied to all the above locations. The LUT region that below the signal range is also updated in a similar manner. All of these operations operate at the rising edge of the 50MHz reference clock and is enabled by the STITCHING_CLK signals. Thus, it is required that the STITCHING_CLK signal needs to be high for a long enough time in order for the signal to propagates through all the unused LUT locations.

A critical part of this design is to determine whether the LUT location is within or above or below the signal range. As discussed in Section 5.5.2 the two number *hit_A* and *hit_B* can track if the LUT location is hit or not. However, to distinguish the "above" and "below" case the two additional *above_A* and *above_B* numbers are considered. For a continuous input signal which cover a small range of the LUT, the two binary number *hit_A* and *hit_B* always have the form of

$$hit = 0 \cdots 0 \quad \underbrace{11 \cdots 1}_{\text{used region}} \quad 0 \cdots 0 \tag{5.14}$$

Taking the opposite of this gives:

$$above = -hit = 1 \cdots 1 \quad \underbrace{00 \cdots 01}_{\text{used region}} \quad 0 \cdots 0 \tag{5.15}$$

As (5.15) shows, for unused LUT locations, the corresponding bit in parameter *above* can indicates if the location is above or below the signal range. It should be noted that, the technique presented in this section to realize the "stiching" algorithm only works when the input signal is continuous. In other word, the signal never skips any LUT location; therefore, the only possible unused LUT locations are at the two ends of the LUT characteristic curve.

5.6 SRAM Controller

The DE2-115 board has 2MB SRAM memory with 16-bit data width. This SRAM can be controlled by the FPGA by the MEMORY CONTROLLER block as shown in Figure 5.16.

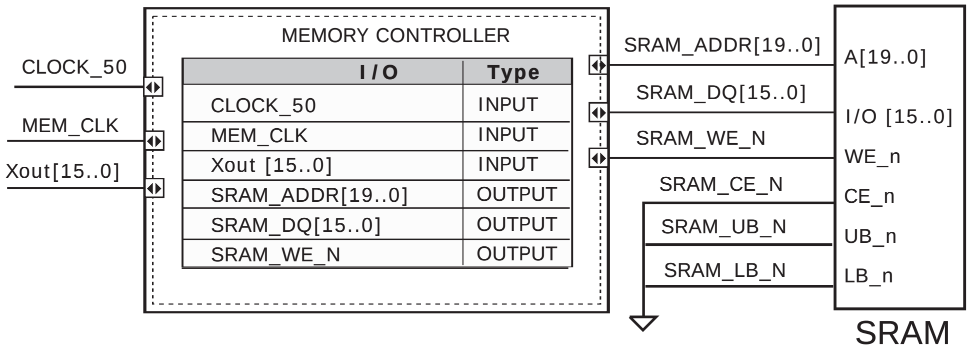


Figure 5.16: MEMORY CONTROLLER block

The operation of the SRAM is straight forward. Firstly the chip is low enabled by the *CE_n* signal; and the upper byte and lower byte are also accessed by power-down *UB_n* and *LB_n*. The 16 bit data input of the SRAM is directly connected to the corrected digital output $Nout$ via the SRAM_DQ bus. The SRAM_WE_N signal controls both writing and reading of the memory, while the address of the memory is set by SRAM_ADDR. Since the data $Nout$ is sampled by the 48.8KHz ADC_CLK clock, the address at which the data is saved also needs to increment at the same rate. However, if the address is controlled by the same clock as the data, the SRAM will record the data at the time when it changes the value; this causes uncertainty in the measurement. In order to avoid this problem, the MEM_CLK signal is used to clock the address of the memory. To increment the address and control the write enable signal, a simple counter is implemented as shown in Figure 5.17.

The first bit of the counter output is connected to the write enable signal. To visualize the operation of the SRAM, the write enable signal is also connected to

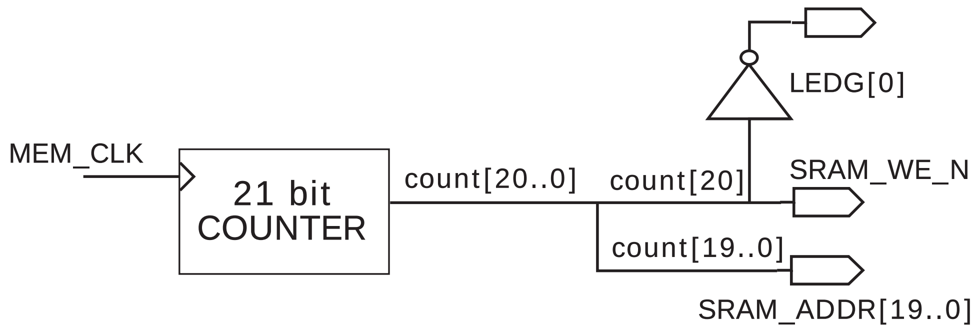


Figure 5.17: Digital circuit implementation controlling the address and write enable signal of the SRAM

an LED via an inverter; therefore, when the SRAM is in “write” mode, the LED is lighted up. The other 20 bits of the *count* output control the address of the SRAM. After the address changes and sweeps over the entire 2MB memory, the most significant bit of *count* is 1, thus disabling the write mode of the SRAM, turning off the LED. At this point, the “DE2-115 Control Panel” feature can then be activated to load the data to the host computer.

Chapter 6

Results

The split VCO-based ADC was built and the calibration technique was implemented in the DE2-115 board. The output data was collected and analyzed. This section discusses the resulted performance of the ADC. The first section investigates the offline calibration technique. The linearity of the ADC as well as the LMS loop convergence of the background calibration technique is then provided in the next section. Measure results show $\approx 10X$ improvement in the INL error of the ADC after calibrated using the background calibration technique.

6.1 Offline LUT Calibration

Offline calibration technique is not the main focus of this project since it requires the need to take the ADC offline and develop a known input signal. However, it is important to firstly implement the offline calibration to ensure the proper function of the LUT correction. Figure 6.1 captures how to estimate the LUT coefficients a_i offline in MATLAB.

Firstly, a full scale triangle analog input is used and the raw output count is then measured. After that, a corresponding desired linear digital output is estimated in

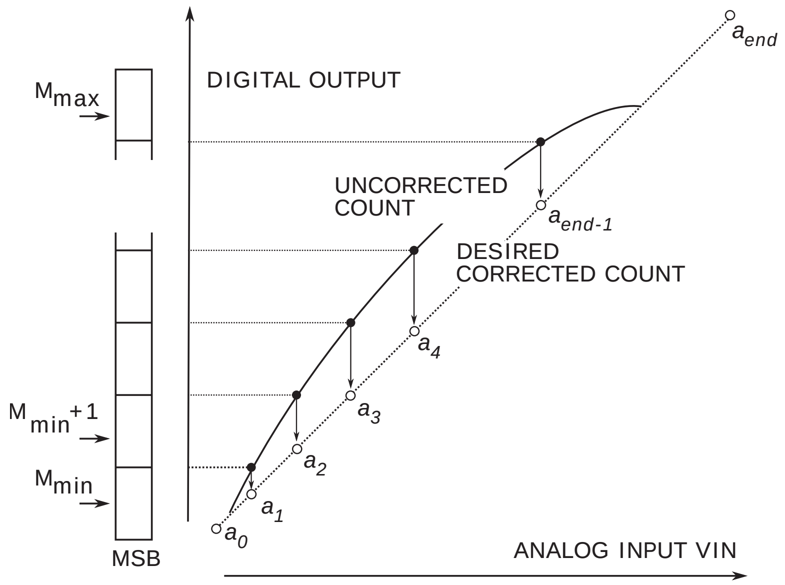


Figure 6.1: Offline calibration technique to estimate LUT coefficients

MATLAB. There are several ways to estimate this linear ramp output. One could be to do a best-fit straight line based on the measured data. Another simpler way is to form a straight line from the maximum and the minimum of the raw output count. The second method was implemented as shown in Figure 6.1. The LUT coefficients a are determined to be the desired linear output corresponding to the raw counts of MSB $\cdot\, 2^L$, where L is the number of LSBs. However, one problem with this simple approach is that the LUT can only cover the range from $n = MSB_{min} \cdot 2^L$ to $n = MSB_{max} \cdot 2^L$. Therefore, all numbers at the two end regions are not calibrated. In order the solve this problem, two more LUT coefficients at two ends are extrapolated, which are denoted by a_0 and a_{end} in Figure 6.1. Suppose the minimum measured count n_{min} has least significant bits of LSB_{min} and the maximum count n_{max} has least significant bits of LSB_{max}. Since both these two points are in the estimated straight line, linear interpolation gives:

$$n_{min} = \frac{a_1 - a_0}{2^L} LSB_{min} + a_0 \tag{6.1}$$

$$n_{max} = \frac{a_{end} - a_{(end-1)}}{2^L} LSB_{max} + a_{(end-1)} \tag{6.2}$$

Solving the two equations results in:

$$a_0 = \frac{2^L n_{min} - LSB_{min} a_1}{2^L - LSB_{min}} \tag{6.3}$$

$$a_{end} = \frac{2^L n_{max} - 2^L a_{(end-1)}}{LSB_{max}} + a_{end-1} \tag{6.4}$$

This offline calibration process was performed in MATLAB. The resulted LUT coefficients were then implemented in FPGA to correct the signal. The plot for output count before and after the calibration was shown in Figure 6.2. Six LSBs out of the 12 bits were used for the LUT initialization. The corrected output is much more linear compared to the uncorrected count.

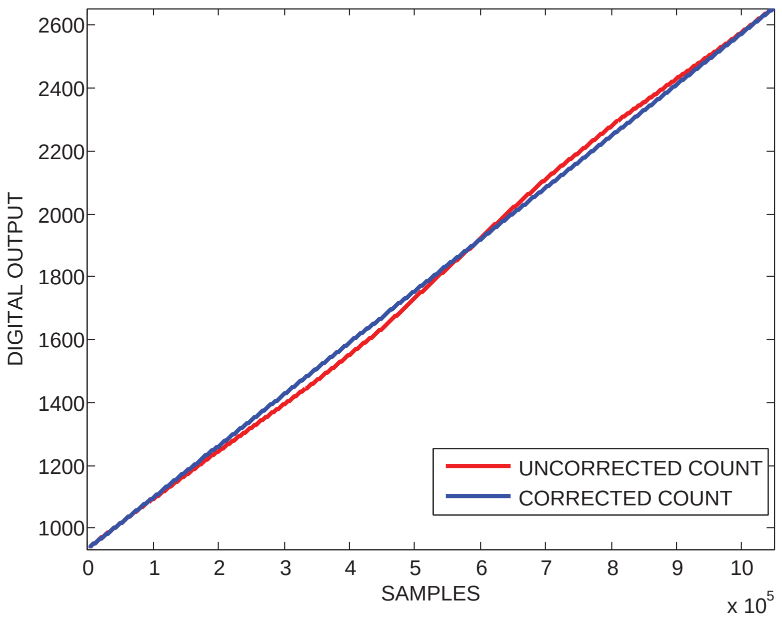


Figure 6.2: Raw output count and corrected output count using offline calibration

The histogram test was performed to evaluate the DC linearity improvement of the ADC. Different numbers of LSBs and MSBs and their effects on linearity were also investigated. Shown at the top of Figure 6.3 are the DNL and INL plots of the uncalibrated signal. The ADC exhibits very low DNL error due to the natural smooth characteristic of the VCO. However, the issue of the VCO-based ADC is the global nonlinearity which is shown by large peak INL error of -37/50 LSB. This INL error is improved greatly for calibrated output. As Figure 6.3 shows, the larger the LUT size is, the smaller the INL error. The peak INL is 3.0/-2.5 LSB for the calibrated ADC with LUT size of 15. When the LUT size is increased to 29, the INL is reduced to 2.60/-1.27 LSB; while this value is 2.0/-1.16 LSB for a 55 point LUT. Although enlarge the LUT size can improve the linearity of the ADC, it also increases the complexity and power consumption of the circuit.

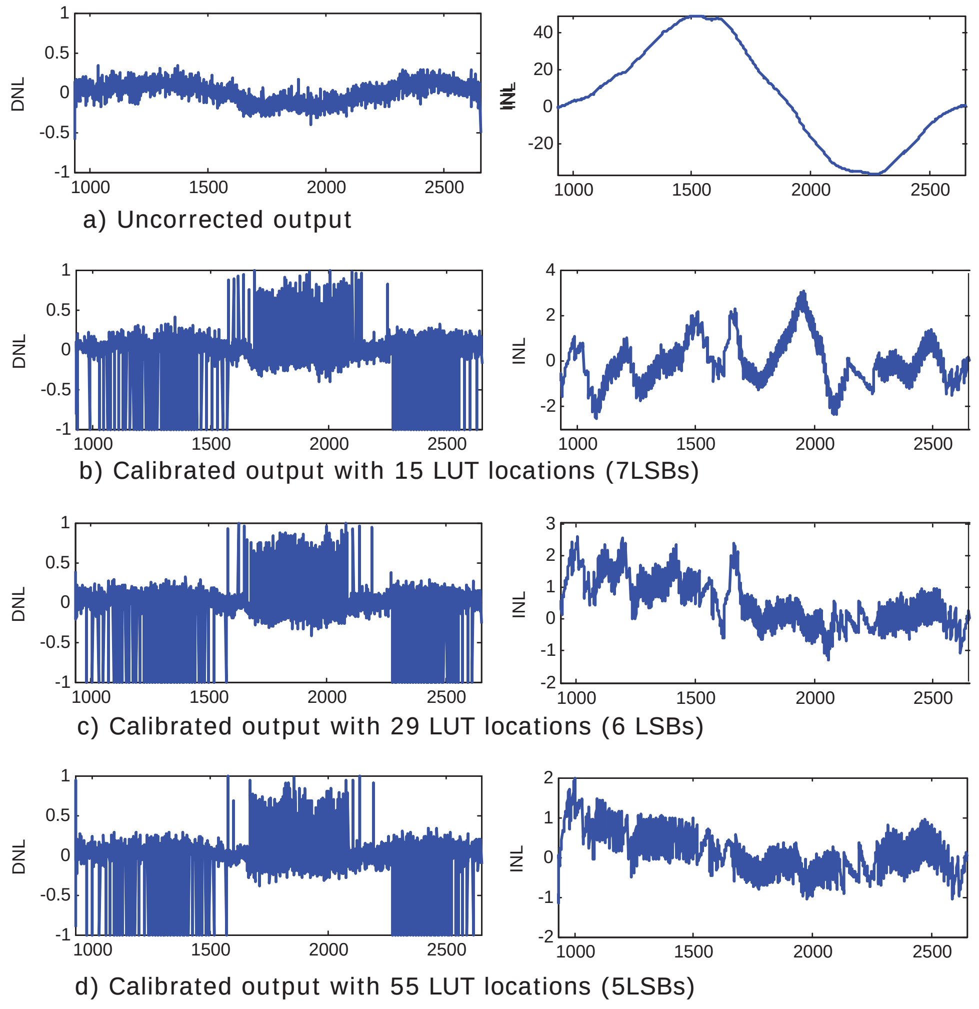


Figure 6.3: DC linearity improvement using offline calibration

Another interesting aspect can be observed from the DNL plots in Figure 6.3 is the fact that LUT correction actually "degrades" the DNL performance of the ADC. In fact, for all three LUT sizes, the peak DNL errors are about +1/-1 LSB. Figure 6.4 zooms in the corrected output waveform at the point where the DNL error is -1LSB. As the figure shows, there is a code missing from the calibrated output. This is a common problem in LUT correction when LUT transfer function has a slope greater than unity. In order to avoid missing code in LUT correction a redundancy factor R must be introduced [18].

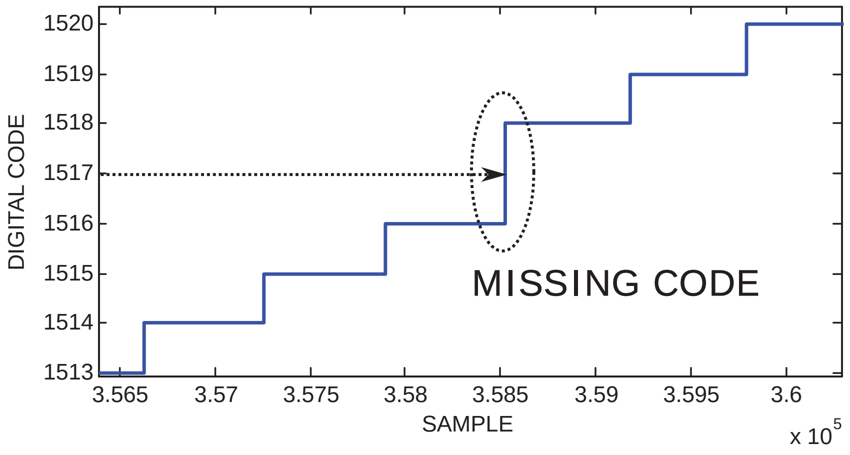


Figure 6.4: Missing code at the calibrated output

Figure 6.5 shows the DC linearity of the offline calibrated ADC with redundancy factor $R = 2$ and LUT size of 29. As the measurement indicates, introducing redundancy factor of 2 can reduce both the DNL and INL error by the same factor, thus eliminating the missing code issue. In fact the peak DNL is only around ± 0.5 LSB and the INL is less than ± 1 LSB

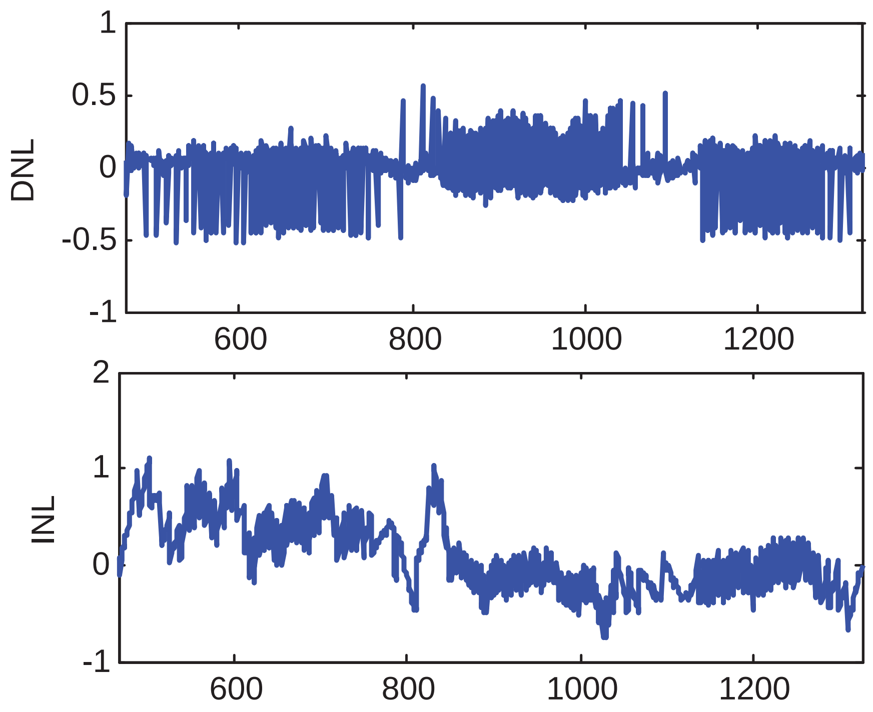


Figure 6.5: Introducing R = 2 reduces DC linearity by factor of 2

6.2 Background LUT Calibration

This section provides measured result of background calibration technique. Background correction and calibration were performed using a DE2-115 FPGA as discussed in Chapter 5. Six MSBs were used as the address to the LUT; and six LSBs were used for the linear interpolation. The ADC was able to perform the calibration and correction in digital domain transparently to the main signal path. Although a LUT size of 40 was used, only 30 LUT locations were covered by the signal range. The corrected output is stored in the SRAM and analyzed using MATLAB. Firstly, the DC linearity of the calibrated ADC is presented in Section 6.2.1. Section 6.2.2 then covers the LMS loop convergence. Finally, a pathological failure in the calibration algorithm, resulting in a divergence in the LMS loop is discussed and the solution is provided.

6.2.1 DC Linearity

The calibrated digital output of a full scale triangle analog input is shown in Figure 6.6. The digital output range is from 360 to 1860 corresponding to analog voltage range of 1.64V - 2.05V. It should be noted that the positive offset of 360 of the output is desired in order ensure that the minimum LUT coefficient is positive.

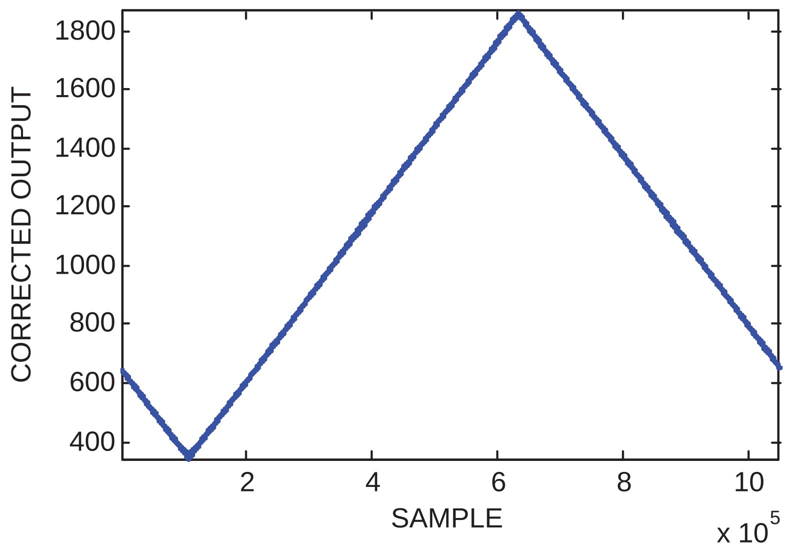


Figure 6.6: Digital output of a full-scale triangle wave input

Figure 6.7 captures the DC linearity reported at 12 bit level of the calibrated ADC. Compared to the INL of the uncorrected signal of -37/50 LSB as shown in Figure 6.3a, the INL for corrected signal is improved by a factor of $\approx 10X$. Measurement shows a peak INL error of +5.6/-3.3 LSB and peak DNL of +0.37/-0.82 LSB. For a 10 bit level (redundancy factor R = 4), the linearity error would be reduced further by a factor of 4.

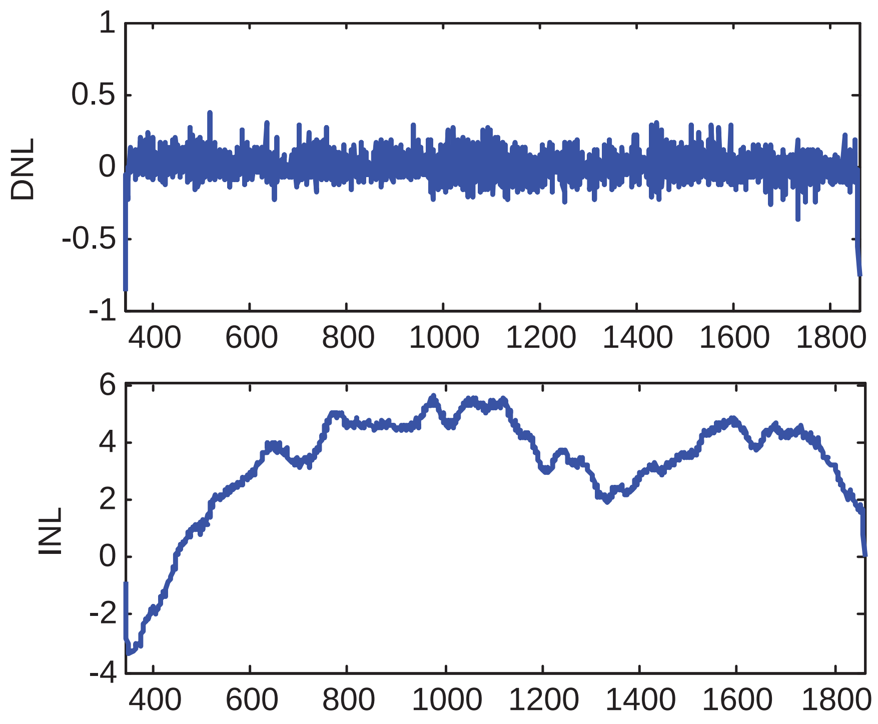


Figure 6.7: DC linearity error of the background calibrated output

6.2.2 LMS Convergence Investigation

To test the LMS convergence loop, the analog input was set to a DC value and the adaptation transient of the LUT coefficient was observed. The top plot in Figure 6.8 shows a particular LUT coefficient for different values of LMS parameter μ, while the bottom one shows the corresponding error term $deltax = x_B - X_A - 2D$. The final LUT values of three cases are different as the condition varies for each measurement. However, all the LUT coefficients converge to a value that drives the error term $deltax$ to zero. The response exhibits stable behavior with no overshoot or ringing. The convergence response indicates the usual trade off for an adaptive LMS loop. Smaller values of μ grant a more stable but slower loop convergence. For all three different μ values, the LUT converges within 100K conversions. Another interesting aspect of the LMS convergence loop is that the ensemble size has no effect on the speed of the convergence. Larger ensemble size can accumulate a larger error terms; on the other hand, it also decreases the frequency, at which the LUT is updated, by the same factor. For this project, the ensemble size was chosen to be 1024 conversions.

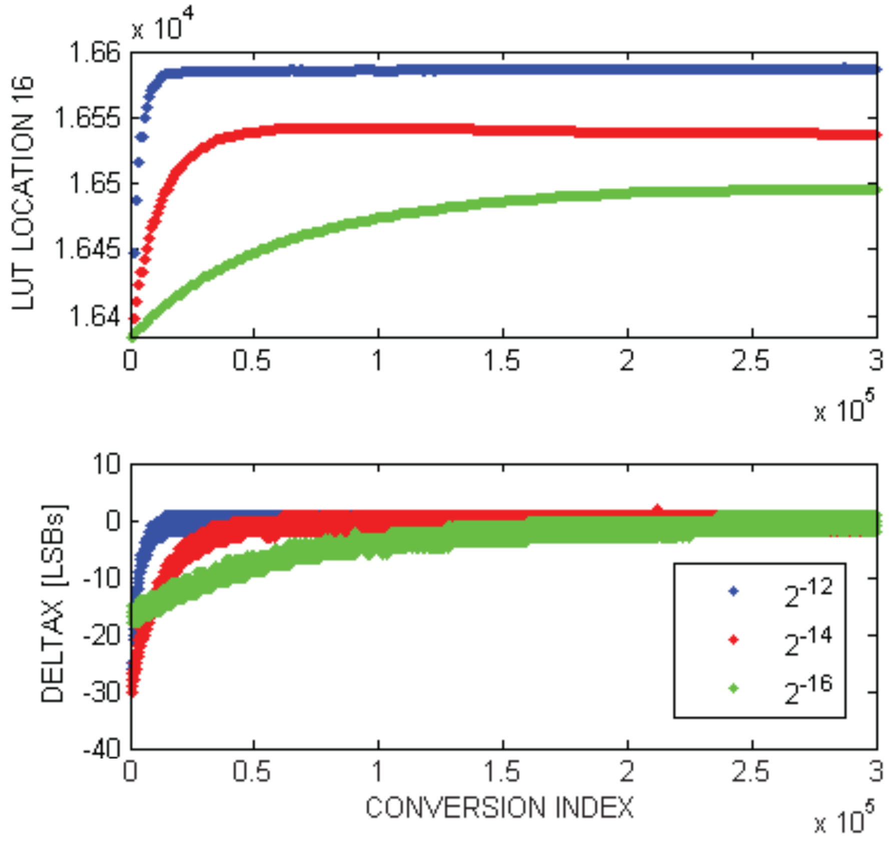


Figure 6.8: LUT convergence for different parameter μ

6.2.3 Divergence in LMS loop

As the calibration algorithm was implemented in FPGA, there were several situations when the algorithm diverges. One of them is when the LMS parameter μ is larger than 2^{-10}; another one is when the analog signal range is small such that only a few LUT locations were covered. One possible reason is that both these two cases result in a large estimate of the error terms which are then subtracted by the LUT coefficients. Relying too much on the accuracy of the estimated error can cause the loop to diverge. However, there is another more important issue with the calibration technique that leads to this divergence problem. Shown in Figure 6.9 is the measured digital output of a triangle wave input when the calibration diverges. At the bottom of the figure is the zoomed in version of the waveform. As the figure shows, even though the overall characteristic of the output looks linear, the divergence problem occurs locally and periodically for each small segments of the curve.

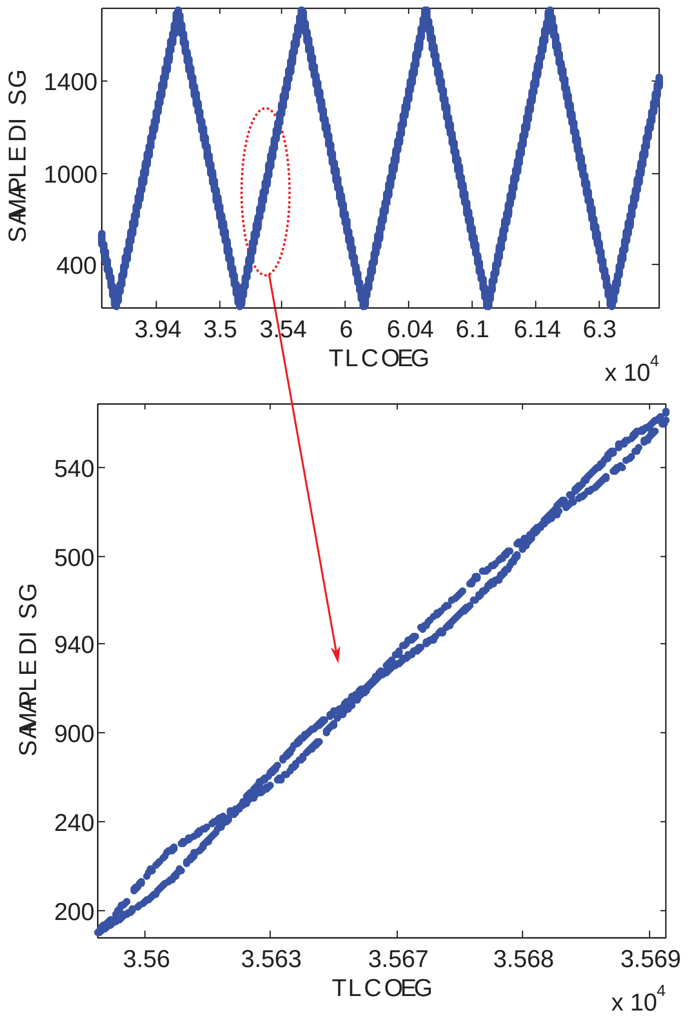


Figure 6.9: Digital output of the triangle wave input when the algorithm diverges

To understand this problem, it is important to readdress the calibration technique implemented in this project. The analog signal is split into two channels which share the same structure, and are expected to produce the same output codes. Therefore, the difference between the two outputs can be used to align the two characteristics. However, since this simple technique has no vision on the slope of the characteristics, the analog inputs to these two channels are intentionally separated by $2\Delta V$ corresponding to a known output code of $2D$ by introducing the dither. The calibration process is now based on the error term $x_A - x_B - 2D$. By doing this, slope of "ADC A" at $V_{in} = V$ can be "aligned" to the slope of "ADC B" at $Vin = V \pm \Delta V$ which then again turns out the be aligned to slope of channel A at $Vin = V \pm 2\Delta V$. Sweeping through all value of V can ensure the same constant slope of the two characteristics.

However, this intuitive reasoning does not hold true in every case, particularly for a LUT implementation in which the slope of the transfer function is inherently discontinuous between two adjacent segments. Moreover, in this project the distance between two LUT locations was accidentally set up to be twice the output dither size D. This choice of the LUT spacing turns out to be the main reason underlying the divergence issue. Shown in Figure 6.10 are the outputs of the two channels when the calibration technique actually converges to an unstable mode; the slopes of the two characteristics periodically changes for every LUT segment. In the figure the dashed red segment (ADC A) and the corresponding dashed blue segment (ADC B) are separated by $2\Delta V$ in the voltage domain and $2D$ in digital domain; similarly for the solid segments. Thus the "slope alignment" condition mentioned above is still preserved. However, the two adjacent segments of both the two channels have different slopes; and the average of these two slopes is equal to the desired value. As the error estimation process slightly moves the dashed segments to a wrong direction, all the solid ones are also incorrectly adjusted, and vice versa, leading to the divergence of the whole LUT.

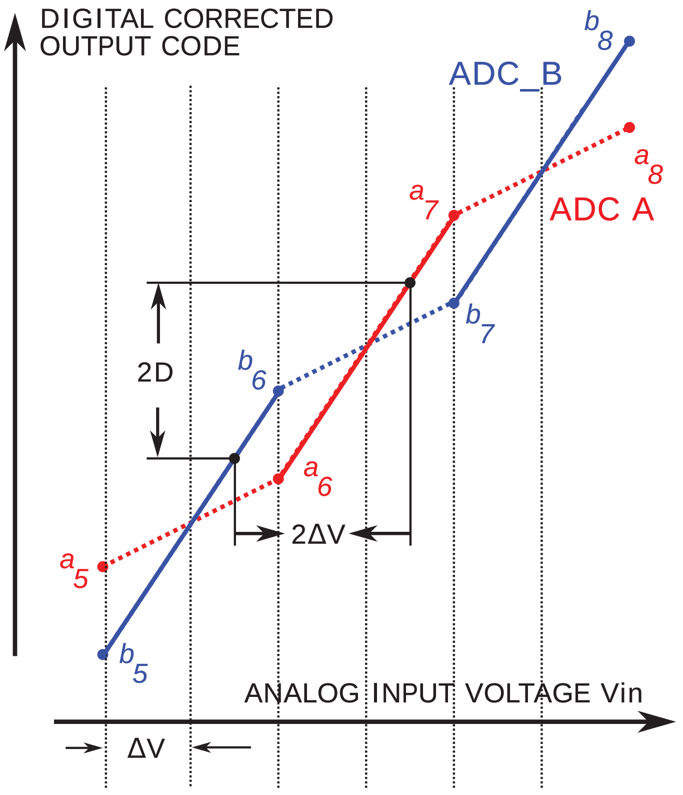


Figure 6.10: Digital output of two channels when the algorithm diverges

This problem can be fixed easily by reducing the dither size ΔV so that when the analog signal steps by $\pm\Delta V$ the calibration block can "see" the difference in the slopes of the dashed lines and the solid lines and corrects them. Another more general solution is not to make the dither size $2D$ to be a multiple of the LUT spacing. This can avoid the periodic behavior of the slope discontinuity of the LUT. In other word, when the analog is swept through all LUT locations, as one channel move to another LUT segment, changing the slope, the other should not. Therefore, the slope variation of two adjacent segments can be visualized and calibrated.

Chapter 7

Conclusions

The design of a 10b VCO-based ADC was presented in this project. A background calibration and correction technique to linearize the VCO-based ADC was implemented in an FPGA. Chapter 2 and Chapter 3 provided a detailed background on the VCO-based ADC architecture and the calibration technique. An improvement in the algorithm compared to [1, 2, 3] to preserve the LUT continuity was introduced. The detailed implementation of the ADC in the PCB and FPGA is discussed in Chapter 4 and Chapter 5. Chapter 6 discusses the resulted measurement of the ADC regarding its DC linearity and LMS loop convergence. A divergence issue of the calibration technique was detected and a solution was proposed.

Table 7.1 summarizes the design specifications of the VCO-based ADC implemented in this project. Two VCOs with an input range of 1.63V - 2.05V and a frequency range of 40MHz - 140MHz were used. Two 12 bit counters were implemented to count the number of phase transitions of the VCO output. The ADC was operated at a sampling frequency of 48.8KSps.

The calibration and correction can correct the output in digital domain, transparently to the main signal path. Although offline calibration shows a great im-

Table 7.1: VCO-based ADC System Parameters / Results

PARAMETER / RESULT		VALUE	UNITS
VCO	Input Range Used	1.63 - 2.05	V
	Frequency Range	40 - 140	MHz
ADC	Resolution	10	bits
	Sample Rate f_S	48.8	KSps
	Dither ΔV	1/32	FS
	INL	+5.6 / -3.3	LSB
	DNL	+0.37 / -0.82	LSB
LUT	Size	40	points
	Counts / Segment	64	counts
LMS Loop	Parameter μ	2^{-15}	
	Convergence Time	100k	convs
	Ensemble Size	1024	convs
Internal Digital Precision		32	bits

provement in the linearity of the ADC, it cannot adapt to any increases in the input signal ranges or changes in the VCO characteristics due to temperature. Moreover, the ADC needs to be taken offline; and the known input signal is required.

Background calibration technique was implemented in the FPGA. All calculations in the digital domain were performed in 32 bit level precision. When the loop converged, the ADC showed $\approx 10X$ improvement in the INL error. For parameter μ smaller than 2^{-10}, the algorithm was able to converge stably within $100K$ conversions.

When the dither size $2D$ is equal to an integer number of LUT spacing, the algorithm can converge to an unstable mode where slope of the ADC characteristic toggles between 2 values for every two adjunct segments. The size of these segments is determined by dither size $2D$. Eventually, this leads to the divergence of the whole LUT. The LUT spacing or dither size need to be designed carefully in order to avoid this problem.

7.1 Future work

Due to time constrain, several aspects of the design has not been considered in this project and would be a motivation for future similar project. The first one is the dynamic performance of the ADC with background calibration. Another important factor is the power consumption and the digital complexity of the design. Increase the LUT size can improve the ADC's linearity; on the other hand it also increases the digital complexity and power consumptions. Additionally, it would be worthwhile to investigate different VCO architectures and there characteristics to optimize the figure of merit of the ADC.

A solution to eliminate the divergence issue mentioned in section 6.2.3 is to reduce the dither size ΔV. However, for the PCB implemented in this project, as the dither size was reduced, the two VCOs were coupled together, producing the same output. This might be due to internal coupling issue of the PCB that has not been investigated. Additionally, even for large dither, the way the dither was added to the two channels might introduce additional nonlinearity to the ADC. Therefore, a different analog front end also needs to be considered for future work.

Although an improvement in the LUT calibration technique was presented in this thesis project, it only can preserve the continuity for unused LUT regions at the two ends when the input signal is continuous. Skipping over some LUT locations would also results in discontinuity in the LUT. One option to estimate of the error for the skipped LUT locations would be to take the average error of the two nearest used LUT locations on the two sides. Finally, if the application require absolute offset calibration, a technique presented in [1] should be implemented in future projects.

Bibliography

[1] J. McNeill, R. Majidi, J. Gong, and C. Liu, "Lookup-Table-Based Background Linearization for VCO-Based ADCs," *IEEE Custom Integrated Circuits Conference*, Sept. 2014.

[2] J. McNeill, R. Majidi, and J. Gong, " 'Split ADC' Background Linearization of VCO-Based ADCs," *IEEE Trans. Circuits Syst. I*, vol. 62, no. 1, Jan. 2015, pp. 49-58.

[3] J. McNeill, M Coln, D. R. Brown, and Brian Larivee, "Digital Background Calibration Algorithm for "Split ADC" Architecture," *IEEE Trans. Circuits Syst. I*, vol. 56, no. 2, Feb. 2009, pp. 294-306.

[4] Robert H. Walden, "Analog-to-Digital Converter Survey and Analysis" *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 4, Apr. 1999 pp. 539-550.

[5] M. Z. Straayer and M. H. Perrott, "A 12-bit, 10-MHz bandwidth, continuous-time ADC with a 5-bit, 950-MS/s VCO-based quantizer," *IEEE Journal of Solid-State Circuits*, April 2008.

[6] Park, M.; Perrott, M.H., "A 78 dB SNDR 87 mW 20 MHz Bandwidth Continuous-Time ADC with VCO-Based Integrator and Quantizer Implemented in 0.13μm CMOS," *IEEE Journal of Solid-State Circuits*, Dec. 2009.

[7] G. Taylor and I. Galton, "A Mostly-Digital Variable-Rate Continuous-Time Delta-Sigma Modulator ADC," *IEEE Journal of Solid-State Circuits*, 2010.

[8] J. Hamilton, S. Yan, and T. R. Viswanathan, "An uncalibrated 2MHz, 6mW, 63.5dB SNDR discrete-time input VCO-based ADC," *IEEE Custom Integrated Circuits Conference (CICC2012)*, Sept. 2012.

[9] K. Lee, Y. Yoon, and N. Sun, "A 1.8mW 2MHz-BW 66.5dB-SNDR ADC Using VCO-Based Integrators with Intrinsic CLA," *IEEE Custom Integrated Circuits Conference (CICC2013)*, Sept. 2013.

[10] J. Kim, T. -K. Jang, Y. -G. Yoon and S. Cho, “Analysis and design of voltage-controlled oscillator based analog-to-digital converter,” *IEEE Transactions on Circuits and Systems I*, Jan. 2010.

[11] J. Li and U. Moon, “Background calibration techniques for multistage pipelined ADCs with digital redundancy, *IEEE Trans. Circuits Syst. II*, vol. 50, no. 9, pp. 531-538, Sep. 2003.

[12] Analog Devices technical staff, “Data Conversion Handbook,” Massachusetts: Newnes, 2015.

[13] R. Jacob Baker, “CMOS Circuit Design, Layout, and Simulation (Third Edition),” Jew Jersey: Wiley, 1964.

[14] Behzad Razavi, “Design of Analog CMOS Integrated Circuits,” New York: McGraw-Hill, 2001.

[15] Robert Pierret, “Semiconductor DEvice Fundamentals,”Massachusetts: Addison Wesley, 1996.

[16] Doug Amos, Austin Lesea and Rene Richter, “FPGA-Based Prototyping Methodology Manual Best Practices in Design-for-Prototyping” Synopsys, Inc, Mountain View, 2010.

[17] Joakim Bergs. Design of a VCO based ADC in a 1802m CMOS Process for use in Positron Emission Tomography. M.S. thesis, Lulea University of Technology, Sweden, 2009.

[18] R. W. Hamming, “Numerical Methods for Scientists and Engineers,” New York: Dover, 1986.

[19] W. H.Press, S. A.Teukolsky, W. T. Vetterling, and B. P. Flannery, “Numerical Recipes in C: The Art of Scientific Computing.” Cambridge Univ. Press, Cambridge, 1992.

[20] G. Golub and C. F. van Loan, “Matrix Computations (Third Edition),” Baltimore, Maryland: Johns Hopkins University Press, 1996.

Appendix A

Verilog Code

A.1 CLOCK GENERATOR block

```
//  Module Declaration
module CLOCK_GEN_2
(
// {{ALTERA_ARGS_BEGIN}} DO NOT REMOVE THIS LINE!
CLOCK_50, ADC_CLK, CTR_CLR, VCO_GATE, MEM_CLK, ENSEMBLE_CLK,
ENSEMBLE_CLK_RESET, ENSEMBLE_CLK_DELAY, ERROR_CORRECTION
// {{ALTERA_ARGS_END}} DO NOT REMOVE THIS LINE!
);
// Port Declaration

// {{ALTERA_IO_BEGIN}} DO NOT REMOVE THIS LINE!
input CLOCK_50;
output ADC_CLK;
output CTR_CLR;
output VCO_GATE;
output MEM_CLK;
output ENSEMBLE_CLK;
output ENSEMBLE_CLK_RESET;
output ENSEMBLE_CLK_DELAY;
output ERROR_CORRECTION;
// {{ALTERA_IO_END}} DO NOT REMOVE THIS LINE!

reg [9:0] cnt = 10'b0;
reg [19:0] cnt1;

always@(posedge CLOCK_50) begin
        cnt <= cnt+1'b1;
        cnt1 <= cnt1 +1'b1;
end

assign ADC_CLK = (cnt == 10'd2);
assign CTR_CLR = ~(cnt == 10'd9);
assign VCO_GATE = ~(cnt <= 10'd10);
assign MEM_CLK = (cnt == 10'd15);
assign ENSEMBLE_CLK = (cnt1 == 20'd20);
assign ENSEMBLE_CLK_RESET = (cnt1 == 20'd128);
assign ENSEMBLE_CLK_DELAY = (cnt1 == 20'd120);
assign ERROR_CORRECTION = (cnt1 > 20'd25)&&(cnt1 < 20'd100);
endmodule
```

A.2 MEMORY CONTROLER block

```
module SRAM_CONTROLER_2
(
// {{ALTERA_ARGS_BEGIN}} DO NOT REMOVE THIS LINE!
MEM_CLK, DATA, CLOCK_50, SRAM_WE_N, SRAM_ADDR, SRAM_DQ
// {{ALTERA_ARGS_END}} DO NOT REMOVE THIS LINE!
);
// Port Declaration

// {{ALTERA_IO_BEGIN}} DO NOT REMOVE THIS LINE!
input MEM_CLK;
input [15:0] DATA;
input CLOCK_50;
output SRAM_WE_N;
output [19:0] SRAM_ADDR;
output [15:0] SRAM_DQ;
// {{ALTERA_IO_END}} DO NOT REMOVE THIS LINE!


reg [19:0] address = 20'b0;
reg [20:0] write_enable = 21'b0;
always@(posedge CLOCK_50) begin
        if (MEM_CLK) begin
                address <= address + 1'b1;
                write_enable <= write_enable +1'b1;
        end
end
assign SRAM_WE_N = write_enable[20];
assign SRAM_ADDR = address;
assign SRAM_DQ = ((write_enable[20]==0))?   DATA : 16'bz;

endmodule
```

A.3 CALIBRATION block

```
module CALIBRATION_2
(
// {{ALTERA_ARGS_BEGIN}} DO NOT REMOVE THIS LINE!
Nout_A, Nout_B, ENSEMBLE_CLK, p, MEM_CLK, ENSEMBLE_CLK_RESET,
CLOCK_50, ENSEMBLE_CLK_DELAY,ERROR_CORRECTION,
Xout_A, Xout_B, Xout, deltax, CHECK
// {{ALTERA_ARGS_END}} DO NOT REMOVE THIS LINE!
);
// Port Declaration

// {{ALTERA_IO_BEGIN}} DO NOT REMOVE THIS LINE!
input [11:0] Nout_A; input [11:0] Nout_B; input ENSEMBLE_CLK;
input p; input MEM_CLK; input ENSEMBLE_CLK_RESET;
input ENSEMBLE_CLK_DELAY; input ERROR_CORRECTION;
input CLOCK_50;
output [15:0] Xout_A; output [15:0] Xout_B;
output [15:0] Xout; output [15:0] deltax;
output reg [15:0] CHECK;
// {{ALTERA_IO_END}} DO NOT REMOVE THIS LINE!


parameter D = 16'd1024; parameter LUTlength =7'd39

reg  [31:0] LUT_A [0:LUTlength]; reg  [31:0] LUT_B [0:LUTlength];
reg [LUTlength:0] hit_A; reg [LUTlength:0] hit_B;
reg [LUTlength:0] above_A; reg [LUTlength:0] above_B;
wire [5:0] LUTloc_A = Nout_A[11:6]-6'd10;
wire [5:0] LUTloc_B = Nout_B[11:6]-6'd10;

reg signed [31:0] rawea [0:LUTlength];
reg signed [31:0] raweb [0:LUTlength];
reg signed [31:0] ea [0:LUTlength];
reg signed [31:0] eb [0:LUTlength];
reg signed [31:0] eavalid [0:LUTlength];
reg signed [31:0] ebvalid [0:LUTlength];
```

```
initial begin // INITIALIZATION
 hit_A = 0;
 hit_B = 0;
 LUT_A[ 0] = 32'd 0;LUT_B[ 0] = 32'd 0;
 LUT_A[ 1] = 32'd 67108864 ;LUT_B[ 1] = 32'd 67108864 ;
 LUT_A[ 2] = 32'd 134217728 ;LUT_B[ 2] = 32'd 134217728 ;
 LUT_A[ 3] = 32'd 201326592 ;LUT_B[ 3] = 32'd 201326592 ;
 LUT_A[ 4] = 32'd 268435456 ;LUT_B[ 4] = 32'd 268435456 ;
 LUT_A[ 5] = 32'd 335544320 ;LUT_B[ 5] = 32'd 335544320 ;
 LUT_A[ 6] = 32'd 402653184 ;LUT_B[ 6] = 32'd 402653184 ;
 LUT_A[ 7] = 32'd 469762048 ;LUT_B[ 7] = 32'd 469762048 ;
 LUT_A[ 8] = 32'd 536870912 ;LUT_B[ 8] = 32'd 536870912 ;
 LUT_A[ 9] = 32'd 603979776 ;LUT_B[ 9] = 32'd 603979776 ;
 LUT_A[ 10 ] = 32'd 671088640 ;LUT_B[ 10 ] = 32'd 671088640 ;
 LUT_A[ 11 ] = 32'd 738197504 ;LUT_B[ 11 ] = 32'd 738197504 ;
 LUT_A[ 12 ] = 32'd 805306368 ;LUT_B[ 12 ] = 32'd 805306368 ;
 LUT_A[ 13 ] = 32'd 872415232 ;LUT_B[ 13 ] = 32'd 872415232 ;
 LUT_A[ 14 ] = 32'd 939524096 ;LUT_B[ 14 ] = 32'd 939524096 ;
 LUT_A[ 15 ] = 32'd 1006632960 ;LUT_B[ 15 ] = 32'd 1006632960 ;
 LUT_A[ 16 ] = 32'd 1073741824 ;LUT_B[ 16 ] = 32'd 1073741824 ;
 LUT_A[ 17 ] = 32'd 1140850688 ;LUT_B[ 17 ] = 32'd 1140850688 ;
 LUT_A[ 18 ] = 32'd 1207959552 ;LUT_B[ 18 ] = 32'd 1207959552 ;
 LUT_A[ 19 ] = 32'd 1275068416 ;LUT_B[ 19 ] = 32'd 1275068416 ;
 LUT_A[ 20 ] = 32'd 1342177280 ;LUT_B[ 20 ] = 32'd 1342177280 ;
 LUT_A[ 21 ] = 32'd 1409286144 ;LUT_B[ 21 ] = 32'd 1409286144 ;
 LUT_A[ 22 ] = 32'd 1476395008 ;LUT_B[ 22 ] = 32'd 1476395008 ;
 LUT_A[ 23 ] = 32'd 1543503872 ;LUT_B[ 23 ] = 32'd 1543503872 ;
 LUT_A[ 24 ] = 32'd 1610612736 ;LUT_B[ 24 ] = 32'd 1610612736 ;
 LUT_A[ 25 ] = 32'd 1677721600 ;LUT_B[ 25 ] = 32'd 1677721600 ;
 LUT_A[ 26 ] = 32'd 1744830464 ;LUT_B[ 26 ] = 32'd 1744830464 ;
 LUT_A[ 27 ] = 32'd 1811939328 ;LUT_B[ 27 ] = 32'd 1811939328 ;
 LUT_A[ 28 ] = 32'd 1879048192 ;LUT_B[ 28 ] = 32'd 1879048192 ;
 LUT_A[ 29 ] = 32'd 1946157056 ;LUT_B[ 29 ] = 32'd 1946157056 ;
 LUT_A[ 30 ] = 32'd 2013265920 ;LUT_B[ 30 ] = 32'd 2013265920 ;
 LUT_A[ 31 ] = 32'd 2080374784 ;LUT_B[ 31 ] = 32'd 2080374784 ;
 LUT_A[ 32 ] = 32'd 2147483648 ;LUT_B[ 32 ] = 32'd 2147483648 ;
 LUT_A[ 33 ] = 32'd 2214592512 ;LUT_B[ 33 ] = 32'd 2214592512 ;
 LUT_A[ 34 ] = 32'd 2281701376 ;LUT_B[ 34 ] = 32'd 2281701376 ;
 LUT_A[ 35 ] = 32'd 2348810240 ;LUT_B[ 35 ] = 32'd 2348810240 ;
 LUT_A[ 36 ] = 32'd 2415919104 ;LUT_B[ 36 ] = 32'd 2415919104 ;
 LUT_A[ 37 ] = 32'd 2483027968 ;LUT_B[ 37 ] = 32'd 2483027968 ;
 LUT_A[ 38 ] = 32'd 2550136832 ;LUT_B[ 38 ] = 32'd 2550136832 ;
 LUT_A[ 39 ] = 32'd 2617245696 ;LUT_B[ 39 ] = 32'd 2617245696 ;
```

```
rawea[0] = 32'd0;rawea[1] = 32'd0;raweb[0] = 32'd0;raweb[1] = 32'd0;
rawea[2] = 32'd0;rawea[3] = 32'd0;raweb[2] = 32'd0;raweb[3] = 32'd0;
rawea[4] = 32'd0;rawea[5] = 32'd0;raweb[4] = 32'd0;raweb[5] = 32'd0;
rawea[6] = 32'd0;rawea[7] = 32'd0;raweb[6] = 32'd0;raweb[7] = 32'd0;
rawea[8] = 32'd0;rawea[9] = 32'd0;raweb[8] = 32'd0;raweb[9] = 32'd0;
rawea[10] = 32'd0;rawea[11] = 32'd0;raweb[10] = 32'd0;raweb[11] = 32'd0;
rawea[12] = 32'd0;rawea[13] = 32'd0;raweb[12] = 32'd0;raweb[13] = 32'd0;
rawea[14] = 32'd0;rawea[15] = 32'd0;raweb[14] = 32'd0;raweb[15] = 32'd0;
rawea[16] = 32'd0;rawea[17] = 32'd0;raweb[16] = 32'd0;raweb[17] = 32'd0;
rawea[18] = 32'd0;rawea[19] = 32'd0;raweb[18] = 32'd0;raweb[19] = 32'd0;
rawea[20] = 32'd0;rawea[21] = 32'd0;raweb[20] = 32'd0;raweb[21] = 32'd0;
rawea[22] = 32'd0;rawea[23] = 32'd0;raweb[22] = 32'd0;raweb[23] = 32'd0;
rawea[24] = 32'd0;rawea[25] = 32'd0;raweb[24] = 32'd0;raweb[25] = 32'd0;
rawea[26] = 32'd0;rawea[27] = 32'd0;raweb[26] = 32'd0;raweb[27] = 32'd0;
rawea[28] = 32'd0;rawea[29] = 32'd0;raweb[28] = 32'd0;raweb[29] = 32'd0;
rawea[30] = 32'd0;rawea[31] = 32'd0;raweb[30] = 32'd0;raweb[31] = 32'd0;
rawea[32] = 32'd0;rawea[33] = 32'd0;raweb[32] = 32'd0;raweb[33] = 32'd0;
rawea[34] = 32'd0;rawea[35] = 32'd0;raweb[34] = 32'd0;raweb[35] = 32'd0;
rawea[36] = 32'd0;rawea[37] = 32'd0;raweb[36] = 32'd0;raweb[37] = 32'd0;
rawea[38] = 32'd0;rawea[39] = 32'd0;raweb[38] = 32'd0;raweb[39] = 32'd0;


ea[0] = 32'd0;ea[1] = 32'd0;eb[0] = 32'd0;eb[1] = 32'd0;
ea[2] = 32'd0;ea[3] = 32'd0;eb[2] = 32'd0;eb[3] = 32'd0;
ea[4] = 32'd0;ea[5] = 32'd0;eb[4] = 32'd0;eb[5] = 32'd0;
ea[6] = 32'd0;ea[7] = 32'd0;eb[6] = 32'd0;eb[7] = 32'd0;
ea[8] = 32'd0;ea[9] = 32'd0;eb[8] = 32'd0;eb[9] = 32'd0;
ea[10] = 32'd0;ea[11] = 32'd0;eb[10] = 32'd0;eb[11] = 32'd0;
ea[12] = 32'd0;ea[13] = 32'd0;eb[12] = 32'd0;eb[13] = 32'd0;
ea[14] = 32'd0;ea[15] = 32'd0;eb[14] = 32'd0;eb[15] = 32'd0;
ea[16] = 32'd0;ea[17] = 32'd0;eb[16] = 32'd0;eb[17] = 32'd0;
ea[18] = 32'd0;ea[19] = 32'd0;eb[18] = 32'd0;eb[19] = 32'd0;
ea[20] = 32'd0;ea[21] = 32'd0;eb[20] = 32'd0;eb[21] = 32'd0;
ea[22] = 32'd0;ea[23] = 32'd0;eb[22] = 32'd0;eb[23] = 32'd0;
ea[24] = 32'd0;ea[25] = 32'd0;eb[24] = 32'd0;eb[25] = 32'd0;
ea[26] = 32'd0;ea[27] = 32'd0;eb[26] = 32'd0;eb[27] = 32'd0;
ea[28] = 32'd0;ea[29] = 32'd0;eb[28] = 32'd0;eb[29] = 32'd0;
ea[30] = 32'd0;ea[31] = 32'd0;eb[30] = 32'd0;eb[31] = 32'd0;
ea[32] = 32'd0;ea[33] = 32'd0;eb[32] = 32'd0;eb[33] = 32'd0;
ea[34] = 32'd0;ea[35] = 32'd0;eb[34] = 32'd0;eb[35] = 32'd0;
ea[36] = 32'd0;ea[37] = 32'd0;eb[36] = 32'd0;eb[37] = 32'd0;
ea[38] = 32'd0;ea[39] = 32'd0;eb[38] = 32'd0;eb[39] = 32'd0;

end//END LUT INITIALIZATION


wire[31:0] SLOPE_A = LUT_A[LUTloc_A+1'b1] - LUT_A[LUTloc_A];
wire [37:0] DIFF_A = SLOPE_A*Nout_A[5:0];
wire [31:0] DIFF_A1 = DIFF_A[37:6]+LUT_A[LUTloc_A];
assign Xout_A = DIFF_A1[31:16];//divide DIFF by 64 = 6 LSBs

wire[31:0] SLOPE_B = LUT_B[LUTloc_B+1'b1] - LUT_B[LUTloc_B];
wire [37:0] DIFF_B = SLOPE_B*Nout_B[5:0];
wire [31:0] DIFF_B1 = DIFF_B[37:6]+LUT_B[LUTloc_B];
assign Xout_B = DIFF_B1[31:16];//divide DIFF by 64 = 6 LSBs
```

```
//AVERAGE THE TWO CHANNEL AND CALCULATE PARAMETERS NEEDED FOR ERROR
ESTIMATION

wire [16:0] SUM = Xout_A + Xout_B;
assign Xout = SUM [16:1];

assign deltax = (p)? (Xout_B - (Xout_A+D)) : ((Xout_B+D)- Xout_A );
wire signed [6:0] temp_a1 = 7'sd63 - Xout_A[5:0];//convert to signed number
wire signed [6:0] temp_a2 = Xout_A[5:0]; //convert to signed number
wire signed [6:0] temp_b1 = -7'sd63 + Xout_B[5:0];//convert to signed number
wire signed [6:0] temp_b2 = -Xout_B[5:0]; //convert to signed number
wire signed [31:0] update1 = $signed(deltax)*$signed(temp_a1);
wire signed [31:0] update2 = $signed(deltax)*$signed(temp_a2);
wire signed [31:0] update3 = $signed(deltax)*$signed(temp_b1);
wire signed [31:0] update4 = $signed(deltax)*$signed(temp_b2);

//// UPDATE THE ERROR MATRIXES EVERY CONVERSION
always@(posedge CLOCK_50) begin
if(ENSEMBLE_CLK_RESET) begin
hit_A = 0;
hit_B = 0;
rawea[0] = 32'd0;rawea[1] = 32'd0;raweb[0] = 32'd0;raweb[1] = 32'd0;
rawea[2] = 32'd0;rawea[3] = 32'd0;raweb[2] = 32'd0;raweb[3] = 32'd0;
rawea[4] = 32'd0;rawea[5] = 32'd0;raweb[4] = 32'd0;raweb[5] = 32'd0;
rawea[6] = 32'd0;rawea[7] = 32'd0;raweb[6] = 32'd0;raweb[7] = 32'd0;
rawea[8] = 32'd0;rawea[9] = 32'd0;raweb[8] = 32'd0;raweb[9] = 32'd0;
rawea[10] = 32'd0;rawea[11] = 32'd0;raweb[10] = 32'd0;raweb[11] = 32'd0;
rawea[12] = 32'd0;rawea[13] = 32'd0;raweb[12] = 32'd0;raweb[13] = 32'd0;
rawea[14] = 32'd0;rawea[15] = 32'd0;raweb[14] = 32'd0;raweb[15] = 32'd0;
rawea[16] = 32'd0;rawea[17] = 32'd0;raweb[16] = 32'd0;raweb[17] = 32'd0;
rawea[18] = 32'd0;rawea[19] = 32'd0;raweb[18] = 32'd0;raweb[19] = 32'd0;
rawea[20] = 32'd0;rawea[21] = 32'd0;raweb[20] = 32'd0;raweb[21] = 32'd0;
rawea[22] = 32'd0;rawea[23] = 32'd0;raweb[22] = 32'd0;raweb[23] = 32'd0;
rawea[24] = 32'd0;rawea[25] = 32'd0;raweb[24] = 32'd0;raweb[25] = 32'd0;
rawea[26] = 32'd0;rawea[27] = 32'd0;raweb[26] = 32'd0;raweb[27] = 32'd0;
rawea[28] = 32'd0;rawea[29] = 32'd0;raweb[28] = 32'd0;raweb[29] = 32'd0;
rawea[30] = 32'd0;rawea[31] = 32'd0;raweb[30] = 32'd0;raweb[31] = 32'd0;
rawea[32] = 32'd0;rawea[33] = 32'd0;raweb[32] = 32'd0;raweb[33] = 32'd0;
rawea[34] = 32'd0;rawea[35] = 32'd0;raweb[34] = 32'd0;raweb[35] = 32'd0;
rawea[36] = 32'd0;rawea[37] = 32'd0;raweb[36] = 32'd0;raweb[37] = 32'd0;
rawea[38] = 32'd0;rawea[39] = 32'd0;raweb[38] = 32'd0;raweb[39] = 32'd0;


end else if (MEM_CLK) begin
CHECK = deltax;
hit_A[LUTloc_A] = 1'b1;
hit_B[LUTloc_B] = 1'b1;
hit_A[LUTloc_A+1'b1] = 1'b1;
hit_B[LUTloc_B+1'b1] = 1'b1;

rawea[LUTloc_A] = rawea[LUTloc_A] - update1; //This is 32 * rawea in the
matlab file
rawea[LUTloc_A+1'b1] = rawea[LUTloc_A+1'b1] - update2;//Same here
raweb[LUTloc_B] = raweb[LUTloc_B] - update3; //This is 32 * rawea in the
matlab file
raweb[LUTloc B+1'b1] = raweb[LUTloc B+1'b1] - update4;//Same here
```

```
LUT_A[15] = LUT_A[15]- $signed(eavalid[15][31:4])-eavalid[15][31];
LUT_B[15] = LUT_B[15]- $signed(ebvalid[15][31:4])-ebvalid[15][31];
LUT_A[16] = LUT_A[16]- $signed(eavalid[16][31:4])-eavalid[16][31];
LUT_B[16] = LUT_B[16]- $signed(ebvalid[16][31:4])-ebvalid[16][31];
LUT_A[17] = LUT_A[17]- $signed(eavalid[17][31:4])-eavalid[17][31];
LUT_B[17] = LUT_B[17]- $signed(ebvalid[17][31:4])-ebvalid[17][31];
LUT_A[18] = LUT_A[18]- $signed(eavalid[18][31:4])-eavalid[18][31];
LUT_B[18] = LUT_B[18]- $signed(ebvalid[18][31:4])-ebvalid[18][31];
LUT_A[19] = LUT_A[19]- $signed(eavalid[19][31:4])-eavalid[19][31];
LUT_B[19] = LUT_B[19]- $signed(ebvalid[19][31:4])-ebvalid[19][31];
LUT_A[20] = LUT_A[20]- $signed(eavalid[20][31:4])-eavalid[20][31];
LUT_B[20] = LUT_B[20]- $signed(ebvalid[20][31:4])-ebvalid[20][31];
LUT_A[21] = LUT_A[21]- $signed(eavalid[21][31:4])-eavalid[21][31];
LUT_B[21] = LUT_B[21]- $signed(ebvalid[21][31:4])-ebvalid[21][31];
LUT_A[22] = LUT_A[22]- $signed(eavalid[22][31:4])-eavalid[22][31];
LUT_B[22] = LUT_B[22]- $signed(ebvalid[22][31:4])-ebvalid[22][31];
LUT_A[23] = LUT_A[23]- $signed(eavalid[23][31:4])-eavalid[23][31];
LUT_B[23] = LUT_B[23]- $signed(ebvalid[23][31:4])-ebvalid[23][31];
LUT_A[24] = LUT_A[24]- $signed(eavalid[24][31:4])-eavalid[24][31];
LUT_B[24] = LUT_B[24]- $signed(ebvalid[24][31:4])-ebvalid[24][31];
LUT_A[25] = LUT_A[25]- $signed(eavalid[25][31:4])-eavalid[25][31];
LUT_B[25] = LUT_B[25]- $signed(ebvalid[25][31:4])-ebvalid[25][31];
LUT_A[26] = LUT_A[26]- $signed(eavalid[26][31:4])-eavalid[26][31];
LUT_B[26] = LUT_B[26]- $signed(ebvalid[26][31:4])-ebvalid[26][31];
LUT_A[27] = LUT_A[27]- $signed(eavalid[27][31:4])-eavalid[27][31];
LUT_B[27] = LUT_B[27]- $signed(ebvalid[27][31:4])-ebvalid[27][31];
LUT_A[28] = LUT_A[28]- $signed(eavalid[28][31:4])-eavalid[28][31];
LUT_B[28] = LUT_B[28]- $signed(ebvalid[28][31:4])-ebvalid[28][31];
LUT_A[29] = LUT_A[29]- $signed(eavalid[29][31:4])-eavalid[29][31];
LUT_B[29] = LUT_B[29]- $signed(ebvalid[29][31:4])-ebvalid[29][31];
LUT_A[30] = LUT_A[30]- $signed(eavalid[30][31:4])-eavalid[30][31];
LUT_B[30] = LUT_B[30]- $signed(ebvalid[30][31:4])-ebvalid[30][31];
LUT_A[31] = LUT_A[31]- $signed(eavalid[31][31:4])-eavalid[31][31];
LUT_B[31] = LUT_B[31]- $signed(ebvalid[31][31:4])-ebvalid[31][31];
LUT_A[32] = LUT_A[32]- $signed(eavalid[32][31:4])-eavalid[32][31];
LUT_B[32] = LUT_B[32]- $signed(ebvalid[32][31:4])-ebvalid[32][31];
LUT_A[33] = LUT_A[33]- $signed(eavalid[33][31:4])-eavalid[33][31];
LUT_B[33] = LUT_B[33]- $signed(ebvalid[33][31:4])-ebvalid[33][31];
LUT_A[34] = LUT_A[34]- $signed(eavalid[34][31:4])-eavalid[34][31];
LUT_B[34] = LUT_B[34]- $signed(ebvalid[34][31:4])-ebvalid[34][31];
LUT_A[35] = LUT_A[35]- $signed(eavalid[35][31:4])-eavalid[35][31];
LUT_B[35] = LUT_B[35]- $signed(ebvalid[35][31:4])-ebvalid[35][31];
LUT_A[36] = LUT_A[36]- $signed(eavalid[36][31:4])-eavalid[36][31];
LUT_B[36] = LUT_B[36]- $signed(ebvalid[36][31:4])-ebvalid[36][31];
LUT_A[37] = LUT_A[37]- $signed(eavalid[37][31:4])-eavalid[37][31];
LUT_B[37] = LUT_B[37]- $signed(ebvalid[37][31:4])-ebvalid[37][31];
LUT_A[38] = LUT_A[38]- $signed(eavalid[38][31:4])-eavalid[38][31];
LUT_B[38] = LUT_B[38]- $signed(ebvalid[38][31:4])-ebvalid[38][31];

end
end
```

```
// UPDATE THE UNUSED PORTION OF THE LUT USING EAVALID AND EBVALID MATRIX


always@(posedge CLOCK_50) begin
if (ERROR_CORRECTION) begin
eavalid[ 0  ]<=(hit_A[   0  ])?   ea[   0  ]:  eavalid[1];
eavalid[1]<=(hit_A[1])? ea[1]: (above_A[1])? eavalid[0]: eavalid[2];
eavalid[2]<=(hit_A[2])? ea[2]: (above_A[2])? eavalid[1]: eavalid[3];
eavalid[3]<=(hit_A[3])? ea[3]: (above_A[3])? eavalid[2]: eavalid[4];
eavalid[4]<=(hit_A[4])? ea[4]: (above_A[4])? eavalid[3]: eavalid[5];
eavalid[5]<=(hit_A[5])? ea[5]: (above_A[5])? eavalid[4]: eavalid[6];
eavalid[6]<=(hit_A[6])? ea[6]: (above_A[6])? eavalid[5]: eavalid[7];
eavalid[7]<=(hit_A[7])? ea[7]: (above_A[7])? eavalid[6]: eavalid[8];
eavalid[8]<=(hit_A[8])? ea[8]: (above_A[8])? eavalid[7]: eavalid[9];
eavalid[9]<=(hit_A[9])? ea[9]: (above_A[9])? eavalid[8]: eavalid[10];
eavalid[10]<=(hit_A[10])? ea[10]: (above_A[10])? eavalid[9]: eavalid[11];
eavalid[11]<=(hit_A[11])? ea[11]: (above_A[11])? eavalid[10]: eavalid[12];
eavalid[12]<=(hit_A[12])? ea[12]: (above_A[12])? eavalid[11]: eavalid[13];
eavalid[13]<=(hit_A[13])? ea[13]: (above_A[13])? eavalid[12]: eavalid[14];
eavalid[14]<=(hit_A[14])? ea[14]: (above_A[14])? eavalid[13]: eavalid[15];
eavalid[15]<=(hit_A[15])? ea[15]: (above_A[15])? eavalid[14]: eavalid[16];
eavalid[16]<=(hit_A[16])? ea[16]: (above_A[16])? eavalid[15]: eavalid[17];
eavalid[17]<=(hit_A[17])? ea[17]: (above_A[17])? eavalid[16]: eavalid[18];
eavalid[18]<=(hit_A[18])? ea[18]: (above_A[18])? eavalid[17]: eavalid[19];
eavalid[19]<=(hit_A[19])? ea[19]: (above_A[19])? eavalid[18]: eavalid[20];
eavalid[20]<=(hit_A[20])? ea[20]: (above_A[20])? eavalid[19]: eavalid[21];
eavalid[21]<=(hit_A[21])? ea[21]: (above_A[21])? eavalid[20]: eavalid[22];
eavalid[22]<=(hit_A[22])? ea[22]: (above_A[22])? eavalid[21]: eavalid[23];
eavalid[23]<=(hit_A[23])? ea[23]: (above_A[23])? eavalid[22]: eavalid[24];
eavalid[24]<=(hit_A[24])? ea[24]: (above_A[24])? eavalid[23]: eavalid[25];
eavalid[25]<=(hit_A[25])? ea[25]: (above_A[25])? eavalid[24]: eavalid[26];
eavalid[26]<=(hit_A[26])? ea[26]: (above_A[26])? eavalid[25]: eavalid[27];
eavalid[27]<=(hit_A[27])? ea[27]: (above_A[27])? eavalid[26]: eavalid[28];
eavalid[28]<=(hit_A[28])? ea[28]: (above_A[28])? eavalid[27]: eavalid[29];
eavalid[29]<=(hit_A[29])? ea[29]: (above_A[29])? eavalid[28]: eavalid[30];
eavalid[30]<=(hit_A[30])? ea[30]: (above_A[30])? eavalid[29]: eavalid[31];
eavalid[31]<=(hit_A[31])? ea[31]: (above_A[31])? eavalid[30]: eavalid[32];
eavalid[32]<=(hit_A[32])? ea[32]: (above_A[32])? eavalid[31]: eavalid[33];
eavalid[33]<=(hit_A[33])? ea[33]: (above_A[33])? eavalid[32]: eavalid[34];
eavalid[34]<=(hit_A[34])? ea[34]: (above_A[34])? eavalid[33]: eavalid[35];
eavalid[35]<=(hit_A[35])? ea[35]: (above_A[35])? eavalid[34]: eavalid[36];
eavalid[36]<=(hit_A[36])? ea[36]: (above_A[36])? eavalid[35]: eavalid[37];
eavalid[37]<=(hit_A[37])? ea[37]: (above_A[37])? eavalid[36]: eavalid[38];
eavalid[38]<=(hit_A[38])? ea[38]: (above_A[38])? eavalid[37]: eavalid[39];
eavalid[ 39  ]<=(hit_A[   39  ])?   ea[   39  ]:  eavalid[38];
end
end
```

```
always@(posedge CLOCK_50) begin
if (ERROR_CORRECTION) begin
ebvalid[ 0  ]<=(hit_B[   0  ])? eb[   0  ]: ebvalid[1];
ebvalid[1]<=(hit_B[1])? eb[1]: (above_B[1])? ebvalid[0]: ebvalid[2];
ebvalid[2]<=(hit_B[2])? eb[2]: (above_B[2])? ebvalid[1]: ebvalid[3];
ebvalid[3]<=(hit_B[3])? eb[3]: (above_B[3])? ebvalid[2]: ebvalid[4];
ebvalid[4]<=(hit_B[4])? eb[4]: (above_B[4])? ebvalid[3]: ebvalid[5];
ebvalid[5]<=(hit_B[5])? eb[5]: (above_B[5])? ebvalid[4]: ebvalid[6];
ebvalid[6]<=(hit_B[6])? eb[6]: (above_B[6])? ebvalid[5]: ebvalid[7];
ebvalid[7]<=(hit_B[7])? eb[7]: (above_B[7])? ebvalid[6]: ebvalid[8];
ebvalid[8]<=(hit_B[8])? eb[8]: (above_B[8])? ebvalid[7]: ebvalid[9];
ebvalid[9]<=(hit_B[9])? eb[9]: (above_B[9])? ebvalid[8]: ebvalid[10];
ebvalid[10]<=(hit_B[10])? eb[10]: (above_B[10])? ebvalid[9]: ebvalid[11];
ebvalid[11]<=(hit_B[11])? eb[11]: (above_B[11])? ebvalid[10]: ebvalid[12];
ebvalid[12]<=(hit_B[12])? eb[12]: (above_B[12])? ebvalid[11]: ebvalid[13];
ebvalid[13]<=(hit_B[13])? eb[13]: (above_B[13])? ebvalid[12]: ebvalid[14];
ebvalid[14]<=(hit_B[14])? eb[14]: (above_B[14])? ebvalid[13]: ebvalid[15];
ebvalid[15]<=(hit_B[15])? eb[15]: (above_B[15])? ebvalid[14]: ebvalid[16];
ebvalid[16]<=(hit_B[16])? eb[16]: (above_B[16])? ebvalid[15]: ebvalid[17];
ebvalid[17]<=(hit_B[17])? eb[17]: (above_B[17])? ebvalid[16]: ebvalid[18];
ebvalid[18]<=(hit_B[18])? eb[18]: (above_B[18])? ebvalid[17]: ebvalid[19];
ebvalid[19]<=(hit_B[19])? eb[19]: (above_B[19])? ebvalid[18]: ebvalid[20];
ebvalid[20]<=(hit_B[20])? eb[20]: (above_B[20])? ebvalid[19]: ebvalid[21];
ebvalid[21]<=(hit_B[21])? eb[21]: (above_B[21])? ebvalid[20]: ebvalid[22];
ebvalid[22]<=(hit_B[22])? eb[22]: (above_B[22])? ebvalid[21]: ebvalid[23];
ebvalid[23]<=(hit_B[23])? eb[23]: (above_B[23])? ebvalid[22]: ebvalid[24];
ebvalid[24]<=(hit_B[24])? eb[24]: (above_B[24])? ebvalid[23]: ebvalid[25];
ebvalid[25]<=(hit_B[25])? eb[25]: (above_B[25])? ebvalid[24]: ebvalid[26];
ebvalid[26]<=(hit_B[26])? eb[26]: (above_B[26])? ebvalid[25]: ebvalid[27];
ebvalid[27]<=(hit_B[27])? eb[27]: (above_B[27])? ebvalid[26]: ebvalid[28];
ebvalid[28]<=(hit_B[28])? eb[28]: (above_B[28])? ebvalid[27]: ebvalid[29];
ebvalid[29]<=(hit_B[29])? eb[29]: (above_B[29])? ebvalid[28]: ebvalid[30];
ebvalid[30]<=(hit_B[30])? eb[30]: (above_B[30])? ebvalid[29]: ebvalid[31];
ebvalid[31]<=(hit_B[31])? eb[31]: (above_B[31])? ebvalid[30]: ebvalid[32];
ebvalid[32]<=(hit_B[32])? eb[32]: (above_B[32])? ebvalid[31]: ebvalid[33];
ebvalid[33]<=(hit_B[33])? eb[33]: (above_B[33])? ebvalid[32]: ebvalid[34];
ebvalid[34]<=(hit_B[34])? eb[34]: (above_B[34])? ebvalid[33]: ebvalid[35];
ebvalid[35]<=(hit_B[35])? eb[35]: (above_B[35])? ebvalid[34]: ebvalid[36];
ebvalid[36]<=(hit_B[36])? eb[36]: (above_B[36])? ebvalid[35]: ebvalid[37];
ebvalid[37]<=(hit_B[37])? eb[37]: (above_B[37])? ebvalid[36]: ebvalid[38];
ebvalid[38]<=(hit_B[38])? eb[38]: (above_B[38])? ebvalid[37]: ebvalid[39];
ebvalid[ 39  ]<=(hit_B[   39  ])?   eb[   39  ]:  ebvalid[38];
end
end
endmodule
```

Appendix B

MATLAB Code

B.1 Offline LUT linearization

```
function [LUT, max_data_quantized, min_MSB_data] = LUTinit(LSBnum)
id = fopen('raw_data.bin');
raw_data1 = floor(single(fread(id,'*uint16'))/16);

%==========================================================
% Process the raw data and get the useful data
% Eliminate the first number
% Because in FPGA the first data in memory is always 0
data3 = sort(raw_data1(2:end),'ascend');
data4 = data3(1:end);
%==========================================================
% Find the slope of offline LUT
ramp = linspace(min(data4),max(data4),length(data4));

MSB_data = floor(data4/(2^(LSBnum)));
LSB_data = data4 - MSB_data*(2^LSBnum);
data_quantized = data4 - LSB_data;
data_quantized_unique = unique(data_quantized);
% Initial LUT filled with 0
LUT = zeros(size(data_quantized_unique));
% Find ai, and put ai into LUT
for i = 1:1:length(data_quantized_unique)
    index = find(data_quantized ==

    LUT(i) = floor(ramp(index));
end
LUT(1) = (ramp(1)-LUT(2)*LSB_data(1)/(2^LSBnum))
```

```
                                        /(1-LSB_data(1)/(2^LSBnum));
temp = (ramp(end) - LUT(end))/LSB_data(end)
                                       *(2^LSBnum) + LUT(end);

LUT = [LUT;temp];
max_data_quantized = max(data_quantized);
min_MSB_data = min(MSB_data);
```

B.2 Linearity test

```
function [dnl, inl] = dnlinl(data)
% Set edges for DNL
edge = (min(data)-0.5:1:max(data)+0.5);
% Count numbers of data in different areas
dnl_count=histc(data, edge);
% For DNL, chop out first 2 and last 2 edge_points
dnl=dnl_count(2:end-2);
% normalize
dnl=dnl/mean(dnl)-1;
% inl as cumsum
inl=cumsum(dnl);
```

B.3 Offline calibration linearity test

```
clear all;
close all;
clc;
LSBnum = 6;          %5 LSBs for linear interpolation
id = fopen('raw_data.bin');
raw_data = floor((single(fread(id,'*uint16')))/16);

figure(1)
plot(raw_data)
grid on
%==========================================================
%pre_process the raw data
%eliminate the first number (always 0)
data2 = sort(raw_data(2:end),'ascend');
data1 = data2(1:end);
[LUT, max_data_quantized, min_MSB_data] = findai(LSBnum);
data = data1;

% %eliminate data which exceed the LUT range
% k = 1;
% for i = 1:1:length(data1)
%     if (data1(i) < max_data_quantized)
%         data(k) = data1(i);
%         k = k+1;
%     end
% end
MSB_data = floor(data/(2^(LSBnum)));
LSB_data = data - MSB_data*(2^LSBnum);
%==========================================================
% Correct data based on LUT
% Initial with 0
data_correct = zeros(size(data));
% Interpolation
% data_correct = ai + (ai+1-ai)*(LSB/2^L)
for i=1:1:length(data)
    data_correct(i) = floor (LUT(MSB_data(i)-min_MSB_data+1)
                + (LUT(MSB_data(i)-min_MSB_data+2)-LUT(MSB_data(i)
                -min_MSB_data+1))*(LSB_data(i)/(2^LSBnum)));
end
%==========================================================
[DNL_raw, INL_raw] = dnlinl(data);
[DNL_correct, INL_correct] = dnlinl(data_correct);
%==========================================================
figure(2)
plot(data,'r','linewidth',2);
hold on;
plot(data_correct,'b','linewidth',2);
axis([0 length(data)  min(data) max(data)]);
xlabel('SAMPLES');
ylabel('DIGITAL OUTPUT');
legend('UNCORRECTED COUNT', 'CORRECTED COUNT')


figure(3)
title('Error or raw data')
subplot(2,1,1)
plot(DNL_raw,'LineWidth',2)
ylabel('DNL raw')
title('Error of uncalibrated data')
axis([0 length(DNL_raw) floor(min(DNL_raw)) ceil(max(DNL_raw))]);
subplot(2,1,2)
plot(INL_raw,'LineWidth',2)
```

```
ylabel('INL raw')
axis([0 length(INL_raw) floor(min(INL_raw)) ceil(max(INL_raw))]);

figure(4)
subplot(2,1,1)
plot(DNL_correct,'LineWidth',2)
ylabel('DNL  correct')
title('Error of data calibrated using MATLAB')
axis([0 length(DNL_correct) floor(min(DNL_correct)),
                            ceil(max(DNL_correct))]);
subplot(2,1,2)
plot(INL_correct,'LineWidth',2)
ylabel('INL correct')
axis([0 length(INL_correct) floor(min(INL_correct)),
                            ceil(max(INL_correct))]);

fclose('all');
```