1-1-2002

Testing and calibration of Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs)

Kumar Lakshmi Parthasarathy
Iowa State University

Testing and calibration of Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs)

by

Kumar Lakshmi Parthasarathy

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Randall Geiger, Major Professor
Degang Chen
Stuart Birrell
Turker Kuyel

Iowa State University

Ames, Iowa

2002

Graduate College

Iowa State University

This is to certify that the Master's thesis of

Kumar Lakshmi Parthasarathy

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Dr. Randall Geiger, for giving me the opportunity to work under his guidance at Iowa State University. His timely suggestions and his motivating and enlightening discussions has been a major driving force during my entire career at Iowa State University. His years of experience in the field of analog design have allowed him to focus in on the critical issues of any problem and helped me identify the challenging part of my research work.

I would like to thank Dr. Degang Chen for his guidance throughout the Master's degree. His insightful comments and technical discussions has been very valuable. I am indebted to Dr Turker Kuyel for giving me a chance to work with him during my internship at Texas Instruments, and for his helpful suggestions on my research work. I also thank Dr. Stuart Birrell for agreeing to be my committee member.

I am also greatly indebted to my family who has supported me throughout the years and who have been a constant source of encouragement and support when I needed them the most. Thanks are also due to my friends, the most wonderful people that I have come across. I really appreciate their support, for listening to my complains during the tough times and for cheering with me during the good times.

I wish to thank everyone at the VLSI Design Laboratory at Iowa State University. It has been a real privilege to work with such a wonderful group of talented and helpful people.

ABSTRACT

This research work focuses on two main aspects of Data Converter design. The first part of the research work focuses on a new calibration scheme for INL correction of a 16-bit Resistor string DAC. The suggested method is to digitally calibrate the DAC based on the performance results obtained from the tester. At the time of final testing, INL of the uncalibrated DAC will be evaluated using the standard tester and the linearity information (INL values) at certain specific input codes will be stored in registers. The values stored in the registers are then used to calibrate the DAC.

The second part of the research work concentrates on ADC testing. The emergence of high-performance, high-density devices have increased the demand on the number of testing steps and the required testing accuracy. Commercial testers used in the production environment not only results in huge initial investment on the testers, but also indirect investment in terms of tester time. This has resulted in a move towards Built-In-Self-Test (BIST) structures and testing strategies. A new BIST approach for measuring the Integral Non-Linearity (INL) and Differential Non-Linearity (DNL) of an Analog-to-Digital Converter (ADC) that uses histogram information is introduced in this work. Unlike most existing algorithms, this method does not require the generation of accurate input signals so offers potential for use in a BIST environment.

1. INTRODUCTION

With the rapid growth in the application of mixed-signal circuits in the fields of signal processing and telecommunications, the requirements on the performance of these circuits have also increased. One of the main mixed-signal modules that are widely used in communication and signal processing systems is a data converter, comprising either an Analog-to-Digital converter (ADC) or a Digital-to-Analog Converter (DAC) or a combination of both. Data converters bridge the gap between the analog domain and the digital world, thereby forming an important entity of many circuits. Current and future telecommunication standards have put very high requirements on data converters and this has been an area of active research for the past few decades.

Significant research has been done in the design of data converters to meet market specifications. Depending on the application, the specifications may vary from high resolution, high speed to low power devices. Varied architectures suitable for different needs have been successfully designed and implemented. To account for limitations posed by the available fabrication and processing technology, circuits with calibration capabilities are of growing interest. Most of the high end products (either A/D or D/A) are equipped with calibration modules, implemented either as background calibration or foreground calibration.

Apart from the above mentioned design aspects another major issue relating to data converters that has been an area of active research is related to testing. With the shrinking feature size and the availability of more sophisticated processing facilities and with the decreasing cost of silicon, more and more devices are being packed into a single chip, thereby increasing the demands on testing. Testing is perhaps becoming the fastest growing portion of manufacturing cost and the main bottle-neck in the reduction of IC prices. The high cost involved with testing comes mainly from the money that is directly invested in high cost mixed-signal testers and the indirect cost associated with

tester time. This has paved way for the emergence of Built-In-Self-Test (BIST) structures. These are structures that are in built within the chip along with the main circuitry and which help in partial or full characterization and calibration of the device, thereby either partially or completely eliminating the need for an external tester.

In this research work, two important aspects relating to data converters have been studied. A new calibration scheme has been proposed for digital calibration of a Digital-to-Analog converter and a new BIST scheme for testing an Analog-to-Digital Converter has been introduced.

In Chapter 2 of the thesis some of the standard definitions relating to specifications of a DAC have been described. Various existing definitions and the associated ambiguities are addressed. Efforts towards defining specifications unambiguously have been made and are explained in the chapter. This chapter is comprised of a paper that addresses this subject that was presented at the IEEE Midwest Symposium on Circuits and Systems, 2000. The newly proposed calibration scheme for digital calibration of a DAC is discussed in Chapter 3. Details regarding the calibration algorithm and the associated circuit design are included. A new BIST scheme for characterizing an Analog-to-Digital converter is explained in Chapter 4. Simulation and experimental results are also included.

2. UNAMBIGUOUS CHARACTERIZATION AND SPECIFICATION OF D/A CONVERTER PERFORMANCE

A paper published in the Proceedings of 2000 Midwest Symposium on Circuits and Systems

Kumar L Parthasarathy[1] and Randall Geiger[2]

2.1. Abstract

Existing parameters used to characterize the performance of D/A converters are often not rigorously defined or are based upon ambiguous nested definitions. This makes it difficult for the user to determine how a particular D/A will really perform in a specific application and leaves some room for uncertainty when testing D/As. Key performance parameters that affect the static and low frequency performance of Nyquist Rate Converters are unambiguously defined in this paper. Emphasis is on definitions that realistically predict how a D/A will perform and that make parameter measurements in the laboratory practical.

2.2. Introduction

The exponential growth witnessed recently in the area of digital communication equipment used in wireless communications has resulted in a great deal of interest in the design of data converters. This has led to a rapid growth of data converter markets, with the availability of devices having a wide range of performance parameters. Even though all D/A and A/D converters of a given resolution ideally exhibit identical transfer characteristics, they differ in practice by considerable amounts.

---

[1] Primary author
[2] Professor

Often the appropriate selection of hardware for a given application becomes a difficult task due to the shortcomings in the way these devices are characterized. Though many manufacturers/authors have tried to define even the most basic performance parameters, there still exists considerable ambiguity in the way the parameters are specified. The need exists to precisely define converter performance parameters to enable the user to choose an appropriate device based on their actual performance requirements.

Another factor driving the need for a standard performance characterization is the pending emergence of Built-In-Self-Test (BIST) structures for data converters. With the cost of silicon in many data converter products becoming dominated by testing costs, Built-In-Self-Test (BIST) structures offer potential for not only reducing the direct cost of testing, but also reducing the indirect cost associated with production time. Although some variants in performance characterization parameters may not adversely affect the characterization of a converter, the implications of these variants on testing algorithms and testing time on existing testers can be substantial but the implications on BIST implementations can be even more significant. For BIST for data converters to become viable, it is desirable that testing schemes and structures be as simple as possible. Since data converters traditionally have a long product life, it is also desirable to have the data converter design community agree on specifications that will not change often. This requires identification of the key parameters and precise definitions of these parameters.

Several different terms are widely used when discussing the performance of D/A converters. These include "Resolution", "Integral Nonlinearity (INL)", and "Differential Nonlinearity (DNL)" along with several others. These terms are so commonly used that one would be tempted to assume that there is universal agreement on what they mean and how they should be defined. However, not only are there differences in definition from one source to another, but also, most attempts to define these parameters involve parameters that are not precisely or unambiguously defined. Of course, if

these alternate definitions resulted in differences in perception about the overall performance of the D/A that were inconsequential, the imprecise nature or ambiguity would be only of academic interest. Unfortunately, the differences are of sufficient magnitude to cause possible misinterpretation of the real performance of a D/A in some applications.

In what follows, several key performance features for D/A converters are discussed, comparisons are made between alternate parameters that are used to characterize these features and unambiguous and consistent definitions of parameters are proposed that both capture the basic essence of key performance features and make measurement of the parameters manageable.

2.3. Specifications

A. Resolution

This is one of the most basic terms. In [1] it is defined as

"...the number of distinct analog levels corresponding to the different digital words. Thus, an N-bit resolution implies that the converter can resolve 2^N distinct analog levels";

In [2] it is defined as

"An N-bit binary converter should be able to provide 2^N distinct and different analog output values corresponding to the set of N-bit binary words. A converter that satisfies this criterion is said to have a resolution of N bits",

whereas in [4] it is defined by

" Resolution is the smallest level separation (input levels for A/D and output levels for D/A) that is unambiguously distinguishable over the full-scale range of the converter".

The first two definitions imply more or less the same central idea; a converter with N-bit resolution has 2^N outputs (in the case of a D/A). The first two definitions are totally determined by architecture and need not be measured. They provide little insight into performance. They are also dimensionless. The third is much different. It requires measurement and will have units of either volts or amps.

In the first two definitions, no mention is made about whether N is an integer although that assumption is often made as well. Though almost all D/A converters available in the market today deal with an integer number for resolution ('N' input bits and 2^N output levels), the concept of 'N' being an integer is traditional. Applications abound in which the number of distinct levels required at the output of a D/A is not an integer power of 2. A device which outputs a temperature value in 1°C degree increments from 0°C to 100°C or a percent value is an example where the number of required output levels equals 100. While using a 6-bit converter is not sufficient due to the maximum availability of 64 output levels, the use of 7-bit converter will result in an extra 28 levels which are not required and the quantization levels of the N-bit converter will likely not be aligned with those of the 100-level system.

A slightly modified definition based upon the philosophy behind the first two definitions which is suitable from a testing perspective is to define resolution based upon the number of transitions encountered while testing by the relation

$$R = \log_2(S+1) \tag{2.1}$$

where S is the number of distinct transitions. With this definition, R can be either an integer or a real number depending on the value of S.

B. Effective Resolution

The resolution as defined by (2.1) gives little indication about the resolving capability of a D/A. The concept of the effective resolution which is conveyed in the third definition [4] above does address the resolving capability. We will define the resolving capability as the smallest increment that can be guaranteed to be resolved with increasing input codes or the smallest decrement that can be guaranteed to be resolved with decreasing input codes.

This definition of resolving capability is motivated by the philosophy on which many control applications operate. These applications are based upon algorithms which increase the output of the DAC by increasing the input code or decrease the output of the DAC by decreasing the input code. The effective resolution will now be formally defined. To do this, consider the output levels for consecutive input codes as depicted in Figure.2.1. that are given by the sequence δ.

$$\delta = < X_0, X_1, X_2, \ldots\ldots\ldots X_k, X_{k+1} \ldots X_S > \tag{2.2}$$

where the element X_k corresponds to input code C_k.

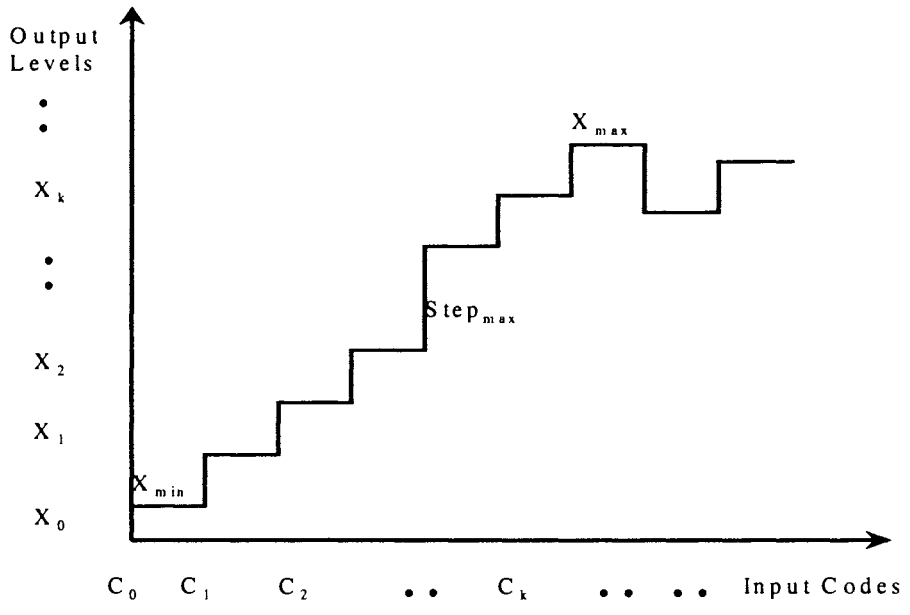


Figure 2.1 DAC Transfer Characteristics

In an ideal DAC, the sequence δ is not only monotonic but also the elements in the sequence increase linearly with the input code. In an actual DAC, the linear increase with input code is not guaranteed and the sequence may actually become non-monotonic. Consider now the two new sequences obtained from δ by rank ordering the elements of δ in increasing order and decreasing order. Denote these two sequences by

$$\delta_I = <Y_0, Y_1, Y_2, \ldots\ldots\ldots Y_S> \tag{2.3}$$

and

$$\delta_D = <Z_S, \ldots\ldots\ldots Z_2, Z_1, Z_0> \tag{2.4}$$

Note the arbitrary elements Y_k and Z_k no longer correspond to the input code C_k. Since the input codes corresponding to the elements of the sequence in δ_I may not be in increasing order, a new monotone sequence $\hat{\delta}_{Inc}$ is derived from δ_I. The new sequence, $\hat{\delta}_{Inc}$ has the property that both the output levels and the corresponding input codes show an increasing trend. The new sequence $\hat{\delta}_{Inc}$ will be obtained by the following iterative procedure:

(1) Let $h = 0$

(2) Define the sequence $\hat{\delta}_{Inc}(0) = \delta_I$

(3) Check the input code of $\hat{\delta}_{Inc}(h)$ for monotonicity (assume the elements of $\hat{\delta}_{Inc}(h)$ are denoted as Y_{hi} for $i = 1,2,\ldots\ldots\ldots,S_h$)

(a) If the input code of $\hat{\delta}_{Inc}(h)$ is monotonically increasing, define $\hat{\delta}_{Inc} = \hat{\delta}_{Inc}(h)$.

(b) If the input codes for $\hat{\delta}_{Inc}(h)$ are not in increasing order, then

A. Locate Y_{hi} which is the smallest element of the sequence $\hat{\delta}_{Inc}(h)$ where there exists a Y_{hj}, , $j > i$, where the input code of Y_{hj} is less than the input code of Y_{hi}.

B. Define the new sequence $\hat{\delta}_{Inc}(h+1)$ to be the sequence obtained by eliminating Y_{hi} from the sequence $\hat{\delta}_{Inc}(h)$.

C. Increase h by 1.

D. Return to the start of Step 3.

The new sequence obtained is then,

$$\hat{\delta}_{Inc} = < \hat{Y}_0, \hat{Y}_1, \hat{Y}_2, \ldots\ldots\ldots \hat{Y}_{L-1} > \tag{2.5}$$

Define the sequence of step sizes by

$$Step+ = <S_{0+}, S_{1+}, S_{2+}, \ldots\ldots S_{k+}, \ldots\ldots S_{(L-2)+}> \tag{2.6}$$

where $S_{k+} = \hat{Y}_{k+1} - \hat{Y}_k$, $k \in \{0,1,\ldots L-2\}$ and define $Step_{max+}$ by,

$$Step_{max+} = Max \{ S_{0+}, S_{1+}, S_{2+}, \ldots S_{k+}, \ldots\ldots S_{(L-2)+}\} \tag{2.7}$$

We now define the effective resolution (for increasing input code) by

$$R_{eff+} = \log_2((\hat{Y}_{L-1} - \hat{Y}_0)/ Step_{max+} + 1) \tag{2.8}$$

and the number of effective transitions to be

$$T_{eff+} = L - 1 \tag{2.9}$$

Similarly, a new sequence $\hat{\delta}_{Dec}$ is obtained from δ_D such that the output levels and the corresponding input codes show a decreasing trend. The new sequence obtained is then,

$$\hat{\delta}_{Dec} = < \hat{Z}_{M-1}, \ldots\ldots\ldots, \hat{Z}_2, \hat{Z}_1, \hat{Z}_0 > \tag{2.10}$$

Define the sequence of step sizes by

$$Step- = <S_{(M-2)-}, \ldots\ldots\ldots, S_{k-}, \ldots\ldots S_{1-}, S_{0-}> \tag{2.11}$$

where $S_{k-} = \hat{Z}_{(k+1)} - \hat{Z}_{(k)}$, $k \in \{M-2,\ldots,1,0\}$ and define $Step_{max-}$ by,

$$Step_{max-} = Max \{ S_{(M-2)-}, \ldots\ldots\ldots, S_{k-}, \ldots\ldots S_{1-}, S_{0-}\} \tag{2.12}$$

We now define the effective resolution (for decreasing input code) by

$$R_{eff-} = \log_2((\hat{Z}_{M-1} - \hat{Z}_0)/ Step_{max-} + 1) \quad (2.13)$$

and the number of effective transitions to be

$$T_{eff-} = M - 1 \quad (2.14)$$

The effective resolution of the converter is then defined to be

$$R_{eff} = Min\{R_{eff+}, R_{eff-}\} \quad (2.15)$$

and the number of effective transitions to be

$$T_{eff} = Min\{T_{eff+}, T_{eff-}\} \quad (2.16)$$

C. X_{LSB}

X_{LSB} is a term closely related to resolution and is defined in [1] as

"....the voltage change when 1LSB changes"

where X is a voltage signal, and is mathematically given by

$$\hat{X}_{LSB} = X_{ref}/2^N \quad (2.17)$$

Even though the definition is seemingly simple, whether this relationship between a reference signal and the constant N really represents an LSB is not apparent. X_{ref} is usually the input reference signal (can be any physical quantity like voltage, charge or current). Ideally the output voltage of a D/A converter for maximum input (all bits '1') is related to the reference signal by

$$X_{omax} = X_{ref}(1-2^{-N}) \quad (2.18)$$

But when a device is tested, seldom does the output of the converter actually equal this value of $X_{ref}(1-2^{-N})$ for maximum input code. Thus the definition of X_{LSB}, based on the ideal X_{ref} value is not quite correct. It is more reasonable to relate X_{LSB} to the maximum signal value reached at the output of the converter while testing. X_{LSB} can be defined as

$$X_{LSB} = (X_{max} - X_{min})/T_{eff} \tag{2.19}$$

where X_{max} is the maximum output, X_{min} is the minimum output and T_{eff} is the number of effective transitions as given in (2.16). This definition, although similar to the existing ones, deals more clearly with the practical case and is convenient to measure from a testing point of view.

D. Offset Error & Gain Error

Offset error is defined in [1] as

"In a D/A converter, the offset error, E_{off}, is defined to be the output that occurs for the input code that should produce zero output"

mathematically

$$\hat{X}_{Off} = X_{out}|_{00...00} / \hat{X}_{LSB} \tag{2.20}$$

A minor modification would be to define this in terms of X_{LSB} as per (2.19), and the new definition is

$$X_{off} = X_0 / X_{LSB} \tag{2.21}$$

Gain error in LSB is defined in [1] as

"...the difference at the full-scale value between the ideal and actual curves when the offset error has been reduced to zero."

For a D/A converter, the gain error, in units of LSBs, is

$$\hat{E}_{gain} = (X_{out}|_{11...1} - X_{out}|_{00...0}) / \hat{X}_{LSB} - (2^N - 1) \tag{2.22}$$

or equivalently,

$$\hat{E}_{gain} = (X_{(2^N-1)} - X_0) / \hat{X}_{LSB} - (2^N - 1) \tag{2.23}$$

Since the number of transitions in a D/A may not necessarily be one less than an integral power of 2, we will define the gain error by

$$E_{gain} = (X_S - X_0)/ X_{LSB} - T_{eff} \tag{2.24}$$

E. Differential Nonlinearity (DNL)

INL and DNL are two closely related parameters which are usually specified and are considered as critical performance parameters. In [1] the author defines DNL as

"DNL is defined as the variation in analog step sizes away from 1LSB (typically, once gain and offset errors have been removed)"

while in [2] the author states it as

"In a D/A converter, any two adjacent digital codes should result in measured output values that are exactly 1LSB apart (2^{-N} of full scale for an N-bit converter). Any positive or negative deviation of the measured "step" from the ideal difference is called Differential Nonlinearity, expressed in (sub)multiples of 1LSB."

Often, DNL is specified as the worst case deviation obtained from 1LSB. In terms of the modified definition of X_{LSB}, DNL of i_{th} element is defined as

$$DNL_i = (Stepactual_i/ X_{LSB}) - 1 \tag{2.25}$$

where

$$Stepactual_i = X_{i+1} - X_i \tag{2.26}$$

DNL is then

$$DNL = Max\{DNL_i\} \tag{2.27}$$

F. Integral Nonlinearity (INL)

INL is defined in [1] as

"INL error is defined to be the deviation from a straight line"

where two different methods of obtaining straight line based on either end points of the converter's transfer characteristics or a best-fit line such that the mean squared error is reduced, has been described.

and in [3] as

"....the deviation of the output signal or output code of a converter from a straight line drawn through zero and full scale. Output signals or output codes must be corrected from a possible zero offset".

The straight line drawn can either be "end-point line" connecting the end points of the characteristics or a "best-fit line". There can be different "best-fit" lines but the minimum mean square fit is often used. Each one has its own pros and cons, and there is no single factor specifying one to be superior to the other. While the "end-point line" is easy to measure (from testing perspective), it does not provide good insight into converter operation. On the other hand, the "best-fit line", though more tedious to measure, gives a better indication of the Total Harmonic Distortion (THD) that will be exhibited. DNL and INL are related as

The INL of any code is the summation of DNL of all codes below it.

$$INL_i = \sum_{j=0}^{i} DNL_j \tag{2.28}$$

INL is then,

$$\text{INL} = \text{Max}\{\text{INL}_i\} \tag{2.29}$$

Both DNL and INL are important as they are sensitive to different issues. INL is more sensitive to cumulative effects while DNL is more sensitive to individual codes.

G. Monotonicity

Monotonicity is one another parameter usually specified for any converter. Monotonic converter as defined in [1] is

"A monotonic D/A converter is one in which the output always increases as the input increases. In other words, the slope of the D/A converter's transfer response is of only one sign. If the maximum DNL error is less than 1LSB, then a D/A converter is guaranteed to be monotonic "

whereas in [3] it is defined as

"Monotonicity of a converter means that the output of, for example, a D/A converter never decreases with an increasing digital input code. A minimum increase of zero is allowed for a 1LSB increase in input signal in a D/A converter".

Monotonicity is guaranteed if the DNL_i is not more negative than -1LSB for all 'i' but it can be more positive than 1LSB. Equivalently, a D/A is monotonic iff,

$$Stepactual_i >= 0;\ i = 1....S \tag{2.30}$$

2.4. Conclusion

A comparison of basic specifications of Data Converter performance has been made and the ambiguities related to them have been addressed. An attempt has been made to present precise, realistic, easily measured and unambiguous definitions of the key performance parameters. Although

not all the parameters have been dealt with, most of the key issues relating to a converter's static performance (in particular D/A's) have been considered.

2.5. References

[1] David A. Johns and Ken Martin. *Analog Integrated Circuit Design.* John Wiley & Sons, Inc., 1997.

[2] Analog Devices. *Analog-Digital Conversion Handbook.* Prentice Hall, Englewood Cliffs, New Jersey, 1986.

[3] R. Van de Plassche. *Integrated Analog-to-Digital and Digital-to-Analog Converters.* Kluwer Academic Publishers, Dordrecht, the Netherlands, 1994.

[4] S.K.Tewksbury et al., *Terminology Related to the Performance of S/H, A/D and D/A Circuits,* IEEE Transactions on Circuits and Systems, CAS-25, Vol.CAS-25, pp.419-426, July 1978.

3. DAC CALIBRATION

This chapter discusses in detail a new approach for calibrating the Integral Nonlinearity (INL) of a 16-bit resistor string Digital-to-Analog Converter (DAC). The block level description of the DAC under consideration is shown in Figure 3.1. The 10 MSBs of the DAC are realized using a resistor string that divides the supply range into 1024 levels, each ideally equal to 1 Least Significant Bit (LSB) at the 10-bit resolution level. Two adjacent nodes of the resistor string, dependent upon the 10-bit digital input to the MSB array, are presented to an Interpolating amplifier. The 6LSBs are then used to resolve the voltage between the two nodes into 64 levels using an interpolating amplifier architecture (designed and patented by Texas Instruments, Inc.), resulting in an overall 16 bit resolution.

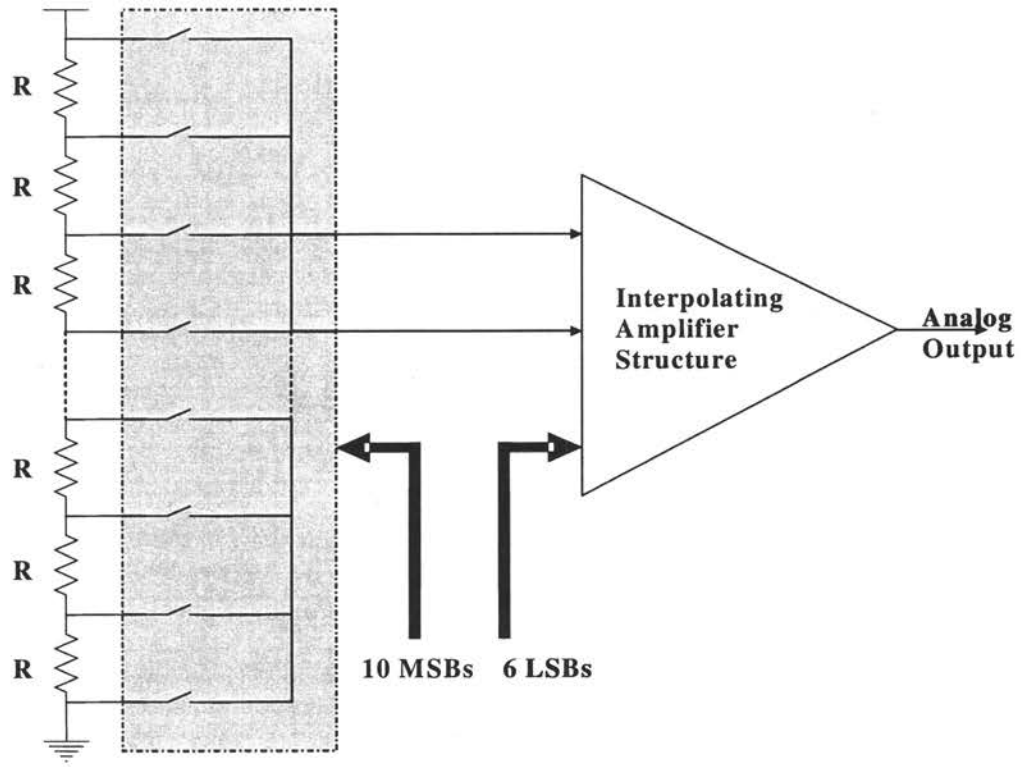


Figure.3.1 MainDAC Architecture

Bench test results of typical DAC structures fabricated by Texas Instruments, in a basic CMOS process with the above architecture show INL readings to be in the range of +/- 64LSBs, thus giving an effective INL performance at the 9-bit level. These performance limitations are primarily due to the matching accuracy of the unit resistors in the resistor string. In the Texas Instruments environment, the resistors are typically laser trimmed after production to achieve the desired 16-bit performance. But laser trim in turn results in a higher production costs and hence high IC costs. The aim of this work is to improve the performance of the DAC by digital calibration, thereby eliminating the need for the laser trim.

The suggested method is to digitally calibrate the DAC based on performance results obtained from a tester. At the time of final testing, the INL and nonlinearity of the uncalibrated DAC will be measured using the tester and the linearity information at certain specific input codes will be permanently stored in registers. The values stored in the registers are then used to calibrate the DAC. The registers can be implemented by using a field programmable ROM or a fuse array.

For any given code, INL calibration is performed by first obtaining the approximate value of the INL at that code based on the calibration data stored in the registers. A signal corresponding to the same magnitude as the estimated INL is then subtracted from the output of the MainDAC to obtain the final signal. As a result, the calibration approach suggested in this work does not actually modify the architecture of the original DAC but just digitally calibrates the performance.

The calibration steps and the architectural details are explained in Section 3.1. The different digital and analog blocks required to implement the scheme are described in Section 3.2 and Section 3.3 respectively. Extensions of this calibration scheme that can result in further improvements in performance are proposed in Section 3.4.

3.1. Calibration Algorithm

As explained in Chapter 2, there are various ways in which the Integral non-linearity of a device is defined. In this work, Integral non-linearity will refer to the deviation of the output signal (or output code) of a converter from a straight line drawn by connecting the end-point of the converter's transfer characteristics. INL_i is the Integral non-linearity at output code 'i' in the case of an ADC or the Integral non-linearity at the analog output corresponding to a digital input code 'i' in the case of a DAC. INL of the converter then refers to the maximum of all INL_i values.

3.1.1. Calibration Steps

Consider as an example a DAC with the INL profile shown in Figure 3.2. The INL depicted in the figure is a sample INL plot. The horizontal axis is the input code to the DAC and the vertical axis shows the INL corresponding to each input code. Bench test results of the 16-bit DACs described earlier indicate that the maximum INL of the parts obtained with an existing design in an existing process range up to 64 LSBs resulting in 9-bit effective INL performance. The goal of the research work is to use the information that is obtained from the tester along with some additional circuitry to perform digital calibration. The targeted improvement in INL is 3-4 bits of improvement in performance. It will be assumed that the INL of the uncalibrated structure is dominantly of a low spatial frequency.

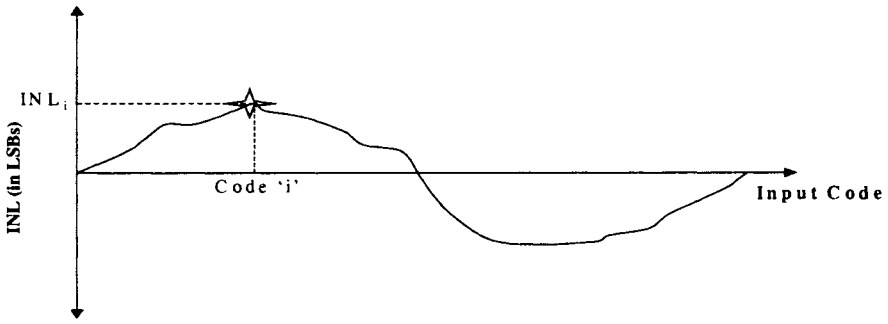


Figure.3.2 Sample INL profile for a typical DAC

The proposed method is based on capturing the low spatial frequency characteristics of the INL by obtaining a piecewise linear approximation to the actual INL data. This can be achieved by storing the low spatial frequency INL information at some specific predetermined codes during final testing. The INL at the intermediate codes can then be obtained by linear interpolation. The architectures used to implement this calibration concept will be introduced in the next section.

To understand the concept, consider a segment of the INL profile shown by the dotted line in Figure.3.3. This is another example INL profile. Based on the wave shape of the INL, a predefined set of N input codes are first selected. Each code location 'X_i', and the corresponding measured INL_i at that code, 'Y_i', constitute what we term a "control point", C (X_i, Y_i). The strategy behind choosing the x-coordinates of the control points will be discussed in a later section. The 'Y_i' values are obtained from the tester during final testing and are permanently stored in registers. With these stored values, a piecewise linear approximation to the INL curve is obtained by linear interpolation of the INL values at the control points. This curve captures the low spatial frequency characteristics of the INL data. The approximate INL for any input code can then be found as described below:

Let X_1, X_2, X_3.....X_N be the x-axis coordinates of the control points and Y_1, Y_2, Y_3.....Y_N be the measured INL value, corresponding to the y-axis coordinates of the control points. It will be assumed that any input code x_k is bounded by X_1 and X_N, that is, $X_1 \leq x_k \leq X_N$. Let x_k be any arbitrary input code. To obtain an estimate of the INL value (y_k) corresponding to the given x_k, the following operations need to be performed.

Step 1: Select the pair of control points (C_i and C_{i+1}) between which the input code lies.

Step 2: Interpolate to obtain an estimate of the INL at the input code x_k, by using the INL values at the control points obtained in step 1. The equation relating the INL values at the control points to the INL value at the given input code, y_k, is given by

$$y_k = Y_i + \left(\frac{Y_{i+1} - Y_i}{X_{i+1} - X_i} \right)(x_k - X_i) \tag{3.1}$$

Once an approximate value of the INL is obtained for the given input code by using (3.1), a calibration DAC (CalDAC) is used to convert the interpolated INL which can be represented with a digital code to an analog signal which is then subtracted from the output of the MainDAC. The architecture that is used to realize this calibration scheme is given in the next subsection.

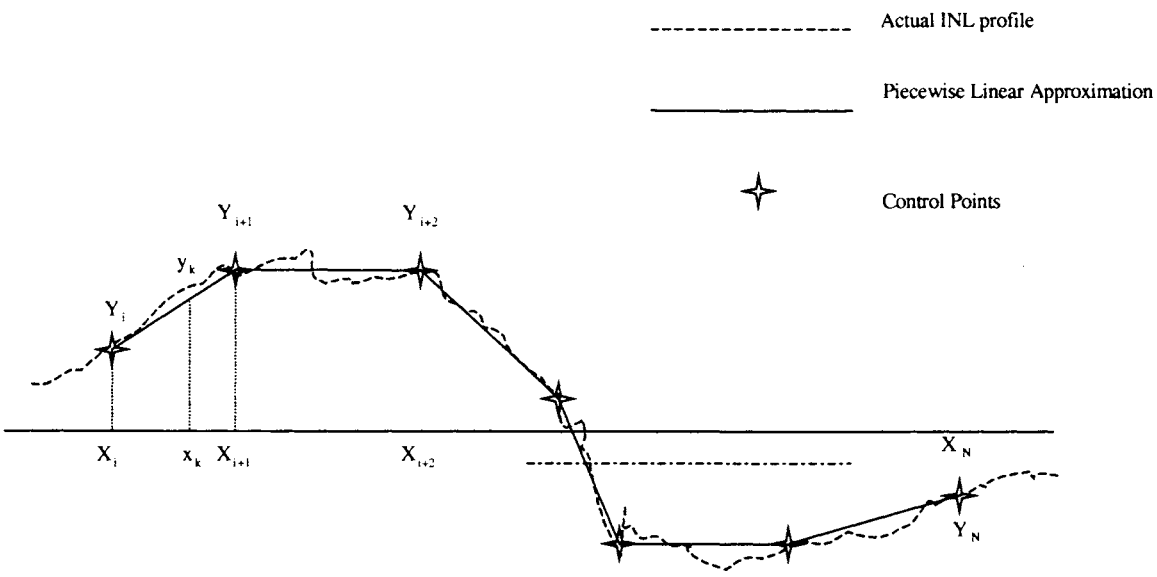


Figure.3.3 Piecewise linear approximation to INL curve

3.1.2. Calibration Details

The block diagram of the proposed architecture that is used to implement the calibration scheme is given in Figure 3.4. The system consists of a MainDAC (16-bit string DAC which is to be calibrated), a Calibration DAC (CalDAC), a fuse block to store INL values at control points, data registers and digital blocks for the various arithmetic operations that need to be performed while evaluating (3.1). As shown in Figure 3.4 the output of the MainDAC is first sent to the tester. The

output is digitized and the INL is evaluated from the captured data. Fuses are then blown based on the evaluated INL values at the control points. Each bank of fuses stores the INL value (in LSBs) at a particular control point.

For any given input code, the correct pair of control points, between which the input lies, is first obtained. The INL value corresponding to the input is then derived by using the slope of the line joining the 'Y' axis values of the two control points. The arithmetic logic blocks shown in the figure perform these operations. The output of the digital block is then sent to the CalDAC, whose analog output is subtracted from the MainDAC output.

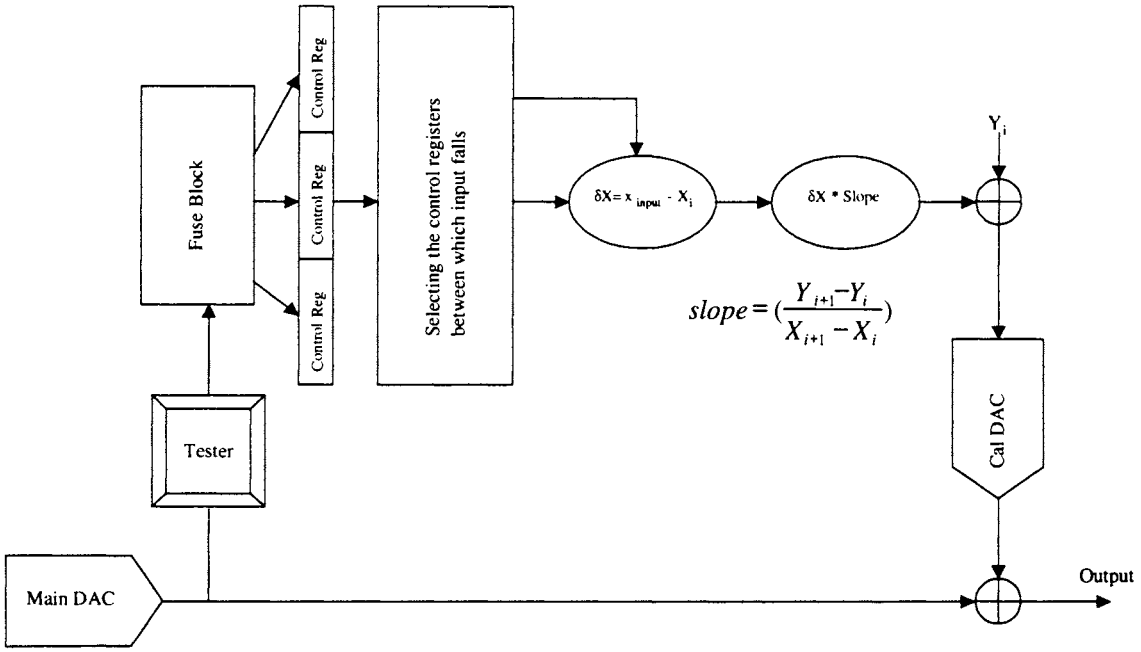


Figure.3.4 Block diagram of the proposed architecture

Having decided the general architecture, the next step is to obtain the specifications of the building blocks. The description of the various blocks and the associated design constraints are listed below.

Number of control points, fuses & control word length: – The first parameter to be determined is the total number of control points. It is apparent that more the number of control points, the more accurate approximation of the INL can be obtained. But on the other hand, increasing the number of control points increases the number of fuses that needs to be burned and this jeopardizes the reliability. Typically each control point requires two sets of fuses. One set of fuses is required for storing the x-axis coordinates of the control point and the other for storing the y-axis value. However, by careful selection of the control points the required number of fuses can be reduced. By selecting the control points that have fixed and predetermined x-axis coordinates, the need for blowing fuses to store the x-axis values can be eliminated. One set of fuses to store the y-axis values is then sufficient for each control point. The first and last control points can represent the offset and gain error of the DAC. Assuming that the device has +/- 64LSB INL as worst case, 7 bits are sufficient for coding the INL value (including the information of polarity) at each control point. The error in INL representation will then be within +/-1LSB at each control point. Any more bits would be over-killing due to the inherent inaccuracy associated with piecewise linear approximation. Thus, 7 fuses will be required for each control point. With the total number of codes in a 16-bit converter being 65536, and if the entire range is divided into 8 segments (each 8192 codes wide), then 9 control points are required (including zero scale and full-scale points). This corresponds to a total of 63 fuses. This is high but not unreasonable.

Control Block: - This is the block that determines the correct pair of control points between which the given input code lies. This is achieved by comparing the first few MSBs of the input code and that of the different control points. Since the entire input range is divided into 8 segments, the 3 MSBs would uniquely identify the segment corresponding to any given input code. The exact architecture of the block with the various input/output signals is explained in the next section.

Simplification of the arithmetic blocks

Division operation: - The calculation of slope in (3.1) requires division by the difference in the x-axis value of two control points. To make this division operation simple, the control points are chosen such that the difference between the x-axis values of adjacent control point is an 'integer power of 2'. The division then corresponds to a simple arithmetic right shift. The right shift operation can be performed without any additional hardware by just connecting the data buses in an appropriate manner in the final circuit. This completely eliminates the requirement of the shift register. As explained above, if the entire input range is divided into 8 segments, then each segment corresponds to 8192 codes, and the division operation is just an arithmetic right shift by 13 bits.

Digital Multiplier: - The numerator in (3.1) requires the computation of the product of $(Y_{i+1} - Y_i)$ and $(x_k - X_i)$. Based on the INL specifications of +/- 64LSBs for the uncalibrated DAC, the difference of the y-axis of two control points can range from +128 to -128. If it is assumed that the maximum difference can be 127 on the positive scale instead of 128, then 8 bits are sufficient to code this data. Also, as will be explained in the discussion on monotonicity, minimum 14 bits are required to represent the difference between the x-axis value of the previous control point and the input code $(x_k - X_i)$. Hence a 14*8 multiplier is needed to calculate the product term in (3.1).

Accuracy of the Calibration DAC (CalDAC): - From worst-case test result scenario, we know that INL of the uncalibrated DAC can be +/-64LSBs amounting to equivalent accuracy of 9-bits (+/-0.5LSB at 9 bits). Since the MainDAC is just 9-bit accurate, the signal that is obtained from the output of the CalDAC needs to be at least 11-12 bits accurate to make calibration reasonable. But this raises a fundamental question. If we cannot design a more accurate MainDAC, how can we design a more accurate CalDAC? In the proposed approach, design redundancy and tester feedback will be used to solve this problem. The CalDAC range is first scaled by a factor of 256 of MainDAC full-

scale range. The scaled range then corresponds to +/-1LSBs at 9 bits of MainDAC, which is twice the MainDAC INL range. Since scaling cannot be done accurately in silicon, assuming a +/- 25% full-scale error during scaling, the CalDAC range can still cover the MainDAC INL range.

Consider the situation shown in Figure 3.5. It is necessary that the CalDAC full-scale range is nearly twice the INL range of the MainDAC. This is because, if +/- 25% overall scaling error is assumed as discussed above, the CalDAC should still have enough range to correct for the MainDAC INL. Thus as shown in Figure 3.5, with a –25% scaling error on the CalDAC high reference voltage and +25% error on the CalDAC low reference voltage, the CalDAC still has enough range to cover the worst case MainDAC INL.

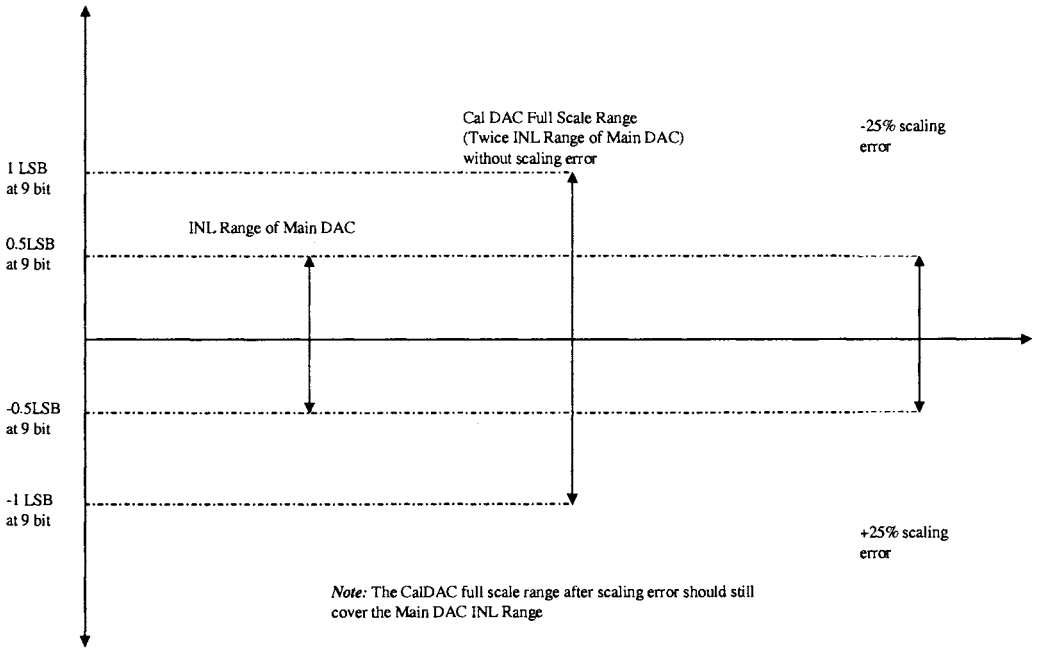


Figure.3.5 CalDAC range vs. MainDAC range

It is also necessary that the CalDAC has sufficient resolution (8-10bits) to resolve the range with minor CalDAC INL and DNL errors (more explanation on resolution is provided in the discussion on monotonicity). It is reasonable to expect that the zero-scale and the full-scale error of the CalDAC will dominate the overall CalDAC errors. The zero-scale error will simply add as an offset to the MainDAC. The full-scale error however needs to be accounted for. This effect of CalDAC full-scale error can be eliminated by evaluating a partial INL once more in the test program, as explained below.

Step 1: - The output of the CalDAC is first set to zero and the tester measures the INL of the MainDAC. INL is measured only at the control points and is fast (need to measure only at 9 control points). Based on the INL data at the control points, the trim words are calculated and written into temporary registers.

Step 2: - The tester measures the INL again at the control points with the temporary trim words in effect. By using the trim words evaluated in Step 1 and by using the CalDAC along with the MainDAC, the INL should ideally be within 1LSB at the control points. Any deviation from this will be due to the errors in the CalDAC. From the new INL readings, the CalDAC errors can be estimated and can be accounted for by readjusting the trim coefficients.

Step 3: - Fuses are blown according to the readjusted trim coefficients. 'All-codes' INL is evaluated for specification compliance.

The calibration cycle thus consists of two partial INL evaluations (only at control points) and one final INL evaluation (at all codes). The first partial INL evaluation gives information about the MainDAC inaccuracies and the second partial INL evaluation gives information about the CalDAC inaccuracies. The trim coefficients are then decided based on the results of both partial INL evaluations. Thus by repeating the test twice, the CalDAC is also calibrated.

Monotonicity: - As per the specification, the untrimmed resistor string DAC described above is 16-bit monotonic, with typical DNL of +/- 0.5LSBs. It is essential to make sure that the calibration step does not improve INL at the cost of DNL. The DNL budget for calibration should therefore be limited to not more than +/- 0.125 LSBs at 16 bits, in order to maintain monotonicity. If the CalDAC range is +/- 1LSB at 9 bits (assuming scale-down by a factor of 256), and if the CalDAC has 8 bits of resolution, we get a CalDAC related DNL error of +/- 0.5LSBs at 16 bits, which is more than the allotted margin. Therefore, the CalDAC needs to have at least 10 or more bits of resolution to get a CalDAC related DNL at the output equal to or less than +/- 0.125LSB. This is no extra burden if we use the R-2R architecture for CalDAC.

With a MainDAC INL specification of +/- 64LSBs, the worst case INL difference between two consecutive control points can be +128LSBs or -128LSBs. This requires a 10-bit calibration DAC as explained above. However, if we include a '2x redundancy' in INL specs of the MainDAC, we need to set the CalDAC range to +/- 256LSBs at 16-bit. Considering a 7-bit range division (instead of 8 bit as explained earlier), and resolution of 0.125LSBs at 16bits, a 12-bit calibration DAC is then required. So to ensure monotonicity, we need a range divider by 128 and a 12-bit CalDAC.

Monotonicity should not only be guaranteed by the choice of CalDAC architecture but also with the quantization of the arithmetic. The DNL transitions related to CalDAC outputs should be "smooth" between input codes. Let us assume that there are 8 piecewise linear segments of 8192 codes and 9 control points, and a +128 LSB INL error occurs between two control points as a worst case condition. To maintain monotonicity, we need to ensure that no CalDAC increment larger than 0.125LSB is added at the output for any two consecutive input-codes. At steps of 0.125LSBs, 1024 different steps are required to correct 128LSB INL error. As each segment is 8192 codes wide, it follows that 8 consecutive input codes result in same CalDAC output. Thus the last three LSBs of the MainDAC input word can be neglected in the arithmetic operations without loss of any information.

In other words, at least 13 MSBs should be used to represent the input code and the control point x-axis value to guarantee step sizes not larger than 0.125LSBs at the output. With some margin, we will use 14MSBs to represent the data.

Re-writing (3.1) to reflect the arithmetic quantization, we get:

$$Cal\_dac\_out_{12MSBs} = Y_{i7MSBs} + \left(\frac{(Y_{i+1} - Y_i)_{8MSBs}}{(X_{i+1} - X_i)_{14MSBs}} (x_i - X_k)_{14MSBs} \right)_{11MSBs} \quad (3.2)$$

The terms in (3.2) are left aligned. The subscripts 'I' *MSBs* mean that 'I' most significant digits are used in the arithmetic. Also since each segment is of length 8192 (13 bit representation), and the last two LSBs of 16bits are ignored, the division by (X_{i+1} - X_i) is effectively an arithmetic right shift by 13-2=11bits.

To summarize, the CalDAC uses 7bits of reference scaling and is of 12bit resolution. Effectively it has 19 bits of resolution. The first 7 bits are obtained by scaling the supply range by a factor of 128. This is twice the range required to correct for worst case MainDAC INL. CalDAC full-scale inaccuracy can also be measured and corrected within this range. The remaining 12 bits consists of an R-2R (or a partially segmented R-2R) type architecture (the rationale behind the choice of architecture is explained in a later section).

3.2. Digital Circuitry

The design and implementation of the digital portion of the calibration circuit is explained in this section. Consider the block diagram shown in Figure 3.6. The various digital sub modules that need to be designed are:

- Fuse Block for storing INL values of control points (not considered in this work)
- Multiplier block
- Adder block
- Subtractor block
- Control block - required for deciding the two control points between which the input lies.

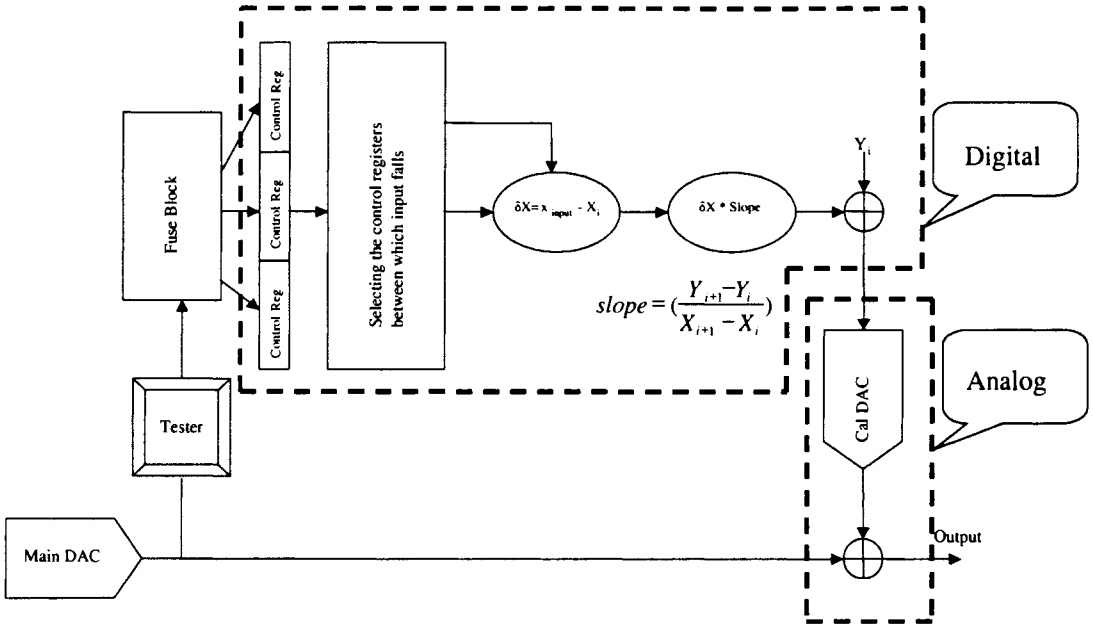


Figure.3.6 Analog and Digital portion of the architecture

The output of the digital block is a 12-bit binary word that is given as input to the CalDAC. The analog output of the CalDAC is then subtracted from the MainDAC signal.

The analog blocks required are:

- CalDAC
- Summing Circuit
- Reference Signal Generators

- Vref,High for CalDAC
- Vref,Low for CalDAC
- Voffset (to be discussed later)

This section describes in detail the design, implementation and testing of the digital blocks. Section 3.3 deals with the various analog blocks listed above.

3.2.1. Module Description

To determine the input-output specifications of the various modules, let us consider (3.2) again:

$$Cal\_dac\_out_{12MSBs} = Y_{i\,7MSBs} + \left(\frac{(Y_{i+1} - Y_i)_{8MSBs}}{(X_{i+1} - X_i)_{14MSBs}} (x_i - X_k)_{14MSBs} \right)_{11MSBs} \tag{3.2}$$

Step 1: The first step is to find the appropriate segment to which the given input code belongs. This enables us to determine the control points INL values (namely Y_{i+1} and Y_i). This is achieved in the Control Block. As there are only 8 segments the top 3 MSBs are sufficient to determine the appropriate segment to which the input code belongs. Only the 3 MSBs along with the fuse values are given as input to the control block. Based on the 3 MSB values the control block determines the appropriate segment to which the input belongs and outputs the corresponding end point INL values.

Step 2: Once Y_{i+1} and Y_i are obtained the difference of the two INL values is calculated (refer to (3.2)). Each control point INL values can range from –64 to 63 (assumed 63 and not 64 so that it can be coded in 7 bits) and are stored in 2's compliment form in the fuses. Since the difference of Y_{i+1} and Y_i can range from –128 to 127, the output of the subtractor block needs to be 8 bits wide.

Step 3: The difference between the given input code and the input code corresponding to the control point representing the lower end of the segment to which the input belongs, $x_i - X_k$, is then calculated.

From (3.2), we see that 14MSBs need to be considered while finding the difference. At first look, it may seem that a 14-bit subtractor is necessary to estimate this value. But the lower 13 bits of the control point x-axis values are always '0' (for e.g. when the input code corresponding to control points at 8192, 16384 etc are represented in binary format, the lower 13 bits are always '0'). Thus the difference between x_i and X_k corresponds to just the lower 13 bits of the input code. The 13LSBs of input code, (which can range from 0 when the input code is equal to the lower end of the segment, to 8191 when the input code is equal to the upper end of the segment) then provides the data corresponding to $x_k - X_i$. Refer to Figure 3.7 for an example. Also since all data are represented in 2's compliment form, this difference (which can range from 0 to 8191) is also in 2'compliment and is 14bit wide.

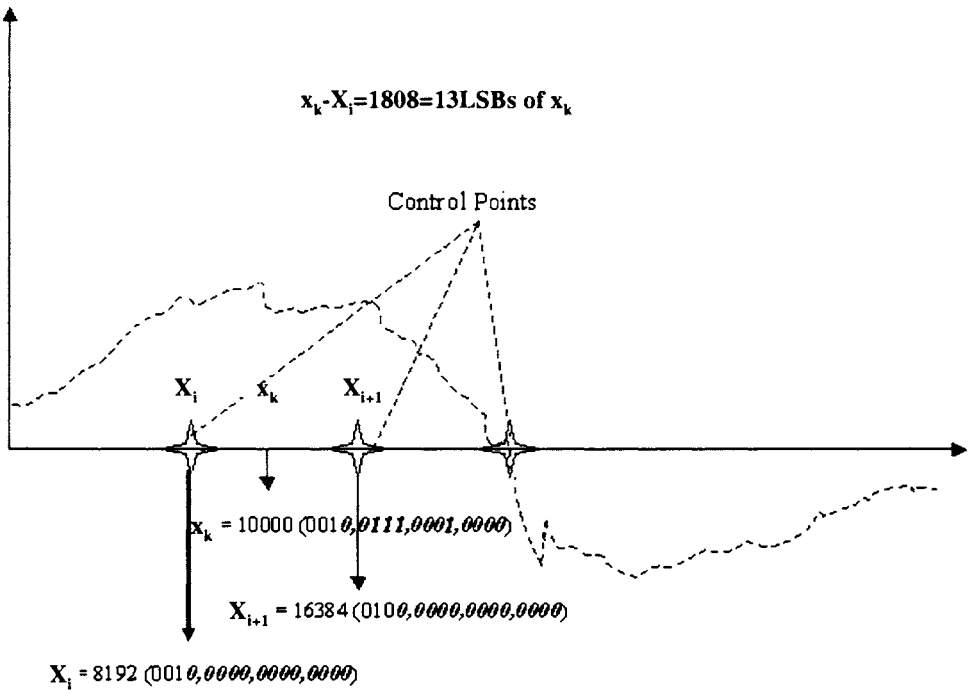


Figure.3.7 Example showing the estimation of $x_k - X_i$

Step 4: As the control points are placed at equal distances of 8192 input codes, the division operation by $(X_{i+1} - X_i)$ is equivalent to binary right shift by 13 bits.

Step 5: The next module to be determined is the multiplier which takes as input the difference of control point INL values (obtained in Step 2) and the difference of the input code and segment end point (obtained in Step 3).

- Difference of control point INL values $(Y_{i+1} - Y_i)$ can range from –128 to 127 (represented in 8 bits)
- Difference of input code and segment end point can range from 0 to 8191 (represented in 14 bits)
- The product can range from –1048448 to 1040257 (can be represented in 22 bits)

Step 6: As seen earlier the division by $(X_{i+1} - X_i)$ corresponds to division by 8192. However, the INL values stored in fuses are in terms MainDAC LSBs. However, since one CalDAC LSB equals 1/8 MainDAC LSB, a factor of 8 needs to be multiplied to get the appropriate output that needs to be supplied to CalDAC. Instead of first dividing the product obtained from the multiplier block by 8192 and then multiplying by a factor of 8, the product is divided by 1024. This reduces the error due to finite bit representation that could arise if the multiplication is first performed followed by a division operation.

Step 7: The 22bit output of the multiplier is reduced to 12bit by dropping the 10 LSBs; amounting to division by a factor of 1024.

Step 8: Since Y_i is represented in 7 bits (in 2's compliment), it first needs to be converted to 12 bits before it can be added to the value obtained in step 7. Also Y_i needs to be multiplied by a factor of '8' to take care of conversion from MainDAC LSB to CalDAC LSB. The multiplication by '8' is achieved by appending three '0's to the end of the 7-bit representation of Y_i. This converts the 7-bit data to 10-bit. To convert from 10bit to 12bit, 2 MSBs are added which is of the same value as the SIGN bit (of the existing 10bit word).

Step 9: The two 12 bit data are then added to get a 13bit output. The MSB is dropped to get a 12 bit data. The value represented by the 12-bit is centered on '0' (from –2048 to 2047). To shift the range such that it represents 0-4095, the MSB is inverted.

Note: By inverting MSB in hardware, the range of values represented by the word shifts from -2048 ~ 2047 to 0 ~ 4095. The 12 bit word obtained after inverting the MSB is then used as input to CalDAC.

Source of Error: Since 10 LSBs are truncated from the multiplier output, the data stored in them is lost, which leads to finite error in the output. The maximum error occurs when all 10LSBs of the 22-bit word are '1', which corresponds to nearly 1LSB of remaining 12-bit value. This is equal to 1 CalDAC LSB or equivalently 0.125 MainDAC LSB. So, the maximum error due to finite arithmetic is 0.125LSB.

Based on the above explanation, the input/output word length of the various blocks can be summarized as follows:

- Fuse block – 7-bit wide and totally 9 blocks amounting to 9*7 = 63 fuses
- Registers – 9 registers (each 7-bit wide) to store the fuse values. In the final design, these registers will be used to store the temporary trim words during the two partial INL evaluation steps. However, in this work these registers are also used to emulate the fuse block for various simulations in the design flow
- Control block – Takes as input the 9 fuse words and 3 MSBs and outputs Yi+1 and Yi (each 7-bit wide)
- Subtractor block – Takes as input Y_{i+1} and Y_i (7-bit each) and outputs the difference (8-bit wide)

- Multiplier block – Takes 2 inputs; one 14-bit vector and other 8-bit vector and outputs 22-bit vector
- Adder block – Takes as input two 12-bit vectors and outputs 13-bit vector.

The complete flowchart of the digital block showing the data paths is given in Figure 3.8.

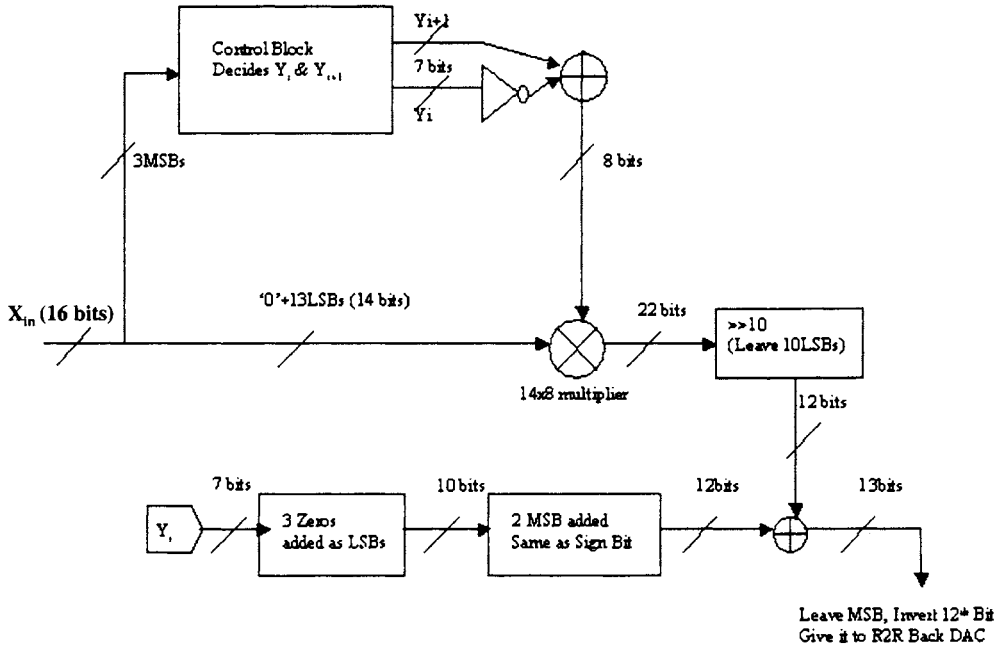


Figure.3.8 Data path flow

3.2.2. Design Flow

The standard design methodology consisting of the following steps was adopted while designing the digital modules.

- Behavioral coding in VHDL

- Synthesizing the behavioral code in synopsys and generating the gate level netlist
- Importing the netlist in cadence and simulating the circuit

Descriptions of the above standard steps along with the results obtained in each step are reported in the sections below.

3.2.2.1. VHDL Coding of Digital Blocks

Register (7-bit)

The VHDL code giving the behavioral description of a register is given in Appendix A - Code1. The first few lines are related to including the standard library and logic. This is followed by the entity description. The register is represented by the term *"Memory"* and has a clock and reset (active if equal to '0') as input along with a 7-bit input vector that represents the fuse value. On the rising edge of the clock, the input vector is copied onto the output vector. This is described in the architecture section of the VHDL code. Also if reset signal is '0', the output is asynchronously forced to '0'. Once the behavioral description was coded, the next step was to write a test-bench and check the functionality. One simple test bench including some of the sample cases is given in Appendix A – Code 2. The output obtained using this sample test-bench is shown in Figure 3.9. As seen from the test bench and the output plot, at every rising edge of the clock, the input is transferred to the output and also when reset signal goes '0', the output is forced to '0', thereby confirming the functionality of the module.

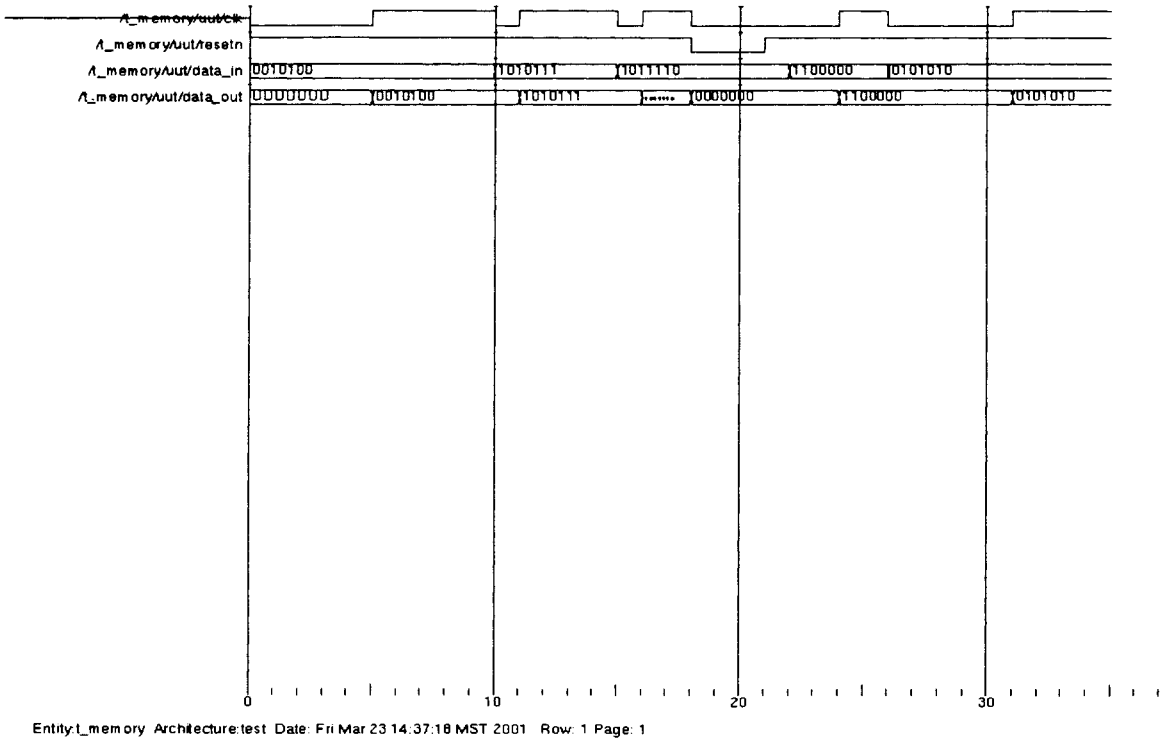


Figure 3.9 Modelsim simulation result for register block

Control Block

The control block, as explained earlier, is responsible for finding the segment to which the given input code belongs and for finding the two end point INL values corresponding to that segment. It takes as input the 3MSBs of the input code and the different fuse values and outputs two 7-bit words that correspond to the INL values at the two control points. The behavioral code of the control block in given in Appendix A – Code 4.

After the initialization of the libraries, the entity description of the control block is given showing the various input/output ports. The control block consists of 9 registers to store the INL values at the 9 control points. These registers are realized using the 7-bit register block described

earlier. A component declaration of the register block (named as *Memory*) is included before instantiation it 9 times to represent the 9 different registers. The instances are termed register 1 to register 9. This is followed by the process description of the control block. Depending on the 3 MSB values, different set of register values (fuse values) are copied to the two outputs as shown in the code. Appendix A – Code 5 gives a sample test bench used for verifying the functionality of the control block. The output of the simulation is shown in Figure 3.10.

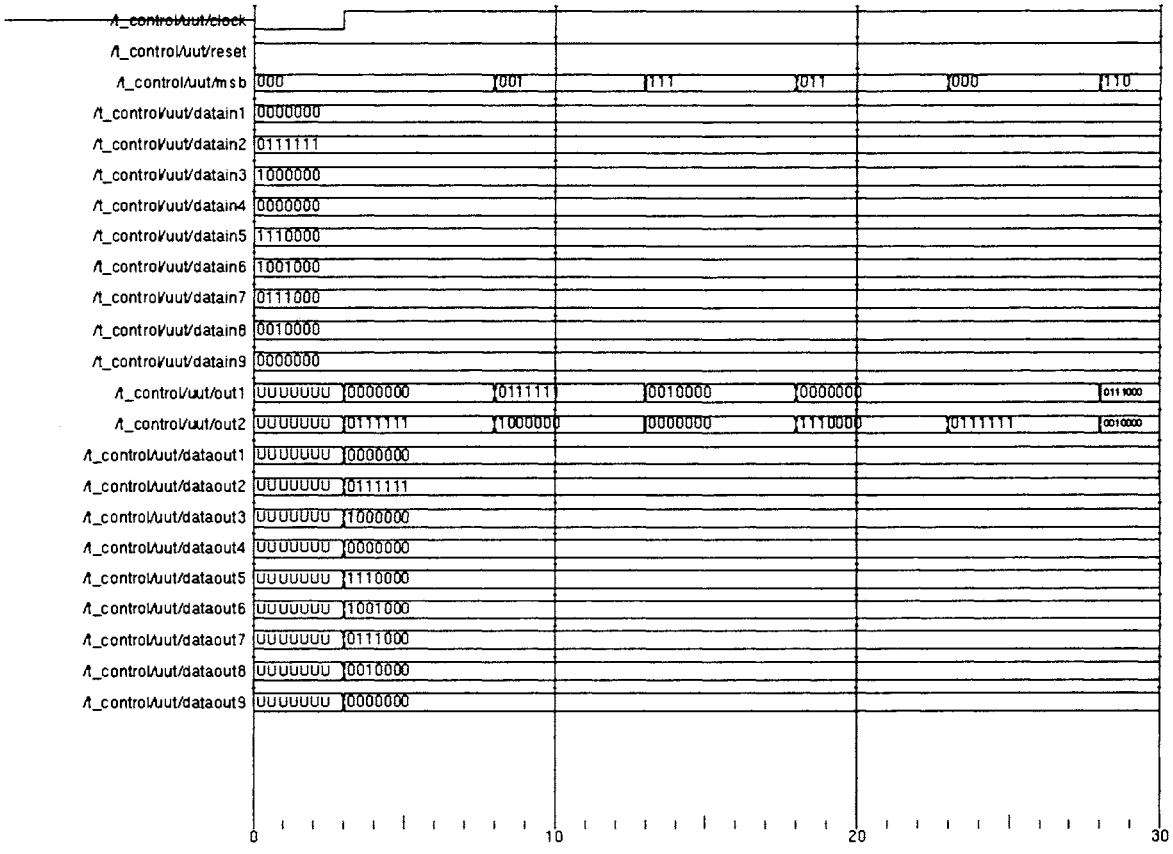


Figure 3.10 Modelsim simulation result for control block

As seen from the test bench and the output plot, all the 9 input vectors (denoted by variable *datain1* through *datain9*) were initialized with some values (that correspond to the INL values in the actual case). At time = 3 ns, clock signal was made '1' and this caused all the 9 registers to latch the fuse values (denoted by *dataout1* to *dataout9*). The 3-bit input code was then changed and the output was

checked for the correct end point fuse values. The different values of input code and the corresponding 7-bit outputs (names as *OUT1* and *OUT2*) are shown in the output plot (Figure 3.10).

Subtractor Module

The behavioral description of the subtractor module is given in Appendix A – Code 7. This block computes the difference of the end point INL values of the segment to which the given input code belongs. It takes as input the two 7-bit vectors that are obtained from the control block. Each of these vectors can range from –64 to 63. The output is represented as an 8-bit vector as the difference can range from –128 to 127. The behavioral code of the subtraction operation is described in a single line as

sum < = a – b

where 'sum' is the output vector and 'a', 'b' are the two 7-bit input vectors.

However when the VHDL code was executed with input vector ('a' and 'b') each represented as 7-bit vector and output vector (sum) represented as 8-bit vector, there was a compilation error due to different word length on either side of the operation. The input vectors were then changed to 8-bit (even though the maximum value that it could assume was still limited to 7-bit). This is achieved by adding 1 bit before the MSB which is of same value as MSB. Appendix A – Code 8 gives the test bench that was used to verify the functionality of the module and the output obtained using the sample test bench is shown in Figure 3.11.

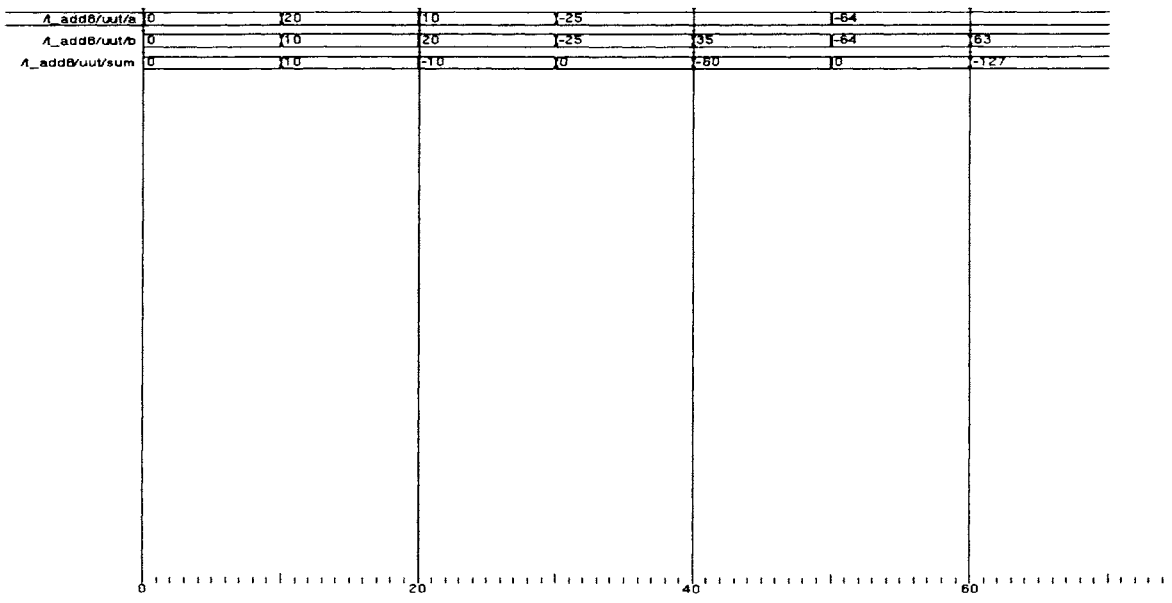


Figure.3.11 Modelsim simulation result for subtractor

Note: To maintain consistency with VHDL requirement, all the signals and variables were chosen to be of the type STD_LOGIC. The fuse values are represented in binary format in 2'compliment notation. The test bench shown above considers different values for 'a' and 'b' (the two input signals) such that all the extreme cases are covered. To enable easy understanding, the output plot gives the value of 'a', 'b' and 'sum' in integer format, although in actual behavioral code it is of data type STD_LOGIC.

14x8 Multiplier

This module computes the product of the data obtained from the subtractor block and the 13LSBs of the input vector. The value obtained from the subtractor block represents the difference of INL values and the 13LSBs of the input code gives the difference between the given input code and the beginning code of the corresponding segment. For reasons explained earlier in the discussion on

monotonicity the multiplier block needs to be 14x8 bit wide. To obtain 14 bit input, a '0' was added at the beginning of the 13LSBs of the input code. The output of this block is 22 bits wide.

When a simple code as given below was tried,

Prod<=a*b

the results generated were incorrect due to the way the numbers were interpreted. The input data was always assumed to be positive while evaluating the product. The case when one of the inputs is negative and the other is positive was not identified properly. As the main focus of this research work was to prove the concept and not to generate an optimized design, a simpler approach was adopted to solve the problem.

It is known that the 14-bit input ('a') is always positive (since it gives the difference of the input code and the lower end code of the segment). The MSB of the 8-bit data ('b') is checked for polarity. If it is '1', then it is a negative number. The 2's compliment of the data is generated and stored in a variable called 'Invert'. The product of 'a' and Invert is then computed (where both of them are positive) and is stored in another variable called 'FirstProd'. Two's compliment of this value then gives the final product. To summarize, if the data 'b' is negative, then it is converted to positive value before multiplying and the output is re-converted to negative value. This is explained in Appendix A – Code 10. Appendix A – Code 11 gives the test bench covering various values of 'a' and 'b'. The output generated using the sample test bench is shown in Figure 3.12.

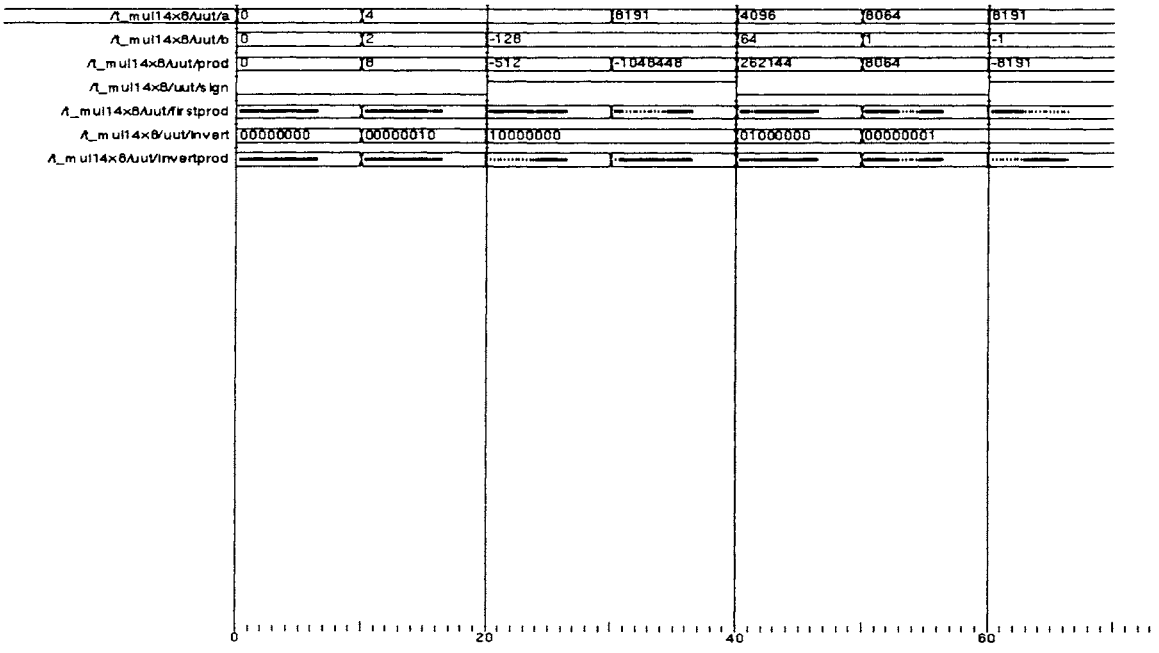


Figure.3.12 Modelsim simulation result for multiplier

What need to be observed from the plot are the values of 'a', 'b' and 'prod'. These variables are shown in integer format to enable easy understanding.

12-bit adder

The adder block calculates the sum of the two 12-bit words that are obtained as explained in step 7 and step 8 in Section 3.2.1. The behavioral code includes a simple addition operation **(sum<=a+b)**. All the signals are represented in 2's compliment and hence the nature of the signal (positive or negative) is not of concern. As in the case of subtraction block, if the input and output were of different length, the code was not properly compiled. Hence the input and output were made 13-bit wide, even though each of the inputs could be represented in just 12 bits. The extra bit is obtained by copying the MSB. Also as the CalDAC is of only 12-bit resolution, the MSB of the output word (13-bit) is dropped before feeding the output of adder to the CalDAC.

Appendix A – Code 13 and Appendix A - Code 14 gives the behavioral description and the test bench of the adder block respectively. The output generated using the sample test bench is shown in Figure 3.13. The signals are represented in integer format to reduce the difficulty of verifying the results.

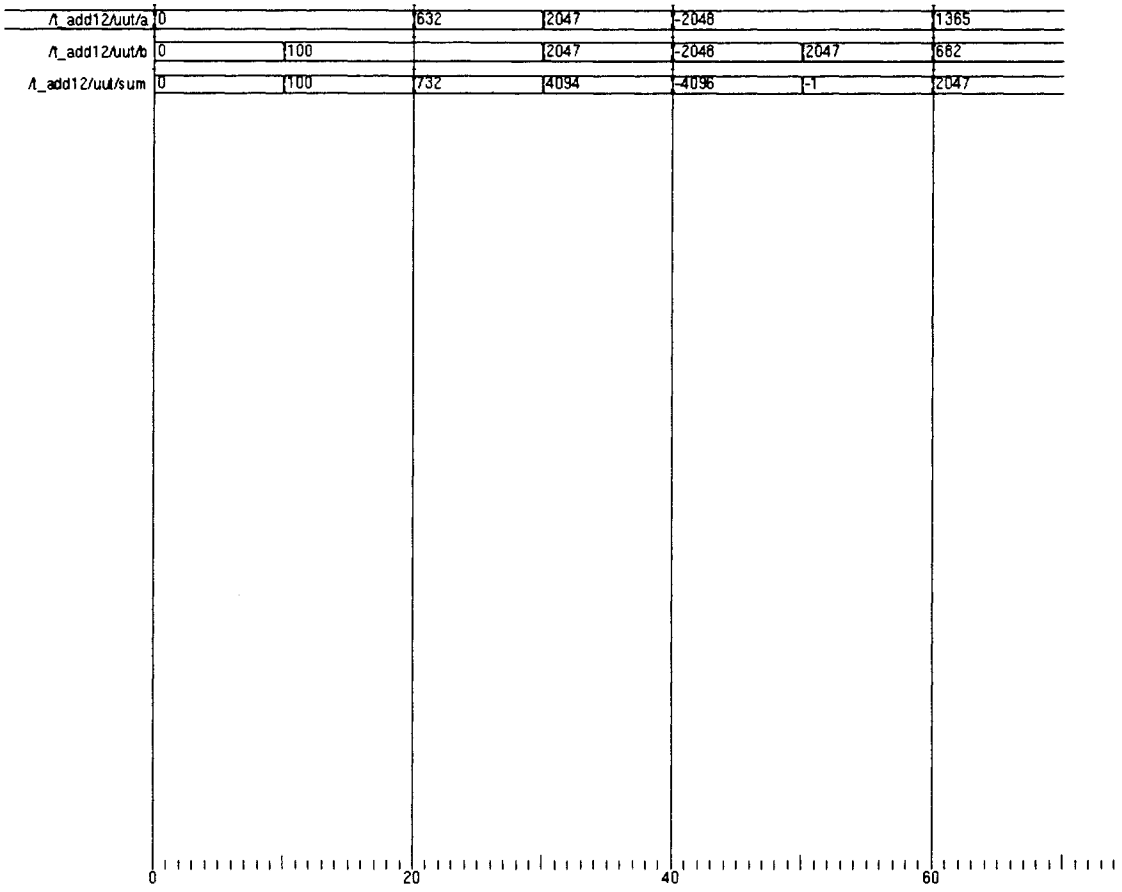


Figure.3.13 Modelsim simulation result for adder

As will be explained in a later section, the CalDAC is designed with a segmented architecture with the 3MSBs being thermometer coded and the 9 LSBs being binary coded. This then requires a decoder block for converting the 3MSB of the adder output to a thermometer code. Appendix A- code 18 and Code 19 gives the behavioral code and test bench that are used for the decoder. Figure 3.14 shows the result obtained using the sample test bench.

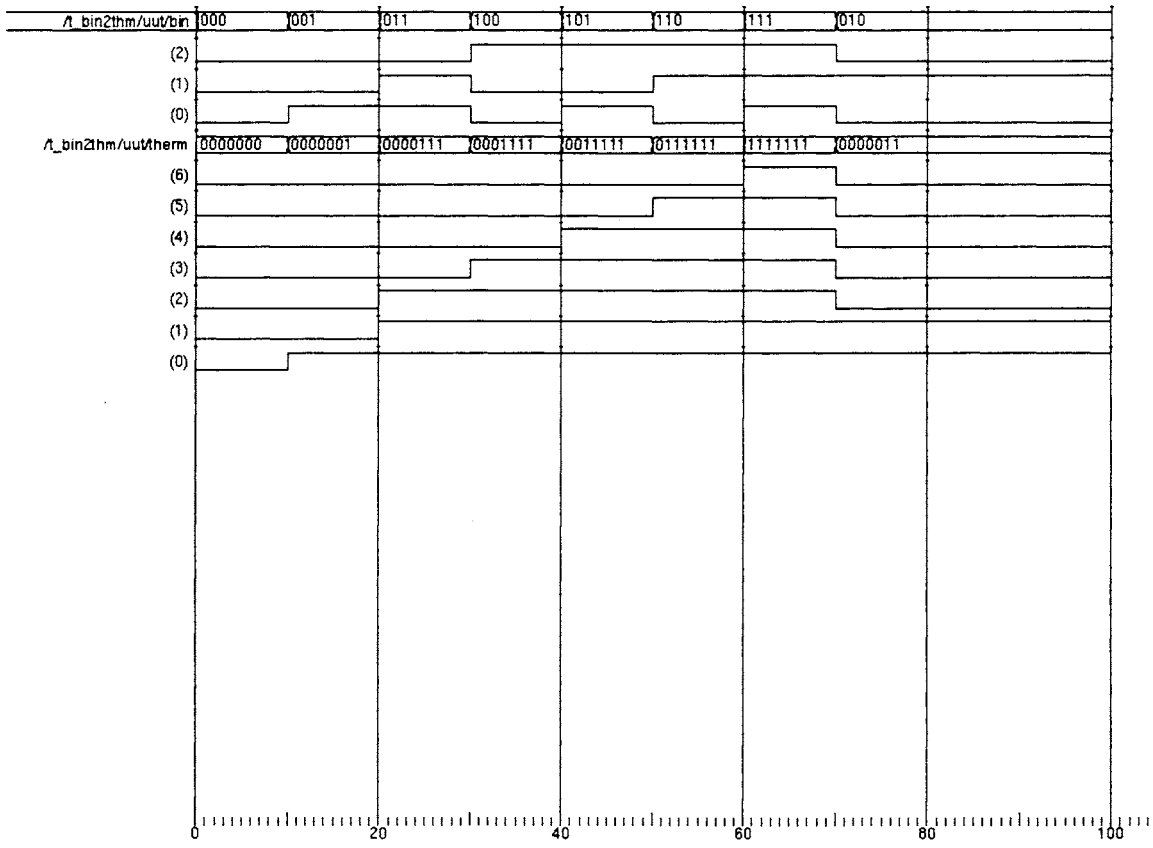


Figure.3.14 Modelsim simulation result for decoder

Note:

As the main idea of this research work was to establish the concept, not much stress was laid in optimizing the code and design of the different digital modules. However, there are two places where the architecture explained till now can be simplified.

Case 1: In the multiplier block, the MSB of the 14-bit input is always '0'. Effectively the input is just 13-bit wide. The multiplier can then be reduced from 14x8 to 13x8. This will result in 21-bit output, the lower 10 bits of which can be dropped.

Case 2: The 12-bit adder explained earlier gives a 13-bit output vector, the MSB of which is neglected before sending the other 12 bits to the CalDAC. As the MSB is not being used, it is not necessary to compute it and the 12-bit adder can be reduced to a 11-bit adder.

Having designed the digital blocks in VHDL, the next step is to synthesize them in Synopsys to obtain the gate level netlist. This is explained in the following section.

3.2.2.2. Synthesis in SYNOPSYS

Register (7-bit)

A standard design flow methodology of importing the VHDL/verilog code in synopsys was first performed. The code was then synthesized in synopsys under different area and timing constraints. Since in this research work, the speed of operation is not very critical (the MainDAC operates at very low speed), the main design criterion was to go for reduced area rather than reduced speed. All the blocks were synthesized with area minimization as the main constraint. The schematic generated for the register block using the synopsys tool is shown in Figure 3.15.

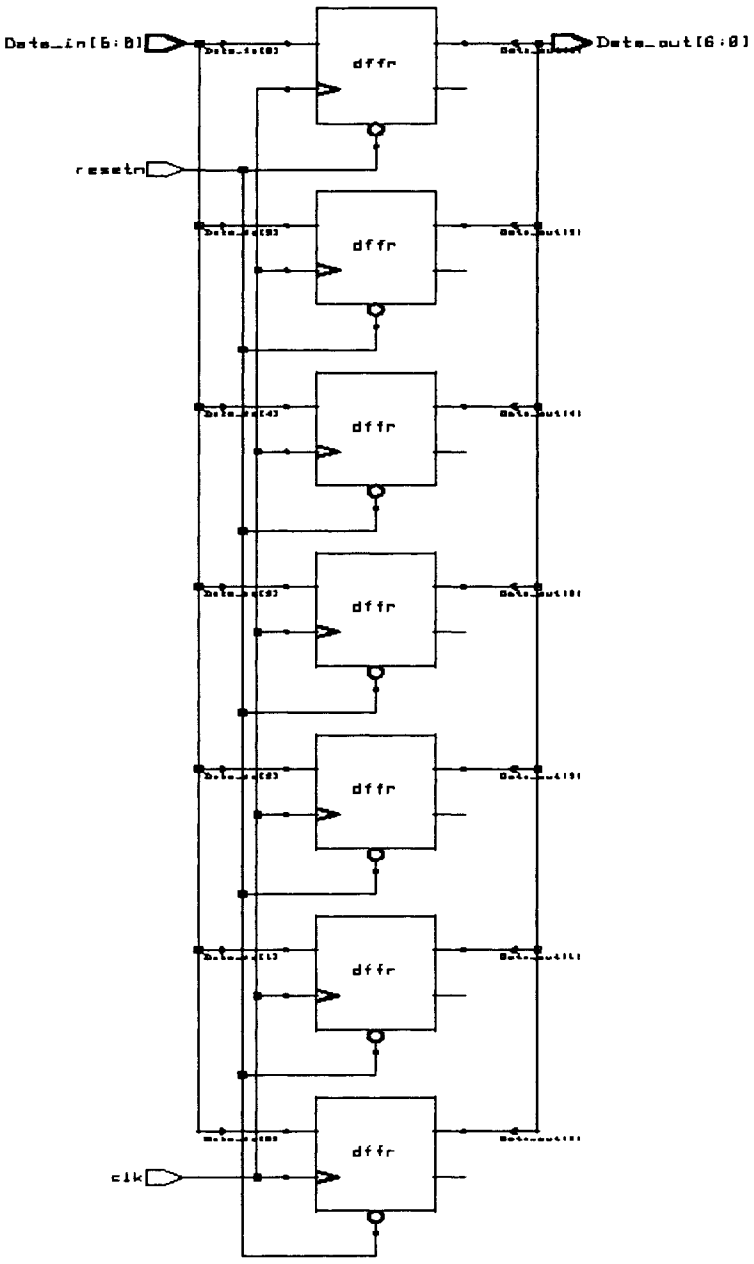


Figure.3.15 Schematic of the register block as generated in Synopsys

Comparing the VHDL code and the schematic obtained above, it can be seen that the register is realized by using an array of D-Flip-Flops, each with a clock and reset pin as input. The gate level netlist was stored as a verilog file *(memory.v)* so that it can be imported in cadence. The sample verilog file for the register block (obtained from synopsys) is given in Appendix A – Code 3.

Control Block

The VHDL code of the control block was synthesized to get a gate level netlist as explained above. As control block uses instances of memory units (which are the register blocks), the register file was read prior to reading the VHDL code of the control block. Figure 3.16 and Figure 3.17 gives the symbol and the gate level schematic of the module as generated using synopsys. The gate level netlist was stored as a verilog file (Appendix A – Code 6) to be imported in cadence.

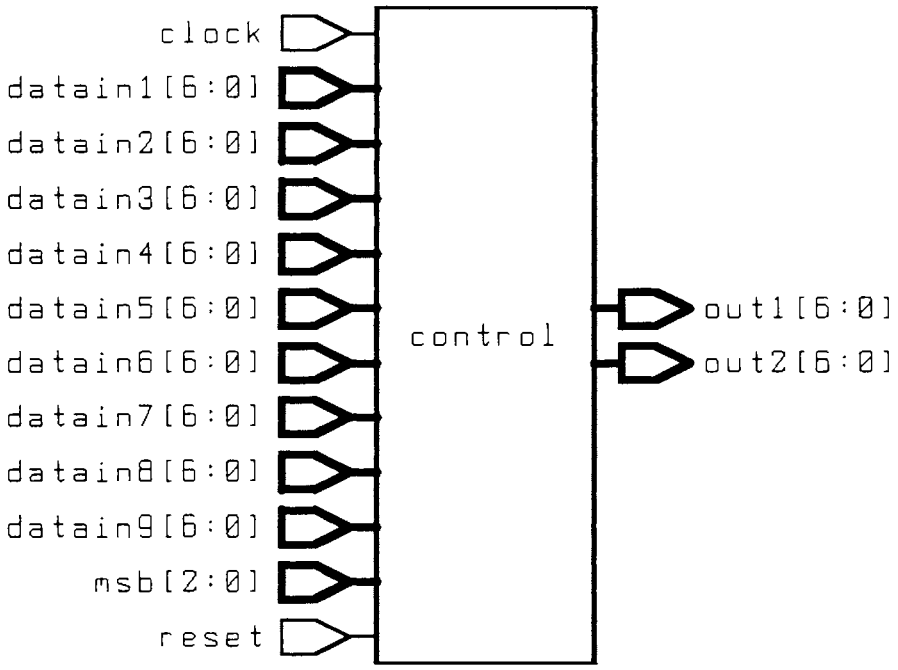


Figure.3.16 Symbol of the control block as generated in Synopsys

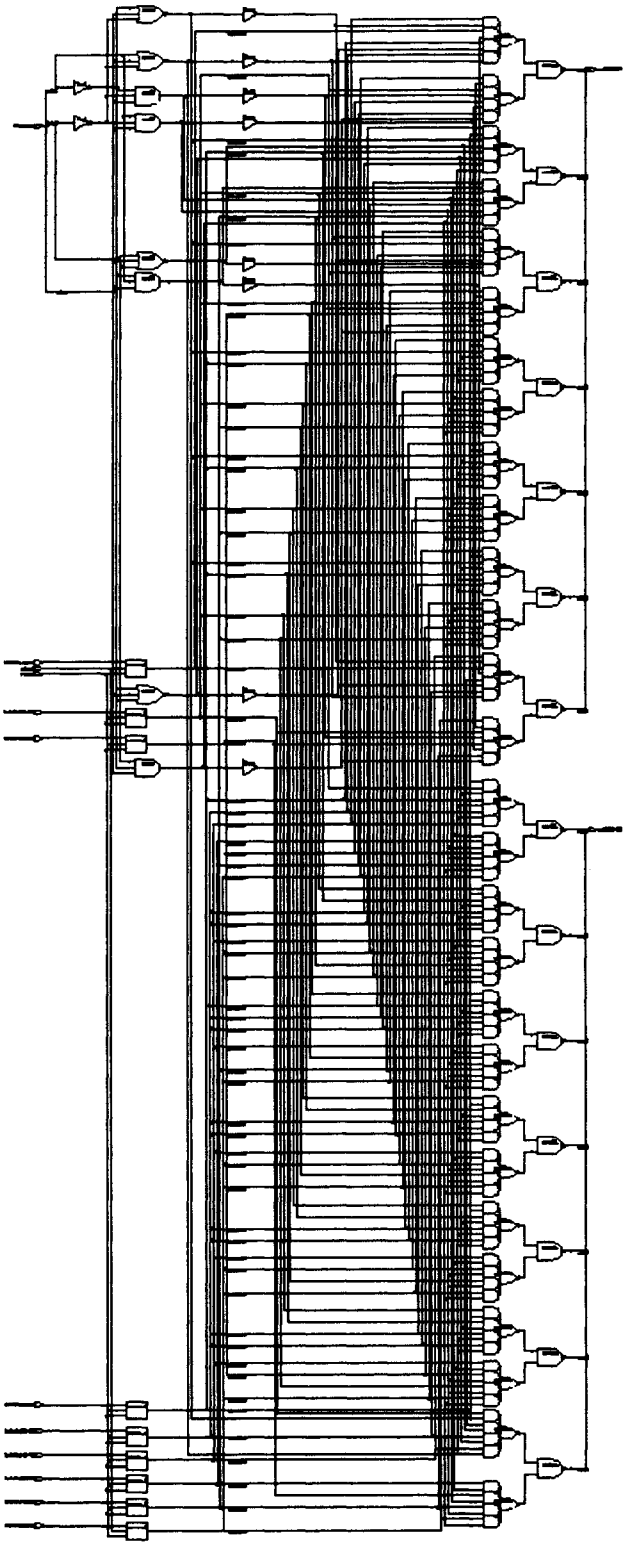


Figure.3.17 Schematic of the control block as generated in Synopsys

Subtractor Block

The subtractor was again synthesized with minimum area constraint. The subtractor file can be read in design analyzer directly without reading any other file, as it does not use any instance of registers or other sub blocks. Figure 3.18 shows the gate level schematic of the subtractor and the associated verilog netlist is given in Appendix A – Code 9.

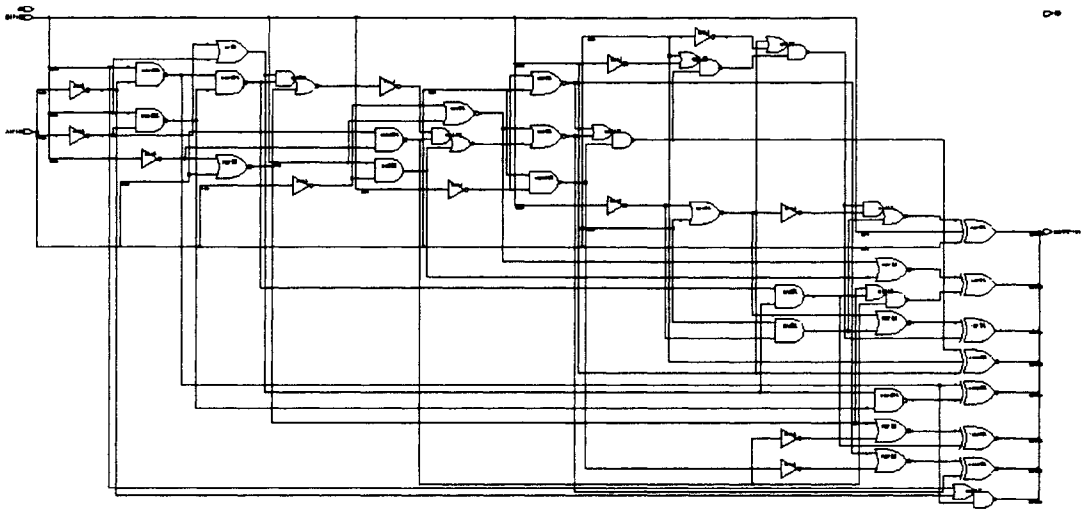


Figure.3.18 Schematic of the subtractor as generated in Synopsys

Multiplier Block

The most area-consuming block in the digital portion of the design is the multiplier block. The VHDL code described in the previous section was synthesized with minimum area constraint. Since these blocks are to be used in the calibration loop and as the MainDAC speed is not very high, the speed of these blocks is not critical. The top-level schematic of the multiplier is given in Figure 3.19.

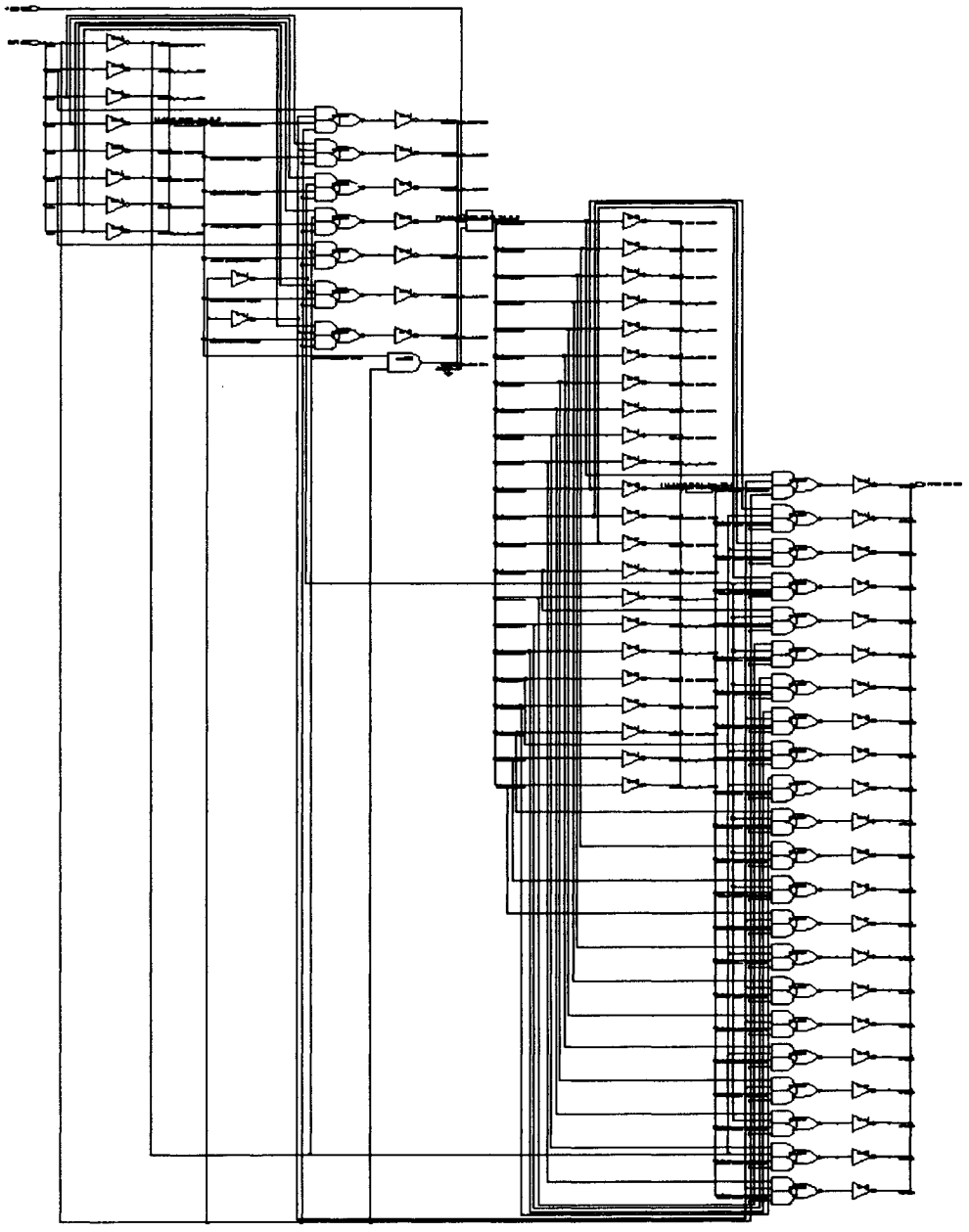


Figure.3.19 Schematic of the multiplier as generated in Synopsys

The circuit in Figure 3.19 consists of various sub blocks, the schematics of which are given in Appendix C. The verilog netlist is given in Appendix A – Code 12.

12 Bit adder

Figure 3.20 shows the schematic of the 12-bit adder. It is known that among the various adder architectures available, ripple carry adder is the best candidate for minimum area. It can be seen that the design synthesized in synopsys is also a ripple carry structure. Appendix A- Code 15 gives the verilog netlist of the schematic.

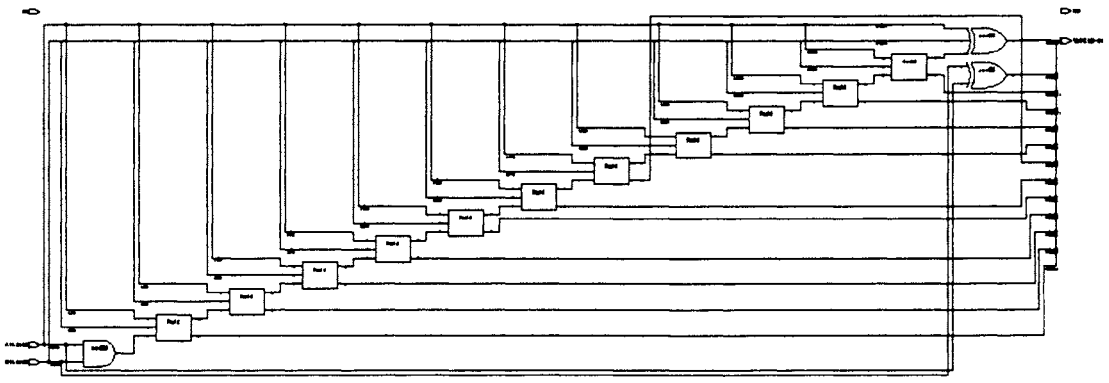


Figure.3.20 Schematic of the adder as generated in Synopsys

Binary-to-Thermometer Decoder

The final module to be designed is the binary to thermometer decoder. Similar to other modules, the netlist was first imported in synopsys and synthesized with minimum area constraint. Figure 3.21 shows the gate level implementation of the decoder. The verilog netlist is given in Appendix A – Code 20.

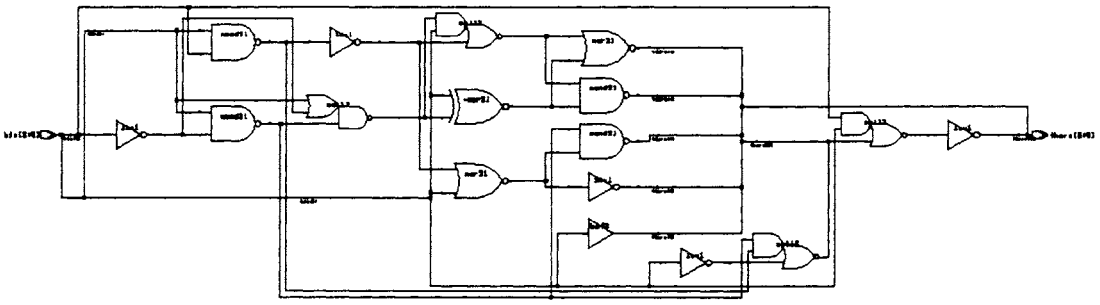


Figure.3.21 Schematic of the decoder as generated in Synopsys

3.2.2.3. Importing in Cadence and Testing

Following the synthesis of the modules in synopsys, the final step is to import these circuits in cadence and simulate. The different blocks were separately instantiated and the top level routing was done in cadence. Each of the blocks was separately tested first before simulating the entire digital portion. The test setup and simulation results of each block are given below.

Register (7-bit)

The circuit synthesized in synopsys was imported in cadence and simulated to verify the functionality. As in the case of VHDL test bench, different combinations of input and clock signal were given. The simulation outputs are shown in Figure 3.23. For the purpose of clarity, the input signal and output signals are shown in different plots (Figure 3.23a and Figure 3.23b). 'Data_in' refers to the input signal and 'Data_out' refers to the output signal.

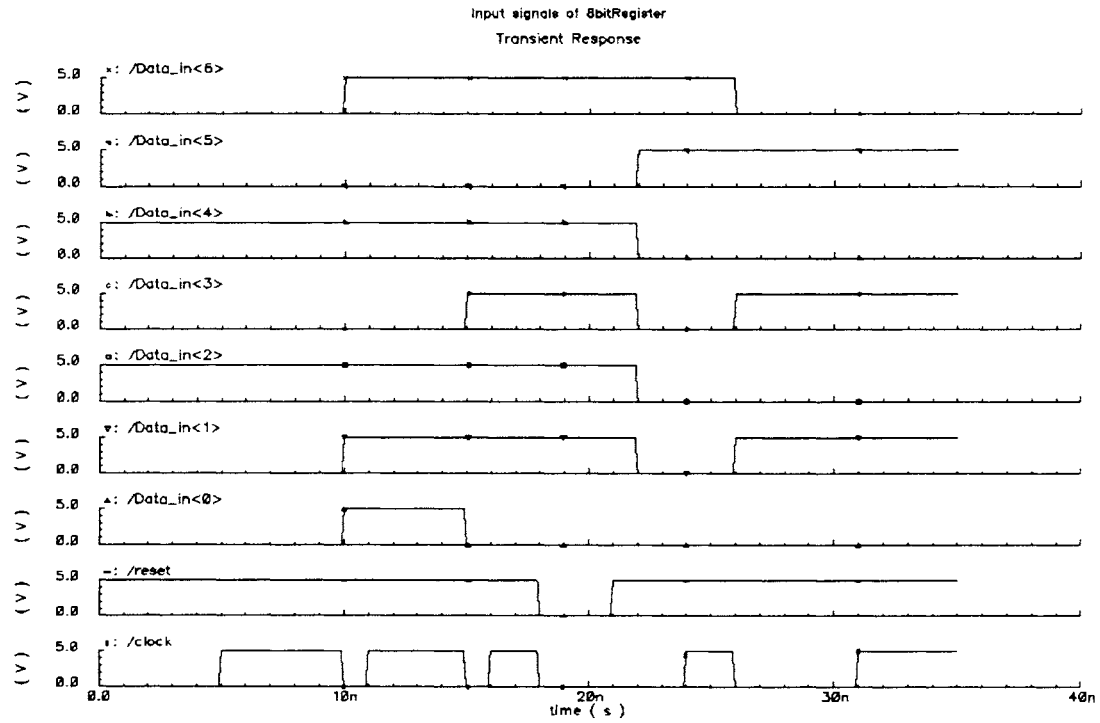


Figure.3.23a Input signal given to the register block in cadence

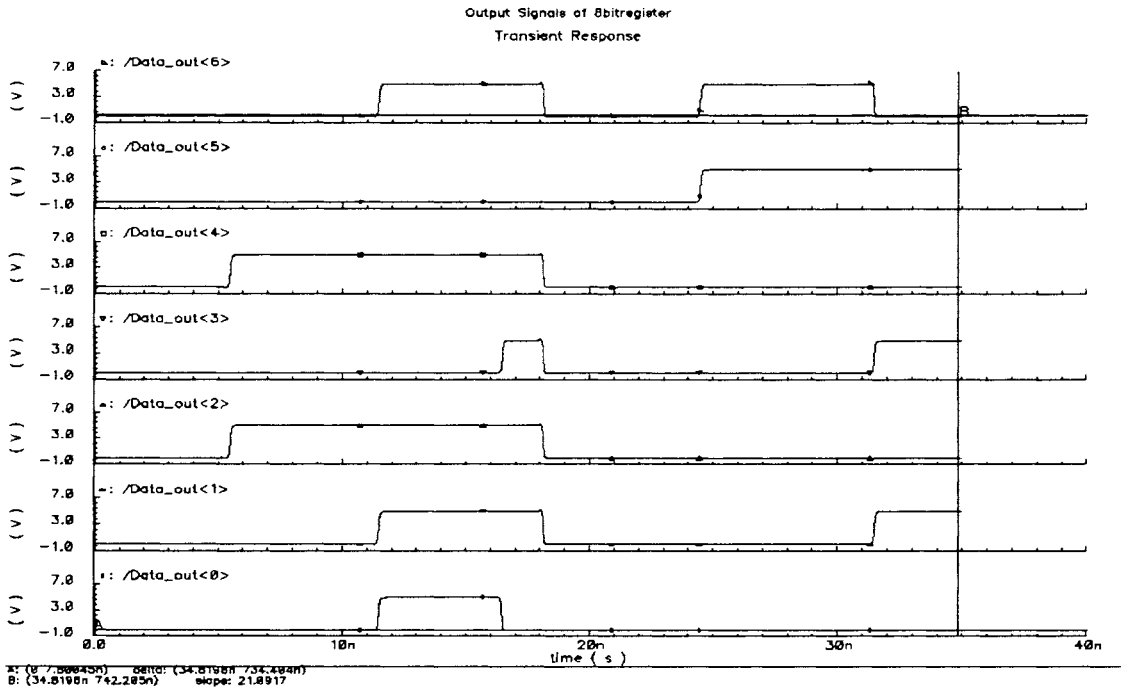


Figure.3.23b Output signals of the register unit

It can be seen from the plots that the input is transferred to the output at the rising edge of each clock cycle. It is also to be noted that even though the trigger signal is referred to as "clock", it need not necessarily have a constant duty cycle and time period.

Control Block

Figure 3.24 shows the test setup of the control module. The block on the left is just a set of voltage source providing the fuse values.

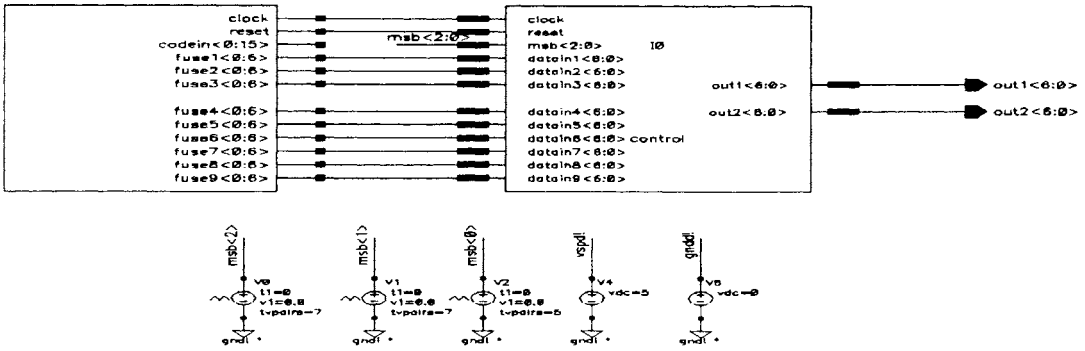


Figure.3.24 Test setup for the control block

Table 3.1 gives a set of sample fuse values and their binary representation that were used to test the circuit. For the set of fuse values given above, the expected outputs of the control block for different input codes (or equivalently different possible combinations of the 3 MSBs of the input code) are as given in Table 3.2.

Table.3.1 Sample fuse values used to verify the functionality of the control block

Fuse	**Binary Representation**	**Decimal Value**
1	0000000	0
2	0111111	63
3	1000000	-64
4	0000000	0
5	1110000	-16
6	1001000	-56
7	0111000	56
8	0010000	16
9	0000000	0

Table.3.2 Expected output for different input combinations

MSBs	**OUT1**	**OUT2**
000	0000000	0111111
001	0111111	1000000
010	1000000	0000000
011	0000000	1110000
100	1110000	1001000
101	1001000	0111000
110	0111000	0010000
111	0010000	0000000

The output obtained from cadence simulation for arbitrary combination of the 3MSBs of the input code is shown in Figure 3.25. The 3MSB signals are also included in the figure. Again for the purpose of clarity the two outputs (OUT1 and OUT2) are expanded and are split into different plots (Figure 3.25a and Figure 3.25b). The output plots match with the expected values given in Table 3.2.

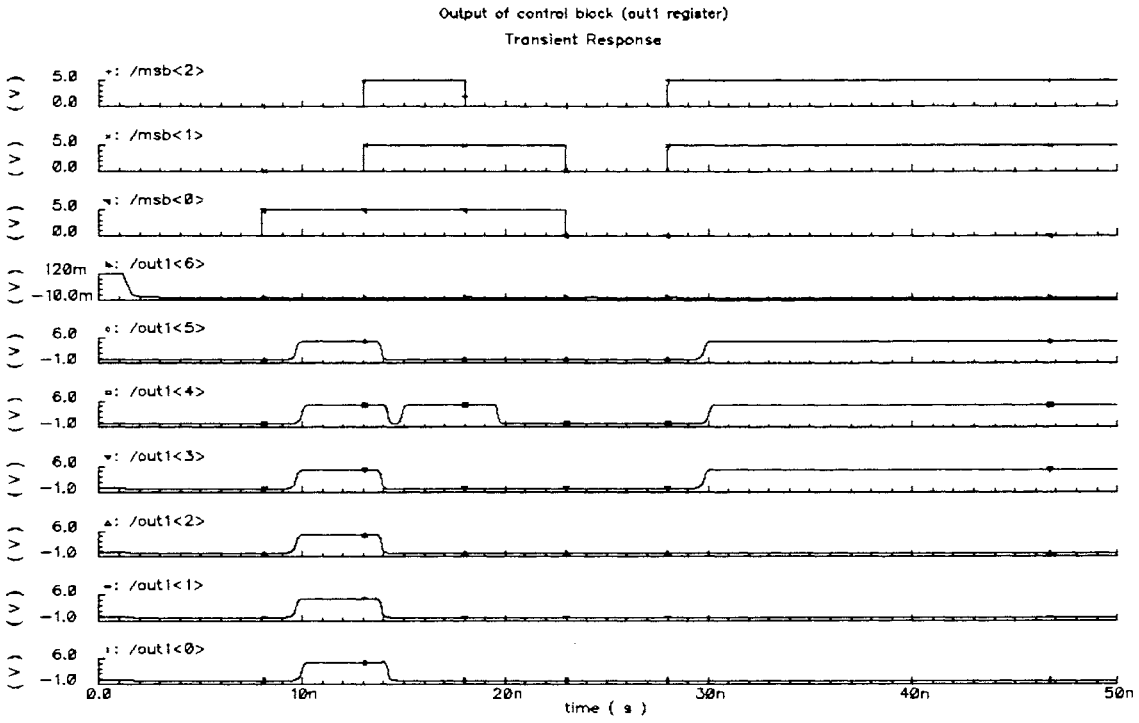


Figure.3.25a Expected value of out1 signal

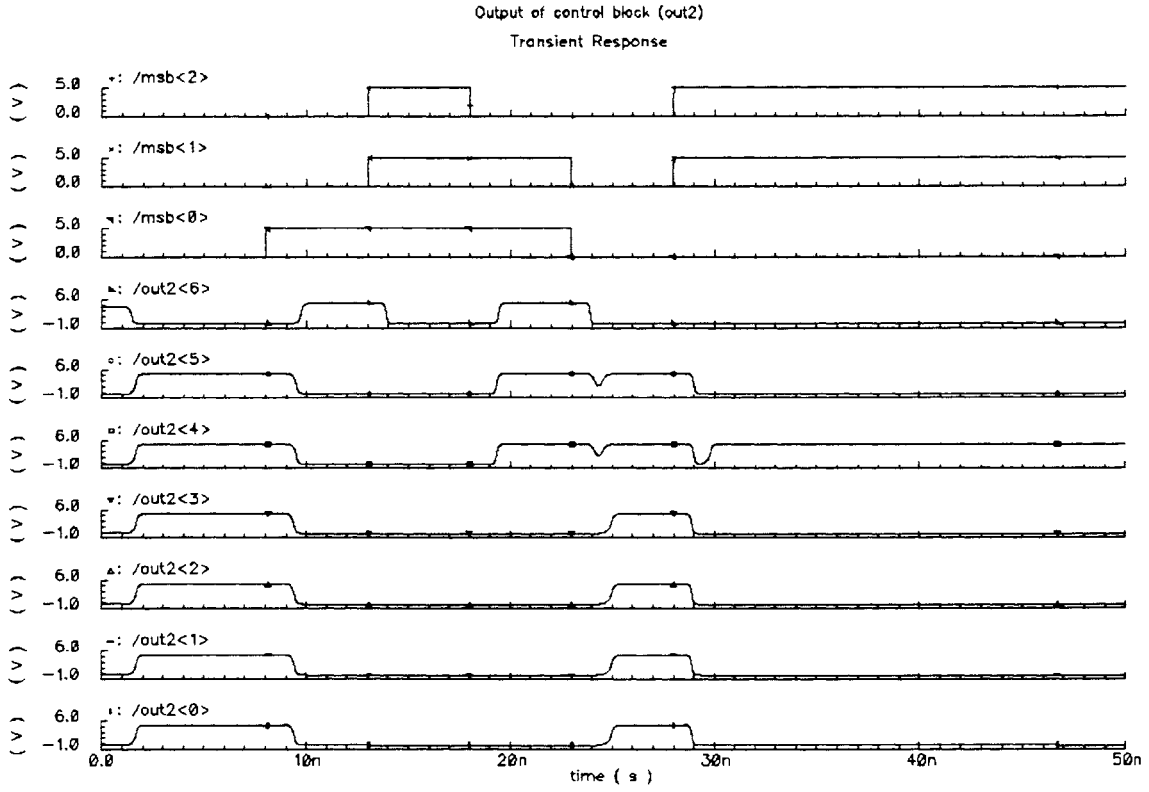


Figure.3.25b Expected value of out2 signal

Subtraction Block

The circuit was simulated with the same set of inputs as in the case of VHDL test bench. The values of input 'a' and 'b' and the expected output are given in Table 3.3. The output from spectre simulation is shown in Figure 3.27. It agrees with the expected value as given in Table 3.3. The various glitches that are seen are due to sharp rise and fall time and the small time period that was used in the simulation (10ns). Since this circuit is not optimized for speed (but for area), propagation delay between input and output is evident in the output signals. But at slower operating speeds, propagation delay will not be of much concern.

Table.3.3 Input-Output test cases for the subtractor

Time (in ns)	a	b	Difference	Binary Representation
0-10	0	0	0	0000,0000
10-20	20	10	10	0000,1010
20-30	10	20	-10	1111,0110
30-40	-25	-25	0	0000,0000
40-50	-25	35	-60	1100,0100
50-60	-64	-64	0	0000,0000
60-70	-64	63	-127	1000,0001

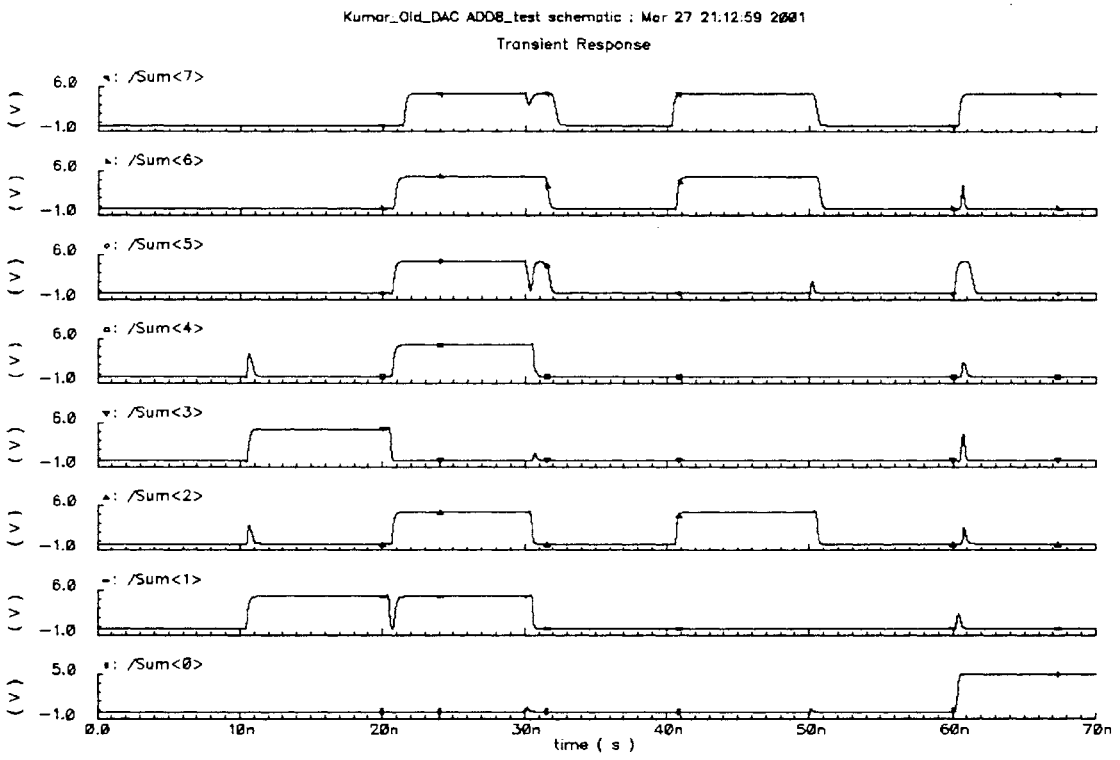


Figure.3.27 Output of the subtractor module

Multiplier

The input-output test vector that was used in the simulation is given in Table 3.4. The spectre outputs are divided into two figures (Figure 3.29a and 3.29b). Again it can be observed that the output from cadence is in agreement with that given in Table 3.4.

Table.3.4 Input-output samples for the multiplier

Time (in ns)	**a**	**b**	**Prod**	**Binary Representation**
0-10	0	0	0	00,0000,0000,0000,0000,0000
10-20	0	0	0	00,0000,0000,0000,0000,0000
20-30	31	2	62	00,0000,0000,0000,0011,1110
30-40	480	-16	-7680	11,1111,1110,0010,0000,0000
40-50	8191	15	122865	00,0001,1101,1111,1111,0001
50-60	7711	-16	-123376	11,1110,0001,1110,0001,0000

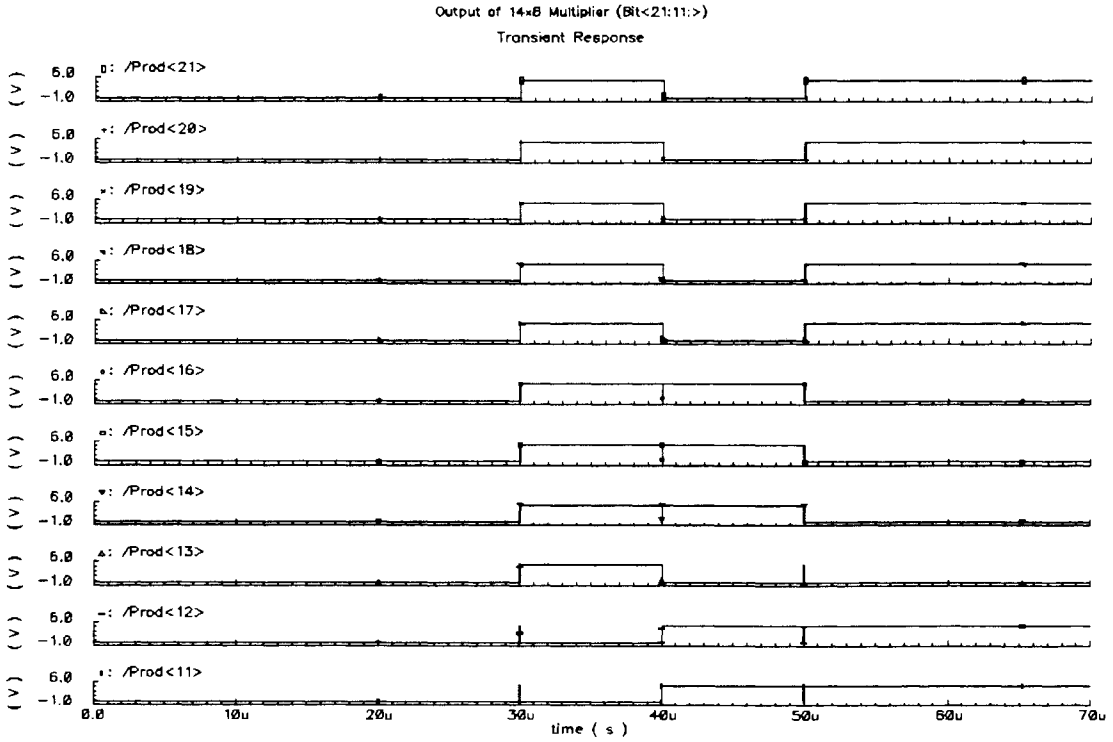


Figure.3.29a Output of the multiplier –Signals Prod<21> to Prod<11>

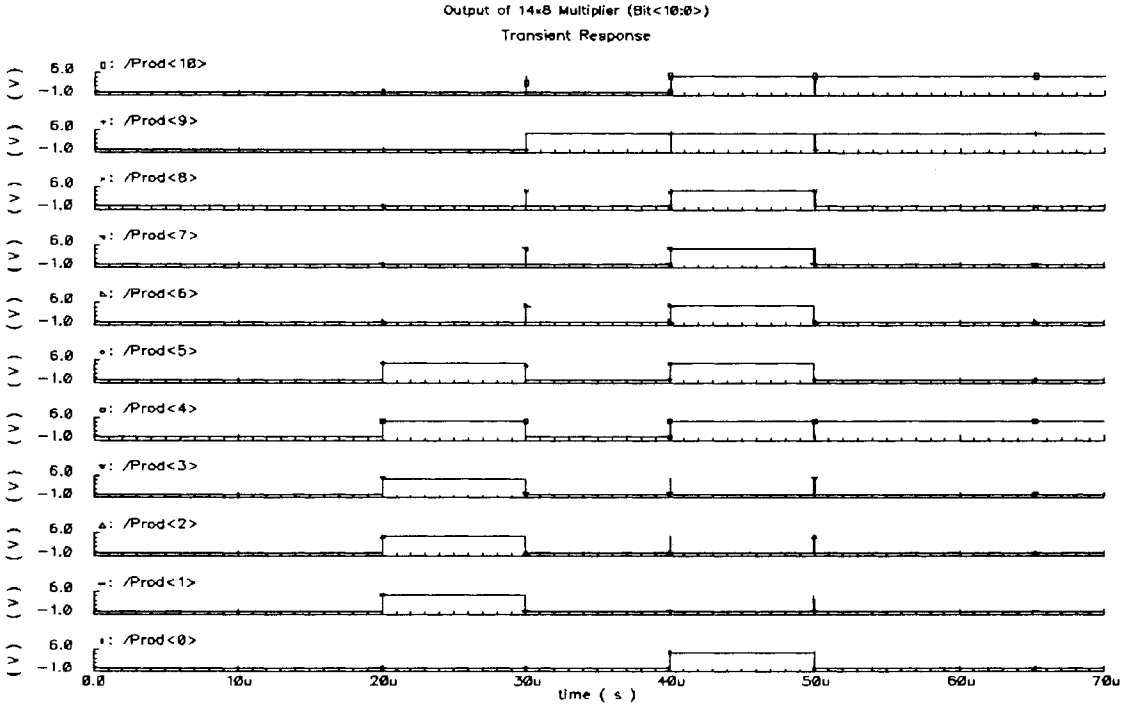


Figure.3.29b Output of the multiplier –Signals Prod<10> to Prod<0>

Adder Block

Table 3.5 shows the various inputs that were given to the adder block and the spectre outputs are shown in Figure 3.31. The output waveform agrees with the expected value reported in the table for different time slots.

Table.3.5 Sample test bench for the adder

Time (in ns)	**a**	**b**	**sum**	**Binary Representation**
0-10	0	0	0	0,0000,0000,0000
10-20	0	100	100	0,0000,0110,0100
20-30	632	100	732	0,0010,1101,1100
30-40	2047	2047	4094	0,1111,1111,1110
40-50	-2048	-2048	-4096	1,0000,0000,0000
50-60	-2048	2047	-1	1,1111,1111,1111
60-70	1365	682	2047	0,0111,1111,1111

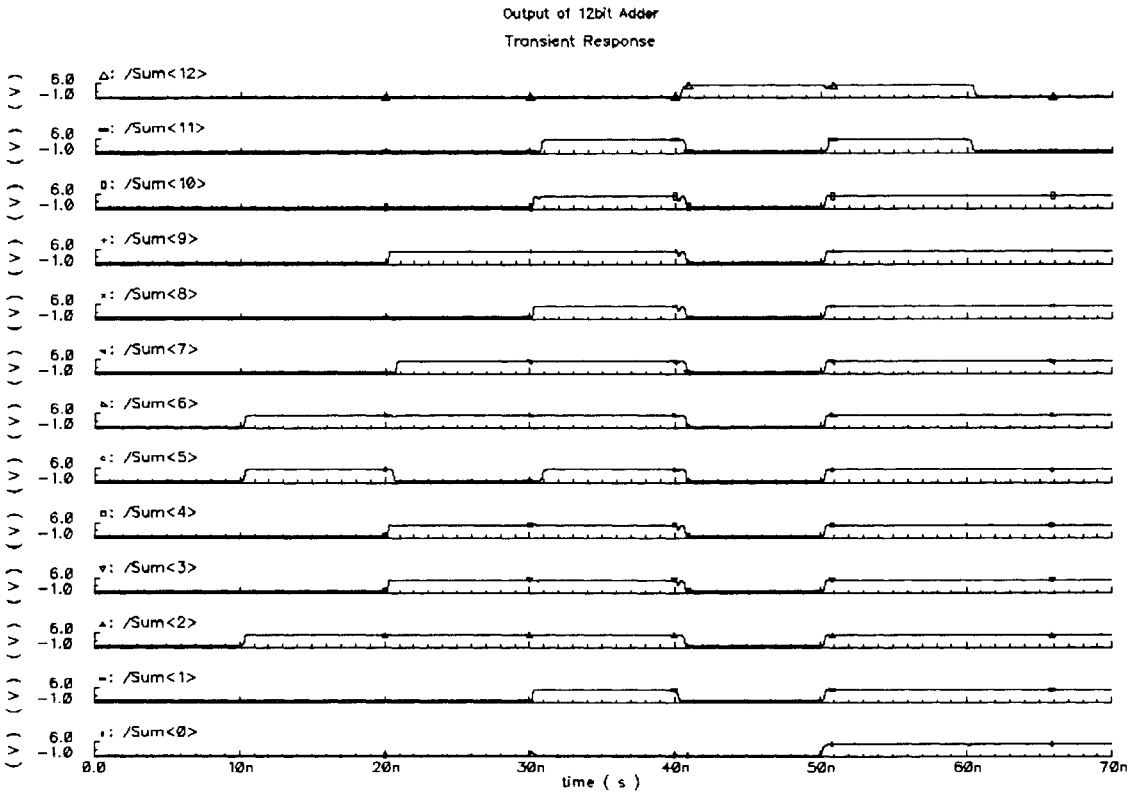


Figure.3.31 Output of the adder

Decoder

The final block that needs to be tested is the decoder. The output plot showing the 3-bit binary input and the 7-bit thermometer output is given in Figure 3.33.

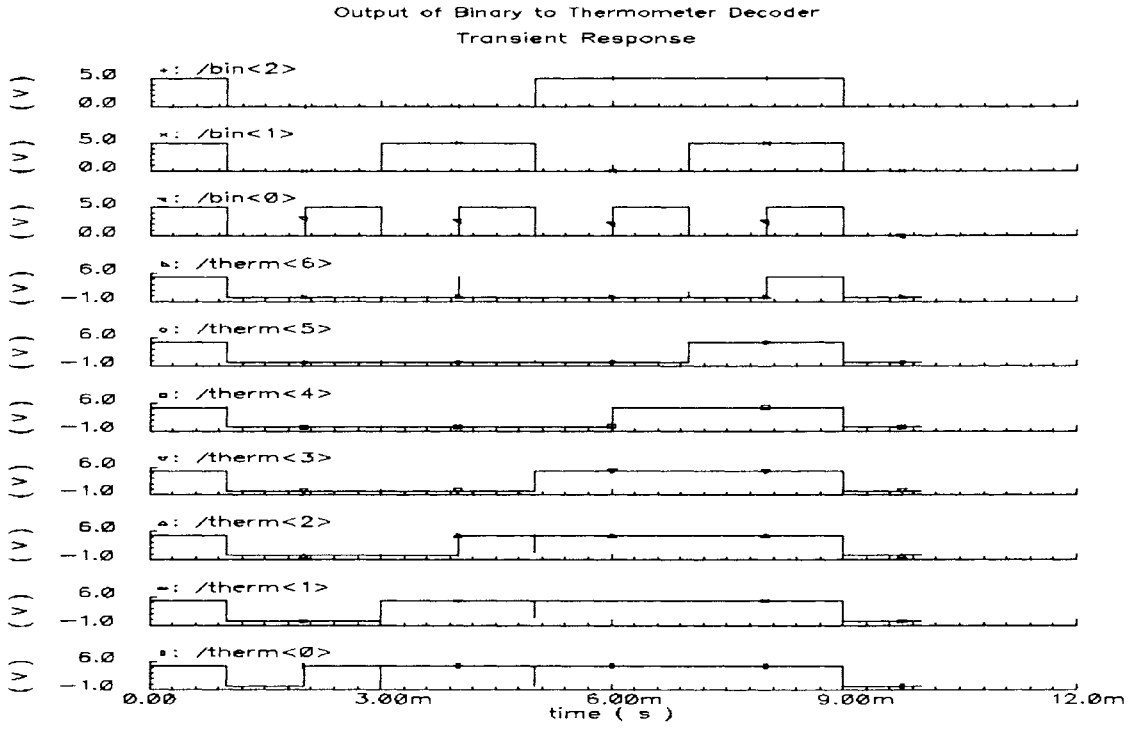


Figure.3.33 Decoder output

Full Block Simulation

Having confirmed the functionality of the individual modules, a full-block simulation was first performed in VHDL using all the entities described in the previous sections. Appendix A – Code 17 gives the code of the entire block. The different module instantiation and the corresponding input-output signals are described below. The actual VHDL syntax that is used in Code 17 is included in the description below and is italized.

ControlUnit: Instantiates control block which is responsible for loading data in registers and for deciding the correct pair of control values for any given input. The VHDL syntax used in the code is:

ControlUnit: Control(clock, reset, fuse1, fuse2, fuse3, fuse4, fuse5, fuse6, fuse7, fuse8, fuse9, Y1, Y2)

where Y1 and Y2 are the two output referring to the two end point INL values.

Subtractor: Instantiates the 8-bit subtractor block for finding the difference between the two Y values obtained from control unit. To convert 7-bit to 8-bit the MSB is repeated as shown below.

Y18<=Y1(6)&Y1; Y28<=Y2(6)&Y2;

Subtractor: ADD8(Y28, Y18, difference);

where 'difference' is the signal storing the subtractor output.

Multiplier: Instantiates MUL14x8 block. This takes as input the signal "difference" (output of subtractor) and a 14-bit input which is just the difference of the input code and control point x-axis value. Since the difference between the input code and control point x-axis is 13-bit wide, a '0' is added in the front, and the resulting 14-bit data is used as the second input to the multiplier.

Vectnew0 <= '0'&codein(12 downto 0);

Multiplier: MUL14x8(vectnewo, difference, product);

where 'product' is the 22-bit vector storing the multiplier output.

Adder: Instantiates the adder block. This block needs two input each 13-bit wide. One of the 13-bit signals is obtained from the multiplier (by dropping the 10 LSBs from multiplier output).

Newproduct <= product (21) & product (21 downto 0);

The other 13-bit vector is obtained from Y1. Three '0' are appended to the end, to multiply Y1 by a factor of '8'. This is required to convert the data from MainDAC LSBs to CalDAC LSBs. The MSB is repeated 3 times to convert the resulting 10-bit vector to 13-bit.

Vectnew<=Y1(6)&Y1(6)&Y1(6)&Y1&"000";

Adder: ADD12(vectnew, newproduct ,total);

where 'total' is the final 13-bit output obtained. After dropping the MSB, the remaining 12 LSBs are then used as input to the CalDAC.

Modelsim Simulation

A sample set of fuse values was chosen such that the INL profile of the uncalibrated DAC is as shown in Figure 3.34. In the actual device, the fuse values will however be written based on the result obtained from the tester. The input code was then swept through the entire range in steps of 128.

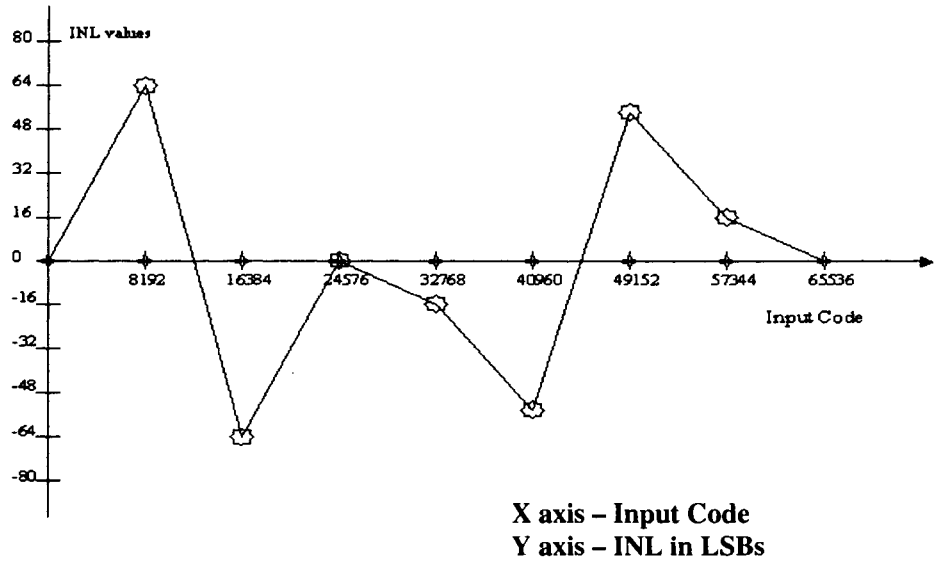


Figure.3.34 Sample INL profile of the uncalibrated DAC

Using the piecewise linear approximation, the expected CalDAC output for each region of input for the above given set of fuse values can then be formulated as shown in Table 3.6. The VHDL code (Appendix A - Code17) was simulated with the above fuse values. The output obtained from Modelsim simulation is shown in Figure 3.35. Only a small segment of the output is shown in the figure. The values of 'codein', 'codeout' and fuse values are included in the figure along with the various other internal signals.

Table 3.6 Expected CalDAC output for different region of inputs

Code In (*Cin*)	**Equation for the expected CalDAC output**
0-8192	$8\left[\frac{(63-0)}{8192}Cin+0\right]=\frac{63}{1024}Cin$
8192-16384	$8\left[\frac{-127}{8192}(Cin-8192)+63\right]=1520-\frac{127}{1024}Cin$
16384-24576	$8\left[\frac{64}{8192}(Cin-16384)-64\right]=\frac{64}{1024}Cin-1536$
24576-32768	$8\left[\frac{-16}{8192}(Cin-24576)-0\right]=384-\frac{16}{1024}Cin$
32768-40960	$8\left[\frac{-40}{8192}(Cin-32768)-16\right]=1152-\frac{40}{1024}Cin$
40960-49152	$8\left[\frac{112}{8192}(Cin-40960)-56\right]=\frac{112}{1024}Cin-4928$
49152-57344	$8\left[\frac{-40}{8192}(Cin-49152)+56\right]=2368-\frac{40}{1024}Cin$
57344-65536	$8\left[\frac{-16}{8192}(Cin-57344)+16\right]=1024-\frac{16}{1024}Cin$

The simulation output was then compared to the expected value obtained using the equations given in Table 3.6. Any difference observed is due to finite word length that has been used in the arithmetic and also due to the fact that the 10LSBs of the multiplier output is been neglected. As explained earlier, this can result in a maximum error of 1CalDAC LSB.

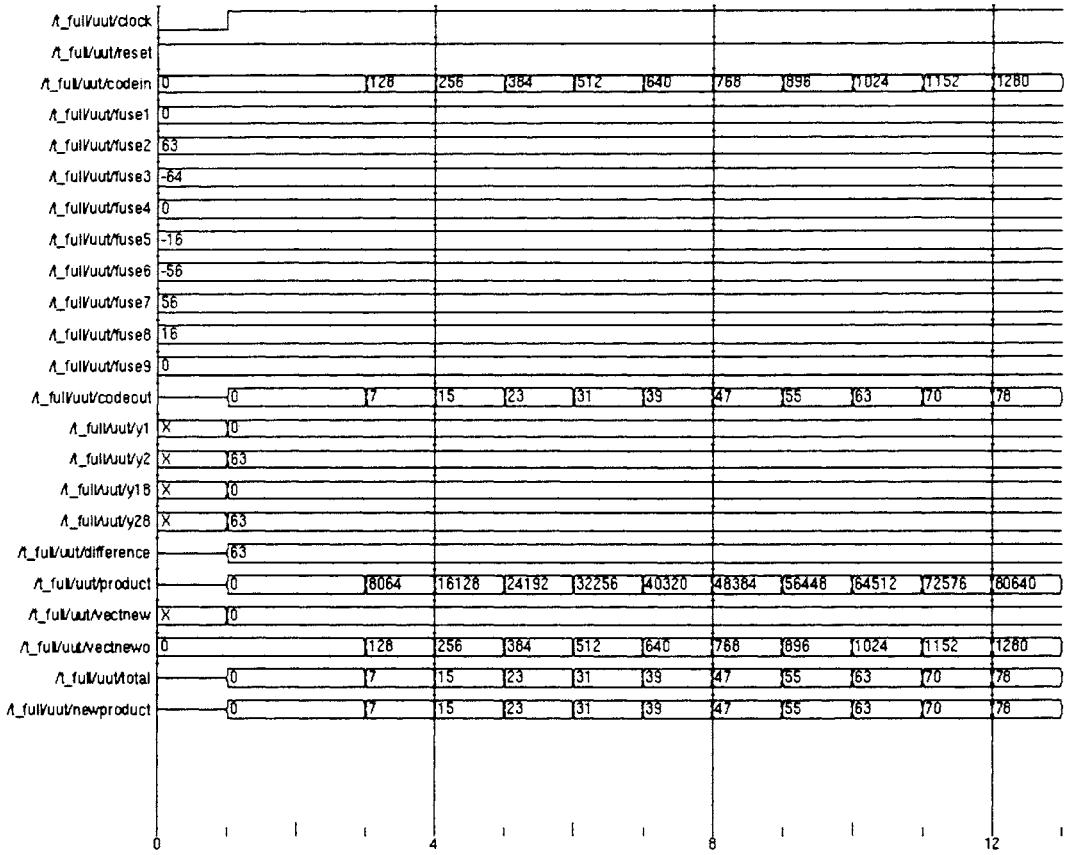


Figure.3.35 Top level simulation result in Modelsim

Some samples cases of input codes, expected output from equations and that obtained from VHDL simulation are reported in Table 3.7. The error in simulation results are within 1 LSB.

Table.3.7 Comparison of Modelsim results with the expected results

Code In	**From Equation (CalDAC LSBs)**	**From VHDL (CalDAC LSBs)**	**Code In**	**From Equation (CalDAC LSBs)**	**From VHDL (CalDAC LSBs)**
448	27.5625	27	27456	-45	-45
960	59.0625	59	34048	-178	-178
1536	94.5	94	39040	-373	-373
1984	122.0625	122	43392	-182	-182

Table.3.7 Continued

Code In	From Equation (CalDAC LSBs)	From VHDL (CalDAC LSBs)	Code In	From Equation (CalDAC LSBs)	From VHDL (CalDAC LSBs)
7488	460.6875	460	50048	413	413
8512	464.3125	464	57472	126	126
12032	27.75	27	59968	87	87
14016	-218.3125	-219	64000	24	24
20032	-284	-284			

Cadence Simulation

Once the functionality was confirmed in VHDL, the top-level schematic was generated in cadence by instantiating the various blocks as shown in Figure 3.36. The signals were routed in exactly the same manner as explained in VHDL code.

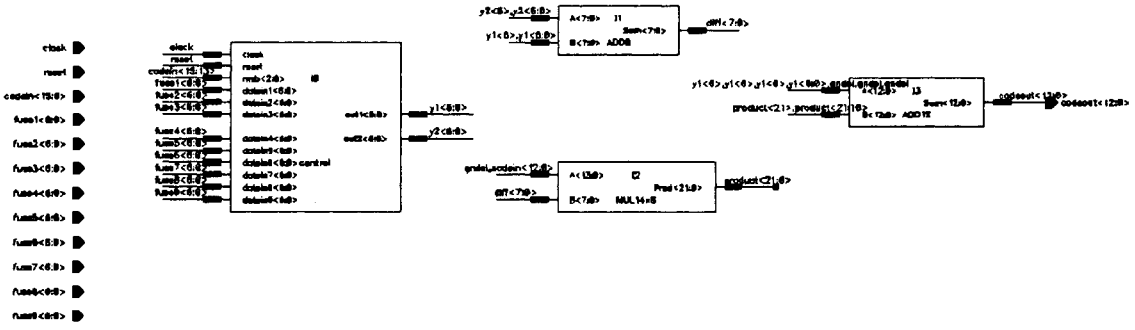


Figure.3.36 Top level schematic of the digital circuitry in Cadence

The fuse values were given through DC sources and inputs through VPULSE sources. The output from the digital block is in 2's compliment form and 13-bit wide. Before sending this data to the CalDAC, the MSB was dropped to convert the 13-bit vector to 12-bit. Also the MSB was inverted before thermometer coding, to shift the range as explained earlier. The signals were then given to the CalDAC (CalDAC design to be explained in Section 3.3).

A transient simulation was performed and the plot of the cadence output is given in Figure 3.37. Sharp glitches can be seen at different locations due to very short rise and fall time of the input signals and different propagation delay through the different modules in the circuit. These glitches will be smoothened once the output is passed through the analog circuitry. The different architectures that were considered for the analog modules and the corresponding schematics and simulation results are discussed in Section 3.3.

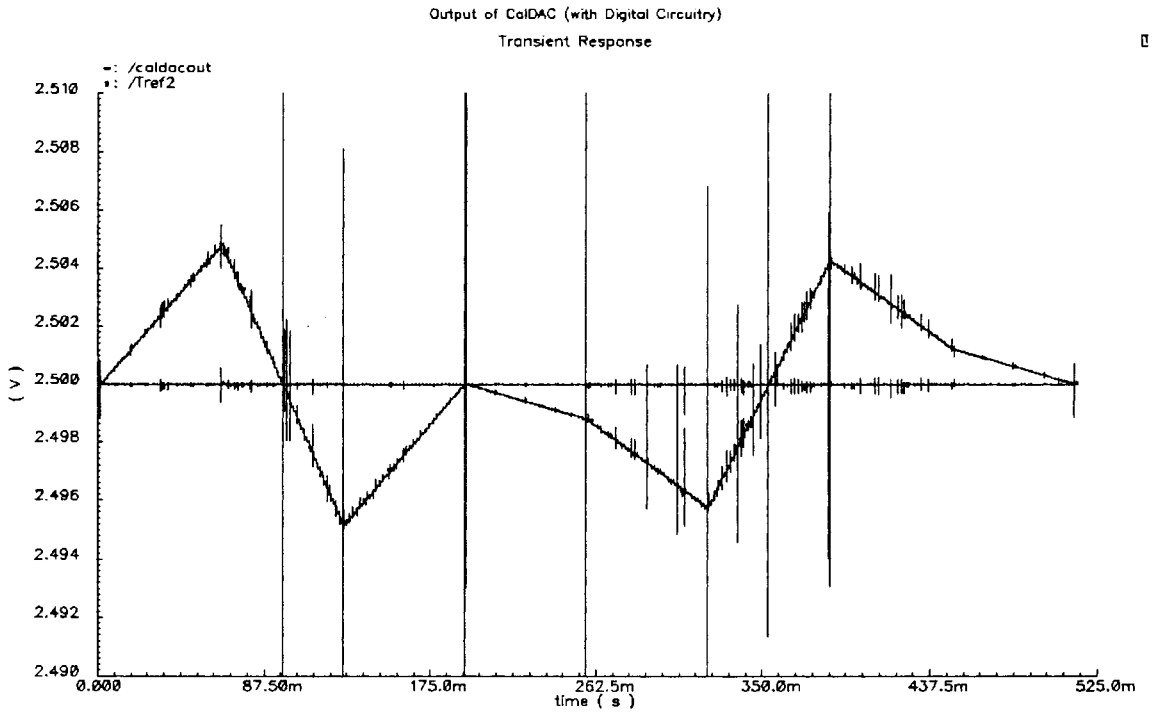


Figure.3.37 Output obtained in Cadence for the top level simulation

3.3. Analog Circuitry

This section deals with the design and implementation of the various analog modules that are required in the calibration circuit. The major blocks in the analog portion are:

- 12-bit Calibration DAC (CalDAC)
- Summing Circuit
- Reference Signal Generators

The architecture adopted for each block along with the associated design issues are described below.

3.3.1. Calibration DAC (CalDAC)

Based on the description given in Section 3.1, the required specifications of the CalDAC are:

Specifications: Resolution = 12-bit

Full scale reference voltage = 5/128 = 40mV (approx.)

Area = Minimum

CalDAC range = Vref,High to Vref, Low

Two different architecture were initially considered for this purpose

- R-2R with equal current sources
- Back DAC

Figure 3.38 shows the R-2R with equal current sources and Figure 3.39 shows the Back DAC architecture.

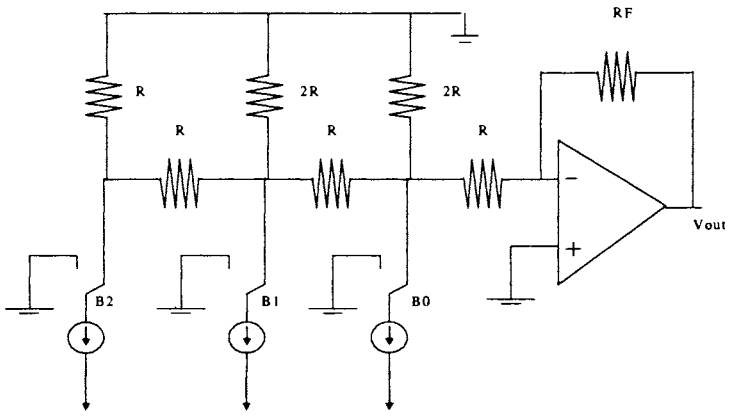


Figure.3.38 R-2R with equal current sources

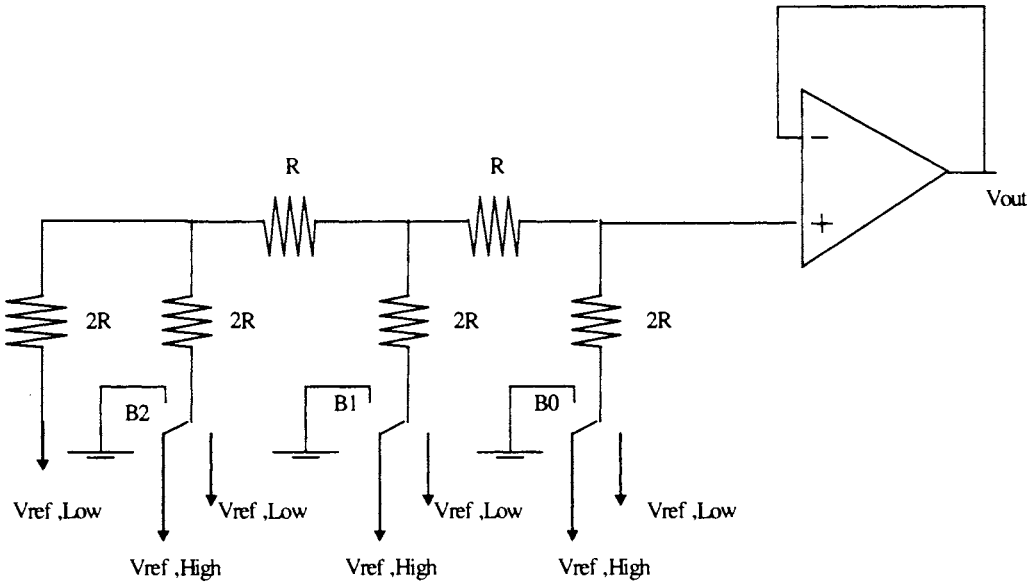


Figure.3.39 Back DAC architecture

The advantages and disadvantages of both the architectures were compared to choose the optimal structure for our requirement. Some of the main issues related to the two architectures are summarized below:

- As evident from the figures, both the architectures require resistors and switches to connect to 'Vref,High' or 'Vref,Low' reference, and buffer at the output to isolate the DAC from the output load.
- The R-2R with constant current source requires in addition, equal current sources. These current sources need to be cascoded to achieve high output impedance, and needs additional biasing circuitry.
- Choosing a high value of resistor (based on area and power dissipation trade-off) can considerably reduce the current consumption in BackDAC.
- The current in the case of R-2R with equal current sources cannot be reduced beyond a certain limit. This may result in current source transistor sizes that are either huge or impractical to layout.

Based on the above issues and also keeping into consideration the complexity of the CalDAC, the Back-DAC approach was adopted in this work.

Back-DAC

The variables to be decided in the design of Back-DAC are:

- ✓ Resistance value
- ✓ Switch sizes
- ✓ Switching voltage

Ideally we need the switches to have zero resistance. However in practice, switches have a finite resistance depending on the W/L ratio and the gate-source switching voltage (Vgs) as given by

$$R_s = \frac{L}{\mu C_{ox} W (V_{gs} - V_t)} \tag{3.3}$$

It can be seen from the above equation that:

a) Higher the W/L ratio of the transistors, higher the area but lesser the resistance.

b) The more Vgs of the transistors, lesser the resistance, but Vgs in turn is limited by the source voltage.

c) The lesser W/L of transistors, more the 'ON resistance' of the switch and 'R' required in the Back DAC design is high to ensure monotonicity.

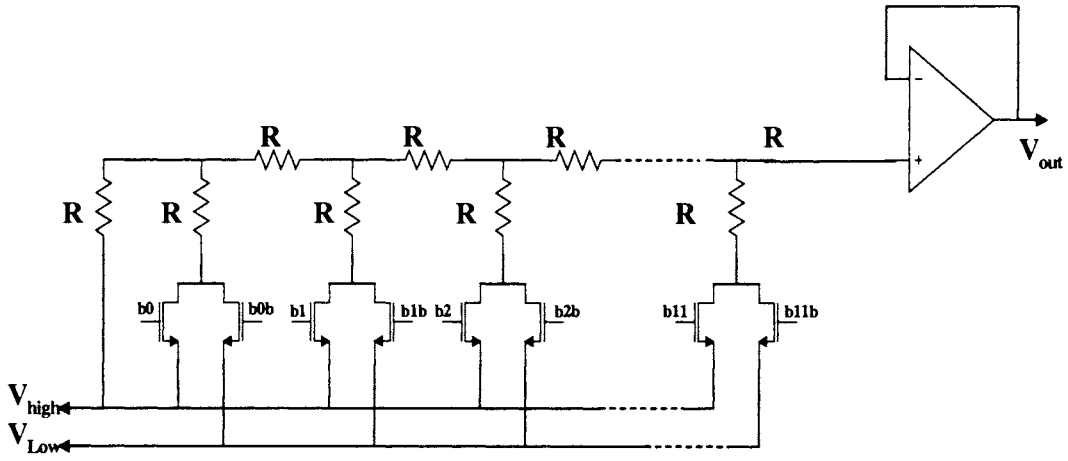
d) More 'R' results in less power dissipation (due to less current), but results in more area.

Thus we see that all the above parameters are inter-related and a compromise has to be made in terms of resistor area, switch area, power dissipation and linearity.

Simulation and Results

To understand the effect of each of the above parameter on the performance, a set of simulations were performed. A 12-bit full binary R2R Back DAC as shown in Figure 3.40 was first considered. The length of all transistor (that are used to realize the switches) were fixed at 0.5u and the widths were varied in binary fashion, with the width of the LSB transistor being 'W'.

Table 3.8 gives the voltage drop across MSB and LSB switches (an indication of the switch resistance) when W, R and Vgs were changed. Ideally we would want the drop across the switches to be zero. But due to finite resistance of the switches, a voltage divider is created between the resistor ladder and the switch. The ratio of switch resistance to ladder resistance should be very small to avoid any inaccuracy in output.



b0 = b1 =....= b11 = ON
b0b = b1b =....= b11b = OFF

Length of transistors = 0.5u
Transistor width varied binarily
Width of LSB = 'W'

Figure.3.40 Test circuit to analyze the effect of switch resistance

Table 3.8 Voltage drop across the switches

Width (W) (in um)	**R**	**Vgs (V)**	**Vds,MSB**	**Vds,LSB**	**Current**
50	1k	5	339.8n	413.0u	13.24u
100	1k	5	174.0n	223.1u	13.28u
500	1k	5	39.84n	53.57u	13.32u
50	10k	5	31.10n	41.94u	1.332u
100	10k	5	16.59n	22.49u	1.332u
500	10k	5	3.946n	5.368u	1.332u
50	10k	2	52.44n	70.18u	1.332u

Observations:

i. As seen from the table, for a given value of 'R', as W increases, the switch resistance decreases, and the drop across the switch decreases.

ii. For a fixed value of switch width (W) as resistance is increased, the drop across the switch decreases. Also the current drawn decreases as resistance are increased.

iii. As Vgs decreases, the resistance of the switch increases and drops across the switch increases.

With an above understanding of the effect of various parameters on switch resistance, transient simulations were performed to analyze the linearity of the converter; results of which are summarized below:

Run 1: A 12-bit segmented CalDAC was implemented, with the 3MSBs thermometer coded and the 9 LSBs binary weighted. The design parameters were chosen as follows:

Width of LSB switch=250u; Length of switches=0.5u; R=10k; Vgate = 5V; Vref,High = 40mV (approx. 1/128 of 5V); Vref,low =0; Transient pulse width = 100u; Trise of switch input = Tfall of switch input = 1us; Total transient simulation time = 409.6ms (for 12-bit)

The above parameters were not rigorously estimated for this initial run. Figure 3.41 shows a small segment of the output plot and Figure 3.42 gives the INL and DNL of the DAC output. The matlab code that is used to calculate the INL and DNL of the converter is given in Appendix B.

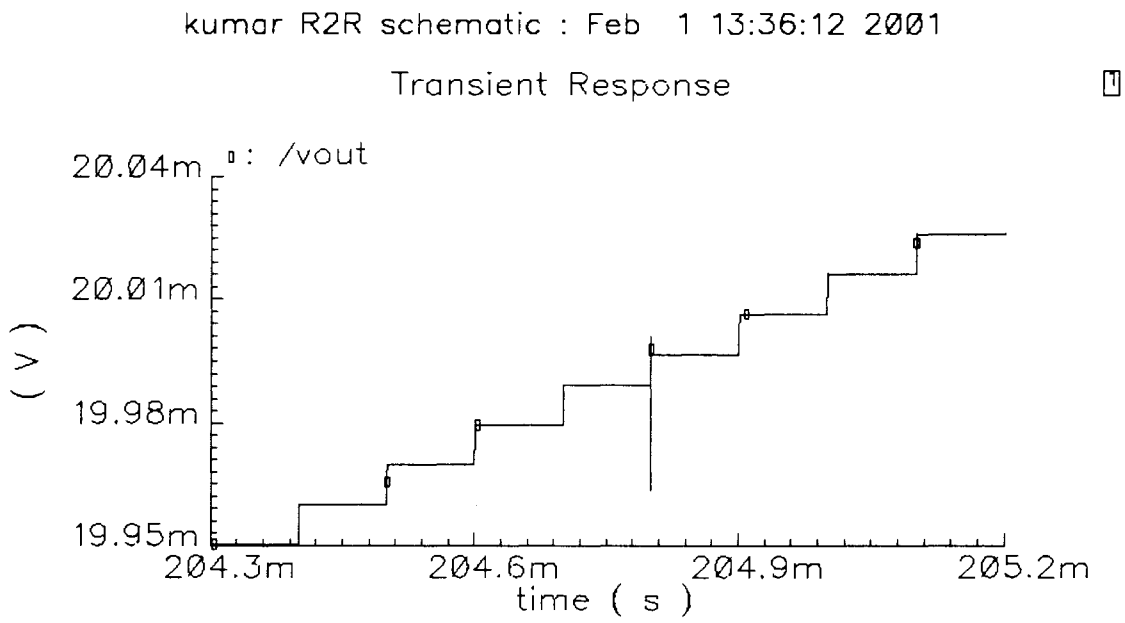


Figure.3.41 Zoomed plot of a segment of the output

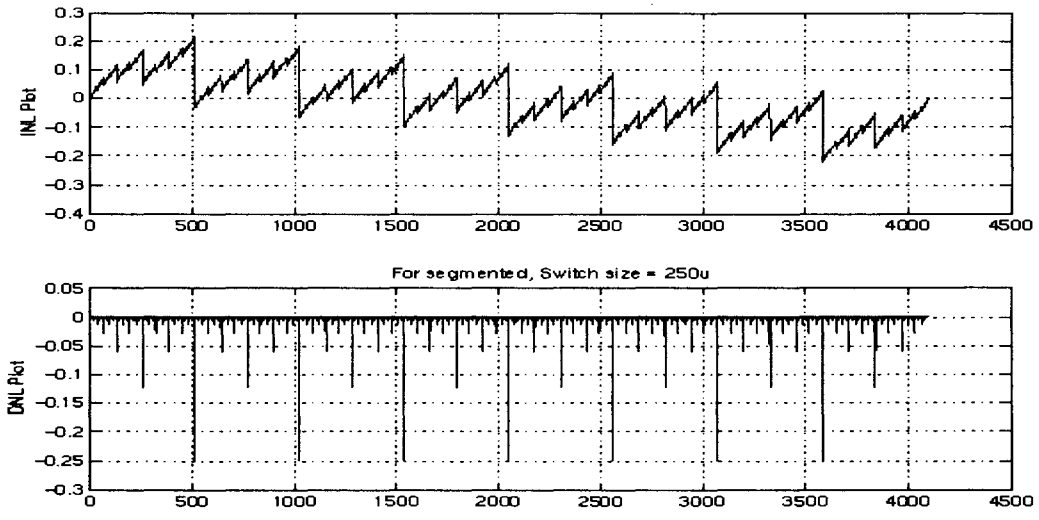


Figure.3.42 INL and DNL of the converter for Run1 *(Y axis corresponds to CalDAC LSBs)*

It can be seen from the output plot that the INL varies from +0.25 LSB to -0.25 LSB confirming 12-bit performance. However, approximate area estimation needs to be done to analyze the feasibility of the design and to aid in deciding the parameter values. With the resistance of Poly-I equal to 32 ohms for a segment of size W/L=20/100 in the given process, and the size of the LSB switch approximately equal to 250u/0.5u; the total area needed for the design with the above chosen parameter is very high. After considering issues like minimum dimension of the resistor required to ensure proper matching, interconnect area, it was decided to keep the total resistance for the entire BackDAC design to be less than 100K. From Figure 3.40, it can be seen that the total resistance required realizing the DAC structure is 43R, where R is the unit resistance value. To satisfy the above area constraint, the unit resistance then needs to be limited to approximately 2K.

Run 2: For this value of resistance, different widths (W) of switches were simulated again. Also as the output of the CalDAC swings from 0 to 40mV, it is difficult to design an output buffer with a similar output swing. Hence the Vref,high and Vref,low were selected so that it is spaced around a common mode voltage (of 1.25V). This has an associated disadvantage that the Vgs of the switches reduces (since gate voltage cannot go beyond 5V) and a higher W of switches may be required to get the desired linearity, resulting in increased area.

Linearity simulation was performed again with the following conditions:

Segmented architecture; Switches scaled in a binary fashion with $W_{LSB} = 0.5u$; $W_{MSB}=512*W_{LSB}$;

L=0.5u; R=2k; Vgate = 5V;

Vcommon-mode = 1.25; Vgs = 5 - 1.25 = 3.75;

Vref,High = 1.29V; Vref,low =1.25;

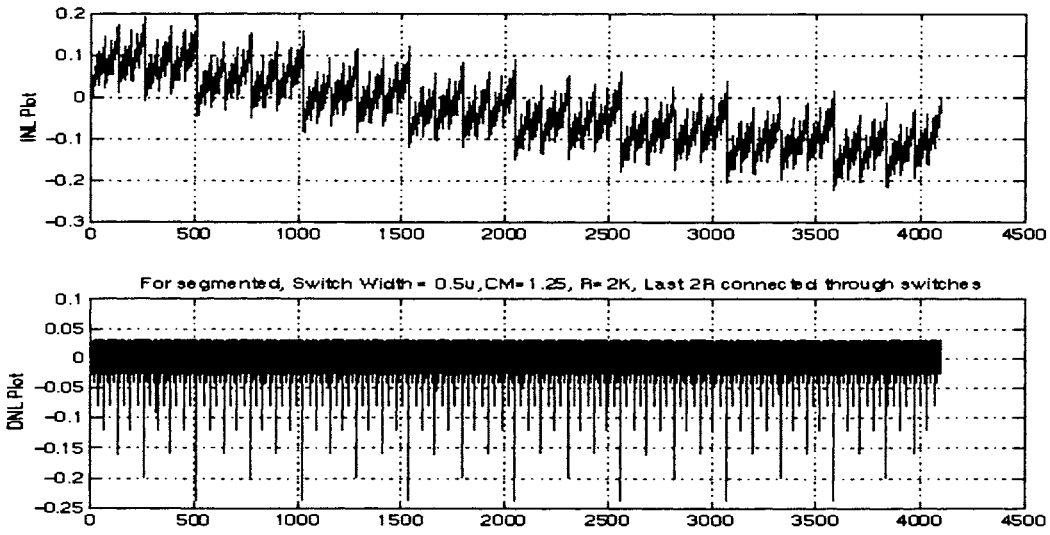


Figure.3.43 INL and DNL of the converter for Run2*(Y axis corresponds to CalDAC LSBs)*

Although the results obtained in Run 2 are satisfactory and meets the specifications, after few more trials, a more optimized structure with the following parameters was finally chosen for the design.

CalDAC parameters

- ✓ R=1k; Total Resistance = 43K.
- ✓ Total switch area = 512*8*2*width*length (the factor '8' corresponds to 7 thermometer coded MSB and 1 extra to consider all the 9 LSB bits; 2 corresponds to 2 switches per bit).
- ✓ Total area = $2048u^2$.
- ✓ Vcommon-mode=2.5V

3.3.2. Summing Circuit

With a preliminary design of the CalDAC available at hand, the next module to be designed is the summing circuit. Two different schemes were considered for adding the CalDAC output to the

MainDAC output. The first scheme involved multiple amplifiers and was finally dropped. A short description of the scheme is given below.

Scheme 1:

Refer to Figure 3.44 for the block diagram of Scheme 1. The structure is just a traditional summing circuit.

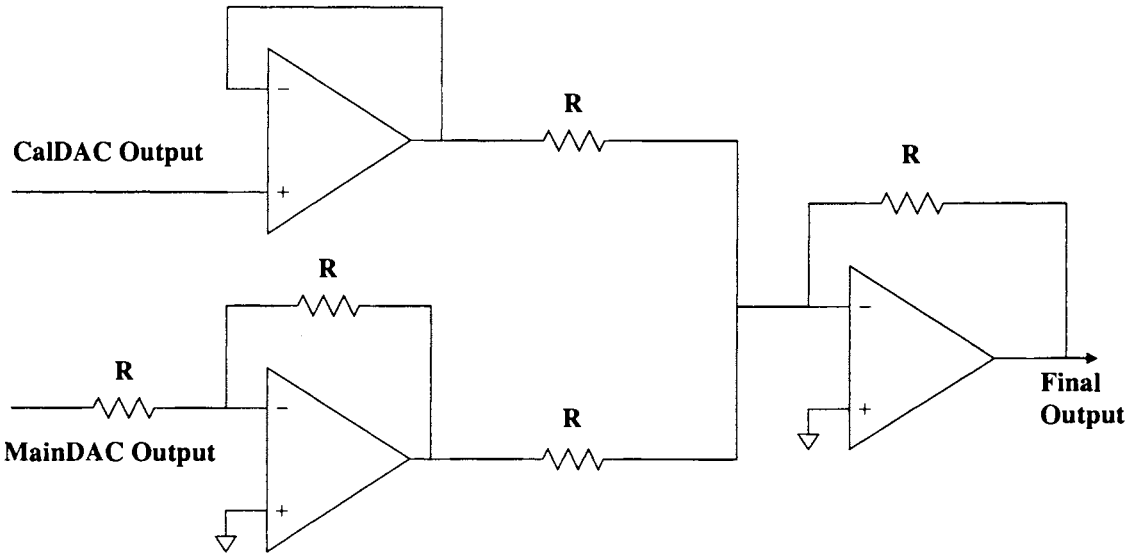


Figure.3.44 One proposed scheme to add CalDAC output to MainDAC output

It requires 3 opamps as shown in the figure. In addition it also requires the generation of various reference signals and extra resistors to realize the summing operation. The above circuit was initially designed and the entire block (including the R2R CalDAC, Digital portion, summing circuit) was simulated. Results confirmed the functionality of the entire circuitry. However, the main bottleneck with this scheme was the area and power consumption. The analog portion of the entire scheme was approximately 0.8 times the size of the original MainDAC. Hence an alternate scheme was decided upon.

Scheme 2:

A new structure based on the Wolfson architecture was considered. A simplified diagram of the scheme is shown in Figure 3.45. Unlike the previous scheme, in this method the calibration signal is added in the form of a current signal rather than a voltage signal.

The voltage at the output node is given by,

$$Vout = (1 + \frac{R1}{R2 + R3}) * Vin - I * R3 * \frac{R1}{R2 + R3} \tag{3.4}$$

where '*Vin*' is the MainDAC signal and '*I*' is the CalDAC signal. If the resistors are chosen such that R1=R2+R3, then

$$Vout = 2 * Vin - I * R3 \tag{3.5}$$

Note: In the design of the MainDAC, a dummy resistor string of the same value as the main resistor string is connected in series with the main string to satisfy certain matching requirements. This results in division of supply voltage (5V) by a factor of '2' and hence a 'gain of 2' stage is required at the output. In the original design, the interpolating amplifier at the output is used to solve the dual purpose of interpolation and 'gain by 2'. From (3.5) it can be seen that by using the structure shown in Figure 3.45, we preserve the 'gain by 2' operation. If the already existing interpolating amplifier is used to realize the proposed scheme, then it eliminates the need of any additional amplifiers thereby reducing the area requirement considerably.

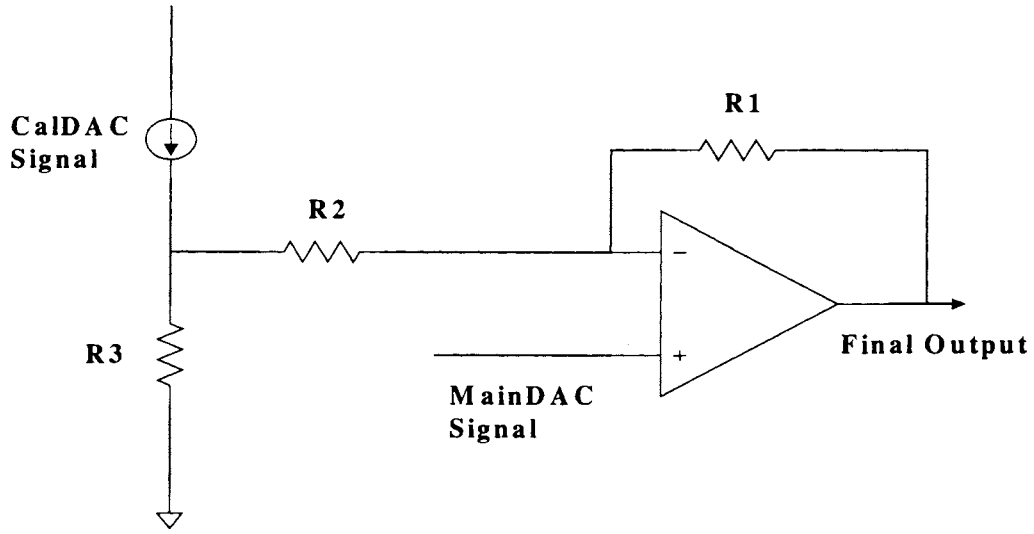


Figure.3.45 Alternate scheme to add CalDAC output to MainDAC output

However, this scheme has two potential problems:

Temperature variation of the resistance: The performance of the circuit is sensitive to resistor variation due to temperature change. The gain of '2' is not affected since it depends on the ratio of resistance, but if the value of R3 changes with temperature, the calibration value also changes.

Subtraction of current: It is difficult to subtract and add current signals using one simple circuit.

The first problem could be solved by making sure that the current varies with temperature in the same manner as the resistance. In other words, if both I and R3 changes such that the product remains same, then the circuit is temperature insensitive. A simple scheme as shown in Figure 3.46 could be used to realize this. Resistance R4 is of the same value as R3. The CalDAC output is converted to current using the feedback circuitry (the opamp required is not complicated). The current is then mirrored and is fed into the node between R2 and R3. Opamp O1 fixes one end of the resistor R4 at the CalDAC output voltage while the other end of the resistors is connected to some constant offset

voltage (explained later). The feedback circuitry biases the current mirror such that an appropriate current flows through the resistor. Any variation in temperature will result in similar variation in R3 and R4. The change in R4 results in proportional change in current, such that I*R3=I*R4=constant.

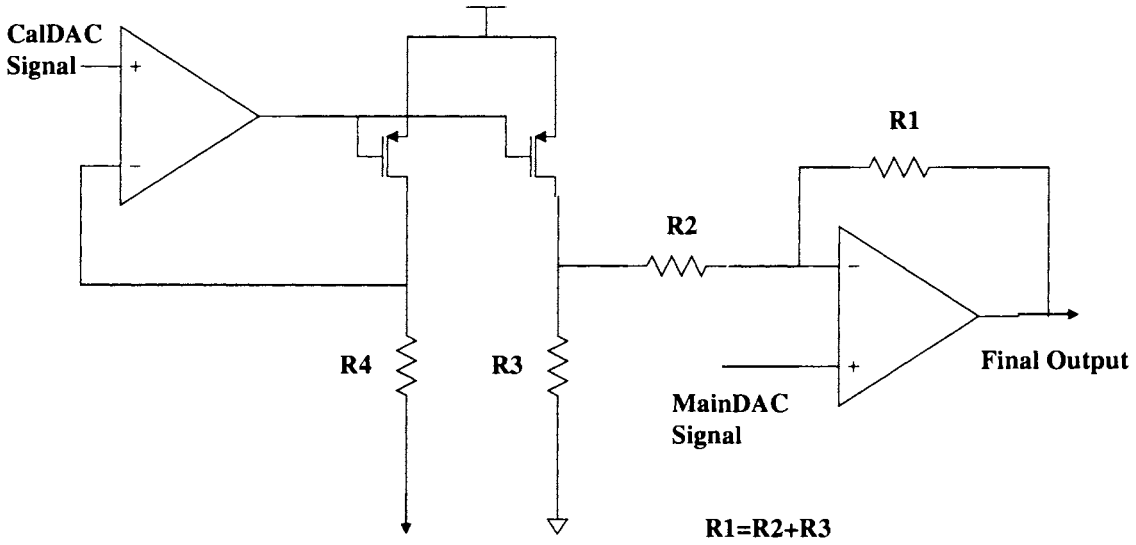


Figure.3.46 Scheme to make the circuit temperature insensitive

The second issue is related to addition and subtraction of current. Depending on the value of CalDAC signal, I in (3.5) could be positive or negative resulting in addition or subtraction of current. The circuit shown in Figure 3.46 can just be used to add current. To enable both addition and subtraction of current signal, two possible solutions were considered

Solution 1: Using a similar structure consisting of NMOS current source that operates when Vcalout is negative could possibly solve the issue. The proposed schematic is shown in Figure 3.47. When current needs to be injected, the circuit consisting of O1, MP1 and MP2 is operational and when current needs to be subtracted (or removed) O2, MN1 and MN2 is operational. Though this solution seems fine conceptually, it requires that MP1 and MN1 are properly matched and it is also necessary

to ensure that only one of the two circuits operate at any given time (depending on whether Vcalout is positive or negative). This raises various timing and switching issues thereby complicating the entire architecture.

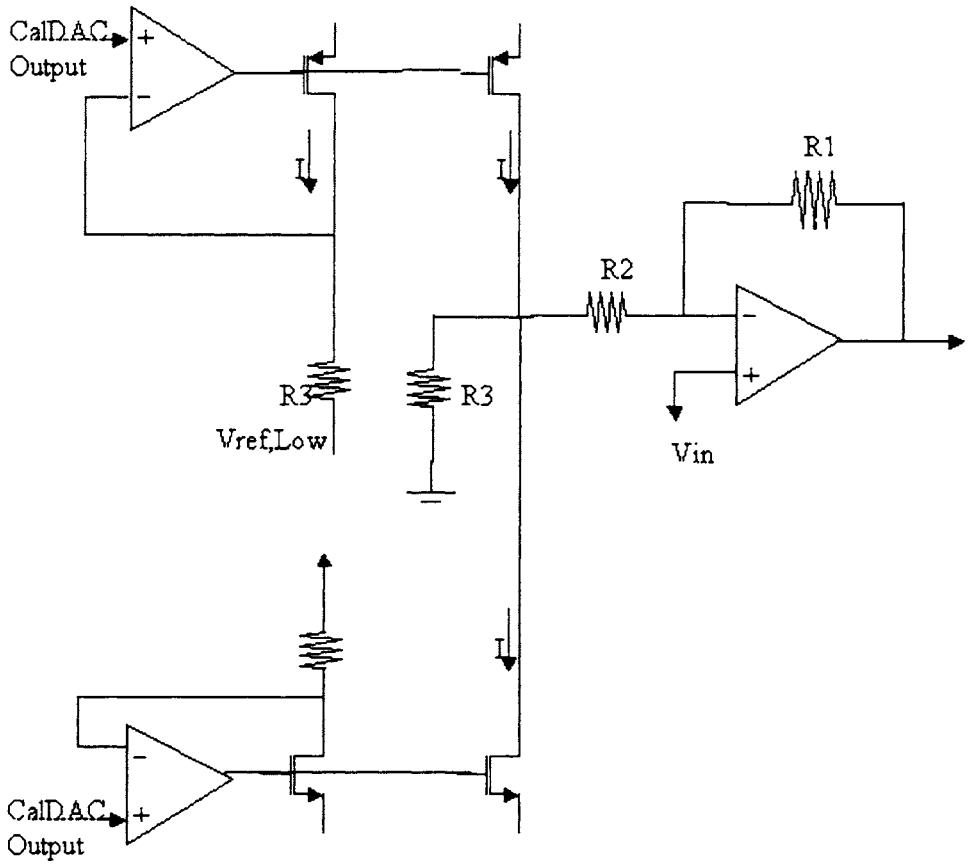


Figure.3.47 One proposed scheme to add and subtract current

Solution 2: The entire calibration signal range can be shifted such that the calibration voltage is always negative and current needs to be injected in all cases. From the description of CalDAC design, we know that

when required calibration voltage is '0', the CalDAC output is 2.5V

when required calibration voltage is positive, CalDAC output is between Vref,Low to 2.5

when required calibration voltage is negative, CalDAC output is between 2.5 to Vref,High

The lower end of resistor R4 in Figure 3.47 can be tied to Vref,Low instead of 2.5V. This ensures that for any voltage from CalDAC (ranging from Vref,Low to Vref,High), the current through the resistor flows in one direction. However this causes an offset of magnitude (2.5 - Vref,Low), which is equal to half the CalDAC full-scale range. Since the output amplifier has a gain of '2', to compensate for the offset an equivalent voltage of (2.5 – Vref,Low)/2 is to be given at the positive terminal of the output amplifier. This offset voltage can be combined with the signal generated from the resistor string of the MainDAC itself.

In the following set of simulations, the offset required is not considered. The circuit is first designed for required linearity, area and power dissipation. The offset can finally be calculated and added to the positive terminal.

The entire summing structure then consists of (from Figure 3.46):

Opamp O1 – Needs to be high gain so that it can set the voltage at the node connecting R4 to be exactly the same as Vcalout. No output stage is required as the output is connected to the gate of the current mirror and current drawn is zero.

Opamp O2 – This one is responsible for 'gain of 2' and also for calibration signal addition. The existing interpolating amplifier in the MainDAC can be used for this purpose. Hence this does not add any extra circuitry.

Current Mirror/Cascode – The sizes of the transistors needs to be designed such that they are in proper regions of operation over the entire range.

R1, R2 and R3(or R4)– The value of these resistors needs to be decided. In the existing design of the MainDAC, R1 was fixed at 44.2975K for reasons of stability. The same value is used in this design also. The other resistors then need to be chosen such that R1=R2+R3.

Deciding the value of R3 (or equivalently R4)

Resistance R4 determines the variation in current with variation in calibration voltage. A low value of R4 implies a large variation in current over full scale. For instance, with a CalDAC full scale range of 40mV, if R4 is chosen as 250 ohms, the variation in current is nearly 0 to 160uA, with I_{LSB}=40nA. Not only does this choice of resistor result in a high value of full-scale current, but also results in current mirror transistors moving out of saturation region of operation for low values of current.

On the other hand a very high value of R4 (or R3) results in smaller value of R2 (since their sum is fixed, equal to R1). This implies that the voltage at the node connecting R2 and R3 varies considerably with MainDAC signal. The voltage fluctuation at node connecting R2 and R3 is determined by the voltage divider ratio of R2 and R3. Any change at this node implies variation of drain voltage of the cascode transistors and hence current variation. To minimize this effect, R3 needs to be considerably small so that MainDAC signal gets attenuated. Based on the above two constraints, a value of R3=1K was chosen initially (this value was later changed to 10K as will be explained later). This may not be the most optimized choice of resistance. But as explained earlier, since the main idea of the work is to prove the calibration concept, not much emphasis was put on optimizing the design in the first run.

Requirement of Constant Offset Voltage

The calibration voltage can range between Vref,Low and Vref,High (the range is approximately 40mV). If the lower end of resistor R4 is connected to Vref,Low then current varies from:

'0' when Calibration voltage = Vref,Low to

'Imax' when Calibration voltage = Vref,High – Vref,Low

This results in transistors moving out of saturation at lower values of current. For R4=R3=1K, the current variation with calibration voltage is from 0-40uA in steps of 10nA (4096 steps). This corresponds to a huge order of variation in current. To reduce this effect an offset current of approximately 20uA is added, by connecting the lower end of R4 to some constant voltage that is less than Vref,Low. The current swing is then from 20uA to 60uA. The resulting offset at the output can be accounted for by shifting the value of the input from the MainDAC.

Deciding the value of offset current

Simulations were performed to see the effect of offset current on the output linearity.

Case 1: R4 was chosen as 1K (resulting in full-scale current = 40mV/1k = 40uA); I_{offset} = 5uA; Current variation = 5uA ~ 45uA. The linearity plot of the output is given in Figure 3.48.

Case 2: R4 was chosen as 1K; I_{offset} = 80uA; Current variation = 80uA ~ 120uA. The linearity plot of the output is given in Figure 3.49.

From the two plots, it can be seen that the linearity improves if the offset current increases. However the INL values are not within the requirement. An alternate approach of reducing the full-scale current range was then adopted. R4 was increased to 10K resulting in 4uA current variation over the

entire range. The offset current was set to 20uA, resulting in the full scale current variation from 20uA~24uA. The linearity results of the simulation are shown in Figure 3.50. It can be seen that INL of the converter is within 0.25LSBs confirming 12-bit performance. Hence, these set of parameters were finally decided for the summing circuit.

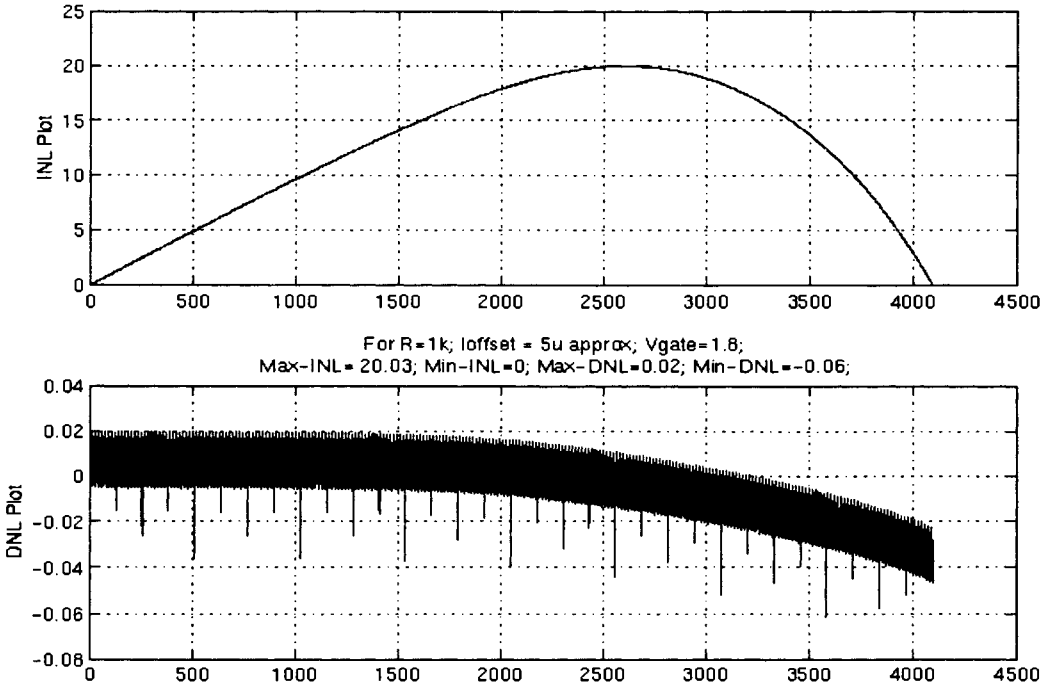


Figure.3.48 Linearity plot for 5uA offset current (Y axis corresponds to CalDAC LSBs)

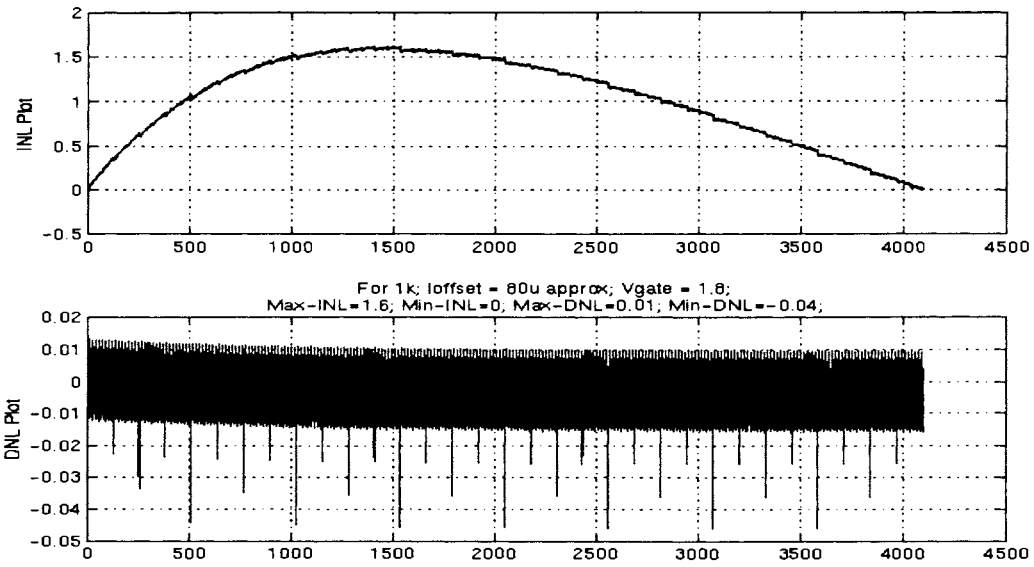


Figure.3.49 Linearity plot for 80uA offset current (Y axis corresponds to CalDAC LSBs)

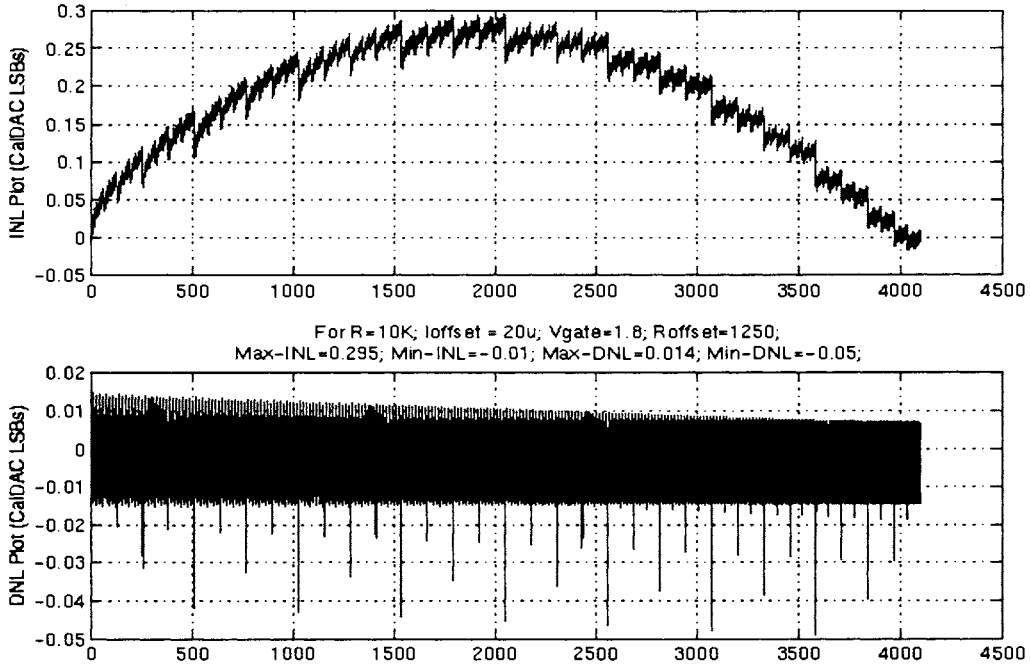


Figure.3.50 Linearity plot when R4=10K and offset current = 20uA

3.3.3. Reference Signal Generators

The reference signals that are required to implement the calibration circuit are:

- Vref,High
- Vref,Low
- Voffset

Although all these reference signals could be tapped from the MainDAC resistor string with slight modification in the existing circuitry, in order not to modify the existing layout, it was decided to include another string to generate these voltages. After analyzing the trade-off associated with area availability and power consumption, a resistor string of 70K was chosen. Voltage at various nodes of the string was tapped to generate the above reference signals. This approach of reference generation however suffers from issues like resistor mismatch due to process variations and random effects, resulting in errors in generated reference signals. But as explained in Section 3.1, the exact scaling of the supply range is not required as the calibration circuit has been over designed to account for scaling errors.

In order to avoid loading of the resistor string (that is used to generate the reference voltages) by the CalDAC, buffers need to be used while tapping the reference voltages. A single buffer design is sufficient as all the three cases (buffers for the three reference signals) have similar current and voltage swing requirements. As these buffers have to drive the CalDAC, an output stage is necessary to prevent gain drop due to external loading. It needs to be observed that the load seen by the buffers (due to CalDAC) is not constant and changes with change in input digital code to the CalDAC. This leads to different current being drawn from the reference sources for different inputs. It is necessary to make sure that the amplifier's gain is sufficient even when the worst case current is drawn. Also it

is important to ensure that the gain of the amplifier is constant over all values of output current. Based on the requirements and keeping into consideration the design complexity a simple structure consisting of a telescopic cascode as first stage and source-follower as the output stage was chosen. A class-AB output stage would be the ideal candidate for this application, in terms of power dissipation. But to keep the design challenge simple a source follower output stage has been used in this work. There exists room for improvement of these amplifiers by implementing class-AB output stage. The schematic of the amplifier with operating points is given in Figure 3.51.

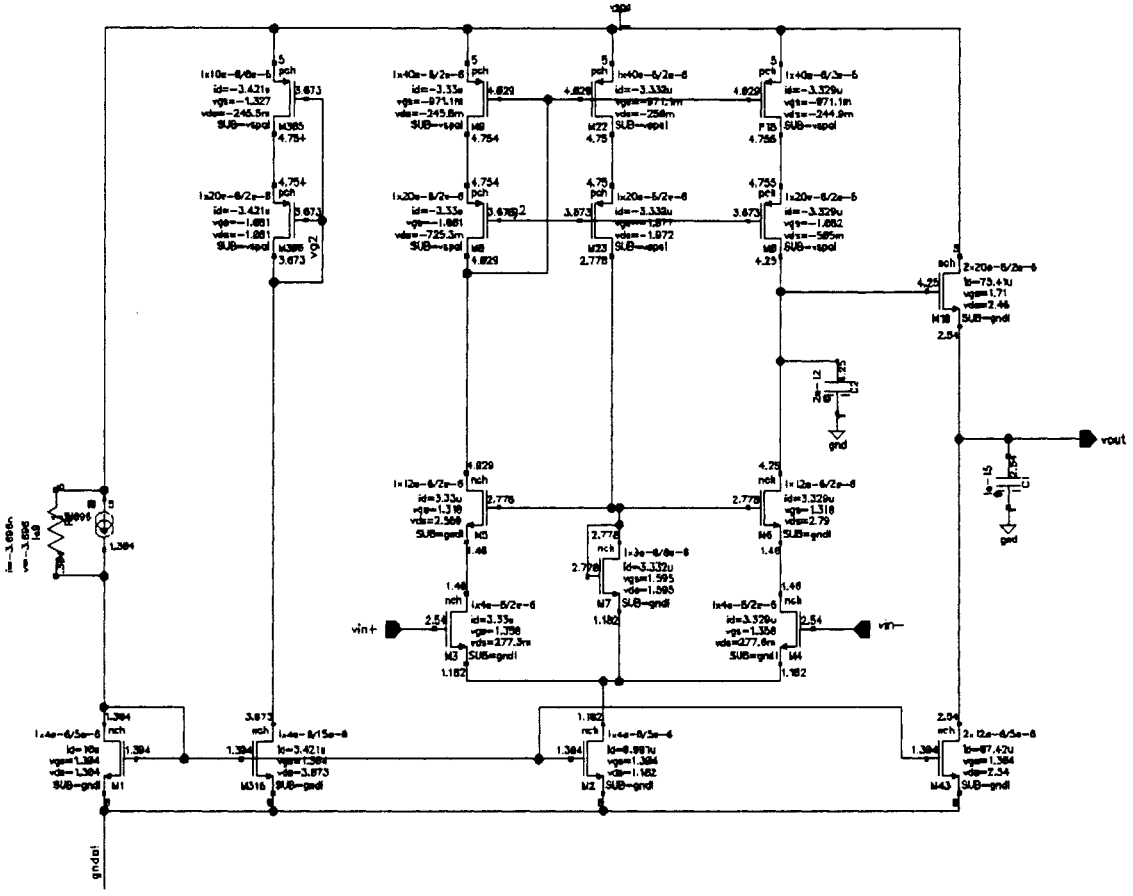


Figure.3.51 Schematic of the buffer used in reference signal generation

The AC response of the amplifier (for different value of load resistance, seen due to CalDAC) is shown in Figure 3.52. It can be seen that the gain in nearly constant over the frequency range of

interest for all loading conditions imposed by the CalDAC. Also the phase margin is > 70degrees for all cases.

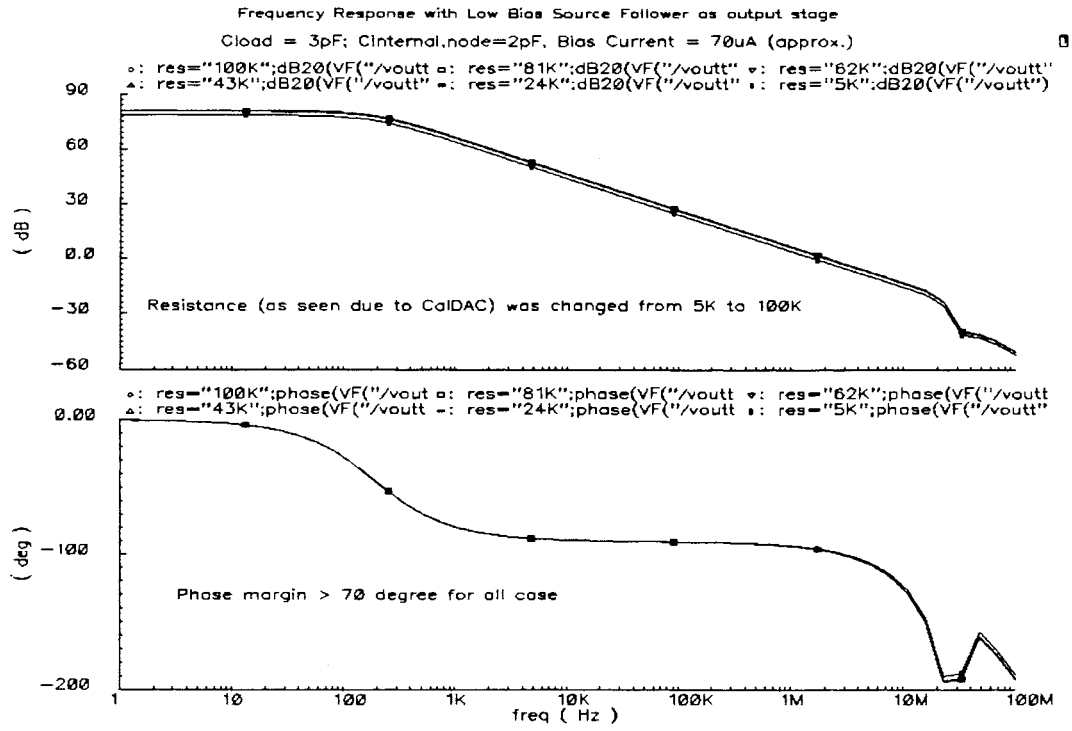


Figure.3.52 AC response of the buffer used in reference signal generation

The last block to be designed is the amplifier O1 that is required in the feedback loop as shown in Figure 3.46. This amplifier was also realized using a simple telescopic structure. The amplifier sees no resistive load as it is connected to the gate of the current mirror transistor. This eliminates the need of an output stage. The schematic with operating points for Vin=2.5V and 1pF load capacitance is shown in Figure 3.53.

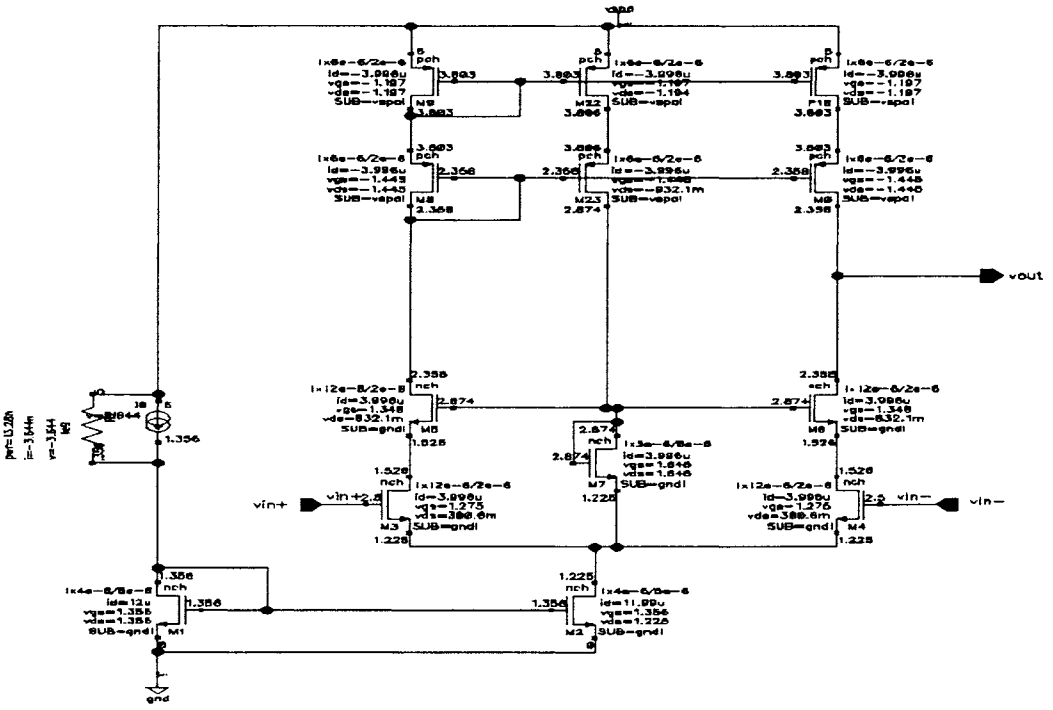


Figure.3.53 Amplifier (O1) used in the feedback loop in figure 3.46

The AC and transient response of the amplifier shown above is given in Figure 3.54.

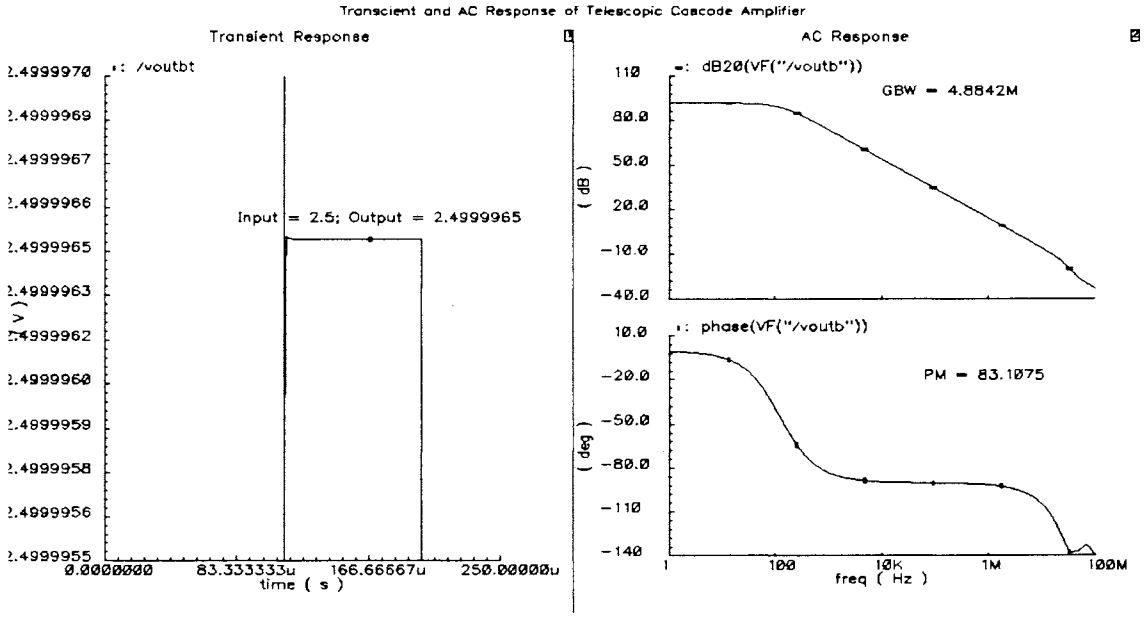


Figure.3.54 AC response of the amplifier (O1) used in feedback loop in figure 3.46

The bias current required for all the buffers described above were generated in the bias current generation block of the interpolating amplifier used in the MainDAC. Extra current mirrors were added to the existing circuit to generate more bias currents and hence did not require much additional circuitry.

The entire block including the CalDAC, summing circuit along with the above-described voltage reference generators was simulated. The schematic is shown in Figure 3.55.

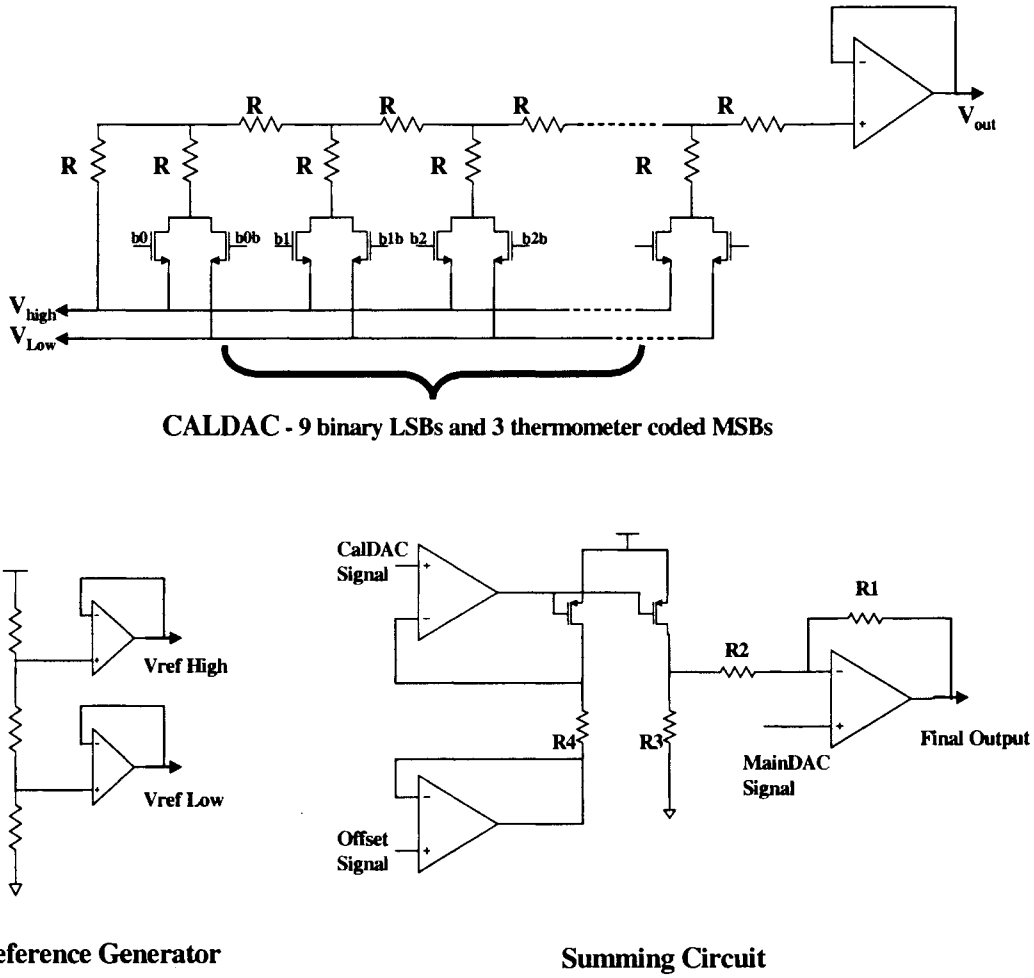


Figure.3.55 Schematic of the entire structure

The INL and DNL of the output signal are shown in Figure 3.56. The INL is within 0.3LSBs and is within the linearity specification.

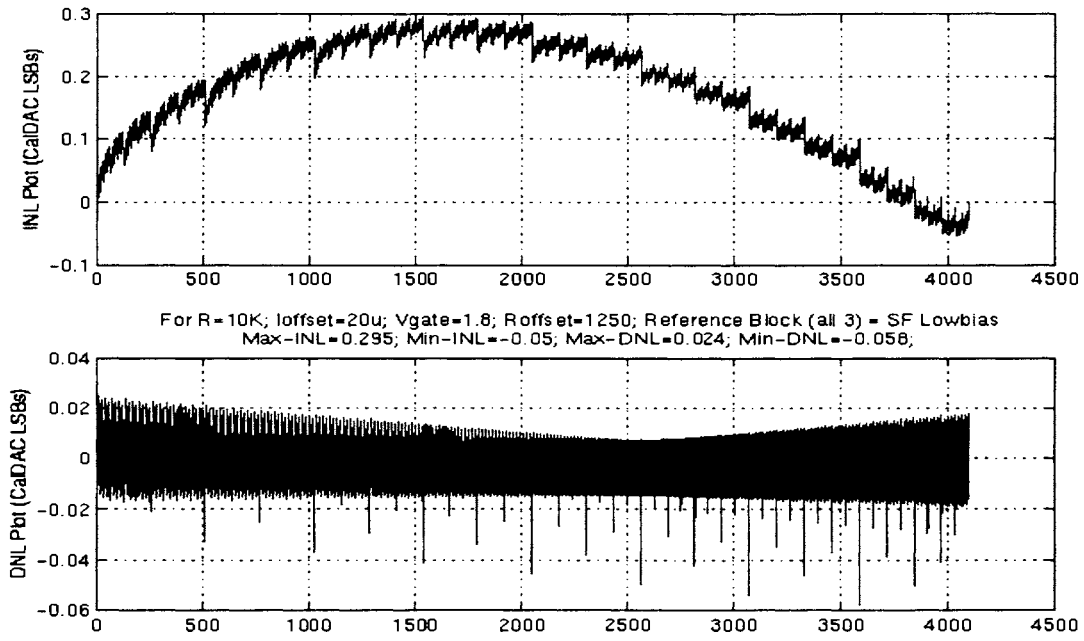


Figure.3.56 Plots of INL and DNL of the output signal

Effect of Temperature on Output Linearity

As explained in the Section 3.3.2, the summing circuit is expected to be temperature insensitive. To confirm the rationale behind the reasoning, the above circuit was simulated at different temperature. As expected the output linearity did not change considerably with temperature. Figure 3.57, 3.58 and 3.59 gives the INL and DNL at 27, -45 and 125 degrees respectively.

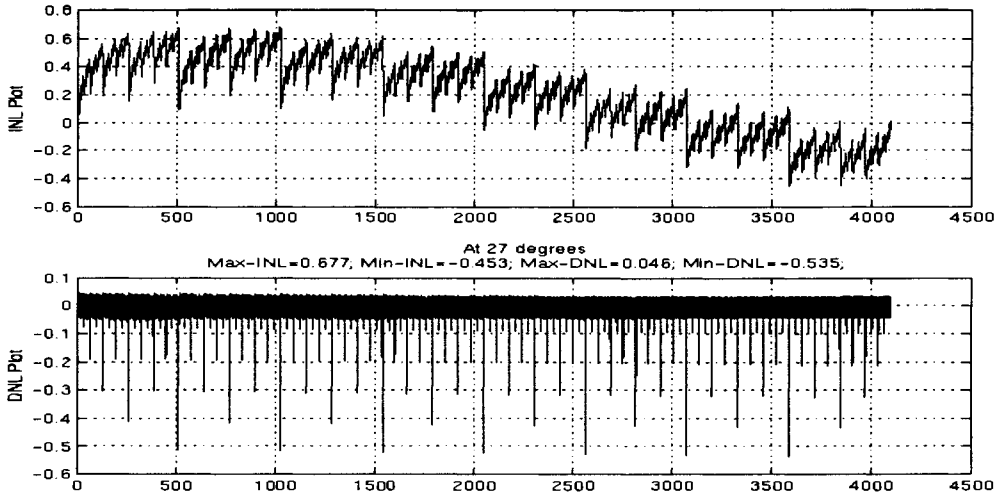


Figure.3.57 Linearity plot at 27 degrees (Y axis corresponds to CalDAC LSBs)

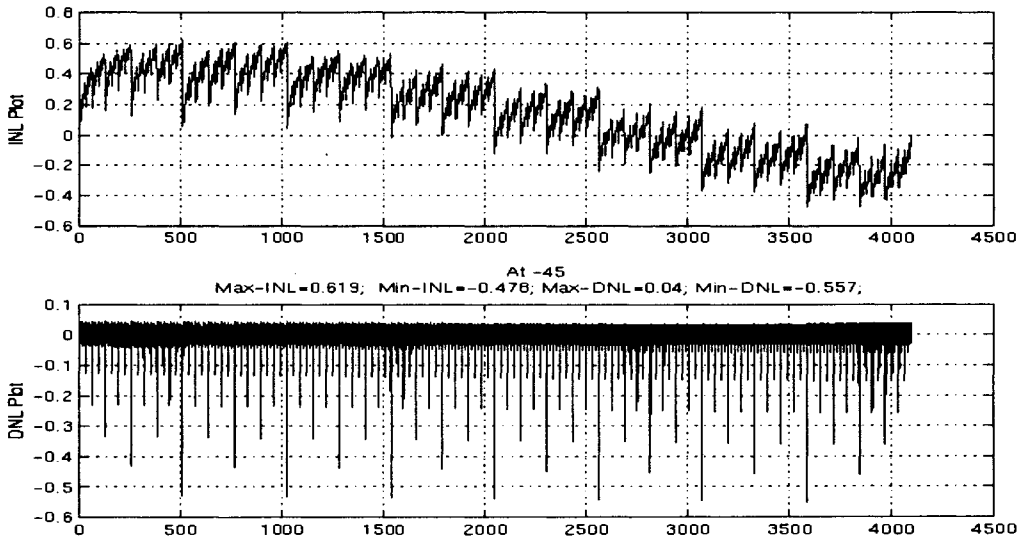


Figure.3.58 Linearity plot at -45 degrees (Y axis corresponds to CalDAC LSBs)

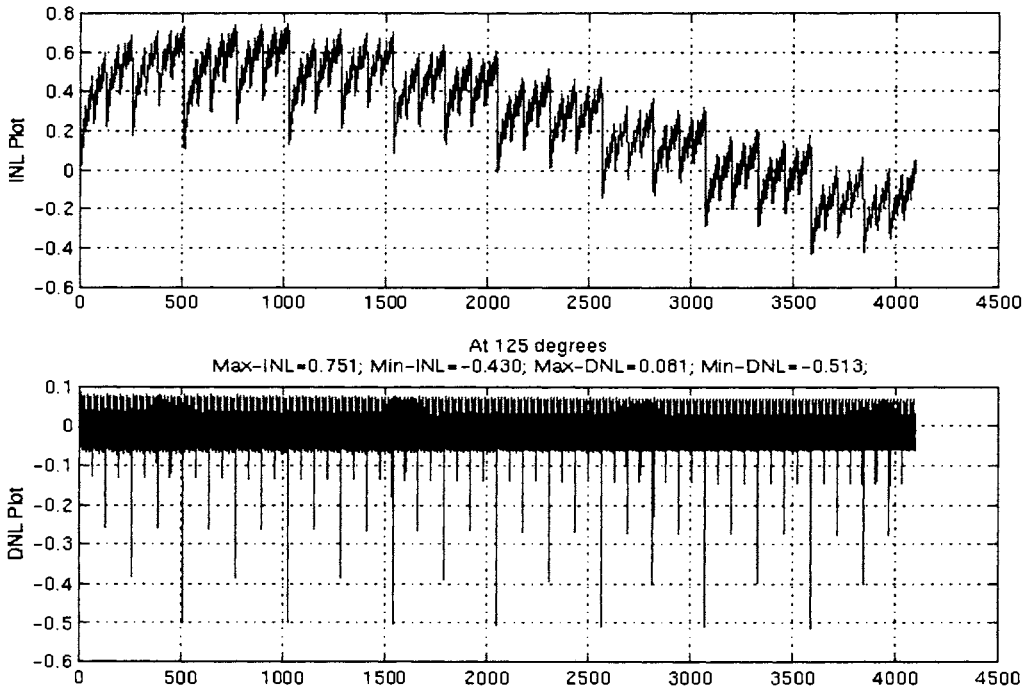


Figure.3.59 Linearity plot at 125 degrees (Y axis corresponds to CalDAC LSBs)

Also the Max-INL, Min-INL, Max-DNL, Min-DNL values are reported in the figures. A complete area and power estimation of this new architecture has been done and is included in Appendix D.

3.3.4. Top Level Simulation

All the modules including the digital and analog portion of the calibration circuit were finally hooked together and the entire circuit was simulated. The same set of fuse values as used in Section 3.2 was used for the simulation. The accurate value of offset voltage was calculated and was tapped from the main string. A complete transient simulation was performed and the output is shown in

Figure 3.60. The output seems opposite to the plot shown in Figure 3.34. This is because, Figure 3.34 shows the variation of INL values, and Figure 3.60 represents the following equation.

$$Vout = 2 * Vin - Vcalout \tag{3.6}$$

Since the calibration voltage is subtracted, the output is opposite. It is evident from the output plot that the calibration circuitry performs the desired operation.

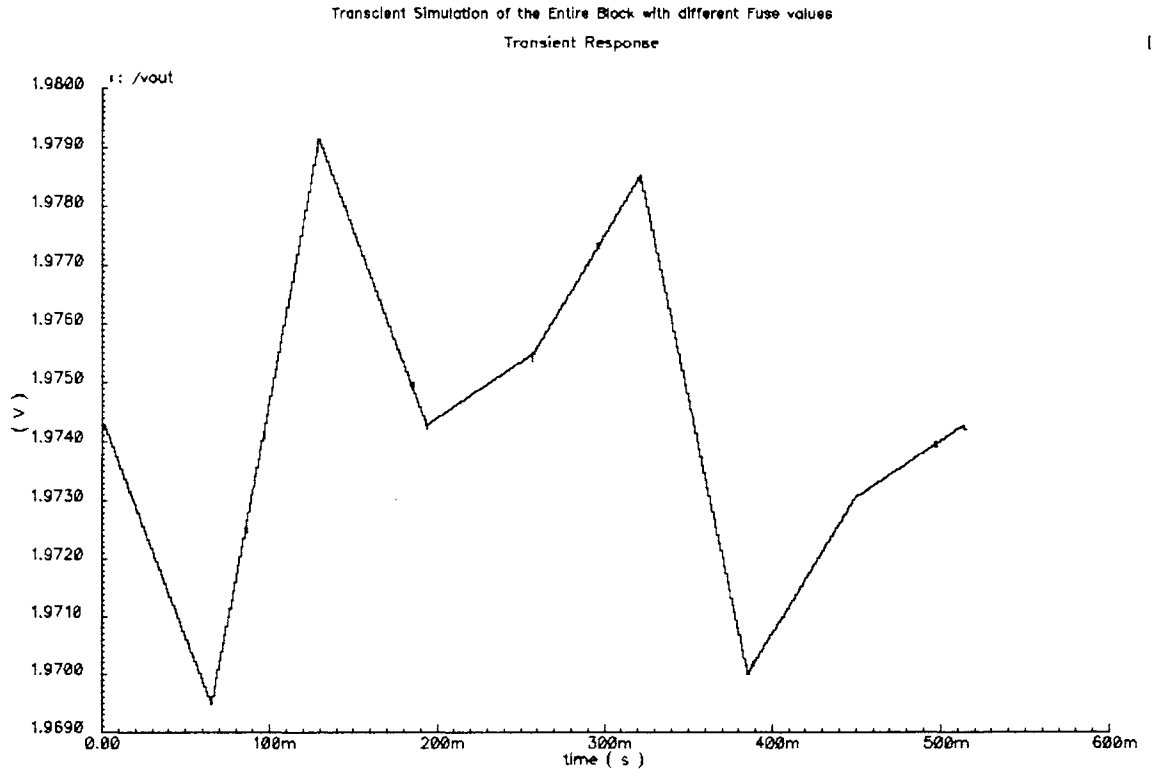


Figure.3.60 Output obtained from simulation of the entire schematic

Having designed all the blocks and confirmed the functionality of each block, the final step is to generate the layout of the entire design and fabricate the circuitry. The above project being part of Texas Instruments product cycle, the layout and fabrication is being held till all the logistics relating

to the project are worked out. Final silicon results and estimation of the improvement that could be obtained from this scheme is yet to be done.

3.4. Future Work

In this section suggestion on future work on this project has been proposed. There are potentially three areas where more work can be done. The first suggestion is to get a more optimized design for the method adopted in this work. Since the primary aim of this work was to prove the concept, not much emphasis was laid on optimizing the design of various modules. However, as emphasized in different part of the report, with more attention, many of the modules can be simplified and a more optimized design in terms of area and power consumption can be obtained. More systematic coding in VHDL, and a better area and power minimized synthesis in synopsys could be done. Also the various analog blocks can be designed for low power operation.

The second suggestion involves adopting a more complicated approach to solving the problem than that is described in this work. The suggested method takes the technique discussed in this work a step further. Instead of limiting the INL estimation to piecewise linear approximation, certain spline approximation could be done by taking more control points. This could result in more accurate approximation of INL at each input code, thereby resulting in a better correction. It is been conjectured that by using a spline approximation the performance of the DAC can be improved from uncalibrated 10-bit performance to nearly 14-15 bit performance. However this will also require additional circuitry which may be more involved and complicated.

The third suggestion involves a new approach for digital calibration of the device. A brief description of the method is given below. Not much work has been done towards this method and is still an open area of research.

Brief description of the algorithm

Looking at the typical INL profile of DAC under consideration, it looks like choice of knots that are equidistant may not always provide the best solution. A slight modification of the method, by placing the knots in an optimum manner could result in an improvement in performance. Consider the INL plot of a typical device as shown in Figure 3.61 (obtained from a product data sheet). A piecewise linear curve that would be obtained from the choice of 9 equidistant knots is also included in the figure. It can be seen that in segments 3, 4, 7 etc, the piecewise approximation is much different from the actual profile since the INL at knot locations are near zero. However, for the same number of knots, a more optimal allocation of points could result in a piecewise curve that follows the INL profile more closely as shown in Figure.3.62.

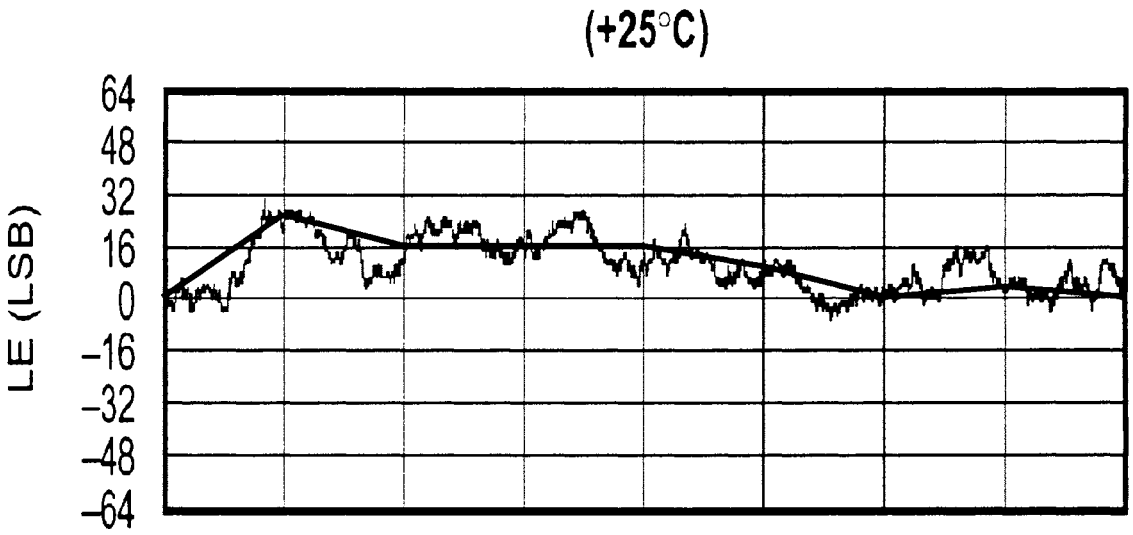


Figure.3.61 Piecewise Linear Approximation using equidistant allocation of points

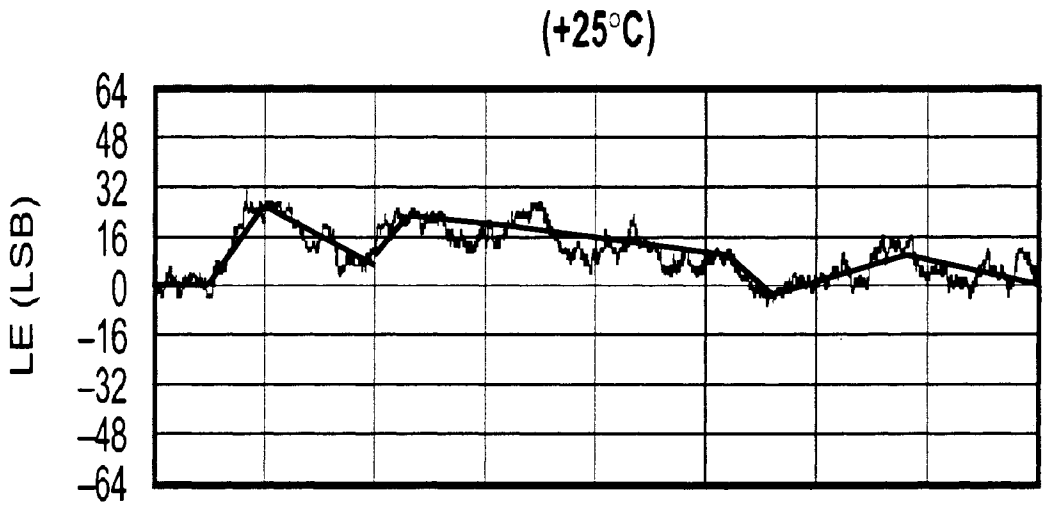


Figure.3.62 Piecewise Linear Approximation using optimal allocation of points

Let us again consider an INL profile as shown in Figure.3.63. This sort of INL has been introduced just to prove the need for optimal allocation of points. As can be seen from the figure, the second case gives much better approximation than first due to the correct choice of knots. In the first graph, the INL follows a nearly sine wave profile and since the control points are at zero crossings, the 1st order non-linearity is not captured.

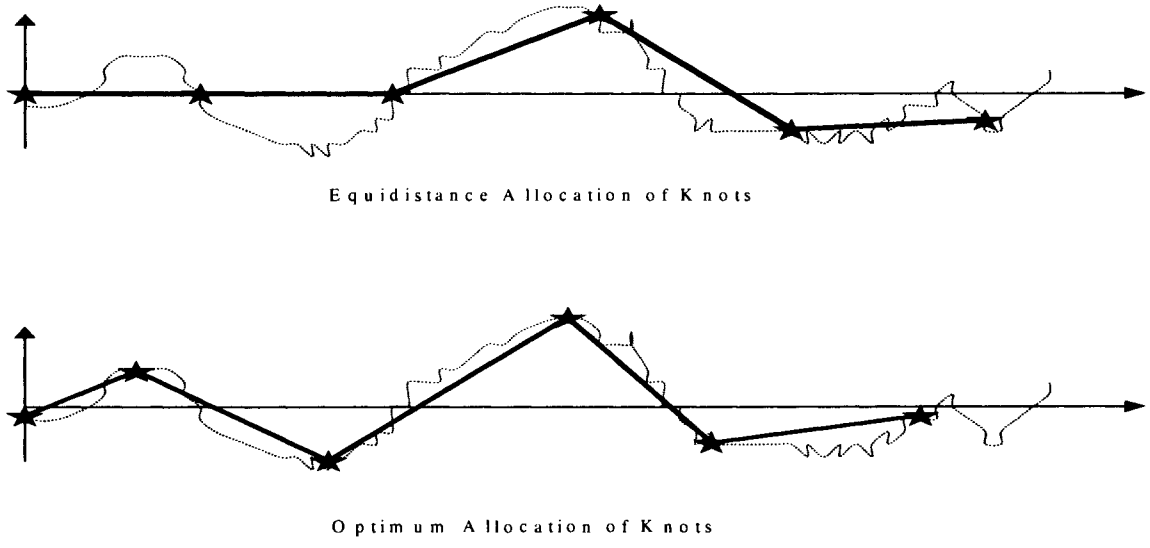


Figure.3.63 Effect of Optimal allocation of knots

Design Issues

For the above approach it may seem like optimal allocation of knots may result in storing the address values (or equivalently x-axis value) of each knot resulting in an increase in the number of fuses. In addition, since the difference between control points is no longer a power of 2, it can result in the need for a divider circuit.

Both the problems can be solved by a proper choice of control points. Instead of choosing the segment length to be equal to 8192 (2^{13}), the segment length can be chosen to be an appropriate power of 2 (for example, 2^{10},2^{11},2^{12},2^{13},2^{14} etc). It is evident that by even restricting the segment length to different powers of 2, a much better approximation can be achieved than that obtained using "equidistant knots". Thus, division can still be restricted to shift operation, except that the length of different segments will then be different powers of 2.

The next issue is related to storing the address values of each knot locations. This can be addressed as follows:

Presently 63 fuses are required to store the INL values at 9 control points. By optimal allocation of the knots we can possibly reduce the number of control points required. If the number of control points can be reduced to 8 by optimal knot locations, then 7 fuses would be available and can be used for other purpose. Since the optimal knot locations are not known before the fabrication of the chip, different combinations of 8-knot locations can be chosen beforehand and the x-coordinates of the knot locations can be stored in a look-up table. For instance, if 128 different combinations of 8-knot locations are considered before fabrication, then 128 different piecewise approximations can be obtained. Once the chip is fabricated and the INL values at all codes are obtained from the final test, an optimization algorithm can be used to give the best choice of 8 knot locations (among the 128 different combinations) that will best fit the given data. The 7 fuses can then be programmed to point

to the corresponding location in the look-up table. Figure 3.64 gives a simple representation of the proposed scheme.

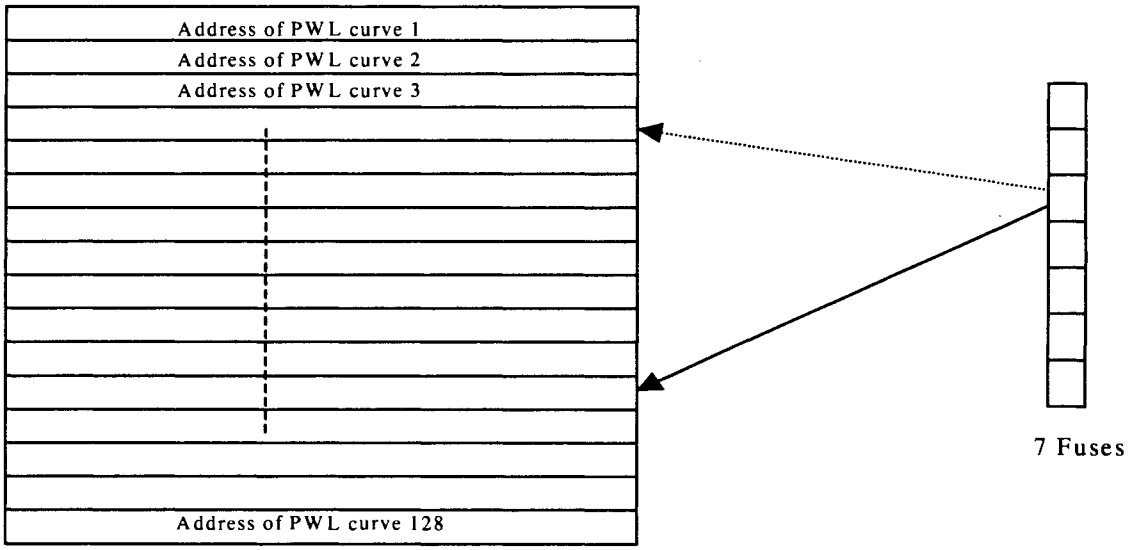


Figure.3.64 Addressing scheme for the modified algorithm

Note:

- The digital blocks required will be a bit complicated as the segments lengths are not all same. However, as the size of the digital block is less compared to other circuitry, this is probably affordable.
- This concept could be extended, for e.g., to store 256 different combinations of 8-knot location with just an addition of 1 extra fuse.
- Since the digital portion is less in area, the memory needed to store the address of different PWL curves is hopefully not too high.
- A second order spline approximation could also be done instead of a piecewise linear approximation. Nevertheless, the complexity of the implementation is then an issue.

4. AM-BIST STRATEGY FOR ADC CHARACTERIZATION

4.1. Introduction

In the last decade, the semiconductor industry has witnessed a rapid growth in the application of increasingly complex mixed-signal circuits in the communication and signal processing arenas. This growth coupled with industry-wide improvements in semiconductor processing has created a large market for low-cost mixed signal integrated circuits. Paralleling this downward cost pressure are increased demands on the number, accuracy and complexity of testing steps in the characterization and production test environment. As a result, production testing costs are becoming a rapidly growing and increasingly significant portion of the overall manufacturing costs [1] for several important classes of circuits thus causing the emergence of testing as the main bottleneck limiting the reduction of IC production cost for circuits in these classes.

The conventional way of testing and characterizing a circuit involves the use of high cost commercial testers. Testers used in this environment are often termed ATE (Automatic Testing Equipment). These ATEs are used for initial characterization of the devices and for mass production testing. The initial testing of prototype devices includes extensive testing steps to evaluate the performance under worst case conditions. Tests under varied loading conditions and different temperatures are typically performed as well. Testing also needs to be done over a large number of devices and over several lots of material before the results can be considered valid. This makes characterization testing a high cost and economically unviable step for high-volume production. Typically once the worst case characteristics are identified, production test program is created from a subset of the characterization steps. This production test program drives a commercial tester and

provides a high level of confidence that almost all bad parts are identified and purged prior to shipment of parts to a customer.

In a test setup, the ATE serves the dual purpose of generating precision stimuli to be input to the device under test (DUT) and analyzing the various outputs obtained from the device. This requires the tester to be equipped with high precision signal generators that are more accurate than the device to be tested. Also various software routines and digital processing circuits are required to post-process the test output. Mixed-signal testers are normally designed to test varied modules each requiring special tests to measure application-specific parameters. They are therefore designed for general purpose testing, equipped with a broad range of functionality. This calls for a large investment in the design and maintenance of commercial testers. The testing cost is mainly attributed to two factors. The direct initial investment in the million dollar testers and the ongoing operational and maintenance costs of using these testers on each device that is tested. This has been the main driving fore behind a growing research emphasis on Analog and Mixed-Signal Built-In-Self-Test (AM-BIST) approaches to device characterization. Although BIST solutions for digital circuitry have been in effect for a long time and have matured, the BIST approach for analog and mixed-signal circuits is still in its infancy. In this work, efforts have been made towards generation of a new algorithm for characterization of one useful class of mixed-signal circuits, namely data converters.

One of the mostly widely used mixed-signal modules is the Analog-to-Digital Converter (ADC). With the design and production of more and more ADCs with varied specifications, the associated aspect of testing has become very challenging and costly. Although the testing of lower-speed low to medium resolution ADCs (12bits and below) has become a known art, testing challenges remain for converters with resolution at the 16bit level and above while 14bit converters can be considered borderline in terms of test capability and test cost on presently available production

testers. On the other hand, communication ADC testing is still a challenge at even the 10-12bit level due to high-speed stimulus and low clock jitter requirements.

For high resolution ADCs (16bits and above), testing cost is determined primarily by the device resolution and not by the ADC sampling rate. This non-intuitive result can be explained by the following observation. As the number of codes increases, the linearity requirements of the source driving the ADC (linearly) increases. This performance requirement enforces slow, high-precision signal generator architectures that require long settling times. Even if the sampling rate of the ADC is in the order of tens of megahertz (i.e. Pipeline ADCs), the ADC must still wait for the source to settle, resulting in overall long testing time. For example, to test a 16bit ADC, if a 20-bit delta-sigma DAC were used with 1 millisecond settling time, and if 10 steps per code were used, approximately 11minutes of test time would be required to test all codes of the ADC for a single temperature cycle. Assuming 3 temperature testing, the test time can exceed half an hour for a single ADC. Therefore, cost of performing an all-codes test on a high performance mixed signal tester can exceed $30 per ADC which is prohibitive for most market opportunities.

Due to cost constraints, ADC manufacturers have chosen to do "reduced code testing" using servo-loop techniques to dramatically reduce test times. However, this approach has its own disadvantages. It improves the test times by reducing test coverage. Reduced code testing techniques can only be used for ADC architectures that can be fully characterized by reduced codes, such as the SAR architecture. Even for the SAR architecture the smooth non-linearity of the S/H circuit cannot be fully characterized by reduced code testing. Also, servo-loop methods cannot guarantee no missing codes.

Apart from the tester time, another disadvantage associated with long test times is the effect on linearity performance due to 'voltage drift'. As the measurement duration increases the source is no

longer stationary within the accuracy of the ADC. It is possible for a source to drift by hundreds of microvolts when turned on for short durations. This demands methods of fast stimulus generation to reduce test cost. Fast stimulus generation will also increase the test coverage, as more ADC codes can be tested for linearity if the source is sufficiently fast, increasing the verified quality of the product. It will also improve the accuracy of the test since voltage drift will be lower with a shorter duration test.

In this work, methods that can use a nonlinear but stationary source, such a resistor string DAC to test the linearity of the ADC has been proposed. String DACs of 16bit resolution are currently available in the market place with 10us settling time and ultra low drift and DACs with 18bits of resolution are under development. A string DAC offers about 10-11bits of linearity, 18+ bits of monotonicity and they are two orders of magnitude faster than their high performance delta-sigma counterparts. The proposed method reduces data capture time but increases the computation time because of the need to mathematically remove the effects of source non-linearity. For sources that are monotonic and stationary, the solution is independent of the amount of source linearity. As long as a low cost production tester is capable of driving this nonlinear but stationary source and capturing data from the ADC, the proposed test solution can be implemented on-chip, as an AM-BIST (analog and mixed signal built-in self-test) structure. Unlike the traditional approaches [2, 3, 4] that require linear and stationary input stimuli, the new approach is based upon nonlinear and stationary stimuli. To validate the performance of the Device-Under-Test (DUT), post-processing of output data with Digital Signal Processing (DSP) algorithms is used.

The potential advantages of BIST approaches are explained in Section 4.2. The various standard testing approaches and the associated test problems are described in Sections 4.3 and 4.4 respectively. The details of the input signal generator and the Device-Under-Test (DUT) system are covered in Section 4.5. The new algorithm is explained in Section 4.6 followed by the modeling of

the test setup in Section 4.7. Simulation results and experimental results to corroborate the performance of the algorithm are given in Sections 4.8 and 4.9 respectively.

4.2. BIST Solutions: Value-added With On-Chip Solutions

The obvious advantage of a BIST solution and the end-goal in most proposed BIST approaches is the reduction or elimination of the costs associated with using production testers. Unfortunately, the cost-benefit tradeoffs with this end goal coupled with the limited success at developing a viable strategy for testing middle to high-end parts have not provided sufficient incentives to industry to warrant development or adoption of existing BIST solutions proposed in the literature. However, the concept of a general purpose BIST capability for testing both stand-alone mixed-signal parts and ultimately analog functionality on larger System-On-Chip (SOC) structures is attractive from alternative viewpoints. One big challenge in building SOC circuits is the ability to test/debug the performance of internal blocks that can not practically be presented to an output test port without compromising the performance of the block itself. In these situations, the use of a production tester may not be viable whereas BIST structures can inherently provide the ability to test deeply embedded analog functionality without requiring an external analog I/O interface.

A second potential advantage of using BIST solutions can be realized with modest architectural modifications. The major purpose of using a commercial tester is to identify and eliminate bad parts. With this strategy, the "tester" does not enhance fundamental device performance and provides no yield enhancement. Currently, much of the yield loss in many analog and mixed-signal parts is due to various soft faults associated with random variations of matching critical devices. Most of these soft faults can be partially or completely corrected using calibration algorithms and the corresponding calibration circuitry. Some self-calibration algorithms used in

analog and mixed-signal parts actually already provide much of the functionality that is needed for implementing BIST and some BIST approaches can be extended to provide self-calibration as a part of the design process with modest architectural modifications. By jointly considering the issues of Analog and Mixed-Signal BIST (AM-BIST) and built-in calibration, value can be added with an AM-BIST capability by salvaging devices that would be scrapped due to soft-faults with a commercial production tester. This feature will become increasingly important in SOC structures where a large number of functional blocks must all meet stringent performance requirements for the SOC structure to meet acceptable performance standards.

A third advantage associated with an AM-BIST approach coupled with the built-in calibration capability is the ability to actually enhance performance specifications while still maintaining yield targets. By combining AM-BIST strategies with the built-in calibration capability, inherent matching requirements at design can be relaxed thus decreasing device size and correspondingly increasing the speed of operation of some classes of speed-critical circuits. These advantages can provide value-added to AM-BIST solutions that should help offset some of the concern about the silicon costs associated with the area overhead needed for the AM-BIST circuitry.

Even though many BIST solutions have been proposed in literature, there has been only limited industrial adoption of these approaches. This is because most of the existing approaches to AM-BIST attempt to replicate the existing production testing approach, including the tester, on silicon with some application-specific reduction in the flexibility of the "tester". Such approaches are attractive because the long-term investments in production testing strategies by the semiconductor industry have resulted in the evolution and maturing of testing algorithms that are both effective and time efficient. It is envisioned that by replicating the "tester" on silicon, similar algorithms can be used. But since the performance requirements of a commercial tester are generally much more stringent than those of the Device-Under-Test (DUT), there exist major concerns that the challenges

involving the design of the on-chip tester may be more challenging than the design challenges for the DUT itself. This has been the major stumbling block in the practical implementation of most of the proposed solutions. By contrast, in this work, an AM-BIST strategy is proposed that attempts to overcome this major concern by exploiting lower accuracy analog I/O components for testing that can be easily designed and that are much less complex than the DUT.

4.3. Standard Testing Approaches

Testing of analog and mixed-signal parts involves the verification of various performance parameters that depend on the circuit under consideration. In data converters these specifications include the INL, DNL, offset, ENOB, SNR, SFDR and others. In amplifiers they may include gain accuracy, distortion, noise and bandwidth. Mid-band gain, band edges and distortion levels may be the dominant parameters of concern in filter structures, while in communication circuits they typically include BER, jitter and frequency accuracy.

A significant subset of the tests involve using a high performance tester to excite the circuit with a precisely known excitation along with observing the output of the DUT either directly from the DUT output or through a precise piece of test equipment. The output may vary depending on the circuit and application. The output is then compared with the expected output value to assess the performance of the DUT. The type of signals used for the excitation varies from application to application. It could be a ramp signal for testing ADCs, very high spectral purity sinusoids for testing linearity and dynamic range characteristics of circuits, or coherent test tones for testing a range of functions including filter gain and linearity characteristics. Irrespective of the type of excitation used, in almost all cases, the quality of the input signal is critical and it must be much better than the characteristics of the DUT that are measured.

It follows that in many attempts to replicate the tester on silicon, one of the main tasks is to generate high quality analog signals on-chip that can be used as input stimuli [5]. A second task in some applications is to develop very high quality measurement circuitry for measuring the output characteristics. Once accurate inputs to and outputs from the DUT are realized, standard algorithms can be used for conducting the tests, algorithms, that have been developed with the assumption that the input stimuli are perfect and that all non-idealities seen at the output are attributable to the DUT.

However, in spite of having made significant progress at generating on-chip signals that are very precise, there is always a practical upper bound on the maximum accuracy that can be achieved with such stimuli. A more general approach to the on-chip testing problem is formulated in the following section. As a proof of concept, this will be followed by the development of a specific on-chip testing strategy for one useful class of circuits.

4.4. Fundamental Testing Problem

A typical test setup is shown in Figure.4.1. It consists of the system or device to be tested (DUT), an input signal generator that is used to excite the system, and a system measurement block, that has information about the output obtained from the DUT.

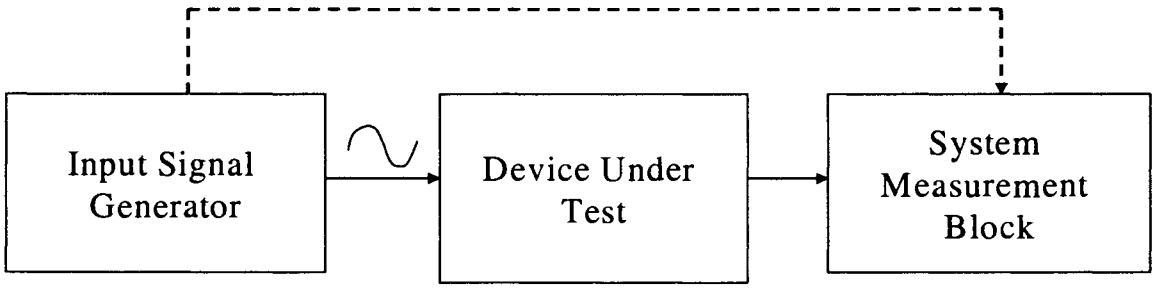


Figure.4.1 Typical Test Setup

The measured output is comprised of the combined effects of the DUT characteristics, the input characteristics, and the characteristics of the measurement block. If complete information is available about the input stimuli and the measurement block, the problem of characterizing the DUT or equivalently of identifying the system under test is reasonably straightforward and well-studied. Almost all conventional system identification approaches follow this concept of comparing the measured output with the ideal or desired output, for a particular known and ideal input to the system.

One of the main challenges for implementing a BIST structure for mixed-signal devices is the difficulty of generating an ideal input or, if the input is not ideal, of obtaining sufficient information about the input signal to accurately characterize the DUT from the measured response. This is similar to the problem of blind approximation often faced in many fields of engineering, in which an unknown system driven by an inaccessible signal must be identified solely based on the system's output. Even though the identification of a system with little or no information about the system or the input seems to be a daunting task within the BIST community, elegant solutions can be achieved if some a priori knowledge is available about the input and/or the system. The controls literature provides some insight into how this problem can be approached.

With this broader concept in mind, a new strategy for BIST becomes apparent. Considerable information, actually very detailed information, is available about the DUT since invariably sophisticated device models were used in the design of the DUT. Correspondingly, good models for an integrated signal generator and any integrated measurement circuits can also be developed. Thus, the natural question that arises is: What properties of the system, the excitation, and the measurement circuits are necessary so that the information available at the output is sufficient to practically identify the performance characteristics of interest of the DUT? Such issues are normally related to "controllability" and "observability" in various fields of engineering and significant work has been

done towards addressing them in other contexts. In this work, the same issues are dealt with in a general way.

4.5. Input Signal Generator/ DUT Systems

Many testing environments require both an Input Signal Generator and a System Measurement Block to test the DUT as indicated in Figure.4.1. In some applications, the DUT itself provides a digital output and this digital output contains sufficient information to characterize the DUT if sufficient information about the Input Signal Generator can be obtained without the use of a System Measurement Block. In these applications, the test environment can be simplified as indicated by the Input Signal Generator/ DUT structure of Figure.4.2. ADCs are one class of circuits that offer potential for being tested with this test environment. This section focuses on the formulation of an Input Signal Generator/ DUT test environment.

Consider the block diagram shown in Figure.4.2, where **H** is the ideal transfer characteristics of the DUT;

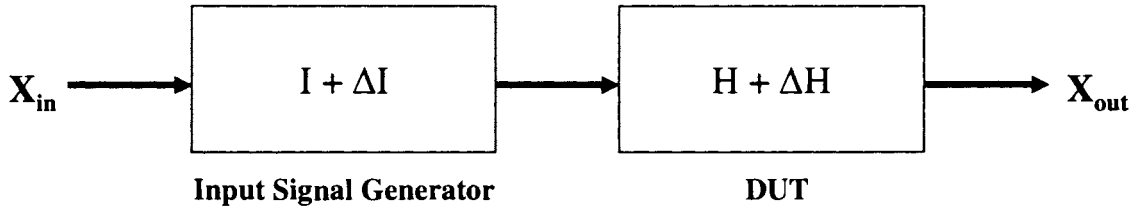


Figure.4.2 Block Diagram of Input Signal Generator/ DUT Test Environment

I characterize an ideal signal generator and ΔH and ΔI are the non-idealities of the system and the signal generator respectively. The output of the DUT which we will assume is a digital signal, can be observed directly. Define $I(X_{in})$ to be the desired "perfect" input and

$$X_{out:DES} = H(I(X_{in})) \tag{4.1}$$

to be the desired output. Two special cases will now be considered followed by a discussion of the more general structure.

Case 1: Ideal Signal Generator

For an ideal Input Signal generator, ΔI=0. The output X_{out} can be expressed by (4.2).

$$X_{out} = (H + \Delta H)(I(X_{in})) \tag{4.2}$$

If we assume the function (H+ΔH) can be approximately expressed as the sum of the functions H and ΔH, it follows that,

$$X_{out} \cong H(I(X_{in})) + \Delta H(I(X_{in})) \tag{4.3}$$

If we define ΔX_{out} by

$$\Delta X_{out} = X_{out} - X_{out:DES} \tag{4.4}$$

it follows from (4.1), (4.3) and (4.4) that

$$\Delta X_{out} \cong \Delta H(I(X_{in})) \cong X_{out} - H(I(X_{in})) \tag{4.5}$$

Assumption 1: When ΔI=0, ΔX_{out} contains all the information needed to identify ΔH.

This is the underlying assumption that is made while using any of the standard testing algorithms whether they are used on standard production testers or in most existing proposed BIST structures. It is assumed that when the input signal is ideal then the output of the system can be used to uniquely identify the system non-idealities.

Case 2: Ideal DUT

For an ideal DUT, ΔH=0. In this case the deviation of the output from the ideal output is given by

$$\Delta X_{out} = H((I + \Delta I)(X_{in})) - H(I(X_{in})) \tag{4.6}$$

If we now assume that the function I+ΔI can be approximately expressed as the sum of the two functions, I and ΔI, it follows that

$$\Delta X_{out} \cong H(I(X_{in}) + \Delta I(X_{in})) - H(I(X_{in})) \tag{4.7}$$

If we further assume that H(X+ΔX) can be expressed as H(X) + H(ΔX), it follows that

$$\Delta X_{out} \cong H(I(X_{in})) + H(\Delta I(X_{in})) - H(I(X_{in})) = H(\Delta I(X_{in})) \tag{4.8}$$

Assumption 2: When ΔH=0, ΔX_{out} contains all the information needed to identify ΔI.

This is a key assumption needed to identify the excitation because ΔX_{out} is assumed to be the only observable output in the structure of Figure.4.2.

Having considered the above two cases, we now address the more general and practical case in which both the system and signal generator have non-idealities.

If $\Delta H \neq 0$ and $\Delta I \neq 0$; then the deviation of the output from the ideal output is given by

$$\Delta X_{out} = (H + \Delta H)((I + \Delta I)(X_{in})) - H(I(X_{in})) \tag{4.9}$$

If we now assume the function (H+ΔH) can be approximately expressed as the sum of the functions H and ΔH, it follows that

$$\Delta X_{out} \cong H((I+\Delta I)(X_{in})) + \Delta H((I+\Delta I)(X_{in})) - H(I(X_{in})) \tag{4.10}$$

If we further assume that the function I+ΔI can be approximately expressed as the sum of the function I and ΔI and that H(X+ΔX) can be expressed as H(X)+H(ΔX) it follows that

$$\Delta X_{out} \cong [H(I(X_{in})) + H(\Delta I(X_{in})) - H(I(X_{in}))] + [\Delta H(I(X_{in})) + \Delta H(\Delta I(X_{in}))] \tag{4.11}$$

This can be simplified to

$$\Delta X_{out} \cong H(\Delta I(X_{in})) + \Delta H(I(X_{in})) + \Delta H(\Delta I(X_{in})) \tag{4.12}$$

The first term in (4.12) is the same as ΔX_{out} in Case 2 and the second term is the same as ΔX_{out} in Case 1. From (4.12), a necessary condition to identify the system and input can be formulated. If the third term on the right side of equation (4.12) is either negligible or if it can be eliminated by some alternate means, then we can identify both the system (DUT) and the input provided the first and second terms in the equation do not cancel each other. If not sufficiently small, it is conjectured that by using multiple inputs, the third term can be eliminated and an appropriate X_{in} can be chosen so that the first two terms do not cancel each other.

With this general idea in mind, one class of mixed-signal circuits, namely flash Analog-to-Digital converters (ADCs) have been considered as the DUT. A new approach has been studied that use multiple inputs and the corresponding outputs to identify both the circuit (DUT) and the input signal characteristics. In a BIST environment, the information about input signal is ultimately of no interest and is discarded. The extracted information about the DUT is essentially the desired test information about DUT. It should be apparent that this approach places no strict constraints on the input signal accuracy provided the function partitioning needed to obtain (4.12) can be justified.

A first-generation test algorithm that offers potential for use in a BIST environment or a production test environment will be described in the following section. This algorithm is being introduced as a "proof of concept" vehicle to demonstrate that a DUT can be tested with a relaxed performance Input Signal Generator. This algorithm uses code density information obtained by sweeping the input through a predetermined input range and taking samples of the output at the normal clock frequency of the ADC. If an ADC has N output levels ($N=2^n$, where 'n' is the bits of resolution of the ADC), then there are N output codes designated by C_0, ...C_{N-1}. The number of occurrences of the output code C_k in the test is termed the tally of code C_k and is denoted as C(k). The relative value of the output codes is termed the "code density". The code density information is often visualized or plotted as a histogram plot. Testing based upon this approach is widely used in industry and it is often referred to as "histogram based testing" or "code density testing". The digital output from the DUT in such an environment is precisely the code density data itself.

Since the sampling rate is generally fixed for a given data converter architecture, the number of samples that are taken during testing depends upon how rapidly the input changes. If T_{CL} is the clock period for the ADC and T_{SIG} is the duration of the input, then the total number of samples that will be taken is

$$N_{SAMP} = \frac{T_{SIG}}{T_{CL}} \tag{4.13}$$

and the average number of samples per code is

$$N_{AVG} = \frac{N_{SAMP}}{N} \tag{4.14}$$

If x is a time-dependent input to the ADC, the relationship between x and t can be expressed as

$$x = \Re(t) \tag{4.15}$$

where the function $\Re(t)$ defines the time dependence. Often the input waveform $\Re(t)$ is a single ramp in which all samples are taken as the ramp signal swings between its low value and its high value. For practical reasons, it is occasionally more convenient to have multiple ramps in the sampling period T_{SIG} or other time domain excitations such as sinusoidal input functions. In these cases, the time rate of change of the input will be large when compared to the time rate of change of the input with a single ramp. Irrespective of the exact nature of the time-domain input signal in the interval from 0 to T_{SIG}, it will be assumed that the time-domain input varies sufficiently slowly so that the output of the ADC at any sample time is not dependent upon the previous output code. Since a multitude of time-domain functions can give the same output code densities, it is of more interest when doing code-density or histogram testing to consider the code density or the accumulated code density of the input rather than the time-domain input waveform. An accumulated code density input function $F(x)$ is defined to be the expected number of inputs of $\Re(t)$ in the interval $0 < t < T_{SIG}$ that are less than x. The normalized accumulated code density function $F_N(x)$ is defined by

$$F_N(x) = \frac{F(x)}{N_{SAMP}} \tag{4.16}$$

When doing code density testing, the exact nature of the time domain input function is not of interest but rather the accumulated code-domain input function, specifically the normalized accumulated code density function $F_N(x)$, serves as the input.

If the time-domain input function is a linear ramp starting at 0 and ending at V_{REF} or is comprised of several identical linear ramps, then $F_N(x)$ is also linear in the input domain and $F_N(x)$ can be expressed as

$$F_N(x) = \begin{cases} 0 & for\ x < 0 \\ \dfrac{x}{V_{REF}} & for\ 0 \le x \le V_{REF} \\ 1 & for\ x > V_{REF} \end{cases} \tag{4.17}$$

If the time-domain input function is a linear ramp (or several linear ramps) starting at V_1 and ending at V_2, where V_1 and V_2 are not necessarily 0 or V_{REF} but where $V_1 \le 0$ and $V_2 \ge V_{REF}$, then $F_N(x)$ remains linear and is given by the expression

$$F_N(x) = \begin{cases} 0 & for\ x < V_1 \\ x\dfrac{1}{(V_2 - V_1)} - \dfrac{V_1}{(V_2 - V_1)} & for\ V_1 \le x \le V_2 \\ 1 & for\ x > V_2 \end{cases} \tag{4.18}$$

It should be apparent from (4.17) and (4.18) that if the time domain input $\Re(t)$ is linear in time, then the accumulated code density input function is linear in x.

If the time-domain input has some undesired nonlinearities or if it has a different functional form such as a sinusoidal relationship, the accumulated code-density input function will become nonlinear in x as well. If the nonlinearities in the accumulated code-density input function are precisely characterized, such a nonlinear code-domain input function can still often be used for testing the ADC provided the nonlinearities of the input function are appropriately considered when

interpreting the code density outputs. If the nonlinearities of the code density input function are not precisely characterized, the task of determining the performance of the ADC from the output code density becomes much more difficult or impossible.

If the input waveform $\Re(t)$ is a monotone and continuous time-domain function for $0 < t < T_{SIG}$, the accumulated input code density waveform relates to t by the expression

$$F(x(t)) = t\frac{N_{SAMP}}{T_{SIG}} \tag{4.19}$$

and from (4.15), it follows the inverse of $\Re(t)$, denoted at $\Re^{-1}(x)$, exists. Thus $F(x)$ can be expressed as

$$F(x) = \Re^{-1}(x)\frac{N_{SAMP}}{T_{SIG}} \tag{4.20}$$

Thus, in general, the nonlinearities of $F(x)$ are a scaled version of the nonlinearities of $\Re^{-1}(x)$. Since (4.17) and (4.18) indicated a linear time domain ramp input resulting in a linear accumulated input code density, it may be useful to consider an example to show that the inverse relationship given in (4.20) is consistent with the previous results.

Example: Linear Ramp Excitation

The linear ramp described in the presentation of (4.17) can be expressed as

$$\Re(t) = t\frac{V_{REF}}{T_{SIG}} \tag{4.21}$$

But since $\Re(t) = x$, it follows that

$$t = \Re^{-1}(x) = x\frac{T_{SIG}}{V_{REF}} \tag{4.22}$$

Substituting (4.22) into (4.20), it follows that

$$F(x) = x\frac{N_{SAMP}}{V_{REF}} \tag{4.23}$$

Normalizing this function with respect to N_{SAMP}, it follows that

$$F_N(x) = \frac{x}{V_{REF}} \tag{4.24}$$

which is what was previously given in (4.17).

End of Example

Corresponding to any accumulated code density function $F(x)$ is a code density function $\ell(x)$. The code density function is obtained from the accumulated code density function by differentiation, that is

$$\ell(x) = \frac{\partial F}{\partial x} \tag{4.25}$$

Thus, a linear accumulated code density function $F(x)$ corresponds to a constant code density function and an ideal linear time domain input waveform will result in a constant code density input function.

4.6. First-Generation Test Algorithm

In this section, a new histogram-based algorithm based upon the general formulation presented in the previous section that is suitable for testing the INL and DNL of a flash ADC with a highly nonlinear excitation is discussed. The standard way in which an ADC is tested in a production test environment is by using a linear time-domain ramp as the input stimuli. When a full-scale ideal ramp signal serves as the input to an ideal ADC, the digital output will have a uniform code density. However, when an ideal ramp is input to a non-ideal ADC, the output histogram is non-uniform as shown in Figure 4.3.

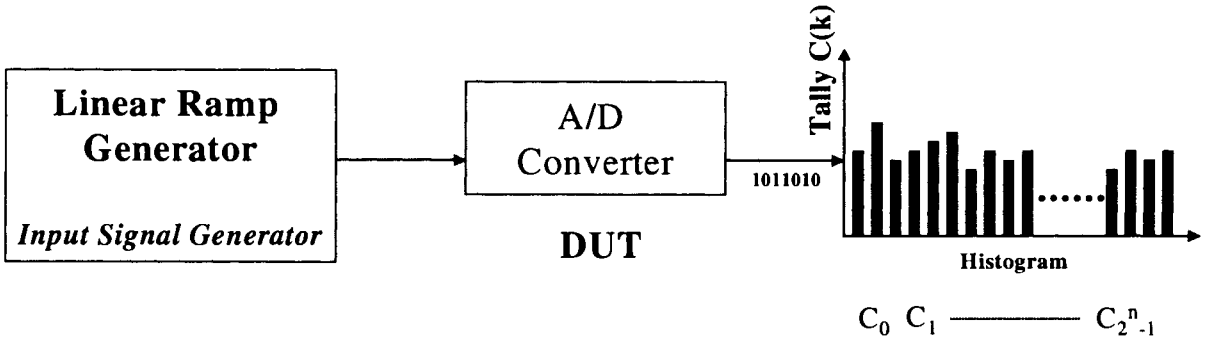


Figure.4.3 Histogram based testing of ADC with Ideal Ramp Generator

4.6.1. Code-Density Testing of INL and DNL with Linear Code-Domain Excitation

With code density testing Tally and Weight arrays [6] are used to characterize the code density of the output. Repeating the notation introduced in the previous section, it will be assumed that the ADC is of n-bit resolution with output codes denoted as C_0, C_1,.....,C_{N-1}. The Tally array, denoted by 'C', has N-2 elements corresponding to C_1, C_2,...C_{N-2} [7]. The tallies for code C_0 and C_{N-1} are not of interest because their input ranges may assume extreme values and thus the tallies for these

codes may be very large. The elements of the Tally array $C(i)$, are the number of occurrences of output code C_i. The average code width, corresponding to 1 LSB, is given by

$$\overline{C} = \frac{\sum_{i=1,2....N-2} C(i)}{N-2} \tag{4.26}$$

The INL and DNL, as generally defined, can not be precisely obtained from histogram data due to the quantization effect that is inherently associated with any code density testing scheme since the actual ADC break points are not determined from the code density measurements. Good estimates of the INL and DNL can, however, be obtained from code density data. In what follows, we will define a "code density" INL and a "code density" DNL and will not distinguish between the "code density" INL and DNL and the actual INL and DNL. It can be shown that the difference between the "code density" INL and the DNL and the standard INL and DNL is quite small provided $\overline{C}$ is not too small.

The code density DNL_i, in LSBs, is defined for $1 \le i \le N-2$ by the expression

$$DNL_i = \frac{\hat{C}(i) - \hat{C}(i-1) - \overline{C}}{\overline{C}} \tag{4.27}$$

where the accumulative tally function $\hat{C}(i)$ is defined for $0 \le i \le N-1$ by

$$\hat{C}(i) = \sum_{k=0}^{i} C(k) \tag{4.28}$$

The DNL expression can equivalently be expressed as

$$DNL_i = \frac{C(i) - \overline{C}}{\overline{C}} \qquad 1 \le i \le N-2 \tag{4.29}$$

The code density INL is defined relative to the fit line between the accumulated tally functions $\hat{C}(0)$ and $\hat{C}(N-2)$. It can be shown that the fit line is given by the equation

$$Y_i = \hat{C}(0) + \frac{\overline{C}(N-2) - \hat{C}(0)}{N-2} i = \hat{C}(0) + i\overline{C} \qquad i = 0,1,....N-2 \tag{4.30}$$

The INL_i, in LSB, is thus given by

$$INL_i = \begin{cases} 0 & i = 0, N-2 \\ \dfrac{\hat{C}(i) - Y_i}{\overline{C}} & 1 \le i \le N-3 \end{cases} \tag{4.31}$$

From (4.29) and (4.31), it should be apparent that this approach to ADC characterization can be easily incorporated into a BIST algorithm to detect the existence of faults in the ADC circuits or non-linearities in the ADC [8]. However this requires the generation of a very precise linear ramp signal that can be used as input. Any nonlinear error in the input ramp can lead to an incorrect interpretation of the results by using (4.29) and (4.31) resulting in false rejection or false acceptance of the devices [9]. Because of these concerns, recent research in histogram based BIST has been focused on generating analog ramp signals with accuracy substantially exceeding that of the device under test. In [10] the authors reported simulation results for INL of a ramp generator that had an INL approximately equal to that of an 11-bit DAC. A Sigma-Delta modulator was used to generate a bit stream that feeds a low pass filter to produce the linear ramp signal used in [11]. With a bit stream of 2^{14} bit length, reported simulation results indicated that an 8 bit ADC can be tested to 5% LSB accuracy. A cascaded current-source ramp generator was used in [12] to obtain reported simulation

results with sufficient linearity to test 14-bit ADCs but these simulations were not validated with experimental data. More recently the authors of [13] reported a current-source based ramp generator with experimentally verified linearity at the 15 bit level.

While a linear ramp signal is an excellent input choice for testing transfer characteristics of ADCs such as offset, gain error, INL and DNL, sinusoidal input signals are more suitable for testing frequency response, harmonic distortion, inter-modulation distortion, and other spectrally related issues [14],[5]. Similar to the case of ramp generation, high accuracy (i.e. excellent spectral purity) of the generated sinusoidal signal (single-tone or coherent multi-tone) is nontrivial to achieve and is critical to the AM-BIST schemes reported in the literature. A single bit Sigma-Delta modulator DAC along with low-pass or band-pass filtering has proven to be an effective way of generating high quality low frequency sinusoidal signals [15]. In [16] and [17], the authors used predefined recycled digital data stored in memory to feed a single bit DAC followed by a filter to generate analog signals. The advantage of this approach is that the bit stream and the corresponding pulse density modulated (PDM) signals are both periodic leading to better spectral properties for the generated signal. In these studies, experimental data confirmed the high quality of the generated signals. Some of these schemes were developed in the context of AM-BIST while others are developed for use with the Arbitrary Waveform Generator (AWG) used in ATE testers. Multi-bit Sigma-Delta modulators can also be used in conjunction with low-pass or band-pass filters to generate sinusoidal analog signals for AM-BIST [18].

4.6.2. Code-Density Testing of INL and DNL with Non-Linear Code-Domain Excitation

Unlike the conventional approach of using highly precise input stimuli, multiple input signals that are of low accuracy but that are highly repeatable and practically and easily realizable on chip will be used to characterize the ADC. A low accuracy low frequency (spatial) "ramp-like" input signal $\Re 1(t)$ is used as the input to the DUT and the corresponding output histogram, H1, is obtained. A related input signal $\Re 2(t)$ obtained by shifting $\Re 1$ by a fixed amount is presented as a second input and histogram H2 is obtained at the output. These two histograms are then used to predict the A/D converter non-linearity. In the context of this work, the term "low accuracy" input means that we will assume we do not know precisely what the input waveform is. It is also assumed that the amount of shift of the input waveform is not precisely known and the shifted waveform may have modestly different nonlinearities than the original ramp-like waveform. What is implicit, however, in this work is the assumption that we can generate this "ramp-like" signal practically on silicon. Details about the preliminary results of using this approach to develop a testing algorithm are given in [19, 20, 21].

Figure.4.4 depicts the transfer-characteristics of the input excitation signal. The vertical axis corresponds to the input voltage to the ADC designated as V_{sig} in Figure.4.4 and the horizontal axis corresponds to time. Overlaid on the vertical axis are the end point fit line and the trip points of the flash converter denoted as I_i and T_i respectively. The number of tallies for code C_i, $C(i)$, relates to the accumulated Tally function $F(x)$ by

$$C(i) = F(T_i) - F(T_{i-1}) \tag{4.32}$$

It follows from (4.19) that

$$C(i) = (t_i - t_{i-1})\frac{N_{SAMP}}{T_{SIG}} \tag{4.33}$$

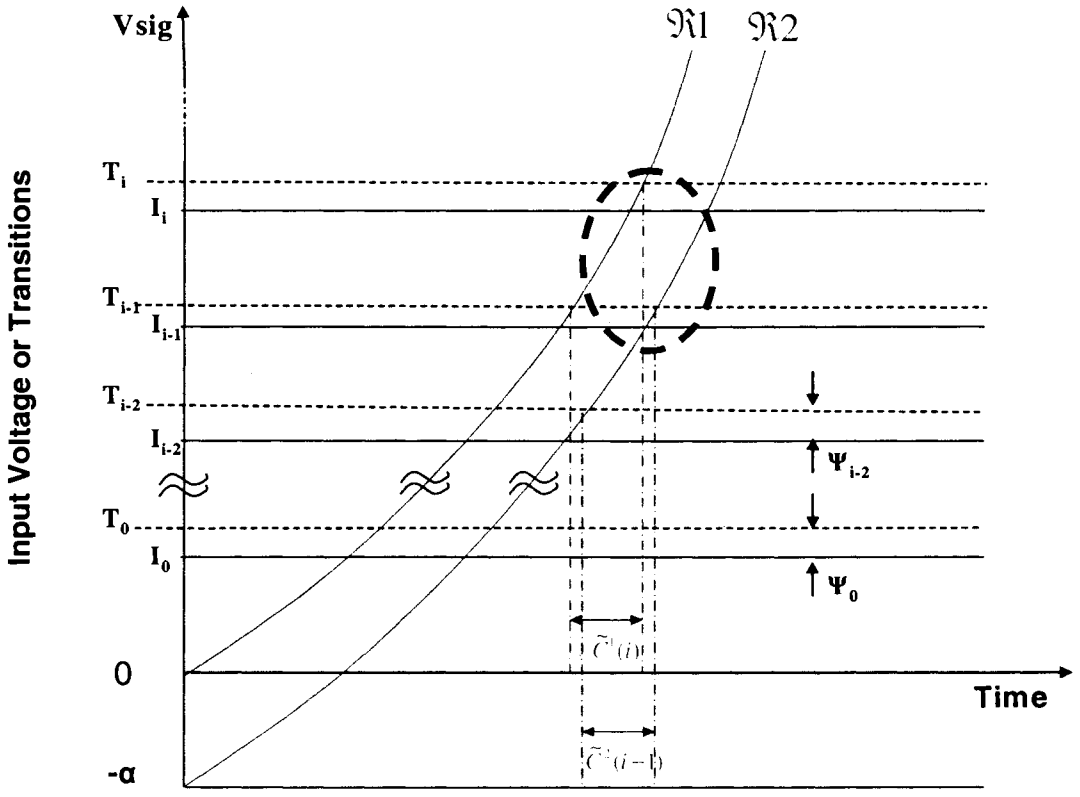


Figure.4.4 Transfer Characteristics of the Input Excitation Signal

The number of samples obtained in each code bin at the output for $\Re 1$ and $\Re 2$ can be obtained by projection onto the horizontal axis as denoted by the intervals $\tilde{C}^1(i)$ and $\tilde{C}^2(i)$ respectively. The number of tallies for code C_i for both $\Re 1$ and $\Re 2$ are proportional to the length of the intervals $\tilde{C}^1(i)$ and $\tilde{C}^2(i)$ respectively and are given by the expressions:

$$C^1(i) = \frac{\tilde{C}^1(i)}{\sum_{k=1}^{N-2} \tilde{C}^1(k)} * (N-2)\overline{C} \tag{4.34}$$

$$C^2(i) = \frac{\tilde{C}^2(i)}{\sum_{k=1}^{N-2} \tilde{C}^2(k)} * (N-2)\overline{C} \tag{4.35}$$

In what follows, it is assumed that $\Re 2$ is shifted down with respect to $\Re 1$ by a fixed but unknown amount α. The various variables that are used in the figure and their relationship with tally parameters appear in Table 4.1.

Table 4.1 Variables and their description

Variables	**Description**
I_i	Ideal i[th] transition level (trip point) of end-point fit through first and last transition points
T_i	i[th] transition level of ADC
Ψ_i	Deviation of T_i from I_i
$\Re 1$	"ramp-like" input signal
$\Re 2$	$\Re 1$ shifted down by α
$\tilde{C}^1(i)$	Width of time interval where $\Re 1$ presents output code C_i
$\tilde{C}^2(i)$	Width of time interval where $\Re 2$ presents output code C_i
$C^1(i)$	Tally of code C_i with $\Re 1$ as input
$C^2(i)$	Tally of code C_i with $\Re 2$ as input

Define the code width for an ADC output code 'C_i' as:

$$\hat{C}(i) = T_i - T_{i-1} \tag{4.36}$$

where T_i and T_{i-1} are the i^{th} and $(i-1)^{th}$ transition points respectively. Any input signal to the ADC between T_i and T_{i-1} will always give the same output code c_i. Ideally, every code width should be of 1LSB size. In reality, however, the code width deviates from 1LSB by some error, denoted as the DNL as was defined in (4.29). The trip points of the actual ADC can be written in terms of the trip points of the fit line through the first and last trip points as:

$$T_i = I_i + \Psi_i \qquad i = 0,1,2,...,N-2 \tag{4.37}$$

where the Ψ_i's denote the deviations from the fit line.

The ideal trip points of the fit line are defined by the expression

$$I_i = T_0 + i\frac{(T_{N-2} - T_0)}{N-2} \qquad 0 \le i \le N-2 \tag{4.38}$$

If we define the LSB to be

$$\bar{I} = \frac{T_{N-2} - T_0}{N-2} \tag{4.39}$$

it follows that

$$I_i = T_0 + i\bar{I} \qquad 0 \le i \le N-2 \tag{4.40}$$

From (4.37) and (4.38), it is apparent that

$$\Psi_0 = \Psi_{N-2} = 0 \tag{4.41}$$

The "trip point" INL in LSB at the remaining trip points can be defined as

$$INL_i = \frac{T_i - I_i}{\bar{I}} = \frac{\Psi_i}{\bar{I}} \qquad 1 \le i \le N-3 \tag{4.42}$$

If we normalized to the LSB by defining the variable, $\overline{\Psi_i}$, to be

$$\overline{\Psi_i} = \frac{\Psi_i}{\bar{I}} \qquad 1 \le i \le N-3 \tag{4.43}$$

it follows

$$INL_i = \overline{\Psi_i} \qquad 1 \le i \le N-3 \tag{4.44}$$

Thus the normalized Ψ_i values give directly the "trip point" INL and if only the INL is desired, we must only determine the $\overline{\Psi_i}$ values. Note the trip point INL defined by (4.42) is actually the standard definition of INL extended to the value at the trip point.

Our goal in identifying the system is to determine the deviations of the ADC under test from those of the end point fit line. In terms of the notation used, the goal is to determine the sequence $< \Psi_0, \Psi_1, \Psi_2, \ldots, \Psi_{N-2} >$.

Although it may appear that there are N-1 unknowns in this set, since the fit line passes through T_0 and T_{N-2}, it was observed in (4.41) that $\Psi_0 = \Psi_{N-2} = 0$. So there are actually N-3 unknowns.

It should be noted that this goal has been established for the specific purpose of determining the INL and DNL of the DUT. If gain and offset errors are also needed, the goal must be extended.

We will restrict our focus to INL and DNL errors because we believe one of the major challenges in testing an ADC rests with determining the INL and the DNL.

Assume the first input signal to the ADC is $x = \Re 1(t)$. The exact nature of the function is not important as long as it is continuously differentiable and its first derivative varies slowly with time. An input with these characteristics is not difficult to achieve practically.

Define the function $f_1(u)$ by

$$f_1(u) = \Re_1(u\frac{T_{SIG}}{N_{SAMP}}) \tag{4.45}$$

It follows from (4.45) and (4.20) that the function $f_1(u)$ maps the accumulated tally values to the ADC transition points by the equation:

$$f_1(\sum_{j=0}^{i} C^1(j)) \cong T_i \qquad i = 0,1,2,...,N-2 \tag{4.46}$$

The "$\cong$" relationship rather than the "=" relationship is indicated in (4.46) because the finite number of samples in each code bin introduces a quantization error that is of the order of magnitude of $\bar{I}/\overline{C}$ LSB. Since $\overline{C}$ is typically much larger than 1, the error is generally a small fraction of an LSB. We will neglect this error throughout the remainder of this chapter.

Substituting (4.37) in (4.46), we get:

$$f_1(\sum_{j=0}^{i} C^1(j)) = I_i + \Psi_i \qquad i = 0,1,2,...,N-2 \tag{4.47}$$

Equation (4.47) is a set of N-1 equations with 2N-2 unknowns; the unknowns of interest, $\{\Psi_1, \Psi_2, \ldots, \Psi_{N-3}\}$, the unknowns T_0 and T_{N-2} in (4.39) and the N-1 unknowns values of f_1. Although the arguments of f_1 are known, without knowing the functional form of f_1, the equations of (4.47) can not be solved to obtain the Ψ values.

Using a second mapping function $x = \Re 2(t)$ and defining the function $f_2(u)$ by

$$f_2(u) = \Re_2(u\frac{T_{SIG}}{N_{SAMP}}) \tag{4.48}$$

we get another set of similar equations for the second input signal and its corresponding histogram values.

$$f_2(\sum_{j=0}^{i} C^2(j)) = I_i + \Psi_i \qquad i = 0,1,2,\ldots,N-2 \tag{4.49}$$

This second set of N-1 equations has resulted in the introduction of an additional N variable, the N-1 values of the function f_2 and the shift parameter α. The 2(N-1) equations of (4.47) and (4.26) with 3N-2 unknowns remains highly under-constrained.

If the second input signal, $\Re 2(t)$, is obtained by shifting the first signal $\Re 1$ by a constant value, α, we have the following N-1 equations relating the code densities:

$$f_2(\sum_{j=0}^{i} C^2(j)) = f_1(\sum_{j=0}^{i} C^2(j)) - \alpha \qquad i = 0,1,2,\ldots,N-2 \tag{4.50}$$

No additional unknowns have been introduced by these N-1 additional equations. We thus have a total of 3N-3 equations and 3N-2 unknowns. Before addressing methods of solving these equations in general, we will consider a simpler special case.

If the functions f_1 and f_2 in the accumulated tallies are linear, that is,

$$f_1(\sum_{j=0}^{i} C^1(j)) = h_0 \sum_{j=0}^{i} C^1(j) \tag{4.51a}$$

$$f_2(\sum_{j=0}^{i} C^2(j)) = h_1 \sum_{j=0}^{i} C^2(j) \tag{4.51b}$$

where h_0 and h_1 are constants, it is trivial to obtain the values of Ψ_i from either equation (4.47) or (4.49).

If the input is nonlinear in time, the task of solving for the Ψ_i values is more complicated. However, by knowing the functional relationship between f_1 and f_2, it is possible to obtain good estimates of the Ψ values. With this strategy in mind, subtracting the $(i-1)^{th}$ equation in (4.49) from the i^{th} equation in (4.47), we get:

$$f_1(\sum_{j=0}^{i} C^1(j)) - f_2(\sum_{j=0}^{i-1} C^2(j)) = \bar{I} + \Psi_i - \Psi_{i-1} \qquad i = 1,2,3,\ldots N-2 \tag{4.52}$$

where $I_i - I_{i-1} = \bar{I}$ is the ideal LSB for the fit line as defined in (4.39). Equation (4.52) relates the nonlinearities and histogram samples of adjacent bins. If the input signal is assumed to be "ramp-like" with just low frequency nonlinearities, then for a small segment of the input (ranging between adjacent bins), the input can be assumed to be linear. An approximation of the left hand side of (4.52) will now be obtained. An expanded version of the circled section of Figure.4.4 using the functions f_1 and f_2 instead of $\Re 1$ and $\Re 2$ is shown in Figure.4.4b.

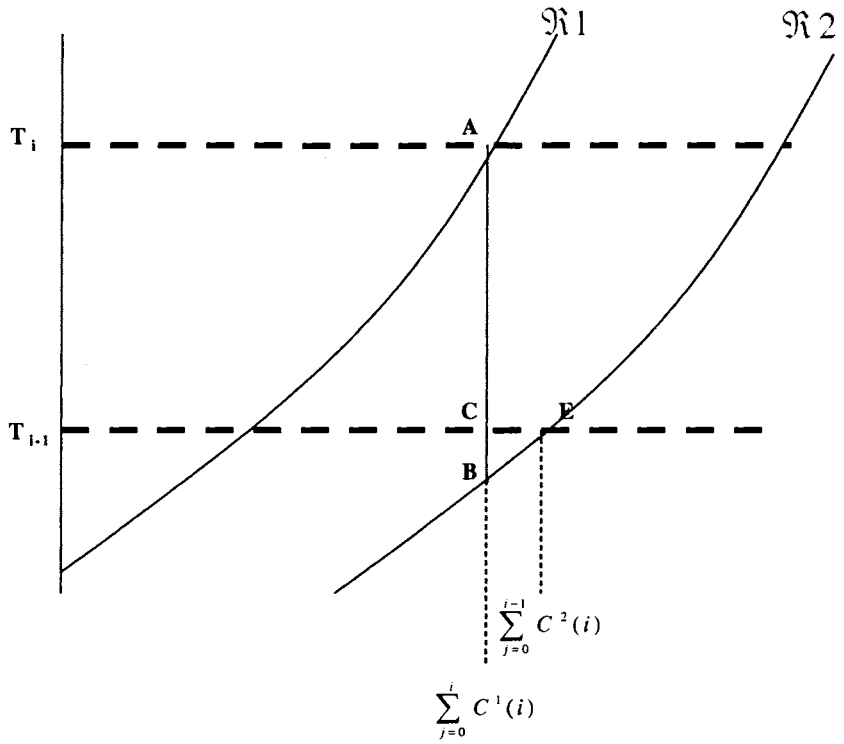


Figure.4.4b Expanded Transfer Characteristics

In this figure, the distance AB is α and the distance AC is the left hand side of (4.52). It thus follows that

$$\overline{AC} + \frac{df_2}{dt}(\xi_i)\overline{CE} \cong \alpha \tag{4.53}$$

where ξ_i is any point close to $\sum_{j=0}^{i} C^1(j)$. Equation (4.53) can be expressed as

$$f_1(\sum_{j=0}^{i} C^1(j)) - f_2(\sum_{j=0}^{i-1} C^2(j)) \cong f_2'(\xi_i)(\sum_{j=0}^{i} C^1(j) - \sum_{j=0}^{i-1} C^2(j)) + \alpha \qquad i = 1,2,3,....N-2 \tag{4.54}$$

An approximation of $f_2'(\xi_i)$ in the neighborhood of $\xi_i = \sum_{j=1}^{i-1} C^2(j)$ is needed. Since we have assumed that $\Re 2(t)$ is slowly varying, there are many ways $f_2'(\xi_i)$ can be estimated. One such estimate of this derivative will be given. If α is sufficiently small so that the derivative of $f_1\left(\sum_{j=0}^{i} C^1(j)\right)$ and $f_2\left(\sum_{j=0}^{i-1} C^2(j)\right)$ do not differ much, then

$$f_2'(\xi_i) \cong \frac{f_1'\left(\sum_{j=0}^{i} C^1(j)\right) + f_2'\left(\sum_{j=0}^{i-1} C^2(j)\right)}{2} \tag{4.55}$$

The derivatives of f_1 and f_2 can be approximated by

$$f_1'\left(\sum_{j=0}^{i} C^1(j)\right) \cong \frac{T_i - T_{i-1}}{C^1(i)} \tag{4.56}$$

and

$$f_2'\left(\sum_{j=0}^{i-1} C^2(j)\right) \cong \frac{T_i - T_{i-1}}{C^2(i)} \tag{4.57}$$

But

$$T_i - T_{i-1} = \bar{I} + \Psi_i - \Psi_{i-1} \tag{4.58}$$

so we obtain the approximation

$$f_2'(\xi_i) \cong \frac{1}{2}\left(\frac{\bar{I} + \Psi_i - \Psi_{i-1}}{C^1(i)} + \frac{\bar{I} + \Psi_i - \Psi_{i-1}}{C^2(i)}\right) \qquad i = 1,2,3,\ldots N-2 \tag{4.59}$$

If α is large and there are substantial third or higher order nonlinearities present, this approximation will not be too good. This approximation is very good, even with large α, however, if third and higher order nonlinearities are not substantial.

Each pair of equations in (4.47) and (4.49) have three unknowns, $f_1(\sum_{j=0}^{i} C^1(j))$, $f_2(\sum_{j=0}^{i} C^2(j))$ and Ψ_i and the set of equations have the common unknown $\bar{I}$. By pair-wise combining these equations and by replacing the difference between the f_1 and f_2 functions by the derivate expression of (4.54) and (4.59), a single set of N-2 equations in the N-1 unknowns Ψ_1, Ψ_2 ,...., Ψ_{N-3} , α and $\bar{I}$ is obtained. From (4.52), (4.54) and (4.59) it follows that

$$\left(\frac{\bar{I}+\Psi_i-\Psi_{i-1}}{2}\right)\left(\frac{1}{C^1(i)}+\frac{1}{C^2(i)}\right)\left[\sum_{j=0}^{i} C^1(j)-\sum_{j=0}^{i} C^2(j)\right]+\alpha=\bar{I}+\Psi_i-\Psi_{i-1} \qquad (4.60)$$

Since it has been assumed that f_1 has only slowly varying components, this approximation is reasonably good. The above equation can be rewritten as follows:

$$\bar{I}=\frac{1}{1-\gamma_i}\alpha+\Psi_{i-1}-\Psi_i \qquad i=1,2,3,...,N-2 \qquad (4.61)$$

where,

$$\gamma_i=\frac{1}{2}\left(\frac{1}{C^1(i)}+\frac{1}{C^2(i)}\right)\left(\sum_{j=0}^{i} C^1(j)-\sum_{j=0}^{i-1} C^2(j)\right) \qquad (4.62)$$

If we normalize α and the Ψ_i values to 1LSB by defining

$$\overline{\alpha}=\frac{\alpha}{\bar{I}} \qquad (4.63)$$

and,

$$\overline{\Psi}_i = \frac{\Psi_i}{\bar{I}} \tag{4.64}$$

(4.61) can be rewritten as

$$1 = \frac{\overline{\alpha}}{1-\gamma_i} + \overline{\Psi}_{i-1} - \overline{\Psi}_i \qquad i = 1,2,3,...,N-2 \tag{4.65}$$

In the normalized variable domain there are now only N-2 unknowns, $<\overline{\Psi}_1, \overline{\Psi}_2, ...\overline{\Psi}_{N-3}, \overline{\alpha}>$ and N-2 equations. It follows from (4.44) that all that is needed to obtain the INL is the normalized Ψ_i values, $\overline{\Psi}_i$. It remains to solve the N-2 linear equations in (4.61) for the $\overline{\Psi}_i$ values.

4.6.3. Preliminary Work on Determining $\overline{\Psi}_i$ and $\overline{\alpha}$

In the original formulation of this problem I only considered the last N-3 equations in (4.47) and (4.49). With this formulation, only the last N-3 equations in (4.61) were obtained. With the need for obtaining N-2 variables, I was one linear equation short of what is needed for a closed-form solution. In what follows, I will describe a solution we obtained. The more general solution which is based upon all equations in (4.61) will be discussed in Section 4.6.4. Thus, I will assume we have the N-3 linear equations

$$1 = \frac{\overline{\alpha}}{1-\gamma_i} + \overline{\Psi}_{i-1} - \overline{\Psi}_i \qquad i = 2,3,...,N-2 \tag{4.66}$$

It can be seen that the equation corresponding to i=1 in (4.47) has not been used in the set of equations derived in (4.66). This equation in conjunction with the case when i=2 in (4.49) results in the equation

$$f_2\left(C^2(1)+C^2(2)\right)-f_1\left(C^1(1)\right)=\bar{I}+\Psi_2-\Psi_1 \tag{4.67}$$

(4.67) is then reduced to a linear equation using the same method explained earlier. Specifically, the left hand side of (4.67) can be approximated by a derivative evaluated at some ξ_i in the neighborhood of $C^1(1)$ as

$$f_1'(\xi)\left(C^2(1)+C^2(2)-C^1(1)\right)=\bar{I}+\Psi_2-\Psi_1+\alpha \tag{4.68}$$

The derivative can be approximated by the expression

$$f_1'(\xi)=\frac{1}{2}\left[\frac{\bar{I}+\Psi_1}{C^1(1)}+\frac{\bar{I}+\Psi_2-\Psi_1}{C^2(2)}\right] \tag{4.69}$$

Substituting (4.69) in (4.68), we get:

$$\alpha+(\beta_2-\beta_1-1)\Psi_1+(1-\beta_2)\Psi_2=(\beta_2+\beta_1-1)\bar{I} \tag{4.70}$$

where,

$$\begin{aligned}\beta_1&=\frac{1}{2}\left[\frac{C^2(1)+C^2(2)-C^1(1)}{C^1(1)}\right]\\ \beta_2&=\frac{1}{2}\left[\frac{C^2(1)+C^2(2)-C^1(1)}{C^2(2)}\right]\end{aligned} \tag{4.71}$$

This can also be written in normalized form by dividing by $\bar{I}$ to obtain

$$\overline{\alpha} + (\beta_2 - \beta_1 - 1)\overline{\Psi}_1 + (1 - \beta_2)\overline{\Psi}_2 = (\beta_2 + \beta_1 - 1) \tag{4.72}$$

If this linear equation (4.72) is added to the N-3 equations of (4.66), we obtain a set of N-2 linear equations in N-2 unknowns. It remains to solve these N-2 linear equations. Two methods of solving these equations will now be discussed.

Matrix Inversion Approach

The set of N-2 equations, the equations of (4.66) along with the equation (4.72) can be arranged in a matrix as shown below:

$$\begin{bmatrix} 1 \\ 1 \\ : \\ 1 \\ \beta_2 + \beta_1 - 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{1-\gamma_2} & 1 & -1 & .. & 0 \\ \frac{1}{1-\gamma_3} & 0 & 1 & .. & 0 \\ : & : & : & .. & : \\ \frac{1}{1-\gamma_{N-2}} & 0 & 0 & : & 1 \\ 1 & \beta_2 - \beta_1 - 1 & 1-\beta_2 & .. & 0 \end{bmatrix} \begin{bmatrix} \overline{\alpha} \\ \overline{\Psi}_1 \\ : \\ \overline{\Psi}_{N-4} \\ \overline{\Psi}_{N-3} \end{bmatrix} \tag{4.73}$$

or,

$$M_2 \quad = \quad M_1 \quad \overline{\Psi}$$

where M_2, M_1 and $\overline{\Psi}$ refer to matrices as indicated above. Multiplying both sides by M_1^{-1} we obtain:

$$\overline{\Psi} = M_1^{-1} M_2 \tag{4.74}$$

Simulation results using the above method to characterize the ADC were found to be very good. The transitions of the ADC were identified to a much higher accuracy than the resolution of the DUT. Sections 4.7 & 4.8 give more detail on the simulation setup and the results obtained.

However, as seen from equation (4.74), the solution to the set of linear equations requires a 'matrix inversion' step. The computational complexity associated with a standard matrix inversion algorithm for ADC resolution in excess of 9~10 bits is significant. Although the simulation results we will present focus on a flash converter architecture and although flash converters with resolution in excess of 10-bits are uncommon, the algorithms presented here should be applicable to other structures as well. As such, methods of reducing the computational requirements deserve consideration.

The matrix M_1 in (4.73) is very sparse and, except for the first and last rows and the first column, has no entries far away from the main diagonal. There are often much simpler methods for inverting a dominantly diagonal matrix.

Non Matrix Inversion Approach (Alpha Estimation Approach)

On closer look into (4.61), it can be seen that if the value of $\overline{\alpha}$ can be identified independently, then a much simpler set of equations is obtained as explained below. Let us consider equation (4.73) again. By replacing the equation in the first row with the sum of the first N-3 row equations, by replacing the second row with the sum of the equation in this row through that of the N-2 row, etc, we obtain an alternate set of equations. The resultant equation set can be re-formulated as shown below:

$$\begin{bmatrix} N-3 \\ N-4 \\ \vdots \\ 1 \\ \beta_2+\beta_1-1 \end{bmatrix} = \begin{bmatrix} \sum_{i=2}^{N-2}\frac{1}{1-\gamma_i} & 1 & 0 & .. & 0 \\ \sum_{i=3}^{N-2}\frac{1}{1-\gamma_i} & 0 & 1 & .. & 0 \\ \vdots & \vdots & \vdots & .. & \vdots \\ \frac{1}{1-\gamma_{N-2}} & 0 & 0 & : & 1 \\ 1 & \beta_2-\beta_1-1 & 1-\beta_2 & .. & 0 \end{bmatrix} \begin{bmatrix} \overline{\alpha} \\ \overline{\Psi}_1 \\ \vdots \\ \overline{\Psi}_{N-4} \\ \overline{\Psi}_{N-3} \end{bmatrix} \tag{4.75}$$

Writing (4.75) as a set of equations, we get:

$$\begin{aligned} N-3 &= \sum_{i=2}^{N-2}\frac{1}{1-\gamma_i} * \overline{\alpha} + \overline{\Psi}_1 \\ N-4 &= \sum_{i=3}^{N-2}\frac{1}{1-\gamma_i} * \overline{\alpha} + \overline{\Psi}_2 \\ &\vdots \\ &\vdots \\ \beta_2+\beta_1-1 &= \overline{\alpha} + (\beta_2-\beta_1-1) * \overline{\Psi}_1 + (1-\beta_2) * \overline{\Psi}_2 \end{aligned} \tag{4.76}$$

If the value of $\overline{\alpha}$ can be estimated by some alternate means, then the equations in (4.76), expect for the last equation, can be re-arranged as follows:

$$\begin{bmatrix} N-3 \\ N-4 \\ \vdots \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} \sum_{i=2}^{N-2}\frac{1}{1-\gamma_i} \\ \sum_{i=3}^{N-2}\frac{1}{1-\gamma_i} \\ \vdots \\ \vdots \\ \frac{1}{1-\gamma_{N-2}} \end{bmatrix} * \overline{\alpha} + \begin{bmatrix} \overline{\Psi}_1 \\ \overline{\Psi}_2 \\ \vdots \\ \vdots \\ \overline{\Psi}_{N-3} \end{bmatrix} \tag{4.77}$$

or equivalently as

$$\begin{bmatrix} \overline{\Psi}_1 \\ \overline{\Psi}_2 \\ \vdots \\ \vdots \\ \overline{\Psi}_{N-3} \end{bmatrix} = \begin{bmatrix} N-3 \\ N-4 \\ \vdots \\ \vdots \\ 1 \end{bmatrix} - \begin{bmatrix} \sum_{i=2}^{N-2} \frac{1}{1-\gamma_i} \\ \sum_{i=3}^{N-2} \frac{1}{1-\gamma_i} \\ \vdots \\ \vdots \\ \frac{1}{1-\gamma_{N-2}} \end{bmatrix} * \overline{\alpha} \tag{4.78}$$

The computational complexity of solving (4.78) is proportional to the number of equations (N-3), in contrast to the complexity for matrix inversion which is of the order of $(N-2)^3$. The solution of (4.78) is more attractive from the complexity viewpoint provided $\overline{\alpha}$ can be determined by some alternate method with sufficient accuracy. It is necessary that $\overline{\alpha}$ be determined accurately because the coefficient of $\overline{\alpha}$ in (4.78) may be quite large, and any error in $\overline{\alpha}$ estimation will be amplified by a large number and will appear in the $\overline{\Psi}$'s. In what follows, an algorithm for estimating $\overline{\alpha}$ in which we use the information stored in $C^1(i)$ and $C^2(i)$ will be described.

'$\overline{\alpha}$' Estimation

Summing all the equations in (4.61), we get

$$N-3 = \overline{\alpha} \sum_{i=2}^{N-2} \frac{1}{1-\gamma_i} + \overline{\Psi}_1 \tag{4.79}$$

Approximating $\overline{\Psi}_1 = 0$, we get:

$$\overline{\alpha} = \frac{N-3}{\sum_{i=2}^{N-2} \frac{1}{1-\gamma_i}} \tag{4.80}$$

It should be observed that the error in $\overline{\alpha}$ by assuming $\overline{\Psi}_1 = 0$ is very small. If $\overline{\Psi}_1$ were as large as 1LSB, a highly unlikely scenario since the fit line goes through T_0, the error in $\overline{\alpha}$ would be only approximately 1 part in N of an LSB. The approximation of $\overline{\alpha}$ by the above method was found to be very close to the actual value. Results using this 'alpha-estimation approach' and the comparison with the 'matrix-inversion method' are presented in Section 4.8.

'$\overline{\alpha}$' Determination

The $\overline{\alpha}$ -approximation strategy appears to introduce a small error in the measurement of $\overline{\alpha}$ but simulation results suggest the algorithm is somewhat sensitive to $\overline{\alpha}$ mismatch. If we consider the first, second and last equations in (4.76), it can be observed that these 3 equations contain only the variables $\overline{\alpha}$, $\overline{\Psi}_1$ and $\overline{\Psi}_2$. Since these are assumed to be independent linear equations, they can be solved for $\overline{\alpha}$ yielding

$$\overline{\alpha} = \frac{\beta_1 (N-4)}{(\beta_2 - \beta_1 - 1)\sum_{i=2}^{N-2}\left(\frac{1}{1-\gamma_i}\right) + (1-\beta_2)\sum_{i=3}^{N-2}\left(\frac{1}{1-\gamma_i}\right) - 1} \tag{4.81}$$

Ideally, determining $\overline{\alpha}$ this way should give the same overall results as were obtained with the direct matrix inversion approach provided numerical round-off errors are kept adequately small.

4.6.4. Alternative Method for Determining $\overline{\Psi}_i$

The complexity in the matrix inversion algorithm discussed in the previous section was primarily due to the last row in the matrix of (4.73). If we consider now the set of N-2 equations described by (4.65), repeated as

$$1 = \frac{\overline{\alpha}}{1-\gamma_i} + \overline{\Psi}_{i-1} - \overline{\Psi}_i \qquad i = 1,2,3,...,N-2 \tag{4.82}$$

It follows by adding these N-2 equations that all $\overline{\Psi}_i$ disappear resulting in the simple linear equation

$$N-2 = \overline{\alpha}\sum_{i=1}^{N-2}\frac{1}{1-\gamma_i} \tag{4.83}$$

from which we obtain $\overline{\alpha}$ as

$$\overline{\alpha} = \frac{N-2}{\sum_{i=1}^{N-2}\frac{1}{1-\gamma_i}} \tag{4.84}$$

Once $\overline{\alpha}$ is obtained, we obtain (since $\overline{\Psi}_0 = 0$)

$$\overline{\Psi}_1 = \frac{\overline{\alpha}}{1-\gamma_1} - 1 \tag{4.85}$$

and the recursive relationship

$$\overline{\Psi}_i = \overline{\Psi}_{i-1} + \frac{\overline{\alpha}}{1-\gamma_i} - 1 \qquad 2 < i \le N-3 \tag{4.86}$$

The computational complexity needed to obtain the $\overline{\Psi}_i$ values is minimal. For completeness, we will represent the solution of these equations in matrix form. In matrix form, (4.82) becomes

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ : \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{1-\gamma_1} & 1 & 0 & 0 & : & 0 & 0 \\ \frac{1}{1-\gamma_2} & 1 & -1 & 0 & : & 0 & 0 \\ \frac{1}{1-\gamma_3} & 0 & 1 & -1 & : & 0 & 0 \\ : & : & : & : & : & : & : \\ : & : & : & : & : & 1 & -1 & 0 \\ : & : & : & : & : & 0 & 1 & -1 \\ \frac{1}{1-\gamma_{N-2}} & 0 & 0 & 0 & : & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \overline{\alpha} \\ \overline{\Psi}_1 \\ \overline{\Psi}_2 \\ : \\ : \\ : \\ \overline{\Psi}_{N-3} \end{bmatrix} \tag{4.87}$$

If we now replace row 'k' for $1 \le k \le N-2$ with the sum of all rows below it, we obtain the alternate set of equations

$$\begin{bmatrix} N-2 \\ N-3 \\ : \\ : \\ : \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} \sum_{i=2}^{N-2} \frac{1}{1-\gamma_i} & 0 & 0 & 0 & : & 0 & 0 & 0 \\ \sum_{i=3}^{N-2} \frac{1}{1-\gamma_i} & 1 & 0 & 0 & : & 0 & 0 & 0 \\ : & : & : & : & : & : & : & : \\ : & : & : & : & : & : & : & : \\ : & : & : & : & : & : & : & : \\ \sum_{i=N-3}^{N-2} \frac{1}{1-\gamma_i} & : & : & : & : & 0 & 1 & 0 \\ \frac{1}{1-\gamma_{N-2}} & 0 & 0 & 0 & : & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \overline{\alpha} \\ \overline{\Psi}_1 \\ \overline{\Psi}_2 \\ : \\ : \\ : \\ \overline{\Psi}_{N-3} \end{bmatrix} \tag{4.88}$$

The first row in this matrix represents a single equation that can be solved for $\overline{\alpha}$ as

$$\overline{\alpha} = \frac{N-2}{\sum_{i=2}^{N-2} \frac{1}{1-\gamma_i}} \tag{4.89}$$

The remaining N-3 equations can be expressed as

$$\begin{bmatrix} N-3 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} \sum_{i=2}^{N-2} \frac{1}{1-\gamma_i} \\ \sum_{i=3}^{N-2} \frac{1}{1-\gamma_i} \\ \vdots \\ \vdots \\ \sum_{i=N-3}^{N-2} \frac{1}{1-\gamma_i} \\ \frac{1}{1-\gamma_{N-2}} \end{bmatrix} \overline{\alpha} + \begin{bmatrix} 1 & 0 & : & : & 0 & 0 \\ 0 & 1 & : & : & 0 & 0 \\ : & : & : & : & : & : \\ : & : & : & : & : & : \\ 0 & 0 & : & : & 1 & 0 \\ 0 & 0 & : & : & 0 & 1 \end{bmatrix} \begin{bmatrix} \overline{\Psi}_1 \\ \overline{\Psi}_2 \\ \\ \\ \\ \overline{\Psi}_{N-3} \end{bmatrix} \tag{4.90}$$

But since the square matrix is the identity matrix, it follows that

$$\begin{bmatrix} \overline{\Psi}_1 \\ \overline{\Psi}_2 \\ \\ \\ \\ \overline{\Psi}_{N-3} \end{bmatrix} = \begin{bmatrix} N-3 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ 1 \end{bmatrix} - \begin{bmatrix} \sum_{i=2}^{N-2} \frac{1}{1-\gamma_i} \\ \sum_{i=3}^{N-2} \frac{1}{1-\gamma_i} \\ \vdots \\ \vdots \\ \sum_{i=N-3}^{N-2} \frac{1}{1-\gamma_i} \\ \frac{1}{1-\gamma_{N-2}} \end{bmatrix} \overline{\alpha} \tag{4.91}$$

from which we find the solution for $\overline{\Psi}_k$, $1 \le k \le N-3$ as

$$\overline{\Psi}_k = N-k-2-\left(\sum_{i=k+1}^{N-2} \frac{1}{1-\gamma_i}\right)\overline{\alpha} \qquad 1 \le k \le N-3 \tag{4.92}$$

A comparison of (4.89) with (4.84) confirms both approaches give the same value for $\overline{\alpha}$. The equivalence of the recursive relationships of (4.85) and (4.86) to the direct solution of (4.92) is less apparent but it can be shown that both forms of the solution are identical.

4.7. Modeling of the Test Setup

To validate the solutions discussed in the previous section, the system involving the input signal generator and the A/D converter was modeled in MATLAB. The Flash ADC was assumed to be a basic structure comprising of a resistor-string and a comparator array as shown in Figure 4.5. The main source of errors in such a flash ADC are the resistor variations and the comparator offsets. In this work, the non-idealities of the converter were restricted to the resistor variations but the concept can be extended to include the comparator offsets as well.

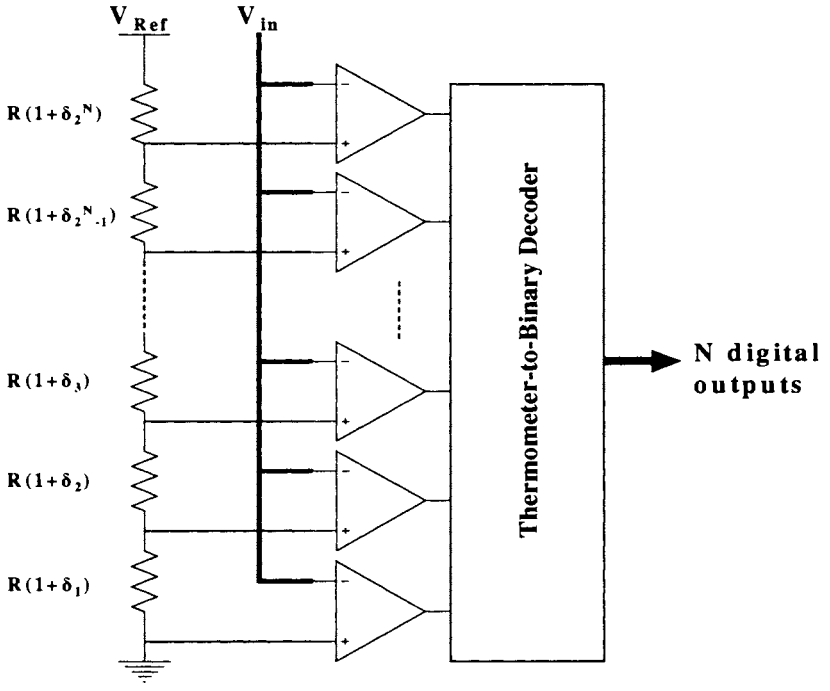


Figure.4.5 Flash A/D Converter

Each of the resistors was modeled as:

$$R_i = R_0(1+\delta_i) \qquad i = 1,2,\cdots,2^N \tag{4.93}$$

where R_0 is a nominal value of the resistance and the random variable δ_i represents the resistance deviation of the i^{th} resistor from the nominal value and was assumed to be uniformly distributed in the interval [-0.5,0.5]. It was further assumed that all such random variables were uncorrelated. For notational convenience, layout gradient effects were not included in the model but we do not believe the gradient effects will substantially impact the performance of the testing algorithms. The large variance used in the resistor-string model will cause an INL of tens of LSBs for 9-10bit ADCs due to the random-walk effect introduced in successive output voltages for the resistor string array.

Having modeled the non-ideal A/D converter, the next requirement is to model the input signals that are used as excitations. (4.94) gives the model of a general input signal,

$$x(t) = s(t) + \sum_{i=1}^{n} a_i f_i(t) \tag{4.94}$$

where s(t) is the desired excitation, $\{f_i: i=1,2,\ldots,n\}$ is a set of basis functions that model the non-linearity in the signal, and $\{a_i\}$ are the weighting coefficients.

To verify the performance of the algorithms, the desired input s(t) was chosen as a linear ramp. The number of basis functions was limited to one resulting in a second-order polynomial used to represent the non-linearity in the excitation. Specifically, the two inputs that were used for the algorithms are given as:

$$x_1(t) = t + a(t-t^2) \text{ and } x_2(t) = t + a(t-t^2) - \alpha \tag{4.95}$$

Both have the same nonlinear term $a(t-t^2)$. The magnitude of the coefficient 'a' determines the accuracy of the input signal. The signal $x_2(t)$ is related to $x_1(t)$ by a fixed difference of α.

4.8. Simulation Results

The following set of simulation results were obtained with the approach described in Section 4.6.4. To emphasize the importance of precise input signal requirements while using the standard algorithms, the following test was first performed. An ideal 10-bit converter was considered and signal of 4-bit linearity (a = -0.05 in (4.95)) was used as input. The trip points of the ADC (or equivalently INL at the trip points) were predicted using both the conventional and newly proposed histogram algorithm. The conventional algorithm was based upon the assumption that the input was linear and the code density INL algorithm of (4.31) was used. The proposed algorithm was based on the model of (4.65) using the matrix inversion algorithm of Section 4.6.4. Since the ADC under consideration is an ideal device, the trip points should be spaced 1LSB apart. If the algorithm works perfectly, it would predict zero INL at the trip points. The results obtained from both the algorithms are shown in Figure 4.6.

It can be seen that the conventional histogram algorithm predicts a maximum INL (at the trip points) of approximately 64 LSBs. This is because, the conventional algorithm computes the linearity characteristics assuming the input signal is a perfect ramp. As the input in this case was just 4-bit linear, we get an equivalent residual error of 6 bits or 64 LSBs. However, the proposed algorithm takes into consideration the non-linearity in the input signal and is thus able to predict the linearity parameters very accurately. From the figure it can be seen that the INL (at the trip points) calculated by the algorithm is nearly zero as expected.

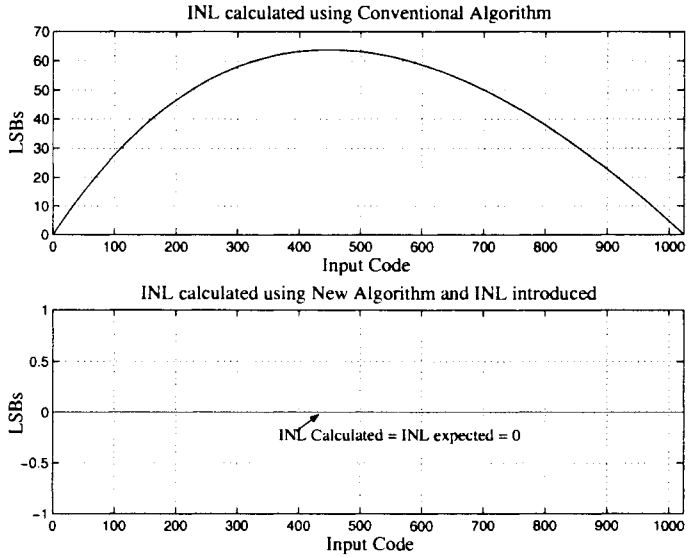


Figure.4.6 Trip point calculation a 10-bit ideal converter using both algorithms

In the above simulation the linearity of the input signal was intentionally restricted to 4 bits. As explained in Section 4.6, although generation of ramp signal of nearly 10-11 bit linearity on chip have been reported in literature, to reduce simulation time we have focused on a 10-bit converter with 4-6 bit linear ramp. The concept can be extended to higher bit converters where a 10-12 bit linear excitation ramp will be a limiting factor. However what needs to be observed is the importance of input signal linearity requirements on the performance of traditional histogram algorithm.

Matrix-Inversion Approach

A non-ideal ADC was then simulated as modeled in Section 4.7. As explained earlier, only the resistor values were modified and the comparators were assumed to be ideal. An 11bit flash converter was considered. An input signal of approximately 3-bit linearity (a = -0.1 in (4.95)) was given to the DUT. The linearity of the input signal can be controlled by changing the value of the coefficient term 'a'. The second signal was obtained by shifting the first signal by 300LSBs (the

amount of shift is not critical). It is to be noted that this value of shift is just used to generate the input signal and is not used in the algorithm. The algorithm assumes 'α' as a variable and it is computed along with all the $\overline{\Psi}$'s as explained in Section 4.6. The average number of samples in each bin (total number of samples obtained divided by the number of bins) was chosen to be 32. The exact value of the number of samples in each bin is not very important. However, the larger the number of samples, the better the results as the slope calculations is more accurate. The effect of the average number of samples on the performance of the algorithm will be discussed later.

The histogram at the output of the device was obtained with the above nonlinear signal and then the matrix-inversion algorithm was implemented to characterize the device. The matlab code for the algorithm is given in Appendix E. The shift 'α' estimated by the matrix-inversion algorithm for this sample run was 299.8908LSB, which is very close to the actual value. Figure.4.7. shows the plot of the actual trip point errors introduced, and those estimated by the algorithm and the resultant error in prediction.

The top portion of Figure 4.7 consists of two plots. One representing the introduced trip point error (by varying the resistor values in matlab) and the other as estimated by the algorithm. From the figure it can be observed that the trip point error of the device ranges from -3~10 LSBs approximately and the estimated value closely matches the actual value. The above INL plot is just a sample plot that is obtained based on the random choice of resistor values in the particular run. Even though it does not necessarily indicate the actual performance of any 11-bit ADC, it gives a good representation of a typical INL of such devices.

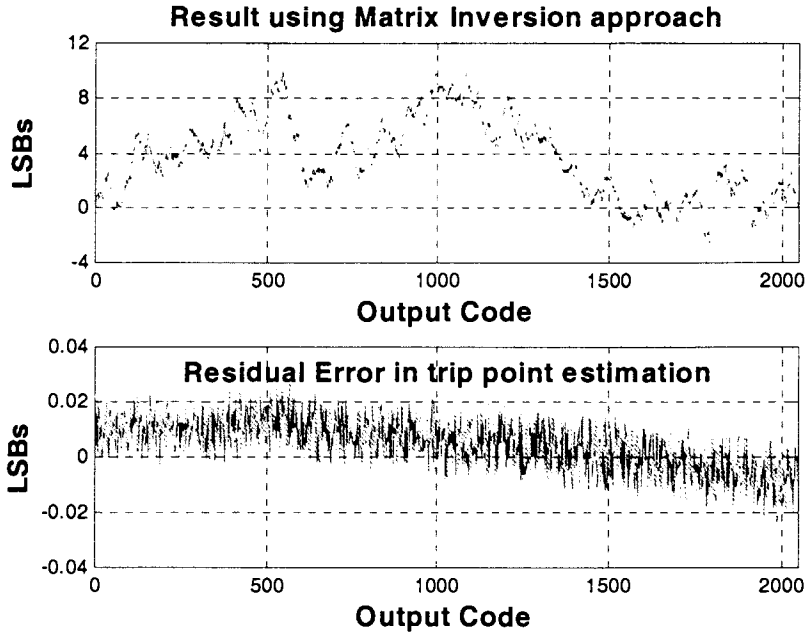


Figure.4.7 Result of matrix-inversion algorithm: Introduced and estimated trip point

The bottom portion of the figure gives the error in trip point prediction obtained by using the matrix-inversion algorithm of Section 4.6.4. It can be seen that the error is less than 0.02LSB. However, what is more important is the fact that with the information that is obtained from the algorithm, by adding some additional calibration (not included in this work) the trip points can be adjusted to approximately 14 bit accuracy. Such good estimation of trip points is possible due to the precise estimation of the relationship between the two signals with the algorithm.

Once all the $\overline{\Psi}$'s (i.e. INL at the trip points or trip point error) are calculated, the calculation of DNL is straight forward. The relationship between INL and DNL for a device is explained in Chapter 2. Figure.4.8. shows the plot of the DNL introduced and DNL estimated. Again the top portion of Figure 4.8 consists of two curves. One corresponding to the introduced DNL values and the other to the calculated DNL values. The error in DNL estimation is shown in the bottom portion of the figure. The magnitude of the error in DNL prediction is less than 0.02LSBs. This is actually limited by the number of samples taken in each bin.

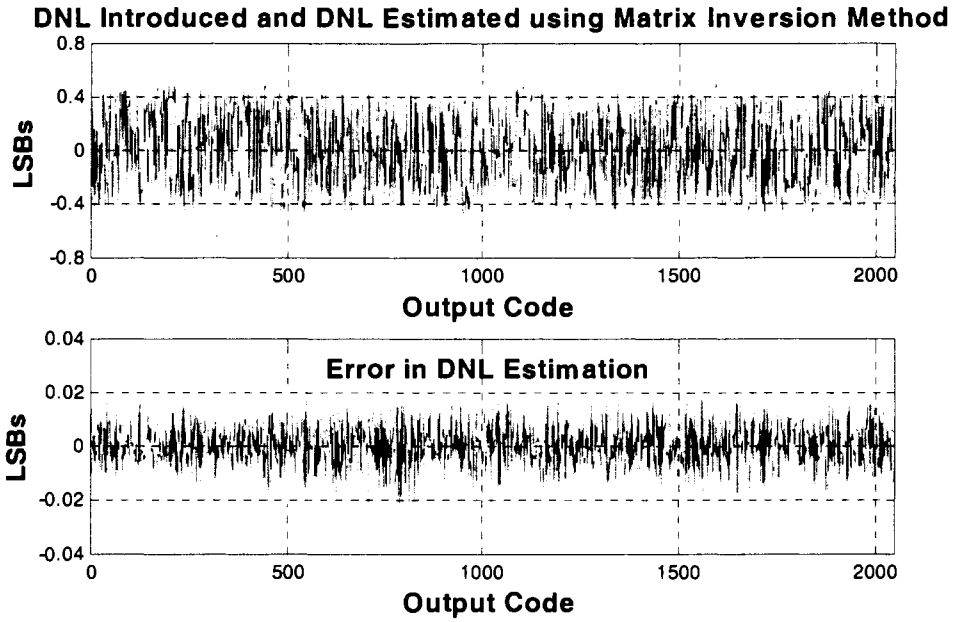


Figure.4.8 Result of matrix-inversion algorithm: Introduced and estimated DNL values for 11-bit converter

The effect of the quantization due to finite number of samples can be seen from the DNL error shown in Figure.4.9. The same converter as the previous simulation was considered again, but the number of samples in each bin was increased from 32 to 200. It can be seen that the DNL error is now limited to less than 0.005LSBs. The reason for the better performance is, as we take more number of samples, we effectively resolve the bins into much smaller portions and hence the trip points can be calculated with more accuracy.

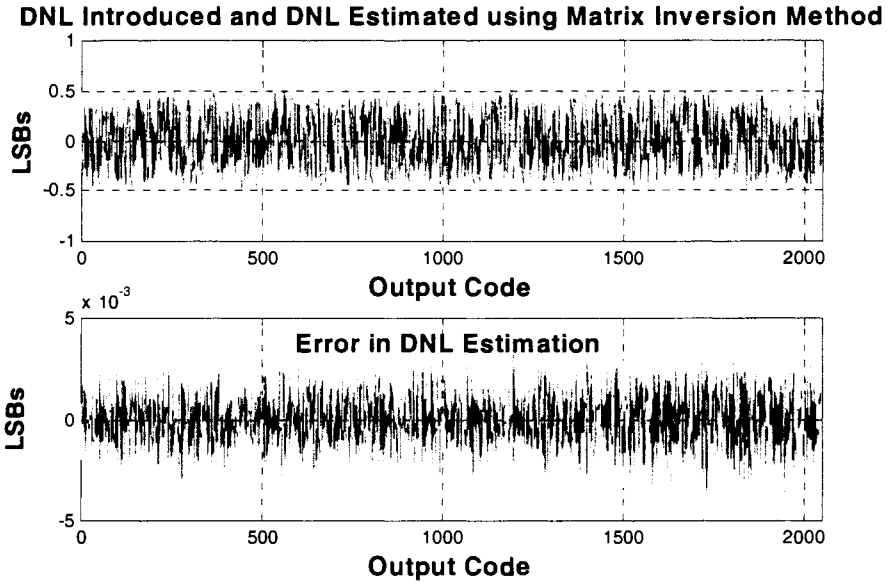


Figure.4.9 Effect of number of samples on the performance of the algorithm

Next a set of 50 runs of an 11-bit ADC was performed. A different random resistor combination was used each time to model different trip point error patterns. The matrix-inversion algorithm was used to predict the trip points for each run. The maximum trip point error (maximum of the INL values at the trip points) was then obtained for each run. The top part of Figure.4.10 gives the maximum trip point error introduced in each run and that calculated using the algorithm. The error in prediction for each run is shown in the bottom portion of the graph.

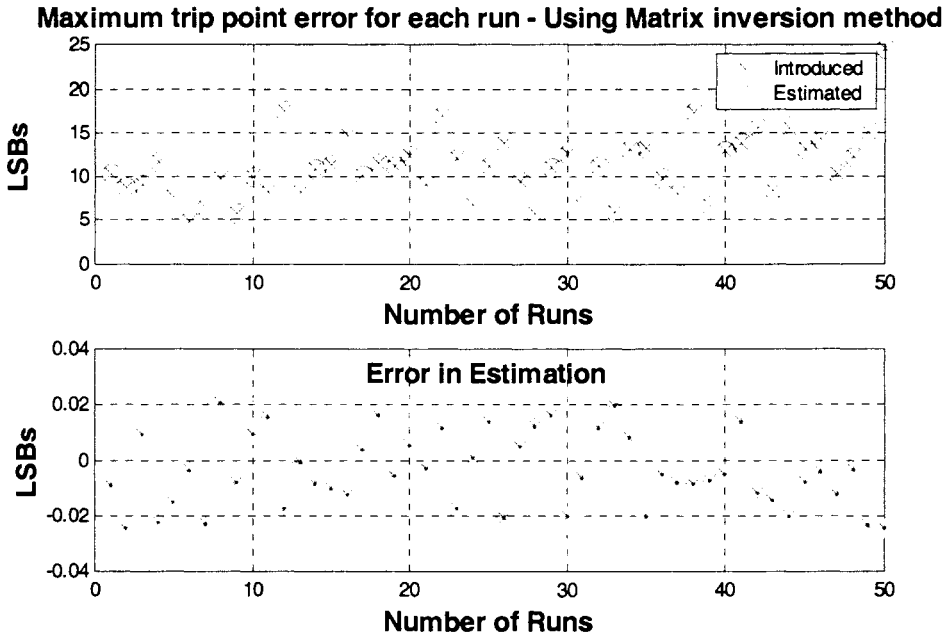


Figure.4.10 Result of 50 runs – Matrix Inversion Method

It can be seen that using the algorithm, the trip points of the converter can be identified to approximately -0.03LSB~0.02LSB (i.e. 14 bits accuracy).

Alpha-Estimation Approach

Similar simulations as were performed using the matrix-inversion approach of Section 4.6.4 were repeated for alpha-estimation technique of Section 4.6.4. To compare the performance of both the approaches, a similar case of an 11-bit converter was considered again. The linearity of the input signal was limited to 3 bits (a = -0.01 in (4.95)) and the number of samples was chosen as 32. Figure 4.11 shows the trip points introduced, trip points estimated (both shown in top portion of the figure) and the error in estimation (bottom portion of the figure).

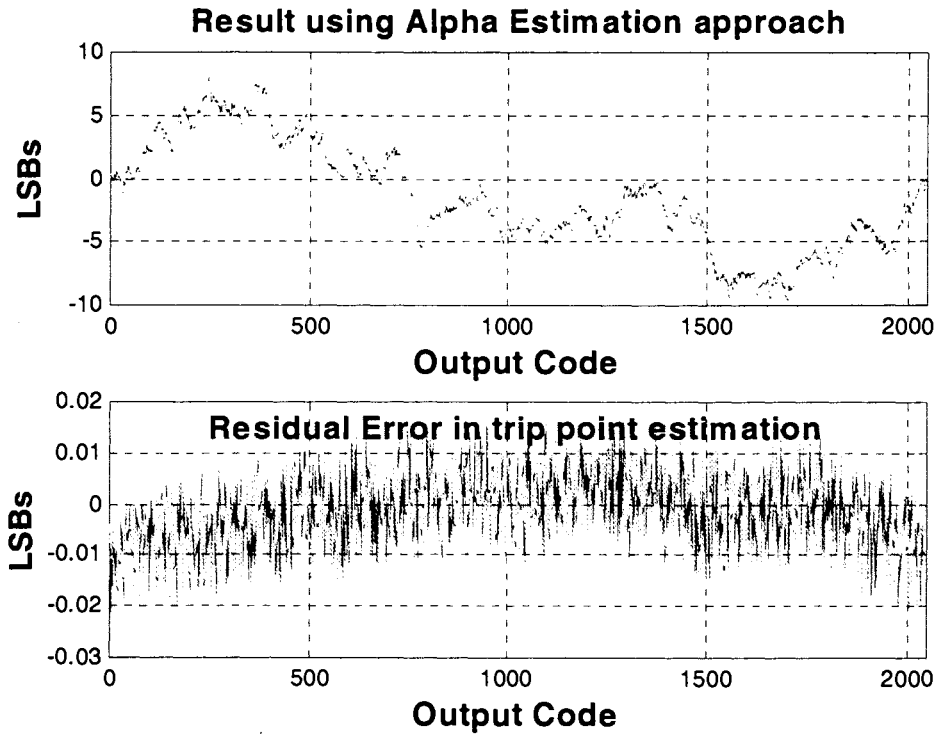


Figure.4.11 Result of Alpha estimation algorithm: Introduced and estimated trip point errors for 11-bit converter

It can be seen from the figure that the residual error is of the order of 0.02LSB. The results obtained using alpha estimation method is similar to that obtained using matrix inversion approach. The results are not exactly identical since different devices were considered for the matrix inversion and alpha estimation approaches. For same device, the prediction will be exactly identical. The primary reason behind similar results is because the two results are obtained from the same set of equations, just solved in a different manner. It has to be noted again that with the data obtained using this algorithm, and by incorporating additional calibration circuitry, the trip points can be adjusted to a very good accuracy. The trip points of the actual device ranges from -10LSB~8LSB (for this sample run). And by using the data predicted by the algorithm, the trip points can be reduced to within 0.02LSB after calibration. This corresponds to a significant improvement in device performance. With all the Ψ's available, DNL can be estimated and is shown in Figure 4.12.

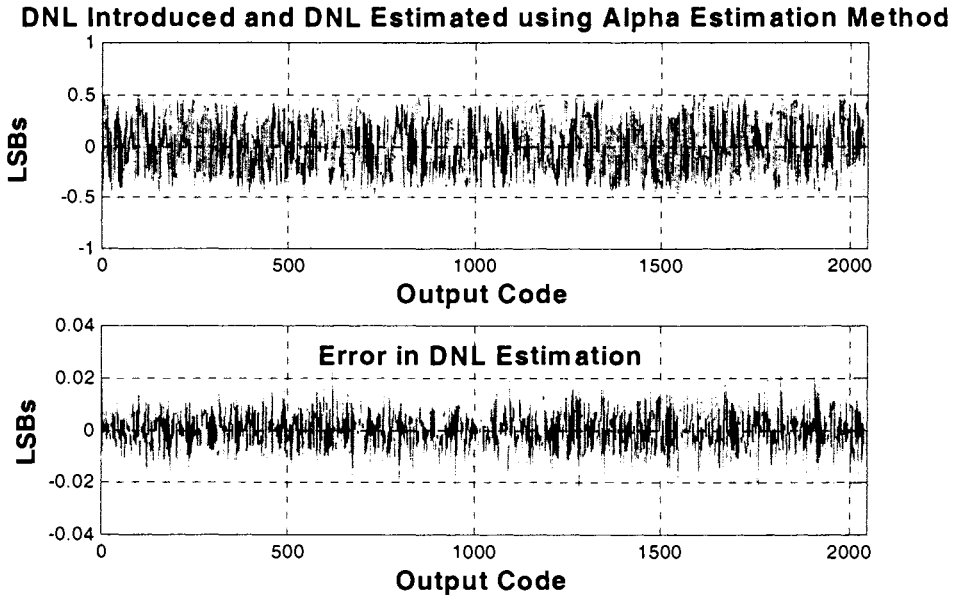


Figure.4.12 Result of Alpha estimation approach: Introduced and estimated DNL values

With the α-estimation approach, again a set of 50 runs of 11 bit ADCs was performed. Different values of resistors were considered in each run to simulate different devices. Figure.4.13. gives the plot of maximum trip point error introduced in each run and the residual error in prediction.

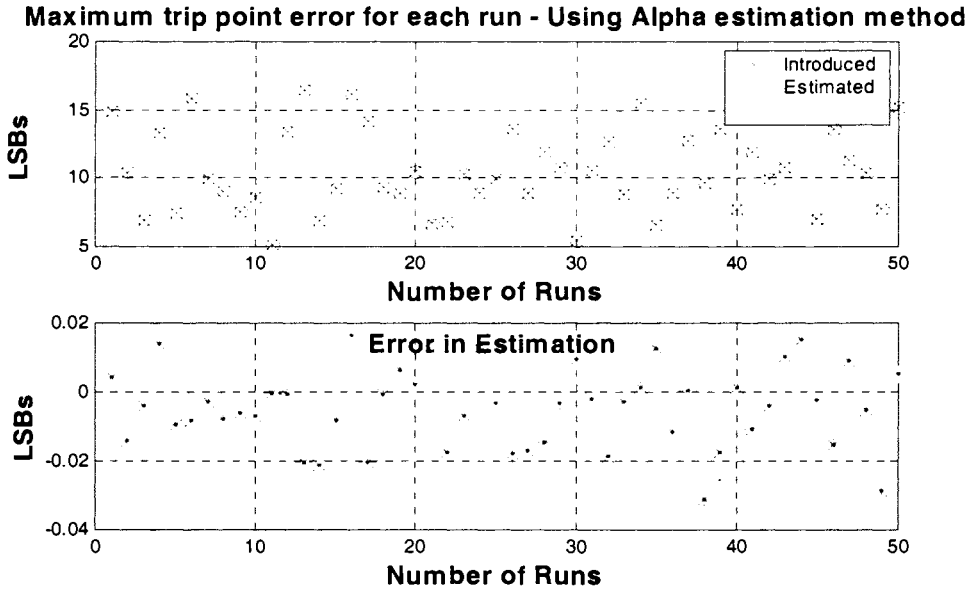


Figure.4.13 Result of 50 runs – Alpha Estimation Method

It can be seen that the residual error is now approximately -0.02LSB~0.02LSB, which implies that we can identify the trip points to nearly the 14 bit level, similar to that obtained using matrix inversion method.

To compare the complexity of the two methods, the number of floating point operations (in MATLAB) were compared for different simulation and the results are summarized in Table 4.2. As the number of bits of the ADC is increased from 9 to 10, the FLOPS required by matrix inversion approach increases by approximately 8 times, where as the FLOPS required by alpha-estimation just doubles. This is because, as explained in Section 4.6, the complexity of matrix inversion approach increases as 2^{3N}, while the complexity of alpha-estimation increase as 2^{N}. For the case of 11 bit devices considered in the simulations above, the order reduces from 5.1e10 for Matrix Inversion method to 3.6e4 for Non-Matrix Inversion method, corresponding to a significant reduction in complexity. This makes it viable for this scheme to be implemented in a production test solution. The results presented below are that obtained with the approaches explained in Section 4.6.3. The formulation of Section 4.6.4 which does a precise matrix inversion with essentially the same number of calculations as in the α-estimation approach was not considered in these comparisons. From a computational point of view, the matrix inversion approach of Section 4.6.4 should be very attractive.

Table 4.2 Computational Complexity of the Proposed Methods

MI – Matrix Inversion; NMI – Non Matrix Inversion (alpha estimation)

Bits	**Method**	**DC shift of input signal**	**Estimation of shift**	**FLOPS (in Matlab)**
9	MI	50	50.02	806188802
9	NMI	50	49.9845	9214
10	MI	100	100.004	6.4429e9

Table 4.2 Continued

Bits	Method	DC shift of input signal	Estimation of shift	FLOPS (in Matlab)
10	NMI	100	99.9956	18430
11	MI	200	200.007	5.1551e10
11	NMI	200	199.999	36862
16	MI	-	-	1.6e15 (extrapolated)
16	NMI	2500	2500.1	1179646

4.9. Experimental Results

The newly proposed algorithm was tested on different commercial ADCs to verify the functionality. Support in testing, in terms of the tester time, different DUTs and raw data were obtained through industrial liaisons sponsoring the project. The various requirements for introducing nonlinearity into the excitation were initially supplied to the liaison. Histogram data alone was then obtained without knowing the input waveform, and the algorithm was implemented on the supplied data. The predicted results were compared with those obtained by the liaison using a high cost commercial tester.

A 10bit pipeline ADC was considered as the device-under-test to prove the concept. Although the algorithm above has been described assuming the ADC is of a flash architecture, the concept can be extended to other types of architectures. A commercial tester used in production testing was used to generate the input signals and collect the output histograms. The tester was programmed to generate the non-linear input signals. A very highly non-linear input signal (approximately just 2-3 bit linear) was used as input to the device. As explained in Section 4.6.1,

although signals of much better linearity can be generated on-chip, to confirm the robustness of the algorithm and to consider the case of high resolution ADCs (14 bit and above) where 8-9 bit linear input signals are limiting factor, the input signals in the test runs were intentionally limited to 2 bits. The range of the "ramp-like" input signal was limited to the ADC full-scale range. The second input signal was obtained by adding a DC shift value to the first signal, amounting to approximately 10 LSBs of the ADC. This information about the non-linearity of the input signal and the amount of shift was initially known only to the liaison and was not supplied to us. The parameters were independently calculated as part of the algorithm and were then confirmed through the liaison.

The two "ramp-like" input signals were fed to the ADC and the output histogram was obtained. The signals were sampled such that on an average, the number of samples in each bin is nearly 32. The two raw histogram data (one obtained from each of the input signals) were then used with the algorithm. Apart from the two nonlinear signals, a highly linear third signal was fed to the ADC and the output histogram was obtained. This signal was generated using a high cost tester with the ramp being at least 3-4 bit more linear than the device-under-test. The histogram obtained using this linear signal was used to determine the actual characteristic of the device, so that the results of the proposed algorithm can be compared and analyzed. Although the main aim of this work is to solve the dual purpose of eventually reducing the requirement on signal generation on the commercial tester, and if used as a built-in-self-test approach, to reduce the requirement on on-chip signal generation; this high linearity ramp signal was used in this work, just to compare and corroborate the results of the algorithm and would not be required in the real time testing.

The histogram from the linear ramp was used to determine the INL and DNL of the particular device that was tested. Figure 4.14 gives the plot of INL of the device. Although we are presenting this data first, it was not available to us until after estimates of the INL were made using the algorithms of Section 4.6. It can be seen that the INL of the device is within 2 LSBs. Although the

simulations in Section 4.8 considered ADCs with INL in tens of LSBs, devices with INL in the range of +/- 2LSBs are not uncommon in the market. This is primarily because of two reasons. With the amount of knowledge in the area of circuit design and with the matching accuracies available in the current processes, designing devices with the below specifications is a trivial task. Also the ADC architecture considered in Section 4.8 is of type flash, which suffers from the random walk problem. The nonlinearities in the device tend to add up resulting in large values of INL, in contrast with the pipelined architecture that has INL signature patterns as shown below.

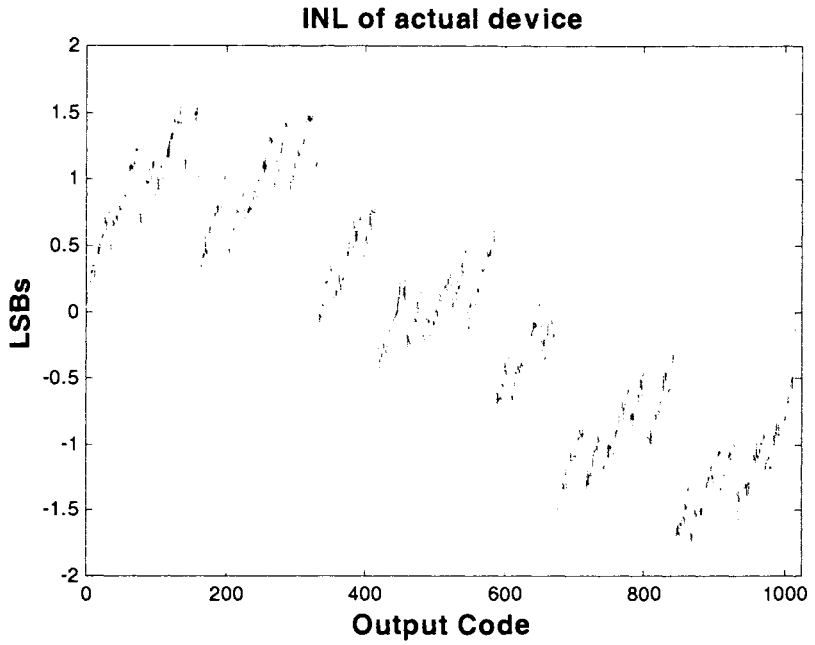


Figure.4.14 INL of the device under consideration

Figure 4.15 shows the DNL of the device, obtained using the linear histogram data and the conventional histogram approach.

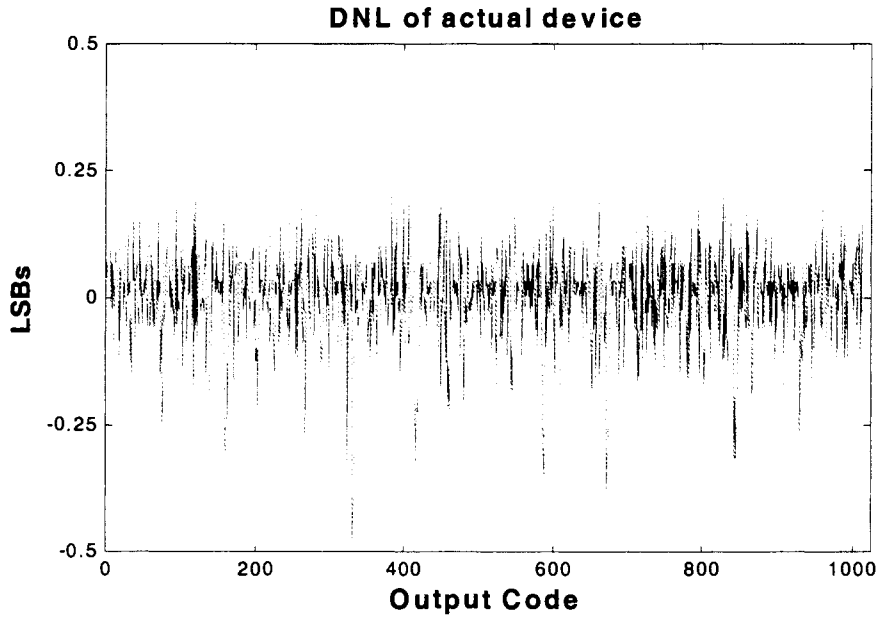


Figure.4.15 DNL of the device under consideration

Having obtained the linearity specifications of the actual device, the two algorithms were then implemented on the nonlinear histogram data and the results are summarized below.

Result of Matrix-Inversion approach

Figure 4.16 shows the plot of the INL calculated using the matrix inversion approach of Section 4.6.4. Overlaid on top is the actual INL of the device (as shown in Figure 4.14).

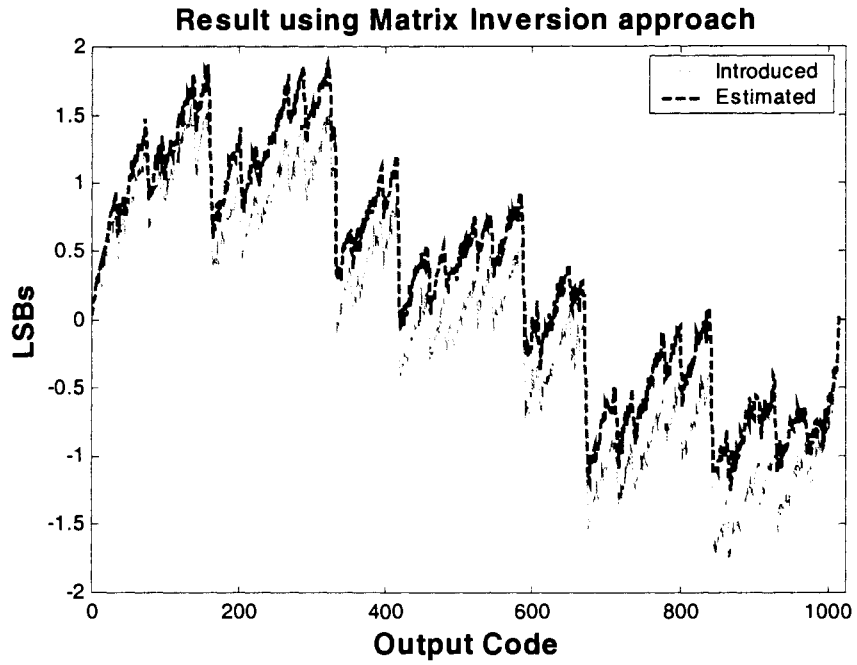


Figure.4.16 INL of the device under consideration – Calculated using matrix inversion

It can be seen from the figure that the predicted INL values closely follows the actual value. The error in prediction is shown in Figure 4.17. The prediction error is approximately 0.6LSB.

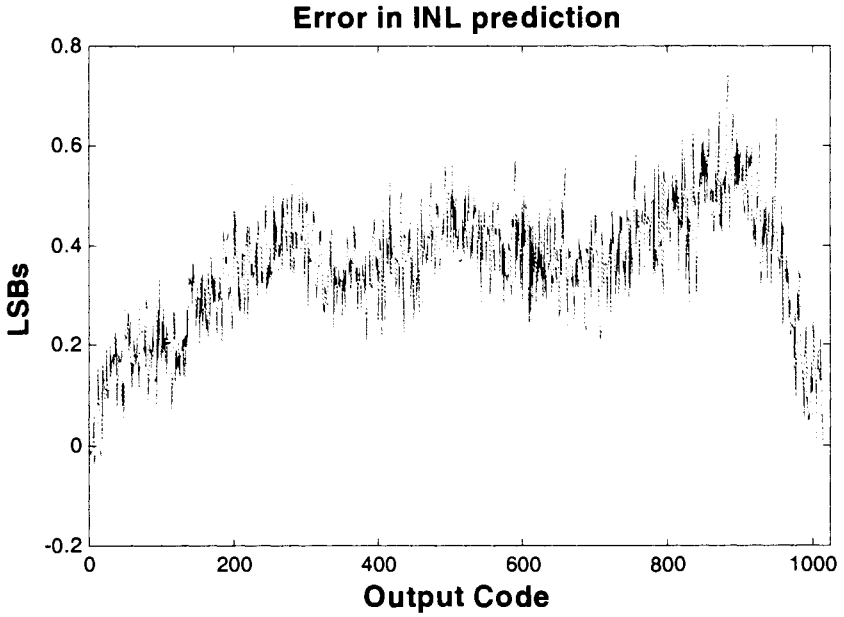


Figure.4.17 Error in INL prediction

The results obtained from the actual experimental results are not as good as that obtained from simulations. This is mainly because of effect like noise in the ADC, noise in the measurement system and tester and additional higher order nonlinearities in the input signals that were not considered in the original simulations. That said, the results are still quite good. However what is of more importance is the fact that with an input signal of just 2 bit linearity, a 10 bit ADC can be characterized to within 0.6 LSBs. This result supports the existence of a promising solution to the traditional testing problems. Not only can this approach be used to advantage in commercial testers by reducing the complexity of the signal generator, but also in BIST approaches, where a simple current-source charging a capacitor can be used to generate a pseudo ramp, like the one used in this work. It is conjectured that with some additional detailed work on noise and related aspects, the performance of the algorithm can be improved.

The input signal nonlinearity and the shift between the two signals that were estimated by the algorithm were then compared with the actual conditions provided by the liaison while testing the device. The shift estimated by the algorithm was 10.8314 LSBs, close to that provided by the liaison. Also the nonlinearity of the input signal as obtained from the algorithm is shown in Figure 4.18.

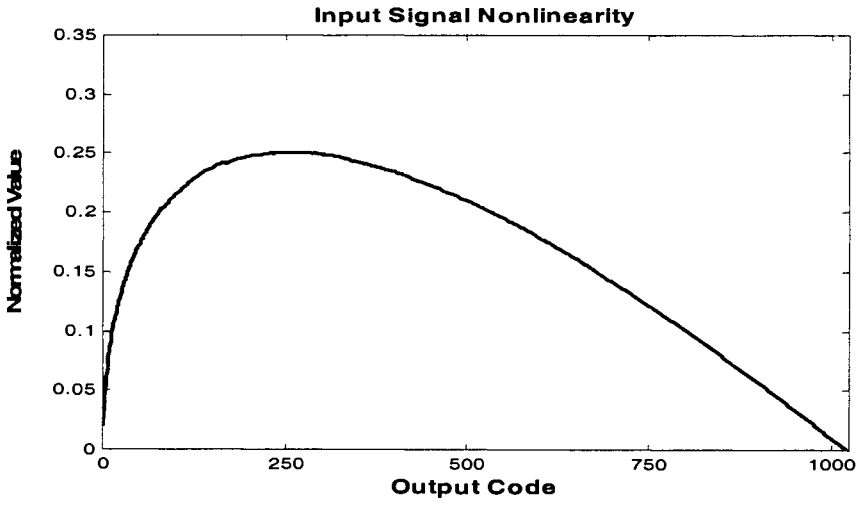


Figure.4.18 Input signal nonlinearity

Result of Alpha-Estimation approach

The histogram data obtained from the same device was then used with the alpha-estimation method that is explained in Section 4.6.4 to characterize the device again. Figure 4.19 gives a plot of the INL predicted along with the actual INL of the device.

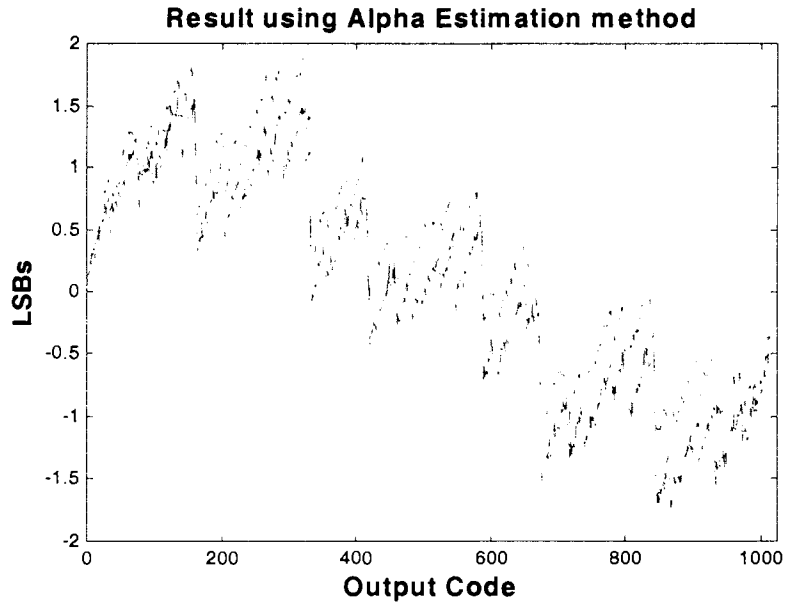


Figure.4.19 INL of the device under consideration – Using alpha-estimation approach

The difference between the estimated value and the actual data is given in Figure 4.20. The error in prediction is exactly the same as that obtained using the matrix inversion approach. This is because, as explained earlier, both the algorithm uses the same information but formulated in a slightly different manner.

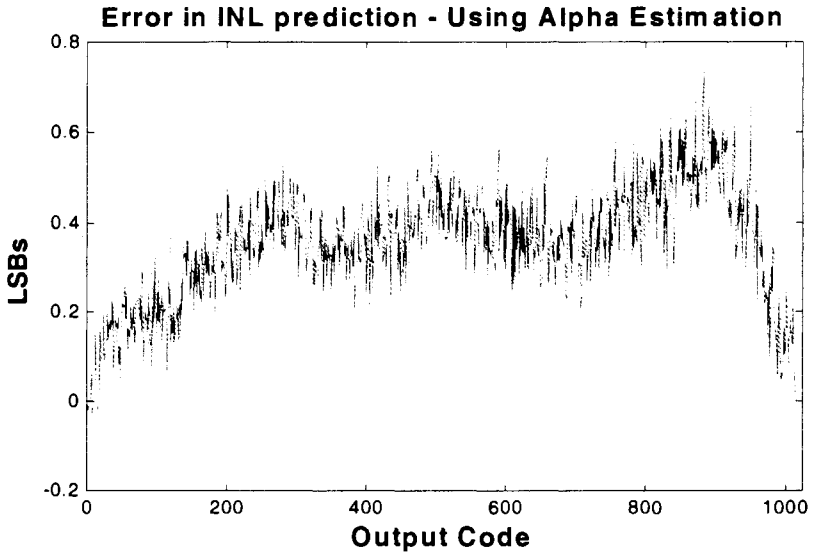


Figure.4.20 Error in INL estimation - using alpha-estimation approach

Further simulations were performed to understand the effect of noise in the test setup. Typically, when a histogram test is performed, multiple ramps are input to the device and the histograms at the output are averaged to get the final histogram. This is essential because the noise in the ADC plus the measurement setup together results in samples corresponding to a particular bin falling in adjacent bins, thereby changing the histogram count. To eliminate this random noise, multiple runs are performed and the results averaged. However, this does not completely eliminate the noise in the measured data. This unfiltered noise is one of factors affecting the results of the algorithm in the experimental tests.

To see the effect of averaging, one similar 10 bit part was randomly picked. The same histogram test described above was performed three times, but with the number of ramps (used in averaging) being different each time. For the first run, the histograms were collected using just one ramp signal. The actual INL of the device and that estimated from the histogram data are shown in Figure 4.21.

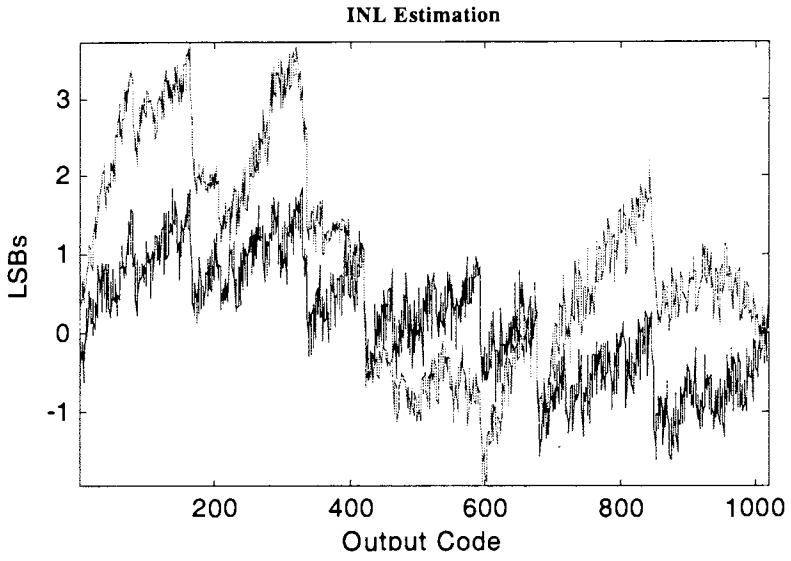


Figure.4.21 INL estimation – for 1 ramp histogram data

The error in prediction is given in Figure 4.22. It can be seen that the error is nearly around -2.5LSB ~ 1.5LSB (i.e. 4LSBs). This is mainly due to the noise in the histogram measurement.

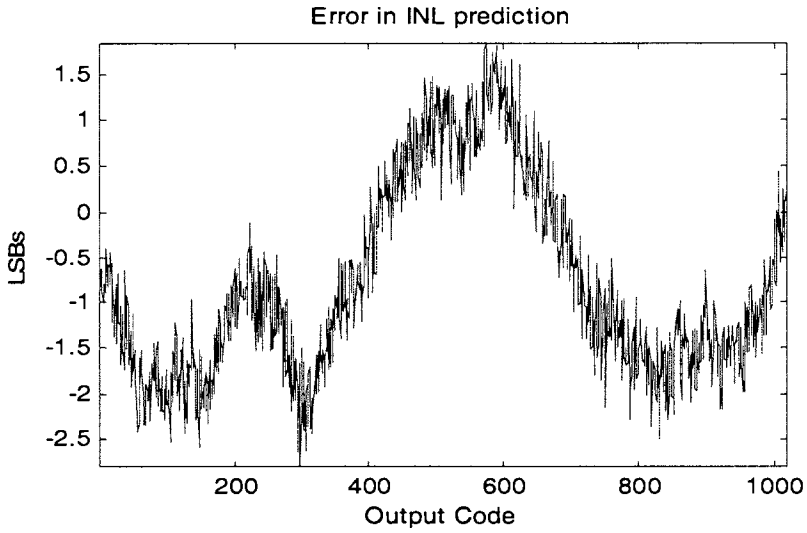


Figure.4.22 Error in INL estimation - using 1 ramp histogram data

In the second run, the experiment was repeated, but with histogram averaged over 10 runs. The algorithm was implemented on the output histogram data and the error in INL prediction is shown in Figure 4.23. The error in prediction now reduces to nearly -1.5LSB for the same device.

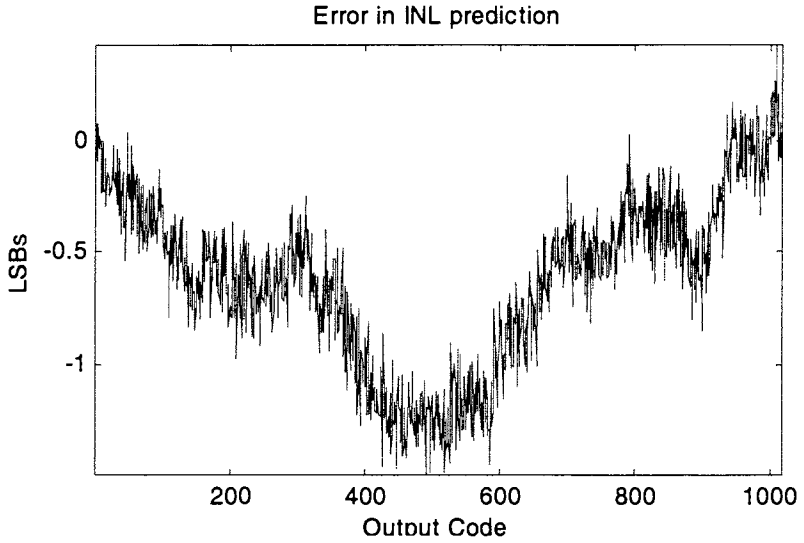


Figure.4.23 Error in INL estimation - using 10 ramps averaged histogram data

In the third run, the experiment was repeated 50 times and the output histograms were averaged, before using them with the algorithm. The error in prediction reduces to 0.6LSB as shown in Figure 4.24, indicating that noise is a major contributor of error in the experimental results. However, considering the fact that a 2 bit linear signal is used to characterize a 10 bit device, the estimation is now within reasonable range.

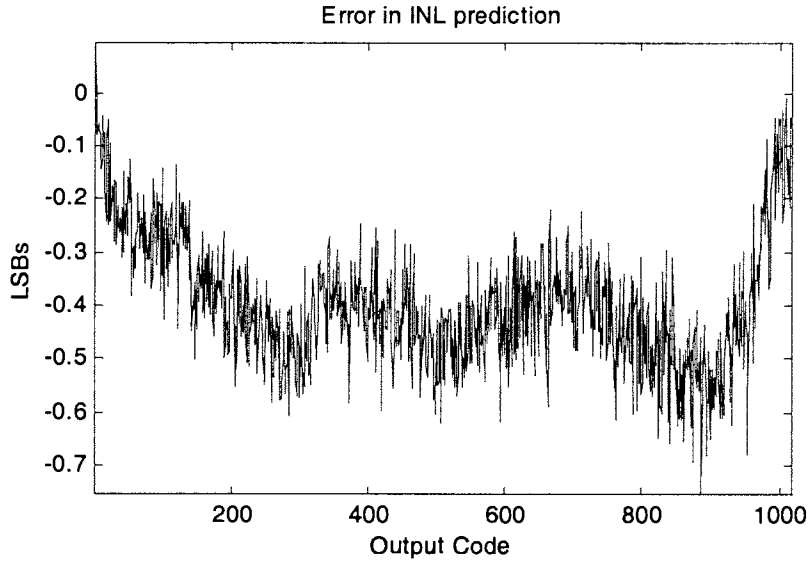


Figure.4.24 Error in INL estimation - using 50 ramps averaged histogram data

Having seen the performance of both the algorithm on the actual test data (for 1 device) and the effect of noise in the measurement setup, the experiments were repeated on 20 different samples (different ADCs). Similar input signal (nearly 2 bit linear) was used to test all the devices. The amount by which the second nonlinear signal is shifted with respect to the first one was nearly 10LSBs for all devices. The histograms obtained for the devices were then used to characterize the devices. Figure 4.25 shows the maximum error in prediction for each device.

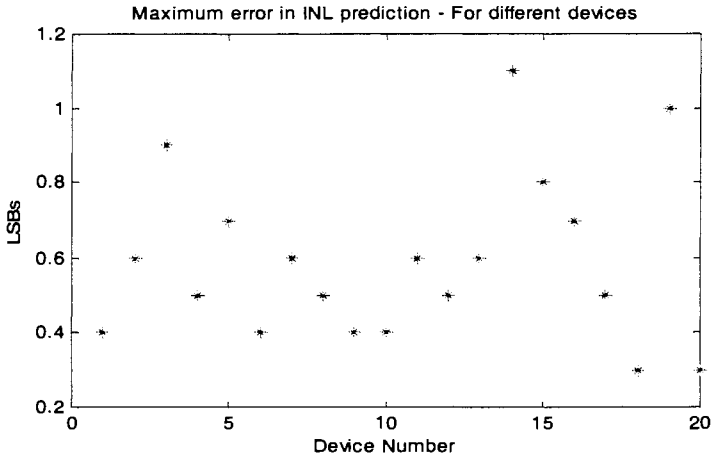


Figure.4.25 Error in INL estimation – for 20 devices

It can be seen that for most of the parts, the error in prediction is within 0.6LSBs.

4.10. Conclusion and Future Work

A new histogram based approach for characterizing an analog-to-digital converter has been proposed. Although the description in this work has been mainly targeted towards flash A/D converter, the concept can be extended to other architectures as well. The approach is mainly based upon identifying the system under test (DUT) in the presence of a highly non-ideal input stimulus. In contrast to the standard approach of feeding a highly precise and known input stimuli and

characterizing the device base on the output, the method proposed here uses multiple unknown inputs; inputs that are easily and practically realizable on-chip. The outputs obtained from the multiple inputs are then used to characterize the device and the input as well.

Three approaches for solving the problem have been proposed in this work. One method is computationally intensive, but results in better characterization, while the other is less intensive and is based on some approximations. A third algorithm was introduced that has both high accuracy and a very low computational requirement. Simulation results indicating the performance of the algorithm have been described. It has been shown that devices with INL in the range of tens of LSBs can be identified to within a fraction of a LSB using a very low linearity input signal. The proposed method has been tested on various commercial ADCs and the results have also been summarized. The experimental results support the performance of the algorithm.

One main area where more work needs to be done is the effect of noise in the experimental setup on the performance of the algorithm. The issue of missing codes in ADC output deserves consideration as does the topic of non-monotonicity. Also to realize a complete Built-In-Self-Test solution, the entire set up consisting of the signal generator, the device, the measurement block and the post processing system needs to be fully implemented on chip. A full circuit level implementation of the above algorithm could be a potential topic of research. Also alternate algorithm based on the general concept introduced in this work for characterizing the device need to be studied.

4.10. References

[1] M.Burns and G.W.Roberts, "*An Introduction to Mixed-Signal IC Test and Measurement,*" Oxford University Press, New York, USA 2000.

[2] Doernberg, J., Lee, H.S., and Hodges, D.A., *"Full-Speed Testing of A/D Converters,"* IEEE J. Solid-State Circuits, December 1984, SC-19, pp.820-827.

[3] J. Blair, *"Histogram measurement of ADC nonlinearities using sine waves"*, IEEE Trans. Instrum. Meas., Vol.43, pp.373-383, June 1994.

[4] Kuyel, T., *Linearity Testing Issues of Analog to Digital Converters*, Test Conference, 1999. Proceedings, International, 1999, Page(s): 747-756.

[5] M.F.Toner and Gordon W.Roberts, *"A BIST Scheme for a SNR, Gain Tracking, and Frequency Response Test of a Sigma-Delta ADC"*, IEEE Trans. On Circuits and Systems-II: Analog and Digital Signal Processing, Vol.42, No.1, Jan 1995, pp. 1-15.

[6] M.Mahoney, *"DSP-Based Testing of Analog and Mixed-Signal Circuits"*, Computer Society Press, IEEE, Washington, D.C., 1987.

[7] Jiun-Lang Huang and et.al, *"A BIST Scheme for On-chip ADC and DAC Testing"*, Design, Automation and Test in Europe Conference & Exhibition, 2000, pp.216-220.

[8] Stephen K.Sunter and Naveen Nagi, *"A Simplified Polynomial-Fitting Algorithm for DAC and ADC BIST"*, 1997 IEEE International Test Conference, pp.389-395.

[9] R. de Vries, T. Zwemstra, E.M.J.G.Bruls and P.P.L.Regtien, *"Built-In Self-Test Methodology for A/D Converters"*, 1997 European Design and Test Conference, pp.353-358.

[10] B.Provost and E.Sanchez-Sinencio, *"Auto-Calibrating Analog Timer for On-Chip Testing"*, Proc. Of International Test Conference, 1999, pp.541-548.

[11] Jiun-Lang Huang, Chee-Kian Ong, Kwnag-Ting Cheng, "*A BIST Scheme for On-chip ADC and DAC Testing*", Design, Automation and Test in Europe Conference and Exhibition 2000., pp.216-220.

[12] Jing Wang, E.Sanchez-Sinencio, Franco Maloberti, "*Very Linear Ramp-Generators for High Resolution ADC BIST and Calibration*", 2000 IEEE Midwest Symposium on Circuits and Systems, August 2000.

[13] F.Azais et.al, "*A Low-Cost Adaptive Ramp Generator for Analog BIST Applications*", 19th IEEE Proc. On VLSI Test Symposium, 2001, pp.266-271.

[14] Gordon W.Roberts and Benoit Dufort, "*Making Complex Mixed-Signal Telecommunication integrated circuits testable*", IEEE Communications Magazine, June 1999, pp.90-96.

[15] Xavier Haurie and Gordon W.Roberts, "*Arbitrary-Precision Signal Generation for Mixed-Signal Built-In Self-Test*", IEEE Trans. On Circuits and Systems-II: Analog and Digital Signal Processing, Vol.45, No.11, Nov 1998, pp.1425-1432.

[16] Evan M.Hawrysh and Gordon W.Roberts, "*An Integration of Memory-Based Analog Signal Generation into Current DFT Architectures*", IEEE Trans. On Instrumentation and Measurement, Vol.47, No.3, June 1998, pp.748-759.

[17] Benoit Dufort and Gordon W.Roberts, "*Increasing the Performance of Arbitrary Waveform Generators Using Periodic Sigma-Delta Modulated Streams*", IEEE Trans. On Instrumentation and Measurement, Vol.49, No.1, Feb 2000, pp.188-199.

[18] Jiun-Lang Huang and Kwang-Ting Cheng, "*A Sigma-Delta Modulation Based BIST Scheme for Mixed-Signal Circuits*", 2000 Asia and S.Pacific Design Automation Conference, pp.605-610.

[19] K.L.Parthasarathy and R.L.Geiger, "*Accurate Self Characterization and Correction of A/D Converter Performance*", 2001 IEEE Midwest Symposium on Circuits and Systems, August 2001.

[20] K.L.Parthasarathy, Le Jin, Degang Chen and R.L.Geiger, "*A Modified Histogram Approach for Accurate Self-Characterization of Analog-to-Digital Converters*", Proceedings of 2002 IEEE ISCAS, Arizona, May 2002.

[21] K.L.Parthasarathy, Le Jin, Turker Kuyel, Degang Chen and R.L.Geiger, "*A Histogram Based AM-BIST Algorithm for ADC Characterization Using Imprecise Stimulus*", Proceedings of 2002 IEEE Midwest Symposium on Circuits and Systems, Oklahoma, Aug 2002.

5. CONCLUSIONS

This thesis work dealt with three different aspects of data converters. One of the main issues is related to how the various parameters of the device are traditionally defined and how the definitions can be misinterpreted. A comparison of the various definitions and the ambiguities related to them has been addressed. A new set of unambiguous definitions for the various specifications relating to Digital-to-Analog converters has been presented.

The second issue related to data converter design is the calibration of these devices to meet the various performance specifications. A new calibration scheme for digital calibration of a 16-bit resistor string Digital-to-Analog converter has been introduced in Chapter 2. Calibration steps, architectural details and the associated design issues have been clearly discussed. The design and implementation of various analog and digital modules have been presented. Recommendations on future work are also made towards the end of the chapter.

A third related aspect involved with testing and characterization of data converters. With the cost of testing becoming the main limiting factor in the reduction of IC prices, BIST solutions and structures has been the topic of research in the past few years. One new approach towards characterizing an Analog-to-Digital converter has been proposed. The algorithm is mainly targeted towards measuring the linearity characteristics of the device under consideration. Unlike the standard method of using a high precision and well defined input stimuli to test the device, a new approach based on blind estimation has been introduced. The approach is based on feeding multiple signals to the device; signals that have relaxed requirements and that can be practically realizable on chip. The multiple outputs obtained from the device are then used to characterize the device. A histogram based algorithm has been proposed and results from both simulation and experiments on commercial products have been included. The results confirm the performance of the algorithm.

APPENDIX A – VHDL Code

Code 1

```
--This block is a 8bit register used for storing Y values (INL values)

library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;

--The entity is called memory
entity memory is
        port(   clk,resetn: in std_logic;
                Data_in : in std_logic_vector(6 downto 0);
                Data_out : out std_logic_vector(6 downto 0));
end memory;

architecture RTL of memory is

begin
        process(clk,resetn)
        begin
                if resetn='0' then
                        Data_out<=(others =>'0');
                elsif clk'event and clk='1' then
                        Data_out<=Data_in;
                end if;
        end process;
end;
```

Code 2

```
--This is the test bench for 8bit register

library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;

entity T_memory is
end T_memory;

architecture  TEST of T_memory is

  component memory
  port (  clk,resetn: in std_logic;
          Data_in : in std_logic_vector(6 downto 0);
          Data_out: out std_logic_vector(6 downto 0));
  end component;

signal clk,resetn  : std_logic;
signal Data_in, Data_out   : std_logic_vector (6 downto 0);

begin
```

```
  ---------------------------------------------------
  -- Instantiate UUT
  ---------------------------------------------------

  UUT : memory port map (clk,resetn,Data_in,Data_out);

  ---------------------------------------------------
  -- Stimulus:
  ---------------------------------------------------
  STIMULUS: process
  begin
                -- Initially reset is HIGH and CLK is LOW
                Data_in<="0010100";resetn<='1';clk<='0';
                wait for 5 ns;

                -- On the Rising edge of CLK, Data_in should be transferred to Data_out
                clk<='1';
                wait for 5 ns;

                --More inputs are tried
                Data_in<="1010111";clk<='0';
                wait for 1 ns;
                clk<='1';
                wait for 4 ns;
                Data_in<="1011110";clk<='0';
                wait for 1 ns;
                clk<='1';
                wait for 2 ns;

                --Reset function is checked by making it asynchronously ZERO
                --Also CLK is made zero so that data can be captured on next rising edge
                resetn<='0';clk<='0';
                wait for 3 ns;
                resetn<='1';
                wait for 1 ns;
                Data_in<="1100000";
                wait for 2 ns;
                clk<='1';
                wait for 2 ns;
                Data_in<="0101010";clk<='0';
                wait for 5 ns;
                clk<='1';
                wait;
  end process STIMULUS;
end TEST;

---------------------------------------------------
-- Configuration...
---------------------------------------------------
configuration CFG_T_memory of T_memory is
  for TEST
  end for;
end CFG_T_memory;
```

Code 3

```
module memory ( clk, resetn, Data_in, Data_out );
output [6:0] Data_out;
input  [6:0] Data_in;
input  clk, resetn;
  dffr \Data_out_reg[6]  ( .Q(Data_out[6]), .CLK(clk), .D(Data_in[6]), .RST(
    resetn) );
  dffr \Data_out_reg[5]  ( .Q(Data_out[5]), .CLK(clk), .D(Data_in[5]), .RST(
    resetn) );
  dffr \Data_out_reg[4]  ( .Q(Data_out[4]), .CLK(clk), .D(Data_in[4]), .RST(
    resetn) );
  dffr \Data_out_reg[3]  ( .Q(Data_out[3]), .CLK(clk), .D(Data_in[3]), .RST(
    resetn) );
  dffr \Data_out_reg[2]  ( .Q(Data_out[2]), .CLK(clk), .D(Data_in[2]), .RST(
    resetn) );
  dffr \Data_out_reg[1]  ( .Q(Data_out[1]), .CLK(clk), .D(Data_in[1]), .RST(
    resetn) );
  dffr \Data_out_reg[0]  ( .Q(Data_out[0]), .CLK(clk), .D(Data_in[0]), .RST(
    resetn) );
endmodule
```

Code 4

```
--This represents the control block, which decides Yi+1 and Yi based on input

library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;

entity control is
        port(   clock,reset     : in std_logic;
                msb             : in std_logic_vector(2 downto 0);
                datain1,datain2,datain3,datain4,datain5     : in std_logic_vector(6 downto 0);
                datain6,datain7,datain8,datain9     : in std_logic_vector(6 downto 0);
                out1            : out std_logic_vector(6 downto 0);
                out2            : out std_logic_vector(6 downto 0));
end control;

architecture RTL of control is

component memory
        port(clk,resetn : in std_logic;
           Data_in : in std_logic_vector(6 downto 0);
           Data_out : out std_logic_vector(6 downto 0));
end component;

signal  dataout1,dataout2,dataout3,dataout4,dataout5 : std_logic_vector(6 downto 0);
signal  dataout6,dataout7,dataout8,dataout9 : std_logic_vector(6 downto 0);
```

```
begin
        register1: memory port map(clock,reset,datain1,dataout1);
        register2: memory port map(clock,reset,datain2,dataout2);
        register3: memory port map(clock,reset,datain3,dataout3);
        register4: memory port map(clock,reset,datain4,dataout4);
        register5: memory port map(clock,reset,datain5,dataout5);
        register6: memory port map(clock,reset,datain6,dataout6);
        register7: memory port map(clock,reset,datain7,dataout7);
        register8: memory port map(clock,reset,datain8,dataout8);
        register9: memory port map(clock,reset,datain9,dataout9);

process(msb,dataout1,dataout2,dataout3,dataout4,dataout5,dataout6,dataout7,dataout8,dataout9)
        begin

        case msb is
                when "000" => out1<=dataout1;
                 out2<=dataout2;
                when "001" => out1<=dataout2;
                 out2<=dataout3;
                when "010" => out1<=dataout3;
                 out2<=dataout4;
                when "011" => out1<=dataout4;
                 out2<=dataout5;
                when "100" => out1<=dataout5;
                 out2<=dataout6;
                when "101" => out1<=dataout6;
                 out2<=dataout7;
                when "110" => out1<=dataout7;
                 out2<=dataout8;
                when "111" => out1<=dataout8;
                 out2<=dataout9;
                when others => out1<=dataout1;
                out2<=dataout2;
        end case;
        end process;
end;
```

Code 5

```
--This is the test bench for control block

library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;

entity T_control is
end T_control;

architecture  TEST of T_control is

  component control
  port (  clock,reset : in std_logic;
        msb: in std_logic_vector(2 downto 0);
```

```
        datain1,datain2,datain3,datain4,datain5 : in std_logic_vector(6 downto 0);
        datain6,datain7,datain8,datain9 : in std_logic_vector(6 downto 0);
        out1 : out std_logic_vector(6 downto 0);
        out2 : out std_logic_vector(6 downto 0));
  end component;

        signal msb : std_logic_vector(2 downto 0);
        signal clock,reset  : std_logic;
        signal datain1,datain2,datain3,datain4,datain5,out1,out2        : std_logic_vector(6 downto 0);
        signal datain6,datain7,datain8,datain9          : std_logic_vector(6 downto 0);

begin
  ---------------------------------------------------
  -- Instantiate UUT
  ---------------------------------------------------
  UUT : control port map (clock,reset,msb,datain1,datain2,datain3,datain4,datain5,datain6,
        datain7,datain8,datain9,out1,out2);
  ---------------------------------------------------
  -- Stimulus:
  ---------------------------------------------------
  STIMULUS: process
  begin
datain1<="0000000";datain2<="0111111";datain3<="1000000";reset<='1';msb<="000";clock<='0';
datain4<="0000000";datain5<="1110000";datain6<="1001000";datain7<="0111000";
datain8<="0010000";datain9<="0000000";
                    wait for 3 ns;
                    clock<='1';
                    wait for 5 ns;
                    msb<="001";
                    wait for 5 ns;
                    msb<="111";
                    wait for 5 ns;
                    msb<="011";
                    wait for 5 ns;
                    msb<="000";
                    wait for 5 ns;
                    msb<="110";
                    wait;
  end process STIMULUS;
end TEST;

---------------------------------------------------
-- Configuration...
---------------------------------------------------
configuration CFG_T_control of T_control is
  for TEST
  end for;
end CFG_T_control;
```

Code 6

```
module memory_0 ( clk, resetn, Data_in, Data_out );
output [6:0] Data_out;
```

```
input  [6:0] Data_in;
input  clk, resetn;
   dffr \Data_out_reg[6]  ( .Q(Data_out[6]), .CLK(clk), .D(Data_in[6]), .RST(
      resetn) );
   dffr \Data_out_reg[5]  ( .Q(Data_out[5]), .CLK(clk), .D(Data_in[5]), .RST(
      resetn) );
   dffr \Data_out_reg[4]  ( .Q(Data_out[4]), .CLK(clk), .D(Data_in[4]), .RST(
      resetn) );
   dffr \Data_out_reg[3]  ( .Q(Data_out[3]), .CLK(clk), .D(Data_in[3]), .RST(
      resetn) );
   dffr \Data_out_reg[2]  ( .Q(Data_out[2]), .CLK(clk), .D(Data_in[2]), .RST(
      resetn) );
   dffr \Data_out_reg[1]  ( .Q(Data_out[1]), .CLK(clk), .D(Data_in[1]), .RST(
      resetn) );
   dffr \Data_out_reg[0]  ( .Q(Data_out[0]), .CLK(clk), .D(Data_in[0]), .RST(
      resetn) );
endmodule


module memory_1 ( clk, resetn, Data_in, Data_out );
output [6:0] Data_out;
input  [6:0] Data_in;
input  clk, resetn;
   dffr \Data_out_reg[6]  ( .Q(Data_out[6]), .CLK(clk), .D(Data_in[6]), .RST(
      resetn) );
   dffr \Data_out_reg[5]  ( .Q(Data_out[5]), .CLK(clk), .D(Data_in[5]), .RST(
      resetn) );
   dffr \Data_out_reg[4]  ( .Q(Data_out[4]), .CLK(clk), .D(Data_in[4]), .RST(
      resetn) );
   dffr \Data_out_reg[3]  ( .Q(Data_out[3]), .CLK(clk), .D(Data_in[3]), .RST(
      resetn) );
   dffr \Data_out_reg[2]  ( .Q(Data_out[2]), .CLK(clk), .D(Data_in[2]), .RST(
      resetn) );
   dffr \Data_out_reg[1]  ( .Q(Data_out[1]), .CLK(clk), .D(Data_in[1]), .RST(
      resetn) );
   dffr \Data_out_reg[0]  ( .Q(Data_out[0]), .CLK(clk), .D(Data_in[0]), .RST(
      resetn) );
endmodule


module memory_2 ( clk, resetn, Data_in, Data_out );
output [6:0] Data_out;
input  [6:0] Data_in;
input  clk, resetn;
   dffr \Data_out_reg[6]  ( .Q(Data_out[6]), .CLK(clk), .D(Data_in[6]), .RST(
      resetn) );
   dffr \Data_out_reg[5]  ( .Q(Data_out[5]), .CLK(clk), .D(Data_in[5]), .RST(
      resetn) );
   dffr \Data_out_reg[4]  ( .Q(Data_out[4]), .CLK(clk), .D(Data_in[4]), .RST(
      resetn) );
   dffr \Data_out_reg[3]  ( .Q(Data_out[3]), .CLK(clk), .D(Data_in[3]), .RST(
      resetn) );
   dffr \Data_out_reg[2]  ( .Q(Data_out[2]), .CLK(clk), .D(Data_in[2]), .RST(
      resetn) );
```

```
  dffr \Data_out_reg[1] ( .Q(Data_out[1]), .CLK(clk), .D(Data_in[1]), .RST(
    resetn) );
  dffr \Data_out_reg[0] ( .Q(Data_out[0]), .CLK(clk), .D(Data_in[0]), .RST(
    resetn) );
endmodule


module memory_3 ( clk, resetn, Data_in, Data_out );
output [6:0] Data_out;
input  [6:0] Data_in;
input  clk, resetn;
  dffr \Data_out_reg[6] ( .Q(Data_out[6]), .CLK(clk), .D(Data_in[6]), .RST(
    resetn) );
  dffr \Data_out_reg[5] ( .Q(Data_out[5]), .CLK(clk), .D(Data_in[5]), .RST(
    resetn) );
  dffr \Data_out_reg[4] ( .Q(Data_out[4]), .CLK(clk), .D(Data_in[4]), .RST(
    resetn) );
  dffr \Data_out_reg[3] ( .Q(Data_out[3]), .CLK(clk), .D(Data_in[3]), .RST(
    resetn) );
  dffr \Data_out_reg[2] ( .Q(Data_out[2]), .CLK(clk), .D(Data_in[2]), .RST(
    resetn) );
  dffr \Data_out_reg[1] ( .Q(Data_out[1]), .CLK(clk), .D(Data_in[1]), .RST(
    resetn) );
  dffr \Data_out_reg[0] ( .Q(Data_out[0]), .CLK(clk), .D(Data_in[0]), .RST(
    resetn) );
endmodule


module memory_4 ( clk, resetn, Data_in, Data_out );
output [6:0] Data_out;
input  [6:0] Data_in;
input  clk, resetn;
  dffr \Data_out_reg[6] ( .Q(Data_out[6]), .CLK(clk), .D(Data_in[6]), .RST(
    resetn) );
  dffr \Data_out_reg[5] ( .Q(Data_out[5]), .CLK(clk), .D(Data_in[5]), .RST(
    resetn) );
  dffr \Data_out_reg[4] ( .Q(Data_out[4]), .CLK(clk), .D(Data_in[4]), .RST(
    resetn) );
  dffr \Data_out_reg[3] ( .Q(Data_out[3]), .CLK(clk), .D(Data_in[3]), .RST(
    resetn) );
  dffr \Data_out_reg[2] ( .Q(Data_out[2]), .CLK(clk), .D(Data_in[2]), .RST(
    resetn) );
  dffr \Data_out_reg[1] ( .Q(Data_out[1]), .CLK(clk), .D(Data_in[1]), .RST(
    resetn) );
  dffr \Data_out_reg[0] ( .Q(Data_out[0]), .CLK(clk), .D(Data_in[0]), .RST(
    resetn) );
endmodule


module memory_5 ( clk, resetn, Data_in, Data_out );
output [6:0] Data_out;
input  [6:0] Data_in;
input  clk, resetn;
  dffr \Data_out_reg[6] ( .Q(Data_out[6]), .CLK(clk), .D(Data_in[6]), .RST(
```

```
    resetn) );
  dffr \Data_out_reg[5] ( .Q(Data_out[5]), .CLK(clk), .D(Data_in[5]), .RST(
    resetn) );
  dffr \Data_out_reg[4] ( .Q(Data_out[4]), .CLK(clk), .D(Data_in[4]), .RST(
    resetn) );
  dffr \Data_out_reg[3] ( .Q(Data_out[3]), .CLK(clk), .D(Data_in[3]), .RST(
    resetn) );
  dffr \Data_out_reg[2] ( .Q(Data_out[2]), .CLK(clk), .D(Data_in[2]), .RST(
    resetn) );
  dffr \Data_out_reg[1] ( .Q(Data_out[1]), .CLK(clk), .D(Data_in[1]), .RST(
    resetn) );
  dffr \Data_out_reg[0] ( .Q(Data_out[0]), .CLK(clk), .D(Data_in[0]), .RST(
    resetn) );
endmodule


module memory_6 ( clk, resetn, Data_in, Data_out );
output [6:0] Data_out;
input  [6:0] Data_in;
input  clk, resetn;
  dffr \Data_out_reg[6] ( .Q(Data_out[6]), .CLK(clk), .D(Data_in[6]), .RST(
    resetn) );
  dffr \Data_out_reg[5] ( .Q(Data_out[5]), .CLK(clk), .D(Data_in[5]), .RST(
    resetn) );
  dffr \Data_out_reg[4] ( .Q(Data_out[4]), .CLK(clk), .D(Data_in[4]), .RST(
    resetn) );
  dffr \Data_out_reg[3] ( .Q(Data_out[3]), .CLK(clk), .D(Data_in[3]), .RST(
    resetn) );
  dffr \Data_out_reg[2] ( .Q(Data_out[2]), .CLK(clk), .D(Data_in[2]), .RST(
    resetn) );
  dffr \Data_out_reg[1] ( .Q(Data_out[1]), .CLK(clk), .D(Data_in[1]), .RST(
    resetn) );
  dffr \Data_out_reg[0] ( .Q(Data_out[0]), .CLK(clk), .D(Data_in[0]), .RST(
    resetn) );
endmodule


module memory_7 ( clk, resetn, Data_in, Data_out );
output [6:0] Data_out;
input  [6:0] Data_in;
input  clk, resetn;
  dffr \Data_out_reg[6] ( .Q(Data_out[6]), .CLK(clk), .D(Data_in[6]), .RST(
    resetn) );
  dffr \Data_out_reg[5] ( .Q(Data_out[5]), .CLK(clk), .D(Data_in[5]), .RST(
    resetn) );
  dffr \Data_out_reg[4] ( .Q(Data_out[4]), .CLK(clk), .D(Data_in[4]), .RST(
    resetn) );
  dffr \Data_out_reg[3] ( .Q(Data_out[3]), .CLK(clk), .D(Data_in[3]), .RST(
    resetn) );
  dffr \Data_out_reg[2] ( .Q(Data_out[2]), .CLK(clk), .D(Data_in[2]), .RST(
    resetn) );
  dffr \Data_out_reg[1] ( .Q(Data_out[1]), .CLK(clk), .D(Data_in[1]), .RST(
    resetn) );
  dffr \Data_out_reg[0] ( .Q(Data_out[0]), .CLK(clk), .D(Data_in[0]), .RST(
```

```
    resetn) );
endmodule


module memory_8 ( clk, resetn, Data_in, Data_out );
output [6:0] Data_out;
input  [6:0] Data_in;
input  clk, resetn;
  dffr \Data_out_reg[6]  ( .Q(Data_out[6]), .CLK(clk), .D(Data_in[6]), .RST(
    resetn) );
  dffr \Data_out_reg[5]  ( .Q(Data_out[5]), .CLK(clk), .D(Data_in[5]), .RST(
    resetn) );
  dffr \Data_out_reg[4]  ( .Q(Data_out[4]), .CLK(clk), .D(Data_in[4]), .RST(
    resetn) );
  dffr \Data_out_reg[3]  ( .Q(Data_out[3]), .CLK(clk), .D(Data_in[3]), .RST(
    resetn) );
  dffr \Data_out_reg[2]  ( .Q(Data_out[2]), .CLK(clk), .D(Data_in[2]), .RST(
    resetn) );
  dffr \Data_out_reg[1]  ( .Q(Data_out[1]), .CLK(clk), .D(Data_in[1]), .RST(
    resetn) );
  dffr \Data_out_reg[0]  ( .Q(Data_out[0]), .CLK(clk), .D(Data_in[0]), .RST(
    resetn) );
endmodule


module control ( clock, reset, msb, datain1, datain2, datain3, datain4,
  datain5, datain6, datain7, datain8, datain9, out1, out2 );
input  [6:0] datain7;
input  [6:0] datain9;
input  [6:0] datain6;
input  [6:0] datain8;
output [6:0] out1;
input  [2:0] msb;
input  [6:0] datain1;
input  [6:0] datain3;
input  [6:0] datain4;
input  [6:0] datain2;
input  [6:0] datain5;
output [6:0] out2;
input  clock, reset;
  wire \dataout1[5] , \dataout1[3] , \dataout2[0] , \dataout7[4] ,
    \dataout9[2] , \dataout2[6] , \dataout2[4] , \dataout4[3] ,
    \dataout7[0] , \dataout9[6] , \dataout1[1] , \dataout4[1] ,
    \dataout9[4] , \dataout7[2] , \dataout2[2] , \dataout4[5] ,
    \dataout7[6] , \dataout9[0] , \dataout3[6] , \dataout3[2] ,
    \dataout5[1] , \dataout6[2] , \dataout8[4] , \dataout5[5] ,
    \dataout8[0] , \dataout6[6] , \dataout6[4] , \dataout1[6] ,
    \dataout1[4] , \dataout1[0] , \dataout8[2] , \dataout3[4] ,
    \dataout3[0] , \dataout5[3] , \dataout8[6] , \dataout6[0] ,
    \dataout3[5] , \dataout5[2] , \dataout6[1] , \dataout5[6] ,
    \dataout6[5] , \dataout8[3] , \dataout3[3] , \dataout3[1] ,
    \dataout8[1] , \dataout5[4] , \dataout2[3] , \dataout5[0] ,
    \dataout6[3] , \dataout8[5] , \dataout4[4] , \dataout9[1] ,
    \dataout2[5] , \dataout4[2] , \dataout4[0] , \dataout7[3] ,
```

```
   \dataout9[5] , \dataout7[1] , \dataout1[2] , \dataout4[6] ,
   \dataout9[3] , \dataout7[5] , \dataout2[1] , n8, n9, n10, n11, n12,
   n13, n14, n15, n16, n17, n18, n19, n20, n21, n22, n23, n24, n25, n26,
   n27, n28, n29, n30, n31, n32, n33, n34, n35, n36, n37, n38, n39, n40,
   n41, n42, n43, n44, n45, n46, n47, n48, n49, n50, n51, n52, n53;
memory_8 register1 ( .clk(clock), .resetn(reset), .Data_in(datain1),
   .Data_out({\dataout1[6] , \dataout1[5] , \dataout1[4] , \dataout1[3] ,
   \dataout1[2] , \dataout1[1] , \dataout1[0] }) );
memory_7 register2 ( .clk(clock), .resetn(reset), .Data_in(datain2),
   .Data_out({\dataout2[6] , \dataout2[5] , \dataout2[4] , \dataout2[3] ,
   \dataout2[2] , \dataout2[1] , \dataout2[0] }) );
memory_6 register3 ( .clk(clock), .resetn(reset), .Data_in(datain3),
   .Data_out({\dataout3[6] , \dataout3[5] , \dataout3[4] , \dataout3[3] ,
   \dataout3[2] , \dataout3[1] , \dataout3[0] }) );
memory_5 register4 ( .clk(clock), .resetn(reset), .Data_in(datain4),
   .Data_out({\dataout4[6] , \dataout4[5] , \dataout4[4] , \dataout4[3] ,
   \dataout4[2] , \dataout4[1] , \dataout4[0] }) );
memory_4 register5 ( .clk(clock), .resetn(reset), .Data_in(datain5),
   .Data_out({\dataout5[6] , \dataout5[5] , \dataout5[4] , \dataout5[3] ,
   \dataout5[2] , \dataout5[1] , \dataout5[0] }) );
memory_3 register6 ( .clk(clock), .resetn(reset), .Data_in(datain6),
   .Data_out({\dataout6[6] , \dataout6[5] , \dataout6[4] , \dataout6[3] ,
   \dataout6[2] , \dataout6[1] , \dataout6[0] }) );
memory_2 register7 ( .clk(clock), .resetn(reset), .Data_in(datain7),
   .Data_out({\dataout7[6] , \dataout7[5] , \dataout7[4] , \dataout7[3] ,
   \dataout7[2] , \dataout7[1] , \dataout7[0] }) );
memory_1 register9 ( .clk(clock), .resetn(reset), .Data_in(datain9),
   .Data_out({\dataout9[6] , \dataout9[5] , \dataout9[4] , \dataout9[3] ,
   \dataout9[2] , \dataout9[1] , \dataout9[0] }) );
memory_0 register8 ( .clk(clock), .resetn(reset), .Data_in(datain8),
   .Data_out({\dataout8[6] , \dataout8[5] , \dataout8[4] , \dataout8[3] ,
   \dataout8[2] , \dataout8[1] , \dataout8[0] }) );
buf3 U10 ( .Y(n8), .A(n49) );
buf3 U11 ( .Y(n9), .A(n50) );
buf3 U12 ( .Y(n10), .A(n48) );
buf3 U13 ( .Y(n11), .A(n51) );
buf3 U14 ( .Y(n12), .A(n47) );
buf3 U15 ( .Y(n13), .A(n52) );
buf3 U16 ( .Y(n14), .A(n46) );
buf3 U17 ( .Y(n15), .A(n53) );
nand21 U18 ( .Y(out1[0]), .A(n16), .B(n17) );
nand21 U19 ( .Y(out1[1]), .A(n18), .B(n19) );
nand21 U20 ( .Y(out1[2]), .A(n20), .B(n21) );
nand21 U21 ( .Y(out1[3]), .A(n22), .B(n23) );
nand21 U22 ( .Y(out1[4]), .A(n24), .B(n25) );
nand21 U23 ( .Y(out1[5]), .A(n26), .B(n27) );
nand21 U24 ( .Y(out1[6]), .A(n28), .B(n29) );
nand21 U25 ( .Y(out2[0]), .A(n30), .B(n31) );
nand21 U26 ( .Y(out2[1]), .A(n32), .B(n33) );
nand21 U27 ( .Y(out2[2]), .A(n34), .B(n35) );
nand21 U28 ( .Y(out2[3]), .A(n36), .B(n37) );
nand21 U29 ( .Y(out2[4]), .A(n38), .B(n39) );
nand21 U30 ( .Y(out2[5]), .A(n40), .B(n41) );
nand21 U31 ( .Y(out2[6]), .A(n42), .B(n43) );
```

```
inv1 U32 ( .Y(n44), .A(msb[0]) );
inv1 U33 ( .Y(n45), .A(msb[1]) );
and31 U34 ( .Y(n46), .A(msb[1]), .B(msb[0]), .C(msb[2]) );
and31 U35 ( .Y(n47), .A(msb[1]), .B(n44), .C(msb[2]) );
and31 U36 ( .Y(n48), .A(msb[0]), .B(n45), .C(msb[2]) );
and31 U37 ( .Y(n49), .A(n44), .B(n45), .C(msb[2]) );
nor31 U38 ( .Y(n50), .A(n44), .B(msb[2]), .C(n45) );
nor31 U39 ( .Y(n51), .A(msb[0]), .B(msb[2]), .C(n45) );
nor31 U40 ( .Y(n52), .A(msb[1]), .B(msb[2]), .C(n44) );
nor31 U41 ( .Y(n53), .A(msb[1]), .B(msb[2]), .C(msb[0]) );
aoi2222 U42 ( .Y(n43), .A(n46), .B(\dataout9[6] ), .C(n47), .D(
  \dataout8[6] ), .E(n48), .F(\dataout7[6] ), .G(n49), .H(\dataout6[6] )
  );
aoi2222 U43 ( .Y(n42), .A(n50), .B(\dataout5[6] ), .C(n51), .D(
  \dataout4[6] ), .E(n52), .F(\dataout3[6] ), .G(n53), .H(\dataout2[6] )
  );
aoi2222 U44 ( .Y(n41), .A(\dataout9[5] ), .B(n46), .C(\dataout8[5] ), .D(
  n47), .E(\dataout7[5] ), .F(n48), .G(\dataout6[5] ), .H(n49) );
aoi2222 U45 ( .Y(n40), .A(\dataout5[5] ), .B(n50), .C(\dataout4[5] ), .D(
  n51), .E(\dataout3[5] ), .F(n52), .G(\dataout2[5] ), .H(n53) );
aoi2222 U46 ( .Y(n39), .A(\dataout9[4] ), .B(n46), .C(\dataout8[4] ), .D(
  n47), .E(\dataout7[4] ), .F(n48), .G(\dataout6[4] ), .H(n49) );
aoi2222 U47 ( .Y(n38), .A(\dataout5[4] ), .B(n50), .C(\dataout4[4] ), .D(
  n51), .E(\dataout3[4] ), .F(n52), .G(\dataout2[4] ), .H(n53) );
aoi2222 U48 ( .Y(n37), .A(\dataout9[3] ), .B(n46), .C(\dataout8[3] ), .D(
  n47), .E(\dataout7[3] ), .F(n48), .G(\dataout6[3] ), .H(n49) );
aoi2222 U49 ( .Y(n36), .A(\dataout5[3] ), .B(n50), .C(\dataout4[3] ), .D(
  n51), .E(\dataout3[3] ), .F(n52), .G(\dataout2[3] ), .H(n53) );
aoi2222 U50 ( .Y(n35), .A(\dataout9[2] ), .B(n46), .C(\dataout8[2] ), .D(
  n47), .E(\dataout7[2] ), .F(n48), .G(\dataout6[2] ), .H(n49) );
aoi2222 U51 ( .Y(n34), .A(\dataout5[2] ), .B(n50), .C(\dataout4[2] ), .D(
  n51), .E(\dataout3[2] ), .F(n52), .G(\dataout2[2] ), .H(n53) );
aoi2222 U52 ( .Y(n33), .A(\dataout9[1] ), .B(n46), .C(\dataout8[1] ), .D(
  n47), .E(\dataout7[1] ), .F(n48), .G(\dataout6[1] ), .H(n49) );
aoi2222 U53 ( .Y(n32), .A(\dataout5[1] ), .B(n50), .C(\dataout4[1] ), .D(
  n51), .E(\dataout3[1] ), .F(n52), .G(\dataout2[1] ), .H(n53) );
aoi2222 U54 ( .Y(n31), .A(\dataout9[0] ), .B(n46), .C(\dataout8[0] ), .D(
  n47), .E(\dataout7[0] ), .F(n48), .G(\dataout6[0] ), .H(n49) );
aoi2222 U55 ( .Y(n30), .A(\dataout5[0] ), .B(n50), .C(\dataout4[0] ), .D(
  n51), .E(\dataout3[0] ), .F(n52), .G(\dataout2[0] ), .H(n53) );
aoi2222 U56 ( .Y(n29), .A(\dataout8[6] ), .B(n14), .C(\dataout7[6] ), .D(
  n12), .E(\dataout6[6] ), .F(n10), .G(\dataout5[6] ), .H(n8) );
aoi2222 U57 ( .Y(n28), .A(\dataout4[6] ), .B(n9), .C(\dataout3[6] ), .D(
  n11), .E(\dataout2[6] ), .F(n13), .G(\dataout1[6] ), .H(n15) );
aoi2222 U58 ( .Y(n27), .A(\dataout8[5] ), .B(n46), .C(\dataout7[5] ), .D(
  n47), .E(\dataout6[5] ), .F(n48), .G(\dataout5[5] ), .H(n49) );
aoi2222 U59 ( .Y(n26), .A(\dataout4[5] ), .B(n50), .C(\dataout3[5] ), .D(
  n51), .E(\dataout2[5] ), .F(n52), .G(\dataout1[5] ), .H(n53) );
aoi2222 U60 ( .Y(n25), .A(\dataout8[4] ), .B(n14), .C(\dataout7[4] ), .D(
  n12), .E(\dataout6[4] ), .F(n10), .G(\dataout5[4] ), .H(n8) );
aoi2222 U61 ( .Y(n24), .A(\dataout4[4] ), .B(n9), .C(\dataout3[4] ), .D(
  n11), .E(\dataout2[4] ), .F(n13), .G(\dataout1[4] ), .H(n15) );
aoi2222 U62 ( .Y(n23), .A(\dataout8[3] ), .B(n46), .C(\dataout7[3] ), .D(
  n47), .E(\dataout6[3] ), .F(n48), .G(\dataout5[3] ), .H(n49) );
```

```
  aoi2222 U63 ( .Y(n22), .A(\dataout4[3] ), .B(n50), .C(\dataout3[3] ), .D(
    n51), .E(\dataout2[3] ), .F(n52), .G(\dataout1[3] ), .H(n53) );
  aoi2222 U64 ( .Y(n21), .A(\dataout8[2] ), .B(n46), .C(\dataout7[2] ), .D(
    n47), .E(\dataout6[2] ), .F(n48), .G(\dataout5[2] ), .H(n49) );
  aoi2222 U65 ( .Y(n20), .A(\dataout4[2] ), .B(n50), .C(\dataout3[2] ), .D(
    n51), .E(\dataout2[2] ), .F(n52), .G(\dataout1[2] ), .H(n53) );
  aoi2222 U66 ( .Y(n19), .A(\dataout8[1] ), .B(n46), .C(\dataout7[1] ), .D(
    n47), .E(\dataout6[1] ), .F(n48), .G(\dataout5[1] ), .H(n49) );
  aoi2222 U67 ( .Y(n18), .A(\dataout4[1] ), .B(n50), .C(\dataout3[1] ), .D(
    n51), .E(\dataout2[1] ), .F(n52), .G(\dataout1[1] ), .H(n53) );
  aoi2222 U68 ( .Y(n17), .A(\dataout8[0] ), .B(n14), .C(\dataout7[0] ), .D(
    n12), .E(\dataout6[0] ), .F(n10), .G(\dataout5[0] ), .H(n8) );
  aoi2222 U69 ( .Y(n16), .A(\dataout4[0] ), .B(n9), .C(\dataout3[0] ), .D(
    n11), .E(\dataout2[0] ), .F(n13), .G(\dataout1[0] ), .H(n15) );
endmodule
```

Code 7

```
--This block is a 8bit subtractor
--This block is used to evaluate Yi+1 - Yi

library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;

entity ADD8 is
        port(   A,B     : in std_logic_vector(7 downto 0);
                Sum     : out std_logic_vector(7 downto 0));
end ADD8;

architecture RTL of ADD8 is

begin
Sum<=A-B;
end;
```

Code 8

```
--This is the test bench for 8bit adder/subtractor block

library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;

entity T_ADD8 is
end T_ADD8;

architecture  TEST of T_ADD8 is
  component ADD8
```

```
  port (  A,B: in std_logic_vector(7 downto 0);
          Sum : out std_logic_vector(7 downto 0));
  end component;

        signal A,B  : std_logic_vector(7 downto 0);
        signal Sum : std_logic_vector(7 downto 0);

begin
  ------------------------------------------------
  -- Instantiate UUT
  ------------------------------------------------
  UUT : ADD8 port map (A,B,Sum);
  ------------------------------------------------
  -- Stimulus:
  ------------------------------------------------
  STIMULUS: process
  begin
                  A<="00000000";B<="00000000";
                  wait for 10 ns;
                  A<="00010100";B<="00001010";
                  wait for 10 ns;
                  A<="00001010";B<="00010100";
                  wait for 10 ns;
                  A<="11100111";B<="11100111";
                  wait for 10 ns;
                  A<="11100111";B<="00100011";
                  wait for 10 ns;
                  A<="11000000";B<="11000000";
                  wait for 10 ns;
                  A<="11000000";B<="00111111";
                  wait;
  end process STIMULUS;
end TEST;
------------------------------------------------
-- Configuration...
------------------------------------------------
configuration CFG_T_ADD8 of T_ADD8 is
  for TEST
  end for;
end CFG_T_ADD8;
```

Code 9

```
module ADD8_DW01_sub_8_0 ( A, B, CI, DIFF, CO );
input  [7:0] A;
input  [7:0] B;
output [7:0] DIFF;
input  CI;
output CO;
   wire n50, n51, n52, n53, n54, n55, n56, n57, n58, n59, n60, n61, n62, n63,
      n64, n65, n66, n67, n68, n69, n70, n71, n72, n73, n74, n75, n76, n77,
      n78, n79, n80, n81, n82, n83, n84, n85, n86, n87;
```

```
  aoi12 U4 ( .Y(n50), .A(n53), .B1(n51), .B2(n52) );
  oai12 U5 ( .Y(DIFF[0]), .A(n55), .B1(B[0]), .B2(n54) );
  aoi12 U6 ( .Y(n56), .A(n59), .B1(n57), .B2(n58) );
  nor21 U7 ( .Y(n60), .A(n61), .B(n53) );
  nor21 U8 ( .Y(n62), .A(n63), .B(n64) );
  nor21 U9 ( .Y(n65), .A(n66), .B(n67) );
  nor21 U10 ( .Y(n68), .A(n59), .B(n69) );
  xor21 U11 ( .Y(DIFF[6]), .A(n60), .B(n51) );
  xnor21 U12 ( .Y(DIFF[4]), .A(n62), .B(n70) );
  xor21 U13 ( .Y(DIFF[3]), .A(n65), .B(n71) );
  xnor21 U14 ( .Y(DIFF[2]), .A(n68), .B(n72) );
  xnor21 U15 ( .Y(DIFF[1]), .A(n55), .B(n73) );
  inv1 U16 ( .Y(n74), .A(B[6]) );
  inv1 U17 ( .Y(n75), .A(B[4]) );
  inv1 U18 ( .Y(n76), .A(B[2]) );
  inv1 U19 ( .Y(n77), .A(A[1]) );
  inv1 U20 ( .Y(n78), .A(A[3]) );
  inv1 U21 ( .Y(n79), .A(A[5]) );
  inv1 U22 ( .Y(n54), .A(A[0]) );
  xor31 U23 ( .Y(DIFF[7]), .A(n50), .B(B[7]), .C(A[7]) );
  xnor31 U24 ( .Y(DIFF[5]), .A(n80), .B(A[5]), .C(B[5]) );
  oai12 U25 ( .Y(n80), .A(n81), .B1(n63), .B2(n70) );
  nand21 U26 ( .Y(n81), .A(A[4]), .B(n75) );
  inv1 U27 ( .Y(n64), .A(n81) );
  and21 U28 ( .Y(n67), .A(B[3]), .B(n78) );
  and21 U29 ( .Y(n72), .A(n58), .B(n57) );
  or21 U30 ( .Y(n57), .A(n77), .B(B[1]) );
  nand21 U31 ( .Y(n82), .A(A[2]), .B(n76) );
  inv1 U32 ( .Y(n69), .A(n82) );
  oai12 U33 ( .Y(n71), .A(n82), .B1(n59), .B2(n72) );
  and21 U34 ( .Y(n53), .A(A[6]), .B(n74) );
  nor21 U35 ( .Y(n66), .A(n78), .B(B[3]) );
  nor21 U36 ( .Y(n59), .A(n76), .B(A[2]) );
  nand21 U37 ( .Y(n55), .A(B[0]), .B(n54) );
  nand21 U38 ( .Y(n83), .A(B[1]), .B(n77) );
  nand21 U39 ( .Y(n58), .A(n55), .B(n83) );
  aoi12 U40 ( .Y(n84), .A(n67), .B1(n85), .B2(n82) );
  nor21 U41 ( .Y(n70), .A(n66), .B(n84) );
  nor21 U42 ( .Y(n63), .A(n75), .B(A[4]) );
  oai12 U43 ( .Y(n86), .A(n80), .B1(A[5]), .B2(n87) );
  oai12 U44 ( .Y(n51), .A(n86), .B1(B[5]), .B2(n79) );
  nor21 U45 ( .Y(n61), .A(n74), .B(A[6]) );
  inv1 U46 ( .Y(n52), .A(n61) );
  nand21 U47 ( .Y(n73), .A(n57), .B(n83) );
  inv1 U48 ( .Y(n85), .A(n56) );
  inv1 U49 ( .Y(n87), .A(B[5]) );
endmodule


module ADD8 ( A, B, Sum );
input  [7:0] A;
input  [7:0] B;
output [7:0] Sum;
  ADD8_DW01_sub_8_0 \sub_18/minus/minus  ( .A(A), .B(B), .CI(1'b0), .DIFF(
```

```
    Sum) );
Endmodule
```

Code 10

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;

entity MUL14x8 is
        port(   A       : in std_logic_vector(13 downto 0);
                B       : in std_logic_vector(7 downto 0);
                Prod    : out std_logic_vector(21 downto 0));
end MUL14x8;

architecture RTL of MUL14x8 is
signal sign       : std_logic;
signal FirstProd : std_logic_vector(21 downto 0);
signal Invert     : std_logic_vector(7 downto 0);
signal InvertProd : std_logic_vector(21 downto 0);

begin
        process(A,B,Invert)
        begin
                if B(7)='1' then
                        sign<='1';
                        Invert<=not B + "00000001";
                else
                        sign<='0';
                        Invert<=B;
                end if;
        end process;
FirstProd<=A*Invert;
        process(FirstProd,InvertProd,sign)
        begin
                if sign='1' then
                        InvertProd<= not FirstProd + "00000000000000000000001";
                else
                        InvertProd<=FirstProd;
                end if;
        end process;
Prod<=InvertProd;
end;
```

Code 11

```
--------------------------------------------------------------------
---This is the test bench for the 14x8 multiplier.
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;
```

```
entity T_MUL14x8 is
end T_MUL14x8;

architecture  TEST of T_MUL14x8 is
        component MUL14x8
        port(   A       : in std_logic_vector(13 downto 0);
                B       : in std_logic_vector(7 downto 0);
                Prod    : out std_logic_vector(21 downto 0));
        end component;

        signal A : std_logic_vector(13 downto 0);
        signal B : std_logic_vector(7 downto 0);
        signal Prod       : std_logic_vector(21 downto 0);

begin
  ---------------------------------------------------
  -- Instantiate UUT
  ---------------------------------------------------
  UUT : MUL14x8 port map (A,B,Prod);
  ---------------------------------------------------
  -- Stimulus:
  ---------------------------------------------------
  STIMULUS: process
  begin
                A<="00000000000000";B<="00000000";
                wait for 10 ns;
                A<="00000000000100";B<="00000010";
                wait for 10 ns;
                A<="00000000000100";B<="10000000";
                wait for 10 ns;
                A<="01111111111111";B<="10000000";
                wait for 10 ns;
                A<="01000000000000";B<="01000000";
                wait for 10 ns;
                A<="01111110000000";B<="00000001";
                wait for 10 ns;
                A<="01111111111111";B<="11111111";
                wait;
  end process STIMULUS;
end TEST;

---------------------------------------------------
-- Configuration...
---------------------------------------------------
configuration CFG_T_MUL14x8 of T_MUL14x8 is
  for TEST
  end for;
end CFG_T_MUL14x8;
```

Code 12

```
module MUL14x8_DW01_add_20_0 ( A, B, CI, SUM, CO );
```

```
input  [19:0] A;
input  [19:0] B;
output [19:0] SUM;
input  CI;
output CO;
   wire n49, n59, n60, n61, n62, n63, n64, n65, n66, n67, n68, n69, n70, n71,
      n72, n73, n74, n75, n76, n77, n80, n81, n82, n83;
   and21 U5 ( .Y(n49), .A(A[13]), .B(B[13]) );
   buf2 U6 ( .Y(SUM[7]), .A(A[7]) );
   buf2 U7 ( .Y(SUM[12]), .A(A[12]) );
   buf2 U8 ( .Y(SUM[3]), .A(A[3]) );
   buf2 U9 ( .Y(SUM[10]), .A(A[10]) );
   buf2 U10 ( .Y(SUM[8]), .A(A[8]) );
   buf2 U11 ( .Y(SUM[0]), .A(A[0]) );
   buf2 U12 ( .Y(SUM[11]), .A(A[11]) );
   buf2 U13 ( .Y(SUM[9]), .A(A[9]) );
   buf2 U14 ( .Y(SUM[2]), .A(A[2]) );
   inv1 U15 ( .Y(SUM[1]), .A(n59) );
   inv1 U16 ( .Y(SUM[5]), .A(n60) );
   xor21 U17 ( .Y(SUM[19]), .A(B[19]), .B(n63) );
   inv1 U18 ( .Y(n76), .A(A[4]) );
   inv1 U19 ( .Y(n60), .A(A[5]) );
   inv1 U20 ( .Y(n77), .A(A[6]) );
   aoi12 U21 ( .Y(n61), .A(B[18]), .B1(A[18]), .B2(n62) );
   aoi12 U22 ( .Y(n63), .A(n61), .B1(n64), .B2(n65) );
   nor21 U23 ( .Y(SUM[13]), .A(n66), .B(n49) );
   xor31 U24 ( .Y(SUM[17]), .A(n67), .B(B[17]), .C(A[17]) );
   xor31 U25 ( .Y(SUM[16]), .A(n68), .B(B[16]), .C(n69) );
   xor31 U26 ( .Y(SUM[15]), .A(n70), .B(A[15]), .C(n71) );
   xor31 U27 ( .Y(SUM[14]), .A(n49), .B(A[14]), .C(B[14]) );
   inv1 U28 ( .Y(n59), .A(A[1]) );
   inv1 U29 ( .Y(n71), .A(B[15]) );
   inv1 U30 ( .Y(n72), .A(A[15]) );
   aoi12 U31 ( .Y(n68), .A(n73), .B1(B[15]), .B2(A[15]) );
   inv1 U32 ( .Y(n69), .A(A[16]) );
   oai12 U33 ( .Y(n67), .A(n74), .B1(n68), .B2(n69) );
   oai12 U34 ( .Y(n64), .A(n75), .B1(A[17]), .B2(n67) );
   inv1 U35 ( .Y(n65), .A(A[18]) );
   xor31 U36 ( .Y(SUM[18]), .A(n65), .B(B[18]), .C(n64) );
   inv1 U37 ( .Y(SUM[4]), .A(n76) );
   inv1 U38 ( .Y(SUM[6]), .A(n77) );
   nor21 U39 ( .Y(n66), .A(A[13]), .B(B[13]) );
   oai12 U40 ( .Y(n80), .A(n49), .B1(B[14]), .B2(A[14]) );
   aoi12 U41 ( .Y(n70), .A(n81), .B1(B[14]), .B2(A[14]) );
   aoi12 U42 ( .Y(n73), .A(n70), .B1(n71), .B2(n72) );
   inv1 U43 ( .Y(n82), .A(n68) );
   oai12 U44 ( .Y(n74), .A(B[16]), .B1(A[16]), .B2(n82) );
   aoi12 U45 ( .Y(n83), .A(B[17]), .B1(n67), .B2(A[17]) );
   inv1 U46 ( .Y(n62), .A(n64) );
   inv1 U47 ( .Y(n81), .A(n80) );
   inv1 U48 ( .Y(n75), .A(n83) );
endmodule
```

```
module MUL14x8_DW02_mult_14_8_0 ( A, B, TC, PRODUCT );
input  [13:0] A;
input  [7:0] B;
output [21:0] PRODUCT;
input  TC;
  wire \ab[13][0] , \ab[9][1] , \ab[3][7] , \SUMB[12][1] , \CLA_SUM[15] ,
    \SUMB[5][2] , \CARRYB[6][3] , \CARRYB[13][6] , \CARRYB[2][1] ,
    \ab[7][5] , \ab[12][3] , \ab[8][2] , \SUMB[13][2] , \CARRYB[9][4] ,
    \ab[6][6] , \CARRYB[3][2] , \CARRYB[12][5] , \ab[2][4] , \SUMB[4][1] ,
    \CARRYB[7][0] , \SUMB[1][4] , \CARRYB[13][2] , \CARRYB[2][5] ,
    \ab[13][4] , \ab[9][5] , \ab[7][1] , \ab[3][3] , \SUMB[5][6] ,
    \SUMB[12][5] , \CLA_SUM[11] , \ab[6][2] , \ab[2][0] , \SUMB[4][5] ,
    \CARRYB[8][3] , \CARRYB[7][4] , \CLA_SUM[18] , \CARRYB[12][1] ,
    \CARRYB[3][6] , \ab[12][7] , \ab[8][6] , \CARRYB[9][0] , \SUMB[13][6] ,
    \ab[12][5] , \CARRYB[9][2] , \ab[8][4] , \ab[2][2] , \SUMB[13][4] ,
    \CARRYB[7][6] , \CARRYB[3][4] , \ab[13][6] , \ab[6][0] ,
    \CARRYB[12][3] , \CLA_SUM[13] , \CLA_SUM[20] , \ab[9][7] ,
    \CARRYB[8][1] , \ab[7][3] , \SUMB[1][6] , \CARRYB[13][0] , \ab[6][4] ,
    \ab[3][1] , \SUMB[5][4] , \CARRYB[6][5] , \CARRYB[3][0] , \ab[2][6] ,
    \SUMB[4][3] , \CARRYB[7][2] , \ab[12][1] , \ab[8][0] , \CARRYB[9][6] ,
    \ab[13][2] , \ab[7][7] , \ab[3][5] , \CARRYB[6][1] , \SUMB[1][2] ,
    \CARRYB[13][4] , \CARRYB[2][3] , \CARRYB[8][5] , \ab[11][4] ,
    \ab[9][3] , \SUMB[12][3] , \CLA_SUM[17] , \SUMB[9][2] ,
    \CLA_CARRY[15] , \ab[5][1] , \ab[1][3] , \SUMB[10][5] , \SUMB[7][6] ,
    \SUMB[3][4] , \CARRYB[11][2] , \ab[10][7] , \SUMB[11][6] , \ab[4][2] ,
    \SUMB[8][1] , \CLA_SUM[7] , \CARRYB[10][1] , \CARRYB[1][6] ,
    \SUMB[6][5] , \CARRYB[5][4] , \ab[11][2] , \ab[11][0] , \ab[5][5] ,
    \ab[1][7] , \CARRYB[11][6] , \SUMB[10][1] , \SUMB[7][2] ,
    \CARRYB[4][3] , \ab[10][3] , \ab[4][6] , \ab[0][4] , \SUMB[9][6] ,
    \SUMB[6][1] , \CARRYB[5][0] , \CLA_SUM[3] , \SUMB[2][3] ,
    \CLA_CARRY[18] , \CARRYB[10][5] , \CARRYB[1][2] , \SUMB[8][5] ,
    \ab[10][1] , \SUMB[11][2] , \ab[4][4] , \ab[0][6] , \SUMB[6][3] ,
    \CARRYB[5][2] , \SUMB[10][3] , \SUMB[2][1] , \CARRYB[1][0] ,
    \SUMB[9][4] , \CLA_CARRY[20] , \ab[5][7] , \ab[4][0] , \ab[1][5] ,
    \SUMB[3][2] , \CARRYB[11][4] , \CLA_SUM[8] , \CARRYB[4][1] ,
    \ab[0][2] , \SUMB[2][5] , \CARRYB[10][3] , \CARRYB[1][4] ,
    \CARRYB[5][6] , \SUMB[11][4] , \ab[10][5] , \ab[5][3] , \ab[1][1] ,
    \SUMB[8][3] , \CLA_SUM[5] , \SUMB[7][4] , \CARRYB[4][5] , \SUMB[3][6] ,
    \CARRYB[11][0] , \ab[11][6] , \CLA_CARRY[17] , \ab[10][4] , \ab[4][1] ,
    \ab[0][3] , \SUMB[2][4] , \CARRYB[10][2] , \CARRYB[1][5] ,
    \SUMB[6][6] , \SUMB[11][5] , \SUMB[8][2] , \CLA_SUM[4] , \ab[1][0] ,
    \SUMB[7][5] , \CARRYB[4][4] , \ab[11][7] , \ab[5][2] , \CLA_CARRY[16] ,
    \SUMB[9][1] , \CARRYB[11][1] , \SUMB[10][6] , \ab[11][3] , \ab[10][0] ,
    \SUMB[11][1] , \SUMB[8][6] , \ab[4][5] , \ab[0][7] , \CARRYB[5][3] ,
    \SUMB[6][2] , \CARRYB[10][6] , \CARRYB[1][1] , \ab[5][6] ,
    \SUMB[10][2] , \SUMB[9][5] , \SUMB[3][3] , \CARRYB[11][5] ,
    \CLA_SUM[9] , \ab[5][4] , \ab[1][4] , \SUMB[7][1] , \CARRYB[4][0] ,
    \SUMB[3][1] , \ab[11][5] , \ab[11][1] , \ab[1][6] , \SUMB[7][3] ,
    \CARRYB[4][2] , \ab[10][2] , \ab[4][7] , \ab[0][5] , \CARRYB[5][1] ,
    \CLA_CARRY[19] , \SUMB[2][2] , \CARRYB[10][4] , \CARRYB[1][3] ,
    \SUMB[8][4] , \SUMB[11][3] , \CLA_SUM[2] , \SUMB[10][4] , \SUMB[9][3] ,
    \CLA_CARRY[14] , \ab[1][2] , \CARRYB[4][6] , \ab[10][6] , \ab[5][0] ,
    \SUMB[3][5] , \CARRYB[11][3] , \ab[12][0] , \ab[6][5] , \ab[4][3] ,
    \CLA_SUM[6] , \ab[0][1] , \SUMB[2][6] , \CARRYB[10][0] ,
```

```
    \CARRYB[5][5] , \SUMB[6][4] , \CARRYB[12][6] , \CARRYB[3][1] ,
    \ab[2][7] , \SUMB[4][2] , \CARRYB[7][3] , \ab[8][1] , \SUMB[13][1] ,
    \ab[13][3] , \ab[7][6] , \ab[3][4] , \SUMB[5][1] , \CARRYB[6][0] ,
    \SUMB[1][3] , \CARRYB[13][5] , \CARRYB[2][2] , \SUMB[12][2] ,
    \CARRYB[8][4] , \CLA_SUM[16] , \ab[9][2] , \CARRYB[9][3] , \ab[12][4] ,
    \SUMB[13][5] , \ab[8][5] , \ab[6][1] , \ab[2][3] , \SUMB[4][6] ,
    \CARRYB[12][2] , \CARRYB[3][5] , \ab[9][6] , \SUMB[12][6] ,
    \CLA_SUM[12] , \CARRYB[8][0] , \CARRYB[13][1] , \ab[7][2] ,
    \CARRYB[2][6] , \ab[3][0] , \SUMB[5][5] , \CARRYB[6][4] ,
    \CARRYB[2][4] , \ab[13][5] , \ab[9][4] , \ab[7][0] , \SUMB[1][5] ,
    \CARRYB[13][3] , \ab[3][2] , \CARRYB[6][6] , \ab[6][3] , \ab[2][1] ,
    \SUMB[12][4] , \CLA_SUM[10] , \SUMB[4][4] , \CARRYB[8][2] ,
    \CARRYB[7][5] , \CLA_SUM[19] , \CARRYB[12][0] , \CARRYB[9][1] ,
    \ab[13][1] , \ab[12][6] , \ab[8][7] , \SUMB[13][7] , \ab[9][0] ,
    \CARRYB[8][6] , \CLA_SUM[14] , \ab[7][4] , \ab[3][6] , \SUMB[5][3] ,
    \CARRYB[6][2] , \SUMB[1][1] , \CARRYB[2][0] , \ab[12][2] , \ab[8][3] ,
    \SUMB[13][3] , \ab[6][7] , \CARRYB[9][5] , \CARRYB[3][3] ,
    \CARRYB[12][4] , \ab[2][5] , \CARRYB[7][1] , n10, n11, n12, n13, n14,
    n15, n16, n17, n31, n32, n33, n34, n35, n36, n37, n38, n39, n40, n41,
    n42, n43, n44, n45, n46, n47, n48, n50, n51, n52, n53;
inv3 U5 ( .Y(n10), .A(B[1]) );
inv3 U6 ( .Y(n11), .A(B[4]) );
inv3 U7 ( .Y(n12), .A(B[7]) );
inv3 U8 ( .Y(n13), .A(B[6]) );
inv3 U9 ( .Y(n14), .A(B[0]) );
inv3 U10 ( .Y(n15), .A(B[5]) );
inv3 U11 ( .Y(n16), .A(B[2]) );
inv3 U12 ( .Y(n17), .A(B[3]) );
and21 U13 ( .Y(\CLA_CARRY[18] ), .A(\CARRYB[13][4] ), .B(\SUMB[13][5] ) );
xor21 U14 ( .Y(\CLA_SUM[18] ), .A(\CARRYB[13][4] ), .B(\SUMB[13][5] ) );
and21 U27 ( .Y(\CLA_CARRY[14] ), .A(\CARRYB[13][0] ), .B(\SUMB[13][1] ) );
xor21 U28 ( .Y(\CLA_SUM[14] ), .A(\CARRYB[13][0] ), .B(\SUMB[13][1] ) );
and21 U29 ( .Y(\CARRYB[1][0] ), .A(\ab[1][0] ), .B(\ab[0][1] ) );
xor21 U30 ( .Y(PRODUCT[1]), .A(\ab[1][0] ), .B(\ab[0][1] ) );
and21 U31 ( .Y(\CLA_CARRY[15] ), .A(\CARRYB[13][1] ), .B(\SUMB[13][2] ) );
xor21 U32 ( .Y(\CLA_SUM[15] ), .A(\CARRYB[13][1] ), .B(\SUMB[13][2] ) );
and21 U33 ( .Y(\CARRYB[1][1] ), .A(\ab[1][1] ), .B(\ab[0][2] ) );
xor21 U34 ( .Y(\SUMB[1][1] ), .A(\ab[1][1] ), .B(\ab[0][2] ) );
and21 U35 ( .Y(\CLA_CARRY[17] ), .A(\CARRYB[13][3] ), .B(\SUMB[13][4] ) );
xor21 U36 ( .Y(\CLA_SUM[17] ), .A(\CARRYB[13][3] ), .B(\SUMB[13][4] ) );
and21 U37 ( .Y(\CLA_CARRY[16] ), .A(\CARRYB[13][2] ), .B(\SUMB[13][3] ) );
xor21 U38 ( .Y(\CLA_SUM[16] ), .A(\CARRYB[13][2] ), .B(\SUMB[13][3] ) );
and21 U39 ( .Y(\CARRYB[1][4] ), .A(\ab[1][4] ), .B(\ab[0][5] ) );
xor21 U40 ( .Y(\SUMB[1][4] ), .A(\ab[1][4] ), .B(\ab[0][5] ) );
and21 U41 ( .Y(\CARRYB[1][3] ), .A(\ab[1][3] ), .B(\ab[0][4] ) );
xor21 U42 ( .Y(\SUMB[1][3] ), .A(\ab[1][3] ), .B(\ab[0][4] ) );
and21 U43 ( .Y(\CARRYB[1][2] ), .A(\ab[1][2] ), .B(\ab[0][3] ) );
xor21 U44 ( .Y(\SUMB[1][2] ), .A(\ab[1][2] ), .B(\ab[0][3] ) );
and21 U45 ( .Y(\CLA_CARRY[19] ), .A(\CARRYB[13][5] ), .B(\SUMB[13][6] ) );
xor21 U46 ( .Y(\CLA_SUM[19] ), .A(\CARRYB[13][5] ), .B(\SUMB[13][6] ) );
and21 U47 ( .Y(\CARRYB[1][5] ), .A(\ab[1][5] ), .B(\ab[0][6] ) );
xor21 U48 ( .Y(\SUMB[1][5] ), .A(\ab[1][5] ), .B(\ab[0][6] ) );
and21 U50 ( .Y(\CLA_CARRY[20] ), .A(\CARRYB[13][6] ), .B(\SUMB[13][7] ) );
xor21 U51 ( .Y(\CLA_SUM[20] ), .A(\CARRYB[13][6] ), .B(\SUMB[13][7] ) );
```

```
and21 U52 ( .Y(\CARRYB[1][6] ), .A(\ab[1][6] ), .B(\ab[0][7] ) );
xor21 U53 ( .Y(\SUMB[1][6] ), .A(\ab[1][6] ), .B(\ab[0][7] ) );
nor21 U54 ( .Y(\ab[12][2] ), .A(n31), .B(n32) );
nor21 U55 ( .Y(\ab[8][0] ), .A(n33), .B(n34) );
nor21 U56 ( .Y(\ab[2][7] ), .A(n35), .B(n36) );
nor21 U57 ( .Y(\ab[0][1] ), .A(n37), .B(n38) );
nor21 U58 ( .Y(\ab[9][0] ), .A(n39), .B(n14) );
nor21 U59 ( .Y(\ab[3][7] ), .A(n35), .B(n40) );
nor21 U60 ( .Y(\ab[1][1] ), .A(n37), .B(n41) );
nor21 U61 ( .Y(\ab[7][2] ), .A(n31), .B(n42) );
nor21 U62 ( .Y(\ab[2][4] ), .A(n43), .B(n36) );
nor21 U63 ( .Y(\ab[13][6] ), .A(n44), .B(n45) );
nor21 U64 ( .Y(\ab[3][4] ), .A(n43), .B(n40) );
nor21 U65 ( .Y(\ab[13][1] ), .A(n37), .B(n45) );
nor21 U66 ( .Y(\ab[12][5] ), .A(n46), .B(n32) );
nor21 U67 ( .Y(\ab[11][0] ), .A(n33), .B(n47) );
nor21 U68 ( .Y(\ab[10][0] ), .A(n33), .B(n48) );
nor21 U69 ( .Y(\ab[6][5] ), .A(n46), .B(n50) );
nor21 U70 ( .Y(\ab[3][3] ), .A(n51), .B(n40) );
nor21 U71 ( .Y(\ab[7][5] ), .A(n46), .B(n42) );
nor21 U72 ( .Y(\ab[5][0] ), .A(n33), .B(n52) );
nor21 U73 ( .Y(\ab[4][0] ), .A(n33), .B(n53) );
nor21 U74 ( .Y(\ab[3][2] ), .A(n31), .B(n40) );
nor21 U75 ( .Y(\ab[2][3] ), .A(n51), .B(n36) );
nor21 U76 ( .Y(\ab[1][6] ), .A(n44), .B(n41) );
nor21 U77 ( .Y(\ab[0][6] ), .A(n44), .B(n38) );
nor21 U78 ( .Y(\ab[10][1] ), .A(n37), .B(n48) );
nor21 U79 ( .Y(\ab[7][4] ), .A(n43), .B(n42) );
nor21 U80 ( .Y(\ab[5][2] ), .A(n31), .B(n52) );
nor21 U81 ( .Y(\ab[13][0] ), .A(n33), .B(n45) );
nor21 U82 ( .Y(\ab[11][1] ), .A(n37), .B(n47) );
nor21 U83 ( .Y(\ab[6][4] ), .A(n43), .B(n50) );
nor21 U84 ( .Y(\ab[12][7] ), .A(n35), .B(n32) );
nor21 U85 ( .Y(\ab[12][4] ), .A(n43), .B(n32) );
nor21 U86 ( .Y(\ab[8][6] ), .A(n44), .B(n34) );
nor21 U87 ( .Y(\ab[7][7] ), .A(n35), .B(n42) );
nor21 U88 ( .Y(\ab[5][1] ), .A(n37), .B(n52) );
nor21 U89 ( .Y(\ab[6][7] ), .A(n35), .B(n50) );
nor21 U90 ( .Y(\ab[4][1] ), .A(n37), .B(n53) );
nor21 U91 ( .Y(\ab[9][6] ), .A(n39), .B(n13) );
nor21 U92 ( .Y(\ab[9][1] ), .A(n39), .B(n10) );
nor21 U93 ( .Y(\ab[12][3] ), .A(n51), .B(n32) );
nor21 U94 ( .Y(\ab[4][6] ), .A(n44), .B(n53) );
nor21 U95 ( .Y(\ab[1][0] ), .A(n33), .B(n41) );
nor21 U96 ( .Y(\ab[5][6] ), .A(n44), .B(n52) );
nor21 U97 ( .Y(PRODUCT[0]), .A(n33), .B(n38) );
nor21 U98 ( .Y(\ab[8][1] ), .A(n37), .B(n34) );
nor21 U99 ( .Y(\ab[11][6] ), .A(n44), .B(n47) );
nor21 U100 ( .Y(\ab[6][3] ), .A(n51), .B(n50) );
nor21 U101 ( .Y(\ab[3][5] ), .A(n46), .B(n40) );
nor21 U102 ( .Y(\ab[12][1] ), .A(n37), .B(n32) );
nor21 U103 ( .Y(\ab[10][6] ), .A(n44), .B(n48) );
nor21 U104 ( .Y(\ab[1][2] ), .A(n31), .B(n41) );
nor21 U105 ( .Y(\ab[8][3] ), .A(n51), .B(n34) );
```

```
nor21 U106 ( .Y(\ab[7][3] ), .A(n51), .B(n42) );
nor21 U107 ( .Y(\ab[2][5] ), .A(n46), .B(n36) );
nor21 U108 ( .Y(\ab[4][2] ), .A(n31), .B(n53) );
nor21 U109 ( .Y(\ab[5][4] ), .A(n43), .B(n52) );
nor21 U110 ( .Y(\ab[4][4] ), .A(n43), .B(n53) );
nor21 U111 ( .Y(\ab[13][5] ), .A(n46), .B(n45) );
nor21 U112 ( .Y(\ab[11][7] ), .A(n35), .B(n47) );
nor21 U113 ( .Y(\ab[10][4] ), .A(n43), .B(n48) );
nor21 U114 ( .Y(\ab[9][3] ), .A(n39), .B(n17) );
nor21 U115 ( .Y(\ab[7][1] ), .A(n37), .B(n42) );
nor21 U116 ( .Y(\ab[5][7] ), .A(n35), .B(n52) );
nor21 U117 ( .Y(\ab[6][1] ), .A(n37), .B(n50) );
nor21 U118 ( .Y(\ab[4][7] ), .A(n35), .B(n53) );
nor21 U119 ( .Y(\ab[11][4] ), .A(n43), .B(n47) );
nor21 U120 ( .Y(\ab[10][7] ), .A(n35), .B(n48) );
nor21 U121 ( .Y(\ab[9][7] ), .A(n39), .B(n12) );
nor21 U122 ( .Y(\ab[13][2] ), .A(n31), .B(n45) );
nor21 U123 ( .Y(\ab[11][3] ), .A(n51), .B(n47) );
nor21 U124 ( .Y(\ab[6][6] ), .A(n44), .B(n50) );
nor21 U125 ( .Y(\ab[3][0] ), .A(n33), .B(n40) );
nor21 U126 ( .Y(\ab[12][6] ), .A(n44), .B(n32) );
nor21 U127 ( .Y(\ab[10][3] ), .A(n51), .B(n48) );
nor21 U128 ( .Y(\ab[7][6] ), .A(n44), .B(n42) );
nor21 U129 ( .Y(\ab[2][0] ), .A(n33), .B(n36) );
nor21 U130 ( .Y(\ab[9][4] ), .A(n39), .B(n11) );
nor21 U131 ( .Y(\ab[8][7] ), .A(n35), .B(n34) );
nor21 U132 ( .Y(\ab[4][3] ), .A(n51), .B(n53) );
nor21 U133 ( .Y(\ab[1][5] ), .A(n46), .B(n41) );
nor21 U134 ( .Y(\ab[13][3] ), .A(n51), .B(n45) );
nor21 U135 ( .Y(\ab[11][2] ), .A(n31), .B(n47) );
nor21 U136 ( .Y(\ab[10][2] ), .A(n16), .B(n48) );
nor21 U137 ( .Y(\ab[8][4] ), .A(n43), .B(n34) );
nor21 U138 ( .Y(\ab[5][3] ), .A(n17), .B(n52) );
nor21 U139 ( .Y(\ab[0][5] ), .A(n46), .B(n38) );
nor21 U140 ( .Y(\ab[0][2] ), .A(n31), .B(n38) );
nor21 U141 ( .Y(\ab[2][1] ), .A(n10), .B(n36) );
nor21 U142 ( .Y(\ab[0][7] ), .A(n35), .B(n38) );
nor21 U143 ( .Y(\ab[3][1] ), .A(n37), .B(n40) );
nor21 U144 ( .Y(\ab[1][7] ), .A(n12), .B(n41) );
nor21 U145 ( .Y(\ab[8][5] ), .A(n46), .B(n34) );
nor21 U146 ( .Y(\ab[0][4] ), .A(n43), .B(n38) );
nor21 U147 ( .Y(\SUMB[13][7] ), .A(n35), .B(n45) );
nor21 U148 ( .Y(\ab[6][2] ), .A(n31), .B(n50) );
nor21 U149 ( .Y(\ab[9][5] ), .A(n39), .B(n15) );
nor21 U150 ( .Y(\ab[1][4] ), .A(n11), .B(n41) );
nor21 U151 ( .Y(\ab[12][0] ), .A(n14), .B(n32) );
nor21 U152 ( .Y(\ab[9][2] ), .A(n39), .B(n16) );
nor21 U153 ( .Y(\ab[4][5] ), .A(n46), .B(n53) );
nor21 U154 ( .Y(\ab[1][3] ), .A(n51), .B(n41) );
nor21 U155 ( .Y(\ab[8][2] ), .A(n31), .B(n34) );
nor21 U156 ( .Y(\ab[5][5] ), .A(n46), .B(n52) );
nor21 U157 ( .Y(\ab[2][2] ), .A(n31), .B(n36) );
nor21 U158 ( .Y(\ab[0][3] ), .A(n51), .B(n38) );
nor21 U159 ( .Y(\ab[13][4] ), .A(n43), .B(n45) );
```

```
nor21 U160 ( .Y(\ab[6][0] ), .A(n33), .B(n50) );
nor21 U161 ( .Y(\ab[3][6] ), .A(n13), .B(n40) );
nor21 U162 ( .Y(\ab[11][5] ), .A(n15), .B(n47) );
nor21 U163 ( .Y(\ab[10][5] ), .A(n46), .B(n48) );
nor21 U164 ( .Y(\ab[7][0] ), .A(n33), .B(n42) );
nor21 U165 ( .Y(\ab[2][6] ), .A(n44), .B(n36) );
inv1 U166 ( .Y(n39), .A(A[9]) );
inv1 U167 ( .Y(n35), .A(B[7]) );
inv1 U168 ( .Y(n44), .A(B[6]) );
inv1 U169 ( .Y(n46), .A(B[5]) );
inv1 U170 ( .Y(n43), .A(B[4]) );
inv1 U171 ( .Y(n51), .A(B[3]) );
inv1 U172 ( .Y(n31), .A(B[2]) );
inv1 U173 ( .Y(n37), .A(B[1]) );
inv1 U174 ( .Y(n33), .A(B[0]) );
inv1 U175 ( .Y(n34), .A(A[8]) );
inv1 U176 ( .Y(n42), .A(A[7]) );
inv1 U177 ( .Y(n50), .A(A[6]) );
inv1 U178 ( .Y(n52), .A(A[5]) );
inv1 U179 ( .Y(n53), .A(A[4]) );
inv1 U180 ( .Y(n40), .A(A[3]) );
inv1 U181 ( .Y(n36), .A(A[2]) );
inv1 U182 ( .Y(n41), .A(A[1]) );
inv1 U183 ( .Y(n45), .A(A[13]) );
inv1 U184 ( .Y(n32), .A(A[12]) );
inv1 U185 ( .Y(n47), .A(A[11]) );
inv1 U186 ( .Y(n48), .A(A[10]) );
inv1 U187 ( .Y(n38), .A(A[0]) );
MUL14x8_DW01_add_20_0 FS ( .A({1'b0, \CLA_SUM[20] , \CLA_SUM[19] ,
  \CLA_SUM[18] , \CLA_SUM[17] , \CLA_SUM[16] , \CLA_SUM[15] ,
  \CLA_SUM[14] , \CLA_SUM[13] , \CLA_SUM[12] , \CLA_SUM[11] ,
  \CLA_SUM[10] , \CLA_SUM[9] , \CLA_SUM[8] , \CLA_SUM[7] , \CLA_SUM[6] ,
  \CLA_SUM[5] , \CLA_SUM[4] , \CLA_SUM[3] , \CLA_SUM[2] }), .B({
  \CLA_CARRY[20] , \CLA_CARRY[19] , \CLA_CARRY[18] , \CLA_CARRY[17] ,
  \CLA_CARRY[16] , \CLA_CARRY[15] , \CLA_CARRY[14] , 1'b0, 1'b0, 1'b0,
  1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0, 1'b0}), .CI(1'b0
  ), .SUM(PRODUCT[21:2]) );
fadd S2_7_4 ( .S(\SUMB[7][4] ), .CO(\CARRYB[7][4] ), .CI(\SUMB[6][5] ),
  .B(\CARRYB[6][4] ), .A(\ab[7][4] ) );
fadd S2_2_2 ( .S(\SUMB[2][2] ), .CO(\CARRYB[2][2] ), .CI(\SUMB[1][3] ),
  .B(\CARRYB[1][2] ), .A(\ab[2][2] ) );
fadd S2_12_1 ( .S(\SUMB[12][1] ), .CO(\CARRYB[12][1] ), .CI(\SUMB[11][2] ),
  .B(\CARRYB[11][1] ), .A(\ab[12][1] ) );
fadd S2_3_2 ( .S(\SUMB[3][2] ), .CO(\CARRYB[3][2] ), .CI(\SUMB[2][3] ),
  .B(\CARRYB[2][2] ), .A(\ab[3][2] ) );
fadd S4_2 ( .S(\SUMB[13][2] ), .CO(\CARRYB[13][2] ), .CI(\SUMB[12][3] ),
  .B(\CARRYB[12][2] ), .A(\ab[13][2] ) );
fadd S2_6_4 ( .S(\SUMB[6][4] ), .CO(\CARRYB[6][4] ), .CI(\SUMB[5][5] ),
  .B(\CARRYB[5][4] ), .A(\ab[6][4] ) );
fadd S2_5_1 ( .S(\SUMB[5][1] ), .CO(\CARRYB[5][1] ), .CI(\SUMB[4][2] ),
  .B(\CARRYB[4][1] ), .A(\ab[5][1] ) );
fadd S2_10_4 ( .S(\SUMB[10][4] ), .CO(\CARRYB[10][4] ), .CI(\SUMB[9][5] ),
  .B(\CARRYB[9][4] ), .A(\ab[10][4] ) );
fadd S3_12_6 ( .S(\SUMB[12][6] ), .CO(\CARRYB[12][6] ), .CI(\ab[11][7] ),
```

```
   .B(\CARRYB[11][6] ), .A(\ab[12][6] ) );
fadd S2_11_4 ( .S(\SUMB[11][4] ), .CO(\CARRYB[11][4] ), .CI(\SUMB[10][5] ),
   .B(\CARRYB[10][4] ), .A(\ab[11][4] ) );
fadd S2_4_1 ( .S(\SUMB[4][1] ), .CO(\CARRYB[4][1] ), .CI(\SUMB[3][2] ),
   .B(\CARRYB[3][1] ), .A(\ab[4][1] ) );
fadd S1_6_0 ( .S(\CLA_SUM[6] ), .CO(\CARRYB[6][0] ), .CI(\SUMB[5][1] ),
   .B(\CARRYB[5][0] ), .A(\ab[6][0] ) );
fadd S1_10_0 ( .S(\CLA_SUM[10] ), .CO(\CARRYB[10][0] ), .CI(\SUMB[9][1] ),
   .B(\CARRYB[9][0] ), .A(\ab[10][0] ) );
fadd S2_11_3 ( .S(\SUMB[11][3] ), .CO(\CARRYB[11][3] ), .CI(\SUMB[10][4] ),
   .B(\CARRYB[10][3] ), .A(\ab[11][3] ) );
fadd S3_6_6 ( .S(\SUMB[6][6] ), .CO(\CARRYB[6][6] ), .CI(\ab[5][7] ), .B(
   \CARRYB[5][6] ), .A(\ab[6][6] ) );
fadd S2_9_1 ( .S(\SUMB[9][1] ), .CO(\CARRYB[9][1] ), .CI(\SUMB[8][2] ),
   .B(\CARRYB[8][1] ), .A(\ab[9][1] ) );
fadd S3_7_6 ( .S(\SUMB[7][6] ), .CO(\CARRYB[7][6] ), .CI(\ab[6][7] ), .B(
   \CARRYB[6][6] ), .A(\ab[7][6] ) );
fadd S2_8_1 ( .S(\SUMB[8][1] ), .CO(\CARRYB[8][1] ), .CI(\SUMB[7][2] ),
   .B(\CARRYB[7][1] ), .A(\ab[8][1] ) );
fadd S2_10_3 ( .S(\SUMB[10][3] ), .CO(\CARRYB[10][3] ), .CI(\SUMB[9][4] ),
   .B(\CARRYB[9][3] ), .A(\ab[10][3] ) );
fadd S1_11_0 ( .S(\CLA_SUM[11] ), .CO(\CARRYB[11][0] ), .CI(\SUMB[10][1] ),
   .B(\CARRYB[10][0] ), .A(\ab[11][0] ) );
fadd S2_3_5 ( .S(\SUMB[3][5] ), .CO(\CARRYB[3][5] ), .CI(\SUMB[2][6] ),
   .B(\CARRYB[2][5] ), .A(\ab[3][5] ) );
fadd S2_6_3 ( .S(\SUMB[6][3] ), .CO(\CARRYB[6][3] ), .CI(\SUMB[5][4] ),
   .B(\CARRYB[5][3] ), .A(\ab[6][3] ) );
fadd S1_7_0 ( .S(\CLA_SUM[7] ), .CO(\CARRYB[7][0] ), .CI(\SUMB[6][1] ),
   .B(\CARRYB[6][0] ), .A(\ab[7][0] ) );
fadd S4_5 ( .S(\SUMB[13][5] ), .CO(\CARRYB[13][5] ), .CI(\SUMB[12][6] ),
   .B(\CARRYB[12][5] ), .A(\ab[13][5] ) );
fadd S2_2_5 ( .S(\SUMB[2][5] ), .CO(\CARRYB[2][5] ), .CI(\SUMB[1][6] ),
   .B(\CARRYB[1][5] ), .A(\ab[2][5] ) );
fadd S2_7_3 ( .S(\SUMB[7][3] ), .CO(\CARRYB[7][3] ), .CI(\SUMB[6][4] ),
   .B(\CARRYB[6][3] ), .A(\ab[7][3] ) );
fadd S2_10_2 ( .S(\SUMB[10][2] ), .CO(\CARRYB[10][2] ), .CI(\SUMB[9][3] ),
   .B(\CARRYB[9][2] ), .A(\ab[10][2] ) );
fadd S2_11_2 ( .S(\SUMB[11][2] ), .CO(\CARRYB[11][2] ), .CI(\SUMB[10][3] ),
   .B(\CARRYB[10][2] ), .A(\ab[11][2] ) );
fadd S2_2_4 ( .S(\SUMB[2][4] ), .CO(\CARRYB[2][4] ), .CI(\SUMB[1][5] ),
   .B(\CARRYB[1][4] ), .A(\ab[2][4] ) );
fadd S2_7_2 ( .S(\SUMB[7][2] ), .CO(\CARRYB[7][2] ), .CI(\SUMB[6][3] ),
   .B(\CARRYB[6][2] ), .A(\ab[7][2] ) );
fadd S4_4 ( .S(\SUMB[13][4] ), .CO(\CARRYB[13][4] ), .CI(\SUMB[12][5] ),
   .B(\CARRYB[12][4] ), .A(\ab[13][4] ) );
fadd S2_6_2 ( .S(\SUMB[6][2] ), .CO(\CARRYB[6][2] ), .CI(\SUMB[5][3] ),
   .B(\CARRYB[5][2] ), .A(\ab[6][2] ) );
fadd S2_3_4 ( .S(\SUMB[3][4] ), .CO(\CARRYB[3][4] ), .CI(\SUMB[2][5] ),
   .B(\CARRYB[2][4] ), .A(\ab[3][4] ) );
fadd S2_3_3 ( .S(\SUMB[3][3] ), .CO(\CARRYB[3][3] ), .CI(\SUMB[2][4] ),
   .B(\CARRYB[2][3] ), .A(\ab[3][3] ) );
fadd S2_6_5 ( .S(\SUMB[6][5] ), .CO(\CARRYB[6][5] ), .CI(\SUMB[5][6] ),
   .B(\CARRYB[5][5] ), .A(\ab[6][5] ) );
fadd S4_3 ( .S(\SUMB[13][3] ), .CO(\CARRYB[13][3] ), .CI(\SUMB[12][4] ),
```

```
   .B(\CARRYB[12][3] ), .A(\ab[13][3] ) );
fadd S2_7_5 ( .S(\SUMB[7][5] ), .CO(\CARRYB[7][5] ), .CI(\SUMB[6][6] ),
   .B(\CARRYB[6][5] ), .A(\ab[7][5] ) );
fadd S2_2_3 ( .S(\SUMB[2][3] ), .CO(\CARRYB[2][3] ), .CI(\SUMB[1][4] ),
   .B(\CARRYB[1][3] ), .A(\ab[2][3] ) );
fadd S3_3_6 ( .S(\SUMB[3][6] ), .CO(\CARRYB[3][6] ), .CI(\ab[2][7] ), .B(
   \CARRYB[2][6] ), .A(\ab[3][6] ) );
fadd S1_3_0 ( .S(\CLA_SUM[3] ), .CO(\CARRYB[3][0] ), .CI(\SUMB[2][1] ),
   .B(\CARRYB[2][0] ), .A(\ab[3][0] ) );
fadd S2_11_5 ( .S(\SUMB[11][5] ), .CO(\CARRYB[11][5] ), .CI(\SUMB[10][6] ),
   .B(\CARRYB[10][5] ), .A(\ab[11][5] ) );
fadd S2_10_5 ( .S(\SUMB[10][5] ), .CO(\CARRYB[10][5] ), .CI(\SUMB[9][6] ),
   .B(\CARRYB[9][5] ), .A(\ab[10][5] ) );
fadd S3_2_6 ( .S(\SUMB[2][6] ), .CO(\CARRYB[2][6] ), .CI(\ab[1][7] ), .B(
   \CARRYB[1][6] ), .A(\ab[2][6] ) );
fadd S1_2_0 ( .S(\CLA_SUM[2] ), .CO(\CARRYB[2][0] ), .CI(\SUMB[1][1] ),
   .B(\CARRYB[1][0] ), .A(\ab[2][0] ) );
fadd S2_2_1 ( .S(\SUMB[2][1] ), .CO(\CARRYB[2][1] ), .CI(\SUMB[1][2] ),
   .B(\CARRYB[1][1] ), .A(\ab[2][1] ) );
fadd S2_12_2 ( .S(\SUMB[12][2] ), .CO(\CARRYB[12][2] ), .CI(\SUMB[11][3] ),
   .B(\CARRYB[11][2] ), .A(\ab[12][2] ) );
fadd S4_1 ( .S(\SUMB[13][1] ), .CO(\CARRYB[13][1] ), .CI(\SUMB[12][2] ),
   .B(\CARRYB[12][1] ), .A(\ab[13][1] ) );
fadd S2_3_1 ( .S(\SUMB[3][1] ), .CO(\CARRYB[3][1] ), .CI(\SUMB[2][2] ),
   .B(\CARRYB[2][1] ), .A(\ab[3][1] ) );
fadd S2_5_2 ( .S(\SUMB[5][2] ), .CO(\CARRYB[5][2] ), .CI(\SUMB[4][3] ),
   .B(\CARRYB[4][2] ), .A(\ab[5][2] ) );
fadd S2_8_5 ( .S(\SUMB[8][5] ), .CO(\CARRYB[8][5] ), .CI(\SUMB[7][6] ),
   .B(\CARRYB[7][5] ), .A(\ab[8][5] ) );
fadd S2_9_5 ( .S(\SUMB[9][5] ), .CO(\CARRYB[9][5] ), .CI(\SUMB[8][6] ),
   .B(\CARRYB[8][5] ), .A(\ab[9][5] ) );
fadd S2_4_2 ( .S(\SUMB[4][2] ), .CO(\CARRYB[4][2] ), .CI(\SUMB[3][3] ),
   .B(\CARRYB[3][2] ), .A(\ab[4][2] ) );
fadd S2_4_5 ( .S(\SUMB[4][5] ), .CO(\CARRYB[4][5] ), .CI(\SUMB[3][6] ),
   .B(\CARRYB[3][5] ), .A(\ab[4][5] ) );
fadd S2_9_2 ( .S(\SUMB[9][2] ), .CO(\CARRYB[9][2] ), .CI(\SUMB[8][3] ),
   .B(\CARRYB[8][2] ), .A(\ab[9][2] ) );
fadd S2_8_2 ( .S(\SUMB[8][2] ), .CO(\CARRYB[8][2] ), .CI(\SUMB[7][3] ),
   .B(\CARRYB[7][2] ), .A(\ab[8][2] ) );
fadd S2_5_5 ( .S(\SUMB[5][5] ), .CO(\CARRYB[5][5] ), .CI(\SUMB[4][6] ),
   .B(\CARRYB[4][5] ), .A(\ab[5][5] ) );
fadd S2_12_5 ( .S(\SUMB[12][5] ), .CO(\CARRYB[12][5] ), .CI(\SUMB[11][6] ),
   .B(\CARRYB[11][5] ), .A(\ab[12][5] ) );
fadd S5_6 ( .S(\SUMB[13][6] ), .CO(\CARRYB[13][6] ), .CI(\ab[12][7] ), .B(
   \CARRYB[12][6] ), .A(\ab[13][6] ) );
fadd S2_5_4 ( .S(\SUMB[5][4] ), .CO(\CARRYB[5][4] ), .CI(\SUMB[4][5] ),
   .B(\CARRYB[4][4] ), .A(\ab[5][4] ) );
fadd S2_8_3 ( .S(\SUMB[8][3] ), .CO(\CARRYB[8][3] ), .CI(\SUMB[7][4] ),
   .B(\CARRYB[7][3] ), .A(\ab[8][3] ) );
fadd S2_10_1 ( .S(\SUMB[10][1] ), .CO(\CARRYB[10][1] ), .CI(\SUMB[9][2] ),
   .B(\CARRYB[9][1] ), .A(\ab[10][1] ) );
fadd S2_9_3 ( .S(\SUMB[9][3] ), .CO(\CARRYB[9][3] ), .CI(\SUMB[8][4] ),
   .B(\CARRYB[8][3] ), .A(\ab[9][3] ) );
fadd S2_11_1 ( .S(\SUMB[11][1] ), .CO(\CARRYB[11][1] ), .CI(\SUMB[10][2] ),
```

```
    .B(\CARRYB[10][1] ), .A(\ab[11][1] ) );
  fadd S2_4_4 ( .S(\SUMB[4][4] ), .CO(\CARRYB[4][4] ), .CI(\SUMB[3][5] ),
    .B(\CARRYB[3][4] ), .A(\ab[4][4] ) );
  fadd S3_8_6 ( .S(\SUMB[8][6] ), .CO(\CARRYB[8][6] ), .CI(\ab[7][7] ), .B(
    \CARRYB[7][6] ), .A(\ab[8][6] ) );
  fadd S2_7_1 ( .S(\SUMB[7][1] ), .CO(\CARRYB[7][1] ), .CI(\SUMB[6][2] ),
    .B(\CARRYB[6][1] ), .A(\ab[7][1] ) );
  fadd S3_11_6 ( .S(\SUMB[11][6] ), .CO(\CARRYB[11][6] ), .CI(\ab[10][7] ),
    .B(\CARRYB[10][6] ), .A(\ab[11][6] ) );
  fadd S2_12_4 ( .S(\SUMB[12][4] ), .CO(\CARRYB[12][4] ), .CI(\SUMB[11][5] ),
    .B(\CARRYB[11][4] ), .A(\ab[12][4] ) );
  fadd S1_8_0 ( .S(\CLA_SUM[8] ), .CO(\CARRYB[8][0] ), .CI(\SUMB[7][1] ),
    .B(\CARRYB[7][0] ), .A(\ab[8][0] ) );
  fadd S1_9_0 ( .S(\CLA_SUM[9] ), .CO(\CARRYB[9][0] ), .CI(\SUMB[8][1] ),
    .B(\CARRYB[8][0] ), .A(\ab[9][0] ) );
  fadd S3_10_6 ( .S(\SUMB[10][6] ), .CO(\CARRYB[10][6] ), .CI(\ab[9][7] ),
    .B(\CARRYB[9][6] ), .A(\ab[10][6] ) );
  fadd S2_6_1 ( .S(\SUMB[6][1] ), .CO(\CARRYB[6][1] ), .CI(\SUMB[5][2] ),
    .B(\CARRYB[5][1] ), .A(\ab[6][1] ) );
  fadd S3_9_6 ( .S(\SUMB[9][6] ), .CO(\CARRYB[9][6] ), .CI(\ab[8][7] ), .B(
    \CARRYB[8][6] ), .A(\ab[9][6] ) );
  fadd S3_4_6 ( .S(\SUMB[4][6] ), .CO(\CARRYB[4][6] ), .CI(\ab[3][7] ), .B(
    \CARRYB[3][6] ), .A(\ab[4][6] ) );
  fadd S1_4_0 ( .S(\CLA_SUM[4] ), .CO(\CARRYB[4][0] ), .CI(\SUMB[3][1] ),
    .B(\CARRYB[3][0] ), .A(\ab[4][0] ) );
  fadd S1_12_0 ( .S(\CLA_SUM[12] ), .CO(\CARRYB[12][0] ), .CI(\SUMB[11][1] ),
    .B(\CARRYB[11][0] ), .A(\ab[12][0] ) );
  fadd S4_0 ( .S(\CLA_SUM[13] ), .CO(\CARRYB[13][0] ), .CI(\SUMB[12][1] ),
    .B(\CARRYB[12][0] ), .A(\ab[13][0] ) );
  fadd S3_5_6 ( .S(\SUMB[5][6] ), .CO(\CARRYB[5][6] ), .CI(\ab[4][7] ), .B(
    \CARRYB[4][6] ), .A(\ab[5][6] ) );
  fadd S2_12_3 ( .S(\SUMB[12][3] ), .CO(\CARRYB[12][3] ), .CI(\SUMB[11][4] ),
    .B(\CARRYB[11][3] ), .A(\ab[12][3] ) );
  fadd S1_5_0 ( .S(\CLA_SUM[5] ), .CO(\CARRYB[5][0] ), .CI(\SUMB[4][1] ),
    .B(\CARRYB[4][0] ), .A(\ab[5][0] ) );
  fadd S2_4_3 ( .S(\SUMB[4][3] ), .CO(\CARRYB[4][3] ), .CI(\SUMB[3][4] ),
    .B(\CARRYB[3][3] ), .A(\ab[4][3] ) );
  fadd S2_9_4 ( .S(\SUMB[9][4] ), .CO(\CARRYB[9][4] ), .CI(\SUMB[8][5] ),
    .B(\CARRYB[8][4] ), .A(\ab[9][4] ) );
  fadd S2_8_4 ( .S(\SUMB[8][4] ), .CO(\CARRYB[8][4] ), .CI(\SUMB[7][5] ),
    .B(\CARRYB[7][4] ), .A(\ab[8][4] ) );
  fadd S2_5_3 ( .S(\SUMB[5][3] ), .CO(\CARRYB[5][3] ), .CI(\SUMB[4][4] ),
    .B(\CARRYB[4][3] ), .A(\ab[5][3] ) );
endmodule


module MUL14x8_DW01_inc_22_0 ( A, SUM );
input  [21:0] A;
output [21:0] SUM;
  wire n1, n2, n3, n4, n5, n6, n7, n8, n9, n54, n55, n56, n57, n58, n78, n79,
    n84, n85, n86, n87, n88, n89, n90, n91, n92, n93, n94, n95, n96, n97;
  inv1 U5 ( .Y(SUM[0]), .A(A[0]) );
  nor21 U6 ( .Y(n1), .A(n2), .B(n3) );
  nor21 U7 ( .Y(n4), .A(n5), .B(n6) );
```

```
  nor21 U8 ( .Y(n7), .A(n8), .B(n9) );
  nand21 U9 ( .Y(n54), .A(A[1]), .B(A[0]) );
  inv1 U10 ( .Y(n55), .A(A[2]) );
  nor21 U11 ( .Y(n56), .A(n55), .B(n54) );
  inv1 U12 ( .Y(n57), .A(A[5]) );
  nand31 U13 ( .Y(n58), .A(A[3]), .B(n56), .C(A[4]) );
  nor21 U14 ( .Y(n78), .A(n58), .B(n57) );
  nand21 U15 ( .Y(n2), .A(A[6]), .B(n78) );
  inv1 U16 ( .Y(n3), .A(A[7]) );
  nor31 U17 ( .Y(n79), .A(n3), .B(n2), .C(n84) );
  nand21 U18 ( .Y(n85), .A(n79), .B(A[9]) );
  inv1 U19 ( .Y(n86), .A(A[10]) );
  nor21 U20 ( .Y(n87), .A(n86), .B(n85) );
  inv1 U21 ( .Y(n88), .A(A[13]) );
  nand31 U22 ( .Y(n89), .A(A[11]), .B(n87), .C(A[12]) );
  nor21 U23 ( .Y(n90), .A(n89), .B(n88) );
  nand21 U24 ( .Y(n8), .A(A[14]), .B(n90) );
  inv1 U25 ( .Y(n9), .A(A[15]) );
  nor31 U26 ( .Y(n91), .A(n9), .B(n8), .C(n92) );
  nand21 U27 ( .Y(n93), .A(n91), .B(A[17]) );
  inv1 U28 ( .Y(n94), .A(A[18]) );
  nor21 U29 ( .Y(n95), .A(n94), .B(n93) );
  nand21 U30 ( .Y(n5), .A(A[19]), .B(n95) );
  inv1 U31 ( .Y(n6), .A(A[20]) );
  xor21 U32 ( .Y(SUM[9]), .A(A[9]), .B(n79) );
  xnor21 U33 ( .Y(SUM[8]), .A(n84), .B(n1) );
  xnor21 U34 ( .Y(SUM[7]), .A(n2), .B(A[7]) );
  xor21 U35 ( .Y(SUM[6]), .A(A[6]), .B(n78) );
  xnor21 U36 ( .Y(SUM[5]), .A(A[5]), .B(n58) );
  xnor21 U37 ( .Y(SUM[4]), .A(n96), .B(A[4]) );
  xor21 U38 ( .Y(SUM[3]), .A(A[3]), .B(n56) );
  xnor21 U39 ( .Y(SUM[2]), .A(n54), .B(A[2]) );
  xor21 U40 ( .Y(SUM[21]), .A(A[21]), .B(n4) );
  xnor21 U41 ( .Y(SUM[20]), .A(n5), .B(A[20]) );
  xnor21 U42 ( .Y(SUM[1]), .A(A[1]), .B(SUM[0]) );
  xor21 U43 ( .Y(SUM[19]), .A(A[19]), .B(n95) );
  xnor21 U44 ( .Y(SUM[18]), .A(n93), .B(A[18]) );
  xor21 U45 ( .Y(SUM[17]), .A(n91), .B(A[17]) );
  xnor21 U46 ( .Y(SUM[16]), .A(n92), .B(n7) );
  xnor21 U47 ( .Y(SUM[15]), .A(n8), .B(A[15]) );
  xor21 U48 ( .Y(SUM[14]), .A(A[14]), .B(n90) );
  xnor21 U49 ( .Y(SUM[13]), .A(A[13]), .B(n89) );
  xnor21 U50 ( .Y(SUM[12]), .A(n97), .B(A[12]) );
  xor21 U51 ( .Y(SUM[11]), .A(A[11]), .B(n87) );
  xnor21 U52 ( .Y(SUM[10]), .A(n85), .B(A[10]) );
  nand21 U53 ( .Y(n96), .A(A[3]), .B(n56) );
  nand21 U54 ( .Y(n97), .A(A[11]), .B(n87) );
  inv1 U55 ( .Y(n84), .A(A[8]) );
  inv1 U56 ( .Y(n92), .A(A[16]) );
endmodule


module MUL14x8_DW01_inc_8_0 ( A, SUM );
input  [7:0] A;
```

```
output [7:0] SUM;
   wire n98, n99, n100, n101, n102, n103, n104, n105;
   inv1 U5 ( .Y(SUM[0]), .A(A[0]) );
   and21 U6 ( .Y(n98), .A(n99), .B(A[6]) );
   nand21 U7 ( .Y(n100), .A(A[1]), .B(A[0]) );
   inv1 U8 ( .Y(n101), .A(A[2]) );
   nor21 U9 ( .Y(n102), .A(n101), .B(n100) );
   nand31 U10 ( .Y(n103), .A(A[3]), .B(n102), .C(A[4]) );
   inv1 U11 ( .Y(n104), .A(A[5]) );
   nor21 U12 ( .Y(n99), .A(n104), .B(n103) );
   xor21 U13 ( .Y(SUM[7]), .A(A[7]), .B(n98) );
   xor21 U14 ( .Y(SUM[6]), .A(A[6]), .B(n99) );
   xnor21 U15 ( .Y(SUM[5]), .A(n103), .B(A[5]) );
   xnor21 U16 ( .Y(SUM[4]), .A(n105), .B(A[4]) );
   xor21 U17 ( .Y(SUM[3]), .A(A[3]), .B(n102) );
   xnor21 U18 ( .Y(SUM[2]), .A(n100), .B(A[2]) );
   xnor21 U19 ( .Y(SUM[1]), .A(A[1]), .B(SUM[0]) );
   nand21 U20 ( .Y(n105), .A(A[3]), .B(n102) );
endmodule


module MUL14x8 ( A, B, Prod );
input  [13:0] A;
input  [7:0] B;
output [21:0] Prod;
   wire \Invert[3] , \"+"-return153[18] , \L38[4] , \"+"-return153[11] ,
      \Invert[7] , \L38[0] , \"+"-return153[15] , \L38[2] ,
      \"+"-return153[17] , \Invert[5] , \L38[6] , \"+"-return153[20] ,
      \"+"-return153[13] , \Invert[1] , \"+"-return153[4] , \L144[3] ,
      \"+"-return47[1] , \FirstProd[17] , \FirstProd[6] , \L144[14] ,
      \FirstProd[2] , \L144[7] , \"+"-return153[0] , \L144[19] ,
      \FirstProd[20] , \L144[10] , \FirstProd[13] , \"+"-return153[9] ,
      \"+"-return47[5] , \FirstProd[0] , \FirstProd[18] , \FirstProd[11] ,
      \L144[21] , \L144[12] , \FirstProd[9] , \"+"-return153[2] , \L144[5] ,
      \FirstProd[15] , \FirstProd[4] , \L144[16] , \L144[8] ,
      \"+"-return47[3] , \L144[1] , \FirstProd[14] , \"+"-return153[6] ,
      \L144[17] , \FirstProd[5] , \L144[9] , \"+"-return47[2] , \L144[0] ,
      \"+"-return153[7] , \"+"-return47[6] , \FirstProd[1] , \FirstProd[10] ,
      \L144[20] , \L144[13] , \FirstProd[19] , \FirstProd[8] ,
      \"+"-return153[3] , \L144[4] , \L144[6] , \L144[18] ,
      \"+"-return153[1] , \FirstProd[3] , \"+"-return153[8] ,
      \FirstProd[21] , \FirstProd[12] , \L144[11] , \"+"-return47[4] ,
      \"+"-return153[5] , \L144[2] , \FirstProd[16] , \L144[15] ,
      \"+"-return47[0] , \FirstProd[7] , \L38[7] , \"+"-return153[12] ,
      \Invert[0] , \n49[7] , \L38[3] , \"+"-return153[16] , \Invert[4] ,
      \n155[21] , \Invert[6] , \L38[1] , \"+"-return153[14] , \Invert[2] ,
      \"+"-return153[19] , \L38[5] , \"+"-return153[10] , n179, n180, n181,
      n182, n183, n184, n185, n186, n187, n188, n189, n190, n191, n192, n193,
      n194, n195, n196, n197, n198, n199, n200, n201, n202, n203, n204, n205,
      n206, n207, n208, n209;
   inv1 U24 ( .Y(n180), .A(B[7]) );
   inv1 U25 ( .Y(n179), .A(B[7]) );
   inv1 U26 ( .Y(\L144[21] ), .A(\FirstProd[21] ) );
   inv1 U27 ( .Y(\L144[20] ), .A(\FirstProd[20] ) );
```

```
inv1 U28 ( .Y(\L144[19] ), .A(\FirstProd[19] ) );
inv1 U29 ( .Y(\L144[18] ), .A(\FirstProd[18] ) );
inv1 U30 ( .Y(\L144[17] ), .A(\FirstProd[17] ) );
inv1 U31 ( .Y(\L144[16] ), .A(\FirstProd[16] ) );
inv1 U32 ( .Y(\L144[15] ), .A(\FirstProd[15] ) );
inv1 U33 ( .Y(\L144[14] ), .A(\FirstProd[14] ) );
inv1 U34 ( .Y(\L144[13] ), .A(\FirstProd[13] ) );
inv1 U35 ( .Y(\L144[12] ), .A(\FirstProd[12] ) );
inv1 U36 ( .Y(\L144[11] ), .A(\FirstProd[11] ) );
inv1 U37 ( .Y(\L144[10] ), .A(\FirstProd[10] ) );
inv1 U38 ( .Y(\L144[9] ), .A(\FirstProd[9] ) );
inv1 U39 ( .Y(\L144[8] ), .A(\FirstProd[8] ) );
inv1 U40 ( .Y(\L144[7] ), .A(\FirstProd[7] ) );
inv1 U41 ( .Y(\L144[6] ), .A(\FirstProd[6] ) );
inv1 U42 ( .Y(\L144[5] ), .A(\FirstProd[5] ) );
inv1 U43 ( .Y(\L144[4] ), .A(\FirstProd[4] ) );
inv1 U44 ( .Y(\L144[3] ), .A(\FirstProd[3] ) );
inv1 U45 ( .Y(\L144[2] ), .A(\FirstProd[2] ) );
inv1 U46 ( .Y(\L144[1] ), .A(\FirstProd[1] ) );
inv1 U47 ( .Y(\L144[0] ), .A(\FirstProd[0] ) );
inv1 U48 ( .Y(\L38[7] ), .A(B[7]) );
inv1 U49 ( .Y(\L38[6] ), .A(B[6]) );
inv1 U50 ( .Y(\L38[5] ), .A(B[5]) );
inv1 U51 ( .Y(\L38[4] ), .A(B[4]) );
inv1 U52 ( .Y(\L38[3] ), .A(B[3]) );
inv1 U53 ( .Y(\L38[2] ), .A(B[2]) );
inv1 U54 ( .Y(\L38[1] ), .A(B[1]) );
inv1 U55 ( .Y(\L38[0] ), .A(B[0]) );
and21 U56 ( .Y(\Invert[7] ), .A(\n49[7] ), .B(B[7]) );
aoi22 U57 ( .Y(n181), .A(\FirstProd[9] ), .B(n179), .C(\"+"-return153[9] ),
   .D(B[7]) );
aoi22 U58 ( .Y(n182), .A(\FirstProd[8] ), .B(\L38[7] ), .C(
   \"+"-return153[8] ), .D(B[7]) );
aoi22 U59 ( .Y(n183), .A(\FirstProd[7] ), .B(n180), .C(\"+"-return153[7] ),
   .D(B[7]) );
aoi22 U60 ( .Y(n184), .A(\FirstProd[6] ), .B(n179), .C(\"+"-return153[6] ),
   .D(B[7]) );
aoi22 U61 ( .Y(n185), .A(\FirstProd[5] ), .B(\L38[7] ), .C(
   \"+"-return153[5] ), .D(B[7]) );
aoi22 U62 ( .Y(n186), .A(\FirstProd[4] ), .B(n179), .C(\"+"-return153[4] ),
   .D(B[7]) );
aoi22 U63 ( .Y(n187), .A(\FirstProd[3] ), .B(n179), .C(\"+"-return153[3] ),
   .D(B[7]) );
aoi22 U64 ( .Y(n188), .A(\FirstProd[2] ), .B(\L38[7] ), .C(
   \"+"-return153[2] ), .D(B[7]) );
aoi22 U65 ( .Y(n189), .A(\FirstProd[21] ), .B(n179), .C(\n155[21] ), .D(B
   [7]) );
aoi22 U66 ( .Y(n190), .A(\FirstProd[20] ), .B(n180), .C(
   \"+"-return153[20] ), .D(B[7]) );
aoi22 U67 ( .Y(n191), .A(\FirstProd[1] ), .B(n180), .C(\"+"-return153[1] ),
   .D(B[7]) );
aoi22 U68 ( .Y(n192), .A(\FirstProd[19] ), .B(n180), .C(
   \"+"-return153[19] ), .D(B[7]) );
aoi22 U69 ( .Y(n193), .A(\FirstProd[18] ), .B(\L38[7] ), .C(
```

```
   \"+"-return153[18] ), .D(B[7]) );
aoi22 U70 ( .Y(n194), .A(\FirstProd[17] ), .B(\L38[7] ), .C(
   \"+"-return153[17] ), .D(B[7]) );
aoi22 U71 ( .Y(n195), .A(\FirstProd[16] ), .B(n179), .C(
   \"+"-return153[16] ), .D(B[7]) );
aoi22 U72 ( .Y(n196), .A(\FirstProd[15] ), .B(n180), .C(
   \"+"-return153[15] ), .D(B[7]) );
aoi22 U73 ( .Y(n197), .A(\FirstProd[14] ), .B(n180), .C(
   \"+"-return153[14] ), .D(B[7]) );
aoi22 U74 ( .Y(n198), .A(\FirstProd[13] ), .B(n180), .C(
   \"+"-return153[13] ), .D(B[7]) );
aoi22 U75 ( .Y(n199), .A(\FirstProd[12] ), .B(n180), .C(
   \"+"-return153[12] ), .D(B[7]) );
aoi22 U76 ( .Y(n200), .A(\FirstProd[11] ), .B(\L38[7] ), .C(
   \"+"-return153[11] ), .D(B[7]) );
aoi22 U77 ( .Y(n201), .A(\FirstProd[10] ), .B(\L38[7] ), .C(
   \"+"-return153[10] ), .D(B[7]) );
aoi22 U78 ( .Y(n202), .A(\FirstProd[0] ), .B(n179), .C(\"+"-return153[0] ),
   .D(B[7]) );
aoi22 U79 ( .Y(n203), .A(B[6]), .B(n179), .C(\"+"-return47[6] ), .D(B[7])
    );
aoi22 U80 ( .Y(n204), .A(B[5]), .B(n180), .C(\"+"-return47[5] ), .D(B[7])
    );
aoi22 U81 ( .Y(n205), .A(B[4]), .B(\L38[7] ), .C(\"+"-return47[4] ), .D(B
   [7]) );
aoi22 U82 ( .Y(n206), .A(B[3]), .B(\L38[7] ), .C(\"+"-return47[3] ), .D(B
   [7]) );
aoi22 U83 ( .Y(n207), .A(B[2]), .B(n180), .C(\"+"-return47[2] ), .D(B[7])
    );
aoi22 U84 ( .Y(n208), .A(B[1]), .B(n179), .C(\"+"-return47[1] ), .D(B[7])
    );
aoi22 U85 ( .Y(n209), .A(B[0]), .B(n179), .C(\"+"-return47[0] ), .D(B[7])
    );
inv1 U86 ( .Y(Prod[9]), .A(n181) );
inv1 U87 ( .Y(Prod[8]), .A(n182) );
inv1 U88 ( .Y(Prod[7]), .A(n183) );
inv1 U89 ( .Y(Prod[6]), .A(n184) );
inv1 U90 ( .Y(Prod[5]), .A(n185) );
inv1 U91 ( .Y(Prod[4]), .A(n186) );
inv1 U92 ( .Y(Prod[3]), .A(n187) );
inv1 U93 ( .Y(Prod[2]), .A(n188) );
inv1 U94 ( .Y(Prod[21]), .A(n189) );
inv1 U95 ( .Y(Prod[20]), .A(n190) );
inv1 U96 ( .Y(Prod[1]), .A(n191) );
inv1 U97 ( .Y(Prod[19]), .A(n192) );
inv1 U98 ( .Y(Prod[18]), .A(n193) );
inv1 U99 ( .Y(Prod[17]), .A(n194) );
inv1 U100 ( .Y(Prod[16]), .A(n195) );
inv1 U101 ( .Y(Prod[15]), .A(n196) );
inv1 U102 ( .Y(Prod[14]), .A(n197) );
inv1 U103 ( .Y(Prod[13]), .A(n198) );
inv1 U104 ( .Y(Prod[12]), .A(n199) );
inv1 U105 ( .Y(Prod[11]), .A(n200) );
inv1 U106 ( .Y(Prod[10]), .A(n201) );
```

```
  inv1 U107 ( .Y(Prod[0]), .A(n202) );
  inv1 U108 ( .Y(\Invert[6] ), .A(n203) );
  inv1 U109 ( .Y(\Invert[5] ), .A(n204) );
  inv1 U110 ( .Y(\Invert[4] ), .A(n205) );
  inv1 U111 ( .Y(\Invert[3] ), .A(n206) );
  inv1 U112 ( .Y(\Invert[2] ), .A(n207) );
  inv1 U113 ( .Y(\Invert[1] ), .A(n208) );
  inv1 U114 ( .Y(\Invert[0] ), .A(n209) );
  MUL14x8_DW02_mult_14_8_0 \mul_31/mult/mult ( .A(A), .B({\Invert[7] ,
    \Invert[6] , \Invert[5] , \Invert[4] , \Invert[3] , \Invert[2] ,
    \Invert[1] , \Invert[0] }), .TC(1'b0), .PRODUCT({\FirstProd[21] ,
    \FirstProd[20] , \FirstProd[19] , \FirstProd[18] , \FirstProd[17] ,
    \FirstProd[16] , \FirstProd[15] , \FirstProd[14] , \FirstProd[13] ,
    \FirstProd[12] , \FirstProd[11] , \FirstProd[10] , \FirstProd[9] ,
    \FirstProd[8] , \FirstProd[7] , \FirstProd[6] , \FirstProd[5] ,
    \FirstProd[4] , \FirstProd[3] , \FirstProd[2] , \FirstProd[1] ,
    \FirstProd[0] }) );
  MUL14x8_DW01_inc_22_0 \add_35/plus/plus ( .A({\L144[21] , \L144[20] ,
    \L144[19] , \L144[18] , \L144[17] , \L144[16] , \L144[15] , \L144[14] ,
    \L144[13] , \L144[12] , \L144[11] , \L144[10] , \L144[9] , \L144[8] ,
    \L144[7] , \L144[6] , \L144[5] , \L144[4] , \L144[3] , \L144[2] ,
    \L144[1] , \L144[0] }), .SUM({\n155[21] , \"+"-return153[20] ,
    \"+"-return153[19] , \"+"-return153[18] , \"+"-return153[17] ,
    \"+"-return153[16] , \"+"-return153[15] , \"+"-return153[14] ,
    \"+"-return153[13] , \"+"-return153[12] , \"+"-return153[11] ,
    \"+"-return153[10] , \"+"-return153[9] , \"+"-return153[8] ,
    \"+"-return153[7] , \"+"-return153[6] , \"+"-return153[5] ,
    \"+"-return153[4] , \"+"-return153[3] , \"+"-return153[2] ,
    \"+"-return153[1] , \"+"-return153[0] }) );
  MUL14x8_DW01_inc_8_0 \add_25/plus/plus ( .A({\L38[7] , \L38[6] , \L38[5] ,
    \L38[4] , \L38[3] , \L38[2] , \L38[1] , \L38[0] }), .SUM({\n49[7] ,
    \"+"-return47[6] , \"+"-return47[5] , \"+"-return47[4] ,
    \"+"-return47[3] , \"+"-return47[2] , \"+"-return47[1] ,
    \"+"-return47[0] }) );
endmodule
```

Code 13

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;

entity ADD12 is
        port(   A,B     : in std_logic_vector(12 downto 0);
                Sum     : out std_logic_vector(12 downto 0));
end ADD12;

architecture RTL of ADD12 is

begin
Sum<=A+B;
end;
```

Code 14

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;

entity T_ADD12 is
end T_ADD12;

architecture  TEST of T_ADD12 is
  component ADD12
  port(   A,B: in std_logic_vector(12 downto 0);
          Sum : out std_logic_vector(12 downto 0));
  end component;

signal A,B,sum : std_logic_vector(12 downto 0);

begin
  ---------------------------------------------------
  -- Instantiate UUT
  ---------------------------------------------------
  UUT : ADD12 port map (A,B,Sum);
  ---------------------------------------------------
  -- Stimulus:
  ---------------------------------------------------
  STIMULUS: process
  begin
                    A<="0000000000000";B<="0000000000000";
                    wait for 10 ns;
                    A<="0000000000000";B<="0000001100100";
                    wait for 10 ns;
                    A<="0001001111000";B<="0000001100100";
                    wait for 10 ns;
                    A<="0011111111111";B<="0011111111111";
                    wait for 10 ns;
                    A<="1100000000000";B<="1100000000000";
                    wait for 10 ns;
                    A<="1100000000000";B<="0011111111111";
                    wait for 10 ns;
                    A<="0010101010101";B<="0001010101010";
                    wait;
  end process STIMULUS;
end TEST;

---------------------------------------------------
-- Configuration...
---------------------------------------------------
configuration CFG_T_ADD12 of T_ADD12 is
  for TEST
  end for;
end CFG_T_ADD12;
```

Code 15

```
module ADD12_DW01_add_13_0 ( A, B, CI, SUM, CO );
input  [12:0] A;
input  [12:0] B;
output [12:0] SUM;
input  CI;
output CO;
  wire \carry[9] , \carry[4] , \carry[2] , \carry[6] , \carry[12] ,
    \carry[11] , \carry[10] , \carry[8] , \carry[7] , \carry[3] ,
    \carry[5] , \carry[1] ;
  xor31 U1_12 ( .Y(SUM[12]), .A(A[12]), .B(B[12]), .C(\carry[12] ) );
  and21 U4 ( .Y(\carry[1] ), .A(A[0]), .B(B[0]) );
  xor21 U5 ( .Y(SUM[0]), .A(B[0]), .B(A[0]) );
  fadd U1_1 ( .S(SUM[1]), .CO(\carry[2] ), .CI(\carry[1] ), .B(B[1]), .A(A
    [1]) );
  fadd U1_6 ( .S(SUM[6]), .CO(\carry[7] ), .CI(\carry[6] ), .B(B[6]), .A(A
    [6]) );
  fadd U1_8 ( .S(SUM[8]), .CO(\carry[9] ), .CI(\carry[8] ), .B(B[8]), .A(A
    [8]) );
  fadd U1_11 ( .S(SUM[11]), .CO(\carry[12] ), .CI(\carry[11] ), .B(B[11]),
    .A(A[11]) );
  fadd U1_7 ( .S(SUM[7]), .CO(\carry[8] ), .CI(\carry[7] ), .B(B[7]), .A(A
    [7]) );
  fadd U1_2 ( .S(SUM[2]), .CO(\carry[3] ), .CI(\carry[2] ), .B(B[2]), .A(A
    [2]) );
  fadd U1_9 ( .S(SUM[9]), .CO(\carry[10] ), .CI(\carry[9] ), .B(B[9]), .A(A
    [9]) );
  fadd U1_10 ( .S(SUM[10]), .CO(\carry[11] ), .CI(\carry[10] ), .B(B[10]),
    .A(A[10]) );
  fadd U1_3 ( .S(SUM[3]), .CO(\carry[4] ), .CI(\carry[3] ), .B(B[3]), .A(A
    [3]) );
  fadd U1_4 ( .S(SUM[4]), .CO(\carry[5] ), .CI(\carry[4] ), .B(B[4]), .A(A
    [4]) );
  fadd U1_5 ( .S(SUM[5]), .CO(\carry[6] ), .CI(\carry[5] ), .B(B[5]), .A(A
    [5]) );
endmodule


module ADD12 ( A, B, Sum );
input  [12:0] A;
input  [12:0] B;
output [12:0] Sum;
  ADD12_DW01_add_13_0 \add_15/plus/plus  ( .A(A), .B(B), .CI(1'b0), .SUM(Sum
    ) );
endmodule
```

Code 16

```
--This contains the entire block

library IEEE;
use IEEE.Std_Logic_1164.all;
```

```
use IEEE.Std_Logic_Unsigned.all;


entity full is
        port(   clock,reset    : in std_logic;
                codein   : in std_logic_vector(15 downto 0);
                fuse1,fuse2,fuse3 : in std_logic_vector(6 downto 0);
                fuse4,fuse5,fuse6 : in std_logic_vector(6 downto 0);
                fuse7,fuse8,fuse9 : in std_logic_vector(6 downto 0);
                codeout  : out std_logic_vector(12 downto 0));
end full;


architecture RTL of full is

--------------------------------------------------------------------------
-------------------- COMPONENT DECLARATION ------------------------------
--------------------------------------------------------------------------

component control
        port(   clock,reset          : in std_logic;
                msb      : in std_logic_vector (2 downto 0);
                datain1,datain2,datain3,datain4,datain5        : in std_logic_vector (6 downto 0);
                datain6,datain7,datain8,datain9       : in std_logic_vector (6 downto 0);
                out1     : out std_logic_vector (6 downto 0);
                out2     : out std_logic_vector (6 downto 0));
end component;

component memory
        port(   clk,resetn : in std_logic;
                Data_in : in std_logic_vector(6 downto 0);
                Data_out : out std_logic_vector(6 downto 0));
end component;

component ADD8
        port(   A,B      : in std_logic_vector(7 downto 0);
                Sum      : out std_logic_vector(7 downto 0));
end component;

component ADD12
        port(   A,B      : in std_logic_vector (12 downto 0);
                Sum      : out      std_logic_vector(12 downto 0));
end component;

component MUL14x8
        port(   A        : in std_logic_vector(13 downto 0);
                B        : in std_logic_vector(7 downto 0);
                Prod     : out std_logic_vector(21 downto 0));
end component;

--------------------------------------------------------------------------
-------------------- SIGNAL DECLARATION ---------------------------------
--------------------------------------------------------------------------
signal y1 : std_logic_vector(6 downto 0);
```

```
signal y2 : std_logic_vector(6 downto 0);
signal y18:       std_logic_vector(7 downto 0);
signal y28: std_logic_vector(7 downto 0);
signal difference : std_logic_vector(7 downto 0);
signal product    : std_logic_vector(21 downto 0);
signal vectnew    : std_logic_vector(12 downto 0);
signal vectnewo   : std_logic_vector(13 downto 0);
signal total      : std_logic_vector(12 downto 0);
signal newproduct         : std_logic_vector(12 downto 0);

begin
        controlunit: control port map(clock,reset,codein(15 downto 13),fuse1,fuse2,fuse3,fuse4,fuse5,

        fuse6,fuse7,fuse8,fuse9,y1,y2);
        subtractor        : ADD8 port map(y28,y18,difference);
        multiplier        : MUL14x8 port map(vectnewo,difference,product);
        adder    : ADD12 port map(vectnew,newproduct,total);

        process(codein,y1,y2,total,product)
                        begin
                        vectnew<=y1(6)&y1(6)&y1(6)&y1&"000";
                        vectnewo<='0'&codein(12 downto 0);
                        newproduct<=product(21)&product(21 downto 10);
                        y18<=y1(6)&y1;
                        y28<=y2(6)&y2;
                        codeout<=total;
        end process;
end;
```

Code 17

```
--This contains the test bench for full.vhd

library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;

entity T_full is
end T_full;

architecture TEST of T_full is

        component full
                port(   clock,reset    : in std_logic;
                        codein   : in std_logic_vector(15 downto 0);
                        fuse1,fuse2,fuse3 : in std_logic_vector(6 downto 0);
                        fuse4,fuse5,fuse6 : in std_logic_vector(6 downto 0);
                        fuse7,fuse8,fuse9 : in std_logic_vector(6 downto 0);
                        codeout  : out std_logic_vector(12 downto 0));
        end component;
        component control
                port(   clock,reset         : in std_logic;
                        msb      : in std_logic_vector (2 downto 0);
```

```
                    datain1,datain2,datain3,datain4,datain5      : in std_logic_vector (6 downto 0);
                    datain6,datain7,datain8,datain9      : in std_logic_vector (6 downto 0);
                    out1     : out std_logic_vector (6 downto 0);
                    out2     : out std_logic_vector (6 downto 0));
    end component;

    component memory
        port(   clk,resetn : in std_logic;
                Data_in : in std_logic_vector(6 downto 0);
                Data_out : out std_logic_vector(6 downto 0));
    end component;
    component ADD8
        port(   A,B      : in std_logic_vector(7 downto 0);
                Sum      : out std_logic_vector(7 downto 0));
    end component;

    component ADD12
        port(   A,B      : in std_logic_vector (12 downto 0);
                Sum      : out    std_logic_vector(12 downto 0));
    end component;

    component MUL14x8
        port(   A        : in std_logic_vector(13 downto 0);
                B        : in std_logic_vector(7 downto 0);
                Prod     : out std_logic_vector(21 downto 0));
    end component;

--------------------------------------------------------------------------
------------------- SIGNAL DECLARATION ---------------------------------
--------------------------------------------------------------------------
signal clock,reset : std_logic;
signal codein      : std_logic_vector(15 downto 0);
signal fuse1,fuse2,fuse3,fuse4,fuse5          : std_logic_vector(6 downto 0);
signal fuse6,fuse7,fuse8,fuse9       : std_logic_vector(6 downto 0);
signal codeout     : std_logic_vector(12 downto 0);
--file results : text open write_mode is "outputvalues";

begin
UUT : full port map(clock,reset,codein,fuse1,fuse2,fuse3,fuse4,fuse5,fuse6,fuse7,fuse8,fuse9,codeout);

STIMULUS: process
    begin
        reset<='1';clock<='0';fuse1<="0000000";fuse2<="0111111";fuse3<="1000000";
        fuse4<="0000000";fuse5<="1110000";fuse6<="1001000";
        fuse7<="0111000";fuse8<="0010000";fuse9<="0000000";
        codein<="0000000000000000";
        wait for 1 ns;
        clock<='1';
        wait for 2 ns;
        for i in 0 to 1100 loop
            codein<= codein+128;
            wait for 1 ns;
        end loop;
        wait;
```

```
        end process STIMULUS;
end TEST;

configuration CFG_T_full of T_full is
        for TEST
        end for;
end CFG_T_full;
```

Code 18

```
----------This fuction does binary to thermometer conversion----------

library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Unsigned.all;

entity bin2thm is
  port(bin : in std_logic_vector(2 downto 0);
     therm: out std_logic_vector(6 downto 0));
end bin2thm;

architecture RTL of bin2thm is

begin
        therm<= "0000001" when (bin="001") else
                "0000011" when (bin="010") else
                "0000111" when (bin="011") else
                "0001111" when (bin="100") else
                "0011111" when (bin="101") else
                "0111111" when (bin="110") else
                "1111111" when (bin="111") else
                "0000000";
end;
```

Code 19

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity T_bin2thm is
end T_bin2thm;

architecture  TEST of T_bin2thm is

  component bin2thm
  port (bin : in std_logic_vector(2 downto 0);
      therm: out std_logic_vector(6 downto 0));
  end component;

 signal bin  : std_logic_vector(2 downto 0);
 signal therm  : std_logic_vector (6 downto 0);
```

```
begin
  ----------------------------------------------------
  -- Instantiate UUT
  ----------------------------------------------------

  UUT : bin2thm port map (bin,therm);

  ----------------------------------------------------
  -- Stimulus:
  ----------------------------------------------------
  STIMULUS: process
  begin
                    bin<="000";
                    wait for 10 ns;
                    bin<="001";
                    wait for 10 ns;
                    bin<="011";
                    wait for 10 ns;
                    bin<="100";
                    wait for 10 ns;
                    bin<="101";
                    wait for 10 ns;
                    bin<="110";
                    wait for 10 ns;
                    bin<="111";
                    wait for 10 ns;
                    bin<="010";
                    wait;
  end process STIMULUS;
end TEST;

-----------------------------------------------------
-- Configuration...
-----------------------------------------------------
configuration CFG_T_bin2thm of T_bin2thm is
  for TEST
  end for;
end CFG_T_bin2thm;
```

Code 20

```
module bin2thm ( bin, therm );
input  [2:0] bin;
output [6:0] therm;
   wire n377, n378, n379, n380, n381, n382, n383, n384, n385, n386;
   buf2 U60 ( .Y(therm[3]), .A(bin[2]) );
   aoi12 U61 ( .Y(n377), .A(therm[5]), .B1(bin[0]), .B2(bin[2]) );
   nor21 U62 ( .Y(therm[6]), .A(n378), .B(n379) );
   nor21 U63 ( .Y(n380), .A(n381), .B(bin[2]) );
   nand21 U64 ( .Y(therm[1]), .A(n382), .B(n380) );
   nand21 U65 ( .Y(therm[0]), .A(n378), .B(n379) );
   xnor21 U66 ( .Y(n379), .A(bin[2]), .B(n383) );
   inv1 U67 ( .Y(n384), .A(bin[0]) );
   inv1 U68 ( .Y(n385), .A(bin[2]) );
```

```
   aoi12 U69 ( .Y(n378), .A(n381), .B1(n383), .B2(bin[2]) );
   aoi12 U70 ( .Y(therm[5]), .A(n385), .B1(n382), .B2(n386) );
   nand21 U71 ( .Y(n382), .A(bin[1]), .B(n384) );
   oai12 U72 ( .Y(n383), .A(n382), .B1(bin[1]), .B2(n384) );
   nand21 U73 ( .Y(n386), .A(bin[1]), .B(bin[0]) );
   inv1 U74 ( .Y(n381), .A(n386) );
   inv1 U75 ( .Y(therm[2]), .A(n380) );
   inv1 U76 ( .Y(therm[4]), .A(n377) );
endmodule;
```

APPENDIX B – Matlab Code for Linearity Calculation

Code 1:

```
%This program is for calculating the INL and DNL of DACs
%This program was used for the linearity calculation of R2R DACs

format long;
clear

%Reading input from the file
fid1 = fopen('out','r');
[ql points] = fscanf(fid1,'%f');

%LSB Calculation based on Full Scale
steps = points
fullscale = 40e-3;
LSB = fullscale/points

%Calculation of Ideal Output Values
for i = 1:steps
  ideal(i) = (i-1)*LSB;
end

Ideal = ideal';

for i= 1:steps
  Actual(i) = ql(i);
end

%Offset Correction
for i= 1:steps
  Actual_off(i) = Actual(i) - Actual(1);
end

%Calculating Slope for Gain Error Correction
diff = Actual_off(steps) - Ideal(steps);
diffratio = diff/(steps -1);

%Gain Error Correction
for i= 1:steps
  actual(i) = Actual_off(i) - diffratio*(i-1);
end

%INL Calculation
for i= 1:steps
  inl(i) = actual(i) - Ideal(i);
end

%DNL Calculation
for i = 2:steps
  dnl(i) = actual(i) - actual(i-1) - LSB;
end

inl_lsb = inl/LSB;
```

```
dnl_lsb = dnl/LSB;

%Reporting the INL and DNL values and plotting the output
max(inl_lsb)
min(inl_lsb)
max(dnl_lsb)
min(dnl_lsb)

grid on
subplot(2,1,1),plot(inl_lsb);grid on
ylabel('INL Plot');
subplot(2,1,2),plot(dnl_lsb);grid on
ylabel('DNL Plot');
```

Code 2:

```
%This program was used for the linearity calculation of the opamp.

format long;
clear

%Reading Input from file
%Output obtained from simulation
fid1 = fopen('tempout','r');
[ql points] = fscanf(fid1,'%f');

%Input given to opamp
fid2 = fopen('tempin','r');
[q2 points2] = fscanf(fid2,'%f');

%LSB Calculation
steps = points
fullscale = 5;
LSB = fullscale/2^16

%Ideal Output Expected
for i=1:steps
   ideal(i)=2*q2(i);
end

Ideal = ideal';

%Actual Output obtained from simulation
for i= 1:steps
  Actual(i) = ql(i);
end

%Offset calculation
offset = Actual(1)-Ideal(1);

%Offset Correction
for i= 1:steps
  Actual_off(i) = Actual(i) - offset;
end
```

```
%Slope calculation for Gain Error Correction
diff = Actual_off(steps) - Ideal(steps);
diffratio = diff/(steps -1);

%Gain Error Correction
for i= 1:steps
  actual(i) = Actual_off(i) - diffratio*(i-1);
end

%INL Calculation
for i= 1:steps
  inl(i) = actual(i) - Ideal(i);
  inl_lsb(i)=inl(i)/LSB;
end

%DNL Calculation
for i=1:steps
   Exact_diff(i)=Actual(i)-Ideal(i);
   Exact_lsb(i)=Exact_diff(i)/LSB;
end

%Plotting the outputs
grid on
subplot(2,1,1),plot(Exact_lsb);grid on
ylabel('EXACT Plot');
subplot(2,1,2),plot(inl_lsb);grid on
ylabel('INL Plot');
```

Code 3:

```
%This program was used for the linearity calculation at the output
%of the opamps with the caldac included

format long;
clear

%Reading input from File
fid1 = fopen('out','r');
[ql points] = fscanf(fid1,'%f');

%LSB Calculation for MainDAC and CalDAC
steps = points
fullscale = 5;
LSB = fullscale/2^16

caldac_fullscale = 39.0625e-3;
caldac_LSB=caldac_fullscale/2^12

%Calculation of Ideal Output
for i=1:steps
   %ideal_output(i)=40e-3 - (i-1)*caldac_LSB;
   ideal_output(i)=(i-1)*caldac_LSB;
end
Ideal_output = ideal_output';
```

```
%Actual Output Obtained
for i= 1:steps
  Actual(i) = ql(i);
end

%Calculating Offset
offset = Actual(1)-Ideal_output(1);

%Offset Correction
for i= 1:steps
  Actual_off(i) = Actual(i) - offset;
end

%Gain Error Correction NOT PERFORMED
%diff = Actual_off(steps) - Ideal(steps);
%diffratio = diff/(steps -1);


actual =Actual_off;

%INL Calculation
for i= 1:steps
  inl(i) = actual(i) - Ideal_output(i);
  inl_lsb(i)=inl(i)/caldac_LSB;
end

%DNL Calculation
for i = 2:steps
  dnl(i) = actual(i) - actual(i-1) - caldac_LSB;
  dnl_lsb(i)=dnl(i)/caldac_LSB;
end

%Plotting the Output
%Note INL at last point is NOT zero, because Gain
%error Correction is NOT performed
max(inl_lsb)
min(inl_lsb)
max(dnl_lsb)
min(dnl_lsb)

grid on
subplot(2,1,1),plot(inl_lsb);grid on
ylabel('INL Plot');
subplot(2,1,2),plot(dnl_lsb);grid on
ylabel('DNL Plot');
```

APPENDIX C – Multiplier Schematic

In this section, the various sub-blocks of the multiplier block are given. The schematics are shown to give an idea of the complexity of the entire block:

Figure.C.1 – MUL14x8

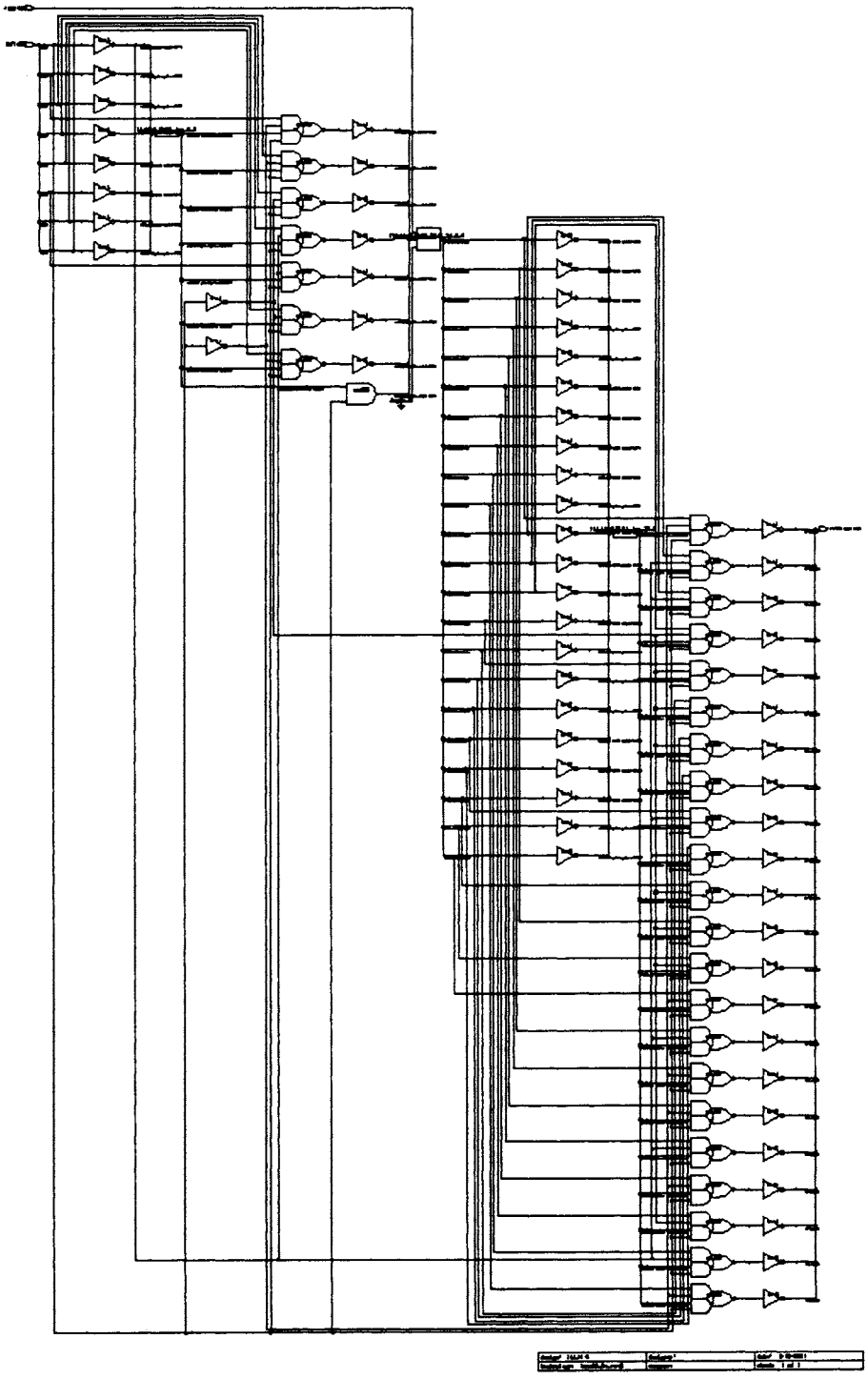


Figure.C.2 – MUL14x8_DW01_inc_8_0

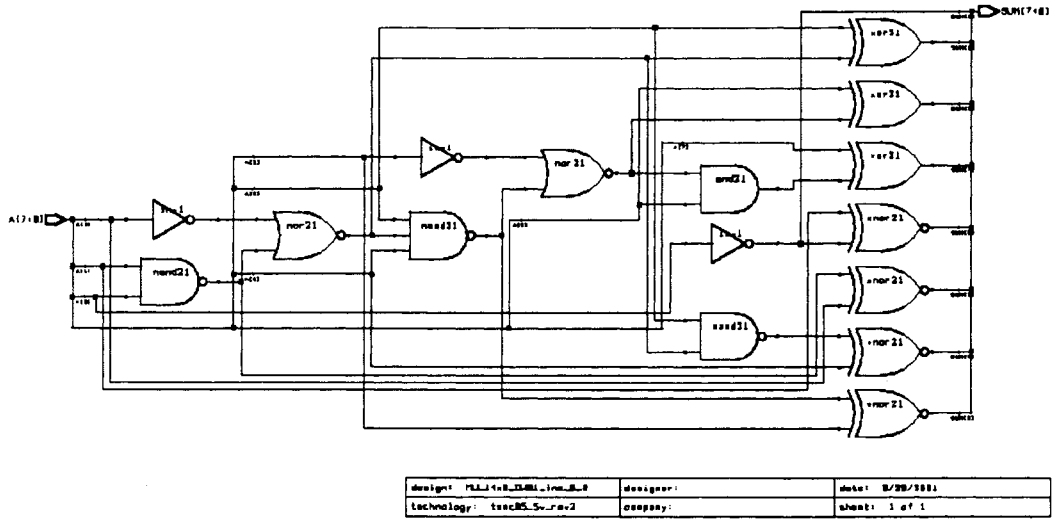


Figure.C.3 – MUL14x8_DW02_mult_14_8_0

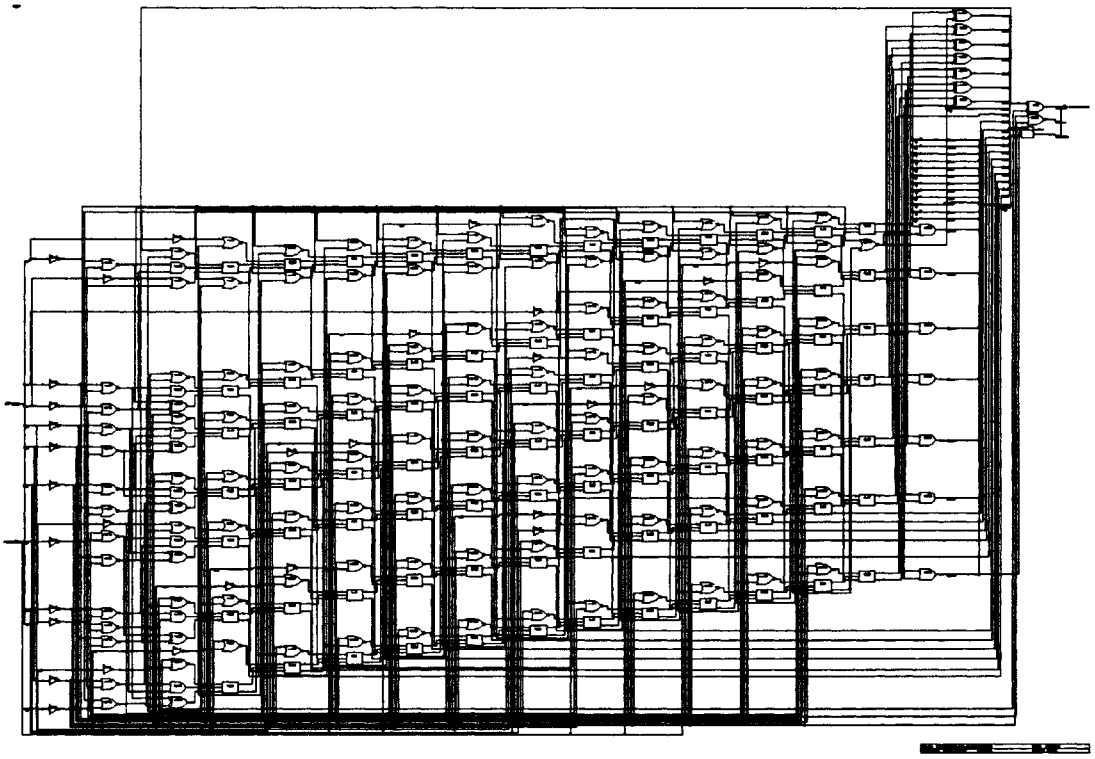


Figure.C.4 – MUL14x8_DW01_add_20_0

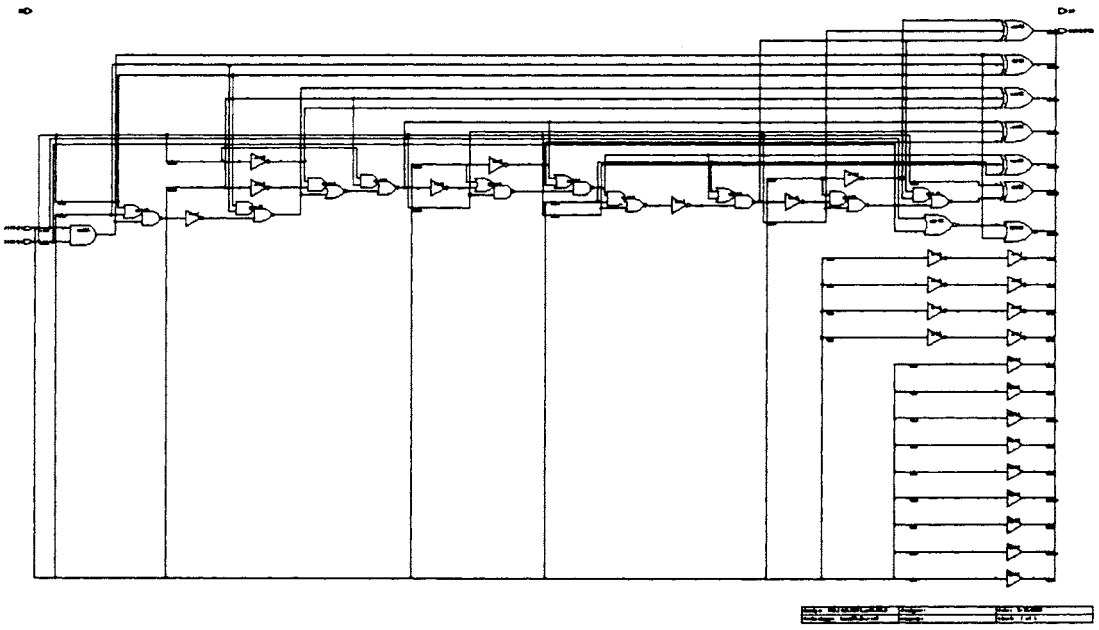
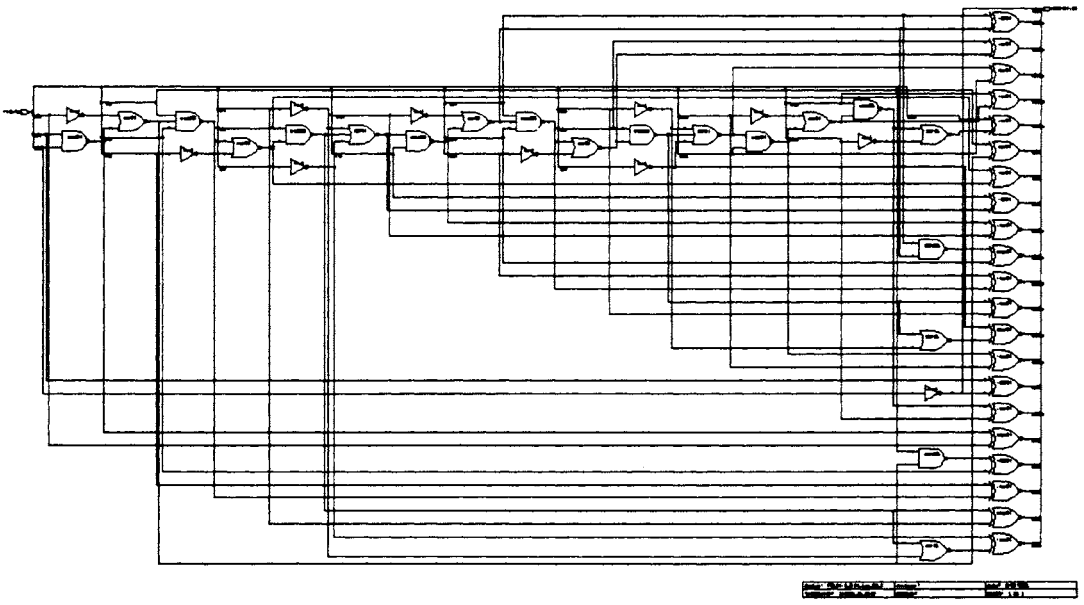


Figure.C.5 – MUL14x8_DW01_inc_22_0



APPENDIX D – Area and Power Estimation for DAC Calibration

This section deals with the area and power consumed by the entire calibration block. For the sake of comparison, first Scheme 1 with multiple amplifiers has been considered. This is followed by the area and power estimation of Scheme 2. Prior to that a simple cost analysis has been performed which is given below.

Cost Analysis:
A rough estimate of area of chip based on package size and profit was done:
At $650/ 6"wafer, if the cost of each die should be not more than 50¢ then we need 1300 dies per wafer.
Area of each die ➔ Area of wafer/Number of dies ➔ 14.03mm^2.
Assuming some loss, area of each die is 10 mm^2. (3.3mm * 3.3mm)

Original Design:
Area of original design: - 1750u * 750u ➔ 1/8(New Area)

Packaging : 3.22mm^2 for 8-pin package SOP-8
2.5mm^2 for TSSOP-8

Conclusion: The new die can be 2 times original size from packaging point of view
The new die can be 8 times original size from profit point of view.

Scheme 1

Area Estimation:
The total active area taken by each block is summarized below:
Interpolating Amplifier:

Bias Block Area	- 3682.45u^2 + 2.328pF
Amplifier	- 4782.45u^2 + 26kΩ + 20pF
Total (assuming only 1 input stage)	- 8464.9u^2 + 26KΩ + 22.328pF
Total (assuming 64 input stages)	**- 14764.9u^2 + 26KΩ + 22.328pF**

Calibration Blocks:
R2R Calibration DAC:

Resistor	- 43KΩ
Switch Area	- 2048u^2

Reference Generators : (Existing Design)

CalDAC Buffer :	- 1272u^2 + 3pF
Buffer B1 (used for reference generation):	- 5800u^2 + 1pF
Buffer B2 (used for reference generation):	- 208u^2
Buffer B3 (used for reference generation):	- 5800u^2 + 1pF
Buffer B4 (used for reference generation):	- 446u^2

Resistor String

Total Resistance	- 64KΩ

Total FeedBack Resistance- 8*44.2975KΩ = 354.38KΩ

2 Extra Interpolating amplifier used in Calibration Block: Each of them requiring
- 8464.9u^2 + 26KΩ + 22.328pF

Total area of Calibration block (analog) - **30455.8u^2 + 461.38KΩ + 49.656pF**
In terms of Interpolating amplifier (64 stages) - **2*Interpolating amplifier + 400KΩ (approx).**

Area Comparison with DAC8531:
Original Structure:
Interpolating Amplifier - ¼ DAC (approx.)
140KΩ resistance - 1/10 DAC

Calibration Block - 2*Interpolating amplifier + 400KΩ
- 2*(1/4 DAC) + (400/140)(1/10DAC)
- 0.78 DAC 8531

Note: The reference current generators block for all the amplifiers have not yet been included. This may add up to some more area. The digital portion also has to be included.

Power Consumption
Run 1: With the existing design the approximate current drawn by each block are given below:

Inverting Configuration Block	- 40u
Buffer B1	- 700u
Buffer B3	- 700u
CalDAC Buffer	- 100u
R-String	- 100u
Other Blocks	- 400u (includes output amplifier whose power dissipation depends on the input)

TOTAL (without output buffer) - 1.64mA
Power (without output buffer) - **5V*1.64mA = 8.2mW (very high)**
Area - **0.78 DAC**

Run 2: The Resistance in the CalDAC was increased from 1K to 10K.
Additional Resistance required - 387K = 0.27 DAC
The buffer (for Vref,High and Vref,Low) was modified to supply the required current for this CalDAC.
With this modification, the currents in various blocks are:

Inverting Configuration Block	- 40u
Buffer B1	- 90u
Buffer B3	- 90u
CalDAC Buffer	- 100u
R-String	- 100u
Other Blocks	- 400u

TOTAL (without output buffer) - 456uA
Power (without output buffer) - **5V*456uA = 2.28mW**
Area - **0.95 DAC**

Conclusion : Thus the scheme 1 suffers from both area and power requirement. This problem could be solved by using Scheme 2.

Scheme 2

R2R DAC	- 215K + 2048u^2
Vref,High Buffer	- 868 u^2 +3pF
Vref,Low Buffer	- 868 u^2 + 3pF
Voffset Buffer	- 868 u^2 + 3pF
Feedback Buffer	- 208 u^2
Current Mirror	- 2400 u^2 + 10K

Total Area (analog)	- 5K + 7260 u^2 + 10pF **- 0.275DAC**

Note : It is assumed that no reference string is required and offset voltages can be tapped from the main string.

Power Consumption:

Vref,High Buffer	- 75uA
Vref,Low Buffer	- 75uA
Voffset Buffer	- 75uA
Feedback Buffer	- 20uA
Current mirror offset current	- 20uA
Output buffer	- 220uA (Power dissipation in output buffer depends on input)
Power (without output buffer)	**- 5V*265uA = 1.325mW**

Area of Digital Circuitry: (obtained from area and time report from synopsys)

Multiplier	- 120643u^2
Control block	- 71322 u^2
8bit subtractor	- 8350 u^2
12 bit adder	- 8869 u^2

Total area	**- 209184 u^2**
Area of the original chip	**- 1750u x 750u = 1312500u^2**
Digital portion	**- 0.16DAC**

Total calibration Area	**- 0.275DAC for analog + 0.16DAC for digital** **- 0.435DAC**

Note: **The fuse blocks has not been included. That may contribute to additional 0.2DAC**

Conclusion: Thus as seen above, scheme 2 has much better area and power specifications than scheme 1.

Scope for improvement in present Design:

- The buffers for VrefHigh, VrefLow and Voffset can be modified to dissipate less power.
- The reference voltages can be tapped from existing string, there by saving area and power in the additional string.
- Interpolating opamp can be simplified (possibly) depending on the requirement for the design.
- The Inverting configuration can be combined with the Interpolating amplifier, thereby reducing area (already achieved in scheme 2)

APPENDIX E – Matlab Code for ADC Characterization

Code 1:

```
function input_generation();
global Bits start NOB signal1 signal2 Total_points LSB_shift;

Bits = input('\n\nEnter the Number of Bits:\n');
Weight1=input('\n\nEnter the value of Alpha1:\n');
Samples_per_LSB =input('\nEnter the number of samples per LSB\n');
LSB_shift = input('\nEnter the LSB shift:\n');

NOB=2^Bits;
disp(['Input Signal Generation!!'])
LSB=1/NOB;
Weight2=1-Weight1;
Number_of_samples = Samples_per_LSB*2^Bits;
additional=LSB_shift*2;
Total_points=Number_of_samples+Samples_per_LSB*additional;
extra = -LSB_shift*LSB;
signal1(1:Total_points,1)=0;
signal2(1:Total_points,1)=0;
t=(1:Total_points)'/Number_of_samples;
rand('state', sum(100*clock));

%%Input Signal Generation
sig_std=0;
signal1=t+(1-Weight1)*(t.^2-t);
signal2=signal1+extra;
```

Code 2:

```
function [hist1,hist2,inl_int,dnl_int]=Modeling_adc();
global Bits start NOB signal1 signal2 Total_points;
disp(['ADC Modeling!!'])

%%Variable Initialization
hist1=zeros(NOB,1);
hist2=zeros(NOB,1);
r=zeros(NOB,1);
trans=r;

%%Random Number Generation
rand('state',sum(100*clock));
for i=1:NOB
   r(i)=.5+rand;
end
rpart=cumsum(r);
rsigma=sum(r(1:NOB));

%%INL & DNL Introduced
trans=rpart/rsigma;
inl_int=(NOB-start)*cumsum(r(start:NOB-1))/sum(r(start:NOB-1))-(1:NOB-
start)';
dnl_int=inl_int;
dnl_int(2:NOB-start)=diff(inl_int);
```

```
%Calculating Histograms
bin=1;

for value=1:Total_points
   if(signal1(value)<trans(bin))
      hist1(bin)=hist1(bin)+1;
   else
       if bin<NOB
           bin=bin+1;
       end
       hist1(bin)=hist1(bin)+1;
   end
end
bin2=1;
for value=1:Total_points
   if(signal2(value)<trans(bin2))
      hist2(bin2)=hist2(bin2)+1;
   else
       if bin2<NOB
           bin2=bin2+1;
       end
       hist2(bin2)=hist2(bin2)+1;
   end
end
```

Code 3:

```
function [Estimated_Shift,inl_cal,dnl_cal]=Alpha_estimation(c1,c2);

NOB=length(c1);

s1=cumsum(c1);
s2=cumsum(c2);

cs1=c1(2:NOB);
cs2=c2(2:NOB);
N=length(cs1);

gamma=.5*(s2(1:NOB-1)-s1(2:NOB)).*(1./cs1+1./cs2);
alpha_est=N/sum(1./(1+gamma));
Estimated_Shift=alpha_est;
dnl_cal=alpha_est./(1+gamma)-1;
inl_cal=cumsum(dnl_cal);

disp(['Alpha Estimation Method!!'])
```

Code 4:

```
close all;
clear all;
clc;

global Bits start NOB signal1 signal2 LSB_shift;
input_generation;
```

```
start=5;

%%ADC Modeling
[hist1,hist2,inl_int,dnl_int]=Modeling_adc;
N=length(hist1);

a=hist1;
b=hist2;
if sum(hist1(1:start))>sum(hist2(1:start))
    hist1=b;
    hist2=a;
end

c1=hist1(start-1:N-1);
c2=hist2(start-1:N-1);
c1(1)=0;
c2(1)=sum(hist2(1:start-1))-sum(hist1(1:start-1));


[Estimated_Shift,inl_cal,dnl_cal]=Alpha_estimation(c1,c2);
disp(['End.'])
residual_inl = inl_cal - inl_int;

%%Figure 1 : Plot of INL introduced, INL estimated and Error in estimation
figure
subplot(2,1,1)
plot(inl_int);
hold on;
grid on;
plot(inl_cal,'r--');
title('Result using Alpha Estimation approach');
xlabel('Output Code');
ylabel('LSBs');

subplot(2,1,2)
plot(residual_inl);
grid on;
title('Residual Error in trip point estimation');
xlabel('Output Code');
ylabel('LSBs');

%%Figure 2 : Plot of DNL introduced, DNL estimated and Error in estimation
figure
subplot(2,1,1)
plot(dnl_int);
hold on;
grid on;
plot(dnl_cal,'r--');
title('DNL Introduced and DNL Estimated using Matrix Inversion Method');
xlabel('Output Code');
ylabel('LSBs');

subplot(2,1,2)
plot((dnl_int - dnl_cal),'r');
grid on;
```

```
title('Error in DNL Estimation');
xlabel('Output Code');
ylabel('LSBs');

%Summarising the results
Introduced_INL=max(abs(inl_int))
Residual_INL=max(abs(residual_inl))
Shift_Introduced=LSB_shift
Shift_Estimated=Estimated_Shift
```