IMPLEMENTATION OF LMS CALIBRATION ALGORITHM FOR 12-BIT PIPELINED ADC

A Dissertation Submitted In Partial Fulfilment of the Requirements for the award of the Degree of

MASTER OF TECHNOLOGY

In

VLSI Design

Submitted By

VISHALI

Roll no. 601361030

Under the guidance of

Dr. Alpana Agarwal

Associate Professor, ECED

Department of Electronics and Communication Engineering

THAPAR UNIVERSITY

(Established under the section 3 of UGC Act, 1956)

PATIALA 147004 (PUNJAB)

July 2015

DECLARATION

I hereby declare that the work which is being presented in dissertation titled **"IMPLEMENTATION OF LMS CALIBRATION ALGORITHM FOR 12-BIT PIPELINED ADC"** in partial fulfilment of the requirement for the award of degree of Master of Technology in VLSI Design submitted in Electronics and Communication Engineering Department of Thapar University, Patiala is an authentic record of my study carried out as under the guidance of **Dr. Alpana Agarwal,** Associate Professor, ECED during 2013-2015.

Date: 15 Jul 2015

Vishali

VISHALI

Roll no.601361030

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Alpana 15/7/15

(Dr. Alpana Agarwal)

Associate Professor, ECED,

Thapar University,

Patiala 147004, (Punjab)

Countersigned by:

(Dr. Sanjay Sharma)

Head

ECED, Thapar University,

Patiala, 147004

Dean of Academics Affairs

ECED, Thapar University,

Patiala, 147004

ACKNOWLEDGEMENT

To discover, analyze and to present something new is to venture on an untrodden path towards and unexplored destination is an arduous adventure unless one gets a true torch bearer to show the way. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various people. Words are often too less to reveals one's deep regards. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis. I acknowledge with gratitude and humility my indebtedness to **Dr. Alpana Agarwal, Associate Professor,** Electronics and Communication Engineering Department, Thapar University, Patiala, under whose guidance I had the privilege to complete this thesis. I wish to express my deep gratitude towards her for providing individual guidance and support throughout the thesis work.

I convey my sincere thanks to **Head of the Department, Dr. Sanjay Sharma** as well as **PG Coordinator, Dr. Amit Kohli, Associate Professor,** Electronics and Communication Engineering Department, entire faculty and staff of Electronics and Communication Engineering Department for their encouragement and cooperation.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the Almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down at the time of crisis and showing me the silver lining in the dark clouds. I do not find enough words with which I can express my feelings of thanks to my dear friends for their help, inspiration and moral support which went a long way in successful completion of the present study.

(VISHALI)

ABSTRACT

Pipelined ADCs are the most powerful, most efficient among all and widely used ADC because it offers an attractive combination of high speed, high accuracy and low power consumption. But there are some errors in the pipelined ADC which limits its accuracy and speed. These errors are finite gain error, capacitor mismatch, offset errors and amplifier non-linearity.

As the technology advances to deep sub-micron technology, high speed, lesser area and lower power consumption are the major concerns. This performance levels can easily be met in scaling dimensions of MOSFET for digital circuits. But, in analog circuits it is difficult to achieve all upto a remarkable level without degrading the performance. It happens due to certain errors in them as mentioned above. So, in order to meet the requirements of low power, low area and high speed in analog circuits as well, digital calibration algorithms are used based on digital signal processing techniques.

The present work is the implementation of one such calibration algorithm using LMS technique to calibrate 12-bit pipelined ADC with 1.5 bit/stage architecture. First of all the various errors are studied and an equation is derived considering the effect of all the errors. Then that equation is modelled in verilog in binary. This requires a use of floating point multipliers and adders which are implemented using booth's radix-8 algorithm in order to reduce area and delay. For reducing the delay in adding the partial products generated by Booth's algorithm, various compression techniques are introduced like the use of carry save adders. Then the design is synthesized and verified on FPGA in order to generate a calibration hardware.

After simulation, it is analysed that the calibration algorithm helped in reducing DNL and INL of the pipelined ADC. The analysis and simulation work is done in MATLAB and XILINX ISE 14.5 with ISIM simulator and the target FPGA device is Spartan 6.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

MATLAB	Matrix Laboratory
ADC	Analog to Digital Convertor
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
DAC	Digital to Analog Convertor
CMOS	Complementary Metal Oxide Semiconductor
LSB	Least Significant Bit
MSB	Most Significant Bit
VLSI	Very Large Scale Integration
SINAD	Signal to Noise and Distortion
SNR	Signal to Noise Ratio
SNDR	Signal to Noise and Distortion Ratio
INL	Integral Non-Linearity
DNL	Differential Non-Linearity
SOC	Systems on Chip
LMS	Least Mean Square
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
ISE	Integrated Software Environment
LUT	Look Up Table
RTL	Register Transfer Level
RAM	Random Access Memory
CPU	Central Processing Unit

XST	Xilinx Synthesis Technology
RDS	Redundant Signed Digit
OTA	Operational Transconductance Amplifier
CSA	Carry Save Adder
RCA	Ripple Carry Adder
RTN	Round Towards Negative
RTA	Round Ties To Away
RTE	Round Ties To Even
RTZ	Round Towards Zero
FSM	Finite State Machine

Chapter 1

INTRODUCTION

1.1 Background and Motivation

Today, modern SOCs demand integration of analog and digital systems together on a chip. The new technology trends targets decreasing the size of transistors to fabricate more and more of them on a single chip, reducing the requirement of area, power and increasing speed.

Practically, this technology scaling has both benefits and challenges. Advantages are higher speed and lower power due to lower capacitance loading and the lower supply voltage. On the other hand, the reduction in intrinsic device gain and available signal swing negatively impact the performance.

Thus, performance criteria can be easily met as desired in digital systems by scaling sizes of MOSFETs as compared to analog blocks. The predication made by Moore over 40 years ago called Moore's Law, concludes that every 18 to 24 months, the logic gates' density doubles. But practically, analog circuits like analog-to-digital converter only double their speed-resolution period every five years or more.

The graph given in Fig. 1.1 depicts the reason why analog circuits can't be scaled. Two transistors are used at 130 and 40-nm CMOS technologies. It can be seen that, there is a much larger change in current in the saturation region in the 40-nm process. This change depicts that the transistor has much less control over the current. Therefore, amplifier design has become difficult because the realizable gain per stage is extensively reduced. In order to compensate for that reduction in gain, more complicated circuits are required, which will lead to larger area and higher power consumption.

Another factor is the reduction in supply voltages due to which the available signal range gets shrinked. At 1V supply level, the signal range may be 0.7 V or less. So, in

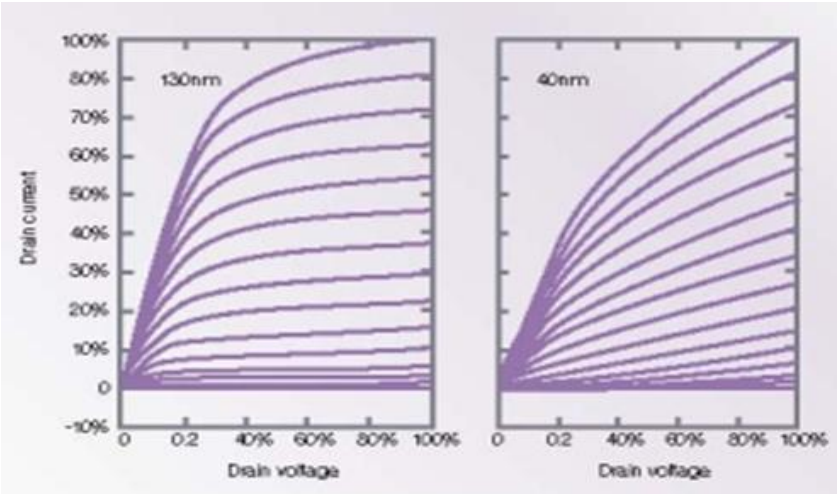

Fig. 1.1: The devices characteristics have degraded significantly from 130-nm process (LEFT) down to 40-nm process (RIGHT). [20]

order to maintain the same dynamic range, lower noise levels are required for this lower signal range. Using larger capacitors can help in reducing the noise level in mixed-signal circuits such as ADCs (using switched capacitors). If the signal range reduces by 2X, the capacitors must be increased 4X to compensate for it, making it quite difficult to scale down the area. Also, larger load capacitances require larger currents for charging and discharging, leading to higher power consumption.

Therefore, to achieve accurate performance at lower technology levels, analog architectures must be modified. A method called calibration provides different techniques which enable corrections for deviations in the analog front end by implementation of digital signal processing for digital correction of analog circuits, improving the efficiency of analog circuits at lower technology level. By this, it is possible to relax the performance of individual analog blocks making them simpler, smaller and lower in power consumption.

The challenge in these techniques is to identify algorithms that allow the calibration or correction to be self-directed, rather than at the fabrication stage. The circuit must be able to approximate its own errors and then apply an appropriate compensation without interrupting the normal operation.

1.2 ADC calibration

As the real world is essentially analog and hence in order to do real time processing, the digital devices have to be interfaced with the analog world. This interfacing is done with the help of A/D converters.

1.2.1 Choosing the ADC architecture:

There are many A/D converters architectures available each having their own advantages and disadvantages, like Flash ADC has high speed which provides high conversion rate, Oversampling A/D converters offers very good resolution, Interpolating and Folding ADCs have good resolution at fairly good speed *etc*. But all of them are coupled with one or other disadvantage like high power dissipation, high settling time, parasitic capacitance, low speed *etc*. Along with that, the errors introduced by capacitor mismatch, finite op-amp-gain, comparator offset voltage, *etc*. influence the conversion rate and resolution of A/D converter, which are amongst few most important factors in selecting an A/D converter. The graph shown in Fig. 1.2 can help in choosing the ADC architecture.

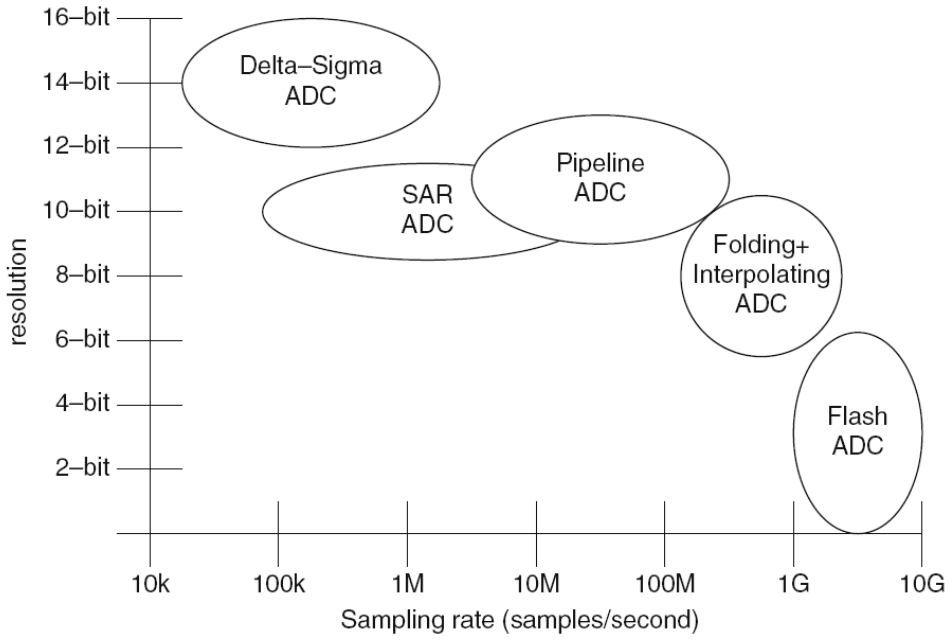

Fig. 1.2: comparison of different ADC architectures.

The graph depicts that pipeline ADC has a good resolution along with quite good sample rate (speed), thus much calibration work could be done using this architecture.

1.3 Pipelined ADC architecture

A pipelined ADC is actually a cascade of a number of stages *i.e.* it divides the conversion task into multiple stages hence overcomes the limitation of flash ADC as it provides higher resolution compared to flash ADC using equal number of comparators, but at the cost of increase in conversion time. Pipelined ADC with *p* stages requires *p* cycles for its operation but a flash ADC requires only one cycle. But since each stage has its own sample and hold, so, these *p* conversions are in progress simultaneously resulting in same throughput as that of flash ADC *i.e.* one conversion per cycle. One limitation is the latency (of *p* cycles) introduced in pipelined architecture. Also, a fixed period clock is required for the conversion process.

Each stage consists of a sub-ADC and DAC, both of 2-bits. Firstly, the sample and hold circuit samples the input and passes the samples to the sub-ADC which is a flash convertor and gives a 2-bit digital output which is further passed as input to the DAC , converting it back to analog. This converted DAC result is subtracted from the original sample and the difference is called residue which is amplified by the factor of 2 and passed on to the next stage. This happens at every stage and the number of stages required is determined by the resolution desired. More precisely, a pipelined ADC with *p* number of stages with each stage having a flash sub-ADC of *m*-bits, the resolution obtained in bits is $n = p \times m$ using $p \times (2m-1)$ comparators.

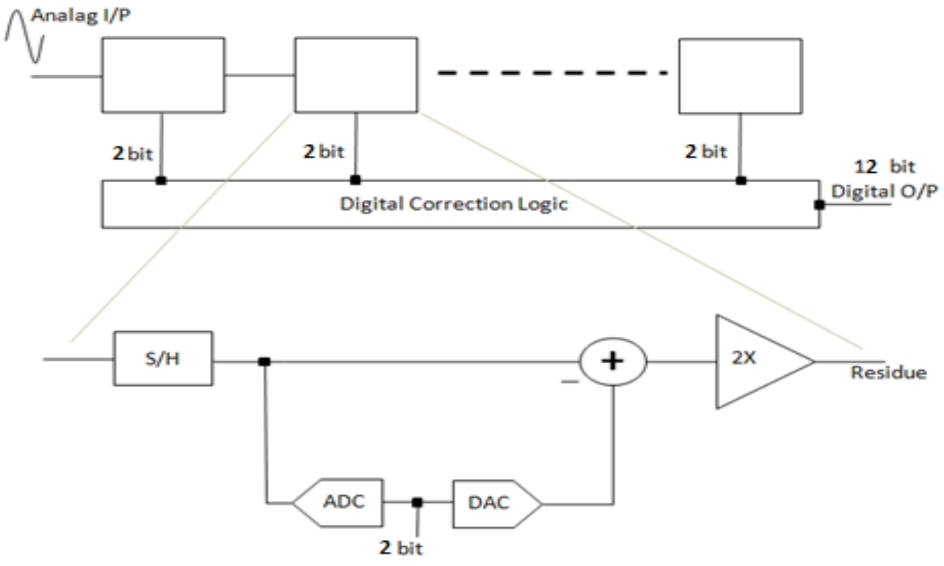

Fig. 1.3: Pipelined ADC architecture.

A 12-bit pipelined ADC using 1.5 bit per stage resolution is shown in Fig. 1.3, where each stage is approximating the input into 1.5 bits *i.e.* one bit with 0.5 bit overlap. This overlap is the redundancy to provide a large tolerance for component tolerances and imperfections. A digital correction algorithm eliminates this redundancy.

The sub-ADC in 1.5-bit stage divides the operating range in three parts *viz.* High (H), Medium (M) and Low (L). The dividing voltages are two levels, V_H and V_L. The gain of the amplifier is 2. Therefore, these voltage levels must lie within the range of $-V_{REF}/2$ and $+V_{REF}/2$. Commonly used values are $V_H = 0.25\ V_{REF}$ and $V_L = -0.25V_{REF}$ allowing the circuit imperfections.

Therefore the values above V_H are considered in High range, those between V_H and V_L in Medium range and those lesser than V_L are considered in Low range. This system is called Redundant Signed Digit (RDS) because the High range is labelled as +1, Mid range as 0, and Low range as –1. The Table 1.1 depicts the two bit sub-ADC output (B1 and B0) as 00, 01, and 10 for V_{IN} in the L, M, and H input ranges, respectively. The DAC outputs are $-V_{REF}$, 0, and $+V_{REF}$ for V_{IN} in the L, M, and H input ranges and the corresponding residue voltages are shown for all the three ranges.

Table 1.1: Sub-ADC, DAC and residue output in various ranges.

Vin	**Range**	**B1**	**B0**	**DAC O/P**	**Residue**
$V_{IN} > V_H$	H	1	0	$+V_{REF}$	$2V_{IN} - V_{REF}$
$V_L < V_{IN} < V_H$	M	0	1	0	$2V_{IN}$
$V_{IN} < V_L$	L	0	0	$-V_{REF}$	$2V_{IN} + V_{REF}$

Fig. 1.4 shows the ideal and actual V_{OUT} (residue voltage)/V_{IN} transfer function for one stage of pipelined ADC.

The change in slope represents gain error and overshooting or undershooting the peak represents offset error. These errors are fully passed on to the subsequent stage, so

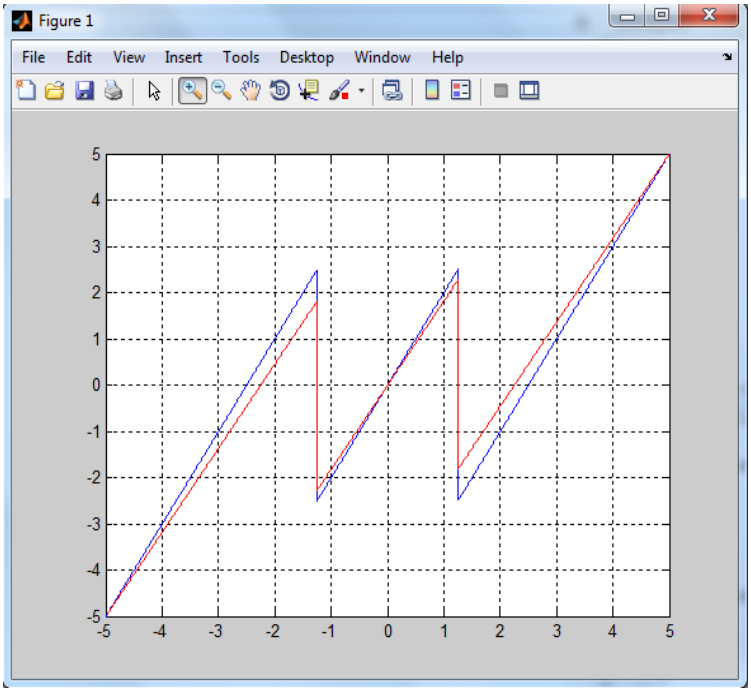

Fig. 1.4: Ideal (blue) and actual (red) $V_{RESIDUE}$ (y-axis) vs. V_{IN} (x axis) transfer characteristics.

long as the offset error does not exceed 0.25 V_{REF}, as shown in Fig. 1.5. The subsequent stage detects an over-range error, which may be digitally eliminated by adding or subtracting a bit from the digital output (depending on whether the error was an over or under range error).

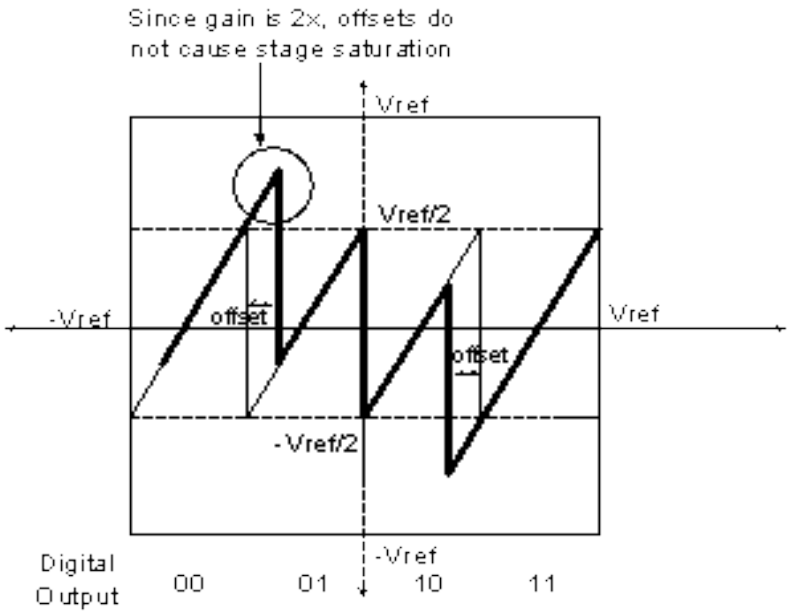

Fig. 1.5: Impact of errors on stage transfer function.

Now, after getting 2-bits as outputs from each stage, the final one bit per stage digital output is obtained through digital correction algorithm. It consists of a series of adders used to eliminate the redundancy. For example, consider a three stage pipelined ADC as shown in Fig. 1.7, it gives three outputs of 2-bits each. The final 3 bit output is obtained by adding all the 2 -bit outputs altogether with one bit overlap between adjacent stages as shown in Fig. 1.6.

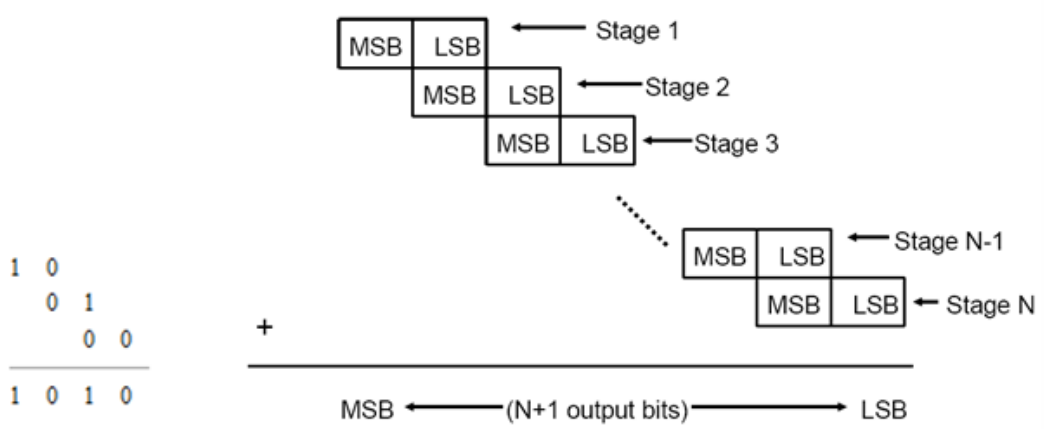

Fig. 1.6: Concept of digital correction.

The final 3-bit output is "101" by ignoring the rightmost bit. This process could be implemented by series of adders.

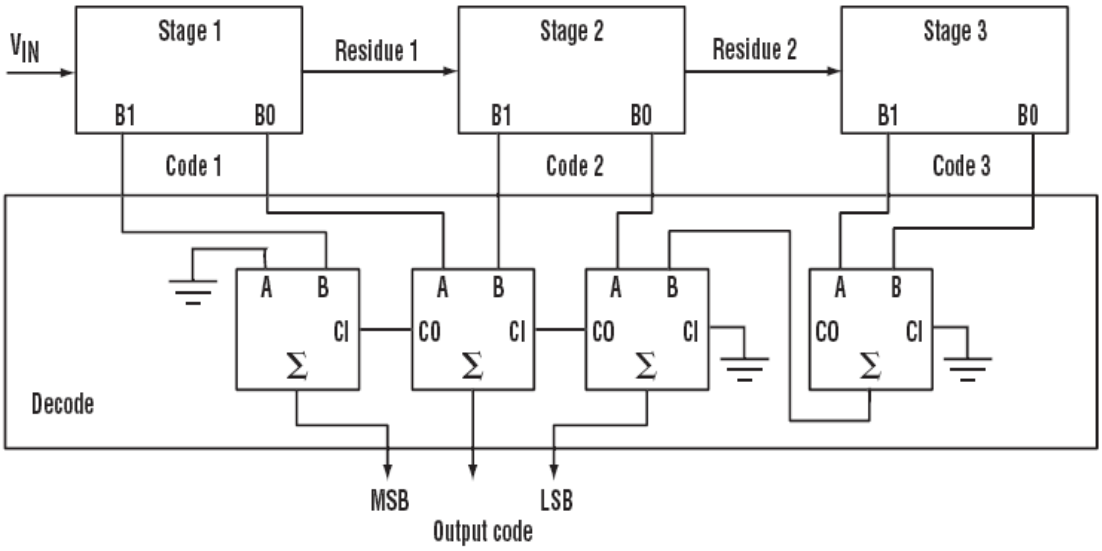

Fig. 1.7: Implementation of redundancy removal algo for 3 stages of 1.5 bit each.

The pipelined stage behaviour explained above is ideal without any component errors. Practically, there are lot of parameters that vary and cause imperfections in the output. Some of these are offset voltages in the comparators and amplifiers, gain error of the amplifier, amplifier settling time, converter nonlinearities, capacitor voltage dependency, *etc*. The front stages require more accuracy as compared to the later ones because the error terms from later stages gets divided by the preceding stage amplification factor.

1.4 ADC specifications.

The errors shift the ideal transfer function in some amount. The impact of these errors could be seen in the following ADC specifications.

1.4.1 Static specifications

1) DNL (Differential nonlinearity)

DNL tells how far a code is from an adjacent code. It is the difference between an actual step width and the ideal value of 1 LSB. If this difference is zero, the DNL is zero, making the ADC ideal. For good performance of an ADC, DNL should be <±1LSB. It means, as the input voltage is swept over its range, output of the ADC contains all the possible code combinations for a given resolution. But if the DNL becomes ±1 LSB or greater, there will be some missing codes in the output. DNL is evaluated as:

$$DNL = \max[\frac{V(i+1)-V(i)}{V_{LSB}} - 1] \quad (1)$$

So, due to component errors, non-linearity and other imperfections at lower technology, the ideal transfer curve for 4-bit pipelined ADC got shifted as shown in Fig. 1.8 introducing the DNL error.

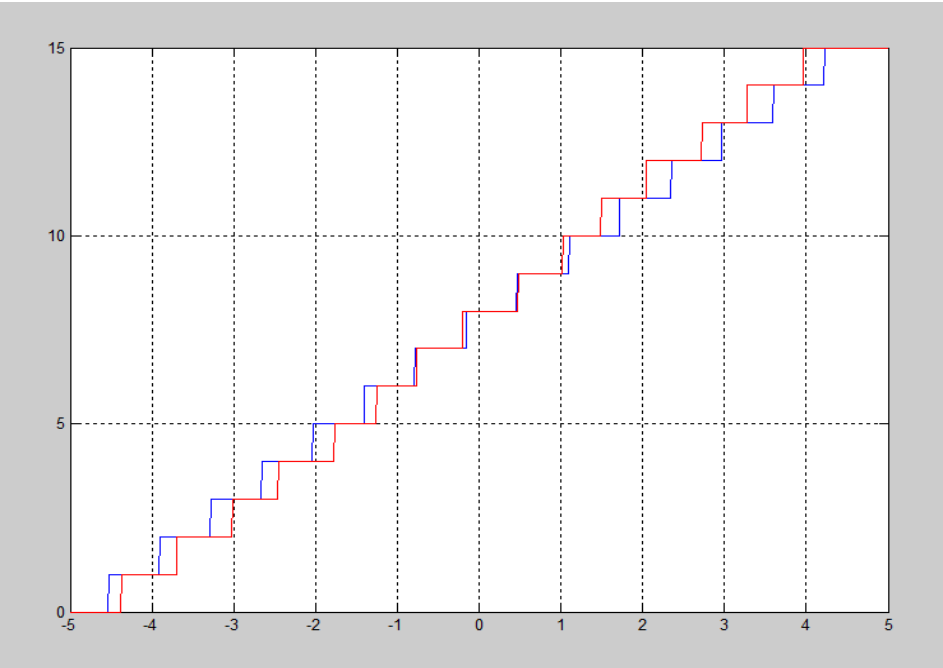

Fig. 1.8: Ideal transfer curve (blue) and actual transfer curve due to errors (red) of 4-bit pipelined ADC.

2) INL (Integral nonlinearity)

INL represents the integral of the DNL error. More precisely, it is the deviation of the actual transfer function from the ideal transfer function which is taken as a straight line joining the end points of the ideal transfer curve after the nullification of gain and offset errors as shown on Fig. 1.9.

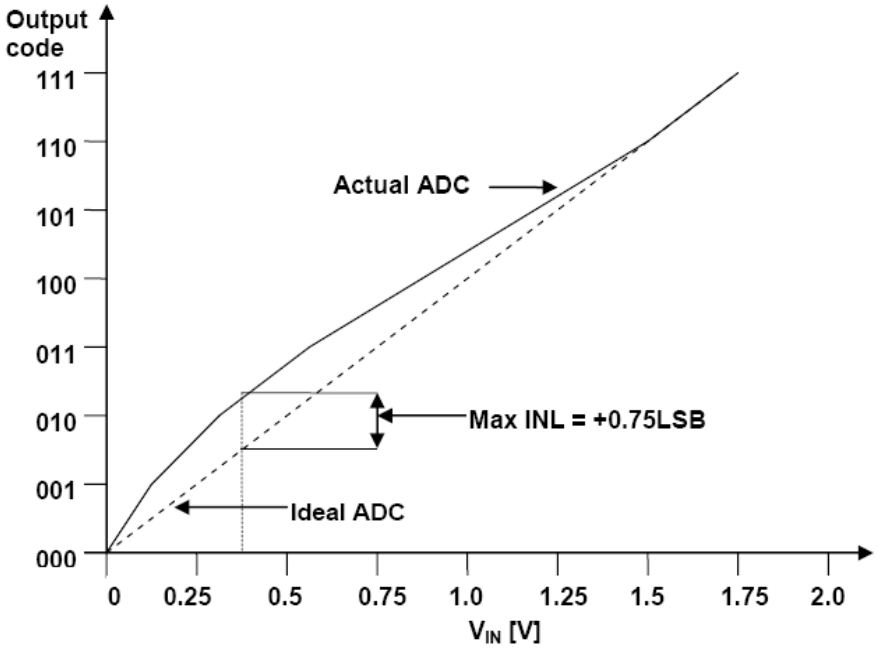

Fig. 1.9: Ideal and actual ADC transfer curve representing INL.

3) Gain and offset error

Offset error shifts all the quantization levels in the ideal transfer curve by a same amount called offset. It is measured graphically in LSBs. Basically, it is the difference between ideal and real input values so as to make the output zero.

Gain error is the change of slope of actual transfer curve as compared to the ideal one. A positive gain error leads to the reduction of input signal dynamic range and a negative gain error leads to the reduction the output code range. The gain error in 12-bit pipelined ADC due to imperfections at lower technology is shown in Fig. 1.10.

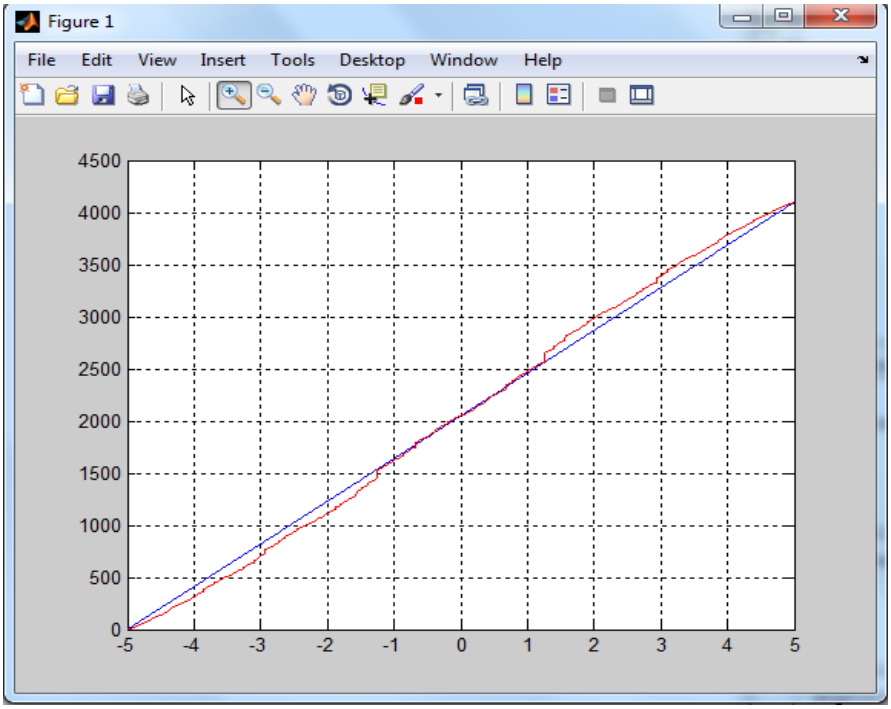

Fig. 1.10: Actual (red) and ideal(blue) transfer function of a 12-bit pipelined ADC.

4) Temperature drift

Fig. 1.11 shows how reference voltage of ADC is affected by temperature drift depending upon the resolution. For temperature range (-40°C to +85°C), a 12-bit ADC can withstand a temperature drift of about 4ppm/°C maintaining its accuracy. As no such ADC is available as of now, so relaxing the requirement, it can withstand maximum 25ppm/°C.

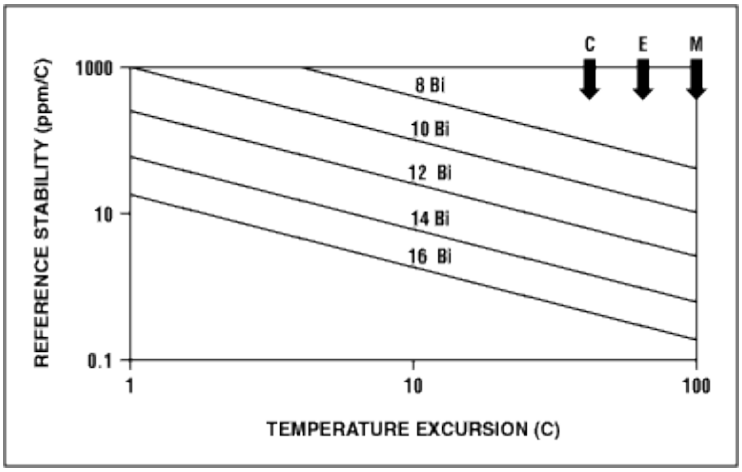

Fig. 1.11: Voltage-reference-drift requirements relate to ADC resolution [22].

1.4.2 Dynamic specifications

1) SINAD

The Signal-to-Noise and Distortion (SINAD) presents information regarding the noise and harmonic energy present in the frequency spectrum. It is specified in terms of dB as,

$$SINAD = \frac{P_{signal} + P_{noise} + P_{distortion}}{P_{noise} + P_{distortion}} \quad (2)$$

where, P is the average power of the signal, noise and distortion components. A lower SINAD value means worse performance of the system.

2) SNR

SNR stands for Signal to Noise Ratio. It is described as the ratio of signal power to the noise power, and is usually represented in dBs. It is expressed as:

$$SNR = 10log_{10}\frac{P_s}{P_n} \quad (3)$$

where P_s denotes signal power and P_n denotes noise power.

SNR usually degrades with increase in input frequency because the accuracy of the comparator within the ADC degrades. SNR for an ideal ADC is given by:

$$SNR\ (dB) = 6.02\ N + 1.76(dB) \quad (4)$$

where N is the resolution of the ADC.

3) SNDR

SNDR stands for signal to noise and distortion ratio. It is almost equal to SNR, but in the term noise power, harmonic distortion is also included. Mathematically, it can be expressed in dBs as

$$SNDR(dB) = 20log_{10}\frac{A_{signal}}{A_{noise}} \quad (5)$$

Where A_{signal} depicts the output signal level and A_{noise} represents the noise (in volts or ampere).

1.5 Thesis organisation

The main aim of this thesis is to implement a calibration algorithm using verilog to compensate for the errors in pipelined ADC at lower technology, without degrading the performance. This thesis is organised into eight chapters. The remainder of the thesis is organised as follows:

- Chapter 2- Literature Review.
- Chapter 3- Study of working of pipelined ADC and its errors, classification of error sources and their effects, deriving the equations for the ADC considering those errors.
- Chapter 4- Calibration algorithm used – LMS (Least Mean Square) and modelling the LMS using verilog, representation of floating point numbers in binary half precision IEEE-754 format, modelling of floating point multiplier using booth-8 algorithm, adding the partial products generated using carry save adders to reduce the delay, modelling of signed floating point adder.
- Chapter 5- FPGA Implementation of LMS algorithm & Results.
- Chapter 6- Conclusion and Future scope.

Chapter 2

LITERATURE REVIEW

Firstly, the pipelined ADC structure was studied and then a literature survey was done for various algorithms for calibration and their implementation. Broadly, it can be classified into following sections:

a) Calibration algorithm for the calibration of pipelined ADC
b) Fast multiplication techniques using Booth's algorithm.
c) Partial product reduction techniques.
d) Floating point multiplier using IEEE-754 formats.

2.1 Literature review on Calibration algorithm for pipelined ADC

[1] John P. Keane, Paul J. Hurst, and Stephen H. Lewis, "Background Interstage Gain Calibration Technique for Pipelined ADCs", *IEEE Transactions on Circuits and Systems-I,* Vol.52, No.1, Jan. 2005. [1]

It proposed a background self-calibration technique called LMS that can correct both linear and nonlinear errors in the interstage amplifiers of pipeline and algorithmic analog-to-digital converters (ADCs). To measure the errors being used as inputs to the calibration algo, stage redundancy is used in a pipeline architecture that is corrected using digital post-processing. It has been analysed that for highly non-ideal residue amplifiers, significant improvement in ADC linearity is possible by using this technique. The technique has less dependence on input signal statistics and permits a fast convergence. For non-ideal interstage residue amplifiers in 12-bit pipelined ADC, after calibration, the simulations show a signal-to-noise-and-distortion-ratio performance of 72 dB and a spurious-free dynamic range performance of 112 dB, with calibration tracking time constants of approximately 8* 105 sample periods, which is more than ten times faster. With interstage gain calibration only, the peak

DNL is comparatively unchanged at 0.7 least significant bit (LSB), while the peak INL is reduced from over 20 to 5 LSB. With nonlinearity calibration added, the peak DNL is less than 0.25 LSB, and the peak INL is less than 0.21 LSB at a 12-bit level.

[2] Ashutosh Verma and Behzad Razavi, "A 10-bit 500MS/s 55-mW CMOS ADC", *IEEE Journal of Solid State Circuits,* Vol.44, No.11, Nov.2009.[2]

Is has also used the LMS calibration technique which is a digital foreground calibration technique to correct the errors due to capacitor mismatch, gain error, and op-amp non-linearity. For a 10-bit ADC it used 13 stages. The first two stages are calibrated for residue gain error, DAC gain error and op-amp non-linearity. The next four stages for residue gain and DAC gain error only. The remaining seven are calibrated for only residue gain error. The calibration algorithm was synthesized for the estimation of power and area used. It has a complexity of about 20,000 gates. The non-linearity function in the calibration algorithm acquire a considerable area and power penality but the number of bits used to represent each number in binary are very less and are truncated if required in order to reduce area. It has used a conversion rate of 500 MHz, supply voltage of 1.2V. The results obtained are:

The uncalibrated ADC has a INL of 40 LSB. After calibration, the DNL and INL fall below 0.4 LSB and 1 LSB, respectively. Before calibration, the SNDR is 28.28 dB and after calibration, it has been improved to 55.64 dB. Total power dissipation of the calibration logic is about 8mW with a supply voltage of 1.2 V.

[3] Meysam Mohammadi, Amir Alipour, Esmaeil N. Aghdam and Khosrov D. Sadeghipour, "A New Technique for Background Calibration of Pipelined ADCs", *Electrical Engineering (ICEE), 2013 21st Iranian Conference on*. IEEE, 2013.[3]

This paper presents a new background calibration technique for 13-bit 1.5bit/stage architecture of pipelined ADC by means of slow and high accurate ADC. The algorithm used has eliminated the need for switching the extra ADC between main ADC stages and thus don't need to complex digital processing, which simplify both analog and digital circuits from power stand point.

Earlier methods used for calibration uses a matrix of correction coefficients updated by LMS which results in more complex digital circuit. The number of these coefficients can be reduced by switching an extra ADC to the input of stage under calibration. But, it increased the time of calibration and introduced the loading effect and thus increases the volume of digital processing.

Here, the problem of complex digital processing is mitigated by means of stage by stage calibration and it uses a test signal by which input frequency doesn't affect the process of coefficient estimation.

In this, errors due to finite and non-linear gain of interstage operational amplifier are calibrated. Linear gain error is corrected in all stages and third order non-linearity is corrected in the first five stages. Correction coefficients are estimated by using LMS algorithm. It used μ_1 and μ_2 = 0.1. Working in foreground mode which interrupts the normal operation of ADC for calibration, is one of the limitations of this work.

Obtained results from the simulation are:

DNL reduced to -0.8 LSB from -47.03 LSB. The INL also reduced from 66.48 LSB to 2.84 LSB. And, SNDR improves from 40.64 dB to 73.55 dB after calibration.

2.2 Literature review for fast multiplication techniques using Booth's algorithm.

[4] K.L.S Swee and L.H Hiung, "Performance Comparison Review of Radix-Based Multiplier Designs," in *4th International Conference on Intelligent and Advanced Systems(ICIAS)* , pp. 854-859, 12-14 Jun.2012. [4]

In this paper, a detailed comparison has been done between the performance of radix-2, radix-4, radix-8, radix-16 and radix-32 booth encoding multipliers. It is found that as the radix number of multiplier is increased, both the area and delay performance are degraded due to increase in complexity of the multiplier. The results obtained shows that the radix-4 booth multiplier gives best performance in terms of area and speed constraints.

2.3 Literature review for partial products reduction techniques using a variety of adders.

[5] C. Nagenndra, M.J. Irwin and R.M. Owens, "Area-Time-Power Tradeoffs in Parallel Adders," in *IEEE Transactions on Circuits and Systems-II,* Vol.43, No.10, Oct.1996. [5]

This paper has surveyed several classes of parallel, synchronous adders based on their power, delay and area characteristics. The adders studied include the linear time ripple carry adder and Manchester carry chain adders, the square root time carry skip and carry select adders, the logarithmic time carry look-ahead adder and its variations, and the constant time signed-digit and carry save adders. All the adders have been simulated using HSPICE. The ELM adder was found to be best among the two's complement adders for entire range of precision studied.

[6] J. Kaur, N.K. Gaahlan and P. Shukla, "Delay-Power Performance Comparison of Array Multiplier in VLSI Design," in *International Journal of Advanced Research in Computer Science and Electronics Engineering,* Vol. 1, No.3, May.2012.[6]

It used compressors for increasing the speed of the multiplier. The conventional multiplier was compared with the multiplier containing compressors. It is found that the use of compressors not only reduced the vertical critical path but also the stage operations. The use of compressors in multipliers greatly reduced the time delay of the multiplier.

[7] P. Gurjar, R. Solanki, P. Kansliwal and M. Vucha, " VLSI Implementation of Adders for High Speed ALU," in Proc. *India Conference(INDICON),* pp.1-6,16-18 Dec.2011.[7]

This paper described the construction of high speed adder circuit using Hardware Description Language (HDL). The motivation behind this is that an adder is a very basic building block of arithmetic and plays an important role in determining the performance of central processing unit (CPU). Various adders have been simulated and synthesised like full adder, ripple carry adder, carry look-ahead adder, carry skip adder, carry select adder and carry save adder. The simulated results are verified and

functionality of high speed adders was compared for parameters like speed, area *etc*. The result obtained for 8-bit, 16-bit and 32-bit concluded that carry save adder is the best adder in terms of speed and area consumption.

2.4 Literature review for binary floating point multiplier.

[8] A. Jain, B. Dash and A. Panda, "FPGA Design of a Fast 32-bit Floating Point Multiplier Unit," in Proc. *International Conference on Devices, Circuits and Systems(ICDCS)*, pp.545-547,15-16 Mar.2012. [8]

This paper has proposed an architecture for a fast 32-bit floating point multiplier. The design was planned to make a multiplier faster by reducing the delay caused by the propagation of carry by implementing adders having the least power delay constant. After comparing various adders on basis of power and delay, it has been concluded that Kogge-Stone adder is the fastest. The main idea behind this design is to increase the speed of multiplier by reducing delay at every stage using the optimal adder design. Kogge-stone adder has been used at each stage of the multiplier to increase the speed of the multiplier.

[9] G. Renxi,Z. Shangjun, Z. Hainman and M. xiaobi, "Hardware Implementation of a High Speed Floating Point Multiplier Based on FPGA," in Proc. *International Conference on Computer Science & Education(ICCSE)*, pp.1902-1906,25-28 Jul.2009. [9]

A high speed floating point multiplier has been implemented in this paper. In the design of the floating point multiplier, a new Radix-4 Booth's encoding algorithm has been used for generating the partial products. For the reduction of the partial products, the improved 4:2 compression structure has been introduced. The final sum and carry vectors generated by compressors are added by a carry look-ahead adder to obtain the final product. The timing simulation results show that the floating point multiplier can be stably run at the frequency of 80 MHz

Chapter 3

STUDY OF WORKING OF PIPELINED ADC AND ITS ERRORS

3.1 Working of pipelined ADC

The overview of the architecture of pipelined ADC is already discussed in section 1.3. Here, in this chapter, its detailed working is explained.

Pipeline ADC requires a two phase, non-overlapping clock signal say, ϕ1 and ϕ2 as shown in Fig. 3.1 such that ϕ1. ϕ2=0 every time so that all the stages of the pipeline ADC may work in parallel.

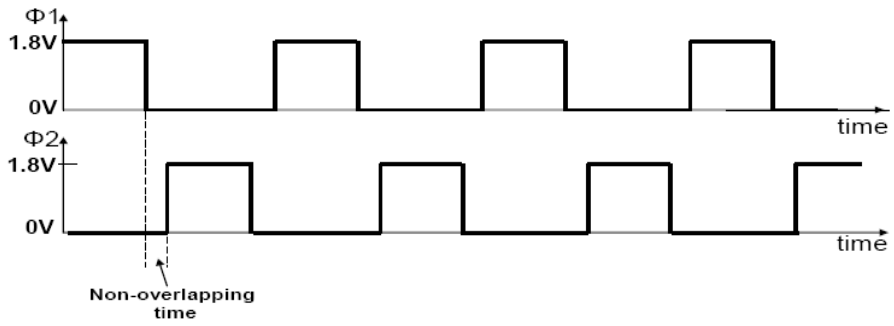

Fig. 3.1: Two phase non- overlapping clock.

So, during ϕ1 phase, each stage of the pipelined ADC will sample the input coming from the previous stage. For example, stage 2 will sample the output residue voltage of stage 1 and at the same time stage 1 will sample the input voltage V_{in}. So, during ϕ1, all the stages are operating in parallel in their sampling phase. Also the sub-ADC of each stage convert the input of that stage into 2 bit digital output in this phase only.

Second phase is when ϕ2 will be on. During this phase, residue is calculated by every stage in parallel and the difference is amplified by a factor of 2 and passed on to the next stage. Thus it is called amplification phase. Therefore, total twelve clock pulses are required for a 12- bit pipelined ADC to generate the digital result.

Within a stage, when the S/H circuit will sample the signals, at the same time amplifier will amplify the residue *i.e.* ϕ2 will be high. And, when sample and hold circuit is in hold phase, the sampled signal will be converted to digital by sub-ADC of the stage *i.e.* ϕ1 will be high. Likewise, every stage operates as depicted in Fig. 3.2.

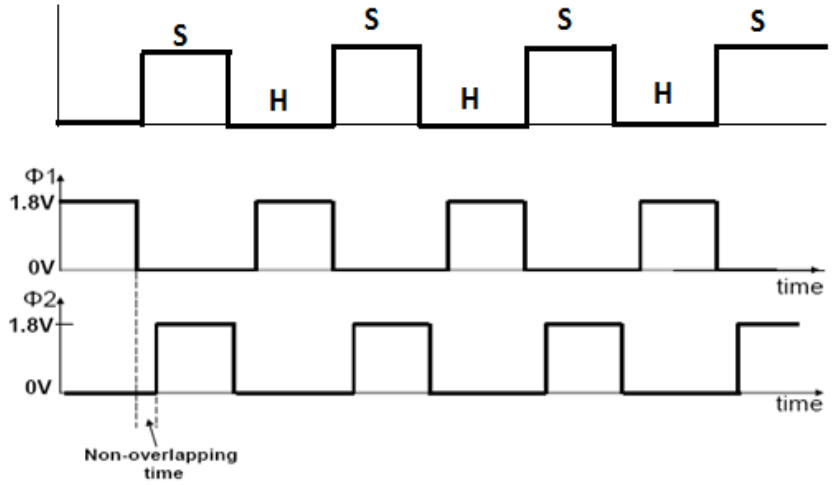

Fig. 3.2: Sample and hold phase operation of each stage.

3.2 1.5 bit ADC stage operation

Switched capacitor implementation of one stage is shown in Fig. 3.3. It consists of a sub-ADC, a DAC and the operational transconductance amplifier (OTA) for amplification purpose. The sub-ADC consists of a comparator and an encoder to encode into binary. And, DAC is a multiplexor to select its V_{dac} according to the sub-ADC output. C_s and C_f are input capacitors and C_L is the parasitic load capacitor.

3.2.1 Sampling phase

During the first phase when ϕ1 is high, M1, M2 and M4 are on as shown in Fig. 3.4. The input signal V_{in} ranging from $-V_{ref}$ to $+V_{ref}$ is applied to the input of the sub-ADC. Simultaneously, V_{in} is applied to sampling capacitors C_s and C_f. At the end of the first clock phase, V_{in} is sampled across C_s and C_f, and the output of the sub-ADC is latched *i.e.* 2 bit output B1 and B0. This output could be 00, 01 or 10 depending upon whether the input signal is less than $-V_{ref}$ /4, between $-V_{ref}$ /4 and $+V_{ref}$ /4 or greater than $+V_{ref}$ /4 respectively. According to the sub-ADC output, DAC will

calculate its output V_{dac} and store it in parasitic capacitor C_p. As M4 is on, so due to the virtual short property of op-amp, the right plate of capacitor C_s will get V_{cm}.

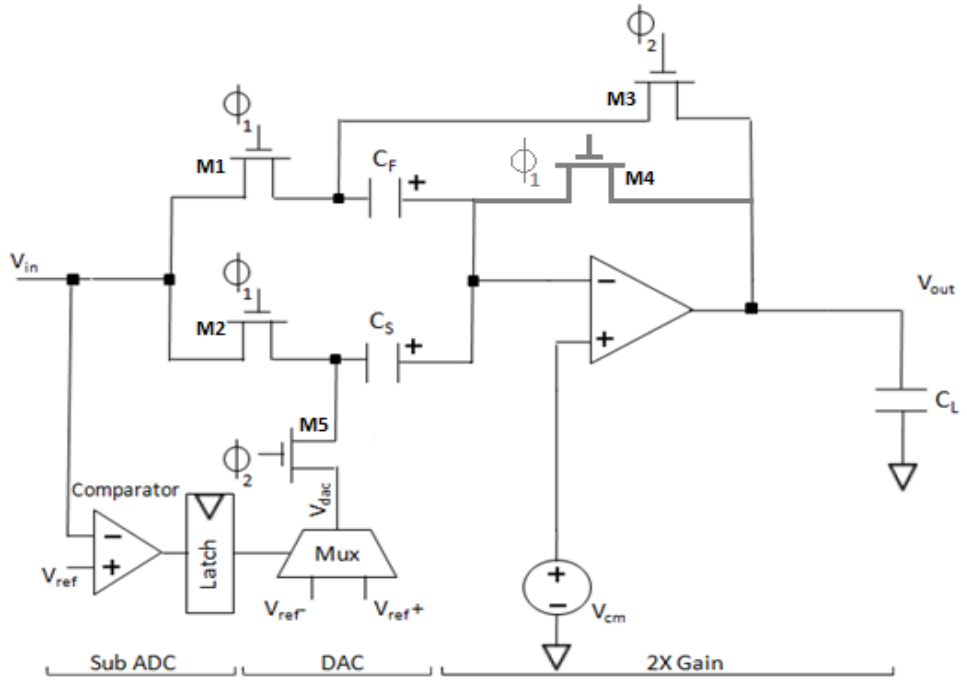

Fig. 3.3: Switched capacitor implementation of each pipeline stage.

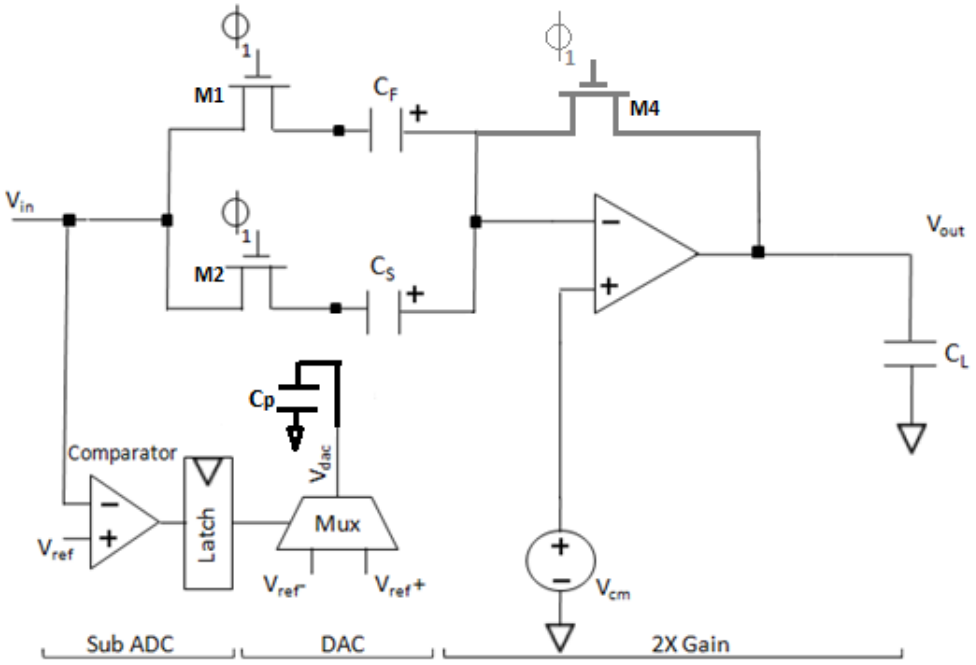

Fig. 3.4: Sampling phase operation of one stage.

3.2.2 Amplification phase

During the second clock phase when ϕ2 is high, M4 and M5 will be on and all other switches off as shown in Fig. 3.5. C_f closes a negative feedback loop around the op amp, thus the positive plate of C_f will get connected to the V_{out}. And, the top plate of C_s is switched to the digital-to-analog converter (DAC) output. This configuration generates the stage residue at V_{out}.

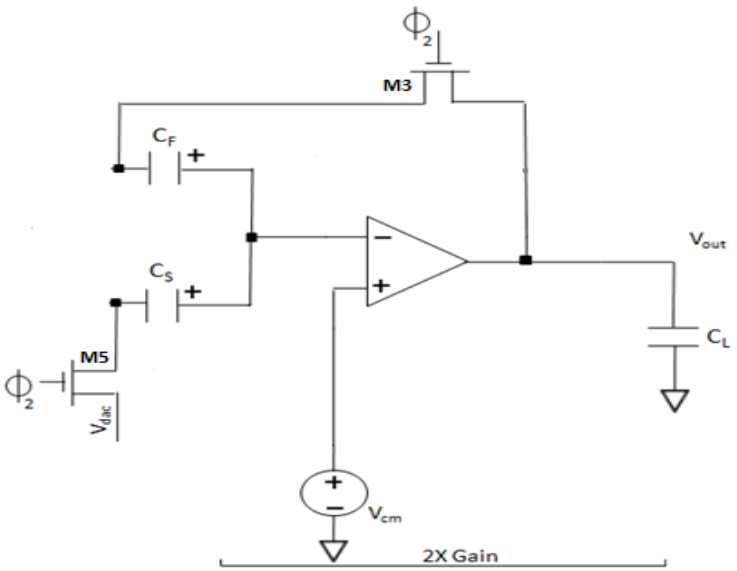

Fig. 3.5: Amplification phase configuration.

3.3 Mathematical Analysis

During ϕ1, C_s and C_f are charged to $V_{cm} - V_{in}$. Therefore, the charge stored Q_s in sampling phase on both the capacitors could be written as:

$$Q_s = (V_{cm} - V_{in})(C_s + C_f) \tag{6}$$

Where, V_{cm} is the dc bias voltage required to keep the input MOS of op-amp in saturation.

During ϕ2, charge stored in amplification phase is given by:

$$Q_a = (V_{cm} - V_{dac})C_s + (V_{cm} - V_{out})C_f \tag{7}$$

Where V_{dac} is the output voltage from DAC. Now applying charge conservation theorem,

$$Q_s = Q_a \tag{8}$$

Solving equations (6), (7) and (8) for V_{out} ,

$$V_{out} = \left(1 + \frac{C_s}{C_f}\right) V_{in} - \left(\frac{C_s}{C_f}\right) V_{dac} \tag{9}$$

If C_s=C_f (normally equal to 0.1pF) then,

$$V_{out} = 2V_{in} - V_{dac} \tag{10}$$

Where V_{out} is the residue voltage generated by the stage.

If $V_{in} > V_{REF}/4$, V_{dac} will be V_{REF}, which means V_{out} by (10) will be

$$V_{out} = 2V_{in} - V_{REF} \tag{11}$$

If $-V_{REF}/4 < V_{in} < +V_{REF}/4$, V_{dac} will be zero, thus V_{out} becomes:

$$V_{out} = 2\ V_{in} \tag{12}$$

If $V_{in} < -V_{REF}/4$, Vdac will be $-V_{REF}$, thus V_{out} will be given by:

$$V_{out} = 2\ V_{in} + V_{REF} \tag{13}$$

3.4 Errors in pipelined ADC, their causes and effects

The following types of errors are suspected as we implement the pipeline ADC at lower technology thus making the ADC non-ideal.

3.4.1 Finite open loop gain, A_o of the op-amp

Scaling the technology affects the gain. At lower technologies, the op-amp becomes non-ideal as it keeps on fluctuating its open loop gain. Thus it will no longer be infinite as assumed earlier. Let us suppose, it has a finite open loop gain A_o and C_p is the input parasitic capacitance of the op-amp as shown in Fig. 3.6. Now, this V_X will no longer be virtually ground but will have a finite value.

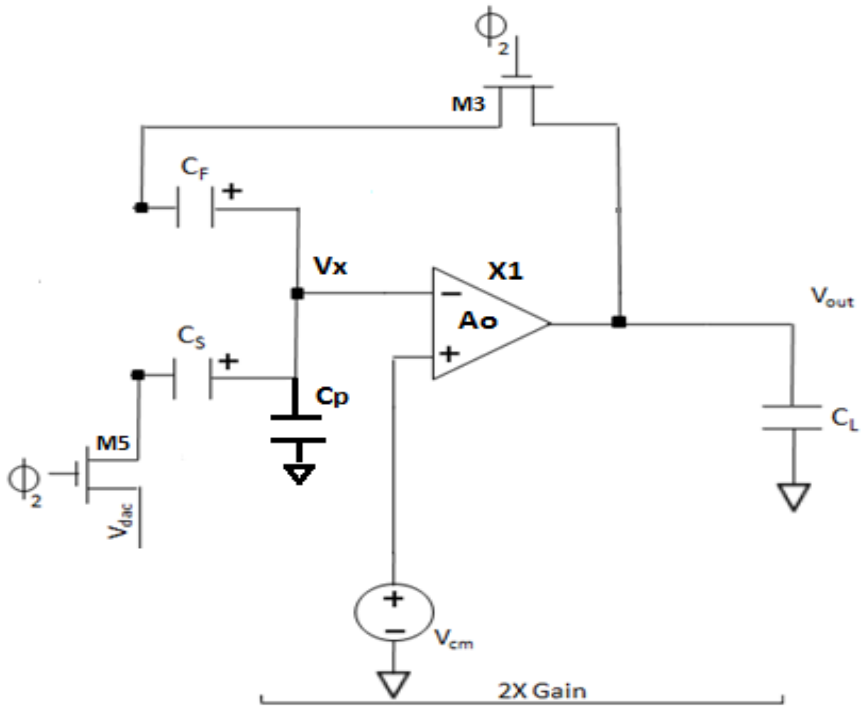

Fig. 3.6: Op-amp with open loop gain A_o in amplification phase.

During sampling phase when ϕ1 is on, total charge on the capacitors C_s and C_f is given by:

$$Q_s = (V_X - V_{in})(C_s + C_f) \tag{14}$$

During amplification phase when ϕ2 is on, total charge on C_s and C_f is given by:

$$Q_a = (V_X - V_{dac})C_s + (V_X - V_{out})C_f + V_X C_p \tag{15}$$

Similarily, applying charge conservation principle,

$$Q_s = Q_a \tag{16}$$

Solving (14), (15) and (16) for V_{out}, we get:

$$V_{out} = \left(1 + \frac{C_s}{C_f}\right)V_{in} - \left(\frac{C_s}{C_f}\right)V_{dac} + \left(\frac{C_s + C_f + C_p}{C_f}\right)V_X \tag{17}$$

Where $\left(\frac{C_f}{C_s + C_f + C_p}\right)$= β is called the feedback factor. (18)

Also, V_X can be expressed as

$$V_X = -\frac{V_{out}}{A_o} \tag{19}$$

Solving (17), (18) and (19) for V_{out} gives:

$$V_{out} = \left(1 + \frac{C_s}{C_f}\right) V_{in} - \left(\frac{C_s}{C_f}\right) V_{dac} - \frac{V_{out}}{\beta A_o} \tag{20}$$

Which further implies,

$$V_{out} = \frac{\left(1+\frac{C_s}{C_f}\right)V_{in} - \left(\frac{C_s}{C_f}\right)V_{dac}}{1+\frac{1}{\beta A_o}} \tag{21}$$

By Taylor's expansion,

$$V_{out} = \left[\left(1 + \frac{C_s}{C_f}\right) V_{in} - \left(\frac{C_s}{C_f}\right) V_{dac}\right]\left[1 - \frac{1}{\beta A_o}\right] \tag{22}$$

Comparing this result to that of the ideal one, it is inferred that there is a fractional error = ($\frac{1}{\beta A_o}$) in the V_{out} due to the finite open loop gain.

Justifying it, if A_o is increased to a much higher value (or infinite), this error term will reduce proportionally making the ADC close to the ideal one.

For a N bit ADC with B bits per stage architecture, the first stage gain error should be less than ½ LSB of the full range of the second stage $\left({FS}/{2^{N-B}}\right)$ to prevent any missing codes. Therefore,

$$\frac{1}{\beta A_o} < \frac{1}{2}\left(\frac{FS}{2^{N-B}}\right) \tag{23}$$

Which means open loop gain should be greater than,

$$A_o > \frac{2(2^{N-B})}{\beta\ FS} \tag{24}$$

If V_{REF} is taken as 5, then FS=5-(-5) =10 which implies A_o = 579.35(approx.). Practically, it should be even larger than this in order to compensate for capacitor mismatch effects and change in feedback factor due to input parasitic capacitance of op-amp.

3.4.2 Errors due to charge injection

In switched capacitor circuits, problems occur due to channel charge injection. As switches are modelled by MOSFETs, so for a MOSFET to be on, a channel must exist at the oxide silicon surface. The total charge in the channel could be expressed as:

$$Q_{ch} = WLC_{ox}(V_{DD} - V_{in} - V_{th}) \tag{25}$$

Where W and L are width and effective channel length of MOSFET, C_{ox} is the gate capacitance and V_{th} is the threshold voltage.Now, charge injection is the phenomenon that when a MOSFET is turned off, the channel charge Q_{ch} exits through the source and drain terminals as depicted in Fig. 3.7.

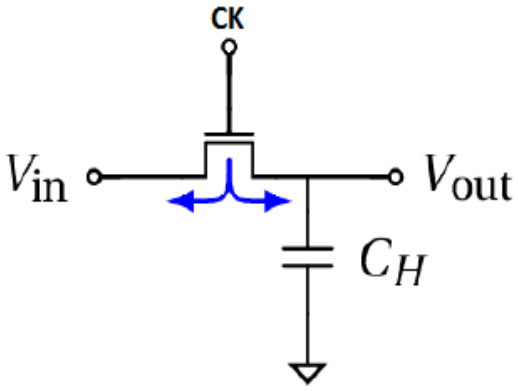

Fig. 3.7: Charge injection when switch turns off.

The charge injected to the left side is absorbed by the input source causing no error. But the charge injected to the right side will introduce an error in the charge stored on sampling capacitors C_s and C_f in the pipelined ADC architecture leading to offset errors. For example, if half the charge is injected to the right side, the error corresponding to it will be

$$\Delta V = \frac{WLC_{ox}(V_{DD} - V_{in} - V_{th})}{2C_H} \tag{26}$$

As a worst case estimate, it is assumed that the whole channel charge gets deposited on capacitor C_H, changing the output voltage V_{out} by $2\Delta V$. Thus V_{out} becomes:

$$V_{out} = V_{in} - \frac{WLC_{ox}(V_{DD} - V_{in} - V_{th})}{C_H} \tag{27}$$

3.4.3 Errors due to capacitor mismatch

Idealy, for a single stage, if DAC output , V_{DAC} =$K_1 D_1$Vref (where K_1 is the gain of DAC assumed to be 1), the input of single stage could be written as :

$$V_{in} = K_1 D_1 V_{ref} + \frac{1}{A1} V_{out} \tag{28}$$

If residue amplifier is implemented by a closed loop architecture (shown below) in Fig. 3.8.

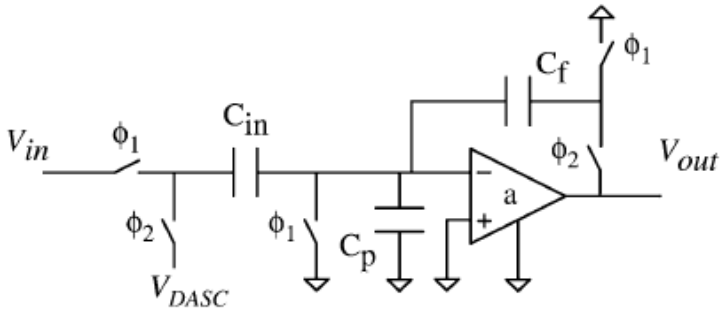

Fig. 3.8: charge transfer closed loop residue amplifier. [1]

Here, the input V_{in} is sampled onto C_{in} during phase 1, while C_f is reset. During phase 2, V_{DAC} is applied to the input causing charge to transfer to C_f . The input V_{in} in this case is

$$V_{in} = K_1 D_1 V_{ref} + \left(\frac{C_f}{C_{in}} + \frac{1}{a}\left(1 + \frac{C_p + C_f}{C_{in}}\right)\right) V_{out} \tag{29}$$

If the op-amp gain a→ ∞, the residue gain A_1 is C_{in}/C_f, which depends only on capacitor matching. Thus the residue gain of closed loop amplifier depends on both capacitor matching and op-amp gain (high).

Generally, the input capacitors C_s *and* C_f are of range 1pF. Any fluctuation in their value is possible leading to mismatch between them let say mismatch of 0.1 (e=0.1). So, C_s *and* $C_f$could be written as:

$$C_f = C(1 + e) \tag{30}$$

$$C_s = C(1 - e) \tag{31}$$

Now, the op-amp characteristics could be written as output y is a function of x. Output of op-amp is V_{out} and input is V_X . Therefore, it could be written as:

$$V_{out} = f(-V_X)$$

Or , $$V_X = -f^{-1}(V_{out})$$

The effect of finite open loop gain has already been derived in section 4.1 of this chapter. Now, capacitor mismatch can be introduced in the same equation. So equation (17) is:

$$V_{out} = \left(1 + \frac{C_s}{C_f}\right) V_{in} - \left(\frac{C_s}{C_f}\right) V_{dac} + \left(\frac{C_s + C_f + C_p}{C_f}\right) V_X$$

It can also be modelled as under:

$$V_{out} = \left(1 + \frac{C_s}{C_f}\right) V_{in} - \left(\frac{C_s}{C_f}\right) D_{out} V_{REF} - \left(\frac{C_s + C_f + C_p}{C_f}\right) f^{-1}(V_{out}) \quad (32)$$

Putting (30) and (31) in (32) gives:

$$V_{in} = \frac{1}{2}(1 - e) D_{out} V_{REF} + \frac{1}{2}(1 + e) V_{out} + \left(1 + \frac{C_p}{2C}\right) f^{-1}(V_{out}) \quad (33)$$

I II III

Term I – represents DAC gain error

Term II – represents residue gain error, and

Term III – represents non-linearity of op-amp

Thus capacitor mismatch and finite open-loop gain together leads to these three error *viz*. DAC gain error, residue gain error and op-amp non-linearity.

Term II and III together can be approximated by a third order polynomial as:

$$V_{in}(approx.) = \alpha_1 V_{out} + \alpha_3 {V_{out}}^3 + \frac{1}{2}(1 - e) D_{out} V_{REF} \quad (34)$$

And the term I can also be written as $w_j D_{out}$ where,

$$w_j = \frac{1}{2}(1 - e) V_{REF} \quad (35)$$

If V_{REF} is 1 and no mismatch error, *i.e.* in ideal case, w_j equals to 0.5.

Justification for the third order assumption:-

For this transient analysis is performed on the op-amp with e = 0 and taking V_{in} accordingly so that DAC output is zero. By this the equation (34) will get reduced to

$$V_{in}\,(approx.) = \alpha_1 V_{out} + \alpha_3 V_{out}{}^3 \qquad (36)$$

Now, V_{in} is varied from $-0.5\,V_{REF}$ to $+0.5\,V_{REF}$ and V_{out} is measured. Keeping the values of V_{in} and V_{out} in reduced equation, estimate the best fit values for α1 and α3 for which the mean square error between actual V_{in} and V_{in} (approx.) is less than 1 LSB. If it happens, then the third order approximation is justified.

The graph shown in Fig. 3.9, shows the variation of error in the estimation of V_{in} assuming a third order polynomial with α1= 0.52 and α3 = 0.004.The Fig. 3.9 depicts that the maximum error is of 0.3464 which is approximately 0.14 LSB and therefore acceptable.

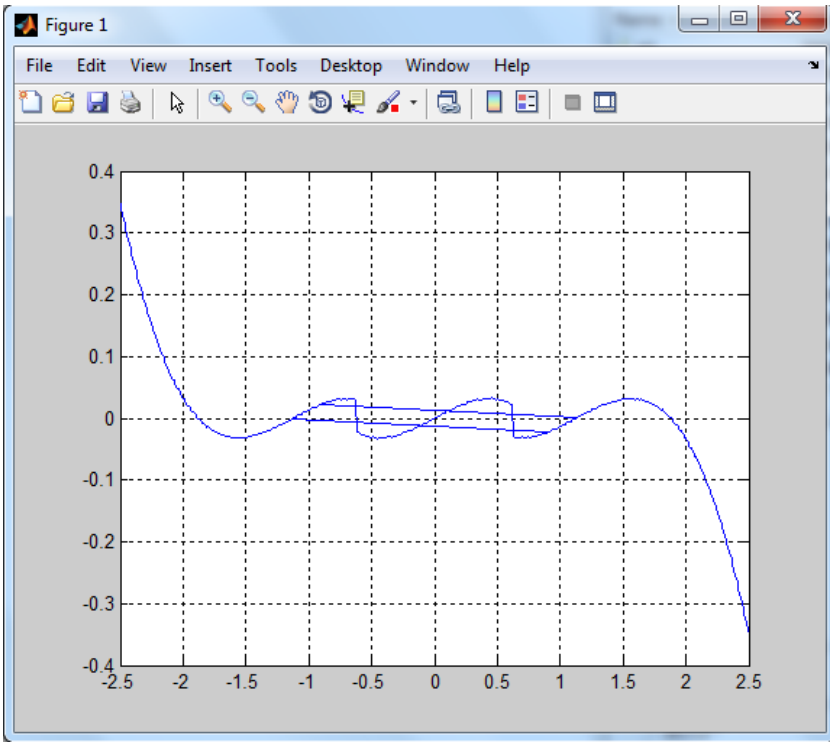

Fig. 3.9: Error in estimation of V_{in} (y-axis) vs input voltage (x-axis) with third order polynomial with V_{REF} =5.

Chapter 4

LEAST MEAN SQUARE (LMS) CALIBRATION ALGORITHM & ITS IMPLEMENTATION

4.1 Calibration concept

Calibration starts from the last stage. Each stage will use a digital correction logic explained later in this chapter and output from that stage is generated which will be almost equal to the input to that stage. Means, that particular stage has become closer to ideal. Similarly, the idealization will keep on moving from last stage to front stage.

To understand the process more briefly, let us consider one-stage of a pipeline ADC as shown in Fig. 4.1. For this stage, all the stages following it are already idealized. For example, stage j can assume stages (j+1) and higher are ideal. So, they are named as ideal backend ADC as they will output the correct digital output D_{BE} which will be used by this stage for its calibration or idealization.

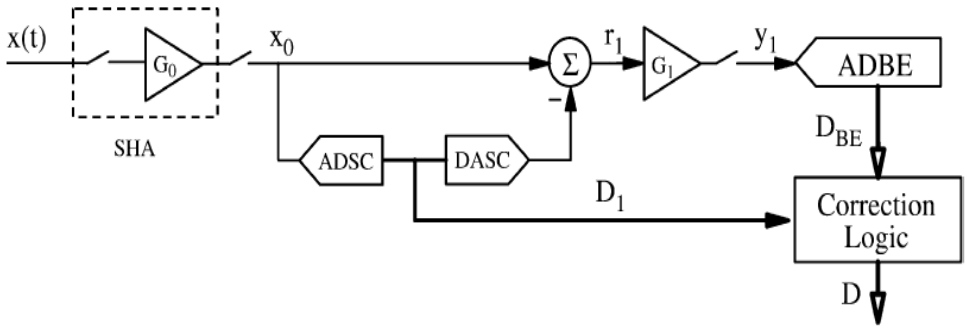

Fig. 4.1: A single pipelined converter stage to be calibrated by correction logic. [1]

Ideally, the correct digital output of a stage is given by:

$$D_{corr} = D_1 + \frac{1}{G_1} D_{BE} \tag{37}$$

Where D1 is the two bit digital output of that stage, G1 is the closed loop gain of the op-amp equals to 2 and D_{BE} is the ideal output from the next stages *i.e.* D_{corr} from the stages following this stage.

But due to the errors discussed like finite open loop gain, non-linearity *etc*, this G1 is no longer equal to 2. Therefore, the output will not be ideal. Thus a calibration is required for every stage.

For each stage, g_j(x) denotes the residue calculation from the input V_{in}.

And (1/G1) = 0.5 (in ideal case) which is a inverse function of amplification, is approximated as a third order polynomial denoted by ${g_j}^{-1}(x)$ for non-ideal stage as discussed already in section 4.3.

$$ {g_j}^{-1}(x) = \alpha_1 x + \alpha_3 x^3 \tag{38} $$

From section 4.3, the non-ideal behaviour is modelled as under which is the base of calibration algorithm

$$ V_{in}\,(approx.) = \frac{1}{2}(1-e)D_{out}V_{REF} + \alpha_1 V_{out} + \alpha_3 {V_{out}}^3 \tag{39} $$

To make it easy to analyse this equation being conceptually similar to equation (37), it can be written as under and shown in Fig. 4.2.

$$ V_{in}\ (approx) = \frac{1}{2}(1-e)D_{out}V_{REF} + {g_j}^{-1}(D_{BE}) \tag{40} $$

Where V_{out} and D_{BE} will be nearly equal (with some quantization noise) due to the idealization of the backend stages.

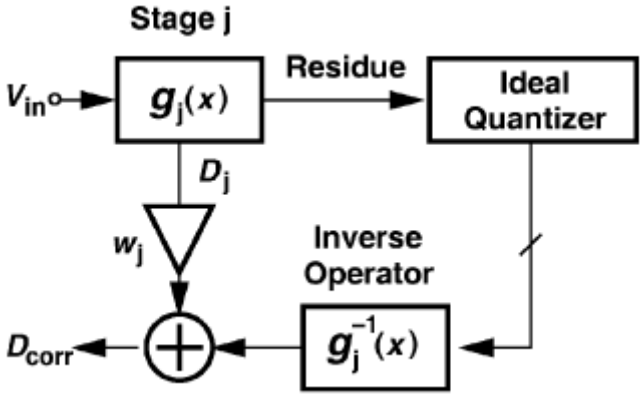

Fig. 4.2: Calibration concept. [2]

Since, equation (39) has modelled the effect of all the errors *viz.* DAC gain error, residue gain error due to capacitor mismatch and non-linearity of op-amp, hence it can be used to calibrate each stage of the pipelined ADC.

4.2 Calibration procedure

Consider one stage of the pipelined ADC named as stage j to be calibrated, therefore the stages next to it are considered ideal. The calibration arrangement is shown in Fig. 4.3. The reference DAC provides dc input to stage j, the sub-ADC produces the digital output $D_{out,j}$ and the backend stages generate D_{BK}. This D_{BK} is the actual digital representation of the residue of stage j being passed on to the next stages. Now, D_{BK} is applied to inverse function ${g_j}^{-1}(.)$ so as to undo the non-linearity created by stage j. Then the result is combined with $w_j D_{out,j}$ to calculate the overall output D_{tot} according to the following equation

$$D_{tot} = \alpha_1 D_{BK} + \alpha_3 {D_{BK}}^3 + w_j D_{out,j} \tag{41}$$

In ideal case, this D_{tot} must be equal to digital representation by reference DAC of V_{in} which is denoted here by D_{cal}. In non-ideal case, there will be a difference in D_{tot} and D_{cal} called error which is to be minimized by calibration algorithm.

To minimize this error, Least Mean Square (LMS) algorithm is used.

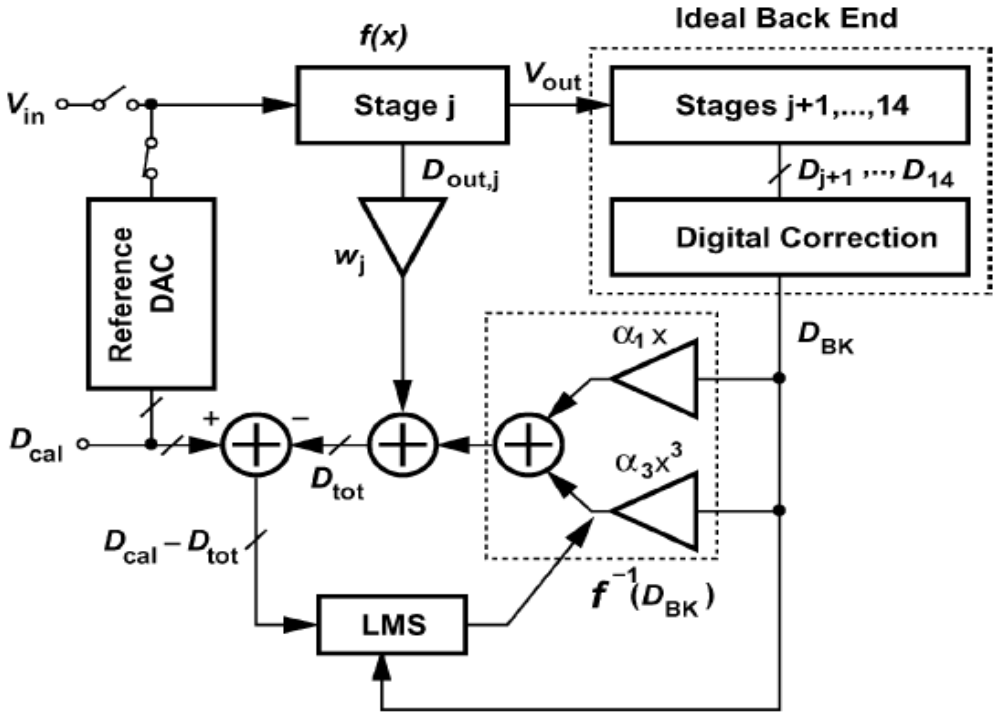

Fig. 4.3: Digital calibration of stage j.[2]

4.3 LMS algorithm

In contrast to traditional calibration techniques, it is a new calibration technique that treats analog imperfections in analogy to distortion in communication channels can be employed to remove errors from all pipeline stages simultaneously using an adaptive digital filter. With this approach, the analog signal path is not disturbed during calibration and thus can maintain maximum conversion speed allowed by a certain technology. This approach is able to correct errors caused by capacitor mismatch, signal-dependent finite op-amp gain, and switch induced offset error. By relaxing requirements on precision matching and high open-loop op-amp gain, least-mean-square (LMS) calibration method can improve conversion accuracy and speed and can reduce power consumption.

Using LMS in calibrating the pipelined ADC stage, consists of two steps: estimation of α_1 and α_3 without interaction with w_j, and estimation of w_j with α_1 and α_3 set properly.

a) Estimation of α_1 and α_3

Firstly, the dc inputs between the range $+V_{REF}/4$ and $-V_{REF}/4$ are produced by the reference DAC to be applied to the input to stage j, so that it is made to work only as a multiply by 2 circuit such that $D_{out,j} = 0$. Thus, from the Fig. 4.3 above,

$$D_{tot} = \alpha_1 D_{BK} + \alpha_3 {D_{BK}}^3 \tag{42}$$

Next, the LMS algorithm will adjust α_1 and α_3 in order to drive the mean square error (D_{cal} -D_{tot}) to zero. The adjustment of these parameters require their updation which is described by following equations:

$$\alpha_1(k+1) = \alpha_1(k) + \mu_1(D_{cal} - D_{tot})D_{BK} \tag{43}$$

$$\alpha_3(k+1) = \alpha_3(k) + \mu_3(D_{cal} - D_{tot}){D_{BK}}^3 \tag{44}$$

b) Estimation of w_j

The term w_j corrects the effect of capacitor mismatch on DAC operation (as depicted from first term in equation (39)). In this case, the mode of operation of the pipeline ADC is regular without any boundation on the input *i.e.* all the three operations are performed sampling, sub-ADC and DAC operation and multiplication by 2. For any value of input, the D_{tot} in this case is:

$$D_{tot} = \alpha_1 D_{BK} + \alpha_3 {D_{BK}}^3 + w_j D_{out,j} \quad (45)$$

The value of w_j is adjusted so as to minimize the difference of D_{cal} and D_{tot}. The error which is less than ½ LSB is acceptable.

c) Backend stages

As the backend stages contribute less in the total error because the backend stages error terms are divided down by the preceding stages interstage gains. Thus the later stages require lesser accuracy. Therefore, a lesser complexity calibration algorithm is used for them. The last seven stages are calibrated only for residue gain error. Since

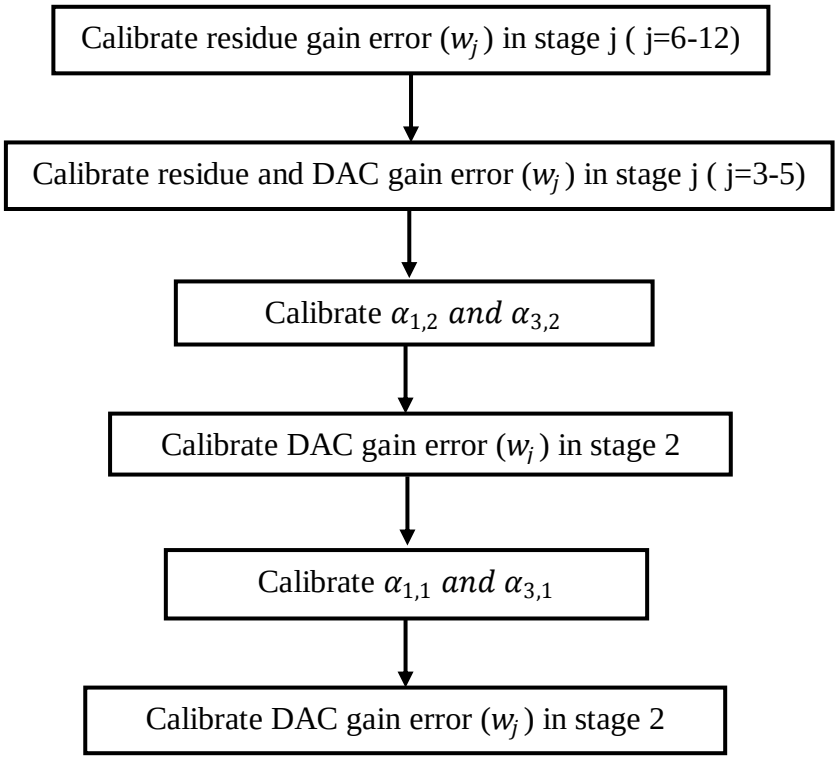

Fig. 4.4: Calibration flow chart.

the capacitor mismatch is negligible here, only the finite gain of the op-amp produces gain error. Next, the stages 3, 4 and 5 are calibrated for both residue gain error and DAC gain error. The first two stages are calibrated fully for all the errors *i.e.* residue gain error, DAC gain error and non-linearity of op-amp. Fig. 4.4 shows a flow chart following this scheme of calibration .

The whole calibration arrangement for the 12-bit pipelined ADC is shown below in Fig. 4.5.

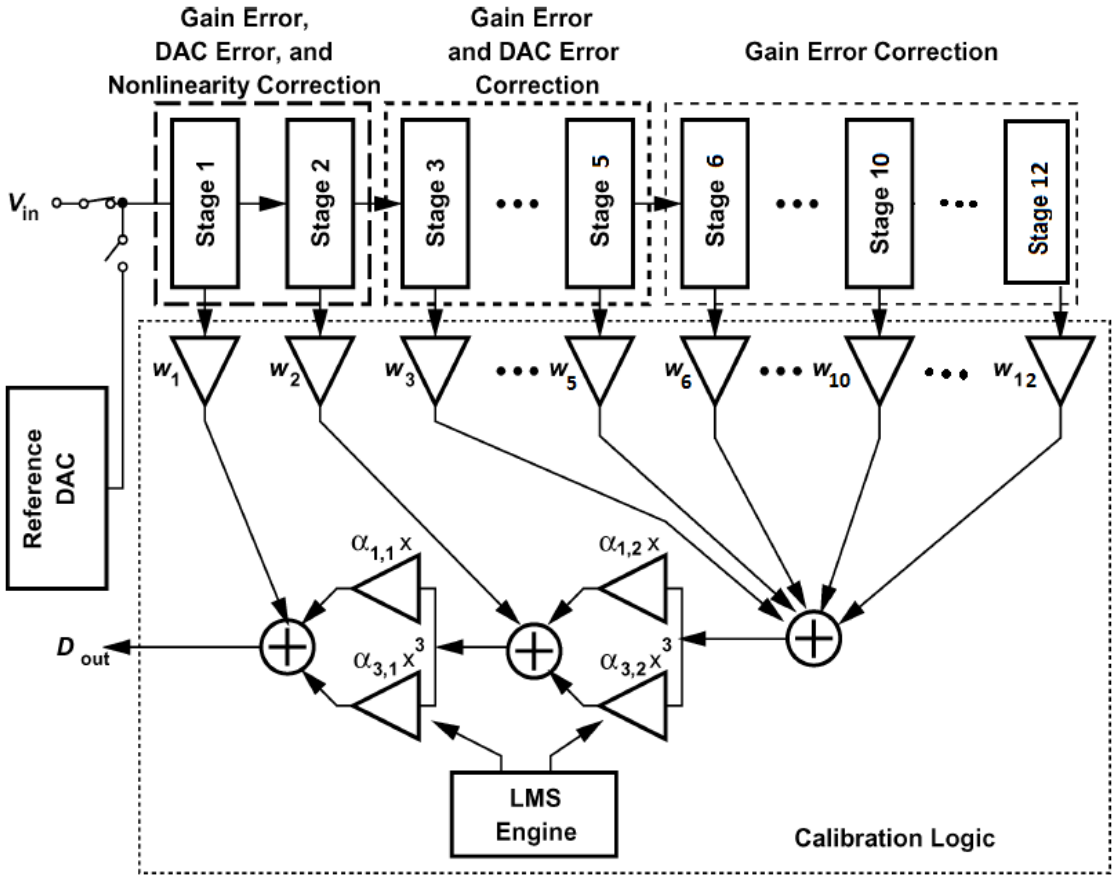

Fig. 4.5: 12 bit Pipeline ADC architecture with calibration.

4.4 Implementation of LMS algorithm

The implementation of the above algorithm is done using verilog in order to generate a calibration hardware and is implemented on FPGA to verify it. Modelling equation (41) in verilog requires floating point multipliers and adders that too signed because

the terms can have any value either positive or negative. Therefore, the designing of binary floating point multipliers and adders is discussed in further sections of this chapter.

The terms to be multiplied or added are in decimal, so they must be first converted to binary to perform multiplication. There are various binary representation formats for it which are discussed in next sections. Next, to reduce the time delay during multiplication, a technique called booth algorithm is discussed. Then if the partial products generated by booth algorithm are large in number, then to reduce the time delay in adding them to generate multiplication result, various adders are discussed. Next comes the floating point adder.

4.4.1 Binary floating point representation

To represent binary numbers, an IEEE -P754 standard is used. It was developed by IEEE in 1985. The revised version of this standard was IEEE P754-2008 which was introduced in 2008.

There are six basic formats specified by IEEE-754 to represent floating point numbers in binary. Out of them, four are binary and two decimal formats. The binary formats are called half, single, double and quadruple precision while the decimal formats are known as double and quad. These formats are different in terms of their precision, exponent range and radix and each format represents a unique set of floating-point data. Table 4.1 shows the basic formats used for floating point numbers.

Table 4.1: IEEE basic formats for floating point numbers.

Basic format	Base value	Digits or bits	Exponent maximum
Binary16	2	10 bits	+15
Binary32	2	23 bits	+127
Binary64	2	52 bits	+1023
Binary128	2	112 bits	+16383
Decimal64	10	16 bits	+384
Decimal128	10	34 bits	+6144

As in the algorithm to be implemented for calibration of ADC, the magnitude of numbers is not very large. Thus, a format with lesser number of bits is preferred in order to reduce the area of the calibration hardware. The table signifies that Binary16 *i.e*. half precision has least number of bits. So, it is the best among all to use.

1) Half precision format:

In half precision floating point format, 16 bits are used to represent a number. To store the sign of a number, the most significant bit is used. For exponent storage, next lower 5 bits are used and remaining 10 bits are used for storing significant of a number. Fig. 4.6 shows the representation of half precision floating point number. To make negative numbers possible to represent and to avoid extra bit for sign, a bias value of 15 is added to the actual exponent and then it is represented.

So, the IEEE P754 standard defines a binary16 format as below:

- Sign bit: 1 bit
- Exponent width: 5 bits
- Mantissa precision: 10 bits

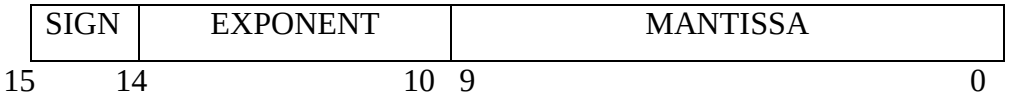

Fig. 4.6: Representation of half precision format for floating point number.

2) Rounding rules:

IEEE P754 standard defines five rounding modes which are RoundTiesToAway (RTA), RoundTiesToEven (RTE), RoundTowardNegative (RTN), RoundTowardZero (RTZ) and (RTP). The first two rounding modes are round to nearest modes and rest three modes are direct rounding modes. In RoundTiesToEven mode, if the number falls in the middle, then it will be rounded to nearest value with an even LSB. By default, this rounding mode is used for binary floating point numbers. In RoundTiesToAway mode, if the number falls in the middle, then it is rounded to nearest value below for negative numbers and above for positive numbers.

RoundTowardZero mode directly rounds the result towards zero while RoundTowardPositve and RoundTowardNegative modes rounds towards positive and negative infinity.

3) Exceptions:

This standard defines five exceptions. These are divide by zero, invalid operation, underflow, overflow and inexact. When a operation is performed which is not defined in the given standard, then invalid operation will occur. For example, square root of a negative number will result in invalid operation. Divide by zero exception will occurs if a given number is divided by zero or result a infinite number. If a given number is too large to fit in exponent field, then overflow will occur. If a number is too small to fit in exponent field, then underflow will occur. If the obtained result is by default incorrect or inexact, then inexact exception will occur.

4) Example:

For example, to represent 0.52, the binary of 0.52 is 0.10000101000111101011 1...and so on. Firstly, this has to be adjusted or shifted to get a significant 1 before decimal point. It is called normalization of a number. Therefore, it becomes:

$$1.00001010001111010111\ldots\ldots\times 2^{-1}$$

Now, exponent is -1, a negative number, so a bias of 15 has to be added to represent it in 5 bits. After adding bias exponent becomes 14, which can be represented in 5 bits as "01110". As the number 0.52 is positive, hence sign bit is '0'. After normalizing the number, the first 10 bits after the decimal point are used to represent mantissa.

Hence, the Binary16 format representation of 0.52 is shown in Fig. 4.7.

SIGN	EXPONENT	MANTISSA
0	01110	0000101000

15 14 10 9 0

Fig. 4.7: Representation of the number 0.52 in Binary16.

4.4.2 Multiplying two Binary16 format floating point numbers:

Floating multiplication is similar to the integer multiplication. In addition it involves some extra steps. The multiplication of mantissa part is similar to unsigned integer multiplication. But exponent calculation, sign calculation and normalization are extra operations that must be carried out. The general equation for a normalized floating point number can be written as

$$Z = (-1)^S \times 2^{E-bias} \times (1.M) \tag{46}$$

Where M is mantissa part, E is exponent part and S is sign bit of the number. If two numbers of such a kind has to be multiplied, then different steps are needed to be carried out for obtaining the correct result. After multiplication of the two numbers, the equation for resultant number can be written as

$$Z = (-1)^{S1\ xor\ S2} \times 2^{E1+E2-bias} \times (1.M1 \times 1.M2) \tag{47}$$

where S1 and S2 are signs of two numbers, E1 and E2 are exponents of two numbers and M1and M2 are mantissas of two numbers. Fig. 4.8 shows the complete block diagram for floating point multiplier. The various steps involved in calculating the product are calculation of sign, calculation of exponent and calculation of significand. The steps included are described below.

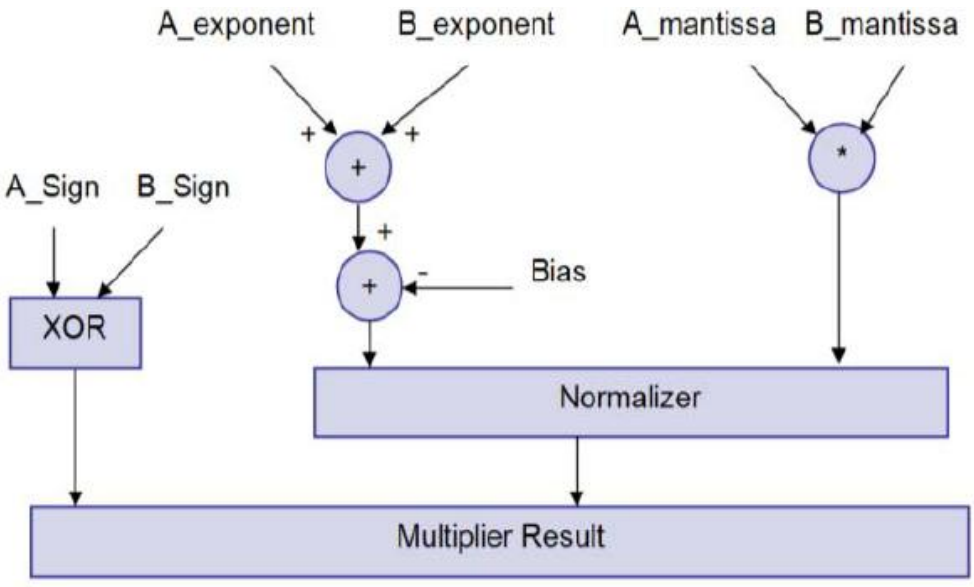

Fig. 4.8: Block diagram for binary floating point multiplication.

4.4.2.1 Calculation of sign bit:

The sign bit of the product is calculated by performing exclusive-or operation on the sign bit of two operands. If both numbers are positive or negative then the result is positive. But if the numbers are of opposite sign then result will be negative. The similar operation is performed by an ex-or gate. A zero sign bit represent positive number and a one sign bit represent negative number.

4.4.2.2 Calculation of the exponent:

Exponent is calculated by adding exponents of two operands and then subtracting the bias. Initially the operands are in biased notation, so the bias is added twice. Hence a bias has to be subtracted to get the correct result. Bias value is fixed *i.e.* 15 for half precision floating point multiplier. So we require two adders, one for adding the exponents and second for adding the intermediate result and 2's complement of the bias. Resultant exponent can be written as

$$E = E1+E2- bias$$

Fig. 4.9 shows the block diagram for exponent calculation.

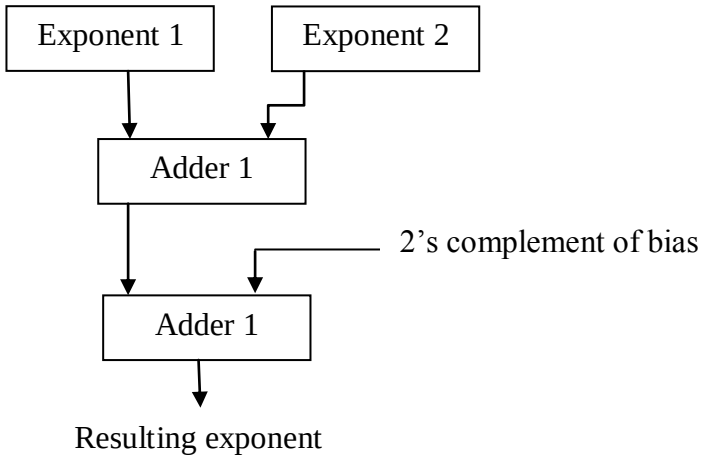

Fig. 4.9: Block diagram for exponent calculation.

4.4.2.3 Calculation of the significand (or mantissa):

Significand is calculated by unsigned multiplication of two numbers after appending '1' to mantissa of each number, because this 1 is hidden. The calculation of significand involves different steps. First partial products are generated, then partial products are passed through different compressors. At last stage carry propagate or

other fast adder is used to compute the final result. The steps for calculating significand are depicted in Fig. 4.10 and are also described below in detail.

1) Partial product generation:

Partial products can be generated by multiplying the multiplicand by a '0' or '1'. If a '0' is multiplied then the result is string of zeros so there is no need to add that partial product. But if a '1' is multiplied then the result is the multiplicand itself so that partial product must be added. Multiplication can be done by using AND gates because the AND gate performs the multiplication of the input bits applied to it. But it takes large amount of area. Hence for high performance systems this method of multiplication is not used. The operation of multiplication could be made fast if the generated partial products are reduced. Different algorithms are used for reducing the partial products. Partial products generated can be reduced by using booth's multiplier. In next section booth's multiplier is described.

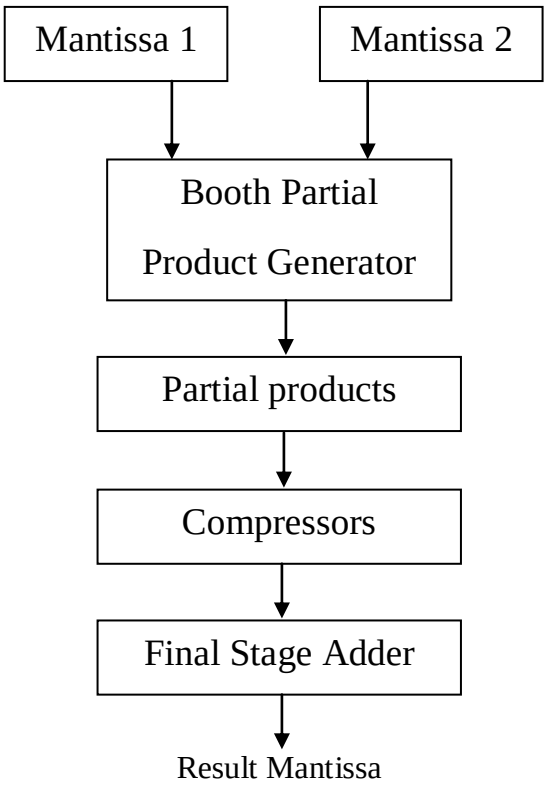

Fig. 4.10: Steps for calculating resulting mantissa while multiplication.

a) Booth's multiplier

Booth's multiplier uses booth's algorithm. The algorithm was proposed by A.D. Booth in 1951. Using this algorithm multiplication time can be reduced. The exact amount of time reduction depends on bit pattern of the multiplier. If the multiplier has a stream of 1's, the number of addition required is minimized. This concept is illustrated in the following example. Consider a multiplier having *m* consecutive is in it.

$$M * \text{“00111110”} = M * (2^5 + 2^4 + 2^3 + 2^2 + 2^1) = M * 62$$

If the multiplier is modified as below, only 2 partial products will be generated instead of *m*

…001 1 1 1 1 0…=...01000000…-...00000010…

$$M * \text{“010000-10”} = M * (2^6 - 2^1) = M * 62$$

First partial product will be added and the second will be subtracted because of the negative sign.

Booth's algorithm can be used with both signed and unsigned number. It uses 2's complement representation for handling the signed numbers. There are different versions of booth's algorithm. These are radix-2, radix-4, radix-8 *etc*. Higher is the radix, more fast is the multiplier. Thus Radix-8 booth's algorithm is chosen and is explained below.

b) Radix-8 booth algorithm:

In this algorithm first a zero is appended to the LSB of the multiplier. Then four adjacent bits of the multiplier starting from the LSB and appended zero are examined. Depending upon the four bits a specific operation is carried out which is shown in Table 4.2. In this manner 3 bits of the multiplier number is eliminated in each pass. Thus number of partial products generated are reduced which is responsible for speeding up the operation of multiplier.

Table 4.2: Booth-8 encoding table.

X_{i+2}	X_{i+1}	X_i	X_{i-1}	Partial products
0	0	0	0	0
0	0	0	1	$+Y$
0	0	1	0	$+Y$
0	0	1	1	$+2Y$
0	1	0	0	$+2Y$
0	1	0	1	$+3Y$
0	1	1	0	$+3Y$
0	1	1	1	$+4Y$
1	0	0	0	$-4Y$
1	0	0	1	$-3Y$
1	0	1	0	$-3Y$
1	0	1	1	$-2Y$
1	1	0	0	$-2Y$
1	1	0	1	$-Y$
1	1	1	0	$-Y$
1	1	1	1	0

After carrying out the operations according to the 4-bit pair value using the booth encoding table, partial product are obtained. These partial products are now added by shifting left the second one by [4-1=3] bits, third one by [2*(4-1) = 6] bits, fourth by [3*(4-1)] bits and so on. While adding them to equalize the number of bits in each partial product, they undergo sign extension as explained below.

2) Sign extension:

If the MSB of the group of four bits is '0' means the operation performed is multiplication by positive number or the four bit pair is "1111", then append 0's in the front of the partial product. Otherwise, append 1's in front of it to equalize the number of bits in all the partial products.

Now, these partial products are reduced or compressed to reduce time delay in adding them which is explained in next section.

3) Partial product reduction:

Partial products can be reduced by using compressors. Compressors are nothing but carry save adders that generates separate bit strings for sum and the carry. In carry save adder the carry bit is not propagated rather generated for each bit. This makes operation faster than ripple carry adder. Fig. 4.11 shows a 3:2 compressor. In 3:2 compressor sum and carry bit is calculated for each set of 3 input bits.

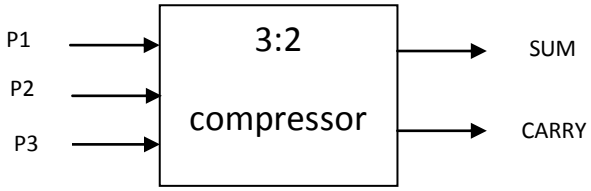

Fig. 4.11: A 3:2 compressor.

a) Carry Save Adder (CSA):

A carry-save adder is a digital adder that can be used in computer micro architecture to calculate the sum of three or more n-bit numbers in binary. It differs from other adders in a way that it produces two outputs of the same dimensions as the inputs, one output is a sequence of partial sum bits and another output is a sequence of carry bits. A carry save adder can be used as 3:2 compressors. An n-bit carry save adder can be designed by using n separate 1-bit full adders. This adder save the carry out bits instead of using it immediately to compute the final sum, that's why it is known as carry save adder. For a large number of input bit stream, carry-save adders are useful. Since the design automatically removes the delay in carry out bits. A 4-bit carry save adder to add four partial products of 4 –bit each is shown in the Fig. 4.12 *i.e* a 4:2 compressor. To accumulate the sum and carry streams obtained from carry save adder, a ripple carry adder is used at the end.

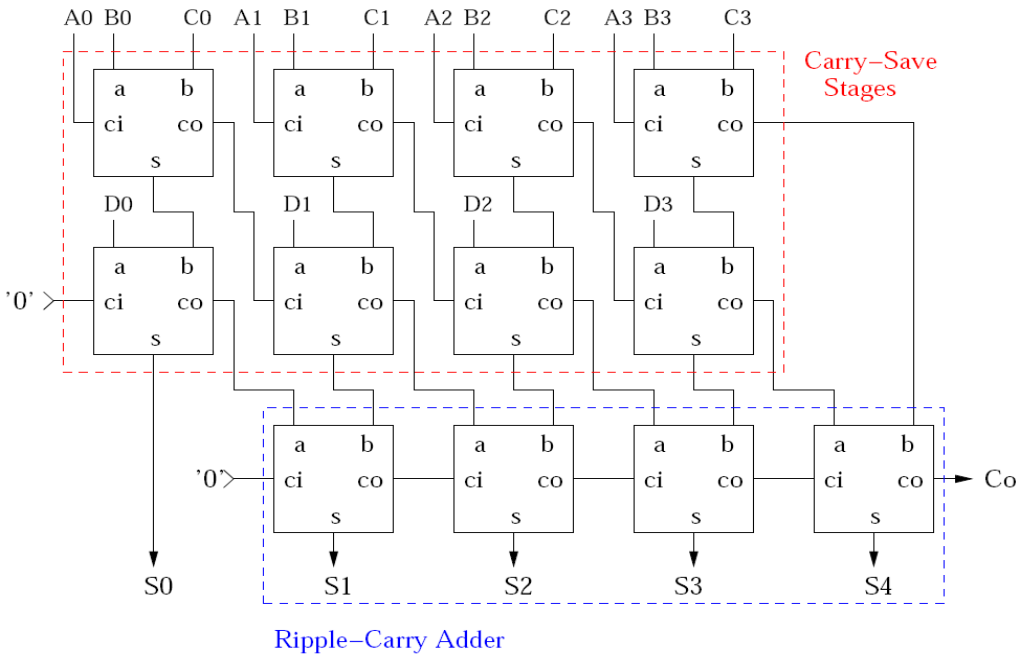

Fig. 4.12: A 4:2 compressor with two carry save stages and a ripple carry at the end.

CSA or compressors can be combined in different configurations to form different tree structures. Wallace tree is such a tree structure in which different inputs are applied to the different compressor and then their outputs are further given to other compressors as shown in Fig. 4.13. This process continues until the output only has 2 bit streams, one is final sum string and other is final carry string. This two bit-strings must be added by carry propagate adder or ripple carry adder.

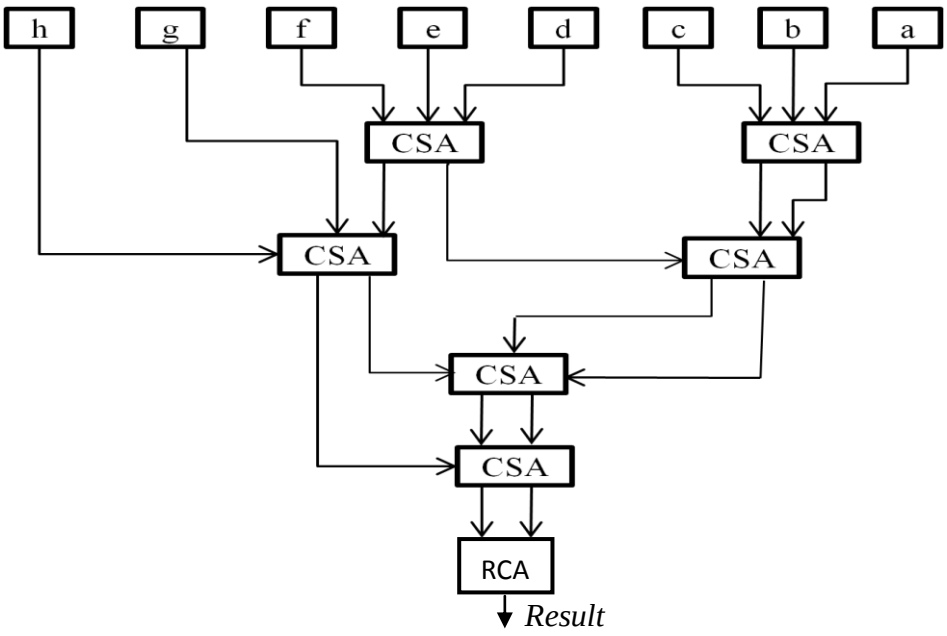

Fig. 4.13:Wallace tree structure using CSA with ripple carry adder at the end.

4) Normalizing the result after multiplication:

Normalizing means getting a '1' at MSB of the significand. As two significands with a '1' appended in front of each are multiplied, therefore, a 11-bit multiplier is needed which will give a 22-bit result from 0^{th} bit to 21^{st} bit. Now, '1' must be present at left of decimal point *i.e.* after 19^{th} bit of the significand. So a '1' is detected and exponent is adjusted accordingly. Two cases may arise. A '1' may be either at 20^{th} bit or at 21^{st} bit of final 22 bit result. If leading '1' is at 20^{th} bit the number is already a normalized number and no shift is needed. The resulting significand of the multiplication result will be the bits from 19 to 10. While if the leading '1' is at 21^{st} bit then decimal is shifted to the left and exponent is incremented by one. In this case, the resulting significand bits will be from 20 to 11. Hence, the correct result is obtained only after normalizing the number.

4.4.2.4 Example for multiplication of two numbers in binary 16 format

Consider two floating point numbers in their Binary16 format. Let say, they are 0.52 and 0.7272. They are represented in 16 bits as:

0.52 = 0 01110 0000101000

0.7272 = 0 01110 0111010001

To multiply two Binary16 format numbers, the sign bit of the result is obtained by Ex-ORing the sign bits of both numbers. So, '0' XOR '0'= '0'. Therefore, the resulting sign bit is '0'.

The exponent is calculated by adding the two biased numbers which gives "11100" and then adding the 2's complement of bias value (= "10001") to it which gives " 01101" with carry '1'. Since, the carry is '1', means the number is positive, therefore, neglecting the carry, the result obtained = "01101" is the final exponent of the multiplication result.

The significands are appended by '1' in front of them, thus they become 11- bit each.

The resulting significand is calculated by 11-bit multiplier using booth algorithm for multiplying two 11-bit significands. The 22- bit result obtained is :

01100000101100101010000

The 20th bit of this result is '1', therefore the number is already normalized. Hence, the resulting significand is the bits from 19 to 0 *i.e.*

1000001011

Therefore, the final result after multiplying two binary 16 format numbers is:

0 01101 1000001011

4.4.3 Floating point adder

To add two floating point numbers represented in Binary16 format, example to add 0.378 and 0.00154.

0.378 = 0 01101 1000001011

0.00154 = 0 00101 1001001001

First of all, check the difference in the exponents. For this example, the difference in exponents is 8. Append the mantissas of both the numbers with '1' in front. Then the mantissa of the number having smaller exponent is shifted right by the difference, so, that the exponents of both the numbers are now same each equal to the greater exponent. Here, in this example, 0.00154 has smaller exponent, thus its mantissa is shifted right by 8, and it becomes:

0.00154 = 0 01101 0.0000000110 and

0.378 = 0 01101 1.1000001011

The mantissas of both are of 11-bit each. Now, check the signs of both the numbers, accordingly there will be two different cases:

Case1: If both the numbers have same sign

Final sign will be same as the sign of both numbers. Now, simply add the 11-bit mantissas of the two numbers to get a 11- bit addition result and a carry bit.

1) If the carry out is '1'. Means, we have to normalize the result. Therefore, the final mantissa will simply be the higher 10 bits excluding the carry bit and final exponent will be one increment in the greater exponent.
2) If carry out is '0'. The final mantissa will be the lower 10 bits excluding the carry bit and the next bit and final exponent will be equal to the greater exponent.

Case 2: If one number is positive and other is negative

First of all, take the 2's compliment of the mantissa of the number whose sign bit is '1'. Then add the mantissas together to get a 12-bit result in whose 12^{th} bit is carry and the rest 11 bits is the sum.

1) If carry is '1', means the result is positive. Therefore, the final sign will be '0'.

- If the MSB of the sum *i.e.* 11^{th} bit is '1', final mantissa will be the lower 10 bits of the sum excluding MSB of sum and final exponent will be the greater exponent.
- If the MSB of the sum *i.e.* 11^{th} bit is '0',*i.e.* we have to normalize the number. Count and shift left the entire bits of sum until the MSB bit of sum is '1'. Lets say, it is shifted k times. Therefore, the final mantissa will be the lower 10 bits of the sum excluding MSB after shifting and the final exponent will be the greater exponent minus k.

2) If carry is '0', means the result is negative. Therefore, the final sign will be '1'. And, take the 2's compliment of the mantissa sum of 11 bits excluding carry. After taking 2's compliment,

- If the MSB of the sum is '1', final mantissa will be the lower 10 bits of the sum excluding MSB of sum and final exponent will be the greater exponent.
- If the MSB of the sum *i.e.* 11^{th} bit is '0',*i.e.* we have to normalize the number. Count and shift left the entire bits of sum until the MSB bit of sum is '1'. Lets say, it is shifted k times. Therefore, the final mantissa will be the lower 10 bits of the sum excluding MSB after shifting and the final exponent will be the greater exponent minus k.

Now, for the example taken earlier, both the numbers are having the same sign, thus case1 is valid for this. Accordingly,

Final sign will be same as of both *i.e.* '0'.

Adding the 11 bit mantissas, the 12-bit sum is :

01.1000010001 having the 12^{th} bit *i.e.* carry is '0'. Therefore, the final mantissa is the lower ten bits of the sum. *i.e.*

1000010001

And, the final exponent is the greater exponent = 01101.

Thus, the Binary16 representation of the final addition result of the two numbers is given by:

0 01101 1000010001

This was all about the floating point adder and the floating point multiplier. All these are implemented in verilog . Using these, the whole calibration algorithm is modelled in verilog for one- stage in order to get the $D_{tot} = D_{cal}$ and the error = 0. Likewise, further it can be modelled for all the stages and the final D_{tot} can be calculated as the result of 12- bit pipelined ADC. The whole project can then be verified by dumping it on FPGA.

Chapter 5

FPGA IMPLEMENTATION & RESULTS

5.1 Introduction to FPGAs

FPGAs are semiconductor devices which has extremely useful property of field programming. Field programming means user or designer can reprogram it after its manufacturing. FPGA contains configurable logic blocks, input/output blocks and programmable interconnections. They also contain storing elements like flip flops or blocks of memory. The FPGA implementation of designs are done to check their functionality on actual hardware. The cost of implementation and design cycle time of ASICs are large therefore the bigger designs are first checked on FPGA and if they give satisfactory results then there ASIC implementation is done. Fig. 5.1 shows the different steps involved in the FPGA implementation. It includes design entry through HDL, behaviour simulation, logic synthesis, design implementation, bit stream generation and finally programming the FPGA. Xilinx provides all the functions necessary for implementing a design on FPGA. The Xilinx ISE suite includes ISIM simulator for functional and timing simulation, XST for synthesis applications, plan-ahead tool for floor planning, Xpower analyzer for estimating power, iMPACT for directly configuring FPGA device and chipscope pro analyser for debug and verification purposes.

5.2 FPGA implementation using XILINX

The FPGA that is used for the implementation of the circuit is the Xilinx Spartan 6 (Family), XC6SLX45 (Device). The working environment/tool for the design is Xilinx ISE 14.2 for FPGA Design flow of Verilog code. The different steps involved in FPGA implementation using xilinx are described below.

1) Design Entry:

This is the first step in FPGA implementation. In design entry source files are created for representing the given design. The source file can be HDL file such as Verilog or VHDL, schematic file, embedded processor file, or EDIF file.

2) Behavioral simulation:

Behavioral simulation is performed for checking the design functionality. Simulation can be performed on HDL source files, HDL test benches, waveform files, Simulation-only HDL source files. To perform the simulation xilinx either uses ISIM or modelsim simulator. Simulation can also be done after synthesizing the design. The simulation after place and route is called timing simulation.

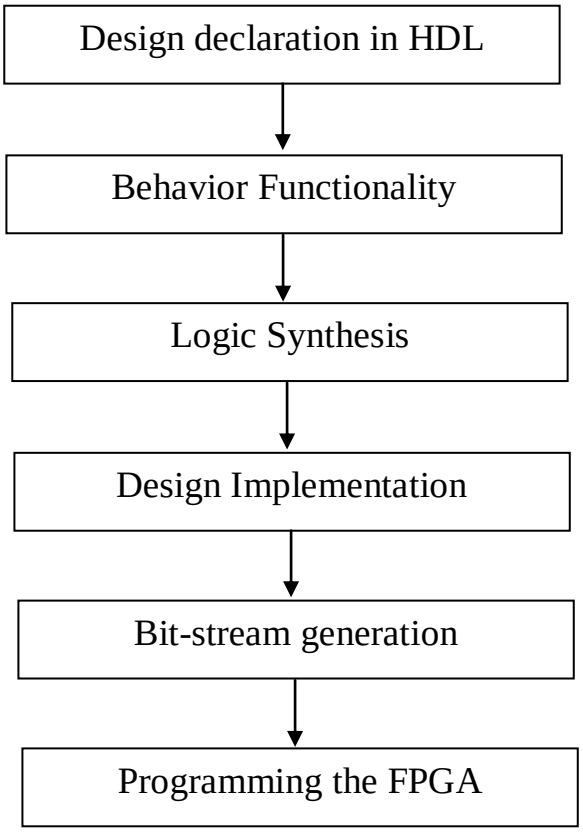

Fig. 5.1: FPGA design flow.

3) Design synthesis:

Xilinx uses XST (xilinx synthesis technology) synthesizer for synthesizing VHDL and Verilog codes. It generates netlist file for given design. This file contains logic design data and constraints. The netlist file has an extension .NGC and serves as input for the translate process. This file is the result of three steps *i.e.* HDL parsing, HDL synthesis and low level optimization. In HDL parsing, code is checked for syntax errors. In HDL synthesis part code is analysed and inferred to basic macros like RAM, multiplexers, adders *etc*. This is done for efficient implementation of technology. In HDL synthesis part FSM recognition is also done. Synthesizer chooses one of the several FSM encoding algorithms according to the optimization goal for efficient implementation. In low level optimization the macros are transferred into technology specific components such as carry logic, shift registers, clock buffers *etc*. This step also contains timing optimization, technology mapping and register replication. So the NGC file is generated after HDL parsing, HDL synthesis and low level optimization. Fig. 5.2 shows the complete process of synthesis from HDL code to NGC file generation.

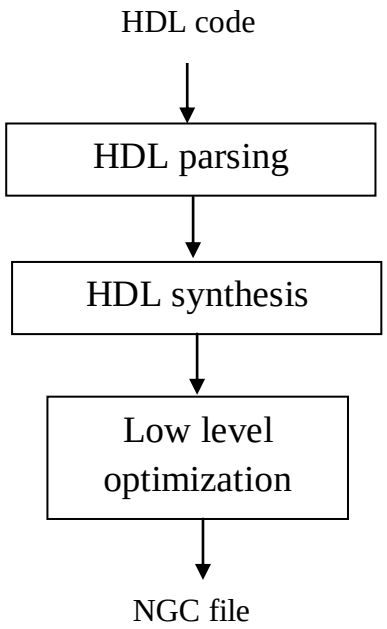

Fig. 5.2: Steps in synthesis process.

Besides generating NGC file XST generates RTL schematic, technology schematic and synthesis report.

- **RTL schematic:** This is the register transfer level representation of the pre optimized design. In this representation basic building blocks like adders, AND gate, OR gate *etc*. are used. This view helps in discovering issues in starting phase of design because the design is still not mapped on the target device.
- **Technology schematic:** This is the representation of NGC file in terms of logic elements of the targeted device like LUTs, I/O buffers *etc*. The technology schematic is generated after optimization of the design.
- **Synthesis report:** This report is the result of entire synthesis run. It contains area, delay and timing estimations. This report consists of several other reports like HDL synthesis report, advanced synthesis report, Device utilization summary, Partition resource summary, timing report *etc.*

5.3 Implementing the design

Using implementation process the design is transferred onto the desired device. Implementation of design is done in three different steps. These steps are translate, map and route. The output NGC file from synthesis process is taken as input for the design implementation process. After translate, map and place and routing operation routed NCD file is generated which is used to generate bit file. The generated bit file can be burned on the FPGA. Different files are generated in this whole process. Fig. 5.3 shows different intermediate files generated in the process of implementation.

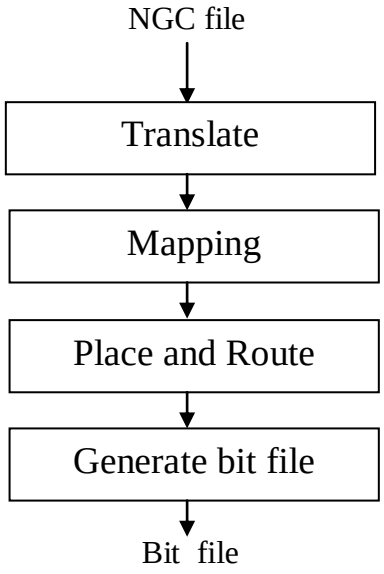

Fig. 5.3: Different files generated in implementation process.

Steps involved in the implementation process are written below:

- **Translate:** In this process all the input netlists and design constraints are combined and saved in a file called as native generic database file. The ports available in design are assigned to physical elements of the target device. Timing requirements of the design are also specified in translate process. Translate properties can also be changed by modifying them.
- **Mapping:** After translate process mapping is done. In mapping the circuit is divided into sub-blocks. The sub-blocks are made so that they can fit into FPGA sub-blocks. A file is generated called as native circuit description file. This file contains our design mapped into components of FPGA.
- **Place and route:** sub-blocks of map process are converted into logic blocks and connected in place and route step. This process takes NCD file as input and outputs the routed NCD file. Here placement and routing of blocks is done.
- **Generate Programming file:** In this process bit file is generated for particular xilinx device from the routed NCD file. The output bit file contains binary bits necessary to program the device. Sometimes this process is also called as bit-stream generation. The generated bit file is used to program the FPGA device.

5.4 Analysing design using chipscope pro

The FPGA designs are becoming more complex, due to need of faster designs and shorter design times. Debugging and verification is important factor in determining the complete design time. It takes almost 50% of the design time. But the xilinx chipscope pro software performs faster debugging and verification. It shrinks overall design by 25%. It is a powerful tool that is easy to use. It is used for debug, verification and inserting short signal sequences. Chipscope pro uses FPGA resources like block RAM for trigger and data storage, slice logic for trigger comparison. It uses three types of flows *i.e.* core generator, core inserter and planahead flow. The core inserter flow is similar to plan-ahead flow and provided in plan-ahead software. In core generator flow the core is instantiated in source HDL, while in core inserter flow the core is inserted into generated file after synthesis. Different types of cores are used

by chipscope software like ICON core, ILA core, VIO core, IBA core. Some of them are explained below:

- **VIO core:** It defines and generates virtual I/Os. It is used to apply stimulus and read outputs transition on the node we want to select. This core is used to generate virtual input and outputs. It provides options for synchronous and asynchronous inputs and outputs, where each can have width of 256 bits. There is also option of clock. It can be system clock or JTAG clock The outputs of the design which we want to implement are connected to the input of the VIO core and inputs of the design are connected to the output of the VIO core therefore the inputs are virtual LEDs and outputs are virtual DIP Switches. This core is controlled by the ICON core. So a control port is also provided. VIO core uses FPGA logic not RAM. It is only used in core generator flow.

- **ICON core:** It is used to control up to 15 capture cores. It acts as interface between JTAG interface and capture cores. The main function of this core is to control the other cores. It can be used in both the core generator and core inserter flows.

- **ILA core**: It is a capture core. It can be used to create custom triggers when activated causes data to be stored during circuit operations. Signals can be stored depending on the condition specified by used. A design can contain upto 15 ILA cores.

- **Agilent trace core:** It used to store large amount of data off chip or when customer uses agilent analyzer. ILA with agilent trace is similar to ILA except data is captured off chip or by agilent trace port analyzer.

5.5 Implementing the LMS algorithm and Results.

5.5.1 Implementation in MATLAB

The methodology used is, firstly, the ideal pipelined ADC and non-ideal pipelined ADC is modelled in MATLAB keeping V_{REF} = 5V and sampled analog ramp input in the range –5V to 5V. The step size of the 12-bit ADC is thus = $[5 – (–5) / 2^{12}] =$

$0.00244\,V$. Digital output from all the twelve stages is obtained by modelling the redundancy removal algorithm using cascade of adders. It is analysed that there is a huge variation in the transfer curve of the ADC in non-ideal case from the ideal one. This can also be understood in terms of DNL and INL before and after calibration.

The DNL before calibration is calculated at a sample rate of 500Hz and the results obtained are shown in Fig. 5.4. It has a maximum value of 1.4571 = 597.172 LSB.

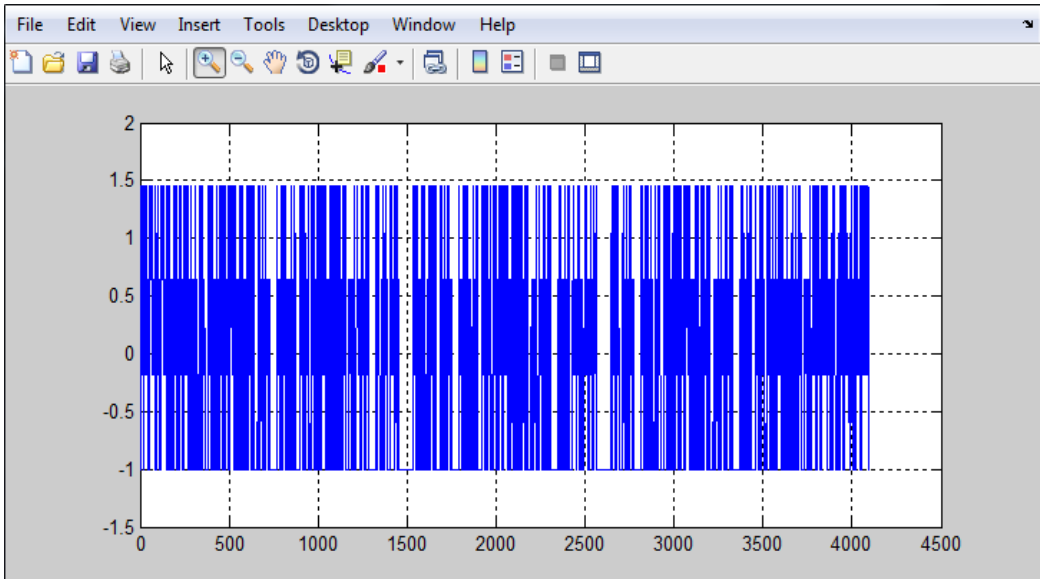

Fig. 5.4: DNL before calibration.

Fig. 5.5: INL before calibration.

Similarly, the INL at the same sample rate before calibration is shown in Fig. 5.5. The maximum value of INL before calibration is 55923.65 LSB.

Fig. 5.6 and 5.7 shows DNL and INL after calibration.

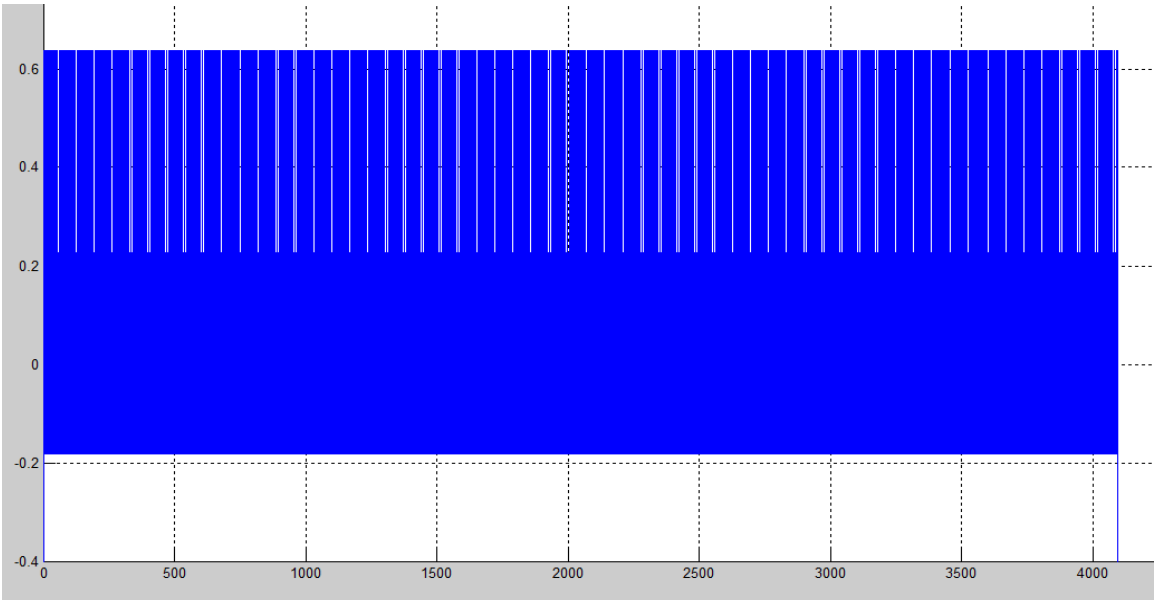

Fig. 5.6: DNL after calibration.

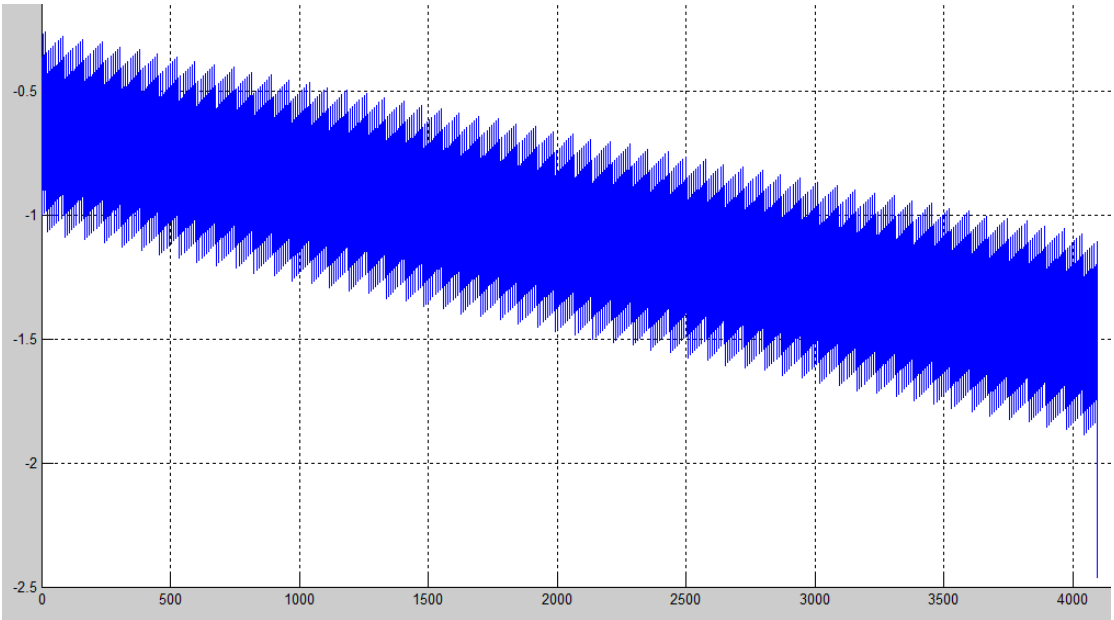

Fig. 5.7: INL after calibration.

The graphs in Fig. 5.6 and 5.7 depicts that DNL reduced to 0.638 = 261.475 LSB and INL reduced to 1010.779 LSB.

5.5.2 Implementation in Xilinx using verilog

The tool used is XILINX ISE 14.5 and the working environment of all the designs is :

- Family : Spartan 6E
- Device : XC6SLX45
- Speed : -3
- Package: CSG324
- Simulator : ISIM
- Total slices: 27288
- Total LUTs: 54576
- Optimization Goal : Speed

The calibration algorithm is modelled in verilog for one-stage using the floating point multipliers and adders as discussed in this thesis report earlier. The code is of about 5000 lines. Take any initial value of α_1 and α_3 and w_j say, $\alpha_1 = 0.52$ and $\alpha_3 = 0.004$. and w_j is calculated to be equal to 0.45. The D_{BK} , D_{cal} and DAC output required as inputs for the calibration algo for this stage are imported from MATLAB modelling. After successful operation of the algorithm, the output $D_{cal} - D_{tot}$ *i.e.* error is obtained on ISIM simulator is equal to zero as shown in Fig. 5.8. The test inputs taken are:

D_{cal} = 0.4 = 0 01101 1001100110

α_1 = 0.52 = 0 01110 0000101000

α_3 = 0.004 = 0 00111 0000011000

D_{BK} = 0.7272 = 0 01110 0111010001

w_j = 0.45 = 0 01101 1100110011

dd (DAC output) = 0 00000 0000000000

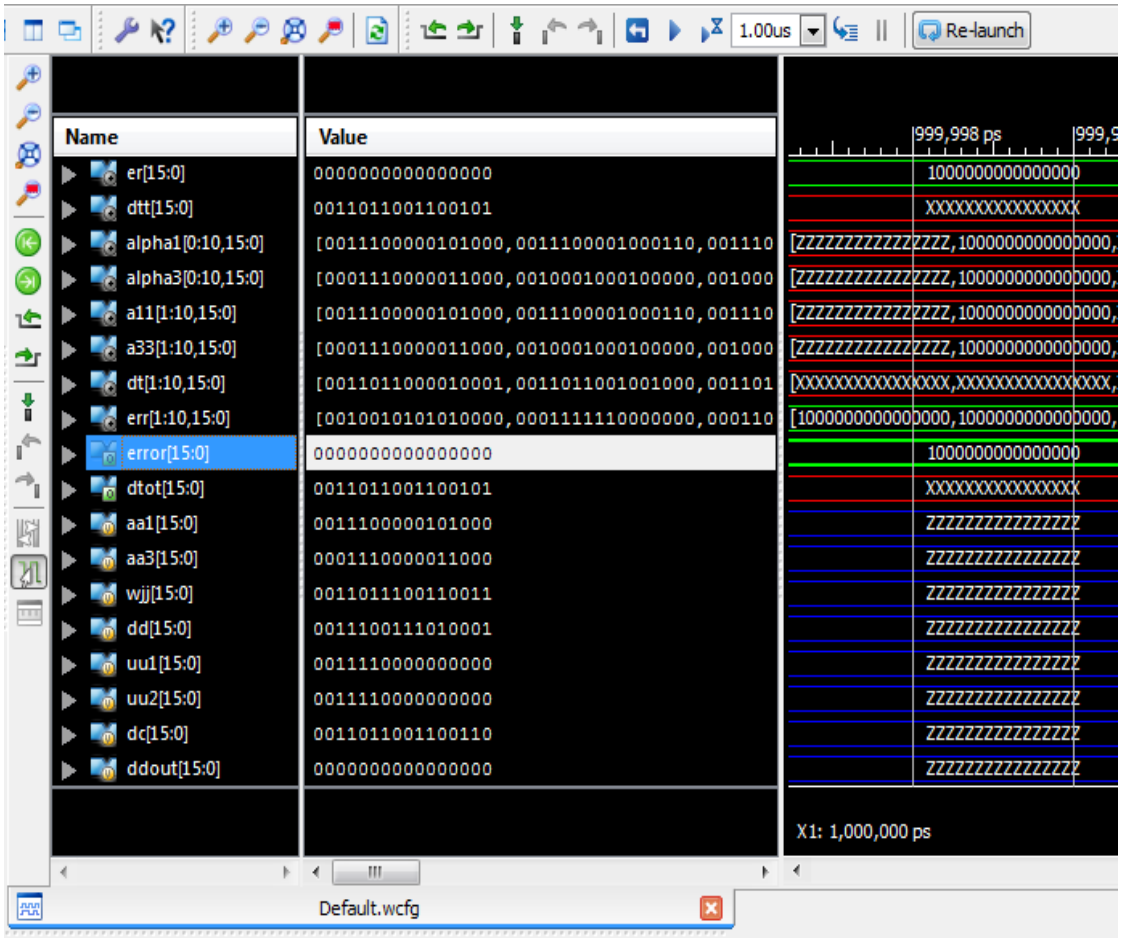

Fig. 5.8: simulated output of calibration algorithm for one-stage.

The calibration is run off-chip but a detailed gate level synthesis of the calibration logic is performed to estimate the area required and the associated power dissipation. Fig. 5.10 shows the block diagram of the synthesized system.

After synthesis on Spartan 6, the results obtained are shown on Fig. 5.9.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	1155	54576	2%
Number of Slice LUTs	60123	27288	220%
Number of fully used LUT-FF pairs	1155	60123	1%
Number of bonded IOBs	160	218	73%

Fig. 5.9 Device utilization results for one-stage calibration circuit.

The device utilization summary depicts that the area utilization is very high, this is the limitation . The maximum combinational path delay for calibration circuit for one-stage comes out to be 1656.921 ns.

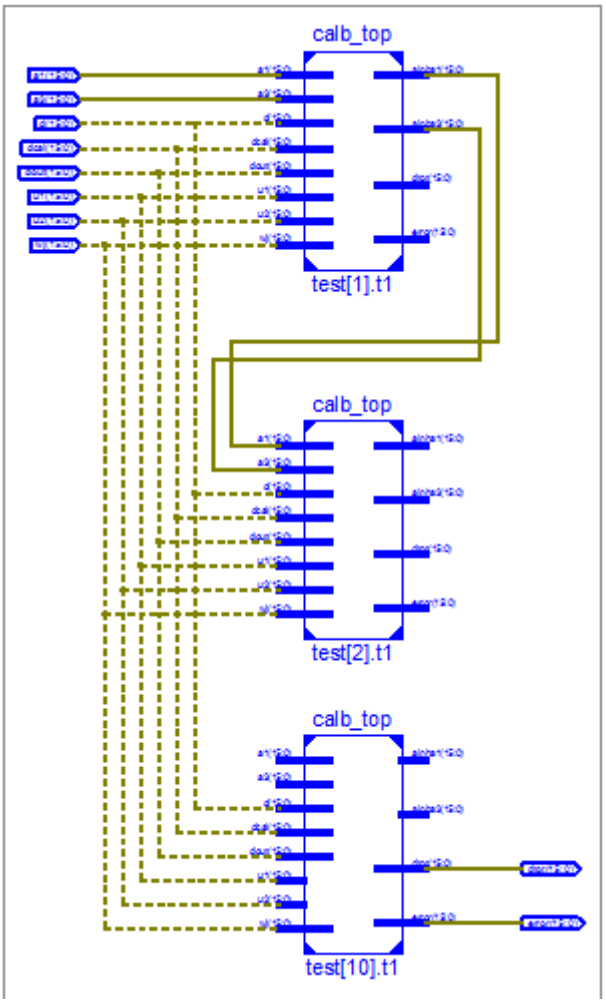

Fig. 5.10: Block diagram of loop iterations of the calibration circuit for one-stage.

The internal detailed block diagram of module calb_top in Fig. 5.10 is shown in Fig. 5.12.

The booth multiplier used in making Binary16 floating point multiplier is of 11-bit and after synthesization it was analysed that the maximum combinational path delay is 18.871 ns and power, calculated by X-power analyser, comes out to be 0.036 W. The device utilization summary for 11-bit booth multiplier is shown in Fig. 5.11.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice LUTs	434	27288	1%
Number of fully used LUT-FF pairs	0	434	0%
Number of bonded IOBs	44	218	20%

Fig. 5.11: Device utilization Summary for 11-bit booth multiplier.

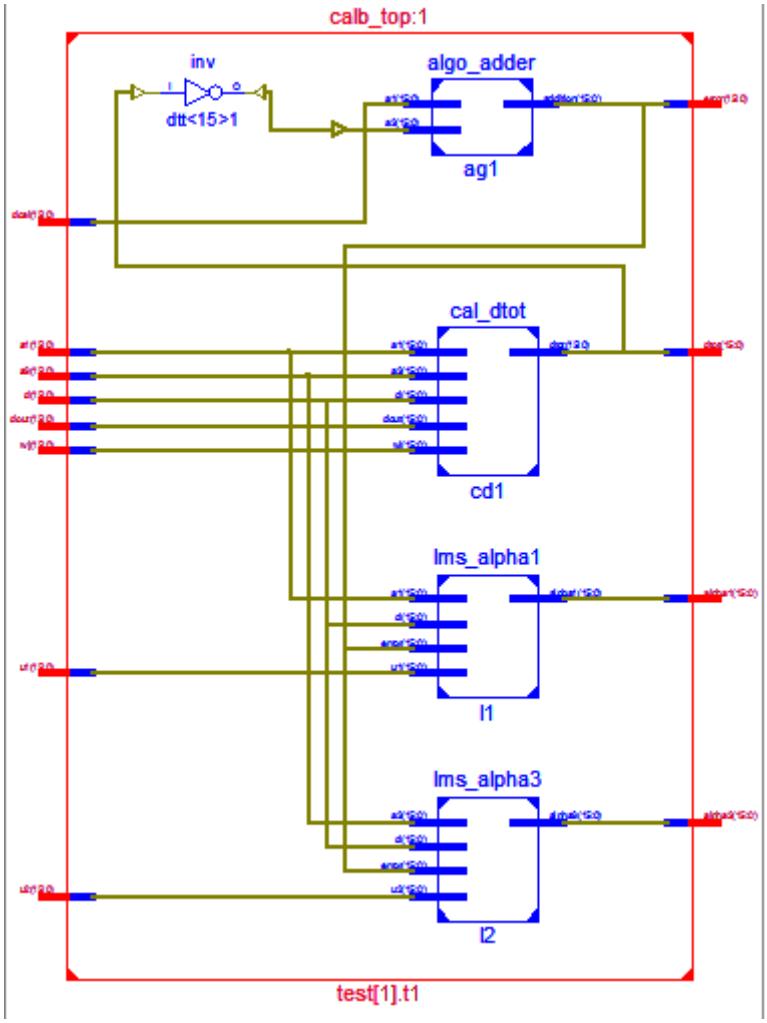

Fig. 5.12: Internal Block diagram of each loop iteration of calibration circuit for one-stage.

Chapter 6

CONCLUSION & FUTURE SCOPE

6.1 Conclusion

As the CMOS technology is decreasing day by day and analog circuits does not perform so well at lower technology, therefore, there is a great need to calibrate all the errors instead of calibrating for only residue gain error. Such an algorithm is successfully implemented in verilog and MATLAB using LMS algorithm to remove residue gain error, DAC gain error and non-linearity of the op-amp. The calibration hardware has also been successfully built for 12- bit pipelined ADC. It has been analysed that DNL reduced from 597.172 LSB to 261.475 LSB and INL reduced from 55923.65 LSB to 1010.779 LSB.

6.2 Future Scope

- Due to limitation of hardware while implementing it on FPGA, the maximum iterations possible in the implementation are only 10. So, for the analog input values for which, less than or equal to 10 iterations of LMS are required, the pipelined ADC stages are successful calibrated. So, much work could be done in this respect to analyse the statistical data so that □$_1$ and □$_2$ used to update α_1 and α_3 could be made variable with the input such that minimum number of iterations are required.

- More efficient adders can be used to accumulate the partial products generated while multiplication of floating point numbers in order to reduce area and delay so as to make the calibration hardware compact.

- Lesser number of bits can be used to represent the floating point numbers in binary so as to reduce the area.

REFERENCES

[1] John P. Keane, Paul J. Hurst, and Stephen H. Lewis, "Background Interstage Gain Calibration Technique for Pipelined ADCs", *IEEE Transactions on Circuits and Systems-I,* Vol.52, No.1, Jan.2005.

[2] Ashutosh Verma and Behzad Razavi, "A 10-bit 500MS/s 55-mW CMOS ADC", *IEEE Journal of Solid State Circuits,* Vol.44,No.11, Nov. 2009.

[3] Meysam Mohammadi, Amir Alipour, Esmaeil N. Aghdam and Khosrov D. Sadeghipour, "A New Technique for Background Calibration of Pipelined ADCs", *Electrical Engineering (ICEE), 2013 21st Iranian Conference on*. IEEE, 2013.

[4] K.L.S Swee and L.H Hiung, "Performance Comparison Review of Radix-Based Multiplier Designs," in *4th International Conference on Intelligent and Advanced Systems(ICIAS)* , pp. 854-859, 12-14, Jun.2012.

[5] C. Nagenndra, M.J. Irwin and R.M. Owens, "Area-Time-Power Tradeoffs in Parallel Adders," in *IEEE Transactions on Circuits and Systems-II*, Vol.43, No.10, Oct.1996.

[6] J. Kaur, N.K. Gaahlan and P.shukla, "Delay-Power Performance Comparison of Array Multiplier in VLSI Design," in *International Journal of Advanced Research in Computer Science and Electronics Engineering*, Vol.1, No.3, May.2012.

[7] P. Gurjar, R. Solanki, P. Kansliwal and M. Vucha,"VLSI Implementation of Adders for High Speed ALU", *Proc. India Conference(INDICON).*pp.1-6,16-18,2011.

[8] A. Jain, B. Dash and A. Panda, "FPGA Design of a Fast 32-bit Floating Point Multiplier Unit," in Proc. *International Conference on Devices,*

Circuits and systems(ICDCS), pp.545-547,15-16 Mar. 2012.

[9] G. Renxi,Z. Shangjun, Z. Hainman and M. xiaobi, "Hardware Implementation of a High Speed Floating Point Multiplier Based on FPGA," in Proc. *International Conference on Computer Science & Education(ICCSE)*, pp.1902-1906, 25-28 , July.2009.

[10] Behzad Razavi, "Design of Analog CMOS Integrated Circuits", *Tata McGraw-Hill edition*, 2002.

[11] IEEE 754-2008, "IEEE Standard for Floating-Point Arithmetic," 2008.

[12] S. Shafiulla Basha, Syed. Jahangir Badashah, "Design and Implementation of Radix-4 Based High Speed Multiplier for ALU's Using Minimal Partial Products", *International Journal of Advances in Engineering & Technology, (IJAET)* ,Vol. 4, Issue 1, pp. 314-325, July. 2012.

[13] P.K. Patil and K. laxmi," High Speed-Low Power Radix-8 Booth Decoded Multiplier", *Proc. International Journal of Computer Applications*, Vol.73, No.14, 2013.

[14] K. Babulu, G.Parasuram , "FPGA Realization of Radix-4 Booth Multiplication Algorithm for High Speed Arithmetic Logics", *(IJCSIT) International Journal of Computer Science and Information Technologies*, Vol. 2 (5) , 2011, 2102-2107

[15] R.P.P. Singh, P. Kumar and B. Singh, "Performance Analysis Of Fast Adders Using VHDL", *Proc. International conference on Advances in Recent Technologies in Communication and Computing*.pp.189-193,27-28,2009.

[16] Sounak Roy, Bibhudatta sahoo and Swapna Banerjee, "Radix Based Digital Calibration Technique for Pipelined ADC Using Nyquist Sampling of

Sinusoid", *IEEE International Symposium on Circuits and Systems (ISCAS),* IEEE, 2012.

[17] ANSI/IEEE 754-1985, "Standard for Binary Floating-Point Arithmetic," 1985.

[18] M. Morris Mano, "Computer System Architecture," *Pearson Education India, 3rd edition.*

[19] Paladugu Srinivas Teja, "Design of Radix-8 Booth Multiplier Using Koggestone Adder for High Speed Arithmetic Applications", *Emerging Trends in Electrical, Electronics & Instrumentation Engineering: An international Journal* (EEIEJ), Vol. 1, No. 1, Feb. 2014.

[20] Carlos Azeredo Leme and Navraj Nandra, article "Analog Circuits Benefit from Scaling Trends", Synopsys.

[21] Bei Peng, Guanzhong Huang, Hao Li, Peiyuan Wan and Pingfen Lin,"A 48-mW, 12-bit, 150-MS/s Pipelined ADC with Digital Calibration in 65nm CMOS", *Custom Integrated Circuits Conference (CICC),* IEEE, 2011.

[22] Maxim Integrated, "Understanding pipelined ADCs", tutorial 1023.

[23] Andrew N. Karanicolas, Hae-Seung Lee and Kantilal L. Bacrania, "A 15-b 1Msample/s Digitally Self-calibrated Pipeline ADC", *IEEE Journal of Solid-State Circuits,* Vol.28, No.12, pp. 1207-1215, Dec.1993.

[24] Jipeng Li, and Un-Ku Moon, "Background Calibration Techniques for Multistage Pipelined ADCs with Digital Redundancy", *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing,* Vol.50, No.9, Sept. 2003.