A 16 BIT 500KSPS LOW POWER SUCCESSIVE APPROXIMATION ANALOG TO DIGITAL CONVERTER

By

KUN YANG

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

December 2009

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of KUN YANG find it satisfactory and recommend that it be accepted.

_____

George S. La Rue, Ph.D., Chair

_____

John Ringo, Ph.D.

_____

Deuk Heo, Ph.D.

ACKNOWLEDGEMENT

First of all, I would like to thank Professor George La Rue for his guidance and support to my master's study at Washington State University.

Secondly, I want to thank Professor David Rector of the Department of Veterinary Science for funding my research project.

I would also like to thank Dr. Dirk Robinson and Lanchuan Zhou, for their advice and help to my research work.

Many thanks to Ding Ma, Hari Krishnan, Saurabh Mandhanya, Bill Hamon, Wei Zheng, Prasanna Upadhyaya, Xiangjun Fan, Peter Osheroff, Lowell Hanson and all my Chinese friends in Pullman, who have made my life here so much more fun.

Finally, I want to thank my parents, Qide Yang and Huixian Zhou, for bringing me up and providing me with their love and support.

A 16 BIT 500KSPS LOW POWER SUCCESSIVE APPROXIMATION

ANALOG TO DIGITAL CONVERTER

ABSTRACT

By Kun Yang, M.S.
Washington State University
December 2009

Chair: George S. La Rue

A 16 Bit, 500KSps low power successive approximation analog-to-digital converter (ADC) is designed in JAZZ 0.18um SiGe CMOS process using only CMOS devices. The speed of 500KSps is achieved by interleaving between two capacitor arrays with shared input buffer. Power efficient comparators with shutdown mode are designed to reduce the power consumption of the ADC. Capacitance ratio error, offset error and gain error of the ADC are calibrated using self-calibration circuits. Rail-to-rail input buffer is designed to drive the ADC. The ADC works under ± 1V power supplies, ± 0.8V references and a 10MHz clock. The total power dissipation is 4.59mW including the input buffer and the noise is 25.1uV with the input buffer and 16.9uV without. The SNDR, ENOB of the ADC are 87.4dB, 14.22 bits. The FOM of the ADC with and without the input buffer is 240.5fJ/Conv. Step and 44.0fJ/Conv. Step respectively.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Background

Signals in the real world are by nature analog. In other words, it changes continuously with respect to time. However, modern signal processing is done primarily in the digital domain, because digital signals have the advantages of being able to perform complex calculations easier to test, less sensitive to noise, and so on. Interface circuits to convert signals between the analog and digital domains are important. At the electronic circuit level, the conversion is realized with analog-to-digital converters (ADCs) and digital-to-analog converters (DACs). The former measures an analog signal and convert it into digital data, while the latter outputs an analog signal based on its digital input.

Studying and measuring neural activity is a growing research field of animal science. This project is directed toward this application. Researchers implant electrodes into an animal's brain to sense the small neural signals. Then, the analog signals are amplified using an amplifier integrated circuit (IC). The amplified signals are then converted into digital signals using an ADC for recording and processing. The noise of the amplifier and ADC need to be low for accurate measurements. Some experiments that measure neural activity of small animals, such as mice, while they are free roaming, have further requirements on the amplifier IC and the ADC to have low power and small volume since they must be carried by animals and along with a power source, such as a battery or some other power sources. The conversion rate of the ADC falls into the moderate range in order to sufficiently monitor the highest frequencies of the brain activity of these animals.

To meet all the requirements for this application, a 16 BIT, 500KSps successive approximation register (SAR) ADC is designed and presented is this thesis. Successive Approximation ADCs are one of the most popular approaches for realizing ADCs due to their reasonably quick conversion time, yet moderate circuit complexity [1]. However, there are some design choices for the SAR ADC that must be addressed to obtain low power and high accuracy. First of all, to obtain low power at a sampling rate of 500KSps, two capacitor arrays are used, each with a conversion rate of 250KSps [2]. The input buffer charges the first array while the voltage on the second array is determined using the comparator. Then the input buffer charges the second array and the first array is converted. Thus only one input buffer is needed. The ADC conversion errors caused by process variation, device mismatch, comparator offset and parasitic effects need to be overcome in order to achieve an accuracy of 16 bits[3][4]. Thus Capacitor Ratio, Offset, Parasitic Capacitance calibration circuits are added into the design. Also, a dual comparator approach is utilized to reduce the power consumption of the ADC [4]. For the 8 MSB conversions, a low power coarse comparator is used while for the 8 LSB conversions, a high power fine comparator is used. Whenever one comparator is used, the other one is shut down, avoiding unnecessary power dissipation. In addition, the design of the input buffer for the ADC is a challenge because it needs to be low power, low noise (lower than 1/4 of the resolution of the ADC) and fast enough to charge the large capacitor arrays of the ADC. In addition, it should be able to provide large output swing - no less than the input range of the ADC. A high speed operation-amplifier (op-amp) with Rail to Rail input stage and Class AB output stage is designed to serve as the input buffer.

The ADC is designed in JAZZ 0.18um SiGe CMOS process using only the CMOS devices. The power supply is ± 1V and the clock frequency is 10MHz. Simulation results show that the ADC total power consumption is 4.59mW and the total noise is 25.1uV. The signal-to-noise-and-distortion-ratio (SNDR), effective number of bits (ENOB) of the ADC are 87.4dB and 14.22 respectively.) The figure of merit (FOM) of the ADC is 240.5fJ/Conv. Step including input buffer and 44.0 fJ/Conv. Step without it.

The input buffer is the main source of noise and power dissipation of the ADC, and is a topic for future research.

1.2 Organization

This thesis is organized into 6 chapters. Chapter 2 gives an overview of the design, with a description of the ADC architecture and basic conversion process. Design challenges are also discussed in this chapter. Chapter 3 elaborates on the design of the building blocks of the ADC, with emphasis on the design techniques used to achieve the desired performance. Chapter 4 gives a detailed discussion of the calibration algorithms implemented in this design to achieve 16 bit accuracy. Chapter 5 presents some of the simulation results and chapter 6 is the conclusion.

2. SAR ANALOG TO DIGITAL CONVERTER

2.1 Architecture

Successive Approximation ADCs are popular for many applications. They have the advantages of low power, high resolution, small size and moderate speed. Basically, SAR ADC converts an analog input voltage into a digital number with a binary search algorithm. The resolution of the ADC, the least-significant-bit (LSB), is determined by dividing the voltage reference by the number-of-bits (NOB) of the ADC. Meanwhile, due to its higher noise rejection ratio, a fully differential design is adopted in this thesis. For simplification the architecture of the fully differential ADC with one capacitor array and no interleaving will be discussed first. This architecture is shown in Fig.2.1.

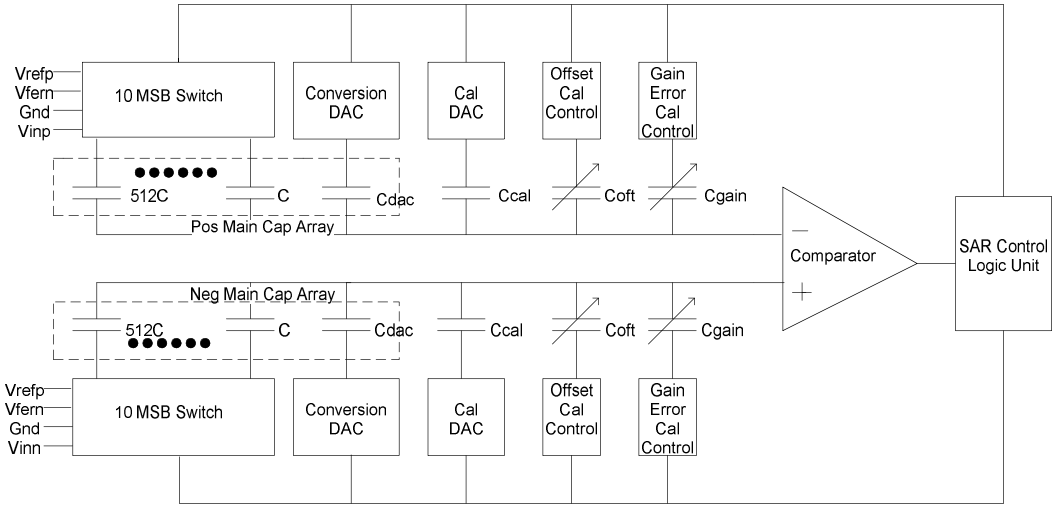


Figure 2.1 Architecture of the SAR ADC

As shown in the figure, the ADC includes of two main capacitor arrays, two DACs with one for conversion and the other for calibration, offset and gain error capacitor arrays, comparators and SAR Control Logic Unit. The two main capacitor arrays are made of 11 capacitors, with the

first 10 capacitors binary weighted. The capacitance for the most-significant-bit (MSB) is 512 C and the capacitance of the 10th MSB is C. The extra capacitor C_{dac} has a capacitance of C and is connected to a 6-bit DAC. The top plates of the main capacitor arrays are connected to the inputs of the comparator during the conversion process and to ground during sampling. The bottom plates are controlled by the SAR Control Logic Unit, which connects them to voltages V_{refp} (positive voltage reference), V_{refn} (negative voltage reference), ground or the input. The first 10 capacitors are used in determining the 10 MSBs while the last 6 bits of the ADC are determined using a resistor-string 6-bit DAC. Two Capacitance Ratio Error (CRE) calibration capacitors are connected to the main capacitor arrays. They remove CRE by coupling the output of an 8-bit Calibration DAC, which is also implemented as a resistor string, to the main capacitor array, as will be explained later. In addition, two adjustable capacitor arrays, the Offset Calibration and Gain Error Calibration capacitor arrays are included to calibrate for offset and gain error of the ADC. The whole system is coordinated by the SAR Control Logic Unit, which is the digital part of the ADC.

2.2 Conversion Process

There are two phases for each conversion. The first is the sampling phase, and the second is the conversion phase.

2.2.1 Sampling Phase

As shown in Fig.2.2, during sampling, the bottom plates of the main capacitors are connected to the differential input driven by the input buffer. The top plates, which are disconnected from the comparator, are grounded. So the input is sampled across the main capacitor arrays. The

sampling time for the ADC is 2us, and during this time, the comparator is in shutdown mode to save power.

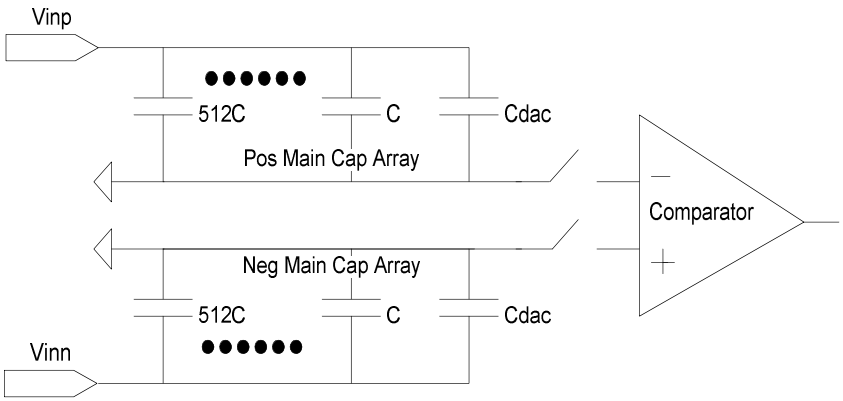


Figure 2.2 Sampling Phase of the ADC

2.2.2 Conversion Phase

After sampling, the ADC enters conversion phase, which is a bit cycling process. The switching pattern of the Positive Main Capacitor Array will be illustrated here to explain how the conversion algorithm works.

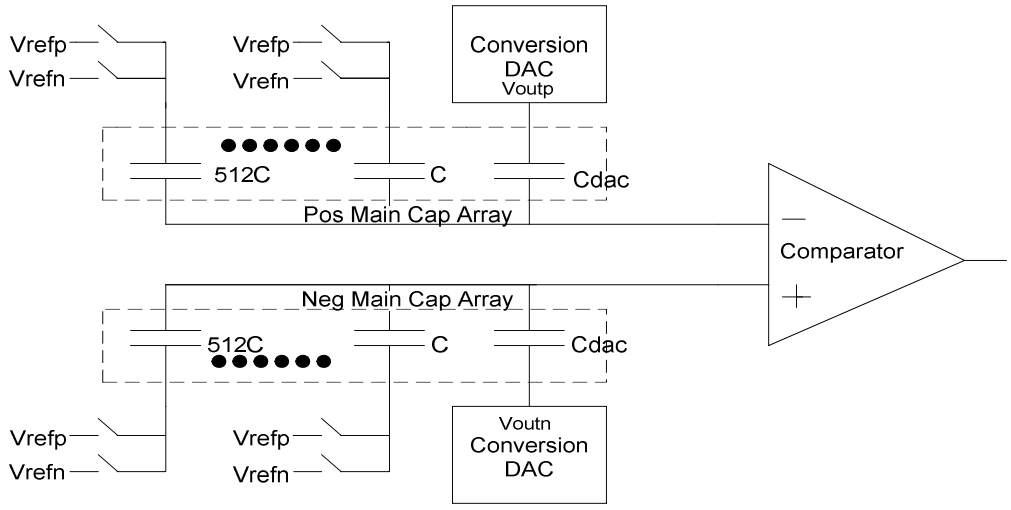


Figure 2.3 Conversion Phase of the ADC

First of all, the MSB capacitor is connected to the positive reference V_{refp}, while the rest of the capacitors are connected to the negative reference V_{refn}. After switching, if the comparator output is 1, the MSB capacitor will stay connected to V_{refp}. Otherwise it will be switched to V_{refn}. Then the 2nd MSB capacitor is connected to V_{refp}, while the remaining MSB capacitors are connected to V_{refn}, and depending on whether the comparator output is 1 or 0, the 2nd MSB capacitor will be connected to V_{refp} or V_{refn}. The bit cycling process will continue for the 10 MSBs. Then, for the last 6 bits, the conversion DAC is used. The positive output of the DAC is coupled to the Positive Capacitor Array through C_{dac}. The conversion algorithm is still the same. First, the MSB of the DAC is set to be 1, and if the comparator output is 1, the MSB will be kept as 1. Otherwise it will be set to 0. Then the remaining bits are set in the same way.

The switching of the Negative Main Capacitor Array is the same, except that the connection to either V_{refp} or V_{refn} is reversed compared to the Positive Main Capacitor Array. For example, if the MSB capacitor of the Positive Capacitor Array is connected to V_{refp}, then the MSB capacitor of the Negative Capacitor Array is connected to V_{refn}.

2.3 Design Challenge

To achieve low power, high speed and high accuracy, there are certain challenges to overcome. These challenges will be explained in detail in this section.

2.3.1 Power

Since the ADC will be operated by battery, its power needs to be as low as possible so that the battery can last several hours. The main source of power consumption comes from the input buffer. Since the input is the front end of the ADC, its accuracy and noise determines the

accuracy and noise level of the whole system. At the same time, it needs to be able to drive large capacitive load, charging them before the sampling time is over. These requirements make the input buffer power hungry. Also, the comparator needs to have low noise and high speed, which is not compatible with low power.

2.3.2 Mismatch, Parasitics, Charge Injection

Mismatch, Parasitic and Charge Injection are all sources of errors for the ADC. First of all, since the ADC works by redistributing electronic charge among capacitors, the mismatch between these capacitors can lead to conversion errors. Also, since there is also parasitic capacitance associated with the top plate of a capacitor, the difference in parasitics can lead to a gain error of the ADC. In addition, Charge Injection from the switches to the capacitors can vary with different input levels. To remove these errors, it is necessary to add calibration circuitry to the ADC.

2.3.3 Noise

The ADC is 16 bit, with the first bit being the sign bit. Meanwhile, the reference voltages are +0.8V and -0.8V. Therefore, the LSB of the ADC is $1/2^{15}$ of the reference voltages difference (1.6V), which is 48uV. The noise of the whole system should be kept under 1/4 LSB, which is approximately 12uV. Noise comes from input buffer, comparator and the kT/C noise of the capacitor array. For the input buffer, a huge input pair is needed to keep it low noise. For the comparator, which has multiple stages, the noise of the first stage should be controlled to a low level. This again can be done by increasing the size of the input pair. To reduce kT/C noise, large capacitor values can be chosen at the expense of die area. For this design, to make the kT/C noise

lower than 12uV, the total capacitance of each capacitor array is chosen to be 50pF, resulting in a kT/C noise of 9.1uV for each array.

2.4 Interleaving of two ADCs

As mentioned at the beginning of this section, to achieve a low power 500KSps ADC with 16 bit accuracy, interleaving is used. Two capacitor arrays, two comparators and two sets of calibration circuitry are put combined, controlled by a single digital block to implement an interleaving ADC. Whenever one capacitor array is converting, the other is sampling, with its comparators shut down to save power.

3. ADC BUILDING BLOCKS

As shown in Fig.2.1, the analog building blocks of the ADC are the comparator, a 6-bit DAC and an 8-bit DAC. In addition, an input buffer is needed. This chapter will discuss the design of these blocks.

3.1 Input Buffer

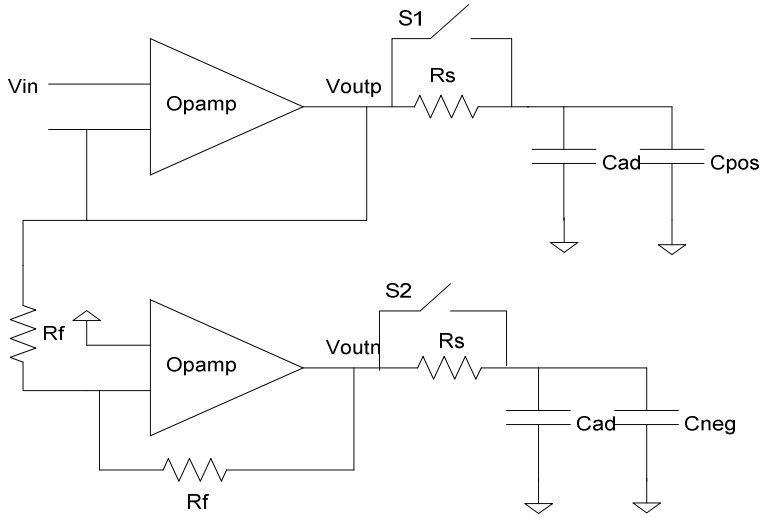


Figure 3.1 Input Buffer Architecture

The architecture of the input buffer is shown in Fig.3.1. It takes in a single ended input analog signal and converts it into a differential input pair, which will be sampled onto the main capacitor array during sampling. It consists of two identical Operational Amplifiers (op-amp), one configured in unit gain feedback and the other configured in inverting feedback mode. The feedback resistors of the inverting buffer are chosen to be 1K to minimize their thermal noise. The output of the unit gain and inverting buffer are not directly connected to the positive and negative capacitor array. Instead, low pass RC filters are added to filter out the high-frequency

noise of the buffer. The filter is made of R_s and C_{ad} as shown in Fig.3.1, with R_s equal to 1.8K and Cad equal to 50pF. Taking into account the capacitance of the positive and negative capacitor array, the RC time constant is 180ns, setting the cutoff frequency of the filter to be 884KHz. The filters effectively limit the noise of the op-amp and feedback resistors from being applied to the capacitor arrays.

This architecture poses a great challenge on the design of the op-amp used within the buffer. Since the power supply of the ADC is only ± 1V, and the input range is ± 0.8V, the op-amp needs to have both large input range and output range. Moreover, ideally the output noise needs to be lower than 1/4 of the LSB of the ADC, which is about 15uV, so the op-amp needs to be low noise. Also, it needs to be able to charge the capacitor array plus the Cad, each with a capacitance of 50pF, within the sampling time of 2us. In addition, the op-amp should be low power. Below is a discussion of the op-amp design.

First, a simple two stage op-amp with Class A output stage was designed. The problem with two-stage op-amp is that it can't handle the large input range. If the input pair is PMOS, it won't be able to work when the input is close to the positive power supply. On the other hand, if NMOS input pair is chosen, the op-amp won't work for input close to the negative power supply. In addition, Class A output stage can't drive the 1K feedback resistors to the voltages needed. Next, a Current Mirror op-amp (CM op-amp) was designed. For CM op-amp, the output driving capability is improved, but still not good enough to drive the 1K resistors.

The final design of the op-amp is shown in Fig.3.2. It consists of a Rail to Rail input stage and a Class AB output stage. A cascode intermediate stage is added to boost the gain of the op-amp. The Rail to Rail input stage can handle input close to either the positive or negative power supply.

For an input close to positive supply, the PMOS input pair will be cut off, but the NMOS pair still works. For an input close to negative power supply, the NMOS pair will be cut off, but the PMOS pair still works. The current going through the PMOS pair should be N times the current going through the NMOS pair, where $N = \mu_N / \mu_P$, and μ_N and μ_P are the mobility of electrons and holes respectively. This current ratio will make sure that the transconductance, G_m of the NMOS pair and PMOS are the same, thus reducing the Gm variation over the input range. Constant Gm Rail to Rail input stages have been studied extensively [5] [6], but for this application it is not necessary to have constant Gm over the input range because the op-amp always work in feedback configuration. It is worth noting that without constant Gm, the noise at the output node can vary with input level, but again, for this application, the variation is only 1uV, which is negligible.

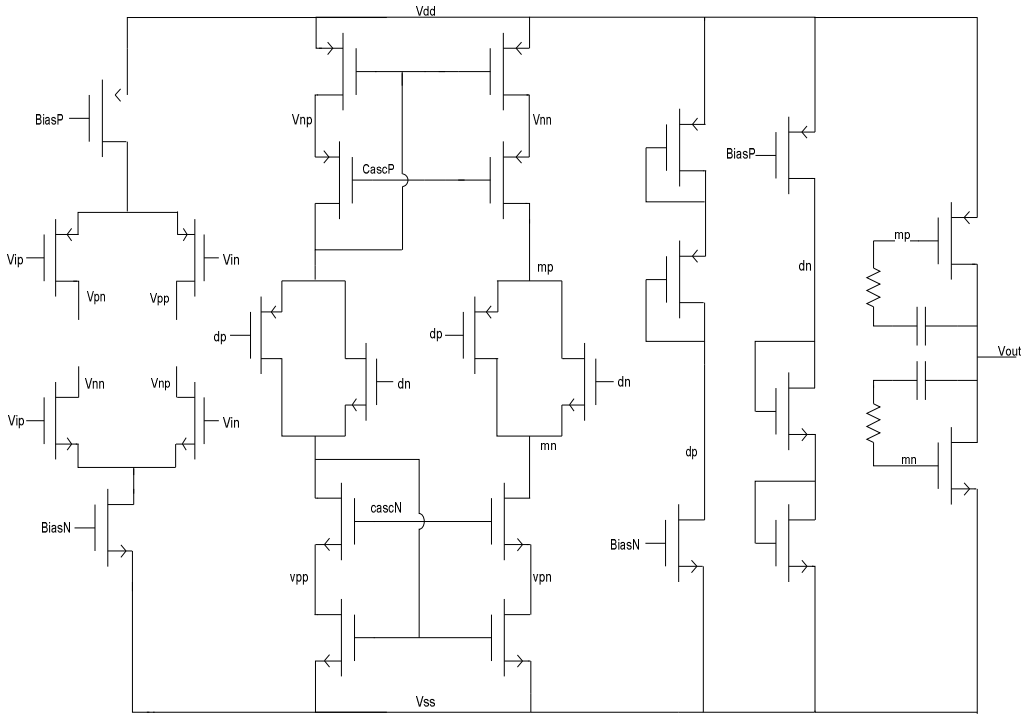


Figure 3.2 Rail to Rail Input, Class AB Output Op-amp

To keep the op-amp noise low, it is crucial to have extremely high Gm. This is because in unity gain feedback, the noise at the output is equal to the open loop noise divided by the op-amp's open loop gain. Thus huge input pairs are used for both the NMOS and PMOS pair.

The cascode stage is commonly referred to as the current summing stage. It adds the current signals of the complementary input stage, and transfers the sum to the output stage. In this design, the control circuit for the Class AB is shifted into the current summing stage. This is because in the Class AB output stage, the bias circuit is always noisy. Shifting the bias into the current summing stage can reduce the noise contribution from the Class AB bias [7].

Lead compensation is needed at the output stage. In fact since the first pole of the op-amp is very big due to the large G_m at the input stage, the capacitor needed for Lead Compensation is fairly large.

As shown in Fig.3.1, because of the RC filters added into the buffer to reduce its noise, there is a RC delay from the buffer output to the load capacitors. This RC delay slows down the circuit substantially. As a result, two switches, S1 and S2, are introduced into the buffer. During the first microsecond of the sampling time, S1 and S2 are closed, thus shorting the resistors in the RC network. In this way the capacitors are charged quickly to be very close to their final value. Then, during the remaining one microsecond of the sampling phase, S1 and S2 are opened to let the RC network filter out the buffer noise.

3.2 Comparator

As mentioned earlier, the comparator consists of two blocks, a coarse comparator which is low power and high noise, and a fine comparator which is high power and low noise. During sampling, both comparators are shut down with their input nodes reset to ground. During

conversion, for the first 8 MSB bit cycling, the coarse comparator is used while the fine comparator is shut down. Then, the fine comparator is turned on for later conversion while the coarse comparator is shutdown. There is possibly some hysteresis associated with the coarse comparator, so when the transition from coarse to fine conversion is made, an extra hysteresis removal algorithm is added, as will be explained later.

3.2.1 Coarse Comparator

The block diagram of the coarse comparator is shown in Fig.3.3. It consists of a first stage high speed op-amp and a latch comparator. The schematic of the op-amp and the latch comparator are shown in Fig.3.4 and Fig.3.5 respectively.

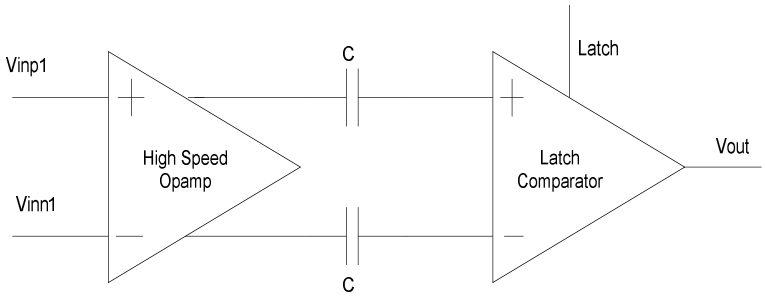


Figure 3.3 Coarse Comparator

As shown in Fig. 3.4, the high speed op-amp is a cross coupled positive feedback op-amp. An output stage is added to shift the DC component of the output to the middle of the rail. To achieve high speed, it is important to reduce the size of the cross coupled loading FETs M3-M6, because their gate capacitance can slow down the charging and discharging of the first stage output node, thus slowing down the output of the op-amp. The input pair FETs are not very big for high speed consideration. The loading at the second stage are two diode connected NMOS

transistors. Diode connection is chosen because of low gain, high speed consideration. A shutdown switch is added into the circuit. It shuts down the op-amp by disconnecting the current source from the op-amp.

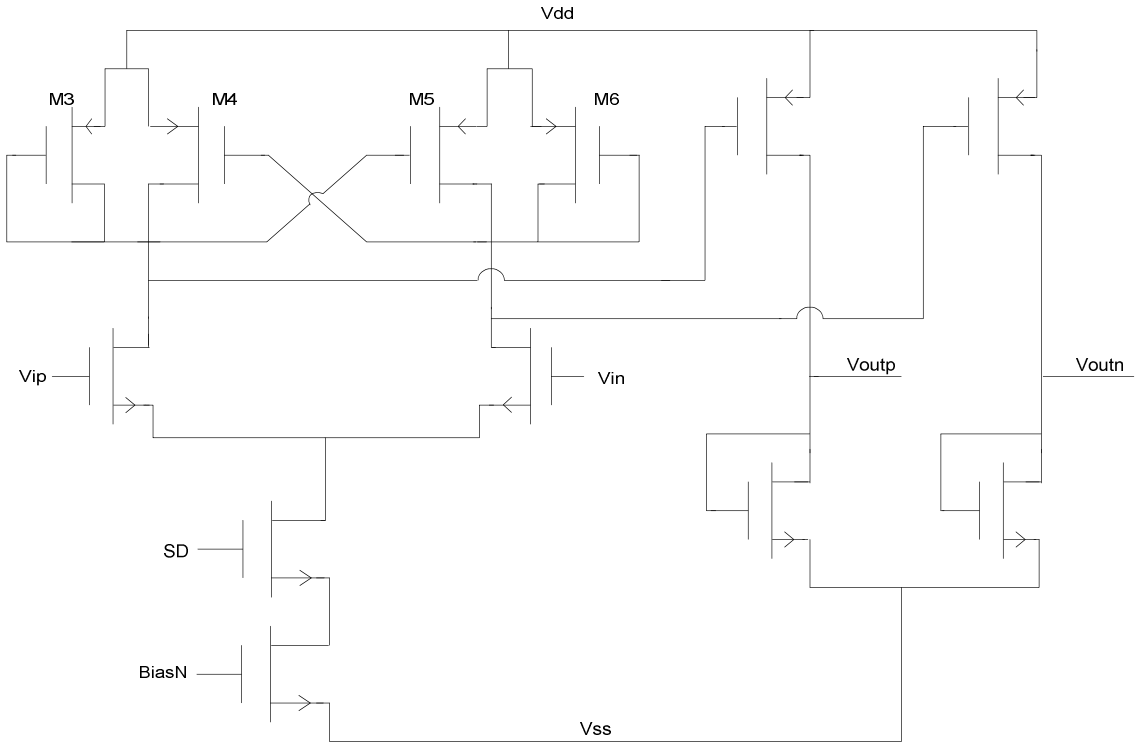


Figure 3.4 High Speed Cross Coupled Op-amp

The schematic of the latch comparator is shown in Fig.3.5. Clock Phi1 and Phi2 are non-overlapping clocks generated by an input latch clock. When the Phi1 is low, the comparator is in reset phase. As soon as the Phi1 goes high, the comparator will be latched to either V_{dd} or V_{ss} depending on the input. The kickback from the second stage to the input stage when Phi1 goes low needs to be minimized because it can cause errors for LSB conversions. In this design, extra capacitors are added at the input nodes of the comparator to reduce kickback.

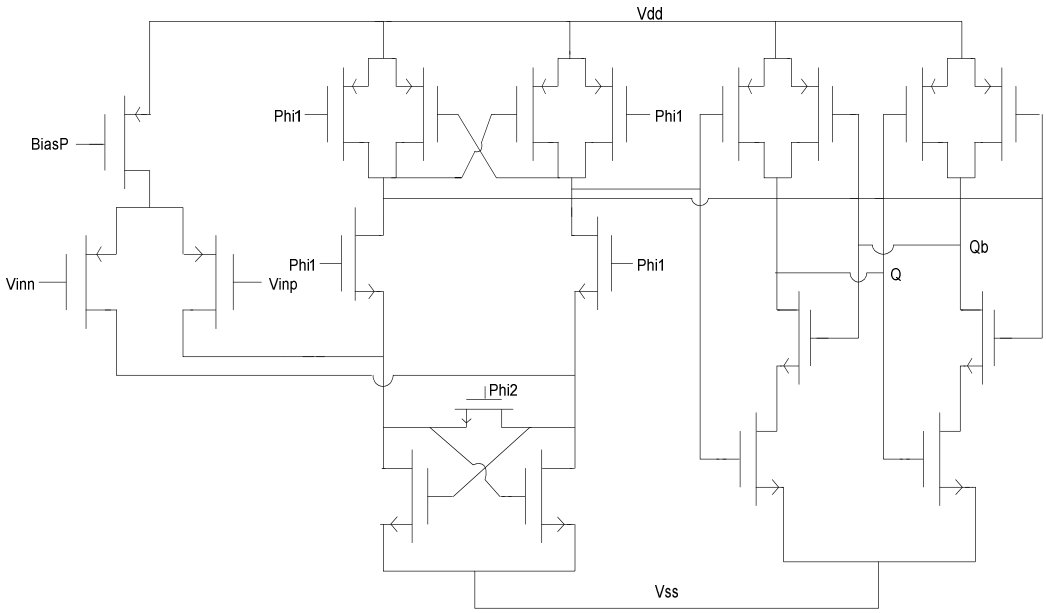


Figure 3.5 Latch Comparator

3.2.2 Fine Comparator

The block diagram for the fine comparator is shown in Fig.3.6. It has two amplifying paths, one for normal conversion and the other for calibration. During normal conversion, the calibration path is shut down while during calibration the normal conversion path is shut down.

The normal conversion path and the calibration path share the first two stage high speed op-amps, which are the same op-amp as the high speed op-amp used for coarse comparator as shown in Fig.3.4. The normal conversion path is a latch comparator with the same design as the coarse comparator (shown in Fig.3.5). As for the calibration path, two extra stages are added to achieve higher gain and therefore higher resolution, because during calibration an extra two-bits of accuracy is needed. The first stage op-amp is again a high speed op-amp, and the last stage is a simple two stage op-amp, as shown in Fig.3.7.

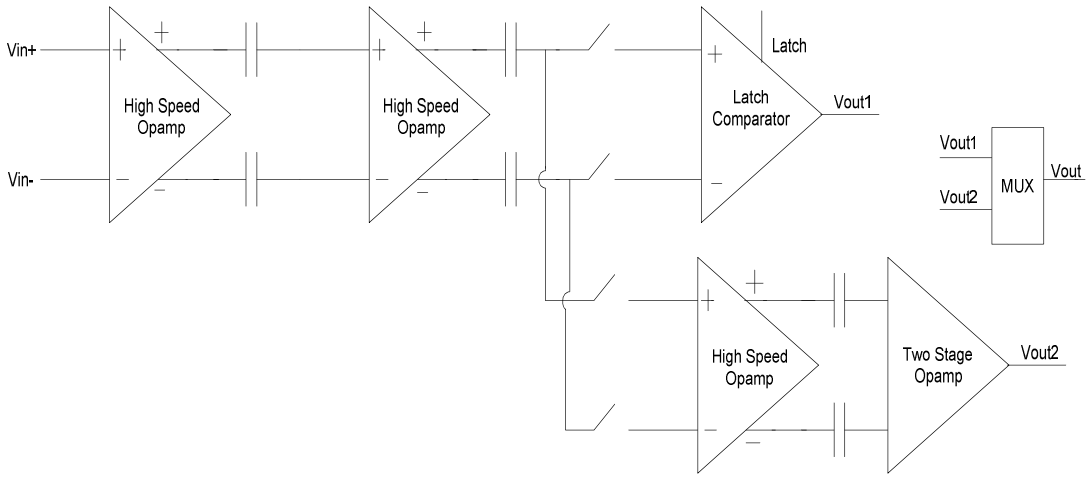


Figure 3.6 Fine Comparator

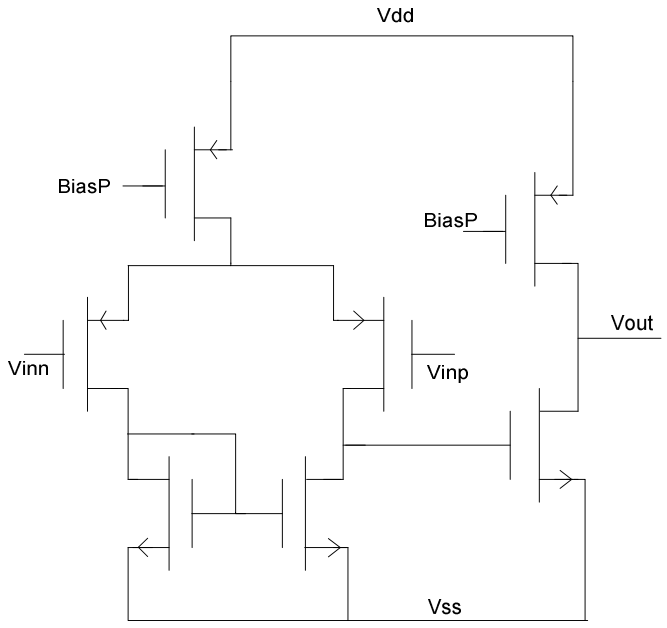


Figure 3.7 Two Stage Op-amp In Calibration Path

3.2.3 Coarse to Fine Transition, Hysteresis Removal

As mentioned before, during normal conversion, the coarse comparator assumes the conversion of the first 8 MSBs while the fine comparator converts the last 8 bits. When switching from coarse comparator to fine comparator, an extra hysteresis removal process is

needed. This is due to the fact that the coarse comparator is handling large input signals, and under large input stress, a time-dependent offset shift will occur in the comparator. Assuming that the error produced by the coarse comparator is 1 bit at the 8th bit level, a hysteresis removal algorithm is implemented in this design [3].

Basically, the algorithm works this way: suppose the 8 MSB conversions are done and the digital output from the coarse comparator is D, which is an 8 digit number. Next, the connection of the main capacitor arrays will be kept as D, and the fine comparator is used to compare the voltages on the top plates of the main capacitor arrays, and the comparison result will be recorded. Then the connection of the main capacitor arrays will be set to be D+1 and the fine comparator will compare the top plate voltages again. The final result will be set according to the fine comparator output for D and D+1, as shown in Table 3.1.

Fine Comparator, D	**Fine Comparator, D+1**	**Final Result**
0	0	D-1
1	0	D
1	1	D+1

Table 3.1 Hysteresis Removal Algorithm

3.3 Resistor String DACs

The ADC utilizes two DACs, one is 6 bits and is used for conversion and the other is 8 bits and is used for calibration. Both of these DACs are resistor string DACs, but with different decoding methods. Fig.3.8 shows the design of the 6-bit DAC. A 64 resistor string is connected

between V_{refp} and V_{refn} to generate the 64 different voltage levels. The middle tap of the string is tied to ground to reduce the gain error of the DAC. Then the 64 levels go into a multiplexer, and the output will be selected according to the input code. Once the resistor value and the loading capacitance are fixed, the speed of the DAC depends primarily on the size of the switches in the multiplexer. On one hand, the parasitic capacitance of the switches can add up to a large value, thus resulting in a large RC delay time constant. On the other hand, if the switches are too small, then their high resistance can also add to the RC delay at the output. As a result, an optimal size is required.

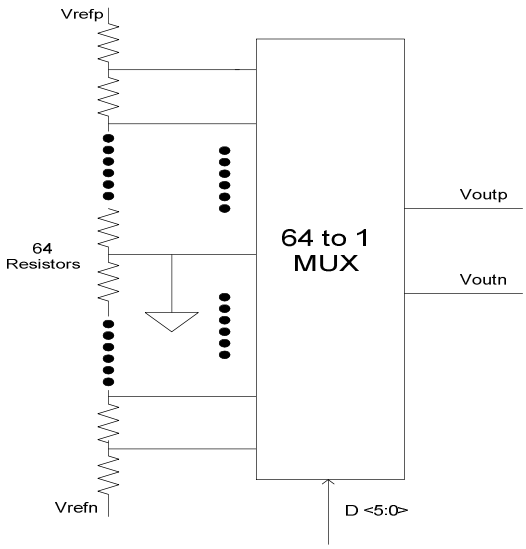


Figure 3.8 Topology of the 6 BIT DAC

As for the 8 bit DAC used for the calibration, the decoding method is different. At first the same architecture as the 6 bit DAC is used, but for an 8-bit DAC, a 256 to 1 multiplexer is needed, so the parasitic capacitance from the switches is 4 times that of the 6 bit DAC. This slows down the circuit greatly, so using minimum size switches does not help much with the speed. To obtain high speed, a different decoding scheme is utilized and shown in Fig.3.9. In this

design, the 256 resistors are divided into 16 blocks, each with 16 resistors. The 16 different levels generated by the 16 resistors are fed into a 16 to1 multiplexer, where the inputs of the multiplexer are the 4 LSBs of the DAC input. Since there are 16 blocks, there are 16 voltage levels coming out of the 16 multiplexers. Then, these 16 levels are again fed into a 16 to 1 multiplexer, where the inputs of the multiplexer are the 4 MSBs of the DAC input. In this tree topology, the delay of the DAC output is reduced significantly, because the parasitic capacitance at the output node is reduced by a factor of 16.

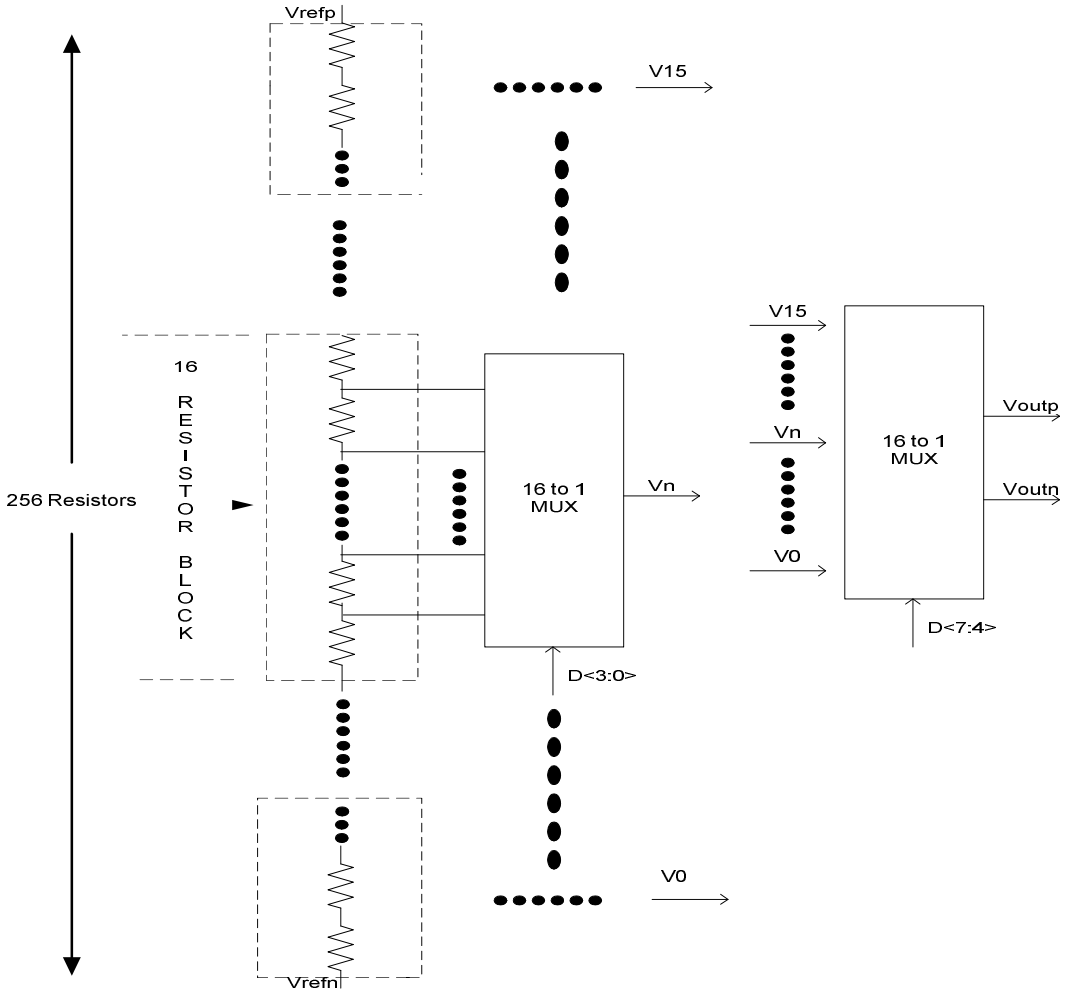


Figure 3.9 Topology of the 8 BIT DAC

4. CALIBRATION ALGORITHM

While the speed of SAR Charge Redistribution ADCs is determined primarily by the fabrication process, its accuracy depends largely on design techniques [4]. Several error mechanisms exist which set limits to the accuracy of SAR ADC. These errors will be explained in this chapter. Calibration circuits used to remove these errors are designed and presented as well.

4.1 Capacitor Ratio Calibration

Capacitor Ratio Error (CRE) caused by process variation is the largest source of error for SAR ADC [4]. To remove CRE, a calibration algorithm is implemented in this design [8] [9]. Fig.4.1 shows the detailed calibration circuit.

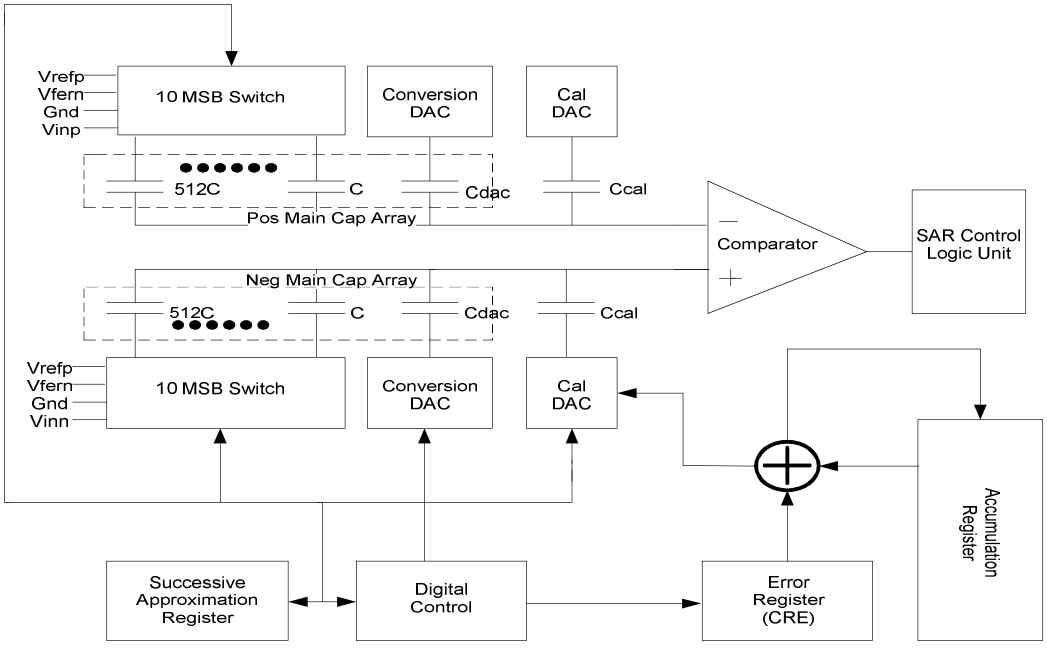


Figure 4.1CRE Calibration Algorithm

The calibration works by measuring the conversion error caused by each of the MSB capacitors and storing them in the Successive Approximation Registers. Later, during normal conversion, the stored errors are subtracted from the conversion result through the calibration DAC.

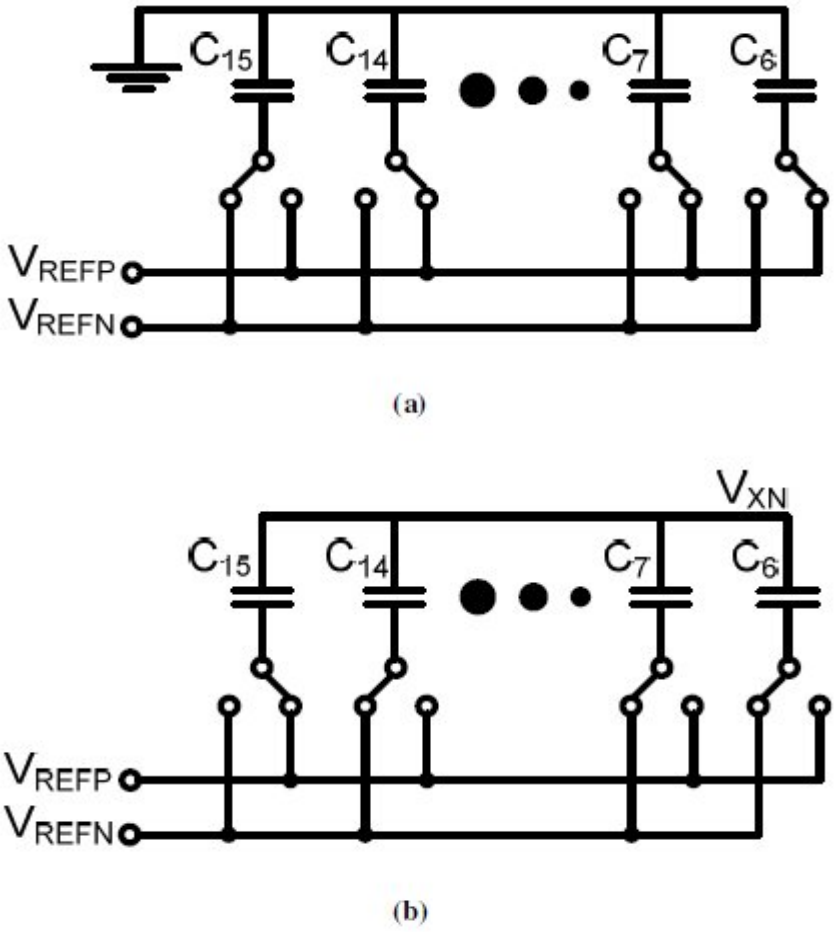


Figure 4.2 CRE Calibration Process (a) CRE charging cycle. (b) Charge Redistribution to obtain error residual voltage

As shown in Fig.4.2 [9], the process for measuring CRE errors begins by sampling V_{refp} on the MSB capacitor C_{15} and V_{refn} on the rest of the capacitors. Then, C15 is connected to V_{refn} and the rest of the capacitors are connected to V_{refp}. During this reversing process, if there is a mismatch

between C_{15} and the rest of the capacitors, i.e. C_{15} is not equal to the sum of the rest of the capacitors, there will be a remaining non-zero voltage V_{RVN} left on the top plates of the capacitor array. The relation between V_{RVN} and the error voltage introduced by C_{15} V_{ERRN} is given by equation (4.1) [9]

$$V_{RVN} = 2V_{ERRN} \tag{4.1}$$

The error voltage will be converted into a digital number using the 8 bit DAC and stored in Successive Approximation Registers. The errors due to the rest of the capacitors are obtained in the same way, using equation (4.2) [3]

$$V_{ERRn} = \frac{1}{2}\left(V_{RVn} - \sum_{i=n+1}^{N} V_{ERRi} \right), n = 6,7...N-1 \tag{4.2}$$

Digitally, this can be written as

$$D_{ERRn} = \frac{1}{2}\left(DV_{RVn} - \sum_{i=n+1}^{N} DV_{ERRi} \right), n = 6,7...N-1 \tag{4.3}$$

Where DV_{ERRi} is the digitized CRE due to the nth MSB capacitor and DV_{RVn} is the accumulated digitized error during the calibration of the nth MSB capacitor respectively.

The CRE calibration is performed on each of the 10 MSB capacitors. Then during normal conversion, during the bit cycling process, if the current bit being tested is the nth bit, then the error corresponding to the nth capacitor, will be added to the accumulative register as shown in Fig.4.1. Then, if the comparator output is one, which means the current bit will be set to 1, the error will be kept in the accumulative register. Otherwise the error will be discarded. The CRE calibration algorithm described here doesn't require complex analog circuit or complicated digital logic to implement, yet can remove the CRE error effectively.

4.2 Offset Calibration

The offset error of the ADC comes from two sources:

1. Charge Injection from the switches in the capacitor array.

2. Offset of the comparator due to device mismatch.

To cancel the offset, a known signal is coupled to the top plates of the capacitor through the offset calibration capacitor.

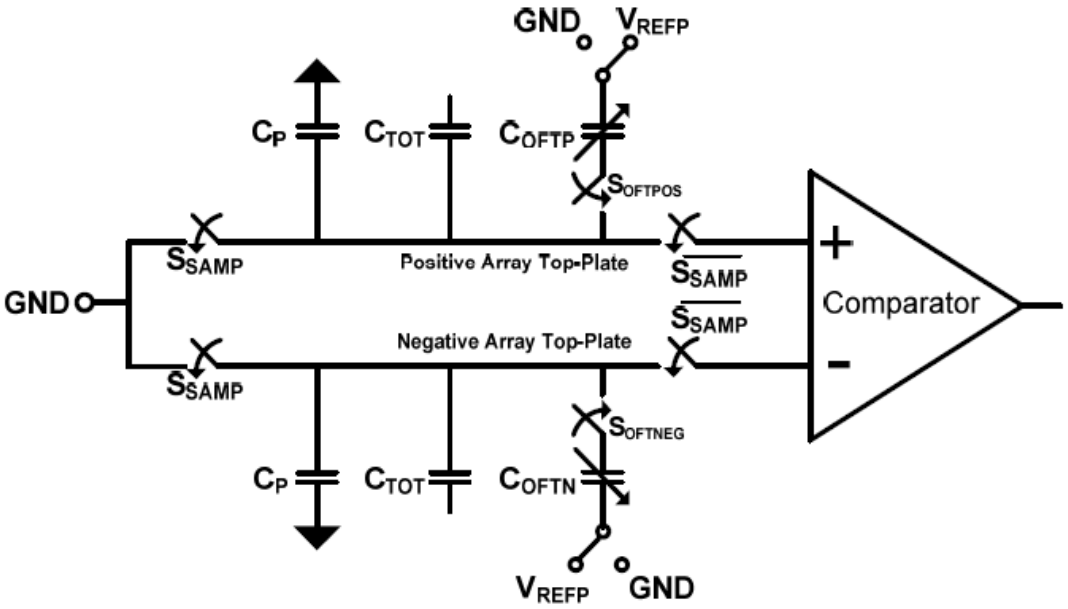


Figure 4.3 Offset Calibration Algorithm

Fig 4.3 shows the offset calibration algorithm [8] [9]. There are two offset calibration capacitor C_{OFTP} and C_{OFTN}. Depending on whether the ADC has a positive or negative offset, C_{OFTP} or C_{OFTN} will be utilized, controlled by the two switches S_{OFTPOS} and S_{OFTNEG} as shown in the figure. Suppose the ADC offset is positive, which means the voltage at the positive array top plate is higher than that of the negative array when the input is 0, then an extra circuit is needed

to lower the positive array top plate voltage. In this case, the top plate of C_{OFTP} will be connected to the positive array top plate. On the other hand, the bottom plate of C_{OFTP} will be connected to V_{refp} during sampling phase, and to ground during conversion phase. That is to say, entering conversion phase, the voltage at bottom plate of C_{OFTP} is reduced by V_{refp}, which will force the top plate of the C_{OFTP}, and hence reduce the voltage on the top plate of the positive array to balance the voltage change at the bottom plate of C_{OFTP}.

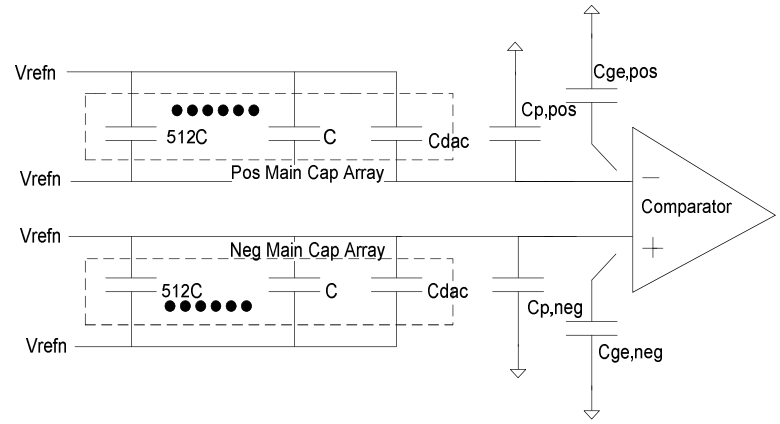
Since the offset of the comparator can be any value, it is important to choose the right capacitance for C_{OFTP} so as to cancel the offset accordingly. To do so, two binary weighted capacitor array are used as C_{OFTP} and C_{OFTN} , and an algorithm to find the optimal capacitance value is introduced.

First of all, without C_{OFTP} and C_{OFTN} , the ADC will convert a 0V input. If the conversion result is larger than 0, then the switch from C_{OFTP} to the positive array top is closed. Then, for the next conversion, the MSB capacitor of the C_{OFTP} array is used, with the rest of the capacitors left unconnected. If the conversion result is still larger than 0, then the MSB capacitor will be kept and the 2nd MSB capacitor will be added in. Otherwise the MSB capacitor will be discarded and only the 2nd MSB capacitor will be used for the next conversion. This process continues until the ADC conversion result is 0. By the end of the bit cycling, the optimal value for offset cancelling capacitor will be found.

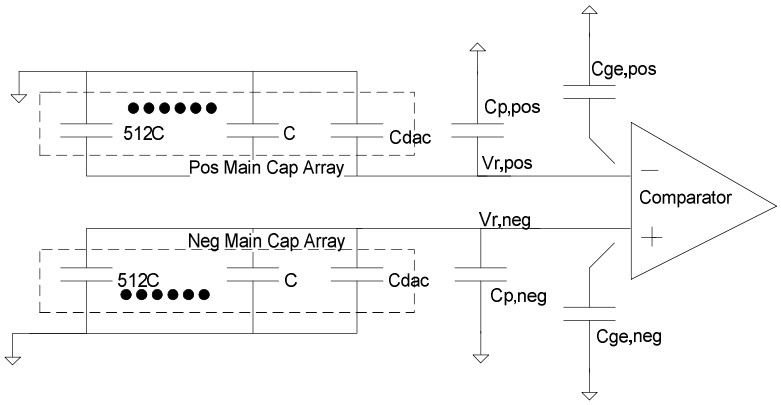
During normal conversion, the already trimmed binary weighted capacitor array will be applied as described in Fig.4.3 to cancel the offset of the ADC.

4.3 Gain Error Calibration

In CMOS process, there is always parasitic capacitance associated with capacitors. In this ADC design, the top plates of both the positive and negative capacitor arrays will have some parasitic capacitance to ground after being fabricated on chip. If there is a difference in the parasitic for the positive and negative array, then a Gain Error will be introduced into the ADC.



(a)



(b)

Figure 4.4 Gain Error Calibration Algorithm

(a) Sample V_{refn} to both plates of the capacitors (b) Ground the bottom plates

To make the parasitics equal, a Gain Error Calibration algorithm is implemented in the design and is shown in Fig.4.4. The $C_{p,pos}$ and $C_{p,neg}$ represent the parasitic capacitance at the top plates of the positive and negative array. $C_{p,pos}$ and $C_{p,neg}$ are two binary weighted capacitor arrays used to compensate for the gain error of the ADC. Both $C_{p,pos}$ and $C_{p,neg}$ are disconnected from the top plates at the beginning of calibration. The calibration starts by connecting both the top and bottom plates of the main capacitor arrays to V_{refn}. $C_{p,pos}$ and $C_{p,neg}$ are also charged to V_{refn}. Then, the bottom plates are connected to ground while the top plates are left floating. Ideally, without $C_{p,pos}$ and $C_{p,neg,}$ the top plates should be at 0V. However, due to the existence of $C_{p,pos}$ and $C_{p,neg}$, there are some residual voltage left at both of the top plates. If there is a mismatch between $C_{p,pos}$ and $C_{p,neg}$, the comparator will detect the mismatch. Then, either $C_{p,pos}$ or $C_{p,neg}$ will be added to the capacitor array to compensate for the mismatch, i.e. the difference between $C_{p,pos}$ and $C_{p,neg}$. Same as in offset calibration, the value of $C_{p,pos}$ or $C_{p,neg}$ is found by bit cycling.

5. SIMULATION RESULT

The ADC is designed in 0.18um SiGe CMOS process. In this chapter, the performances of the building blocks are presented first. Then, the performance of the ADC as a whole is presented.

5.1 Input Buffer

5.1.1 Power

As discussed in Chapter 3, the input buffer is built using a Rail to Rail input, Class AB output op-amp. The overall power dissipation and noise of the buffer depends on the input level, since the Rail to Rail input stage is not designed with Constant Gm bias. Fig 5.1 shows a DC sweep of the current dissipation of the buffer.

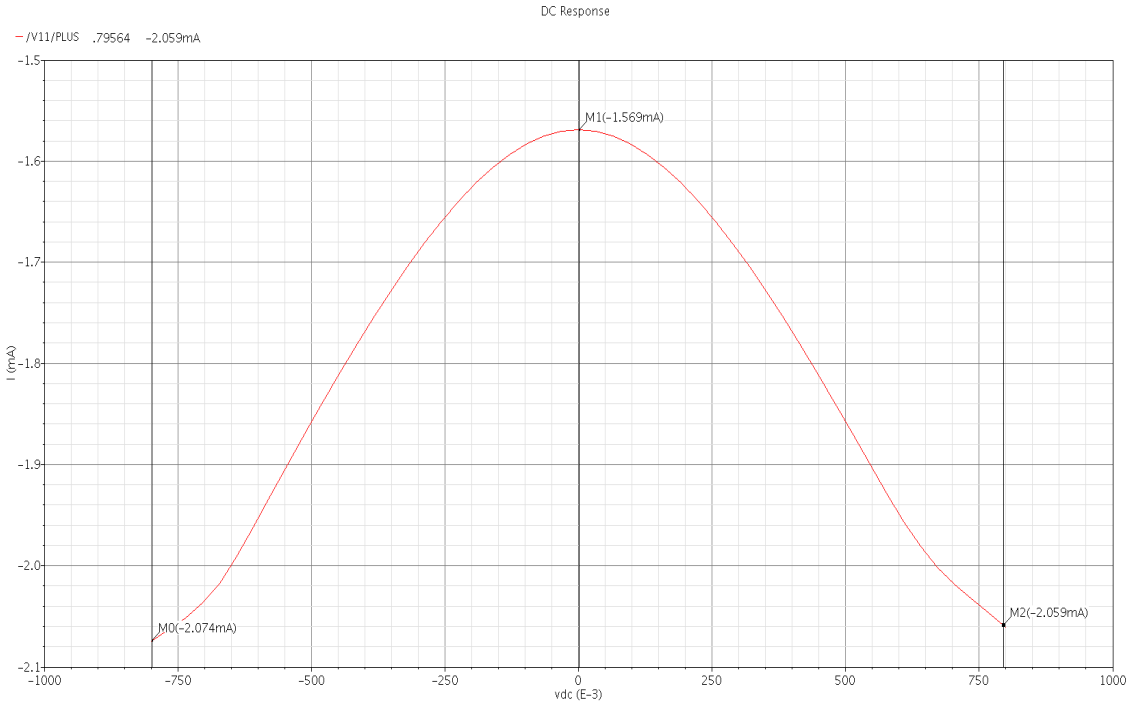


Figure 5.1 Current Dissipation of the Input Buffer

As can be seen, at 0 input, the buffer consumes a minimum current of 1.57mA. At ± 0.8V inputs, the current goes up to 2mA. The average current dissipation is 1.825mA over the input range, thus the average power of the input buffer is 3.65mW.

5.1.2 Noise

The buffer is consuming a lot of power due to its low noise requirement. Ideally, the buffer noise should be kept at a quarter of the voltage resolution of the ADC, which is 15uV in this design. The integrated noise of the buffer is found to be 19.1uV at both 0.8 and -0.8V input, and 18.0uV at 0V input. The noise variation is caused by the change of Gm of the buffer op-amp over the input range. At 0V, both the PMOS and NMOS pair in the Rail to Rail input stage are working, thus the Gm is large. At 0.8 or -0.8V, PMOS or NMOS pair is not working, thus Gm is reduced. Fig.5.2 shows a sweep of the buffer noise at 1Hz.

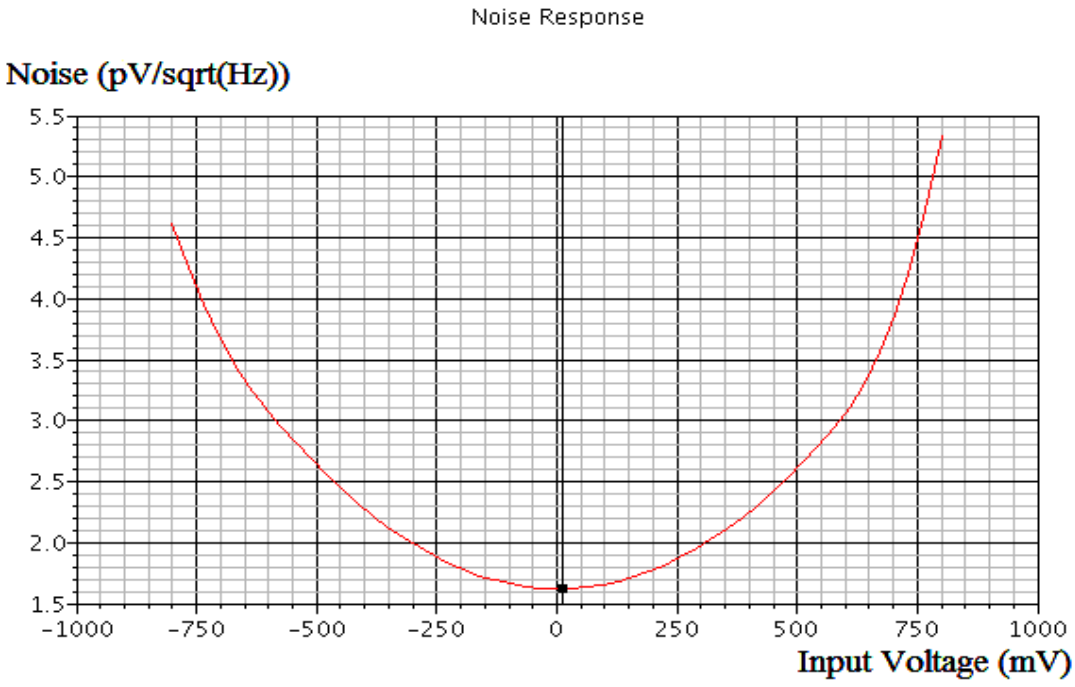


Figure 5.2 Noise at 1Hz Over The Input Range

On average, the integrated noise of the buffer is 18.6uV, which is larger than the ideal value of 15uV. The main noise sources are the feedback resistors as shown in Fig3.1. The value of these resistors, R_f, is set to be 1K. To cut the noise down further, R_f needs to be reduced, meaning more power will be needed to drive the resistors. Since the buffer is already dissipating 3.65mW of power, no further effort is made to reduce the noise.

5.1.3 Accuracy

The accuracy of the buffer is also an important factor, because it determines the accuracy of the ADC. Fig.5.3 plots the difference between ideal and actual buffer outputs. The difference is 113.1uV at -0.8V input and -147.4uV at 0.8V input. As can be seen, except for a small gain error, the buffer works well over the input range.

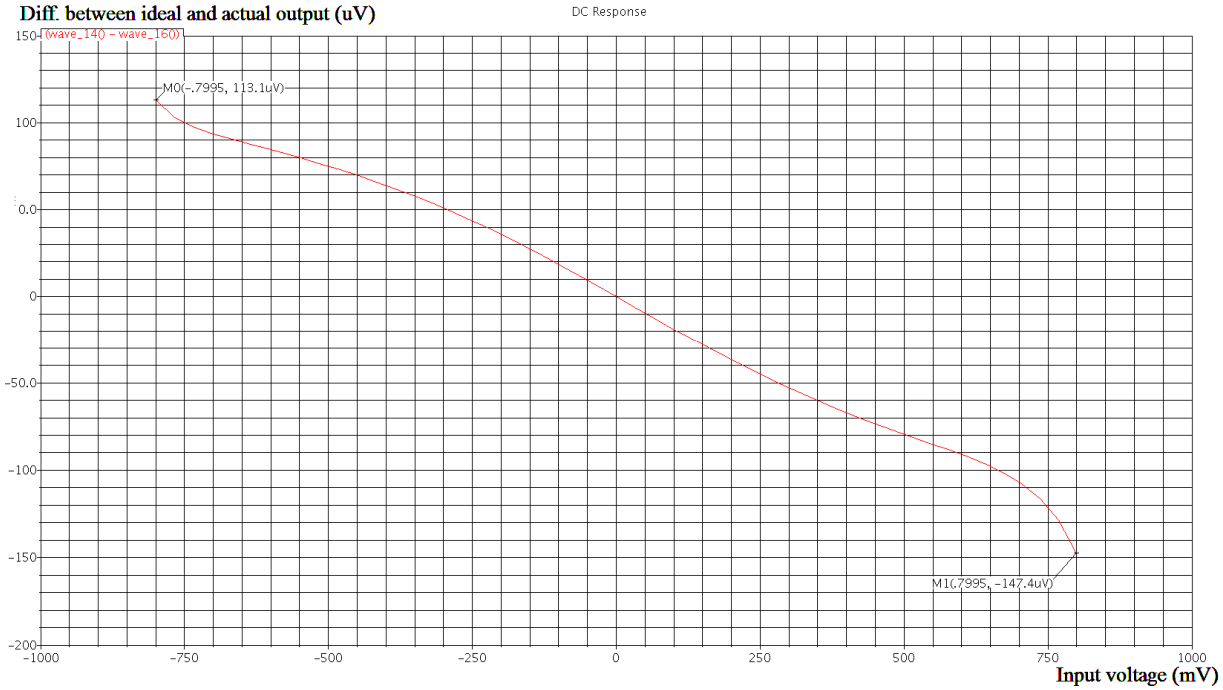


Figure 5.3 Output DC Sweep of the Buffer

5.1.4 Speed

One of the challenges of the buffer design is speed as discussed in Chapter 3. Because of the introduction of switches S1 and S2, the problem is solved.

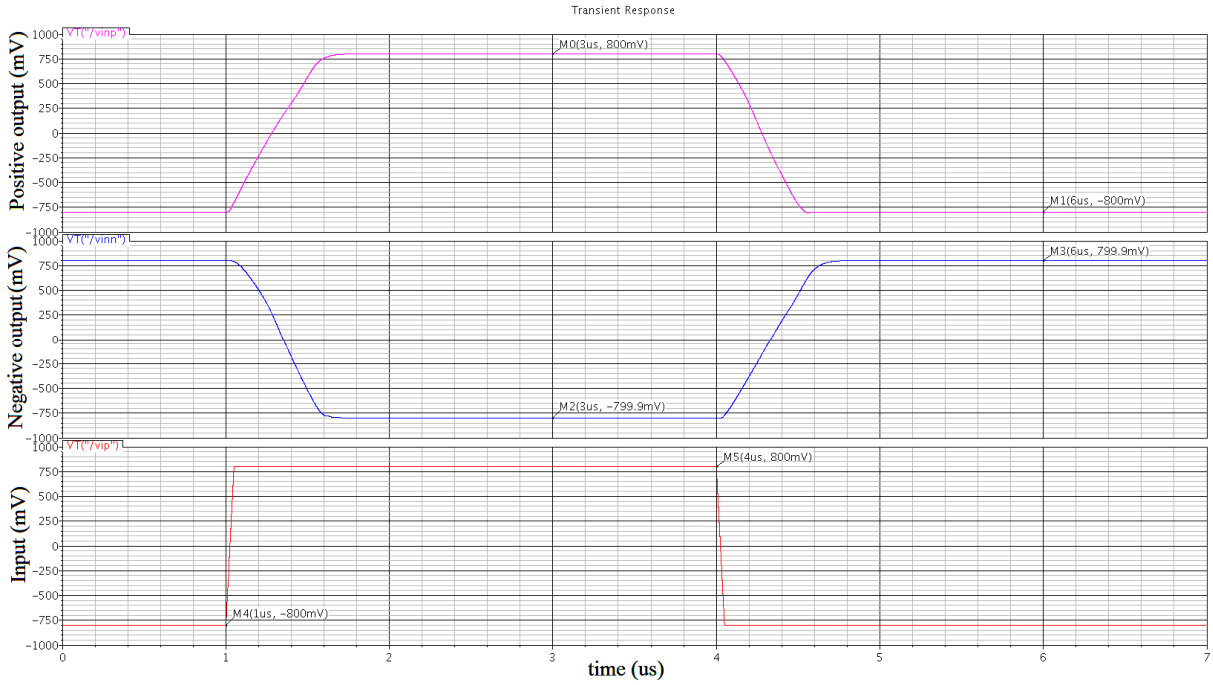


Figure 5.4 Transient Behavior of the Input Buffer

Fig.5.4 shows the transient response of the input buffer under a square wave input signal with amplitude of 0.8V. As can be seen, both positive output and negative output can slew quickly, i.e. within the sampling time of 2us, between +0.8V and -0.8V. In addition to large slew rates, no oscillation is observed at the output nodes.

Overall the input buffer designed is low noise, high speed and high accuracy, while consuming reasonable amount of power. However, further study on the architecture level needs to be conducted to reduce its noise without extra power dissipation.

5.2 Comparator

The main design consideration for the comparator is power, speed and noise.

5.2.1 Power

To reduce power consumption, shutdown signals are introduced to both the coarse and fine comparator. This effectively reduces the power of the comparator by half.

Simulation results show that the power of the coarse comparator is 190uW, and that of the fine comparator is 940uW. During normal conversion, the coarse comparator is used for the first 8 clock cycles, while the fine comparator is used for the remaining 12 clock cycles. The weighted total power of the comparator is calculated as

$$P_{total} = P_{coarse} \times 8/20 + P_{fine} \times 12/20 = 640uW$$

It is worth noting that the majority of the power is consumed in the first stage of the fine comparator, because it needs to be low noise.

5.2.2 Noise

For the coarse comparator, noise is not an issue because it only converts the 8 MSBs of the ADC, and for MSB conversion, the signal voltage is very large compared to the noise. As a result, the noise of the coarse comparator will not be covered here.

The fine comparator noise comes mainly from its first stage. The noise of the first stage comes mainly from its input pair. Large input pair FETs are selected to minimize their flicker noise. Fig.5.5 shows the noise distribution at the output node of the first stage. The input referred noise integrated up to 10GHz is 11uV, which is lower than 1/4 V_{LSB} (12uV).

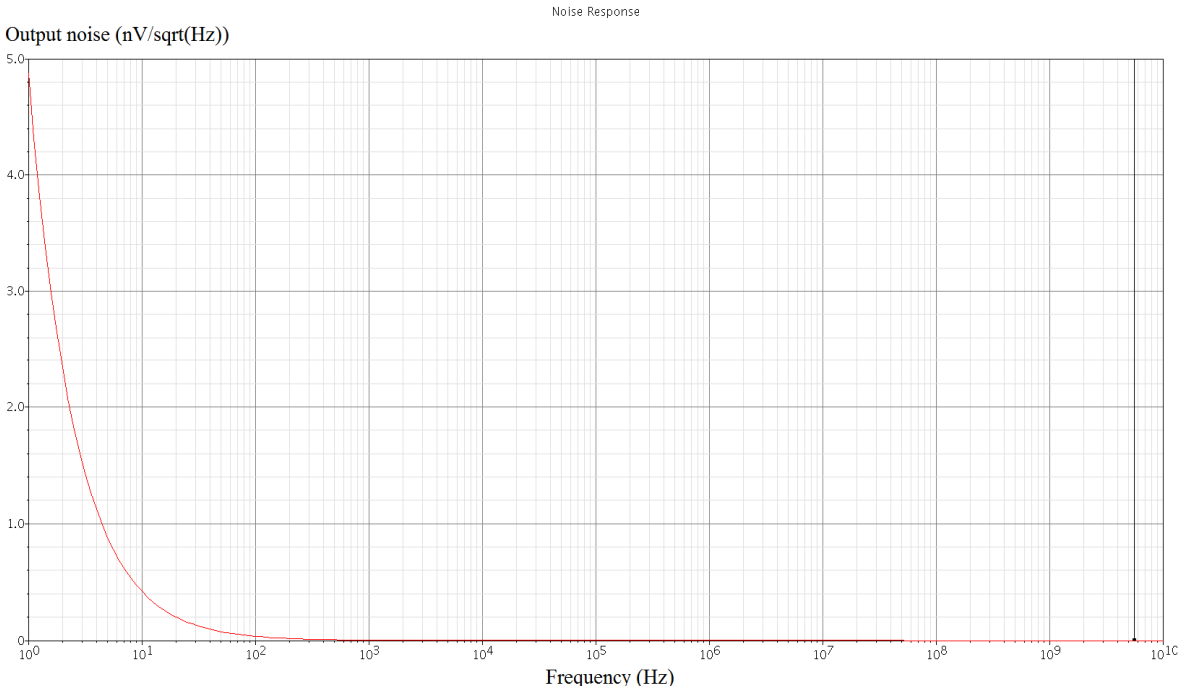


Figure 5.5 Noise Distribution of the First Stage Op-amp Output

5.3 DACs

5.3.1 Power, Noise

The total resistance between V_{refp} and V_{refn} is 64K for both the 6 bit and the 8 bit DAC, and the power is calculated to be 50uW, which is small compared to input buffer and comparator power. Since there are two sets of DACs, the total power dissipated by them is 4×50uW, which is 200uW.

The noises of the DACs are not an issue because the thermal noise coming from the resistors are negligible when compared to the V_{LSB}s of the DACs, which are on the order of mVs.

5.3.2 Speed

The only challenge associated with the DAC is the speed. As discussed in Chapter 3, small switches are used for the multiplexers in the DAC decoders. In addition, a tree topology decoder is used for the 8-bit DAC. Simulation shows that both DACs are fast enough to meet the design requirement. To illustrate this, the worst case scenario is presented here.

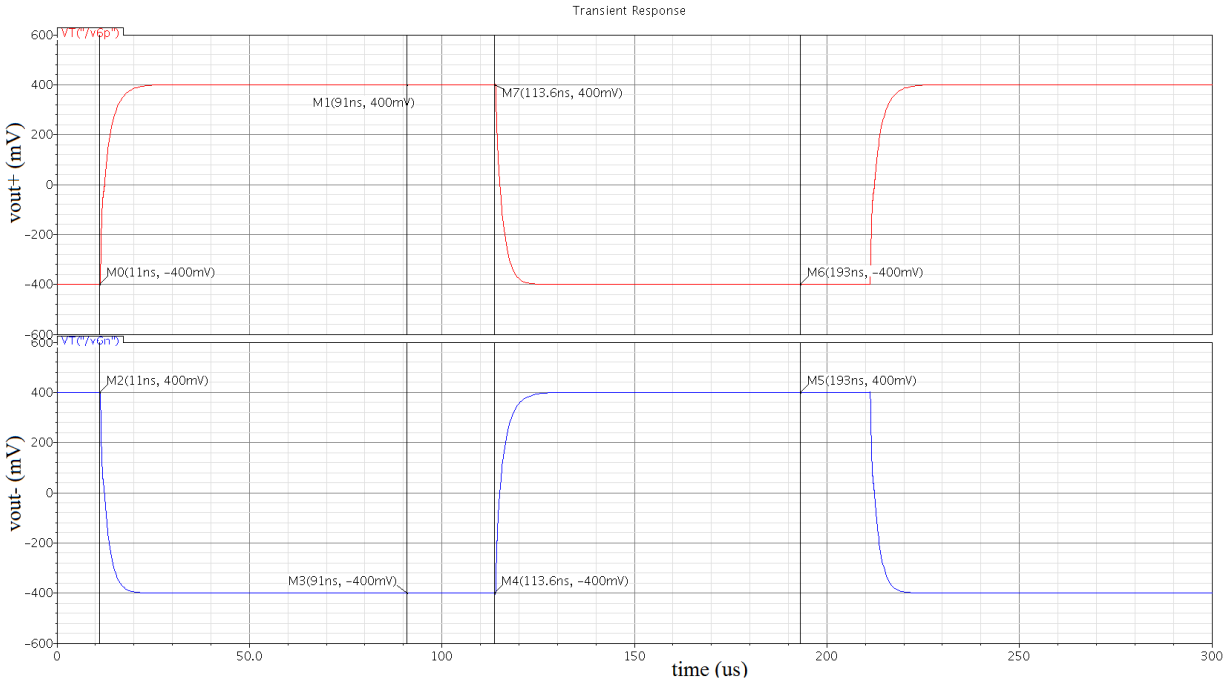


Fig.5.6 Six-bit DAC Transient Response

For both DACs, the largest RC delay they can experience is when the DAC output is changing from -0.4V to 0.4V or vice versa, since the middle of the resistor string is tied to ground. Fig.5.6 and Fig.5.7 show the DAC outputs under this scenario. As can be seen, both DACs can swing to their final values within 80ns, which as explained earlier, is the time period before the latched comparator is fired during conversion process.

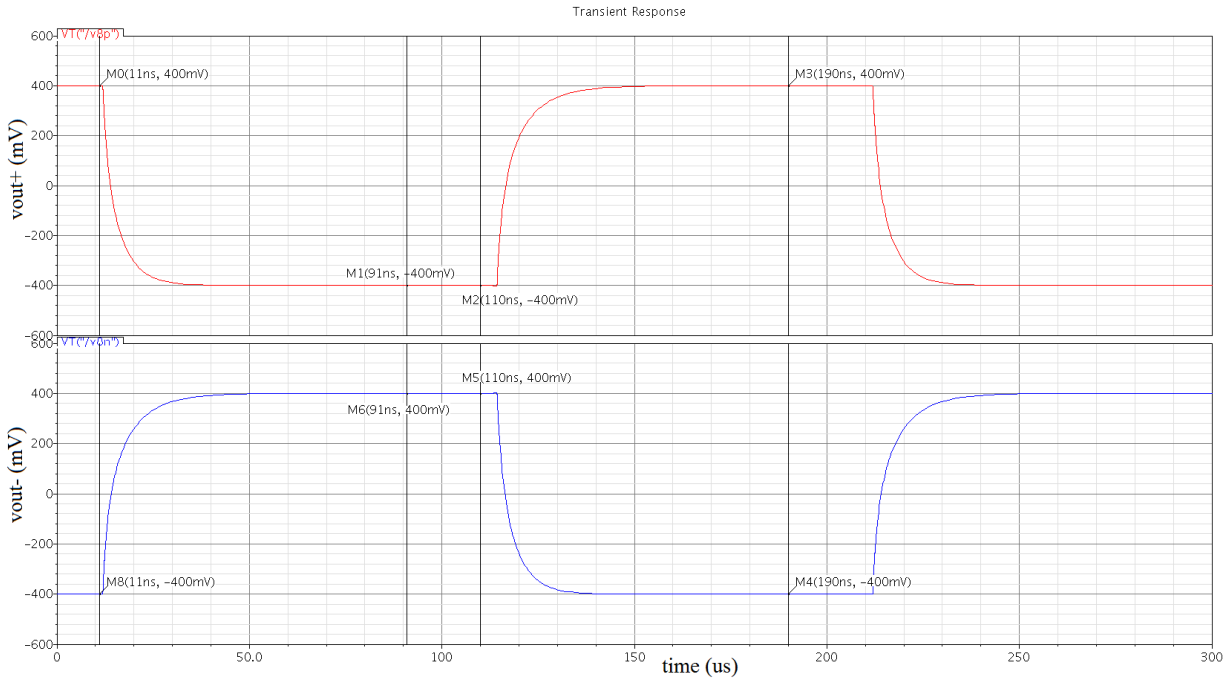


Fig.5.7 Eight-bit DAC Transient Response

5.4 ADC Performance

The ADC performance metrics, including power, noise, signal-to-noise-and-distortion-ratio (SNDR), effective number of bits (ENOB), and so on, are provided in this section.

5.4.1 Power

The ADC power is calculated by adding the power of the Input Buffer, the Comparator and the DACs. As mentioned earlier in this chapter, the power dissipations from these three components are 3.65mW, 0.64mW and 0.2mW respectively. Therefore the total power of the ADC is

$$P_{ADC,\,total} = 3.65mW + 0.64mW + 0.2mW = 4.49mW$$

The power of the ADC not including the input buffer is 0.84mW. As can be seen, Input Buffer consumes most of the ADC power, thus future effort should be focused on reducing its power in order to reduce the ADC power.

5.4.2 Noise

The total noise of the ADC system, including the Input Buffer, can be calculated as the RMS sum of the noise from its components as equation (5.1) shows [9].

$$V_{n,\,tot} = \sqrt{\left(\frac{kT}{C_N}\right) + \left(\frac{kT}{C_P}\right) + (V_{n,\,comp})^2 + (V_{n,\,ib})^2} \tag{5.1}$$

Where C_N and C_P are the total capacitance of the negative and positive array, $V_{n,\,comp}$ and $V_{n,\,ib}$ are the noise from the Comparator and the Input Buffer respectively.

In this way, the total noise is calculated to be

$$V_{n,\,tot} = \sqrt{\left(\frac{4.142\times10^{-21}}{50\times10^{-12}}\right) + \left(\frac{4.142\times10^{-21}}{50\times10^{-12}}\right) + (11\times10^{-6})^2 + (18.6\times10^{-6})^2 V} = 25.1uV$$

On the other hand, the noise of the Input Buffer is usually excluded from ADC noise calculation. In this way, the ADC noise can be reevaluated by

$$V_{n,\,tot,\,ADC} = \sqrt{\left(\frac{kT}{C_N}\right) + \left(\frac{kT}{C_P}\right) + (V_{n,\,comp})^2} \tag{5.2}$$

Then calculation gives

$$V_{n,\,tot,\,ADC} = \sqrt{\left(\frac{4.142\times10^{-21}}{50\times10^{-12}}\right) + \left(\frac{4.142\times10^{-21}}{50\times10^{-12}}\right) + (11\times10^{-6})^2 V} = 16.9uV$$

Where $V_{n,tot,ADC}$ is higher than the ideal value of 12uV (1/4 V_{LSB}). The capacitor array total capacitance can be increased to reduce the thermal noise from the switches.

5.4.3 Other Metrics

One of the very important metrics for evaluating ADC performance is the Signal to Noise and Distortion Ratio (SNDR). SNDR is defined as the ratio of the RMS value of the input signal to that of noise and distortion of the ADC combined. To calculate SNDR, the Total Harmonic Distortion (THD) and the Quantization Error of the ADC need to be calculated first.

A full scale sinusoid input wave of frequency 15.625K is applied to the ADC, and the converted digital outputs are fed into an ideal 16 bit DAC. Fig.5.8 shows the output of the DAC.

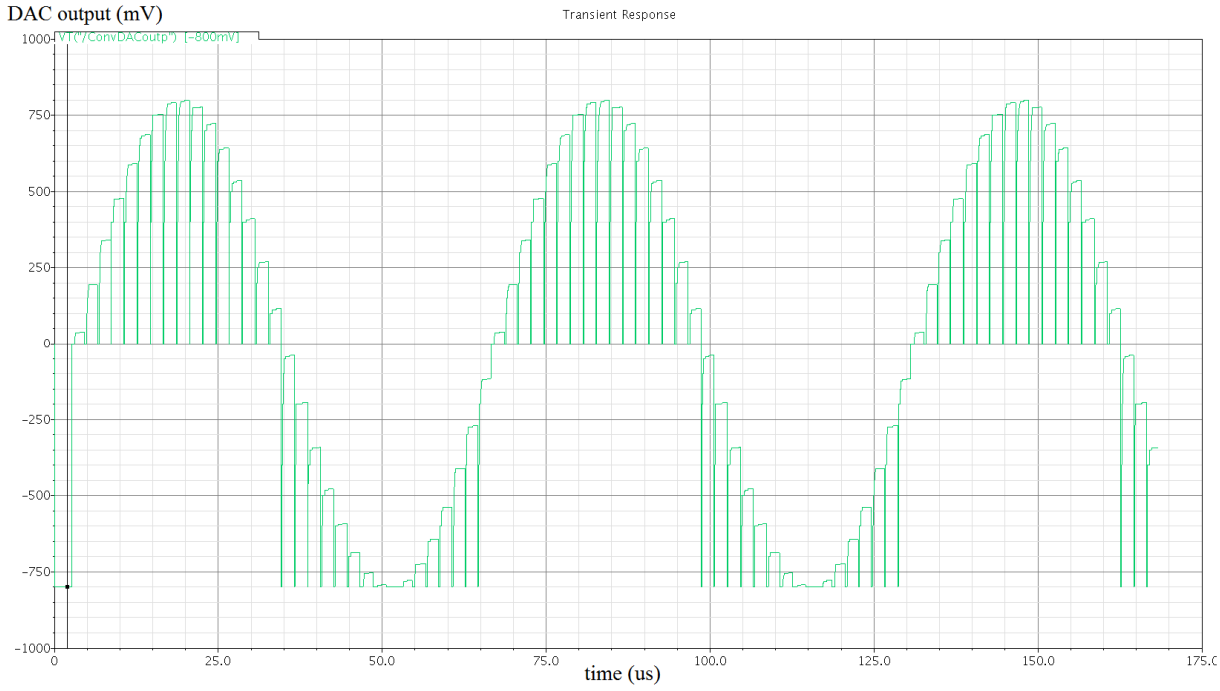


Fig.5.8 Output of the 16 bit DAC

64 DAC output samples are taken and used to compare with the ADC input to study the ADC's SNDR. First, the differences between the ADC inputs and the DAC outputs are calculated, as shown in Fig.5.9.

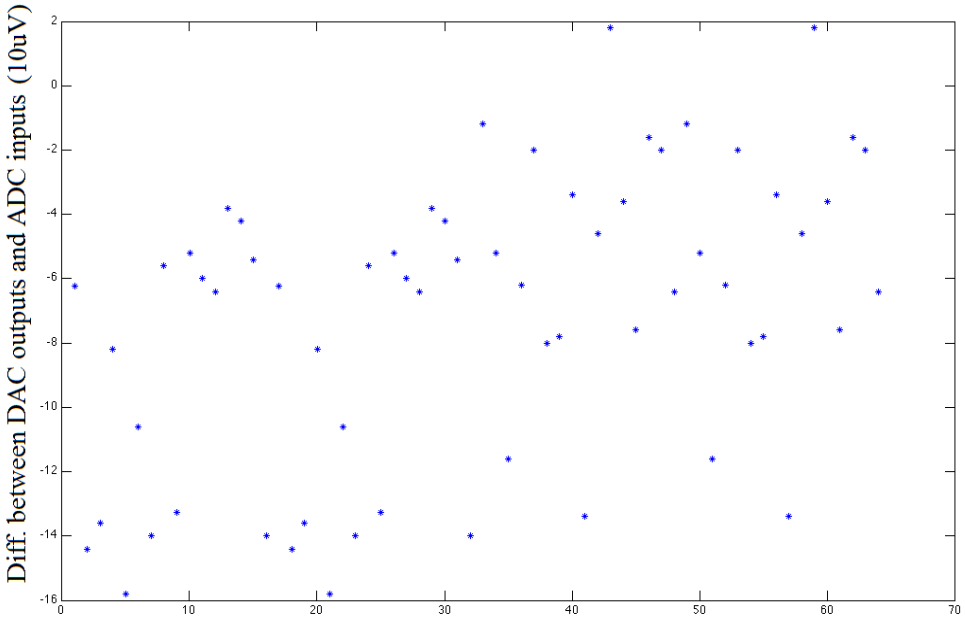


Fig.5.9 Difference between ADC input and DAC output

The standard deviation of the differences V_Q (in other words, Quantization Error) is calculated to be 45.2uV. This is close to V_{LSB} (48uV), indicating non-linearity in the ADC transfer function.

Discrete Fourier Transform is performed on the 64 output samples to study the distortion of the ADC, and is shown in Fig.5.10. Since the ADC is fully differential, even order harmonics of the input signals are suppressed. Their values are lower much lower than -125dB, and thus do not show in the plot in Fig.5.10.

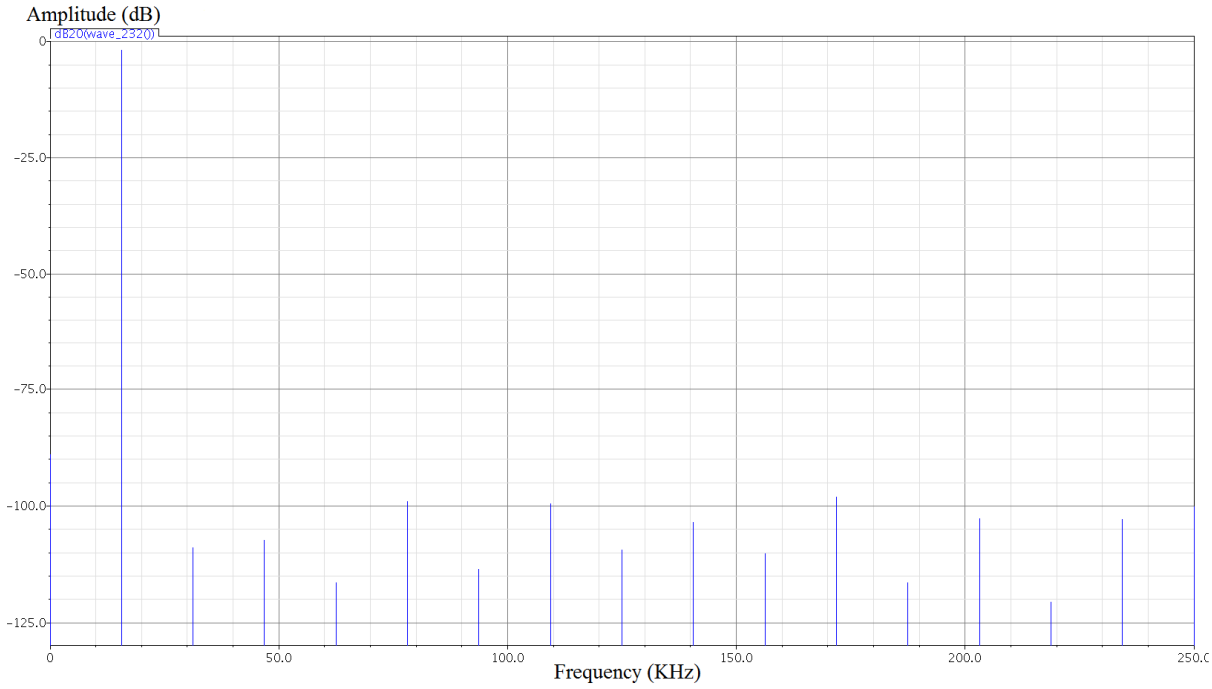


Fig.5.10 Output Spectrum of the ADC

The SNDR of the ADC is given by

$$SNDR = 20\log(\frac{V_{in,RMS}}{\sqrt{(V_Q)^2 + (V_N)^2}}) = 20\log(\frac{1.6/\sqrt{2}}{\sqrt{(45.2uV)^2 + (16.9uV)^2}}) = 87.4dB$$

where V_Q and V_N are the quantization error and ADC noise respectively.

The ideal SNDR of a SAR ADC is equal to 6.02N + 1.76dB, where N is the number of bit [1]. For 16 bits, ideal SNDR is 98.08dB. The SNDR calculated above is lower than the ideal value, caused by noise and non-linearity of the ADC.

The ENOB of the ADC is associated with its SNDR by [9]

$$ENOB = \frac{SNDR_{db} - 1.76}{6.02} = \frac{87.4 - 1.76}{6.02} = 14.22$$

The FOM of the ADC is given by [9]

$$FOM = \frac{P_{ADC}}{2 \times 2^{ENOB} \times BW} = \frac{0.84mW}{2 \times 2^{14.22} \times 500K} = 44.0\, fJ / Conv.Step$$

On the other hand, if the power of the input buffer is included, than

$$FOM' = \frac{P_{TOT}}{2 \times 2^{ENOB} \times BW} = \frac{4.59mW}{2 \times 2^{14.22} \times 500K} = 240.5\, fJ / Conv.Step$$

ENOB is less than 16, again due to noise and distortion of the ADC. On the other hand, FOM is comparable to state-of-the-art ADC because shutdown of comparators and interleaving are introduced into the ADC to save power dissipation.

6. CONCLUSION

A Successive Approximation Charge Redistribution Analog to Digital Converter is designed in JAZZ 0.18um SiGe CMOS process using only CMOS devices. Self Calibration Circuitry is designed to remove the Capacitor Ratio, Offset and Gain Error of the ADC. Low Power Comparator with Shutdown mode is designed to reduce power consumption of the ADC. An input Buffer with moderate power, high speed, high accuracy and low noise is designed to drive the ADC. Interleaving Architecture is implemented to lower power dissipation.

The ADC operates from a power supply of ± 1V, reference voltage of ± 0.8V and a clock rate of 10MHz. The power dissipation is 4.59mW. The noise of the ADC is 16.9uV without the Input Buffer and 21.5uV with the Input Buffer. The SNDR and ENOB of the ADC are 87.4dB and 14.22 bits. The FOM with and without input buffer is 240.5fJ/Conv. Step and 44.0fJ/Conv. Step respectively.

Overall the ADC works well according to simulation results. But verification is needed after the chip is fabricated. The major power consumption and noise of the ADC comes from the Input Buffer, so future effort can be directed toward the design of the Input Buffer. Also, there is non-linearity in the ADC transfer function, which results in degraded SNDR and ENOB values. Further study should be conducted to solve this problem to achieve better performance.

BIBLIOGRAPHY

1. D.Johns, K.Martin, Analog Integrated Circuit Design, Toronto: Johns Wiley & Sons, Inc, 1997.
2. H.Guo, et al, “A Low-power 16-bit 500 kS/s ADC,” 2005 IEEE Workshop on Microelectronics and Electron Devices, pp.84 – 87, Apr. 2005.
3. H.S.Lee, et al, “A Self-Calibration 15-bit CMOS A/D Converter,” IEEE J. Solid- State Circuits, Vol. SC-19, pp. 813-819, Dec. 1984.
4. K.S.Tan et al, “Error Correction Techniques For High-Performance Differential A/D Converters,” IEEE J. Solid-State Circuits, Vol. 25, no. 6, pp. 1318-1327, Dec.1990.
5. J.H.Huijsing et al, “Low Voltage Operational Amplifier With Rail-To-Rail Input And Output Stages,” IEEE J. Solid-State Circuits, Vol. SC-20, no. 6, pp. 1144-1150, Dec.1985.
6. G.Ferri et al, “A Rail-To-Rail Constant Gm Low-Voltage CMOS Transconductance Amplifier,” IEEE J. Solid-State Circuits, Vol. 32, no. 10, pp. 1563-1567, Oct.1997.
7. R.Hogervost et al, “A Compact Power-Efficient 3-V CMOS Rail-To-Rail Input/Output Operational Amplifier For VLSI Cell Libraries,” IEEE J. Solid-State Circuits, Vol. 29, no. 12, pp. 1505-1513, Dec.1994.
8. H.Guo, “ A Low Power Low-Noise High Accuracy Sensor IC,” Doctoral Dissertation, Department of Electrical Engineering and Computer Science, Washington State University, Pullman, Washington, the USA, 2006.

9. P.Upadhyaya, “A Self-Calibrating Low Power 16-BIT 500KSps Charge Redistribution SAR Analog-To-Digital Converter,” Master Thesis, Department of Electrical Engineering and Computer Science, Washington State University, Pullman WA, USA, 2008.

APPENDIX

1. Capacitor Ratio Calibration Logic

```
sCreCalc1:
      begin
            if ( (result == 16'h00FF) | (result == 16'h0000) )
            begin
                  nextState     <= sSample;
                  nextCount     <= 5'b00110;
                  next_sw_int <= 16'b0111111111111111;
                  nextBitDone <= 16'b0111111111111111;
                  nextTempReg <= 16'b0;
                  nextCreSamp <= 1'b1;

                  if (!ABswitch)
                  nextIncCREcap1 <= 1'b1;
                  else
                  nextIncCREcap2 <= 1'b1;

                  if ( CreOftIter != 2'b00)
                  begin
                  nextOftRef      <= 1'b1;
                  nextOftGnd     <= 1'b0;
                  end

            end

            else
            begin
                  nextState <= sCreCalc2;
                        next_sw_int    <= 16'b0;
                  if (result[7]) begin
                        nextResult   <= {1'b0, result[6:0]};
                        nextError <= {1'b0, result[6:0]} - accError;
                        end
                        else begin
                        nextResult   <= 8'b10000000 + result;
                        nextError <= 8'b10000000 + result - accError;
                        end
            end
      end


sCreCalc2:     // Calculate the digital Error and update the accumulated error
      begin
              nextState      <= sCreCalc3;
              nextError     <= result - accError;
              nextResult   <= 16'b0;
```

```
            if (Error[7])
                sgnBit <= 1'b1;
            else
                sgnBit <= 1'b0;
        end

                                                                                    //
sCreCalc3:   // Divide by 2 to get the digital error
        begin
            nextState   <= sCreCalc4;
               nextError     <= {sgnBit, Error[7:1]};

            if (!ABswitch)                  // Bit Error for the First Cap Array
            begin
             if (bitDone[15:14] == 2'b01) bit_addr <= 0;
             if (bitDone[14:13] == 2'b01) bit_addr <= 1;
             if (bitDone[13:12] == 2'b01) bit_addr <= 2;
             if (bitDone[12:11] == 2'b01) bit_addr <= 3;
             if (bitDone[11:10] == 2'b01) bit_addr <= 4;
             if (bitDone[10:9]   == 2'b01) bit_addr <= 5;
             if (bitDone[9:8]    == 2'b01) bit_addr <= 6;
             if (bitDone[8:7]    == 2'b01) bit_addr <= 7;
             if (bitDone[7:6]    == 2'b01) bit_addr <= 8;
             if (bitDone[6:5]    == 2'b01) bit_addr <= 9;
            end

            else                            //   Bit Error for the Second Cap Array
            begin
             if (bitDone[15:14] == 2'b01) bit_addr <= 10;
             if (bitDone[14:13] == 2'b01) bit_addr <= 11;
             if (bitDone[13:12] == 2'b01) bit_addr <= 12;
             if (bitDone[12:11] == 2'b01) bit_addr <= 13;
             if (bitDone[11:10] == 2'b01) bit_addr <= 14;
             if (bitDone[10:9]   == 2'b01) bit_addr <= 15;
             if (bitDone[9:8]    == 2'b01) bit_addr <= 16;
             if (bitDone[8:7]    == 2'b01) bit_addr <= 17;
             if (bitDone[7:6]    == 2'b01) bit_addr <= 18;
             if (bitDone[6:5]    == 2'b01) bit_addr <= 19;
            end

            bit_error_we <= 1'b1;

        end
//                                                                                  //
        sCreCalc4:
        begin
            bit_error_we <= 1'b0;
            if(bitDone[6])
            begin
                    nextState    <= sSample;
```

```
            nextCount     <= 5'b00110;
            next_sw_int <= bitDone >> 1;
            nextBitDone <= bitDone >> 1;
            nextError     <= 8'b0;
            nextAccError<= Error + accError;
            nextCreSamp <= 1'b1;
            nextSwTop1     <= 1'b1;
            nextSwTop2     <= 1'b1;

            if ( CreOftIter != 2'b00)
            begin
            nextOftRef      <= 1'b1;
            nextOftGnd     <= 1'b0;
            end

        end

        else
        begin
            nextState          <= sSample;
            nextSwTop1          <= 1'b1; // For the top plates to be VGNDed during sample
            nextSwTop2          <= 1'b1;
            nextCount          <= 5'b00111;
            next_sw_int       <= 16'b1111111111111111;
            nextBitDone <= 16'b0;
            nextTempReg <= 16'b0;
            nextSmid <= 1'b1;

            if(!ABswitch)
            begin
            nextOftPos1         <= 1'b0;
            nextOftNeg1         <= 1'b0;
            end
            else
            begin
            nextOftPos2         <= 1'b0;
            nextOftNeg2         <= 1'b0;
            end

            nextCreCal      <= 1'b0;
            nextOftCal      <= 1'b1;   // Test : 1'b0;
            nextOftGnd     <= 1'b0;
            nextOftRef      <= 1'b1;
            nextCalCount  <= 3'b001;
        end

    end
```

2. Offset Calibration Logic

```
sOftCalc:
        begin
          nextState <= sSample;
          nextOftGnd   <= 1'b0;
          nextOftRef   <= 1'b1;

          if (calCount == 3'b001)
          begin

              next_sw_int <= 16'b1111111111111111;

              if (result == 16'h8000)
              begin
                  nextOftCal      <= 1'b0;
                  nextVrefnSig      <= 1'b1; // Sample Vrefn during Sampling
                  nextCmpsw     <= 1'b0;
                  nextTempReg       <= 11'h0;
                  nextCMEparaCrt   <=1'b1;
                  nextCMEcnt        <= 1'b1;
                  nextSwTop1         <= 1'b0;
                  nextSwTop2         <= 1'b0;
                  nextCount         <= 5'b00111;
                  nextCalsw         <= 1'b1;
              end

              else
              begin
                  nextTempReg <= 16'b0000000100000000;
                  nextSwTop1         <= 1'b1; // For the top plates to be VGNDed during sample
                  nextSwTop2         <= 1'b1;
                  nextSmid <= 1'b1;
                  nextCount     <= 5'b00110;
                  nextCalCount <= 3'b0;
                  nextCalsw         <= 1'b0;

                  if (result > 16'b1000000000000000)
                  begin
                  if (!ABswitch)
                  nextOftNeg1   <= 1'b1;
                  else
                  nextOftNeg2   <= 1'b1;
                  end

                  else        // if (result < 16'b1000000000000000)
                  begin
                  if (!ABswitch)
                  nextOftPos1    <= 1'b1;
                  else
                  nextOftPos2    <= 1'b1;
```

```
            end
        end

end


else
begin      // begin of if calcount != 001
        if ( (tempReg[0]) | (result == 16'h8000) )
        begin
            nextOftCal     <= 1'b0;
            nextResult     <= 16'b0;
            nextCount          <= 5'b00111;

            if (!ABswitch)
            nextOftCtrlF1 <= (bitDone | tempReg);
            else
            nextOftCtrlF2 <= (bitDone | tempReg);

            if ( CreOftIter == 2'b10)
            begin
            nextCreOftIter   <= 2'b0;
            nextVrefnSig      <= 1'b1; // Sample Vrefn during Sampling (bypass ResDAC)
            nextCmpsw     <= 1'b0;
            nextTempReg       <= 11'h0;
            nextCMEparaCrt   <=1'b1;
            nextCMEcnt <= 1'b1;
            next_sw_int       <= 16'b1111111111111111;
            nextSwTop1          <= 1'b0; // For the top plates to be VGNDed during sample
            nextSwTop2          <= 1'b0;
            end

            else
            begin
            nextCreOftIter <= CreOftIter + 1'b1;
            next_sw_int <= 16'b0111111111111111; // CRE calibration
            nextBitDone <= 16'b0111111111111111;
            nextTempReg <= 16'b0;
            nextError     <= 8'b0;
            nextAccError<= 8'b0;
            nextCreCal   <= 1'b1;
            nextCreSamp <= 1'b1;
            nextSwTop1          <= 1'b1; // For the top plates to be VGNDed during sample
            nextSwTop2          <= 1'b1;
            end
        end

        else if (   (oftNeg1 & (!ABswitch))   | (oftNeg2 & ABswitch) )
        begin
            nextTempReg <= tempReg>>1;
```

```
                nextSwTop1          <= 1'b1; // For the top plates to be VGNDed during sample
                nextSwTop2          <= 1'b1;
                nextSmid      <= 1'b1;
                next_sw_int <= 16'b1111111111111111;
                nextCount     <= 5'b00110;
                if (result > 16'h8000)
                nextBitDone <= (bitDone | tempReg);
            end   //   end if (oftneg)

            else if (   (oftPos1 & (!ABswitch))   | (oftPos2 & ABswitch) )
            begin      // begin of if (oftpos)
                nextTempReg <= tempReg>>1;
                nextSwTop1          <= 1'b1; // For the top plates to be VGNDed during sample
                nextSwTop2          <= 1'b1;
                nextSmid      <= 1'b1;
                next_sw_int <= 16'b1111111111111111;
                nextCount     <= 5'b00110;
                if (result < 16'h8000)
                nextBitDone <= (bitDone | tempReg);
            end

        end   // end of if (calcount != 001)

    end
```

3. Gain Error Calibration Logic

```
if(sw_int[15] & CMEparaCrt)
            begin
            nextConv <= 1'b0;

            if (CMEcnt==1)
            begin
                nextCMEcnt <= CMEcnt-1;
                nextTempReg <= 11'h400;
                nextState <= sSample;
                nextCount   <= 5'b00111;
                next_sw_int <= 16'b1111111111111111;
                nextVrefnSig<= 1'b1;
                nextCmpsw     <= 1'b0;

                if (cmpout)
                begin
                    if(!ABswitch) begin
                    nextCMEparaPos1 <= 1'b1;    // Add more capacitance to pos
                    nextCMEparaNeg1 <= 1'b0;
                    end
                    else   begin
```

```
            nextCMEparaPos2 <= 1'b1;    // Add more capacitance to pos
            nextCMEparaNeg2 <= 1'b0;
            end
        end

        else
        begin
            if(!ABswitch) begin
            nextCMEparaNeg1 <= 1'b1;    // Add more capacitance to neg
            nextCMEparaPos1 <= 1'b0;
            end
            else    begin
            nextCMEparaNeg2 <= 1'b1;    // Add more capacitance to neg
            nextCMEparaPos2 <= 1'b0;
            end
        end
end


else    // if CMEcnt=0,
begin
        if(tempReg[0])
        begin
            nextState       <= sDoNot;
            nextCMEparaCrt    <= 1'b0;
            nextCMEparaBD    <= 11'b0;
            nextSSDcal        <= 1'b1;
            if (ABswitch)
            nextCalibrationDone <= 1'b1;
            else
            nextABswitch    <= 1'b1;

            if (cmpout) begin
                if (   ( CMEparaPos1 & (!ABswitch)) | (CMEparaPos2 & ABswitch) )
        begin
                    if (!ABswitch)
                    nextCMEparaCtrlF1 <= tempReg | CMEparaBD;
                    else
                    nextCMEparaCtrlF2 <= tempReg | CMEparaBD;
                end

                else     begin
                    if (!ABswitch)
                    nextCMEparaCtrlF1 <= CMEparaBD;
                    else
                    nextCMEparaCtrlF2 <= CMEparaBD;
                end
              end

            else begin
                if (   (CMEparaNeg1 & (!ABswitch)) | ( CMEparaNeg2 & ABswitch) )      begin
```

```
                        if (!ABswitch)
                        nextCMEparaCtrlF1 <= tempReg | CMEparaBD;
                        else
                        nextCMEparaCtrlF2 <= tempReg | CMEparaBD;
                    end

                    else        begin
                        if (!ABswitch)
                        nextCMEparaCtrlF1 <= CMEparaBD;
                        else
                        nextCMEparaCtrlF2 <= CMEparaBD;
                    end

                end
            end

            else
            begin
                nextState <= sSample;
                nextTempReg <= tempReg>>1;
                nextCount          <= 5'b00111;
                next_sw_int <= 16'b1111111111111111;
                    nextVrefnSig<= 1'b1;
                nextCmpsw     <= 1'b0;
                if (cmpout)     begin
                    if (   ( CMEparaPos1 & (!ABswitch)) | (CMEparaPos2 & ABswitch) )
                    nextCMEparaBD <= tempReg | CMEparaBD;
                    else
                    nextCMEparaBD <= CMEparaBD;

                end

                else  begin
                    if (   (CMEparaNeg1 & (!ABswitch)) | ( CMEparaNeg2 & ABswitch) )
                    nextCMEparaBD <= tempReg | CMEparaBD;
                    else
                    nextCMEparaBD <= CMEparaBD;
                end

            end

        end
    end
```

4. Sample

```
sSample :
        begin
        if(count <= 5'b0) begin
```

```
        nextResult   <= 16'b0;
        nextState    <= sConv;
        nextSample   <= 1'b0;
        nextConv<= 1'b1;
        nextCmpsw    <= 1'b1;

    if (!normal) begin
        nextSfine       <= 1'b1;
        nextCount        <= 5'b0;
        nextCalsw           <= 1'b1;

        if (creCal) begin
         nextCreCal   <= creCal;
         nextCreSamp <= 1'b0;
         nextCalDone <= 1'b0;
         nextCalReg   <= sw_int ^~ tempReg;
         next_sw_int <= 16'b0000_0000_1000_0000;
         nextError    <= 8'b0;
         nextAccError<= accError;

         if ( CreOftIter != 2'b00)
         begin
         nextOftRef     <= 1'b0;
         nextOftGnd    <= 1'b1;
         end

         else
         begin
         nextOftGnd    <= 1'b0;
         nextOftRef     <= 1'b0;
         end
        end

        else if (CMEparaCrt) begin
         nextCalDone <= 1'b1;
         nextVrefnSig<= 1'b0;
         next_sw_int <= 16'b1000000000000000;
         nextCalReg   <= calReg;
         nextAccError<= 8'b0;
         nextOftGnd    <= 1'b1;
         nextOftRef     <= 1'b0;

        end

        else if (oftCal) begin
         nextSmid      <= 1'b0;
         nextOftGnd   <= 1'b1;
         nextOftRef   <= 1'b0;
         nextCalDone <= 1'b1;
         next_sw_int <= 16'b1000000000000000;
         nextAccError<= 8'b0;
```

```
        end
        else nextState <= sDoNot;

    end

    else begin       // Normal Conversion Turn on the Coarse Comparator

        next_sw_int    <= 16'b1000000000000000;
        nextABswitch   <= ~ABswitch;
        nextin_slew <= 1'b1;
        nextClatch          <= 1'b1;
        nextCalsw    <= 1'b0;
        nextCalDone     <= 1'b1;
        nextScoarse         <= 1'b1;
        nextSfine     <= 1'b0;
        nextAccError<= 8'b0;   //
        nextOftRef       <= 1'b0;
        nextOftGnd       <= 1'b1;

    end
//                          ****                              //
  end


  else   begin
         nextState     <= sSample;
         nextCount     <= count - 1;
         nextCmpsw     <= 1'b0;
         nextConv      <= 1'b0;

         if (CMEparaCrt)   begin
         nextSwTop1     <= 1'b0;
         nextSwTop2     <= 1'b0;
         end
         else if (count == 5'b00001)   begin
         nextSwTop1     <= 1'b0;
               if(!normal)
               nextSwTop2     <= 1'b0;
         end

         else     begin
         nextSwTop1     <= 1'b1;
         nextSwTop2     <= 1'b1;
         end

         if (!normal)
               begin
               nextCalsw <= 1'b0;
               nextSSDcal     <= 1'b0;
```

```
        if (count == 5'b00001) begin
              nextFresetB1 <= 1'b0;
              nextFresetB2 <= 1'b0;
        end

        else begin
              nextFresetB1 <= 1'b1;
              nextFresetB2 <= 1'b1;
        end

        if (creCal) begin
        nextError     <= Error;
        nextAccError<= accError;
              if ( CreOftIter != 2'b00)
              begin
              nextOftRef      <= 1'b1;
              nextOftGnd     <= 1'b0;
              end
              else
              begin
              nextOftGnd     <= 1'b0;
              nextOftRef      <= 1'b0;
              end
        end

        else if (CMEparaCrt) begin
        nextOftRef      <= 1'b1;
        nextOftGnd     <= 1'b0;
        end

        else if (oftCal) begin
        nextOftRef      <= 1'b1;
        nextOftGnd     <= 1'b0;
        end

        else nextState <= sDoNot;

        end


  else begin
        nextSSDcal     <= 1'b1;
        nextCalsw       <= 1'b0;
        nextFresetB1   <= 1'b1;
        nextOftGnd     <= 1'b0;    //
        nextOftRef      <= 1'b1;

        if (count == 5'b00010)
        nextSDcoarse1       <= 1'b0;

        if (count == 5'b00001)
```

```
                nextCresetB1  <= 1'b0;
                else
                nextCresetB1  <= 1'b1;

                if (count == 5'b01010)
                nextin_slew <= 1'b0;

            end
        end
    end
```

5. Conversion

```
sConv :
    begin      // state 6
        nextCmpsw    <= 1'b1;

        if (sw_int[15] & oftCal)
            next_sw_int          <= 16'h0080;      // Oftset Calibration only uses the last 8 bits for results,
        else
        next_sw_int          <= sw_int>>1;

        if (sw_int[2] & normal & ABswitch)
        nextSDcoarse1 <= 1'b0;

        if (sw_int[2] & normal & (!ABswitch))
        nextSDcoarse2 <= 1'b0;

        if (sw_int[1] & normal & ABswitch)
        begin
        nextCresetB1  <= 1'b0;
        nextSwTop1    <= 1'b0;
        end

        if (sw_int[1] & normal & (!ABswitch))
        begin
        nextCresetB2  <= 1'b0;
        nextSwTop2    <= 1'b0;
        end

        if (sw_int[0] & normal)
        begin
        nextSDfine1 <= 1'b1;
        nextSDfine2 <= 1'b1;
        nextFresetB1 <= 1'b1;
        nextFresetB2 <= 1'b1;
        end

        if (creCal) begin
```

```
            nextTempReg <= tempReg | calReg;
            end

// Resolve hysteresis during Normal Conversion
            if (sw_int[9] & normal)
            begin

                nextState   <= sConv;
                // nextCmpsw   <= 1'b1;

                if (!ABswitch)
                 nextSDfine1 <= 1'b0;
                else
                 nextSDfine2 <= 1'b0;

                if (cmpout) begin
                        nextResult   <= result | sw_int;
                    end
                    else begin
                        nextResult   <= result;
                    end

            end


            else if (sw_int[8] & normal)
            begin
                nextState        <= sConv2;
                next_sw_int <= 16'b0;
                nextDecD       <= 1'b0;
                nextCount      <= 5'b00001;
                nextConv<= 1'b1;

                nextClatch      <= 1'b0;
                nextFlatch      <= 1'b0;
                nextCalDone     <= 1'b1;
                nextSfine       <= 1'b1;
                nextScoarse     <= 1'b1;

                if (!ABswitch)
                nextFresetB1 <= 1'b0;
                else
                nextFresetB2 <= 1'b0;

                if (cmpout) begin
                    nextResult   <= result | sw_int;
                    nextD          <= result | sw_int;
                    nextAccError<=   (accError + bitErr);
                end

                else begin
```

```
                nextResult <= result;
                nextD <= result;
                nextAccError<= accError;
            end
        end


// CRE calibration
        else if (sw_int[0] & creCal)
        begin
            nextState   <= sCreCalc1;
            nextConv  <= 1'b0;
            nextCreCal <= creCal;

            if (cmpout)
            nextResult   <= (result | sw_int);
            else nextResult <= result;
        end


// Offset Calibration
        else if (sw_int[0] & oftCal)
        begin   // offset calibration
            nextState   <= sOftCalc;
            nextConv  <= 1'b0;
            nextOftCal       <= oftCal;
            nextAccError      <= 8'b0;
             if (cmpout)
             nextResult   <= result | sw_int;
             else nextResult <= result;
        end


// Normal Conversion
        else if(sw_int[0]) begin
            if (convert) begin
             nextState          <= sConv;
             nextABswitch         <= ~ABswitch;
             next_sw_int     <= 16'b1000000000000000;
             nextCount          <= 5'b10011;
             nextAccError      <= 8'b0;
             nextD              <= 16'b0;
             nextScoarse       <= 1'b1;
             nextSfine    <= 1'b0;
             nextClatch         <= 1'b1;
             nextFlatch         <= 1'b0;
             nextCalDone          <= 1'b1;
             nextOftRef        <= ~oftRef;

             if (!ABswitch)
             nextSwTop1   <= 1'b1;
```

```
                else
                nextSwTop2   <= 1'b1;

                nextin_slew       <= 1'b1;

                if (cmpout)
                        nextResult   <= result | sw_int;
                else
                        nextResult <= result;
                end


        else begin
                nextState     <= sDoNot;
                next_sw_int <= 16'b0;
                if (cmpout)
                nextResult   <= result | sw_int;
                else
                nextResult       <= result;
                end

        end

// Bit cycle and get result
        else begin
                nextState        <= sConv;

                if (cmpout) begin
                        if (sw_int[15] )
                        nextResult   <= 16'h8000;
                        else
                        nextResult   <= result | sw_int;


                        if (creCal) nextAccError <= accError;   // only during CRE calibration
                        else nextAccError<=   (accError + bitErr);
                        end

                else begin
                        if (sw_int[15] & oftCal)
                        nextResult   <= 16'h7f00;
                        else if (sw_int[15] )
                        nextResult   <= 16'h0000;
                        else nextResult   <= result;

                        nextAccError<= accError;
                end

        end
    end
```

```
sConv2:
      begin      // state 7
            nextCalDone <= 1'b1;
            nextCalReg   <= calReg;//
            nextCalCount<= calCount;
            nextTempReg <= tempReg;
            nextBitDone <= bitDone;
            nextSfine    <= 1'b1;
            nextCmpsw    <= 1'b1;
            nextConv     <= 1'b1;
            nextAccError<= accError;
                        //
            if (count == 5'b00000)
            begin

                  nextin_slew       <= 1'b0;
                  nextCount <= 5'b00001;

                  if (result == 16'b1111111100000000) begin
                   nextD      <= result;
                  end
                  else begin
                   nextD      <= {D[15:8] + 1'b1, 8'b0};
                  end

                  nextResult       <= result;
                  nextSfine    <= 1'b1;
                  next_sw_int     <= {result[15:8] - 1'b1, 8'b0}; //****

                  if (cmpout)
                        nextDecD <= 1'b1;
                  else
                        nextDecD <= 1'b0;

                  nextState <= sConv3;
                  nextFlatch <= 1'b0;
                  nextSDcoarse1    <= 1'b1;
                  nextSDcoarse2    <= 1'b1;
                  nextCresetB1     <= 1'b1;
                  nextCresetB2     <= 1'b1;
            end

            else
            begin
                  nextCount        <= count - 1'b1;
                  nextState  <= sConv2;
                  nextD          <= D;
                  nextD2    <=   {D[15:8] - 1'b1, 8'b0};
                  nextResult<= result;
                  nextFlatch <= 1'b1;
```

```
            nextScoarse    <= 1'b0;

        end

    end


sConv3:
    begin      // state 8
        nextCalDone <= 1'b1;
        nextSwTop1    <= 1'b0;
        nextSwTop2    <= 1'b0;
        nextCalReg   <= calReg;
        nextCalCount<= calCount;
        nextTempReg <= tempReg;
        nextBitDone <= bitDone;
        nextCmpsw          <= 1'b1;
        nextSfine       <= 1'b1;
        nextConv<= 1'b1;


        if (count == 5'b00000)
        begin
            nextState <= sConv;
            nextD   <= 16'b0;
            nextCount <= 5'b0;
            nextFlatch <= 1'b1;
            next_sw_int       <= 16'b0000000010000000;
            if (!cmpout & !dec_D) begin

             if (result == 16'b0) begin
                 nextResult        <= result;
                 nextAccError      <= accError;
             end
             else begin
                 nextResult   <= {result[15:8] - 1'b1, 8'b0};
                 nextAccError <= DErr;
                  end
               end
            else if (cmpout & dec_D) begin

             if (result == 16'b1111111100000000) begin
                 nextResult        <= result;
                 nextAccError      <= accError;
             end
             else begin
                 nextResult   <= {result[15:8] + 1'b1, 8'b0};
                 nextAccError<= totErr
             end
            end
```

```
            else begin
              nextResult   <= result;
              nextAccError<= accError;
            end
        end

        else
        begin
            nextCount     <= count - 1'b1;
                    nextState     <= sConv3;
                    nextD          <= D;
                    nextResult   <= result;
            nextDecD <= dec_D;
            nextFlatch   <= 1'b1;
            next_sw_int <= sw_int;
        end

end
```