NTNU

Norwegian University of
Science and Technology

Design of an Analog to Digital Converter with Superior Accuracy/Bandwidth vs. Power Ratio

Kjetil Kvalø

Master of Science in Electronics
Submission date: July 2011
Supervisor: Trond Ytterdal, IET
Co-supervisor: Johnny Bjørnsen, Energy micro / Analog concepts

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Task description

Most of today's microcontrollers contain a general purpose analog to digital converter (ADC) used for audio conversion, battery monitoring and/or sensor input. Traditionally, successive approximation ADC's have been popular due to their scalability and simplicity, but with the always increasing demand for accuracy, speed and lower power consumption, a new and more power-efficient design must be considered.

One way to achieve high performance with low power consumption is by utilizing an ADC which can be customized for the particular application and utilize the minimal power required for the specific performance needed. Another way is to reduce the overall conversion time and duty-cycle the ADC system. On module level, each of the sub-modules should be power optimized individually e.g. by utilizing low power techniques and thoroughly designing each module.

The assignment can be divided into four parts:

- Do a literature study on latest state-of-the-art in SAR ADC architectures and performance.
- Based on the previous Energy Micro SAR ADC, find ways to improve current consumption.
- Build a high level behavioral model of the system, investigate potential sources of errors and evaluate design trade-offs and required sub-module performance.
- Implement the ADC system by designing critical sub-modules at transistor level and verify the overall top-level ADC performance as a whole with SPICE simulations. As a simplification, use a SPICE model for the reference generator.

Abstract

The objective of this thesis was to design a power-efficient general purpose SAR ADC. The ADC's requirements were set by Energy Micro, favoring a very high performance-to-power ratio. The requirements are based on the present Energy Micro ADC, but with a 67% reduction in current consumption, a more modern CMOS technology of 90nm and a supply voltage of 1.2V.

A full SAR ADC model was made using SPICE and VHDL code for the analog and digital sub-systems, respectively. The comparator was thoroughly designed and optimized, to achieve enough performance with as little power as possible. Then the total capacitor value of the sub-DAC was minimized, using extra reference voltages, minimizing the dynamic power consumption of the reference voltage generator. An asynchronous clock was also implemented, substantially increasing the available settling times of the comparator.

The result was a very power-efficient SAR ADC, which fulfills the power-consumption requirement with 114μJ per conversion. Compared to other, similar SAR ADC's which has been researched, the ADC designed in this thesis is found to be very power-efficient. There might be some linearity problems in the ADC, partly from the transmission gates used as switches, but the overall design seems promising.

Acknowledgment

The work presented in this thesis was carried out during the spring of 2011, under the supervision of Johnny Bjørnsen at Analog concepts and Trond Ytterdal at NTNU. I would especially like to thank Johnny Bjørnsen for his initiative in the making of this thesis, as well as Trond Ytterdal for several years of mentoring. I would also like to thank Carolina Fiorella Velezmoro, Daniel Aasbø, Marius Volstad and Torbjørn Løvseth for the good company at room A177.

Contents

List of Figures

List of Tables

1 Introduction

General purpose analog-to-digital converters, such as the one designed in this thesis, can be used for audio conversion, battery monitoring and sensor inputs. Many of these operations must be implemented on portable, battery powered devices, creating a need for low-power high-performance ADC's.

Several approaches can be used in designing a power-efficient ADC, such as decreasing the conversion time and duty-cycling ADC or carefully designing each of the sub-modules. The designer always needs to keep in mind the requirements and application of the ADC, utilizing no extra power for unnecessary performance.

In this thesis, a 90nm 1.2V CMOS technology has been used, while other designs that are used as inspirational sources often have a somewhat older technology[7][9]. Using newer technology enables the designer to use smaller transistors, saving both current and area, while increasing the available bandwidth[10].

Using lower supply voltage, smaller transistors and low-power techniques from several sources[3][11][7], this design will be optimized for current consumption, while achieving the goals given by the thesis assignment:

Table 1: SAR ADC target parameters

Parameter	Value
Accuracy	12bit
Sampling frequency	1MHz
Current consumption	125μA

2 Theory

This chapter covers the basics of analog-to-digital conversion and the SAR ADC. It also contains relevant theoretical equations and descriptions.

2.1 Analog to digital conversion (ADC)

Analog-to-digital converters (ADC's) are often used to collect data from sensors, such as touchscreens, thermometers, camera image sensors and battery meters. To do this, the ADC measures the voltage or current input, and outputs a string of bits, which represents the input voltage or current.

The analog-to-digital conversion is done in two stages, sampling and quantization. The sampling of an analog signal is done at regular intervals, T_s, to ensure periodicity in the frequency spectrum[2]. The input frequencies are then mirrored and repeated around the sampling frequencies[2]. From figure 1, it is clear that the signal bands will overlap if the input frequency exceeds two times the sampling frequency, severely reducing the quality of the output signal. This effect is known as the Nyquist theorem[2]. According to Johns&Martin[2], one must make sure that the input frequencies is below this Nyquist frequency limit, $f_s/2$. One could either use a low-pass filter (anti-aliasing filter, or AAF) before the sampling circuit, or design the sampling circuit itself to become a low-pass filter. A steeper filter will enable the input signal to approach the Nyquist limit. The SAR ADC does not have such a filter incorporated in the design, so an AAF must be inserted in front of the ADC.

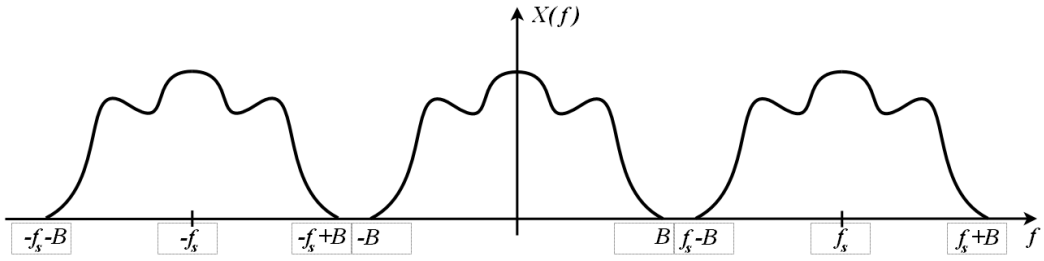

Figure 1: Non-overlapping spectrum[1]

2.2 The SAR ADC

The successive approximation ADC (SAR ADC) architecture employs a 1-bit "binary search algorithm" in a feedback loop to extract a digital value from a analog signal[2], as shown in figure 2. The most popular SAR ADC, the "charge redistribution SAR ADC", uses capacitors to store and modify the input signal[2]. Using this method, the capacitors are discharged and the voltage in the DAC-array can be divided by a power-of-two, as shown in section 2.4.

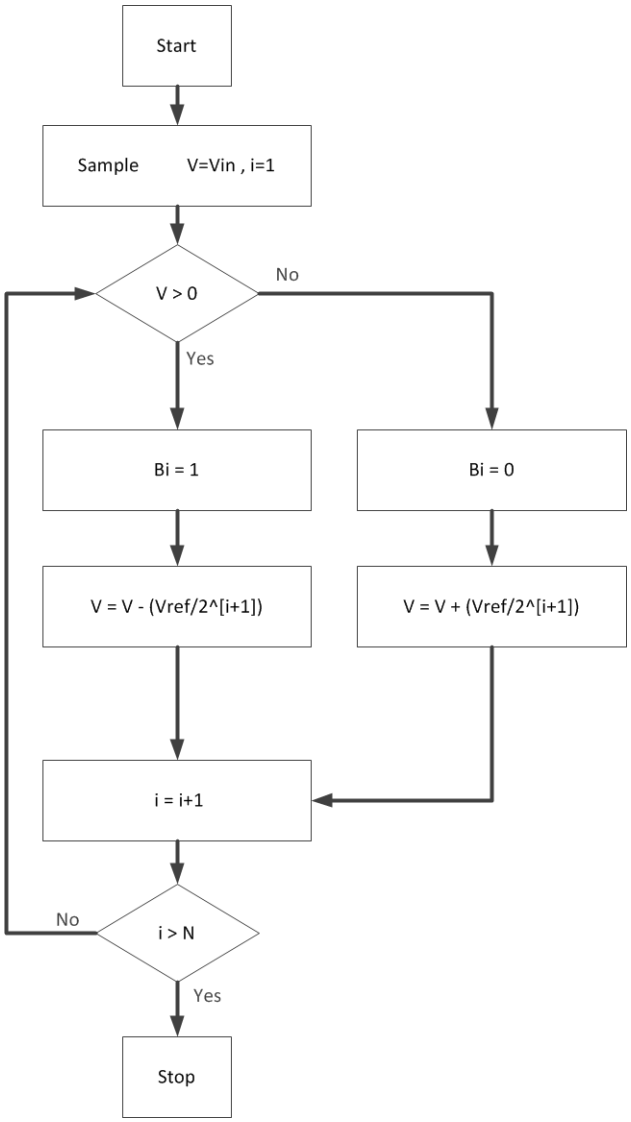

Figure 2: SAR ADC flowchart[2]

The SAR ADC architecture generally consists of a sample-and-hold circuit (S/H), a digital-to-analog capacitor array (sDAC), a comparator and digital logic, and is controlled by a clock, running at N+1 times the sampling frequency, where N is the bit resolution and the extra period is for sampling the input signal[2]. The different parts are shown in figure 3.

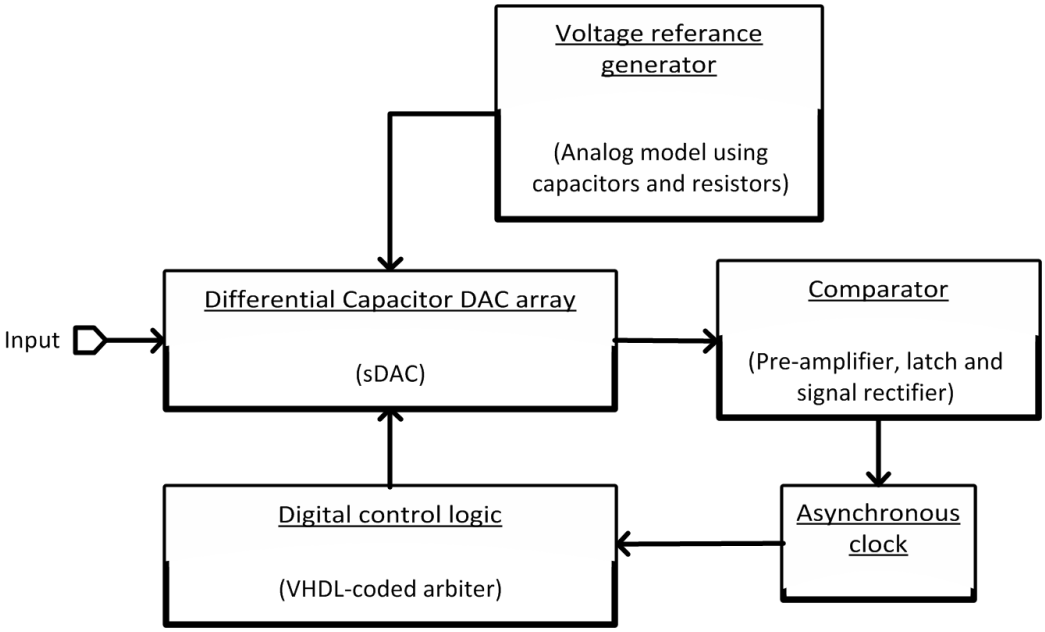

Figure 3: Sar Overview[3]

2.3 Components of a charge redistribution SAR ADC

A SAR ADC consists of four main components[2]; a sub-DAC, a comparator, a reference generator and digital control logic, as shown in figure 3. Those four components will be reviewed in this section.

2.3.1 Sub-DAC

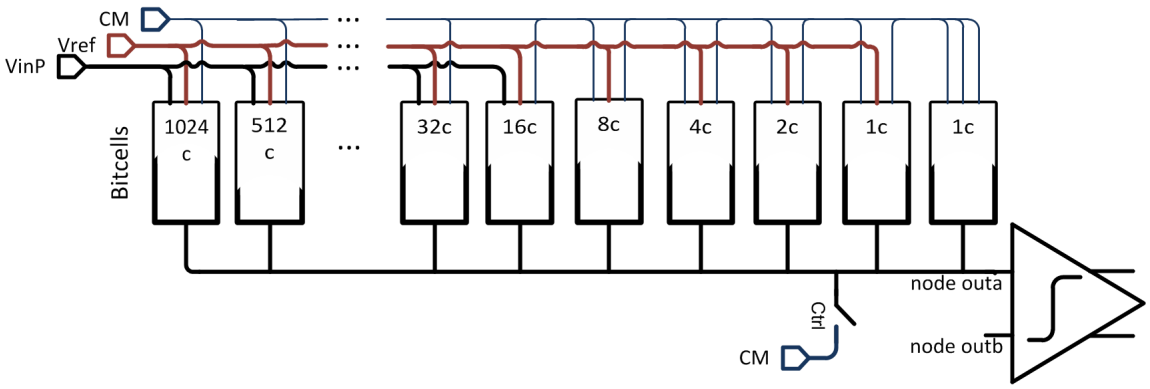

Figure 4: Sub-DAC overview, one half of the differential sub-DAC[3], with bitcells containing capacitors and swithces

The differential sub-digital-to-analog (sDAC) consists of two capacitor arrays, as well as including a sample-and-hold functionality. The capacitors is incorporated in the bitcells, as described in section 4.2. Basically, each bitcell in the capacitor array contains a capacitor with a power-of-two capacitance value, together with switches to connect the capacitance to the input voltage, common mode voltage or a reference voltage.

The sDAC first samples the input signal, uses the comparator to determine the MSB and then modifies the signal, based on the digital input from the digital control logic, as shown in figure 2. The sDAC uses capacitors to divide the differential signal by two, resulting in new voltages that are compared in the comparator.

In addition to the power-of-two capacitances, the arrays will have some parasitic capacitance, limiting the accuracy of the ADC. The complete figure of the sDAC can be found in the appendix, figure 21. The details on voltage modification in the sDAC is reviewed in section 2.4.

2.3.2 Comparator

A differential comparator is used to determine which of the two capacitor arrays have the highest voltage. The comparator will have a high voltage output (vdd) if the positive input has higher voltage than the negative input. In the case the negative input is larger, the output of the comparator will be low (vss). To achieve this, a latch is used.

The standard clocked latch has two phases, a tracking phase and a latching phase. The first mentioned means the latch is locked, but ready to be latched one way or another. The latching phase is normally initiated by the clock going high. The latch will latch to a output, and will not change before the clock changes and the latch is set back to tracking mode.

A latch often has a large offset, so the signal will need amplification before the latch[3]. Pre-amplifiers are used for this, often with several amplifiers in succession. A high gain amplifier not only minimizes offset, but also minimizes input related random noise and kickback noise[2], see sections 2.6.2 and 2.6.4 . The first stage of the first preamplifier will be critical to noise and offset requirements, since the signal will be amplified after this stage, and the noise of the later stages will be suppressed by this amount of gain[4], as shown in equation 1 and illustrated in figure 5.

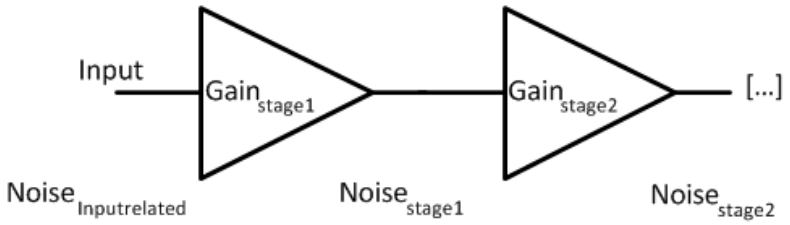

Figure 5: Noise summation through gain stages[4]

$$Noise_{inputrelated} = \frac{Noise_{stage1}}{Gain_{stage1}} + \frac{Noise_{stage2}}{Gain_{stage1} \cdot Gain_{stage2}} + ... \quad (1)$$

To further minimize noise, pmos transistors can be used as input transistors[2]. This is because p-channel MOSFET's carriers (holes) are less likely to be trapped in the the transistor channel, compared to an n-channel MOSFET[2].

At the output of the latch, the signal will mostly be high(vdd) or low(vss), but may also be somewhere in between when the latching process is working,

early in the latching phase. This is evident in figure 6, which is a simulation of the comparator in section 4.3. To shape the signal to a more correct digital signal, signal shapers are used. A much used architecture of a signal shaper is a inverter, made up of a pmos and a nmos transistor, amplifying the signal and effectively increasing the time the signal is either high(vdd) or low(vss).

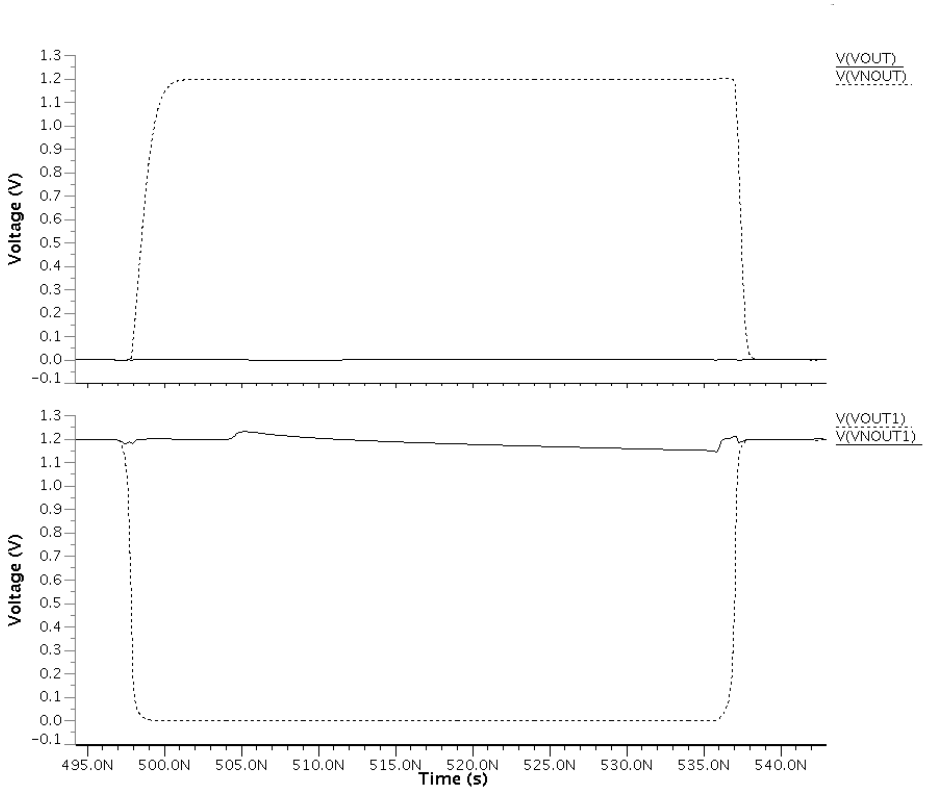

Figure 6: The effect of the signal rectifier in the circuit in section 4.3
Top: Output from signal rectifier, Output(VOUT) and inverse output(VNOUT)
Bottom: Output from latch, Output(VOUT1) and inverse output(VNOUT1)

2.3.3 Digital control logic

The digital logic's tasks includes saving the output bits and controlling the sDAC. This logic usually has a input clock signal and a set of output control signals. In addition, it may include an asynchronous clock input, if an asynchronous clock is implemented in the design. The asynchronous clock will detect a decision from the comparator and start the next phase

prematurely, allowing the voltages of that next phase more time to settle. In this design, the asynchronous clock is made up of three digital logic gates, an inverter, a XOR gate and an AND gate. The digital logic may be designed as a shift register[3].

2.3.4 Reference circuit

The voltage reference circuit generates stable voltages to the SAR ADC. A shifting reference voltage will affect the charge transferred in the sDAC, again shifting the differential voltages, as seen in equation 6 in section 2.4.

2.4 Theoretical model of an ideal sub-DAC

The voltage modifications of the capacitive arrays, is made possible by an array of capacitances. The capacitor number "n" in the array, has a capacitor value of 2^n times the unity capacitor size. Each of these capacitors are connected to switches, switching the capacitor input between voltages V_{in}, V_{ref} and V_{cm}. An extra unit capacitor is added to the array, so that the LSB capacitance switches the right amount of charge, relative to the total charge of the capacitors. The sDAC is illustrated in figure 7.

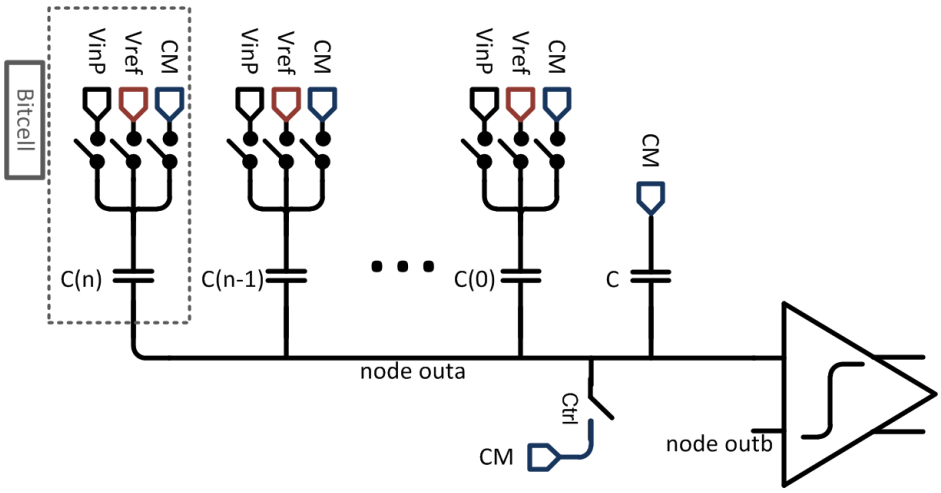

Figure 7: One half of the sub-DAC, for illustration of the equations of section 2.4 and 2.5

When sampling, initially the central node of the positive capacitor array, outa, is connected to V_{cm}, and the capacitors to V_{in}. Then the node outa is disconnected from V_{cm}, becoming a very-high-impedance node, able to store charge. At the same time, the capacitances are switched to V_{cm}. By the law of charge conservation[2], the charge in node outa is conserved:

$$\begin{aligned} Q_{pre-transfer} &= C_{in} \cdot (V_{cm} - V_{inp}) \\ Q_{post-transfer} &= C_{in} \cdot (V_a - V_{cm}) \end{aligned} \tag{2}$$

Using the law of charge conservation[2], the voltage in node outa will be:

$$\begin{aligned} Q_{pre-transfer} &= Q_{post-transfer} \\ (2C_{in}) \cdot V_{cm} &= C_{in} \cdot V_{inp} + (C_{in}) \cdot V_{inp} \\ V_a &= \frac{2C_{in}}{C_{in}} \cdot V_{cm} - \frac{C_{in}}{C_{in}} \cdot V_{inp} \\ V_a &= 2 \cdot V_{cm} - V_{inp} \end{aligned} \tag{3}$$

When switching a bitcell with capacitor value C_x, the differential voltage will be reduced. The equations for charge conservation will be[2]:

$$\begin{aligned} Q_{pre-transfer} &= C_{in} \cdot (V_{a,pre} - V_{cm})) \\ Q_{post-transfer} &= (C_{in} - C_x) \cdot (V_{a,post} - V_{cm}) \\ &\quad + (C_x) \cdot (V_{a,post} - V_{ref}) \end{aligned} \tag{4}$$

Resulting in a new voltage at node outa, by charge conservation[2]:

$$\begin{aligned} Q_{pre-transfer} &= Q_{post-transfer} \\ (C_{in} + C_x - C_x) \cdot V_{a,post} &= (-C_{in} + C_{in} - C_x) \cdot V_{cm} \\ &\quad + (C_{in}) \cdot V_{a,pre} + C_x \cdot V_{ref} \\ &\Downarrow \\ V_{a,post} &= V_{a,pre} + \frac{C_x}{C_{in}} \cdot (V_{ref} - V_{cm}) \end{aligned} \tag{5}$$

If we look at the differential voltage, nodes outa minus outb, we can see the absolute voltage is reduced by $V_{ref}/2^n$, where n is the bitcell number, with the MSB bitcell being number 1. This is since the capacitance of the n'th bitcell ideally is 2^n of the total capacitance. The equation below has $V_{refn} - V_{refp}$, assuming a positive V_{inp}. Should $V_{inp} \leq V_{inn}$, the references will be opposite.

$$V_{a,post} - V_{b,post} = V_{a,pre} - V_{b,pre} - \frac{C_x}{C_{in}} \cdot (V_{refn} - V_{refp}) \tag{6}$$

2.5 Theoretical model of a non-ideal sub-DAC

The design used in this thesis, reviewed in section 4.1, not only has parasittic capacitance, but also bitcells with capacitances that is not sampled towards V_{in}. In the equations below, these capacitances are summed and called C_p, where the rest of the capacitances are called C_{in}. The equations differs somewhat from the ideal equations in section 2.4. The biggest difference is in the sampling, where the absolute voltage in node outa will be somewhat lower then the input voltage.

When sampling, the resulting voltage in node outa will be decided by charge conservation[2]:

$$\begin{aligned} Q_{pre-transfer} &= C_{in} \cdot (V_{cm} - V_{inp}) + C_p \cdot (V_{cm} - V_{cm}) \\ Q_{post-transfer} &= C_{in} \cdot (V_a - V_{cm}) + C_p \cdot (V_a - V_{cm}) \end{aligned} \tag{7}$$

Using charge conservation:

$$\begin{aligned} Q_{pre-transfer} &= Q_{post-transfer} \\ (C_p + 2C_{in}) \cdot V_{cm} &= C_{in} \cdot V_{inp} + (C_{in} + C_p) \cdot V_{inp} \\ V_a &= \frac{2C_{in} + C_p}{C_{in} + C_p} \cdot V_{cm} - \frac{C_{in}}{C_{in} + C_p} \cdot V_{inp} \end{aligned} \tag{8}$$

When switching a bitcell with capacitor value C_x, the equations will be:

$$\begin{aligned} Q_{pre-transfer} &= C_{in} \cdot (V_{a,pre} - V_{cm}) + C_p \cdot (V_{a,pre} - V_{cm}) \\ Q_{post-transfer} &= (C_{in} - C_x) \cdot (V_{a,post} - V_{cm}) \\ &\quad + (C_x) \cdot (V_{a,post} - V_{ref}) + C_p \cdot (V_a - V_{cm}) \end{aligned} \tag{9}$$

Again using the law of charge conservation[2]:

$$\begin{aligned} Q_{pre-transfer} &= Q_{post-transfer} \\ (C_{in} + C_x - C_x + C_p) \cdot V_{a,post} &= (-C_p + C_p - C_{in} + C_{in} - C_x) \cdot V_{cm} \\ &\quad + (C_{in} + C_p) \cdot V_{a,pre} + C_x \cdot V_{ref} \\ &\Downarrow \\ V_{a,post} &= V_{a,pre} + \frac{C_x}{C_{in} + C_p} \cdot (V_{ref} - V_{cm}) \end{aligned} \tag{10}$$

Resulting in a differential voltage, given $V_{inp} \geq V_{inn}$:

$$V_{a,post} - V_{b,post} = V_{a,pre} - V_{b,pre} - \frac{C_x}{C_{in} + C_p} \cdot (V_{refn} - V_{refp}) \tag{11}$$

As the in section 2.4, the references will be opposite if $V_{inp} \leq V_{inn}$.

2.6 Sources of performance degradation

Performance will be limited by a range of error sources, the most important of which will be discussed here. Some sources of error may be minimized by a good design or by using special architectures.

2.6.1 Process variations and mismatch

Process variation may introduce offsets and non-linearities in the SAR ADC. The effect of process variations may be predicted using a Monte Carlo simulation, offsetting circuit element parameters according to a chosen probability distribution[12], as well as parameters set in the library of the circuit element. The effects can generally be reduced by increasing the area of key circuit elements, using careful layout or changing to another foundry or technology. Offsets in the output of the sDAC, or in other critical nodes, is usually minimized using active offset cancellation circuits[3].

In this design, offsets in reference voltage and capacitor values will directly influence the voltages at the output of the sDAC(input of the comparator). The voltage error can be found, using the equations in section 2.5, especially equation 11.

2.6.2 Mosfet random error sources

According to [13], there are 5 noise sources in mosfet devices:

- Thermal noise in the drain-source channel
- Flicker noise
- Noise in the resistive poly gate
- Noise due to the distributed substrate resistance
- Shotnoise associated with the the leakage current of the drain-source reverse diodes

Of these 5 noise sources, the first two are most important, while the rest only need to be considered at very-low-noise designs. The information in the next two paragraphs are obtained from [13].

The channel thermal noise[13] in a mosfet transistor is caused by thermal movement of the electrons between the drain and source connections. This movement causes a statistically fluctuating signal between the drain and source, which results in a thermal noise current. It should be noted that the thermal noise is independent on frequency.

For a mosfet in the linear region ($V_{DS} < (V_{GS}\text{-}V_T)$), the estimated thermal noise current will be as in equation 12, with g_0 equaling channel conductance at zero V_{DS}.

$$I_{n,thermal,lin} = 4 \cdot k \cdot T \cdot g_0 \tag{12}$$

In a mosfet in the saturated region ($V_{DS} \geq (V_{GS}\text{-}V_T)$), the noise can be estimated to:

$$I_{n,thermal,sat} = 4 \cdot k \cdot T \cdot \frac{2}{3} \cdot gm \tag{13}$$

Flicker noise[13] , or 1/f-noise, has been observed in all kinds of devices, but the mechanism behind this noise is still to be discovered. The mosfet transistor generates a relatively large amount of flicker noise, because of the surface conduction mechanism. There are several theory's and models to explain the flicker noise in a mosfet, but they are mainly based on the Hooge empirical relation and the fluctuation model introduced by McWhorter [13][14]. Experiments show that the equation that is most correct, may be one based on the number fluctuation model, as shown in equation 14[13]. This equation was used in early versions of SPICE[14]. Johns & Martin[2] uses the same equation, only with C_{ox} instead of C_{ox}^2. However, another source[14] suggests both formulas to be oversimplified.

$$V_f^2(f) = \frac{K_f}{C_{ox}^2 \cdot W \cdot L \cdot f} \tag{14}$$

It should be noted that random noise will mainly be a problem in the outa node in the sDAC, which is the input to the comparator. After this stage, the signal will be amplified and the random noise sources will be smaller in comparison, as illustrated in section 2.3.2.

2.6.3 Sampling error

The SAR ADC has a built-in sample-and-hold in the sub-DAC. As mentioned in section 2.1, the sampling of the input signal must be done at regular

intervals to avoid a drop in performance. Clock jitter or MOSFET non-linearities will introduce a sampling time uncertainty, τ. When sampling a sine wave, $Asin(2\pi f_{in}t)$, the error will be equal to $\tau \cdot dV_{in}/dt$. It should also be noted that, if τ is assumed uncorrelated with V_{in}, the noise power from this error will be [15] :

$$P_e = 2\pi^2 f_{in}^2 A^2 \tau_{rms}^2 \tag{15}$$

2.6.4 Kickback noise

Kickback denotes the charge transfer either into or out of the inputs of the latch, when the latch goes from tracking mode to latching mode[2]. Without a preamplifier of buffer, this kickback may cause the driving circuit to have very large glitches[2].

2.6.5 Charge injection error

According to Johns & Martin[2], charge injection, or clock feedthrough, is unwanted charges injected into a circuit node when the transistor turns off. The charge error occurs by two mechanisms. First by the channel charge flowing out from the channel region of the transistor to the drain and source junctions. Secondly, a generally much smaller charge occurs due to the overlap capacitance between the gate and the other junctions.

The channel charge of a transistor with zero V_{ds} is given by[2]:

$$Q_{ch} = WLC_{ox}V_{eff} = WLC_{ox}(V_{gs} - V_t) \tag{16}$$

If the gate control signal is fast and the nodes equal in voltage and impedance, the charge can (ideally) be said to divide equally between the drain- and source nodes[2]. Where low-impedance nodes only suffers a temporary glitch when receiving channel charge, the high-impedance nodes will store that extra charge, causing a voltage offset[2].

2.6.6 Hysteresis

If the comparator toggles in one direction, the comparator may tend to toggle in the same direction the next time[2]. This is called hysteresis. Resetting the comparator every clock cycle, as shown in the comparator in section 4.3, will erase the "memory" that comparators may have, eliminating the problem[2].

3 SAR ADC Architecture

The SAR ADC is mainly made up of a sDAC, a comparator, a reference generator and digital logic[2][3]. This project will concentrate on the sDAC and comparator, as well as making use of some design techniques like asynchronous clocking. The reference generator will be represented with a simple model and the digital logic coded in VHDL, and not optimized for current consumption or area.

3.1 Capacitor array

The standard capacitor array, as shown in figure 4, uses a very large area and draws a good deal of current from the voltage reference generator. This is because of a very large total capacitance size of 2048 times the unit capacitor size, equaling to about 50pF for each of the arrays[1]. Several methods can be used to reduce the capacitor sizes or decrease current throughput. Some of them will be reviewed in this section.

One possible way to reduce the total capacitance, according to Trond Ytterdal[11], is to use C2C scaling. For use in a sDAC, the lowest possible capacitance array consists of unity capacitors C1-C2-C3-...-C11[2], with double unity capacitors in between, resulting in a minimum of 31c pr array, equaling about 0.75pF, somewhat lower than the capacitance needed to achieve the noise limitA.4. The biggest challenge with this architecture is the parasitic capacitance from the unity capacitors that is connected with both nodes to the capacitor array. Without special design solutions, this results in massive decrease in linearities and accuracy[5].

To reduce the nonlinearities, both of the capacitor arrays may be split into two sub-arrays each. Both sub-arrays would be standard capacitor arrays, as illustrated in section 2.3.1. In between, a attenuation capacitor would divide the two arrays, minimizing spread in capacitor values. This method is used by Yan Chu[16] and makes the designer able to balance linearity requirements against the total capacitance value. By this method, and by the simulations by Yan Chu[16], a 12bit capacitive array would have 256c

[1]Using a unity capacitance of 24.6fF, the smallest capacitance available in the design, as shown in section A.2

[2]The tag "Cx" is used, where the "C" stands for one unity capacitor and the "x" is the capacitor index number

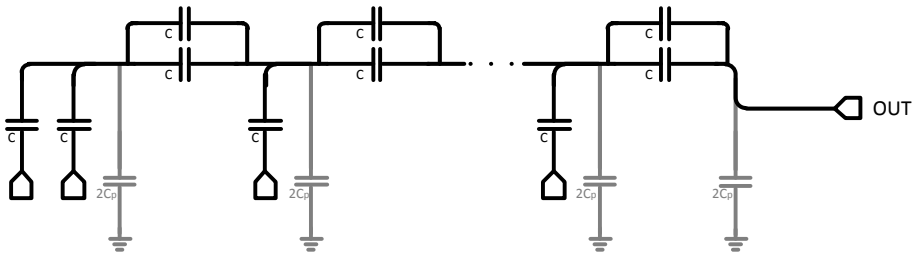

Figure 8: Standard C2C array[5]

(MSB's) + 8c (LSB's) + C_{att}, equaling to about 7pF.

To increase accuracy, one may split up the capacitor arrays into two completely seperate parts, effectively pipelining the capacitor array. In the example of Hung-Wei's paper[9], the ADC first resolves the first 5 MSB's, using 6 clock cycles and an extra bit for error correction. The next 7 bits are resolved more accurately, using 14 clock cycles and an extra amplifier. The result is a 12bit SAR ADC with 380fJ/conversion, a very good result.

Yan Chu's design[16], in addition to containing a split capacitor array, also incorporated a charge recycling method. When switching the voltage over the capacitors in the capacitor array, the current is normally "wasted". Reusing some of this current is possible, and Chu's design saves as much as 90% current[3], compared to a standard design.

Lastly, another way of increasing settling time, and as a result, accuracy or sampling speed, is to use asynchronous logic, as in Wenbo Liu's paper[7]. This is detailed further in section 3.4.

3.2 Comparator

The comparator might be the most important sub-circuit in the SAR ADC. To archieve good accuracy, the comparator must have high enough gain, low enough offset and low noise as well. All this must be balanced against bandwidth and current consumption.

Goll and Zimmermann presents a latch[8] with very good bandwidth and low offset. The latch is designed to handle supply voltages as low as 0.5V at

[3]This number includes the split capacitor array, as mentioned above

400Mhz, with a offset of 21.2mV, a current consumption of 18μW and still achieves a BER of 10^{-9}[17].

The offset of a latch will usually be much higher then required for 12bit accuracy, so one or more preamplifiers must be used. Wenbo Liu uses one preamp in his paper[7], giving a gain of 30dB through 2 sub-stages. The preamp is designed with pmos input transistors to minimize 1/f noise and limit noise bandwidth.

3.3 Error correction

Wenbo Liu[7] uses a perturbation-based calibration system, where the the signal is sampled twice with 14 raw bits and converted to a finished 12bit code. The first raw sample is added with a offset $+\Delta_a$, while the second sample is added with an opposite offset $-\Delta_a$. These two signals are resolved to D_+ and D_- and converted by weighted sum of the individual bits, to d_+ and d_-. The output of the ADC is then made up of ($d_+ + d_- = 2 \cdot$ output code). A second signal, (d_+ - d_- - $2\Delta_a$) is fed back to the ADC. Ideally this signal is zero, hence a nonzero signal will provide information to correct the ad conversion.

This calibration method improves SNDR from 60.15dB to 70.72dB. In total, the SAR ADC from this paper achieves a state-of-the-art result of 45.6 and 31.4 fJ/conversion at 22.5 and 45MS/s, respectively. If capacitor mismatch is the dominant error source, the calibration can be turned off after the bit weights are learned, leading to a doubling in the ADC's sampling speed. An extra SNR boost comes from the double conversion during the calibration, where both quantization noise and comparator noise is reduces by 3dB, effectively increasing the systems SNR.

3.4 Asynchronous clock

An asynchronous clock may be used after the comparator, detecting when a decision is made, instead of waiting for the global clock signal. Using this, the control logic may switch the next capacitor earlier, generally increasing the available settling- and decision time for the comparator, possibly increasing the SAR ADC's accuracy.

A possible design solution is used in an article by Wenbo Liu[7], using a

NOR-port to find if the comparator has settled, together with a AND-port to make sure the asynchronous clock (ACLK) does not go high when the global clock is low.

A known problem with asynchronous clocks are that they are vulnerable to comparator metastability problems. In the article mentioned above, the problem is solved by making the global clock's falling edge force the SAR logic to latch 0, continuing as the metastability issue never occurred.

4 Methods and design

4.1 Capacitance reduced array

The SAR capacitive array, as shown in figure 4, can be improved by reducing the total capacitance down to a level set by noise requirements, see section A.4. To do this, the capacitor values of the smallest bitcells[4] can be reduced by reducing the input reference voltage. This allows all the capacitance multipliers in the array to be reduced, reducing the total capacitance by half for each new reference voltage. In this thesis, it has been used 5 new reference voltages, reducing the total capacitance to $1/(2^5) = 1/32$ of the original value. One half of the sDAC is shown in figure 9 below, while the full sDAC with both capacitor arrays is shown in figure 21 in the appendix.

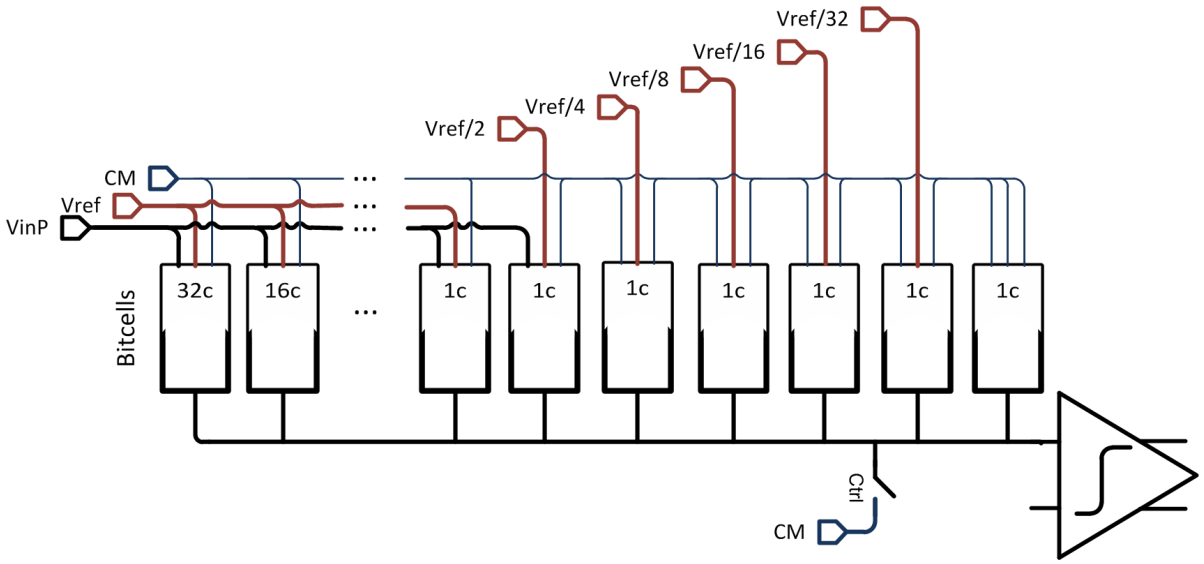

Figure 9: Top half of the capacitive DAC array, along with comparator[3]

[4]A bitcell contains the capacitor and incorporates the possibility to switch between inputs, see section 4.2

4.2 Bitcell

The bitcells contains a capacitor, together with transmission gates(switches) that switches the capacitor's input voltage between V_{inp}, V_{ref} and V_{cm}. To control the switches, two input signals are required. The bitcell design is inspired by a design by Trond Ytterdal[6].

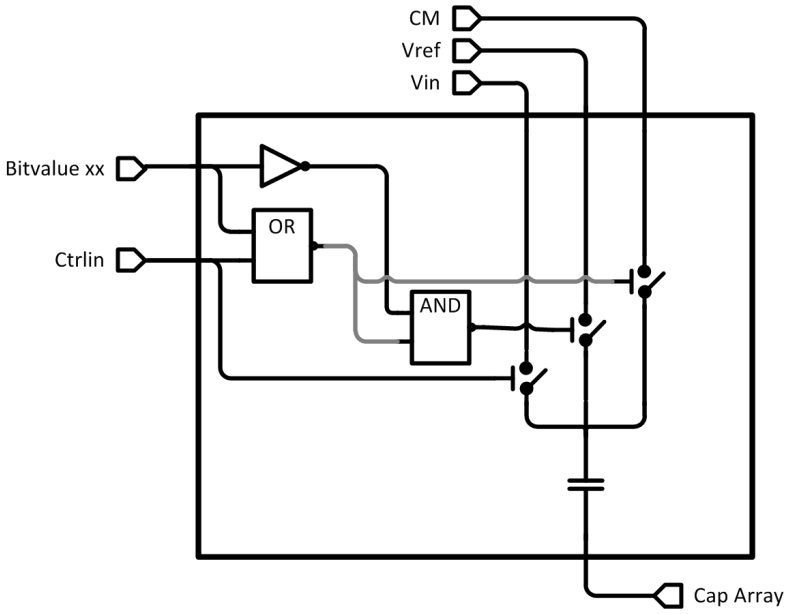

Figure 10: The contents of a bitcell[6]

It should be noted that some of the bitcells in the sDAC never switches to V_{in} or V_{ref}, reducing the logic of these bitcells. Also, the capacitor should be connected in such a way as to way to minimize parasittics at the bitcell output.

4.3 Comparator

For use in this SAR ADC, some requirements were made for the comparator[3]:

Table 2: Comparator design goals

Object	Target
Gain before latch stage	60dB
3dB Bandwidth	65 MHz
Noise referred to input	$\leq 100\mu$V rms
Comparator current consumption	$\leq 60\mu$A average

It was early stated that to eliminate the kickback noise, one has to use several stages[3]. In addition to the kickback noise from the latch, there is also produced kickback noise from the transistor in the 3rd sub-stage, which short circuits the input to the latch, keeping the latch in tracking mode. It was decided that to eliminate the kickbak noise, it was to be used at least 2 sub-stages in front of the 3rd sub-stage. Also, a total of 4 sub-stages distributed in 2 main stages should have enough gain to eliminate offset problems from the input of the latch[3]. Since $V_{lsb,rms}$ is about $207\mu V$[5], the rms noise should be much lower. A safe number of 100μV was set. Also, the bandwidth was targeted at a safe number of 5 times the comparator operating frequency of 13MHz, resulting in a bandwidth of 65MHz[3].

4.3.1 Pre-amplifier, first stage

The first stage used in this design, containing two sub-stages and shown in figure 11, is inspired by the amplifier used in Wenbo Liu's paper[7]. A major modification has been made in the second sub-stage, which is now a copy of the first sub-stage, only the order of the transistor has been reversed, with nmos input transistors and pmos load transistors.

The first stage of the pre-amplifier consists of the first two sub-stages of the comparator, requiring special care in the design phase to achieve low noise and high accuracy. To minimize 1/f noise and limit noise bandwidth, pmos transistors p01 and p02 are used as input transistors[7]. The transistor

[5]see equation 25

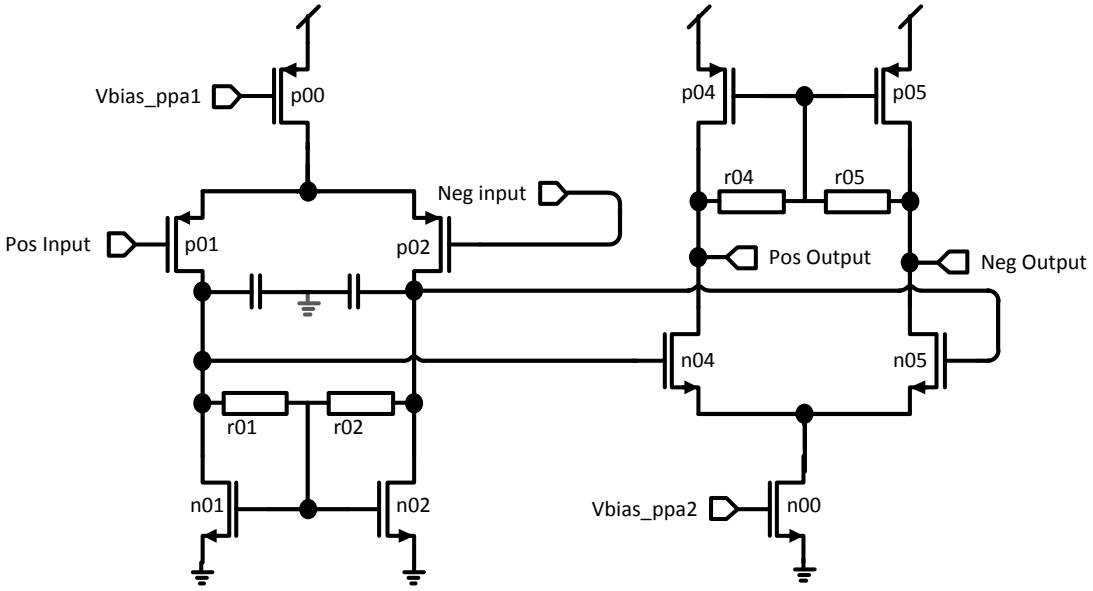

Figure 11: Pre-amplifier, first stage

p00 are used to control bias current, while transistors n01 and n02 are used as loads and n04 and n05 are used as second stage input transistors.

The gain in the first sub-stage can be approximated[2]:

$$Gain_{1st-substage} = gm_{p01} \cdot r_{out} \approx gm_{p01} \cdot rds_{n01} \tag{17}$$

By increasing the w/l-ratio of transistors p01 and p02, increasing gm, or by increasing the length of transistors n01 and n02, increasing rds, we can achieve greater gain. This increase in gain will also decrease input-related noise, as explained in section 2.3.2.

In addition the noise sources of the MOSFET will be decreased by increasing the transistors gm, width and length. The input related noise of a MOSFET can be approximated, according to Johns and Martin[2]:

$$Noise_{MOSFET,input}(f) = 4kT\frac{2}{3}\frac{1}{gm} + \frac{K}{WLC_{ox}f} \tag{18}$$

where k is Boltzmann's constant ($1.38 \cdot 10^{-23}$ JK, T is temperature in Kelvins, gm is the given transistors transconductance, K is a device dependent constant, f is frequency and W, L and C_{ox} is the device width, length and gate capacitance per unit area, respectively. The first part of

the equation is thermal noise, the last part is flicker noise, both which are dominant noise sources in MOSFET design[2].

The input should also work with a wide range of common mode voltages. A problem occurs when both input voltages are low, increasing the voltage at output of the 1st sub-stage, decreasing V_{ds} of the input transistors and resulting in sub threshold operation. This reduces both transconductance and bandwidth[2]. To avoid this, the transistors n01 and n02 should be have a large W/L-ratio, lowering the voltage at the output node of the first sub-stage and reducing the range of input voltages that results in sub-threshold operation.

Table 3: Pre-amplifier parameters, first stage

Mosfet name	Width	Length	Multiplier
p00	0.45	0.10	35
p01, p02	9.00	0.10	2
n01, n02	0.60	0.25	1
p04, p05	9.00	0.10	2
n04, n05	2.40	0.10	2
n00	0.12	0.10	8

Object	Parameter
R01, R02	100kΩ
C01, C02	45fF

4.3.2 Preamplifier, second stage

The preamplifier used in this design is also inspired by the amplifier used in Wenbo Liu's paper[7]. The transistors p01, p02, n04 and n05 are used as input transistors, transistors p00 and n00 are used to control bias current, while transistors n01, n02 and resistors r01,r02 are used as loads. The transistor nclk is used to reset the input to the latch, minimizing hysteresis problems.

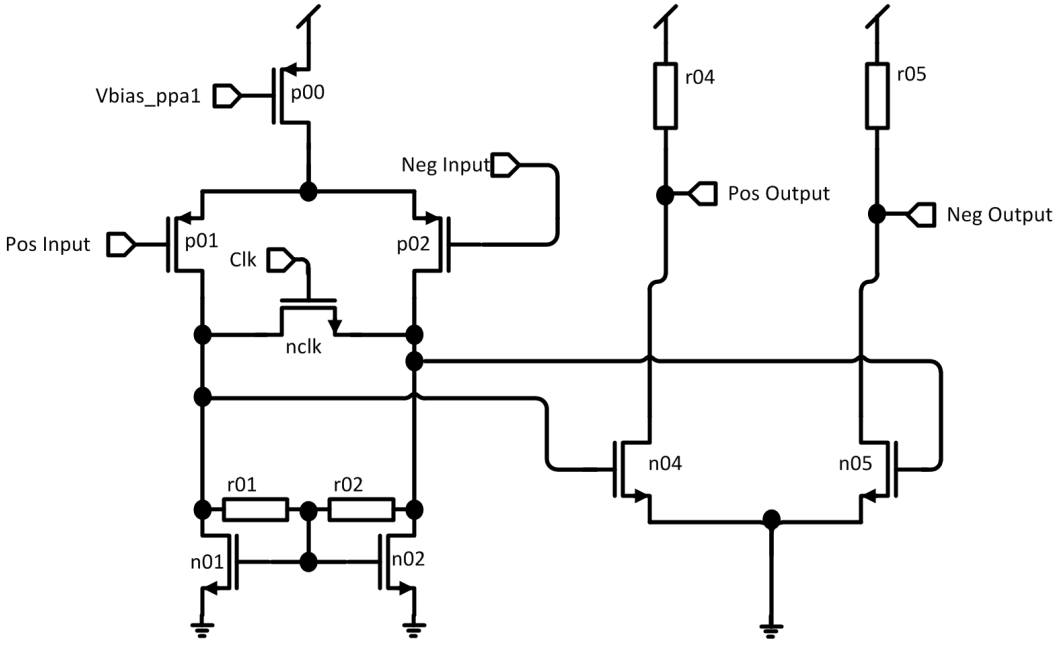

Figure 12: Preamplifier[7]

Table 4: Preamplifier parameters, second stage

Mosfet name	Width	Length	Multiplier
p00	0.45	0.1	4
p01, p02	4.50	0.1	1
n01, n02	0.36	0.2	1
nclk	0.12	0.1	1
n04, n05	0.24	0.4	1

Object	Parameter
R01, R02	500kΩ
R04, R05	300kΩ

4.3.3 Latch

The latch is inspired by a latch made by Bernhard Goll[8]. Input is handled by transistors n03 and n04, bias currents by p05 and p06, while the latch is reset by transistors n00 and n07. The transistors n01, n02, n05 and n06 are positive feedback transistors, latching the output to either high or low.

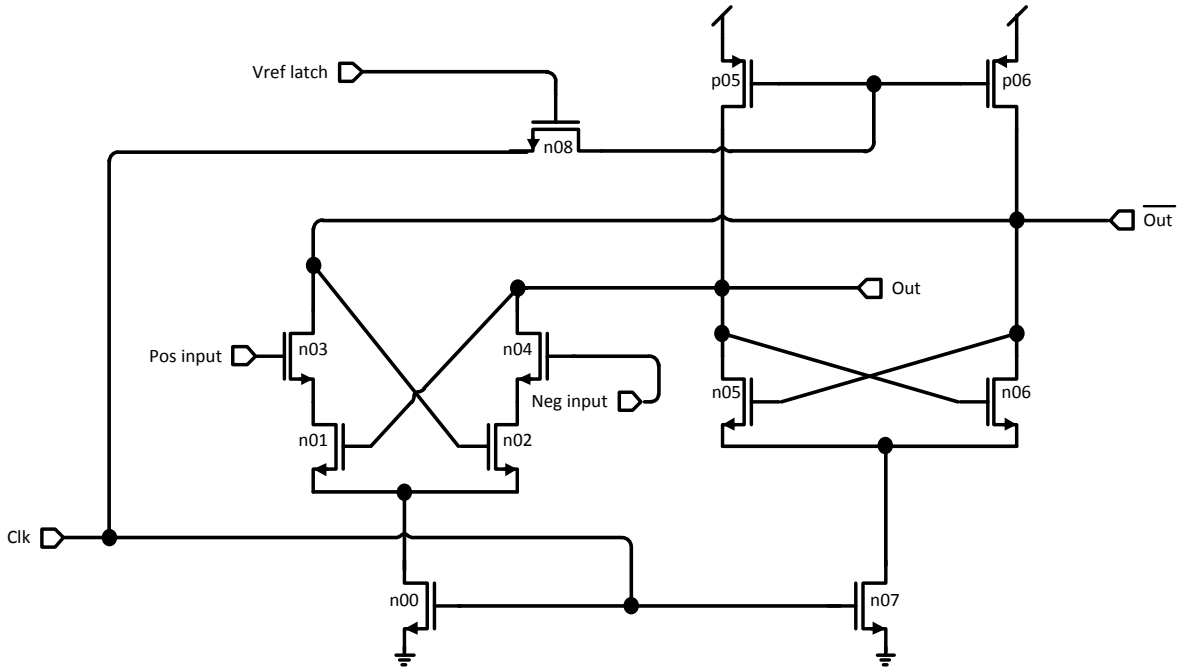

Figure 13: Latch[8]

Table 5: Latch parameters

Mosfet name	Width	Length	Multiplier
n00	0.12	0.1	1
n01,n02	0.36	0.1	1
n03,n04	0.36	0.1	1
n08	0.12	0.1	1
p05, p06	0.18	0.1	1
n05, n06	0.5	0.1	1

4.3.4 Digital signal rectifier

The digital rectifier improves the signal from the latch, severely increasing the time the signal is either vdd or vss, in accordance with digital logic.

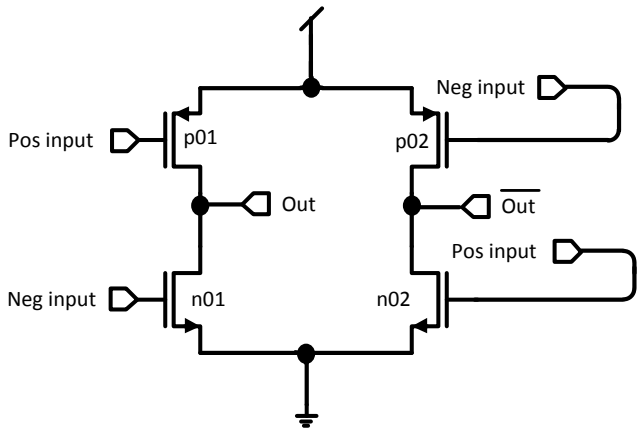

Figure 14: Digital signal rectifier

Table 6: Digital signal rectifier parameters

Mosfet name	Width	Length	Multiplier
n01,n02	0.12	0.1	1
p01, p02	0.12	0.1	1

4.4 Asynchronous clock

The asynchronous clock(aclk) used in this design is inspired by Wenbo Liu's asynchronous clock[7], as described in section 3.4. A little modification has been done to integrate the aclk model into this SAR circuit, the difference being an inverter to invert the input clock signal and an OR-gate to decide if the comparator has finished deciding.

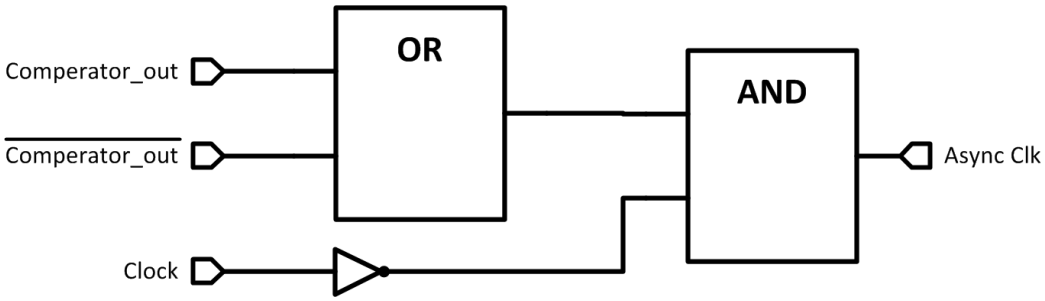

Figure 15: Asynchronous clock, logic blocks

The resulting signals are illustrated in figure 16. The asynchronous clock will go high as the standard clk goes high, but will go low as soon as one of the two outputs from the comparator goes high. If the comparator outputs stay low, the aclk will follow the standard clock.

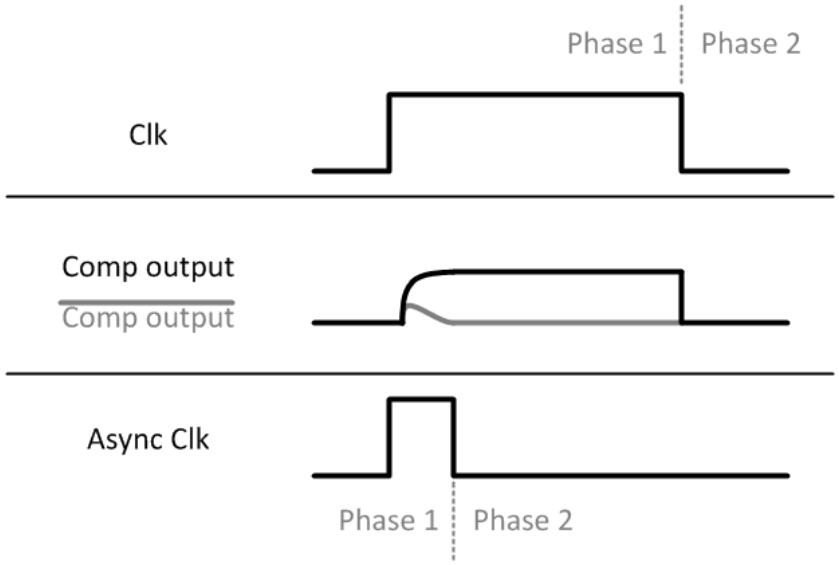

Figure 16: Asynchronous clock, ideal signal inputs and outputs

The asynchronous clock is connected to the digital logic, accelerating the time it takes to go to the next phase for the sDAC. This causes the next voltage change in the sDAC and pre-amplifier to be allowed more time to settle.

As indicated in section 3.4, metastability issues is largely avoided using this design method.

4.5 Switches

This design uses transmission gates as switches, assuring a low on-resistance with a pmos and a nmos transistors in parallel[2]. A low on-resistance not only minimizes the voltage drop across the switch, but also minimizes the time constant $\tau = r \cdot c$, assuring high slew rates[2].

4.6 Logic gates

Outside of the control logic, the SAR ADC still contains a number of logic gates. These have been modeled with cmos transistors, to achieve good simulation accuracy. These models are copied from Sedra & Smith[18].

4.7 Capacitance model

The capacitance models are a part of the BSIM4 spice simulation models made by Berkeley, with the foundry ST-Microelectronics. The capacitance model has parameters for non-idealities, such as parasittic capacitances and offsets. By calculations in section A.2, the minimum unit capacitance is 24.6fF.

4.8 Transistor model

The transistor models are also a part of the BSIM4 spice simulation models, made by Berkeley, with the foundry ST-Microelectronics. The transistor models have parameters for advanced non-idealities and offsets.

4.9 Digital control logic

The digital control logic is coded in VHDL, to be simulated by Questa ADMS. The control logic is designed as a clocked arbiter, with two clock inputs(clk and aclk), comparator decision input, one control signal output, as well as output for the 12bit finished code, also to be used as control signals. The code is somewhat inspired by code written by C.Wulff[19].

5 Simulations and results

5.1 Testbenches

Because of the need to simulate both analog and digital circuits in the design of a SAR ADC, the Questa ADMS mixed signal simulator was used in the overall simulations. For simulations of analog subcircuits, eldo analog circuit simulator was used.

5.1.1 Comparator testbenches

The comparator testbenches were programmed in SPICE and run in Eldo. Some simulation outputs were saved in waveform files, while others were extracted and saved in text files. Ezwave was used analyzing the waveform outputs, as well as processing some of the waveform data, using the included waveform calculator. In the gain simulations, the amplifiers were simulated with a load consisting of transistors equal to the input transistors of the latch.

5.1.2 SAR ADC testbench

To be able to test the individual sub-circuits in a SAR ADC environment, a modular testbench was made. Initially the testbench consisted of ideal circuit elements, which also helped in understanding the workings of a SAR ADC. After designing sub-circuits, such as the sDAC or the comparator, they could easily be implemented in this SAR ADC testbench. Running monte carlo simulations[12], as well as normal simulations, a performance impact of the non-ideal circuit elements could be observed.

5.1.3 Matlab output analysis

The output from the eldo simulation is saved to a file and imported to matlab for analysis[20]. Using a modified matlab script, originally from Carsten Wulff[21], several performance figures is extracted from the output. The script formats the number of eldo outputs to a power-of-two, runs matlab's discete-fourier-transform(fft) on the signal and extracts signal, noise and harmonics. Out of this, one can calculate SNR, SNDR and ENOB, as detailed in the appendix section A.6.

Matlab's fourier transform is computed with a "fast fourier transform" algorithm, but the result is still given by[22]:

$$X(k) = \sum_{j=1}^{N} x(j) \cdot \omega_N^{(j-1)(k-1)} \tag{19}$$

where

$$\omega_N = e^{2\pi i/N} \tag{20}$$

5.2 Comparator

This section contains a selection of relevant simulation results of the comparator and it's sub-circuits.

5.2.1 Offset simulation

Using Eldo's Monte Carlo simulation, some transistor- and capacitor variables will be varied according to a chosen probability distribution. This simulates production imperfections, giving a indication of the quality of the finished circuit. A large number of Monte Carlo runs gives a large sample number to analyze statistically, but each extra run takes the same amount of extra time to simulate.

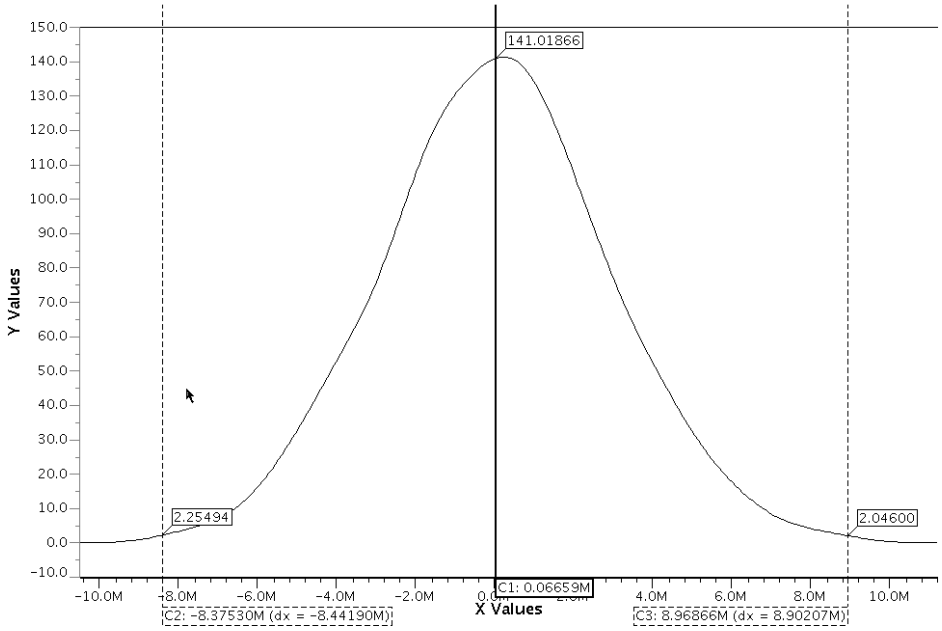

Figure 17: Offset probability distribution of the comparator, simulated by 500 Monte Carlo runs

The comparator offset has been simulated with 500 Monte Carlo runs, resulting in a offset mean of about 0.067mV and a 3-σ deviation of approximately 8.7mV, meaning 99.7% of all the samples are expected to have an offset of less then this value.

In addition, the comparator was simulated at the different corners, with

the results in table 7. The offsets from corner simulations and Monte Carlo simulations should then be added, the results being worst case scenarios.

Table 7: Offset at different process corners

Corner	Offset[mV]
Typical	0.00
Fast-Fast	0.00
Slow-Slow	0.00
Fast-slow	0.00
Slow-Fast	0.00

5.2.2 Noise simulations

The noise is measured at the input of the comparator latch, and divided by the gain to find a practical value for input related noise. The noise is simulated by the .noise command in an ac-simulation in Eldo.

Table 8: Input related noise at process corners

Corner	Noise,rms [μV]
Typical	125.7
Fast-Fast	125.8
Slow-Slow	125.3
Fast-slow	130.3
Slow-Fast	121.9

5.2.3 Gain of preamplifiers

The following results are given by ac-simulations in Eldo.

Table 9: Total DC gain at different process corners

Corner	Total DC gain[dB]
Typical	61.3
Fast-Fast	59.2
Slow-Slow	63.0
Fast-slow	62.0
Slow-Fast	60.0

Table 10: DC gain through the different Stages

Section	DC gain[dB]
1st Sub-section (Input stage)	16.5
2nd Sub-section	14.7
First stage, total	31.2
3rd Sub-section	15.4
4th Sub-section	13.9
Second stage, total	29.3
Pre-amplifier, total	60.5

5.2.4 Bandwidth of preamplifiers

Table 11: Bandwidth at different process corners

Corner	Bandwidth[MHz]
Typical	63.3
Fast-Fast	85.3
Slow-Slow	49.1
Fast-slow	58.2
Slow-Fast	69.0

5.2.5 Current consumption

The comparator was fed a sinus on the positive input and a cosinus on the negative input, both with full amplitudes of 0.6V and a randomly chosen frequency of about 3.62MHz. The currents presented below are the mean value of the absolute value of the currents through the top elements in each sub-circuit.

Table 12: DC gain through the different Stages

Section	Current consumption[μA]
First stage	46.58
Second stage	8.69
Latch	4.50
Signal rectifier	2.50
Comparator, total	62.28

In addition, the currents through vdd in to the circuit was extracted, resulting in a value of 62.09μA. Corresponding closely to the result in table 12 above. In addition to these values, two biasing currents of 1μA each was used, bringing the total absolute mean current consumption to a total of about 64μA.

5.3 SAR ADC overall simulations

The clock was set to 13MHz, with 1ns transition time, together with an extra clock signal, delayed 4ns. Some of the simulations were done with 4096 samples, but most were done with a coherent sampling input frequency of 492.1875kHz, using the method described in section A.5. This keeps keep the sampling number to only 128 samples, reducing the simulation time spent by 97%.

5.3.1 Current consumption

The SAR ADC was simulated with full swing sinus inputs, with 128 conversions. The results in table 13 below are the mean of the absolute values of the current streaming through vdd and vss.

Table 13: Current consumption of the SAR ADC

Section	Current consumption[μA]
From V_{dd}	86.5
To V_{ss}	93.1
Total SAR ADC	$\approx$ 90-95

This is including digital logic ports used in the analog design and asynchronous clock, but excluding the digital control logic and reference generator biasing. Also included in the result is the dynamic current from the reference generator, measured to 24μA. Adding the dynamic current from the reference biasing, together with the comparator current consumption from section 5.3, the result is about 88μA. The remaining 2-7μA is due to the analog logic, switches and the asynchronous clock.

Assuming a total current consumption of 95μA, the power consumption and energy per conversion can be calculated:

$$P = V \cdot I = 1.2V \cdot 95\mu A = 114\mu W \tag{21}$$

$$Energy/conversion = \frac{P}{f_s} = \frac{114\mu A}{1MHz} = 114pJ/conv \tag{22}$$

5.3.2 Error measurements

The SAR ADC was simulated with full-swing sinus inputs of opposite phases at the inputs, at 492.1875kHz. Using the matlab script described in section 5.1.3, SNDR and ENOB was simulated to 69.6dB and 11.27, respectively.

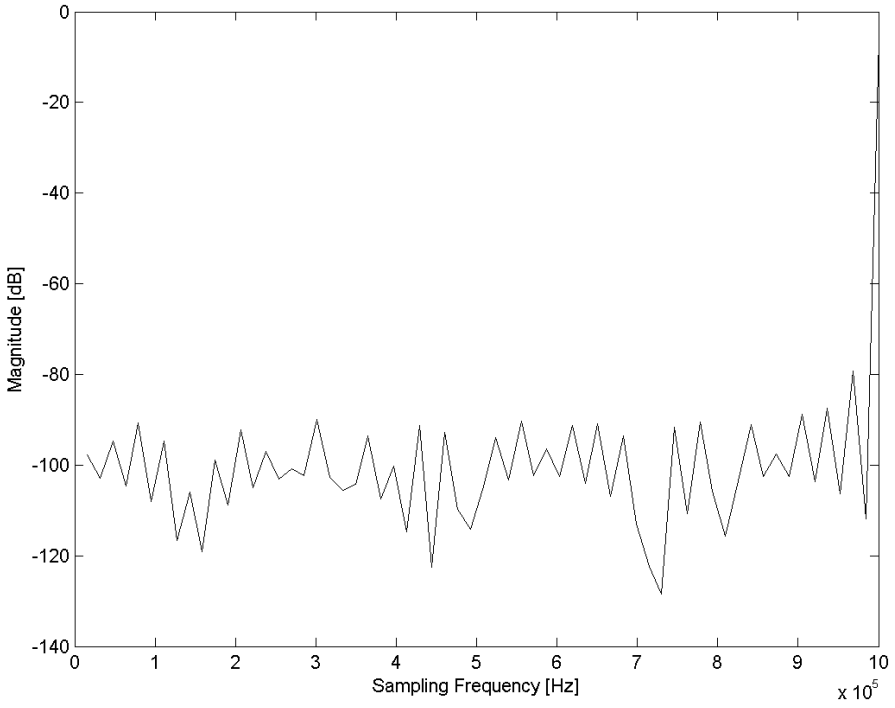

Figure 18: Frequency spectrum of SAR ADC output

Unfortunately, full scale DNL and INL testing is a very time-consuming process. Especially locating the code transitions requires very many simulation cycles. Instead of such simulations, the circuit was simulated for 128cycles of differential sinus inputs. Each input had full-scale sinus input with near-nyquist frequency, but with opposite phases. Ideal signals were created in Matlab and compared to the SAR ADC output. The differences (errors in LSB's) are shown in figure 19. The bit error rate was calculated to about 30%.

Then the circuit was simulated with ideal switches, represented by resistors with voltage-dependent value of 100Ω(ON) or 10GΩ(OFF). The resulting errors are represented by figure 20, equaling about 23% bit errors.

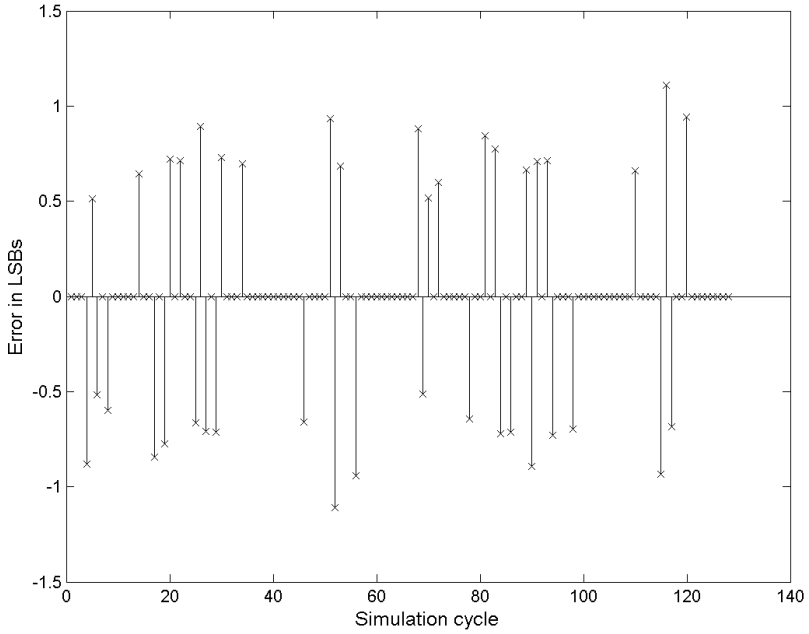

Figure 19: Error in LSB's for a sinus input

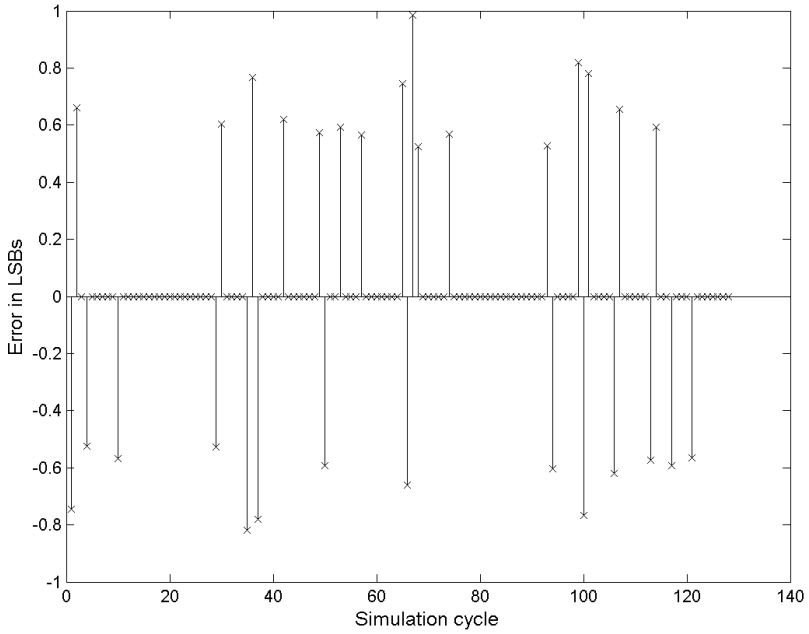

Figure 20: Error in LSB's for a sinus input, with ideal switches

6 Discussion

6.1 Comparator

As shown in section 4.3, the comparator gain requirements are within the specifications given in table 2. The bandwidth is somewhat low, compared to specifications, but had to be balanced against the noise requirements. In the input stage of the pre-amplifier, a capacitor is used to filter random noise. This capacitor increases the time constant[6] of the input node, reducing bandwidth. The results for both noise and bandwidth are acceptable for use in the SAR ADC.

Monte Carlo simulations indicate a 3-sigma offset variation of about 8.7mV, indicating a acceptable level of offset. Offset voltage will usually be removed by offset cancellation techniques, leaving little or no impact on ADC performance[3].

The current consumption was measured for a full swing sinus input, indicating a current consumption of about 64μA, including bias currents. More then 70% of the comparator current consumption is used in the input stage of the comparator, minimizing the input-related noise of the comparator. 64μA is is still close enough to the 60μA current consumption goal for the comparator.

Further work could be done to save more current, for example by duty-cycling the comparator, as mentioned in the next section. Being able to switch off the first comparator stage when it is not needed(about 50% of the time), would lead to a total SAR ADC power reduction of about one third.

6.2 Other sub-components

A asynchronous clock(aclk) was implemented in the design, detecting when a decision was made by the comparator, and then starting the next phase prematurely. This reduces or eliminates any settling issues in the sDAC, using only two extra logic gates and a negligible amount of current. This aclk could, in an eventual future work, be used to reduce the ADC's conversion time and duty-cycle the comparator, reducing current consumption.

The sDAC was implemented using 5 extra voltage references, reducing the

[6]see section A.1

total capacitances needed to a level equal to the thermal noise level. This reduces both settling times, area usage and current consumption from the voltage references. The sDAC will have offsets at the output, due to capacitor mismatch or reference voltage offset, but they should be removable[3]. Future work on the sDAC may include charge recycling, further reducing current consumption of the reference generator.

6.3 Overall SAR ADC

The SAR ADC seems to consume about 90-95μA of current, based on current measurements through the power supply sources V_{DD} and V_{SS}. This is confirmed by the current measurements of the comparator and reference generator, which together measures to about 88μA, excluding only some analog logic ports and switches. The energy per conversion was then estimated to be below 114pJ/conversion.

The error simulations shows some linearity problems in the circuit. Over 128 cycles of sinus input, almost a third of the samples were wrong, although none had more then one LSB wrong. Replacing the transistor switches with near-ideal switches reduced the biterror to about 23%, indicating that some non-linearities originates from the transmission gates. Finding the source of the rest of the non-linearity should be a priority in any future work.

6.4 Topology comparison

Defining the figure of merit(FOM) using nyquist bandwidth, power consumption and the effective number of bits(ENOB):

$$FOM = \frac{power}{2 \cdot bandwidth \cdot 2^{enob}} \tag{23}$$

We can then compare the proposed topology to other relevant topologies:

Table 14: Key number comparison between topologies

Topology	FOM [nJ]	Energy/conversion [pJ/conv]
Energy Micro SAR ADC[3]	537	439
A 12b 22.5/45ms/s SAR ADC[7]	61	160
A 3mw 12b 10ms/s sub-range SAR ADC[9]	381	300
A 12bit 3.125MHz MASH Delta-Sigma[23]	948	864
The proposed SAR ADC	45	114

7 Conclusion

The main goal of the work presented in this thesis was to find and explore new topologies for 1MHz general purpose SAR ADC's, with special care to the power consumption of the overall design. Re-using some techniques from Energy Micro's SAR ADC[3], as well as gathering inspiration from similar SAR ADC topologies, a design was produced in SPICE and VHDL. Using Eldo, Questa ADMS and Matlab, the design was simulated and key parameters was extracted.

Much care was taken in the design of the comparator. A compromise had to be done between current consumption, noise and bandwidth, however the simulated results were close enough to the specifications needed for the SAR ADC.

The reference generator provides the charge needed for the charge-redistribution sDAC used in a SAR ADC. Reducing the amount of capacitance in the sDAC also reduces the amount of dynamic current consumed by the reference generator. Using several extra voltage references, the total capacitance in the sDAC was reduced to a fraction of the original value, reducing the dynamic current consumption of the reference voltage generator. In addition, a asynchronous clock was added, providing extra settling time for the sDAC. The voltage reference generator was represented with an analog model, and should, in an eventual future work, be replaced by a thoroughly designed power-efficient voltage reference.

The SAR ADC, although not completely designed or simulated, is very power-efficient compared to similar general purpose 12bit ADC's. An energy-per-conversion of 114μA is a very good result, and may be improved upon by better use of asynchronous clocking and/or duty-cycling the comparator. The simulations show some non-linearities, which may be a reason for concern. More thorough simulations would be preferred, and could be a natural starting point for a potential future work.

References

[1] aBitAbout, "Nyquist shannon sampling theorem." Picture of non-overlapping spectrums downloaded from:`http://abitabout.com/Nyquist-Shannon+sampling+theorem`.

[2] D. A. Johns and K. Martin, *Analog integrated circuits design.* John Wiley & sons, 1997.

[3] J. Bjørnsen, "Student mentoring," February - June 2011. `j.bjornsen@energymicro.com`.

[4] D. M. Pozar, *Microwave and RF wireless systems.* John Wiley & sons, 2000.

[5] H. Balasubramaniam, W. Galjan, W. Krautschneider, and H. Neubauer, "12-bit hybrid c2c dac based sar adc with floating voltage shield," in *Signals, Circuits and Systems (SCS), 2009 3rd International Conference on*, pp. 1 –5, nov. 2009.

[6] T. Ytterdal, "Email correspondence," February 2011. `ytterdal@iet.ntnu.no`.

[7] W. Liu, P. Huang, and Y. Chiu, "A 12b 22.5/45ms/s 3.0mw 0.059mm2 cmos sar adc achieving over 90db sfdr," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pp. 380 –381, feb. 2010.

[8] B. Goll and H. Zimmermann, "A 0.12 um cmos comparator requiring 0.5v at 600mhz and 1.5v at 6ghz," in *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pp. 316 –605, feb. 2007.

[9] H.-W. Chen, Y.-H. Liu, Y.-H. Lin, and H.-S. Chen, "A 3mw 12b 10ms/s sub-range sar adc," in *Solid-State Circuits Conference, 2009. A-SSCC 2009. IEEE Asian*, pp. 153 –156, nov. 2009.

[10] S. Borkar, "Design challenges of technology scaling," *Micro, IEEE*, vol. 19, pp. 23 –29, jul-aug 1999.

[11] T. Ytterdal, "Student mentoring," February - June 2011. `ytterdal@iet.ntnu.no`.

[12] MentorGraphics, "Eldo's user manual," 2005.

[13] A. National Institute for Subatomic Physics, "Noise sources in mosfet transistors." www.nikhef.nl/~jds/vlsi/noise/sansen.pdf, 1999.

[14] K. H. Lundberg, "Noise sources in bulk cmos," 2002. http://web.mit.edu/klund/www/papers/UNP_noise.pdf.

[15] B. Razavi, *Principles of Data conversion system design*. IEEE Press, 1995.

[16] Y. Zhu, U.-F. Chio, H.-G. Wei, S.-W. Sin, S.-P. U, and R. Martins, "A power-efficient capacitor structure for high-speed charge recycling sar adcs," in *Electronics, Circuits and Systems, 2008. ICECS 2008. 15th IEEE International Conference on*, pp. 642 –645, 31 2008-sept. 3 2008.

[17] RadioElectronics, "Bit error rate testing." http://www.radio-electronics.com/info/rf-technology-design/ber/bit-error-rate-tutorial-definition.php.

[18] A. S. Sedra and K. C. Smith, *Microelectronic circuits*. Oxford university press, 2004.

[19] C. Wulff, "Mulitplying digital to analog converter," 2008. a VHDL code simulating an multiplying DAC, handed out at the university course CMOS2.

[20] C. Wulff, "Mulitplying digital to analog converter," 2008. a VHDL code that writes simulated signals to a file, handed out at the university course CMOS2.

[21] C. Wulff, "Dofft," 2008. a matlab script to find SNDR and ENOB, handed out at the university course CMOS2.

[22] Mathworks, "Mathworks r2010b documentation." http://www.mathworks.com/help/techdoc/ref/fft.html, 2010.

[23] A. Gharbiya and D. Johns, "A 12-bit 3.125 mhz bandwidth 0-3 mash delta-sigma modulator," *Solid-State Circuits, IEEE Journal of*, vol. 44, pp. 2010 –2018, july 2009.

[24] Berkeley, "Berkeley bsim4 svt transistor models," 2006. Obtained from NTNU.

[25] Maxim, "Coherent sampling calculator," April 2004. http://www.maxim-ic.com/app-notes/index.mvp/id/3190.

Appendix

A Calculations

A.1 General equations

The root mean square (RMS) voltage is:

$$V_{rms} = \frac{V}{\sqrt{2}} \tag{24}$$

given that V is a sinusoidal wave.

The LSB voltage , the voltage equivalent to the least significant bit (LSB) is:

$$V_{LSB} = \frac{V_{range}}{2^n} \tag{25}$$

where n is the number of bits in the ADC.

The time constant of a node is defined by:

$$\tau = R \cdot C \tag{26}$$

where R and C is the node's total resistance and capacitance, respectively. The step respons of the node is then defined as:

$$V_{response} = V_{initial} \tag{27}$$

A.2 Capacitors

The simulated capacitors are metal-insulator-metal(MIM) capacitors, from berkeley and ST Microelectronics. The unit capacitance can be calculated from formulas and parameters in the cmim library v1.2:

$$C_{unit} = \frac{\epsilon_0 \epsilon_{ox} W_e L_e}{t_{ox}} = 24.6fF \tag{28}$$

from the following calculations and typical parameters:

$$L_e = \frac{cperim}{4} + \frac{sqrt_delt}{2} = \frac{14 \cdot 10^{-6}}{4} = 3.5 \cdot 10^{-6} \tag{29}$$

$$We = \frac{caream}{L_e} = \frac{12.25 \cdot 10^{-12}}{3.5 \cdot 10^{-6}} = 3.5 \cdot 10^{-6} \tag{30}$$

$$\begin{aligned} \epsilon_0 &= 8.854187 \cdot 10^{-12} \\ E_{tox} &= 7.25 \\ t_{ox} &= 3.2 \cdot 10^{-8} \\ caream_{default} &= 12.25 \cdot 10^{-12} \\ cperim_{default} &= 14.00 \cdot 10^{-6} \end{aligned} \tag{31}$$

A.3 Oxide unit capacitance C_{ox}

The oxide unit capacitance[2], with transistor data from berkeley BSIM4 SVT models[24]:

$$\begin{aligned} C_{OX} &= \frac{\epsilon_0 \cdot \epsilon_r}{t_{ox}} \\ C_{OX,nmos} &= \frac{8.854 \cdot 10^{-12} \cdot 3.9}{1.7772 \cdot 10^{-9}} = 0.01943 \\ C_{OX,pmos} &= \frac{8.854 \cdot 10^{-12} \cdot 3.9}{1.8039 \cdot 10^{-9}} = 0.01914 \end{aligned} \tag{32}$$

where t_{ox} is the thickness of the oxide layer, ϵ_0 is the vacuum permittivity and ϵ_r is the relative permittivity.

A.4 Thermal noise limit

The thermal noise in a system equals to[2]:

$$V_{no,rms}^2 = \frac{k \cdot T}{C_{node}} \tag{33}$$

where k i Boltzmann's constant, T is the temperature in kelvins (400K, or 125degrees C) and C_{node} is the capacitance in the given node. We want the thermal rms noise to be no higher then the quantization noise[3][2]:

$$V_{no,rms} \leq V_{q,rms} = \frac{V_{lsb}}{\sqrt{12}} = \frac{293.0\mu V}{\sqrt{12}} = 84.6uV \tag{34}$$

which gives a minimum array capacitance of:

$$C_{minimum} = \frac{k \cdot T}{V_{no,rms}^2} = \frac{1.38 \cdot 10^{-23} \cdot 400}{(84.6\mu V)^2} = 771.3fF \tag{35}$$

A.5 Coherent sampling

To avoid simulating 4096 samples, one can use the coherent sampling technique. According to Maxim IC[25], coherent sampling produces the best quality in high quality FFT's, the alternative being window sampling. The purpose of this sampling is to force an integer number of input cycles within a sampling windows[25]. This can be expressed by[25]:

$$\frac{f_{in}}{sample} = \frac{N_{window}}{N_{record}} \tag{36}$$

where ff_{in} is the input frequency, f_{sample} is the sampling frequency, N_{window} is the integer(odd or prime) number of cycles within the sampling window and N_{record} is the number of sampled data points[25].

One wants a input frequency that is close to, but lower then the sampling frequency, because of the Nyquist f_{fin}/f_{sample} criteria[2]. Since N_{window} must be a odd or prime number, and $N_{window}/N_{record} \leq 0.5$, N_{window} is set to 63. This results in a input frequency of 492.1875 kHz.

A.6 Performance figures

SNR - Signal to noise ratio is the signal power relative to the total noise power.

$$P_{SNR,dB} = 10 \cdot log_{10}(\frac{P_{signal}}{P_{noise}}) = 20 \cdot log_{10}(\frac{V_{signal}}{V_{noise}}) \tag{37}$$

SNDR / SINAD - Signal to noise and distortion ratio is equal to the SNR, but with harmonic distortion (HD) added to the noise power.

$$P_{SNDR,dB} = 20 \cdot log_{10}(\frac{V_{signal}}{V_{noise} + V_{HD-total}}) \tag{38}$$

ENOB - Effective number of bits, a measure for accuracy of an ADC.

$$ENOB = \frac{SNDR_{dB} - 1.76dB}{6.01} \tag{39}$$

B Figures

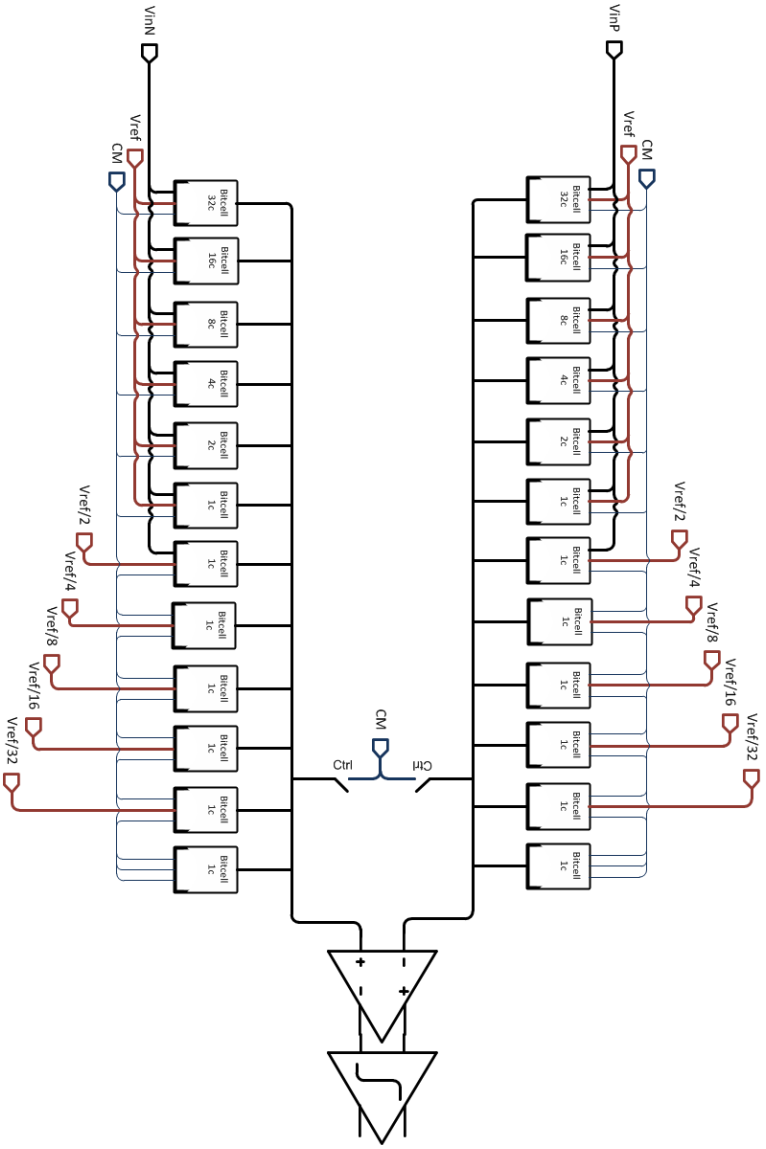

Figure 21: Capacitive DAC array, along with amplifier and latch

C SAR ADC source code

C.1 SAR ADC core, SPICE

```
***Bitcells with metal-insulator-metal(MIM) capacitors, c_unit
   =24.573 fF
.subckt bitcell2 vdd vss ctrlin b in ref cm out multip=1
*inv1 b bb lmod vhi='vdd' vlo='vss'
xkinv1 vdd vss b bb kinverter
xknor1 vdd vss ctrlin b ctrlgnd knor
xknor2 vdd vss ctrlin bb ctrlref knor
xksi in ctrlin c kswitch3
xksr ref ctrlref c kswitch3
xksc cm ctrlgnd c kswitch3
xcap1 c out vss cmimmk mult=multip
.ends

***A "singleended" dac-core
.subckt dac ctrlin b01 b02 b03 b04 b05 b06 b07 b08 b09 b10 b11 in
   ref ref2 ref4 ref8 ref16 vdd vss cm out
xb01 vdd vss ctrlin b01 in ref cm out bitcell multip=1024
xb02 vdd vss ctrlin b02 in ref cm out bitcell multip=512
xb03 vdd vss ctrlin b03 in ref cm out bitcell multip=256
xb04 vdd vss ctrlin b04 in ref cm out bitcell multip=128
xb05 vdd vss ctrlin b05 in ref cm out bitcell multip=64
xb06 vdd vss ctrlin b06 in ref cm out bitcell multip=32
xb07 vdd vss ctrlin b07 in ref cm out bitcell multip=16
xb08 vdd vss ctrlin b08 in ref cm out bitcell multip=8
xb09 vdd vss ctrlin b09 in ref cm out bitcell multip=4
xb10 vdd vss ctrlin b10 in ref cm out bitcell multip=2
xb11 vdd vss ctrlin b11 in ref cm out bitcell multip=1
xbu vdd vss ctrlin vss in cm cm out bitcell multip=1
.ends

***A "singleended" dac-core, capacitance-reduced, multip=capsize,
   MIM-capacitances
.subckt jdac2 ctrlin b01 b02 b03 b04 b05 b06 b07 b08 b09 b10 b11 in
    ref ref2 ref4 ref8 ref16 ref32 vdd vss cm out
xb01 vdd vss ctrlin b01 in ref cm out bitcell2 multip=32
xb02 vdd vss ctrlin b02 in ref cm out bitcell2 multip=16
xb03 vdd vss ctrlin b03 in ref cm out bitcell2 multip=8
```

```
xb04 vdd vss ctrlin b04 in ref cm out bitcell2 multip=4
xb05 vdd vss ctrlin b05 in ref cm out bitcell2 multip=2
xb06 vdd vss ctrlin b06 in ref cm out bitcell2 multip=1
xb07 vdd vss ctrlin b07 in ref2 cm out bitcell2 multip=1
xb08 vdd vss ctrlin b08 cm ref4 cm out bitcell2 multip=1
xb09 vdd vss ctrlin b09 cm ref8 cm out bitcell2 multip=1
xb10 vdd vss ctrlin b10 cm ref16 cm out bitcell2 multip=1
xb11 vdd vss ctrlin b11 cm ref32 cm out bitcell2 multip=1
xbu vdd vss vss vss cm cm cm out bitcell2 multip=1
.ends

***Reference generator
.subckt refgen ref ref2 ref4 ref8 ref16 ref32 cm rmult=25 volt=625m
vrefsrc refin cm volt
rin refin ref '(rmult*32)*((1/0.96)-1)'
cin ref cm 5p
r32 ref ref2 '16*rmult'
r16 ref2 ref4 '8*rmult'
r8 ref4 ref8 '4*rmult'
r4 ref8 ref16 '2*rmult'
r2 ref16 ref32 '1*rmult'
r1 ref32 cm '1*rmult'
.ends

***Reference switch
.subckt refsw vdd vss refp refn refctrl refout
*invrefsw refctrl refctrl_inv lmod vhi='vdd' vlo='vss'
xkinvrefsw vdd vss refctrl refctrl_inv kinverter
xkrefsw1 refp refctrl refout kswitch3
xkrefsw2 refn refctrl_inv refout kswitch3
.ends

***Differential SAR ADC core
.subckt dacdiff ctrlin b00 b01 b02 b03 b04 b05 b06 b07 b08 b09 b10
   b11 inp inn refp refp2 refp4 refp8 refp16 refp32 refn refn2
   refn4 refn8 refn16 refn32 vdd vss cm outa outb
***Choosing Vref on A
xrefsw_a1 vdd vss refp refn b00 refa refsw
xrefsw_a2 vdd vss refp2 refn2 b00 refa2 refsw
xrefsw_a4 vdd vss refp4 refn4 b00 refa4 refsw
xrefsw_a8 vdd vss refp8 refn8 b00 refa8 refsw
```

```
xrefsw_a16 vdd vss refp16 refn16 b00 refa16 refsw
xrefsw_a32 vdd vss refp32 refn32 b00 refa32 refsw

***Choosing Vref on B
xrefsw_b1 vdd vss refn refp b00 refb refsw
xrefsw_b2 vdd vss refn2 refp2 b00 refb2 refsw
xrefsw_b4 vdd vss refn4 refp4 b00 refb4 refsw
xrefsw_b8 vdd vss refn8 refp8 b00 refb8 refsw
xrefsw_b16 vdd vss refn16 refp16 b00 refb16 refsw
xrefsw_b32 vdd vss refn32 refp32 b00 refb32 refsw

***C-arrays
xdacp ctrlin b01 b02 b03 b04 b05 b06 b07 b08 b09 b10 b11 inp refa
   refa2 refa4 refa8 refa16 refa32 vdd vss cm outa jdac2
xdacn ctrlin b01 b02 b03 b04 b05 b06 b07 b08 b09 b10 b11 inn refb
   refb2 refb4 refb8 refb16 refb32 vdd vss cm outb jdac2

***Switches between C-arrays
xswa outa ctrlin cm kswitch2
xswb outb ctrlin cm kswitch2

.ends
```

C.2 SAR ADC top, SPICE

```
.subckt saradc clk clka vdd vss inp inn cm ctrlin bo00 bo01 bo02
   bo03 bo04 bo05 bo06 bo07 bo08 bo09 bo10 bo11 outcomp

.inc source/sarcore.cir
.inc source/comparator.cir
.inc source/unsync_clk.cir
.inc source/models.cir

***Making smaller references
xrefgenp refp refp2 refp4 refp8 refp16 refp32 cm refgen volt
   ='0.625'
xrefgenn refn refn2 refn4 refn8 refn16 refn32 cm refgen volt
   ='-0.625'

***Sar-core
xsaradc_core ctrlin b00 b01 b02 b03 b04 b05 b06 b07 b08 b09 b10 b11
    inp inn refp refp2 refp4 refp8 refp16 refp32 refn refn2 refn4
   refn8 refn16 refn32 vdd vss cm outa outb dacdiff

***Comparator - ideal
*comp1 outa outb outcomp voff=0 vdef=0
*+vhi=vdd vlo=vss
*invert1 outcomp outcompn lmod vhi='vdd' vlo='vss'

***Comparator - cmos090
xcomp1 vdd vss outa outb clk clka outcomp outcompn comparator

**Unsyncronised clock
xuclk vdd vss clk outcomp outcompn uclk unsync_clk

***Digital control
.model dig(ideal) macro lang=vhdlams
Ydigctrl dig(ideal)
+ PORT: outcomp ctrlin uclk clk clka (b00 b01 b02 b03 b04 b05 b06
   b07 b08 b09 b10 b11) (bo00 bo01 bo02 bo03 bo04 bo05 bo06 bo07
   bo08 bo09 bo10 bo11)

.ends
```

C.3 Modules, SPICE

```
***Ideal switch model
.subckt kswitch a c z
rr a z value={eval(v(c)>{vdd/2}?100:10g)}
.ends

***Non-ideal switch model2
.subckt kswitch2 a c z
vsssw1 vss 0 0
vddsw1 vdd 0 1.2
*invsw c c1 lmod vhi='1.2' vlo='0'
xsw1 a c z vss nsvt w='0.12*2' l='0.1*2' nfing=1.0 mult=15
   srcefirst=1.0 mismatch=1.0
*xsw2 z c1 a vdd psvt w='0.12*20*4.3' l=0.1 nfing=1.0 mult=15
   srcefirst=1.0 mismatch=1.0
.ends
***Non-ideal switch model3
.subckt kswitch3 a c z
vsssw1 vss 0 0
vddsw1 vdd 0 1.2
invsw c c1 lmod vhi='1.2' vlo='0'
xsw1 a c z vss nsvt w='0.12*5' l=0.1 nfing=1.0 mult=1 srcefirst=1.0
    mismatch=1.0
xsw2 z c1 a vdd psvt w='0.12*5*4.3' l=0.1 nfing=1.0 mult=1
   srcefirst=1.0 mismatch=1.0
.ends

***Inverter
.subckt kinverter vdd vss in out
xp01 out in vdd vdd psvt w=0.12 l=0.1 nfing=1.0 mult=1.0 srcefirst
   =1.0 mismatch=1.0
xn01 out in vss vss nsvt w=0.12 l=0.1 nfing=1.0 mult=1.0 srcefirst
   =1.0 mismatch=1.0
.ends

*******LOGIC GATES FROM SEDRA & SMITH*******
***NAND
.subckt knand vdd vss in1 in2 out
```

```
xp01 out in1 vdd vdd psvt w=0.12 l=0.1 nfing=1.0 mult=1.0 srcefirst
    =1.0 mismatch=1.0
xp02 out in2 vdd vdd psvt w=0.12 l=0.1 nfing=1.0 mult=1.0 srcefirst
    =1.0 mismatch=1.0
xn01 out in1 n01 vss nsvt w=0.12 l=0.1 nfing=1.0 mult=1.0 srcefirst
    =1.0 mismatch=1.0
xn02 n01 in2 vss vss nsvt w=0.12 l=0.1 nfing=1.0 mult=1.0 srcefirst
    =1.0 mismatch=1.0
.ends

***AND
.subckt kand vdd vss in1 in2 out
xnand1 vdd vss in1 in2 out1 knand
xinv1 vdd vss out1 out kinverter
.ends

***NOR
.subckt knor vdd vss in1 in2 out
xp01 n02 in1 vdd vdd psvt w=0.12 l=0.1 nfing=1.0 mult=1.0 srcefirst
    =1.0 mismatch=1.0
xp02 out in2 n02 vdd psvt w=0.12 l=0.1 nfing=1.0 mult=1.0 srcefirst
    =1.0 mismatch=1.0
xn01 out in1 vss vss nsvt w=0.12 l=0.1 nfing=1.0 mult=1.0 srcefirst
    =1.0 mismatch=1.0
xn02 out in2 vss vss nsvt w=0.12 l=0.1 nfing=1.0 mult=1.0 srcefirst
    =1.0 mismatch=1.0
.ends

.subckt kor vdd vss in1 in2 out
xnor1 vdd vss in1 in2 out1 knor
xinv1 vdd vss out1 out kinverter
.ends
```

C.4 Pre-amplifier top, SPICE

```
.subckt comparator vdd vss vinp vinn clk clka vout vnout

.inc latch.cir
.inc preamp.cir
.inc prepreamp.cir
```

```
***Source size
.param iref = 1u
.param tbias = 1.2

***Noise capacitor sizes
.param cnoise=45f
.param cnoise2=10f
*-------------------------------------------------------
* Transistor sizes
*-------------------------------------------------------
***Tran size
.param wminsize={0.12}
.param lminsize={0.1}
.param upun=3.75

**general transistors (xn08)
.param nwidth = {1*wminsize}
.param nlength = {1*lminsize}
.param pwidth = {1*wminsize*upun}
.param plength = {1*lminsize}

.param cmirr_mult=1
*-------------------------------------------------------
* Biasing
*-------------------------------------------------------
vtbias ntbias vss tbias

xp00cmirr vrefppa1 vrefppa1 vdd vdd psvt w='pwidth' l='plength'
   nfing=1.0 mult=cmirr_mult srcefirst=1.0 mismatch=1.0
icmirr vrefppa1 vss iref

xn004cm vrefppa2 vrefppa2 vss vss nsvt w=nwidth l=nlength nfing=1.0
    mult=1 srcefirst=1.0 mismatch=1.0
icm4 vdd vrefppa2 iref

*--------CIRCUIT---------------------------------------
xprepreamp vdd vss vinp vinn vrefppa1 vrefppa2 ppa_outp ppa_outn
   prepreamp
xpreamp vdd vss ppa_outp ppa_outn clk vrefppa1 pa_outp pa_outn
   preamp
```

```
xlatch vdd vss pa_outp pa_outn clka ntbias vout1 vnout1 latch
xpostlatch vdd vss vout1 vnout1 vout vnout postlatch2

.ends comparator
```

C.4.1 1st stage, SPICE

```
    ***************
***PreAmp subcircuit***
    ***************
.SUBCKT prepreamp vdd vss vinp vinn vrefppa1 vrefppa2 n05 n04
***Tran size
.param wminsize={0.12}
.param lminsize={0.1}
.param upun=3.75

.param pwidth_ppa_in = {20*upun*wminsize}
.param plength_ppa_in = {1*lminsize}
.param ppa_in_mult = 2

.param nwidth_ppa_n01 = {16*wminsize}
.param nlength_ppa_n01 = {2*lminsize}

.param pwidth_ppa_p01 = {1*wminsize*upun}
.param plength_ppa_p01 = {1.0*lminsize}

.param pwidth_ppa_p00 = {1*wminsize*upun}
.param plength_ppa_p00 = {1*lminsize}
.param cmirr_ppa_mult = 35

.param nwidth_ppa_n00 = {wminsize}
.param nlength_ppa_n00 = {1*lminsize}
.param cmirr_ppa_mult2 = 8

.param nwidth_ppa_out = {20*wminsize}
.param nlength_ppa_out = {1*lminsize}
.param ppa_out_mult = 2

.param rval1_ppa='100k'
.param rval2_ppa='250k'
```

```
***************
***Section 1***
***************
xp00 n00 vrefppa1 vdd vdd psvt w=pwidth_ppa_p00 l=plength_ppa_p00
    nfing=1.0 mult=cmirr_ppa_mult srcefirst=1.0 mismatch=1.0

xp01 n01 vinn n00 n00 psvt w=pwidth_ppa_in l=plength_ppa_in nfing
    =1.0 mult=ppa_in_mult srcefirst=1.0 mismatch=1.0
xp02 n02 vinp n00 n00 psvt w=pwidth_ppa_in l=plength_ppa_in nfing
    =1.0 mult=ppa_in_mult srcefirst=1.0 mismatch=1.0

r01 n03 n01 rval1_ppa
r02 n03 n02 rval1_ppa
*vrefr n03 vss 659m

c01 n01 ncn1 cnoise
vncn1 ncn1 vss 0.6
c02 n02 ncn1 cnoise

*.param ireferance=900n
*in01 vss n01 ireferance
xn01 n01 n03 vss vss nsvt w=nwidth_ppa_n01 l=nlength_ppa_n01 nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0
*in02 vss n02 ireferance
xn02 n02 n03 vss vss nsvt w=nwidth_ppa_n01 l=nlength_ppa_n01 nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0

***************
***Section 2***
***************
#com
r04 vdd n04 rval2_ppa
r05 vdd n05 rval2_ppa
#endcom
xp04 n04 n07 vdd vdd psvt w=pwidth_ppa_p01 l=plength_ppa_p01 nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0
xp05 n05 n07 vdd vdd psvt w=pwidth_ppa_p01 l=plength_ppa_p01 nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0
r04 n04 n07 rval2_ppa
r05 n05 n07 rval2_ppa
```

```
#com
c04 n04 ncn4 'cnoise2'
vncn4 ncn4 vss 0.6
c05 n05 ncn4 'cnoise2'
#endcom

xn04 n04 n01 n004 vss nsvt w=nwidth_ppa_out l=nlength_ppa_out nfing
    =1.0 mult=ppa_out_mult srcefirst=1.0 mismatch=1.0
xn05 n05 n02 n004 vss nsvt w=nwidth_ppa_out l=nlength_ppa_out nfing
    =1.0 mult=ppa_out_mult srcefirst=1.0 mismatch=1.0

xn004 n004 vrefppa2 vss vss nsvt w=nwidth_ppa_n00 l=nlength_ppa_n00
     nfing=1.0 mult=cmirr_ppa_mult2 srcefirst=1.0 mismatch=1.0

.ENDS prepreamp
```

C.4.2 2nd stage

```
***************
***PreAmp subcircuit***
***************
.SUBCKT preamp vdd vss vinp vinn clk vrefpa n05 n04

***Tran size
.param wminsize={0.12}
.param lminsize={0.1}
.param upun=3.75

.param pwidth_pa_in = {10*wminsize*upun}
.param plength_pa_in = {1*lminsize}

.param nwidth_pa_n01 = {3*wminsize}
.param nlength_pa_n01 = {2*lminsize}

.param pwidth_pa_p00 = {1*wminsize*upun}
.param plength_pa_p00 = {1*lminsize}
.param cmirr_pa_mult = 4

.param nwidth_pa_clk = {wminsize}
```

```
.param nlength_pa_clk = {lminsize}

.param nwidth_pa_out = {2*wminsize}
.param nlength_pa_out = {4*lminsize}

.param rval1_pa='1meg/2'
.param rval2_pa='150k*2'
***************
***Section 1***
***************
xp00 n00 vrefpa vdd vdd psvt w=pwidth_pa_p00 l=plength_pa_p00 nfing
   =1.0 mult=cmirr_pa_mult srcefirst=1.0 mismatch=1.0

xp01 n01 vinn n00 vdd psvt w=pwidth_pa_in l=plength_pa_in nfing=1.0
    mult=1.0 srcefirst=1.0 mismatch=1.0
xp02 n02 vinp n00 vdd psvt w=pwidth_pa_in l=plength_pa_in nfing=1.0
    mult=1.0 srcefirst=1.0 mismatch=1.0

xnclk n01 clk n02 vss nsvt w=nwidth_pa_clk l=nlength_pa_clk nfing
   =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0

r01 n03 n01 rval1_pa
r02 n03 n02 rval1_pa

xn01 n01 n03 vss vss nsvt w=nwidth_pa_n01 l=nlength_pa_n01 nfing
   =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0
xn02 n02 n03 vss vss nsvt w=nwidth_pa_n01 l=nlength_pa_n01 nfing
   =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0

***************
***Section 2***
***************
r04 vdd n04 rval2_pa
r05 vdd n05 rval2_pa

xn04 n04 n01 vss vss nsvt w=nwidth_pa_out l=nlength_pa_out nfing
   =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0
xn05 n05 n02 vss vss nsvt w=nwidth_pa_out l=nlength_pa_out nfing
   =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0

.ENDS preamp
```

C.4.3 Latch and signal rectifier, SPICE

```
.SUBCKT latch vdd vss vinp vinn clk tbias vout vnout
***Tran size
.param wminsize={0.12}
.param lminsize={0.1}
.param upun=3.75

.param nwidth_la_00 = {1*wminsize}
.param nlength_la_00 = {lminsize}

**latch 01 og 02
.param nwidth_la_01 = {3*wminsize}
.param nlength_la_01 = {lminsize}

**in-latch
.param nwidth_la_in = {3*wminsize}
.param nlength_la_in = {lminsize}

**out-latch
.param nwidth_la_out = {1.25*wminsize}
.param nlength_la_out = {lminsize}
.param pwidth_la_out = {wminsize*upun/2.5}
.param plength_la_out = {lminsize}

**general transistors (xn08)
.param nwidth = {1*wminsize}
.param nlength = {1*lminsize}
.param pwidth = {1*wminsize*upun}
.param plength = {1*lminsize}

***************
***Section 1***
***************
xn00 n00 clk vss vss nsvt w=nwidth_la_00 l=nlength_la_00 nfing=1.0
    mult=1.0 srcefirst=1.0 mismatch=1.0

xn01 n01 vnout n00 vss nsvt w=nwidth_la_01 l=nlength_la_01 nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0
xn02 n02 vout n00 vss nsvt w=nwidth_la_01 l=nlength_la_01 nfing=1.0
     mult=1.0 srcefirst=1.0 mismatch=1.0
```

```
xn03 vout vinn n01 vss nsvt w=nwidth_la_in l=nlength_la_in nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0
xn04 vnout vinp n02 vss nsvt w=nwidth_la_in l=nlength_la_in nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0

xn08 clkr tbias clk vss nsvt w=nwidth l=nlength nfing=1.0 mult=1.0
    srcefirst=1.0 mismatch=1.0

***************
***Section 2***
***************
xp05 vout clkr vdd vdd psvt w=pwidth_la_out l=plength_la_out nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0
xp06 vnout clkr vdd vdd psvt w=pwidth_la_out l=plength_la_out nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0

xn05 vout vnout n03 vss nsvt w=nwidth_la_out l=nlength_la_out nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0
xn06 vnout vout n03 vss nsvt w=nwidth_la_out l=nlength_la_out nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0

xn07 n03 clk vss vss nsvt w=nwidth_la_00 l=nlength_la_00 nfing=1.0
    mult=1.0 srcefirst=1.0 mismatch=1.0

.ENDS latch


.SUBCKT postlatch2 vdd vss vinp vinn vout vnout
***Tran size
.param wminsize={0.12}
.param lminsize={0.1}
.param upun=3.75

.param nwidth_pl_in = {1*wminsize}
.param nlength_pl_in = {lminsize}
.param pwidth_pl_in = {1*wminsize}
.param plength_pl_in = {lminsize}

xn01 vout vinn vss vss nsvt w=nwidth_pl_in l=nlength_pl_in nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0
```

```
xp01 vout vinn vdd vdd psvt w=pwidth_pl_in l=plength_pl_in nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0

xn02 vnout vinp vss vss nsvt w=nwidth_pl_in l=nlength_pl_in nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0
xp02 vnout vinp vdd vdd psvt w=pwidth_pl_in l=plength_pl_in nfing
    =1.0 mult=1.0 srcefirst=1.0 mismatch=1.0

.ENDS postlatch2
```

C.5 Asynchronous clock, SPICE

```
.SUBCKT unsync_clk vdd vss clk inp inn uclk
**idealComp
xkinvert1 vdd vss clk invclk kinverter
**realComp
xkor1 vdd vss inp inn n1 kor
xkand2 vdd vss invclk n1 uclk kand
.ends unsync_clk_ideal
```

C.6 Digital controller logic, VHDL

```
    library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;
USE IEEE.STD_LOGIC_1164.ALL;
library MGC_AMS;
use MGC_AMS.eldo.all;
entity DIG is
  port (
   terminal vcomp, ctrlin : Electrical; --Analog Input/Output
   signal uclk,phi1,phi1a : in std_logic:='0'; -- Clock Signals
   signal bcurr :inout std_logic_vector (0 to 11) :="000000000000";
      --current digital
   signal bout :out std_logic_vector (0 to 11) :="000000000000" --
      finished digital out
   );
end DIG;

architecture ideal of DIG is
  type state_type is (SAMP,COMP,B01,B02,B03,B04,B05,B06,B07,B08,B09,
     B10,B11);
  signal state :state_type:=SAMP;
  signal nextstate :state_type:=SAMP;
  signal b : std_logic_vector (0 to 11) :="000000000000";

-- terminal op_ctrlin : electrical;
  -- Define voltages and currents with quantities
  quantity q_vcomp across vcomp;

  quantity q_ctrlin across i_ctrlin through ctrlin;
  -- Internal analog signals
   signal q_op_vcomp : real := 0.0;
   signal s_ctrlin : real := 0.0;
begin

   q_ctrlin == s_ctrlin'ramp(10.0e-12);

   arbiter: process(uclk,phi1)
   variable q_vdd : real :=1.2;
   variable q_vss : real :=0.0;
```

```
variable q_vcm : real :=0.6;
variable setctrlin : integer := 0;

begin

if phi1'event and phi1='0' and setctrlin=1 then
  s_ctrlin <= q_vss;
  setctrlin := 0;
  bcurr <= "000000000000";
  end if;

q_op_vcomp <= q_vcomp;
if uclk'event and uclk='0' then
   --Sampling

CASE state IS
when SAMP =>
   if(q_op_vcomp > q_vcm) then -- Sett LSB
      --bcurr(11) <= not bcurr(0); --negative input, Va-
         Vb=positiv
   b(11) <= not bcurr(0);
   else
      --bcurr(11) <= bcurr(0); --postive input, Va-Vb=
         negative
   b(11) <= bcurr(0);
   end if;
   bcurr <= "111111111111";
   s_ctrlin <= q_vdd;
    nextstate <= COMP;

when COMP =>
    setctrlin := 1;
   --s_ctrlin <= q_vss;
   --bcurr <= "000000000000";
   nextstate <= B01;
   bout <= b;

when B01 =>
   if(q_op_vcomp > q_vcm) then
      bcurr(0) <= '0'; --negative input, Va-Vb=positive
   else
```

```
      bcurr(0) <= '1'; --postive input, Va-Vb=negative
    end if;
    bcurr(1) <= '1'; --trekker fra vrefrange/4
    nextstate <= B02;

when B02 =>
    if(q_op_vcomp > q_vcm) then
      bcurr(1) <= not bcurr(0); --Va-Vb=positive
    else
      bcurr(1) <= bcurr(0); --Va-Vb=negative
    end if;
    bcurr(2) <= '1'; --trekker fra vrefrange/4
    nextstate <= B03;

when B03 =>
    if(q_op_vcomp > q_vcm) then
      bcurr(2) <= not bcurr(0); --negative input, Va-Vb=
         positive
    else
      bcurr(2) <= bcurr(0); --postive input, Va-Vb=negative
    end if;
    bcurr(3) <= '1'; --trekker fra vrefrange/4
     nextstate <= B04;

when B04 =>
    if(q_op_vcomp > q_vcm) then
      bcurr(3) <= not bcurr(0); --negative input, Va-Vb=
         positive
    else
      bcurr(3) <= bcurr(0); --postive input, Va-Vb=negative
    end if;
    bcurr(4) <= '1'; --trekker fra vrefrange/4
        nextstate <= B05;

when B05 =>
    if(q_op_vcomp > q_vcm) then
```

```
      bcurr(4) <= not bcurr(0); --negative input, Va-Vb=
         positive
   else
      bcurr(4) <= bcurr(0); --postive input, Va-Vb=negative
   end if;
   bcurr(5) <= '1'; --trekker fra vrefrange/4
   nextstate <= B06;


when B06 =>
   if(q_op_vcomp > q_vcm) then
      bcurr(5) <= not bcurr(0); --negative input, Va-Vb=
         positive
   else
      bcurr(5) <= bcurr(0); --postive input, Va-Vb=negative
   end if;
   bcurr(6) <= '1'; --trekker fra vrefrange/4
    nextstate <= B07;


when B07 =>
   if(q_op_vcomp > q_vcm) then
      bcurr(6) <= not bcurr(0); --negative input, Va-Vb=
         positive
   else
      bcurr(6) <= bcurr(0); --postive input, Va-Vb=negative
   end if;
   bcurr(7) <= '1'; --trekker fra vrefrange/4
       nextstate <= B08;


when B08 =>
   if(q_op_vcomp > q_vcm) then
      bcurr(7) <= not bcurr(0); --negative input, Va-Vb=
         positive
   else
      bcurr(7) <= bcurr(0); --postive input, Va-Vb=negative
   end if;
   bcurr(8) <= '1'; --trekker fra vrefrange/4
   nextstate <= B09;
```

```
when B09 =>
   if(q_op_vcomp > q_vcm) then
      bcurr(8) <= not bcurr(0); --negative input, Va-Vb=
         positive
   else
      bcurr(8) <= bcurr(0); --postive input, Va-Vb=negative
   end if;
   bcurr(9) <= '1'; --trekker fra vrefrange/4
     nextstate <= B10;


when B10 =>
   if(q_op_vcomp > q_vcm) then
      bcurr(9) <= not bcurr(0); --negative input, Va-Vb=
         positive
   else
      bcurr(9) <= bcurr(0); --postive input, Va-Vb=negative
   end if;
   bcurr(10) <= '1'; --trekker fra vrefrange/4
   nextstate <= B11;


when B11 =>
   if(q_op_vcomp > q_vcm) then
      bcurr(10) <= not bcurr(0); --negative input, Va-Vb=
         positive
   b(10) <= not bcurr(0);
   else
      bcurr(10) <= bcurr(0); --postive input, Va-Vb=negative
   b(10) <= bcurr(0);
   end if;
   bcurr(11) <= '1'; --trekker fra vrefrange/4
   nextstate <= SAMP;
   b(0) <= bcurr(0);
   b(1) <= bcurr(1);
   b(2) <= bcurr(2);
   b(3) <= bcurr(3);
   b(4) <= bcurr(4);
   b(5) <= bcurr(5);
   b(6) <= bcurr(6);
```

```
      b(7) <= bcurr(7);
      b(8) <= bcurr(8);
      b(9) <= bcurr(9);

   end case;
   end if;
   STATE <= NEXTSTATE;
   end process;
end ideal;
```

C.7 Output recorded to file, VHDL

```
library IEEE;
use IEEE.math_real.all;
use IEEE.electrical_systems.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use STD.TEXTIO.all;
use IEEE.std_logic_textio.all;
entity OutputRecord is
  port(
    clk : in std_logic;
    ctrlin : in std_logic;
    output : in std_logic_vector (0 to 11)

    );
end entity OutputRecord;

architecture struct of OutputRecord is

begin

  p1 : process (clk,ctrlin)
    file my_output : text open write_mode is "simulation/output.dat";
    variable my_ol : line;
    variable i : real := 0.0;
    variable temp : std_logic := '0';
    variable first : std_logic := '1';
  begin
```

```
    if ctrlin'event and ctrlin='0' then
    temp:='1';
    i:=3.0;
    end if;

    if clk'event and clk = '1' and temp='1' then
      if i < 1.0 then
      if first='0' then
      write(my_ol, output);
        writeline(my_output, my_ol);
     i := 0.0;
     temp:='0';
      else
     first:='0';
     i := 0.0;
     temp:='0';
      end if;
      else
        i := i- 1.0;
    end if;
    end if;


  end process p1;

end struct;

architecture structmc of OutputRecord is

begin

  p1 : process (clk,ctrlin)
    file my_output : text;
    variable my_ol : line;
    file my_backup : text;
    variable my_bl : line;
    variable i : real := 0.0;
    variable temp : std_logic := '0';
    variable first : std_logic := '1';
     variable havecopy : std_logic := '0';
     variable havemerged : std_logic := '0';
```

```
    variable nrlines : integer := 0;
    VARIABLE fstatus : File_open_status;
begin
  if havecopy='0' then
       File_open(fstatus, my_output, "simulation/outputmc.dat",
           read_mode);
       File_open(fstatus, my_backup, "simulation/backup.dat",
           write_mode);
       while not(endfile(my_output)) loop
       readline(my_output, my_bl);
       writeline(my_backup, my_bl);
       end loop;
       file_close(my_backup);
       file_close(my_output);
       havecopy := '1';
       File_open(fstatus, my_output, "simulation/outputmc.dat",
           write_mode);
   end if;
   if havemerged='0' and nrlines=128 then
       File_open(fstatus, my_backup, "simulation/backup.dat",
           read_mode);
       while not(endfile(my_backup)) loop
       readline(my_backup, my_bl);
       writeline(my_output, my_bl);
       end loop;
       file_close(my_backup);
       file_close(my_output);
       havemerged := '1';
   end if;


  if ctrlin'event and ctrlin='0' and havemerged='0' then
       temp:='1';
       i:=3.0;
  end if;

  if clk'event and clk = '1' and temp='1' and havemerged='0' then
       if i < 1.0 then
            if first='0' then
                 write(my_ol, output);
                 writeline(my_output, my_ol);
```

```
                    nrlines := nrlines + 1;
                    i := 0.0;
                    temp:='0';
110             else
                    first:='0';
                    i := 0.0;
                    temp:='0';
                end if;
115         else
                i := i- 1.0;
            end if;
    end if;


120
  end process p1;

end structmc;
```

D SAR ADC source code

D.1 Overall testbench, SPICE

```
testbench_SAR12bit_byEnNordlending

.param res=20n

*.option RGNDI=1G
*.option nowarn=240
*.option msgnode = 0
.option aex

.option RGNDI=1G
.option nowarn=240
.option msgnode = 0
.option reltol=res
.option vntol=res
.opt eps=res
*-------------------------------------------------------
* LIBRARIES
*-------------------------------------------------------
.inc source/saradc.cir
.lib key=mos ~/cmos090eldo/cmos090_tt.mod
*.lib key=mimcap ~/cmos090eldo/cmim.lib

*-------------------------------------------------------
* PARAMS1
*-------------------------------------------------------
.param vdd=1.2
.param vss=0
.param cm={vdd/2}

*-------------------------------------------------------
* PARAMS2
*-------------------------------------------------------
*#com
*maxim calculated 128 samples
.param fs = 1e6
.param fb = 492187.5
*.param fb = 257812.5
```

```
.param swing = 0.6
.param refn = 0.0
.param refp = 1.2
.param tsim={(129/fs)}
*#endcom
#com
*maxim calculated 4096 samples
.param fs = 1.000001536e6
.param fb = 497315.217
.param swing = 0.6
.param refn = 0.0
.param refp = 1.2
.param tsim={(4097/fs)}
#endcom


**clk v11
.param fclk = 13e6
.param clk_rf='transisiontime'
.param clk_std={(1/(2*fclk))-clk_rf}
.param clk_per = {1/fclk}
.param clk_hper = {0.5/fclk}

*-------------------------------------------------------
* INTERFACE CONVERTERS AND DIGITAL MODELS
*-------------------------------------------------------
.param transisiontime = 1n
*.param transisiontime = 0n

.model d2a_eldo d2a mode=std_logic vhi=vdd vlo=vss tcom=
   transisiontime
.model a2d_eldo a2d mode=std_logic vth={vdd/2} tcom=transisiontime
   cin=10f
.defhook d2a_eldo
.defhook a2d_eldo

.model lmod logic vhi='vdd' vlo='vss' vth='cm' trise=transisiontime
    tfall=transisiontime tpd=transisiontime cin=10f

*-------------------------------------------------------
* SOURCES
```

```
*-----------------------------------------------------
vclka clka vss pulse(vss vdd 0 clk_rf clk_rf clk_std clk_per)
delay1 clka clk {4n}

vinn vinn 0 sin (cm {swing*1} fb 0 0 0)
vinp vinp 0 sin (cm {swing*1} fb 0 0 180)

***increasing dnl/inl test
*vinp vinp 0 pwl(file="pwl.dat" col=1 istep=1)
*vinn vinn 0 pwl(file="pwl.dat" col=2 istep=1)
*.param tsim='128.25u'

vsssource vss 0 vss
vddsource vdd 0 vdd
vcmsource cm 0 cm



*-----------------------------------------------------
* Circuit
*-----------------------------------------------------
.param unit_cap=30f
.param switch_ron=100

xsaradc clk clka vdd vss vinp vinn cm ctrlin b00 b01 b02 b03 b04
    b05 b06 b07 b08 b09 b10 b11 outcomp saradc


*** Digital output ***
*.model OutputRecord(struct) macro lang=vhdlams
*yrec OutputRecord(struct) port: clk ctrlin (b00 b01 b02 b03 b04
    b05 b06 b07 b08 b09 b10 b11)
*** analog output ***
*.probe tran -R v(*)
*.probe tran -R S(*)



*** Digital output with Monte Carlo***
.model OutputRecord(structmc) macro lang=vhdlams
yrec OutputRecord(structmc) port: clk ctrlin (b00 b01 b02 b03 b04
    b05 b06 b07 b08 b09 b10 b11)
*.mc 50
```

```
115
*** SIMULATION ***
*.tran 5n 6u
.tran res {tsim}

120
*** Measurements ***
*** .xsaradc.
.defwave ref_current = 'abs(i(xsaradc.xrefgenp.vrefsrc))+abs(i(
   xsaradc.xrefgenn.vrefsrc))'
.extract mean(w(ref_current))
125 .defwave ref_dyn_current = 'abs(i(xsaradc.xrefgenp.vrefsrc))-abs(i(
   xsaradc.xrefgenp.vcmsrc))+abs(i(xsaradc.xrefgenn.vrefsrc))-abs(i
   (xsaradc.xrefgenn.vcmsrc))'
.extract mean(w(ref_dyn_current))
.defwave clk_current = 'abs(i(vclka))'
.extract mean(w(clk_current))

130 .defwave vdd_current = 'abs(i(vddsource))'
.extract mean(w(vdd_current))
.defwave vss_current = 'abs(i(vsssource))'
.extract mean(w(vss_current))
```

D.2 Comparator testbench, transient

```
testbenk


.option aex
5 .lib key=mos ~/cmos090eldo/cmos090_tt.mod

.inc latch.cir
.inc preamp.cir
.inc prepreamp.cir
10 *-------------------------------------------------------
* PARAMS
*-------------------------------------------------------
.option RGNDI=1G
.option nowarn=240
```

```
.option msgnode = 0
.option reltol=res
.option vntol=res
.opt eps=res

***Clk params
.param clk_f='13e6'
.param clk_rf={(1/(2*clk_f))/10}
.param clk_std={(1/(2*clk_f))-clk_rf}
.param clk_per = {1/clk_f}
.param clk_hper = {0.5/clk_f}

***Source size
.param vs=0.0
.param vd=1.2
.param iref = 1u
.param tbias = 1.2
.param vinp = 0.61
.param vinn = 0.59
.param cm = 0.6

***Out params
.param rout=100k
.param cout='50f'

***Noise capacitor sizes
.param cnoise=45f
.param cnoise2=10f

***Tran size
.param wminsize={0.12}
.param lminsize={0.1}
*.step param wminsize 0.12 '2*0.12' '0.12/4'

*-------------------------------------------------------
* Transistor sizes
*-------------------------------------------------------
*.param upun=3.29
.param upun=3.75

**mirror transistors
```

```
.param nwidth = {1*wminsize}
.param nlength = {1*lminsize}
.param pwidth = {1*wminsize*upun}
.param plength = {1*lminsize}

*-------------------------------------------------------
* Biasing
*-------------------------------------------------------
vtbias ntbias vss tbias

xp00cmirr vrefppa1 vrefppa1 vdd vdd psvt w='pwidth' l='plength'
   nfing=1.0 mult=1 srcefirst=1.0 mismatch=1.0
icmirr vrefppa1 vss iref

xn004cm vrefppa2 vrefppa2 vss vss nsvt w=nwidth l=nlength nfing=1.0
    mult=1 srcefirst=1.0 mismatch=1.0
icm4 vdd vrefppa2 iref

*-----SOURCES-------------------------------------
vss vss 0 vs
vdd vdd vss vd

vclka clka vss pulse(vs vd clk_std clk_rf clk_rf clk_std {1/clk_f})
delay1 clka clk {1.5*clk_rf}



**Linear increasing
vinp vinp vss pwl (0 {cm+5m} {tsim} {cm-5m})
vinn vinn vss pwl (0 {cm-5m} {tsim} {cm+5m})
.param tsim = {100*clk_per}

**BER-testing
.param cm=0.6
*.step param cm 0.1 1.1 0.25
*vinp vinp vss pwl (0 {cm-10e-3} {0.75*clk_per} {cm-10e-3} {0.8*
   clk_per} {cm+1e-3} {1.75*clk_per} {cm+1e-3} {1.8*clk_per} {cm-10
   e-3} {2.75*clk_per} {cm-10e-3} {2.8*clk_per} {cm+1e-4} {3.75*
   clk_per} {cm+1e-4} {3.8*clk_per} {cm-10e-3} {4.75*clk_per} {cm
   -10e-3} {4.8*clk_per} {cm+1e-5} {5.75*clk_per} {cm+1e-5} {5.8*
   clk_per} {cm-10e-3} {6.75*clk_per} {cm-10e-3} {6.8*clk_per} {cm
   +1e-6} {7.75*clk_per} {cm+1e-6} {7.8*clk_per} {cm+10e-3} {8.75*
```

```
    clk_per} {cm+10e-3} {8.8*clk_per} {cm-1e-3} {9.75*clk_per} {cm-1
    e-3} {9.8*clk_per} {cm+10e-3} {10.75*clk_per} {cm+10e-3} {10.8*
    clk_per} {cm-1e-4} {11.75*clk_per} {cm-1e-4} {11.8*clk_per} {cm
    +10e-3} {12.75*clk_per} {cm+10e-3} {12.8*clk_per} {cm-1e-5}
    {13.75*clk_per} {cm-1e-5} {13.8*clk_per} {cm+10e-3} {14.75*
    clk_per} {cm+10e-3} {14.8*clk_per} {cm-1e-6} {15.75*clk_per} {cm
    -1e-6} {15.8*clk_per} {cm+10e-3} {16.75*clk_per} {cm+10e-3})
*vinn vinn vss dc cm

*old vinp
*vinp vinp vss pwl (0 {cm-5e-5} {0.75*clk_per} {cm-5e-5} {0.8*
    clk_per} {cm+1e-3} {1.75*clk_per} {cm+1e-3} {1.8*clk_per} {cm-5e
    -5} {2.75*clk_per} {cm-5e-5} {2.8*clk_per} {cm+1e-4} {3.75*
    clk_per} {cm+1e-4} {3.8*clk_per} {cm-5e-5} {4.75*clk_per} {cm-5e
    -5} {4.8*clk_per} {cm+1e-5} {5.75*clk_per} {cm+1e-5} {5.8*
    clk_per} {cm-5e-5} {6.75*clk_per} {cm-5e-5} {6.8*clk_per} {cm+1e
    -6} {7.75*clk_per} {cm+1e-6} {7.8*clk_per} {cm+5e-5} {8.75*
    clk_per} {cm+5e-5} {8.8*clk_per} {cm-1e-3} {9.75*clk_per} {cm-1e
    -3} {9.8*clk_per} {cm+5e-5} {10.75*clk_per} {cm+5e-5} {10.8*
    clk_per} {cm-1e-4} {11.75*clk_per} {cm-1e-4} {11.8*clk_per} {cm
    +5e-5} {12.75*clk_per} {cm+5e-5} {12.8*clk_per} {cm-1e-5}
    {13.75*clk_per} {cm-1e-5} {13.8*clk_per} {cm+5e-5} {14.75*
    clk_per} {cm+5e-5} {14.8*clk_per} {cm-1e-6} {15.75*clk_per} {cm
    -1e-6} {15.8*clk_per} {cm+5e-5} {16.75*clk_per} {cm+5e-5})


*.param tsim = {17*clk_per*1}


*-----In/out-------------------------------------
*---ResCap load
*rout04 vout vss rout
*rout05 vnout vss rout
cout04 vout vss cout
cout05 vnout vss cout

*--------CIRCUIT----------------------------------------
xprepreamp vdd vss vinp vinn vrefppa1 vrefppa2 ppa_outp ppa_outn
    prepreamp
xpreamp vdd vss ppa_outp ppa_outn clk vrefppa1 pa_outp pa_outn
    preamp
```

```
xlatch vdd vss pa_outp pa_outn clka ntbias vout1 vnout1 latch
xpostlatch vdd vss vout1 vnout1 vout vnout postlatch2

*--------SIMULATION--------------------------------------
.param res=10n

*.noisetran fmin=0 fmax={1.5*13e6} nbrun=30 mrun
.tran {res} {tsim}
.mc 500 all

.option OUT_STEP={res}
.plot v(vout)

*.alter
*.lib key=mos ~/cmos090eldo/cmos090_ff.mod
*.alter
*.lib key=mos ~/cmos090eldo/cmos090_ss.mod
*.alter
*.lib key=mos ~/cmos090eldo/cmos090_fs.mod
*.alter
*.lib key=mos ~/cmos090eldo/cmos090_sf.mod

.defwave ppa_current = {abs(id(xprepreamp.xp00.m1))+abs(id(
   xprepreamp.xn004.m1))}
.extract mean(w(ppa_current))
.defwave pa_current = {abs(id(xpreamp.xp00.m1))+abs(i(xpreamp.r04)+
   i(xpreamp.r05))}
.extract mean(w(pa_current))
.defwave ampcurrent = {abs(w(ppa_current))+abs(w(pa_current))}
.extract mean(w(ampcurrent))
.defwave latchcurrent = {abs(id(xlatch.xp05.m1))+abs(id(xlatch.xp06
   .m1))+abs(id(xpostlatch.xp01.m1))+abs(id(xpostlatch.xp02.m1))}
.extract mean(w(latchcurrent))
.defwave totalcurrent = {abs(w(ampcurrent))+abs(w(latchcurrent))}
.extract mean(w(totalcurrent))
.defwave biascurrent = {abs(i(icmirr))+abs(i(icm4))}
.extract mean(w(biascurrent))

*.defwave
.defwave intvout= integ(v(vout))
```

```
.defwave offperiods = '(w(intvout)*1e6/0.04655)-50'
.defwave voffset = '(w(offperiods)*100e-6)'
.extract w(voffset)
```

D.3 Comparator testbench, AC

```
testbenk


.option aex
.lib key=mos ~/cmos090eldo/cmos090_tt.mod

.inc latch.cir
.inc preamp.cir
.inc prepreamp.cir
*-------------------------------------------------------
* PARAMS
*-------------------------------------------------------
.option RGNDI=1G
.option nowarn=240
.option msgnode = 0
.option reltol=res
.option vntol=res
.opt eps=res


***Clk params
.param clk_f='13e6'
.param clk_rf={(1/(2*clk_f))/10}
.param clk_std={(1/(2*clk_f))-clk_rf}
.param clk_per = {1/clk_f}
.param clk_hper = {0.5/clk_f}


***Source size
.param vs=0.0
.param vd=1.2
.param iref = 1u
.param tbias = 1.2
.param vinp = 0.61
```

```
.param vinn = 0.59
.param cm = 0.6


***Out params
.param rout=100k
.param cout='50f'


***Noise capacitor sizes
.param cnoise=45f
.param cnoise2=10f




***Tran size
.param wminsize={0.12}
.param lminsize={0.1}
*.step param wminsize 0.12 '2*0.12' '0.12/4'

*-------------------------------------------------------
* Transistor sizes
*-------------------------------------------------------
.param upun=3.75

**mirror transistors
.param nwidth = {1*wminsize}
.param nlength = {1*lminsize}
.param pwidth = {1*wminsize*upun}
.param plength = {1*lminsize}


*-------------------------------------------------------
* Biasing
*-------------------------------------------------------
vtbias ntbias vss tbias

xp00cmirr vrefppa1 vrefppa1 vdd vdd psvt w='pwidth' l='plength'
    nfing=1.0 mult=1 srcefirst=1.0 mismatch=1.0
icmirr vrefppa1 vss iref
```

```
xn004cm vrefppa2 vrefppa2 vss vss nsvt w=nwidth l=nlength nfing=1.0
    mult=1 srcefirst=1.0 mismatch=1.0
icm4 vdd vrefppa2 iref

*-----SOURCES-------------------------------------
vss vss 0 vs
vdd vdd vss vd

vclka clka vss pulse(vs vd clk_std clk_rf clk_rf clk_std {1/clk_f})
delay1 clka clk {1.5*clk_rf}

**AC analysis
vicm vicm vss dc cm
vinp vinp vicm ac 0.5
vinn vinn vicm ac -0.5

*-----In/out--------------------------------------
*---ResCap load
*rout04 vout vss rout
*rout05 vnout vss rout
*cout04 vout vss cout
*cout05 vnout vss cout
xp00cmirr vdd pa_outp vdd vdd psvt w='2*pwidth' l='2*plength' nfing
   =1.0 mult=1 srcefirst=1.0 mismatch=1.0
xp00cmirr vdd pa_outn vdd vdd psvt w='2*pwidth' l='2*plength' nfing
   =1.0 mult=1 srcefirst=1.0 mismatch=1.0

*--------CIRCUIT-------------------------------------
xprepreamp vdd vss vinp vinn vrefppa1 vrefppa2 ppa_outp ppa_outn
   prepreamp
xpreamp vdd vss ppa_outp ppa_outn clk vrefppa1 pa_outp pa_outn
   preamp

*--------SIMULATION----------------------------------
.ac dec 10 1 100e9


.defwave in=abs(v(vinp)-v(vinn))
.plot wdb(in)
.defwave out=abs(v(pa_outp)-v(pa_outn))
.plot wdb(out)
```

```
.defwave gain=w(out)/w(in)
.plot wdb(gain)

.defwave ppa_out=abs(v(ppa_outp)-v(ppa_outn))
.plot wdb(ppa_out)
.defwave ppa_gain=w(ppa_out)/w(in)
.plot wdb(ppa_gain)

.defwave pa_gain=w(out)/w(ppa_out)
.plot wdb(pa_gain)

.defwave ppa_vn01=abs(v(xprepreamp.n02)-v(xprepreamp.n01))
.plot wdb(ppa_vn01)
.defwave ppa_gain01=w(ppa_vn01)/w(in)
.plot wdb(ppa_gain01)

.defwave pa_vn01=abs(v(xpreamp.n02)-v(xpreamp.n01))
.plot wdb(pa_vn01)
.defwave pa_gain01=w(pa_vn01)/w(ppa_out)
.plot wdb(pa_gain01)

.defwave total_gain=abs(v(xlatch.n02)-v(xlatch.n01))
.plot wdb(pa_vn01)

*.probe ac all

.noise v(pa_outp,pa_outn) vinp 20
.plot noise inoise onoise
*.noise v(xpreamp.n04,xpreamp.n05) vinp 50
*.defwave wintinoise=inoise
*.extract label=intinoise integ(w(wintinoise),1,10e9)
.defwave in_noise={onoise/ppa_gain}
.plot w(in_noise)

***Plot phase
.defwave pa_out_phase=vp(pa_outp,pa_outn)
.plot w(pa_out_phase)
.defwave ppa_out_phase=vp(ppa_outp,ppa_outn)
.plot w(ppa_out_phase)

```

```
.extract rms(inoise,1,{10g})
.extract label=out_noise_ext rms(onoise,1,{10g})
.extract rms(inoise,1,{65meg})
.extract rms(onoise,1,{65meg})
.extract rms(inoise,1,{1meg})
.extract rms(onoise,1,{1meg})

*.extract label=dcgain yval(wm(ppa_gain),1000)
.extract label=out_noise_over_DCgain 'extract(out_noise_ext)/yval(
   wm(gain),1000)'
*.plot w(out_noise)

.extract label=dcgain yval(wm(gain), 1000)
.extract label=ppa_dcgain yval(wm(ppa_gain), 1000)
.extract label=ppa_n01_dcgain yval(wm(ppa_gain01), 1000)

.param res=1n
.option OUT_STEP={res}

#com
.step param(P(cnoise)) LIST
+(0)
+(2f)
+(4f)
+(8f)
+(16f)
+(32f)
+(64f)
+(128f)
#endcom
*.alter
*.step param rval2_ppa 0 100k 10k
*#com
.alter
.lib key=mos ~/cmos090eldo/cmos090_ff.mod
.alter
.lib key=mos ~/cmos090eldo/cmos090_ss.mod
.alter
.lib key=mos ~/cmos090eldo/cmos090_fs.mod
.alter
.lib key=mos ~/cmos090eldo/cmos090_sf.mod
```

```
*#endcom
*.step param cm 0.0 1.2 0.025
```