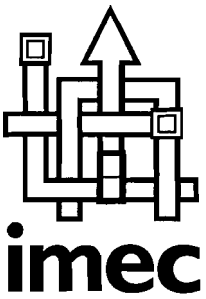
High-level modeling of high-speed flash A/D converters

de Jong, P.W.T.

Award date:
2000

7681



High-level modeling of high-speed flash A/D converters

P.W.T. de Jong

11th April 2000

Eindhoven University of Technology
Department of Electrical Engineering
Mixed-signal Microelectronics Group
Den Dolech 2
Postbus 513
5600 MB Eindhoven
The Netherlands

IMEC vzw
DESICS division - MIRA group
Kapeldreef 75
B-3001 Leuven
Belgium

Coaches (T.U. Eindhoven):	dr.ir. J.A. Hegt
	prof.dr.ir. A.H.M. van Roermund
	dr.ir. L.K.J. Vandamme
Coaches (IMEC, Leuven):	dr.ir. S. Donnay
	dr.ir. P. Wambacq
	ir. J.P.R Compiet

Abstract

Improving technologies allow to integrate more and more complex systems on a chip. This demands complex design methodologies, especially for analog and mixed analog/digital systems. So the need for high-level simulation tools is growing, that can explore and trade-off different architectures in a fast and accurate way. FAST (Front-end Architecture Simulation Tool) is such a tool, which is currently being developed at the MIRA group of IMEC. This tool is used for end-to-end simulations of digital telecommunication transceivers. The Analog-to-Digital Converter (ADC) is important for complete front-end architecture simulations including the analog and digital part. This report describes a high-level model for n-bit flash ADCs, that can be implemented in FAST.

The model meets with the following demands. It is flexible, it is easy to change the architecture and the number of bits. It is efficiently implemented in C++, and it is based only on circuit and device parameters, such as transconductances, capacitances, poles and zeros, in order to give a designer a suitable set of design parameters.

The non-idealities of the ADC are divided in functional blocks, that are modeled separately. The following blocks with non-ideal behavior are discussed extensively: Input feedthrough, mismatches, clock jitter and the comparator function.

The feedthrough of the input signal on the reference ladder is caused by the capacitive coupling between the input nodes of the comparator, that couples the input to the resistor ladder. The input feedthrough is dependent on the number of bits, the comparator and the value of the resistor in the reference ladder. The presented model can deal with all different values of the resistor, the number of bits and the types of comparators.

The main effect of mismatches between transistors in a comparator is an offset voltage at the input. A new method is described that can predict the variance of the offset voltage of a comparator in a faster and more accurate way.

All blocks in the clock path, from the oscillator to the ADC included, contribute to the clock jitter. A summary is given about the various aspects of clock jitter and a realistic way to model the clock jitter is described.

The comparator is the most important part of the ADC. Three different models are studied and finally the best model is selected.

To verify the validity of the model, a comparison is made with a 4-bit ADC, which is designed in HSPICE. The same input signal is applied to HSPICE and the C++ model and the output codes are compared. The results show a good agreement between the simulated output codes of HSPICE and the model. The model is at least 6000 times faster than HSPICE.

Finally a program is written, that determines the performance of the ADC after every simulation, by calculating the SNR and the INL/DNL values.

An article is published about this research at the "Southwest Symposium on Mixed-Signal Design 2000", with the title "High-level modeling of a high-speed flash A/D converter for mixed-signal simulations of digital telecommunication front-ends". This paper can be found at the end of this report. An updated version of this paper is written, also located at the end of this report.

Abbreviations and Conventions

This page contains abbreviations and conventions, that are used throughout the report.

The conception of *sampling* refers to the sampling of the ADC and also to the making of discrete values out of continuous values used in simulations. Therefore, to avoid confusion, the term sampling is used as little as possible. *Simulation frequency* refers to the time-steps used in a simulation and *clock frequency* refers to the sampling of the ADC.

A complete comparator consists out of three parts, the input stage, the regenerative comparator (= actual comparator) and the output latch. *Comparator* refers to the complete comparator and *regenerative comparator* is used, when only the actual comparator is meant.

$clklength$	Number of simulation points in a clock-period
$clkperiods$	Number of clock-periods in a simulation
ADC	Analog-to-Digital converter
DFT	Discrete Fourier Transform
DNL	Differential NonLinearity
ENOB	Effective Number Of Bits
FFT	Fast Fourier Transform
INL	Integral NonLinearity
LSB	Least Significant Bit
PLL	Phase Locked Loop
S/H	Sample and Hold
SNR	Signal to Noise Ratio
SFDR	Spurious Free Dynamic Range
T_{clk}	The clock period of the ADC
$T_{duration}$	The length in seconds of the simulated time interval
T_{sim}	The time in seconds between the used points in a simulation
VCO	Voltage Controlled Oscillator

Contents

1 Introduction **1**

2 Literature study **2**

3 The ADC circuit **3**

3.1 Block diagram of the flash ADC 3

3.2 The comparator 4

3.2.1 Principle of the regenerative comparator 4

3.2.2 Circuit of the comparator 5

3.3 Thermometer decoder 6

4 The high-level model **8**

4.1 Block processing 8

4.2 The block diagram of the ADC 9

4.3 Filters in C++ 10

4.4 Differential equations 11

5 Reference ladder with input feedthrough **12**

5.1 Input signal feedthrough 12

5.2 Simulation results 14

5.3 Maximum input feedthrough 15

5.4 Input feedthrough in C++ 16

5.5 The total reference ladder 17

6 The comparator **18**

6.1 Differential equation model with a sine function 18

6.1.1 The model 18

6.1.2 Simulation results 19

6.2 Comparator with 4-MOS model 20

6.2.1 The model 20

6.2.2 Simulation results 21

6.3 Comparator model with filter 22

6.3.1 The model 22

6.4 Conclusions 23

7 Clock jitter **24**

7.1 Jitter in theory 24

7.1.1 Phase noise in the oscillator 24

7.1.2 Phase noise in the PLL 24

7.1.3 Internal jitter 25

7.1.4 Maximum jitter 25

7.1.5 Noise power 26

7.2 Modeling clock jitter 27

7.2.1 The place of clock jitter in the ADC 27

7.2.2 Clock jitter implementation 27

7.2.3 Conclusions 29

8 Mismatch in the comparator **30**

8.1 Theory 30

8.2 Mismatch in the comparator 30

8.3 Offset voltage calculation method 1 31

8.4 Offset voltage calculation method 2 32

8.5 Simulation results 34

9 Calculating the ADC performance **36**

9.1 Signal to Noise Ratio 36

9.2 INL and DNL 37

10 Simulation results **40**

10.1 Simulation parameters 40

10.2 Accuracy 40

10.3 Simulation time 42

11 Conclusions **44**

12 Further investigations **45**

1 Introduction

The fast growing market of telecommunications demands faster and cheaper ways to design front-ends of transceivers for digital telecommunications. This requires a high level simulation environment, that can simulate complete end-to-end systems, including the analog front-end, mixed-signal and digital functionality. These tools must be able to explore and trade-off different architectures in a fast and accurate way. The Front-end Architecture Simulation Tool (FAST), that is currently being developed at the MIRA group at IMEC, is such a mixed-signal simulation tool.

The goal of this research is to make a high level model of an n-bit flash ADC, that matches as good as possible the transistor level circuit. The model has to include non-idealities, such as mismatches, non-linearities, input feedthrough and clock jitter. Only circuit and device parameters can be used, in order to give a designer a suitable set of design parameters. The model has to be efficiently implemented in C++. It must be easy to change the building blocks, number of bits and the architecture.

As reference case a 4-bit, 400MHz, flash ADC is available, which is designed in HSPICE in 0.35 μm digital CMOS technology. The simulation results of the C++ model must accurately match the HSPICE simulations. The first step is to make an accurate model, the next step is make that model faster.

First, the different methods of high level modeling are treated in the literature study of chapter 2. Secondly, the HSPICE model of the ADC is described in chapter 3. Subsequently the non-idealities of the ADC are divided in functional blocks in chapter 4. The functional blocks are input signal feedthrough, the comparator, clock jitter and the mismatches. These are described in chapter 5 to 8. The structures of these chapters are similar, first the model is described, next simulation results are presented.

After each simulation in C++ the INL/DNL and the SNR are calculated with the method described in chapter 9. Simulation results of the C++ model are compared with HSPICE simulations in chapter 10. Conclusions are drawn in chapter 11. Finally topics for further investigation are listed in chapter 12.

At the end of some sections the location and name of the described program is cited, for example: file:/imec/users/dejong/c/totaladc.cpp

2 Literature study

A literature study is performed before starting to design a new model of an ADC. There is looked for a model that satisfies the following demands: The model must be accurate, flexible, include all non-idealities and based on circuit and device parameters. Not only models of complete ADCs are of interest, but also models of a single comparator, because the comparator will point out to be the most difficult and important part of an ADC. The types of models can be divided in groups. A good overview of the different types is given in [1]. Various types are listed here.

Circuit model: A circuit model is an equivalent circuit with less and usually more ideal circuit elements, which approximate the original behavior of the original circuit. But it is difficult to model second order effects such as mismatches and feedthrough. Circuit models for a comparator are described in [2] and [3]. These models are not suitable, because second order behavior is not taken into account. Circuit model is sometimes called macromodel.

Statistical model: A statistical model tries to model every block by its statistical behavior. In [4], [5], [6] and [7] models are presented that use a static ideal ADC as base. All the non-idealities, such as mismatch and clock jitter, are added as statistical errors to the ideal ADC. This kind of model is not suitable, because the statistical property demands much simulations to prove the correctness of the model.

Analytical model: Every block of the ADC is modeled by a mathematical expression. The analytical models can be divided in two groups:

Explicit model: This type of model calculates the output directly from the input with a mathematical expression. A closed form expression is needed, that describes the input-output relation. However this is not always possible. Models that calculate the output transient of a comparator are given in [8] and [9]. The models use too less parameters and are therefore not accurate.

Implicit model: When no explicit model can be found, then a mathematical model description can be used, such as differential equations and filters in the s or z-domain. In [10] an accurate implicit model is presented. This model is not suitable, because it makes use of parameters that can not directly be extracted from HSPICE.

Tabular model: First the circuit is simulated in HSPICE and the behavior is stored in a table. Then during a simulation the needed values are fetched from the table. This method can model difficult behavior very easily. The model only describes the static behavior and it is not flexible, because the table has to be extracted for every different configuration of the ADC. A good example of a tabular model is [11].

There are no models available that satisfy the demands. So a new model has to be developed. The implicit and explicit behavioral models are probably the best approaches.

3 The ADC circuit

The circuit of the flash ADC is presented in this chapter. There is a HSPICE netlist available for simulation purposes. In section 3.1 a block diagram is given of the total ADC. Thereafter, two important parts of the ADC are explained in section 3.2 and 3.3, namely the comparator and the thermometer decoder.

3.1 Block diagram of the flash ADC

For an n-bits flash ADC, the input signal is applied to $2^n - 1$ comparators. The comparators compare the input signal with $2^n - 1$ different reference voltages. These reference voltages are made by a reference ladder, that consist out of 2^n resistors. The endings of the reference ladder are connected to two reference voltage sources, V_{ref}^+ and V_{ref}^-. The reference voltages are equal to the input range of the ADC ($V_{inputrange} = V_{ref}^+ - V_{ref}^-$) , see Figure 1.

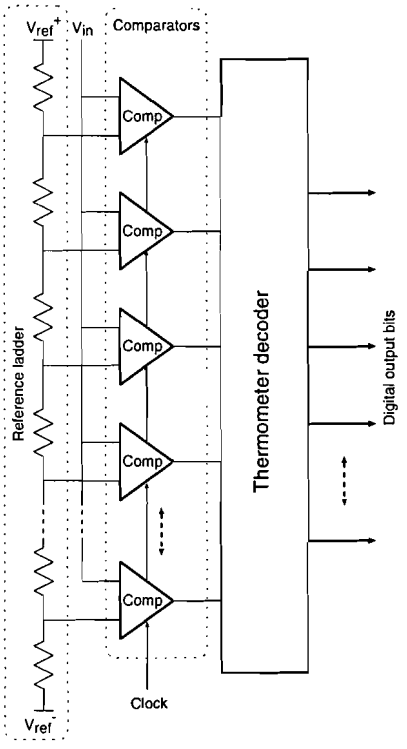


Figure 1: General architecture of a flash ADC.

The flash ADC can be extended easily for different numbers of bits. However, a complete simulation in HSPICE, including the encoding part, with more than 4-bits is very difficult, because of the long simulation time which results in memory problems. The final C++ model does not have this problem. The design of the reference ladder is very straightforward, in contrast with the comparator and the thermometer decoder. The comparator is described in more detail in section 3.2 and the thermometer decoder in section 3.3.

file:/imec/other/adsba/STUDENTS/dejong/spice/cmos035/flash/flash4b.sp
file:/imec/other/adsba/STUDENTS/dejong/spice/cmos035/reflad/reflad_def4b.sp

3.2 The comparator

A regenerative comparator is used because it is the fastest comparator. First the principle of the regenerative comparator is explained and subsequently this theory is used to explain the comparator circuit.

3.2.1 Principle of the regenerative comparator

A regenerative comparator consists of three parts, the input stage, the actual regenerative comparator and the output latch, see Figure 2

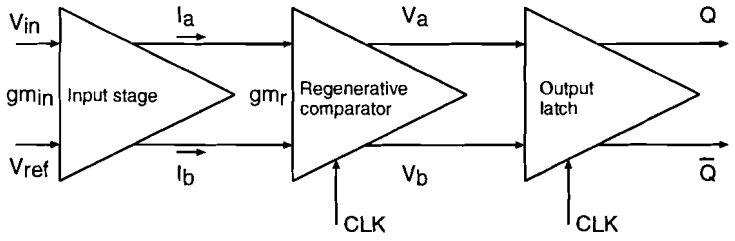


Figure 2: General architecture of a regenerative comparator.

As is shown, the complete comparator is designed differentially, because this minimizes the effects of mismatches. The input stage is a continuous time amplifier with a differential input voltage and a differential output current. The transconductance of the input stage is gm_{in}.

The regenerative comparator compares the output currents from the input stage by amplifying the difference between the currents in a circuit with positive feedback. This circuit consists out of a clock switch and two inverting amplifiers, which are placed in a loop, see Figure 3.

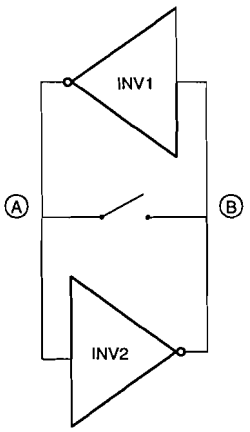


Figure 3: The principle of regenerative amplification.

The following steps are made for one comparison. At the beginning the switch is closed. The input signal from the input stage is injected in the loop and causes a voltage difference between the nodes A and B. When the switch is opened, the voltage difference is amplified by the two inverting amplifiers. This process is called the regeneration. At the end of this regeneration the digital output is available at the nodes A and B. For the next compare phase the nodes A and B are reset by closing the switch.

From now on, *regenerative comparator* refers only to the regenerative part of the comparator, and *comparator* refers to the complete comparator.

The output latch conditions the output of the regenerative comparator, in a way that it can be used by digital circuits.

3.2.2 Circuit of the comparator

Figure 4 shows the input stage and the regenerative comparator. The input stage is a transconductor, that converts the input voltage difference ($v_{in} - v_{ref}$) to a current difference ($i_{in\ a} - i_{in\ b}$). The transistors $M2a, b$ and $M3a, b$ are identical and form two current mirrors. The gm of the input transistors $M1a, b$ is the gain factor gm_{in} of the input stage.

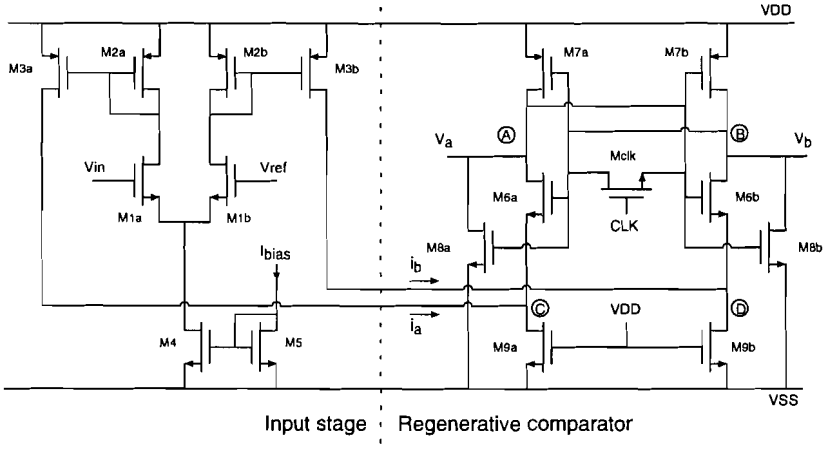


Figure 4: Circuit diagram of the input stage and the regenerative comparator of comparator.

The transistor pairs $M6a$, $M7a$ and $M6b$, $M7b$ form two inverters. The output of one inverter is connected to the input of the other. In between is the clock transistor $Mclk$. When $Mclk$ is closed the regenerative comparator is in the reset phase. The transistors $M9a$ and $M9b$ convert the differential current from the input stage to a voltage difference between nodes C and D and consequently in a voltage difference between nodes A and B. When $Mclk$ is opened the initial voltage difference between A and B is amplified. The transistors $M6a$ en $M6b$ can not pull the output to VSS because they are limited by the drain source voltage of $M9$, therefore the transistors $M8a$ and $M8b$ are added. These transistors can decline the output voltage to VSS.

Figure 5 shows a typical output response of the outputs of the regenerative comparator. For $t < 0$ the comparator is in the reset phase. At $t = 0$ the clock voltage starts decreasing and the comparator starts comparing. During the reset phase, the output voltage Va and Vb are at approximately half the VDD, but start now to increase or decrease to VDD or VSS, depending on the sign of the input voltage of the comparator. The little bump after $t = 0$ is explained in the following way. The clock input is coupled capacitively with the output nodes. A change of the clock voltage causes a common mode effect on the the nodes A and B. This will not affect the accuracy of the comparator.

The output of the regenerative comparator is only valid at the end of the compare phase. So the output has to be latched in order to use the output. The circuit of the output latch is plotted in Figure 6. Two separate output latches are connected to the outputs of the regenerative comparator, so that the capacitive load of the comparator will be symmetrical. When v_{clock} is low, the input voltage is passed to the output. The output voltage from the latch is preserved, when v_{clock} is high due to the capacitive

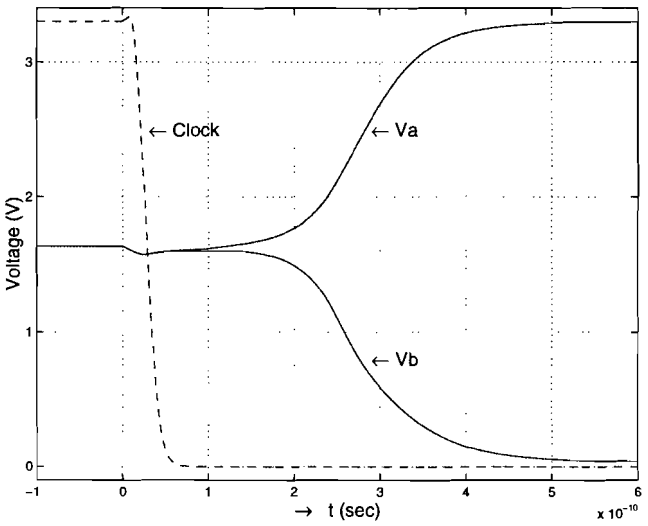


Figure 5: Typical output response of the regenerative comparator.

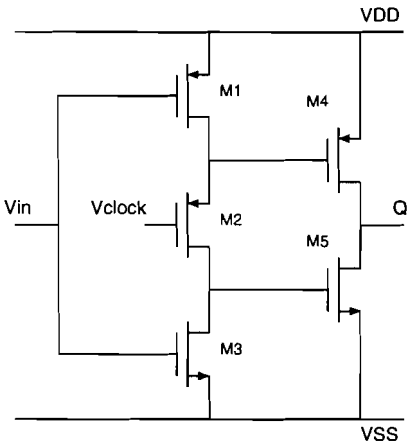


Figure 6: Circuit diagram of the output latch.

load of the gates of $M4$ and $M5$. The regenerative comparator and the output latch use the same clock.

file:/imec/other/adsba/STUDENTS/dejong/spice/cmos035/comparator/comparator.sp

3.3 Thermometer decoder

The comparators in a flash ADC produce an output pattern that is known as a thermometer code. Every time the input signal reaches a new reference level a '1' is added to the output code of the comparators. The thermometer decoder converts the $2^n - 1$ output codes of the comparators into a normal binary code with n-bits.

A comparator, that generates a false response, causes a bubble in the thermometer code pattern. Therefore, a thermometer decoder with error correction is used. Figure 7 shows a thermometer decoder, using a Wallace tree ([12]). The Wallace tree performs also error correction. Every block in Figure 7 is an adder, that counts the number of logical ones at its entries and outputs a two bit binary coded output.

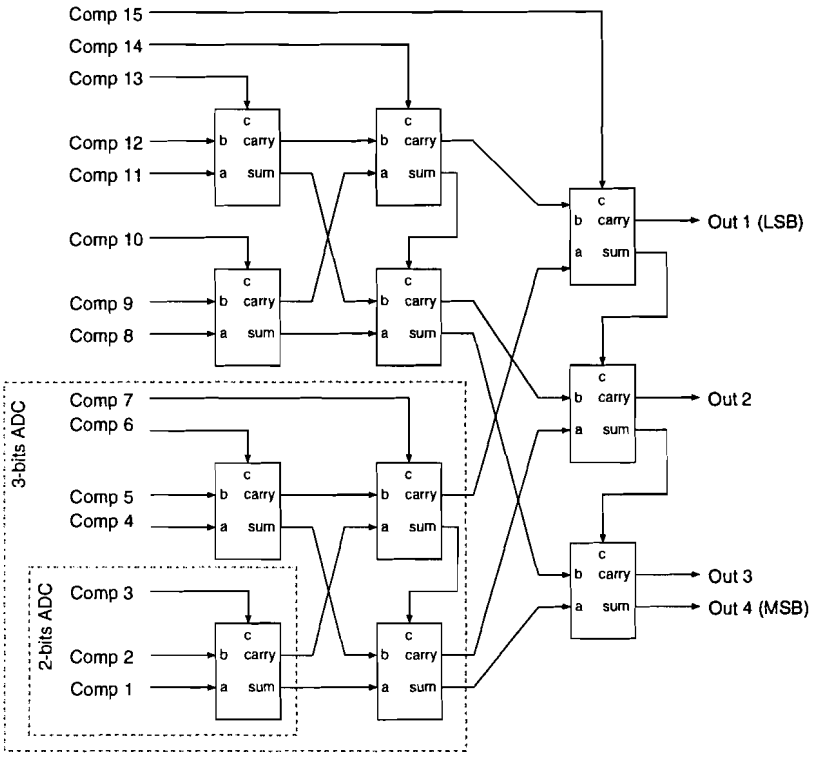


Figure 7: Wallace tree thermometer decoder for a 4-bit ADC, only full adders are used.

every next stage of adders decreases the number of codes. This Wallace tree is flexible, because it can easily be adapted to the number of bits of the ADC.

file:/imec/other/adsba/STUDENTS/dejong/spice/cmos035/encode/encode_wallace.sp

4 The high-level model

The outline of the ADC model is given in this chapter. First in section 4.1 the used method of the simulation is described, namely block processing. Secondly the ADC is divided in functional blocks in section 4.2. In the sections 4.3 and 4.4 is described how filters and differential equations are solved in the C++ model.

4.1 Block processing

The simulation method used by FAST for the ADC is called block processing, see [13]. An input array is loaded and processed and afterwards passed on to the next block. In case of the ADC, an array with the analog input voltage is loaded and converted to an array with the digital output code.

Three important simulation timing parameters must be adjusted for every simulation, see also figure 8:

1. $T_{duration}$: The duration of the simulation.
2. T_{sim} : The time step used by the simulation.
3. T_{clk} : The clock time of the ADC. $1/T_{clk}$ is the clock frequency of the ADC.

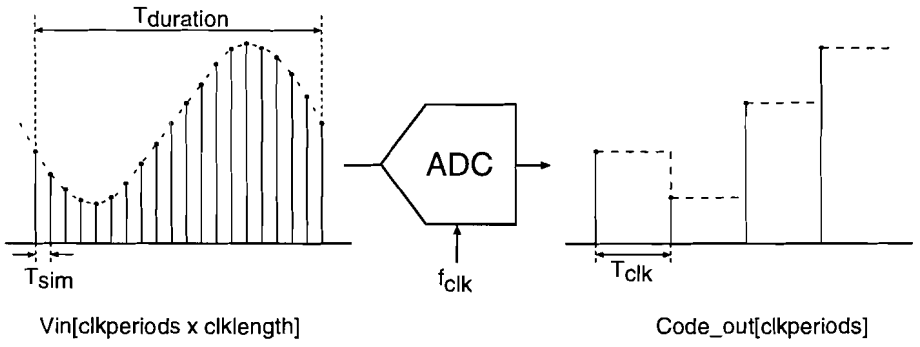


Figure 8: The definition of the three timing parameters $T_{duration}$, T_{sim} and T_{clk} necessary for every simulation.

The number of samples per clock period ($clklength$) is given by:

$$clklength = \frac{T_{clk}}{T_{sim}} \tag{1}$$

And the total amount of clock periods is called $clkperiods$:

$$clkperiods = \frac{T_{duration}}{T_{clk}} \tag{2}$$

The input array of the ADC contains $clklength$ elements per clocking period and outputs one value per clocking period. So the number of input values is $clklength \cdot clkperiods$ and the number of output values is $clkperiods$.

file:/imec/other/adsba/STUDENTS/dejong/c/include/globals.h

4.2 The block diagram of the ADC

The ADC circuit described in chapter 3 is divided in blocks. The ADC model is hierarchically built-up, see figure 9.

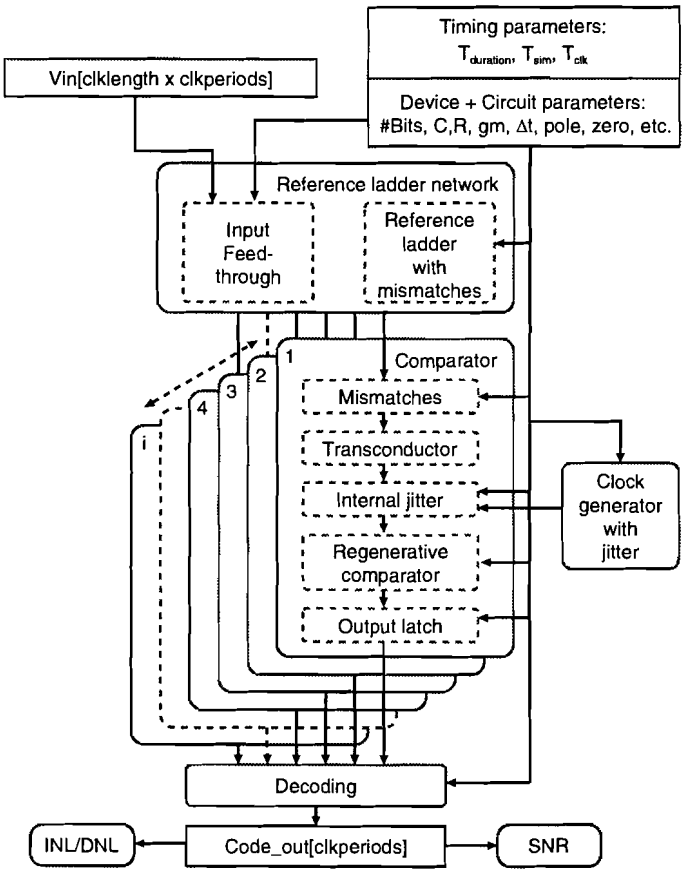


Figure 9: The block diagram of the ADC in C++.

The input of the ADC is an array with the input signal, with $clkperiods \cdot clklength$ elements, and a list of parameters. These are timing parameters as well as device and circuit parameters. The timing parameters are described in section 4.1. The device and circuit parameters are derived from simple HSPICE operation point and AC simulations. The following chapters will describe how this is done in more detail. The output of the ADC is an array called $Code\_out$ with $clkperiods$ elements. The device and circuit parameters are distributed over all the blocks in the model.

The reference ladder constructs $2^n - 1$ different reference voltages with $clklength \cdot clkperiods$ elements, that are passed on to the comparators. The reference ladder is composed with resistors with mismatches, that make $2^n - 1$ DC voltages. An unwanted effect in the reference ladder is the feedtrough from the input signal on the reference ladder. This is caused by the capacitive coupling of the the two inputs of a comparator, that couples the input of the ADC to the reference ladder. The variation of the reference voltages caused by the feedthrough is modeled separately and added as extra with the DC voltages from the resistor ladder, see chapter 5.

The main effect of mismatches in a comparator is a constant offset voltage at the input. The first block in the comparator corrects the input voltage by adding the offset voltage to the input voltages of the comparator. The effects of mismatches are treated in chapter 8 .

The comparator is the most important part of the ADC, especially the transconductor and the regenerative comparator. Therefore it is studied extensively, which resulted in various models, that are described in chapter 6. The output of the comparator is digital, so the output code consists out of $clkperiods$ number of elements.

The total clock jitter is partially generated in the clock generator and partially in the comparator itself. Therefore the jitter is divided, one jitter model in the clock generator and a jitter model in every comparator. The clock jitter is discussed in chapter 7.

The decoder converts the thermometer code from the comparators to a normal binary output code ($Code\_out$). The modeling of the decoder of section 3.3 is very simple, because the decoder is completely digital. There is no further section dedicated to the decoder, because the implementation is straightforward.

The Signal to Noise Ratio (SNR) and the Integral and Differential NonLinearity (INL/DNL) are calculated from the output code, see chapter 9.

file:/imec/other/adsba/STUDENTS/dejong/c/totaladc.cpp

4.3 Filters in C++

Filters will appear to be very useful to model a number of behaviors of elements in the ADC. The modeling of the filters is described in this section.

The poles and zeros in the s-domain of the transfer functions are saved in the parameter list of the C++ model. The filter is transformed to the z-domain in C++ with a bilinear transformation. The bilinear transformation replaces every s according to equation (3).

$$s = \frac{2}{T_{sim}} \frac{z-1}{z+1} \tag{3}$$

A representation of a filter in the z-domain contains information about the simulation time (T_{sim}), the filter can only be used at that simulation time. Therefore the filter information is saved in the s-domain, so by doing the s to z transformation in C++ the filter can be used at any simulation frequency.

The first to fourth order filter are programmed in C++ as subroutines. An example is now given for the following filter:

$$H(s) = \frac{a_0 s + a_1}{b_0 s + b_1} \tag{4}$$

This filter must be transformed to a filter in the z-domain as shown by equation (5). Note that capitals are used for the coefficients in the z-domain.

$$H(z) = \frac{A_0 z + A_1}{B_0 z + B_1} \tag{5}$$

Equation (3) is substituted for both s in equation (4) to get a representation of the filter like (5). This results in the following calculation of the coefficients.

$$\begin{aligned} A_0 &= T_{sim} \cdot a_1 + 2 \cdot a_0; \quad A_1 = T_{sim} \cdot a_1 - 2 \cdot a_0 \\ B_0 &= 2 \cdot b_0 + b_1 \cdot T_{sim}; \quad B_1 = -2 \cdot b_0 + b_1 \cdot T_{sim} \end{aligned} \tag{6}$$

These equations give the s to z transformation of the coefficients for a first order filter using the bilinear transformation. These equations are programmed in C++, for a first to a fourth order filter. The next step is to normalize the coefficients with respect to B_0:

$$\hat{A}_0 = \frac{A_0}{B_0}; \quad \hat{A}_1 = \frac{A_1}{B_0}; \quad \hat{B}_1 = \frac{B_1}{B_0} \tag{7}$$

The input signal, $x[n]$, is filtered in the following way ($y[0] = 0$):

$$y[n] = \hat{A}_0 \cdot x[n] + \hat{A}_1 \cdot x[n-1] - \hat{B}_1 \cdot y[n-1] \tag{8}$$

file:/imec/other/adsba/STUDENTS/dejong/c/include/filter.h

4.4 Differential equations

A differential equation will be used in the sections 6.1 and 6.2. In this section is described how these equations are solved in C++. The differential equations are of the form:

$$\frac{dv}{dt} = f(t, v) \tag{9}$$

The Euler forward formula is used in order to solve this equation numerically:

$$v_{i+1} = v_i + T_{sim} \cdot f(t_i, v_i) \tag{10}$$

The variable v_i is the numerical expression for $v(i \cdot T_{sim})$. Equation (10) is solved at run-time in C++.

file:/imec/other/adsba/STUDENTS/dejong/c/include/comparator.h

5 Reference ladder with input feedthrough

Input feedthrough is the coupling of the input signal to the reference ladder. This input feedthrough causes errors in the output code, and is therefore modeled. This chapter is organized as follows. Section 5.1 describes a model that calculates the input feedthrough. Simulation results are given in section 5.2. In section 5.3 the maximum allowable input frequency in the reference ladder is calculated. The implementation in C++ is treated in section 5.4.

5.1 Input signal feedthrough

The input feedthrough is caused by the capacitive coupling of the two input nodes of the comparators, that couples the input of the ADC to the reference ladder, see Figure 1. Figure 10 shows the input gain stage of the comparator of Figure 4. The gate source capacitances of the transistors $M1a$ and $M1b$ cause the input feedthrough. For the calculation of the input feedthrough, a comparator is modeled by one capacitor (C). The value of C is the serial value of the two gate source capacitances ($C = C_{gs}/2$).

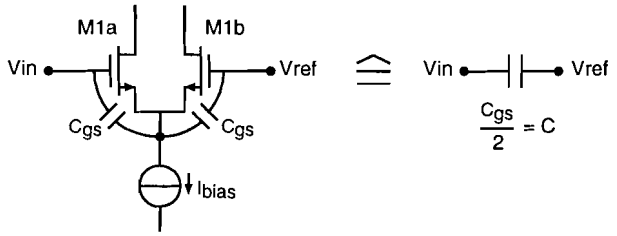


Figure 10: The input stage of the comparator is modeled by one capacitor in order to calculate the input feedthrough.

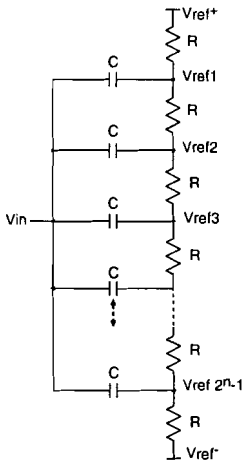


Figure 11: The input signal feedthrough model of the ADC. The comparators are replaced by a capacitor in order to calculate the feedthrough of the input on the reference ladder.

When the comparators are replaced by a capacitor, the input feedthrough calculation model looks like Figure 11. Figure 11 is redrawn in Figure 12a for further calculations. The input feedthrough is modeled by calculating the transfer function from the input signal to a node on the reference ladder. Afterwards the variation of the voltage in the reference ladder is subtracted from the input voltage

difference of the comparator.

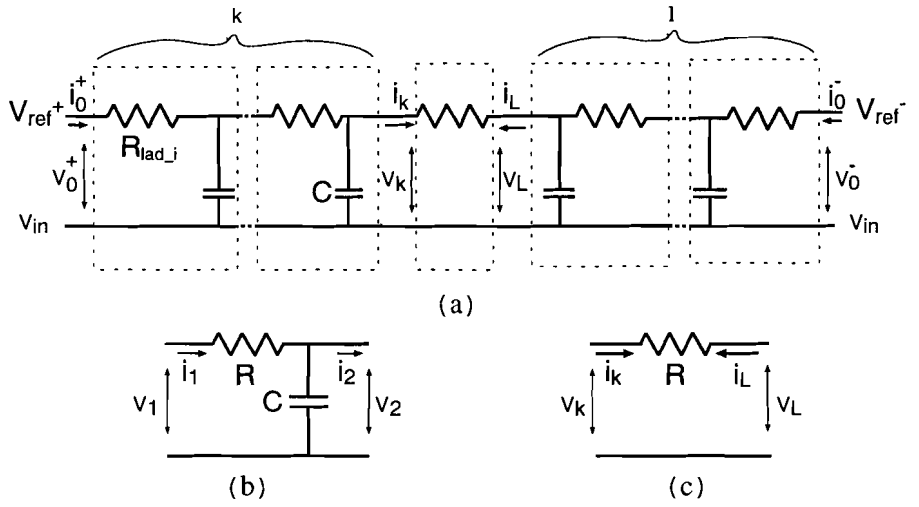


Figure 12: Input signal feedthrough calculation model.

When the reference ladder is connected between V_{ref}^{+} and V_{ref}^{-}, and the input voltage is v_{in}, then $v_o^+ = V_{ref}^+ - v_{in}$ and $v_o^- = V_{ref}^- - v_{in}$. The model is subdivided in two types of blocks, see Figure 12b and 12c. By placing k sections of Figure 12b to the left and l sections to the right of the middle section, Figure 12c, it is possible to calculate v_l for any tap and any length of the ladder, according to the formula $k = 2^n - 1 - l$.

$$\begin{bmatrix} v_2 \\ i_2 \end{bmatrix} = W \cdot \begin{bmatrix} v_1 \\ i_1 \end{bmatrix} \quad \text{with:} \quad W = \begin{bmatrix} 1 & -R \\ -j\omega C & 1 + j\omega RC \end{bmatrix} \tag{11}$$

$$\begin{bmatrix} v_k \\ i_k \end{bmatrix} = M \cdot \begin{bmatrix} v_l \\ i_l \end{bmatrix} \quad \text{with:} \quad M = \begin{bmatrix} 1 & -R \\ 0 & -1 \end{bmatrix} \tag{12}$$

Formulas (11) and (12) describe the transfer matrix of Figure 12b and 12c. The relation between v_l, i_l and v_o^-, i_o^- is given by formula (13):

$$\begin{bmatrix} v_l \\ i_l \end{bmatrix} = W^l \cdot \begin{bmatrix} v_0^- \\ i_0^- \end{bmatrix} \tag{13}$$

Combining (13) with (12) according to Figure 12a results in:

$$W^l \cdot \begin{bmatrix} v_0^- \\ i_0^- \end{bmatrix} = M \cdot W^k \cdot \begin{bmatrix} v_0^+ \\ i_0^+ \end{bmatrix} \tag{14}$$

The formulas (13) and (14) form a system with four equations and four unknowns (v_l, i_l, i_o^+, i_o^-). When v_l is solved from this system, the order of the transfer function is too high for efficient implementation in C++. Therefore an approximation is made:

$$W^k = [I + B]^k \approx I + k \cdot B + \frac{k \cdot (k-1) \cdot B \cdot B}{2} \tag{15}$$

with $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & -R \\ -j\omega C & j\omega RC \end{bmatrix}$

Formula (15) describes a second order approximation for W^k. This approximation is valid ([14]) if the absolute values of the eigenvalues are much smaller than 1, so at high frequencies the approximation is not valid any more. The approximation results in a fourth order transfer function for v_l/v_{in}, which is implemented in the model as an IIR filter.

5.2 Simulation results

The approximating model of section 5.1 will be compared with HSPICE in this section. The approximation in equation (15) must be checked, before comparing HSPICE with the model from section 5.1. When the approximation is approved, the comparison is made with HSPICE.

As a reference case a 4-bit 400 MHz ADC with $R_{ref} = 100\ \Omega$ is used. The symbolic simulator ISAAC ([15]) gives results for the model of section 5.1 without making approximations. The results consist of 15 transfer functions from the input to every node in the reference ladder ($v_{ref1} - v_{ref15}$). The model is symmetric around the middle node (v_{ref8}). Therefore a transfer function of a node above the middle is equal to his opposite node beneath the middle ($v_{ref1}(s)/v_{in}(s) = v_{ref15}(s)/v_{in}(s)$). So when all the transfer functions are plotted only 8 different lines will be perceived for the 4-bit case.

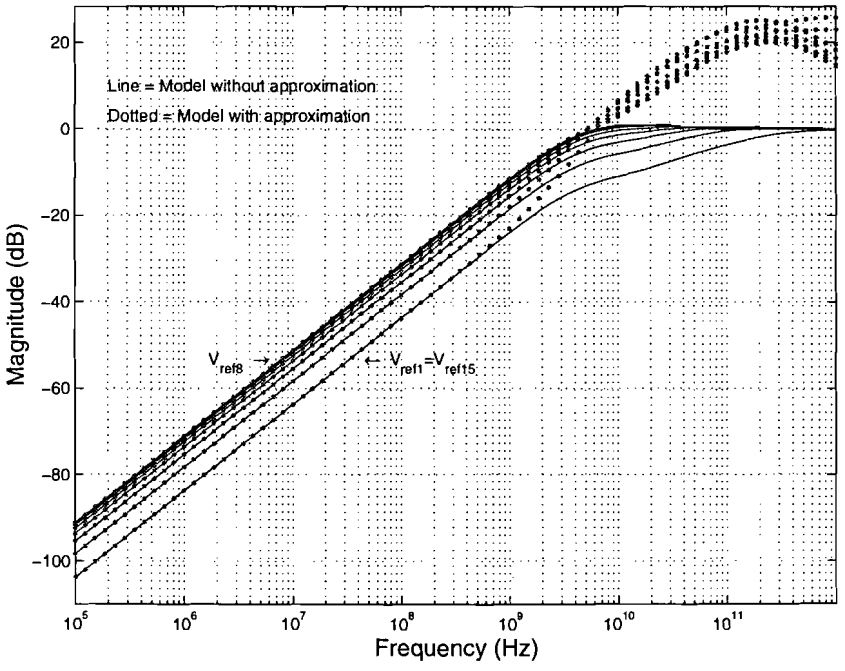


Figure 13: The 8 different curves of the input feedthrough calculated with the model of section. 5.1 with and without approximation

The exact ISAAC results are plotted together with the results of the model with approximation in Figure 13. The lowermost curve is the transfer function to the nodes v_{ref1} and v_{ref15}. The uppermost curve belongs to the middle node (v_{ref8}), in which is seen that the input feedthrough has the most influence on the middle node. The approximated model fits very well for frequencies beneath 1 GHz. This frequency is high enough, because the Nyquist bandwidth of the ADC is 200 MHz.

In Figure 14 the model of section 5.1 is compared with HSPICE. The value of C_{gs} is extracted from a HSPICE simulation and inserted into the model. The C_{gs} value of the middle comparator is chosen, because the input feedthrough has the most effect around the middle node. The HSPICE curves beneath

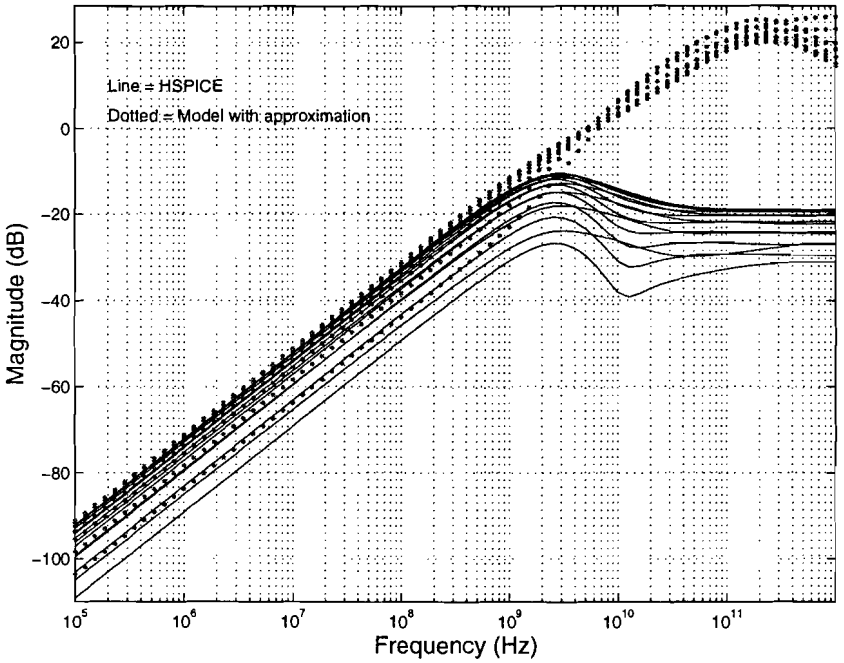


Figure 14: The input feedthrough calculated by the model of section 5.1 and HSPICE

1 GHz are parallel to the curves of the model and deviate in a range of 0.51 dB to 5.37 dB from each other. Note, that the HSPICE curves of symmetric nodes are not equal. This can be explained by the different operation points for the various transistors in the input stages of the comparators, because they are connected to different reference voltages ($V_{ref,i}$). The operation points of the transistors change also for different input voltages (v_{in}). The differences between HSPICE and the model for every node are listed in Table 1.

Although the model deviates almost 6 dB in some cases, no further research is done for a more accurate model in this report. It is perhaps interesting for later studies.

Table 1: The difference between HSPICE and the model in dB for $f_{input} < 1\ GHz$

v_{ref}	Difference	v_{ref}	Difference	v_{ref}	Difference
1	1.21 dB	6	0.61 dB	11	2.79 dB
2	0.96 dB	7	0.85 dB	12	3.47 dB
3	0.72 dB	8	1.19 dB	13	4.14 dB
4	0.56 dB	9	1.63 dB	14	4.77 dB
5	0.51 dB	10	2.17 dB	15	5.37 dB

file: /imec/other/adsba/STUDENTS/dejong/input_feedthrough/inp_ft15_sym.m
file: /imec/other/adsba/STUDENTS/dejong/c/include/filter.h

5.3 Maximum input feedthrough

An important design problem is the value of the resistors in the reference ladder. The resistors must be as high as possible, because this results in a low power dissipation. However, when the resistor values

increase the input feedthrough becomes a problem. So there is a trade-off between power and input feedthrough. An equation is derived in this section, that determines an optimum value for the resistors.

Only the voltage on the middle node of the reference ladder is studied, because the maximum input feedthrough occurs at that node. A general expression for the transfer function of the input to the middle node (v_{mid}/v_{in}) is taken from section 5.1. This formula is simplified by truncating small terms and results in:

$$\frac{v_{mid}}{v_{in}} = \frac{\pi}{4} f_{in} RC 2^{2n} \tag{16}$$

n is the number of bits of the ADC. When f_c is defined as $f_c = \frac{1}{2\pi RC}$, formula 16 can be rewritten as:

$$\frac{v_{mid}}{v_{in}} = \frac{f_{in}}{f_c} 2^{2n-3} \tag{17}$$

When a maximum allowable input feedthrough ($v_{mid,max}$) is chosen, an optimum value for R is calculated by using formula 16. The optimum value for R is named R_{opt} and is given by:

$$R_{opt} = \frac{4 \frac{v_{mid,max}}{v_{in}}}{\pi 2^{2n} f_{in} C} \tag{18}$$

ϕ is defined as the maximum feedthrough of the input signal on the middle node in the reference ladder in LSB, see formula (19).

$$\phi = \frac{v_{mid,max}}{LSB} = \frac{v_{mid,max}}{v_{in}/2^n} \tag{19}$$

When formula (19) is inserted in formula (18), the optimum value for R is then given by:

$$R_{opt} = \frac{4\phi}{\pi 2^{3n} f_{in} C} \tag{20}$$

For example, a 4-bit ADC is considered with a maximum input frequency of 200 MHz and a maximum of 1 LSB input feedthrough is allowed, and the comparator of Figure 4 is used ($C = 14\ fF$). A reference ladder resistor of 110 Ω has to be chosen.

In [16] an other way to calculate the maximum input feedthrough is described, the results are equal.

5.4 Input feedthrough in C++

The input feedthrough routine in C++ calculates the voltage on a node of the reference ladder, by filtering the input signal. The fourth order filter of section 5.1 is used. The input feedthrough routine performs $2^n - 1$ filtering functions. Therefore the following general formula is defined for the input feedthrough:

$$v_{refx}[clklength \cdot clkperiods] = INP\_FT(v_{in}[clklength \cdot clkperiods], R, C, x) \tag{21}$$

The input variables of the input feedthrough routine (INP_FT) are the array with the input signal with $clklength \cdot clkperiods$ elements (v_{in}), the value for R, C and the node in the reference ladder (x). This function is used $2^n - 1$ times every simulation with variating values for x from 1 to $2^n - 1$.

The curves of Figure 14 have an increase of 20 dB per decade within the Nyquist bandwidth of the ADC. A first order high pass filter has also an increase of 20 dB per decade. However it is not possible to model the fourth order transfer function of the model with a first order transfer function, because the phase shift of the fourth order transfer function is larger than 90 degrees.

5.5 The total reference ladder

The reference ladder network in the ADC model, see Figure 9, consists out of the input feedthrough and the reference ladder with mismatches. The reference ladder with mismatches calculates the DC voltages on the nodes in the reference ladder. In the input parameter list is saved the nominal value of a resistor and the standard deviation of the resistors. At the beginning of a simulation 2^n resistor values are randomly generated on a Gaussian way with a mean of the nominal value and the adjusted standard deviation. Then the DC reference voltages are calculated with these resistor values.

The total reference voltages are these DC voltages in addition with the input feedthrough.

file:/imec/other/adsba/STUDENTS/dejong/c/include/ref_ladder.h

6 The comparator

The comparator is the most important and difficult part of an ADC. This chapter describes various new methods to model the behavior of a comparator.

6.1 Differential equation model with a sine function

In this section the comparator behavior is modeled by studying the relation between the Output voltage (v_a) of the comparator and its derivative to time (dv_a/dt). First the model is described and afterwards simulation results are given.

6.1.1 The model

The basic comparator operation is modeled with a nonlinear differential equation:

$$\frac{\delta v_a}{\delta t} = f(v_a(t), t) \tag{22}$$

with the comparator output $v_a(t)$ and the differential input current $gm_{in} \cdot (v_{in}(t) - v_{ref}(t))$ as $i_a(t) - i_b(t)$, see Figure 4. The regenerative comparator has two stable positions (VSS and VDD) and one meta-stable position (VDD/2) to which the comparator is reset every clock period. In (22) $\dot{f} > 0$ in the meta-stable point and this first derivative describes the exponential growth in the beginning of the comparison phase. The function $f = 0$ and the derivate $\dot{f} < 0$ in the stable points.

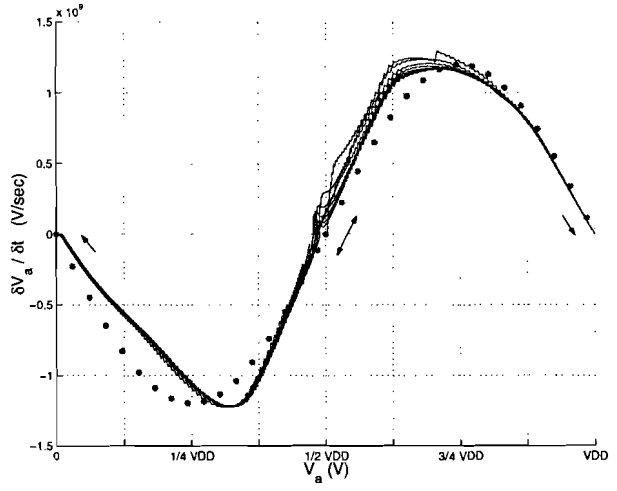


Figure 15: Sine wave dependency of $\delta v_a(t)/\delta t$ as function of $v_a(t)$ is feasible as modeling function.

In HSPICE the transients are studied for different input voltages at the comparator outputs. These functions are differentiated with respect to time and plotted in Figure 15 as function of the output voltage $v_a(t)$. The sine wave like behavior as depicted in Figure 15 is modeled with physical input parameters which define the speed of the comparator:

$$\frac{\delta v_a(t)}{\delta t} = -A \cdot \sin\left(\frac{2\pi \cdot v_a(t)}{VDD}\right) \tag{23}$$

$$\frac{\delta\, v_a(t)}{\delta\, t}|_{v_a(t)\to 0} \approx -A \cdot \frac{2\pi \cdot v_a(t)}{VDD} \tag{24}$$

In [8] it is proven that:

$$\frac{2\pi A \cdot v_a(t)}{VDD} = \frac{gm_r}{C} \cdot v_a(t) \Rightarrow A = \frac{gm_r}{C} . \frac{VDD}{2\pi} \tag{25}$$

If (25) is filled out in (23) and corrected for the difference in value of the input currents, the suggested differential equation for modeling the behavior of the comparator during the compare phase is:

$$\frac{\delta\, v_a(t)}{\delta\, t} = \frac{i_a(t) - i_b(t)}{C} - \frac{gm_r}{C} . \frac{VDD}{2\pi} . \sin\left(\frac{2\pi . v_a(t)}{VDD}\right) \tag{26}$$

The start value of v_a is determined by the difference $i_a(t) - i_b(t)$, this is the differential current from the input stage to the regenerative comparator, $i_a(t) - i_b(t) = gm_{in} \cdot (v_{in}(t) - v_{ref}(t))$. So different input values result in different time delays at the output of the comparator. It is also seen that the input difference $i_a(t) - i_b(t)$ has still influence during the comparing. The parameter gm_{in} is the transconductance of the input stage, which is equal to the gm of transistor $M1$ in the comparator circuit, so gm is a constant and independent of the input frequency. gm_r is the transconductance of the regenerative comparator. In the circuit of Figure 4 is gm_r given by $gm_r = gm_7 + gm_8 + gm_6$, according to [17]. C is the total capacity on a regenerative node including the capacitive load of the output latch.

6.1.2 Simulation results

The parameters gm_{in}, gm_r and C are extracted from HSPICE operating point simulations only once and inserted into the model. The differential equation (26) is efficiently solved at runtime in the C++ program. The time domain results are compared with HSPICE simulations in Figure 16 and show a good agreement.

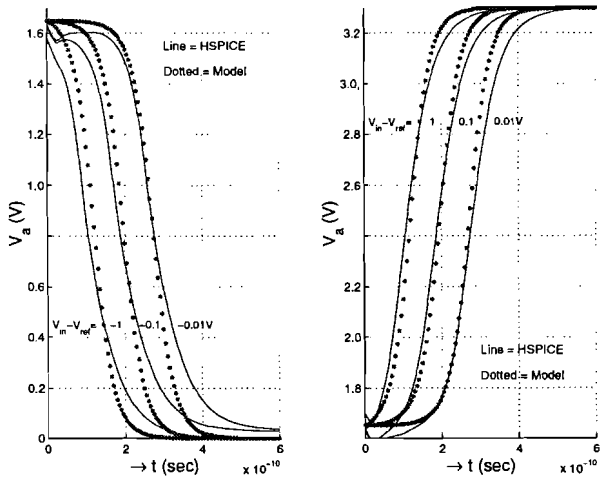


Figure 16: Transients of C++ model and HSPICE simulations agree well for the positive and negative edges.

Both graphs start at $VDD/2$ ($= 1.65\ V$) and end at VSS or VDD ($= 3.3\ V$). The maximum differential input signal is set to 1 V_{pp} (input range flash ADC). With this input voltage the comparator reacts fast, as shown in Figure 16. Further, it is seen in this figure that if the input signal is almost equal to the reference voltage, then the comparator reacts much slower. During this small value for v_a the current difference $i_a(t) - i_b(t)$ can still influence v_a and can cause an error in the output code of the ADC. Note, the logarithmic dependancy of the delay on the input voltage difference.

file: /imec/other/adsba/STUDENTS/dejong/c/include/comparator.h

6.2 Comparator with 4-MOS model

In this section is described another way to model the output transients of the regenerative comparator. First the model is described in section 6.2.1 and next simulation results are given in section 6.2.2.

6.2.1 The model

The basic circuit of a regenerative comparator consists of two inverters. A NMOS and a PMOS form an inverter. A new model is presented in Figure 17, that is named 4-MOS model. The currents from the transconductance are modeled with the current sources i_a and i_b. The speed of the regenerative comparator is determined by the total capacitances (C_a and C_b) on the output nodes A and B. This model calculates the current that loads the capacitances, in order to calculate the output voltage (v_A and v_B).

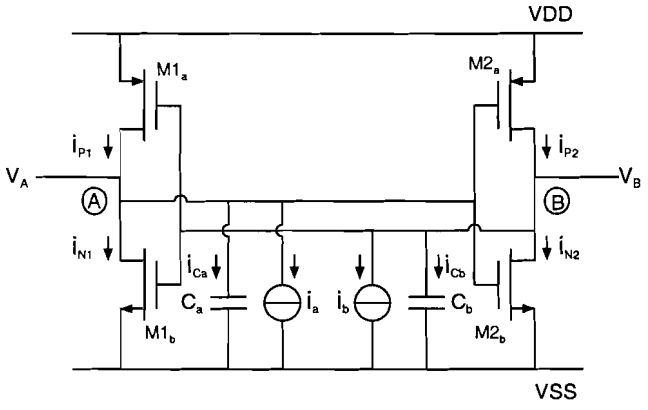


Figure 17: The macromodel used for the 4-MOS model.

This model is used to describe the behavior of the circuit of Figure 4. As is seen the transistors $M8$ and $M9$ of this circuit are not modeled, which can cause deviations in the results.

The model starts in the meta-stable situation, $v_a(0) = v_b(0) = V_{DD}/2$. First it calculates the current through the capacitances, i_{Ca} and i_{Cb}. This is the current from the PMOS minus the current trough the NMOS and the transconductance, see the following equation:

$$i_{Ca} = i_{P1} - i_{N1} - i_a \quad \text{and} \quad i_{Cb} = i_{P2} - i_{N2} - i_b \tag{27}$$

Secondly a value for dv/dt is calculated with the formula: $i_C = C \cdot dv/dt$. Finally, the next value for $v_{a,k+1}$ and $v_{b,k+1}$ is calculated, with the following formula:

$$v_{a,k+1} = v_{a,k} + \frac{dv_a}{dt} \cdot T_{sim} \quad \text{and} \quad v_{b,k+1} = v_{b,k} + \frac{dv_b}{dt} \cdot T_{sim} \tag{28}$$

T_{sim} is the simulation time step. This process is repeated, until v_a and v_b have reached V_{SS} or V_{DD}. Two transistor functions, i_N and i_P, are defined, that calculate the drain current as a function of v_a and v_b. These functions use the familiar MOS model with three working regions, subthreshold, linear and a saturation region. The transistor function, i_N, for a NMOS is given by:

$$i_N(v_{gs}, v_{ds}) = \begin{cases} 0 & v_{gs} < V_T \\ \beta \cdot (v_{gs} - V_T - v_{ds}/2) \cdot v_{ds} & v_{ds} < v_{gs} - V_T \wedge v_{gs} > V_T \\ \beta/2 \cdot (v_{gs} - V_T)^2 & v_{ds} > v_{gs} - V_T \end{cases} \tag{29}$$

The values for β and V_T are extracted from HSPICE. The transistor function, i_P is similar to the i_N, only v_{gs} and v_{ds} are replaced with $V_{DD} - v_{gs}$ and $V_{DD} - v_{gs}$, in such a way that i_P becomes a function of v_a and v_b. The dv/dT is given by:

$$\frac{dv_{a,k}}{dt} = \frac{-i_N(v_{a,k}, v_{b,k}) + i_P(v_{b,k}, v_{a,k}) - i_a}{C_1} \tag{30}$$

$$\frac{dv_{b,k}}{dt} = \frac{-i_N(v_{b,k}, v_{a,k}) + i_P(v_{a,k}, v_{b,k}) - i_b}{C_2} \tag{31}$$

Thus, by starting with $v_a(0) = v_b(0) = V_{DD}/2$ and repeating formula (30), (31) and (28) the subsequent transient voltages for v_a and v_b are calculated.

6.2.2 Simulation results

The values for C, β and V_T are extracted from HSPICE and inserted in the model. This model is programmed in C++. The first comparison of the HSPICE and C++ was not satisfying. A better matching was found with an other value for β. The transients with an adapted β are shown in Figure 18.

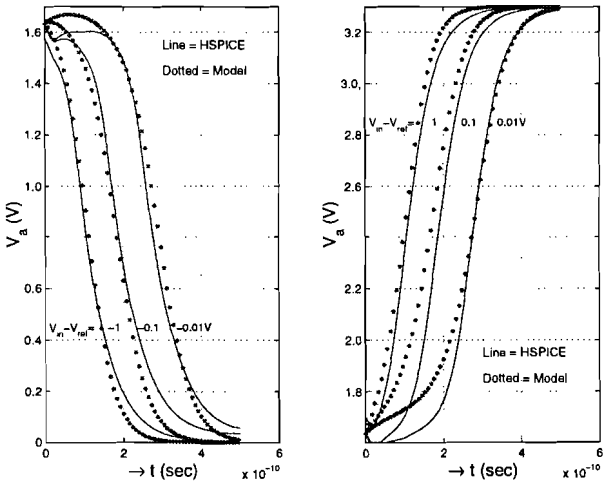


Figure 18: Transients of the 4-MOS model and HSPICE (a) Negative slope (b) positive slope.

This figure shows that the 4-MOS model can simulate the transients of HSPICE very well. The deviations in the β from HSPICE and the adapted value in the C++ model can be explained by the following reasons. The transistors M8 and M9 of Figure 4 are not taken into account. The capacitance of C_a and C_b are dependent on the voltage across the capacitors.

The disadvantage of this model is that the parameters in this model can not directly be calculated or extracted from HSPICE. The calculation of the parameters is done by trial and error. Therefore this model is no subject for further study.

file: /imec/other/adsba/STUDENTS/dejong/c/include/comparator.h
file: /imec/other/adsba/STUDENTS/dejong/matlab/gerd/mosx4.m

6.3 Comparator model with filter

The study of the models from section 6.1 and 6.2 resulted in some recommendations for a new model, which is presented in this section.

6.3.1 The model

The switch of the comparator does not open in an infinite small time. During the opening of the switch and while the regeneration starts, the input signal still has some influence on the regenerative nodes. Especially when the input is changing fast. For example: The input voltage is very small at the start of the compare phase, and then changes rapidly from sign, the comparator can give the wrong output value. No clear simulation results show this effect.

The influence of the input signal after the clock instant is taken into account in the models from section 6.1 and 6.2. These models also simulate the complete output transients very good. However the shape of the output transient is not interesting, because only the final digital output (high/low) is of interest.

However the delay from the input nodes to the regenerative nodes is not considered. The input stage is modeled by the constant value of the transconductance, gm_{in}. The model in this section only takes into account the filtering function of the input stage and not the shape of the transient or the influence of the input during the beginning of the compare phase.

The regenerative comparator amplifies the voltage difference between the regenerative nodes (nodes A and B in Figure 4), that is present at the beginning of the compare phase. The comparator model presented in this section uses the transfer function from the input voltage to the voltage difference on the regenerative nodes. Figure 19 shows the complete model.

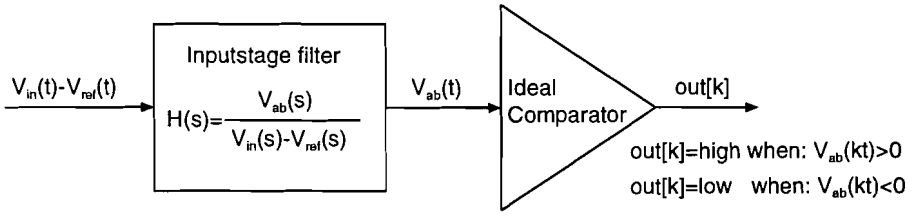


Figure 19: The comparator model with the input stage filter and an ideal comparator.

The voltage difference between the regenerative nodes is named $v_{ab}(t)$ $(= v_a(t) - v_b(t))$. An ideal clocked comparator is connected to the output of the filter. HSPICE calculated the transfer function from the input voltage difference to the voltage difference between the regenerative nodes ($H(s)$), see Figure 20.

A third order IIR filter is fitted on the HSPICE curve. This third order filter is implemented in C++.

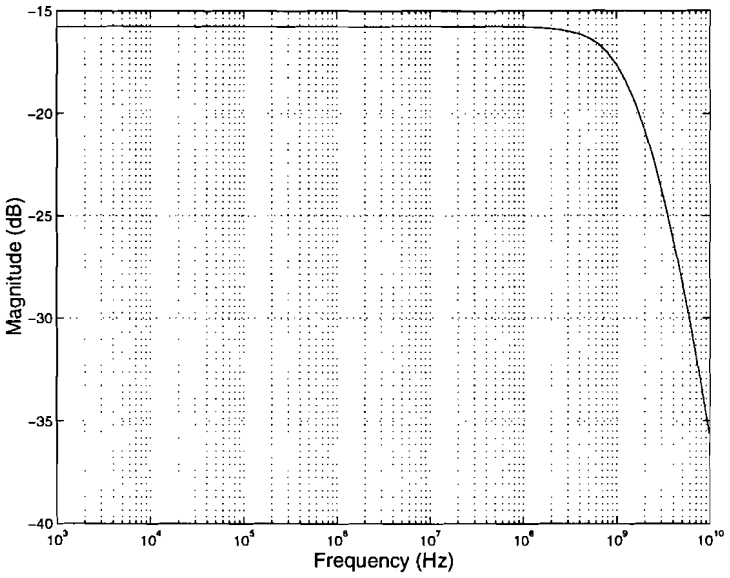


Figure 20: The transfer function from the input voltage difference to regenerative nodes voltage difference calculated by HSPICE, $H(s) = \frac{v_{ab}(s)}{v_{in}(s) - v_{ref}(s)}$.

The curve is almost flat for frequencies beneath 200 MHz. The phase shift at 200 MHz is only 0.35 degrees. However simulation showed, that this small phase shift is very important.

6.4 Conclusions

The models from section 6.1 and 6.2 describe the transient of the output of the comparator very good, however it has no importance in the ADC model. Only the final output codes of the comparator are interesting. These models have a constant value gm_{in} for the input stage, and take into account the effect of the input signal at the start of the compare phase. Simulations showed that this effect is marginal. The filtering of the input stage before the actual comparison is more important and has more influence on the output code.

This conclusion is only valid for this comparator circuit in HSPICE. In other situations it may be possible that an ideal comparator is not sufficient and an entirely different model is needed.

The model from section 6.3 models this filter function. The regenerative comparator is modeled by a simple ideal comparator, in contrast with the models of sections 6.1 and 6.2, that unnecessarily model the output transient of the regenerative comparator. Simulations showed, that the model from section 6.3 is much better. Therefore only this model is used from now on.

The resistance of the clock transistor is small during the reset phase and large during the compare phase. When the compare phase starts, the resistance changes gradually. So the regeneration starts also gradually. This effect is not modeled in any of the models in this chapter.

7 Clock jitter

The modeling of clock jitter in an ADC is treated in this chapter. Section 7.1 describes the theory of clock jitter. This theory is only a glance at the complicated origin of jitter. In section 7.2 the actual modeling of the clock jitter is described.

7.1 Jitter in theory

This section is only an introduction to clock jitter theory, because it is very difficult and is beyond the scope of this report. The theory and the references can be used as a beginning for further study.

There are two types of jitter in an ADC. One is the instability in the phase (phase noise) of the clock signal, the other is the internal jitter of the comparator and the S/H. The phase noise of the clock signal is caused by the oscillators and frequency dividers. First, in section 7.1.1 the phase noise from the oscillator is described. The influence on phase noise of a special type of frequency divider, a PLL, is described in section 7.1.2. In section 7.1.3 the internal jitter is explained. A maximum allowable value for the clock jitter is defined in section 7.1.4. Formulas to calculate the phase noise power at the output are given in section 7.1.5.

7.1.1 Phase noise in the oscillator

The output of an ideal sinusoidal oscillator can be characterized with $v_{out} = A\cos(\omega_0 t + \phi)$. However in a practical oscillator the output is given in a more general way by:

$$v_{out}(t) = A(t) \cdot f[\omega_0 t + \phi(t)] \tag{32}$$

The fluctuations of the amplitude ($A(t)$) and the phase ($\phi(t)$) are caused by noise in the oscillator. The fluctuation of the phase is called "phase noise". The phase noise is the interesting part, because the influence of the amplitude fluctuations is reduced by amplitude limiting mechanism. The most used way to characterize the phase noise is by looking at the frequency spectrum. The spectrum of an ideal oscillator should only contain a pair of impulses at $\pm\omega_0$. However, by the fluctuations in the phase and amplitude, the spectrum contains sidebands close to the oscillator frequency, see Figure 21a. The phase noise is quantified in decibels below the carrier per Hertz (dBc), see equation (33).

$$L(\Delta\omega) = 10 \cdot \log\left[\frac{P_{sideband}(\omega_0 + \Delta\omega, 1\ Hz)}{P_{carrier}}\right], \quad [dBc] \tag{33}$$

This is the ratio of the power, in a unit bandwidth ($1\ Hz$) at an offset frequency $\Delta\omega$, to the power of the carrier ω_0, see Figure 21a.

A measured spectrum of an output of an oscillator can be divided in three parts. In a part with a decline of $1/f^3$, $1/f^2$ and a constant part, see Figure 21b. Note that the frequency axis is logarithmic. The model described in [18] and [19] calculates these curves.

7.1.2 Phase noise in the PLL

A PLL can be used to scale down the clock frequency for an ADC. It has his own influence on the phase noise, therefore it is treated here. The phase noise at the output of a PLL, is caused in two ways.

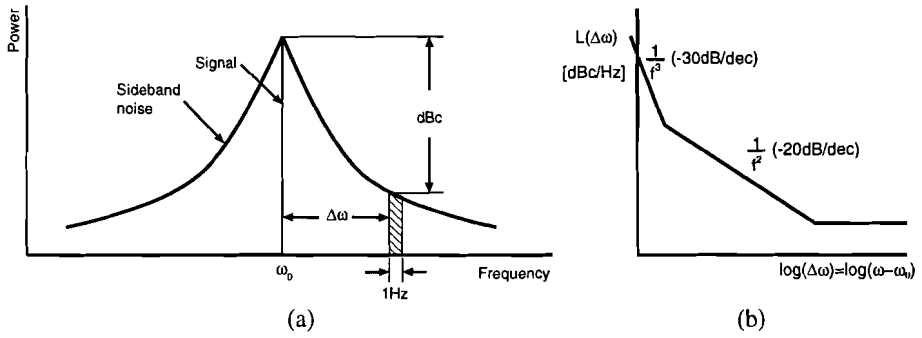


Figure 21: Phase noise (a) definition of phase noise (b) typical phase noise spectrum of an oscillator.

The PLL introduces phase noise itself and it shapes the input phase noise. The introduced phase noise is generated in all the building blocks of a PLL. However the contribution of the VCO is dominant. The phase noise of the VCO experiences a high-pass transfer function, when it propagates to the output. The input phase noise from the oscillator is shaped by the characteristic low-pass transfer function of the PLL. For more about noise in a PLL see [20].

7.1.3 Internal jitter

The internal jitter is caused by various effects, such as slewing dependency and finite aperture time. The internal jitter is signal slew dependent, this is the sensitivity for variations in the rise and fall time of the clock signal. It is not possible to take a sample in an infinite small time, there is always some finite aperture time necessary. During this aperture time the output can still change. When the regenerative comparator starts it's comparing, the transconductance can still inject current in the regenerative comparator, which influences the output of the comparator. The study of a S/H is not in the scope of this report. The S/H is implemented as an ideal S/H.

7.1.4 Maximum jitter

The maximum (Δt_{max}) is defined as the deviation in sampling time for which the sampled value deviates one LSB from the value at the ideal sampling instant. The maximum jitter (Δt_{max}) is easily calculated, with equation (34) from [21]. Δt_{max} is the number of bits of the ADC is n and the input frequency is f_{in}.

$$\Delta t_{max} = \frac{2^{-n}}{\pi \cdot f_{in}} \tag{34}$$

Note that Δt_{max} is only dependent on the input frequency and the number of bits, which has a direct relation with the value of LSB. For example, Δt_{max} is calculated for a 4-bit ADC with a clock frequency of 400 MHz. Jitter has more influence at higher frequencies, so the Nyquist frequency ($f_{in} = 200MHz$) is inserted in equation (34). Then a value of $100ps$ is found for Δt_{max}.

7.1.5 Noise power

Formulas are enumerated in this section, that calculate the Noise power in the output signal, caused by jitter (P_j). Noise power is caused by not taking a sample at $k \cdot T$ however at $k \cdot T + \Delta t_k$. When the input signal is $v_{in}(t) = A\cos(2\pi f_{in} t)$ and N samples are considered, then [22] declares the following equation:

$$P_j = 1 - \frac{Q(1-Q^N)}{N(1-Q)} \quad with: \; Q = E(e^{j2\pi f_{in}\Delta t_k}) \tag{35}$$

$E(\cdot)$ is the expected value operator. Note that the distribution of Δt_k does not have to be Gaussian.

In a situation, when the distribution (σ) is Gaussian and the input is not necessarily one sine wave, the noise can be calculated according to [23]. The Fourier transform ($F(j\omega)$) of v_{in} must be known.

$$P_j = 2\int_{-\infty}^{\infty} |F(j\omega)|^2 [1 - exp(-\frac{\omega^2\sigma^2}{2})] d\omega \tag{36}$$

In an other situation, when the input signal is of the form $v_{in}(t) = A\cos(2\pi f_{in} t)$ and the distribution of Δt_k is Gaussian, the noise power is given by (see [22]):

$$P_j = A^2(1 - exp(-2\pi^2 f_{in}^2 \sigma^2)) \tag{37}$$

Since the signal power is $A^2/2$, the SNR is given by:

$$SNR = -10\, log\, 2\, (1 - exp(-2\pi^2 f_{in}^2 \sigma^2)) \quad [dB] \tag{38}$$

When for f_{in} the Nyquist frequency is chosen and the Gaussian distribution is known, then the maximum achievable SNR can be calculated. Also the other way around, when a certain SNR is wanted, equation (38) gives a maximum value for the variance (σ) of the jitter. For example, a situation with a 4-bit ADC with a sample frequency of 400 MHz. For the input frequency the Nyquist frequency ($F_{in} = 200MHz$) is chosen. When a minimum SNR of 24 dB is desired, the maximum standard deviation of the jitter is then, $\sigma = 60ps$.

This model can be extended with noise caused by the finite aperture time effect (P_a). This effect is caused by the impossibility of sampling a signal in an infinite small time, a signal (v_{in}) is always sampled during a certain aperture time (δ_t). Then the output (v_{out}) can be approximated with:

$$v_{out}(t) = \frac{1}{\delta t}\int_{t-\delta t/2}^{t+\delta t/2} v_{in}(\tau) d\tau \tag{39}$$

A rectangular aperture window is assumed. When the input signal is given by $v_{in} = A\sin(2\pi f_{in} t)$, then the noise power ($P_a$) according to [23] is given by:

$$P_a = \frac{A^2}{2}\left(1 - sinc(2\pi f_{in}\frac{\delta t}{2})\right)^2 \tag{40}$$

Finally, when the jitter Δt_k has a Gaussian distribution with a variance σ^2, and the aperture time is δt, then the total noise power (P_{j+a}) caused by aperture jitter and the finite aperture time effects is [23]:

$$P_{j+a} = \frac{A^2}{2}[1 - sinc(2\pi f_{in}\frac{\delta t}{2})]^2 + sinc(2\pi f_{in}\frac{\delta t}{2}) \cdot (A^2[1 - exp(-2\pi^2 f_{in}^2 \sigma^2)]) \quad (41)$$

7.2 Modeling clock jitter

In this chapter the actual modeling of the clock jitter is described. The place of the clock jitter model in the ADC is first treated in section 7.2.1. Section 7.2.2 describes the implementation of the clock jitter model.

7.2.1 The place of clock jitter in the ADC

When a S/H is connected to the ADC it is not necessary to add a jitter model to the comparators, because the S/H keeps the input voltage of the comparator constant during the sampling of the comparator. It does not matter if the comparator takes a sample somewhat earlier or later. So, there are two situations to be distinguished, a situation with and without a S/H. First the situation is described without a S/H.

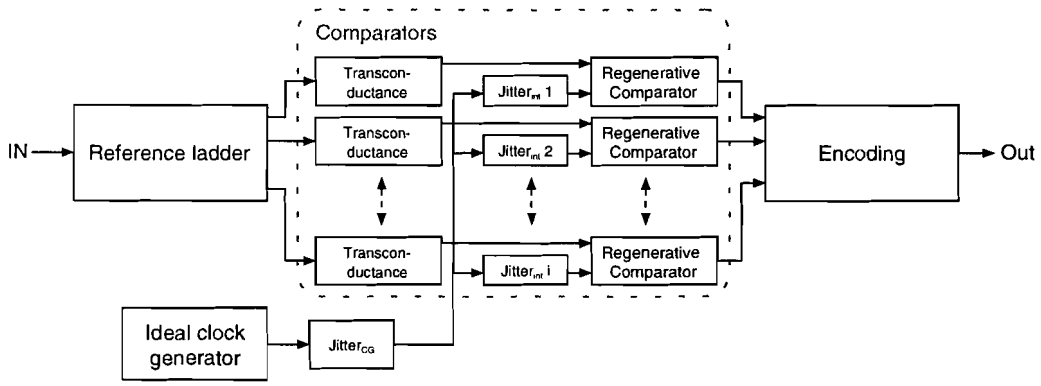


Figure 22: The clock jitter in case no S/H is used.

The jitter is split in two parts, a jitter caused by the clock generator ($Jitter_{CG}$) and a jitter caused by the comparators itself ($Jitter_{int}$). The contribution of the clock generator to the total jitter is the same for all comparators. However the internal jitter of the comparators is different in every comparator, see Figure 22. So, there is correlation of the total jitter in the comparators. This correlation is caused by the clock generator part.

When a S/H is placed before the ADC, the situation becomes more simple. Because the S/H keeps the input voltage of the comparator constant, which makes an accurate modeling of the jitter in the comparators unnecessary. So there is only one jitter model needed in the ADC, see Figure 23. Furthermore, it is not necessary to split the jitter in a comparator and clock generator part.

7.2.2 Clock jitter implementation

There is not enough information about clock jitter at IMEC. The theory of section 7.1 does not provide enough information to estimate the phase noise in a realistic way. However to get an impression of the effect of clock jitter in an ADC a simple model is used, which is described in this section. The clock jitter model is integrated in a regenerative comparator or in a S/H. The clock jitter model calculates a

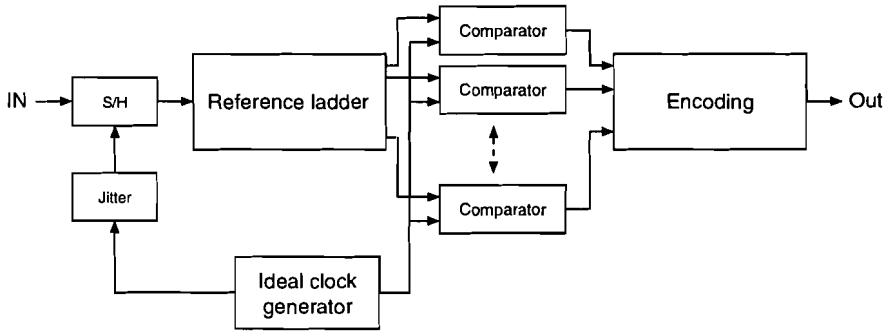


Figure 23: The clock jitter when S/H is used.

value for Δt_k. The regenerative comparator or S/H uses this value to take a sample at an ideal sampling instant plus Δt_k. The calculation of the various Δt_k values is performed by a random generator with a Gaussian distribution. The choice for a Gaussian distribution is supported by [24]. However to estimate a realistic value for the variance is not possible. First the clock jitter model is described in combination with a regenerative comparator, then in combination with a S/H.

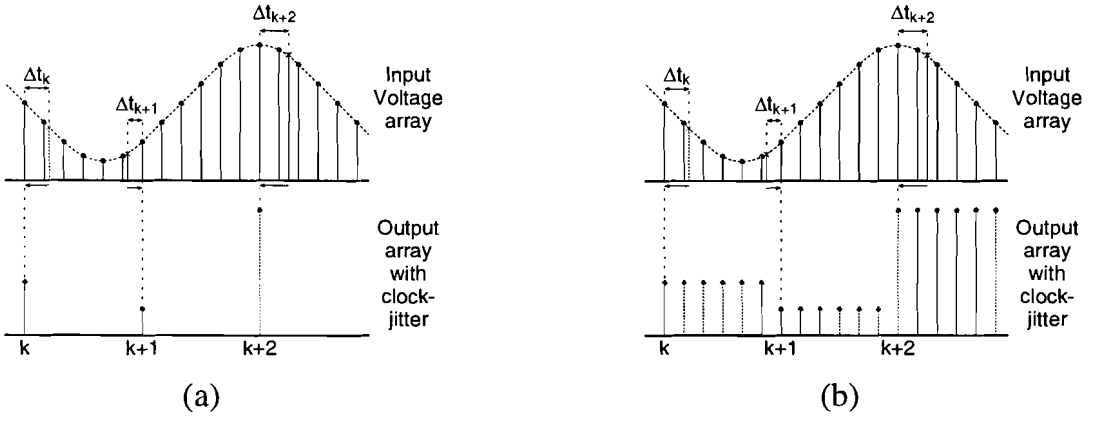


Figure 24: Clock jitter model (a) in a regenerative comparator (b) in a S/H.

The clock jitter model in a comparator constructs from the input array with $clklength \cdot clkperiods$ elements an output array with $clkperiods$ elements. This output array contains the samples at the sampling time with jitter $(k \cdot clklength + \Delta t_k)$. The jitter (Δt_k) is a summation of the clock generation jitter plus the internal jitter $(\Delta t_{int,i,k} + \Delta t_{CG,k} = \Delta t_k)$. The values for $\Delta t_{int,i,k}$ and $\Delta t_{CG,k}$ are calculated separately by a random generator. The clock jitter model is located between the input stage filter and the ideal comparator of Figure 19. The final implementation in C++ is showed in Figure 9.

The ideal comparator uses the new sample array with $clkperiods$ elements to calculate an output array with $clkperiods$ output codes of zeros and ones. Mostly, Δt is not a multiple of the simulation time step, so the input array is interpolated to get a value for the output array.

The clock jitter model in a S/H passes the value of the input voltage at sampling time with jitter $(k \cdot clklength + \Delta t_k)$ to the output. The S/H keeps this value constant during one clock period, see Figure 24b. The number of elements in the input array is equal to that of the output array. Δt_k is the total of the various jitter contributions.

file:/imec/other/adsba/STUDENTS/dejong/c/sample_hold.h

7.2.3 Conclusions

There are two situations. One with a clock jitter model for the S/H. And there is an other situation with a clock generator with jitter and an internal clock jitter model in every comparator in case no S/H is used.

It is difficult to give quantitative values to the jitter. A Gaussian distribution with an arbitrary variance is chosen to model the jitter. The variances of Δt_{int} and Δt_{CG} ares saved in the input parameter list of the C++ model.

8 Mismatch in the comparator

The unwanted effects of mismatches in a comparator are studied in this chapter. First the theory of mismatch is treated briefly. The main effect of mismatch in a comparator is an offset voltage at the input. This is explained in section 8.2. Two methods to calculate the offset voltage are described in section 8.3 and 8.4, the first method is taken from [25] and the second method is a new and faster one. Finally some simulation results are presented in section 8.5.

8.1 Theory

Mismatch is the variation in physical quantities of identically designed components. This is caused by random variations occurring during processing. There are several models that try to describe mismatch in MOS transistors. The model described by Pelgrom ([26]) is used; this model uses less parameters and is the most known. This section explains the model.

Variations in the width (W), the length (L) and the distance between two transistors (D) contribute to the mismatch. These variations have a Gaussian distribution, with a mean of zero and a standard deviation σ. Three MOS model parameters are identified that are influenced by the variations in W, L and D. These parameters are the threshold voltage (V_T), the current factor (β) and the substrate factor (K). The following formulas describe a relation between W, L and D and the variance (σ^2) of the transistor parameters (V_T, β, K) by defining technology constants (A_x, S_x).

$$\sigma^2(V_T) = \frac{A_{V_T}^2}{WL} + S_{V_T}^2 D^2 \; ; \quad \sigma^2(K) = \frac{A_K^2}{WL} + S_K^2 D^2 \; ; \quad \frac{\sigma^2(\beta)}{\beta^2} = \frac{A_\beta^2}{WL} + S_\beta^2 D^2 \tag{42}$$

These variances describe the statistical behavior of the differences between two identically designed transistors. The terms in the formulas with S_x are omitted because their impact is negligible, when the distance between two transistors is kept below 100 μm. The variance of K is very small and is therefore also left out. So the next formulas will be used in the mismatch calculation:

$$\sigma^2(V_T) = \frac{A_{V_T}^2}{WL} \; ; \quad \frac{\sigma^2(\beta)}{\beta^2} = \frac{A_\beta^2}{WL} \tag{43}$$

8.2 Mismatch in the comparator

Because of the differential structure of the regenerative comparator, the offset voltage of the comparator should be zero. However by mismatches in the transistors, the offset voltage differs from zero. The method described in this section ([25]) calculates the input referred offset voltage (V_{off}) of the comparator caused by the mismatches of all transistors.

First the offset voltage with no mismatches is calculated ($V_{off,nominal}$), this should be zero. Second the effect of every statistical variable (σ_{V_T},σ_β) in one transistor on the offset is calculated separately. One statistical variable, V_T or β, in one transistor is changed with the half of the mismatch, while all other statistical variables are left at their mean value (0). Equations (43) gives a relation for the difference between two transistors. Thus, by giving every transistor half of the mismatch, it is not necessary to investigate which transistors form a pair. Then the offset voltage is calculated again. The following formula defines the sensitivity ($\delta V_{off}/\delta i$). The sensitivity is the relation between the input referred offset voltage and one changed statistical variable.

$$\frac{\delta V_{off}}{\delta i} = \frac{V_{off,i} - V_{off,nominal}}{\sigma_i} \tag{44}$$

When all the sensitivities are known, it is possible to calculate the total variance of the input referred offset voltage, by applying the following formula:

$$\sigma^2_{Voff} = \sum_{i=1}^{n} \left(\left(\frac{\delta V_{off}}{\delta V T_i} \sigma_{V_T,i} \right)^2 + \left(\frac{\delta V_{off}}{\delta \beta_i} \sigma_{\beta,i} \right)^2 \right) \tag{45}$$

Where n is the number of transistors. So there are $2 \cdot n + 1$ separate offset voltage calculations necessary to determine the offset voltage variance. Formula (45) assumes, that the mismatches are uncorrelated. Two methods to perform these offset voltage calculations are described in the following two sections.

8.3 Offset voltage calculation method 1

In [25] a method is described to calculate the offset voltage. This method is programmed in Perl and is described now. The program runs a lot of different HSPICE simulations with various analog input voltages and uses the digital output.

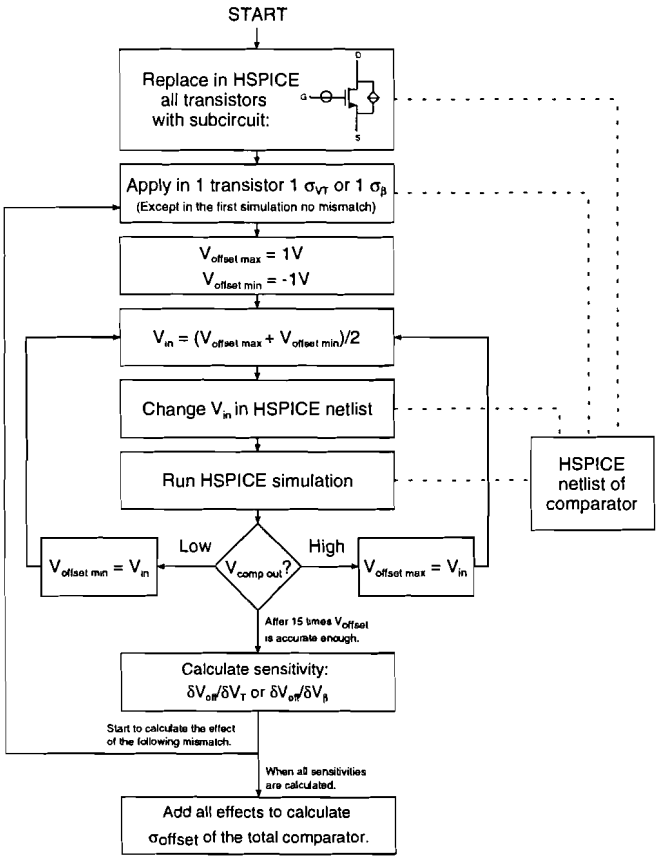


Figure 25: Program that calculates the input referred offset voltage.

In Figure 25 a flow diagram is plotted of the program. The input is a HSPICE netlist file of a comparator. The first step is to replace all the transistors with a subcircuit in which the mismatch can be

adjusted, see Figure 26. The current factor $\Delta\beta/\beta$ is modeled by a current source between the drain and the source. And the threshold voltage, V_T, is modeled by placing a voltage source before the gate.

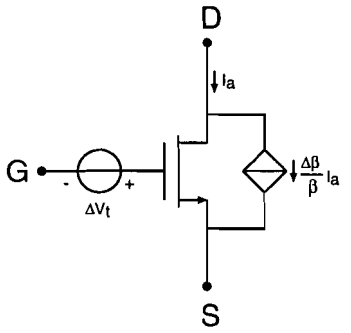


Figure 26: Each transistor is replaced by this subcircuit in order to apply mismatches to the transistor.

In the following block one mismatch is applied in the netlist, in order to calculate the offset voltage it causes. Only no mismatch is applied in the first offset voltage calculation in order to calculate the nominal offset ($V_{off,nominal}$). The offset calculation starts with the initialization of two variables, an upper boundary and a lower boundary for the offset ($V_{offsetmax} = 1V$ and $V_{offsetmin} = -1V$). Then the middle of these boundaries is applied as input voltage of the comparator ($v_{in} = (V_{offsetmax} + V_{offsetmin})/2$) in the HSPICE netlist. Next a simulation is executed in HSPICE and the digital output of the comparator is examined. When the output is high, the input voltage is above the offset voltage ($v_{in} > V_{off}$). So it is clear that the upper boundary can be lowered to v_{in}. When the output is low, the input voltage is beneath the offset voltage ($v_{in} < V_{off}$). So the lower boundary becomes v_{in}. So the goal is to narrow the interval of $V_{offsetmax} - V_{offsetmin}$, while the zero crossing of the output voltage is kept in the interval. This procedure is repeated until a sufficient accurate offset voltage is obtained. In the program 15 iterations are used. Next the sensitivity of the offset for one mismatch is calculated with equation 44. When all the sensitivities are calculated, the variance of the offset voltage of the comparator is calculated with formula 45.

file:/imec/other/adsba/STUDENTS/dejong/mismatch/mismatch1.perl

8.4 Offset voltage calculation method 2

In this section a new method is described to calculate the variance of the offset of a comparator caused by mismatches. This method uses one operation point simulation per mismatch, in contrast with the method described in section 8.3, that runs a lot of transient analysis to calculate the effect of one mismatch.

In order to calculate the offset the regenerative comparator is simplified to a model as shown in Figure 27. The clock transistor is closed in the metastable situation and represented by a resistor R_{clk}. The nodes connected to the clock transistor are called the regenerative nodes (A and B).

Block TC represents all transistors, that are between the input nodes, v_{in}, v_{ref} and the regenerative nodes. This TC block propagates the input voltage difference to the regenerative nodes ($v_A - v_B = TC(v_{in} - v_{ref})$).

The offset is calculated in three steps. The first step is to determine the function of TC. Second the effect of each mismatch on the voltage difference between the regenerative nodes is calculated. Last, all these voltage differences are calculated back to the input of the comparator with the function TC

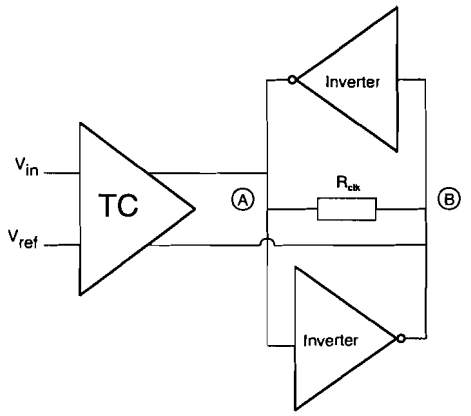


Figure 27: Model of the comparator used for the offset calculation.

and combined to the total offset of the comparator.

1. The function TC is determined by variating the input voltage difference ($v_{in} - v_{ref}$), while looking at the voltage difference $v_A - v_B$. The function TC of the comparator circuit of Figure 4 is plotted in figure 28.The figure shows several curves for the complete input range of v_{in} and v_{ref}, these curves coincide. Only a small region around zero is interesting for the mismatch calculation. This method uses the linear property of this region.

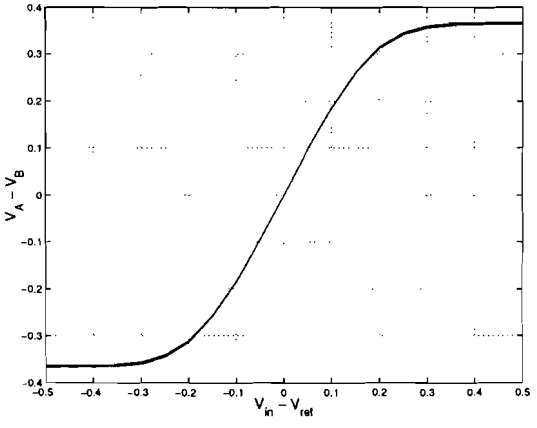


Figure 28: Function TC; the relation between the differential input voltage and the differential voltage on the regenerative nodes.

2. Both inputs are connected to a voltage source at the middle of the input range. A mismatch is applied as described in section 8.3. Then the voltage difference between the regenerative nodes is determined with an operation point simulation. This differential voltage is determined for every mismatch. One operation point simulation is performed without applying a mismatch to determine the nominal offset ($V_{off,nominal}$).

3. All the differential voltages on the regenerative nodes are calculated back to the input using the function TC. The formula (44) calculates the sensitivity of the input voltage for every mismatch in the comparator. The final variance of the offset voltage of the comparator is calculated with formula (45).

This method is programmed in Perl, that automatically executes the necessary HSPICE simulations and uses the simulation results to calculate the variance of the offset voltage.

8.5 Simulation results

The methods, that are described in section 8.3 and 8.4, are used to calculate the offset of the comparator of Figure 4. The simulation results of both methods are listed in table 2. The simulation time for method 1 is much longer, because a lot of transient simulations are performed per mismatch, although method 2 uses only one operation point simulation per mismatch. The nominal offset voltage calculated with method 1 is not zero, so method 1 has at least an inaccuracy of $4.2 \cdot 10^{-5}$ V.

Table 2: Simulation results of the mismatch calculations.

	method 1	method 2
σ_{Voff}	0.0180 V	0.0172 V
$V_{off,nominal}$	$4.2 \cdot 10^{-5}$ V	0
Simulation time	27 min.	30 sec.

The sensitivity per mismatch per method is listed in table 3. The difference between the two methods is small. An opposite value of the sensitivity in a transistor pair is expected, because the comparator is designed completely differential. This is not always the case for method 1. For example $\delta V_{off}/\delta V_T$ for m1a (1.00 V) is not the opposite of m1b (-1.04 V). This is always correct for method 2. So the accuracy of method 2 is bigger than the accuracy of method 1. The value of the variance of the offset voltage of method 2 will be used in simulation.

Table 3: Comparison of the sensitivities calculated by method 1 and 2.

Transistor	$\delta V_{off}/\delta V_T$		$\delta V_{off}/\delta\beta$	
	method 1	method 2	method 1	method 2
m1a	1.00	1.00	0.10	0.082
m1b	-1.04	-1.00	-0.081	-0.082
m2a	0.28	0.28	0.081	0.081
m2b	-0.30	-0.28	-0.090	-0.081
m3a	-0.34	-0.32	-0.11	-0.099
m3b	0.32	0.32	0.11	0.099
m6a	-0.34	-0.33	0.12	0.14
m6b	0.33	0.33	-0.12	-0.14
m7a	-0.65	-0.61	-0.48	-0.42
m7b	0.64	0.61	0.48	0.42
m8a	-0.41	-0.40	0.23	0.25
m8b	0.40	0.40	-0.23	-0.25
m9a	-0.038	-0.029	0.13	0.14
m9b	0.028	0.029	-0.14	-0.14

The assumption, that the mismatches are uncorrelated, is probably not correct. However there is no information about correlation between mismatches whatsoever. The following conclusion can be drawn: Method 2 is faster and more accurate, only a source of inaccuracy is the assumption of no correlation between the mismatches.

The variance of the offset voltage (0.0172 V) is saved in the input parameter list of the C++ model. Offset voltages for each comparator are generated randomly with this variance at the beginning of a simulation. The mismatch block in every comparator block of Figure 9 adds these DC offset voltages to the input voltage of the comparator.

9 Calculating the ADC performance

Important specifications of an ADC are Signal to Noise Ratio (SNR) and Integral and Differential NonLinearity (INL/DNL). They provide a quantitative measure of the performance of the ADC. These are calculated after every simulation in C++. In section 9.1 the calculation of the SNR and also an additional specification, the Spurious Free Dynamic Range (SFDR) is described. The calculation of the INL/DNL is treated in section 9.2.

9.1 Signal to Noise Ratio

The SNR is the most important specification of an ADC. The SNR takes in account not only the quantization noise but also all the non-idealities, such as distortion, nonlinearities and sampling time uncertainty. The signal to noise ratio is the signal power (S) divided by the noise power (N_{ADC}) and is usually expressed in decibels, see the following equation.

$$SNR = 10 \cdot^{10} \log\left(\frac{S}{N_{ADC}}\right) \quad [dB] \tag{46}$$

When the input signal is of the form $v_{in}(t) = A \cdot \sin(2\pi f t)$, the power is of the input signal is given by:

$$S = \frac{A^2}{2} \tag{47}$$

The noise power is more difficult to calculate. Two DFT's are performed in order to calculate the noise power. One DFT is calculated on the input signal and one on the output code. The two calculated spectra are subtracted from each other. The remaining is the noise added by the ADC. The Noise power calculation is now described in more detail.

The input signal contains $clklength \cdot clkperiods$ elements and the output code contains $clkperiods$ elements, see section 4.1. The values of the input signal at the sampling instants are used for the calculation of the input signal spectrum. The number of values used for the input signal is in this way equal to the number of output codes. The noise spectrum is given by calculating the DFT on the difference between the input signal and the output code, see equation (48).

$$N(f) = \frac{DFT(v_{in}[k \cdot clklength] - code\_out[k])}{clkperiods} \tag{48}$$

After a DFT calculation, the results must be divided by the number of used points ($clkperiods$) to get the same voltage y-axis as in the time domain. Integrating the noise spectrum gives the noise power added by the ADC:

$$N_{ADC} = \int_0^{f_{clk/2}} |N(f)|^2 df \tag{49}$$

The DFT calculation is performed in C++ by a FFT, see [27]. After the noise power is calculated, the SNR can be calculated with formula (46). The Effective Number Of Bits (ENOB) can be calculated easily, when the SNR is known. The relation between SNR and ENOB is given by (see [21]):

$$ENOB = \frac{SNR(dB) - 1.76}{6.02} \tag{50}$$

The spectrum of the output and the SFDR are also calculated in every simulation in C++. In Figure 29 is plotted an example of an output spectrum. No windowing is applied. The frequency of the input signal is 97 MHz and the ADC has four output bits and the clock frequency is 400 MHz.

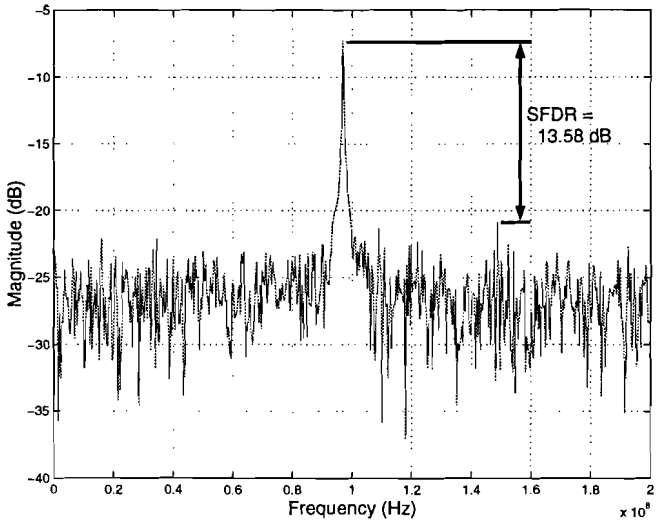


Figure 29: An example of the spectrum of the output code ($Fin = 97MHz$)

The SFDR specification is used when the spectral purity of the ADC is important. The SFDR is the ratio between the input signal and the largest distortion component. The SFDR is 13.58 dB in the example of Figure 29.

file:/imec/other/adsba/STUDENTS/dejong/c/include/snr.h

9.2 INL and DNL

INL and DNL are measures for the linearity of an ADC. INL is the difference between the transition level of an ideal ADC ($V_{ideal\ t}$) and a tested ADC (V_t). A transition level is the input voltage at which the digital output code changes. DNL is the difference between two adjacent transition levels of the tested ADC minus 1·LSB. The transfer functions from analog input to digital output of an ideal ADC and an ADC under test are plotted in Figure 30.

A code density calculation [28] uses a histogram to calculate the INL and DNL of an ADC. This calculation is now explained. A triangular waveform is applied at the input of an ADC, and the number of occurrence for every output code is counted. A histogram is composed from the counted samples per bin. An equal number of samples per bin is expected. However when the ADC has nonlinearities, the occurrence of an output code will change. The INL and DNL can be calculated from the histogram.

Nonlinearity in the ramp is a problem in practice, because the accuracy of the input signal must be better than the tested ADC. Using a sine wave is a good solution. A sine wave can be generated with very low harmonic distortion. The number of samples per bin are not expected to be equal in this case. So the results have to be transformed back to the case with a triangular wave at the input. Most HSPICE and C++ model simulations use a sine wave as input signal, so with a back transformation the INL and DNL can be calculated.

The cumulative histogram is calculated from the Histogram $H[j]$ with (51).

$$CH[i] = \sum_{j=0}^{i} H[j] \tag{51}$$

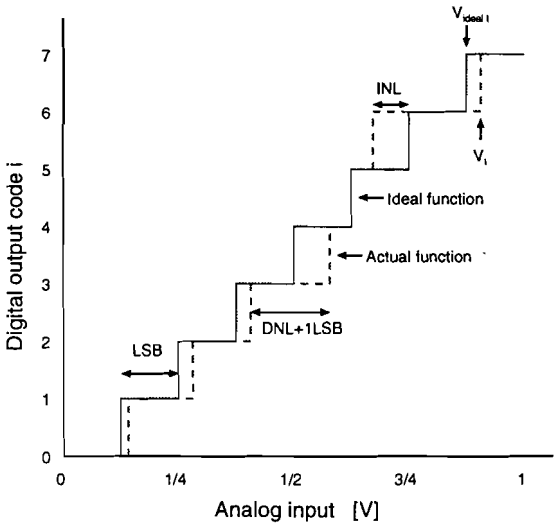


Figure 30: Transfer curve of a 3-bit ADC

Further the transition levels (V_t) are calculated with (52) or (53). Formula (52) is used for a ramp input. Formula (53) is the back transformation, in case a sine wave is used.

$$V_t[i] = CH[i] \tag{52}$$

$$V_t[i] = -A \cdot \cos\left(\frac{\pi \cdot CH[i]}{N_{total}}\right) \tag{53}$$

INL is the deviation between the transition level of the tested ADC and an ideal ADC expressed in LSB, see (54).

$$INL[i] = \frac{V_t[i] - V_{ideal\,t}[i]}{LSB} \tag{54}$$

The ideal transient level are given by $V_{ideal\,t}[i] = LSB \cdot i$. The expected distance between two transition levels of a tested ADC is 1·LSB. DNL is the deviation of this distance expressed in LSB, see (55) .

$$DNL[i] = \frac{V_t[i+1] - V_t[i] - LSB}{LSB} \tag{55}$$

To estimate the number of samples needed for a certain accuracy formula (56) is used from [28]. The minimum number of samples is N and n is the number of bits of the ADC. The value of Φ can be looked up in the table of the standard normal distribution. The confidence is $100(1-\alpha)$ with a β bit precision. For example, when you want to know the INL and DNL of a 4- bit ADC with a confidence of 95% and with a precision of 0.1 bit, 9600 samples are needed.

$$N \geq \frac{\Phi(1-\alpha/2)^2 \pi 2^{n-1}}{\beta^2} \tag{56}$$

The advantage of the code density method is, that it can be performed at full speed, so all the dynamic nonlinearities are taken into account. However the need for much samples for an accurate estimation of the INL and DNL is a major disadvantage.

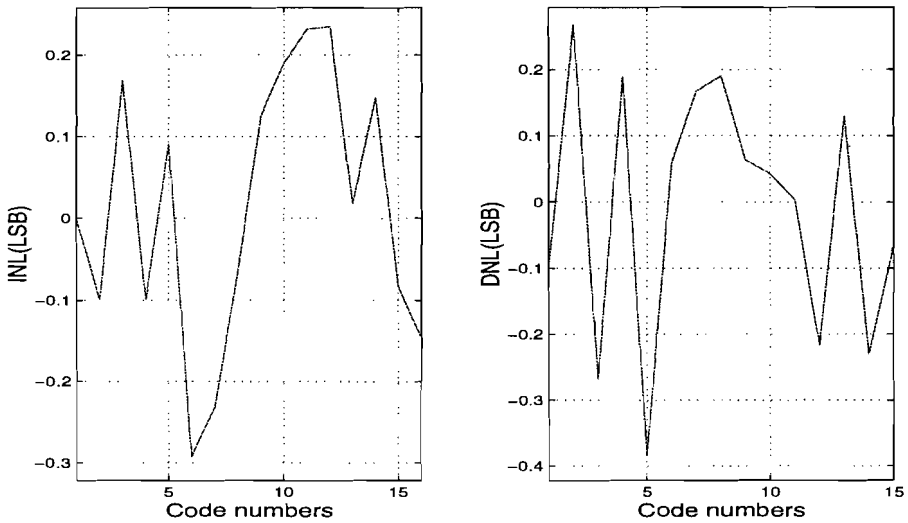


Figure 31: An example of the results of a INL and DNL calculation.

In Figure 31 is an example plotted of an INL and DNL calculation. The mismatches in the resistors of the reference ladder is set to 5% in this case and 400 samples are used of a 4-bit ADC.

file:/imec/other/adsba/STUDENTS/dejong/c/include/inl_dnl.h

10 Simulation results

The C++ model is compared with HSPICE in this chapter. The performance of the model is checked, both the accuracy and the simulation time.

10.1 Simulation parameters

In this section all the parameters are enumerated, that have to be adjusted for every simulation. These values can be changed easily in the C++ model.

- Timing parameters: $T_{duration}$, T_{sim} and T_{clk}
- The input signal has to be defined ($v_{in}[clkperiods \cdot clklength]$).
- The number of bits, n and the reference voltages V_{ref}^{+}, V_{ref}^{-}
- The parameters for the input feedthrough are: C_{gs} of the input transistor of the comparator and R, the resistor value of the reference ladder. The variance of the resistor values is saved in order to calculate the DC reference voltages.
- The variance of the clock generator jitter (Δt_{CG}) and the internal jitter (Δt_{int}).
- The parameters of the comparator are the poles and zeros of the transfer function of the input stage and the variance of the offset voltage.

These parameters are adjusted in the input parameter list, see Figure 9. A Gaussian random generator is implemented, to choose values for the resistors in the reference ladder, the clock jitter and the input referred offset voltage of the comparators.

file: /imec/other/adsba/STUDENTS/dejong/c/include/globals.h
file: /imec/other/adsba/STUDENTS/dejong/c/include/random.h

10.2 Accuracy

The accuracy of the C++ model is compared with the HSPICE transient simulations for a 4-bit 400 MHz ADC. To verify the accuracy, several transient simulations in HSPICE are performed for different input frequencies. The C++ model should generate samples that accurately match the HSPICE simulation. The input signal generated in HSPICE simulations is saved and used by the C++ model, so both simulations use exactly the same input signal array.

The comparator of section 6.3 is used and all mismatches and clock jitter are adjusted to zero. The length of the simulation is chosen the same for all simulations, namely $T_{duration} = 1000\ ns$. The sampling time is 2.5 ns ($= T_{clk}$), so 400 samples are generated. The simulation time step is 5 ps ($= T_{sim}$), this results in 500 simulation points per clock period. To compare the C++ model with HSPICE the output codes are compared, as shown in Figure 32.

In Figure 32 a part of the generated HSPICE and C++ output codes are plotted, also is plotted the output code of an ideal quantizer. The left y-axis is the analog input voltage, the sine curve belongs to this axis. The horizontal dotted lines are the quantization levels of the ADC. The right axis displays the 16 digital output levels. The ADC samples the sine wave every 2.5 ns, which results in output values indicated by a o, $\triangle$ or $\times$. Figure 32 shows that the ideal quantizer ($\times$) differs 17 of the 40

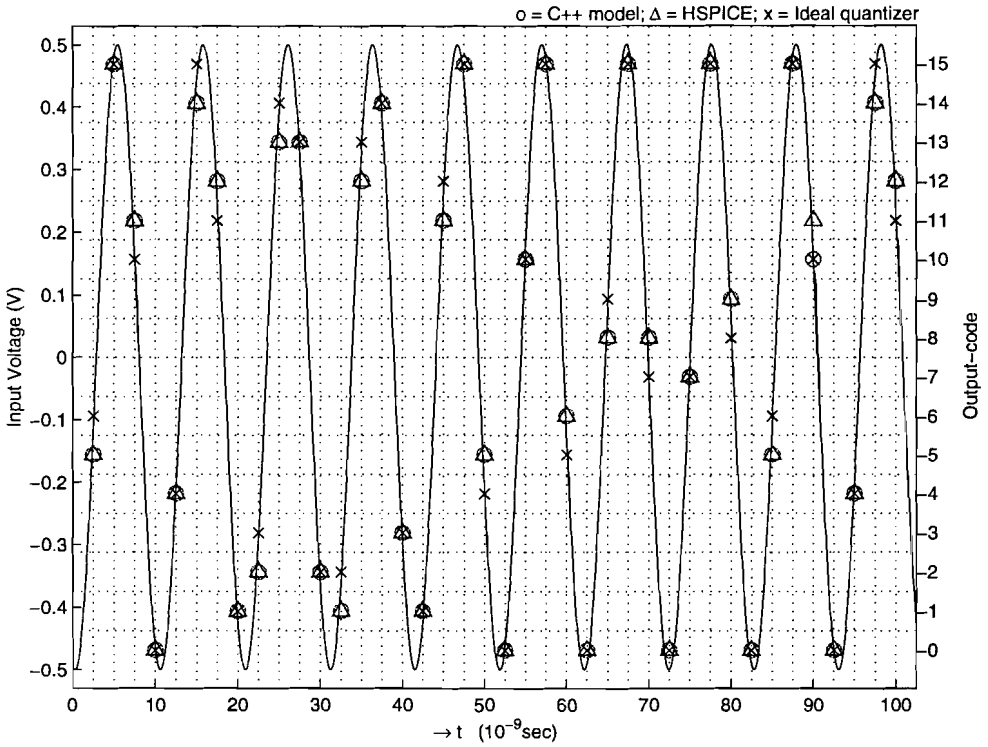


Figure 32: Transient 4-bit flash ADC C++ model compared with HSPICE.

samples from the HSPICE simulations (△). Also shown is that the C++ model (o) differs one time from HSPICE at the sample point of 90ns.

Table 4: Percentage of wrong output codes of the C++ model and an ideal quantizer compared with HSPICE.

Input freq. [MHz]	HSPICE in comparison with:	
	Ideal quantizer	our C++ model
9.7	17/400	0
97	153/400	6/400
147+33	132/400	3/400

In Table 4 the differences between the output codes of HSPICE and C++ are listed for various input frequencies. The last row shows the results for the input signal $v_{in}(t) = 0.5 \cdot \cos(2\pi\ 147 \cdot 10^6 t) + 0.5 \cdot \cos(2\pi\ 33 \cdot 10^6 t)$.

With an input frequency of 97 MHz, 153 of the the 400 output codes of the ideal quantizer model are different compared to HSPICE. The C++ model tracks this HSPICE output code with only 6 errors. For low input frequencies, there is no difference between HSPICE and the C++ model.

HSPICE is compared with an ideal quantizer, because it shows that the behavior of the HSPICE model is more complex than an ideal quantizer, so the C++ model has to include non-idealities. However the 153 different output codes do not mean that the HSPICE model is designed badly. The 153 deviations in the code are explained by the delay from the input to the output of the ADC. This delay does not result in a worse SNR.

The differences between the HSPICE and C++ results can be explained by the following error sources:

1. The transfer functions of the input feedthrough model of chapter 5 differ a few dB's from HSPICE.
2. The gradual increase of the resistance of the clock transistor at the beginning of the compare phase is not taken into account.
3. The C_{gs} and C_{ds} of the clock transistor inject some current into the regenerative nodes, while the clock voltage changes.
4. Analog filters are approximated by digital filters.

The 6 errors in case a 97 MHZ input signal is used are explained in the following way: 5 errors are caused by error source 2, and 1 error is caused by error source 1. The effect of error source 3 is marginal. No better results were obtained, when a fourth order filter for the input stage of the comparator transfer function is used.

10.3 Simulation time

To compare the simulation CPU time results of the C++ model with full transistor-level HSPICE transient simulations, the same time step, sample frequency and simulation length are taken as above. All the simulations have been performed on a HP9000 design station with a HP-UX10 operating system.

Table 5 gives an indication of the simulation speed improvement compared with HSPICE. Simulating the 5-bit case in HSPICE resulted in memory problems. The reported simulation times in C++ don't include the loading and saving of the input signal and output data.

Table 5: CPU-time of C++ model compared with HSPICE.

# bits ADC	HSPICE sim. time [s]	C++ model sim. time [s]
3	21,000	3.35
4	190,000	7.48
5	n.a.	15.17

The SNR calculation takes 0.01 seconds and INL/DNL calculation for a 8-bit ADC takes 0.04 seconds. The most time consuming step in the C++ model is the filter function of the input feedthrough and the transconductance. These functions take 0.49 seconds per comparator. The total C++ simulation time can be approximated by the following formula:

$$Simulation\ time \approx 0.49 \cdot (2^n - 1) \quad [sec.] \tag{57}$$

n is the number of bits in the ADC. When T_{sim} is taken larger, the simulation time decreases and the number of errors increases.

11 Conclusions

In this work an efficient and accurate model for an n-bits ADC is developed. The model meets with the assignment demands: it is flexible, efficiently implemented in C++ and based only on circuit and device parameters. These parameters can be extracted from simple HSPICE simulations.

The feedthrough of the input signal on the reference ladder is caused by the capacitive coupling between the input nodes of the comparator, that couples the input to the resistor ladder. The input feedthrough is dependent on the number of bits, the comparator circuit and the value of the resistor in the reference ladder. The presented model can deal with all different combinations of these parameters.

The main effect of mismatches between transistors in a comparator is an offset voltage at the input. A new method is described that can predict the variance of the offset voltage of a comparator in a faster and more accurate way.

There is little published about clock jitter in an ADC. All blocks in the clock path, from the oscillator to the ADC included, contribute to the clock jitter. A summary is given about the various aspects of clock jitter and a realistic way to model the clock jitter is described.

The comparator is the most important part of the ADC. Three different models are studied and finally the best model is selected. The final model consists out of an input stage filter and an ideal comparator.

To verify the validity of the model, a comparison is made with a 4 bit ADC, which is available in HSPICE. The same input signal is applied to HSPICE and the C++ model and the output codes are compared. The results show a good agreement between the simulated output codes of HSPICE and the model. The model is at least 6000 times faster than HSPICE.

Finally a program is written, that determines the performance of the ADC after each simulation, by calculating the SNR and the INL/DNL values.

It is important to be careful with applying conclusions, that are made in this report, on other ADCs, because the conclusions are only valid on this circuit in HSPICE.

The used ADC circuit is only the base of many different ADC architectures. A real ADC contains more additional circuits, such as interpolating, folding, S/H, differential reference ladder and integrating resistor network. So, in a more realistic ADC with additional circuits, the needed model can be entirely different.

Keep in mind that HSPICE gives no perfect reproduction of a real circuit, however it is the best available way to have an impression of a real circuit. No layout of the ADC is available, so all the effects of extra parasitics are not included. This can give rise to a complete different model.

12 Further investigations

Subjects that can be studied as an extension of the current ADC model are listed in this chapter.

A possible way to decrease drastically the simulation time in C++, is to use larger simulation time steps (T_{sim}). However when T_{sim} is increased, the number of errors increases also. A topic for further investigation is the origin of the errors when T_{sim} is increased.

A possible explanation for the extra errors in a situation with larger simulation time steps is the inaccuracy in the transformation of the filters. A solution for better results can be pre-warping or the use of the Simpson integral for the s to z transformation.

The input feedthrough transfer functions are calculated with a mathematical model, however it is also possible to calculate the transfer functions with an AC analysis in HSPICE. This should be more accurate. It is interesting to test if these transfer functions from HSPICE AC simulations result in zero errors caused by the input feedthrough model.

A topic for further investigation is a comparator model, that includes the gradual change of the resistance of the clock transistor in the transition from reset to compare phase.

The effect of substrate noise in a comparator can be studied.

References

[1] G. E. Gielen and J. E. Franca, "CAD Tools for data converter design: An overview," *Transactions on circuits and systems-II: Analog and digital signal processing*, vol. 43, pp. 77–88, Feb 1996.

[2] E. I. Getreu, A. D. Hadiwdjaja, and J. M. Brinch, "An Integrated-Circuit Comparator Macromodel," *IEEE Journal of Solid-State Circuits*, vol. SC-11, pp. 826–833, Dec 1976.

[3] C. Turchetti and G. Masetti, "A macromodel for integrated all-MOS operational amplifiers," *IEEE Journal of Solid-State Circuits*, vol. 18, pp. 389–394, Aug 1983.

[4] E. Liu, A. Sangiovanni-Vincentelli, G. Gielen, and P. Gray, "A behavioral representation for Nyquist rate A/D converters," *ICCAD*, pp. 386–389, 1991.

[5] E. Liu and A. Sangiovanni-Vincentelli, "Behavioral simulation for noise in mixed-mode samples-data systems," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 322–6, 1992.

[6] E. Liu and A. Sangiovanni-Vincentelli, "Verification of Nyquist data converters using behavioral simulation," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 14, pp. 493–502, April 1995.

[7] A. Baccigalupi and M. D'Apuzzo, "Analog-to-digital converter modeling: a survey," *Measurement.*, vol. 19, pp. 139–46, Nov.-Dec. 1996.

[8] G. M. Yin and F. O. E. W. Sansen, "A high-speed CMOS comparator with 8-b resolution," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 208–11, Feb 1992.

[9] P. Cusinato, M. Bruccoleri, D. D. Caviglia, and M. Valle, "Analysis of the behavior of a dynamic latch comparator," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 45, pp. 294–298, March 1998.

[10] G. Ruan, "A behavioral model of A/D converters using a mixed-mode simulator," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 283–90, March 1991.

[11] H. A. Mantooth and P. E. Allen, "Behavioral simulation of a 3-bit flash ADC," *IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 1356–9, 1990.

[12] F. Kaess, R. Kanan, B. Hochet, and M. Declercq, "New encoding scheme for high-speed flash ADC's," *Proceedings of 1997 IEEE International Symposium on Circuits and Systems.Circuits and Systems in the Information Age, ISCAS*, vol. 1, pp. 5–8, 1997.

[13] P. Wambacq, G. Vandersteen, S. Donnay, M. Engels, I. Bolsens, E. Lauwers, P. Vanassche, and G. Gielen, "High-level simulation and power modeling of mixed-signal front-ends for digital telecommunications," *Proceedings of the IEEE Int. Conference on Electronics, Circuits and Systems*, pp. 420–430, Sep. 1999.

[14] G. H. Golub and C. F. V. Loan, *Matrix computations.* The Johns Hopkins University Press, Baltimore, 1990.

[15] P. Wambacq, G. Gielen, and W. Sansen, "ISAAC: a symbolic analysis program for linear and weakly nonlinear analogue circuits," *Proc. 1st International Workshop on Symbolic Methods and Applications to Circuit Design, Bagneux (France)*, 1991.

[16] A. G. W. Venes and R. J. van de Plassche, "An 80 MHz 80 mW 8 b CMOS folding A/D converter with distributed T/H preprocessing," *IEEE International Solid State Circuits Conference*, pp. 318–19, 1996.

[17] R. Roovers, *High speed A/D converters in standard CMOS technology*. PhD thesis, Katholieke Universiteit Leuven departement elktrotechniek, 1996.

[18] A. Hajimiri and T. H. Lee, "A general theory of phase noise in electrical oscillators," *IEEE Journal of Solid State Circuits*, vol. 33, pp. 179–94, Feb 1998.

[19] A. Hajimiri, S. Limotyrakis, and T. H. Lee, "Jitter and phase noise in ring oscillators," *IEEE Journal of Solid-State Circuits*, vol. 34, pp. 790–804, June 1999.

[20] B. Razavi, *RF Microelectronics*. Prentice Hall PTR, Upper Saddle River NJ, 98.

[21] R. J. van de Plassche, *High-speed and high-resolution analog-to-digital and digital-to-analog converters*. Kluwer academic publishers Dordrecht, 1989.

[22] S. S. Awad, "Analysis of accumulated timing-jitter in the time domain," *IEEE Transactions on Instrumentation and Measurement*, vol. 47, pp. 69–73, Feb 1998.

[23] H. Kobayashi, M. Morimura, K. Kobayashi, and Y. Onaya, "Aperture jitter effects in wideband sampling systems," *Proceedings of the 16th IEEE Instrumentation and Measurement Technology Conference*, vol. 2, pp. 880–884, 1999.

[24] M. Shinagawa, Y. Akazawa, and T. Wakimoto, "Jitter analysis of high-speed sampling systems," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 220–224, feb 1990.

[25] G. van der Plas, J. Vandenbussche, W. Verhaegen, G. Gielen, and W. Sansen, "Statistical behavioral modeling for A/D-converters," *The 6th IEEE international Conference on Electronics, Circuits and Systems*, pp. 1319–1323, 1999.

[26] M. J. M. Pelgrom and A. C. J. D. A. P. G. Welbers, "Matching properties of MOS transistors," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 1433–9, Oct 1989.

[27] S. J.Orfanidis, *Introduction to signal processing*. Prentice-Hall Signal Processing series Upper Saddle River, New Jersey, 1996.

[28] J. Doernberg, H. S. Lee, and D. A. Hodges, "Full-speed testing of A/D converters," *IEEE Journal of Solid-State Circuits*, vol. SC-19, pp. 820–827, Dec 1984.

HIGH-LEVEL MODELING OF A HIGH-SPEED FLASH A/D CONVERTER FOR MIXED-SIGNAL SIMULATIONS OF DIGITAL TELECOMMUNICATION FRONT-ENDS

J. Compiet, P. de Jong, P. Wambacq, G. Vandersteen, S. Donnay, M. Engels and I. Bolsens*

IMEC v.z.w., DESICS-MIRA,
Kapeldreef 75, B-3001 Leuven, Belgium
John.Compiet@imec.be

ABSTRACT

A hierarchical high-level model of a high-speed flash ADC is presented. The input parameter list is extracted from a 400MHz, 4-bit, flash ADC designed in HSPICE in a $0.35\mu m$ CMOS technology. A speedup in simulation time of 5000 is reported compared to the 3-bit flash ADC HSPICE simulations. The accuracy of the model is verified with HSPICE simulations and shows a good agreement.

I. INTRODUCTION

Evolution of deep-submicron CMOS technology has made it possible to integrate more and more analog functionality together with large digital processing systems on a single transceiver chip. The exploration and design of mixed-signal systems can be supported with accurate high-level mixed-signal simulation tools. An important building block in mixed-signal systems is an analog-to-digital converter (ADC). However, high-speed ADC models that are supplied by commercial high-level simulation tools [1] [2] often only take into account the nominal behavior (e.g. ideal sampling and quantization). As a result, the simulation results are often inaccurate, leading to wrong conclusions/decisions at the system level.

There is a need for accurate high-level models of analog blocks that can be used in a front-end architecture simulation tool [9]. For such simulation tool it is very important that the models of the different front-end blocks can be evaluated efficiently. The difficulty in modeling analog blocks at the system level is that, while the first-order, linear behavior is relatively easily modeled, the nonlinear behavior requires a careful study and even advanced mathematical methods [6] [8]. In this paper a high-level ADC model is presented that combines a high accuracy with evaluation efficiency. The main advantage of this model is that it still has a link with physical design parameters, such as transconductances, capacitors, poles and zeros. This makes the model also useful for top-down design. Further, a high accuracy is obtained and due to its efficient implementation in C++ the simulation time is drastically reduced.

The paper is organized as follows. Section II gives an introduction in the used high-level model for a flash ADC architecture. The models for the different sub-blocks are explained in section III, IV and V. The implementation of all these sub-blocks into the C++ model is explained in section VI. The experimental results comparing the model with transistor level simulations in HSPICE are given in section VII and finally conclusions are drawn in section VIII.

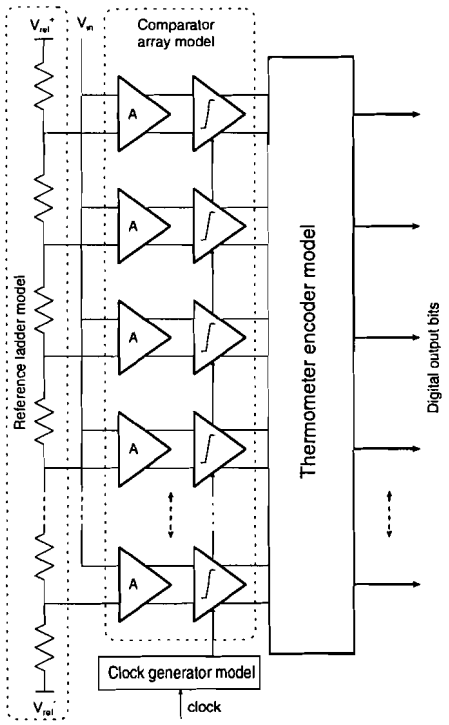


Figure 1: Hierarchical flash ADC model

II. A HIERARCHICAL HIGH-LEVEL ADC MODEL

The modeling of important second-order effects in ADCs is not straightforward since several of these effects

* MS degree student at the Eindhoven University of Technology, The Netherlands

highly depend on the architecture of the ADC. Therefore, first the non-linearities of the simplest high-speed ADC (a flash ADC) are studied and modeled. The high-speed flash ADC model consists of building blocks like comparators, a reference ladder and an encoding block, which are accurately described in C++. With these basic building blocks different high-speed ADC architectures can be hierarchically modeled on a higher abstraction level. All the sub-blocks, as depicted in Figure 1 for a flash ADC model, are accurately modeled with transistor and/or design parameters. The ladder network model takes into account the input signal feed-through and mismatches (see section III). The most important block in the model of a flash ADC is the modeling of a high-speed comparator and is explained in more detail in section IV. The modeling of the encoder, discussed in section V, can be adapted to the number of bits, encoding method and takes into account the extra delay. This model also includes error correction techniques.

As reference case to verify this model, a 3.3V 400-MHz, 4-bit flash ADC has been designed in HSPICE in a standard digital $0.35\mu m$ CMOS technology. The comparator architecture is based on [10] and the encoding is described in [5]. The input model parameters for each sub-block are derived from simple HSPICE operating point and AC simulations.

III. REFERENCE LADDER MODEL

A. Model implementation

The reference ladder model generates the $2^n - 1$ reference voltages for the n–bit ADC. Non-idealities of the reference ladder are subdivided into two parts: input signal feed-through and mismatches of the resistors in the ladder network. First, the input signal feed-through is derived as function of the resistor value R, the total coupling capacitance C between the input signal node and a reference ladder tap. These parameters can be easily simulated in HSPICE and transfered to the input parameters list of the model. The input feed-through has a high-pass filter characteristic on the reference ladder taps. The variation of the reference levels as function of the input signal together with the mismatch of the resistors in the reference ladder distorts the INL and DNL performance of the ADC.

B. Input signal feed-through

The input feed-through is modeled by calculating the transfer-function from the input signal to a node on the reference ladder. The variation of the voltage in the reference ladder is afterwards subtracted from the input voltage of the comparator. The model in Figure 2a is used.

The capacitor C is the total capacity between the two inputs of a comparator. When the reference ladder is connected between V_{ref}^+ and V_{ref}^-, and the input voltage is V_{in}, then $V_o^+ = V_{ref}^+ - V_{in}$ and $V_o^- = V_{ref}^- - V_{in}$. The

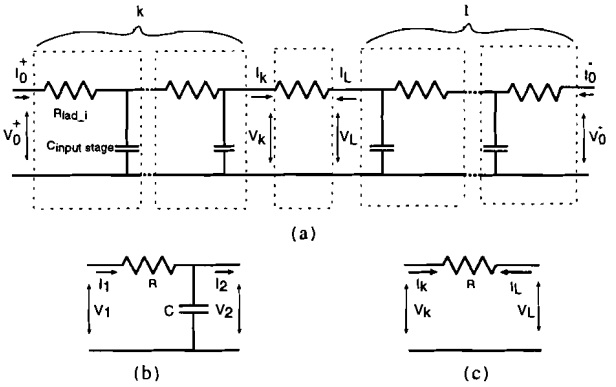


Figure 2: Input signal feed-through model

model is subdivided in two types of blocks, see Figure 2b and 2c. By placing k sections of Figure 2b to the left and l sections to the right of the middle section, Figure 2c, it is possible to calculate V_l for any tap and any length of the ladder, according to the formula $k = 2^n - 1 - l$.

$$\begin{bmatrix} V_2 \\ I_2 \end{bmatrix} = W \cdot \begin{bmatrix} V_1 \\ I_1 \end{bmatrix} \tag{1}$$

with $W = \begin{bmatrix} 1 & -R \\ -j\omega C & 1 + j\omega RC \end{bmatrix}$

$$\begin{bmatrix} V_k \\ I_k \end{bmatrix} = M \cdot \begin{bmatrix} V_l \\ I_l \end{bmatrix} \tag{2}$$

with $M = \begin{bmatrix} 1 & -R \\ 0 & -1 \end{bmatrix}$.

Formulas (1) and (2) describe the transfer-matrix of Figure 2b and 2c. The relation between V_l, I_l and V_o^-, I_o^- is given by formula (3):

$$\begin{bmatrix} V_l \\ I_l \end{bmatrix} = W^l \cdot \begin{bmatrix} V_0^- \\ I_0^- \end{bmatrix} \tag{3}$$

Combining (3) with (2) according to Figure 2a results in:

$$W^l \cdot \begin{bmatrix} V_0^- \\ I_0^- \end{bmatrix} = M \cdot W^k \cdot \begin{bmatrix} V_0^+ \\ I_0^+ \end{bmatrix} \tag{4}$$

The formulas (3) and (4) form a system with four equations and four unknown (V_l, I_l, I_o^+, I_o^-). When V_l is solved from this system, the order of the transfer-function is too high for efficient implementation in C++. Therefore an approximation is made:

$$W^k = [I + B]^k = I + k \cdot B + \frac{k \cdot (k-1) \cdot B \cdot B}{2} \tag{5}$$

with $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & -R \\ -j\omega C & j\omega RC \end{bmatrix}$

Formula (5) describes a second-order approximation for W^k. This approximation is valid [4] if the absolute

values of the eigenvalues are much smaller than 1. The approximation results in a fourth order transfer-function for V_l/V_{in}, which is implemented in the model as an IIR-filter.

C. Mismatch

An ideal ADC has equally spaced reference-voltages. But by variations in the resistor values of the ladder, due to process variations, the reference voltages differ from the ideal voltages, and cause quantization errors in the ADC. This offset is modeled by generating 2^n resistor values having a Gaussian distribution.

IV. HIGH-LEVEL COMPARATOR MODEL

A. Hierarchical comparator model

The high-speed comparator in the ADC model as shown in Figure 1 is based on the architecture used in [10]. The proposed general comparator model consists of three hierarchical blocks: a transconductor as input stage, a regenerative differential comparator and output latches as depicted in Figure 3.

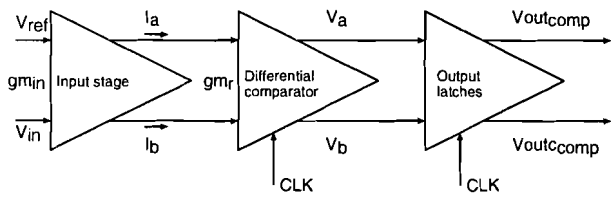


Figure 3: General hierarchical macro model for a high-speed comparator

In the following subsections the different sub-blocks will be discussed in more detail.

B. Transconductor model

The input amplifier of the comparator amplifies the differential input voltage and drives the regenerative part of the comparator. The transconductor is modeled with a linear transfer function with poles and zeros. A first-order transfer function was sufficient to model the behavior of the transconductor accurately and is implemented in the model as an IIR filter with coefficients that are automatically derived from the user-specified poles and zeros.

C. Regenerative comparator model

The differential comparator implementation is the main part of the model. This model takes into account the comparator delay, its saturation behavior, as well as mismatches. A clocked comparator has two stable positions (VSS and VDD = 3.3V) and one meta-stable position (VDD/2) to which the comparator is reset every clock period. The basic comparator operation is modeled with a nonlinear differential equation:

$$\frac{\delta\, Vout}{\delta\, t} = f(V_{\text{out}}(t), t) \tag{6}$$

with the comparator output $V_a(t) - V_b(t)$ as $V_{\text{out}}(t)$ and the differential input current $gm_{\text{in}} \cdot (V_{\text{ref}}(t) - V_{\text{in}}(t))$ as $I_a(t) - I_b(t)$. In (6) $\dot{f} > 0$ in the meta-stable point and this first derivative describes the exponential growth in the beginning of the comparison phase. The function $f = 0$ and the deriviate $\dot{f} < 0$ in the stable points.

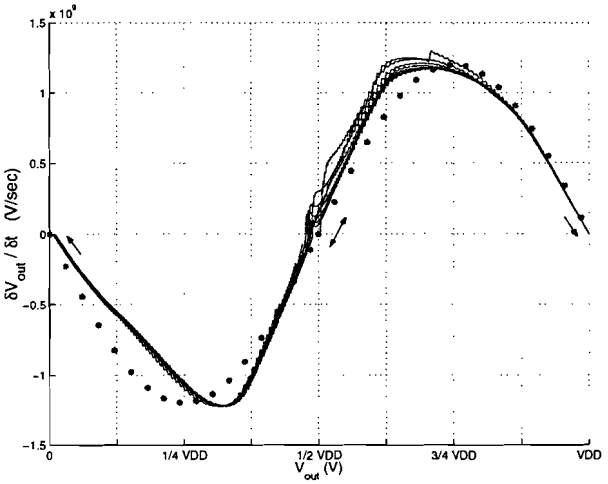


Figure 4: Sine-wave dependency of $\delta V_{out}(t)/\delta t$ as function of $V_{out}(t)$ is feasible as modeling function

In HSPICE the transients are studied for different input voltages at the comparator outputs. These functions are differentiated with respect to time and plotted in Figure 4 as function of the output-voltage $V_{\text{out}}(t)$. The sine-wave like behavior as depicted in Figure 4 is modeled with physical input parameters which defines the speed of the comparator:

$$\frac{\delta\, V_{\text{out}}(t)}{\delta\, t} = -A \cdot \sin\left(\frac{2\pi \cdot V_{\text{out}}(t)}{VDD}\right) \tag{7}$$

$$\frac{\delta\, V_{\text{out}}(t)}{\delta\, t}\Big|_{V_{\text{out}}(t)\to 0} \approx -A \cdot \frac{2\pi \cdot V_{\text{out}}(t)}{VDD} \tag{8}$$

In [10] it is proven that:

$$\frac{2\pi A \cdot V_{\text{out}}(t)}{VDD} = \frac{gm_{\text{r}}}{C} \cdot V_{\text{out}}(t) \Rightarrow A = \frac{gm_{\text{r}}}{C} \cdot \frac{VDD}{2\pi} \tag{9}$$

From Figure 4 it can be seen that for values of $V_{out}(t)$ in the neighborhood of $\pm$ VDD/2 the derivate $\delta V_{out}(t)/\delta t$ is depending on the value of the input current. For values $V_{out}(t)$ in the neighborhood of the stable points, $\delta V_{out}(t)/\delta t$ is independent of the value of the input current. If (9) is filled out in (7) and corrected for the difference in value of the input currents, the suggested differential equation for modeling the speed of the comparator

is:

$$\frac{\delta V_{\text{out}}(t)}{\delta t} = \frac{I_{\text{a}}(t) - I_{\text{b}}(t)}{C} - \frac{gm_{\text{r}}}{C} \cdot \frac{VDD}{2\pi} \cdot \sin\left(\frac{2\pi . V_{\text{out}}(t)}{VDD}\right) \quad (10)$$

It is seen that this equation takes into account the time delay at the output of the comparator, which occurs due to variations in the differential input voltage. The parameters gm_{in} (transconductance of the input stage), gm_r (transconductance of the regenerative part of the differential comparator) and C (total capacity on the output nodes) are extracted from HSPICE operating point simulations only once.

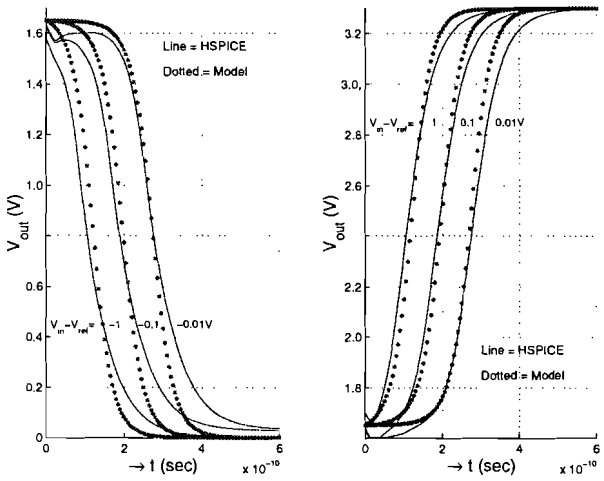


Figure 5: Transients of C++ model and HSPICE simulations agree well for the positive and negative flanks

The differential equation (10) is efficiently solved at runtime in the C++ program. The time domain results are compared with HSPICE simulations in Figure 5 and show a good agreement. The maximum differential input signal is set to $1V_{pp}$ (input range flash ADC). With this input voltage the comparator reacts fast, as shown in Figure 5. Further, it is seen in this figure that if the input signal is almost equal to the reference voltage, then the comparator reacts much slower and generates more errors in the output code of the ADC.

When simulating flash ADCs for frequencies $\ll f_s/2$, the effect modeled in (10) is marginal and can be replaced by an ideal comparator function. The filtering taken into account before the actual comparison is more important and has more influence on the output-code than (10). However, when applying fast varying analog input signals ($f \approx f_s/2$) to ADC resolutions of 4-bit and more, the effect of (10) is noticeable and can be important. Possible errors occuring in the comparator are proportional to the signal slope and the minimal differential input signal value and are well covered with (10).

The output latch function is modeled only as an extra delay with the same transient curve as the comparator. The output latch has two stable positions and is clocked.

V. ENCODER MODEL

The thermometer code to binary encoding of the outputs of the latches is modeled with standard functions [5] and are generalized to the n-bit case. The importance of these functions is that they model what's happening inside the encoder, taking into account the extra delay and error correction methods. The implemented model can handle different encoding schemes. Non-linearities are not taken into account in this part of the model.

VI. IMPLEMENTATION IN C++

A. Hierarchical build-up high-level flash ADC model

Figure 6 details the hierarchical implementation of the flash ADC in C++. The device and circuit parameters are derived from simple HSPICE operation point and AC simulations and saved in the input parameters list.

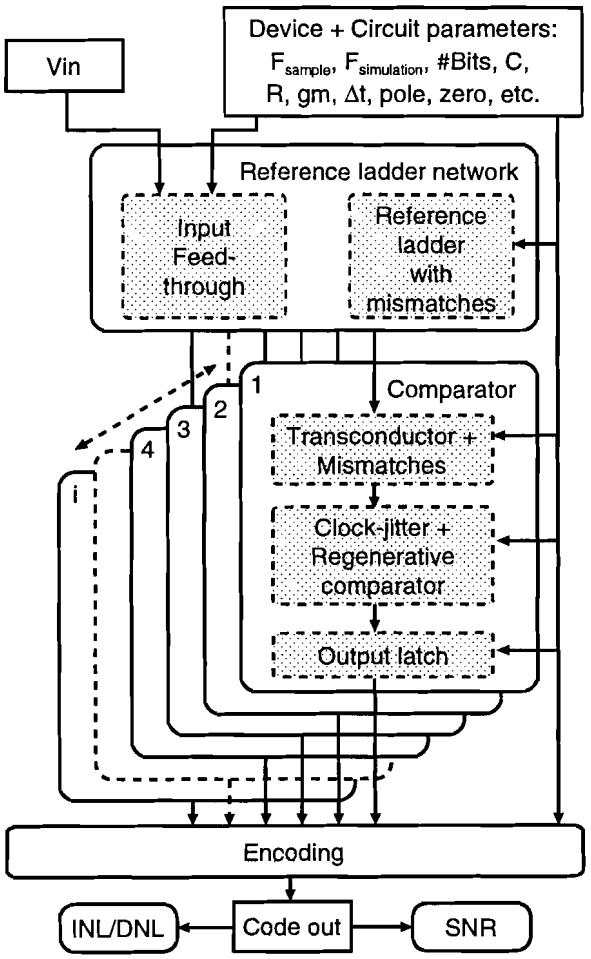


Figure 6: Flow diagram of flash ADC C++ model is hierarchical build-up

From the generated output-code, the ADC performance parameters INL/DNL and SNR are derived according [3]. The advantage of the hierarchical model is that it can be easily extended to other high-speed architectures.

B. Comparator mismatch implementation

Mismatch is the variation in physical quantities of identically designed transistors. This is caused by random variations occurring during processing. It is important to model mismatches in a comparator, because when the number of bits of an ADC increases, it will become a limiting factor.

In [7] two transistor model parameters (ΔV_t, $\Delta\beta$) are identified that describe the influence of mismatch in a transistor. The parameter $\Delta\beta$ is the current factor and is modeled by adding a current source between the drain and the source in the HSPICE netlist, and the threshold voltage ΔV_t is modeled by placing a voltage source before the gate.

The main effect of mismatches in a comparator is that it causes an offset voltage on the input. The method of [8] is used to write a program that determines the variance of the offset voltage of a comparator. This program runs several HSPICE simulations and calculates the effect of one mismatch on the offset voltage at the time. These mismatch simulations take several minutes, but have to be done only once for each type of comparator used. The calculated variance of the offset voltage is added to the input parameter list of the C++ model. In the C++ model every comparator in the ADC has a randomly generated offset voltage, with a Gaussian distribution according to the calculated variance. Also a worst case method is possible.

C. Clock-jitter implementation

A source of inaccuracy is jitter on the applied clock. This is an external error and therefore it is discussed separately.

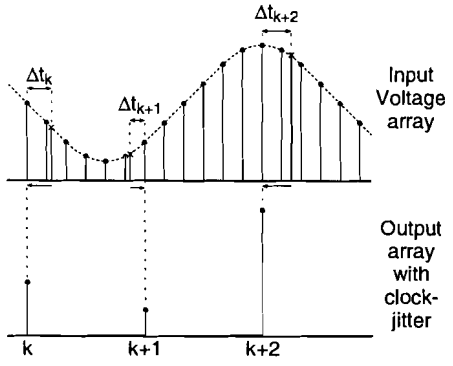


Figure 7: Efficient clock-jitter implementation in C++ model

The simulation method [9] used in the C++ program is block processing. An input array is loaded and processed and afterwards passed on to the next block. The clock-jitter is implemented in the model just before the regenerative comparator. The input array has a high oversampling rate to simulate accurately analog effects. From the input array a new array is constructed, with only samples on the sampling clock. This model works as follows: for every sample the exact sampling time is calculated from a random generator with a Gaussian distribution and adjustable variance, Δt, around the ideal clock instance (see Figure 7). The regenerative comparator uses this new sample array as input vector. Mostly, Δt is not a multiple of the simulation-timestep, so the input-array is interpolated to get a value for the output-array.

VII. EXPERIMENTAL RESULTS COMPARING C++ FLASH ADC MODEL WITH HSPICE SIMULATIONS

A. Accuracy

The accuracy of the 4-bit flash ADC model is compared with the HSPICE transient simulations. To verify the accuracy, several transient simulations in HSPICE are performed for different input frequencies. The C++ model should generate samples that accurately match the HSPICE simulation, which is used as a reference here. The C++ model uses exactly the same input signal array, with the same timestep (5ps) and length (1000ns) to make a valid comparison. With a sampling time of 2.5ns, 400 samples are generated. In Table 1 the in differences between the output codes of HSPICE and C++ are shown.

Table 1: Percentage of wrong output codes of our C++ model and an ideal quantizer

Input freq. [MHz]	HSPICE in comparison with:	
	Ideal quantizer	our C++ model
9.7	4%	0%
97	38%	2%

With an input frequency of 97 MHz, 38% of the the output codes of the ideal quantizer model are wrong compared to HSPICE. The C++ model tracks this HSPICE output code with only 2% errors. For lower input frequencies, there is no difference between HSPICE and our C++ model.

In Figure 8 a part of the generated HSPICE and C++ output codes are plotted for a 97MHz input signal with a sample frequency of 400MHz. Figure 8 shows that the ideal quantizer differs 17 of the 40 samples from the HSPICE simulations. Also shown is that the C++ model generates one error at the sample point of 90ns.

B. Simulation time

To compare the simulation CPU-time results of the C++ model with full transistor-level HSPICE transient simulations, the same time-step, sample frequency and simulation length are taken as above. All the simulations

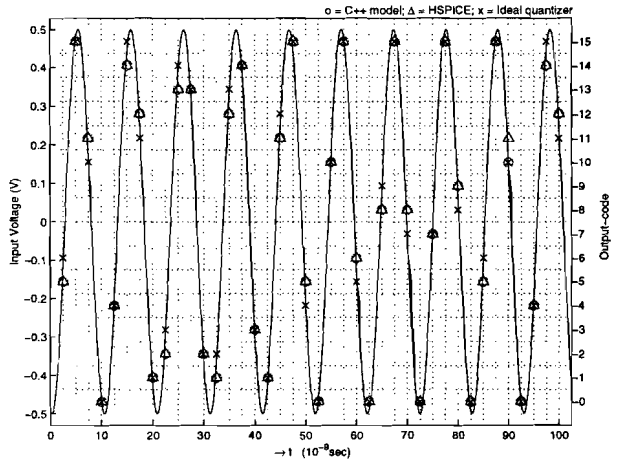


Figure 8: Transient 4-bit flash ADC C++ model compared with HSPICE

have been performed on a HP9000 design station with a HP-UX10 operating system.

Table 2: CPU-time of C++ model compared with HSPICE

# bits ADC	HSPICE sim. time [h:m:s]	C++ model sim. time [s]
3	5:53:50	3.66
4	52:40:18	13.15
5	n.a.	27.32

Table 2 gives an indication of the simulation speed improvement compared with HSPICE. Simulating the 5-bit case in HSPICE resulted in memory problems. The reported simulation times in C++ don't include the loading and saving of the input signal and output data. The C++ 4- and 5-bit ADC is simulated with the comparator model included as discussed in section IV. In the 3-bit ADC, the influence of this comparator model is negligible, and is replaced by an ideal comparator function.

VIII. CONCLUSIONS AND FUTURE WORK

In this work an efficient and accurate model for an n-bit flash ADC is developed. The model is built up hierarchically and can be easily extended to other high-speed ADC architectures due to its physical, technology-dependent model parameters. These parameters can be extracted from simple HSPICE simulations or an experienced designer can estimate them. This is the most time consuming step; nevertheless it has to be done only once. The total simulation time speedup factor is at least a factor 5000 compared to HSPICE. The model is in good agreement with simulations in HSPICE.

Future work will incorporate extension of the hierarchical model to other high-speed ADC architectures like folding/interpolating, pipe-lining etc.

Acknowledgments

This research was sponsored by the ESPRIT SALOMON project and the Flemish IWT FRONT-ENDS project.

REFERENCES

[1] ADS, Hewlett-Packard, http://www.tm.agilent.com/tmo/hpeesof/products/ads/

[2] Cierto NCSPW, Cadence, http://www.cadence.com/technology/funct/products/

[3] J. Doernberg, H. Lee, D. Hodges, "Full-Speed testin of A/D converters," *IEEE Journal of Solid-State Circuits*, vol. 19, no. 6, pp. 820–827, Dec. 1984.

[4] G. H. Golub, C. F. Van Loan, "*Matrix computations*," 2nd ed., Baltimore, 1990, The Johns Hopkins University Press.

[5] F. Kaes, R. Kanan, B. Hochet, M. Declerq, "New encoding scheme for high speed flash ADCs"*Proceedings of the IEEE Int. Symposium on Circuits and Systems*, IEEE, NY, USA, vol. 1, pp. 5–8, 1997.

[6] E. Liu and A. Sangiovanni-Vincentelli, "Verification of Nyquist Data Converters using behavioral simulation," *IEEE Trans. C.A.D of integrated circuits and systems*, vol. 14, no. 4, Apr. 1995.

[7] M. J. M. Pelgrom, "Matching properties of MOS transistors," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 5, pp. 1433–1439, Oct. 1989.

[8] G. Van der Plas, J. Vandenbussche, W. Verhaegen, G. Gielen, W. Sansen, "Statistical Behavioral modeling for A/D-converters," *Proc. IEEE Int. Conference on Electronics, Circuits and Systems*, pp. 1713–1716, Sep. 1999.

[9] P. Wambacq, G. Vandersteen, S. Donnay, M. Engels, I.'Bolsens, E. Lauwers, P. Vanassche and G. Gielen, "High-level simulation and power modeling of mixed-signal front-ends for digital telecommunications," *Proceedings of the IEEE Int. Conference on Electronics, Circuits and Systems*, pp. 420–430, Sep. 1999.

[10] G. M. Yin, F. Op't Eynde and W. Sansen, "A high-speed CMOS Comparator with 8-bit Resolution," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 2, pp. 208–211, Feb. 1992.

HIGH-LEVEL MODELING OF HIGH-SPEED FLASH A/D CONVERTERS

P.W.T. de Jong

Coaches (T.U. Eindhoven):
dr.ir. J.A. Hegt, prof.dr.ir. A.H.M. van Roermund, dr.ir. L.K.J. Vandamme;
Coaches (IMEC, Leuven):
dr.ir. S. Donnay, dr.ir. P. Wambacq, ir. J.P.R Compiet

ABSTRACT

A high-level model of a high-speed flash ADC is presented. The input parameter list is extracted from a 400MHz, 4-bit, flash ADC designed in HSPICE in a $0.35\mu m$ CMOS technology. A minimal speedup in simulation time of 6000 is reported. The accuracy of the model is verified with HSPICE simulations and shows a good agreement.

I. INTRODUCTION

The fast growing market of telecommunications demands faster and cheaper ways to design front-ends of transceivers for digital telecommunications. This requires a high level simulation environment, that can simulate complete end-to-end systems, including the analog front-end, mixed-signal and digital functionality. These tools must be able to explore and trade-off different architectures in a fast and accurate way. The Front-end Architecture Simulation Tool (FAST), that is currently being developed at the MIRA group at IMEC, is such a mixed-signal simulation tool.

The goal of this research is to make a high level model of an n-bit flash ADC, that matches as good as possible the transistor level circuit. The model has to include non-idealities, such as mismatches, non-linearities, input feed-through and clock-jitter. Only circuit and device parameters can be used, in order to give a designer a suitable set of design parameters. The model has to be efficiently implemented in C++. It must be easy to change the building blocks, number of bits and the architecture.

As reference case a 4 bit, 400MHz, flash ADC is available, which is designed in HSPICE in 0.35 μm digital CMOS technology. The simulation results of the C++ model must accurately match the HSPICE simulations. First, the architecture of the ADC in HSPICE and the high-level model in C++ are described in section II. In this section the ADC is divided in four blocks: reference ladder, comparator, clock-jitter and mismatches in the comparator. Which are treated in sections III to VI. Simulation results of the C++ model are compared with HSPICE simulations in section VII. Finally, conclusions are drawn in section VIII.

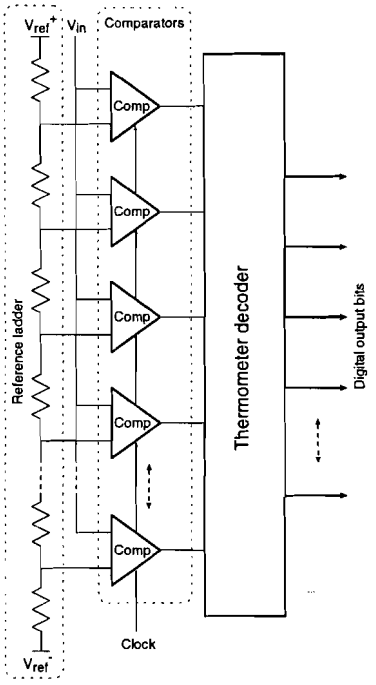


Figure 1: General architecture of a flash ADC.

II. THE ADC MODEL

A. The ADC in HSPICE

In Figure 1 is the general architecture plotted of a flash ADC. Three blocks can be recognized, a reference ladder, a comparator block and a thermometer decoder. For an n-bits ADC, the input signal is applied to $2^n - 1$ comparators. The comparators compare the input signal with $2^n - 1$ different reference voltages. These reference voltages are made by a reference ladder, that consist out of 2^n resistors.

The flash ADC can be extended easily for different numbers of bits. However, a complete simulation in HSPICE, including the encoding part, with more than 4 bits is very difficult, because of the long simulation time which results in memory problems. The final C++ model does not have this problem.

The comparators in a flash ADC produce an output pattern that is known as a thermometer code. Every time the input signal reaches a new reference level a '1' is added to the output code of the comparators. The thermometer

decoder converts the $2^n - 1$ output codes of the comparators into a normal binary code with n bits.

B. High-level model of the ADC in C++

The simulation method used by FAST for the ADC is called block-processing, see [1]. An input array is loaded and processed and afterwards passed on to the next block. In case of the ADC, an array with the analog input voltage (Vin) is loaded and converted to an array with the digital output code ($Code\_out$). The number of elements of the input array is $clkperiods \cdot clklength$ and the number of output codes is $clkperiods$.

The proposed high-level model in C++ is depicted in Figure 2.

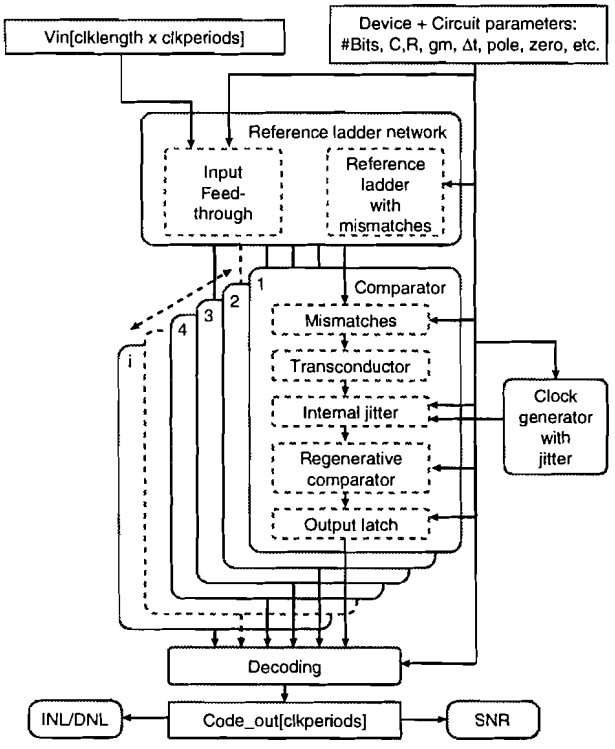


Figure 2: The block-diagram of the ADC in C++.

The input of the ADC is the input array Vin and a list of device and circuit parameters. The parameters are derived from simple HSPICE operation point and AC simulations. The device and circuit parameters are distributed over all the blocks in the model.

The reference ladder constructs $2^n - 1$ different reference voltages with $clklength \cdot clkperiods$ elements, that are passed on to the comparators. The reference ladder is composed with resistors with mismatches, that make $2^n - 1$ DC voltages. An unwanted effect in the reference ladder is the feed-trough from the input signal on the reference ladder. This is caused by the capacitive coupling of the the two inputs of a comparator, that couples the input of the ADC to the reference ladder. The variation of the reference voltages caused by the feed-through is modeled separately and added as extra with the DC voltages from the resistor ladder with mismatches, see section III.

The comparators are of the regenerative type and can be split in the transconductor, the regenerative comparator and the output latch. These functions are treated in section IV

The total clock-jitter is partially generated in the clock-generator and partially in the comparator itself. Therefore the jitter is divided, one jitter-model in the clock-generator and a jitter-model in every comparator. The clock-jitter is discussed in chapter V.

The main effect of mismatches in a comparator is a constant offset voltage at the input. The first block in the comparator corrects the input voltage by adding the DC offset voltage to the input voltages of the comparator. The effects of mismatches are treated in section VI .

The decoder converts the thermometer code from the comparators to a normal binary output code ($Code\_out$). The modeling of the decoder from [2] is digital and therefore straightforward, no further section is dedicated to the decoder.

From the generated output-code, the ADC performance parameters INL/DNL and SNR are derived according [3].

III. REFERENCE LADDER NETWORK

The reference ladder network consist out of two parts the input feed-through, see section A, and the reference ladder with mismatches, section B.

A. Input signal feed-through

The input feed-through is caused by the capacitive coupling of the two input nodes of the comparators, that couples the input of the ADC to the reference ladder, see Figure 1. The input feed-through is modeled by calculating the transfer-function from the input signal to a node on the reference ladder. The variation of the voltage in the reference ladder is afterwards subtracted from the input voltage of the comparator. The model in Figure 3a is used.

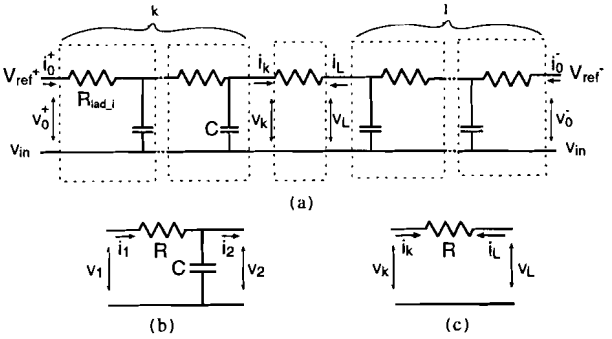


Figure 3: Input signal feed-through calculation model.

The capacitor C is the total capacity between the two inputs of a comparator. The value of C is the serial value of the gate source capacitances of the differential input transistor pair. When the reference ladder is connected between V_{ref}^{+} and V_{ref}^{-}, and the input voltage is v_{in}, then

$v_o^+ = V_{ref}^+ - v_{in}$ and $v_o^- = V_{ref}^- - v_{in}$. The model is subdivided in two types of blocks, see Figure 3b and 3c. By placing k sections of Figure 3b to the left and l sections to the right of the middle section, Figure 3c, it is possible to calculate v_l for any tap and any length of the ladder, according to the formula $k = 2^n - 1 - l$.

$$\begin{bmatrix} v_2 \\ i_2 \end{bmatrix} = W \cdot \begin{bmatrix} v_1 \\ i_1 \end{bmatrix} \quad (1)$$

with $W = \begin{bmatrix} 1 & -R \\ -j\omega C & 1 + j\omega RC \end{bmatrix}$

$$\begin{bmatrix} v_k \\ i_k \end{bmatrix} = M \cdot \begin{bmatrix} v_l \\ i_l \end{bmatrix} \quad (2)$$

with $M = \begin{bmatrix} 1 & -R \\ 0 & -1 \end{bmatrix}$.

Formulas (1) and (2) describe the transfer-matrix of Figure 3b and 3c. The relation between v_l, i_l and v_o^-, i_o^- is given by formula (3):

$$\begin{bmatrix} v_l \\ i_l \end{bmatrix} = W^l \cdot \begin{bmatrix} v_0^- \\ i_0^- \end{bmatrix} \quad (3)$$

Combining (3) with (2) according to Figure 3a results in:

$$W^l \cdot \begin{bmatrix} v_0^- \\ i_0^- \end{bmatrix} = M \cdot W^k \cdot \begin{bmatrix} v_0^+ \\ i_0^+ \end{bmatrix} \quad (4)$$

The formulas (3) and (4) form a system with four equations and four unknowns (v_l, i_l, i_o^+, i_o^-). When v_l is solved from this system, the order of the transfer-function is too high for efficient implementation in C++. Therefore an approximation is made:

$$W^k = [I + B]^k \approx I + k \cdot B + \frac{k \cdot (k-1) \cdot B \cdot B}{2} \quad (5)$$

with $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 0 & -R \\ -j\omega C & j\omega RC \end{bmatrix}$

Formula (5) describes a second-order approximation for W^k. This approximation is valid ([4]) if the absolute values of the eigenvalues are much smaller than 1, so at high frequencies the approximation is not valid any more ($f > 1\ GHz$). The approximation results in a fourth order transfer-function for $v_{ref,i}/v_{in}$, which is implemented in the model as an IIR-filter. The values of R and C are stored in the input parameter list.

B. Reference ladder with mismatches

The reference ladder network in the ADC model, see Figure 2, consists out of the input feed-through and the reference ladder with mismatches. The reference ladder with mismatches calculates the DC voltages on the nodes in the reference ladder. In the input parameter list is saved the nominal value of a resistor and the standard deviation of the resistors. At the beginning of a simulation 2^n resistor values are randomly generated on a Gaussian way with a mean of the nominal value and the adjusted standard deviation. Then the DC reference voltages are calculated with these resistor values.

The total reference voltages are these DC voltages in addition with the AC input feed-through signal.

IV. THE COMPARATOR

The comparator is based on [5]. A regenerative comparator consists out of three parts, the input stage, the actual regenerative comparator and the output latch.

The input stage is a continuous time amplifier which propagates the differential input voltage to the input of the regenerative part of the comparator. At the clock instant the regenerative comparator makes its outputs high or low dependent on the sign of the applied voltage difference on the input.

The output latch conditions the output of the regenerative comparator, in a way that it can be used by digital circuits.

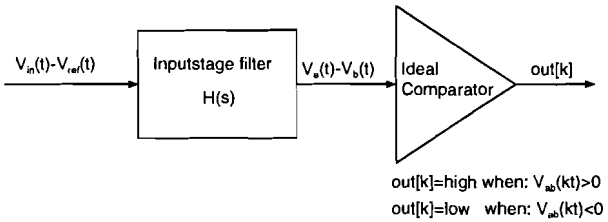


Figure 4: The comparator model with the input stage filter and an ideal comparator.

Figure 4 shows the model of the comparator. The input stage of a comparator is modeled by a transfer function. HSPICE calculated the transfer-function ($H(s)$) from the input voltage difference ($v_{in}(t) - v_{ref}(t)$) to the voltage difference between the regenerative nodes ($v_a(t) - v_b(t) = V_{ab}(t)$) of the comparator. This resulted in a low pass filter with the -3 dB point at approximately 1 GHz.

A third order filter is fitted on the HSPICE curve. This third order filter is implemented in C++. The curve is almost flat for frequencies beneath 200 MHz. The phase shift at 200 MHz is only 0.35 degrees. However simulation showed, that this small phase shift is very important. The poles and zeros are stored in the input parameter list of the comparator

Simulations showed that it is sufficient to model the difficult behavior of the regenerative comparator by an ideal comparator. Simulations showed also that the modeling of the input stage transfer function is much more important.

V. CLOCK-JITTER

There are two types of jitter in an ADC. One is the instability in the phase (phase noise) of the clock generator, the other is the internal jitter of the comparator. The

phase noise of the clock signal is caused by the oscillators and frequency dividers. The contribution of the clock generator to the total jitter is the same for all comparators. However the internal jitter of the comparators is different in every comparator, see Figure 2.

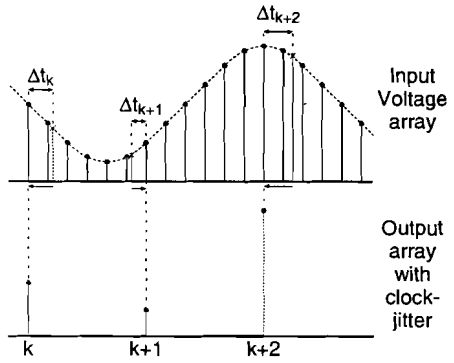


Figure 5: Clock-jitter model in the comparator.

The clock-jitter model in a comparator constructs from the input array with $clklength \cdot clkperiods$ elements an output array with $clkperiods$ elements. This output array contains the samples at the sampling time with jitter ($k \cdot clklength + \Delta t_k$). The jitter ($\Delta t_k$) is a summation of the clock generation jitter plus the internal jitter ($\Delta t_{int,i,k} + \Delta t_{CG,k} = \Delta t_k$). The values for $\Delta t_{int,i,k}$ and $\Delta t_{CG,k}$ are calculated separately by a random generator. The clock-jitter model is located between the input stage filter and the ideal comparator of Figure 4.

The ideal comparator uses the new sample array with $clkperiods$ elements to calculate an output array with $clkperiods$ output codes of zeros and ones. Mostly, Δt is not a multiple of the simulation-time-step, so the input array is interpolated to get a value for the output array.

It is difficult to give quantitative values to the jitter. A Gaussian distribution with an arbitrary variance is chosen to model the jitter. The variances of Δt_{int} and Δt_{CG} ares saved in the input parameter list of the C++ model.

VI. MISMATCH IN THE COMPARATOR

Mismatch is the variation in physical quantities of identically designed elements. This is caused by random variations occurring during processing. It is important to model mismatches in a comparator, because when the number of bits of an ADC increases, it will become a limiting factor. The following formulas from [6] are used to model the statistical effects of mismatch:

$$\sigma^2(V_T) = \frac{A_{V_T}^2}{WL}\,; \quad \frac{\sigma^2(\beta)}{\beta^2} = \frac{A_\beta^2}{WL} \tag{6}$$

These formulas describe the relation between two MOS-model parameters (V_T, β) and the variations in W and L by defining technology constants (A_β, A_{V_T}). The variations in the width (W) and the length (L) have a mean of zero and a standard deviation of σ. These variances describe the statistical behavior of the differences between two identically designed transistors.

The main effect of mismatches in the transistors of a comparator is that it causes an offset voltage at the input. The following method calculates the variance of the input referred offset voltage. The method is based on [7].

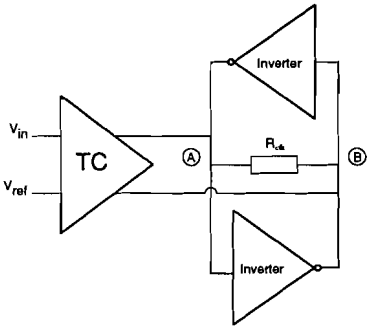


Figure 6: Model of the comparator used for the offset calculation.

The regenerative comparator is simplified to a model as shown in Figure 6. The clock transistor is closed and represented by a resistor R_{clk}. The nodes connected to the clock transistor are the regenerative nodes (A and B). Block TC represents all transistors, that are between the input nodes, v_{in}, v_{ref} and the regenerative nodes. TC is a linear amplifier, that amplifies the input voltage difference to a voltage difference between the regenerative nodes ($v_A - v_B = TC(v_{in} - v_{ref})$).

The offset is calculated in three steps. The first step is to determine the function of TC. Second the effect of each mismatch on the voltage difference between the regenerative nodes is calculated. Last, all these voltage differences are calculated back to the input of the comparator with the function TC and combined to the total offset of the comparator.

1. The function TC is determined by variating the input voltage difference ($v_{in} - v_{ref}$), while looking at the voltage difference $v_A - v_B$. This results in a linear relation between $v_{in} - v_{ref}$ and $v_A - v_B$ in a large enough region around zero.

2. Both inputs are now connected to a voltage source at the middle of the input range. Each transistor is replaced with the following sub-circuit in order to apply mismatches:

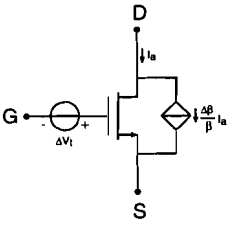


Figure 7: Each transistor is replaced by this sub-circuit for mismatch calculations.

The parameter $\Delta\beta$ is the current factor and is modeled by adding a current source between the drain and the source in the HSPICE netlist. The variations of the threshold voltage ΔV_T is modeled by placing a voltage source before the gate. Then the effect of every statistical variable (σ_{V_T},σ_β) in every transistor on the offset is calculated separately. One statistical variable, V_T or β, in one transistor is changed with the half of the mismatch, while all other statistical variables are left at their mean value (0). Equations (6) give a relation for the difference between two transistors. Thus, by giving every transistor half of the mismatch, it is not necessary to investigate which transistors form a pair. Then the voltage difference between the regenerative nodes is determined with an operation point simulation. This differential voltage is determined for every mismatch.

3. All the differential voltages on the regenerative nodes are calculated back to the input using the function TC. The formula (7) calculates the sensitivity of the input voltage for every mismatch in the comparator.

$$\frac{\delta V_{off}}{\delta i} = \frac{V_{off,i}}{\sigma_i} \tag{7}$$

When all the sensitivities are known, it is possible to calculate the total variance of the input referred offset voltage, by applying the following formula:

$$\sigma^2_{Voff} = \sum_{i=1}^{n} \left(\left(\frac{\delta V_{off}}{\delta V T_i} \sigma_{V_T,i} \right)^2 + \left(\frac{\delta V_{off}}{\delta \beta_i} \sigma_{\beta,i} \right)^2 \right) \tag{8}$$

Where n is the number of transistors. So there are $2 \cdot n$ separate offset voltage calculations necessary to determine the offset voltage variance. Formula (8) assumes, that the mismatches are uncorrelated. This assumption is probably not correct. However there is no information about correlation between mismatches whatsoever.

The variance of the offset voltage (0.0172 V) is saved in the input parameter list of the C++ model. Offset voltages for each comparator are generated randomly with this variance at the beginning of a simulation. The mismatch block in every comparator block of Figure 2 adds these DC offset voltages to the input voltage of the comparator.

VII. EXPERIMENTAL RESULTS COMPARING C++ FLASH ADC MODEL WITH HSPICE SIMULATIONS

A. Accuracy

The accuracy of the C++ model is compared with the HSPICE transient simulations for a 4-bit 400 MHz ADC. To verify the accuracy, several transient simulations in HSPICE are performed for different input frequencies. The C++ model should generate samples that accurately match the HSPICE simulation. The C++ model uses exactly the same input signal array, with the same time step (5 ps) and length (1000 ns) to make a valid comparison. With a sampling time of 2.5 ns, 400 output codes are generated. All mismatches and the clock-jitter are adjusted to zero. In Table 1 the differences between the output codes of HSPICE and C++ are listed for various input frequencies. The last row shows the results for the input signal $v_{in}(t) = A/2 \cdot \cos(2\pi\, 147 \cdot 10^6 t) + A/2 \cdot \cos(2\pi\, 33 \cdot 10^6 t)$.

Table 1: Percentage of wrong output codes of the C++ model and an ideal quantizer compared with HSPICE.

Input freq. [MHz]	HSPICE in comparison with:	
	Ideal quantizer	our C++ model
9.7	17/400	0
97	153/400	6/400
147+33	132/400	3/400

With an input frequency of 97 MHz, 153 of the the 400 output codes of the ideal quantizer are wrong compared to HSPICE. The C++ model tracks this HSPICE output code with only 6 errors. For lower input frequencies, there is no difference between HSPICE and the C++ model.

In Figure 8 a part of the generated HSPICE and C++ output codes are plotted for a 97MHz input signal with a sample frequency of 400MHz. The left y-axis is the analog

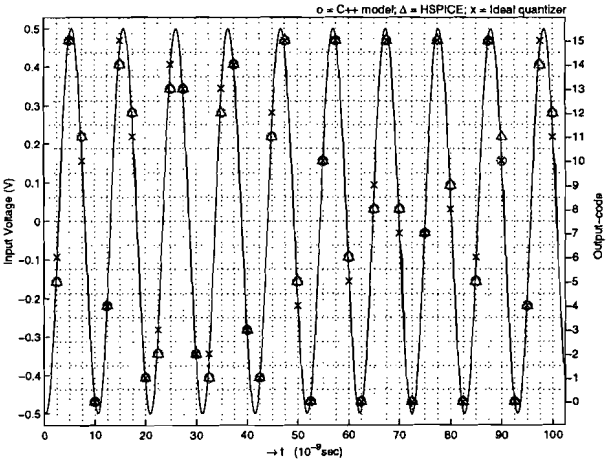


Figure 8: Transient 4-bit flash ADC C++ model compared with HSPICE.

input voltage, the sine curve belongs to this axis. The horizontal dotted lines are the quantization-levels of the ADC. The right axis displays the 16 digital output levels. The ADC samples the sine-wave every 2.5 ns, which results in output values indicated by a o, Δ or $\times$. Figure 8 shows that the ideal quantizer ($\times$) differs 17 of the 40 samples from the HSPICE simulations (Δ). Also shown is that the C++ model (o) differs one time from HSPICE at the sample point of 90ns.

The differences between the HSPICE and C++ results can be explained by the following error sources:

1. The transfer functions of the input feed-through model of chapter III differ a few dB's from HSPICE.

2. The switching behavior of the regenerative comparator is not taken into account.

3. The third order approximation of the input stage of the comparator transfer function is not accurate enough.

The 6 errors in case a 97 MHZ input signal is used are explained in the following way: 5 errors are caused by error source 2, and 1 error is caused by error source 1. The effect of error source 3 is marginal. No better results were obtained, when a fourth order filter for the input stage of the comparator transfer function is used.

B. Simulation time

To compare the simulation CPU-time results of the C++ model with full transistor-level HSPICE transient simulations, the same time-step, sample frequency and simulation length are taken as above. All the simulations have been performed on a HP9000 design station with a HP-UX10 operating system.

Table 2: CPU-time of C++ model compared with HSPICE.

# bits ADC	HSPICE sim. time [s]	C++ model sim. time [s]
3	21,000	3.35
4	190,000	7.48
5	n.a.	15.17

Table 2 gives an indication of the simulation speed improvement compared with HSPICE. Simulating the 5-bit case in HSPICE resulted in memory problems. The reported simulation times in C++ don't include the loading and saving of the input signal and output data. The most time consuming step in the C++ model is the filter function of the input feed-through and the transconductance. These functions take 0.49 seconds per comparator. The total C++ simulation time can be approximated by $0.49 \cdot (2^n - 1)$ $[sec]$. When the simulation time step is taken larger, the simulation time decreases however the number of errors increases.

VIII. CONCLUSIONS

In this work an efficient and accurate model for an n-bits ADC is developed The model meets with the assignment demands: it is flexible, efficiently implemented in C++ and based only on circuit and device parameters. These parameters can be extracted from simple HSPICE simulations.

The feed-through of the input signal on the reference ladder is caused by the capacitive coupling between the input-nodes of the comparator, that couples the input to the resistor-ladder. The input feed-through is dependent on the number of bits, the comparator circuit and the value of the resistor in the reference ladder. The presented model can deal with all different combinations of these parameters.

The main effect of mismatches between transistors in a comparator is an offset voltage at the input. A new method is described that can predict the variance of the offset voltage of a comparator in a faster and more accurate way.

All blocks in the clock-path, from the oscillator to the ADC included, contribute to the clock-jitter. A realistic way to model the clock-jitter is described.

The comparator is modeled by an input stage filter and an ideal comparator.

To verify the validity of the model, a comparison is made with a 4 bit ADC, which is available in HSPICE. The same input-signal is applied to HSPICE and the C++ model and the output codes are compared. The results show a good agreement between the simulated output codes of HSPICE and the model. The model is at least 6000 times faster than HSPICE.

REFERENCES

[1] P. Wambacq, G. Vandersteen, S. Donnay, M. Engels, I. Bolsens, E. Lauwers, P. Vanassche, and G. Gielen, "High-level simulation and power modeling of mixed-signal front-ends for digital telecommunications," *Proceedings of the IEEE Int. Conference on Electronics, Circuits and Systems*, pp. 420–430, Sep. 1999.

[2] F. Kaess, R. Kanan, B. Hochet, and M. Declercq, "New encoding scheme for high-speed flash ADC's," *Proceedings of 1997 IEEE International Symposium on Circuits and Systems.Circuits and Systems in the Information Age, ISCAS*, vol. 1, pp. 5–8, 1997.

[3] J. Doernberg, H. S. Lee, and D. A. Hodges, "Full-speed testing of A/D converters," *IEEE Journal of Solid-State Circuits*, vol. SC-19, pp. 820–827, Dec 1984.

[4] G. H. Golub and C. F. V. Loan, *Matrix computations*. The Johns Hopkins University Press, Baltimore, 1990.

[5] G. M. Yin and F. O. E. W. Sansen, "A high-speed CMOS comparator with 8-b resolution," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 208–11, Feb 1992.

[6] M. J. M. Pelgrom and A. C. J. D. A. P. G. Welbers, "Matching properties of MOS transistors," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 1433–9, Oct 1989.

[7] G. van der Plas, J. Vandenbussche, W. Verhaegen, G. Gielen, and W. Sansen, "Statistical behavioral modeling for A/D-converters," *The 6th IEEE international Conference on Electronics, Circuits and Systems*, pp. 1319–1323, 1999.