Moscow-Bavarian Joint Advanced Student School (MB-JASS) 2009:
Design Methods for Micro- and Nanoelectronic ICs and Systems

Probabilistic CMOS Technology

Sebastian Schießl

sebastian.schiessl@mytum.de

Technische Universität München

Contents

1 Introduction

Scaling CMOS devices into the nanometer regime creates several new challenges, such as parameter variations and noise susceptibility, which can lead to unreliable behavior of those devices. Especially when regarding thermal noise, the resulting behavior is probabilistic, what is undesired in many applications. If you take a calculator, for instance, 2+2 must always equal 4, it is not acceptable that the result would be 5 in some cases.

However, there are some applications that can tolerate probabilistic behavior at the device level or even benefit from it. In this article, we will focus on CMOS devices which show controlled probabilistic behavior and its effects on those applications.

2 Basic Principles

Our main consideration in this article is enhancing energy effiency and performance of applications by using devices based on probabilistic CMOS (PCMOS) technology. We will not look at devices rendered probabilistic or even completely faulty by factors like parameter variations or other pertubations, which may be caused by device scaling into the nanometer regime. We will only look at CMOS devices which show controlled probabilistic behavior due to controlled reduction of the supply voltage V_{dd} and the presence of thermal noise. Those devices are called PCMOS devices. In figure 1(a) you see a CMOS inverter which is rendered probabilistic due to thermal noise. The probabilities that this noise leads to wrong outputs are shown as gray-shaded regions in figure 1(b). Therefore, the probability of correctness p for this PCMOS switch can be expressed as:

$$p = 1 - \frac{1}{2} erfc\left(\frac{V_{dd}}{2\sqrt{2}\sigma}\right) \tag{1}$$

As shown in [ACKP06, p. 2, eq.(4)], the switching energy E for this switch, while $E = \frac{1}{2}CV_{dd}^2$, can be lower-bounded like this:

$$E(p, C, \sigma) > C\sigma^2 \left(\frac{4}{1.275}\right) \ln\left(\frac{0.28}{1-p}\right) \tag{2}$$

It can be directly seen from this equation, that the switching energy E grows quadratically with σ. As you see figure 1(b), this is quite natural, because V_{dd} must grow linearly with σ if p remains constant, and therefore E must grow quadratically with σ.

It is also shown in [ACKP06, pp. 2-3] that the switching energy E grows with the probability of correctness p, and that this growth is lower-bounded by an exponential in p.

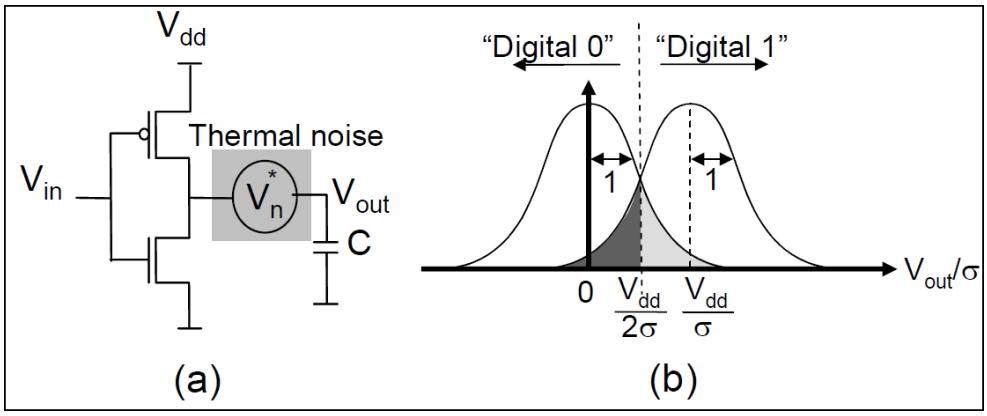

Figure 1: (a) PCMOS switch (b) Representation of digital values 0 and 1 and the probability of error for a PCMOS switch [ACKP06, p. 2]

3 Applications

3.1 Applications that harness probabilistic behavior

While probabilistic behavior is undesired in many applications, there are some algorithms which benefit from it. In this article, we will focus on applications based on four of those algorithms: *Bayesian Inference*, *Probabilistic Cellular Automata*, *Random Neural Networks* and *Hyperencryption*. They will be abbreviated as BN, PCA, RNN and HE. All of these algorithms use a so-called *core probabilistic step* (Table 1) with an associated probability parameter p.

This core probabilistic step will be implemented on a PCMOS coprocessor, while the determistic parts of the algorithms are executed on a host processor, in our example the StrongARM SA-1100 host. The coprocessor is connected to the host with memory mapped IO (Figure 2). This architecture will be called *probabilistic system on a chip* (PSOC) architecture.

We will compare this implementation of the probabilistic algorithms with two other implementations: The first competitor is a single host processor like the StrongARM SA-1100, which will execute both the deterministic and the probabilistic part of the algorithm, while the probabilistic part uses a pseudorandom number generator to create randomness (Figure 3(b)). The second competitor runs the deterministic part of the algorithm also on the normal StrongARM SA-1100 host, but uses a CMOS-based coprocessor for the probabilistic part, which again uses pseudorandom numbers for randomness. The coprocessor is connected to the host with memory mapped IO (Figure 3(c)). Figure 3(a) shows an implementation for a completely deterministic algorithm. There are some probabilistic algorithms which have a completely deterministic counterpart, for example the *celebrated probabilistic tests for primality* [ACKP06, p. 3] of course have deterministic counterparts, conventional deterministic algorithms for primality testing. But since not all the algorithms have such a deterministic counterpart, this issue will not be further investigated in

Algorithm	Application Scenarios	Implemented(s) Application(s)	Core Probabilistic Step
Bayesian Inference [MacKay 1992]	SPAM Filters, Cognitive applications, Battlefield Planning [Pfeffer 2000]	Windows printer trouble shooting, Hospital Patient Management [Beinlich et al. 1989]	Choose a value for a variable from a set of values based on its conditional probability
Random Neural Network [Gelenbe 1989]	Image and pattern classification, Optimization of NP-hard problems	Vertex cover of a graph	Neuron firing modeled as a Poisson process
Probabilistic Cellular Automata [Wolfram 2002]	Pattern classification	String classification [Fuks 2002]	Evaluating the probabilistic transition rule
Hyper-Encryption [Ding and Rabin 2002]	Security	Message encryption	Generation of a random string and encryption pad generation from this string

Table 1: Core probabilistic steps for the discussed algorithms [CKAP07, p. 9]

this article. [CKAP07, pp. 5-6]

Executing the core probabilistic step in a PCMOS device can basically be described as tossing a coin or throwing a dice, except that the results will only be 1 or 0 instead of heads or tails and 1 to 6. The chances for getting *1*s are specified with probability parameters p, according to the probability parameters of the core probabilistic steps of an algorithm. This is implemented with probabilistic switches with fixed input values (1 or 0). Their probability parameter p is controlled through the supply voltage V_{dd}, V_{dd} is lowered so that thermal noise will lead to random outputs (see chapter 2). For example, the probability parameter of 0.40 (for a logical 1) will be generated from a probabilistic inverter, with a 1 at the input and a probability of correctness of 0.60.

3.1.1 Metric to analyze gains

We will compare the different implementations regarding performance (running time) and the amount of energy consumed. For analyzing this, we introduce only one single metric: The *energy performance product* or EPP as the product of energy consumed and running time. According to [CKAP07, pp.7-8], this metric captures both metrics of interest, energy and time. Furthermore, if you would enhance performance by replication or voltage scaling,

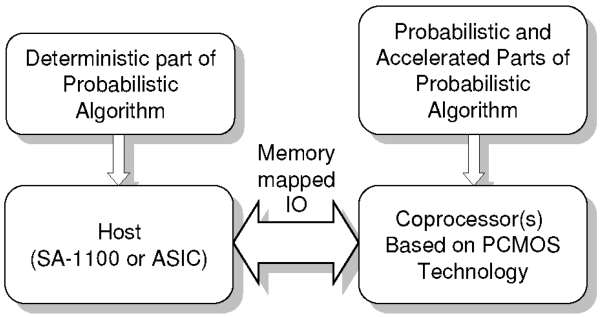

Figure 2: PSOC architecture [CKAP07, p. 6]

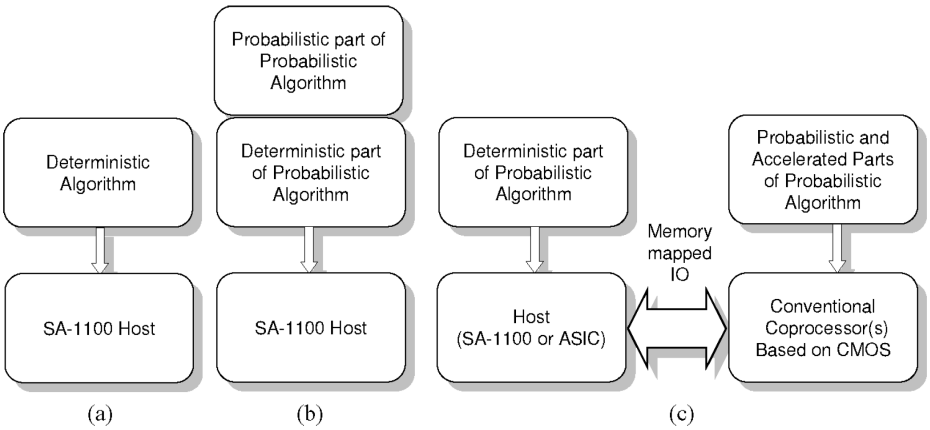

Figure 3: Conventional implementation alternatives [CKAP07, p. 6]

this would consume more energy accordingly. But the EPP would remain almost constant in this case, making it a reasonable metric to compare different implementations.

To analyze the benefits of our PCMOS implementation $\mathcal{I}$ compared the baseline implementation β, we use the *energy performance product gain* $\Gamma_{\mathcal{I}}$ as the ratio of the EPP of the baseline β to the EPP of the implementation $\mathcal{I}$:

$$\Gamma_{\mathcal{I}} = \frac{Energy_{\beta} \times Time_{\beta}}{Energy_{\mathcal{I}} \times Time_{\mathcal{I}}} \tag{3}$$

The implementation $\mathcal{I}$ will be based on PSOC architecture with a probabilistic CMOS (PCMOS) coprocessor, whereas the baseline β will be the single host processor (Fig. 3(b)) or a host processor connected to a conventional CMOS based coprocessor (Fig. 3(c)).

3.1.2 Analyzing gains

The EPP gains vary broadly from application to application. When the baseline implementation β is a software which runs completely on the StrongArm SA-1100 host (as shown in figure 3(b)), the EPP gains for PSOC architectures based on PCMOS devices

are shown in table 2. This means that for random neural networks (RNN), the PSOC implementation $\mathcal{I}$ with PCMOS-based coprocessor can run up to 300 times faster than the baseline implementation β, while energy consumption remains the same. Although the

Algorithm	$\Gamma_{\mathcal{I}}$	
	Min	Max
BN	3	7.43
RNN	226.5	300
PCA	61	82
HE	1.12	1.12

Table 2: EPP gains [CKAP07, p. 10]

PCMOS technology saves a little energy on the device level (due to lowered voltage levels), the main reason for saving time and energy when using a PSOC architecture is found on the application level, because complex and expensive pseudorandom number generation becomes unnecessary. The various factors which influence the EPP gains are described in detail in [CKAP07, Chapter 4.3]. When the StrongARM SA-1100 host is replaced for both the baseline implementation and the PSOC implementation by a more efficient host built with custom ASIC logic, these gains increase even more, to 9.38 for *hyperencryption* and 561 for *probabilistic cellular automata*, what indicates that PSOC architectures will become even more efficient when host processors get better. [CKAP07, Chapter 4.4]

3.1.3 Reducing the number of different voltage levels

As mentioned before, different probability parameters are usually archieved by different voltage levels. In the *probabilistic cellular automata* algorithm, there are probabilistic transition rules for each cell, and the probability of a cell transition depends on its current state and the state of its two nearest neighbours [CKAP07, pp. 20-21], which results in $2^3 = 8$ different transition rules. So in this application there are typically 8 distinct probability parameters, which will result in up to 8 different voltage levels. However, the number of different voltage levels should be small, because each voltage level has to be generated and distributed on the chip, which increases complexity and power consumption.

How can the number of different voltage levels be reduced? In some cases, application quality might not be reduced if a probability parameter is eliminated and replaced by another one to reduce the number of voltage levels. In other cases, logical operations can be used to archieve the desired probability parameter. For example, if your PCMOS device uses two different voltage levels which create probability parameters of 0.40 and 0.50, the probability parameter 0.20 can easily be created by connecting the former two with a logical AND. Table 3 shows a way to generate even more probability parameters with logical networks, where only 0.50 and 0.40 are generated directly. Note that you

can always get the complementary probabilities $1 - p$ (higher than 50%) by just adding a deterministic logical NOT at the end. By the way, 50% correctness is the "worst" you can get. 0% correctness means the output is always 1 when it should be 0 and vice versa. This is nothing but a normal, deterministic inverter and cannot be made from probabilistic elements.

Application-Required Probability Parameters	Composition Tree
0.05	[[(0.4)AND(0.5)]AND(0.5)]AND(0.5)
0.10	[(0.4)AND(0.5)]AND(0.5)
0.15	[(0.5)AND[NOT(0.4)]]AND(0.5)
0.20	(0.4)AND(0.5)
0.25	(0.5)AND(0.5)
0.30	(0.5)AND[NOT(0.4)]
0.35	[NOT[(0.5)AND[NOT(0.4)]]AND0.5
0.40	0.40
0.45	[NOT[[(0.4)AND(0.5)]AND(0.4)]]
0.50	0.50

Table 3: Reduction of number of probability parameters through logical operations [CKAP07, p. 25]

3.1.4 Quality of Randomness

Especially in encryption algorithms the quality of random or pseudorandom bits is crucial. The bits must be statistically uncorrelated. This means there must be no kind of "pattern" within the sequence of bits, or else the strength of encryption will suffer severely. If the randomness of PCMOS-based devices originates only from the randomness of thermal noise, and if that thermal noise is really a perfect white (uncorrelated) noise, the randomness should also be perfect. But if, for example, some physical effect would cause some kind of oscillation in the PCMOS device, then output bits would be correlated and not completely random. But the statistical properties of PCMOS devices were tested using the NIST suite with very favorable results, although further research is necessary. [CKAP07, Chapter 6.4]

3.2 Applications that can tolerate probabilistic behavior

Apart from applications that benefit from probabilistic behavior on the device level, there are also applications which are deterministic, but can tolerate probabilistic behavior. Many applications in digital signal processing fall into this category. Some of them naturally have to trade off between energy consumed and quality in the form of the *signal-to-noise* ratio SNR. As an example, let's view on an FIR filter in the context of the H.264 decoding algorithm. In an effort to reduce power consumption, the supply voltage of an FIR filter could be lowered, which results in a lower probability of correctness p. Figure 4(b) shows what happens if the supply voltage is lowered uniformly for each bit. It can be estimated that the supply voltage was lowered to 70%, so the power consumption of this element

should have halved, but it results in a major degradation of picture quality. However, if you reduce the supply voltage non-uniformly (Fig. 4(c), so that the most significant bits will operate more or less correctly, the picture quality degrades only slightly, while still saving a significant amount of energy. However, note that in Figure 4(c) you can see quite a lot of different voltage levels, what should be avoided. See chapter 3.1.3 for more information. [ACKP06, p. 4]

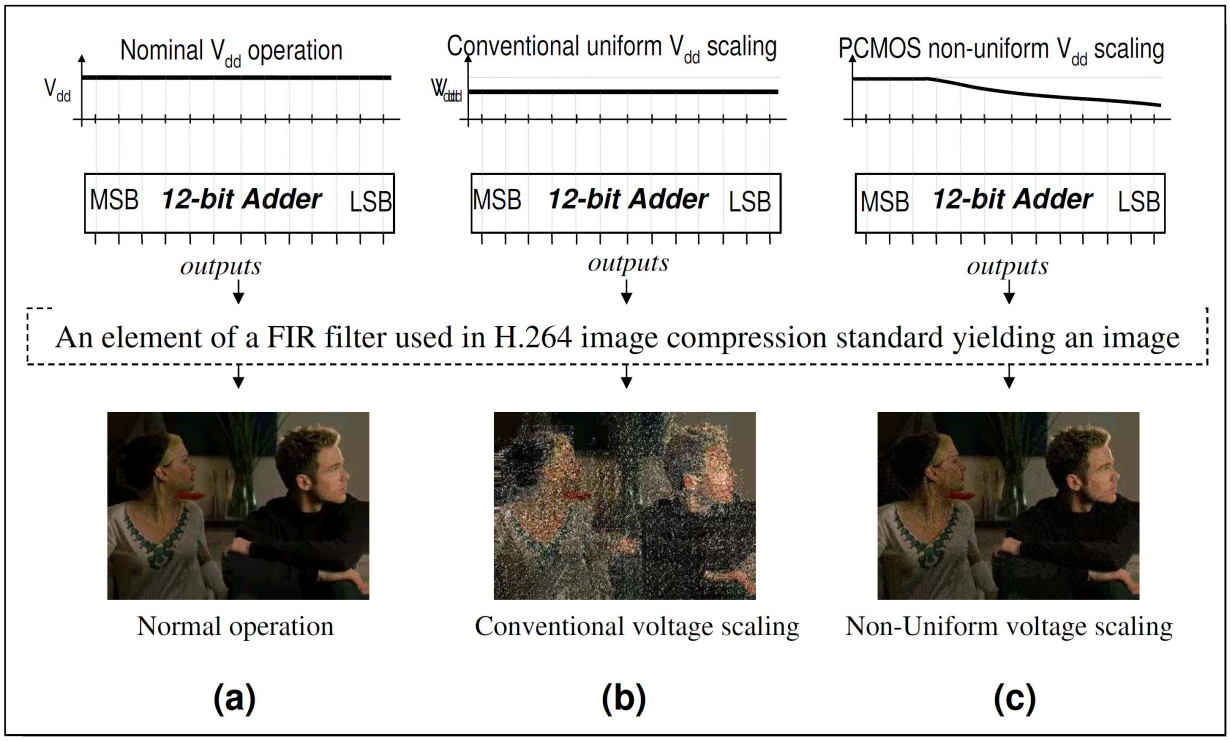

Figure 4: Image processing with (a) normal operation, (b) uniform voltage scaling and (c) non-uniform voltage scaling [ACKP06, p. 5]

3.3 Applications that can not tolerate probabilistic behavior

For applications that can not tolerate probabilistic behavior, but have to be realized on unreliable, probabilistic devices, reliable computation must be accomplished with unreliable components. This can be done with redundant systems of those unreliable components, combined with reliable arbitrators. But there are also other approaches like speculative execution on unreliable logic and verification by reliable logic elements to save time and/or energy. One example for this: a digital division, which usually needs much calculating time, could be calculated on unreliable logic, which could save a lot of time or energy, and the result would be verified by a fast multiplication on the conventional, reliable processor.

4 Conclusion

The probabilistic behavior of PCMOS devices described in this paper does not result from problems in manufacturing chips, but is well-directed and controlled through the supply voltage. There are some applications which benefit greatly from this specific probabilistic behavior, resulting in huge gains of performance and energy effiency. Other applications can tolerate that probabilistic behavior and trade off the resulting reduction of quality with the energy saved through lowering the supply voltage. This is a first and already very valueable step in the shift of design paradigm from deterministic to probabilistic architectures. This shift in design paradigm will most likely be forced by the ongoing downscaling of CMOS devices into the nanometer regime, because probabilistic behavior at the device level will become inevitable.

References

[ACKP06] B. E. S. Akgul, L. N. Chakrapani, P. Korkmaz, and P. V. Palem. Probabilistic cmos technology, a survey and future directions. *In the Proceedings of the IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*, 2006.

[CKAP07] L. N. Chakrapani, P. Korkmaz, B. E. S. Akgul, and K. V. Palem. Probabilistic system-on-a-chip architectures. *ACM Transactions on Design Automation of Electronic Systems, Vol. 12, No.3, Article 29*, August 2007.