Ultra Efficient Embedded SOC Architectures based on *Probabilistic* CMOS (PCMOS)

(Extended Abstract)

Krishna V. Palem, Lakshmi N. Chakrapani, Bilge E. S. Akgul, Pinar Korkmaz
Suresh Cheemalavagu and Balasubramanian Seshasayee

Abstract

Probabilistic CMOS *has been featured as a significant technological innovation and has attracted attention as a viable means of realizing ultra low-energy computing (for example, please see Port [21] and Cheemalavagu, Korkmaz, Palem, Akgul and Chakrapani [4]).* PCMOS *based semiconductor devices are inherently probabilistic in that they are guaranteed to compute correctly with a probability p, a design parameter, and by design, they are expected to compute incorrectly with a probability* $(1 - p)$*. In this paper, we demonstrate for the first time that* PCMOS *not only yields energy savings at the device level, but also yields significant savings, simultaneously to the energy consumed, as well as improvements to the performance (measured in terms of the execution time) in the context of probabilistic applications drawn from the embedded computing domain. These benefits are derived using a novel algorithm-technology co-design methodology for* PCMOS*, that we introduce in this paper. This methodology is a novel contribution of this work, and is crucially dependent on the "probabilistic content" of an application, quantified as "flux" in this paper. All of our application-level savings are quantified using the product of the energy consumed (measured in Joules) and the performance (measured in number of clock cycles) denoted energy* $\times$ *performance: the* PCMOS *based savings range from a substantial multiplicative factor of over* 1000 *when compared to a competing custom* ASIC *realization, in the context of probabilistic cellular automata [8]. Additional algorithms for which we demonstrate orders of magnitude improvements to the (energy* $\times$ *performance) metric through* PCMOS *include algorithms that implement bayesian inference [24, 12], random neural networks [9] and hyper-encryption [5]; these algorithms are collectively used in the context of applications such as face and speech recognition, security and a range of optimization problems.*

1. Introduction

As CMOS technology scales down into the nanometer region the increasing number of challenges posed by noise and other perturbations (see Sano [15, 26], Kish [11] and Shepard [28] for example) offer unique "opportunities" for low-energy embedded system design. The surprising premise that noise can be harnessed as a resource rather than be viewed as an impediment, was shown by Palem [16, 17, 18] using foundational principles and models. In this work, we

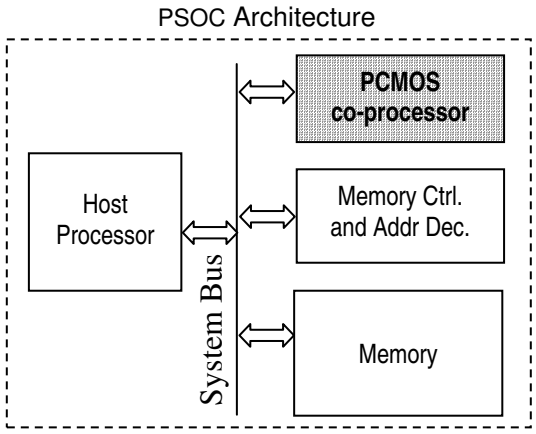


Figure 1. A canonical PSOC architecture.

demonstrate for the first time that these principles, when interpreted in the domain of CMOS computing platforms can yield significant improvements simultaneously to the *energy consumed* as well as to the *running time*—collectively characterized as the energy-performance product (EPP)—of an application. A singular innovation through which these savings are accomplished is the particular form of CMOS that is affected by ambient (thermal) noise—we refer to it as *probabilistic* CMOS or PCMOS. The development of a methodology for using PCMOS to realize ultra efficient embedded computing platforms in the energy-performance sense, and the demonstration of the value of this novel technology in the context of a range of embedded applications of interest, are the two significant contributions reported in this paper.

To this end, we first develop a methodology (akin to hardware-software co-design) that we refer to as *algorithm-technology* co-design. Our methodology is aimed at realizing an extremely efficient, if not optimal, *probabilistic system-on-a-chip* (PSOC) architecture using PCMOS devices. As shown in Figure 1, a canonical PSOC architecture will consist of a host processor used to compute most of the control-intensive *deterministic* components of an application, whereas the co-processor realized using PCMOS, will be used as an energy-performance (EPP) accelerator—a novel concept by itself, since gains in energy through conventional techniques such as voltage scaling are typically derived *at the expense* of performance, and vice-versa. The reasons for emphasizing the development of this co-design methodology as a significant contribution is based on the following two novel observations. First, the probabilistic content of the algorithm becomes a novel resource to be managed and treated, much as space requirements, flexibility and IP-reuse are treated in the traditional co-design context. Furthermore, as we will see in the sequel, considerations of design efficiency differ significantly in the context of PCMOS when compared to those arising in the context of conventional CMOS since, in the former case, the probability p (with which the design is to function correctly) is a parameter! Thus, in contrast to conventional co-design, the probability p of correct behavior of the system and the probabilistic content of the application are both design parameters.

Given a probabilistic application, the host processor and the PCMOS (co-processor) technology parameters, our framework allows the algorithm designer to analyze alternate ways of partitioning it between the CMOS and the PCMOS components of the PSOC. In the end, the *partitioned* implementation of the application's probabilistic component will be executed on the co-processor of the PSOC; throughout this paper, the co-processors are *application specific*.

The degree to which benefits can be derived through PCMOS technology in the context of an application is dependent on its "probabilistic content". For example, if an application has high probabilistic content, then there will be more frequent activations of PCMOS-based components

	SA-1100	CMOS	PCMOS
Energy (pJ)	2,002,874	25,920	12.8
Delay (μs)	34.4	9.6	1
EPP	68,898,865	248,832	12.8
Gain to EPP	1X	277X	5,382,724X

Table 1. Energy and time needed to produce a 32-bit random number, the EPP, and the EPP gains for the three implementation alternatives.

and less dependence on conventional CMOS components. Formally this notion of probabilistic content is defined to be the *probabilistic flux* $\mathcal{F}$ (referred to simply as flux for convenience) in this paper. An application that embodies a greater amount of flux affords greater opportunity for gleaning benefits from PCMOS technology.

In order to evaluate the true benefits of using PCMOS technology as a means for significant EPP acceleration, we will consider three alternate implementations of the probabilistic applications wherein

1. the probabilistic content of the application is realized entirely using software on the host processor,
2. the co-processor is realized using conventional CMOS technology, and
3. the co-processor (Figure 1) is realized using PCMOS technology.

In the first two cases, the probabilistic content of the application is emulated using pseudo-random execution, as opposed to using a naturally occurring "random source" such as thermal noise in the PCMOS context (see Cheemalavagu, Korkmaz, Palem, Akgul and Chakrapani [3], [4], and Stein [31], for details). Whenever it is appropriate, these alternate ways of realizing a probabilistic application will be compared against the deterministic case, wherein a deterministic algorithm will be used to realize the same application entirely on the host processor and thus, there is no probabilistic component at all.

As an early glimpse of the potential PCMOS benefits, let us consider a probabilistic operation, $\mathcal{O}$, that consumes one 32-bit random number in some application. In Table 1, we compare three alternative implementations in terms of the energy consumed and the performance (delay) it takes to produce one 32-bit random number to be consumed by this operation $\mathcal{O}$. As seen from the table, $\mathcal{O}$ costs 2,002,874 pJ and $34.4\mu s$ when the StrongARM SA-1100 processor (case (1)) is used (where the 32-bit random number is implemented in software using the Park-Miller algorithm [19] from the GNU scientific library), 25,920 pJ and $9.6\mu s$ when a CMOS implementation (of a pseudo-random number generator or PRNG [19] which is case (2)) is used, and a remarkably small 12.8 pJ and $1\mu s$ when PCMOS is used (case (3)). To reiterate, in the first case, a co-processor is not used whereas in the second and the third cases the co-processor is realized using CMOS (see Park and Miller [19]) and PCMOS (see Cheemalavagu, Korkmaz, Palem, Akgul and Chakrapani [4]) respectively.

More generally, let *Gain* ($\mathcal{O}$, T, T') be the ratio of the EPP when the operation $\mathcal{O}$ is realized using technology T, to the EPP when realized using technology T'. For example, from Table 1, *Gain*($\mathcal{O}$, SA-1100, CMOS) is a factor of 277 whereas *Gain*($\mathcal{O}$, SA-1100, PCMOS) is a staggering factor of 5,382,724. Stated informally, the operation of producing 32 independent probabilistic bits is a multiplicative factor of 277 more efficient if a dedicated pseudo-random

number generator (realized in $0.25\mu m$ CMOS technology) is used whereas it is $5,382,724$ times more efficient if the co-processor is realized using PCMOS—in both cases, the baseline relative to which these improvements are determined is a StrongARM SA-1100 processor computing the random bits in software in isolation without a co-processor. The details of computing gain of alternate technology realizations across various applications can be found in Section 4.

A systematic study and evaluation of gains gleaned through using PCMOS technology will be the topic of Section 4. In this section where the *second* contribution of this work will be described, we will detail application specific PSOC implementations and demonstrate order(s)-of-magnitude savings using PCMOS technology. In this context, the probabilistic algorithms that we consider include those that implement *random neural networks* (RNN) [9], *bayesian networks* (BN) [12, 24], *probabilistic cellular automata* (PCA) [8] and *hyper-encryption* (HE) [5]. These algorithms are used in wide-ranging embedded applications domains—BN for windows printer troubleshooting and hospital patient management [1], RNN for image texture analysis and interactive and probabilistic image retrieval, PCA in the context of remote sensors [27], in a range of classification algorithms and pattern matching and HE in a wide range of security applications.

A significant consideration in our work is to establish that a PCMOS-based co-processor is much more efficient than a competing co-processor made from CMOS. Thus, while it is easy to show the substantial benefits of using a PCMOS based co-processor (case 3) over the case where all the computation is performed in software using a StrongARM SA-1100 processor (case 1), it is indeed more challenging to establish significant gains when a co-processor based on CMOS is available as a competing choice (case 2). While a detailed discussion of this point is beyond the scope of this introduction—details can be found in Section 4—we observe that as the host processor's energy (or running time) increase significantly relative to that of the co-processor, alternate realizations of the co-processor such as variations from CMOS to PCMOS cannot be differentiated through the EPP metric. Therefore, one important factor that we study is the host efficiency, which we characterize through the *host energy ratio* metric. The host energy ratio is the ratio of the energy consumed by the host through the execution of an application to the energy consumed due to one activation of a PCMOS based co-processor in a PSOC. The lower the host energy ratio, the higher the application level EPP gains of PCMOS over CMOS. Otherwise, if the host energy ratio is high, which corresponds to the case where the host is not efficient, then the benefits of a PCMOS co-processor over a CMOS co-processor will be relatively low (and hidden) at the application level, as in both cases (PCMOS and CMOS) the dominant consumer of energy and time would be the host. For example, the EPP gain for a SOC including a StrongARM host processor and a CMOS co-processor for the hyper-encryption application is 1.12 and not different from the corresponding EPP gain value for a PSOC including the same host processor, StrongARM, and a PCMOS co-processor, which is also 1.12. To establish this dependence of the EPP on the energy and performance values of the host processor, we also instantiated the host processor as a custom designed ASIC. Thus with such a host processor which is more efficient than the StrongARM processor, the corresponding EPP values for CMOS based SOC and PCMOS-based PSOC are significantly different, the EPP value of PCMOS based implementation being 9.48 times better than the EPP value of CMOS based implementation, significantly favoring PCMOS.

With these concepts as an introductory backdrop, we will consider two independent dimensions: namely (1) the application's flux and (2) the host energy ratio. Together with the application-algorithm dependent parameters, these two attributes will yield a particular value of the application level benefit quantified through the EPP metric. These analytically computed gains can then be verified through actual implementations of the corresponding designs.

In application domains employing probabilistic algorithms, independent probabilistic bits are needed in copious quantities. For example, in the novel and breakthrough technique referred to as *hyper-encryption*, by Aumann, Ding and Rabin [5] deliberately increase the need for randomness relative to established methods for encryption such as RSA (see Rivest, Shamir and Adelman [25]) with the intention of significantly increasing the degree of (cryptographic) security in a model proposed by Maurer [13]. This trend is very favorable to our design approach since it increases the flux, as evidenced from the very positive gains in EPP that we can derive for hyper-encryption through PCMOS. Nevertheless, to quote Brian Hayes from the *American Scientist* [10], "The more randomness you consume, the better it has to be." Thus, a secondary yet crucial metric that we consider is the *quality of probabilistic implementation*. Specifically, we show in Section 4.3 that while yielding significant gains to the EPP, PCMOS technology also yields significantly better quality random bits, verified by applying the tests provided by the National Institute of Standards and Technology (NIST) [23].

The rest of the paper is organized as follows. In Section 2, we review a probabilistic switch that we use as a building block to realize a PSOC. In Section 3, we list our applications and their characteristics. In Section 4.1, we introduce the vocabulary for describing the metrics that would be used to compare implementations, and in Section 4.2, we identify the application level benefits for our application suite. We introduce the gain achieved from PCMOS in terms of the quality of randomness in Section 4.3. Next, we present our experimental methodology in Section 5. In Section 6, we present our algorithm-technology co-design framework driven by several new metrics used to characterize the efficiency of a PSOC design at the application level, the alternate PSOC design choices, and the associated application level benefits. Concluding remarks and directions for future research are presented in Section 7.

2. PCMOS Technology

It was established earlier that ultra low-energy probabilistic computing switches can be realized by using noise as a resource (please see Cheemalavagu, Korkmaz, Palem, Akgul and Chakrapani [3, 4]). That study presents the analytical modeling and HSpice simulation based characterizations of a PCMOS inverter (switch), used ubiquitously as a building-block in architecture design. In this paper, we use these PCMOS switches as a building block to demonstrate their benefits to applications in the context of a typical PSOC architecture.

3. The Suite of Applications

Probabilistic algorithms, in computer science parlance, "toss coins" (or consume bits from a source of randomness). Examples of such algorithms include the celebrated test for *primality*, referred to as the Rabin-Strassen-Solovay algorithm [22] [30], used as a key building block in RSA public-key cryptosystems. To demonstrate the utility and efficacy of a PSOC we consider applications that employ probabilistic algorithms that implement *bayesian networks* (BN) [12], *random neural networks* (RNN) [9], *probabilistic cellular automata* (PCA) [8] and *hyper-encryption* (HE) [5]. Probabilistic content and utility in a wide range of application scenarios are the common characteristics of these algorithms. The algorithms as well as the domains in which they find use are summarized in Table 2.

For concreteness, let us consider the PCA algorithm. In this algorithm, the probabilistic function involves the transition of an automaton which decides the next state, based on the current state and a probability p associated with the transition function. For each of the candidate algorithms, a similar core probabilistic function can be identified and implemented in PCMOS.

The remaining part of the application that is deterministic, is executed on the host processor. The algorithms, applications based on these algorithms and the core probabilistic function within each of the applications of interest are summarized in Table 2.

Algorithm	Application Scenarios	Implemented Application(s)	Core Probabilistic function
BN [12]	SPAM Filters, Cognitive applications, Battlefield Planning [20]	Windows printer trouble shooting, Hospital Patient Management [1]	Choose a value for a variable from a set of values based on its conditional probability
RNN [9]	Image and pattern classification, Optimization of NP-hard problems	Vertex cover of a graph	Neuron firing modeled as a Poisson process
PCA [33]	Pattern classification	String classification [8]	Evaluating the probabilistic transition rule
HE [5]	Security	Message encryption	Generation of a random string and encryption pad generation from this string

Table 2. Summary of probabilistic algorithms, their corresponding applications and core probabilistic content.

Bayesian Networks (BN) **:** A Bayesian network is a directed acyclic graph G of nodes V representing variables and edges E representing dependence relations among the variables. The variable represented by a node u can take a value from a finite set of values Σ_u. Each value σ in the set Σ_u has a conditional probability $P(\sigma/\sigma')$ associated with it, where σ' is the set of values of the variables represented by the *parents* of u. Variables whose values are known apriori are referred to as *evidences*, based on which other variables are *inferred*. For comparison, we consider a deterministic junction tree algorithm [20] and a probabilistic likelihood weighting algorithm [20].

PCMOS **Methodology :** Since the conditional probabilities are known apriori for a Bayesian network G, a PCMOS component is designed for each node in the graph. The PCMOS component corresponding to a node u is implemented as a table whose row is indexed by the set of values of the parents of u. Each column of the table corresponds to a value σ from the set Σ_u, and each element identified by a *(row,column)* pair corresponds to the conditional probability $P(column/row)$, of inferring the corresponding value σ; it is implemented by a PCMOS switch specialized to the probability value $P(column/row)$. Our algorithmic implementations involve additional innovations to take advantage of the special attributes of PCMOS (over CMOS) and the properties of a bayesian network such as using the same PCMOS switch for two table elements whose associated conditional probabilities have the same value.

Random Neural Networks (RNN) **:** A random neural network [9] is an undirected graph G of nodes V and edges E. Each node has an associated potential ψ which is incremented (decremented) by incoming (outgoing) actions referred to as "firings". Node firings occur with a rate that is modeled as a Poisson process with rate ψ. When a node fires, the polarity and destination node of the firing are determined by probability parameters p_i and p_d respectively. Through a suitable combination of network topology, probability parameters and firing rates, intractable optimization problems such as finding the minimum vertex-cover of a graph can be "solved" [9] heuristically.

PCMOS **Methodology :** To approximate the Poisson process determining the firing of a node with associated potential ψ, we use the Bernoulli approximation of a Poisson process [6]. The

PCMOS co-processor consists of a bank of PCMOS switches each modeling a Bernoulli trial, generating a 1 with probability p and 0 with probability $(1-p)$.

Probabilistic Cellular Automata (PCA) **:** Cellular automata are simple regular structures with local (typically nearest neighbor) communication. Each automaton is associated with a *state* and a simple *transition rule*, which specifies state transitions based on its current state and the states of its neighbors. In a *probabilistic* cellular automaton, this transition rule also admits a probability parameter p as its input.

PCMOS **Methodology :** A set of PCMOS switches determine the transition probabilities. Each PCMOS switch is used to implement a transition rule. The control-intensive part of choosing a transition rule to apply, based on the state of the neighbors, is implemented using the host processor.

Hyper-Encryption (HE) **:** This breakthrough is a provably secure encryption technique for everlasting security [5]. This scheme consists of generating an *encryption pad* based on a *publicly* available random string α and a shared secret key k, known only to the sender and the receiver. Message encryption is performed by a bitwise XOR operation of the encryption pad with the message.

PCMOS **Methodology :** In the PSOC architecture, the random string is generated entirely using PCMOS. In a variant custom implementation, the entire algorithm is implemented so that the functionality of the deterministic host processor is realized using a full-custom hardware design.

4. Application Level Benefits of PCMOS

4.1. Metrics for Quantifying the Application Level Benefits

In order to characterize, and quantify PCMOS benefits at the application level, we will now define a variety of derived metrics. Subsequently we will summarize the application level benefits using these metrics.

1. **Metric - Energy Performance Product:** EPP described earlier, is defined as the product of the application level energy and the execution time, and will be used as the chief metric of interest to evaluate various implementations.

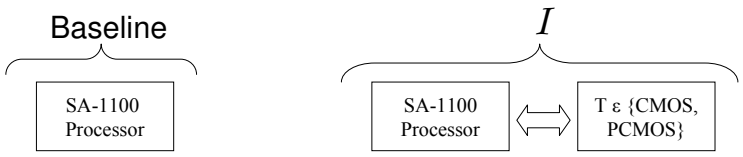
   Given the EPP of two alternate realizations they can be compared by computing the ratio of their individual EPP values. For example, the case wherein in one realization, the entire algorithm is implemented as software executing on the host and is referred to as the *baseline*, and a second realization wherein the deterministic part of the algorithm executing on the host with the probabilistic part executing on a CMOS co-processor.

2. **Energy Performance Product Gain:** Γ is the ratio of the EPP of the baseline to the EPP of a particular implementation $\mathcal{I}$ (e.g., a PSOC or an SOC) using a technology T (e.g., PCMOS or CMOS). This ratio is calculated as follows:

   $$\Gamma_T = \frac{Energy_{Baseline} \times Time_{Baseline}}{Energy_{\mathcal{I}} \times Time_{\mathcal{I}}} \tag{1}$$

   For calculating Γ, *in the sequel, the baseline always corresponds to the case when the entire computation is done on the host processor and therefore, there is no co-processor.* Such an example in the context of the RNN application is depicted in Figure 2, where the baseline is the StrongARM SA-1100 computing the deterministic as well as the probabilistic content

and $\mathcal{I}$ is the combination of the StrongARM SA-1100 as the host computing the deterministic content and the co-processor computing the probabilistic content implemented with technology T. The co-processor is realized using technology $T \in \{$ CMOS, PCMOS $\}$. Note that, in the figure, the term $Time_{SA-1100} + Time_T$ is replaced with $Time_{SA-1100}$ in the denominator because of the fact that the time spent on the co-processor ($Time_T$) can be overlapped completely with the execution of the host.



$$\Gamma_T = \frac{E_{SA-1100} \times Time_{SA-1100}}{(E_{SA-1100} + E_T) \times (Time_{SA-1100} + Time_T)}$$

$$\Gamma_T = \frac{E_{SA-1100} \times Time_{SA-1100}}{(E_{SA-1100} + E_T) \times (Time_{SA-1100})}$$

Figure 2. Calculation of Γ for RNN application, where the baseline is the SA-1100 and $\mathcal{I}$ is the combination of SA-1100 and a co-processor.

Now, to compare pairs of alternate choices for realizing the co-processor, for example, CMOS and PCMOS, we introduce the notion of a "relative energy performance product" as follows.

3. **Metric - Relative Energy Performance Product with technology parameters T, T':** REPP of an application is defined as the ratio of its EPP when it is implemented on a (conventional) SOC with a co-processor realized using technology T to the EPP when it is implemented on a PSOC with a co-processor realized using technology T'. The host processor is invariant in both cases. Given the EPP gains Γ_T and $\Gamma_{T'}$ respectively, $\text{REPP}_{T,T'}$ is calculated as follows.

$$\text{REPP}_{T,T'} = \frac{\Gamma_{T'}}{\Gamma_T} \tag{2}$$

For example, in the context of the RNN application depicted in Figure 3, the individual gain values for $\Gamma_{T \equiv CMOS}$ (when co-processor is CMOS) and $\Gamma_{T' \equiv PCMOS}$ (when co-processor is PCMOS) are used to compute the $\text{REPP}_{T,T'}$.

4. **Metric - Quality of Probabilistic Implementation:** is defined empirically based on the statistical tests from the NIST suite [23] and is detailed in Section 4.3.

4.2. Application Level Gains of PCMOS

The gains outlined in the introduction (see Table 1), is a measure of the savings that could be achieved per probabilistic operation in an application. We refer to these savings achieved

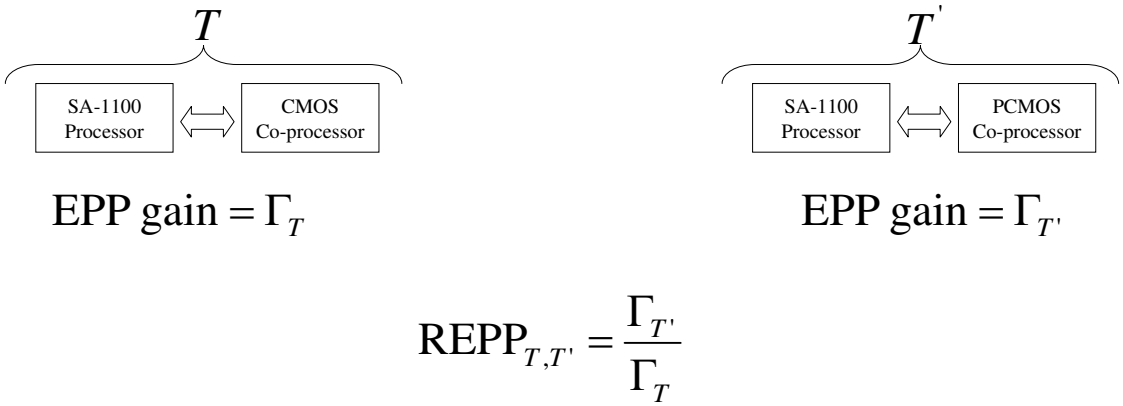


Figure 3. Calculation of $\text{REPP}_{T,T'}$.

Application	gain over SA-1100	gain over CMOS
BN	9.99×10^7	2.71×10^6
RNN	1.25×10^6	2.32×10^4
PCA	1.56×10^5	2.9×10^3
HE	1.56×10^5	2.03×10^3

Table 3. The EPP gain of PCMOS over SA-1100 and over CMOS for each primitive core probabilistic operation of all the four applications.

per probabilistic operation (or function) as the *raw technology gain* (please see Section 6.1 for definition of raw technology gain). We summarize the raw technology gains of PCMOS over StrongARM SA-1100 and over CMOS for each of the applications in Table 3. Each row of this table corresponds to one of the four distinct applications of interest to us and presents the gains achieved per probabilistic function. As can be readily seen from Table 3, these gains are substantially more for PCMOS—orders of magnitude greater—than those for CMOS. These per-operation gains would of course be valuable at the level of an entire application, only if the application embodies significant opportunity, characterized by its flux $\mathcal{F}$ and its normalized flux $\mathcal{NF}$ (please see Section 6.1 for definitions of flux).

Algorithm	Flux (as percentage of total operations)	EPP Gain $=\Gamma$	
		Min	Max
BN	0.25%-0.75%	12.5	291
RNN	16.4%-19.7%	226.5	300
PCA	4.19%-5.29%	83	110
HE	12.5%	9.48	9.48

Table 4. Application level max and min EPP gains of PCMOS over the baseline implementation with increasing flux, where the baseline for HE is an ASIC implementation of the host processor, with the StrongARM SA-1100 serving as the baseline in the remaining three cases.

We will now describe the gain Γ_T where the technology T is PCMOS in Table 4. As shown in

the table, these gains at the scope of an entire application range from a factor of (about) one order of magnitude for the HE application, to a factor of about 300 in the context of the RNN application. A range of EPP gains are observed whenever multiple data points are available, for example, in the context of the bayesian network where different data points correspond to different networks, the flux varies from 0.25 % to 0.75 %, the corresponding gain increases from a factor of 12.5 to an impressive factor of 291 due to increase in flux. Similar increases are observed for the other applications as well, caused by an increase in the flux values as shown in the table.

4.3. The Value of PCMOS to Quality of Randomness

Whereas EPP gains for applications have been our foremost metric of concern to demonstrate the utility of PCMOS the *quality* of the implementation of a probabilistic algorithm is a characteristic of interest as well. Many probabilistic algorithms have been found to be sensitive to the quality of random bits. For example, Monte Carlo simulations have been shown to yield incorrect results when poor quality pseudo-random number generators are used as a source of random bits [7]. In addition, the strength of encryption schemes like Hyper-Encryption can be severely compromised if random sequences can be "guessed." For such applications whose correctness, and hence utility depend on the quality of random bits, it is important to compare the quality of randomization afforded by PCMOS and conventional CMOS based implementations.

In Figure 4, we show the statistical tests (from the NIST suite [23]) applied to compare and evaluate the quality of random bit sequences generated by a PCMOS inverter and those generated by a CMOS random number generator implementing the Park-Miller algorithm [19]. The random sequences in the case of PCMOS inverter have been produced from the actual chip measurements of a $0.25\mu m$ TSMC prototype and those of CMOS from HSpice simulations of a $0.25\mu m$ TSMC based hardware design. In both cases, p is considered to be 0.5.

Among these tests and to highlight a few, the *frequency* tests evaluate the frequency of 0s and 1s—whether the fraction of 1s to 0s is close to 0.5. The *runs* test determine contiguous sequence of 1s in a block. The *rank* test checks the linear dependence, while the *FFT* and *approximate entropy* tests detect periodicity and frequency of overlapping patterns. The *template* matching tests detect repetitions of non-periodic patterns and the *universal statistical* test as well as the *lempel-ziv* test detect whether the random sequence can be compressed. The tests are run on random bit sequences of length 20,000,000. In evaluating the test results, we use the same testing strategy and criteria as recommended by the NIST suite. Specifically, the test results shown in parenthesis in the table are compared against a threshold (which is 0.93) used to determine whether the sequence passes or fails a test. The tests are run on random bit sequences of length 20,000,000. The result indicates the proportion of subsequences (tested through iterations) that pass from the random sequence being tested. As seen from the figure, PCMOS passes 11 tests out of 14 whereas CMOS passes only 7 of these tests.

5. Energy and Performance Modeling and Experimental Methodology

The performance and energy consumption of an algorithm is modeled in four contexts illustrated in Figure 5. These contexts are as follows: (a) The *best possible* deterministic algorithm solving the same problem, implemented completely in software and executing on the host processor (in our case a StrongARM SA-1100 processor), (b) The probabilistic counterpart executing completely on the host processor, with pseudo random bits generated by a software implemen-

Test	PCMOS	CMOS
Frequency	PASS (0.98)	FAIL (0.84)
Block-frequency	PASS (1.00)	PASS (0.98)
Cumulative sum	PASS (0.98)	FAIL (0.86)
Runs	PASS (0.98)	PASS (0.96)
FFT	PASS (1.00)	PASS (1.00)
Approximate entropy	PASS (0.98)	FAIL (0.92)
Long-run	PASS (1.00)	PASS (1.00)
Rank	PASS (1.00)	FAIL (0.00)
Non-overlapping template	PASS (0.9375)	PASS (0.9375)
Overlapping template	FAIL (0.8889)	FAIL (0.00)
Lempel-Ziv	FAIL (0.8125)	FAIL (0.0625)
Linear complexity	PASS (1.00)	PASS (1.00)
Universal Statistical	FAIL (0.725)	FAIL (0.8889)
Serial	PASS (1.00)	PASS (1.00)

(result > 0.93) → PASS
(result < 0.93) → FAIL

Figure 4. Comparison of quality of randomization for PRNG and PCMOS.

tation of a well known algorithm for generating pseudo-random bits [19] (c) The probabilistic algorithm executing on the host processor with a conventional CMOS co-processor or (d) with a PCMOS co-processor. Collectively these four cases (Figure 5) encompass all reasonable alternate implementations of the application using COTS (commercial-of-the-self) CMOS, custom CMOS and PCMOS.

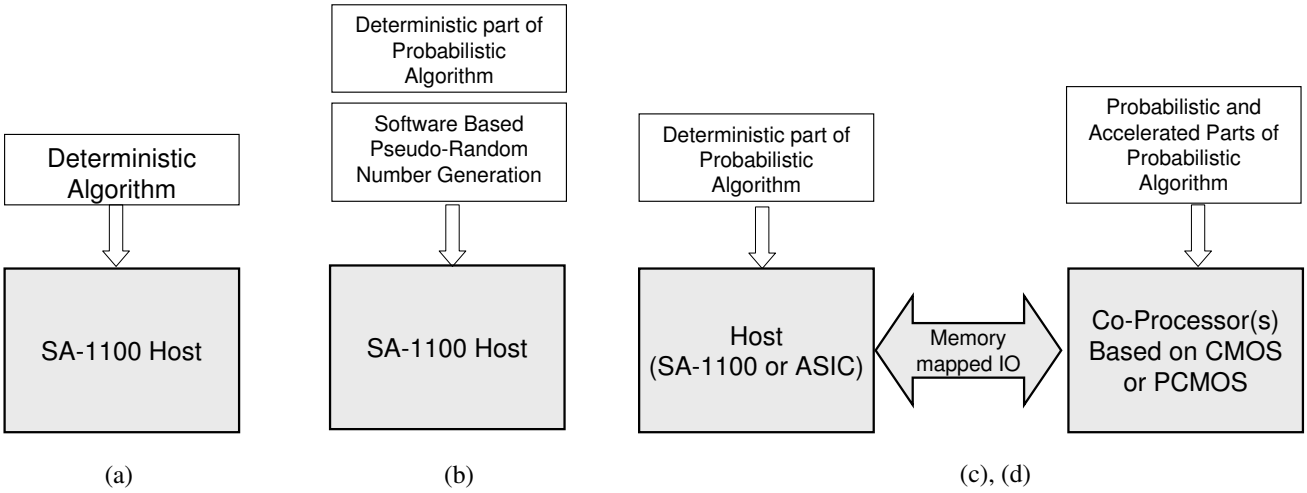


Figure 5. The Host + Co-processor style implementations that are compared: purely deterministic algorithm, probabilistic algorithm using software based pseudo-random number generation, using a CMOS co-processor and using a PCMOS co-processor.

Performance Estimation: To estimate the performance of these PSOC and SOC architectures, the IMPACT simulator of the Trimaran infrastructure [32] has been modified to measure the number of cycles taken by an application executing on a StrongARM SA-1100 core. The simulator records a trace of the activity of the PCMOS and CMOS co-processors. This information combined with the performance models of the co-processors, typically obtained through HSpice simulations or physical chip measurements yields the PSOC (SOC) performance in terms of execution time.

Energy Estimation: The energy consumption of an application executing on a PSOC (SOC) architecture is the sum of the energy consumed by the host, the energy consumed by the PCMOS (CMOS) co-processor(s), and the energy cost of communication between the host and the co-

processor(s). Since the co-processors are memory mapped, communication is through load-store instructions executed on the core. To quantify the energy consumed by the SA-1100 core, the JouleTrack model introduced by Sinha and Chandrakasan [29] is used. This model is reported to be within 3% of the energy measured on an actual SA-1100 core. The power modeling techniques applied to various components of the PSOC (SOC) architecture are illustrated in Figure 6.

Since the design of the co-processors are application-specific, the energy consumed by the co-processor is different for each of the applications. The CMOS based co-processor involves a 32-bit pseudo random number generator (PRNG) [19] that is designed and synthesized into TSMC $0.25\mu m$ process and its energy cost is derived from HSpice simulations. In the context of extensions based on PCMOS, the energy cost of the co-processor is derived from *physical chip measurements* of functioning probabilistic switches realized in TSMC $0.25\mu m$ process.

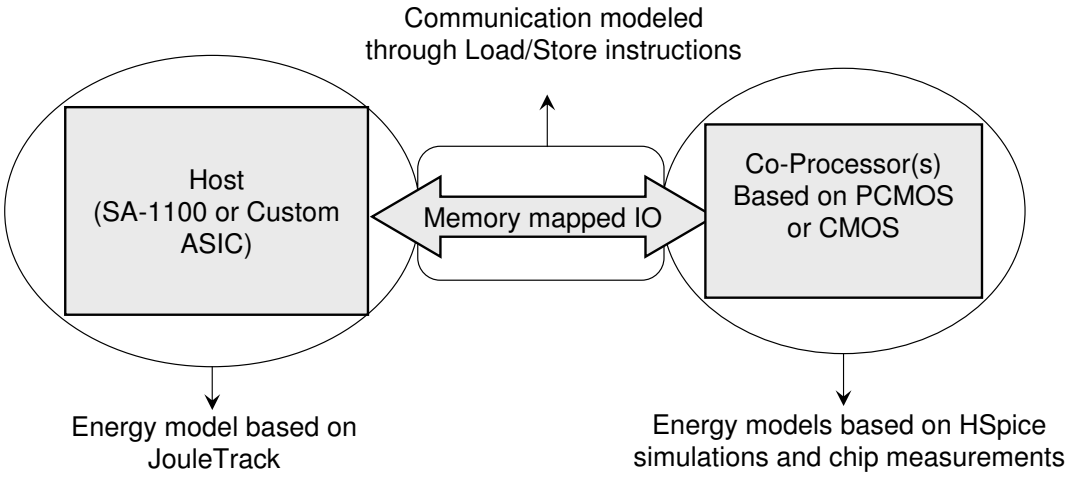


Figure 6. The Host+Co-processor Architecture and Power Modeling.

A cursory analysis of absolute and relative gains to EPP values might indicate that these gains are solely determined by the raw gain (Table 1 in Section 1). However, deeper analysis quickly reveals that the relative gain $\text{REPP}_{T,T'}$ is very dependent on the efficiency of the host processor. An analysis of this dependence and its impact will be the topic of Section 6. To facilitate this comparison, we examine a second style of PSOC implementation: that of a custom implementation where the host is an ASIC as shown in Figure 6. The energy and performance is obtained through HSpice simulations in the context of the hyper-encryption custom logic that is designed in TSMC $0.25\mu m$ process.

In all of the styles of PSOC implementations, to account for the probabilistic nature of the applications of interest, several "runs" are averaged for a given set of inputs to the application.

6. The PSOC Co-design Framework

We will now identify the attributes of the probabilistic algorithms and its implementation in PCMOS and CMOS which have an impact on the quality of a PSOC architecture for applications of interest. In each case, we will identify the benefits and then propose a quantitative metric to characterize the value of this benefit.

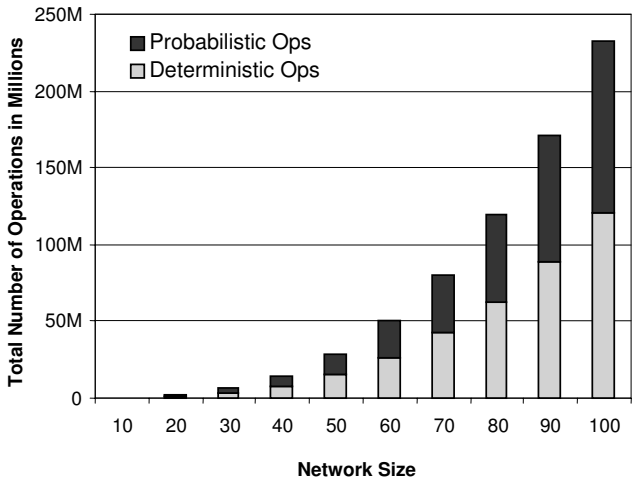
6.1. The Independent Attributes that Influence Gain

1. PCMOS **Characteristic**: PCMOS is particularly efficient in computing with ultra-low energy. For example, the energy consumed for generating one random bit by PCMOS is 0.4 pico Joules. By contrast, the Park-miller algorithm [19] implemented in custom hardware consumes about 2025 *times* that energy.

   (a) ***Raw Technology Gain***: $Gain(\mathcal{O}, T, T')$ is defined as the ratio of the $energy \times performance$ of realizing a probabilistic core primitive (within an application) in technology T to that of realizing it in T'. Table 3 shows the raw technology gain of PCMOS over SA-1100 and CMOS (for each application), wherein T' is PCMOS and T is SA-1100 and CMOS, respectively.

2. **Application Characteristic**: Given this benefit, it is natural to expect that higher amounts of "probabilistic content" in the algorithm will yield greater opportunities gaining from the use of PCMOS. Thus the amount of "probabilistic content" will be a good figure of merit, which we refer to as flux $\mathcal{F}$.

   (a) ***Probabilistic Flux*** : $\mathcal{F}$ is defined as the *total number of primitive probabilistic operations* $\mathcal{O}$ of the algorithm. For example, in Figure 7 we show the deterministic as well as probabilistic content of the RNN application and the corresponding flux values for different network sizes.



Network Size	Flux	Normalized Flux
10	64952	0.925501831
20	737031	0.943837889
30	2574157	0.949407735
40	6547047	0.953054812
50	13162084	0.954861134
60	23203457	0.956295214
70	37548426	0.957184059
80	56632194	0.957938529
90	81604921	0.958515242
100	112351178	0.959146208

Figure 7. Flux $\mathcal{F}$ and normalized flux $\mathcal{NF}$ for the RNN application.

   (b) ***Normalized Probabilistic Flux*** : $\mathcal{NF}$ is defined as the ratio of the energy $\times$ performance corresponding to the probabilistic operations alone, to the energy $\times$ performance value of the entire application. The Table in Figure 7 shows the normalized flux values for different network sizes of the RNN application. As seen from the figure, while the probabilistic content as well as the deterministic content increase, this increase is almost in equal proportion and hence the increase in $\mathcal{NF}$ is very small.

3. PSOC **and** SOC **Characteristic**: In addition to the characteristics mentioned above, PSOC and SOC may involve communication costs between the host processor and the co-processors. For example, in a host + co-processor architecture where the co-processors

are modeled as memory mapped devices, the communication cost is the cost of load-store instructions executing on the host. On the other hand, as mentioned in Section 1, one important characteristic is the efficiency of the host processor when compared to the co-processor(s) used in a PSOC or a SOC; the more efficient the host, the higher the (observed) benefits of PCMOS over CMOS in the overall.

(a) **Metric - *Communication Cost***: The energy cost due to communication is captured through the communication parameter $\mathcal{C}$.

(b) **Metric - *Host Energy Ratio***: The ratio of the application-level energy consumed by the host to the energy of one activation of the PCMOS co-processor.

6.2. Analytical Characterization of Application Level Gains

In this section, we focus on the comparison of two alternate choices for realizing the co-processor—specifically, PCMOS and CMOS—using the concept of REPP and the associated metric (defined in Section 4.1), which captures the gains resulting from the choice of PCMOS as the co-processor over a CMOS co-processor. In both cases, and to reiterate, the host processor is a custom ASIC implementation. In the interests of staying within the mandated space limits, we will restrict our analysis, to the HE application. Recall that our focus is on illustrating the (less obvious) impact of the efficiency of the host processor on the relative gain, quantified through the REPP metric. To this end, for this application, two distinct implementation are shown in Figures 8 and 9. The former case corresponds to the situation wherein the host processor is the StrongARM SA-1100, whereas the latter corresponds to the case where the host processor is an ASIC. *In both cases, the co-processor is* PCMOS *based*. The main goal of illustrating these two cases is to show the significant improvement in the value of REPP in going from the case of StrongARM (Figure 8) to the more efficient custom ASIC host (Figure 9), implying that PCMOS is much more efficient than CMOS. In both cases, REPP is shown as a function of the host energy ratio. As can be seen from figures, the high values of the host energy ratio corresponds to the case when the host processor is a StrongARM (Figure 8) and it corresponds to the case when the host is a custom ASIC with low values (Figure 9).

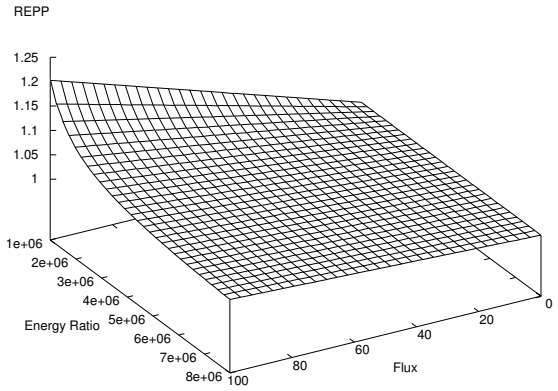


Figure 8. Relationship between flux and REPP with very high value of host energy ratio (e.g., when host is the StrongARM processor).

Specifically we wish to delineate the impact of the efficiency of the host processor. Starting with Figure 5 wherein the host processor is a StrongARM we note that the range of EPP values

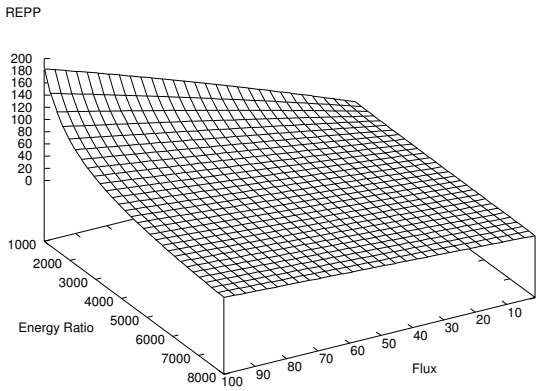


Figure 9. Relationship between flux and REPP with a low value of host energy ratio (e.g., when host is a custom ASIC).

of this COTS processor for the core operation in HE of producing a string of probabilistic bits is over six orders of magnitude less efficient than the corresponding PCMOS realization (see "energy ratio" axis in Figure 8). In this rather inefficient regime, $\text{REPP}_{T,T'}$ where $T \equiv$ CMOS and $T' \equiv$ PCMOS is nearly 1. For example, a point A in the Figure 8 refers to the host being the StrongARM processor, wherein the host energy ratio is 7000000 and flux is 32 for the HE application. As seen from the figure the $\text{REPP}_{T,T'} = 1.06$, which indicates that the EPP of CMOS co-processor is about the same as that of the PCMOS co-processor.

By contrast and moving away from StrongARM to a host processor realized from custom ASIC logic, the corresponding host energy ratio values are significantly better—the host is now (only) about three orders of magnitude less efficient than PCMOS (see "energy ratio" axis in Figure 9). This change in the host's efficiency causes the corresponding REPP values to rise to higher possible values as illustrated in the figure. For example, a point B in the Figure 9 refers to the host being a custom ASIC implementation, wherein the host energy ratio is 7600 and flux is 32 for the HE application. As seen from the figure, the $\text{REPP}_{T,T'}$ is now much higher; the value of REPP improves significantly to a factor of 9.38 with a custom ASIC host.

These gains at the application level are based on the flux, the efficiency of the host and of course the per-operation gains (referred to as raw technology gains) that PCMOS has to offer over CMOS. An analytical formulation of this relationship is the basis of the graphical depiction of the "feasible surface" for application level gains from PCMOS over CMOS in the context of the HE application. This analytical formulation is a key component of our co-design methodology.

6.3. Experimental Validation of Analytically Derived Gains

The above characterization, essentially analytical, can be validated by implementing PSOC and SOC architectures corresponding to selected points in the "feasible surface". The points chosen for this comparison (in Table 5) both involve a custom (ASIC) logic based host processor executing the deterministic content of the application; as usual, the probabilistic content is implemented using alternate choices using CMOS and PCMOS technologies. Using the experimental infrastructure described in Section 5, the REPP gains are measured and presented in Table 5. As seen from the table, analytically derived and measured REPP values for the HE and PCA applications correlate extremely well. Similar validations can also be made for the remaining two applications.

Algorithm	Analytically derived REPP (with ASIC Host)	Experimentally measured REPP (with ASIC Host)
Hyper-Encryption	9.38	9.48
Probabilistic Cellular Automata	1969.6	1969.7

Table 5. REPP gains for the HE and PCA applications.

7. Concluding Remarks

We have demonstrated the value of the novel PCMOS technology within the context of embedded applications, which, by their nature often admit probabilistic behaviors. Thus, numerous applications from the speech, graphics and related media dependent domains are all inherently probabilistic in the following sense. The notion of the "quality of a solution", as well as the particular algorithms employed—face recognition applications realized using neural networks serve a very good example—respectively exhibit the role of probability both in the context of the quality of the solution as well as in context of the algorithm used to solve the problem. The improvements that we were able to demonstrate, which were *orders of magnitude* over application specific CMOS designs, surprised us as well—the raw gains from PCMOS over CMOS are in fact even higher and can be: for producing random bits, as high as a factor of **2,700,000** (from Table 3) per a probabilistic operation! We find this activity sufficiently promising that we wish to extend this work along two directions. First, we wish to explore a larger suite of applications, significantly from the signal processing (DSP) domain wherein the probability of correctness p at the device level manifests itself naturally as the *signal-to-noise* ratio at the level of a kernel such as a filter. Thus, we are actively developing filters wherein, PCMOS allows us to "tune" the signal-to-noise ratio and trade it for the energy consumed. Second, we are actively designing and fabricating the SOC implementations presented in this paper, as well as CMOS and PCMOS platforms for serving as platforms in the context of additional applications. An independent and equally interesting direction of inquiry involves investigating the applicability of the ideas, methods, and constructs presented here to the overarching question of realizing reliable computing from unreliable elements—such "probabilistic designs" are considered central to sustaining Moore's law and its concomitant benefits in the nanometer regime of CMOS designs (please see Borkar [2] and Moore [14]).

References

[1] I. Beinlich, G. Suermondt, R. Chavez, and G. Cooper. The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. *Proceedings of the Second European Conference on AI and Medicine*, pages 247– 256, 1989.

[2] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. *40th Design Automation Conference*, pages 338–342, 2003.

[3] S. Cheemalavagu, P. Korkmaz, and K. V. Palem. Ultra low-energy computing via probabilistic algorithms and devices: CMOS device primitives and the energy-probability relationship. In *Proc. of The 2004 International Conference on Solid State Devices and Materials*, pages 402 – 403, Tokyo, Japan, Sept. 2004.

[4] S. Cheemalavagu, P. Korkmaz, K. V. Palem, B. E. S. Akgul, and L. N. Chakrapani. A probabilistic CMOS switch and its realization by exploiting noise. *Submitted for publication, Available at http://www.crest.gatech.edu/palempbitscurrent/.*

[5] Y. Z. Ding and M. O. Rabin. Hyper-encryption and everlasting security. *Lecture Notes In Computer Science; Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, 2285:1–26, 2002.

[6] W. Feller. *An Introduction to Probability Theory and its Applications.* Wiley Eastern Limited, 1984.

[7] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong. Monte carlo simulations: Hidden errors from "good" random number generators. *Phys. Rev. Let*, 69:3382–3384, 1992.

[8] H. Fuks. Non-deterministic density classification with diffusive probabilistic cellular automata. *Physical Review E, Statistical, Nonlinear, and Soft Matter Physics*, 66(066106), 2002.

[9] E. Gelenbe and F. Batty. Minimum graph covering with the random neural network model. In *Neural Networks: Advances and Applications*, volume 2, 1992.

[10] B. Hayes. Computing science: Randomness as a resource. *American Scientist*, 89(4):300–304, 2001.

[11] L. B. Kish. End of Moore's law: thermal (noise) death of integration in micro and nano electronics. *Physics Letters A*, 305:144–149, Dec. 2002.

[12] D. MacKay. Bayesian interpolation. *Neural Computation*, 4(3), 1992.

[13] U. Maurer. Conditionally perfect secrecy and a provable secure randomized cipher. *Journal of cryptology*, 5:53–66, 1992.

[14] Moore's Law. http://www.intel.com/technology/silicon/mooreslaw/.

[15] K. Natori and N. Sano. Scaling limit of digital circuits due to thermal noise. *Journal of Applied Physics*, 83:5019–5024, May 1998.

[16] S. Nikoletseas, K. Palem, P. Spirakis, and M. Yung. Short vertex disjoint paths and multiconnectivity in random graphs: Reliable network computing. In *21st International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science Vol. 820 (Springer Verlag)*, pages 508–519, 1994.

[17] K. V. Palem. Energy aware algorithm design via probabilistic computing: from algorithms and models to Moores law and novel (semiconductor) devices. In *Proc. Intl. Conf. on Compilers, Architecture and Synthesis for Embedded Systems*, pages 113–117, San Jose, California, 2003.

[18] K. V. Palem. Energy aware computing through probabilistic switching: A study of limits. *IEEE Trans. Comput.*, to appear September 2005.

[19] S. Park and K. W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31, Oct. 1988.

[20] A. Pfeffer. *Probabilistic Reasoning for Complex Systems.* PhD thesis, Stanford Univeristy, 2000.

[21] O. Port. Chips that thrive on uncertainty. BusinessWeek, February 4, 2005.

[22] M. O. Rabin. Probabilistic algorithms, Algorithms and Complexity, New Directions and Recent Trends (ed. J. F. Traub). page 2939, 1976.

[23] Random Number Generation and Testing. http://csrc.nist.gov/rng/.

[24] J. M. Rehg, K. P. Murphy, and P. W. Fieguth. Vision-based speaker detection using bayesian networks. In *Computer Vision and Pattern Recognition*, pages 110–116, Ft. Collins, CO, June 1999.

[25] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[26] N. Sano. Increasing importance of electronic thermal noise in sub-0.1mm Si-MOSFETs. *The IEICE Transactions on Electronics*, E83-C:1203–1211, Aug. 2000.

[27] M. Schrder, H. Rehrauer, K. Seidel, and M. Datcu. Interactive learning and probabilistic retrieval in remote sensing image archives. *IEEE Trans. Geoscience and Remote Sensing*, 38(5), Sept 2000.

[28] K. L. Shepard and V. Narayanan. Conquering noise in deep-submicron digital ICs. *IEEE Design and Test of Computers*, 15:51–62, Jan. 1998.

[29] A. Sinha and A. P. Chandrakasan. Jouletrack a web based tool for software energy profiling. *Proceedings of the 38th conference on Design automation*, pages 220–225, 2001.

[30] R. Solovay and V. Strassen. A fast monte-carlo test for primality. *SIAM Journal on Computing*, 1977.

[31] K.-U. Stein. Noise-induced error rate as a limiting factor for energy per operation in digital ICs. *IEEE J. Solid-State Circuits*, 12:527–530, Oct. 1977.

[32] Trimaran Consortium. Trimaran: An infrastructure for research in instruction-level parallelism.

[33] S. Wolfram. *A New Kind of Science*. Wolfram Media, 2002.