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

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

Center for Research on Embedded Systems and Technology
Georgia Institute of Technology
Atlanta, Georgia, USA 30332.
{nsimhan,palem}@ece.gatech.edu

Abstract

Major impediments to technology scaling in the nanometer regime include power (or energy) dissipation and "erroneous" behavior induced by process variations and noise susceptibility. In this paper, we demonstrate that CMOS *devices whose behavior is rendered probabilistic by noise (yielding probabilistic* CMOS *or* PCMOS*) can be harnessed for ultra low energy and high performance computation.* PCMOS *devices are inherently probabilistic in that they are guaranteed to compute correctly with a probability* $1/2 < p < 1$ *and thus, by design, they are expected to compute incorrectly with a probability* $(1 - p)$*. In this paper, we show that* PCMOS *technology yields significant improvements, both in the energy consumed as well as in the performance, for probabilistic applications with broad utility. These benefits are derived using an application-architecture-technology (A^{2}T) co-design methodology introduced here, yielding an entirely novel family of probabilistic system-on-a-chip (*PSOC*) architectures. All of our application and architectural savings are quantified using the product of the energy and the performance denoted (energy* $\times$ *performance): the* PCMOS *based gains are as high as a substantial multiplicative factor of over* 560 *when compared to a competing energy-efficient* CMOS *based realization.*

1. Introduction

As CMOS technology scales down into the nanometer region, hurdles introduced by *noise* and other device perturbations (see Sano [14, 22], Kish [12] and Shepard [23]) pose several challenges. The surprising premise that noise can be harnessed as a resource, rather than viewed as a hurdle was validated for the first time using foundational principles and theoretical models (see Palem [15, 16, 17]). Building on these foundations, we have designed and studied CMOS devices [5] that are "unstable" or "noisy". In earlier work, we demonstrated for the first time that computation based on such noisy CMOS devices can yield orders of magnitude improvements simultaneously to the *energy consumed* as well as to the *running time*—collectively characterized as the *energy-performance product* (EPP)—of an application. The particular form of CMOS that is affected by ambient (thermal) noise—we refer to it as *probabilistic* CMOS or PCMOS—was the singular innovation through which these improvements were accomplished. The two significant contributions of this paper are (i) the demonstration of PCMOS based ultra efficient (embedded) computing architectures, where efficiency is quantified through EPP and (ii) a demonstration of the value of this novel technology in the context of a range of applications.

To demonstrate the utility and the efficacy of PCMOS, we first develop a methodology (akin to hardware software co-design and described in Section 3) that we refer to as *application-architecture-technology* (or A^{2}T) co-design. Our methodology is aimed at realizing extremely efficient *probabilistic system-on-a-chip* (PSOC) architectures using PCMOS devices. As shown in Figure 2, a canonical PSOC architecture consists of a (conventional) deterministic host processor used to compute most of the control-intensive components of an application, whereas the co-processor realized using PCMOS devices will be used as an energy-performance (EPP) "accelerator". In this novel co-design methodology, the "probabilistic content" (formalized later as *flux*) of the algorithm or application 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 architectural design efficiency differ significantly in the context of PCMOS, by contrast with those arising in the context of conventional CMOS based architectures.

Applications based on *probabilistic algorithms* with significant flux benefit the most from PSOC architectures. Probabilistic algorithms have found wide use in a range of embedded applications drawn from speech and pattern recognition, security and other domains. To evaluate the benefits of PCMOS based architectures, we considered a set of applications (Section 3) and four competing architectural realizations in silicon (Section 2); the associated gains are presented in Section 4. In Section 5, we study another crucial and novel aspect of computing architectures that implement probabilistic algorithms. Specifically, in application domains employing probabilistic algorithms, independent "probabilistic bits" are needed in copious quantities. Nevertheless, techniques for producing independent random bits are difficult and are an extensive area of study dominated by pseudo random number generators (PRNG) [19] with several complex approaches yielding unsatisfactory results [8]. Here, we show that in addition to yielding significant gains to the EPP, PCMOS technology also yields random bits of a high quality. We establish this by performing the tests provided by the National Institute of Standards and Technology (NIST) [21]. Concluding remarks and directions for future research are the subject of Section 6.

* This work is supported in part by DARPA under seedling contract #F30602-02-2-0124, by the DARPA ACIP program under contract #FA8650-04-C-7126 through a subcontract from USC-ISI and by an award from Intel Corporation.

2. Probabilistic system on a chip (PSOC) architectures

As mentioned in the introduction, the surprising premise that CMOS devices rendered probabilistic due to noise, are useful and yield energy and performance benefits at the application level will now be demonstrated using probabilistic system on a chip architectures (PSOCs). For completeness, we first present a brief overview of *probabilistic* CMOS (PCMOS) technology (for a detailed description, please see [5]) on which such architectures are based.

2.1. PCMOS technology

By studying PCMOS gates whose outputs are computed with a probability $p < 1$, we have shown [5] that the switching energy (E) grows exponentially with the probability of correctness (p). In addition, the noise magnitude quantified as its RMS and the switching energy E were shown to be quadratically related. These two relationships characterize the behavior of PCMOS devices. The behaviors were derived from analytical models of PCMOS gates and switches, and have been extensively studied and verified using HSpice simulations. In this paper, we use these PCMOS gates and their derived switches as building blocks, to demonstrate their benefits to applications through PSOC architectures.

2.2. PSOC architectures

To compare PSOC architectures with computing platforms based on conventional CMOS technology, algorithm and architecture realizations in four different scenarios (Figure 1) were considered: (a) when appropriate, the *best known* deterministic algorithm, implemented completely in software and executing on a low-energy host processor (in our case a StrongARM SA-1100), (b) a probabilistic algorithm for realizing the same application, with pseudo random bits generated by a software implementation of a well known algorithm [19] (both executing completely on the host processor), (c) the same probabilistic algorithm executing on the host processor, with a conventional CMOS co-processor (collectively referred to as the "conventional CMOS based SOC" or SOC for succinctness) and (d) with a functionally identical PCMOS based co-processor or a PSOC.

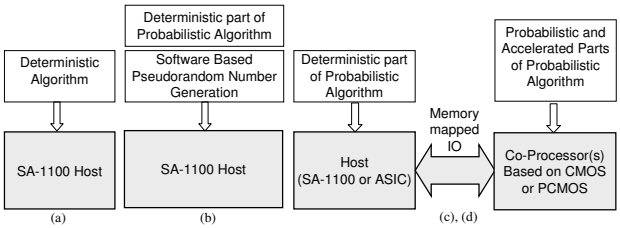

Figure 1. The host and co-processor realizations that are compared

These four cases encompass all reasonable alternate implementations of the application. Throughout this study, the co-processors are specific realizations using CMOS (case (c)) and PCMOS (case (d)), respectively of the probabilistic application in question; thus the co-processors are application-specific.

2.3. Performance and energy modeling of PSOC architectures

To estimate the performance of PSOC and SOC architectures, the simulator of the Trimaran infrastructure [11] (also see [4]) has been configured to determine the number of cycles taken by an application executing on a StrongARM SA-1100 host. This simulator also records a trace of the activity on the CMOS and PCMOS components of the PSOC. This information combined with the performance models of the co-processors obtained through HSpice simulations of PCMOS switches yields the PSOC and SOC performance in terms of execution time.

The energy consumption of an application executing on a PSOC or a 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 for communicating between the host and the co-processor(s). In this study, the co-processors are memory mapped and therefore, communication is realized through load-store instructions executed on the host. In all cases and to quantify the energy consumed by the SA-1100 host, the model described in [24] is used. This model is reported by its authors to be within 3% of the energy measured on an actual SA-1100 host. The energy modeling techniques applied to various components of the PSOC (SOC) architecture are illustrated in Figure 2. Since the

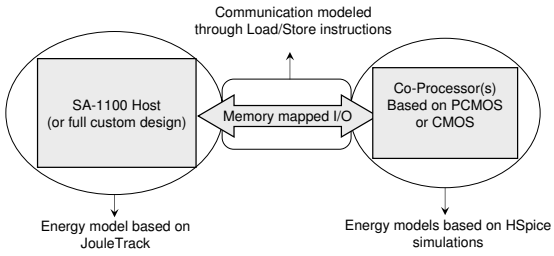

Figure 2. The host and co-processor architecture of a SOC and its energy-performance modeling.

co-processors are application-specific, the energy consumed by a particular co-processor varies with the application. The CMOS based co-processors were designed and synthesized using TSMC $0.25\mu m$ process, and the associated energy cost was determined from HSpice simulations. In the context of co-processors realized with PCMOS technology, the energy cost of the co-processor is derived similarly (and through physical measurements not reported here).

3. The A^{2}T co-design framework

In this study, we consider applications based on probabilistic algorithms that include *Bayesian inference* [13], *Random Neural Networks* [10], *Probabilistic Cellular Automata* [9] and *Hyper-Encryption* [6]. Any PSOC implementation of a probabilistic application involves *partitioning* the application between the host and the (application specific) PCMOS based co-processor. Even though the manner in which these applications are partitioned vary across individual applications, they follow a common theme, thus allowing us to suggest a methodology. The notion of a *core probabilistic step* with its associated probability parameter p is one such theme common to all of these applications, and across probabilistic algorithms in general. In our work, this core probabilistic step is identified by hand and implemented in PCMOS. The deterministic parts of the application are implemented as software executing on the host processor or as a customized application specific circuit (ASIC) when appropriate. This co-design methodology is unique in the sense that as opposed to traditional SOC designs, several unique algorithm and technology characteris-

tics explicitly motivated and grounded in PCMOS and its probabilistic behavior need to be considered, to obtain highly efficient designs.

3.1. Algorithm and technology characteristics influencing co-design

PCMOS is particularly efficient in computing with ultra-low energy. For example, the energy consumed for generating one random bit using PCMOS is 0.4 pico Joules [18]. By contrast, the Park-Miller algorithm [19] implemented in custom hardware in ASIC consumes about 2025 *times* this amount of energy. Given this dramatic difference and hence benefit, it is to be expected that having higher amounts of "probabilistic content" in the algorithm will yield greater opportunities for deriving benefits from PCMOS technology. Thus, the amount of "probabilistic content", which we refer to as the application's *flux*, and denoted by $\mathcal{F}$, will be a figure of merit. Flux is defined as the *ratio of probabilistic operations to the total number of operations* of the algorithm.

Though PCMOS is extremely energy efficient, the operating frequency of our current design is low [18], and has been determined to be about 1 MHz. By contrast, CMOS based pseudo-random bit generators produce pseudo-random bits at a rate as high as 4 million bits per second or more. Given this potential limitation, the peak rate at which an application consumes random bits, or the (peak) *application demand bandwidth* is a characteristic of interest. If the peak application demand bandwidth exceeds the bandwidth of the PCMOS based design—a design being an element or a building block that is PCMOS based, the PCMOS devices need to be replicated. Thus the need for extra bandwidth will be met through parallelism, and the amount is quantified as the *replication factor* $\mathcal{R}$. Based on these technology and algorithm characteristics, the applications of interest are partitioned, optimized and implemented as PSOC designs.

3.2. The suite of applications

In this section, we (due to space constraints) summarize the suite of applications, their partitioning and optimization, leading to the design of efficient PSOC architectures.

Bayesian Networks (BN) Bayesian inference is a statistical inference technique mimicking the human decision making process. *Hypotheses* and their corresponding *probability weights* are notions central to this technique. The probability weights are interpreted to be the *degrees of belief* associated with the corresponding hypotheses. Based on *evidences*, the degree of belief in a hypothesis is incremented (or decremented) till it approaches 1 (or 0) in which case the hypothesis is very likely (unlikely). A Bayesian network is used to perform a task referred to widely as Bayesian inference, and is modeled as a directed acyclic graph G of nodes V representing variables and edges E representing dependence relations between the variables. Each variable u uniquely represented by a node $v \in V$ can be assigned a value from a finite set of values Σ_u. Each value $\sigma \in \Sigma_u$ has a conditional probability $p(\sigma/\sigma' \in \Sigma')$ associated with it, where $\Sigma' \in (\Sigma_1 \times \Sigma_2 \times \Sigma_3 \cdots \Sigma_l)$ is the string of values of the variables represented by all of the l parents of u. Variables whose values are known apriori are called *evidences* and based on such evidence, other variables are *inferred*. The particular Bayesian networks considered in this study is a part of the following applications: a hospital patient management system and printer troubleshooting in a Windows operating system environment.

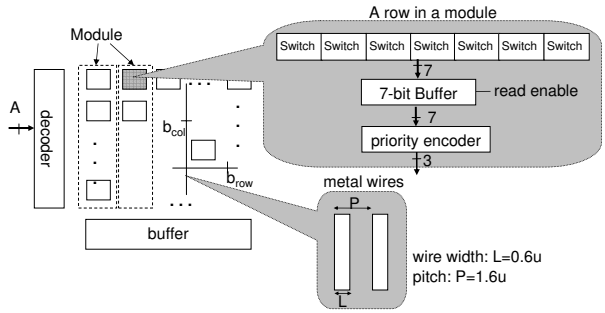

Figure 3. The co-processor architecture of a PSOC which implements Bayesian inference.

Partitioning and Optimization We choose the likelihood weighting algorithm [20] for Bayesian inference. The random experiment (used for inference) in this probabilistic algorithm, is implemented in the PCMOS co-processor (consisting of several *modules*), with the remainder implemented as software executing on the host. In a Bayesian network G, the conditional probabilities associated with each value of the variables of a node are known apriori, and are used to design a module of PCMOS switches (inverters), one module per node v in the graph. As an example, consider a node u with $\Sigma_u = \{0, 1, 2\}$. As before, let Σ' be an instance of the string of values associated with the parents of u. Let $0 \leq p(0/\sigma'), p(1/\sigma'), p(2/\sigma') \leq 1$ be the conditional probabilities associated with $0, 1, 2 \in \Sigma_u$ respectively, given that the parents of the node v have outputs $\sigma' \in \Sigma'$. In our PSOC architecture, Bayesian inference will be performed by three PCMOS switches A, B and C corresponding to $0, 1, 2$ respectively. The inputs for these switches are fixed at 0 and the probability of correctness associated with A, B, C is by design, $p(0/\sigma')$, $\frac{p(1/\sigma')}{1-p(0/\sigma')}$ and $\frac{p(2/\sigma')}{1-p(0/\sigma')-p(1/\sigma')}$ respectively. Thus, when the switches are inspected in the order $< A, B, C >$, the value which corresponds to the first switch whose output is the value 1 is the value inferred by node u. In the PSOC design, the set of switches $\{A, B, C\}$ will be referred to as a *row* and each distinct switch in this set will be referred to as an *element*. Since a row is associated with each element of the set Σ', many rows are required to implement the strings associated with the space of all possible outputs corresponding to the parents of the node u from Σ'. These set of rows will be referred to as a *table*.

As shown in Figure 3, the PCMOS module corresponding to a node u implements a table, whose row is indexed by a particular string σ' of values associated with the parents of u computed earlier. The number of columns in the table is $|\Sigma_u|$, where each column corresponds to a value from the set Σ_u; in our example, $|\Sigma_u| = 3$. An element in the table, identified by <row, column> is a specialized PCMOS switch whose probability of correctness is computed as indicated above. Finally a conventional priority encoder is connected to the outputs of a row to determine the final result of the random experiment; it performs the function of inspecting the values of a row and choosing the final output associated with u.

Random Neural Network (RNN) Following Gelenbe [10], a random neural network consists of *neurons* and *connections* between the neurons. Information is exchanged between the neurons in the form of *bipolar signal trains*. Neurons have potentials associated with them, which are defined to be the sums of incoming signals. This potential in turn, influences the rate of firing. The particular neural network considered in this study is used to heuristically determine the vertex-cover

of a graph due to Gelenbe and Batty [10].

Partitioning and Optimization The Poisson process which models the "firing" of a neuron is implemented in the PCMOS co-processor, with the rest of the computation implemented to execute on the host processor. To realize the Poisson process characterizing a neuron firing, the Bernoulli approximation of a Poisson process [7] is used. As an example of a methodological step in our A^2T co-design approach, since the rate at which random bits are required by the host exceeds the rate at which PCMOS based switches can compute, the "neurons" in the co-processor of the PSOC are replicated to match the required rate. In the interests of efficiency, and as another example of our A^2T methodology, the application is restructured to reduce the replication factor $\mathcal{R}$, by interleaving the demand for random bits and the processing of these bits on the host—distributing the firings more evenly over the course of the entire application's execution. This has the effect of reducing the *peak* application demand bandwidth.

Probabilistic Cellular Automata (PCA) are a class of cellular automata used to model stochastic processes [9]. Cellular automata consist of *cells* with local (typically nearest neighbor) communication. Each cell is associated with a *state* and a simple *transition rule* which specifies its next state, based on its current state and typically, the states of its neighbors. In the *probabilistic* string classification application due to Fuks [9], the state of each cell assumes a value of 0 or 1, giving rise to 8 possible transition rules (each rule has two possible outcomes, 0 or 1). In addition, each transition rule is probabilistic: for a transition rule τ_i ($0 \leq i \leq 7$), the probability that the output state of the rule is 0 is denoted by $p_{i,0}$ and the probability that the output state is 1 is denoted by $p_{i,1}$.

Partitioning and Optimization Each transition rule is implemented in the co-processor by a PCMOS switch whose input is a 0. The probability of correctness associated with the i^{th} switch is $p_{i,1}$. Again, the control-intensive part of choosing a transition rule (based on the state of a cell and the states of its neighbors) and updating the states upon evaluating the rules are all implemented on the host processor. Since the rate at which the transition rules are evaluated exceeds that supported by PCMOS devices, this structure is again replicated many times with concomitant optimizations.

Hyper-Encryption (HE) is a provably secure encryption technique proposed by Ding and Rabin [6] in the bounded storage model. This scheme consists of generating an *encryption pad* based on a publicly available random string α and a shared secret key between the sender and the receiver. The secret key S is a sequence of whole numbers $S = s_1, s_2, s_3 \cdots s_k$ such that each number $0 \leq s_i < |\alpha|$. If $\alpha[j]$ is the j^{th} bit of α, the encryption pad is generated by $\alpha[s_1] \oplus \alpha[s_2] \oplus \ldots \alpha[s_k]$, where $\oplus$ denotes the pairwise exclusive OR (XOR) function. Message encryption is performed by a bit-wise XOR operation of the encryption pad with the message.

Partitioning and Optimization In the PSOC, the random string α is generated using PCMOS while the generation of the encryption pad as well as the encryption are performed by the host. Both in the context of PCA and Hyper-Encryption (as shown in Figure 4), the SA-1100 host is also replaced by custom hardware.

4. Metrics, results and analysis

In order to characterize and quantify benefits derived through PSOC architectures, we now define a variety of metrics. In the interests of staying within the stipulated page limits, our development will be brief.

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	4.17×10^4	7.7×10^2
HE	1.56×10^5	2.03×10^3

Table 1. The EPP gain of PCMOS over SA-1100 and over CMOS for the core probabilistic step

4.1. Metrics for quantifying the application level benefits

Energy performance product: EPP described earlier, is defined as the product of the energy consumed by the application and its execution time. This metric will be used as the primary figure of merit to evaluate alternate implementations, including SOC and PSOC variants. Given the EPP of two alternate realizations, they can be compared as follows.

Energy performance product gain: $\Gamma_{\mathcal{I}}$ is the ratio of the EPP of the baseline denoted by the symbol β to the EPP of a particular implementation $\mathcal{I}$ (e.g., a PSOC or an SOC). This ratio is calculated as follows:

$$\Gamma_{\mathcal{I}} = \frac{Energy_{\beta} \times Time_{\beta}}{Energy_{\mathcal{I}} \times Time_{\mathcal{I}}} \quad (1)$$

For determining $\Gamma_{\mathcal{I}}$, and unless otherwise stated, the baseline (and hence, the numerator of $\Gamma_{\mathcal{I}}$) always corresponds to the case when the entire computation is performed on the host processor. The StrongARM SA-1100 serves as the baseline processor, and therefore, there is no co-processor. For example, in the context of the RNN application solving the vertex cover, the baseline is the StrongARM SA-1100 computing the deterministic as well as the probabilistic content, whereas $\mathcal{I}$ is the combination of the StrongARM SA-1100 as the host computing the deterministic component and the co-processor computing the probabilistic components of the application.

Quality of probabilistic implementation: This attribute is characterized empirically based on the statistical tests from the NIST suite [21] and will be the subject of Section 5.

4.2. Gains of core probabilistic steps through PCMOS

The application level gains in energy and performance (when compared to the baseline case where there is no co-processor) is attributed to the efficiency of the co-processor while executing the core probabilistic operations. We summarize these gains of PCMOS over StrongARM SA-1100, and over custom CMOS implementation for the *core probabilistic step* for each of the applications in Table 1. Each row of this table corresponds to one of the four distinct applications of interest to us and the gains achieved per core probabilistic step are shown there. As can be readily seen from Table 1, these gains are substantial—orders of magnitude greater—in both contexts. 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 $\mathcal{F}$.

4.3. Application level gains of PCMOS

As summarized in Table 2, gains at the scope of an entire application range from a factor of about 80 for the PCA application, to a factor of about 300 in the context of the RNN application. As mentioned earlier, the baseline implementation for HE, PCA and RNN applications is the StrongARM SA-1100 computing the deterministic as well as the probabilistic content and $\mathcal{I}$ is a PSOC executing an identical probabilistic algorithm. For the BN case, the baseline is the StrongARM SA-

1100 computing the deterministic junction tree algorithm and $\mathcal{I}$ is a PSOC executing the likelihood weighting algorithm. A range of EPP gains are observed whenever multiple data points are available, for example, in the context of the Bayesian inference 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 largely due to increase in flux. Similar increases are observed for the other applications as well, caused by greater flux values in the application as shown in the table.

Algorithm	Flux $\mathcal{F}$ (as percentage of total operations)	$\Gamma_{\mathcal{I}}$	
		Min	Max
BN	0.25%-0.75%	12.5	291
RNN	16.4%-19.7%	226.5	300
PCA	4.19%-5.29%	61	82
HE	12.5%	1.12	1.12

Table 2. Application level flux, maximum and minimum EPP gains of PCMOS over the baseline implementation where the implementation $\mathcal{I}$ has a StrongARM SA-1100 host and a PCMOS based co-processor

4.4. Impact of host efficiency

We will now consider and extend the gains to the entire application suite. We will delineate the (less obvious) impact of the efficiency of the host processor on the gain of an implementation $\Gamma_{\mathcal{I}}$. Referring back to the Table 2, the striking aspect of the gain is evident in the context of the HE application. From Table 1 we note that the energy consumed by each core probabilistic step in the context of the HE application is over a factor of 150000 while using the SA-1100 host, when compared to a PCMOS based design. Furthermore, as seen in Table 2, the HE application has high flux—much higher than the corresponding values of BN and PCA applications. Yet, the HE application does not seem to demonstrate any gain at all, since $\Gamma_{\mathcal{I}} = 1.12$. We will devote the rest of this section to try and understand this potential "anomaly".

Once again, in the interests of staying within the mandated space limits, we will restrict detailed discussion to the HE application. The reason for this "anomaly" is that the StrongARM host is extremely inefficient. Thus the relative savings of PCMOS, while significant are rendered insignificant as an overall proportion of the entire application, when the StrongARM host is included. Thus gains through PCMOS—the limits being substantial as shown in Table 1—can be truly achieved only if the amount of effort spend in the co-processor is comparable in terms of EPP units to that spent in the host. To verify this hypothesis, a baseline SOC architecture in which the host processor and the co-processor are both custom ASIC architectures (Figure 4) is considered. With this notion, moving away from a StrongARM host processor to one realized from custom ASIC logic, amount of energy and running time spent in the host is considerably lower. Thus and perhaps counter intuitively, increasing the efficiency of the competing approach enhances the value of PCMOS gains at the application level. In the context of the HE application, and with this change to the baseline, the gain $\Gamma_{\mathcal{I}}$ increases to 9.38 - almost an order of magnitude. Similarly when a baseline with a custom ASIC host is used, the $\Gamma_{\mathcal{I}}$ value in the context of the probabilistic cellular automata application increases to 561. We view this fact as being extremely favorable for PSOC based designs. Thus, as host processors become more efficient with future technology generations, the gains of PSOC architectures over conventional SOC architectures increase.

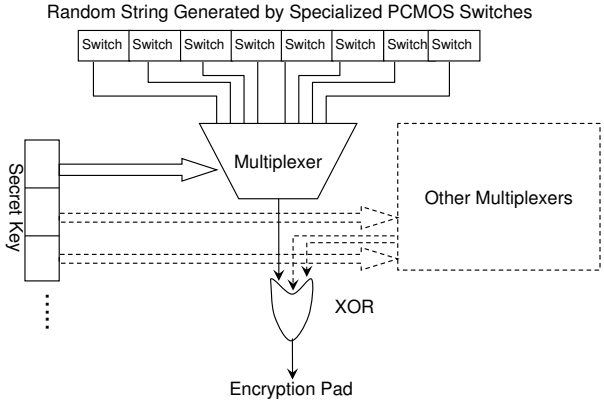

Figure 4. The Custom ASIC host and its PCMOS co-processor constituting a PSOC implementation, for Hyper-Encryption

5. The value of PCMOS to quality of randomness

While the EPP gains for applications have been our significant concern to demonstrate the utility of PCMOS, the *quality* of the implementation of a probabilistic algorithm is a characteristic of interest as well. Random bits of low quality affect application behavior—from the correctness of Monte Carlo simulations [8] to the strength of encryption achieved by schemes such as Hyper-Encryption [6]. We employed statistical tests from the NIST suite [21] to assess to quality of randomness in a preliminary way. The random sequences in the case of PCMOS have been produced from physical measurements of a probabilistic inverter fabricated using the $0.25\mu m$ TSMC process, whereas the pseudo-random bits derived using Park-Miller [19] algorithm were evaluated using the output of a custom design simulated using HSpice. In both cases, $p = 0.5$.

The results of these comparisons are shown in Figure 5. Among these tests and to highlight a few, the *runs* test, is used to determine a contiguous sequence of bits with a value 1 in a block. The *rank* test is used to check the linear dependence, while the *FFT* and *approximate entropy* tests detect periodicity and frequency of overlapping patterns. In evaluating the test results, we employed the testing strategy and criteria as recommended by NIST. Specifically, the test results shown in parenthesis in the table are compared against a threshold (the recommended value being 0.93) used to determine whether the sequence passes (or fails) a test. The tests are performed on random bit sequences of length $20,000,000$. The result indicates the proportion of sub sequences (tested through iterations) that pass, from the random sequence being tested. As seen from the figure, the quality of random sequences generated by PCMOS is higher than that of those generated by CMOS, since more tests in the former case yield a pass result compared to the latter—eleven tests with a pass score in the context of PCMOS whereas seven in the CMOS context, out of a total of fourteen tests.

6. Conclusion and remarks

We have demonstrated the value of the novel PCMOS technology within the context of realizing ultra efficient PSOC architectures, over a range of applications ubiquitous to embed-

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 (0.93)	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 5. Comparison of quality of randomization for PRNG and PCMOS.

ded computing and beyond. The improvements that we were able to demonstrate were *orders of magnitude* over application specific CMOS designs. Next, we wish to explore a larger suite of applications and associated PSOC architectures, 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 computational kernel such as a filter. Another interesting and valuable intellectual direction to pursue in the future involves a thorough and in-depth exploration and study of the quality of randomness. We note in passing that PCMOS has the ability of producing random bits as opposed to the pseudo-random bits that conventional random number generators produce. Measuring the quality of randomness remains a challenge, if approaches other than empirical—such as those advocated by NIST—are sought, since the complexity of provably correct tests for randomness can be undecidable [3]. This question of generating pseudo-random bits has deep roots in computer science with connections to important questions about the inherent difficulty of computations (see Blum and Micali [1]). In a sense, A^2T co-design methodology based on PCMOS, and thus a source of high quality *random* bits can be viewed as a technological response to the significant challenge embodied in Yao's comment from 1982—*"If, in an application, it is possible to isolate some simple randomness properties that can guarantee success, then a statistical test based on the desired randomness properties can be used to screen and select a appropriate generator. This, however, is seldom the case. Furthermore, the performance of a pseudo-random number generator under a particular statistical test is usually hard to determine analytically, and often has to rely on empirical evidence."* [25]. Finally an independent and equally interesting direction, 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 in the nanometer regime of CMOS based architectures [2].

References

[1] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984.

[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] G. Chaitin. Algorithmic information theory. *IBM Journal of Research and Development*, pages 350–359, 1977.

[4] L. N. Chakrapani, J. Gyllenhaal, W. mei W. Hwu, S. A. Mahlke, K. V. Palem, and R. M. Rabbah. *Trimaran: An Infrastructure for Research in Instruction-Level Parallelism*, volume 3602. Springer-Verlag Berlin Heidelberg, Aug. 2005.

[5] 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. *Proceedings of the IFIP international conference on very large scale integration*, 2005.

[6] 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.

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

[8] 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.

[9] H. Fuks. Non-deterministic density classifiation with diffusive probabilistic cellular automata. *Physical Review E, Statistical, Nonlinear, and Soft Matter Physics*, 66, 2002.

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

[11] http://www.trimaran.org. Trimaran: An infrastructure for research in instruction-level parallelism.

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

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

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

[15] 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.

[16] K. V. Palem. Proof as experiment: Probabilistic algorithms from a thermodynamic perspective. In *Proc. Intl. Symposium on Verification (Theory and Practice),*, Taormina, Sicily, June 2003.

[17] K. V. Palem. Energy aware computing through probabilistic switching: A study of limits. *IEEE Transactions on Computers*, 54(9):1123–1137, 2005.

[18] K. V. Palem, L. N. Chakrapani, B. E. S. Akgul, and P. Korkmaz. Realizing ultra-low energy application specific soc architectures through novel probabilistic CMOS (PCMOS) technology. In *Proceedings of The International Conference on Solid State Devices and Materials (SSDM)*, Sept. 2005.

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

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

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

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

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

[24] 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.

[25] A. Yao. Theory and application of trapdoor functions. *Proceedings of the 23rd symposium on the foundations of computer science*, pages 80–91, 1982.