

Low-Power Instruction Bus Encoding for Embedded Processors

Peter Petrov, *Student Member, IEEE*, and Alex Orailoglu, *Member, IEEE*

Abstract—This paper presents a low-power encoding framework for embedded processor instruction buses. The encoder is capable of adjusting its encoding not only to suit applications but furthermore to suit different aspects of particular program execution. It achieves this by exploiting application-specific knowledge regarding program hot-spots, and thus identifies efficient instruction transformations so as to minimize the bit transitions on the instruction bus lines. Not only is the switching activity on the individual bus lines considered but so is the coupling activity across adjacent bus lines, a foremost contributor to the total power dissipation in the case of nanometer technologies. Low-power codes are utilized in a reprogrammable application specific manner. The restriction to two well-selected classes of simply computable, functional transformations delivers significant storage benefits and ease of reprogrammability, in the process obtaining significant power savings. The microarchitectural support enables reprogrammability of the encoding transformations in order to track code particularities effectively. Such reprogrammability is achieved by utilizing small tables that store relevant application information. The few transformations that result in optimal power reductions for each application hot-spot are selected by utilizing short indices stored into a table, which is accessed only once at the beginning of the transformed bit sequence. Extensive experimental results show significant power reductions ranging up to 80% for switching activity on bus lines and up to 70% when bus coupling effects are also considered.

I. INTRODUCTION

THE proliferation of nanometer technologies with ever decreasing geometric sizes have led to proportionally unprecedented levels of system-on-a-chip (SOC) integration. Embedding a multitude of processor cores has become a widespread design practice due to its advantages of highly reduced time-to-market, lower design cost, and easily reprogrammable implementations. Yet, increased silicon integration, together with ever increasing clock frequencies, have led to proportional increases in terms of power.

Energy efficiency has already been well established as an important product quality characteristic. In mobile applications, such as handheld and wireless devices, not only does it impact directly cost of ownership, but furthermore may severely undermine the usability and acceptance of the product. Consequently, techniques for minimizing system power consumption are of significant importance for achieving high-product quality.

Embedded processors, deeply integrated and utilized for a multitude of SOC subsystems, have already established themselves as the unquestionable leader in terms of volumes, electronic market share, and revenues, greatly surpassing their close relatives, the general-purpose processors. By virtue of their very nature, many general-purpose optimization techniques are either not directly applicable due to their nontrivial overhead, or just scratch the surface of otherwise promising optimization opportunities, being severely limited by their generality. The dramatic volumes in the embedded processor marketplace necessitate innovative approaches, particularly to surmount the obstacles posed by power considerations in a diverse set of embedded processor market segments. Meeting the more stringent power challenges in the embedded processor marketplace necessitates an ability to exploit the one aspect that differentiates embedded processors from general-purpose processors, namely, advance knowledge regarding their application context.

The elaborate exploitation of application knowledge and its subsequent utilization as a driving force for customizing the embedded processor microarchitecture constitute a conceptual framework that establishes new grounds, enabling a plethora of opportunities for drastically improving both system power and performance [1]–[3]. We have shown previously that customizable microarchitectural components, such as instruction/data caches capable of utilizing precise application knowledge regarding data reuse regularities can achieve significant performance improvements through miss rate reductions [1], while information regarding the particular memory layout can be exploited in a way that leads to highly optimized tag operations with the concomitant drastic power reductions [3]. Application information regarding data memory address streams can be efficiently exploited [2] in order to achieve significant power reductions on the address bus to data memories as more than 90% of the address traffic can be completely eliminated. In all these cases, an application-specific customizable microarchitecture takes informed decisions as to how to handle various architecture-specific actions. As this framework enables the microarchitectural utilization of global application properties identified off-line by compile/link time algorithms, traditional mainstream compiler algorithms remain unaffected. Since flexible and cost-efficient implementations together with ease of maintenance are the primary advantages buttressing the wide acceptance of processor-centric system designs, it is of paramount importance that the customization support be designed as a microarchitecturally reprogrammable implementation capable of dynamic recustomization. This reprogrammability is achieved on a microarchitectural level, rather than through gate-level approaches such as field programmable gate arrays

Manuscript received July 20, 2003; revised December 27, 2003. This work was supported by the National Science Foundation under Grant 0082325.

The authors are with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: ppetrov@cs.ucsd.edu).

Digital Object Identifier 10.1109/TVLSI.2004.831468

(FPGAs), thus achieving cost-efficient performance and power consumption.

Instruction memories not only occupy a significant portion of many SOC designs but also constitute a significant contributor in terms of power consumption as they are typically accessed at each execution cycle. The power associated with instruction memories can be divided into two major parts. The first part is the power consumption associated with the read/write operations to the memory arrays, which directly correlates to *memory size and organization*. To alleviate this problem, extended research work has been performed targeting code compression [4], [5] with the aim of code size minimization. Minimizing the program code size engenders smaller memories, which in turn translates into lower power when reading from these memory arrays.

Communicating the instructions from memories to the processor front-end on long interconnect buses constitutes the second major part of the power consumption associated to instruction memories. This bus power overhead has emerged as a determinative factor only recently with the wide adoption of SOC design styles. A typical SOC design contains several embedded processor cores responsible for various parts of the total system functionality. Groups of these processors usually share on-chip instruction memories or instruction caches as they frequently implement common functionality. This architecture leads to star-like interconnect topologies to on-chip instruction memories and caches which in turn exhibit long interconnect buses. This type of SOC architecture is common for digital signal processing (DSP) heavy products where clusters of DSP processor cores, implementing a common task in parallel, are mapped to a shared instruction memory. Furthermore, individual instruction caches for such multiprocessor platforms are impractical as they lead to significant area and power overhead. Consequently, communicating instructions to the embedded processors relies on the utilization of long interconnect buses, which exhibit high capacitance. An access to these memories is typically performed each cycle in order to fetch the next instruction to be executed, further exacerbating the associated power consumption. The frequent utilization of the very large instruction word (VLIW) processors in multimedia oriented tasks further aggravates the power problem associated to bus interconnects as this class of embedded processors exploits extremely wide instruction buses. Consequently, the interaction between a processor and its instruction memory significantly contributes to the total power consumption associated with instruction memories. Bus encoding techniques that address this problem and aim at minimizing bit transitions [6]–[9] have emerged, subsequently, as a means to reduce the bus communication power.

Code compression techniques [4], [5] aim primarily at minimizing memory volume, thus leading to narrower buses and possibly to reduce power consumption on the instruction bus. These techniques, orthogonal and complementary to the techniques that we propose in this paper, are effective in extracting reductions only if regularity exists in the underlying code. They furthermore exacerbate the need for the techniques we propose as in the process of code compression any inherent code regularity is converted into increased entropy, leading to greater

power consumption in the form of increased toggles on bit transitions and cross-coupling effects.

We present in this paper an application-specific customization methodology for minimizing the significant power dissipation on instruction buses communicating the program code between processors and their instruction memories. After compilation, the application binary code is analyzed with particular emphasis on the major application loops. A cost-efficient, energy-aware encoding is identified and applied on the bit streams formed by each bit position of the particular instruction sequence, with the objective of minimizing the net effect of both the single and the coupled bit transition events. The transformed binary code is stored into the instruction memory, while the particular application-specific functional transformations identified by the post-compile coding algorithm that we propose in this paper are communicated to a special front-end microarchitectural support. These transformations are subsequently utilized during program execution in order to restore the original bit sequence for each bus line by applying the transformations.

While it is essential to reduce the number of bits needed for indexing the applicable transformation, the domain of transformations needs to suffer no such limitation as long as a “transformation window” is able to restrict the class of transformations applicable at any instance. In this paper, we present a microarchitecture capable of windowing to select among a complete set of transformations. The proposed methodology offers a comprehensive set of transformation classes selected judiciously so as to achieve optimal power reductions, yet with extremely cost-efficient hardware overhead. The inclusion of purely recurrent transformations, capable of generating the complete original instruction word out of a highly limited set of initial bits, results in dramatic power savings. For such purely recurrent transformations, most of the encoded bits are effectively transformed into “don’t-care” values, which in turn enables their specification and transmission in ways solely aimed at minimizing the targeted power cost function.

In contrast to all previous work in low-power bus encoding, the techniques we propose in this paper require no hardware support on the memory side and introduce no additional bus lines to the already existing bus organization and topology. This is due to the fact that all the required algorithmic support for identifying the optimal transformations and obtaining the power optimized instruction code is performed off-line when compiling and linking the program code. The only required hardware support is the decoding circuitry on the processor front-end. The absence of any encoding hardware on the critical memory side and of any bus modifications whatsoever gives the approach we propose clear advantages in terms of performance impact and hardware area overhead compared to the existing low-power bus encoding techniques.

The proposed hardware support is not only highly *cost-efficient* due to the functional nature of the proposed bit sequence transformations, but is also *programmable* and accessible by software, thus enabling the design, implementation, and manufacturing of a *single and fixed processor architecture*, while allowing optimal per-application program encoding for drastic power reductions achieved in a completely reprogrammable way with no need for costly design iterations.

II. RELATED WORK

Recently, minimizing the number of bit transitions on communication buses is a problem being actively pursued by a number of research groups. A multitude of techniques for low-power *address bus* encoding, exploiting the regularity of these buses have been developed. The *T0* approach [7] introduces an additional line to the instruction memory address bus in order to exploit the sequentiality typical of the instruction addresses. When the additional line is asserted, the memory controller computes the new address by simply incrementing the previous one. The requirement for an additional redundant control line is eliminated in [8] by observing that any new address transmission is sufficient in recognizing that the address incrementation mode is no longer in effect. The slight problem of signaling the address that originally started the incrementation mode is resolved by transmitting the actual value of the incrementation instead. The locality of reference in addresses to instruction and data memories is actively exploited in the address bus encoding technique proposed in [10]. Frequently accessed zones are dynamically identified and only an offset from the previous address within the zone together with a zone identifier are sent to the memory. While these techniques represent effective solutions for instruction addresses, it is difficult to see how they can be applied to the instructions themselves which unlike their addresses exhibit no apparent regularity. The low-power encoding proposed in [11] utilizes self-organizing lists in order to achieve an optimal encoding for the most frequently accessed addresses. By utilizing self-organizing lists, the approach exploits the temporal and spatial locality of the addresses on both instruction and data address buses, albeit at a significant hardware cost. All these approaches target address buses by effectively exploiting the highly regular patterns associated with address streams. Their applicability to buses with uncorrelated data streams would be highly limited.

Data buses, typically exhibiting highly uncorrelated data patterns, need a different approach, one that does not rely on any assumption about data regularity. The *Bus-Invert* method [6] and its derivatives [12], [13] invert the bus content whenever this leads to a smaller Hamming distance compared to the previous value on the bus; an additional bus line informs the receiver whether the bus content has been inverted. An in-depth theoretical analysis of the *Bus-Invert* method has been presented in [14]. For buses with uniformly distributed data, an expected value analysis shows benefits of around 10% for 32-bit buses, in line with reported experimental results. Increasing the bus width results in reduced expected power reductions, necessitating partitioning of bus lines so as to extract power savings, though at increased additional bus line cost. An approach that combines *Bus-Invert* with *transition signaling* is presented in [15]. With transition signaling, a bit value of 1 is sent on the bus line only if a bit transition from the previously transferred value needs to be indicated. It has been shown that the approach performs well for data buses in DSP systems, while its utility for instruction buses would be extremely limited. In [9], source-coding schemes are proposed to reduce bit transitions on buses with data sources characterized in a probabilistic manner. Encoding schemes comprising of a pair of functions, decorrelation and entropy coding,

are proposed with various practical instances of such functions evaluated and analyzed. While this approach works well for correlated data sets delivering power reductions of up to 35%, its applicability to data streams with no correlation, i.e., instruction buses, is circumscribed with improvements limited to 19%.

Furthermore, in the case of deep submicron technologies, the capacitance between bus lines and the substrate, i.e., the *self-capacitance*, is not the sole parasitic capacitance that needs to be considered for efficient power dissipation modeling. As shown in recent studies [12], [16], [17], the effects of the *coupling-capacitance* between adjacent bus lines are becoming ever more prominent. Therefore, in order to successfully model and optimize the bus power dissipation for deep submicron technologies, a power minimization approach has to target both the single bus transitions and the coupling activities. Such an approach, a derivative of the *Bus-Invert* method, is proposed in [12] and [18].

A methodology for low-power instruction set architecture (ISA) encoding has been proposed in [19]. Statistical data concerning instruction adjacency is collected from instruction set simulations on a set of applications. The opcode space is selected in such a way so as to minimize the Hamming distance between frequently encountered instruction pairs. As this is a technique for designing a power efficient, yet fixed, ISA encoding, its benefits directly depend on the existence of a strong bias in the occurrence frequencies of instruction pairs for all programs to be eventually executed in this processor. Even if such a statistical bias exists, the statistical benefits may be attenuated, if not outright contradicted, for particular program subsets. Moreover, the proposed encoding targets only instruction opcodes, while the remaining part of the instructions is left intact, as no such coding transformation can be effected in cases such as immediate operands, for example.

III. POWER MODEL FOR DEEP SUBMICRON BUSES

Bit transitions on bus lines are one of the major contributing factors to the power consumption associated with communicating data on on-chip and off-chip buses. The amount of power dissipated due to this switching activity is proportional to both the voltage level on the bus and the capacitance between the bus line and ground. The defining equation for power dissipation is $P = (1/2)\alpha.C.V_{dd}^2.f$, where α indicates the average number of bit transitions, V_{dd} the bus voltage level and f the bus clock frequency. As the power dissipation formula indicates, total power consumption is directly proportional to the number of transitions α that charge and discharge the capacitance C . However, when considering communication buses, there is more than one type of capacitance that needs to be taken into account while aiming at minimizing the total power consumption.

The traditional techniques for bus power minimization so far have been focused mainly on reducing the number of bit transitions on each individual bus line. This optimization goal has stemmed from the fact that for process technologies with larger feature sizes the self-capacitance between a bus line and ground, as denoted by C_s in Fig. 1(a), has been the dominating capacitance factor determining the power dissipation. In the first power

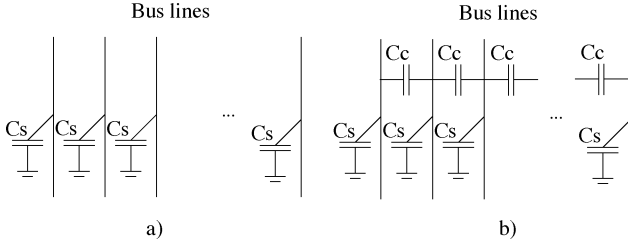


Fig. 1. Self and coupling capacitances for interconnect buses.

model explored in this paper the bus power consumption is proportional to the total number of bit transitions on each bus line. This is the traditional power model used in most of the research efforts regarding bus encoding for low power [6]–[9] and we refer to it hereafter as the *Self-Capacitance (SC) model*.

With the development of nanometer technologies, the coupling capacitance between adjacent bus lines plays an increasingly important role in determining bus power dissipation. This capacitance is pointed out in Fig. 1(b) and denoted as C_c . In the case of such technologies, both the self-capacitance and the coupling-capacitance need to be considered in order to produce efficient power reductions.

Bit transitions on each individual bus line result in charging and discharging C_s which in turn leads to power dissipation in the amount of $C_s V^2/2$. The coupling capacitance C_c in between the adjacent bus lines is charged/discharged when there exist bit transitions on the neighboring lines that cause a voltage level difference on these lines. There are two basic types of such coupling events on adjacent bus lines. A detailed treatment of this issue with low-level electrical analysis can be found elsewhere, such as in [12] and [17]. The first type of coupling events occurs in the cases of bit transitions on only one of the bus lines from the pair, for example $\{00 \rightarrow 01\}$, $\{10 \rightarrow 11\}$, etc. The power dissipated for this set of single bit transition events is measured as $C_c V^2/2$. The other coupling event occurs when there are bit transitions on both the lines, but in opposite directions. There are two such cases, namely $\{01 \rightarrow 10\}$, $\{10 \rightarrow 01\}$. These two cases are also known as *toggling* events. For these cases though, the amount of power dissipated is $2C_s V^2$, four times larger than for the first type of coupling events! The remaining cases, i.e., no bit transitions on either line or transitions in the same directions, do not lead to any activity on the coupling capacitance. This power model, referred hereafter as a coupling and self capacitance (CSC) model, is the second model that we utilize in this paper. As this model handles adjacent bus lines and the coupling transitions between them, a more sophisticated approach for encoding the bus lines is needed so as to achieve a solution which aims at minimizing both the coupling and the self bit transitions.

Considering all these cases, the total power consumption can be expressed as the sum of these components, i.e., $P_{\text{bus}} = \text{SBT} * C_s V^2/2 + \text{CBTr} * C_c V^2/2 + \text{CBTg} * 2C_c V^2$, where SBT denotes the total number of single bus line bit transitions, while CBTr and CBTg denote the number of cross bus line bit transitions and toggles. As suggested in [12] and [17], C_c and C_s are of similar magnitude, simplifying the equation to $P_{\text{bus}} = (\text{SBT} + \text{CBTr} + 4 * \text{CBTg}) * C_c V^2/2$. As the values

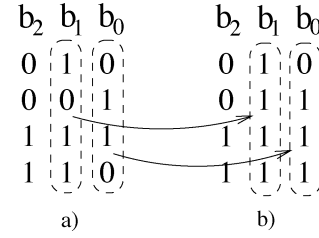


Fig. 2. Example bit sequences of three bus lines.

of C_c and V are fixed for a given technology process, minimizing the first parenthesized factor implies the minimization of the total power consumption. Consequently, the power dissipation cost (PDC) objective function for the CSC power model can be defined as $\text{PDC}_{\text{CSC}} = \text{SBT} + \text{CBTr} + 4 * \text{CBTg}$. Similarly, the PDC objective function for the SC model can be simply defined as $\text{PDC}_{\text{SC}} = \text{SBT}$. Minimizing the PDC function for each of the power models is the ultimate goal of the methodology that we propose in this paper.

Fig. 2(a) shows an example consisting of three bus lines and a sequence of four bits being sent on these lines. The total number of single line bit transitions SBT is equal to 5. For this example one can see that $\text{CBTr} = 3$ and $\text{CBTg} = 1$, resulting in the cost function being computed as $\text{PDC}_{\text{CSC}} = 5 + 3 + 4 * 1 = 12$.

In this paper, we propose a framework for low-power instruction encoding that targets both the CSC and SC power models. The SC model is the traditional power model utilized in most of the low-power bus encoding techniques found in the literature. The CSC model has appeared only recently as a viable power model for deep submicron buses as it targets not only single bit transitions but also the coupling transitions across adjacent bus lines. At the end of the paper we quantitatively evaluate the proposed methodology for both power models.

IV. LOW-POWER INSTRUCTION BUS ENCODING

A. Functional Overview

It is a well-known property of programs that they tend to spend most of their execution time within a relatively small part of the program code. This effect can be readily observed in virtually all multimedia applications, where most of the execution time is spent within a set of heavily utilized DSP kernels. Such heavily executed program fragments, also known as *application hot-spots*, are a natural candidate for optimization efforts. Furthermore, the various application hot-spots are not only quite tractable due to their limited size, but are also extremely independent, thus enabling locally applied and therefore inexpensive optimization techniques.

Consequently, we concentrate our efforts in terms of instruction bus power minimization on the few application hot-spots. The design flow of applying the proposed methodology is illustrated in Fig. 3. After the original program code is generated by the compiler/linker, an optimal application-specific low-power code is identified and subsequently applied on the program binary. After identifying the power efficient transformations for the stream of bits corresponding to each instruction bus line, the original program binary is encoded by using the optimal transformations. The transformed binary program is subsequently

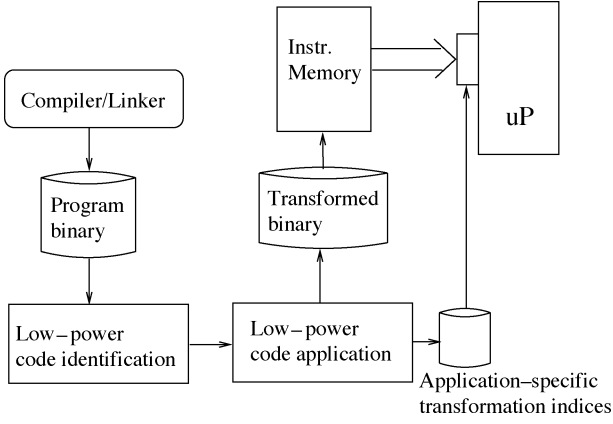


Fig. 3. Operational framework.

loaded into the instruction memory and fetched by the processor. A specialized hardware support on the processor side is responsible for restoring back the original code. The application-specific information regarding the sequence of applied functional transformations is transferred to the specialized hardware support and utilized during run-time in order to restore the original bit sequences.

The purpose of the power efficient instruction encoding proposed in this paper is to transform the bit sequences transmitted on the instruction bus, i.e., the program binary code, into new bit sequences so as to minimize the PDC. Fig. 2(b) presents an example in which the bit sequences on lines b_0 and b_1 from Fig. 2(a) have been transformed to new bit sequences prior to their transmission on the bus. One can observe that in this case, the cost $PDC_{csc} = 2 + 1 + 4 * 0 = 3$ is drastically reduced from its initial value of 12. While the PDC is determined as a function of *all* the bus lines, each line is transformed in an independent way, i.e., only the encoded bits from the same bus line are needed in order to restore the original bit sequence. Transforming independently the bit streams results in the ability to utilize extremely efficient recurrent functional transformations, which in turn leads to a very cost-efficient implementation of the circuitry responsible for restoring the original program.

B. Encoding Fundamentals

Let us consider an arbitrary sequence of bits, a *block word* $X = \{\dots, x_{n+3}, x_{n+2}, \dots, x_{n-3}, \dots\}$. We want to find an alternative sequence of bits, a *code word* $\tilde{X} = \{\dots, \tilde{x}_{n+3}, \tilde{x}_{n+2}, \dots, \tilde{x}_{n-3}, \dots\}$ and a transformation τ such that the PDC of \tilde{X} is minimized and $X = \tau(\tilde{X})$.

As instructions are fetched sequentially and execution speed is of utmost importance, the universe of the possible transformations needs to be restricted to transformations capable of restoring the sequence bit by bit, instead of decoding only upon receiving a complete block. While previous bits can be straightforwardly handled through buffering, consideration of future bits requires fetchahead, unnecessarily complicating the pipeline. The transformation τ should consequently be a function of the current bit and of a highly limited number, h , of history bits in the form of $x_n = \tau(\tilde{x}_n, x_{n-1}, \dots, x_{n-h})$. The number of history bits determines the total number of

transformations that can be utilized and thus need to be implemented as a part of the specialized hardware support at the processor front-end. Consequently, in order to control the hardware overhead associated with implementing the domain of possible transformations, the number of history bits h needs to be restricted.

1) *Transformation Classes*: The transformation classes that will be used for identifying low-power codes crucially impact both the transformation quality and the hardware overhead, necessitating their careful identification.

The first class of transformations, which we consider in this methodology are the transformations with a single history bit, i.e., $h = 1$, of the following form: $x_n = \tau(\tilde{x}_n, x_{n-1})$; we hereafter refer to this transformation class as single history bit (SHB) transformations. The SHB transformations use the currently received encoded bit together with the previously decoded bit in order to decode the current bit. As the transformations in this class depend on the encoded bit \tilde{x}_n , they are extremely flexible and can be applied to *any* bit sequence. They also embed the identity transformation $x_n = \tau(\tilde{x}_n)$, which enables the preservation at least of the original number of bit transitions if no better transformation can be identified for a particular bit sequence.

The second class of transformations that we utilize is of the form $x_n = \tau(x_{n-1}, x_{n-2})$ and are referred hereafter as double history bits (DHB) transformations. This latter transformation class represents a purely recurrent formula for computing the next element of a sequence, based upon the previous two elements given. When a DHB transformation is utilized, only the first two bits from the particular bit sequence need to be sent to the receiver, as the subsequent ones are to be “generated” by the functional recurrence. This results in at most a single bit transition for the block words for which this transformation can be applied. This transformation class achieves dramatic bit transition reductions for the block words for which it can be applied. Nonetheless, as it does not depend on an encoded bit, it is more restrictive in nature and can only be applied on bit sequences for which the recurrence can be satisfied.

2) *Code-Word Identification*: The low-power encoding methodology proposed in this paper utilizes objective functions as defined in Section III and aims at minimizing the corresponding PDC function. Given a bit sequence X of size k , the problem of finding a code word \tilde{X} that satisfies the particular objective function is defined as the problem of identifying a transformation τ and a word \tilde{X} of size k such that $X = \tau(\tilde{X})$ and \tilde{X} satisfies the objective function. The transformation τ has to satisfy either *one of the* following systems of binary equations defining the two classes of transformations:

For the *SHB* transformations

$$x_0 = \tilde{x}_0 \quad \forall 1 \leq i \leq k, x_i = \tau(\tilde{x}_i, x_{i-1})$$

and for the *DHB* transformations

$$x_0 = \tilde{x}_0; x_1 = \tilde{x}_1 \quad \forall 1 < i \leq k, x_i = \tau(\tilde{x}_{i-1}, x_{i-2}).$$

The process of finding a code word with a given property can be realized as a search process through the space of binary words of size k . In order to minimize the search time, an approach that eliminates the infeasible code word candidates and traverses the

feasible ones by starting with the most beneficial ones can be exploited. We employ such a strategy when identifying code words for the SC power model, as this model employs an objective function that naturally orders the binary words of fixed length. In this case, the code word candidates are considered by starting with the ones with the fewest bit transitions, which leads to quick identification of the optimal code words. In general, at each iteration of the search process, the word being considered is tested as to whether it satisfies the associated objective function; if so, the transformation τ , for which the corresponding system of equations is solvable, is identified.

3) *Illustrative Example:* We attempt to illustrate the proposed process by providing an example focused on the simpler of the two minimization objectives studied, that of SBT. We start the search process by analyzing for a given initial bit sequence X a candidate encoded bit sequence \tilde{X} starting with zero-bit transition candidates, then proceeding with one-bit transition and so on, until a sequence that satisfies the transformation equations is encountered.

Let us assume that the original bit sequence is 010. An initial candidate is the 111 word as it exhibits zero-bit transitions. However, neither class of transformations is able to map the word 111 to the original word 010 simply because the first equation in both systems, namely $x_0 = \tilde{x}_0$, would be violated. When we subsequently consider the word 000, which also exhibits zero-bit transitions, it can be shown that there does exist a transformation that maps it to the original bit sequence. Mapping the word 000 to the block word 010 through a *SHB* transformation implies that a boolean function τ must exist, such that the equations defining this class of transformations are satisfied. That is the middle bit 0 of the possible code word 000 and the rightmost bit of the block word 010 produce the middle 1 of the block word, i.e., $\tau(0, 0) = 1$, while the leftmost bit 0 from the code word and the middle bit 1 of the block word generate the leftmost bit 0 of the block word, i.e. $\tau(0, 1) = 0$. Evidently, the transformation $\tau(x, y) = \bar{y}$ is such a transformation. Consequently, the original word 010 can be obtained from the encoded word 000 by utilizing the transformation $\tau(x, y) = \bar{y}$ of the *SHB* type.

It is possible that multiple transformations exist of either the *SHB* or *DHB* types that satisfy the corresponding systems of equations. Let's, for instance, consider now the block word 1011 and the code word candidate 1111. The two equations that utilize the rightmost two bits of the code word 1111 and restore the middle two bits of the block word for a *SHB* transformation $\tau(1, 1) = 1, \tau(1, 1) = 0$ constitute an evidently contradictory set of constraints! Yet, the *DHB* class of transformations does provide a solution, since the set of equations $\tau(1, 1) = 0, \tau(0, 1) = 1$ can be satisfied by a number of boolean functions, namely $x \oplus y, x', x' + y'$, and $x'y$. Furthermore, it is even possible for some block words to have multiple code words, as the objective function may be satisfied by multiple code words utilizing either the *SHB* or the *DHB* class of transformations. This fundamental property of multiple solutions is actively exploited in our methodology for low-power code generation as this degree of freedom helps to drastically minimize, at no adverse effect on code quality, the total number of transformation instances per application hot-spot.

V. METHODOLOGY DESIGN FLOW

Since the proposed technique is applied on the application hot-spots, when searching for optimal code words only the bit streams within the particular application fragment need to be considered. In this way, a small set of block words will be extracted for each application hot-spot that is targeted by the low-power encoding. Subsequently, the corresponding set of code words and the set of transformations associated with it have to be identified.

In order to be able to decode the original bit sequence, the special hardware support on the processor side needs to know which transformation is associated to each block word. Consequently, information regarding the transformation instance for each block word needs to be communicated to the specialized hardware support. This information is communicated prior to entering the hot-spot, thus introducing no performance overhead in practice. However, the amount of this information determines the area and power consumption of the specialized hardware. It is, therefore, of paramount importance to minimize this amount, while achieving significant reduction of bit transitions. Consequently, the additional constraints of minimization of the total number of transformations that are to be used for the particular application hot-spot and of the cardinality of the set of block words have to be imposed in generating the encoding.

Both classes of transformations are very efficient to compute, since they correspond to simple binary functions of two variables. The total number of such logic functions is $2^{2^2} = 16$ for each of the two transformation types, but as will be discussed subsequently, a much smaller subset of all the transformations capable of achieving significant bit transition reductions would be identified for a given bit stream. Such a transformation window minimization is performed in order to reduce the overhead associated with the storage requirements on the processor side where transformation indices need to be stored and used for applying the transformation. While the decoding architecture will support all 32 distinct functionals, only the active small subset of transformations identified for the particular application hot-spot would be utilized at any instance.

A. Block-Word Extraction

As the cardinality of the candidate set of block words is one of the determining factors for both the total number of transformations needed and, therefore, the complexity of the hardware support, a strategy for effectively reducing the numbers of block words leads to reduced hardware overhead. The block size is the factor that directly determines the total number of block words, with larger block sizes leading to a reduced number of block words. Yet, the larger block words can potentially result in reduced power benefits, as the number of equations that define both transformation classes is equal to the number of bits, reducing in a statistical sense the ability to identify an efficient code word.

In any case, the control-flow limitations of any program naturally put bounds on the block word size. When the program execution crosses basic block boundaries, i.e., in executing a branch instruction, the particular execution path to be followed cannot consistently be known at compile time. Therefore, all the block words must be confined to within basic blocks. This consideration imposes a natural limit on the size of the block words.

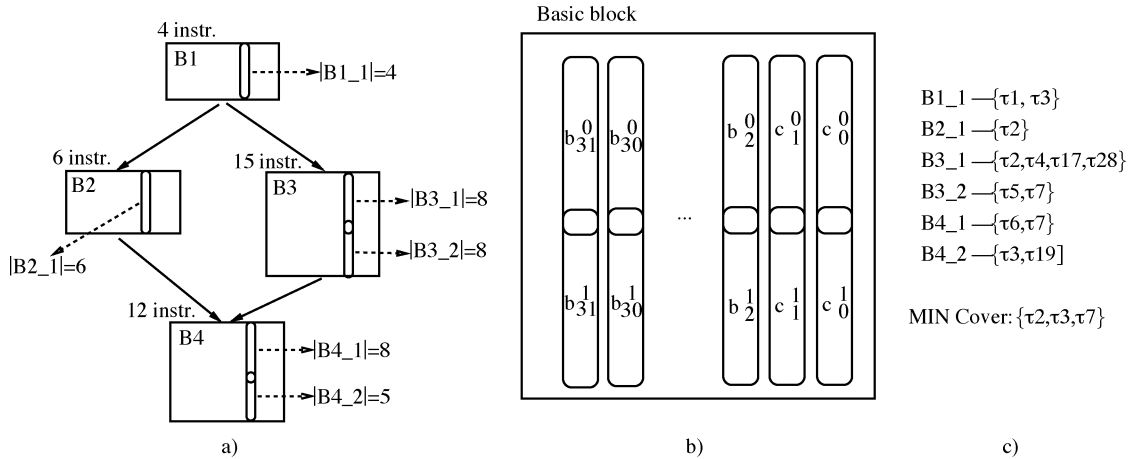


Fig. 4. Block-word extraction and code word generation.

A straightforward strategy for block word extraction, therefore, is to have a predefined threshold of a maximal block size. In this case, all basic blocks containing fewer instructions than the predefined threshold will be covered by a single block word. The remaining basic blocks are split into block words of size equal to the prespecified threshold. This strategy leads to a set of block words with lengths varying up to the prespecified block-word length. In a subsequent section of this paper we report experimental results for various thresholds on block word sizes.

An important consequent issue that needs to be addressed is the bit transitions across block-word boundaries. If blocks are defined to be disjointed, no improvement in bit transitions can be effected for the bits straddling the block. Overlapping blocks suffer no such limitation but impose, on the other hand, an additional constraint on one of the blocks, subsequently reducing the potential efficacy of the transformations. Since the overlapped bits would be determined by the first code word, the subsequent code word must preserve them, which imposes a constraint on the code word search algorithm. Consequently, in order to minimize this impact, an overlap of a one-bit position only needs to be considered. If, for example, the block size is to be fixed to four bits, then the sequence X is split into two blocks $X_1 = (x_{n+3}, x_{n+2}, x_{n+1}, x_n)$ and $X_2 = (x_n, x_{n-1}, x_{n-2}, x_{n-3})$ with bit x_n belonging to both groups. It is possible that distinct transformations are assigned to X_1 and X_2 .

Fig. 4(a) illustrates the aforementioned block word extraction strategy. The example shows an application hot-spot that contains four basic blocks, B1–B4. The lengths, i.e., instruction numbers, for these basic blocks are given as well. A maximum block word size of 8 has been assumed here and the set of all block words for particular bit positions has been extracted. Since basic blocks B1 and B2 contain fewer than eight instructions, they have been covered by a single block word, B1_1 of size 4 and B2_1 of size 6, respectively. The bit sequences within basic block B3 have been split into two block words each of size 8, while the bit sequences within basic block B4 have been split into two block words of size 8 and 5, respectively.

B. Code-Word Generation

The subsequent phase of the proposed methodology is the code-word generation phase. By this stage, the set of block words has

been identified and depending on the power model under consideration, different approaches for identifying the corresponding optimal code words needs to be employed. In the case of the SC power model, the code word search algorithm presented in the previous section is simply applied on each block word. This procedure is applied on all the bus lines in an independent way as the code word selection step depends solely on the block word under consideration. On the other hand, as the quality of a particular code word for the CSC power model depends on the code words on the two neighboring lines, the problem of assigning code words becomes a multiconstrained optimization problem. In order to avoid the exponential search space, we utilize for the CSC model an iterative algorithm that sweeps on the bus lines and assigns the best code words for a line given the previously assigned code word for the adjacent line.

Fig. 4(b) shows a basic block for which the block words for the first two bus lines have already been transformed into code words and the algorithm is to start with the third bus line. Since the primary objective consists of the minimization of both the single bus line bit transitions and the transitions on the adjacent lines, the code word search algorithm has to employ as an objective function a predicate that considers these effects. Consequently, the cost function to be utilized at each iteration of the algorithm has to be computed by considering the current candidate for code word and the already computed code word on the previous bus line.

Let us consider the block word b_2^0 from the example in Fig. 4(b). When searching for an appropriate code word c_2^0 , the neighboring code word c_1^0 generated on the previous iteration of the algorithm is to be considered for PDC minimization. The code word search algorithm is then iteratively executed until it successfully finds code words. In this way, the set of code words with minimal PDC with respect to the code word on the adjacent bus line is identified. Subsequently, the algorithm proceeds with the next block word.¹

For both the SC and the CSC power models, each block word can be associated with more than one optimal code word, as

¹The handling of the first column in the algorithm exhibits particularities due to the absence of a neighboring column to consider in the CSC model. This can be handled either by applying only the SC model for the first column or by assuming a special neighboring preinitialized column and applying the CSC model.

was explained earlier. Furthermore, each code word could be associated with more than one transformation, as the system of equations that define the transformations might be satisfied by multiple boolean functions. Since all the optimal code words have the same cost in terms of bit transitions, the code word selected for the particular block word is of no importance. Fig. 4(c) lists an example set of transformations associated to each block word. For instance, block word B4_1 is associated with transformations τ_6 and τ_7 , implying that either transformation maps an optimal code word to the block word B4_1.

C. Transformation Minimization and Bounding

The final step of the low-power code generation algorithm consists of the minimization of the number of transformations that will be used for the given application hot-spot. Since the application specific information regarding the mapping between a block word and the transformation associated to it has to be available to the processor front-end for restoring the original block words, the total number of transformations used to encode a particular hot-spot is a significant factor in determining the complexity of the specialized hardware support. For instance, if a total of four transformations are to be used, then these transformations can be identified with 2 bits only. Of course, an additional special hardware mechanism would be needed to restrict access to only these four transformations from the pool of all available 32 simple boolean functions. The principles and architecture of this transformation subset selection mechanism are described in Section VI; we focus in this section on discussing the algorithmic foundations.

Fundamentally, the problem of finding the minimal number of transformations that covers all the block words is the well known *minimal set cover problem*. Consequently, we are searching for a subset of transformations such that each block word is covered by at least one transformation from that set. The minimal cover problem is known to be NP-hard and a large number of efficient heuristics have been developed and documented [20]. To ensure the identification of the minimal cover we have used exhaustive search, where even for the largest block words in our experiments, the algorithm completes within a second. For the example in Fig. 4(b), the minimal cover consists of the three transformations (τ_2, τ_3, τ_7) only.

As efficient hardware solutions necessitate support only up to a prespecified number of transformations per application hot-spot, a method is needed to further *bound* the number of transformations in the cases when the minimal cover exceeds the threshold. At this step the low-power code generation algorithm eliminates the block words of the less frequently executed basic blocks. Typically, a number of infrequently executed basic blocks exist for any hot-spot and correspond to pieces of code for handling error conditions or other rare events. A further selection criterion for basic block elimination is the achieved bit transition reduction; basic blocks with minimal contribution to total bit transition reduction can be selected for elimination at this stage. Subsequent to this basic block selection step, the minimal transformation cover for the remaining set of block words is computed. This iterative process continues until the minimal cover consists of a set of transformations whose cardinality does not exceed the hardware allowed upper bound.

A final step in this transformation bounding step is to identify a policy of how to handle the discarded basic blocks. A straightforward approach is to leave them intact, which is equivalent to utilizing the identity transformation for all the discarded blocks. A preferable approach that we adopt though is the exploitation of the transformation set associated to the transformed basic blocks followed by the identification of new code words for the discarded basic blocks. These new code words, even though not optimal, result in insignificant deviations from optimality typically, as they deliver quite considerable bit transition reductions even for the discarded block words. Consequently, bounding the cardinality of the minimal transformation cover does not impact the low-power code quality in practice. This proposition is experimentally verified and a set of data confirming its validity presented in Section VII.

VI. HARDWARE SUPPORT

A. Architectural Overview

The hardware support for implementing the proposed application-specific low-power encoding technique needs to be able to identify the transformations associated with the current bit sequences and apply them in order to restore the original program opcodes. In order to perform instruction decoding, the processor's fetch unit needs to capture the order in which the code words arrive from the instruction memory and assign a transformation to the currently active blocks on all the bus lines.

An important aspect in designing the hardware architecture for supporting the proposed low-power customization technique is the requirement of achieving a reprogrammable implementation. The architecture needs to be able to utilize the application-specific low-power encoding, i.e., the sequence of transformations selected for the particular application loop. Consequently, a mechanism is needed for transferring this information to the fetch engine of the processor core before executing the application loop. It is evident that the support architecture needs to contain tables with the information about the low-power transformations being selected and their usage order. Two alternatives can be implemented as a possible solution. The first one consists of loading the content of these tables at the same time as the application code upload to the instruction memory. This approach is particularly suitable for firmware applications, as their code changes infrequently. The second alternative is to have the application-specific information transferred by software as all the tables containing the power transformation information can be easily made accessible as memory mapped I/O, for example. An insignificant volume of information is needed, since it corresponds only to application hot-spots, and can be easily written to this memory by a set of instructions inserted within the application code and executed just prior to entering the loop under consideration.

B. Hardware Architecture

The hardware architecture of the proposed implementation is presented in Fig. 5. The transformation table (TT) stores transformation indices associated to each block word. An entry in the TT contains the set of transformations for a particular block word on all the bus lines. The TT table can be easily implemented as a small SRAM array with a very limited number of entries. A TT entry, as

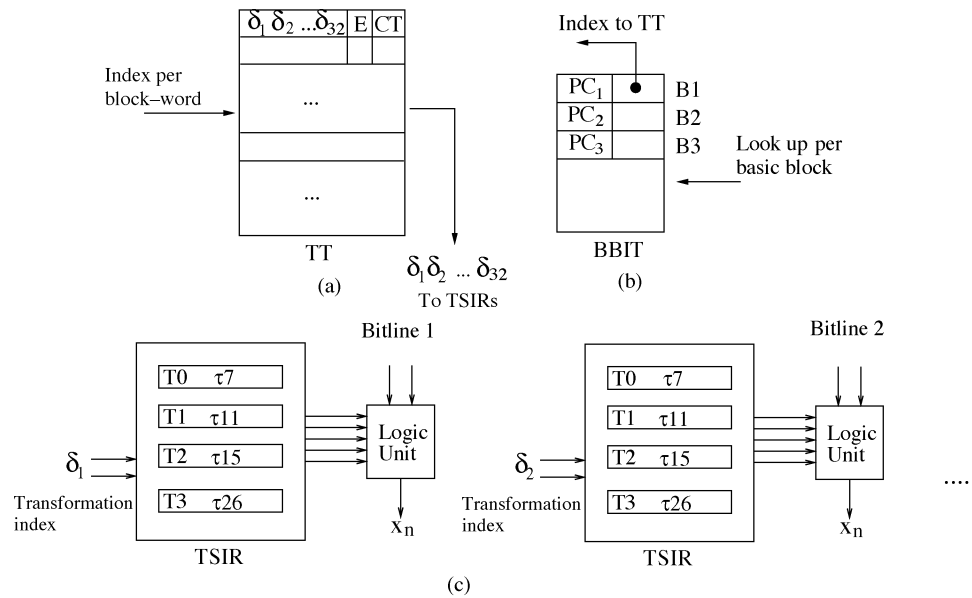


Fig. 5. Hardware architecture illustrated on a loop code.

shown in Fig. 5, contains the control bits for selecting the transformation associated to each bit sequence. The fields for the control bits in the figure are denoted with δ_i . For instance, if the number of distinct transformations per bit line is limited to four transformations, the δ_i fields consist of 2 bits only. Each TT entry contains as many δ indices as there are instruction bus lines. For instance, a processor with a 32-bit instruction bus results in a TT entry containing 32 transformation identifying indices. As the low-power encoding cannot span through the application loop basic blocks, a contiguous *set* of entries in the TT is allocated for each basic block targeted. A TT entry contains the transformation information needed to handle a single block word for each bus line. Consequently, the number of TT entries determines the number of block words that can be handled per application hot-spot. The last TT entry for a particular basic block must contain information about how long the last code word is. The *End* (E) bit field in the TT entry is asserted for the entries that correspond to the block word for a given basic block. The final field, denoted by CT, is a counter corresponding to the size of the last bit sequence; it is only read for the cases in which the E field is set. Once this entry is utilized, the value of the counter is decremented with each instruction fetched and decoded; a zero would indicate that the sequence of the tail instructions for the basic block has been completed.

Fundamentally, the E bit is a delimiter, which separates the set of entries in the TT corresponding to an application loop basic block. Once a basic block from the control flow graph (CFG) is executed, information is needed about the next application basic block to be executed. More specifically, an index into the TT is required so that the decoding transformations for the subsequent block of instructions can be identified. In order to achieve this, the basic block identification table (BBIT), shown in Fig. 5(b), is introduced. The number of entries in this table corresponds to the number of CFG basic blocks for the particular application loop. Each entry contains the program counter (PC) of the starting instruction together with an index into the TT. This index points to the first entry in the TT for this basic

block. Therefore, once a basic block is completed, a lookup into the BBIT is needed in order to produce the TT index for the next basic block. An additional information stored in the BBIT entries is the block word size for each basic block. When the program execution proceeds to a subsequent basic block, the block word size is stored into a global counter, whose purpose is to keep track of the block word length and consequently to identify when a new entry from the TT needs to be read when a new block word is to be started.

As each application hot-spot utilizes a small subset of all the available functions, a reprogrammable hardware mechanism is needed so that the selected subset of transformations can be accessed efficiently through minimal indices. Fig. 5(c) shows the proposed hardware architecture for this mechanism. The transformation subset identification registers (TSIR) module is proposed as a means of selecting a predetermined subset of transformations. The TSIR contains a set of five-bit registers. The values stored in these registers are used as control signals for selecting any one of the supported 32 boolean functions, which can be easily implemented in the form of a *universal logic unit*. Even though the logic unit supports all 32 transformations, it only needs to implement the set of all 16 boolean functions of 2 input variables; each 2-input boolean function can be used to implement the corresponding SHB and DHB transformations by hooking appropriately the (\tilde{x}_n, x_{n-1}) and the (x_{n-1}, x_{n-2}) pairs of bits. The number of registers implemented within the TSIR represents the maximal number of transformations that can be used per application hot-spot. The example architecture presented in Fig. 5(c) supports up to four transformations. The transformation indices selected for a particular program hot-spot are stored into these registers prior to entering the loop. Subsequently, each of the selected transformations is encoded as a short index, in the case of Fig. 5(c), two bits, which is stored in the TT table to represent the specific transformation. The index is used to select the TSIR register, which in turn produces the actual five bit transformation index.

C. Overhead Analysis

It is evident that the hardware overhead for the proposed customization methodology is determined by the size of the TT and BBIT arrays as well as the TSIR structure. The power overhead of this hardware architecture is determined not only by the size of the tables being accessed but also by the frequency of these operations.

The TSIR logic consists of four or eight registers depending on the maximal number of transformations that will be supported per application hot-spot. The values of these registers are stored before entering the hot-spot and remain constant during the hot-spot execution. A power efficient implementation of the logic unit consists of implementing the 16 different boolean functions by combining the active minterms, while gating, upon entering the hot-spot, the minterm inputs that are not included in the currently selected subset of transformations. In this way, the only activity at any point within the logic unit will be on the few gates that implement the corresponding transformation, which in terms of power consumption is negligibly small even in comparison to a single bit transition on the long instruction bus lines, as we show subsequently in this section.

The number of entries in the TT determines the total number of block words per application hot-spot that can be handled by the proposed methodology. Since an entry in the TT is allocated per block word, the longer the block-word size, the fewer the number of entries in the TT need to be. In Section VII, we show how many block words for various block lengths have been extracted for the benchmarks we have used. It can be seen that a TT table with 32 entries suffices for all practical purposes. The consequent hardware overhead is insignificant as these two extremely small tables can be accessed quickly with negligible amount of additional power spent in performing these accesses. Moreover, the TT table is accessed only in the beginning of a block word, which further diminishes its power overhead.

The number of BBIT entries determines the number of basic blocks that can be handled per application hot-spot. Our experience with embedded applications tells us that the majority of program hot-spots are essentially tight loops that contain basic blocks well within the range of 16–32. The eight benchmarks that we have utilized in our experimental studies confirm this observation. Consequently, a practical design of the BBIT table consisting of 16, and up to 32 entries can cover any hot-spot. As the BBIT table is accessed only when a new basic block is to be executed, its associated power overhead is even further reduced. If function calls within the loop exist, the function code is treated as a part of the loop and utilizes the low-power encoding, if the total number of application basic blocks can be accommodated in the BBIT. In the highly unlikely case where a basic block number higher than the number of BBIT entries is encountered, the loss in power-reduction capabilities is solely restricted to the overflowed blocks. Alternatively, the whole basic block set can be handled by employing an extended BBIT implementation with banked structure where only the required minimal number of subbanks are activated to cover the particular set of basic blocks.

To quantitatively evaluate the introduced power overhead, we have utilized the Cacti tool [21] to obtain the energy dissipation associated with the TT and the BBIT tables. A 0.18- μm process technology with 1.7-V voltage level has been used. We have

found that an access to the TT table dissipates approximately 12 μJ of energy while an access to the BBIT table dissipates 29 μJ . The power consumption overhead of the selected transformation τ logic is negligibly small and amounts to 0.038 μJ . As the TT table is read once per code word on *all* 32 bus lines and the BBIT table is read only once per basic block, the power consumption numbers associated with the TT and the BBIT tables have to be divided accordingly. Assuming three block words per basic block, the total power overhead associated with one code word on all the bus lines can thus be computed to amount to 21.7 μJ . Under the rather conservative assumption of 15 pF capacitance,² the overall power overhead of the technique we introduce comes out to be equivalent to what can be saved by reducing a single-bit transition on the entire bus. Considering the large number of transition reductions delivered by our technique, the dramatic overall power savings can be easily seen. To quantitatively represent the power overhead, one can see that if we follow the conservative assumption that the power overhead is equal to the power consumption for a single bit transition, we can see that the power overhead is $1/16 = 6.25\%$ where 16 is the average number of single line bit transitions. The conservative nature of this estimation can also be seen from the fact that the reduction in coupling transitions are not taken into account. As our technique reduces a significant portion of the bit transitions on *all* the bus lines, the power overhead of the specialized hardware support is insignificant in practice. The area overhead of the proposed hardware support is even more insignificant as the introduced transistors, estimated to be in the range of 2000–4000, constitute a microscopic fraction of the real estate budget of any modern embedded processor typically amounting to several million transistors.

In terms of performance overhead, it is important to notice that indexing the TT table, looking up the BBIT table, and indexing the TSIR registers, is outside the critical path as all these events happen only once at the beginning of code word or once at the beginning of basic block (in the case of the BBIT). All these activities can be performed during the cycle prior to their effective use and their results latched appropriately. The only delay that needs to be tolerated is the delay incurred by the logic unit implementing the transformation classes. As these transformations are essentially all 16 boolean functions of 2 variables, the associated delay corresponds to the delay exhibited by a two level sum-of-products logic consisting of four product terms. Furthermore, this small delay can be easily overlapped with any other logic in the processor front-end that operates with the PC as input, such as PC increment, branch target buffer (BTB), or branch prediction logic.

Even though such a detailed hardware overhead analyses is typically lacking in the previously published papers on bus-encoding techniques in the literature, our estimation shows that our hardware and power overhead is analogous to that in [6] and [17] and much smaller than that of [10] and [11]. The general-purpose nature of most of the previous bus encoding techniques [10]–[12], and [17] requires the support of complex data structures and control in hardware in order to dynamically identify

²As reported in [9], the average bus line capacitance can go up to 30–50 pF, which diminishes the impact of the introduced overhead even further.

TABLE I
NUMBER OF BLOCK WORDS

block size	fdct	mmul	sor	tri	ej	lu	fft	rand
9	14	8	8	22	18	24	10	24
8	14	8	9	26	19	26	10	32
7	16	9	9	30	21	28	12	32
6	18	10	10	33	24	29	15	40
5	18	11	11	39	29	33	16	40

the suitable bus encoding. Furthermore, all of these techniques necessitate a separate encoder and decoder circuitry on both the processor and memory side of the bus, while our methodology requires no additional hardware on the memory side. The existence of encoding hardware in the previous bus encoding approaches not only exacerbates the hardware and the associated power but also introduces additional delay within the extremely critical path of instruction or data memory access.

VII. EXPERIMENTAL RESULTS

We have conducted a set of experiments in order to quantitatively evaluate the utility of the proposed application-specific low-power codes. The algorithm for low-power code generation was implemented as a stand-alone tool. This tool reads in a special data file that contains profile and static post-compile information regarding all the hot-spots for the particular application benchmark. This information, generated by a modified version of the SimpleScalar toolset [22], includes the control-flow graph for each hot-spot together with the list of the binary representations of the instructions. It also contains the execution frequencies for all the basic blocks, to be utilized by the application-specific low-power code-generation algorithm as described earlier in the paper.

For our experimental study we have used a set of eight benchmarks. This set contains various DSP and numerical kernels, which are heavily utilized in a multitude of embedded products. The following application benchmarks were used: fast discrete cosine transform (fdct) DSP kernel; matrix multiplication (mmul) on a matrix of size 50×50 ; successive over-relaxation (sor) on a matrix with size 128×128 ; a tridiagonal linear system solver (tri) on a matrix of size 128×128 . Extrapolated Jacobi-iterative method (ej) on a 128×128 grid; *lu-decomposition* (lu) algorithm on a matrix of size 128×128 ; and fast Fourier transform (fft) with block size of 256 samples. An additional “benchmark,” denoted by *rand*, is added to the set with the purpose of evaluating the performance of the proposed technique on a stream of random data. This benchmark consists of eight “basic blocks” each containing 25 randomly generated 32-bit words.

Table I presents data regarding the number of block words extracted for each benchmark. Each row corresponds to a different block word threshold as outlined in Section V-A. The block size that we experimented with was in the range of 5–9. It can be observed that the total number of block words is quite limited for all the benchmarks and as to be expected, this number consistently decreases as the block size is increased. We do not report on sizes smaller than 5 as decreasing further the block word size would lead to impractically large number of block words to be supported thus making the hardware overhead in terms of TT size unacceptable.

TABLE II
SC MODEL PDC REDUCTIONS (%) WITH NO τ LIMIT

block size	fdct	mmul	sor	tri	ej	lu	fft	rand	AVG
9	22.6	14.0	13.8	28.4	40.0	24.0	36.5	43.6	27.9
8	24.6	15.1	28.0	35.8	41.1	34.1	40.7	46.5	33.2
7	50.4	26.9	33.7	41.1	45.9	25.9	44.9	52.7	40.2
6	72.4	57.1	74.2	54.9	64.3	41.7	53.2	64.0	60.2
5	78.1	73.9	77.1	79.2	83.9	76.0	86.2	83.8	79.8

TABLE III
CSC MODEL PDC REDUCTIONS (%) WITH NO τ LIMIT

block size	fdct	mmul	sor	tri	ej	lu	fft	rand	AVG
9	34.0	29.9	29.4	35.2	40.1	36.4	40.0	42.4	35.9
8	33.3	35.5	38.3	46.8	48.1	39.6	42.1	43.5	40.9
7	45.0	35.4	37.7	42.1	52.0	34.5	45.4	51.9	43.0
6	62.2	58.6	61.0	60.1	65.4	49.9	48.6	59.3	58.1
5	62.8	68.5	69.3	76.7	74.3	69.3	72.9	76.3	71.3

TABLE IV
SC MODEL PDC REDUCTIONS (%) WITH LIMIT OF 8 τ 's

block size	fdct	mmul	sor	tri	ej	lu	fft	rand	AVG
9	22.6	14.0	13.8	28.4	40.0	24.0	36.5	34.3	27.9
8	24.6	15.1	28.0	35.8	41.1	34.1	40.7	37.4	33.2
7	50.4	26.9	33.7	41.1	45.9	25.9	44.9	48.4	40.2
6	72.4	57.1	74.2	54.9	64.3	41.7	53.2	54.4	60.2
5	78.1	73.9	77.1	79.2	83.9	76.0	86.2	59.1	79.8

For instance, in the case of block words of size 4, the Transformation Table limited to only 4 transformations per bit line would have a size equal to 50% of the hot-spot code size targeted for power reduction. Furthermore, the time complexity for identifying the minimal set of transformations and bounding them below a certain threshold increases drastically.

Most of the previous work [7], [8], [10], and [11] in the area of bus encoding has focused either on address buses or data buses where a particular data regularity has been assumed and exploited. Of the work that assumes no data regularity but rather a uniform value distribution, i.e., the bus invert method [6] and its derivatives [12] and [13], maximum bit transition reductions in the range of 10%–15% have been reported, which are well below the reductions obtained by our methods even for the worst cases. Given the clear power reduction superiority of the approach we propose, we focus in this section on reporting the detailed results that our techniques achieve and analyze the impact of the various parameters, such as block size and transformation bounds, on the total obtained bit transition reductions.

Tables II–VII show the bit transition reductions in percents for the proposed techniques for the cases of no limit on the number of transformations (effectively 32), and upper bounds of 8 and 4 transformations, respectively. Each table entry contains a number corresponding to the percentage improvement for the corresponding power model. The rows of the tables correspond to block word lengths ranging from 5 to 9. The last column in all the tables shows the average reduction for all the eight benchmarks. As can be expected, the shorter the maximal block word size, the higher the improvement in the transition reductions, since the code word search algorithm finds better code words for shorter block sizes; as the system that needs to be satisfied by the transformation candidate contains fewer equations, it has more degrees of freedom in selecting transformations and thus delivers code words with fewer bit transitions.

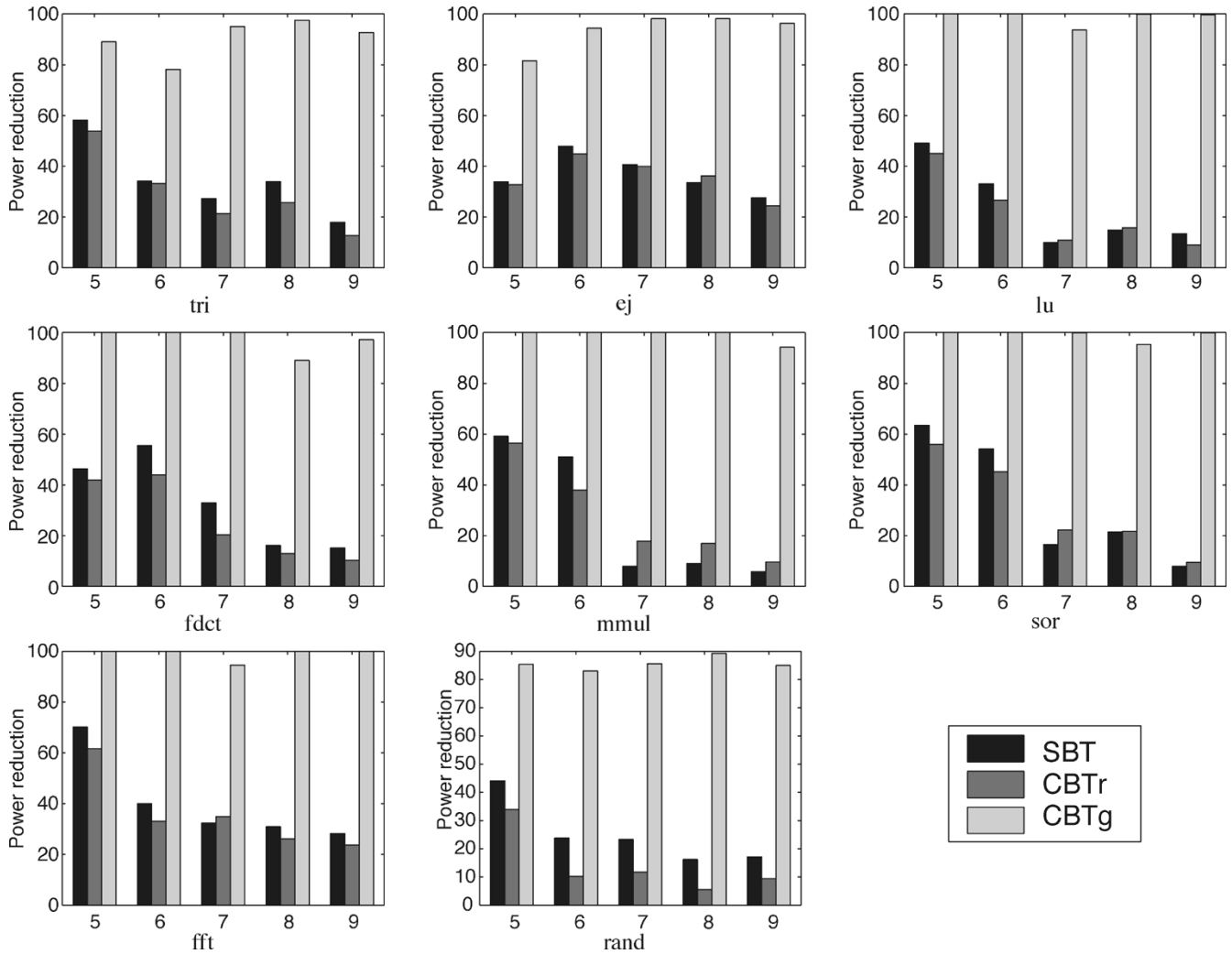

 Fig. 6. SBT, CBTr, and CBTg reductions in percentage with limit of four τ 's.

 TABLE V
 CSC MODEL PDC REDUCTIONS (%) WITH LIMIT OF 8 τ 's

block size	fdct	mmul	sor	tri	ej	lu	fft	rand	AVG
9	34.0	29.9	29.4	35.2	40.1	36.4	40.0	42.4	35.9
8	33.3	35.5	38.3	46.8	48.1	39.6	42.1	43.3	40.9
7	45.0	35.4	37.7	42.1	52.0	34.5	45.4	50.7	43.0
6	62.2	58.6	61.0	60.1	65.4	49.9	48.6	54.6	58.1
5	62.8	68.5	69.3	76.7	74.3	69.3	72.9	74.2	71.3

 TABLE VI
 SC MODEL PDC REDUCTIONS (%) WITH LIMIT OF 4 τ 's

block size	fdct	mmul	sor	tri	ej	lu	fft	rand	AVG
9	22.6	14.0	13.8	27.5	40.0	23.9	36.5	29.7	26.0
8	24.6	15.1	28.0	32.7	29.5	34.0	40.6	26.5	28.9
7	50.4	26.9	33.7	35.0	34.8	25.5	44.8	29.8	35.1
6	72.3	57.1	74.2	41.3	34.2	41.4	53.1	31.8	50.7
5	73.1	73.8	77.1	40.7	44.8	71.4	86.0	33.5	62.6

It can be seen that for four of the benchmarks, namely *fdct*, *mmul*, *sor*, and *lu*, the optimal bit-transition reductions for block sizes larger than 5 can be achieved by using at most four transformations. Allowing eight transformations or even the absence of any limitations on the number of transformations delivers no power reduction improvements. The insensitivity to the increase in the number of transformations can be accounted for by the

 TABLE VII
 CSC MODEL PDC REDUCTIONS (%) WITH LIMIT OF 4 τ 's

block size	fdct	mmul	sor	tri	ej	lu	fft	rand	AVG
9	34.0	29.9	29.4	34.2	39.8	36.3	40.0	37.0	35.1
8	33.3	35.5	38.3	46.2	47.5	39.5	42.1	36.9	39.9
7	44.9	35.4	37.7	41.6	51.6	34.1	45.4	40.1	41.4
6	62.2	58.3	61.0	44.6	55.8	49.7	48.5	39.0	52.4
5	58.2	68.5	68.8	64.0	42.8	62.0	72.3	54.4	61.4

relatively smaller number of block words extracted for these benchmarks. For the remaining four benchmarks, increasing the upper bound on the transformations to 8 leads to some additional improvements, in some cases by more than 10%. Nonetheless, for all of the benchmarks it can be observed that a set of eight transformations always suffices to achieve the minimal bit transition reductions. Allowing excess transformations contributes no improvement for any of the application benchmarks. The direct relation between block word size and the quality of the encoding can be readily observed as well.

As the proposed low-power bus encoding technique requires storage for capturing the transformations associated to each block word, an important question that needs to be answered is regarding the ratio of the stored information versus the original code size. For each block word, a small index that

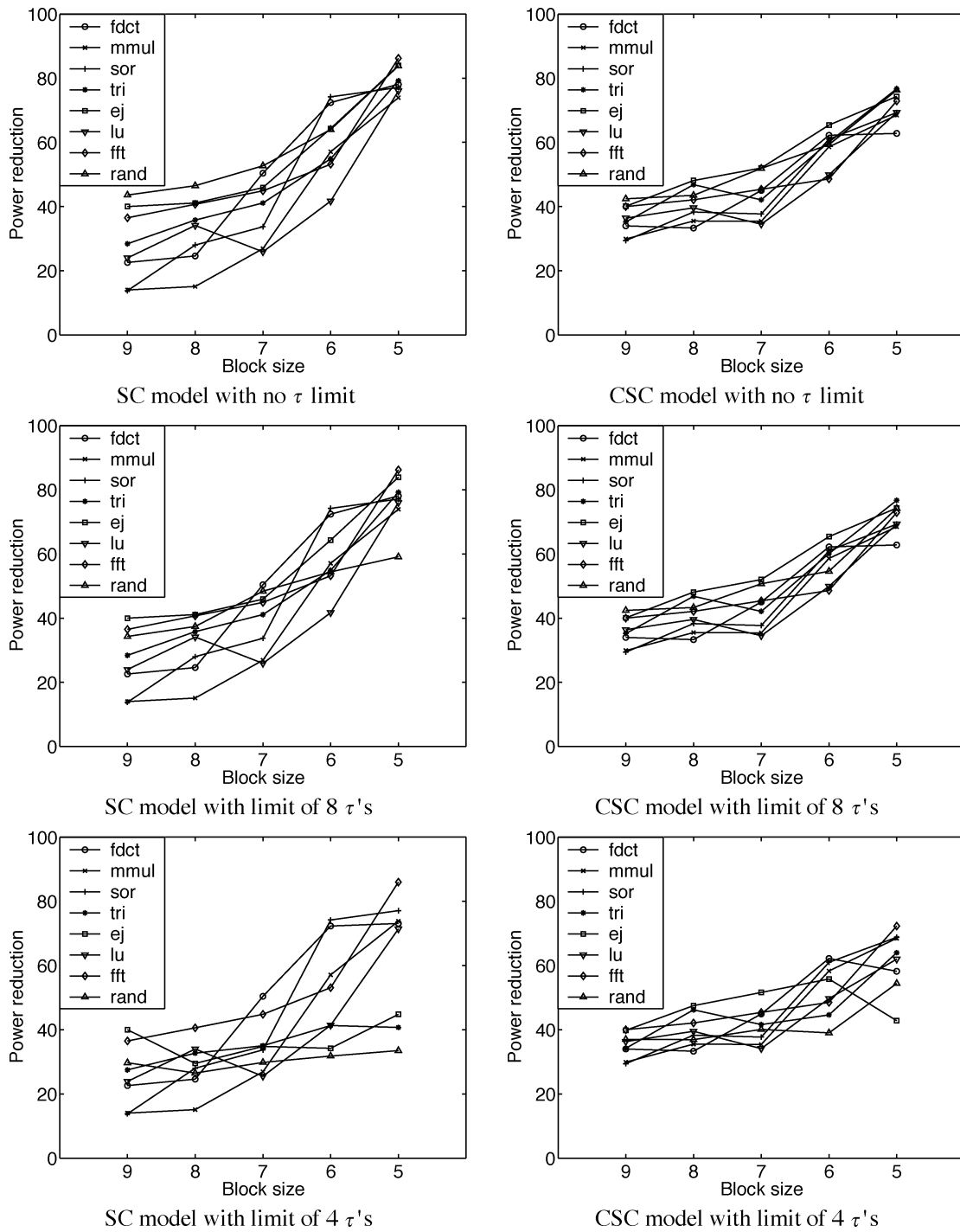


Fig. 7. Block-word size impact on the power reduction.

represents the transformation associated to its corresponding code word is stored in the TT, as elaborated in the previous section. Depending on the number of distinct transformations allowed per bit line, the size of these indices can be 2 or 3 bits for four or eight transformations, respectively. Given the block word size, the ratio between stored information and original code size can be easily computed. For instance, in the case of block word size of 6 and assuming a bound of 4 on the number of distinct transformations per bit line (as shown in the experimental results to achieve near optimality), it can be seen that this ratio is 2/6. One can easily observe that the proposed technique needs to store only a small amount of information as

compared to an I-cache, which would have to store the entire code and furthermore incur the extra overhead of tag operations and higher associativity as well.

Table VI shows the percentage reductions of the power dissipation cost function components of single line bit transitions (SBT), cross line bit transitions (CBTr), and cross line bit toggles (CBTg). Since the weight of *CBTg* is highest, it is to be expected that this component will be the one with the largest reduction. The component reduction for the architecture with the limit of 4 τ 's per bus line is shown herein, as this configuration is the most practical case that achieves near optimal results with extremely small hardware overhead. Table VI consists of eight

bar graphs, one for each benchmark, comparing the reduction for the three PDC components for the cases of block words of sizes 5, 6, 7, 8, and 9. The black bar corresponds to the SBT component, while the dark gray and the light gray bars show the percentage reduction of the CBT_r and the CBT_g components, respectively. As can be seen for these graphs, the most important component of the power dissipation, CBT_g, is consistently reduced by more than 95%; in more than half of the cases seen here, the CBT_g reduction is actually 100%. It is also evident that for shorter block sizes the reductions for SBT and CBT_r are relatively higher compared to the longer blocks. This observation is easily explained by the fact that for shorter block words there exist more degrees of freedom when searching for optimal transformations as fewer constraints are imposed on the transformation.

Fig. 7 presents a detailed comparison study regarding the correlation between block word size and power reductions. For all the benchmarks, we have plotted the achieved bit transition reduction as a function of the block size. This comparison has been performed for both power models and for the three cases of bounding the total number of transformation. For all the cases, one can observe the steady tendency of improved bit transition reductions as the block-size is decreased. Table VII furthermore illustrates that the improvement trends for the various block sizes for the CSC model are more consistent across all the benchmarks as the optimization function for this model relies on three distinct components, thus having more degrees of freedom in consistently obtaining “good” code words.

VIII. CONCLUSION

In this paper, we have presented a framework for bit transition minimization for deep submicron instruction buses. The objective of the technique is the minimization of not only the single line bit transitions, but also the cross-coupling transitions on the adjacent bus lines. In order to achieve this objective, we have proposed the utilization of two classes of transformations that aim at transforming the original instruction bit sequences into new ones but only with drastically reduced bit transitions on and across the bus lines. The instruction memory stores the transformed instruction bits, while the original bit sequences are restored on the processor side upon fetching the transformed code. The ability to select an optimal subset of transformations out of prespecified transformation classes engenders an extremely efficient and reprogrammable hardware implementation, while obtaining maximal power reductions.

The proposed methodology constitutes an application-specific low-power encoding of instruction memories in embedded processors. The low-power codes are not only applied in an application-specific manner, but are also generated by utilizing application information, thus resulting in an extremely efficient low-power instruction memory encoding for each application hot-spot. An algorithm for extracting the application information and generating the low-power code has been presented. Furthermore, an efficient design flow and hardware architecture for transferring, capturing, and utilizing the application-specific low-power codes have been outlined. The proposed hardware support, inherently reprogrammable and yet extremely efficient

in terms of power and performance overhead, contributes to defining a fixed-silicon yet dynamically customizable embedded processor architecture. Such an architecture not only helps retain the financial benefits of high-volume productions but it also enables the utilization of the proposed application-specific power minimization methodology across multiple applications and application domains. The proposed customization methodology ensures that the memory communication of such a flexible embedded processor architecture is fully supported and capable of attaining significant power reductions as well.

REFERENCES

- [1] P. Petrov and A. Orailoglu, “Performance and power effectiveness in embedded processors—customizable partitioned caches,” *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 1309–1318, Nov. 2001.
- [2] —, “Low-power data memory communication for application-specific embedded processors,” in *Proc. IEEE Int. Symp. System Synthesis*, Oct. 2002, pp. 219–224.
- [3] —, “Power efficient embedded processor IP’s through application-specific tag compression in data caches,” *Des. Arid Test Automation Eur.*, pp. 1065–1071, Mar. 2002.
- [4] H. Lekatsas, J. Henkel, and W. Wolf, “Code compression for low power embedded system design,” in *Proc. IEEE Design Automation Conf.*, June 2000, pp. 294–299.
- [5] L. Benini, A. Macii, E. Macii, and M. Poncino, “Selective instruction compression for memory energy reduction in embedded systems,” in *Proc. IEEE Int. Symp. Low-Power Electronics and Design*, Aug. 1999, pp. 206–211.
- [6] M. R. Stan and W. P. Burleson, “Bus-invert coding for low-power I/O,” *IEEE Trans. VLSI Syst.*, vol. 3, pp. 49–58, Mar. 1995.
- [7] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano, “Asymptotic zero-transition activity encoding for address buses in low-power microprocessor-based systems,” in *Proc. IEEE 7th Great Lakes Symp. VLSI*, Mar. 1997, pp. 77–82.
- [8] Y. Aghaghiri, F. Fallah, and M. Pedram, “Irredundant address bus encoding for low-power,” in *Proc. IEEE Int. Symp. Low-Power Electronics and Design*, Aug. 2001, pp. 182–187.
- [9] S. Ramprasad, N. R. Shanbhag, and I. N. Hajj, “A coding framework for low-power address and data buses,” *IEEE Trans. VLSI Syst.*, vol. 7, pp. 212–221, June 1999.
- [10] E. Musoll, T. Lang, and J. Cortadella, “Working-zone encoding for reducing the energy in microprocessor address buses,” *IEEE Trans. VLSI Syst.*, vol. 6, pp. 568–572, Dec. 1998.
- [11] M. Mamidipaka, D. Hirschberg, and N. Dutt, “Low power address encoding using self-organizing list?,” in *Proc. IEEE Int. Symp. Low-Power Electronics and Design*, Aug. 2001, pp. 188–193.
- [12] Y. Zhang, J. Lach, K. Skadron, and M. R. Stan, “Odd/even bus invert with two-phase transfer for buses with coupling,” in *Proc. IEEE Int. Symp. Low-Power Electronics and Design*, 2002, pp. 754–757.
- [13] Y. Shin, S. Chae, and K. Choi, “Partial bus-invert coding for power optimization of application-specific systems,” *IEEE Trans. VLSI Syst.*, vol. 9, pp. 377–383, Apr. 2001.
- [14] R. Lin and C. Tsai, “Theoretical analysis of bus-invert coding,” *IEEE Trans. VLSI Syst.*, vol. 10, pp. 929–934, Dec. 2002.
- [15] Y. Shin and Y. Chang, “Narrow bus encoding for low-power DSP systems,” *IEEE Trans. VLSI Syst.*, vol. 9, pp. 656–660, Oct. 2001.
- [16] C. N. Taylor, S. Dey, and Y. Zhao, “Modeling and minimization of interconnect energy dissipation in nanometer technologies,” in *Proc. IEEE Design Automation Conf.*, 2001, pp. 754–757.
- [17] P. P. Sotiriadis and A. Chandrakasan, “Low-power bus coding techniques considering inter-wire capacitance,” in *Proc. IEEE Custom Integrated Circuits Conf.*, 2000, pp. 507–510.
- [18] K.-W. Kim, K.-H. Baek, N. Shanbhag, C. L. Liu, and S.-M. Kang, “Coupling-driven signal encoding scheme for low-power interface design,” in *Proc. IEEE Int. Conf. CAD*, Nov. 2000, pp. 318–321.
- [19] L. Benini, G. De Micheli, A. Macii, E. Macii, and M. Poncino, “Reducing power consumption of dedicated processors through instruction set encoding,” in *Proc. IEEE 8th Great Lakes Symp VLSI*, Feb. 1998, pp. 8–12.
- [20] O. Coudert, “On solving covering problems,” in *Proc. IEEE DAC*, June 1996, pp. 197–202.

- [21] P. Shivakumar and N. Jouppi, "CACTI 3.0: An integrated cache timing, power and area model," Western Research Lab., 2001.
- [22] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," *IEEE Computer*, vol. 35, pp. 59–67, Feb. 2002.

Peter Petrov (S'84) received the B.S. and M.S. degrees in computer science from Sofia University, Sofia, Bulgaria, in 1996 and 1998, respectively, and is currently working toward the Ph.D. degree in computer engineering at the University of California at San Diego (UCSD), La Jolla.

Since 1998, he has been with the Department of Computer Science and Engineering, UCSD. His research interests include application-specific embedded processors, low-power processors, embedded systems, and hardware/software codesign.

Alex Orailoglu (M'84) received the S.B. Degree (cum laude) in applied mathematics from Harvard University, Cambridge, MA, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign.

Since 1987, he has been a faculty member with the University of California at San Diego (UCSD), where he is currently a Professor in the Department of Computer Science and Engineering. Previously, he was a Senior Member of the Technical Staff with Gould Research Laboratories, Rolling Meadows, IL. His research interests include digital and analog test, fault tolerant computing, computer-aided design and embedded processors.

Prof. Orailoglu serves on the technical, organizing and/or steering committees of the major VLSI Test and Design Automation conferences and workshops. He is an Associate Editor of the *IEEE Design and Test Magazine*. He served as the Technical Program Chair of the 1998 High-Level Design Validation and Test (HLDVT) Workshop and as the General Chair of HLDVT'99. He is the Founding Chair of the Workshop on Application Specific Processors (WASP). He is also a Member of the IEEE Test Technology Technical Council (TTTC), Executive Committee, and currently serves as Vice Chair of TTTC, and as Chair of the Test Technology Education Program subgroup. He has previously held the positions of Technical Activities Committee Chair and Planning Cochair of TTTC.