Power Efficiency through Application-Specific Instruction Memory Transformations*

Peter Petrov
University of California at San Diego
CSE Department
ppetrov@cs.ucsd.edu

Alex Orailoglu
University of California at San Diego
CSE Department
alex@cs.ucsd.edu

ABSTRACT

The instruction memory communication path constitutes a significant amount of power consumption in embedded processors. We propose an encoding technique that exploits application information to reduce the associated power consumption. The microarchitectural support enables reprogrammability of the encoding transformations so as to track code particularities effectively. The restriction to functional transformations enables effective coding while delivering major power savings, in the process obviating furthermore the necessity to rely on dictionary lookup, one of the major shortcomings of prior approaches. The frugal functional transformation, reliant on a single bit logic gate, introduces no impact to the critical fetch stage of the processor pipeline while delivering fully all the theoretically achievable power savings. The reprogrammable hardware implementation enables flexible and inexpensive switches between the transformations. Extensive experimental results on numerical and DSP codes confirm the theoretically expected magnitude of power savings, evincing reductions that range up to half of the original transitions.

1. INTRODUCTION

The ever-growing improvements in process technology have made the utilization of System-On-a-Chip (SOC) design approaches highly attractive. Improved time-to-market, cost-efficient designs, easy design reuse, and flexible implementation constitute some of the many SOC advantages. Embedded processor cores typically constitute the cornerstones of such systems, enabling better time-to-market, lower design cost, and easily reprogrammable implementations. However, increased silicon integration, together with ever increasing clock frequencies, have led to proportional increases in terms of power.

While power consumption is an increasingly important characteristic in general purpose processor architectures, it assumes a much heightened importance when embedded processor architectures are considered. In mobile applications, such as hand-held and wireless devices, not only does it impact directly cost of ownership, but furthermore may severely undermine the usability and acceptance of the product by limiting its spatial and temporal range. Consequently, techniques for minimizing system power consumption are of significant importance for achieving high product quality. These techniques can be applied on various design abstraction levels, from circuit level to system architecture.

While a number of power saving techniques have been promulgated for general purpose processor architectures, their success is constrained by the lack of application-specific information that can be utilized, severely restricting both impact and predictability of the results in particular contexts. 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. Outpacing the solutions provided by the general purpose processor architecture community to satisfy the more stringent power challenges in the embedded processor marketplace necessitates an ability to exploit the one aspect that differentiates embedded processors, namely, advance knowledge regarding their application context.

A typical SOC design contains several embedded processor cores responsible for various parts of the total system functionality. Each of these processors accesses an on-chip or off-chip instruction memory containing the application code. An access to these memories is typically performed each cycle in order to fetch the next instruction to be executed. Therefore, the interaction between a processor and its instruction memory significantly contributes to the total power consumption. It is well known that transferring addresses and data on long interconnect busses consumes a significant amount of power, due to the high capacitance of the bus lines [1]. This effect is further aggravated if the instruction memory is off-chip (e.g. external flash memory) due to the significantly higher capacitance of the buslines going through the system I/O pins. We propose in this paper a technique for minimizing this significant power overhead in processor memory communication. The technique we propose is reprogrammable, enabling the design, implementation and manufacturing of a single chip, while allowing the appropriate microarchitectural customizations to reduce power drastically.

We present an application-specific dynamic customization technique for power minimization in the data bus of the instruction memory. Prior to its deployment to the system's instruction memory, the application code is analyzed with particular emphasis on the major application loops. Efficient and simple transformations for each bit position of the loop code are identified, so that the number of bit transitions in the resultant sequence is highly reduced. The processor fetch unit restores the original bit sequence for each bus line by applying the transformations. In contrast to previously proposed techniques for compression, our technique delivers power reduction results that are essentially independent of the particular input values or of the input value distributions.

The technique that we propose minimizes the total number of transitions on each bit line on the data bus from the instruction memory. Fundamentally, it utilizes application-specific information for identifying an optimal power encoding. The encoded instructions are directly stored into the memory, while the information about the transformation is provided to the processor core either when loading the program or by software prior to entering the application hot spot being targeted. This information is used to efficiently restore the original bit sequence on each bus line. Not only is the hardware support cost-efficient, but is reprogrammable as well, thus preserving the generality of the processor core while applying the application-specific power encoding that we propose.

2. RELATED WORK

The problem of minimizing the number of transitions on communication busses within a microprocessor-based system has been attacked recently by a number of research groups. Various techniques for low-power address bus encoding exploiting the regularity of these

---

busses have been developed. The *T0* approach [2] introduces an additional line to the instruction memory address bus in order to exploit the typical sequentiality of the instruction addresses. When this line is asserted, the memory controller computes the new address by incrementing the previous one. In [3] this technique is extended and the requirement for an additional redundant control line eliminated. The low-power encoding proposed in [4] utilizes self-organizing lists in order to achieve an optimal encoding for the most frequently accessed addresses. By utilizing a rather complex hardware implementation of self-organizing lists, the approach exploits the temporal and spatial locality of the addresses on both instruction and data address busses. The *Bus-Invert* method [5] inverts the bus content whenever this leads to a smaller Hamming distance compared to the previous value on the bus; an additional bus signal informs the receiver whether the bus content has been inverted. The approach is applicable to any communication bus, but its extremely general nature limits relatively the power savings benefits deliverable on data streams exhibiting regularities. In [6] a methodology for low power ISA encoding has been proposed. 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 that the Hamming distance between frequently encountered pairs of instructions is minimized. A code compression technique utilizing Markov statistics and arithmetic coding has been proposed in [7], with various implementation architectures being evaluated in terms of hardware budget and system power reduction.

3. MOTIVATION

Fundamentally all encoding approaches, whether they be for code compression or for power minimization, are an exercise in code transformation. The ability to exploit arbitrary transformations provides optimal results, but runs a sizable bill in terms of communicating the transformation mapping, itself an avid consumer of bandwidth, power and/or storage. Bounding the communication cost can be achieved by restricting the number of transformations, yet possibly at the expense of reducing the ensuing benefits. Statistical techniques, such as Huffman encoding, rely on the assumption of an underlying nonuniform distribution in the input space, an assumption frequently challenged as the input size increases. In the context of instruction fetching, such statistical techniques, frequently reliant on operating in sizable and variable length chunks of input streams, are further challenged, particularly in the face of frequent branch instructions.

A straightforward attempt to ameliorate this set of shortcomings consists of the application of the transformation techniques on short segments of code and altering the transformation mapping continuously as the segments are traversed. The appreciable cost of continuous transmission of the changing transformation mapping needs to be paid though either through transfers, thus degrading performance and power, or in the form of predetermined and stored memory tables. In either case, the significant cost precludes its use in practical settings.

Practicality can be possibly restored by ameliorating the appreciable cost of continuously communicating dictionary tables of arbitrary transformation mappings through restricting the universe of mappings to ones that can be expressed as *functional* transformations, which can be communicated a lot more economically.

We propose to examine in this paper and intend to show that affirmative answers to the following questions can provide the basis of an innovative, powerful technique for the design of embedded processors with frugal memory consumption in the all important instruction memory path:

- Is it possible to construct functional mappings with high locality capable of ideally delivering high levels of power reduction?
- Is it possible to further restrict the number of functionals in such codes and still attain the theoretical maximal reductions?

- What are the appropriate hardware implementations that deliver the ability to reprogram the transformation mapping expeditiously and inexpensively?

We propose to show that affirmative answers exist to all these questions, thus enabling drastic savings in instruction memory communication at low hardware cost. We intend to show that not only can the functionals be restricted in locality but that quite frugal transformations reliant on a single two-input logic gate only, fundamentally using single bit history, can be effectively utilized with no significant delays introduced to the embedded processor front-end.

These transformations can be further restricted while retaining essentially all achievable power benefits, for realistic block sizes that track basic block sizes in programs. The ensuing hardware cost can be thus sharply limited both vertically by tracking basic block sizes and also horizontally by reducing the width of the index that pinpoints the appropriate transformation functional for each block. The approach we propose, theoretically innovative and well matching program structure in practice, coupled with the frugal supporting hardware structure, promises to deliver new flexible and programmable power-efficient architectures for fixed-silicon embedded processors.

4. OPERATIONAL OVERVIEW

An application typically spends most of its execution time within a few tight loops. A relatively short sequence of instructions is repetitively executed after fetching them one by one either from instruction memory or instruction cache. The transfers on the data bus from the instruction storage cause a significant amount of transitions on each bus line. An example can be seen in Figure 1a in which a sequence of four instructions is shown. Fetching these instructions sequentially causes frequent bit transitions on all of the bus lines.

Since power consumption is proportional to the transition numbers for all bus lines, it is evident that targeting directly and independently the bit sequences corresponding to the bus lines will have a significant impact on the total energy dissipation. Figure 1b shows the "vertical" bit sequences targeted for power encoding. This type of encoding would consider the bit streams associated to each bus line in an independent way. If, for example, we consider the leftmost bit-line column in Figure 1b, one might observe that each bit can be easily generated by simply applying inversion on the previous bit on the original bit sequence. This rather simple transformation takes into account only the original value of the predecessor bit. Therefore, the "1010" sequence can be easily stored as "1000" in the instruction memory as shown in Figure 1c. The latter has two fewer transitions, delivering significant power savings. The purpose of these transformations is to identify any functional correlation that might exist between the original pattern and a pattern with significantly fewer number of transitions. Consequently, the more power efficient bit sequence can be utilized instead in the instruction storage. Applying the transformation upon receiving each bit from the encoded sequence reveals the original bit value, thus restoring the original instruction.

An application-specific processor can take advantage of the application knowledge and utilize an optimal power encoding of the type described in the previous paragraph. The application knowledge in this case is the program code; it can be analyzed and its major loops pinpointed. Power efficient transformations can be identified within the basic blocks and the stream of bits corresponding to each instruction
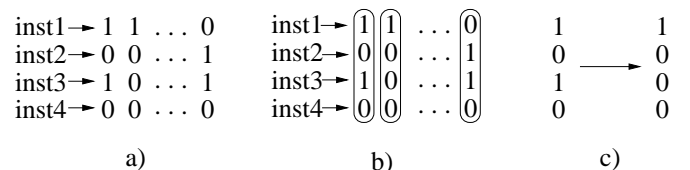

inst1 → 1 1 ... 0	inst1 → [1][1] ... [0]	1	1
inst2 → 0 0 ... 1	inst2 → [0][0] ... [1]	0 → 0
inst3 → 1 0 ... 1	inst3 → [1][0] ... [1]	1	0
inst4 → 0 0 ... 0	inst4 → [0][0] ... [0]	0	0

a)   b)   c)

Figure 1: "Vertical" instruction code transformation

X	$\tilde{X}$	τ	T_x	$T_{\tilde{x}}$	X	$\tilde{X}$	τ	T_x	$T_{\tilde{x}}$
000	000	x	0	0	100	100	x	1	1
001	111	$\bar{x}$	1	0	101	111	$\bar{y}$	2	0
010	000	$\bar{y}$	2	0	110	000	$\bar{x}$	1	0
011	011	x	1	1	111	111	x	0	0

Figure 2: Power efficient transformations for three bit blocks

Size	2	3	4	5	6	7
TTN	2	8	24	64	320	384
RTN	0	2	10	32	180	234
Impr(%)	100.0	75.0	58.3	50.0	43.8	39.1

Figure 3: Transition improvements for various block sizes

bus line stored in an encoded and power efficient form. Significantly less power would be consumed when transferring the bit sequences to the processor core thereupon. The processor core having the information about the selected transformations would restore the original code sequence. The information about the optimal transformations is provided to the hardware support prior to entering the particular application loop.

As the power reductions to be effected occur in bus transitions on the bitline, the proposed technique is applied to the same bitline of a sequence of instructions vertically, as shown in Figure 1b. Each bit, or column in Figure 1b, undergoes a distinct encoding analysis and consequent transformation, as there is no consideration of power reductions to be effected across bits of a single instruction.

5. THEORETICAL FRAMEWORK

5.1 Transformation structure

Let's consider an arbitrary sequence of bits $X = \{..., x_{n+3}, x_{n+2}, ..., x_{n-3},...\}$. We want to find an alternative sequence of bits $\tilde{X} = \{...,\tilde{x}_{n+3},\tilde{x}_{n+2},...,\tilde{x}_{n-3},... \}$ and a transformation τ such that the total number of bit flips within $\tilde{X}$ is less than the bit flips in X and $X = \tau(\tilde{X})$. Since the bit sequence length can be arbitrarily long, identifying a single transformation that maps $\tilde{X}$ to X and providing the hardware support needed for it would lead to an extremely expensive and even more power consuming solution than the original bit sequence. Therefore, in order to control the complexity of the transformations and their hardware support, the initial bit sequence needs to be divided into smaller blocks, for which power efficient transformations with inexpensive hardware support can be utilized. The next step in further restricting the universe of the possible transformations is to notice that the transformation needs to be capable of restoring the sequence bit by bit, instead of decoding only upon receiving a complete block. While previous bits can be buffered, future bits require fetchahead, unnecessarily complicating the pipeline. The transformation τ should be a function of the current bit and a highly limited number, h, of history bits in the form of $x_n = \tau(\tilde{x}_n, x_{n-1}, ...x_{n-h})$. While transformations with various history lengths can be considered, in this paper we concentrate our attention on transformations with one bit history, $h = 1$, of the following form: $x_n = \tau(\tilde{x}_n, x_{n-1})$. These types 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 $2^{2^2} = 16$, but as will be discussed subsequently, a smaller subset of all the transformations does suffice in achieving an optimal solution in terms of minimizing the bit transitions for a fixed block size.

Given a block size k, the task of identifying the optimal subset of transformations then becomes the problem of finding the transformation $\tau(\tilde{x}_n, x_{n-1})$ for every block word, such that $X = \tau(\tilde{X})$ and the number of bit transitions within $\tilde{X}$ is minimal. Therefore, this transformation needs to satisfy the following system of binary equations:

$$x_n = \tilde{x}_n; \;\; \forall i \leq k, x_{n+i} = \tau(\tilde{x}_{n+i}, x_{n+i-1})$$

In order to find the optimal transformation τ for all the block words, this system of equations with variable τ needs to be solved for all 2^k block words. In order to evaluate the global optimal solution, solutions to this problem possibly exploring any of the 16 possible logic transformations were identified for block sizes up to seven. Determination

of the optimal solution involves an algorithm that exhaustively tries to find the code word with minimal number of transitions, such that it can be mapped to the original block word with one of the 16 possible transformations. This transformation mapping for a block size of 3 is shown in Figure 2. The columns X and $\tilde{X}$ show the original and the encoded bit sequence, respectively. The columns denoted by τ show the analytical form of the selected transformations. For this code we can see that the total number of transitions for the original code words is 8, while the transitions within the code words are only 2, hence achieving 75% bit transition reduction.

Let's consider the step of identifying the optimal code word for a given block word, for instance 010. Initially we try to assign a code word with 0 transitions, i.e. 111 or 000. The 111 code word is impossible to map to the initial word 010 by using a transformation τ with the properties specified above, simply because the first equation, namely $x_n = \tilde{x}_n$, would be violated. The 000 word can be mapped to 010 only if a transformation τ exists, such that the equations defining it are satisfied. That is the middle bit 0 of the possible code word 000 and the rightmost bit of the image word 010 produce the middle 1 of the image word, i.e. $\tau(0, 0) = 1$, while the leftmost bit 0 from the code word and the middle bit 1 of the image word generate the leftmost bit 0 of the image, i.e. $\tau(0, 1) = 0$. Evidently, the transformation $\tau(x, y) = \bar{y}$ is such a transformation. Consequently, the initial block word 010 can be obtained from the more power efficient code word 000 by utilizing the transformation $\tau(x, y) = \bar{y}$, thus effectively eliminating both bit transitions associated with the initial word 010. Let's consider now the block word 011. Similarly, as a first step we try to assign a code word with 0 transitions, the only possibility in this case being 111. Yet the two equations defining the transformation $\tau(1, 1) = 1$, $\tau(1, 1) = 0$ constitute an apparently contradictory set of constraints! Consequently, the next step is to try to find code words with a single transition only, which are the next best candidates for power efficient codes. As the original word has a single bit transition, it is easily observed that by utilizing the *identity* transformation $\tau(x, y) = x$, the original value can be used as a code word itself. The identity transformation ensures that the worst case transition behavior of the technique we propose is never inferior to the original code.

These two simple examples illustrate how the optimal code words can be generated, under the assumption that the complete gamut of transformations is allowed. The detailed results for block words of size 3 have been given in Figure 2; in a similar manner, the original total transition numbers (TTN), the consequent reduced transitions numbers (RTN), and the resultant reductions are all computed and shown in Figure 3 for block sizes of 2 to 7. Since the total and the reduced transition numbers are computed by counting the transitions for all binary blocks of a given length, this percentage can be interpreted as the bit transition reduction of the proposed power efficient encoding on a bit stream with uniform bit value distribution.

One can immediately track the numbers shown in row two of Figure 3 by observing that the TTN number 8 is exactly the sum of the entries in column T_x from Figure 2, while the RTN number 2 is the sum of all entries in column $T_{\tilde{x}}$ of the same table. The transition reduction numbers are particularly high for short block sizes, an aspect examined in detail in the next subsection. Nonetheless, even for large block sizes the expected average improvements are still quite significant. Because of a hardware area and block size trade-off, examined subsequently in the paper, a practical solution would be to utilize the proposed low-power encoding technique at larger block sizes.

X	$\tilde{X}$	τ	T_x	$T_{\tilde{x}}$	X	$\tilde{X}$	τ	T_x	$T_{\tilde{x}}$
00000	00000	x	0	0	01000	11000	$x \oplus y$	2	1
00001	11111	$\bar{x}$	1	0	01001	00111	$\overline{x \vee y}$	3	1
00010	11100	$\bar{x}$	2	1	01010	00000	$\bar{y}$	4	0
00011	00011	x	1	1	01011	00011	$x \overline{\oplus} y$	3	1
00100	00100	x	2	2	01100	01100	x	2	2
00101	01111	$x \oplus y$	3	1	01101	10011	$\bar{x}$	3	2
00110	11000	$\bar{x}$	2	1	01110	10000	$\bar{x}$	2	1
00111	00111	x	1	1	01111	01111	x	1	1

Figure 4: Power efficient transformations for five bit blocks

5.2 Optimal codes with fewer transformations

In a quest to further enhance the practicality of the technique, one can explore the sensitivity of the power savings function (the optimal form having been shown in Figure 3) to a restriction in the space of possible transformations. In order to answer this question, we proceed to identify the optimal codes for block sizes up to seven with the additional constraint of also minimizing the number of transformations. This examination approach reveals that a unique subset of only 8 transformations always exists and provides a solution identical to the globally optimal but transformation-unrestricted power encoding! It is noteworthy that this subset consists of the *identical* set of eight transformations for all block sizes up to seven. Reducing the number of transformations down to eight with no impact on encoding optimality leads to highly practical and power efficient hardware.

Figure 4 shows the optimal power encoding for block sizes of five when only the set of 8 transformations is utilized; interestingly, this restriction in no way impacts the achievement of global optimality. Figure 4 uses the identical set of definitions for the rows and columns as in Figure 2. One can observe that only the following functions are utilized: *identity*, *inversion*, *XOR*, *XNOR*, and *NOR*. The columns T_x and $T_{\tilde{x}}$ represent the number of transitions in X and $\tilde{X}$, respectively. The table shows only the first half of the lexicographically ordered bit strings of size five. The second half of the set, i.e., the bit sequences starting with 1, are not shown as the cases are completely symmetrical. The inherent symmetry can be conceptualized as the case in which all the bits from X and $\tilde{X}$ are inverted, effecting a change in the transformations by interchanging *XOR* with *XNOR*, and *NOR* with *NAND*, while retaining intact inversion and identity functions. This set of eight binary functions turns out to be fully sufficient for identifying the optimal transformations for all blocks of size up to seven. Consequently, the control information that needs to be stored at the processor side for selecting the transformation can be reduced to three bits per block, while the number of logic gates needed to implement the transformations is reduced from 16 down to 8 resulting in reduced decoder sizes on the processor side for controlling the transformation selection. Since the number of control bits is fixed, the longer the block, the smaller the overhead.

The transformations τ additionally exhibit the property of leaving intact the rightmost bit in the binary sequence, as this property enables the overlap of blocks with one bit and thus solves the problem of minimizing bit transitions between neighboring blocks. This issue is discussed in detail in the next section.

It is evident that the efficacy of power encoding depends on the length of the block size. Of course, this insensitivity of the power savings to the number of functionals that can be utilized is likely to weaken as block sizes are increased, since the longer the block size, the larger the set of constraints that needs to be satisfied by the transformation τ. For extremely long bit sequences, it would be frequently impossible to find any transformation of the type described above, because of conflicting equations in defining the optimal transformations.

Selecting the appropriate block size is a trade-off between hardware area overhead and efficacy of the solution. Having longer blocks leads to smaller hardware overhead at the processor side, where the informa-

tion about the selected transformations needs to be stored. Of course, the longer the block, the lower the efficacy of the transformation as was shown in Figure 3. The overall discussion in this section points out that block sizes of 5 and 6 should receive primary consideration in possible processor implementations of the proposed techniques.

6. APPLYING THE POWER CODES

In applying the proposed technique on program memories, the bit lines are considered as independent streams. Each bit stream is split into the blocks of fixed size and the optimal transformations for each block are communicated to the special hardware support either during program load or by software prior to entering the application hot spot. The special hardware support is responsible for restoring the original bit sequence. The power encoding methodology is applied only for the major application loops, which contribute most of the program execution time and constitute a significantly small fraction from the total program code. Applying the technique only on an extremely small part of the application code delivers the benefits of reduced hardware requirements as the size of the tables for storing the transformation indices needed for restoring the original bit sequence is reduced.

An important issue that needs to be addressed is the bit transitions across block boundaries. Were blocks to be disjoint, no improvement can be effected. Overlapping blocks, on the other hand, impose an additional constraint on one of the blocks, subsequently reducing the potential efficacy of the transformations. Consequently, in order to minimize this impact, an overlap with 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: $X_1 = (x_n, x_{n-1}, x_{n-2}, x_{n-3})$ and $X_2 = (x_{n+3}, x_{n+2}, x_{n+1}, x_n)$ with bit x_n belonging to both groups. Transformations τ_1 and τ_2 would be assigned to X_1 and X_2, respectively. These transformations would have the property of mapping the power efficient sequences $\tilde{X}_1$ and $\tilde{X}_2$ to X_1 and X_2 in the following way.

$$x_{n-3} = \tilde{x}_{n-3}; \quad x_{n-i+1} = \tau_1(\tilde{x}_{n-i+1}, x_{n-i}), \; i = 3, 2, 1$$
$$x_n = \tilde{x}_n; \quad x_{n+i} = \tau_2(\tilde{x}_{n+i}, x_{n+i-1}), \; i = 1, 2, 3$$

The single bit overlap among the subsequent blocks and the encoding assignment to the overlapped bit by the previous block force τ to use the *encoded* bit value for *both* its arguments in the initial instance of the equation of the system defining τ (i.e. τ_1 uses $\tilde{x}_{n-3}$ instead of x_{n-3} and similarly τ_2 uses $\tilde{x}_n$ instead of x_n). Therefore, the transformation selected for a given block depends on the transformation selected for the previous block. The mutual dependence of the transformations dooms the chances of simple iterative algorithms, such as greedy, delivering provably optimal solutions. The sizable experiments that we have performed on randomly generated bit sequences of length 1000, show though that in all the cases the total reduction in bit transitions was within 1% of the expected value of 50% for codes with block size of five bits, with the small deviations, both on the positive and the negative side, due to slight deviations from the uniform distribution and to the single bit overlap across the blocks. In conclusion, the iterative approach leads in practice to optimal results.

7. IMPLEMENTATION

7.1 General framework

The hardware support for implementing the proposed low-power encoding technique needs to be able to identify the transformation associated with the current bit sequences and apply them in order to restore the original program opcodes. The program memory is loaded directly with the encoded application code and needs no special hardware support. In order to perform instruction decoding, the processor's fetch unit needs to capture the order in which the blocks arrive
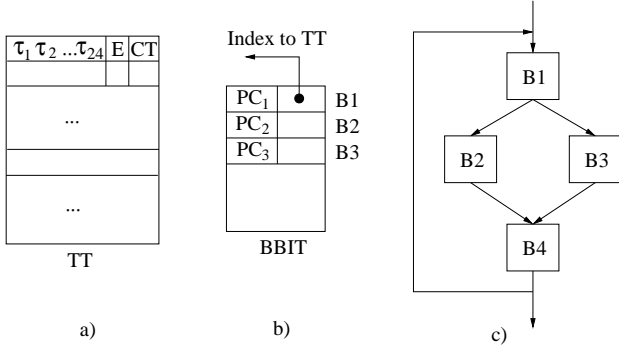
Figure 5: The hardware architecture illustrated on a loop code

from the instruction memory and assign a transformation to the currently active blocks on all the bus lines.

When the program execution crosses basic block boundaries, i.e. in executing a branch instruction, which particular execution path is to be followed cannot consistently be known at compile time. Therefore, a block considered for low-power encoding cannot span through basic block boundaries. Consequently, some encoded blocks that comprise the final set of instructions from a basic block might be incomplete in length. Knowledge about their length is needed in order to switch to the transformation for the bit streams from the next basic block to be executed. Furthermore, some application basic blocks might have extremely low execution frequency or extremely few instructions; it is preferable that such infrequently utilized blocks be left intact in the program memory with no transformation applied, thus avoiding the associated hardware overhead. For these basic blocks, information is needed that instructs the fetch engine to treat these instructions without applying transformations (or equivalently just to use the identity transformation).

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, namely, 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 would contain tables with information about the low-power transformations being selected and their usage order. Two alternatives can be implemented as a possible solution. The first one is to load 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. The tables containing the power transformation information can be accessed as a memory of a special peripheral device. The amount of information needed is insignificant in volume, 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.

7.2 Hardware architecture

The hardware architecture of the proposed implementation is presented in Figure 5. The *Transformation Table (TT)* stores transformation indices associated to each encoded block of bits from the instruction memory. An entry in the *TT* contains the set of transformations for the fixed-length bit sequences 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 shown in Figure 5a, contains

the control bits for selecting the transformation associated to each bit sequence. As the low-power encoding cannot span through the application loop basic blocks, a *set* of entries in the *TT* is allocated for each basic block targeted. Figure 5c shows an example application loop represented as a control-flow graph. Four basic blocks exist and for each of these basic blocks, a contiguous set of entries from the *TT* is allocated. The last *TT* entry for a particular basic block must contain information about how long the last bit sequence targeted with the transformation is. The *End (E)* bit field in the *TT* entry is asserted for the entries that correspond to the tail end sequence for a given basic block. The final field, denoted by *CT*, is a counter corresponding to the size of the last bit sequence. This field is only read from the table 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 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 Figure 5, 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, when an application loop basic block is complete, a lookup into the BBIT produces the TT index for the next basic block.

It is evident that the hardware overhead for the proposed low-power encoding technique is the size of the TT and BBIT arrays. The number of the BBIT entries corresponds exactly to the maximum number of basic blocks that the application loop's CFG can contain. Typically, this is a very small number in the range of 10. Note that, if function calls within the loop exist, then their code can be handled in the traditional way without applying any low-power encoding; only upon return to the application loop is the low-power decoding resumed. An alternative solution is to consider the function code as a part of the loop and utilize the low-power encoding, if the total number of application basic blocks can be accommodated in the BBIT. In terms of added power consumption, we must note that a lookup into the BBIT is performed only in the beginning of a basic block; given the small size of this array, the overhead is insignificant.

The number of entries in the *TT* determines the total number of instructions within the application loop that can be handled by the proposed methodology. Since an entry in the *TT* is allocated per code block, the longer this size, the larger the *TT* utilization. For example, if the low-power code utilizes sequences of size 7, then a 16 entry *TT* can handle a total of $7 * 16 = 112$ instructions, which in practice is well beyond the total number of instructions typically encountered in embedded application loops.

The existence of a number of basic blocks from the loop code, executed extremely infrequently, is to be commonly expected. The contribution of such basic blocks to the total power consumption is quite low and applying the low-power encoding technique on them and allocating entries from the TT might not be worthwhile. Such basic blocks can be easily accommodated by our hardware architecture, by simply allocating only a single entry in the *TT* for all of them. An identity transformation, with no encoding utilized, would be selected and the original code would be processed as in normal execution mode. The *E* field in this entry would be set and its *CT* value would be set to the number of instructions in this basic block.

	mmul	sor	ej	fft	tri	lu
#TR	14.0	3.3	113.4	0.2	8.1	63.8
# 4-block	7.9	1.8	61.8	0.15	3.9	43.0
Reduction(%)	44.0	44.3	45.5	20.6	51.6	32.7
# 5-block	8.6	2.3	69.4	0.1	5.0	48.8
Reduction(%)	39.2	30.5	38.8	17.5	37.8	23.6
# 6-block	10.3	2.1	69.6	0.2	5.6	51.6
Reduction(%)	26.7	35.3	38.7	13.4	31.1	19.1
# 7-block	10.1	2.6	87.3	0.2	6.1	57.8
Reduction(%)	28.5	20.1	23.1	0.0	24.4	9.4

Figure 6: Transition reduction results

Figure 7: Percentage reduction comparison

8. EXPERIMENTAL RESULTS

In our experimental studies, we have measured the effectiveness of the proposed approach by observing the reduction of the transitions on the data bus to the instruction memory. We have assumed a baseline architecture of a typical embedded processor front-end, which fetches and executes instructions in order and one at a time. The instructions are fetched from an instruction storage, possibly an instruction cache or memory; the type of storage bears no impact on the bit transition reductions we attain. We have utilized six benchmarks from the domain of DSP and numerical computations, frequently encountered in many embedded products and capable of exhibiting the strength of the suggested technique due to their inclusion of frequently executed loops: *Matrix multiplication (mmul)* of matrices with size 100x100; *successive over-relaxation (sor)* [8] on a matrix with size 256x256; *extrapolated Jacobi-iterative method (ej)* [9] on a 128x128 grid; *fast fourier transform (fft)* with block size of 256 samples; *tridiagonal system solver (tri)* with matrix size of 128x128; and *lu-decomposition (lu)* algorithm on a matrix of size 128x128.

We have utilized the SimpleScalar toolset [10], which utilizes a MIPS-like instruction set architecture. We have modified the simulation tool, so that we can evaluate the number of transitions on all the bus-lines of the instruction bus for the baseline architecture and for an architecture utilizing the proposed low-power encoding methodology with a *transformation table* containing up to 16 entries. All the low-power codes, presented in this paper, with block sizes from four to seven, were evaluated and the results shown in Figure 6. The first row from the table, denoted by TR, shows the total number of the transitions in millions for the baseline architecture. The subsequent two rows, show the number of bit transitions for a code with block size of four and their corresponding percentage reduction compared to the bit transitions in the original bit sequence. The next three pairs of rows represent the same information for code sizes of 5, 6, and 7.

The results show that the achieved improvement is higher for codes with shorter block sizes. This is consistent with the reduction numbers expected as we have presented in the theoretical analysis of these codes. The improvements for the *fft* benchmark are significantly worse compared to the rest of the benchmarks, as a number of very short basic blocks exist within the major loop with significant contribution to the bit transition numbers.

The improvements in bit transitions range from 10% to 52% with average improvements around 35%-40% for codes with block sizes of four and five bits. As expected, the improvements for codes with larger block sizes are smaller and for codes with six and seven bit blocks the reductions range around 20%-25% on average. Figure 7 depicts graphically the percentage reduction for all the benchmarks and power codes.

9. CONCLUSION

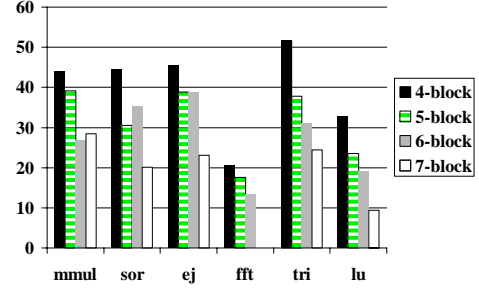In this paper we have presented a methodology for low-power encoding on data busses from instruction memories. The technique utilizes application-specific information with regards to the program to be executed and exploits the associated properties through statically identified power-minimizing transformations upon the application code. The instruction code of the major application loops is analyzed and efficient power codes are assigned to the bit sequences on the bus-lines.

Low-power instruction code transformations have been proposed and formally analyzed. The transformations presented in this paper achieve significant bit transition reductions, while utilizing only an extremely small and a fixed set of binary transformations with various block sizes. The same extremely small set of transformations has been shown to achieve globally optimal results on all practical block sizes. This fundamental property enables an extremely efficient hardware support for the proposed transformations in the form of a few logic gates selected by compact control signals. The inherent reprogrammability of the proposed hardware support enables the application of the proposed scheme in a post-manufacturing fashion. By efficiently utilizing the application-driven low-power code transformations in a reprogrammable manner, the proposed technique can be applied to a large class of embedded processor platforms and numerous modern applications with stringent power requirements.

10. REFERENCES

[1] S. Ramprasad and N. R. Shanbhag, "A coding framework for low-power address and data busses", *IEEE TVLSI*, vol. 7, n. 2, pp. 212–221, June 1999.

[2] L. Benini, G. De Micheli, E. Macii, D. Sciuto and C. Silvano, "Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems", in *7th GLS*, pp. 77–82, March 1997.

[3] Y. Aghaghiri, F. Fallah and M. Pedram, "Irredundant address bus encoding for low-power", in *ISLPED*, pp. 182–187, 2001.

[4] M. Mamidipaka, D. Hirschberg and N. Dutt, "Low power address encoding using self-organizing lists", in *ISLPED*, pp. 188–193, 2001.

[5] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O", *IEEE TVLSI*, vol. 3, n. 1, pp. 49–58, March 1995.

[6] L. Benini, G. De Micheli, A. Macii, E. Macii and M. Poncino, "Reducing power consumption of dedicated processors through instruction set encoding", in *8th GLS*, pp. 8–12, February 1998.

[7] H. Lekatsas, J. Henkel and W. Wolf, "Code compression for low power embedded system design", in *DAC*, pp. 294–299, 2000.

[8] M. E. Wolf and M. S. Lam, "A Data Locality Optimizing Algorithm", in *PLDI*, pp. 30–44, June 1991.

[9] S. Nakamura, *Applied Numerical Methods with Software*, Prentice-Hall, Englewood Cliffs, N.J., 1991.

[10] T. Austin, E. Larson and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling", *IEEE Computer*, vol. 35, n. 2, pp. 59–67, February 2002.