Power Optimization of Core-Based Systems by Address Bus Encoding

Luca Benini, Giovanni De Micheli, *Fellow, IEEE,* Enrico Macii, *Member, IEEE,* Massimo Poncino, *Member, IEEE,* and Stefano Quer

***Abstract*— This paper presents a solution to the problem of reducing the power dissipated by a digital system containing an intellectual proprietary core processor which repeatedly executes a special-purpose program. The proposed method relies on a novel, application-dependent low-power address bus encoding scheme. The analysis of the execution traces of a given program allows an accurate computation of the correlations that may exist between blocks of bits in consecutive patterns; this information can be successfully exploited to determine an encoding which sensibly reduces the bus transition activity. Experimental results, obtained on a set of special-purpose applications, are very satisfactory; reductions of the bus activity up to 64.8% (41.8% on average) have been achieved over the original address streams. In addition, data concerning the quality and the performance of the automatically synthesized encoding/decoding circuits, as well as the results obtained for a realistic core-based design, indicate the practical usefulness of the proposed power optimization strategy.**

***Index Terms*— Bus encoding, integrated circuit, intellectual property, low power, power optimization.**

I. Introduction

THE use of intellectual proprietary components, such as core processors and microcontrollers, as basic blocks for the development of dedicated (i.e., special-purpose) digital systems is becoming a well-established design strategy in the microelectronics industry. Financial reasons motivate this choice. The core-based design style is, in fact, the hardware counterpart of the software programming paradigm based on the reuse of library functions. A reduced product turn-around time is thus guaranteed with a reasonably good quality and limited economical effort.

In this paper, we focus on the design of low-power, special-purpose systems. More specifically, we face the problem of reducing the power dissipated by a design containing a core processor, or a microcontroller, through the application of system-level optimization techniques.

It is well known that, due to the intrinsic capacitances of system-level buses, a considerable amount of power is dissipated at the input/output interface of a processor when binary patterns have to be transmitted over the communication channel. More precisely, the capacitive load on the processor's input/output drivers is usually much larger (up to three orders of magnitude) than that on the internal nodes of the processor [1]. As a consequence, dramatic optimizations of the average power consumption of a processor-based system can be achieved by minimizing the number of transitions (i.e., the switching activity) on the buses connected to the primary outputs of the processor.

One way of accomplishing this task is to encode the binary patterns that must be transmitted over the bus. Most of the proposed schemes leverage some statistical properties of typical bus streams. In particular, in the case of address buses, the high probability of memory addresses to be consecutive is exploited to reduce the average number of signal transitions [2]–[4].

However, even in the situations where address streams have low sequentiality, it may well be the case that other types of correlations exist between the patterns that are being transmitted. More specifically, we have experimentally observed that time-adjacent addresses usually show high *block correlations*. For processors adopting segment/page-based memory architectures, this can be easily justified by the fact that intrasegment/page branches and jumps are much more frequent than intersegment/page ones. Therefore, even though the strict sequentiality of addresses may be destroyed by a branch/jump instruction, some portions of the patterns may still be highly correlated.

We exploit block correlations in address streams to automatically generate encoding schemes which minimize the average bus switching activity. Our approach, called in the following *the Beach solution*,[1] can be summarized as follows. Starting from typical address bus traces, we collect statistical information identifying possible block correlations. We then group the bus lines in clusters according to their correlations, i.e., lines belonging to the same cluster are highly correlated. For each cluster, we automatically generate an encoding function, that is, a one-to-one combinational Boolean function, that translates each bit configuration in the original cluster into a new one.

The algorithm that finds the encoding functions targets the minimization of the switching activity. Thus, well established technology initially developed in the context of finite state machine synthesis can be successfully exploited. The output

Manuscript received December 12, 1997; revised May 15, 1998. This work was supported in part by the CIS and the NSF under Grant MIP-9421129.

L. Benini is with the Dipartimento di Elettronica Informatica e Sistemistica, Universitá di Bologna, Bologna 40136 Italy.

G. De Micheli is with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305 USA.

E. Macii, M. Poncino, and S. Quer are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino 10129 Italy.

Publisher Item Identifier S 1063-8210(98)08520-5.

[1] *The Beach Club* is the name of the place where the method was initially conceived by the authors during DAC'96 in Las Vegas.

of the encoder is a stream with reduced average number of bus line transitions. At the receiving end of the bus, the original encoding is required; then, the inverse function must also be calculated.

Since the target is a reduction of the power consumed by the system as a whole, it is mandatory to guarantee that savings achieved are not offset by the extra power dissipated by the encoding and decoding circuitry. In addition, bus latency is usually a critical design constraint. Therefore, simultaneous power and timing optimization must be targeted during the synthesis of the logic for address encoding/decoding. The Beach solution is a step forward in addressing these two issues. The encoder and the decoder are the gate-level implementations of the encoding and decoding functions, respectively. Since they operate on blocks, their speed can be easily controlled by specifying a maximum block size. However, if the timing constraints are not tight, our approach can be used to explore the opportunities for power savings that become available when large blocks are allowed.

If the internal description of the core is accessible and modifiable, the encoding operation can be incorporated directly into the address generation step. On the other hand, for black-box IP components, resorting to an encoder is mandatory; thus, its design is key for making the Beach solution applicable in practice.

II. Previous Work

In this section, we review previous work in the area of power minimization by reduction of the switching activity in the memory-processor interface. We can categorize the existing approaches in two broad classes: *bus encoding* techniques and *memory organization* techniques.

A. Bus Encoding Techniques

The switching activity on a communication bus can be reduced by encoding the binary patterns before they are transmitted. Depending on the type of information to be exchanged, several low-power encoding schemes, exploiting distinctive spectral characteristics of the pattern streams have been recently proposed.

Stan and Burleson have introduced the use of the *bus-invert* code [5]. The method performs well when patterns to be transmitted are randomly distributed in time and no information about pattern correlation is available; therefore, it seems appropriate for encoding the information traveling on data buses.

When address buses are considered, the temporal correlation between successive addresses is usually strong. In fact, streams are typically composed of bursts of in-sequence addresses, intermingled with out-of-sequence ones (corresponding to taken branches and jumps) [6]. The high frequency of consecutive patterns is fruitfully exploited by codes such as *Gray* [2] or $T0$ [3], [4]. However, if the percentage of in-sequence addresses decreases, the effectiveness of the aforementioned codes diminishes as well.

The recently proposed *working zone* encoding [7] tackles some of the limitations of Gray and T0, and it is well suited for address buses of both instructions and data streams. It is based on the observation that many programs access multiple data arrays. The accesses to each array are mainly in-sequence, but unfortunately they are often interleaved; then, the sequentiality on the bus is destroyed. The working zone scheme restores sequentiality by memorizing the reference addresses of each working zone on the receiver side and by sending only the highly sequential offset. Whenever the data access moves to a new working zone, this information is communicated to the receiver with a special codeword. The receiver changes the default reference address and offset transmission can resume.

Although working zone encoding is more flexible than Gray and T0, it still relies on strong assumptions on the patterns in the stream. If the data access policy is not array-based, or if the number of working zones is too large, this encoding scheme looses effectiveness. Moreover, similarly to the case of the T0 code, it requires extra bus wires for communicating a working zone change. This requirement might not be acceptable because it changes standard bus widths and chip pinouts.

B. Memory Organization Techniques

Bus encoding schemes reduce interface power by changing the format of the information transmitted on the processor-memory bus. An alternative approach to the problem consists of changing the way the information is stored in memory so that the address streams generated by the processor have already low transition activity. Two research directions have been pursued.

The first one targets the optimal exploitation of memory hierarchies [8]–[10]. Higher levels of the memory hierarchy can be accessed at a low-power cost, but they have limited storage capacity. Power can be reduced by organizing the data in such a way that the higher levels of memory hierarchy are optimally exploited and data transfers from lower levels are minimized. The second area of research targets specifically the bus-memory interface [8], [11], [12]. Data are allocated in memory trying to emphasize sequential accesses since they have inherently lower switching activity than unordered accesses.

Both bus encoding and memory organization have advantages and limitations. Bus encoding can be applied with limited or no knowledge about the functionality of the application being executed, while memory organization techniques need detailed information on data structures, loops and control flow structures. On the other hand, bus encoding imposes hardware overhead (the encoder and decoder) while memory organization exploits the existing hardware.

Memory organization techniques and bus encoding are not mutually exclusive. On the contrary, optimal power minimization strategies should leverage their synergy. For instance, data allocation emphasizing sequential memory access can be used in conjunction with T0 encoding to heuristically minimize the number of bus line transitions. For hierarchical memory systems, bus encoding can be implemented at any level of the memory hierarchy, for example between cache and main memory.

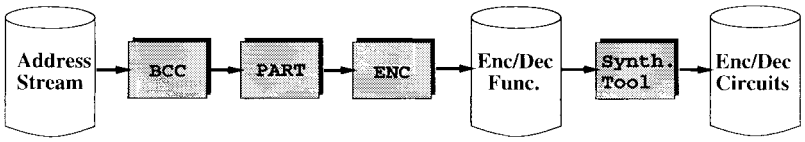


Fig. 1. The Beach solution.

III. The Beach Solution

The solution we propose to minimize the power dissipated by the processor-memory path of a core-based system belongs to the category of bus encoding techniques. In particular, we target a reduction of the number of transitions occurring on the address bus. The Beach solution differs from previously presented low-power encoding schemes in that it is strongly application oriented. In fact, the encoding and decoding functions are properly determined for a given program based on the analysis of the address streams produced by one or more executions of such program. For this reason, the technique is not applicable to general-purpose, multiuser, and multitasking computing systems, where several programs (possibly characterized by address streams of substantially different nature) can run concurrently. On the other hand, it has proven to be particularly suitable to special-purpose machines, where the same portion of embedded code is executed over and over by the core processor.

A. Overview

A high-level block diagram of the basic operations required to apply the Beach solution is depicted in Fig. 1.

The entry point is the address stream produced by one or more runs of the embedded code on the core processor or microcontroller. Such stream is fed to the tool, called BCC (bit correlation calculator), whose task is to perform some statistical analysis of the patterns appearing in the stream, and to extract from the result of such analysis a measure of the correlation that may exist between pairs of bus lines. This information is then processed by program PART, whose objective is to determine a partition of the bus lines in disjoint clusters. Each cluster is finally processed by the ENC program, whose basic function is to encode the bus lines belonging to the cluster so that the number of bus transitions occurring when the embedded code is executed again gets minimized. The output of the ENC program is a set of encoding and decoding functions, one for each bus line, whose implementation in combinational logic through an automatic synthesis tool originates the encoder and decoder circuitry.

B. Correlation Measures

As outlined in the previous section, our technique targets a reduction in power dissipation by decreasing the frequency of transitions of multiple bus lines. To achieve this goal, patterns that are often consecutively transmitted on the bus should be reencoded to patterns with similar codes, i.e., codes with a minimum Hamming distance (possibly equal to one). In principle, it is possible to measure exactly the probability of every pair of patterns to be sent consecutively on the bus. Given a stream of l patterns, $X_k, k = 1, 2, \cdots, l$, a set of 3-tuples (X_F, X_S, c) should be stored, one for each different pair of consecutive patterns that appear in the stream. The first pattern of the pair is X_F, the second one is X_S, and the number of occurrences of the pair in the stream is c.

There are two practical problems with this idea. First, the amount of memory required to store the 3-tuples is, in the worst case, proportional to the length of the stream l. This is unacceptable because the stream length may be in the order of millions. Second, in many cases we are interested in encoding only blocks of bits. If, for example, the most significant bits of the pattern have extremely low switching activity, encoding them is not really useful.

For these reasons, we decided to use a more compact measure and we focus on the *correlations* between groups of bits. We measure correlations using a *pairwise* approximation. The key advantage of pairwise correlation measures is that they can be stored in $O(N^2)$ where N is the bus width.

Let us call $X(t) = (x_1^{(t)}, x_2^{(t)}, \cdots, x_N^{(t)})$ the pattern transmitted on the bus at time t, where each x_i is a bit of the pattern. Let us consider two bits x_i and x_j transmitted on the bus, in the same position, i.e., $i = j$, or in two different positions, i.e., $i \neq j$, and at the same time, i.e., $t = \tau$, or at two different but consecutive clock cycles, i.e., $t = \tau_1$ and $t = \tau_2 = \tau_1 + 1$.

For each bit of the bus we define variable χ_i as the symmetric encoding of bit x_i: $\chi_i = 1$ when $x_i = 1$, and $\chi_i = -1$ when $x_i = 0$. Then, given a pattern stream of length l, we define for each variable χ_i its average value μ_{χ_i} and its standard deviation σ_{χ_i}

$$\mu_{\chi_i} = \frac{\sum_{t=0}^{l-1} \chi_i^{(t)}}{l} \qquad \sigma_{\chi_i} = \sqrt{\frac{\sum_{t=0}^{l-1} (\chi_i^{(t)} - \mu_{\chi_i})^2}{l-1}}.$$

In addition, we define the *covariance* of two variables, χ_i and χ_j, and a time lag τ as:

$$\mathrm{Cov}(i,j,\tau) = \frac{\sum_{t=\tau}^{l-1} (\chi_i^{(t)} - \mu_{\chi_i})(\chi_j^{(t-\tau)} - \mu_{\chi_j})}{l-1-\tau} \tag{1}$$

and the *correlation coefficient* between bit x_i and bit x_j as

$$\rho_{i,j,\tau} = \frac{\mathrm{Cov}(i,j,\tau)}{\sigma_{\chi_i}\sigma_{\chi_j}}. \tag{2}$$

By setting $\tau = 0$ or $\tau = 1$ and using equal or different values of i and j we obtain different types of correlations that we call *spatial*, *temporal*, and *spatio-temporal*. Roughly speaking, the first type, obtained by setting $\tau = 0$ and $i \neq j$, expresses the likelihood of correctly predicting the value of one bit of pattern X_k knowing the value of another bit in the same pattern. The second type, obtained by setting $\tau = 1$

and $i = j$, expresses the likelihood of correctly predicting the value of a bit in pattern X_k by observing its value on the previous pattern. In general, for $\tau \neq 0$ and $i \neq j$ we have *spatio-temporal* correlation.

Since we are interested in measuring the likelihood of concurrent switching of more than one bus line, only *spatial* (S) and *spatio-temporal* (ST) correlations have some relevance to us. If spatial correlation between bits x_i and x_j is high and both x_i and x_j have high transition activity, the likelihood of a double transition is high as well. A similar reasoning holds for spatio-temporal correlations.

Although spatial and spatio-temporal correlations do contain useful information, it is possible to formulate a measure of correlation that is more directly related to the probability of multiple switchings. We define the *switching* (SW) correlation as the spatial correlation between pairs of *transition variables*. A transition variable η_i is defined as follows:

$$\eta_i^{(t)} = +1 \cdot (x_i^{(t)} \cdot (x_i^{(t-1)})') - 1 \cdot (x_i^{(t)})' \cdot x_i^{(t-1)}$$

$\eta_i^{(t)}$ has value "1" if bit x_i makes a raising transition from clock cycle $t-1$ to t, it has value -1 in case of a falling transition, and it is zero otherwise. We can compute the switching correlation covariance between transition variables η_i and η_j $(i \neq j)$ and the switching correlation coefficient between bits x_i and x_j using (1) and (2), in which χ is replaced with η, and τ is set to zero.

Switching correlation directly measures the likelihood of having a concurrent transition on two bus lines, therefore we expect it to be a more reliable source of information. Notice, however, that all correlation measures we have defined are approximate. The information on how transitions on *groups* of bits are correlated with transitions on other groups is completely lost. Consequently, we cannot claim that switching correlation is always the best measure, and the results of our experiments have confirmed this fact.

Notice that the complexity of the procedure for extracting the correlation coefficients is $O(N^2 l)$, since the entire stream (of length l) has to be analyzed. However, we do not need to store the stream in memory. In fact, the correlations can be computed on-the-fly by a filter-like program.

C. Clustering Heuristics

The pairwise correlation coefficients, computed with one of the methods outlined in Section III-B, can be collected in a $N \times N$ *correlation matrix*, $\boldsymbol{C}$. We try to exploit the correlation information to identify subsets of bits that are suitable to be encoded together. Intuitively, we want to cluster together bits that have high pairwise correlation, since this is an indication that the probability distribution of bit patterns in the cluster is highly nonuniform. If this is the case, encoding can be very effective in reducing the average number of concurrent transitions for bits in the cluster.

The correlation matrix $\boldsymbol{C}$ can be seen as the adjacency matrix of a weighted undirected graph $G(V, E, W)$. The set of vertices, V, of G represents the set of bus lines; two vertices, v_i and v_j, are connected by an edge $e_{i,j}$ if at least one of the correlation coefficients $\rho_{i,j,\tau}$ and $\rho_{j,i,\tau}$ is nonzero; in this case, the edge weight is $w(i,j) = \rho_{i,j,\tau} + \rho_{j,i,\tau}$.

One approach for determining a good clustering of the bus lines is that of computing the strongly connected components of a graph, G_P, derived from G through an edge pruning step. Such step, consisting of the removal of the edges in E whose weights are below a given threshold $w_{\min}$, is useful to filter out correlations that are not statistically significant. The problem with this simple solution is that only a rough control of the size and the granularity of the clusters is allowed through the selection of the threshold value $w_{\min}$. In particular, several experiments have shown that the procedure tends to create large clusters.

In principle, larger clusters enable the computation of better encodings, since a more global optimality can be achieved. In practice, however, they should be avoided, since the hardware cost of encoder and decoder rapidly increases with the cluster size, and so does the complexity of the data collection and the encoding procedure. Therefore, we have developed a greedy, yet effective clustering algorithm, described next, which offers a high degree of control on the maximum cluster size, as well as a good quality of the clusters.

The clustering procedure operates on the pruned graph, G_P, and its flow is the following ($k_{\max}$ indicates the user-specified maximum cluster size):

- select an edge with maximum weight;
- cluster the head and tail vertices of the edge together into a single vertex;
- in case vertex clustering induces an edge merging, assign to the merged edge the maximum weight among those of the original edges;
- store in the clustered vertex the number k of original vertices in it (initially, $k = 1$ for each vertex);
- if a clustered vertex has $k = k_{\max}$, eliminate that vertex and all the edges connected to it;
- repeat all the steps above until the graph is empty or no edge is left;

The clustering algorithm, whose run time is linear in the number of edges of G_P, returns a set of clusters with number of elements bounded by $k_{\max}$. Such clusters are the starting points for the encoding algorithm which is described in the next section.

Example 1: Fig. 2(a) shows a pruned graph, G_P, with six vertices: v_1 to v_6, each of which has initially $k = 1$. The procedure applied to this graph with $k_{\max} = 4$ proceeds as follows. The edge with maximum weight, $e_{1,4}$, is selected, and the head and tail vertices, v_1 and v_4, are clustered, thus producing vertex v_A with $k = 2$ in Fig. 2(b). Now, edge $e_{A,3}$ is selected, and vertices v_A and v_3 are clustered, originating vertex v_B with $k = 3$ [Fig. 2(c)]. The vertex merging operation also induces the merging of edges $e_{A,5}$ and $e_{3,5}$. The merged edge, $e_{B,5}$, has now weight $w(B,5) = \max(w(A,5), w(3,5)) = 0.7$. Edge $e_{B,5}$ is selected next, and vertex v_C with $k = 4$ is created [Fig. 2(d)]. At this point, node v_C identifies a cluster with maximum allowed size; therefore, it is eliminated from the graph, and all the edges connected

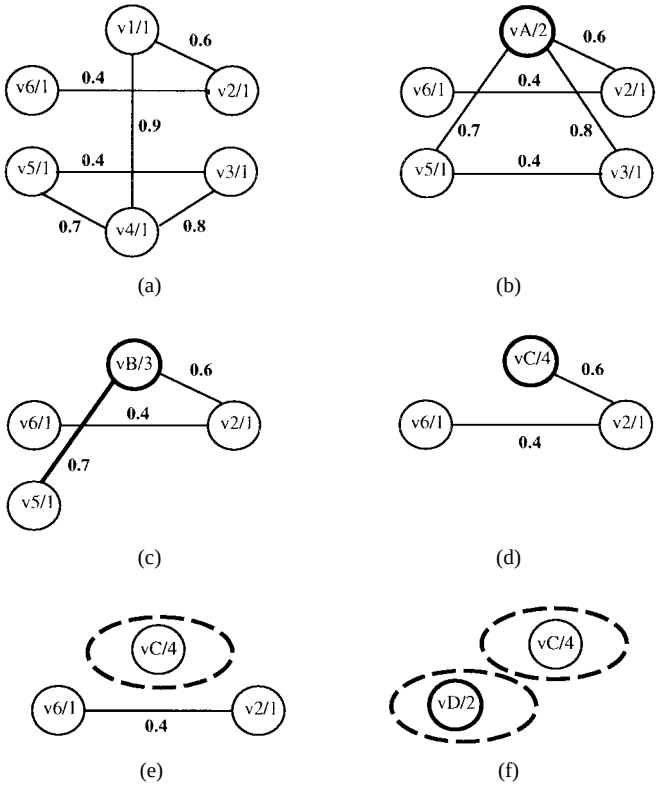


Fig. 2. Clustering example.

to it (i.e., edge $e_{C,2}$) are removed [Fig. 2(e)]. Finally, vertices v_2 and v_6 are clustered, giving the final result of Fig. 2(f).

D. Synthesis of the Encoding/Decoding Logic

The clusters obtained using the algorithm of Section III-C are further manipulated in order to collect more accurate statistical information. In particular, for each cluster of size k, we build a new weighted graph $Q = (T, J, Z)$, called the *transition graph*. The set of vertices T is the set of combinations of the bit lines (belonging to the cluster) that appear in the stream. The cardinality of T is $|T| \leq 2^k$, and it is generally much smaller than the upper-bound because not all combinations appear in the sample. The weights on the edges J are the frequencies of transitions between the bit configurations associated to the vertices connected by the edges.

Example 2: If a block includes $k = 3$ lines, we have a maximum of 8 vertices in T. Assume that the three lines in the sample only take on the values 001, 000, 111, and 100. The graph Q has then four vertices: $t_1 = 001$, $t_2 = 000$, $t_3 = 111$, and $t_4 = 100$. If in the sample there are 120 transitions $000 \rightarrow 001$ and 268 transitions $001 \rightarrow 000$, the weight on the edge between vertices t_1 and t_2 is 388. The resulting graph Q is shown in Fig. 3.

The output of the PART program (see Fig. 1) is a set of graphs $\mathcal{Q} = \{Q_i\}$, one for each cluster. Given that the edge weights of transition graphs are transition frequencies, we might change the vertex codes of each graph so as to minimize the following cost function:

$$\text{Cost} = \sum_{(t_i,t_j)} z_{i,j} \cdot H_{i,j} \tag{3}$$

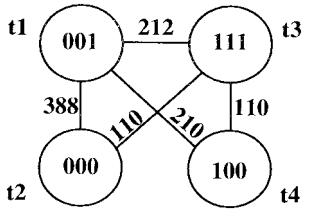


Fig. 3. Transition graph example.

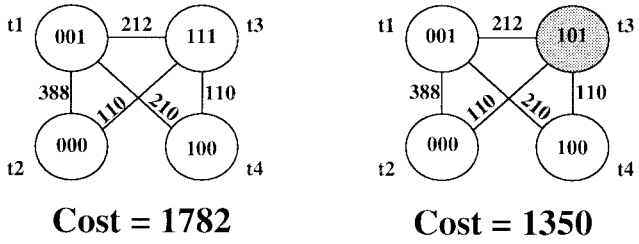


Fig. 4. Reencoding to minimize the number of transitions.

where $z_{i,j} \in Z$ is the weight of edge (t_i, t_j), and $H_{i,j}$ is the Hamming distance between the two codes of vertices t_i and t_j. The rationale is to assign closer (in the Hamming sense) codes to vertices joined by "heavy" edges.

Example 3: The cost for the graph of Fig. 3 is 1782, as shown in Fig. 4 (left part). By reencoding vertex t_3 from 111 to 101, we have now that t_3 is "closer" to vertices t_1, t_2, and t_4. The total amount of transitions between the vertex pairs (t_3, t_1), (t_3, t_2), and (t_3, t_4) is thus reduced, and the value of the new cost function is 1350.

It has been experimentally observed that, in general, transition graphs contain relatively few vertices (that is, combinations of patterns) if compared to the number of all possible k-bit patterns, 2^k, where k is the size of the corresponding cluster. This characteristics is particularly desirable when trying to reencode a graph, since it provides many degrees of freedom that can be exploited in reassigning codes to the various vertices.

The ENC tool reencodes the transition graph of each cluster. It uses the algorithm proposed in [14], which is fully based on implicit representations of Boolean and pseudo-Boolean (i.e., real-valued) functions by means of binary decision diagrams (BDD's) and ADD's. The reencoding problem, whose exact solution is NP-hard, is solved heuristically.

The program provides two heuristics; the first one is based on the solution of a *maximum weighted matching*, while the second heuristics is based on a recursive version of the Kernighan–Lin [15] partitioning algorithm. The two heuristics can be used to trade off accuracy for memory requirements of the BDD/ADD representations; while the matching heuristics provides more accurate results, the minimum-cut one is less memory consuming.

For a graph $Q_i \in \mathcal{Q}$ representing a cluster of k_i bits, the reencoding information is given as a set of *reencoding functions* $E = (E_1, \cdots, E_{k_i})$, where each function E_j expresses each bit j as a function of all the other bits. In other terms, the reencoding can be thought of as an input/output relation $E = E(x_1, \cdots, x_{k_i}, y_1, \cdots, y_{k_i})$ which binds reencoded bits y_j to

the original bits x_j. In practical terms, the construction and synthesis of the encoding and decoding logic is obtained by building a Boolean expression for the relation E, represented with BDD's, and eventually dumping the BDD representation to a file as a network of multiplexors. This procedure yields the netlist for the encoder, that can be optimized using standard techniques.

Given the relation E, obtaining its inverse E^{-1} (which represents the decoding relation) is quite easy, since it is sufficient to swap the sets of x's and y's, that is, $E^{-1}(x_1,\cdots,x_{k_i},y_1,\cdots,y_{k_i}) \equiv E(y_1,\cdots,y_{k_i},x_1,\cdots,x_{k_i})$. The netlist for the decoder is again obtained through logic optimization of the network of multiplexors derived from the BDD's of E^{-1}.

IV. Experimental Results

In this section, we present experimental results concerning the use of the Beach solution. We first show data regarding the reduction in switching activity that we have obtained for software programs commonly executed by core processors of embedded systems. All these experiments have been executed on the MIPS R4000 microprocessor running in single-user/single-task mode. Because of its architecture—the address bus is multiplexed between instruction and data addresses—this microprocessor well simulates the behavior of most of the core processors and microcontrollers that are available on the market. Then, we present an application of the Beach solution to a real system consisting of a core processor, a memory containing the program to be executed, and some glue logic.

A. Special Purpose Programs

We have selected a set of software functions which are usually executed by dedicated systems for image processing, automotive control, DSP, robotics, plant control, and so on. We have collected the address streams generated by typical runs of such applications, we have encoded them using our approach, and we have simulated the new traces to determine the total number of bus transitions. Finally, we have compared the so obtained bus activity results to those computed before the encoding was applied.

Table I reports the outcome of our investigation. For each application, we give the length of the address stream considered for the experiment, the number of bus transitions when no encoding is used, the number of bus transitions after encoding, and the percentage of savings achieved with respect to the unencoded case. Also shown (two right-most columns of the table) are the number of transitions and the percentage of savings achieved when the working zone encoding (WZE) scheme of [7] is applied to the original trace. In our experiments, WZE has outperformed all previously proposed bus encoding methods, including Gray, T0, Bus_Invert, T0 + Bus_Invert, Dual_T0, and Dual_T0 + Bus_Invert [4].

Results are highly satisfactory; in fact, a 41.8% average savings has been obtained, with a peak improvement of 64.8% for the `vxv_mul` example. Also, the comparison with WZE is clearly in favor of the Beach solution. Running times of the overall encoding procedure are always within a few minutes. It should be noticed that the data concerning the `dashb` example do not refer to a single run of the embedded code but, rather, to an average value taken over a total of 10 runs with different input conditions. This is necessary to mitigate the impact of data dependency on the order of execution of the program's instructions.

TABLE I
Results for Special Purpose Programs

Program	*Stream Length*	*Binary Trans.*	*Beach*		*Working Zone*	
			Trans.	*Sav.*	*Trans.*	*Sav.*
`dashb`	84918	619690	443115	28.4	452605	26.9
`dct`	13769	48917	31472	35.6	36258	25.8
`fft`	23441	138526	85653	38.1	99814	27.9
`mat_mul`	22156	105947	60654	42.7	72881	31.2
`vxv_mul`	19417	133272	46838	64.8	85473	35.8
Average				41.8		29.5

`dashb`: Car Dashboard Controller [16].
`dct`: Discrete Cosine Transform.
`fft`: Fast Fourier Transform.
`mat_mul`: Matrix Multiplication.
`vxv_mul`: Vector by Vector Scalar Multiplication.

TABLE II
Encoder and Decoder Implementation

Program	*Circuit*	*Area*	*Delay*	*Power*
`dashb`	Encoder	4878	1.06	291.6
	Decoder	5022	1.30	283.2
`dct`	Encoder	6990	1.85	464.4
	Decoder	5622	1.84	485.9
`fft`	Encoder	8748	1.36	567.0
	Decoder	8748	1.24	574.7
`mat_mul`	Encoder	14346	2.70	970.8
	Decoder	15732	2.93	810.8
`vxv_mul`	Encoder	9378	2.03	493.4
	Decoder	8712	2.18	577.7

An issue that cannot be neglected regards the complexity, and thus the speed and the power consumption, of the encoding/decoding logic that must be added at the bus ends. In Table II, we report the characteristics of the circuits (that is, area in μm^2, delay in ns, and power in μW) obtained through automatic synthesis and optimization of the encoding/decoding functions. The results are obtained using Synopsys DesignCompiler for the synthesis, and Synopsys DesignPower for the power estimation. The circuits are optimized for delay, and mapped onto a 0.35 μm, 3.3 V gate-library from SGS-Thomson containing approximately 120 cells.

B. Case Study: Core-Based System

In this section, we show an application example of the Beach solution to a realistic, yet simplified core-based design [17]. The system implements a three-tap digital filter which is characterized by the following equation:

$$y = c_0 + c_1 z^{-1} + c_2 z^{-2}$$

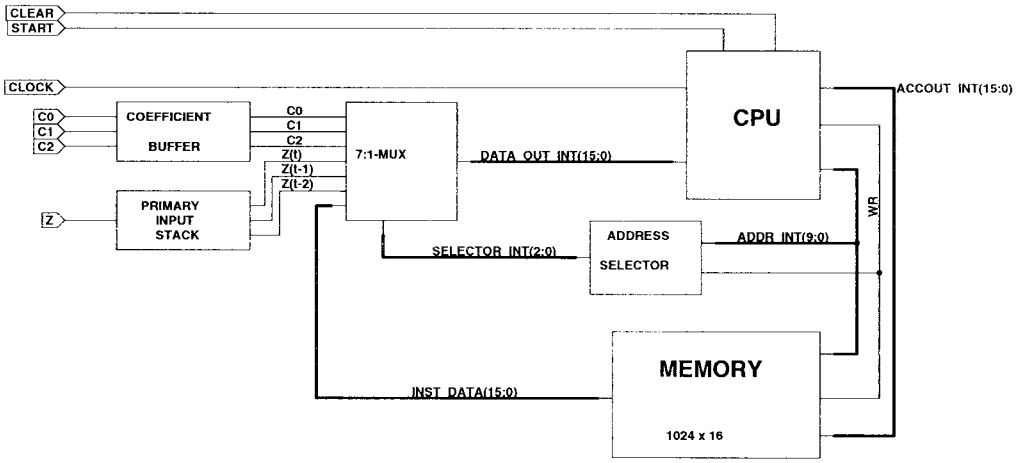


Fig. 5. High-level system architecture.

The hardware realization of the filter uses, as embedded core, the load/store CPU distributed with the SpeedChart tool [18], a program for Statecharts manipulation. The assembly language of the CPU consists of eight 16-bit instructions, each of which is composed of a 10-bit field for the address and a 6-bit field for the instruction op-code.

The architecture of the system is shown in Fig. 5. Besides the core processor, it contains a 1024×16 RAM, an address selector, a 7-to-1 multiplexor, and two input buffers. Memory addresses are 10-bit wide, while data are 16-bit wide.

The assembly code of the filter consists of a total of 38 instructions, which are stored in the RAM starting at address $(010)_{\text{Hex}}$. After the initialization phase (which involves the access to some constants which are stored in the bottom 16 memory cells), the program executes the following operations for an infinite number of times, that is, until the system is halted:

1) the coefficients c_0, c_1, and c_2 are loaded from the primary inputs and stored in cells $(0FB)_{\text{Hex}}$, $(0FC)_{\text{Hex}}$, and $(0FD)_{\text{Hex}}$;
2) the sample, z, to be processed is loaded from the primary inputs and stored in cell $(0FE)_{\text{Hex}}$;
3) the intermediate terms $c_1 z^{-1}$ and $c_2 z^{-2}$ are computed through iterative sums, and then accumulated in cell $(0FF)_{\text{Hex}}$;
4) the value of y, now contained in cell $(0FF)_{\text{Hex}}$, is moved to one of the top 512 cells of the memory at the address contained in cell $(1FF)_{\text{Hex}}$;
5) the content of cell $(1FF)_{\text{Hex}}$ is incremented by 1, using a modulo-512 arithmetic;
6) A jump back is made to restart the sequence of operations.

Notice that the coefficients of the filter can be modified before a new sample is processed, and that the results are stored in a 512-word circular queue (spanning addresses from $(200)_{\text{Hex}}$ to $(3FF)_{\text{Hex}}$) for subsequent usage.

Table III shows the results of the application of the Beach solution to the address streams generated by the core processor (bus ADDR_INT in Fig. 5) when the filtering routine is executed. The data refer to a stream of approximately 465 000 patterns, corresponding to the processing of a total of 5,000 input samples. The coefficients of the filter are reprogrammed every 500 samples.

TABLE III
RESULTS FOR THE CORE-BASED SYSTEM

Stream Length	*Binary Trans.*	*Beach*		*Working Zone*	
		Trans.	*Sav.*	*Trans.*	*Sav.*
464856	2714472	787715	71.0	1750277	35.5

TABLE IV
ENCODER AND DECODER FOR THE CORE-BASED SYSTEM

Circuit	*Area*	*Delay*	*Power*
Encoder	9792	2.32	542.3
Decoder	9324	2.03	546.1

The encoder/decoder circuits have been implemented as for the case of the special-purpose programs; the results of the synthesis (i.e., area, delay, and power dissipation) are summarized in Table IV.

In order to estimate the usefulness of the proposed solution, we need to compare the amount of power saved on the bus to the additional power dissipated by the encoding/decoding circuitry, whose value is: $P_{\text{Enc/Dec}} = 1088.4\ \mu$W. We have conservatively assumed a 100 MHz clock frequency (i.e., the clock period is 10 ns). The additional delay introduced by the encoder and decoder on the processor-memory path is 4.35 ns, which amounts to less than 50% of the clock period. The supply voltage is $V_{dd} = 3.3$ V.

We now define the *equivalent capacitance* for the encoder and decoder as the capacitance that, when switched at every clock cycle, causes a power dissipation equal to that of the encoder and decoder. In symbols

$$C_{\text{Enc/Dec}} = \frac{2P_{\text{Enc/Dec}}}{V_{dd} \cdot f} = 1.852\ \text{pF}.$$

Clearly, the adoption of the Beach solution becomes convenient if it reduces the average switched capacitance on the bus by more than $C_{\text{Enc/Dec}}$.

Let us call C_l the capacitance of each bus line, and assume that all lines have the same capacitance. The average total

switched capacitance on the bus with the standard binary encoding is

$$C_{\text{Bin}} = \frac{2714472}{464856} \cdot C_l = 5.839 \cdot C_l.$$

With our encoding, such capacitance becomes

$$C_{\text{Beach}} = \frac{787715}{464856} \cdot C_l = 1.694 \cdot C_l.$$

Therefore, the savings in switched capacitance per clock cycle we get by applying the Beach solution amounts to $4.145 \cdot C_l$. Then, the encoding becomes advantageous when $4.145 \cdot C_l > C_{\text{Enc/Dec}}$, or, equivalently:

$$C_l > \frac{1.852 \text{ pF}}{4.145} = 0.446 \text{ pF}.$$

This is not a large value even for an on-chip bus line [19]. In fact, a typical value of capacitance per unit length for a 1.5 μm wide wire is 0.2 fF/μm. Thus, a 1 cm wire has a capacitance of 2 pF. Off-chip lines have generally much larger capacitance values [20]. We conclude that, for this case study, the Beach solution provides a viable power optimization technique.

V. Conclusions

We have presented a new solution to the problem of encoding the address bus of microprocessor-based systems to reduce the total bus switching activity. The proposed technique, unlike existing approaches, is strongly application-dependent. Therefore, it is suitable for achieving power optimizations in application-specific systems consisting of a core processor or a microcontroller which repeatedly executes a special-purpose program.

Given the address bus traces corresponding to the execution of the embedded program, statistical information identifying possible block correlations between the bus lines is collected and exploited to group the bus lines into clusters. The encoding and decoding functions are then generated for each cluster, and the corresponding logic circuits, to be placed at the address bus terminals, are automatically synthesized and optimized.

Experimental results have shown that large improvements over application-independent encoding schemes can be achieved with the proposed technique. In addition, the size, the delay, and the power consumption of the synthesized encoding and decoding circuits have shown to be small enough to make the Beach solution applicable in practice.

Acknowledgment

The authors wish to thank L. Lavagno for the C code of the `dashb` example and F. Ferrandi and F. Fummi for the VHDL code of the core processor.

References

[1] M. Stan and W. P. Burleson, "Limited-weight codes for low-power," in *Proc. IWLPD-94: IEEE/ACM Int. Workshop Low Power Design*, Napa Valley, CA, Apr. 1994, pp. 209–214.

[2] C. L. Su, C. Y. Tsui, and A. M. Despain, "Saving power in the control path of embedded processors," *IEEE Design Test Comput.*, vol. 11, pp. 24–30, Winter 1994.

[3] 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. GLS-VLSI-97: IEEE Great Lakes Symp. VLSI*, Urbana, IL, Mar. 1997, pp. 77–82.

[4] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano, "Address bus encoding techniques for system-level power optimization," in *Proc. DATE-98: IEEE Design Automation Test Europe*, Paris, France, Feb. 1998, pp. 861–866.

[5] 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.

[6] J. L. Hennessy and D. A. Patterson, *Computer Architecture—A Quantitative Approach*, 2d Ed. New York: Morgan Kaufmann, 1996

[7] E. Musoll, T. Lang, and J. Cortadella, "Exploiting the locality of memory references to reduce the address bus energy," in *Proc. ISLPED-97: ACM/IEEE Int. Symp. Low Power Electron. Design*, Monterey, CA, Aug. 1997, pp. 202–207.

[8] S. Wuytack, F. Catthoor, L. Nachtergaele, and H. De Man, "Global communication and memory optimizing transformations for low power design," in *Proc. IWLPD-94: ACM/IEEE Int. Workshop Low Power Design*, Napa Valley, CA, Apr. 1994, pp. 203–208.

[9] ———, "Power exploration for data dominated video applications," in *Proc. ISLPED-96: ACM/IEEE Int. Symp. Low Power Electron. Design*, Monterey, CA: Aug. 1996, pp. 359–364.

[10] J. P. Diguet, S. Wuytack, F. Catthoor, and H. De Man, "Formalized methodology for data reuse exploration in hierarchical memory mappings," in *Proc. ISLPED-97: ACM/IEEE Int. Symp. Low Power Electron. Design*, Monterey, CA, Aug. 1997, pp. 30–35.

[11] P. R. Panda and N. D. Dutt, "Reducing address bus transitions for low power memory mapping," in *Proc. EDTC-96: IEEE European Design Test Conf.*, Paris, France, Mar. 1996, pp. 63–67.

[12] ———, "Low power mapping of behavioral array to multiple memories," in *Proc. ISLPED-96: ACM/IEEE Int. Symp. Low Power Electron. Design*, Monterey, CA, Aug. 1996, pp. 289–292.

[13] J. Heinrich, *MIPS R4000 Microprocessor User's Manual*, 2nd Ed., MIPS Technologies, 1994.

[14] G. D. Hachtel, M. Hermida, A. Pardo, M. Poncino, and F. Somenzi, "Re-encoding sequential circuits to reduce power dissipation," in *Proc. ICCAD-94: IEEE/ACM Int. Conf. Computer-Aided Design*, San Jose, CA, Nov. 1994, pp. 70–73.

[15] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291–307, Feb. 1970.

[16] C. Passerone, L. Lavagno, C. Sansoé, M. Chiodo, and A. Sangiovanni, "Trade-off evaluation in embedded system design via co-simulation," in *Proc. ASP-DAC-97: IEEE Asia South Pacific Design Automation Conf.*, Chiba, Japan, Jan. 1997, pp. 291–297.

[17] F. Ferrandi, F. Fummi, E. Macii, M. Poncino, and D. Sciuto, "Testing core-based digital systems: A symbolic methodology," *IEEE Design Test Comput.*, vol. 13, pp. 69–77, Winter 1997.

[18] *SpeedChart Project Designer User's Manual*, Version 3.2.0, Speed S.A., 1995.

[19] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, 2nd Ed. Reading, MA: Addison-Wesley, 1992.

[20] H. Johnson and M. Graham, *High-Speed Digital Design: A Handbook of Black Magic*. Englewood Cliffs, NJ: Prentice Hall, 1993.

Luca Benini received the Dr.Eng. degree in electrical engineering from Università di Bologna, Italy, in 1991, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1994 and 1997, respectively.

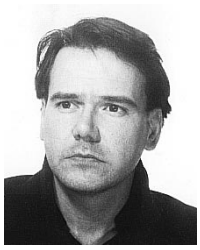
Since 1997, he has been a Research Associate at Università di Bologna and a Postdoctoral fellow at Stanford University. He also holds a position as Visiting Scientist at the Hewlett-Packard Laboratories, Palo Alto, CA. His research interests are in all aspects of computer-aided design of digital circuits, with special emphasis on low-power applications.

Giovanni De Micheli (S'79–M'82–SM'89–F'94) is Professor of Electrical Engineering, and by courtesy, of Computer Science at Stanford University, Stanford, CA. His research interests include several aspects of the computer-aided design of integrated circuits and systems, with particular emphasis on automated synthesis, optimization, and validation. He is the author of *Synthesis and Optimization of Digital Circuits* (New York: McGraw-Hill, 1994), coauthor of *Dynamic Power Management: Circuit Techniques and CAD Tools* (New York: Kluwer, 1998), and of three other books.

Dr. De Micheli received the 1987 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN/ICAS Best Paper Award and two Best Paper Awards at the Design Automation Conference in 1983 and 1993, respectively. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS.

Enrico Macii (M'92) received the Dr.Eng. degree in electrical engineering from Politecnico di Torino, Italy, in 1990, the Dr.Sc. degree in computer science from Università di Torino, Italy, in 1991, and the Ph.D. degree in computer engineering from Politecnico di Torino in 1995.

From 1991 to 1994, he was an Adjunct Faculty Professor at the University of Colorado, Boulder. Currently, he is an Assistant Professor at Politecnico di Torino. His research interests include synthesis, verification, and simulation and testing of digital circuits and systems.



Massimo Poncino (M'97) received the Dr.Eng. degree in electrical engineering in 1989 and the Ph.D. degree in computer engineering in 1993, both from Politecnico di Torino, Italy.

From 1993 to 1994, he was a Visiting Faculty Professor at the University of Colorado, Boulder. Currently, he is an Assistant Professor at Politecnico di Torino. His research interests include synthesis, verification, and simulation and testing of digital circuits and systems.

Stefano Quer received the Dr.Eng. degree in electrical engineering and the Ph.D. degree in computer engineering from Politecnico di Torino, Italy, in 1991 and 1996, respectively.

Since 1995, he has been receiving a Postdoctoral fellowship at the Computer Engineering Department of Politecnico di Torino. His research interests include logic synthesis, formal verification, and simulation and testing of digital circuits and systems.