System-Level Power Optimization of Special Purpose Applications: The Beach Solution

Luca Benini [#] Giovanni De Micheli [#] Enrico Macii [‡] Massimo Poncino [‡] Stefano Quer [‡]

[#] Stanford University
Computer Systems Laboratory
Stanford, CA 94305

[‡] Politecnico di Torino
Dip. di Automatica e Informatica
Torino, ITALY 10129

Abstract

This paper describes a new approach to low-power bus encoding, called "The Beach Solution", which is thought for power optimization of digital systems containing an embedded processor or a microcontroller executing a special-purpose software routine. The main difference between the proposed method and existing bus encoding techniques is that it is strongly application-dependent, in the sense that it is based on the analysis of the execution stream of a given program. This allows an accurate computation of the correlations that may exist between blocks of bits in consecutive patterns, and that can be successfully exploited to determine an encoding which minimizes the bus transition activity. Experimental results, obtained on a set of special-purpose applications, are very promising; reductions of the bus activity up to 64.8% (41.9% on average) have been achieved over the original address streams.

1 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 are obviously at the basis of 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 limited economical effort and quality penalty.

In this paper, the focus is on the design of low-power, special-purpose systems. More specifically, we face the problem of reducing the power dissipated by a digital design containing an embedded 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 required at the input/output pins of a processor when binary patterns have to be transmitted over the communication channel. More precisely, it has been estimated that the capacitance driven by the processor's input/output nodes is usually much larger (up to three orders of magnitude [1]) than the one seen by the internal nodes of the processor. 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 input/output pins of the processor.

This task can be accomplished by encoding the binary patterns transmitted over the bus. Depending on the type of information to be exchanged, several low-power encoding schemes, exploiting distinctive spectral characteristics of the pattern streams have been proposed recently.

Stan and Burleson have introduced the use of the *bus-invert code* [2]. 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, because streams are typically composed of bursts of addresses in sequence, intermingled with out-of-sequence addresses (corresponding to taken branches and jumps) [3]. The high frequency of consecutive patterns can be fruitfully exploited by dedicated (e.g., *Gray* [4, 5] and *T0* [6]) or mixed encoding schemes [7]. Clearly, if the percentage of in-sequence addresses decreases, the effectiveness of the aforementioned codes diminishes as well (see Figure 1, where the average number of transitions per bus line is plotted for streams in which out-of-sequence addresses are inserted with controlled probability).

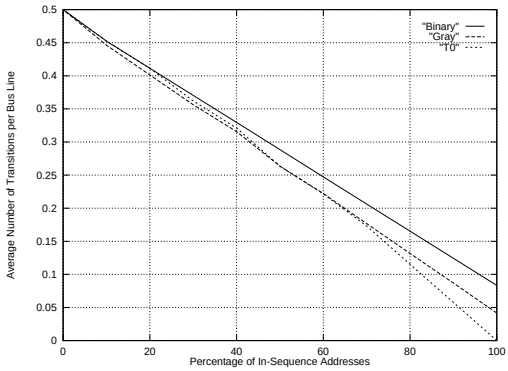

Figure 1: Performance of Binary, Gray, and T0 Codes.

We have experimentally observed that addresses generated by real microprocessors running general-purpose programs, such as compilers, word processors, and data compression tools, are usually characterized by high sequentiality. On the contrary, streams for special-purpose software applications (e.g., image processing, matrix calculus, automotive control) have a much smaller percentage of in-sequence addresses. As an example, in Table 1 we report statistics on the streams generated by the MIPS R4000 RISC microprocessor [8], configured in single-user mode, for some benchmark programs: The left half of the table contains the data for general-purpose programs, while the right half presents the data for software functions which are often conveniently implemented in hardware with dedicated machines.

Program	*In-Seq. Addr.*	*Appl.*	*In-Seq. Addr.*
ghostview	58.2%	dashb	38.3%
gzip	57.4%	dct	44.1%
latex	55.5%	fft	43.6%
matlab	65.3%	mm	39.8%
oracle	59.1%	qsort	38.2%
xdvi	63.3%	vm	44.8%

Table 1: In-Sequence Addresses for Different Applications.

The observation that the majority of the address streams in special-purpose systems have reduced sequentiality makes the use of codes such as the Gray and the T0 ineffective when the target is power optimization. This is because the savings achievable through switching activity reduction are easily offset by the increase in power caused by the insertion of the encoding/decoding logic at the bus boundaries.

Intuitively, it may well be the case that other types of temporal correlations exist between the patterns that are being transmitted. More specifically, it has been noted 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 intra-segment/page branches and jumps are much more frequent than inter-segment/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 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, that is, lines belonging to the same cluster are highly correlated. For each cluster we automatically generate an encoding function, namely a one-to-one Boolean function. Each bit configuration in the original cluster is translated into a new one. The algorithm which finds the encoding function targets the minimization of the switching activity; thus, well established technology, initially developed for logic synthesis applications [9, 10, 11, 12], can be successfully exploited. The output of the transformation is an encoded stream for which the average number of bus line transitions between two successive patterns is minimized. At the receiving end of the bus, the original encoding is obviously 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 deconding 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.

[1] The name comes from the place, *The Beach Club*, where the algorithm was initially conceived by the authors during DAC'96 in Las Vegas.

2 The Beach Solution

Our solution differs from previously proposed 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, multi-user, and multi-tasking computing systems, where several programs (possibly characterized by address streams of substantially different nature) can run concurrently. On the other hand, it has proved to be particularly suitable to special-purpose machines, where the same portion of embedded code is executed over and over.

2.1 Overview

A high-level block diagram of the basic operations required to apply The Beach Solution is depicted in Figure 2.

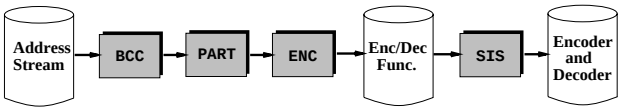

Figure 2: The Beach Solution.

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 Computer), 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 clusters 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 logic originates the encoder and decoder circuitry.

2.2 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 re-encoded to patterns with similar codes, i.e., codes with a minimum Hamming distance (possibly equal to one). Theoretically, 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, W_i, $i = 1, 2, ..., L$, a set of 3-tuples (W_F, W_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 W_F, the second one is W_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 proportional to the length of the stream L (in the worst case). 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 $W(t) = (x_1^{(t)}, x_2^{(t)}, ..., x_N^{(t)})$ 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., τ, or at two different but consecutive clock cycles, i.e., τ_1 and $\tau_2 = \tau_1 + 1$.

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

$$\mu_{\chi_i} = \frac{\sum_{t=0}^{L-1} \chi_i}{L} \qquad \sigma_{\chi_i} = \frac{\sum_{t=0}^{L-1} \chi_i^2}{L} - \mu_{\chi_i}^2$$

Then, we define the *covariance* of $\chi_i^{\tau_1}$ and $\chi_j^{\tau_2}$ as:

$$Cov(\chi_i^{\tau_1}, \chi_j^{\tau_2}) = \frac{1}{L-1} \sum_{t=0}^{L-1} \chi_i^{\tau_1} \chi_j^{\tau_2} - \left(\frac{1}{L-1} \sum_{t=0}^{L-1} \chi_i^{\tau_1}\right) \cdot \left(\frac{1}{L-1} \sum_{t=1}^{L} \chi_j^{\tau_1}\right) \quad (1)$$

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

$$\rho_{i,j} = \frac{Cov(\chi_i^{\tau_1}, \chi_j^{\tau_2})}{\sigma_{\chi_i} \sigma_{\chi_j}} \quad (2)$$

Setting τ_2 equal to τ_1 or equal to $\tau_1 + 1$ 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 setting $\tau_2 = \tau_1$ and $i \neq j$, expresses the likelihood of correctly predicting the value of one bit of pattern W_i knowing the values of one other bit in the same pattern. The second type, obtained by setting $\tau_2 = \tau_1 + 1$ and $i = j$, expresses the likelihood of correctly predicting the value of a bit in pattern W_i by observing its value on the previous pattern. In general, $\tau_2 \neq \tau_1$ and $i \neq j$ and we have the *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 for 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 bits*. A transition bit is defined as follows:

$$\eta_i^{\tau} = +1 \cdot (x_i^{\tau} \cdot (x_i^{\tau-1})') - 1 \cdot (x_i^{\tau})' \cdot x_i^{\tau-1}$$

The transition bit η_i^{τ} has value 1 if bit x_i makes a raising transition from clock cycle $\tau - 1$ to τ. It has value -1 in case of a falling transition, and it is zero otherwise. We can compute the switching correlation covariance and the switching correlation coefficient between transition bits η_i and η_j ($i \neq j$) with Equations 1 and 2, by replacing χ with η.

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 are completely lost. Consequently, it is not possible to claim that switching correlation is always the best measure, and the results of our experiments have confirmed this fact.

The analysis based on pairwise correlations produces three $N \times N$ matrices. The complexity of the procedure is $O(N^2 L)$, because the entire stream (of length L) has to be analyzed. However, we do not need to store the stream in memory. The correlations can be computed on the fly by a filter-like program.

2.3 Clustering Heuristics

The pairwise correlation coefficients, computed with one of the methods outlined in Section 2.2, can be collected in a *correlation matrix* $\mathbf{C}$. The correlation matrix can be seen as the adjacency matrix of a weighted, directed graph $G(V, E, W)$, where the vertices represent bits in the bus, the weighted edges represent pairwise correlations, and the weights are the elements of $\mathbf{C}$.

We exploit the information on correlations contained in G, or equivalently $\mathbf{C}$, to extract subsets of bits that are suitable to be encoded. 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 non-uniform. If this is the case, encoding can be very effective in reducing the average number of concurrent transitions for bits in the cluster. Clearly, we cannot allow excessively large clusters, because 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.

We propose two algorithms for clustering. The first one is based on the computation of the strongly connected components (SCCs) of graph G. First, we apply a *threshold* on the correlation matrix. All elements with absolute value smaller than a user-defined $|\rho_{min}|$ are set to zero. This is useful to filter out correlations that are not statistically significant. The clusters are simply the SCCs of the graph after thresholding. The main limitation of this algorithm lies in the lack of control on the size of the clusters. Since there is no control on cluster size, in many cases we obtain excessively large clusters and we cannot tune the algorithm by specifying the granularity of the partition.

To overcome this limitation, we have developed a customized clustering procedure that allows the user to specify the maximum cluster size. The algorithm is heuristic in nature, but we have experimentally observed that it produces high-quality partitions. The high-level operation flow of our new clustering procedure is the following:

- Build an undirected weighted graph where the weights are obtained as $\rho_{i,j} + \rho_{j,i}$.
- Put a threshold on the edge weights: If $|\rho_{i,j}| < |\rho_{min}|$ delete the edge (i.e., set its weight to zero).
- Partition the graph in clusters with high mutual correlation and user-specified maximum size k_{max}. We use the following simple pairwise clustering strategy:
  - Select the edge with maximum weight.
  - Cluster the head and tail of the edge together into a single vertex.
  - Store in the clustered vertex the number k of original vertices in it ($k = 2$ in the first step).
  - If a clustered vertex has $k = k_{max}$, eliminate that vertex and all the edges connected to it.
  - Continue until the graph is empty or no edge is left.

The partitioning algorithm returns a set of clusters with number of elements bounded by k_{max}. The run time of the algorithm is linear in the number of vertices, N. The clusters are the starting points for the encoding algorithm that is described in the next section.

2.4 Synthesis of the Encoding/Decoding Logic

The clusters obtained using the partitioning algorithm of Section 2.3 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 1 *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 000, 001, 100, and 111. The graph Q has then four vertices: $S_1 = 001$, $S_2 = 000$, $S_3 = 111$, and $S_4 = 100$. If in the sample there are 120 transitions $000 \rightarrow 001$ and 268 transitions $001 \rightarrow 000$, the weight on the edge between vertices S_1 and S_2 is 388. The resulting graph Q is shown in Figure 3.*

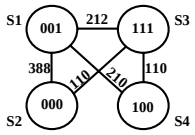

Figure 3: Transition Graph Example.

The output of the PART program (see Figure 2) 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:

$$\text{Cost} = \sum_{\text{each vertex pair } i,j} Z_{i,j} \cdot H_{i,j} \qquad (3)$$

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

Example 2 *The cost for the graph of Figure 3 is 1782, as shown in Figure 4 (left part). By re-encoding vertex S_3 from 111 to 101, we have now that S_3 is "closer" to vertex S_1, S_2 and S_4. The total amount of transitions between the vertex pairs (S_3, S_1), (S_3, S_2) and (S_3, S_4) is thus reduced, and the value of the new cost function is 1350.*

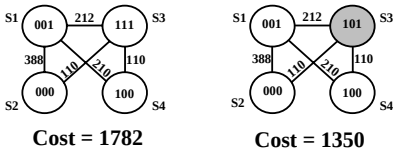

Figure 4: Re-Encoding To Minimize the Number of Transitions.

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 re-encode a graph, since it provides a large slack that can be exploited in re-assigning codes to the various vertices.

The ENC tool re-encodes the transition graph of each cluster. It uses the algorithm proposed in [12], which is fully based on implicit representations of Boolean and pseudo-Boolean (i.e., real-valued) functions by means of BDDs and ADDs. The re-encoding problem, whose exact solution is NP-hard, is solved heuristically; this is acceptable, since our purpose is to handle graphs whose sizes are larger than the ones that can be managed by traditional (i.e., based on explicit representation of the graph) methods, rather than obtaining an exact solution.

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 [13] partitioning algorithm. The two heuristics can be used to trade-off accuracy for memory requirements of the BDD/ADD representation; 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 re-encoding information is given as a set of *re-encoding functions* $E = (E_1, \ldots, E_{k_i})$, where each function E_j expresses each bit j as a function of all the other bits. In other terms, the re-encoding can be thought of as an input/output relation $E = E(x_1, \ldots, x_{k_i}, y_1, \ldots, y_{k_i})$ which binds re-encoded 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 BDDs, 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 trivial, since it suffices to swap the set of x's and y's, that is, $E^{-1}(x_1, \ldots, x_{k_i}, y_1, \ldots, y_{k_i}) \equiv E(y_1, \ldots, y_{k_i}, x_1, \ldots, x_{k_i})$. Again, dumping the BDD representation to a file as a network of multiplexors, and then applying logic optimization, yields the netlist for the decoder.

3 Experimental Results

In this section, we report some experimental data concerning the use of The Beach Solution to reduce power consumption. In particular, we first present a brief case study in which we examine the impact of some user-selectable parameters on the quality of the results produced by our encoding scheme. Then, we show data regarding the reduction in switching activity that we have obtained for software programs commonly implemented as special-purpose computing systems. Some information on the speed and the power consumption of the encoding/decoding logic is also provided in order to give a feeling on what kind of cost must be afforded when our solution is adopted. All the experiments have been executed on the MIPS R4000 RISC 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. However, it is important to stress the fact that our approach is completely general, and it is thus applicable to architectures having different organizations of the input/output interfaces (e.g., separated instruction/data address buses).

3.1 Case Study

The first analysis we perform deals with the impact that the length of the address streams may have on the computation of bit correlations and, in turn, on the reduction of the number of transitions occurring on the address bus. To do that, we have run a C routine implementing the insertion-sort algorithm on three vectors of different sizes and containing randomly generated integer numbers. The ST correlation measure has been chosen to drive the clustering phase, and clusters of size 8 have been used.

Table 2 reports the results of the experiments, from which it is easy to evince how both the percentage of in-sequence addresses and the reduction of switching activity with respect to the original (i.e., binary) address representation are quite insensitive from the length of the stream. This nice behavior has been observed for almost all the examples we have considered, indicating that the correlation measures we have proposed have the desirable capability of capturing the spectral characteristics of the address streams in a very short time. This fact obviously facilitates the overall process of computing the encoding/decoding functions.

Vector Size	*Stream Length*	*In-Seq. Addr.*	*Binary Tran.*	*Beach*	
				Tran.	*Sav.*
50	21212	46.32%	181589	78454	56.7%
100	69188	46.30%	583289	264239	54.6%
150	141879	46.28%	1194553	538656	54.9%
200	275306	46.24%	2320427	1042075	55.1%

Table 2: Case Study: Address Streams of Varying Lengths.

Regarding the type of correlation measure and the size of the clusters used to determine the encoding/decoding functions, we have performed an extensive experimentation. As expected, the larger the clusters, the better the transition savings. This behavior is simple to understand: Grouping together many bus lines reduces the loss of information induced by the clustering. Quite surprisingly, on the other hand, the type of correlation measure which produces the best results depends on the specific application. This result is somewhat unexpected, and indicates that even more sophisticated and powerful statistical techniques than the ones we have proposed here may be needed to increase the analytical strength of the `CCM` tool (see Figure 2.1). A sample of the experimental data we have collected is shown in Table 3; it refers to the insertion-sort routine considered earlier in this section, and applied to a 50-element vector of random numbers (see the first row of Table 2 for the number of transitions of the unencoded address stream).

Cluster Size	*Correlation Measure*					
	S		*ST*		*SW*	
	Tran.	*Sav.*	*Tran.*	*Sav.*	*Tran.*	*Sav.*
4	102093	43.7%	95124	47.6%	94782	47.8%
6	98584	45.7%	83289	54.1%	93760	48.3%
8	96901	46.6%	78454	56.7%	84693	53.3%

Table 3: Case Study: Correlation Measures and Cluster Sizes.

3.2 Special Purpose Systems

We have selected a set of software functions which are usually implemented in hardware as parts of 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.

Table 4 reports the outcome of our investigation. For each application, we give the length of the address streams considered for the experiment, the percentage of in-sequence addresses, the number of bus transitions when no encoding, that is, the binary code, is used, the number of bus transitions after encoding, and the savings achieved with respect to the unencoded case. Notice that the data 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 the reason why, for example, the numbers in column *In-Seq. Addr.* slightly differ from the ones reported in Table 1, where a single run has been considered. Results are highly satisfactory; in fact, a 41.9% average savings has been obtained, with a peak improvement of 64.8% for the `vm` example. Running times of the overall encoding procedure are always within a few minutes.

In the right-most columns of Table 4 we show the best results produced by the following codes: Gray, T0, Bus_Invert, T0_Bus+Bus_Invert, Dual_T0, and Dual_T0+Bus_Invert. The comparison is clearly in favor of The Beach Solution.

Appl.	*In-Seq. Addr.*	*Binary Tran.*	*Beach*		*Others*	
			Tran.	*Sav.*	*Tran.*	*Sav.*
dashb	39.1%	619690	443115	28.4%	486200	21.5%
dct	45.4%	48917	31472	35.6%	39327	19.6%
fft	43.4%	138526	85653	38.1%	100127	27.7%
mm	40.4%	105947	60654	42.7%	77384	26.9%
qsort	38.8%	182673	96306	42.2%	129235	29.2%
vm	45.1%	133272	46838	64.8%	91194	31.5%
Avg.				41.9%		26.0%

Table 4: Results for Special Purpose Systems.

dashb:	Car Dashboard Controller [14].
dct:	Discrete Cosine Transform.
fft:	Fast Fourier Transform.
mm:	Matrix Multiplication.
qsort:	Quick Sort for Vectors of Integers.
vm:	Vector by Vector Scalar Multiplication.

An issue which cannot be neglected regards the complexity, and thus the speed and the power consumption, of the encoding/decoding logic which must be added at the bus ends. In Table 5, we report the characteristics of the circuits (that is, gate count, power in μW, and delay in $nsec$) obtained through automatic synthesis and optimization of the encoding/decoding functions. The implementations have been generated using SIS; since delay is the most critical constraint for these circuits, the `script.delay` and the `map -n1 -AFG` commands have been used for logic optimization and technology mapping onto a 1.0 μm, 5 *Volt* gate-library containing buffers and inverters with three different strengths, and NAND/NOR gates with up to four inputs. Power estimates have been calculated using the IRSIM-CAP transistor-level simulator.

Appl.	*Circuit*	*Gates*	*Power*	*Delay*
dashb	Encoder	153	187	8.90
	Decoder	177	124	8.98
dct	Encoder	1297	462	20.07
	Decoder	1211	308	15.25
mm	Encoder	489	349	12.54
	Decoder	473	125	17.63
fft	Encoder	272	325	8.03
	Decoder	304	257	9.59
qsort	Encoder	125	149	5.86
	Decoder	124	130	6.60
vm	Encoder	304	282	10.93
	Decoder	268	137	10.70

Table 5: Encoder and Decoder Implementations.

By looking at the data in the table, and in consideration of previous experience on the subject [6, 7], we can claim that power and delay of the encoders/decoders enable the application of The Beach Solution for power optimization of ordinary off-chip buses, as the ones used in core-based systems.
Since after application of our technique the spectral characteristics of the new streams are different from the original ones, one may be tempted to apply a further step of encoding using one of the already existing methods. Unfortunately, on average, the percentage of in-sequence addresses in the new streams tends to decrease remarkably; therefore, encoding schemes such as the Gray and the T0 are not expected to produce any benefit. On the contrary, some additional savings might be achieved through application of techniques which rely, at least in part, on the bus-invert principle.
In Table 6 we report the data we have obtained on the same applications of Table 4 by cascading the bus-invert encoding scheme to The Beach Solution. Results are absolutely negative, since improvements in switching activity are negligible in most of the cases; potential power savings are then likely to be offset by the additional cost introduced by the heavy bus-invert encoding/decoding logic.

Appl.	*In-Seq. Addr.*	*Tran.*	*Beach+Bus-Invert*	
			Tran.	*Sav.*
dashb	0.1%	443115	428245	3.4%
dct	19.7%	31472	31044	0.1%
fft	0.0%	85653	85653	0.0%
mm	29.2%	60654	58123	0.1%
qsort	0.0%	96306	96253	0.1%
vm	27.4%	46838	46835	0.0%
Avg.				0.6%

Table 6: Cascade Application of Beach and Bus-Invert Codes.

4 Conclusions and Future Work

We have presented a new low-power bus encoding technique that, unlike the existing approaches, is strongly application-dependent, and is based on the analysis of the execution streams of a program. Experimental results have shown the effectiveness of the proposed solution in application-specific systems consisting of embedded processors or microcontrollers, and executing special-purpose application.
Although our encoding algorithm is quite successful, there are still margins of improvement. First, our procedure for finding clusters of bits suitable for encoding is highly heuristic and driven by approximate information (only pairwise correlations are considered). Several experiments revealed that clustering is paramount to achieve good results and more powerful clustering strategies may greatly improve the power savings. Second, the synthesis procedure for the encoder and decoder can be improved as well; one direction could be that of resorting to Zero-Suppressed BDDs [15] for directly incorporating some synthesis-oriented criteria in the translation from the functional representation to the circuit description, as done in [16]. Finally, we are investigating the applicability of The Beach Solution to symbolic encoding problems such as selection of power-optimal opcodes for instructions or microcode power minimization.

Acknowledgments

We wish to thank Luciano Lavagno for providing us with the dashb example, and Riccardo Scarsi for helping us with IRSIM.

References

[1] P. R. Panda, N. D. Dutt, "Reducing Address Bus Transitions for Low Power Memory Mapping," *EDTC-96: IEEE European Design and Test Conference*, pp. 63-67, Paris, France, March 1996.

[2] M. R. Stan, W. P. Burleson, "Bus-Invert Coding for Low-Power I/O," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 3, No. 1, pp. 49-58, March 1995.

[3] J. L. Hennessy, D. A. Patterson, *Computer Architecture - A Quantitative Approach*, Second Edition, Morgan Kaufmann Publishers, 1996

[4] C. L. Su, C. Y. Tsui, A. M. Despain, "Saving Power in the Control Path of Embedded Processors," *IEEE Design and Test of Computers*, Vol. 11, No. 4, pp. 24-30, Winter 1994.

[5] H. Mehta, R. M. Owens, M. J. Irwin, "Some Issues in Gray Code Addressing," *GLS-VLSI-96: IEEE 6th Great Lakes Symposium on VLSI*, pp. 178-180, Ames, IA, March 1996.

[6] L. Benini, G. De Micheli, E. Macii, D. Sciuto, C. Silvano, "Asymptotic Zero-Transition Activity Encoding for Address Busses in Low-Power Microprocessor-Based Systems", *GLS-VLSI-97: IEEE 7th Great Lakes Symposium on VLSI*, pp. 77-82, Urbana, IL, March 1997.

[7] L. Benini, G. De Micheli, E. Macii, D. Sciuto, C. Silvano, "Address Bus Encoding Techniques for System-Level Power Optimization," *EuroDAC-97: IEEE European Design Automation Conference*, Dusseldorf, Germany, November 1997, To Appear.

[8] J. Heinrich, *MIPS R4000 Microprocessor User's Manual*, Second Edition, MIPS Technologies, Mountain View, CA, 1994.

[9] K. Roy, S. C. Prasad, "Circuit Activity Based Synthesis for Low Power Reliable Operations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 1, No. 4, pp. 503-513, December 1993.

[10] L. Benini, G. De Micheli, "State Assignment for Low Power Dissipation," *IEEE Journal of Solid State Circuits*, Vol. 30, No. 3, pp. 258-268, March 1995.

[11] C. Y. Tsui, M. Pedram, A. M. Despain, "Low Power State Assignment Targeting Two- and Multi-Level Logic Implementations," *ICCAD-94: IEEE/ACM International Conference on Computer-Aided Design*, pp. 82-87, San Jose, CA, November 1994.

[12] G. D. Hachtel, M. Hermida, A. Pardo, M. Poncino, F. Somenzi, "Re-Encoding Sequential Circuits to Reduce Power Dissipation," *ICCAD-94: IEEE/ACM International Conference on Computer-Aided Design*, pp. 70-73, San Jose, CA, November 1994.

[13] B. W. Kernighan, S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Technical Journal*, Vol. 49, pp. 291-307, February 1970.

[14] C. Passerone, L. Lavagno, C. Sansoé, M. Chiodo, A. Sangiovanni-Vincentelli, "Trade-Off Evaluation in Embedded System Design via Co-simulation," *ASP-DAC-97: IEEE Asia South-Pacific Design Automation Conference*, pp. 291-297, Chiba, Japan, January 1997.

[15] S-I. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," *DAC-30: ACM/IEEE Design Automation Conference*, pp. 272-277, Dallas, TX, Jun. 1993.

[16] B. Kumthekar, I. H. Moon, F. Somenzi, "A Symbolic Algorithm for Low-Power Sequential Synthesis," *ISLPED-97: ACM/IEEE International Symposium on Low Power Electronics and Design*, Monterey, CA, August 1997.