Clustered Loop Buffer Organization for Low Energy VLIW Embedded Processors.

Citation for published version (APA):

Jayapala, M., Barat, F., Aa, van der, T., Catthoor, F., Corporaal, H., & Deconinck, G. (2005). Clustered Loop Buffer Organization for Low Energy VLIW Embedded Processors. *IEEE Transactions on Computers*, 54(6), 672-683. <https://doi.org/10.1109/TC.2005.92>

DOI:

[10.1109/TC.2005.92](https://doi.org/10.1109/TC.2005.92)

Document status and date:

Published: 01/01/2005

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Clustered Loop Buffer Organization for Low Energy VLIW Embedded Processors

Murali Jayapala, *Student Member, IEEE*, Francisco Barat, *Student Member, IEEE*,
Tom Vander Aa, *Student Member, IEEE*, Francky Catthoor, *Fellow, IEEE*,
Henk Corporaal, and Geert Deconinck, *Senior Member, IEEE*

Abstract—Current loop buffer organizations for very large instruction word processors are essentially centralized. As a consequence, they are energy inefficient and their scalability is limited. To alleviate this problem, we propose a clustered loop buffer organization, where the loop buffers are partitioned and functional units are logically grouped to form clusters, along with two schemes for buffer control which regulate the activity in each cluster. Furthermore, we propose a design-time scheme to generate clusters by analyzing an application profile and grouping closely related functional units. The simulation results indicate that the energy consumed in the clustered loop buffers is, on average, 63 percent lower than the energy consumed in an uncompressed centralized loop buffer scheme, 35 percent lower than a centralized compressed loop buffer scheme, and 22 percent lower than a randomly clustered loop buffer scheme.

Index Terms—RISC/CISC, VLIW architectures, real-time and embedded systems, memory management, memory design, low-power design.

1 INTRODUCTION

CURRENT embedded systems for multimedia applications, such as mobile and hand-held devices, are typically battery operated. Therefore, low energy is one of the key design goals for such systems. Many such systems often rely on very long instruction word (VLIW) application specific instruction set processors (ASIPs) [1]. However, power analysis of such processors indicates that a significant amount of power is consumed in the instruction caches [2], [3]. For example, in the TMS320C6000, a VLIW processor from Texas Instruments, up to 30 percent of the total processor energy is consumed in the instruction caches alone [2]. Loop buffering (or L0 buffering) is an effective scheme to reduce energy consumption in the instruction memory hierarchy [4], [5]. In a typical multimedia application, a significant amount of execution time is spent in small program segments. Hence, by storing them in a small L0 buffer instead of a conventionally large instruction cache, energy can be reduced. Furthermore, by adopting a counter-based indexing mechanism, the expensive tags of the loop cache can be eliminated [6]. By coupling compiler optimizations with the L0 buffer organization, the energy efficiency can be further enhanced [7], [8]. In particular, the

authors of [8] have shown that up to 90 percent of the operations can be fetched from a 256-operation L0 buffer, thus reducing the energy per transfer of instruction.

In this context, our contributions are the following: 1) We propose a clustered L0 buffer organization, as shown in Fig. 1, where the L0 buffers are partitioned and functional units are logically grouped to form instruction clusters,¹ along with two schemes for the buffer control which regulate the activity in each cluster. 2) The formation of clusters at design time is steered by functional unit activity of a given application instead of grouping arbitrarily. In this paper, we present results that are application specific. However, this cluster generation technique can also be applied over an application domain.

The rest of the paper is organized as follows: An overview of key motivations for a clustered organization is presented in Section 2. The proposed clustered organization is described in Section 3. The profile-based clustering algorithm is outlined in Section 4. In Section 5, a detailed analysis of the proposed schemes is provided. In Section 6, related work is discussed. Finally, in Section 7, a brief summary is given and future work is outlined.

2 MOTIVATIONS

Thus far, the L0 buffer organizations proposed and analyzed in the literature are, to a large extent, centralized, i.e., a single logical cluster is assumed and a single controller controls the indexing into the buffer to store and fetch instructions. However, such an organization in the context of VLIW processors is energy inefficient and its scalability is limited. First, the wordlines of the buffers

• M. Jayapala, F. Barat, T. Vander Aa, and G. Deconinck are with ESAT/ELECTA, Kasteelpark Arenberg 10, K.U. Leuven, Leuven-Heverlee, Belgium 3001.

E-mail: {mjayapala, fbarat, tvandera, gdec}@esat.kuleuven.ac.be.

• F. Catthoor is with IMEC vzw, Kapeldreef 75, Leuven-Heverlee, Belgium 3001. E-mail: catthoor@imec.be.

• H. Corporaal is with the Electrical Engineering Department, Technical University Eindhoven (TU/e), Den Dolech 2, 5612 Eindhoven, The Netherlands. E-mail: h.corporaal@tue.nl.

Manuscript received 13 Feb. 2004; revised 2 Aug. 2004; accepted 8 Oct. 2004; published online 15 Apr. 2005.

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-0050-0204.

Fig. 1. The clustered L0 buffer organization.

should be at least as wide as the number of issue slots or the number of functional units (FUs) in the datapath in order to provide a desired throughput of one instruction per cycle. Realistically, in an embedded VLIW processor like the TI C6x series from Texas Instruments [9], this width would be about 256 bits (eight FUs with 32-bit operations). Even if the L0 buffers store compressed instructions (NOP Compression), the width of the buffer still needs to be as wide as the uncompressed case in order to provide the necessary best case throughput. With an increase in the number of FUs, the width of the wordlines is bound to increase. In general, memories with wide wordlines tend to be energy inefficient. Partitioning or subbanking is a known technique to avoid long wordlines. However, these techniques are applied at the microarchitectural level or at the hardware level. In contrast, we propose raising the notion of partitioning to the architectural level, where certain features of the application can be exploited to achieve higher energy efficiency. Since we expose the partitions at the architectural level, certain necessary extensions to the local controllers have to be made. In the next section, we propose two schemes for the local controllers.

3 CLUSTERED L0 BUFFER ORGANIZATION

The essentials of the proposed clustered L0 buffer organization are illustrated in Fig. 1. The L0 buffers are partitioned and grouped with certain FUs in the datapath to form an instruction cluster or an L0 cluster. In each cluster, the buffers store only the operations of a certain loop destined to the FUs in that cluster. Furthermore, the buffers are placed close to the FUs. By closeness, it is meant that the latency of transfer of the instructions from the buffers to the FUs is minimal and also the physical distances between the buffers and FUs in a cluster are as small as possible.

The operation of the clustered L0 organization is as follows: By default, the L0 buffers are not accessed during the normal phase of the execution. Parts of the program that are to be fetched from L0 buffers should be marked explicitly either by the programmer or the compiler. A special instruction, *lbon* (loop buffer on), should be inserted at the beginning of the program segment along with the number of

TABLE 1
Characteristics of the Benchmarks

Benchmark	Type	Avg ILP (in loops)	Max Size insts(ops)
adpcm dec	Audio Decoding	4.10	22 (176 ops)
adpcm enc	Audio Encoding	2.65	22 (176 ops)
cavity detector	Image Processing	1.50	26 (208 ops)
g721 dec	Audio Decoding	4.10	12 (96 ops)
g721 enc	Audio Encoding	4.00	51 (408 ops)
gsm	Wireless	2.75	55 (440 ops)
jpeg enc	Image Encoding	2.38	38 (304 ops)
mpeg2 dec	Video Decoding	2.80	47 (376 ops)

instructions in the program segment. The program segment can be any loop with conditional constructs, nested loop, or even parts of loops. By arranging the code in a proper layout, any generic program segment can be mapped. For our analysis, we have chosen small loops that have significant weight in the program execution (refer to Table 1). An example illustrating this process is shown in Fig. 2. Here, a loop is explicitly marked by the compiler to be mapped onto the L0 buffers and, also, the number of instructions in the loop (five instructions) is indicated.

3.1 Filling Clustered L0 Buffers

Once the instruction containing the *lbon* operation is encountered during the program execution, the processor pipeline is stalled and the instructions that immediately follow *lbon* are prefetched and distributed over the different L0 partitions. The number of instructions prefetched will be as indicated in the *lbon* operation (five instructions in the example illustrated). Alternatively, clever prefetching schemes can be adopted in order to avoid the stalls. However, we do not consider any such schemes for the analysis in this paper. For every instruction pre-fetched, the instruction dispatch stage issues the operations to their corresponding clusters. Once the instructions are stored in the L0 buffers, the execution is resumed, with instructions now being fetched from L0 buffers. The dispatch logic does not decode the operations, but partially decodes the instructions to extract operations for each cluster. Here, we assume that this logic is very small and neglect it for further analysis. Additionally, the buffers can also be used to store decoded operations. However, this decision requires analysis of the instruction encoding and the trade-off between sizes of L0 buffers before and after decoding. This analysis is beyond the scope of this paper.

Alternatively, the L0 buffers can be filled with the instructions of the loop by simultaneously feeding the

Fig. 2. A part of the program segment mapped on to the L0 buffers.

Fig. 3. L0 buffer operation with activation-based control scheme.

Fig. 4. L0 buffer operation with activation-based control along with index translation.

datapath during the first iteration of the loop, thus avoiding the stall cycles. However, this alternative is suitable only for loops without conditional constructs. For a loop with conditional constructs, some of the basic blocks may not be executed in the first iteration. In the worst case, one of the basic blocks may not be executed till the last iteration. In this scenario, instructions would still be fetched from expensive L1 cache instead of L0 buffers. However, this can be solved to some extent by employing code transformation techniques like function in-lining [10] and loop splitting [8].

3.2 Regulating Access

One of the key features of our clustered organization is that we can restrict the accesses to partitions that are not active in an instruction cycle. We achieve this by providing an *activation trace* (AT) in the local controller (ITC) of each cluster. While operations of each instruction in the loop are prefetched and distributed among the partitions, a one or a zero is stored in the activation trace register, indicating that the partition is active or inactive, respectively. Fig. 3 shows the activation trace for the example illustrated in Fig. 2. For instance, during the execution of the third instruction of the loop, partitions one and four are active, while partitions two and three are inactive. Thanks to this activation trace we can now restrict the access to partitions two and three through the "enable" signal, thus saving energy consumption.

3.3 Indexing into L0 Buffer Partitions

In order to store and fetch the instructions, indexes that point to appropriate locations in each L0 partition have to be

generated. One of the following two schemes can be adopted for the index generation. In the first scheme, a common index (NEW_PC in Fig. 3) is generated for all the L0 partitions. This index is derived directly from the program counter as $NEW_PC = fn(PC, START_ADDRESS)$. Having only one index for all the L0 partitions implies that the operations of an instruction that are stored in different partitions have to be stored in identical locations in the corresponding cluster. For instance, the third instruction of the example illustrated in Fig. 2 has two operations op13 and br'x' stored in L0 partitions one and four at location two. Although only two operations are stored, the corresponding locations in L0 partitions, two and three, cannot be reused to store operations of other instructions. Furthermore, this also implies that the number of words in each partition has to be identical. One of the advantages of this scheme is that the index generation is simple and its implementation can be heavily optimized, but this comes at the expense of inefficient storage utilization.

In the second scheme, instead of only one index for all the partitions, separate indexes for each L0 partition are generated and stored in an index translation table (ITT) (refer to Fig. 4). Here, a counter (not shown in figure) keeps track of the next free location available in each partition and this is incremented only when an operation is stored in that partition. Furthermore, all the ITTs are in turn indexed by the NEW_PC, which is generated as described above. The operation of this indexing scheme is illustrated in Fig. 4. For instance, the operations of the third instruction in the above example are stored in locations one in the first partition and

one in the fourth partition, while nothing is stored in partitions two and three, thus utilizing the storage space more efficiently than the first scheme. However, this efficiency comes at the expense of increased complexity and cost of index translation in each partition. Unlike the previous scheme, where only one index is used for all the partitions, the local controller in this scheme requires a storage for the index translation of width = $\log_2(\text{Depth of L0 Buffers in a partition})$ in addition to the activation trace and depth of $\max(\#instructions \text{ mapped})$.

3.4 Fetching from L0 Buffers or L1 Cache

When the *lbon* instruction is encountered during execution, the address location of the first instruction of the loop and the address of the last instruction of the loop are stored in the start and end registers provided in the Loop Buffer Control or the LBC (not shown in the figure). When the program counter points to a location within this address range, the instructions are fetched from the L0 buffers instead of the L1 cache. The signal *L0 buffer enable* (or *L1 Cache Disable*) in Fig. 1 selects the appropriate inputs of the multiplexers and enables or disables the fetch from the L1 cache.

The start register is comparable to a tag in conventional caches. Typically, when the instruction *lbon* is encountered during execution, the start address of the loop body following that instruction is compared with the start address already stored in the start register. If there is a match, then the instructions that are already stored in L0 buffers are used. On the other hand if there is a mismatch, only then are the instructions of the loop body following the *lbon* instruction prefetched and stored in the buffers. This prevents unnecessary refetching of the same instructions.

For the above example (Fig. 2), a detailed illustration of the operation of clustered L0 buffers with two schemes of controller is provided in the Appendix.

4 PROFILE-BASED CLUSTERING

Essentially, two aspects are important in generating clusters. Namely, the access pattern to the memories and the trade-off between the energies of the L0 buffers and the local controllers.

At the architectural level, we can exploit certain features of the application, namely, the access pattern to the memories. The basic observation which aids in clustering is that, typically, in an instruction cycle, not all the FUs are active. For instance, in a schedule of a certain instruction cycle, it is conceivable that four operations are mapped for a certain datapath of eight FUs. Let us also assume that the operations are scheduled to FUs 1, 3, 4, and 8. Now, these FUs could be grouped in many ways, of which four relevant cases are illustrated in Fig. 5. In case 1, FUs 1 and 3 are grouped into one cluster, FUs 4 and 8 are grouped into another cluster, and the remaining FUs 2, 5, 6, and 7 in grouped in another cluster. In case 2, FUs 1 and 2 grouped in one cluster, FUs 3 and 4 are grouped in another cluster, and FUs 5, 6, 7, and 8 are grouped in another cluster. In case 1, only two accesses are needed to two small clusters. However, in case 2, three accesses are needed to all three

Fig. 5. Motivation for clustering: importance of access patterns and trade-off.

clusters. Cluster configuration in case 1 is more energy efficient than case 2 since a lower number of accesses are needed to smaller clusters. Without the knowledge of the access pattern to the memory, it was not possible to recognize that case 1 is better than case 2. Had it been only at the microarchitectural level (case 2), the better configuration of case 1 could have been unnoticed.

In case 3, all the FUs are grouped into a single cluster with a buffer storing the corresponding operations. In case 4, each FU is grouped in a separate cluster with a buffer storing the corresponding operation. In case 3, one large buffer and one local controller are needed, while, in case 4, one local controller for each FU is needed. There is a trade-off between the local controller cost and the buffer cost. The reduction in the buffer sizes and the number of accesses those buffers should compensate for adding more local controllers.

The example in Fig. 5 illustrates the clustering possibilities for just one instruction. However, all the instructions that are mapped onto the L0 buffers have an effect on clustering. The tool described in the remainder of this section explores the two aspects described above for a given program.

The process of generating L0 clusters is as follows (refer to Fig. 6). For a given profile (dynamic and static), the L0 buffer is partitioned and the functional units are grouped into clusters so as to minimize energy consumption. This problem is formulated as a 0-1 assignment optimization problem. The formulation is as follows:

$$\begin{aligned} & \min \{ L0Cost(L0clust, D Prof_{loops}, S Prof_{loops}) \} \\ & \text{Subject To } \sum_{i=1}^{N_{max_CLUST}} L0clust_{ij} = 1; \forall j, \end{aligned}$$

Fig. 6. Functional unit activity-based clustering algorithm.

where

$$L0Clust_{ij} = \begin{cases} 1 & \text{if } j\text{th FU is assigned} \\ & \text{to cluster } i \\ 0 & \text{otherwise} \end{cases}$$

N_{FU} total number of FUs

$N_{maxCLUST}$ max number of feasible clusters = N_{FU}

(At most each FU can be in an $L0cluster$).

The $L0Cost(L0clust, DProf_{loops}, SProf_{loops})$ represents the energy consumption in the L0 buffers for any valid clustering. The result of this optimization is that a centralized uncompressed L0 buffer is partitioned (with optimal sizes for each partition) and the functional units are grouped together to form L0 clusters. The grouping of functional units is represented in the matrix $L0Clust_{ij}$. The implementation of this optimization can be done in several ways. An exhaustive search is feasible for a small number of FUs. However, for a large number of FUs, an approximation algorithm could be used. The detailed description of the implementation is beyond the scope of this paper.

$DProf_{loops}$ is the dynamic profile of an application. This contains the activity (1 if active, 0 if inactive) of each FU in each cycle during execution of loops. For a given set of FUs, the total number of accesses to the L0 buffer partition corresponding to these FUs can be estimated by analyzing the pattern of 1s. Based on the parametric model for the L0 buffer, energy per access can be estimated for a given size. Based on these two values, the energy of L0 buffers in each cluster can be estimated (refer to Section 5).

$SProf_{loops}$ is the static profile of an application. This contains an instruction map of all the loops mapped onto the L0 buffers. For each instruction, it contains a series of 1s and 0s, one for each FU. If an operation is issued to the corresponding FU, a 1 is marked or a 0 otherwise. The loop boundaries of all the mapped loops are also marked. Based on this profile, for a given set of FUs, the depth of the L0 buffer partition in all the clusters for the scheme in Fig. 3 can be estimated as the maximum number instructions among all the loops mapped to the L0 buffer. And, the depth of the L0 buffer partition in each cluster for the

scheme in Fig. 4 can be estimated by analyzing the pattern of 1s corresponding to the FUs in each cluster.

To remove the effects of data dependency, an average profile can be generated over multiple runs of the application with different input data. For evaluation and analysis in this paper, the profiles are generated per application, hence the results presented are application specific. However, using some statistical techniques, an average profile can be generated over all of the applications in an *application domain*. By using these profiles, the technique presented in this paper can be used as is to generate domain specific solutions instead of application specific solutions.

5 EVALUATION AND ANALYSIS

For our evaluation to be realistic, we have modeled the L0 buffer organization based on a known embedded VLIW processor from the TI C6x processor series [9], with eight FUs (eight issue slots) and an instruction width of 256 bits with 32-bit operations for each FU. Using the compiler and simulator of the Trimaran tool suite [11], applications were mapped onto this processor model and simulated to generate the profiles. The compiler, in particular, has been extended to identify loops which have less than 512 operations (64 instructions) and which have significant weight in the execution time, to be mapped onto the L0 buffers.

Since our domain of interest is embedded multimedia applications, we have chosen the benchmarks for our evaluation from Mediabench [12]. Some characteristics of these benchmarks are shown in Table 1.

The energy consumption of the L0 buffers and the local controllers is represented by the equation

$$E = \sum_{i=1}^{N_{clusters}} (E_i * N_i + LC_i),$$

where E_i is the energy consumed for any random access, N_i is the number of accesses made during the program execution, and LC_i is the local controller energy per cluster. For all L0 buffers and the local controllers, the E_i are obtained by modeling them as single read, single write port register files in Wattch [13] in a 0.18 μm technology.

5.1 Energy Reduction Due to Clustering

Clustering the storage at an architectural level aids in reducing the energy consumption in two ways. First, smaller and distributed memories can be employed. Second, at the architectural level, an explicit control over the accesses to these memories can be imposed (through the local controller).

As described in Section 3, with the aid of the ITT, the depths of the L0 partitions can be optimized independently in each partition. This corresponds to reduction in effective buffer energy per access ($\sum E_i$). Fig. 7 shows the reduction in effective buffer energy per access² for increasing the number of clusters. For instance, when the number of clusters is equal to four, the effective buffer energy per accesses is reduced by

2. For a single cluster, the energy for AT+ITT is slightly more than the energy for AT. This difference is due to the additional address decoder used for the buffer instead of one-hot encoding.

Fig. 7. Reduction of effective buffer energy per access (ΣE_i) due to the index translation table (ITT) with an increasing number of clusters ($N_{clusters}$).

about 20 percent. We see that, with the increase in the number of clusters, the effective buffer energy per access reduces and it is minimal when the number of clusters is equal to the number of functional units.

By restricting the accesses to the buffers, we can reduce the amount of switching energy in the L0 buffers. Fig. 8 shows the reduction in the effective number of accesses (ΣN_i) with the increase in the number of clusters. Here, the effective number of accesses is defined as the sum of all the accesses per functional unit. We see that, with the increase in the number of clusters, the effective accesses keep on reducing and are minimal when the number of clusters is equal to the number of functional units. The reduction is fairly intuitive because, with the increase in the number of clusters, the degree of control over the effective number of accesses per functional unit increases and, when each functional unit has its own buffer partition, this degree is maximal.

The aforementioned reductions reduce the buffer energy. However, this reduction is traded off against the increase in local controller energy. Fig. 9 summarizes the trade-off between the buffer energy ($\Sigma E_i * N_i$) and the local controller

Fig. 8. Reduction in the effective number of accesses (ΣN_i) due to activation trace (AT) with an increasing number of clusters ($N_{clusters}$).

Fig. 9. Reduction in buffer energy and increase in local controller energy.

(ΣLC_i) for the two proposed schemes and Fig. 10 shows the total energy reduction for the two schemes. Since, for the scheme represented by Fig. 4, buffer energy is reduced due to both regulating the accesses and reducing the effective size, this reduction is greater than the energy reduced for the scheme represented by Fig. 3, where only accesses are regulated. As expected, the local controller energy in the former is larger than the local controller energy in the latter due to increased complexity. However, Fig. 10 shows that, in some cases, increased complexity in the local controller pays off against reductions in buffer energy.

5.2 Energy Reduction Due to Closely Related Functional Unit Grouping

By grouping closely related functional units to form a cluster, we can reduce the energy further for any given number of clusters. The variation of total energy in L0 buffers, including the overhead of the local controllers, is shown in Fig. 12. The curve corresponding to legend "Random" is obtained by generating clusters randomly³ and the curve corresponding to legend "FU Grouping" is obtained by generating clusters using the algorithm presented in Section 4. Fig. 11 shows the reduction obtained by grouping closely related functional units against randomly clustered for four clusters for the proposed organization represented by Fig. 4. On average, 22 percent of energy can be reduced additionally over random grouping.

This additional reduction can be explained as follows: First, by grouping closely related functional units, the effective buffer energy per access (ΣE_i) for a certain clustering can be reduced. For instance, when $N_{clusters} = 4$, the effective buffer energy per access is reduced by an additional 10 percent. Second, by grouping closely related functional units, the effective number of accesses (ΣN_i) can also be reduced. For instance, when $N_{clusters} = 4$, the effective number of accesses is reduced by an additional 20 percent. Fig. 12 shows the summary of the reduction in energy by grouping closely related functional units over random clustering for an increasing number of clusters. Here, the corresponding energies have been averaged over all the benchmarks under consideration.

3. By random, we mean not using any knowledge about functional units activity or specialization.

Fig. 10. Reduction in total energy for two L0 buffer schemes.

5.3 Proposed Organization versus Centralized Organizations

We have evaluated two centralized L0 buffer schemes, namely, a centralized uncompressed scheme and a centralized compressed scheme, against our proposed organizations, a clustered L0 buffer with activation trace and a clustered L0 buffer with activation trace and index translation. Fig. 13 summarizes the energy reductions of various schemes. On average, the energy consumption in the proposed clustered organization is about 63 percent lower than the energy consumed in an uncompressed centralized scheme and about 35 percent lower than the energy consumed in a centralized compressed scheme.

For the centralized uncompressed scheme, the size of the L0 buffer, for each application, will be the maximum number of instructions among all the loops that were identified by the compiler. However, Table 1 indicates that the average ILP is typically less than the width of eight operations per word in the L0 buffer and, hence, the L0 buffer is unnecessarily large and energy inefficient.

In contrast, a centralized compressed L0 buffer efficiently utilizes the storage and the depth of the L0 buffer can be made smaller. For the benchmark "mpeg2dec," we observed that the depth of the L0 buffer could be reduced from 47 to 18. This reduction comes from the fact that the instructions are of variable length and the operations in an instruction are tightly packed, eliminating the NOPs. Here, we have adopted the instruction fetch model from the TI C6x processor series,

Fig. 11. Energy reduction by random clustering and closely related functional unit clustering (for $N_{clusters} = 4$).

Fig. 12. Reduction in the total energy ($\Sigma(E_i * N_i + LC_i)$) of the L0 buffer schemes for random clustering and functional unit grouping.

where every fetch to the L0 buffer partition fetches an instruction packet of eight operations. This packet is stored in an additional buffer and the operations are fed to the datapath from this buffer every instruction cycle. A new instruction packet is fetched only when operations in the additional buffers are used up. Based on this model, we can see that, on average, 44 percent of energy can be reduced over an uncompressed centralized scheme. The number of fetches to the L0 partition is reduced significantly, but at the expense of adding an additional buffer. However, in most cases, this overhead is compensated by the reductions in the L0 buffer except for one particular benchmark, g721dec. For this benchmark, the energy reduction in the L0 buffer (reduction in depth) was not sufficient to compensate for the overhead (refer to Fig. 13).

In the clustered scheme with AT and ITT as opposed to clustered scheme with AT, in addition to reducing the number of accesses in each partition, the depths of the L0 buffers in each partition can be further optimized. This reduction in the L0 buffer size comes at the expense of increased complexity and energy consumption of the controller. However, this increase in energy is just large enough not to be compensated by the reduction in the L0 buffer energy. Fig. 13 shows that the energy consumption of clustered organization as proposed in Fig. 4 is slightly more than the energy consumption of the clustered organization proposed in Fig. 3. In our analysis of the clustered organizations, we have assumed that only one type of local controller is used throughout. However, a hybrid scheme could also be employed where some clusters have an activation trace while others have both an activation trace and an index translation table. Currently, we have not made any analysis regarding the hybrid scheme and we leave such an analysis for future work.

5.4 Performance Issues

From Section 3.4, we can deduce that the number of cycles lost by stalls due to prefetching depends on the number of instructions in the loops that are mapped to the L0 buffers. However, in comparison with the number of cycles in which the instructions in the loops are executed, the stall cycles are

Fig. 13. Energy consumption of clustered organization in comparison with other schemes.

negligible. From Fig. 14, we can observe that the performance degradation due to prefetching is less than 5 percent.

In the clustered organization as shown in Fig. 4, two storage blocks have to be accessed sequentially in one cycle. While this requirement may seem to constrain the cycle time, in reality it does not. In embedded processors which operate at low frequencies (100MHz range), two storage blocks can be accessed within one instruction cycle. For instance, in the benchmark “gsm,” the L0 buffer size of one partition is about 3kbits and the corresponding size of the local controller is about 0.5kbits. For the register file model in (0.18 μm technology), the access times to the buffer and the controller are about 2.5 ns and 2.0 ns, respectively. Together, the critical-path length is about 4.5 ns, translating to about 250 MHz, which is about the same as the operating frequency of some of the TI C6x processor series in 0.15 μm technology [2]. However, even if the access times are not within the critical path length of the processor, the L0 buffer access in the proposed scheme can be *pipelined*. In the first stage, the local controller can be accessed to get the activation and the index, while, in the second stage, the operations stored in the buffer can be retrieved.

6 RELATED WORK

Many complementary approaches have been proposed to reduce energy consumption in different aspects of the instruction memory hierarchy through different levels of

system abstraction [14]. Several bus encoding schemes [15], [16], [17] have been applied to reduce the effective switching on the (instruction and address) buses, thus saving energy. Code size reduction techniques, both hardware [18], [19], [20] and software [21], [22], [23], in addition to saving energy in buses (due to smaller widths and less traffic), reduce the size of the program memory and thus reduce energy. On the other hand, software transformations [24], [25] aiming at utilizing the underlying memory hierarchy efficiently have also been applied in the context of instruction memory.

In a more direct relation to the concepts presented in this paper, the available literature falls under two broad categories. The first category encompasses the literature available in relation to L0 buffers or loop buffers, which is one of the central concepts of our proposed organization. We give an overview of different flavors of L0 buffer organization and indicate that our approach is complementary to most of them. The second category encompasses the literature available in relation to partitioned or decentralized organization. We give an overview of different partitioned organizations, especially in relation to the instruction memory and the processor front end.

The concept of using small buffers has been applied to optimize both performance and energy. Jouppi [26] has studied the performance advantages of small prefetch buffers or stream buffers. On the other hand, the reduction of energy using small buffers was first observed by Bunda [27], and this idea was more generalized by Kin et al. as Filter cache [28]. The authors have shown that up to 58 percent of instruction memory power can be reduced with a performance degradation of about 21 percent. To mitigate the loss in performance, Tang et al. [29] have proposed a hardware predictive filter cache. Alternatively, the authors in [5], [4], [30] proposed using these buffers only for loops, thus reducing the loss in performance while still retaining the large reductions in energy.

Since the identification of loops to be mapped onto the L0 buffers is largely hardware controlled and dynamic, loops with small iteration count could also be mapped onto

Fig. 14. Performance degradation due to filling in L0 buffers.

the L0 buffer leading to thrashing. Gordon-Ross and Vahid [31] have analyzed this situation and they propose a preloaded loop cache, where the loops with large instruction counts are identified by profiling and only these are mapped on to the loop cache. Furthermore, their scheme also support loops with control constructs and various levels of nesting.

In a partitioned organization [32], [33], a buffer is divided into smaller partitions in order to reduce the wordline width. However, the process of partitioning is largely arbitrary. The operations of a certain functional unit are not necessarily bound to a few partitions, they can be placed in any of the partitions. Thus, no correlation exists between the process of partitioning and the functional unit activity. A correlation between the two should be explicitly imposed in order to physically place the partitions over different functional units in the datapath and ease the constraints on the interconnect. Otherwise, an operation for a functional unit may need to be fetched from a partition which is physically placed close to a different functional unit, thus constraining the interconnect severely. In this sense, we follow a partitioning or clustering scheme which is different and at a higher level of abstraction than the conventional partitioning scheme.

At a conceptual level, the clustered L0 buffer organization proposed in this paper is similar to an n -way associative cache with way-prediction [34] or even horizontally partitioned caches with cache-prediction [35]. In associative caches, instructions are stored in different "ways," while, in our case, the instructions are stored in different clusters. The way-predictors predict the "ways" that are to be accessed in any instruction cycle, while, in our case, the local controllers regulate the accesses to each cluster. In spite of these similarities, the underlying details of associative caches and clustered L0 buffers are different. First, the way prediction schemes are much more complex than the local controller schemes proposed in this paper and they still rely on tags for addressing. Second, most of the way, prediction schemes have been applied in the context of hardware controlled caches and, thus, there is a possibility of misprediction. However, in our case, the L0 buffers are software mapped and, hence, the activation of each partition can be known beforehand, thus avoiding any misprediction.

In traditional clustered VLIW processors, the notion of clustering is applied to minimize the complexity of the register files (datapath clusters) [36], [37]. In recent years, this notion has also been applied to minimize the complexity of the front end (instruction fetch) in the form of multi-VLIWs [38], [39]. However, in such multi-VLIWs, an instruction cluster (L0 cluster) is always synonymous with a datapath cluster. In contrast, we differentiate between an instruction cluster and a datapath cluster. In a datapath cluster, the functional units derive "data" from a single register file. In an instruction cluster, the functional units derive "instructions" from a single L0 buffer. Even though, in both cases, the main aim of partitioning is to reduce energy (power) consumption, the principle of partitioning is different and the decisions can be taken independently.

Superscalar processors are high-performance (*GHz range*) and high power consuming (*10-100W*) desktop oriented processors. On the other hand, embedded processors are high-performance (*100Mhz range*) and at least two to three orders of magnitude lower power consuming (*0.1-2W*). Their processor characteristics vary significantly [40]. However, the notion of clustering is also seen in some of the superscalar processors. Here, we mention only a few clustered (decentralized) architectures. Zyuban and Kogge [41] have analyzed the effects of clustering the front end of a superscalar processor, particularly on energy. They also propose a complexity-effective multicenter architecture that is inherently energy efficient. Many other research groups have proposed some form of decentralized organizations [42], [43]. However, their primary concern was mainly performance and not energy costs.

7 SUMMARY AND FUTURE WORK

In summary, we have presented a clustered L0 buffer organization in the context of VLIW processors for low energy embedded systems, with two different schemes to control the activation and indexing of the L0 buffer partition in each cluster. Additionally, unlike the conventional partitioned schemes where the clustering is largely arbitrary, we follow a clustering scheme which is steered by the functional unit activity in a given application. Through simulations, we demonstrated that the energy consumed in the clustered L0 buffers is, on average, 63 percent lower than the energy consumed in an uncompressed centralized L0 buffer scheme, 35 percent lower than a centralized compressed L0 buffer scheme, and 22 percent lower than a randomly clustered L0 buffer scheme.

Currently, the generation of L0 clusters is performed as an architectural optimization, where the compiler generates a schedule and, based on the given schedule, L0 clusters are generated. Since the result of the clustering depends on the given schedule, it offers an interesting design space to explore the effects of clustering by altering the schedule to increase energy efficiency. As part of our future work, scheduler algorithms for L0 clusters will be investigated. Additionally, datapath clusters and L0 clusters can coexist. However, current schemes for generating datapath clusters and L0 clusters are mutually exclusive and the resulting clusters might be in conflict. The synchronicity between datapath and L0 clusters needs to be investigated in more detail.

APPENDIX

Fig. 15 illustrates the clustered L0 buffer operation with only activation trace. For simplicity, each FU is assumed to have a separate L0 buffer partition. A sample loop and its corresponding schedule are shown at the top of the figure.

When the instruction *lbon* is encountered during the execution, the operations within the loop are distributed into the corresponding clusters. After this initiation, at the end of CYCLE N, the operations are now fetched from the L0 buffers. During the first cycle (refer to stage CYCLE N+1 in Fig. 15) of the loop, the NEW_PC indexes into the activation trace. If a "1" is stored at that index, the corresponding L0 buffer is accessed. During this cycle, the fourth cluster is not accessed.

Fig. 15. An illustration of clustered L0 buffer operation with only activation trace.

During the second cycle (refer to stage CYCLE N+2 in Fig. 15), the first cluster is not accessed. Additionally, a branch is encountered, BNZ'X'. Assuming that the result of the branch is available within one cycle and the result indicates the branch to be taken, then the NEW_PC is updated so that, in the next cycle, it points to the appropriate instruction. During the third cycle (refer to stage CYCLE N+3 in Fig. 15), the NEW_PC points to the fourth instruction of the loop and executes the appropriate operations. During the fourth cycle (refer to stage CYCLE N+4 in Fig. 15), the operations of the last instruction are executed. If the execution is not in the last iteration, then the branch points to the first instruction and the execution continues with instructions being fetched from L0 buffers. If the execution is in the last iteration, then the branch points to a location out of the address range of the loop and the instructions are now fetched from the L1 cache.

Fig. 16 illustrates the clustered L0 buffer operation with activation trace and index translation table. The execution is similar to the scheme illustrated in Fig. 15, but for two main differences. First, the sizes of the L0 buffers are optimized according to the active operations in the loop. Second, the NEW_PC indexes into activation trace and an index translation table. For a certain NEW_PC, the index stored

Fig. 16. An illustration of clustered L0 buffer operation with activation trace & index translation table.

in the translation table points to the exact location of the operation to be executed.

ACKNOWLEDGMENTS

This work has been supported in part by MESA under the MEDEA+ program.

REFERENCES

- [1] M.F. Jacome and G. de Veciana, "Design Challenges for New Application-Specific Processors," *IEEE Design & Test of Computers*, special issue on design of embedded systems, Apr.-June 2000.
- [2] Texas Instruments Inc., *TMS320C6000 Power Consumption Summary*, <http://www.ti.com>, Nov. 1999.
- [3] L. Benini, D. Bruni, M. Chinosi, C. Silvano, and V. Zaccaria, "A Power Modeling and Estimation Framework for VLIW-Based Embedded System," *ST J. System Research*, vol. 3, pp. 110-118, Apr. 2002.
- [4] R.S. Bajwa, M. Hiraki, H. Kojima, D.J. Gorny, K. Nitta, A. Shridhar, K. Seki, and K. Sasaki, "Instruction Buffering to Reduce Power in Processors for Signal Processing," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 5, pp. 417-424, Dec. 1997.
- [5] L.H. Lee, W. Moyer, and J. Arends, "Instruction Fetch Energy Reduction Using Loop Caches for Embedded Applications with Small Tight Loops," *Proc. Int'l Symp. Low Power Electronic Design (ISLPED)*, Aug. 1999.

- [6] A. Gordon-Ross, S. Cotterell, and F. Vahid, "Exploiting Fixed Programs in Embedded Systems: A Loop Cache Example," *Proc. IEEE Computer Architecture Letters*, Jan. 2002.
- [7] N. Bellas, I. Hajj, C. Polychronopoulos, and G. Stamoulis, "Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High Performance Microprocessors," *Proc. Int'l Symp. Low Power Electronic Design (ISLPED)*, Aug. 1998.
- [8] J.W. Sias, H.C. Hunter, and W.M.W. Hwu, "Enhancing Loop Buffering of Media and Telecommunications Applications Using Low-Overhead Predication," *Proc. 34th Ann. Int'l Symp. Microarchitecture (MICRO)*, Dec. 2001.
- [9] Texas Instruments Inc., TMS320C6000 CPU and Instruction Set Reference Guide, <http://www.ti.com>, Oct. 2000.
- [10] N. Liveris, N.D. Zervas, D. Soudris, and C.E. Goutis, "A Code Transformation-Based Methodology for Improving I-Cache Performance of DSP Applications," *Proc. Design Automation and Test in Europe (DATE)*, Mar. 2002.
- [11] Trimaran: An Infrastructure for Research in Instruction-Level Parallelism, <http://www.trimaran.org>, 1999.
- [12] C. Lee et al., "Mediabench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," *Proc. Int'l Symp. Microarchitecture*, pp. 330-335, 1997.
- [13] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proc. 27th Int'l Symp. Computer Architecture (ISCA)*, pp. 83-94, June 2000.
- [14] S.V. Adve, D. Burger, R. Eigenmann, A. Rawsthorne, M.D. Smith, C.H. Gebotys, M.T. Kandemir, D.J. Lilja, A.N. Choudhary, J.Z. Fang, and P.-C. Yew, "Changing Interaction of Compiler And Architecture," *Computer*, vol. 30, no. 12, pp. 51-58, Dec. 1997.
- [15] C. Lee, J.K. Lee, and T. Hwang, "Compiler Optimization on Instruction Scheduling for Low Power," *Proc. Int'l Symp. System Synthesis (ISSS)*, Sept. 2000.
- [16] M. Mahendale, S.D. Sherlekar, and G. Venkatesh, "Extensions to Programmable DSP Architectures for Reduced Power Dissipation," *Proc. VLSI Design*, Jan. 1998.
- [17] W.-C. Cheng and M. Pedram, "Power-Aware Bus Encoding Techniques for I/O and Data Busses in an Embedded System," *J. Circuits, Systems, and Computers*, vol. 11, pp. 351-364, Aug. 2002.
- [18] L. Benini, A. Macii, E. Macii, and M. Poncino, "Selective Instruction Compression for Memory Energy Reduction in Embedded Systems," *Proc. Int'l Symp. Low Power Electronic Design (ISLPED)*, Aug. 1999.
- [19] P. Centoducatte, G. Araujo, and R. Pannain, "Compressed Code Execution on DSP Architectures," *Proc. Int'l Symp. System Synthesis (ISSS)*, Nov. 1999.
- [20] H. Lekatsas, J. Henkel, and W. Wolf, "Code Compression for Low Power Embedded System Design," *Proc. Design Automation Conf. (DAC)*, June 2000.
- [21] S. Debray, W. Evans, R. Muth, and B.D. Sutter, "Compiler Techniques for Code Compaction," *ACM Trans. Programming Languages and Systems (TOPLAS)*, vol. 22, pp. 378-415, Mar. 2000.
- [22] A. Halambi, A. Shrivastava, P. Biswas, N. Dutt, and A. Nicolau, "An Efficient Compiler Technique for Code Size Reduction Using Reduced Bit-Width ISAs," *Proc. Design Automation Conf. (DAC)*, Mar. 2002.
- [23] T. Ishihara and H. Yasuura, "A Power Reduction Technique with Object Code Merging for Application Specific Embedded Processors," *Proc. Design Automation and Test in Europe (DATE)*, Mar. 2000.
- [24] S. Steinke, L. Wehmeyer, B.-S. Lee, and P. Marwedel, "Assigning Program and Data Objects to Scratchpad for Energy Reduction," *Proc. Design Automation and Test in Europe (DATE)*, Mar. 2002.
- [25] S. Parameswaran and J. Henkel, "I-Copes: Fast Instruction Code Placement for Embedded Systems to Improve Performance and Energy Efficiency," *Proc. Int'l Conf. Computer Aided Design (ICCAD)*, Nov. 2001.
- [26] N.P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," *Proc. Int'l Symp. Computer Architecture (ISCA)*, May 1990.
- [27] J.D. Bunda, "Instruction-Processing Optimization Technique for VLSI Microprocessors," PhD dissertation, Univ. of Texas at Austin, May 1993.
- [28] J. Kin, M. Gupta, and W.H. Mangione-Smith, "Filtering Memory References to Increase Energy Efficiency," *IEEE Trans. Computers*, vol. 49, no. 1, pp. 1-15, Jan. 2000.
- [29] W. Tang, R. Gupta, and A. Nicolau, "Design of a Predictive Filter Cache for Energy Savings in High Performance Processor Architectures," *Proc. Int'l Conf. Computer Design (ICCD)*, Sept. 2001.
- [30] T. Anderson and S. Agarwala, "Effective Hardware-Based Two-Way Loop Cache for High Performance Low Power Processors," *Proc. Int'l Conf. Computer Design (ICCD)*, Sept. 2000.
- [31] A. Gordon-Ross and F. Vahid, "Dynamic Loop Caching Meets Preloaded Loop Caching—A Hybrid Approach," *Proc. Int'l Conf. Computer Design (ICCD)*, Sept. 2002.
- [32] W.-T. Shiue and C. Chakrabarti, "Memory Exploration for Low Power Embedded Systems," *Proc. Design Automation Conf. (DAC)*, June 1999.
- [33] T.M. Conte, S. Banerjia, S.Y. Larin, and K.N. Menezes, "Instruction Fetch Mechanisms for VLIW Architectures with Compressed Encodings," *Proc. 29th Int'l Symp. Microarchitecture (MICRO)*, Dec. 1996.
- [34] M.D. Powell et al., "Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping," *Proc. 34th Int'l Symp. Microarchitecture (MICRO)*, Nov. 2001.
- [35] S. Kim, N. Vijaykrishnan, M. Kandemir, A. Sivasubramaniam, M.J. Irwin, and E. Geethanjali, "Power-Aware Partitioned Cache Architectures," *Proc. ACM/IEEE Int'l Symp. Low Power Electronics (ISLPED)*, Aug. 2001.
- [36] R. Colwell, R. Nix, J. O'Donnell, D. Papworth, and P. Rodman, "A VLIW Architecture for a Trace Scheduling Compiler," *IEEE Trans. Computers*, vol. 37, no. 8, pp. 967-979, Aug. 1988.
- [37] V. Lapinskii, M.F. Jacome, and G. de Veciana, "High Quality Operation Binding for Clustered VLIW Datapaths," *Proc. IEEE/ACM Design Automation Conf. (DAC)*, June 2001.
- [38] P. Faraboschi, G. Brown, J. Fischer, G. Desoli, and F. Homewood, "Lx: A Technology Platform for Customizable VLIW Embedded Processing," *Proc. 27th Int'l Symp. Computer Architecture (ISCA)*, June 2000.
- [39] J. Sánchez and A. González, "Modulo Scheduling for a Fully-Distributed Clustered VLIW Architectures," *Proc. 29th Int'l Symp. Microarchitecture (MICRO)*, Dec. 2001.
- [40] M.J. Flynn, P. Hung, and K.W. Rudd, "Deep-Submicron Microprocessor Design Issues," *IEEE MICRO*, vol. 19, no. 4, July-Aug. 1999.
- [41] V.V. Zyuban and P.M. Kogge, "Inherently Lower-Power High-Performance Superscalar Architectures," *IEEE Trans. Computers*, vol. 50, no. 3, pp. 268-285, Mar. 2001.
- [42] M. Franklin, "The Multiscalar Architecture," PhD dissertation, Univ. of Wisconsin Madison, Nov. 1993.
- [43] S. Palacharla, N. Jouppi, and J. Smith, "Complexity-Effective Superscalar Processor," *Proc. Int'l Symp. Computer Architecture (ISCA)*, June 1997.

Computer Society, and the ACM.

He is a student member of the IEEE, the IEEE Computer Society, and the ACM.

Murali Jayapala received the Master of Engineering degree in systems science and automation in 1999 from the Indian Institute of Science, Bangalore, India. Currently, at the Katholieke Universiteit Leuven, he is pursuing the PhD degree in applied sciences. His research interests are in the field of low-power embedded systems, focusing on microprocessor architectures, compilers, and automation. He is a student member of the IEEE, the IEEE

Francisco Barat received the engineering degree in telecommunications from the Polytechnic University of Madrid, Spain, in 1999. That same year, he joined the Katholieke Universiteit Leuven, Belgium, where he is currently pursuing the PhD degree in applied sciences. His current research interests are in the field of multimedia embedded systems and include RPGs, microprocessor architectures, compiler design, and low-power optimizations.

Tom Vander Aa received the MSc degree in informatics and the MEng degree in artificial intelligence from the Katholieke Universiteit Leuven, Belgium, in 1998 and 1999, respectively, where he is currently pursuing the PhD degree in applied sciences. His research interests are in the field of multimedia embedded systems, focusing on low-power instruction memory implementations of microprocessor architectures. He is a student member of the IEEE and the IEEE Computer Society.

Francky Catthoor received the engineering degree and the PhD degree in electrical engineering from the Katholieke Universiteit Leuven KU Leuven, Belgium in 1982 and 1987, respectively. Since 1987, he has headed several research domains in the area of high-level and system synthesis techniques and architectural methodologies, all within the Design Technology for Integrated Information and Telecom Systems (DESICS—formerly VSDM) division at the Interuniversity Micro-Electronics Center (IMEC), Heverlee, Belgium. Currently, he is an IMEC fellow. He is a part-time full professor in the Electrical Engineering Department at KU Leuven. In 1986, he received the Young Scientist Award from the Marconi International Fellowship Council. He has been an associate editor for several IEEE and ACM journals, such as the *IEEE Transactions on VLSI Signal Processing*, *IEEE Transactions on Multimedia*, and *ACM Transactions on Design Automation of Electronic Systems*. He was the program chair of several conferences, including ISSS '97 and SIPS '01. He is a fellow of the IEEE and a member of the IEEE Computer Society.

Henk Corporaal received the MSc degree in theoretical physics from the University of Groningen and the PhD degree in electrical engineering (in the area of computer architecture) from Delft University of Technology. Currently, he is a professor of embedded system architectures at the Eindhoven University of Technology (TU/e) in The Netherlands and director of research of DTI, the joint Design Technology Institute of TU/e and NUS (National University of Singapore). Previously he taught at several schools for higher education, worked at the Delft University of Technology in the field of computer architecture and code generation, and has been department head and chief scientist within the DESICS (Design Technology for Integrated Information and Communication Systems) division at IMEC, Leuven, Belgium. He has coauthored many papers in the processor architecture and design area and written a book on a new class of VLIW architectures, the Transport Triggered Architectures.

Geert Deconinck received the MSc degree in electrical engineering and the PhD degree in applied sciences from the Katholieke Universiteit Leuven (KU Leuven), Belgium, in 1991 and 1996, respectively, where he has been an associate professor (hoofddocent) since 2003 and a staff member of the research group ELECTA (Electrical Energy and Computing Architectures) in the Department of Electrical Engineering (ESAT). His research interests include the design and assessment of embedded systems with dependability, real-time, or cost constraints. In this field, he has authored and coauthored more than 140 publications in international journals and conference proceedings. He was a visiting professor (bijzonder gastdocent) at the KU Leuven since 1999 and a postdoctoral fellow of the Fund for Scientific Research-Flanders (Belgium) in the period 1997-2003. In 1995-1997, he received a grant from the Flemish Institute for the Promotion of Scientific-Technological Research in Industry (IWT). He is a Certified Reliability Engineer (ASQ), a member of the Royal Flemish Engineering Society, a senior member of the IEEE, and a member of the IEEE Reliability, Computer, and Power Engineering Societies.

► For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.