

Low-Power Data Forwarding for VLIW Embedded Architectures

Mariagiovanna Sami, *Senior Member, IEEE*, Donatella Sciuto, *Member, IEEE*, Cristina Silvano, *Member, IEEE*, Vittorio Zaccaria, *Student Member, IEEE*, and Roberto Zafalon

Abstract—In this paper, we propose a low-power approach to the design of embedded very long instruction word (VLIW) processor architectures based on the forwarding (or bypassing) hardware, which provides operands from interstage pipeline registers directly to the inputs of the function units. The power optimization technique exploits the forwarding paths to avoid the power cost of writing/reading short-lived variables to/from the register file (RF). Such optimization is justified by the fact that, in application-specific embedded systems, a significant number of variables are short-lived, that is, their liveness (from first definition to last use) spans only few instructions. Values of short-lived variables can thus be accessed directly through the forwarding registers, avoiding writeback to the RF by the producer instruction and successive read from the RF by the consumer instruction. The decision concerning the enabling of the RF writeback phase is taken at compile time by the compiler static scheduling algorithm. This approach implies a minimal overhead on the complexity of the processor control logic and, thus, no critical path increase. The application of the proposed solution to a VLIW embedded core has shown an average RF power saving of 7.8% with respect to the unoptimized approach on the given set of target benchmarks (patent pending owned by ST Microelectronics).

Index Terms—Low-power design, microprocessors, VLSI design.

I. INTRODUCTION

THE embedded processor market is growing rapidly, and the complexity of embedded applications is growing even faster. Recently, very long instruction word (VLIW) architectures have been proposed for high-performance embedded systems as an interesting alternative to more conventional CPUs, to balance performance with hardware complexity and scalability. A VLIW processor can execute, in each clock cycle, a set of explicitly parallel *operations*; this set of operations is statically scheduled to form a VLIW. The performance/complexity tradeoff of such architectures is made possible by a sophisticated instruction-level parallelism (ILP) compiler infrastructure, which aims at scheduling high-performance parallel code at *compile time* rather than at *run time*.

Low-power dissipation is an increasingly relevant requirement for such embedded processors [1], [2]. Low-power de-

sign techniques are widely adopted during microprocessor design to meet the stringent power constraints, while avoiding performance degradations. For high-performance processors, such as VLIW, some low-power solutions target the reduction of the *effective switched capacitance* of each logic gate. The effective switched capacitance of a logic gate is defined as the product of the load capacitance at the output of the gate and its output switching activity.

While the reduction of the effective switched capacitance is suitable for the combinational and in-core sequential part of the processor, custom modules need special techniques for power reduction. Multiported register files (RFs) represent the majority of such processor modules (apart from instruction and data caches) and, as a matter of fact, they consume a portion of the power budget that can reach up 20% [3]. Zyuban *et al.* [4] compare several RF power optimization techniques aiming at the reduction of the power consumption of such RFs and propose a model for the RF energy consumption in each clock cycle that is a weighted sum of the energy of a read and write operation

$$E_{\text{cycle}}(t) = n_w(t) * E_{\text{write}} + n_r(t) * E_{\text{read}} \quad (1)$$

where $E_{\text{cycle}}(t)$ is the RF energy consumption during clock cycle t , $n_w(t)$ ($n_r(t)$) is the number of write (read) accesses to the RF during clock cycle t , and E_{write} (E_{read}) is the energy consumption per write (read) access.

As can be noted, energy consumption is modeled by software-dependent parameters ($n_w(t)$, $n_r(t)$) as well as circuit/architectural level parameters (E_{write} , E_{read}). While various techniques do exist for the reduction of the circuit-level parameters (see [4] for a comprehensive overview), software level parameters have not yet been directly addressed in the low-power literature.

The aim of this paper is to propose an energy optimization technique consisting of the reduction of the number of reads and write accesses per clock cycles exploiting the *forwarding* network of a VLIW processor. Such optimization can be applied to all *short-lived variables*, whose value can be retrieved from the forwarding network without writing back (or reading) to (from) the RF. The approach that we propose is tailored to specific embedded applications, with an a priori known behavior, that expose a significative amount of short-lived variables.

II. BACKGROUND

Forwarding (also called *bypassing* and sometimes *short-circuiting*) is a hardware technique that tries to reduce performance penalties due to the data hazards introduced by the

Manuscript received May 15, 2001; revised February 1, 2002 and March 12, 2002.

M. Sami, D. Sciuto, and V. Zaccaria are with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy (e-mail: sami@elet.polimi.it; sciuto@elet.polimi.it; zaccaria@elet.polimi.it).

C. Silvano is with the Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Italy (e-mail: silvano@dsi.unimi.it).

R. Zafalon is with the Advanced System Technology, STMicroelectronics, Agrate Brianza, Italy (e-mail: roberto.zafalon@st.com).

Digital Object Identifier 10.1109/TVLSI.2002.801617

microprocessor pipeline [5]. The forwarding hardware bypasses the results of previous instructions from the interstage pipeline registers directly back to the function units that require them. To implement the bypassing mechanism, the necessary forwarding paths and the related control logic must be included in the processor design [5], [6]. In general, this technique requires an interconnection topology and multiplexers to connect any output pipeline register to the inputs of any function unit. Data bypassed from an instruction to some function unit in early pipeline stages are then stored in the RF when the instruction reaches the last pipeline stage (i.e., writeback stage).

In general, forwarding paths for VLIW processors present an implementation complexity that can impact the processor critical path. In [7], the authors show such design issues and present different bypassing interconnection networks that can be used for increased levels of connectivity among the functional units of the processor. The analysis includes both frequency of stalls and cycle time penalties and shows that a bypassing network that completely connects all of the function units does not provide an improvement in performance when the cycle time penalty and the area of the chip are taken into account.

The concept of taking advantage of register values that are bypassed during pipeline stages has been combined with the introduction of a small register cache in [8] to improve performance. In this architecture, called “register scoreboard and cache,” the RF is replaced with a register cache and a suitable backing store. If the normal bypass mechanism supplies a sufficient fraction of all register operands, the register cache can be very small. Such cache can reduce the critical path delay due to the RF of typical processor, enabling frequency improvements.

For superscalar processors, a scheme that includes a compiler analysis and an architecture extension to avoid the useless commits of short-lived variables has been proposed in [9]. The advantages provided by this approach have been evaluated by the authors, mainly in terms of reduction of write ports to the RF and decreasing the amount of spill code needed, thus improving execution time. Performance improvements of this solution also have been reported, while the power impact has not been considered.

The concept of dead value information (DVI) in the RF has been introduced in [10], where dead register values can be exploited to aggressively reclaim physical registers used for renaming architectural registers. This allows the physical RF to be smaller and faster and potentially increasing the processor clock rate. The authors propose also to use DVI to eliminate unnecessary save/restore instructions from the execution stream at procedure calls and across context switches.

III. MAIN CONTRIBUTION OF THIS PAPER

The main goal of the paper is to define an optimization technique that exploits data forwarding for short-lived variables to save power in VLIW pipelined architectures. The basic idea consists of reducing the RF activity by avoiding power-expensive writing of short-lived variables through the exploitation of interstage registers and forwarding paths. Short-lived variables are simply stored by the producer instruction in the interstage registers and fed to the consumer instruction by exploiting the

Cycle	Instruction
I_n	$\$r2 \leftarrow \dots;$
I_{n+1}	$\dots;$
I_{n+2}	$\dots \leftarrow \$r2;$
I_{n+3}	$\$r2 \leftarrow \dots;$

Fig. 1. Fragment of code showing a short-lived variable.

forwarding paths. No writeback to the RF is performed by the producer instruction, and no read from the RF is affected by the consumer instruction.

In our approach, the decision to enable the RF writeback phase is anticipated at compile time by the compiler static scheduler. This approach requires a small modification to the control logic of the processor, i.e., additional decoding logic and one-gate delay to control the write enable signal of the RF. To apply this optimization to a generic register R , the compiler must compute the *liveness length* L of the n th assignment to R , defined as the distance between its n th assignment and its last use. This information allows the compiler to decide if it must be stored in the RF for further use or if its use is in fact limited within few clock cycles. In the latter case, the variable is *short-lived*, and its value can be passed as an operand to next instructions by using the forwarding paths, thus avoiding having to write it back to the RF.

As an application example, let us consider the fragment of code in Fig. 1. In this example, the writeback of $\$r2$ in I_n can be avoided, and the successive use in I_{n+2} can be performed directly from the forwarding network since the *liveness* of $\$r2$ is equal to two instructions.

The proposed architectural optimization becomes particularly attractive in some classes of embedded applications, where the register liveness analysis (see Section IV-A) has shown that more than half of the total register definitions are limited to the next two instructions.

To evaluate the proposed approach, we have set up an experimental methodology to evaluate the power savings on a set of multimedia applications executed by an industrial VLIW embedded processor, jointly developed by Hewlett-Packard and STMicroelectronics [11] for which an optimized compiler has been developed to exploit the architecture parallelism. Preliminary evaluations of the proposed approach have been reported in [12].

This paper is organized as follows. The proposed low-power forwarding architecture for VLIW processors is presented in Section IV. Section V discusses the problem of exception handling. Section VI shows the results of the analysis of the register liveness on a set of embedded digital signal-processing algorithms and the power savings achievable on an industrial VLIW processor. Concluding remarks and future directions of our work are reported in Section VII.

IV. LOW-POWER FORWARDING ARCHITECTURE

In this section, we show the hardware and software issues involved in the proposed low-power optimization technique, outlining the problems addressed and their solutions.

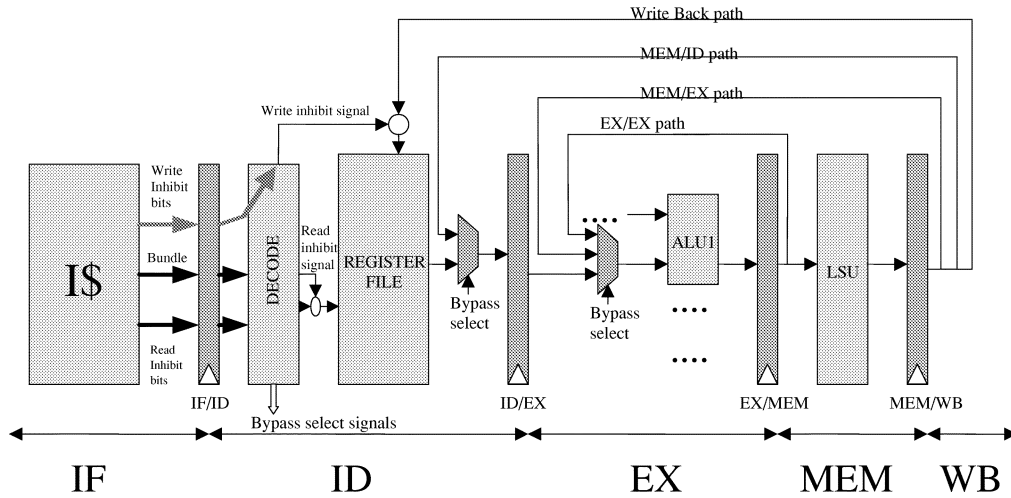


Fig. 2. The proposed low-power five-stage pipelined VLIW forwarding architecture.

A. Hardware Support

To analyze the hardware support needed by the proposed low-power optimization technique, let us introduce a generic four-way VLIW processor architecture with a five-stage pipeline provided with forwarding logic (see Fig. 2). The pipeline stages are:

- IF instruction fetch from I-cache;
- ID instruction decode and operands read from RF;
- EX instruction execution in one-cycle latency arithmetic logic units (ALUs);
- MEM load/store memory accesses;
- WB writeback of operands in the RF.

Three forwarding paths (EX-EX, MEM-EX, and MEM-ID) provide direct connections between pairs of stages through the EX/MEM and MEM/WB interstage registers. Given the forwarding network of Fig. 2 and considering a sequence $W = w_1, \dots, w_k, \dots, w_n$ of VLIWs, a generic instruction w_k can read its operands from the following instructions:

- w_{k-1} through the EX/EX forwarding path (used when w_k is in the EX stage);
- w_{k-2} through the MEM/EX forwarding path (used when w_k is in the EX stage);
- w_{k-3} through the MEM/ID forwarding path (used when w_k is in the ID stage);
- w_{k-n} where $n > 3$ through the RF.

The proposed power optimization requires a dedicated logic in the ID stage to decide whether or not the source/destination registers must be accessed in the RF. The instruction format must encode dedicated Read/Write Inhibit bits to enable RF accesses. The Write Inhibit bit of the instruction format is decoded to deassert the Write Enable signal (see Fig. 1) in the RF write port. The Read Inhibit bit is used to maintain unchanged the values on the input read addresses of the RF. This action reduces the switching activity of the read ports in the RF. Globally, the hardware overhead is equal to one-gate delay added on the processor writeback cycle.

To encode the Read (Write) Inhibit bit, we propose three different approaches.

- 1) *To add specific operation bits in the encoding of the long instruction format.* This solution has to be adopted at the

time of instruction set architecture (ISA) definition, at the expense of an increase of the instruction encoding length. For each operation in the bundle, we add 1 bit for each source (destination) register to indicate whether it is a read (write) inhibited register or not. For example, a bundle composed of four 32-bit ternary operations (one destination and two sources) would require 12 additional bits. In this case, the dimensions of the instruction cache, instruction buffers, etc., must be increased to take into account these additional bits, and this could lead to increased power consumption to be accounted for during the evaluation of the power savings. For such bundles, we can roughly estimate that the additional cache energy consumption can be limited within the 10% (12 bits over 128 bits).

- 2) *To exploit unused instruction encoding bits.* This solution is suitable when the ISA has been already defined; it saves instruction length, but it restricts the application of the proposed approach only to the subset of ISA operations with unused bits in the instruction format. In general, unused operation bits are very few. If we assume one unused bit per operation, we can consider using it to avoid only useless writes. Note that, in this case, there is no power overhead at all.
- 3) *To implement directly in hardware the control logic of the Read (Write) Inhibit bit.* In this case, the hardware provides a more limited window for the application of the basic idea. In particular, the hardware must check Write-After-Write dependencies, because the last read on a register, if it exists, is recognizable by the hardware only by monitoring two consecutive writes to the same register. On an in-order five-stage pipeline such as that depicted in Fig. 2, the window is between the decode stage and the writeback stage, thus individuating up to three cycles to check Write-After-Write dependencies. The hardware solution implies a power overhead due to the introduced control logic, to be accounted for during the evaluation of the power savings.

To quantify the register liveness that can be exploited by the proposed approach, Section IV-B describes the adopted

methodology. The methodology allows us to evaluate the maximum power savings within a basic block. This is useful to determine the exact power savings for the first two implementation approaches cited above. For the hardware approach, this analysis must be seen as an upper bound because the hardware has a more limited visibility than the basic block.

B. Register Liveness Analysis

In a VLIW architecture, all scheduling decisions concerning data, resource, and control dependencies are solved at compile time during static instruction scheduling [13]. Thus the decision on whether the destination (source) register must be write (read) inhibited or not can be determined by the compiler, limiting the hardware overhead.

To set specific write/read inhibit bits at compile time, the compiler must compute the liveness length of each assignment to a register and compare it with the limit implied by the architecture to exploit the forwarding network.

For this purpose, let us define the *liveness length* L of the n th assignment to a register R as the distance (measured as the number of instructions) between the n th assignment and its last use

$$L_n(R) = U_n(R) - D_n(R) \quad (2)$$

where $D_n(R)$ is the instruction fetch clock cycle of the instruction that performed the n th assignment to R ; and $U_n(R)$ is the instruction fetch clock cycle of the last instruction that used the n th assignment to R before the redefinition of R during the $(n+1)$ th assignment $D_{n+1}(R)$.

To simplify the analysis that the compiler has to perform, we assume a valid $L_n(R)$ when the following conditions hold:

- U_n and D_n are in the same basic block;
- D_{n+1} and D_n are in the same basic block.

These rules force us to consider only liveness ranges that do not cross basic block boundaries. However, this assumption does not represent a major concern, since most modern VLIW compilers tend to maximize the size of basic blocks, thus generating a relevant number of intrabasic block liveness ranges.

To clarify the concept with an example, let us analyze a portion of a four-way VLIW assembly trace executing a discrete cosine transform (DCT) (see Fig. 3). The analyzed code is composed of four long instructions (namely, n , $n+1$, $n+2$, and $n+3$) where each long instruction is identified by the instruction fetch clock cycle, a set of one to four operations that is terminated by a semicolon. In this example, we can observe a basic block terminating boundary at instruction $n+3$ (the conditional branch operation).

Let us consider the liveness of the assignment of $\$r18$ in $n(D_n)$. This definition is used for the last time in $n+1$, since there is another definition of $\$r18$ in the same cycle (i.e., D_{n+1}). L_n of $\$r18$ is thus equal to one clock cycle. Note that we cannot compute L_{n+1} of $\$r18$ because there are neither last uses U_{n+1} or redefinitions D_{n+2} in the same basic block. Considering the architecture of Fig. 2: one can see that the architectural maximum liveness length to be exploited is three. Thus, the write inhibit bit of the assignment to $\$r18$ performed in clock cycle n can be set to true by the compiler, while the

```

n   shr $r16 = $r16, 8           # shift right $r16 by 8
    sub $r18 = $r18, $r7        # $r18 <- $r18 - $r7
    add $r17 = $r17, $r19       # $r17 <- $r17 + $r19
    sub $r19 = $r19, $r15 ;

n+1 shr $r18 = $r18, 8
     shr $r17 = $r17, 8
     shr $r19 = $r19, 8
     mul $r20 = $r20, 181 ;    # $r20 <- $r20 * 181

n+2 sub $r10 = $r10, $r8
     mul $r11 = $r11, 3784
     sub $r5 = $r12, $r9 ;

n+3 sub $r10 = $r10, $r3
     add $r20 = $r20, 128
     brf $r26, label_232 ;    # branch to label_232 if
                               # $r26 is false

```

Fig. 3. Example of a segment of a four-way VLIW assembly execution trace of a DCT algorithm.

assignment to $\$r18$ performed in clock cycle $n+1$ must have the write inhibit bit set to *false*.

Note that for processors with more than five pipeline stages, we can have more than three forwarding paths, so we can extend the application of the proposed low-power optimization to variables whose liveness length is higher than three.

Note that our analysis is carried out within basic blocks of the compiled code and that the compiler used was not aware of the power optimization introduced. However, it becomes evident that the early application of the proposed idea during the scheduling and register allocation phases of the compilation would provide larger power savings. This could be done by modifying the scheduler and register allocator cost function in order to prefer schedules that present shorter register liveness.

V. EXCEPTION HANDLING AND CACHE MISSES

In this section, we analyze the problem of exception and cache miss handling in the proposed low-power VLIW forwarding architecture. For our analysis, we assume that the state of the processor can be decomposed in:

- 1) a *permanent* architectural state stored in the RF;
- 2) a *nonpermanent* architectural state stored in the registers between the pipeline stages from which the forwarding network retrieves source operands.

The nonpermanent architectural state is handled as a first-in, first-out memory, whose depth is equal to the number of stages during which the result of an operation can be stored in the pipeline (in the proposed five-stage pipeline architecture, the depth is equal to three).

In general, a pipelined processor assures that, when an element exits the nonpermanent state, it is automatically written back in the RF. On the contrary, in our low-power optimization, when an element exits the nonpermanent state and is no longer used, it can be discarded, avoiding the writeback in the RF. This behavior can create some problems when an exception occurs. In this section, we analyze how exceptions can be managed to maintain data consistency.

In our architecture, an exception can occur during the ID, EX, or MEM stages, and it can be serviced in the WB stage.

Cycle	Pipeline Stage		
	EX	MEM	WB
x	w_k	w_{k-1} (Exec. Signaled)	w_{k-2}
$x+1$	w_k	w_{k-1} (Exec. Served)
$x+2$	Deleted	Deleted	Deleted
....		
....	Exec. Handler	Exec. Handler	Exec. Handler
....		
$x+nn$	w_k	Exec. Handler	Exec. Handler

Fig. 4. An example of exact mode exception handling.

According to the exception taxonomy in [6], user-recoverable exceptions can be either precise or imprecise. We assume that the processor adopts the precise mode exception handling mechanism. Under this assumption, precise exceptions can be in turn either *exact* or *inexact*. An *exact* exception, caused by an instruction w , forces the visibility of the architectural state changes to all instructions issued after w . Furthermore, all state changes of instructions issued before w are visible to the exception handler. When an *inexact* exception occurs, the instructions in the pipeline are executed until completion, and the exception is served afterward. In this case, instructions issued immediately after the excepting instruction do not see architectural state changes due to the exception handler.

Let us analyze the behavior of the proposed solution in the two cases of exact and inexact exception handling.

Assume that exceptions are handled in *exact* mode. When the excepting instruction reaches the WB stage, instructions in the pipeline are flushed and reexecuted. Let us consider the example shown in Fig. 4, where at cycle x an instruction w_k reads its values from a write-inhibited w_{k-2} instruction through the forwarding network. Meanwhile, let us assume that instruction w_{k-1} generates an exception during the MEM stage. The results of w_{k-2} would have been lost, but we need these values to be used during the reexecution of w_k . Since neither the forwarding network nor the RF contains the results of w_{k-2} , the architectural state seen during the reexecution of w_k (at cycle $x + nn$) will be incorrect.

To guarantee that instructions in the pipeline are reexecuted in the correct processor state, write-inhibited values must be written in the RF *anytime an exception signal is generated in the ID, EX, or MEM stages* and read-inhibited operation must read operands from the register file after the exception. Thus, the Read/Write Inhibit bits can safely be ignored by the microarchitecture in the case of an exception.

In the previous example (where w_{k-1} generates an exception in the MEM stage), the proposed solution forces the writeback of the results of w_{k-1} and w_{k-2} in the RF; similarly, we need to force the read of the operands from the RF for the successive read-inhibited instructions. Therefore, during the reexecution of w_k at cycle $x + nn$, the operands are read from the RF.

Let us now assume that exceptions are handled in *inexact* mode. In this case, to guarantee a semantically correct execution, all instructions in the pipeline must be forced to write back the results in the RF.

The proposed low-power architecture shown in Fig. 2 supports both exception handling mechanisms.

- 1) When the exceptions are *exactly* handled, the supported register liveness is less than or equal to two clock cycles (through the EX/EX and the MEM/EX paths).
- 2) When exceptions are *inexactly* handled, the exploiting register liveness can be extended to three clock cycles (through the EX/EX, MEM/EX, and MEM/ID paths).

Let us briefly consider the case of interrupts and cache misses. Due to the asynchronous nature of interrupts, they can be treated as inexact exceptions by forcing each long instruction in the pipeline to write back the results before interrupt handling. Instruction cache misses produce bubbles/stalls flowing through the pipeline. Therefore, whenever a miss signal is raised by the cache control logic, we force the writeback of the results of the instructions in the pipeline. We assume that data cache misses stall the pipeline until the needed block is ready. In this situation, the normal write-inhibition mechanism can be tolerated since the relative timing of the instructions is not modified.

VI. AN APPLICATION EXAMPLE

To provide experimental evidence to our approach, we considered the *Lx* family of embedded VLIW cores, jointly developed by Hewlett-Packard Laboratories and STMicroelectronics [11]. *Lx* is a family of customizable, multiclustered VLIW architectures, where each *Lx* cluster is a multi-issue six-stage pipelined VLIW core composed of four 32-bit integer ALUs, two 16×32 multipliers, and one load/store unit, with in-order execution and exact exception handling. The used version of the *Lx* is a four-issue architecture with six pipeline stages: instruction fetch (IF), instruction decode (ID), register read (RR), execution 1 (EX1), execution 2 (EX2), and writeback (WB). The forwarding paths are EX1-EX1 and EX2-EX1. The EX2-RR path is the normal writeback path giving a total of three forwarding paths that can be exploited for our low-power optimization. The *Lx* ISA is a RISC integer instruction set that supports speculative execution and prefetching. The RF provides 64 32-bit general-purpose registers and eight 1-bit branch registers. A commercial software toolchain supports the development of embedded software for the *Lx* architecture and provides a sophisticated compiler, based on the trace scheduling [14]: an instruction scheduling method consisting of high-level optimizations and aggressive code motions best exploiting the ILP.

A. Results of Register Liveness Analysis

To evaluate the impact of the proposed low-power optimization on the *Lx* architecture, we set up an experimental environment to analyze the liveness length of registers in a set of multimedia benchmarks and embedded real-world DSP algorithms written in *C* for which the *Lx*-architecture is targeted. Our main goal is to measure the dynamic percentage of register definitions in the application code that can be directly read from the forwarding network, without being written in the RF.

Although the register liveness analysis could be performed statically by the compiler, we decided to perform a dynamic analysis, consisting of the inspection of an execution trace of

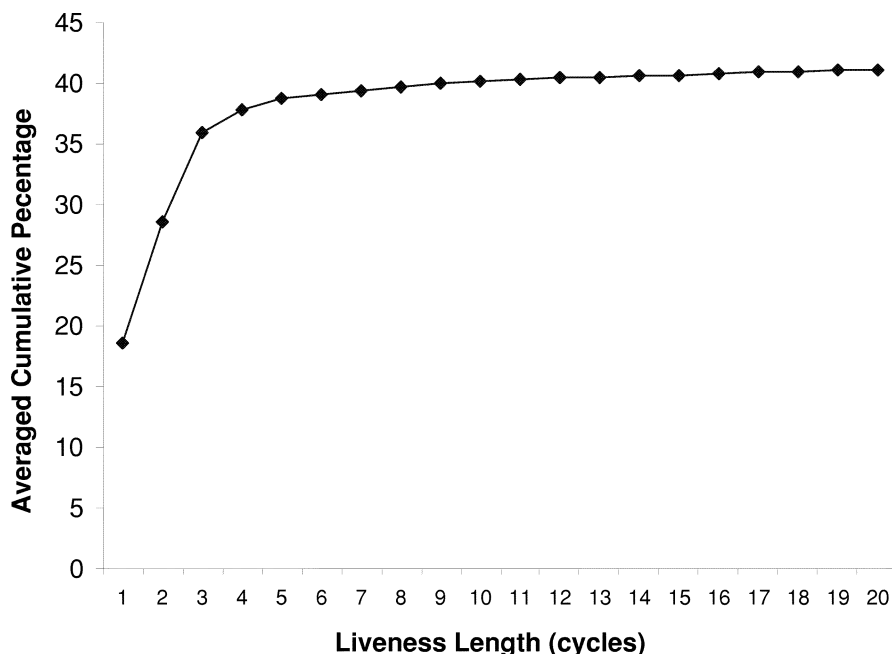


Fig. 5. Average cumulative distribution of register liveness.

the program, because it provides us accurate run-time profiling information on how many register accesses can be inhibited.

It is clear that the application of the proposed idea during the early phases of scheduling and register allocation of the compilation flow would provide greater power savings. This could be achieved by modifying the scheduler and register allocator algorithm in order to prefer schedules that present shorter register liveness.

Each benchmark program has been compiled with the Lx compiled simulator, which is part of the Lx toolchain.

In our analysis, the compiled simulator source has been instrumented by in-house automatic tools to trace, each time a very long instruction is executed, the following profile information: register definitions, register uses, basic block boundaries encountered, and cache misses occurred.

Once the simulator has been compiled and executed, the generated traces are used to perform the register liveness analysis. To perform such analysis, we selected a set of DSP algorithms (written in C) that represent a significant set of target applications for the embedded Lx -architecture. Most of these algorithms are part of the Mediabench suite [15], while the remaining part of algorithms are in-house optimized versions of classical transforms (see [16] and [17] for an overview).

The average behavior of the register liveness for the interval [1–20] is depicted in Fig. 5, where each point represents the percentage of definitions of a given liveness averaged on all the benchmarks. Note that for liveness lengths greater than four, the cumulative percentage does not increase significantly.

The behavior of the register liveness for the analyzed algorithms is reported in Fig. 6, where, for a fixed benchmark, each bar represents the percentage of register definitions whose liveness is comprised within a fixed interval. For example, the black bar represents the percentage of definitions that have a liveness length between one and four clock cycles.

Even with our simplifying assumptions, we can observe that, on average, 28.6% of register definitions have $L_n \in [1-2]$, 35.8% of register definitions have $L_n \in [1-3]$, and moving to a [1–4] interval does not change significantly from the case $L_n \in [1-3]$. The peak percentage reaches 71.8% in the case of a [1–3] cycle interval, which represents the interval of interest for our architecture.

In our analysis, we do not take into account the case where a register is never read between two successive definitions. In fact, register overwriting can occur, for example, across basic blocks or during context switches. Analysis of this type of register overwriting is out of the scope of our approach.

B. Results of Power Analysis

To derive the power savings implied by our optimization, we simulated at the transistor level the eight read-ports and four write-ports Lx RF characterized by 64 registers. We then modeled the RF with an extension of the linear model proposed in [4] that takes into account also a base power cost independent from the number of accesses that are performed during a clock cycle

$$P_{RF} = \text{BaseCost} + n_w P_{1w} + n_r P_{1r} \quad (3)$$

where P_{RF} is the average RF power consumption during a clock cycle, BaseCost is a constant power value due to the overall switching activity due to the clock signal, n_w is the average number of simultaneous write accesses to the RF (from zero to four), P_{1w} is the average power cost of one write access, n_r is the average number of simultaneous read accesses to the RF (from zero to eight), and P_{1r} is the average power cost of one read access.

From simulations of the given RF implementation during different and simultaneous RF accesses, we derived the averaged

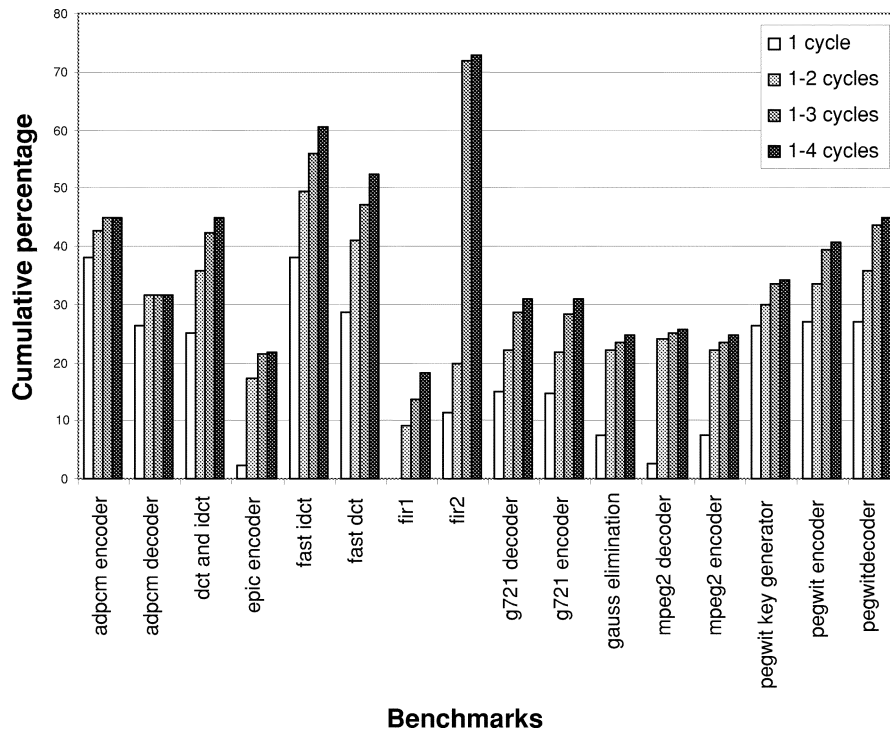


Fig. 6. Cumulative distribution of register liveness for the selected benchmark set.

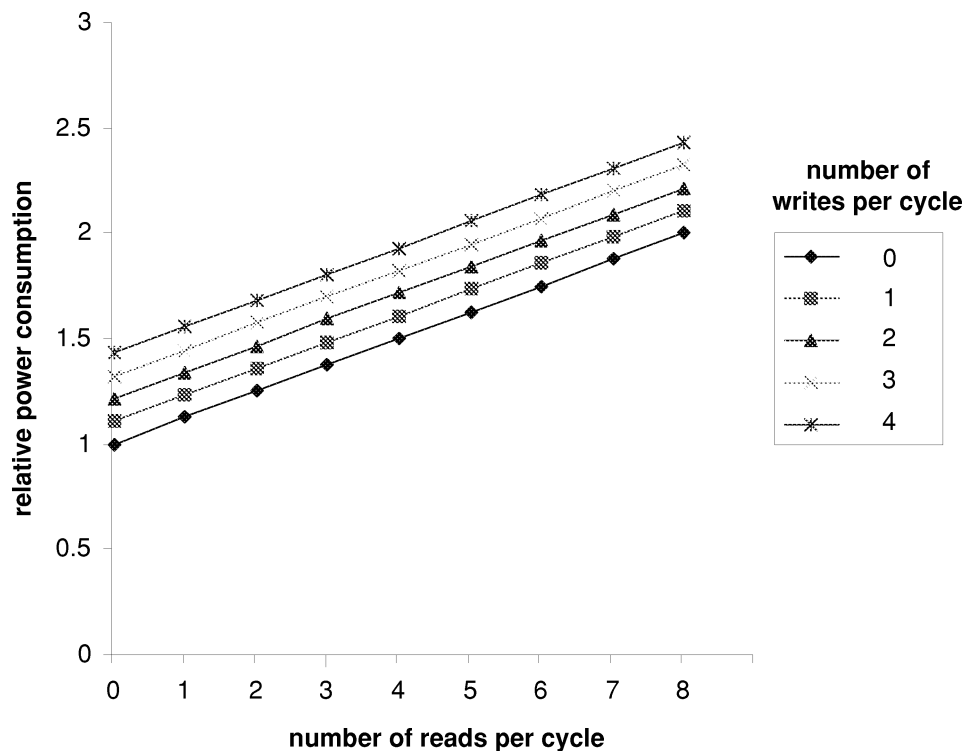


Fig. 7. Average power consumption of RF read/write accesses (values are normalized to BaseCost).

and normalized power results shown in Fig. 7. We can note that, when the RF is heavily stressed (eight reads and four writes per cycle), the power consumption is 2.5 times the base power cost.

The power analysis of the proposed forwarding optimization has been derived by combining the power figures of our RF model and the savings in the average number of RF accesses per cycle, obtained by profiling each benchmark program through the instrumented *Lx* compiled simulator.

Fig. 8 reports the power saving by applying the proposed low-power solutions to registers whose liveness length is less than or equal to two (3) clock cycles. In the case of two clock cycles, the average power saving of the RF power for the given set of benchmarks is 7.4%, while the saving can reach up to 25%. In the case of three clock cycles, the average power saving for the given set of benchmark is 7.8%, while the saving can reach up to 26.1%.

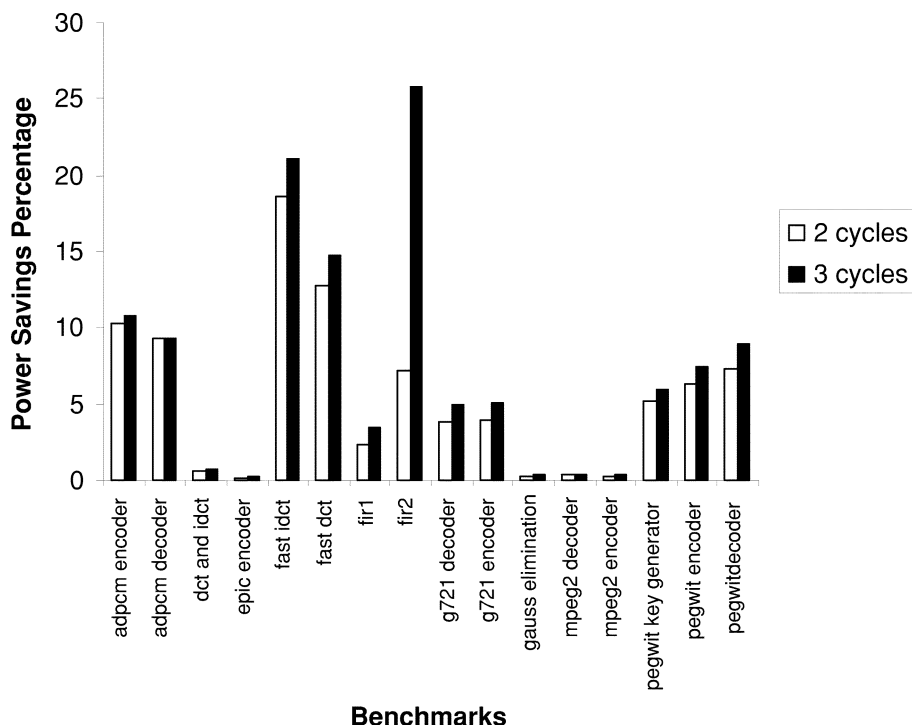


Fig. 8. Register file power savings percentage of our approach applied to registers liveness ≤ 2 (and ≤ 3) clock cycles for the selected benchmarks. The average value for two (three) clock cycle is 7.4% (7.8%).

As expected, higher power savings have been obtained by simulating algorithms whose kernel is composed of instructions that tend to reuse, as much as possible, available registers in the short term. This leads to very short register liveness that can be exploited by our approach. In this scenario, the proposed idea can be further exploited by changing the register-picking algorithm in the register allocator as well as by modifying the scheduler.

The algorithms whose power savings are very low are characterized by a large amount of instruction cache misses and control instructions (such as branches) that tend to limit the application of our optimization, even if they present a significant amount of short definitions with respect to total definitions. This justifies the fact that the power savings of algorithms with similar register definition behavior can be very different (e.g., adpcm encoder and dct & idct).

Globally, the obtained results suggest us that the highest power savings can be achieved for embedded systems characterized by a high register reuse on the short term and a limited number of instruction cache misses and control instructions, instead of general-purpose systems.

Concerning the overall power budget of the core plus the RF, we estimated that, for the given architecture, the percentage of the power consumption due to the RF is 19.9% [3]. Thus, for an RF power saving of 26.1%, the corresponding overall power saving is around 5%. However, in the case in which we add some bits to encode the Read/Write Inhibit bit, we could consider a power overhead that could compromise the obtained power savings.

However, it should be noted that the target core is not optimized for power. Thus, this modest value is expected to be increased if we would have considered a low-power processor architecture where the percentage of the core power consumption is much more reduced.

Even considering the target architecture, the limited power saving must be considered in conjunction with other techniques for low power that can be orthogonally applied to the core and the RF, globally providing a more considerable power savings.

VII. CONCLUSION

In this paper, an architectural solution to reduce the power consumption in VLIW pipelined embedded processors has been proposed. The proposed technique exploits the forwarding network to save transition activity in accessing the RF. This paper discusses how the low-power solution can be implemented at compile time. An industrial application example demonstrates the effectiveness of the proposed technique from the power standpoint by reaching a maximum power saving on the RF of 26.1% at a cost of relatively few architectural modifications to the processor forwarding network. We are currently working on a larger set of benchmark programs to provide a more extended experimental evidence of the advantages of our approach. Moreover, our ongoing work aims at extending the proposed approach by introducing a new level on the memory hierarchy consisting of the pipeline microregisters level. This extension would virtually increase RF space and, thus, reduce register spilling code and their related power cost.

REFERENCES

- [1] A. Chandrakasan and R. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, no. 4, pp. 498–523, 1995.
- [2] K. Roy and S. C. Prasad, *Low-Power CMOS VLSI Circuit Design*. New York: Wiley-Interscience, 2000.
- [3] L. Benini, D. Bruni, M. Chinosi, C. Silvano, V. Zaccaria, and R. Zafalon, "A power modeling and estimation framework for vliw-based embedded systems," in *Proc. Int. Workshop Power and Timing Modeling, Optimization and Simulation (PATMOS'01)*, Sept. 26–28, 2001.

- [4] V. Zyuban and P. Kogge, "The energy complexity of *RF*," in *Proc. Int. Symp. Low-Power Electronic Design (ISLPED98)*, Monterey, CA, 1998, pp. 305–310.
- [5] J. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd ed. San Mateo, CA: Morgan Kaufmann, 1996.
- [6] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*. London, U.K.: Wiley, 1997.
- [7] A. Abnous and N. Bagherzadeh, "Pipelining and bypassing in a VLIW processor," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, pp. 658–663, June 1994.
- [8] R. Yung and N. C. Wilhelm, "Caching processor general registers," in *Proc. IEEE Int. Conf. Computer Design (ICCD'95)*, 1995, pp. 307–312.
- [9] L. A. Lozano and G. R. Gao, "Exploiting short-lived variables in superscalar processors," in *Proc. 28th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO-28)*, 1995, pp. 292–302.
- [10] M. M. Martin, A. Roth, and C. N. Fischer, "Exploiting dead value information," in *Proc. 30th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO-30)*, 1997, pp. 125–135.
- [11] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. Homehood, "Lx: A technology platform for customizable VLIW embedded processing," in *Proc. Int. Symp. Computer Architecture (ISCA'00)*, Vancouver, BC, Canada, 2000, pp. 203–213.
- [12] M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon, "Exploiting data forwarding to reduce the power budget of VLIW embedded processors," in *Design, Automation Test Europe 2001: Conf. Exhibition 2001 Proc.*, March 2001, pp. 252–257.
- [13] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Reading, MA: Addison-Wesley, 1986.
- [14] J. Fisher, "Trace scheduling: A technique for global microcode compaction," *IEEE Trans. Comput.*, vol. C-30, pp. 478–490, July 1981.
- [15] J. K. Kin, M. Gupta, and W. H. Mangione-Smith, "Filtering memory references to increase energy efficiency," *IEEE Trans. Comput.*, vol. 49, Jan. 2000.
- [16] W. H. Chen, C. H. Smith, and S. C. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Trans. Commun.*, vol. COM-25, pp. 1004–1009, Sept. 1977.
- [17] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge, U.K.: Cambridge Univ. Press, 1993.



Cristina Silvano (M'93) received the Dr. Eng. degree in electrical engineering from Politecnico di Milano, Italy, in 1987 and the Ph.D. degree in information engineering from the Università degli Studi di Brescia, Italy, in 1999.

From 1987 to 1996, she was a Senior Design Engineer with R&D Labs, Bull HN Information Systems, Pregnana M., Italy. From 1996 to 1998, she was with the Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia. From 1998 to 1999, she was a Consultant Researcher in the Electronic Design Automation Area, Center for the Research and the Education in Information Engineering, Milan. Currently, she is an Assistant Professor at the Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano. Her primary research interests are in the areas of complex computer architectures based on microprocessors and design automation methodologies of VLSI circuits and systems, with particular emphasis on low-power design techniques, high-level power estimation methodologies, and hardware/software codesign methodologies for embedded systems.



Vittorio Zaccaria (S'98) received the Dr. Eng. and Ph.D. degrees in computer engineering from the Politecnico di Milano, Italy, in 1998 and 2002, respectively.

Since December 2001, he has been an Assistant Professor at the Dipartimento di Elettronica e Informazione, Facoltà di Ingegneria, Politecnico di Milano. His primary research interests include algorithms, methodologies, and tools for the design of low-power digital systems with particular emphasis on microprocessor-based embedded architectures.

Mariagiovanna Sami (M'69–SM'99) received the Dr. Ing. degree in electronic engineering from Politecnico di Milano, Italy, in 1966 and the Libera Docenza (computing and switching theory) degree in 1971.

Since then, she has been with the Department of Electronics and Information, Politecnico di Milano, where she has held various research and teaching positions and has been a full Professor since 1980. Her research interests are in the areas of fault tolerance, highly parallel VLSI and WSI architectures, and high-level synthesis. She has published more than 150 papers in international journals and conference proceedings. She is coauthor of *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays* (Cambridge, MA: MIT Press). She was Editor-in-Chief of the *EUROMICRO Journal*.

Prof. Sami is a member of EUROMICRO. She was on the Board of Editors of the IEEE TRANSACTIONS ON COMPUTERS.



Donatella Sciuto (S'84–M'87) received the Laurea degree in electronic engineering in 1984 and the Ph.D. degree in electrical and computer engineering from the University of Colorado, Boulder, in 1988.

She was an Assistant Professor at the Dipartimento di Elettronica per l'Automazione, Università di Brescia, Italy, until 1992. She is currently a full Professor at the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy. She is member of various program committees of EDA conferences, including DAC, ICCAD, DATE,

CODES/CASHE, DFT, FDL, member of the executive committee of ICCAD and DATE. She is an Associate Editor of *Design Automation of Embedded Systems* and the *Journal of System Architecture*. Her research interests cover mainly methodologies for codesign of embedded systems, including system verification, design for low power consumption, HW/SW partitioning, and test generation techniques.

Prof. Sciuto is a member of IFIP 10.5 and EDAA. She is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS. She was Guest Editor of a special issue of *IEEE Design & Test Magazine* in 2000.



Roberto Zafalon received the Laurea degree in electrical engineering from the University of Padova, Padova, Italy, in 1987.

He is Manager of the Competence Centre for Low Power System Design, Advanced System Technology group, STMicroelectronics, Agrate Brianza, Milano, Italy. Since 1999, he has been in charge of the Advanced Power and Timing team, Central R&D Design Platform group. After two years with Necsy (ITALTEL group, Padova), he joined ST in 1989 with the Central R&D Design Automation group. Since then, he has taken various responsibilities as Project Leader and Team Manager on corporate CAD projects for innovative ASIC design solutions and methodologies in the field of timing verification and power-conscious design, high-level power modeling, optimization, and signal-integrity analysis, in order to achieve the full industrial exploitation of deep submicrometer technologies for complex system-on-chip designs. He participated in a number of European Community (IST and MEDEA) and national research projects and cooperates with several international academies and research institutes. He has contributed to many international scientific publications and has received three European and two U.S. patents in the field of low-power design. He is currently serving on the Technical Program Committee of a number of international conferences, including the Design Automation Conference, Design Automation and Test in Europe, International Symposium on Low Power Electronics and Design, and Power and Timing Modeling Optimization and Simulation.

Dr. Zafalon received the National Award from the Italian Electrical and Electronic Association for the best undergraduate in electronics in 1981. He held full tutorials on research topics related to low-power tools and methodologies at three IEEE international conferences.