Exploiting Data Forwarding to Reduce the Power Budget of VLIW Embedded Processors

M. Sami[§] D. Sciuto[§] C. Silvano[§] V. Zaccaria[§] R. Zafalon[†]

[§]Politecnico di Milano
Dip. di Elettronica e Informazione
Milano, ITALY 20133
{sami,sciuto,silvano,zaccaria}@elet.polimi.it

[†]STMicroelectronics
Advanced System Technology
Agrate Brianza (Milan), ITALY 20041
roberto.zafalon@st.com

Abstract

In this paper, a low-power approach to the design of embedded VLIW processor architectures is proposed. To solve the most part of data hazards in the pipeline, processors use forwarding (or bypassing) hardware to provide the required operands from the inter-stage pipeline registers directly to the inputs of the function units. The operands are then stored in the Register File during the write-back pipeline stage. In this paper, we propose a power optimization technique based on the exploitation of the forwarding paths in the processor to avoid the power cost of writing/reading short-lived variables to/from the Register File. In application-specific embedded systems, experimental evidence has shown that 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 be accessed directly through the forwarding registers, avoiding write-back. An application example of our solution to a VLIW embedded core, when accessing the Register File, has shown a power saving up to 35% with respect to the unoptimized approach on the given set of target benchmarks. The performance overhead is equal to one-gate delay to be added on the processor critical-path.

Keywords: Low-Power, Pipeline Processors, VLIW Embedded Architectures, Forwarding.

1 Introduction

The embedded processor market is growing rapidly and the complexity of embedded applications is growing even faster. Recently, VLIW architectures have been proposed as an interesting alternative to more conventional CPUs, to balance performance with hardware complexity and scalability. The performance/complexity tradeoff is made possible by a sophisticated ILP (Instruction Level Parallelism) compiler infrastructure that aims at scheduling high performance parallel code at compile time rather than at run time. As in all modern CPUs, including VLIW architectures, pipelining is extensively adopted. Although granting good CPI, pipelining introduces data hazards.

In most cases, the problem of data hazards can be solved by the introduction of the hardware technique called *forwarding* (also called *bypassing* and sometimes *short-circuiting*) [1]. The forwarding technique uses the inter-stage pipeline registers to pass the results of previous instructions directly back to the function units that require them. A result can be forwarded from the output of a function unit to the inputs of the following one, as well as looping from the output to the inputs of the same unit.

To implement the bypassing mechanism, the necessary forwarding paths and the related control must be included in the processor design. In general, this technique requires a forwarding path from any pipeline register to the inputs of any function unit, such as in the *DLX* architecture [1]. Data bypassed to function units in early pipeline stages are then stored in the Register File (RF) during the last pipeline stage (i.e. write-back stage) for further use. Processors using the forwarding technique give higher performance due to the reduction of stall cycles introduced by data hazards.

Main issues involved in the forwarding mechanism for VLIW processors have been investigated in literature [2], [3], [4]. The authors of [2] analyze the performance advantages of various bypassing schemes, in terms of their effectiveness in solving data hazards in both four-stage and five-stage pipelines. 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 [4] to improve performance. In this architecture, called 'Register Scoreboard and Cache', pipeline operands are supplied by either the register cache or by the bypass network.

The concept of avoiding dead value information in the RF has been explored in [5]. Register values are considered dead, when they will not be read before being overwritten. The advantages of this approach have been evaluated in terms of RF size reduction and elimination of unnecessary save/restore instructions from the execution stream at procedure calls and across context switches.

Low-power dissipation is an increasingly relevant requirement for embedded processors [6], [7]. Low-power design techniques are widely adopted during microprocessor design to meet the stringent power constraints, while preventing from any performance degradation. For high-performance processors, low-power solutions target the reduction of the effective switched capacitance C_{EFF} of the processor nets. The C_{EFF} parameter of a net is defined as the product of the load capacitance C_L and the switching activity α of the node. In digital CMOS processors, significant power savings can be achieved by the minimization of the transition activity of high-capacitance buses, such as data-path buses and I/O buses. Another substantial portion of the power budget in modern processors is due to multi-port RF accesses and other on-chip cache accesses [8].

The main goal of the present paper is to define an optimization technique that exploits data forwarding for short-lived variables to save power in VLIW pipelined architectures. Our basic idea consists of reducing the RF activity by avoiding long-term power-expensive writing of

short-lived variables by exploiting inter-stage registers and forwarding paths. Short-lived variables are simply stored by the producer instruction in the inter-stage registers and fed to the consumer instruction by exploiting the forwarding paths. No write-back to the RF is performed by the producer instruction, and no read from the RF is affected by the consumer instruction. As an application example, let us consider the following fragment of code:

```
I_n :     $r2 ← ...;
I_n+1 :   ...;
I_n+2 :   ... ← $r2;
I_n+3 :   $r2 ← ...;
```

In this example, the write-back 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 \$r2 liveness is equal to two instructions.

The decision to enable the RF write-back 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 register is *short-lived*, and its value can be passed as an operand to next instructions by using the forwarding paths, thus avoiding to write it back to the RF.

The proposed architecture becomes particularly attractive in some classes of embedded applications, where the register liveness analysis (see Subsection 4.1) has shown that more than half of the total register definitions is limited to the next two instructions.

The most relevant features addressed in the present paper are:

- the proposal of a power-oriented architectural extension to the processor forwarding network to avoid the writing and successive reading to/from the RF of short-lived variables;
- the analysis of the impact of the proposed low-power architectural solution for VLIW processors on the ISA and the compiler;
- the analysis of the exception handling;
- the introduction of a linear RF power model;
- the set up of an experimental methodology to evaluate the power savings of the proposed low-power architectural extension compared to real-world applications running on an industrial VLIW embedded processor jointly developed by Hewlett Packard and STMicroelectronics[10].

The paper is organized as follows. The proposed low-power forwarding architecture for VLIW processors is presented in Section 2. Section 3 discusses the problem of exceptions handling. Section 4 shows the results of the analysis of the register liveness on a set of embedded DSP algorithms and the power savings achievable on an industrial VLIW processor. Concluding remarks and future directions of our work are finally reported in Section 5.

2 Low-Power Forwarding Architecture

Starting from the basic DLX architecture [1], we consider a generic 4-way VLIW processor architecture with 5-stage pipeline provided with forwarding logic. 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 ALUs.
- MEM: Load/Store memory accesses.
- WB: Write-Back of operands in the RF.

Three forwarding paths (EX-EX, MEM-EX and MEM-ID) provide direct connections between pairs of stages (see Figure 1) through the EX/MEM and MEM/WB inter-stage registers. Given the above forwarding network, let us consider a sequence $W = w_1 ... w_k ... w_n$ of very long instruction words. 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.

As stated in the introduction, the proposed approach inhibits the write and subsequent reads of operands in the RF whenever written values can be retrieved from the forwarding network due to their short liveness (see Subsection 4.1 for the definition of register liveness). We assume that an instruction w_d assigns a register R whose liveness is less than or equal to three and that w_k uses R during this live interval. For processors with more than five pipeline stages, we can have more than three forwarding paths, so we can extend the application of our low-power approach to variables whose liveness length is higher than three.

Our basic idea consists of reducing power consumption by:

- disabling the write of R in the WB stage of w_d;
- inhibiting w_k from asserting the read address of the RF to read R (retrieved from the bypass network).

The proposed low-power processor architecture supports both features. A simplified architecture can also be envisioned, which supports just the write (or read) inhibition to minimize the required hardware overhead.

In a VLIW architecture, all scheduling decisions concerning data, resource and control dependencies are solved at compile time during static instruction scheduling [9]. Thus the decision whether the destination (source) register must be write (read) inhibited or not, has to be demanded to the compiler, limiting the hardware overhead.

The proposed power optimization requires a dedicated logic in the ID stage to decide whether or not the source/destination registers must be read accessed in the RF. The instruction format must be extended to include dedicated Read/Write Inhibit bits to enable RF accesses.

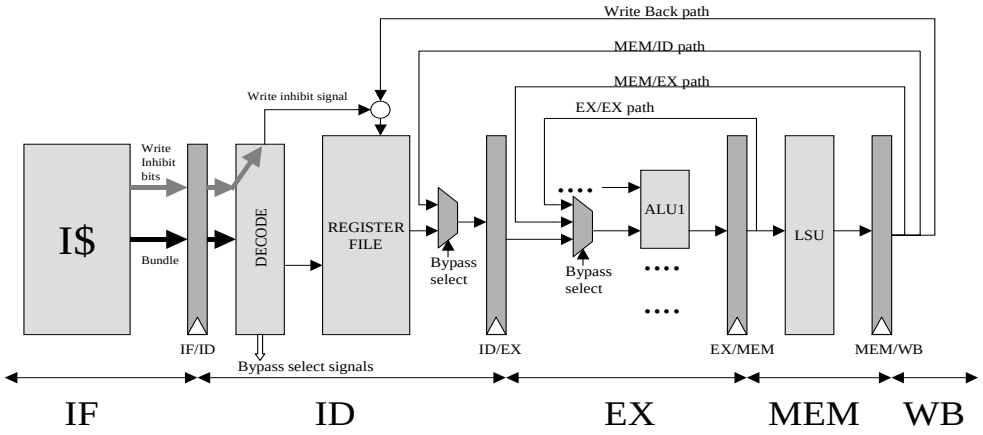

Figure 1: The proposed low-power 5-stage pipelined VLIW forwarding architecture.

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 critical path.

To encode the Read (Write) Inhibit bit, we propose two different approaches:

1. To add specific operation bits in the encoding of the long instruction format. This solution is suitable during 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 4 ternary operations (1 destination and 2 sources) would require 12 additional 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 suppose to use it to avoid only useless writes.

In both cases, while we minimize the RF switching activity, we slightly increase switching activity of the memories used to store instructions.

3 Exception Handling

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

1. A *permanent* architectural state stored in the RF.

2. A *volatile* architectural state stored in the registers between the pipeline stages from which the forwarding network retrieves source operands.

The volatile architectural state is handled as a FIFO 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 5-stage pipeline architecture the depth is equal to 3).

In general, a pipelined processor assures that, when an element exits the volatile state, it is automatically written-back in the RF. On the contrary, in our low-power optimization, when an element exits the volatile state and it is no longer used, it can be discarded avoiding the write back 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 can be serviced in the WB stage.

According to the exception taxonomy in [3], we assume that the processor adopts the precise mode exception handling mechanism. Under this assumption, exceptions can be *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, without seeing the effects of the exception, that is served after. 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 exceptions are handled in *exact* mode. When the excepting instruction reaches the WB stage, instructions in the pipeline are flushed and re-executed. Let us consider the example shown in Table 1, 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 re-execution of w_k. Since neither the forwarding network nor the RF contain the results of w_{k-2}, the architectural state seen during the re-execution of w_k (at cycle $x + nn$) will be incorrect.

To guarantee that instructions in the pipeline are re-executed 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.

In the previous example (where w_{k-1} generates an exception in the MEM stage), the proposed solution forces the write-back of the results of $w_{k-1}$and w_{k-2} in the RF, therefore during the re-execution 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 Figure 1 supports both exception handling mechanisms:

- When the exceptions are *exactly* handled, the supported register liveness is less than or equal to 2 clock cycles (through the EX/EX and the MEM/EX paths);
- When exceptions are *inexactly* handled, the exploiting register liveness can be extended to 3 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 flowing through the pipeline, therefore whenever a miss signal is raised by the cache control logic, we force the write back of the results of the instructions in the pipeline.

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

Table 1: An example of exception handling.

4 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 STMicroelectonics [10]. Lx is a multi-clustered VLIW architecture, where each Lx cluster is a 4-issue 6-stage pipelined VLIW core composed of four 32-bit integer ALUs, two 16x32 multipliers and one load/store unit. The six pipeline stages are: Instruction Fetch (IF), Instruction Decode (ID), Register Read (RR), Execution 1 ($EX1$), Execution 2 ($EX2$), Write-Back (WB). The forwarding paths are EX1-EX1 and EX2-EX1. The EX2-RR path is the normal write-back path. Globally three paths can be exploited as in the DLX architecture. 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 8 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 [11]: an instruction scheduling method consisting of high-level optimizations and aggressive code motions best exploiting the ILP.

4.1 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 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 on the inspection of an execution trace of the program, because it provides us accurate run-time profiling information on how many register accesses can be inhibited.

Each benchmark program has been compiled with the Lx Compiled Simulator, that is part of the Lx toolchain. The Lx compiled simulator accepts a generic C program and elaborates it in two steps:

1. Translation of the program C source into its Lx assembler equivalent;
2. Conversion of the generated Lx assembler program into a new C source that simulates the program on the Lx architecture (*Compiled Simulator source*). The Compiled Simulator source simulates the Lx architecture by updating a simulated state of the machine each time an assembly statement is executed. Besides it also collects profiling information to supply a hint on the performance of the program.

In our analysis, the Compiled Simulator source has been instrumented by in house automatic tools (written in *perl*), to trace, each time a very long instruction is executed, the following figures: (*i*) register definitions; (*ii*) register uses; (*iii*) basic block boundaries encountered.

Once the simulator has been compiled and executed, the generated traces are used to perform the register liveness analysis. For this purpose, we defined 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:

$$L_n(R) = U_n(R) - D_n(R)$$

where $D_n(R)$ is the trace index of the instruction that performed the n-th assignment to R and $U_n(R)$ is the index of the last instruction that used the n-th assignment to R before the redefintion of R during the $(n+1)$-th assignment $D_{n+1}(R)$. To simplify our analysis, we have computed $L_n(R)$ with the following restrictions:

- 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 maximize the size of basic blocks, thus generating a relevant number of intra-basic block liveness ranges.

To clarify the concept with an example, let us analyze a portion of a 4-way VLIW assembly trace executing a DCT. The analyzed code is composed of four long instructions (namely 27268, 27269, 27270, and 27271):

```
27268   shr $r16 = $r16, 8
        sub $r18 = $r18, $r7
        add $r17 = $r17, $r19
        sub $r19 = $r19, $r15 ;

27269   shr $r18 = $r18, 8
        shr $r17 = $r17, 8
        shr $r19 = $r19, 8
        mul $r20 = $r20, 181 ;

27270   sub $r10 = $r10, $r8
        mul $r11 = $r11, 3784
        sub $r5 = $r12, $r9 ;

27271   sub $r10 = $r10, $r3
        add $r20 = $r20, 128
        brf $r26, label_232 ;
```

where each long instruction is identified by an index of execution, a set of one to four operations and are terminated by a semicolon. In this example, we can observe a basic block terminating boundary at instruction 27271 (the conditional branch operation). Let us consider the liveness of the assignment of $\$r18$ in 27268 (D_n). This definition is used for the last time in 27269, 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.

To perform the register liveness analysis, we selected a set of DSP algorithms (written in C) that represent a significant set of target applications for the embedded Lx-architecture:

- A FIR routine (32-tap, 16-bit);
- An unoptimized DCT and IDCT transform applied to a 64 half-word block and extracted from an MPEG2 software;
- An optimized version of the DCT derived from [12];
- An optimized version of the IDCT derived from [13];
- A wavelet transform derived from [13].

To improve performance, the optimized versions of the DCT/IDCT algorithms are characterized by a lower number of memory accesses and a higher register re-use with respect to the other benchmarks.

The distribution of register liveness for the analyzed algorithms are reported in Table 2 (the corresponding cumulative distibution is graphically summarized in Figure 2). For each benchmark, the columns of Table 2 represent the percentage of registers whose liveness is equal to a given length L - in the range from 1 to 8 clock cycles (instructions).

	Register Liveness Length							
Algorithm	1	2	3	4	5	6	7	8
FIR	0%	13%	10%	10%	0%	0%	0%	0%
DCT/IDCT	28%	12%	8%	3%	2%	1%	1%	0%
DCT (opt.)	32%	14%	11%	6%	2%	1%	0%	0%
IDCT (opt.)	42%	12%	6%	5%	2%	1%	1%	1%
Wavelet	7%	17%	1%	0%	2%	0%	0%	0%

Table 2: Percentage of registers whose liveness is equal to a given length L in the range from 1 to 8 clock cycles.

Even with our simplifying assumptions, we can observe that, for optimized algorithms, there is more than half of the total register definitions with liveness length within 2 clock cycles (46% and 54% for DCT and IDCT respectively). On

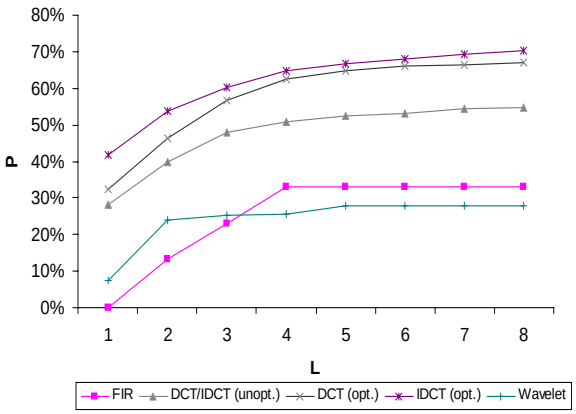

Figure 2: Cumulative distribution of registers liveness for the selected benchmark set.

average, in 35.4% of instances the distance from register definition to last use is less or equal to 2 clock cycles, while in 42.6% the distance is less or equal to 3 clock cycles.

In our analysis, we do not take into account the case in which a register is never read between two successive definitions. In fact, register overwriting can occur, for example, across basic blocks or during context switches, but it cannot be statically generated by an optimizing compiler within a basic block. Therefore, this type of register overwriting is out of the scope of our approach, since we are focusing on optimization applicable within a basic block during the VLIW static compilation phase.

4.2 Power Analysis

To provide evidence of the advantages of our low-power approach, we designed and simulated an in-house RF featuring 8 read-ports and 4 write-ports. The RF is functionally equivalent to the Lx RF, but reduced in terms of size.

The general problem of evaluating the power consumption of RFs has recently been afforded in [8]. The paper compares various RF design techniques in terms of energy consumption, as a function of architectural parameters such as the number of registers and the number of ports. In our work, we propose a simple parametric power model suitable for our RF architecture. In our model, the RF power behavior is linear with respect to the number of simultaneous read and write accesses:

$$P_{RF} = BaseCost + n_w P_{1w} + n_r P_{1r}$$

where $BaseCost$ is a constant power value due to the activity of the clock signal, n_w is the average number of simultaneous write accesses to the RF (from 1 to 4), 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 1 to 8), 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 and normalized power results shown in Figure 3. From these measures, showing a plain linear behavior, we extracted the values of the parameters for our RF power model.

The power analysis of the proposed forwarding optimization has been derived by combining the power figures of our

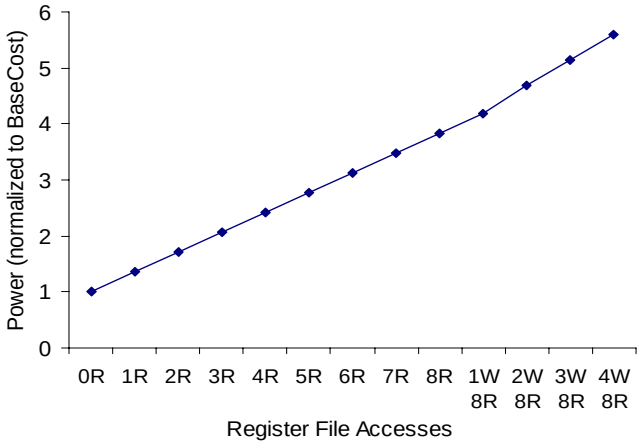

Figure 3: Average power consumption of RF read/write accesses (Values are normalized to BaseCost).

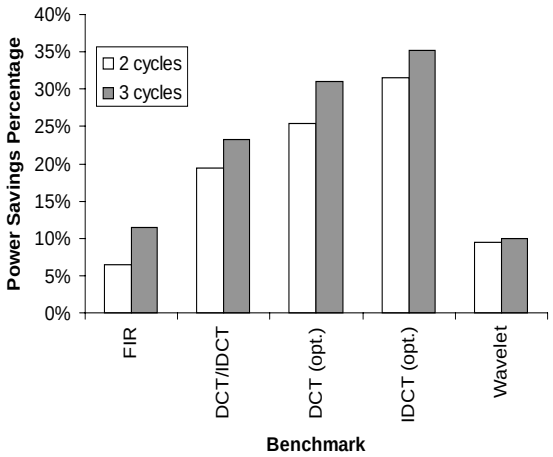

Figure 4: Power saving percentage of our optimization approach applied to registers liveness less than or equal to 2 (and 3) clock cycles.

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.

Figure 4 reports the power saving by applying the proposed low-power solutions to registers whose liveness length is less than or equal to 2 (3) clock cycles. In the case of 2 clock cycles, the average power saving for the given set of benchmarks is 18%, while the saving can reach up 32%. In the case of 3 clock cycles, the average power saving for the given set of benchmark is 22%, while the saving can reach up 35%.

As expected, higher power savings have been obtained by simulating the optimized versions of the DCT and the IDCT algorithms. In fact, the kernel of these algorithms 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.

5 Conclusions

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. The 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 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.

References

[1] J. Hennessy and D. A. Patterson, *"Computer Architecture: A Quantitative Approach,"* Morgan Kaufmann Publishers, San Mateo, CA, Second Edition, 1996.

[2] A. Abnous and N. Bagherzadeh, "Pipelining and Bypassing in a VLIW Processor," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 6, June 1994, pp. 658-663.

[3] H. Corporaal, *"Microprocessor Architectures: from VLIW to TTA,"* John Wiley and Sons, England, 1997.

[4] R. Yung and N. C. Wilhelm, "Caching Processor General Registers," *ICCD '95. Proceedings of IEEE International Conference on Computer Design*, 1995, pp. 307-312.

[5] M. M. Martin, A. Roth, C. N. Fischer, "Exploiting Dead Value Information," *MICRO-30, Proceedings of 30th Annual IEEE/ACM International Symposium on Microarchitecture*, 1997, pp. 125-135.

[6] A. Chandrakasan and R. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits," *Proc. of IEEE*, 83(4), pp. 498-523, 1995.

[7] K. Roy, S. C. Prasad *"Low-Power CMOS VLSI Circuit Design,"* John Wiley and Sons, Inc., Wiley-Interscience, 2000.

[8] V. Zyuban and P. Kogge, "The Energy Complexity of Register Files," *ISLPED98, Proceedings of International Symposium on Low-Power Electronic Design*, Monterey, CA-USA, 1998, pp. 305-310.

[9] A. V. Aho, R. Sethi, J. D. Ullman, *"Compilers: Principles, Techniques, and Tools,"* Addison-Wesley, 1986.

[10] P. Faraboschi, G. Brown, J. A. Fisher, G. Desoli, and F. Homehood, "Lx: A Technology Platform for Customizable VLIW Embedded Processing," *ISCA00, Proceedings of International Symposium on Computer Architecture*, Vancouver, BC, Canada, 2000, pp. 203-213.

[11] J. Fisher, *"Trace Scheduling: A Technique for Global Microcode Compaction"* IEEE Trans. on Computers, C-30(7):478-490. 1981.

[12] 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.

[13] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery *"Numerical Recipes in C : The Art of Scientific Computing "* Cambridge Univ Press. Jan 1993.