Exploiting reconfigurability for low-power control of embedded processors

L. Carro, E. Corrêa, R. Cardozo, F. Moraes, S. Bampi*
Universidade Federal do Rio Grande do Sul – Instituto de Informática
**Pontifícia Universidade Católica do RGS -- Faculdade de Informática*

Abstract

This paper proposes and evaluates a new implementation for the lowest level of instruction cache memories, which provides considerable power savings in the control memory subsystem of standard embedded processors. Instead of using power-demanding 6-T SRAMs for small I-caches, we exploit the possibility of using smaller switching capacitances, substituting the memory array with a specialized programmable logic circuit. Switching gains provided by this substitution are presented, illustrating a dramatic potential for power reduction in processor architectures for portable multimedia embedded applications.

1. Introduction

The memory subsystem is by far the most power demanding part of state-of-the-art electronic embedded processors [1-5]. The demanding power requirements of the memory subsystem can be traced both to significant volume requirements and to even stronger performance requirements. The volume requirements typically force long bit lines with large capacitances and power hungry sense amplifiers. The performance requirements arise from trying to match the speed of the CPU, forcing the switching of a large capacitance during a short period of time.

Yet another pressure increasing the power consumption of embedded processors stems from the current move towards 64-bit word widths, with the Intel MMX, HP Max-2 and Sun VIS [7-9] being preeminent examples of this trend to accommodate the large addressing space required by many programs. The use of 64-bit words implies that numerous bits on the instruction memory remain frequently unused. Experimental results to be shown indeed verify this hypothesis. Hence, power is continuously being expended only to read a value that is most likely invariable for longer instruction words.

In order to avoid power dissipation in the memory, the best alternative would be the avoidance of the memory accesses altogether. Actually, this has been experimented in [3], where a very small cache filter was used to concentrate accesses of frequently used instructions. Although this exploits the intense temporal locality of multimedia applications, it still does not address the shortcoming of storing redundant information (e.g. many zeros in a large instruction word) or the fact that many instructions are repeated as an intrinsic characteristic of RISC machines.

This work tries to exploit a different perspective on the instruction memory usage, by examining the substitution of the instruction memory by a reconfigurable combinational circuit, with far less power requirements. This proposition is based on the observation that, since only one row of the memory is active at any given cycle, switching activity of an equivalent logic circuit could be far less demanding than the whole SRAM array column switching, even when one considers that sense amplifiers do not require full swing switching in the bitlines.

This paper is organized as follows. Section 2 presents previous work on the issue of power savings in embedded processor architectures. The basic reasoning underlying the proposed approach is shown in section 3, while section 4 presents results comparing the required energy of the proposed solution with the energy expended on regular caches on a set of benchmarks representative of the target application domain. Finally, section 5 presents brief conclusions and outlines a few ideas for possible future work.

2. Previous work

The memory subsystem has been recognized as a major source of power dissipation in embedded processors. A set of approaches for dealing with this problem focuses on instruction compression [10-13]. Published results show gains on power savings, but it must be noted that, although there are fewer bits accessed per instruction memory read, the memory is still active, and hence it drains power. This places a fundamental limitation on compression-based mechanisms: a larger compression rate is harder to achieve, because one loses the favorable statistics on instruction distribution, and a smaller compression rate means more frequent memory access, and hence more power.

As mentioned before, the move towards 64-bit word widths implies that numerous bits on the instruction memory might be frequently unused and/or are invariable. These facts have already been used to save power, through dynamic reduction of the bit width of the datapath [1,14], and at the cache memory level through partial shut-off of memory columns [23].

On another end of the hardware implementation, FPGAs have been long used as the substrate target for computation acceleration, exploiting the characteristics of the problem to transform the underlying hardware [15]. In order to cope with the changing characteristics of each application during the execution of a program, partially reconfigurable devices have been proposed, and used mostly in the digital signal processing domain [16].

0-7803-7761-3/03/$17.00 ©2003 IEEE

3. Proposed Instruction Memory Block

As stated before, the memory subsystem is responsible for a large percentage of the total power dissipated in an embedded processor system. Any circuit that could replace the cache memory should behave accordingly in a miss, possibly without any added complexity.

In the approach proposed in this paper we try to exploit different concepts to reduce power. On the circuit level, we reduce the required energy per memory access by replacing the memory array by an equivalent low power switching circuit, at the price of extra area, while maintaining performance. At the architecture level, we exploit characteristics of code behavior (strong locality for a large set of important applications) and characteristics of modern architectures, where a few instructions are often reused (with a lot of memory words carrying the same bit-content in entire sub-fields), and the oversized instruction length (meaning that some information is present but not required). The grouping of all these concepts leads to a power efficient control architecture, as it will be showed. All the proposed modifications in the memory subsystem have small impact on the cache miss performance, allowing a smooth transition from the current memory hierarchy to the proposed circuit for a meaningful set of embedded applications.

We propose a fixed routing reconfigurable device that could provide the same bit information of a memory. Figure 1 presents a simplified schematic of the proposed reconfigurable circuit, showing the generation of products and the output sum for an example with 256 rows per 64 columns. The actual circuit of figure 1 is simply a set of configurable sums-of-products. This way, any logic function can be defined. $PP_{0..16383}$ are the latches programming the output product. By changing the contents of the control flip-flops, one can change the logic function of the memory substitution circuit. The total number of latches that must be used to program the output logic function is equal to the number of memory cells being replaced. Also, it should be noted that the decoder in figure 1 is the same as the one present in the memory circuit. In our proposal, a second decoder is needed for memory writes. Only one out of possible 256 rows will be active, at any given time.

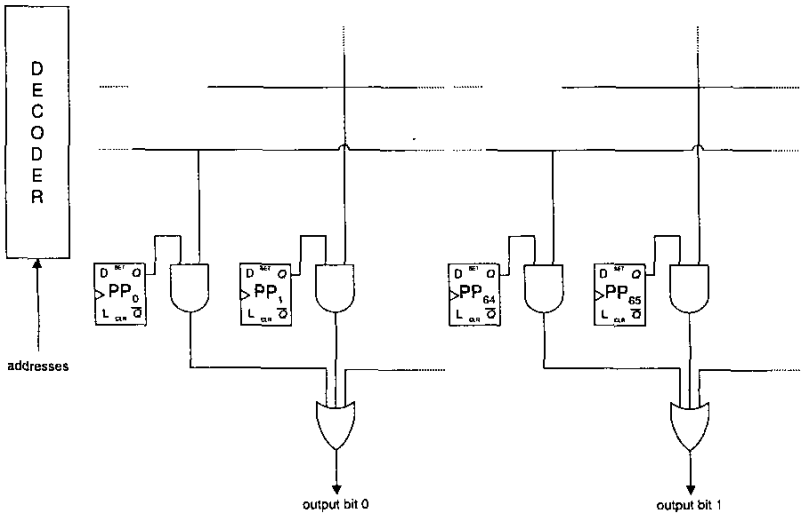

Figure 1 – Simplified schematic of reconfigurable circuit

The latches present in figure 1 will behave as the reconfiguration memory, and will be storing the same information present in the memory column. However, it should be noted that none of the latches is in the critical path during a memory read, only during a memory write. The purpose of the circuit then, is just to diminish the active capacitance in the critical path, maintaining performance and reducing power, at the cost of increased area.

In order to evaluate the actual power savings, one must take into consideration the effective switching of the OR-gate. In the way it is being used, only one of each possible row will be active. However, this does not mean that the active row will take part in the computation of the output. While reading a memory, some bits will be one, and some will be zero. The decoder will switch one line to zero (active to non-active, A→NA) and another from zero to one (NA→A). Taking an even distribution among ones and zeros, the activity at the OR output could be written as

$$\text{Activity} = \tfrac{1}{4}*A\rightarrow NA + \tfrac{1}{4}*NA\rightarrow A + \tfrac{1}{4}*A\rightarrow A + \tfrac{1}{4}*NA\rightarrow NA \quad (1)$$

So, at least ¼ of the transitions in the decoder will maintain the output of the OR at zero, and hence reduce power in the same proportion.

4. Experimental Results

The results focus on some multimedia benchmarks, so that the validity of the suggested approach can be verified. We have chosen to work with the following examples from [17]:

- g721 encoder-decoder;
- jpeg_DCT;
- adpcm coder-decoder.

The above examples are likely to be found in many applications, and they can also be present in the same application. In addition to these general-purpose benchmarks, we have also used some dedicated applications that have been developed for portable embedded devices, which are:

- Dic_translator: a subroutine to search a dictionary of words for language translation [18];
- Dic_acquisition: a subroutine that gets ASCII characters from a neural network and assembles them into a single word [18];
- Ocr_grade: one part of a neural network that recognizes individual characters [18];
- Ocr_propagation: second part of a neural network that recognizes individual characters [18];
- Crane: a crane control that reads various sensors and controls the speed of a crane moving on a chart [19];
- Ej: core of Extrapolated Jacobi-Iterative Method on a 128x128 grid [20];
- Sor: core of Successive Over-Relaxation algorithm on a *256x256 matrix [21]*;
- Tom: a fundamental part of the *tomcatv* SPEC95fp benchmark, with matrix size 128x128.

For each one of the above examples, we investigated the number of products that could be obtained from the instruction memory. The binary code for each program was studied, and a logic function obtained. Each of the examples was compiled using the SimpleScalar tool set [22], and the dumped binary code was analyzed. Each instruction word has 64 bits. For each memory bit line, we extracted a logic function, expressed it as a sum of products and counted all different products for each output bit. This number gives us the required number of logical ones and zeros to be stored in the reconfigurable circuit. All examples were analyzed in chunks of 256 instruction words, i.e., 8 bits of real addressing were used for a small I-cache access.

The mean value of needed products for all examples was 36, meaning that effectively, only a small part of the memory needs to be active. The analysis of the examples confirmed our initial hypothesis: a lot of instructions are repeated in the code, as well as register values, leading to a small number of products required to express the logic function. Moreover, a lot of memory outputs are zero-valued for the entire code block of 256 instructions, being kept unchanged during the application execution of said block.

In order to verify the proposed technique, we have used SPICE electrical simulator to design both the proposed circuit and a standard SRAM array. Allresults are for an AMS 0.35 micron, 3.3V CMOS technology. The memory size was fixed at 128 rows of 64-bits each, with 2ns to change the bit-line by 0.2V. The switching activity of a 128-input OR gate, constructed as a combination of AOI gates was also measured. The logic circuit was developed using an automatic tool for layout generation [24], and both circuits were simulated after pos-layout capacitance extraction. For routines or programs larger than 128 words, we assume that a replacement mechanism will take place, in the same way as it is done for a cache memory of the same size.

Table 1 presents the comparison results obtained by post-layout electrical simulations. Power dissipation in the memory accesses is independent of data, and was measured as 0.69mW per column, or 88.32 pJoules per read access in 2ns. The row P_r_circ in table 1 presents the power taken by the proposed circuit in the worst case switching situation, assuming there is one transition every cycle. The row P_r_3/4 presents results when there is an even distribution of ones and zeros in the output function, so that only ¼ of all reads are actual transitions. The row P_r_2/3 shows the power when we consider that the memory stores actual instructions, with only 1/3 of the products being needed for a certain output.

Table 1 – energy (pJ) required for each read access

Situation	Prog. Circuit	Memory/circuit
P_r_circ	110.87	1.25
P_r_3/4	67.70	0.77
P_r_2/3	27.86	0.31

As it can be seen in table 1, we changed the constant power dissipated in a read for a data dependent power. Depending of the set of instructions being read (as was the case in the used multimedia benchmark), a reduction by a factor of 3 can be achieved.

There remains to be tackled the fact that most programs nowadays use many more than 128 addresses to perform their computation. Examples are DCT_jpeg and g721. The writing of a new value in one control FF for each column takes roughly 817.92pJ. This is the value to be considered as the cost of replacing a single line of cache in the reconfigurable circuit.

The area overhead is a factor of 3.74, since each row of latches has to be programmed separately. There is the possibility of programming a different number of bits at a time, and also of changing each part of the circuit individually, as a normal cache would behave. As mentioned before, the replacement policy of the proposed circuit can be exactly the same as the one present in the cache it is trying to substitute.

Nevertheless, what actually matters is not the cost of a single replacement, but the total cost of replacements during execution of the whole algorithm. We have studied the cost of misses by using a cache simulator and by specifying the same cache capacity as the equivalent reconfigurable circuit (that is, 128 instruction words). In both cases, a miss rate less than 0.5% was found while executing the g721 and the DCT_jpeg routines. The number of misses is quite a small fraction of the total number of accesses requiring an instruction.

Even with the extra 817.92 pJ spent during a miss, the overall gain is still high, whenever the application has the same behavior as the ones here studied, when a lot of time is spent in relatively small loops.

6. Conclusions

This paper has shown results of a promising technique to substitute embedded processor memories or caches with functionally equivalent logic circuits. Because of larger word sizes, and the intensive use of a limited instruction set and register file, repeatability of instructions and registers favors the implementation of logic functions that switch only 30% of the time, when compared to a memory of the same capacity. The equivalent logic function can then be synthesized in a logic structure that allows aggressive power savings in the embedded memory subsystem.

It is important also to notice that the proposed technique provides potential gains far beyond instruction compression. This is because the memory is used only whenever there is an instruction miss, and not almost continuously, as under the compression paradigm. While reported power savings with compression are in the order of 50%, this architectural change can achieve three fold power savings, without compromising the critical path and retaining the traditional overall memory hierarchy.

References

[1] O. Hebert, I.C. Kraljic, Y. Savaria. **A method to derive application-specific embedded processing cores**. Codes 2000, pp. 88-92.

[2] Tomiyama, H.; Ishihara, T.; Inoue, A.; Yasuura, H. **Instruction Scheduling for Power Reduction in Processor-Based System Design**. DATE 1998, pp. 855-860.

[3] J. Kin, M. Gupta, W. Mangione-Smith. **Filtering Memory References to Increase Energy Efficiency**. IEEE Trans. on Computers, v.49, n.1, Jan 2000, pp. 1-15.

[4] W.-T. Shiue, C. Chakrabarti. **Memory Exploration for Low Power, Embedded Systems**. ISCAS 1999, v.1. pp. 250-253

[5] S. Wilton, N. Jouppi. **Cacti: An enhanced cache access and cycle time model**. IEEE Journal of Solid-State Circuits, vol. 31, no. 5, May 1996. pp. 677-688.

[6] A. Chandrakasan, R. Brodersen, editors. **Low-Power CMOS Design**. New York, IEEE Press, 1998, p. 629.

[7] A. Peleg, U. Weiser. **MMX Technology Extension to the Intel Architecture.** IEEE Micro, v. 16, n. 4, Aug. 1996. pp. 42-50.

[8] R. Lee. **Subword parallelism with MAX-2**. IEEE Micro, v.16, n.4, Aug. 1996. pp. 51-59.

[9] Kohn, L.; Maturana, G.; Tremblay, M.; Prabhu, A.; Zyner, G. **The Visual Instruction Set (VIS) in UltraSparc**. Proceedings COMPCON, 1995, pp. 462-469.

[10] Y.Yoshida. **A Low Power Consumption Architecture for Embedded Processors**. Electronics and Communications in Japan, Scripta Technica, Part 3, vol.81, no.10, Oct. 1998. pp. 83-90.

[11] H.Lekatsas, W. Wolf. **SAMC: A Code Compression Algorithm for Embedded Processors**. IEEE Trans. on CAD, v. 18, n. 12, Dec. 1999. pp. 1689-1701.

[12] T.Ishihara, H. Yasuura. **A Power Reduction Technique with Object Code Merging for Application Specific Embedded Processors**. DATE 2000, pp. 617-623.

[13] L. Benini, E. Macii, M. Poncino. **Selective instruction compression for memory energy reduction in embedded systems**. Proceedings of the International Symposium on Low Power Electronics and Design, 1999, pp. 206-211.

[14] D. Brooks, M. Martonosi. **Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance**. HPCA 1999, pp. 13-22.

[15] R. Hartenstein. **A decade of reconfigurable computing: a visionary retrospective**. Date 2001, pp. 642-649.

[16] Xilinx. **Virtex Series Configuration Architecture User Guide**. XAPP 151, v1.5, 2000, p. 45.

[17] Lee, C.; Potkonjak, M.; Mangione-Smith, W.H. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. Proc. Micro 30, 1997.

[18] D. Franco, L. Carro. **FPGA-based systems with linear and non-linear signal processing capabilities**. Proceedings of Euromicro 2000, IEEE Computer Society Press, pp. 260-264.

[19] E. Moser, W. Nebel. **Case study: system model of crane and embedded control**. DATE 1999, pp. 721-723.

[20] S. Nakamura. **Applied Numerical Methods with Software**. Prentice-Hall, Englewood Cliffs, N.J., 1991.

[21] M.E. Wolf, M.S. Lam, M. S. **A Data Locality Optimizing Algorithm**. Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation, June 1991, pp. 30-44.

[22] Burger, D.; Austin, T. **The SimpleScalar Tool Set, Version 2.0**. Technical Report 1342, University of Wisconsin-Madison, CS Dept., June 1997.

[23] Villa, L.; Zhang, M.; Asanovic, K. **Dynamic zero compression for cache energy reduction**. Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture, Monterey, CA, USA, 10-13 Dec. 2000. p.214-220.

[24] Moraes, F.; Robert, M; Auvergne, D. A Virtual CMOS Library Approach for Fast Layout Synthesis. In: **VLSI: Systems on a Chip**. Lisbon, 1999, p. 415-426.