Address Bus Power Exploration in Programmable Processors for Realization of Multimedia Applications

K. Tatas, M. Dasygenis, A. Argyriou,
N. Kroupis, D. Soudris, and A. Thanailakis

VLSI Design and Testing Center, Dept. of Electrical and Computer Eng.,
Democritus University of Thrace, Xanthi 67100, Greece
{ktatas, mdasyg, dsoudris}@ee.duth.gr

Abstract. Address bus encoding schemes are used in this paper to reduce the address bus power consumption in a general multimedia architecture executing four common motion estimation algorithms. The effect of previously applied data reuse transformations in order to reduce the power consumed on the data as well as on the instruction memories of the programmable architectures in combination with these encoding techniques is thoroughly explored and the results are extended to a multiprocessor environment.

1 Introduction

The increasing demand for portable multimedia systems has made the once ignored power dissipation, the dominant design consideration, which begins at the higher design levels, where the largest gains can be obtained [1]. At the same time, these systems require increased processing power in order to handle data-intensive algorithms in real-time. Towards the confrontation of this complex problem, mainly two approaches exist: the use of custom hardware processors, or the use of programmable ones, each of which has its own advantages and drawbacks. Still, both of them require the utilization of data-reuse transformations [2], in order to exploit the inherent data-reusability of the application, combined with partitioning [3, 4], to meet the tight performance and power constraints. This has been thoroughly explored in [5, 6], for a different number of processors and a plethora of motion estimation algorithms. The utilization of programmable processors, provides increased flexibility at the cost of additional power consumption due to the contribution of the instruction memory power to the power budget [5] which often overruns the savings obtained in the data memory, by the use of data-reuse transformations.

But in such microprocessor based systems, significant power savings can be achieved through the reduction of transition activity on the instruction memory address buses. Several encoding techniques have been proposed, some of them more suitable for data buses [7, 8], others more appropriate for address buses [9, 10]. Still, there is no related work focusing on multimedia applications or examining the buses in combination with memory hierarchies. In other words, the effect of data reuse transformations on the microprocessor address bus, has not been explored. This paper addresses exactly this oversight, tackling four widely used motion estimation algorithms, which are run on a general single or multiprocessor architecture. Interaction between data reuse transformations applied in order to reduce the data memory consumption and four common bus encoding techniques used to reduce the power consumption on the microprocessor address bus is explored in detail. Exploration is done for a single processor and the results are then extended to a multiprocessor environment.

The rest of the paper is organized as follows: Section 2 describes the architecture we have considered and the applications that were run on it, Section 3 briefly presents the existing work on bus encoding for low-power and the encoding schemes we applied in particular, while also describing data-reuse transformations. Section 4 gives the necessary high-level power estimation metrics and clarifies the methodology used in our calculations. Then, our experimental results are documented, illustrated and analyzed in Section 5. The final sixth Section is dedicated to the summation of the derived conclusions and a brief mention of our near-future plans.

2 Target Architecture and Applications

The general architecture of the embedded multimedia system we have considered is illustrated in Fig. 1. In [5], data memory hierarchy optimization for three memory hierarchy models (Distributed Memory Hierarchy (DMA), Shared Memory Hierarchy (SMA), Shared Distributed Memory Hierarchy (SDMH)) and for a varying number of processors (one or two), combined with 21 data-reuse transformations, led to the reduction of the data memory consumption. The dominant role of the instruction memory in the total power budget became evident then, and the use of caches was employed in order to reduce it [11]. But the power consumption on the address bus was not taken into account. This work comes as an extension of that effort, calculating the bus power consumption for the same transformations for a single processor and then extending the results in the case of multiple-processor architectures. Our applications are a group of widely used motion estimation algorithms known as Full-Search (FS), Three-Step Logarithmic Search (3-step Log.), Hierarchical Search (HS), and Parallel- Hierarchical One- Dimensional Search (PHODS) [12].

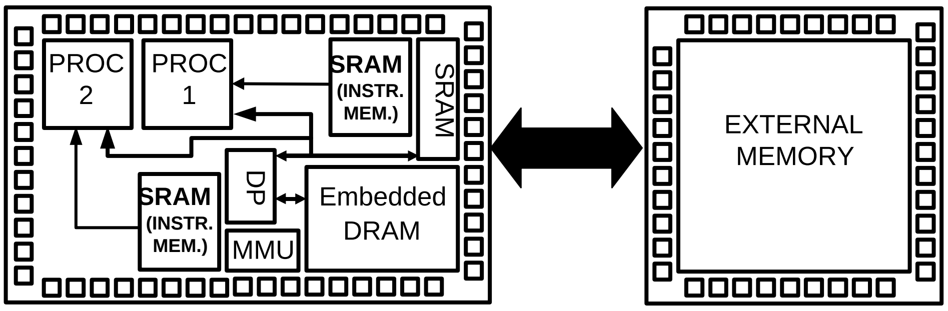

Fig. 1. General architecture of an embedded multimedia system

3 Bus Encoding for Low- Power and Data-Reuse Transformations

For the sake of completeness, basic concepts and some of the existing work on bus encoding for low-power as well as the subject of data-reuse transformations is briefly discussed here. Originally, encoding schemes were used for error correction and attempted to maximize the Hamming distance between transmitted words. Obviously, in the case of bus encoding for low-power the objective is the opposite one: Hamming distance between consecutive messages should be minimized. Bus encoding schemes can be classified in many ways, according to application (data or address bus), spatial or temporal redundancy (addition of lines or cycles) and generation principle (algebraic, permutation and probabilistic). Among the most popular encoding schemes are the ones we applied:

1. Bus- Invert code [7]: If the Hamming distance of the word to be transmitted is more than half the number of bus lines, the word is inverted. An additional line is used to inform the decoder if the received word should be inverted or not. This code guarantees a maximum of half transitions per line for random data. It is effective for data buses, but not for address buses where transmitted words tend to be sequential. It can be classified as a spatially redundant, algebraic data bus -oriented code.
2. Gray code [8]: A permutation code that guarantees one transition each time for sequential with step one. Suitable for address buses, when each address is incremented by one each time.
3. T0 code [9]: An encoding scheme that guarantees no transitions for an infinite stream of sequential (with any constant stride value) words, which makes it suitable for address buses. If the word to be transmitted is the previous one plus the stride value, the previous word is sent, so that no transition activity occurs. Otherwise, the word itself is transmitted. An additional line is used to inform the decoder if the received uncoded word is an increment of the previous one, or the coded word itself.
4. T0 - XOR code [10]: An encoding scheme that exploits the decorrelating abilities of the XOR function instead of a redundant line. Also effective for address buses.

The objective of employing data-reuse transformations is the efficient manipulation of memory data transfers. For that purpose, we performed an exhaustive data reuse exploration of the application's data. In order to employ data reuse transformations, we determined the specific data sets, which are used very frequently and within a short period of time.

The reused data can be stored in smaller on-chip memories, which require less power per access. In this way, redundant accesses to large off-chip memories are removed, resulting into a less power consumption. Of course, data reuse exploration has to decide which data sets are appropriate to be placed in separate memory. Otherwise, we would need a lot of different memories for each data set resulting in a significant area penalty. These data-reuse transformations have been thoroughly explored in [2] and [5] for similar multimedia applications.

4 Bus Power Consumption Metrics and Simulation Methodology

The average power dissipated on a bus line with a capacitance of C, operating frequency f, power supply voltage V_{dd} , and an average number of n transitions, is given by the formula [7]:

$$P_{average} = \frac{1}{2} C V_{dd}^2 n f \tag{1}$$

For a stream of encoded words of length L, the average number of transitions is given by:

$$n = \frac{\sum_{t=0}^{L-1} H(B(t), B(t+1))}{L-1} \tag{2}$$

where $B(t)$ and $B(t+1)$ are the code words transmitted on the bus and $H(a, b)$ the Hamming distance between two words a and b. The microprocessor architecture considered in our experiments was the ARM 7 and the ARMulator tool [13] was used in our calculations. ARM has a 32-bit address bus. The Hamming distance between the addresses of the generated instructions for the previously mentioned motion estimation algorithms was calculated for binary encoding (reference), plus four popular bus encoding schemes. Namely, the bus-invert code (BI) [7], the Gray code [8], The T0 code [9], and the TO-XOR code [10]. Then (1) was used to calculate the average power dissipation on the address bus. We assumed a 5 pF capacitance value for the on-chip address bus, and a 25 MHz frequency for the processor and the bus. Our methodology can be illustrated in Fig. 2.

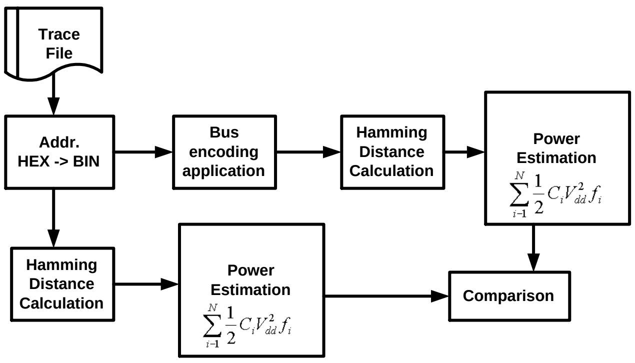
Trace File
Addr. HEX -> BIN
Bus encoding application
Hamming Distance Calculation
Power Estimation
$\sum_{i=1}^{N} \frac{1}{2} C_i V_{dd}^2 f_i$
Hamming Distance Calculation
Power Estimation
$\sum_{i=1}^{N} \frac{1}{2} C_i V_{dd}^2 f_i$
Comparison

Fig. 2. Bus power consumption high-level estimation flow

The ARMulator provides the trace file that contains a list of the instruction addresses in hexadecimal form. Those are first converted to binary. Then the flow is split in two branches. In particular the first branch, calculates the power consumption without the application of an encoding technique (binary encoding), while the second one calculates the address bus power consumption after the application of any encoding scheme. In the first case, the Hamming distance between consecutive addresses is calculated. After that, power estimation according to (1) for the parameter values mentioned above, is straightforward. In the second case, after the conversion of the instruction addresses from hexadecimal to binary, high-level (C code) descriptions of the four bus-encoding schemes are applied to the trace file (after conversion to binary) and the Hamming distance is calculated again for each of the encoding schemes. Power estimation is then performed in exactly the same way as in the first branch. Finally, comparisons among the estimated power, led to the results documented in the next section. Since Hamming distance calculation is time-consuming due to huge trace file sizes (over 700 Mbytes and 20 million instructions), this whole procedure was executed for one iteration of the outermost loop of the applications and was automated as much as possible, in order to minimize manual effort. That required the application of global loop transformations to the HS and PHODS algorithms, in order to give them such a loop structure. The assumption that the results obtained by one iteration of the outermost loop of each application would be accurate, was confirmed by a number of experiments [11].

5 Experimental Results

Relative power for all combinations of transformation and encoding scheme for each of the FS, 3-step Log., HS and PHODS motion applications are illustrated in Fig. 3, 4, 5, and 6, respectively. The leftmost transformation on the x- axis (designated “0”) is the original algorithm, in other words no data-reuse transformation has been applied. Bus-Invert encoding has virtually no effect, since it is a bus encoding scheme best suited for data buses, and relies in guaranteeing half or less of the lines making a transition in time, which is good for random data, but microprocessor instruction addresses are highly correlated if not sequential. Therefore, as can be seen in Fig. 3, the binary and Bus-Invert curves, are identical. Gray encoding which is generally considered a good option for address bus encoding, fails to obtain any power savings, and actually increases the number of transitions, because the addresses in our case are not incremented by one, but mostly by a stride value of four. The reason is that in the ARM7 an address refers to a byte, not a word. Therefore the program counter is incremented by 4 every time (in the absence of a branch) [14]. Employing the T0 encoding scheme, using four as the stride value, yields promising results, reducing power consumption by 50-80 % for all four applications. In particular, in Full- Search and PHODS the average savings are approximately 55%, while in Hierarchical Search and 3- step Logarithmic Search they are in the order of 75%. The T0-XOR though for the same stride value does not produce the expected promising results, probably due to the jumps in the program flow whose effect cannot be balanced by the decorrelating ability of the XOR operation.

The efficiency of the T0 encoding technique with a stride value of four, depends solely on how many of the addresses are incremented by the stride value, or how few “jumps” can be found in the trace file. The reason T0 encoding yields better results when applied to the 3-step Logarithmic and HS is that there are fewer jumps in the address

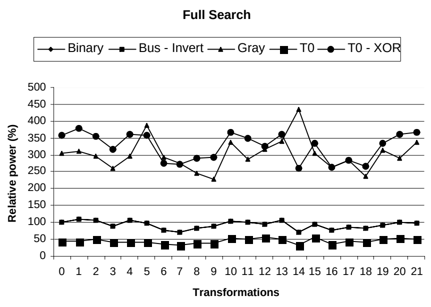

Fig. 3. Relative power consumption in comparison to the original algorithm with binary encoding for Full-Search

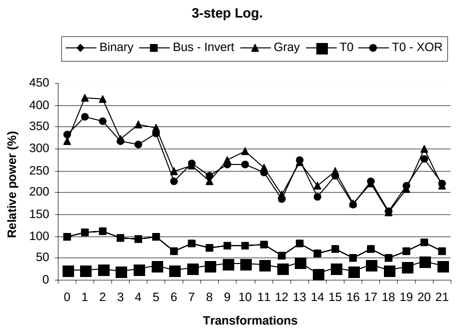

Fig. 4. Relative power consumption in comparison to the original algorithm with binary encoding for 3-step Logarithmic Search

stream. In order to justify this, one must look into the complexity and loop structure of the four algorithms [12]. A jump occurs when there is a branch in the assembly program flow. Loops are implemented as branches [15]. Therefore, a jump generally occurs when the C program flow goes from the execution of an inner loop command to an outer loop one. Address generation is done by the compiler automatically. The FS algorithm has the most regular loop structure, but the highest number of iterations (complexity). PHODS on the other hand, has low complexity (few iterations), but the most irregular loop structure, as it performs two separate searches, one in each dimension. That is why T0 performs less efficiently on those two algorithms. HS has the lowest complexity and a pretty regular loop structure, after the application of global loop transformations, and 3-step Logarithmic Search is of medium number of iterations and regularity. That explains why T0 performs more efficiently on those two.

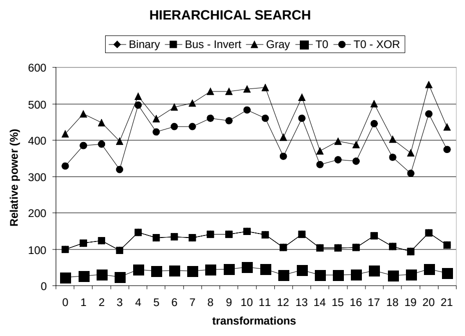

Fig. 5. Relative power consumption in comparison to the original algorithm with binary encoding for Hierarchical Search

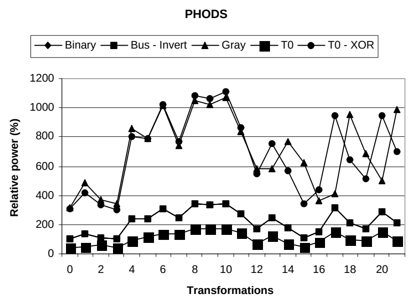

Fig. 6. Relative power consumption in comparison to the original algorithm with binary encoding for Parallel- Hierarchical One- Dimensional Search

After careful observation of the above charts in combination with the instruction memory power consumption results obtained in [11], the conclusion that the curves closely follow the instruction memory power consumption becomes visible. The reason is that each time there is a transfer on the bus (which results to switching activity and therefore power consumption), there is a corresponding instruction fetch, from the instruction memory. Since transfers, are the dominant cause for power dissipation in instruction memories and not storage (since they are mostly EPROMS), the power consumption on the bus closely resembles that of the instruction memory power consumption.

The above important conclusion can be generalized in the case of a greater number of processors. In that case, partitioning of the application does lead to less energy being dissipated by each processor because only a fraction of the original number of iterations is executed. The code size remains approximately the same, but an increased number of control operations causes some transformations to increase the instruction memory consumption dramatically. The same thing is expected to happen to the instruction bus power consumption and should be further studied. Also this conclusion would indicate that instruction memory power optimization techniques (such as the utilization of instruction caches), would also positively affect the address bus power consumption too.

The effect of the encoding on the transformations is illustrated in Fig. 7, where the relative power savings for the T0 encoding scheme is plotted against the transformations. If we attempt a comparison between this figure and the corresponding one giving the relative power savings for the instruction memory consumption for each transformation [13], it seems that in most cases, the transformations that yield the least efficient results without encoding are also the ones additionally affected in the least by the power encoding techniques. In other words, it seems that the more power-efficient the transformation (instruction memory) by itself (without encoding) is, the more efficient the encoding also becomes, saving even more power.

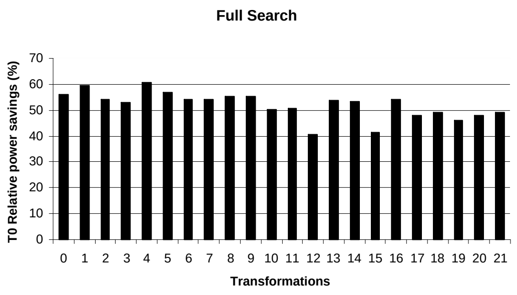

Fig. 7. Relative power savings by T0 encoding for each transformation for Full- Search

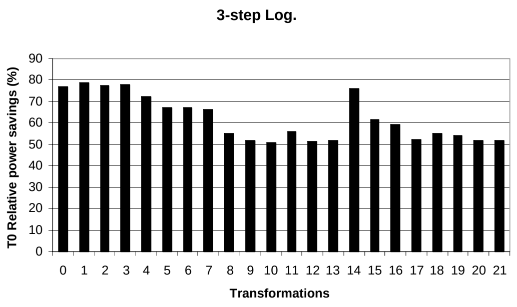

Fig. 8. Relative power savings by T0 encoding for each transformation for 3-step Logarithmic Search

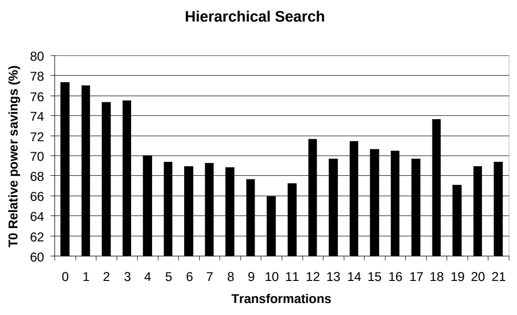

Fig. 9. Relative power savings by T0 encoding for each transformation for Hierarchical Search

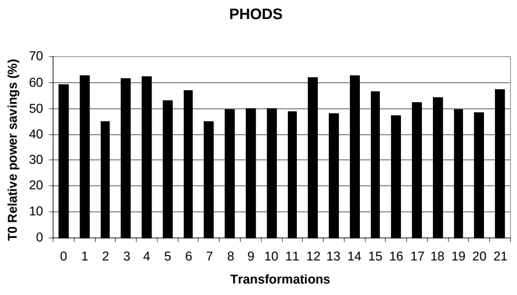

Fig. 10. Relative power savings by T0 encoding for each transformation for Parallel- Hierarchical One- Dimensional Search

6 Conclusions - Future Work

Popular encoding schemes were used in order to reduce the address bus power consumption in a programmable multimedia system running four common multimedia kernels, where data- reuse transformations had been previously applied in order to reduce the data memory power consumption. Exhaustive exploration of the interaction between the encoding schemes and the data-reuse transformations has revealed the close connection between the instruction memory power consumption and the address bus power consumption and allowed the generalization in the case of a multiprocessor platform. Our plans for the continuation of this work include the exploration of the interaction between instruction memory caches and bus encoding for low power on one hand and the introduction of new encoding schemes specific for applications like those mentioned here, taking into account the particular features of this type of algorithms

Acknowledgement

This work was supported by the ED 501 PENED‘99 project funded by G. S. R. T. of the Greek Ministry of Development and European Union

References

1. A. P. Chandrakasan, R. W. Brodersen: Low Power Digital CMOS Design. Kluwer Academic Publishers, Boston (1998)
2. F. Catthoor, S. Wuytack et al.: Custom Memory Management Methodology. Kluwer Academic Publishers, Boston (1998)
3. K. Masselos, F. Catthoor, and C.E. Goutis: Strategy for Power Efficient Design of Parallel Systems. IEEE Trans. on VLSI, vol. 7, No. 2 (1999) 258-265
4. U. Eckhardt, R. Merker: Hierarchical Algorithm Partitioning at System Level for an Improved Utilization of Memory Structures. IEEE Trans. on CAD, Vol 18, No 1, (1999) 14-23
5. D. Soudris, N. D. Zervas, A. Argyriou, M. Dasygenis, K. Tatas, C. Goutis, and A. Thanailakis: Data-Reuse and Parallel Embedded Architectures for Low-Power, Real-Time Multimedia Applications. IEEE International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), Gttingen, Germany (2000) 243- 254
6. K.Tatas, A. Argyriou, M. Dasigenis, D. Soudris and N. D. Zervas: Memory Hierarchy Optimization of Multimedia Applications on Programmable Embedded Cores. IEEE International Symposium on Quality Electronic Design (ISQED) 2001, San Jose, USA (2001) 456-461
7. M. R. Stan and W.P. Burleson: Coding a terminated bus for low-power. Proc. Of Fifth Great Lakes Symp. On VLSI (1995) 70-73
8. C. L. Su, C. Y. Tsui and A. M. Despain: Saving power in the control path of embedded processors. IEEE Design and Test of Computers, Vol. 11, No. 4 (1994) 24-30
9. L. Benini, G. DeMicheli, E. Macii, D. Sciuto, and C. Silvano: Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems. Proc. of The Seventh Great Lakes Symp. On VLSI (1997) 77-82
10. W. Fornaciari, M. Polentarutti, D. Sciuto, and C. Silvano: Power optimization of system-level address buses based on software profiling. Proc. of the Eighth Int‘l Workshop on Hardware/Software Codesign (2000) 29-33
11. M. Dasigenis, N. Kroupis, A. Argyriou, K. Tatas, D. Soudris, and A. Thanailakis: A Memory Management Approach for Efficient Implementation of Multimedia Kernels on Programmable Architectures. Proc. of the IEEE Computer Society Workshop on VLSI (WVLSI) 2001, Orlando, Florida (2001) 171-176
12. V. Bhaskaran and K. Konstantinides: Image and Video Compression Standards. Kluwer Academic Publishers, Boston (1998)
13. ARM software development toolkit, v2.11, Advanced RISC Machines (1996-7)
14. Dave Jaggar: Advanced RISC Machines Architectural Reference Manual. London: Prentice Hall (1995)
15. Wayne Wolf: Computers as Components, Principles of Embedded Computing System Design. Morgan Kaufmann Publishers (2000)