Using an Energy Aware Compiler Framework to Evaluate Changes in Register File Size towards ASIP-Design *

L. Wehmeyer, M.K. Jain, S. Steinke, P. Marwedel, M. Balakrishnan
{wehmeyer,steinke,marwedel}@ls12.cs.uni-dortmund.de, {manoj,mbala}@cse.iitd.ernet.in

February 9, 2001

Abstract

Interest in synthesis of Application Specific Instruction Set Processors (ASIPs) has increased considerably in the past few years and a number of methodologies have been proposed for ASIP design. A major part of this design consists of the steps application analysis, architectural design space exploration and instruction set generation. Using a configurable, energy aware compiler framework, we show that the number of registers available in the target processor has a strong effect on the quality of the generated code. The effects of varying the number of registers on code size, execution time as well as power and energy consumption were observed. Improvements in energy consumption by an average of 22% are possible by adding one more register. Our goal is to determine how many registers an ASIP should have to provide adequate application performance. We also identify some application characteristics that indicate the optimum number of registers.

1 Introduction

Application Specific Instruction Set Processors (ASIPs) are an intermediate form between Application Specific Integrated Circuits (ASICs) and general purpose (programmable) processors [1, 2, 3]. The advantage of ASIPs is that they offer high flexibility and performance at relatively low costs. ASIC development is often too expensive whereas general purpose processors do not provide the required performance. Thus, ASIPs can be efficiently utilized in a variety of embedded systems used e.g. in digital signal processing or mobile phones [2, 3]. Usually, ASIPs are designed for a small set of applications, thus enabling the designer to exploit knowledge about application characteristics so as to meet the requirements concerning performance, cost and power.

A recent survey on the approaches suggested for ASIP design methodologies [4] identified, among others, the following key steps in ASIP design: application analysis, architectural design space exploration and instruction set generation. To support these steps, we investigated one factor in ASIP design that has not been studied extensively: the recommended number of registers. This figure has a direct influence on the performance of the application, since too few registers lead to the insertion of spill code, which results in an increase in code size, execution time and energy consumption.

The compiler *encc* [5] was used in order to gather information about the behavior of applications that are run on processors having varying numbers of registers. This attribute of a proposed target processor can easily be changed by reconfiguring the compiler. We analyzed the behavior of generated assembly code during simulation. This information allows the ASIP designer to make a first judgement about the number of registers required within the processor to meet the criteria using a certain cost model. Using the compiler framework, it is possible to perform profiling runs of compiled applications to determine their energy consumption. In this way, the dimension of energy can be added to the considered design space. We show in this work that an average reduction in energy consumption by 22% can be achieved by adding one more register.

The rest of this paper is organized as follows: Chapter 2 gives an overview of the related work performed in this area. Chapter 3 describes the environment and general structure of the *encc*-compiler framework. The experimental setup used to gather the information presented in this work can be found in chapter 4. Chapter 5 shows the obtained observations and results. Also, some conclusions on application characteristics that explain the observed behavior are given. Chapter 6 concludes this work.

2 Related Work

Even though a number of parameters like pipeline depth, number and kind of functional units, issue width and the size of caches have been considered in the architecture models suggested by various approaches [6, 7, 8, 9], additional parameters have to be identified to further enlarge the design space. At the same time, these new parameters have to be constrained in such a way that the search for an acceptable solution remains feasible. Several design methodologies have been suggested for ASIP design [4]. Some approaches [2, 10, 11] use an application analysis tool to extract suitable application characteristics to assist the further steps in the ASIP design process. All approaches employ a parameterized architecture model for design space exploration. One new architectural parameter which has not been considered for design space exploration at higher level so

*This work has been supported by Agilent Technologies, USA

far is the number of registers. Though Gupta et al. [10] have considered this parameter, resource constraints are ignored during evaluation in their approach.

One recent publication by Brooks et al. [12] presents a framework enabling power analysis and investigation of the effect of hardware modifications as well as compiler optimizations. Their power analysis is based on parameterizable power models of common structures found in modern mircoprocessors.

To perform experiments and obtain observations, we chose to use a reconfigurable compiler that could easily be adapted to support an arbitrary number of registers. There are a number of compilers that support this feature, e.g. the Trimaran compiler [13] which uses the machine description language MDES to model the underlying hardware. The work of Leupers [14] supports retargeting a compiler for the DSP domain to different target architectures described in MIMOLA [15], a VHDL-like hardware description language.

The compiler used in this work, *encc*, is being developed at the University of Dortmund, Germany. The energy cost function incorporated into *encc* is based on Tiwari's power model [16], but this model was extended to additionally take into account the effect of accesses to different kinds of memory.

3 The *encc* compiler framework

In this work, the *encc* compiler framework was chosen for code generation and performance evaluation. *encc* is an energy aware compiler developed for the RISC class of architectures, capable of generating code with reduced energy consumption. Due to its reconfigurability, it is relatively easy to vary architectural parameters like the number of registers and generate code for a processor featuring an arbitrary number of registers. In this work, the configuration was initially set up to describe the *ARM7TDMI* processor [17]. Only the number of registers was then successively changed to generate a compiler for processors differing in their number of registers. The performance of the generated programs was evaluated in a simulation run and conclusions were drawn from the observed results. Additionally, some observations were taken using an initial setup suitable for the LEON-processor, a free SPARC V8-implementation developed at ESA [18]. Since the results are quite similar to those using the ARM-processor, the results obtained with the LEON architecture are not presented in this work.

3.1 Power Model used in *encc*

The cost model for power incorporated in *encc* enables the compiler to take decisions depending on the amount of energy dissipated when instructions are executed on the target processor. This power information is also used for profiling during simulation runs as we investigate the overall energy consumption of benchmark programs.

The power model used in the compiler is based on the model developed by Tiwari et al. [16], which distinguishes between basic costs and inter-instruction effects. The basic costs consist of the measured current during execution of a single instruction in a loop. The energy dissipated due to the change of internal circuit states for sequences of different instructions are summed up in the inter-instruction effects which are ignored here since their overall impact on our findings is negligible.

Detailed instruction level information on energy consumption for the *ARM7TDMI* processor is not readily available. Therefore, the values have to be determined by experimental physical measurement using an evaluation board. During this stage, one instruction is placed in a loop many times (to avoid influences caused by loop overhead) and executed over a longer period of time. The average current is measured and captured in a power database. This database is then used to take decisions within the compiler.

The values contained in the power database represent the average current measured during the execution of an instruction. This is sufficient to represent power since the supply voltage can be assumed to be constant and power is defined as $P = V * I$. To obtain information on energy, the power information has to be multiplied with the number of cycles required to complete the concerned instruction, since $E = P * t$.

The database is not only used during code generation, but also to determine the amount of energy dissipated by a complete program. This is done by counting all instructions during simulation of the program and adding up their respective energy contributions.

One major enhancement to Tiwari's power model is that *encc* also considers power consumption of the memory system. This contribution should not be ignored since the energy consumed by off chip memory can be significantly higher than the amount dissipated by the processor itself.

3.2 Workflow within the *encc* framework

The input for *encc* consists of an application program written in C. This file is parsed and standard optimizations are performed using the LANCE frontend [19]. The backend of the compiler then generates assembly instructions using the tree pattern matcher *olive* [20].

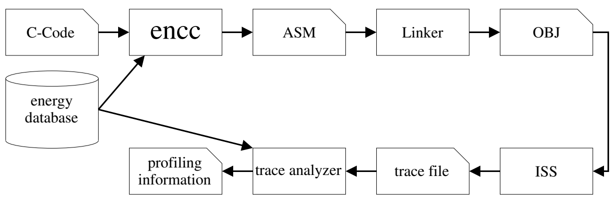

Figure 1: Workflow within the *encc* framework

During this phase, optimization can be performed with respect to one of the following criterions: execution time, code size or energy. In the case of energy, the power database mentioned above is accessed and information about instruction sequences is gathered to decide which combination should be used.

After assembly and linking, the program is executed using an instruction set simulator contained in the ARM Software Development Toolkit [17]. The result of this simulation is a trace file containing all instructions executed as well as some information on memory accesses. This data is passed to the trace analyzer which calculates the number of accesses to different kinds of memories, the number of bytes occupied in instruction and data memory as well as overall energy consumption.

4 Experimental Setup

This section briefly describes the chosen benchmark applications and the ARM7TDMI processor by ARM Ltd. [17]. The ARM7TDMI was chosen as the inital platform for the compiler *encc* because it is widely used in embedded systems today, and is particularly recommended for low energy applications, making it a suitable target for an energy aware compiler.

4.1 Benchmark Suite

To investigate the effect of changing the register file size, a number of benchmark applications are necessary. These benchmarks were taken from the typical domain of the ARM7TDMI processor:

• biquad_N_sections	DSP
• lattice_init	DSP
• matrix-mult	mult. of two m x n matrices
• me_ivlin	media application
• bubble_sort	standard sorting algorithm
• heap_sort	standard sorting algorithm
• insertion_sort	standard sorting algorithm
• selection_sort	standard sorting algorithm

Table 1: *Benchmarks used for the experiments*

Note that the ARM7TDMI does not feature a floating point unit. Since the use of floats or doubles within the benchmarks would result in inefficient library calls, only data of type of integer is used in our benchmarks.

4.2 The *ARM7TDMI* Processor

The *ARM7TDMI* [17] is a 32-bit RISC processor offering high performance combined with low energy consumption. It features a 32 bit RISC instruction set (called ARM), 16 general purpose registers with 32 bits each as well as an ALU, a multiplication unit and a dedicated barrel shifter, but no data or instruction cache.

To reduce energy consumption, the *ARM7TDMI* features an additional 16-bit THUMB instruction set which has a reduced functionality compared to the 32 bit ARM version, e.g. only eight of the 16 general purpose registers are available in THUMB mode. Energy savings compared to the ARM instruction set are due to the higher code density and the fact that only 16 bits of instruction data have to be fetched for each instruction instead of 32. Since memory accesses consume a large amount of energy, this technique results in lower energy dissipation during program execution.

5 Results

In this section we first present the methodology used to gather the data presented in this work. Along with the obtained values, some explanations for the observed results are given.

5.1 Methodology

Using the compiler *encc* described in chapter 3, the number of registers available for the application is varied in the compiler configuration file in the range from three to eight. This range was chosen since most of our benchmarks do not benefit from more than eight registers. The remaining parameters in the configuration file are always kept at the values appropriate for the *ARM7TDMI* processor. Note however, that some of the potential positive effects when using less registers, i.e. less power dissipation due to less chip area in the ASIP as well as the resulting different instruction set, are not considered in this work.

For each number of registers, a customized compiler is generated. This compiler framework is then used to generate code and, subsequently, profiling information for our benchmark programs.

encc supports code optimization for execution time, code size or energy. In this work we show the values obtained using the 'time' option, since the other optimization goals show similar trends in the results.

The values taken consist of

- ratio of spill instructions to total static code size
- number of executed instructions
- number of cycles
- average power dissipation
- energy consumption

encc supports the use of different memory types in such a way that the developer may choose where instructions and data are to be stored. We assumed the memory layout of the Atmel AT91M40400 evaluation board [21] that was also used to take the measurements for the power model. For the results presented in this work, instructions are stored in the 4kB on chip memory, data is stored in off chip memory (cf. figure 2). This coincides with the situation often found in embedded systems, where the program is stored in low-power ROM and only the data is stored in external RAM.

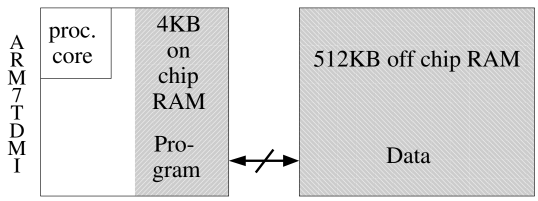

Figure 2: Memory configuration of the Atmel AT91M40400 evaluation board

5.2 Results for ratio of spill instructions to total static code size

The results obtained for ratio of spill instructions to total static code size are shown in figure 3. Spilling instructions are inserted into the code when a value needs to be saved (*"spilled"*) to memory. The number of spill instructions (made up of load, store and address calculation) is determined at compile time.

As one would expect, the ratio of spill code decreases with increasing number of registers. This is due to the fact that spill code is only introduced during compilation when there are insufficient registers available within the processor to hold all values that are required for a later instruction (so called 'live' values) [22, 23]. In our case, we find that the 'saturation' of the benchmark applications, i.e. the point where no more spillcode is introduced, varies between 7 and 19 registers (the latter value for *me_ivlin* is not visible in the figure).

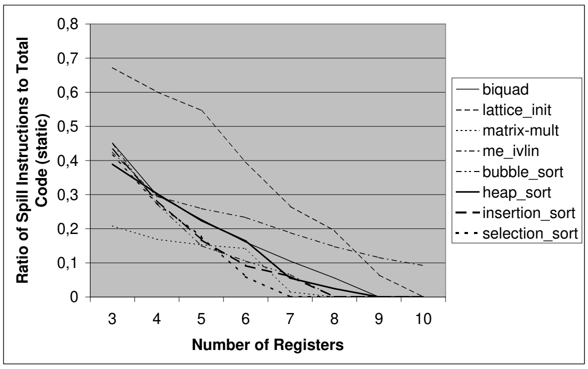

Figure 3: Ratio of spill instructions to total number of instructions

These results show the register pressure, i.e. the number of values that are live at the same time. However, since this analysis is purely static, the information is of little use for dynamic runtime analysis of applications: Spill code introduced within the innermost loop of an application is executed many times, but only counted once for this matter. Therefore, more observations concerning the impact on performance and energy have to be taken based on a profiling simulation of the application.

5.3 Results for Number of Executed Instructions

The results obtained for the number of executed instructions are shown in figure 4. The values for the different benchmark application were scaled in this figure to produce the results in a single plot. This is acceptable since the trends of the curves are preserved.

Once again, the relationship between the number of executed instructions and the number of registers is due to spill code. Since this analysis was performed on a dynamic basis, i.e. the number of instructions was counted during execution of the program, the resulting plots now show more explicit trends. Spill code inserted within an inner loop is now counted each time the loop is executed.

It is important at this stage that the compiler used for the experiments has a method of determining potential spill costs so that register values are spilled in a suitable order. If the very first register spilled to memory is used within an inner loop (with high spill costs), the resulting program will show a severe performance degradation, even if only this one register is spilled. *encc*, as many other compilers, supports this kind of analysis.

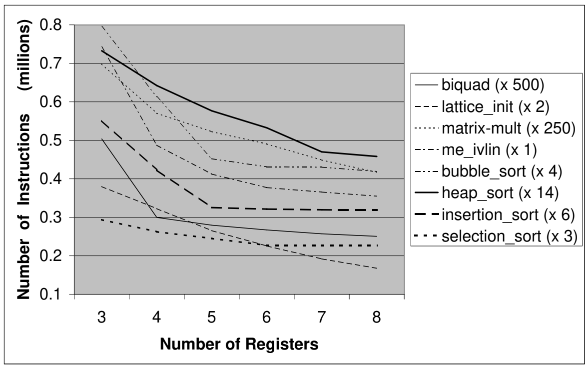

Figure 4: Number of Executed Instructions over number of registers

In some curves, there is one noticeable sharp bend (knee), e.g. in the programs *bubble_sort* and *insertion_sort* that shall be discussed here exemplarily. These applications both have their knee at five registers, meaning that if the number of registers is decreased below this number, application performance degrades drastically.

Looking at the source code, we found an application characteristic that is responsible for this phenomenon: the source code of both programs contains an inner loop containing statements that require five registers to hold all live values. This explains the knee in the curve: since five registers are required to hold all live values, reducing the number of registers to four will result in spilling within the innermost loop, leading to a severe performance degradation. The knees in most of the other applications' curves can be explained in a similar way.

5.4 Results for Number of Cycles

The results obtained for number of cycles are shown in figure 5. Again, the values are scaled appropriately.

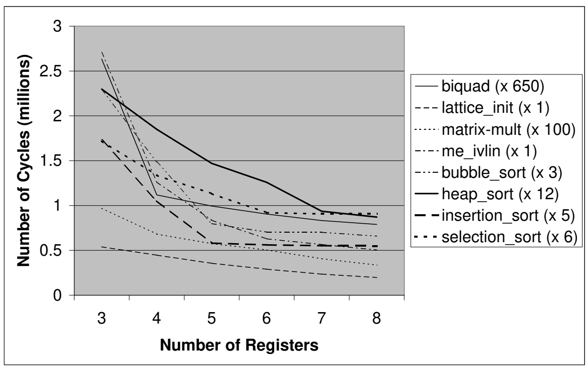

Figure 5: Number of cycles over number of registers

The general behavior of these curves is similar to the previous figure due to the fact that the cycles are linked to the instructions via a simple linear function associating each instruction with its execution time in cycles. In the case of the ARM7TDMI, the number of cycles required for the execution of one instruction can be assumed to vary from one to five.

The observed behavior of the applications becomes even clearer in this figure since the inserted spill instructions consist mainly of load and store instructions, which in general require more cycle time than e.g. arithmetic operations. Therefore, one additional spill instruction will usually result in the application requiring several more cycles every time the spill instruction is executed.

The knees for the curves all remain at the same spots, it is just the slope that changes. Thus, the same application characteristics that were discussed in the previous section can be used to explain these observations as well.

5.5 Results for Power Consumption

The results obtained for the average power consumption in relation to the number of available registers are shown in figure 6.

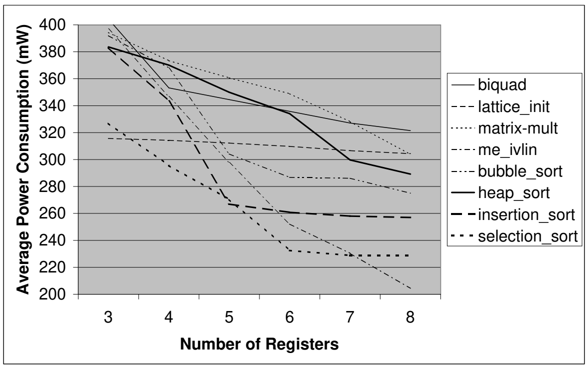

Figure 6: Average power consumption over number of registers

Once again, the shape of the curves can be explained by the inserted spill instructions. Data load and store instructions are introduced, increasing power consumption significantly since accesses to off chip memory are very power consuming in RISC architectures. Since the instructions are stored in the on chip memory and only data elements are stored in off chip memory for our observations, the data access instructions related to spilling consume a large percentage of the overall power of the application. Some data was also taken with both instructions and data in off chip memories, but in this case, the power values were nearly constant, i.e. independent of the number of registers since energy dissipation for fetching instructions was clearly dominant.

Some of the curves have knees similar to those in figure 5. This is due to the fact that spill instructions consume both more time and more power than other, e.g. arithmetic, instructions.

However, it is also interesting to take a look at the absolute values for power consumption of the various benchmarks, especially since all curves are shown to scale:

The power values are highest for the program *matrix-mult*. This is because a multiplication of two matrices requires two array elements (which are always stored in memory) to be accessed for reading. Then one multiplication must be performed, and finally the result must be written back to memory. Since three memory accesses are required and multiplication is also a power consuming instruction for the *ARM7TDMI* processor, the overall power consumption of this application is comparatively high.

As for the sorting algorithms, their power consumption ranges from high (*heap_sort*) to rather low (*selection_sort*). This is mainly due to the extent of data accesses to off chip memory, which is very high for the used implementation of *heap_sort* and very much lower for *selection_sort*. *bubble_sort* and *heap_sort* show comparable results since their inner loops have a similar structure, which also holds for the number of off chip memory accesses.

5.6 Results for Energy Consumption

The figure for energy consumption related to number of registers (fig. 7) is only included for completeness reasons, since the information contained in this chart is already given: energy is the product of power and time, i.e. $E = P * t$. Since the curves for power and time (expressed in number of cycles) were already presented, the qualitative behavior of the curves is obvious.

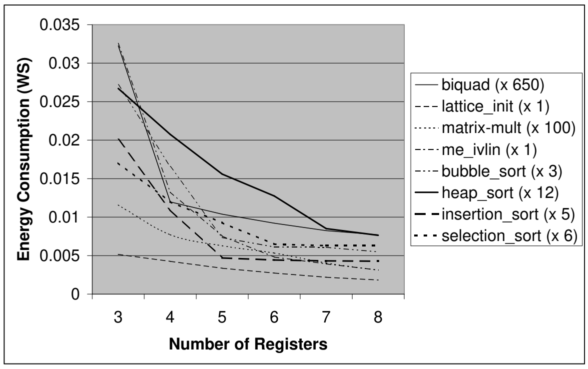

Figure 7: Energy consumption over number of registers

For memory intensive applications, one has to consider their inevitably higher energy consumption. Even an infinite number of registers would not decrease power dissipation below a certain value.

The fact that the number of registers is an important factor during the design process of an ASIP is underlined by the results given in table 2 which shows the relative energy savings possible by increasing the number of available registers one by one. The average energy savings possible amount to approximately 22%. One should bear in mind, however, that each additional register will also consume additional energy due to added capacitances and required chip area. This energy has not been considered in our approach, but will be the subject of future work.

benchmark	3 → 4	4 → 5	5 → 6	6 → 7	7 → 8
biquad	62,89%	13,35%	11,26%	10,42%	6,71%
lattice_init	17,45%	20,97%	18,92%	19,78%	16,84%
matrix-mult	33,44%	18,57%	14,99%	24,26%	23,28%
me_ivlin	59,28%	43,16%	36,29%	18,24%	20,47%
bubble_sort	39,07%	55,63%	16,78%	0,74%	9,63%
heap_sort	22,35%	24,87%	18,27%	33,24%	10,16%
insertion_sort	45,47%	57,11%	5,60%	2,63%	0,92%
selection_sort	29,68%	22,59%	30,14%	2,94%	0,00%

Table 2: *Energy savings by changing register file size*

6 Conclusion

The number of registers for the *ARM7TDMI* processor was changed within the *encc* compiler. Several benchmark applications were compiled using the customized compiler. Some application characteristics responsible for the observed results concerning number of spill instructions, number of executed instructions, cycle time required for execution as well as power consumption and energy dissipation were identified. An average energy saving of 22% was found if only one more register is inserted into the processor. Similar trends were found for the LEON architecture. The results obtained are useful during the design of ASIPs, especially at an early stage of design space exploration where one can decide how many registers a certain application will need to work within the required time and/or energy constraints for the final processor. Future work may consist of taking additional observations for different architectures, including the LEON architecture which in contrast to the ARM7TDMI features both instruction and data cache. Also, more observations will have to be taken using a compiler that supports specific features of the targeted ASIP. Further future work will be to design and implement algorithms for the automatic extraction of application characteristics to help in an early estimation for number of required registers in the ASIP design flow.

References

[1] Hoon Choi, In-Cheol Park, Seung Ho Hwang, and Chong-Min Kyung. Synthesis of Application Specific Instructions for Embedded DSP Software. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design ICCAD 98. Digest of Technical Papers*, pages 665–671, November 1998.

[2] J. Sato, M. Imai, T. Hakata, A.Y. Alomary, and N. Hikichi. An integrated Design Environment for Application Specific Integrated Processors. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors ICCD '91*, pages 414–417, October 1991.

[3] B.R. Childers and Davidson J.W. Application Specific Pipelines for Exploiting Instruction-Level Parallelism. Technical Report No. CS-98-14, University of Virginia, May 1998.

[4] M.K. Jain, M. Balakrishnan, and A. Kumar. ASIP Design Methodologies: Survey and Issues. In *Proceedings of the IEEE / ACM International Conference on VLSI Design (to appear)*, 2001.

[5] Stefan Steinke and Lars Wehmeyer. encc: An Energy Aware C Compiler. http://ls12-www.cs.uni-dortmund.de/∼steinke/LOW_POWER/encc.html, 2000.

[6] Gong J., D.D. Gajski, and A. Nicolau. Performance Evaluation for Application-Specific Architectures. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 3 Issue 4*, pages 483–490, December 1995.

[7] N.N. Binh, M. Imai, and A. Shiomi. A new HW/SW Partitioning Algorithm for Synthesizing the Highest Performance Pipelined ASIPs with Multiple Identical FUs. In *Proceedings of the Design Automation Conference, 1996, with EURO-VHDL '96 , EURO-DAC '96*, pages 126–131, September 1996.

[8] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf. The Construction of a Retargetable Simulator for an Architecture Template. In *Proceedings of the Sixth International Workshop on Hardware/Software Codesign 1998 (CODES/CASHE '98)*, pages 125–129, March 1998.

[9] J. Kin, Chunho Lee, W.H. Mangione-Smith, and M. Potkonjak. Power Efficient Mediaprocessors: Design Space Exploration. In *Proceedings of the 36th Design Automation Conference*, pages 321–326, June 1999.

[10] T.V.K. Gupta, P. Sharma, M. Balakrishnan, and S. Malik. Processor Evaluation in an Embedded Systems Design Environment. In *Proceedings of the Thirteenth International Conference on VLSI Design*, pages 98–103, January 2000.

[11] N. Ghazal, R. Newton, and Jan Rabaey. Retargetable Estimation Scheme for DSP Architecture Selection. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 485–489, January 2000.

[12] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of the ACM ISCA 2000*, June 2000.

[13] Trimaran. Trimaran Homepage. http://www.trimaran.org/index.html, 1999.

[14] Rainer Leupers. *Retargetable Code Generation for Digital Signal Processors*. Kluwer Academic Publishers, 1997.

[15] Reinhard Jöhnk and Peter Marwedel. MIMOLA Reference Manual Version 3.45. Technical Report 470, University of Dortmund, Dept. of Computer Science, Chair XII, March 1993.

[16] V. Tiwari, S. Malik, and A. Wolfe. Power Analysis Of Embedded Software: A First Step Towards Software Power Minimization. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design ICCAD*, pages 384–390, November 1994.

[17] ARM Ltd. ARM Ltd. Homepage. http://www.arm.com/, 2000.

[18] ESA. LEON-1 VHDL model. http://www.estec.esa.nl/wsmwww/leon/, 2000.

[19] Rainer Leupers. LANCE V2.0. http://ls12-www.cs.uni-dortmund.de/∼leupers/lanceV2/lanceV2.html, 2000.

[20] Christopher W. Fraser, David R. Hanson, and Todd A. Proebsting. Engineering a Simple, Efficient Code Generator Generator. Technical report, AT & T Bell Laboratories, 1992.

[21] Atmel. Atmel Corporation Homepage. http://www.atmel.com, 2000.

[22] St.S Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers, San Francisco, California, 1. edition, 1997.

[23] Andrew W. Appel. *Modern Compiler Implementation in C.* Cambridge University Press, Cambridge, New York u.a., 1998.