

Energy-Conscious HW/SW-Partitioning of Embedded Systems: A Case Study on an MPEG-2 Encoder

Jörg Henkel

C&C Research Laboratories
NEC USA, Princeton, NJ 08540
henkel@ccrl.nj.nec.com

Yanbing Li

Dept. of EE, Princeton University
Princeton, NJ 08544
yanbing@ee.princeton.edu

Abstract

Energy dissipation is a hot topic in the design of – especially mobile – embedded systems. This is because applications like digital video cameras, cellular phones etc. draw their current from batteries that spend a limited amount of energy only. In this paper we show that energy-conscious HW/SW-partitioning can lead to drastic reductions of energy dissipation of a whole embedded system. Subject of investigation is an MPEG-2 encoder. Therefore, we introduce our framework for estimating and optimizing system energy as well as all conducted design steps. The obtained results show energy savings up to 59% while the performance remains approximately the same or becomes even slightly higher. As a main result, energy-conscious HW/SW-partitioning is a promising method to be deployed in addition to classical energy and/or power reduction methods.

1 Introduction

Reducing energy dissipation is a prominent task during the design phase of embedded systems. One of the manifold reasons is that the market share of one class – the *mobile embedded systems* – is steadily increasing. A characteristic peculiarity of such a system is that it draws the current from batteries that spend a limited amount of energy only. Hence, the design goal is to minimize the energy dissipation in order to enable a longer operation period. Practical examples of mobile embedded systems are consumer products like digital video cameras, cellular phones etc. Reduction of energy dissipation increases their "mobility". These small embedded systems come often as a *systems-on-a-chip* where components like processors, caches, application specific hardware and even the main memory [1] reside on the same piece of silicon. The according design process is often referred to as *core-based system design*. In this context, a *core* can be an already optimized hardware (a *hard core* or *firm core*) like a processor core or a more abstract description (e.g. algorithm in C or VHDL) that still offers the possibility for parameterizing, optimizing and even gives the leeway to choose the form of implementation (e.g. software or hardware implementation). The latter is called a *soft core*. As a result, the designer has powerful design aids to tailor an embedded system that exactly matches the required constraints. Energy-conscious system design requires a sophisticated adaptation and tailoring of all system resources to prevent energy dissipations caused by under-utilized system parts. As one major step, tailoring includes the selection of

an appropriate HW/SW-partitioning as well as fixing of parameters like cache sizes, cache associativities, cache line sizes, main memory size etc. Core-based system design makes these design goals possible.

In this case study we investigate the power dissipation of an MPEG-2 encoder for different HW/SW-partitions and different system configurations like data cache size, instruction cache size, cache line size, cache associativity and main memory size. Therefore, we deploy our research framework in order to estimate and optimize energy dissipation for most of the system parts. The remaining system parts are treated by in-house production tools.

This paper is structured as follows: the next section summarizes the research activities for estimating and minimizing energy and/or power dissipation of various system parts and also recent approaches in co-synthesis for low energy/power. In 3 the estimation models of our framework are presented. The MPEG-2 application as well as the performed design steps and the partitioning considerations are outlined in 4 whereas the obtained results are presented and discussed in 5. Finally, section 6 gives a conclusion.

2 Related Research

Estimating and optimizing energy/power dissipation has been addressed from a software point of view¹ as well as from a hardware point of view (application specific hardware, memory hierarchy).

Tiwari and Malik [2] investigated the energy that is dissipated during the execution of programs within a processor. They found out that the energy dissipation is sensitive to the executed instruction. From these investigations they derived specific optimization techniques to minimize the software energy. They did not take into consideration the energy dissipation of the memory hierarchy and its impact on the whole system's energy dissipation. Ong and Ynn [3] have shown that the energy dissipation may drastically vary depending on the algorithms running on a processor. A power and performance simulation tool for a RISC design has been developed by Sato et al. [4]. Their tool has been used by designers to conduct architectural-level optimizations.

¹This means the power that is dissipated during the execution of a software program on a programmable processor. Since the power dissipation varies according to the executed instruction, the term *software energy* is justified. We will use this term in the following.

Further work deals with energy/power dissipation from a hardware point of view. Investigations of the major sources of power dissipation within a processor have been undertaken by Burd and Peters [5]. Gonzales and Horowitz [6] explored the power dissipation depending on the processor architecture (pipelined, un-pipelined, super-scalar).

Other system parts like caches have been analyzed by Kamble and Ghose [7]. A model for main memory power dissipation from a behavioral level point of view has been derived by Itoh et al. [20]. A method for estimating power dissipation of a synthesized hardware (ASIC) from a behavioral-level of abstraction has been proposed by Raghunathan and Dey [8]. Optimizing energy dissipation by means of high-level transformations has been addressed by Potkonjak et al. [9], for example.

As a conclusion, none of the described approaches offers a comprehensive solution i.e. an approach that takes into consideration the interdependency of *all relevant* system parts in order estimate/minimize energy/power dissipation of the *complete system*. This lack has been recognized by recent research activities in low-power co-synthesis [10, 11] even if these approaches do not reflect all system resources (no caches). Co-synthesis driven by other constraints (performance and in parts hardware effort) has been explored by [12, 13, 14, 15] among others.

The case study presented in this paper quantifies the interdependency of various system parts in terms of energy dissipation. Our framework estimates and optimizes system energy dissipation and is prominent for the conducted case study.

3 Estimation Models

A brief summary of the energy estimation models within our framework is presented in this section. More details can be retrieved from [16].

3.1 Data and Instruction Cache

The models for data and instruction caches are generic: *cache size*, *associativity* and *line size* are the parameters. Modeled are the switching capacitances of the major sources for energy dissipation (due to [7]): *word-line capacitances*, *bit-line capacitances*, *decoder capacitances* and capacitances of the *output driver*. The topology of the tag array and the decoder depends on the cache size *and* caches parameters whereas the topology of the data array can directly be derived from the cache size. Data and tag array are composed of 6-transistor SRAM cells. Our model captures the dynamic energy dissipation i.e. the energy dissipated during accesses to the caches. This is the final formula in its simplified version:

$$E_{cache} = \frac{1}{2} \cdot V_{DD}^2 \cdot (N_{acc} \cdot C_{bit,r} + N_{acc} \cdot C_{word} + k_1 \cdot C_{bit,w} + k_2 \cdot C_{dec} + k_3 \cdot C_{od}) \quad (1)$$

$C_{bit,r}$, C_{word} , $C_{bit,w}$, C_{dec} and C_{od} are the effective capacitances for a bit-line read access, a word-line access, a bit-line write access, decoder and output driver, respectively.² N_{acc} gives the number of caches accesses and is obtained by a cache simulator (see section 3.3). Note, that the capacitances are itself complex expressions composed of basic

² k_1 , k_2 and k_3 are expressions that contain in parts statistical access numbers.

capacitances³. The basic capacitances have been obtained through the tool *Cacti* [17], an analytical suite of models for a 0.8 μ m CMOS technology.

3.2 Software Energy

The model is based on the techniques described in [2] but enhanced by the impact the number of cache miss reads ($N_{miss,r}$), cache miss writes ($N_{miss,w}$) and instruction fetch misses ($N_{miss,f}$) have on the energy dissipation of the processor. It is modeled as follows:

$$E_{prg} = V_{DD} \cdot (T_{100\% hit} \cdot \sum_{i=0}^{N-1} (I_{instr,i} \cdot N_{cyc,i}) + T_{cyc} \cdot (\underbrace{N_{miss,r} \cdot N_{cyc,r-penalty} \cdot I_{instr,nop}}_{\text{data read miss penalty}} + \underbrace{N_{miss,w} \cdot N_{cyc,w-penalty} \cdot I_{instr,nop}}_{\text{data write miss penalty}} + \underbrace{N_{miss,f} \cdot N_{cyc,f-penalty} \cdot I_{instr,nop}}_{\text{instruction fetch miss penalty}})) \quad (2)$$

There it is, N the number of instructions of the program, $N_{cyc,i}$ the number of cycles instruction i resides in the processor pipeline and $I_{instr,i}$ the current that is drawn during the execution of i . Furthermore, T_{cyc} is the cycle time, $N_{cyc,r-penalty}$, $N_{cyc,w-penalty}$, and $N_{cyc,f-penalty}$ are the number of cycles that are imposed through a cache read/write/fetch penalty, respectively. The drawn current during penalty cycles is given by $I_{instr,nop}$. $T_{100\% hit}$ is the execution time of the program with an assumption of a 100% cache hit rate.⁴ In the current state this model is implemented for a 33MHz SPARCLite processor. The total program execution time is modeled by:

$$T_{prg} = T_{100\% hit} + N_{miss,r} \cdot N_{cyc,r-penalty} + N_{miss,w} \cdot N_{cyc,w-penalty} + N_{miss,f} \cdot N_{cyc,f-penalty} \quad (3)$$

3.3 Design Flow of our Framework

The input is a C program (Fig. 1) of the application⁵. The upper branch comprises a behavior simulator [23] that is attached to our software energy and performance models (Eq. 2 and 3). The branch below contains the trace tool *QPT* and the *Dinero III* [19] cache simulator who feeds software, cache and main memory energy models with the numbers of total cache accesses, data cache hits, data cache misses and instruction fetch misses. Output are the numbers for software energy dissipation and performance numbers. The feedback loop contains an optimization algorithm⁶ that adapts cache sizes, associativities and cache line sizes in order to reduce the total energy dissipation.

³Gate and drain capacitances, line capacitances etc.

⁴This is because the deployed behavior simulator primarily does not reflect the impact of caches. It assumes a 100% cache hit rate. So, the equation reflects the corrected model.

⁵i.e. only those parts that are implemented as software.

⁶The implementation of the optim. loop is described in [16].

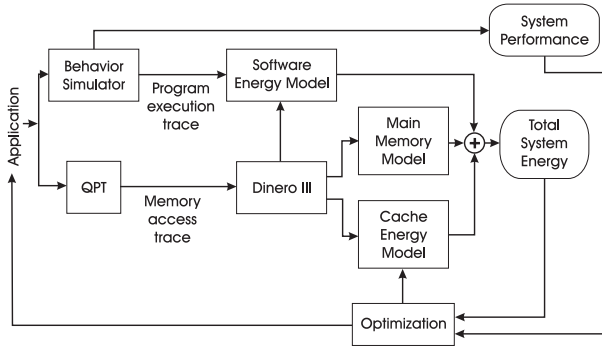


Figure 1: Design flow of our framework for estimating and optimizing system energy dissipation.

4 Approach and Design Methodology

Subject of investigation for this case study is an MPEG-2 encoder which is intended to run on the architecture shown in Fig. 2. It features a CPU, instruction and data cache, a main memory and an application specific hardware (ASIC). In the following sub-section the approach for minimizing/reducing system energy dissipation is described.

4.1 The Approach

It should be noted that decreasing power dissipation can not be achieved by reducing the performance since an MPEG-2 application is a real-time application that has to meet performance constraints. Rather than that, the principal idea to reduce energy dissipation comprises the following two steps:

A. Energy-conscious HW/SW-partitioning

A general purpose processor⁷ is not designed to handle efficiently special applications like an MPEG-2 encoder. As one example, the number of available FUs (Functional Units) is limited. As a consequence, the processor executes slower compared to a hardware with multiple FUs. During this time, other FUs might be un-utilized (e.g. multiplier in an application that features no multiplications) but still contribute to the total energy dissipation without increasing the performance.⁸ On the other side, a hardware can be designed in such a way that the utilization of each FU is much higher as opposed to the processor i.e. "wasting of energy" can be minimized through removal of low or non-utilized FUs. Even if the power dissipation of the hardware is higher, there is still a hope that the dissipated energy is smaller since the instruction-level parallelism leads to much shorter execution times and hence, to a smaller energy dissipation. This effect of energy saving is the larger the smaller the selected application is because a smaller part can be better adapted i.e. high utilization of each FU is much more likely.

For our case study that means, we have to detect small system parts that can efficiently (high utilization) be synthesized in hardware. All other system parts are still carried

⁷In the following we will use the terms "processor" when we mean "general purpose processor" and "hardware" when we mean an "application specific hardware" i.e. an "ASIC".

⁸This is correct in cases where no gated clocks are used.

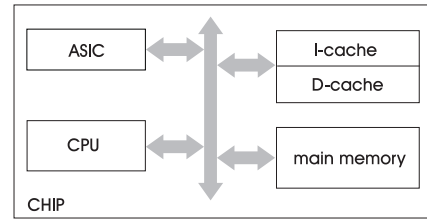


Figure 2: The target architecture.

out by the processor.

B. Optimizing the remaining system parameters

Besides the processor and the hardware, the memory hierarchy is an additional major source for energy dissipation. In case of dis-configuration it can even dominate. Due to the chosen size of data cache, instruction cache and their parameters (associativity, line size etc.) the contribution of each system part to the total energy dissipation can vary drastically and hence, it is hard to predict which parameters and sizes lead to a minimum energy dissipation of the *whole system*.

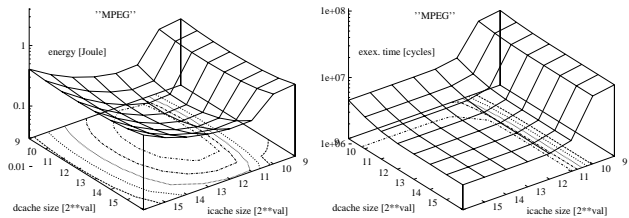


Figure 2a: Energy dissipation (left) and execution time (right) of the MPEG-2 Encoder for various data and instruction cache sizes.

As an illustration, Fig. 2a shows the energy dissipation in Joule (left) and execution time in cycles (right) of the MPEG-2 encoder for different data and instruction cache sizes⁹. This is part of the results of a pre-investigation (i.e. no hardware considered). Surprisingly, neither the largest nor the smallest caches sizes lead to a minimum energy dissipation. Additionally, not even the largest cache sizes lead to the highest performance (i.e. smallest number of cycles). What actually happens is, demonstrates Eq. 1: large caches sizes increase the switching capacitances, such that each cache access dissipates more energy. In such a case the caches dominate the total energy dissipation. Small cache sizes lead to more data and instruction caches misses. Due to Eq. 2, the software energy increases. Apparently, this is a complex optimization problem. It is addressed by our framework.

4.2 The MPEG-2 Encoder

Figure 3 shows a part of the data-flow within the MPEG-2 encoder. Compression for both, spatial and temporal (motion) reduction of redundancy is based on a DCT (discrete cosine transformation). Quantization (Q) is the process that determines what information can be discarded without a significant loss. Following the flow, Huffman coding is used

⁹Given in the figures is the value of the exponent of the cache sizes: a value of 12 for example means a cache size of $2^{12} = 2048$ bytes. The results have been obtained by our framework.

to code the entropy¹⁰. Dequantization (Q^{-1})¹¹ and inverse DCT (IDCT) are deployed as a part of generating the future frame. For more details, see [21], for example. The basis for our case study is a complete MPEG-2 software encoder ($\approx 200\text{KB}$ of C source code). The code has been explored by techniques of profiling, behavior simulation, scheduling for different resource constraints and energy investigation on the software implementation (see last section). Finally three functions have been selected for a hardware implementation: $fdct()$ as a part of the DCT, $idctcol()$ as a part of the the IDCT as well as $dist()$ as part of the quantization process (Q).

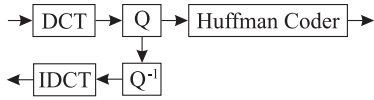


Figure 3: The interesting parts of the MPEG-2 encoder.

4.3 Design Methodology

For each of the selected co-design configurations, the following steps have been performed:

- I. The candidate hardware part is synthesized and analyzed. This incorporates the following sub-steps (see also Fig. 4):
 - a) The respective functionality is encoded in BDL and synthesized by means of the high-level synthesis tool Cyber¹².
 - b) The VHDL-RTL code output of Cyber is simulated with VSIM [22]. It delivers the number of clock cycles for execution and allows to evaluate the RTL code.
 - c) The RTL and Logic synthesis tool Varchsyn¹³ is fed with the VHDL code. Technology mapping is done by means of NEC’s CMOS6 library. Output of Varchsyn is a netlist in PWC format as well as information about the total number of cells and the maximum possible clock frequency.
 - d) Gate-level simulation is performed by CSIM¹⁴. One operation mode of CSIM delivers the energy dissipation based on the switching activities. For that step, stimuli vectors have to be provided.
- II. Our framework optimizes the data cache size, the instruction cache size and the according associativities and line sizes. It provides furthermore detailed information about the energy dissipation of the software, the main memory and the caches. Also, the performance of the software part is provided.
- III. Results of the previous steps are energy and performance data of the whole HW/SW-system as well as system configuration parameters that ensure low energy dissipation.

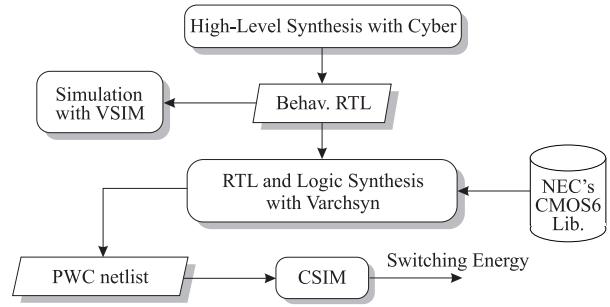


Figure 4: Hardware synthesis steps

case	Energy [mJ]					
	i-cache	d-cache	mem	sw	hw	total
0	44.79	17.98	2.305	74.32	n/a	140.92
1	40.54	14.84	2.464	49.33	14.09	121.26
2	35.36	13.14	2.944	27.16	0.709	79.31
3	39.43	15.93	2.520	49.51	0.156	107.55
4	14.59	4.127	1.372	16.99	22.27	59.35

Table 1: Contribution of all system resources to total energy dissipation.

5 Results

We distinguish five different cases in the following discussion:

case 0: the all-software solution

case 1: $dist()$ is synthesized in hardware, rest is software

case 2: $fdct()$ in hardware, rest is software

case 3: $dctcol()$ in hardware, rest is software

case 4: $dist()$ and $fdct()$ in hardware, rest is software

Our primary goal is to reduce the total energy dissipation rather than the power dissipation (see section 1). In practice that means, even an (ASIC-) hardware featuring an average power dissipation close to, for example, the processors power dissipation, might be an appropriate candidate for reducing system energy in case the throughput is high due to high utilization of internal resources (FUs). Furthermore, we assume that the clock of the whole processor and hardware is gated. So, whenever the processor is performing and the hardware is idle (they perform mutual exclusive), no energy will be dissipated by the hardware. In the complementary case, the processor behaves accordingly. For practical reasons (keeping simulation times of VHDL simulation and behavioral simulation reasonable), all obtained numbers refer to a sequence of six frames running through the MPEG-2 encoder. Each frame has a size of 64×64 pixels. This configuration is appropriate to ensure that all parts of the encoder work almost like in the steady-state.

The current energy model of our framework is valid for a SPARCLite processor running at 33MHz. The hardware is working at 15MHz (delivered by logic synthesis through use of NEC’s CMOS6 library) The memory parameters have been optimized for low energy dissipation by use of our framework: instruction cache size and data cache size are 2KB each, the associativity is 2 and the line size is 4 for both caches. The main memory size is 128KB. The main results are illustrated in Tab. 1: Each of the selected co-designs of cases 1 to 4 yields a energy reduction (up to 59%, see also Fig. 5) compared to the reference case 0. The average

¹⁰Entropy is a measure for randomness and disorder.

¹¹The inverse of quantification.

¹²Cyber is NEC’s in-house high-level synthesis tool. BDL is the HDL used in Cyber.

¹³Varchsyn is NEC’s in-house RTL and Logic synthesis tool

¹⁴NEC’s in-house gate-level simulator.

cs	Software		Hardware (ASIC)			
	time [s]	# cyps (33MHz)	time [ms]	# cyps (15MHz)	# cells	av. Pw. [mW]
0	0.155	5,167,958	n/a	n/a	n/a	n/a
1	0.110	3,682,978	87.27	1,309,068	43,364	161.48
2	0.051	1,697,951	23.82	357,408	8,600	29.79
3	0.111	3,693,138	3.31	49,632	18,412	47.07
4	0.033	1,093,911	110.09	n/a	51,298	202.28

Table 2: Execution time of SW and HW. Last two columns: hardware effort and average power dissipation of synthesized hardware.

power dissipation is shown in Tab. 2, last column. According to Tab. 1, a larger reduction of the energy dissipation through an even larger hardware is hardly possible since a great amount of energy is dissipated within the memory hierarchy (e.g. 47.7 % in case 1).

Of course, there is a simple way to reduce energy dissipation by simply reducing the performance. But our aim was to keep the performance approximately the same while reducing the energy. These results are shown in Tab. 2, columns 2 and 4 (time) and columns 3 and 5 (cycles). The results in terms of energy reduction and performance change (compared to case 0) are shown in Fig. 5. Only in case 1 a performance decrease has occurred. In all other cases, the performance is even moderately higher.

The decrease in energy dissipation is due to the high operator parallelism within the hardware as well as to the high degree of utilization of each FU. To achieve these results, the system parts to be implemented in hardware, have to be selected thoroughly. In case, the presupposition described in 4.1 are not fulfilled, a hardware will even increase the energy dissipation.

Other restrictions are the hardware costs of an ASIC hardware. In our case the largest ASIC comprised a number 51k cells, the smallest less than 9k cells (Tab.2). Furthermore, it should be noted that the partitioning was done from a designers point of view – the functional level. Investigation of the schedule revealed that in parts of the synthesized functions the utilization rate of HW resources was much smaller than the average utilization within the same function. Apparently, an even better reduction of energy dissipation would have been achieved through a finer-grain partitioning.

6 Conclusion

We have presented a case study for energy-conscious HW/SW-partitioning. Based on the idea that an application specific hardware can be much more energy efficient due to higher resource utilization, energy savings up to 59% have been achieved compared to a pure software solution. Furthermore, it has been shown that estimating energy dissipation of a whole system is not a trivial task since the sources of energy dissipation are manifold and depend on each other. This requires tools that can capture these interdependencies. Therefore, we presented our framework for estimating and optimizing system energy dissipation.

7 Acknowledgment

We would like to thank Bo Tao from Princeton University who provided us with his version of the MPEG-2 encoder source code.

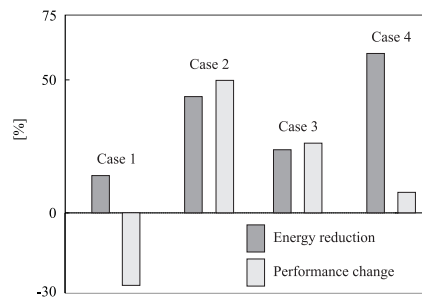


Figure 5: Energy reduction and performance change in %.

References

- [1] Y. Nunomura, T. Shimizu, O. Tomisawa, *M32/D-Integrating DRAM and Microprocessor*, IEEE Micro Magazine, Vol.17, No.6, pp.40-48, 1997.
- [2] V. Tiwari, S. Malik, A. Wolfe, *Instruction Level Power Analysis and Optimization of Software*, Kluwer Academic Publishers, Journal of VLSI Signal Processing, pp. 1-18, 1996.
- [3] P.-W. Ong, R.-H. Ynn, *Power-Conscious Software Design – a framework for modeling software on hardware*, IEEE Proc. of Symposium on Low Power Electronics, pp. 36-37, 1994.
- [4] T. Sato, M. Nagamatsu, H. Tago, *Power and Performance Simulator: ESP and its Application for 100 MIPS/W Class RISC Design*, IEEE Proc. of Symposium on Low Power Electronics, pp. 46-47, 1994.
- [5] T. Burd, B. Peters, *A Power Analysis of a Microprocessor: A Study of an Implementation of the MIPS R3000 Architecture*, Technical Report, University of California at Berkeley, May 1994.
- [6] R. Gonzales, M. Horowitz, *Energy Dissipation in General Purpose Processors*, IEEE Proc. of Symposium on Low Power Electronics, pp. 12-13, 1995.
- [7] M.B. Kamble, K. Ghose, *Analytical Energy Dissipation Models For Low Power Caches*, IEEE Proc. of Symposium on Low Power Electronics and Design, pp. 143-148, 1997.
- [8] A. Raghunathan, S. Dey, N. Jha, *estimation Register-Transfer Level Estimation Techniques for Switching Activity and Power Consumption*, IEEE Proc. of Int. Conf. on CAD (ICCAD96), pp.158-165, 1996.
- [9] I. Hong, D. Kirovski, M. Potkonjak, *Potential-Driven Statistical Ordering of Transformations*, IEEE Proc. of 34th. Design Automation Conference (DAC97), pp.347-352, 1997.
- [10] D. Kirovski, M. Potkonjak, *System-Level Synthesis of Low-Power Hard Real-Time Systems*, IEEE Proc. of 34th. Design Automation Conference (DAC97), pp.697-702, 1997.
- [11] B.P. Dave, G. Lakshminarayana, N.K. Jha, *COSYN: Hardware-Software Co-Synthesis of Embedded Systems*, IEEE Proc. of 34th. Design Automation Conference (DAC97), pp.703-708, 1997.
- [12] R.K. Gupta, G.D. Micheli, *System-level Synthesis using Re-programmable Components*, IEEE/ACM Proc. of EDAC'92, pp. 2-7, 1992.
- [13] F. Vahid, D.D. Gajski, J. Gong, *A Binary-Constraint Search Algorithm for Minimizing Hardware during Hardware/Software Partitioning*, IEEE/ACM Proc. of EuroDAC'94, pp. 214-219, 1994.
- [14] P.V. Knudsen, J. Madsen, *PACE: A Dynamic Programming Algorithm for Hardware/Software Partitioning*, IEEE Proc. of Codes/CASHE'96, pp.85-92, 1996.
- [15] J. Henkel, R. Ernst, *A Hardware/Software Partitioner using a dynamically determined Granularity*, IEEE Proc. of 34th. Design Automation Conference (DAC97), pp.691-696, 1997.
- [16] J. Henkel, Y. Li, *An Approach for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems*, NEC USA Inc., Technical Report # 97-C071-4-5110-2, Oct.1997.
- [17] S.J.E Wilton, N.P. Jouppi, *An Enhanced Access and Cycle Time Model for On-Chip Caches*, DEC, WRL Research Report 93/5, July 1994.
- [18] D.L. Weaver, T. Germond, *The SPARC Architecture Manual, Version 9*, PTR Prentice Hall, 1994.
- [19] M. D. Hill, J. R. Laurus, A. R. Lebeck et al., *WARTS: Wisconsin Architectural Research Tool Set*, Computer Science Department University of Wisconsin.
- [20] K. Itoh, K. Sasaki and Y. Nakagome, *Trends in Low-Power RAM Circuit Technologies*, Proceedings of the IEEE, VOL. 83, No. 4, April 1995.
- [21] P.K. Andleigh, K. Thakrar, *Multimedia Systems Design*, Prentice Hall, 1996.
- [22] *V-System/Workstation User's Manual, VHDL Simulation for Workstations*, Model Technology, 1995.
- [23] R. Ernst, W. Ye, *Embedded program timing analysis based on path clustering and architectural classification*, IEEE Proc. of Int. Conf. on CAD (ICCAD97), pp.598-604, 1997.