Customization of a CISC Processor Core for Low-Power Applications

You-Sung Chang, Bong-Il Park, In-Cheol Park, and Chong-Min Kyung
Dept. of EE, KAIST, Taejon, Korea
caviar@maestro.kaist.ac.kr, bipark@duo.kaist.ac.kr, {icpark,kyung}@ee.kaist.ac.kr

Abstract

This paper describes a core-customization process of a CISC processor core for a given application program. It aims at the power reduction in the CISC processor core by fully utilizing the microcode-based control scheme, that is one of the most characterizing features of a CISC processor. The optimization process includes two key techniques, generation of application-specific complex instructions (ASCI) and low-power-oriented microcode-ROM compilation, which independently operate at the two different levels of optimization. As a means of architectural level of optimization, application-specific complex instructions are generated so as to reduce the activities of fetch and decode units, and in the point of physical level of optimization, the microcode-ROM is compiled to reduce the bitline toggling for each microcode-ROM access. Our experimental results based on transistor-level simulation show the proposed techniques can jointly reduce the total power consumption of the CISC processor core by up to 41%.

1. Introduction

Lowering power consumption has been one of the major issues in the design of embedded processor core, due to the ever-increasing demand of mobile applications. However, most commercial processor cores are just straightforward low-power versions of the originals, which work in lower voltage and support some system-level power management modes. Those aim at a convenient and expeditious software migration for a variety of ASIC applications, but do not give any chance of possible core optimization for a given application program. On the other hand, recent works on application-specific processors include fully application-driven architecture that is often too aggressive or academic to be realistic[1, 13] and focus on speed and area optimization of the system[2, 3].

In system-level, it is reported that code compression can be used to decrease power consumption of the external memory[4]. Given an application program, it extracts a set of instructions that is used in the application program, then it compresses the application program by substituting each instruction with the corresponding set-index of the instruction. However, the code compression has no relation to power reduction of the processor core itself. Moreover, the overhead of the additional code decompressor and address converter is often too large to accept. The power consumption of transform table of instruction decompressor is never negligible and the latency of instruction fetch also increases.

This paper describes a core-customization process of a CISC processor core with a given low-power application program. It fully utilizes the high flexibility of microcode-based control schemes for power reduction and pursues the maximal power reduction under the limitation of a fixed architecture without loosing the merit of the fast software migration.

The remainder of the paper is organized as follows. Section 2 briefly introduces a CISC processor core, that is the object of the core-customization process, and identifies the tackling point for the power optimization with the help of power spectrum of the processor core. Section 3 describes the overview of the core-customization process and focuses the stages that are developed for low-power optimization. Then, Section 4 details the main techniques used in the stages. Section 5 presents the simulation results and discusses about the results. Finally, we conclude in Section 6.

2. Architecture and Power Spectrum of the Target CISC Processor Core

The object of the power optimization is a CISC processor core that is designed based on the 80386 microprocessor architecture[14, 15]. The choice of x86 as an embedded processor architecture is very reasonable when it is assumed that an embedded processor core of some generalized functionality is needed. The x86 processor architecture enables immediate access to the huge software base that is already available. Moreover, in the view point of power consumption, which is the major concern of embedded processor design, CISC-type core is superior to RISC-type core, because CISC-type core has typically less memory references than RISC-type core.

The target CISC processor core has the IA32 instruction set and keeps the binary compatibility with the x86 family processors in a somewhat simplified manner. As shown in Fig. 1, the processor core only incorporates prefetch, decode, micro-control, execution and segmentation, i.e., memory management unit. In the design, the memory management is restricted to segmentation without paging, because a two-level address translation mechanism is not required for most applications for embedded processor cores.

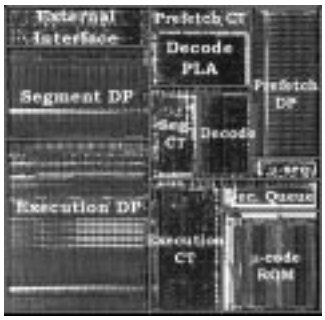


Figure 1. Micro-photograph of the CISC processor core, the object of the core-customization process; It consists of 0.24 million transistors and occupies 4.4mm x 4.4mm die area when fabricated in a 0.6μm, TLM CMOS process.

In addition, most protection checks are omitted for speed-up and simplicity of the design, and provided only for software debugging in the simulation stage. Only the increment in the complexity is that the processor core has controllable clock branches for power saving control.

To get an insight for the power optimization of the processor core, it is needed to analyze the power spectrum of the processor core. Fig. 2 shows the percentage of estimated power consumption for each section in the processor core during OS booting with the well-known operating system, MS-DOS 6.0.

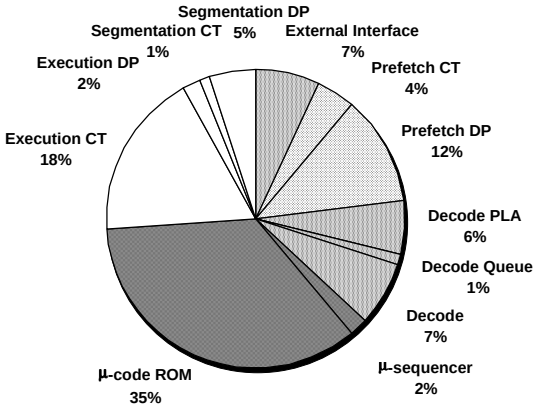


Figure 2. Simulated power dissipation on each section of the the processor core that is to be customized for power

In Fig. 2, it needs to be noted that quite a large portion of total power is dissipated by the external interface, prefetch and decode sections that have no direct relation to data processing. Approximately 37% of the total power is dissipated in these sections. In Fig. 2, microcode-ROM appears as the most power consuming section. It alone dissipates about 35% of the total power. The high power consumption of the microcode-ROM in CISC processors is resulted from the frequent access of every cycle as is the code cache in RISC processors[9].

In this paper, we use transistor-level simulation to estimate and evaluate the power consumption inside the processor core. Actually, two circuit simulators are used for the simulation; 'IRSIM'[1] for most blocks and 'HSPICE' for some sub-blocks for which 'IRSIM' is unusable. As input files for simulation, circuit descriptions are extracted from the physical layouts of the functionally divided sections inside the processor core. For a given application program, stimulus vectors are gathered through functional simulation of the processor core model with the system model. The signal dump files generated by 'VERILOG' simulation are translated into the 'IRSIM' command files as stimulus vectors through an in-house conversion tool. The power estimation is done in the manner of divide-and-conquer to speed up the estimation.

3. Overview of the Core-Customization Process

Based on the observation of the power spectrum, the core-customization process focuses on the power reduction in fetch, decode and microcode-ROM sections. The core-customization process consists of two key techniques; one is a use of application-specific complex instruction(ASCI) for power reduction in fetch and decode sections, and the other is a novel compilation technique for microcode-ROM to have reduced power consumption.

Fig. 3 shows the typical system development flow using the CISC processor core. Given an application specification, once the parts to be integrated into a single chip are determined, application software for the processor core and on-chip peripherals are developed in parallel with the system board design. After the software development, core-customization begins with the sequence of program profiling, analysis, and instruction set redefinition.

To reflect the change in the instruction set, microcode-ROM and decode-PLA is recompiled. A new entry of instruction set is registered in the form of a microcode sequence. The microcode sequences of instructions included in the newly defined instruction set are gathered and translated into microcode-ROM bit patterns for the microcode-ROM generation. Informations of new instruction entries such as operand size and function codes are also registered in the content of decode table. Decode-PLA is generated based on the information of the decode table and the macro instruction start address table. The macro instruction start address table is obtained by microcode-ROM compilation and contains the start addresses of all instructions included in microcode-ROM.

After all the process, a customized ASIC is completed by integrating the microcode-ROM, decode-PLA and selected functional units or peripherals to the processor core.

[1] an event-driven simulator for MOS transistor-level circuits.

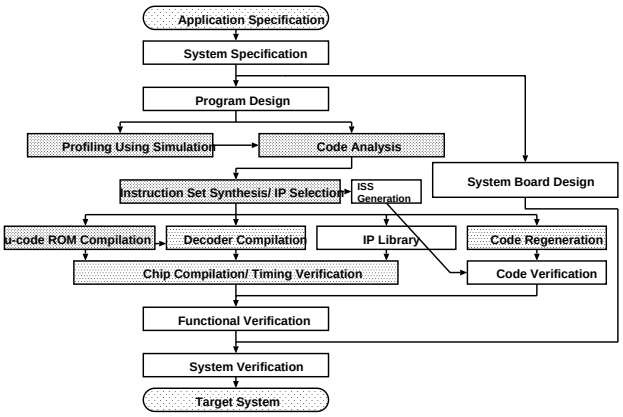


Figure 3. System development flow using the CISC processor core

Software development and core-customization can be completed in parallel if key-code analysis in algorithm level is possible before the complete version of the program.

For the conventional purpose of speed and area optimization, program profiling and analysis is done for necessary IP selections or functional unit definitions. Generally, the newly introduced functional units are attached externally through I/O ports. Those can be attached directly to the internal data and control bus to satisfy strict timing constraints, and require redefinition of the instruction set.

In contrast to that, the present concern is the power optimization. The proposed core-customization process keeps the same development flow, and just two parts are improved to include a new objective for power optimization. The instruction set redefinition stage is utilized to generate ASCIs for the purpose of power reduction, and microcode-ROM compiler is replaced with one of the enhanced compilation techniques.

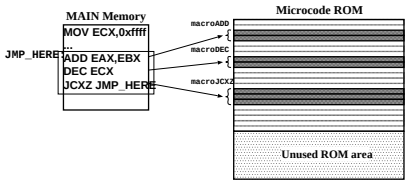
4. Customization Techniques for Low Power

4.1. Generation of ASCIs for Low Power Optimization

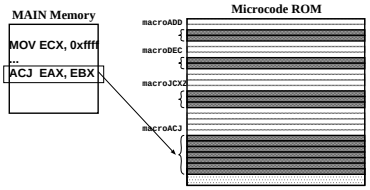
Core-customization process of the embedded processor core for low-power includes a generation of ASCI given an application program. ASCI can be used with various purposes in embedded processors. It can provide a control and/or communication channel to application-specific functional blocks. For the purpose of code size reduction, a repetitive routine or loop can be translated into a single ASCI. It also can be a complex branch instruction to remove the branch penalty in a timing critical routine.

In the use of ASCI, we notice that ASCI has a great potential for power reduction by removing redundant activities in the external interface, prefetch and decode sections. The translation of a sequence of n simple instructions in a program block into a single ASCI removes $(n-1)$ fetch and decode operations for each execution of the block. The massive iteration for data processing, that can be found in most applications for the embedded processors, guarantees a large amount of power reduction in the core-customization using ASCI.

Fig. 4 shows an example emphasizing the potentiality of the core-customization process. In the example, the repeated 196604 fetch-and-decode operations is removed by translating the instruction sequence of $ADD, DEC, JCXZ$ into a single ASCI, ACJ. The behavior of a new ASCI gathered from the microcodes of the primitive instructions with operand embedding, where necessary immediate and displacement values are stored in constant ROM for the operand embedding. The remaining microcode-ROM space is used to store the microcode sequence of the newly introduced ASCI as shown in Fig. 4-(b).



(a) Code flow with conventional instructions



(b) Code flow with an ASCI, ACJ

Figure 4. Core-customization by defining ASCIs; Fetch-and-decodes of $[0xffff]_{HEX}$ times the instruction sequence of $\{ADD, DEC, JCXZ\}$ are substituted by a single fetch-and-decode of an ASCI, ACJ

The benefits obtained by generating the ASCIs for low-power is amplified by incorporating it with the microcode-controlled clock-enabling scheme. Fig 5 shows the clock tree and controllable branches of the target processor core. Clock branches to the prefetch and decode sections are controlled by the microcode programming. A separate microcode field is used not to invoke performance loss by field conflict with other functionalities. To get a performance enhancement, inherent complex operations and ASCIs containing a large size of microcode sequence are encapsulated by a starting tag of disabling the branches and a ending tag of enabling the branches.

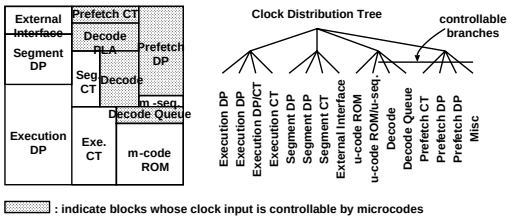


Figure 5. Clock branches that is controllable by microcodes

Fig. 6 presents a instruction flow in the pipeline when it used with the controllable clock branches. In the example, it is assumed that all instructions have single-cycle execution. Where the instruction i is an ASCI containing three primitive instructions. At the first execution cycle, it disables the clock branches for fetch and decode sections and at the third execution cycle, it enables the clock branches. During the interval with the disabled clock, fetch and decode sections has no activities.

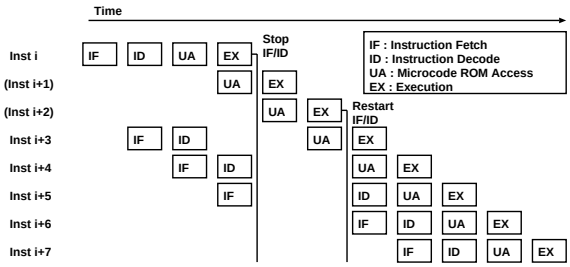


Figure 6. An instruction flow in pipeline of the processor core with microcode-controlled clock branches for fetch and decode sections.

As shown in the previous examples, most of the fetch and decode operations can be eventually removed by adequate block selections and their translations into ASCIs. The remaining problem is how to select the block. The block selection problem can be formulated in a similar form of 0/1 Knapsack Problem as follows;

$$\max \sum \alpha_i \cdot (P_i - 1) \cdot x_i \;\; s.t. \sum w_i \cdot x_i \leq M \qquad (1)$$

where i is a candidate instruction, x_i has '1' or '0' to indicates whether an instruction i is selected or not, P_i is the number of primitive instructions included in the instruction i, and α_i is the used number of an instruction i to complete a cover for the given application trace for which the processor core is optimized. w_i presents the microcode length of the instruction candidate i, and M is the total capacity of the microcode-ROM.

In the problem, the space of instruction candidates is $2^n - 1$ where n is the number of primitive instructions. Moreover, the problem is extremely complicated by the possible variation of α_i according to the use of instructions to make a cover.

To solve the problem, we develop a more simplified heuristic algorithm. In the heuristic algorithm, the candidate instructions are limited to the 'basic blocks' in the application program, where the 'basic blocks' are defined as the blocks that have no entering and exiting points in the middle of the blocks. Then, it start with a set of primitive instructions required to cover the given application trace. This assumption removes the condition for the trace cover. By these simplifications, the problem is changed to a pure 0/1 knapsack problem. Although the 0/1 knapsack is also NP-complete problem, it can be solved effectively by dynamic programming. Among the solution of the knapsack problem, the basic blocks that are always appears in a combined form are merged into a super block. After all, the super blocks and remaining basic blocks are translated into ASCIs.

4.2. Microcode ROM Compile using Strip Inversion

A single memory access can be divided into two separate sub-operations, wordline driving by address decoders, and bitline driving by storage elements. The bitline driving works with the assistance of sense amplifiers. For the both sub-operations, most power is dissipated in the large memory array due to the high capacitance of bitlines and wordlines inside the array. Moreover, the power consumed inside memory is dominated by the power burned to drive bitlines[7]. Typically, tens of bitlines are driven for a single wordline activation, and the capacitance of a bitline is a few times larger than the capacitance of a wordline. In the microcode-ROM of the processor core, the power consumption by wordline driving is just 3.43% of that of bitline. To reduce the power consumed by microcode-ROM, we try to generate a microcode-ROM such that it has a reduced bitline toggling.

To get high access speed and small area, mask-ROMs use the precharging scheme. Every cycle, it precharges each bitline with a reference voltage. Once a wordline is driven, then each bitline is evaluated with the value that is stored in the addressed element. Here, the bitline evaluation with the same value as the precharging value does not make discharging of bitlines during the memory access. The precharged value on the bitline remains until the next precharging operation. Consequently, no dynamic power dissipation exists for the bit readings which has the same value as the precharging value.

The simple observation brings a great chance to reduce the power reduction in the microcode-ROM. Given an application program, the microcode-ROM can be compiled to have a reduced power consumption by selective inverting the microcode bit patterns block by block. However, a block of arbitrary shape can not be used for inversion, because it has a difficulty in restoring the original data. In the actual process, the inversion was taken strip by strip vertically, therefore the original microcode readings are easily restored by attaching inversion buffers for the inverted column instead of normal buffers. The behavior of the microcode-ROM appears the same in the outside.

The inversion of a vertical strip is decided based on the statistics gathered running a given application. Program profiling with rare core processor reports the ratio of bit readings of '1's and '0's for each vertical strip. If a vertical strip has more readings of '0's that are different from the precharging value '1' and require discharging for evaluation, the vertical strip is inverted in the microcode-ROM compilation. Fig. 7 gives an example of the vertical strip inversion. For easy explanation, we assume the uniform probability for wordline activation. In the case, the ratio of bit readings of '1's and '0's is the same as the ratio of the number of bit values directly counted from the ROM bit patterns. The vertical strips in which the ratio of the discharging bits is over 50%, are inverted as chosen in the Fig. 7.

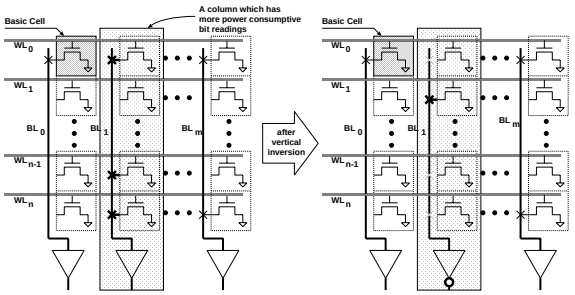


Figure 7. An illustration of vertical strip inversion

For quantitative illustration of the strip inversion, the power dissipation in the bitline arrays can be represented as Eq. 2.

$$P = \sum_i C_i \cdot V^2 \cdot f \cdot (\frac{N_a}{N})_i \quad (2)$$

where i means a i-th bitline in the microcode-ROM array, C_i is the capacitance of the i-th bitline, V is the voltage swing of the bitline, f is the frequency and $(\frac{N_a}{N})_i$ is the activity factor of the i-th bitline. With the fixed V and f, the customization of microcode-ROM changes $(\frac{N_a}{N})_i$ and C_i. The activity factor $(\frac{N_a}{N})_i$ represents the ratio of the number of the discharging bit readings over the number of the conserving bit readings. If the i-th strip is inverted, the activity factor $(\frac{N_a}{N})_i$ is substituted with $1 - (\frac{N_a}{N})_i$ leading to the power reduction of $C_i V^2 f(2(\frac{N_a}{N})_i - 1)$ in the i-th bitline. In addition, C_i is decreased for most cases because the inversion tends to result in less number of the drain junctions as is in the Fig.7.

Fig. 8 shows an implementation of microcode-ROM using the vertical strip inversion. In the real implementation, a microcode-ROM is split into multiple banks and separately optimized bank by bank.

5. Simulation Results and Discussion

Table 1 shows the estimated power reduction obtained by the use of ASCIs for power optimization. The microcode-ROM has 1024 lines in the processor core. In Table 1, the total power reductions are spread from 5.8% up to 33.6%.

The result is very promising because the proposed scheme has a great potential to decrease the system power

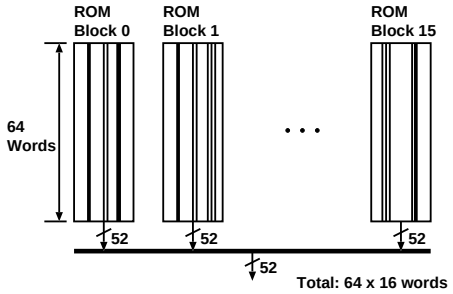


Figure 8. Microcode ROM configuration in the processor core; black vertical lines inside each block indicate inverted vertical strips.

in addition to the power of the processor core. Normally, the external bus traffics can be decreased by the reduction of the code fetches. If a code cache exists, cache hit ratio is also increased due to the reduced code size. Consequently, quite a portion of the power consuming bus traffics can be eliminated.

Table 1. Simulated power reduction caused by the introduction of ASCIs; the percentages present the reduction of the total power dissipation in the CISC processor core

Programs	Power Reduction (%)
DOS 6.0	5.8
JPEG	17.9
FFT	33.6
PID Control	27.2

Table 2 shows the power reduction in microcode-ROM when the proposed microcode-ROM compilation technique is used. The amount of power reduction seems to be somewhat bound in a limited range. Especially the power reductions for the JPEG, FFT, and PID Control applications are nearly the same. This phenomenon is resulted from the nearly same decisions for inversions across the three applications because of the compiler's limited use of instructions.

Anyway, in spite of the simplicity, the effect of the microcode-ROM compilation is large enough to be an effective means for reducing the power of the processor core. Moreover, we think that the strip inversion scheme can be generalized for power optimization any kind of mask-ROMs.

In the core-customization process, power reduction obtained by the two optimization techniques are orthogonal. The object sections for the power reduction are also different. Therefore, the total power reduction is acquired by simple addition of the amount of each reduction.

The proposed core-customization process can be easily adapted for any other processor core. For most CISC processor cores, in which the fetch, decode and microcode-

Table 2. Simulated power reduction caused by the vertical strip inversion schemes for microcode-ROM; the percentages present the reduction of the total power dissipation in the CISC processor core

Programs	Power Reduction (%)
DOS 6.0	10.5
JPEG	7.6
FFT	7.7
PID Control	7.4

ROM sections appears the most power consuming, the core-optimization can be effectively applied.

6. Conclusion

In this paper, we proposed a customization process of a CISC processor core for low-power applications. The core-customization process takes the advantage of the microcode-based control scheme of a CISC processor. The process concentrates on the power reduction in fetch, decode and microcode-ROM units, that are identified as the most power consuming blocks in the power analysis of the processor core. To reduce the power consumption in the fetch and the decode units, the generation of ASCIs has been introduced in the core-customization process. It removes repetitive fetch and decode operations by changing a looping or iterative basic block into a single ASCI, and is coupled with the microcode-controlled clock-enabling scheme to maximize the benefit. In addition, a novel ROM compiling technique has been developed to reduce the power consumption in the microcode-ROM, It induces the power reduction by decreasing the number of bitline toggling for each microcode-ROM access.

Simulation results show that the proposed schemes altogether can reduce the total power consumption of the target processor core by up to 41%. It's an outstanding result considering little modification on the processor core. In addition to the power reduction inside the processor core, it has more potential to decrease the power in the system-level. The missing instruction fetches by the inclusion of ASCIs reduce memory access, bus traffic and eventually power consumption of the external memory and I/O bus.

References

[1] Joseph A. Fisher, Paolo Faraboschi and Guiseppe Desoli, "Custom-Fit Processors: Letting Applications Define Architectures", *Proceedings of the 29th Annual International Symposium on Microarchitecture*, December 1996, pp. 324-335.

[2] C. Alba, L. Carro, A. Lima and A. Suzim, "Embedded Systems Design with Frontend Compilers", *Proceedings of the 1996 International Conference on Computer Design*, October 1996, pp. 200-205.

[3] Steve T. Fu, Daniel F. Zucker, and Michael J. Flynn, "Memory Hierarchy Synthesis of a Multimedia Embedded Processor", *Proceedings of the 1996 International Conference on Computer Design*, October 1996, pp. 176-184.

[4] Y. Yoshida, B. Song, and H. Okuhata, "An Object Code Compression Approach to Embedded Processors", *Proceedings of the 1997 International Symposium on Low Power Electronics and Design*, August 1997, pp. 265-268.

[5] S. Shigematsu, S. Mutoh, and Y. Matsuya, "Power Management Technique for 1-V LSIs using Embedded Processor", *Proceedings of the 1996 IEEE Custom Integrated Circuits Conference*, May 1996, pp. 111-114.

[6] A. P. Chandrakasan, A. Burstein, and R. W. Brodersen, "A Low-Power Chipset for a Portable Multimedia I/O Terminal", *IEEE Journal of Solid-State Circuits*, December 1994, pp. 1415-1428.

[7] B. S. Amrutur and M. Horowitz, "Techniques To Reduce Power In Fast Wide Memories", *Proceedings of the 1994 International Symposium on Low Power Electronics*, August 1994, pp. 92-93.

[8] M. Ukita, S. Murakami, T. Yamagata, H. Kuriyama, Y. Nishimura, and K. Anami, "A Single Bitline Cross-Point Cell Activation Architecture for Ultra Low Power SRAMS", *International Solid State Circuits Conference*, February 1994, pp. 252-253.

[9] J. Montanaro, R. T. Witeck, K. Anne, A. J. Black, E. M. Cooper, D. W. Dobberpuhl, P. M. donahue, J. Eno, G. W. Hoeppner, D. Kruckemyer, T. H. Lee, P. C. M. Lin, L. Madden, D. Murray, M. H. Pearce, S. Santhanam, K. J. Snyder, R. Stephany, and T. Stephen C, "A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor", *IEEE Journal of Solid-State Circuits*, November 1996, pp. 1703-1714.

[10] H. Morimura and N. Shibata, "A 1-V 1-Mb SRAM for Portable Equipment", *Proceedings of the 1996 International Symposium on Low Power Electronics and Design*, August 1996, pp. 61-66.

[11] A. Cao, et al, "CAD Methodology for the Design of UltraSPARC ^{TM}-I Microprocessor at Sun Microsystems Inc.", *Proceedings of the 32nd Design Automation Conference*, June 1995, pp. 19-22.

[12] D. Herrmann, E. Mass, M. Trawny,R. Ernst, P. Ruffer, M. Seitz, and S. Hasenzahl, "High Speed Video Board as a Case Study for Hardware-Software Co-Design", *Proceedings of the 1996 International Conference on Computer Design*, October 1996, pp. 185-190.

[13] Elliot Waingold, et al, "Baring It All to Software: Raw Machines", *IEEE COMPUTER*, September 1997, pp. 86-93.

[14] Intel Corporation, *Microprocessor II*, Intel Literature Co, 1993.

[15] Intel Corporation, *80386 Programmer's Reference Manual*, Intel Literature Co, 1986.