A System Level Memory Power Optimization Technique Using Multiple Supply and Threshold Voltages

Tohru Ishihara Kunihiro Asada

VLSI Design and Education Center, The University of Tokyo
7–3–1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
E-mail: {ishihara,asada}@vdec.u-tokyo.ac.jp

Abstract— A system level approach for a memory power reduction is proposed in this paper. The basic idea is allocating frequently executed object codes into a small subprogram memory and optimizing supply voltage and threshold voltage of the subprogram memory. Since large scale memory contains a lot of direct paths from power supply to ground, power dissipation caused by subthreshold leakage current is more serious than dynamic power dissipation. Our approach optimizes the size of subprogram memory, supply voltage, and threshold voltage so as to minimize memory power dissipation including static power dissipation caused by leakage current. A heuristic algorithm which determines code allocation, supply voltage, and threshold voltage simultaneously so as to minimize power dissipation of memories is proposed as well. Our experiments with some benchmark programs demonstrate significant energy reductions up to 80% over a program memory which does not employ our approach.

1 Introduction

An important class of digital systems includes applications, such as video image processing and speech recognition, which are extremely memory-intensive. In such systems, a much power is consumed by memory accesses. In most of today's microprocessors, an instruction memory including a cache memory is one of the main power consumers. The on-chip caches of the 21164 DEC Alpha chip dissipates 25% of the total power of the processor. The StrongARM SA-110 processor from DEC, which specifically targets low power applications, dissipate about 27% of the power in the instruction cache[1]. Thus, employing low-power memory can greatly reduce the overall power consumption in digital systems. The most effective way to reduce the power dissipation in digital systems is to lower the supply voltage. However, lowering the supply voltage causes an increase of access delay to the memory. Although lowering threshold voltage is effective to improve access time to the memory, this can not be applied to a large scale memory. Because, lowering threshold voltage causes explosive increase of subthreshold leakage current. Especially for a large scale memory, an increase of the subthreshold current is critical, because it has a lot of leakage paths from power supply to ground. There has been proposed some techniques which cut off leakage current of logic circuits with MT-CMOS (Multiple Threshold CMOS) when systems are inactive[2]. However, these techniques are not applicable for memory, because they can't maintain the data in memory when the power source is cut off. Optimization techniques considering both static and dynamic power dissipation is required.

Especially for embedded systems, low power oriented applications also require design flexibility, which results in the need for implementation by using preoptimized cores. Current semiconductor technology allows the integration of processor cores and memory modules on a single die, which enables the implementation of a system on a single chip. Consequently, the design productivity along with the traditional synthesis process has not followed the exponential growth of both applications and implementation technology. The shrunk time-to-market has made this situation worse. There is a wide consensus that only a reuse of highly optimized cores can match the demands of the pending applications and the ultra large scale integration. Therefore, a core-based system design methodology attracts much interest of all silicon vendors.

In this paper, we propose an optimization technique for power reduction by using small subprogram, multiple supply and threshold voltages. Frequently executed sequences of object codes are allocated into the subprogram memory. Low supply and threshold voltage are assigned to the subprogram memory so as to minimize power dissipation under a time constraint. Our technique targets a typical application specific system-on-chip, consisting of a simple processor core and two simple instruction memories : a main program memory and a subprogram memory. A compiler simul-

taneously determines size of the two memories, supply voltage, and a threshold voltage.

The rest of the paper is organized in the following way. In Section 2, we discuss the motivations for our work and present our concept to optimize the instruction memory. A power optimization technique based on the code allocation and voltage scaling is proposed in Section 3. Section 4 presents experimental results and discussion on the effectiveness of the approach. Section 5 concludes this paper.

2 Motivations and Our Approach

2.1 Voltage Scaling in Memory

Voltage scaling is one of the most effective way to reduce the power consumption in CMOS VLSI circuits, because active power dissipation in CMOS circuits is quadratically proportional to the supply voltage as given by

$$E_{total} = E_{active} + E_{stand-by} \quad (1)$$

$$E_{active} \propto V_{dd}^2 \quad (2)$$

where E_{active} is active energy dissipation, $E_{stand-by}$ stand-by energy dissipation, and V_{dd} supply voltage[3]. However lowering the supply voltage causes performance degradation as shown in (3).

$$t_{pd} \propto \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \quad (3)$$

t_{pd} denotes propagation delay, and V_{th} threshold voltage of the device. α is a factor depending on the carrier velocity saturation and is about 1.3 in advanced MOSFETs.

It is clear from (3) that the circuit delay (t_{pd}) in CMOS circuits can be improved by lowering the threshold voltage (V_{th}). However, this causes explosive increase of stand-by leakage current. The stand-by energy consumption can be given by

$$E_{stand-by} \propto 10^{-\frac{V_{th}}{S}} \cdot V_{dd} \quad (4)$$

where S is subthreshold factor and its smallest limit at room temperature is 60 mV/dec. Considering these design tradeoffs in CMOS circuits, scaling down the V_{dd} and the V_{th} of only a small part of memory is effective way to reduce energy consumption without performance degradation.

2.2 Memory Partitioning

In todays memory circuits, an array part is usually partitioned into several segments and only one of them is activated at a time so as to reduce the power consumption as shown in figure1[4, 5, 6]. In such memories, the power consumption can be calculated by the sum of the power dissipated in a single segment and the power dissipated for charging global bit lines.

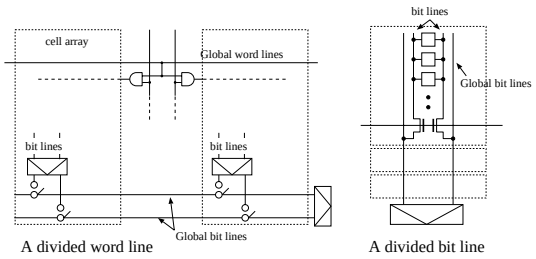

Figure 1: Divided bit and word line structure

Let us assume that the power dissipated for charging the global bit line is in proportional to the number of partitioned segments, and the power dissipated in a single segment is in proportional to the size of the segment. Under these assumptions, the memory power consumption can be approximated by (5), where N_{seg}, N_{word}, α_e, β_e, and γ_e denote the number of segments, the number of words, and coefficients for each term, respectively.

$$E_{memory} = \alpha_e \cdot N_{seg} + \beta_e \cdot \frac{N_{word}}{N_{seg}} + \gamma_e \quad (5)$$

First and second terms of (5) represent the energy dissipated for charging the global bit line and the energy dissipated in a single memory segment, respectively. The last term represents a constant factor in memory power consumption.

Figure 2 shows energy consumption in 32 bits ROMs ranging in words from 512 to 4,096 which are generated by Alliance CAD System Ver. 3.0 with $0.5\mu m$ double metal CMOS technology. The approximated values shown in figure 2 are derived by (5). α_e (= 725.7), β_e (= 0.895), and γ_e (= 1.40) are determined by using the *least squares method*. Figure 2 demonstrates very good accuracy of the approximation.

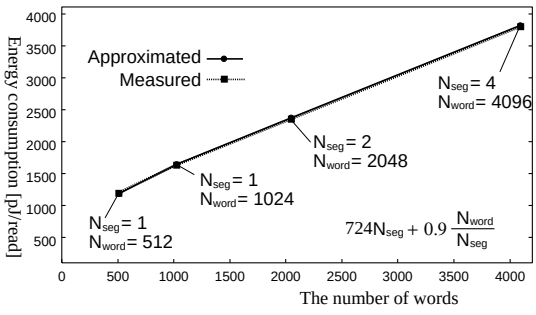

Figure 2: The number of words VS. read energy consumption

From formula(5), it is easy to derive that the number of memory segments which minimizes the memory power consumption is $\sqrt{(\beta_e/\alpha_e) \cdot N_{words}}$. Therefore, power consumption of memory whose array part is optimally partitioned into several segments are formulated by

(6).

$$E_{memory} = 50.97 \cdot \sqrt{N_{words}} + 1.40 \ [pJ/cycle] \quad (6)$$

Memory access time can also be formulated as (7), where α_d and β_d are coefficients of first and second term, and γ_d is constant factor.

$$D_{access} = \alpha_d \cdot N_{seg} + \beta_d \cdot \frac{N_{word}}{N_{seg}} + \gamma_d \quad (7)$$

From measured memory access time in actual memory, $\alpha_d (= 0.149)$, $\beta_d (= 0.61\ e^{-3})$, and $\gamma_d (= 3.236)$ can be determined by using the *least squares method*. If the number of partitioned memory block is determined so as to minimize energy consumption, memory access time is formulated as (8)

$$D_{access} = 0.0226 \cdot \sqrt{N_{words}} + 3.296 \ [ns] \quad (8)$$

We use (6) and (8) in this paper as energy and delay models of memory.

2.3 Our Approach

Memory reference locality is well known fact in many kinds of applications[7, 8]. This means that only a few address in memory are frequently accessed. Therefore, allocating these frequently executed sequences of object codes into a low power subprogram memory reduces total energy of memory[8, 7]. However, if low V_{dd} is used to reduce the energy dissipation, we have to employ low V_{th} to improve performance degradation. In such conditions, the size of subprogram memory should be small so as not to enlarge stand-by leakage current even if low V_{th} is used for the subprogram memory. Our power optimization technique finds optimal point in these trade-offs, where total energy consumption is minimized under a performance constraint.

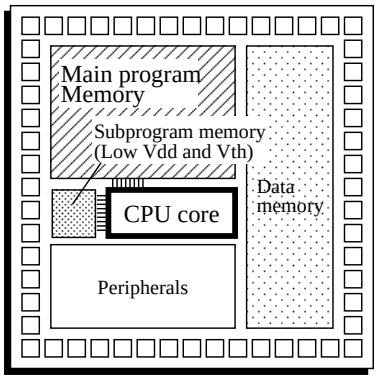

Figure 3: Power reduction considering memory reference locality

In many embedded systems, object codes of the application programs need not to be modified after system design is completed. Therefore, the object code of the programs are stored into ROM. Our approach also targets systems which assume to employ embedded ROM. The most important merit of our approach is a suitability for IP-base system design. Our technique requires no modification to the internal processor architecture. Only inserting extra jump instructions at compiling phase for subprogram calls and for returns from subprogram are required to implement our approach.

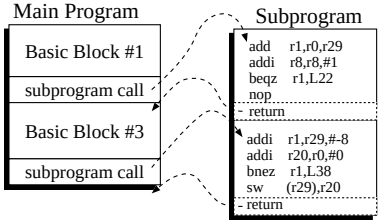

Figure 4: Code allocation to a subprogram memory

3 Memory Power Optimization with Multiple V_{dd} and V_{th}

At first, we present a power optimization technique based on voltage scaling and code allocation. Next, we formulate a *memory optimization problem* as a combinational optimization problem. The *memory optimization problem* is a problem to determine a code allocation, V_{th}, and V_{dd} for two program memories so as to minimize energy consumption under a time constraint.

3.1 Optimization Flow

The procedure of our power optimization technique is described below.

1. Object codes of a target program and a energy dissipation and access time models of ROM as a function of size, V_{dd}, and V_{th} are given. In this paper we use (2), (4), (3), (6) and (8) as models of ROM.

2. A compiler measures the execution count of each basic block using some sample data. A basic block is characterized by execution counts and the number of instructions. We present a detailed explanation about the *memory optimization problem* in section 3.2.

3. For a given set of basic blocks and models of ROM, a compiler optimizes code allocation, V_{th}, and V_{dd} simultaneously under a time constraint.

3.2 Problem Formulation

In this section, we present a problem formulation for the *memory optimization problem*. Firstly, we give notations used in the formulation. Next, we present a problem formulation.

- S : A subthreshold factor. We use $S = 0.08[V/dec]$ in this paper.
- α : A factor depending on the carrier velocity saturation. We use $\alpha = 1.3$ in this paper.
- B : A set of basic blocks appeared in a given program.
- b_i : The ith basic block. $b_i = (N_i, X_i, a_i) \in B$
- N_i : The number of instructions included in ith basic block.
- X_i : The execution count of ith basic block.
- T_{exe} : Total execution time.
- W_m, W_s : The number of instructions allocated in main program memory and subprogram memory, respectively.
- A_m, A_s : Access counts to main program memory and subprogram memory, respectively.
- E_m, E_s : Energy consumption per read access to main program and subprogram memory, respectively.
- E_{oh}, D_{oh} : Energy and delay over head caused by employing a subprogram memory. We use $E_{oh} = \alpha_e$, $D_{oh} = \alpha_d$ in this paper (see section 2.2).
- a_i : 0-1 integer variables associated with b_i. $a_i = 1$ if b_i is allocated in subprogram memory, otherwise, $a_i = 0$.
- vd_m, vd_s : Supply voltage assigned to main program and subprogram memory, respectively.
- vt_m, vt_s : Threshold voltage used in main program and subprogram memory, respectively.
- $k_{1..6}$: Coefficients. k_3, k_4, k_5, and k_6 are determined according to (6) and (8).

$$O = k_1 \cdot (W_m \cdot vd_m \cdot 10^{-\frac{vt_m}{S}} + W_s \cdot vd_s \cdot 10^{-\frac{vt_s}{S}}) \cdot T_{exe} + k_2 \cdot \{A_m \cdot (E_m + E_{oh}) + A_s \cdot E_s\} \quad (9)$$

$$T_{exe} \leq T_{const} \quad (10)$$

$$1.5 \geq vd_m > 3vt_m > 0, \quad 1.5 \geq vd_s > 3vt_s > 0 \quad (11)$$

$$W_m = \sum_{i=0}^{|B|} \{N_i \cdot (1 - a_i) + a_i \cdot (1 - a_{i-1})\} \quad (12)$$

$$W_s = \sum_{i=0}^{|B|} \{N_i \cdot a_i + a_i \cdot (1 - a_{i+1})\} \quad (13)$$

$$A_m = \sum_{i=0}^{|B|} \{N_i \cdot X_i \cdot (1 - a_i) + X_i \cdot a_i \cdot (1 - a_{i-1})\} \quad (14)$$

$$A_s = \sum_{i=0}^{|B|} \{N_i \cdot X_i \cdot a_i + X_i \cdot a_i \cdot (1 - a_{i+1})\} \quad (15)$$

$$E_m = \left(k_3 + k_4 \cdot \sqrt{W_m}\right) \cdot vd_m^2 \quad (16)$$

$$E_s = \left(k_3 + k_4 \cdot \sqrt{W_s}\right) \cdot vd_s^2 \quad (17)$$

$$T_{exe} = \frac{\left(k_5 + k_6\sqrt{W_m} + D_{oh}\right) \cdot vd_m}{(vd_m - vt_m)^\alpha} A_m + \frac{\left(k_5 + k_6\sqrt{W_s}\right) \cdot vd_s}{(vd_s - vt_s)^\alpha} A_s \quad (18)$$

An object function of this optimization problem is (9). The O represents the total energy consumption including dynamic energy consumption and stand-by energy consumption. Constraints are (10) and (11). The first constraint is timing constraint. The variables to be determined are vd_m, vd_s, vt_m, vt_s, and a set of a_i.

The *memory optimization problem* is formally defined as follows. " For a given time constraint T_{const} and a given set of basic blocks B, find vd_m, vd_s, vt_m, vt_s, and a set of a_i which minimize O under a time constraint".

3.3 Algorithm

The worst case computation time to solve the *memory optimization problem* strongly depends on the time to search an optimal code allocation $\{a_i\}$, because complexity of finding optimal vd_m, vd_s, vt_m, and vt_s is trivial in contrast with the complexity of finding optimal code allocation. If a naive algorithm is applied, worst case computation time is $O(2^{|B|})$, where B is a set of basic blocks appeared in a given program. In this paper we use heuristic algorithm described in Fig. 5. The complexity of this heuristic algorithm is $O(|B|^2)$.

The inputs to algorithm *Memory optimization* are a set of basic blocks B, where each basic block $b_i \in B$ is characterized by its execution count X_i, its size N_i, and its allocation a_i. All the a_i is set to zero at first step. This means that all the basic blocks are allocated in the main program memory. Next, the algorithm selects a basic block which can most improve the object function O by relocating this basic block to the subprogram memory and assigning optimal vd_m, vd_s, vt_m, and vt_s. The selected single basic block which can most improve O is allocated to the subprogram memory. This process is done repeatedly while the algorithm reduces O. If the O becomes not to

Given: a set of basic blocks B, where each basic block $b_i \in B$ is characterized by its execution count X_i, its size N_i, and its location a_i.
Algorithm Memory optimization
  for each $b_i \in B$
    a_i=0;
  end for
  while (the algorithm leads to reduction in O)
    $B_{main} = \{b_j \in B \mid a_j = 0\}$
    for each $b_i \in B_{main}$
      Find a single b_k which can most improve O after V_{dd} and V_{th} are optimized under a T_{const}
      Relocate the b_k into the subprogram memory
    end for
  end while
  Output a set of basic blocks B ;
end Algorithm

Figure 5: Algorithm for the memory optimization

be improved, the algorithm stops after outputting the updated set of basic block B, and voltage assignments.

4 Experimental Results

We use three benchmark programs shown in Table 1, in this experiments. The benchmark programs are compiled by gcc-dlx compiler which is based on GNU CC Ver. 2.7.2 for DLX architecture[9]. Table 2 shows description of three kinds of sample video images used as input to the MPEG2 program. Three kinds of sample input data were also used for Arithmetic calculator and TV remote controller.

At first, we evaluate the following four cases using MPEG2 decoder and Image1.

Case1 Optimizing V_{dd} and V_{th} of a program memory without using a subprogram memory.

Case2 Optimizing memory allocation to main program and subprogram memory without voltage scaling. $V_{dd} = 1.5V$ and $V_{th} = 0.4V$ are used.

Table 1: Description of benchmark programs

Benchmark	# of basic blocks
Arithmetic Calculator	2,103
TV Remote Controller	2,994
MPEG2 Decoder	5,361

Table 2: Description of sample data

Data	# of frames	The size of frames
Image1	50	416x386
Image2	26	352x224
Image3	14	704x480

Case3 Optimizing memory allocation, V_{dd} and V_{th} of two program memories. However, the optimization is limited to $vd_m = vd_s$ and $vt_m = vt_s$.

Case4 Solving the *memory optimization* problem by our algorithm described in section 3.3.

The results are shown in Figure 6. Vertical axis represents energy consumption normalized at minimized energy consumption in Case4. Horizontal axis represents time constraints normalized at an execution time of a program when $V_{dd} = 1.5V$, $V_{th} = 0.5V$ and the subprogram is not used.

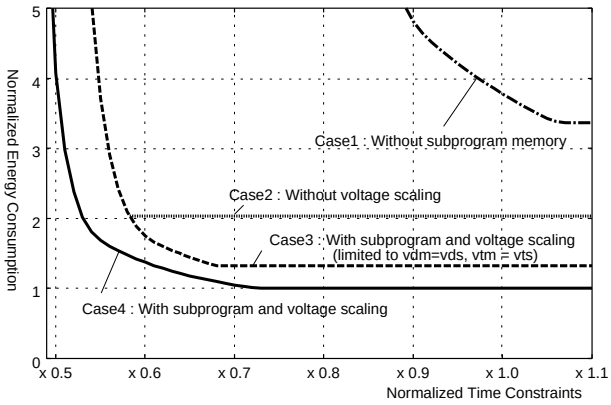

Figure 6: Energy consumption results of MPEG2 decoder under time constraints

The results show that the most effective way to reduce the memory power consumption is using subprogram memory. We can reduce the memory power consumption to less than one third by using the subprogram memory. And also, voltage scaling can halve the memory power consumption. If the voltages are not scaled, relaxing time constraint does not leads to an energy reduction, because a code allocation optimized for high performance is almost same from an allocation optimized for power reduction. The results of Case3 and Case4 demonstrates that about 30% power reduction can be achieved by using multiple voltages. In other point of view, we can achieve 10% speed up without increasing energy consumption by using multiple supply voltages. Especially for a very strict time constraint, using different threshold voltages for the two memories is effective, because using low threshold voltage for the large main program memory leads to explosive increase in subthreshold leakage current. In total, an energy reduction by our approach is up to 80%, when the time constraint is a "$\times$ 0.9'.

The optimized memory sizes in words, $V_{dd}s$, and $V_{th}s$ of main program and subprogram memories are shown in Table 3. A "$\times$ 0.7" is used

Table 3: The results of optimized memories

Arithmetic Calculator			
	Data1-1	Data1-2	Data1-3
W_m	10,620	10,542	10,524
vd_m	1.42V	1.46V	1.48V
vt_m	0.431V	0.437V	0.434V
W_s	333	413	441
vd_s	1.50V	1.50V	1.50V
vt_s	0.300V	0.305V	0.307V
TV Remote Controller			
	Data2-1	Data2-2	Data2-3
W_m	13,889	14,396	13,822
vd_m	1.47V	1.34V	1.50V
vt_m	0.438V	0.445V	0.452V
W_s	1,315	740	1,396
vd_s	1.50V	1.49V	1.50V
vt_s	0.333V	0.335V	0.341V
MPEG2 decoder			
	Image1	Image2	Image3
W_m	28,244	28,188	28,384
vd_m	1.47V	1.50V	1.47V
vt_m	0.490V	0.500V	0.490V
W_s	1,297	1,369	1,145
vd_s	1.08V	1.11V	1.07V
vt_s	0.353V	0.356V	0.351V

as a time constraint. The results indicate that the size of the subprogram memories are from 4% to 10% of main program memories. This is a key reason of a drastic energy reduction in memory. In some cases, the supply voltage of subprogram memory is higher than that of main program memory, because assigning higher voltage to subprogram memory reduces total execution time and this makes slack time which enables supply voltage of main program memory to be lowered. As we can see from the results, the optimal size, V_{dd}, and V_{th} of memories strongly depend on time constraints, data, and application programs. Therefore, a design flexibility which enables system designers to easily optimize design trade-offs is very important.

5 Conclusion

As device size is shrunk, supply voltage of CMOS circuits is also scaled down. In near future, 1.5V or 1.0V supply voltage will become common. In such era, scaling the threshold voltage becomes to have strong impacts on both energy consumption and circuit delay. In this paper, we have proposed a system level memory power reduction technique based on voltage scaling and code allocation. Experimental results demonstrated that the energy consumption in the memories optimized by our approach can be less than 20% of energy in memory which does not employ our approach. Since the optimal size, V_{dd}, and V_{th} of memories strongly depend on time constraints, kinds of data, and kinds of applications, easy programmability of design parameters is indispensable for sophisticated design. Design flexibility of our approach must be a key technology for large and low power SoC design.

Our future work will be devoted to extend the proposed optimization technique considering the data memory.

References

[1] Nikolaos Bellas, and Ibrahim Hajj,. "Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High Performance Microprocessors". In *Proc. of International Symposium on Low Power Electronics and Design*, pages 70–75, 1998.

[2] S. Mutoh S. Date, N. Shibata and J. Yamada. 1-V, 30-MHz Memory-Macrocell-Circuit Technology with a $0.5\mu m$ Multi-threshold CMOS. In *Proc. of IEEE Symposium on Low Power Electronics*, pages 90–91, 1994.

[3] T. Hiramoto and M. Takamiya. "Low Power and Low Voltage MOSFETs with Variable Threshold Voltage Controlled by Back-Bias". *IEICE Trans. on Electronics*, E83–C(2):161–169, February 2000.

[4] H. Shinohara T. Yoshihara H. Takagi S. Nagao S. Kayano M. Yoshimoto, K. Anami and T. Nakano. "A Divided Word-Line Structure in the Staticture in the Static RAM and its Application to a 64K Full CMOS RAM". *IEEE Journal of Solid-State Circuits*, pages 479–485, June 1983.

[5] M. Isobe, J. Matsunaga, T. Sakurai, T. Ohtani, K. Sawada, H. Nozawa, T. Iszuka and S. Kohyama. "A Low Power 46ns 256K bit CMOS Static RAM with Dynamic Double Word Line". *IEEE Journal of Solid State Circuits*, SC-19(5):578–585, May 1984.

[6] Edwin de Angel and Jr. Earl E. Swartslander. "Survey of Low Power Techniques for ROMs". In *Proc. of Int'l Symposium on Low Power Electronics and Design*, pages 7–11, 1997.

[7] L. Benini, A. Macii, E. Macii, and M. Pancino. "Synthesis of application-specific memories for power optimization in embedded systems". In *Proc. of 37th Design Automation Conference*, pages 300–303, June 2000.

[8] T. Ishihara and H. Yasuura. "A Power Reduction Technique with Object Code Merging for Application Specific Embedded Processors". In *Proc. of Design, Automation and Test in Europe*, pages 617–623, March 2000.

[9] J. L. Hennessy and D. A. Patterson. *"Computer Architecture: A Quantitative Approach"*. Morgan Kaufmann Publishers, Inc., 2nd edition, 1996.