ZOOM: A Performance-Energy Cache Simulator

by

Regina Sam

S.B. Electrical Engineering and Computer Science, June 2002
Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2003

Author . . . .
Department of Electrical Engineering and Computer Science
May 23, 2003

Certified by . . . .
Krste Asanović
MIT Thesis Supervisor

Certified by . . . .
Jude A. Rivers
VI-A Company Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

ZOOM: A Performance-Energy Cache Simulator

by

Regina Sam

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2003, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

On-chip cache sizes are growing with each generation of microprocessors in an attempt to bridge the ever-widening memory-processor performance gap. At the same time, energy dissipation in caches has become an important consideration due to increased integration and operating speeds and the explosive growth of battery-operated devices. These trends indicate that there is much to be gained from making energy and area, as well as performance, front-end design issues. This thesis presents ZOOM, a fast, flexible and robust framework for optimizing and characterizing the performance, energy and area of low-power caches in the early-stages of design. ZOOM consists of a timing-sensitive functional simulator and a micro-architecture simulator with HSPICE® netlist generation capability. Preliminary evaluation indicates that the micro-architecture simulator estimates access time and energy to within 10% and 15%, respectively, of HSPICE simulated estimates for both SRAM and CAM-based caches.

MIT Thesis Supervisor: Krste Asanović
Title: Associate Professor

VI-A Thesis Supervisor: Dr. Jude A. Rivers
Title: Research Staff Member, IBM T. J. Watson Research Center

Acknowledgments

This thesis would not have been possible without the support, generosity and kindness of several people. My advisors, Jude Rivers and Krste Asanović, guided this project from its infancy, suggesting enhancements and patiently explaining the nuances of cache design to me. I am very grateful for the opportunity to work with them. I thank the Assam Group for their comments and suggestions on earlier versions of ZOOM. Their suggestions made a big difference in the quality of this thesis. I also thank researchers at IBM for discussions on cache design and simulations.

Chizzy patiently reviewed earlier versions of this document, while providing support for the final stretch. I thank him for his love and encouragement.

Words cannot express my gratitude to Mrs. Sally Honny whose generosity, inspiration and determination made my dream of an American education possible; and to Penny who provided me a home and family away from mine, and supported me every step of the way. My family provided an equally important help by guaranteeing a good laugh whenever I spoke with them. I'm thankful to them for putting up with my prolonged absence from home.

And last but certainly not least, I thank God for seeing me through.

This research was supported by an IBM VI-A fellowship, DARPA/PACE award number F30602-00-2-0562 and a stipend award from the MIT Graduate Students Office.

Contents

1 Introduction **17**

1.1 Terminology 18

1.2 Overview of ZOOM 20

2 Related Work **23**

3 Assumptions and Methodology **27**

3.1 Assumptions 27

3.1.1 Cache Architecture Overview 28

3.1.2 Circuit Style 31

3.1.3 Technology Assumptions 32

3.2 Methodology 35

3.2.1 Capacitances and Resistances 35

3.2.2 Delay Estimation 41

3.2.3 Repeaters 43

3.2.4 Energy Estimation 46

3.2.5 Area Estimation 48

3.2.6 Energy-Delay-Area Optimization 49

4 Micro-Architecture Level Cache Modeling 51

4.1 Decoder 52

4.1.1 Decoding Structure 53

4.1.2 Sizing the Decoder 55

4.1.3 Managing Interconnect Delay in the Decoder 58

4.1.4 Wordline Delay and Energy 58

4.2 SRAM Data Path 61

4.2.1 Data Path Delay and Energy 64

4.2.2 Divided Bitlines 70

4.3 SRAM Tag Comparators 77

4.4 CAM Tag Path 83

4.4.1 CAM Tag Path Delay and Energy 85

4.5 Data Routing 95

4.5.1 Local Output Bus 96

4.6 Putting It All Together 99

4.6.1 Cache Access Time and Energy 99

4.6.2 Applications of the EDA Optimizer 104

4.7 Comparison with CACTI 107

5 Simulator Framework 109

5.1 Functional Simulator 109

5.1.1 Simulator Components 111

5.1.2 Simulator Inputs and Outputs 115

5.2 Micro-Architecture Level Simulator 118

5.2.1 Automated Component Netlist Generation 122

6 Conclusion 123

6.1 Future Plans 124

A General Simulator Information 125

A.1 Running the Simulator 125

A.2 Sample I/O Files for Functional Simulator 126

A.3 Sample I/O Files for Micro-Arch Simulator 130

List of Figures

1-1 Overview of ZOOM 20

3-1 Address decomposition in a set-associative or direct-mapped cache . . 28

3-2 Organization of a direct-mapped SRAM-based cache 29

3-3 CAM cache overview 30

3-4 (a) 6-T SRAM cell (b) 10-T CAM cell 31

3-5 Discharging a capacitor though an NMOS 36

3-6 Main contributions of transistor capacitance 37

3-7 (a) Wire capacitance model (b) Wire dimensions 40

3-8 Extrinsic delay of a logic gate driving a load through an interconnect line 42

3-9 (a) Normalized delay of 3 cm Al-1 wire for varying number of repeaters. Dashed line shows the point at which the delay comes within 5% of the minimum. (b) Normalized energy of 3 cm Al-1 wire for varying number of repeaters. Dashed line shows the factor reduction in energy for the value of m whose delay comes within 5% of the minimum. . . 45

3-10 Short-circuit currents during transients (from [31]) 47

3-11 Summary of lambda-based design rules (in units of λ) 48

4-1 SRAM partitioning example with 8 sub-arrays 52

4-2 Decoding structure for a 4-16 decoder 53

4-3 (a) Growth rate of path effort of n-input NAND implementation using one stage of an n-input NAND gate *vs.* multiple stages of 2-input NANDs and inverters. (b) Parasitic delay of n-input NAND implementation using n-input NAND gate *vs.* multiple stages of 2-input NANDs and inverters for a fixed number of stages. 54

4-4 Critical Path of Decoder 55

4-5 Comparison of decoder delay and energy estimates with HSPICE simulated values 57

4-6 Decoder delay before and after repeater insertion 59

4-7 Wordline Architecture 60

4-8 Comparison of wordline delay and energy estimates with HSPICE simulated values 60

4-9 Basic structure of SRAM array (from [8]) 62

4-10 Sense amplifier circuit 63

4-11 Schematic of bitline structure 65

4-12 Current source driving a RC π network 67

4-13 Comparison of bitline delay and energy estimates with HSPICE simulated values 69

4-14 Divided bitline example with $M = 4$ 70

4-15 Schematic of divided bitline 72

4-16 RC network for divided bitline 73

4-17 (a) Normalized DBL delay for arrays of height 256 and 1024 for metal-to-bitline capacitance ratios of 0.9 and 0.56 (b) Normalized DBL energy for arrays of height 256 and 1024 for metal-to-bitline capacitance ratios of 0.9 and 0.56 74

4-18 Impact of metal-to-bitline capacitance ratio on delay scalability with M 77

4-19 SRAM-tag comparator structure 78

4-20 Equivalent circuit of SRAM comparator discharge path 79

4-21 Example of branching in the comparator buffer chain 81

4-22 Critical path of comparator buffer chain 82

4-23 Comparison of comparator delay and energy estimate with HSPICE simulated values 83

4-24 Basic structure of a CAM array 84

4-25 Split comparator architecture / CAM path 86

4-26 CAM critical path 86

4-27 Equivalent RC of CAM compare critical path 88

4-28 (a) CAM path delay scaling with number of tags (columns) per half section for 32 rows. (b) CAM path delay scaling with number of rows (associativity) for a fixed half-section column width of 13 bits 91

4-29 CAM path energy scaling with associativity (number of rows) for a fixed column width of 13 bits per half section 94

4-30 Floor Planning example for SRAM cache with 32 data sub-arrays . . 96

4-31 Local bus structure 97

4-32 (a) Access time scaling with cache size for SRAM cache using single-phase access (b) Access energy scaling with cache size for SRAM cache using single-phase access 103

4-33 (a) Access time scaling with cache size for CAM cache (b) Access energy scaling with cache size for CAM cache 105

4-34 (a) Energy vs Delay plot of 256KB, direct-mapped cache (b) Area vs. Delay plot for 256KB, direct-mapped cache 106

4-35 Relative delay, energy and area efficiencies for CACTI and ZOOM . . 108

5-1 Overview of Functional Simulator 110

5-2 Flow diagram of access-processing in functional simulation 113

5-3 Overview of micro-architecture simulator 119

5-4 Flow of micro-architecture level simulation 121

List of Tables

3.1 Scaling of Relevant Parameters for Short-Channel Devices 32

3.2 Metal layer Assignments 34

3.3 Wiring Dimensions 34

3.4 Logical effort and intrinsic delay of common 2:1 sized gates 39

Chapter 1

Introduction

A *cache* is the first level of the memory heirarchy encountered once an address leaves the computer processing unit (CPU) [23]. It is a rather small (compared to the memories on other levels of the heirarchy), albeit expensive, temporary storage that is able to supply most of the information requests of the CPU, due to some tailored strategies that control its operation.

On-chip cache sizes are growing with each generation of microprocessors in an attempt to bridge the ever-widening memory-processor performance gap. At the same time, energy dissipation in caches has become an important consideration due to increased integration and operating speeds and the explosive growth of battery-operated devices. According to a literature survey in [24], caches consume 25 to 50% of total chip energy, and account for 15 to 40% of total chip area. Whereas designers have traditionally focussed their design efforts on improving cache performance, these statistics and technology trends indicate that there is much to be gained from making energy and area, as well as performance, front-end design issues. This thesis presents

ZOOM, a fast, flexible and robust framework for optimizing and characterizing the performance, energy and area of low-power caches in the early-stages of design. In order to understand the description of ZOOM and related work that follows, one must be aware of the terminology used to describe caches and cache events. The rest of this chapter presents a brief explanation of this terminology followed by an overview of ZOOM.

1.1 Terminology

Three calssifications of caches are possible, depending on the type of information they store. An *instruction cache* stores CPU instructions, a *data cache* stores data for the running application and a *unified cache* stores both instructions and data. The basic requests to a cache are *reads* and *writes*. For reads, the cache receives, from the CPU, the binary *address* for a location in memory and it returns the information stored at that address. Write requests consist of an address and the new information to be written to the location specified by the address. If the location specified by the address is stored in the cache, a *hit* results, otherwise, a *miss* results and the request is forwarded to the next memory in the heirarchy. A *read miss* is a read request that results in a miss and a *write miss* is a write request that results in a miss.

A *block* is the name typically given to the smallest unit of information that may be present in the cache. Three categories of cache organization result depending on where a given block may be placed in the cache. If the number of possible locations for each block is one, the cache is said to be *direct-mapped*. If a block can be placed anywhere in the cache, the cache is said to be *fully-associative*. If a block can be placed only in one of a restricted *set* of n places, the cache is said to be *n-way set-associative*.

When a miss occurs, the cache must select a block to be replaced with the data being fetched from the next-level memory. In a direct-mapped cache, there is only one choice: the block that was checked for a hit. In a set-associative or fully-associative cache, any of the blocks in the set may be replaced. There are three primary *replacement policies* for choosing which block is replaced in this case. A *FIFO policy* replaces the oldest block in the set, an *LRU policy* replaces the **l**east-**r**ecently **u**sed block, and a *Random policy* randomly chooses any of the blocks in the set to be replaced.

Since the cache is a temporary storage, writes to a given block need to be propagated to main memory before the modified block is replaced on a miss. A *write-through* cache modifies its own copy of the information and the copy stored in main memory at the time of the write. A *copy-back* cache modifies its own copy of the stored information at the time of the write, but only updates the copy in main memory when the modified block is selected for eviction. Whereas read misses typically result in the requested information being fetched into the cache, write misses do not necessarily require that the cache fetch the modified block. The block is loaded on a write miss if the cache is using the *write-allocate* strategy, otherwise the write request is simply forwarded and the modified data is not loaded into the cache. In this case, the cache is said to be *non-allocating*.

For the interested reader, a detailed treatment of caches is found in [23]. We now proceed with an overview of ZOOM.

1.2 Overview of ZOOM

ZOOM operates at two levels of abstraction via two stand-alone simulators, which may be optionally combined. Figure 1-1 is a high-level block diagram of ZOOM.

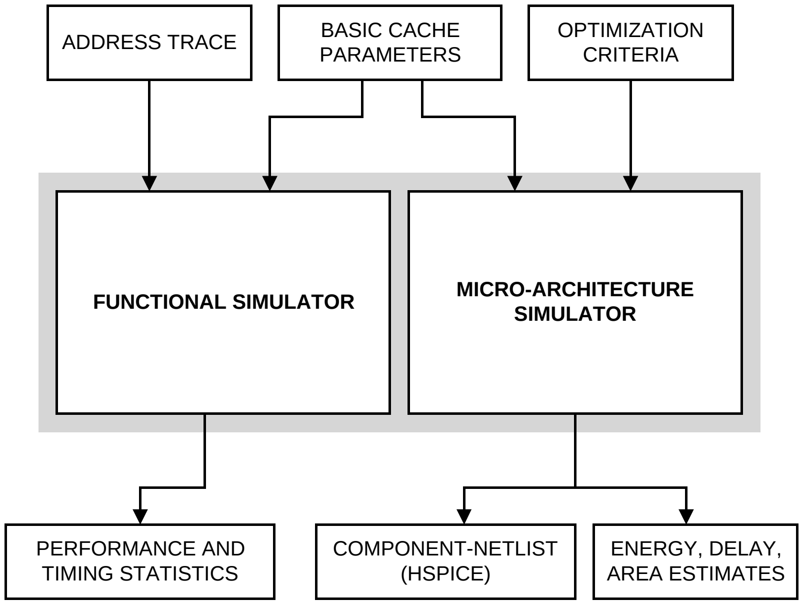

ADDRESS TRACE
BASIC CACHE PARAMETERS
OPTIMIZATION CRITERIA
FUNCTIONAL SIMULATOR
MICRO-ARCHITECTURE SIMULATOR
PERFORMANCE AND TIMING STATISTICS
COMPONENT-NETLIST (HSPICE)
ENERGY, DELAY, AREA ESTIMATES

Figure 1-1: Overview of ZOOM

The *Functional Simulator* is a behavioral model for dynamic cache evaluation. Using an address trace and some basic cache information such as cache size, block size and associativity, the simulator provides performance statistics and cycle-accurate estimates of average access latencies and other performance penalties. These performance statistics are independent of the circuit-level implementation of the cache and may assist in architecture-level choices such as the appropriate number of cache ports, cache size, block size, associativity, and so on.

The *Micro-Architecture Simulator* consists of analytical transistor-level models for estimating access time, energy and area of the cache. Its inputs include basic cache information similar to that used by the Functional Simulator, a set of optimization criteria and a process technology specified by its supply voltage and feature size. Energy, delay and area estimates from the analytical models are used in a general optimization scheme to determine the optimum cache configuration for the specified optimization criteria. The outputs from the simulator include estimates for energy, delay and area and HSPICE® netlists for the main components of the proposed cache configuration.

ZOOM is designed to be easily adaptable to existing and novel cache configurations. The Functional Simulator has built-in support for common single and multi-level cache configurations, and a simple interface for modeling multi-level caches with special purpose buffers or those with atypical interactions between caches on different levels.

The Micro-Architecture simulator supports both SRAM (*Static Random-Access Memory*) and CAM (*Content-Addressable Memory*) caches. Although the underlying design idealogy emphasizes low-power techniques, only those techniques with minimal performance penalties are emphasized, making the simulator also suitable for high-performance cache characterization. The component netlist returned by the built-in Netlist Generator facilitates use of the proposed model as a starting point for actual designs by reducing the labor involved in generating netlists from scratch.

ZOOM relies on analytical models and design techniques proposed by many researchers. Chapter 2 briefly describes a sampling of these related works; Chapter 3 discusses the general estimation methodology and assumptions about the cache architecture and process technology. Chapter 4 presents circuit-level models for the main components of the cache as well as an evaluation of these models. The software framework of ZOOM is described in Chapter 5 and we conclude with a summary of future improvements in Chapter 6.

Chapter 2

Related Work

Caches have been the subject of many architectural and micro-architectural studies, which have proposed various techniques to improve their energy and access times. This discussion of related work focusses on a sampling of works that proposed models for early-stage cache simulation or for SRAM design and characterization. Related work can be grouped into functional cache simulators, and analytical transistor-level models and simulators.

Functional simulators are typically employed in the earliest stages of cache design. They are used, together with the address trace of a benchmark program to assist in making architecture-level choices such as an appropriate cache size, block size, replacement policy, write hit and write miss strategies etc. Typical statistics from functional simulators include estimates of hit rates and the cycle penalties of cache stalls.

Sim-Cache, which is part of the *Simple-Scalar Simulations Tools Set* [28] and *ml-cache* [30], are most closely related to the functional simulator in ZOOM. *Sim-Cache* is a simple functional simulator for basic event-driven cache simulation. It supports multi-level direct-mapped, set-associative and fully-associative write-allocate instruction and data caches using the LRU replacement policy. *Sim-Cache* uses an "access handler" which controls the interaction between caches on different levels in a multi-level cache configuration. This simple interface makes it possible to model some atypical multi-level caches. However, since the "access handler" is only invoked upon misses, it can only support multi-level caches with atypical miss-processing. Read and write misses in *Sim-Cache* always result in the block being placed in the cache. Hence, it cannot model caches using special-purpose buffers such as the Victim buffer [27] used to hold evicted blocks in some cache configurations.

mlcache [30] is an event-driven, timing-sensitive functional simulator based on the Latency Effects cache timing model. It consists mainly of a library of cache state and data movement functions that may be configured or assembled to model *multi-lateral* caches. *Multi-lateral* caches are caches that contain two or more data stores that have disjoint contents and operate in parallel. *mlcache* places no restrictions on the interactions between caches on different levels. This versatility makes it attractive for cache architecture studies. However, the lack of a standardized interface for cache modeling greatly reduces its usability.

The Functional Simulator in ZOOM combines the simplicity of *Sim-Cache* with most of the flexibility of *mlcache*. Like *Sim-Cache*, ZOOM uses an "access handler" to control interactions between caches on different levels. However, unlike *Sim-Cache*, ZOOM allows the user to determine when blocks are placed in the cache rather than automatically fetching blocks on misses. ZOOM also supports a wider range of caches and also includes a timing model similar to that used in *mlcache*.

Related work at the micro-architecture level consists mainly of analytical models for energy, delay and area of SRAMs and SRAM-based caches.

The CACTI model [1, 18], based on a cache model by Wada *et al.* in [2], is a cycle time, energy and area estimator for SRAM and (recently) CAM caches. CACTI gives delay estimates to within 10% of HSPICE® simulated values for their assumed circuit structures. However, the base CACTI cache model is not particularly energy efficient. According to studies performed in [9], CACTI over-estimates the energy of caches organized for reduced power dissipation by as much as a factor of 10 for a single sub-bank. It also uses a static optimization scheme that does not allow simulation of different design targets.

Analytical models for energy and delay of SRAMs have been studied by many authors. Evans and Franzon develop analytical models in [4] for the energy consumption of a SRAM as a function of its organization and evaluate various simulation approaches for their relative speed and accuracy. They conclude that rectangular SRAM organizations with fewer columns than rows result in lower energy dissipation in SRAMs while square-ish organizations that balance the wordline and bitline paths result in reduced access times. They also found that simulation approaches that combine circuit level extractions with analytical models can be almost as accurate as conventional circuit simulators but are orders of magnitudes faster, and not much slower than their purely analytical counterparts which are much less accurate.

In [6] and [8], Amrutur *et al.* extend the delay models of [1] to include the effects of interconnect resistance and introduce a partitioning scheme that uses multiplexors inside the bitline. They combine these with energy and area models for extensive treatment of SRAM design and analysis and also introduce heuristics for optimally sizing the decoder in SRAMs for different energy and delay optimization preferences.

CAM-based caches have not received as much research spotlight as SRAM-based caches. However, they have been used in many leading commercial processors due to their energy-efficiency at high associativities. For example, the *ARM3* [10] has a 4KB 64-way set-associative CAM-based cache and the Intel^{TM} XScale processor [25] employs 64-way set-associative CAM tags. Zhang and Asanović show in [9] that CAM-based caches have comparable access latency, but give lower hit energy and higher hit rates than SRAM-based set-associative caches at the expense of approximately 10% area overhead. Their results demonstrate that CAM-based caches are well-suited for both high-performance and low-power designs since they are fast, even at high associativities, and energy-efficient.

The Micro-Architecture Simulator in ZOOM exploits many of the low-power design techniques and models proposed by these authors and introduces models for energy, delay and area estimation of CAM-based caches. We also introduce models for divided bitlines, an approach for reducing the delay and energy consumption in SRAM bitlines introduced in [17]. ZOOM is the first work in its class of cache simulators to combine many of these techniques in an optimization scheme in a unified software framework.

Chapter 3

Assumptions and Methodology

This chapter presents the general assumptions made about the cache design and a a general overview of the methodology used to estimate energy, delay and area, as well as an overview of the optimization scheme. Assumptions discussed include those made about the cache architecture, circuit styles, interconnect and process technology scaling. This excludes component-specific assumptions, which will be presented along with the component models in Chapter 4.

3.1 Assumptions

Whereas technology scaling provides the most important means for reducing delay and energy of caches, various innovative circuit techniques and partitioning schemes achieve significant improvements in energy and delay at the expense of extra silicon. In order to explore the large cache design space in a tractable manner, ZOOM makes some simplifying assumptions about certain aspects of the design. This section outlines and justifies some of the key assumptions made in modeling the cache.

3.1.1 Cache Architecture Overview

Since the address space is much larger than the capacity of the cache, only a small portion of the address is used to map a given block to a location in the cache. Figure 3-1 is adapted from [23] to help explain this mapping.

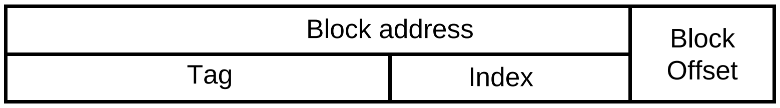


Figure 3-1: Address decomposition in a set-associative or direct-mapped cache

The *index* is used to map the block to a set and is computed as

(Block Address) MOD (Number of sets in the cache)

where a set consists of one block in a direct-mapped cache, and n blocks in an n-way set-associative cache. The *tags* of all the blocks in the set are compared with the tag in the in-coming address to determine the requested block and the block offset is the address of the desired data within the block.

Depending on how the tags are stored and handled, a cache may be classified as *SRAM-based* or *CAM-based*. An SRAM-based cache stores tags in a static RAM array while a CAM-based cache stores tags in a CAM array. Both types of caches store data in SRAM arrays. A synchronous cache architecture is assumed, i.e. a clock triggers the access. These memory structures will be described in detail in Chapter 4.

SRAM-based Cache Architecture

Supported SRAM-based caches may be direct-mapped or set-associative. Fully-associative caches are only supported in a CAM configuration.

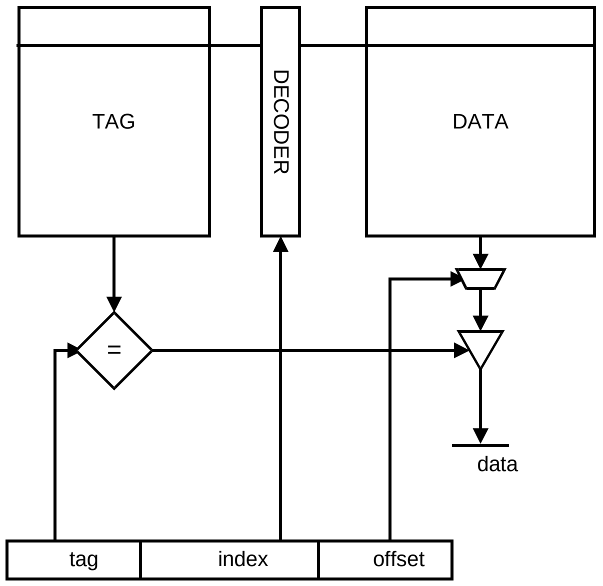


Figure 3-2: Organization of a direct-mapped SRAM-based cache

Figure 3-2 shows the conceptual organization of a traditional direct-mapped cache. An n-way set associative cache would have n of these arrangements and each access to the cache would result in n tag comparisons and the discharge of n data words in parallel. However, only one of these discharged words is eventually driven onto the cache-CPU port if a hit is detected. A slower but more energy-conscious two-phase variation of this sequence serializes the tag comparison and data discharge so that only one data word is discharged when a hit is detected in its associated tag. Two-phase accesses practically double the access time of the cache but achieve about a factor of n reduction in energy consumption in the data array. Both single and two-phase SRAM cache accesses are supported in ZOOM.

There is a wide variation in the physical implementation and layout of SRAM caches. For simplicity, the tags and data are kept in completely independent arrays and do not share any part of the decoder. To trade area for reduced access time and energy, large arrays are typically partitioned into smaller sub-arrays each of which stores all or part of the accessed word. The number of sub-arrays in the cache and their dimensions (or aspect ratio) are determined by the user-specified optimization criteria supplied as inputs to the micro-architecture level simulator. According to research in [4], low-energy solutions tend to have rectangular sub-arrays while high-speed solutions have square-ish sub-arrays.

CAM Cache Architecture

CAM caches are only supported in the set-associative and fully-associative configurations. Figure 3-3 shows the basic configuration of one set of a k-way set-associative

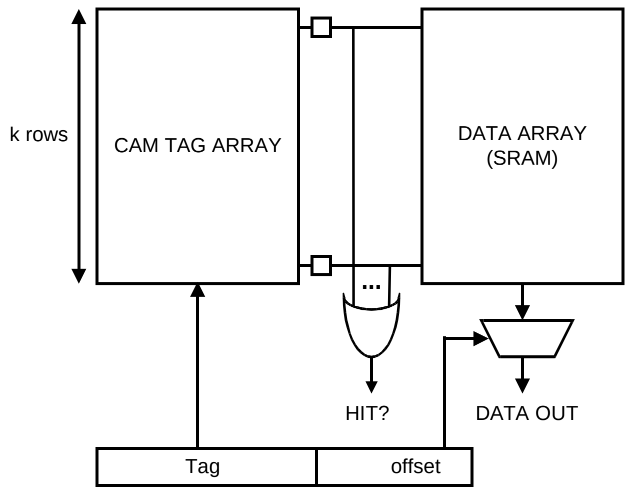


Figure 3-3: CAM cache overview

CAM cache. A cache with m sets has m such arrangements. Accesses to a CAM cache are inherently two-phased: data stored at a given location is addressed by the tag stored in its corresponding tag entry in the CAM array. Hence, the tag check needs to be completed before the appropriate data can be located and discharged. Traditionally, CAMs were used only in the fully-associative sense. However, many modern processors employ them in set-associative configurations in which a set is entirely contained in one of many sub-arrays, with one sub-array enabled per access by pre-decoding part of the address. Unlike the SRAM cache, partitioning of a CAM cache is largely dictated by the associativity; and each data sub-array is placed close to its corresponding tag sub-array. The data sub-array associated with each CAM sub-array may be vertically partitoned to reduce the number of words discharged per access.

3.1.2 Circuit Style

Traditional 6-transistor and 10-transistor memory cell designs are assumed for the SRAM and CAM arrays respectively. Figures 3-4 (a) and (b) show the SRAM and CAM cell respectively.

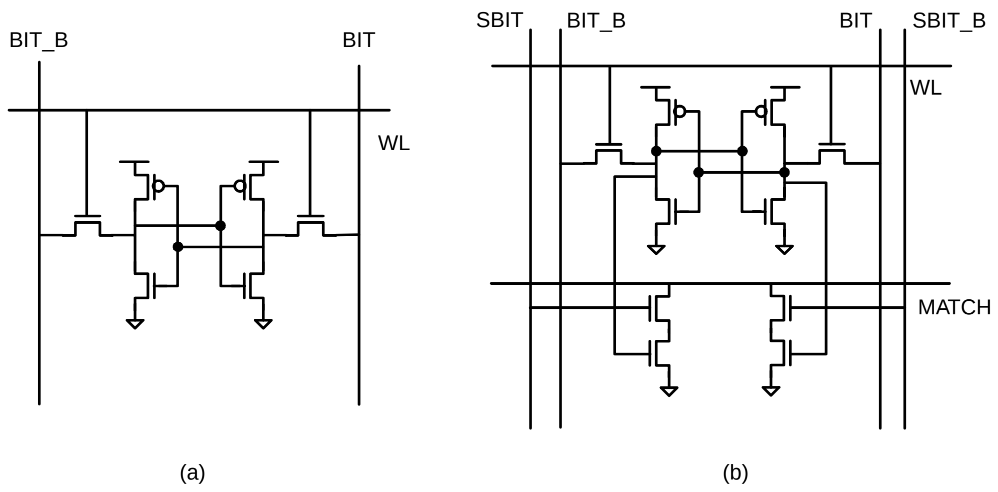


Figure 3-4: (a) 6-T SRAM cell (b) 10-T CAM cell

A static circuit style in which the PMOS transistors are sized to have twice the width of the NMOS transistors is assumed for all the gates. This results in the PMOS pull-up and NMOS pull-down yielding approximately the same delay. In actual cache implementations, gates may be skewed to improve the critical path. The impact of this skewing on the delay is quantified in [8].

3.1.3 Technology Assumptions

ZOOM is written for short-channel devices. Aluminum metallurgy is assumed for the base 0.25μm-2.5V technology and copper metallurgy is assumed for the 0.18μm generation onwards. A convenient process-independent unit of length, λ, which is equal to half the drawn minimum feature size, is used to describe geometric parameters.

Table 3.1: Scaling of Relevant Parameters for Short-Channel Devices

Parameter	Relation	Scaling factor	Comments
W, L, t_{ox}		$1/S$	Device width, length, oxide thickness
V_{dd}, V_t		$1/U$	Supply voltage, threshold voltage
A	WL	$1/S^2$	Area/ Device
C_{ox}	$1/t_{ox}$	S	Gate capacitance per unit area
C_{gate}	$C_{ox}WL$	$1/S$	Gate capacitance
k_n, k_p	$C_{ox}W/L$	S	NMOS and PMOS gain factor
I_{sat}	$C_{ox}WV$	$1/U$	Saturation current
R_{on}	V/I_{sat}	1	"on" resistance
τ	$R_{on}C_{gate}$	$1/S$	Delay of an inverter driving a same-sized inverter
P	$I_{sat}V$	$1/U^2$	Power

Feature size and supply voltage may be scaled independently from the base technology via *general scaling rules*. These are a generalized set of rules that allow the supply voltage and device sizes to be scaled differently [31]. General scaling is the feasible alternative to full scaling, which scales both device sizes and voltages to the same extent in order to preserve the electric field patterns of the original devices. Full scaling is not feasible in many cases since designers have to adhere to well-defined standards for supply voltage and signal levels (such as those specified in [32]) to meet compatibility requirements.

For a given feature size, λ and supply voltage V, let

$$S = \frac{\lambda_{base}}{\lambda} \text{ and } U = \frac{V_{base}}{V}$$

S and U are greater than 1 for a reduction in feature size and voltage respectively. The relevant process parameters scale, with respect to S and U, from the base 0.25μm technology as shown in Table 3.1 [31]. Note that this analysis only considers short-channel devices with a linear dependence between control voltage and saturation current.

Interconnect Assumptions

The cache uses four metal layers (excluding ground and supply wires). The metal layer assignments are shown in Table 3.2. The assumed wiring dimensions for the various layers for the base 0.25μm technology is shown in Table 3.3. As in [33], the widths of higher level metals are scaled to yield a larger cross-section to reduce their resistance. However the heights are increased only by the square root of the factor increase in the width. To mitigate the effect of decreasing wire widths in deep sub-micron processes, the aspect ratio (W/H) of each layer is scaled up by the square

Table 3.2: Metal layer Assignments

Wire Description	SRAM-Based Cache	CAM-Based Cache
Bitlines	M1	M1
Searchlines	-	M1
Wordlines	M2	M2
Pre-decoder Wires	M2	M2
Matchlines	M3	M2
Local bus	M3	M3
Global Bus	M4	M4

root of the factor change in feature size. This slows the RC delay degradation of the wires with further process shrinks.

Table 3.3: Wiring Dimensions

Layer	W_MIN(λ)	HEIGHT(λ)	PITCH(λ)
M1	3	4.5	3
M2	4	6	4
M3	4	6	8
M4	8	7	15

3.2 Methodology

In order to sustain accuracy across process generations and meet its goal of usability, the use of extracted parameters is limited to only those that may be accurately projected from one process generation to another via the general scaling rules presented in Table 3.1. For small structures such as gates, general scaling is permissible. For larger circuit networks, simple equations are used to capture the non-linear effects on energy, delay and area that result when these simple MOS structures interact. This section discusses these equations as well as the general methodology for optimizing the delay, energy and area of the cache.

3.2.1 Capacitances and Resistances

The effective capacitance and resistance of complex gates are derived from that of an inverter, whose capacitance and resistance values are in turn derived from the basic models described in this section.

Equivalent "on" Resistance of Transistors

The resistance of a transistor changes as its input voltage changes between the high and low signal levels. However, since we are usually only interested in the resistance in the region of transition where the transistor is considered "on", we use a simple model presented in [31] to derive a value for this resistance. This calculation of the equivalent resistance, R_{eq}, of a transistor is based on the simplifying assumption that the transistor is nothing more than a *switch* with a finite "on" resistance and infinite "off" resistance. The derivation of R_{eq} for the common scenario of (dis)charging a capacitor through a transistor is illustrated in Figure 3-5.

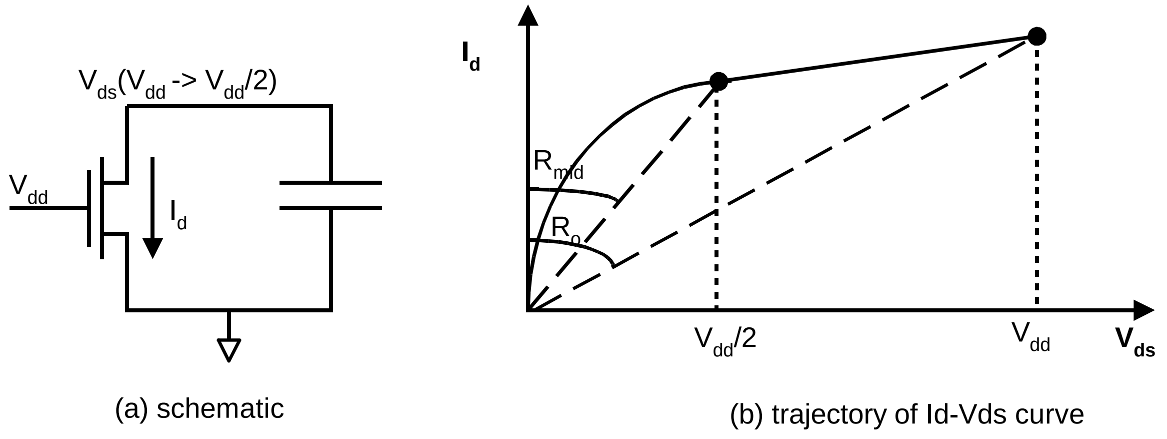


Figure 3-5: Discharging a capacitor though an NMOS

R_{eq} is computed as the average of the resistance when the voltage across it is $V_{dd}/2$ and when the capacitor is fully (dis)charged. This computation is shown in Equation 3.1.

$$R_{eq} = \frac{1}{2}\left(\frac{V_{dd}}{I_{dsat}(1+\lambda V_{dd})} + \frac{V_{dd}/2}{I_{dsat}(1+\lambda V_{dd}/2)}\right) \tag{3.1}$$

where I_{dsat} is the saturation current of the device and depends on the W/L ratio of the device, and λ is the *channel-length modulation*, which is an empirical parameter inversely proportional to the channel length. The resistance is therefore inversely proportional to the W/L ratio of the device. For our base $0.25\mu m$ technology, the equivalent resistance for an NMOS and a PMOS with a W/L ratio of 1 is 13 kΩ and 31 kΩ respectively. The resistance for larger W/L ratios is obtained by dividing these estimates by the W/L ratio.

The unit equivalent resistance of the inverter is the average of "on" resistance for the PMOS and NMOS. Based on our assumption of a 2:1 sizing ratio, the resistance for an inverter whose NMOS has a W/L of 1 is given by Equation 3.2

$$R_{inv} = \frac{R_{non} + R_{pon}}{2} = \frac{R_{eqn} + R_{eqp}/2}{2} = \frac{13k\Omega + 31k\Omega/2}{2} = 14.25k\Omega \tag{3.2}$$

The "on" resistance does not scale with process technology. However, the "on" resistance per unit width of an inverter decreases as λ shrinks.

Equivalent Capacitance of Transistors

Figure 3-6 shows the main contributions to the capacitance of the transistor. The

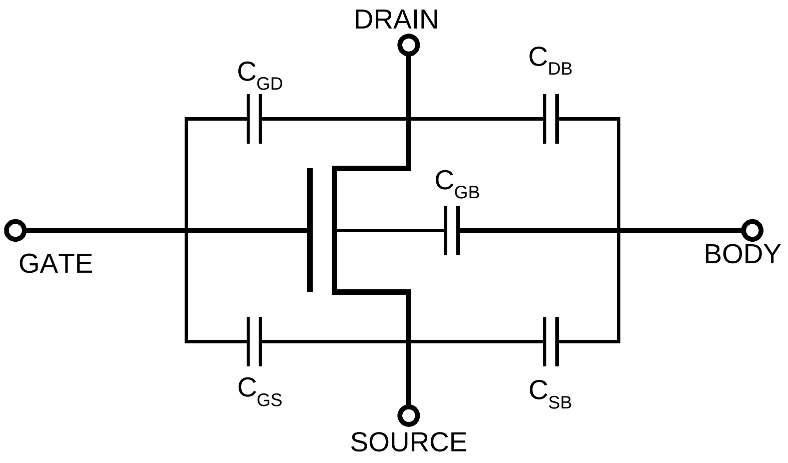


Figure 3-6: Main contributions of transistor capacitance

capacitance of the MOS device consists of gate and drain coupling capacitances. The gate capacitance of a transistor consists of 2 components, the capacitance of the channel and the capacitance of the poly-silicon line going into the gate. The value of the gate capacitance depends of whether the transistor is being used as a pass transistor, or as a pull-up or pull-down transistor. The drain capacitance depends on the transistor's geometry. Large transistors are folded to reduce their drain capacitance. The total drain capacitance of a series stack of transistors is less than the sum of their individual, unstacked drain capacitance due to contact-sharing. Detailed equations for estimating the drain and gate capacitances of transistors in different scenarios are given in [1].

Resistance and Capacitance of Complex Logic Gates

The gate or input capacitance of an inverter is estimated as the average of the gate capacitance of the pull-down and pull-up paths respectively. A similar method is used to estimate its drain capacitance.

The gate capacitance of more complex gates is estimated from that of an inverter using the concept of logical effort introduced in [12]. The logical effort of a gate, which depends only on its topology, captures the complexity of the gate relative to an inverter whose pull-up and pull-down paths are sized in the same ratio as the gate's. This is the extra effort the complex gate exerts due to the logical function it implements. Since the gate is sized to have equal rise and fall times, a single size parameter, w, which is the size of the NMOS transistor in an equivalent inverter having the same output resistance, is used to represent the gate [6]. If C_g is the input capacitance per unit width and R_g is the output resistance per unit width of an inverter, then the output resistance of the gate, R_{gate}, and input capacitance, C_{gate}, are given by Equations 3.3 and 3.4

$$R_{gate} = R_g/w. \tag{3.3}$$

$$C_{gate} = C_g \times 3w \times le \tag{3.4}$$

where le is the logical effort of the gate.

Table 3.4: Logical effort and intrinsic delay of common 2:1 sized gates

Gate Type	Logical Effort	Intrinsic Delay
inverter	1	p
n-input NAND	$\frac{(n+2)}{3}$	np
n-input NOR	$\frac{(2n+1)}{3}$	np
n-way MUX	2	$2np$
2-input XOR	4	$4p$

The drain capacitance of the gate may be similarly derived from that of an inverter as the product of the drain capacitance of an inverter and the number of transistors connected to the gate's output node. The drain capacitance is the main contributing factor to the gate's parasitic delay. Hence, the parasitic delay of a gate may be derived from that of an inverter (sized in the same ratio) using the same scaling constants for relative drain capacitance. Table 3.4 lists the logical effort and intrinsic delays of common gates used in the cache. Here, p is the intrinsic delay of an inverter given by R_gC_g.

The logical effort of a gate is a strong function of the ratio-sizing between the pull down and pull-up paths. To this extent, it is possible, and often favorable, to skew the gates on the critical path to improve the logical effort. Due to velocity saturation, the logical effort of short-channel devices will be slightly smaller than given in Table 3.4. These effects are not modeled in ZOOM.

Interconnect RC

Interconnect resistance and capacitances are estimated using the Berkeley Predictive Technology Model for interconnect [34]. The resistance, R_w, of a wire of length l, width w and thickness t is given by Equation 3.5, where ρ is the resistivity of the metal. $\rho = 2.2\mu\Omega$-cm for copper and $\rho = 3.3\mu\Omega$-cm for aluminum.

$$R_w = \frac{\rho}{t} \cdot \frac{l}{w} \tag{3.5}$$

The main contributions of metal capacitance are the area and fringe flux to the underlying plane, and the coupling capacitance with nearby wires (Figure 3-7 (a)). The total capacitance for a wire of length l, width w, thickness t, separated by

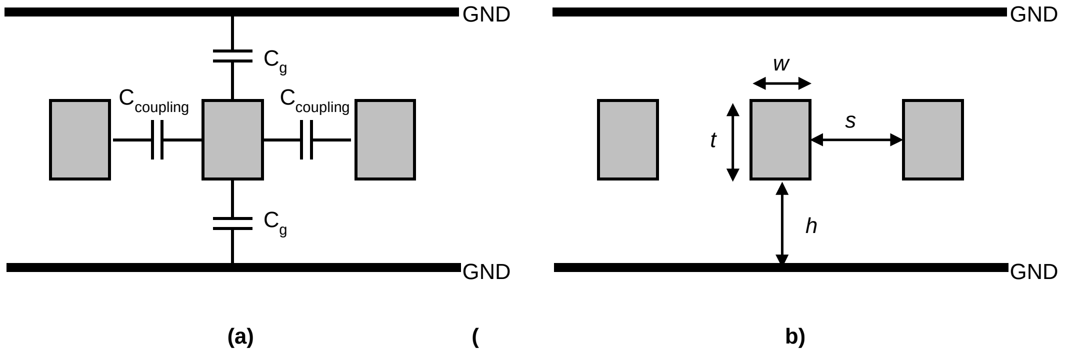


Figure 3-7: (a) Wire capacitance model (b) Wire dimensions

a distance h from the underlying planes and a distance s from neigboring wires (Figure 3-7 (b)) is given by the following set of equations:

$$C_{total} = 2C_g + 2C_c \tag{3.6}$$

where

$$C_g = \epsilon \cdot l[\frac{w}{h} + 2.217(\frac{s}{s+0.702h})^{2.193} + 1.171(\frac{s}{s+1.510h})^{0.7642} \cdot (\frac{t}{t+4.532h})^{0.1204}] \quad (3.7)$$

$$C_c = \epsilon \cdot l[1.41\frac{t}{s}exp(-\frac{4s}{s+8.014h}) + 2.37(\frac{w}{w+0.31s})^{0.257}(\frac{h}{h+9s})^{0.76}exp(-\frac{2s}{s+6h})] \quad (3.8)$$

The formulae in Equations 3.7 and 3.8 are hopelessly complicated and ill-suited to manual analysis/optimization. However, a cursory analysis shows that the main drivers of the fringing and area capacitances are the wire widths, w and the distance between layers, h. C_g increases with increasing w and decreasing h. The coupling capacitance is inversely proportional to the separation between same-layer wires. It also worsens with increasing wire thickness, while the metal resistance improves with increasing wire thickness. The wire dimensions therefore have to be carefully determined to optimize the RC delay of the interconnect.

3.2.2 Delay Estimation

There are three main components of delay in CMOS gates: the extrinsic delay due to the external load, the short-circuit delay due to non-zero input rise and fall times, and the intrinsic delay due to the junction capacitance of the gate.

$$D_{total} = D_{extrinsic} + D_{short-circuit} + D_{intrinsic} \quad (3.9)$$

Extrinsic delay is a strong function of fanout (the ratio of the output load to the input

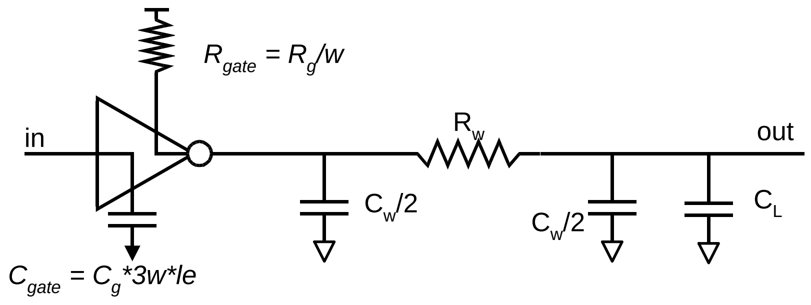


Figure 3-8: Extrinsic delay of a logic gate driving a load through an interconnect line

drive). The extrinsic delay of a logic gate of size w driving a load C_L through a wire of resistance R_w and capacitance C_w (Figure 3-8) is estimated by using the simple approximation proposed by Elmore in [36], which is summarized in Equation 3.10.

$$D_{extrinsic} = ln(2) \cdot \left(R_{gate} \cdot (C_L + C_w) + R_w \cdot (\frac{C_w}{2} + C_L)\right) \tag{3.10}$$

The total RC time constant is multiplied by $ln(2)$ because we are interested in the 50% rise or fall time of the gates.

Short-circuit delay results from non-zero input rise and fall times which create a direct path between V_{dd} and ground for a short period of time during switching. Short-circuit currents are globally minimized when both the input and output rise and fall at the same rate [31]. Since the overall impact of short-circuit delay is small in well-designed circuits, $D_{short-circuit}$ is approximated as 10% of the switching delay in all cases where the input/output rise and fall times can be kept equal to within a constant (as in the case of buffer/logic chains). Short-circuit currents have a more pronounced impact on delay in the case of slow rising inputs such as the wordlines at the input of the bitline pass transistors. This special case will be discussed in the next chapter when the bitlines are modeled.

The intrinsic delay of the gate is estimated in Table 3.4 (on page 39) in Section 3.2.1.

3.2.3 Repeaters

The most popular design approach to reducing the propagation delay of long wires is to insert intermediate buffers, or *repeaters*, in the wire. Analytical equations for minimizing the delay of a wire using repeaters are derived in Chapter 9 of [31]. This section summarizes their analysis, derives equations for the path energy and presents the methodology for trading path delay for reduced path energy in ZOOM.

The delay of an interconnect line with a unit capacitance c, unit resistance r and length L is given by

$$T = \frac{rcL^2}{2} \tag{3.11}$$

Making an interconnect line m times shorter reduces the delay by a factor of m^2 and is sufficient to offset the extra delay of the repeaters needed to drive the segments. Since the delay of the repeaters is a function of the load, the interconnect delay may be further enchanced by sizing the repeaters. The delay of the interconnect chain is obtained by modeling the repeater as an RC network and, by using the Elmore delay [36] approach. With R_d and C_d the resistance and capacitance respectively, of a minimum-sized inverter, the delay is given by:

$$t_p = m\left(0.69 \cdot \frac{R_d}{s}\left(2sC_d + \frac{cL}{m}\right) + 0.69 \cdot \left(\frac{rL}{m}\right)(sC_d) + 0.38rc\left(\frac{L}{m}\right)^2\right) \tag{3.12}$$

where s is the repeater sizing factor and we have assumed an inverter self-loading factor of 1. The optimum sizing factor, s_{opt}, optimum number of stages, m_{opt}, and critical wire length, L_{crit}, for delay minimization are determined by partially differentiating (3.12) with respect to s and with respect to m.

$$m_{opt} = L\sqrt{\frac{0.38rc}{0.69 \cdot 2 \cdot R_d C_d}} = \sqrt{\frac{t_{pwire(unbuffered)}}{t_{pinv}}} \tag{3.13}$$

$$s_{opt} = \sqrt{\frac{R_d c}{r C_d}} \tag{3.14}$$

$$L_{crit} = \frac{L}{m_{opt}} = \sqrt{\frac{t_{pinv}}{0.38rc}} \tag{3.15}$$

where $t_{pinv} = 0.69 \cdot 2 \cdot R_d C_d = 2p$ is the delay of an inverter with a fanout of 1 and intrinsic delay p. The delay of a segment of critical length is always given by

$$t_{crit} = \frac{t_{pinv}}{m_{opt}} = 2\left(1 + \sqrt{\frac{0.69}{2 \cdot 0.38}}\right) t_{pinv} = 3.9 t_{pinv} = 7.8p$$

and is independent of the routing layer. Inserting repeaters to reduce the wire delay only makes sense if the wire is at least twice as long as the critical length.

The energy of the path is half the sum of the energies of the wire segments and repeaters since only half of the segments and repeaters in the chain can make a low-to-high transition.

$$\begin{aligned} E_{path} = E_{segments} + E_{repeaters} &= \frac{1}{2}V_{dd}^2 \cdot \left(\frac{1}{2}(cL) + \frac{1}{2}\left(C_d \sum_{i=0}^{m-1} s^i\right)\right) \\ &= \frac{1}{4}V_{dd}^2 \cdot \left(cL + C_d \cdot \frac{1 - s^m}{1 - s}\right) \end{aligned} \tag{3.16}$$

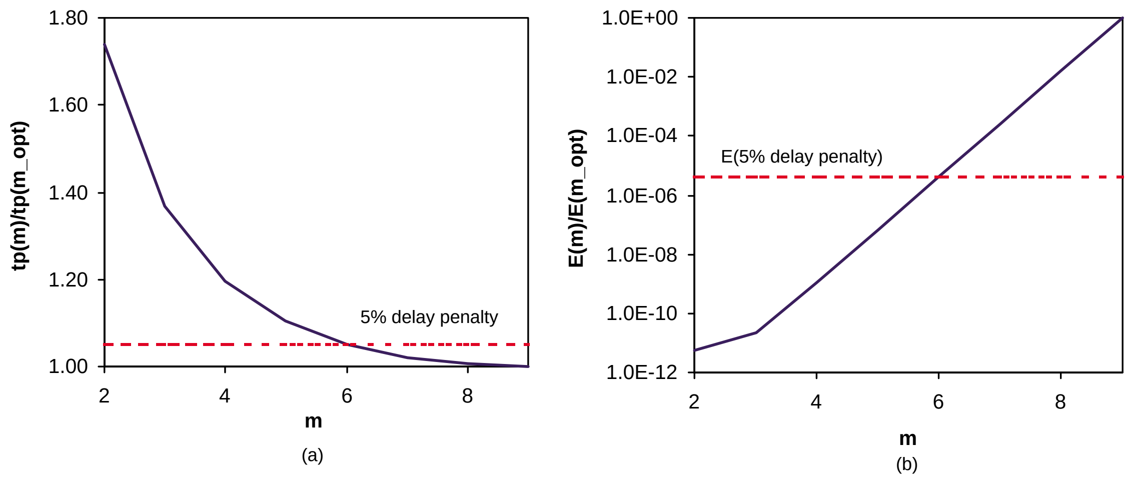


Figure 3-9: (a) Normalized delay of 3 cm Al-1 wire for varying number of repeaters. Dashed line shows the point at which the delay comes within 5% of the minimum. (b) Normalized energy of 3 cm Al-1 wire for varying number of repeaters. Dashed line shows the factor reduction in energy for the value of m whose delay comes within 5% of the minimum.

Absolute delay minimization leads to prohibitive energy levels for long wires. Equation (3.16) shows that the energy consumed by the repeaters grows exponentially with m, while (3.12) shows the delay benefit of increasing m is subject to diminishing returns. For example, minimizing the delay of a 3 cm Al-1 wire in a typical 0.25-μm CMOS technology with $p = 16$ ps, $R_d = 7.8$ kΩ, $C_d = 3$ fF, $c = 110$ aF/μm and $r = 0.075$ Ω/μm requires 9 repeaters with a sizing factor, $s = 62$. This reduces the delay from 2.82 ns to 1.18 ns. However, the energy of the interconnect line increases from 10pJ to 1J! The delay for the line, if 8 repeaters were used instead, would be 1.19 ns and the energy would drop to 16mJ, which is still prohibitive, but orders of magnitude below the energy for minimum delay with less than a 1% delay penalty. Figures 3-9 (a) and 3-9 (b) plot the normalized path delay and normalized path energy respectively for various values of m. As shown by the dashed lines in the two graphs,

$m = 6$ brings the the delay to within 5% of the optimum while reducing the energy by five orders of magnitude. Hence, rather than inserting repeaters to minimize the wire delay, ZOOM formulates the problem as one of inserting repeaters to bring the delay to within some percentage of the minimum. This percentage is set by the *EDA Optimizer* which will be discussed in Section 3.2.6. Setting m to a value less than the optimum increases the critical length for repeater insertion for a given metal layer. The scaling factor, s may also be adjusted to reduce energy consumption. However, the delay is more sensitive to s than m for the tolerable trade-off percentage ranges, and the energy benefits of reducing s are less phenomenal.

3.2.4 Energy Estimation

There are 3 main sources of energy dissipation in CMOS circuits: dynamic switching energy, dissipation from short-circuit currents and energy loss due to leakage currents.

Dynamic Switching Energy

This is the energy consumed in switching the output capacitances. It is the largest source of energy dissipation in CMOS circuits. The dynamic switching energy is computed as:

$$E_{switching} = \alpha \cdot \frac{1}{2} C V_{dd} \cdot \delta V \tag{3.17}$$

where

$\alpha =$ the activity factor (assumed to be 0.5 in ZOOM),

$C =$ the effective capacitance of the output load, and

δV = the voltage swing of the output node

The switching energy may be reduced by reducing the supply voltage and the switching activity in the cache. The latter is exploited by circuit innovations that limit the swing on high-C lines to some small δV and use sensing techniques on low-C sense amplifiers to achieve full-rail results.

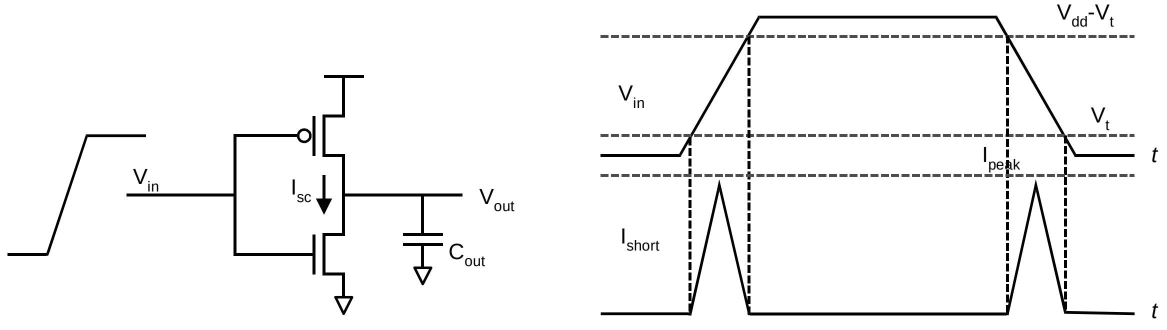


Figure 3-10: Short-circuit currents during transients (from [31])

Dissipation from Short-Circuit Currents

The finite slopes of input signals causes direct-path currents to flow through the gate for a short time during switching (Figure 3-10). The energy consumed per switching is computed in [31] as

$$E_{dp} = V_{dd}\frac{I_{peak}t_{sc}}{2} + V_{dd}\frac{I_{peak}t_{sc}}{2} = t_{sc}V_{dd}I_{peak} \tag{3.18}$$

where t_{sc} is the time both nMOS and pMOS devices are conducting and I_{leak} is determined by the saturation current of the devices. Like short-circuit delay, the short-circuit energy is minimized when the input and output signals have equal rise and fall times. In [31], Rabaey *et al.* show that when the rise and fall times of input and output signals are matched, most of the energy is associated with dynamic switching and only a minor fraction (less than 10%) is devoted to short-circuit currents. Hence, in the first phase of its implementation, ZOOM estimates short-circuit energy as 10% of the dynamic switching energy.

Energy Loss Due to Leakage Currents

This is the steady-state energy dissipation in an idle circuit due to sub-threshold currents flowing through the drain when $V_{gs} < V_t$. Idle-mode leakage energy can be estimated as

$$E_{leakage} = (1 - \alpha)V_{dd} \cdot I_{leakage} \cdot \frac{1}{f} \tag{3.19}$$

where $I_{leakage}$ is the sub-threshold leakage current of the device under zero-bias at room temperature, and f is the operating frequency. Sub-threshold currents increase exponentially with decreasing threshold voltage and increasing temperature.

3.2.5 Area Estimation

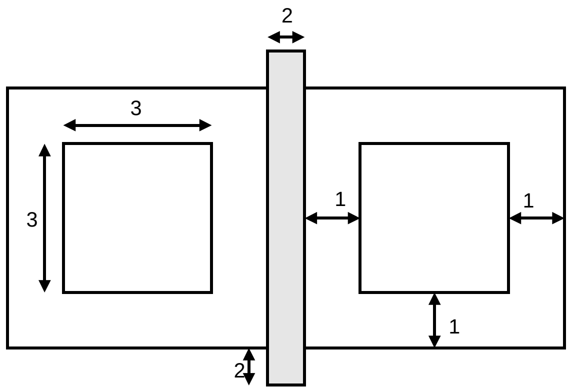


Figure 3-11: Summary of lambda-based design rules (in units of λ)

Lambda-based layout rules (Figure 3-11) are used to obtain an estimate of the area of various components of the cache in order to obtain accurate wire lengths for delay and energy estimation. The area of the memory cells are estimated from the bit-height and bit-width values and may be modified by the program user.

3.2.6 Energy-Delay-Area Optimization

The design targets for a given system often have stringent delay requirements. For instance, the access time of the primary cache may be strictly required to fit in one or two pipeline cycles of fixed duration. The goal of energy-delay-area (EDA) optimization is to achieve this minimum delay requirement at the lowest cost possible to energy and area. The optimization criteria, which is supplied by the program user, consist of the following:

- D_{target}: the target cache access time
- δt: the *tolerance*, which specifies the maximum allowed percentage deviation from the target delay
- β: an integer specifying the weight attached to energy minimization, the bigger the value of β, the lower the energy of the proposed optimum model;
- α: an integer specifying the weight attached to area minimization, the bigger the value of β, the lower the area of the proposed optimum model.

Let D, E and A be the access time, access energy and area, respectively, of a given cache configuration. The general optimization scheme can be summarized as follows:

For $i = 1$ to $i =$ number of possible cache configurations,

$opt\_cache = cache_i$ if

$$\left(\frac{|D_{target} - D_i|}{D_{target}} \leq \delta t\right) \bigwedge \left((E_i^{\beta} \cdot A_i^{\alpha}) < (E_{opt\_cache}^{\beta} \cdot A_{opt\_cache}^{\alpha})\right)$$

The degrees of freedom used in optimization include transistor-sizing via the global fanout, array partitioning, interconnect optimization strategies (via repeater critical segment modifications) and other bitline optimizations strategies. We discuss these in Chapter 4 when the component models are presented.

There are the special cases when the specified target delay is either too large or too small, so that the optimizer is not able to find a configuration that meets the optimization criteria, given the base assumptions made about the cache structure. To improve usability, (especially in cases where the designer simply wants the minimum delay ZOOM can find), the optimizer uses a "best-effort" approach to return the closest match. If D_{target} is too large, the configuration that minimizes the objective function, $E^{\beta}A^{\alpha}$, is returned as the optimum. If the delay is too small, the optimizer returns the configuration with the minimum delay. There is however no guarantee on the energy or area efficiency of the proposed cache configuration.

Chapter 4

Micro-Architecture Level Cache Modeling

Chapter 3 presented the general methodolgy used and assumptions made in modeling the cache. This section presents component-specific circuit-level models used in the micro-architecture level simulator.

For the purposes of modeling, the cache access path is broken into four main components: the data path (SRAM and CAM arrays), the decoder, the comparators and the local and global buses. The accuracy of the energy of the modelas are verified using HSPICE® simulations. The HSPICE® netlists used in these simulations were extracted from ZOOM using the built-in Netlist Generator. Where necessary, interconnect capacitances and resistances are modeled as distributed RC networks at bit granulity. All circuits are set in the TSMC 0.25μm technology with a supply voltage of $2.5V$. The simulations were conducted under typical conditions at room temperature.

4.1 Decoder

The decoder is the starting point of the access. A cache of size C bytes and associativity A, using B byte data blocks has $S = C/(B \cdot A)$ sets, and the decoder has to decode $log(S)$ bits to select the set to which the requested word belongs. The decoder consists of the *global pre-decoder*, which is shared by all the sub-arrays within an array, and the *row decoder*, which is local to each pair of sub-arrays (Figure 4-1). The global pre-decoder selects one set of sub-arrays to enable for the access while the row decoder selects the appropriate row within each sub-array. As discussed in Chapter 3, the data and tag arrays in an SRAM cache have separate decoders. Due to its organization, only one global pre-decoder is needed in a CAM cache. The decoder design problem is two-fold: determining the optimal decode structure and finding the optimal number of stages and sizing for each level.

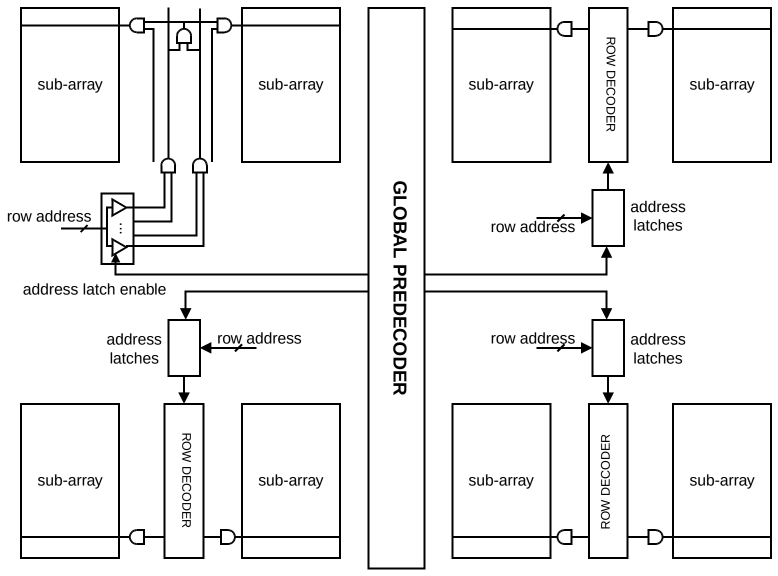


Figure 4-1: SRAM partitioning example with 8 sub-arrays

4.1.1 Decoding Structure

A simple NAND-INVERTER decoding structure using 2-input NAND gates is assumed. To implement an r-to-2^r decoder, two sets of $\left(\frac{r}{2}\right)$-to-$2^{r/2}$ decoders are first implemented and the outputs combined in the final stage. Figure 4-2 illustrates the assumed decoding structure using the example of a 4-to-16 decoder. The split decoder approach results in cheaper, reduced energy and often faster implementations.

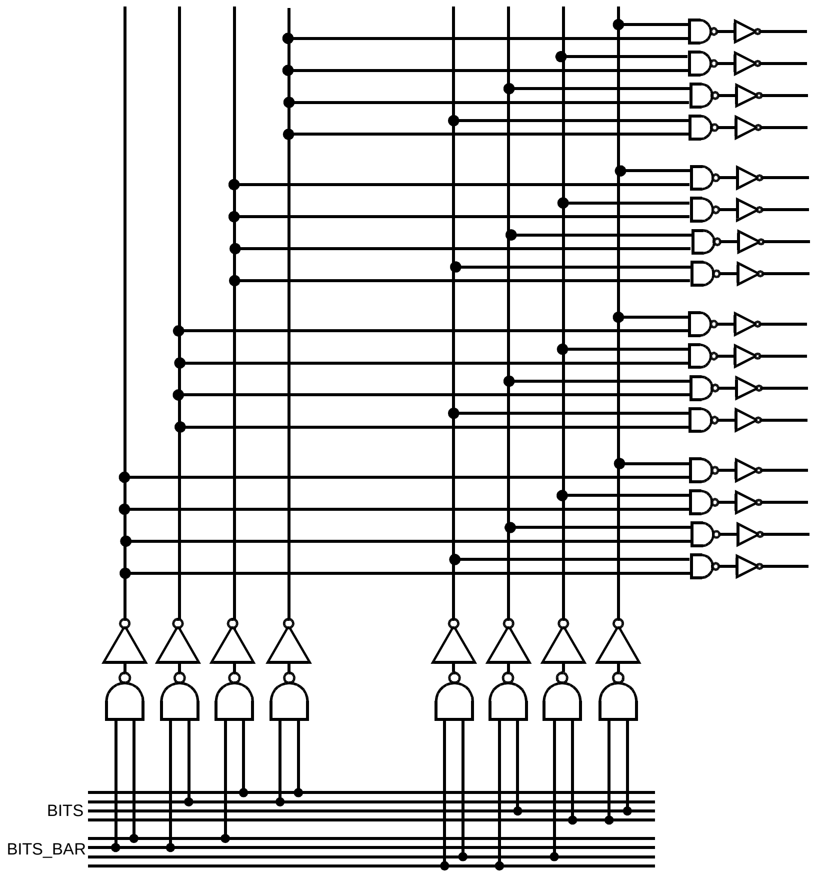


Figure 4-2: Decoding structure for a 4-16 decoder

The NAND-INVERTER implementation implies that each AND stage is expanded to 2 stages, which could result in an excessive number of stages. However, since the decoder often has to drive a large load on its output, extra buffers are often needed to improve its delay. In physical implementations, the number of stages needed to drive the external load is often larger than the number of stages needed to implement the decoding function using 2-input gates.

Another advantage of using smaller gates is that the overall logical effort of the decode path is reduced. The logical effort of a NAND gate increases linearly with the number of inputs while that of a chain of gates increases logarithmically with the number of inputs (see Figure 4-3).

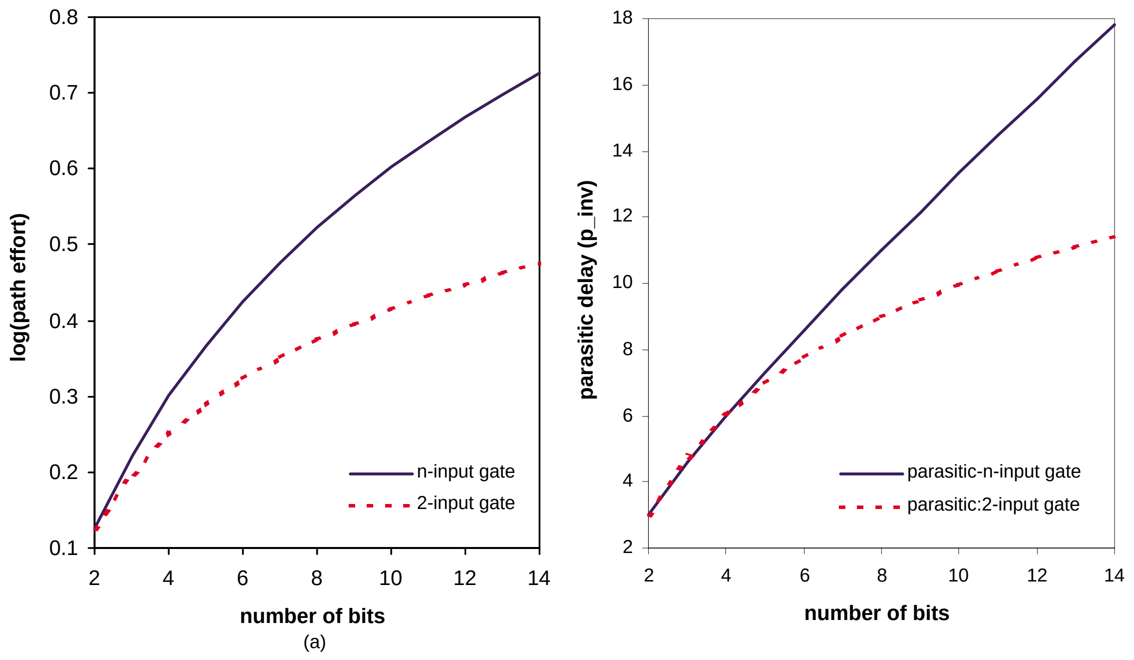


Figure 4-3: (a) Growth rate of path effort of n-input NAND implementation using one stage of an n-input NAND gate *vs.* multiple stages of 2-input NANDs and inverters. (b) Parasitic delay of n-input NAND implementation using n-input NAND gate *vs.* multiple stages of 2-input NANDs and inverters for a fixed number of stages.

4.1.2 Sizing the Decoder

The problem of optimally sizing the decoder is similar to the well-understood problem of optimally sizing a chain of inverters [38] - [41]. The key features of the decode path which distinguish it from a simple inverter path are the presence of logic gates other than inverters, branching at some of the intermediate nodes and the presence of interconnect inside the path. Figure 4-4 shows the critical path for the assumed decoding structure.

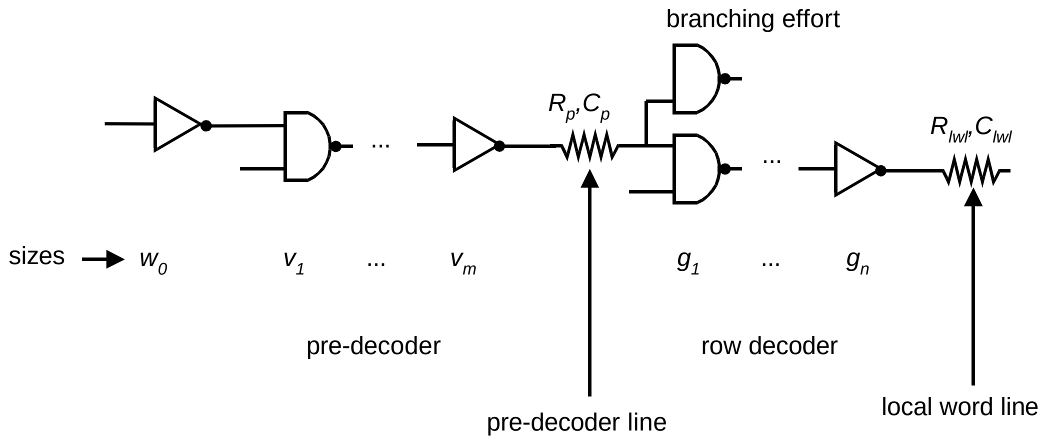


Figure 4-4: Critical Path of Decoder

Logic gates and branching are easily handled using the concept of logical effort introduced in Section 3.2.2. In general, for an r to 2^r decode, the total branching effort of the critical path from the address input to the output is 2^{r-1} since each address driver drives half the outputs. The total branching for the row decoders in ZOOM is 2^r since each decoder is shared by 2 sub-arrays, each of which contains 2^r rows. Ignoring interconnect capacitances and resistances, the theoretical optimal stage effort, f, and optimal number of stages, n, is given by Equation 4.1

$$f^n = \frac{C_{outputload} \cdot 2^{r-1} \cdot \text{LogicalEffort}(r\text{-input-AND})}{C_0} \tag{4.1}$$

Intermediate path interconnect is handled by independently sizing the sub-chains before and after the interconnect as in [8]. The interconnect and the first gate of the combine stage are treated as loads to the pre-decode stage. The optimal fanout for the decoder is set by the user-specified optimization biases by iteratively increasing the number of stages from the number of stages required to implement the logical function to some fixed maximum. For delay and energy-delay product minimization, the optimum fanout is about 4 in most cases. For energy minimization, the optimal fanout is higher, at around 6-8, depending on the maximum fanout allowed in ZOOM. These trends are very much in line with theoretical results reported in [38] - [41] and those derived in [8] where the optimal number of stages for energy minimization was found to be 1 (i.e the maximum stage fanout).

For a decoder with n stages preceeding the wordline driver and an effective stage fanout f, if τ_{inv} is the intrinsic delay of an inverter, then the delay of the decode path is given by:

$$D = n \cdot f \cdot \tau_{inv} + parasitics + D_{wire} \tag{4.2}$$

where the parasitic delays for the gates are given by Table 3.4 on page 39. The wire delay, D_{wire}, is the delay of the pre-decoded line before the final NAND stage. It is given by $0.38 R_{wire} C_{wire}$ where 0.38 is the simulated co-efficient for the 50% rise time of a wire [31], R_{wire} is the resistance of the interconnect line and C_{wire} is the sum of the capacitance of the wire and the loading from the row-select NAND gates distributed evenly along the wire.

An activity factor of 0.5 is assumed in estimating the energy of the decoder. The capacitances of the gates and wires are added together to obtain the total capacitance of the decoder and are used to estimate the energy. Since energy is only consumed for low to high transitions, only half the total capacitance of the inverter chain need to be included in the decoder capacitance.

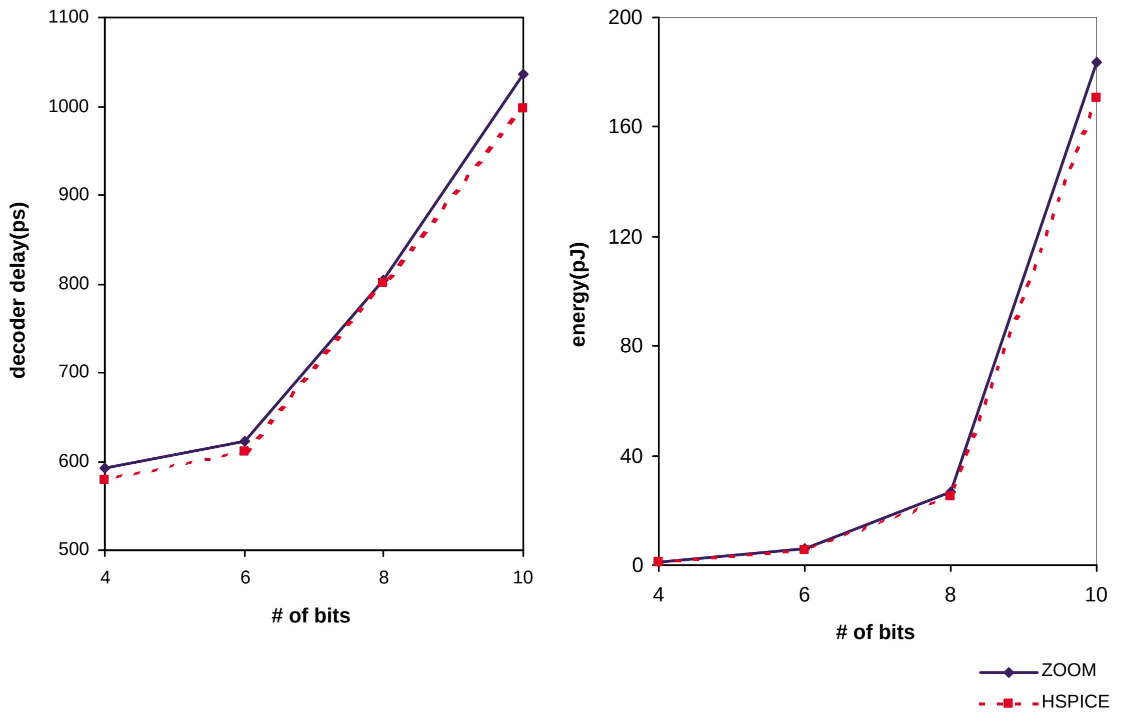


Figure 4-5: Comparison of decoder delay and energy estimates with HSPICE simulated values

Figure 4-5 compares the estimated decoder delay and energy with those of HSPICE for the same decoding structures. The delay and energy estimates are within 5% and 8%, respectively of HSPICE simulated values.

The increase in area due to the row decoders is the sum of the area occupied by $2 \cdot 2^{r/2}$ vertical pre-decode tracks, $2 \cdot 2^{r/2}$ gates for each of the $\frac{r}{2}$-input logic groups and $2 \cdot 2^r$ final NAND gates and wordline drivers. This area is multiplied by half the number of sub-arrays to obtain the total area contribution of the row-decoders. A similar approach is used to calculate the area of the global pre-decoders.

4.1.3 Managing Interconnect Delay in the Decoder

The interconnect delay quickly dominates the delay of the decoder as the number of sub-arrays or rows increase. Due to compactness and pitch-matching requirements, the delay degradation in the row decoder is handled by increasing the number of stages before the wire. Wire delays are even more pronounced in the sub-array-select decoders which have to drive long wires to each of the local decoders. Since there are no strict pitch-matching or compactness requirements, the degraded wire delays in the sub-array-select decoder are handled by inserting a cascade of sized-up, equally spaced repeaters in the wires. This has the effect of linearizing the wire delay, which would otherwise increase quadratically with the wire length. The methodology for sizing and calculating the delay of the wire segments is explained in Section 3.2.3. Figure 4-6 compares the wire delays of the decoder before and after the insertion of repeaters.

4.1.4 Wordline Delay and Energy

The wordline is really the final stage of the decoder. However for analysis purposes, it is beneficial to treat it as a separate entity. The wordline consists of an inverter (the wordline driver) and an interconnect line uniformly loaded with SRAM pass

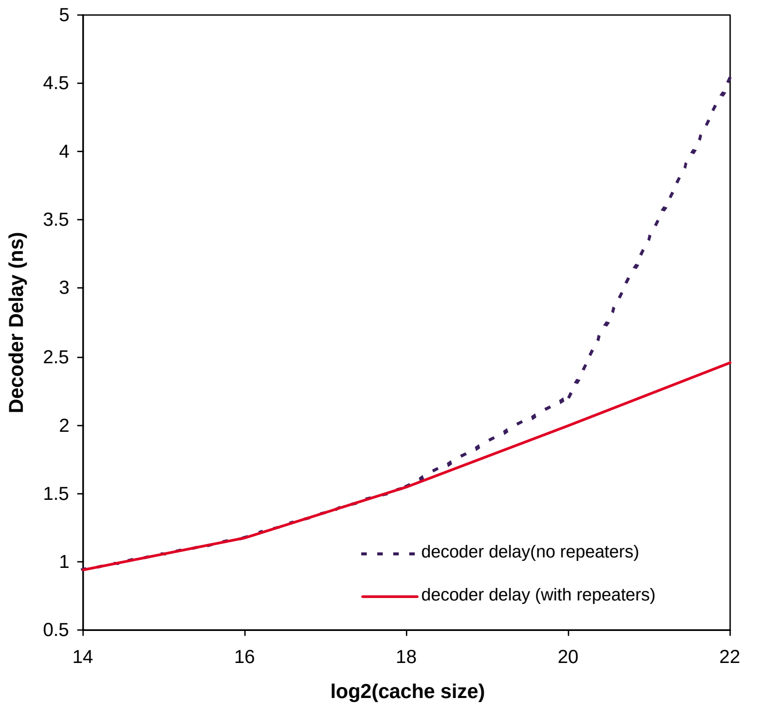


Figure 4-6: Decoder delay before and after repeater insertion

transistors. The interconnect line traverses the width of the SRAM (sub)array and connects to the gates of each pass transistor in the SRAM cells on the row. When the wordline goes 'high', all the pass transistors are activated and connected to their respective bitlines (Figure 4-7). The 50% rise time of the wordline is given by:

$$T_{wordline} = \ln(2) \cdot \left(R_{driver} \cdot (C_{driver} + C_{wire} + 2nC_{pass}) + R_{driver} \cdot \frac{C_{wire} + 2nC_{pass}}{2} \right) \tag{4.3}$$

where

R_{driver}, C_{driver}= the resistance and capacitance respectively of the wordline driver,

R_{wire}, C_{wire}= the resistance and capacitance respectively of the interconnect line,

n = the number of bitline columns in the array, and

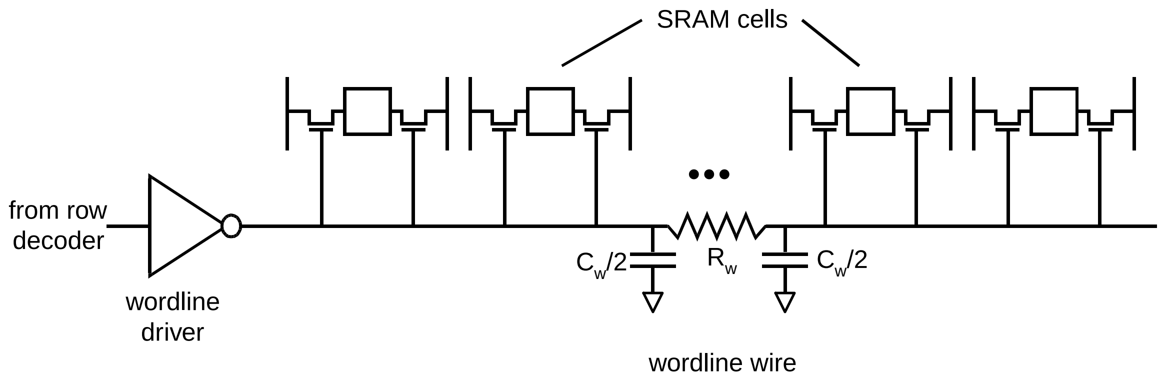


Figure 4-7: Wordline Architecture

C_{pass} = the gate capacitance of one SRAM pass transistor

Since only one wordline makes a low-to-high transition, the energy of the wordline is simply

$$\frac{1}{2} \cdot {V_{dd}}^2 \cdot (C_{driver} + C_{wire} + 2nC_{pass})$$

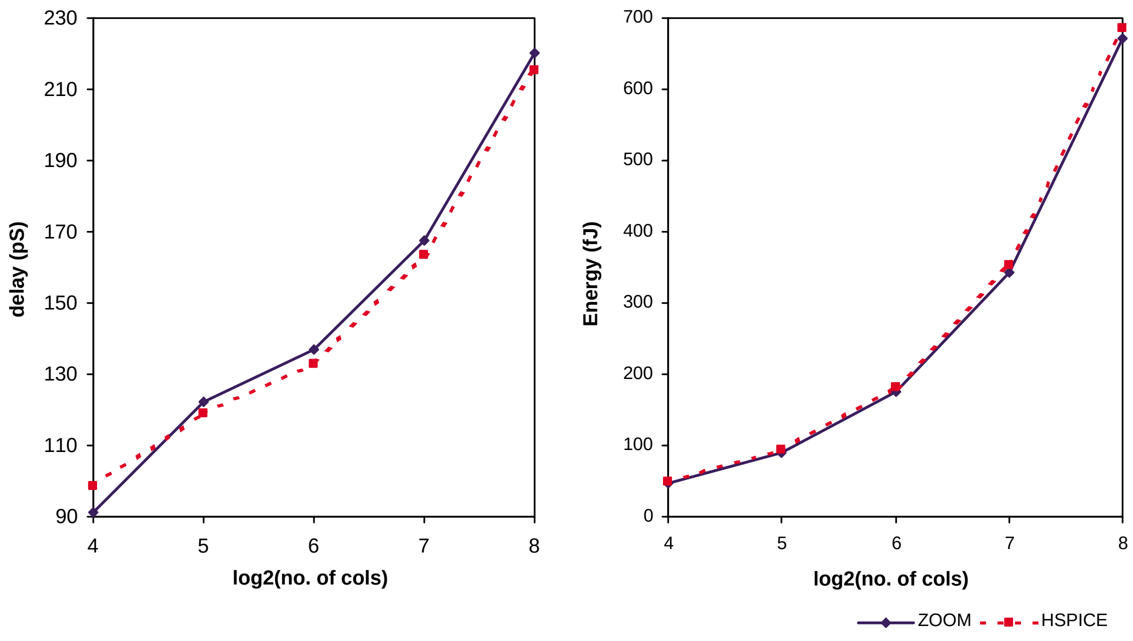


Figure 4-8: Comparison of wordline delay and energy estimates with HSPICE simulated values

Figure 4-8 compares the estimated wordline delay and energy for a range of column widths with those of HSPICE. To model the interconnect line in HSPICE, the total wire capacitance and resistance obtained from the Berkeley Predictive Technology Models [34] were divided into n pieces and connected between the pass transistors in the SRAM cells. The delay and energy estimates are both within 3%, of HSPICE® simulated values.

4.2 SRAM Data Path

The SRAM data path is made up of SRAM cells and other peripheral circuits such as wordline drivers, precharge, column multiplexors and sense amplifiers that make it possible to access the bits stored on the cells. The SRAM array consists of a matrix of 2^m rows by 2^n word-columns of memory cells. Figure 4-9 shows the basic structure of an SRAM array. Each memory cell contains a pair of cross-coupled inverters which form a bi-stable element. These inverters are connected to a pair of bitlines through nFET pass transistors, which provide differential read and write access. The $m + n$ address bits, which identify the cells to be accessed, are split into m row address bits and n column address bits. The row decoder activates one of the 2^m wordlines which connects the memory cells of that row to their respective bitlines. The column decoder sets a pair of column switches, which connects one of 2^n word columns to the peripheral circuits. The local wordlines in the RAM are pulsed to limit the bitline swing [37].

The bitlines are pre-charged to some reference voltage before each access to the

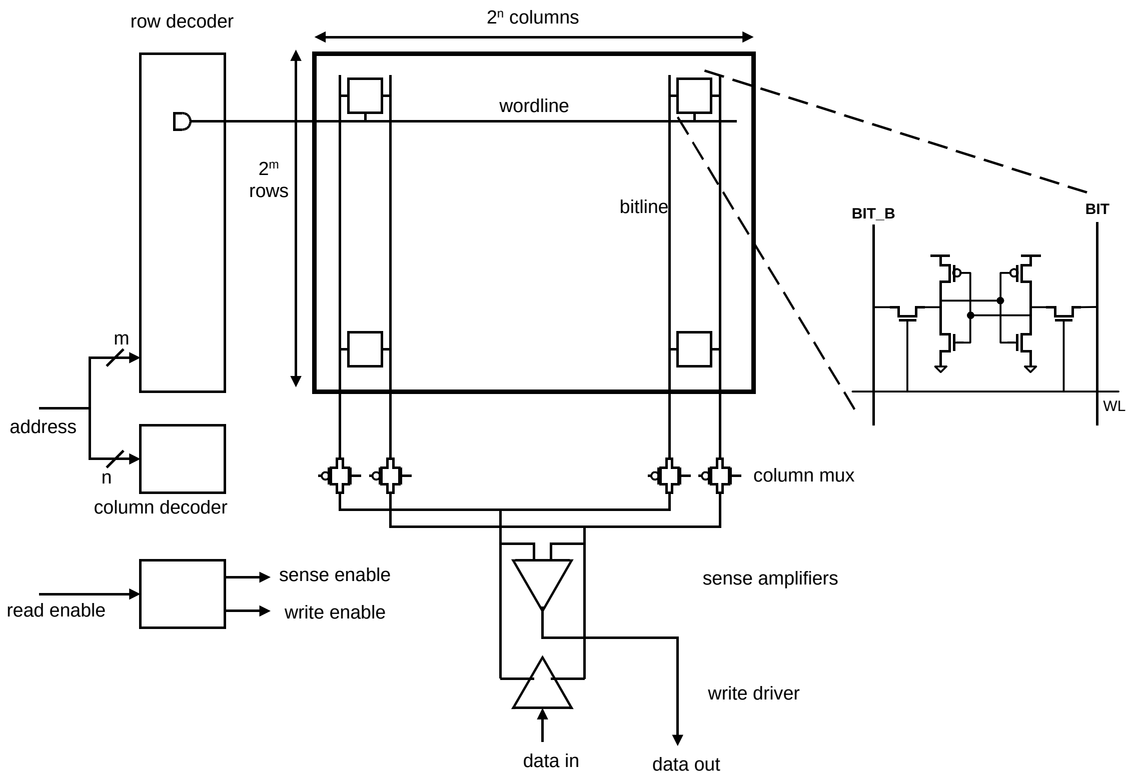


Figure 4-9: Basic structure of SRAM array (from [8])

SRAM array. During a read operation, the wordline goes high, causing the access nFET connected to the cell node storing a '0' to start discharging the bitline, while the complementary bitline remains in its pre-charged state. This results in a differential voltage drop across the bitline pair. Since SRAM cells are optimized to minimize the cell area, their cell currents are very small, resulting in a slow bitline discharge rate. To speed up the RAM access, sense amplifiers are used to amplify the small bitline signal to a valid value.

During a write operation, the data to be written is transferred to the desired columns by driving them onto the bitline pairs by grounding either the bit line or its complement. If the cell data is different from the write data, then the '1' node is discharged when the access nFET connects it to the discharged bitline, thus causing the cell to be written with the bitline value.

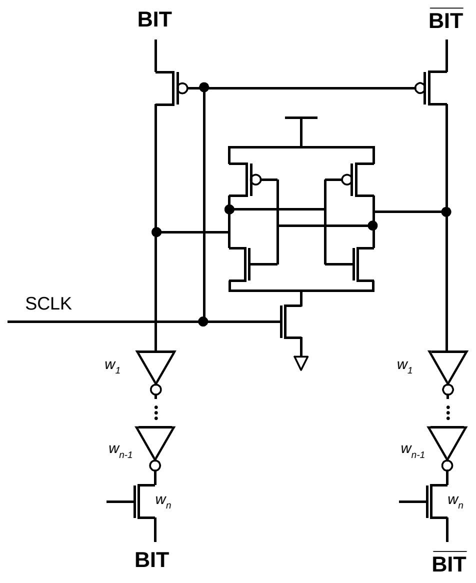


Figure 4-10: Sense amplifier circuit

Figure 4-10 shows the assumed sense amplifier structure. There is a wide variety in the types of sense amplifier circuits used in SRAMs. A latch-style differential voltage sense amplifier with perfect timing control is assumed because it provides the fastest implementation among known sense amplifier circuits and can function over a wide range of voltages. Since they are inherently clocked, they also consume very low power. The pair of cross-coupled gain stages in the sense amplifier are activated in their meta-stable state by a clock after an adequate voltage gap is created. Positive feedback quickly forces the output to a stable operating point. The output is then buffered and driven onto the gates of the output nFET drivers which create a small voltage differential at their outputs by discharging previously precharged datalines. The sense amplifier structure for the tag array differs from that of the data array only in the final stage, where the output is connected to the gate inputs of nFETs in the tag comparators rather than drain inputs of dataline drivers as shown in Figure 4-10.

Two main factors affect the sensitivity and, ultimately, the delay of the sense amplifier: the non-scaling mismatch voltage and noise on the supply voltage. The allowed voltage swing on the bitlines has to be larger than both of these fluctuations combined. Muzino *et al.* show in [35] that the dominant source of threshold variations in closely spaced transistors in deep sub-micron technologies is the random fluctuations in the channel dopant concentrations. This portion of the threshold mismatch remains constant with process scaling and has an estimate of about 50mV [8] for the assumed sense amplifier structure. The bitline swing adequate for sensing for a given supply voltage is computed as

$$\delta V = \left(\frac{V_{dd}}{20}\right) \cdot 1.25 + 50mV$$

subject to a minimum of $100mV$. This computation assumes a margin of 25% of the optimal swing to allow for operating point fluctuations. For a supply voltage voltage of $2.5V$, the allowed bitline swing evaluates to $205mV$, which is enough to ensure that the sense amplifiers operate correctly over a wide range of temperature and other threshold voltage fluctuations [6].

4.2.1 Data Path Delay and Energy

Figure 4-11 shows the schematic of the SRAM data path. The delay of the data path is the time it takes for the output of the sense amplifiers located at the bottom of the array to reach $V_{dd}/2$ after the wordline reaches $V_{dd}/2$. This includes the delay through all the the elements on the path: SRAM cells, column multiplexors, isolation transistors and sense amplifiers.

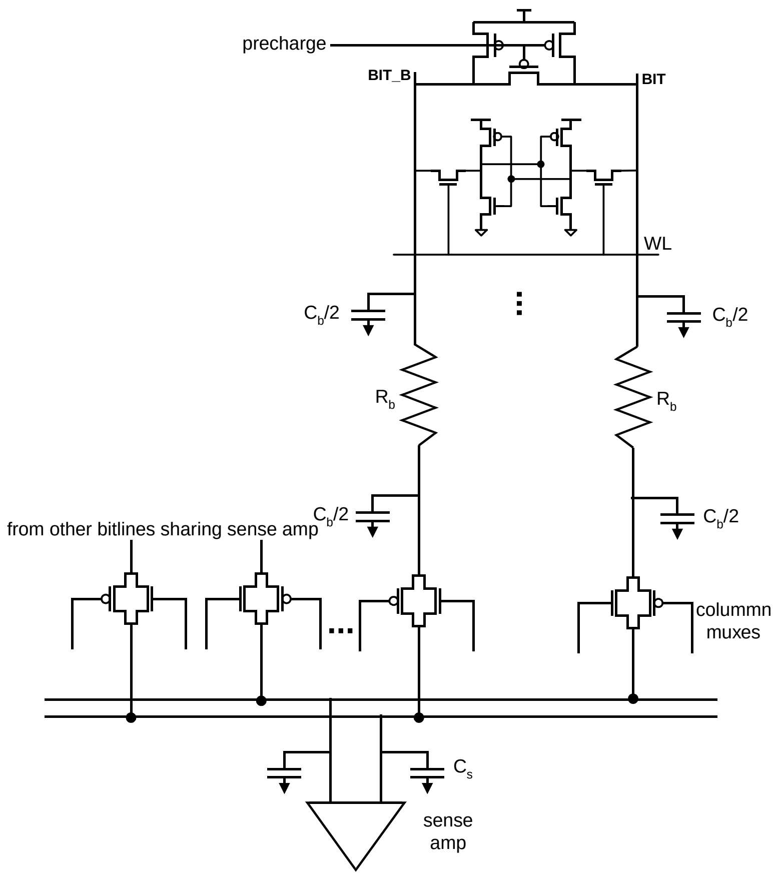


Figure 4-11: Schematic of bitline structure

The bitline column is most accurately modeled as a non-ideal current source due to the low voltage levels (between 100-200mV) of the signals at the input of the sense amplifiers. A current-source model for the bitline delay was developed in [8]. They estimate the total delay of the bitline as the sum of the delay needed to swing the total bitline capacitance by δV using the cell current and the RC time constant of the bitline modeled as a lumped π RC network. An additional factor of delay was added to account for the impact of the wordline rise-time on the bitline delay.

$$D_{bitline} = z \cdot D_{wl} + (C_b + C_{jmux} \cdot (k+1) + C_s) \cdot \frac{\delta V}{I_{cell}} + \tau_{RC} \tag{4.4}$$

where

C_s is the input capacitance of the sense amplifier;

δV is the voltage swing at the input of the sense amplifier;

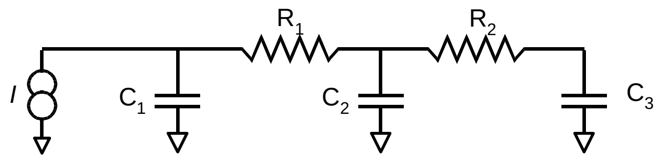
C_b is the bitline capacitance and is the sum of the wire capacitance, drain capacitances of the pass transistors and the capacitance of the pre-charge circuitry;

$C_{jmux} \cdot (k+1)$ is the total drain capacitance of the multiplexor with C_{jmux} being the unit junction capacitance of the bitline mux and k the number of bitlines multiplexed into one sense amplifier;

$z \cdot D_{wl}$ is the impact of the wordline rise time; z is a proportionality constant determined in HSPICE to be 0.3 for a wide range of block sizes [6]; D_{wl} is the wordline rise time;

I_{cell} is the cell current which is the current through the cell pass and pull-down transistors. If I_n is the saturation current per unit width of an nFET and w is the effective width of the pass and pull down transistors, then $I_{cell} = I_n \cdot w$. The effective width of the series configuration of these transistors is half the average width: $w = (w_{pass} + w_{pd})/4$ where w_{pass} is the width of the pass transistor and w_{pd} is the width of the pull-down transistor;

τ_{RC} is the bitline time constant which is estimated using the Elmore delay models [36] for a distributed RC network driven by a current source. The delay for an RC network driven by a current source is slightly different from that driven by a voltage source. Figure 4-12 is an example of time constant calculation for a current-driven RC π network. The time constant is the sum of the product of each resistance with a capacitance that is the equivalent of all the downstream capacitances lumped together in series with all the upstream capacitances lumped together.



$$\tau_{RC} = R_1 \cdot \frac{(C_1)\cdot(C_2+C_3)}{(C_1+C_2+C_3)} + R_2 \cdot \frac{(C_1+C_2)\cdot C_3}{(C_1+C_2+C_3)}$$

Figure 4-12: Current source driving a RC π network

The delay of the sense amplifier structure is the sum of the delay of the latch, the buffers and the nFET drivers (see Figure 4-10). The delay of the latch depends on its sensitivity, the required voltage gain and the speed of the bitline swing. The amplification delay of the latch is proportional to the logarithm of the required gain and the loading on the amplifier outputs [42]. According to studies conducted in [42], for a gain of about 20 with only the self loading of the sense amplfier, the delay is found to be about two *fanout-of-4* inverter delays.

If we assume that all transistors in the latch are sized in the same proportion, then the output resistance and input capacitance of the latch can be expressed as a simple function of the device width w_s [8]. The delay of the sense amplifier structure is given by equation 4.5 plus some parasitics. The optimal sizes $w_s, w_1 \cdots w_n$ are chosen so that each term in (4.5) evaluates to the fanout-of-4 delay of an inverter.

$$T_s = D_b(w_s) + \tau_s + \frac{R_s \cdot 3 \cdot C_g \cdot w_1}{w_s} + \frac{R_g \cdot 3 \cdot C_g \cdot w_2}{w_1} + \cdots + \frac{R_g \cdot C_g \cdot w_n}{w_{n-1}} \tag{4.5}$$

$$D_b(w_s) \approx \frac{G_s \cdot w_s \cdot \delta V}{I_{cell}} \tag{4.6}$$

where

$\tau_s = 2\tau_{fo4}$; the amplification delay of the latch amplifier

R_s = sense amplifier input capacitance per unit width

G_s = unit sense amplifier output resistance

w_s = width of sense amplifier

R_g, C_g = output resistance and input capacitance per unit width of a 2:1 inverter

The delay of the final stage (for w_n) depends on the loading on the sense amplifiers and is included in the comparator delay for tag paths, and output bus delay for data paths. The total delay of the datapath is the sum of the delay of the bitline structure (4.4) and the delay of the sense amplifier structure (4.5).

$$D_{datapath} = D_{bitline} + T_s \tag{4.7}$$

The total energy of the bitlines is the product of the total capacitance on the bitline, the allowed voltage swing and the supply voltage, and the number of bitline columns in the array: $E_{bitline} = C_{bitline} \cdot \delta V \cdot V_{dd} \cdot n_{columns}$. The energy of the sense amplifier structure is the sum of the energy of the latch amplifier and output buffers. The energy of the buffer and nFET drivers are easily estimated using the capacitance estimation methodologies discussed in Section 3.2.1. The energy of the latch is difficult to accurately capture analytically due to the metastable nature of the cross-coupled gain stages. In [6], Amrutur *et al.* estimate the energy of the sense amplifier as 12fJ/λ. This estimate was confirmed by running HSPICE® simulations for various sense amplifier sizes and gains. Estimates were similarly obtained for the sense amplifier input capacitance and output resistance.

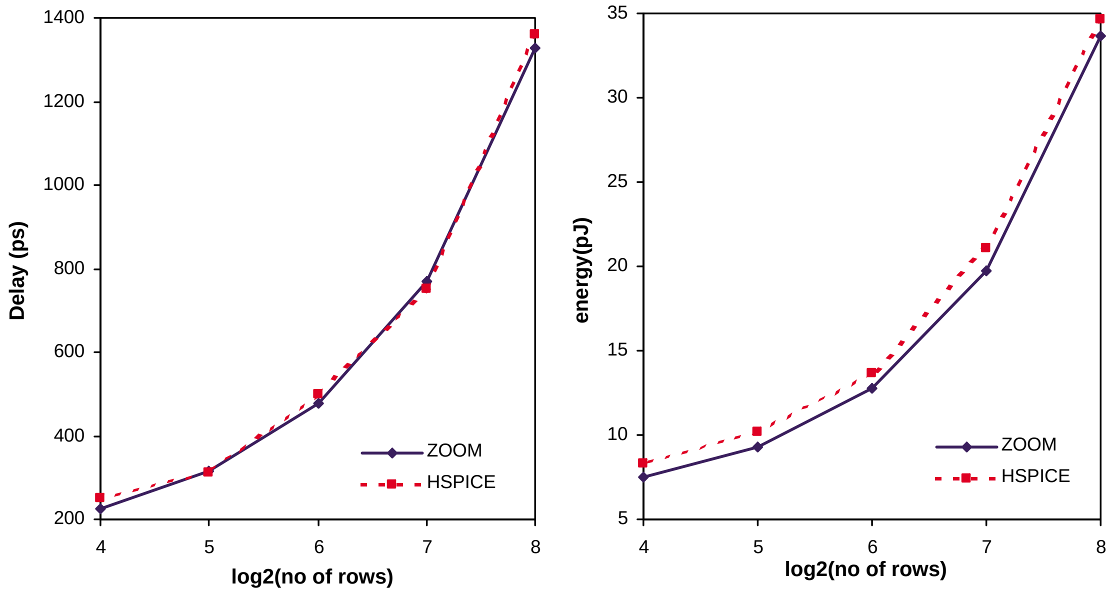


Figure 4-13: Comparison of bitline delay and energy estimates with HSPICE simulated values

Figure 4-13 compares analytical estimates of data path delay and energy with those of HSPICE® simlations for a range of bitline column heights. A column width of 128 bits was used with four bitlines multiplexed into one sense amplifier. The delay and energy estimates are within 5% and 8% respectively, of HSPICE® simulated values.

4.2.2 Divided Bitlines

Divided bitlines is one of the bitline configuration options considered during optimization. The goal of the divided-bitline approach is to reduce the delay and active energy of the SRAM bitline by reducing its total capacitance. Unilike array partitioning which achieves a similar goal by breaking up a tall and wide array into smaller sub-arrays, the divided bitline approach achieves its goal by reducing the number of pass transistors connected to the bitlines.

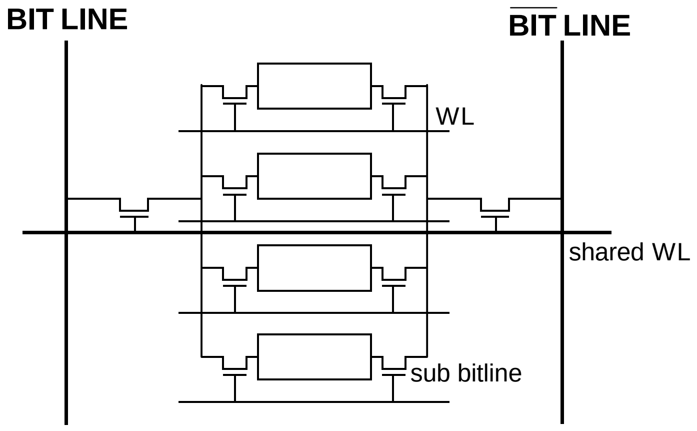


Figure 4-14: Divided bitline example with $M = 4$

In a traditional SRAM array, the pass transistor of each SRAM cell is connected to the bitline, although only one of these pass transistors is driving the bitline at any point in time. The divided bitlines (DBL) concept, introduced by Karandikar and Parhi in [17], uses a two-level heirarchy of pass transistors in which the first level of pass transistors is connected to a sub-bitline shared by M cells. This sub-bitline is then connected to the main bitline through one pass transistor, thereby reducing the total drain capacitance of the bitline by a factor of M. Figure 4-14 is an example of the DBL approach with $M = 4$. Four SRAM cells are combined together and connected to the bitline via one pass transistor. Hence, the number of pass transistors connected to the bitline is reduced by a factor of 4.

The sensing techniques evaluated in the previous section may be applied to the main bitline in DBL to reduce its energy consumption. The sub-bitlines, however, have full-rail swings. To access the DBL array, the local wordline is pulsed, together with the shared wordline. The rest of the access proceeds much like the case of traditional SRAMs.

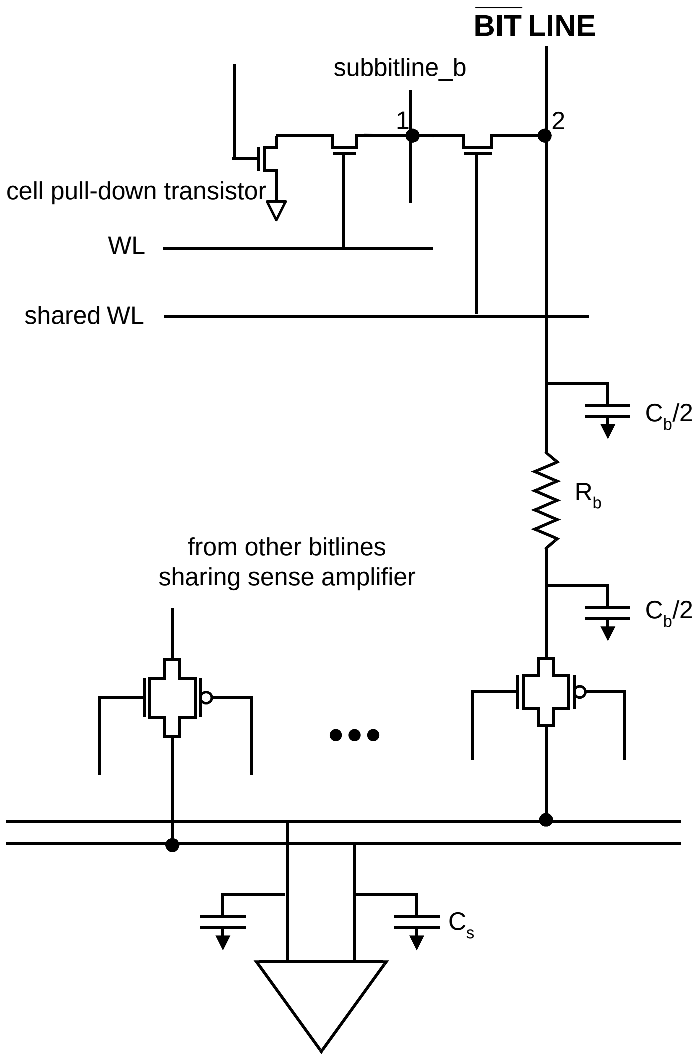
The delay of the divided bitline structure is modeled as a non-ideal current source with the time constant of a nested RC π network. The methodology is similar to that used to model regular bitlines. Figure 4-15 shows the shematic of the divided bitline. Only one of the pair of bitlines is shown for legibility. Figure 4-16 is the RC network representation of the divided bitline. The resistances and capacitances on the path are defined as follows:

$C_1 =$ one-fourth the drain capacitance of M pass transistors

$C_2 =$ one-fourth the drain capacitance of M pass transistors

$+$ the drain capacitance of the shared pass transistor

$R_1 =$ half the resistance of the sub-bitline (assuming the shared pass transistor is located in the middle of the sub-bitline)



$\overline{\text{BIT}}$ LINE
subbitline_b
1
2
cell pull-down transistor
WL
shared WL
$C_b/2$
R_b
from other bitlines sharing sense amplifier
$C_b/2$
C_s

Figure 4-15: Schematic of divided bitline

R_2 = the resistance of the shared pass transistor

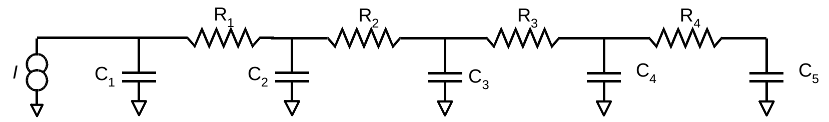
$C_3 = C_4$= half the reduced bitline capacitance

R_3 = the bitline resistance

R_4 = the output resistance of the column multiplexor

C_5 = the drain capacitance of the bitline mux

+ input capacitance of the sesense amplifier



$$\tau_{RC} = R_1 \cdot \frac{(C_1)\cdot(C_2+C_3+C_4+C_5)}{(C_1+C_2+C_3+C_4+C_5)} + R_2 \cdot \frac{(C_1+C_2)\cdot(C_3+C_4+C_5)}{(C_1+C_2+C_3+C_4+C_5)} + \cdots + R_4 \cdot \frac{(C_1+C_2+C_3+C_4)\cdot(C_5)}{(C_1+C_2+C_3+C_4+C_5)}$$

Figure 4-16: RC network for divided bitline

The delay of the bitline path is given by

$$D_{dbl} = z \cdot D_{wl} + (C_{sb}/2 + C_b + C_{jmux} \cdot (k+1) + C_s) \cdot \frac{\delta V}{I_{sb}} + \tau_{RC} \tag{4.8}$$

where C_{sb} is the sub-bitline capacitance, τ_{RC} is estimated as shown in figure 4-16 and I_{sb} is the current through the series configuration of the cell pull-down transistor, the local pass transistor and the shared pass tansistor. The remaining terms are as defined for Equation 4.4 on page 66. If we assume that the shared and local pass transistors are identically sized, then the effective width of the two pass transistors is half that of a regular bitline. Therefore $I_{sb} \approx I_{nmos} \cdot (w_{pass}/2 + w_{nmos})/4$, where I_{nmos}, w_{pass} and w_{nmos} are the saturation current per unit width of an nFET and cell pass and pull-down transistors respectively. The sense amplifier delay, T_s, estimated in (4.5) is added to obtain the total data path delay for the divided bitline structure.

The energy of the DBL structure is the sum of the energy consumed in the full-swing sub-bitline and the low-swing bitlines. Since the sub-bitlines are not precharged, the maximum energy is consumed when they make transitions in opposite direction, effectively doubling the dynamic capacitance.

$$E_{dbl} = \frac{1}{2} \cdot V_{dd} \cdot (C_b \cdot \delta V + 2C_{sb} \cdot V_{dd}) \cdot n_{columns} \tag{4.9}$$

Figure 4-17 plots normalized DBL delay and energy for array heights of 256 and

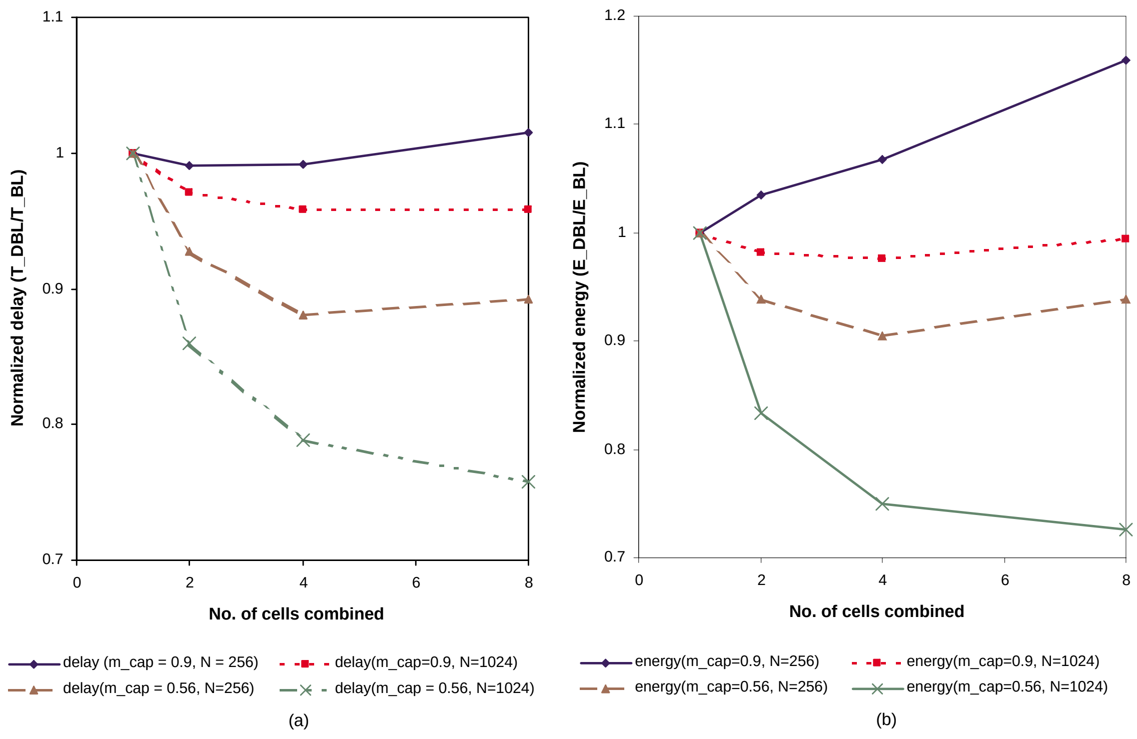


Figure 4-17: (a) Normalized DBL delay for arrays of height 256 and 1024 for metal-to-bitline capacitance ratios of 0.9 and 0.56 (b) Normalized DBL energy for arrays of height 256 and 1024 for metal-to-bitline capacitance ratios of 0.9 and 0.56

1024 rows for the worst-case and best-case metal-to-bitline capacitance ratios. These are labeled as $m_{cap} = 0.9$ and $m_{cap} = 0.56$ respectively on the graphs. The worst-case metal capacitance assumes worst-case coupling and the best-case metal capacitance assumes no coupling to neighboring wires and only half the coupling to top and bottom layers. The array width was fixed at 128 bits and four column muxing for all plots.

The optimum value of M for minimum energy consumption for an array of height N rows may be derived from (4.9) by differentiation. Let α be the ratio of metal capacitance to total capacitance of original bitline, and C be the total drain capacitance of N pass transistors. Ignoring some irrelevant portions of (4.9),

$$\begin{aligned} E(M) &= (C_b \cdot \delta V + 2C_{sb} \cdot V_{dd}) \\ &= C \cdot (\frac{1}{M} + \alpha) \cdot \delta V + 2C\frac{M+1}{N}V_{dd} \end{aligned} \tag{4.10}$$

Differentiating (4.10) with respect to M yields

$$M_{opt} = \sqrt{\left(\frac{N}{2} \times \frac{\delta V}{V}\right)} \tag{4.11}$$

Interestingly, the optimum value of M for energy minimization depends on the voltage swing of the main bitline. This is not surprising since the energy consumed in the full-swing sub-bitlines has to be more than compensated for by the energy reduction in the main bitline (with reduced capacitance) for DBL to be effective as an energy-reduction strategy. If δv is very small, the impact of the sub-bitline energy will be more pronounced and hence, M has to be small to reduce the impact of teh sub-bitline energy on the total energy.

The effectiveness of the divided bitline approach as a bitline delay-reduction strategy depends largely on the ratio of interconnect capacitance to the total bitline capacitance, α. If we ignore interconnect resistance in the sub-bitline and assume that the local and shared pass transistors have identical widths, then the delay for an array height of N rows may be approximated, to first order, as (see [17] for derivation):

$$\begin{aligned} & RC\left(\frac{M}{2N}\right) + RC\left(\frac{M}{2N} + \frac{2}{M} + 2\alpha\right)\left[\ln\left(\frac{\frac{M}{2N} + \frac{2}{M} + 2\alpha}{\frac{M}{N} + \frac{2}{M} + 2\alpha}\right) + \ln\left(\frac{1}{v(t)}\right)\right] \\ & \leq T_{dbl} \leq \\ & RC\left(\frac{M}{2N}\right) + RC\left(\frac{M}{N} + \frac{2}{M} + 2\alpha\right) \cdot \ln\left(\frac{1}{v(t)}\right) \end{aligned} \tag{4.12}$$

where R is the resistance of a pass transistors, C is the total drain capacitance of N pass transistors and

$$v(t) = 1 - \frac{\delta V}{V_{dd}}$$

If we assume that $M \ll N$, the delay expression in (4.12) simplifies to:

$$T_{dbl} \approx RC\left(\frac{M}{N} + \frac{2}{M} + 2\alpha\right) \cdot \ln\left(\frac{1}{v(t)}\right) \tag{4.13}$$

The delay of the traditional bitline is similarly approximated as

$$T_{bl} \approx (1 + \alpha) \cdot RC \cdot \ln\left(\frac{1}{v(t)}\right)$$

Normalizing (4.13) with the delay of the traditional bitline, and slightly re-arranging gives,

$$T_{dbl} = \frac{1}{1+\alpha} \cdot \left(\frac{M}{N} + \frac{2}{M}\right) + \frac{2\alpha}{1+\alpha} \tag{4.14}$$

The second term in the delay equation in (4.14) depends only on the metal-to-bitline capacitance ratio, α. This portion of delay, which tends to 1 as α increases, cannot be improved by DBL. Infact, it worsens slightly since the insertion of the sub-bitline and extra pass transistors lengthens both the bitline and wordline wires. Figure 4-18 plots the scaling of this portion of the divided bitline delay with α. The impact of α on the effectiveness of DBL is also observed in Figure 4-17 which plots delay and energy for various values of M using ZOOM's DBL model.

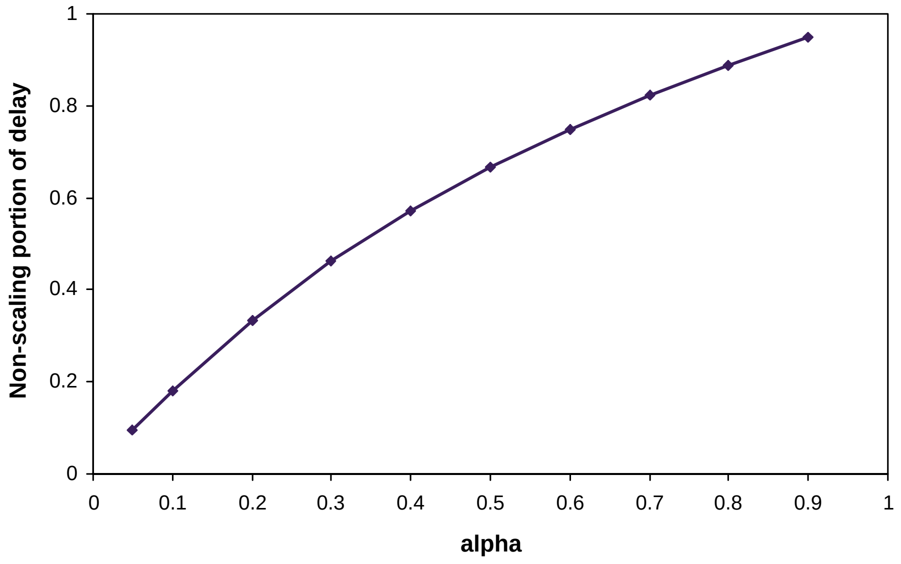


Figure 4-18: Impact of metal-to-bitline capacitance ratio on delay scalability with M

4.3 SRAM Tag Comparators

The comparators for the SRAM tag array are located at the bottom of the array. The comparators are made up of mirroring pairs of two-series nFETs that compute the XOR of the stored tag bits and incoming tags to determine a hit. The comparators are connected to a matchline which is connected to the supply voltage via a pre-charge pFET device and to ground via a foot nFET that triggers the evaluate phase of the comparator. To reduce the worst-case comparison delay, the comparators are split into two equal and independent sections each of which compares half the tag bits (in parallel). The results from the two halves are then combined with a NAND gate and an inverter to generate the match signal. Figure 4-19 shows the split implementation of the comparator.

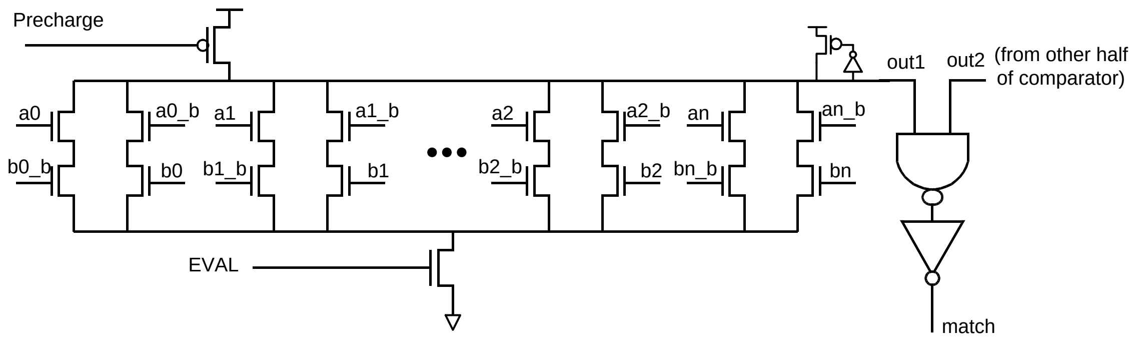


Figure 4-19: SRAM-tag comparator structure

The outputs from the tag sense amplifiers, which arrive after the incoming tags have stabilized, are connected to the inputs labeled a_n and $\bar{a}_n$ which are closer to the output to improve their stabilization delay. The b_n and $\bar{b}_n$ inputs are driven by tag bits in the address. Tag comparison begins with the matchline precharged high and the evaluation transistor turned off. A mismatch in any bit closes one pull-down path and discharges the output. In order to ensure that the output is not discharged before the a_n bits become stable, the EVAL signal is often controlled using a self-timed circuit. This timing circuit is not modeled in ZOOM, instead it is assumed that the EVAL signal is perfectly timed to coincide with the stabilization of the a_n inputs which are connected to the output of the sense amplfiers. When EVAL goes high, the foot transistor connects the comparators to ground and the comparator output, if there is a mismatch, begins to discharge.

The worst-case delay of the comparator occurs when only 1 bit mismatches such that the entire matchline capacitance is discharged though the series configuration of this mismatched path and the foot transistor. The delay of the comparator is the sum of the delay to to discharge the output and the delay to combine the two match signals and buffer the output to the dataline tristate buffers.

$$T_{comp} = T_{discharge} + T_{drivers} \tag{4.15}$$

Discharge Delay

Since we assume that both the incoming and stored tag bit inputs will be stable by the time EVAL goes high, $T_{discharge}$ is the time to discharge the output loaded through the series configuration of the foot nFET transistor and the mismatched path. Figure 4-20 shows the equivalent circuit of the discharge path.

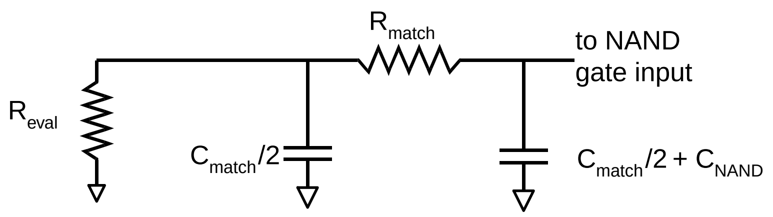


Figure 4-20: Equivalent circuit of SRAM comparator discharge path

The delay of the RC comparator network is the 50% fall time given by

$$T_{discharge} = ln(2) \cdot \tau_{RC} \tag{4.16}$$

$$\tau_{RC} = R_{eval} \cdot C_{match}/2 + (R_{eval} + R_{match}) \cdot (C_{match}/2 + C_{NAND}) \tag{4.17}$$

where

R_{eval} is the "on" resistance of the evaluation foot transistor

$R_{match} = 2 \cdot R_{non}/w_{comp} + R_{matchwire}$ is the resistance of the mismatched nFET path and the matchline wire; here, w_{comp} is the width of a comparator nFET.

C_{match} is the sum of the drain capacitances of the comparators and precharge PFET and the capacitance of the matchline wire

C_{NAND} is the input capacitance of the NAND gate.

The evaluation transistor and the comparators are sized so that $T_{discharge}$ is approximately equal to the rise time of the NAND gate to minimize the impact of short-circuit currents on the delay.

Buffering Delay

For a B-bit word access, the match signal has to drive $2B$ tristate buffers corresponding to the requested word and its complement since the global sense amplifiers require differential inputs. If multiple tags are multiplexed into one sense amplifier, then the match results has to be combined with a select bit, generated in parallel with the tag read/comparison, to generate a separate match signal for data arrays for each additional tag multiplexed. This introduces a level of branching into the comparator path. Another level of branching results if a data block is split across multiple sub-arrays. Figure 4-21 is an example of the branches involved in a tag array in which two tags share a single sense amplifier. The data block associated with each tag is split into two sub-arrays. These branches are handled using the the logical effort approach.

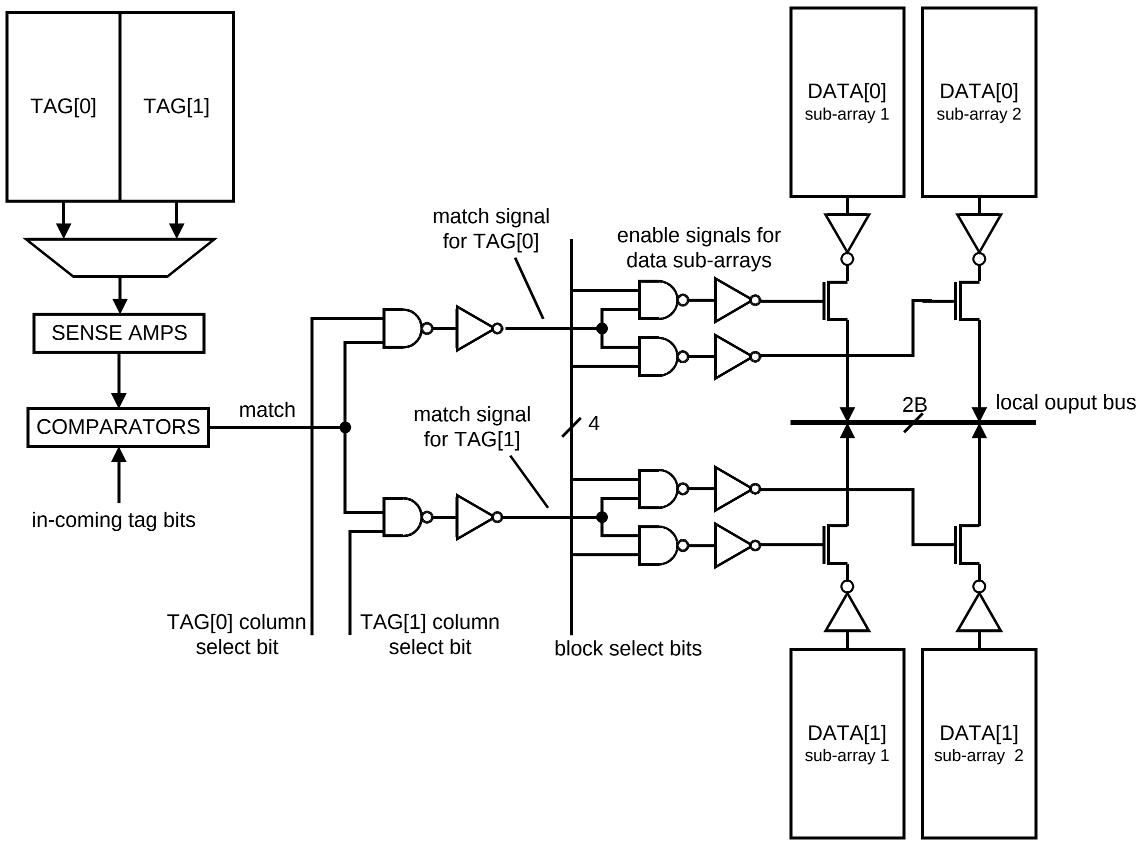


Figure 4-21: Example of branching in the comparator buffer chain

Since the tag and data sub-arrays are kept in completely separate array matrices, the connecting wire between the match output and the appropriate sub-array could get quite long and dominate the comparator delay. To manage the wire delays, a cascade of sized-up repeaters is inserted on the path if the wire length is greater than some critical length. If we assume that the data array is organized such that sub-arrays sharing control signals are placed close to each other, then there is some degree of freedom in the location of the branching points relative to the interconnect line and repeaters. We assume that the first level of branching occurs before, while the second level of branching occurs after the cascade of repeaters.

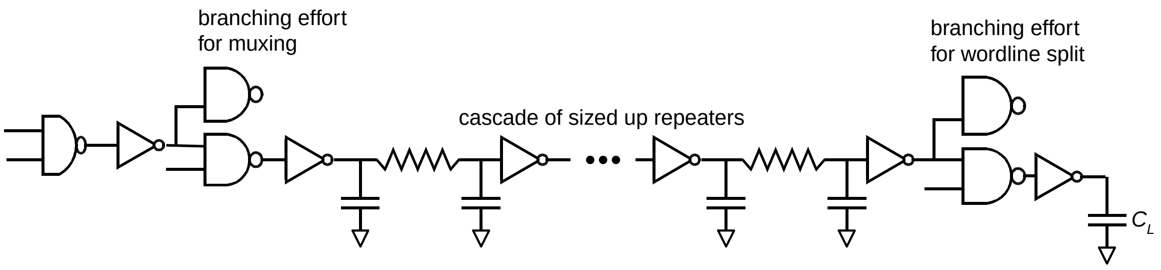


Figure 4-22: Critical path of comparator buffer chain

The critical path of the buffer chain is shown in figure 4-22. The gates on the path are sized so that each stage has the delay of a fanout-of-4 inverter. The path is treated as two independent sub-chains separated by a delay element (the repeater cascade); the first driving a load consisting of a wire segment and one repeater, and the second driven by the final repeater in the cascade and driving a load, C_L, which is the the total capacitance of $2B$ tristate buffers. The delay of the interconnect with repeaters, T_{wire}, is estimated using the methodology described in Section 3.2.3. Assuming n stages in the first sub-chain and k stages in the second sub-chain, the total delay of the path is given by

$$T_{drivers} = (n + k) \cdot T_{fo4} + T_{wire} + parasitics \tag{4.18}$$

Figure 4-23 compares analytical estimates for the comparator delay and energy with simulated values from HSPICE®. The delay and energy estimates are within 2% and 5% of HSPICE simulated values respectively.

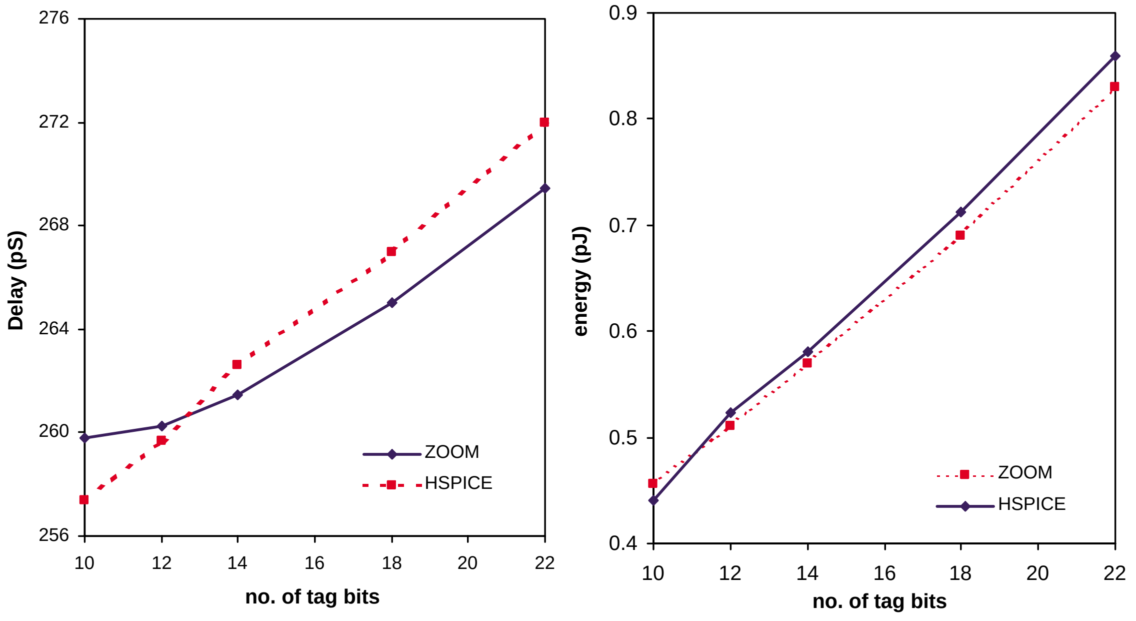


Figure 4-23: Comparison of comparator delay and energy estimate with HSPICE simulated values

4.4 CAM Tag Path

CAMs (content-addressable memories) are used in many search structures. This discussion focusses on their use as tag stores and comparators in a CAM cache architecture as described in Section 3.1.1. The CAM tag path is made up of CAM cells and other peripheral circuits such as sense amplifiers and precharge circuitry. The CAM array for a k-way set associative cache with m bits per tag consists of a matrix of k rows by m columns of memory cells. Figure 4-24 shows the basic structure of a CAM array.

Each CAM cell consists of an SRAM cell and a pair of two-series nFETs for performing comparisons. When the wordline of the cell is asserted, it behaves like an SRAM and may be read and written to as described in Section 4.2. When the wordline is not asserted, the complementary contents of the SRAM cell are propagated to the

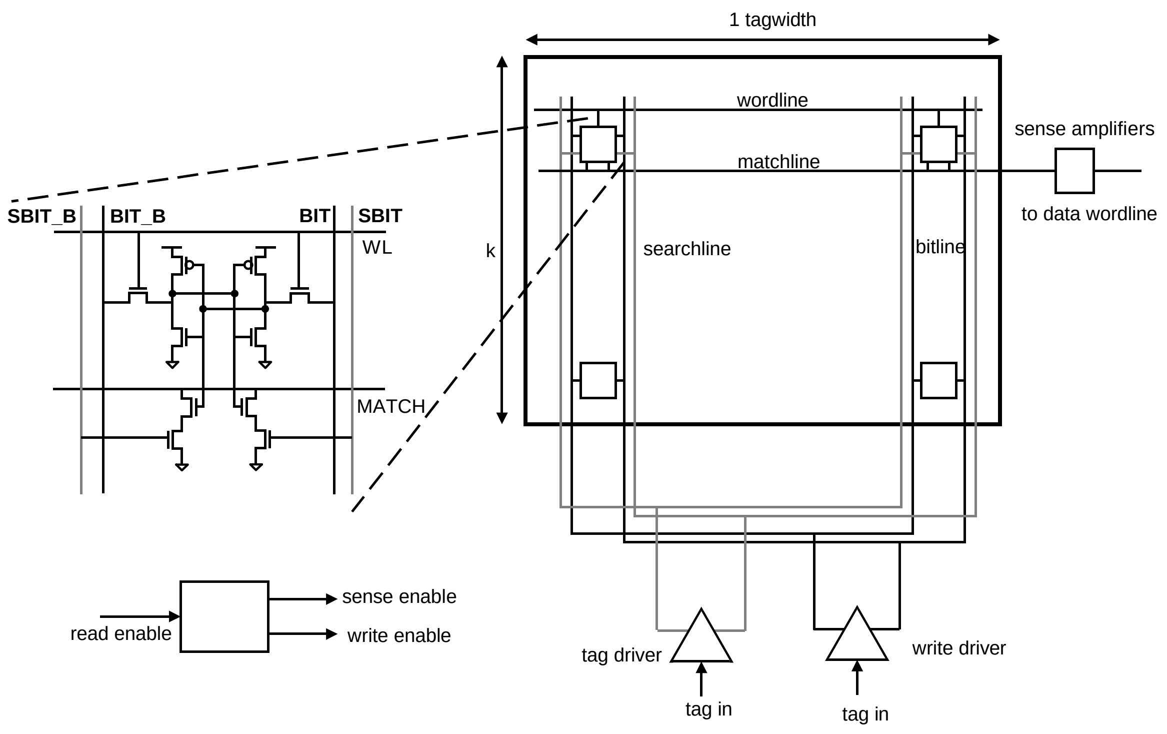


Figure 4-24: Basic structure of a CAM array

gate inputs of one of the mirroring pair of nFETs in the embedded series configuration where they may be compared to another complementary bit set placed on the other pair of mirroring nFETs.

A read operation to the cache is a compare operation to the CAM array. A compare operation begins with the MATCH line pre-charged to a '1' and the incoming tag bits broadcast on SBIT and SBIT_B. With the wordline low, the value stored in the cross-coupled inverter is compared to the complementary values on SBIT and SBIT_B. A mismatch creates a direct path to ground through the series nFETs in the comparator, causing the MATCH line to discharge. Single-ended sense amplifiers located at the end of the MATCH line sense and amplify the results of the tag comparison, which is connected to the wordline of the appropriate data block in the corresponding data array. Writes to the CAM array proceed in the same manner as writes to the SRAM array, with the new tag bits transferred onto the bitlines.

Since the bitlines and search lines are never used simultaneously, some area-conscious designs combine them into one multiplexed line (i.e. both the SRAM pass transistor and one of the transistors in the comparator are connected to the same line). This approach is sub-optimal for both high-speed and low-energy solutions since the capacitance of the multiplexed line is then practically double that of dedicated lines. In addition to driving extra capacitance for every access, which almost doubles the access energy, bigger line drivers are needed to meet delay targets. For this reason, ZOOM uses dedicated search and bitlines for the CAM array to improve energy consumption at the cost of about a 10% increase in CAM array area. However, since the tag array area is only a small fraction of the total area of the cache, the overall impact is much less than 10% of total area.

4.4.1 CAM Tag Path Delay and Energy

Figure 4-25 shows the elements on the critical path of the CAM row. To reduce the worst-case delay of the comparators, the tag path is split into two equal and independent sections each of which compares half the tag bits (in parallel). The results from the two halves are then combined with a NAND gate and an inverter to generate the match signal for the row.

This modification, which also applies to SRAM tag comparators, not only reduces the delay but also potentially reduces the energy consumed by tag compares. The top half of the tag mismatches (and hence discharges) infrequently due to the spatial

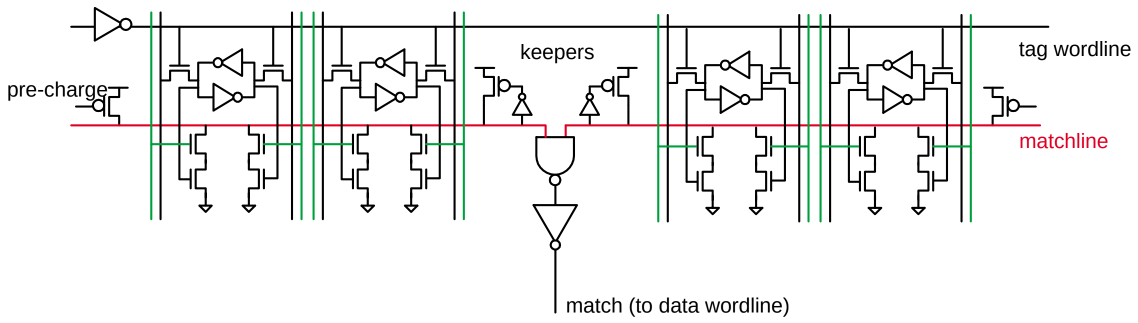


Figure 4-25: Split comparator architecture / CAM path

locality of cached data. For delay reduction purposes, splitting the comparator only makes sense if the total delay of the lumped implementation, including output buffers if needed, is greater than two fanout-of-4 inverter delays. This is because the total minimum delay of the NAND gate, which is the sum of the parasitic, switching and short-circuit delays, is about that of a fanout-of-4 inverter even if fanout-of-1 sizing is used; and the delay of the inverter on the output of the NAND is about another fanout-of-4 delay. In most cases, extra buffer stages are needed to drive the compare results to the input of the data wordline driver so that the split comparator architecture does not come at an extra delay cost.

Figure 4-26 shows the full critical path of the CAM path. The numbered black

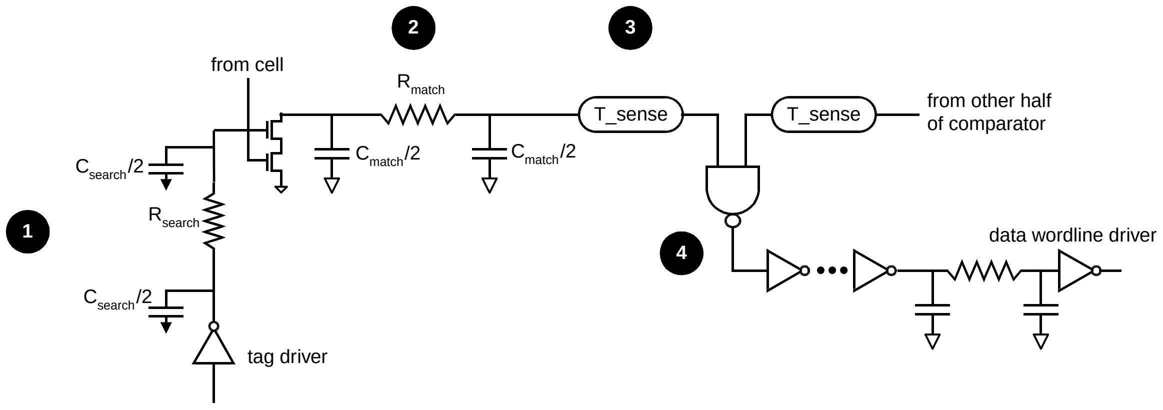


Figure 4-26: CAM critical path

dots indicate the four main sections of the critical path as it is traversed during an access. Dot '1' is the portion of the critical path from the input of the tag driver to the input of the comparator. This delay corresponds to the time it takes to drive the tag bits onto the search lines. This delay is on the critical path because it is assumed that the inputs of the tag driver for a given CAM array are only activated after pre-decoding. Dot '2' is the portion of the critical path from the input of the comparators to the input of the sense amplifier. This is the time it takes to drop the pre-charged matchline by some δV that can be amplified by the sense amplifier. Dot '3' is the portion of the critical path from the input to the output of the sense amplifier and Dot '4' is the delay to combine the match results and buffer the signal to the input of the data wordline. The sum of all the delays on the path gives the total delay of the CAM path (4.19).

$$T_{cam} = T_{search} + T_{compare} + T_{sense} + T_{drivers} \tag{4.19}$$

The next few paragraphs will briefly derive analytical models for the delays in Equation 4.19.

Delay of the Search Lines

The search line is simply a loaded wire driven at one end by a large driver which is sized to have a delay of a fanout-of-4 inverter. Its 50% delay is estimated using the

π model described in Section 3.2.2 as:

$$T_{search} = \ln(2) \cdot \left(R_{driver} \cdot (C_{driver} + C_{matchline}) + \frac{R_{searchline} \cdot C_{searchline}}{2} \right) \qquad (4.20)$$

where

R_{driver} is the output resistance of the tag driver,

C_{driver} is total gate and drain capacitance of the tag driver,

$C_{searchline}$ is the capacitance of the searchline wire and gate capacitances of all the comparator nFETs connected to it, and

$R_{searchline}$ is the resistance of the searchline wire.

Delay of the Match Lines

The worst-case delay of the compare path occurs when only one bit mismatches such that the entire matchline is discharged though 1 series path. Assuming that the input of the comparator connected to the cell is stable by the time the broadcast tag bits stabilize, the critical path of the compare will not include the time to read the stored bit onto the comparator. Figure 4-27 shows the equivalent RC circuit of

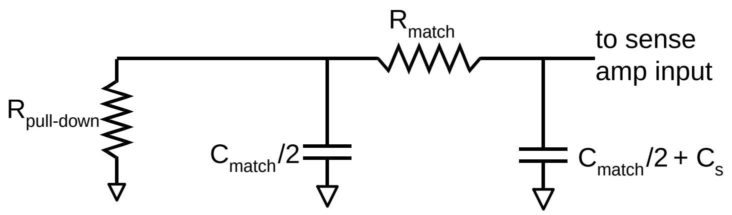


Figure 4-27: Equivalent RC of CAM compare critical path

the critical path of the matchline. The compare output connected to the input of the sense amplifier only needs to discharge by some δV large enough to be accurately sensed and amplified by the single-ended sense amplifier. However, the internal nodes need to charge up before the final node can reach this voltage. The current-source model used for the low-swing SRAM bitlines assumes that the time constant of the RC network is much smaller than the time to charge the output to δV, so that the internal nodes slew at the same rate as the output node in steady state. For distributed RC networks with many intermediate nodes, this simplifying assumption introduces only a small error in the estimates. This is not the case for the matchline. Hence, we estimate its delay using a modified π RC model that accounts for the different node slew rates. The delay of the matchline is estimated as the sum of the time constant of the matchline and the time to charge up C_s to δV. This delay is summarized by Equation 4.21 below:

$$T_{compare} = R_{pull-down} \cdot \frac{C_{match}}{2} + (R_{pull-down} + R_{match}) \cdot \left(\frac{C_{match}}{2} + C_s \cdot \ln \left(\frac{1}{1 - \frac{\delta V}{V_{dd}}} \right) \right) \tag{4.21}$$

$$\begin{aligned} T_{compare} &= R_{pull-down} \cdot \frac{C_{match}}{2} + (R_{pull-down} + R_{match}) \cdot \frac{C_{match}}{2} \\ &+ \ln \left(\frac{1}{1 - \frac{\delta V}{V_{dd}}} \right) \cdot (R_{pull-down} + R_{match}) \cdot C_s \\ &+ 0.3 \cdot T_{search} \end{aligned} \tag{4.22}$$

where

$R_{pull-down}$ is the resistance of the 2 series nFETs in the comparator;

R_{match} is the resistance of the matchline wire

C_{match} is the sum of the capacitance of the matchline wire, the drain capacitances of all the comparators and the drain capacitance of the precharge pFET

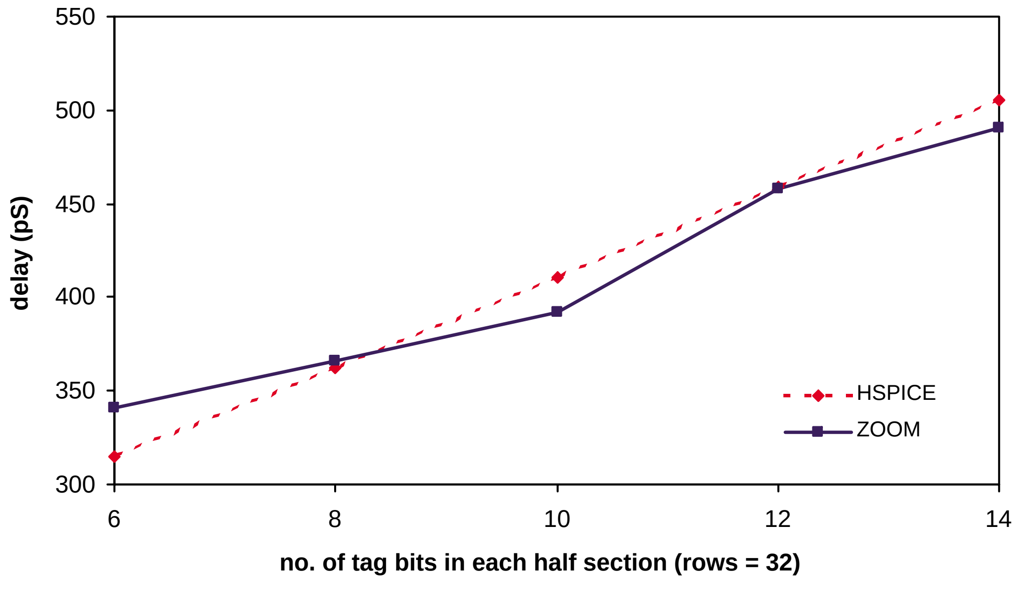
C_s is the input capacitance of the single-ended sense amplifier.

For matchlines with fewer that ten columns, the 50% RC delay, instead of the time constant (or 63% rise time), of the final node preceeding sense amplifier is used in the delay estimation.

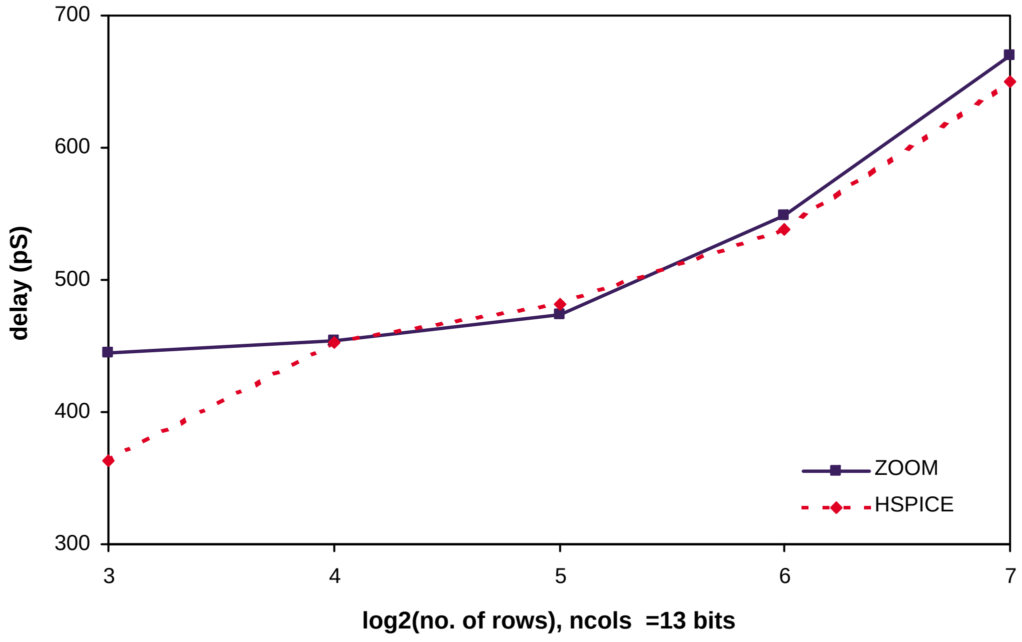
Figure 4-28 (a) plots the total delay of the matchline and searchline as a function half-array width. The height of the CAM array (associativity) was fixed at 32 rows. The analytical estimates are within 3% of HSPICE® simulated values for arrays wider than 6 bits. Figure 4-28 (b) plots the campath delay for a range of heights for a fixed half-array width of 13. The analytical estimates are within 3% of HSPICE® simulated values if the column height is greater than 8.

Delay of the Sense Amplifier and Buffers

A latch-style sense amplifier structure similar to that described in Section 4.2.1 is assumed. The assumed single-ended sense amplifier consists of a cross-coupled inverters, much like the internals of the latch-style sense amplifier structure described in Section 4.2.1. The delay of the sense amplifier depends on its sensitivity, the required voltage gain and the speed of the matchline swing. Two main factors affect the sensitivity and, ultimately, the delay of the single-ended sense amplifier: the non-scaling mismatch voltage and noise on the supply voltage. The single-ended sense amplifiers used to amplify the compare results are not as sensitive as their differential counterparts (e.g. the latch-style sense amplifier) whose differential pair of inputs are



(a)



(b)

Figure 4-28: (a) CAM path delay scaling with number of tags (columns) per half section for 32 rows. (b) CAM path delay scaling with number of rows (associativity) for a fixed half-section column width of 13 bits

able to reject common-mode noise. The voltage drop needed for adequate sensing, δV, is assumed to be approximately 15% of the supply voltage, allowing for a 10% noise margin on the supply lines and a constant mismatch of 50mV due to dopant level fluctuations. The delay of the sense amplifier buffer chain is given by Equation 4.23.

$$\begin{aligned} T_{sense} &= \tau_s + D_b(w_s) + \frac{R_s \cdot 3 \cdot C_g \cdot w_{NAND} \cdot le(NAND,2)}{w_s} + \frac{R_g \cdot 3 \cdot C_g \cdot w_1}{w_{NAND}} + \\ &\cdots + \frac{R_g \cdot 3 \cdot C_g \cdot w_n}{w_{n-1}} + 0.38 \cdot R_{wire} \cdot C_{wire} + parasitics \end{aligned} \tag{4.23}$$

$$D_b(w_s) \approx \frac{C_s \cdot w_s \cdot \delta V}{I_{nmos} \cdot 0.5 \cdot w_{comp}} \tag{4.24}$$

where

τ_s = the amplification delay $\approx T_{fo4}$

$le(NAND, 2)$ = the logical effort of a 2-input NAND gate

w_s = the width of the transistors in the sense amplifier

R_{wire}, C_{wire} = resistance and capacitance of wire to data wordline driver input

$w_1 \cdots w_n$ = width of output buffers

I_{nmos} = unit saturation current of an nFET

In many cases, the data sub-array associated with a particular CAM tag array is split along the wordlines to reduce the energy dissipated per access. If k is the number of wordline divisions, the match signal from each row is connected to k wordline drivers through a logic chain similar to that discussed in Section 4.3. Since there is no column multiplexing the CAM tag array, only one level of branching is needed.

CAM Array Energy

The energy of the CAM array is the sum of the energy in the matchlines and search lines. Since there is at most one match per access, all but one of the match lines discharge. Unlike the SRAM bitlines, the sensing technique in the CAM does not limit the swing. All the match signals are *OR-ed* together using some optimum implementation of an m-input OR gate to generate a *hit* signal. This signal is used by the pipeline control to initiate pipeline stalls. It also serves to trigger access to the next level cache if a miss is detected. Since the CAM cache itself does not use the hit signal in its access path, its generation is not on the critical path of the access. Hence, we model only the energy consumed in the OR gate but not its delay. Let m and k be the number of rows an columns respectively in the CAM array. The total energy is given by (4.25)

$$E_{campath} = E_{searchlines} + E_{matchlines} + E_{sense+drivers} + E_{OR} \tag{4.25}$$

where

$$E_{searchlines} = \frac{1}{2} \cdot V_{dd}^2 \cdot k \cdot C_{search} \tag{4.26}$$

$$E_{matchlines} = \frac{1}{2} \cdot V_{dd}^2 \cdot (m-1) \cdot C_{match} \tag{4.27}$$

$$E_{sense+drivers} = \frac{1}{2} \cdot V_{dd}^2 \cdot (m-1) \cdot C_{sense+gates} \tag{4.28}$$

The capacitances in the equations above are as defined for their corresponding delay equations. The energy of the OR gate, E_{OR}, depends on its implementation. Since m is typically greater than four, it is economical, (for both energy and delay) to use a heirarchical logic structure with smaller gates with alternating *DeMorganization* of NOR and NAND inputs and outputs. The optimal structure and stage sizing are determined using a methodology similar to that used for decoders in Section 4.1.

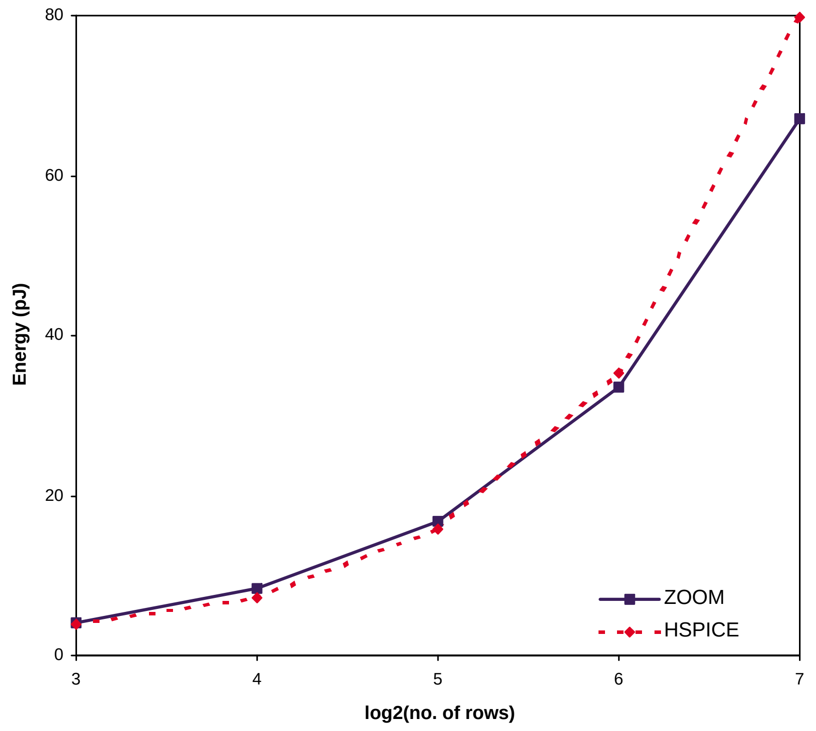


Figure 4-29: CAM path energy scaling with associativity (number of rows) for a fixed column width of 13 bits per half section

Figure 4-29 compares analytical estimates for the energy of the CAM path from the input tag driver to the input of the OR gate with simulated values from HSPICE®. The column width for each section was fixed at 13 bits for these measurements.

4.5 Data Routing

The data has to be routed from the output of the local sense amplifiers to the cache-CPU port. The routing mechanism depends on the floor plan. The sub-arrays are arranged in a matrix structure to equalize the dimensions as much as possible (i.e. achieve an aspect ratio close to 1) since this also minimizes the total routing delay. Since the data and tag sub-arrays for an SRAM cache are kept in separate matrices, apect ratio minimization is only constrained by the dimensions of the sub-arrays. For a CAM cache, data blocks are kept in close proximity to, and on the same row as, their corresponding CAM tag arrays to limit the impact of interconnect in the matchlines. This further constrains aspect ratio minimization for CAM caches. Figure 4-30 shows the floor plan for the data array of an SRAM cache with 32 sub-arrays.

Data routing is performed in two stages. First the local dataline selects one of four differential data sets from a 2×2 sub-block of sub-arrays using a two-way multiplexor on two shared local buses. The output of the multiplexor is driven to vertical global buses running between the sub-blocks where it is selectively driven onto the global bus using tristate buffers. The tristate buffers are enabled by a combination of select bits made up of match signals and sub-array pre-decode signals. By construction, only one of the data sets is the requested word and therefore, only one can be placed on the vertical global bus. The data on the vertical buses are then propagated to the cache-CPU port using a series of mux stages on a horizontal bus. The wire delay of the global bus is controlled by inserting a cascade of sized-up repeaters (Section 3.2.3).

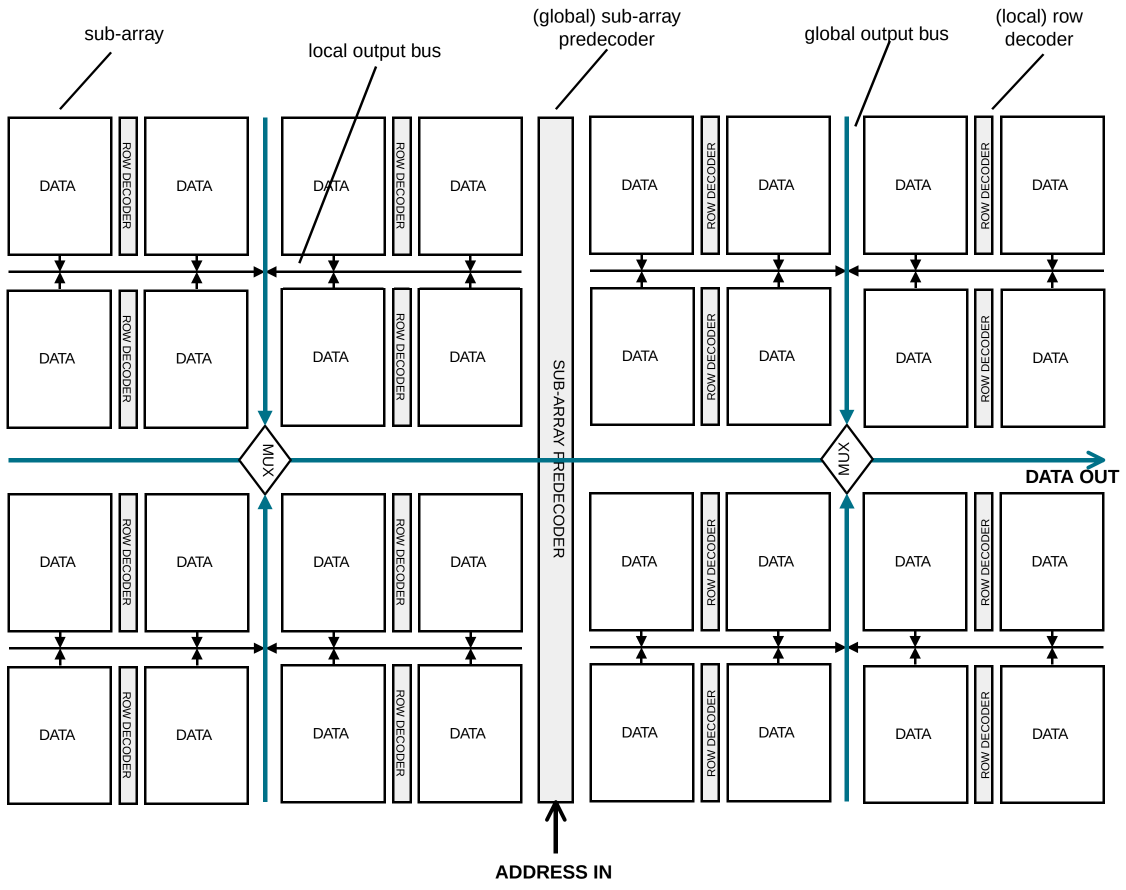


Figure 4-30: Floor Planning example for SRAM cache with 32 data sub-arrays

Low-swing and sensing techniques are used on the local bus.

4.5.1 Local Output Bus

The local output bus routes data from the output of the bitline sense amplifiers to the vertical global buses. The local bus consists of some wires to route data between the two adjacent sub-arrays, precharge circuitry, two-way multiplexors and differetial sense amplifiers. The local bus operates much like the SRAM bitlines. The complementary datalines are initially precharged to ‘1’. When the match signal goes high, the sense amplifier outputs are connected to the datalines through a pair of

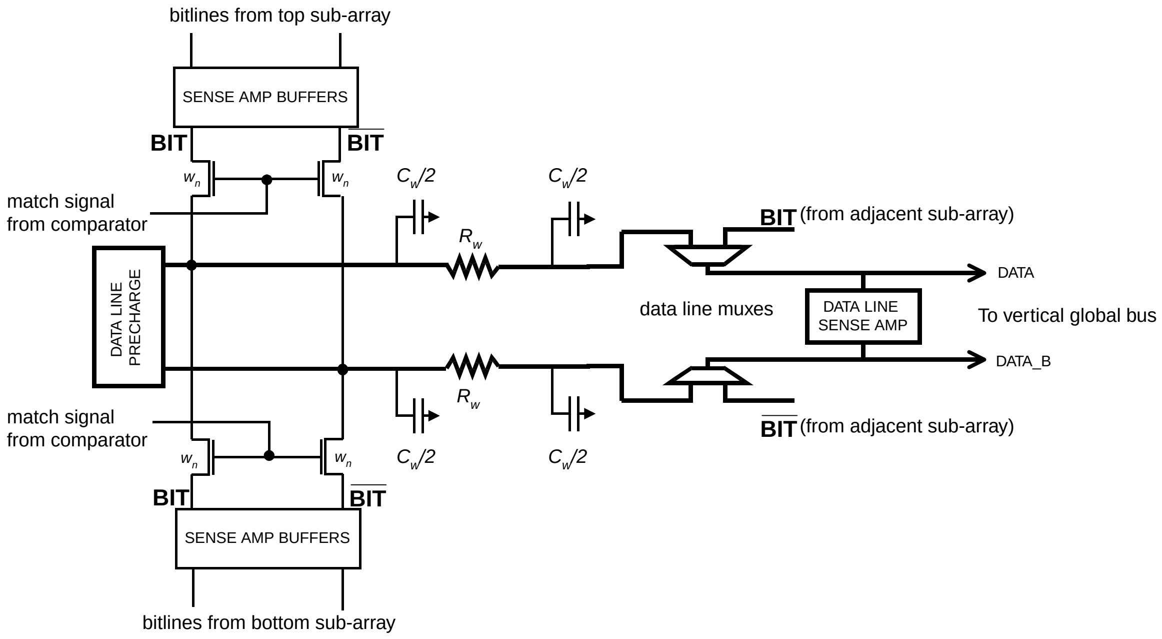


Figure 4-31: Local bus structure

nFET drivers. One of the datalines is discharged. Two-way muxes select between data from a pair of datalines. The output of the muxes are amplified through a differential sense amplifier and driven to the vertical buses. The schematic for the dataline path is shown in Figure 4-31. As shown in the figure, each dataline is shared by a pair of mirroring sub-arrays. And all four sub-arrays in the 2×2 sub-block share one sense amplifier via the two-way multplexors.

Due to the structural and operational similarities of the bitline and dataline paths, the delay equations (4.4, 4.5, 4.7) derived in Section 4.2.1 for the bitline structure also apply to the local dataline structure. Here, the cell current is replaced by the current through the nFET drivers, $I_n \cdot w$, where I_n is the saturation current through a unit nFET and w is the width of the nFET driver. The energy estimation methodology for the bitline structure also applies.

The global bus is simply loaded wires, repeaters and multiplexors. Its energy and delay is estimated using a straightforward application of the estimation methodologies discussed in Chapter 3. The delay of the global bus is the time it takes to route data from the sub-array situated farthest from the cache-CPU port. Its energy is the average of the energy to route data from each of the vertical wires to the data port given by:

$$E_{global} = n_{bits} \cdot \frac{1}{2} {V_{dd}}^2 \cdot \frac{1}{n_{vwires}} \cdot \left(n_{vwires} \cdot C_{vwire} + \frac{n_{vwires}(n_{vwires}+1)}{2} \cdot (C_{hwire} + 2C_{mux}) \right) \tag{4.29}$$

where

n_{bits} = the number of bits in the requested word

C_{vwire} = the capacitance of a vertical wire segment and its load of tristate buffers

n_{vwires} = the number of vertical wires (per bit) in the array (= 2 in Figure 4-30)

C_{hwire} = the capacitance of a horizontal wire segment and its repeaters

C_{mux} = the capacitance of the two-way mux in the global path

The capacitance of the multiplexor, C_{mux}, is multiplied by 2 since two 2-way muxes are needed to select one of three signals. The three signals consist of the in-coming signal on the horizontal global line and the signals on the two vertical lines from the top and bottom halves of the array as shown in Figure 4-30.

4.6 Putting It All Together

This section derives access time and energy equations for the cache using the models derived in the preceeding sections, followed by some applications of the EDA optimizer.

4.6.1 Cache Access Time and Energy

The equations derived in the previous section are combined to obtain the access time and energy of the cache. The critical path of the cache depends on whether it uses two-phase (*early-select*) or single-phase (*late-select*) accesses.The performance penalty of two-phase accesses is only tolerable if associativity is high enough so that the total factor decrease in energy is at least the same as the factor increase in delay. Since the energy savings of two-phase accesses in direct-mapped caches are only realized in case of misses (which are rare), direct-mapped caches are only supported in the single-phase access configuration

The access time for single-phase accesses is generally given by:

$$T_{access} = max(T_{tag}, T_{data}) + T_{routing} \tag{4.30}$$

Direct-mapped caches do not need to wait for the match result before placing the data on the output bus. Hence, for a direct-mapped cache, the routing delay only consists of the global routing delay:

$$T_{tag} \quad = \quad T_{tag\_decoder} + T_{tag\_wordline} + T_{tag\_datapath} + T_{compare}$$

$$\begin{aligned} T_{data} &= T_{data\_decoder} + T_{data\_wordline} + T_{word\_datapath} + T_{local\_bus} \\ T_{routing} &= T_{global\_routing} \end{aligned}$$

where

$$T_{(tag,data)\_decoder} = T_{(tag,data)\_global-decoder} + T_{(tag,data)\_row-decoder}$$

For a set-associative cache, the data from the sense amplifier has to be selectively loaded on the local bus (using the match signal) since there is more than one potential data source even after decoding. The delay terms for single-phase access in set-associative caches are defined as:

$$\begin{aligned} T_{tag} &= T_{tag\_decoder} + T_{tag\_wordline} + T_{tag\_datapath} + T_{compare} \\ T_{data} &= T_{data\_decoder} + T_{data\_wordline} + T_{word\_datapath} \\ T_{routing} &= T_{local\_bus} + T_{global\_routing} \end{aligned}$$

In an A-way set-associative cache, $\lceil A/2 \rceil$ data and tag row decoders and A each of data and tag sub-arrays are activated for each access. Since only one of these data sub-arrays may supply the requested word, we assume that only one local bus undergoes transitions. The comparators in all but one tag sub-array discharge. The access energy is therefore given by:

$$\begin{aligned} E_{access} &= E_{data-global-predecoder} + E_{tag-global-predecoder} \\ &+ (E_{data-row-decoder} + E_{tag-row-decoder}) \cdot \left\lceil \frac{A}{2} \right\rceil \end{aligned}$$

$$
\begin{aligned}
&+ \quad (E_{data-wordline} + E_{tag-wordline}) \cdot A \\
&+ \quad (E_{word-datapath} + E_{tag-datapath}) \cdot A \\
&+ \quad E_{local\_bus} + E_{global\_routing} + E_{compare} \cdot \lceil A - 1 \rceil
\end{aligned}
\tag{4.31}
$$

The access time and access energy equations for two-phase accesses depend on how serialized the path is and whether CAM or SRAM tag stores are used. If we assume that global pre-decoding in the data and tag arrays for an SRAM cache occur in parallel, then the delay for global pre-decoding in the data array is not on the critical path of the access (i.e. we assume data pre-decoding is completed by the time the tag compare result is available). The delay for a two-phase access to an SRAM cache is therefore given by:

$$
\begin{aligned}
T_{access} &= T_{tag\_global-decoder} + T_{tag\_row-decoder} \\
&+ \quad T_{tag\_wordline} + T_{tag\_datapath} + T_{compare} \\
&+ \quad T_{data\_row-decoder} + T_{data\_wordline} + T_{word\_datapath} \\
&+ \quad T_{local\_bus} + T_{global\_routing}
\end{aligned}
\tag{4.32}
$$

The energy for two-phase accesses is generally given by:

$$
E_{access} = E_{tag} + E_{data} + E_{routing} \tag{4.33}
$$

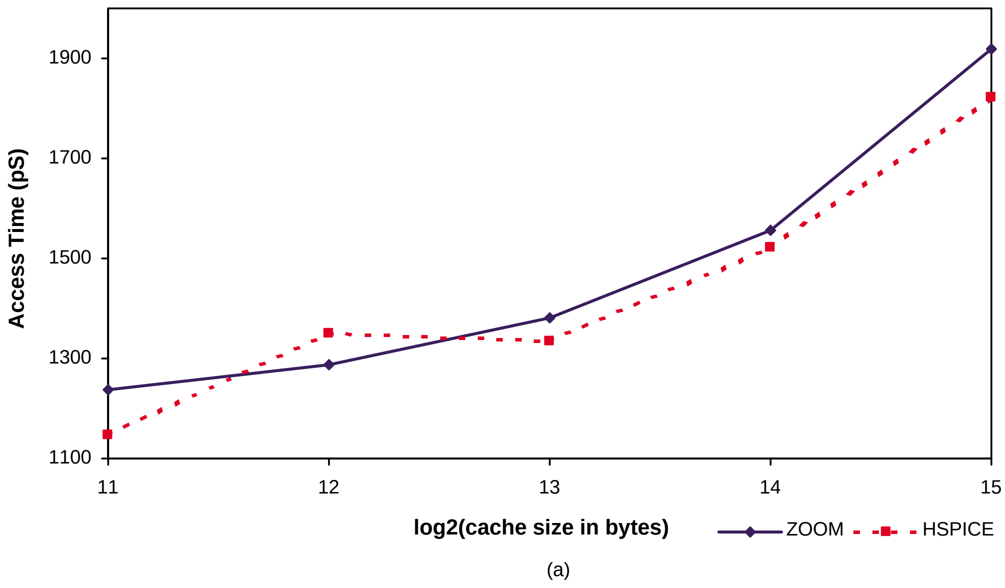
During a two-phase access to an A-way set-associative SRAM cache, $\lceil A/2 \rceil$ tag row decoders and A tag sub-arrays are activated for each access. If a hit is detected, one tag row decoder and data sub-array is activated and only one local bus transitions. The comparators in all but one tag sub-array discharge. The terms for the two-phase access energy in Equation (4.33) are defined as follows for an A-way set-associative SRAM cache:

$$\begin{aligned} E_{tag} &= E_{tag-global-predecoder} + \left\lceil \frac{A}{2} \right\rceil \cdot E_{tag-row-decoder} \\ &+ (E_{tag-wordline} + E_{tag-datapath} + E_{compare}) \cdot A \qquad (4.34) \\ E_{data} &= E_{data-global-predecoder} + E_{data-row-decoder} \\ &+ E_{data-wordline} + E_{word-datapath} \qquad (4.35) \\ E_{routing} &= E_{local\_bus} + E_{global\_routing} \qquad (4.36) \end{aligned}$$

During a CAM cache access, only the global pre-decoder, one CAM sub-array and associated data sub-arrays are activated. The terms for the two-phase access energy in Equation (4.33) are defined as follows for a CAM cache:

$$\begin{aligned} E_{tag} &= E_{global-predecoder} + E_{campath} \qquad (4.37) \\ E_{data} &= E_{data-wordline} + E_{word-datapath} \qquad (4.38) \\ E_{routing} &= E_{local\_bus} + E_{global\_routing} \qquad (4.39) \end{aligned}$$

Figure 4-32 (a) and (b) show the scaling of access time and energy respectively, with cache size for SRAM-based caches using single-phase accesses. All caches are direct-mapped. For the cache sizes observed, the SRAM model estimates access time



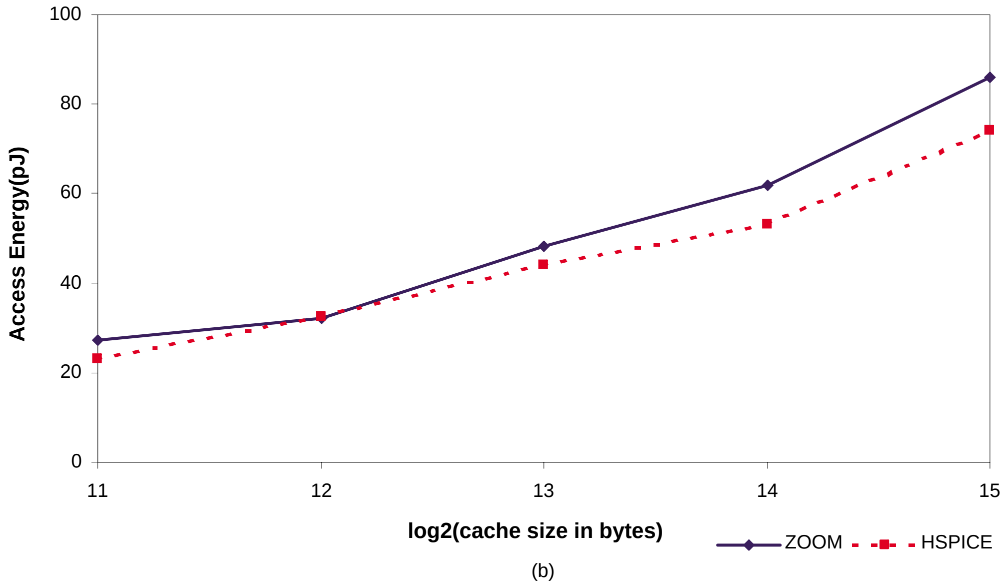


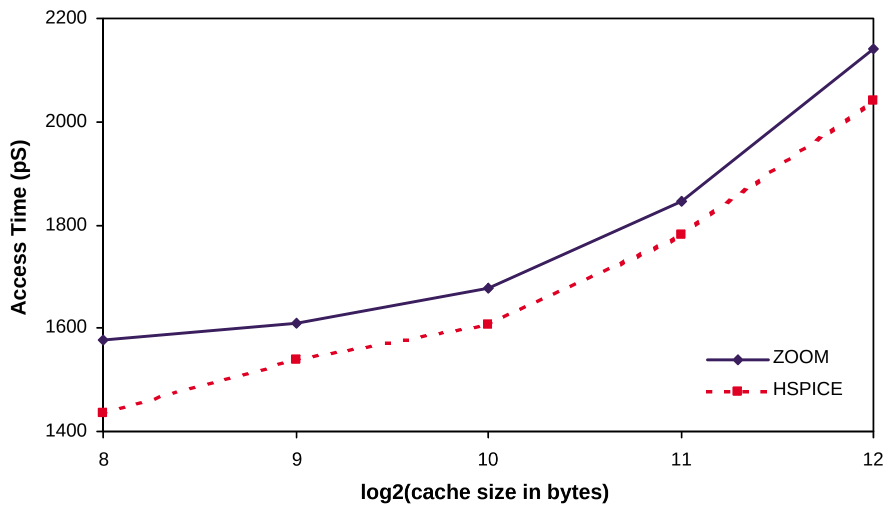
Figure 4-32: (a) Access time scaling with cache size for SRAM cache using single-phase access (b) Access energy scaling with cache size for SRAM cache using single-phase access

to within 8% of HSPICE simulated delays. Access energy estimates are within 15% of HSPICE measurements for cache sizes greater than 1KB.

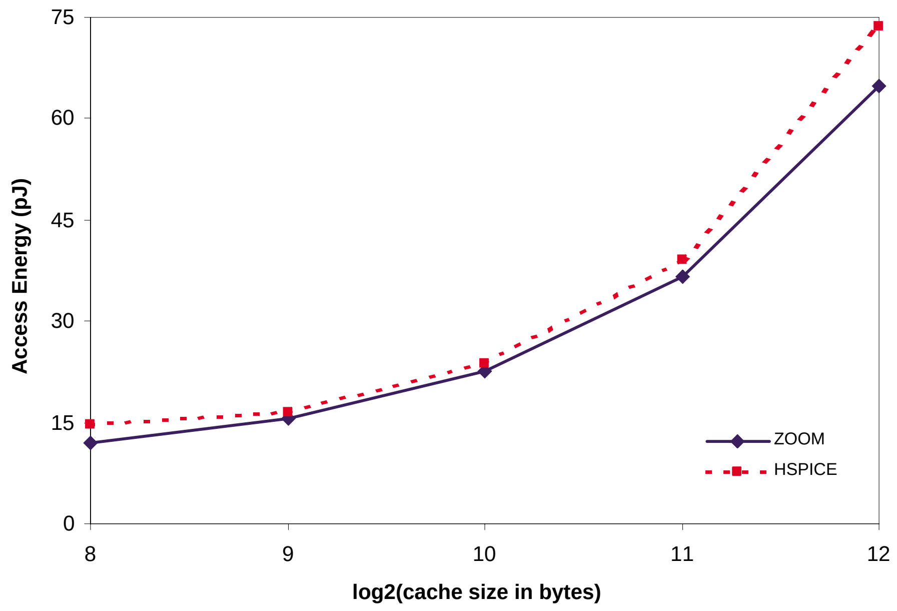
Figure 4-33 (a) and (b) show the scaling of access time and energy respectively, with cache size for CAM-based caches. For the cache sizes observed, the model estimates access time to within 10% of HSPICE simulated delays. Access energy estimates are within 12% of HSPICE measurements.

4.6.2 Applications of the EDA Optimizer

One of the primary uses of the micro-architecture simulator is to study the shape of the energy-delay and area-delay curves. Sample curves produced using the EDA optimizer in ZOOM are shown in Figures 4-34.

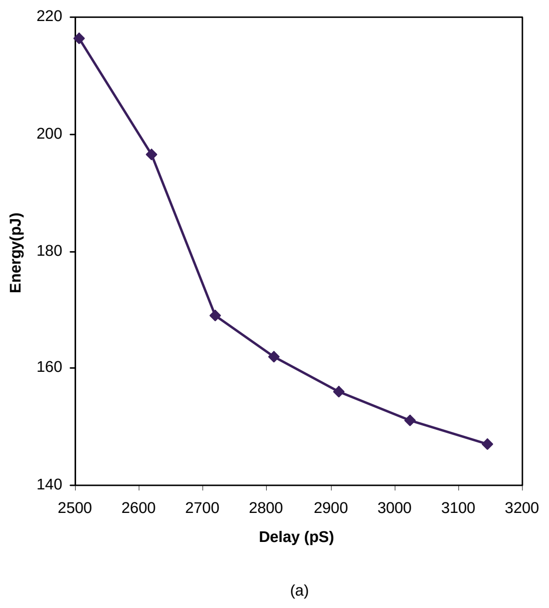


(a)



(b)

Figure 4-33: (a) Access time scaling with cache size for CAM cache (b) Access energy scaling with cache size for CAM cache



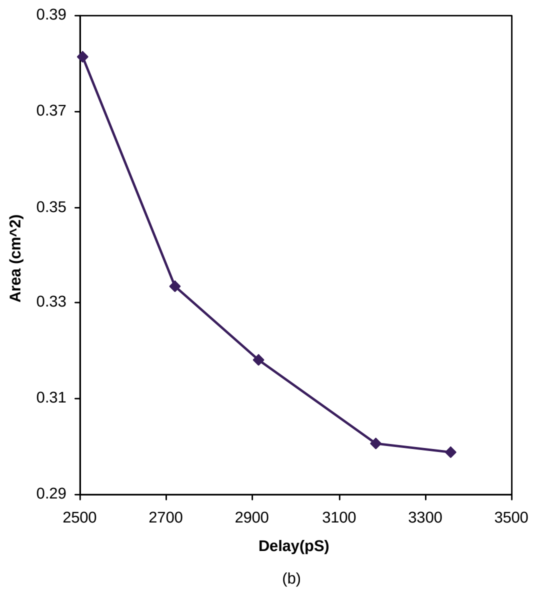


Figure 4-34: (a) Energy vs Delay plot of 256KB, direct-mapped cache (b) Area vs. Delay plot for 256KB, direct-mapped cache

4.7 Comparison with CACTI

CACTI [1] is the most commonly used early-stage cache access time and energy estimating tool. CACTI uses the 0.8μm technology as its base technology. Metrics for caches set in other process generations are estimated by scaling their corresponding metrics in 0.8μm by a "fudge factor" approximately equal to $\frac{0.4}{\lambda_{process}}$. Delay estimates from CACTI are typically within 10% of the delay of energy-efficient cache configurations. However, it over-estimates the energy for these configurations by at least a factor of 10 as observed in [9]. Hence, as part of the evaluation process for ZOOM, energy estimates from the two simulators are compared.

Direct-mapped caches ranging from 8 KB to 256 KB with a blocksize of 32 B and an access width of 32 bits were observed in the 0.25μm technology. Delay estimates from CACTI were used as target delays for the optimizer in ZOOM with a maximum allowed deviation of 5%. The optimization function used in ZOOM was modified to match as closely as possible, the area efficiency of CACTI's optimum configuration. Hence, instead of minimizing the energy-area product (subject to some exponential biases), it minimized the product of energy and deviation of area efficiency from some target efficiency; the target efficiency being CACTI's. The optimization bias for minimizing the deviation was set to an atypical factor of 100 above that of energy to force the optimizer to reject energy-efficient solutions that were not as area-efficient as that proposed by CACTI. Figure 4-35 shows the normalized estimates for the three metrics, with the variables defined as follows:

$$\text{normalized delay} = \frac{\text{Delay from ZOOM}}{\text{Delay from CACTI}}$$

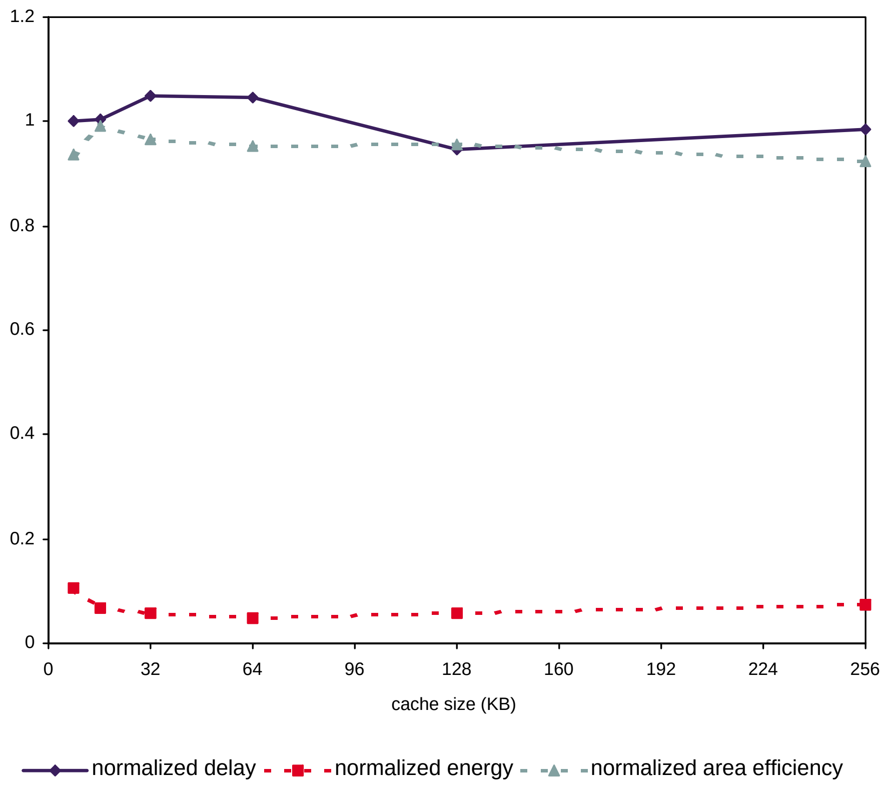


Figure 4-35: Relative delay, energy and area efficiencies for CACTI and ZOOM

$$\text{normalized energy} = \frac{\text{Energy from ZOOM}}{\text{Energy from CACTI}}$$

$$\text{normalized area efficiency} = \frac{\text{Area Efficiency from ZOOM}}{\text{Area Efficiency from CACTI}}$$

The cache configurations proposed by CACTI were up to 8% more area-efficient than those proposed by ZOOM for the same caches sizes and delays. The delay estimates from ZOOM were less than 1% slower on average, with a maximum deviation of 4.9%. The energy estimates from CACTI were between 10 to 20 times higher than estimates from ZOOM, which is corroborated by the observations made in [9].

Chapter 5

Simulator Framework

As mentioned in Chapter 1, ZOOM consists of a Functional Simulator and a Micro-Architecture Simulator. The previous sections focussed mainly on the circuit models used in the Micro-Architecture Simulator. This section presents the software framework supporting the Functional and Micro-Architecture Simulators. ZOOM is implemented in the C programming language and runs on Unix and Linux platforms. The Functional and Micro-Architecture simulators do not share communication channels in this preliminary implementation of ZOOM. However, they are both structured to be highly portable and may be easily integrated if need be. We now proceed to discuss the structures of the two simulators.

5.1 Functional Simulator

The Functional Simulator in ZOOM provides a tractable means for accurately evaluating the effects of architectural changes on multi-level cache performance using a cross-section of benchmarks. Figure 5-1 is an overview of the framework of the functional simulator. The inputs consist of a parameter file for each cache level, and an

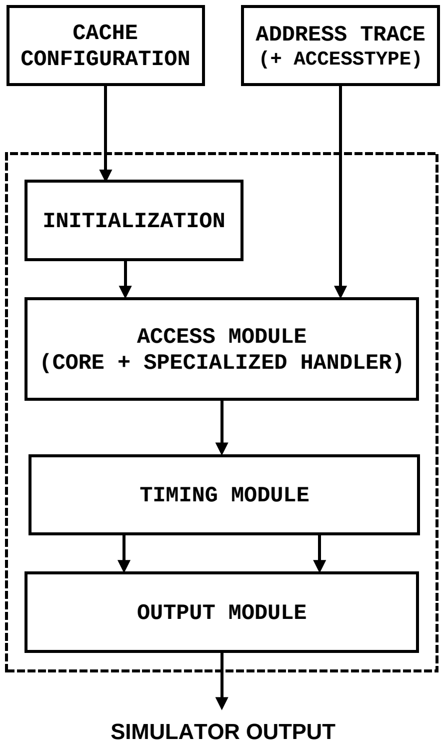


Figure 5-1: Overview of Functional Simulator

address trace which also specifies the access type for each address. The core of the simulator is an access-processing unit which receives and processes in-coming memory references. This core is supported by initialization, timing and output-formatting modules. The simulator returns an output file containing performance statistics and other timing information gathered during simulation. It may be invoked as a stand-alone simulator in a trace-driven simulation using an address trace stored in a file, or it may be ported to a processor simulator in an event-driven simulation. The command line for trace-driven simulation is:

```
zoom <trace file> {cache parameter files} > <output file>
```

where `{cache parameter files}` is a list of parameter files. If ported to a processor simulator, memory requests are passed in from the processor simulator to ZOOM as they arrive. The command line for invoking ZOOM depends on the processor simulator but must include the `{cache parameter files}` input in a format to be specified in Section 5.1.2. The next few paragraphs describe the internal structure of the simulator as well as the contents of the input and output files.

5.1.1 Simulator Components

As shown in Figure 5-1, the functional simulator consists of four main components: the `INITIALIZATION`, `ACCESS-PROCESSING`, `TIMING`, and `OUTPUT-FORMATTING` modules. The `ACCESS-PROCESSING` unit forms the core of the simulator; the other modules are considered auxiliary.

The `INITIALIZATION` module allocates and initializes the state of each cache using values specified in the input parameter files. The `TIMING` module tracks latency effects such as bus traffic and cache port availability. The `OUTPUT-FORMATTING` module is made up of a set of functions that format the statistics gathered during the simulation and prints them to an output file.

The `ACCESS-PROCESSING` unit is made up of a base access unit that fetches and processes new accesses, and a re-configurable `ACCESS_HANDLER` that dictates what actions are taken by the simulator when a hit or miss is detected in a given level of the cache. To improve flexibility without sacrificing usability, a *Read-Allocation* policy is added to the basic cache operation policies. *Read-allocate* is to read misses, what write-allocate is to write misses. A typical L1 cache always fetches new blocks when a read miss occurs and therefore, operates as a read-allocate cache while a Victim buffer [27] and other special-purpose buffers that act as backups to main caches only store blocks that meet some specified criteria and hence, do not operate as read-allocate caches.

The `ACCESS_HANDLER` for a given cache contains two functions: one that is called when a hit occurs in the cache, `<cache>_hit_fn`, and one that is called when a miss occurs, `<cache>_miss_fn`. A set of access functions is needed for each cache simulated in a multi-level cache configuration. Hence, simulating two-level instruction and data caches in one run requires four sets of access functions. ZOOM has built-in access functions for common cache configurations. New functions are therefore only needed if the interaction between caches at different levels deviates from that of say, a traditional two-level cache. Even with specialized access handlers, the basic cache policies (replacement, write-hit, write-allocation, read-allocation) are still in effect and should be appropriately set to avoid conflicting cache state updates, which could result in unpredictable errors.

The re-configurability of the access handlers and the added flexibility of both specialized miss and hit functions, combined with the ability to control when misses result in new blocks being placed in the cache, allows ZOOM to support a wide range of atypical cache configurations without sacrificing usability.

Figure 5-2 shows a flow diagram of the operation of the `ACCESS-PROCESSING` unit. The main machinery of the base access unit is a `cache_access` function whose inputs are a pointer to the cache being accessed, an address, the size of the word to fetch and

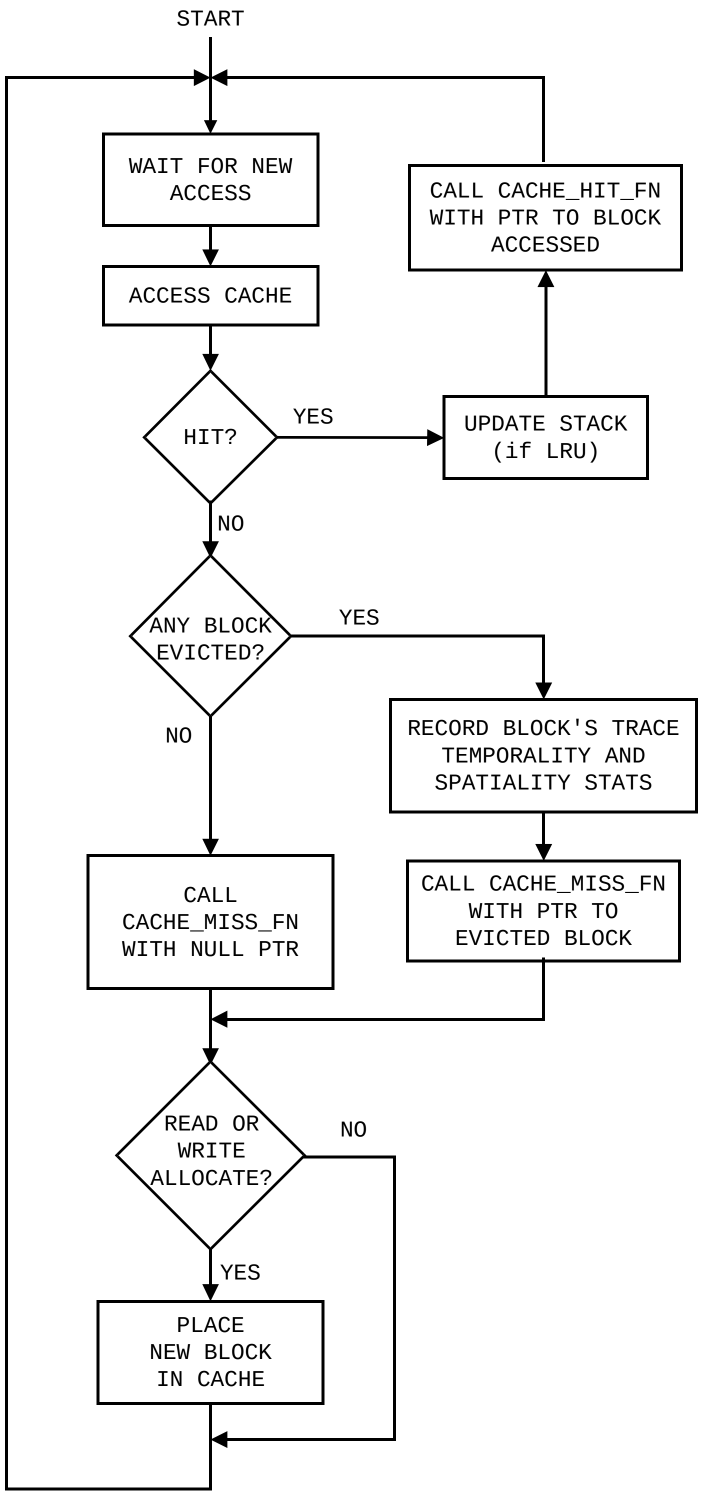

START
WAIT FOR NEW ACCESS
ACCESS CACHE
HIT?
YES
UPDATE STACK (if LRU)
CALL CACHE_HIT_FN WITH PTR TO BLOCK ACCESSED
NO
ANY BLOCK EVICTED?
YES
RECORD BLOCK'S TRACE TEMPORALITY AND SPATIALITY STATS
CALL CACHE_MISS_FN WITH PTR TO EVICTED BLOCK
NO
CALL CACHE_MISS_FN WITH NULL PTR
READ OR WRITE ALLOCATE?
NO
YES
PLACE NEW BLOCK IN CACHE

Figure 5-2: Flow diagram of access-processing in functional simulation

a time of access expressed in cycles. Setting the time of access to a value greater than the current cycle is equivalent to delaying the access for (access-cycle - current cycle-count) cycles. This technique is often used in ZOOM to delay cache state updates resulting from a given access until preceeding ones are completed.

The L1 cache is treated as the *entry-level* cache. Hence, all accesses to multi-level cache configurations begin with an access to the L1 cache. If a hit is detected in the entry-level cache, the state of the data stack is updated (if LRU replacement policy is used). The hit-processing function in the `ACCESS_HANDLER` of the entry-level cache is then invoked with a pointer to the accessed block and other timing information. If a miss is detected, the miss-processing function is invoked with a pointer to the evicted block (or a null pointer if no block was evicted) and the address. The miss-processing function for an L1 cache in a traditional two-level cache configuration would, for instance, initiate an access to the corresponding L2 cache when invoked. If the cache is operating as a read or write allocate cache, the block is placed in the cache. However, it is only available for accessing when the cycle-count equals the value stored in its "block-ready" entry, which is the sum of the access cycle time plus the miss latency of the cache. If the block was already returning to the cache on behalf of a previous miss, the earlier completion time is used.

ZOOM's functional simulator provides a simple interface for porting to various processor simulators. This interface consists of a top-level function each for initialization, access-processing and output-formatting. The initialization function is invoked in the processor simulator's initalization environment. The inputs to the initialization function is the portion of the command line for `{cache parameter files}`. The access function is called with either the L1 data cache or the L1 instruction cache depending on the type of memory reference. The output-formatting function is called when the processor simulator exits the simulation loop. We now describe the inputs and outputs of the functional simulator.

5.1.2 Simulator Inputs and Outputs

The cache parameter file specifies the basic physical properties, timing parameters and strategies/policies that guide the operation of the cache. The details of these groups of inputs are as follows:

physical properties:	the cache size, associativity, blocksize, subblocksize
timing parameters:	hit and miss latencies (in cycles) for each cache level,
	cache-CPU port width, cache-memory bus width
	number of separate or combined read/write ports
cache policies:	replacement policy (LRU, Random, FIFO),
	write-hit policy (copy-back, write-through),
	write allocation policy (write-allocate, no-write-allocate),
	read allocation policy (read-allocate, no-read-allocate)

A parameter file is needed for each cache in a multi-level cache configuration. Thus, a simulation run that uses separate two-level instruction and data caches would need four parameter files (one each for data L1, instruction L1, data L2 and instruction L2); one that simulates a two-level heirarchy with separate instruction and data L1 caches, and a unified L2 cache needs three parameter files. Each parameter file is preceeded by an "orphan" which indicates which cache it belongs to. Supported orphans include:

`-dl1`: Level 1 data cache	`-dl2`: Level 2 data cache
`-il1`: Level 1 instruction cache	`-il2`: Level 2 instruction cache
`-ul1`: Level 1 unified cache	`-ul2`: Level 2 unified cache

The orphans are ordered so that first-level caches are specified before second level caches and within each level, data caches are specified before instruction caches. Any unused cache may be skipped.

The address trace consists of duplexes, each duplex consisting of an *access-type* and a hexidecimal address. The *access-type* is an integer which is 0 if the access is an instruction read, 1 if the access is a data read, 2 for a data write accesses and 3 if the request is to invalidate the data stored at the specified address.

The output file from the simulator includes performance statistics, latency effects and locality-based trace classification.

The performance statistics include the local and global hit, block replacement and write-back rates. The hit rate is defined as the fraction of accesses that result in a hit in the cache. The block replacement rate is the number of valid block replacements divided by the total number of accesses. This fraction approaches the miss rate as the number of accesses increases since there is then a higher probability that every location in the cache holds a valid block, and hence, a valid block will be evicted for each miss. The write-back rate is the number of modified blocks written back to memory divided by the number of memory references. The "dirty bit", which identifies modified blocks is set both at the block and word level. For a given dirty block, only the modified words are written back to memory and hence, only those contribute to bus traffic. The local rates are limited to the particular cache level and hence, only include accesses made to that cache level. The global rates includes are still level-specific, but include all accesses made to all levels in the cache.

Latency effects include *average access latency*, cycle penalties resulting from limited port or bus width and access dependencies (*delayed hits*). The *average access latency* is a commonly used metric for comparing caches. It is defined as

$$\text{Average Access Latency} = \text{Hit Latency} + \text{Miss Rate} \times \text{Miss Penalty}$$

where *miss penalty* is the time it takes to service a regular miss. *Delayed hits* (accesses to data currently returning to the cache on behalf of earlier misses to the same block) are treated as misses for statistics purposes, and included in the miss rate. Their reduced latencies are however accounted for when calculating the average access time.

Trace classification is based on the degree of *spatiality* and *temporality* in the addresses. A block is considered *temporal* if any word in it is accessed more than once during one of the time intervals that it spends in the cache (between an allocation and its subsequent eviction). This time interval is called a *tour* in [30]. A block is considered *spatial* if more than one word is accessed during a tour. A given block can be both temporal and spatial, or neither. We therefore have four classifications [30]. A given block may be:

1. spatial and temporal,
2. spatial and non-temporal,

3. non-spatial and temporal, or
4. non-spatial and non-temporal

The address trace derives its classification from the classification of the blocks. For example, if the majority of the blocks are spatial, the trace is spatial and the degree of spatiality is the ratio of the number of spatial blocks (which may also be temporal) divided by the total number of blocks in the four block classifications. This is easily extended to fine-grain trace classification, along the same lines as the blocks.

5.2 Micro-Architecture Level Simulator

The timing-sensitive Functional Simulator described in Section 5.1 assumes hit and miss latencies in terms of number of cycles without knowing the duration of that cycle, or the absolute access time of the cache. The micro-architecture simulator is a layer of abstraction below that of the functional simulator. It estimates and optimizes the basic metrics of the cache (access time, energy and area) using the circuit-level models presented in Chapters 3 and 4.

Figure 5-3 shows an overview of the micro-architecture simulator. The simulator can model only one single-level cache at a time. The command line for invoking the simulator is:

```
zoom-micro [-net] configuration_file [width_file]
```

Arguments in '`[ ]`' are optional. The `-net` argument invokes the Netlist Generator. If omitted, no netlist is created. The inputs to the simulator consist of a configuration file and an optional transistor-width specifications file. The configuration file

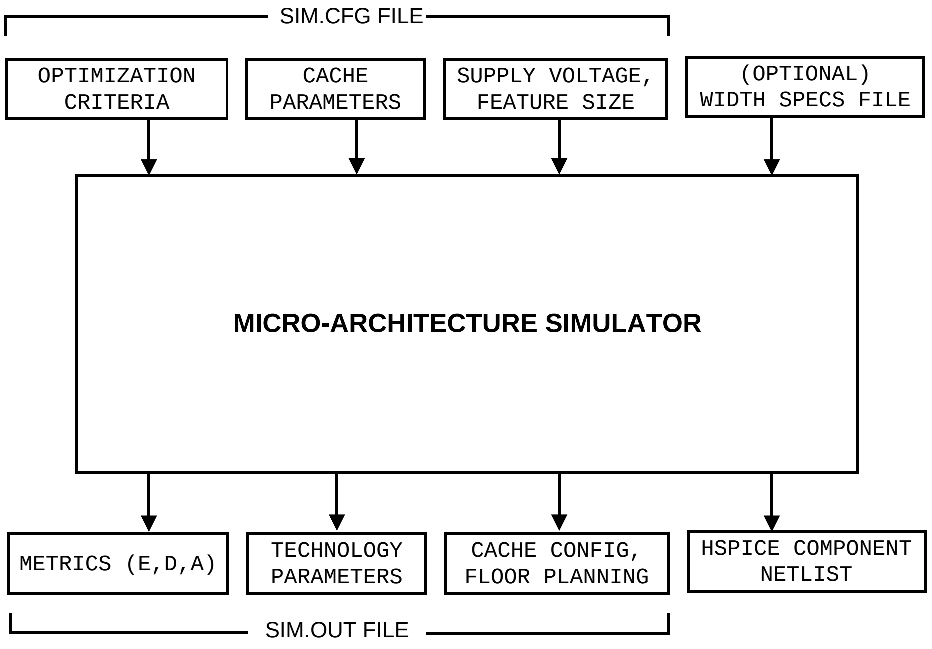


Figure 5-3: Overview of micro-architecture simulator

contains three groups of input parameters: the physical parameters of the cache under study, a set of optimization criteria and a process technology specification consisting of the supply voltage and λ, which is half the minimum feature size. The optional width specification file consists of transistor widths for gates that are not sized during optimization and is used instead of a default widths file. The output file from the simulator contains component-based delay, energy and area estimates; derived configuration information for the proposed cache; and derived technology parameters. If the Netlist Generator was invoked, an HSPICE® file containing netlists for the main components is also returned.

The cache parameters required by the simulator include a name for the cache (for naming the output and netlist files) and the following:

tag type: 1 for SRAM, 2 for CAM, or 0 to use ZOOM's default

sizes (in bytes): cachesize, blocksize, subblocksize, wordsize

associativity: has to be a power of 2 if cache size is a powers of 2

cache ports: the number of read, write, or combined read-write ports

Only single-ported caches with combined read and write ports are supported in this version of ZOOM. The optimization criteria consist of a target delay in nanoseconds, the maximum allowed deviation from this target (as a fraction of the target delay), and β and α, which are integers specifying the weights attached to energy and delay, respectively, during optimization.

Figure 5-4 shows the simulation flow of the micro-architecture simulator. The supply voltage and feature size are used to generate basic process parameters from the base 0.25μm technology using well-defined scaling rules. These parameters are then fed, together with the cache configuration, to analytical energy, delay and area models to obtain estimates which are used as a starting point for optimization. This preliminary implementation of the optimizer performs an exhaustive serial search through all available configuration options to identify the optimium configuration. Hence, the speed of the optimization is linearly dependent on the number of choices, which grows quickly with cache size. Future implementations will employ some parallelism to improve optimization speed at the expense of extra system memory.

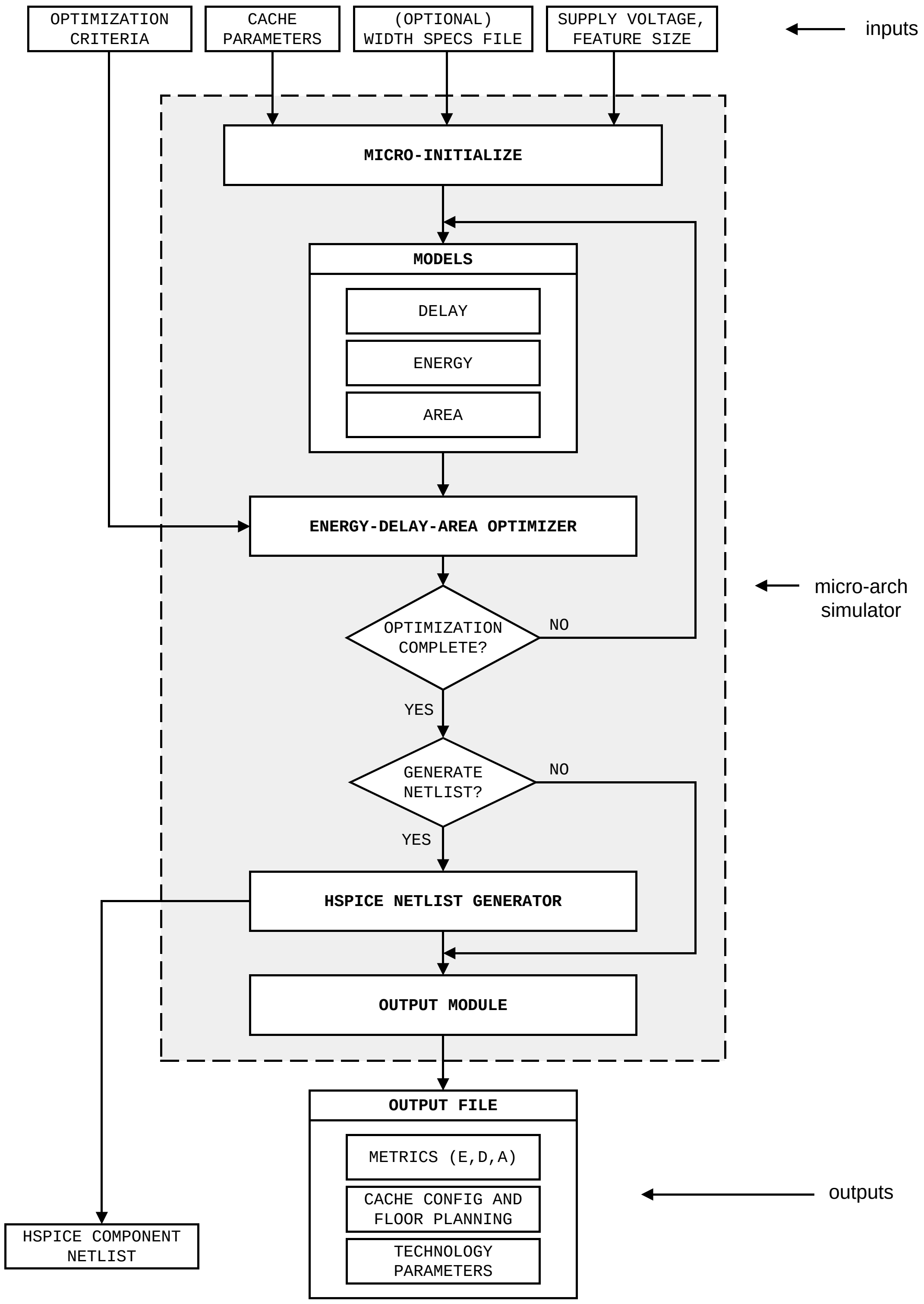


Figure 5-4: Flow of micro-architecture level simulation

5.2.1 Automated Component Netlist Generation

Since it is anticipated that designers and researchers may want to use the cache model proposed by ZOOM as a starting point for their designs, extracting gate and circuit parameters in some machine-readable format is desirable. This is the motivation for including HSPICE® netlist-generation capabilities in ZOOM. The returned netlist is not intended for use as a complete functional cache. It is intended, mainly, to assist in validating the estimates returned by ZOOM.

Netlist generation may be invoked using the `-net` commandline option. The netlist generator assumes library cell definitions for basic cells such as the SRAM and CAM cells and simple gates such as inverters, NAND gates, multiplexors, sense amplifiers, etc. The netlist is generated heirarchically by building smaller sub-circuits and assembling instances of these sub-circuits to form the component. Interconnect resistances and capacitances are distributed at bit-level granulity. The generated netlist contains all the components needed to build a "sub-cache." The sub-cache has an instance of a fully connected SRAM data path for both data and tag arrays, a tag and data decoder, comparators and the local output bus. It assumes control signals such as the precharge enable, the sense clock and the evaluate signal used by the SRAM tag comparators are available and independently generated to arrive at the appropriate time. However, it does not create a netlist for the control circuitry needed to generate these signals. The current implementation does not generate a netlist for global routing or assemble instances of these "sub-caches" into the complete cache. However, future versions of ZOOM may have this extended capability.

Chapter 6

Conclusion

This thesis presented the implementation and evaluation of ZOOM, a framework for characterizing and optimizing SRAM and CAM-based caches. Various energy-reduction techniques were modeled and used in a general optimization scheme to propose energy-efficient cache models for a given target delay. Preliminary evaluation of the circuit-level models used in the micro-architecture simulator show that delay estimates are within 10% of HSPICE simulated delays for both SRAM and CAM caches and energy estimates are within 15% of HSPICE measurements for both types of caches while measurements show that ZOOM is at least 1,000,000 times faster than HSPICE. Together, the functional and micro-architecture simulators provide designers a tractable means to evaluate the effects of architectural changes on cache energy and performance using a cross-section of benchmarks that are representative of target applications.

6.1 Future Plans

Future plans consist mainly of continued evaluation and fine-tuning of the models. The next phase of evaluation is to examine how closely the sizing algorithms and assumptions about aspects of the cache design match those of actual (commercial) cache desgins, and fine-tune these assumptions to improve the value of the model for commercial cache designers. Since leakage in SRAM and CAM arrays is becoming a dominant issue, an expanded leakage model in ZOOM will be valuable. The tests in this thesis were run using the 0.25μm technology. The accuracy of the simulator needs to be verified across technology boundaries due to non-linear scaling effects and the models and base assumptions improved to accurately represent these effect.

Appendix A

General Simulator Information

A.1 Running the Simulator

The command line for trace-driven simualtion is:

```
zoom <trace> [-opt1] <ccf1> [-opt2] <ccf2> [-opt3] <ccf3> [-opt4] <ccf4>
```

where ccf1...4 are the cache configuration files and the allowed values for [-opt1], [-opt2], [-opt3] and [-opt4] are as follows:

```
-opt1:  -dl1|-il1|-ul1

-opt2: blank|-il1|-dl2|-il2|-ul2

-opt3: blank|-dl2|-il2|-ul2

-opt4: blank|-il2
```

Most realistic configurations are supported. These include configurations with separate or unified instruction and data caches of up to 2 levels. Some supported configuration examples are provided below:

```
zoom <trace> -dl1 <cachefile> -il1 <cachefile>
zoom <trace> -ul1 <cachefile> -ul2 <cachefile>

zoom <trace> -dl1 <cachefile> -il1 <cachefile> -ul2 <cachefile>

zoom <trace> -dl1 <dl1_file> -il1 <il1_file> -dl2 <dl2_file> -il2 <il2_file>
```

A.2 Sample I/O Files for Functional Simulator

Sample Input File

```
16384 #cachesize_in_bytes
32    #blocksize_in_bytes
32    #subblocksize_in_bytes
2     #cache_associativity_(number_of_elements_per_set)
1     #cache_read_latency
1     #cache_write_latency
r     #cache_replacement_policy:l=LRU,f=FIFO,r=RANDOM
d     #cache_fetch_policy:d=DEMAND,for_others_see_fetch.c
c     #cache_write_hit_policy:w=writethrough,c=copyback
w     #cache_write_miss_policy:w=writeallocate,n=non-allocating
0     #cache_read-allocate-policy:=1_if_special_purpose_buffer;0_otherwise

10    #main_memory_read_latency
1     #use_r/w_ports:1_for_combined_r/w_ports,0_for_separate
1     #number_of_r/w_cache_ports
0     #number_of_read_cache_ports
0     #number_of_write_cache_ports
1     #return_policy:1_for_requested_word_first,0_for_first_word_first
8     #CPU-to-cache_bus_width_in_bytes
32    #cache-to-memory_bus_width_in_bytes
100   #NOA;set_high_for_nonblocking

0     #TIMING_MODEL;0=Full_LE,1=LE-Nominal,2=NO_timing
1000000     #simulation_cutoff
```

Sample Output File

```
Date and Time of Simulation: Apr  4 2003 19:56:45

Data L1 Cache Configuration
----------------------------

cache size         = 16.0K
associativity      = 2
number of subbanks = 1 (16.0K/subbank)
read latency       = 1 cycle(s)
write latency      = 1 cycle(s)
replacement policy = r
blocksize          = 32B
subblocksize       = 32B

Inst. L1 Cache Configuration
----------------------------

cache size         = 16.0K
associativity      = 2
number of subbanks = 1 (16.0K/subbank)
read latency       = 1 cycle(s)
write latency      = 1 cycle(s)
replacement policy = r
blocksize          = 32B
subblocksize       = 32B

General Memory Properties
--------------------------
main memory latency  = 10 cycles
# read/write ports = 1
word return policy = requested_word_first
c-p buswidth       = 8B
m-c buswidth       = 32B

cache blocks after 100 outstanding access

Simulation Setup
----------------
Using Full LE timing model
Maximum # of Instructions to simulate: 1000000

Reading from trace file... test_trace.in

---Simulation begins....
```

```
---Simulation complete.
Used Cycles: 2723000.0cycles
Real Time:      3.00seconds


***** Simulation Results for Data L1  Cache *****

cache size         = 16.0K
associativity      = 2
number of subbanks = 1 (16.0K/subbank)
read latency       = 1 cycle(s)
write latency      = 1 cycle(s)
replacement policy = r
blocksize          = 32B
subblocksize       = 32B

accesses:   400000
reads   :   244692
writes  :   155308
hits    :   212957
delayed hits  :   134757 (treated as misses in metrics)

Parameter      Quantity     local rate     global rate
----------     ---------     ----------     ----------
misses            187043         0.4676        0.4676
readmisses         33951         0.1387
writemisses        18335         0.1181
writebacks         34545         0.0864        0.0864
block replacements 51774         0.1294        0.1294

total cycles:       1641324
avg. latency (# of cycles):    4.1033

Block Classification
--------------------
NTNS:      45.25%
NTS:       10.84%
TNS        26.55%
TS:        17.36%

Trace Classification
--------------------
%Spatial= 28.2 %Temporal = 43.91
```

```
***** Simulation Results for Inst L1 Cache *****

cache size         = 16.0K
associativity      = 2
number of subbanks = 1 (16.0K/subbank)
read latency       = 1 cycle(s)
write latency      = 1 cycle(s)
replacement policy = r
blocksize          = 32B
subblocksize       = 32B

accesses:   243888
reads   :   243888
writes  :        0
hits    :   242085
delayed hits  :      562 (treated as misses in metrics)

Parameter      Quantity     local rate     global rate
----------     ---------     ----------     ----------
misses            1803        0.0074         0.0074
readmisses        1241        0.0051
writemisses          0        0.0000
writebacks           0        0.0000         0.0000
block replacements 876        0.0036         0.0036

total cycles:        259802
avg. latency (# of cycles):        1.0653

Block Classification
--------------------
NTNS:      6.75%
NTS:       21.38%
TNS        31.47%
TS:        40.40%

Trace Classification
--------------------
%Spatial= 61.78 %Temporal = 71.87


Overall Summary
---------------
Total # of Inst Memory References:        243888
Total # of Data Memory Refs:              400000
Total # of Memory Refs:                   643888
Total # of Hits:                          455042
Extra Port Cycles:                             0
```

A.3 Sample I/O Files for Micro-Arch Simulator

Sample Input file

```
  0.125   #lambda_in_microns
  2.5     #Vdd_in_volts
  test    #name_for_the_cache(used_to_name_output_file_<40_characters)
  1       #tag_array_type;0_for_ZOOMDECIDE;1_for_SRAM;2_for_CAM
  32768   #cachesize_in_bytes
  1       #cache_associativity_(number_of_elements_per_set)
  32      #blocksize_in_bytes
  32      #subblocksize_in_bytes
  4       #wordsize_bytes
  0       #number_of_read_cache_ports
  0       #number_of_write_cache_ports
  1       #number_of_read_write_ports
  2.0     #target_delay_in_ns;
  .05     #tolerance_as_a_fraction_(eg,.1)
  4       #energy_bias(has_to_be_an_integer);
  1       #area_bias(has_to_be_an_integer);
```

Sample Output file

```
Date and Time of Simulation: May 27 2003 00:50:20
command line: micro-zoom cache.cfg

1. CACHE PARAMETERS
===================
Cache Type = SRAM-Tag cache
Cache size(KB) = 32.0
Associativity =  1
Blocksize(B) = 32
Subblocksize(B) = 32
# of r_ports =  0
# of w_ports =  0
# of r/w ports =  1
Cache uses LATE SELECT

OPT. CRITERIA: max_delay=2.0000, tolerance = 0.0500 E_bias=4, A_bias=1
OPTIMIZATION REPORT: optimum delay=2.0949, deviation = 0.0475

2. MICRO-ARCHITECTURE LEVEL SIMULATION RESULTS
==============================================
 Read Access Time(with global routing) =  2.0949ns
 Read Access Time(w/o global routing)  =  1.9465ns
```

```
  Tag path Delay(ns)         =  1.9171
  Data path Delay(ns)        =  1.7572
  Data Routing Delay(ns)     =  0.1485
Write Access Time (ns) =  3.0741

Read Access Energy(nJ) =  0.0476
Write Access Energy(nJ)=  0.0476
  Address Drivers(nJ)       =  0.0091
  Data Routing Energy(nJ) =  0.0000

Total cache area(mm^2) =  4.0290
  floorplan (rows x cols): [2 x 4] subarrays
  Cache Aspect Ratio =   1.53
  Area Efficiency = 0.6710

Energy & Delay Breakdown by Component
-------------------------------------
TAGPATH
  component            delay(ns)       energy(nJ)      % total energy
  ---------            ---------       ----------      ---------------
  Decoder:                1.1920          0.0078            16.5
  Local Wordline:         0.0657          0.0001             0.2
  Bitline & Col Muxes:    0.1761          0.0005             1.0
  Sense Amplifiers:       0.2328          0.0022             4.5
  Comp & mux driver:      0.2506          0.0004             0.8


DATAPATH
  component            delay(ns)       energy(nJ)       % total energy
  ---------            ---------       ----------       ---------------
  Decoder:                0.9335          0.0112            23.6
  Local Wordline:         0.1046          0.0004             0.8
  Bitline & Col Muxes:    0.4863          0.0102            21.6
  Sense Amplifiers:       0.2328          0.0038             8.1
  Local Output Bus:       0.1892          0.0023             4.9


3. DERIVED COMPONENT PARAMETERS
===============================
 ARRAY_PARTITIONING
 ------------------
  no. of data sub-arrays = 8 [4096B, 128 cols by 256 rows]
  no. of tag sub-arrays = 16 [216B, 27 cols by 64 rows]
    1 x (17 tagbits + 9 dirty bits + 1 valid bit) per row
  no. of sets per sub-array = 256
 DECODER
 =======

 DATA_PREDECODER
 ---------------
  no. of repeaters = 0
  repeater scaling (from min width) =  0.00

  no. of bits = 2
  stage fanout =5.06
  no. of stages = 2
  stage sizes(lambda) =  6 21

 DATA ROW DECODER
 ----------------
  no. of bits = 8
  stage fanout =5.06
  no. of stages = 5
  stage sizes(lambda) =  3 3 10 54 3

 TAG_PREDECODER
 ---------------
  no. of repeaters = 0
  repeater scaling (from min width) =  0.00

  no. of bits = 4
  stage fanout =5.06
  no. of stages = 4
  stage sizes(lambda) =  3 6 6 38

 TAG ROW DECODER
 ----------------
  no. of bits = 6
  stage fanout =5.06
  no. of stages = 5
  stage sizes(lambda) =  3 3 6 34 6

4. TECHNOLOGY SCALING PARAMETERS
================================
 Lambda(um) =   0.125(L_eff =  0.250)
 Vdd(V) =   2.50
 Rg_inv(KOhm-um)=   3.56
 Cg_inv(fF/um) =   1.29
 Cj_inv(fF/um) =   1.72
 Tfo4(pS) =  90.00
 T_int(pS) =  20.00
 I_nmos(uA/um) = 508.60
```

Bibliography

[1] S. Wilton, N. P. Jouppi. An Enhanced Access and Cycle Time Model for On-chip Caches. *Technical Report 93/5, Compaq Western Research Lab*, July 1994

[2] T. Wada, S. Rajan, and S. A. Przybylski. An Analytical Access Time Model for On-chip Cache Memories,*IEEE Journal of Solid-State Circuits, vol. 27, pp. 1147-1156*, August 1992.

[3] R. J. Evans and P. D. Franzon. "Energy consumption modeling and optimization for SRAMs, *IEEE J. Solid-State Circuits, vol. 30, pp. 571-579*, May 1995.

[4] R. J. Evans. Energy consumption modeling and optimization for SRAMs, *PhD dissertation, Dept. of Electrical and Computer Engineering, North Carolina State Univ.*, July 1993.

[5] N. Bellas, I. Hajj, C. Polychronopoulos. An analytical, transistor-level energy model for SRAM-based caches. *IEEE*

[6] B. S. Amrutur, M. A. Horowitz. Speed and power scaling of SRAMs, *IEEE Transactions on solid-state circuits, vol. 35, No. 2*, February 2000

[7] K. Ghose and M. B. Kamble. Reducing Power in Super-scalar Processor Caches using Subbanking, Multiple Line buffers and Bitline Segmentation.*International Symposium on Low-Power Electronic Devices.pp 70-75*, August 1999

[8] B. S. Amrutur Design and Analysis of Fast Low Power SRAMs. *PhD Thesis Stanford University*, August 1999

[9] M. Zhang, K. Asanovic. Highly-associative caches for low-power processors, *Kool Chips Workshop, 33rd International Symposium on Microarchitecture*, December 2000

[10] S. Furber *et al.* ARM3 - 32b RISC processor with 4kbyte on-chip cache. *In G. Musgrave and U. Lauther, editors, Proc. IFIP TC 10/WG 10.5 Int. Conf. on VLSI, pp 35-44. Elsevier (North Holland)*, 1989

[11] B. S. Amrutur, M. A. Horowitz. Fast low-power decoders for RAMs, *IEEE Journal of Solid-State Circuits, vol. 36, No. 10*, October 2001

[12] I. E. Sutherland, R. F. Sproull. Logical Effort: Designing for speed on the back of an envelope, *Advanced Research in VLSI*, 1991

[13] M. A. Horowitz. Timing models for MOS Circuits. *Technical Report, SEL83-303. Integrated Circuits Laboratory, Stanford University*, 1983

[14] Victor Zyuban. "Excerpts from the MS Thesis of Victor Zyuban"

[15] V. Zyuban, P. Kogge. Energy Complexity of Register Files, *International Symposium on Low-Power Electronic Devices*, Aug 1998

[16] K. Itoh *et. al* Trends in Low-Power RAM circuit technologies; *Proceeding of the IEEE, V83, N4*, 1995

[17] A. Karandikar, K.K. Parhi. Low Power SRAM Design using Hierarchical Divided Bit-line Approach, *ICCD*, Oct '98

[18] S. Wilton, N.P. Jouppi. CACTI 2.0: An Enhanced Access and Cycle Time Model for On-chip Caches. *Technical Report 02, Compaq Western Research Lab*, July 2001

[19] Regina Sam, ZOOM: A Flexible Cache Power and Performance Estimating Tool. *MIT Advanced Undergraduate Project Report* May '02

[20] Regina Sam, Proposed Energy-Delay Models for Micro-Architecture Level Simulation in ZOOM. *Summer 2002 Project report*

[21] A. Efthymiou, J. D. Garside. An Adaptive Serial-Parallel CAM Architecture for Low-Power Cache Blocks, *International Symposium on Low-Power Electronic Devices*, August 2002

[22] I. Hsiao, D. Wang, C. Jen. Power Modeling and Low-Power Design of Content Addressable Memories. *IEEE Proceedings on International Symposium on Circuits and Systems. pp 926-929*, May 2001

[23] J. L. Hennessy and D. Patterson. Computer Architecture: A Quantitative Approach, *2nd Edition Morgan Kaufmann Publishers*

[24] M. B. Kamble, K. Ghose. Analytical Energy Dissipation Models for Low-Power Caches. *International Symposium on Low-Power Electronic Devices.pp 143-148*, August 1997

[25] L. T. Clark et. al. An Embedded 32-b Microprocessor Core for Low-Power and High-Performance Applications. *IEEE Journal of Solid-State Circuits, Vol. 36, No. 11 pp 1599-1608*, November 2001

[26] R. A. Uhlig, T. N. Mudge. Trace-Driven Memory Simulation: A Survey. *ACM Computing Surveys, Vol. 29, No. 2, pp 128-170*, June 1997

[27] N. P. Jouppi, Improving Direct-Dapped Dache Performance by the Addition of a Small, Fully Associative Cache and Prefetch Buffers, *17th Annual International Symposium on Computer Architecture* May 1990 pp. 364-373

[28] D. Burger and T. M. Austin, Evaluating Future Microprocessors: the SimpleScalar Tool Set, *Technical Report #1342, University of Wisconsin*, June 1997.

[29] M. D. Hill, DineroIII Documentation, *Unpublished UNIX-style Man Page, University of California, Berkeley*, October 1985.

[30] E. S. Tam *et. al* mlcache: A Flexible Multi-Lateral Cache Simulator *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* Jul 1998 pp 19-26

[31] J. M. Rabeay, A. Chandrakasan, B. Nikolic. Digital Integrated Circuits: A Design Perspective, *2nd Edition*

[32] 1997 National Technology Roadmap for semiconductor

[33] H. B. Bakoglu and J. D. Meindl, Optimal interconnection circuits for VLSI, *IEEE Trans. Electron Devices, vol. ED-32, pp 903-909*, May 1985

[34] Berkeley Predictive Technology Model. Online Document source: *http://www-device.eecs.berkeley.edu/~ptm/*, Version Date: 7/31/2002

[35] T. Muzino et. al. Experimental Study of Threshold Voltage fluctuation due to statistical variation of channel dopant number in MOSFETs. *IEEE Transactions on Electron Devices, vol 41 pp 2216-2221*, Nov 1994

[36] W. C. Elmore. The Transient Response of Damped Linear networks with particular Regard to WideBand Amplifiers.*Journal of Applied Physics, vol. 19, pp 55-63*, 1948

[37] B.S. Amrutur, M. A. Horowitz. A Replica Technique for Wordline and Sense Control in Low-Power SRAMs.*IEEE Journal on Solid-State Circuits, vol. 33, pp 1208-1219*, Aug 1998

[38] R. C. Jaeger, Comments on 'An optimized output stage for MOS integrated circuits', *IEEE Journal of Solid State Circuits, vol. SC-10, no. 3, pp. 185-186*,June 1975.

[39] N. C. Li, et. al., CMOS tapered buffer, *IEEE Journal of Solid State Circuits, vol. 25, no. 4, pp. 1005-1008*, August 1990.

[40] J. Choi, et. al., Design of CMOS tapered buffer for minimum power-delay product, *IEEE Journal of Solid State Circuits, vol. 29, no. 9, pp. 1142-1145*, September 1994.

[41] B. S. Cherkauer and E. G. Friedman, A unified design methodology for CMOS tapered buffers, *IEEE Journal of Solid State Circuits, vol. 3, no. 1, pp. 99-110*, March 1995.

[42] C. L. Portmann and H. Y. Meng, Metastability in CMOS library elements in reduced supply and Technology applications, *IEEE Journal of Solid State Circuits, vol. 30, no. 1, pp. 39-46*, Jan 1995.