JULY 1994

WRL
Research Report 93/5

An Enhanced Access and Cycle Time Model for On-Chip Caches

Steven J.E. Wilton and Norman P. Jouppi

digital **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

The Western Research Laboratory (WRL) is a computer systems research group that was founded by Digital Equipment Corporation in 1982. Our focus is computer science research relevant to the design and application of high performance scientific computers. We test our ideas by designing, building, and using real systems. The systems we build are research prototypes; they are not intended to become products.

There is a second research laboratory located in Palo Alto, the Systems Research Center (SRC). Other Digital research groups are located in Paris (PRL) and in Cambridge, Massachusetts (CRL).

Our research is directed towards mainstream high-performance computer systems. Our prototypes are intended to foreshadow the future computing environments used by many Digital customers. The long-term goal of WRL is to aid and accelerate the development of high-performance uni- and multi-processors. The research projects within WRL will address various aspects of high-performance computing.

We believe that significant advances in computer systems do not come from any single technological advance. Technologies, both hardware and software, do not all advance at the same pace. System design is the art of composing systems which use each level of technology in an appropriate balance. A major advance in overall system performance will require reexamination of all aspects of the system.

We do work in the design, fabrication and packaging of hardware; language processing and scaling issues in system software design; and the exploration of new applications areas that are opening up with the advent of higher performance systems. Researchers at WRL cooperate closely and move freely among the various levels of system design. This allows us to explore a wide range of tradeoffs to meet system goals.

We publish the results of our work in a variety of journals, conferences, research reports, and technical notes. This document is a research report. Research reports are normally accounts of completed research and may include material from earlier technical notes. We use technical notes for rapid distribution of technical material; usually this represents research in progress.

Research reports and technical notes may be ordered from us. You may mail your order to:

Technical Report Distribution
DEC Western Research Laboratory, WRL-2
250 University Avenue
Palo Alto, California 94301 USA

Reports and notes may also be ordered by electronic mail. Use one of the following addresses:

Digital E-net:	`JOVE::WRL-TECHREPORTS`
Internet:	`WRL-Techreports@decwrl.pa.dec.com`
UUCP:	`decpa!wrl-techreports`

To obtain more details on ordering by electronic mail, send a message to one of these addresses with the word ‘‘`help`’’ in the Subject line; you will receive detailed instructions.

An Enhanced Access and Cycle Time Model

for On-Chip Caches

Steven J.E. Wilton and Norman P. Jouppi

July, 1994

Abstract

This report describes an analytical model for the access and cycle times of direct-mapped and set-associative caches. The inputs to the model are the cache size, block size, and associativity, as well as array organization and process parameters. The model gives estimates that are within 10% of Hspice results for the circuits we have chosen.

Software implementing the model is available from DEC WRL.

digital **Western Research Laboratory** 250 University Avenue Palo Alto, California 94301 USA

Table of Contents

List of Figures

List of Tables

1. Introduction

Most computer architecture research involves investigating trade-offs between various alternatives. This can not adequately be done without a firm grasp of the costs of each alternative. As an example, it is impossible to compare two different cache organizations without considering the difference in access or cycle times. Similarly, the chip area and power requirements of each alternative must be taken into account. Only when all the costs are considered can an informed decision be made.

Unfortunately, it is often difficult to determine costs. One solution is to employ analytical models that predict costs based on various architectural parameters. In the cache domain, both access time models [8] and chip area models [5] have been published. In [8], Wada et al. present an equation for the access time of a cache as a function of various cache parameters (cache size, associativity, block size) as well as organizational and process parameters. In [5], Mulder et al. derive an equation for the chip area required by a cache using similar input parameters.

This report describes an extension of Wada's model. Some of the new features are:

- an additional array organizational parameter
- improved decoder and wordline models
- pre-charged and column-multiplexed bitlines
- a tag array model with comparator and multiplexor drivers
- cycle time expressions

The goal of this work was to derive relatively simple equations that predict the access/cycle times of caches as a function of various cache parameters, process parameters, and array organization parameters. The cache parameters as well as the array organization parameters will be discussed in Section 4. The process parameters will be introduced as they are used; Appendix II contains the values of the process parameters for a 0.8μm CMOS process [3].

Any model needs to be validated before the results generated using the model can be trusted. In [8], a Hspice model of the cache was used to validate their analytical model. The same approach was used here. Of course, this only shows that the model matches the Hspice model; it does not address the issue of how well the assumed cache structure (and hence the Hspice model) reflects a real cache design. When designing a real cache, many circuit tricks could be employed to optimize certain stages in the critical path. Nevertheless the relative access times between different configurations should be more accurate than the absolute access times, and this is often more important for optimization studies.

The model described in this report has been implemented, and the software is available from DEC WRL. Section 2 explains how to obtain and use the software. The remainder of the report explains how the model was derived. For the user who is only interested in using the model, there is no need to read beyond Section 2.

2. Obtaining and Using the Software

A program that implements the model described in this report is available. To obtain the software, log into gatekeeper.dec.com using anonymous ftp. (Use "anonymous" as the login name and your machine name as the password.) The files for the program are stored together in "/archive/pub/DEC/cacti.tar.Z". Get this file, "uncompress" it, and extract the files using "tar".

The program consists of a number of C files; *time.c* contains the model. Transistor widths and process parameters are defined in *def.h*. A makefile is provided to compile the program.

Once the program is compiled, it can be run using the command:

```
cacti C B A
```

where *C* is the cache size (in bytes), *B* is the block size (in bytes), and *A* is the associativity. The output width and internal address width can be changed in *def.h*.

When the program is run, it will consider all reasonable values for the array organization parameters (discussed in Section 4) and choose the organization that gives the smallest access time. The values of the array organization parameters chosen are included in the output report.

3. Cache Structure

Before describing the model, the internal structure of an SRAM cache will be briefly reviewed. Figure 1 shows the assumed organization. The decoder first decodes the address and selects the appropriate row by driving one wordline in the data array and one wordline in the tag array. Each array contains as many wordlines as there are rows in the array, but only one wordline in each array can go high at a time. Each memory cell along the selected row is associated with a pair of bitlines; each bitline is initially precharged high. When a wordline goes high, each memory cell in that row pulls down one of its two bitlines; the value stored in the memory cell determines which bitline goes low.

Each sense amplifier monitors a pair of bitlines and detects when one changes. By detecting which line goes low, the sense amplifier can determine what is in the memory cell. It is possible for one sense amplifier to be shared among several pairs of bitlines. In this case, a multiplexor is inserted before the sense amps; the select lines of the multiplexor are driven by the decoder. The number of bitlines that share a sense amplifier depends on the layout parameters described in the next section. Section 6.4 discusses this further.

The information read from the tag array is compared to the tag bits of the address. In an *A*-way set-associative cache, *A* comparators are required. The results of the *A* comparisons are used to drive a valid (hit/miss) output as well as to drive the output multiplexors. These output multiplexors select the proper data from the data array (in a set-associative cache or a cache in which the data array width is larger than the output width), and drive the selected data out of the cache.

It is important to note that there are two potential critical paths in a cache read access. If the time to read the tag array, perform the comparison, and drive the multiplexor select signals is larger than the time to read the data array, then the tag side is the critical path, while if it takes

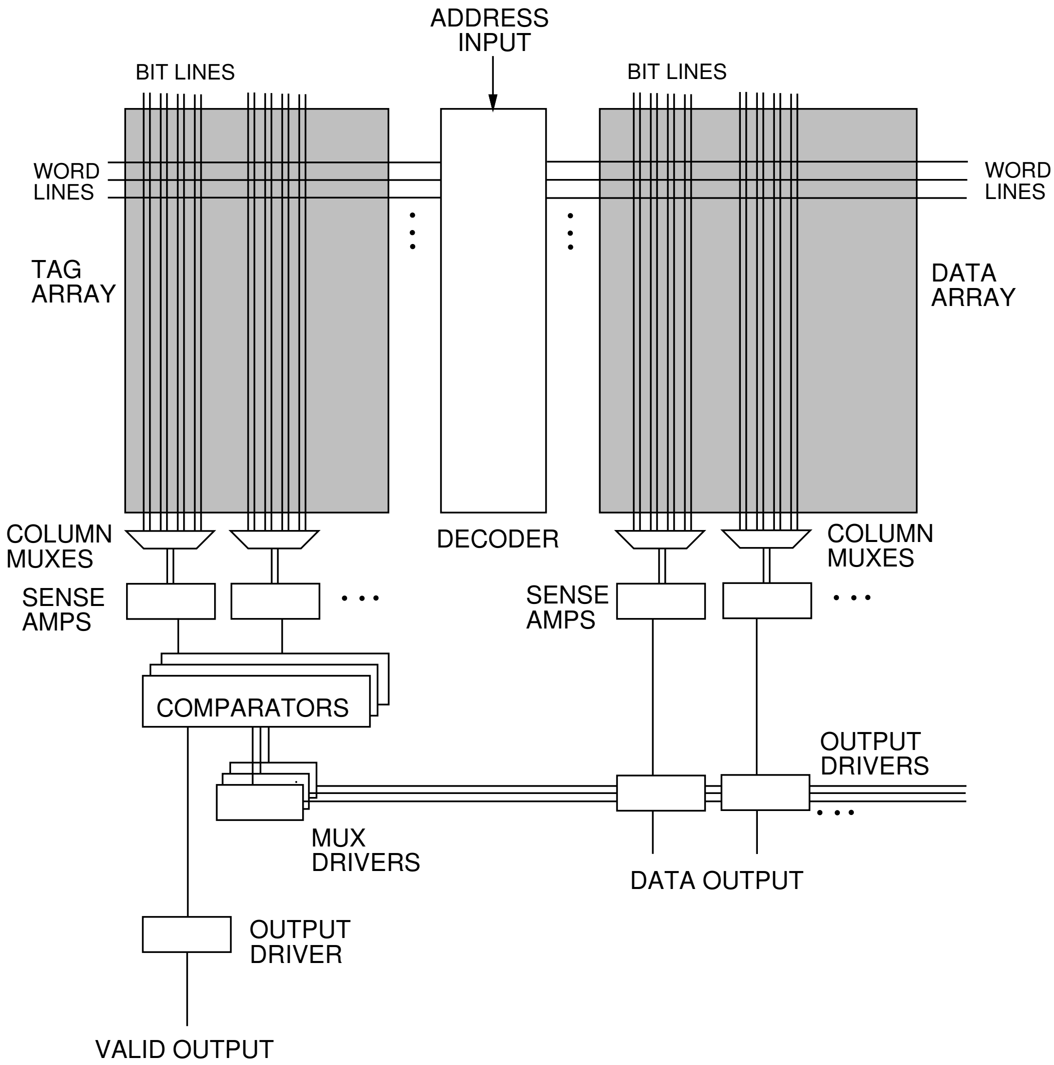

Figure 1: Cache structure

longer to read the data array, then the data side is the critical path. Since either side could determine the access time, both must be modeled in detail.

4. Cache and Array Organization Parameters

The following cache parameters are used as inputs to the model:

- C: Cache size in bytes
- B: Block size in bytes
- A: Associativity
- b_o: Output width in bits
- b_{addr}: Address width in bits

In addition, there are six array organization parameters. In the basic organization discussed by Wada [8], a single set shares a common wordline. Figure 2-a shows this organization, where B is the block size (in bytes), A is the associativity, and S is the number of sets ($S = \frac{C}{B \times A}$). Clearly, such an organization could result in an array that is much larger in one direction than the other, causing either the bitlines or wordlines to be very slow. This could result in a longer-than-necessary access time. To alleviate this problem, Wada describes how the array can be broken horizontally and vertically and defines two parameters, N_{dwl} and N_{dbl} which indicates to what extent the array has been divided. The parameter N_{dwl} indicates how many times the array has been split with vertical cut lines (creating more, but shorter, wordlines), while N_{dbl} indicates how many times the array has been split with horizontal cut lines (causing shorter bitlines). The total number of subarrays is $N_{dwl} \times N_{dbl}$.

Figure 2-b introduces another organization parameter, N_{spd}. This parameter indicates how many sets are mapped to a single wordline, and allows the overall access time of the array to be changed without breaking it into smaller subarrays.

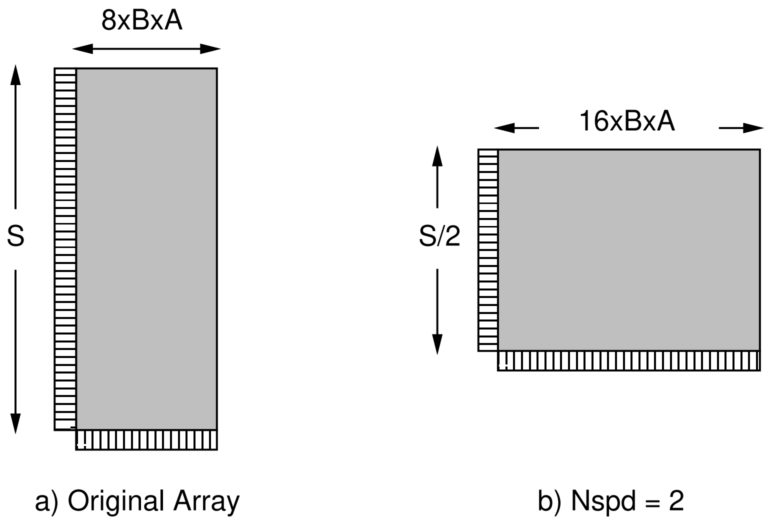

Figure 2: Cache organization parameter N_{spd}

The optimum values of N_{dwl}, N_{dbl}, and N_{spd} depend on the cache and block sizes, as well as the associativity.

Notice that increasing these parameters is not free in terms of area. Increasing N_{dbl} or N_{spd} beyond one increases the number of sense amplifiers required, while increasing N_{dwl} means more wordline drivers are required. Except in the case of a direct-mapped cache with the block length equal to the processor word length and all three parameters equal to one, a multiplexor is required to select the appropriate sense amplifier output to return to the processor. Increasing N_{dbl} or N_{spd} increases the size of the multiplexor.

Using these organizational parameters, each subarray contains $\frac{8 \times B \times A \times N_{spd}}{N_{dwl}}$ columns and $\frac{C}{B \times A \times N_{dbl} \times N_{spd}}$ rows.

We assume that the tag array can be broken up independently of the data array. Thus, there are also three tag array parameters: N_{twl}, N_{tbl}, and N_{tspd}.

5. Methodology

The analytical model in this paper was obtained by decomposing the circuit into many equivalent RC circuits, and using simple RC equations to estimate the delay of each stage. This section shows how resistances and capacitances were estimated, as well as how they were combined and the delay of a stage calculated.

5.1. Equivalent Resistances

The equivalent resistance seen between drain and source of a transistor depends on how the transistor is being used. For each type of transistor (p and n), we will need two resistances: full-on and switching.

5.1.1. Full-on Resistance

The full-on resistance is the resistance seen between drain and source of a transistor assuming the gate voltage is constant and the gate is fully conducting. This resistance can be used for pass-transistors that (as far as the critical path is concerned) are always conducting. Also, this is the resistance that is used in the Horowitz approximation discussed in Section 5.5.

It was assumed that the equivalent resistance of a conducting transistor is inversely proportional to the transistor width (only minimum-length transistors were considered). The equivalent resistance of any transistor can be estimated by:

$$res_{n,on}(W) = \frac{R_{n,on}}{W} \qquad (1)$$

$$res_{p,on}(W) = \frac{R_{p,on}}{W}$$

where $R_{n,on}$ and $R_{p,on}$ are technology dependent constants. Appendix II shows values for these two parameters in a 0.8µm CMOS process.

5.1.2. Switching Resistance

This is the effective resistance of a pull-up or pull-down transistor in a switching static gate. For the most part, our model uses an inverter approximation due to Horowitz (see Section 5.5) to model such gates, but a simpler method using the static resistance is used to estimate the wordline driver size and the precharge delay.

Again, we assume the equivalent resistance of a conducting transistor is inversely proportional to the transistor width. Thus,

$$res_{n,switching}(W) = \frac{R_{n,switching}}{W} \qquad (2)$$

$$res_{p,switching}(W) = \frac{R_{p,switching}}{W}$$

where $R_{n,switching}$ and $R_{p,switching}$ are technology dependent constants (see Appendix II). The values shown in Appendix II were measured using Hspice simulations with equal input and output transition times.

5.2. Gate Capacitances

The gate capacitance of a transistor consists of two parts: the capacitance of the gate, and the capacitance of the polysilicon line going into the gate. If L_{eff} is the effective length of the transistor, L_{poly} is the length of the poly line going into the gate, C_{gate} is the capacitance of the gate per unit area, and $C_{polywire}$ is the poly line capacitance per unit area, then a transistor of width W has a gate capacitance of:

$$gatecap = W \times L_{eff} \times C_{gate} + L_{poly} \times L_{eff} \times C_{polywire}$$

The same formula holds for both NMOS and PMOS transistors.

The value of C_{gate} depends on whether the transistor is being used as a pass transistor, or as a pull-up or pull-down transistor in a static gate. Thus, two equations for the gate capacitance are required:

$$gatecap(W, L_{poly}) = W \times L_{eff} \times C_{gate} + L_{poly} \times L_{eff} \times C_{polywire} \quad (3)$$

$$gatecap_{pass}(W, L_{poly}) = W \times L_{eff} \times C_{gate,pass} + L_{poly} \times L_{eff} \times C_{polywire}$$

Values for C_{gate}, $C_{gate,pass}$, $C_{polywire}$, and L_{eff} are shown in Appendix II. A different L_{poly} was used in the model for each transistor. This L_{poly} was chosen based on typical poly wire lengths for the structure in which it is used.

5.3. Drain Capacitances

Figures 3 and 4 show typical transistor layouts for small and large transistors respectively. We have assumed that if the transistor width is larger than 10µm, the transistor is split as shown in Figure 4.

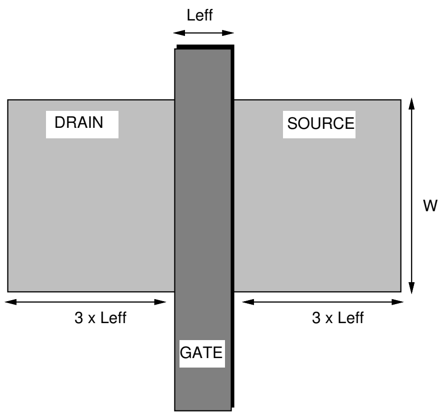

Figure 3: Transistor geometry if width < 10µm

The drain capacitance is composed of both an area and perimeter component. Using the geometries in Figures 3 and 4, the drain capacitance for a single transistor can be obtained. If the width is less than 10µm,

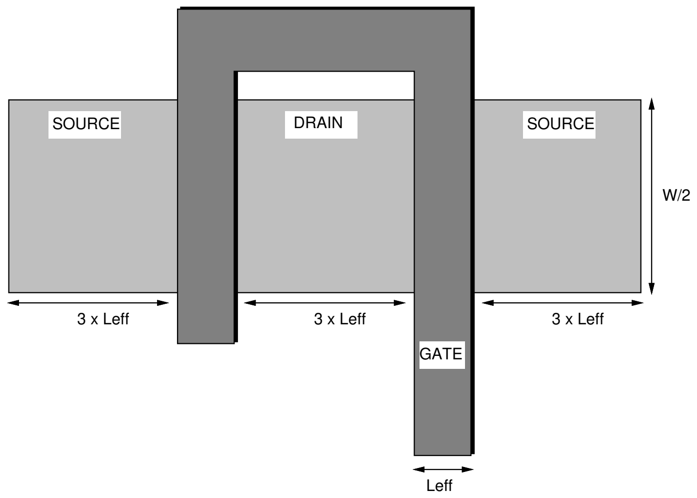

Figure 4: Transistor geometry if width >= 10µm

$$draincap(W) = 3\,L_{eff} \times W \times C_{diffarea} + (6\,L_{eff} + W) \times C_{diffside} + W \times C_{diffgate}$$

where $C_{diffarea}$, $C_{diffside}$, and $C_{diffgate}$ are process dependent parameters (there are two values for each of these: one for NMOS and one for PMOS transistors). $C_{diffgate}$ includes the junction capacitance at the gate/diffusion edge as well as the oxide capacitance due to the gate/source or gate/drain overlap. Values for n-channel and p-channel $C_{diffgate}$ are also given in Appendix II.

If the width is larger than 10µm, we assume the transistor is folded (see Figure 4), reducing the drain capacitance to:

$$draincap(W) = 3\,L_{eff} \times \frac{W}{2} \times C_{diffarea} + 6\,L_{eff} \times C_{diffside} + W \times C_{diffgate}$$

Now, consider two transistors (with widths less than 10µm) connected in series, with only a single $L_{eff} \times W$ wide region acting as both the source of the first transistor and the drain of the second. If the first transistor is on, and the second transistor is off, the capacitance seen looking into the drain of the first is:

$$draincap(W) = 4\,L_{eff} \times W \times C_{diffarea} + (8\,L_{eff} + W) \times C_{diffside} + 3\,W \times C_{diffgate}$$

Figure 5 shows the situation if the transistors are wider than 10µm. In this case, the capacitance seen looking into the drain of the inner transistor (x in the diagram) assuming it is on but the outer transistor is off is:

$$draincap(W) = 5\,L_{eff} \times \frac{W}{2} \times C_{diffarea} + 10\,L_{eff} \times C_{diffside} + 3\,W \times C_{diffgate}$$

To summarize, the drain capacitance of x stacked transistors is:

$$\begin{aligned}
&\text{if } W < 10\,\mu m \\
&draincap_n(W,x) = 3\,L_{eff} \times W \times C_{n,diffarea} + (6\,L_{eff} + W) \times C_{n,diffside} + W \times C_{n,diffgate} + \\
&\qquad (x-1) \times \{L_{eff} \times W \times C_{n,diffarea} + 2\,L_{eff} \times C_{n,diffside} + 2\,W \times C_{n,diffgate}\} \\
&draincap_p(W,x) = 3\,L_{eff} \times W \times C_{p,diffarea} + (6\,L_{eff} + W) \times C_{p,diffside} + W \times C_{p,diffgate} + \\
&\qquad (x-1) \times \{L_{eff} \times W \times C_{p,diffarea} + 2\,L_{eff} \times C_{p,diffside} + 2\,W \times C_{p,diffgate}\}
\end{aligned} \tag{4}$$

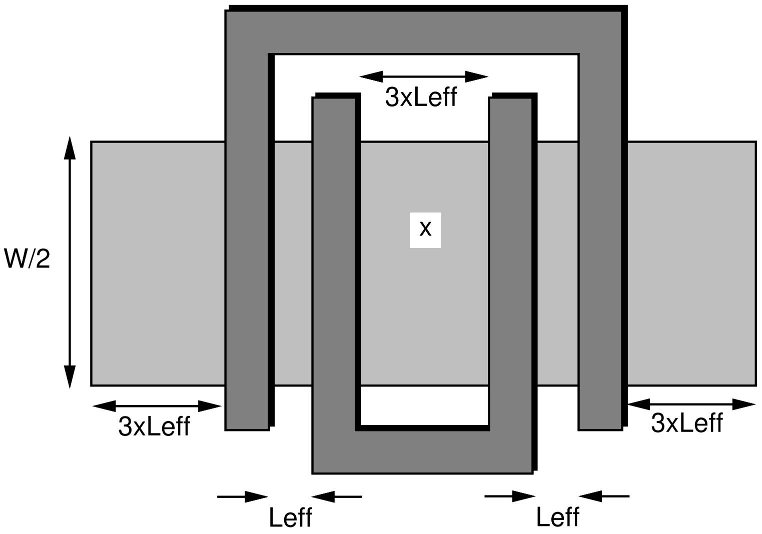

Figure 5: Two stacked transistors if each width >= 10µm

if $W >= 10\,\mu m$

$$draincap_n(W,x) = 3\,L_{eff} \times W/2 \times C_{n,diffarea} + 6\,L_{eff} \times C_{n,diffside} + W \times C_{n,diffgate} + (x-1) \times \{L_{eff} \times W \times C_{n,diffarea} + 4\,L_{eff} \times C_{n,diffside} + 2\,W \times C_{n,diffgate}\}$$

$$draincap_p(W,x) = 3\,L_{eff} \times W/2 \times C_{p,diffarea} + 6\,L_{eff} \times C_{p,diffside} + W \times C_{p,diffgate} + (x-1) \times \{L_{eff} \times W \times C_{p,diffarea} + 4\,L_{eff} \times C_{p,diffside} + 2\,W \times C_{p,diffgate}\}$$

5.4. Other Parasitic Capacitances

Other parasitic capacitances such as metal wiring are modeled using the values for $_{bitmetal}$ and $C_{wordmetal}$ given in Appendix II. These capacitance values are fixed values per unit length in terms of the RAM cell length and width. These values include an expected value for the area and sidewall capacitances to the substrate and other layers. Besides being used for parasitic capacitance estimation of the bitlines and wordlines themselves, they are also used to model the capacitance of the predecode lines, data bus, address bus, and other signals in the memory. Although the capacitance per unit length would probably less for many of these busses than for the bit lines and word lines, the same value is used for simplicity of modeling.

5.5. Horowitz Approximation

In [2], Horowitz presents the following approximation for the delay of a static inverter with a rising input:

$$delay_{rise}(t_f, t_{rise}, v_{th}) = t_f \sqrt{(\log[v_{th}])^2 + 2\,t_{rise}\,b\,(1-v_{th})/t_f}$$

where v_{th} is the switching voltage of the inverter (as a fraction of the maximum voltage), t_{rise} is the input rise time, t_f is the output time constant (assuming a step input), and b is the fraction of the swing in which the input affects the output (we used b=0.5).

For a falling input with a fall time of t_f, the above equation becomes:

$$delay_{fall}(t_f, t_{fall}, v_{th}) = t_f \sqrt{(\log[1 - v_{th}])^2 + 2\, t_{fall}\, b\, v_{th} / t_f}$$

In this case, we used b=0.4.

The delay of an inverter is defined as the time between the input reaching the switching voltage (also called threshold voltage) of the inverter and the output reaching the switching voltage of the following gate. If the inverter drives a gate with a different switching voltage, the above equations need to be modified slightly. If the switching voltage of the switching gate is v_{th1} and the switching voltage of the following gate is v_{th2}, then:

$$delay_{rise}(t_f, t_{rise}, v_{th1}, v_{th2}) = t_f \sqrt{(\log[v_{th1}])^2 + 2\, t_{rise}\, b\,(1 - v_{th1}) / t_f} + t_f\,(\log[v_{th1}] - \log[v_{th2}]) \qquad (5)$$

$$delay_{fall}(t_f, t_{fall}, v_{th1}, v_{th2}) = t_f \sqrt{(\log[1 - v_{th1}])^2 + 2\, t_{fall}\, b\, v_{th1} / t_f} + t_f\,(\log[1 - v_{th1}] - \log[1 - v_{th2}])$$

6. Delay Model

This section derives the cache read access and cycle time model. From Figure 1, the following components can be identified:

- Decoder
- Wordlines (in both the data and tag arrays)
- Bitlines (in both the data and tag arrays)
- Sense Amplifiers (in both the data and tag arrays)
- Comparators
- Multiplexor Drivers
- Output Drivers (data output and valid signal output)

The delay of each these components will be estimated separately (Sections 6.1 to 6.10), and will then be combined to estimate the access and cycle time of the entire cache (Section 6.11).

6.1. Decoder

6.1.1. Decoder Architecture

Figures 6 and 7 show the decoder architecture. It is assumed that each subarray has its own decoder; therefore, there are $Ndwl \times Ndbl$ decoders associated with the data array, and $Ntwl \times Ntbl$ tag array decoders. One driver drives all the data array decoders, while another drives the tag array decoders.

The decoder in Figure 7 contains three stages. Each block in the first stage takes three address bits (in true and complement), and generates a 1-of-8 code. This can be done with 8 NAND gates. Since there are

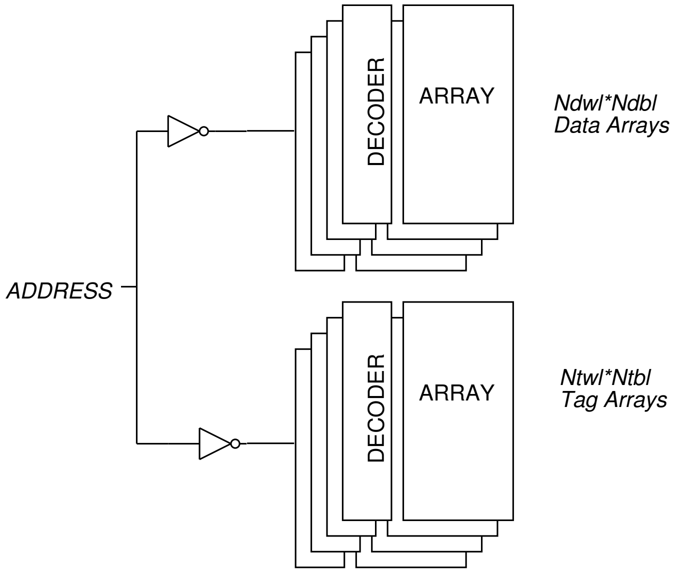

Figure 6: Decoders with decoder driver

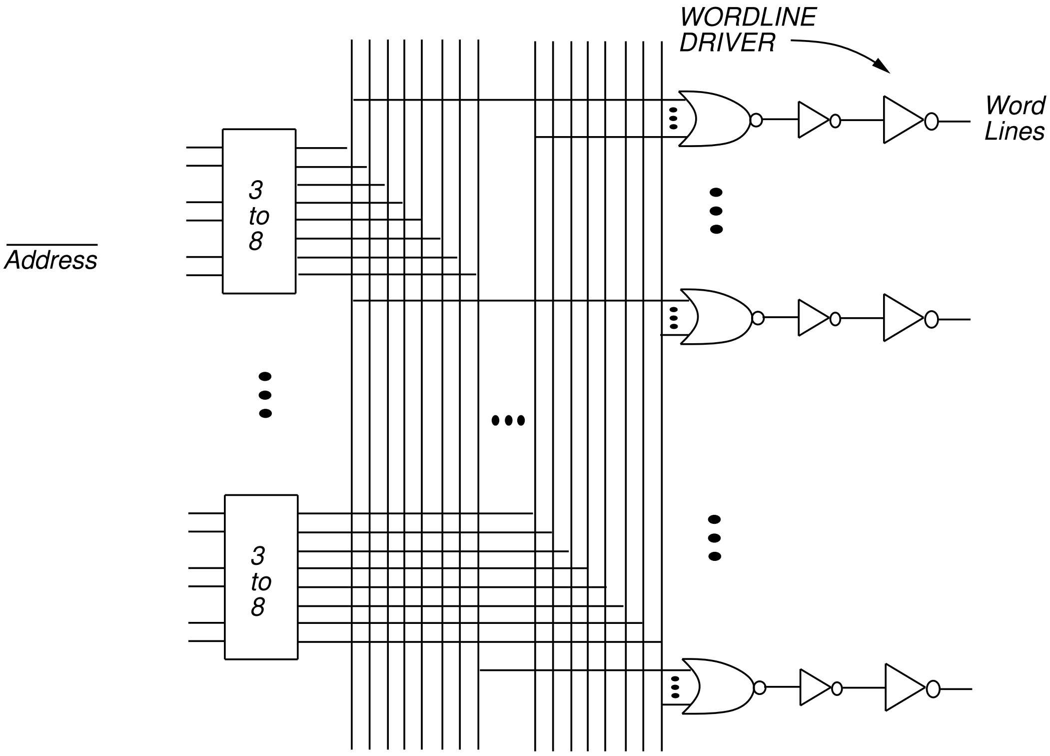

Figure 7: Single decoder structure

$$\log_2\left(\frac{C}{BAN_{dbl}N_{spd}}\right)$$

bits that must be decoded, the number of 3-to-8 blocks required is simply:

$$N_{3to8} = \left\lceil \frac{1}{3}\log_2\left(\frac{C}{BAN_{dbl}N_{spd}}\right)\right\rceil \tag{6}$$

(if the number of address bits is not divisible by three, then 1-of-4 codes can be used to make up the difference, but this was not modeled here).

These 1-of-8 codes are combined using NOR gates in the second stage. One NOR gate is required for each of the $\frac{C}{B \times A \times N_{dbl} \times N_{spd}}$ rows in the subarray. Each NOR gate must take one input from each 3-to-8 block; thus, each NOR gate has N_{3to8} inputs (where N_{3to8} was given in Equation 6).

The final stage is an inverter that drives each wordline driver.

6.1.2. Decoder Delay

Figure 8 shows a transistor-level diagram of the decoder. The decoder delay is the time after the input passes the threshold voltage of the decoder driver until *norout* reaches the threshold voltage of the final inverter (before the wordline driver). Notice that the delay does not include the time for the inverter to drive the wordline driver; this delay depends on the size of the wordline driver and will be considered in Section 6.2.

Since, in many caches, *decbus* will be precharged before a cache access, the critical path will include the time to discharge *decbus*. This occurs after *nandin* rises, which in turn, is caused by address bits (or their inverses) falling. Once *decbus* has been discharged, *norout* will rise, and after another inverter and the wordline driver, the selected wordline will rise.

Only one path is shown in the diagram; the extra inputs to the NAND gates are connected to other outputs of the decoder driver, and the extra inputs to the NOR gates are connected to the outputs of other NAND gates. The worst case for both the NAND and NOR stages occurs when all inputs to the gate change. This is the case that will be considered when estimating the decoder delay.

6.1.3. Input Fall Time

The delay of the first stage depends on the fall time of the input. To estimate a reasonable value for the input fall time, two inverters in series as shown in Figure 9 are considered. Each inverter is assumed to be the same size as the decoder driver (the first inverter in Figure 8).

The Horowitz inverter approximation of Equation 5 is used to estimate the delay of each inverter (and hence the output rise time). The time constant, t_f, of the first stage is $R_{eq} \times C_{eq}$ where R_{eq} is the equivalent resistance of the pull-up transistor in the inverter (the full-on resistance, as described in Section 5.1) and C_{eq} is the drain capacitance of the two transistors in the first inverter stage plus the gate capacitance of the two transistors in the second stage (Sections 5.3 and 5.2 show how these can be calculated). The input fall time of the first stage is assumed to be 0 (a step input), and the initial and final threshold voltages are the same. Thus, the delay of the first inverter can be written using the notation in Section 5 as:

$$T_1 = delay_{fall}(t_f, 0, v_{thdecdrive}, v_{thdecdrive})$$

where

$$\begin{aligned} t_f = {} & res_{p,on}(W_{decdrivep}) \times \\ & (draincap_n(W_{decdriven}, 1) + draincap_p(W_{decdrivep}, 1) + \\ & gatecap(W_{decdriven} + W_{decdrivep})) \end{aligned}$$

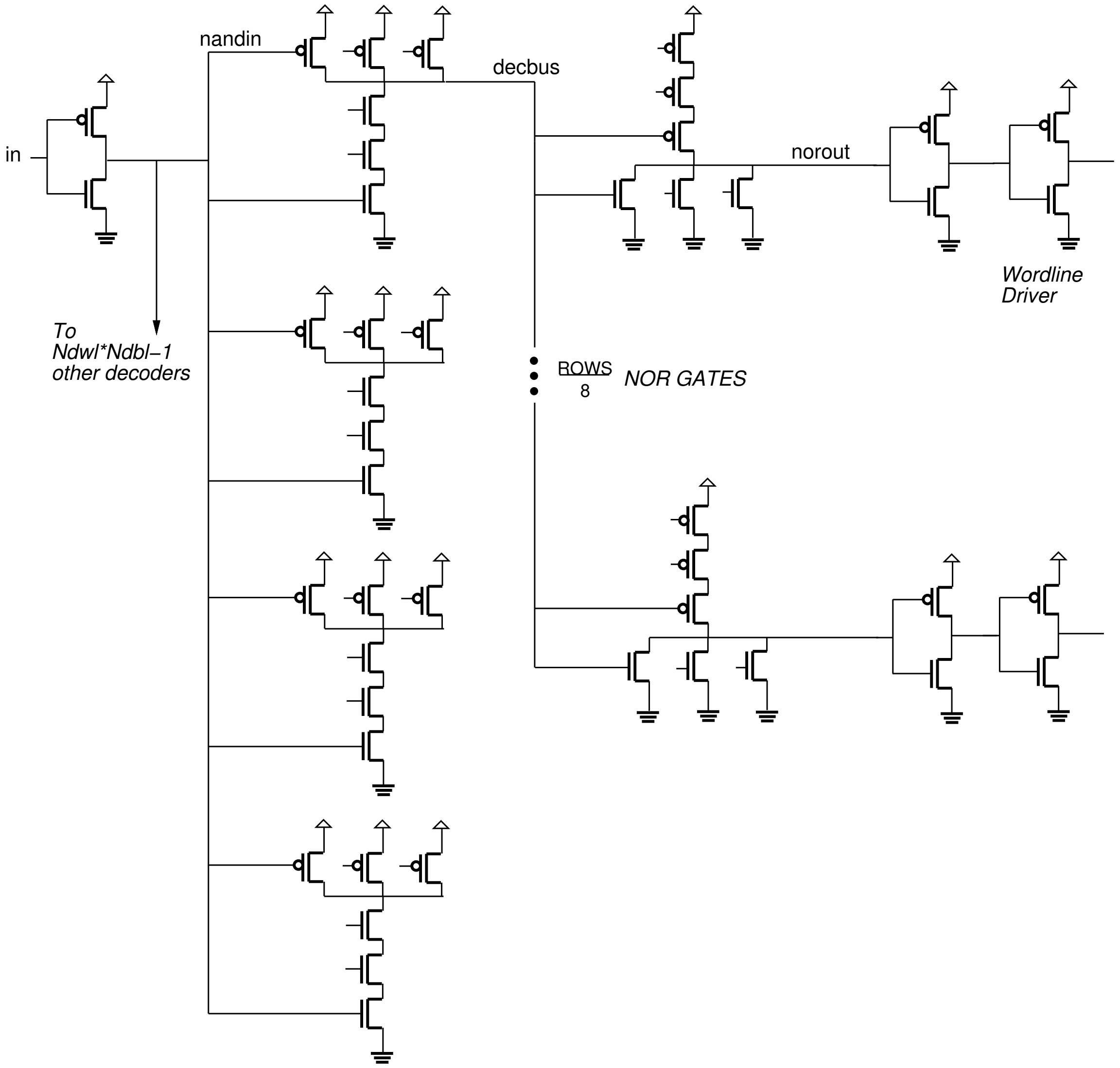

Figure 8: Decoder critical path

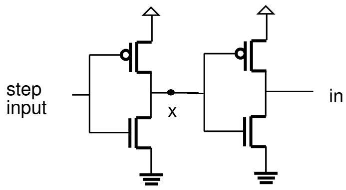

Figure 9: Circuit used to estimate reasonable input fall time

In the above equation, the widths of the transistors in the inverter transistors are denoted by $W_{decdriven}$ and $W_{decdrivep}$ and the threshold (switching) voltage of the inverter is denoted by $v_{thdecdrive}$. Appendix I shows the assumed sizes and threshold voltages for each gate on the critical path of the cache.

From the above equation, the rise time to the second stage can be approximated as $\frac{T_1}{v_{thdecdrive}}$.

The second stage can be worked out similarly:

$$T_2 = delay_{rise}\left(t_f , \frac{T_1}{v_{thdecdrive}} , v_{thdecdrive} , v_{thdecdrive} \right)$$

From this, the fall time of the second inverter output (and hence a reasonable fall time for the cache input) can be written as:

$$infalltime = \frac{T_2}{1 - v_{thdecdrive}} \tag{7}$$

Note that the above expressions for T_1 and T_2 will not be included in the cache access time; they were only derived to estimate a reasonable input fall time (Equation 7).

6.1.4. Stage 1: Decoder Driver

This section estimates the time for the first inverter in Figure 8 to drive the NAND inputs. Each inverter drives $4 \times N_{dwl} \times N_{dbl}$ NAND gates (recall that both address and address-bar are assumed to be available; thus, each driver only needs to drive half of the NAND gates in its 3-to-8 block).

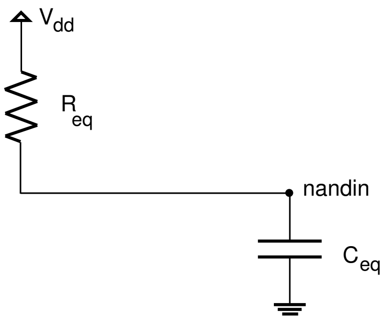

Figure 10: Decoder driver equivalent circuit

Figure 10 shows a simplified equivalent circuit. In this figure, R_{eq} is the equivalent pull-up resistance of the driver transistor plus the resistance of the metal lines used to tie the NAND outputs to the NOR inputs. The wire length can be approximated by noting that the total edge length of the memory is approximately $8 \times B \times A \times N_{dbl} \times N_{spd}$ cells. If the memory arrays are grouped in two-by-two blocks, and if the predecode NAND gates are at the center of each group, then the connection between the driver and the NAND gate is one quarter of the sum of the array widths (see Figure 11). In large memories with many groups the bits in the memory are arranged so that all bits driving the same data output bus are in the same group, reducing the required length of the data bus.

Thus, if $R_{wordmetal}$ is the approximate resistance of a metal line per bit width, then R_{eq} is:

$$R_{eq} = res_{p,on}(W_{decdrivep}) + R_{wordmetal} \times \frac{8BAN_{dbl}N_{spd}}{8}$$

Note that we have divided the $R_{wordmetal}$ term by an additional two; the first order approximation for the delay at the end of a distributed RC line is RC/2 (we assume the resistance and capacitance are distributed evenly over the length of the wire).

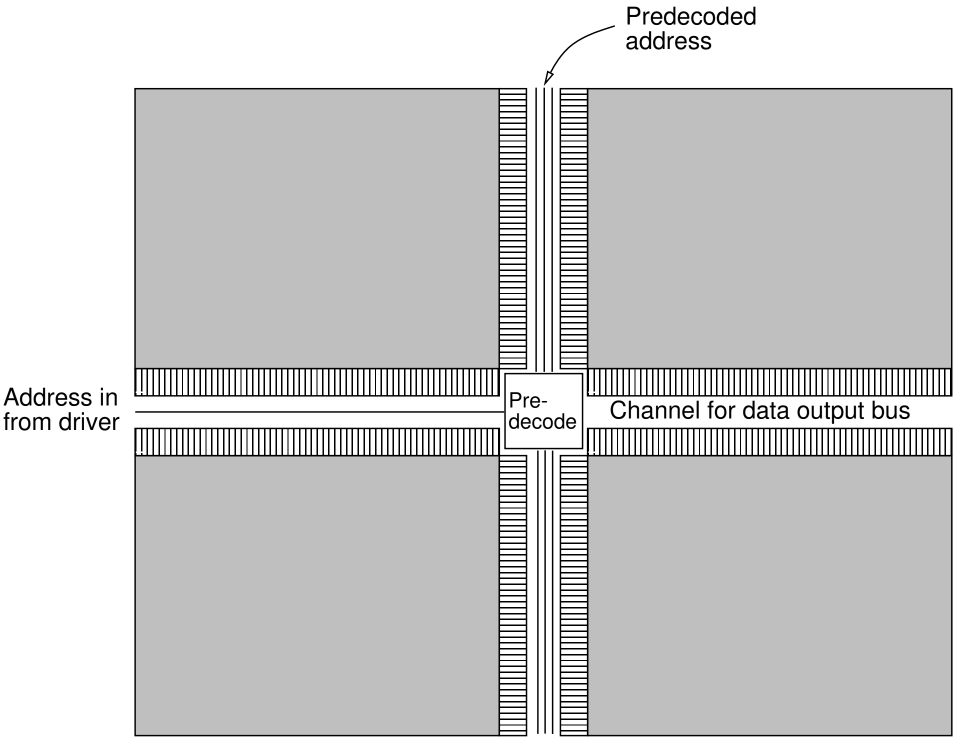

Figure 11: Memory block tiling assumptions

The equivalent capacitance C_{eq} in Figure 10 can be written as:

$$C_{eq} = draincap_p(W_{decdrivep}, 1) + draincap_n(W_{decdriven}, 1) + 4 N_{dwl} N_{dbl}\, gatecap(W_{dec3to8n} + W_{dec3to8p}, 10) + 2 B A N_{dbl} N_{spd} C_{wordmetal}$$

where $C_{wordmetal}$ is the metal capacitance of a metal wire per bit width.

Using R_{eq} and C_{eq}, the delay can be estimated as:

$$T_{dec,1} = delay_{fall}(C_{eq} R_{eq}, infalltime, v_{thdecdrive}, v_{thdec3to8}) \quad (8)$$

where *infalltime* is from Equation 7.

6.1.5. Stage 2: NAND Gates

This section estimates the time required for a NAND gate to discharge the decoder bus (and the inputs to the NOR gates). The equivalent circuit is shown in Figure 12. In this diagram, R_{eq}

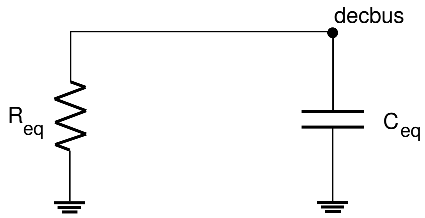

Figure 12: Decoder driver equivalent circuit

is the equivalent resistance of the three pull-down transistors (in series). The total resistance is approximated by $3\,res_{n,on}(W_{dec3to8n})$. Since all three inputs are changing simultaneously (in the worst case), each transistor has about the same resistance. In our CMOS 0.8µm process, this approximation induces an error of about 10%-20%. The resistance R_{eq} also includes the metal resistance of the lines connecting the NAND to the NOR gate. Since there are $\frac{C}{BAN_{dbl}N_{spd}}$ rows in the subarray,

$$R_{eq} = 3\,res_{n,on}(W_{dec3to8n}) + R_{bitmetal} \times \frac{C}{2BAN_{dbl}N_{spd}}$$

where $R_{bitmetal}$ is the metal resistance per bit height.

The capacitance C_{eq} is the sum of the input capacitances of $\frac{C}{8BAN_{dbl}N_{spd}}$ NOR gates, the drain capacitances of the NAND driver, and the wire capacitance. Thus,

$$C_{eq} = 3\,draincap_p(W_{dec3to8p}, 1) + draincap_n(W_{dec3to8n}, 3) + \frac{C}{8BAN_{dbl}N_{spd}} \times gatecap(W_{decnorn} + W_{decnorp}, 10) + \frac{C}{2BAN_{dbl}N_{spd}} \times C_{bitmetal}$$

The delay of this stage is given by:

$$T_{dec,2} = delay_{rise}\left(R_{eq} \times C_{eq}, \frac{T_{dec,1}}{v_{thdec3to8}}, v_{thdec3to8}, v_{thdecnor}\right) \tag{9}$$

where $T_{dec,1}$ is from Equation 8.

6.1.6. Stage 3: NOR Gates

The final part of the decoder delay is the time for a NOR gate to drive *norout* high. An equivalent circuit similar to that of Figure 10 can be used. In this case, the pull-up resistance of the NOR gate is approximated by $N_{3to8} \times res_p(W_{decnorp})$ where N_{3to8} is the number of inputs to each NOR gate (from Equation 6). The capacitance C_{eq} is

$$C_{eq} = N_{3to8}\,draincap_n(W_{decnorn}, 1) + draincap_p(W_{decnorp}, N_{3to8}) + gatecap(W_{decinvn} + W_{decinvp})$$

Then,

$$T_{dec,3} = delay_{fall}\left(R_{eq} \times C_{eq}, \frac{T_{dec,2}}{1 - v_{thdecnor}}, v_{thdecnor}, v_{thdecinv}\right) \tag{10}$$

where $T_{dec,2}$ is from Equation 9. Note that the value of $v_{thdecnor}$ depends on the number of inputs to each NOR gate (Appendix I contains several values for $v_{thdecnor}$).

6.1.7. Total decoder delay

By adding equations 8 to 10, the total decoder delay can be obtained:

$$T_{decoder,data} = T_{dec,1} + T_{dec,2} + T_{dec,3} \tag{11}$$

6.1.8. Analytical vs. Hspice Results

Figure 13 shows the decoder delay predicted by Equation 11 (solid lines) as well as the delay predicted by Hspice (dotted lines). The transistor sizes used in the Hspice model are shown in Appendix I and the technology parameters used are shown in Appendix II. The Hspice deck used in this (and all other graphs in this paper) models an entire cache; this ensures that the input slope and output loading effects of each stage are properly modeled.

The horizontal axis of Figure 13 is the number of rows in each subarray (which is $\frac{C}{B\,A\,N_{dbl}\,N_{spd}}$). The results are shown for one and eight subarrays. The analytical and Hspice results are in good agreement. The step in both results is due to a change from 3-input to 4-input NOR gates in the final decode when moving from 9 address bits to 10 address bits.

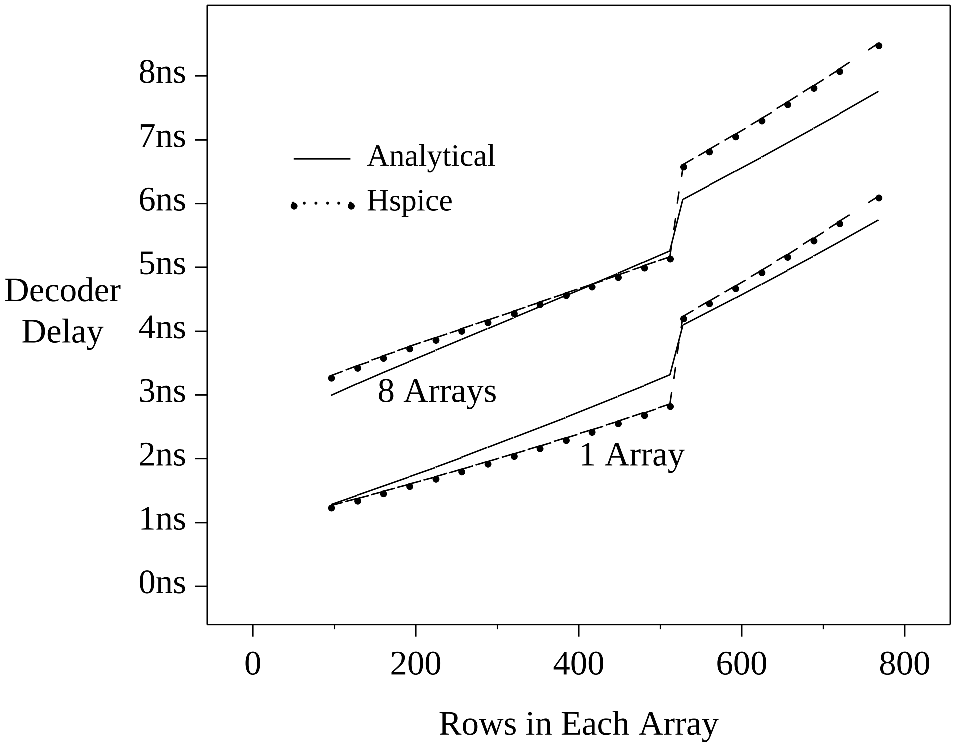

Figure 13: Decoder delay

6.1.9. Tag array decoder

The equations derived above can also be used for the tag array decoder. The only difference is that N_{dwl}, N_{dbl}, and N_{spd} should be replaced by their tag array counterparts.

6.2. Wordlines

6.2.1. Wordline Architecture

Figure 14 shows the wordline driver driving a wordline. The two inverters are the same as the final two inverters in Figure 8 (recall that the decoder equations do not include the time to discharge the decoder output).

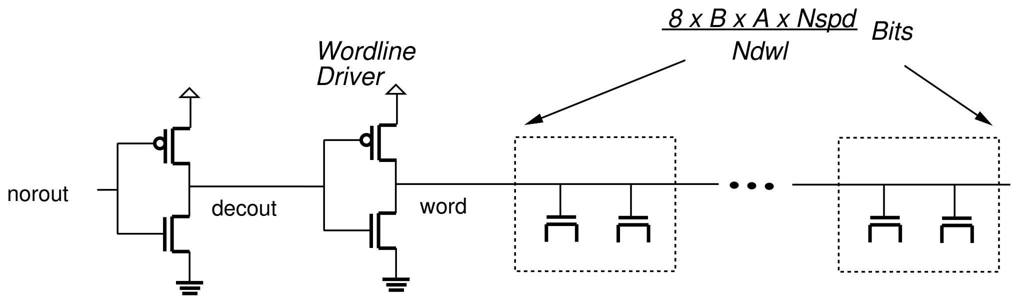

Figure 14: Word line architecture

In Wada's access time model, it was assumed that wordline drivers are always the same size, no matter how many columns are in the array. In this model, however, the wordline driver is assumed to get larger as the wordline gets longer. Normally, a cache designer would choose a target rise time, and adjust the driver size appropriately. Rather than assuming a constant rise time for caches of all sizes, however, we assume the desired rise time (to a 50% word line swing) is:

$$\text{rise time} = k_{rise} \times \ln(\text{cols}) \times 0.5$$

where

$$\text{cols} = \frac{8BAN_{spd}}{N_{dwl}}$$

The constant k_{rise} is a constant that depends on the implementation technology. To obtain the transistor size that would give this rise time, it is necessary to work backwards, using an equivalent RC circuit to find the required driver resistance, and then finding the transistor width that would give this resistance.

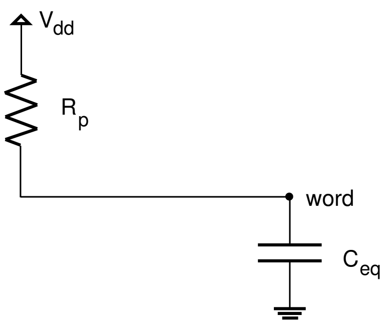

Figure 15: Equivalent circuit to find width of wordline driver

Figure 15 shows the equivalent circuit for the wordline. The pull-up resistance can be obtained using the following RC equation

$$R_p = \frac{-\text{ rise time}}{C_{eq} \times \ln(v_{thwordline})} \tag{12}$$

where $V_{thwordline}$ is inverter threshold (relative to V_{dd}). This is significantly higher than the voltage (V_t) at which the pass transistors in the memory cells begin to conduct. The use of the inverter threshold gives a more intuitive delay for the wordline but it can result in negative bitline delays.

The line capacitance is approximately the sum of gate capacitances of each memory cell in the row (a more detailed equation will be given later):

$$C_{eq} = \text{cols} \times (2 \times gatecap_{pass}(W_a, 0) + C_{wordmetal})$$

This equation was derived by noting the wordline drives the gates of two pass transistors in each bit (the memory cell is shown in Figure 19).

Once R_p is found using Equation 12, the pull-up transistor's width can be found using:

$$W_{datawordp} = \frac{R_{p,switching}}{R_p}$$

where $R_{p,switching}$ is a constant that was discussed in Section 5.1.2. When calculating capacitances, we will assume that the width of the NMOS transistor in the driver is half of $W_{datawordp}$.

6.2.2. Wordline Delay

There are two components to the word-line delay: the time to discharge the input of the wordline driver, and the time for the wordline driver to charge the wordline.

Consider the first component. The capacitance that must be discharged is:

$$C_{eq} = draincap_n(W_{decinvn}, 1) + draincap_p(W_{decinvp}, 1) + gatecap(W_{datawordp} + 0.5\,W_{datawordp}, 20)$$

The equivalent resistance of the pull-down transistor is simply

$$R_{eq} = res_{n,on}(W_{decinvn})$$

The delay is then

$$T_{word,1} = delay_{rise}\left(R_{eq} \times C_{eq}, \frac{T_{dec,3}}{v_{thdecinv}}, v_{thdecinv}, v_{thworddrive}\right) \tag{13}$$

where $T_{dec,3}$ is the delay of the final decoder stage (from Equation 10). Note that in general, $v_{thworddrive}$ will depend on the size of the wordline driver. If a constant ratio between the widths of the NMOS and PMOS driver transistors is used, however, the threshold voltage is almost constant.

From the previous section, the delay of the second stage is approximately

$$T_{word,2,approx} = k_{rise} \times \ln(\text{cols}) \times v_{thwordline}$$

This equation, however, does not take into account changes in the input slope or wiring resistances and capacitances. To get a more accurate approximation, Horowitz's equation can once again be used, with:

$$R_{eq} = res_{p,on}(W_{datawordp}) + \frac{\text{cols} \times R_{wordmetal}}{2}$$

$$C_{eq} = 2\ \text{cols} \times gatecap_{pass}(W_a, \text{BitWidth} - 2W_a) + draincap_p(W_{datawordp}, 1) + draincap_n(0.5W_{datawordp}, 1) + \text{cols} \times C_{wordmetal}$$

The quantity *BitWidth* in the above equation is the width (in µm) of one memory cell.

Using C_{eq} and R_{eq}, the time to charge the wordline is:

$$T_{word,2} = delay_{fall}\left(R_{eq} \times C_{eq}, \frac{T_{word,1}}{1 - v_{thworddrive}}, v_{thworddrive}, v_{thwordline}\right) \tag{14}$$

Equations 13 and 14 can then be combined to give the total wordline delay:

$$T_{wordline,data} = T_{word,1} + T_{word,2} \tag{15}$$

6.2.3. Analytical and Hspice Comparisons

As before, the analytical model was compared to results obtained from Hspice simulations. The technology parameters and transistor sizes shown in Appendices I and II were used, and the results in Figure 16 were obtained. The wordline in the Hspice deck was split into 8 sections, each section containing one eighth of the memory cells. The sections were separated by one eighth of the wordline resistance. As the graph shows, the equation matches the Hspice measurements very closely.

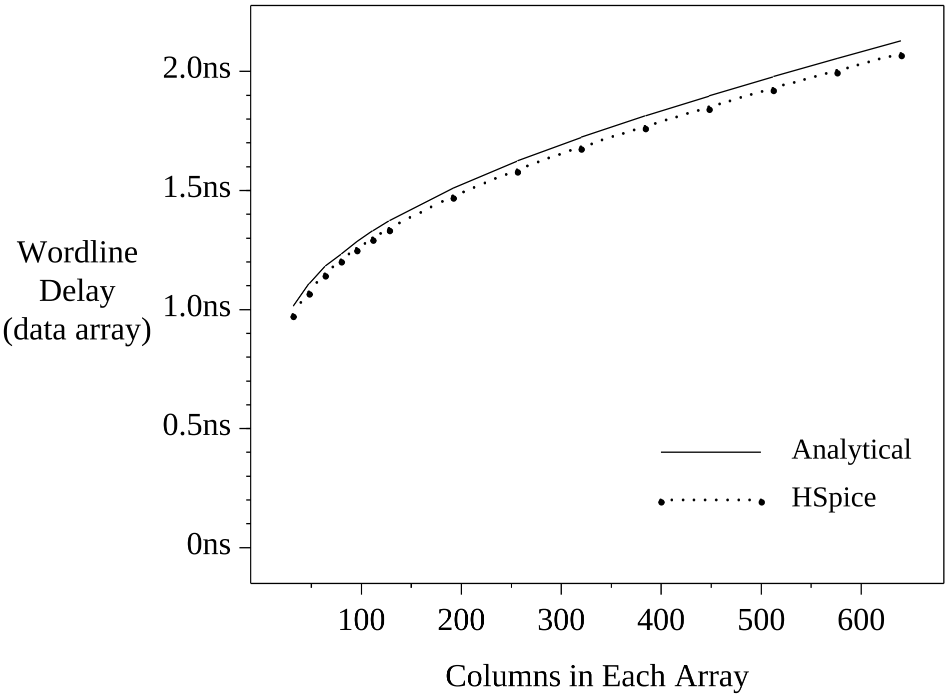

Figure 16: Wordline results

6.3. Tag Wordline

Unlike the driver for the data array wordlines, it was assumed that the size of the wordline driver in the tag array is constant for all cache sizes since the tag array is (usually) much narrower than the data array.

Figure 14 can be used to estimate the delay of the tag wordline. Again, there are two components to the delay: the time to discharge the wordline driver, and the time to charge the wordline itself. For the first component, the previous equations can be used:

$$C_{eq} = draincap_n(W_{decinvn}, 1) + draincap_p(W_{decinvp}, 1) + gatecap(W_{tagwordp} + W_{tagwordn}, 20)$$

$$R_{eq} = res_{n,on}(W_{decinvn})$$

$$T_{tagword,1} = delay_{rise}\left(R_{eq} \times C_{eq}, \frac{T_{dec,3}}{v_{thdecinv}}, v_{thdecinv}, v_{thtagworddrive}\right)$$

where $T_{dec,3}$ is the delay of the final decoder stage (from Equation 10). Note that in these equations, $W_{tagwordp}$ and $W_{tagwordn}$ are constants (unlike the equations in the previous section).

The second component is slightly different. If an address contains b_{addr} bits, then the number of bits in a tag is:

$$\text{tagbits} = b_{addr} - \log_2(\text{cache size in bytes}) + \log_2(\text{associativity}) + 2 \tag{16}$$

The "+2" is because of the valid and dirty bits. This quantity can then be used in:

$$R_{eq} = res_{p,on}(W_{tagwordp}) + \frac{\text{tagbits} \times R_{wordmetal}}{2}$$

$$C_{eq} = 2\ \text{tagbits} \times gatecap_{pass}(W_a, \text{BitWidth} - 2 \times W_a) + draincap_p(W_{tagwordp}, 1) + draincap_n(W_{tagwordn}, 1) + \text{tagbits} \times C_{wordmetal}$$

to give

$$T_{tagword,2} = delay_{fall}\left(R_{eq} \times C_{eq}, \frac{T_{tagword,1}}{1 - v_{thtagworddrive}}, v_{thtagworddrive}, v_{thwordline}\right) \tag{17}$$

The equations for $T_{tagword,1}$ and $T_{tagword,2}$ can then be summed to give the total delay attributed to the tag array wordline:

$$T_{wordline,tag} = T_{tagword,1} + T_{tagword,2} \tag{18}$$

Figure 17 shows the wordline delay times from both the analytical model (solid line) and obtained from Hspice (dotted line) using our CMOS 0.8µm process. In this case, the wordline in the Hspice deck was divided into four sections, each section separated by one quarter of the total wordline resistance. As can be seen, the two models agree very closely.

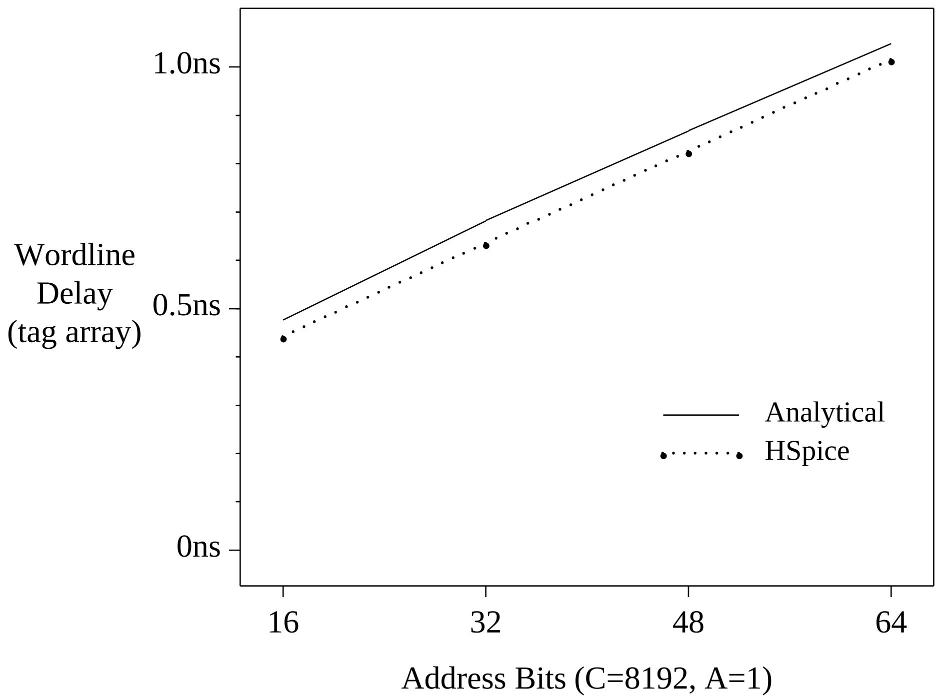

Figure 17: Wordline of tag array

6.4. Bit Lines

6.4.1. Bitline architecture

Each column in the array has two bitlines: *bit* and *bitbar*. After one of the wordlines goes high, each memory cell in the selected row begins to pull down one of its two bitlines; which bitline drops depends on the value stored in that memory cell.

In most memories, the bitlines are initially precharged high and equilibrated using a circuit like the one shown in Figure 18. During the precharge phase, both bitlines are charged to the same voltage, V_{bitpre}. The four NMOS transistors in the figure are connected as diodes; their only purpose is to drop the precharged voltage from V_{dd} to V_{bitpre} (in our process, V_{dd} is 5 volts and V_{bitpre} is 3.3 volts). The sense amplifier that will be described in the next section requires that the common mode voltage on the bitlines be less than V_{dd}.

A typical SRAM cell is shown in Figure 19. When the wordline goes high, the W_a transistors will begin to conduct, discharging one of the bitlines.

In many memories, a column select multiplexor is used to reduce the number of sense amplifiers required. Figure 20 shows such a multiplexor. The gate of the pass transistor is driven by signals from the output of the decoder. In this paper, the degree of multiplexing, that is, the number of bitlines that share a common sense amplifier, is $N_{spd} \times N_{dbl}$. Thus, $8BA$ sense amplifiers are required for the data array. Notice that the output of the column multiplexor is precharged during the precharge phase (the precharging circuit is not shown, but is the same as Figure 18).

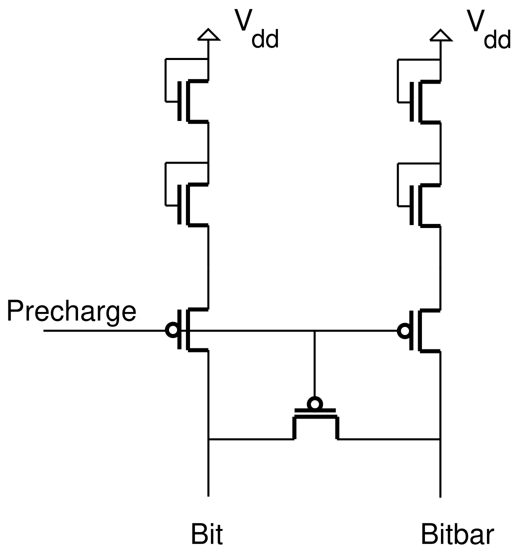

Figure 18: Precharging and equilibration transistors

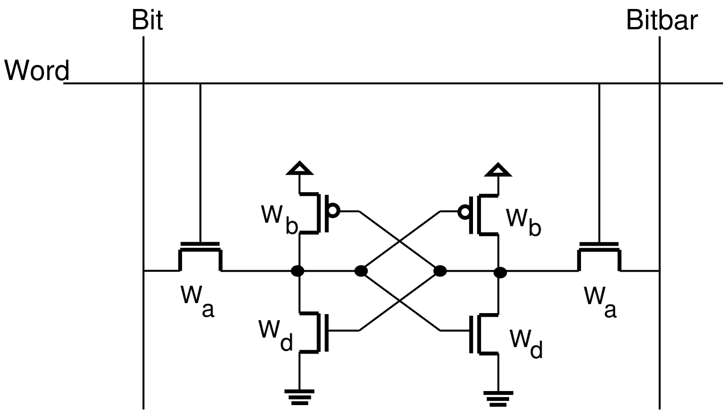

Figure 19: One memory cell

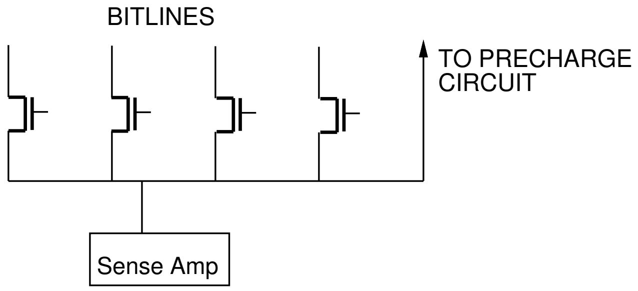

Figure 20: Column select multiplexor

6.4.2. Bitline delay

The delay of the bitline is defined as the time between the wordline going high (reaching $V_{thwordline}$) and one of the bitlines going low (reaching a voltage of $V_{bitsense}$ below its maximum value).

6.4.3. Equivalent Circuit

As previously mentioned, in each row, either *bit* or *bitbar* will go low. Consider the case when *bit* goes low. The equivalent circuit in Figure 21 can be used. The transistors labeled W_a

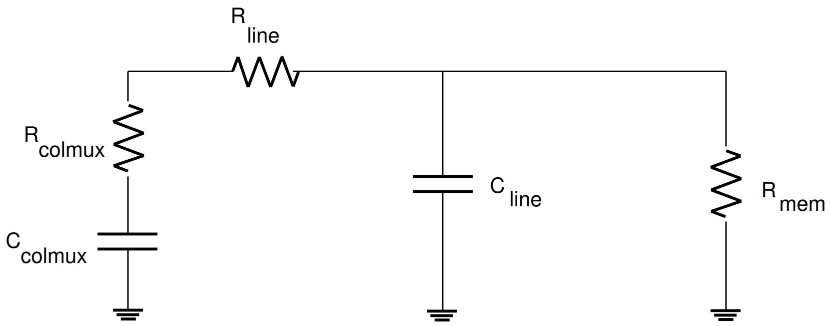

Figure 21: Bitline equivalent circuit

and W_d have been replaced by a resistor with resistance R_{mem}. The capacitance C_{line} includes the capacitance of the memory cells sharing the bitline, the metal line capacitance, the drain capacitance of the precharge circuit, and the drain capacitance of the column multiplexor pass transistor:

$$C_{line} = (\text{rows}) \times (\frac{1}{2}\, draincap_n(W_a, 1) + C_{bitmetal}) + 2\, draincap_p(W_{bitpreequ}, 1) + draincap_n(W_{bitmuxn}, 1) \tag{19}$$

where

$$\text{rows} = \frac{C}{BAN_{dbl}N_{spd}} \tag{20}$$

The drain capacitance of each W_a transistor is divided by two since each contact is shared between two vertically adjacent cells.

The capacitance C_{colmux} in Figure 21 is the capacitance seen by the output of the conducting column multiplexor pass transistor. It includes the drain capacitance of all pass transistors connected to this sense amplifier and the input capacitance of the sense amplifier:

$$C_{colmux} = (N_{spd}N_{dwl}) \times draincap_n(W_{bitmuxn}, 1) + 2\, gatecap(W_{senseQ1to4}, 10) \tag{21}$$

The resistance R_{colmux} is simply:

$$R_{colmux} = res_{n,on}(W_{bitmuxn}) \tag{22}$$

If $N_{dbl} \times N_{spd} = 1$, then there is no column multiplexors, and equations 19 to 22 can be replaced by:

$$C_{line} = (\text{rows}) \times [\frac{1}{2}\, draincap_n(W_a, 1) + C_{bitmetal}] + 2\, draincap_p(W_{bitpreequ}, 1) \qquad (23)$$

$$C_{colmux} = 2\; gatecap(W_{senseQ1to4}, 10)$$

$$R_{colmux} = 0$$

The resistances R_{mem} and R_{line} do not depend on the value of $N_{dbl}{\times}N_{spd}$. R_{mem} is the equivalent resistance of the conducting transistors in the memory cell:

$$R_{mem} = res_{n,on}(W_a, 1) + res_{n,on}(W_d, 1) \qquad (24)$$

Finally, R_{line} is the metal resistance of the bitline. As before, we assume that the resistance is clumped rather than distributed over the entire line. Thus,

$$R_{line} = \frac{\text{rows}}{2} R_{bitmetal} \qquad (25)$$

where the number of rows is as in Equation 20.

6.4.4. Equivalent circuit solution

Figure 21 can be viewed as an RC tree as described in [7]. Using the simple single time constant approximation, the delay can be written as:

$$T_{step} = [R_{mem} C_{line} + (R_{mem} + R_{line} + R_{colmux}) C_{colmux}] \ln(\frac{V_{bitpre}}{V_{bitpre} - V_{bitsense}}) \qquad (26)$$

6.4.5. Non-zero wordline rise times

Equation 26 assumes that there is a step input on the wordline. This subsection describes how the non-zero wordline rise time can be taken into account.

Figure 22 shows the wordline voltage (the input to the bitline circuit) as a function of time assuming a step input on the wordline. The time difference T_{step} shown on the graph is the time after the wordline rises until the bitline reaches $V_{bitpre} - V_{bitsense}$ (the bitline voltage is not shown on the graph). T_{step} is given by Equation 26. During this time, we can consider the bitline being "driven" towards $V_{bitpre} - V_{bitsense}$. Because the current sourced by the access transistor i can be approximated as

$$i \approx g_m(V_{gs} - V_t)$$

the shaded area in the graph can be thought of as the amount of charge discharged before the output reaches $V_{bitpre} - V_{bitsense}$. This area can be calculated as:

$$\text{area} = T_{step} \times (V_{dd} - V_t)$$

(V_t is the voltage at which the NMOS pass transistor begins to conduct).

Since the same amount of discharging is required to drive the bitline to $V_{bitpre} - V_{bitsense}$ regardless of the shape of the input waveform, we can also calculate the bitline delay for an arbitrary wordline waveform. Consider Figure 23. If we assume the area is the same as in Figure 22, then we can calculate the value of $T_{bitline,data}$. This value is the corrected bitline delay (V_s is the switching point of the NMOS pass transistor). Using simple algebra, it is easy to show that

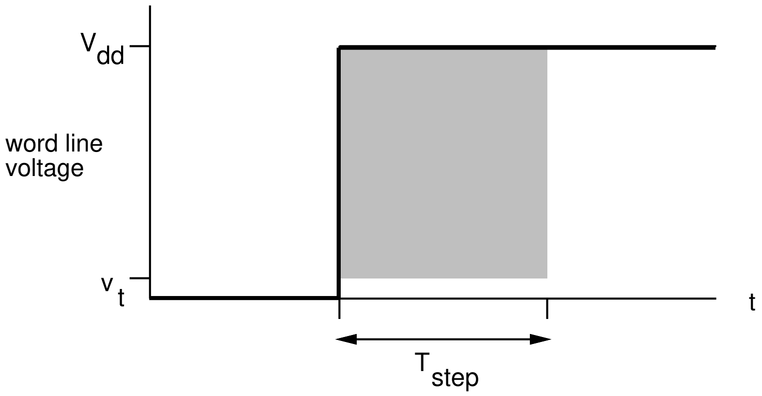

Figure 22: Step input on wordline

$$T_{bitline,data} = \frac{-2\,(v_s - v_t) + \sqrt{4\,(v_s - v_t)^2 - 4 \times m \times c}}{2 \times m} \tag{27}$$

where

$$c = \frac{1}{m}(v_s - v_t)^2 - 2\,T_{step}\,(V_{dd} - v_t)$$

and m is the slope of the input waveform.

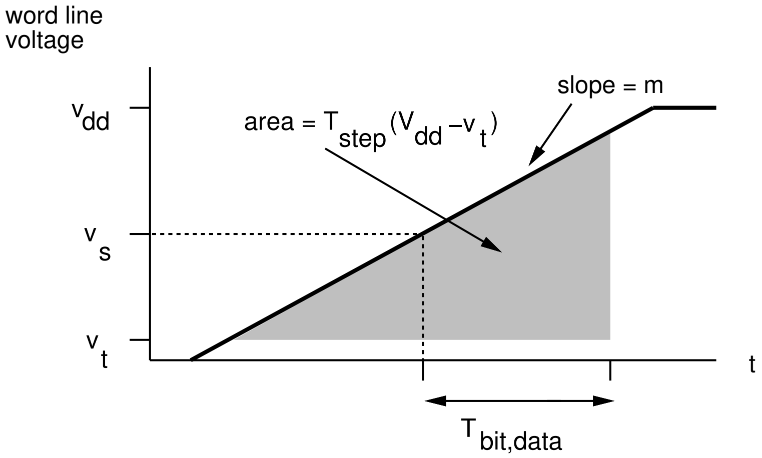

Figure 23: Slow-rising wordline

If the wordline rises quickly, as shown in Figure 24, then the algebra is slightly different. In this case,

$$T_{bitline,data} = T_{step} + \frac{V_{dd} + V_t}{2\,m} - \frac{v_s}{m} \tag{28}$$

The cross-over point between the two cases for $T_{bitline}$ occurs when:

$$T_{step} = \frac{V_{dd} - V_t}{2\,m}$$

or

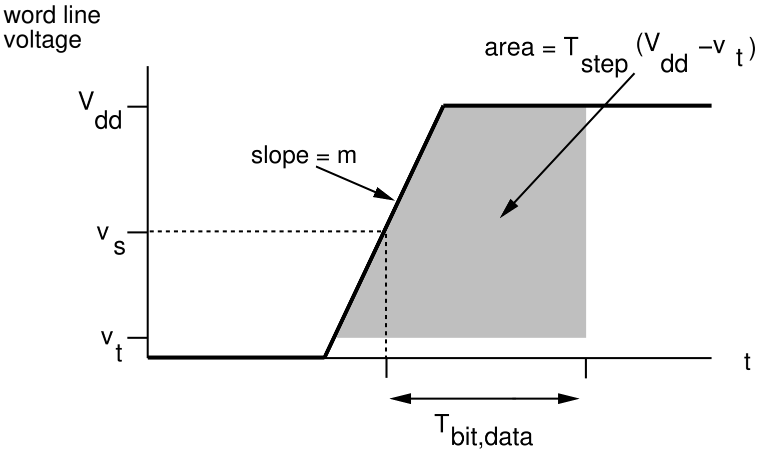

Figure 24: Fast-rising wordline

$$m = \frac{V_{dd} - V_t}{2\ T_{step}}$$

Calculating the slope, m, for a given wordline rise time is simple. From section 6.2:

$$\text{rise time} = k_{rise} \times \ln(\text{cols})$$

Thus,

$$m = \frac{V_{dd}}{k_{rise} \times \ln(\text{cols})}$$

For most practical cases, the input rise time is slow enough that Equation 27 should be used. However, it is always important to check the size of m before calculating $T_{bitline,data}$.

6.4.6. Analytical vs. Hspice Results

Again, an Hspice model was used to validate the analytical equations. Figure 25 shows the bitline delay for an array with 128 columns and no column-multiplexing. The lower two lines show the analytical and Hspice results assuming a step input on the wordline. The upper lines show the results if a non-zero wordline rise time is assumed. As the graph shows, for a wide range of array sizes (number of rows) the analytical predictions closely match the Hspice results. (The bitlines appear to have a negative delay for very small numbers of rows due to the relative thresholds used for the wordline and bitline delays.)

Figure 26 shows the same thing if 8-way column multiplexing is used; that is, a single sense amplifier is shared among 8 pairs of bitlines. The error is somewhat larger than in Figure 25, but the analytical and Hspice results are still within 0.1ns of each other.

Figure 27 shows the bitline delay as a function of the number of columns in the array. In the ideal case, in which the wordline rise time does not affect the bitline delay, the number of columns should have no effect. Indeed, the corresponding lines in Figure 27 are flat. The other two curves show how well the approximate method to take into account a non-zero rise time works.

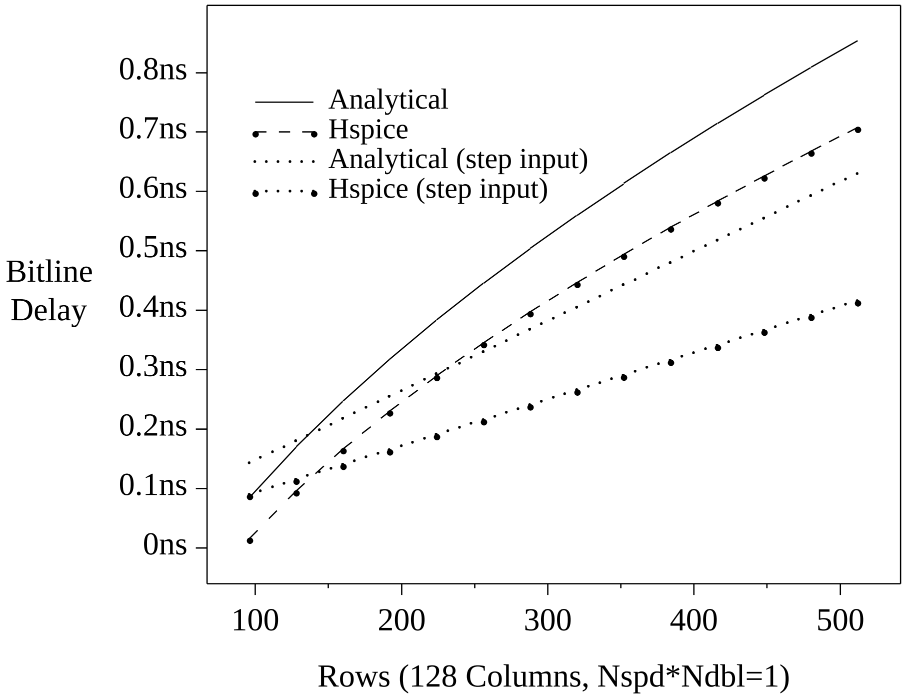

Figure 25: Bitline results without column multiplexing

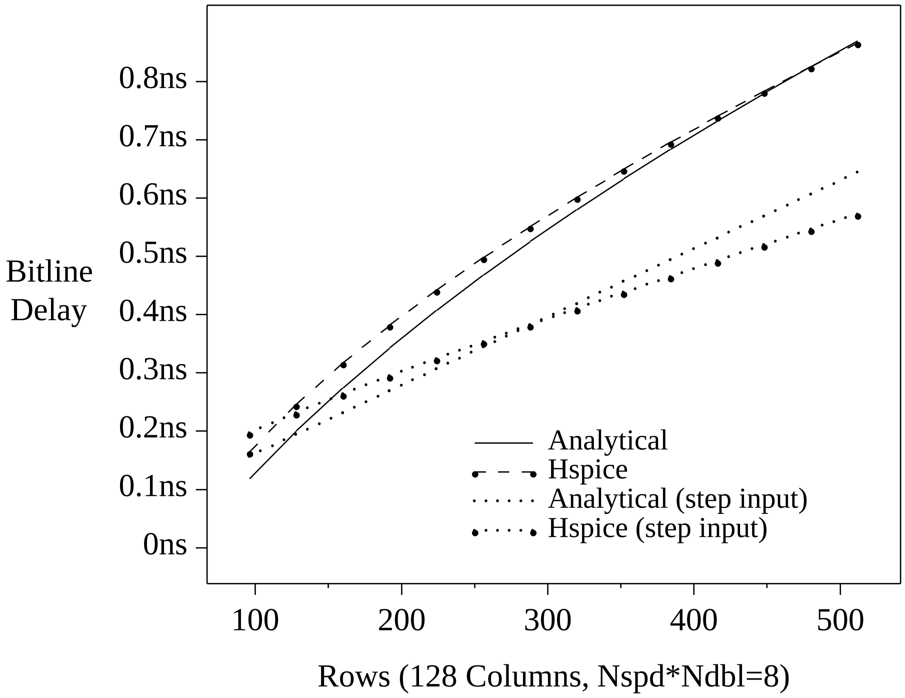

Figure 26: Bitline results with column multiplexing

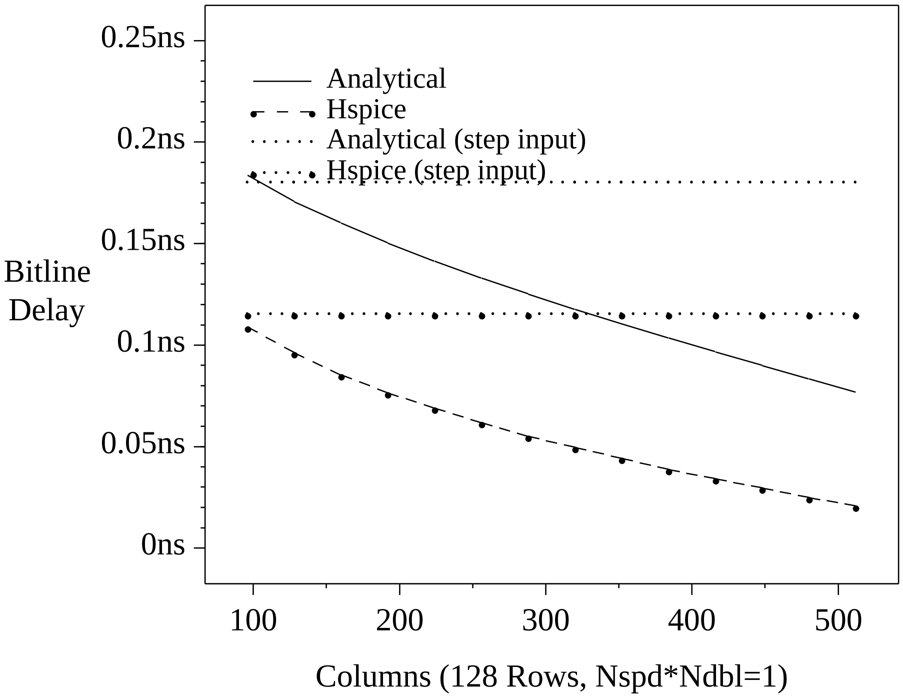

Figure 27: Bitline results vs. number of columns

Finally, Figure 28 shows how the delay is affected by the degree of column multiplexing. As expected, the larger the degree of multiplexing, the higher the delay, since more capacitance must be discharged when the bitline drops. Again, there is good agreement between the Hspice and analytical results.

6.4.7. Tag array bitlines

The equations derived in this section can be used for the tag array bitlines as well. The only difference (besides replacing the data array organizational parameters with the corresponding tag array parameters) is the calculation of the input rise time. The wordline rise time can be approximated by:

$$\text{rise time} = \frac{T_{tagword,2}}{v_{thwordline}}$$

where $T_{tagword,2}$ was given in Equation 17. This leads to:

$$m = \frac{V_{dd}\, v_{thwordline}}{T_{tagword,2}}$$

The value for $T_{bitline,tag}$ can then be obtained from either Equation 27 or 28.

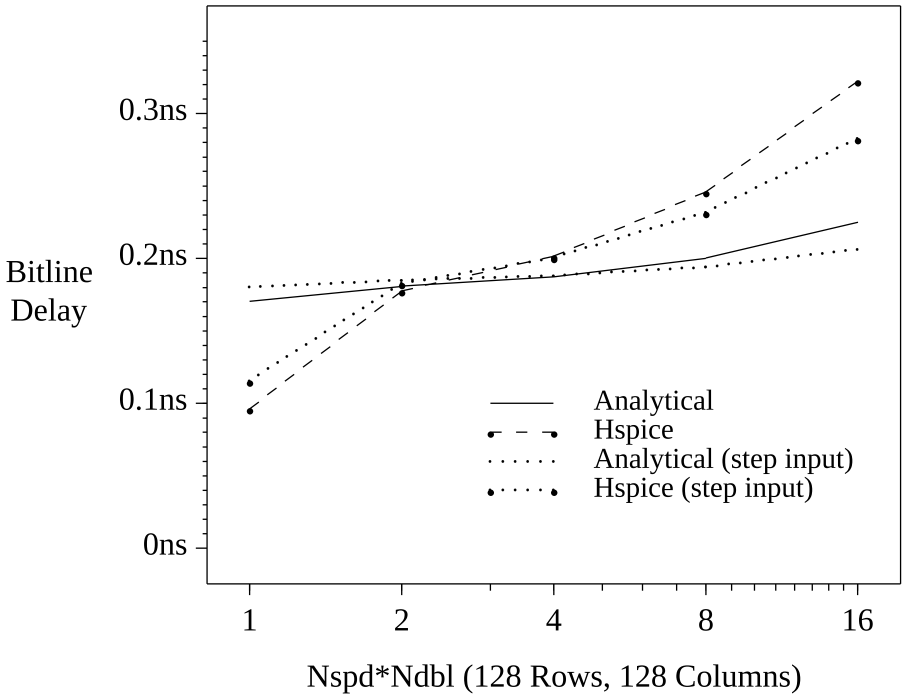

Figure 28: Bitline results vs. degree of column multiplexing

6.5. Sense Amplifier

Wada's sense amplifier (reproduced in Figure 29) amplifies a voltage difference of $2\times V_{bitsense}$ to V_{dd}. In [8], an approximation of the delay of the sense amplifier is written in terms of various process parameters. In this model, we encapsulate several of these parameters into a single process parameter, $t_{sense,data}$, which is the delay of the sense amp:

$$T_{sense,data} = t_{sense,data} \tag{29}$$

The value of $t_{sense,data}$ can be estimated from Hspice simulations. Figure 30 shows the delay measured from Hspice and the constant delay predicted by the model as a function of input fall-time (neither the analytical model used here nor Wada's model took into account the effects of a non-zero bitline fall time). As the graph shows, the error is small.

The delay of the sense amplifier in the tag array can also be approximated by a constant:

$$T_{sense,tag} = t_{sense,tag} \tag{30}$$

Comparing Figures 31 and 30, $t_{sense,tag}$ is less than $t_{sense,data}$, even though the structures of the sense amplifiers are identical. The difference is due to the output capacitance driven by each sense amp; the sense amplifier in the tag array drives a comparator, while the data array sense amplifier drives an output driver.

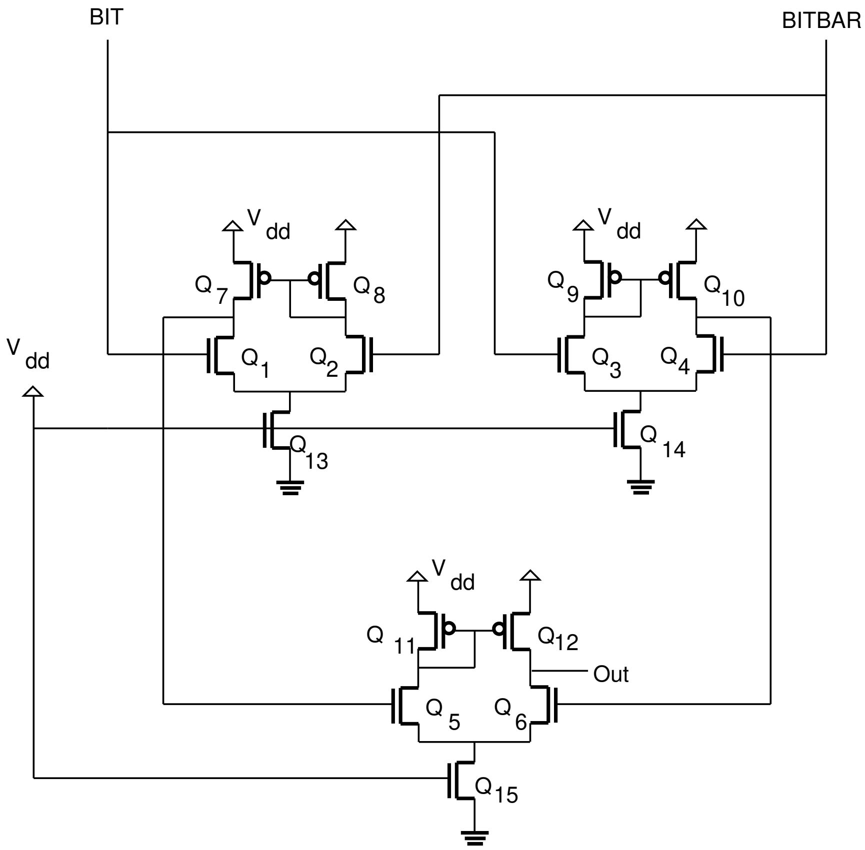

Figure 29: Sense amplifier (from [8])

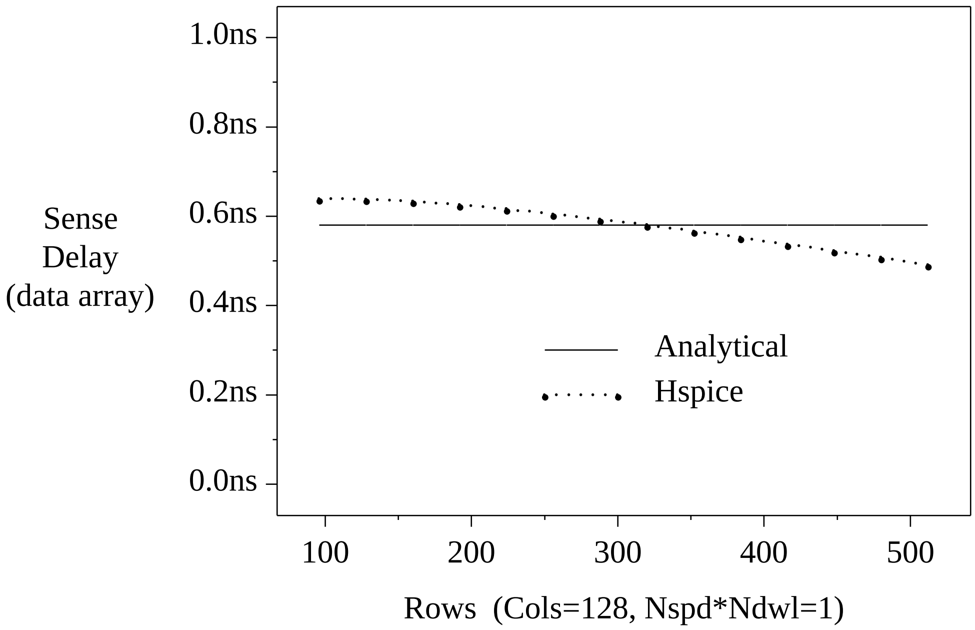

Figure 30: Data array sense amplifier delay

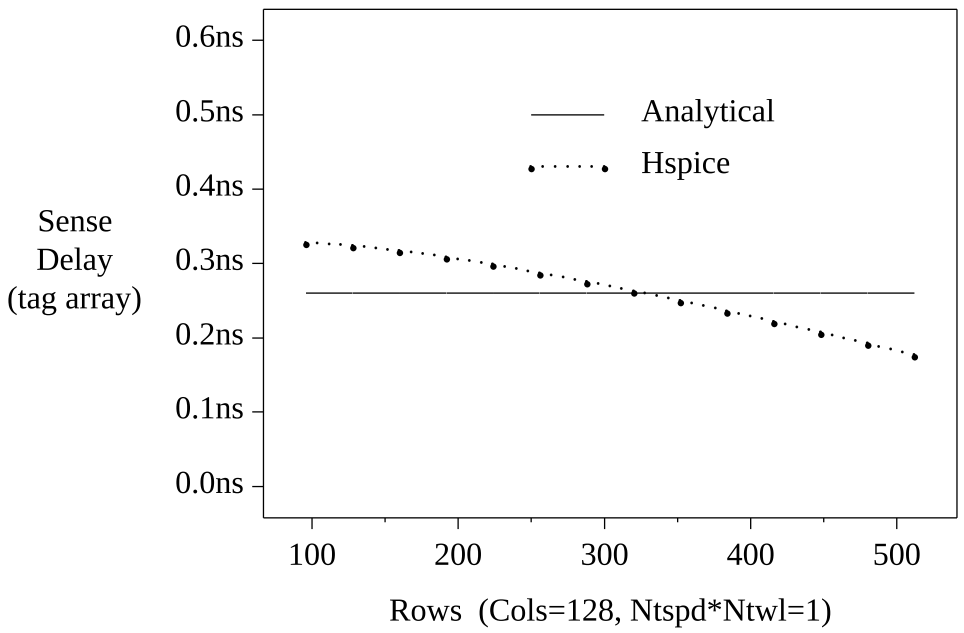

Figure 31: Tag array sense amplifier delay

The following stages will also require an approximation of the fall time of the sense amplifier output. A constant fall time for each sense amp was assumed. The fall times will be denoted by $tfall_{sense,data}$ and $tfall_{sense,tag}$; the values for our process are shown in Appendix I.

6.6. Comparator

6.6.1. Comparator Architecture

The comparator that was modeled is shown in Figure 32. The outputs from the sense amplifiers are connected to the inputs labeled b_n and b_n-bar. The a_n and a_n-bar inputs are driven by tag bits in the address. Initially, the output of the comparator is precharged high; a mismatch in any bit will close one pull-down path and discharge the output. In order to ensure that the output is not discharged before the b_n bits become stable, node *EVAL* is held high until roughly three inverter delays after the generation of the b_n-bar signals. This is accomplished by using a timing chain driven by a sense amp on a dummy row in the tag array. The output of the timing chain is used as a "virtual ground" for the pull-down paths of the comparator. When the large NMOS transistor in the final inverter in the timing chain begins to conduct, the virtual ground (and hence the comparator output if there is a mismatch) begins to discharge.

6.6.2. Comparator Delay

Since we assume that the a_n and b_n bits will be stable by the time EVAL goes low, the critical path of the comparator is the propagation delay of the timing chain plus the time to discharge the output through the NMOS transistor in the final inverter.

First consider the timing chain. The chain consists of progressively larger inverters; we assume that size of each stage is double the size of the previous stage (see Appendix I). Each stage can be worked out using a simple application of Horowitz’s approximation described in Section 5.5:

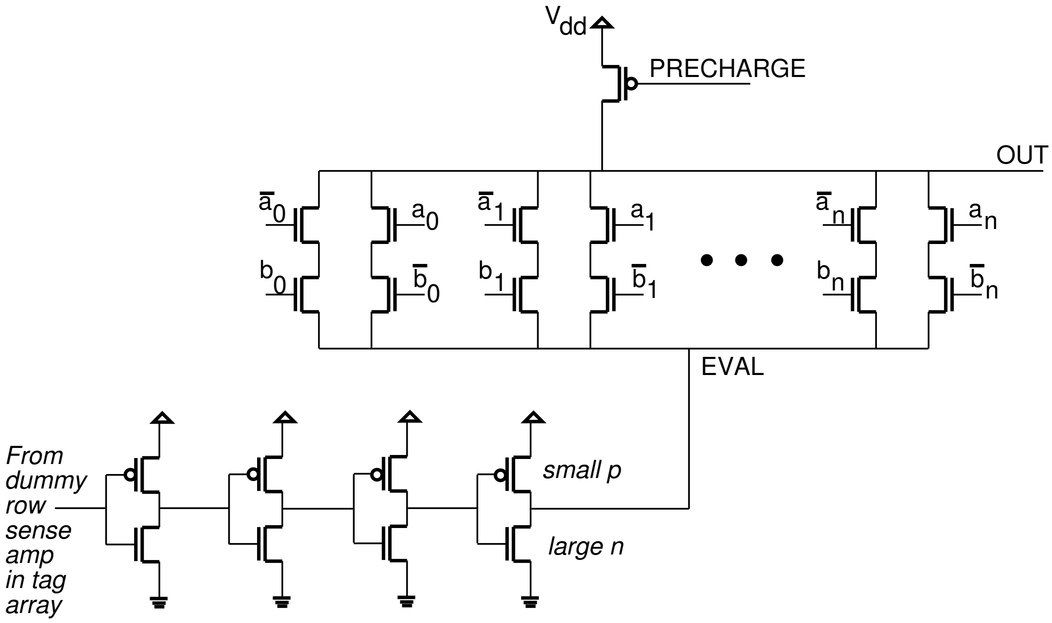

Figure 32: Comparator

$$t_{f,1} = res_{p,on}(W_{compinvp1}) \times [gatecap(W_{compinvn2} + W_{compinvp2}, 10) + draincap_p(W_{compinvp1}, 1) + draincap_n(W_{compinvn1}, 1)]$$

$$t_{f,2} = res_{n,on}(W_{compinvn2}) \times [gatecap(W_{compinvn3} + W_{compinvp3}, 10) + draincap_p(W_{compinvp2}, 1) + draincap_n(W_{compinvn2}, 1)]$$

$$t_{f,3} = res_{p,on}(W_{compinvp3}) \times [gatecap(W_{evalinvn} + W_{evalinvp}, 10) + draincap_p(W_{compinvp3}, 1) + draincap_n(W_{compinvn3}, 1)]$$

$$T_{comp,1} = delay_{fall}(t_{f,1}, tfall_{sense,tag}, v_{thcompinv1}, v_{thcompinv2})$$

$$T_{comp,2} = delay_{rise}\left(t_{f,2}, \frac{t_{f,1}}{v_{thcompinv2}}, v_{thcompinv2}, v_{thcompinv3}\right)$$

$$T_{comp,3} = delay_{fall}\left(t_{f,3}, \frac{t_{f,2}}{1 - v_{thcompinv3}}, v_{thcompinv3}, v_{thevalinv}\right)$$

The final stage involves discharging the output through a pull-down path and the NMOS transistor of the final inverter driver. An equivalent circuit is shown in Figure 33. The resistance R_{evaln} is the resistance of the pull-down transistor in the final inverter:

$$R_{evaln} = res_{n,switching}(W_{evalinvn})$$

In the worst case, only one pull-down path is conducting; the resistance $R_{pulldown}$ is the path's equivalent resistance. Since it was assumed that the inputs are stable when the evaluation takes place, we are interested in the full-on resistance of the pull-down path:

$$R_{pulldown} = 2\ res_{n,on}(W_{compn})$$

In Figure 32, it is clear that approximately half of the capacitance is on the input side of the pull-down path, and half is on the output side. These capacitances will be denoted by C_{eqbot} and C_{eqtop}. The first can be written as:

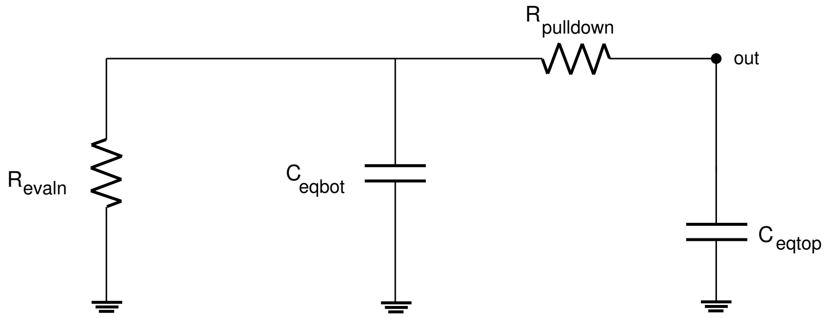

Figure 33: Comparator equivalent circuit

$$C_{eqbot} = \text{tabits} \times [\, draincap_n(W_{compn},1) + draincap_n(W_{compn},2)\,] + draincap_p(W_{evalinvp},1) + draincap_n(W_{evalinvn},1)$$

where *tagbits* is the number of tag bits (from Equation 16). Note that there are $2\times$tagbits pull-down paths (two for each bit); half of the "off" paths have the top transistor off, and half have the bottom transistor off. We have also included the drain capacitances of the final inverter stage.

The capacitance C_{eqtop} can be written similarly:

$$C_{eqtop} = \text{tabits} \times (draincap_n(W_{compn},1) + draincap_n(W_{compn},2)) + draincap_p(W_{compp},1) + gatecap(W_{muxdrv1n} + W_{muxdrv1p}, 20) + \text{tagbits} \times N_{tbl} N_{tspd} C_{wordmetal}$$

The output capacitance is taken to be the input capacitance of either the multiplexor driver described in Section 6.7 or the valid signal driver in Section 6.9 (the first stage of both structures are the same, so they have the same input capacitance). We have also included metal capacitance of the metal output (it is assumed that the metal crosses the entire width of the tag array).

The circuit in Figure 33 is equivalent to the circuit in Figure 21, so the same solution can be used. The result is:

$$T_{step} = [R_{evaln}\, C_{eqbot} + (R_{evaln} + R_{pulldown})\, C_{eqtop}]\, \ln\left(\frac{1}{v_{thmuxdrv1}}\right) \tag{31}$$

The non-zero input fall time can be taken into account using the same method as Section 6.4. There are two possible equations; which one should be used depends on the slope of the input. For a slow rising input:

$$T_{eval} = \frac{-2\,(v_s - v_t) + \sqrt{4\,(v_s - v_t)^2 - 4 \times m \times c}}{2 \times m} \tag{32}$$

where

$$c = \frac{1}{m}(v_s - v_t)^2 - 2\, T_{step}\,(V_{dd} - V_t)$$

and m is the slope of the input waveform:

$$m = \frac{T_{comp,3}}{v_{thevalinv}}$$

The above equation should be used when:

$$m < \frac{V_{dd} - V_t}{2\ T_{step}}$$

For a quickly rising input (m greater than $\frac{V_{dd} - V_t}{2\ T_{step}}$):

$$T_{eval} = T_{step} + \frac{V_{dd} + V_t}{2\ m} - \frac{v_s}{m} \quad (33)$$

The above equations can be combined to give the total delay due to the comparator:

$$T_{compare} = T_{comp1} + T_{comp2} + T_{comp3} + T_{eval} \quad (34)$$

6.6.3. Hspice Comparisons

Figure 34 compares the analytical model to a Hspice model of the circuit. As can be seen, there is good agreement between the two models.

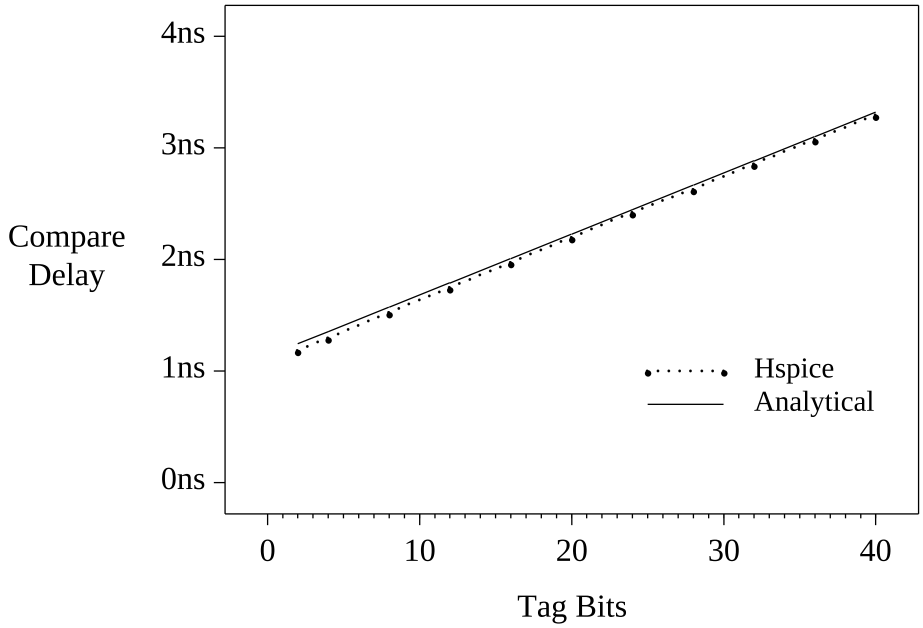

Figure 34: Comparator delay

6.7. Multiplexor Driver

In a set-associative cache, the result of the A comparisons must be used to select which of the A possible blocks are to be sent out of the cache. The structure of the output multiplexors will be described in the next section; here we concentrate on the driver that drives the select lines of these multiplexors. Figure 35 gives the overall context of the multiplexor drivers and output driver circuits in an A-way set-associative organization. Each multiplexor driver is responsible for controlling the multiplexing of the $8B$ bits from each cache line onto a data bus that reads out b_o bits. This is repeated A times in an A-way set-associative cache.

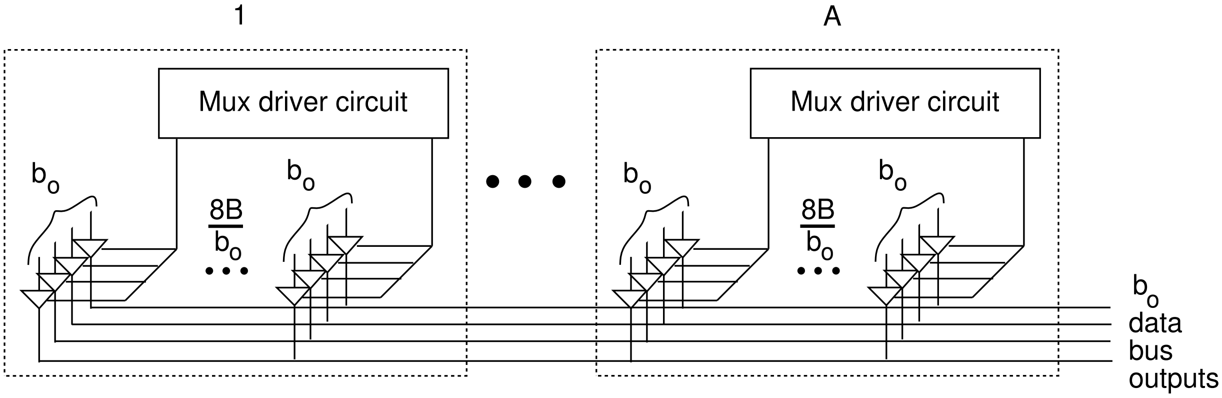

Figure 35: Overview of data bus output driver multiplexors

6.7.1. Multiplexor Driver Architecture

Figure 36 shows the structure of the three-stage multiplexor driver that we have assumed (since there are *A* comparators, *A* copies of this block are required). The output of the comparator is first inverted. This inverted match signal, *matchb*, is used to drive $\frac{8B}{b_o}$ NOR gates (recall that b_o is the number of output bits of the cache). The other inputs to the NOR gates are derived from the address bits; we assume they are stable before the comparator result is valid. The output of each NOR gate is again inverted (to produce *selb*) and the inverted signal is used to drive the select lines of b_o multiplexors.

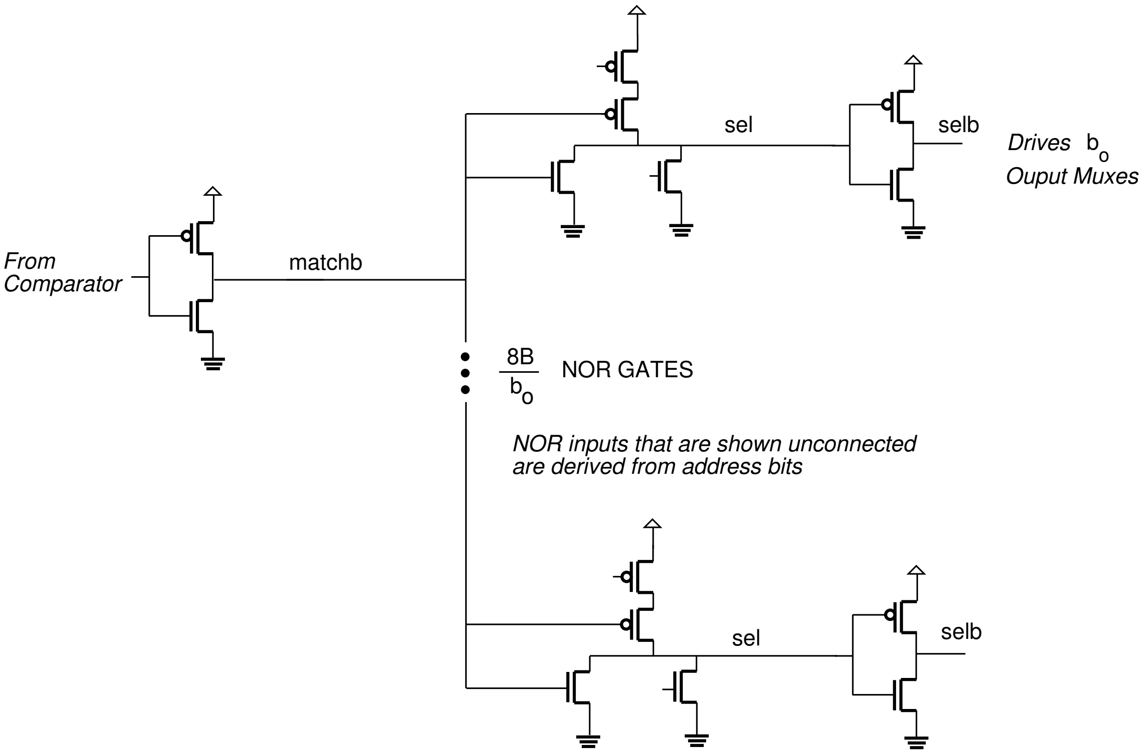

Figure 36: One of the *A* multiplexor driver circuits in an *A*-way set-associative cache

6.7.2. Multiplexor Driver Delay

The delay of the three stages can be found in a manner similar to that used for the previously discussed circuits. Using Horowitz's approximation gives:

$$t_{f,1} = res_{p,on}(W_{muxdrv1p}) \times [\frac{8B}{b_o} gatecap(W_{muxdrvnorn} + W_{muxdrvnorp}, 15) + draincap_p(W_{muxdrv1p}, 1) + draincap_n(W_{muxdrv1n}, 1)]$$

$$t_{f,2} = res_{n,on}(W_{muxdrvnorn}) \times [gatecap(W_{muxdrv3n} + W_{muxdrv3p}, 15) \; draincap_p(W_{muxdrvnorp}, 2) + 2\, draincap_n(W_{muxdrvnorn}, 1)]$$

$$t_{f,3} = [res_{p,on}(W_{muxdrv3p}) + BAN_{spd}N_{dbl}R_{wordmetal}] \times [gatecap(W_{outdrvseln} + W_{outdrvselp} + W_{outdrvnorn} + W_{outdrvnorp}, 35) + draincap_p(W_{muxdrv3p}, 1) + draincap_n(W_{muxdrv3n}, 1) + 4BAN_{spd}N_{dbl}C_{wordmetal}]$$

$$T_{muxdr,1} = delay_{fall}(t_{f,1}, \frac{T_{eval}}{1 - v_{thmuxdrv1}}, v_{thmuxdrv1}, v_{thmuxdrvnor})$$

$$T_{muxdr,2} = delay_{rise}(t_{f,2}, \frac{t_{muxdr,1}}{v_{thmuxdrvnor}}, v_{thmuxdrvnor}, v_{thmuxdrv3})$$

$$T_{muxdr,3} = delay_{fall}(t_{f,3}, \frac{t_{muxdr,2}}{1 - v_{thmuxdrv3}}, v_{thmuxdrv3}, v_{thoutdrvsel})$$

Note that in the second-stage NOR gate, we have assumed that only one pull-down path is conducting. Also, we have included the metal resistance of the *selb* line (we assume the line travels half the width of the cache).

The total multiplexor driver delay is simply:

$$T_{muxdriver} = T_{muxdr,1} + T_{muxdr,2} + T_{muxdr,3}$$

6.7.3. Hspice comparisons

Figures 37, 38 and 39 show the delay for the analytical and Hspice models as a function of b_{addr}, number of NOR gates ($\frac{8B}{b_o}$), and b_o (although the delay is not strictly a function of b_{addr}, the fall time of the comparator is, and this affects the multiplexor delay through Horowitz's inverter approximation). As the graphs show, the analytical model matches the Hspice results very well.

6.8. Output Driver

6.8.1. Architecture

The structure of the output driver is shown in Figure 40. Each sense amplifier in the data array drives the *senseout* input of an output driver; since there are $8BA$ sense amplifiers, there are that many output drivers. Typically, the number of cache output bits, b_o is less than $8BA$. Therefore, each of the output drivers is actually a tri-state driver. Each driver is turned on and off using one of the *selb* signals generated in the multiplexor driver described in Section 6.7.

There are two potential critical paths through the output driver. In a set-associative cache, the correct output can not be driven until both the *senseout* and *selb* signals are stable. Two expres-

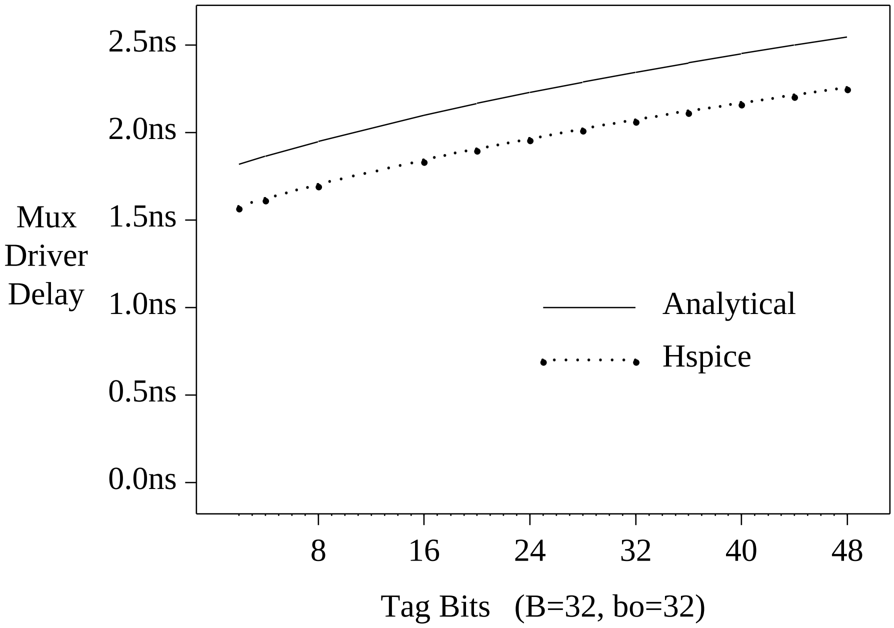

Figure 37: Multiplexor driver delay as a function of b_{addr}

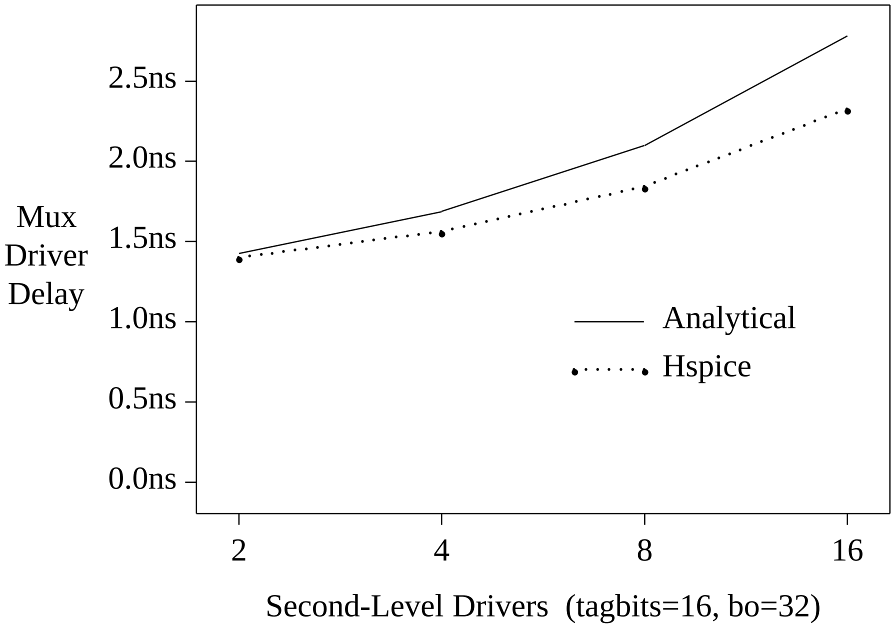

Figure 38: Multiplexor driver delay as a function of $\frac{8B}{b_o}$

sions will be derived. The first is the time after *selb* becomes stable until the inverted signal at the NAND input is valid. The second expression is the time after *sel* and *senseout* are both valid until the output of the driver is stable. Section 6.11 will show how these two quantities are used when estimating the overall cache delay.

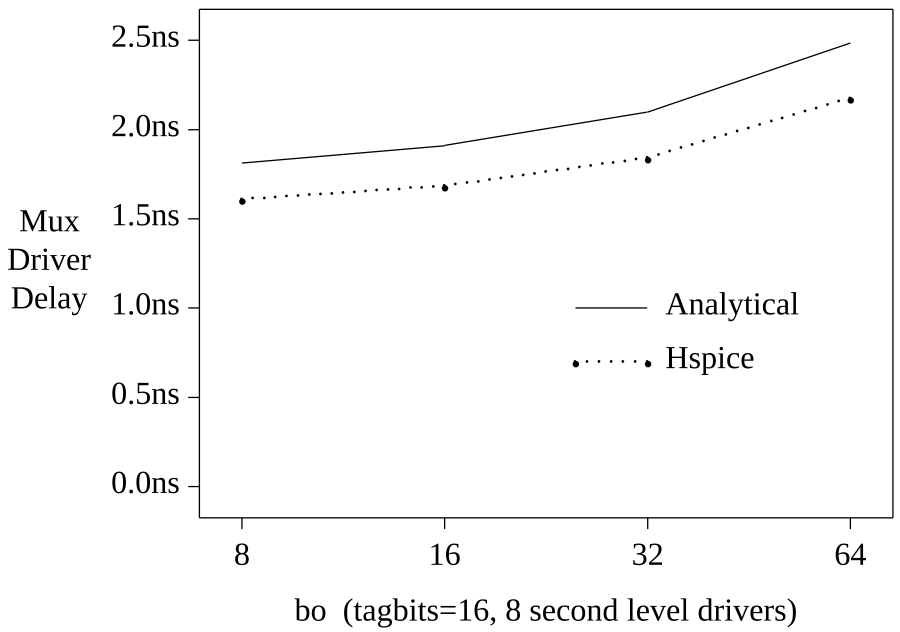

Figure 39: Multiplexor driver delay as a function of b_o

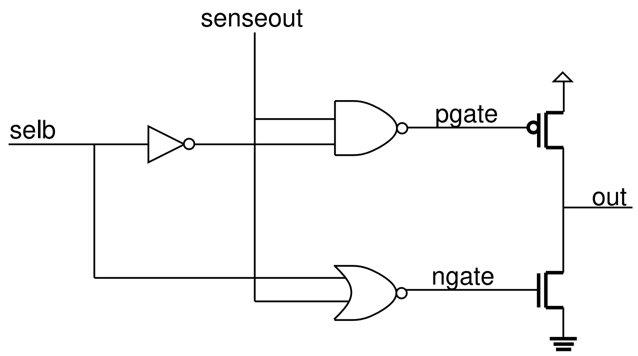

Figure 40: Output driver

6.8.2. Select invert stage

The time constant of the inverter that inverts *selb* can be estimated as:

$$t_{f,inv} = res_{n,on}(W_{outdrvseln}) \times (gatecap(W_{outdrvnandn} + W_{outdrvnandp}, 10) + draincap_p(W_{outdrvselp}, 1) + draincap_n(W_{outdrvseln}, 1)$$

which gives:

$$T_{outdrive,inv} = delay_{rise}\left(t_{f,inv}, \frac{t_{muxdr,3}}{v_{thoutdrvsel}}, v_{thoutdrvsel}, v_{thoutdrvnand}\right) \qquad (35)$$

Figure 41 compares the model predictions and Hspice measurements.

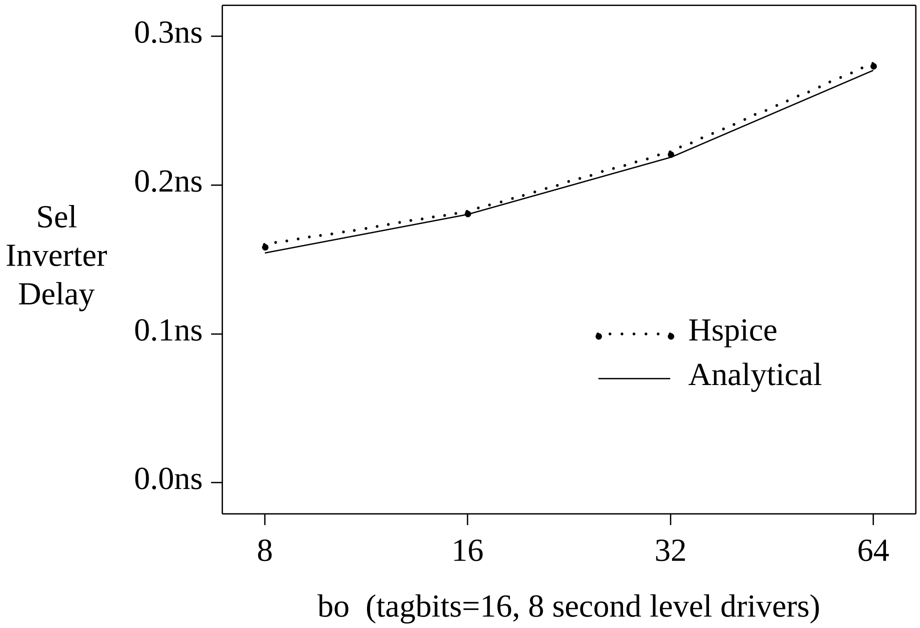

Figure 41: Output driver delay as a function of b_o: selb inverter

6.8.3. Data output path

We assume the transistor sizes in the NAND and NOR gates are such that the delay through each is the same. This model will include the delay through the NOR gate as follows:

$$t_{f,nor} = 2\,res_{p,on}(W_{outdrvnorp}) \times [gatecap(W_{outdrivern}, 10) + draincap_p(W_{outdrvnandp}, 2) + 2\,draincap_n(W_{outdrvnandn}, 1]$$

$$T_{nor} = delay_{fall}(t_{f,nor}, tfall_{sense,data}, v_{thoutdrvnor}, v_{thoutdriver})$$

The output driver stage will be treated as an inverter:

$$t_{f,final} = [res_{p,on}(W_{outdriverp}) + R_{wordmetal}\frac{8BAN_{spd}n_{vstack}}{2}] \times [\frac{8BA}{b_o}(draincap_p(W_{outdriverp}, 1) + draincap_n(W_{outdrivern}, 1)) + C_{wordmetal} \times (8BAN_{spd}n_{vstack}) + C_{out}]$$

$$T_{final} = delay_{rise}(t_{f,final}, \frac{T_{nor}}{v_{thoutdriver}}, v_{thoutdriver}, 0.5)$$

where C_{out} is the output capacitance of the cache and n_{vstack} is the number of arrays stacked vertically (the arrays are assumed to be laid out so as to make the resulting structure roughly square).

The total delay of the second part of the driver can then be written as:

$$T_{outdrive,data} = T_{nor} + T_{final} \quad (36)$$

Figure 42 shows the analytical and Hspice estimations of $T_{outdrive,data}$.

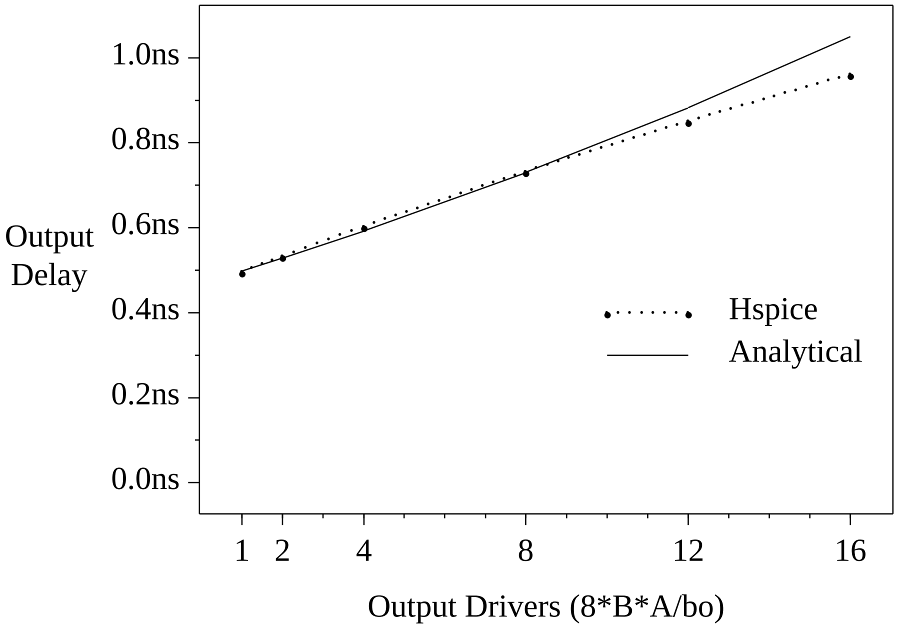

Figure 42: Output driver delay: data path

6.9. Valid Output Driver

The comparator also drives a *valid* output. In a set-associative cache, this driver is not on the critical path, but in a direct-mapped cache, it could be. Thus, it is necessary to estimate the delay of this driver.

The valid signal driver is simply an inverter with transistor widths of $W_{muxdrv1n}$ and $W_{muxdrv1p}$. The equations for the delay of the driver are:

$$t_f = res_{p,on}(W_{muxdrv1p}) \times [draincap_n(W_{muxdrv1n}, 1) + draincap_p(W_{muxdrv1p}, 1) + C_{out}]$$

$$T_{valid} = delay_{fall}\left(t_f, \frac{T_{eval}}{1 - v_{thmuxdrv1}}, v_{thmuxdrv1}, 0.5\right)$$

Figure 43 shows the analytical and Hspice delays as a function of the number of bits in a tag. The number of tag bits affects the comparator fall time; this affects the valid output driver delay.

6.10. Precharge Time

This section derives an estimate for the extra time required after an access before the next access can begin. This difference between the access time and the cycle time can vary widely depending on the circuit techniques used. Usually the cycle time is a modest percentage larger than the access time, but in pipelined or post-charge circuits [6, 1] the cycle time can be less than the access time. We have chosen to model a more conventional structure with the cycle time equal to the access time plus the precharge.

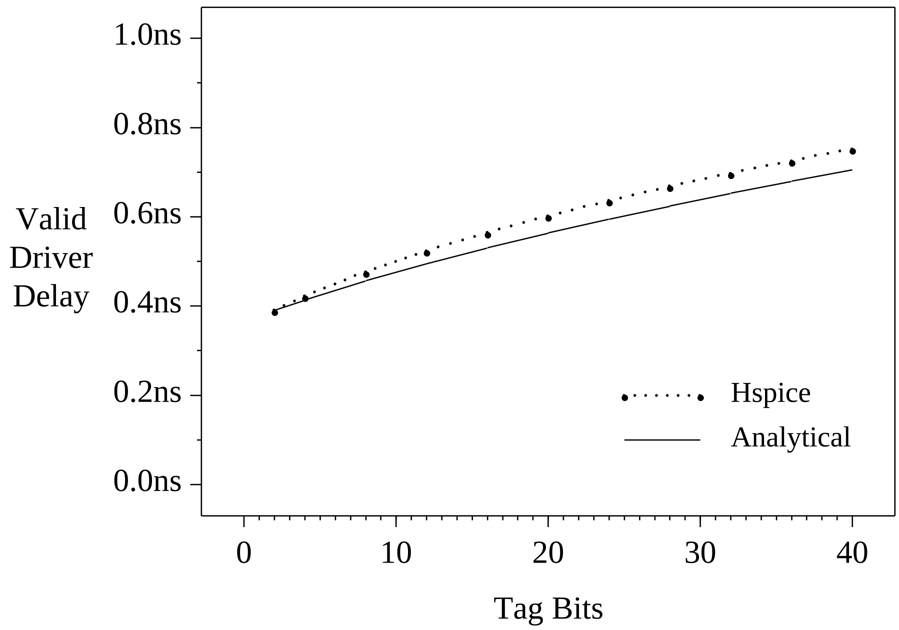

Figure 43: Valid output driver delay

Since the decoder, comparator, and bitlines need to be precharged, the extra time can be written as:

$$\text{cycle time} - \text{access time} = \max(\text{data wordline fall time} + \text{data bitline charge},\ \text{tag wordline fall time} + \text{tagbitline charge},\ \text{comparator charge time},\ \text{decoder charge time})$$

(note that the asserted wordline has to fall before the bitlines can be discharged).

The precharge times for the four precharged elements are somewhat arbitrary, since the precharging transistors can be scaled in proportion to the loads they are driving, while presenting a parasitic capacitance proportional to the other loads. If this is done then only the delay driving the precharge transistors changes with cache size. The precharge transistors are assumed to be driven by a taper buffer during the time the wordline is being discharged, so this time is not included in the precharge time. We assume that the time for the wordline to fall and bitline to rise in the data array is the dominant term in the above equation; therefore, the comparator, decoder, and tag bitline charge time will also be ignored.

Assuming the wordline drivers have properly ratioed NMOS and PMOS transistors, the wordline fall time is the same as the wordline rise time derived earlier. After the wordline has dropped, it is necessary to wait until the two bitlines (*bit* and *bitbar*) are within $V_{bitsense}/2$ of each other. It is assumed that the bitline precharging transistors are such that a constant (over all cache organization) bitline charge time is obtained. This constant will, of course, be technology dependent. In the model, we assume that this constant is equal to four inverter delays (each with a fanout of four):

$$T_{prebitline} = 4\,delay_{fall}(t_f, 0, 0.5, 0.5)$$

where

$$t_f = res_{p,on}(W_{decinvp}) \times [draincap_n(W_{decinvn}, 1) + draincap_p(W_{decinvp}, 1) + 4\,gatecap(W_{decinvp} + W_{decinvn}, 0)]$$

The total precharge time can, therefore, be written as:

$$T_{precharge} = T_{wordline,data} + T_{prebitline} \tag{37}$$

6.11. Access and Cycle Times

This section shows how the equations derived in the previous sections can be combined to give the hit access and cycle times of a cache. First consider the access time (the time after the address inputs are valid until the requested data is driven from the cache). For a direct-mapped cache, the access time is simply the larger of the path through the tag array or the path through the data array:

$$T_{access,dm} = \max (T_{dataside} + T_{outdrive,data} \ , \ T_{tagside,dm} \tag{38}$$

where:

$$T_{dataside} = T_{decoder,data} + T_{wordline,data} + T_{bitline,data} + T_{sense,data}$$

$$T_{tagside,dm} = T_{decoder,tag} + T_{wordline,tag} + T_{bitline,tag} + T_{sense,tag} + T_{compare} + T_{valid})$$

In a set-associative cache, the tag array must be read before the data signals can be driven. Thus, the access time of a set-associative cache can be written as:

$$T_{access,sa} = \max (T_{dataside} \ , T_{tagside,sa}) + T_{outdrive,data} \tag{39}$$

where:

$$T_{tagside,sa} = T_{decode,tag} + T_{wordline,tag} + T_{bitline,tag} + T_{sense,tag} + T_{compare} + T_{muxdriver} + T_{outdrive,inv}$$

Figures 44 to 47 show analytical and Hspice estimations of the data and tag sides for direct-mapped and 4-way set-associative caches. To gather these results, the model was first used to find the optimal array organization parameters via exhaustive search for each cache size. These optimum parameters are shown in the figures (the six numbers associated with each point correspond to N_{dwl}, N_{dbl}, N_{spd}, N_{twl}, N_{tbl}, and N_{tspd} in that order). The parameters were then used in the Hspice model. As the graphs show, the difference between the analytical and Hspice results is less than 10% in every case.

The cycle time of the cache (the minimum time between the start of consecutive accesses) is the access time plus the time to precharge the bitline, comparator, and internal decoder bus. Using Equation 37, the cycle time can be written as:

$$T_{cycle} = T_{access} + T_{precharge} \tag{40}$$

7. Applications of Model

This section gives examples of how the analytical model can be used to quickly gather data that can be used in architectural studies.

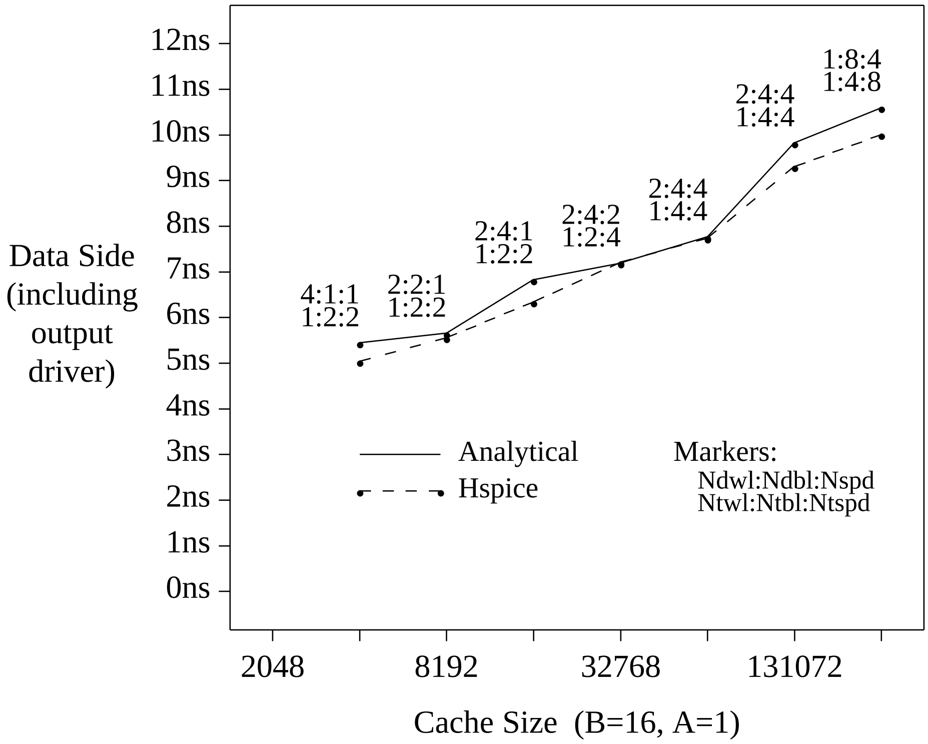

Figure 44: Direct mapped: $T_{dataside} + T_{outdrive,data}$

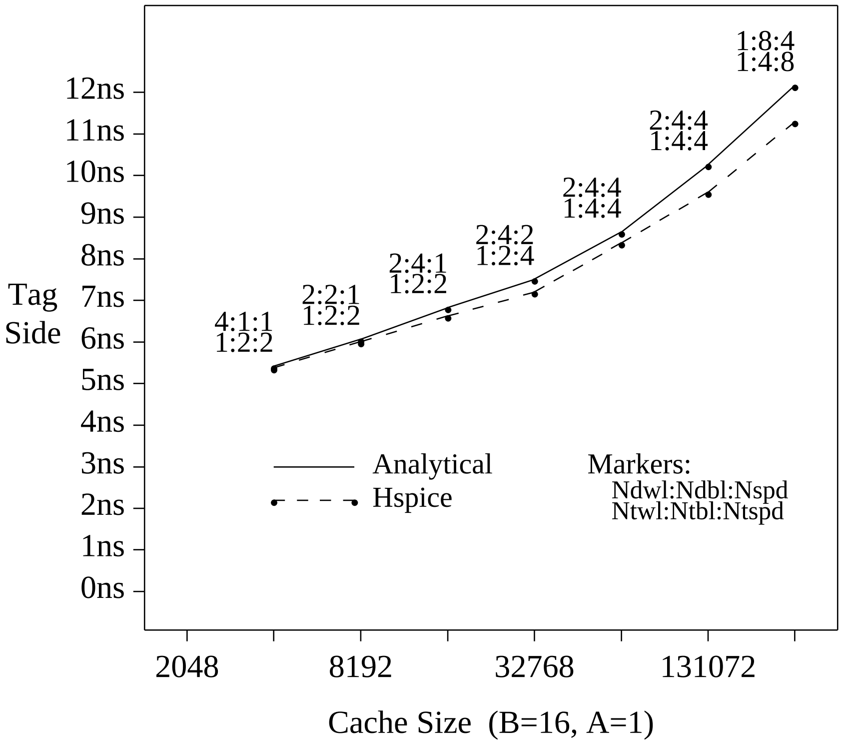

Figure 45: Direct mapped: $T_{tagside,dm}$

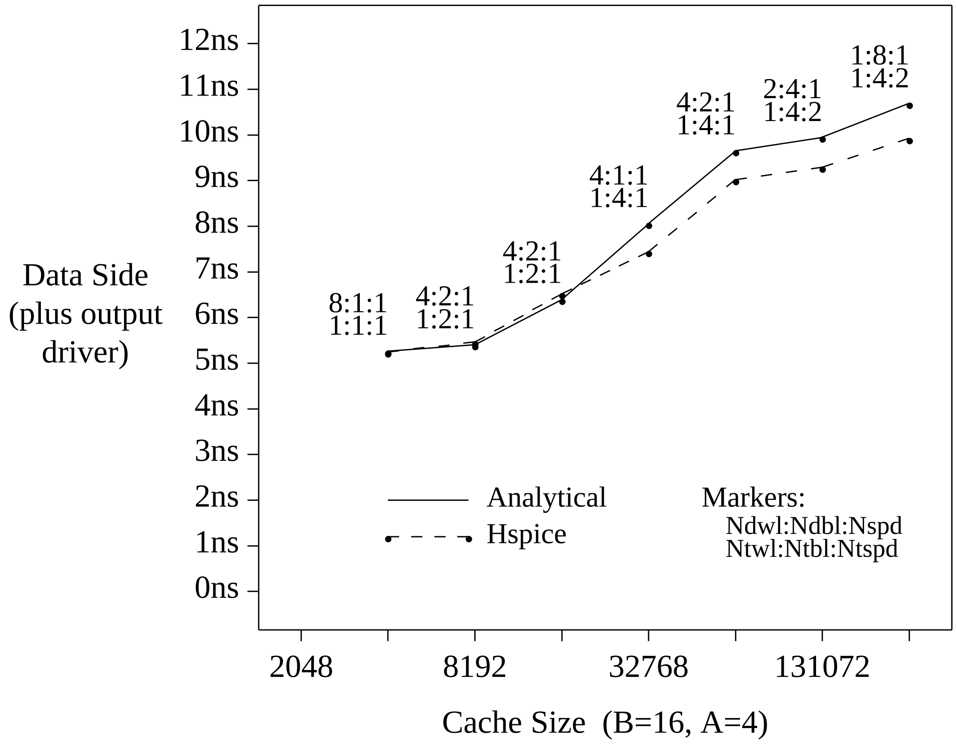

Figure 46: 4-way set associative: $T_{dataside} + T_{outdrive,data}$

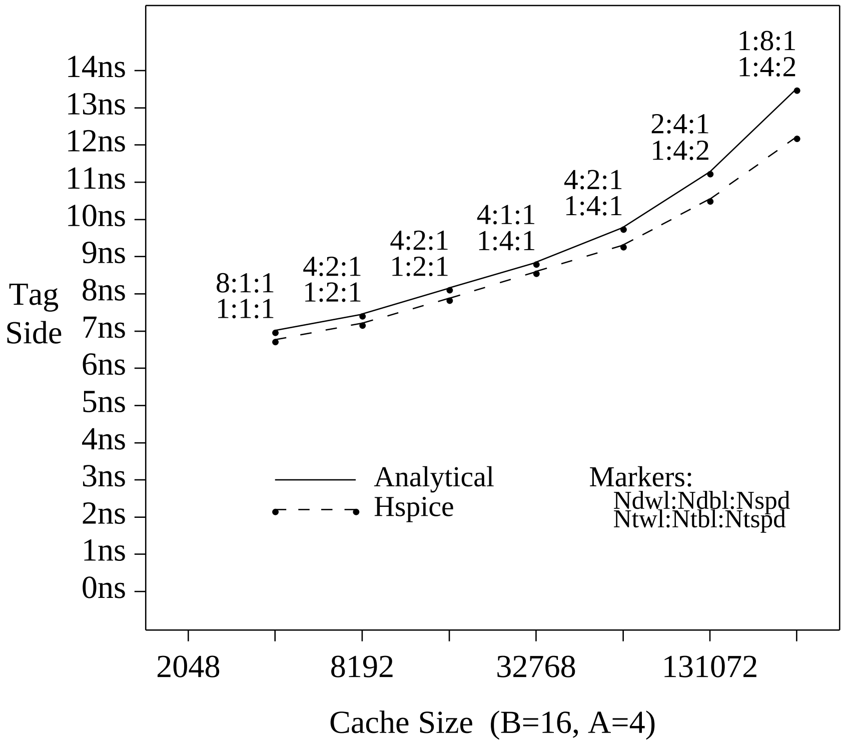

Figure 47: 4-way set associative: $T_{tagside,sa}$

7.1. Cache Size

First consider Figures 48 and 49. These graphs show how the cache size affects the cache access and cycle times in a direct-mapped and 4-way set-associative cache. In these graphs (and all graphs in the remainder of this report), $b_o = 64$ and $b_{addr} = 32$. For each cache size, the optimum array organization parameters were found (these optimum parameters are shown in the graphs as before; the six numbers associated with each point correspond to N_{dwl}, N_{dbl}, N_{spd}, N_{twl}, N_{tbl}, and N_{tspd} in that order), and the corresponding access and cycle times were plotted. In addition, the graph breaks down the access time into several components.

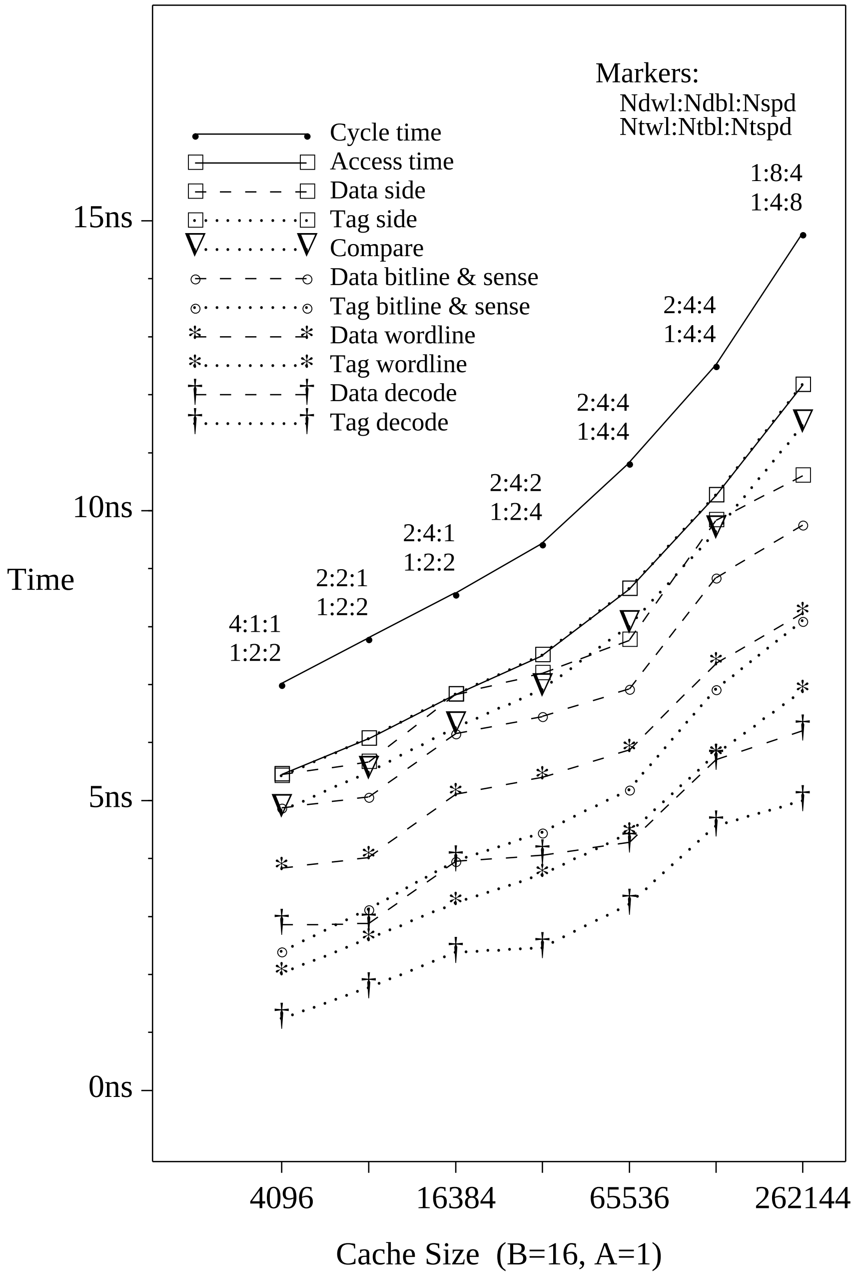

Figure 48: Access/cycle time as a function of cache size for direct-mapped cache

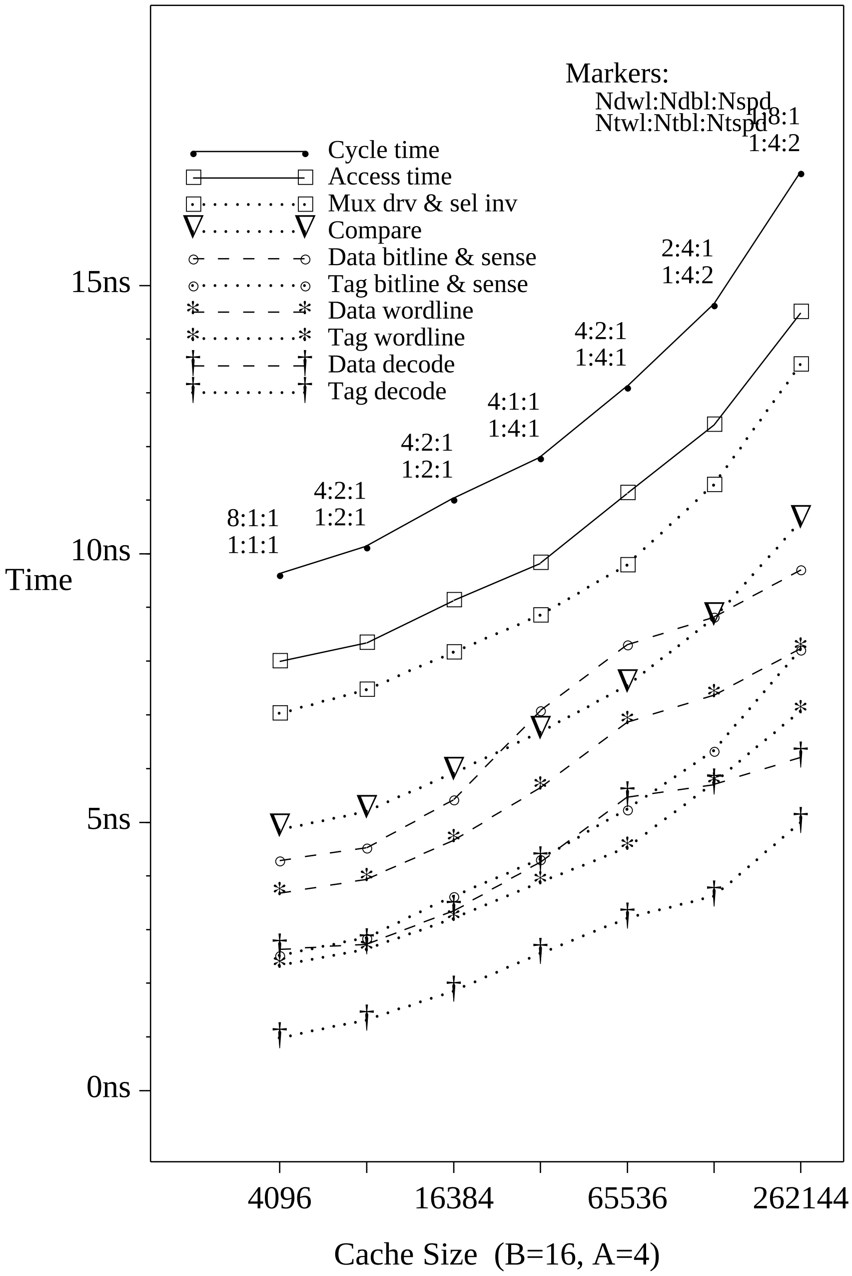

Figure 49: Access/cycle time as a function of cache size for set-associative cache

There are several observations that can be made from the graphs. Starting from the bottom, it is clear that the time through the data array decoders is always longer than the time through the tag array decoders. For all the organizations selected, there are more data subarrays ($N_{dwl}{\times}N_{dbl}$) than tag subarrays ($N_{twl}{\times}N_{tbl}$). The more data arrays, the slower the first decoder stage.

In all caches shown, the comparator is responsible for a significant portion of the access time. Another interesting trend is that the tag side is always the critical path in the cache access. In the direct-mapped cases, organizations are found which result in very closely matched tag and data sides, while in the set-associative case, the paths are not matched nearly as well. This is due primarily to the delay of the multiplexor driver.

7.2. Block Size

Figures 50 and 51 show how the access and cycle times are affected by the block size (the cache size is kept constant). In the direct-mapped graph, the access and cycle times drop as the block size increases. Most of this is due to a drop in the decoder delay (a larger block decreases the depth of each array and reduces the number of tags required).

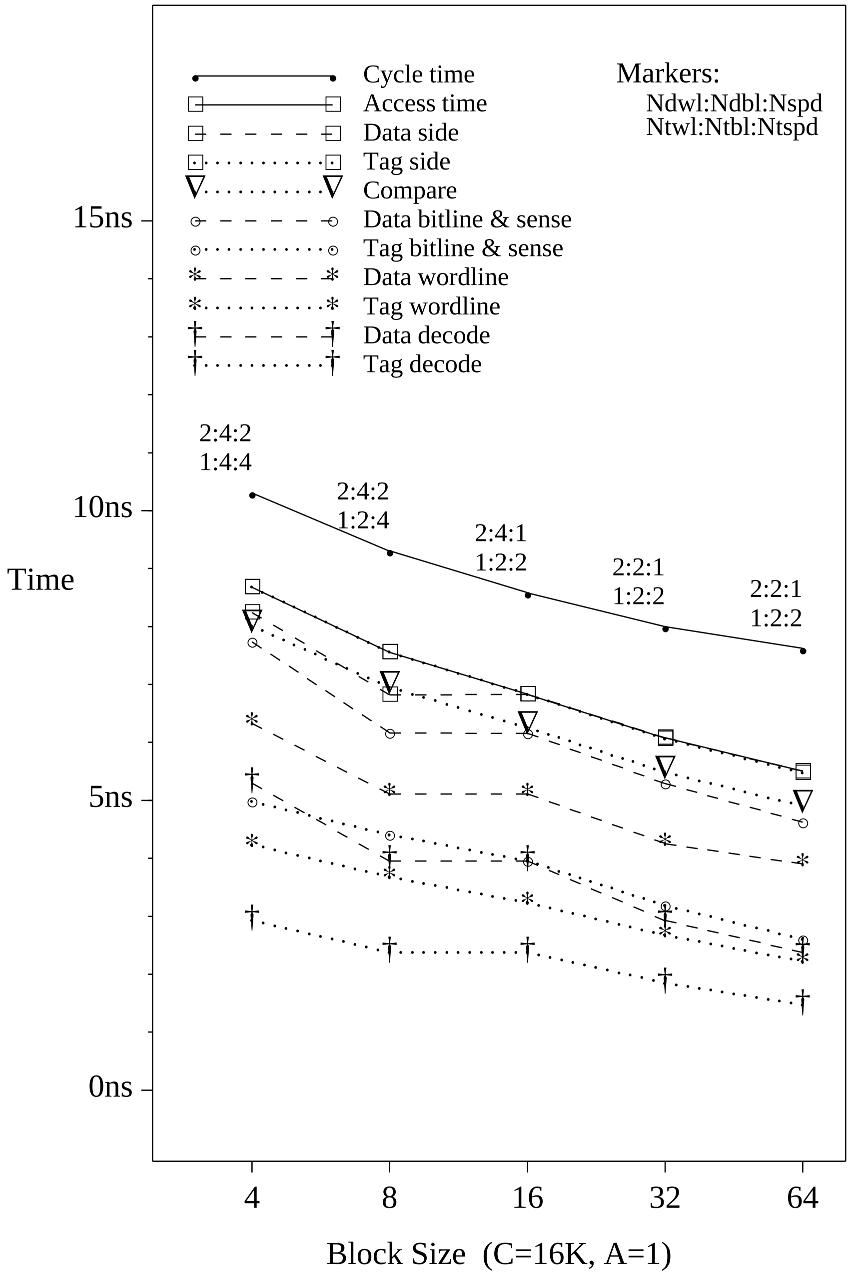

Figure 50: Access/cycle time as a function of block size for direct-mapped cache

In the set-associative case, the access and cycle time begins to increase as the block size gets above 32. This is due to the output driver; a larger block size means more drivers share the same

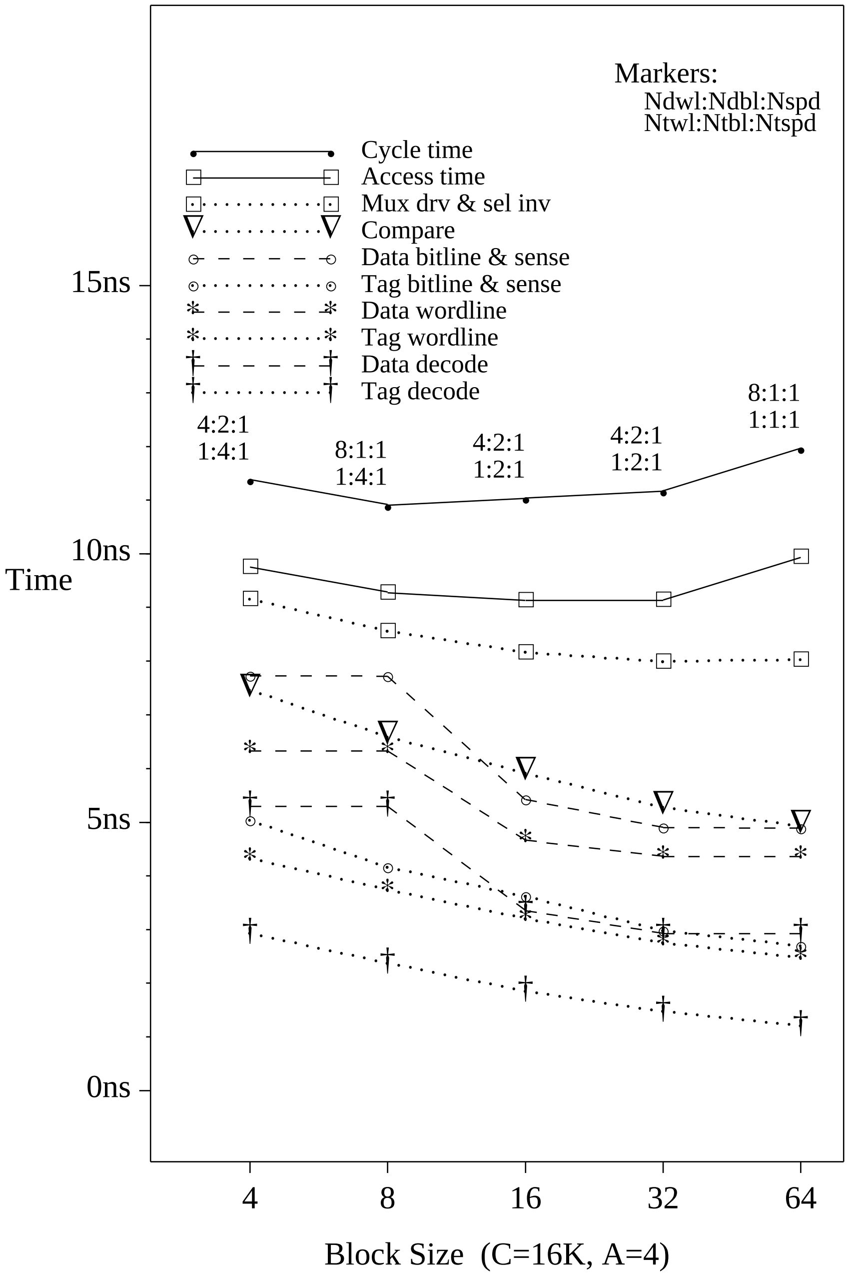

Figure 51: Access/cycle time as a function of block size for set-associative cache

cache output line, so there is more loading at the output of each driver. This trend can also be seen in the direct-mapped case, but it is much less pronounced. The number of output drivers that share a line is proportional to *A*, so the proportion of the total output capacitance that is the drain capacitance of other output drivers is smaller in a direct-mapped cache than in the 4-way set associative cache. Also, in the direct-mapped case, the slower output driver only affects the data side, and it is the tag side that dictates the access time in all the organizations shown.

7.3. Associativity

Finally, consider Figures 52 and 53 which show how the associativity affects the access and cycle time of a 16KB and 64KB cache. As can be seen, there is a significant step between a direct-mapped and a 2-way set-associative cache, but a much smaller jump between a 2-way and a 4-way cache (this is especially true in the larger cache). As the associativity increases further, the access and cycle time begin to increase more dramatically.

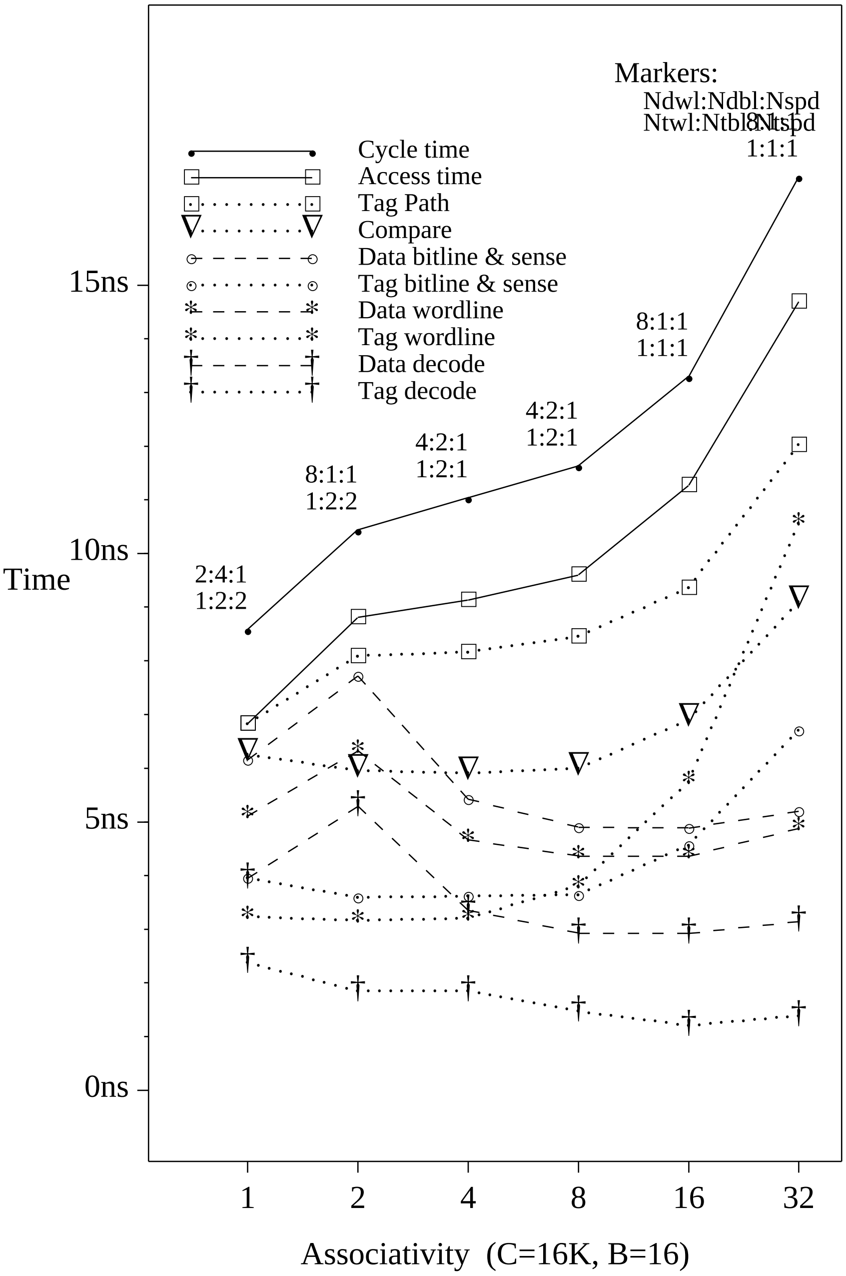

Figure 52: Access/cycle time as a function of associativity for 16K cache

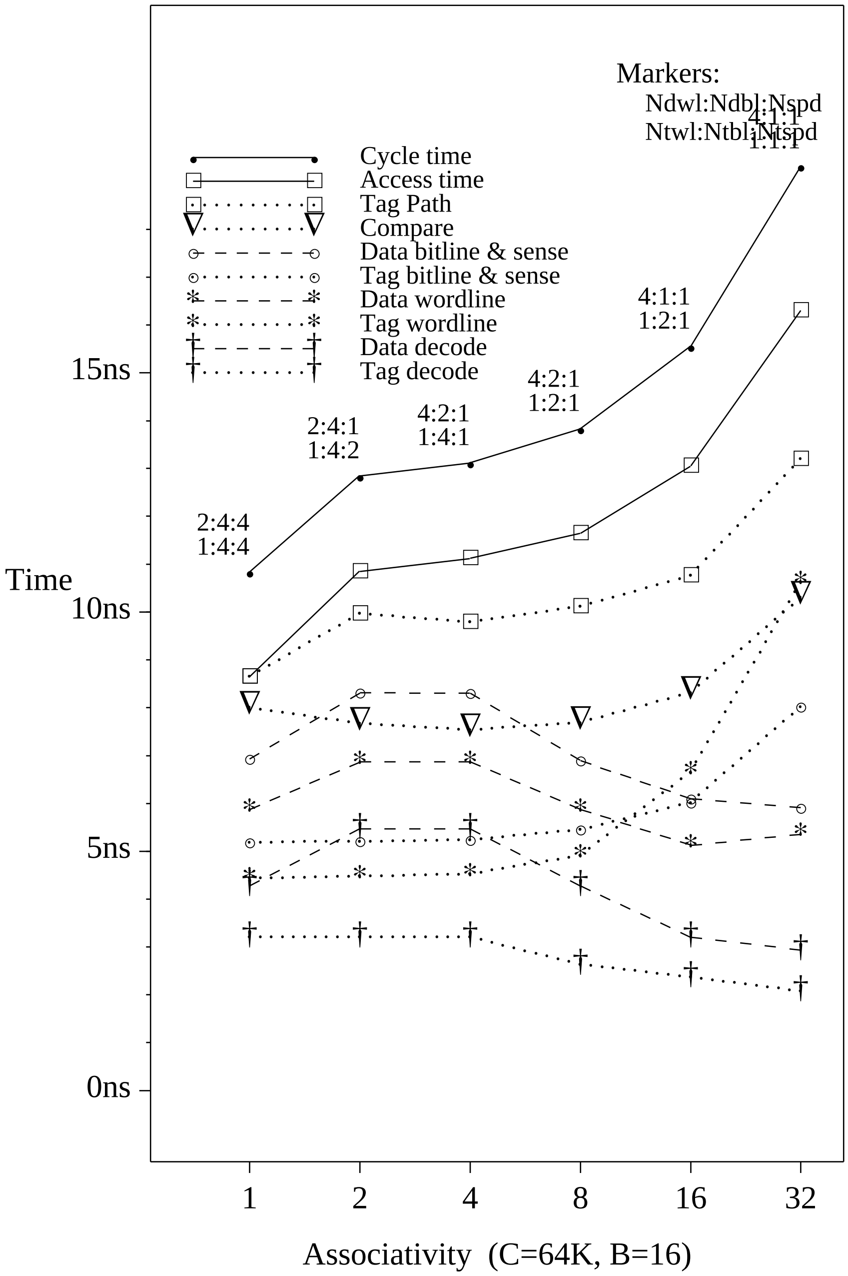

Figure 53: Access/cycle time as a function of associativity for 64K cache

The real cost of associativity can be seen by looking at the tag path curve in either graph. For a direct-mapped cache, this is simply the time to output the valid signal, while in a set-associative cache, this is the time to drive the multiplexor select signals. Also, in a direct-mapped cache, the output driver time is hidden by the time to read and compare the tag. In a set-associative cache, the tag array access, compare, and multiplexor driver must be completed before the output driver can begin to send the result out of the cache.

Looking at the 16KB cache results, there seems to be an anomaly in the data array decoder at A=2. This is due to a larger N_{dwl} at this point. This doesn't affect the overall access time, however, since the tag access is the critical path.

Another anomaly appears for large associativities, in which the bitline appears to have a negative delay. For slow rising wordlines, the bitline can switch completely (recall that the two bitlines only need to be V_{sense} volts apart) before the wordline has reached its gate threshold voltage.

8. Conclusions

In this report, we have presented an analytical model for the access and cycle time of a cache. By comparing the model to an Hspice model, the model was shown to be accurate to within 10%. The computational complexity, however, is considerably less than Hspice; measurements show the model to be over 100,000 times faster than Hspice.

It is dangerous to make too many conclusions from the graphs presented in this report. Figures 52 and 53 seem to imply that a direct-mapped cache is always the best. While it is always the fastest, it is important to remember that the direct-mapped cache will have the lowest hit-rate. Hit rate data obtained from a trace-driven simulation (or some other means) must be included in the analysis before the various cache alternatives can be fairly compared. Similarly, a small cache has a lower access time, but will also have a lower hit rate. In [4], it was found that when the hit rate and cycle time are both taken into account, there is an optimum cache size between the two extremes.

Acknowledgments

The authors wish to thank Stefanos Sidiropoulos for his circuit advice, as well has Stuart Oberman for his help setting up the Hspice simulations. Stefanos Sidiropoulos and Russell Kao provided very helpful comments on an early draft of the manuscript.

Appendix I. Circuit Parameters

The transistor sizes and threshold voltages used in the circuits in this report are given in Table I-1 (all transistor lengths are 0.8µm).

Stage	Symbol	Value
Decoder Driver	$W_{decdrivep}$	100µm
	$W_{decdriven}$	50µm
	$v_{thdecdrive}$	0.438
Decoder NAND	$W_{dec3to8p}$	60µm
	$W_{dec3to8n}$	90µm
	$v_{thdec3to8}$	0.561
Decoder NOR	$W_{decnorp}$	12µm
	$W_{decnorn}$	2.4µm
	$v_{thdecnor}$	
	(one input)	0.503
	(two inputs)	0.452
	(three inputs)	0.417
	(four inputs)	0.390
Decoder inverter	$W_{decinvp}$	10µm
	$W_{decinvn}$	5µm
	$v_{thdecinv}$	0.456
Wordline driver	$W_{worddrivep}$	varies
	$W_{worddriven}$	varies
	$v_{thworddrive}$	0.456
	k_{rise}	0.4ns
Tag wordline driver	$W_{tagwordp}$	10µm
	$W_{tagwordn}$	5µm
	$v_{thtagworddrive}$	0.456
Memory Cell (Fig 19)	W_a	1µm
	W_b	3µm
	W_d	4µm
	$v_{thwordline}$	0.456
	BitWidth	8.0µm
	BitHeight	16.0µm
Bitlines	$W_{bitpreequ}$	80µm
	$W_{bitmuxn}$	10µm
	V_{bitpre}	3.3 volts
	$V_{bitsense}$	0.1 volts
	v_t	1.09 volts

Sense Amp (Fig 29)	Q1-Q4	4µm
	Q5-Q6	8µm
	Q7-Q10	8µm
	Q11-Q12	16µm
	Q13-Q14	8µm
	Q15	16µm
	$t_{sense,data}$	0.58ns
	$t_{sense,tag}$	0.26ns
	$tfall_{sense,data}$	0.70ns
	$tfall_{sense,tag}$	0.70ns
Comparator inverter 1	$W_{compinvp1}$	10µm
	$W_{compinvn1}$	6µm
	$v_{thcompinv1}$	0.437
Comparator inverter 2	$W_{compinvp2}$	20µm
	$W_{compinvn2}$	12µm
	$v_{thcompinv2}$	0.437
Comparator inverter 3	$W_{compinvp3}$	40µm
	$W_{compinvn3}$	24µm
	$v_{thcompinv3}$	0.437
Comparator eval	$W_{evalinvp}$	20µm
	$W_{evalinvn}$	80µm
	$v_{thevalinv}$	0.267
Comparator	W_{compp}	30µm
	W_{compn}	10µm
Mux Driver Stage 1	$W_{muxdrv1p}$	50µm
	$W_{muxdrv1n}$	30µm
	$v_{thmuxdrv1}$	0.437
Mux Driver Stage 2	$W_{muxdrvnorp}$	80µm
	$W_{muxdrvnorn}$	20µm
	$v_{thmuxdrvnor}$	0.486
Mux Driver Stage 3	$W_{muxdrvselp}$	20µm
	$W_{muxdrvseln}$	12µm
	$v_{thmuxdrvsel}$	0.437
Output Driver (sel inv)	$W_{outdrvselp}$	20µm
	$W_{outdrvseln}$	12µm
	$v_{thoutdrvsel}$	0.437
Output Driver NAND	$W_{outdrvnandp}$	10µm
	$W_{outdrvnandn}$	24µm

	$v_{thoutdrvnand}$	0.441
Output Driver NOR	$W_{outdrvnorp}$	40µm
	$W_{outdrvnorn}$	6µm
	$v_{thoutdrvnor}$	0.431
Output Driver (final)	$W_{outdriverp}$	80µm
	$W_{outdrivern}$	48µm
	$v_{thoutdriver}$	0.425
	C_{out}	0.5 pF

Table I-1: Transistor sizes and threshold voltages

Appendix II. Technology Parameters

The technology parameters used in this report are given in Table II-1. The Spice models used from [3] are given in Figure II-1.

Parameter	Value
$C_{bitmetal}$	4.4 fF/bit
C_{gate}	1.95 fF/μm^2
$C_{gatepass}$	1.45 fF/μm^2
$C_{ndiffarea}$	0.137 fF/μm^2
$C_{ndiffside}$	0.275 fF/μm
$C_{ndiffgate}$	0.401 fF/μm
$C_{pdiffarea}$	0.343 fF/μm^2
$C_{pdiffside}$	0.275 fF/μm
$C_{pdiffgate}$	0.476 fF/μm
$C_{polywire}$	0.25 fF/μm
$C_{wordmetal}$	1.8 fF/bit
L_{eff}	0.8 μm
$R_{bitmetal}$	0.320 Ω/bit
$R_{n,switching}$	25800 Ω*μm
$R_{n,on}$	9723 Ω*μm
$R_{p,switching}$	61200 Ω*μm
$R_{p,on}$	22400 Ω*μm
$R_{wordmetal}$	0.080 Ω/bit
V_{dd}	5 volts

Table II-1: 0.8μm CMOS process parameters

```
.model nt nmos ( level=3
+   vto=0.77        tox=1.65e-8     uo=570          gamma=0.80
+   vmax=2.7e5      theta=0.404     eta=0.04        kappa=1.2
+   phi=0.90        nsub=8.8e16     nfs=4e11        xj=0.2u
+   cj=2e-4         mj=0.389        cjsw=4.00e-10   mjsw=0.26
+   pb=0.80         cgso=2.1e-10    cgdo=2.1e-10    delta=0.0
+   ld=0.0001u      rsh=0.5 )
*

.model pt pmos ( level=3
+   vto=-0.87       tox=1.65e-8     uo=145          gamma=0.73
+   vmax=0.00       theta=0.233     eta=0.028       kappa=0.04
+   phi=0.90        nsub=9.0e16     nfs=4e11        xj=0.2u
+   cj=5e-4         mj=0.420        cjsw=4.00e-10   mjsw=0.31
+   pb=0.80         cgso=2.7e-10    cgdo=2.7e-10    delta=0.0
+   ld=0.0001u      rsh=0.5 )
```

Figure II-1: Generic 0.8um CMOS Spice parameters [3]

References

[1] Terry Chappell, et. al. A 2ns cycle, 3,8ns access 512kb CMOS ECL RAM with a fully pipelined architecture. *IEEE Journal of Solid-State Circuits, Vol. 26, No. 11* :1577-1585, Nov., 1991.

[2] Mark A. Horowitz. *Timing Models for MOS Circuits*. Technical Report Technical Report SEL83-003, Integrated Circuits Laboratory, Stanford University, 1983.

[3] Mark G. Johnson and Norman P. Jouppi. Transistor model for a Synthetic 0.8um CMOS process. *Class notes for Stanford University EE371* , Spring, 1990.

[4] Norman P. Jouppi and Steven J.E. Wilton. Tradeoffs in Two-Level On-Chip Caching. In *Proceedings of the 21th Annual International Symposium on Computer Architecture*. 1994.

[5] Johannes M. Mulder, Nhon T. Quach, Michael J. Flynn. An Area Model for On-Chip Memories and its Application. *IEEE Journal of Solid-State Circuits, Vol. 26, No. 2* :98-106, Feb., 1991.

[6] Robert J. Proebsting. Post Charge Logic permits 4ns 18K CMOS RAM. 1987.

[7] Jorge Rubinstein and Paul Penfield and Mark A. Horowitz. Signal Delay in RC Tree Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 2, No. 3* :202-211, July, 1983.

[8] Tomohisa Wada, Suresh Rajan, Steven A. Przybylski. An Analytical Access Time Model for On-Chip Cache Memories. *IEEE Journal of Solid-State Circuits, Vol. 27, No. 8* :1147-1156, Aug., 1992.

[9] Neil H. E. Weste and Kamran Eshraghian. *Principles of CMOS VLSI Design.* Addison Wesley, 1993. Second edition.

WRL Research Reports

‘‘Titan System Manual.’’
Michael J. K. Nielsen.
WRL Research Report 86/1, September 1986.

‘‘Global Register Allocation at Link Time.’’
David W. Wall.
WRL Research Report 86/3, October 1986.

‘‘Optimal Finned Heat Sinks.’’
William R. Hamburgen.
WRL Research Report 86/4, October 1986.

‘‘The Mahler Experience: Using an Intermediate Language as the Machine Description.’’
David W. Wall and Michael L. Powell.
WRL Research Report 87/1, August 1987.

‘‘The Packet Filter: An Efficient Mechanism for User-level Network Code.’’
Jeffrey C. Mogul, Richard F. Rashid, Michael J. Accetta.
WRL Research Report 87/2, November 1987.

‘‘Fragmentation Considered Harmful.’’
Christopher A. Kent, Jeffrey C. Mogul.
WRL Research Report 87/3, December 1987.

‘‘Cache Coherence in Distributed Systems.’’
Christopher A. Kent.
WRL Research Report 87/4, December 1987.

‘‘Register Windows vs. Register Allocation.’’
David W. Wall.
WRL Research Report 87/5, December 1987.

‘‘Editing Graphical Objects Using Procedural Representations.’’
Paul J. Asente.
WRL Research Report 87/6, November 1987.

‘‘The USENET Cookbook: an Experiment in Electronic Publication.’’
Brian K. Reid.
WRL Research Report 87/7, December 1987.

‘‘MultiTitan: Four Architecture Papers.’’
Norman P. Jouppi, Jeremy Dion, David Boggs, Michael J. K. Nielsen.
WRL Research Report 87/8, April 1988.

‘‘Fast Printed Circuit Board Routing.’’
Jeremy Dion.
WRL Research Report 88/1, March 1988.

‘‘Compacting Garbage Collection with Ambiguous Roots.’’
Joel F. Bartlett.
WRL Research Report 88/2, February 1988.

‘‘The Experimental Literature of The Internet: An Annotated Bibliography.’’
Jeffrey C. Mogul.
WRL Research Report 88/3, August 1988.

‘‘Measured Capacity of an Ethernet: Myths and Reality.’’
David R. Boggs, Jeffrey C. Mogul, Christopher A. Kent.
WRL Research Report 88/4, September 1988.

‘‘Visa Protocols for Controlling Inter-Organizational Datagram Flow: Extended Description.’’
Deborah Estrin, Jeffrey C. Mogul, Gene Tsudik, Kamaljit Anand.
WRL Research Report 88/5, December 1988.

‘‘SCHEME->C A Portable Scheme-to-C Compiler.’’
Joel F. Bartlett.
WRL Research Report 89/1, January 1989.

‘‘Optimal Group Distribution in Carry-Skip Adders.’’
Silvio Turrini.
WRL Research Report 89/2, February 1989.

‘‘Precise Robotic Paste Dot Dispensing.’’
William R. Hamburgen.
WRL Research Report 89/3, February 1989.

‘‘Simple and Flexible Datagram Access Controls for Unix-based Gateways.’’
Jeffrey C. Mogul.
WRL Research Report 89/4, March 1989.

‘‘Spritely NFS: Implementation and Performance of Cache-Consistency Protocols.’’
V. Srinivasan and Jeffrey C. Mogul.
WRL Research Report 89/5, May 1989.

‘‘Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines.’’
Norman P. Jouppi and David W. Wall.
WRL Research Report 89/7, July 1989.

‘‘A Unified Vector/Scalar Floating-Point Architecture.’’
Norman P. Jouppi, Jonathan Bertoni, and David W. Wall.
WRL Research Report 89/8, July 1989.

‘‘Architectural and Organizational Tradeoffs in the Design of the MultiTitan CPU.’’
Norman P. Jouppi.
WRL Research Report 89/9, July 1989.

‘‘Integration and Packaging Plateaus of Processor Performance.’’
Norman P. Jouppi.
WRL Research Report 89/10, July 1989.

‘‘A 20-MIPS Sustained 32-bit CMOS Microprocessor with High Ratio of Sustained to Peak Performance.’’
Norman P. Jouppi and Jeffrey Y. F. Tang.
WRL Research Report 89/11, July 1989.

‘‘The Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance.’’
Norman P. Jouppi.
WRL Research Report 89/13, July 1989.

‘‘Long Address Traces from RISC Machines: Generation and Analysis.’’
Anita Borg, R.E.Kessler, Georgia Lazana, and David W. Wall.
WRL Research Report 89/14, September 1989.

‘‘Link-Time Code Modification.’’
David W. Wall.
WRL Research Report 89/17, September 1989.

‘‘Noise Issues in the ECL Circuit Family.’’
Jeffrey Y.F. Tang and J. Leon Yang.
WRL Research Report 90/1, January 1990.

‘‘Efficient Generation of Test Patterns Using Boolean Satisfiablilty.’’
Tracy Larrabee.
WRL Research Report 90/2, February 1990.

‘‘Two Papers on Test Pattern Generation.’’
Tracy Larrabee.
WRL Research Report 90/3, March 1990.

‘‘Virtual Memory vs. The File System.’’
Michael N. Nelson.
WRL Research Report 90/4, March 1990.

‘‘Efficient Use of Workstations for Passive Monitoring of Local Area Networks.’’
Jeffrey C. Mogul.
WRL Research Report 90/5, July 1990.

‘‘A One-Dimensional Thermal Model for the VAX 9000 Multi Chip Units.’’
John S. Fitch.
WRL Research Report 90/6, July 1990.

‘‘1990 DECWRL/Livermore Magic Release.’’
Robert N. Mayo, Michael H. Arnold, Walter S. Scott, Don Stark, Gordon T. Hamachi.
WRL Research Report 90/7, September 1990.

''Pool Boiling Enhancement Techniques for Water at Low Pressure.''
Wade R. McGillis, John S. Fitch, William R. Hamburgen, Van P. Carey.
WRL Research Report 90/9, December 1990.

''Writing Fast X Servers for Dumb Color Frame Buffers.''
Joel McCormack.
WRL Research Report 91/1, February 1991.

''A Simulation Based Study of TLB Performance.''
J. Bradley Chen, Anita Borg, Norman P. Jouppi.
WRL Research Report 91/2, November 1991.

''Analysis of Power Supply Networks in VLSI Circuits.''
Don Stark.
WRL Research Report 91/3, April 1991.

''TurboChannel T1 Adapter.''
David Boggs.
WRL Research Report 91/4, April 1991.

''Procedure Merging with Instruction Caches.''
Scott McFarling.
WRL Research Report 91/5, March 1991.

''Don't Fidget with Widgets, Draw!.''
Joel Bartlett.
WRL Research Report 91/6, May 1991.

''Pool Boiling on Small Heat Dissipating Elements in Water at Subatmospheric Pressure.''
Wade R. McGillis, John S. Fitch, William R. Hamburgen, Van P. Carey.
WRL Research Report 91/7, June 1991.

''Incremental, Generational Mostly-Copying Garbage Collection in Uncooperative Environments.''
G. May Yip.
WRL Research Report 91/8, June 1991.

''Interleaved Fin Thermal Connectors for Multichip Modules.''
William R. Hamburgen.
WRL Research Report 91/9, August 1991.

''Experience with a Software-defined Machine Architecture.''
David W. Wall.
WRL Research Report 91/10, August 1991.

''Network Locality at the Scale of Processes.''
Jeffrey C. Mogul.
WRL Research Report 91/11, November 1991.

''Cache Write Policies and Performance.''
Norman P. Jouppi.
WRL Research Report 91/12, December 1991.

''Packaging a 150 W Bipolar ECL Microprocessor.''
William R. Hamburgen, John S. Fitch.
WRL Research Report 92/1, March 1992.

''Observing TCP Dynamics in Real Networks.''
Jeffrey C. Mogul.
WRL Research Report 92/2, April 1992.

''Systems for Late Code Modification.''
David W. Wall.
WRL Research Report 92/3, May 1992.

''Piecewise Linear Models for Switch-Level Simulation.''
Russell Kao.
WRL Research Report 92/5, September 1992.

''A Practical System for Intermodule Code Optimization at Link-Time.''
Amitabh Srivastava and David W. Wall.
WRL Research Report 92/6, December 1992.

''A Smart Frame Buffer.''
Joel McCormack & Bob McNamara.
WRL Research Report 93/1, January 1993.

‘‘Recovery in Spritely NFS.’’
Jeffrey C. Mogul.
WRL Research Report 93/2, June 1993.

‘‘Tradeoffs in Two-Level On-Chip Caching.’’
Norman P. Jouppi & Steven J.E. Wilton.
WRL Research Report 93/3, October 1993.

‘‘Unreachable Procedures in Object-oriented Programing.’’
Amitabh Srivastava.
WRL Research Report 93/4, August 1993.

‘‘An Enhanced Access and Cycle Time Model for On-Chip Caches.’’
Steven J.E. Wilton and Norman P. Jouppi.
WRL Research Report 93/5, July 1994.

‘‘Limits of Instruction-Level Parallelism.’’
David W. Wall.
WRL Research Report 93/6, November 1993.

‘‘Fluoroelastomer Pressure Pad Design for Microelectronic Applications.’’
Alberto Makino, William R. Hamburgen, John S. Fitch.
WRL Research Report 93/7, November 1993.

‘‘A 300MHz 115W 32b Bipolar ECL Microprocessor.’’
Norman P. Jouppi, Patrick Boyle, Jeremy Dion, Mary Jo Doherty, Alan Eustace, Ramsey Haddad, Robert Mayo, Suresh Menon, Louis Monier, Don Stark, Silvio Turrini, Leon Yang, John Fitch, William Hamburgen, Russell Kao, and Richard Swan.
WRL Research Report 93/8, December 1993.

‘‘Link-Time Optimization of Address Calculation on a 64-bit Architecture.’’
Amitabh Srivastava, David W. Wall.
WRL Research Report 94/1, February 1994.

‘‘ATOM: A System for Building Customized Program Analysis Tools.’’
Amitabh Srivastava, Alan Eustace.
WRL Research Report 94/2, March 1994.

‘‘Complexity/Performance Tradeoffs with Non-Blocking Loads.’’
Keith I. Farkas, Norman P. Jouppi.
WRL Research Report 94/3, March 1994.

‘‘A Better Update Policy.’’
Jeffrey C. Mogul.
WRL Research Report 94/4, April 1994.

‘‘Boolean Matching for Full-Custom ECL Gates.’’
Robert N. Mayo, Herve Touati.
WRL Research Report 94/5, April 1994.

WRL Technical Notes

‘‘TCP/IP PrintServer: Print Server Protocol.’’
Brian K. Reid and Christopher A. Kent.
WRL Technical Note TN-4, September 1988.

‘‘TCP/IP PrintServer: Server Architecture and Implementation.’’
Christopher A. Kent.
WRL Technical Note TN-7, November 1988.

‘‘Smart Code, Stupid Memory: A Fast X Server for a Dumb Color Frame Buffer.’’
Joel McCormack.
WRL Technical Note TN-9, September 1989.

‘‘Why Aren’t Operating Systems Getting Faster As Fast As Hardware?’’
John Ousterhout.
WRL Technical Note TN-11, October 1989.

‘‘Mostly-Copying Garbage Collection Picks Up Generations and C++.’’
Joel F. Bartlett.
WRL Technical Note TN-12, October 1989.

‘‘The Effect of Context Switches on Cache Performance.’’
Jeffrey C. Mogul and Anita Borg.
WRL Technical Note TN-16, December 1990.

‘‘MTOOL: A Method For Detecting Memory Bottlenecks.’’
Aaron Goldberg and John Hennessy.
WRL Technical Note TN-17, December 1990.

‘‘Predicting Program Behavior Using Real or Estimated Profiles.’’
David W. Wall.
WRL Technical Note TN-18, December 1990.

‘‘Cache Replacement with Dynamic Exclusion’’
Scott McFarling.
WRL Technical Note TN-22, November 1991.

‘‘Boiling Binary Mixtures at Subatmospheric Pressures’’
Wade R. McGillis, John S. Fitch, William R. Hamburgen, Van P. Carey.
WRL Technical Note TN-23, January 1992.

‘‘A Comparison of Acoustic and Infrared Inspection Techniques for Die Attach’’
John S. Fitch.
WRL Technical Note TN-24, January 1992.

‘‘TurboChannel Versatec Adapter’’
David Boggs.
WRL Technical Note TN-26, January 1992.

‘‘A Recovery Protocol For Spritely NFS’’
Jeffrey C. Mogul.
WRL Technical Note TN-27, April 1992.

‘‘Electrical Evaluation Of The BIPS-0 Package’’
Patrick D. Boyle.
WRL Technical Note TN-29, July 1992.

‘‘Transparent Controls for Interactive Graphics’’
Joel F. Bartlett.
WRL Technical Note TN-30, July 1992.

‘‘Design Tools for BIPS-0’’
Jeremy Dion & Louis Monier.
WRL Technical Note TN-32, December 1992.

‘‘Link-Time Optimization of Address Calculation on a 64-Bit Architecture’’
Amitabh Srivastava and David W. Wall.
WRL Technical Note TN-35, June 1993.

‘‘Combining Branch Predictors’’
Scott McFarling.
WRL Technical Note TN-36, June 1993.

‘‘Boolean Matching for Full-Custom ECL Gates’’
Robert N. Mayo and Herve Touati.
WRL Technical Note TN-37, June 1993.