To appear in IEEE Journal of Solid State Circuits 1996

CACTI: An Enhanced Cache Access and Cycle Time Model

Steven J.E. Wilton
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada M5S 1A4
(416) 978-1652
wilton@eecg.toronto.edu

Norman P. Jouppi
Digital Equipment Corporation Western Research Lab
250 University Avenue
Palo Alto, CA 94301
(415) 617-3305
jouppi@pa.dec.com

May 14, 1996

Abstract

This paper describes an analytical model for the access and cycle times of on-chip 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 6% of Hspice results for the circuits we have chosen.

This model extends previous models and fixes many of their major shortcomings. New features include models for the tag array, comparator, and multiplexor drivers, non-step stage input slopes, rectangular stacking of memory subarrays, a transistor-level decoder model, column-multiplexed bitlines controlled by an additional array organizational parameter, load-dependent size transistors for wordline drivers, and output of cycle times as well as access times.

Software implementing the model is available via ftp.

1 Introduction

Most computer architecture research involves investigating trade-offs between various alternatives. This can not be done adequately without a firm grasp of the costs of each alternative. For 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 chip area models [1] and access time models [2] have been published.

In [2], Wada et al. present an equation for the access time of an on-chip cache as a function of various cache parameters (cache size, associativity, block size) as well as organizational and process parameters. Unfortunately, Wada's access time model has a number of significant shortcomings. For example, the cache tag and comparator in set-associative memories are not modeled, and in practice, these often constitute the critical path. Each stage in their model (e.g., bitline, wordline) assumes that the inputs to the stage are step waveforms; actual waveforms in memories are far from steps and this can greatly impact the delay of a stage. In the Wada model, all memory subarrays are stacked linearly in a single file; this can result in aspect ratios of greater than 10:1 and overly pessimistic access times. Wada's decoder model is a gate-level model which contains no wiring parasitics. In addition, transistor sizes in Wada's model are fixed independent of the load. For example, the wordline driver is always the same size independent of the number of cells that it drives. Finally, Wada's model predicts only the cache access time, whereas both the access and cycle time are important for design comparisons.

This paper describes a significant improvement and extension of Wada's access time model. The enhanced model is called CACTI. Some of the new features are:

- a tag array model with comparator and multiplexor drivers
- non-step stage input slopes
- rectangular stacking of memory subarrays
- a transistor-level decoder model
- column-multiplexed bitlines and an additional array organizational parameter

- load-dependent transistor sizes for wordline drivers
- cycle times as well as access times

The enhancements to Wada's model can be classified into two categories. First, the assumed cache structure has been modified to more closely represent real caches. Some examples of these enhancements are the column multiplexed bitlines and the inclusion of the tag array. The second class of enhancements involve the modeling techniques used to estimate the delay of the assumed cache structure (e.g. taking into account non-step input rise times). This paper describes both classes of enhancements. After discussing the overall cache structure and model input parameters, the structural enhancements are described in Section 4. The modeling techniques used are then described in Section 5. A complete derivation of all the equations in the model is far beyond the scope of a journal paper but is available in a technical report [3].

Any model needs to be validated before the results generated using the model can be trusted. In [2], a Hspice model of the cache was used to validate the authors' analytical model. The same approach was used here; Section 6 compares the model predictions to Hspice measurements. Of course, this only shows that the analytical 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 different circuit implementations are possible. In architecture studies, however, the relative differences in access/cycle times between different cache sizes or configurations are usually more important than absolute access times. Thus, even though our model predicts absolute access times for a specific cache implementation, it can be used in a wide variety of situations when an estimate of the cost of varying an architectural parameter is required. Section 7 gives some examples of how the model can be used in architectural studies.

The model described in this paper has been implemented, and the software is available via ftp. Appendix A explains how to obtain and use the CACTI software.

2 Cache Structure

Figure 1 shows the organization of the SRAM cache being considered. 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

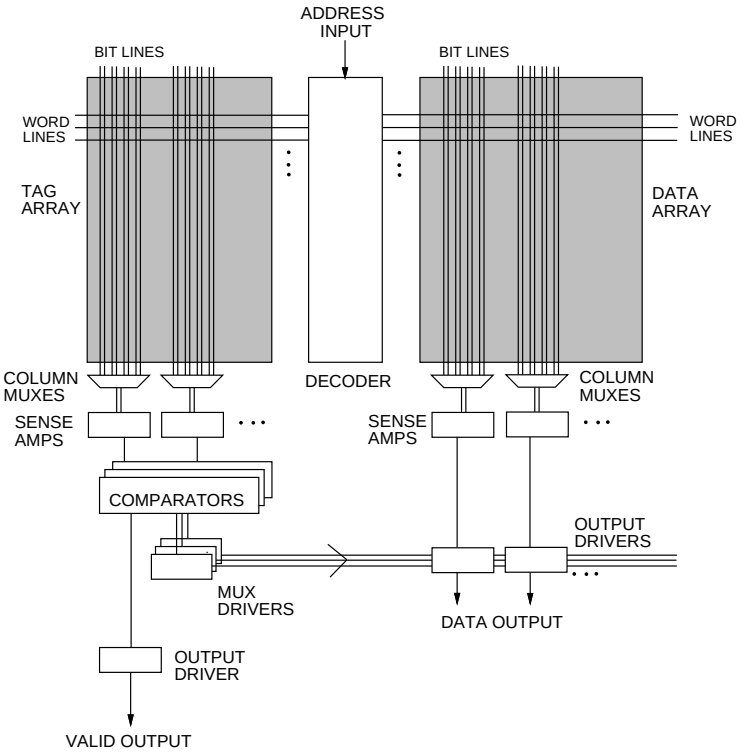


Figure 1: Cache structure

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 the contents of the selected 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.

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.

3 Cache and Array Organization Parameters

Table 1 shows the five model input parameters.

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

Table 1: CACTI input parameters

In addition, there are six array organization parameters that are used to estimate the cache access and cycle time. In the basic organization discussed by Wada [2], 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 ($S = \frac{C}{B*A}$). 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. 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. 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. The optimum values of N_{dwl}, N_{dbl}, and N_{spd} depend on the cache and block sizes, as well as the associativity.

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

4 Model Components

This section gives an overview of key portions of the cache read access and cycle time model. The access and cycle times were derived by estimating delays due to the following components:

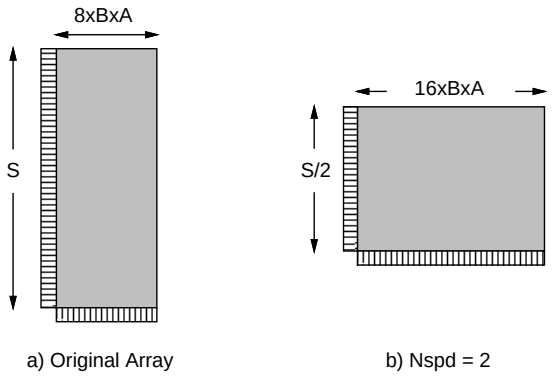


Figure 2: Cache organization parameter N_{spd}

- 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 is estimated separately and the results combined to estimate the access and cycle time of the entire cache. A complete description of each component can be found in [3]. In this paper, we focus on those parts that differ significantly from Wada's model.

4.1 Decoder

Wada's model contains a gate-level decoder model without any parasitic capacitances or resistances. It also assumes all memory sub-arrays are stacked single file in a linear array. We have used a detailed transistor-level decoder that includes both parasitic capacitances and resistances. We have also assumed that sub-arrays are placed in a two-dimensional array to minimize critical wiring parasitics.

Figure 3 shows the logical structure of the decoder architecture used in this model. The decoder in Figure 3 contains three stages. Each block in the first stage takes three address bits (in true and complement), and generates a 1-of-8 code, driving a precharged decoder bus. These 1-of-8 codes are combined using NOR gates in the second stage. The final

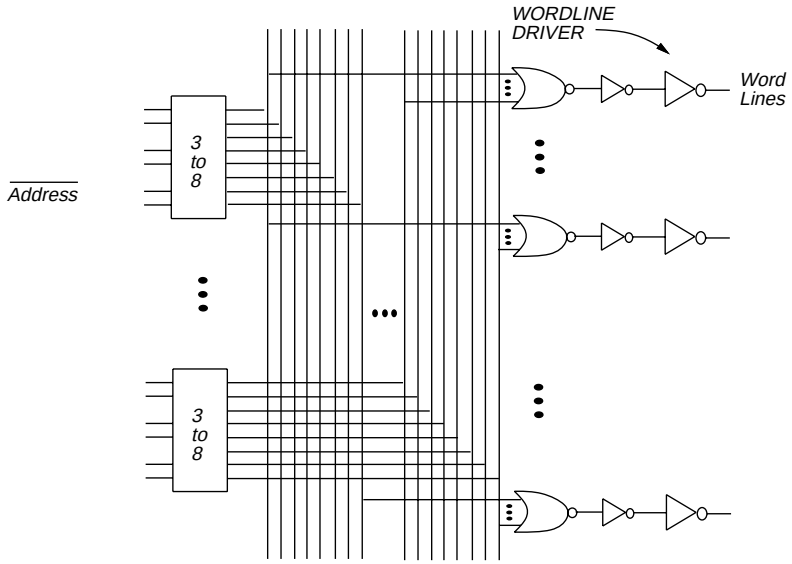


Figure 3: Single decoder structure

stage is an inverter that drives each wordline driver. We also model separate decoder driver buffers for driving the 3-to-8 decoders of the data arrays and the tag arrays.

Estimating the wire lengths in the decoder requires knowledge of the memory tile layout. As mentioned in Section 3, the memory is divided into $N_{dwl} * N_{dbl}$ subarrays; each of these arrays is $\frac{8BAN_{spd}}{N_{dwl}}$ cells wide. If these arrays were placed side-by-side, the total memory width would be $8 * B * A * N_{dbl} * N_{spd}$ cells. Instead, we assume they are grouped in two-by-two blocks, with the 3-to-8 predecode NAND gates at the center of each block; Figure 4 shows one of these blocks. This reduces the length of the connection between the decoder driver and the predecode block to approximately one quarter of the total memory width, or $2 * B * A * N_{dbl} * N_{spd}$. The length of the connection between the predecode block and the NOR gate is then (on average) half of the subarray height, which is $\frac{C}{BAN_{dbl}N_{spd}}$ cells. 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, shortening the data bus.

4.2 Variable Size Wordline Driver

The size of the wordline driver in Wada's model is independent of the number of cells attached to the wordline; this severely overestimates the wordline delay of large arrays. Our model assumes a variable-sized wordline driver. Normally, a cache designer would choose a target wordline rise time, and adjust the driver size appropriately. Rather than

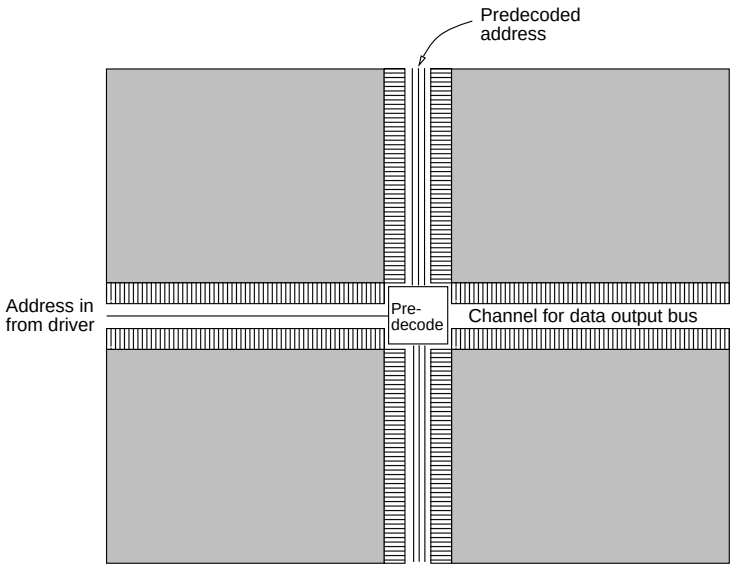


Figure 4: Memory block tiling assumptions

assuming a constant rise time for caches of all sizes, however, we assume the desired rise time (to a 50% word line swing) is:

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

where

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

and 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. This is described in Section 5.5.

4.3 Bitlines and Sense Amplifiers

Wada's model does not apply to memories with column multiplexing. Our model allows column multiplexing using NMOS pass transistors between several pairs of bitlines and a shared sense amp. In our model, the degree of column multiplexing (number of pairs of bitlines per sense amp) is $N_{spd} * N_{dbl}$.

Although we use the same sense amp as Wada's model, we precharge the bitlines to two NMOS diodes less than V_{dd} since the sense amp performs poorly with a common-mode

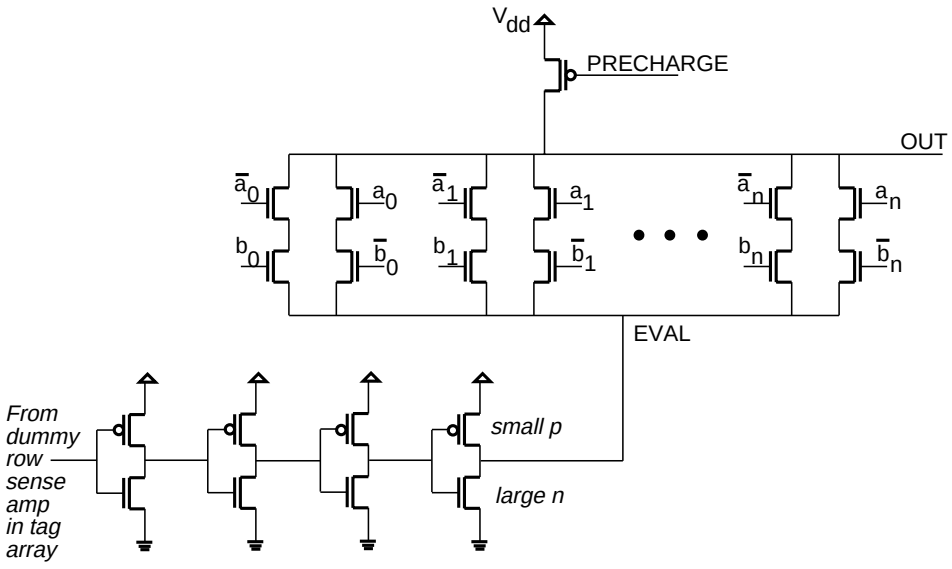


Figure 5: Comparator

voltage of V_{dd}.

4.4 Comparator

Although Wada's model gives access times for set-associative caches, it only models the data portion of a set-associative memory. However, the tag portion of a set-associative memory is often the critical path. Our model assumes the tag memory array circuits are similar to those on the data side with the addition of comparators to choose between different sets.

The comparator that was modeled is shown in Figure 5. 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.

4.5 Set Multiplexor and Output Drivers

In a set-associative cache, the result of the A comparisons must be used to select which of the A possible data blocks are to be sent out of the cache. Since the width of a block ($8B$) is usually greater than the cache output width (b_o), it is also necessary to choose part of the selected block to drive the output lines. An A-way set-associative cache contains A multiplexor driver blocks, as shown in Figure 6. Each multiplexor driver uses a single comparator output bit, along with address bits, to determine which b_o data array outputs drive the output bus.

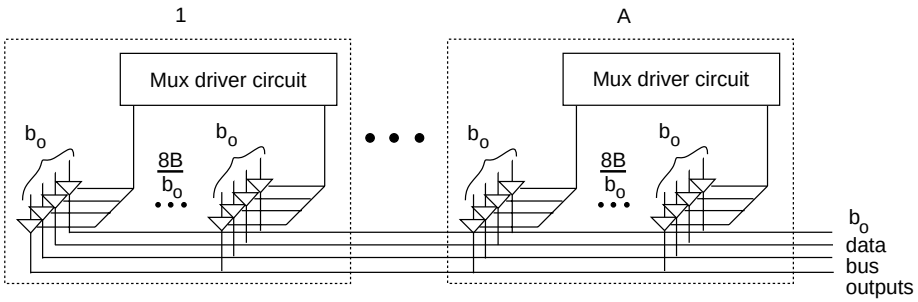


Figure 6: Overview of data bus output driver multiplexors

5 Model Derivation

The delay of each component was estimated by decomposing each component into several equivalent RC circuits, and using simple RC equations to estimate the delay of each stage. This section shows are resistances and capacitances were estimated, as well as how they were combined and the delay of a stage calculated. The stage delay in our model depends on the slope of its inputs; this section also describes how this was done.

5.1 Estimating Resistances

To use the RC approximations described in Sections 5.5 and 5.6, it is necessary to estimate the full-on resistance of a transistor. 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 also be used for pass transistors that (as far as the critical path is concerned) are fully conducting.

It is assumed that the equivalent resistance of a conducting transistor is inversely pro-

portional to the transistor width (only minimum-length transistors were used). Thus,

$$\text{equivalent resistance} = \frac{R}{W}$$

where R is a constant (different for NMOS and PMOS transistors) and W is the transistor width.

5.2 Estimating Gate Capacitances

The RC approximations in Section 5.5 and 5.6 also require an estimation of a transistor's gate and drain capacitances. The gate capacitance of a transistor consists of two parts: the capacitance of the gate itself, 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:

$$\text{gatecap(W)} = W * L_{eff} * C_{gate} + L_{poly} * L_{eff} * C_{polywire}$$

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 different values of C_{gate} are required.

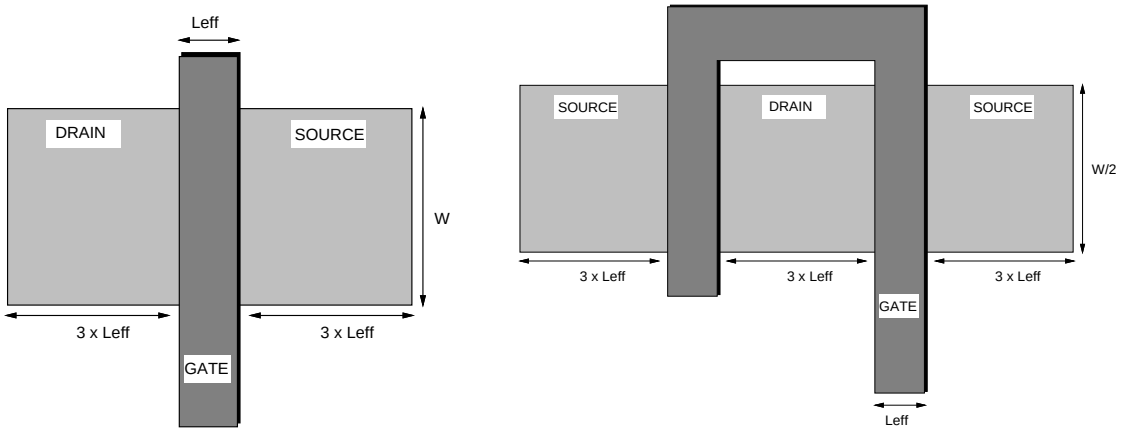
5.3 Drain Capacitances

Figure 7 shows typical transistor layouts for small and large transistors. We have assumed that if the transistor width is larger than $10\mu m$, the transistor is split as shown in Figure 7-b.

The drain capacitance is composed of both an area and perimeter component. Using the geometries in Figure 7, the drain capacitance for a single transistor can be obtained. If the width is less than $10\mu m$,

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

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). $C_{diffgate}$ is the



a) width $< 10\mu m$ b) width $\geq 10\mu m$

Figure 7: Transistor Geometries

sum of the junction capacitance due to the diffusion and the oxide capacitance due to the gate/source or gate/drain overlap.

If the width is larger than $10\mu m$, we assume the transistor is folded (see Figure 7-b), reducing the drain capacitance to:

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

Now, consider two transistors (with widths less than $10\mu m$) connected in series, with only a single $L_{eff} * W$ 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:

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

Figure 8 shows the situation if the transistors are wider than $10\mu m$. 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:

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

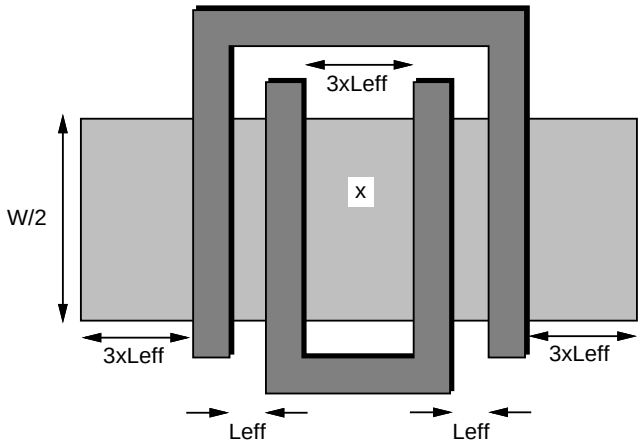


Figure 8: Two stacked transistors if each width $\geq 10\mu m$

5.4 Other parasitic effects

Parasitic resistances and capacitances of the bitlines, wordlines, predecode lines, and various other signals within the cache are also modeled. These resistances and capacitances are fixed values per unit length; the capacitance includes an expected value for the area and sidewall capacitances to the substrate and other layers.

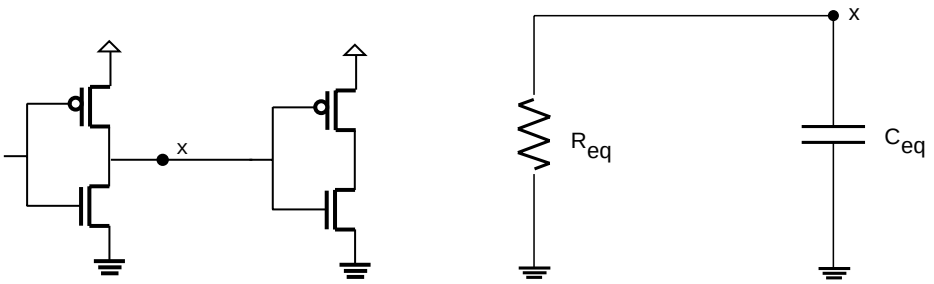
5.5 Simple RC Circuits

Each component described in Section 4 can be decomposed into several first or second order RC circuits. Figure 9-a shows a typical first-order circuit. The time for node x to rise or fall can be determined using the equivalent circuit of Figure 9-b. Here, the pull-down path (assuming a rising input) of the first stage is replaced by a resistance, and the gate capacitances of the second stage and the drain capacitance of the first stage are replaced by a single capacitor. The resistances and capacitances are calculated as shown in Sections 5.1 to 5.3. In stages in which the two gates are separated by a long wire, parasitic capacitances and resistances of the wire are included in C_{eq} and R_{eq}.

The delay of the circuit in Figure 9 can be estimated using an equation due to Horowitz [4] (assuming a rising input):

$$\text{delay} = t_f * \sqrt{\left[\log\left(\frac{v_{th}}{V_{dd}}\right)\right]^2 + 2t_{rise}b\left(1 - \frac{v_{th}}{V_{dd}}\right)/t_f}$$

where v_{th} is the switching voltage of the inverter, t_{rise} is the input rise time, t_f is the output



a) schematic b) equivalent circuit

Figure 9: Example Stage

time constant assuming a step input ($t_f = R_{eq} * C_{eq}$), and b is the fraction of the input swing in which the output changes (we used $b = 0.5$). For a falling input with a fall time of t_{fall}, the above equation becomes:

$$\text{delay} = t_f * \sqrt{\left[\log\left(1 - \frac{v_{th}}{V_{dd}}\right)\right]^2 + \frac{2t_{fall} * b * v_{th}}{t_f * V_{dd}}}$$

In this case, we used $b = 0.4$.

The delay of a gate is defined as the time between the input reaching the switching voltage (threshold voltage) of the gate, and the output reaching the threshold voltage of the following gate. If the gate drives a second 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:

$$\text{delay} = t_f * \sqrt{\left[\log\left(\frac{v_{th1}}{V_{dd}}\right)\right]^2 + 2t_{rise}b\left(1 - \frac{v_{th1}}{V_{dd}}\right)/t_f} + t_f\left[\log\left(\frac{v_{th1}}{V_{dd}}\right) - \log\left(\frac{v_{th2}}{V_{dd}}\right)\right]$$

for a rising input, and

$$\text{delay} = t_f * \sqrt{\left[\log\left(1 - \frac{v_{th1}}{V_{dd}}\right)\right]^2 + \frac{2t_{rise} * b * v_{th1}}{t_f * V_{dd}}} + t_f\left[\log\left(1 - \frac{v_{th1}}{V_{dd}}\right) - \log\left(1 - \frac{v_{th2}}{V_{dd}}\right)\right]$$

for a falling input.

As described in Section 4.2, the size of the wordline driver depends on the number of cells being driven. For a given array width, the capacitance driven by the wordline driver can be estimated by summing the gate capacitance of each pass transistor being driven by the wordline, as well as the metal capacitance of the line. Using this, and the desired rise

time, the required pull-up resistance of the driver can be estimated by:

$$R_p = \frac{-\text{desired rise time}}{C_{eq} * \ln(0.5)}$$

(recall that the desired rise time is assumed to be the time until the wordline reaches 50% of its maximum value).

Once R_p is found, the required transistor width can be found using the equation in Section 5.1. Since this "backwards analysis" did not take into account the non-zero input fall time, we then use R_p and the wordline capacitance and calculate the adjusted delay using Horowitz's equations as described earlier. These transistor widths are also used to estimate the delay of the final gate in the decoder.

5.6 RC-Tree Solutions

All but two of the stages along the cache's critical path can be approximated by simple first-order stages as in the previous section. The bitline and comparator equivalent circuits, however, require more complex solutions. Figure 10 shows an equivalent circuit that can be used for the bitline and comparator circuits. From [5], the delay of this circuit can be written as:

$$T_{step} = [R_2C_2 + (R_1 + R_2)C_1] \ln\left(\frac{v_{start}}{v_{end}}\right)$$

where v_{start} is the voltage at the beginning of the transistion, and v_{end} is the voltage at which the stage is considered to have "switched" ($v_{start} > v_{end}$). For the comparator, v_{start} is V_{dd} and v_{end} is the threshold voltage of the multiplexor driver. For the bitline subcircuit, v_{start} is the precharged voltage of the bitlines (v_{pre}), and v_{end} is the voltage which causes the sense amplifier output to fully switch ($v_{pre} - v_{sense}$).

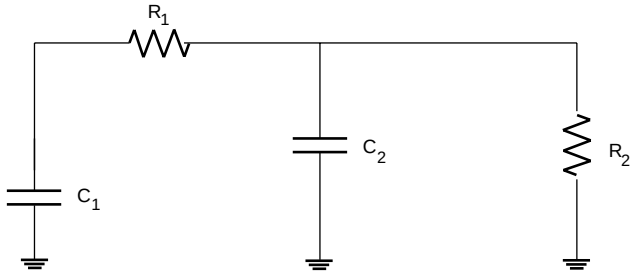


Figure 10: Equivalent circuit for bitline and comparator

When estimating the bitline delay, a non-zero wordline rise time is taken into account as follows. Figure 11 shows the wordline voltage as a function of time for a step input. The time difference T_{step} shown on the graph is the time after the input rises (assuming a step input) until the output reaches $v_{pre} - v_{sense}$ (the output voltage is not shown on the graph). An equation for T_{step} is given above. During this time, we can consider the bitline being "driven" low. Because the current sourced by the access transistor 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 $v_{pre} - v_{sense}$. This area can be calculated as:

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

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

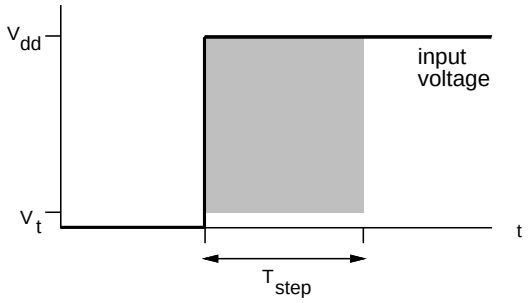
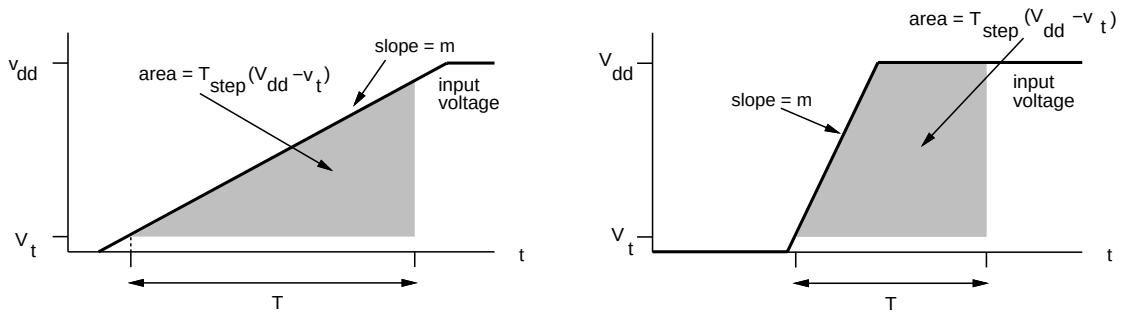


Figure 11: Step input

If we assume that the same amount of "drive" is required to drive the output to $v_{pre} - v_{sense}$ regardless of the shape of the input waveform, then we can calculate the output delay for an arbitrary input waveform. Consider Figure 12-a. If we assume the area is the same as in Figure 11, then we can calculate the value of T (delay adjusted for input rise time). Using simple algebra, it is easy to show that

$$T = \sqrt{\frac{2 * T_{step} * (V_{dd} - V_t)}{m}}$$

where m is the slope of the input waveform (this can be estimated using the delay of the previous stage). Note that the bitline delay is measured from the time the wordline reaches



a) Slow-rising input b) Fast-rising input

Figure 12: Non-zero input rise time

V_t. Unlike the version of the model described in [3], the wordline rise time in this model is defined as the time until the bitlines begin to discharge; this happens when the wordline reaches V_t.

If the wordline rises quickly, as shown in Figure 12-b, then the algebra is slightly different. In this case,

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

The cross-over point between the two cases for T occurs when:

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

The non-zero input rise time of the comparator can be taken into account similarly. The delay of the comparator is composed of two parts: the delay of the timing chain and the delay discharging the output (see Figure 5). The delay of the first three inverters in the timing chain can be approximated using simple first-order RC stages as described in Section 5.5. The time to discharge the comparator output through the final inverter can be estimated using the equivalent circuit of Figure 10 and taking into account the non-zero input rise time using the same technique that was used for the bitline subcircuit. In this case, the "input" is the output of the third inverter in the timing chain (we assume the timing chain is long enough that the a_n and b_n lines are stable). The discharging delay of the comparator output is measured from the time the input reaches the threshold voltage of the final timing chain inverter. The equations for this case can be found in [3].

6 Total Access and Cycle Time

This section describes how delays of the model components described in Section 4 are combined to estimate the cache read access and cycle times.

6.1 Access Time

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 longer to read the data array, then the data side is the critical path. In many cache implementations, the designer would try to margin the cache design such that the tag path is slightly faster than the data path so that the multiplexor select signals are valid by the time the data is ready. Often, however, this is not possible. Therefore, either side could determine the access time, meaning both sides must be modeled in detail.

In a direct-mapped cache, the access time is the larger of the two paths:

$$T_{access\_dm} = \max(T_{dataside} + T_{outdrive\_data}, T_{tagside\_dm} + T_{outdrive\_valid})$$

where $T_{dataside}$ is the delay of the decoder, wordline, bitline, and sense amplifier for the data array, $T_{tagside\_dm}$ is the delay of the decoder, wordline, bitline, sense amplifier, and comparator for the tag array, $T_{outdrive\_dm}$ is the delay of the cache data output driver, and $T_{outdrive\_valid}$ is the delay of the valid signal driver.

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

$$T_{access,sa} = \max(T_{dataside}, T_{tagside\_sa}) + T_{outdrive\_data}$$

where $T_{tagside\_sa}$ is the same as $T_{tagside\_dm}$, except that it includes the time to drive the select lines of the output multiplexors.

Figures 13 to 16 show analytical and Hspice estimations of the data and tag sides for direct-mapped and 4-way set-associative caches. A $0.8\mu m$ CMOS process was assumed [6]. To gather these results, the model was first used to find the array organization parameters which resulted in the lowest access time via exhaustive search for each cache size. These

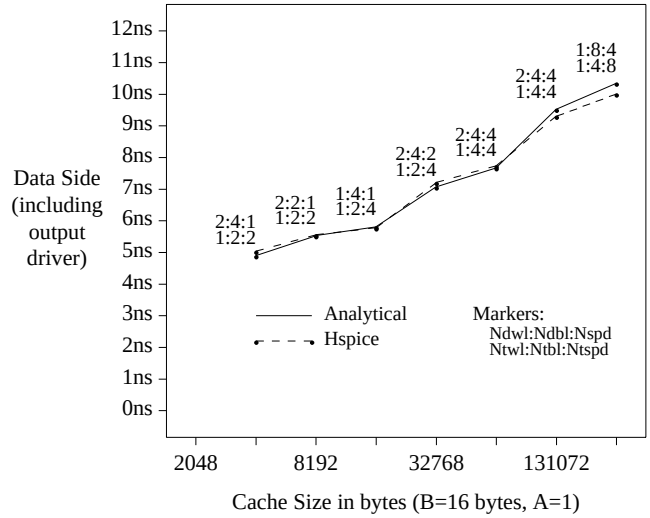


Figure 13: Direct mapped: $T_{dataside} + T_{outdrive\_data}$

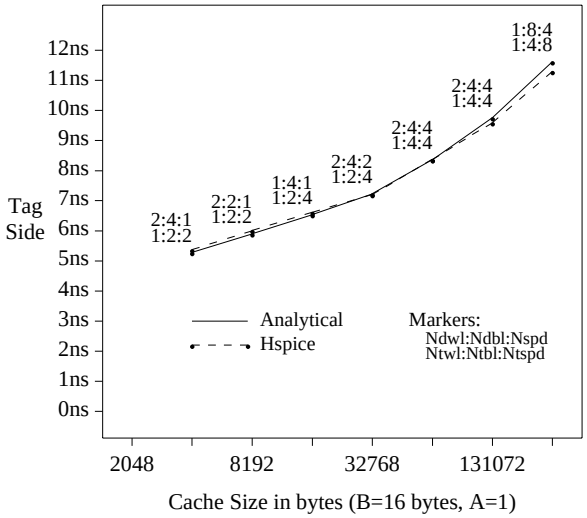


Figure 14: Direct mapped: $T_{tagside\_dm}$

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 6% in every case.

6.2 Cycle Time

The difference between the access and cycle time of a cache varies 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 [7, 8] the cycle time can be less than

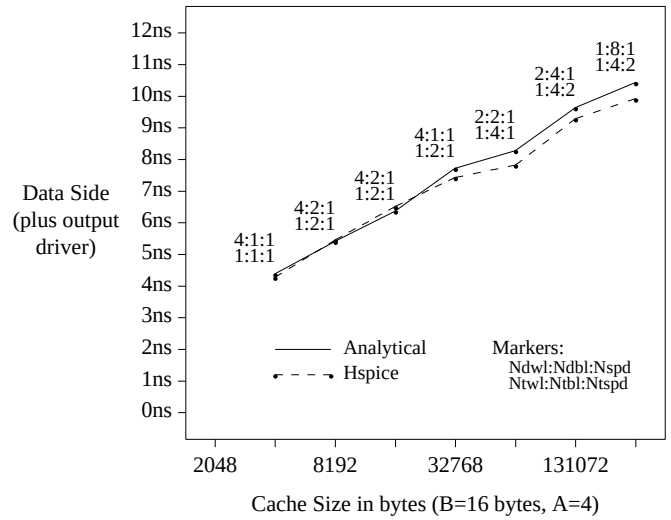


Figure 15: 4-way set associative: $T_{dataside} + T_{outdrive\_data}$

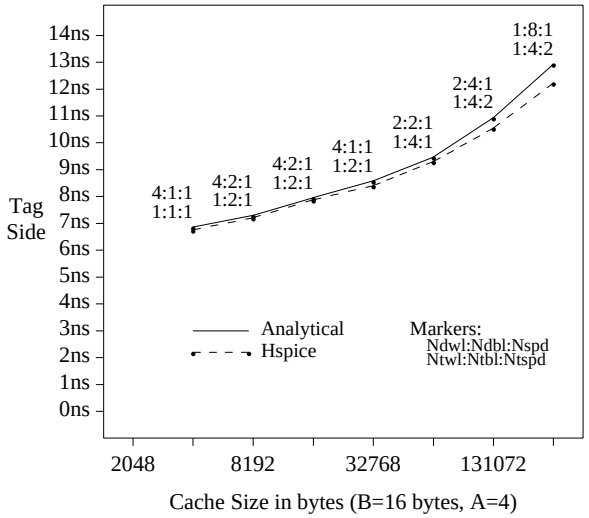


Figure 16: 4-way set associative: $T_{tagside,sa}$

the access time. We have chosen to model a more conventional structure with the cycle time equal to the access time plus the precharge.

There are three elements in our assumed cache organization that need to be precharged: the decoders, the bitlines, and the comparator. The precharge times for these elements are somewhat arbitrary, since the precharging transistors can be scaled in proportion to the loads they are driving. We have assumed that the time for the wordline to fall and bitline to rise in the data array is the dominant part of the precharge delay. Assuming properly ratioed transistors in the wordline drivers, the wordline fall time is approximately the same as the wordline rise time. It is assumed that the bitline precharging transistors are

scaled such that a constant (over all cache organizations) 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). Thus, the cycle time of the cache can be written as:

$$T_{cache} = T_{access} + T_{wordline\ delay} + 4 * (\text{inverter delay})$$

7 Applications of the Model

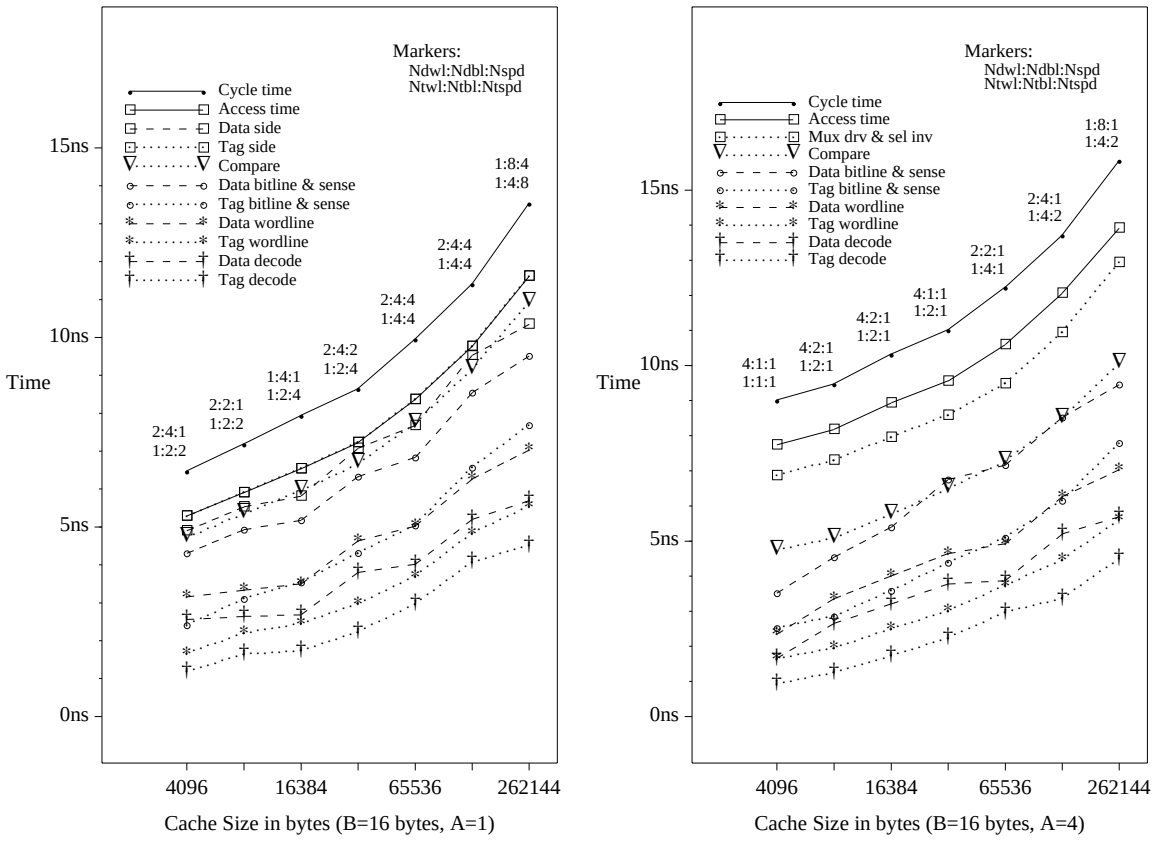
This section gives examples of how the analytical model can be used to quickly gather data that can be used in architectural studies.

7.1 Cache Size

First consider Figure 17. 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 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.

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 but one of the organizations selected, there are more data subarrays ($N_{dwl} * N_{dbl}$) than tag subarrays ($N_{twl} * N_{tbl}$). This is because the total tag storage is usually much less than the total data storage.

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 driving select lines of the output multiplexor.



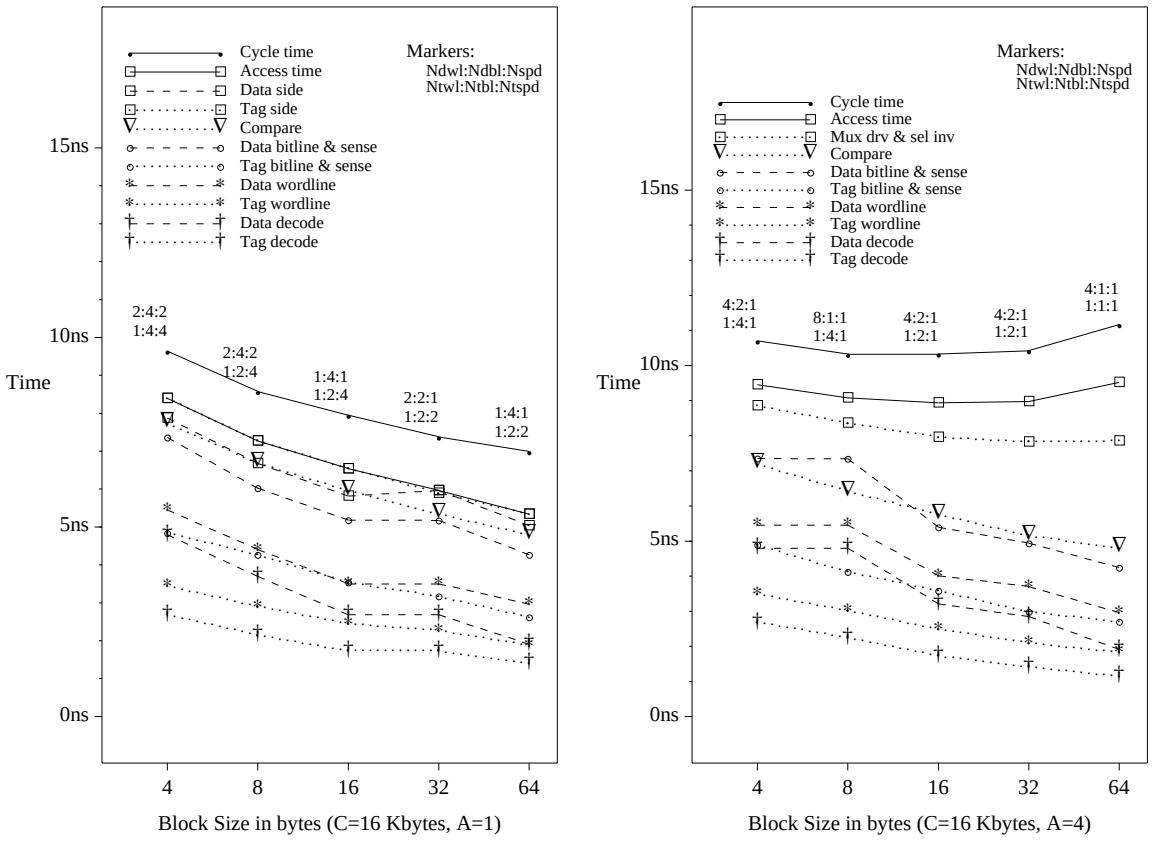
a) direct-mapped b) set-associative

Figure 17: Access/cycle time as a function of cache size

7.2 Block Size

Figure 18 shows 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).

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 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



a) direct-mapped b) set-associative

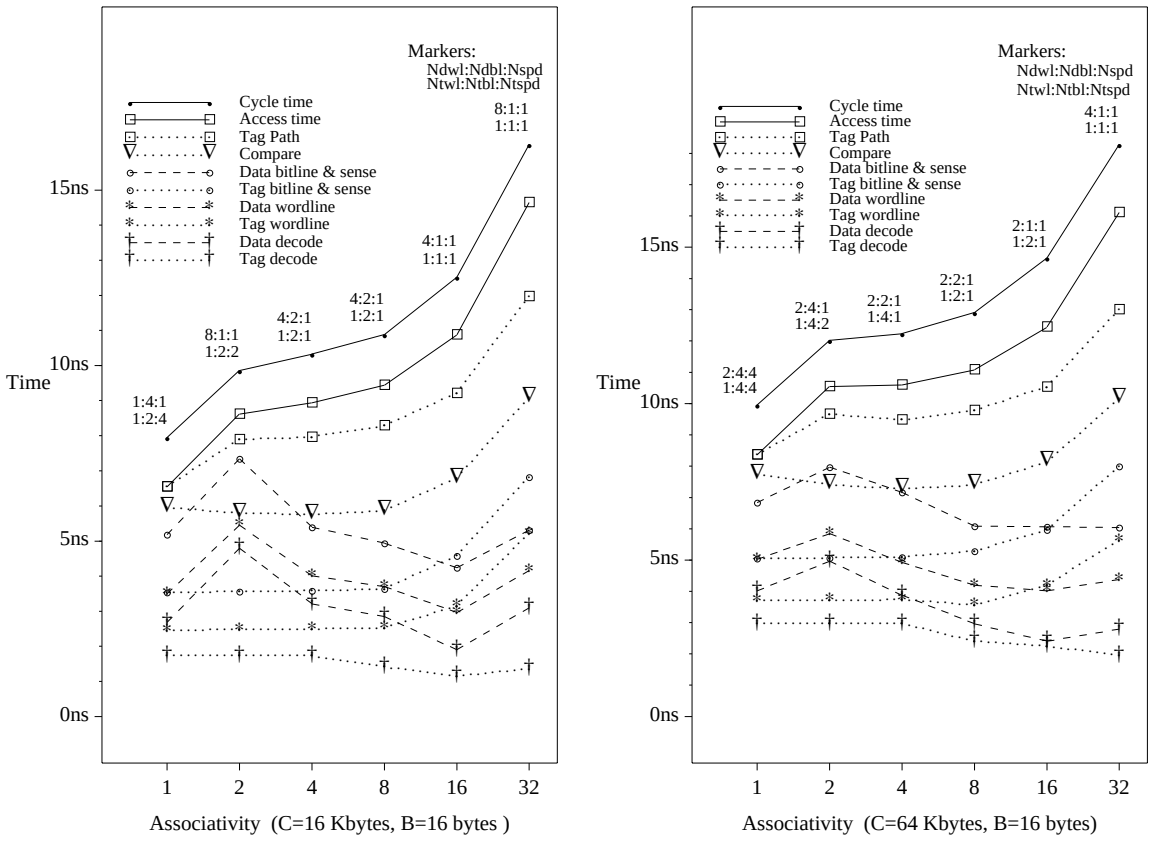
Figure 18: Access/cycle time as a function of block size

the access time in all the organizations shown.

7.3 Associativity

Finally, consider Figure 19 which shows 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.

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



a) 16KB cache b) 64KB cache

Figure 19: Access/cycle time as a function of associativity

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.

8 Conclusions

In this paper, we have presented an analytical model for the access and cycle time of a cache. By comparing the model to an Hspice model, CACTI was shown to be accurate to within 6%. The computational complexity, however, is considerably less than Hspice.

Although previous models have also given results close to Hspice predictions, the underlying assumptions in previous models have been very different from typical on-chip cache memories. Our extended and improved model fixes many major shortcomings of previous

models. Our model includes the tag array, comparator, and multiplexor drivers, non-step stage input slopes, rectangular stacking of memory subarrays, a transistor-level decoder model, column-multiplexed bitlines, an additional array organizational parameter and load-dependent transistor sizes for wordline drivers. It also produces cycle times as well as access times. This makes the model much closer to real memories.

It is dangerous to make too many conclusions directly from the graphs without considering miss rate data. Figure 19 seems 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 [9], 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.

Appendix A: Obtaining and Using the CACTI Software

A program that implements the CACTI model described in this paper 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 3) and choose the organization that gives the smallest access time. The values of the array organization parameters chosen are included in the output report.

The extended description of model details is available on the World Wide Web at URL "http://nsl.pa.dec.com/wrl/techreports/93.5.html#93.5".

Acknowledgements

The authors wish to thank Stefanos Sidiropoulos, Russell Kao, and Ken Schultz for their helpful comments on various drafts of the manuscript. Financial support was provided by the Natural Sciences and Engineering Research Council of Canada and the Walter C. Sumner Memorial Foundation.

References

[1] J. M. Mulder, N. T. Quach, and M. J. Flynn, "An area model for on-chip memories and its application," *IEEE Journal of Solid-State Circuits, Vol. 26, No. 2*, pp. 98–106, Feb. 1991.

[2] T. Wada, S. Rajan, and S. A. Przybylski, "An analytical access time model for on-chip cache memories," *IEEE Journal of Solid-State Circuits, Vol. 27, No. 8*, pp. 1147–1156, Aug. 1992.

[3] S. J. Wilton and N. P. Jouppi, "An access and cycle time model for on-chip caches," Tech. Rep. 93/5, Digital Equipment Corporation Western Research Lab, 1994.

[4] M. A. Horowitz, "Timing models for mos circuits," Tech. Rep. Technical Report SEL83-003, Integrated Circuits Laboratory, Stanford University, 1983.

[5] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal delay in RC tree networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 2, No. 3*, pp. 202–211, July 1983.

[6] M. G. Johnson and N. P. Jouppi, "Transistor model for a synthetic 0.8um CMOS process," *Class notes for Stanford University EE371*, Spring 1990.

[7] R. J. Proebsting, "Post charge logic permits 4ns 18K CMOS RAM." 1987.

[8] T. I. Chappell, B. A. Chappel, S. E. Schuster, J. W. Allen, S. P. Klepner, R. V. Joshi, and R. L. Franch, "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*, pp. 1577–1585, Nov. 1991.

[9] N. P. Jouppi and S. J. Wilton, "Tradeoffs in two-level on-chip caching," in *Proceedings of the 21th Annual International Symposium on Computer Architecture*, pp. 34–45, April 1994.