AUGUST 2001

WRL Research Report 2001/2

CACTI 3.0: An Integrated Cache Timing, Power, and Area Model

Premkishore Shivakumar and Norman P. Jouppi

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

The Western Research Laboratory (WRL), located in Palo Alto, California, is part of Compaq's Corporate Research group. WRL was founded by Digital Equipment Corporation in 1982. We focus on information technology that is relevant to the technical strategy of the Corporation, and that has the potential to open new business opportunities. Research at WRL includes Internet protocol design and implementation, tools to optimize compiled binary code files, hardware and software mechanisms to support scalable shared memory, graphics VLSI ICs, handheld computing, and more. As part of WRL tradition, we test our ideas by extensive software or hardware prototyping.

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, conference papers, or magazine articles. We use technical notes for rapid distribution of technical material; usually this represents research in progress.

You can retrieve research reports and technical notes via the World Wide Web at:

http://www.research.compaq.com/wrl/

You can request printed copies of research reports and technical notes, when available, by mailing your order to us at:

Technical Report Distribution
Compaq Western Research Laboratory
250 University Avenue
Palo Alto, California 94301 U.S.A.

You can also request reports and notes by sending e-mail to:

WRL-techreports@research.compaq.com

CACTI 3.0: An Integrated Cache Timing, Power, and Area Model

Premkishore Shivakumar and Norman P. Jouppi

Compaq Computer Corporation Western Research Laboratory
pkishore@cs.utexas.edu, norm.jouppi@compaq.com

August, 2001

Abstract

CACTI 3.0 is an integrated cache access time, cycle time, area, aspect ratio, and power model. By integrating all these models together users can have confidence that tradeoffs between time, power, and area are all based on the same assumptions and hence are mutually consistent. CACTI is intended for use by computer architects so they can better understand the performance tradeoffs inherent in different cache sizes and organizations.

This report details enhancements to CACTI 2.0 that are included in CACTI 3.0. CACTI 3.0 includes modeling support for the area and aspect ratio of caches, caches with independently addressed banks, reduced sense-amp power dissipation, and other improvements to CACTI 2.0.

Contents

1 Introduction **1**

1.1 Capabilities and Limitations of the Previous Version 1

1.2 New Features in CACTI 3.0 1

2 The CACTI 3.0 Area Model **2**

2.1 The Set-Associative and Direct-Mapped Cache Area Models 3

2.1.1 Data Array 3

2.1.2 Tag Array 6

2.1.3 Data Subarray Area Calculation 8

2.1.4 Tag Subarray Area Calculation 9

2.2 The Fully-Associative Cache Area Model 10

2.2.1 The Fully-Associative Subarray - Data and Tag Array Together 11

2.2.2 Fully-Associative Subarray 12

2.3 Area Estimation of Individual Circuits 13

3 Power Reduction in the Sense Amplifier **14**

4 Cache Banking Model **14**

4.1 Banking Area Model 15

4.2 Banking Time Model 15

4.3 Banking Power Model 16

5 Integration of All the Models **17**

6 CACTI 3.0 Results **18**

6.1 Area Model Results 18

6.2 Time and Power Model Results 21

7 Area Model Validation **24**

8 Future Work **26**

A CACTI Syntax **28**

B CACTI Output **29**

C Meaning of the Terms Used in the Equations and Text **32**

List of Figures

1 Basic Set-Associative Cache Structure 2

2 Subarray Placement: Power of 4, 2*Power of 4 4

3 2x2 Subarray Organization 5

4 Htree Routing Structure 5

5 2x2 Subarray Tag Structure 6

6 Data Subarray Dissection 9

7 Tag Subarray Dissection 10

8 Fully-Associative Cache Structure 11

9 2x2 Fully-Associative Subarray Structure 12

10 Fully-Associative Subarray Dissection 13

11 Control Logic For Turning the Foot Transistors of the Sense Amplifier ON 14

12 Bank Placement 16

13 **Area Model Sensitivity to Cache Size and Associativity.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different associativities. 19

14 **Area Model Sensitivity to Size and Number of Ports.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different number of ports. 20

15 **Area Model Sensitivity to Size and Banking.** The Y-axis is on a linear scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different number of banks. 20

16 **Area Model Sensitivity to Size, Associativity, and Ports.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different number of banks. The different graphs are for different numbers of ports. 21

17 **Power Model Sensitivity to Size, Associativity, and Ports.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different associativities. The different graphs are for different numbers of ports. 22

18 **Time Model Sensitivity to Size and Associativity.**. The y-axis is on a log scale. The x-axis ranges over a variety of cache configurations. Each cache size is given with different associativities. 22

19 **Time Model Sensitivity to Size, Associativity, and Banks.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is broken down into different number of banks. The different graphs are for the direct mapped and fully associative caches. 23

20 **Time Model Sensitivity to Size and Banks.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different number of banks. 24

21 **Power Model Sensitivity to Cache Size, Associativity, and Banking.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different associativities. The different graphs are for different numbers of banks. 25

List of Tables

1 Cache configurations used in the validation of the area model. 25

2 Results of comparison of actual cache areas to CACTI 3.0 model results. 26

CACTI 3.0: An Integrated Cache Timing, Power, and Area Model

Premkishore Shivakumar - pkishore@cs.utexas.edu
Norman P. Jouppi - norm.jouppi@compaq.com

1 Introduction

CACTI[1] was originally developed to aid computer architects in quantifying the access and cycle time tradeoffs between different on-chip direct-mapped and set-associative cache configurations.

1.1 Capabilities and Limitations of the Previous Version

The first version of CACTI was extended to support fully-associative caches, multiported caches, feature size scaling, and power modeling in CACTI 2.0[2]. It takes as input the cache capacity, associativity, cacheline size, the number of read/write ports and the feature size. It uses analytical models to compute the access time of the cache and the energy consumed by it for different configurations (each dividing the data and tag array into some number of subarrays) and returns the configuration that has the best access time and energy consumption as determined by its optimization function.

The CACTI 2.0 optimizing function attaches greater importance to the access time of the cache and uses the energy consumption as a secondary criteria. CACTI 2.0 does not have a conception of the total area or the efficiency (percentage of area occupied by the bits alone) of each configuration. It tries to account for the wire capacitance and resistance that is associated with the wires driving the address bits to the predecoder, from the predecoder to the NOR gates, from the output driver to the edge of the data array, and from the mux drivers to the data output drivers, but since it does not have an area model these distances are approximate. Also since the optimizing function takes only the access time and the energy consumption of the cache into account, the final cache configuration may have very low area efficiency or an undesirably large aspect ratio.

1.2 New Features in CACTI 3.0

CACTI 3.0 begins by adding adding a detailed cache area model to CACTI 2.0. CACTI now calculates the area occupied by each component of the cache for each configuration that it iterates through. This also provides us with the efficiency of that particular configuration and the aspect ratio of the whole cache. We have altered the optimizing function so that it now decides the best configuration taking access time, power consumption, efficiency of the layout and aspect ratio all into account.

The detailed area model also provides us with the capability of accurately calculating the wire-lengths (and hence the associated capacitance and resistance) of the address and data routing

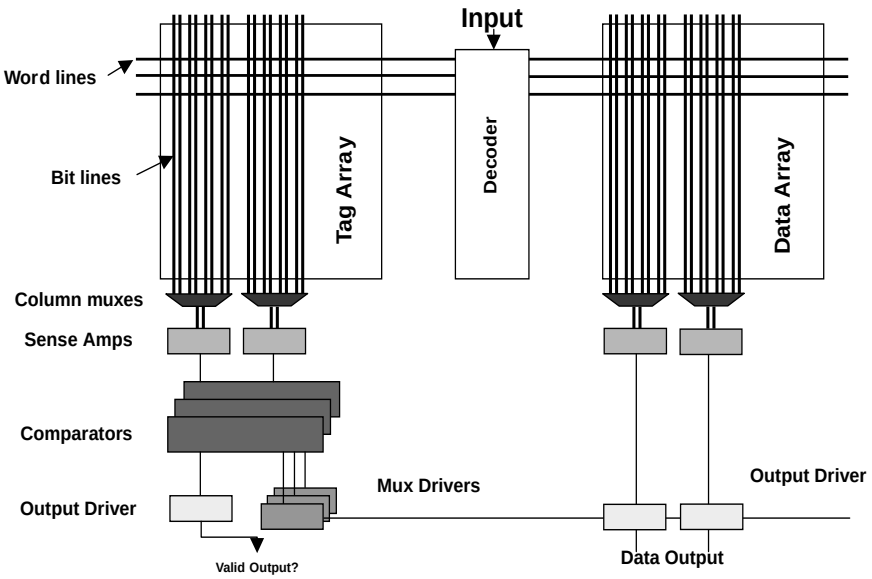

Figure 1: Basic Set-Associative Cache Structure

tracks. We have made appropriate changes to the time and power model of CACTI 2.0 to incorporate these changes. These enhancements are described in Section 2.

It was also observed that the sense amplifier in the earlier version of CACTI consumed a lot of static power when the bitlines are in the precharged state. We have tried to reduce this by enabling the sense amplifier for a much shorter duration. This results in more realistic power estimates. A detailed explanation of this improvement is given in Section 3.

We have also added support for fully-independent banking of caches. This allows the user to split the cache into a number of banks each with its own set of address and data buses. A very simple model for banking caches has also been implemented in the time, power and area models. This work is described in Section 4.

In Section 5 we describe the new CACTI 3.0 optimization function. Section 6 gives the cache area computed by the area model as a function of various cache parameters. These results are compared with data from microprocessor designs in Section 7. Section 8 gives attractive areas for further enchancements to CACTI.

For those unfamiliar with previous versions of CACTI, we suggested that the reader refer to the Appendix C at the end of the report to familiarize themselves with the meanings of the abbreviations and the terminology used in the equations and the text.

2 The CACTI 3.0 Area Model

The goal of the area model is to be able to provide CACTI with the area of each configuration of the cache that it evaluates. It has to account for the area of the decoders, bit cells, input and

output circuitry and the routing tracks. The transistor widths of individual circuits of the cache have already been tuned to achieve the best access time for each configuration. The area model uses the same transistor widths as the time and the power models. We will use a hierarchical top-down approach to describe the area model and we will deal with set-associative and fully-associative caches separately.

2.1 The Set-Associative and Direct-Mapped Cache Area Models

The approach that CACTI takes in arriving at the best configuration is by iterating through all possible allowed configurations *((Ndwl, Ndbl, Nspd), (Ntwl, Ntbl, Ntspd))*, calculating the time, power, and area of each and selecting the best based on an optimization function. Each configuration which hereafter means a particular ordered set of *((Ndwl, Ndbl, Nspd), (Ntwl, Ntbl, Ntspd))* translates into a certain number of subarrays, each with a certain number of rows and columns. The data array and tag array for a set-associative cache are of very different sizes, and so we have taken the approach of implementing them as separate arrays.

2.1.1 Data Array

Since we are taking a top-down approach we will start with the placement of the subarrays and the global wiring tracks.

The number of subarrays is always a power of two. If the number of subarrays is an exact power of 4 then they are divided into 4 blocks and placed in the form of a square. Each block is in turn divided into 4 blocks internally and this process is recursively done until we arrive at the terminal state where we have to place 4 subarrays in the form of a square. If the number of subarrays is 2 times a power of 4 then it is divided into 2 blocks each, which is now an exact power of 4. Then the process for "exact powers of 4" is carried out until we arrive at the whole placement for the 2 blocks internally. Now the 2 blocks are placed side by side or on top of each other, depending on which gives the better aspect ratio.

The global tracks are the address lines to the predecoder (which is placed at the center of the subarrays), the predecode lines from the predecoder going to the NOR gates (that are shared by 2 subarrays), the select line to the column multiplexer, the select lines to the output driver for the read operation, and the select lines for writing the memory cell. The data bits also have to be routed, but the number of tracks that are routed vary as we go deeper into the array. Figures 3 and 4 show the tracks for a single port. If we have multiple ports these tracks get multiplied by the appropriate number of ports.

$$Data\ Tracks = BITOUT\ [RWP + ERP + EWP]\ Widthtrack$$

$$Output\ Driver\ and\ Write\ Select\ Tracks = [2RWP + ERP + EWP]\frac{8BA}{BITOUT}Widthtrack$$

$$Column\ Multiplexing\ Tracks = NdblNspd\,[RWP + EWP + ERP]Widthtrack$$

$$Predecode\ Tracks = [RWP + ERP + EWP]8 \cdot N3to8 \cdot Widthtrack$$

$$Address\ Tracks = AddressBits\,[RWP + ERP + EWP]Widthtrack$$

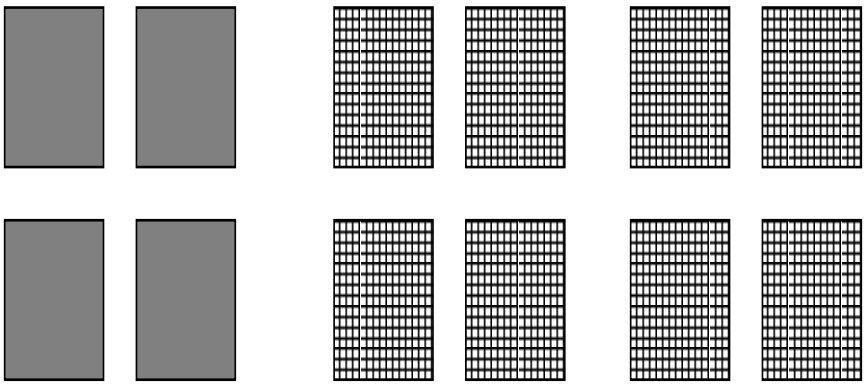

Figure 2: Subarray Placement: Power of 4, 2*Power of 4

$$\begin{aligned} Fixed\ Internal\ Tracks &= Column\ Multiplexing\ Tracks \\ &\quad + Predecode\ Tracks + Output\ Driver\ and\ Write\ Select\ Tracks \end{aligned}$$

$$\begin{aligned} Fixed\ External\ Tracks &= Column\ Multiplexing\ Tracks \\ &\quad + Address\ Tracks + Output\ Driver\ and\ Write\ Select\ Tracks \end{aligned}$$

$$Variable\ Tracks = Data\ Tracks$$

Let us assume for now that we know the subarray height and width for each cache configuration. Then as we already know the exact method of placement of the subarrays and the exact number of routing tracks (and the wire pitch), we can calculate the height and width of the entire data array. The area (height,width) of the predecoder is determined with the help of a layout. Sample expressions for the height and the width of a data array with 8 subarrays is given below.

$$\begin{aligned} Data\ Array\ Height &= 2\,[DataSubblock \cdot height] \\ &\quad + Fixed\ Internal\ Tracks + \frac{Variable\ Tracks}{2} \end{aligned}$$

$$\begin{aligned} Data\ Array\ Width &= 2\left[2\,[DataSubblock \cdot width] + \frac{Variable\ Tracks}{4}\right] \\ &\quad + Fixed\ External\ Tracks + Variable\ Tracks \end{aligned}$$

The time taken to drive the address lines to the predecoder is calculated accurately now since we have the knowledge of the exact length between the predecoder and the edge of the data array.

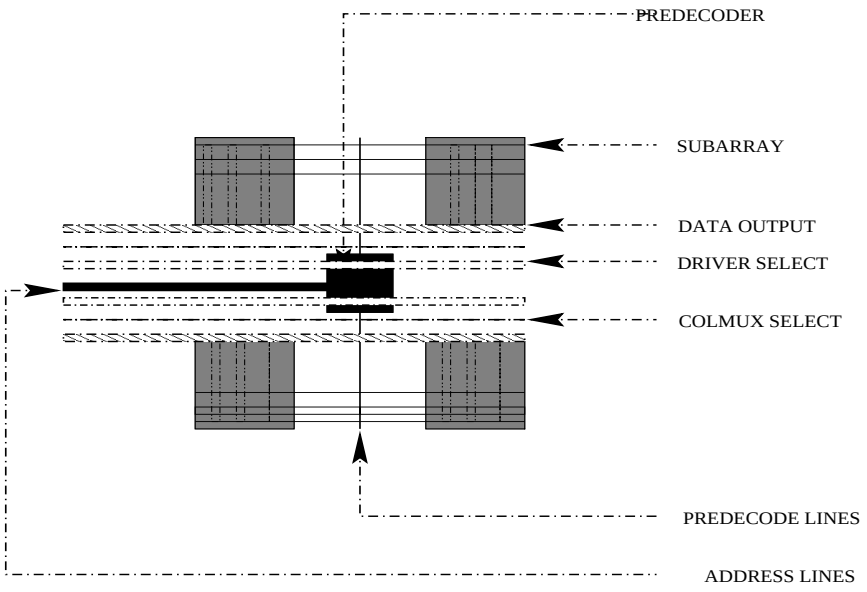

Figure 3: 2x2 Subarray Organization

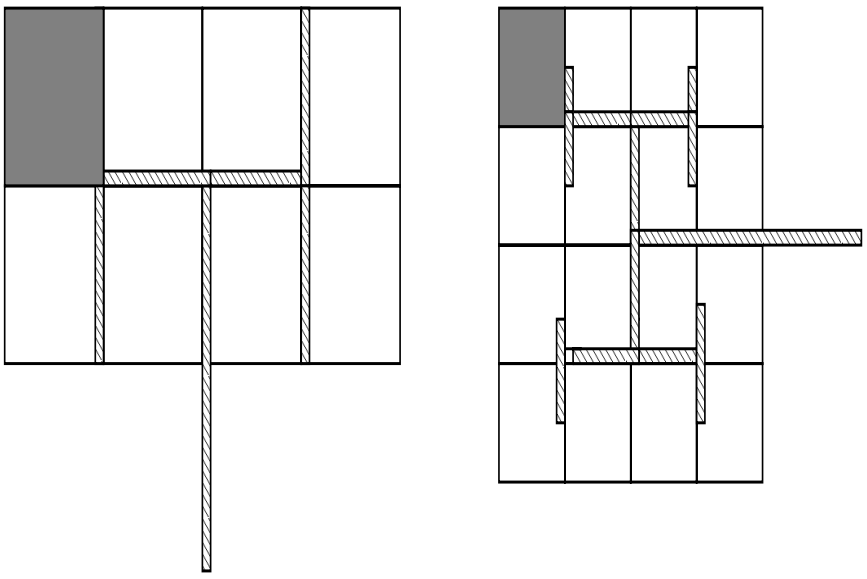

Figure 4: Htree Routing Structure

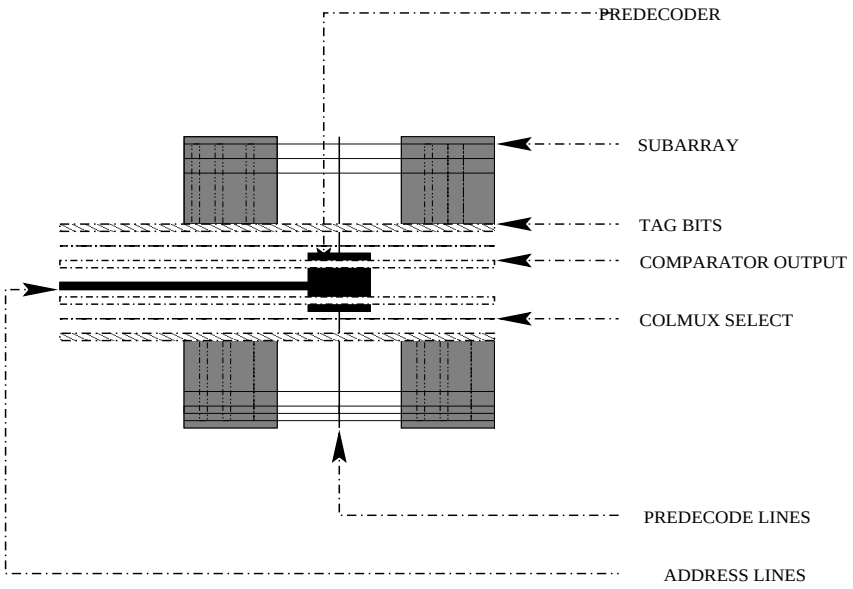

Figure 5: 2x2 Subarray Tag Structure

The predecode lines travel in a H-tree fashion to the subarrays and their length is also known. So the decoding time (in the time model) is modified to take into account these exact lengths. Also the wire capacitance that is calculated from this length directly goes into the power calculation. The same holds true for the data output time also. The output driver has to drive the data (in a H-tree) to the edge of the data array. The wire capacitance and resistance is calculated for this exact length and inserted in the time model to account for the time and the energy consumed.

$$Cwire = Cwordmetal \cdot horizontal\ dim + Cbitmetal \cdot vertical\ dim$$

$$Rwire = \frac{Rwordmetal \cdot horizontal\ dim}{2} + \frac{Rbitmetal \cdot vertical\ dim}{2}$$

2.1.2 Tag Array

We will again start with the placement of the subarrays and the global wiring tracks. The placement of the subarrays follow exactly the same procedure as the data array. The global tracks are the address lines to the predecoder (which is present at the center of the subarray placement), the predecode lines from the predecoder going to the NOR gates (that are shared by 2 subarrays), the select line for the column multiplexer, the outputs of the comparators which effectively say whether the data is present in the cache or not (they are 'associativity' in number), and the select lines for writing the tag bits. The tag bits have also to be routed, but the number of tracks that are routed vary as we go deeper into the array. Figure 5 shows the tracks for a single port. If we have multiple ports, these tracks get multiplied by the appropriate number of ports.

$$Tag\ Tracks = Tagbits\ [RWP + ERP + EWP]\ Widthtrack$$

$$Comparator\ Tracks = [RWP + ERP]\ \ A\ Widthtrack$$

$$Column\ Multiplexing\ Tracks = NdblNspd\,[RWP + EWP + ERP]\,Widthtrack$$

$$Predecode\ Tracks = [RWP + ERP + EWP]\,8 \cdot N3to8 \cdot Widthtrack$$

$$Address\ Tracks = AddressBits\,[RWP + ERP + EWP]\,Widthtrack$$

$$\begin{aligned} Fixed\ Internal\ Tracks &= Column\ Multiplexing\ Tracks \\ &\quad + Predecode\ Tracks + Comparator\ Tracks \end{aligned}$$

$$\begin{aligned} Fixed\ External\ Tracks &= Column\ Multiplexing\ Tracks \\ &\quad + Address\ Tracks + Comparator\ Tracks \end{aligned}$$

$$Variable\ Tracks = Tag\ Tracks$$

Let us assume for now that we know the tag subarray height and width for each cache configuration. Then as we know the exact method of placement of the subarrays and the exact number of routing tracks (and the wire pitch) we can calculate the height and width of the entire tag array. Sample expressions for the height and the width of a tag array with 8 subarrays is given below.

$$\begin{aligned} Tag\ Array\ Height &= 2\,[TagSubblock \cdot height] \\ &\quad + Fixed\ Internal\ Tracks + \frac{Variable\ Tracks}{2} \end{aligned}$$

$$\begin{aligned} Tag\ Array\ Width &= 2\left[2\,[TagSubblock \cdot width] + \frac{Variable\ Tracks}{4}\right] \\ &\quad + Fixed\ External\ Tracks + Tag\ Tracks \end{aligned}$$

The decoding time (in the time model) has been modified to take into account these exact lengths in the same way as for the data array. Also the additional wire capacitance that is calculated from this length is taken into account for the power calculation.

In a set-associative cache the tag array has to provide the data array with the information that tells it which block it has to drive to the output lines. Also the output bus size may be smaller than the cache line size, in which case only a part of the block is driven on to the output bus. These select lines have to be driven from the edge of the data array to each subarray. This time is also incorporated into the multiplexer driver time in the tag side in the time model.

$$Cwire = Cwordmetal \cdot horizontal\ dim + Cbitmetal \cdot vertical\ dim$$

$$Rwire = \frac{Rwordmetal \cdot horizontal\ dim}{2} + \frac{Rbitmetal \cdot vertical\ dim}{2}$$

2.1.3 Data Subarray Area Calculation

We will analyze the width and the height of the subarray separately. The bit cells alone form an array to which we abut the decode NOR gates to add to the width of the subarray, and all the I/O circuitry which add to the height of the subarray. The number of columns of the subarray is a function of the cache configuration. In particular the number of columns in the data subarray is a function of *Nspd* and *Ndwl*. The width of the bit cells alone is a function of the number of the columns and the number of ports per each SRAM cell. The extra width to the data subarray is contributed by the decode NOR gates. There are as many of them as there are ports per bit cell. The number of rows in the subarray is a function of *Ndbl* and *Nspd*. The height of each bit cell is again a function of the number of ports per memory cell.

$$No \cdot ofRows = \left[\frac{C}{[B\ A\ Ndbl\ Nspd]} \right]$$

$$No \cdot ofCols = \left[8\ B\ A\ \frac{Nspd}{Ndwl} \right]$$

$$\begin{aligned} Bit\ Cell\ Height &= Single\ Port\ Bit\ Cell\ Height \\ &\quad + 2\,[RWP + ERP + EWP - 1]\ Widthtrack \end{aligned}$$

$$\begin{aligned} Bit\ Cell\ Width &= Single\ Port\ Bit\ Cell\ Width \\ &\quad + \ [2\,[RWP + ERP - NSER + EWP - 1] + NSER]\ Widthtrack \end{aligned}$$

The I/O circuitry adds to the height of the subarray. The bits from the array are multiplexed to produce the actual output. Depending on how much multiplexing there is we can fold transistors in the I/O circuitry (the sense amplifier etc.,) to reduce the height of the subarray. In the worst case we have no column multiplexing and the sense amplifier, data output driver etc., have to pitch match the width of the bit cell. If the memory cell has multiple ports then there are multiple sense amplifiers, output drivers, etc., and their individual numbers depend on the number of exclusive read ports, exclusive write ports, and read/write ports.

$$\begin{aligned} Data\ Subarray\ Height &= Bit\ cells\ array \cdot height \\ &\quad + Column\ Multiplexer \cdot height\ [RWP + ERP + EWP] \\ &\quad + Precharge \cdot height\,[RWP + EWP] \\ &\quad + Sense\ Amplifier \cdot height\,[RWP + ERP] \\ &\quad + Output\ Driver \cdot height\,[RWP + ERP] \end{aligned}$$

$$\begin{aligned} Data\ Subarray\ Width &= Bit\ cells\ array \cdot width \\ &\quad + Post\ Decode \cdot width\ [RWP + ERP + EWP] \end{aligned}$$

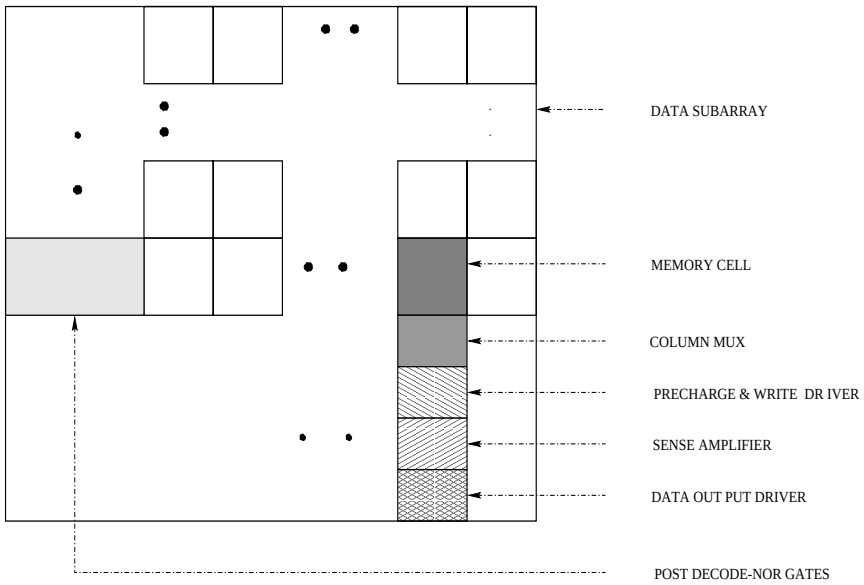

Figure 6: Data Subarray Dissection

2.1.4 Tag Subarray Area Calculation

The number of tag bits is a function of the address bits, cache capacity, and associativity. The number of columns in the tag subarray is a function of Ntspd and Ntwl. The width of the bit cells alone is a function of the number of the columns and the number of ports per each SRAM cell. In the data array the read only ports were single ended in order to reduce the width of the bit cell. The same optimization is not implemented here as the comparator circuit (XOR gate) needs both the bit and bit complement to produce its result. The extra width to the tag subarray is again contributed by the decode NOR gates. There are as many of them as there are ports per bit cell. The number of rows in the subarray is a function of *Ntbl* and *Ntspd*. The height of each bit cell is again a function of the number of ports per memory cell.

$$No \cdot ofRows = \left\lceil \frac{C}{[B\ A\ Ntbl\ Ntspd]} \right\rceil$$

$$No \cdot ofCols = \left\lceil Tagbits\ A\ \frac{Ntspd}{Ntwl} \right\rceil$$

$$\begin{aligned} Tag\ Bit\ Cell\ Height &= Single\ Port\ Bit\ Cell\ Height \\ &\quad + 2\left[RWP + ERP + EWP - 1\right]\ Widthtrack \end{aligned}$$

$$\begin{aligned} Tag\ Bit\ Cell\ Width &= Single\ Port\ Bit\ Cell\ Width \\ &\quad + 2\left[RWP + ERP + EWP - 1\right]\ Widthtrack \end{aligned}$$

The I/O circuitry adds to the height of the subarray. The bits from the array are multiplexed to produce the actual output. Depending on how much multiplexing there is, we can fold transistors

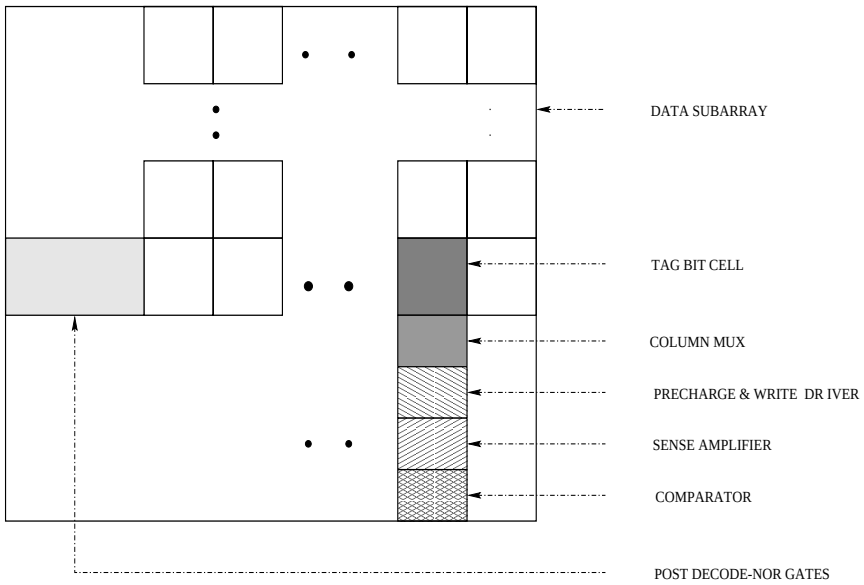

Figure 7: Tag Subarray Dissection

in the I/O circuitry (the sense amplifier, etc.,) to reduce the height of the subarray. In the worst case again we have no column multiplexing when the sense amplifier, comparators, etc., have to pitch match the width of the bit cell. If the memory cell has multiple ports then there are multiple sense amplifiers, comparators, etc., depending on the number of exclusive read ports, exclusive write ports, and read/write ports.

$$\begin{aligned} Tag\ Subarray\ Height &= Tag\ Bit\ cells\ array \cdot height \\ &+ Column\ Multiplexer \cdot height\ [RWP + ERP + EWP] \\ &+ Precharge \cdot height\,[RWP + EWP] \\ &+ Sense\ Amplifier \cdot height\,[RWP + ERP] \\ &+ Comparator \cdot height\,[RWP + ERP] \end{aligned}$$

$$\begin{aligned} Tag\ Subarray\ Width &= Tag\ Bit\ cells\ array \cdot width \\ &+ Post\ Decode \cdot width\ [RWP + ERP + EWP] \end{aligned}$$

2.2 The Fully-Associative Cache Area Model

The fully-associative cache is very different from the set-associative cache in that in the fully-associative array the tag and the data array are one integral unit. One line of the cache will have tag bits, which if compare successfully will drive a line that acts as the wordline for the data bits. So unlike the set-associative cache where we interleave bits from the different blocks and also have

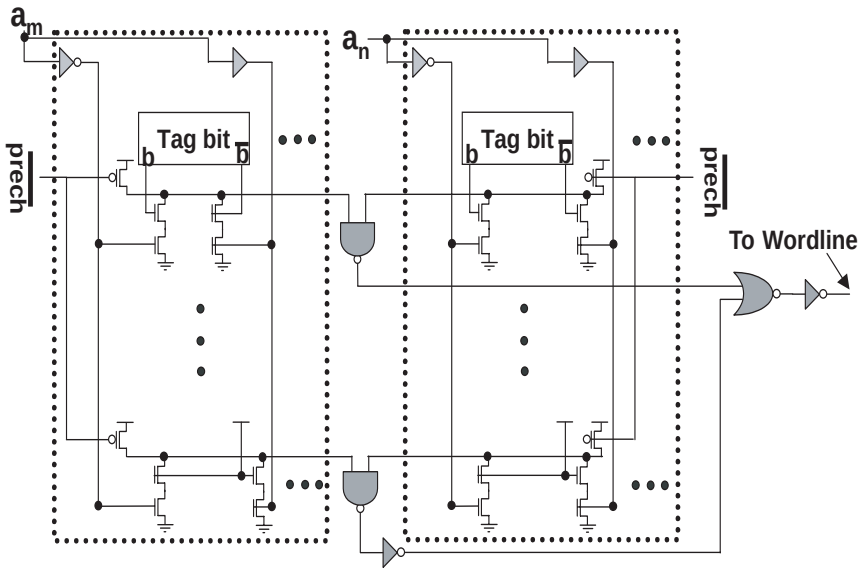

Figure 8: Fully-Associative Cache Structure

the data and the tag array separate, the fully-associative cache has the data bits and the tag bits associated with that address as one unit. Because there is no division of the block to smaller pieces, the different configurations have *Ndwl=Nspd=1* and *Ndbl* is alone varied. Also there are no separate *Ntwl, Ntbl* and *Ntspd.*

2.2.1 The Fully-Associative Subarray - Data and Tag Array Together

There is no separate data and tag array here. The subarrays contain both the data bits and the tag bits. If we consider these as units then the placement of the subarrays is exactly same as before.

The global tracks are the address lines to the predecoder (which is present at the center of the subarray placement), the predecode lines from the predecoder going to the NOR gates (which drive the tag array only - used for writing into the tag array), and the select lines for the output driver to read. In the case of the set-associative cache the data bits from one cache line are interleaved such that all subarrays produce different parts of the same cache line. In a fully-associative array the cache line and its corresponding tag bits are in the same row and consequently there is no interleaving because the whole cache line is driven onto the bitlines if the tag bits match. So the data bits and the tag bits that are routed are constant even as we go deeper into the array. Figure 9 shows the tracks for a single port. If we have multiple ports these tracks get multiplied by the appropriate number of ports.

$$Data\ Tracks = BITOUT\ \ [RWP + ERP + EWP]\ \ Widthtrack$$

$$Tag\ Tracks = Tagbits\ \ [RWP + ERP + EWP]\ \ Widthtrack$$

$$Output\ Driver\ Select\ Tracks = [RWP + ERP + EWP]\ \frac{8B}{BITOUT} Widthtrack$$

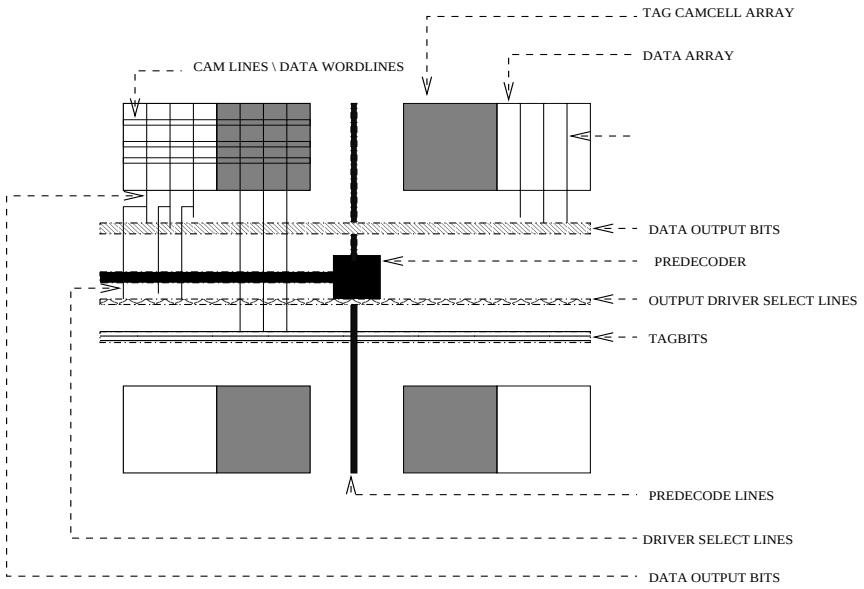

Figure 9: 2x2 Fully-Associative Subarray Structure

$$Predecode\ Tracks = [RWP + ERP + EWP]8 \cdot N3to8 \cdot Widthtrack$$

$$Address\ Tracks = AddressBits\,[RWP + ERP + EWP]Widthtrack$$

Let us assume for now that we know the subarray height and width for each cache configuration. Then as we know the exact method of placement of the subarrays and the exact number of routing tracks (and the wire pitch) we can calculate the height and width of the entire fully-associative array.

The time taken to drive the address lines to the predecoder is calculated accurately now since we have the knowledge of the exact length between the predecoder and the edge of the cache. The predecode lines travel in a H-tree fashion to the subarrays (in particular the tag subarray - used only when writing into the cache) and their length is also known. The time for driving the tagbits to the subarrays is also calculated in a very similar manner. So the decoding time (in the time model) is modified to take into account these exact lengths. Also the wire capacitance that is calculated from this length goes into the power calculation. The output driver has to drive the data (in a H-tree) to the edge of the cache. The wire capacitance and resistance is calculated for this exact length and inserted in the time model to account for the time and the energy consumed.

2.2.2 Fully-Associative Subarray

The number of columns in the subarray is the cache line size plus the tagbits. The data bit cell is a standard SRAM cell. The tag bit cell of the fully-associative array is a CAM cell. The height and width of the CAM cell as a function of the number of read ports, write ports, and read/write ports was evaluated from layouts and incorporated into the area model. One row of the fully-associative array starts from the decode NOR gates that are used for writing the cache, the tagbits

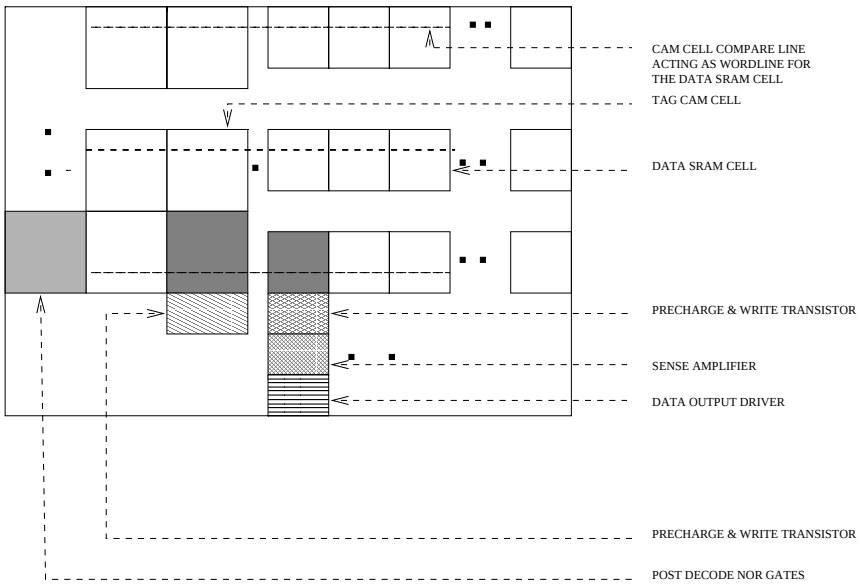

Figure 10: Fully-Associative Subarray Dissection

(CAM cells), and the data bits (the wordline driving these bits come from the compare operation performed on the tag bits). The I/O circuitry has only the sense amplifier and the output driver.

$$No \cdot ofRows = \left\lceil \frac{C}{[B\ Ndbl]} \right\rceil$$

$$No \cdot ofCols = [8\ B] + Tagbits$$

$$\begin{aligned} Fully\ Associative\ Subarray\ Height &= FA\,[CAMcellsandSRAMcells]\,Bit\ cells\ array \cdot height \\ &+ Precharge \cdot height\,[RWP + EWP] \\ &+ Sense\ Amplifier \cdot height\,[RWP + ERP] \\ &+ Output\ Driver \cdot height\,[RWP + ERP] \end{aligned}$$

$$Fully\ Associative\ Subarray\ Width = FA\,[CAMcellsandSRAMcells]\,Bit\ cells\ array \cdot width$$

2.3 Area Estimation of Individual Circuits

The bottom-most terms in the area hierarchy are the area of each individual component such as the bit cell, CAM cell, predecoder, sense amplifier, column multiplexer, etc. The only addition over the previous circuits is the CAM cell. Area estimates of the CAM cell were made by drawing layouts of it. Variants of it with different number and type of ports were also laid out to determine the height and width overhead for each type of port. The widths of the transistors were taken from the time model.

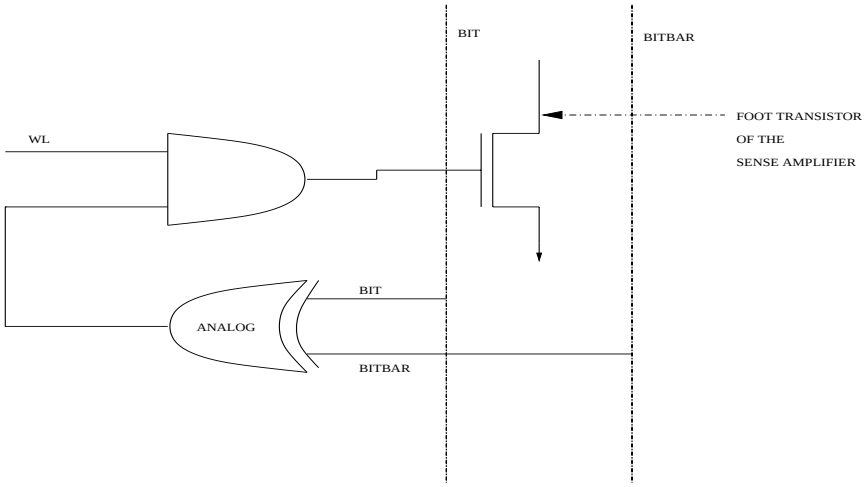

Figure 11: Control Logic For Turning the Foot Transistors of the Sense Amplifier ON

3 Power Reduction in the Sense Amplifier

The sense amplifier is a static circuit. Thus when both bit and bitbar are high (in the precharged state) there is a lot of current that is flowing through the foot of the circuit. This causes a lot of static energy dissipation. We have tried to reduce the time for which this energy is consumed. The signal that enables the 3 foot transistors of the sense amplifier is set high only when the word line at that bitcell reaches the threshold voltage of the pass transistors that control the read operation of the SRAM cell. Until then the foot transistors are not enabled. Hence there is no current flowing through them, and consequently there is zero energy dissipation. The foot is again turned off when the bitlines return to the precharged state. So the sense amplifier is effectively active only for the period from the wordline at the memory cell going high to the end of the total access. There are 2 distinct regions of energy consumption here - the period when the one of the bitlines is discharging, and the period when the bitlines have reached a stable final value (after the bitline has completed discharging). In the first period the current flow is higher and is not constant whereas in the second period the current drops to a low constant value. So there is high current flowing only for the relatively short duration when the SRAM cell is pulling one of the bitlines low.

By selectively enabling the sense amplifier we have been able to save a lot of energy. The energy is determined separately for the 2 regions - the energy in the first region has been encapsulated into a constant, the energy consumed in the second region can be calculated on the fly because both the voltage and the current in this region are constants.

4 Cache Banking Model

Support for more than one access to a memory structure at the same time can be provided by adding more ports to each memory cell. The ports can be exclusive read ports, exclusive write ports, or read/write ports. Multiporting a cache makes the cache capable of handling as many read

or write requests as ports of that type. But multiporting a cache makes each bit cell much bigger, and so the overall area of the memory array can increase enormously for large number of ports. The extra length spanned by the cache also adds directly to the access time and power consumed by the cache.

Another way of supporting simultaneous multiple accesses to a cache is by banking with fully-independent banks. Here the cache is divided into smaller arrays, each mapping a distinct address space. Each smaller array has its own independent address and data lines. This is as effective as having multiple ports if the multiple input addresses map onto different cache banks. But it is possible that more than one input address maps onto the same cache bank in which case the collision has to be resolved. There are many different mechanisms for resolving collisions, and we do not deal with those here. Banking also adds the decoding overhead of routing the appropriate address to the right bank and detecting collisions.

In CACTI 3.0 we have incorporated a very simple model of banking that does not include the extra decoding needed to route the address and data output lines to and from the banks. The mechanism for collision detection and resolution also has not been included. The banking model starts from the assumption that each bank has its own set of address and output buses. The specific extra decoding and the collision circuitry can be modeled on top of this model by the user.

4.1 Banking Area Model

The placement algorithm for banks is exactly the same as for the subarrays within a bank. In the case of a set-associative cache the data and tag arrays are placed together in each bank. In the case of a fully-associative cache the tag and data array were never separate from each other. The routing does not follow a H-tree structure and Figure 12 shows the structure of the routing that is used. The area of each bank is determined by the area model that was employed before. Therefore we know the height and the width of each bank. We account for the extra address and data tracks to calculate the entire area. The expression for calculating the total area of the cache with 8 banks is given below as an example.

$$\begin{aligned} Total\ Memory\ Height &= 2\ [Data\ Array\ Area \cdot width + Tag\ Array\ Area \cdot width] \\ &\quad + 0.5\ NBanks\ [Address\ Bits + BITOUT]\,[RWP + ERP + EWP] \end{aligned}$$

$$\begin{aligned} Total\ Memory\ Width &= 2\,[2Data\ Array\ Area \cdot height + 2\,[RWP + ERP + EWP]\,AddressBits] \\ &\quad + NBanks\ [Address\ Bits + BITOUT]\,[RWP + ERP + EWP] \end{aligned}$$

4.2 Banking Time Model

The cache is divided into banks and the total capacity of the cache is divided by the number of banks to get the capacity of each bank. Each bank is treated as an independent cache and the access time of the cache is calculated based on the earlier time model. Extra delays have to be included now for driving the address lines from the edge of the memory to each bank (the delays

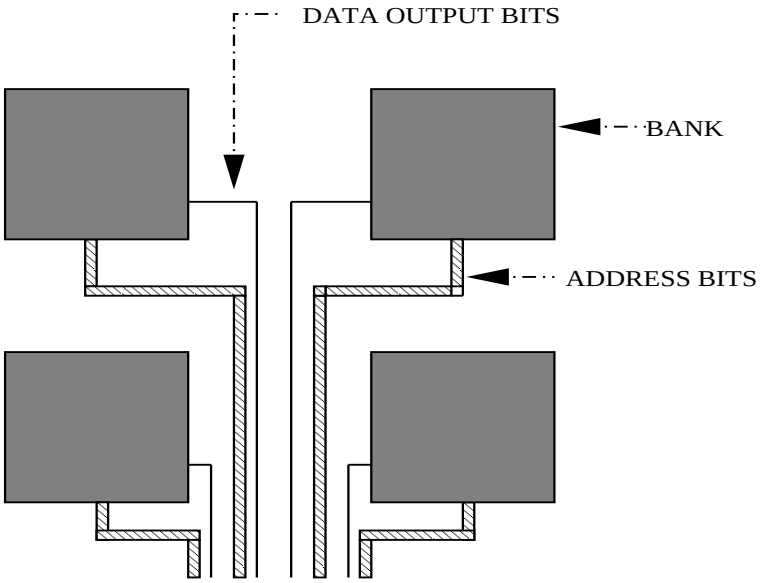

Figure 12: Bank Placement

within the bank have already been taken care of), and similarly for driving the output bits from each bank to the edge of the memory. A H-tree structure was not adopted here even though it is a more organized routing structure and has the advantage of having equal distances to each bank, because it takes more distance to get to some banks than it would it if we did not use a H-tree and the distances across banks are usually large. Not using a H-tree routing structure does not have an advantage in reducing the access time because the farthest distance to a bank which determines the access time remains the same. However, not using a H-tree results in smaller energy consumption because we drive less total distance. Thus to determine the access time of the caches we have to know the distance to the farthest bank for a given number of banks. We have this knowledge from the banking area model, and use this to calculate the extra wire capacitance and the resistance to the farthest bank. The decoder driver sees this additional capacitance and resistance to the predecoder. The data output driver also sees this extra capacitance and resistance to the edge of the memory. These have been incorporated into the time model. The bank routing distance for 16 banks is given below as an example.

$$Wire\ length\ Horizontal\ Dim = Bank\ Width$$

$$Wire\ length\ Vertical\ Dim = Bank\ Height + 2 \cdot Bank\ Height$$

4.3 Banking Power Model

A very simple way of calculating the total power of all the banks would be to multiply the power consumed by the farthest bank (since we know the driving distances already as we used those for the time model) by the number of banks. That will be a very simple method, but it would be wrong because the power that the farthest bank consumes would be the maximum power consumed by

any bank. It is the maximum power because the buffers driving the routing tracks drive the longest distances and hence expend the maximum power. The banking power model takes into account the different distances to each bank and separately accounts for the power spent in driving the address and data lines to and from each bank to the edge of the memory area to arrive at the total routing power. The power of each bank excluding the routing power is multiplied by the number of subbanks and is added to the routing power to get the total power burned by all the banks. The routing distances to each bank can be calculated with the help of the banking area model.

5 Integration of All the Models

The CACTI 2.0 optimization function depends only on the power and the access time. Since we have an area model in CACTI 3.0, we would like to incorporate some area parameters into the function, so that the final cache configuration has a reasonable total area, efficiency, and aspect ratio. We took a similar approach to that which had already been employed in adding the power parameters to the CACTI 2.0 optimization function:

$$\begin{aligned} Function\ to\ be\ Minimized \;=\; & c_1 \cdot \frac{power\ consumption}{maximum\ power\ consumption} \\ & + c_2 \cdot \frac{access\ time}{maximum\ access\ time} \\ & + c_3 \cdot \frac{1.0}{memory\ area\ efficiency} \\ & + c_4 \cdot \frac{aspect\ ratio}{maximum\ aspect\ ratio} \end{aligned}$$

The individual ratios (power/maxpower, time/maxtime, 1/efficiency, aspectratio/maxaspectratio) were observed to understand the ranges in which they operate. We observed that the function that minimizes the time and power alone ends up with pretty good efficiency and aspect ratios for the memory, except in some rare cases. So we decided to give more importance in optimizing for time and power than for efficiency and aspect ratio. We adjust c1, c2, c3, and c4 so that the final ratios are such that the contribution from power and time are approximately equal, the contribution from efficiency is greater than that from the aspect ratio, and both are at least an order of magnitude lower than the values for access time and power. The set-associative and fully-associative caches have different heuristic parameters for the optimizing function.

$$\begin{aligned} Set\ Associative\ Min\ Function \;=\; & 4 \cdot \frac{power\ consumption}{maximum\ power\ consumption} \\ & + 2 \cdot \frac{access\ time}{maximum\ access\ time} \\ & + 4 \cdot \frac{1.0}{memory\ area\ efficiency} \\ & + 1 \cdot \frac{aspect\ ratio}{maximum\ aspect\ ratio} \end{aligned}$$

$$\begin{aligned} \textit{Fully Associative Min Function} = & \; 0.5 \cdot \frac{\textit{power consumption}}{\textit{maximum power consumption}} \\ & + 1 \cdot \frac{\textit{access time}}{\textit{maximum access time}} \\ & + 10 \cdot \frac{1.0}{\textit{memory area efficiency}} \\ & + 0.025 \cdot \frac{\textit{aspect ratio}}{\textit{maximum aspect ratio}} \end{aligned}$$

It should be kept in mind that these are only heuristics and the user can use his or her own constants to optimize for one contribution over the other, or even introduce more properties into the optimization function. The interesting thing to note is that we have control over the time, power, areal efficiency, and aspect ratio of the cache at this final stage and can trade one for another, depending on what is more important for our target application.

6 CACTI 3.0 Results

In this section we first look at the results from the area model. We then consider the impact of CACTI 3.0's improvements on the time and power models.

6.1 Area Model Results

Figure 13 shows the area of the cache as the associativity is varied. The Y-axis is in a log scale. The general observation is that the area of the cache increases very slowly as we increase associativity, and it is quite insensitive to the associativity for a set-associative cache. The area of a fully-associative cache is less than that of a 16-way cache for small cache sizes, but quickly overtakes it for larger caches.

We will now examine the individual factors that contribute to increase in the area of the cache as we increase the associativity. The number of tagbits increase as we increase the associativity. This contributes to a small increase in the area of the tag array. The number of output driver select lines also increase as we increase the associativity. This contributes to a small increase in the area occupied by the decoder circuitry and also increases the area occupied by the routing tracks that go from the tag to the data array. The area of the data subarray is unaffected by the increase in associativity. The number of columns in the subarray is directly proportional to the associativity, but the number of rows is inversely proportional to it. So the number of columns we add is compensated by the reduction in the number of rows. Also the width of the subarray is determined by the number of columns it has and the width of each bit cell, whereas the height of the subarray is the same (as shown in Figure 6). This is true whether there are few sense amplifiers (large number of column multiplexers) or a relatively large number of sense amplifiers (no column multiplexing). This explains the subarray area's insensitivity to associativity. Another thing to keep in mind is the range of allowed values for the configuration parameters (*Ndwl*, *Ndbl*, etc.,). As we increase the associativity, a higher value of *Ndwl* would give a lower access time because of

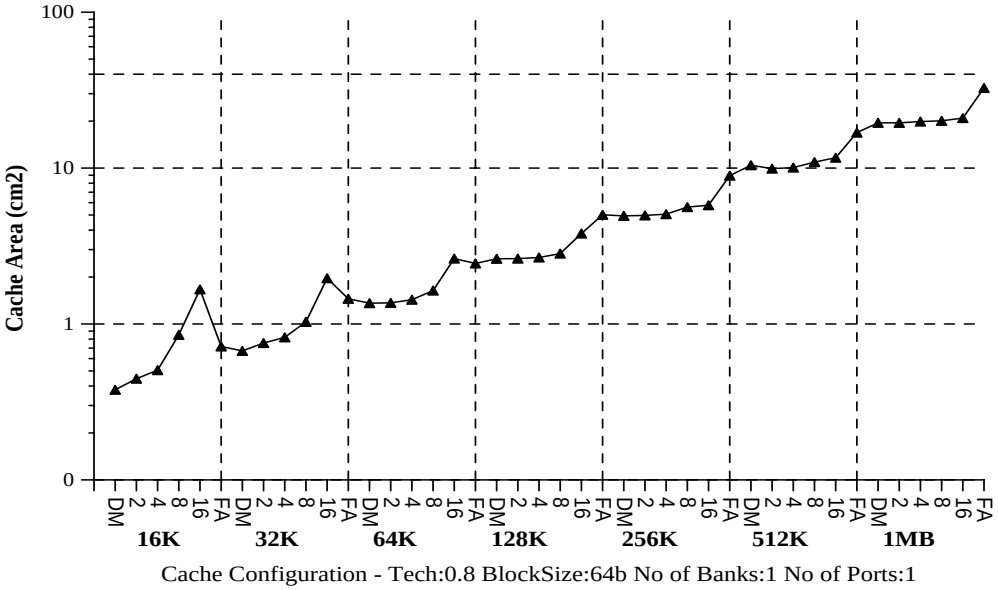

Figure 13: **Area Model Sensitivity to Cache Size and Associativity.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different associativities.

the reduced wordline delay, but these optimizations are possible only if Ndwl is allowed to take these values. The allowed values of these configuration parameters might limit the effectiveness of these optimizations in different situations, and in this particular case the constraints on Ndwl may make the algorithm less effective for large values of associativity. The final set of configuration parameters that we arrive at (which are different for different associativities) decide the layout of the cache and its area. For smaller cache sizes the set-associative cache has low effficiency because of the large number of routing tracks between the tag array and the data array. But as we go to larger caches the bigger size of the fully associative CAM cell outweighs the routing area in the set-associative cache.

Figure 14 shows the area of the cache as the number of ports per each SRAM cell is varied. Each cache shown on the graph has 1 read/write port and the rest are read only ports. The graph shows the very simple trend of the cache area increasing steeply as we increase the number of ports.

Figure 15 shows the total area of the cache as a function of the number of banks it is divided into. The graphs look quite different for small cache sizes as compared to larger caches. We observe that for small cache sizes the cache shows a steady and slow increase in area as we increase the number of banks. This is because the extra overhead of the wiring tracks, I/O circuitry, etc., causes the area to exceed just the product of the area of one bank by the number of banks. So there is no advantage in terms of the cache area alone in having more cache banks. But the shape of the graph is quite different as we move to larger cache sizes. We start observing a different trend at 256K where the graph has a local minimum at 8 banks. For cache sizes 512K and 1MB we see well defined local minima at 8 banks. The larger caches offer greater flexibility to the algorithm to arrive at a better configuration. The 512K cache at 4 banks has each bank of size 128K which

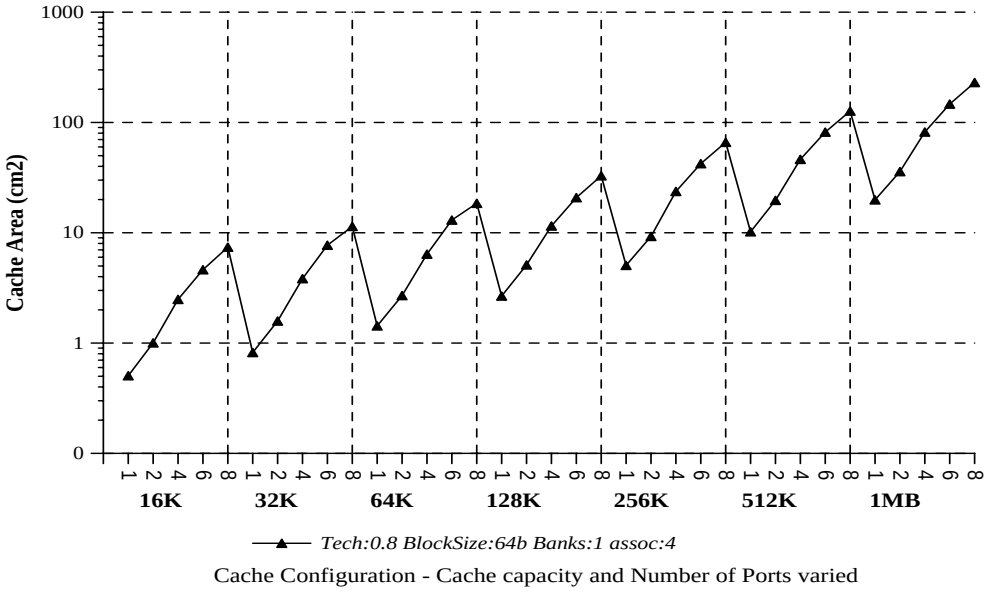

Figure 14: **Area Model Sensitivity to Size and Number of Ports.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different number of ports.

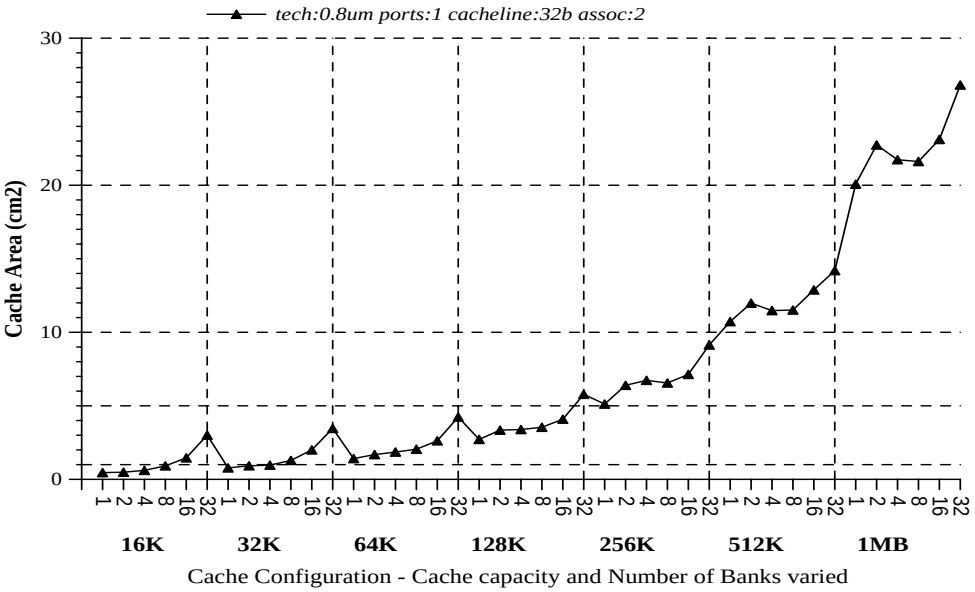

Figure 15: **Area Model Sensitivity to Size and Banking.** The Y-axis is on a linear scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different number of banks.

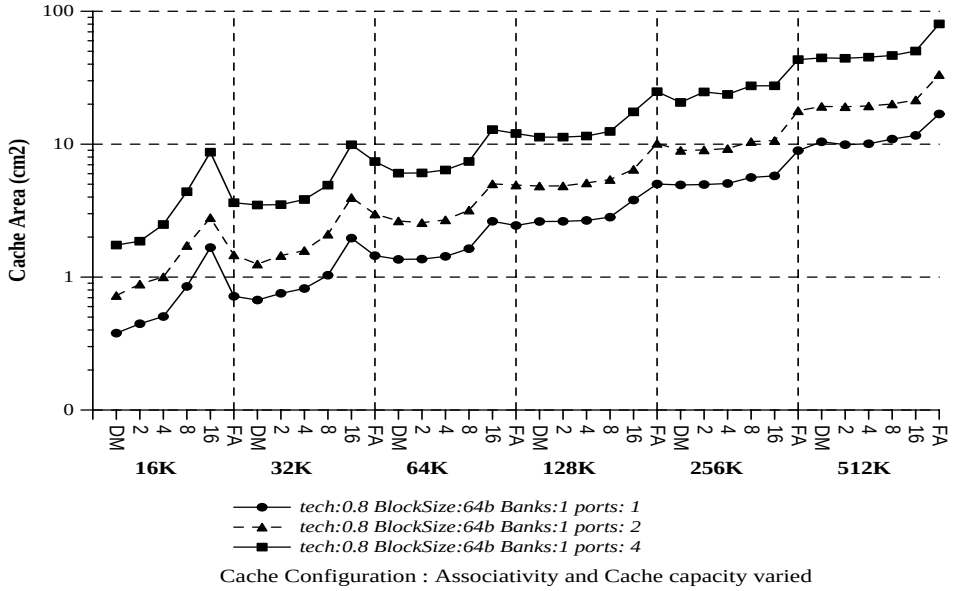

Figure 16: **Area Model Sensitivity to Size, Associativity, and Ports.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different number of banks. The different graphs are for different numbers of ports.

provides enough leeway for the algorithm to arrive at the best subarray parameters *Ndwl*, *Ntwl*, etc. Also the bank routing overhead is relatively low and results in a good overall efficiency and lesser area.

Figure 16 shows the area of the cache as the associativity is varied for different numbers of ports. The Y-axis is in a log scale. The general observation is the same as for the graph in Figure 13. The multiple graphs show the same trend at different number of ports, and we see that for the same value of associativity the cache with greater number of ports has greater area.

6.2 Time and Power Model Results

Figure 17 shows the variation of cache power for different associativities and number of ports. The basic observation is that as the associativity increases, the energy consumed by the cache also increases, especially when the cache size is small. The increase in the size of the tag array leads to a slight increase in the bitline and wordline power in the tag array. The increase in associativity means a greater number of output drivers, sense amplifiers, comparators, etc., and consequently an increase in the energy consumed. We also have to keep in mind that the final configuration parameters can change with associativity and this influences the energy consumption also.

Figure 18 shows the variation of access time as the cache size is increased and the associativity varied for each capacity. This graph is very similar to the graph in the previous CACTI 2.0 report plotting the same variables. The explanation for the trend is the same.

Figure 19 shows cache access time as a function of the number of banks the cache is divided into.

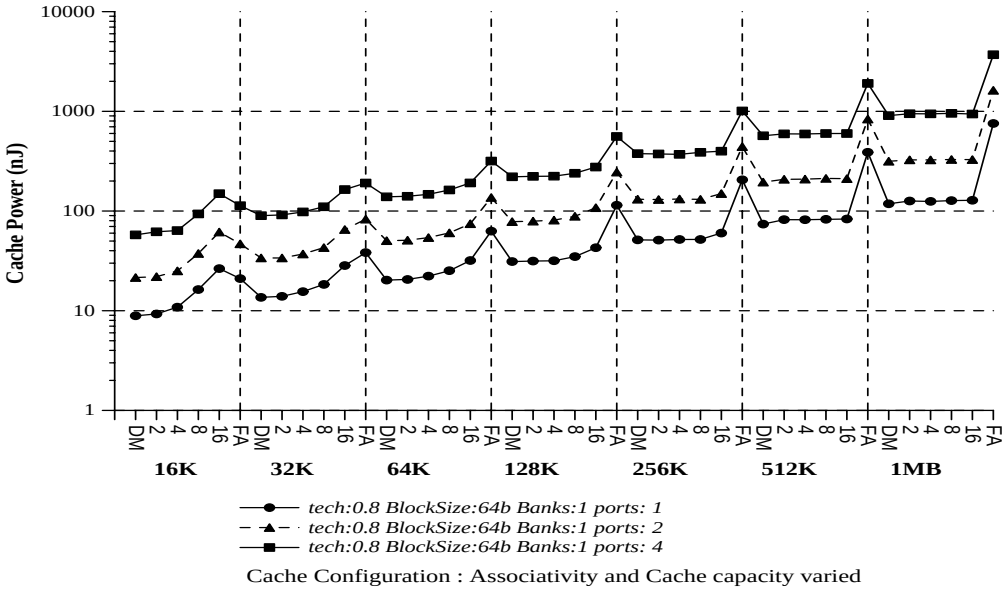

Figure 17: **Power Model Sensitivity to Size, Associativity, and Ports.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different associativities. The different graphs are for different numbers of ports.

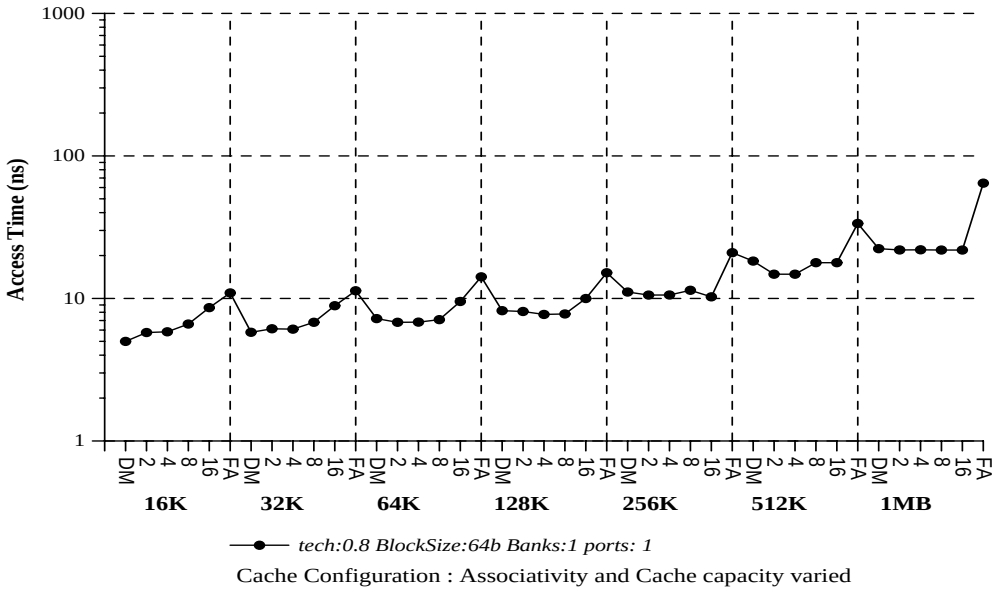

Figure 18: **Time Model Sensitivity to Size and Associativity.**. The y-axis is on a log scale. The x-axis ranges over a variety of cache configurations. Each cache size is given with different associativities.

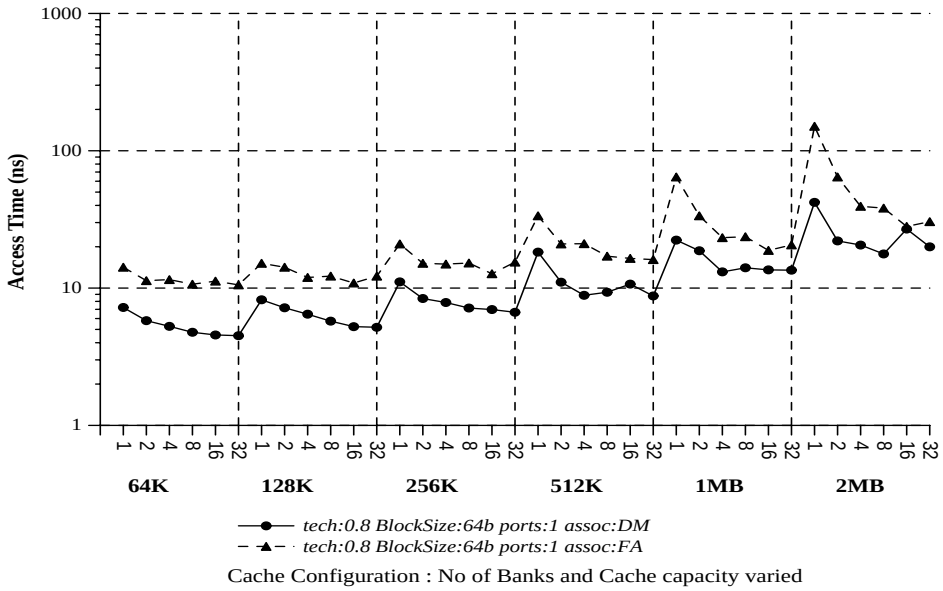

Figure 19: **Time Model Sensitivity to Size, Associativity, and Banks.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is broken down into different number of banks. The different graphs are for the direct mapped and fully associative caches.

The general observation is that for the fully associative cache the access time monotonically drops as the number of banks is increased. There are two effects we observe when we increase the number of banks. First, the individual bank access time decreases because each bank is smaller. Second, the time spent on getting the address bits to the bank and routing the data out of the memory increases because we have to travel a greater distance. The benefits from the reduced bank access time outweigh the increase in routing time, and so we obtain overall reduced total cache access times.

The direct mapped cache access time is dominated by the data side delay. For small cache sizes where the data and the tag delay are comparable, we observe that the access time drops steadily. As we go to bigger caches for a small number of banks, the data delay is much greater than the tag side delay, as the size of the data array is much larger than the tag array. As we increase the number of banks the data side and tag side delay become comparable. When the tag and data side delay are comparable we see a overall lower access time, and that explains the local minima in the graph for large cache sizes. When the number of banks is greater than this optimum, the access time either remains constant or increases because of the greater routing delay that outweighs the benefits from the lower bank access time.

Figure 20 shows cache access time as a function of the number of banks the cache is divided into. The general observation is that for the set-associative cache the access time remains almost constant as the number of banks is increased for small caches. This is because the time to access a bank gets smaller as we decrease the bank size but we need to spend greater time now in routing the data out of the cache because there are more banks. As we go to bigger caches we initially get the

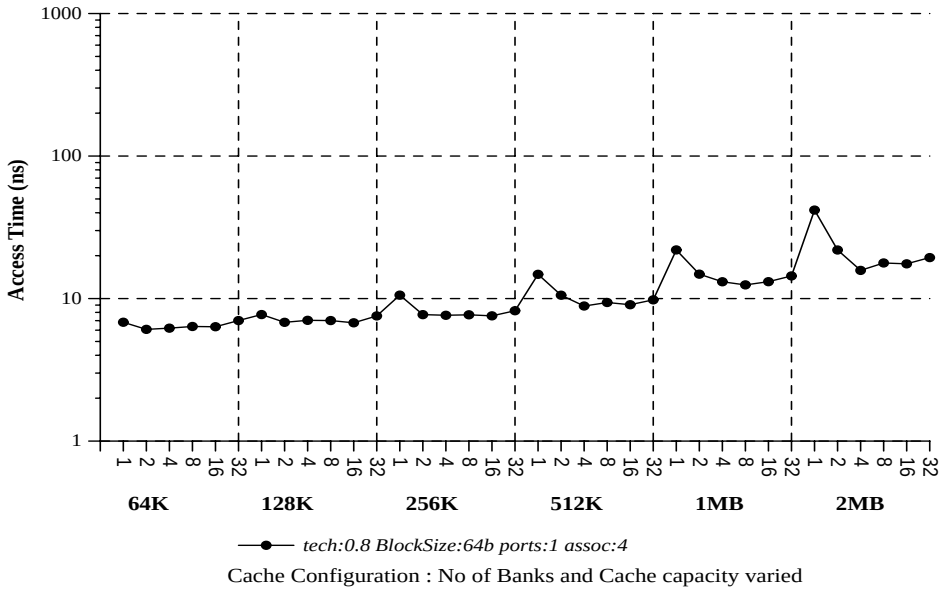

Figure 20: **Time Model Sensitivity to Size and Banks.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different number of banks.

benefit of going to smaller banks in the smaller bank access time. The extra routing delay is still much less than the benefit we get from the reduced bank access time, and so the overall access time drops. But as we go to very small banks sizes we get diminishing returns from the reduced bank access times and the extra routing delay outweighs the drop in bank access time, so the overall access time increases.

Figure 21 shows the total power consumed by the cache as we increase the number of banks. The different graphs are for different number of banks. If we take the graph corresponding to one bank, it is very similar to the graph in Figure 17. The interesting thing to observe here is that as we increase the associativity, the increase in power consumed by the cache is greater if the number of banks is greater. This is because the effect we observed in Figure 17 is magnified by the number of banks the cache is divided into.

7 Area Model Validation

In this section we compare the area of actual caches measured from chip microphotographs[5] with those predicted for the same cache configuration by CACTI 3.0. Table 1 gives the cache configurations of the actual microprocessor caches along with their process technology. The rightmost column in this table contain notes that refer to some important details of the cache that can affect the area calculation. These note numbers index the list below:

1. The EV4 I-cache contains built in self-test and self-repair logic. The Ev4 D-cache uses software-assisted self-test.

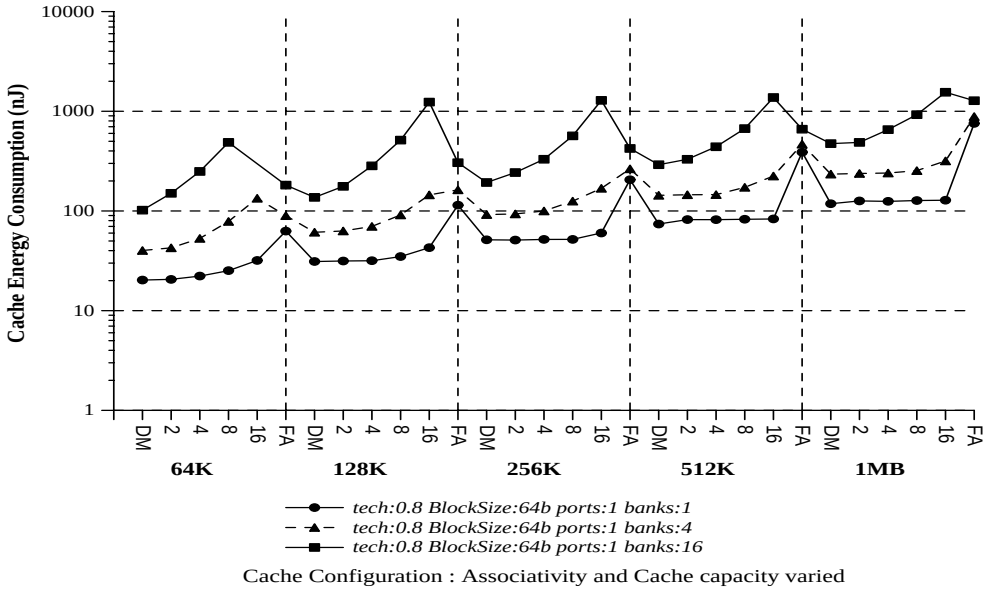

Figure 21: **Power Model Sensitivity to Cache Size, Associativity, and Banking.** The Y-axis is on a log scale. The X-axis ranges over a variety of cache configurations. Each cache size is given with different associativities. The different graphs are for different numbers of banks.

Machine	Size (bytes)	Cacheline (bytes)	Assoc	Gate Length (um)	RWP	ROP	WOP	Banks	Notes
$EV4 - L1I$	8192	32	1	0.75	1	0	0	1	1
$EV4 - L1D$	8192	32	1	0.75	1	0	0	1	1
$EV5 - L1I$	8192	32	1	0.5	1	0	0	1	2
$EV5 - L1D$	8192	32	1	0.5	1	0	0	1	2
$EV5 - L2$	110592	36	3	0.5	1	0	0	1	3
$EV6 - L1I$	65536	64	2	0.35	1	0	0	1	4
$EV6 - L1D$	73728	72	2	0.35	1	0	0	1	3,4

Table 1: Cache configurations used in the validation of the area model.

2. EV5 supports up to two data accesses per cycle. EV5 handles this requirement by implementing the L1D as two 8K caches, whose contents are kept identical. Each 8K cache in the L1D is roughly the same design as the 8K L1I cache.

3. Cache and line size are assumed to be 9/8 larger than data capacity due to use of ECC on write-back caches.

4. EV6 caches are split into 8 banks total[6]. We assume 4 banks are used for each degree of associativity. EV6 data caches use a wave pipelined design, which CACTI does not take into account in its area model.

The area of the caches predicted by CACTI are in all cases within 10% of the actual microprocessor cache areas (see Table 2). This is encouraging, since CACTI models a relatively generic design with relatively square aspect ratio, while the actual microprocessor caches often use RAM cells with special design rules, or their area is increased by aspect ratio changes dictated by global floorplan requirements.

8 Future Work

Embedded DRAM is an architecture which merges dynamic random access memory and logic. It is an important architecture for future VLSI systems because of the advantages it offers in terms of reduced power consumption, larger bandwidth between the memory and the core, and greater packing density. It should be an interesting area to explore in future versions of CACTI.

Acknowledgements

We would like to thank Rob Stets of WRL for his early work in validating the accuracy of the area model.

Machine	From Die Photo			From CACTI			area
	x(mm)	y(mm)	area(mm^2)	area(mm^2)	data(ns)	tag(ns)	error
$EV4-L1I$	9.90	2.06	20.39	20.51	3.35	3.24	-1%
$EV4-L1D$	9.04	2.18	19.70	20.51	3.35	3.24	4%
$EV5-L1I$	6.12	1.45	8.86	9.11	2.23	2.16	3%
$EV5-L1D$	6.45	1.41	9.11	9.11	2.23	2.16	0%
$EV5-L2$	non-rectangular		88.65	96.88	4.55	4.51	9%
$EV6-L1I$	non-rectangular		38.01	34.81	2.15	2.74	-8%
$EV6-L1D$	non-rectangular		37.26	38.67	2.15	2.74	4%

Table 2: Results of comparison of actual cache areas to CACTI 3.0 model results.

References

[1] Steven J. E. Wilton and Norman P. Jouppi. *CACTI: An Enhanced Cache Access and Cycle Time Model.* IEEE Journal of Solid-State Circuits, May 1996, pp 677-688.

[2] Glenn Reinman and Norman P. Jouppi. *CACTI 2.0: An Integrated Cache Timing and Power Model.*

[3] Anantha Chandrakasan, William J. Bowhill, and Frank Fox, editors. *Design of High Performance Microprocessors Circuits.* IEEE Press, Piscataway, NJ, 2001.

[4] Bharadwaj S. Amrutur, Mark A. Horowitz. *Speed and Power Scaling of SRAM's.* IEEE Journal of Solid-State Circuits, Feb 2000, pp 175-185.

[5] Paul E. Gronowski, William J. Bowhill, Ronald P. Preston, Michael K. Gowan, and Randy L. Allmon *High-Performance Microprocessor Design.* IEEE Journal of Solid-State Circuits, May 1998, pp 676-686.

[6] Bruce Gieseke, et. al. *A 600MHz 64b Superscalar RISC Microprocessor with Out-of-Order Execution.* in ISSCC Digest Tech. Papers, February 1997, pp 176-177.

A CACTI Syntax

cacti ⟨*csize*⟩ ⟨*cacheline*⟩ ⟨*assoc*⟩ ⟨*tech*⟩ ⟨*Nbanks*⟩
OR
cacti ⟨*csize*⟩ ⟨*cacheline*⟩ ⟨*assoc*⟩ ⟨*tech*⟩ ⟨*RWP*⟩ ⟨*RP*⟩ ⟨*WP*⟩ ⟨*Nbanks*⟩

csize size of the cache in bytes (i.e. 32768)

cacheline size of the cache line in bytes (i.e. 64)

assoc associativity of the cache (i.e. 2,4 or FA)

direct mapped cache - DM

fully associative cache - FA

tech - feature size in microns (i.e. 0.8, 0.18)

RWP - number of read/write ports (defaults to 1)

RP - number of red ports (defaults to 0)

WP - number of write ports (defaults to 0)

Nbanks - Number of banks

B CACTI Output

The command

```
cacti 32768 32 2 0.80um 2
```

will return the following timing and power analysis for a 16K 2-way set associative cache with a 32-byte blocksize at a 0.80um feature (technology) size with 2 independently addressed banks:

```
Cache Parameters:
  Number of Banks: 2
  Total Cache Size: 32768
  Size (in bytes) of Bank: 16384
  Number of Sets: 512
  Associativity: 2
  Block Size (bytes): 32
  Read/Write Ports: 1
  Read Ports: 0
  Write Ports: 0
  Technology Size: 0.80um
  Vdd: 4.5V

Access Time (ns): 5.84503
Cycle Time (wave pipeline limit) (ns):  1.94834
Total Power all Banks For Read (nJ): 19.2097
Total Power all Banks For Write (nJ): 16.6977
Total Power Without Routing For Read (nJ): 19.2097
Total Power Without Routing For Write (nJ): 16.6977
Total Routing Power (nJ): 0
Maximum Bank Power (nJ):  9.60484

Best Ndwl (L1): 2
Best Ndbl (L1): 4
Best Nspd (L1): 1
Best Ntwl (L1): 1
Best Ntbl (L1): 2
Best Ntspd (L1): 2
Nor inputs (data): 2
Nor inputs (tag): 2

Area Components:

Aspect ratio - total height/width: 8.922045
```

```
Data array (cm^2): 0.417864
Data predecode (cm^2): 0.000759
Data colmux predecode (cm^2): 0.000381
Data colmux post decode (cm^2): 0.000059
Data write signal (cm^2): 0.000327

Tag array (cm^2): 0.031064
Tag predecode (cm^2): 0.000759
Tag colmux predecode (cm^2): 0.000381
Tag colmux post decode (cm^2): 0.000059
Tag output driver decode (cm^2): 0.000664
Tag output driver enable signals (cm^2): 0.000327

Percentage of data ramcells alone of total area: 61.237504 %
Percentage of tag ramcells alone of total area: 4.784180 %
Percentage of total control/routing alone of total area: 33.978316 %

Bank efficiency : 66.021684%
Total efficiency : 64.851059%

Total area for one bank (cm^2): 0.452644
Total area (cm^2): 0.921629

Time and Power Components:
 data side (with output driver) (ns): 4.43827
 tag side (with output driver) (ns): 5.84503
 address routing delay (ns): 0
 address routing power (nJ): 0
 decode_data (ns): 1.71202
             (nJ): 2.72016
 wordline and bitline data (ns): 1.2508
            wordline power (nJ): 0.0467634
             bitline power (nJ): 2.43285
 sense_amp_data (ns): 0.83
                (nJ): 1.28161
 decode_tag (ns): 0.955464
            (nJ): 0.526854
 wordline and bitline tag (ns): 0.629084
           wordline power (nJ): 0.0176002
            bitline power (nJ): 0.657433
 sense_amp_tag (ns): 0.51
               (nJ): 0.350613
 compare (ns): 1.29042
         (nJ): 0.0744832
 mux driver (ns): 1.62476
```

```
           (nJ): 0.241649
sel inverter (ns): 0.189851
             (nJ): 0.0023513
data output driver (ns): 0.645442
                   (nJ): 1.25247
total_out_driver (ns): 0
                 (nJ): 0
total data path (without output driver) (ns): 3.79282
total tag path is set assoc (ns): 5.19958
```

C Meaning of the Terms Used in the Equations and Text

C: Cache Capacity in bytes

B: Cacheline Size

A: Associativity

$BITOUT$: Width of the data output bus in bits

$Ndbl, Ndwl, Nspd$: Configuration parameters for the data array

$Ntbl, Ntwl, Ntspd$: Configuration parameters for the tag array

RWP: Number of read/write ports

ERP: Number of read only ports

EWP: Number of write only ports

$N3to8$: Number of 3 to 8 predecode Nand gates

$Widthtrack$: Wire pitch

$Subarray$: Each Bank is partitioned into a number of Subarrys determined by the values of Ndwl and Ndbl (Ntwl and Ntbl for the tag array)

$NBanks$: Number of Banks the cache is divided into

$DataSubblock$: The 2x2 Data Subarray unit

$TagSubblock$: The 2x2 Tag Subarray unit

$Cwordmetal$: The capacitance of a wire whose length is equal to the width of the bit cell

$Cbitmetal$: The capacitance of a wire whose length is equal to the height of the bit cell

$Rwordmetal$: The resistance of a wire whose length is equal to the width of the bit cell

$Rbitmetal$: The resistance of a wire whose length is equal to the height of the bit cell