ABSTRACT

Title of dissertation: myCACTI: A NEW CACHE DESIGN TOOL FOR PIPELINED NANOMETER CACHES

Samuel Verzola Rodriguez, Doctor of Philosophy, 2006

Dissertation directed by: Associate Professor Bruce L. Jacob
Department of Electrical and Computer Engineering and Institute for Advanced Computer Studies
University of Maryland, College Park

The presence of caches in microprocessors has always been one of the most important techniques in bridging the memory wall, or the speed gap between the microprocessor and main memory. This importance is continuously increasing especially as we enter the regime of nanometer process technologies (i.e. 90nm and below), as industry has favored investing a larger and larger fraction of a chip's transistor budget to improving the on-chip cache. This is the case in practice, as it has proven to be an efficient way to utilize the increasing number of transistors available with each succeeding technology. Consequently, it becomes even more important to have cache design tools that give accurate representations of designs that exist in actual microprocessors.

The prevalent cache design tools that are the most widely used in academe are CACTI [Wilton1996] and eCACTI [Mamidipaka2004], and these have proven to be very useful tools not just for cache designers, but also for computer architects. This dissertation will show that both CACTI and eCACTI still contain major limitations and even flaws in their design, making them unsuitable for use in very-deep submicron and nanometer caches, especially pipelined designs. These limitations and flaws will be discussed in detail.

This dissertation then introduces a new tool, called myCACTI, that addresses all these limitations and, in addition, introduces major enhancements to the simulation framework. Some of the major enhancements are briefly described as follows:

- Use of SPICE BSIM4.0 equations to accurately characterize device behavior for nanometer process technologies. In contrast, CACTI and, to a major extent, eCACTI simply use hardcoded parameters derived for an obsolete 0.80μm process technology.
- The modeling of a typical explicitly-pipelined cache, which accounts for all the overhead in pipelining that will be present in virtually all industry-level microprocessor caches. In contrast,

CACTI and eCACTI model wave-pipelined cache, something that is not representative of commercial designs.

- Inclusion of more optimal variable stage dynamic logic circuits for the decode hierarchy that provides the tool more flexibility in finding optimal implementations. In contrast, both CACTI and eCACTI model a fixed-stage static CMOS decode hierarchy, significantly limiting the optimization search.
- Inclusion of an accurate model and per-process numbers for a typical BEOL-stack that are representative of nanometer processes. The significance of this is made even more important given the tremendous effect of interconnect parasitics on a cache's behavior.
- Inclusion of a gate leakage tunneling current model for improved handling of static power dissipation.
- Inclusion of a very realistic interconnect model that is representative of the interconnects in a real nanometer cache. In contrast, both CACTI and eCACTI have an unrealistic model of the interconnect as they assume the use of interconnect with a single characteristic no matter where it is located and used in the cache.

This dissertation then demonstrates the use of myCACTI in the cache design process. Detailed design space explorations are done on multiple cache configurations to produce pareto optimal curves of the caches to show optimal implementations. Detailed studies are also performed to characterize the delay and power dissipation of different cache configurations and implementations. Some of the more important observations, among the many that were found, are as follows:

- The pipeline power dissipation overhead is very significant and it typically dominates the total power.
- Interesting non-monotonic behavior with respect to delay and power dissipation for caches with different associativities exist, such that we can conclude that some optimal implementations are definitely superior than other optimal implementations In other words, overlapping pareto optimal curves result in some optimal points being reconsidered as optimal.
- The power dissipation due to gate leakage tunneling current is surprisingly not as significant as initially expected.

Finally, future directions to the development of myCACTI are identified to show possible ways that the tool can be improved in such a way as to allow even more different kinds of studies to be performed.

myCACTI: A NEW CACHE DESIGN TOOL FOR PIPELINED NANOMETER CACHES

by
Samuel Verzola Rodriguez

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Advisory Committee:
Associate Professor Bruce L. Jacob, Chair

Associate Professor Gang Qu

Assistant Professor Pamela Abshire

Professor Romel Gomez

Associate Professor Chau-Wen Tseng

ACKNOWLEDGEMENTS

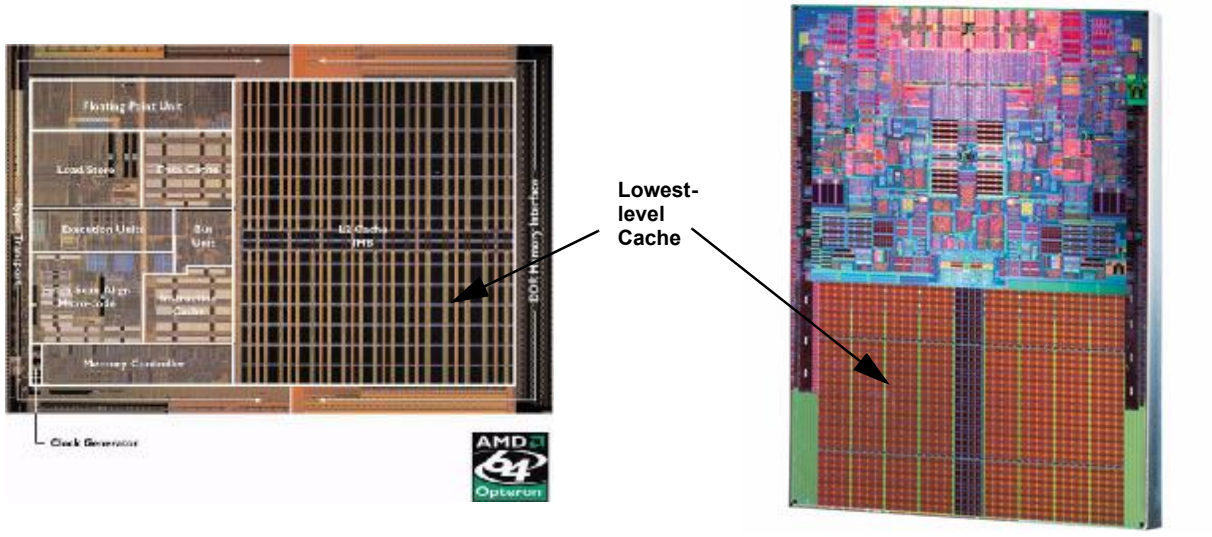
This dissertation would not have been possible without the contribution of scores of people who have helped shape my thesis and my life, either directly or indirectly. First and foremost is my family -- my tatay and nanay, and my siblings Ate Yoyi, LJ and Bads. Their support allowed me to keep my sanity, stay grounded, and above all, remember my pride in being a Filipino. To my advisor, Bruce Jacob -- for all his mentoring and all the help, not to mention the all-important funding, during my stay in Maryland. Going into your office for one of those two-hour talks about anything and everything under the sun was always fun and interesting, to say the least. To Dr. Romel Gomez, I will also always be grateful, for revealing to me the possibilities of grad school, for helping in my admission to Maryland, and for keeping tabs on me the whole time I was there. To the friends I made in College Park -- Xia, Cagdas, Vince, Amol, Tad, Joe, Jimi, Ohm, Ankush, Brinda, Sada, Dave, Jason, Babak and a lot more. Grad school without friends wouldn't be a pretty thing, and I'm glad that even though I'm reticent in getting to know new people, that I had the pleasure of getting to know you guys. To the awesome people at AMD -- I'm grateful to so many of you that I'll have to mention only the guys I had to directly answer to: Rob, Matt, Kathy and Jim. If I name all of you guys, I may end up having to name half of the people at the Boston Design Center. I'm looking forward to renewing acquaintances with you guys once I start working there, and to knowing all of the other people I didn't get a chance to meet. To Leo Tordil and family -- I'll always owe you a favor for providing a roof over my head during the first three months of my stay in the U.S. -- and I offer my apologies for my personality flaw that prevented me from keeping in close touch. To my professors and acquaintances at U.P. -- Gev, Luis, Manuel, Chrd, Marc, CMO, Frank, Eggy, JYG and Chuy, among others -- for encouraging me to go to grad school in the U.S., without which I probably wouldn't have gathered enough courage to start this great and fulfilling journey. To the support guys at Maryland -- Shyam, Jay, To-Anh, Maria, and tons of others. A school is always made by brilliant, hard-working people striving to keep the gears turning. To some of the collegiate sport icons I got to watch at Maryland -- Dixon, Baxter, Blake, Mouton, Wilcox and of course, Vernon Davis -- you made being a Terrapin fun in spite of the weird name -- Go Terps! To my Pisay batchmates who proved that sometimes, it's really a small world by ending up only miles from where I live even though we're eight thousand miles from home -- Toby, Jerry and Andre. Talking with you guys made being alone in the U.S. bearable. Also a very special thanks to Aamer -- a friend, fellow UMD student, research assistant, and the best roommate I ever had. I'll always be eternally grateful to how you put up with me during that awesome but painful, bittersweet six months of 2005. I'm not sure what I would have done without you, and I hope we'll always be friends -- I'll even overlook the fact that you work for Intel. And finally, the person who is the main reason why I strived to do this and to whom this dissertation is dedicated to -- *Tats, para sa iyo ito, kahit hindi mo na ako mahal.*

CHAPTER 1 Introduction

The presence of caches in microprocessors has always been one of the most important techniques in bridging the memory wall, or the speed gap between the microprocessor and main memory. This importance is continuously increasing especially as we enter the regime of nanometer process technologies[1], as industry has favored investing a larger and larger fraction of a chip's transistor budget to improving the on-chip cache, as it has proven to be an efficient way to utilize the increasing number of transistors available with each succeeding technology. This can be visually seen in the die photos of contemporary processors, as shown in Figure 1-1, where the cache area dominates the total die area. Consequently, it becomes even more important to have cache design tools that give accurate representations of designs that exist in actual microprocessors. The power breakdown in a contemporary Intel-based system is shown in Figure xx [Intel2006], while the power breakdown of a representative processor (Alpha 21264 [Gowan1998]) is shown in Figure xx.

1.1 Problem Description

Design tools have always been an integral part of microprocessor design, especially with the onset of VLSI technology that makes it impossible to design, build and debug circuitry the same way it was done



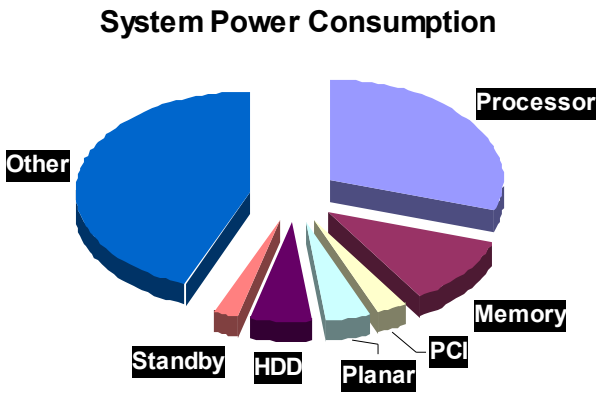
(a) AMD Opteron (b) Intel Woodcrest

Figure 1-1: Die photos of contemporary processors showing how cache dominates the total die area. (a) AMD Opteron. (b) Intel Woodcrest. The lowest-level cache of both processors are pointed to by the arrow.

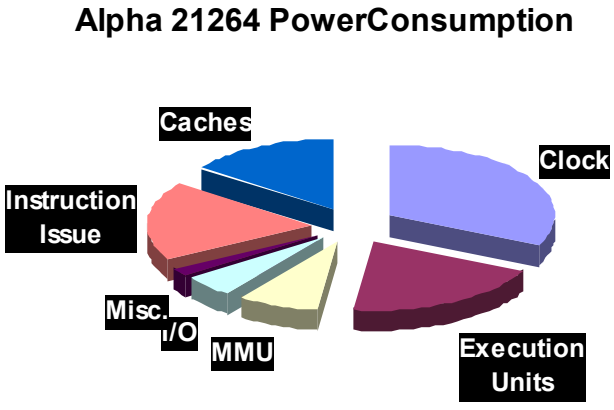
1. Here, we define nanometer process technologies to start from 90nm onwards. All of the studies in this dissertation use process technologies of 90nm, 65nm, 45nm and 32nm, with some studies also containing results for 130nm.

during the small-scale and medium-scale integration era. The use of design tools was important in exploring the available design space and determining optimal implementations given a set of restrictions.

The cache design tool that is the most widely used in academe is CACTI, and it has proven to be a very useful tool not just for cache designers but also computer architects, enabling users to characterize cache behavior without delving too much in some of the more esoteric details of cache design. CACTI, though, was based on assumptions from an obsolete long-channel 0.80µm technology, and with the entry of fabrication process technologies into the submicron and deep-submicron regimes, CACTI results have become aged and may be suspect. eCACTI was written as an enhancement to CACTI and partly updates it to account for some of its limitations, the biggest of which was the inclusion of a subthreshold leakage model to account for the increasing leakage in deep submicron technologies. Even so, both tools still have significant limitations and



(A)



(B)

Figure 1-2: Power breakdowns. (A) Power breakdown in the entire system [Intel2006], and (B) Power breakdown in a representative microprocessor [Gowan1998].

inaccuracies. These inaccuracies may not be overly significant even at deep-submicron technologies, but as we go deeper into very-deep-submicron and eventually nanometer technologies like 90nm, 65nm, 45nm and eventually 32nm, the model limitations of both CACTI and eCACTI, as we will show, will yield inaccurate and misleading results. Some of these major limitations are listed below:

- CACTI and, except for static power computation, eCACTI generate results using a 0.80µm reference technology and then extrapolate it to smaller technologies like present and future nanometer technologies by performing a simple linear scaling of the results, resulting in inaccurate data because many transistor device parameters have not exhibited linear scaling with reference to the 0.80µm process technology.
- Both CACTI and eCACTI implicitly handle cache pipelining by using wavepipelining, resulting in severe misrepresentation of the overhead required in explicitly pipelining a cache to properly interface it with a microprocessor. In addition, not all caches produced by CACTI and eCACTI can be wave-pipelined, as this is dependent on the specifics of the optimal implementation found.
- CACTI and eCACTI have unrealistic assumptions with regards to address decoding circuitry that severely restrict the search for optimal implementations and in the worst case, produces inaccurately optimistic data that hides certain circuit delays. Specifically, the assumption of static full-CMOS logic, along with the use of a fixed-stage decode hierarchy, significantly limits the search for optimal solutions and in many cases, produces inaccurate results.
- The use of a very limited Back-End-Of-Line[1] (BEOL) stack that does not present an accurate representation of nanometer BEOL stacks, resulting in poor modeling of interconnect parasitics. In addition, significant parasitic effects may be ignored by not accounting for the presence of interconnect vias in the circuit.
- The absence of a gate leakage tunneling current model, which may result in significantly underestimating the power dissipation of the cache.

With these limitations in mind, it becomes necessary to fix these first in order to continue using these cache design tools, especially for the design of pipelined, nanometer caches.

1. A BEOL stack refers to all the layers in a process technology fabricated after the active layers. Simply put, it is a description of the interconnect layers in a given process technology.

1.2 Contributions and Significance

The contributions of this dissertation are five-fold. Specifically, these contributions are the following:

1. We provide a very detailed tutorial on prevalent static random-access-memory (SRAM) and cache design techniques that are being used in both the industry and academe.
2. We enumerate the various limitations and inaccuracies of the cache design tools in prevalent use today, mainly CACTI and eCACTI. These limitations and inaccuracies, as well as impractical assumptions and outright program bugs, are discussed in detail.
3. We present a new, vastly enhanced cache design tool that addresses the discussed limitations of both CACTI and eCACTI, and introduces additional enhancements that provide simulation flexibility and ease of use.. We call this new cache design tool myCACTI, which is based on the CACTI and eCACTI infrastructure but introduces radical shifts necessary for nanometer design in both its assumptions and implementation. This dissertation then provides a detailed comparative analysis of myCACTI numbers versus CACTI and eCACTI to demonstrate the inaccuracies that are present with the two existing cache design tools. We show that these inaccuracies are very significant, and could easily lead to false conclusions when these numbers are trusted, especially for cases that may have resulted in hidden impractical implementations during the execution of the program.We describe the major enhancements and improvements of myCACTI over CACTI and eCACTI, namely:
    - The use of SPICE BSIM4.0 equations to accurately characterize device behavior for each possible target process instead of using a single reference process and simply scaling the results to the target process (as done by CACTI and eCACTI). In addition, myCACTI provides the ability to support any BSIM4.0 SPICE deck such that it is not constrained to any single set of process parameters and can accomodate any process technology provided by the user. This way, all device parameters that are used by myCACTI is completely transparent to the user, making it easy to verify whether assumptions still hold, especially in future process technologies.
    - A detailed description of a typical cache pipeline diagram, along with support for explicit pipelining based on the given timing. This allows the designer to explicitly account for the overhead in pipelining a cache (both in terms of delay, latency and power dissipation), resulting in more accurate numbers and at the same time ensuring easy interfacing with external microprocessor circuitry, as I/O external to the cache simply go in and out through explicit pipeline latches
    - Default support for more optimal and more flexible address decoding circuitry. Specifically, the fixed-stage decode hierarchy of CACTI and eCACTI are replaced with a more flexible variable-

stage hierarchy, allowing the simulator tool to accomodate a wider range of loading. In addition, more sophisticated dynamic logic circuits are used to implement these decoders, optimizing both their drive capability and their input load behavior.

- Inclusion of an accurate model and per-process numbers for a typical BEOL-stack that arerepresentative of nanometer processes. The significance of this is made even more important given the tremendous effect of interconnect parasitics on a cache's behavior.
- The computation of transistor gate leakage tunneling currents with the use of SPICE BSIM4.0 equations and inclusion of BSIM4.0-compatible SPICE decks.
- Flexible support for multi-Vt process technologies.
- Support for circuit design techniques that may become prevalent in nanometer cache design (like full-swing single-ended sensing)
- Generation of more detailed delay and power breakdowns which could be used to study implementations in more detail by providing additional relevant information that can be used to justify design tradeoffs.
- More flexible optimization schemes that allow the identification of more than one optimal implementation. For one thing, this allows the designer to generate pareto optimal curves of the design, providing more choices to the designer.
- Various bug fixes to CACTI and eCACTI code.

4. We use myCACTI to provide more accurate insights into different cache configurations. myCACTI is used to provide detailed design space exploration and power and delay break-downs of different nanometer caches, resulting in observations like the folloiwng:
    - The pipeline overhead of a nanometer cache is very significant and in most cases, is one of the dominant contributors to power dissipation. Ignoring this overhead during the design will result in unrealistically optimistic numbers.
    - In deep nanometer nodes, the static power dissipation is very significant, especially in the data bitlines. For single-Vt processes, the static power is typically much more dominant than dynamic power, especially for nodes like 45nm and 32nm, and most of this static power dissipation is due to the bitlines. For dual-Vt processes, the amount of static power is greatly reduced, but is still very significant for the deep nanometer nodes like 45nm and 32nm.

- Gate leakage was found to be surprisingly insignificant, even at deep nanometer nodes. This is mostly due to the fact that the SPICE models we use have been updated to be much less aggressive with regards to oxide scaling, as described by the 2003 and 2005 editions of ITRS.
- Pareto optimal curves have demonstrated that some designs are obviously superior than others, and it would make no sense to implement some designs even if they do seem to be optimal. For instance, we have shown that some points in the pareto optimal curves of a 4-way cache is better in both delay and power dissipation compared to a 1-way cache of the same size. Given also that we expect the 4-way cache to result in better processor IPC, this mandates that some design points in the 1-way pareto curves are not good options, as better implementations are possible without any compromise whatsoever.
- The use of full-swing single-ended sensing becomes more and more practical with each succeeding generation, as its disadvantages decreases while the advantages of low-swing differential sensing also decreases.

5. Finally, we study the reasons why gate leakage tunneling currents do not seem to be a significant contributor to static power dissipation even for deep nanometer technologies like 45nm and 32nm. We see that the slowing down of aggressive scaling in the past few years because of concerns with power dissipation has significantly slowed down the scaling of gate oxide thicknesses, such that we observe that, on a per device basis, gate leakage tunneling currents actually decrease from one generation to the next.

Finally, future directions to the development of myCACTI are identified to show possible ways that the tool can be improved in such a way as to allow even more different kinds of studies to be performed.

All of these contributions are significant, in that it allows the microprocessor and/or cache designer to better characterize the cache behavior, especially that of present and future pipelined nanometer caches.

1.3 Organization of Dissertation

This dissertation aims to provide background on cache design in general, and some cache design tools in particular. A new cache design tool is then demonstrated that provides major additional enhancements compared to existing cache design tools, and the applications of this tool is demonstrated. In this chapter, a brief introduction to the dissertation is given. In chapter 2, a very detailed background on cache design is given to serve as a foundation while discussing the concepts used in cache design tools. In chapter 3, the different cache design tools are discussed. CACTI and eCACTI are discussed in detail. This chapter explores the usage, features and also, the limitations of the two cache design tools. The major enhancements and new features of a

new cache design tool, myCACTI, are then discussed in relation to the existing cache design tools. In chapter 4, detailed comparisons are performed to show the difference in results between CACTI/eCACTI and myCACTI, showing that once the major limitations of CACTI and eCACTI are addressed, the numbers generated by simulation will be significantly different from what they presently would be. This chapter discusses in detail all the different isolated comparative studies that are performed. In addition, the general framework of the comparison is also discussed. In chapter 5, the applications of myCACTI are then demonstrated, starting with detailed design space explorations of a slew of different cache configurations ranging from caches with sizes of 8kB, 16kB, 32kB, 64kB and 128kB; associativities of 1-way, 2-way, 4-way, 8-way and 16-way; and technology nodes of 130nm, 90nm, 65nm, 45nm and 32nm. The output of this study is post-processed to produce pareto optimal plots of the optimal implementations for each cache configuration based on the two criteria of cache read-hit power dissipation and cache clock period. After this general design space exploration is demonstrated, a second set of studies are then performed to show the detailed power dissipation and delay breakdown that myCACTI can provide, and how they can be used by cache designers to identify candidate implementations for further optimization. The study focuses on 64kB 4-way caches and produces detailed data on three of the pareto optimal implementations for each cache configuration. After details of the representative 64kB 4-way cache have been shown, a third set of studies isthen performed, this time to study the effect of three different cache design techniques (mainly single/dual Vt transistors, static/ dynamic decoding and full-swing single-ended/low-swing differential sense amplification) on a 64kB 4-way 32nm cache. During these studies, one of the more important observations is that, contrary to the expectations of the research community, the effects of gate leakage tunneling current with respect to static power dissipation is not significant, even in the very deep nanometer technology nodes like 32nm. Chapter 6 studies this in detail, and shows that precisely because gate leakage tunneling currents are expected to become an even bigger problem than subthreshold leakage currents, the scaling of the gate oxide thickness has been significantly slowed down, and the SPICE models that have been used for the simulations already reflect this. Chapter 7 summarizes this dissertation with concluding remarks.

CHAPTER 2 Background

2.1 Power Dissipation in CMOS Circuits

CMOS circuits were designed to dissipate power only during transitions in state. Once the internal node voltages of a logic gate have been charged or discharged and have reached their steady state, there should exist a very high impedance in between supply and ground such that they are essentially open-circuited.

The dynamic power associated with switching a CMOS gate can be given by:

$$P_{dynamic} = (N \cdot C \cdot V_{DD}^2 \cdot f) + (I_{scc} \cdot V_{DD})$$

The first term is usually the more important because of its omnipresence -- any circuit will always have some internal parasitic capacitance, no matter how small, that has to be charged or discharged in order to change its state and the first term in the equation accounts for the power involved in this capacitive switching process. The second term is due to the so-called short-circuit current, or crowbar current, that exists when the PMOS and NMOS networks are both momentarily active at the same time, presenting a low-impedance path from supply to ground. Although this term may sometimes be significant, careful design of the circuits to ensure reasonable switching edge rates will minimize the time window where the low-impedance Vdd to Vss shorts exist[1], minimizing the contribution of this crowbar current.

From the equation for dynamic power, many solutions suggest themselves in order to reduce the total power. Reducing any of the four terms (number of devices, device capacitance, supply voltage or system frequency) will result in reduced power consumption. The challenge then becomes the minimization of the negative effects involved in changing these parameters when trying to reduce power.

Static power, on the other hand, is generally expressed as:

$$P_{static} = I_{leak} \cdot V_{DD}$$

This equation simply lumps all the leakage existing in the circuit and uses it in the conventional equation for power.

Power consumption, until recently, has not been a primary concern in designs that were not targeted for specific low-power applications (e.g. portable devices), but with the continuous down-scaling of

1. Fast edge rates ensure that the inputs to both the NMOS and PMOS do not linger in an undefined region where it exceeds the threshold of both transistors resulting in a conducting channel where crowbar currents can flow.

technology, the static power dissipation has become increasingly larger. Transistor device channels that were previously considered open circuits are now exhibiting current leakage. Some device structures have become so thin that even though they were designed to be insulators, the very small dimensions involved allow quantum tunneling phenomena to result in significant current flow. Static power dissipation has become a very big problem, and it has greatly contributed to making power consumption a first-order concern even in designs that previously had minimal concern for power at all (e.g. desktop computers).

2.2 Leakage Current Mechanisms

In this section, we provide a brief overview of the leakage mechanisms that are prevalent in a modern MOS transistor. Anis [Anis2003] gives a good overview of these leakage currents, as shown in figure 2-1, and his explanation of these different mechanisms are as follows:

I_1 is the reverse bias pn junction leakage. It has two main components: the minority carrier diffusion/ drift near the edge of the depletion region, and the electron-hole generation in the depletion region of the reverse bias junction.

I_2 is the weak inversion current or subthreshold conduction current between the source and drain in a MOS transistor; I_2 occurs when the gate voltage is below Vt. the carriers move by diffusion along the surface, and the exponential relation between the driving voltage on the gate and the drain current is a straight line in a semi-log plot.

I_3 represents Drain-Induced Barrier Lowering (DIBL). It occurs when the depletion region of the drain interacts with the source near the channel surface to lower the source potential barrier. The source then injects carriers into the channel surface without the gate playing a role.

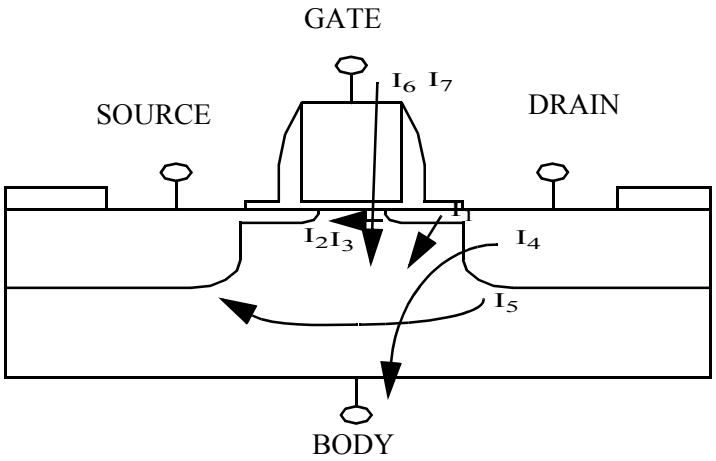


Figure 2-1: Typical leakage current mechanisms in a modern MOSFET.

I_4 refers to the Gate-Induced Drain Leakage (GIDL). GIDL current arises in the high electric field under the gate/drain overlap region, causing a deep depletion and effectively thinning out the depletion width of the drain to the well pn junction. Carriers are generated into the substrate and drain from the direct band-to-band tunneling, trap-assisted tunneling, or a combination of thermal emission and tunneling. the thinner oxide thickness Tox and higher Vdd causes a higher potential between the gate and drain which enhances the electric-field dependent GIDL.

I_5 is the channel punch-through which occurs when the drain and source depletion regions approach each other and electrically touch deep in the channel. Punch-through is a space-charge condition that allows the channel current to exist deep in the sub-gate region, causing the gate to lose control of the sub-gate channel region.

I_6 represents the oxide leakage tunneling. The gate oxide tunneling current Iox which is a function of the electric field Eox can cause direct tunneling through the gate.

I_7 is the gate current due to hot carrier injection. Short-channel transistors are more susceptible to the injection of hot carriers (holes and electrons) into the oxide. These charges are a reliability risk and are measurable as gate and substrate currents.

Typically, I_2 and I_3 dominate the off leakage current[1], and I_6 dominates the gate leakage. A formula for I_{off} [Chandrakasan1996] is the following:

$$I_{off} = \frac{I_0}{W_0} \cdot W \times 10^{-\frac{(V_{GS} - V_t)}{S}}$$

Where Io/Wo is the saturation current per unit width, and S is the subthreshold slope. For devices with zero gate bias, it can be seen that gate leakage is an exponential function of threshold voltage. For a typical technology with a subthreshold slope of 100mV/decade, each 100mV increase in Vth results in an order of magnitude larger current.

In previous generations, this I_{off} has not been a significant problem because of the larger values of Vt involved. But as transistor sizes are scaled down, it often becomes necessary to scale down Vdd correspondingly (i.e. in a constant electric field scaling scheme). Logic gate speeds, as measured by the ability to charge and discharge internal capacitances, are dependent on the gate overdrive voltage Vgs-Vt (where Vgs

1. The off leakage current is typically defined as the device drain-to-source current (Ids) with the gate off (i.e., strong inversion has not been reached such that no conducting channel exists between the drain and source terminals. Often, this off current has been associated and is interchanged with the subthreshold current, simply because subthreshold is often the main, dominant factor.

is most often Vdd when provided by another CMOS gate). Decreasing Vdd tends to decrease the speed of the system (even with a decrease in capacitance associated with device scaling) and this is traditionally compensated for by decreasing the threshold voltage to improve the gate overdrive, resulting in an improved current driving capability of the device. Continuing device scaling results in more and more reduction in Vt such that the device off current starts becoming significant, especially when multiplied by the large number of total devices in the entire chip.

Additionally, even in the presence of scaling, a certain amount of device gate to body capacitance needs to be maintained in order to retain control of the inversion channel. With the same dielectric material, this requires reduction of the oxide layer thickness, tox. But as tox goes down below 20 angstroms with continued scaling, the corresponding oxide layer becomes so thin as to allow quantum tunneling effects to start being significant. As such, gate leakage effects are starting to become important, with some sources stating that the rate of increase of gate leakage compared to subthreshold leakage per technology generation is much faster, and gate leakage current may soon become the most dominant mechanism.

The equation for gate leakage current is more complicated, as given by Clark [Clark2004]. Instead, most techniques that provide solutions for gate leakage instead rely on the characterization of Rao [Rao2003] describing the behavior of gate leakage with respect to its gate-source and drain-source voltage, as shown in figure 2-2. They conclude that for a given gate bias, gate leakage is minimum if the gate-drain voltage (Vgd) is minimum. To minimize gate leakage, we should strive to minimize gate bias (Vgs) and if this is not possible, minimize the gate-drain bias (Vgd).

As a summary, subthreshold leakage are often solved using techniques that increase its threshold voltage, either statically or dynamically. In addition, DIBL effects that contribute to the leakage can be reduced by lowering the drain-source voltage across an off transistor. Gate leakage currents, on the other hand,

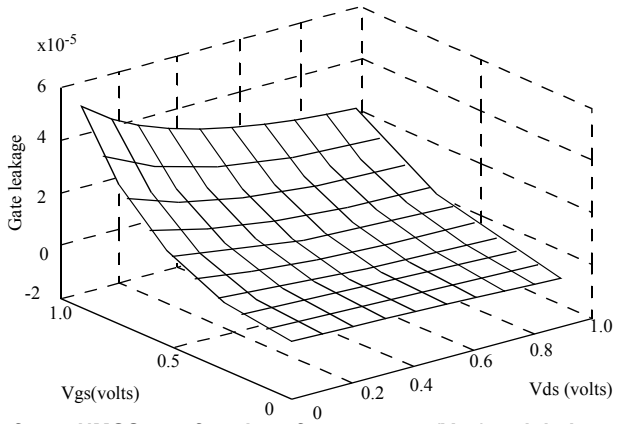


Figure 2-2: Gate leakage current for an NMOS as a function of gate-source (Vgs) and drain-source (Vds) voltage (figure from Rao2003). It shows the highest gate leakage when Vgs is max and Vds is zero (making Vgd max also).

are minimized primarily by reducing the gate bias (Vgs) and secondarily, by minimizing the gate-drain bias (Vgd).

2.3 SRAM and Cache implementation

Caches have become an integral part of the operation of modern general-purpose microprocessors. Its importance has become so paramount that its share of the total transistor budget has steadily increased to the point that almost half of microprocessor's die area is allocated to caches. With its increasing total number of devices, caches are becoming a bigger and bigger contributor to a design's static power dissipation Consequently, it is of utmost importance to develop sound, effective strategies that lower the static power consumption of memory circuits, especially since this will result in a reduction of power consumption during both active and idle modes of the processor. It is therefore necessary to have a detailed knowledge of the design intricacies that are inherent to memory circuits and in particular, SRAMs in order to understand which parts of memory circuits can be modified to reduce leakage currents.

To start things off, we discuss a sample cache operation, a cache read hit, in moderate detail, exposing some of the implementation issues involved in its design.

Figure 2-3 shows an example cache organization: a 2-way set associative cache with virtual addressing, along with a timing diagram showing the various events happening in the cache (to be discussed in much more detail in later sections).

Basically, the following steps involve:

1. Providing address to the cache, along with an address strobe signal (ADS) confirming the validity of the address. A read/write signal (R/W#) is also sent to specify the operation.
2. The index part of the address chooses a word within the tag and data arrays of the 2-way set associative cache (signified by the wordline signal, WL). This in turn causes the internal bitlines to develop a differential, which is amplified by sense amplifiers to produce a full-swing differential output voltage.
3. The translated address from the TLB is compared with the output of the tag array to decided if the cache access hit or miss. In case of a hit, the proper data is chosen among the two ways by controlling the output multiplexer and forwarded. A cache miss requires the cache controller to perform a separate operation to retrieve data from external memory (or another level of cache) and perform a write access to the cache.

We have now demonstrated the basic cache read operation and shown some of the blocks used in implementing a cache. In the next sections, we will discuss much more in-depth details, starting with the

implementation of the basic storage structures comprising the tag and data array, then proceeding to how a cache is implemented and how these data arrays are used. Along the way, we also discuss advanced topics that are related to contemporary cache issues like low-leakage operation.

SRAM Implementation

In this section, we discuss the implementation of a static random access memory (SRAM). This is the type of memory used as the building block of most caches because of its superior performance over other memory structures, specifically DRAM. Along with the fact that it uses the same fabrication process as the CPU core, while high-performance DRAM uses a different process that is suboptimal for logic circuits, making it less feasible and less attractive to integrate DRAM-based memory and the processor in one chip.

Basic 1-bit Memory Cell. This subsection describes how the most basic unit of an SRAM -- a single data bit -- is implemented. For SRAMs, the memory cell is implemented as two cross-coupled inverters accessed using two pass transistors. This topology and its most typical contemporary physical implementation are shown in figure 2-4, along with simple timing diagrams showing the READ and WRITE accesses.

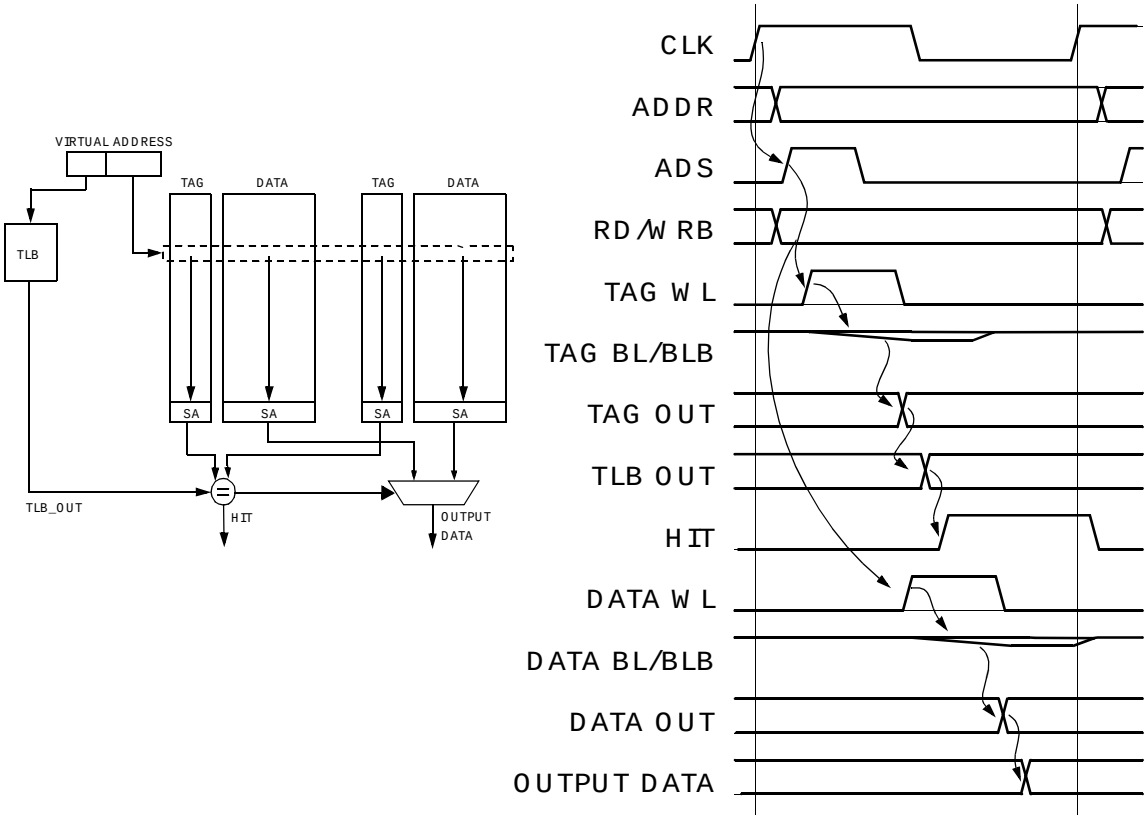


Figure 2-3: Block diagram of a 2-way set associative cache organization, along with the timing diagram showing both internal and external signals for a read operation.

This cross-coupled connection creates regenerative feedback that allows it to indefinitely store a single bit of data. This configuration shows a 1-bit MC with one R/W port that can be used for either a read or a write, but not simulataneously As shown in the figure, the bit is read by asserting WL and detecting the voltage differential between the bitline pair, which are initially precharged to a logic high voltage..The second phase shown in the is a write operation. The bitline is driven with a differential voltage from an external source to force the data onto the memory cell.

One of the assumption of caches, and memory in general, is the ability to store digital data, and we have shown how a single bit can be stored by an SRAM using cross-coupled inverters. This cross-coupled inverter has been implemented in different ways during its evolution. Although we only show the particular implementation that is currently the most practical, numerous implementations have existed over the years, where the main difference has been how the inverter's pullup network is implemented.

Early SRAMs typically used either the full-CMOS or the polysilicon load memory cell configuration. The main advantage then of the poly-load configuration was its smaller area since only four transistors were required for one bit; the pullup was implemented using an additional highly-resistive polysilicon layer such that the load is fabricated on top of the existing transistors, requiring less area. With decrease in feature sizes, the amount of area occupied by the poly-load needed to produce a large enough resistance to overcome leakage currents became too big. Along with its low SER [List86], the poly-load implementation has now become impractical. With the increase in space occupied by the poly-load, it started to be replaced by the poly-PMOS or thin-film-transistor (TFT) PMOS [Minato1987] load configurations, which could still be fabricated on top of the active NMOS but with characteristics superior to the poly-load. With the increasing on-chip integration of caches with microprocessor logic circuits, the additional process complexity required to

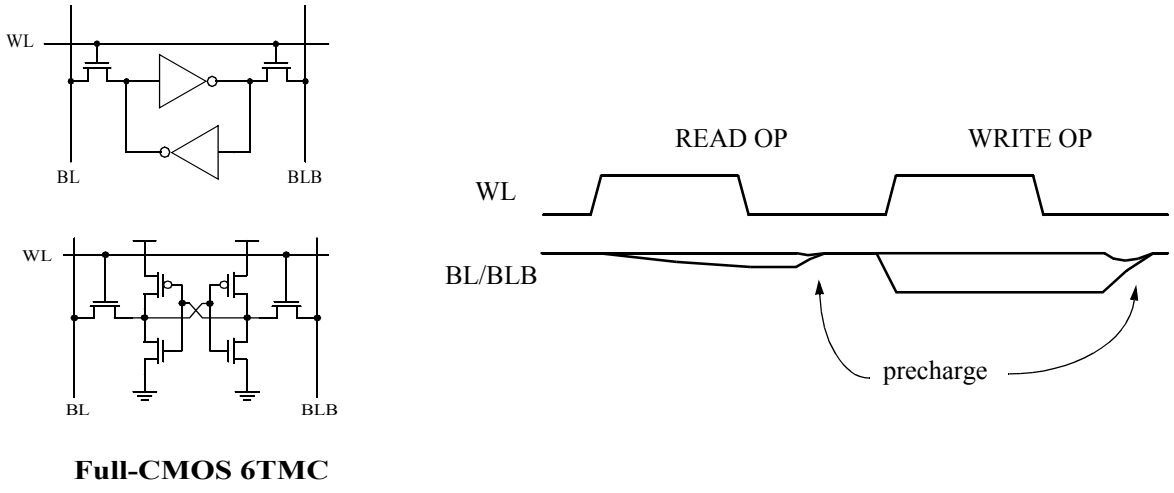


Figure 2-4: The basic SRAM cell showing cross-coupled inverters accessed through pass transistors. The state of important signals are also shown during a read and write access. Also shown is the six-transistor implementation of the memory cell (6TMC).

implement the poly-PMOS/TFT-PMOS circuits on the same die as the digital circuits became too cumbersome. An alternative to these circuits, the loadless four transistor cell (LL4T), completely removes the pullup load [Noda1998]. Although this scheme results in a smaller area compared to the full-CMOS implementation because only four NMOS transistors are involved, the cell has to be very carefully designed to make sure that leakage currents do not interfere with the latching operation of the cell and its ability to retain data. With continuous technology scaling, the charge stored within the cell decreases, while leakage current increases -- making it more difficult to design reliable LL4T cells, especially if external radiation from alpha particles are taken into account.

Currently, most conventional designs use the full-CMOS six transistor memory cell (6T MC), with different variations existing based on issues of sizing, physical layout, and transistor threshold voltages for low power (to be discussed in later sections).

The rest of the discussion will be limited only to the 6T MC variant, and an example layout using MOSIS SCMOS rules for a 0.25μm process is shown in Figure 2-5, along with its transistor-level circuit defining the access, driver, and pullup transistors.

Address Decoding. Address decoding is a conceptually simple process of receiving address information and providing signals to initiate and perform the desired operation on the addressed location. At its simplest, it involves feeding an address value into a binary decoder (n to 2^n) and using the asserted output to activate the

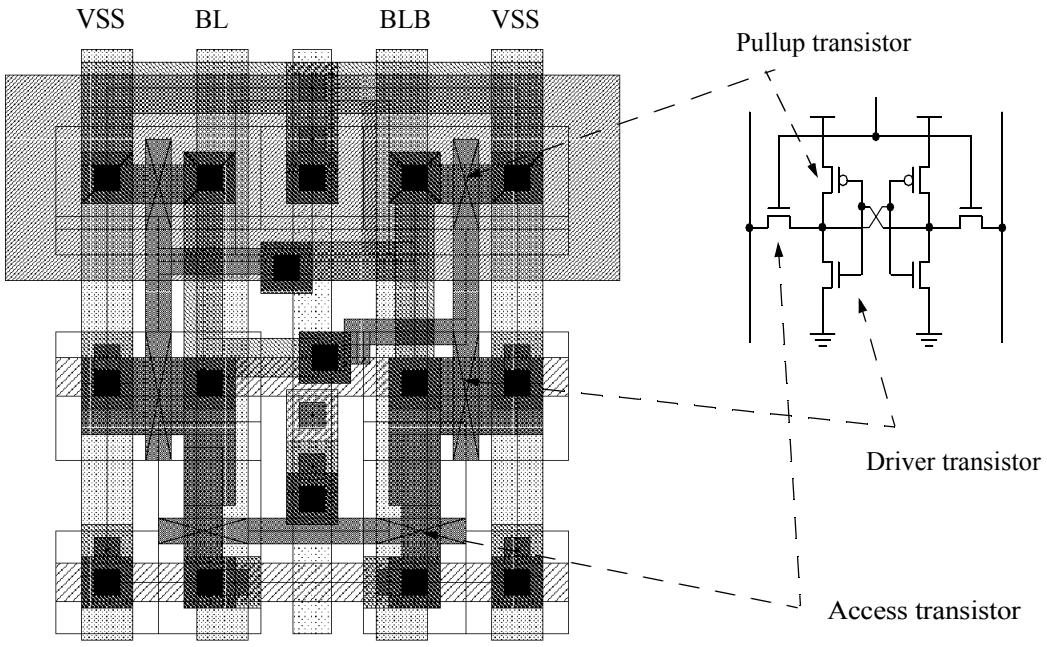


Figure 2-5: Layout of a six transistor memory cell (6T MC) using MOSIS SCMOS rules (SCMOS_SUBM using TSMC0.25μm). Also shown is the definition of the access, driver and pullup transistors.

wordline of a subset of memory cells associated with this address. This involves a single AND operation on the input address, with the output connected to memory cell wordlines, as shown in Figure 2-6.

The main concern in address decoding is the large fan-in and fan-out requirements of typical memories because of the number of address bits being decoded and the large amount of cells that have to be driven. This makes the simple one-level AND structure inefficient, and virtually all SRAM designs implement a multi-level decode hierarchy to implement the logic AND function.

In typical designs, the address decoder contributes significantly to the critical path delay and total power consumption, emphasizing the need to optimize the memory array's decode hierarchy implementation

Predecoding. .One of the main usage of a decode hierarchy is to minimize the fan-in of parts of the decode circuit because higher fan-in gates have a large logical effort [Sutherland1991 and Sutherland1999] making them less efficient. Simply put, logical effort expresses how harder it is to drive an arbitrary gate compared to an inverter of the same drive strength. High fan-in static gates typically have high logical efforts and are less efficient to use.

One of the techniques used in minimizing fan-in is a method called predecoding. Predecoding involves using one level of logic to effectively AND subsets of the address. The outputs of these predecoders are then combined by low fan-in gates to produce the final decoder outputs, which are the wordline signals connected to the memory cells. In this way, predecoding simply involves performing the AND operation using multiple levels of logic.

Consider an example where an 8-bit address is to be decoded. A simplistic approach using 256 (2^8) 8-input AND gates are used, as shown in Figure 2-7(a). Although logically correct, these ANDs will have

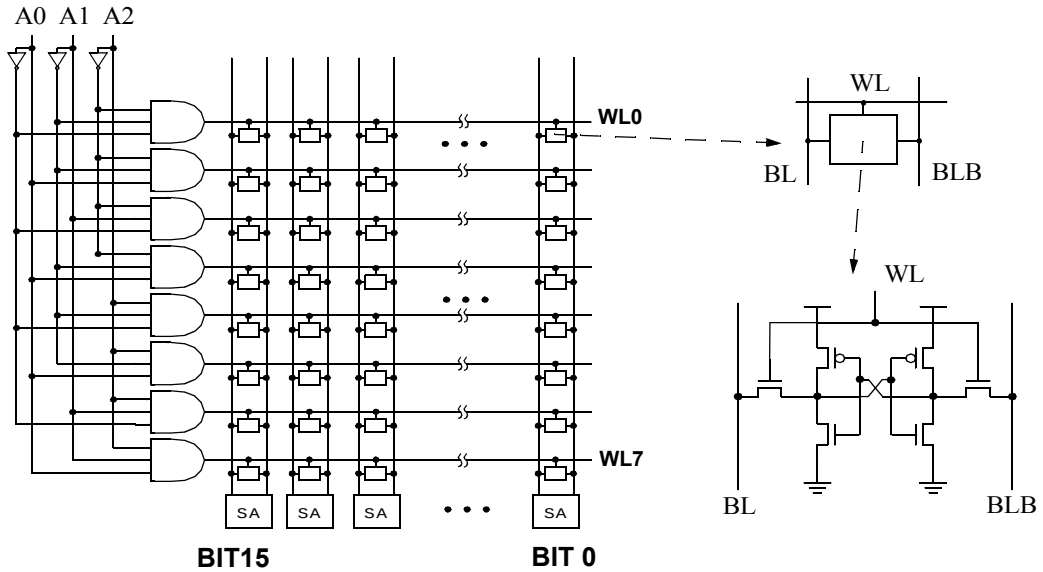


Figure 2-6: Address decoding shown for a simplistic 8 x 16 bit memory showing 8 3-input AND gates that enable th wordlines of a specific number of cells (16 for one wordline). Also shown are the sense amplifiers that generate logic outputs from the small voltages produced by the cells, and is the internals of a single cell.

significantly higher gate capacitances compared to 2-input ANDs, resulting in larger delays, higher power consumption, and large area.

An alternative decoder implementation using predecoding is shown in Figure 2-7(b). The 8-bit address is divided into two subsets, each with 4 bits. For both subsets, sixteen 4-input ANDs are used to generate all possible combinations of the 4-bit address. The final wordlines are generated using 2-input AND gates to combine one output each from the two subset. In this case, 256 2-input AND gates are used to generate all the 256 wordlines.

Although conceptually simple, predecoding has numerous advantages. Since the final AND gates only have two inputs, they will have significantly less gate capacitance compared to the first implementation. This results in a smaller, faster, lower power circuit that is also more scalable than the original. Although some of the area advantage is offset by requiring the initial high fan-in ANDs, this approach is overall still much better. One main reason that is not immediately obvious is that the possible implementation of the predecoders is more flexible since it can be separated from the memory arrays, and will have less restrictions imposed by the memory array dimensions

With this flexibility, the predecoders can be implemented using sophisticated circuit designs that enable circuits with faster speed, lower power and smaller area. Some of these techniques will be covered in more detail in a later subsection..

The application of these advanced circuit techniques to the first approach is not feasible because of the higher cost involved in applying it to a much larger number of gates (In the example, 256 gates for the simplistic approach and only 32 for the predecoder approach). At the same time, the implementation

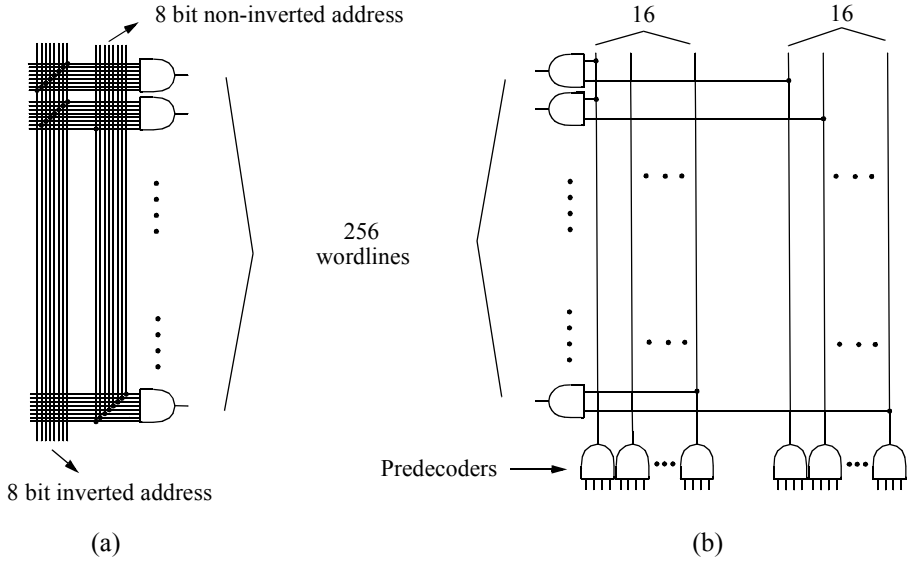


Figure 2-7: Sample wordline decoders for an 8-bit address. (a) A simplistic high fan-in AND approach. (b) Decod with predecoding.

possibilities for gates embedded within the regular structure of the memory array is much more limited since it is affected by other factors including the cell to cell spacing, or pitch, of the memory arrays, as shown in Figure 2-8.

The main disadvantage of predecoding is the need to distribute more wires to propagate all the intermediate predecoder outputs. This is a minor issue, and the advantages of predecoding makes it almost necessary in most SRAM designs.

Row and Column Decoding. For added flexibility in operation, memory arrays like SRAMs (and in later chapters, DRAMs), typically employ two dimensional decoding, where a subset of the address accesses a single row of the array (the row address), and a separate subset is used to select a fraction of all of the columns accessed within the row, as shown in the sidebar.

In figure 2-9, predecoding for both the row and column addresses are not shown for simplicity, and it is assumed that the row and/or column decoder utilize these predecoder outputs. The multiplexer allows

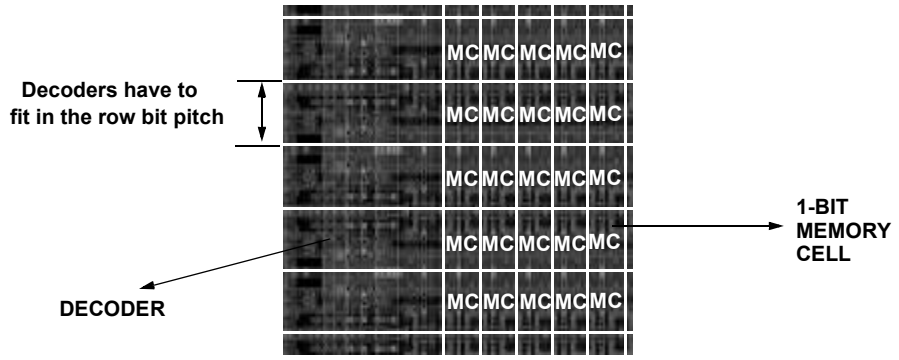


Figure 2-8: Sample floorplan of the decoders beside the memory array showing the bit pitch limitation.

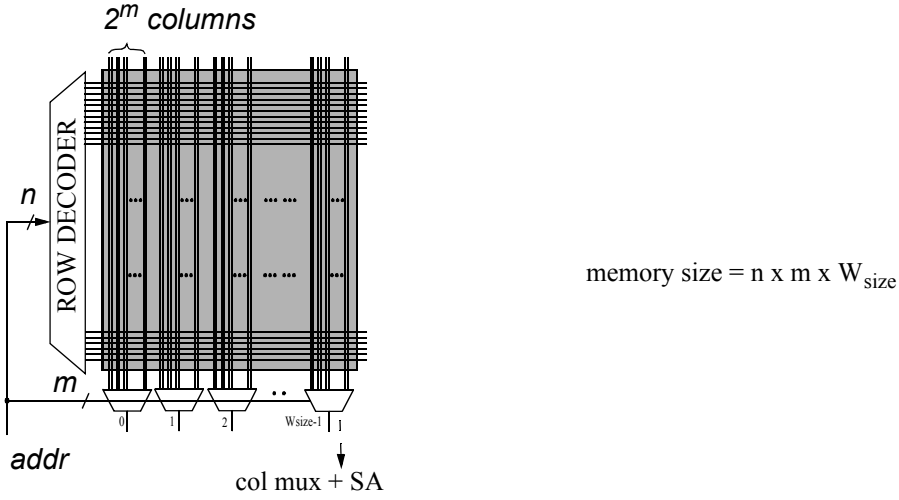


Figure 2-9: Row and column decoders for a simple SRAM.

multiple bitlines to share a single sense amplifier, saving both power and area. These multiplexers are almost always of the pass transistor variety, allowing it multiplex small-swing voltages in the bitlines. To simplify the figure, only the data read path is shown since address decoding for read and write operations are essentially the same.

Non-partitioned. A non-partitioned decode hierarchy refers to a memory organization where all cells in a given row are activated by a single word line output from the row decoder, and the organization of the sample memory system in the previous subsection, as well as the figure in the previous sidebar, are examples of this.

For very small SRAMs (i.e. 1kByte and smaller), this simple non-partitioned approach is sufficient. But as the size of the SRAM increases, several problems start becoming significant:

1. As the number of memory cells in an SRAM increases with increasing memory size, more and more transistors from each memory cell are connected to the row's wordlines and the column's bitlines, increasing the total capacitance resulting in an increase in delay and power consumption.
2. Increasing the number of memory cells result in a physical lengthening of the SRAM array, increasing the wordline wire length and its parasitic wiring capacitance.
3. More power is in the bitlines is wasted during each access because more and more columns are activated by a single wordline even though only a subset of these columns are actually accessed. In a non-partitioned scheme, every column will have an active memory cell, inadvertently discharging the bitlines of the unaccessed columns resulting in wasted power needed to precharge these bitlines back to their original value.

To a certain extent this problem can be mitigated by minimizing the number of columns in an array. For a fixed memory size, this can be done by increasing the number of rows accordingly. This solution is obviously very short-sighted as it will eventually produce its own problems and doesn't solve the original one. Instead, the number of columns (and rows) is used as an additional parameter that can be changed to come up with an optimal memory partitioning and hierarchy, as will be discussed later.

Divided Wordline (DWL). To solve the problems of the non-partitioned approach, the divided word line (DWL) technique was introduced by Yoshimoto [Yoshimoto1983] and is now used in virtually all SRAMs because of its usefulness and minimal disadvantages.

DWL involves dividing the memory array along the top level word line (called the global wordline or GWL) into a fixed number of blocks. Instead of enabling all the cells within a row, the global word line is ANDed with a block select signal (derived from another subset of the input address) and asserts a local wordline (LWL). Only cells connected to this asserted local WL are enabled.

This DWL structure is shown in Figure 2-10 where the memory array is divided into n_B blocks. Assuming a total of n_C columns, each block contains n_C/n_B columns. Since only a fraction of the total cells ($1/n_B$ to be exact) are connected to an asserted local wordline and consequently activated during an access, the total power consumed in the bitlines is reduced drastically to roughly $1/n_B$ of the original. This value is exactly $1/n_B$ for reads since the nature of wasted power is similar to the power consumed by the bitline read access, while the savings is less for a write access because of the higher power consumed by accessed bitlines during a write.

Additionally, since less cells are connected to the local wordlines, the power and delay will be significantly less. The global wordline also only needs to drive the wire capacitance and the much smaller number of local wordline decoders. The total wordline delay (the sum of the GWL and LWL delays), for the DWL, compared to the non-partitioned implementation, will be much less for moderately sized SRAMs and above.

Figure 2-11 shows the effect in column power and wordline delay of changing the number of blocks for an 8k x 8 SRAM. Although this may seem like a small SRAM by today's standards, it is still useful as a basic building block for wide caches. For example, a 128-bit cache can employ sixteen of these small SRAMs resulting in a 128kB cache, which is getting close to the typical size of an L2 cache. The figure shows that even though column power can be continuously reduced by increasing the number of blocks, the benefit to the wordline delay lessens and it reaches a point where it starts to adversely affect the delay.

The implementation of the DWL technique is simple, requiring only additional block select circuits and LWL decoders that can be made simple since they drive less cells. For the 8k x 8 SRAM here, dividing the array into 8 blocks ($n_B = 8$) results in only a 4-6% increase in area while resulting in very significant power (near 800%) and speed (about 300%) gain.

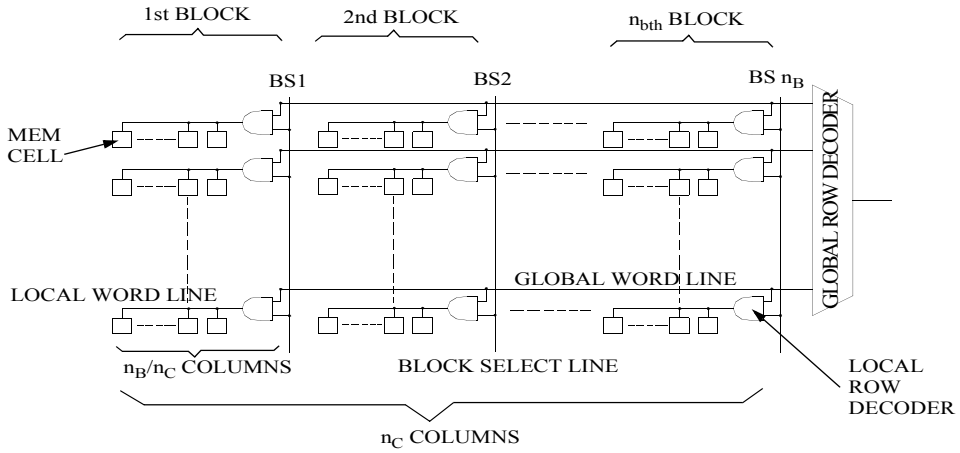


Figure 2-10: An SRAM using the divided word line (DWL) address decoding technique. (Figure taken from Yoshimoto1983).

Hierarchical Word Decoding (HWD). A logical extension of the DWL scheme is a technique called hierarchical word decoding (HWD) [Hirose1990], and is shown in Figure2-12.

As the memory size increases, maintaining a DWL structure requires an increase in the number of blocks, which results in an increase in the GWL capacitance since more wordline decoders tap into the GWL. HWD simply introduces additional levels of hierarchy into the decoding scheme to more efficiently distribute capacitances, with the number of levels determined by the total load capacitance of the word decoding path. At 256kB, the difference between DWL and HWD is insignificant, but a 4Mb SRAM using the HWD architecture can reduce delay time by 20% and total load capacitance by 30%.

Pulsed Word Line . Early SRAM implementations asserted the wordlines for a significant fraction of the cycle time. The wordlines are typically asserted early (after the decode delay) and deasserted late in the access. This method, while functionally correct, is inefficient. [Amrutur1994] For a read access, wordline assertion causes one of the bitline pair to be pulled down, creating a voltage differential across the bitlines. When a sufficient differential exists (the exact value will depend on the process technology and the offset voltage of the sense amplifier), a sense amp is used to amplify this differential and speed up sensing. At this point, any

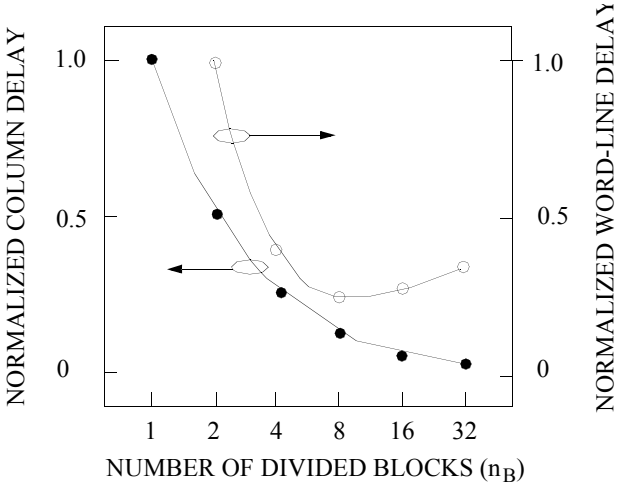


Figure 2-11: Graph showing effect of the number of blocks, nB, in the memory's column power and word line delay (Graph taken from Yoshimoto1993).

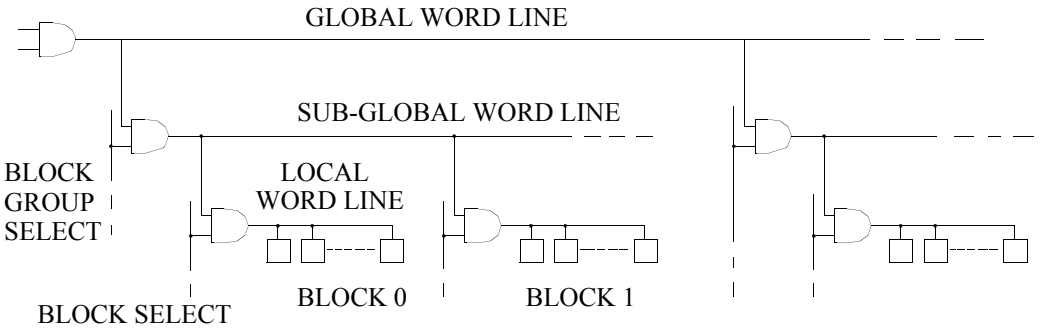


Figure 2-12: Hiearchical word decoding (HWD) architecture showing three levels of decoding. (Picture taken from Hirose1990].

additional differential developed across the bitlines, as a result of the continued assertion of the wordline, will not significantly speed up sensing and will require more power and precharge time.

Most SRAMs now use some kind of pulsed word line, where the wordline is allowed to assert only for a small amount of time necessary to create a sufficient bitline differential voltage after which it is turned off. This technique prevents the development of bitline voltage differential more than is necessary, reducing the power consumed during the precharge process.

The width of the pulsed wordline is controlled either using static delays (where the wordline is turned off after a certain number of delays fixed at design time or fixed by special delay circuitry during built-in self-test), or using some feedback taken from information extracted from the circuits (to be discussed in detail later).

Figure 2-13 shows timing diagrams for a system with and without the pulsed-wordline scheme. As can be seen from the diagram, output data from both systems are produced almost at the same time, but continued assertion of the wordline for non-PWL schemes result in larger differentials developed across the bitline, serving only to consume additional power during precharge.

Physical Decoder Implementation. With the division of the decoder into multiple levels of hierarchy (i.e. predecoders, global wordline decoders, local wordline decoders, column decoders, block decoders, etc.), different circuit styles and implementations can be used for each decode level depending on different factors including power, speed, area, complexity and layout requirements, among others.

The relative importance of these specifications vary within the decoder hierarchy because of the presence of different constraints. As a quick example, predecoders can use relatively power-hungry circuit styles, which is not true of the local wordline decoders simply because of the much larger number of LWL decoders in the circuit compared to predecoders.

Numerous research have been done to optimize the decoders in the entire hierarchy from the initial predecoder to the final LWL decoder. [Yoshimoto1983, Yamamoto1985, Sasaki1988, Aizaki1990,

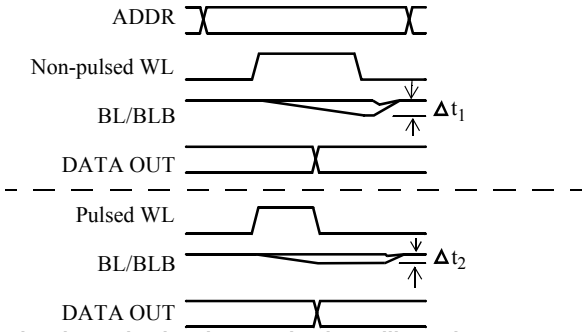


Figure 2-13: Timing diagrams showing pulsed and non-pulsed wordline schemes.

Nakamura1997, Mai1998, Nambu1998, Osada2001]. Because of space limitations, only a small (but very relevant and proven optimal) subset of these circuits will be discussed here in detail.

Digital Logic Styles. Before proceeding, it is beneficial to discuss different logic circuit styles and where in the decode hierarchy each style is suited to. CMOS logic circuits can be generalized by the circuit shown in figure 2-14. The figure shows an output node connected to Vdd by a pullup network, and to ground by a pulldown network, where both networks are controlled by inputs (e.g. data and/or a clock signal).

The pullup network consists of a combination of PMOS transistors that conditionally connects the output to Vdd, while the pulldown network consists of a combination of NMOS transistors that condionally connects the output node to ground. Although the inputs and the clock signal are shown to be provided to both the pullup and pulldown networks, they may or may not be utilized depending on the specific implementation.

Static CMOS. The first logic style we review are static CMOS or full-CMOS circuits, i.e. at steady state, there will always exist a relatively low-impedance path from the output to either Vdd or ground depending on the logic output. This distinction will be clearer when dynamic CMOS is discussed, where we will see that some logic values depend on the charge stored within parasitic capacitances in the gates themselves.

Static CMOS is the easiest and most robust implementation since there are a lot less variations and problems associated with the design. The two variations of the static CMOS style are the static or active load and the full-CMOS styles. 3-input NAND and NOR gates for both styles are shown in Figure 2-15. The main difference of the two styles is the implementation of the pullup net (notice that the pulldown networks are exactly the same). The full-CMOS implementation utilizes a pullup net that is the complement of the pulldown (i.e. parallel connections become series, and vice versa) while the active-load uses an always-on PMOS transistor pulling up the output constantly to Vdd.

The main results of this difference are the following:

1. Typically smaller area for the active load because of less transistors.
2. The total active-load gate capacitance for each input is typically less than half of the full-CMOS implementation. This results in a smaller logical effort, making this gate easier to drive.

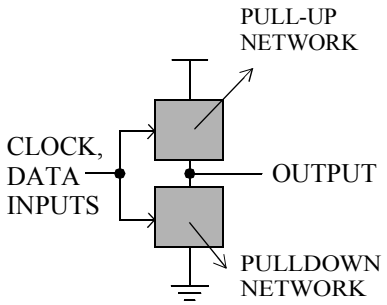


Figure 2-14: Generalized logic circuit.

3. The always-on PMOS will result in significant static power consumption whenever the pulldown net is enabled. In addition, the logic low voltage will not reach the full Vdd value because of the pulling up effect of the PMOS and how it fights the pulldown network. The exact output value will depend on the relative strengths of the transistors..

Speed and power comparisons of these circuits are interesting [Sasaki1988]. Given equal capacitance characteristics, the PMOS-load circuit will produce about 8% more delay than the full-CMOS decoder (because the PMOS-load will tend to fight the pulldown net, producing delay). But the capacitance characteristics of these circuits in practice will usually be different and will tend to make the PMOS-load faster (about 15% faster). In addition, average currents in their decoder (which directly relates to power consumption) show that even though the PMOS-load consumes DC current, there exists a crossover point where cycle times becomes smaller and AC current becomes more dominant. It must be stressed, though, that these current numbers are greatly influenced by the sequencing and control of the gates.

Because of these factors, the speed comparison will be releveant regardless of how the memory is controlled, while the power comparison needs to be studied on a case to case basis to take into account the specific characteristics of a system. For example, a PMOS-load used with pulsed-wordline technique will be active for a smaller amount of time than if a non-PWL technique was used. This makes active-load circuits feasible for use in LWL decoders using PWL since a gain in speed is achieved compared to a full-CMOS system while at the same time not wasting too much power (especially since only a very small fraction of LWL decoders are active at the same time).

Dynamic CMOS. One of the main objectives of dynamic CMOS is to minimize the capacitance of the logic gate inputs. At its simplest, this is done by implementing only either the pulldown or pullup network. Further discussions will involve only dynamic CMOS with pulldown network, as the use of pullup networks are useful only in special applications because of the better characteristics of NMOS transistors compared to their PMOS counterparts.

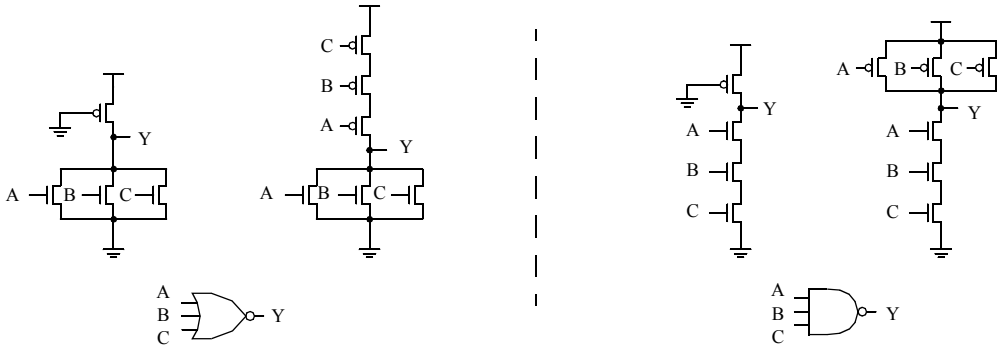


Figure 2-15: Active-load and full-CMOS circuit styles for 3-input NOR and NAND gates.

Dynamic CMOS is similar to the circuit of the PMOS active-load gate. The main difference is instead of having always-on loads, dynamic CMOS use clocked elements that precharge (in this case, charge up to Vdd) the output node during the precharge phase. This output node is later conditionally discharged during the evaluate phase depending on the state of the inputs.

Figure 2-16 shows simple dynamic implementations of 3-input NAND and NOR gates, along with a timing diagram showing the precharge and evaluate phases of the dynamic gate operation.

Note the presence of the additional clocked NMOS in the pulldown net, called "foot" transistors. During the precharge phase, the PMOS precharge transistor precharges the output node Y to a logic high voltage. The precharge NMOS is not enabled to prevent the possible inadvertent early discharge of the output and to prevent forming a low-impedance path from Vdd to ground. The evaluate phase is started when ϕ asserts, disabling the PMOS and enabling the NMOS input transistors. At this point, the output node dynamically stores charge that is conditionally discharged by the pulldown network if the right combination of input signals exist.

It is important to emphasize that inputs to dynamic gates must be carefully designed not to inadvertently discharge the output node. In general, we require the inputs to a dynamic gate be valid before the evaluate phase or to change value monotonically to prevent an inadvertent discharge of the dynamic output node.

Domino Dynamic Logic. In general, individual dynamic logic gates are implemented with buffers at the output to provide more robust driving capability and to protect the dynamic node from being corrupted by noise. These buffers are usually implemented using full-CMOS inverters. This gives rise to a style called domino logic, and Figure 2-17 shows a circuit where domino gates are cascaded together.

During the precharge phase, all internal dynamic nodes are precharged high, causing Y1, Y2, and Y3 to go low (the inverted version of the dynamic nodes). At the start of the evaluate phase ϕ=1, and for

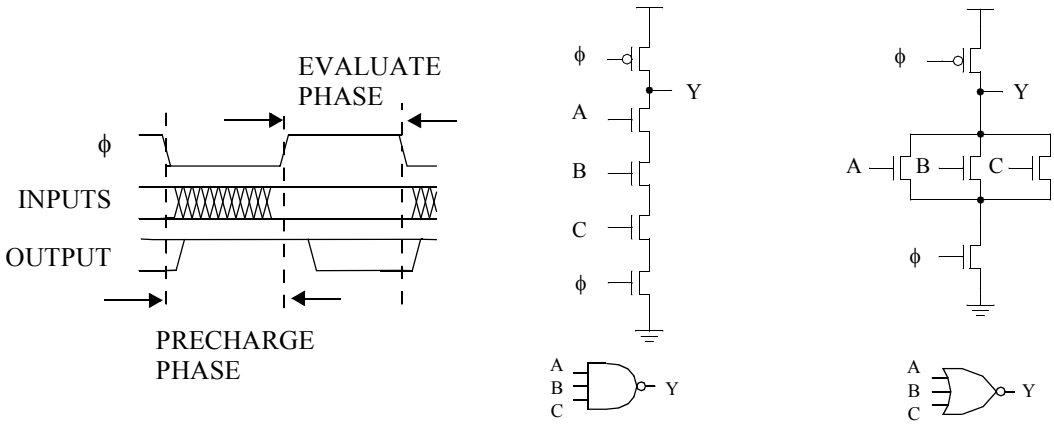


Figure 2-16: Simple dynamic NOR and NAND gates. Also shown is a generic timing diagram showing the precharg and evaluate phase.

simplicity, assuming all other inputs required to form a pulldown path are already high), the dynamic node of gate 1 is discharge to ground and consequently, Y1 goes high. Y1 then discharges the dynamic node of gate 2, eventually causing Y2 to go high. This in turn enables the NMOS in gate 3 and causes Y3 to go high. This sequence of events gives rise to the name "Domino Logic," where the outputs of a gate cause downstream gates to be activated, even though all of the gates are put in the evaluate-phase simultaneously. In domino logic, all outputs are initialized to a precharge value, and once the evaluate phase starts, the primary input causes a domino effect that eventually reaches the primary output

An additional advantage in using cascaded domino gates is the possibility of removing all the NMOS foot transistors in all of the pulldown stages after the first one. When doing this, it must be ensured that no low-impedance path from Vdd to ground is created during the operation. If some inputs are supplied by non-domino-logic, further analysis has to be done to ensure that the discharge path does not form. If all inputs are coming from other domino logic gates, it can be insured that precharging the domino gates is enough to make sure that no pulldown path will be formed, making the clocked-NMOS unnecessary, making the gates both smaller (less transistors) and faster (less effective resistance and hence, larger current drive).

Although domino logic has numerous advantages, it also introduces additional concerns, among them lesser noise tolerance and the need of a clocking scheme to generate the precharge signal. As more and more dynamic gates are used in the system, the power consumed by the precharge signal also increases. Because of this, more sophisticated dynamic circuits have been proposed and used for memory circuits in the form of self-resetting logic that will be discussed later.

Source Coupled Logic (SCL). One dynamic technique that has been used in memory decoders is the source-coupled logic (SCL) (Nambu1998) (Note: It must be mentioned that the name source-coupled logic has been used for other completely unrelated techniques, and we choose to retain the original author's naming convention).

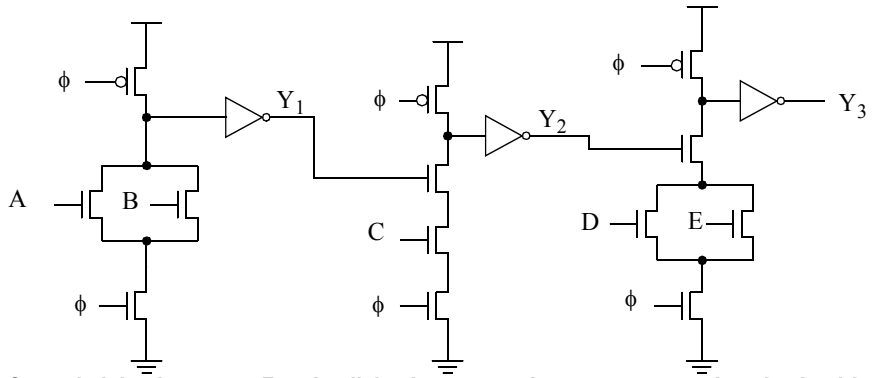


Figure 2-17: Cascaded domino gates. For simplicity, inputs not shown are assumed to also be driven by domino gates.

A 3-input SCL NOR/OR gate is shown in Figure 2-18, along with a timing diagram showing its operation. This SCL circuit is similar to the basic dynamic 3-input NOR gate with an additional cascaded inverting branch and additional cross-coupled PMOS pullups to maintain stability.

This SCL circuit has three main advantages: First, increasing the gate fan-in causes insignificant delay increase because the addition of transistors result only in incremental increase in the capacitance of the NOR output due to the diffusion capacitance from the added transistors. This contrasts with the difficulty in increasing the fan-in of a static NOR configuration where because of the series-connected PMOS transistors in the pull-up network (Although this is true for dynamic NOR gates, in general). Secondly, the output delay of the NOR and OR signals are the same, which is not true of circuits deriving the complement signal using an additional inverter. This reduces the worst-case delay of the gate. Lastly, a subtle but very important advantage of this circuit is shown in Figure 2-19. Here, the gate output is considered active (i.e., selected) when the output is low. The timing diagram shows the activity of a domino OR and an SCL OR for two cycles, the first where both are unselected, and the second where both are selected. The important observation here is that the

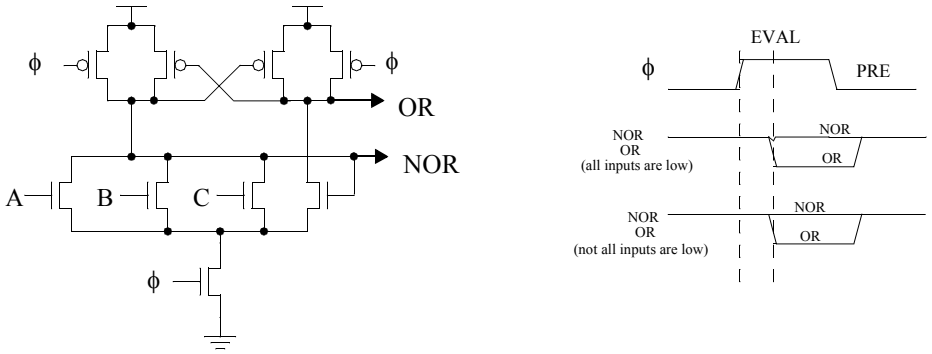


Figure 2-18: 3-input SCL NOR/OR gate.

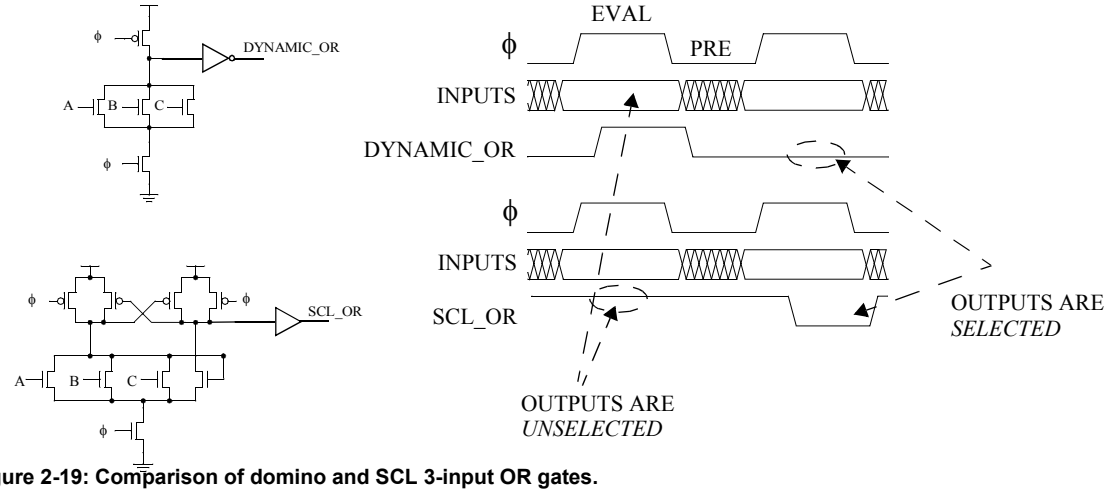


Figure 2-19: Comparison of domino and SCL 3-input OR gates.

dynamic-OR output undergoes a transition when it is unselected, and stays the same otherwise. When this dynamic-OR is used as a predecoder gate that drives the GWL drivers of every row, all the unselected predecode lines will burn power because of the transistions of the unselected outputs. On the other hand, SCL eliminates this unnecessary power consumption by ensuring that only the single selected predecode wire will burn power.

Here, the presence of the BUF-OR is only to emphasize that power is consumed only for transitions of the OR output. the SCL still burns power in the internal node transitions when it is unselected but is almost negligible compared to the power burned by the final gate output when driving its large load.

As mentioned earlier, the advantages of SCL become much clearer in the context of a decode hierarchy, where many NOR gates will exist (equivalent to many multiple-input ANDs) with only a very small minority of them being selected. It is therefore very desirable to have only this small minority burn significant power, as opposed to having most of the gates needlessly burn power.

Buffering and Skewing. Before discussing the next two logic styles, we take some time discuss the concept of buffering and device skewing in the context of memory decoders.

Figure 2-20 shows a typical SRAM floorplan showing where predecoders, GWL decoders, and LWL decoders are located. The figure shows an example amount wire length being driven by the various components of the decoder. The GWL decoder, for example, is driving 2 x 800μm of wire.Given sample parameters of the TSMC 0.25μm process taken from MOSIS, a 1um thick metal 3 line of this length has roughly 200fF effective capacitance.

This kind of load, in addition to the transistor capacitances connected to these lines, cannot be driven efficiently by any single-stage gate and we will have to insert buffer chains to the outputs of our gates to be able to drive the load properly while optimizing the delay. This is done by sizing each stage properly, which has been a very widely studied problem [Jaeger1975, Cherkauer1995]. One of the rule of thumbs [Jaeger1975] is that delay optimization requires the delay of each stage to be the same, and that the practical fan out of each

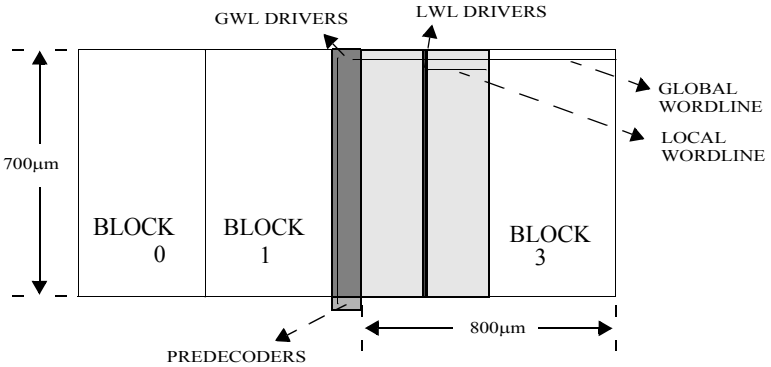


Figure 2-20: Example SRAM floorplan showing the memory arrays and the decoder hierarchy.

gate is set to be about 4. In addition, when it is important to speed up only one specific edge of the output, we can skew the devices inside the gates and buffers to speed up the important transition.

This device skewing is very useful in decoder design because it is important to speed up the assertion of the wordline to activate the memory cell access transistors, while the time to deassert the gates producing the wordline is less critical since it can be done in parallel with other operations within the SRAM. Figure 2-21 shows a gate-level circuit and three transistor-level circuits demonstrating these concepts.

The first circuit shows the gate-level representation that is equivalent to a 2-input NAND gate with the inverters assumed to be sized to drive the load optimally. It also shows that, in this example, we arbitrarily want to optimize the falling edge at the last output. Given this information, we can determine which edge has to be favored at every gate in the chain.

The second circuit simply shows the basic full-CMOS implementation of the 2-input NAND and the inverters where the drive strength of the pull-up and pulldown network are the same to yield roughly the same speed for both the rising and falling edge.

The third circuit is derived directly from the second circuit with all the transistors not participating in the favored edge weakened and made minimum-sized. For example, the last stage has the PMOS weakened since it doesnot help in pulling down or discharging the output to get the desired fast falling edge.

The main advantage of this skewing technique is the reduction of the gate's logical effort because of the reduction of their input capacitances resulting from weakening some of the transistors drastically, making them easier to drive and resulting in significant speedup. The problem in weakening some of the transistors is that the reverse transition will proceed much slower than before, probably negating any speedup in the forward path by requiring a much longer reset period.

To offset this problem, additional reset transistors are inserted, as shown in the third circuit. These reset transistors are as strong as their full-CMOS counterpart, providing enough strength to perform the reset

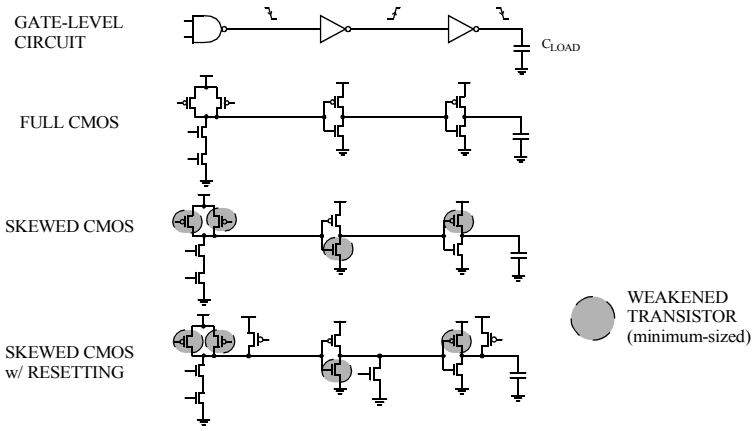


Figure 2-21: Different circuits demonstrating how to speed up an output edge.

properly. A simple way to activate these reset transistors is to treat them as precharge logic for dynamic circuits and control them through an external prechcarge clock (being careful that no path from Vdd to ground is enabled during precharge). This is the simplest method to use since no special circuits have to be designed other than distributing the existing precharge clock (assuming the system already has one). But in the context of decoder design, this method will be very power inefficient because the precharge logic has to be fed to all gates using this technique even though most of these gates that are within the decoder tend to be inactive and will not rquire resetting. This results in an unnecessary increase in the power consumption of the precharge clock. One way to solve this power inefficiency is to generate localized reset signals such that only the few gates that are activated generate reset signals and burn power driving the reset transistors. Two different methods that accomplish this are discussed next.

SRCMOS. Self-resetting CMOS, or SRCMOS [Chappel1991, Park1998] uses its own output pulse to generate a local reset signal to precharge its transistors. As long as no output pulse is produced by the gate, the reset transistors are inactive and do not burn dynamic power.

Figure 2-22 shows two SRCMOS circuits including the timing waveforms as a result of the pulse-mode inputs A and B. The first circuit is derived from the skewed CMOS with reset transistors in Figure 2-21. A delay chain has been inserted to derive the reset signals of all reset transistors from the output pulse. Without this pulse, all these circuits are quiet and burn no AC power. Once the reset signal has initialized the state of the gate, the new state travels back to deassert this reset. In addition, an extra series NMOS has been inserted in the initial NAND gate to disable the pulldown network in case the pulse-mode inputs are not

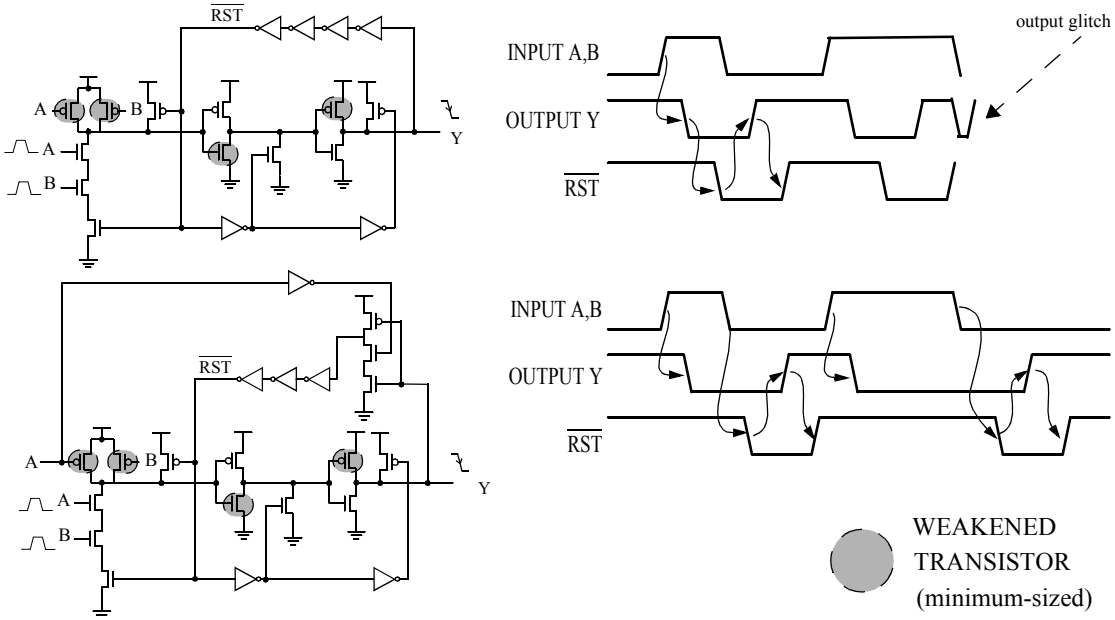


Figure 2-22: SRCMOS 2-input NAND circuits.

guaranteed to turn off in time before the reset signal reenables the NAND PMOS for the next state of inputs, thereby creating a current path from Vdd to ground.

One problem with the first circuit exists if the pulse-widths of the inputs are large enough such that they are still active at the time when the reset signal has had time to go back to its original deasserted state after the entire reset process. When this occurs, an additional, undesired pulse will be produced at the output. In this case, one solution is to predicate the initial reset upon the falling edge of one of the inputs, as shown in the second circuit, where the first inverter in the delay chain requires one of the inputs to be low in order to be activated.

DRCMOS. The extra series NMOS in SRCMOS will tend to increase the logical effort of the gate, which reduces the initial benefits of SRCMOS. DRCMOS [Nambu1998, Amrutur1998, Heald1993] solves this problem by predicating the reset signal activation with the falling input even for the initial propagation around the loop, as shown in Figure 2-23.

At its initial state, the transmission gate NMOS is enabled to initially provide no reset signals. Once the input and NAND output changes, the transmission gate is disabled, to be enabled only when the input goes low again, allowing the reset signal to propagate through the gates and eventually, stop the reset process automatically back to its initial state.

With the removal of the extra series NMOS, the DRCMOS technique will have a lower logical effort and be faster than even SRCMOS logic. the main caveat is that the output pulse width is always longer than the input pulsewidth since the output is reset only after the inputs finish. This limits the number of levels in the decode path that DRCMOS can be used without exceeding the cycle time.

The SRCMOS and DRCMOS techniques can be used for different kinds of logic, including full-CMOS logic used in the previous examples. Figure 2-24 shows how the SCL technique discussed before can utilize DRCMOS logic in implementing a NOR-style decoder that is useful as a 4-to-16 predecoder [Amrutur1998].

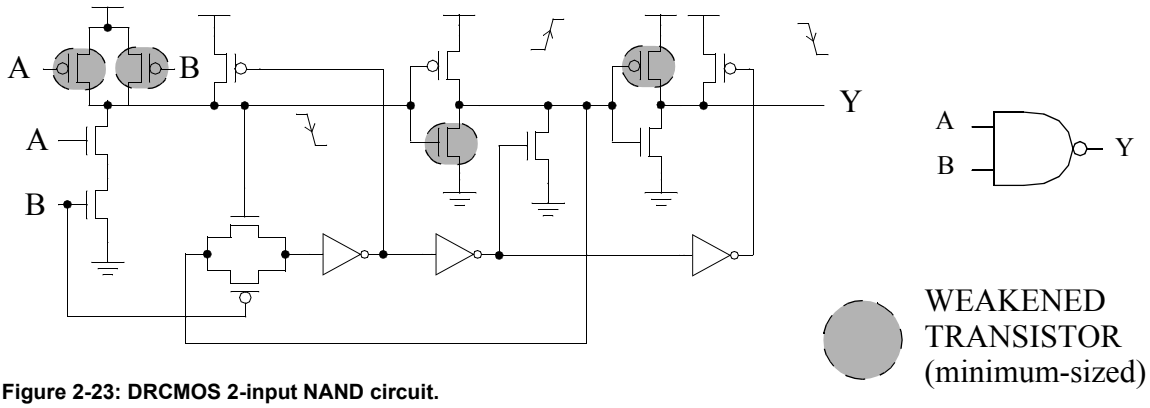


Figure 2-23: DRCMOS 2-input NAND circuit.

In this case, the delayed reset is predicated upon the precharge clock. It must be emphasized that although all gates of these type use the precharge clock, only the gates whose outputs actually underwent transitions will burn power to turn on the reset transistors.

SCL and DRCMOS complement each other in avoiding unnecessary power consumption since SCL logic only undergoes output transitions whenever they are activated and selected by the right combination of inputs. By not undergoing unnecessary transitions, it enables DRCMOS to further save power in the reset circuitry and at the same time, speeding up the favored transition due to its low logical effort and device-skewed implementation while keeping the reset period manageable.

TWD. As an alternative to full-CMOS NAND gates, the transfer-word driver (TWD) shown in Figure 2-25 can be used [Aizaki1990]. By using a single always-on PMOS as a pullup, and by using a single NMOS transistor to perform the ANDing operation, the circuit has a reduced input capacitance compared to a full-CMOS gate (less than half depending on PMOS sizing of the full-CMOS gate). This allows the TWD gate to operate faster than its full-CMOS counterpart. In addition, its simplicity results in a smaller decoder area (although this isn't overly significant -- 20% according to Aizaki) because the area of the decoder circuit will typically be dominated by the drivers, not the initial stage.

It should be noted from the figure that the TWD configuration needs one of its inputs to be low-asserted to perform the same functionality as a NAND gate. This isn't typically a problem when used as part of a decode hierarchy as the inversion can be done using an extra driver stage. This additional stage shouldn't

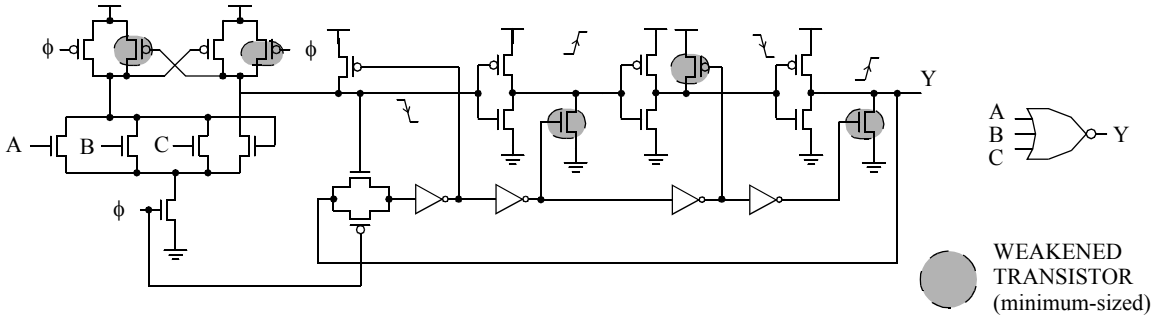


Figure 2-24: 4-input SCL NOR circuit using the DRCMOS technique.

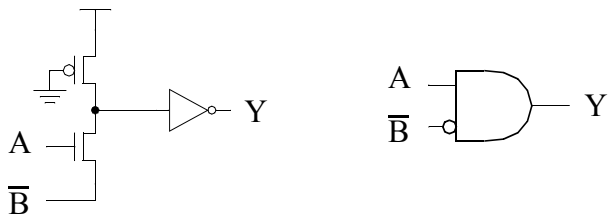


Figure 2-25: Transfer-word driver (TWD) circuit.

affect the critical path because it can be used on the non-critical part of the decode. For example, when used in the LWL decoder, ANDing a GWL and a block select signal, the block select usually arrives earlier than the GWL, and can absorb another stage of inversion, if necessary.

The main potential disadvantage of the TWD circuit is the static current drawn because of the path from Vdd to ground whenever the gate is active (i.e. the NMOS is activated). Although this makes the TWD gate impractical for use in general circuits, this isn't a big problem when used in LWL decoders. The decreased gate capacitance lowers dynamic power and helps compensate for the static power. In addition, only a very, very small fraction of these gates within LWL decoders would actually be activated (and hence, consume static power) during any given access. Moreover, use of PWL techniques serve to minimize the amount of time these gates are active, further decreasing the amount of static power consumption.

Peripheral Bitline Circuits. To support the operation of the SRAM memory cells, they are accompanied by additional peripheral circuitry, as shown in Figure 2-26, parts of which we discuss next.

. Referring again to part of Figure 2-4 (which is partly replicated in Figure 2-27), all bitlines are assumed to be precharged to a given voltage (currently, this is most often Vdd). When WL0 asserts, all the memory cells connected to this wordline have their access transistors enabled. For a read operation, the

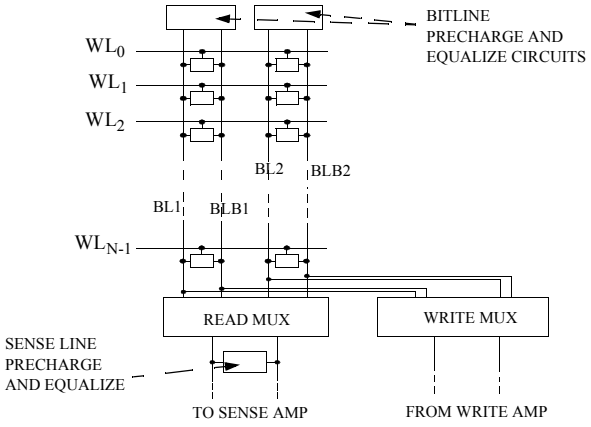


Figure 2-26: Memory cells showing peripheral bitline circuits, including precharge devices and read/write muxes.

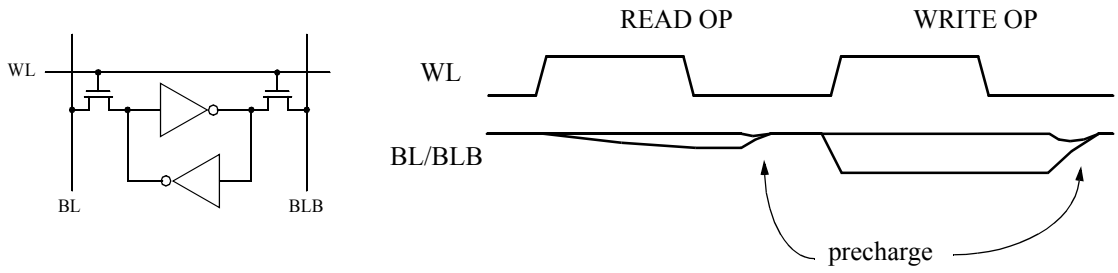


Figure 2-27: Read and write timing diagram for a memory cell, including the precharge operations.

accessed memory cells are allowed to pulldown the voltage of the bitline (the other bitline will be pulled up because of the complimentary signals stored in the cell). Since the initial bitline voltage is already high, it is usually the pulldown operation that is important. The voltage of the bitline being pulled down will steadily drop, with the rate of change dependent on mainly the bitline capacitance (made up of the diffusion capacitances of each access transistor connected to the bitline and various wire parasitics), and the strength of the driver and access transistors. This process continues until WL0 is deactivated. Depending on the type of precharge circuit, the bitline voltage will either start to be precharged or stay constant (more on this later). In the timing diagram shown, it is assumed to stay constant.

During this process, the read mux selects the desired column based on the address input and relays its voltage difference to a sense amplifier that serves to speed up the sensing process. Before the next operation can occur, the bitlines (and the sense lines) have to again be precharged to a fixed initial value. This is done by the precharge and equalization circuit in the bit and sense lines.

As WL0 is asserted for the second time for a write operation, the access transistors connected to WL0 are again enabled. But this time, external data from the write amplifier has imposed a full-swing differential voltage on the bitlines. Once the access transistors are enabled, the higher capacitance of the bitline and the stronger drive strength of the write amp forces the accessed memory cells to have the same logic value as the bitlines. As with the read operation, the bitlines are again precharged to an initial value after the operation.

We now discuss individual parts of the bitline peripheral circuits in more detail.

Precharge and Equalize Circuits. Various ways have been used to perform precharge and equalization, and some representative circuits are shown in Figure 2-28. The first circuit is one of the early implementations. It uses a diode-connected NMOS pair without equalization circuits. This configuration precharges the bitlines to Vdd - Vt where Vt is the threshold votage of the NMOS.

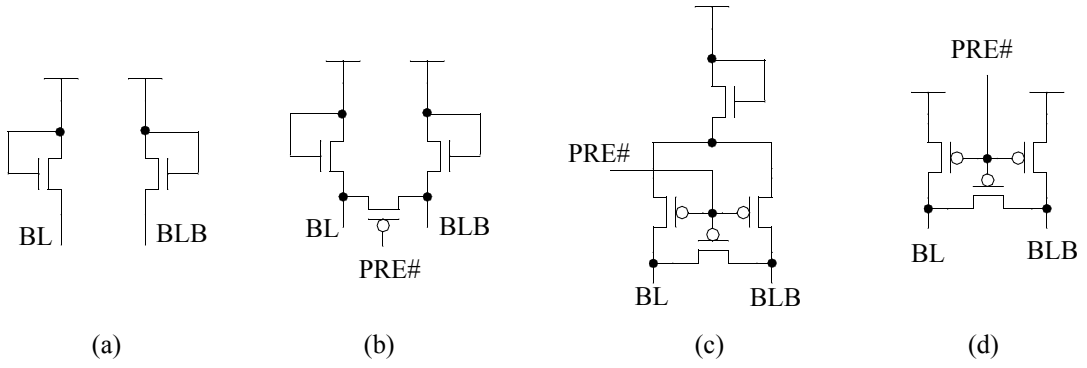


Figure 2-28: Different precharge and equalize circuitry.

This configuration continuously tries to pullup the bitlines high, and this serves both as a strength and weakness. Since there is no need for additional control to enable precharge, this circuit helps to simply the control complexity. But at the same time, the constant current path provided by the transistor towards Vdd tends to slow down development of a bitline voltage differential during read operations (since it fights the memory cell pulling the bitline down) and burns more power during write operations where one of the bitilines is pulled low by the write amp, again fighting this constant pull-up. Alternatively, these pullups can be implemented using always-on PMOS transistors if it is desired to precharge the bitline to the full Vdd level..

The second circuit shows diode-connected NMOS transistors with a PMOS transistor bridging the two bitlines. This PMOS transistor serves to equalize the voltage between the two bitlines whenever it is enabled. This becomes especially important for active NMOS-load because of possible variations between the threshold voltage of the NMOS, causing a difference in the precharge level. As mentioned earlier, the active NMOS pullups can also be replaced by PMOS transistors if the inherent Vt level-shift due to the diode connection is unnecessary. (A typical use of this voltage shift is to adjust the bitline common-mode voltage to fit the sense amplifier's high-gain region)

The third circuit uses a combination of PMOS and NMOS transistors for the precharge and equalization circuits. It uses a single NMOS to establish a precharge level of Vdd-Vt, but it uses PMOS pass transistors to selectively connect the load to the bitlines only during precharge operations (more on this in the explanation of the next circuit). This kind of configuration was typical for moderate supply voltages (e.g. 3.3V*)* or when the sense amplifiers used for amplification performed more optimally only at common mode voltage levels below the supply voltage. But as supply voltages go down, and Vt differences due to process variations may significantly affect the performance, this configuration starts being impractical.

The last circuit shown is currently the most popular implementation. During precharge and equalize operations, all three PMOS transistors are enabled, providing low-impedance paths from both bitlines to Vdd and to each other. When the bitlines have been precharged, these transistors are turned off, resulting in very high impedances isolating the Vdd from both bitlines. With this circuit, development of bitline voltage differential is sped up because the pulldown only has to discharge the existing bitline capacitance. During write operations, no unnecessary power is wasted in the precharge circuit since it doesn't affect the operation of the write amp. The main disadvantage of this technique is the additional complexity and power needed to control the clocked precharge and equalization transistors. Although power consumption during clocking of these precharge elements within the whole SRAM may be lessened by special circuits with conditional locally-generated control signals, there will always be power consumed switching the gates of these transistors on and off.

The sizing of these precharge transistors are dictated by how much time is allocated to the precharge operation. Larger transistors are able to precharge the bitlines faster, but will dissipate more power. For

example, using larger transistors for the hi-Z precharge implementation will dissipate more power in the precharge clock network (either global or local) because of the larger gate capacitances. The write operation often dictates how big these precharge transistors have to be because bitilines during writes are discharged completely, unlike the partial discharge due to a typical read.

To take advantage of the difference between the read and write operation, the precharge circuits are often separated into read precharge and write precharge. The read precharge is usually enabled every time and is sized small enough to effectively charge up the bitlines from the expected bitline voltage drop during a read. The write precharge is enabled only during writes and serves to help the read precharge to charge up the bitline. In clocked schemes, this avoids the power consumed in unnecessarily switching the gate inputs of big write precharge transistors during read operations.

Read and Write Multiplexers. Read and write multiplexers allow multiple bitlines to share common sense amps and write amps, as shown by an example in Figure 2-29 where a single sense amp and write amp are shared by multiple bitline pairs. The mux can easily be expanded to accomodate more bitline pairs, and these considerations are discussed later in the partitioning subsection.

The most efficient way of implementing these muxes is the use of simple pass transistors. For the read mux PMOS transistors are used since the common mode voltages that need to be passed through are near the logic high value, resulting in better device transconductance. For the write mux, the write amp will try to discharge one of the bitlines, requiring an NMOS pass transistor to pass a strong logic low value.

In both cases, only a single pass transistor and not a complimentary transmission gate is needed since the complement transistor will not be used effectively. During reads, an NMOS will often go unused since it will not be able to pass voltages above Vdd-Vt, and modern SRAMs don't develop enough differential to dip below this value. During writes, only a logic low has to be transmitted to one side of the bitline pair, with the perfectly reasonable assumption that the bitlines being precharged high is enough to form a full-swing voltage differential to force a write to the memory cell.

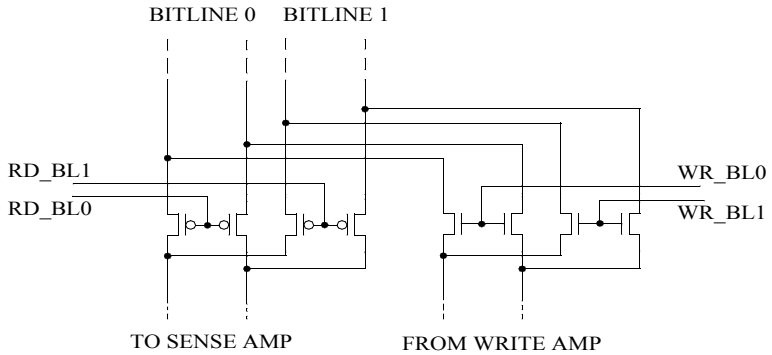


Figure 2-29: A circuit showing two bitline pairs sharing a single sense amp and write amp using read and write mu tiplexers.

Sense Amplifiers. The development of a voltage differential in the bitline pairs during a read access is a slow process because the pulldown device of the memory cell is hard-pressed to discharge the large capacitive load of the bitline.

A sense amplifier can be used to speed up the development of this bitline voltage differential. Figure 2-30 shows the operation of a sense amplifier along with a timing diagram showing some signal waveforms. When the wordline asserts, a differential slowly develops across the bitline pair. After some time, the PMOS pass gate is enabled (usually only after a minimum differential has been established as required by the particular sense amp). The sense amp then amplifies the bitline differential, quickly producing a full-swing differential output.

Note again that further development of bitline differential is unnecessary and will actually be undesired because more power will be consumed to precharge the bitline. The second read access demonstrates how using PWL serves to conserve power (by decreasing the final bitline differential) without any sacrifice in sense speed.

Although often advantageous, use of sense amps are not absolutely necessary for sensing since it simply speeds up the development of the differential across the bitlines, which is continuously being discharged by the memory cell (as long as its wordline is enabled). This is unlike DRAM where the differential voltage being sensed is due to a one-time contribution from the memory cell capacitor, which necessitates the use of sensing. In fact, most very small memories like register files often do not use sense amplifiers as the bitline capacitances of these structures are often small enough to be discharged quickly by a memory cell and by doing so, avoids the significant power consumed by typical sense-amps.

Some contemporary caches that rely on single-ended sensing (like the Intel Itanium0 also do away with the sense-amplifiers in their cache hierarchy. In these implementations, the delay through the bitlines are small enough that they can be treated as full-swing logic signals and fed to conventional logic gates without further amplification.

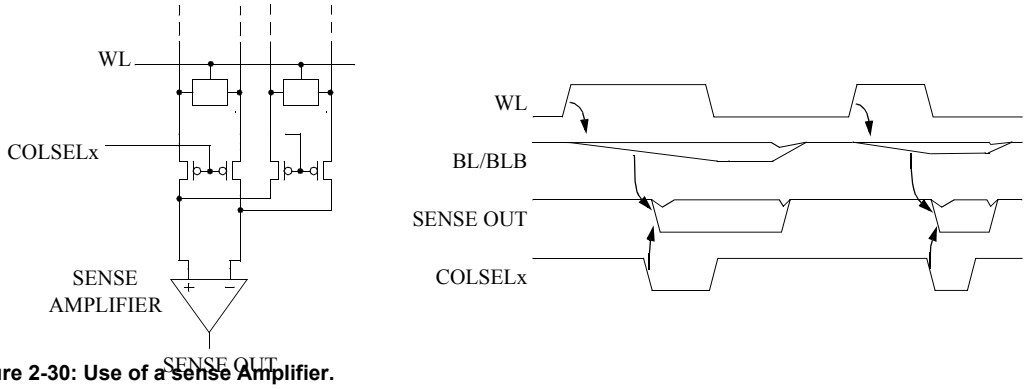


Figure 2-30: Use of a sense Amplifier.

Physical Implementation. A large collection of different sense-amp circuits can be found in the literature. These circuits range from simple cross-coupled inverters to very sophisticated amplifier circuits. Some of the more common sense-amp circuits are shown in Figure 2-31.

The first circuit is the simple latch-type sense-amplifier [Uchiyama1991] that forms cross-coupled inverters whenever the sense-amp is enabled. Because of its simplicity, this sense-amplifier occupies a very small area and often satisfies the speed, area and power tradeoffs involved in designing wide memories that require a significant number of sense amps. One problem with the latch-type sense-amp is the requirement to enable the amplifier only after a minimum voltage differential is present in the bitlines otherwise, the latch could flip in the wrong direction and overpower the bitline differential.

The second and third circuits employ current mirror amplifiers, with the first circuit using a single current mirror, while the second circuit uses dual current mirror amplifiers and is named a paired current-mirror amplifier (PCMA). These amplifier offers fast sense speed, large voltage gain and good output voltage stability. One problem with these circuits is their high power consumption because of the existence of a static current path from Vdd to ground whenever the amplifier is enabled. In addition, this current becomes larger as the amplifiier is designed to become faster, as shown by the sidebar. Moreover, its complexity makes it impractical to implement a sizeable number of these sense-amps, making them unsuitable for use in wide-word applications.

The fourth circuit is the PMOS cross-coupled amplifier (PCCA) [Sasaki1989] and it offers fast sense speed at a much lower current than the PCMA because no static current path exists when the sense-amp is enabled. The problem with the PCCA is its need for some preamplification to achieve its fast operation, and it is often better to pair it with a preamp like a PCMA or a latch-type sense-amp to avoid spurious data output because of its high voltage gain. This is especially true when paired transistors in the amplifiers are mismatched [Sasaki1990].

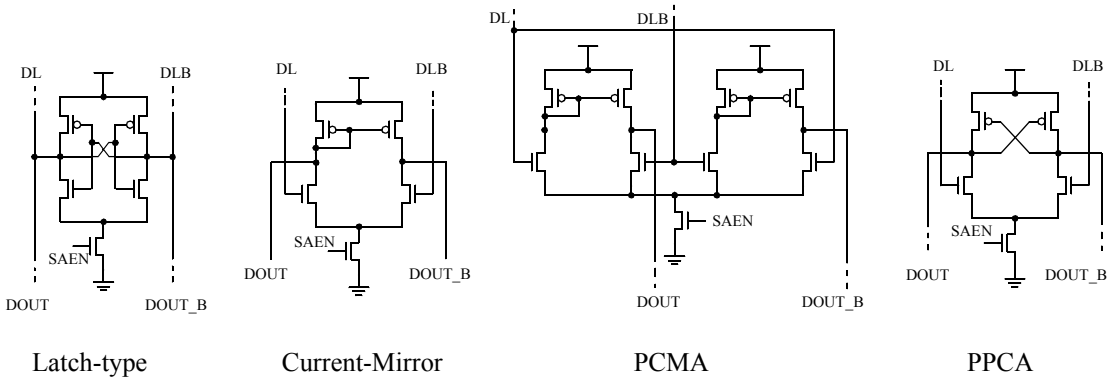


Figure 2-31: Sense amplifier circuits.

Sensing Hierarchy. Large megabit SRAMs typically use multi-level sense-amp hierarchies like the one shown in Figure 2-32. The figure shows a scheme with three levels of hierarchy and a single bit of output data. Oftentimes, amplifiers in different levels have differing topologies to optimize the entire hierarchy.

Although these types of sensing hierarchies are almost a given in discrete SRAMs, they are not widely used in typical cache implementations, where a single level of sensing is most often adequate and, in fact, some contemporary caches even dispense with the amplifier altogether (like the Intel Itanium). The main reason for this is the wide-word nature of caches, which is typically much greater than 32 bits in length, as opposed to most SRAMs which are typically from 1 bit to 9 or 18 bits wide.

The wide output of typical caches often necessitate the use of smaller, identical blocks of SRAM whose outputs are then used to form the required wide data bus. Consequently, use of these similar SRAM blocks do not require the implementation of complex, multi-level sensing hierarchies.

Write Amplifier. Write amplifiers are used to generate the required voltages to flip the state of a memory cell when needed. For the 1 R/W 6TMC, the easiest way the cell can be written is by applying a full-swing differential voltage to the bitlines and enabling the cell's access transistor, as shown in figure 2-33,.

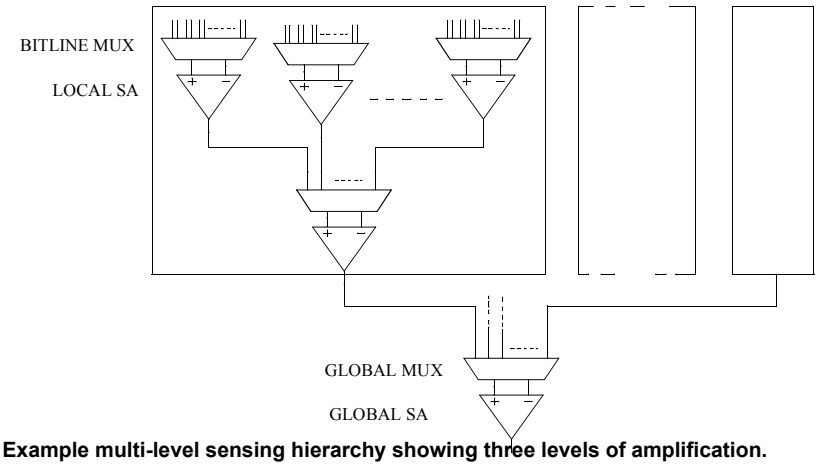


Figure 2-32: Example multi-level sensing hierarchy showing three levels of amplification.

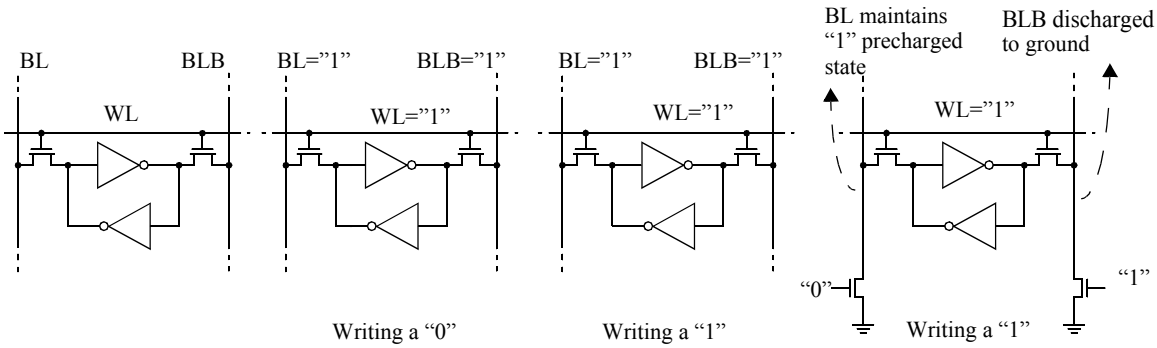


Figure 2-33: Writing to a memory cell.

Typically, the write operation in any SRAM is not critical, so write amplifiers are much more simple compared to sense amps. In addition, if bitlines are assumed to be precharged to a high level, the write amp only needs to discharge one of the bitlines to ground, with the full differential voltage being applied with the help of the precharged high value of the remaining bitline. This concept is demonstrated by the fourth circuit shown in the figure, where the right NMOS transistor discharges BLB to ground, while the left NMOS is left disabled, maintining the precharged high state of BL.

A complete write amp circuit is shown if Figure 2-34, along with some of the other bitline peripheral circuits. In the figure, the NMOS write-mux transistors connect a single write amplifier to their corresponding bitline pair. Each of these write muxes are enabled only if the column they are connected to is chosen during a write operation.

The write amplifier consists of two NMOS transistors and proper buffering and inversion logic to enable the proper discharging transistor. These transistors, along with the write mux, are sized to insure that they can discharge the bitlines within the allotted time window. Typically, this window is dictated by the bitline read time during read operations, so the write transistors need only moderate pulldown discharging capabilities. Also, since write muxes employing differential writes only need to discharge one of the bitlines when it is assumed that all bitlines are precharged high, only NMOS transistors are needed for the write amp.

Methods for current-mode writes have also been developed, aiming to avoid the significant power required to return fully-discharged bitlines back to their precharged states. The main drawback for this method is they typically require the modification of the base memory cell in order to facilitate the current mode write. An example system is shown in Figure TBD [TBD] where an equalizing transistor is needed to put the cross-coupled inverters in the cells to be written to a quasi-stable state, to be eventually pushed to its final state by the relatively small current produced by the current-mode amp.

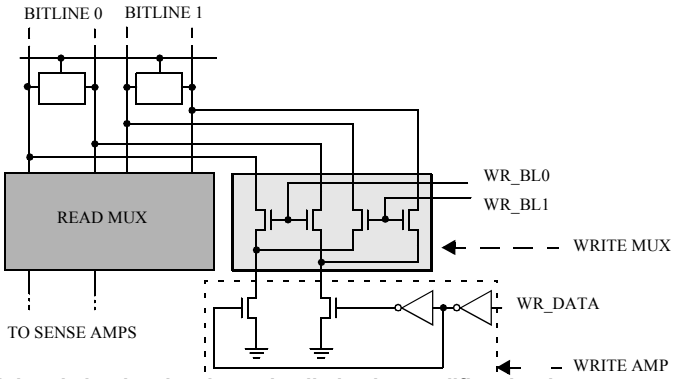


Figure 2-34: Bitline peripheral circuits showing a detailed write amplifier circuit.

SRAM Partitioning. Having discussed many parameters involved in the organization of the SRAM in terms of its decode hierarchy and its peripheral circuits, we now discuss partitioning SRAMS to optimize performance in terms of speed, power and area..

Among the SRAM parameters that have already been discussed (e.g. row height, number of blocks, number of columns per block, etc.), it is relatively easy to judge the isolated effect of varying a single parameter. Reducing the number of columns of an SRAM, for example, will serve to reduce the wordline capacitance, making the decode process faster. Another example is dividing the SRAM into as many blocks as possible to reduce the amount of unnecessary bitline poewr consumption. When properly used, these techniques are effective, but improper application of these techniques without taking their consequences into account will result in a less than optimal implementation. In the first of the two previous example, indiscriminately increasing the number of rows does reduce the wordline load (given constant memory size), but serves to increase the bitline capacitance. In the second example, the increase in the number of blocks does lower bitline power consumption, but will also increase both the decoder power consumption and delay.

The key to balancing these requirements is to partition the SRAM in such a way that the positive effects of changing a certain SRAM parameter is not counterbalanced by negative effects caused by the same parameter.

Amrutur and Horowitz provide a very good discussion of speed and power paritioning for SRAMs [Amrutur2000]. Amrutur and Horowitz model the speed, power and area characteristics of an SRAM and solve for the optimal partitioning based on given priorities for speed, power and area. Figure 2-35 shows the organizational parameters that are used for the optimization, and a plot of optimal area vs. delay (given no power constraints), and energy vs. delay for a 4Mb, 0.25um SRAM. The figure shows a 1024x1024 array partitioned using three organizational parameters: the number of macros (*nm*), the block height (*bh*) and the block width (*bw*). Each macro supplies a subset of the output word (in this case, 16 bits of the 64-bit word), and each macro is divided into subblocks, with each subblock being of size *bh* x *bw*. The output of a macro is supplied by a single subblock selected by the given address. As shown in the figure, the division of a macro into subblocks can be done both vertically or horizontally. When divided vertically, a multi-level sensing

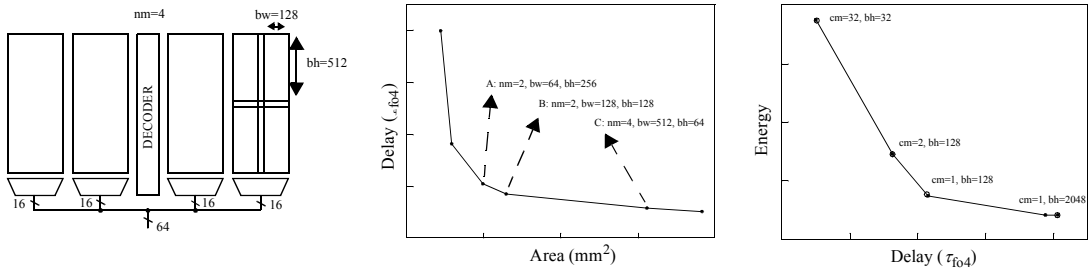


Figure 2-35: SRAM partitioning.

hierarchy can be used where subblcoks in the same vertical axis can share a common global bitline and global sense amplfier.

The first graph in the figure shows the optimal partitioning (given no power constraints) resulting in minimum area for a given delay, with the swee spots labeled as points A and B. Amrutur and Horowitz find that the RAM delay is most sensitive to the block height, and small block heights result in the fastest access times.

The second graph in the figure shows the energy and delay relationship with no area constraints. An additional parameter, the amount of column multiplexing (*cm*, or the number of columns sharing a single sense amp) is shown. It shows that minimum delay is achieved firstly by having a*cm* of 1 where each bit has its own sense amp such that no columns are unnecessarily activated and secondly, having a large block height is desirable to allow the muxing to be performed in the bitlines.

SRAM Control and Timing. This subsection describes how SRAM operation is controlled, both internally and externally. Figure 2-36 shows all of the components of an SRAM necessary to perform a complete read or write access to a specific memory cell. To simplify the figure, only one memory cell in a single column is shown.

It is important for SRAM operation to have a distinct window of time within which it can perform the desired operation and afterwards reinitialize itself to be able to perform the next access. Within this window of time, the SRAM must be allowed to finish the complete sequence of operations before the next access is started. Otherwise, both the present and next access will result in corruption of data. This characteristic is

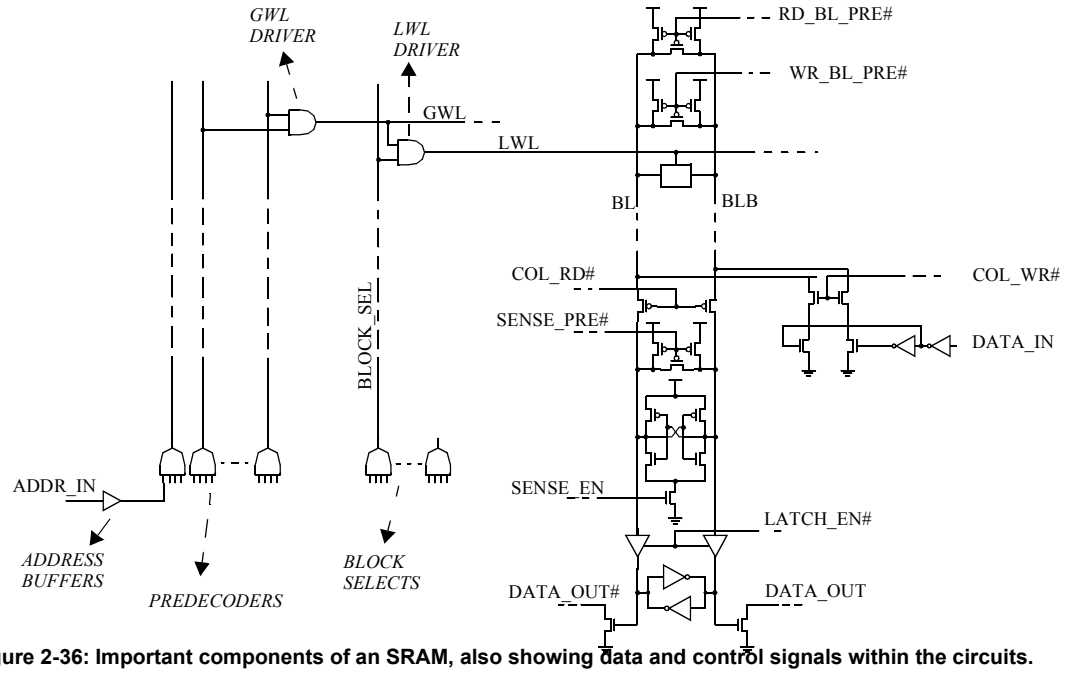


Figure 2-36: Important components of an SRAM, also showing data and control signals within the circuits.

unlike combinational logic whose inputs can be changed at any time while expecting the output to stabilize after a given propagation delay.

With this in mind, in general, there are two ways of starting SRAM operation. The first is to detect transitions in the address or control inputs. An access is started when these inputs are sensed to have changed. This method is called the address transition detection (ATD) method, and a typical circuit implementation is shown in Figure 2-37. The pulse generated by the ATD circuit propagates to control the SRAM sequencing of events, including precharging the SRAM to prepare for the next operation.

ATD controlled SRAMs are classified as asynchronous SRAMs because SRAM operations are started after its input change regardless of the presence of any synchronizing clock in the system.

A second way to start SRAM operation is to use a synchronizing clock signal (or a signal directly related to the clock). It is important to note that in this case, only the interface of the SRAM proceeds synchronously with the clock. Significant parts of the internal SRAM operation proceed asynchronously with the clock, triggered using various methods that we will discuss next. In this case, the clock is important only to define the start and/or end points of the SRAM operation, with the designer being careful that the clock period is long enough to allow the SRAM to perform all the required internal operations.

The start pulse that is generated either by the ATD circuit or derived from the system clock or any other external synchronizing signal then triggers parts of the SRAM in turn. This start pulse will serve a different purpose depending on whether the circuits are implemented using dynamic or static logic. For dynamic logic, the initial edge of the start pulse can be used to start the evaluate phase of the dynamic logic. It is important to emphasize that the address inputs of the decoder, especially the dynamic circuits, have to be stable to ensure the correctness of the SRAM operation. Otherwise, output nodes can be unwantedly discharged, with the circuit recovering only after a precharge operation, which will be too late for the present operation. In addition, we may want the start of a dynamic circuit's precharge to be dependent on a control signal different from the signal that triggered the evaluate phase.

Figure 2-38 shows five waveforms demonstrating this concept. The clock and address signals are external inputs to the SRAM, where the address usually changes right after the clock edge (with the delay

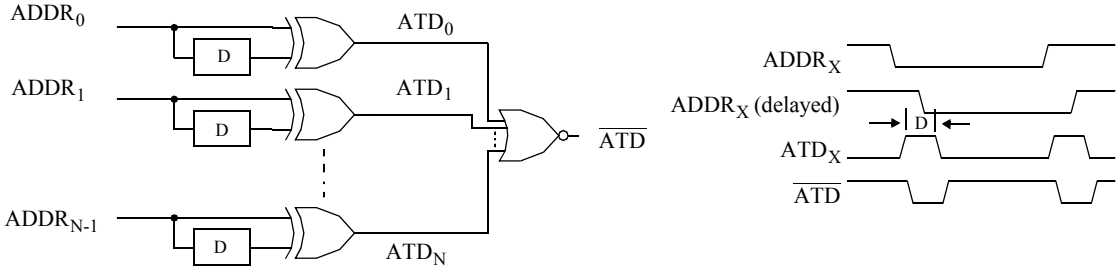


Figure 2-37: Address transition detection (ATD).

representing the clock to output delay of their respective storage elements). The third waveform is an internally generated signal that is a delayed version of the clock with enough delay to ensure the stability of the address. Alternatively, this could also be generated using an ATD circuit, with the falling edge occurring much earlier. In any case, only the first edge is of interest. The fourth waveform is an internally generated signal that is used to signify the start of the precharge phase to prepare the SRAM for the next access (we will discuss how this is generated in a little bit). To serve as a valid precharge/evaluate signal to our dynamic logic, we need something similar to the fifth waveform, whose rising edge follows the START signal's rise and its falling edge follows the END signal's fall. It is also important to note that even though the START signal by itself may be sufficient for the dynamic logic in the circuits, using the signal as is implies that equal time is allotted to the evaluate and precharge phase, which will be inefficient since the precharge phase is given more time than is necessary..

Figure 2-39 shows a circuit that can generate the clock signal of decoder dynamic circuits. Its main functionality is to produce a pulse whose rising edge is initiated by START's rising edge, while its falling edge is initiated by END's falling edge. It consists of level-to-pulse converters that convert the START and END signal to pulses that drive the inputs of a set-reset latch. A START rising edge generates a pulse that sets the latch, while an END falling edge generates a pulse that resets it. The same as with all RS latches, we need to insure that its inputs are not asserted simultaneously (causing indeterminate operation). We will see later that

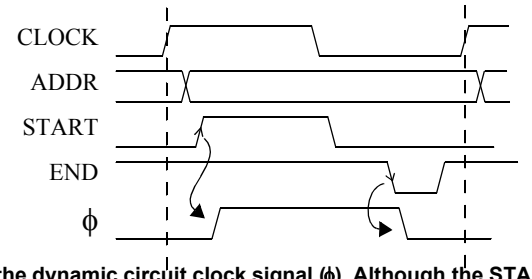


Figure 2-38: Generation of the dynamic circuit clock signal (ϕ). Although the START signal that is generated when the address signals are valid can be used to start the evaluate phase, its falling edge is not suitable to start the precharge phase. The END signal is generated internally when it is safe to start precharging, so it is used to trigger the falling edge of ϕ to start the precharge phase.

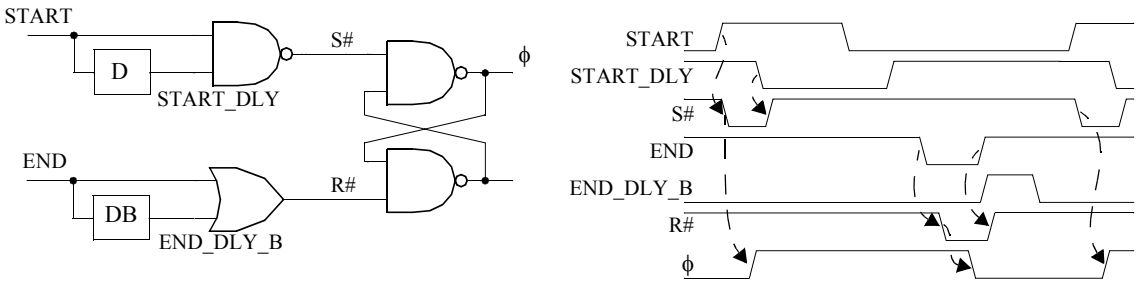


Figure 2-39: Generation of a signal triggered by two separate events. In this case, the rising edge of ϕ is triggered by the rising edge of START, while the ϕ falling edge is triggered by the falling edge of END.

this is trivial for SRAM decoders as the START and END signals involved have very large separations relative to the width of the pulses being generated.

Amrutur and Horowitz [Amrutur2001] state that an optimal decoder hierarchy will have high fan-in inputs only at the initial stage (the predecoder stage), and that all other succeeding blocks of the decoders (e.g. GWL and LWL decoders) should have minimal gate capacitances. The high fan-in requirement for predecoders necessitate the use of dynamic logic for their implementation. Although dynamic logic could also be used for the GWL and LWL decoders, the large number of these circuits will magnify the difficulty associated with dynamic logic.

For static decoders, the use of the START signal to trigger decoder operation is not as critical as in the dynamic implemenetation. The reason is that the static logic will always be able to produce the correct output and assert the correct wordline after some delay once its inputs have stabilize. The same is not true with dynamic logic. The main consequence of allowing the decoder to proceed without the start signal is the creation of glitches in some of the wordlines that are non-critical but nevertheless will increase the power consumption during bitline precharge.

As mentioned earlier, it is beneficial to use a pulsed wordline scheme where the wordline is pulsed only for the minimum amount of time it is needed to develop a sufficient bitline voltage differential that can be used by the sense amp. Limiting the differential reduces the amount of power required to precharge the bitlines to their original state, as well as allowing some of the precharge operations to proceed sooner and in parallel with the sensing operation, possibly resulting in a decrease to the total cycle time of the SRAM.

PWL schemes can be implemented using either open-loop or closed loop schemes. In an open-loop scheme, the word lines are turned off after a certain time delay determined at design time. This is often done with the use of gate delays that produce delayed outputs on parts of the decoder hierarchy which, when they catch up with the original signals, then serve to turn the signals off, producing a pulse whose width is determined by the gate delay, as shown in Figure 2-40. The total gate delay (and the corresponding pulse

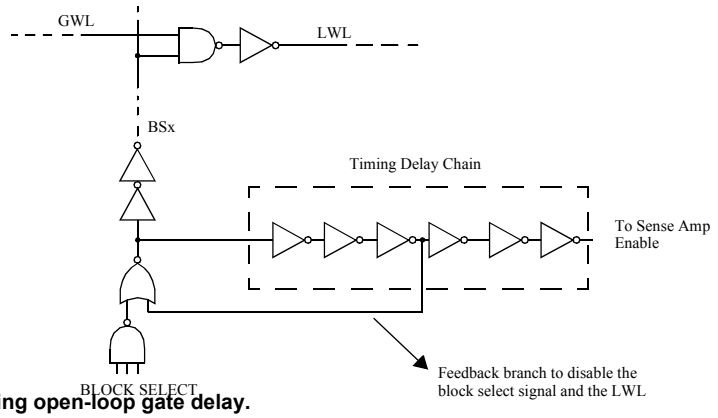


Figure 2-40: PWL using open-loop gate delay.

width) are designed to be long enough such that the memory cells have enough time to discharge the bitlines and develop sufficient differential.

Although this traditional way of implementing a PWL-scheme performed sufficiently well, the effects of technology scaling has made it more difficult to maintain the required correlation between the gate delays and the amount of time required to discharge capacitive bitlines because of the varying extent of how the memory cells and the decoder drivers are affected by scaling. This is further worsened by PVT variations, which affect the circuits' characteristics differently, worsening the correspondence. This problem is due to the bitline delay being dependent on the large bitline capacitance being discharged by a memory cell's (often) minimum-sized pulldown transistor, which is affected more significantly by PVT variations compared to delay of non-minimum-sized gates.

These effects discourage the use of open-loop techniques, necessitating the use of feedback from structures that more closely follow the runtime behavior of the circuit. Different structures have been proposed that use this kind of feedback [Amrutur1998, Nambu1998, Osada2001]. A good example of this scheme is the replica technique by Amrutur and Horowitz [Amrutur1998], as shown in Figure 2-41.

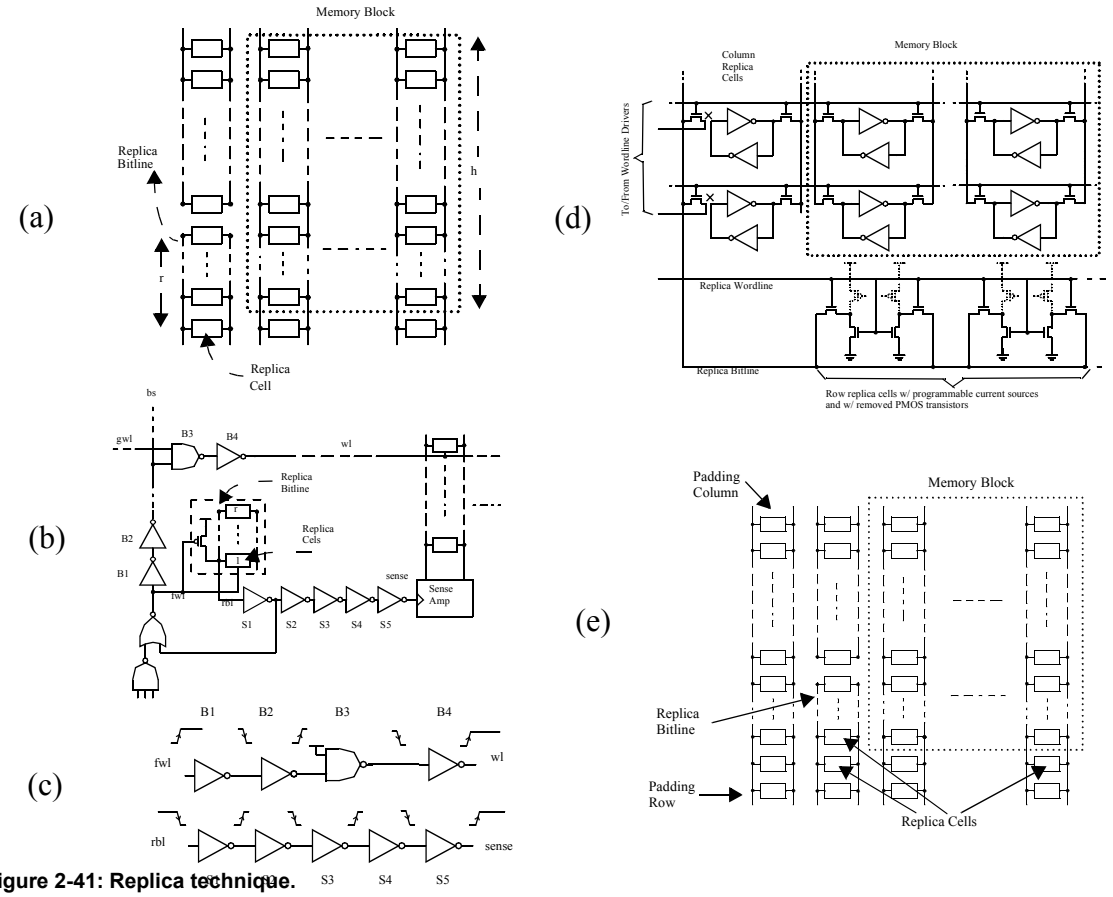


Figure 2-41: Replica technique.

The first three figures show a feedback scheme based on capacitance ratioing. Figure 2-41(a) shows the addition of a replica column to the standard memory array. The replica memory cell is hardwired to store a zero such that it will discharge the replica bitline once it is accessed. Because of its similarity with the actual memory cells (in terms of design and fabrication), the delay of the replica bitline tracks the delays of the real bitlines very well, and can be made roughly equal by varying (at design-time) the number of cells connected to the replica bitline. With this method of generating a delay that is equal to the bitline delay even with significant PVT variations, the problem is shifted to equalizing the delays between two chains of gates, as shown in figure 2-41(c), which is a much easier problem.

Alternatively, feedback based on a current ratioing method can be used instead. The main advantage of this technique over the first method is the ability to generate local resets for each row, which can be used to directly turn off the local word line drivers. This makes the delay balancing easier to do and enables the use of skewed gates even for the LWL decoders, speeding up the decoder delay.

To accomplish this, a replica row and column are added to the regular memory block. In this method, it is the replica row that performs the bitline discharge instead of the replica column. The memory cells within the replica rows become programmable current sources and have their unnencessary PMOS transistors disabled or totally removed, and are configured such that they discharge the replica bitline whenever the replica LWL driver asserts (which is every cycle). Like the previous method, different numbers of current sources can be configured to activate in able to equalize the delays as desired. The signal at the replica bitline then propagates up the replica column (whose memory cells are used only as pass transistors) and propagates to the single LWL driver that is active. This signal can then be used to quickly deactivate the LWL driver and stop the memory cells from further discharging the bitlines.

Figure 2-41(d) shows how this scheme can be integrated within the entire scheme, while Figure 2-41(e) shows the addition of padding rows to minimize the PVT variations between the regular memory cells and the replica cells caused by the replica being in the outer edge of the array.

Whatever scheme is used (and for that matter, whatever control circuitry), it is important to emphasize that the basic concept of these schemes is the generation of a control signal that can reliably track the development of the bitline voltage differential, and hence can be used to control the proper activation of the sense amps for correct operation and the deactivation of the decoder to generate a pulsed word line for lower power consumption

The correct generation of the sense amp enables allow the sense amplifier to function correctly and speed up the sensing of the voltage differential across the bitline. For a single-level sensing hierarchy, the output of the sense amp is typically a full-voltage swing output. It is typically desired to latch this output so that its value is retained until the end of the cycle (and some time after). Without a latch, the value is lost when the sense amp is precharged.

The data is gated on to the latch only after the sense output has stabilized, to prevent a glitching in the output data (which may or may not be critical depending on the downstream circuits). Sophisticated feedback information like the replica technique used for bitline control is unnecessary to control this process since sense amp characterization is easier and more tolerant of PVT variations compared to the minimum-sized pulldowns of the memory cells. Consequently, simple gate delays can be used to generate a delayed version of the sense-enable signals to gate the sense-amp data to the output latches.

To prepare the SRAM for the next access, the dynamic nodes within the SRAM circuits have to be precharged back to their original values. Referring back to Figure 2-39 and Figure 2-38, the END signal can be derived from the signal present at the replica bitline. This END signal then deassertsϕ, starting the precharge phase. This ensures that the address decoder precharge is started only after it has performed its funtion and the selected bitlines have developed sufficient differential.

The END signal can also be used to directly derive the precharge signals of the bitline (including the replica) and the sense amplifier. Precharging of the bitlines (and the assumed early reset of the LWL drivers), also act to precharge the replica bitline back to its high initial state. The complete process is summarized in a timing diagram shown in Figure 2-42.

The write operation shares many of the control sequence used for the read operation, especially the decoder and the replica technique. The main difference, aside from the obvious enabling of different sets of muxes and not enabling the sense amp and data latches, is that the data to be written has to be applied to the bitlines much earlier in the access, even before the local wordline has asserted. This ensures that the moment the access transistors of selected cells are enabled, the full-swing differential is available to flip the cross-

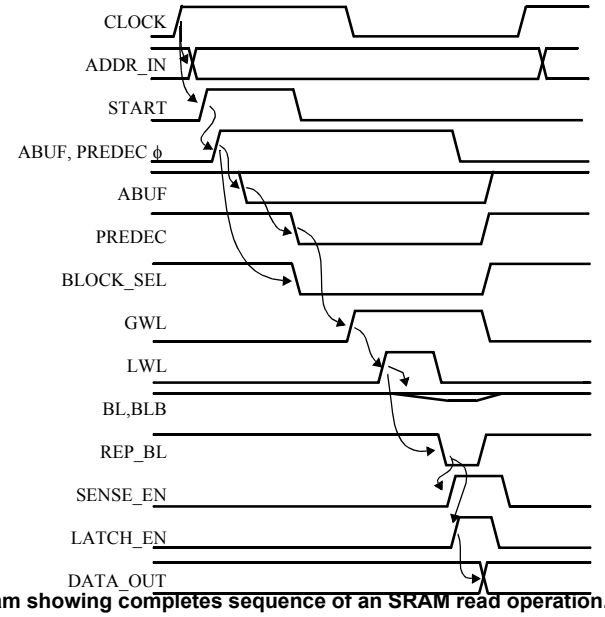


Figure 2-42: Timing diagram showing completes sequence of an SRAM read operation.

coupled inverter of the memory cell (if necessary). This requires that the SRAM receive write data early in the access. For most pipelines that buffer the write data, this requirement isn't a problem because the data is easily available from the write buffer at the early part of the clock cycle. In situations where the data is not available early (like in cases where it is computed in the same cycle), care has to be taken to delay the start of the SRAM write access until it is ensured that the data will be available at the right time.

It must also be ensured that memory cells are properly written during the short time that the wordlines are on. The width of the wordline pulse will be the same as in the read access since the behavior of the replica bitline will be the same for both types of accesses.

Lastly, the discharge of the replica bitline eventually triggers the precharge of the bitlines. In the case of systems with separate write precharge devices (as shown in Figure 2-36), both precharge devices are enabled to help the fully discharged bitline recover faster.

Cache Implementation

Simple Caches. After discussing the implementation of SRAM blocks, we could now use these memory primitives as building blocks for constructing caches. As shown in Figure 2-43, the tag and data part of the cache can be made up of appropriately sized memory arrays. Adding in the complete support circuitry to simple SRAM storage elements results in a complete, functional cache subsystem.

For an N-way associative cache, we use N tag-data pairs (note that these are logical pairs, and that they are not necessarily implemented in the same memory array), an N-way comparator and an N-way multiplexer to determine the proper data and to select it appropriately. For systems that do not employ some form of virtual addressing, the TLB is optional but otherwise, it is needed, especially if physical addresses are stored in the tag area. Also, although the TLB access here is shown to be performed in parallel with the cache tag and data access, it can be performed at any time as long as the translation is available in timefor use by the comparator.

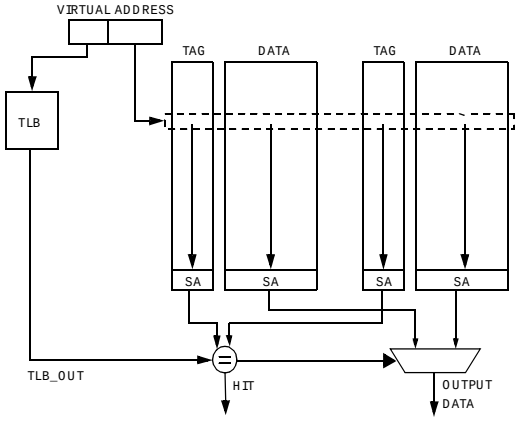


Figure 2-43: Cache block diagram.

In the figure, the cache controller has the responsibility of keeping track of cache operations and accesses to implement additional cache specifications like write-back/write-through behavior. It is also responsible for facilitating other functions like multiporting through banking to control each access and keep track of bank collisions. Lastly, the cache controller is responsible for interfacing to the lower level of memory when the access misses in order to fill the cache. Although caches can be implemented in a myriad of different ways, this simple cache implementation serves as a usable, functional cache design.

Processor Interfacing. Figure 2-44 shows two typical ways of interfacing a microprocessor to a data cache. Figure 2-44(a) shows where the cache could connect to an in-order processor pipeline. It shows the address, control and data being supplied to the cache stage by pipeline registers, and cache access is assumed to fit in one processor cycle, providing result signals (like HIT) and the data, in case of a load. the result signals are then used by the pipeline control circuitry to decide whether to stall the pipe depending on whether the access is a hit or a miss.

Alternatively, Figure 2-44(b) shows where and how the cache could fit in an out-of-order execution pipeline. The figure shows the internals of the load-store unit, and with the assumption that the proper structures exist outside this unit that allow for non-dependent instructions to execute out of order, this unit operates independently of the other functional units, and cache misses do not need to stall the processor core (unless during extreme cases where internal data structures within the load-store unit have filled up and cannot accept new instructions)

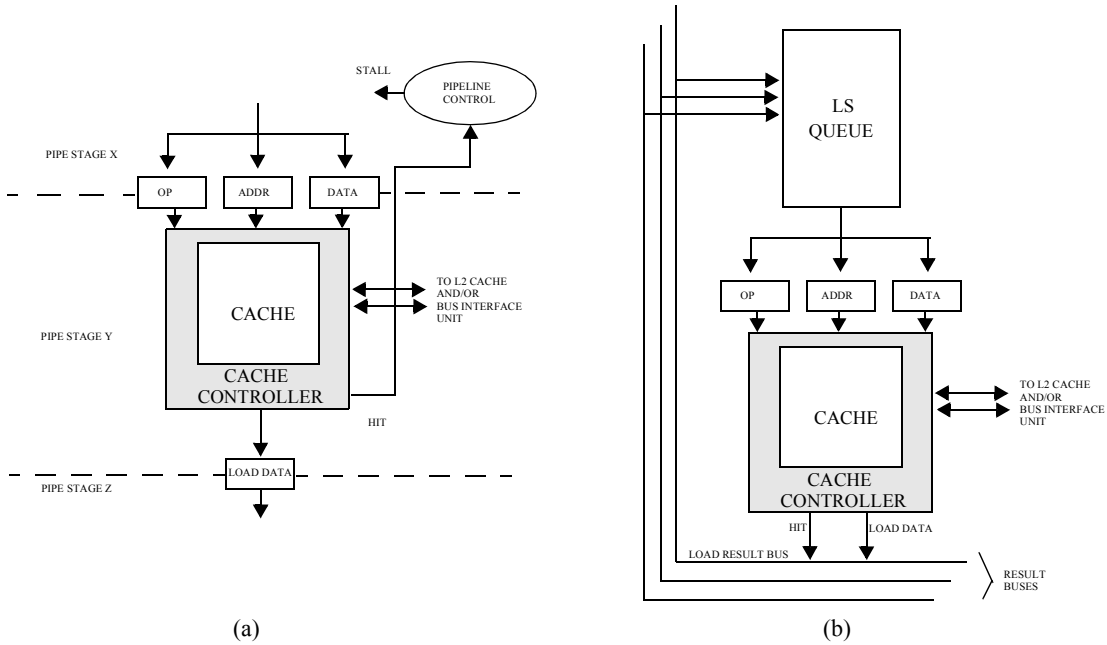


Figure 2-44: Cache-processor interface for an (a) in-order and (b) an out-of-order processor.

The LS unit typically includes a load-store queue used as a holding tank for memory instructions. When a load or store is cleared to access the cache (i.e. it has all its needed operands and doing the access will not cause any memory hazards or inconsistencies), information is retrieved from the load/store instruction and used to access the cache. The results of the access are then used to update the necessary process structures to reflect the the miss (in some implementations, it is transferred to another structure inside the load-store unit that contain accesses that missed, and in some cases, stores that haven't been retired), and is made to wait and retry the operation.

Multiporting. Caches can be multiported using different methods. True multiported caches employ multiported SRAMs that are especially designed to allow concurrent accesses to any location. Two examples of true multiported memory cells are shown in Figure 2-45.

Although using true multiporting allows relatively simple control of the SRAM, the addition of multiple access transistors for each port results in a very significant increase in memory area. Aside from the area increase, this also adversely affects the delay because of wire length increase within the memory.

An alternative to true multiporting is the use of multiple independent banks for the cache, where each bank is implemented as a simple single-ported cache. This configuration can satisfy multiple cache accesses as long as the accesses are to different banks. This configuration is shown in Figure 2-46 for a 2-way associative cache with four banks.

This configuration can perform a maximum of four R/W accesses concurrently. The main consequence of this configuration is the additional complexity and intelligence required in the cache controller to manage the control and I/O of each individual bank, along with keeping trank of bank conflicts that may arise. Even with this complication, the benefits of true multiporting often does not justify the drastic increase in cache size, making the banking approach more popular.

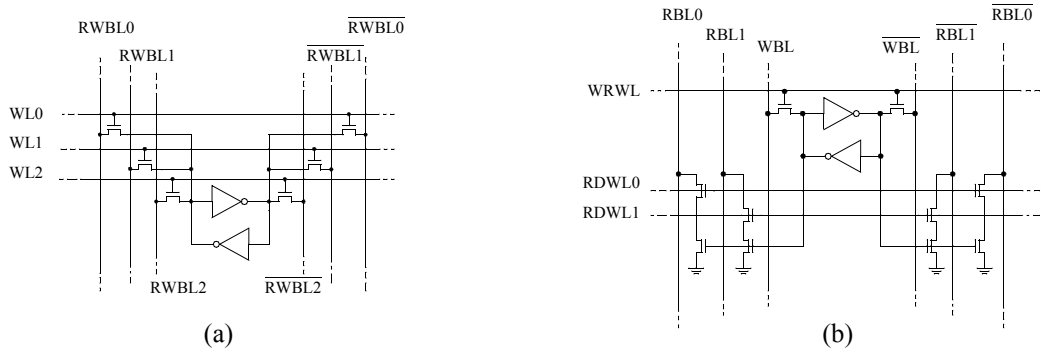


Figure 2-45: Multiported memory cells. (a) 3 R/W ports, (b) 2R 1 R/W port.

2.4 Some power solutions and their limitations

This section describes some of the leakage solutions that have been proposed in the literature. Some of these solutions have proven reliable and robust enough to be incorporated into commercial mass-produced integrated circuits, while some have yet to make the leap.

These solutions are mostly static power solutions, since the background on cache and SRAM design already contain numerous techniques that are designed to reduce dynamic power. addition, many of these static power solutions were initially targeted to remedy subthreshold leakage problems but have been shown capable of being extended to also reduce gate leakage.

2.4.1 Multi-Vt Solutions

One way to increase the transistor threshold voltage is to use process-level techniques to fabricate devices that have higher thresholds. By increasing the transistor's Vt, the subthreshold leakage is decreased significantly [1], but this is done at the expense of reduced current drive resulting in increased delay. To achieve maximum power reduction without significantly affecting the system speed (if at all), we recognize that only the devices in the circuit's critical paths directly affect the clock frequency. Consequently, devices that are off the critical path (typically comprising the majority of devices in random logic) can be designed to have high thresholds to maximize reduction without any sacrifice in speed, while devices on the critical path are designed to have lower thresholds, maintaining their fast speed at the expense of increased subthreshold leakage.

A typical design flow to perform partitioning of devices utilizing different threshold voltages is to initially assign high thresholds for all devices. A preliminary timing check can then be performed, and paths

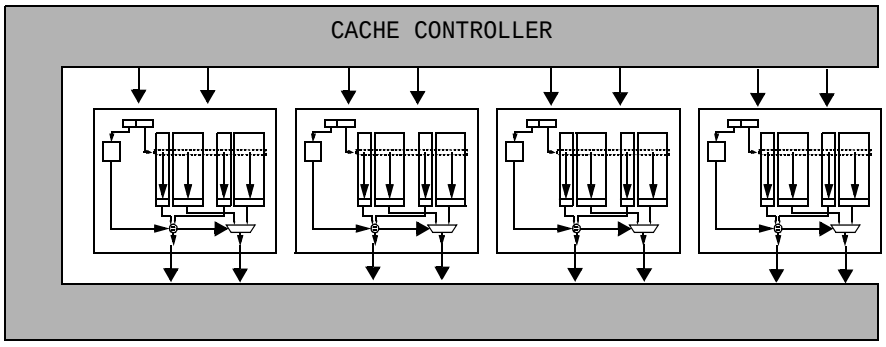


Figure 2-46: Multiported cache using banking. In this case, the cache is implemented as four banks of identical 2-way associative, 1 R/W ported cache.

1. An order of magnitude decrease for every 100mV increase in Vt, assuming a subthreshold slope of 100mV/decade

that are found to have insufficient slack (i.e. paths that have too much delay with reference to a target clock period), then have some of their HVT devices swapped out for LVT devices, and this flow is iterated on until timing closure has been achieved (i.e., all paths have sufficient slack -- the target clock period has been achieved). Note that if some paths still have timing failures even after all devices in that path have been swapped for LVT devices, then the path has an inherent problem that is not related to device threshold voltages and in this case, the problem is solved in some other way (e.g. conventional techniques to speed up circuits like rearranging logic, upsizing some devices to gain speed, using a faster circuit topology, and in the worst case, rearchitecting the path).

For simplicity, device replacement in static logic usually means both the NMOS and PMOS networks are replaced with the corresponding gate using devices with a different Vt. This is due to the widespread use of static logic in standard cell design where cells have been pre-characterized and it becomes cumbersome to design cells with multiple combinations of threshold voltages (where every single combination has to undergo extensive characterization).

Dynamic logic, on the other hand, is often used in custom designs where the circuit designer can afford to do extensive characterizations (this is in fact, required -- to make sure that the circuit works even under worst-case operating conditions), and hence makes arbitrary device swapping feasible. Of course, placement of HVT and LVT transistors have to be done intelligently to ensure maximum speed at the lowest possible consumption (or any desired tradeoff between the two factors).

Figure 2-47 shows a typical domino XOR gate with high-Vt and low-Vt devices. Some circuits are designed to favor a specific transition or edge, even at the sacrifice of the other edge. This is true for typical domino gates, where the transition that changes the state from its precharged value is favored. In the case of the domino XOR, the precharged state of the output is low (since the internal dynamic node is precharged

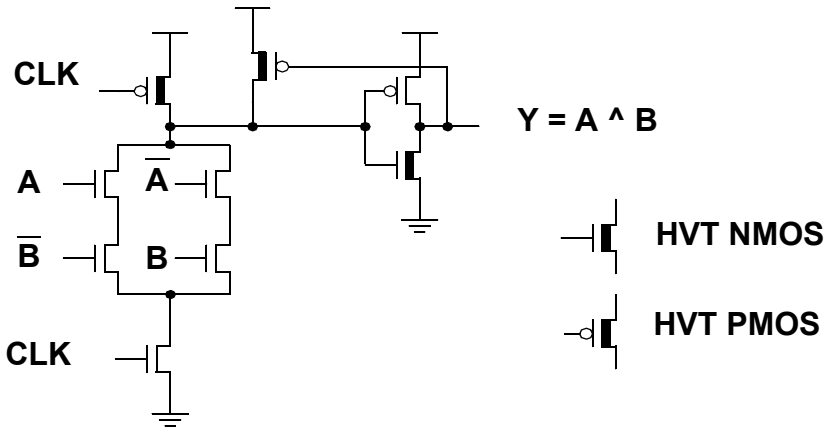


Figure 2-47: XOR dynamic domino logic favoring the rising edge at the output by insertion of HVT transistors in devices that do not contribute to this rising edge. All other transistors are LVT to guarantee speed.

high, and this value goes through an inversion), so the output rising edge is favorde. The transistors that do not directly contribute to this edge can be swapped for HVT devices without affecting the delay of the edge, and this replacement can be seen in the figure. All other transistors have low thresholds to maintain their fast speed. We must note that although the use of HVT devices in this way does not affect the favored transition, it does delay the other transition, which in this case is the transition going back to the precharged state. In most cases, this transition is off the critical path and as such, can be readily accounted for in the design. In addition, the precharge for each dynamic gate in the chain is done simultaneously (initiated by a single precharge signal) and does not propagate from one gate to the next (the way inputs to a logic chain have to propagate from one gate to another until it reaches the output), which makes sure that the additional delay resulting from use of HVT devices in this way are not cumulative.

Although the use of HVT devices reduces the leakage current to some extent, the resulting leakage is dependent on the inputs to the gate and hence the leakage reduction is not maximized. A more efficient sleep mode technique for domino logic that uses additional sleep switches has been proposed [Kursun2004] that maximize the benefit of the inserted HVT devices. This technique is shown in Figure 2-47, showing the additional sleep switch inserted at the output. At sleep mode, the clock is gated ON (the foot transistor is enabled, while the precharge transistor is off), and the sleep transistor is also enabled. In this manner, all leakage currents are forced to flow through the HVT devices, maximizing their effect. This technique has been claimed to provide 461X to 830X leakage reduction in a prototype CLA adder compared to the all LVT implementation, while the simple HVT swap as shown previously only provides a 1.2X to 2.8X reduction.

The application of different threshold voltages to random logic is relatively straightforward since critical paths are well defined and easily isolated, and solutions almost always exist where leakage can be

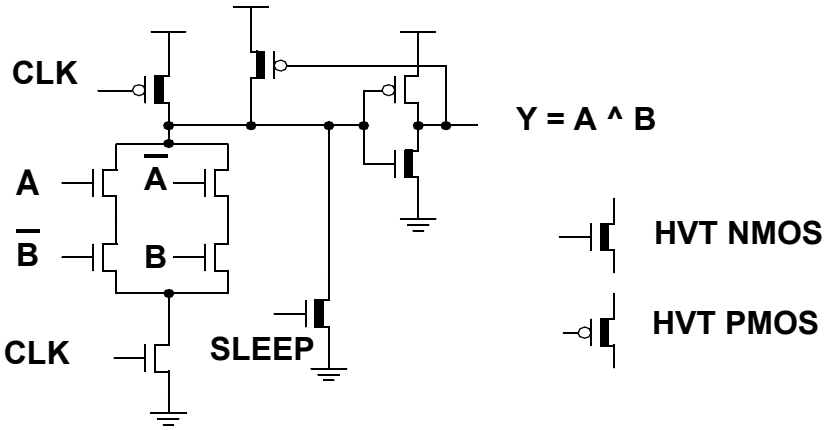


Figure 2-48: XOR dynamic domino logic with an sleep switch to maximize the subthreshold leakage reduction provided by the HVT devices. Activating the sleep switch forces the subthreshold leakage current to flow only through the HVT devices.

reduced without any adverse effect in speed. This is not generally true for memory cells like register files and caches (comprised of SRAMs).

An SRAM's parts can be divided into two classifications -- the actual storage devices or memory cells that remember the stored data, and the rest of the circuits which are used to access these storage devices. Although the same principles of HVT and LVT swaps can be applied to the latter class of circuits, application to the former class is more limited and difficult. The difficulty is due to the fact that memory cells are designed to be as similar to each other as possible to ensure similar behavior throughout the entire array. Unfortunately, the critical path goes through a subset of the memory cells, but the cells in the critical path are difficult to speed up without causing other problems within the array once the array homogeneity is violated. This usually means that any speed optimization to the memory cells in the critical path also have to be applied every memory cell, most probably resulting in a huge increase in leakage current. Conversely, leakage optimization of the non-critical memory cells also have to be applied to the cells in the critical path, most probably resulting in additional delay.

Since the reduction of static power dissipation in memory is critical, it is often acceptable to add additional delay to the critical path as long as the leakage current is reduced. In this case, the problem becomes the proper choice of HVT devices in places that will significantly reduce leakage with mostly minimal circuit slowdown. Different memory cell configurations using a combination of HVT and LVT devices have been proposed in the literature, including the configurations shown in Figure 2-49.

The main disadvantage of the multi-Vt approach, aside from the possible slowdown, is the additional process complexity and cost needed in fabricating devices with diffrent threshold voltages. Different threshold voltages require additional process steps, which add to the cost of the system and, in addition, inevitably lowers the die yield. Approaches that use single-Vt devices still have an advantage in this respect, but as technology scales further and subthreshold leakage contribution to power increases even more, use of multi-Vt circuits is going to almost be necessary, and the challenge would be how to maximize its benefits while minimizing system slowdown.

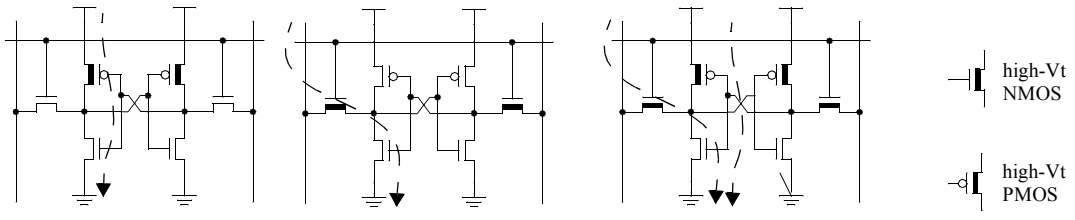


Figure 2-49: Different memory cell configuration using a combination of LVT and HVT devices. The subthreshold leakage currents that each configuration is designed to reduce are also indicated by the arrows.

2.4.2 Forced Stacking

The transistor stack effect has been shown to reduce the subthreshold leakage [Halter1997, Ye1998, Chen1998]. In Figure 2-50, a two-high NMOS stack with both transistors turned off has a much smaller leakage current compared to the single NMOS transistor. Among the reasons is the non-zero source voltage of the top NMOS transistor (because of the voltage drop across the bottom NMOS), resulting in a reverse-biased source-body terminal of the top NMOS and an increased threshold voltage.

This effect is currently being utilized by many solutions, among them a solution called forced stacking. This solution simply implies that the stacking effect is forced in the design by using multiple stacked devices in place of a single device. As shown in figure 2-51, two transistors are stacked to replace a single transistor, with both configuration employing the same function but with possibly different leakage and delay characteristics.

In forced stacking, W1 and W2 depend on which negative side effect of the stacking will be retained and accounted for. When W1=W2=2Wo, the effective drive strength of the two devices become equal, at the expense of presenting twice the input capacitive load (which may or may not have to be accounted for in the

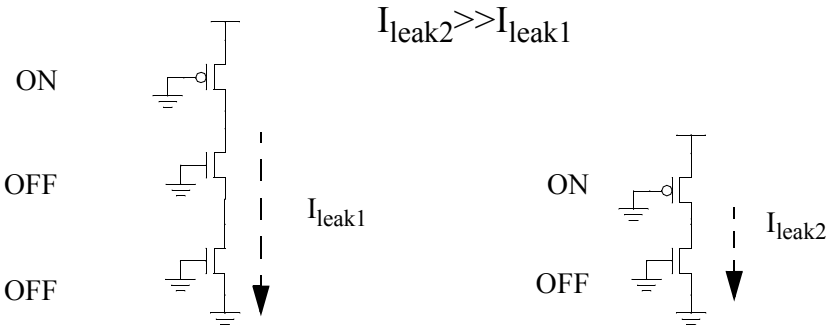


Figure 2-50: Transistor stack effect showing that the subthreshold leakage of a 2-high stack is much less than the leakage of a single transistor. The height of the stack is arbitrary, along with the actual specific topology of the circuit. Transistors are considered stacked if the source-drain of one transistor flows fully into the source or drain of another transistor.

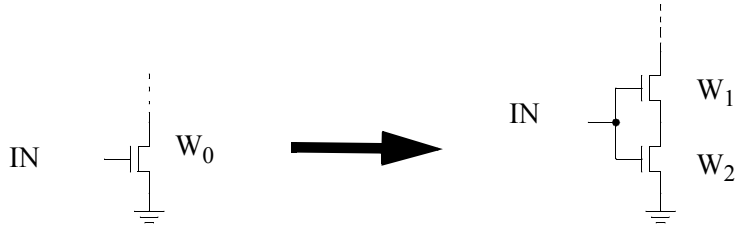


Figure 2-51: Forced stacking method where a single transistor is replaced by a transistor stack. The characteristics of the stack is dependent on the relative widths of the two topologies but the subthreshold leakage of the forced transistor stack will typically be larger than the original. Also, although the source of the original transistor is shown here as grounded, this is not necessary -- the concept is applicable to any transistor regardless of where its drain and source terminals are connected.

preceeding driver, depending on things like current path slack and/or edge rates). When W1=W2=0.5Wo, the input load remains the same, but the device only has 1/4 of the original drive strength. Although not applicable in every case, forced stacking provides another level of flexibility like threshold device swapping without any increase in process complexity [Borkar2004].

Forced stacking was originally targeted to address subthreshold leakage, but it may also in some cases reduce gate leakage, depending on whether the reduction in drain-to-source voltage (Vds) is enough to offset the possible increase in total gate width, as previously explained in the leakage mechanisms section.

2.4.3 Input sleep vector

Another solution that relies on the stacking effect to reduce leakage power is the use of input sleep vectors, which relies on the premise that logiccircuits will consume different static power depending on its present input. For a given circuit configuration, there will then exist an optimal input vector that results in minimal leakage and static power dissipation.

Consider the simple example of a 3-input static CMOS NAND gate, as shown in figure 2-52 showing two input vectors that result in the highest and lowest subthreshold leakage. In this case, the input vector <000> results in the least subthreshold leakage because of the 3-high NMOS stack (all of which are disabled), while the vector <111> results in the largest subthreshold leakage because of the three 1-high PMOS devices in paralel.

It is important to note that it is often not possible to present the optimum vector for every single gate embedded in the design because of restrictions in the outputs that a combination of logic can produce, but nevertheless, a given design will have an optimum sleep vector that results in lowest leakage.

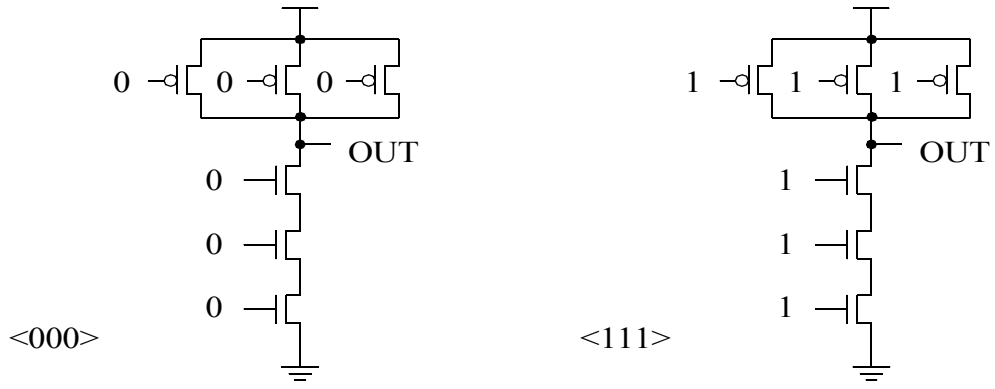


Figure 2-52: Static CMOS 3-input NAND gate showing two different input vectors. The vector <000> results in the smallest subthreshold leakage (because the stacking effect is maximized by the 3-high NMOS stack), while the vector <111> results in the largest subtheshold leakage (because of the three parallel PMOS, all of which are leaking). Accounting for gate leakage currents change which vector results in the least leakage current (most leakage current still results from <111>).

Sleep vectors were initially designed to reduce subthreshold leakage and, as such, the vectors were chosen to maximize the stacking effect. But the increasing contribution of gate leakage to the total leakage current changes the criteria in choosing the optimal sleep vector.

Going back to the 3-input NAND gate example, accounting for gate leakage [Rao2003] results in the vector <110> having the lowest total leakage. This is shown in figure xx, which plots the total subthreshold and gate leakage for every vector input to a 3-input NAND, and shows the contribution to the total leakage by the gate leakage and subthreshold leakage.

Use of sleep vectors to force the internal logic gates into a state that results in the least leakage current (and hence, least static power dissipation) is a big power win, especially for applications where blocks of logic can be put into idle or sleep mode for long periods of time. In cases where the sleep periods are not too long, the energy overhead involved in switching the inputs from its present state to the sleep vector may start to be significant compared to the total energy saved. In the extreme case where logic blocks are put to sleep every cycle that they are idle, the dynamic power dissipated during the switching would probably be larger than any static power saved.

2.4.4 PMOS-based dynamic domino logic

Another technique that attempts to reduce gate leakage current uses PMOS-based pullup networks for domino logic [Hamzaoglu2002] instead of NMOS-based pulldown networks, as shown in figure 2-54. This technique relies on the observation that gate leakage for enabled PMOS devices (using Si02 gate dielectrics) are an order of magnitude smaller than for the corresponding NMOS device [Yeo2000].

Although the inherent speed of the gateis not significantly impacted (assuminig proper PMOS sizing), since the increased capacitance due to the increased pull-up devices is balanced by the skewing at the output of the NMOS resulting in a much smaller PMOS, the gate itself will present 1.5x to 2x the gate load which may or may not be significant, depending on whether the gate is on the critical path or not.

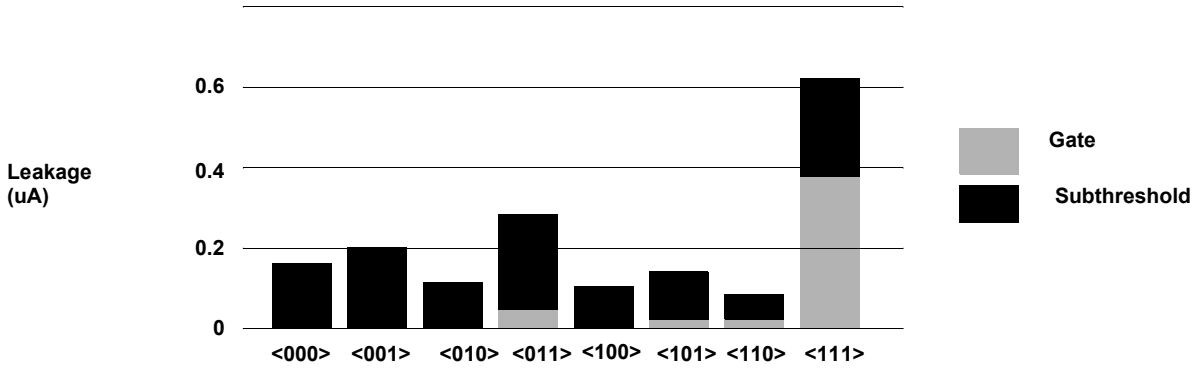


Figure 2-53: Total leakage power for a 3-input static NAND gate with different input vectors. The subthreshold and gate-leakage current component of the total power is also shown.

2.4.5 Body biasing

An alternative to implementing high or low threshold voltage devices by using additional fabrication steps is to change the threshold dynamically using body-biasing techniques. Depending on how the body, or substrate, is biased, the device threshold voltage can be increased for smaller subthreshold leakage, or reduced, for increased speed.

Applying a negative body voltage, or reverse body-bias (RBB), increases the device Vt by making it harder for the device to achieve strong inversion and induce a conducting channel in-between the source and drain terminals. The Vt increase causes a corresponding decrease in subthreshold leakage and the static power dissipation. Increasing the amount of RBB causes greater reduction in subthreshold leakage but this effect saturates with increasing amount of bias. In addition, although increasing RBB reduces the subthreshold leakage, it increases the junction leakage of the source and drain terminal becauses of the bigger depletion region. This results in an optimal RBB value that results in the smallest total leakage, as shown in figure 2-55.

Although RBB reduces the leakage current, the increase in Vt consequently results in increased device delay. For this reason, its simplest and easiest use is during the idle mode of circuits, where device performance is not a factor and the primary concern is to have minimal static power dissipation.

Instead of applying a negative bias, a positive body-bias, or forward body-bias (FBB) can be applied. The resulting reduction in device thresholds give better speed at the expense of higher leakage. A circuit designed in this manner typically starts with all HVT devices and FBB is used for selective devices during active mode to boost device performance.

An interesting application of these two techniques is the adaptive body bias technique (ABB), which is used to speed up slow circuits or reduce power in leaky ones. This recognizes the fact that fabricated designs exhibit varying performances because of process variations. In this manner, some circuits may be

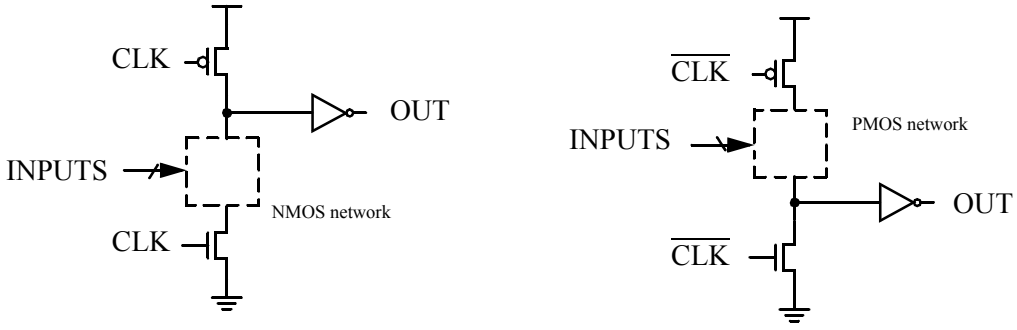


Figure 2-54: NMOS-based and PMOS-based domino logic, where the box represents an arbitrary NMOS pulldown or PMOS pullup network. NMOS-based domino are most often used because of the superior device characteristics of an NMOS compared to a PMOS of the same size, but PMOS-based domino logic has the advantage that PMOS have lower gate leakage than corresponding NMOS devices.

faster than others, and at the same time be more leaky, while others may be slow but exhibit less leakage. A specific example of ABB application [Borkar2004] is in performing frequency bin splits for fabricated die. It has been shown to produce very high yields (100% in some experiments), and produce good, distinct splits between low and high frequency bins (i.e., it's better to have similar characteristics instead of varying ones -- it's better to have a good split between low and high speed variants of the same design than have the frequencies be evenly distributed).

In summary, body biasing is a technique that is orthogonal to a lot of solutions in reducing leakage. RBB, specifically, seems to be a promising solution to reduce the static power in present and future circuits, especially since it doesn't have major disadvantages, it's main disadvantage (a minor one) being the additional complexity involved in designing RBB enabled circuitry (e.g. the need for an additional supply voltage, the need for additional control circuitry, etc.).

2.4.6 Supply gating

Leakage currents operate under the premise that a path exists where electrical charges supplied by Vdd is somehow discharged to ground, consuming power even though the circuit may not be active. A simple concept to reduce leakage, both subthreshold and gate leakage, is to take away the supply itself, either through the Vdd or Vss connections. This can be achieved by using brute-force switches that turn ON or OFF serving

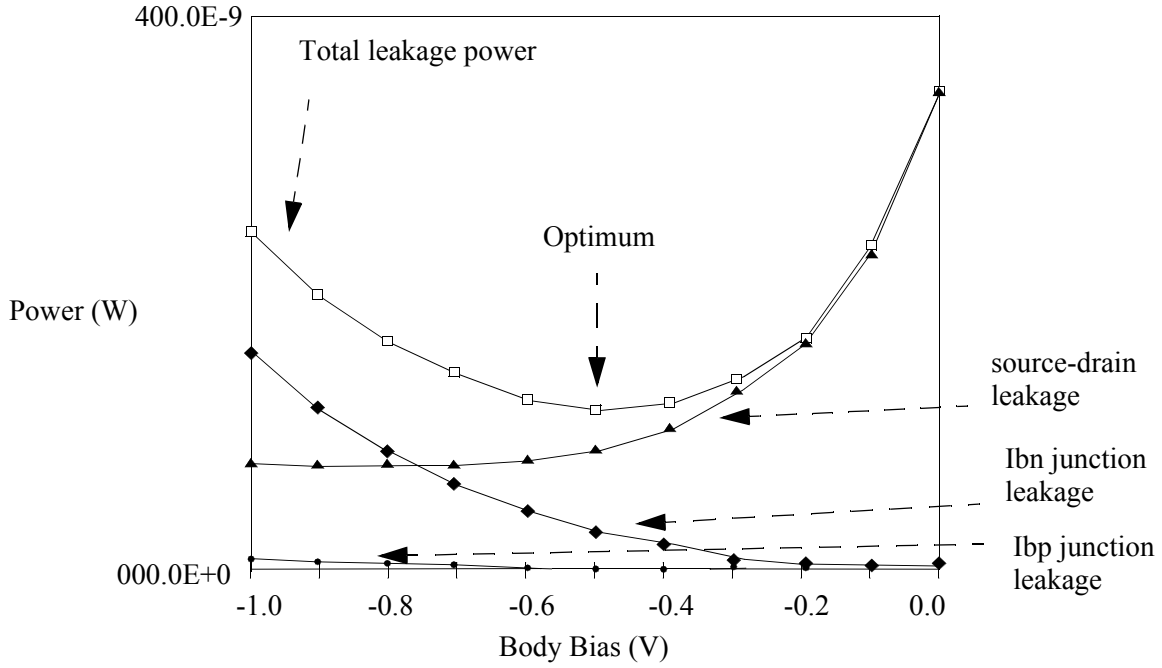


Figure 2-55: Plot of leakage currents in an NMOS device as a function of reverse body-bias. Even though increasing the reverse body-bias monotonically decreases the source-drain leakage (traditionally referred to as the subthreshold leakage), it also tends to increase some of the junction leakage across the drain-body junction. Combination of both leakage currents result in an optimum RBB of around -0.5V which results in the minimum total leakage current (excluding gate leakage).

to connect the power supply to the circuit to be powered. These switches can be implemented by PMOS or NMOS transistors (or both), as shown in figure 2-56.

This situation also relies on the stacking effect by having another transistor in series with any other transistor through which leakage currents can flow through. Of course, these power gating transistors have to be properly sized to provide enough current to the circuits that they are connected to in order to minimize any unwanted voltage drops across the gating device (which appears as power supply noise). This usually requires that these transistors have wide widths and therefore, occupy significant area.

If the leakage reduction resulting from simple power or ground gating is not enough, the gating transistors can be implemented with a higher threshold voltage to reduce the leakage even more. This technique is often called MTCMOS [Mutoh1995], or multi-threshold voltage CMOS in the literature, although the terminology is confusing and easily confused with any other circuit using more than one threshold voltage. Usage of HVT transistors result in a smaller gate overdrive and drive current and might require an increase in device width. A solution to this problem is theuse of a technique called boosted-gate MOS (BGMOS) [Mutoh1995] where a higher voltage is used to bias a high-Vt NMOS ground gater device to achieve a higher drive current without increasing the device area.

Supply gating is a technique that can be applied to both logic and memory, although it is often easier to apply to logic since there is less need to preserve the internal state within a logic block.

Another conceptually simple way to reduce the leakage current is to gate the power supply of the SRAM with a series transistor as shown in Figure 2-57. This is called the Gated-Vdd technique [Powell2000]. With the stacking effect introduced by this transistor, the leakage current is reduced drastically. This technique benefits from both having low-leakage current and a simpler fabrication process requirement since only a single threshold voltage is conceptually required (although the gating transistor can also have a high threshold to decrease the leakage even further at the expense of process complexity).

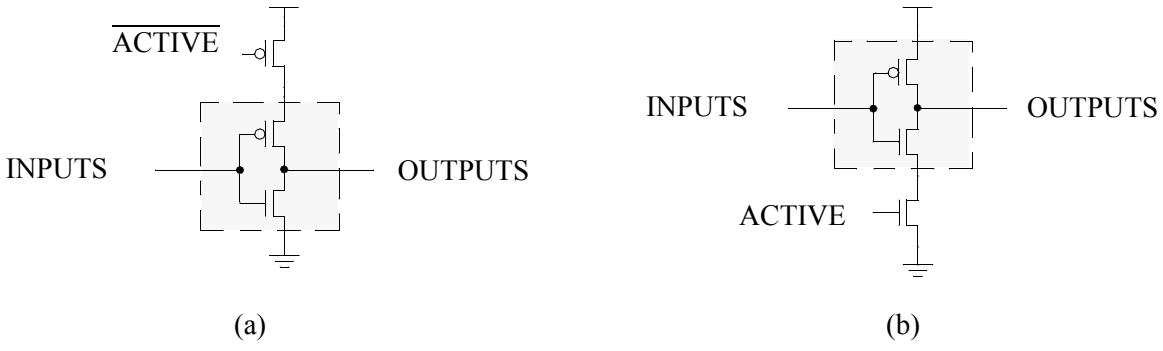


Figure 2-56: Power gating using PMOS and/or NMOS switches. (a) Vdd gating.(b) Ground gating. The shaded rectangle represents any arbitrary circuit where either its ground or power connection is connected to a gating transistor instead of the global supply rail. In this case, the inverter depicted in the shaded box is only a specific example -- the circuit can be as complicated or as simple as the design requires. Additionally, Vdd and ground gating could be used simultaneously, instead of separately as shown in these circuits.

Without any special tricks, the parts of the memory that have their power supply gated will lose their state whenever the gating transistor is turned off. When applied to caches, this technique can be advantageous if the current working set is smaller than cache size such that parts of the cache can be turned off without significantly adverse effect on performance. When the disabled part of the cache is accessed, the access misses, the addressed part of the cache is turned on, and the desired data can be retrieved from the lower-level of the memory hierarchy.

As an alternative to gating the power-supply and corrupting the stored state in the memory, a technique called Data-Retention Gated-ground (DRG) [Agarwal2003] can be used, where memory cells use a virtual ground that is connected to the actual ground through a gating transistor. This has the same effect on leakage current as power-supply gating. The main advantage of this technique is that, with proper sizing, the cells are able to retain their state even without being directly connected to ground. This technique is shown in Figure 2-58.

Aside from the reduced leakage current, this technique has the advantage that no other circuitry besides the gating transistor is required in the implementation. No extra control circuitry is necessary as the gating transistor can be controlled with the wordline supplied to the row (although the decoder drivers may have to be upsized to account for the additional gate load presented by the gating transistor).

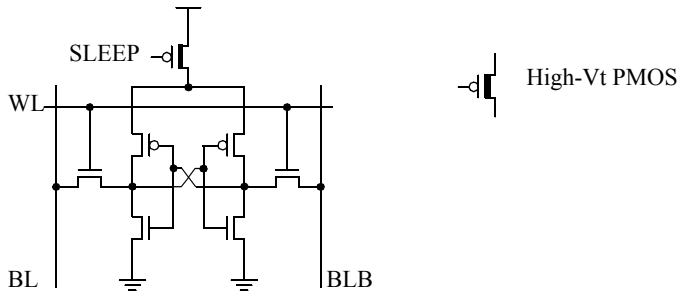


Figure 2-57: Gated-Vdd technique using a high-Vt transistor to gate Vdd. The sleep transistor drastically reduces the leakage current (both subthreshold and gate) during the SLEEP mode, but the state that the memory cell holds is lost. The gating transistor can also be implemented using LVT devices (with less reduction of the leakage.

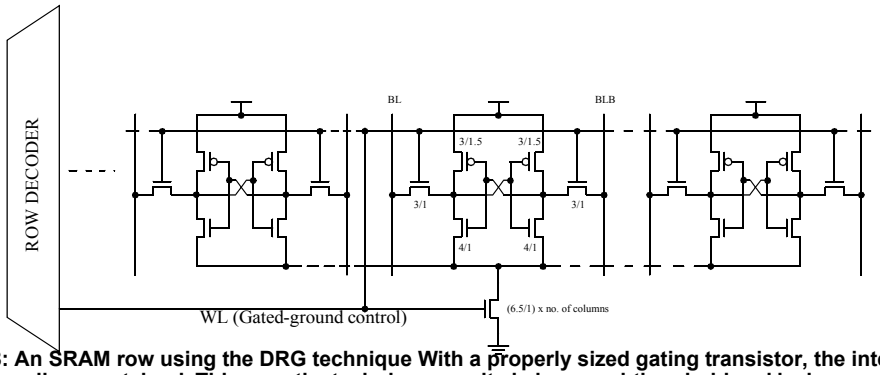


Figure 2-58: An SRAM row using the DRG technique With a properly sized gating transistor, the internal state of the memory cells are retained. This way, the technique results in lower subthreshold and leakage current without increasing the number of cache misses.

In cache applications, this technique has no impact on the cache hit rate since the data stored within the cache is retained even when the gating transistors are turned off. In addition, having the control use existing circuitry means that the cache controller isn't burdened with additional complexity to keep track of which parts of the cache are disabled, as may be necessary for the Gated-Vdd technique.

The main disadvantage of the DRG technique is its reduced tolerance to noise, as evidenced by its smaller SNM margin compared to a conventional implementation, as shown in Figure 2-59. The technique also has a small negative effect on both the delay and area of the circuit because of the introduction of the gating transistor. An additional minor disadvantage is the extra complexity in design required to carefully size the transistor within the memory cell to ensure that the data is actually retained even when the gating transistor is disabled. This design is somehow eased by a refinement of the DRG technique with the insertion of a diode in parallel with the ground-gating transistor [Agarwal2003], as shown in figure 2-60. The insertion of this diode provides an effect similar to transistor stacking in that it limits the voltage of the virtual ground to a maximum of the diode cut-in voltage (a few hundredths of a millivolt). This serves to reverse bias the source

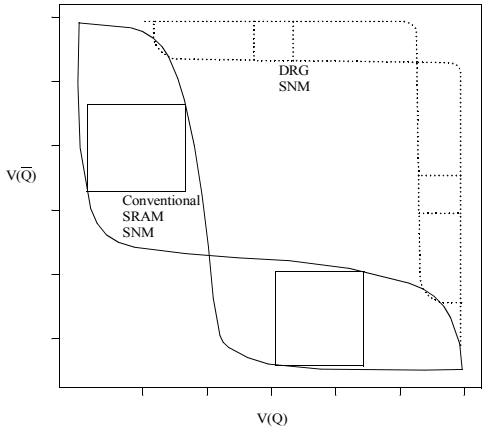


Figure 2-59: SNM of the data retentive gated-ground (DRG) SRAM compared to a conventional SRAM.

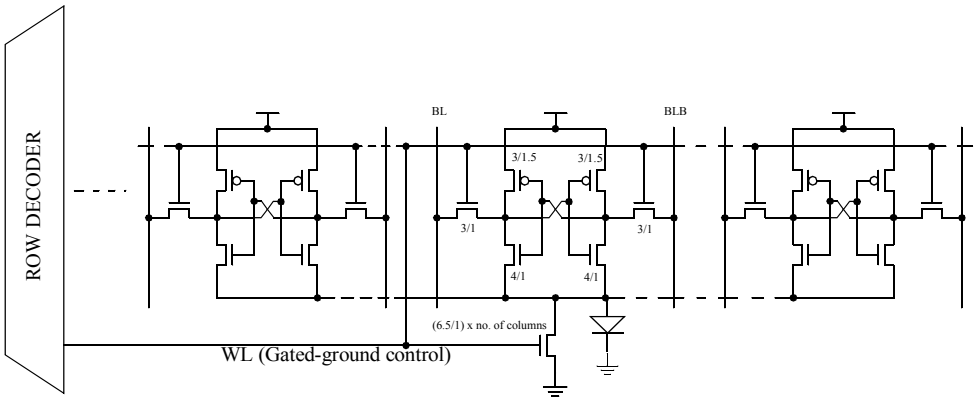


Figure 2-60: Refinement of the data-retention gated-ground (DRG) technique. This is essentially the same except for the insertion of the diode in parallel with the ground-gating transistor. This diode limits the voltage at the virtual ground to a maximum of the the diode's cut-in voltage, significantly improving the state-retention capability of the circuit even when the ground-gating transistor is disabled.

of the NMOS pulldowns, increasing the device threshold and reducing subthreshold leakage. In addition, the transistor drain to source voltage becomes smaller, reducing DIBL.

One last technique we present that reduces leakage power in SRAMs is the Drowsy technique [Kim2004]. This technique has similarities to both the gated-Vdd and DRG techniques discussed before in that it uses a transistor to conditionally enable the power supply to a given part of the SRAM. This method reduces leakage power by putting infrequently accessed parts of the SRAM into a state-preserving, low-power drowsy mode.

This technique uses another power supply with a lower voltage than the regular supply to provide power to memory cells in the drowsy mode. Leakage power is effectively reduced because of its dependence on the value of the power supply. This technique is shown in Figure 2-61(a).

This configuration is more tolerant to noise compared to the DRG technique because the cross-coupled inverters within the memory cells are still active and driving their respective storage nodes, which is not exactly true for the DRG technique. The main disadvantage of this circuit is the requirement of an additional voltage supply that has to be routed over the entirety of the cache arrays. The amount of space neeed for these connections will take up valuable routing resources, which is a very important consideration in memory circuits that are typically wire-limited. A refinement has been proposed [Mudge2004] that uses a transmision gate topology that allows the use of a single power supply, as shown in figure 2-61(b).

2.4.7 Other memory-specific techniques

The previous methods that have been discussed are applicable to both logic and memory (except for the obvious exception of memory-specific variations to the designs). In this section, we discuss optimizations target towards circuit structures that are prevalent in memory circuits.

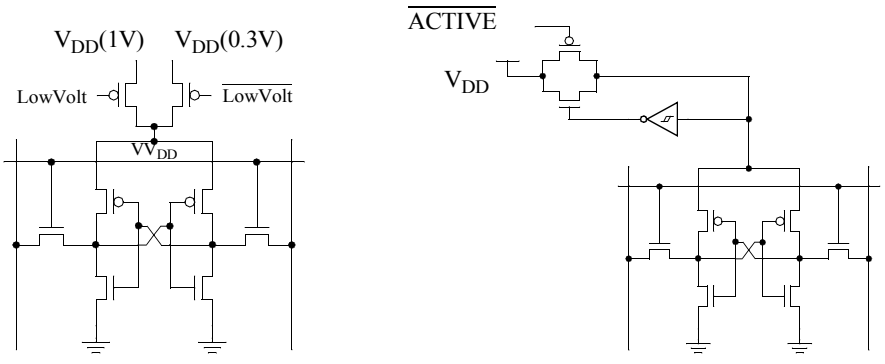


Figure 2-61: A drowsy SRAM cell. (a) Initial topology of the drowsy cell containing the transistors that gate the desired power supply, (b) refinement to the original method where only a single supply is now required.

Way prediction. Way prediction was designed to reduce dynamic power dissipation in set-associative caches, and it is included here mainly as background information for some of the next solutions.

Given the 2-way set associative cache in figure 2-62, the two ways are often designed to be independent of each other and are typically implemented as two independent direct-mapped cache. In a conventional cache, the required data could reside in either of the two ways, so both are accessed simultaneously and only near the end of the access is a compare operation done to determine whether the access is a hit or a miss. In way prediction, a separate predictor structure exists that givesa hint on which cache way contains the reqruied data. This prediction can then be used to access only the predicted way, preventing the other ways from dissipating dynamic power. A wrong prediction requires reaccessing the cache, resulting in additional latency. As long as the predictor is reasonably accurate, the decrease in power may justify the reduction in performance.

Way halting. The main objective of way-halting is similar to that of way prediction in that it attempts to limit the number of ways accessed in a set-associative cache to reduced ynamic power consumption. A big difference is that instead of relying on a predictor, way halting performs early miss-detection to determine which ways are absolutely certain not to contain the desired location. These ways are then prevented from being accessed and dissipating dynamic power.

Early miss-detection is performed by offloading a small subset (e.g. 2-3 bits) of the tag bits, called the halt tags, from the main tag arrays to another memory structure that is accessed simultaneously as the main cache arrays. Careful design of the circuit makes it possible to perform the access of the halt tags, perform the

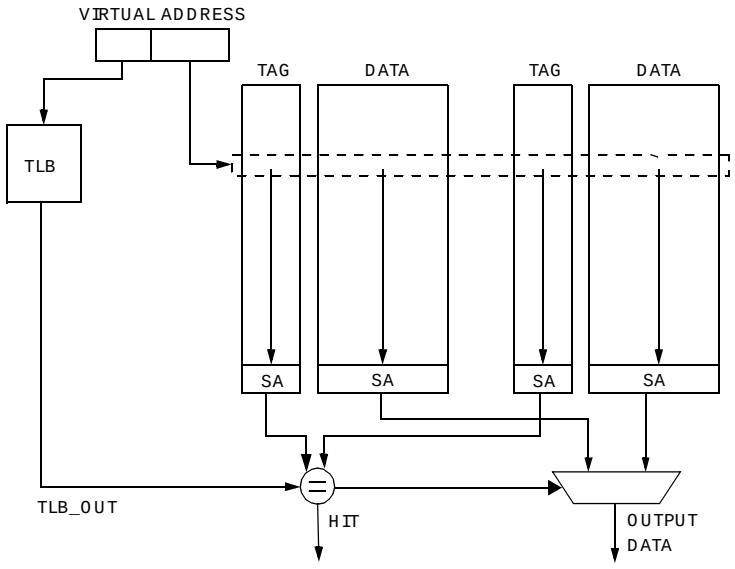


Figure 2-62: 2-way set associative cache.

required comparison, and produce a signal that gates the decoder output activating the parts of the cache, as shown in figure 2-63.

As with way prediction, this technique serves to prevent unnecessary dynamic power dissipation during unnecessary cache way accesses, but it does it in a way that does not affect the cache hit ratio whatsoever. As long as it is ensured that the dynamic power consumed by the early miss detector circuit is significantly less than the dynamic power saved by not accessing unnecessary ways, this solution is also a big power win.

Gated precharge. A technique that serves to limit the amount of static power dissipated because of bitline leakage is called gated precharge [Yang2003]. Figure 2-64 shows a number of cache subarrays whose precharge signals are gated by a control circuit. The subarrays are shown contain a single bitline-pair connected to a number of access transistors from memory cells, along with the necessary precharge circuits.

Assuming the bitlines are precharged high and all the wordlines are disabled, any access transistor whose other end is grounded (e.g. if the memory cell value is "0" on that side), will exhibit leakage and serve to discharge the bitline. If the precharge circuit is implemented as PC1 in the figure, the voltage drop caused by the leakage can be offset by sizing the precharge devices to account for the additional leakage drop, at the expense of slower bitline speed (due to the fight between the NMOS pulldown and the precharge device) and

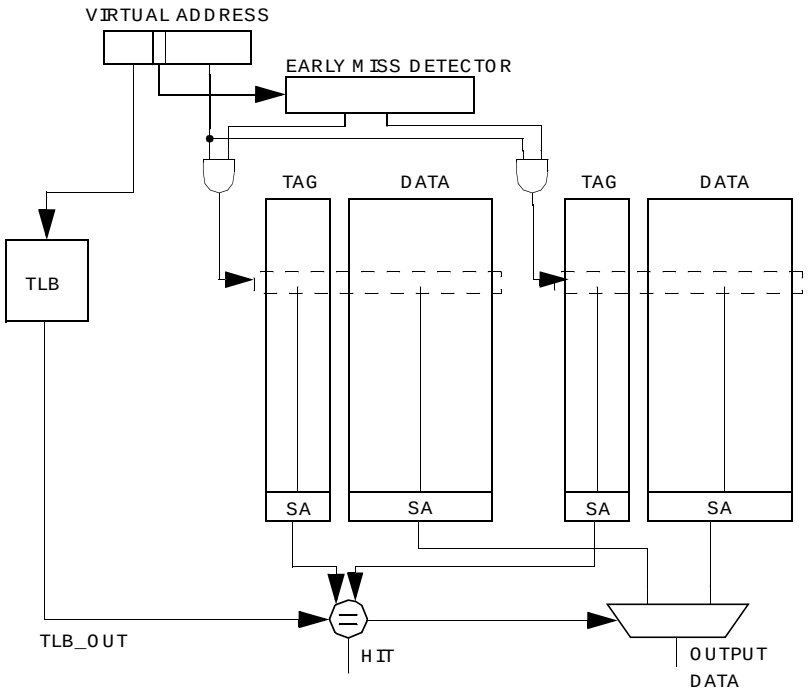


Figure 2-63: Way-halting scheme designed to reduce dynamic power dissipation in caches due to unnecessary way accesses. Instead of doing full tag comparison only after accessing all N words from an N-way cache, a small subset of the tag bits called the halt tag are put in a dedicated early-miss detector block. This block is accessed simultaneously as the decoding, and it provides an output produced just in time to gate the output of the decoders and prevent access to ways that certain not to contain the accessed location (as determined by the tag comparison using the halt tags).

continuous static power dissipation. The main advantage of this approach is that the need for rigorous matching of the precharge and wordline signals is not necessary (since there is actually no precharge signal).

If the precharge circuit is implented as PC2 in the figure and assuminig that the precharge signal activates every clock cycle to ensure that the bitline pairs have the proper voltage, the bitline speed becomes faster since the precharge device is disabled during the bitline voltage development phase and does not fight the memory cell pulling down the bitline. In this case, the timing of the precharge and wordline signals have to be designed carefully such that in no situation do they overlap enough to cause crowbar currents and worse, functional timing failure due to timing faults. In addition, static power is still significant because every bitline in the memory still leak and discharge away the bitline value.

The gated precharge technique is conceptually simple -- it attempts to predict which parts of the cache are likely to be accessed in the near future, and it proceeds to precharge these cache subsets as regularly as in the conventional way. Those parts of the cache that are deemed not likely to be accessed soon then have their precharge signals inhibited and gated off such that the precharge devices in these subsets are not anymore activated and the bitline leakage is allowed to discharge the bitlines to a steady-date value. The main advantage of this approach is that once the steady-state value of the bitline is reached, bitline leakage stops and becomes zero. Even if the prediction fails, the penalty is small (a 1-cycle penalty already being pessimistic).

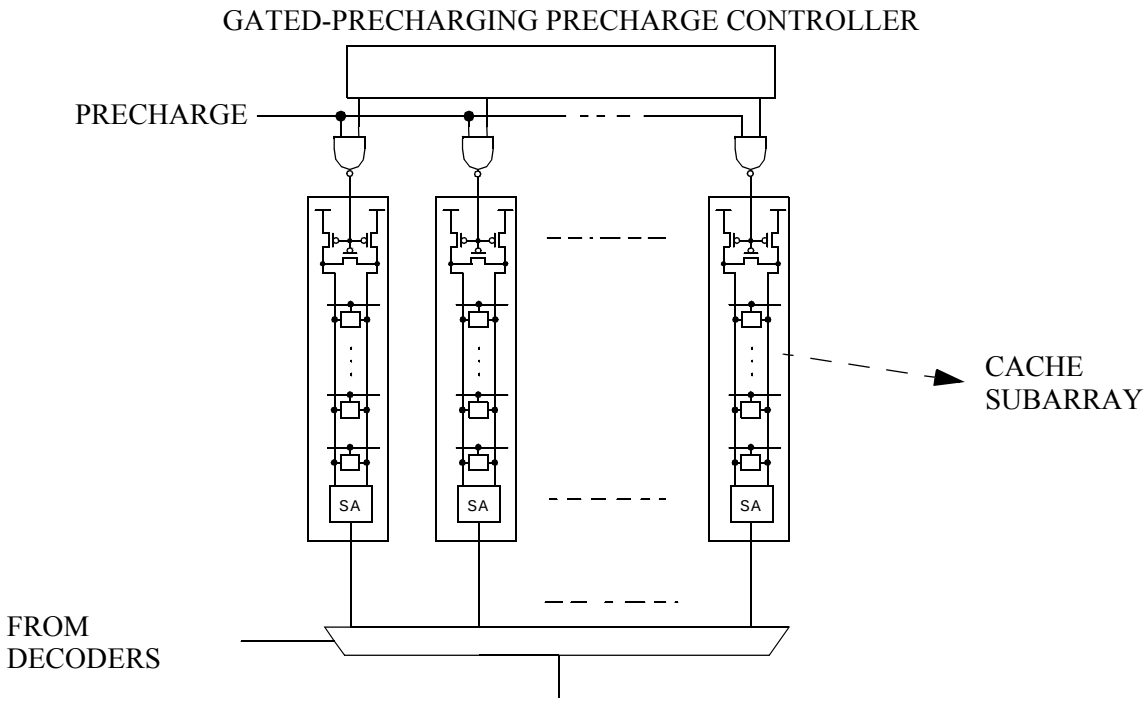


Figure 2-64: Gated precharge. This technique predicts which cache subarrays are least likely to be accessed in the near future, and disables the precharge signals for those particular subarrays. Gating the precharge off results in the discharging of the bitlines through the memory cells but once steady-state is reached, further leakage is minimal -- resulting in a big static power win as long as the prediction is reasonably accurate.

As long as the prediction is reasonably accurate, this method has been shown to significantly reduce the static power dissipation caused by bitline leakage.

CHAPTER 3 Cache Design Tools

3.1 Introduction

In this chapter, we provide an overview of the cache design tools CACTI, eCACTI and myCACTI by providing the basics and some details behind their operation.

We first discuss the operations of CACTI[1] and some of its assumptions. We then discuss how eCACTI extends and improves on the CACTI model. Finally, we show and demonstrate the improvements offered by our own flavor of CACTI which we call myCACTI. Subsequent chapters will focus on detailed simulation comparisons between these different design tools, along with further discussion on the features of myCACTI.

3.2 CACTI

CACTI was first published in 1996 [Wilton1996], and it was done as an extension to an existing cache design model [Wada1992]. Since then, CACTI has undergone at least two major revisions and improvements [Reinman2000, Shivakumar2001].

3.2.1 Basic Operation

At its simplest, CACTI accepts some input design parameters (as shown in Table 3-1), then performs design space exploration to produce the optimal cache implementation as described by the output cache implementation parameter sextet shown in Table 3-2.

Table 3-1: CACTI input parameters

Input Parameter	Use
C	Cache size in bytes
B	Block size in bytes signifying the number of bytes in a single cache entry
A	Cache associativity
TECH	Technology node in micrometers
$N_{subbanks}$	Number of cache subbanks

1. The CACTI version that this dissertation assumes is version 3.0. Going forward, CACTI v3.0 will be referred to simply as CACTI.

Table 3-1: CACTI input parameters

Input Parameter	Use
b_o	Number of bits of output data
b_{addr}	Number of bits of system address

Table 3-2: CACTI output implementation parameters

Output Parameter	Use
N_{dwl}	Number of segmentations of the wordline (Data)
N_{spd}	Aspect ratio control parameter (Data)
N_{dbl}	Number of segmentations of the bitline (Data)
N_{twl}	Number of segmentations of the wordline (Tag)
N_{tspd}	Aspect ratio control parameter (Tag)
N_{tbl}	Number of segmentations of the bitline (Tag)

To determine this optimal configuration, CACTI performs an exhaustive search of all the possible combinations of the implementation sextet of Table 3-2, scoring the performance of each implementation and returning the one with the best score (which is taken to be the optimal based on the optimization algorithm being used). Figure 3-1 demonstrates how CACTI uses these implementation parameter sextet affects the physical layout and implementation of a cache. The most basic implementation is shown in Figure 3-1(a), which is a direct-mapped cache with the data array parameters all set to 1 (A=Ndwl=Ndbl=Nspd=1). For caches that have too much loading on the wordlines, the wordline can be segmented such that the loading per local wordline will now be significantly less. This is shown if Figure 3-1(b) (Ndwl=2, A=Ndbl=Nspd=1). Alternatively, for caches that have too much loading on the bitlines, the bitlines can be segmented such that each bitline has its own sense amplifier and the loading in each bitline segment is significantly reduced. This is shown in Figure 3-1(c), with the parameters (Ndbl=2, A=Ndwl=Nspd=1). Figure 3-1(d) simply shows the same-sized cache, but instead of a direct-mapped cache, the figure shows a 2-way associative cache[2]. For designs that want to subdivide the two-way associative cache further, Figure 3-1(e) shows a two-way

2. CACTI, eCACTI and myCACTI all implement associativity in this manner, where a subarray contains data from all the different ways. An alternative implementation that is used in some design is that each way is implemented in its own independent array, such that the data of each way is kept separate from every other way, unlike this implementation where they are merged together into the subarrays.

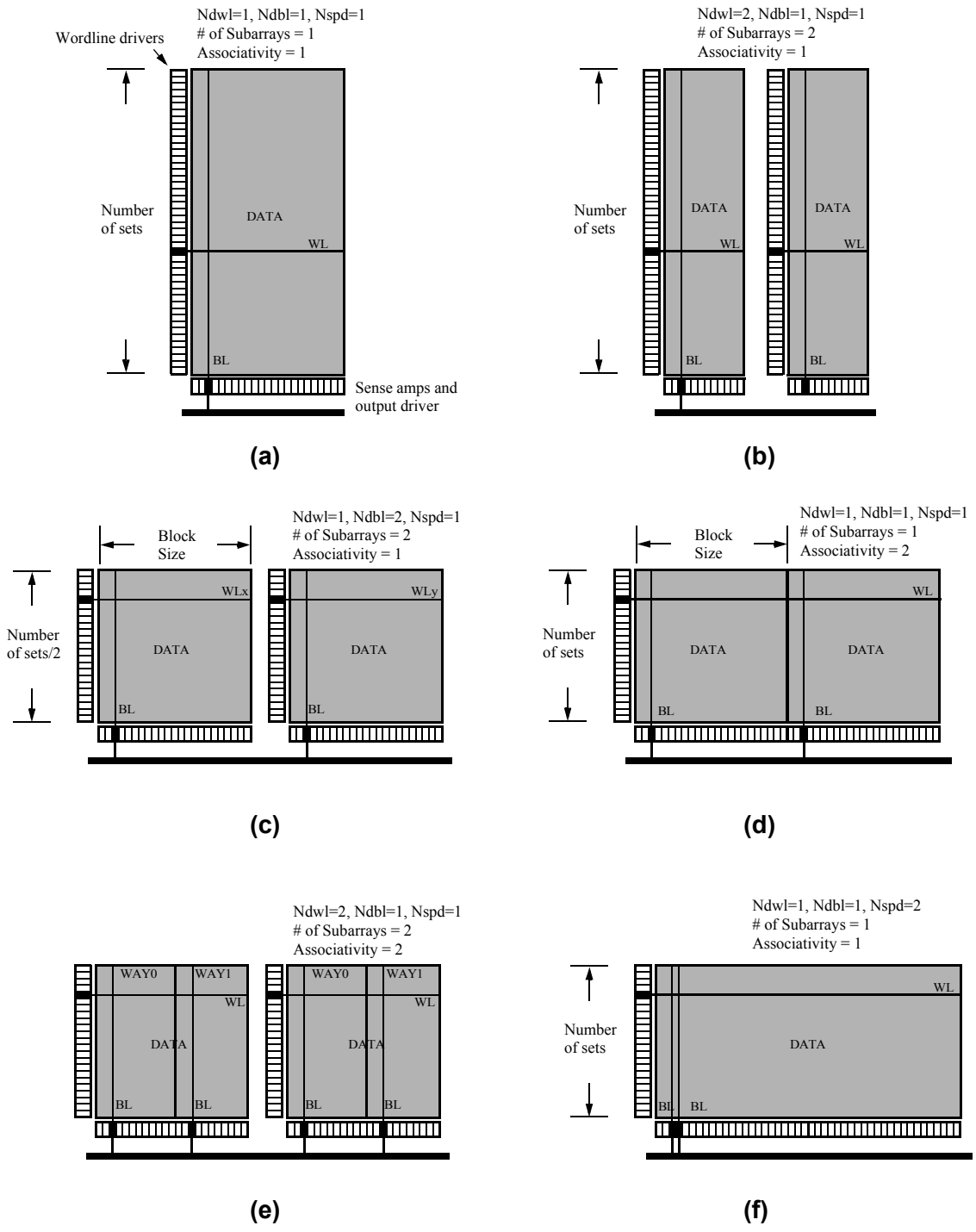


Figure 3-1: CACTI usage of implementation parameter sextet (Ndwl:Nspd:Ndbl-Ntwl:Ntspd:Ntbl). The parameters demonstrated are the data array parameters, but the usage for the tag array parameters are exactly the same. (A) Basic implementation showing direct-mapped with Ndwl=Ndbl=Nspd=A=1. (B) From the basic implementation, the wordline is segmented into two (Ndwl=2, Ndbl=Nspd=A=1). (C) From the basic implementation, the bitline is segmented into two (Ndbl=2, Ndwl=Nspd=A=1). This is almost the same as the previous, except the number of sets is halved. (D) From the basic implementation, the number of associativity is increased to 2 (A=2, Ndwl=Ndbl=Nspd=1). (E) From the previous two-way 1-subarray implementation, the wordline is segmented into two (A=Ndwl=2, Ndbl=Nspd=1). (F) From the basic implementation, the aspect ratio is adjusted by increasing the number of cells mapped to a single set (Nspd=2, Ndwl=Ndbl=A=1). Note that the activated bitlines shown are only for single bits.

associative cache where the wordline has been segmented into two, resulting in two subarrays. Note that each subarray contains part of both ways, although this is not strictly necessary. Finally, the CACTI-original parameter Nspd can be used as shown in Figure 3-1(e) if we desire to adjust the aspect ratio of the subarrays. This figure was derived from the direct-mapped figure of Figure 3-1(a), but with the lower half rows merged with the upper half rows such that each row now is connected to twice the number of memory cells (i.e. there are now twice more number of columns per subarray). This kind of aspect-ratio adjustment is typically useful for further adjustment of the vertical height of the subarray. This is useful since it allows an additional parameter to adjust the loading of the bitlines, which typically needs more attention because of the weaker drive capability of the memory cell, compared to the strong devices that are used for the wordline drivers.

3.2.2 CACTI cache structure

A high-level view of the cache structure implemented by CACTI is shown in Figure 3-2. Assuming a read access, the figure shows the address input entering the cache and being decoded by the address decoders for the data and tag array. Each address decoder then asserts a subset of wordlines that in turn enables all memory cells connected to it. These enabled memory cells then discharge the bitlines, developing a bitline differential voltage. Part of the address is then used to control read column multiplexers to choose the proper subset of bitlines to connect to the sense amplifiers to generate full-swing logic-compatible signals. At this point, the operation of the data and tag arrays diverge. For the data array, the outputs reach the output driver for a final series of multiplexing, where the output drivers are implemented as high-impedance drivers. This data does not reach the output until the output drivers have been selected and driven by the tag array. To accomplish this, the tag array performs a comparison of the tag retrieved from the tag arrays with the rest of the address inputs. Each way performs its own comparison, and these result of these tag compares are then provided to MUX drivers which are simply buffer chains that are responsible for driving the big devices of the output drivers.

The following subsections provide more details to the internal implementation of the CACTI cache.

3.2.3 CACTI address decoder

Figure 3-3 shows the decoder circuit implemented in CACTI. The figure shows an address input being buffered before being given to the predecoders. As discussed earlier, predecoders simply break up the AND function of the decoder into multiple stages to avoid using inefficient high fan-in gates to perform the ANDing in one stage. After the 3-to-8 predecoders have generated (through a simple NAND gate) 8 predecode lines per 3 bit of address inputs (e.g. for a 12-bit address, 32 predecode lines are generated), these predecode lines are then routed to the wordline decoders that perform a NOR function. In the figure, the DeMorgan's equivalent figure of the NOR is shown instead of the its conventional figure to emphasize that the logic we want to be performing in this case is an AND function. Note that the inversion of the previous NAND gate simply cancels out the inversions in the input, such that the combined predecoder-wordline decoder NAND-NOR simply

results in simple ANDing. Finally, the wordline signal generated by the wordline decoders are buffered by two additional stages before driving the final wordline.

CACTI implements all the logic in the decoder as simple full-CMOS static logic. Although this provides simplicity of design and verification, it does have many disadvantages, one of which lower speed. One example of why this is so is in the wordline decoder stage. As discussed before, this stage is implemented by a static NOR gate with fan-in of at least three (which could go as high as five). Because of the high fan-in and the full-CMOS implementation, the input capacitive load presented by each wordline decoder is significantly larger compared to a 1X inverter. Making it worse is the fact that the wordline decoder is a NOR

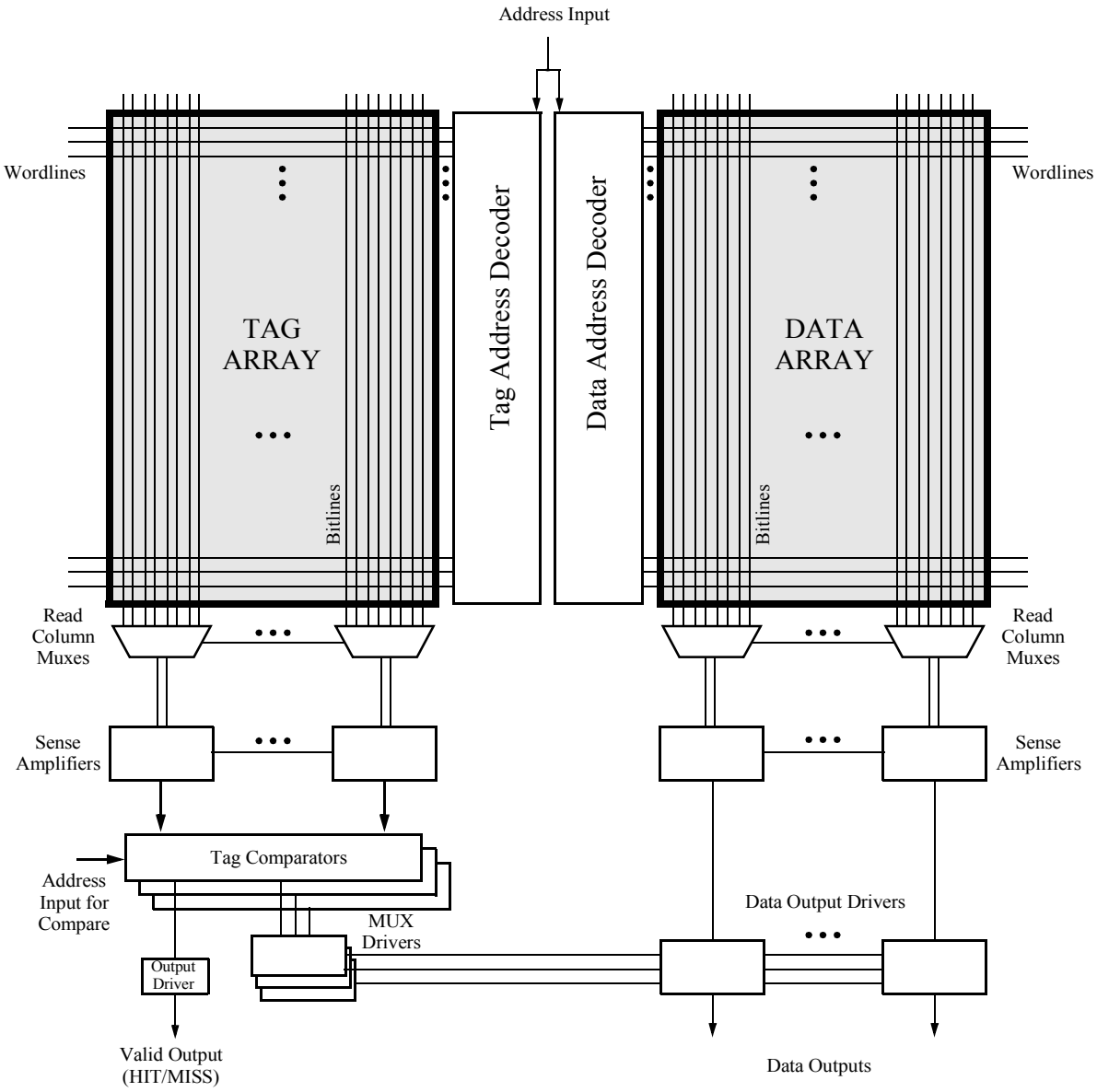


Figure 3-2: CACTI cache structure. This figure shows a high-level block diagram view of the cache structure used by CACTI. Note that even though the data and tag arrays are shown to be the same in the figure, the data array is almost always much larger physically compared to the tag array.

gate, which has a higher logical effort[3] compared to a NAND gate, increasing the input capacitive load even more. Many of these high fan-in gates then connect to the same predecode line, resulting in a huge load that the predecoder has to drive. Use of dynamic CMOS circuits can drastically cut down the logical effort of a gate. In addition, NOR-type dynamic CMOS circuits can implement higher fan-in gates without any penalty in logical effort. Consequently, significantly decreased loading requires much less driving capability, speeding up the operation.

Finally, CACTI supports run-time resizing of the final wordline driver stage to more optimally drive the particular wordline load for a given configuration. Given the number of columns connected to a wordline, a desired rise time is computed. Also given the total capacitance in the wordline (composed of the memory cell gate capacitances, the wire interconnect capacitance and the driver drain diffusion capacitances), an effective resistance can be computed to meet the desired rise time. CACTI then converts this effective resistance into a transistor width. This dynamic resizing, though, is limited only to the final stage of the wordline driver, and all other devices everywhere in the cache have a fixed size (determined at compile-time).

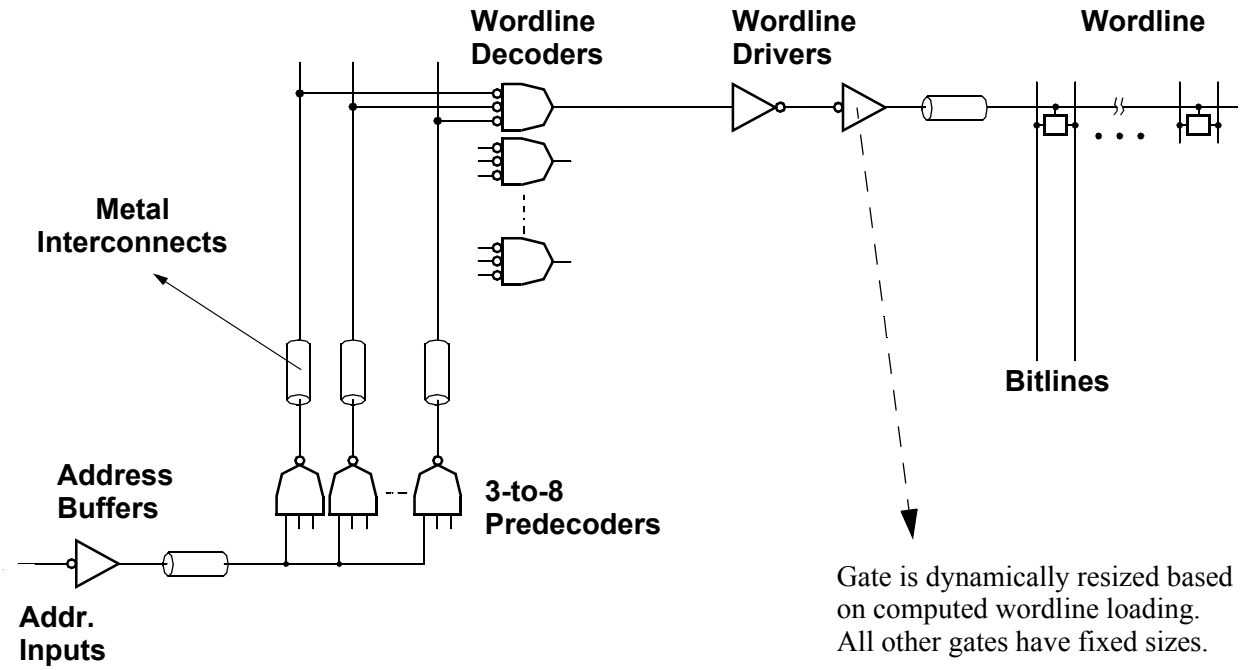


Figure 3-3: CACTI address decoder. All gates shown are implemented with full-CMOS static logic.

3. Logical effort is simply a measure of how much larger the input capacitance of a given gate is compared to an inverter of the same effective drive strength. This can be interpreted as a measure of the "penalty" involved in implementing a certain logic function, as opposed to the simple inversion of an inverter. Inverters have a logical effort of 1, while 2-input NAND gates have a logical effort of 1.333 and 2-input NOR gates have a logical effort of 1.667.

3.2.4 CACTI bitline circuits, tag compare and output drivers

Figure 3-4 shows the bitline circuits, tag compare circuits and output drive circuits modeled by CACTI. These circuits represent the path in the cache after address decoding is performed.

For the data array, assertion of one wordline activates all memory cells connected to it. This activation enables the access transistor in the memory cell, allowing the internal cross-coupled inverter latch to discharge the originally precharged bitlines through the latch side that is storing a logic zero. This discharge of the bitlines result in the development of a differential voltage. Before this differential voltage is sent to the sense amplifiers, another batch of address decoders called the column decoders[4] select the proper column multiplexers to enable based on the input address. Once these column muxes are enabled, the proper bitlines

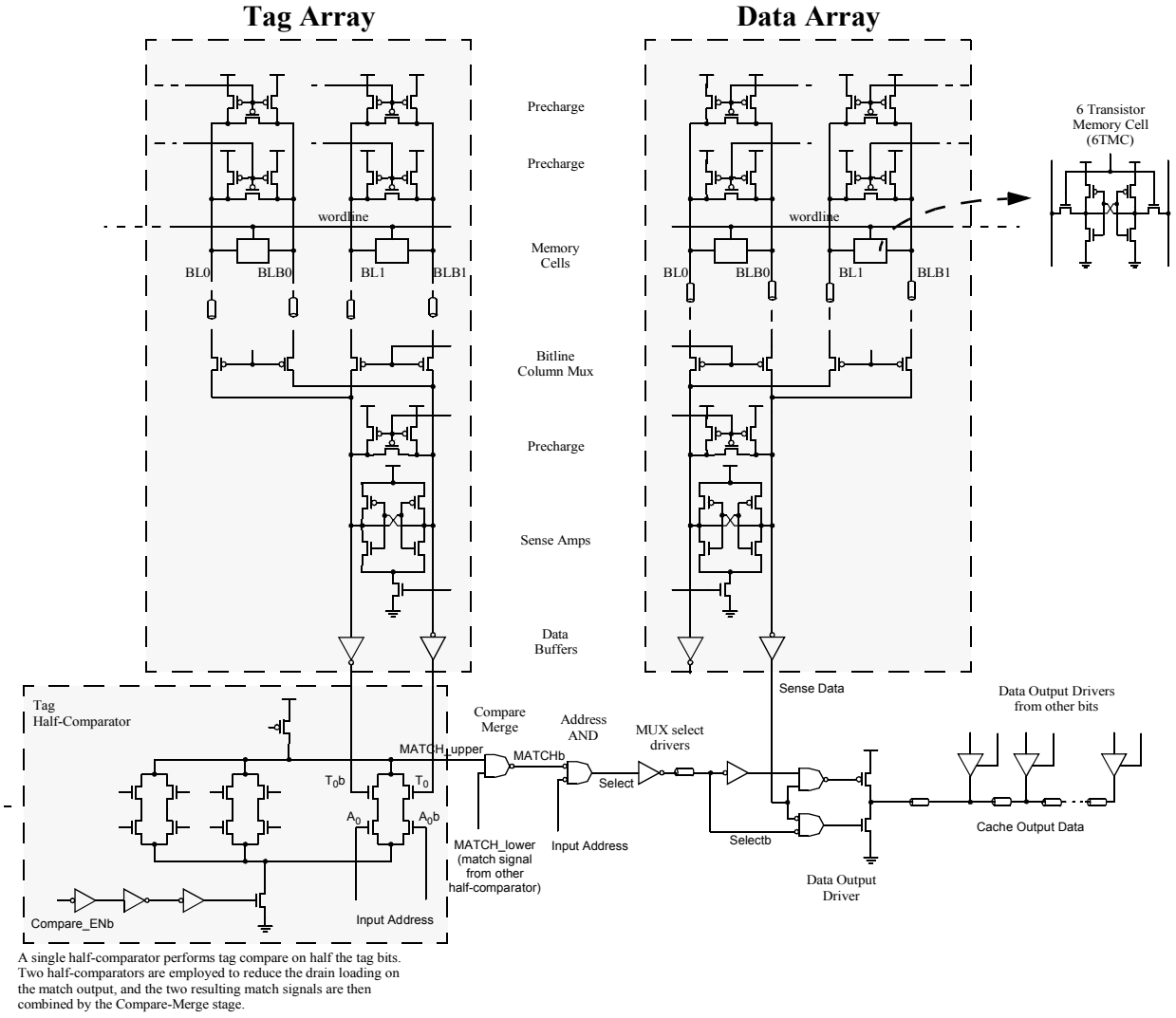


Figure 3-4: CACTI bitline, tag compare and output drive circuits. These circuits show the rest of the path through the cache after address decoding for the wordlines.

are now connected to the sense-amplifier. When enough differential voltage has been developed (and not until then, especially for the drain-fed sense amplifiers used by CACTI), the sense amplifiers are then fired, accelerating the development of differential voltage into full-swing logic-level voltages. It is important to emphasize that for low-swing differential sense-amplification, CACTI assumes adequate self-timing mechanisms such that the wordlines and the column muxes are shut off right after the required voltage differential have been developed in the bitlines. This way, only the minimum discharge is done on the bitlines such that precharging them back to their original high voltage will not dissipate too much power[5]. The output of the sense amplifiers are then buffered before being fed to the data output driver, at which point it either waits for the output drivers to be enabled or, if they already are, they then proceed to output the data onto the data output bus.

The tag array performs essentially the same operation right up to the sense-amplifier output. But after buffering, instead of being sent to data output drivers, the tag array output data is fed to a tag comparator to determine if the access hit or missed in the cache. The comparator essentially performs a wide XOR operation to check if the tag address read from the cache is exactly the same as the tag address provided by the processor, in which case the access is a hit and data from the data array is allowed to drive the output data bus. Otherwise, the access is a miss. The first versions of CACTI employed a comparator similar to the half-comparator shown in Figure 3-4 that operates on the entire tag address (i.e. it is as wide as the tag address). This was modified in succeeding versions, as implementing the XOR in one wide comparator results in very high drain capacitance loading at the output node. This was changed by reducing the fan-in of the original comparator to half the address and then using two of these comparators to produce two match signals which are then merged by a separate stage (which is essentially an AND gate that asserts the MATCH output if its two inputs, namely MATCH_lower and MATCH_upper are both asserted). Before actually driving the data output muxes, the

---

4. These are the column mux address decoders. They are not typically shown in the critical path analysis because they are almost never part of the critical path, since in the worst case, they should arrive at the same time as the wordlines. Moreover, they are actually expected to arrive much earlier because column decoders need to operate on a much smaller part of the input address compared to the main address decoder.

5. Self-timing techniques need to operate under two limitations. The first is that due to the big area occupied by the cache, the distances between different points in the same wire is significant enough such that the two points can be considered to have different state. This means that a self-timing circuit that sense the voltage of a signal at one point is either too late or too early coimpared to the signal at some other point. Typically, the workaround to this limitation is to use pessimistic assumptions. In this case, because wordlines reach some memory cells earlier than others, these memory cells will develop more bitline differntial compared to the ones that are at the far-end of the wordline. We want the self-timing circuit to avoid shutting off bitlines that have not developed enough differential, even at the expense of allowing other bitlines to develop more bitline differential than the minimum required. The second limitation is that with each new generation, process variability tends to get worse, such that self-timing circuits that do not account for this may be flawed if they are not fabricated properly. This is typically solved by including circuits in the self-timing mechanism that captures the PVT variation behavior and adjusts the self-timing accordingly. One good example is the replica technique, which uses redundant memory cells for the timing mechanism [Amrutur1998].

MATCH signal is first qualified by another set of input address, this time to choose which part of the cache output line to drive to the output bus (since the number of data output bits are typically smaller than the number of bits in the cache line). This stage generates the select signal that then drives the pre-drive NAND and NOR gates which then ultimately drive the final output driver devices.

3.2.5 CACTI program flow

Figure 3-5 shows a pseudocode of how CACTI performs design space exploration. CACTI performs an exhaustive search of the design space. It iterates on every possible combination of the cache implementation parameter sextet Ndwl:Nspd:Ndbl-Ntwl:Ntspd:Ntbl, and if the implementation is valid, it solves for the area, power, access time and aspect ratio of this implementation. It then compares this with the current best implementation (as determined by its own optimization algorithm and optimization weights), and if the

```
// Initialize best score
BestConfigScore = 0;
// MAXN is maximum value for Ndwl, Ndbl, Ntwl and Ntbl
for (Nspd=1;Nspd<=MAXSPD;Nspd=Nspd*2) {
  for (Ndwl=1;Ndwl<=MAXN;Ndwl=Ndwl*2) {
    for (Ndbl=1;Ndbl<=MAXN;Ndbl=Ndbl*2) {
      for (Ntspd=1;Ntspd<=MAXSPD;Ntspd=Ntspd*2) {
        for (Ntwl=1;Ntwl<=1;Ntwl=Ntwl*2) {
          for (Ntbl=1;Ntbl<=MAXN;Ntbl=Ntbl*2) {
            params = (C,B,A,Ndbl,Ndwl,Nspd,Ntwl,Ntbl,Ntspd);
            // Proceed only if current parameters are valid
            if (params_are_valid(params)) {
              // solve for area
              (current_area, current_aspect_ratio) = cache_area(params);
              // solve for power
              current_power = cache_power(params);
              // solve for access time
              current_access_time = cache_access_time(params);
             // If this configuration has a better score than the current optimal, save
              CurrentConfigScore =
      Opt_Algo(current_area,current_power,current_access_time,current_aspect_ratio);

              if (  CurrentConfigScore > BestConfig) {
                BestConfigScore = CurrentConfigScore;
                save_best_cfg(Ndbl,Ndwl,Nspd,Ntwl,Ntbl,Ntspd);
              }
            }
          }
        }
      }
    }
  }
}
```

Figure 3-5: CACTI program flow pseudocode. CACTI performs an exhaustive search of the design space by iterating on every possible implementation, scoring it, and looking for the implementation with the highest score, which it then considers as the optimal. The Opt_Algo() function used here is simply a single equation that assigns different weights to the four diffrent performance parameters and performs a multiply-accumulate function to produce a single number which is the score of the implementation.

current implementation is found to have a better score, it is recorded and used as the current best implementation. Once CACTI has iterated on every possible implementation, the current best implementation is now considered to be the optimal with respect to the optimization algorithm being used.

3.3 eCACTI

The cache design tool eCACTI [Mamidipaka2004] was developed to address several limitations of CACTI for deep-submicron technologies. The biggest change introduced by eCACTI was the support of process-specific subthreshold leakage models in addition to the CACTI dynamic power models. This is a very significant addition, as power dissipated by subthreshold leakage currents are expected to increase significantly with each technology generation. In addition, eCACTI now also accounts for the power dissipation due to circuits that are not in the cache critical path, something that was not being done previously by CACTI. Finally, another major change done by eCACTI is the dynamic resizing of many of the cache circuitry to allow for more optimal driving of the signals. Previously, CACTI only performed dynamic resizing on the wordline driver's last stage. eCACTI now dynamically resizes not only the entire address decoder but also significant parts of the caches after the sense amplifiers (e.g. the data output drivers, among other things).

3.3.1 Basic Operation

The basic operation of eCACTI is very similar to CACTI, in that it accepts almost the same input parameters and performs a design space exploration using the same cache implementation sextet Ndwl:Nspd:Ndbl-Ntwl:Ntspd:Ntbl used by CACTI.

eCACTI does have a few enhancements to the basic CACTI operation, including the addition of an "evaluate" mode where instead of doing a design space exploration (the default "explore" mode) to produce the optimal implementation, the program accepts the implementation parameters Ndwl:Nspd:Ndbl-Ntwl:Ntspd:Ntbl as well as the normal inputs (i.e. cache size, associativity, technology node, etc.) and evaluates the behavior of the cache with the given parameters. This is useful for studying the effects of the different parameters in isolation, as well as characterizing the effects of technology shrinks where an existing cache design is to be ported to another technology with minimal changes in its microarchitecture.

3.3.2 eCACTI cache structure

Figure 3-6 shows the cache structure used by eCACTI. The figure shows a cache structure that is very similar to the CACTI cache. The main blocks that were added are the non-critical path logic blocks like the write control logic and write control muxes (among other things), to make their inclusion into the model more explicit.

3.3.3 eCACTI address decoder

Figure 3-7 shows the decoder circuit implemented in eCACTI. This is essentially the same decoder implemented in CACTI except for two major changes. The first is that runtime dynamic device resizing is performed not just for the final wordline driver stage, but for every gate in the decoder. After computing the required device widths for the last stage of the wordline driver, a sizing algorithm works its way backward, sizing the first stage of the wordline driver and then the wordline decoder NOR gate. It then computes how

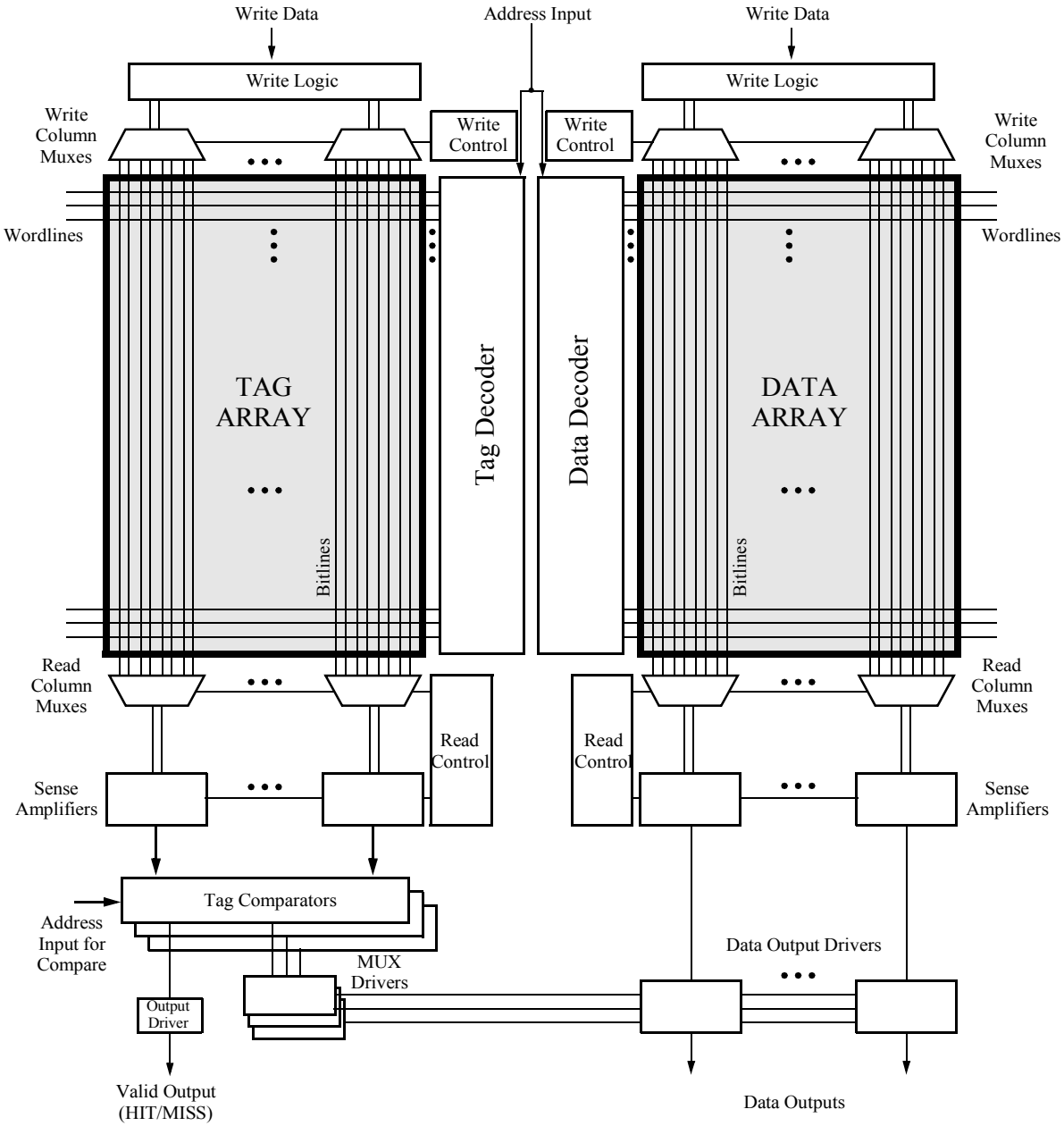


Figure 3-6: eCACTI cache structure. This figure shows a high-level block diagram view of the cache structure used by eCACTI. Note that even though the data and tag arrays are shown to be the same in the figure, the data array is almost always much larger physically compared to the tag array.

many of these NOR gates are connected to each other and after also accounting for the interconnect capacitance, the 3-to-8 predecoder NAND gates are sized accordingly. It then computes how many of these predecoders connect to a single address and again after accounting for the interconnect capacitance, sizes the address buffers accordingly. The second major change with resepect to CACTI is the inclusion of a second address buffer stage, as the sizing algorithms typically result in large 3-to-8 predecoder device widths, necessitating the addition of another stage.

Like CACTI, all of the gates in the decoder are implemented in full-CMOS static logic. As we will show later, the dynamic resizing of the entire address decoder, along with the fixed-stage static full-CMOS decode implementation potentially introduces impractical solutions. What makes this worse is that these solutions are hidden to the user, and there is no way to determine the practicality of the solution without going into the code in detail and inserting debug hooks to provide feedback[6]

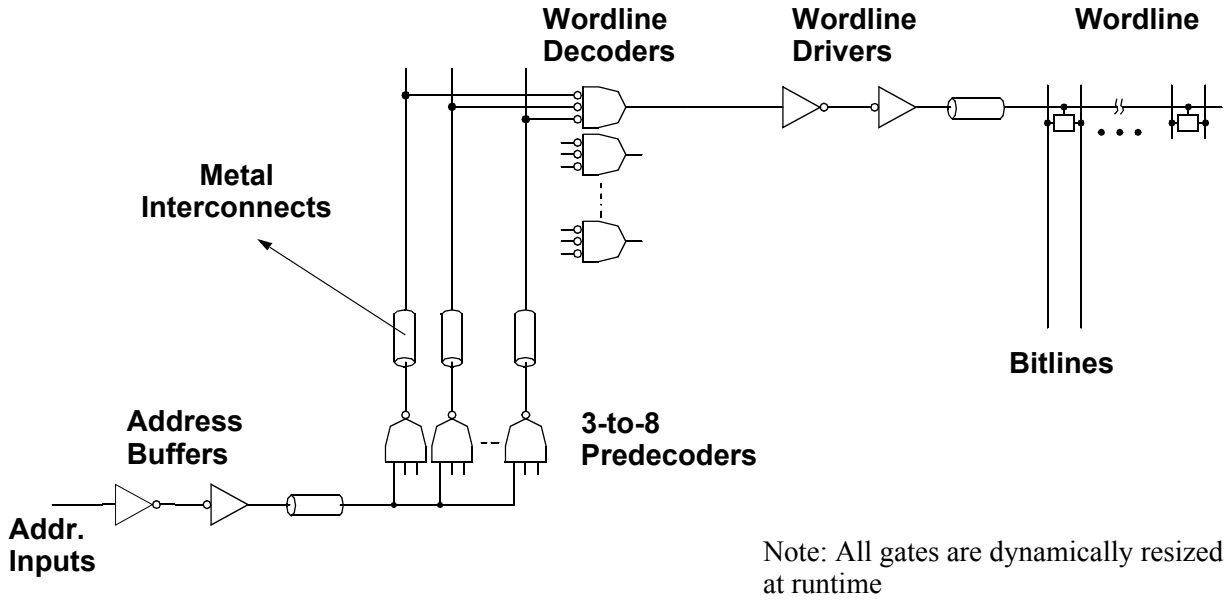


Figure 3-7: eCACTI address decoder. All gates shown are implemented with full-CMOS static logic. Unlike CACTI, all the gates in the eCACTI address decoders are dynamically resized at runtime, starting with the calculation of the wordline driver stage then proceeding backwards until the first address buffer stage that connects to the address input.

6. Basically, the act of dynamically resizing the entire decoder results in large device widths starting from the 3-to-8 predecoder NAND gates. This is because the wordline decoders typically become larger after resizing (where as previously, they had a fixed size regardless of loading). Coupled with the fact that these are implemented as high fan-in static NORs with very high logical effort and that the fixed-stage implementation does not allow for flexibility in resizing (i.e. the given load has to be driven in the given number of stages), really put pressure on the sizing of the predecoder gates. With the address buffers driving multiple predecoder gates, the sizing pressure also affects the address buffers, in many cases making the situation worse. We will later show that in many solutions, some of these devices have very impractical device sizes, but since eCACTI does not report or flag these, the user has no way of knowing whether the solution returned by eCACTI is impractical or not.

3.3.4 eCACTI bitline circuits, tag compare and output drivers

eCACTI essentially implements the same bitline circuits, tag compare circuits, and output drivers as CACTI, as shown in Figure 3-4. The main eCACTI enhancements are not shown in the figure (which essentially shows only the possible critical paths). Specifically, the write path is not shown, including the write control logic and write column muxes.

3.3.5 Additional eCACTI enhancements

In addition to the enhancements already mentioned, eCACTI also offers some more enhancements to its operation. eCACTI also considers not only read power when computing cache power but also write power. The maximum of the two estimates is used for the optimization schemes. In addition, eCACTI also supports use of a look-up table based leakage current model instead of their analytical model in order to reduce minor errors of the analytical model when compared to simulation. Also, eCACTI supports the analysis of specific low-power cache techniques like drowsy caches [Flautner2002] and gated-Vdd [Powell2000, Kaxiras2001], along with use of a dual-Vt process. Finally, as already mentioned, eCACTI has the option to evaluate a specific cache implementation instead of performing a design space exploration. With this feature, the behavior of an existing cache implementation can be extrapolated to newer technologies (i.e. a process shrink).

3.4 myCACTI

CACTI's modeling parameters are based on a reference 0.80um model, and simulation runs that targeted smaller process technologies simply scaled down the results linearly. It should be obvious that this linear scaling will not be entirely accurate, especially for newer technologies, as the long-channel transistor models that were used to derive these parameters do not readily scale in order to describe short-channel behaviour. In addition, many other device behavior will be left uncaptured by using simple scaling. eCACTI partly addresses this by using per-process parameters for deep submicron technologies, but only for computing subthreshold leakage behavior, while all other parameters are still computed by linear scaling. With the shift to nanometer process technologies, device behavior has deviated drastically from the original long-channel behavior used for CACTI and eCACTI. myCACTI addresses this, among other things, by using simulation parameters that are specific for each target process, allowing more accurate modeling. Some of the major enhancements, which will be described later in this section in more detail, is the support for explicit pipelining (which is virtually necessary for high-volume commercial caches), the support for a gate leakage model, and the use of more optimal dynamic circuits in the decode hierarchy, among many others.

3.4.1 Basic operation

The basic myCACTI input parameters are exactly the same as CACTI and eCACTI, as shown in Table 3-1 in the CACTI subsection. In addition, it also supports eCACTI's use of a parameter file to specify additional

options and force the use of specific cache implementations. Table 3-3 shows the switches supported by myCACTI in its parameter file.

Table 3-3: Parameter file switches supported by myCACTI (all are optional)

Switch	Usage
-config N_{dwl}:N_{spd}:N_{dbl}:N_{twl}:N_{tspd}:N_{tbl}	Forces myCACTI to perform a design evaluation of the given parameters instead of the default design exploration
-singleVt	Forces myCACTI to assume a single-Vt process (all LVT transistors) instead of the default dual-Vt process
-static_decode	Force the use of static logic for address decoding instead of the default dynamic logic
-useFreq <freq>	Specifies frequency to use, overriding the per-process hard-coded clock frequency
-use_singlelayer_metal	Force the use of a single-layer metal (similar to CACTI and eCACTI) instead of the default multi-layers
-use_lowK	Force the use of a low-k dielectric for the interconnects (Dielectric constant of 1.0 is used instead of the default 3.0)
-disable_viacap	Forcibly ignores via parasitic capacitances. Default operation includes via caps.
-use_Ldrawn	Use an effective transistor length equal to the drawn length, ignoring drain/source terminal overlap. The default accounts for this overlap by adjusting the effective length by the amount of overlap.
-single_ended_sensing	Use full-swing single-ended sense amplification instead of the default low-swing differential sensing
-use_3pipephases	Use only three pipeline phases (i.e. 1 1/2 clock cycles) instead of the default four phases (i.e. 2 clock cycles) for the complete cache operation
-disable_WLres	Ignores the use (similar to CACTI and eCACTI) of the wordline resistance when computing the wordline driver device widths

In addition, myCACTI uses subarraying exactly the same as CACTI and eCACTI, with use of subarrays demonstrated earlier in Figure 3-1.

The output implementation parameters produced by myCACTI are exactly the same as CACTI and eCACTI, as shown earlier in Table 3-2. In addition, myCACTI produces the files listed in Table 3-4.

Table 3-4: myCACTI output files

Output File	Usage
best_config.txt	Produced during a design space exploration, the file contains the optimal configuration found by myCACTI during the exploration. The format is compatible with the myCACTI input parameter file.
valid.txt	Produced during a design space exploration, this file contains all the valid configurations studied by myCACTI and valid information like delay and power numbers. This file is useful in creating pareto optimal curves for a particular run.
timing.txt	Produced during a design evaluation, this file contains breakdown of the delays in the cache
power_read.txt	Produced during a design evaluation, this file contains power dissipation for a cache read-hit access broken down into cache components, and source of power (i.e. dynamic power, subthreshold leakage power and gate leakage power)
width.txt	Produced during a design evaluation, this file contains the results of the runtime dynamic resizing of the device widths in the cache.

3.4.2 myCACTI cache structure

The high-level view of the cache implemented by myCACTI is shown in Figure 3-8. The cache structure is essentially the same as the eCACTI structure shown in Figure 3-6, with the main difference being the addition of explicit pipeline latches where necessary to support the 2-cycle cache pipeline operation that will be discussed in detail in a later section.

As shown in the figure, it takes three latches to go through both the data and tag array before the data output drivers are reached. For the data array, the address goes through two sets of latches in the decoder before producing the wordline signal. After a memory cell is enabled by this wordline signal, the bitlines are discharged and the sense amplifier converts the bitline signal into a full-swing logic-level signal before going through another latch. For the tag array, the address only goes through one latch, as the tag array decoding is typically simpler and occupies much less area because of the smaller size of the tag array compared to the data array. This latch is right before the wordline drivers. Same as the data array, the wordline driver then enables some of the memory cells, discharging the bitlines. The bitline is then amplified by the sense amp and its outputs are then latched (note that at this point, the wordlines in the data array are also just being latched). The sense amp data are then provided to the tag compare circuit for comparison with the input tag address. The

result of then comparison is again latched and then drives the mux select driver which then drives the final output driver. The data outputs along with the result of the tag comparison are then latched at the cache periphery for use by the circuits external to the cache. At this point, some postprocessing may yet be done, like data-shifting or alignment, or even additional glue logic in cases where cache subbanking is being done using external logic (which combines the results of more than one cache together).

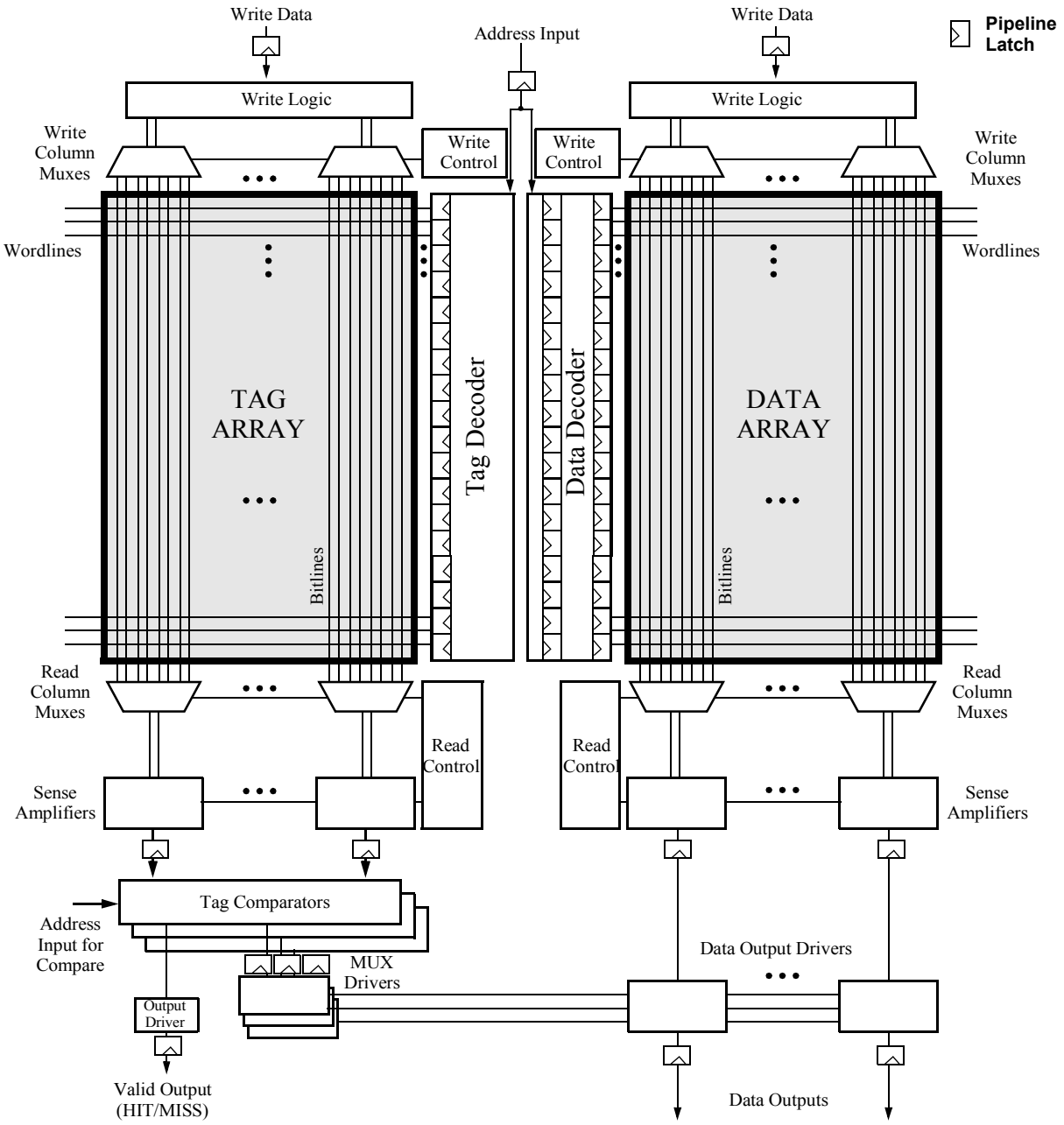


Figure 3-8: myCACTI cache structure. This figure shows a high-level block diagram view of the cache structure used by CACTI. Note that even though the data and tag arrays are shown to be the same in the figure, the data array is almost always much larger physically compared to the tag array.

3.4.3 myCACTI address decoding

The address decode hierarchy of myCACTI is shown in Figure 3-9. This hierarchy is similar to the eCACTI hierarchy (and to a large extent the CACTI decode hierarchy) except for two main differences. The first difference is the use of a NAND gate at the wordline decoder front-end instead of a NOR gate. This is a more optimal implementation, as a NAND gate will have a lower logical effort compared to a NOR gate[7]. Both CACTI and eCACTI were forced to use a NOR gate because they limit the number of drivers to a fixed even number of two. Consequently, since the wordline expects a high-asserted signal, the wordline decoder front-end has to be implemented with a NOR gate for the logic to be correct. In myCACTI, an odd number of stages is implemented, enabling the use of a NAND gate front-end while maintaining the correctness of the logic.

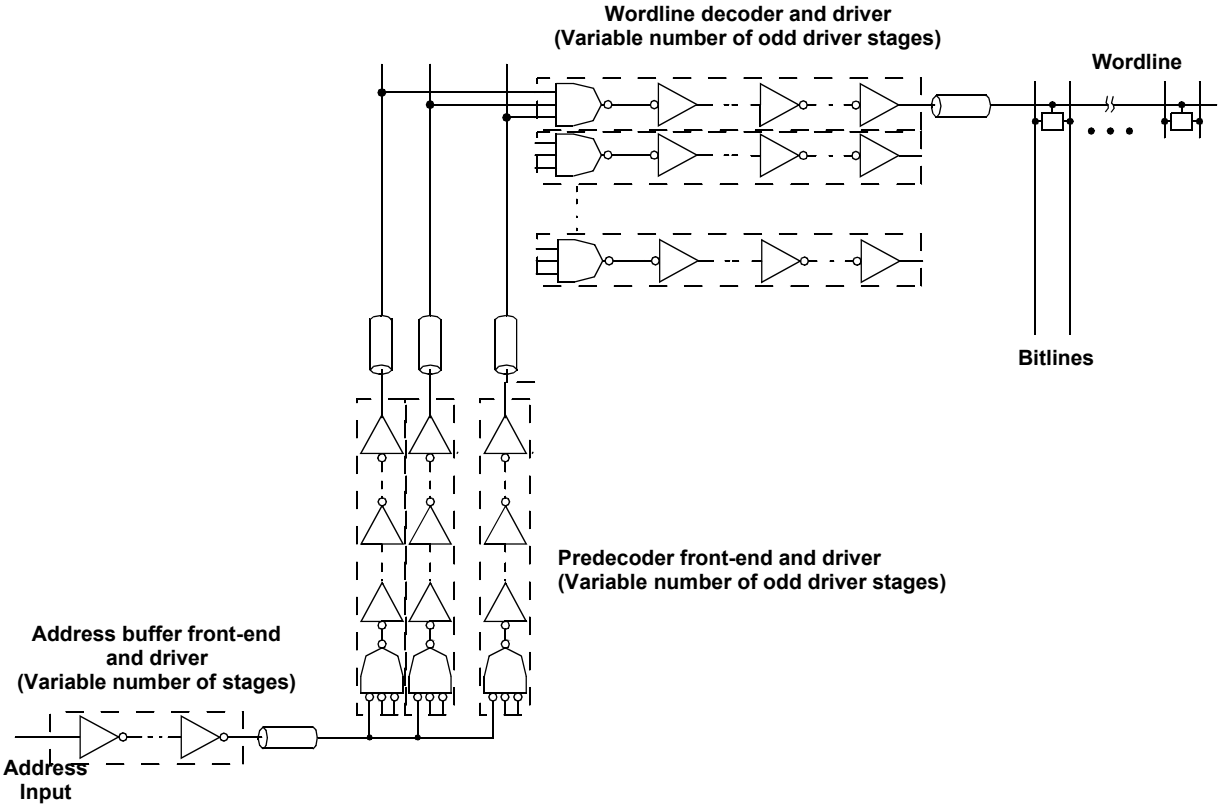


Figure 3-9: Unpipelined myCACTI decode hierarchy. The figure shows the decode hierarchy employed by myCACTI. Although the figure shows the gates using logical symbols, these gates can be implemented in static full-CMOS logic (in which case each gate symbol corresponds to one gate), or dynamic logic (in which case the delineating rectangles show how each part can be merged into one dynamic implementation). Also noted in the figure is the ability of myCACTI to adjust the number of driver stages to more optimally adapt to specific loading conditions, while at the same time minimizing the capacitive loading provided by the inputs to the front-ends to minimize the burden of upstream circuits that drive these frontends. For example, front-end circuits are typically made minimum-sized for optimal implementation [Amrutur2000] and all front-ends as shown here are minimum-sized, with the number of drive stages adjusted accordingly.

7. For example for a fan-in of 3, a NAND gate will have a logical effort of 1.67 compared to 2.33 for the NOR gate. For a fan-in of 5 (the maximum fan-in used for these tools), the NAND gate will only have a logical effort of 2.33 compared to 3.67 for the NOR gate. Again, logical effort is a measure of how much larger the input capacitance is of a gate compared to an inverter of the same drive strength. Obviously, a small logical effort is desirable to minimize the capacitive loading.

The second major difference is the support of myCACTI for a variable number of drive stages for the entire decode chain, with the only requirements being that the predecoder and the wordline decoder drivers have an odd number of stages to maintain the correctness of logic (this is automatically enforced internally). With this capability, more implementation design points can now compete for optimality, as the decoders can now accomodate very lightly loaded or very highly loaded nodes by adjusting the number of stages, as opposed to the current bias towards a certain loading imposed by the fixed stages. In addition, this enables the sizing algorithm to fix the size of the front-end gates to be minimum-sized, which has been proven to be the optimal implementation in address decoders [Amrutur2000]. Even if the front-end gates are minimum-sized, the load can still be properly driven by choosing the right number of stages and sizing them properly. This is not possible with eCACTI's sizing algorithm, as the high fanouts in the wordline decoders often force the 3-to-8 predecoder NAND gate of eCACTI to be sized largely, which then also propagates to the sizing of the address buffers. CACTI also suffers this to some extent, as the 3-to-8 predecoders size, though fixed, is set to a value that is significantly greater than minimum-size (e.g. 16X instead of 1X).

It must be noted that the myCACTI decode hierarchy shows a generic unpipelined implementation of the decode logic. The pipelined version of the address decoder will be discussed in detail in a later subsection. Additionally, these decode logic can be implemented as either full-CMOS static logic similar to both CACTI and eCACTI, or they could be implemented with more optimal circuitry that can both minimize the logical effort even further, and also allow more efficient driving of the signals.

The dynamic implementation of the decode hiearchy is shown in Figure 3-10. Unless overridden by the "-static_decode" option in myCACTI, this dynamic decode implementation is used by default.

The address buffers are implemented using dynamic source-coupled logic (SCL) followed by simple full-CMOS drivers. A big advantage of SCL logic is its capability of producing the signal and its complement with the same delay, typically saving one gate delay compared to a simple static inverter buffer implementation. The number of driver stages of the address buffer can have either odd or even stages, as both the address and its complement are to be used by the predecoder. Adding (or removing) an additional stage only requires switching the wire designation for the signal and its complement version.

The predecoders are implemented with dynamic delayed-reset CMOS (DRCMOS) with SCL front-end [Nambu1998] implementing an OR/NOR function. The main point of delayed-resetting is that it allows the designer to favor the speed of one transition at the expense of slowing down the other transition. In the case of address decoding, it is desirable to speed-up the transition that will result in the assertion of the wordlines. This is done by sizing the devices that are involved in the forward transition normally, but significantly reducing the strength of the other device. This way, the forward transition is sped up because of the significantly reduced gate loading. Even if the other edge is slowed down, this typically does not matter too much as the precharge phase will typically not be in the critical path and as long as it is kept short enough, will

also not degrade the cache cycle time. In the case of pipelined implementations like the one in myCACTI, a full clock phase is given for precharging, ensuring that there is no problem in going back to its original state while ensuring that the forward transition is sped up.

Finally, the wordline decoders are implemented with simple DRCMOS logic (as opposed to SCL DRCMOS) in order to eliminate the need for the additional clock input required by SCL-DRCMOS, at the expense of higher logical-effort because of the full-CMOS NAND front-end. This is a reasonable tradeoff, as supplying a clock to each wordline driver in the cache will be highly inefficient, especially since not all of these wordlines are actually going to be activated. Supplying a clock to these gates will be highly inefficient.

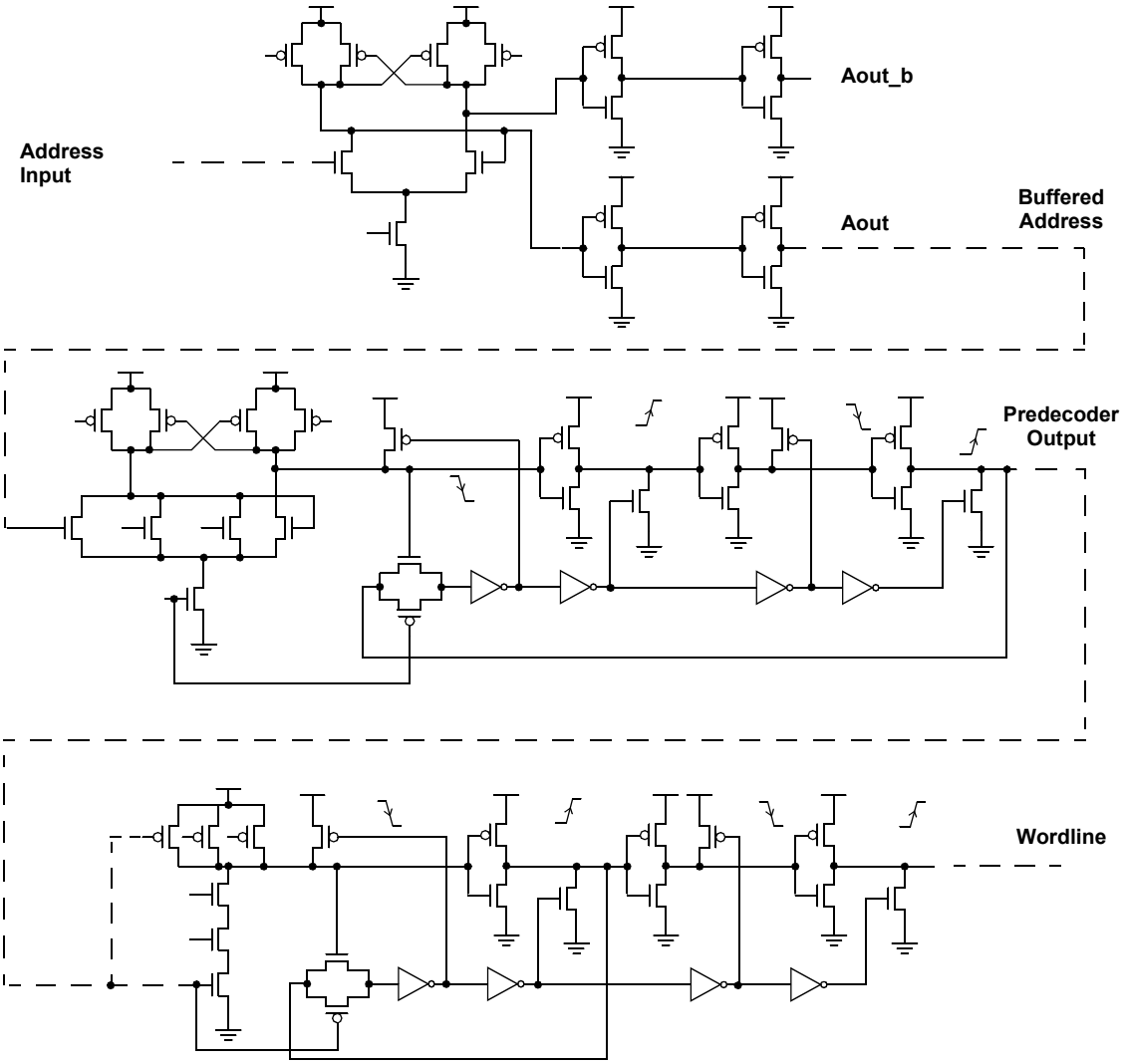


Figure 3-10: Dynamic address decode circuitry implemented in myCACTI. Shown are more optimal dynamic logic implementation of the decoders, which are enabled by default for myCACTI runs (although users can opt to revert back to a static full-CMOS implementation). The address inputs are buffered by an SCL dynamic logic buffer, and the buffered address are then sent to an SCL-DRCMOS decoder (which is shown here to have three drive stages, although this can easily be adjusted to either 1 stage or 5 stages). The predecoded address are then provided to a DRCMOS dynamic gate with a static NAND front-end (which is shown here to also have three driver stages). Finally, the PMOS of the final driver stage pulls up the wordline, effectively enabling the memory cells connected to it.

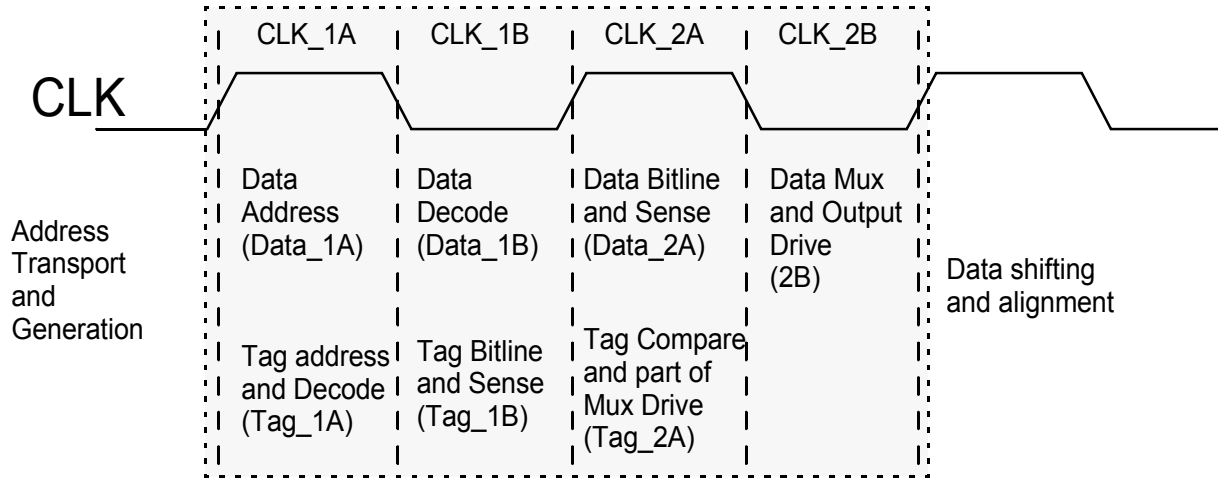


Figure 3-11: Cache pipeline diagram. This pipeline diagram indicates the different stages of the cache pipeline, with the shaded regions showing which stages are included in the modeling. Here, the main cache block has a 2-cycle latency. Also assumed by the diagramis the presence of the TLB-translated address (if applicable) by the start of the CLK_2A phase of the clock in time for use by the Tag_2A stage.

With simple DRCMOS, the predecode outputs can serve as the reset signal, such that only the activated gates actually see this transition and hence, dissipate power.

3.4.4 myCACTI bitline circuits, tag comparators and output drivers

The bitline circuits, tag comparator circuits, and output drivers are exactly similar to CACTI and eCACTI except for the addition of pipelining. These pipelined circuits are discussed in detail in the next section.

3.4.5 myCACTI pipelining

To keep up with the speed of a fast microprocessor core while providing sufficiently large storage capacities, caches are pipelined to subdivide the various delays in the cache into different stages, allowing each individual stage to fit into the core's small clock period. Figure 3-11 shows a typical pipeline diagram for a pipelined microprocessor cache. This pipeline diagram also represents the pipeline implementation used by myCACTI, with the shaded region depicting the phases that are included in the model. The given timing diagram shows operations being performed in both phases of the clock. Stages that are active in a given phase are reset in the next one, such that another independent access can be started every cycle since every stage that will be used have been reset in the previous phase.

Instead of using explicit pipeline state elements, CACTI and eCACTI both use implicit pipelining through wave pipelining, relying on regularity of the delay of the different cache stages to separate signals continuously being shoved through the cache instead of using explicit pipeline state elements. Unfortunately, this is not representative of modern designs, since cache wave-pipelining is not being used by contemporary microprocessors [Riedlinger2002, Weiss2002]. Although wave pipelining has been shown to work in silicon prototypes [Burleson1998], it is not ideally suited for high-speed microprocessor caches targeted for volume

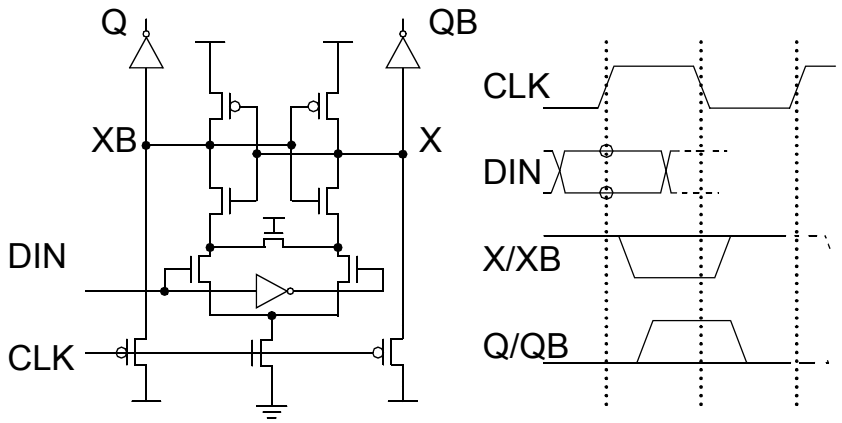


Figure 3-12: Pipeline latch. An example of a pipeline latch that can be used to implement the phase-based operations in the cache [Montanaro1996, Gronowski1996]. Also shown is the latch's timing diagram

production that have to operate with significant process-voltage-temperature (PVT) variations. PVT variations in a wave-pipelined cache cause delay imbalances which, in the worst case, lead to signal races that are not possible to fix by lowering the clock frequency. Hence, the risk for non-functional silicon is increased, resulting in unattractive yields. In addition, wave-pipelining does not inherently support latch-based design-for-test (DFT) techniques that are critical in the debug and test of a microprocessor, reducing yields even further. On the contrary, it is easy to integrate DFT "scan" elements inside pipeline latches that allow their state to be either observed or controlled (preferably both). This ability facilitates debugging of microprocessor circuitry resulting in reduced test times that directly translate to significant cost savings. Although not immediately obvious at first, these reasons make it virtually necessary to implement explicit pipelining for high-volume microprocessors caches. myCACTI supports the modeling of explicit pipelining in order to more accurately describe nanometer caches in the context of a real and practical microprocessor design.

myCACTI pipeline latch. Figure 3-12 shows the pipeline latch implementation that is modeled in myCACTI [Montanaro1996, Gronowski1996]. This is a latch that easily facilitates the phase-based operation of the pipeline that was described earlier. Before its evaluate clock phase, the latch expects its input signal to stabilize in preparation for latching. During the evaluate phase, the input signal can be changed (after it has satisfied a minimum hold time, of course) and the output changes its value based on the given input. This output stays valid for the whole evaluate phase even after the input has been changed. During the next phase, designated as the latch's reset or precharge phase, the latch is reset back to its original state in preparation for another evaluate phase. Although this latch by itself is still not capable of DFT techniques, it can be easily extended to support DFT observability simply by using this existing stage as the "slave" stage in a master slave configuration, with the "master" stage included in the circuit only for the purpose of DFT. The forward path of the latch will be unchanged and does not go through the master stage, minimizing the effect on delay aside from the minimal additional loading at the output gate of the slave stage. Extension of DFT-support for

controllability can also be easily accomplished by simple circuit modifications of the output buffers to make it conditional and qualified by scan clocks.

myCACTI pipeline stages. Figure 3-13 shows the data address decoder of myCACTI along with part of the bitline circuitry. The figure shows the three pipeline stages (with each stage being half a clock cycle or one clock phase long) through the data array from the input address to just before the output driver. These three stages, namely Data_1A, Data_1B and Data_2A are also broken down into different components. Detailed numbers of these components are given in the studies discussed later in this thesis. The pipeline stage Data_1A (which occurs in the A phase or the first phase of clock cycle 1) consists of address buffering, driving the metal interconnect to the predecoders, the predecoder front-end, and associated flop delays. The pipeline stage Data_1B consists of the predecoder drivers, driving the metal interconnect to the wordline decoders, the wordline decoder front-end, and associated flop-delays. Finally, the third data stage data_2A consists of the

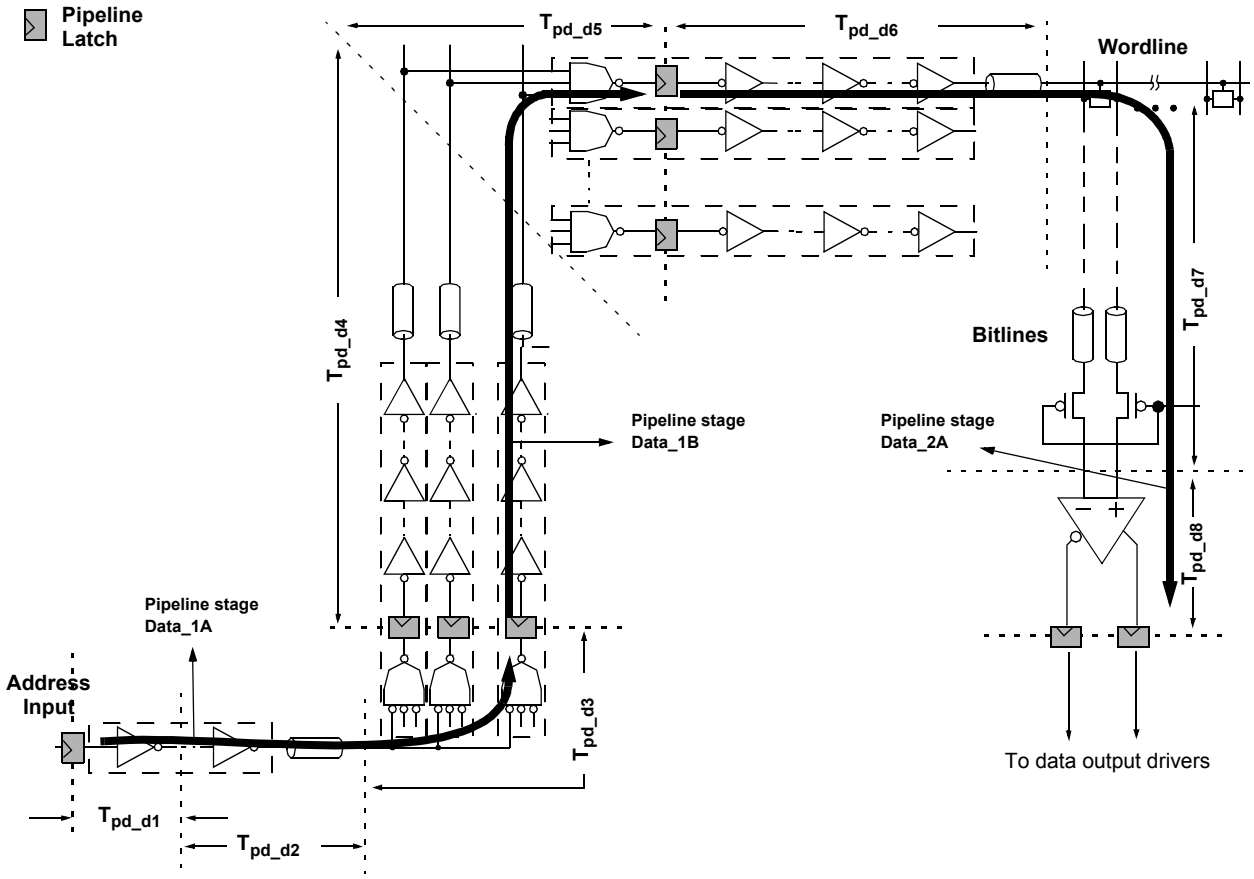


Figure 3-13: myCACTI data decode hierarchy showing pipelining. The figure shows the pipelined decode hierarchy modeled by myCACTI, along with part of the bitline circuits in order to show complete pipeline stages. Shown are the three pipeline stages (where each stage is half a clock or one clock phase long) before the output driver stage. The first stage goes through the data address buffer and the predecoder front-end before being latched. The second stage goes through the predecoder driver and the wordline front-end stage before being latched. Finally, the third stage goes through the wordline drivers, through the memory cells discharging the bitlines, the column multiplexers, and the sense amplifier before being latched. Also shown are individual components of the delay, detailed numbers of which will be provided in later sections.

wordline drivers, driving the interconnect to the furthest memory cell, the memory cell discharging the bitlines, the differential going through the column mux, sense amplification and finally, any associated flop overhead. It is important to note that the placement of these latch points are not arbitrary, as there are limited places in the cache that could serve as latch points. In this case, the two rules of thumb that were observed in latch placement is that the latch output is restricted to have moderate drive capability and that it can not latch analog signals. The first restriction makes it impractical to insert latches after high-strength drivers. Although from a logic viewpoint this does not pose a problem, the fact that the latch output is replacing a high-strength driver means that a high capacitive load is being driven, which is impractical given the low drive strength of the latch. The second restriction simply exposes a limitation of digital latches where only logic level signals can be latched. This disallows the use of any latch point in the bitline outside of the sense-amplifier output, since these signals are most probably low-swing signals (except in the case where single-ended sensing is enabled, in which case it is theoretically possible but still inadvisable because of noise issues with the bitlines).

The pipelined tag array decoder is shown in Figure 3-14. This is essentially the same as the data address decoder except that the tag array has merged the first two stages of the data address decoder into a single stage such that only two stages are now being used. This is possible since the tag address decoder is significantly simpler than the data decoder because of the much smaller storage requirement of the tag array, such that the loads being driven are typically much smaller and hence, each stage has less delay such that more logic can be included in a single stage.

The myCACTI pipelining of the tag compare and output drive circuits are shown in Figure 3-15. Although the pipeline stage containing the bitline and sense amplifiers are also shown, they have already been explained in the previous paragraphs, so no further detailed discussions are supplied. For the data array, once the sense amplifiers produce the data, they are directly supplied to the output predriver. For unpipelined implementations, this is a potential critical path, but for this particular pipelined implementation, this is not the critical path since the latch point is very near the predriver such that the critical path for this particular pipeline stage will always come from the tag comparator stage. For the tag path, in pipeline stage Tag_2A, the tag comparator stage accepts the sense amplifier output from the tag array sense amps, but similar to the data array, this is not the critical path, as the critical path actually goes through the driver that enables the tag comparator since this has more delays in the path. Once comparsion has been done and a match signal has been produced by each comparator, the two match signals from both half-comparators are then merged by the compare-stage gate and then the final output signal is latched. Finally, the data array and the tag array path merge in the pipeline stage 2A, where the match signal from the tag compare goes through some more mux selection, then proceeds to drive the output driver which then drives the output data bus to the cache periphery where the final data is latched.

3.4.6 Gate leakage computation

For power modeling, CACTI only considers dynamic power. eCACTI improves on this by including a subthreshold leakage power model. With the expected increase of gate leakage [ITRS2001], it has received significant attention [Yeo2000, Hamzaoglu2002, Rao2003a, Rao2003b] of the research community. Consequently, myCACTI now includes gate leakage tunneling current modeling to improve the accuracy of power dissipation, especially for nanometer caches where gate leakage is expected to be significant.

Gate leakage tunneling currents are modeled in myCACTI by solivng SPICE BSIM4 equations that support gate leakage computations, along with SPICE decks that explicitly include gate leakage current models. Figure 3-16 shows the different components of gate leakage tunneling currents that are included in the myCACTI gate leakage model.

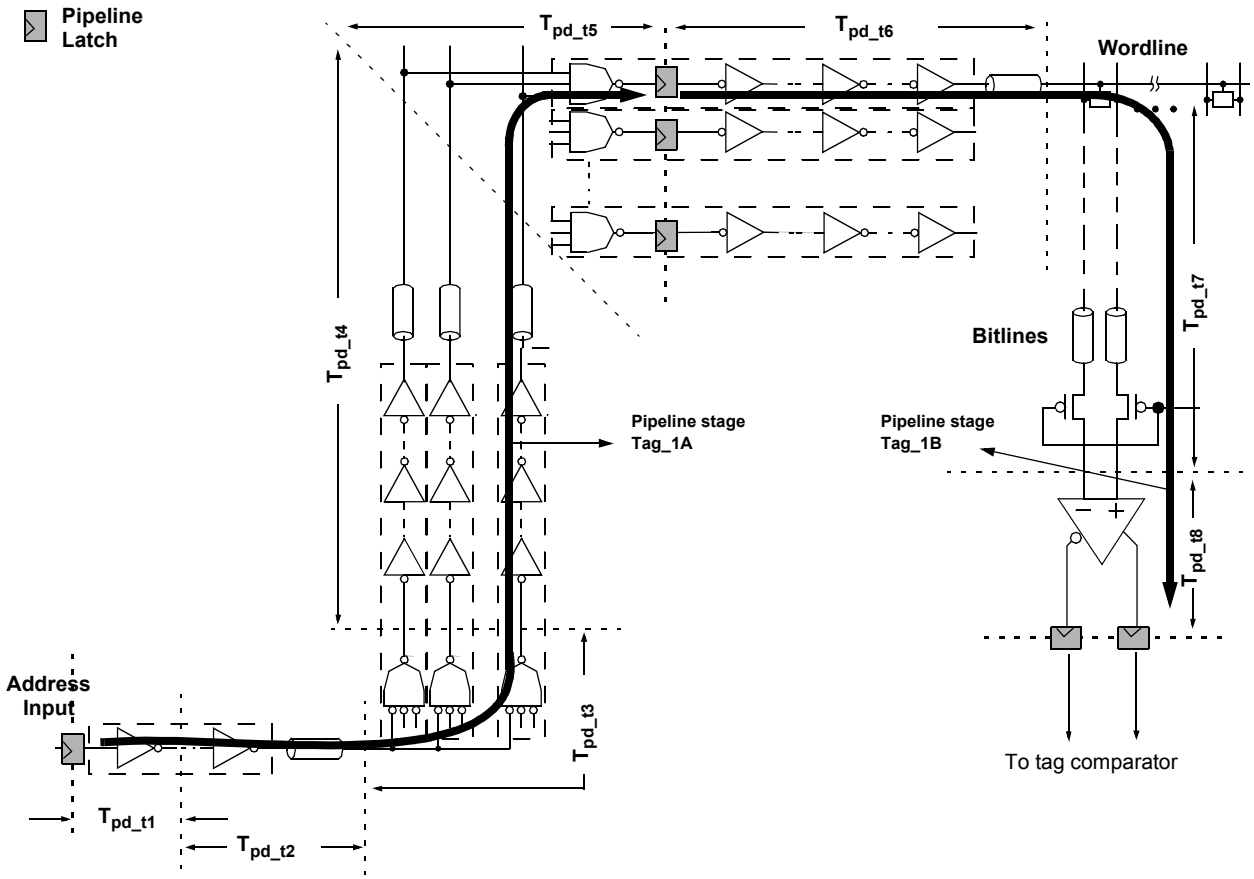


Figure 3-14: myCACTI tag array decode hierarchy showing pipelining. The figure shows the pipelined tag array address decode hierarchy modeled by myCACTI, along with part of the bitline circuits in order to show complete pipeline stages. Shown are the two tag pipeline stages (where each stage is half a clock or one clock phase long) before the tag comparison. The first stage goes through the data address buffer, the predecoder front-end and predecoder driver and the wordline front-end stage before being latched. The second stage goes through the wordline drivers, through the memory cells discharging the bitlines, the column multiplexers, and the sense amplifier before being latched. Also shown are individual components of the delay, detailed numbers of which will be provided in later sections.

3.4.7 Device characterization

During myCACTI simulations, the following device parameters are required to perform the various computations:

- Drive current capability / device effective resistance (delay)
- Gate and diffusion capacitance (delay and power dissipation)
- Subthreshold leakage current (power dissipation)

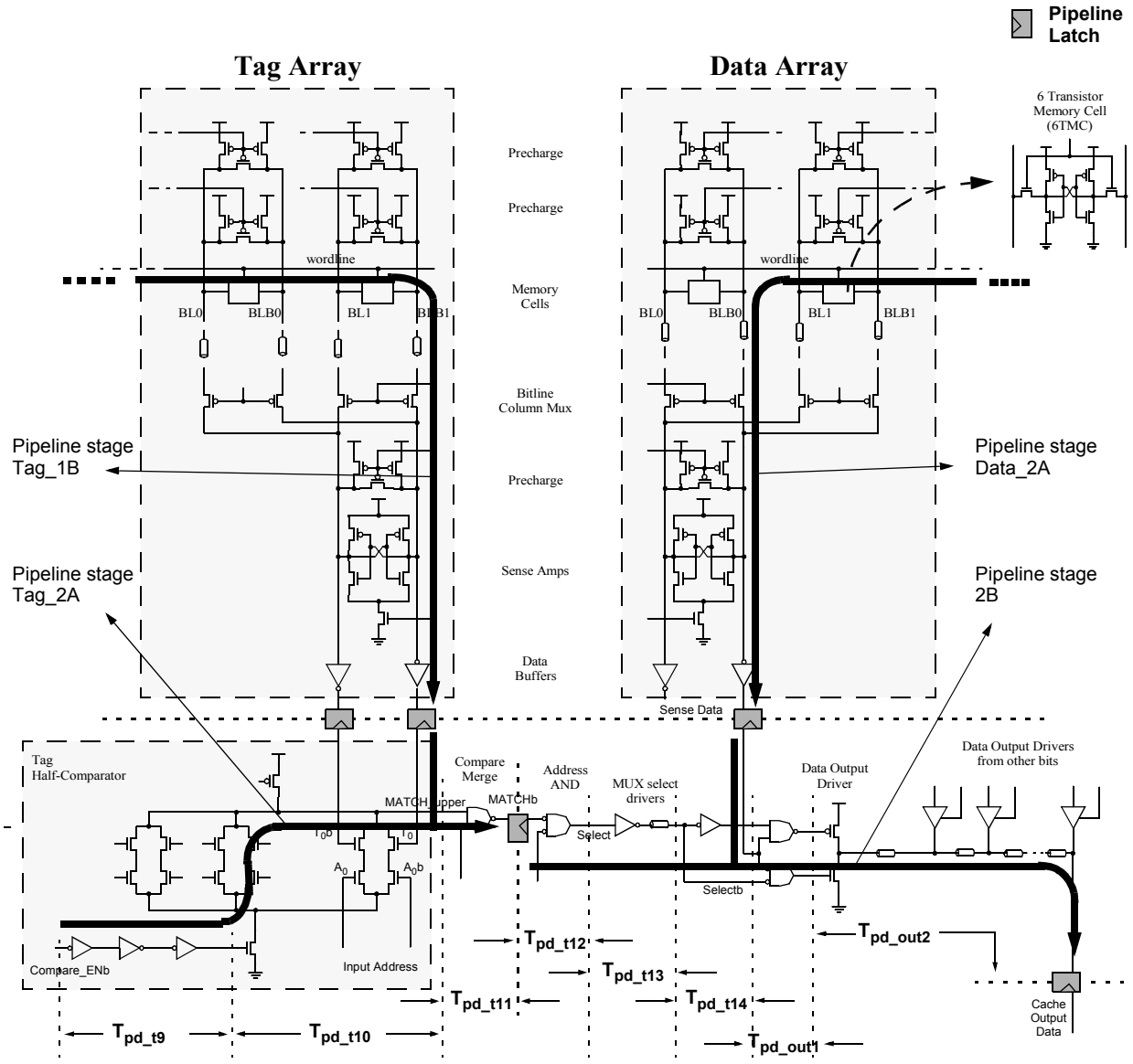


Figure 3-15: myCACTI pipelined bitline, tag compare and output drive circuits. These circuits the pipelining of the bitlines, tag compare, and output driver circuit in myCACTI.

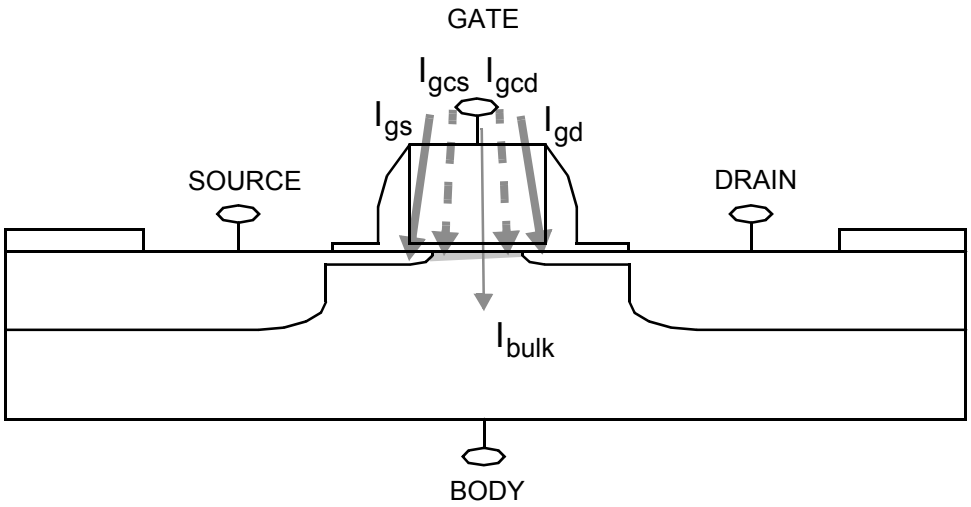


Figure 3-16: MOSFET Gate leakage tunneling currents. The five different gate leakage tunneling currents in a MOSFET are shown. These five can be categorized into three groups: the channel tunneling current, the source/drain junction overlap tunneling current, and the bulk tunneling current.

- Gate leakage tunneling current (power dissipation)

For the first and second parameters, both CACTI and eCACTI rely on hard-coded compile-time parameters that are based on a reference 0.80um process technology. For the third parameter, eCACTI relies on hard-coded compile-time parameters that are based on different targeted process technologies to perform the computation, while CACTI has no support whatsoever. Both CACTI and eCACTI do not support modeling of gate-leakage tunneling current mechanisms.

Using a 0.80um process as a reference will yield (as we will later show) very inaccurate results when extrapolated to nanometer technology nodes like 90nm, 65nm, 45nm and 32nm. Among other reasons, process engineers have, over the years but especially in the last few, redesigned transistors to more favor lower-power operation at the expense of some performance. Extrapolation, therefore, will be inaccurate. To remove these inaccuracies, myCACTI uses device parameters that are derived using SPICE BSIM4 equations directly from SPICE models for each of the targeted nanometer process (130nm, 90nm, 65nm, 45nm and 32nm)[8].

To implement these BSIM4 equations, myCACTI follows a two-pass flow shown in Figure 3-17. The shaded part shows the first-pass of the flow which is responsible for creating a file that contains the individual device parameters. The first-pass gets individual BSIM4 SPICE decks as inputs (in our case, 130nm, 90nm, 65nm, 45nm and 32nm NMOS and PMOS predictive SPICE decks from PTM). A set of Perl and Matlab

8. myCACTI uses predictive SPICE models [PTM2006] for the data that will be shown in later chapters. The myCACTI flow, though, can accomodate any SPICE deck as long as it is in the proper SPICE BSIM4 format. This way, once a SPICE deck is available for an existing process,myCACTI can utlize this to perform cache design space explorations targeted for that particular process.

scripts then solve the BSIM4 equations to solve for the different device parameters like device effective resistance per micrometer, gate and diffusion capacitance per unit width and unit area, subthreshold leakage currents per unit width, and gate leakage tunneling currents per unit width and unit area. In addition, different versions of these parameters are generated to model transistors with different threshold voltages[9]. The different device parameters for the different technology nodes are then collected together into a file "technology.c", which is then merged into myCACTI after recompilation[10]. For simulation runs that use a stable pool of SPICE decks, no recompilation of myCACTI needs to be done after the first one, minimizing the inconvenience introduced by recompilation.

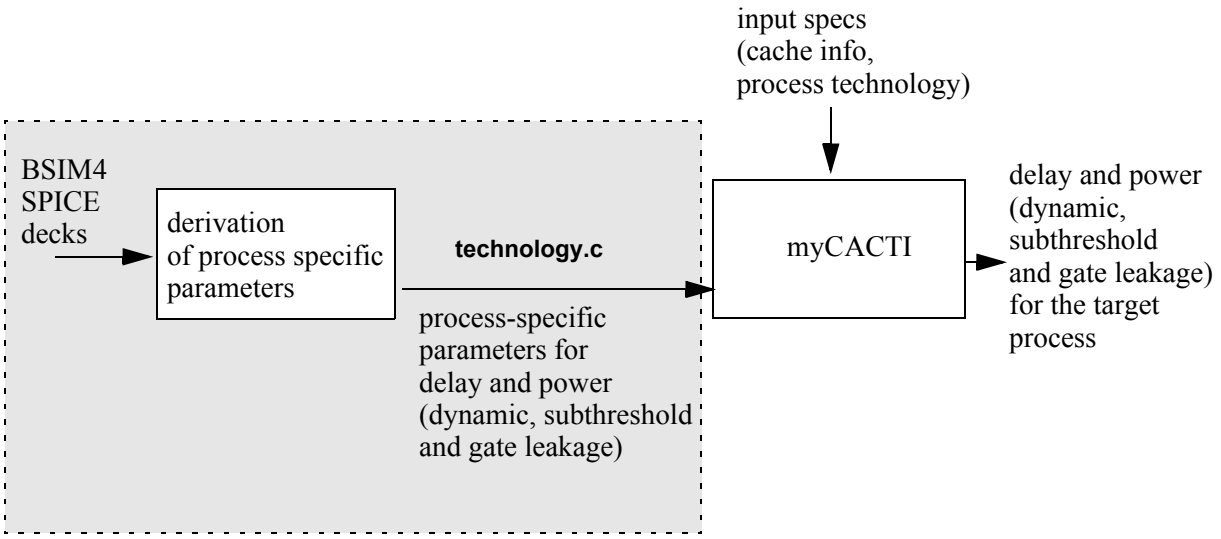


Figure 3-17: myCACTI program flow including device characterization.

9. Note that transistors with different threshold voltages not only have different subthreshold leakage behavior, but also different current driving capabilities and gate leakage tunneling currents.

10. An alternative to this flow is to use a one-pass implementation where myCACTI can simply expect a list of SPICE models to use, then perform all the computation once during runtime. The two-pass flow was chosen because of ease-of-use and debugging of the model during development, along with the fact that this step is only very infrequently performed since the "technology.c" file can include multiple versions of the model such that the first-pass of the flow only has to be reexecuted when a new model is going to be used. But after this recompilation, any future usage of myCACTI can utilize this new model and all the other previous SPICE models such that no recompilation is needed for future runs.

After recompilation, myCACTI can now be used normally the same way CACTI and eCACTI are invoked. The particular device parameters to used are then determined by the input switches specifying the particular technology to be used.

Appendix A contains all the relevant SPICE BSIM4 equations that were used in the derivations of the device parameters.

3.4.8 Interconnect characterization

Similar to device characterization, both CACTI and eCACTI use interconnect parameters that are based on a 0.80um reference process technology that is then extrapolated to the target process[11]. Again, this is inaccurate since the BEOL-stack does not scale linearly. In some cases, two succeeding generations might actually have very similar BEOL-stacks, with only the transistors (and maybe one level of interconnect out of the entire BEOL-stack) being scaled.

To remove these inaccuracies, myCACTI models realistic BEOL-stacks that are specific to the different process technologies being targeted. Specifically, realistic wire widths, thickness, height and spacing, along with process-speicific parameters like metal and dielectric materials, are used for each layer of a given process for every targeted process to compile a list of resistance and capacitance per unit length for every layer and process generation. This is especially important in caches, as interconnect parasitics play a very significant role because of the large areas occupied by the cache.

myCACTI utilizes equations from PTM-interconnect [PTM2006, Wong2000] to solve for the interconnect characteristics, and these are shown in Figure 3-18. In addition, every interconnect modeled in the cache, as shown in Figure 3-13, Figure 3-14 and Figure 3-15 are assigned specific metal layers as determined by optimality and practicality of placement. In addition, the default numbers used by myCACTI for its BEOL stack are shown in Table 3-6.. Finally, the default layer assignment of different signals are shown in Table 3-6.

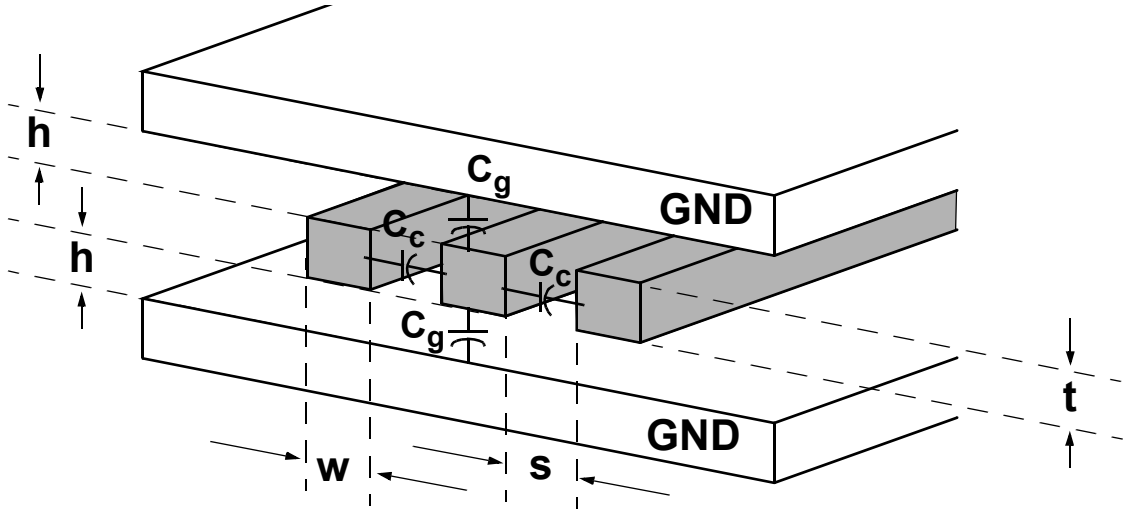
Table 3-5: myCACTI BEOL-stack parameters

tech node	Interconnect level	width (w)	spacing (s)	thickness (t)	height (h)	dielectric constant	metal type
130nm	Global	0.70µm	0.70µm	0.50µm	0.50µm	3.0	copper
	Intermediate	0.22µm	0.20µm	0.32µm	0.32µm	3.0	copper
	Local	0.18µm	0.18µm	0.30µm	0.30µm	3.0	copper

11. More accurately, it is actually extrapolated only for area computations. For delay and power, the 0.80um numbers are used as is, since the entire computation uses 0.80um numbers to produce 0.80um-based results, which are then scaled down linearly to the targeted process.

Table 3-5: myCACTI BEOL-stack parameters

tech node	Interconnect level	width (w)	spacing (s)	thickness (t)	height (h)	dielectric constant	metal type
90nm	Global	0.60µm	0.60µm	0.45µm	0.45µm	3.0	copper
	Intermediate	0.20µm	0.18µm	0.32µm	0.32µm	3.0	copper
	Local	0.15µm	0.15µm	0.28µm	0.28µm	3.0	copper



$$C_g = \varepsilon\left[\frac{w}{h} + 2.04\left(\frac{s}{s+0.54h}\right)^{1.77}\left(\frac{t}{t+4.53h}\right)^{0.07}\right]$$

$$C_c = \varepsilon\left[1.14\frac{t}{s}e^{-4\frac{s}{s+8.01h}} + 2.37\left(\frac{w}{w+0.31s}\right)^{0.28}\left(\frac{h}{h+8.96s}\right)^{0.76}e^{-2\frac{s}{s+6h}}\right]$$

$$C_{total} = 2C_g + 2C_c$$

$$R = \rho\frac{L}{w \cdot t}$$

Figure 3-18: myCACTI derivation of interconnect parameters. Capacitance derivation is for capacitance per unit length [PTM2006, Wong2000], while resistance equation is for absolute resistance.

Table 3-5: myCACTI BEOL-stack parameters

tech node	Interconnect level	width (w)	spacing (s)	thickness (t)	height (h)	dielectric constant	metal type
65nm	Global	0.50µm	0.50µm	0.40µm	0.40µm	3.0	copper
	Intermediate	0.15µm	0.14µm	0.28µm	0.28µm	3.0	copper
	Local	0.15µm	0.14µm	0.28µm	0.28µm	3.0	copper
45nm	Global	0.40µm	0.40µm	0.35µm	0.28µm	3.0	copper
	Intermediate	0.13µm	0.13µm	0.25µm	0.25µm	3.0	copper
	Local	0.10µm	0.10µm	0.15µm	0.175µm	3.0	copper
32nm	Global	0.35µm	0.35µm	0.32µm	0.25µm	3.0	copper
	Intermediate	0.12µm	0.12µm	0.22µm	0.22µm	3.0	copper
	Local	0.08µm	0.08µm	0.15µm	0.15µm	3.0	copper

Table 3-6: Default myCACTI interconnect layer assignment

Cache signal	Interconnect layer
Address buffer to pre-decoders	Intermediate interconnect layer
Predecoders to Wod-line decoders	Intermediate interconnect layer
Wordlines	Intermediate interconnect layer
Bitlines	Local interconnect layer
Data output lines	Global interconnect layer

3.4.9 Via parasitic capacitance

Both CACTI and eCACTI model interconnect RC parasitics in their simulation. These are very important since interconnect RC parasitics often account for a significant amount of the parasitics in cache signals (e.g. wordlines, bitlines, data output, etc.). Unfortunately, both cache design tools ignore the effects of via capacitances. Although via capacitances are, admittedly, significantly smaller compared to metal capacitances and are safely ignored in signals that will expectedly have relatively few via connections (e.g. point to point signals that have only 1 via at each end of the line plus a few others in the middle of the signal when the route switches layers), there are some signals in a cache that will have at least one via (and most probably more than

one) for every row and/or column of the memory array such that the via capacitance accumulates and may become a non-negligible component of the total capacitance.

myCACTI improves on CACTI and eCACTI by accounting for these via capacitances where necessary. In addition, myCACTI differentiates vias such that a signal going from a transistor to a lower-level metal will have significantly lower via compared to a signal going from a transistor to a higher-level metal.

3.4.10 Single-ended sensing

By default, myCACTI implements the same traditional differential sensing scheme as CACTI and eCACTI, as shown in Figure 3-19(a). But as the supply voltage becomes smaller and as process variability becomes larger with each technology generation, it becomes more difficult to maximize the advantages of low-swing differential sensing. These advantages are mainly the speed and power savings involved in needing only a small change in input voltage in order to produce a full-swing version. Currently, industry is forced to include error margins during the operation in order to ensure correct behavior. For example, even if only a 100mV change in the bitline voltage is required by an ideal sense amplifier in order to produce full-swing voltage properly, different problems such as noise and process variability force the designer to produce a change significantly larger than the 100mV minimum. In the process, this margining eats into the delay and power savings of differential sensing.

In comparison, single-ended sensing requires a full-swing change in the bitlines in order to produce the output logic. As we explain later, as the advantages in low-swing differential sensing are reduced with each generation and, at the same time the disadvantages of single-ended sensing are also reduced, single-ended sensing starts to become a more attractive technique that provides designers a much more robust and

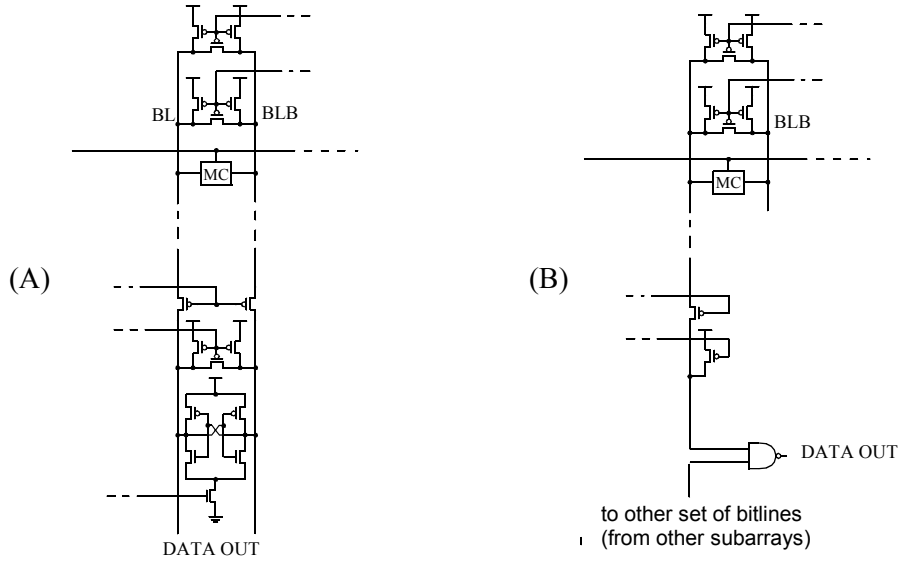


Figure 3-19: Sensing schemes. (A) Low-switng differential sensing using a drain-fed latch sense amp and (B) Full-swing single-ended sensing using a simple NAND gate to perform sensing that assumes bitline can be fully discharged to ground.

easier to design sense-amplifier that has only slightly worse delay and power behavior at deep nanometer nodes compared to a traditional low-swing differential amplifier.

Figure 3-19 shows the low-swing differential sensing scheme that myCACTI (along with CACTI and eCACTI) uses by default, along with an alternative full-swing single-ended sensing mechanism [Weiss2002].

3.4.11 myCACTI limitations

Although myCACTI introduces significant improvements over eCACTI and CACTI, it does impose some limitations which may not be present in the previous two tools.

Fully-associative modeling

myCACTI does not support fully-associative caches. This was a decision that was made based on the fact that fully-associative caches are typically done using custom-design flows that have different assumptions and use different tradeoffs compared to regular caches. Although CACTI and eCACTI do have some sort of support for full-associativity, the resulting implementation is still not an accurate representation of how a fully-associative memory structure is implemented in industry. Extending the assumptions of myCACTI into a fully-associative cache will simply result in inefficient circuits, resulting in modeling numbers that are significantly more pessimistic compared to industry-level fully-associative caches. In other words, fully-associative caches are vastly different from a typical cache, and forcing the assumptions of a typical cache design tool to also accomodate fully-associative caches will result in misleading data.

Subbanking

CACTI (and, by extension, eCACTI), have some nominal support for subbanking, which simply uses glue logic and replicates all the relevant buses and glues together multiple discrete caches. As such, along with the fact that myCACTI models a pipelined cache, myCACTI does not explicitly support subbanking, instead giving the responsibility of extending its implementation to the user. With explicit pipelining, this is a simple extension to make, as signals are partitioned cleanly, facilitating the use of glue logic before and after the cache to easily support subbanking.

Multiple Read/Write ports

As opposed to CACTI and eCACTI, myCACTI only supports the design of caches with 1 read/write port. This is again, a conscious design decision to reflect industry preference in the design of caches. Using multiple ports for a cache drastically increases the area occupied by each memory cell, and consequently, the storage array. This results in much less storage capacity per given area, which is something that is seldom an acceptable decision in commercial design. The accepted way of doing it in industry is to implement subbanking with glue logic that is external to the cache. With multiple banks, multiple accesses can be done to

the cache at any given time as long as different banks are used by the different accesses. This way, the storage capacity of the cache per unit area is not degraded, even at the small expense of IPC whenever accesses need to stall during the (mostly infrequent) occurrence of more than one access to a single subbank at any given time.

3.5 Summary

This chapter summarizes the different capabilities and features of CACTI and eCACTI. It also points out the limitations of these two cache design tools, and describes the improvements, enhancements and additional features introduced by myCACTI. The following chapter produces quantitative comparison between the different design tools.

CHAPTER 4 CACTI/eCACTI vs. myCACTI Comparative Studies

4.1 INTRODUCTION

In this section, we provide some different detailed comparisons that we perform between CACTI/ eCACTI and our own version of the tool to show any differences in behavior between the tools. In addition, we give some details on the reference configuration that we use throughout the comparisons.

The comparisons we perform here can be divided into two categories. The first category studies the major limitations of CACTI/eCACTI while the second category studies new parameters that are useful to study in nanometer cache design. We will show that the differences between CACTI/eCACTI and myCACTI are substantial and use of CACTI/eCACTI for nanometer caches will result in misleading conclusions.

4.2 Background of Comparisons

In this section, all the comparative studies that are done to determine the difference between CACTI/eCACTI runs and myCACTI runs are explained in detail.

4.2.1 Validity of CACTI/eCACTI scaling

CACTI uses hardcoded parameters derived from a 0.80um process technology to produce a solution based on the user's specification. To produce data for a different process technology, CACTI simply scales the data produced using the 0.80um run (mainly delays and dynamic power), and performs a simple linear scaling of these numbers based on the target technology. For example, a run targeted for a 0.20um process will simply take all the data produced using the 0.80um run and scale them down by a factor of 4 (0.80um/0.20um).

eCACTI improves this operation, but only partly. eCACTI uses hardcoded parameters derived from multiple submicron and nanometer nodes (0.18u, 0.13u, 0.10u and 0.07u) in order to compute for the subthreshold leakage within a cache. Unfortunately, eCACTI retains the simple linear scaling of CACTI for computing cache delays and dynamic power.

myCACTI, in contrast to CACTI and eCACTI, uses hardcoded process-specific parameters derived from multiple process technologies (0.25um, 0.18um, 0.13um, 90nm, 65nm, 45nm and 32nm) for the entire solution in order to eliminate the need to extrapolate data values from older process technologies. In addition, myCACTI comes with a feature where users can update these hardcoded parameters by using their own SPICE decks so a design space exploration can be done not just for a generic process node, but for a specific foundry (as long as SPICE decks are available).

This particular comparison explores the validity of this simple linear scaling of cache delays and dynamic power by doing simulations using the parameters for the targeted process, and comparing the results to simulations done for an older process and linearly scaled down to the targeted process.

Figure 4-1 shows a graphical view of how CACTI, eCACTI, and myCACTI perform simulations.

4.2.2 Transistor effective length

Both CACTI and eCACTI assume that the effective minimum length of a transistor[1] (Leff) is equal to the technology node's drawn length (Ldrawn). For example, a 0.25um process will have an Ldrawn of 0.25um, and both CACTI and eCACTI assume that the effective length is the same and set it to be 0.25um also[2].

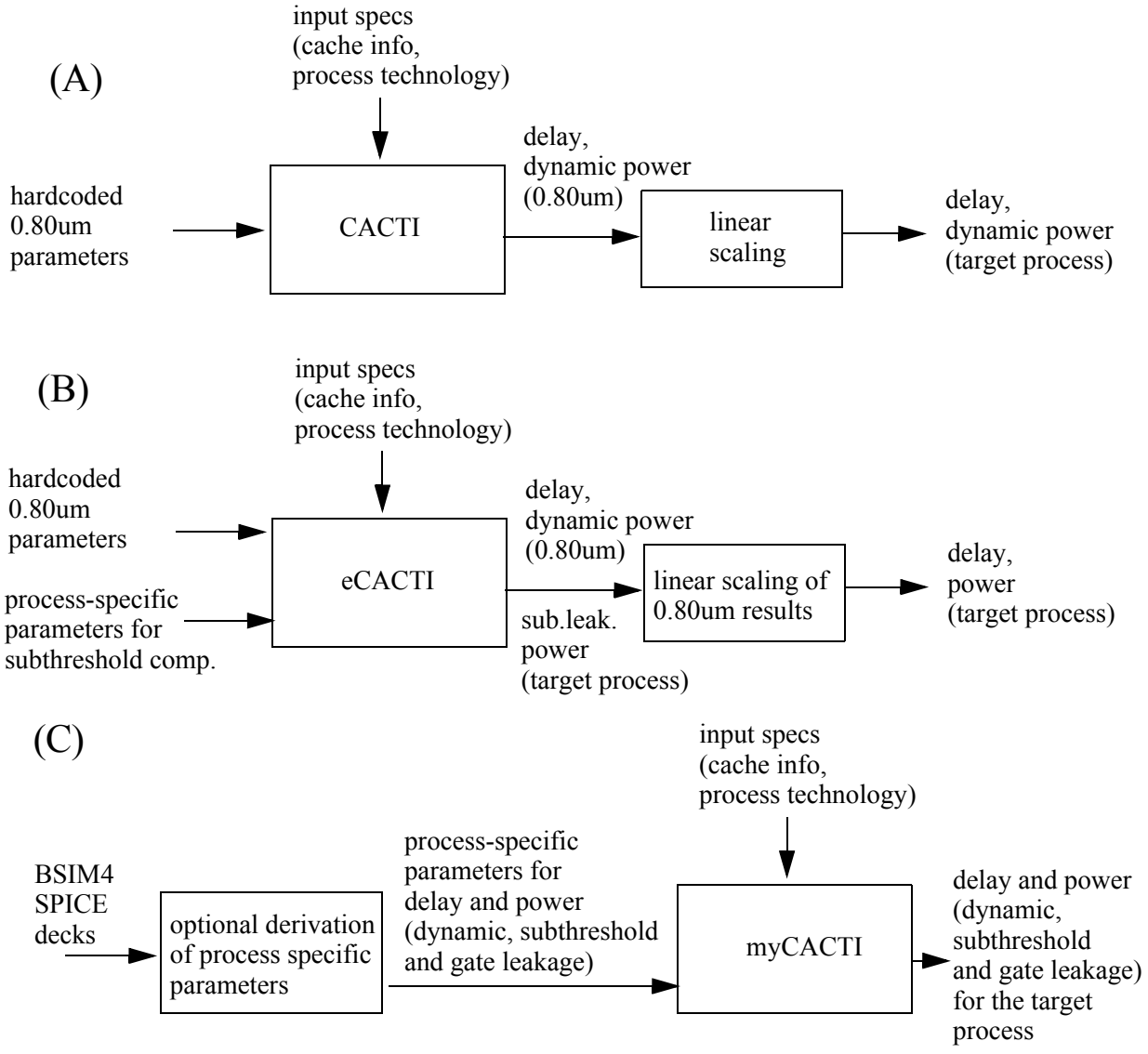


Figure 4-1: High-level view of computation flow for (a) CACTI, (b) eCACTI and (c) myCACTI.

In reality, MOSFETs are fabricated such that there are overlaps between the gate and the source and drain junction. This is done (among other reasons), to ensure that a channel induced under the gate will always bridge the source-drain terminals even if the masks used to fabricate the transistor shift by some reasonable amount. Fabricating a transistor without this overlap completely relies on the perfect alignment of the gate and source junctions during fabrication, which is obviously a very impractical (and dangerous) assumption.

myCACTI accounts for this gate-source/drain overlap region by adjusting the effective length of the transistor in accordance to the expected overlap. Specifically, myCACTI's explicit use of BSIM4 SPICE equations allows it to take advantage of BSIM's inherent support of this overlap region. Consequently, all the operations in myCACTI account for this difference between the process drawn length and the effective length.

Figure 4-2 demonstrates these concepts visually.

4.2.3 Via parasitic capacitance

Both CACTI and eCACTI model interconnect RC parasitics in their simulation. These are very important since interconnect RC parasitics often account for a significant amount of the parasitics in cache signals (e.g. wordlines, bitlines, data output, etc.). Unfortunately, both CACTI and eCACTI ignore the effects of via capacitances. Although via capacitances are, admittedly, significantly smaller compared to metal capacitances and are safely ignored in signals that will expectedly have relatively few via connections (e.g. point to point signals that have only 1 via at each end of the line plus a few others in the middle of the signal when the route switches layers), there are some signals in a cache that will have at least one via (and most probably more than

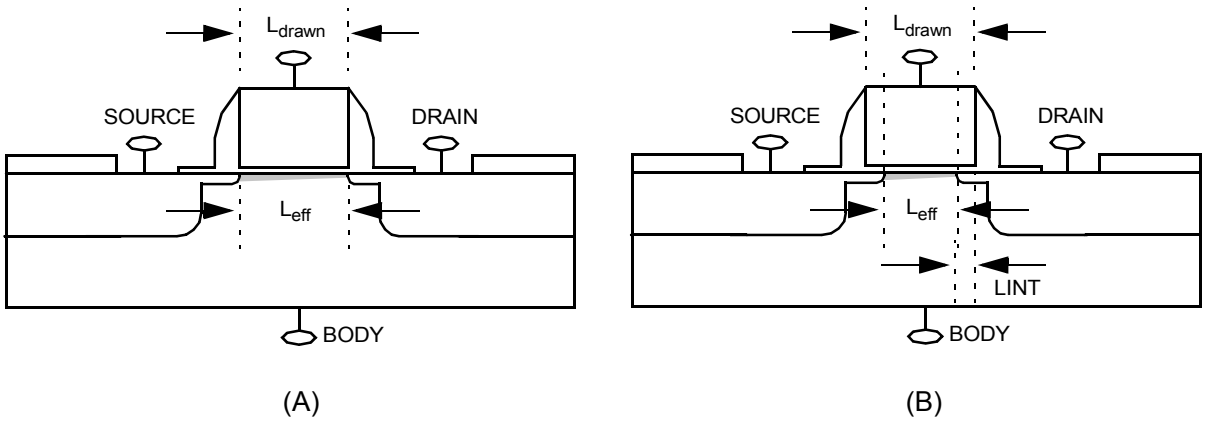


Figure 4-2: Effective transistor length assumptions for (a) CACTI and eCACTI, and (B) myCACTI. For CACTI and eCACTI, Leff is the same as Ldrawn. In myCACTI, Leff is derived from Ldrawn by using a process-specific SPICE parameter, LINT.

1. Leff is simply the transistor length value that is actually used in SPICE equations.

2. Strictly speaking, CACTI, and to some extent eCACTI, uses only an Ldrawn of 0.80um (as explained in the previous section) and just perform linear scaling of the results to adjust it to the desired process technology.

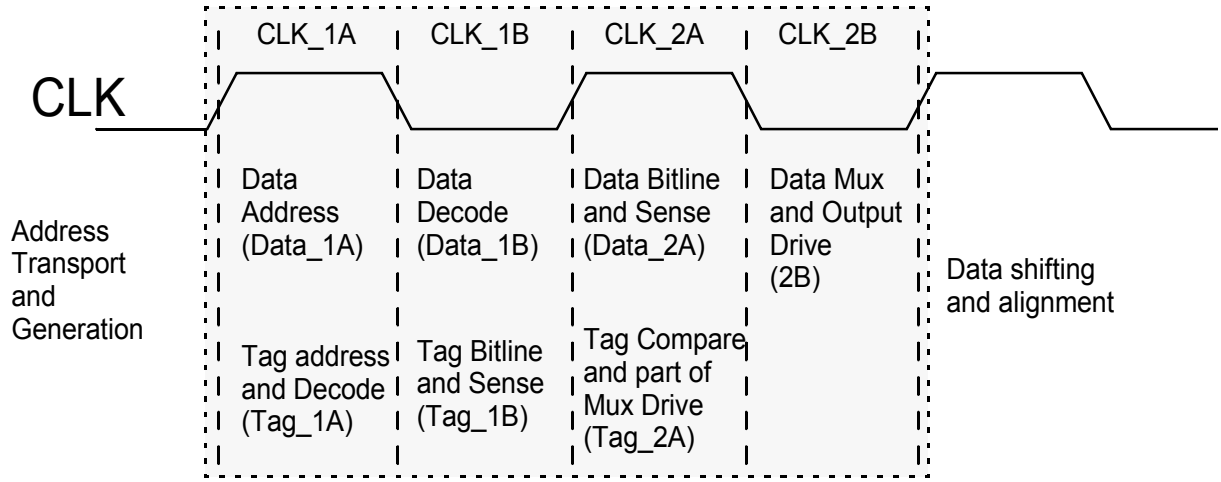


Figure 4-3: Cache pipeline diagram. This pipeline diagram indicates the different stages of the cache pipeline, with the shaded regions showing which stages are included in the modeling. Here, the main cache block has a 2-cycle latency. Also assumed by the diagramis the presence of the TLB-translated address (if applicable) by the start of the CLK_2A phase of the clock in time for use by the Tag_2A stage.

one) for every row and/or column of the memory array such that the via capacitance accumulates and may become a non-negligible component of the total capacitance.

myCACTI improves on CACTI and eCACTI by accounting for these via capacitances where necessary. In addition, myCACTI differentiates vias such that a signal going from a transistor to a lower-level metal will have significantly lower via compared to a signal going from a transistor to a higher-level metal.

4.2.4 Pipelining comparison

To keep up with the speed of a fast microprocessor core while providing sufficiently large storage capacities, caches are pipelined to subdivide the various delays in the cache into different stages, allowing each individual stage to fit into the core's small clock period. Figure 4-3 shows a typical pipeline diagram for a cache. The given timing diagram shows operations being performed in both phases of the clock. Figure 4-4 shows a possible implementation of a pipeline latch that easily facilitates this phase-based operation.

Both CACTI and eCACTI use implicit pipelining through wave pipelining, relying on regularity of the delay of the different cache stages to separate signals continuously being shoved through the cache instead of using explicit pipeline state elements. Unfortunately, this is not representative of modern designs, since cache wave-pipelining is not being used by contemporary microprocessors [Weiss2002, Riedlinger2002]. Although wave pipelining has been shown to work in silicon prototypes [Burleson1998], it is not ideally suited for high-speed microprocessor caches targeted for volume production which have to operate with significant process-voltage-temperature (PVT) variations. PVT variations in a wave-pipelined cache cause delay imbalances which, in the worst case, lead to signal races that are not possible to fix by lowering the clock frequency. Hence, the risk for non-functional silicon is increased, resulting in unattractive yields. In addition,

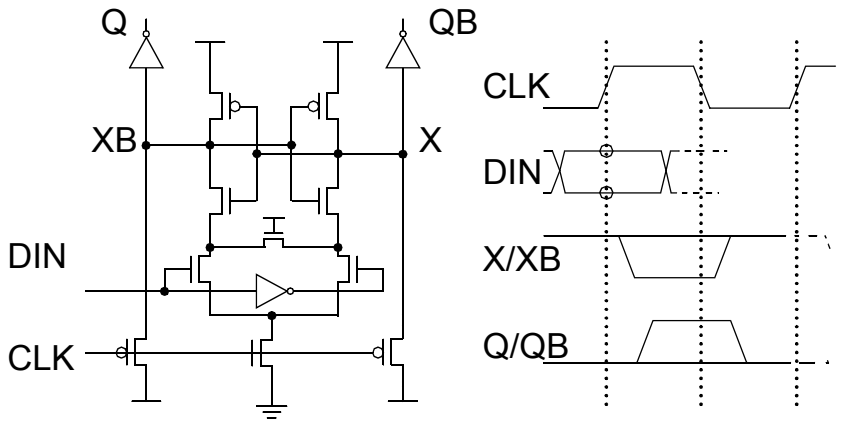


Figure 4-4: Pipeline latch. An example of a pipeline latch that can be used to implement the phase-based operations in the cache [Montanaro1996, Gronowski1996]. Also shown is the latch's timing diagram

wave-pipelining does not inherently support latch-based design-for-test (DFT) techniques that are critical in the debug and test of a microprocessor, reducing yields even further. On the contrary, it is easy to integrate DFT "scan" elements inside pipeline latches that allow their state to be either observed or controlled (preferably both). This ability facilitates debugging of microprocessor circuitry resulting in reduced test times that directly translate to significant cost savings. Although not immediately obvious at first, these reasons make it virtually necessary to implement explicit pipelining for high-volume microprocessors caches.

myCACTI models explicit pipelining (using the pipeline diagram shown in 4-3) to demonstrate the impact of pipelining a cache on its delay and power behavior.

4.2.5 Number of interconnect layers

As previously mentioned, CACTI and eCACTI model interconnect RC when modeling the cache. Unfortunately, botch CACTI and eCACTI limit themselves by using only a single, generic metal layer for every interconnect in the cache regardless of its requirement. This is inaccurate because metal interconnects in cache are typically routed in the layer that has the best match for the requirements of that particular interconnect. For example, bitlines will most likely be routed using the lowest level metal layers in order to minimize the accumulation of excessive via parasitic capacitance had a higher level metal been used. Moreover, although we often want to route signals in the metal layer that best matches its requirement, this is not always possible, as multiple signals may want to populate this particular metal layer and there will often not be enough space to accomodate both. Consequently, one (or more) of these signals will have to be routed using a more inefficient metal layer.

Because of these reasons, assuming a single layer interconnect for use in the entire cache will provide inaccurate results compared to something more realistic (the degree of inaccuracy, as we will demonstrate in the results section, is significant). Making it worse is the fact that modeling single-layer metal for the cache

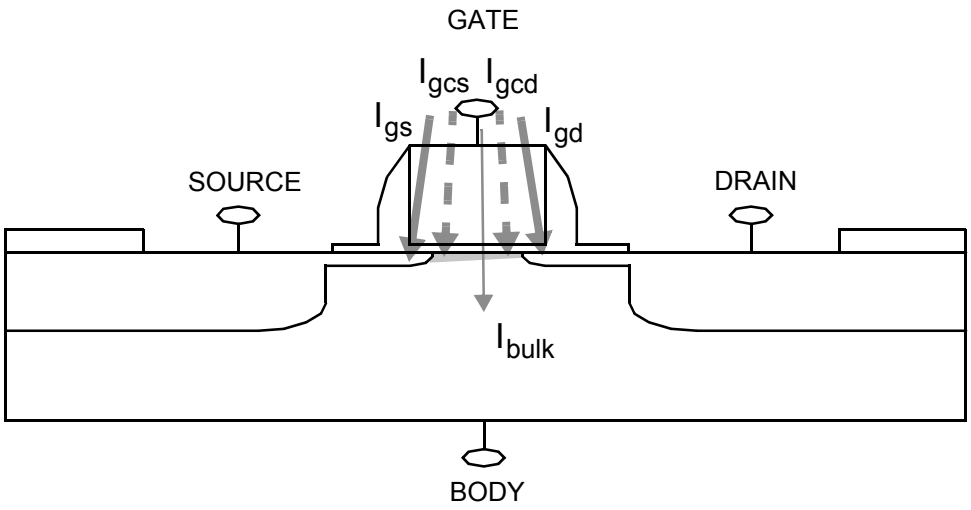


Figure 4-5: MOSFET Gate leakage tunneling currents. The five different gate leakage tunneling currents in a MOSFET are shown. These five can be categorized into three groups: the channel tunneling current, the source/drain junction overlap tunneling current, and the bulk tunneling current.

will give optimistic numbers for some part of the cache, while at the same time possibly giving pessimistic numbers for the rest.

myCACTI improves on this limitation[1] by solving for different interconnect characteristics at runtime and then using the correct interconnect for different signals in the cache (e.g. local interconnect for bitlines, intermediate interconnects for predecode and wordline signals, global interconnect for data out, etc.). In addition, the determination of what interconnect to use for which particular signal can be user determined in order to suit a particular user's requirements.

4.2.6 Gate leakage

In modeling power dissipation, CACTI only accounts for dynamic power. eCACTI extends the CACTI model by including a model to account for static power dissipated by subthreshold leakage currents. As process technologies get deeper and deeper into the nanometer region, the circuit community has been widely concerned about the limiting effects of gate leakage tunneling currents.

myCACTI addresses this by including a model that accounts for gate tunneling leakage currents. Consequently, myCACTI now partitions total power into dynamic power (switching power), and static power (as dissipated by the subthreshold leakage currents and gate leakage tunneling currents). myCACTI accounts for gate leakage by implementing the SPICE BSIM4 model that solves for gate leakage tunneling currents in a MOSFET.

Figure 4-5 shows the different components of gate leakage tunneling currents that are included in the myCACTI gate leakage model.

---

1. myCACTI does this by using BPTM interconnect models and realistic BEOL stacks.

4.2.7 Static/dynamic decoding

CACTI and eCACTI both use static CMOS gates for implementing the decoder. The main advantages of using static CMOS is the ease of design and verification associated with it. Unfortunately, static CMOS gates are typically inefficient and slow compared to dynamic logic. In addition, static CMOS gates performance is really poor for high fan-in circuit design, which is essentially what decoder design is. Designing a decoder, at its simplest, is simply the design of a wide AND gate that enables a single signal (the wordline) for a given input address. Although this wide AND is typically broken up into multiple stages (e.g. the predecoder and the wordline decoder stages), each stage still needs to implement either ANDs (or NANDs) or ORs (or NORs) in order to faciliate the decode operation.

The problem is that the early stages in the decoder will fanout to potentially a large number of the downstream gates, such that if the load that a single gate presents is high, it is amplified and degrades the efficiency of the decoder. For example, a decode hierarchy that combines the result of three 3-to-8 decoders will have 512 rows, where each row combines three signals from a set of 24 to produce the final wordline. In this scheme, each predecoder output fans out to 64 of the wordline decoders (where each wordline decoder will be some sort of AND gate). ANDs (or NANDs) implemented with dynamic CMOS typically only have to implement the pulldown NMOS network, while static CMOS have to implement both the pullup PMOS network and the pulldown NMOS network -- drastically increasing the loading presented by a single gate. The predecoder has to drive 64 of these loads (plus the interconnect and via capacitances) adequately. An inefficient wordline decoder will have its inefficiency amplified, requiring a very strong predecoder in order to drive it. Increasing the predecoder size, in turn, stresses the design of the upstream logic (mainly the address buffers), especially since these address buffers again drive multiple predecoders, again amplifying any inefficiency.

It has been proven [Amrutur2000] that the optimal implementation of a decoder given these loading conditions are ones where the front-end stages are as close to minimum-sized as possible such that even if a large number of these loads have to be driven by a single driver, the accumulated load is not excessively large. Dynamic decoders have significant advantages in this aspect since it is easily realizable to have a very small input loading since the PMOS pullup network does not have to be implemented. In addition, the logic can be transformed such that the input transistors are parallel to each other such that increasing the fan-in does not require the size of each transistor to change (e.g. increasing the fan-in of a NAND gate requires the size of the NMOS transistors to be increased in order ot maintain the same drive strength, while increasing the fan-in of a NOR gate does not require the NMOS widths to be updated).

myCACTI implements dynamic decoding by using dynamic SCL logic for the address buffers, SCL DRCMOS for the predecoders, and simple DRCMOS logic for the wordline drivers.

Figure 4-6 shows the high-level decoding scheme used by CACTI, eCACTI and myCACTI, showing the different parts of the decoder as separate black boxes. Figure 4-7 shows the details of the different decoing schemes that will be compared in this study. myCACTI supports both schemes, with dynamic decoding being used as the default.

4.2.8 Dual-Vt process technologies

CACTI assumes a single Vt for its calculation of cache delay and power. eCACTI extends this by including a user-controlled option for dual-Vt transistors . Unfortunately, eCACTI only accounts for the effect of dual-Vt on the subthreshold leakage modeling, leaving delay computations unaffected. This can be justified by restricting the use of high-Vt transistors only on devices that are not in the critical path (which is partly what myCACTI also does), but in some select cases, this is impractical. This applies the most to the 6T memory cell. During a read, the access and driver transistors in the 6TMC cell are turned on to discharge one side of the bitlines. This path is typically in the critical path such that, from a speed perspective, it is advantageous to implement these two transistors as low-Vt. Unfortunately, as we will show in later sections, a significant part of the subthreshold leakage power dissipated by the cache can be attributed to the bitline discharge, such that implementing the access and driver transistors as high-Vt reduces the bitline leakage by almost an order of magnitude. This means that some sort of tradeoff has to be done to choose which of the two requirements are

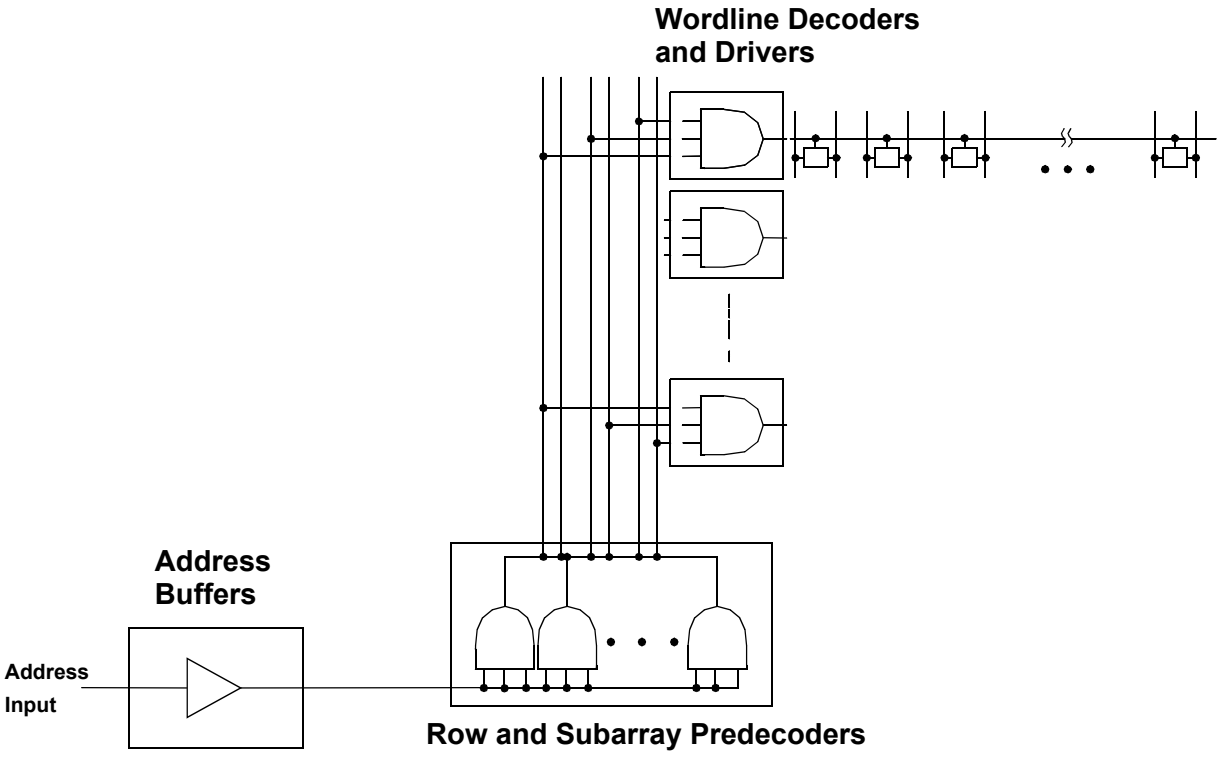


Figure 4-6: Address decoder. The high-level block diagram of the address decoder used by CACTI, eCACTI and myCACTI are shown.

more important. As power becomes a first-order design parameter, cache designers are typically implementing the access and driver transistors as high-Vt, choosing to incur the slight delay degradation in exchange for a very substantial savings in power. In pipelined designs where it can be proven that the critical path is determined by a different stage, this decision is a win-win.

myCACTI implements multi-Vt transistors in its different circuits as shown in Figure 4-7 and Figure 4-8. Figure 4-7 shows a multi-Vt implementation of the address decoders, while Figure 4-8 shows a multi-Vt implementation of the bitline circuits. The implemented transistor threshold designation mostly adheres to the rule of thumb where all critical path devices are implemented in low-Vt, with the exception of the memory cell arrays, which are implemented as high-Vt even though they may potentially be in the critical path, with the knowledge that this slight delay sacrifice will provide a substantial power savings. These multi-Vt circuits,

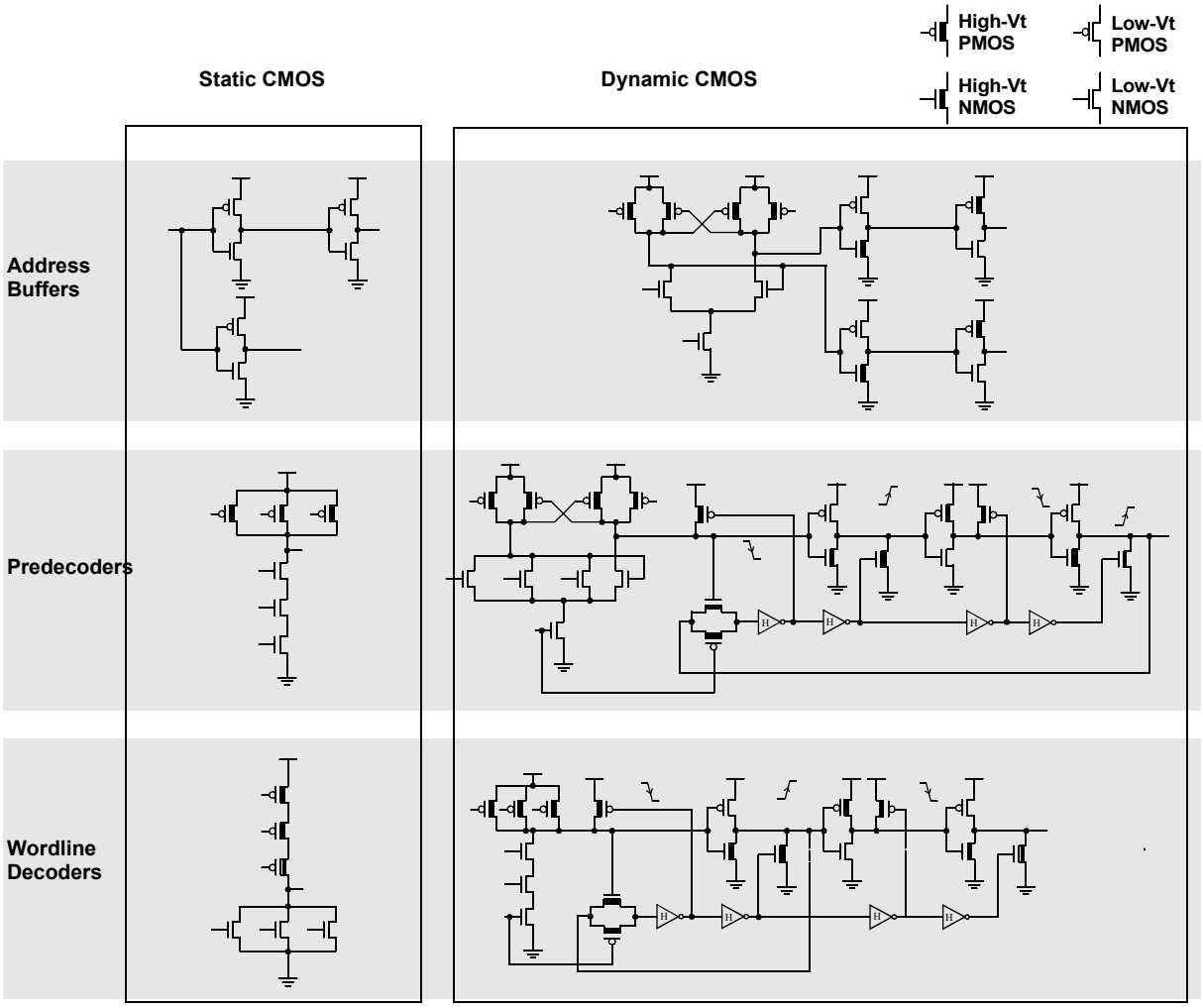


Figure 4-7: Static vs. dynamic decoding. Shown are the different logic families used for the decoders. For the static decoder, all the circuits in the address bufferrs, predecoders and wordline decoders use full-CMOS gates. For the myCACTI dynamic decoders, the address buffers are implemented in SCL dynamic CMOS, the predecoders in SCL-DRCMOS dynamic logic, and the wordline decoders in simple DRCMOS logic. Also shown are the devices that are designated as HVT devices when using a dual-Vt process. These devices revert back to an LVT implementation when a single-Vt process is being used.

of course, assume myCACTI runs that are configured for dual-Vt operation. For myCACTI runs that assume a single-Vt processes, all transistors revert back to LVT devices.

4.2.9 Single-ended sensing

By default, myCACTI implements the same traditional differential sensing scheme as CACTI and eCACTI, as shown in Figure 4-9(a). But as the supply voltage becomes smaller and as process variability becomes larger with each technology generation, it becomes more difficult to maximize the advantages of low-swing differential sensing. These advantages are mainly the speed and power savings involved in needing only a small change in input voltage in order to produce a full-swing version. Currently, industry is forced to include error margins during the operation in order to ensure correct behavior. For example, even if only a 100mV change in the bitline voltage is required by an ideal sense amplifier in order to produce full-swing voltage properly, different problems such as noise and process variability force the designer to produce a change significantly larger than the 100mV minimum. In the process, this margining eats into the delay and power savings of differential sensing.

In comparison, single-ended sensing requires a full-swing change in the bitlines in order to produce the output logic. As we explain later, as the advantages in low-swing differential sensing are reduced with

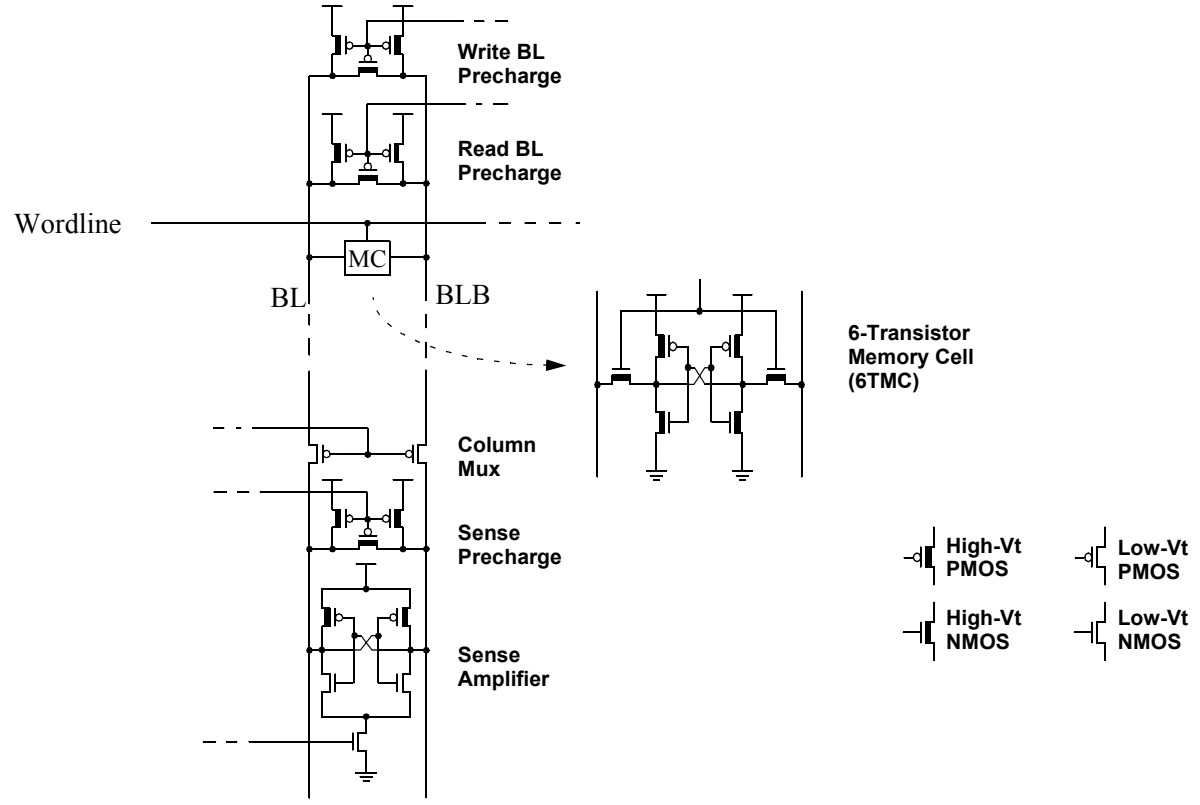


Figure 4-8: myCACTI bitline circuit dual-Vt implementation. This figure shows the LVT and HVT device designation for the circuits in the bitlines including the precharge devices, the memory cell arrays, the column mux transistors and the sense amplifiers.

each generation and, at the same time the disadvantages of single-ended sensing are also reduced, single-ended sensing starts to become a more attractive technique that provides designers a much more robust and easier to design sense-amplifier that has only slightly worse delay and power behavior at deep nanometer nodes compared to a traditional low-swing differential amplifier.

Figure 4-9 shows the low-swing differential sensing scheme that myCACTI (along with CACTI and eCACTI) uses by default, along with an alternative full-swing single-ended sensing mechanism [Weiss2002].

4.2.10 BEOL low-k study

In addition to modeling multi-layer interconnect using BPTM interconnect equations, myCACTI has the capability to recompute interconnect parameters at runtime. Among the parameters that can be modified is the dielectric constant of the inter-layer dielectric (ILD) of the interconnect.

This allows the modeling of materials with varying dielectric constant and enables the study of a cache with low-K dielectrics. These kinds of dielectrics are being foreseen as one of the ways to improve interconnect parasitics, and myCACTI allows this kind of study to be done on a cache in order to determine how much benefit is gained from using low-K dielectrics.

4.2.11 Combination of multiple parameters

In this study, instead of studying the differences of myCACTI versus CACTI and/or eCACTI in isolation, we compare the results when multiple changes are active simultaneously. Specifically, we simulate CACTI/ eCACTI results by enabling static decoding, disabling via capacitances, disabling the WL resistance in computing the wordline driver size, utilizing single-layer interconnects and finally use of Ldrawn as the effective length (disregardinng source/drain junction overlap). We compare this with a myCACTI run that

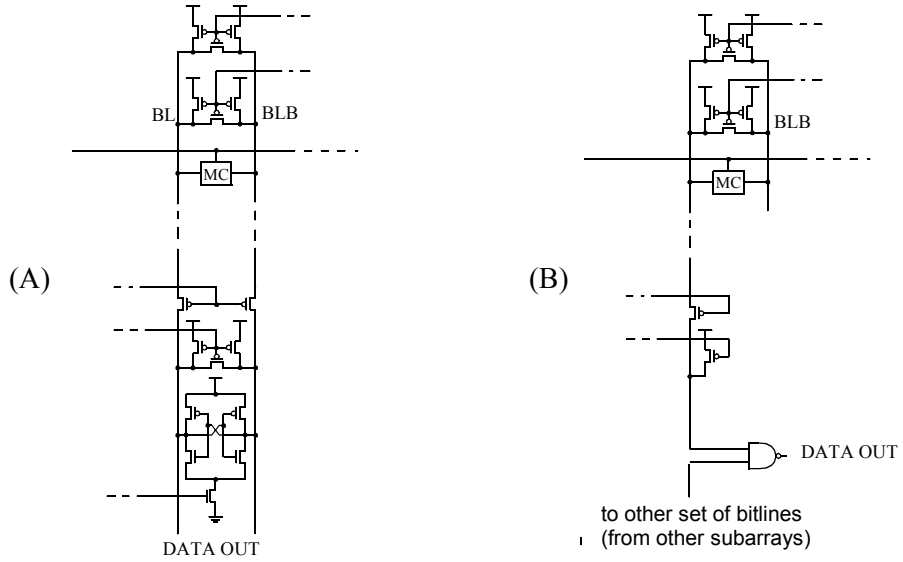


Figure 4-9: Sensing schemes. (A) Low-switng differential sensing using a drain-fed latch sense amp and (B) Full-swing single-ended sensing using a simple NAND gate to perform sensing that assumes bitline can be fully discharged to ground.

uses dynamic decoding, enables via capacitances, enables wordline resistances, accounts for the source/drain junction overlap in the use of Leff, and utilizes multi-layer interconnects.

4.3 Comparison Methodology

The fairest way of comparing myCACTI to both CACTI and eCACTI is to execute a set of runs for each of the three tools and to perform a side-by-side comparison of the three sets of numbers. One problem with this method is that, without inserting too much modifications to CACTI and eCACTI that change their opertaion (other than simple programming hooks to produce more detailed data), it is not possible to isolate the effects of each of the points of comparison that have been brought up. For example, it has been previously stated that eCACTI does not account for via parasitics and multi-layer interconnects. Although myCACTI can be configured to account for either or both (or none) of these two things, eCACTI is fixed and can only produce runs without via parasitics and multi-layer interconnects. We therefore can't perform a comparison where only a single one of them is active.

In addition, CACTI and eCACTI both possess a flaw that severely limits themin terms of properly exploring the design space. This flaw is the assumption of both CACTI and eCACTI to use fixed number of stages in their decoder. Figure 4-10 shows the decoder implemented in CACTI and eCACTI.

In CACTI, the strength of every gate except for the final wordline driver are hardcoded. Both the size in microns and the equivalent drive strength of each gate is shown in the figure. The sizing of the wordline driver is computed from the required effective transistor resistance that is needed to generate a desired wordline rise time given the capacitive loading present in the wordlines. This scheme has three major limitations. The first is that a large capacitive load may result in a large wordline driver such that the buffer right after the NOR gate will find it difficult to drive this load. Nevertheless, this is an inefficiency that is still captured in the program in terms of its effects on delay and power. The second limitation is that as CACTI searches for different implementations (in terms of the sixtet Ndwl:Nspd:Ndbl-Ntwl:Ntspd:Ntbl), the number of wordline drivers driven by the predecoder and/or the number of predecoders driven by the address buffer may change such that the pairing of load with its driver characteristics may be severely suboptimal. Nevertheless, this is again an inefficiency that is captured during program operation. The third limitation is that the input load that the cache presents at its input (as seen by the external circuits that drive the cache address lines) is unreasonably big -- in this case equivalent to a fanout of 100 1X-strength inverters. In a real circuit, this 100X inverter has to be driven by another gate, and it will significantly degrade the total delay of the cache. Even if a delay-optimal buffer chain was used to drive this input and interconnect delays are ignored, an additional delay of at least roughly 3 gate delays will be introduced (and of course, additional power dissipated by these gates have to be accounted for). Currently, this third limitation is something that is completely hidden to the user, resulting in unrealistically optimistic delay numbers.

eCACTI attempts to improve on CACTI's operation by making the size of every gate in the decoder chain variable and to be computed at runtime based on optimal loading. Instead of dynamically sizing only the final wordline driver, eCACTI continues the sizing algorithm and follows it until the final input address buffer, as also shown in Figure 4-10. At first glance, this seems to solve the first two limitations brought up for CACTI. Unfortunately, because the assumption of a fixed number of stages for the decoder was retained, more severe problems are actually introduced by this additional feature. Specifically, the dynamic sizing of the rest of the wordline circuits (as indicated by WdecNORn, WdecNORp, Wdecinvn, Wdecinvp), often result in these gates being upsized in order to properly drive the final wordline driver. The problem is almost exactly similar to the problem previously explained in the comparison of static and dynamic decoding, where multiple NOR gates are being driven by a single predecoder, such that an increase in the NOR gate input load is exacerbated. The sizing algorithm tries to catch up by sizing up the predecoder NAND gate in order to drive the multiple NOR gates. As this process continues, one can see that the decoder starts to have very unrealistic

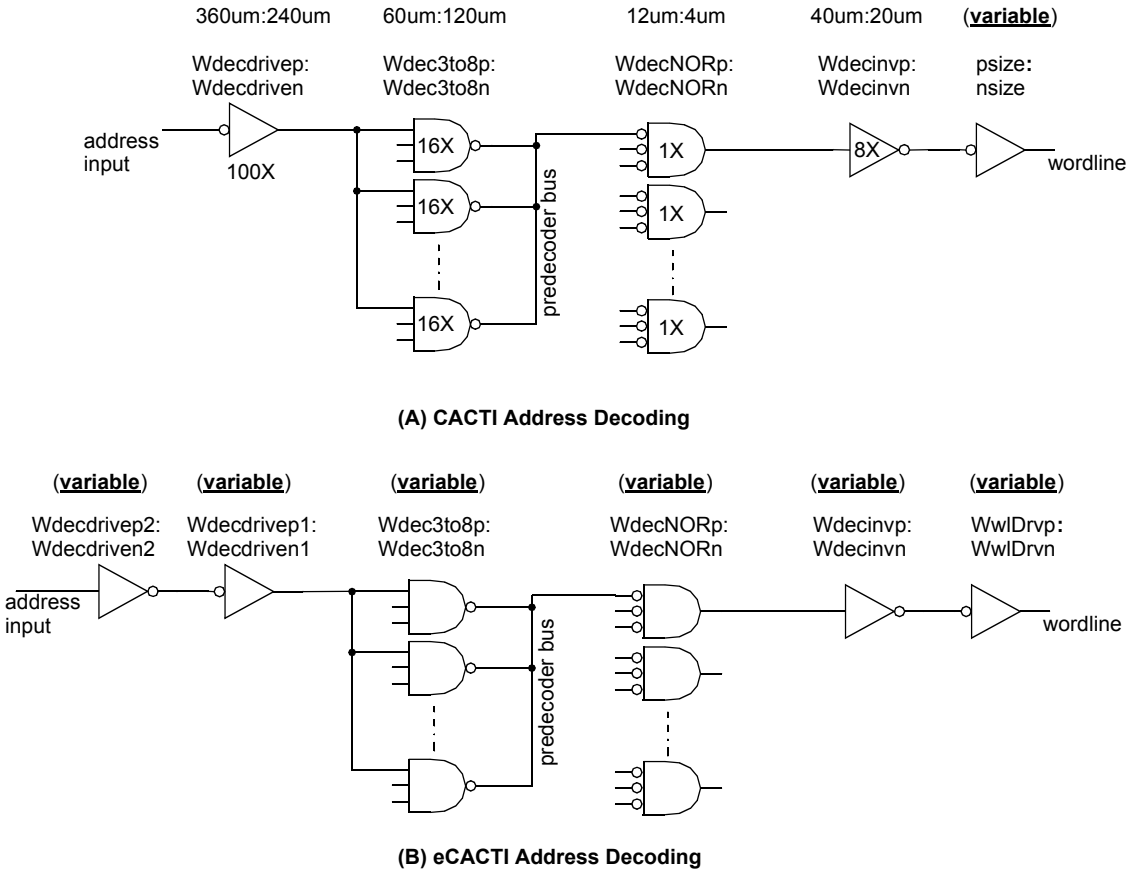


Figure 4-10: CACTI/eCACTI fixed-stage decoding. (A) CACTI address decoding and (B) eCACTI address decoding. Note that even though both decoders dynamically resize at least a part of their decode circuitry to account for loading, both decoders have fixed number of stages, limiting the optimality of their sizing. Also shown for the CACTI diagram is the strength of the fixed-size gates (everything except the final wordline driver). Note the 100X inverter at the address input. This presents a huge loading to the input driver, which will require at least two gates to drive properly, hiding some of the delays. The same holds true for eCACTI but is actually made worse by the fixed stage decoding and resizing every gate, as the input inverter becomes even bigger than its CACTI counterpart, hiding even more delay.

sizing, as we show in Table4-1 which lists the resulting transistor sizes for optimal implementations of various cache configurations, as generated by eCACTI for a 0.13um process technology.

Table 4-1: eCACTI transistor widths for some cache configurations for a 0.13um process

Config	WwlDrvn	WwlDrvp	Wdecinvn	Wdecinvp	WdecNORn	WdecNORp	Wdec3to8n	Wdec3to8p	Wdecdriven1	Wdecdrivep1	Wdecdriven2	Wdecdrivep2
128k_1w	1.70	3.40	0.56	1.12	0.18	1.11	7021.38	4680.91	82394.33	164788.00	27190.15	54380.30
128k_2w	1.70	3.40	0.56	1.12	0.18	1.11	7021.38	4680.91	82394.33	164788.00	27190.15	54380.30
128k_4w	1.70	3.40	0.56	1.12	0.18	1.11	7021.38	4680.91	82394.33	164788.00	27190.15	54380.30
128k_8w	1.70	3.40	0.56	1.12	0.18	1.11	1794.07	1196.04	42111.88	84223.91	13896.95	27793.84
16k_16w	1.70	3.40	0.56	1.12	0.18	0.74	68.59	45.73	1612.77	3225.54	532.21	1064.43
16k_1w	1.70	3.40	0.56	1.12	0.18	1.11	2.21	1.48	11.31	22.63	3.73	7.47
16k_2w	3.02	6.04	1.00	1.99	0.33	1.32	1.03	0.69	12.39	24.79	4.09	8.18
16k_4w	3.02	6.04	1.00	1.99	0.33	1.32	1.03	0.69	12.39	24.79	4.09	8.18
16k_8w	3.02	6.04	1.00	1.99	0.33	1.32	27.87	18.58	165.17	330.33	54.50	109.01
32k_16w	1.70	3.40	0.56	1.12	0.18	0.74	150.86	100.57	3544.45	7088.90	1169.67	2339.33
32k_1w	1.70	3.40	0.56	1.12	0.18	1.11	4.43	2.95	17.81	35.62	5.88	11.75
32k_2w	3.02	6.04	1.00	1.99	0.33	1.97	2.88	1.92	17.82	35.65	5.88	11.76

Table 4-1: eCACTI transistor widths for some cache configurations for a 0.13um process

Config	WwlDrvn	WwlDrvp	Wdecinvn	Wdecinvp	WdecNORn	WdecNORp	Wdec3to8n	Wdec3to8p	Wdecdriven1	Wdecdrivep1	Wdecdriven2	Wdecdrivep2
32k_4w	3.02	6.04	1.00	1.99	0.33	1.32	28.38	18.92	169.32	338.64	55.88	111.75
32k_8w	3.02	6.04	1.00	1.99	0.33	1.32	28.38	18.92	169.32	338.64	55.88	111.75
64k_16w	1.70	3.40	0.56	1.12	0.18	0.74	479.53	319.69	11259.61	22519.25	3715.68	7431.34
64k_1w	1.70	3.40	0.56	1.12	0.18	1.11	18.11	12.07	115.87	231.73	38.24	76.47
64k_2w	3.02	6.04	1.00	1.99	0.33	1.97	30.24	20.16	182.46	364.92	60.21	120.42
64k_4w	3.02	6.04	1.00	1.99	0.33	1.97	30.24	20.16	182.46	364.92	60.21	120.42
64k_8w	3.02	6.04	1.00	1.99	0.33	1.97	30.24	20.16	182.46	364.92	60.21	120.42
8k_16w	1.70	3.40	0.56	1.12	0.18	0.37	47.93	31.96	1127.48	2254.95	372.07	744.13
8k_1w	3.02	6.04	1.00	1.99	0.33	1.32	1.03	0.69	10.76	21.51	3.55	7.10
8k_2w	3.02	6.04	1.00	1.99	0.33	1.32	1.03	0.69	10.76	21.51	3.55	7.10
8k_4w	3.02	6.04	1.00	1.99	0.33	1.32	0.51	0.34	10.89	21.77	3.59	7.18
8k_8w	3.02	6.04	1.00	1.99	0.33	1.32	27.61	18.41	163.09	326.17	53.82	107.64

The values in Table 4-1 were generated simply by inserting output hooks in eCACTI that return the transistor sizing used in the optimal solution. Since raw size numbers produced by eCACTI still assume a 0.80um process technology, these numbers are then linearly scaled down. The numbers shown in the table are for a target 0.13um process. Looking at the table, it is easy enough to see that even for medium-sized caches (e.g. 32kB), the sizing of the predecoder and the address buffers are already unrealistic. If these are converted into generic drive strengths and assuming that a 1X inverter in 0.13um technology has a width of three times

the minimum length (W1x = 3 * Lmin = 0.39um), the predecoder NAND gate for the 32k_16way configuration is already a 170X inverter, while the front-end address buffer is almost a 3000X inverter. Clearly, these are unrealistic numbers.

Moreover, CACTI and eCACTI both have a major bug in the computation of the wordline driver size that results in a significantly undersized wordline driver. Essentially, the wordline size is computed by determining the effective resistance that is needed to drive a given capacitive load while achieving a specified wordline rise time (as determined by the number of columns being driven). After a resistance is computed, both CACTI and eCACTI immediately convert the computed resistance into a transistor width. This clearly ignores the wire resistance in the computation. myCACTI fixes this bug by recognizing that the combination of the transistor effective resistance plus the interconnect resistance must be equal to the computed resistance (instead of just the transistor effective resistance by itself). This means that we first need to subtract the interconnect resistance from the desired resistance before converting this into a transistor width. This introduces a complication for cases where the interconnect resistance is already greater than the desired resistance such that it is impossible to come up with a combination of transistor effective resistance and interconnect resistance that would equal the desired resistance (since this would require that the transistor effective resistance be negative). In effect, these configurations have to be immediately flagged as invalid configurations, as it is impossible for them to meet the desired wordline risetime. Unfortunately, these cases are ignored by both CACTI and eCACTI. Figure 4-11 shows a shmoo plot where additional hooks are inserted

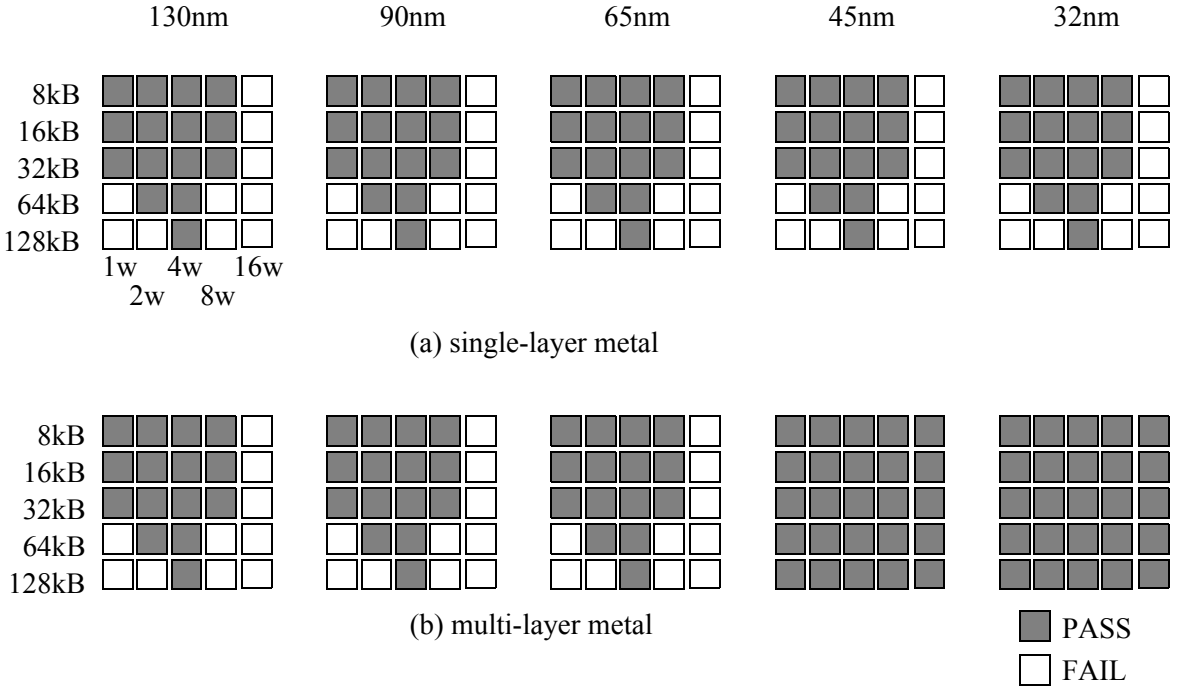


Figure 4-11: Shmoo plot of eCACTI runs. The shmoo plots show which runs that ignore the computation of wordline resistance actually are invalid because of the impossibility of meeting the resistance requirement once wordline resistance are included in the computation of the effective resistance of the wordline driver.

in myCACTI (CACTI and eCACTI are also similar) in order to expose cases where the optimal configuration being returned actually fail in the wordline sizing algorithm (which is common for all three programs). Failing points show configurations that were deemed optimal but actually should be deemed invalid because it is impossible to meet the wordline risetime requirement because the wordline interconnect resistance is already greater than the desired resistance. Points that are not shown as failing and labeled as passing should also raise flags regarding their validity -- that they are considered "passing" only means that there still existed some size of wordline driver that will result in a total resistance equal to the desired resistance. Cases where the interconnect resistance is already very close to the desired resistance would result in unrealistically large wordline drivers in order to meet the requirement. At the very least, accounting for interconnect resistance in sizing the wordline driver will always result in a larger driver than what would be computed by both CACTI and eCACTI. This exacerbates the problems of both, especially eCACTI where the increse in the wordline driver size will propagate up to the predecoder and the address buffers, causing even more unrealistic sizing.

To solve these problems, myCACTI models variable stage decoders, and dynamically decides the number of stages and the sizing of each stage based on the load that has to be driven. Doing this accomplishes two goals -- the first is that we can ensure that the proper sizing algorithms are followed for every cache configuration regardless of the current loading for that specific configuration, and the second is that we can impose the restriction that front-end gates of the decoder subblocks (i.e. the predecoder front-end and the wordline front-end) be minimum-sized such that even in the upstream gates fanout to a large number of these gates, the loading will be minimized as much as possible.

Because of the concerns that have been stated, both CACTI and eCACTI need critical overhauls to their simulation flow. Consequently, instead of comparing myCACTI to largely unmodified versions of CACTI and eCACTI, myCACTI was written as a superset of all the integral features of CACTI, eCACTI, and then includes some additional features that are useful in cache design. For the comparisons, we can now run myCACTI with one or more of these features turned off in order to replicate CACTI and/or eCACTI behavior. This is strictly not an "apples-to-apples" comparison, but is a reasonable approximation that still gives useful, practical results given the limitations of CACTI and eCACTI that have been discussed.

The default operation of myCACTI is shown in Table 4-2, and most of the comparisons enable or disable one of these features in order to compare myCACTI to CACTI or eCACTI-like operation.

Table 4-2: Operational parameters for myCACTI, eCACTI and CACTI

	myCACTI default	"eCACTI" default	"CACTI" default
Linear scaling of results	disabled[a]	enabled	enabled

Table 4-2: Operational parameters for myCACTI, eCACTI and CACTI

	myCACTI default	"eCACTI" default	"CACTI" default
Transistor effective length	Leff = Ldrawn - 2*LINT	Leff = Ldrawn	Leff = Ldrawn
Via parasitic capacitance	enabled	disabled	disabled
Multi-level metal	enabled	disabled	disabled
Explicit pipelining	enabled	disabled	disabled
Gate leakage	enabled	disabled	disabled
Decoding implementation	dynamic	static	static
Transistor Vt	dualVt	single/ dualVt	single
Sensing scheme	low-swing differential	low-swing differential	low-swing differential
ILD dielectric constant	high-K	high-K	high-K
Number of decode stages[b]	variable	variable	variable
WL wire resistance[c]	enabled	enabled	enabled

a. results are computed on a per process basis
b. Note that these are enabled for all three since fixed stage decoding does not make sense during a design space exploration
c. Again, these are enabled for all three since this is essentially a bug fix.

4.4 Comparison Results Summary

Table 4-3 provides a summary of the differences when modeling and not modeling some of the characteristics of caches.

Table 4-3: Delay and power difference for simulations that model or do not model the given implementation parameter.

	Delay Difference	Power Difference
Results scaling	10 - 150ps	15% - 150%
Transistor effective length	up to 15ps	5% - 18%

Table 4-3: Delay and power difference for simulations that model or do not model the given implementation parameter.

	Delay Difference	Power Difference
Via Capacitances	up to 50ps	3%
Explicit pipelining	N/A	20% - 80%
Multiple metal layers	typically up to 300ps (w/ extremes at 800ps)	1% - 5%
Gate leakage	N/A	1%
Combination of parameters[a]	50ps - 200ps (extremes at 500ps)	up to 20%

a. These is a comparison between simulations that account for all or nothing in the parameters listed in the table *except* for pipelining, where both implementations model explicit pipelining.

4.5 Detailed Comparison Results

4.5.1 Validity of CACTI/eCACTI scaling

Run details

To produce these data, myCACTI was made to do a design space exploration using the default settings of myCACTI to find the optimal implementation for every cache size and associativity for the 90nm technology node. After determining the optimal implementations for each point in the 90nm design space (in terms of the cache parameter sextet Ndwl:Nspd:Ndbl-Ntwl:Ntspd:Ntbl), three more sets of runs are performed for each technology node, with each point in the design space utilizing the optimal implementation that was determined for the 90nm node. In essence, the data for 65nm, 45nm and 32nm constitute a sort of "process-shrink" where we take a design in an older process technology and convert it to a newer one. Although the runs for 65nm and below will not show the absolute optimal implementation possible, this method was chosen to provide a simple but valid comparison between different technology nodes. Implementing the same cache configuration (i.e. same cache size and associativity) using two different implementations for different technology nodes would make the comparisons between different cache components very difficult and generalizations will be hard to make.

After generating a complete set of data using myCACTI default settings, we repeat the first step by generating numbers for the 90nm node, and then replicate the CACTI and eCACTI method of simply scaling down the numbers linearly to the target technology node.

Run timing result

Figure 4-12 shows the delay results for all cache configurations for both runs where results are generated through per-process simulations (the myCACTI default, shown in the plots as the solid lines), and through simple linear scaling of results from a 0.13um process (shown as dashed lines). This figure shows the resulting optimal clock period for each configuration, along with the pipe delays for all of the stages in the cache (with the exception of pipe1A_data and pipe1B_data -- the initial decode stages -- since these always have significant slack so they have been omitted from the figure for clarity). Figures 4-14 to Figure 4-18 show the delay component for each individual stage. These individual plots provide a breakdown of the delay of each individual stage by showing the component delays of each. Lastly, Figure 4-20 shows the unpipelined cache access time

The comparison shown in the figure shows very significant differences in access time for many of the configurations. For configurations that don't have the compare as the critical stage (pipe2A_tag), the access time difference is roughly 10ps to 30ps, while configurations that do have the compare stage as their critical path have roughly 60ps to 150ps difference! Detailed experiments on the cause of this difference shows that the compare path can be optimized further by customizing the size and upsizing the transistors even more. But although the compare stage can be optimized further, it must be emphasized that it has been sized to have the

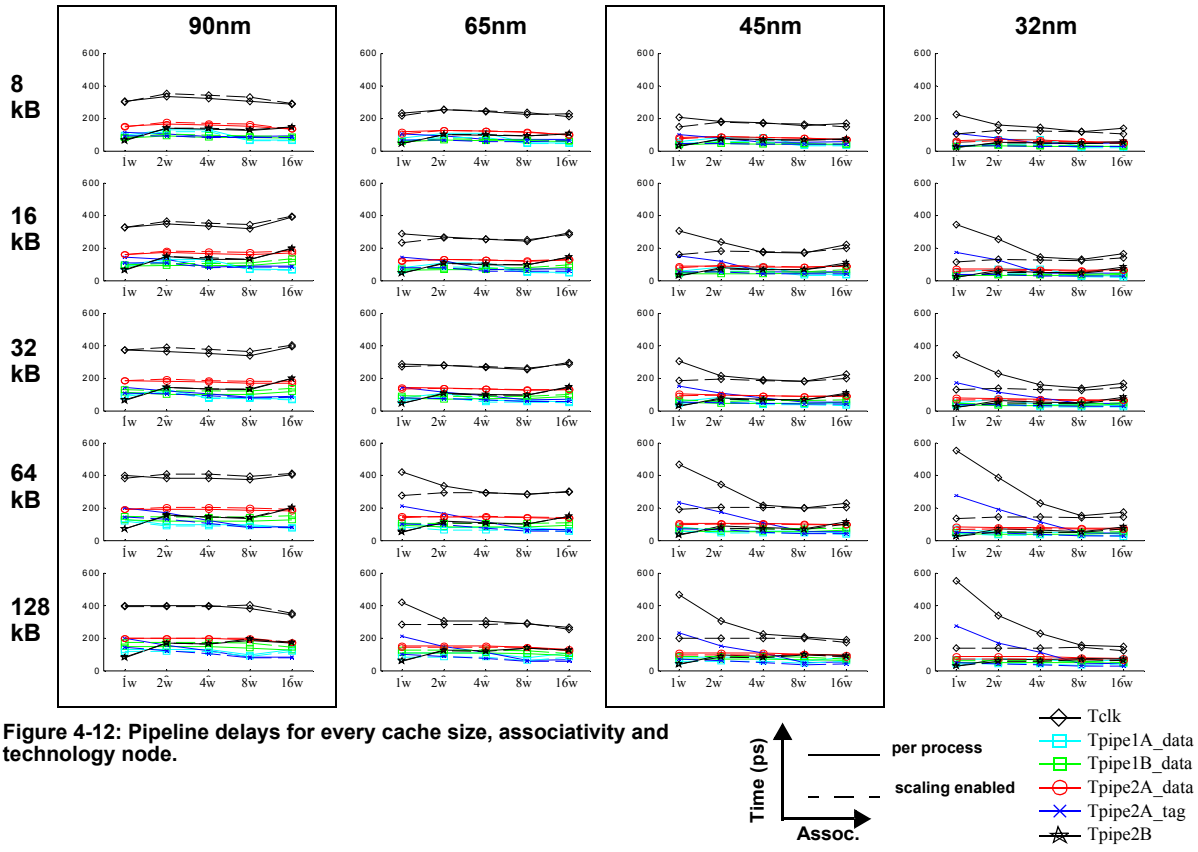


Figure 4-12: Pipeline delays for every cache size, associativity and technology node.

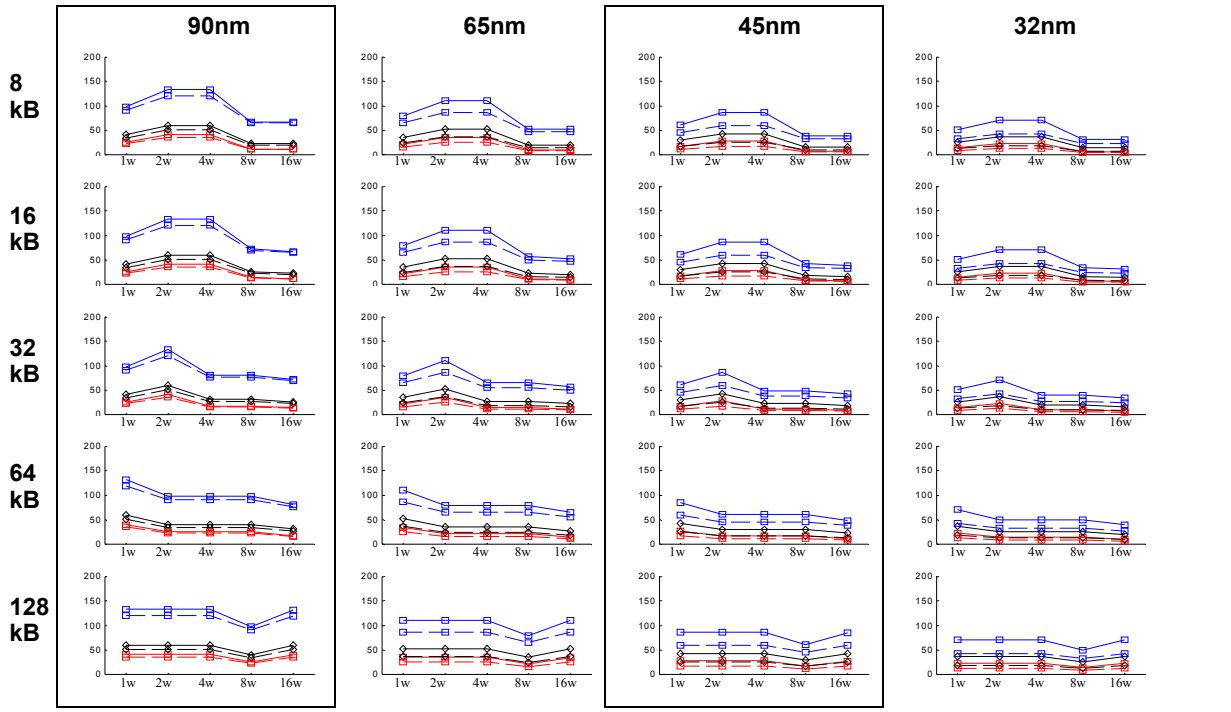


Figure 4-13: Delay components for pipeline stage pipe1A_data . This pipeline stage consists of data address buffering and the predecoder front-end. The solid-lines denote default myCACTI numbers, while dashed lines denote runs that use extrapolation from results for a 0.13um process.

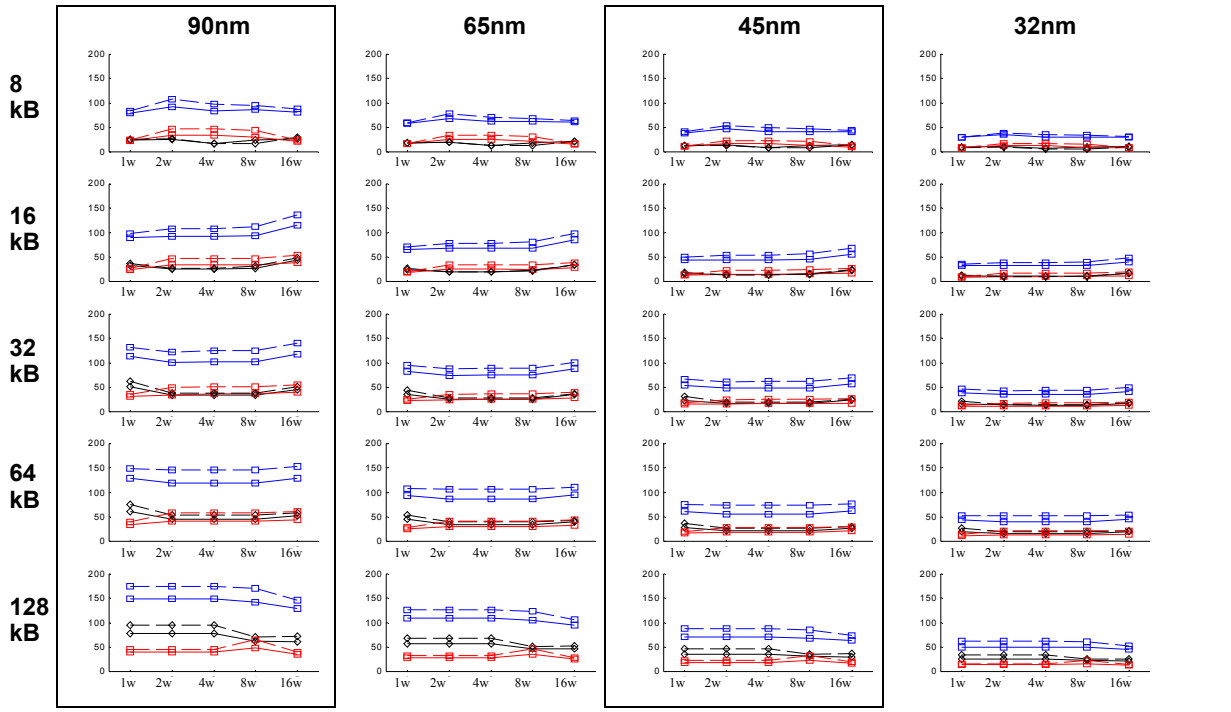


Figure 4-14: Delay components for pipeline stage pipe1B_data . This pipeline stage consists of the data predecoder driver and the wordline front-end. The solid-lines denote default myCACTI numbers, while dashed lines denote runs that use extrapolation from results for a 0.13um process.

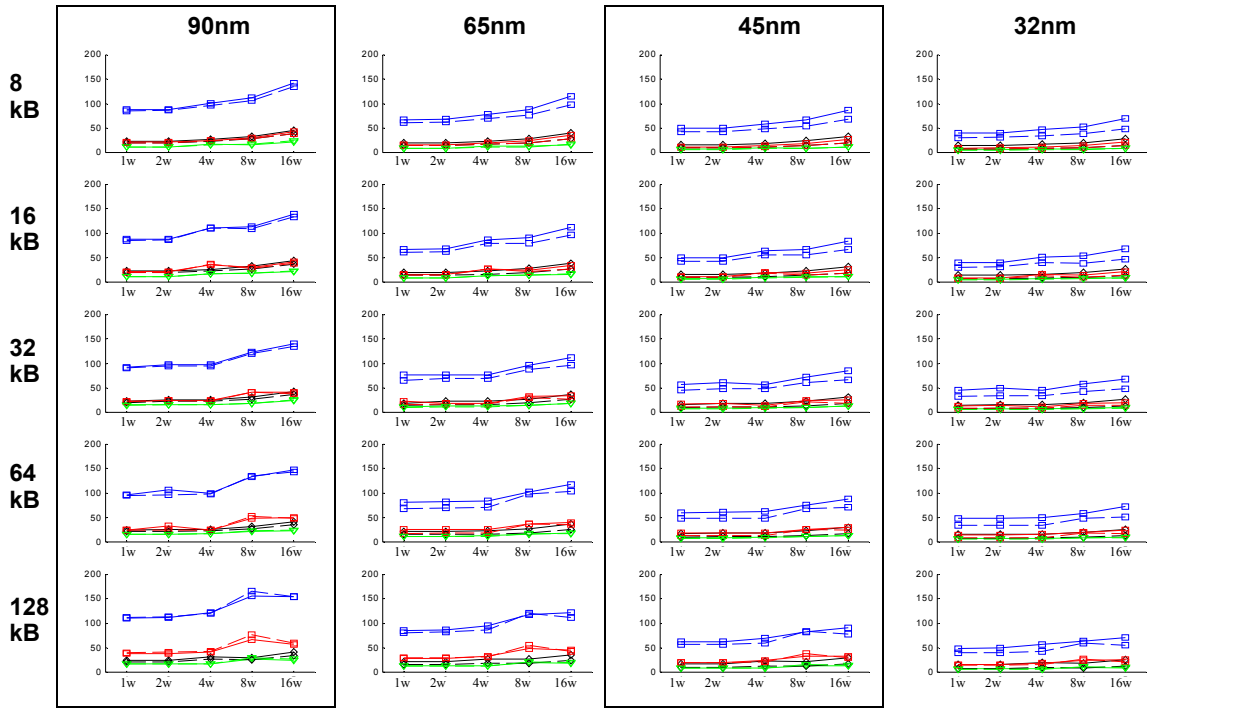


Figure 4-15: Delay components for pipeline stage pipe2A_data . This pipeline stage consists of the wordline drivers, the bitline and the sense amplifier. The solid-lines denote default myCACTI numbers, while dashed lines denote runs that use extrapolation from results for a 0.13um process.

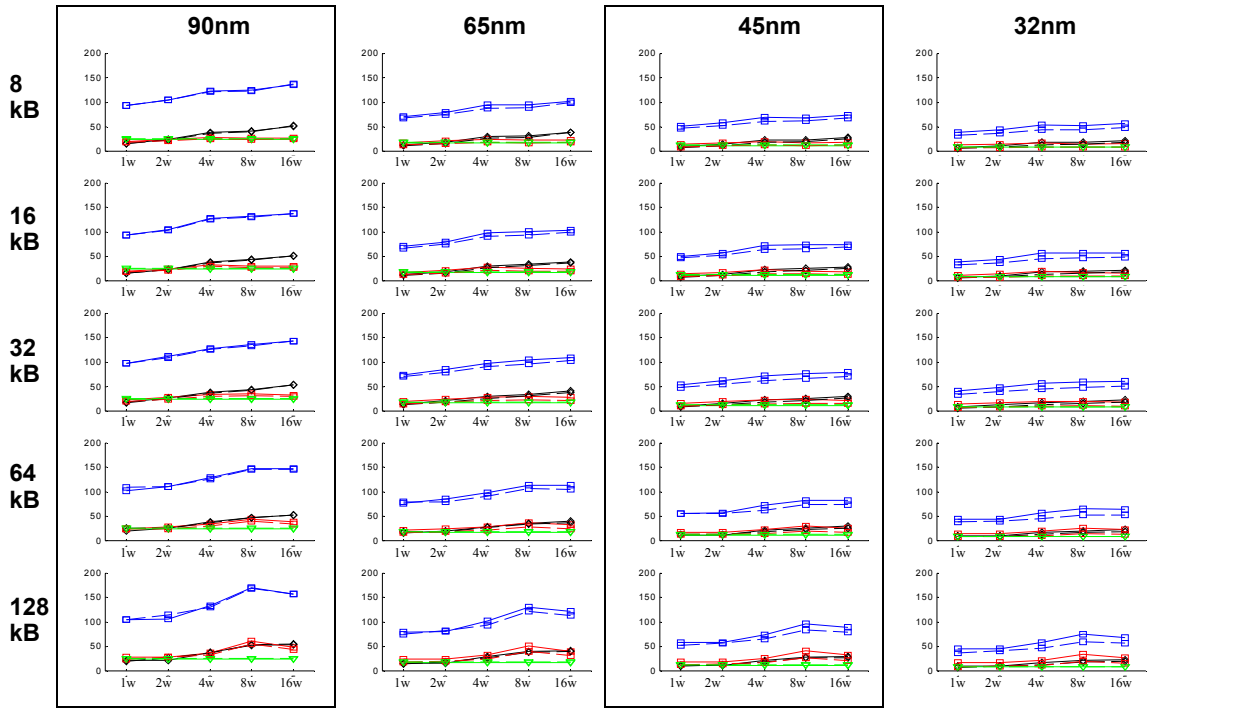


Figure 4-16: Delay components for pipeline stage pipe1A_tag. This pipeline stage consists of tag address buffering, predecoding and the wordline front-end. The solid-lines denote default myCACTI numbers, while dashed lines denote runs that use extrapolation from results for a 0.13um process.

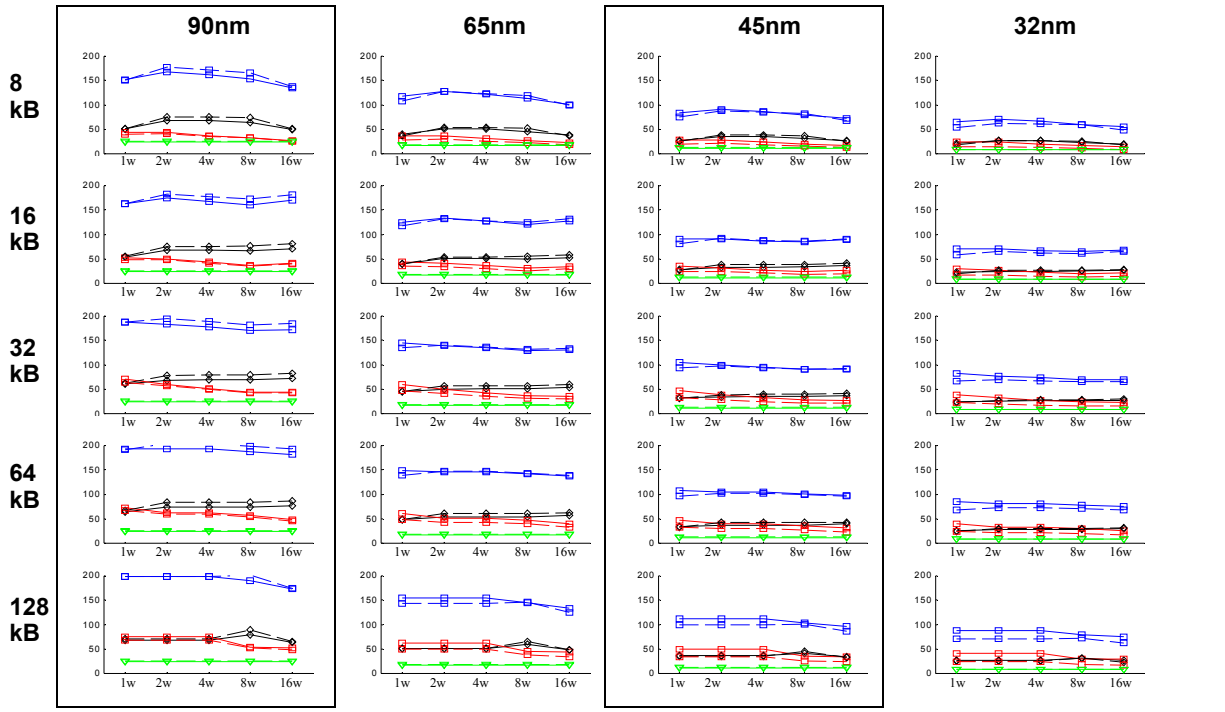


Figure 4-17: Delay components for pipeline stage pipe1B_tag . This pipeline stage consists of the tag wordline drviers, bitline delay and senseamp delay. The solid-lines denote default myCACTI numbers, while dashed lines denote runs that use extrapolation from results for a 0.13um process.

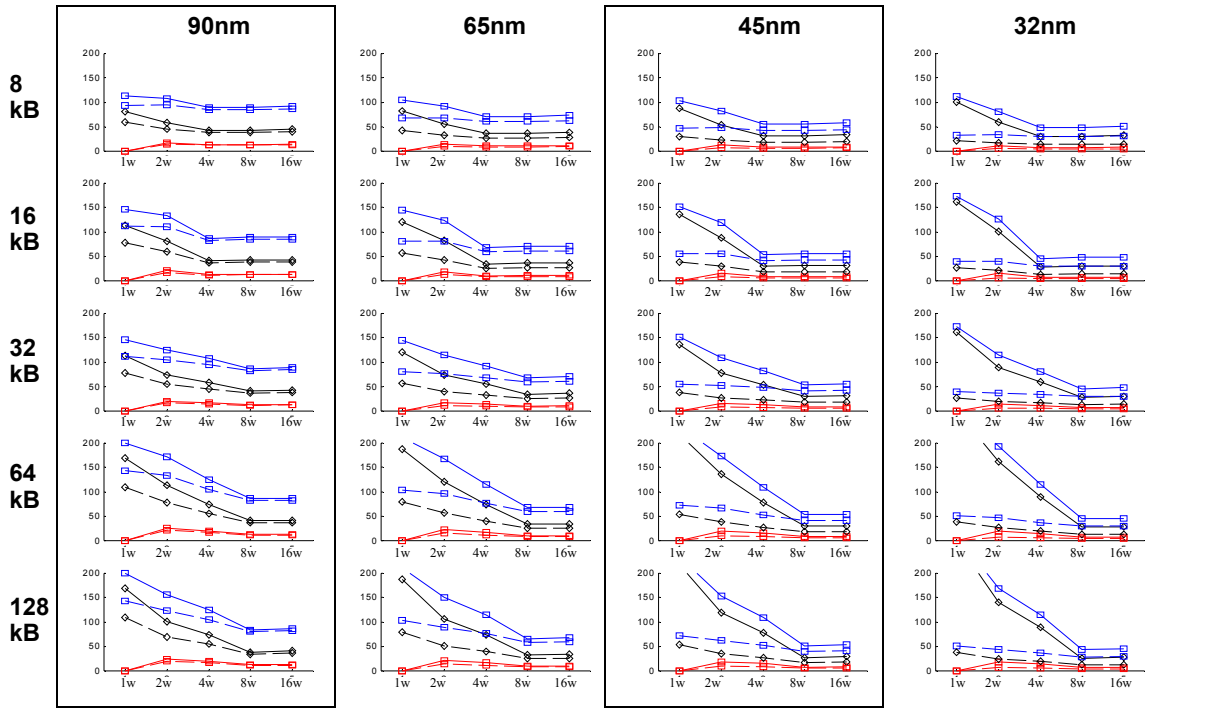


Figure 4-18: Delay components for pipeline stage pipe2A_tag . This pipeline stage consists of enabling the comparator and the actual comparator delay. The solid-lines denote default myCACTI numbers, while dashed lines denote runs that use extrapolation from results for a 0.13um process.

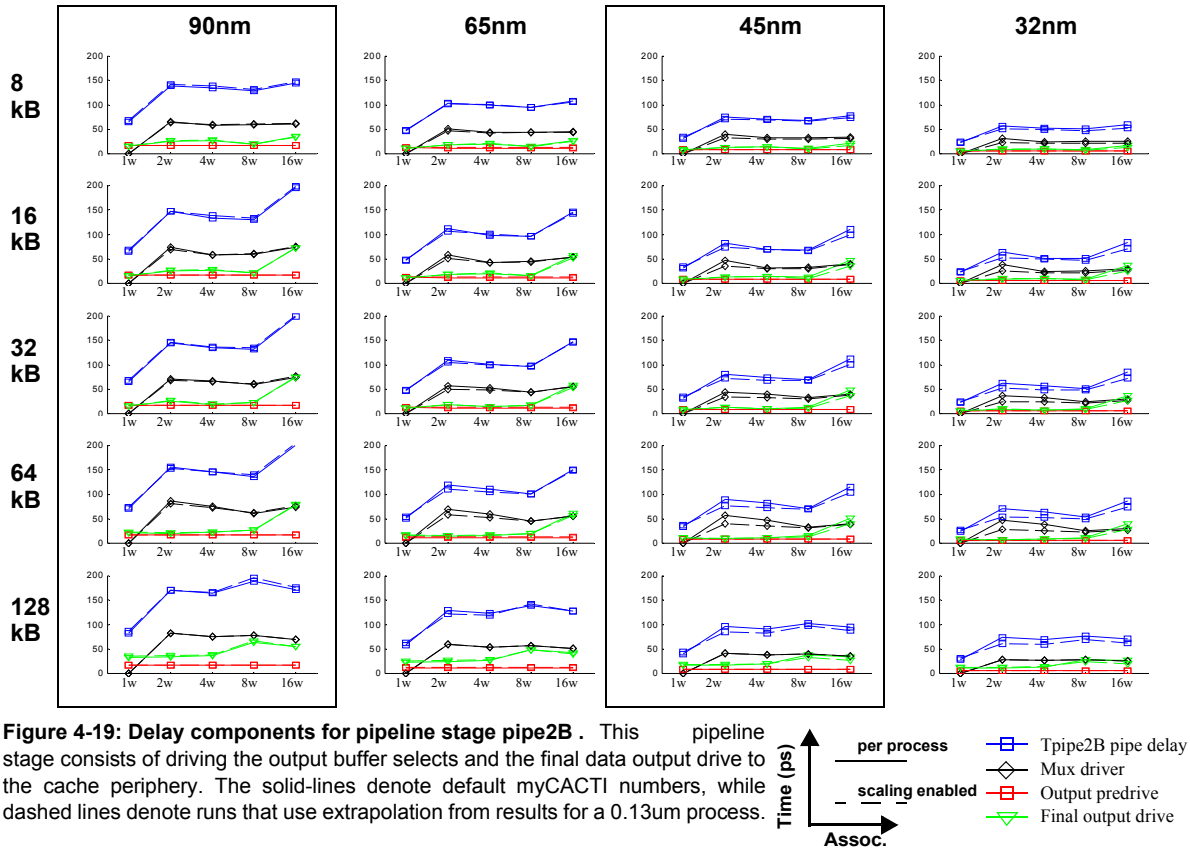


Figure 4-19: Delay components for pipeline stage pipe2B . This pipeline stage consists of driving the output buffer selects and the final data output drive to the cache periphery. The solid-lines denote default myCACTI numbers, while dashed lines denote runs that use extrapolation from results for a 0.13um process.

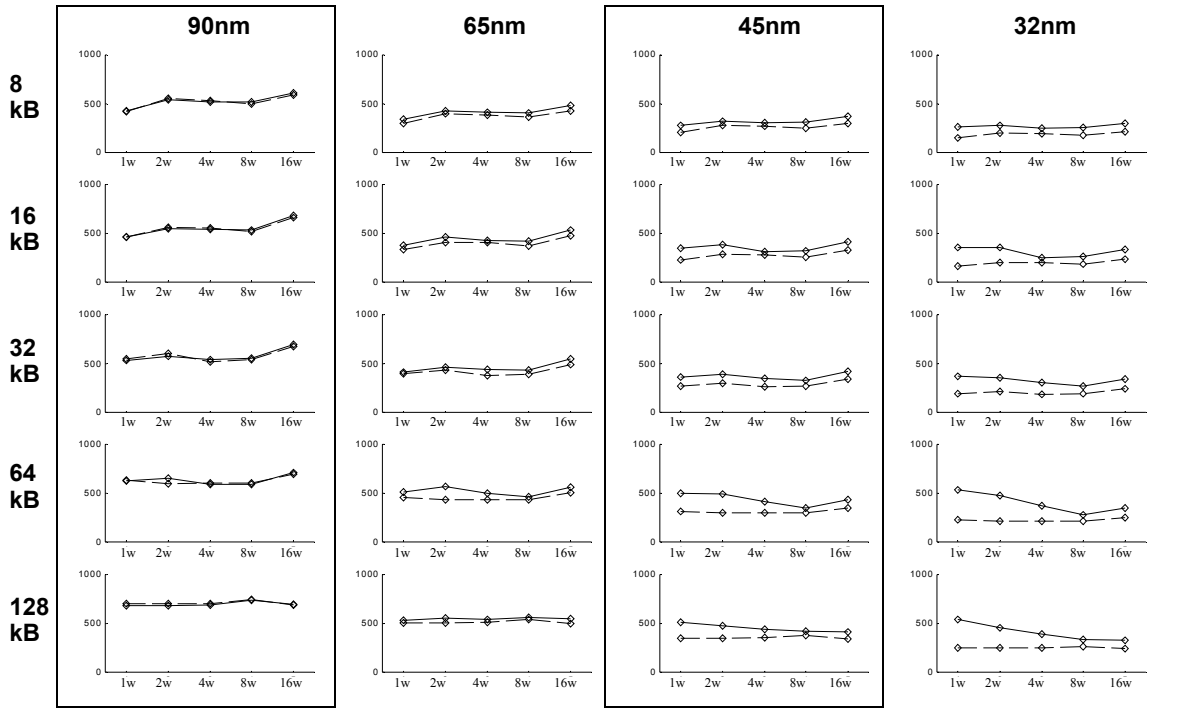


Figure 4-20: Unpipelined cycle time. TThis shows the unpipelined cycle time for the cache. The cycle time is typically greater than the access time as it accounts for the need to reset the devices inside the cache for the next access. The solid-lines denote default myCACTI numbers, while dashed lines denote runs that use extrapolation from results for a 0.13um process.

same drive strength as the original 0.8um circuit. This shows that for this particular circuit, linear scaling of transistor device sizes yield suboptimal delay performance.

An additional problem that can be seen is that scaling from 130nm to 90nm overestimates the cache delay, while scaling to 65nm/45nm/32nm tech nodes underestimates it. This significantly increases the variability of the difference between per-process simulation and simplistically scaling results linearly based on older processes.

Looking at some of the individual pipe stages, we have the following observations:

- The data address buffer and predecoder front-end delays are underestimated when using results scaling, and the difference with explicit simulation becomes larger in the deeper nanometer nodes.
- The data predecoder drive and wordline frontends are overestimated when scaling, with the differences becoming smaller in the deeper nanometer nodes.
- The data wordline driver is underestimated, while bitline delay is erratic, with the differences becoming bigger in the deeper nanometer nodes. This holds true also for the tag array.
- The compare stage is drastically different. The delay is significantly underestimated when using results scaling. This has been alluded to earlier, and is caused by the suboptimal sizing of the compare circuit even though they have the same aspect ratios as the reference technology. This shows that transistor performance has not kept up with technology. One explanation is that the recent motivations in transistor fabrication has also given tremendous importance to transistor power dissipation such that some speed performance may have been sacrificed.
- In the data output driver, the mux drive delay is underestimated, while the final data output driving is overestimated, with the differences becoming smaller in the deeper nanometer nodes.

Run power results

Figure 4-21 shows the total power and its breakdown into dynamic power (switching power) and static power (dissipated by subthreshold leakage and gate leakage currents) for every point in the design space that was considered (i.e. cache size, associativity and technology node). Each plot shows both the results from myCACTI runs that use explicit per-process simulation and runs that linearly scale the results from 0.13um process runs. Figure 4-22 demonstrates the differences more explicitly by plotting the fractional difference between the two runs normalized to the total power of the default myCACTI result. Figure 4-23 shows detailed power breakdowns of some representative design points in order to explore in detail the causes of the differences. In addition, the results from the two runs are placed side by side for easy comparison. Lastly,

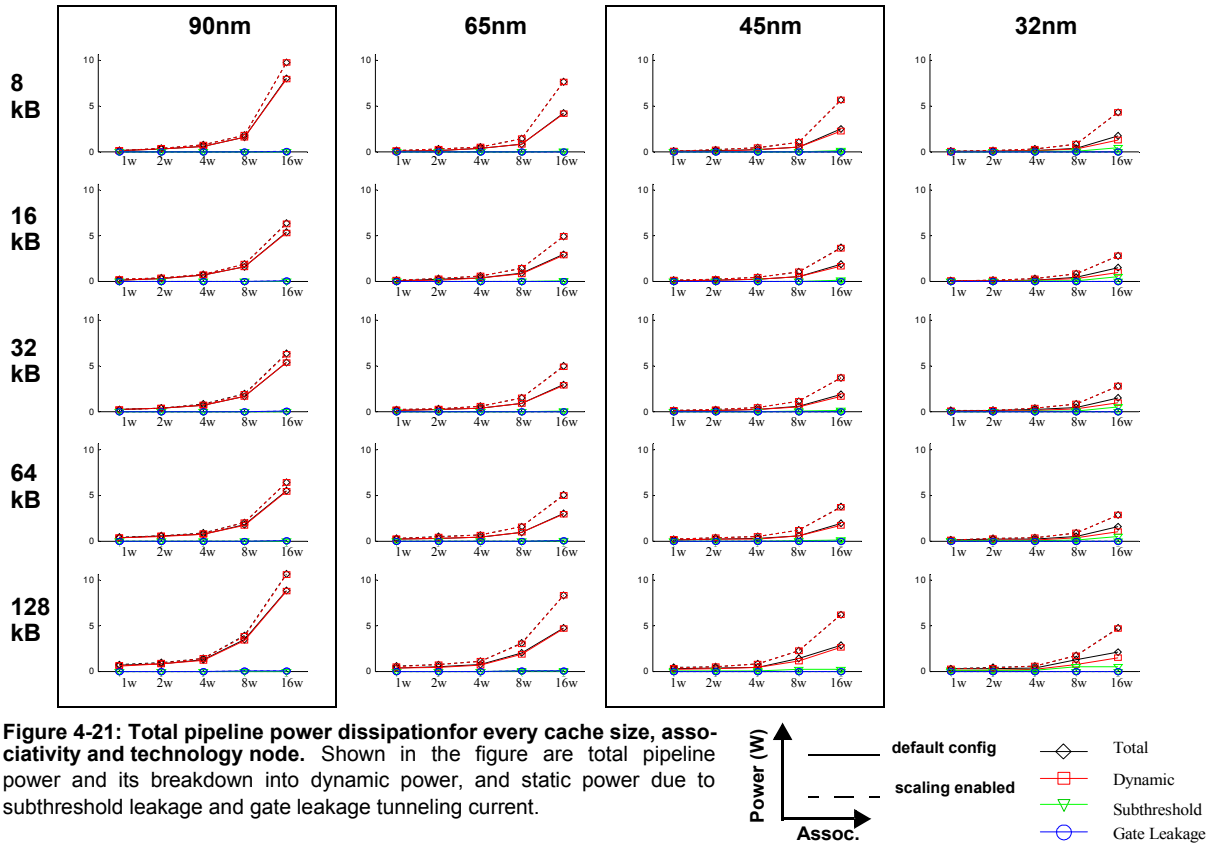


Figure 4-21: Total pipeline power dissipationfor every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

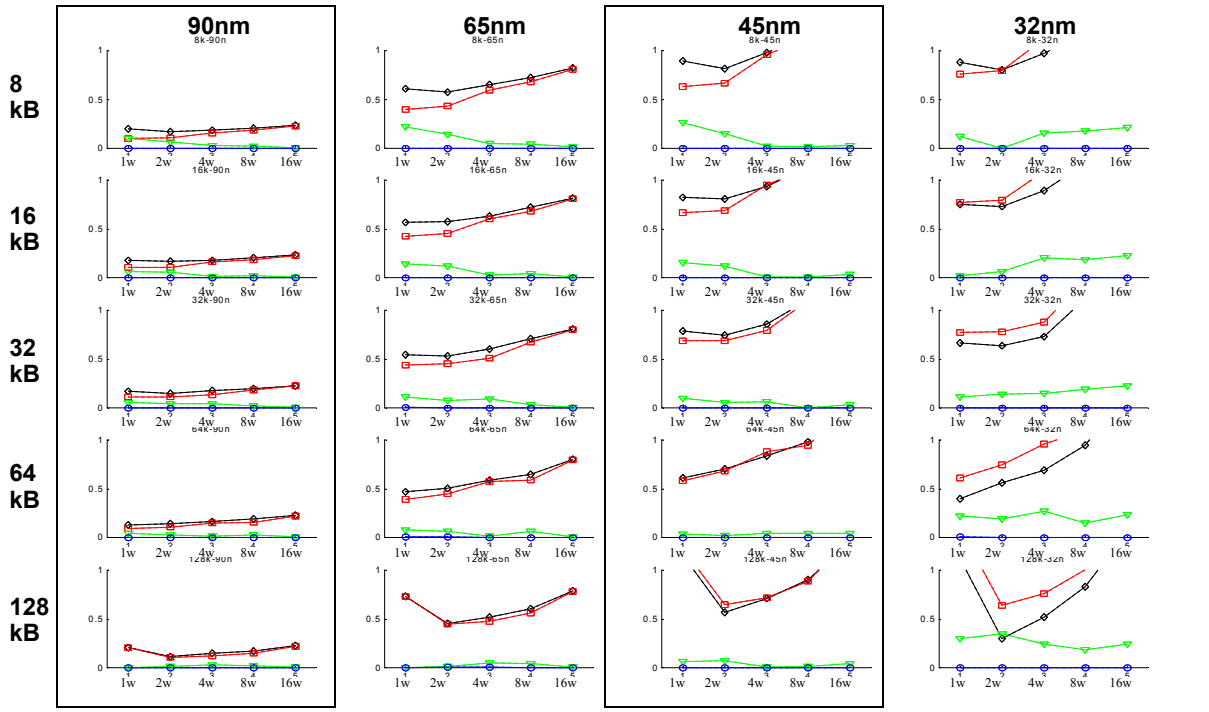


Figure 4-22: Pipeline error difference power dissipation for every cache size, associativity and technology node. Shown in the figure are the error difference for pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

Figure 4-24 identifies how much power is being dissipated by major cache blocks, and again shows the results from the two runs side by side for easy comparison.

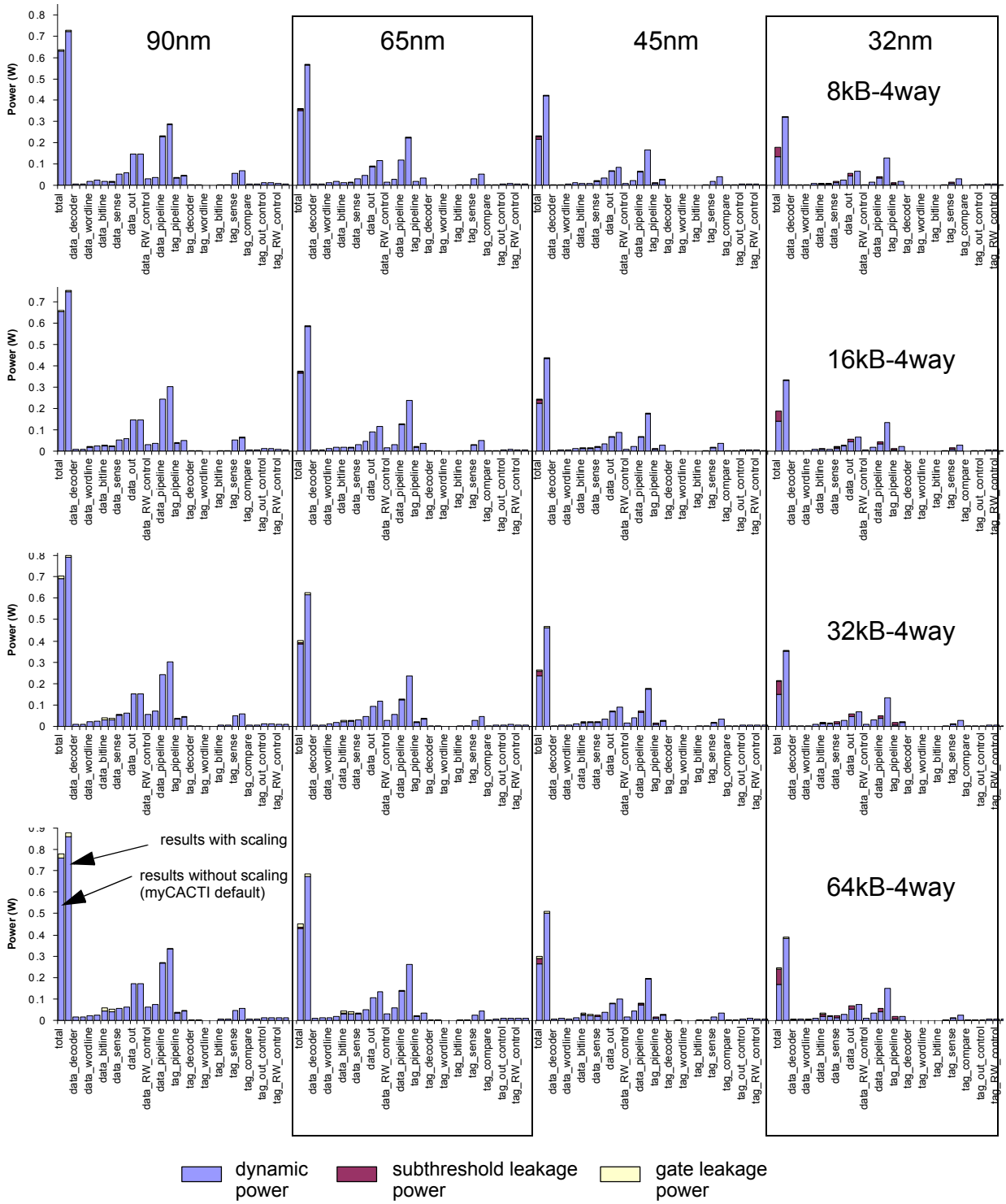


Figure 4-23: Detailed cache power breakdown comparison. Detailed power breakdown for four different cache configurations showing the amount of dynamic, subthreshold leakage and gate leakage power being consumed by different parts of the cache. Each plot shows four sets of values corresponding to different technology nodes. (Note that all three plots use the same scale). The two results are also placed side by side for direct comparison.

From these figures, we can easily see that simple scaling of results using an older process technology instead of explicitly simulating the target process yields significant differences in power. Scaling the results, in many cases, significantly overestimates the power, with the difference becoming larger in the smaller technology nodes. For example, the difference in total power is from 15% to 20% in 90nm. The difference is even much larger in 32nm, with power numbers being overestimated by about 10% to 150%.

Looking at the power breakdowns, one big difference is the very small subthreshold leakage power in the scaled version. This makes sense since the reference technology (0.13um) has, expectedly, a very small subthreshold leakage current, and assuming a linear scaling of this leakage current will result in a huge

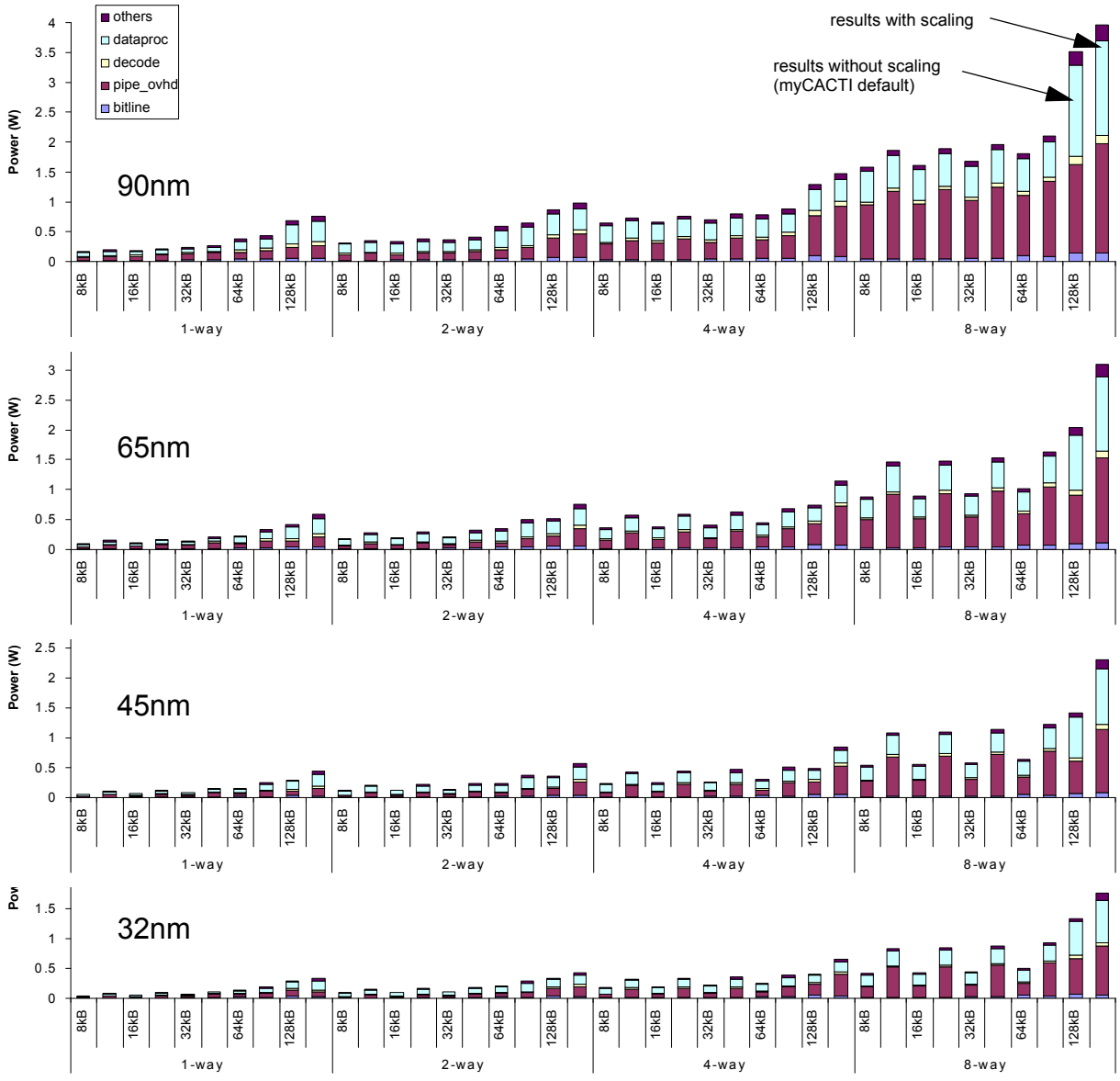


Figure 4-24: Power contributions of different cache parts. Power breakdown showing major cache power contributors for the 65nm and 32nm technology nodes for different cache sizes and associativities. (Note that the y-axis for both plots use the same scale). The two results are also placed side by side for direct comparison.

discrepancy, as the leakage current actually experiences exponential scaling. In addition, the pipeline overhead power difference is very significant, and scaling the results severely overestimates the power value and doesn't account for the possibility that the dynamic power can be reduced superlinearly.

Conclusion

Our results have shown that the simple linear scaling of results from an older reference technology node (in our case, 0.13um), in order to produce results for smaller tech nodes produces significant differences compared to explicitly modeling the target tech node.

We see that both cache delay and power have significant differences. Making it worse is that scaling may overestimate results for one configuration, while underestimating results for another. This makes the variability of the result even bigger and makes the results provided by linear scaling even more unreliable. Since explicitly modeling the target process does not provide a burden to the user (it only makes it more difficult to write the program in order to properly support multiple process technologies instead of just a single one), it is imperative that cache design tools always explicitly simulate the target process instead of doing a linear scaling of the results. This is even more important for deep nanometer technologies, as current trends and design practices that prevail for these technologies may not necessarily be captured by models of older process technologies. An obvious example is the big discrepancy in subthreshold leakage values, as linear scaling doesn't capture the exponential increase of subthreshold leakage with each new generation. A more subtle example is the possible non-scaling of device speeds as process technologies may have implemented techniques that sacrifice speed of operation in exchange for better reliability or lower power.

Finally, it must be emphasized that both CACTI and eCACTI[1] perform linear scaling of results that are based on a 0.80um technology instead of the 0.13um technology that we have used in the comparison. This means that the parameters in this simulation that assumes long-channel transistors will be much more different compared to nanometer transistor behavior as opposed to the submicron technology (0.13um) that we used as the scaling reference in this subsection.

1. An exception exists for eCACTI for subthreshold leakage computation, which it does on a per-process basis instead of using linear scaling.

4.5.2 Transistor effective length

Run details

To produce these data, myCACTI was made to do a design space exploration using the default settings of myCACTI (in particular, using Leff = Ldrawn - 2*LINT) to find the optimal implementation for every cache size and associativity for the 90nm technology node. After determining the optimal implementations for each point in the 90nm design space (in terms of the cache parameter sextet Ndwl:Nspd:Ndbl-Ntwl:Ntspd:Ntbl), three more sets of runs are performed for each technology node, with each point in the design space utilizing the optimal implementation that was determined for the 90nm node. In essence, the data for 65nm, 45nm and 32nm constitute a sort of "process-shrink" where we take a design in an older process technology and convert it to a newer one. Although the runs for 65nm and below will not show the absolute optimal implementation possible, this method was chosen to provide a simple but valid comparison between different technology nodes. Implementing the same cache configuration (i.e. same cache size and associativity) using two different implementations for different technology nodes would make the comparisons between different cache components very difficult and generalizations will be hard to make.

After generating a complete set of data using myCACTI default settings, the process is repeated, this time with each myCACTI run being set to use Leff = Ldrawn (by using the -use_Ldrawn option).

Run timing results

Figure 4-25 shows the delay results for all cache configurations for both runs where results are generated using the two different techniques of computing the transistor effective length. This figure shows the resulting optimal clock period for each configuration, along with the pipe delays for all of the stages in the cache (with the exception of pipe1A_data and pipe1B_data -- the initial decode stages -- since these always have significant slack so they have been omitted from the figure for clarity). Figures 4-26 to 4-30 show the delay component for each individual stage. These individual plots provide a breakdown of the delay of each individual stage by showing the component delays of each. Lastly, Figure 4-33 shows the unpipelined cache access time.

The comparisons shown in the different figure show not only measurable difference in the total pipeline clock period, but also significant differences in delays for all pipe stages, not just a select few. This observation makes sense as accounting for the drain/source junction overlap while computing for the effective transistor length will typically make the effective length shorter than the drawn length (i.e. Leff < Ldrawn) such that the current capability and hence the driving strength of all transistors are effectively increased. Again, it must be emphasized that all the transistors in the cache benefit from this way of modeling the transistor length, regardless of the width. Going back to a transistor's current equation, we see that the current

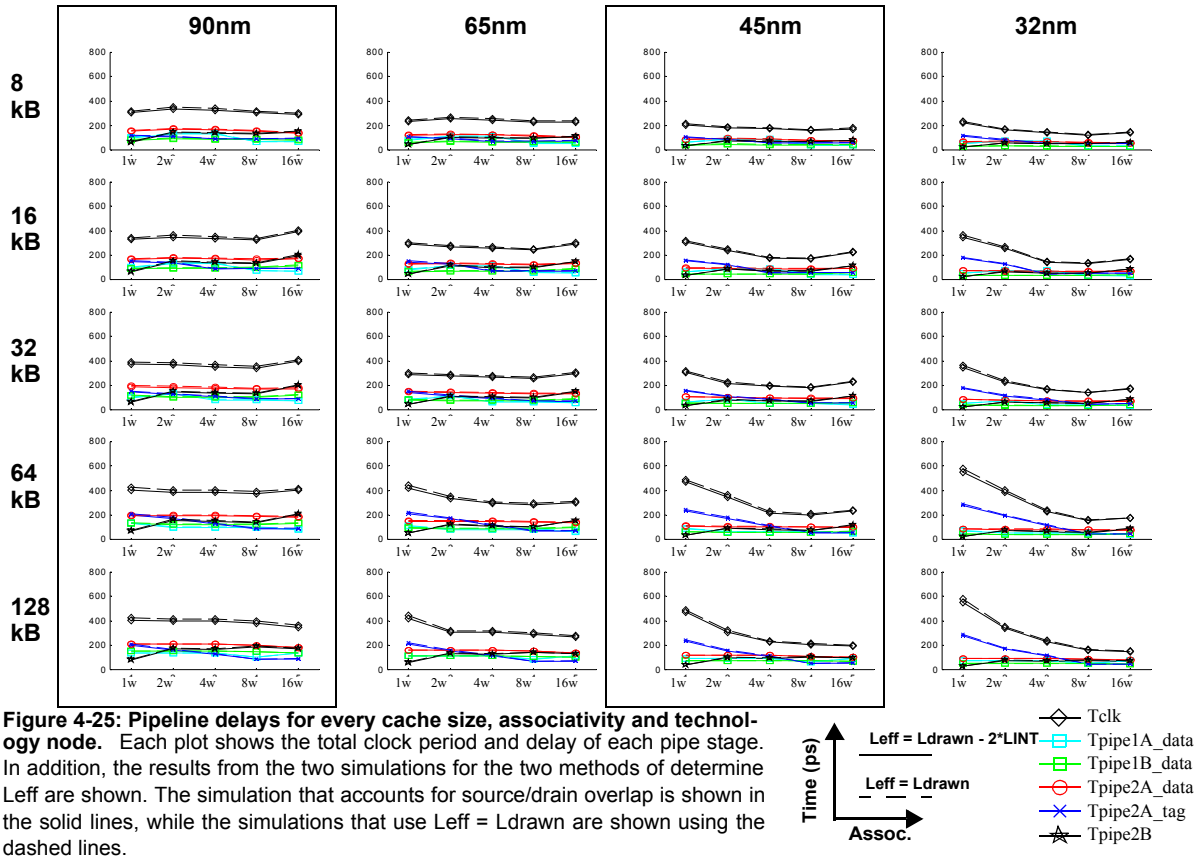


Figure 4-25: Pipeline delays for every cache size, associativity and technology node. Each plot shows the total clock period and delay of each pipe stage. In addition, the results from the two simulations for the two methods of determine Leff are shown. The simulation that accounts for source/drain overlap is shown in the solid lines, while the simulations that use Leff = Ldrawn are shown using the dashed lines.

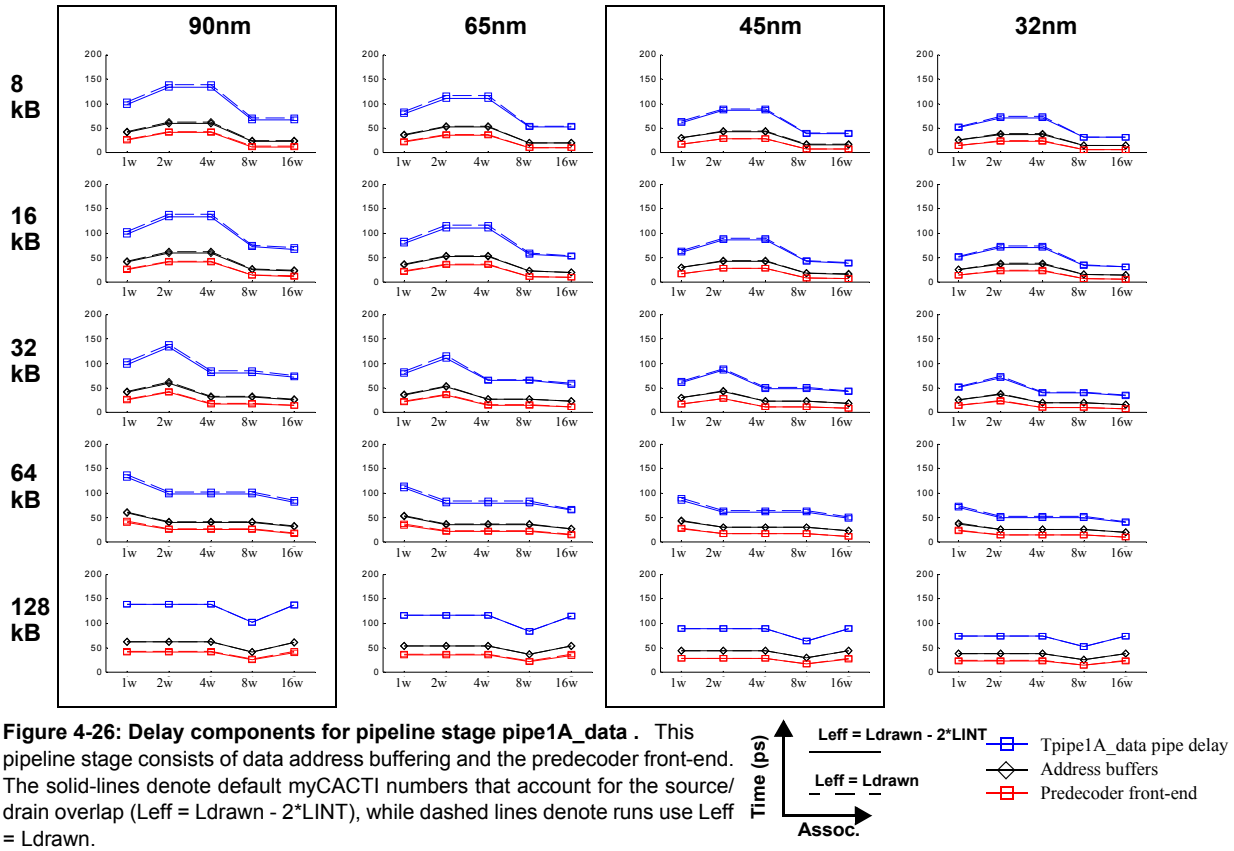


Figure 4-26: Delay components for pipeline stage pipe1A_data . This pipeline stage consists of data address buffering and the predecoder front-end. The solid-lines denote default myCACTI numbers that account for the source/drain overlap (Leff = Ldrawn - 2*LINT), while dashed lines denote runs use Leff = Ldrawn.

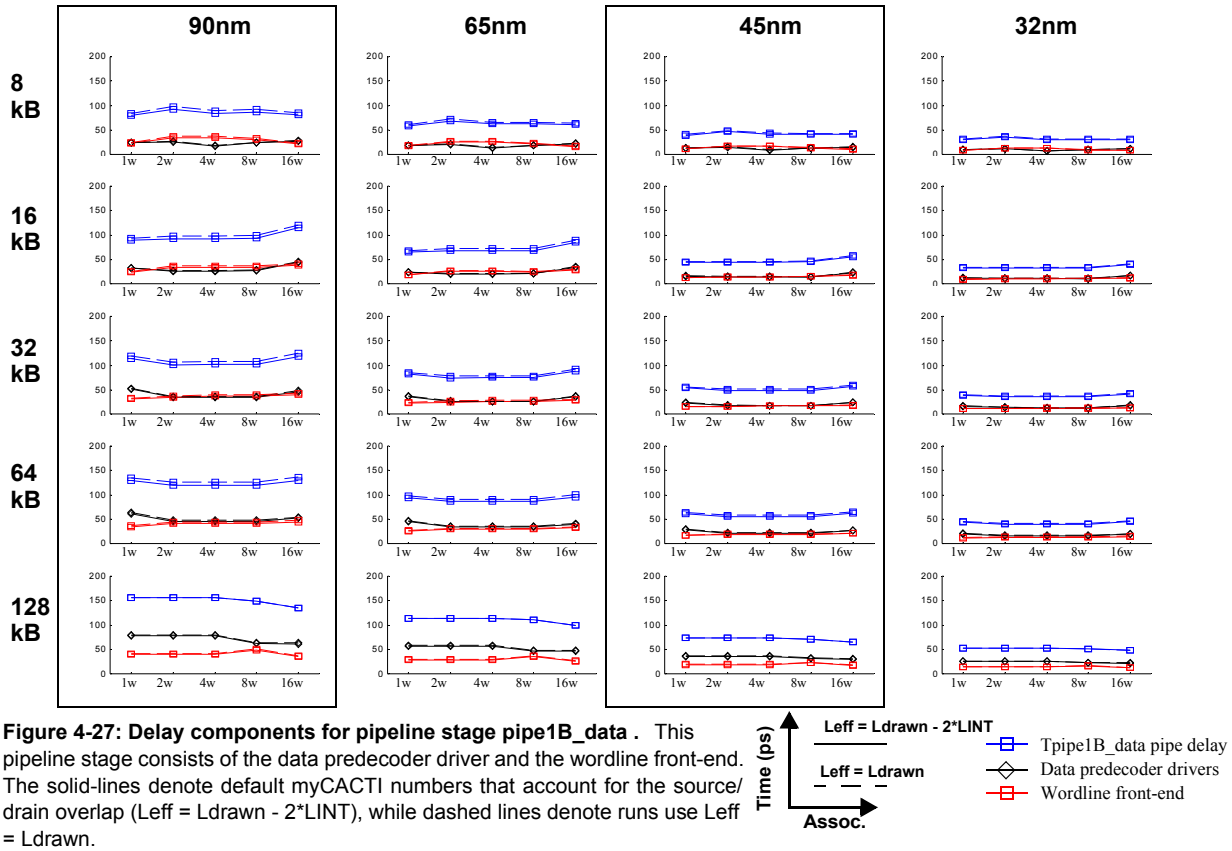


Figure 4-27: Delay components for pipeline stage pipe1B_data . This pipeline stage consists of the data predecoder driver and the wordline front-end. The solid-lines denote default myCACTI numbers that account for the source/drain overlap (Leff = Ldrawn - 2*LINT), while dashed lines denote runs use Leff = Ldrawn.

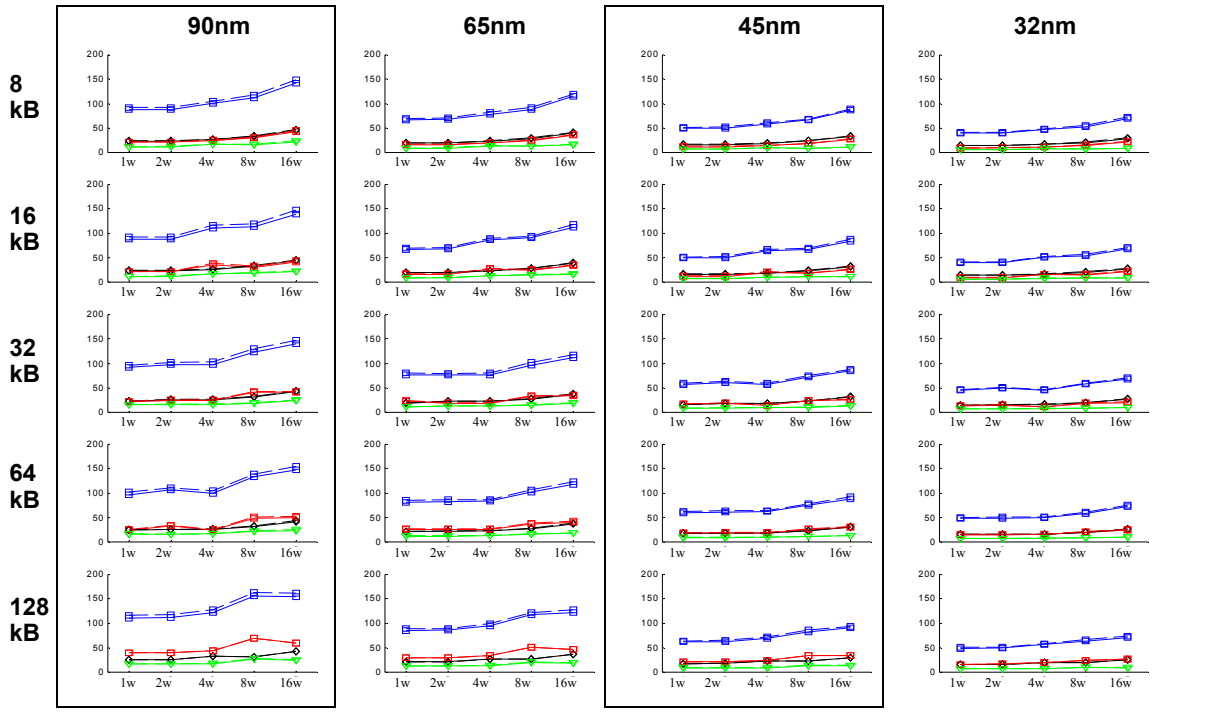


Figure 4-28: Delay components for pipeline stage pipe2A_data . This pipeline stage consists of the wordline drivers, the bitline and the sense amplifier. The solid-lines denote default myCACTI numbers that account for the source/drain overlap (Leff = Ldrawn - 2*LINT), while dashed lines denote runs use Leff = Ldrawn.

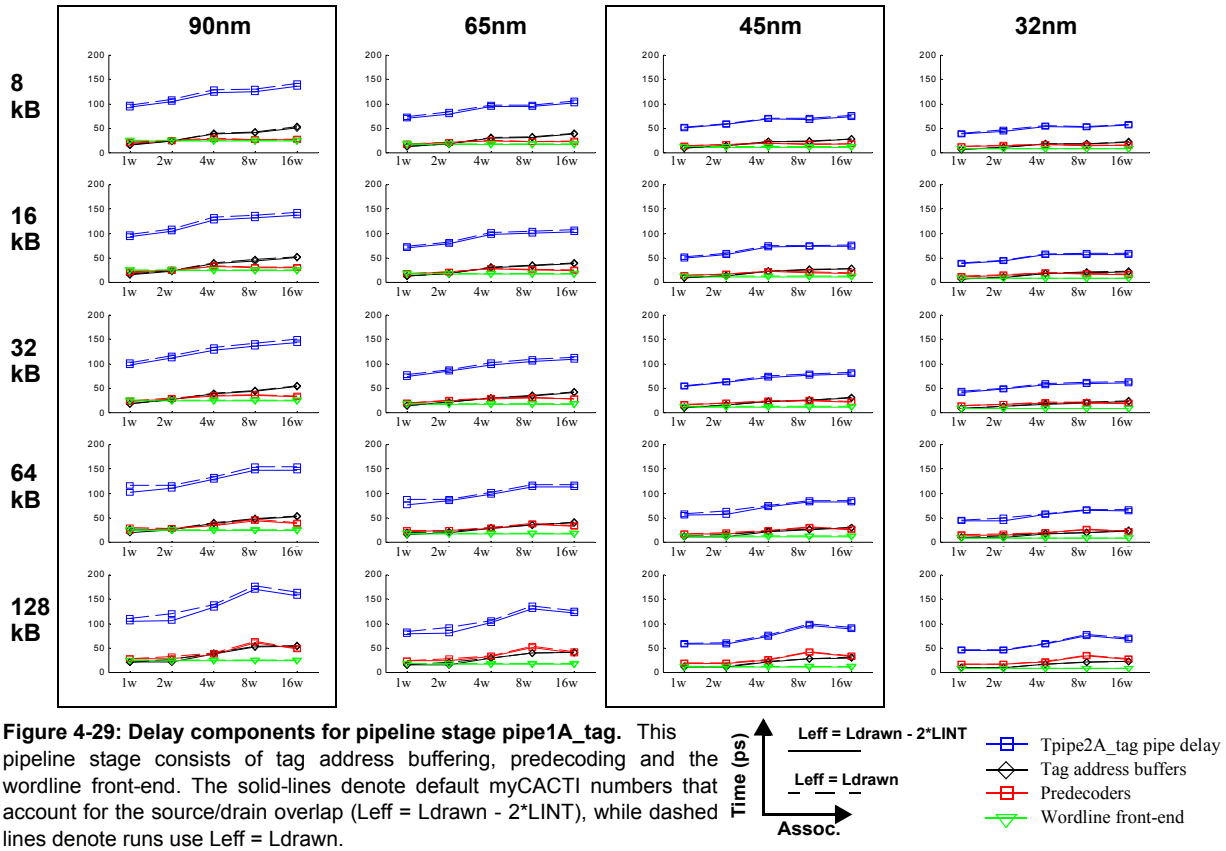


Figure 4-29: Delay components for pipeline stage pipe1A_tag. This pipeline stage consists of tag address buffering, predecoding and the wordline front-end. The solid-lines denote default myCACTI numbers that account for the source/drain overlap (Leff = Ldrawn - 2*LINT), while dashed lines denote runs use Leff = Ldrawn.

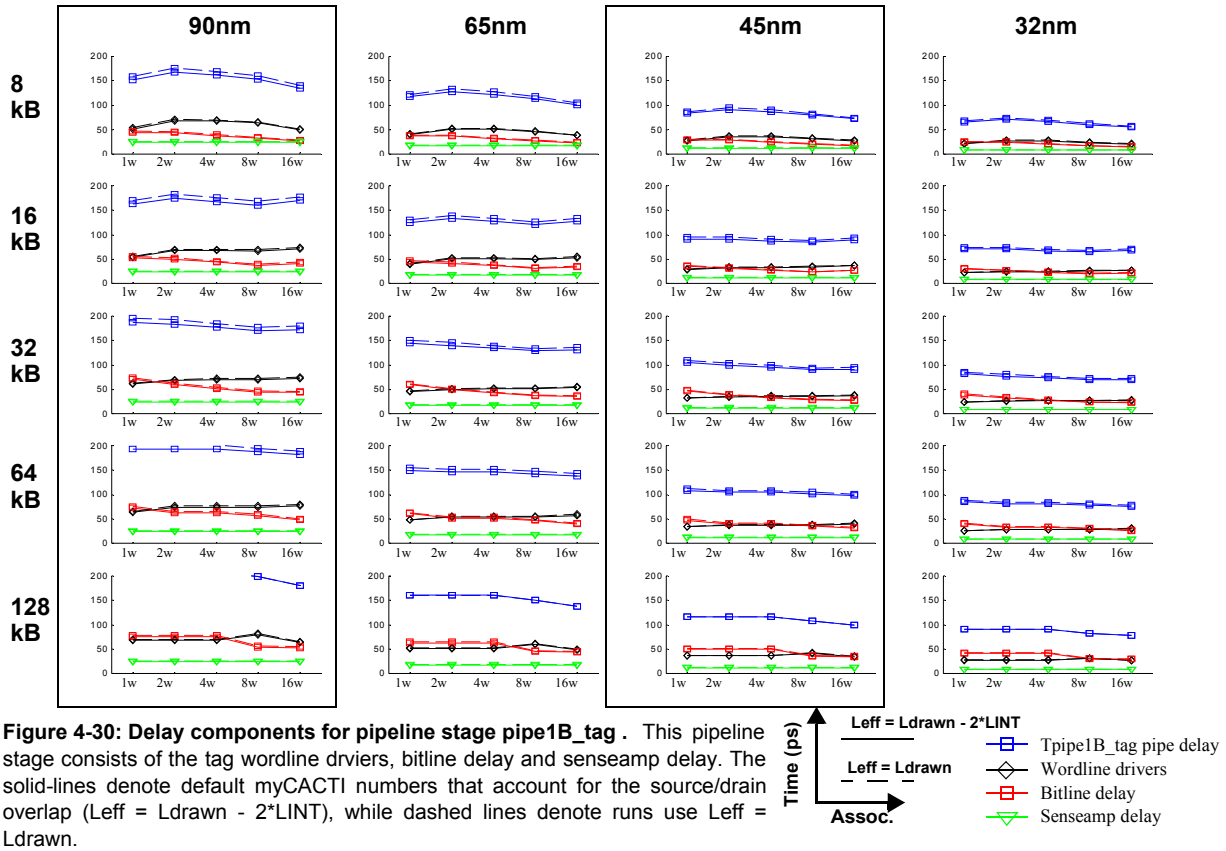


Figure 4-30: Delay components for pipeline stage pipe1B_tag . This pipeline stage consists of the tag wordline drviers, bitline delay and senseamp delay. The solid-lines denote default myCACTI numbers that account for the source/drain overlap (Leff = Ldrawn - 2*LINT), while dashed lines denote runs use Leff = Ldrawn.

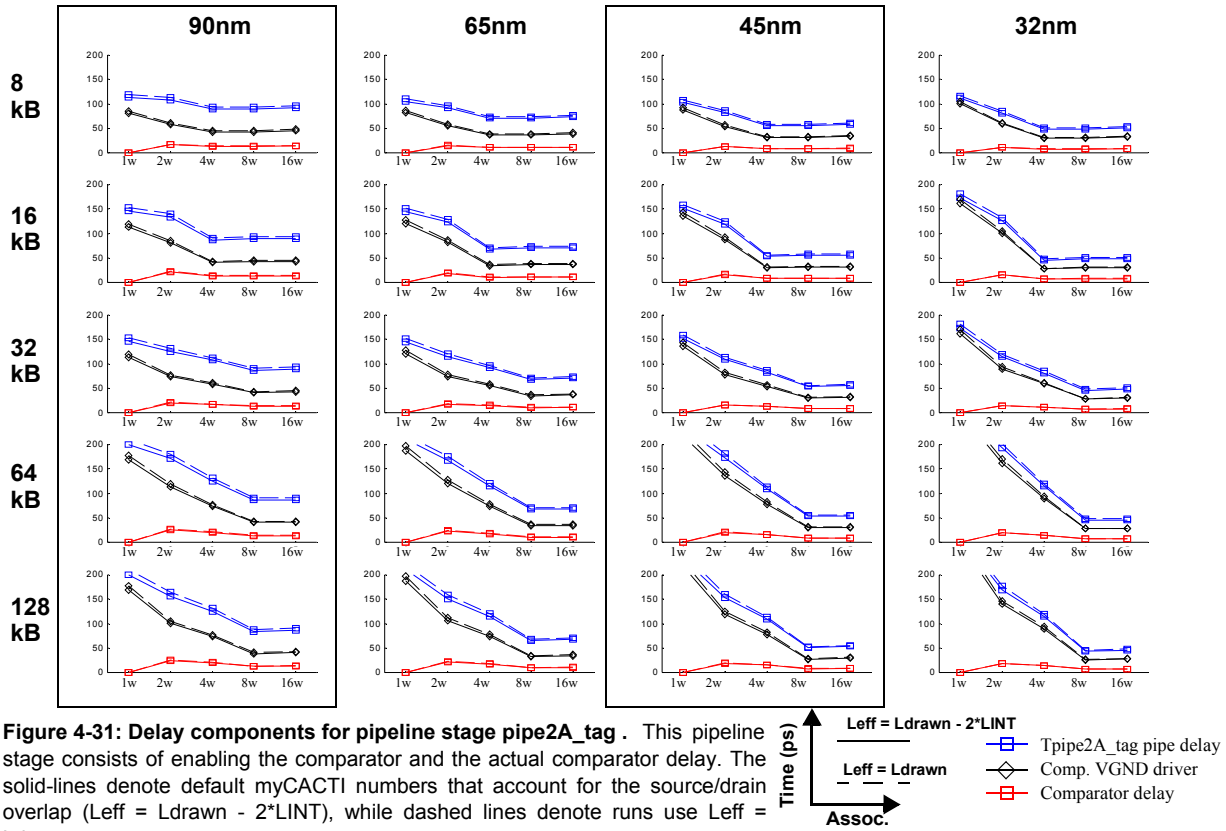


Figure 4-31: Delay components for pipeline stage pipe2A_tag . This pipeline stage consists of enabling the comparator and the actual comparator delay. The solid-lines denote default myCACTI numbers that account for the source/drain overlap (Leff = Ldrawn - 2*LINT), while dashed lines denote runs use Leff = Ldrawn.

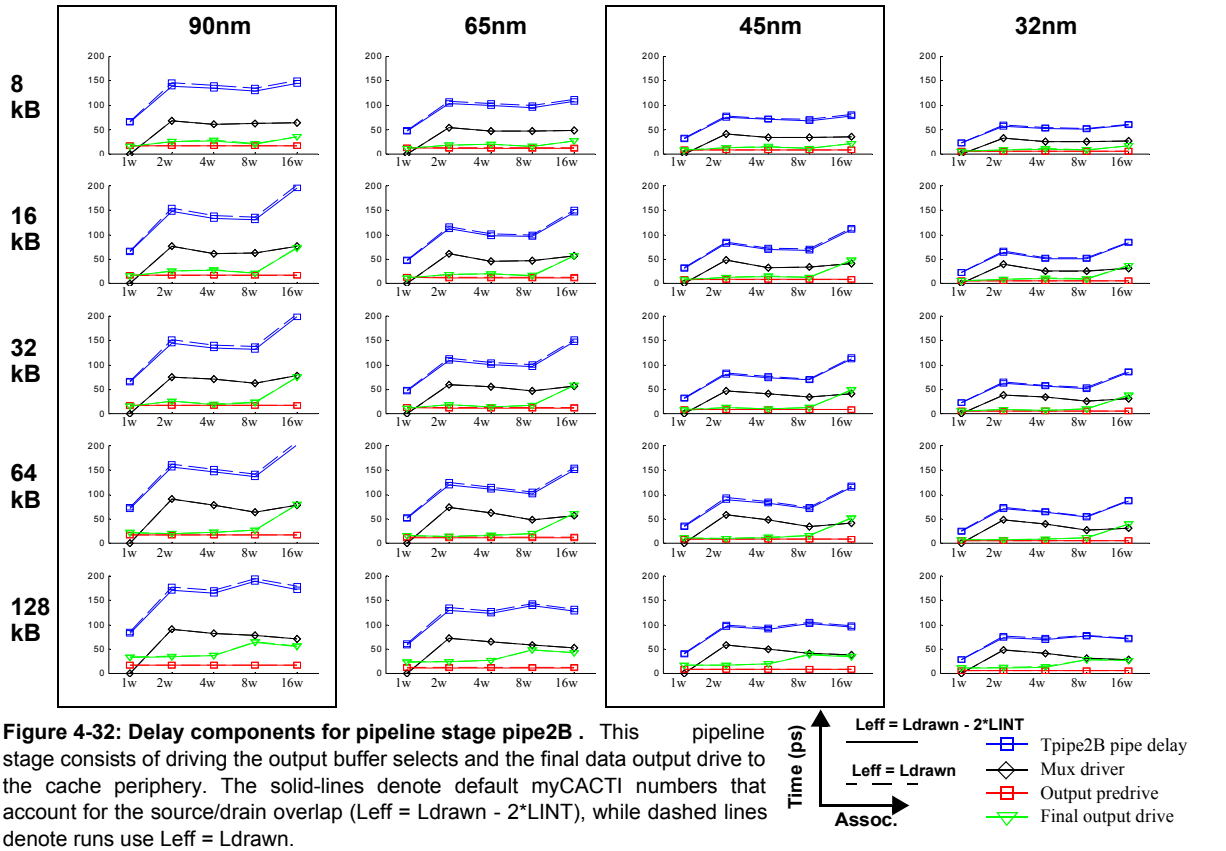


Figure 4-32: Delay components for pipeline stage pipe2B . This pipeline stage consists of driving the output buffer selects and the final data output drive to the cache periphery. The solid-lines denote default myCACTI numbers that account for the source/drain overlap (Leff = Ldrawn - 2*LINT), while dashed lines denote runs use Leff = Ldrawn.

is inversely dependent on length, and having Leff < Ldrawn will result in a higher current (which is modeled by myCACTI as a lower effective resistance) for all transistors used in the modeling.

Looking at the pipe stage pipe2A_data (containing the wordline drivers, the bitline and the senseamp), the bitline sees a larger improvement compared to the wordline driver and sense components. Since the wordline driver and the sense amp transistors both have some freedom to be enlarged and optimized for speed (without too much area or power penalty), these are typically sized reasonably large. In contrast the transistors in the memory cell do not have this freedom and are usually made as close to minimum-sized as possible (of course, while meeting SNM requirements). This means that transistors within the memory cell are almost never optimized for the corresponding load that they drive in the bitline. Consequently, improving the memory cell effective resistance will produce a larger gain compared to improving the effective resistance of near-optimally-sized transistors within the wordline drivers and sense amplifiers.

Finally, the previous observation will probably be even more significant for cache implementations that use more rows per subarrays. For our particular optimizations, the typical run results in the usage of at most 64 rows (with 16 to 32 being more typical). Because of this although we already observe a larger improvement in bitline delay compared to others, we expect this effect to be even more significant for configurations that use more rows. As an example, for 90nm 8kB-8way with 16 rows, the wordline driver

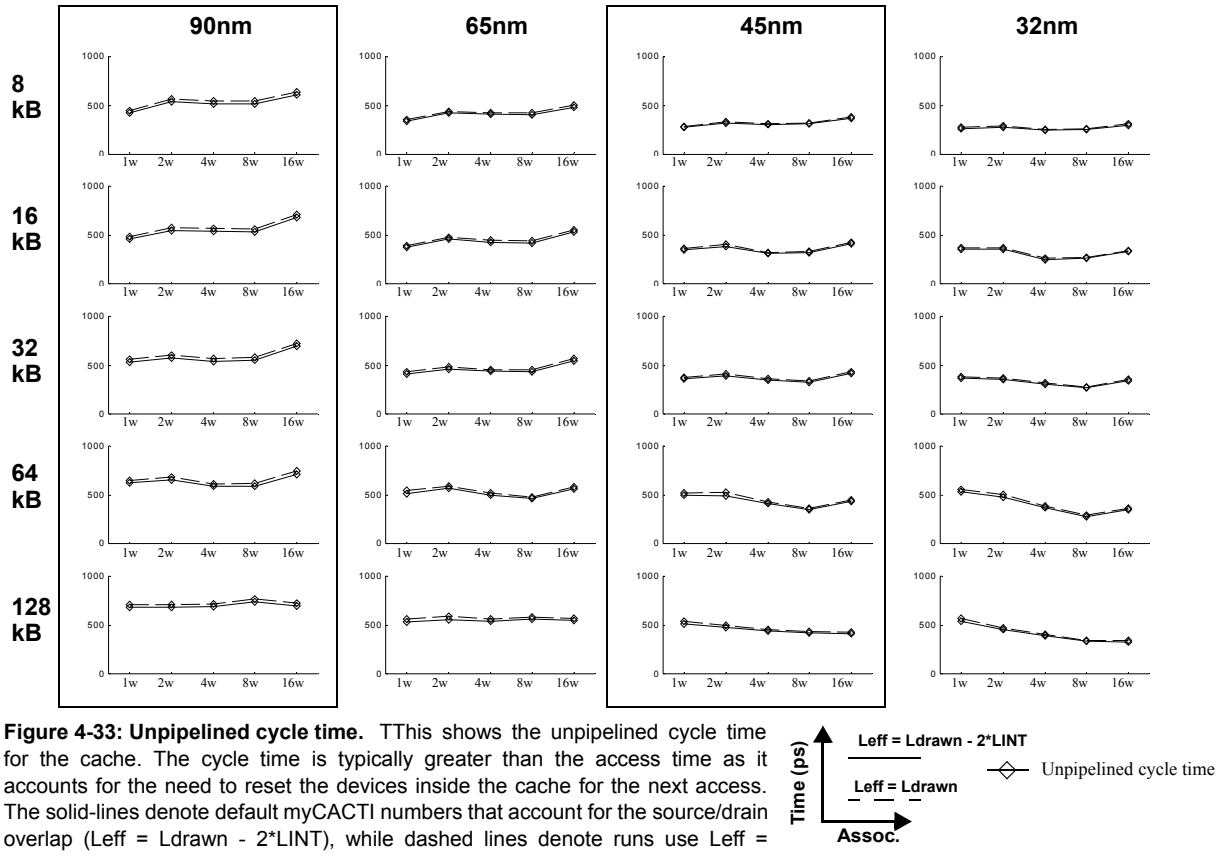


Figure 4-33: Unpipelined cycle time. TThis shows the unpipelined cycle time for the cache. The cycle time is typically greater than the access time as it accounts for the need to reset the devices inside the cache for the next access. The solid-lines denote default myCACTI numbers that account for the source/drain overlap (Leff = Ldrawn - 2*LINT), while dashed lines denote runs use Leff =

improved by 1.5ps while the bitline improved by 1.75ps. In contrast, for 90nm 8kB-1way with 64 rows, the wordine driver improved by 2ps, while the bitline delay improved by 2.5ps.

Run power results

Figure 4-34 shows the total power and its breakdown into dynamic power (switching power) and static power (dissipated by subthreshold leakage and gate leakage currents) for every point in the design space that was considered (i.e. cache size, associativity and technology node). Each plot shows both the results from myCACTI runs that account for the drain/source overlap region and runs that simply use Leff = Ldrawn. Figure 4-35 demonstrates the differences more explicitly by plotting the fractional difference between the two runs normalized to the total power of the default myCACTI result. Figure 4-36 shows detailed power breakdowns of some representative design points in order to explore in detail the causes of the differences. In addition, the results from the two runs are placed side by side for easy comparison. Lastly, Figure 4-37 identifies how much power is being dissipated by major cache blocks, and again shows the results from the two runs side by side for easy comparison.

From these figures, we can see that at the larger nanometer nodes (e.g. 90nm and 65nm), accounting for the drain/source overlap region when computing for the transistor effective length does not result in significant differences in terms of computing for power. For the 45nm node, a reduced Leff produces enough

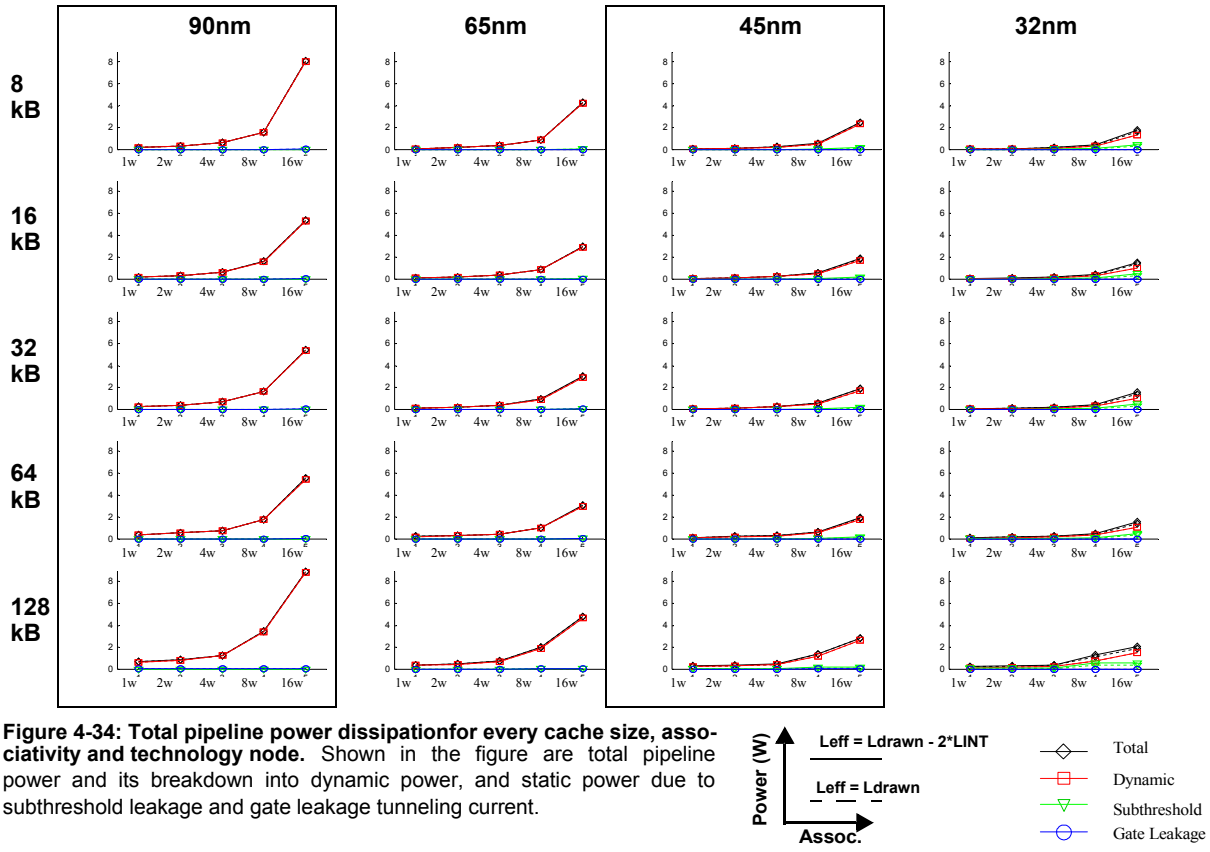


Figure 4-34: Total pipeline power dissipationfor every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

subthreshold leakage power such that the additional power dissipated by subthreshold leakage current (normalized by total power) is up to 5%, and this error is almost an "offset" type of error. For 32nm, the reduced Leff has a larger effect -- increasing the delta from the subthreshold leakage power to the 10% to 18% range. Again, this error is mostly an offset type error, with minor peaking and dipping for some configurations. Note that these observations apply to most of the cache sizes and associativities studied.

It is interesting to note that it is mostly static power dissipated by subthreshold leakage currents that are affected once the modeling accounts for the source/drain overlap region. The gate leakage tunneling current should, in theory, be affected as the tunneling current from the gate to the channel is different from the tunneling to the overlap regions. But since magnitude of the gate leakage is pretty small, the differences we measure are negligible. Dynamic power is largely unaffect because with Cgate = Cchannel + Coverlap, Cchannel is unchanged because it is determined by Ldrawn, while Cchannel is also unchanged because the junction area is also left unchanged.[1]

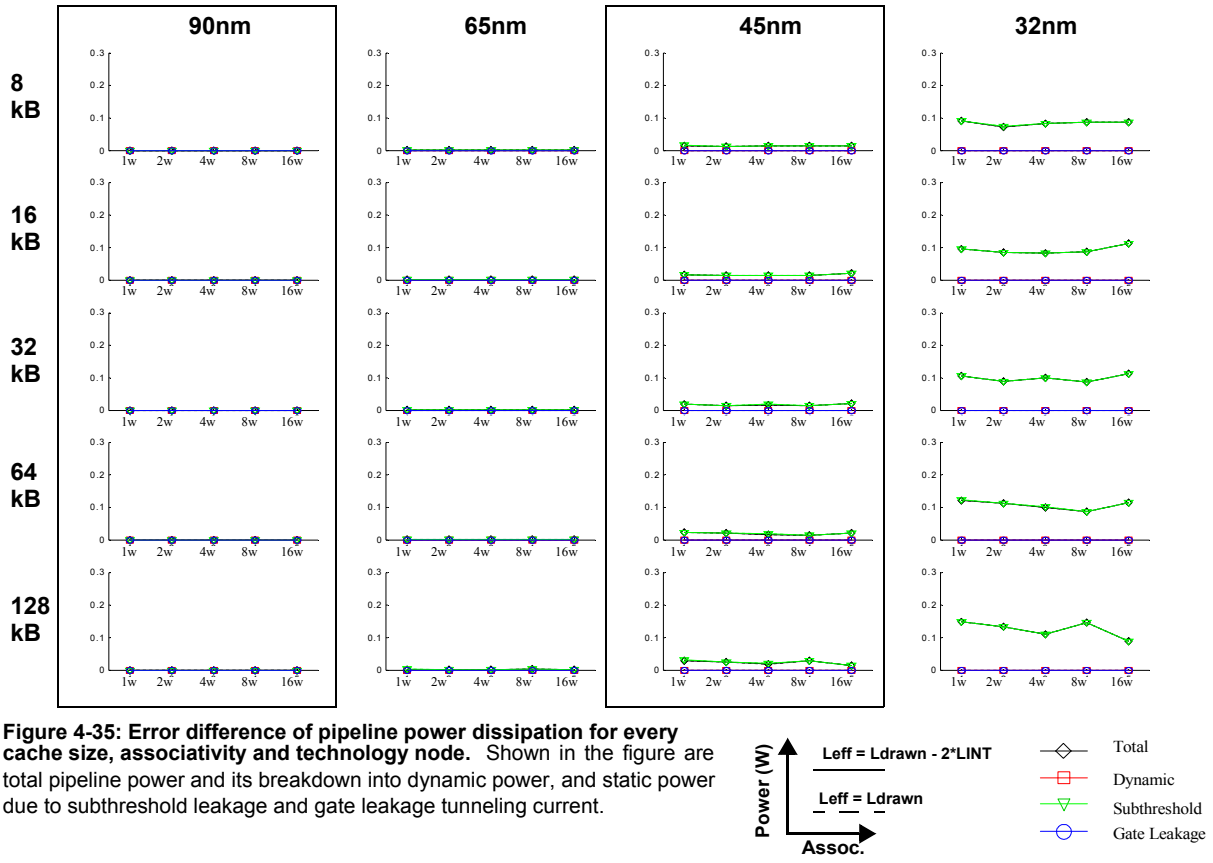


Figure 4-35: Error difference of pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

1. We are assuming here that when we account for the drain/source overlap, the entire drain or source terminal is pulled closer into the transistor and eats into the channel region instead of being extended. With this assumption, the diffusion capacitance of the drain or source terminals are left largely unchanged. In cases where the assumption doesn't hold, the diffusion capacitance increase should be small enough not to result in significant changes.

Looking at the breakdown of power into its component blocks, we again see that for the larger technology nodes (e..g 90nm and 65nm), there is minimal differences in power. At 45nm, all configurations

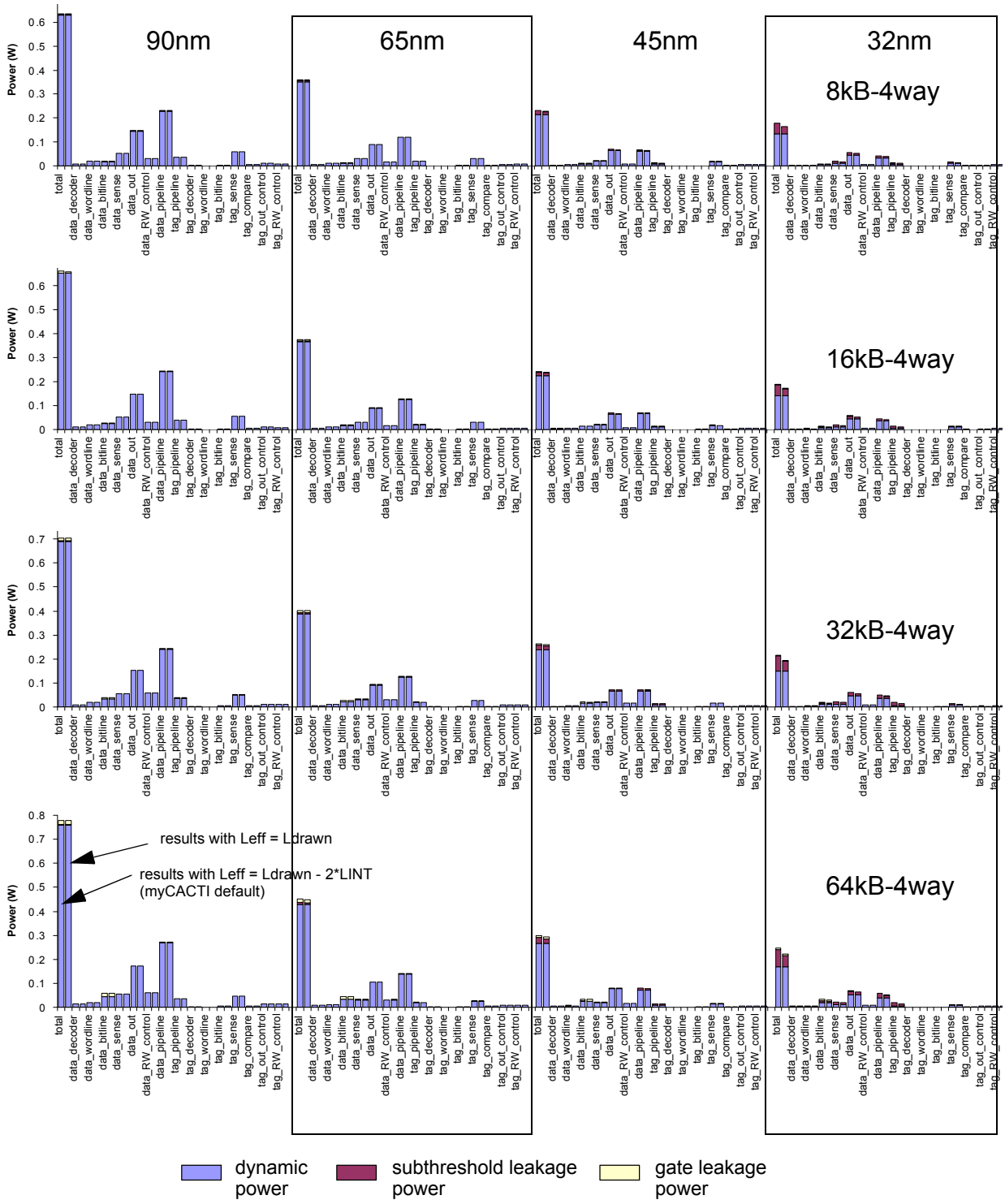


Figure 4-36: Detailed cache power breakdown comparison. Detailed power breakdown for four different cache configurations showing the amount of dynamic, subthreshold leakage and gate leakage power being consumed by different parts of the cache. Each plot shows four sets of values corresponding to different technology nodes. (Note that all three plots use the same scale). The two results are also placed side by side for direct comparison.

studied in detail show measurable differences in subthreshold leakage but only the 128kB (and to some extent, the 64kB) configuration showed really significant changes. It is interesting to note that the total difference is not due to any single component, but is attributable to all cache subblocks in proportion to their original contribution of subthreshold leakage. Put another way, the subthreshold leakage components of each individual block seems to have been affected equally (which makes sense) such that each would contribute to the difference by how much leakage power each block had in the first place. Again, this makes sense, since the the subthreshold leakage of a transistor should be affected equally regardless of its width. This may have been different if we model transistors with lengths greater than the minimum length allowed by the process such

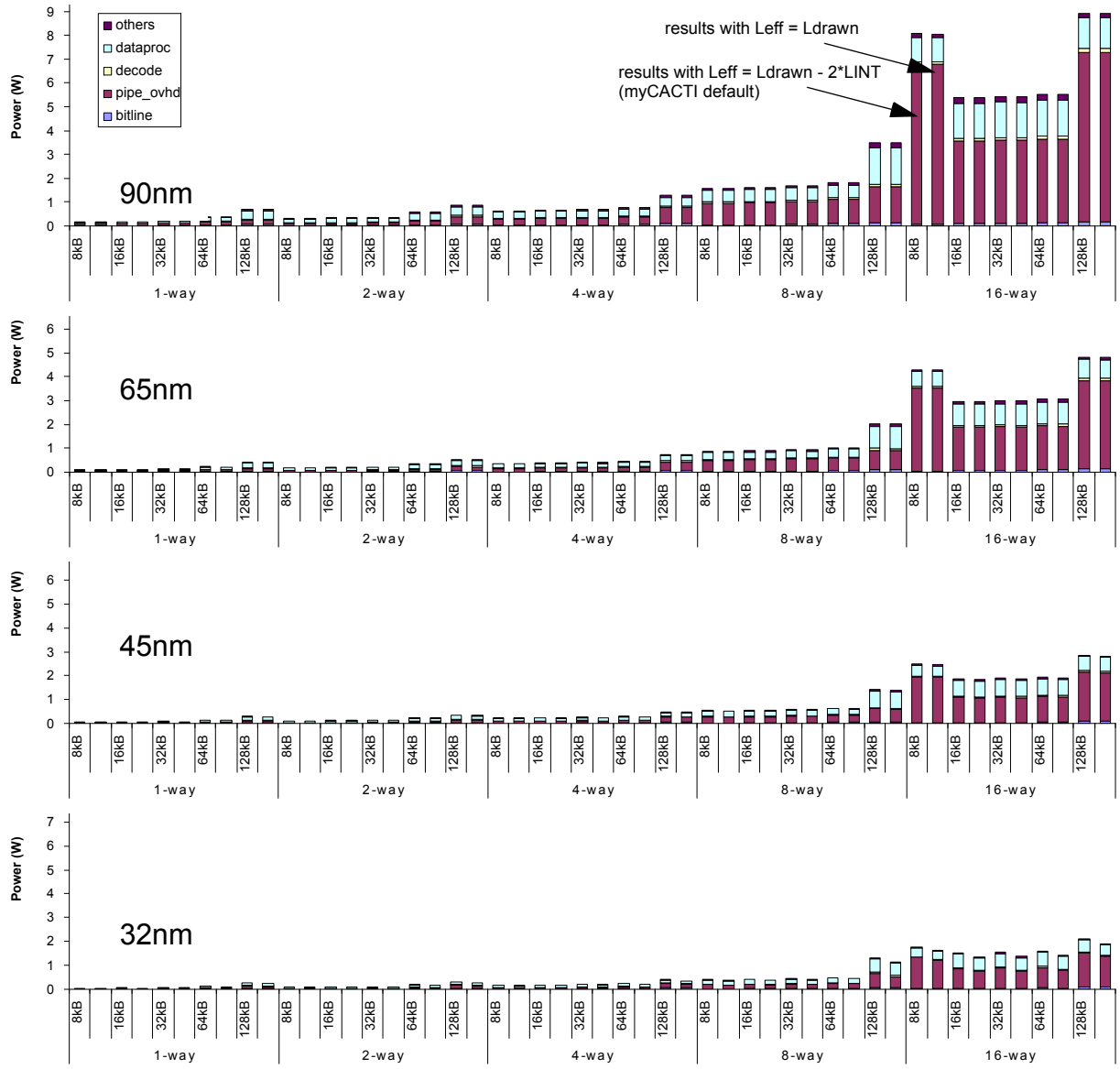


Figure 4-37: Power contributions of different cache parts. Power breakdown showing major cache power contributors for the 65nm and 32nm technology nodes for different cache sizes and associativities. (Note that the y-axis for both plots use the same scale). The two results are also placed side by side for direct comparison.

that the ratio Leff/Ldrawn may be different for these transistors. At 32nm, the previous observations stated have become really significant, but the same trends seem to apply.

Looking at the cache power breakdown into its logical subblocks, we simply see essentially the same observations that have already been stated. An additional observation is that for 128kB in the 45nm node, the power distribution for 16-way is a "u" type curve, while the distribution is more like a "saddle" shape for 32nm. Most of the changes seem to have been caused by changes in the pipeline overhead power, mainly because it accounted for more leakage originally.

Conclusion

We have shown that modeling the presence of the drain/source overlap region when computing for the transistor's effective length generally results in faster delay with a corresponding increase in power. The delay improvement can be observed by all pipeline stages since all transistors within the cache are affected. Additionally, the power increase is almost all due to an increase in subthreshold leak power because of the (often) shorter transistor channel, with dynamic and gate leakage mostly unaffected. This power difference is observed to be mostly significant for the deep nanometer nodes (e.g. 45nm and 32nm), but the delay difference is observed at every point in the design space (i.e. every cache size, associativity and technology node).

4.5.3 Via parasitic capacitance

Run details

To produce these data, myCACTI was made to do a design space exploration using the default settings of myCACTI (in particular, enabling via capacitance) to find the optimal implementation for every cache size and associativity for the 90nm technology node. After determining the optimal implementations for each point in the 90nm design space (in terms of the cache parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl), three more sets of runs are performed for each technology node, with each point in the design space utilizing the optimal implementation that was determined for the 90nm node. In essence, the data for 65nm, 45nm and 32nm constitute a sort of "process-shrink" where we take a design in an older process technology and convert it to a newer one. Although the runs for 65nm and below will not show the absolute optimal implementation possible, this method was chosen to provide a simple but valid comparison between different technology nodes. Implementing the same cache configuration (i.e. same cache size and associativity) using two different implementations for different technology nodes would make the comparisons between different cache components very difficult and generalizations will be hard to make.

After generating a complete set of data using myCACTI default settings, the process is repeated, this with each myCACTI run being set to disable via capacitance (by using the -disable_viacaps option).

Run timing results

Figure 4-38 shows the delay results for all cache configurations for both runs where results are generated with the via capacitance enabled (the myCACTI default, shown in the plots as the solid lines), and with via capacitance disabled (shown as dashed lines). This figure shows the resulting optimal clock period for each configuration, along with the pipe delays for all of the stages in the cache (with the exception of pipe1A_data and pipe1B_data -- the initial decode stages -- since these always have significant slack so they have been omitted from the figure for clarity). Figures 4-39 to 4-45 show the delay component for each individual stage. These individual plots provide a breakdown of the delay of each individual stage by showing the component delays of each. Lastly, Figure 4-46 shows the unpipelined cache access time.

The comparison shows that ignoring the effects of via capacitances results in significant differences in timing (up to roughly 50ps). Simulations that do not model the degradation in delay cause by the additional capaciting loading in the vias result in significantly underestimating the delay. Because of pipelining, where a pipe delay may or may not be the determinant of the clock period, this delay error is not an offset error.

We observe the biggest degradations in the data wordline drivers and the bitline, with the wordline drier seeing more deterioration. This makes sense given the smaller number of rows in our optimal implementations (max of 64, typically 16 to 32) such that the number of memory cells connecting to the bitline (and requiring a via) is significantly fewer than the number of memory cells attached to the wordline (in

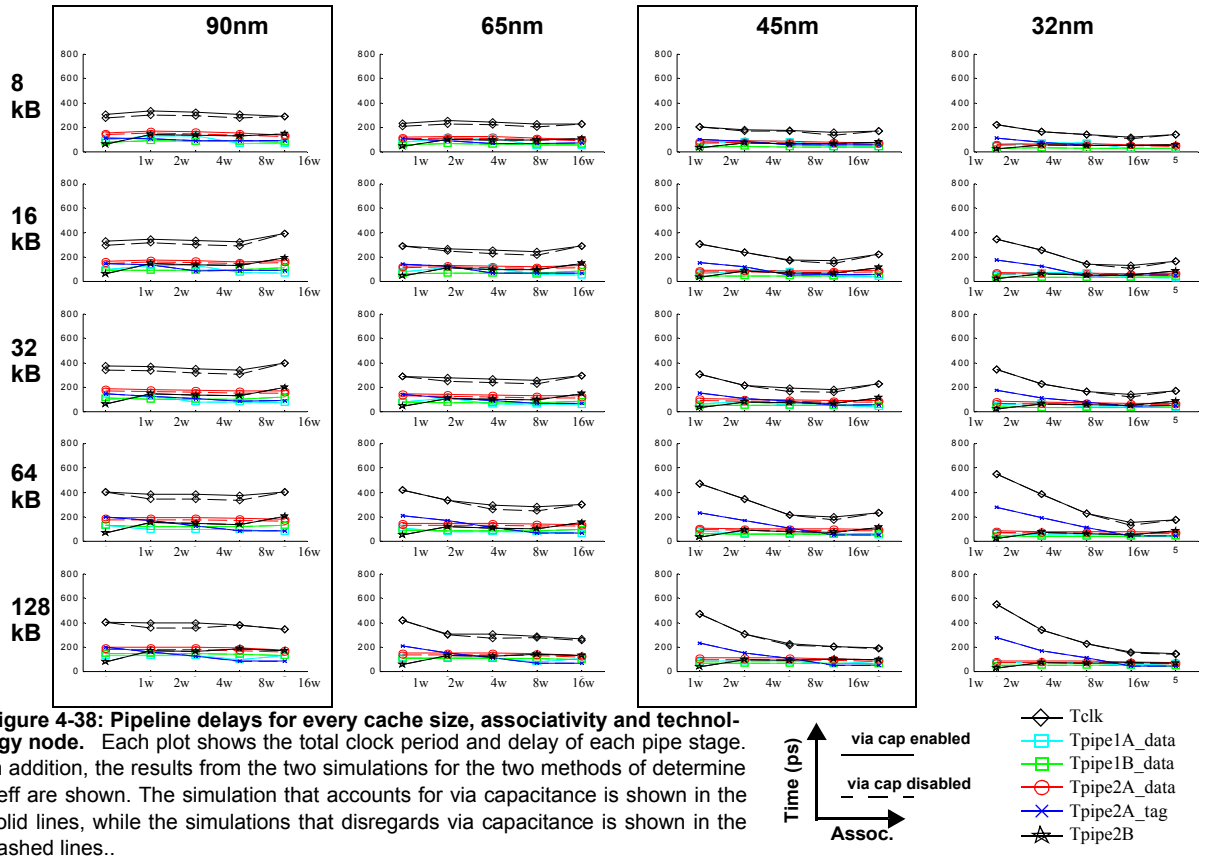


Figure 4-38: Pipeline delays for every cache size, associativity and technology node. Each plot shows the total clock period and delay of each pipe stage. In addition, the results from the two simulations for the two methods of determine Leff are shown. The simulation that accounts for via capacitance is shown in the solid lines, while the simulations that disregards via capacitance is shown in the dashed lines..

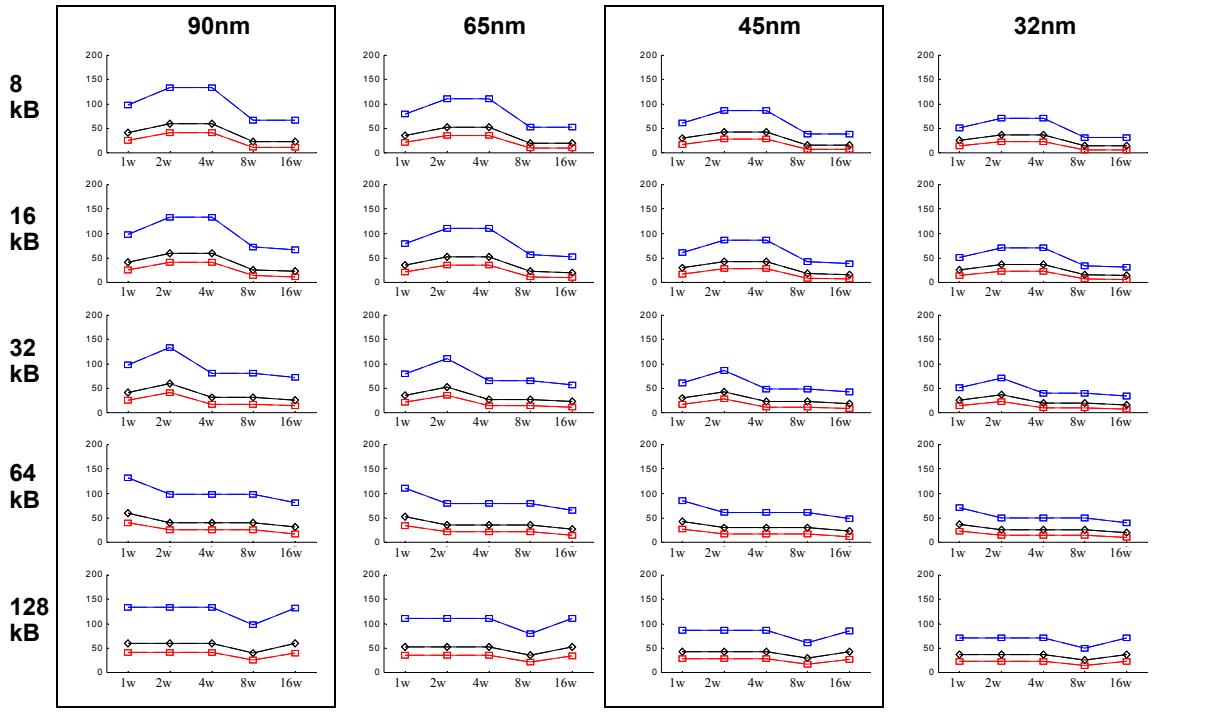


Figure 4-39: Delay components for pipeline stage pipe1A_data . This pipeline stage consists of data address buffering and the predecoder front-end. The solid-lines denote default myCACTI numbers that account for via capacitances, while the dashed lines denote numbers that were produced with via capacitances ignored..

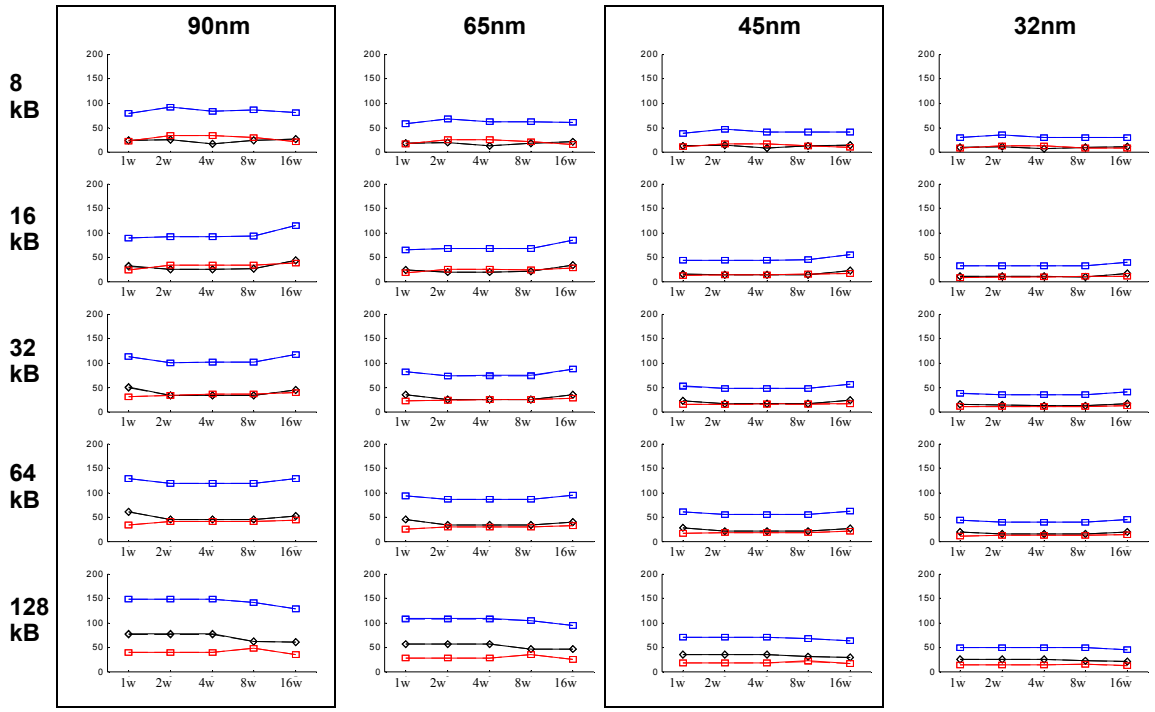


Figure 4-40: Delay components for pipeline stage pipe1B_data . This pipeline stage consists of the data predecoder driver and the wordline front-end. The solid-lines denote default myCACTI numbers that account for via capacitances, while the dashed lines denote numbers that were produced with via capacitances ignored.

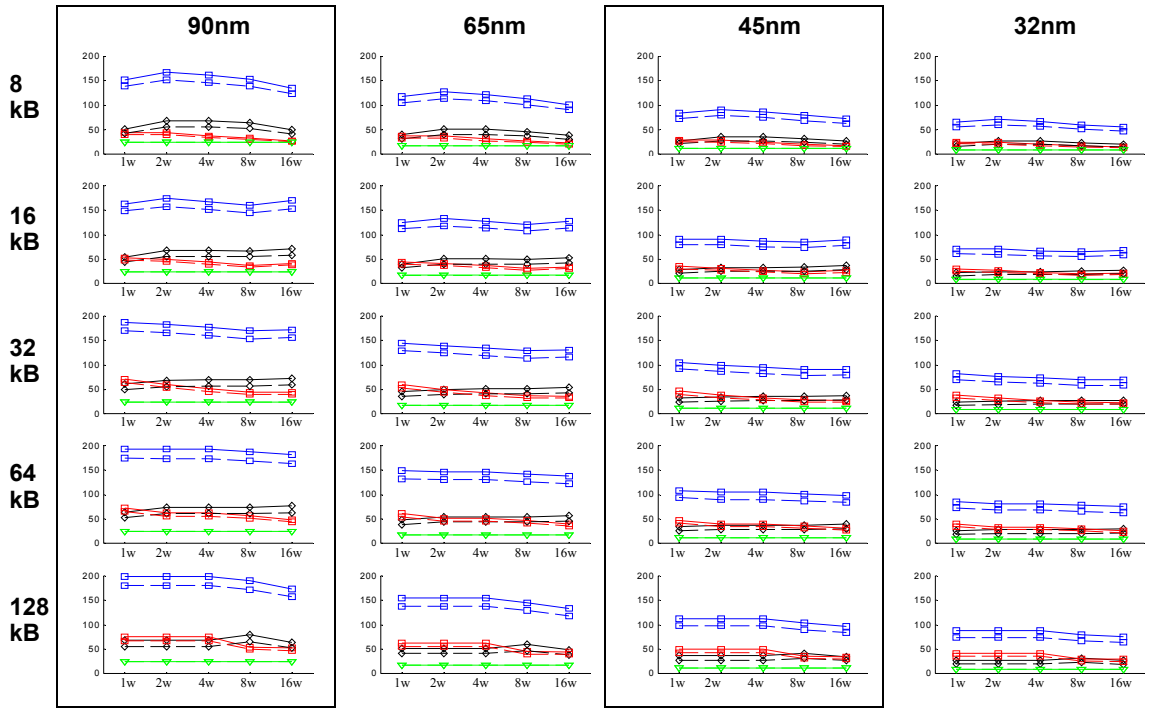


Figure 4-41: Delay components for pipeline stage pipe2A_data . This pipeline stage consists of the wordline drivers, the bitline and the sense amplifier. The solid-lines denote default myCACTI numbers that account for via capacitances, while the dashed lines denote numbers that were produced with via capacitances ignored.

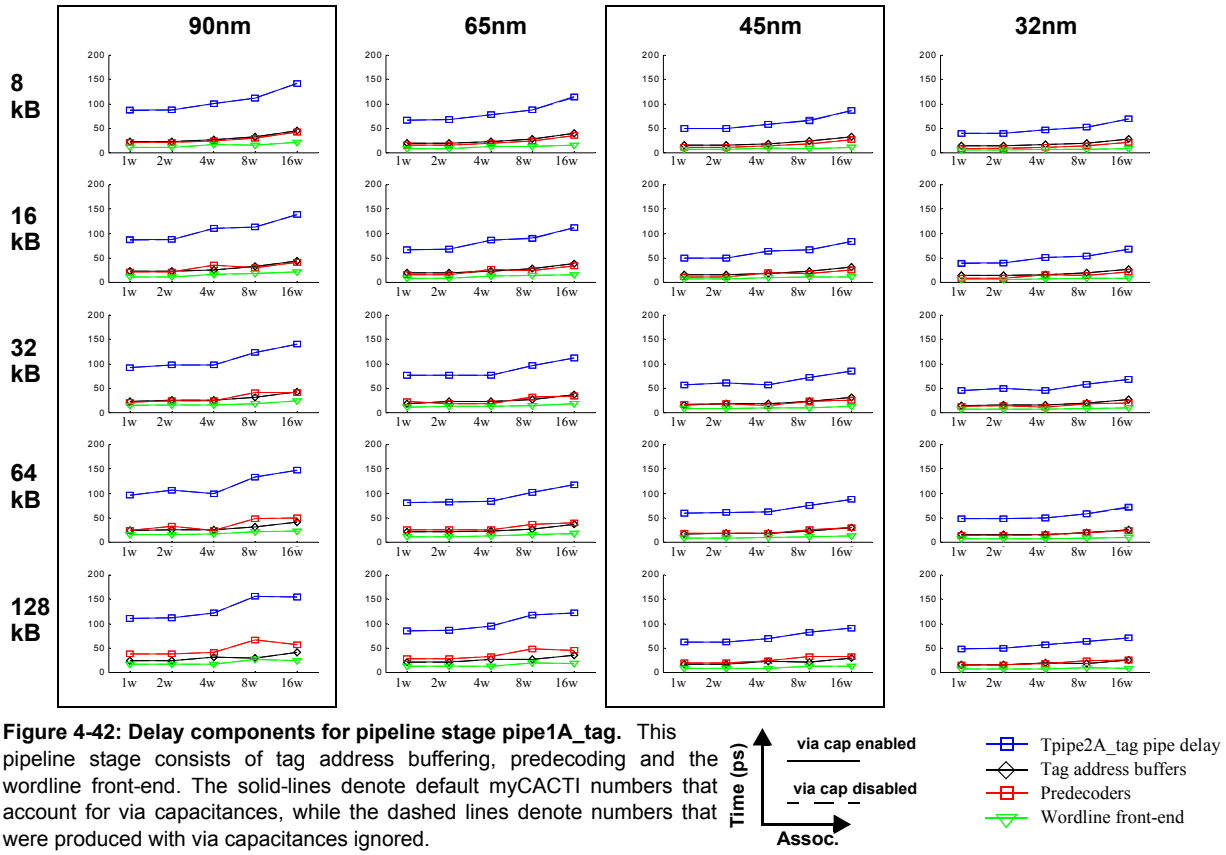


Figure 4-42: Delay components for pipeline stage pipe1A_tag. This pipeline stage consists of tag address buffering, predecoding and the wordline front-end. The solid-lines denote default myCACTI numbers that account for via capacitances, while the dashed lines denote numbers that were produced with via capacitances ignored.

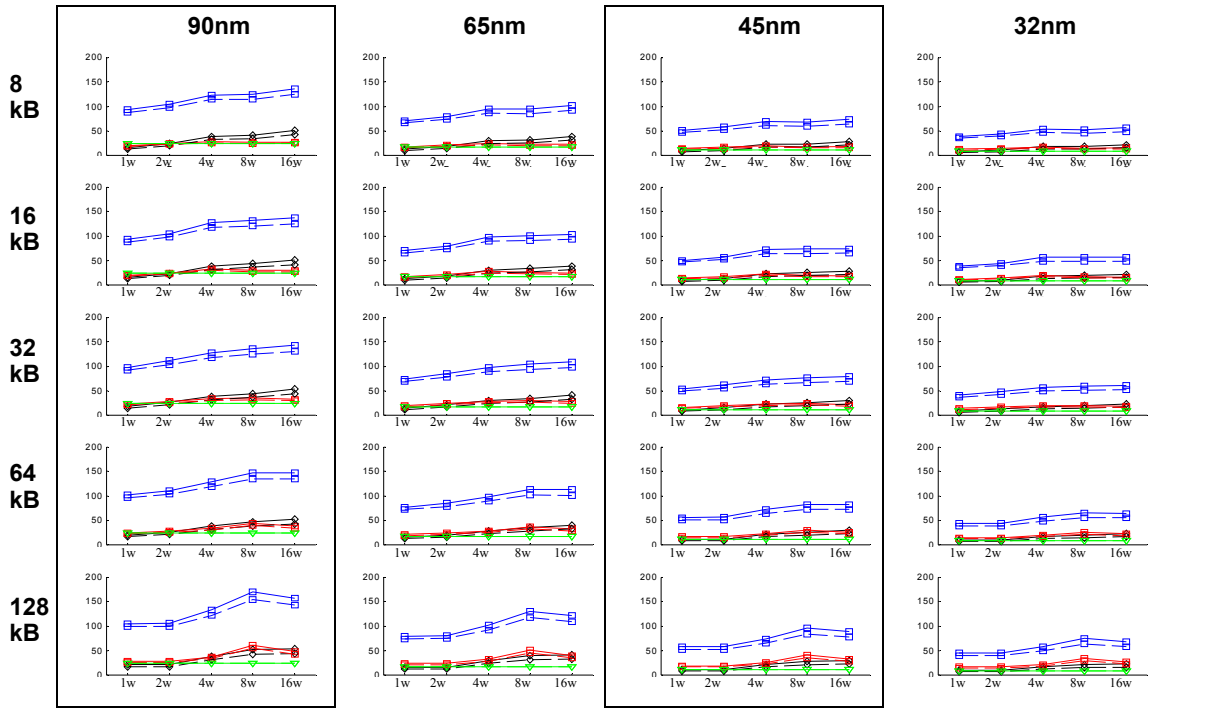


Figure 4-43: Delay components for pipeline stage pipe1B_tag . This pipeline stage consists of the tag wordline drviers, bitline delay and senseamp delay. The solid-lines denote default myCACTI numbers that account for via capacitances, while the dashed lines denote numbers that were produced with via capacitances ignored.

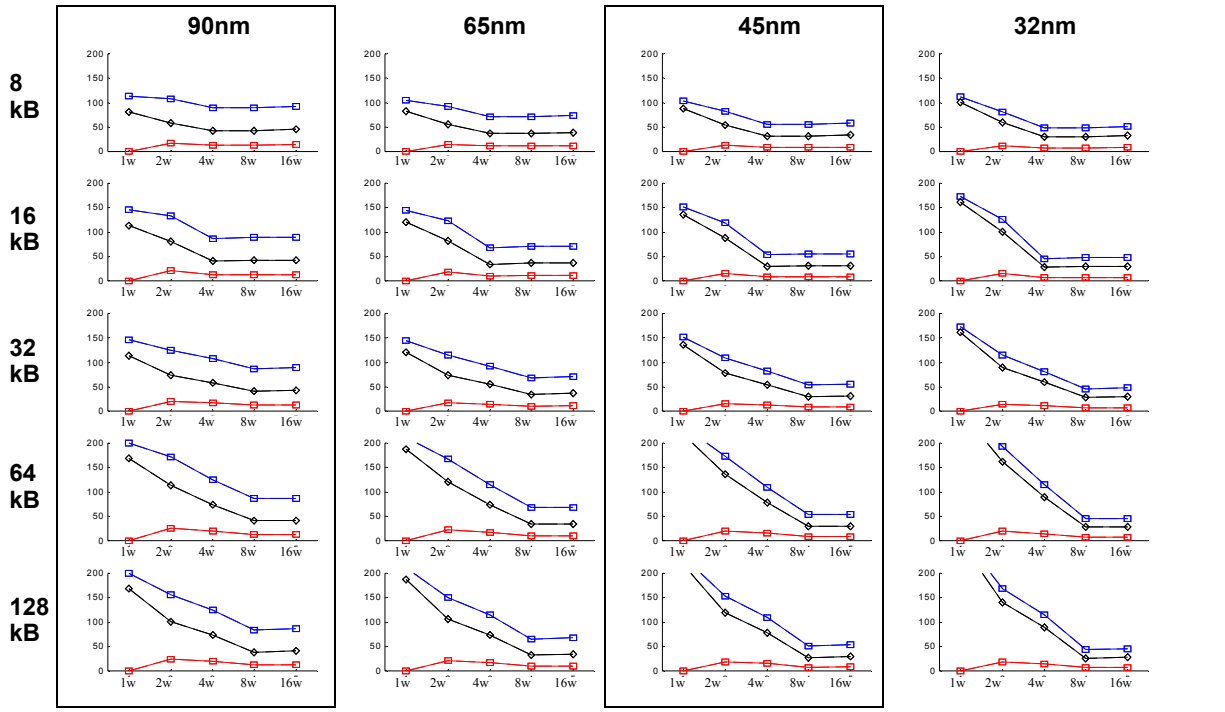


Figure 4-44: Delay components for pipeline stage pipe2A_tag . This pipeline stage consists of enabling the comparator and the actual comparator delay. The solid-lines denote default myCACTI numbers that account for via capacitances, while the dashed lines denote numbers that were produced with via capacitances ignored.

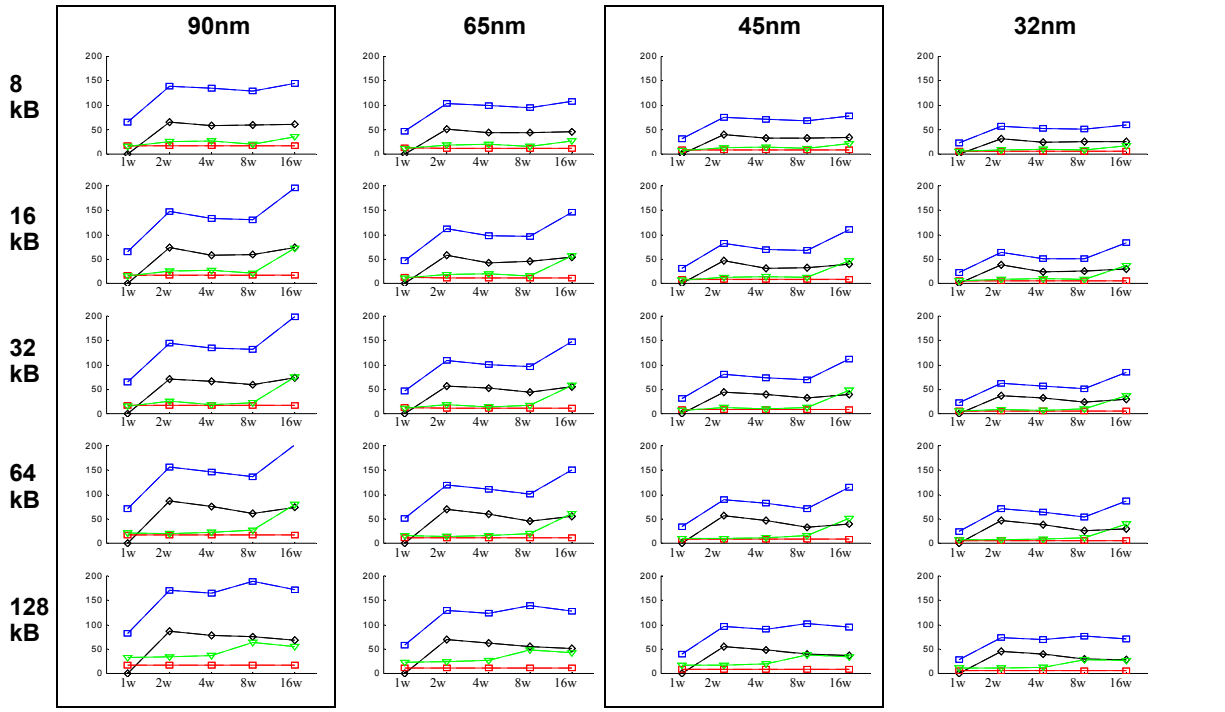


Figure 4-45: Delay components for pipeline stage pipe2B . This pipeline stage consists of driving the output buffer selects and the final data output drive to the cache periphery. The solid-lines denote default myCACTI numbers that account for via capacitances, while the dashed lines denote numbers that were produced with via capacitances ignored.

this case, a roughly equal distribution of 256 and 512 columns). The tag bitline and wordline drivers are affected in the same way, but to a much lesser degree because of the smaller size of the tag array.This effect is typically more significant in the larger technology nodes -- smaller technology nodes still suffer from the poor scaling of the comparator transistors such that the comparator delay often becomes the critical path and serves to hide the degraded delays of the predecoder/wordline/bitline stages, which are the stages most affected by via capacitance loading.

Run power results

Figure 4-47 shows the total power and its breakdown into dynamic power (switching power) and static power (dissipated by subthreshold leakage and gate leakage currents) for every point in the design space that was considered (i.e. cache size, associativity and technology node). Each plot shows both the results from myCACTI runs that consider or ignore the presence of via capacitances. Figure4-48 demonstrates the differences more explicitly by plotting the fractional difference between the two runs normalized to the total power of the default myCACTI result. Figure 4-49 shows detailed power breakdowns of some representative design points in order to explore in detail the causes of the differences. In addition, the results from the two runs are placed side by side for easy comparison. Lastly, Figure 4-50 identifies how much power is being

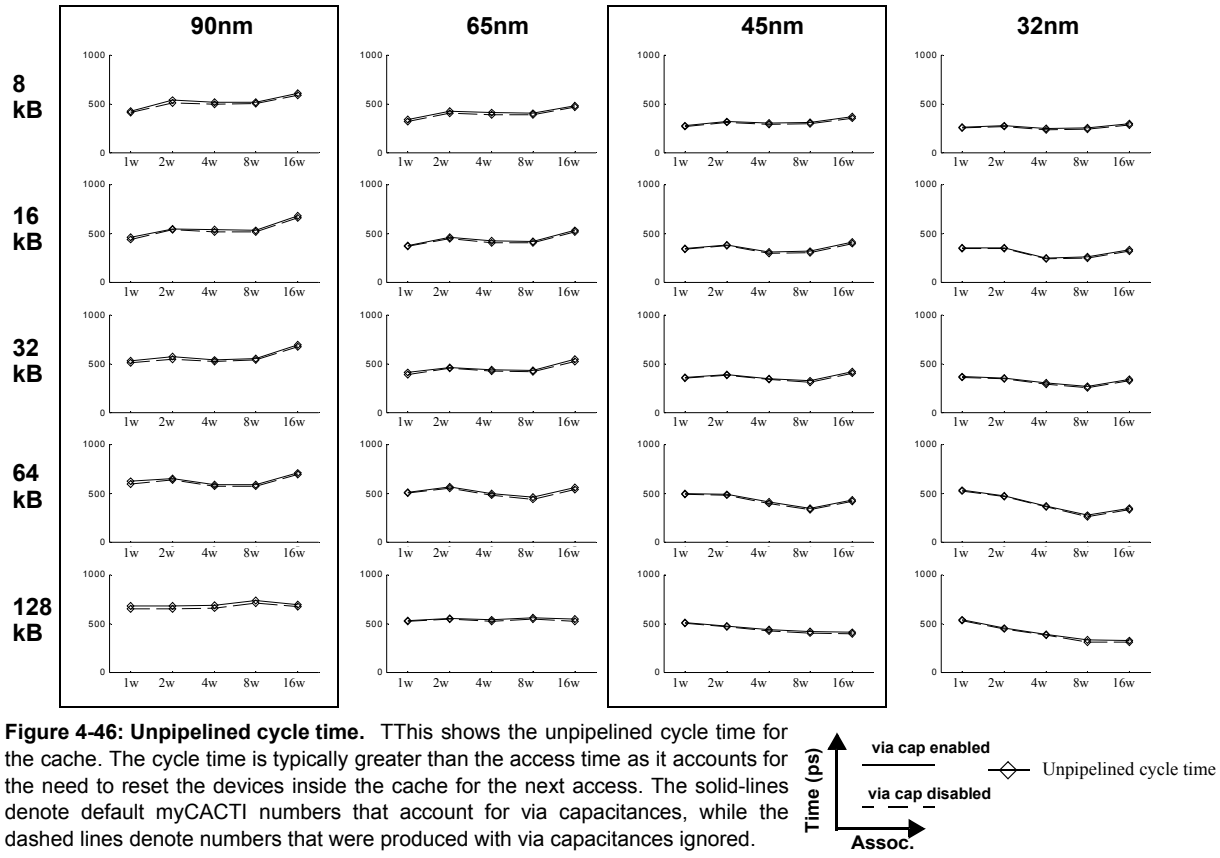


Figure 4-46: Unpipelined cycle time. TThis shows the unpipelined cycle time for the cache. The cycle time is typically greater than the access time as it accounts for the need to reset the devices inside the cache for the next access. The solid-lines denote default myCACTI numbers that account for via capacitances, while the dashed lines denote numbers that were produced with via capacitances ignored.

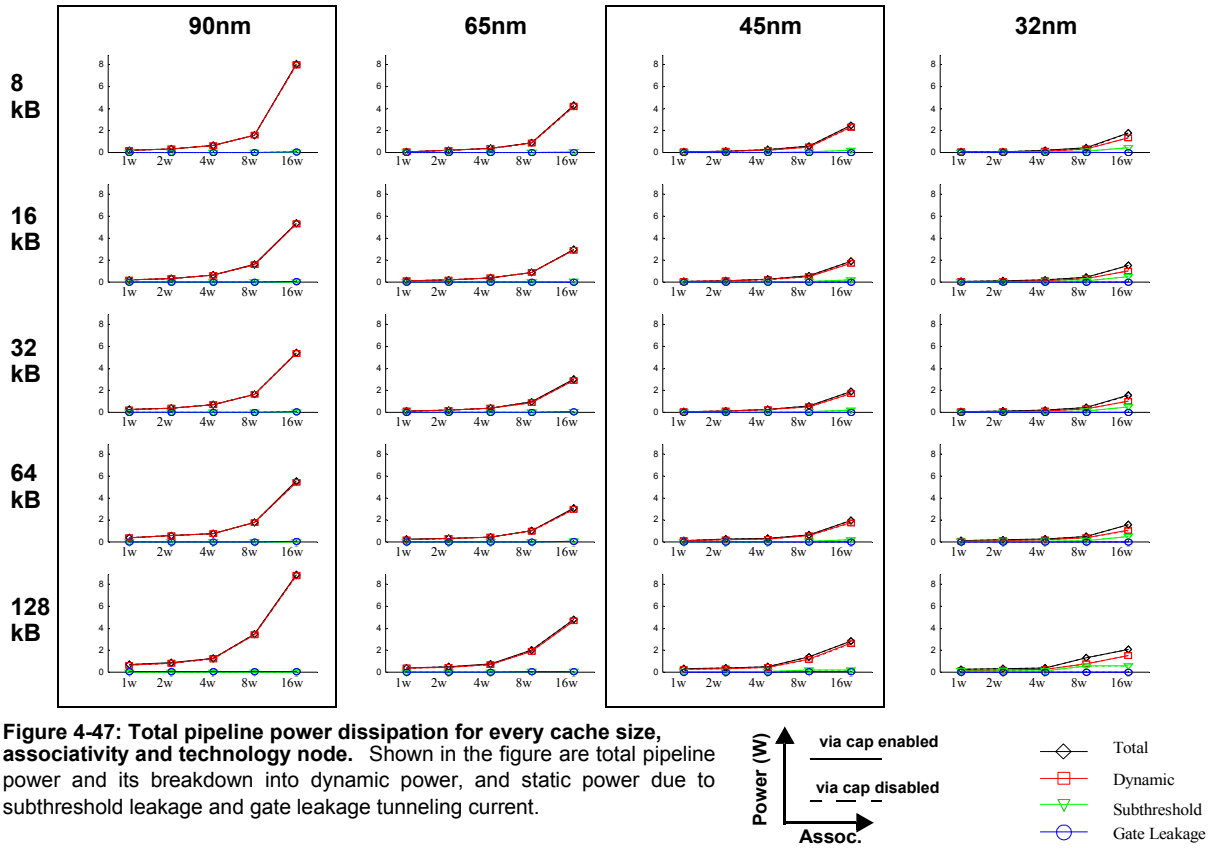


Figure 4-47: Total pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

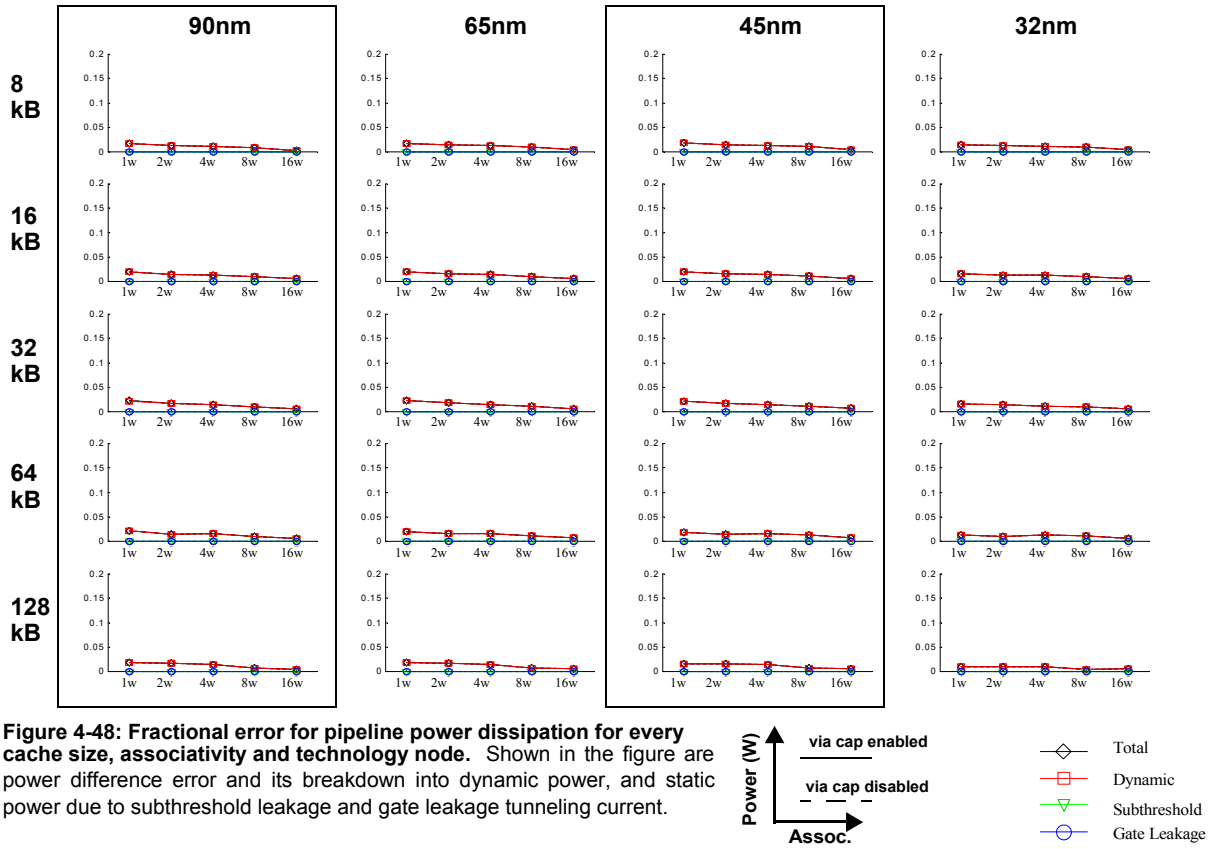


Figure 4-48: Fractional error for pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are power difference error and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

dissipated by major cache blocks, and again shows the results from the two runs side by side for easy comparison.

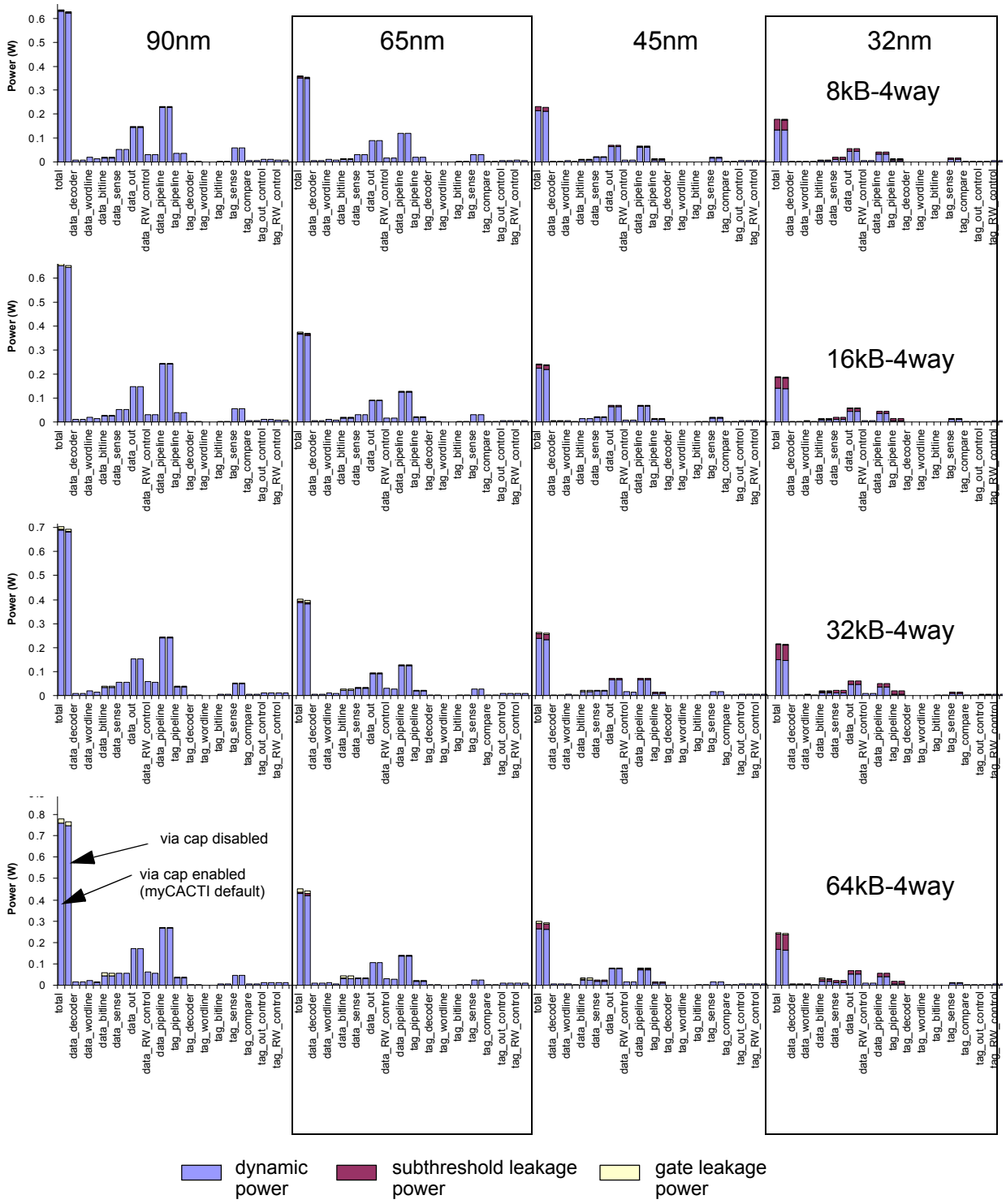


Figure 4-49: Detailed cache power breakdown comparison. Detailed power breakdown for four different cache configurations showing the amount of dynamic, subthreshold leakage and gate leakage power being consumed by different parts of the cache. Each plot shows four sets of values corresponding to different technology nodes. (Note that all three plots use the same scale). The two results are also placed side by side for direct comparison.

From the figures, we see that there are minimal differences in power between simulations that account and do not account for via capacitances. Looking at the error plots, the 1-way configurations that have the via cap enabled typically have about 3% error compared to the runs that ignore the via capacitance, and this error tends to decrease with increasing associativity. This is due to the observation that via caps introduce the most effect in the predec, wl and bitlines, and increasing associativity introduces more dynamic power in terms of latching, comparison and such, but this doesn't increase the need for more vias in the aforementioned three locations.

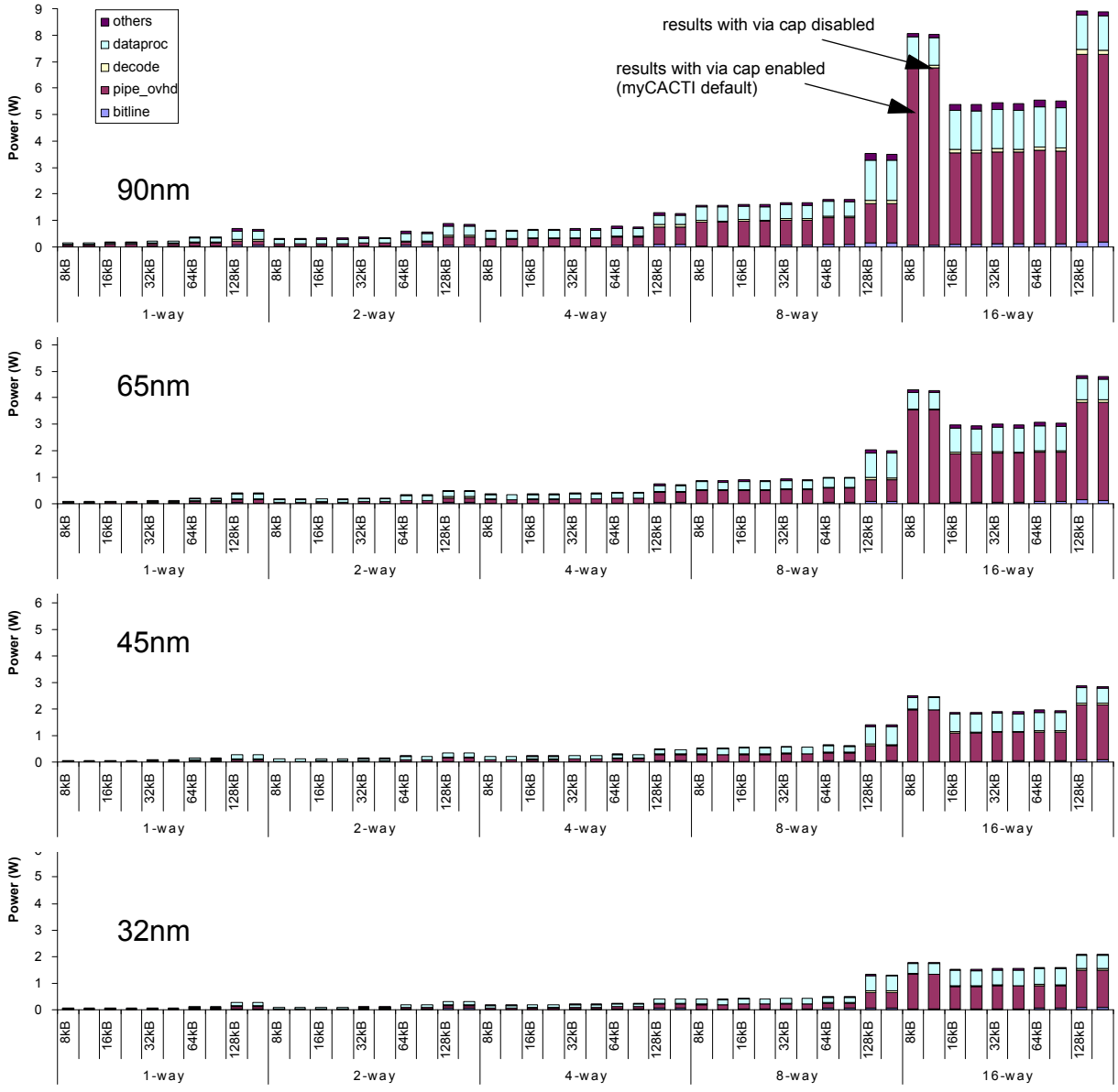


Figure 4-50: Power contributions of different cache parts. Power breakdown showing major cache power contributors for the 65nm and 32nm technology nodes for different cache sizes and associativities. (Note that the y-axis for both plots use the same scale). The two results are also placed side by side for direct comparison.

Although the minimal effect is surprising at first, we recognize that accounting for via capacitances is only expected to increase the dynamic power of the active circuits in the cache. Since the number of active rows and columns constitute only a small portion of the cache, especially in subarrayed implementations, the additional dynamic power due to additional capacitive load because of the vias is minimal.

Conclusion

We have shown that modeling the additional capacitance presented by interconnect vias result in minimal differences in terms of cache power, but significant differences in terms of cache delay. In addition, because of the pipelined nature of the cache that we model, pipeline stages that have their delays degraded by accounting for the additional via capacitances do not necesarily constitute the critical path. Consequently, some configurations, even if some of their pipe stages become stretched out due to the additional delays caused by the via caps, may actually have their effective clock cycle unchanged if the critical path resides in stages that are not largely affected by the via caps (e.g. the tag compare stage). This results in the comparison error not being of the "offset" type and hence, it is difficult to extrapolate the final result given a simulation that does not account for the via capacitance.

We conclude that it is important to include modeling of via capacitances during the simulation, as it potentially has a significant effect on the cache delays.

4.5.4 Pipelining comparison

Run details

To produce these data, myCACTI was made to do a design space exploration using the default settings of myCACTI (in particular, enabling the pipeline latches) to find the optimal implementation for every cache size and associativity for the 90nm technology node. After determining the optimal implementations for each point in the 90nm design space (in terms of the cache parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl), three more sets of runs are performed for each technology node, with each point in the design space utilizing the optimal implementation that was determined for the 90nm node. In essence, the data for 65nm, 45nm and 32nm constitute a sort of "process-shrink" where we take a design in an older process technology and convert it to a newer one. Although the runs for 65nm and below will not show the absolute optimal implementation possible, this method was chosen to provide a simple but valid comparison between different technology nodes. Implementing the same cache configuration (i.e. same cache size and associativity) using two different implementations for different technology nodes would make the comparisons between different cache components very difficult and generalizations will be hard to make.

After generating a complete set of data using myCACTI default settings, the data for the other set of numbers is generated by setting the power from the pipeline overhead components (pipeline latches, clock tree drivers, etc.) to zero.

Run power results

Figure 4-51 shows the total power and its breakdown into dynamic power (switching power) and static power (dissipated by subthreshold leakage and gate leakage currents) for every point in the design space that was considered (i.e. cache size, associativity and technology node). Each plot shows both the results from myCACTI with and without pipelining. Figure4-52demonstrates the differences more explicitly by plotting the fractional difference between the two runs normalized to the total power of the default myCACTI result.

From the results, we see that there are drastic differences in power for all cache configurations if the power due to pipeline overhead is not accounted for. In the 90nm, 65nm and 45nm technology nodes, simulations that ignore pipeline overhead power typically underestimate total power by 50%, with most of this difference due to dynamic power, with the difference exhibiting a saddle shape that peaks around the 2way/ 4way configuration. (This doesn't say anything about optimality, only that the difference between accounting and disregarding pipeline power peaks at these associativities). In the 32nm node, the difference becomes even greater, as subthreshold leakage starts to become really significant and adds rougly 20% more to the power difference.

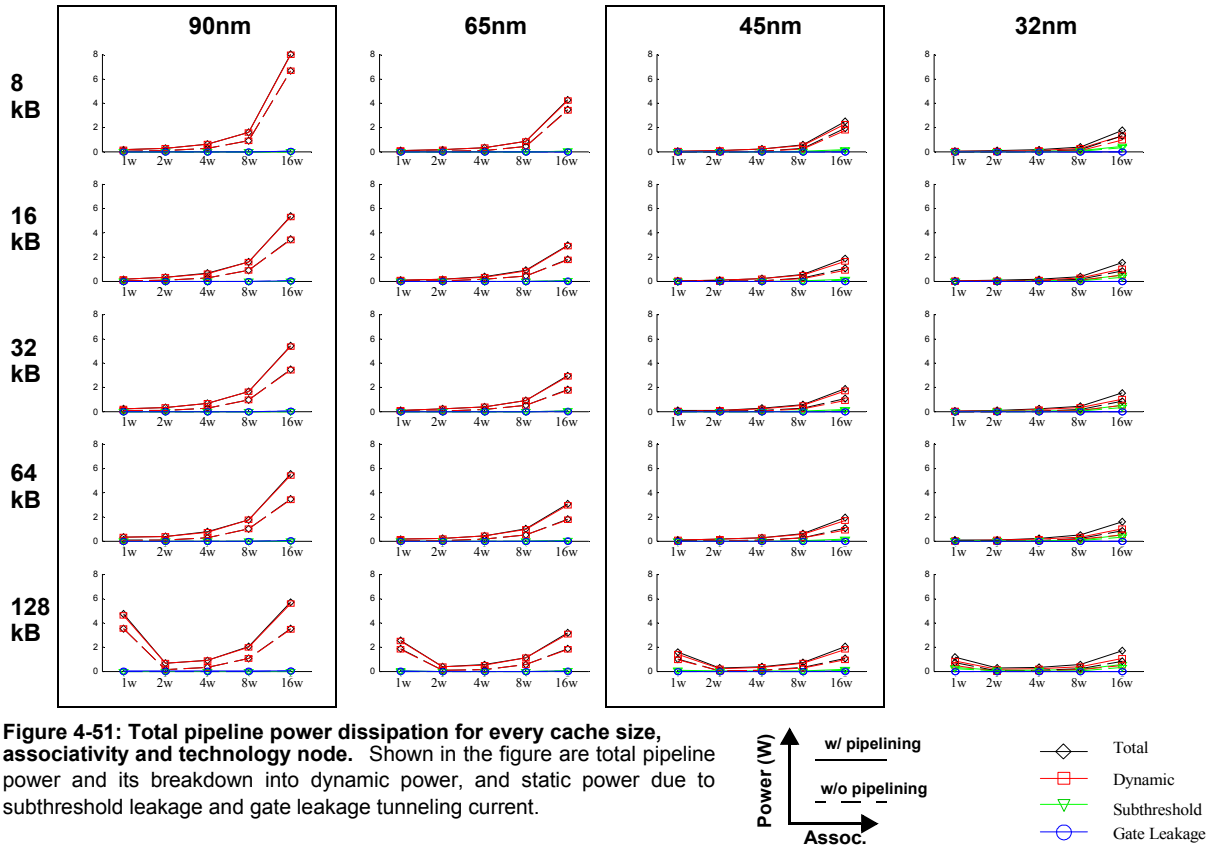


Figure 4-51: Total pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

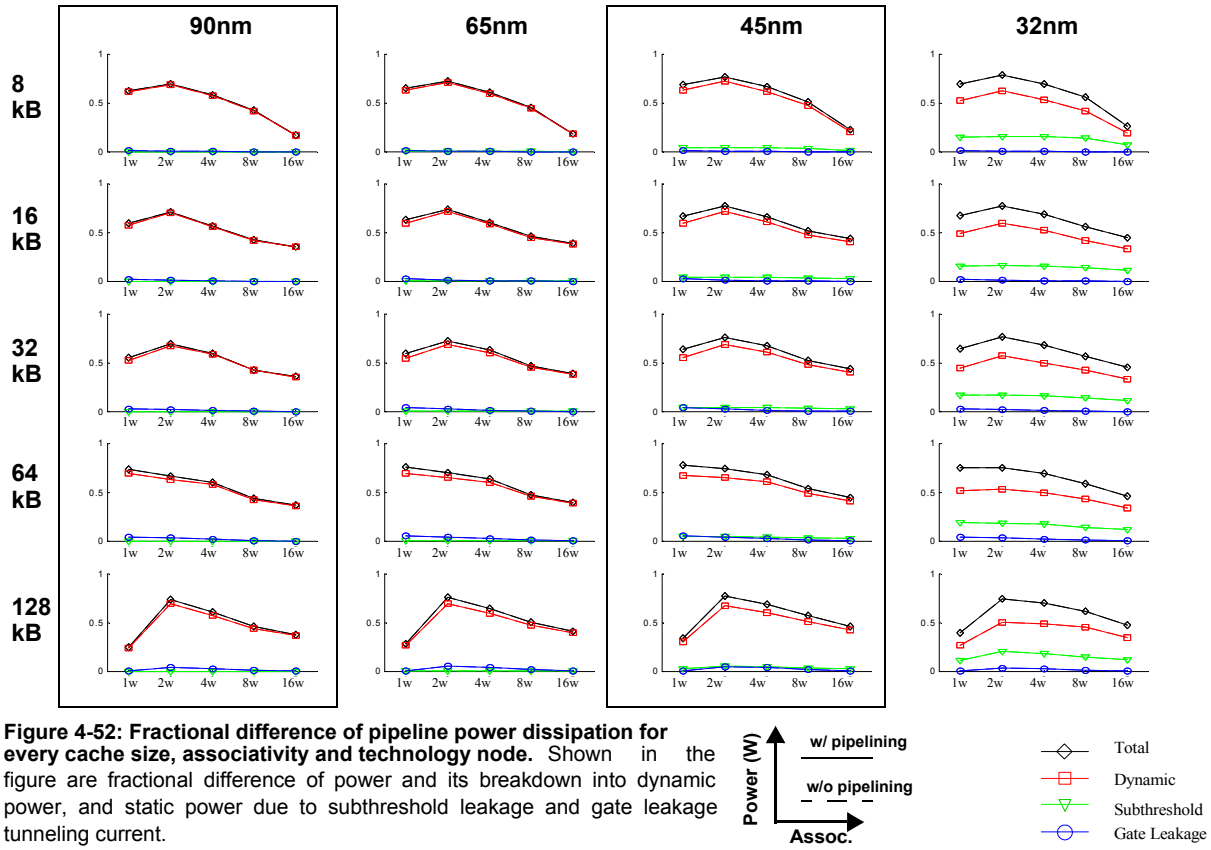


Figure 4-52: Fractional difference of pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are fractional difference of power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

Conclusion

We have shown that cache designers cannot expect to trivially add pipelining to an existing cache design without expecting to drastically alter the power envelope of the circuit, as the overhead required for pipeline elements constitutes a significant fraction of total cache power. Also, we expect the delay behavior of the cache to change because this is inherent to pipelining and is one of the reasons we perform pipelining in the first place, but for this section, we only emphasize the importance of the need for accounting for pipeline power, and pipeline delays are discussed in more detail in a later section.

4.5.5 Number of interconnect layers

Run details

To produce these data, myCACTI was made to do a design space exploration using the default settings of myCACTI (in particular, enabling multiple metal interconnect layers) to find the optimal implementation for every cache size and associativity for the 90nm technology node. After determining the optimal implementations for each point in the 90nm design space (in terms of the cache parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl), three more sets of runs are performed for each technology node, with each point in the design space utilizing the optimal implementation that was determined for the 90nm node. In essence, the data for 65nm, 45nm and 32nm constitute a sort of "process-shrink" where we take a design in an older process technology and convert it to a newer one. Although the runs for 65nm and below will not show the absolute optimal implementation possible, this method was chosen to provide a simple but valid comparison between different technology nodes. Implementing the same cache configuration (i.e. same cache size and associativity) using two different implementations for different technology nodes would make the comparisons between different cache components very difficult and generalizations will be hard to make.

After generating a complete set of data using myCACTI default settings, the process is repeated, this time with each myCACTI run being set to use a single, generic local interconnect layer (by using the -use_singlelayer_metal option).

Run timing results

Figure 4-53 shows the delay results for all cache configurations for both runs where results are generated with the assumption of multi-layer interconnects (the myCACTI default, shown in the plots as the solid lines), and with single-layer interconnects (shown as dashed lines). This figure shows the resulting optimal clock period for each configuration, along with the pipe delays for all of the stages in the cache (with the exception of pipe1A_data and pipe1B_data -- the initial decode stages -- since these always have significant slack so they have been omitted from the figure for clarity). Figures 4-54 to 4-60 show the delay component for each individual stage. These individual plots provide a breakdown of the delay of each individual stage by showing the component delays of each. Lastly, Figure 4-61 shows the unpipelined cache access time.

From the comparison plots, we see that modeling multi-layer interconnects results in very significant delay differences over modeling single-layer interconnects. Of the three typical critical stages (pipe2A_data, pipe2A_tag, and pipe2B), pipe2B, which is the dataout stage, is the one that is typically degraded the most. pipe2A_tag (the compare stage) does mostly a localized operation and as such is almost independent of interconnect characteristics and is largely unaffected. For pipe2A_data, the effect on delay for both the wordline driver and the bitlines depends on the particular configuration. In most cases, the shift from multi-

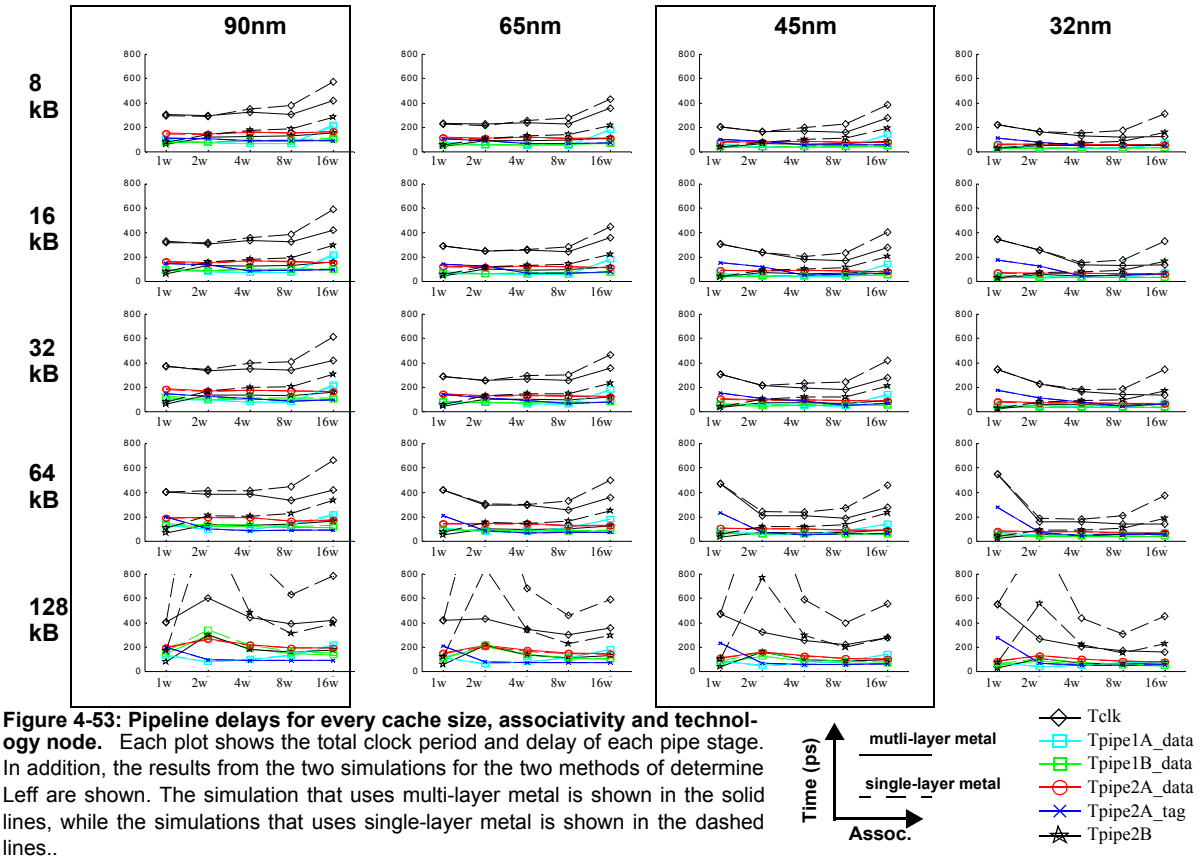


Figure 4-53: Pipeline delays for every cache size, associativity and technology node. Each plot shows the total clock period and delay of each pipe stage. In addition, the results from the two simulations for the two methods of determine Leff are shown. The simulation that uses multi-layer metal is shown in the solid lines, while the simulations that uses single-layer metal is shown in the dashed lines..

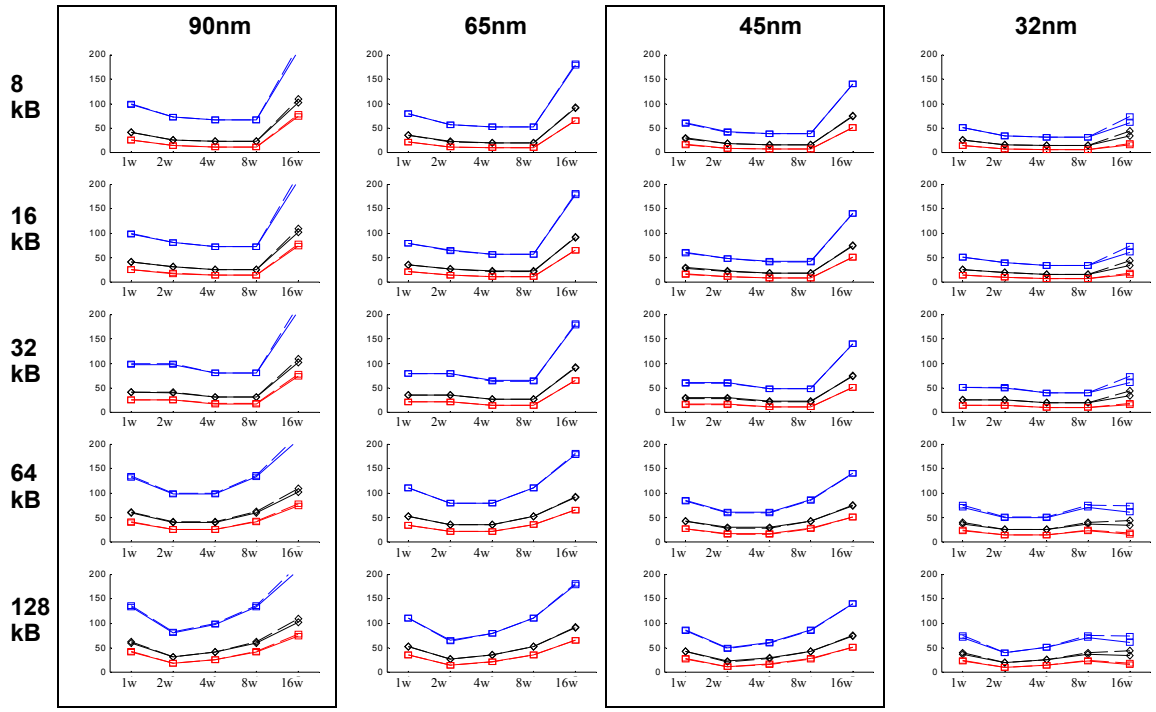


Figure 4-54: Delay components for pipeline stage pipe1A_data . This pipeline stage consists of data address buffering and the predecoder front-end. The solid-lines denote default myCACTI numbers that account for multi-layer metal , while the dashed lines denote numbers that were produced with single-layer metal.

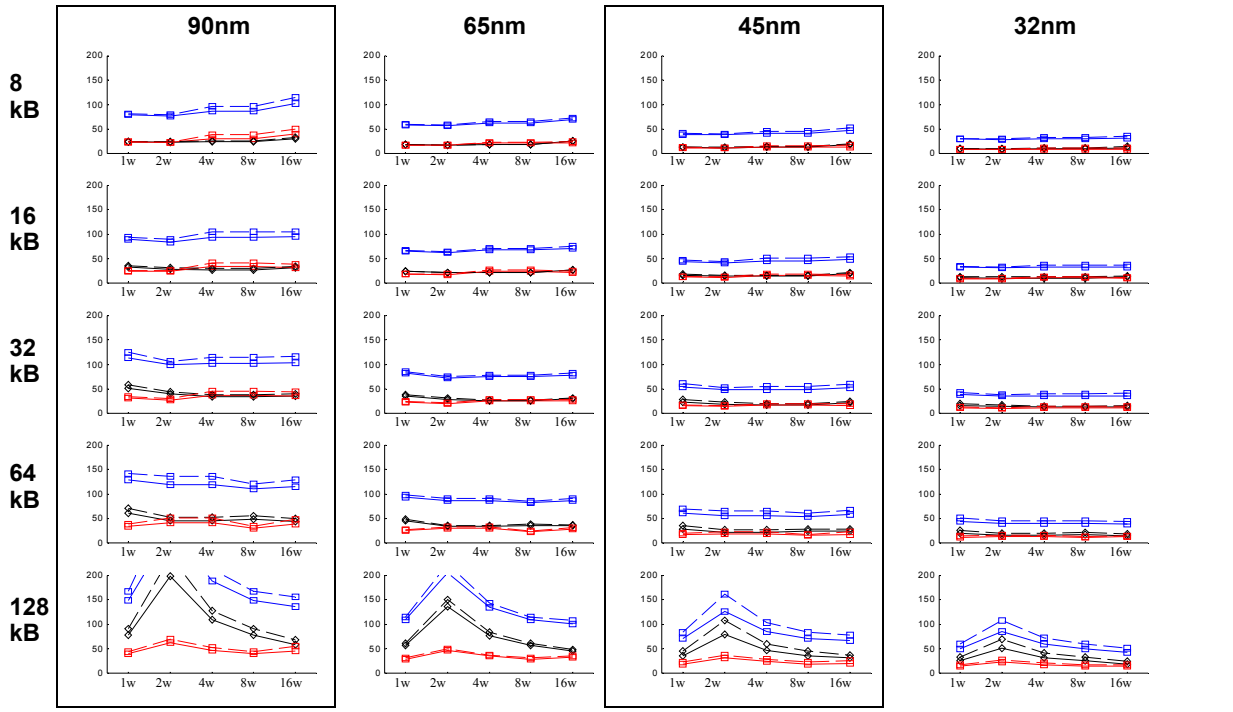


Figure 4-55: Delay components for pipeline stage pipe1B_data . This pipeline stage consists of the data predecoder driver and the wordline front-end. The solid-lines denote default myCACTI numbers that account for multi-layer metal , while the dashed lines denote numbers that were produced with single-layer metal.

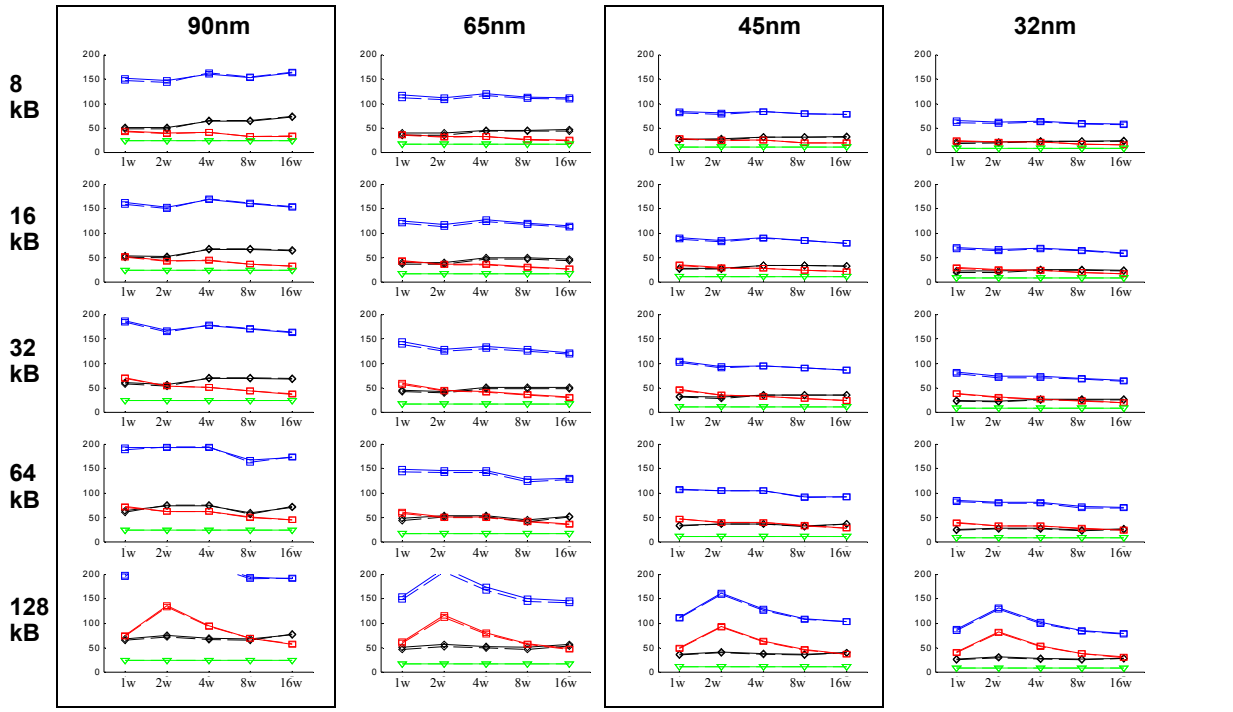


Figure 4-56: Delay components for pipeline stage pipe2A_data . This pipeline stage consists of the wordline drivers, the bitline and the sense amplifier. The solid-lines denote default myCACTI numbers that account for multi-layer metal , while the dashed lines denote numbers that were produced with single-layer metal.

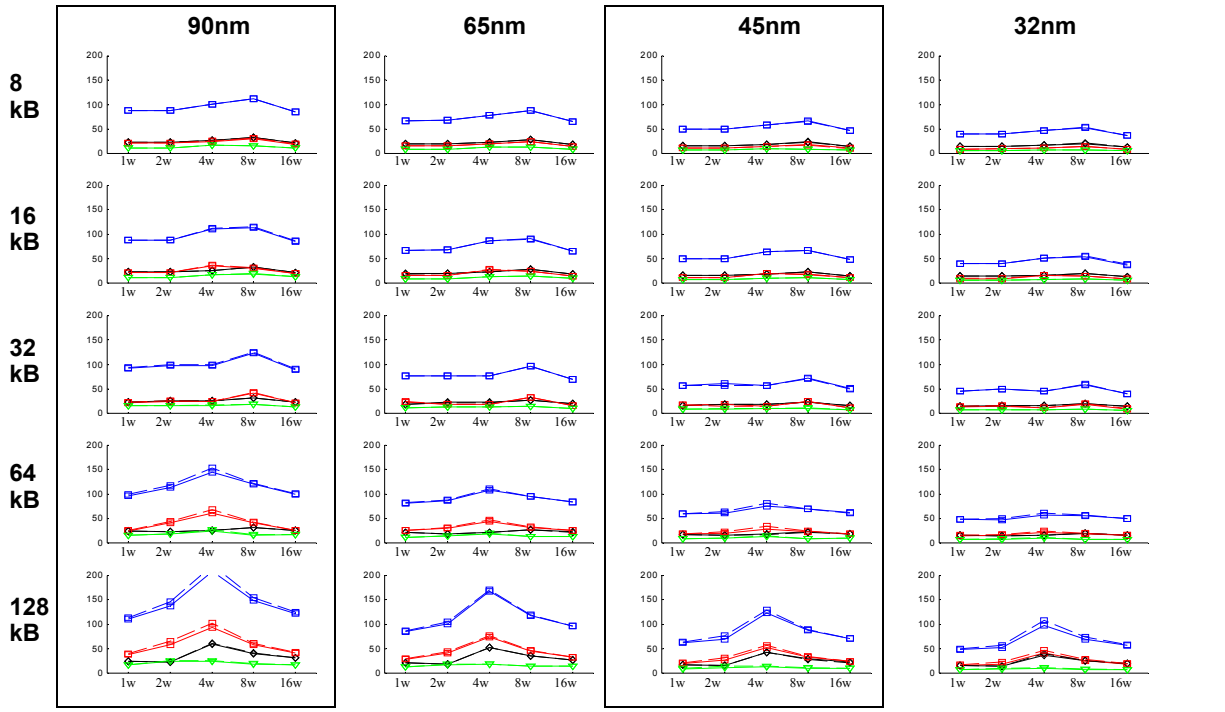


Figure 4-57: Delay components for pipeline stage pipe1A_tag. This pipeline stage consists of tag address buffering, predecoding and the wordline front-end. The solid-lines denote default myCACTI numbers that account for multi-layer metal , while the dashed lines denote numbers that were produced with single-layer metal.

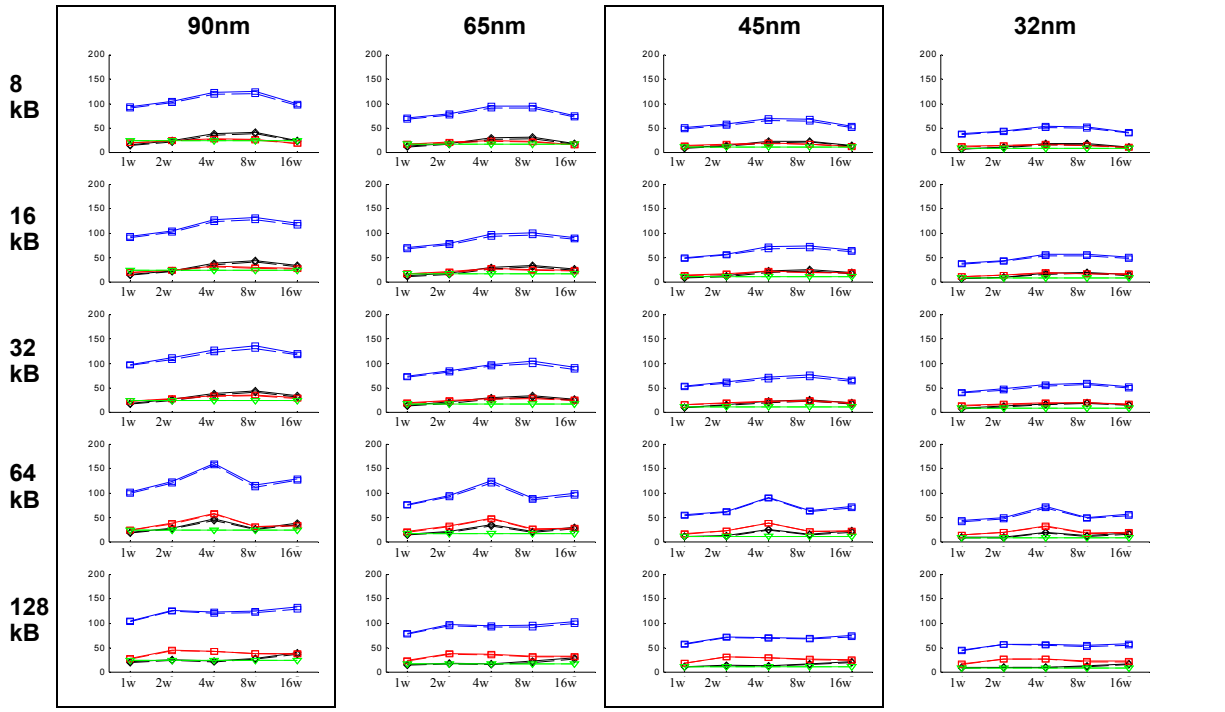


Figure 4-58: Delay components for pipeline stage pipe1B_tag . This pipeline stage consists of the tag wordline drviers, bitline delay and senseamp delay. The solid-lines denote default myCACTI numbers that account for multi-layer metal , while the dashed lines denote numbers that were produced with single-layer metal.

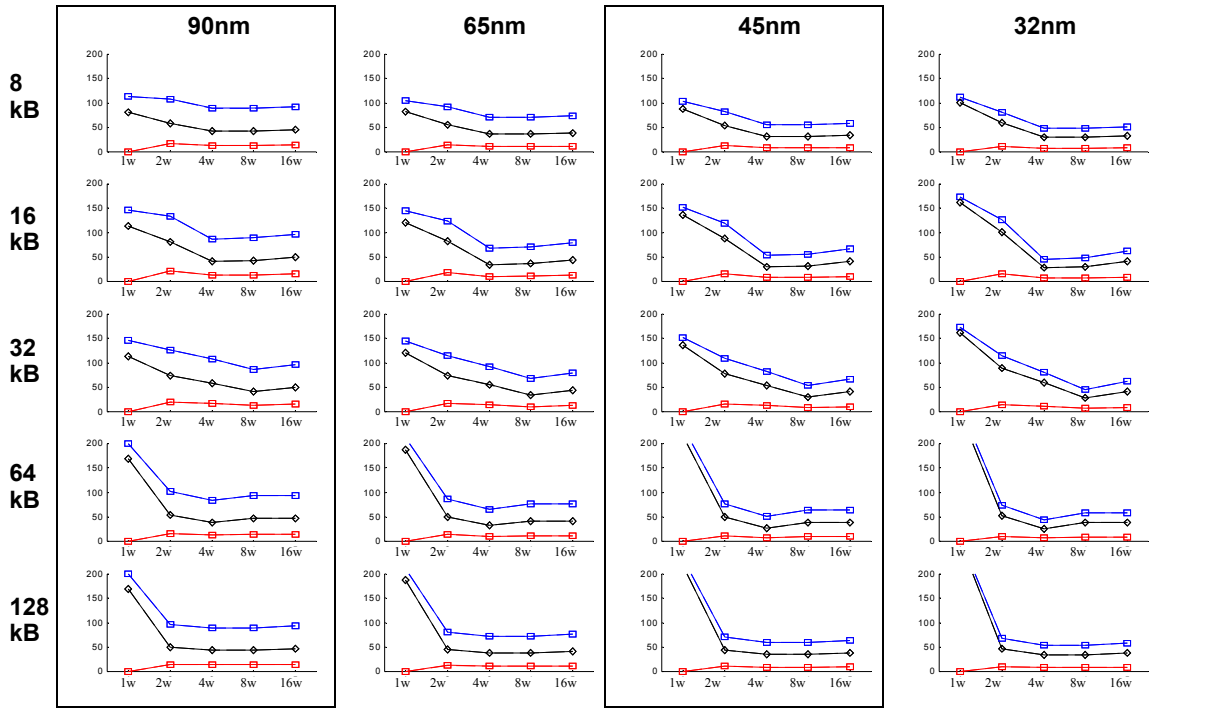


Figure 4-59: Delay components for pipeline stage pipe2A_tag . This pipeline stage consists of enabling the comparator and the actual comparator delay. The solid-lines denote default myCACTI numbers that account for multi-layer metal , while the dashed lines denote numbers that were produced with single-layer metal.

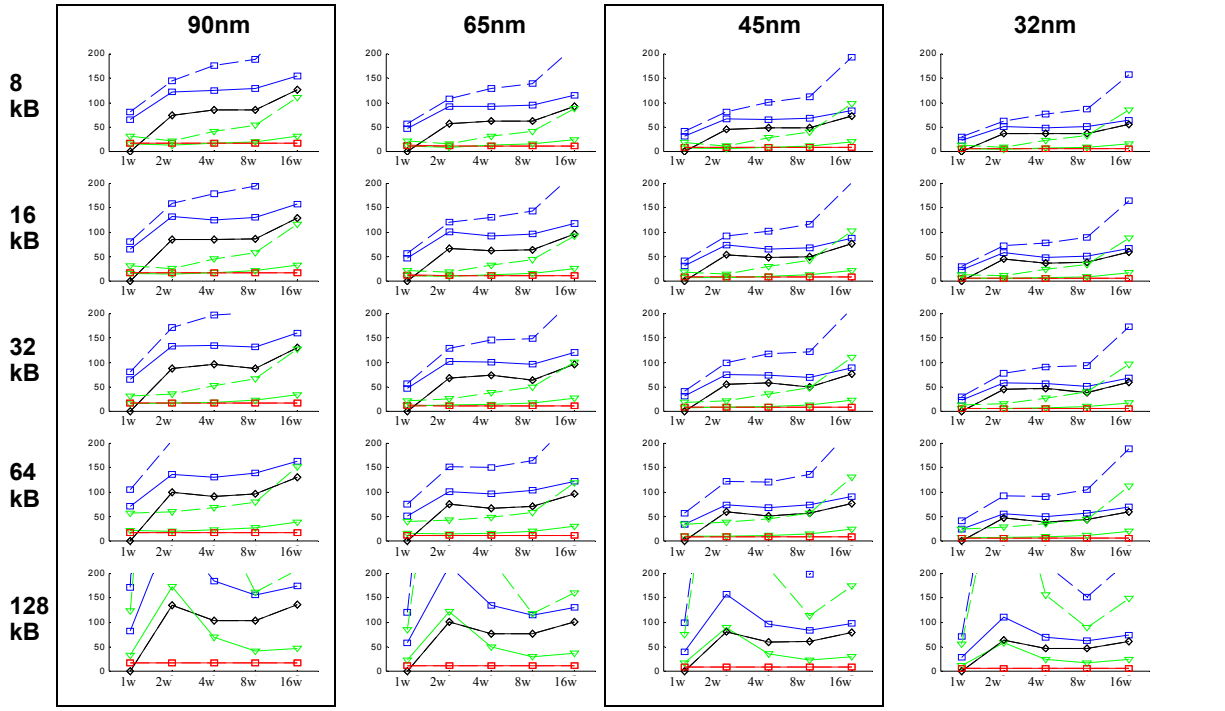


Figure 4-60: Delay components for pipeline stage pipe2B . This pipeline stage consists of driving the output buffer selects and the final data output drive to the cache periphery. The solid-lines denote default myCACTI numbers that account for multi-layer metal , while the dashed lines denote numbers that were produced with single-layer metal.

layer to single-layer actually results in an improvement in delay in both the wordline driver and the bitlines, while in some cases it results in a delay degradation. This is caused by the opposing directions of change in terms of interconnect R and C when going from local to intermediate, to global interconnects (and vice versa). In general, resistance tends to decrease as we go up the BEOL stack (as metals get implemented with larger wires), while capacitance also increases (but not as much). Although the typically better performance of using local interconnects for bitlines and wordlines may be the warranted approach as discussed here, it is not always possible, as there typically is not enough space in the local interconnect layers to accomodate both the bitline and the wordline. Consequently, the BEOL wiring configuration that is used in the multi-layer runs represented a coherent wiring plan that will easily accomodate placement of important signals but at the same time, is close to the optimal in terms of performance.

For pipe2B, the limitation of using a single layer of interconnect is really very significant, as it becomes very difficult to properly drive the data out signal across the entire cache using the local interconnect. Compared to the global interconnect performance, thelocal interconnect will have a slightly smaller C, but a much greater R such that the R dominates the delay and results in very cumbersome delays values. In addition, this is something that will not easily be solved by using a different implementation, as the wiring has to go through a large part of the cache, and the cache area is mostly determined by the cache size (assuming good array efficiencies). The difference in delay typically becomes larger for higher associativities, as more and

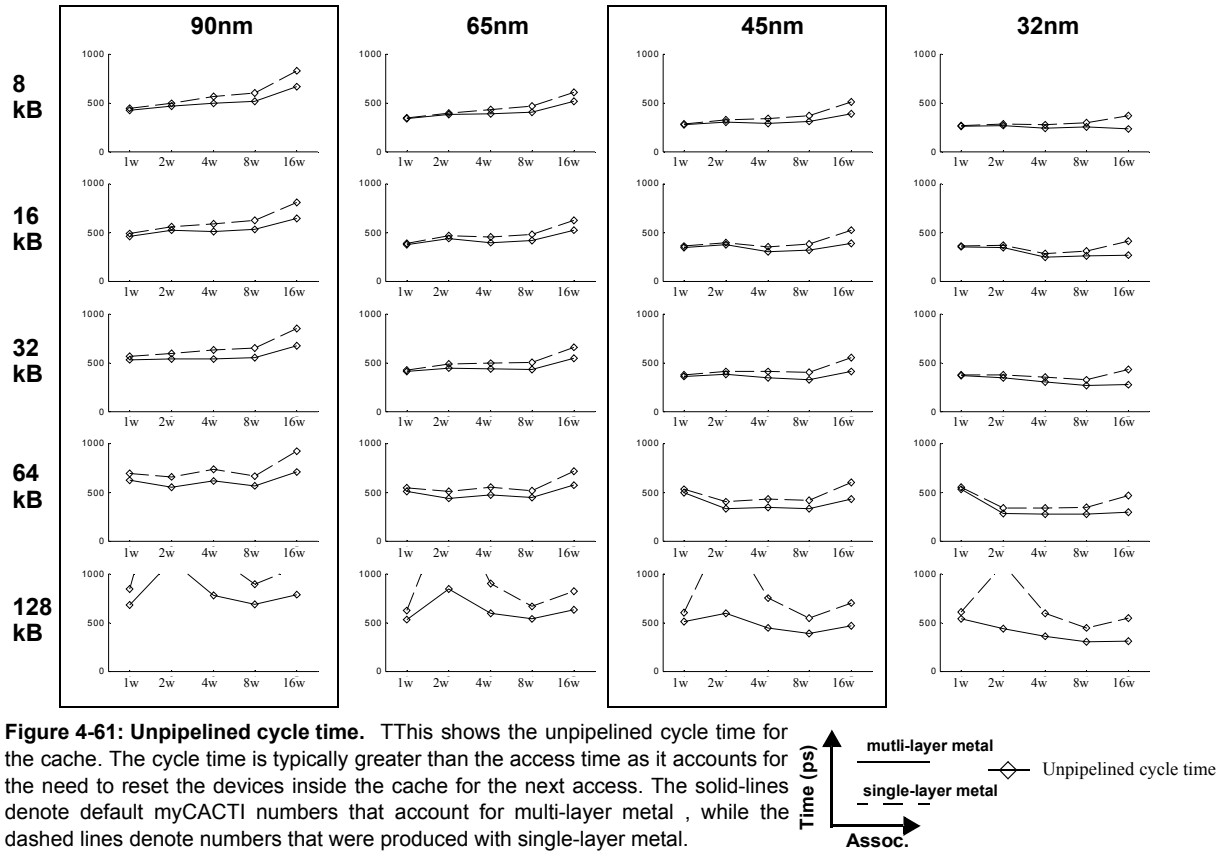


Figure 4-61: Unpipelined cycle time. TThis shows the unpipelined cycle time for the cache. The cycle time is typically greater than the access time as it accounts for the need to reset the devices inside the cache for the next access. The solid-lines denote default myCACTI numbers that account for multi-layer metal , while the dashed lines denote numbers that were produced with single-layer metal.

more drivers are connected to the same wire such that the total wire length that has to be traveled before reaching the cache periphery becomes even greater with increased associativity, degrading the output drive delay even more.

Most other stages (like the address buffers, predecoders, etc.) are also affected (typically degraded) by modeling single-layer metal, but the differences are not big enough to have these stages be considered as the critical path.

Run power results

Figure 4-62 shows the total power and its breakdown into dynamic power (switching power) and static power (dissipated by subthreshold leakage and gate leakage currents) for every point in the design space that was considered (i.e. cache size, associativity and technology node). Each plot shows both the results from myCACTI runs that assume either multi-layer or single-layer metal. Figure 4-63 demonstrates the differences more explicitly by plotting the fractional difference between the two runs normalized to the total power of the default myCACTI result. Figure 4-64 shows detailed power breakdowns of some representative design points in order to explore in detail the causes of the differences. In addition, the results from the two runs are placed side by side for easy comparison. Lastly, Figure 4-65 identifies how much power is being dissipated by major cache blocks, and again shows the results from the two runs side by side for easy comparison.

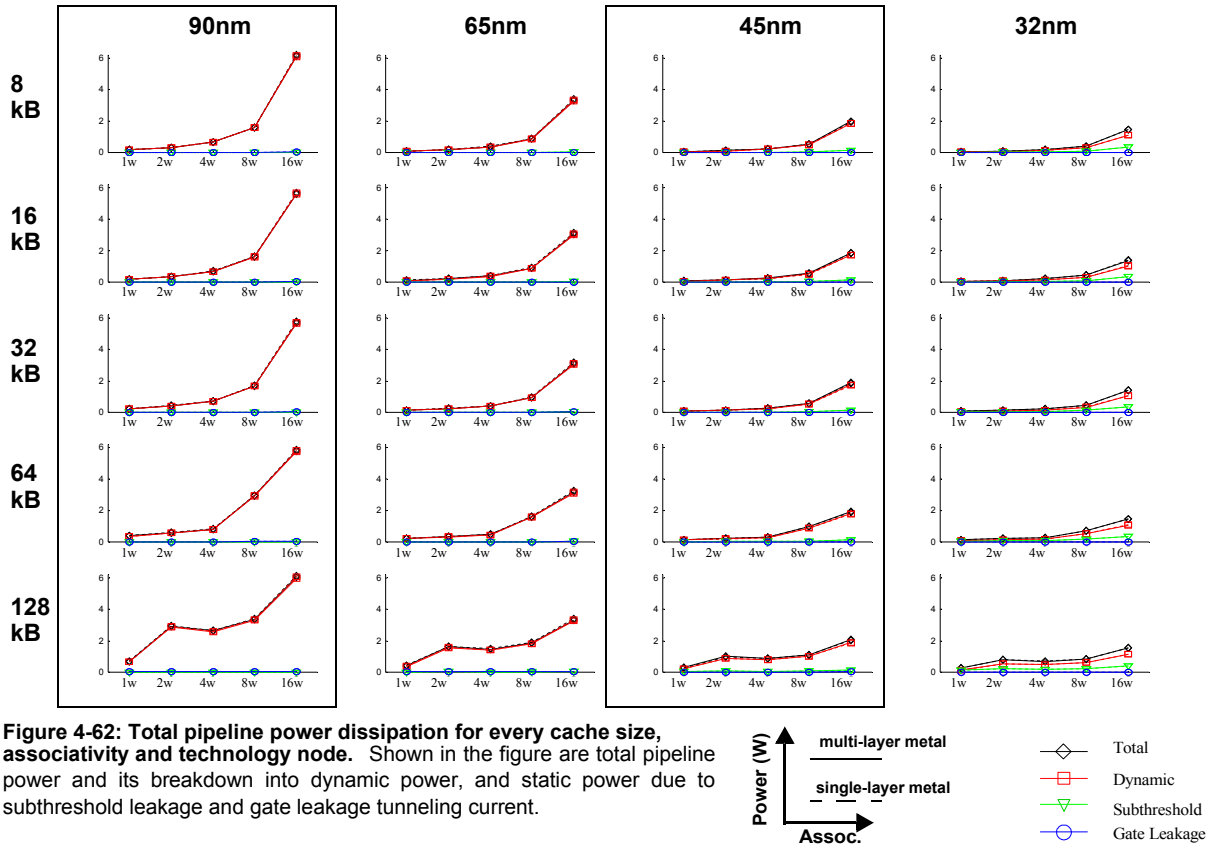


Figure 4-62: Total pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

From the figures, we see that the power differences when modeling either multi-layer or single-layer metal are almost negligible for all configurations and technology nodes. The maximum difference is only about 3%, with the configurations typically having a peak difference with the direct-mapped configuration, then decreasing with increased associativity. Again, this makes sense as total power increases with associativity, but the additional power due to having single-layer metals are not directly affected

This is a surprising result at first, as one would normally expect a larger difference by looking at diagrams of sample BEOLs (or photomicrographs or electron microscope pictures) that the higher-level metal have significantly larger coss-sectional areas compared to the lower-level metals. As such, one would normally expect a large increase in capacitance. This is just part of the story, however, as the spacing between wires actually goes down as you go up the BEOL stack, such that coupling distances are longer. The combination of larger coupling area but longer coupling distance, in the case of the particular BEOL-stack that we model, resulted in only minor capacitance difference between the different layers, explaining the minimal increase in power between modeling multi-layer and single-layer interconnect.

Conclusion

We have shown that modeling interconnect as either single or multi-layer results in negligible differences in power, but very significant differences in delay. For delays, the value can either be

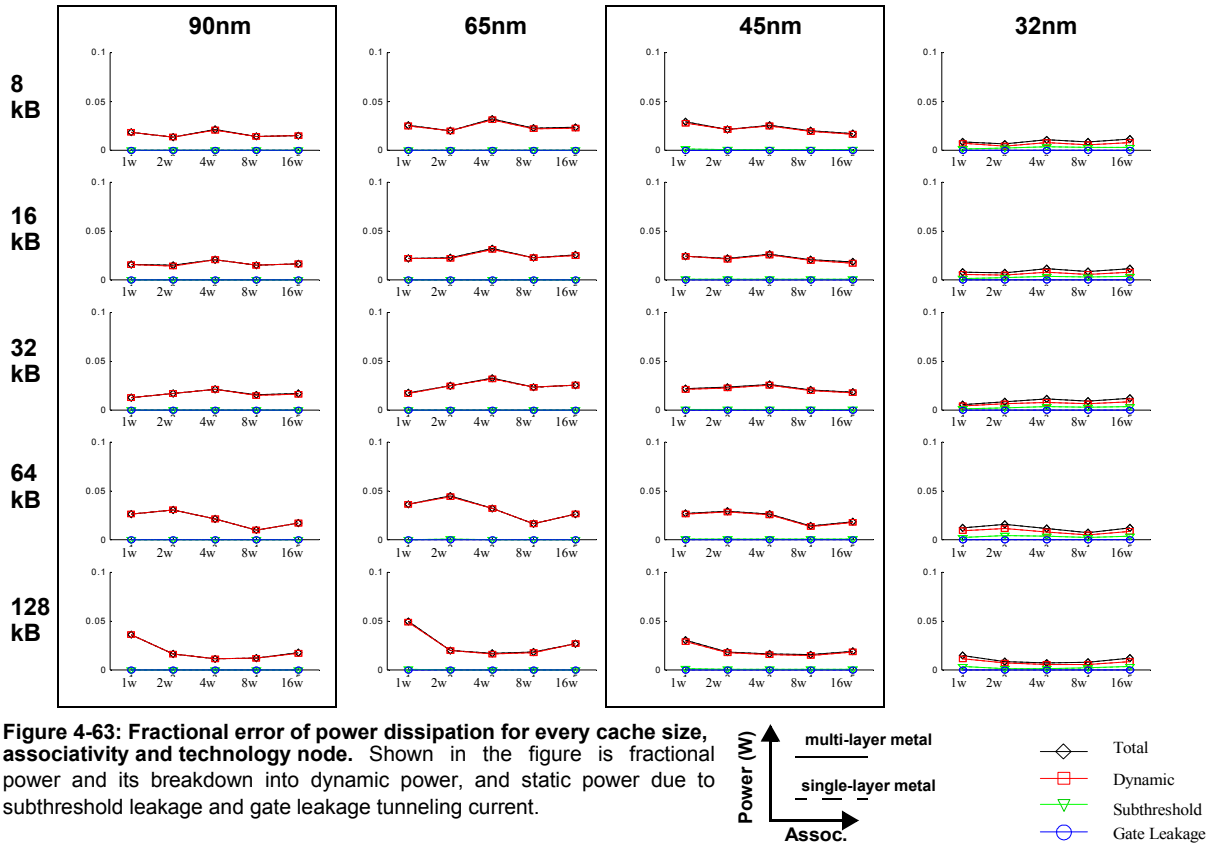


Figure 4-63: Fractional error of power dissipation for every cache size, associativity and technology node. Shown in the figure is fractional power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

overestimated or underestimated by single-layer metal modeling depending on the particular case. Given that resistance tends to decrease while capacitance tends to increase (at a slower rate) as we go higher in the metal

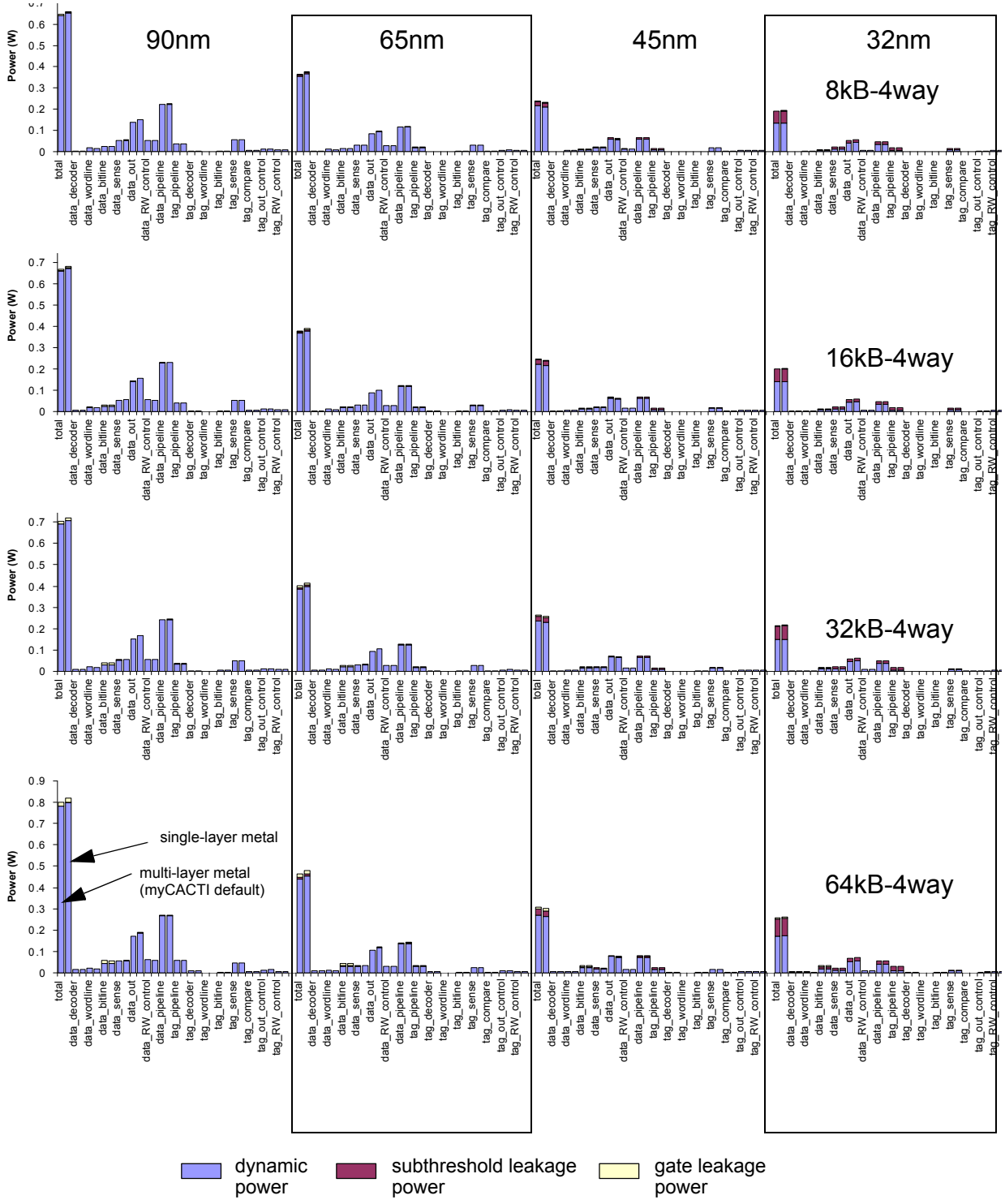


Figure 4-64: Detailed cache power breakdown comparison. Detailed power breakdown for four different cache configurations showing the amount of dynamic, subthreshold leakage and gate leakage power being consumed by different parts of the cache. Each plot shows four sets of values corresponding to different technology nodes. (Note that all three plots use the same scale). The two results are also placed side by side for direct comparison.

stack, the delay change to a particular cache block will depend on which of the resistance or capacitance is dominant. For example, in the bitlines where the small widths of the access and driver transistors result in a large effective driving resistance, using a highly resistive local interconnect layer for the bitlines might be acceptable, especially since this also gives you lower capacitive loading. For strong drivers with low effective resistances, though, increasing the wire resistance might cause more delay degradation than increasing the capacitive loading.

In any case, modeling an unrealistic BEOL stack has been shown to give very pessimistic access time numbers, and although additional pessimism does assure that the real produce has better behavior, it may also

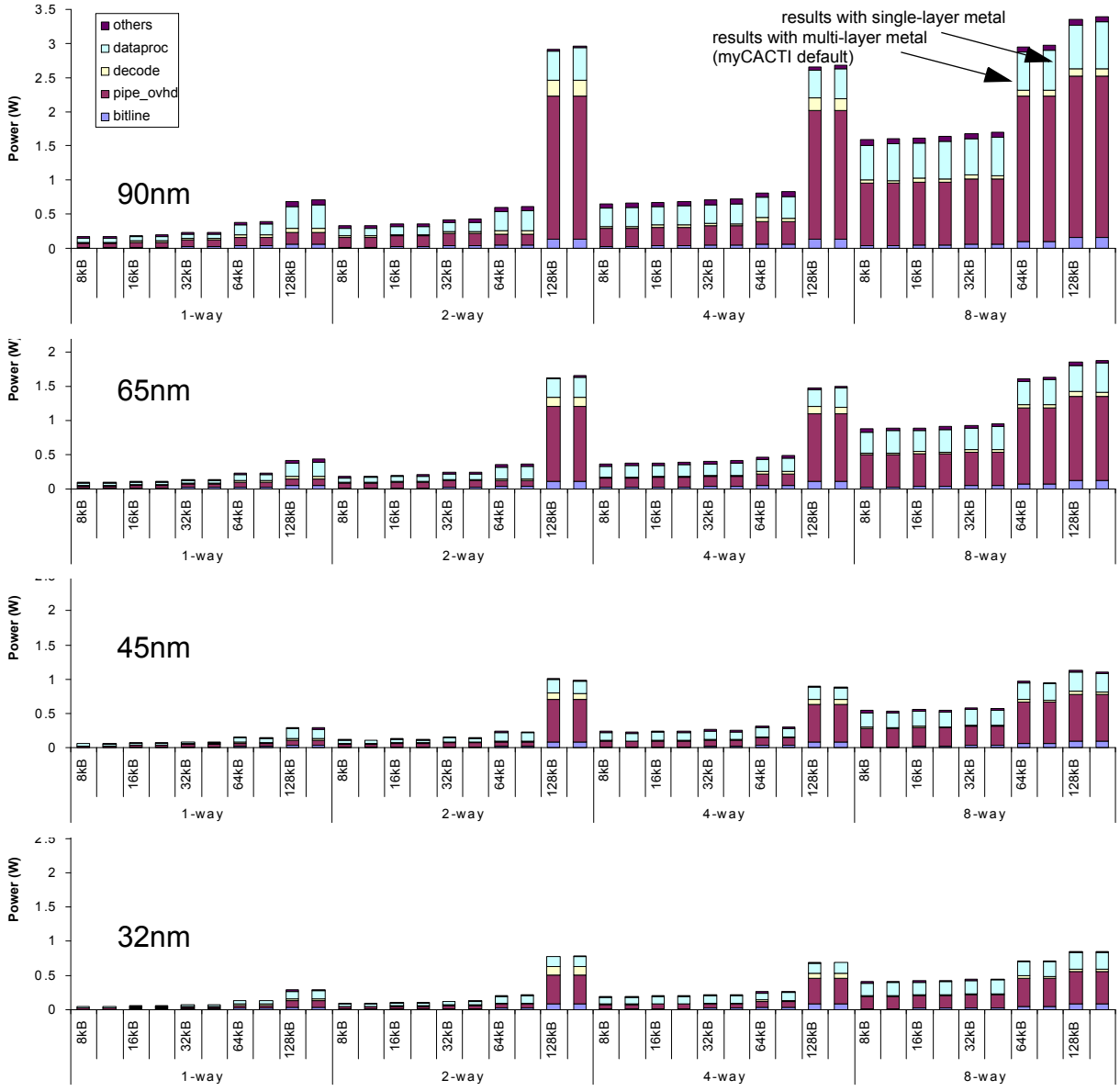


Figure 4-65: Power contributions of different cache parts. Power breakdown showing major cache power contributors for the 65nm and 32nm technology nodes for different cache sizes and associativities. (Note that the y-axis for both plots use the same scale). The two results are also placed side by side for direct comparison.

result in too much overdesign resulting in an inefficient cache. We conclude that it is important to model realistic BEOL stacks by modeling multi-layer interconnects properly.

4.5.6 Gate leakage

Run details

To produce these data, myCACTI was made to do a design space exploration using the default settings of myCACTI to find the optimal implementation for every cache size and associativity for the 90nm technology node. After determining the optimal implementations for each point in the 90nm design space (in terms of the cache parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl), three more sets of runs are performed for each technology node, with each point in the design space utilizing the optimal implementation that was determined for the 90nm node. In essence, the data for 65nm, 45nm and 32nm constitute a sort of "process-shrink" where we take a design in an older process technology and convert it to a newer one. Although the runs for 65nm and below will not show the absolute optimal implementation possible, this method was chosen to provide a simple but valid comparison between different technology nodes. Implementing the same cache configuration (i.e. same cache size and associativity) using two different implementations for different technology nodes would make the comparisons between different cache components very difficult and generalizations will be hard to make.

After generating a complete set of data using myCACTI default settings, the second set of data are generated by setting the gate leakage numbers from the first set to zero.

Run power results

Figure 4-66 shows the total power and its breakdown into dynamic power (switching power) and static power (dissipated by subthreshold leakage and gate leakage currents) for every point in the design space that was considered (i.e. cache size, associativity and technology node). Each plot shows both the results from myCACTI runs that assume the presence of gate leakage, or ignores it. Figure 4-67 demonstrates the differences more explicitly by plotting the fractional difference between the two runs normalized to the total power of the default myCACTI result.

From the comparison plots, we can see that the total power difference when ignoring gate leakage is typically only from 5% to 1%. This power difference monotonically decreases with increasing associativity, as more and more dynamic power is dissipated, reducing the fractional contribution of gate leakage. Lastly, this error is mostly an offset type of error.

Conclusion

We observe that accounting for gate leakage results in a marginal power difference, and this difference is reduced even more with increasing associativity.

This is a largely unexpected result, as gate leakage is widely expected to contribute more and more power with each technology generation. We discuss this interesting finding in a more detailed section later that

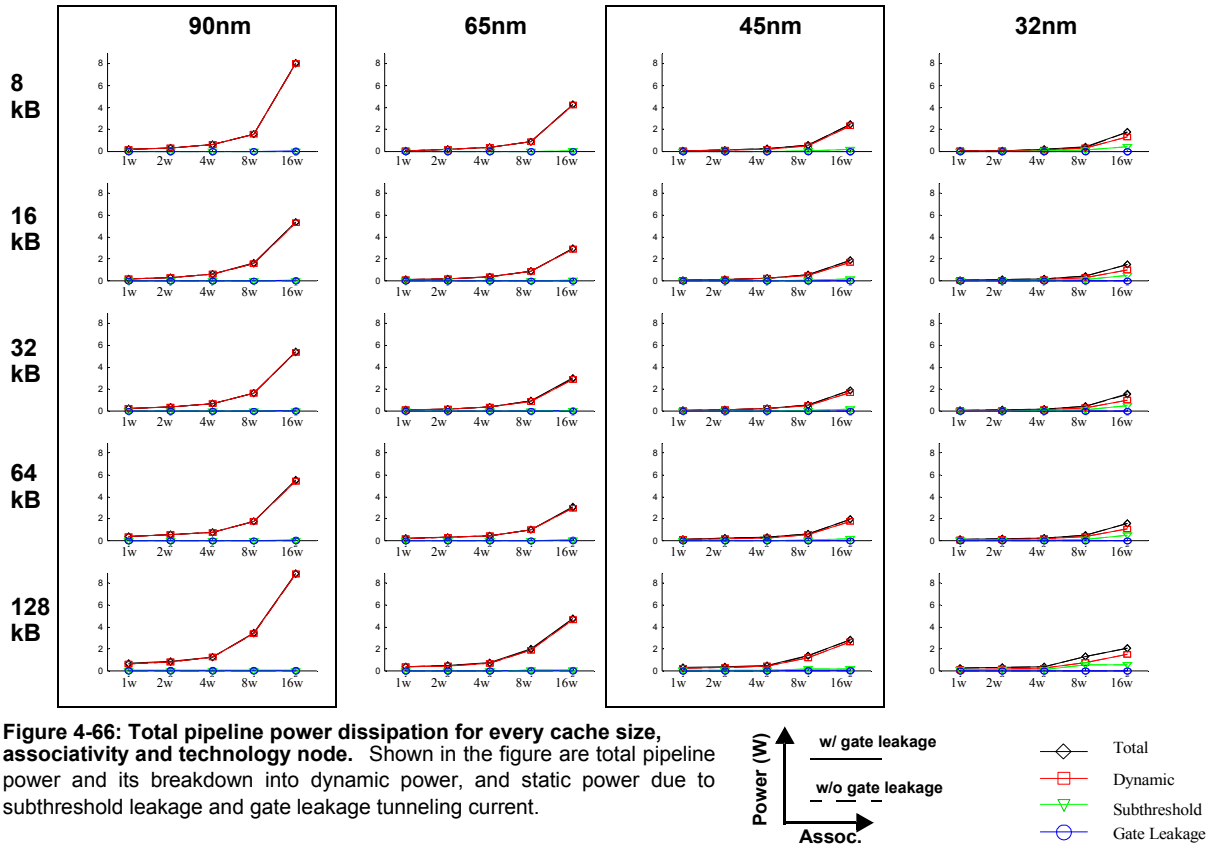


Figure 4-66: Total pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

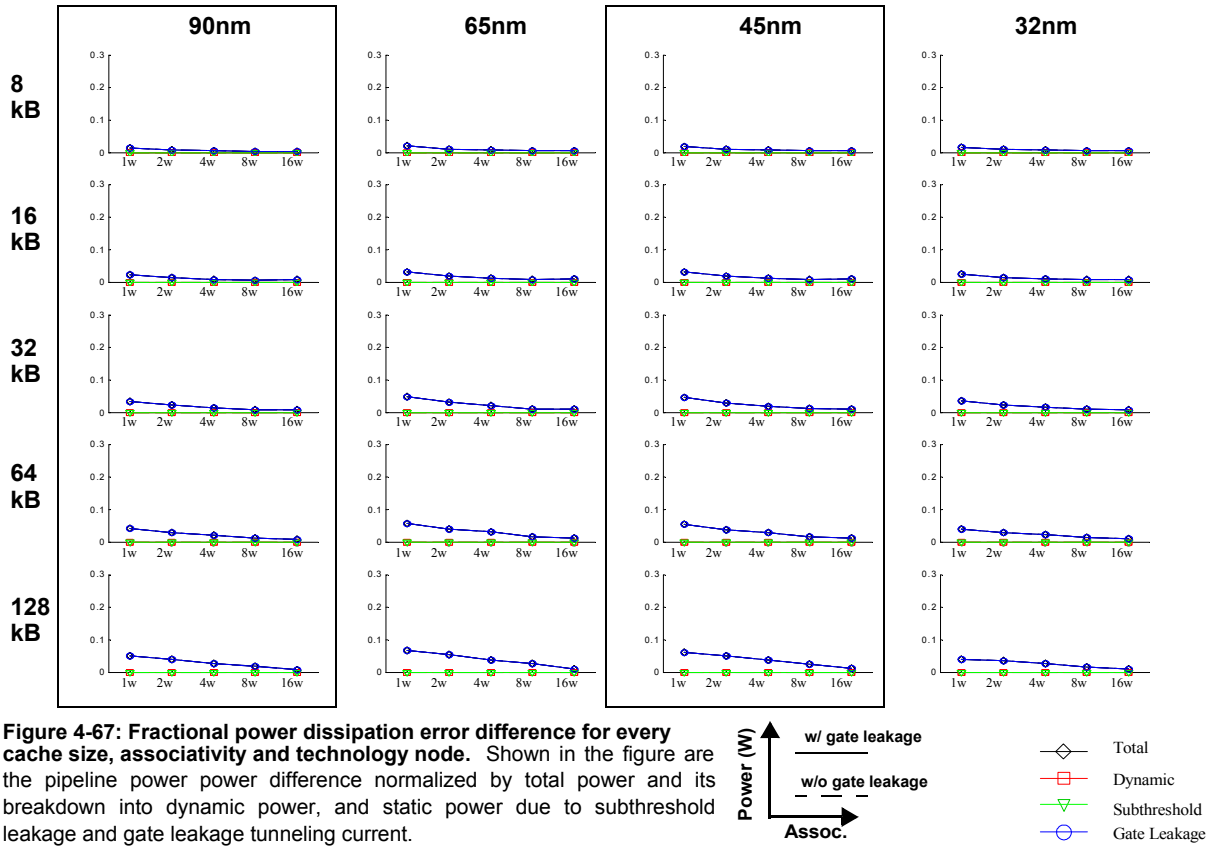


Figure 4-67: Fractional power dissipation error difference for every cache size, associativity and technology node. Shown in the figure are the pipeline power power difference normalized by total power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

explains why gate leakage is not as big of a problem as it was being predicted to be. In essence, the process technology designers have responded to the perceived gate leakage problem and have made various changes

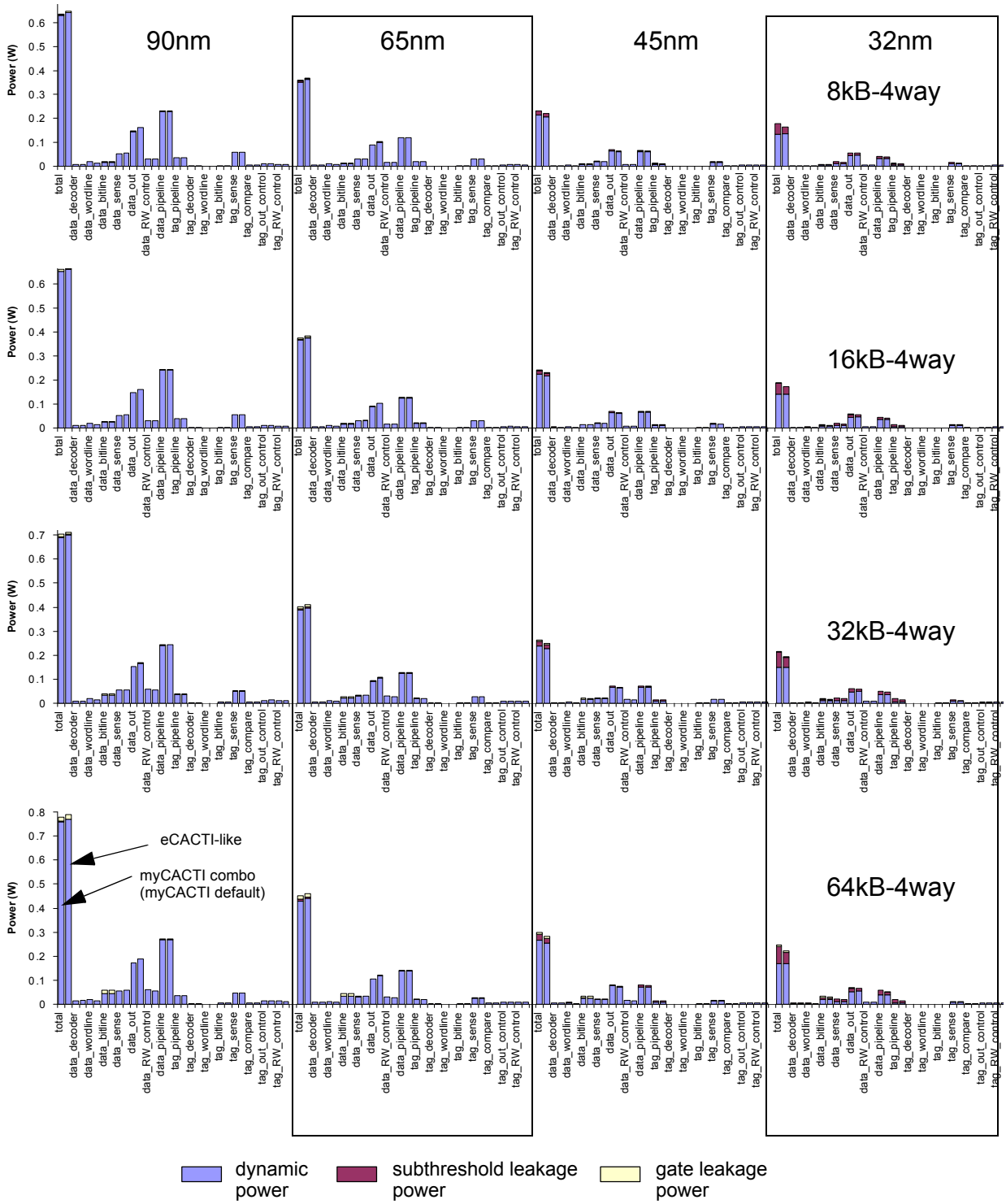


Figure 4-68: Detailed cache power breakdown comparison. Detailed power breakdown for four different cache configurations showing the amount of dynamic, subthreshold leakage and gate leakage power being consumed by different parts of the cache. Each plot shows four sets of values corresponding to different technology nodes. (Note that all three plots use the same scale). The two results are also placed side by side for direct comparison.

to the semiconductor roadmaps such that although gate leakage is still increasing on a per unit area basis, other effects such as decreasing supply voltage and device sizes contribute to actually decreasing the absolute value of gate leakage.

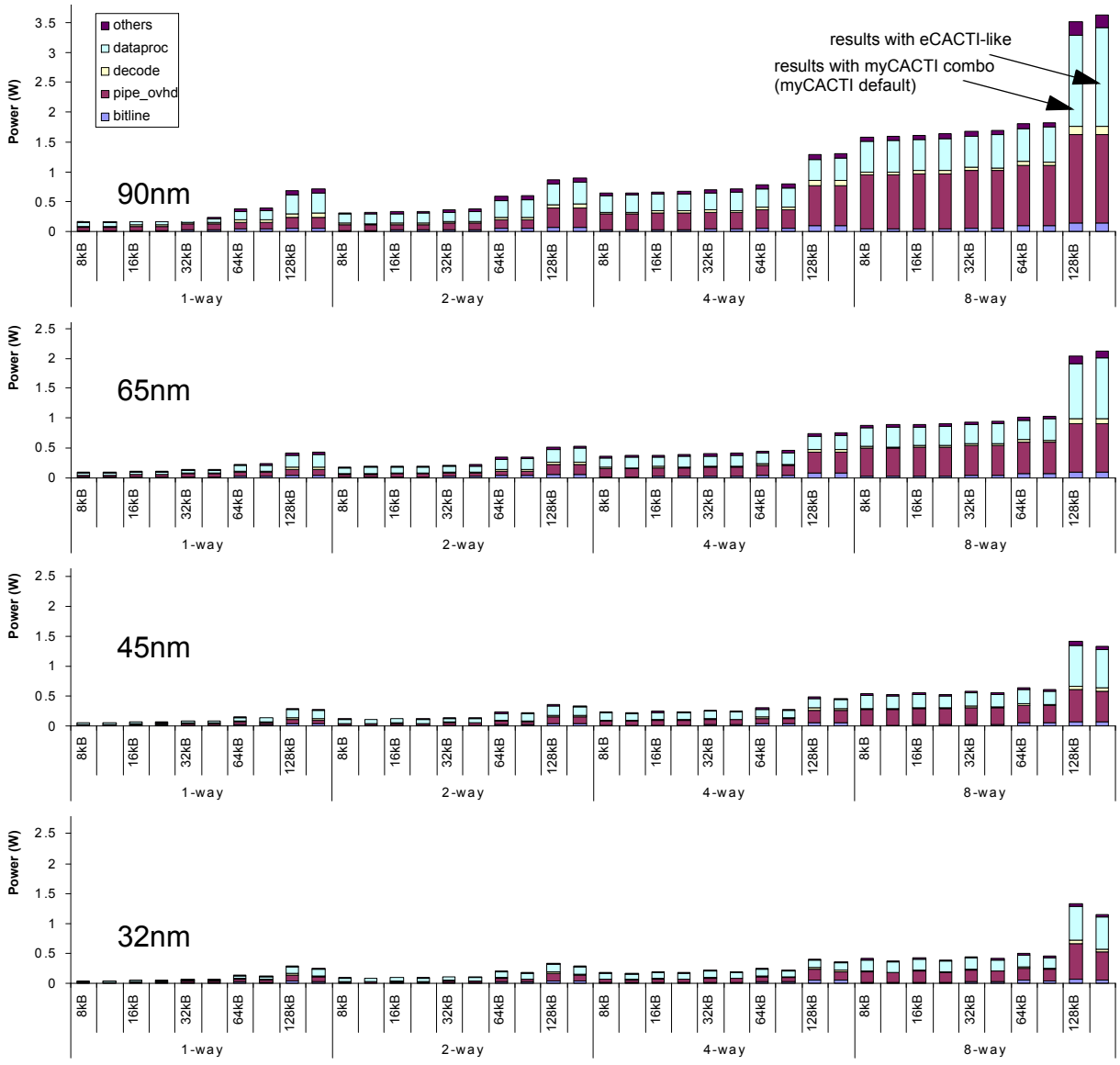


Figure 4-69: Power contributions of different cache parts. Power breakdown showing major cache power contributors for the 65nm and 32nm technology nodes for different cache sizes and associativities. (Note that the y-axis for both plots use the same scale). The two results are also placed side by side for direct comparison.

4.5.7 Static/Dynamic decoding

Run details

To produce these data, myCACTI was made to do a design space exploration using the default settings of myCACTI (in particular, enabling dynamic decoding) to find the optimal implementation for every cache size and associativity for the 90nm technology node. After determining the optimal implementations for each point in the 90nm design space (in terms of the cache parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl), three more sets of runs are performed for each technology node, with each point in the design space utilizing the optimal implementation that was determined for the 90nm node. In essence, the data for 65nm, 45nm and 32nm constitute a sort of "process-shrink" where we take a design in an older process technology and convert it to a newer one. Although the runs for 65nm and below will not show the absolute optimal implementation possible, this method was chosen to provide a simple but valid comparison between different technology nodes. Implementing the same cache configuration (i.e. same cache size and associativity) using two different implementations for different technology nodes would make the comparisons between different cache components very difficult and generalizations will be hard to make.

After generating a complete set of data using myCACTI default settings, the process is repeated, this with each myCACTI run being set to use fully static gates for the data and tag decoders (by using the -static_decode option).

Run timing results

Figure 4-70 shows the delay results for all cache configurations for both runs where results are generated with the assumption of dynamic decoding (the myCACTI default, shown in the plots as the solid lines), and withstatic decoding (shown as dashed lines). This figure shows the resulting optimal clock period for each configuration, along with the pipe delays for all of the stages in the cache (with the exception of pipe1A_data and pipe1B_data -- the initial decode stages -- since these always have significant slack so they have been omitted from the figure for clarity). Figures 4-71 to 4-77 show the delay component for each individual stage. These individual plots provide a breakdown of the delay of each individual stage by showing the component delays of each. Lastly, Figure 4-78 shows the unpipelined cache access time.

From the comparison plots, we see that simulations using dynamic or static decoding have significant differences in delay times, especially for older tech nodes. For the smaller technology nodes, the compare stage starts to become the typical critical path, such that the effective clock period is unaffected by the use of static or dynamic decoding.

Also, although the entire decode chain is affected (i.e. the address buffers, the predecode and the wordline), the effect on the address buffer and predecoder typically do not carry over to the clock period, since these stages generally have enough slack in their time budget to afford incurring additional delays. In effect,

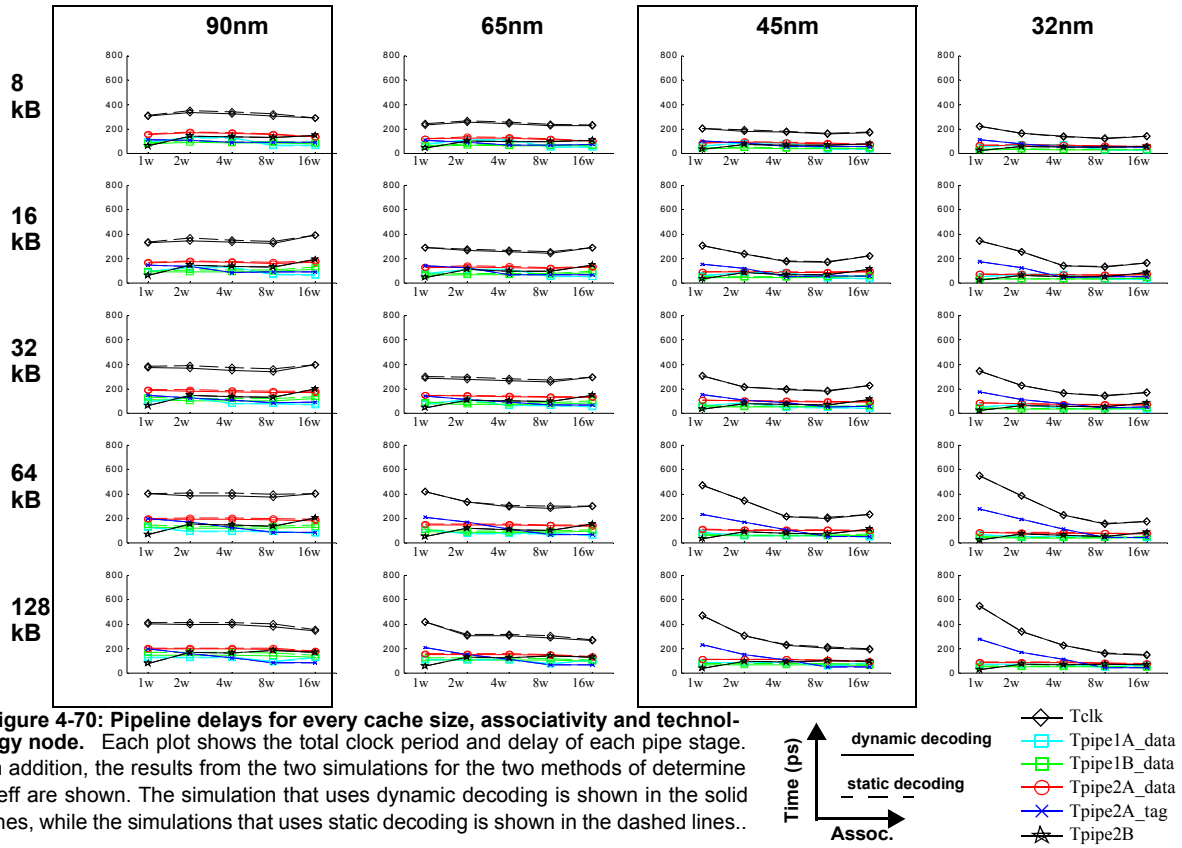


Figure 4-70: Pipeline delays for every cache size, associativity and technology node. Each plot shows the total clock period and delay of each pipe stage. In addition, the results from the two simulations for the two methods of determine Leff are shown. The simulation that uses dynamic decoding is shown in the solid lines, while the simulations that uses static decoding is shown in the dashed lines..

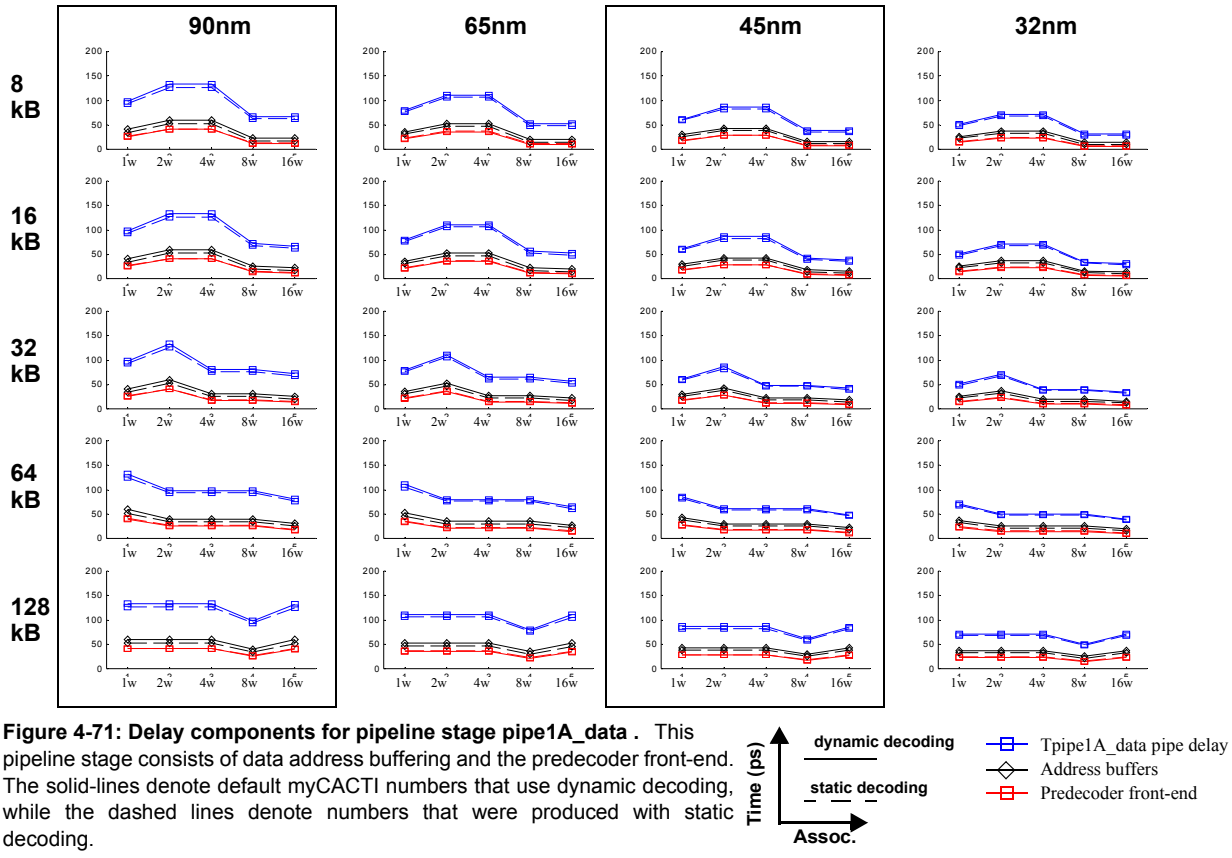


Figure 4-71: Delay components for pipeline stage pipe1A_data . This pipeline stage consists of data address buffering and the predecoder front-end. The solid-lines denote default myCACTI numbers that use dynamic decoding, while the dashed lines denote numbers that were produced with static decoding.

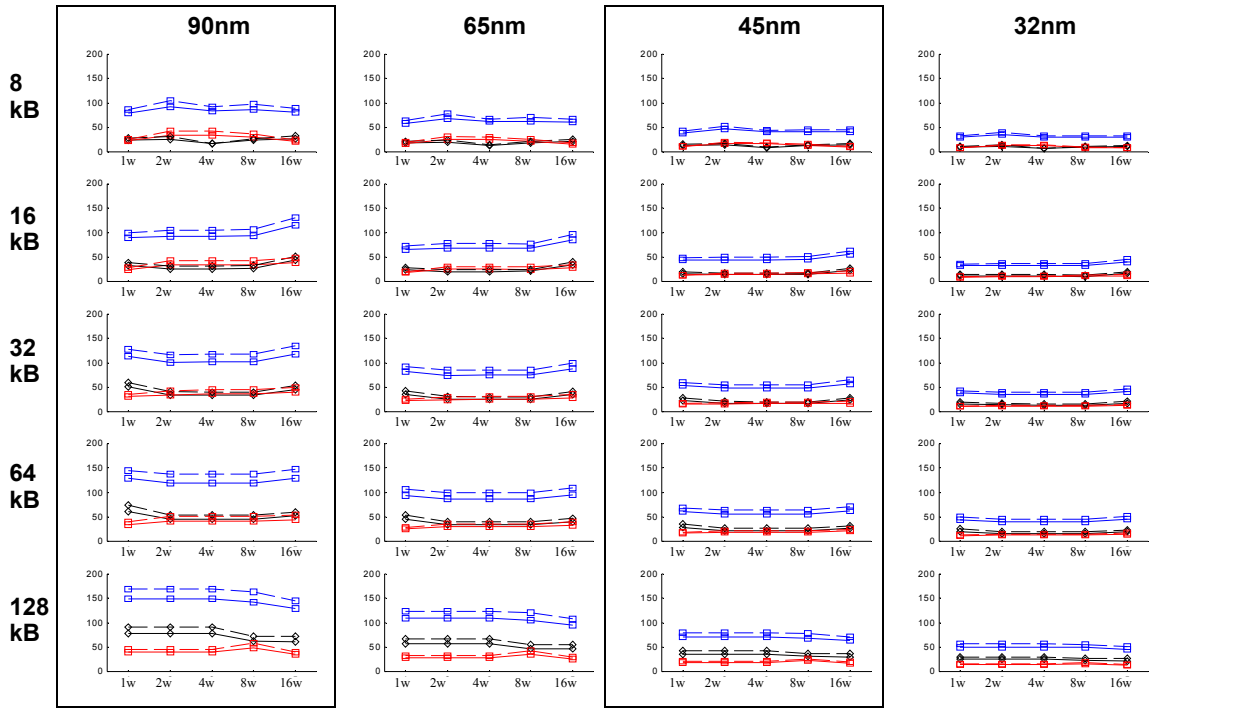


Figure 4-72: Delay components for pipeline stage pipe1B_data . This pipeline stage consists of the data predecoder driver and the wordline front-end. The solid-lines denote default myCACTI numbers that use dynamic decoding, while the dashed lines denote numbers that were produced with static decoding.

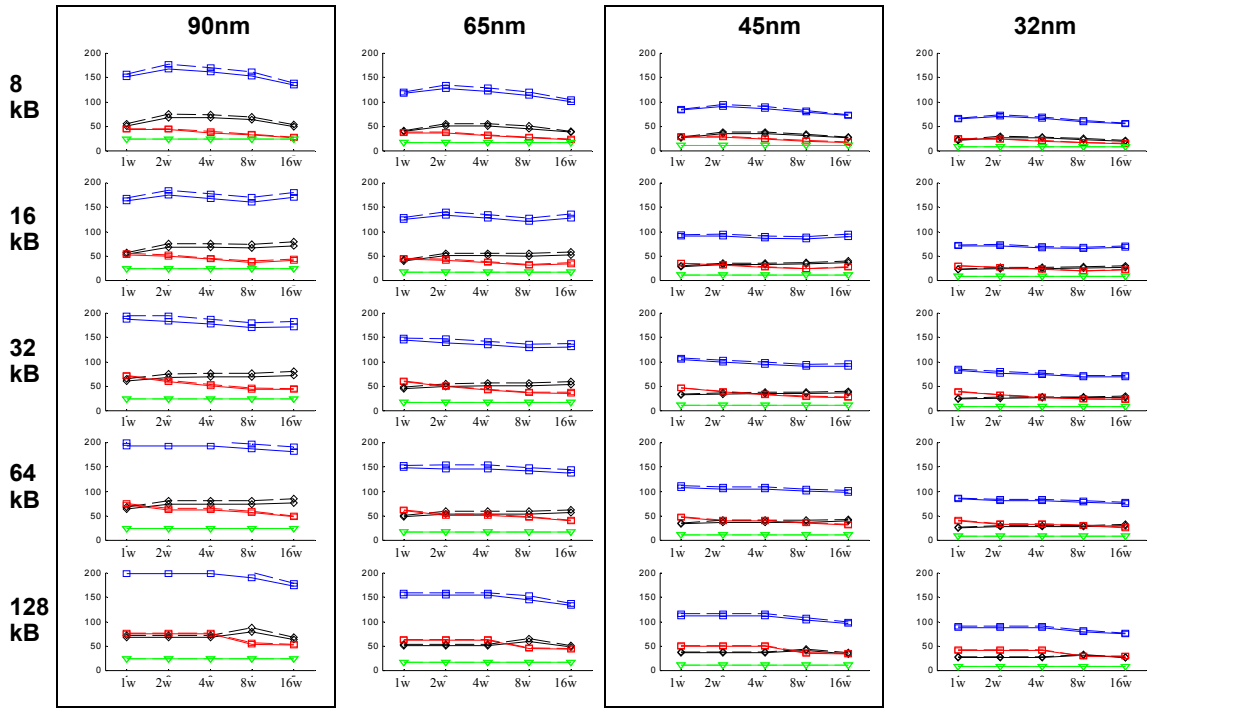


Figure 4-73: Delay components for pipeline stage pipe2A_data . This pipeline stage consists of the wordline drivers, the bitline and the sense amplifier. The solid-lines denote default myCACTI numbers that use dynamic decoding, while the dashed lines denote numbers that were produced with static decoding.

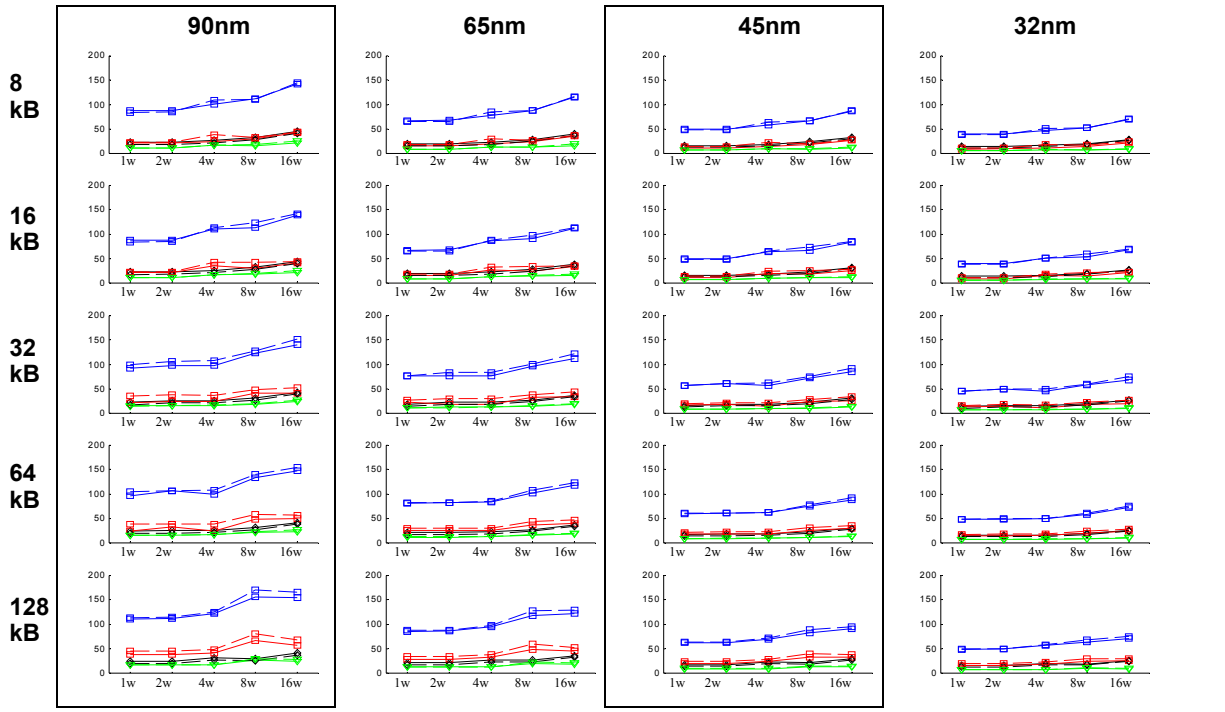


Figure 4-74: Delay components for pipeline stage pipe1A_tag. This pipeline stage consists of tag address buffering, predecoding and the wordline front-end. The solid-lines denote default myCACTI numbers that use dynamic decoding, while the dashed lines denote numbers that were produced with static decoding.

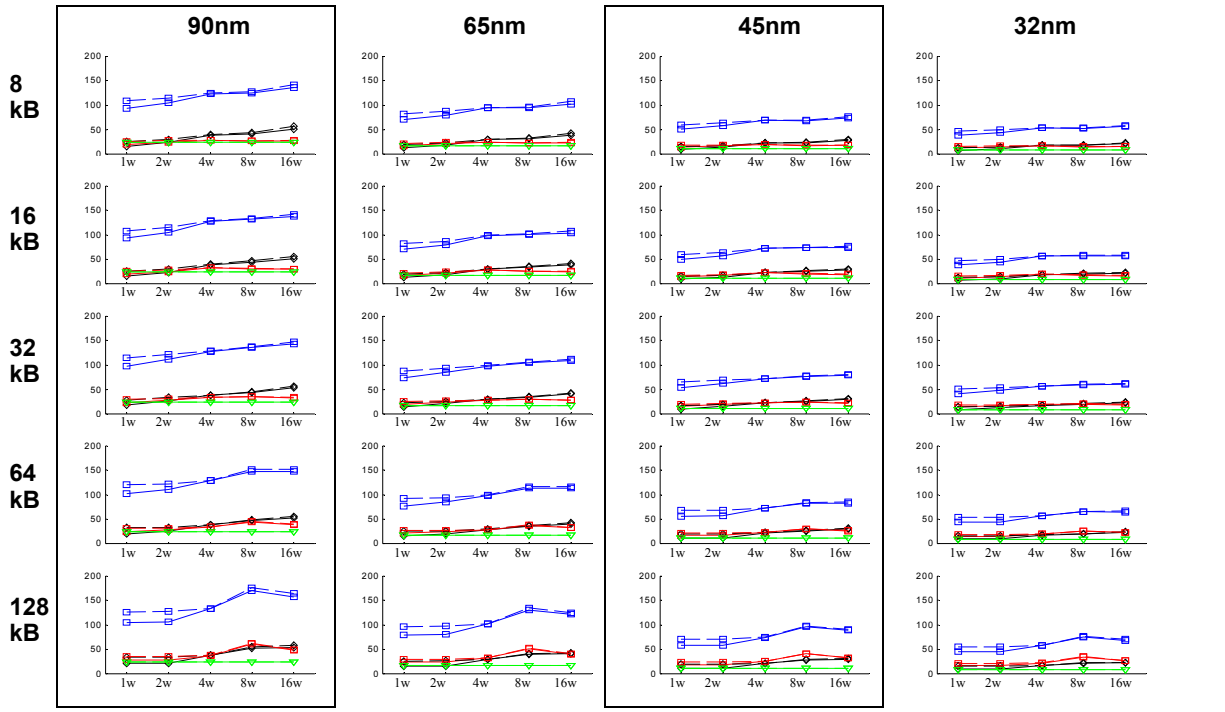


Figure 4-75: Delay components for pipeline stage pipe1B_tag . This pipeline stage consists of the tag wordline drviers, bitline delay and senseamp delay. The solid-lines denote default myCACTI numbers that use dynamic decoding, while the dashed lines denote numbers that were produced with static decoding.

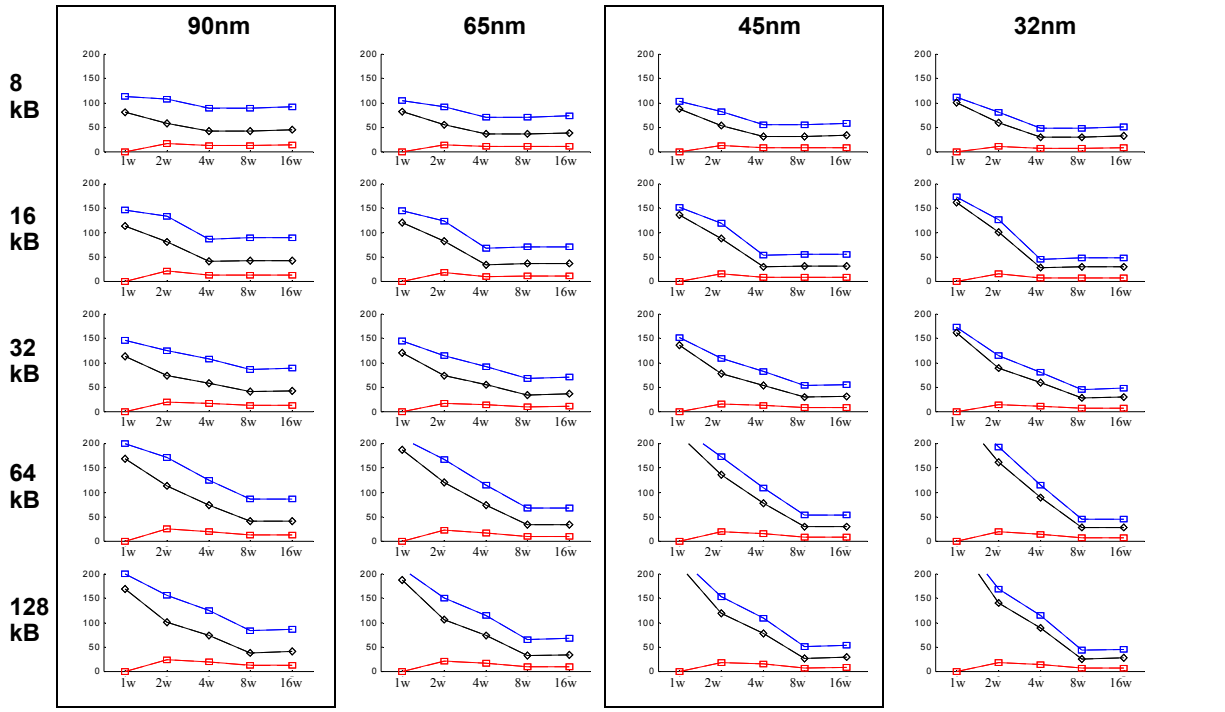


Figure 4-76: Delay components for pipeline stage pipe2A_tag . This pipeline stage consists of enabling the comparator and the actual comparator delay. The solid-lines denote default myCACTI numbers that use dynamic decoding, while the dashed lines denote numbers that were produced with static decoding.

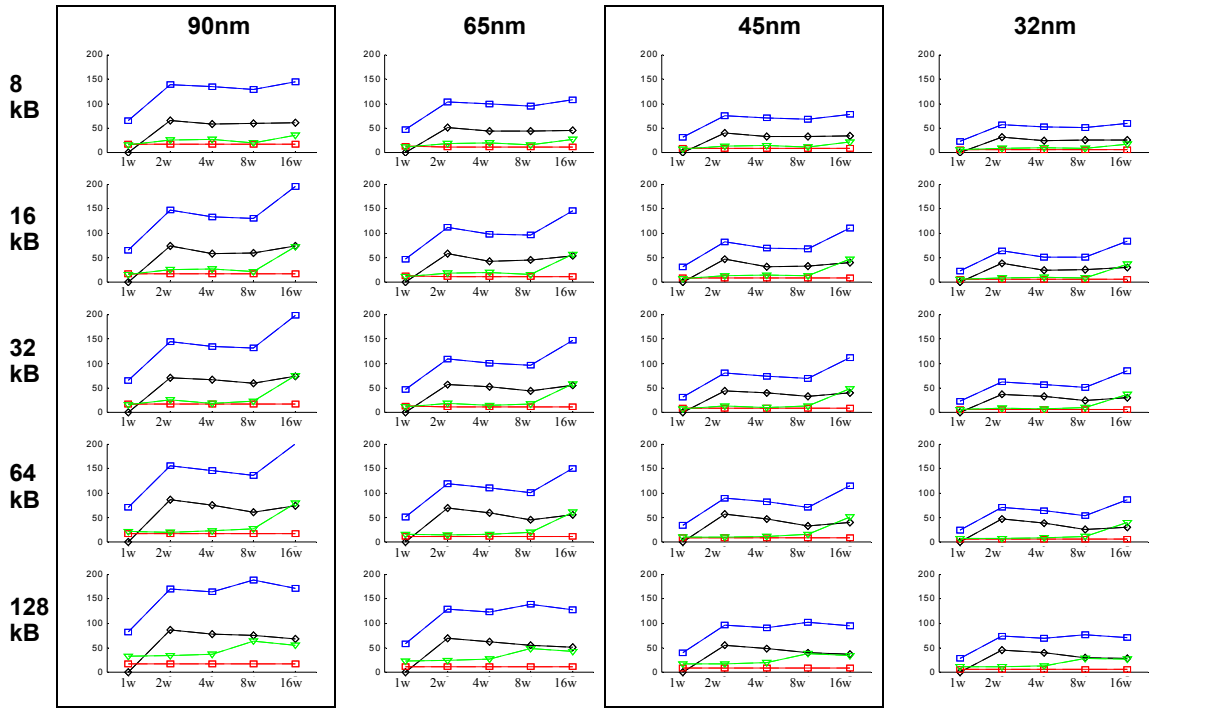


Figure 4-77: Delay components for pipeline stage pipe2B . This pipeline stage consists of driving the output buffer selects and the final data output drive to the cache periphery. The solid-lines denote default myCACTI numbers that use dynamic decoding, while the dashed lines denote numbers that were produced with static decoding.

only the degradation of the wordline driver potentially carries over to the clock period. This shows that the optimized dynamic DRCMOS logic that we model can drive the wordline much easier than static drivers.

One possible implementation that this suggests is the use of static decoding for the non-critical paths, and if necessary, continue using dynamic decoding for the wordline drivers. This is the technique used in the Itanium [Weiss2002], where decoding is not in the critical path and hence, is implemented with static logic -- minimizing the complications introduced by dynamic logic.

In configurations where the compare stage is in the critical path, the stage delay actually degrades more and more with each generation. This is an artifact of the scaling mechanism used to size the comparator circuit. This has been discussed in earlier sections, but it warrants repeating. The comparator devices where sized using a straight-up ratioing with respect to its original sizing in 0.8um technology. This is a problem since the effective drive resistance of transistors for the predictive SPICE decks that we model do not scale linearly. This is true in general, in that devices in nanometer technologies do not scale the same as pre-submicron technologies. This limitation of the comparator can be easily improved by customizing the sizing of the transistors in order to buy more speed in exchange for a bigger area, but we maintain this modeling to emphasize that linear scaling often produces suboptimal results in nanometer technology.

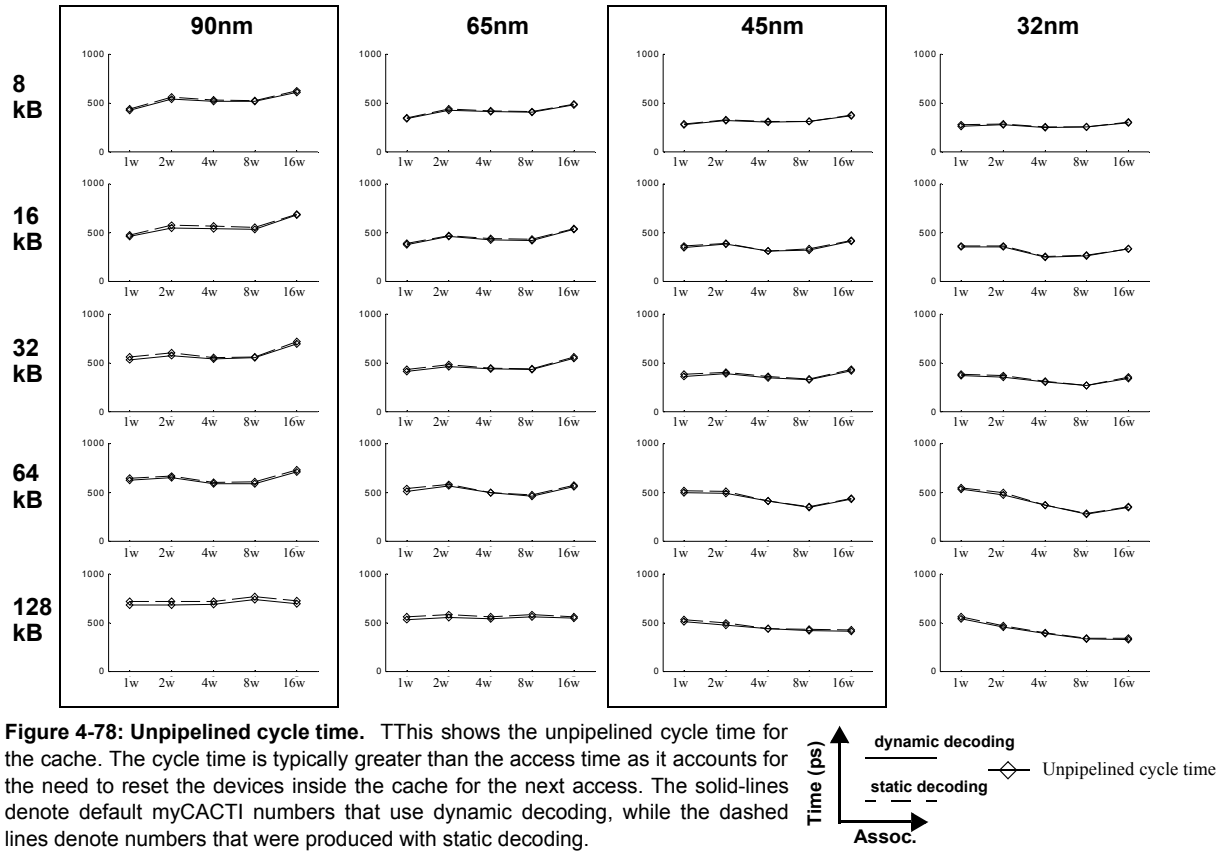


Figure 4-78: Unpipelined cycle time. TThis shows the unpipelined cycle time for the cache. The cycle time is typically greater than the access time as it accounts for the need to reset the devices inside the cache for the next access. The solid-lines denote default myCACTI numbers that use dynamic decoding, while the dashed lines denote numbers that were produced with static decoding.

Run power results

Figure 4-79 shows the total power and its breakdown into dynamic power (switching power) and static power (dissipated by subthreshold leakage and gate leakage currents) for every point in the design space that was considered (i.e. cache size, associativity and technology node). Each plot shows both the results from myCACTI runs that assume either dynamic or static decoding. Figure4-80 demonstrates the differences more explicitly by plotting the fractional difference between the two runs normalized to the total power of the default myCACTI result. Figure 4-81 shows detailed power breakdowns of some representative design points in order to explore in detail the causes of the differences. In addition, the results from the two runs are placed side by side for easy comparison. Lastly, Figure 4-82 identifies how much power is being dissipated by major cache blocks, and again shows the results from the two runs side by side for easy comparison.

From the figures, we see that the power differences when modeling either dynamic or static decoding are almost negligible for all configurations and technology nodes. This is surprising at first, since dynamic and static decoding result in very different circuitry and transistor sizing. But if we account for the fact that dynamic power should change only slightly even with the higher capacitances in static decode circuits because only a small fraction of the decode circuit activates. Since they already constitute a small fraction of total power compared to the bitlines, the effect is minimized even further.

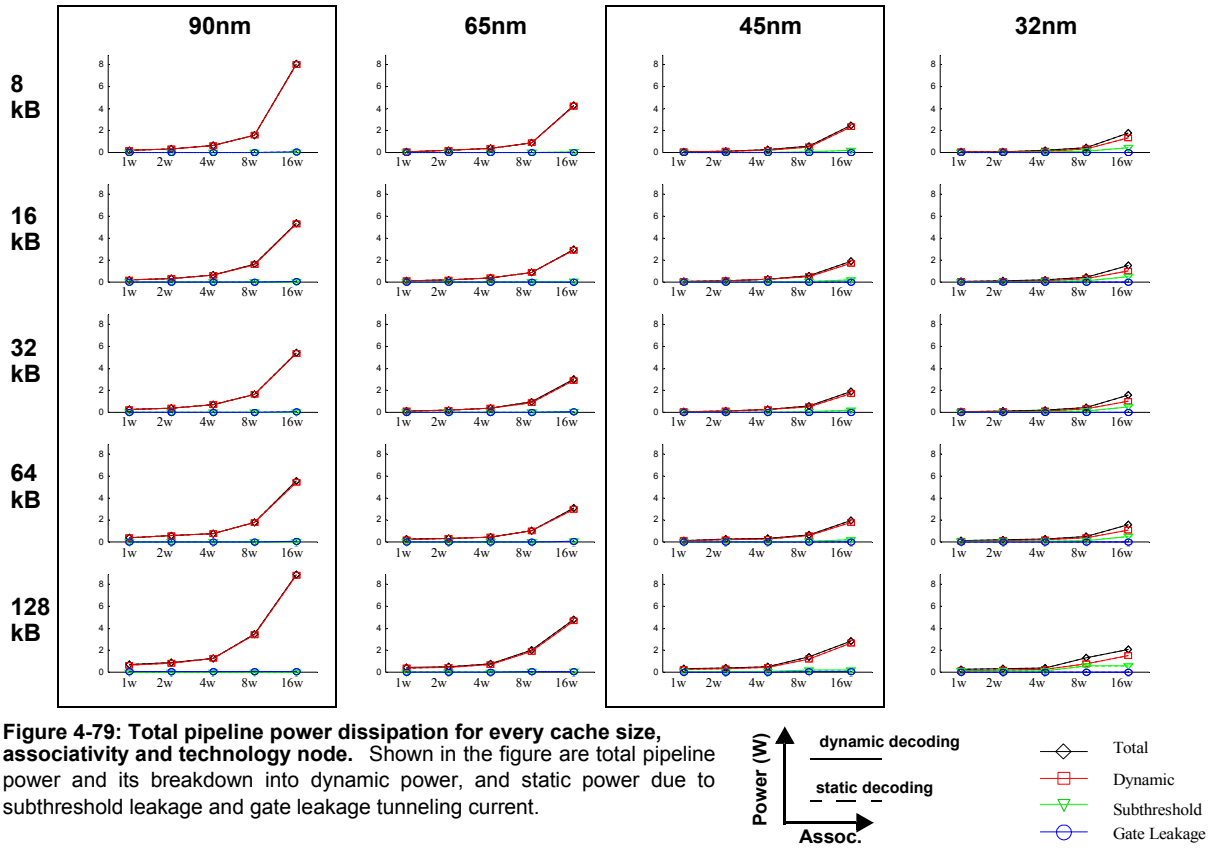


Figure 4-79: Total pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

For static decoding, we would at first expect power to increase because of the increased sizes of the gates in the critical path, but implementation of DRCMOS actually offsets some of this savings because of the presence of its reset devices.

Looking at the error graphs, the observation that there are minimal power differences between static and dynamic decoding (or at least this implementation of dynamic decoding) is really emphasized.

Looking at the detailed power breakdown of a cache, we see that the decoders do not dissipate a significant amount of power. This is a direct result of subarraying -- where a large part of the entire decoder is left inactive with increasing subarrays. In addition to having fewer and fewer rows, the power dissipation of the decoders really become small. Of course, too many subarrays will simply shift the burden into other blocks, so subarraying does have its limits. In this case, more subarrays may tend to increase the data output power (for one), and as shown here, the data_out power is a significant component of total power.

Another observation regarding DRCMOS decoders is that although the forward path through this circuit should have smaller power (both dynamic and static) because of its smaller total width, this is offset by the additional power dissipated in the reset circuitry, such that the resulting power is roughly the same as a static gate.

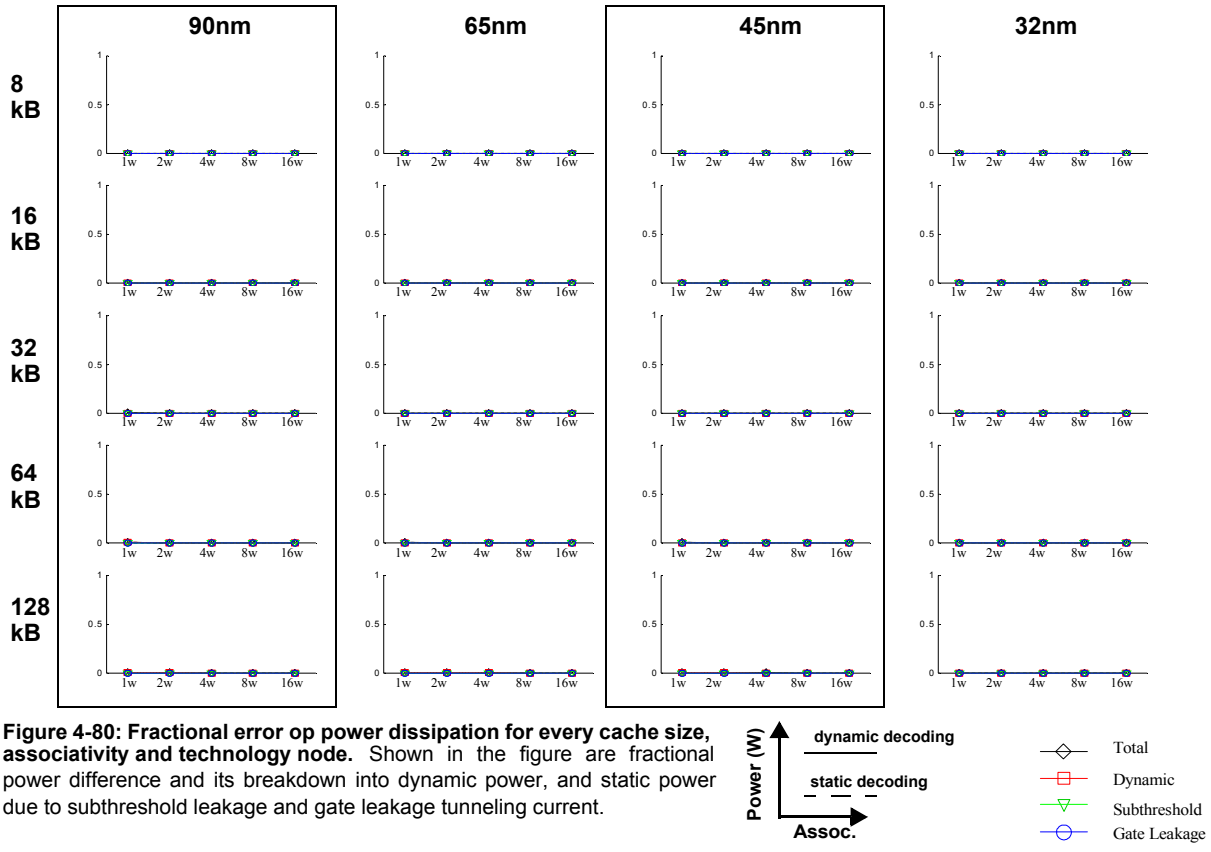


Figure 4-80: Fractional error op power dissipation for every cache size, associativity and technology node. Shown in the figure are fractional power difference and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

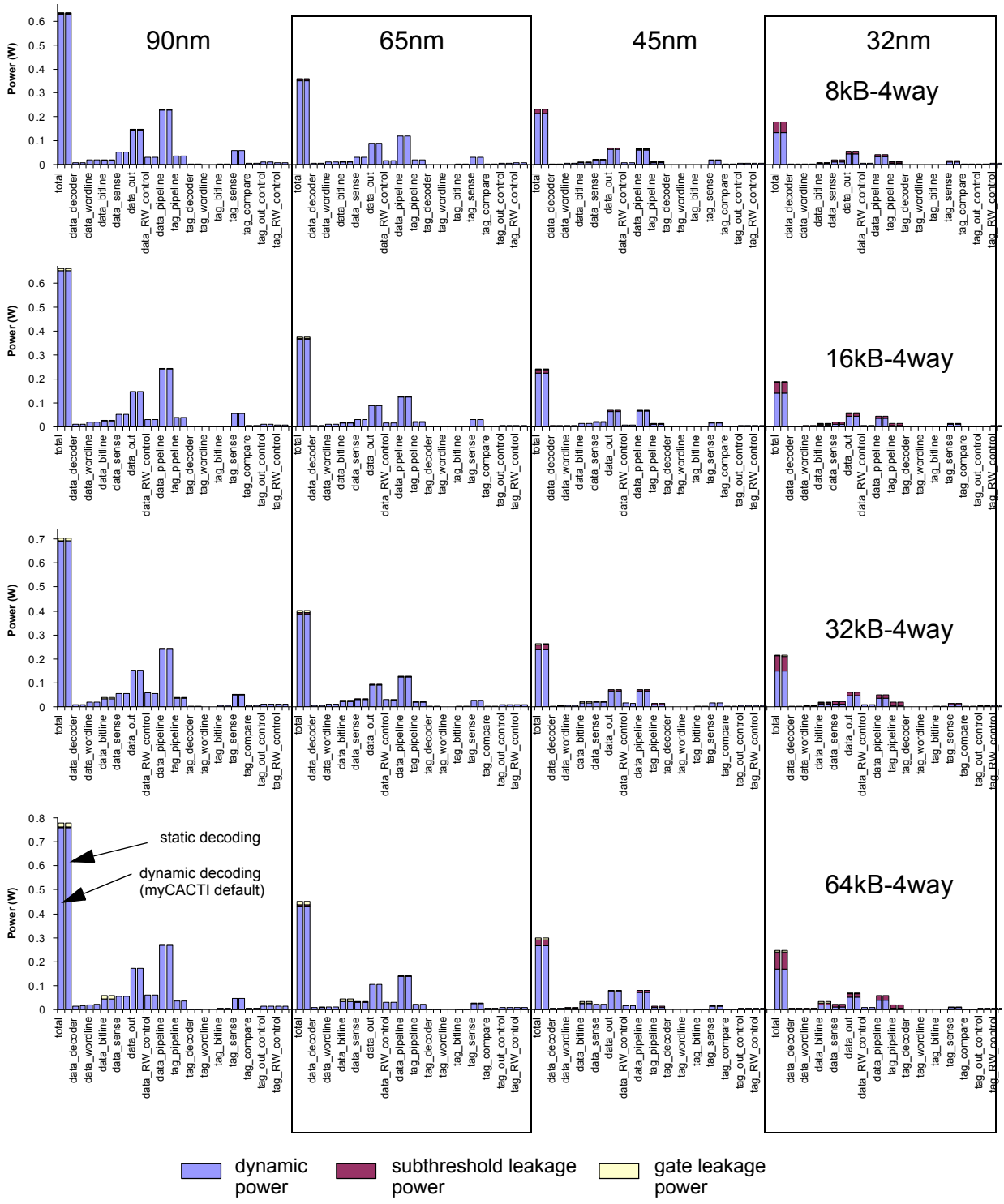


Figure 4-81: Detailed cache power breakdown comparison. Detailed power breakdown for four different cache configurations showing the amount of dynamic, subthreshold leakage and gate leakage power being consumed by different parts of the cache. Each plot shows four sets of values corresponding to different technology nodes. (Note that all three plots use the same scale). The two results are also placed side by side for direct comparison.

Conclusion

We have shown that implementing a decoder using dynamic circuitry results in smaller delays in the decoder, with minimal effect in power. Of course, the additional complexity (in terms of both design, verification, and reliability) of dynamic circuits must be accounted for by the designer, as static circuitry has the advantage in terms of these criteria.

In addition, even though the all of the subparts of a decoder can be sped up by implementing everything using dynamic logic, the speed up in some of the parts of the decoder is typically nullified since these reside in stages that have typically large slack such that any speed up in delay does not affect the critical

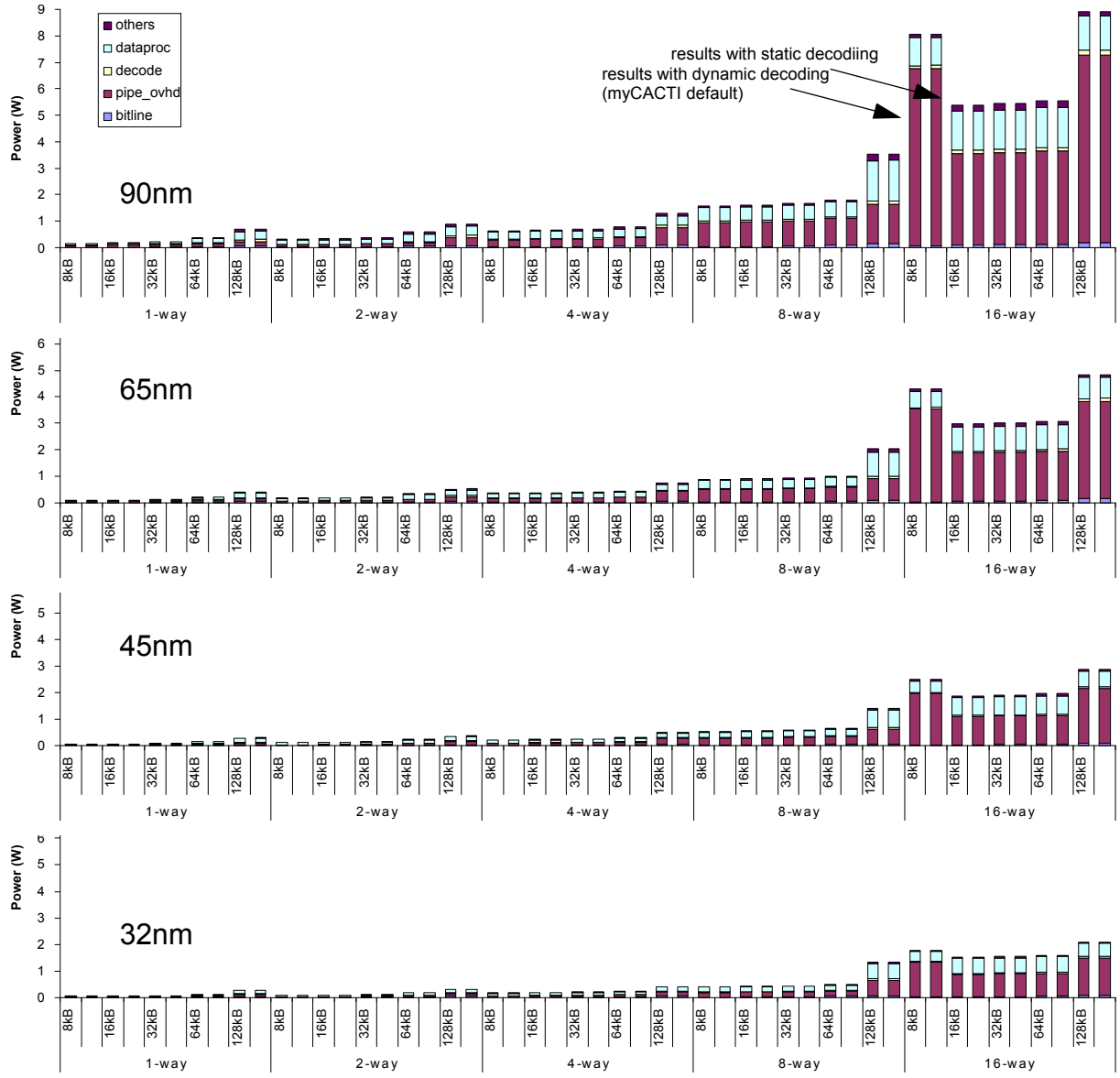


Figure 4-82: Power contributions of different cache parts. Power breakdown showing major cache power contributors for the 65nm and 32nm technology nodes for different cache sizes and associativities. (Note that the y-axis for both plots use the same scale). The two results are also placed side by side for direct comparison.

path at all. This builds the case for only doing a decoder partially in dynamic logic. If the additional speed due to dynamic circuits is warranted, it should only be selectively applied to parts of the circuit that we are relatively sure will be in the critical path, while the rest should be implemented in static logic to minimize design complexity. This way, every part where the designer is paying for the complexity of using dynamic circuits should actually yield some form of delay improvement. In cases where the critical path is not the decoder, we would prefer to have an all-static decoder to simplify the circuit implementation, without really having any negative repercussions -- a win-win situation.

4.5.8 Dual-Vt process technologies

Run details

To produce these data, myCACTI was made to do a design space exploration using the default settings of myCACTI (in particular, enabling dual-Vt transistors) to find the optimal implementation for every cache size and associativity for the 90nm technology node. After determining the optimal implementations for each point in the 90nm design space (in terms of the cache parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl), three more sets of runs are performed for each technology node, with each point in the design space utilizing the optimal implementation that was determined for the 90nm node. In essence, the data for 65nm, 45nm and 32nm constitute a sort of "process-shrink" where we take a design in an older process technology and convert it to a newer one. Although the runs for 65nm and below will not show the absolute optimal implementation possible, this method was chosen to provide a simple but valid comparison between different technology nodes. Implementing the same cache configuration (i.e. same cache size and associativity) using two different implementations for different technology nodes would make the comparisons between different cache components very difficult and generalizations will be hard to make.

After generating a complete set of data using myCACTI default settings, the process is repeated, this with each myCACTI run being set to use purely single-Vt transistors (by using the -singleVt option).

Run timing results

Figure 4-83 shows the delay results for all cache configurations for both runs where results are generated with the assumption of dual-Vt circuits where applicable (the myCACTI default, shown in the plots as the solid lines), and pure single-Vt circuitry (shown as dashed lines). This figure shows the resulting optimal clock period for each configuration, along with the pipe delays for all of the stages in the cache (with the exception of pipe1A_data and pipe1B_data -- the initial decode stages -- since these always have significant slack so they have been omitted from the figure for clarity). Figures 4-84 to 4-90 show the delay component for each individual stage. These individual plots provide a breakdown of the delay of each individual stage by showing the component delays of each. Lastly, Figure 4-91 shows the unpipelined cache access time.

From the comparison plots, we observe the using dual-Vt has a surprisingly minor effect on cache delay. This can be explained in our modeling where time this is because (in our modeling) where the only transistors that reside in the critical path that are implemented with high-Vt transistors are in the bitlines. This tradeoff was chosen because even though implementing the memory cell access and driver transistors as high-Vt will reduce the current driving capability of the memory cell, the reduction in leakage power is significant. Since all of the memory cell array will contribute to static power dissipation even if they are inactive, this is a

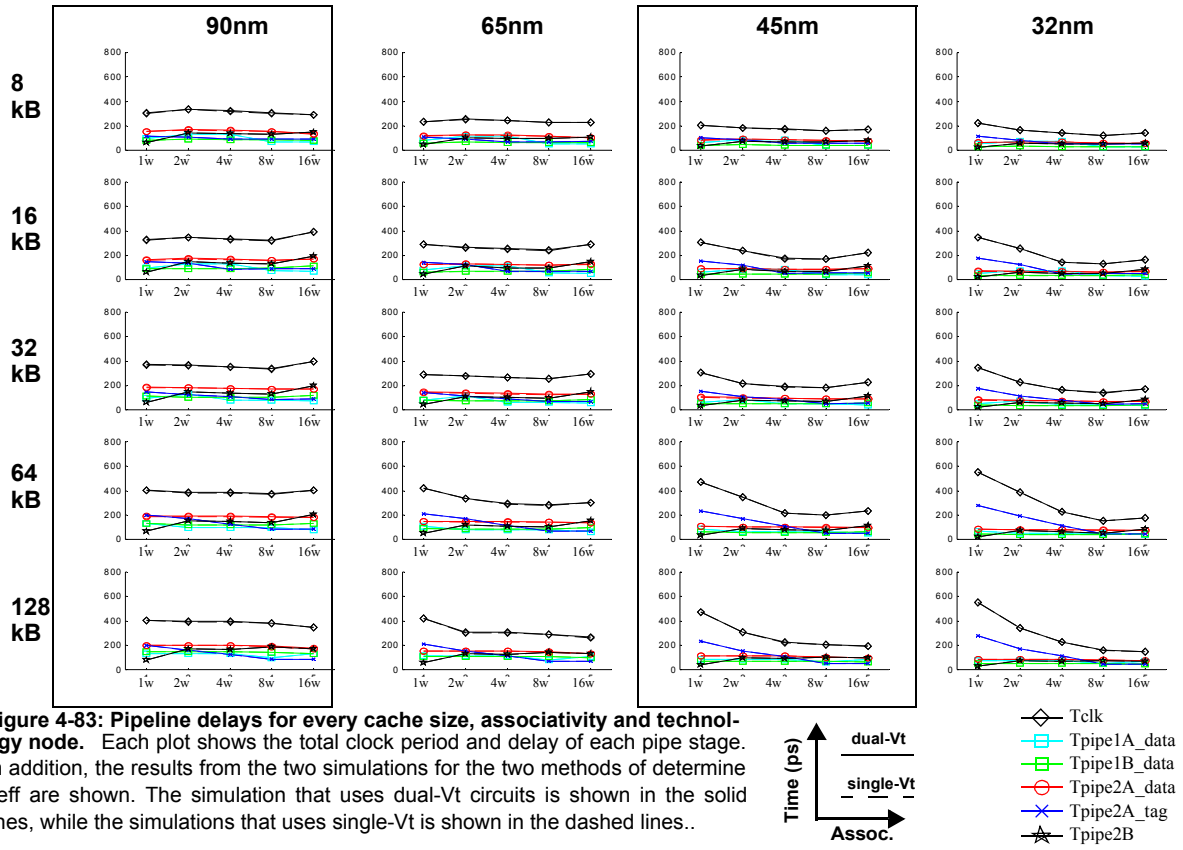


Figure 4-83: Pipeline delays for every cache size, associativity and technology node. Each plot shows the total clock period and delay of each pipe stage. In addition, the results from the two simulations for the two methods of determine Leff are shown. The simulation that uses dual-Vt circuits is shown in the solid lines, while the simulations that uses single-Vt is shown in the dashed lines..

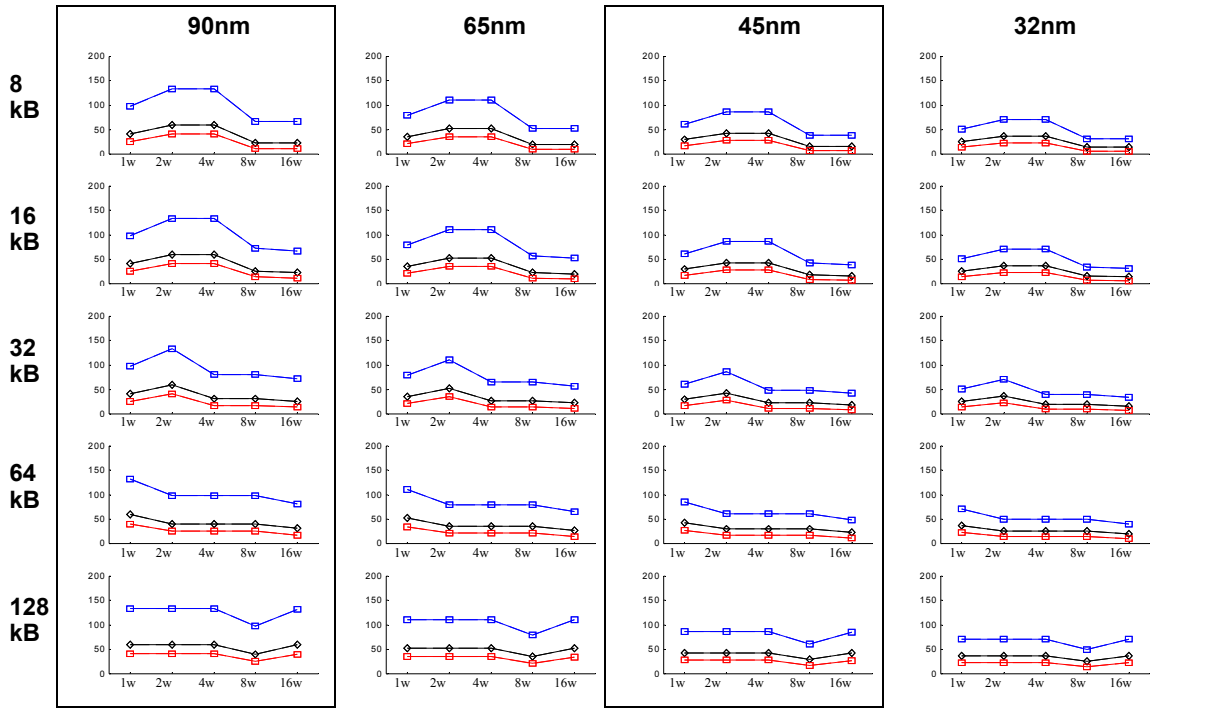


Figure 4-84: Delay components for pipeline stage pipe1A_data . This pipeline stage consists of data address buffering and the predecoder front-end. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

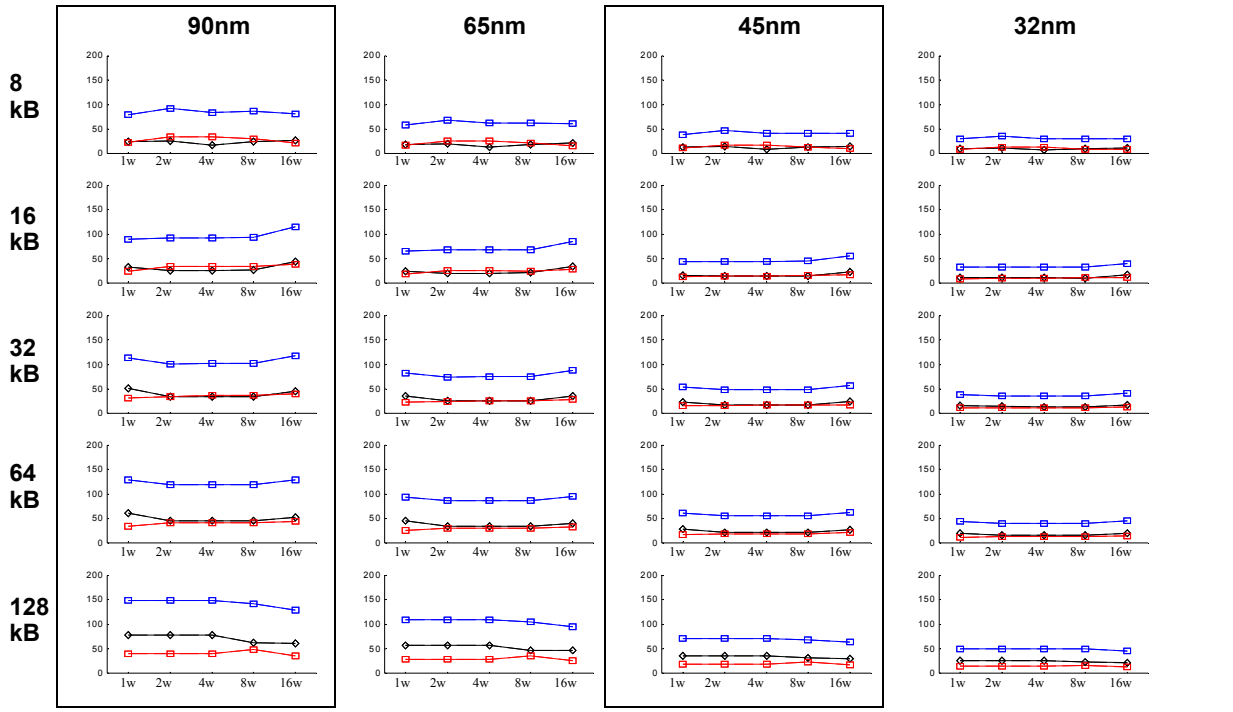


Figure 4-85: Delay components for pipeline stage pipe1B_data . This pipeline stage consists of the data predecoder driver and the wordline front-end. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

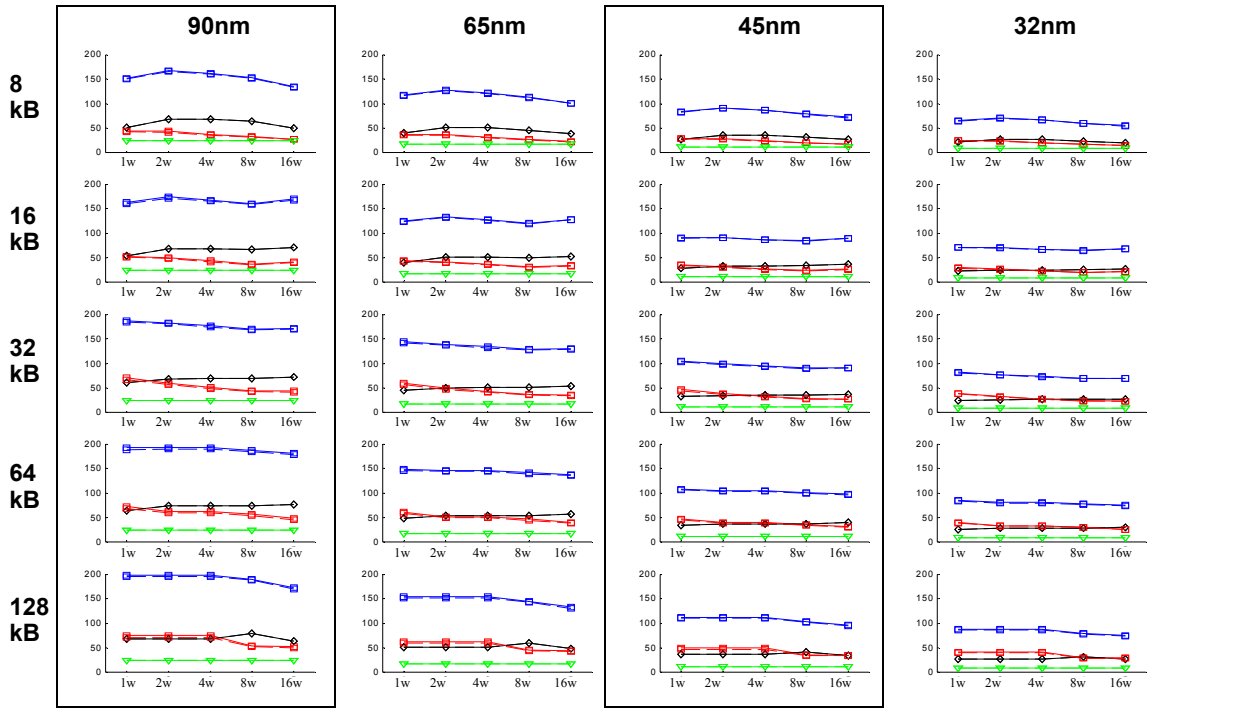


Figure 4-86: Delay components for pipeline stage pipe2A_data . This pipeline stage consists of the wordline drivers, the bitline and the sense amplifier. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

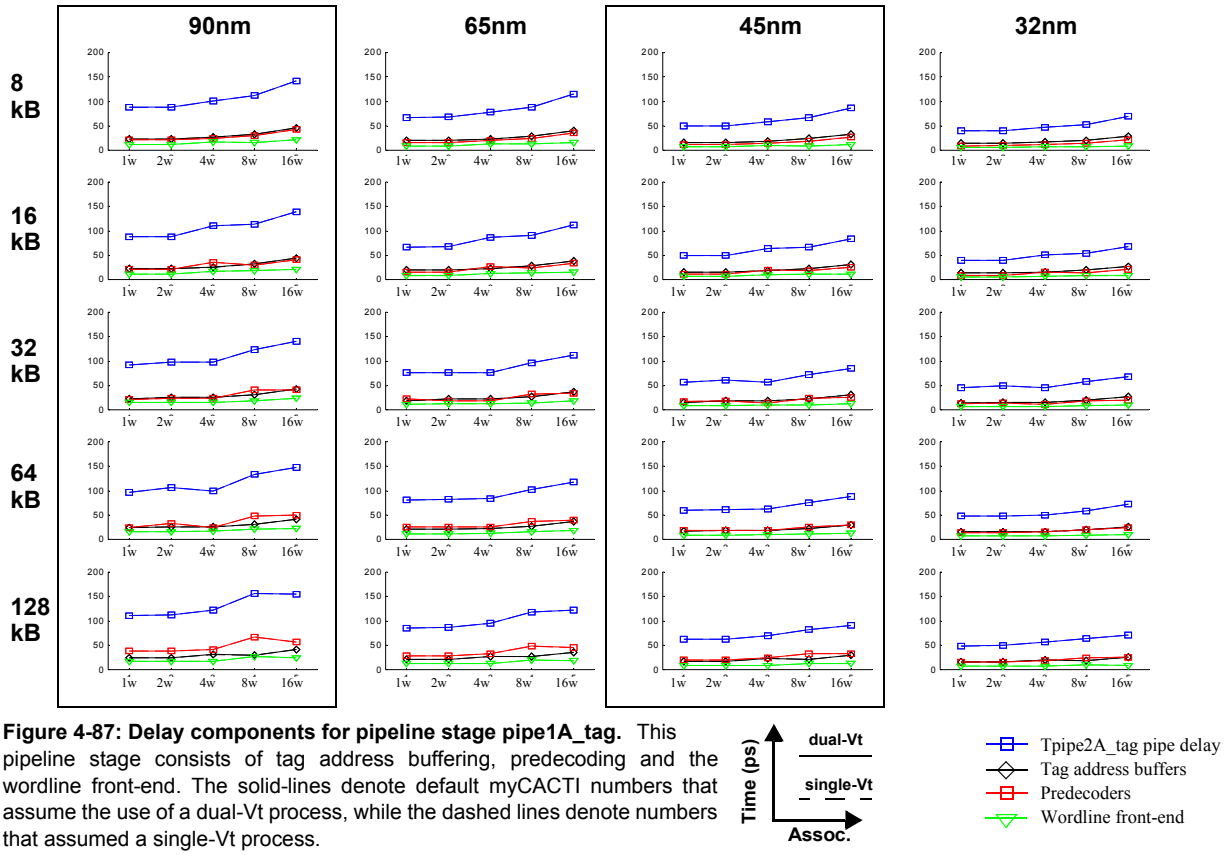


Figure 4-87: Delay components for pipeline stage pipe1A_tag. This pipeline stage consists of tag address buffering, predecoding and the wordline front-end. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

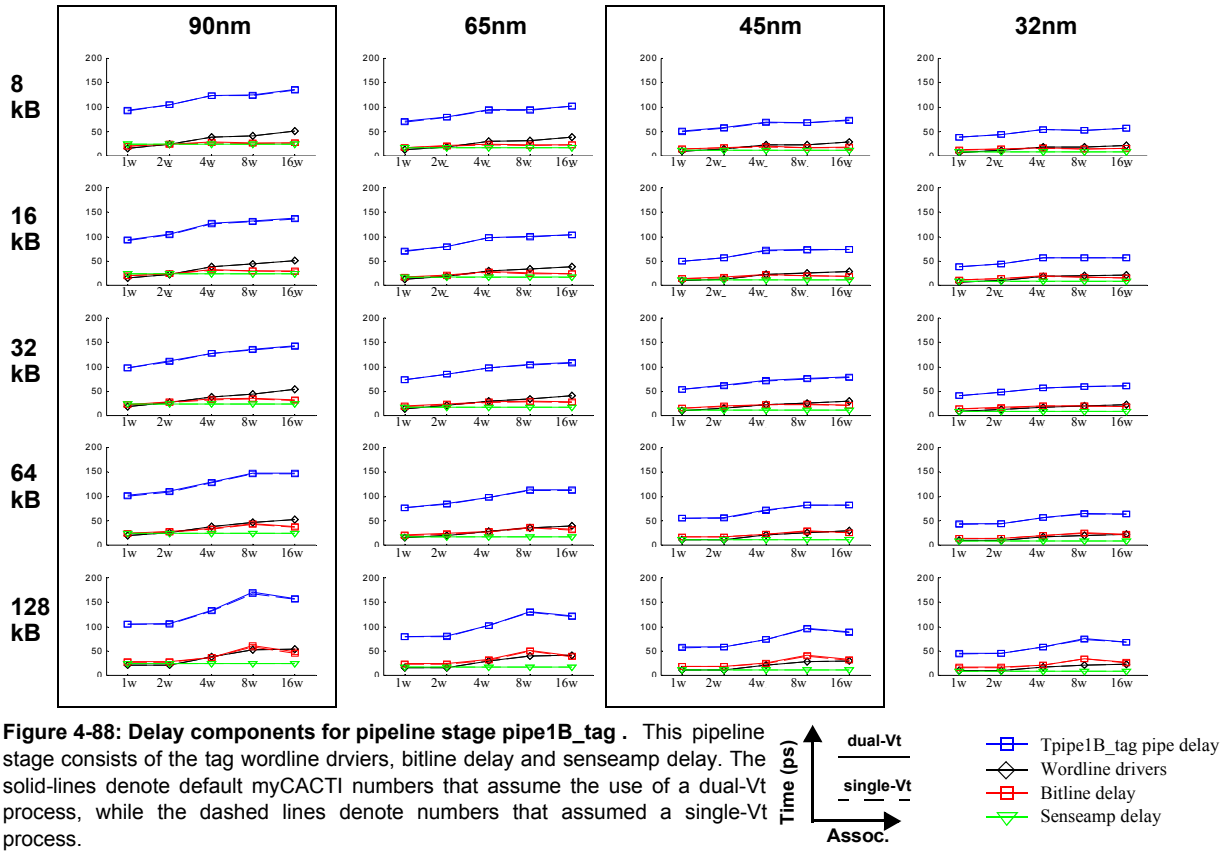


Figure 4-88: Delay components for pipeline stage pipe1B_tag . This pipeline stage consists of the tag wordline drviers, bitline delay and senseamp delay. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

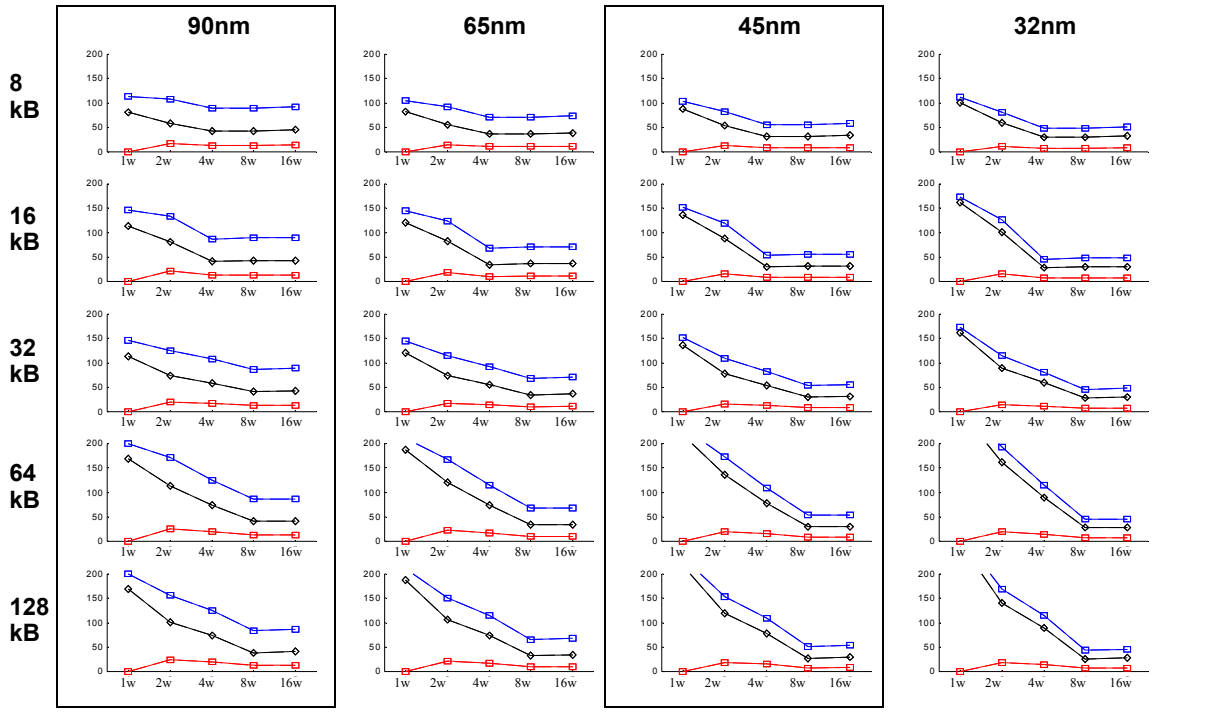


Figure 4-89: Delay components for pipeline stage pipe2A_tag . This pipeline stage consists of enabling the comparator and the actual comparator delay. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

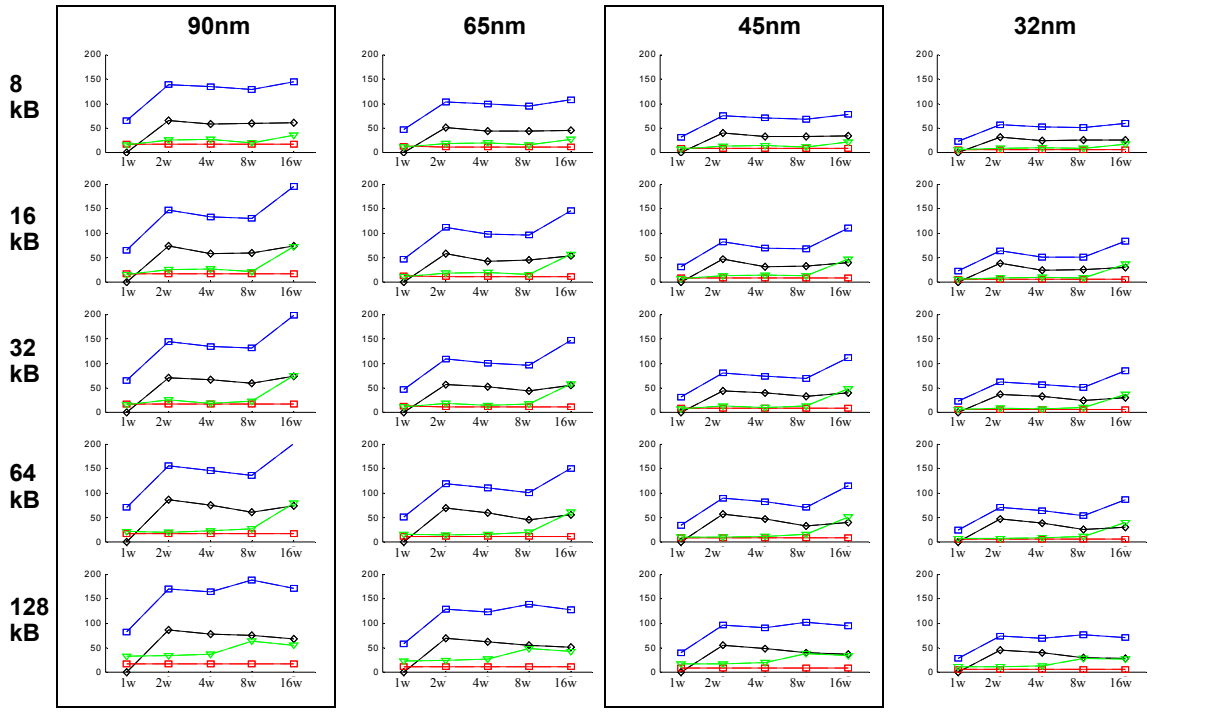


Figure 4-90: Delay components for pipeline stage pipe2B . This pipeline stage consists of driving the output buffer selects and the final data output drive to the cache periphery. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

power vs. speed tradeoff is a very reasonable tradeoff to make. All transistors not in the critical path are implemented with high-Vt to reduce the leakage.

In addition, the clock period of many configurations are actually unaffected and undegraded by using dual-Vt transistors because the critical path in these configurations are determined by circuits that have purely low-Vt transistors in the critical path (e.g. the comparator, data output drivers, etc.). Even so, in the configurations that are degraded, there are mostly only a few picosend difference between the single-Vt and dual-Vt implementation, since, as earlier said, it is only the bitline stage that sees the degradation. Since the optimizaations were done assuming dual-Vt usage, the number of rows have been reduced in order to minimize the load being driven in the bitline such that any degradation is reduced. Implementations that are not optimized in such a manner and consequently have significant loading in the bitlines may show more degraadation than what we have observed here. Looking at the typical number of rows used in our optimal implementations, we see the the maximum number of rows used is 64 (still a pretty low number), while the typical is from 16 to 48 rows.

Looking at the individual pipe stages, we see that the typical critical pipeline stage is either pipe2A_data (wordline drive, bitline and sense), pipe2A_tag (the tag compare), or for highly-associative caches, pipe2B (data output). The stage pipe2A_data has three components (aside from flop overhead) -- the wordline drive, the bitline delay and the senseamp delay. Of these thre components, the wordline drive and the

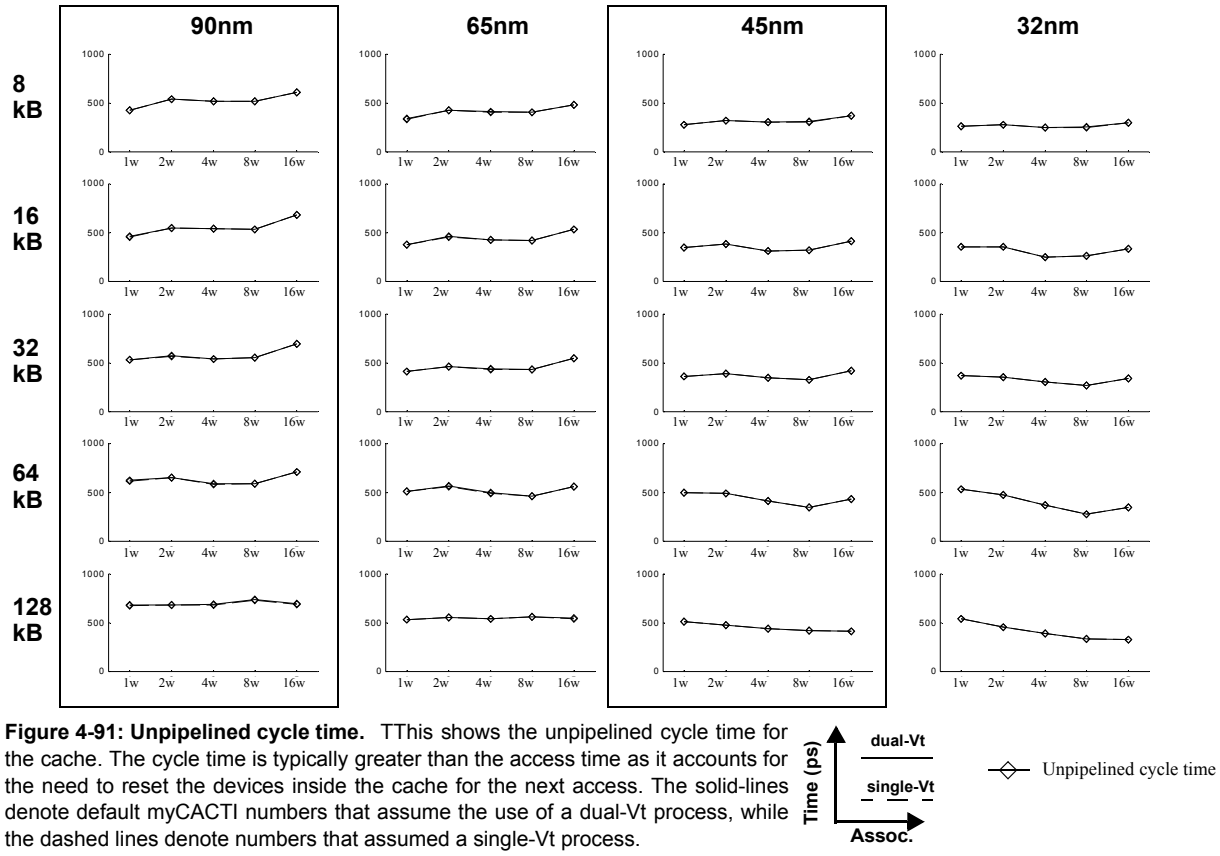


Figure 4-91: Unpipelined cycle time. TThis shows the unpipelined cycle time for the cache. The cycle time is typically greater than the access time as it accounts for the need to reset the devices inside the cache for the next access. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

bitline delay typically account for roughlyl 80% of the delay, with roughly equal distribution of configurations where one is significantly greater than the other. The stage pipe2A_tag has two components -- the actual compare and the compare merge and partly driving the select mux, with the compare always greater than the merge, especially for mid to low associativities (where the merge stage is actually responsible for driving the mux stge and hence, drives a larger load). Lastly, stage pipe2B has three components -- the mux drive, the output predrive and the main output drive, with the mux driver typically being dominant over the pre drive and the final output drive. This makes sense, as we can consider these three stages as one buffer chain that drives the output load, and the predrive and the output drive only account for the last stage of this chain, while the mux driver accounts for the initial three or so stages. The main exceptions are for very highly associative caches (e.g. 16ways), where the RC delay of driving the data from the cache interior to its periphery becomes significant and degrades the performance of the buffer chain's last stage drastically.

Run power results

Figure 4-92 shows the total power and its breakdown into dynamic power (switching power) and static power (dissipated by subthreshold leakage and gate leakage currents) for every point in the design space that was considered (i.e. cache size, associativity and technology node). Each plot shows both the results from myCACTI runs that assume either dual-Vt or single-Vt circuits. Figure4-93 demonstrates the differences

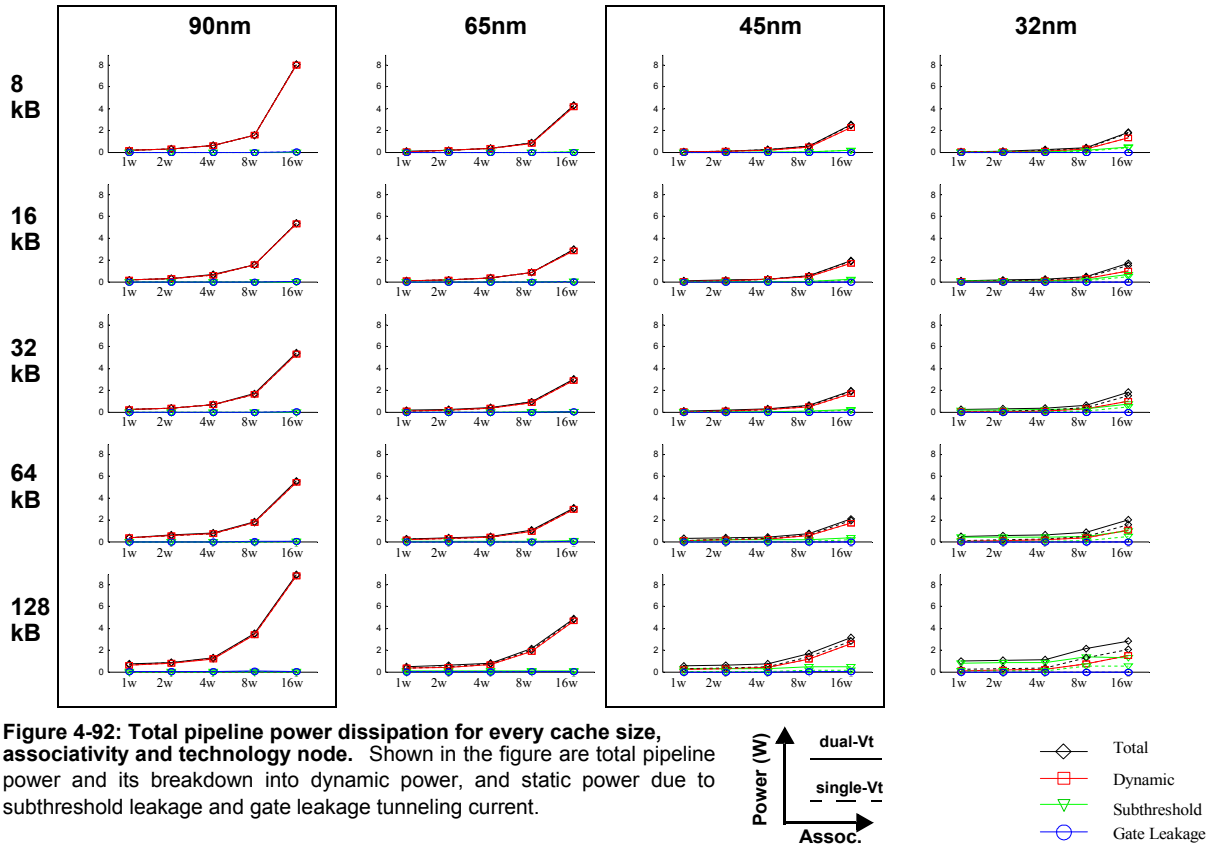


Figure 4-92: Total pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

more explicitly by plotting the fractional difference between the two runs normalized to the total power of the default myCACTI result. Figure 4-94 shows detailed power breakdowns of some representative design points in order to explore in detail the causes of the differences. In addition, the results from the two runs are placed side by side for easy comparison. Lastly, Figure 4-95 identifies how much power is being dissipated by major cache blocks, and again shows the results from the two runs side by side for easy comparison.

From the figures, we see that simulations assuming dual-Vt and single-Vt have minimal differences at the larger technology nodes, but the difference becomes very significant at the smaller technology nodes (especially 32nm). At 90nm dynamic power is the dominant component of total power, such that using dual-Vt transistors does not provide much advantage. At 65nm, the difference starts to become measurable, but is really only significant (in terms of total power) for larger caches with very low associativities (e.g. 128kB 1way/2way) where the subthrehold leakage power is comparable to the dynamic power. At 45nm, the difference becomes even more evident, and at 32nm the difference is very pronounced, not just for subthreshold leakage power, but also for total power, even for the 8kB cache as long as associativity is 8way or below.

In general, using dual-Vt circuits really makes a difference in power for medium to large sized caches in the 45nm and 32nm generation especially for lower associativity caches.

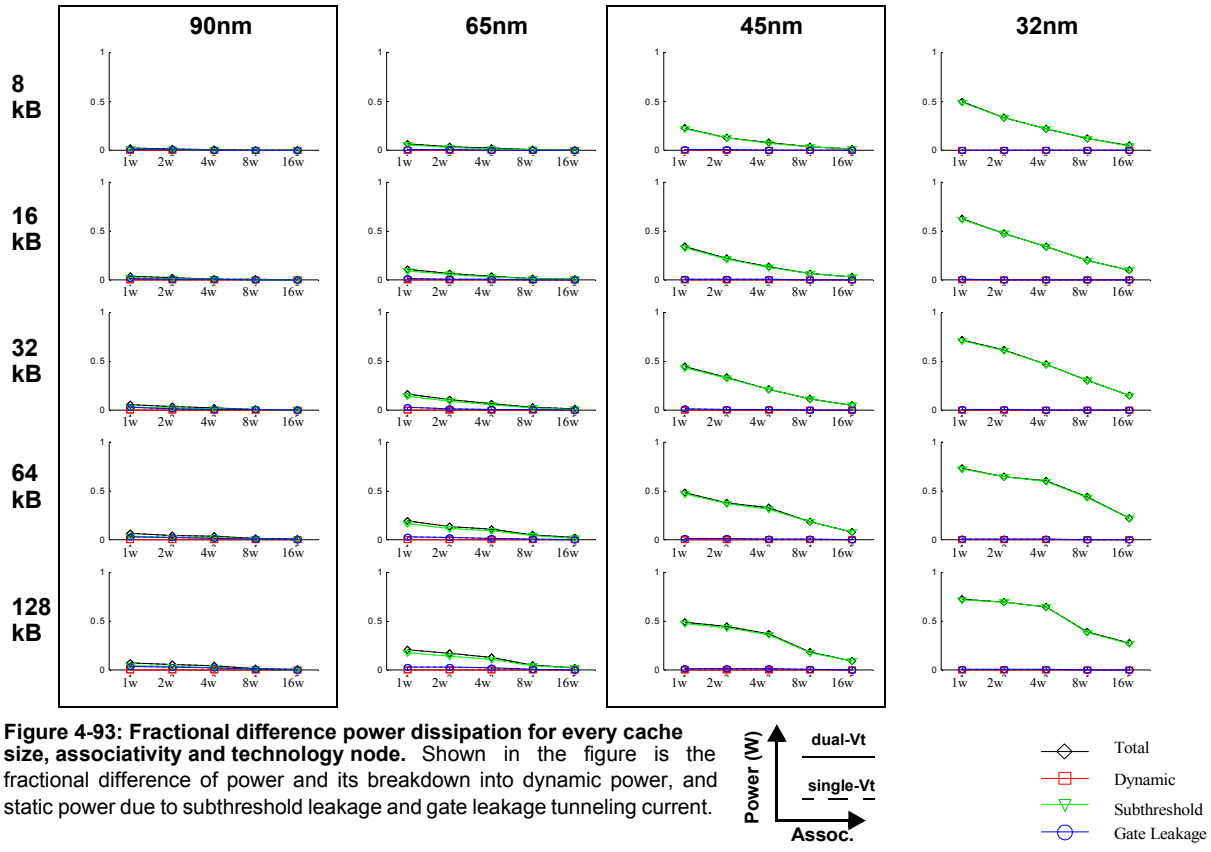


Figure 4-93: Fractional difference power dissipation for every cache size, associativity and technology node. Shown in the figure is the fractional difference of power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

Looking at the detailed power breakdown, the plots essentially show detailes of some of the points shown in the design space exploration (specifically 8kB-4way, 16kB-4way, 32kB-4way and 64kB-4way).

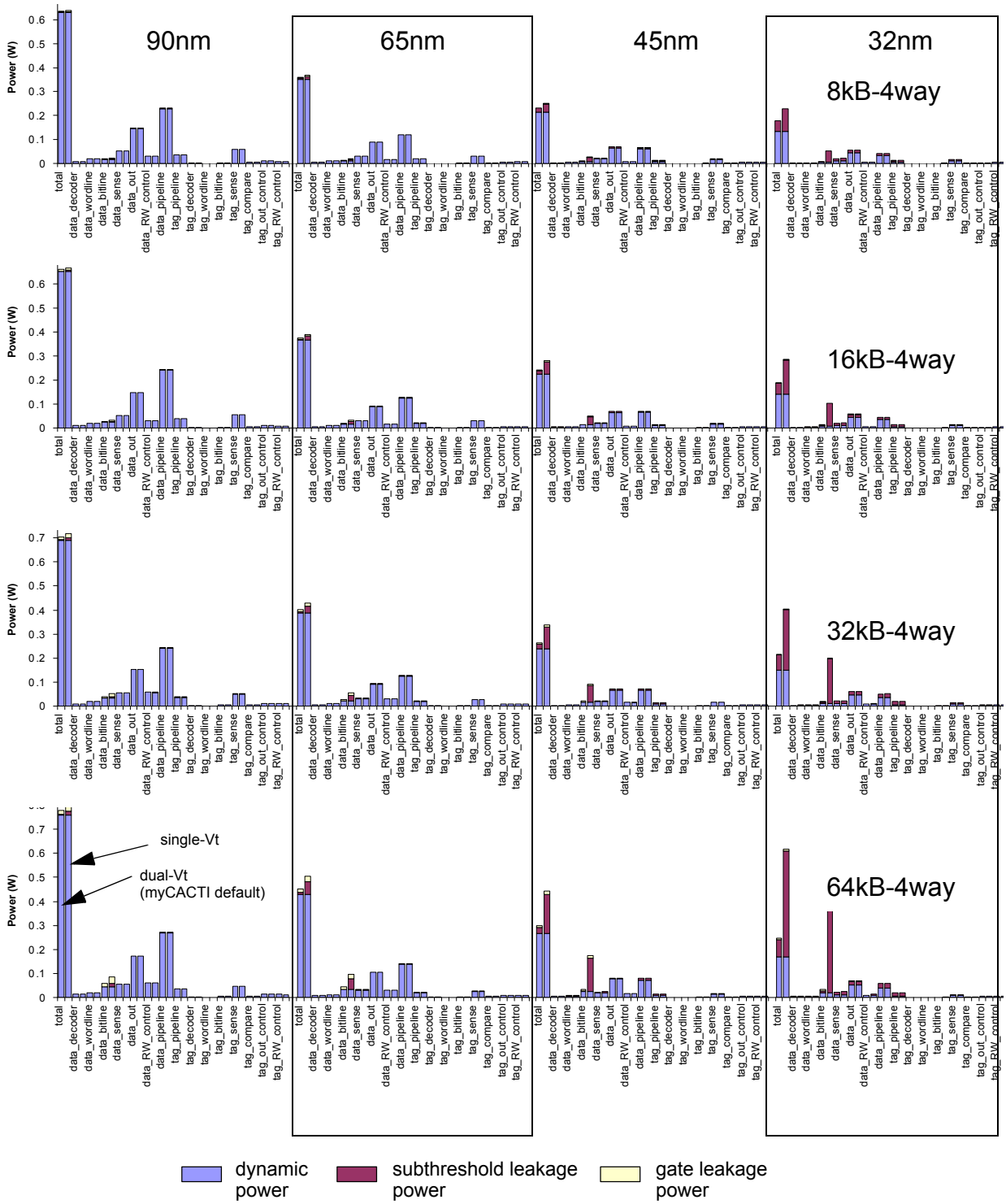


Figure 4-94: Detailed cache power breakdown comparison. Detailed power breakdown for four different cache configurations showing the amount of dynamic, subthreshold leakage and gate leakage power being consumed by different parts of the cache. Each plot shows four sets of values corresponding to different technology nodes. (Note that all three plots use the same scale). The two results are also placed side by side for direct comparison.

Instead of just showing total power subdivided into dynamic, subthreshold and gate leakage, we show how much poewr is dissipated by different parts of the cache and what kind of power is being dissipated. We observe that for 90nm, most of the power being dissipated in the cache is dynamic, with a very significant portion of this power being dissipated in the pipeline overhead circuits. For deeper nanometer nodes, the dynamic power expectedly decreases, while subthreshold leakage increases and, again, gate leakage decreases. Subthreshold leakage starts to become dominant at 65nm (for single-Vt) or at 45nm (for dualVt) with most of the increase being accounted for by the bitlines.

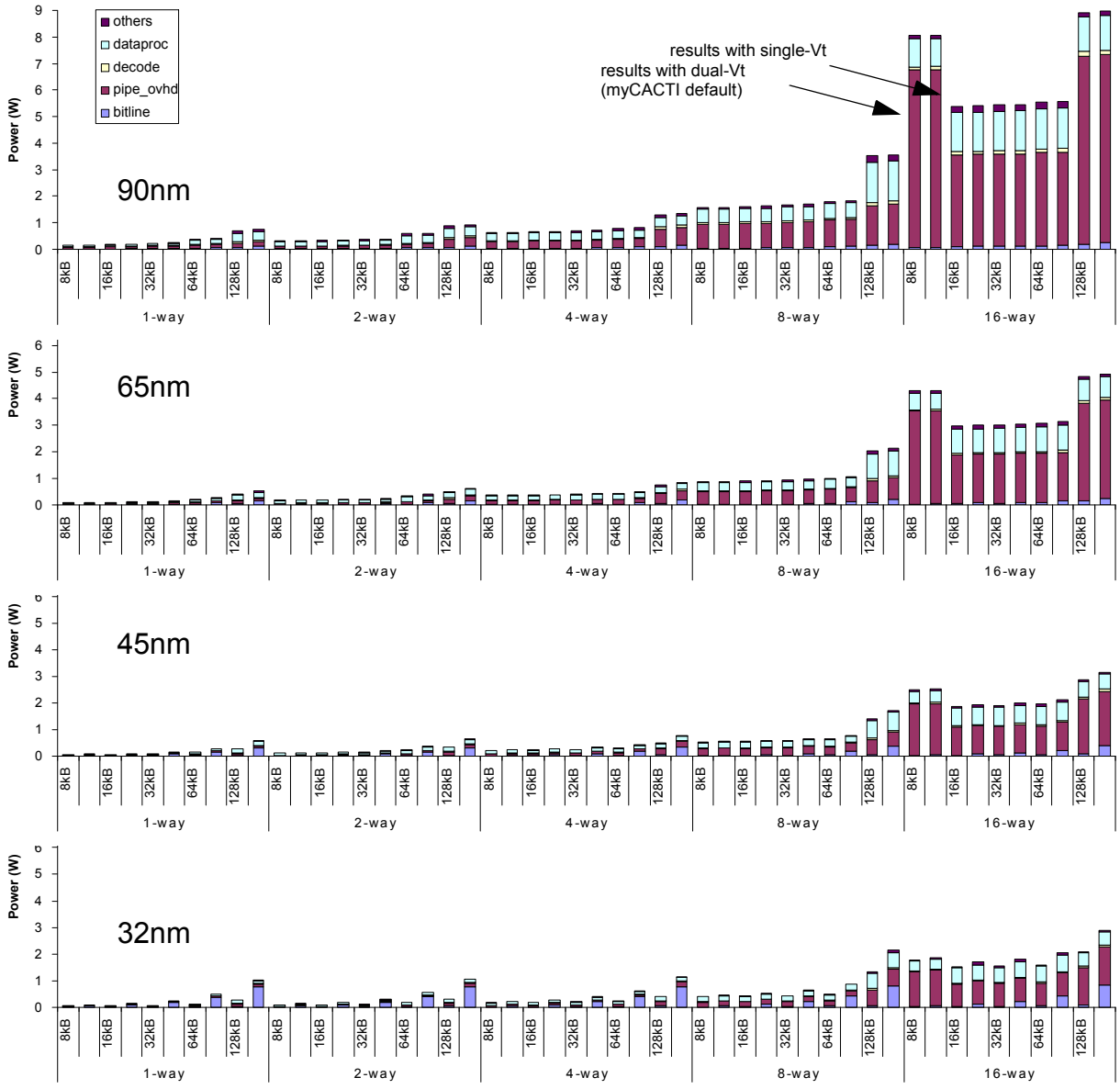


Figure 4-95: Power contributions of different cache parts. Power breakdown showing major cache power contributors for the 65nm and 32nm technology nodes for different cache sizes and associativities. (Note that the y-axis for both plots use the same scale). The two results are also placed side by side for direct comparison.

We also observe that the total power initially tends to decrease with each generation as the dynamic power decreases. At the deep nanometer nodes, depending on whether the cache is single or dual-Vt, the power dissipation trend may reverse (single-Vt) or continue (dual-Vt).

Lastly, the difference in single vs. dual-Vt really is very significant at the 45nm and 32nm nodes for the 32kB-4way and 64kB-4way caches. We actually see a drop in the difference from 64kB-4way to 128kB-8way, and this is partially accounted for by the increase in the dynamic power component of total power, but a significant contributor is also the increse in power (both dynamic and static) in the pipeline overhead, with minimal differences between the single and dualVt implementations since a significant part of the pipeline circuits are tyipcally implemented in low-Vt (e.g. the clock tree drivers).

Conclusion

We have shown that implementing a cache in dual-Vt as opposed to single-Vt has results only in a slight degradation of the total clock period of a pipelined cache, as the only critical path that is affected (with proper Vt placement) is the access and driver transistors in the memory cell. In addition, the power savings when going from a single-Vt implementation to a dual-Vt is significant, especiallly at the 45nm and 32nm node.

We therefore conclude that unless the slight speed degradation is really detrimental, the power savings realized in implementing a cache as dual-Vt more than outweighs the slight degradation in the bitline delay. In implementations where the bitline delay does not reside in the critical path, we can essentially get this power savings for free, as all other critical paths are implemented in low-Vt transistors such that they don't incur any delay, while the whole circuit still enjoys the power savings accrued with using high-Vt transistors in devices outside of the critical path.

4.5.9 Single-ended sensing

Run details

To produce these data, myCACTI was made to do a design space exploration using the default settings of myCACTI (in particular, enabling low-swing differential sensing) to find the optimal implementation for every cache size and associativity for the 90nm technology node. After determining the optimal implementations for each point in the 90nm design space (in terms of the cache parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl), three more sets of runs are performed for each technology node, with each point in the design space utilizing the optimal implementation that was determined for the 90nm node. In essence, the data for 65nm, 45nm and 32nm constitute a sort of "process-shrink" where we take a design in an older process technology and convert it to a newer one. Although the runs for 65nm and below will not show the absolute optimal implementation possible, this method was chosen to provide a simple but valid comparison between different technology nodes. Implementing the same cache configuration (i.e. same cache size and associativity) using two different implementations for different technology nodes would make the comparisons between different cache components very difficult and generalizations will be hard to make.

After generating a complete set of data using myCACTI default settings, the process is repeated, this with each myCACTI run being set to utlize full-swing single-ended sense amplifiers (by using the -single_ended_sensing option).

Run timing results

Figure 4-96 shows the delay results for all cache configurations for both runs where results are generated with the assumption of low-swing differential sense-amplifiers (the myCACTI default, shown in the plots as the solid lines), and full-swing single-ended sense-amplifier circuitry (shown as dashed lines). This figure shows the resulting optimal clock period for each configuration, along with the pipe delays for all of the stages in the cache (with the exception of pipe1A_data and pipe1B_data -- the initial decode stages -- since these always have significant slack so they have been omitted from the figure for clarity). Figures 4-97 to 4-103 show the delay component for each individual stage. These individual plots provide a breakdown of the delay of each individual stage by showing the component delays of each. Lastly, Figure 4-104 shows the unpipelined cache access time.

From the comparison plots, it can be immediately seen that using full-swing single-ended sensing instead of low-swing differential sensing will typically result in a significant degradation of the cache clock period. In some cases greater this degradation can go over 200ps.

In terms of cache size, the degradation in delay typically goes up with cache size, although the maximum degradation found for each cache size is not exactly monotonic. From detailed measurements, we see that the maximum delay of any optimal configuration for 8kB is roughly 100ps, for 16kB is 150ps, for

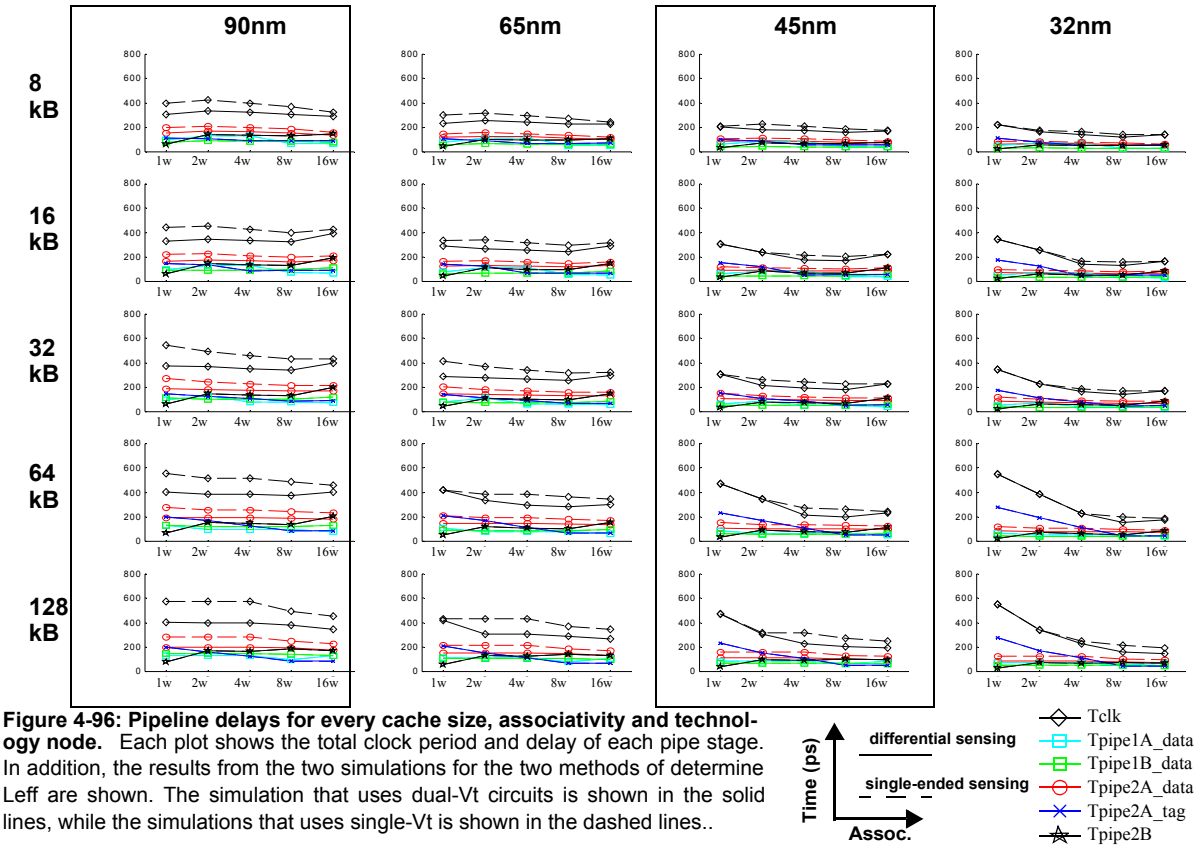


Figure 4-96: Pipeline delays for every cache size, associativity and technology node. Each plot shows the total clock period and delay of each pipe stage. In addition, the results from the two simulations for the two methods of determine Leff are shown. The simulation that uses dual-Vt circuits is shown in the solid lines, while the simulations that uses single-Vt is shown in the dashed lines..

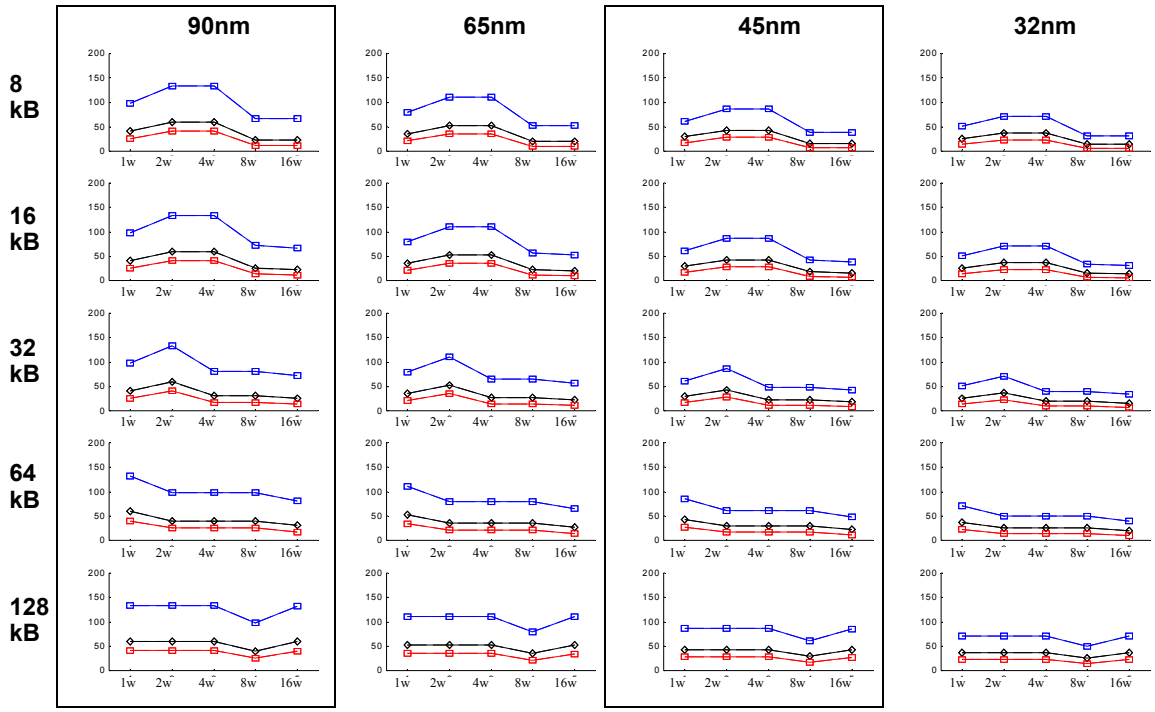


Figure 4-97: Delay components for pipeline stage pipe1A_data . This pipeline stage consists of data address buffering and the predecoder front-end. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

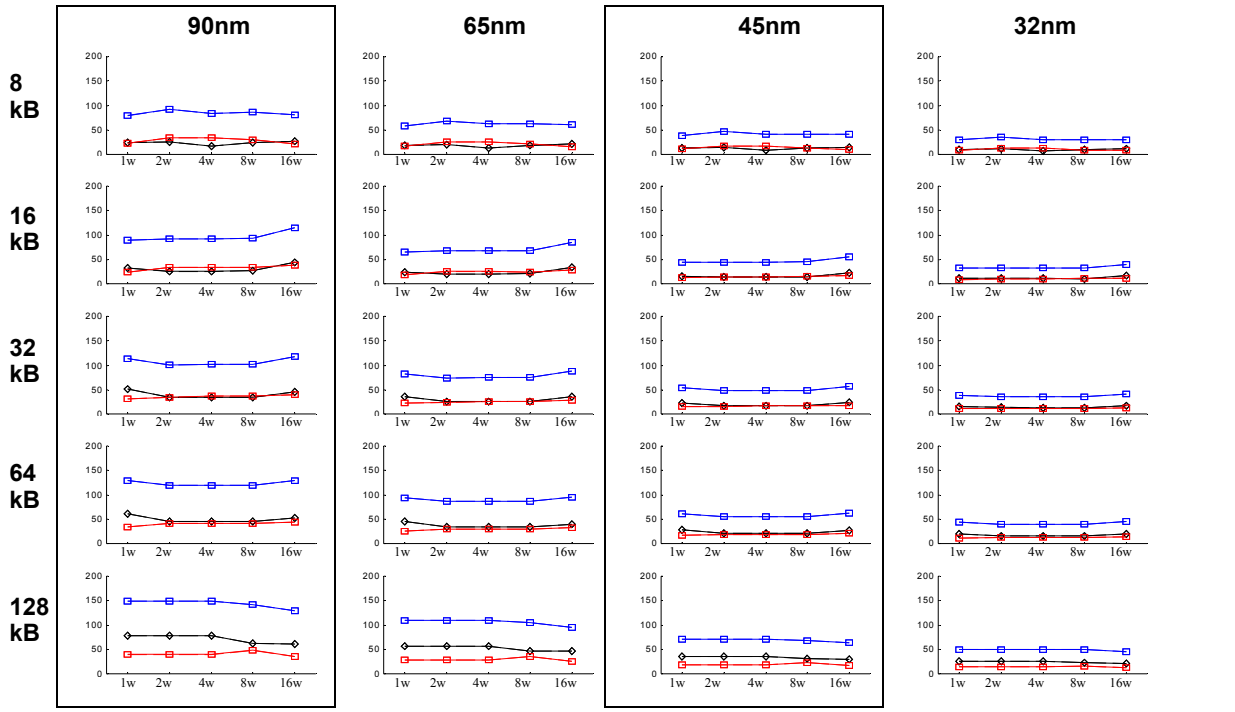


Figure 4-98: Delay components for pipeline stage pipe1B_data . This pipeline stage consists of the data predecoder driver and the wordline front-end. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

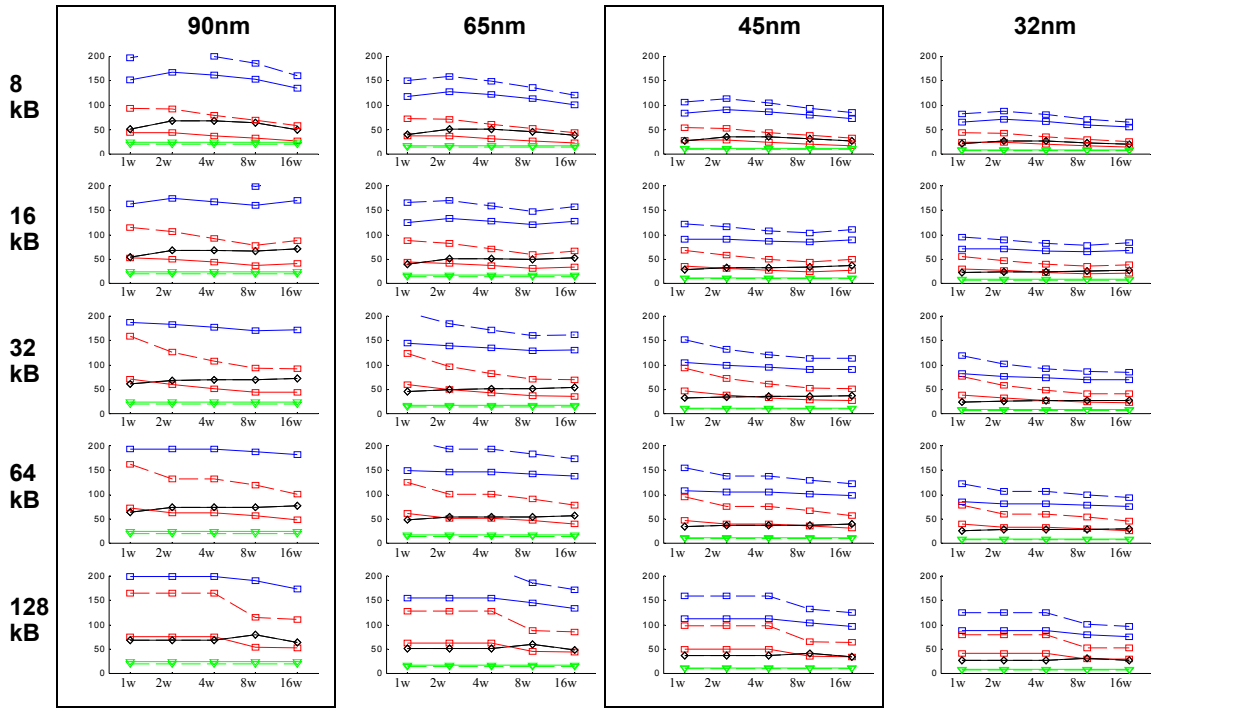


Figure 4-99: Delay components for pipeline stage pipe2A_data . This pipeline stage consists of the wordline drivers, the bitline and the sense amplifier. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

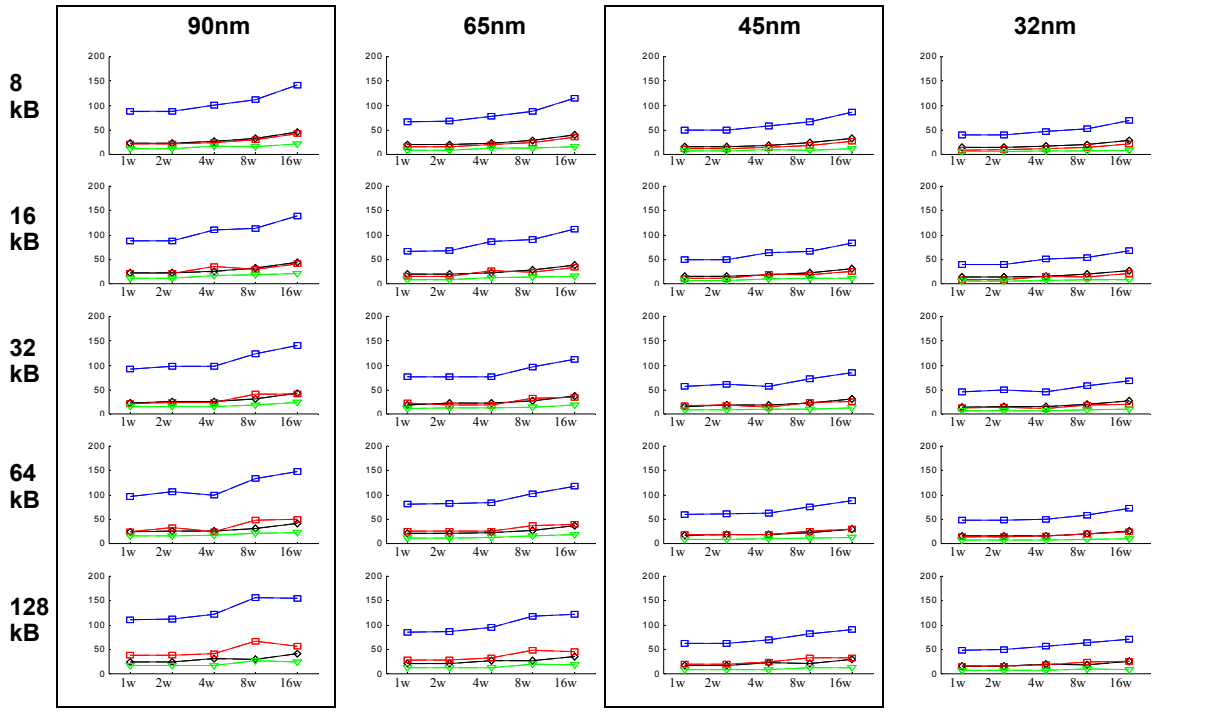


Figure 4-100: Delay components for pipeline stage pipe1A_tag. This pipeline stage consists of tag address buffering, predecoding and the wordline front-end. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

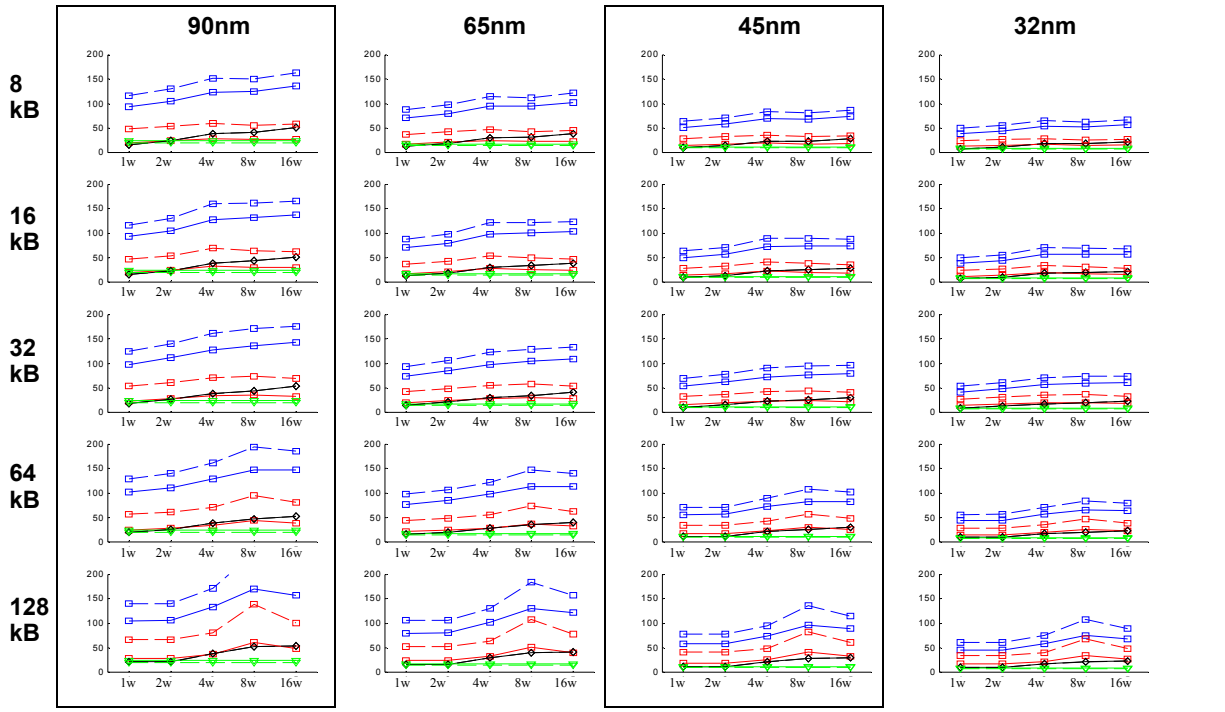


Figure 4-101: Delay components for pipeline stage pipe1B_tag . This pipeline stage consists of the tag wordline drviers, bitline delay and senseamp delay. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

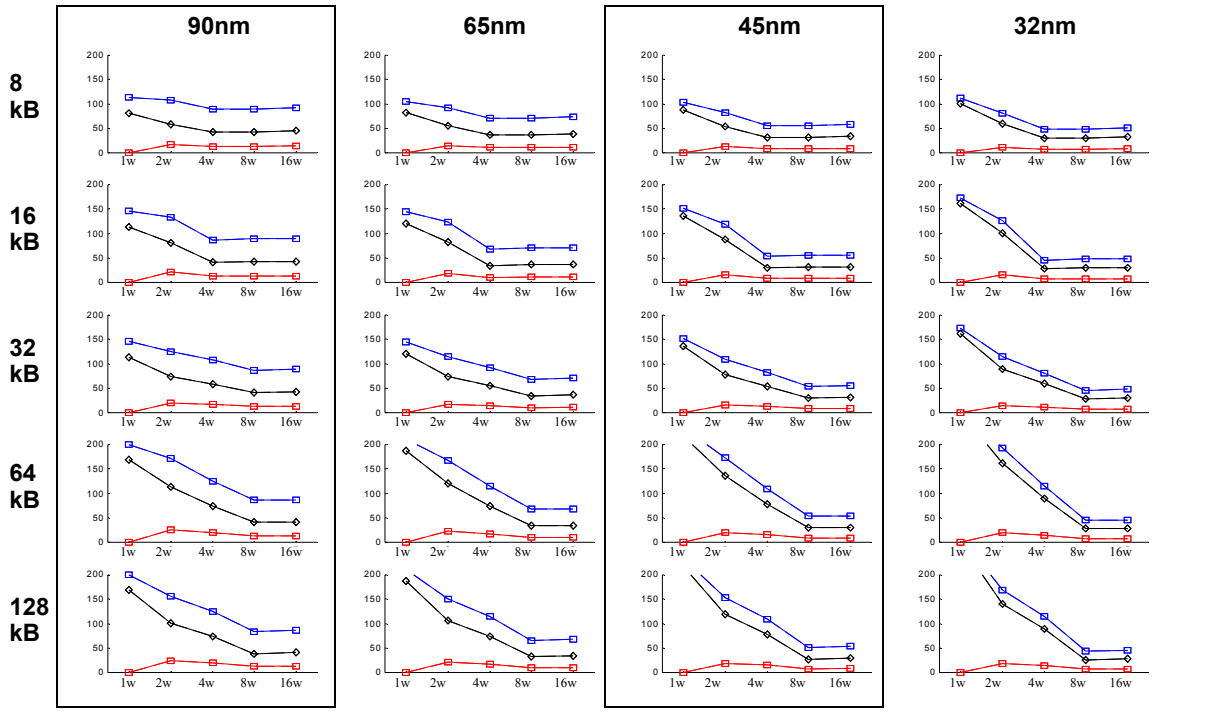


Figure 4-102: Delay components for pipeline stage pipe2A_tag . This pipeline stage consists of enabling the comparator and the actual comparator delay. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

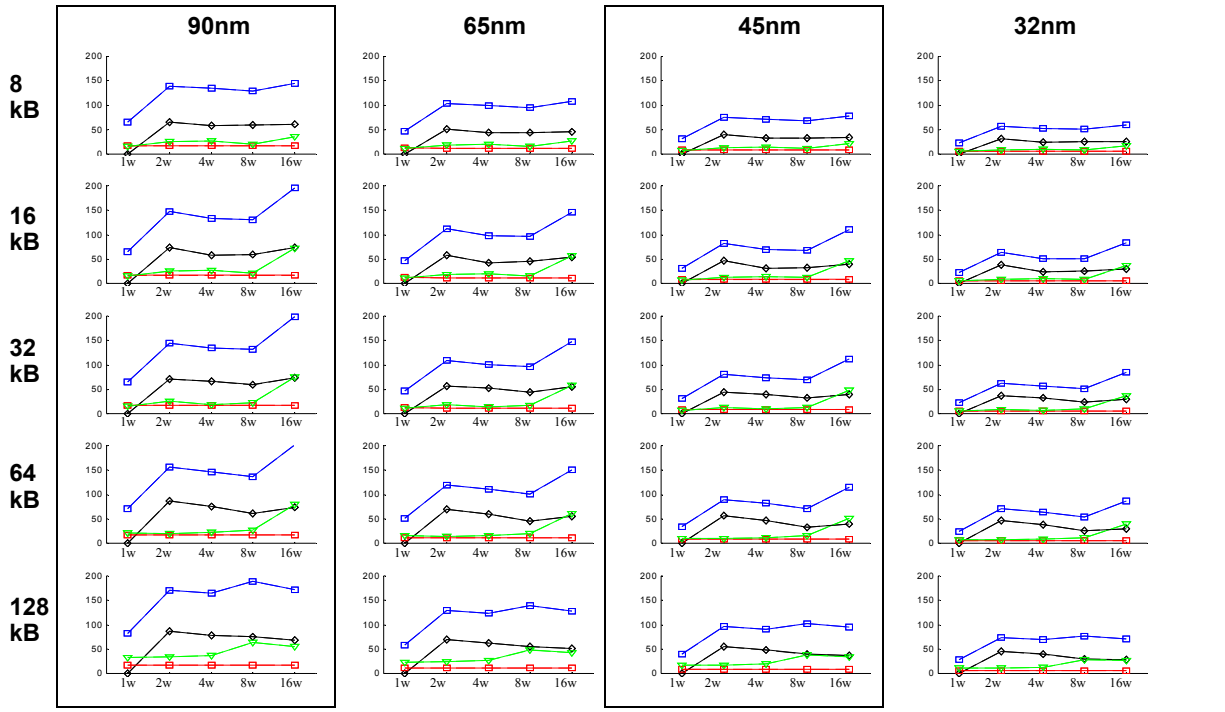


Figure 4-103: Delay components for pipeline stage pipe2B . This pipeline stage consists of driving the output buffer selects and the final data output drive to the cache periphery. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

32kB is 175ps, for 64kB is 150ps, and for 128kB is 160ps. Note that in the larger cache sizes, the maximum delay does not really increases monotonically. Again, this is due to the pipelined nature of the cache, where a specific delay may or may not be the critical path such that even if the delay increased from one configuration to the next, it may not necessarily be the path that determines the clock period as some other delay may exist (and may have been degraded more) that is greater and hence, is the one that sets the cache's critical path and clock period. Looking at the actual degradation to the stage containing the sense amplifier, we do observe a monotonic increase in delay with cache size, but again, even though the delay increases, this is not necessarily the critical path.

The delay degradation when going from differential to single-ended sensing tends to become smaller with higher cache associativity. This would be a result of the cache having fewer rows per bitline as associativity increases, such that the bitline has significantly less loading and suffers less delay degradation even if the memory cell has to produce a full-swing discharge of the bitline (i.e. less capacitance to be de discharged, along with a lower resistance of the line). Again, exceptions are cases where, even with the much slower bitline delay, the compare stage is still the critical path, so we obseve some discontinuities in this trend.

One important thing to emphasize is that, as described earlier in the description of the runs, these implementations were optimized for differential sensing, with the optimal implementations (in terms of the cache's implementation sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntwl) being used for the single-ended runs also,

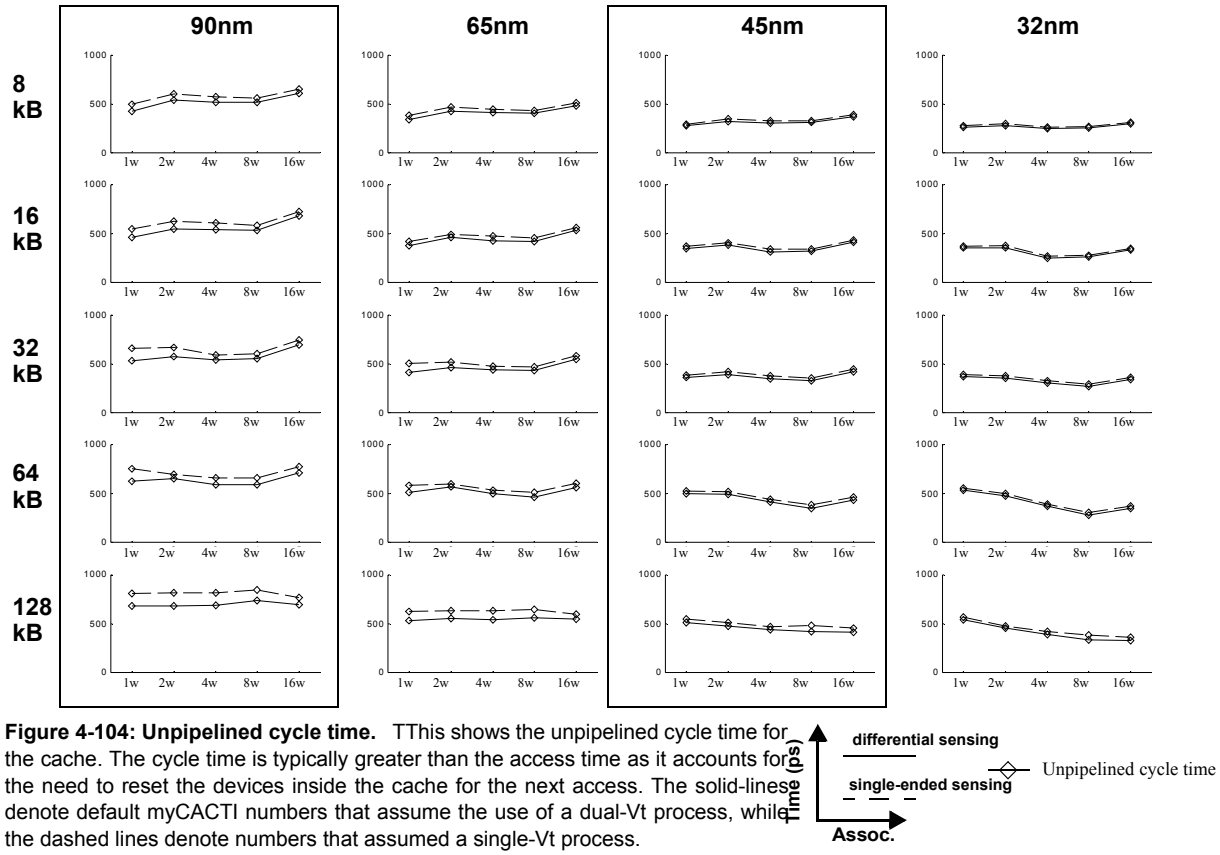


Figure 4-104: Unpipelined cycle time. TThis shows the unpipelined cycle time for the cache. The cycle time is typically greater than the access time as it accounts for the need to reset the devices inside the cache for the next access. The solid-lines denote default myCACTI numbers that assume the use of a dual-Vt process, while the dashed lines denote numbers that assumed a single-Vt process.

in order to have as close a comparison as possible. An optimization run that finds implementations with the assumption of single-ended sensing will most probably result in better delays, but for the purpose of this particular study, we prefer to compare the same cache implementation in order to easier isolate the effects of the change in sense amplifier configuration without introducing any other changes in the rest of the cache.

Another important observation is that the bitline/sense-amp delay degradation is worst for the larger technology nodes. For the smaller technology nodes, the degradation in delay is not so large such that the compare stage typically becomes the critical path. But even if the compare stage was ignored and we simply focus on the delay of the bitline and sense-amplifier stage, the delay degradation still tens to become smaller with technology scaling. One explanation is that with a smaller supply voltage, there is less voltage to discharge such that the bitline delay is reduced to some degree. In addition, the much reduced capacitive loading in the smaller technology nodes would also help reduce the delay significantly.

Lastly, we can see from the different comparison plots that all other pipeline stages that do not have a sense-amplifier do not have measurable differences in terms of delay. Although this should seem obvious at first, there is one place where the change could affect non-sense-amplifier stages, and this is if there are any significant changes in output rise time of the stage such that it degrades the propagation of the next gate. In our case, though, the stage right after the sense amplifiers are the pipeline elements themselves, such that any change in output rise time is also incorporated into the bitline/sense-amp pipeline stage.

Run power results

Figure 4-105 shows the total power and its breakdown into dynamic power (switching power) and static power (dissipated by subthreshold leakage and gate leakage currents) for every point in the design space that was considered (i.e. cache size, associativity and technology node). Each plot shows both the results from myCACTI runs that assume either low-swing differential sense-amplifier or full-swing single-ended sense-amplifier circuits. Figure 4-106 demonstrates the differences more explicitly by plotting the fractional difference between the two runs normalized to the total power of the default myCACTI result. Figure 4-107 shows detailed power breakdowns of some representative design points in order to explore in detail the causes of the differences. In addition, the results from the two runs are placed side by side for easy comparison. Lastly, Figure 4-108 identifies how much power is being dissipated by major cache blocks, and again shows the results from the two runs side by side for easy comparison.

From the figures, we observe that using full-swing single-ended sense amplifiers instead of low-swing differential sense amplifiers results in significant degradation in terms of dynamic and total power for most of the configurations. This power difference is typically larger in the larger technology nodes -- this is reasonable since differential sensing techniques only require the discharge of a small, fixed voltage in the bitlines (largely regardless of technology), while full-swing single-sensing requires discharging the bitline completely, so

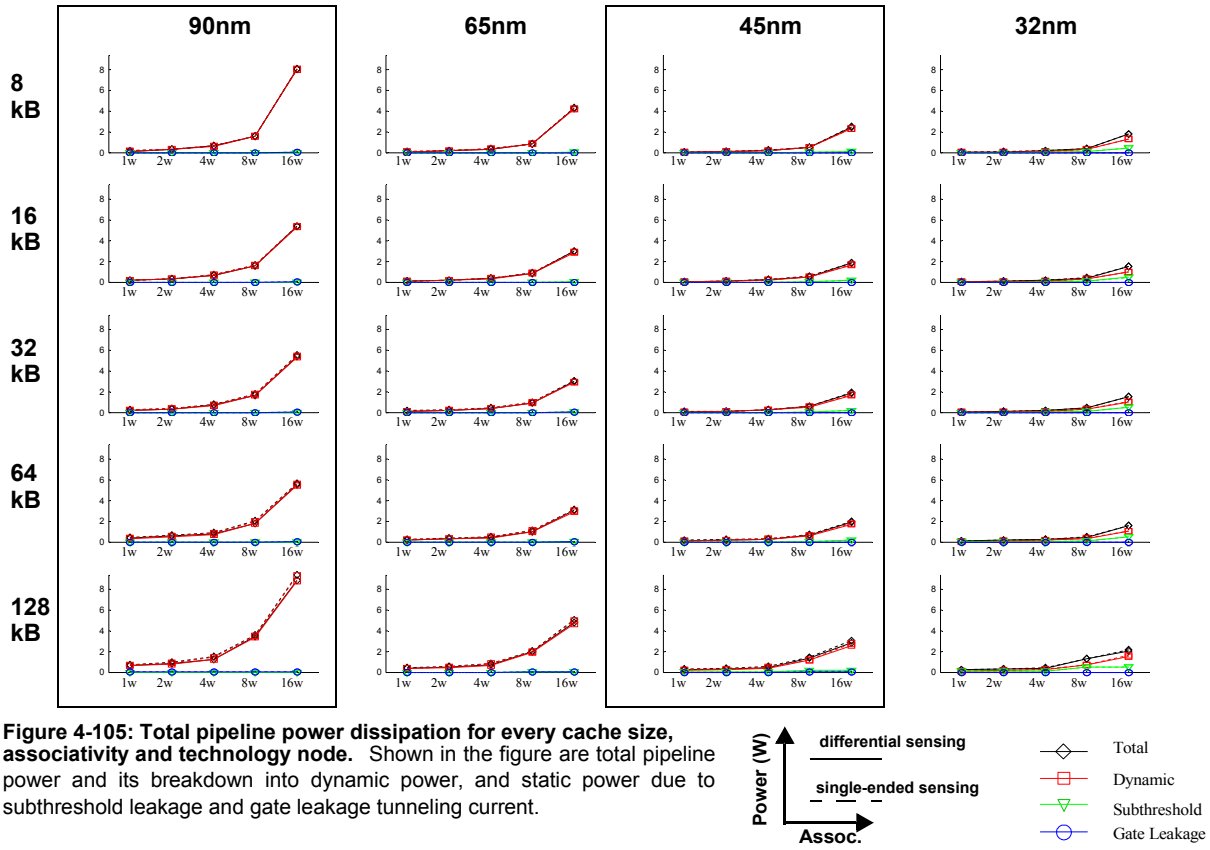


Figure 4-105: Total pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

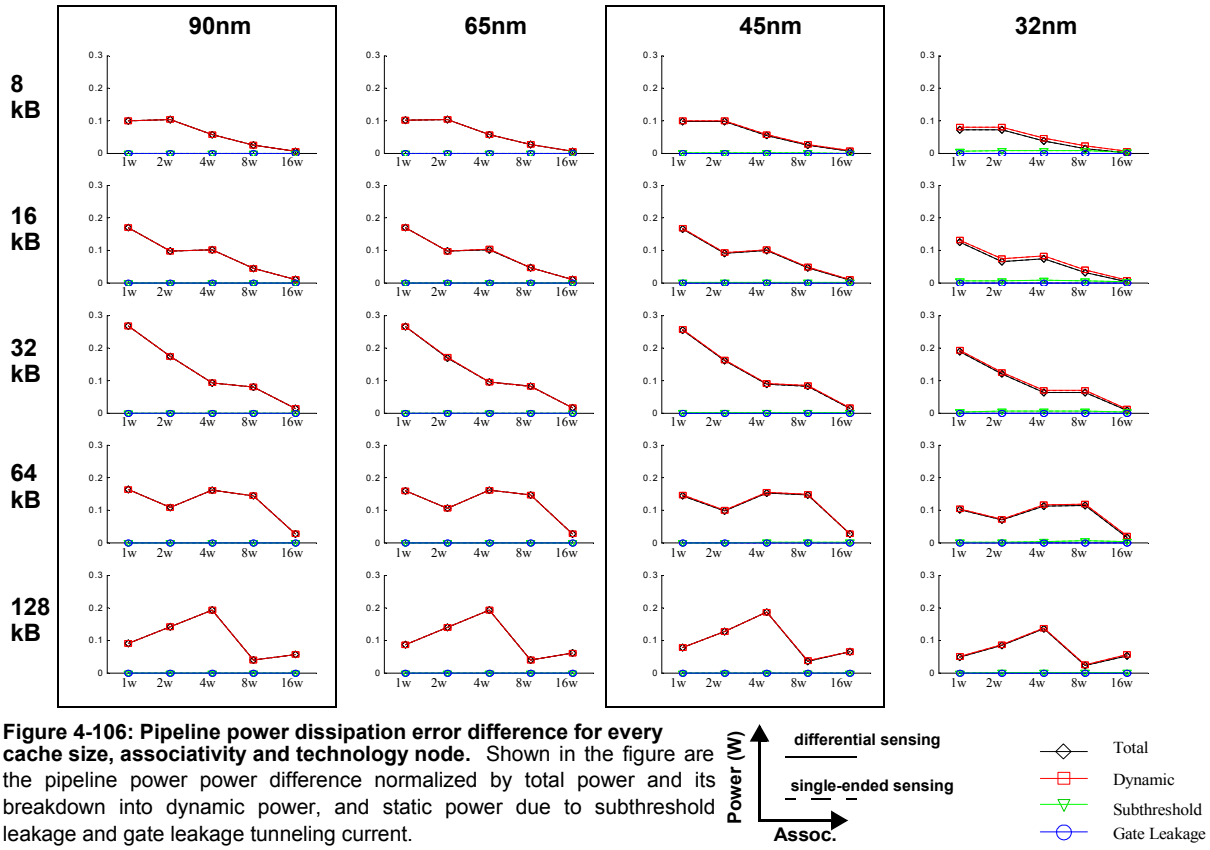


Figure 4-106: Pipeline power dissipation error difference for every cache size, associativity and technology node. Shown in the figure are the pipeline power power difference normalized by total power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

power is dependent on the supply voltage which, in the case of older technologies, is larger. In addition, dynamic power typically becomes smaller (lower Vdd and capacitances) while the subthreshold leakage

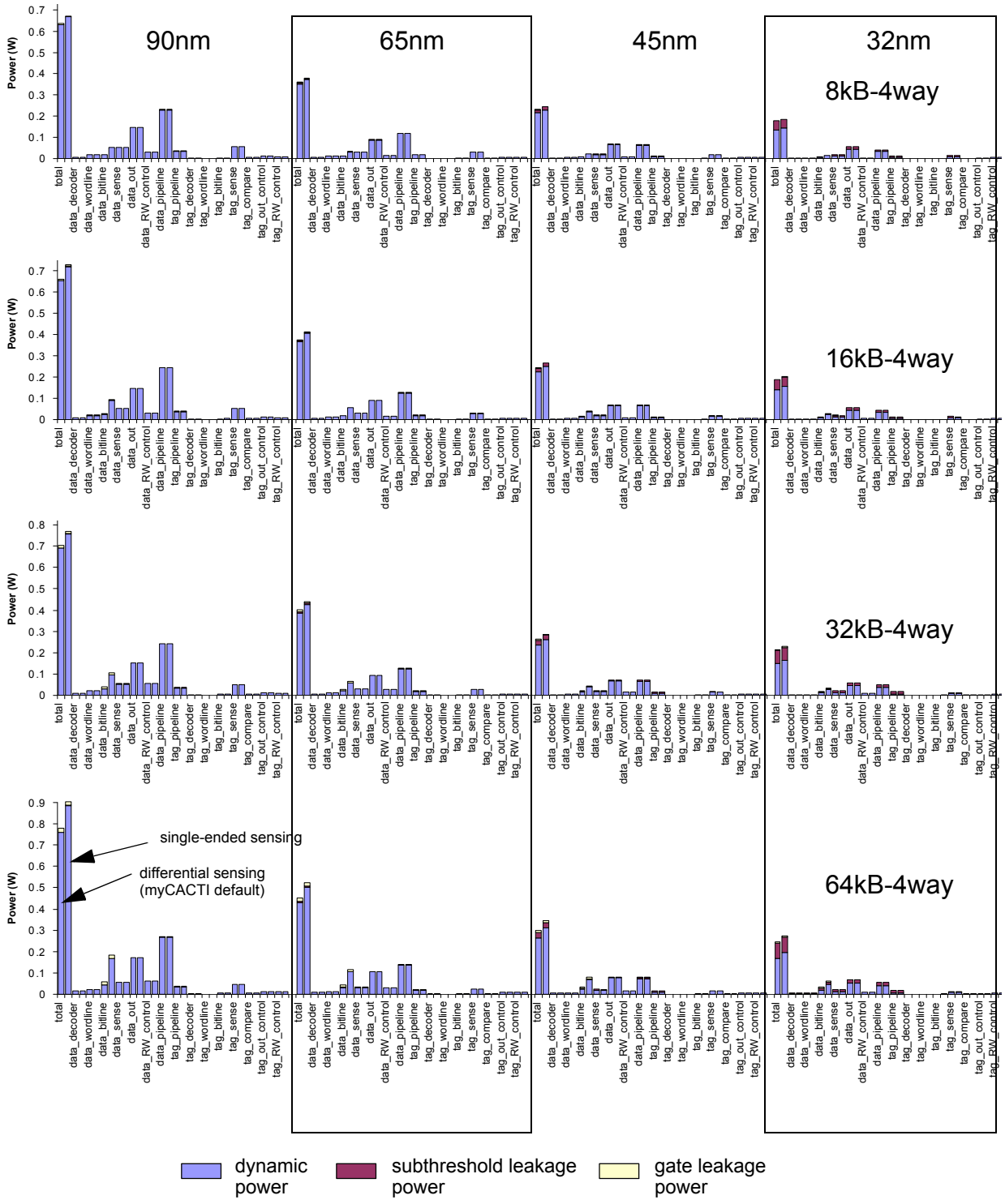


Figure 4-107: Detailed cache power breakdown comparison. Detailed power breakdown for four different cache configurations showing the amount of dynamic, subthreshold leakage and gate leakage power being consumed by different parts of the cache. Each plot shows four sets of values corresponding to different technology nodes. (Note that all three plots use the same scale). The two results are also placed side by side for direct comparison.

component becomes bigger such that the fractional contribution of the change in bitline power is reduced to some extent.

The typical trend with respect to cache associativity is that the difference becomes smaller with increasing associativities, again because higher associativities will typically have a fewer number of rows connected to the bitline such that the bitline capacitance is lower, resulting in a lower power difference compared to caches with lower associativities that tend to have a larger number of rows. This reasoning is supported by actual data, where configurations that have decreasing power with increasing associativities have monotonically decreasing number of rows with increasing associativity.

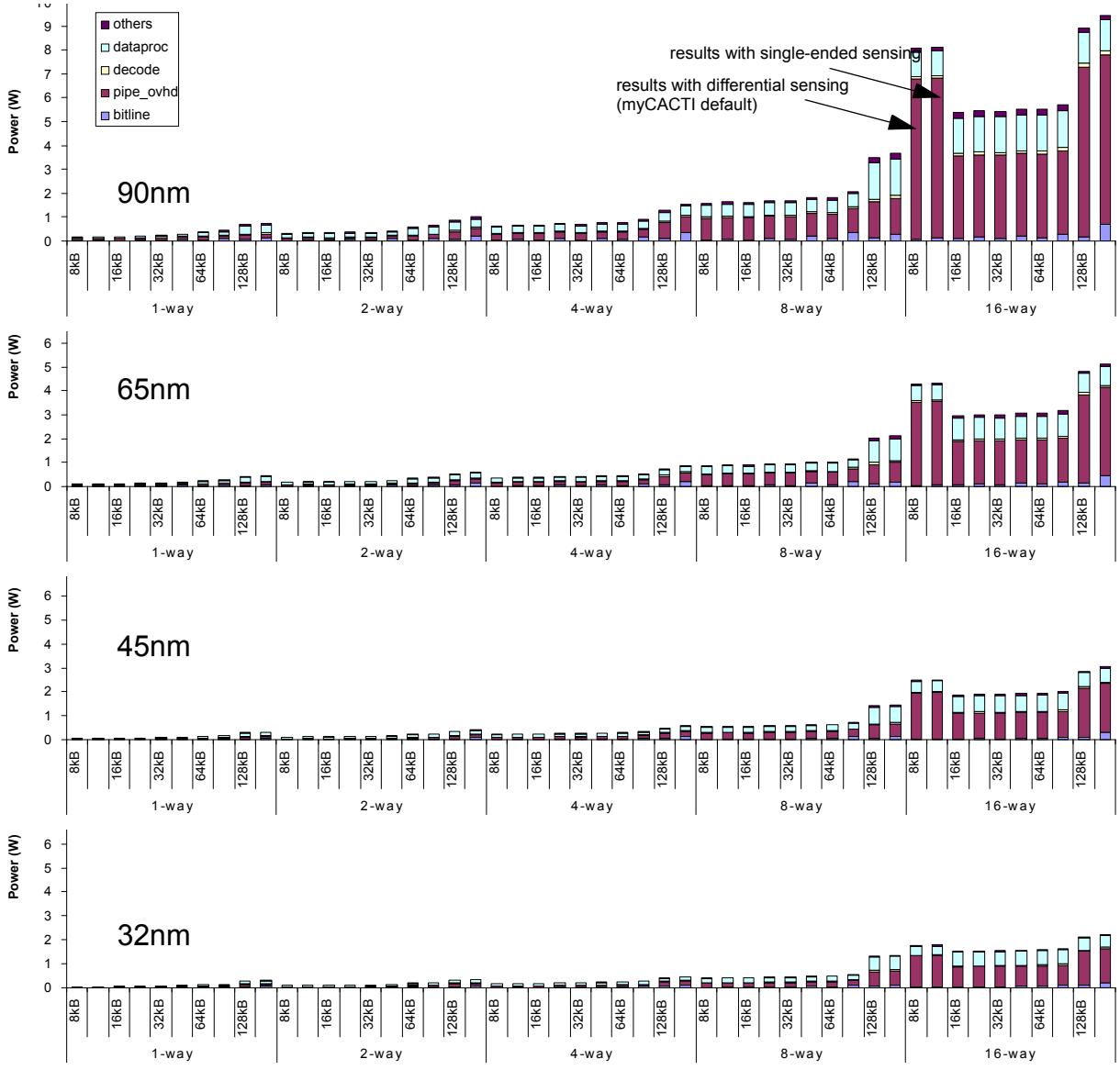


Figure 4-108: Power contributions of different cache parts. Power breakdown showing major cache power contributors for the 65nm and 32nm technology nodes for different cache sizes and associativities. (Note that the y-axis for both plots use the same scale). The two results are also placed side by side for direct comparison.

Looking at the error difference plots, it seems initially surprising that using full-swing single-ended sensing only results in at most 20% additional power compared to low-swing differential sensing. This is surprising since one could reasonably expect a larger difference in power (since discharging around 200mV for low-swing is much less than discharging 1.2V), but this does make sense after considering that in these technolgoy nodes, the dynamic power due to actively discharging the bitlines account for only a small part of the dynamic power (and even a much smaller part of total power). Consequently, although the dynamic power of the bitlines does increase significantly, the effect on the entire cache is not as much.

Looking at the detailed power breakdowns, again, as stated before, for the five configurations studied, the difference in power is greater for the larger technology nodes compared to the smaller ones. In addition, virtually all of these differences are accounted for by dynamic power in the bitlines. This can be seen from the plots where it is only the bitline power component that shows the difference. In addition, the tag bitline only shows visible difference for the larger tech nodes, as it is a much smaller array compared to the data part, and that the number of rows are typically fewer (typically around 16)

For 64kB-4way, the power increase when going from diff to single-ended sensing is about 20% for 90nm, 17% for 45nm, and 13% for 32nm.

It is important to note that differential sensing assumes a reliable self-timing scheme such that the circuit can cutoff the wordlines precisely when enough differential has been developed in the bitlines to allow proper sensing. If we account for flaws in design, PVT variation, margining, and the different times signals reach different points (such that it's difficult to synchronize everything inside the entire array perfectly and it's possible that a wordline can be turned OFF at one point in the route while still being turned ON at some other point because of the propagation delays of the signal through a simple interconnect), it is impossible to reach this ideal situation. Consequently, cache bitline power for differential sensing will typically be higher than the numbers that we show here as the results of our simulations. This does not hold true for single-ended sensing, where it is already assumed that the worst case power dissipation across the bitline will occur (since the entire bitline is being discharged). As such, the numbers for single-ended sensing that we give here are close to what it would be in a real, practical implementation.

Conclusion

At first glance, shifting from low-swing differential to full-swing single-ended sensing seems to be unattractive in that it results in a significant increase in power dissipation (for all configurations), and an even larger increase in access time (for the typical configuration). It would be easy to dismiss the full-swing single-ended sensing technique based on these two criteria.

Other concerns exist, though, that have to be considered before concluding whether this could be a passable technique. With single-ended sensing, only a single-ended bitline signal needs to be routed instead of

a differential bitline signal. With memory arrays often being metal-limited, reducing the track requirements by a single metal route may result in significant area savings. In addition, a single-ended signal has the added benefit of reduced design complexity and verification. Although differential bitline signals have the advantage of typically being faster by virtue of being low-swing and having to discharge a smaller voltage, it comes with the penalty of the need to verify the differential property of these two signals. Instead of a single noise analysis performed on a single-ended bitline, common-mode noise analysis and differential-mode analysis have to be performed on the two-bitlines, making the verification of the design harder and more complicated. In addition, as mentioned earlier, the bitlines are actually margined to provide more differential, such that typical differential sensing numbers that we give are optimistic. In addition, differential noise analysis becomes harder to pass as supply voltage becomes smaller, since increasingly smaller noise values may already serve to cause upsets.

In summary, as the supply voltage gets smaller and smaller, the advantage of small-swing differential bitlines gives us diminishing returns. Coupled with the observation that the delay and power penalties of single-ended sensing gets smaller and smaller for deep nanometer technolgoy nodes, and that the difficulty in designing small-swing differential bitlines becomes much harder, we can conclude that single-ended sensing might not be practical for 90nm and 65nm because of the mentioned degradation in delay and power, but are starting to become more and more attractive with each generational upgrade.

4.5.10 BEOL low-k study

Run details

To produce these data, myCACTI was made to do a design space exploration using the default settings of myCACTI (in particular, using a base value of 3.0 for the ILD dielectric constant) to find the optimal implementation for every cache size and associativity for the 90nm technology node. After determining the optimal implementations for each point in the 90nm design space (in terms of the cache parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl), three more sets of runs are performed for each technology node, with each point in the design space utilizing the optimal implementation that was determined for the 90nm node. In essence, the data for 65nm, 45nm and 32nm constitute a sort of "processshrink" where we take a design in an older process technology and convert it to a newer one. Although the runs for 65nm and below will not show the absolute optimal implementation possible, this method was chosen to provide a simple but valid comparison between different technology nodes. Implementing the same cache configuration (i.e. same cache size and associativity) using two different implementations for different technology nodes would make the comparisons between different cache components very difficult and generalizations will be hard to make.

After generating a complete set of data using myCACTI default settings, the process is repeated, this time with each myCACTI run being set to use lowK ILD of 1.0 (by using the -use_lowK option).

Run timing results

Figure 4-109 shows the delay results for all cache configurations for both runs where results are generated with the assumption of a base interlayer dielectric (ILD) dielectric constant (base or default K) (the myCACTI default, shown in the plots as the solid lines), and low-K ILD (shown as dashed lines). This figure shows the resulting optimal clock period for each configuration, along with the pipe delays for all of the stages in the cache (with the exception of pipe1A_data and pipe1B_data -- the initial decode stages -- since these always have significant slack so they have been omitted from the figure for clarity). Figures 4-110 to Figure 4-116 show the delay component for each individual stage. These individual plots provide a breakdown of the delay of each individual stage by showing the component delays of each. Lastly, Figure 4-117 shows the unpipelined cache access time.

The comparison plots show that using an ILD dielectric constant that is significantly lower than the default (in our case from 3.0 to 1.0) produces significant improvement in delay, especially for highly associative caches where the data output stage typically becomes the critical path because of the need to drive the data from the internals of the cache to the cache periphery. With an improvement in the interconnect, this operation becomes more efficient and may potentially remove the data out path from the critical path. Also, we can see the the configurations that did not see an improvement in the total clock delay are ones with the

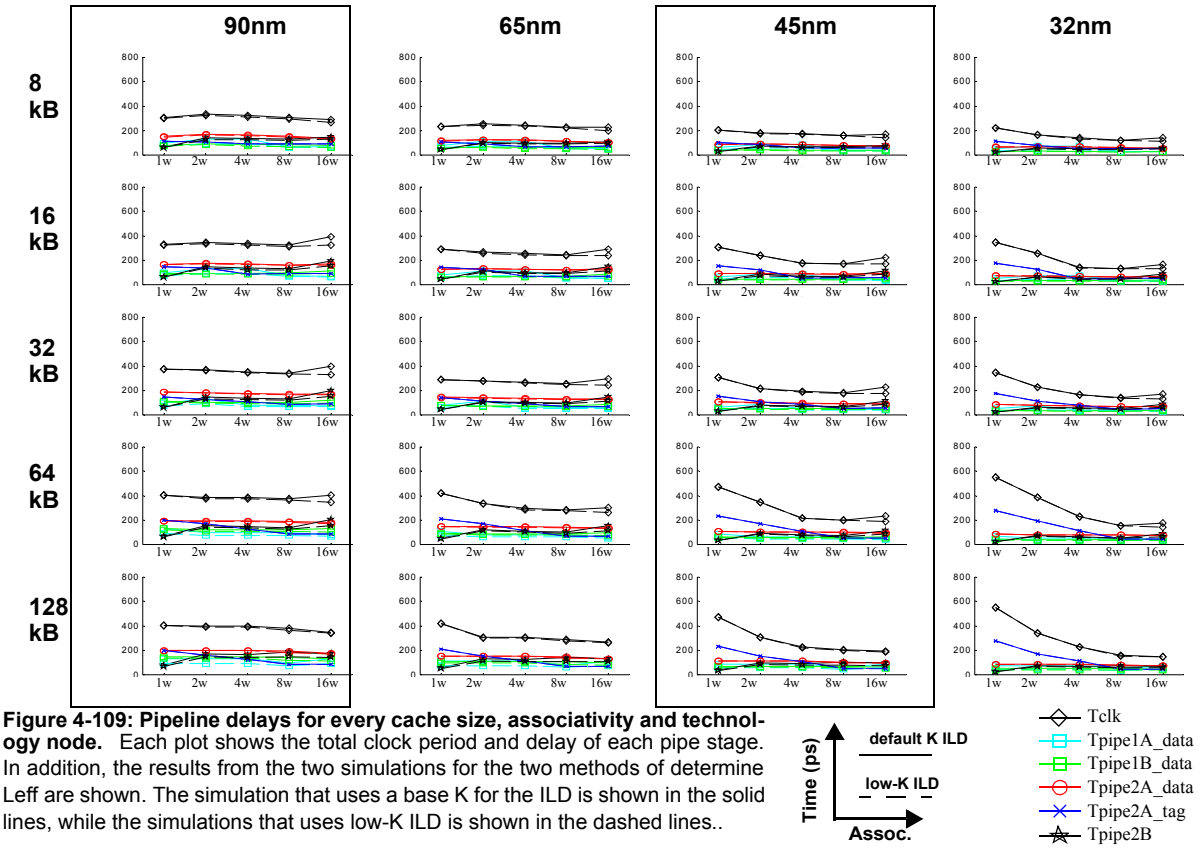


Figure 4-109: Pipeline delays for every cache size, associativity and technology node. Each plot shows the total clock period and delay of each pipe stage. In addition, the results from the two simulations for the two methods of determine Leff are shown. The simulation that uses a base K for the ILD is shown in the solid lines, while the simulations that uses low-K ILD is shown in the dashed lines..

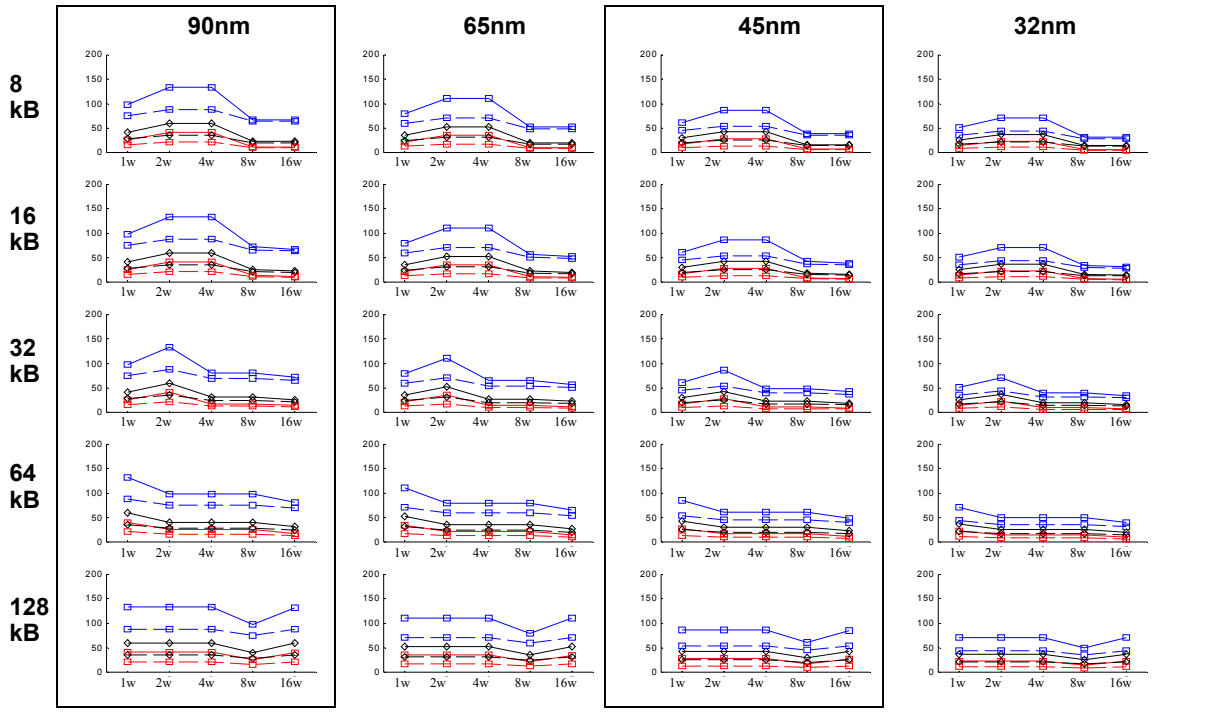


Figure 4-110: Delay components for pipeline stage pipe1A_data . This pipeline stage consists of data address buffering and the predecoder front-end. The solid-lines denote default myCACTI numbers that assume the use of a base K for the ILD, while the dashed lines denote numbers that assumed a low-K ILD process.

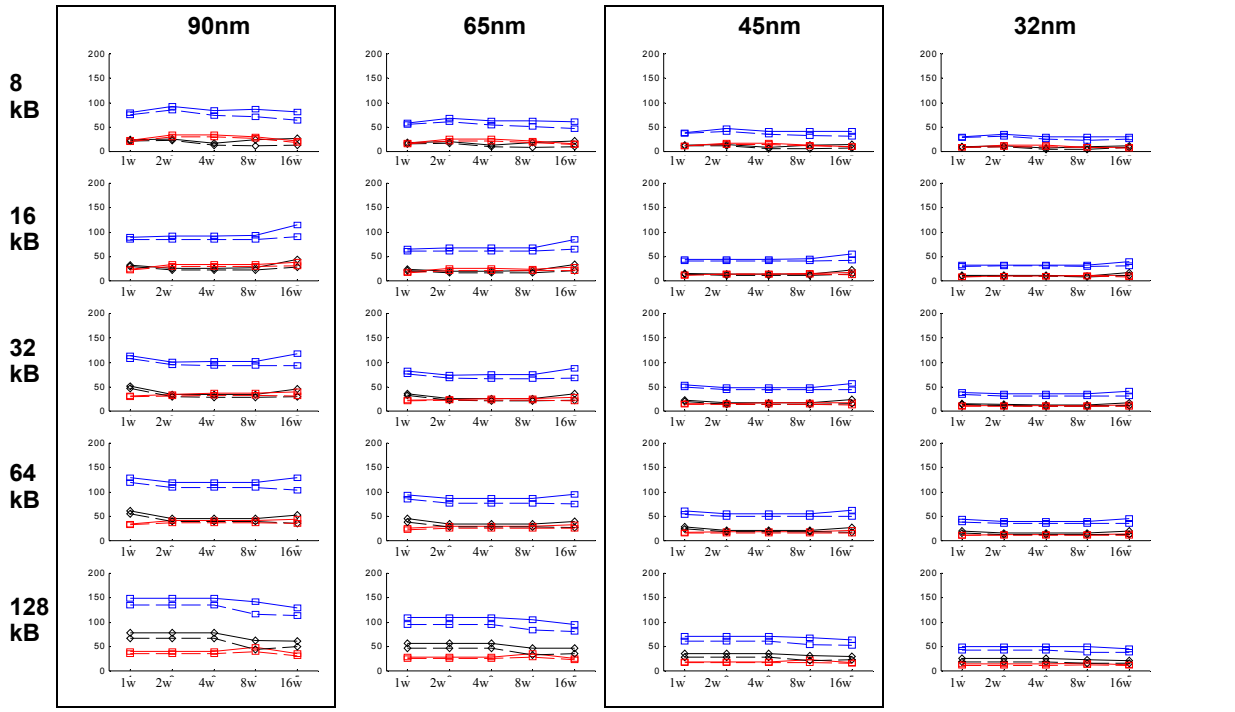


Figure 4-111: Delay components for pipeline stage pipe1B_data . This pipeline stage consists of the data predecoder driver and the wordline front-end. The solid-lines denote default myCACTI numbers that assume the use of a base K for the ILD, while the dashed lines denote numbers that assumed a low-K ILD process.

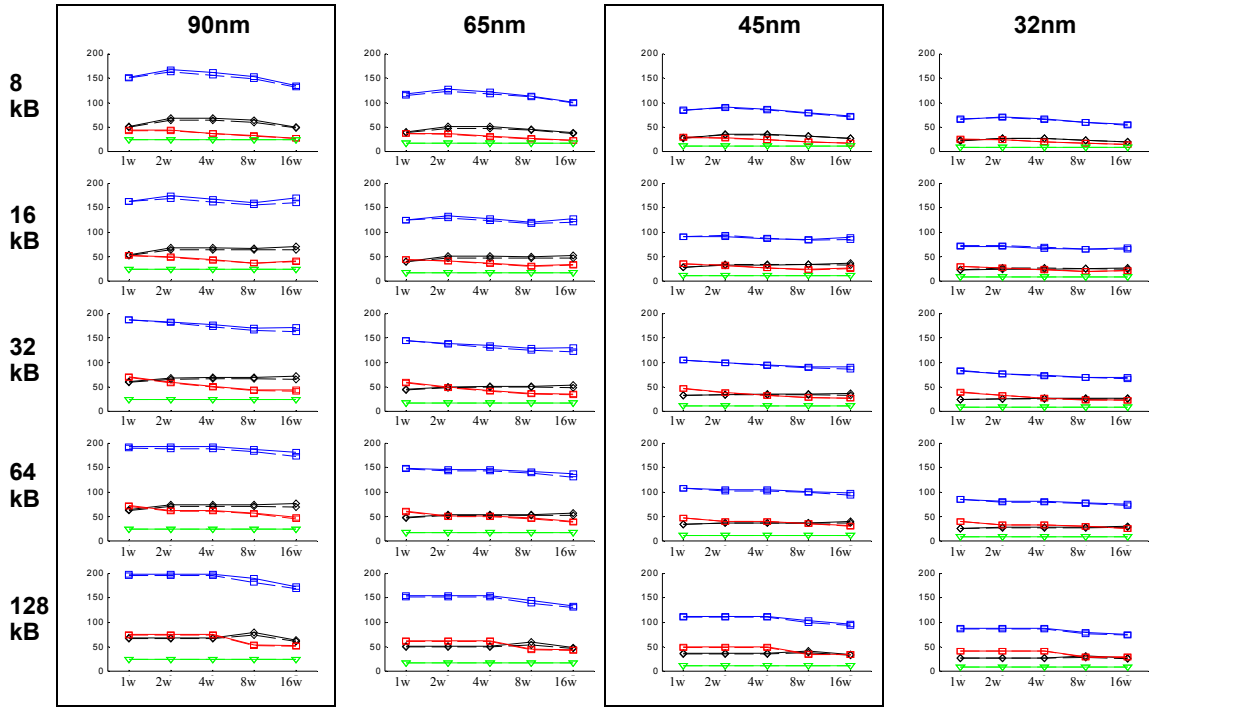


Figure 4-112: Delay components for pipeline stage pipe2A_data . This pipeline stage consists of the wordline drivers, the bitline and the sense amplifier. The solid-lines denote default myCACTI numbers that assume the use of a base K for the ILD, while the dashed lines denote numbers that assumed a low-K ILD process.

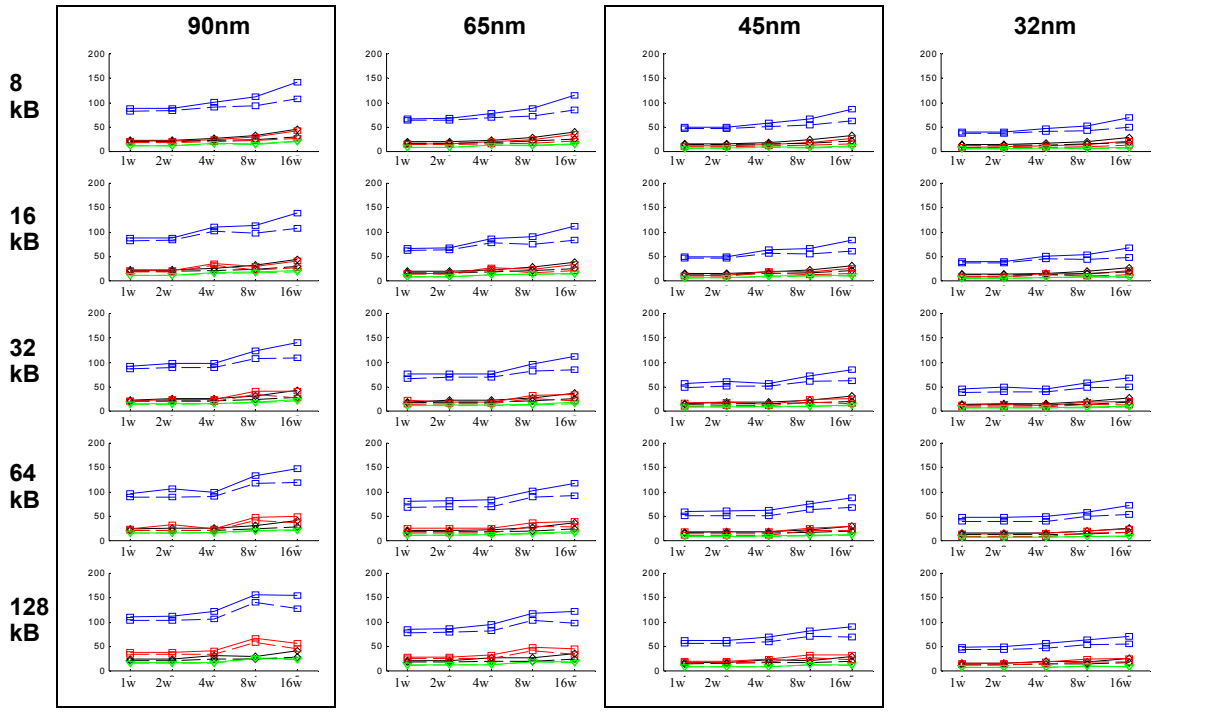


Figure 4-113: Delay components for pipeline stage pipe1A_tag. This pipeline stage consists of tag address buffering, predecoding and the wordline front-end. The solid-lines denote default myCACTI numbers that assume the use of a base K for the ILD, while the dashed lines denote numbers that assumed a low-K ILD process.

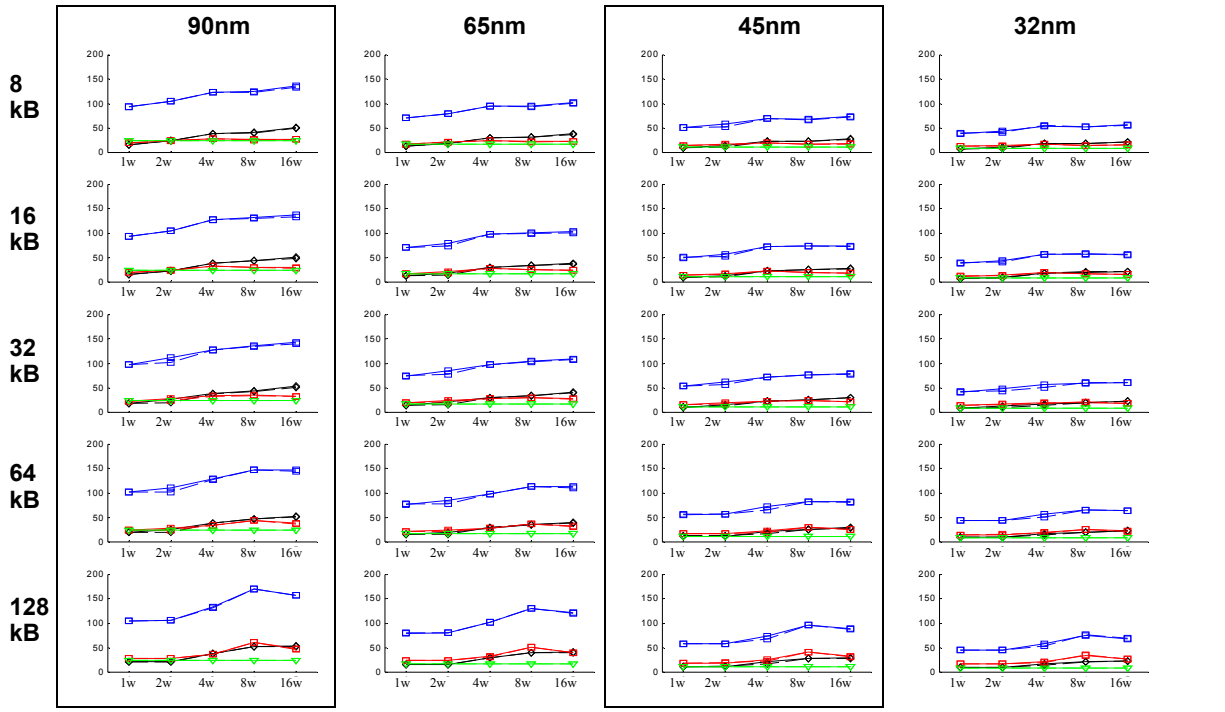


Figure 4-114: Delay components for pipeline stage pipe1B_tag . This pipeline stage consists of the tag wordline drviers, bitline delay and senseamp delay. The solid-lines denote default myCACTI numbers that assume the use of a base K for the ILD, while the dashed lines denote numbers that assumed a low-K ILD process.

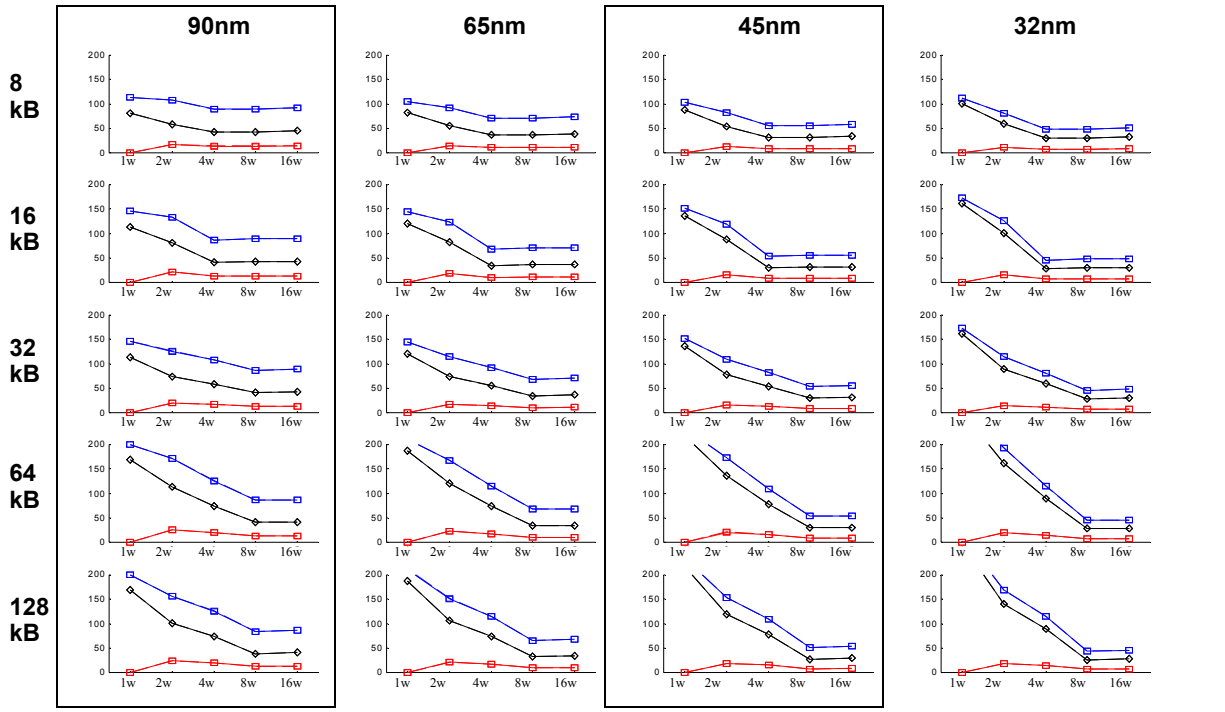


Figure 4-115: Delay components for pipeline stage pipe2A_tag . This pipeline stage consists of enabling the comparator and the actual comparator delay. The solid-lines denote default myCACTI numbers that assume the use of a base K for the ILD, while the dashed lines denote numbers that assumed a low-K ILD process.

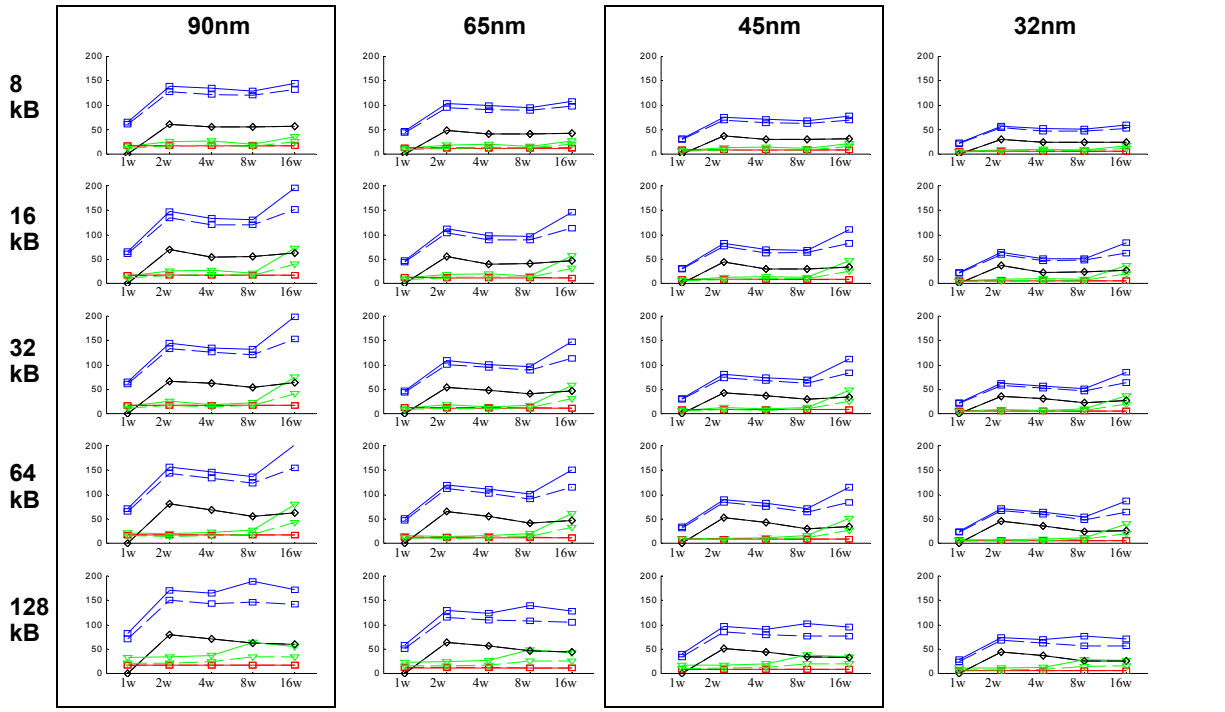


Figure 4-116: Delay components for pipeline stage pipe2B . This pipeline stage consists of driving the output buffer selects and the final data output drive to the cache periphery. The solid-lines denote default myCACTI numbers that assume the use of a base K for the ILD, while the dashed lines denote numbers that assumed a low-K ILD process.

compare stage as the critical path, since this stage is largely independent of interconnect RC. Configurations that have the data wordline-bitline-sense as their critical path see roughly around 10ps improvement in delay, while configurations that have the data output stage as the critical path see roughly 40ps to 60ps of delay improment. Although just by looking at the data output stage we see some configurations improving by up to 45ps (which, considering that these are just phase paths, should translate to a 90ps improvement in the total clock period), this much improvement typically results in some other pipe stage becoming more criticial, such that we don't enjoy the entire delay improvent.

Looking at improvements in the delay for the address buffer and predecoders (around 20ps each), it is curious why the improvement to the wordline driver and the bitlines are not as large. The best explanation for this is that for the address buffers and the predecoder, the interconnect lengths are still short enough such that resistance is not dominant and that improving the capacitance results in large gains. For the wordlines (which have a large number of columns, resulting in long wordlines and higher interconnect resistance) and for bitlines (with weak effective transistor drive strength so the effective resistance is larger), the resistance is high enough to cause significant resistive isolation such that improving the interconnect capacitance does not result in as large a gain.

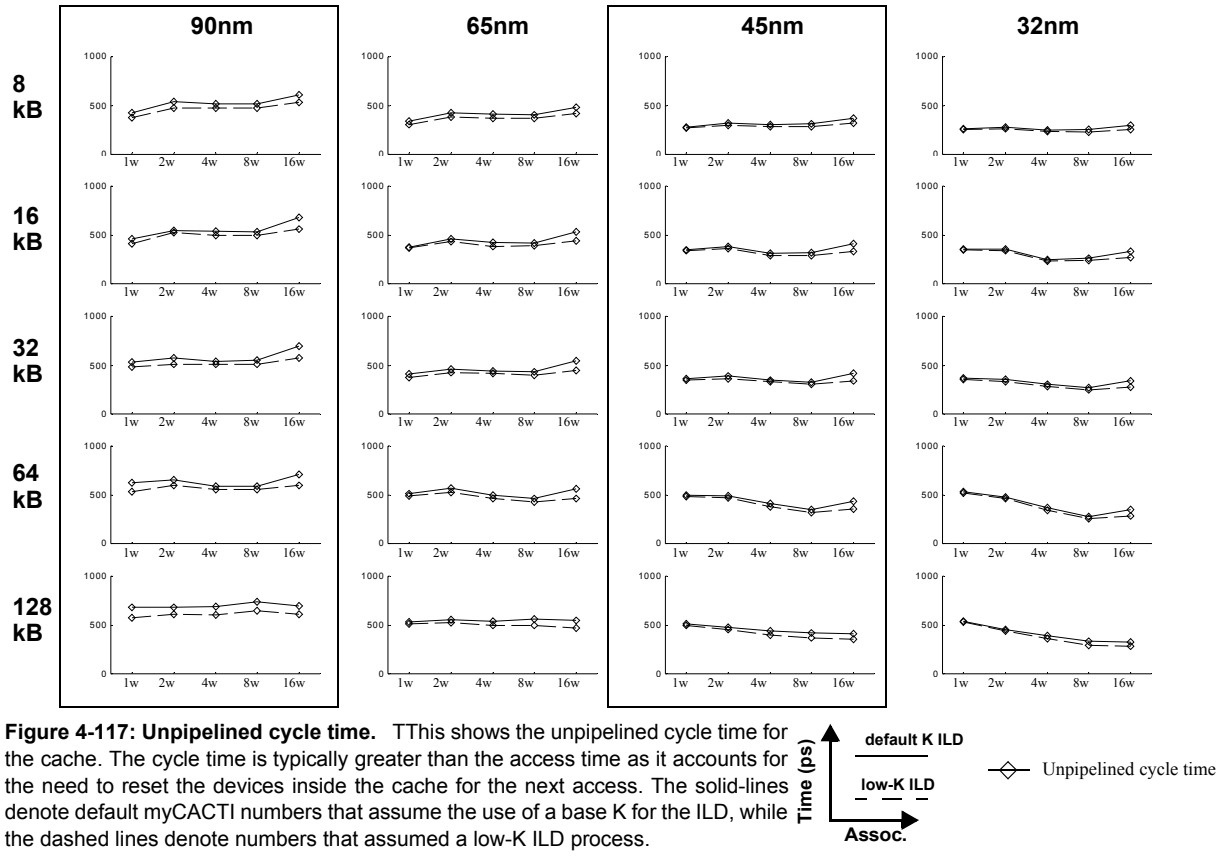


Figure 4-117: Unpipelined cycle time. TThis shows the unpipelined cycle time for the cache. The cycle time is typically greater than the access time as it accounts for the need to reset the devices inside the cache for the next access. The solid-lines denote default myCACTI numbers that assume the use of a base K for the ILD, while the dashed lines denote numbers that assumed a low-K ILD process.

Run power results

Figure 4-118 shows the total power and its breakdown into dynamic power (switching power) and static power (dissipated by subthreshold leakage and gate leakage currents) for every point in the design space that was considered (i.e. cache size, associativity and technology node). Each plot shows both the results from myCACTI runs that assume either a base K or a low-K ILD. Figure4-119 demonstrates the differences more explicitly by plotting the fractional difference between the two runs normalized to the total power of the default myCACTI result. Figure 4-120 shows detailed power breakdowns of some representative design points in order to explore in detail the causes of the differences. In addition, the results from the two runs are placed side by side for easy comparison. Lastly, Figure 4-121 identifies how much power is being dissipated by major cache blocks, and again shows the results from the two runs side by side for easy comparison.

From the figures, we can see that using low-K dielectrics for the interconnect can result in significant improvements in power. Based on the error plots, using low-K interconnects typically results in 3% to 30% power savings (with the typical being roughly 20%) compared to the base K. This is due to significantly smaller interconnect cap (i.e. 1/3 since defK = 3*lowK).

The error difference we observe is also significantly higher in low associative caches (e.g. 1 or 2 way) compared to highly associative caches. Again, this makes sense becuse highly associative caches introduce dynamic power that is only partly dependent on interconnect capacitance, with a significant part being largely

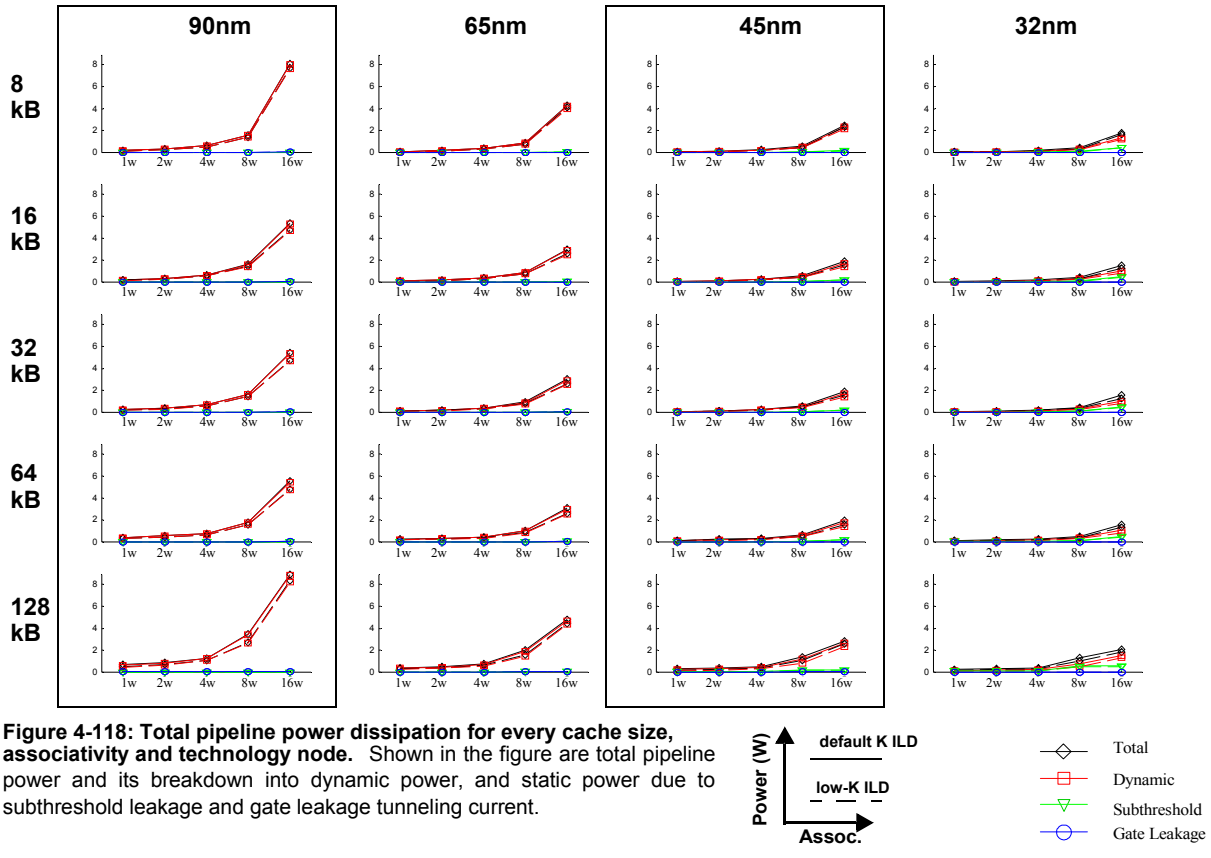


Figure 4-118: Total pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

independent of it (e.g. the tag comparators). With interconnect switching power staying roughly the same, its fraction of the total power goes down. Of course, the flip side of this is that with higher associativiities, some components of the cache that are dependent on interconnect capacitance also increases (mainly the data output -- higher associativities make this component larger). Even though the bitline power should, at first glance, increase because of the more bitlines being read in parallel, the number of rows per bitline decreases with increasing associativity (assuming that the cache size stays constant, of course), such that the bitline power does not fluctuate significantly. Consequently, there is a "sweet spot" in associativity where the power peaks.

An interesting behavior that we observe is the existence of power differences because of additional power dissipated by subthreshold leakage currents when going from the base K to low K. All other things being equal, if the only difference between two circuits is the interconnect dielectric constant, this should not affect the subthreshold leakage as this is not dependent on the amount of capacitive loading of a node. What actually happens is an artifact of the tool's operation, where the data output driver size of the configuration is computed at runtime in order to optimally drive whatever load is presented. Using low-K dielectrics means that the data output driver is presented with a significantly smaller load such that it has less to drive. Consequently, the output drivers are sized smaller than they would be for the base K, resulting in not just a decreased dynamic power, but also decreased power dissipated by subthreshold leakage currents because of the smaller leakage current flowing across the smaller transistors. This can be demonstrated by looking at the

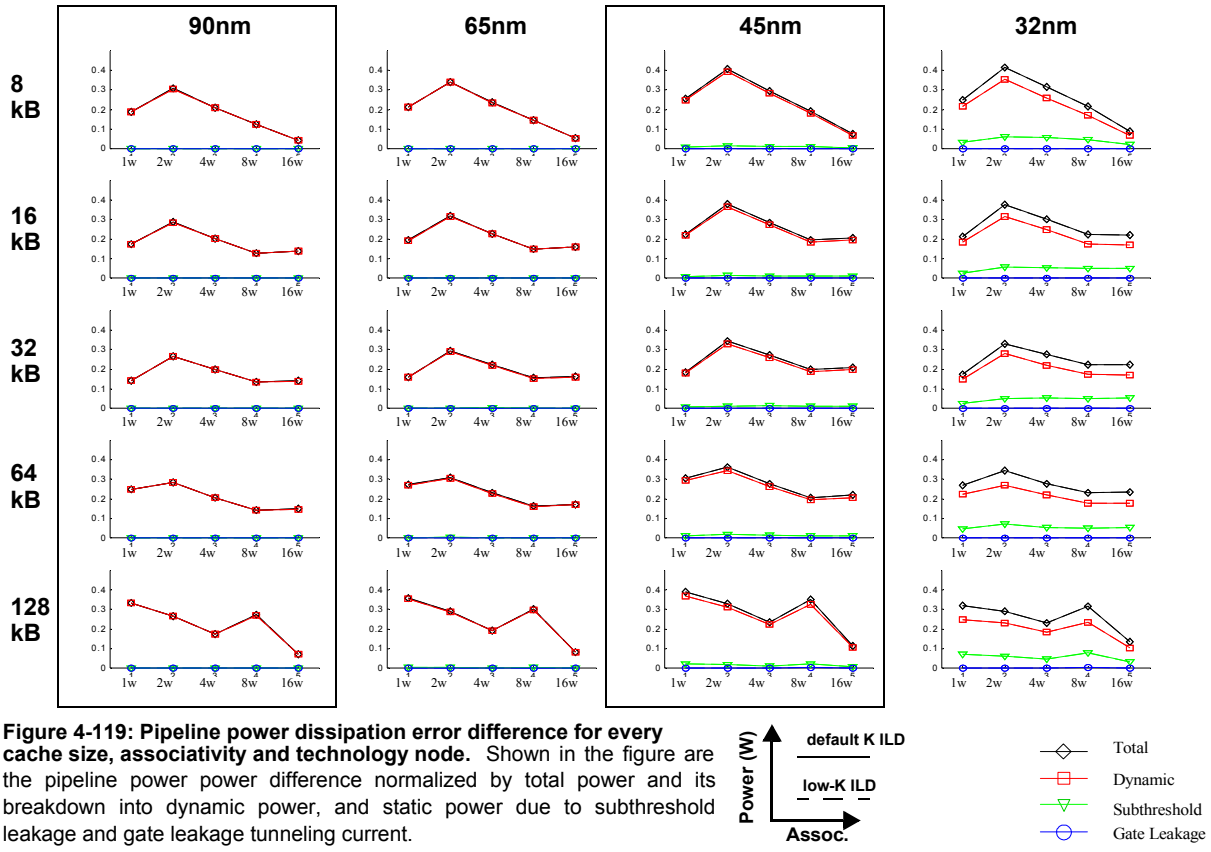


Figure 4-119: Pipeline power dissipation error difference for every cache size, associativity and technology node. Shown in the figure are the pipeline power power difference normalized by total power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

component breakdown of power, where it is shown that the subthreshold leakage component of the data output drivers are seen to be significantly reduced when going from the base K to a low-K configuration. This

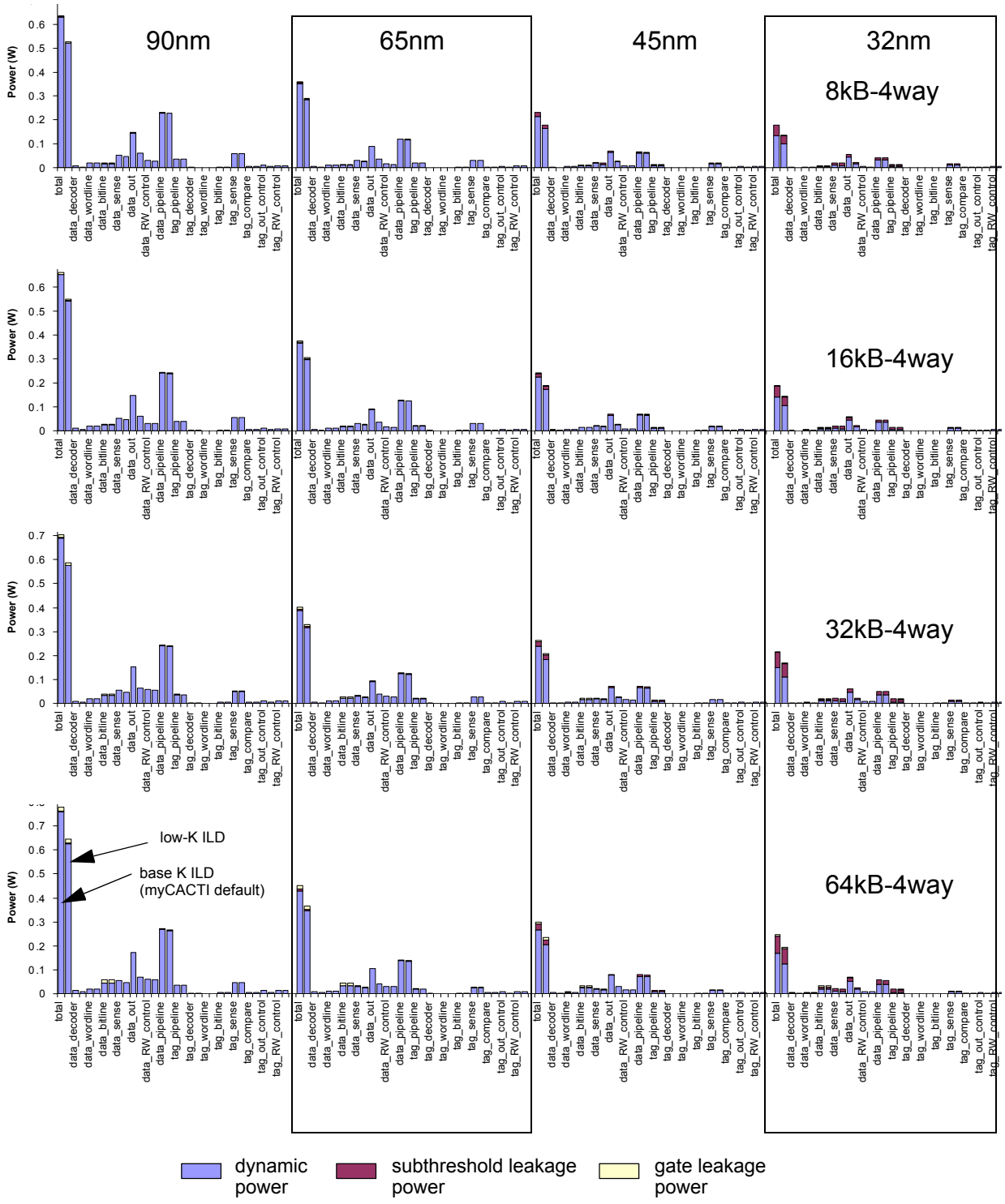


Figure 4-120: Detailed cache power breakdown comparison. Detailed power breakdown for four different cache configurations showing the amount of dynamic, subthreshold leakage and gate leakage power being consumed by different parts of the cache. Each plot shows four sets of values corresponding to different technology nodes. (Note that all three plots use the same scale). The two results are also placed side by side for direct comparison.

accounts for a significant amount of the difference in subthresold leakage indicated earlier. Of course, dynamic power for the data output driver is also reduced as a result of lower interconnect capacitance load, as well as smaller drain capacitance loading. It should be noted, though, that this observation is mainly significant only for the 45nm and the 32nm nodes where the subthreshold leakage current plays a big factor.

Of the change in total power, we can see from the detailed power breakdowns that most of this difference is accounted for by the data out stage -- although there are some variations with some of the other components (e.g. decoder, bitline, wordline, pipeline overhead), it is mostly the data out circuits that account

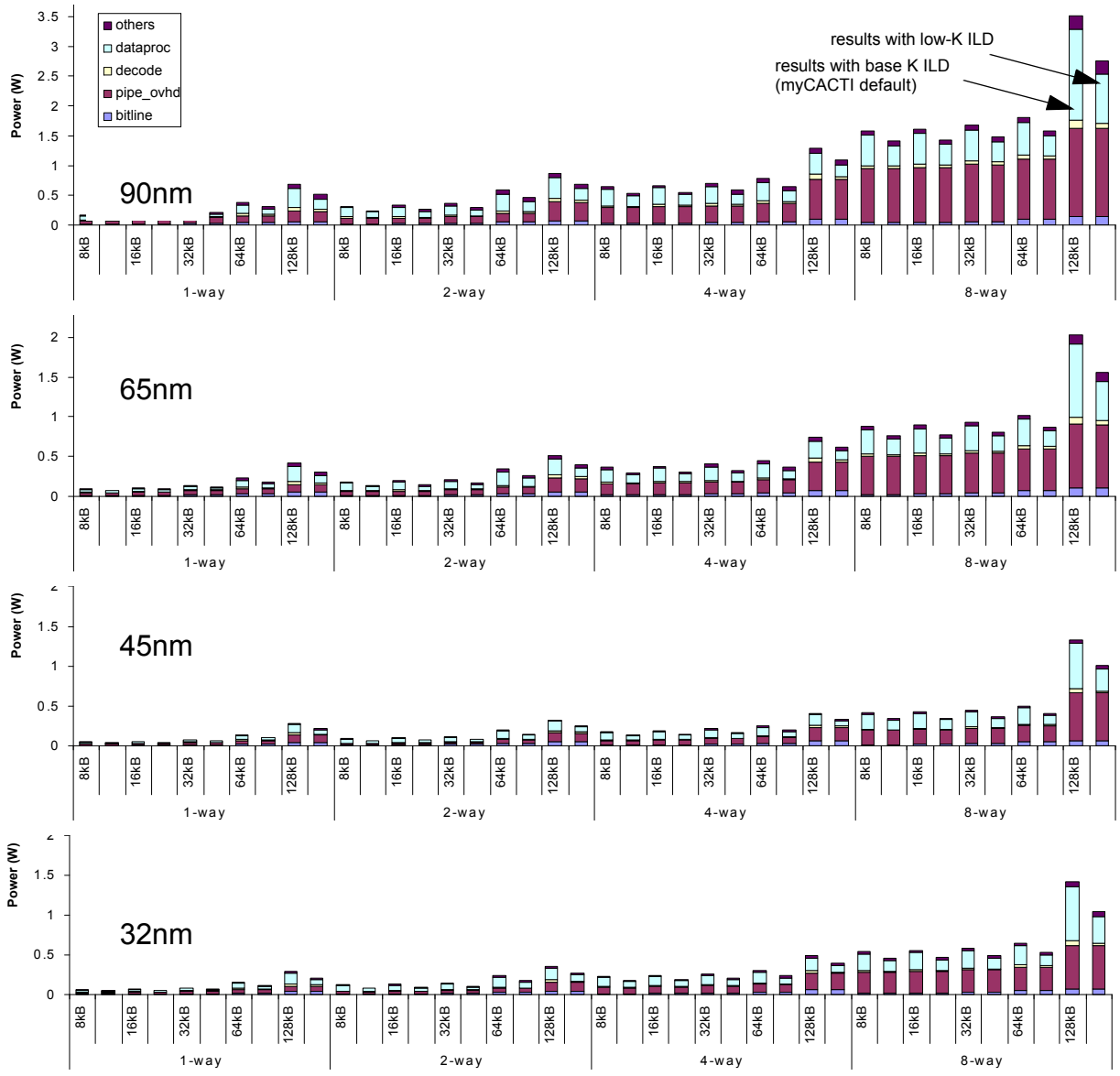


Figure 4-121: Power contributions of different cache parts. Power breakdown showing major cache power contributors for the 65nm and 32nm technology nodes for different cache sizes and associativities. (Note that the y-axis for both plots use the same scale). The two results are also placed side by side for direct comparison.

for the difference, although in some cases (specifically the 90nm nodes), the decoder also accounts for a significant part of the difference (but still a much smaller fraction compared to the data output driver).

Conclusion

We have demonstrated that the power advantages when going from a base ILD K of about 3.0 to a low-K ILD (of about 1.0) results in very significant power savings (roughly around 20%), and delay improvements. We demonstrate that the delay improvements are very significant for configurations that have their critical path going through stages that are very dependent on interconnect performance (like the data wordline-bitline-sense stages and the data out driver stage) and will typically provide around 20ps-60ps delay improvement.

Circuit-wise, there are no disadvantages involved in using low-K dielectrics for the interconnects, and the main problems that prevent low-K from being more prevalent in contemporary circuits are all process-based and are concerned with how easily and reliably they are integrated into an existing process.

4.5.11 Combination of multiple parameters

Run details

To produce these data, myCACTI was made to do a design space exploration using the default settings of myCACTI to find the optimal implementation for every cache size and associativity for the 90nm technology node. After determining the optimal implementations for each point in the 90nm design space (in terms of the cache parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl), three more sets of runs are performed for each technology node, with each point in the design space utilizing the optimal implementation that was determined for the 90nm node. In essence, the data for 65nm, 45nm and 32nm constitute a sort of "process-shrink" where we take a design in an older process technology and convert it to a newer one. Although the runs for 65nm and below will not show the absolute optimal implementation possible, this method was chosen to provide a simple but valid comparison between different technology nodes. Implementing the same cache configuration (i.e. same cache size and associativity) using two different implementations for different technology nodes would make the comparisons between different cache components very difficult and generalizations will be hard to make.

After generating a complete set of data using myCACTI default settings, the process is repeated, this time with each myCACTI run being configured to emulate CACTI/eCACTI operation by using static decoding, disabling via capacitances, ignoring wordline resistance, using Leff = Ldrawn, and using single-layer interconnects.

Run timing results

Figure 4-122 shows the delay results for all cache configurations for both runs where results are generated with the assumption of the base config (the myCACTI default, shown in the plots as the solid lines), and enabling the parameters earlier mentioned(shown as dashed lines). This figure shows the resulting optimal clock period for each configuration, along with the pipe delays for all of the stages in the cache (with the exception of pipe1A_data and pipe1B_data -- the initial decode stages -- since these always have significant slack so they have been omitted from the figure for clarity). Figures 4-123 to Figure 4-129 show the delay component for each individual stage. These individual plots provide a breakdown of the delay of each individual stage by showing the component delays of each. Lastly, Figure 4-130 shows the unpipelined cache access time.

From the comparison plots, we can see that runs that incorporate some of the previous changes that have been detailed (and now myCACTI defaults) show significant differences compared to runs that assume CACTI/eCACTI-like operation. Looking at the design space plots, most of the significant changes seem to be caused by delay degradation in the output stage. From the earlier experiments, this is mostly a result of the single-layer metal, forcing the data output to be routed to the cache using poor resistivity interconnect.

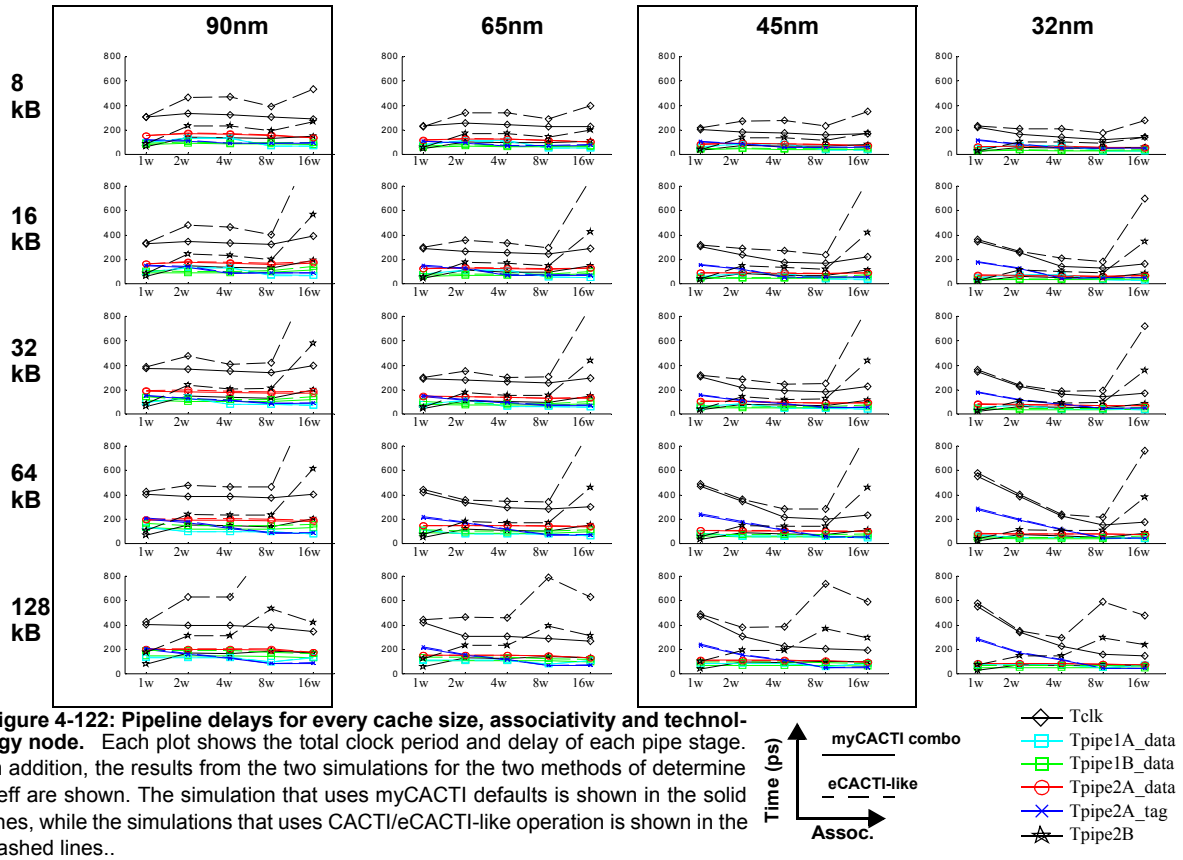


Figure 4-122: Pipeline delays for every cache size, associativity and technology node. Each plot shows the total clock period and delay of each pipe stage. In addition, the results from the two simulations for the two methods of determine Leff are shown. The simulation that uses myCACTI defaults is shown in the solid lines, while the simulations that uses CACTI/eCACTI-like operation is shown in the dashed lines..

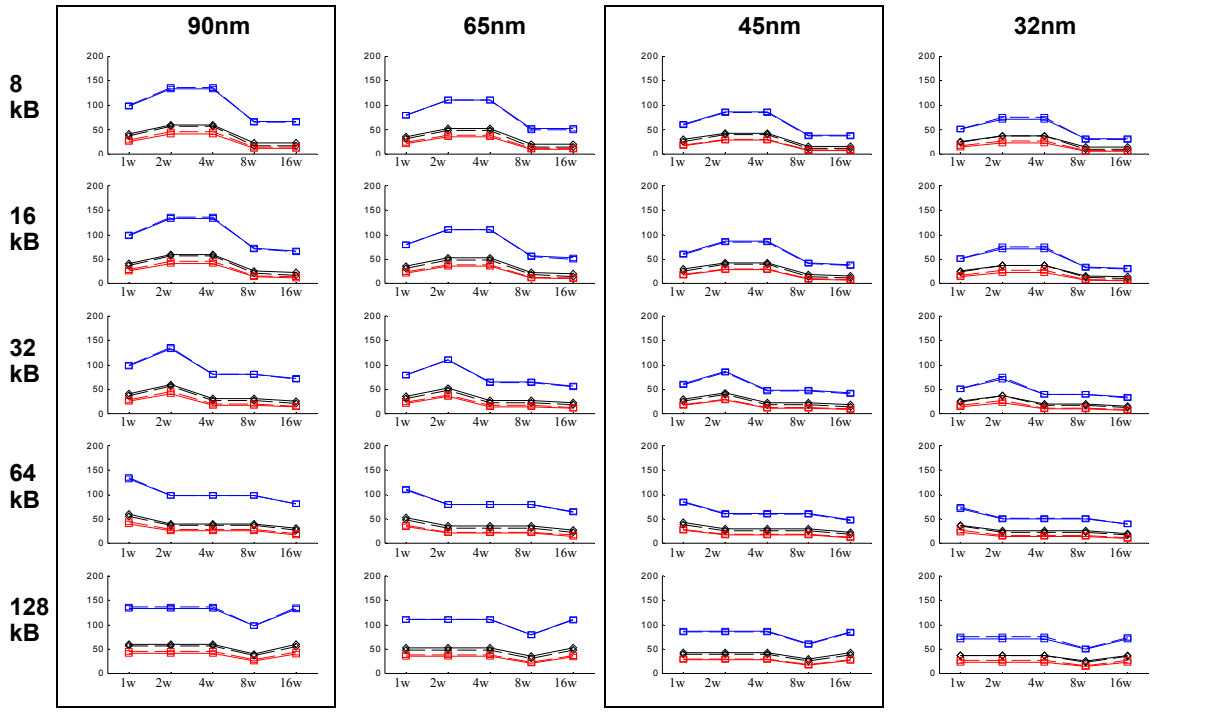


Figure 4-123: Delay components for pipeline stage pipe1A_data . This pipeline stage consists of data address buffering and the predecoder front-end. The solid-lines denote default myCACTI runs that enable multiple new features , while the dashed lines denote numbers that assumed CACTI/eCACTI-like operation

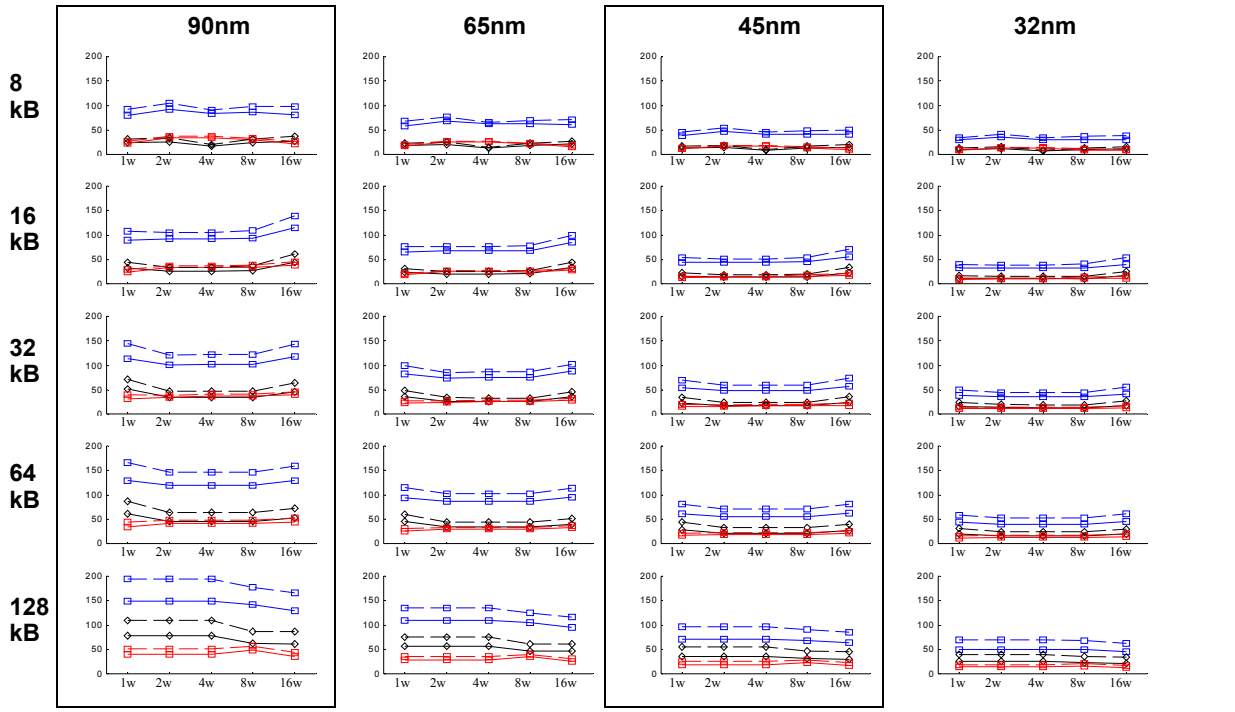


Figure 4-124: Delay components for pipeline stage pipe1B_data . This pipeline stage consists of the data predecoder driver and the wordline front-end. The solid-lines denote default myCACTI runs that enable multiple new features , while the dashed lines denote numbers that assumed CACTI/eCACTI-like operation

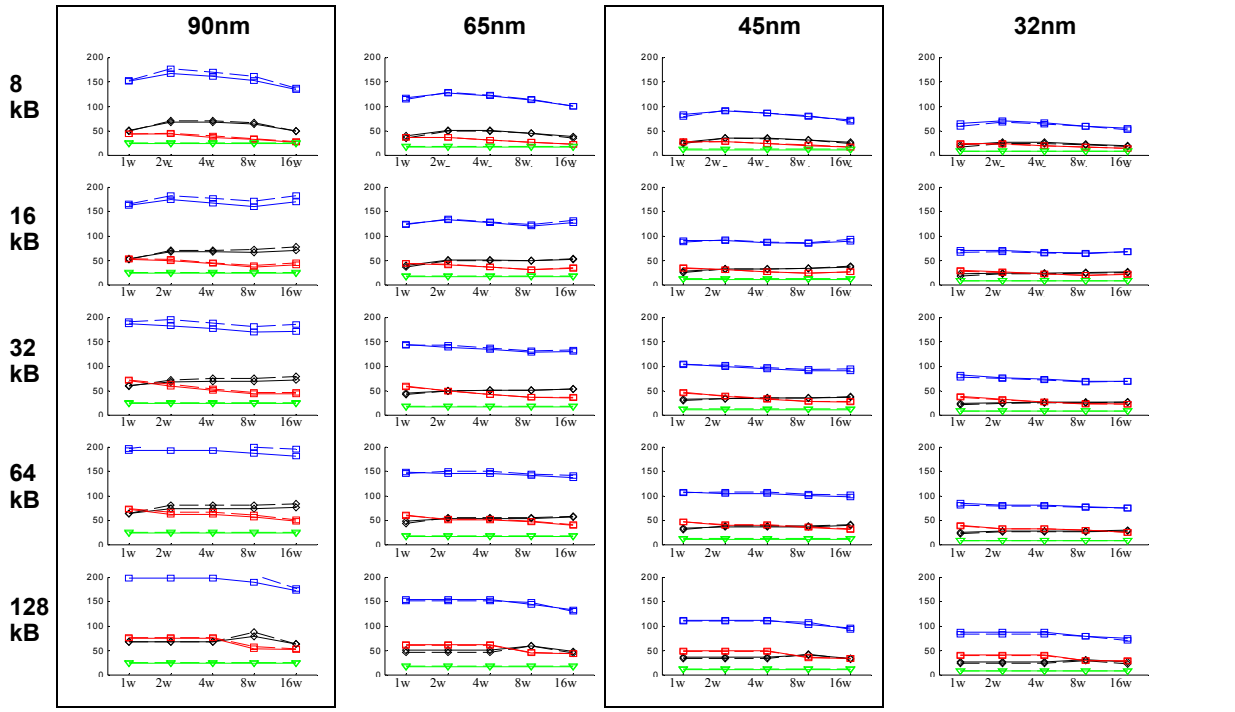


Figure 4-125: Delay components for pipeline stage pipe2A_data . This pipeline stage consists of the wordline drivers, the bitline and the sense amplifier. The solid-lines denote default myCACTI runs that enable multiple new features , while the dashed lines denote numbers that assumed CACTI/ eCACTI-like operation

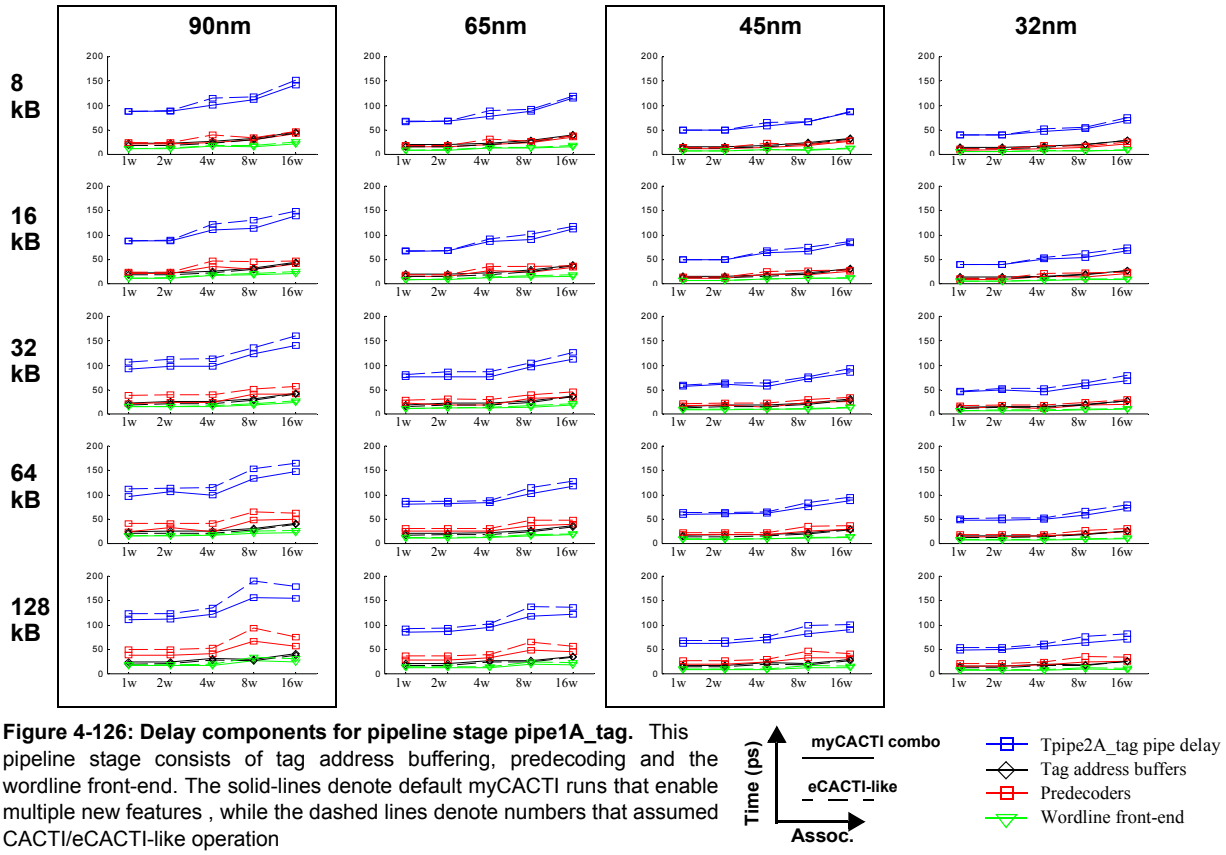


Figure 4-126: Delay components for pipeline stage pipe1A_tag. This pipeline stage consists of tag address buffering, predecoding and the wordline front-end. The solid-lines denote default myCACTI runs that enable multiple new features , while the dashed lines denote numbers that assumed CACTI/eCACTI-like operation

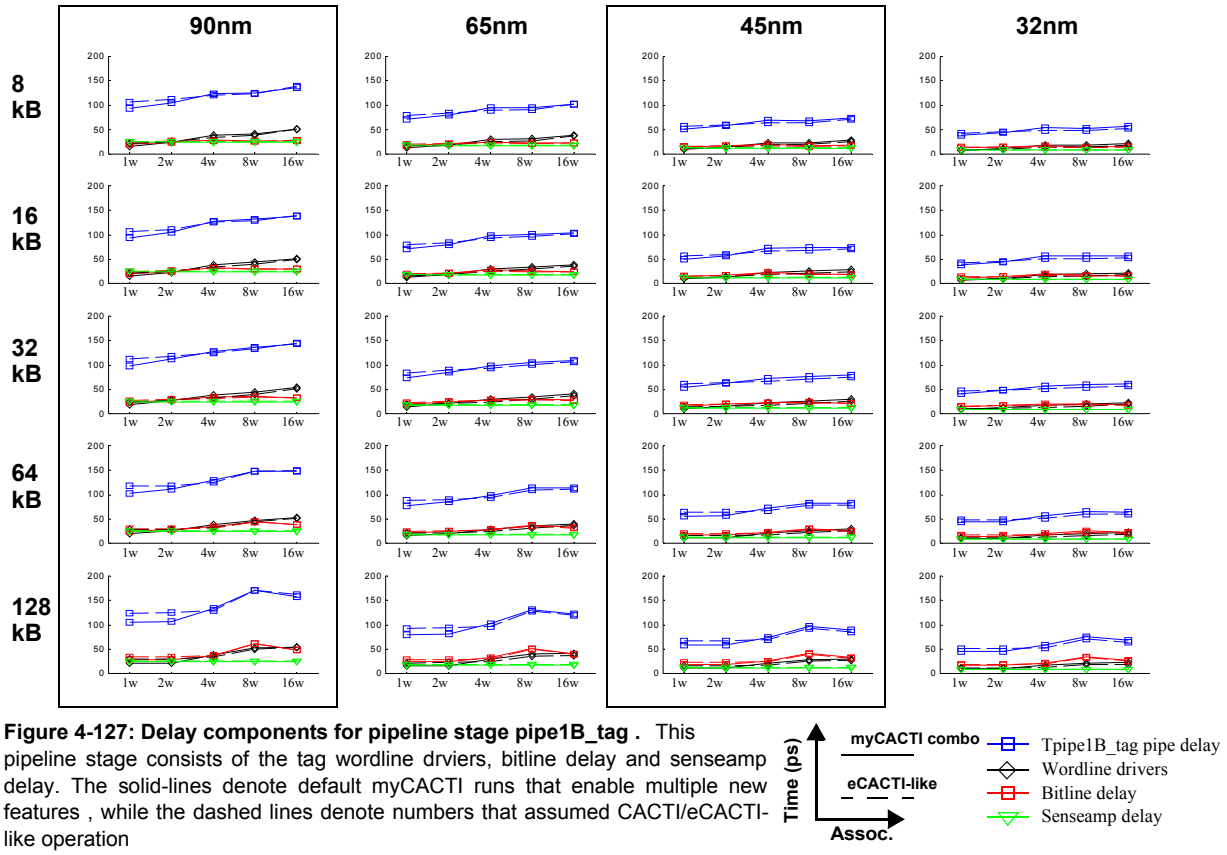


Figure 4-127: Delay components for pipeline stage pipe1B_tag . This pipeline stage consists of the tag wordline drviers, bitline delay and senseamp delay. The solid-lines denote default myCACTI runs that enable multiple new features , while the dashed lines denote numbers that assumed CACTI/eCACTI-like operation

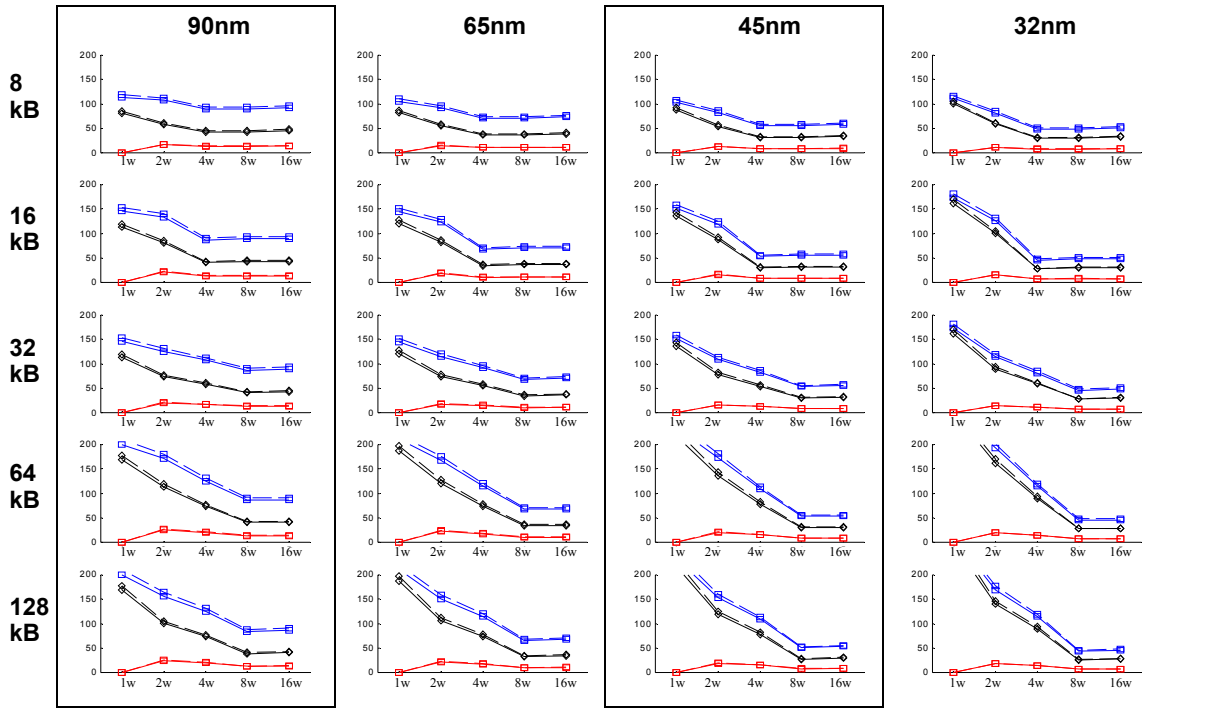


Figure 4-128: Delay components for pipeline stage pipe2A_tag . This pipeline stage consists of enabling the comparator and the actual comparator delay. The solid-lines denote default myCACTI runs that enable multiple new features , while the dashed lines denote numbers that assumed CACTI/eCACTI-like operation

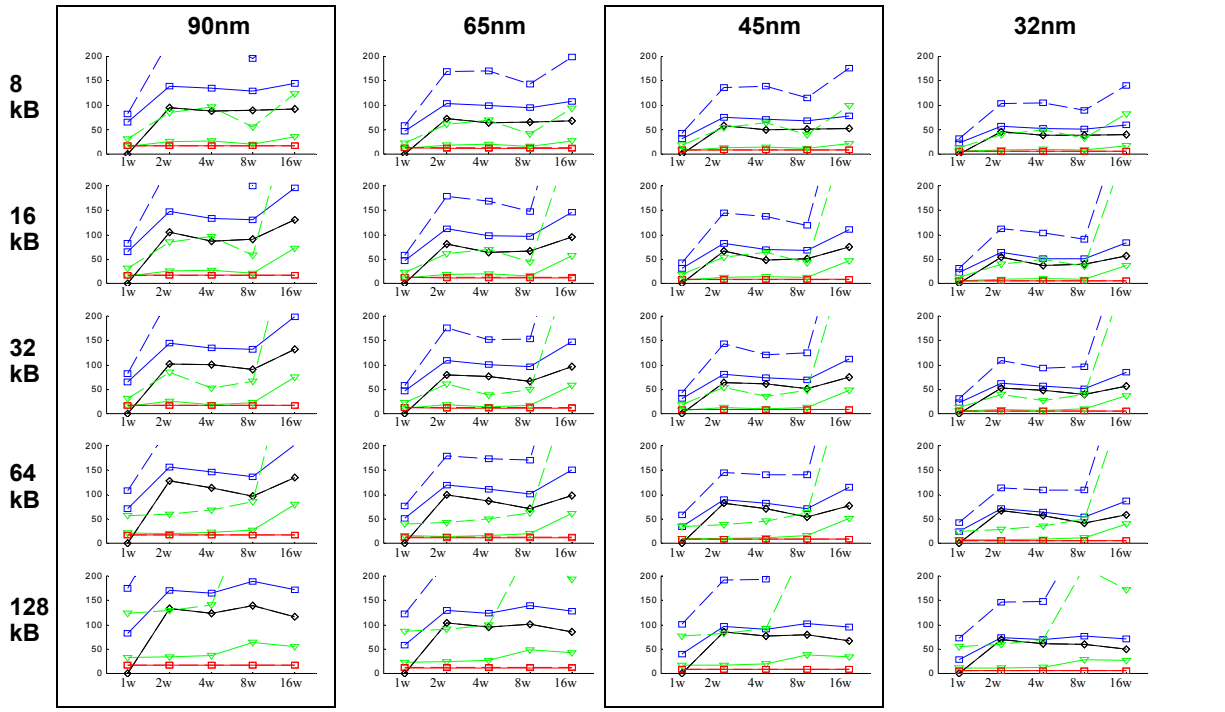


Figure 4-129: Delay components for pipeline stage pipe2B . This pipeline stage consists of driving the output buffer selects and the final data output drive to the cache periphery. The solid-lines denote default myCACTI runs that enable multiple new features , while the dashed lines denote numbers that assumed CACTI/eCACTI-like operation

Configurations that have the output stage as the critical path have typical access time degradations from 80ps to 200ps (with some extreme cases being degraded by more than 500ps), while other configurations have typical degradations of up to 50ps.

For the stage pipe1A_data, the effect on the address buffers and predecoder front-ends are opposite -- for the address buffers, CACTI/eCACTI-like operation underestimate the delay, while it overestimates the delay for the predecoder front-end. For the stage pipe1B_data, the predecoder driver and the wordline front-end are affected the same way -- CACTI/eCACTI-like operation overestimates both delay, although the predecoder driver is affected more than the wordline front-end since the predecoder driver may constitute more than one stage, while the wordline front-end is always one. For the stage pipe2A_data, the bitline delay is affected very significantly. For the tag part, stages pipe1A_tag and pipe2B_tag behave similarly to their data counterparts, while for stage pipe2A_tag, both the comparator enabled and the actual compare operation delay are overestimated by CACTI/eCACTI-like operation, with a total of 15ps difference. For this particular stage, only Ldrawn affects the operation, since the other four parameters (e.g. static/dynamic decoding, enabled/ disable via caps, enabled/disabled wordline resistance and multi/single layer metal) do not really factor in to this particular stage. Finally, for the data output stage pipe2B, as explained earlier, the data output driver stage is really the one that is most degraded. Within this stage, the mux driver and the final data output driver are

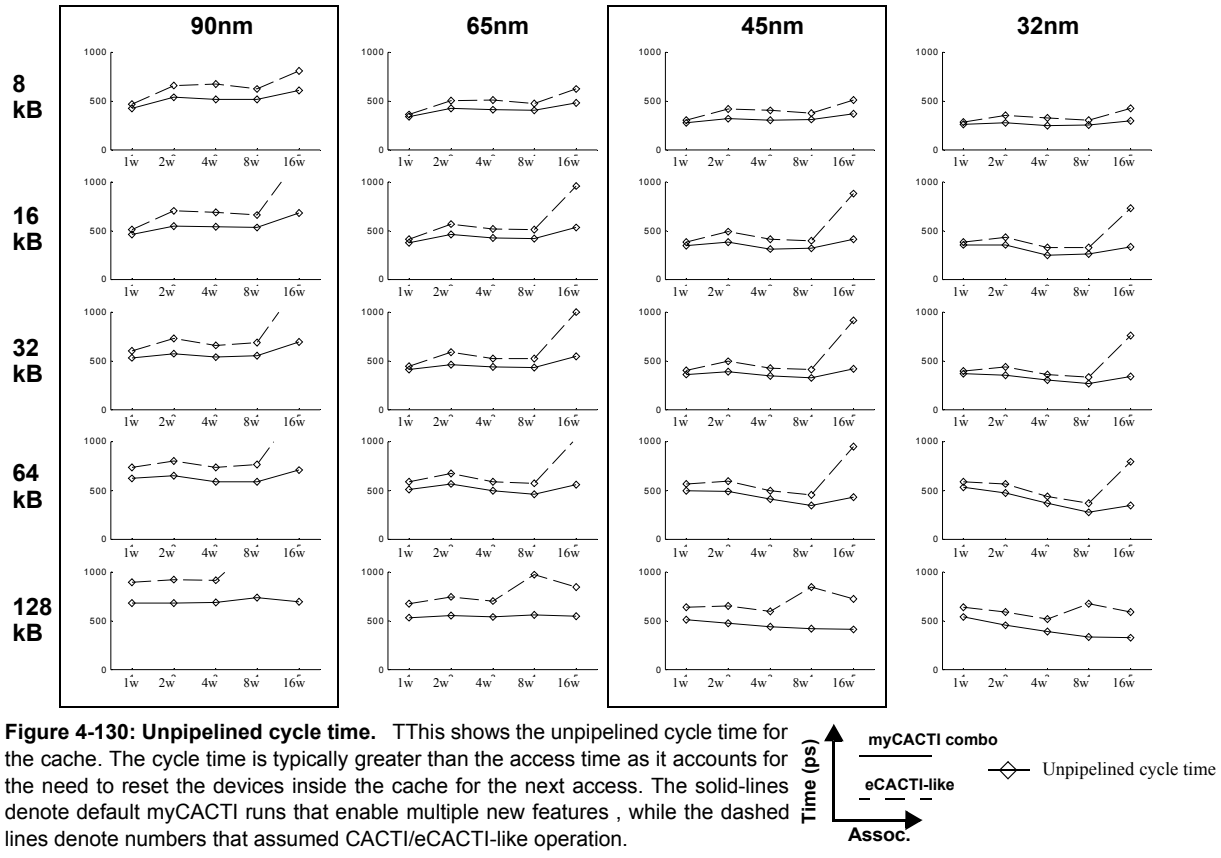


Figure 4-130: Unpipelined cycle time. TThis shows the unpipelined cycle time for the cache. The cycle time is typically greater than the access time as it accounts for the need to reset the devices inside the cache for the next access. The solid-lines denote default myCACTI runs that enable multiple new features , while the dashed lines denote numbers that assumed CACTI/eCACTI-like operation.

both significantly degraded, but since the final output drive typically drives a much longer wire, we observe that it is often degraded more.

Run power results

Figure 4-131 shows the total power and its breakdown into dynamic power (switching power) and static power (dissipated by subthreshold leakage and gate leakage currents) for every point in the design space that was considered (i.e. cache size, associativity and technology node). Each plot shows both the results from myCACTI runs that assume the default myCACTI features or reverts back to old CACTI/eCACTI-like operation. Figure 4-132 demonstrates the differences more explicitly by plotting the fractional difference between the two runs normalized to the total power of the default myCACTI result. Figure 4-133 shows detailed power breakdowns of some representative design points in order to explore in detail the causes of the differences. In addition, the results from the two runs are placed side by side for easy comparison. Lastly, Figure 4-134 identifies how much power is being dissipated by major cache blocks, and again shows the results from the two runs side by side for easy comparison.

From the figures, we can see that for the larger nanometer technology nodes (e.g. 90nm and 65nm), there are no significant power differences -- with the power difference peaking at around 4% of the total power. For the 45nm node, the difference rises to around 10%, and this difference tends to decrease with

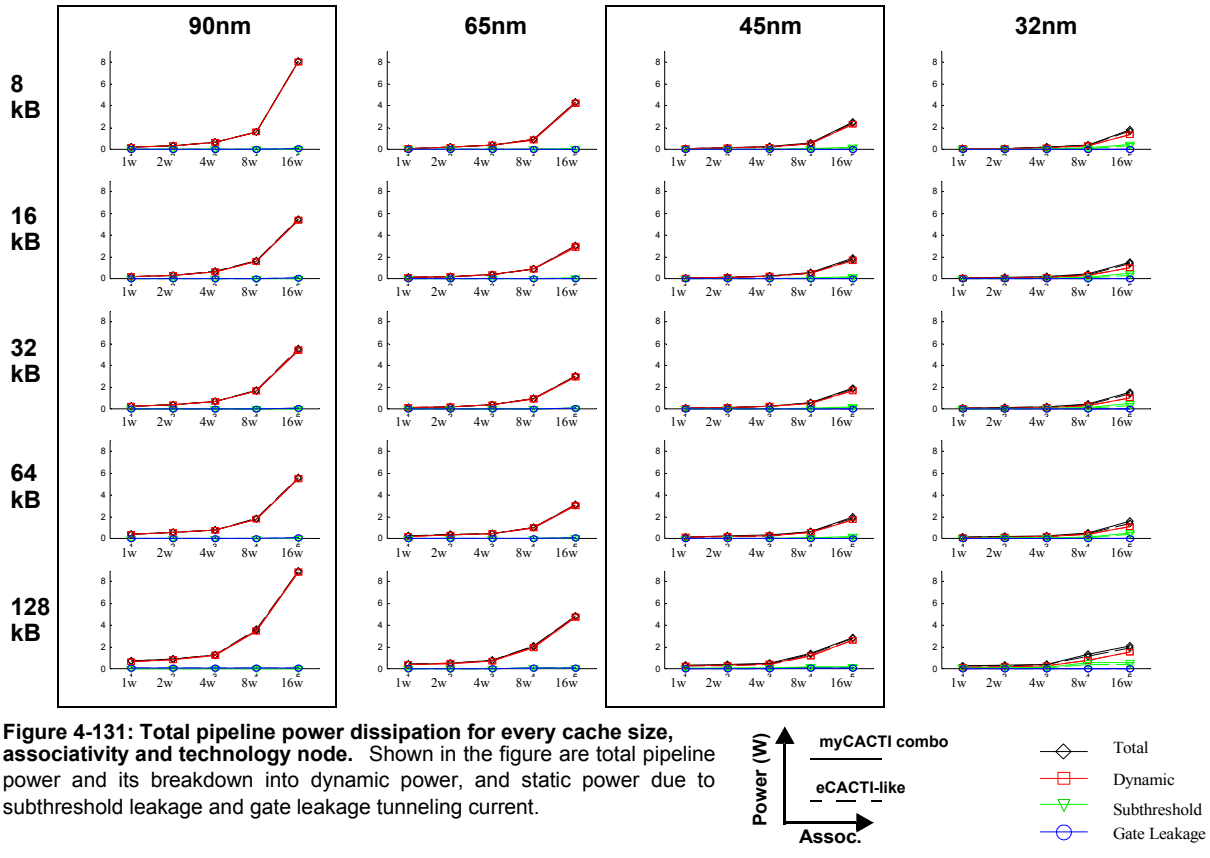


Figure 4-131: Total pipeline power dissipation for every cache size, associativity and technology node. Shown in the figure are total pipeline power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

increasing associativity, and it is made up primarily of dynamic power differences and secondarily by subthreshold leakage power differences. Finally, for the 32nm node, the power difference increases to about 20%, with subthreshold leakage now making up a majority of the difference, followed by dynamic power.

Even though the power differences are small in many of the cases, this does not necessarily mean that the two types of modeling give similar results. Results of inidividual components may vary greatly, but may end up mostly cancelling the gain or loss of some other component, ending up in a total value that may be largely unchanged (at least in the case of the 90nm and 65nm nodes), but hiding the internal component differences such that conclusions regarding power breakdowns using the CACTI/eCACTI-like operation would still be largely inacccurate.

We can demonstrate this further by looking at the power breakdowns in more detail. For 90nm, the data-out component shows a significant difference, but the difference in other components (e.g. data wordline, data bitline and the data RW control) have the opposite polarity, such that the difference when looking solely at the total power is reduced. This plays a major factor when optimizing individual parts of the cache -- even though the total power numbers may be close (at least for the 90nm and 65nm nodes), the potential difference when a specific cache component is being considerred may be much greater. Even at the deep nanometer technology nodes where the polarities of the differences may be the same, using total power as a guide is

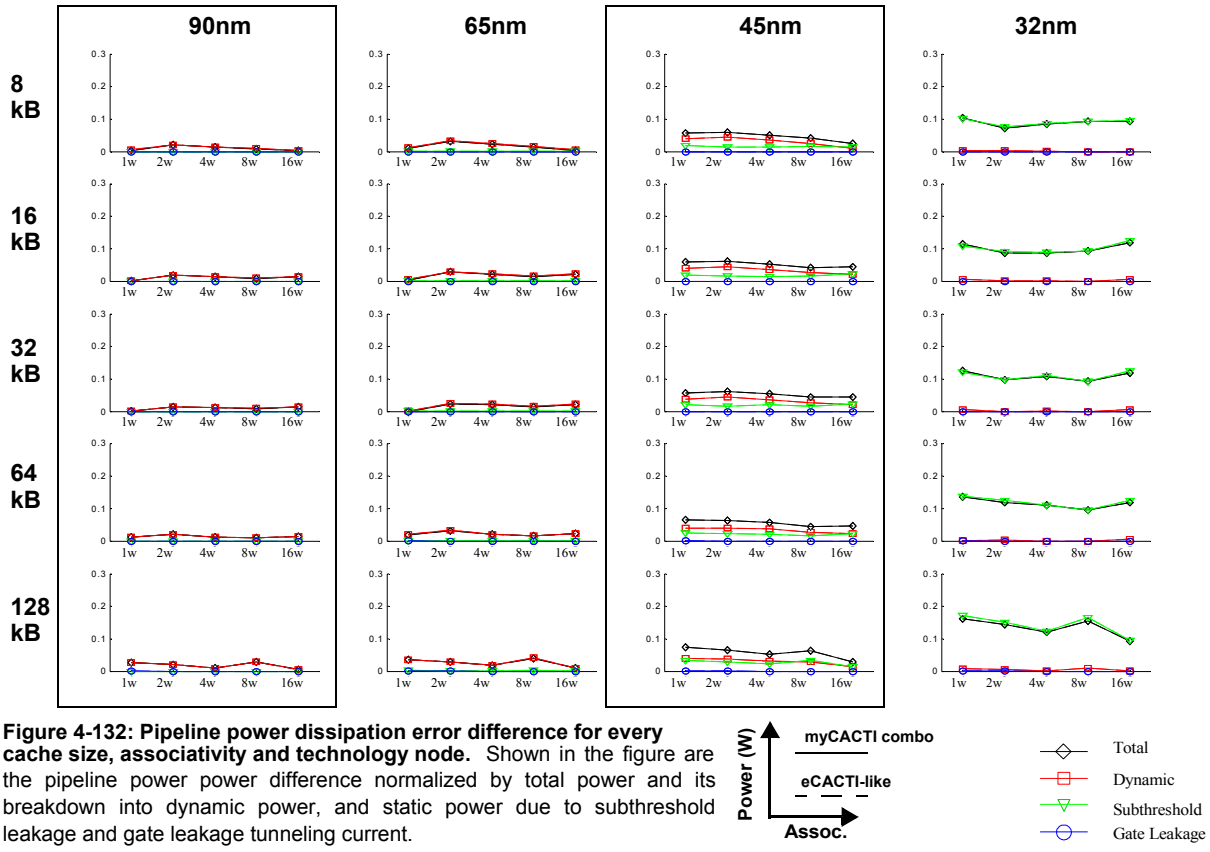


Figure 4-132: Pipeline power dissipation error difference for every cache size, associativity and technology node. Shown in the figure are the pipeline power power difference normalized by total power and its breakdown into dynamic power, and static power due to subthreshold leakage and gate leakage tunneling current.

misleading, as other components may actually be unchanged by the paramaters such that the power differences are only a small subset of the cache power.

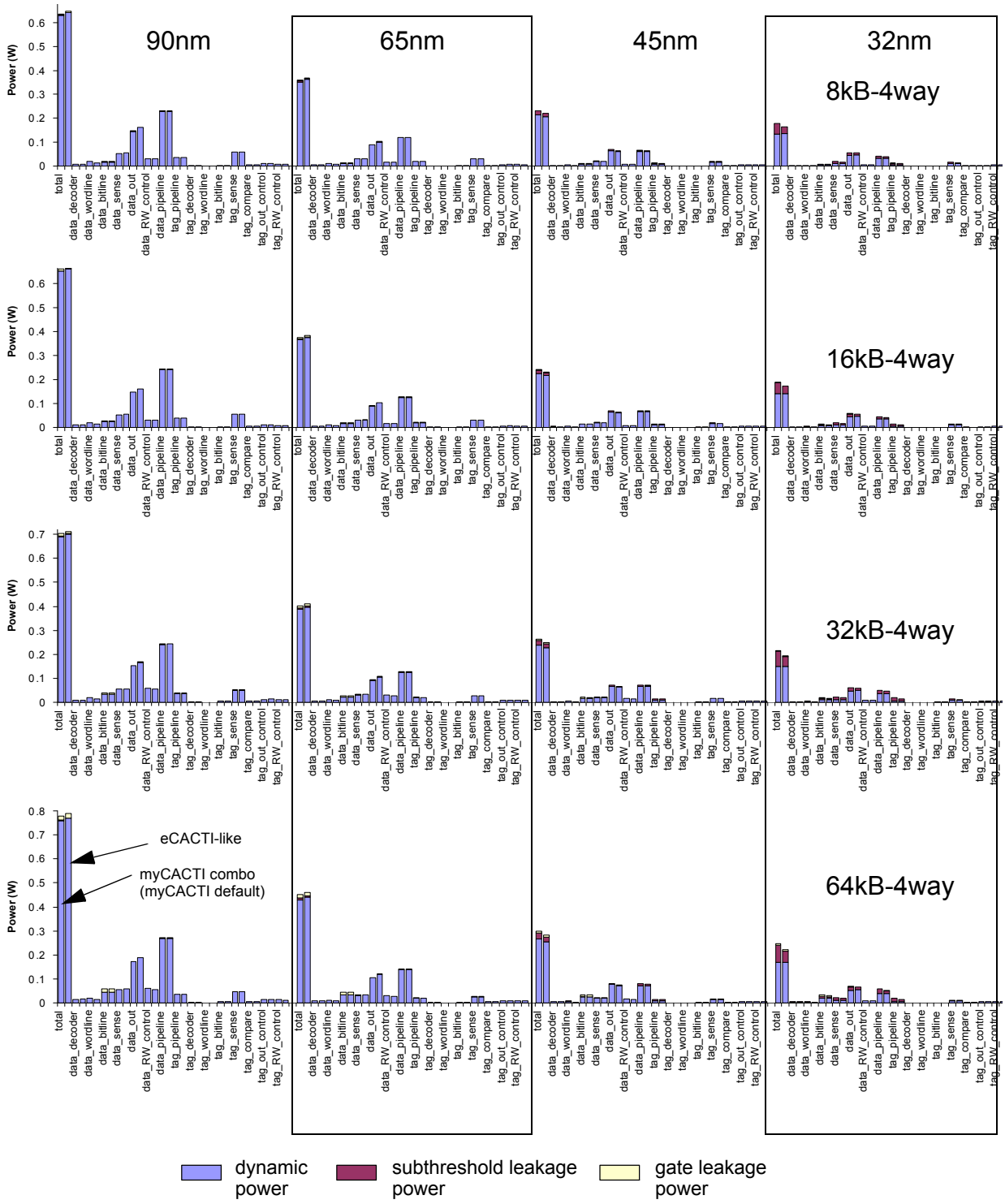


Figure 4-133: Detailed cache power breakdown comparison. Detailed power breakdown for four different cache configurations showing the amount of dynamic, subthreshold leakage and gate leakage power being consumed by different parts of the cache. Each plot shows four sets of values corresponding to different technology nodes. (Note that all three plots use the same scale). The two results are also placed side by side for direct comparison.

Conclusion

Comparing CACTI/eCACTI-like operation vs. myCACTI shows drastic differences in delay and significant differences in power at the deep nanometer nodes. In addition, even though some of the configurations may show only minor "total" differences, these do not mean that the internals are also unchanged. On the contrary, we have shown that some of the internal changes may have opposing polarity such that a significant difference in one block may be partly cancelled by another block. It is therefore patently wrong to conclude that the internals are unaffected. Moreover, both the delay and power differences are not "offset" errors and as such, it is not possible to extrapolate data from a CACTI/eCACTI run and convert it into

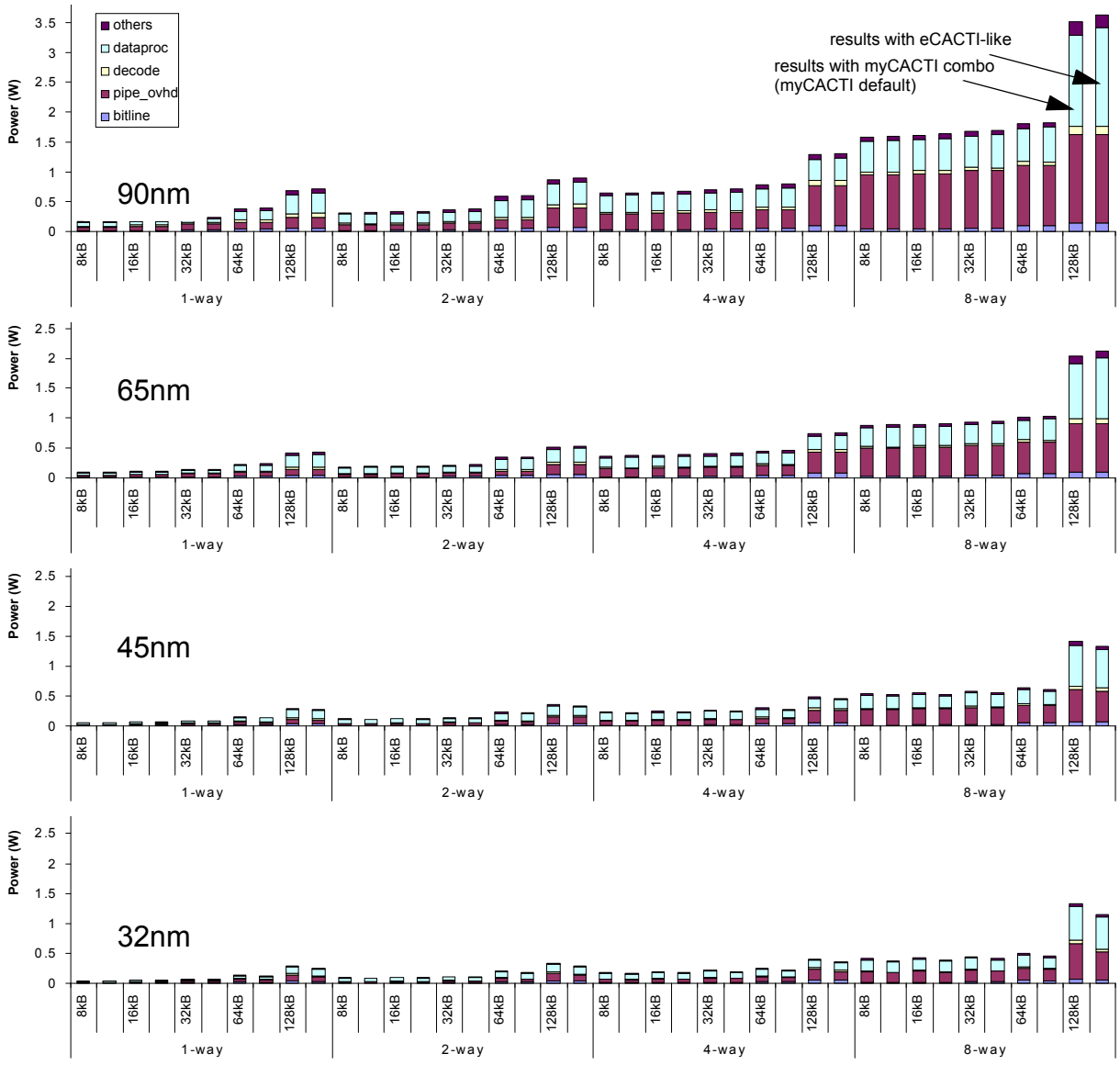


Figure 4-134: Power contributions of different cache parts. Power breakdown showing major cache power contributors for the 65nm and 32nm technology nodes for different cache sizes and associativities. (Note that the y-axis for both plots use the same scale). The two results are also placed side by side for direct comparison.

a more accurate version, since the differences rely greatly on the configuration being used (e.g. cache size, associativity and technology node) and the actual cache implementation.

4.5.12 Direct Output Comparison with CACTI and eCACTI

Run details

In this section, we provide detailed output comparisons between results generated by the three tools. To generate these numbers, CACTI was made to run first to determine the optimal implementations for the particular configurations being studied. eCACTI and myCACTI are then subsequently run design space evaluations (and not explorations), using the optimal implementation generated by CACTI. Although this means that the numbers produced by eCACTI and myCACTI are not the optimal values for the particular cache configuration (since they were not allowed to do an optimizing search), the goal of comparing numbers to like numbers is accomplished, since all three caches will have the exact same implementation. These implementations are shown in Figure 4-135.

Unpipelined access time

The unpipelined access times generated by the three tools for the various caches studied are shown in Figure 4-136 for the 90nm and 65nm node[1]. It is fair to compare these numbers directly, although it must be stated that the myCACTI numbers are artificially generated by computing for the aggregate delay of the critical path through the whole cache (as opposed to just the delay through one stage, which is the default in myCACTI because of its inherent pipeline operation). It can be seen that myCACTI produces caches with

		90nm	65nm
16kB	1-way	2:2:1:1:2:2	2:2:1:1:2:2
	2-way	4:1:1:1:2:1	4:1:1:1:2:1
	4-way	8:1:1:1:2:1	8:1:1:1:2:1
	8-way	16:1:1:1:4:1	16:1:1:1:4:1
32kB	1-way	2:2:1:1:2:4	2:2:1:1:2:4
	2-way	4:2:1:1:2:2	4:2:1:1:2:2
	4-way	8:1:1:1:2:1	8:1:1:1:2:1
	8-way	8:1:1:1:2:1	8:1:1:1:2:1
64kB	1-way	2:2:1:1:4:4	2:2:1:1:4:4
	2-way	4:2:1:1:4:2	4:2:1:1:4:2
	4-way	8:1:1:1:2:1	8:1:1:1:2:1
	8-way	8:1:1:1:2:1	8:1:1:1:2:1
128kB	1-way	2:4:1:1:4:8	2:2:1:1:4:16
	2-way	4:2:1:1:8:4	4:2:1:1:8:4
	4-way	4:2:1:1:4:1	4:2:1:1:2:1
	8-way	8:1:1:1:4:1	8:1:1:1:4:1

Figure 4-135: CACTI-generated optimal implementations.

1. eCACTI only supports 100nm and 70nm, so these values were used in place of 90nm and 65nm.

90nm		CACTI	eCACTI	myCACTI
16kB	1-way	0.5621	0.7837	0.3644
	2-way	0.6315	0.9081	0.3257
	4-way	0.6565	0.8648	0.333
	8-way	0.7438	14.43	0.478
32kB	1-way	0.6514	0.8021	0.4136
	2-way	0.6889	1.067	0.3614
	4-way	0.6852	0.9156	0.3507
	8-way	0.7654	1.01	0.383
64kB	1-way	0.8281	0.8479	0.4958
	2-way	0.765	1.05	0.4148
	4-way	0.7931	1.03	0.3916
	8-way	0.7987	1.08	0.3993
128kB	1-way	1.123	1.09	XXX
	2-way	1.066	1.1	0.5414
	4-way	0.8994	1.549	XXX
	8-way	0.8754	1.198	0.4822

65nm		CACTI	eCACTI	myCACTI
16kB	1-way	0.406	0.5486	0.2888
	2-way	0.456	0.6356	0.2596
	4-way	0.4741	0.605	0.2668
	8-way	0.5372	10.1	0.3897
32kB	1-way	0.4704	0.5615	0.3228
	2-way	0.4975	0.7475	0.2843
	4-way	0.4948	0.6409	0.2799
	8-way	0.5527	0.7076	0.3073
64kB	1-way	0.5981	0.5935	0.3841
	2-way	0.5525	0.7371	0.3314
	4-way	0.5728	0.7245	0.3081
	8-way	0.5768	0.7575	0.3192
128kB	1-way	0.9835	0.7522	0.5181
	2-way	0.7699	0.77	0.4893
	4-way	0.6802	1.14	XXX
	8-way	0.6322	0.839	0.3901

Figure 4-136: Unpipelined access time (ns).

significantly faster access times. In addition, myCACTI labels some values with "XXX", signifying that these configurations as determined by CACTI should actually be invalid for various reasons (discussed earlier, like the impossibility of meeting the wordline transistion time requirement, something that CACTI and eCACTI inaccurately handle)

Pipelined cache clock period

The pipelined cache clock periods generated by the three tools for the various caches studied are shown in Figure 4-137for the 90nm and 65nm node. In this case, the values produced by CACTI and eCACTI are the results of wavepipelining, while the values from myCACTI are from explicit pipelining.

It must be emphasized that it is not possible to do a direct comparison of these numbers. The most important reason for this is that the CACTI and eCACTI implementation of wave-pipelining is unrealistic, and does not account for additional overheads in the cache like bitline precharging. CACTI and eCACTI wave-pipelining assume logic-like behavior where input signals can change beat per beat without any maintenance

90nm		CACTI	eCACTI	myCACTI
16kB	1-way	0.226	0.4555	0.372
	2-way	0.2104	0.3769	0.316
	4-way	0.2188	0.3157	0.296
	8-way	0.3291	13.96	0.389
32kB	1-way	0.2831	0.458	0.42
	2-way	0.2681	0.4007	0.367
	4-way	0.2677	0.322	0.326
	8-way	0.2551	0.3369	0.34
64kB	1-way	0.4108	0.4118	0.503
	2-way	0.3623	0.369	0.42
	4-way	0.361781	0.347	0.376
	8-way	0.297	0.3607	0.373
128kB	1-way	0.634	0.3634	XXX
	2-way	0.577	0.366	0.503
	4-way	0.3943	0.7	XXX
	8-way	0.3949	0.4454	0.427

65nm		CACTI	eCACTI	myCACTI
16kB	1-way	0.1632	0.3188	0.2859
	2-way	0.152	0.2638	0.242
	4-way	0.158	0.221	0.2251
	8-way	0.2377	9.775	0.3154
32kB	1-way	0.2045	0.3206	0.3215
	2-way	0.1936	0.2805	0.2794
	4-way	0.1934	0.2254	0.2494
	8-way	0.1842	0.2358	0.2568
64kB	1-way	0.2967	0.2883	0.3892
	2-way	0.2616	0.2584	0.3215
	4-way	0.2612	0.243	0.2904
	8-way	0.2145	0.2525	0.2831
128kB	1-way	0.6224	0.2636	0.745
	2-way	0.4167	0.2566	0.48
	4-way	0.2845	0.49	XXX
	8-way	0.2852	0.3118	0.3273

Figure 4-137: Pipelined cache clock period (ns).

being done in the circuit. This is not true for the bitlines, which have to be precharged for each instance of an input signal. Hence, the numbers provided by CACTI and eCACTI are vastly optimistic. A second reason is that even disregarding the optimistic numbers, the number of pipeline stages that CACTI and eCACTI vary. Some implementations have two stages, while some have three. This variation is very undesirable during design, as the cache has to interact with the processor at different times in the cycle and not just at the beginning or the end of the access (for example, for retrieving addresses mid-access from the TLB). In any realistic design, this requirement is fixed and made static at some point, and should not depend on the characteristics of the particular cache being studied

Read hit power

Finally, the read-hit power generated by the three tools for the various caches studied are shown in Figure 4-138for the 90nm and 65nm node[1]. Again, it should be emphasized that it is not valid to do a direct comparison between the different numbers. One of the main reasons is the different frequency and supply voltages used by the three tools, as shown in Figure 4-139, which shows that both CACTI and eCACTI assume a fixed frequency of 500MHz regardless of technology node, while myCACTI uses a more realistic, higher frequency that is also dependent on the technology node. Along with the higher voltages used in myCACTI, this serves to significantly increase the power dissipation of myCACTI caches. In addition, as already seen in the data from the previous sections, the pipeline overhead power dissipation is a very significant fraction of total power dissipation, which increases the myCACTI cache power draw even more.

		CACTI	eCACTI	myCACTI
Frequency	90nm	500MHz	500MHz	2.4GHz
	65nm	500MHz	500MHz	2.6GHz
VDD	90nm	1.0V	1.2V	1.4V
	65nm	0.8V	1.0V	1.2V

Figure 4-139: Supply voltage and frequencies used by the three tools.

1. eCACTI and myCACTI produce power values, while CACTI produces energy values. The above numbers attributed to CACTI are generated by converting energy into power using CACTI's own frequency assumption.

90nm		CACTI	eCACTI	myCACTI
16kB	1-way	78.85	57.58	279.6
	2-way	117.1	83.87	502
	4-way	188.95	110.4	1171
	8-way	342.8	476.4	3685
32kB	1-way	92.5	93.87	329.3
	2-way	128.6	107.2	495.3
	4-way	202.45	182.4	1331
	8-way	348.3	225.5	2392
64kB	1-way	119.95	166.6	434.9
	2-way	148.25	181.7	587.2
	4-way	223.85	314.7	1651
	8-way	368.25	365.47	2602
128kB	1-way	184.85	331.1	XXX
	2-way	188.65	328.6	781.2
	4-way	253.2	372.4	XXX
	8-way	397.15	625.2	302

65nm		CACTI	eCACTI	
16kB	1-way	50.8	96.76	158
	2-way	78.1	113.35	265.3
	4-way	130.3	161	596.2
	8-way	239.15	1350	183.7
32kB	1-way	56.95	170.9	188.3
	2-way	83.3	183.8	274.3
	4-way	136.4	272	686.2
	8-way	241.5	318.6	1229
64kB	1-way	69.6	319.9	251.5
	2-way	92.3	335.7	330.8
	4-way	146.2	458	865.7
	8-way	250.65	535	1353
128kB	1-way	102.35	1028	368.4
	2-way	111.05	638	448.5
	4-way	160.6	665.5	XXX
	8-way	263.95	905	1601

Figure 4-138: Read-hit power dissipation (mW).

4.6 Comparative study conclusions

In this subsection, we summarize the results of the the different comparative studies that were performed in this chapter. The different summaries from each subsection are first repeated, then integrated into a single summary.

4.6.1 Validity of CACTI/eCACTI scaling

Our results have shown that the simple linear scaling of results from an older reference technology node (in our case, 0.13um), in order to produce results for smaller tech nodes produces significant differences compared to explicitly modeling the target tech node.

We see that both cache delay and power have significant differences. Making it worse is that scaling may overestimate results for one configuration, while underestimating results for another. This makes the variability of the result even bigger and makes the results provided by linear scaling even more unreliable. Since explicitly modeling the target process does not provide a burden to the user (it only makes it more difficult to write the program in order to properly support multiple process technologies instead of just a single one), it is imperative that cache design tools always explicitly simulate the target process instead of doing a linear scaling of the results. This is even more important for deep nanometer technologies, as current trends and design practices that prevail for these technologies may not necessarily be captured by models of older process technologies. An obvious example is the big discrepancy in subthreshold leakage values, as linear scaling doesn't capture the exponential increase of subthreshold leakage with each new generation. A more subtle example is the possible non-scaling of device speeds as process technologies may have implemented techniques that sacrifice speed of operation in exchange for better reliability or lower power.
Finally, it must be emphasized that both CACTI and eCACTI[1] perform linear scaling of results that are based on a 0.80um technology instead of the 0.13um technology that we have used in the comparison. This means that the parameters in this simulation that assumes long-channel transistors will be much more different compared to nanometer transistor behavior as opposed to the submicron technology (0.13um) that we used as the scaling reference in this subsection.

4.6.2 Transistor effective length

We have shown that modeling the presence of the drain/source overlap region when computing for the transistor's effective length generally results in faster delay with a corresponding increase in power. The delay improvement can be observed by all pipeline stages since all transistors within the cache are affected.

1. An exception exists for eCACTI for subthreshold leakage computation, which it does on a per-process basis instead of using linear scaling.

Additionally, the power increase is almost all due to an increase in subthreshold leak power because of the (often) shorter transistor channel, with dynamic and gate leakage mostly unaffected. This power difference is observed to be mostly significant for the deep nanometer nodes (e.g. 45nm and 32nm), but the delay difference is observed at every point in the design space (i.e. every cache size, associativity and technology node).

4.6.3 Via parasitic capacitance

We have shown that modeling the additional capacitance presented by interconnect vias result in minimal differences in terms of cache power, but significant differences in terms of cache delay. In addition, because of the pipelined nature of the cache that we model, pipeline stages that have their delays degraded by accounting for the additional via capacitances do not necesarily constitute the critical path. Consequently, some configurations, even if some of their pipe stages become stretched out due to the additional delays caused by the via caps, may actually have their effective clock cycle unchanged if the critical path resides in stages that are not largely affected by the via caps (e.g. the tag compare stage). This results in the comparison error not being of the "offset" type and hence, it is difficult to extrapolate the final result given a simulation that does not account for the via capacitance.

We conclude that it is important to include modeling of via capacitances during the simulation, as it potentially has a significant effect on the cache delays.

4.6.4 Pipelining comparison

We have shown that cache designers cannot expect to trivially add pipelining to an existing cache design without expecting to drastically alter the power envelope of the circuit, as the overhead required for pipeline elements constitutes a significant fraction of total cache power. Also, we expect the delay behavior of the cache to change because this is inherent to pipelining and is one of the reasons we perform pipelining in the first place, but for this section, we only emphasize the importance of the need for accounting for pipeline power, and pipeline delays are discussed in more detail in a later section.

4.6.5 Number of interconnect layers

We have shown that modeling interconnect as either single or multi-layer results in negligible differences in power, but very significant differences in delay. For delays, the value can either be overestimated or underestimated by single-layer metal modeling depending on the particular case. Given that resistance tends to decrease while capacitance tends to increase (at a slower rate) as we go higher in the metal stack, the delay change to a particular cache block will depend on which of the resistance or capacitance is dominant. For example, in the bitlines where the small widths of the access and driver transistors result in a large effective driving resistance, using a highly resistive local interconnect layer for the bitlines might be acceptable,

especially since this also gives you lower capacitive loading. For strong drivers with low effective resistances, though, increasing the wire resistance might cause more delay degradation than increasing the capacitive loading.

In any case, modeling an unrealistic BEOL stack has been shown to give very pessimistic access time numbers, and although additional pessimism does assure that the real produce has better behavior, it may also result in too much overdesign resulting in an inefficient cache. We conclude that it is important to model realistic BEOL stacks by modeling multi-layer interconnects properly.

4.6.6 Gate leakage

We observe that accounting for gate leakage results in a marginal power difference, and this difference is reduced even more with increasing associativity.

This is a largely unexpected result, as gate leakage is widely expected to contribute more and more power with each technology generation. We discuss this interesting finding in a more detailed section later that explains why gate leakage is not as big of a problem as it was being predicted to be. In essence, the process technology designers have responded to the perceived gate leakage problem and have made various changes to the semiconductor roadmaps such that although gate leakage is still increasing on a per unit area basis, other effects such as decreasing supply voltage and device sizes contribute to actually decreasing the absolute value of gate leakage.

4.6.7 Static/dynamic decoding

We have shown that implementing a decoder using dynamic circuitry results in smaller delays in the decoder, with minimal effect in power. Of course, the additional complexity (in terms of both design, verification, and reliability) of dynamic circuits must be accounted for by the designer, as static circuitry has the advantage in terms of these criteria.

In addition, even though the all of the subparts of a decoder can be sped up by implementing everything using dynamic logic, the speed up in some of the parts of the decoder is typically nullified since these reside in stages that have typically large slack such that any speed up in delay does not affect the critical path at all. This builds the case for only doing a decoder partially in dynamic logic. If the additional speed due to dynamic circuits is warranted, it should only be selectively applied to parts of the circuit that we are relatively sure will be in the critical path, while the rest should be implemented in static logic to minimize design complexity. This way, every part where the designer is paying for the complexity of using dynamic circuits should actually yield some form of delay improvement. In cases where the critical path is not the decoder, we would prefer to have an all-static decoder to simplify the circuit implementation, without really having any negative repercussions -- a win-win situation.

4.6.8 Dual-Vt process technologies

We have shown that implementing a cache in dual-Vt as opposed to single-Vt has results only in a slight degradation of the total clock period of a pipelined cache, as the only critical path that is affected (with proper Vt placement) is the access and driver transistors in the memory cell. In addition, the power savings when going from a single-Vt implementation to a dual-Vt is significant, especiallly at the 45nm and 32nm node.

We therefore conclude that unless the slight speed degradation is really detrimental, the power savings realized in implementing a cache as dual-Vt more than outweighs the slight degradation in the bitline delay. In implementations where the bitline delay does not reside in the critical path, we can essentially get this power savings for free, as all other critical paths are implemented in low-Vt transistors such that they don't incur any delay, while the whole circuit still enjoys the power savings accrued with using high-Vt transistors in devices outside of the critical path.

4.6.9 Single-ended sensing

At first glance, shifting from low-swing differential to full-swing single-ended sensing seems to be unattractive in that it results in a significant increase in power dissipation (for all configurations), and an even larger increase in access time (for the typical configuration). It would be easy to dismiss the full-swing single-ended sensing technique based on these two criteria.

Other concerns exist, though, that have to be considered before concluding whether this could be a passable technique. With single-ended sensing, only a single-ended bitline signal needs to be routed instead of a differential bitline signal. With memory arrays often being metal-limited, reducing the track requirements by a single metal route may result in significant area savings. In addition, a single-ended signal has the added benefit of reduced design complexity and verification. Although differential bitline signals have the advantage of typically being faster by virtue of being low-swing and having to discharge a smaller voltage, it comes with the penalty of the need to verify the differential property of these two signals. Instead of a single noise analysis performed on a single-ended bitline, common-mode noise analysis and differential-mode analysis have to be performed on the two-bitlines, making the verification of the design harder and more complicated. In addition, as mentioned earlier, the bitlines are actually margined to provide more differential, such that typical differential sensing numbers that we give are optimistic. In addition, differential noise analysis becomes harder to pass as supply voltage becomes smaller, since increasingly smaller noise values may already serve to cause upsets.

In summary, as the supply voltage gets smaller and smaller, the advantage of small-swing differential bitlines gives us diminishing returns. Coupled with the observation that the delay and power penalties of single-ended sensing gets smaller and smaller for deep nanometer technolgoy nodes, and that the difficulty in designing small-swing differential bitlines becomes much harder, we can conclude that single-ended sensing

might not be practical for 90nm and 65nm because of the mentioned degradation in delay and power, but are starting to become more and more attractive with each generational upgrade.

4.6.10 BEOL low-k study

We have demonstrated that the power advantages when going from a base ILD K of about 3.0 to a low-K ILD (of about 1.0) results in very significant power savings (roughly around 20%), and delay improvements. We demonstrate that the delay improvements are very significant for configurations that have their critical path going through stages that are very dependent on interconnect performance (like the data wordline-bitline-sense stages and the data out driver stage) and will typically provide around 20ps-60ps delay improvement.

Circuit-wise, there are no disadvantages involved in using low-K dielectrics for the interconnects, and the main problems that prevent low-K from being more prevalent in contemporary circuits are all process-based and are concerned with how easily and reliably they are integrated into an existing process.

4.6.11 Combination of multiple parameters

Comparing CACTI/eCACTI-like operation vs. myCACTI shows drastic differences in delay and significant differences in power at the deep nanometer nodes. In addition, even though some of the configurations may show only minor "total" differences, these do not mean that the internals are also unchanged. On the contrary, we have shown that some of the internal changes may have opposing polarity such that a significant difference in one block may be partly cancelled by another block. It is therefore patently wrong to conclude that the internals are unaffected. Moreover, both the delay and power differences are not "offset" errors and as such, it is not possible to extrapolate data from a CACTI/eCACTI run and convert it into a more accurate version, since the differences rely greatly on the configuration being used (e.g. cache size, associativity and technology node) and the actual cache implementation.

4.6.12 Final summary

We have demonstrated through these comparative studies, that both CACTI and eCACTI use assumptions that are impractical, especially for pipelined, nanometer caches. Without correction of these assumptions, CACTI and eCACTI results will be inaccurate with respect to their cache power dissipation and delay data.

CHAPTER 5 myCACTI Detailed Studies

5.1 Introduction

This section aims to show different applications of myCACTI as a tool for cache design. The first section provides a high-level view of the possibilities in cache design by doing extensive design space exploration to determine the optimal implementations of caches of different sizes, associativities and technology nodes. The second section then delves deeper by focusing on a particular cache size and associativity (64kB 4-way) and then providing very detailed data, breakdown and analyses of the cache's read-hit power and cache clock period. These first two sections use myCACTI default settings[1] to provide insights of caches that use a given set of design rules and circuit techniques. The third section provides another very detailed design space exploration, but this time also varies the different design rules and circuit techniques used by the cache.

5.2 General Design Space Exploration

This section aims to demonstrate some of the specific applications of myCACTI by doing design space explorations for a slew of cache configurations (cache size from 8kB to 128kB, associativity from 1-way to 16-way, and technology nodes of 130nm, 90nm, 65nm, 45nm and 32nm) and then providing complete pareto optimal curves for each particular cache configuration (i.e. cache size, associativity and technology node). For every cache configuration, myCACTI performs a complete design space exploration and produces a list of valid implementations and their detailed information. We post-process this information by sorting them systematically to come up with 2-D pareto optimal curves (using cache read-hit power and cache clock period as the criteria) for every cache configuration that we study.

This way, instead of being forced to accept a single implementation based on a fixed optimization algorithm (as was done by CACTI and eCACTI), the designer now has the option to choose an implementation based on the particular priorities of the design in mind. In addition, we show in later subsections that the pareto optimal curves and detailed delay and power breakdowns can be used to systematically choose designs that can be later optimized through focused custom design in order to yield an implementation that is more optimal than any point in the existing pareto optimal curves.

1. The default settings used by myCACTI are set to values that would be typical for use in early nanometer technology nodes like 90nm and 65nm

5.2.1 130nm process technology

The pareto plots for every cache size and associativity for the 130nm process technology node is shown in Figure 5-1.

Firstly, for the 130nm node, 16-way associative configurations have very large read-hit power such that, for clarity, they are not included here in order to show the curves for the lower associativities more clearly.

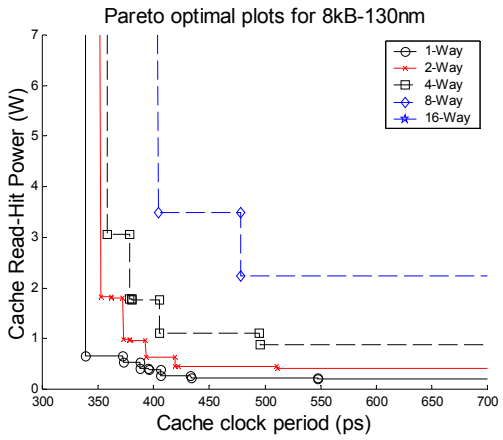
At 8kB, the pareto curves of the different associativities do not overlap. This can be interpreted as one associativity is always more optimal than another and that, in this case, the direct-mapped cache is the most optimal configuration, followed by 2-way then 4-way and so forth. Even though the clock period or the read-hit power of a particular point in a pareto curve may be worse in terms of a single criterion compared to another point in a less optimal curve, the first point will always be taken as more optimal when both criterion (i.e. power AND clock period) are considered.

At 16kB, the pareto curves of the different associativities still largely do not overlap one another except for a single exception in the pareto curve for the 8-way that has a single point more pareto optimal than the 4-way configuration. Of course, it is considered pareto optimal in that it has a valid design point that has a smaller clock period than any point in the 8-way curve, but this improvement in clock period comes at a three-fold increase in read-hit power. Of course, this only tells part of the story with regards to the comparison between the two curves, as the 8-way cache would probably result in a better processor IPC than the 8-way cache because of fewer conflict misses.

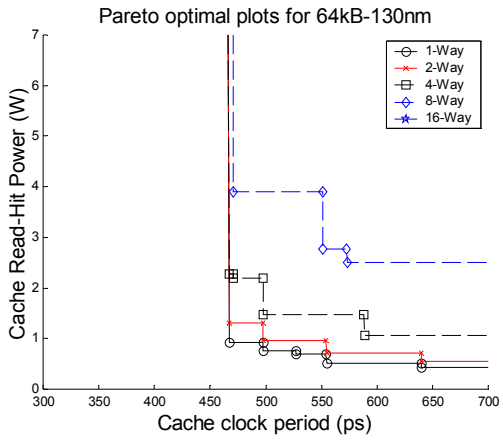
Another interesting observation is that in general, the read-hit power of different points of the 16kB-cache compared to the smaller 8-kB cache are in roughly the same range as their counterparts in the 8-kB cache. In other words, the 16kB cache can typically be implemented (with the same associativity) as a smaller 8kB cache with roughly the same power. In most instances, though, the same does not hold true for the cache clock period (or the cache's critical path delay).

Visually speaking and referring to the pareto plots, when going from a smaller cache to a larger one, the optimal points have values that are close to each other in the y-axis (i.e. read-hit power), but have consistently bigger values in the x-axis (i.e. bigger delay), and this is seen as a distinct shift of the pareto optimal curves to the right (with only a slight shift upwards). This observation seems to roughly hold true for the cache sizes we studied, where the shift of the pareto curves to the right are very distinct, while the shift upwards is much less so.

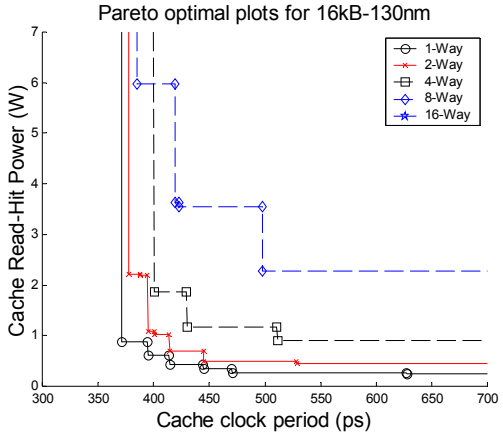
The two previous points show that, when designing caches, it is typically easier to buy lower power at the expense of larger delay, and that buying smaller delay in exchange for higher power is harder and yields less return.



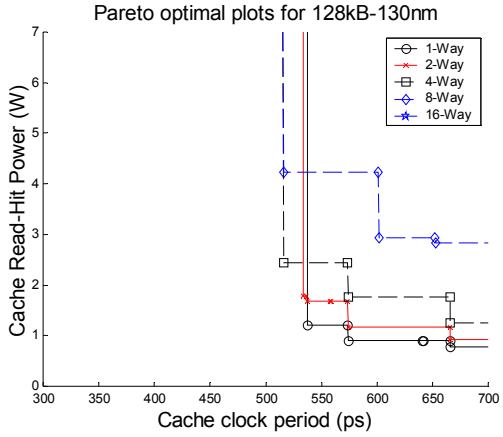
(A)



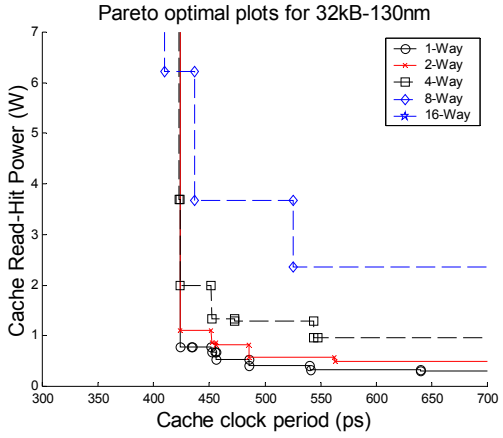
(D)



(B)



(E)



(C)

Figure 5-1: Pareto optimal curves for the 130nm node. (A) 8kB, (B) 16kB, (C) 32kB, (D) 64kB and (E) 128kB. Each plot also superimposes the pareto optimal curves of five different cache associativities (from 1-way to 16-way). Note that, for clarity, the clock period as shown in the x-axis starts at 300ps instead of 0ps.

One last point to make for the 16kB pareto optimal curves is that there is a point in the 16kB 8-way curve that actually has a smaller critical path delay compared to any point in the 8kB 8-way curve. It might seem counterintuitive that a larger cache would have a faster critical path than a smaller cache with the same associativity. The explanation here is that implementing an 8kB cache as 8-way associative results in significant overhead, and these overheads may actually be eliminated with a larger cache that allows better redistribution of the memory array.

At 32kB, the pareto optimal curves start to merge at the lower clock period range, but remain largely distinct from each other at the lower power ranges. Again, this shows that at larger cache sizes, higher associativities allow more opportunity for reimplementing the cache such that faster delay times can be achieved (of course, typically with a significant increase in power). It should also be observed that in the mid-range, the caches with middling associativities offer the possibility of the same clock period in exchange for reasonable increases in access time. This knowledge can be used, in conjunction with processor simulations to determine IPCs using different cache associativities, to decide what particular cache configuration to use to get the optimal design (using the different criteria like cache power, system frequency and processor IPC).

Again, we observe the presence of the single point in the 8-way curve that offers the smallest clock period out of all the points in the pareto optimal curves. The efficiency of having higher associativities is really showing at this cache size.

At 64kB, we see that the fastest points in the four pareto curves (again, the curve for the 16-way configuration is not shown due to clarity) have roughly the same delay, with each point having steadily higher power as associativity increases. At this cache size, the shift of the pareto curves upwards is now more distinct compared to the smaller cache sizes, showing that even the optimal implementations now require more read-hit power compared to smaller caches. Of course, it is also obvious that the shift in read-hit power is much less significant compared to the shift in critical path delay (seen as a right shift in the curves).

Finally, at 128kB, the different curves have merged closer, and we can now see two points (one from the 8-way curve, one from the 4-way curve) that have faster delays than their low-associativity counterparts. The more interesting point is the 4-way point, as the jump in power to this point from the point in the 2-way curve with the best delay is reasonable. Designers can now decide if the additional power is a reasonable sacrifice in exchange for having the faster delay (and of course, the most probably better IPC that comes with having higher associativity). The 8-way point may still be interesting in that it offers a fast clock period (one of the fastest possible) and higher associativity (translating to better IPC) at the expense of significant power dissipation. This point would have been more attractive if it offered a faster clock period compared to its 4-way counterpart.

5.2.2 90nm process technology

The pareto plots for the 90nm process technology for every cache size and associtivity are shown in Figure 5-2.

At 8kB, the pareto optimal curves are still largely non-overlapping, with the only exception being a single point in the 8-way curve that has a slightly smaller critical path delay than any point in the 4-way curves (of course, at the expense of significantly higher power). Also, it is now possible to show the pareto optimal curve for 16-way associativity while still retaining the clarity of the lower associativities, although we can see from the plots that in general, the pareto curves for 16-way associative caches are significantly separated from the other associativities, such that it would only make sense to implement 16-way if the improvement in IPC justifies the slower delays and larger read-hit power compared to even the 8-way associative caches.

At 16kB, we observe a behavior similar to the 130nm node where the pareto curves tend to be shifted towards the right (i.e. increased delay) much more significantly than upwards (i.e. increased power). In addition, the curve for the 16-way configuration stays in roughly the same region (it has actually even improved in terms of delay), again supporting earlier observations that small cache sizes that are highly associative are also highly inefficient in terms of both power and delay. Taken to the extreme, this is a strong reason why fully-associative caches have different implementations (both microarchitecture-wise and circuit-wise) compared to other caches. Implementing fully-associative caches the same way as regular caches would result in very inefficient structures.

At 32kB, as before, we observe that the region of lower delay is now starting to get merged, enough so that the fastest implementations for the caches regardless of associativity (except for 16-way) results in roughly the same clock periods and separated only by their power dissipation in the pareto plots. In addition, we again see that the plot for the 16-way associative cache stays in roughly the same area, showing that 32kB caches can handle the overhead in achieving high associativity better compared to the smaller caches..

At 64kB, we see almost the same things as the 32kB cache, as the pareto curves continue to get shifted towards the right significantly as the cache size increases. Also as before, we observe a much smaller upwards shift of the curve.

At 128kB, the merging of the plots continue, as the 2-way configuration' fastest implementation is now worse than the fastest delay of all other associativities other than 16-way.

5.2.3 65nm process technology

The pareto plots for the 65nm process technology for the different cache sizes and associativities are shown in Figure 5-3.

For the 65nm node, we observe that even at the 8kB cache size, we already observing significant overlapping in the pareto optimal curves. We now see that the 2-way cache has the best possible delay of all

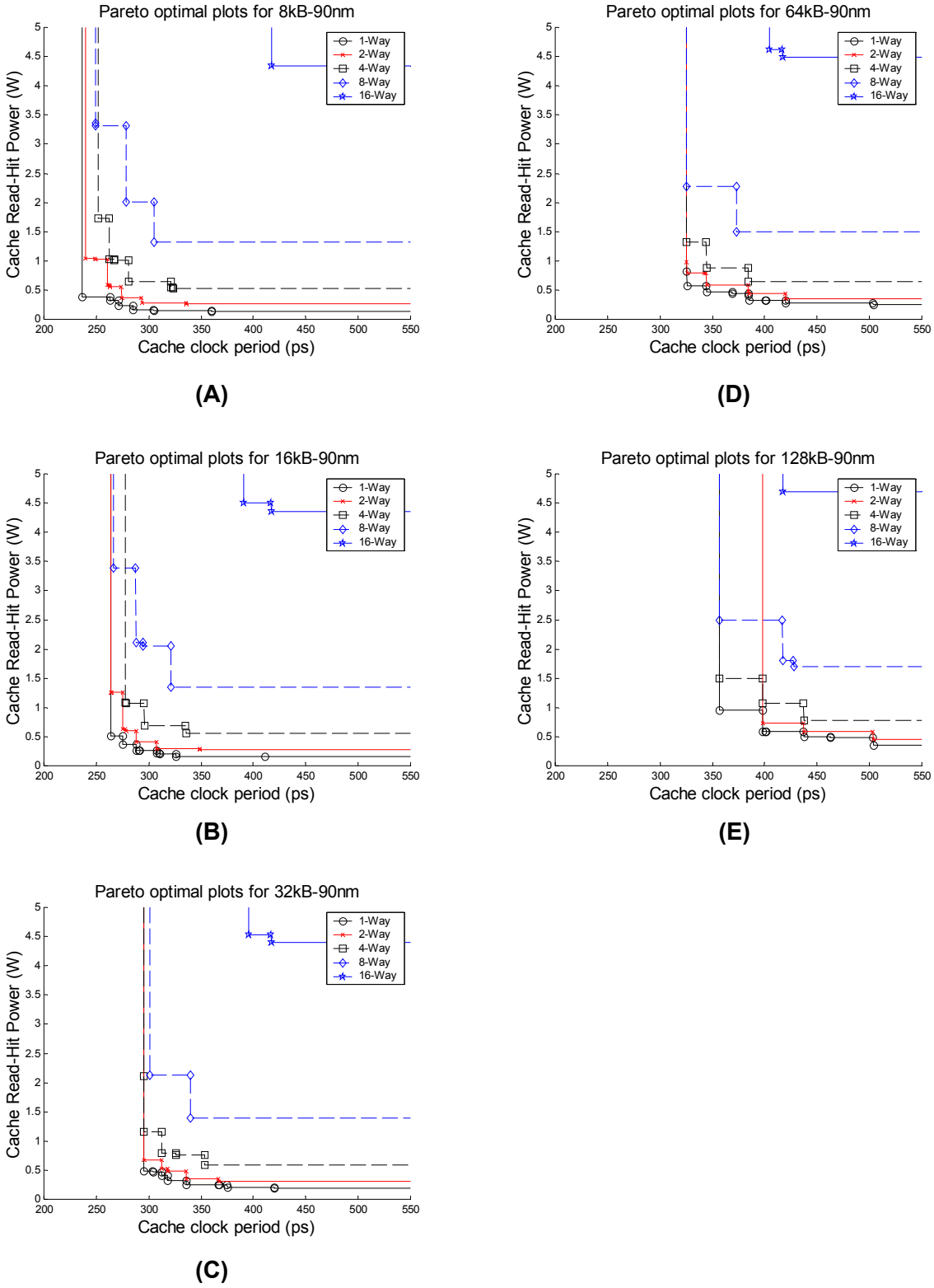


Figure 5-2: Pareto optimal curves for the 90nm node. (A) 8kB, (B) 16kB, (C) 32kB, (D) 64kB and (E) 128kB. Each plot also superimposes the pareto optimal curves of five different cache associativities (from 1-way to 16-way). Note that, for clarity, the clock period as shown in the x-axis starts at 200ps instead of 0ps.

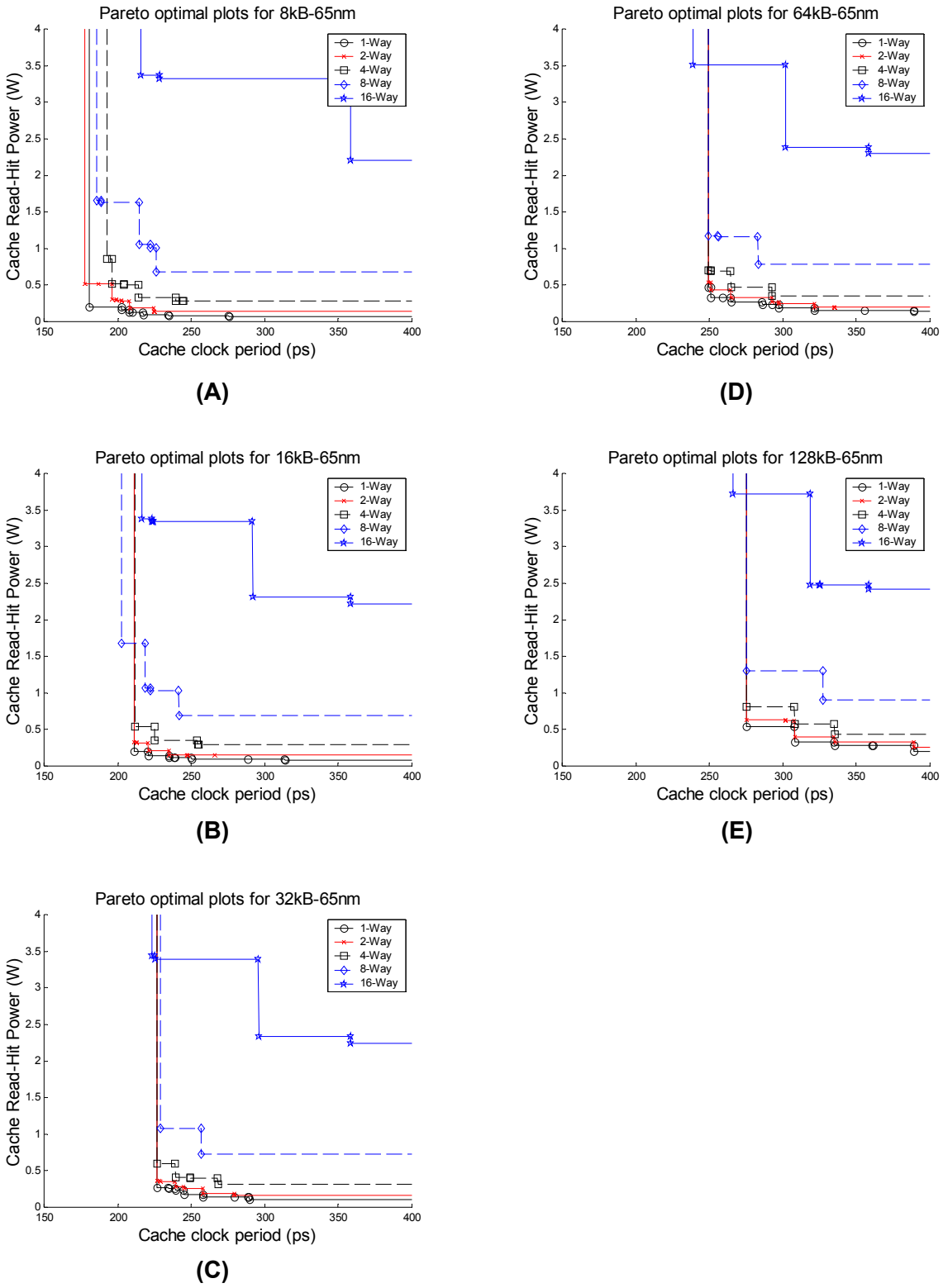


Figure 5-3: Pareto optimal curves for the 65nm node. (A) 8kB, (B) 16kB, (C) 32kB, (D) 64kB and (E) 128kB. Each plot also superimposes the pareto optimal curves of five different cache associativities (from 1-way to 16-way). Note that, for clarity, the clock period as shown in the x-axis starts at 150ps instead of 0ps.

the implementations, while still incurring reasonable increases in power dissipation. We also see that there is significant clustering of optimal points somewhere in the middle representing mid-to-low delays and power dissipation. At this region, the cache designer has a number of choices that can be used to balance clock frequency, power dissipation and processor IPC concerns.

At 16kB, we observe less overlapping compared to 8kB, with the exception of the 8-way configuration having the fastest implementation out of all the other cache associativities and implementatations. It is worthwhile to note that the small improvement in clock period coupled with the anticipated improvement in IPC may be enough to justify the additional power dissipation of this implementation. For applications that really need performance, the 16-way implementations in this case may actually be attractive even though it has a very high power dissipation. The observation that it can be implemented with a clock period that is almost as fast as most other configurations while providing anticipated improvements in IPC might justify this particular implementation for some applications.

At 32kB, again we see a similar behavior in the 16-way implementation where it proves to have the fastest implementation among all the others (at the expense of power). Moreover, we see more clustering than before in the points near the knee of the curves. The more clustered points there are from each of the associativities, the easier the decision of the cache designer it will be, as the tradeoffs will be simpler. For example, a 4-way associative cache that has the same clock frequency as a 2-way cache that dissipates almost the same power will always be more attractive. In essence, we would be gaining IPC for nothing in return.

At 64kB, the 16-way cache now clearly has the fastest implementation. A similar clustering of the points near the knee of the curve is seen, and the previous example still holds true, where it may be easy to choose a 4-way cache implementation here which has roughly the same access time with only a minor increase in power dissipation since we are anticipating an improvment in processor IPC.

Finally, at 128kB, the clustering that we have observed for 32kB and 64kB has somewhat eased, making the decision slightly more difficult in terms of considering the different tradeoffs.

5.2.4 45nm process technology

The pareto plot for the 45nm process technology for the different cache sizes and associativities are shown in Figure 5-4.

For the 8kB cache at the 45nm node, we can see that the pareto optimal curves are already clustered at the knee of the curves, with already at least two implementations resulting in overlaps between the curves (namely the fastest implementations of the 2-way and 8-way associative caches).

At 16kB, the overlapping becomes greater, such that the 4-way and 8-way implementations now seem to have the fastest critical path delays. Choosing between two points would involve a decision between power dissipation and processor IPC. Applications with great need for performance may lean more towards the 8-

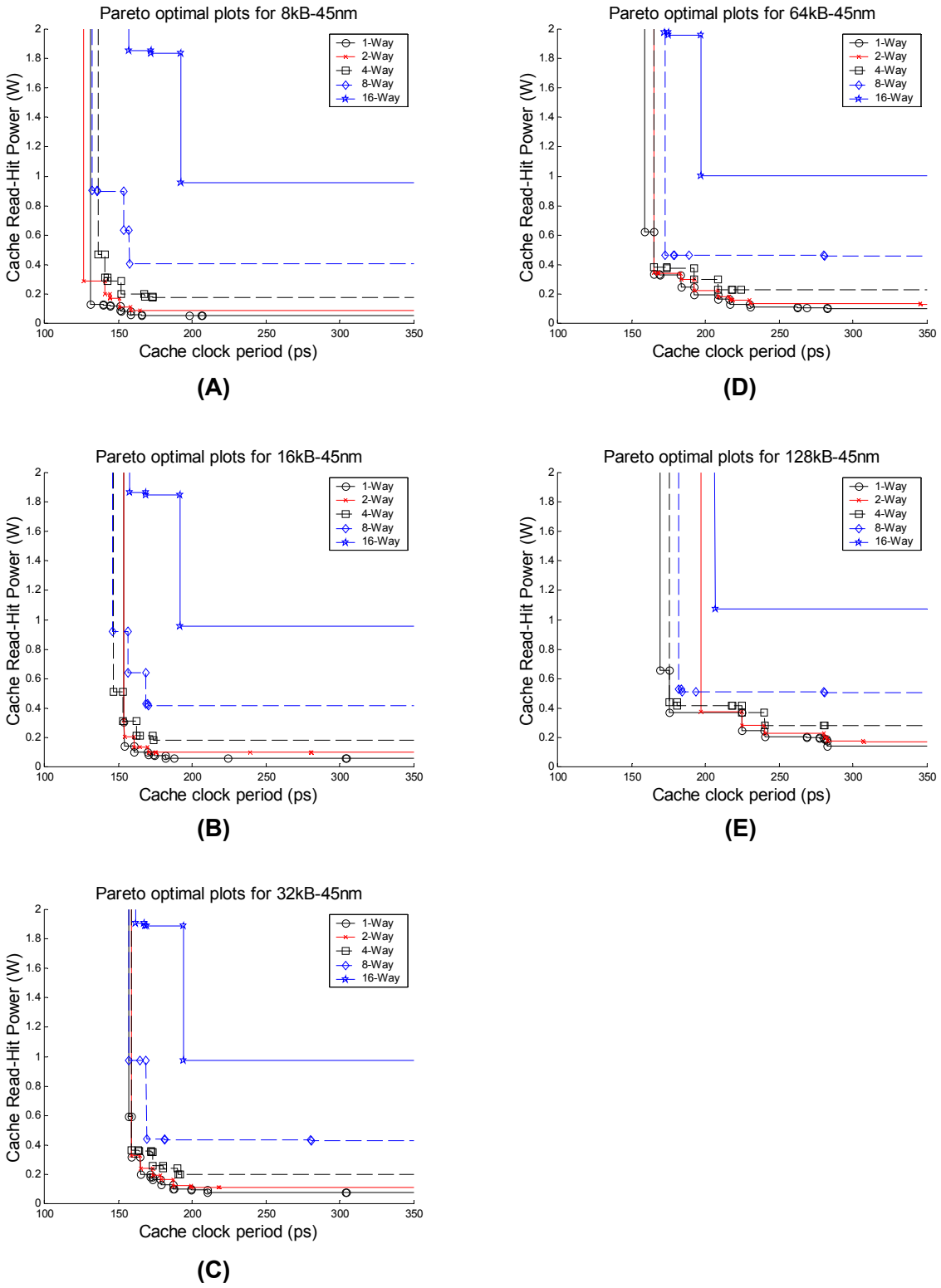


Figure 5-4: Pareto optimal curves for the 45nm node. (A) 8kB, (B) 16kB, (C) 32kB, (D) 64kB and (E) 128kB. Each plot also superimposes the pareto optimal curves of five different cache associativities (from 1-way to 16-way). Note that, for clarity, the clock period as shown in the x-axis starts at 100ps instead of 0ps.

way cache, sacrificing higher power dissipation in anticipation of the IPC improvement of the higher associtivity. In general though, with the observation that typical applications seem to benefit the most from a 4-way cache (i.e. that 4-way associativity seems to be the knee of the curve for typical apps), the 4-way choice might be the best one in this case.

At 32kB and 64kB, we again observe some overlapping and significant clustering at the knee of the curves. Again, this clustering would typically be beneficial since it allows the designer to make a more clear-cut decision over the particular cache configuration and implementation.

Finally, at 128kB, the overlapping is now ver significant with the 4-way implementation approaching the 1-way in terms of the minimum critical path delay, while still having similar power dissipation numbers. In addition, the 8-way configuration now has really approached the curves of the low-associative caches, making it more attractive.

5.2.5 32nm process technology

The pareto plots for the 32nm process technology for the different cache sizes and associativities are shown in Figure 5-5.

For the 32nm technology node at 8kB, the pareto curves of the low-associativity caches already show very significant overlapping. At this cache size, only the 8-way and 16-way configurations show distinct separation from the other curves, although the 8-way curve does exhibit overlapping, with some of its points having faster critical path delays then the 2-way and 4-way associative caches.

At 16kB, the overlapping has eased a bit, showing that the small cache size really contributes to having performance numbers all over the place, as rearranging internal cache structures in the process of finding optimal implementations may result in an optimal solution for one part of the cache while degrading the behavior of some other part. The intertwined behavior of the different parts of the cache become even more pronounced with smaller cache sizes, as there is less of the cache to play with such that each part is exposed to its neighbouring part more.

At 32kB, the overlapping has eased even more as the different implementations become more distinct from each other. At this size, we can see that the 16-way cache can typically be implemented at roughly the same clock frequencies as most of the other implementations. Applications that again need the anticipated IPC benefits of a highly-associative caches may find this attractive even though it has increased power dissipation.

At 64kB, the curves seem to be closer to each other in terms of power dissipation numbers, although overlaps are still few. Having points closer to each other, though, means increased clustering -- again making it easier to decide on tradeoffs involved with cache design.

Finally, the pareto curves for 128kB cache exhibit significant overlapping, with the fastest implementations being the 4-way associative cache followed by the 8-way cache. The 4-way cache really

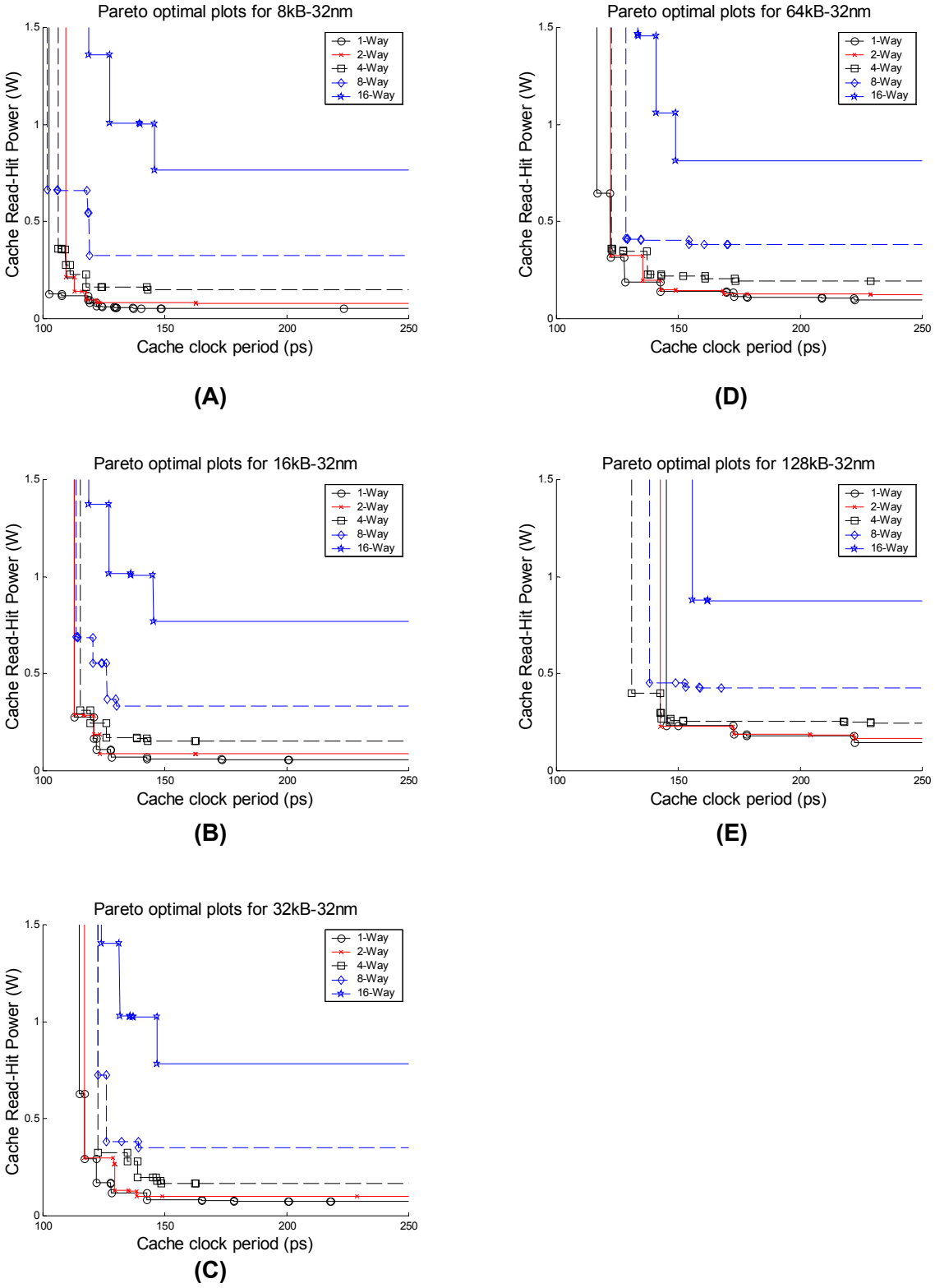


Figure 5-5: Pareto optimal curves for the 32nm node. (A) 8kB, (B) 16kB, (C) 32kB, (D) 64kB and (E) 128kB. Each plot also superimposes the pareto optimal curves of five different cache associativities (from 1-way to 16-way). Note that, for clarity, the clock period as shown in the x-axis starts at 100ps instead of 0ps.

seems to be the most attractive implementation here, as it offers significant improvements in delay over all other implementations (even the second fastest one), while still having reasonable power dissipation numbers. The 4-way configuration even has less power dissipation compared to the next fastest implementation, which is the 8-way cache. Of course the 8-way would have some IPC advantage over the 4-way cache.

5.2.6 Focus on 64kB

The pareto plots of a 64kB cache with different associativities and fabricated in different technology nodes are shown in Figure 5-6.

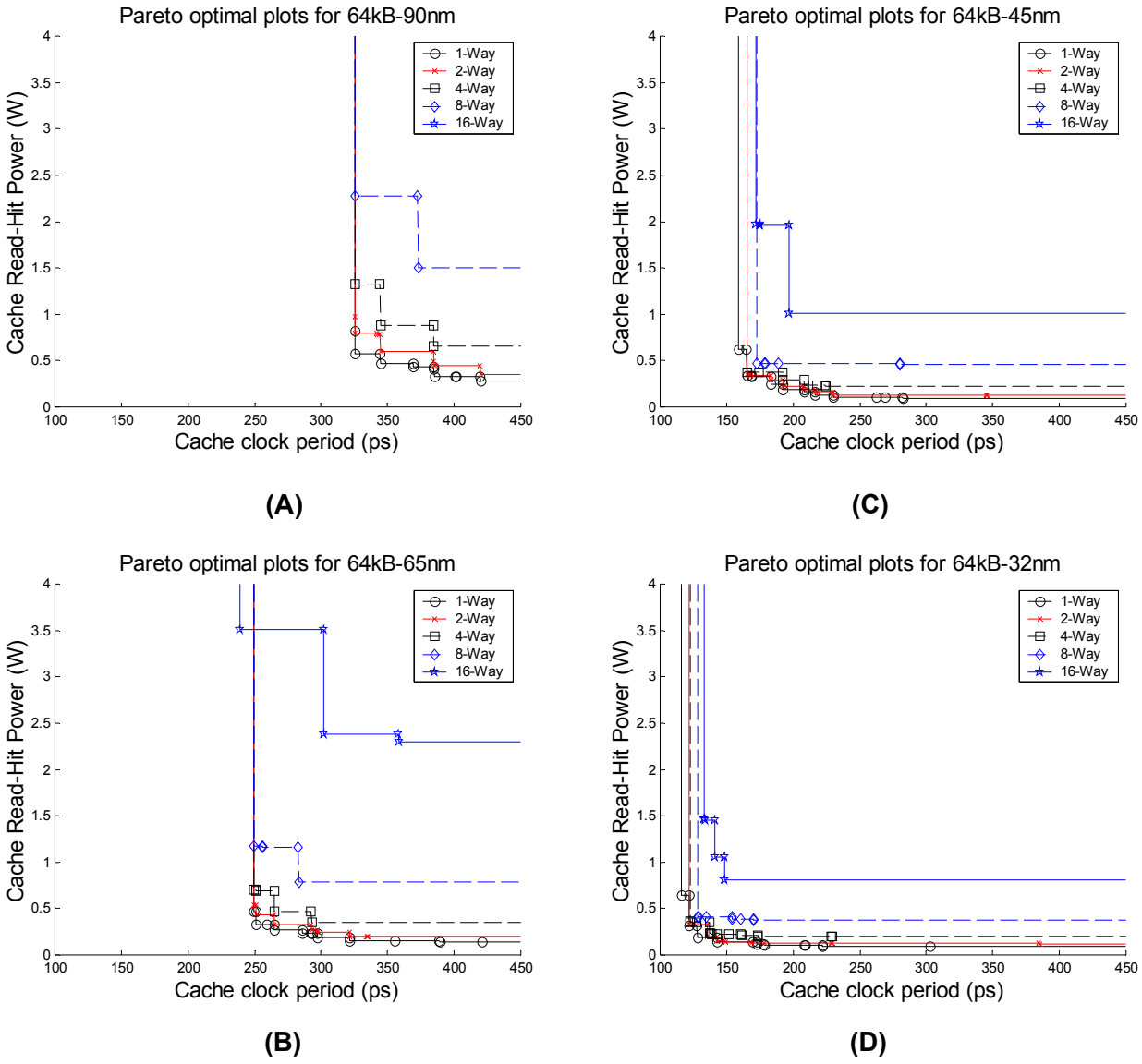


Figure 5-6: Pareto optimal curves for a 64kB cache. (A) 90nm, (B) 65nm, (C) 45nm and (D) 32nm. Each plot superimposes the pareto optimal curves of five different cache associativities (from 1-way to 16-way).

This figure simply repeats the data already shown in some of the previous pareto plots. Instead of grouping plots according to their technology node, this figure now shows plots with the same cache size, specifically 64kB. Four different technology nodes are then shown for a 64kB cache to show the effects of technology node on the pareto plots.

From the figures, we can easily see that the curves get shifted to the left significantly with each technology generation. In other words, caches tend to get faster with improvements in process technology. At the same time, the pareto curves tend to shift downwards with each technology generation, but this vertical shift is significantly less compared to the horizontal shift. This supports what we have seen earlier, where it is easier to buy lower power dissipation in exchange for slower delays compared to buying faster delays in exchange for higher power dissipation.

This observation is consistent with the current trend in industry where power dissipation is becoming a priority concern and, in many instances, the main one, instead of system speed. It is also consistent with the current approach of the microprocessor industry of using multiple cores with middling speeds instead of designing very fast single cores. The kind of behavior we observe in caches jive very well with the current requirements of microprocessor design, where we can achieve lower power without too much difficulty by sacrificing a little on the cache's critical path delay. With clock period requirements not increasing drastically as it used to during the Gigahertz race between Intel and AMD, the tradeoff of getting lower power in exchange for a slower critical path will often turn out not to be a tradeoff at all, since a cache will have more room for slack because of the slower clock frequency requirements.

5.2.7 Summary

This subsection showed one of myCACTI's applications by providing pareto optimal curves for every cache configuration that we study (a crossproduct of cache sizes from 8kB to 128kB, associativities from direct-mapped to 16-way, and the technology nodes 130nm, 90nm, 65nm, 45nm and 32nm). The pareto optimal curves are grouped according to technology nodes for clarity of display and analyses. They are then further grouped into cache sizes, with the pareto optimal curves of different associativities superimposed on the same plot to give a good feel of what happens when a cache is chosen to be a certain associativity instead of another.

We have shown that in many cases, the pareto optimal curves of a cache with a given cache size and technology node will have well-behaved, non-overlapping pareto optimal curves, signifying distinct separations between configurations with different associativities. Although this may at first sound attractive in that we have a "well-behaved" plot this is not the case. This distinct separation actually makes it more difficult to decide between two implementations, as the tradeoffs involved are trickier. We would actually desire the opposite, where we have pareto optimal curves that overlap in some points. This way, it is easier to conclude

that one configuration is better than another, as the overlap will result in particular points being inarguably more optimal compared to another.

Fortunately, we do see this kind of behavior in many cases. For example, we see multiple cases of the 4-way pareto optimal plots overlapping the pareto curves for both direct-mapped and 2-way, signifying that there exists implementations where a 4-way configuration is more optimal (in either power or delay or even both) compared to some other configuration with a lower associativity. Coupled with the fact that we anticipate a better processor IPC for a 4-way cache compared to a cache with a lower associativity (with the same size), this would make it easy to decide that this particular 4-way implementation is more attractive than the other ones.

In any case, the information that myCACTI produces allows the cache designer to make better-informed decisions during cache design.

5.3 Detailed study of a 64kB 4-way cache

The previous section explored a cache design space with the parameters cache size, associativity and technology node, and evaluates every possible implementation of every design point to come up with pareto optimal curves that optimize the cache using read-hit power and cache clock period as the criteria. The previous section shows all the pareto optimal curves of all the configurations, but does not provide any other detailed information. This section aims to provide more details and breakdowns of the cache power dissipation and delays. Figure 5-7 shows the pareto optimal curves for a 64kB 4-way caches for the four nanometer nodes that we study. We can see that the curves are well-behaved in that they are cleanly separated from each other and do not merge or have any overlaps. In other words, the pareto optimal curve of a smaller technology node is always more optimal overall compared to the pareto curve of a larger and older tech node. This should not be a surprising result, given that a process improvement tends to affect the entire cache and as long as the tool is allowed to explore all possible implementations, it will be able to maximize the process improvements of one generation over the previous one.

This last comment is important because the studies in the previous sections that compare CACTI/ eCACTI to myCACTI does not do this exactly. The previous studies, as explained in a previous section, assume a process shrink such that design space exploration is performed on only one technology node. All succeeding technology nodes then are restricted to use the implementation that was found optimal in the original reference node, although since no design space exploration was performed, it is unsure whether the implementation retains its optimality. As explained earlier, although this method does not necessarily produce optimal implementations for the processes that underwent a process shrink, it provides a straightforward method of comparing one process to another, as it allows us to compare caches of the exact same architectural/

microarchitectural implementations that are different only in their physical implementation (i.e. fabrication). Moreover, this process shrink is typically accepted in industry, as it often does not make sense to completely redesign the entire cache for a new process. More often, the exact same implementation for the cache is retained from one generation to another, with changes occuring in sudden jumps (i.e. the design is stable for a few versions of the cache, then when it is justified to sink manpower into a major cache redesign, the implementation may drastically change to take advantage of all the accumulated improvements of the process).

The next subsections will provide detailed power and delay breakdowns of each of the implementation points mentioned in the figure.

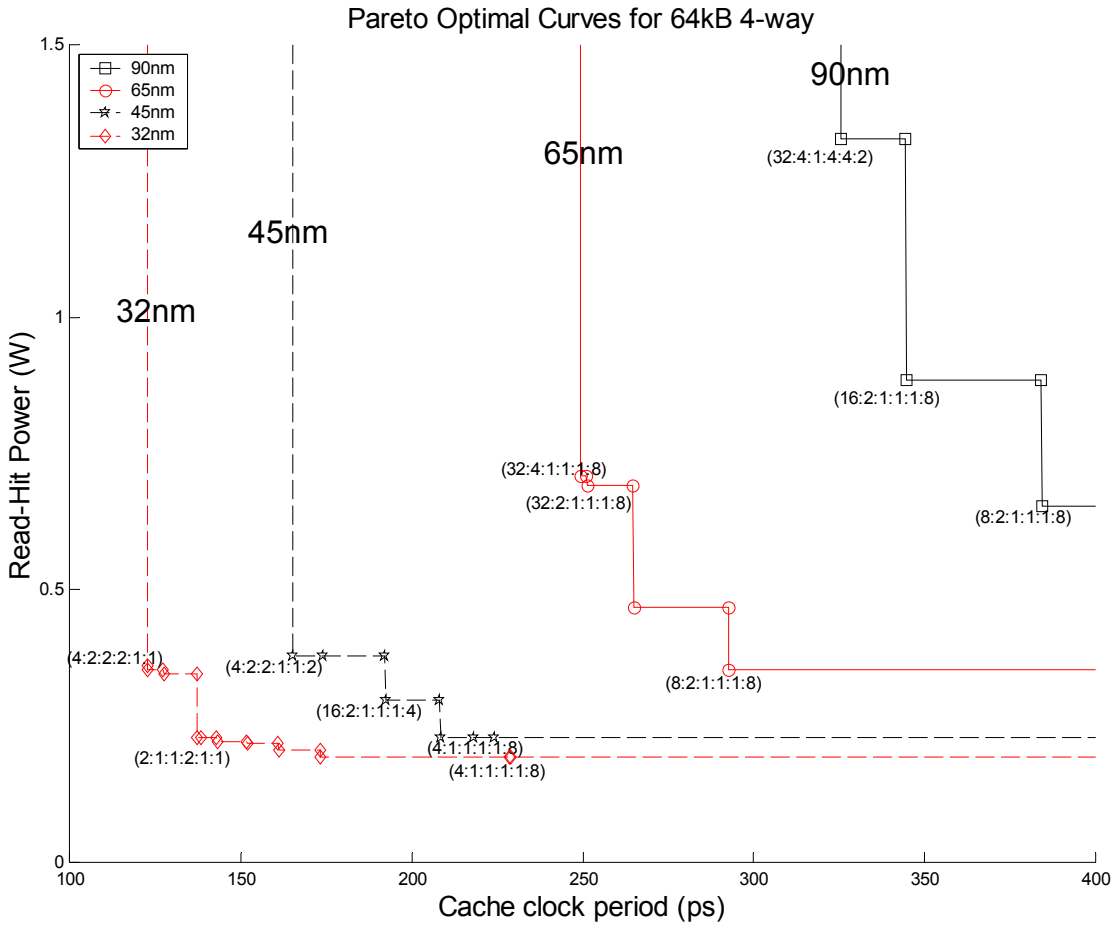


Figure 5-7: Pareto optimal curves for a 64kB-4way cache for different technology nodes. For each technology node, three implementations are identified in terms of the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. These three implementations are representative of three different regions in the pareto curves -- the MinDelay-MaxPower region, the MaxDelay-MinPower region, and the region in between.

The different implementations that are used for the three representative pareto optimal configurations are summarized in Table 5-1.

Table 5-1: Representative Pareto Optimal Implementations for 64kB 4-way caches

	Minimum Delay (Maximum Power)	Medium Delay (Meidum Power)	Maximum Delay (Minimum Power)
90nm	32:4:1-4:4:2	16:2:1-1:1:8	8:2:1-1:1:8
65nm	32:4:1-1:1:8	32:2:1-1:1:8	8:2:1-1:1:8
45nm	4:2:2-1:1:2	16:2:1-1:1:4	4:1:1-1:1:1
32nm	4:2:2-2:1:1	2:1:1-2:1:1	4:1:1-1:1:8

5.3.1 90nm 64kB 4-way detailed breakdowns

Power breakdown. Figure 5-8 shows the detailed power breakdown for a 64kB 4-way cache implemented in the 90nm technology node. The general trend of the cache implementation with power is that the number of data subarrays[1] decreases from 32 to 16 down to 8 serves to also decrease power. For these implementations, the favored way of increasing the subarrays is to increase the wordline partitioning, essentially dividing the global wordlines in order to make the local wordlines shorter. For non-pipelined implementations, subarraying[2] typically yields a sweet spot in power where enough subarrays can be made inactive to conserve power while still not overly increasing the amount of power dissipation required when merging the outputs of

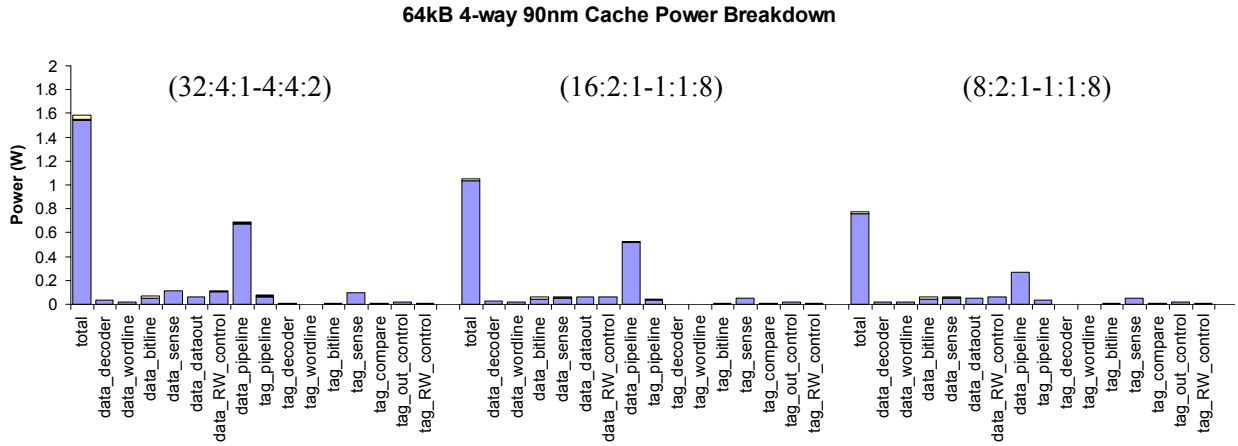


Figure 5-8: Detailed power breakdown for a 64kB 4-way cache for the 90nm technology node.

1. The number of data subarrays for CACTI, eCACTI and myCACTI are computed as the product of Ndwl and Ndbl or, in other words, the product of the amount of wordline and bitline partitioning. The same thing applies to the computation of the number of tag subarrays.

the subarrays together. In the case of our pipelined implementation, though, we see that a majority of the power is dissipated in the pipeline overheads (specifically the data array) such that reducing the number of subarrays significantly reduces the number of pipeline elements needed to implement the pipeline cache. Even though the effects on the other parts of the cache may be more complicated, this reduction in the pipeline overhead with reduced subarraying has such a significant effect on reducing pipeline overhead that it is the main determinant to power.

We see that for the implementation that has the most power, a very significant amount of the power is dissipated in the pipeline overhead. This may not seem like a practical implementation, but we must note that this is still a pareto optimal design point in that it results in the minimum possible critical path delay (and hence cache clock period), but at the expense of significantly increased power dissipation. We can then extend this observation to state that most of this power dissipation increase is due to the increase in subarraying which in turn results in significantly increasing the overhead required in pipelining the required signals within these subarrays.

For the pareto optimal implementation with minimum power, we see that the amount of power dissipated in the pipeline overhead has significantly decreased, although it still constitutes a very significant fraction of the total power dissipation. This again shows that even for the lowest-power configuration, the power dissipated by the pipeline is not trivial and must be accounted for.

Delay breakdown. Figure 5-9 shows the delay numbers of every pipeline stage for the three pareto optimal implementations of a 64kB 4-way 90nm cache, and the delays of each of these stages are broken down and shown in Figure 5-10.

The phase delays of every pipeline stage is shown in the figure for each of the three pareto optimal implementations are shown in the figure, along with the resulting clock period. Note that this clock period is simply twice the phase delay of the critical stage for the particular implementation.

The general trend of the cache implementation in terms of delay times is that the delay increases as the number of subarray increases. Of course, this generalization may not always hold true. Looking at the

---

2. It is important to emphasize that CACTI, eCACTI and myCACTI may use a definition of subarraying that is not exactly the same as other definitions. In some usage of subarrays, the cells can be laid out such that the entire desired data is contained in one subarray such that theoretically, only this particular subarrays has to be enabled to retrieve the data, saving dynamic power in the other disabled subarrays at the expense of additional decoding delay as the subarray enabling has to wait until the proper subarray has been identified. CACTI, eCACTI and myCACTI on the other hand, use a subarraying method that distributes data across multiple subarrays to reduce decoding delay and data output merging delay at the expense of having larger dynamic power because of the need to enable multiple subarrays. As such, increasing the number of subarrays for the three aforementioned cache design tools does not reduce dynamic power as significantly as one would expect if the former subarraying method had been assumed.

different delays of the cache, we see that only a few of the delays actually monotonically increases as the number of data subarrays are increased. Some delays, like the data output decreases with the number of subarrays. We observe that only the data wordline drive-bitline-sense stage experiences significant increases in delay with increased subarraying, but since this stage is consistently the critical path, it directly determines the clock period of the cache.

The detailed delay breakdown of every pipeline stage in the cache is shown for the three pareto optimal configurations, with the dashed rectangle indicating the critical stage. For each stage, the different components of its delay are indicated to show the effects of using different implementations on these delay components.

For the 90nm 64kB 4-way cache, the three pareto optimal implementations all have the pipeline stage Data_2A as their critical stage, with this stage containing the wordline driver, the bitline and the sense amplifier. Also shown for each stage is a fixed flop delay consisting of the initial clock to output delay from the previous stage state elements followed by the final flop set-up time that is required by the current stage's latch to retain the data properly. We see that aside from this stage, the two other stages with the closest delays are the tag compare stage (Tag_2A) and the data output stage (2B), with this being true for all three optimal implementations. This means that even if we improve the Data_2A stage significantly, the total cache clock period will not enjoy the full improvement, as it will see the improvement only until the point is reached where some other pipe stage has the longest delay and as such, becomes directly responsible for the cache clock

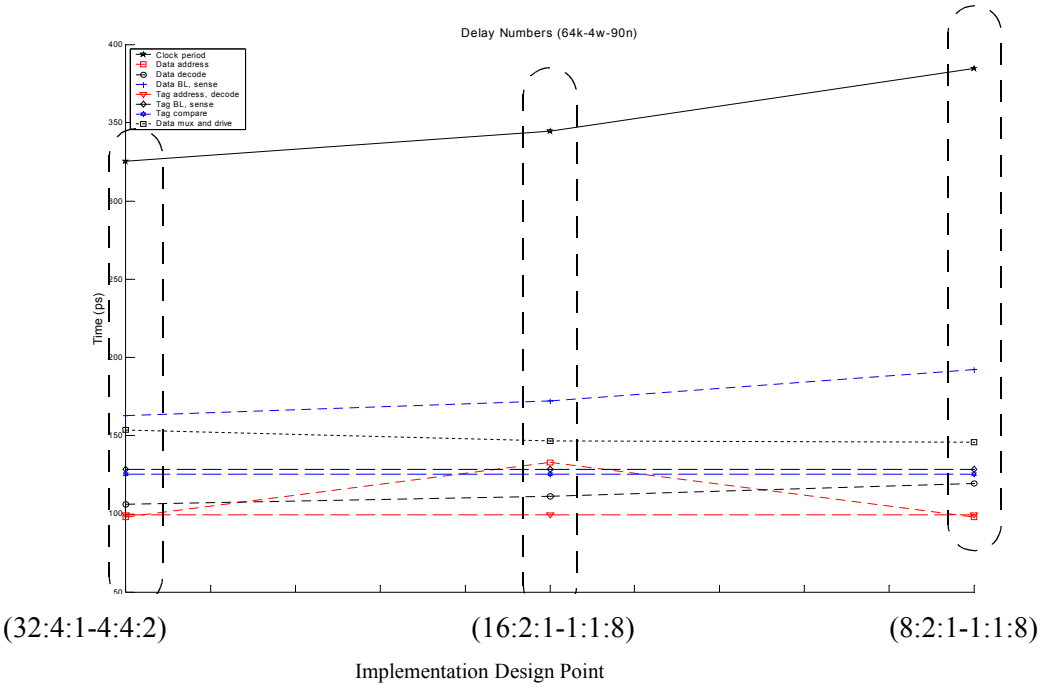


Figure 5-9: Delay numbers for three pareto optimal implementations of a 64kB 4-way 90nm cache. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power), the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the pareto curve.

period. Any further improvements in the delay of the Data_2A stage beyond this point is essentially useless and only serves to increase the timing slack for this particular stage. Note that this is not entirely useless, as having larger timing slack is often desirable up to some point as it reduces the number of important timing paths in the system resulting in less points of failure once PVT variations and other variables are accounted

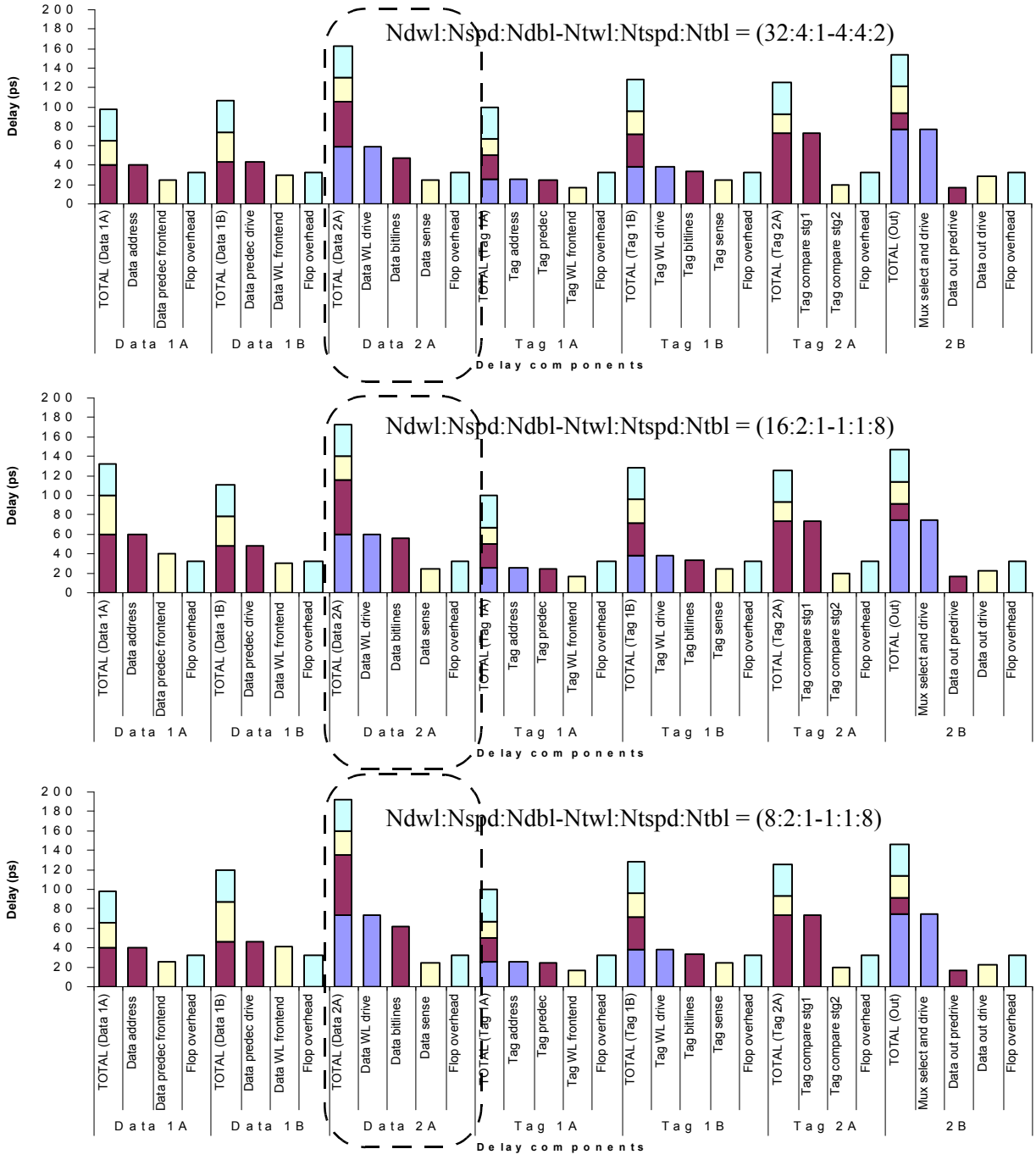


Figure 5-10: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 90nm cache. These plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the dashed .rectangles denote the critical path delay of the cache that sets its clock period.

for. Of course, this will have practical limits, as it doesn't make sense to devote extensive effort to increase a stage's slack if it results in sacrificing some other aspect of the design that more obvious consequences.

5.3.2 65nm 64kB 4-way detailed breakdowns

Power breakdown. The power breakdown of a 65nm 64kB 4-way cache is shown in Figure 5-11. The power trend for the 65nm 64kB 4-way cache is similar to the 90nm implementation, although the particular implementation chosen for the middle optimal point does not reside in the knee of the pareto optimal curve and as such, is close to the first point both in power and delay.

One significant difference that we observe with the power breakdown is the non-monotonicity of the pipeline overhead power, where the middle optimal point actually has a larger power in the pipeline compared to the MaxPower implementation (i.e. the first point). This means that the implementation of the middle point has resulted in significantly smaller power dissipation in the cache in general except for the pipeline overhead. For the purpose of our optimization, we are looking only at the total power (and delay) for the optimization, so it is not of immediate importance which part of the cache dissipates the most power (or accounts for the most delay). In more detailed studies and designs, though, this kind of information is useful in isolating which part of the design wield yield the most gains for a given effort such that it is these parts of the cache that makes most sense to focus on. In this particular case, the cache designer can choose to utilize the middle implementation because of the small power dissipation of majority of the cache components while still having a faster delay compared to the MinPower implementation (i.e. the third point). Design effort can then be focused on designing more power efficient latches that dissipate significantly less dynamic power than the particular latch implementation that we utilize. This is an example of how the information that our tool provides can be utilized during cache design.

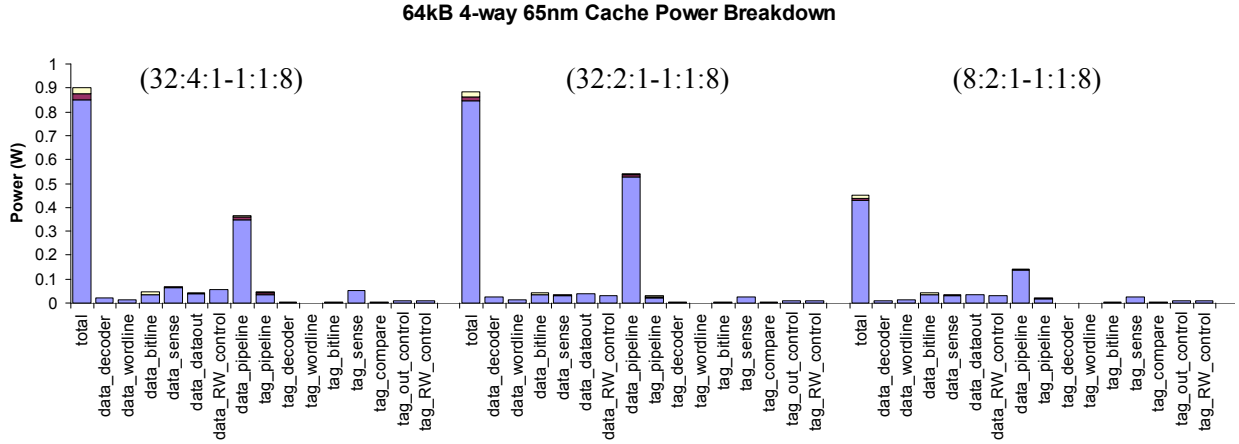


Figure 5-11: Detailed power breakdown for a 64kB 4-way cache for the 65nm technology node.

At this technology node, we can see that the gate leakage and subthreshold leakage are roughly the same, and that they constitute only a very small fraction of the total power. It should be emphasized, though, that we are assuming a dual-Vt process here, such that subthreshold leakage is significantly reduced by using HVT transistors in both the non-critical path, and the entire memory cell significantly reducing leakage.

Delay breakdown. The delay trend for the 65nm 64kB 4-way cache is also similar to its 90nm counterpart, as shown in Figure 5-12 and Figure 5-13. Again, the phase delays of every pipeline stage is shown along with the total cache clock period.

Again, it is the data wordline driver-bitline-sense amp pipeline stage that is the critical path for all three implementations. We also have a similar observation where it is only this stage that shows a monotonic increase in delay, with most other stage delays showing different behavior.

Also similar is the two stages that have the next slowest delays, which are the tag compare stage and the data output driver stage. From what we can see from the delay plots, improving the data wordline driver-bitline-sense amp stage will result only in moderate delay gains before effort has to be spent on improving both the tag compare and data output stage. On the contrary, focusing effort on improving the dealy of the data wordline driver-bitline-sense amp stage for the third implementation (the MinPower-MaxDelay point) has the potential of resulting in significant improvement in delay even without improving any other stage, as the difference in the maximum delay and the next largest delay is much larger compred to the first two

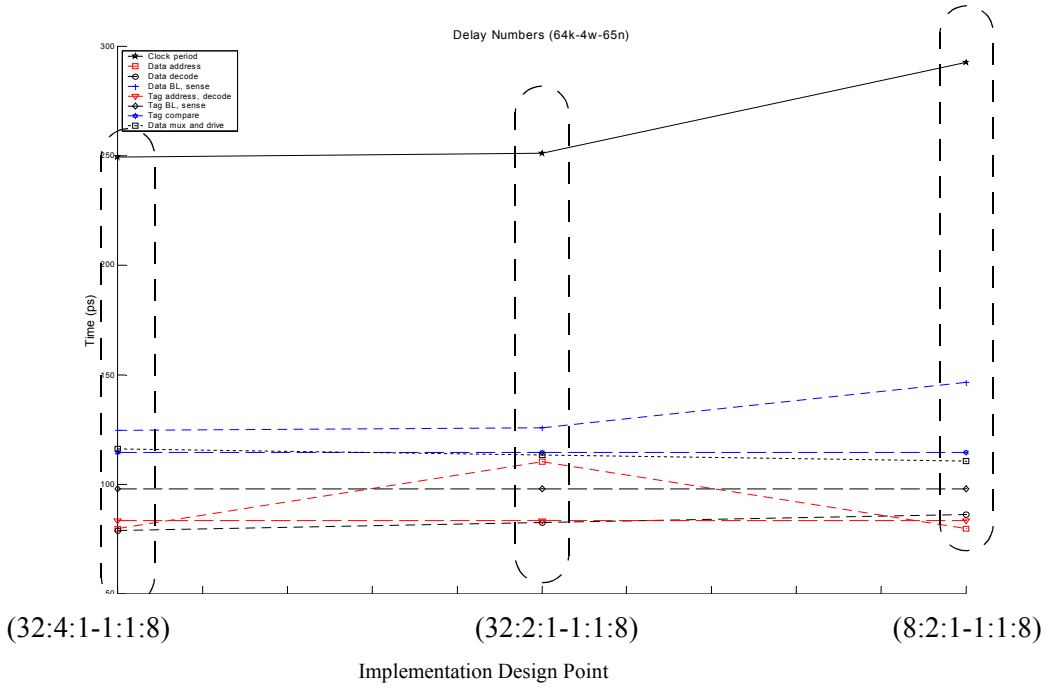


Figure 5-12: Delay numbers for three pareto optimal implementations of a 64kB 4-way 65nm cache. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power), the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the pareto curve.

implementations. Again, this shows a good example of how these data and information can be used to focus design effort properly.

Figure 5-13 shows the detailed breakdown of the delay of every stage for the three pareto optimal implementations. As mentioned earlier, the stage Data_2A sets the cache clock period for all three implementations. It is worthwhile to note that the sense-amp delay is assumed constant for every

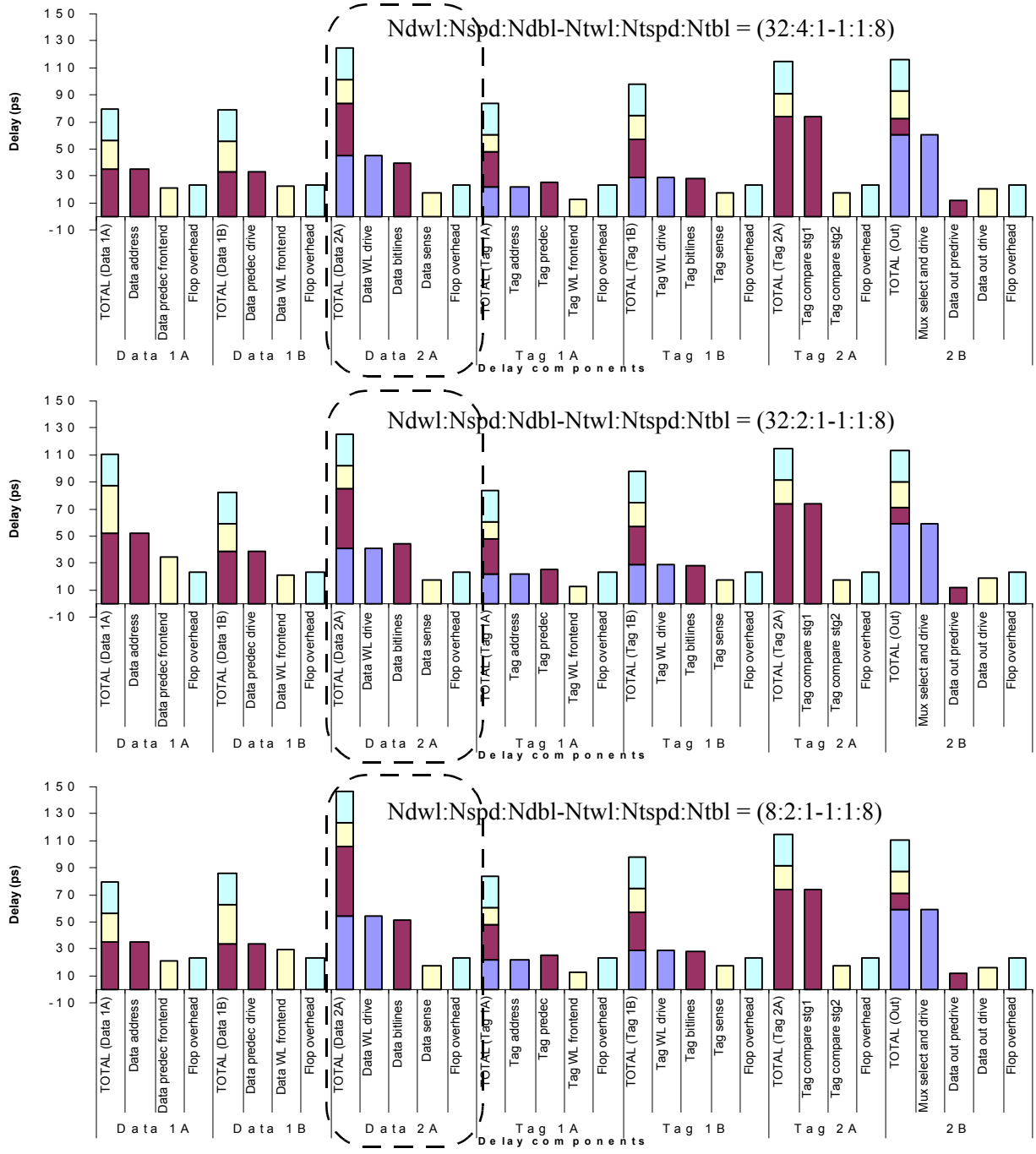


Figure 5-13: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 65nm cache. These plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the dashed .rectangles denote the critical path delay of the cache that sets its clock period.

configuration (this is also done by CACTI and eCACTI). This is a valid assumption, as the only requirement by the sense-amp is the presence of enough differential voltage at its input to differentiate between a logic high and logic low. The algorithms assure that the sense-amp is fired only when enough differential voltage exists, and not before. Once the sense amplifier fires, the input does not play a significant part in determining the

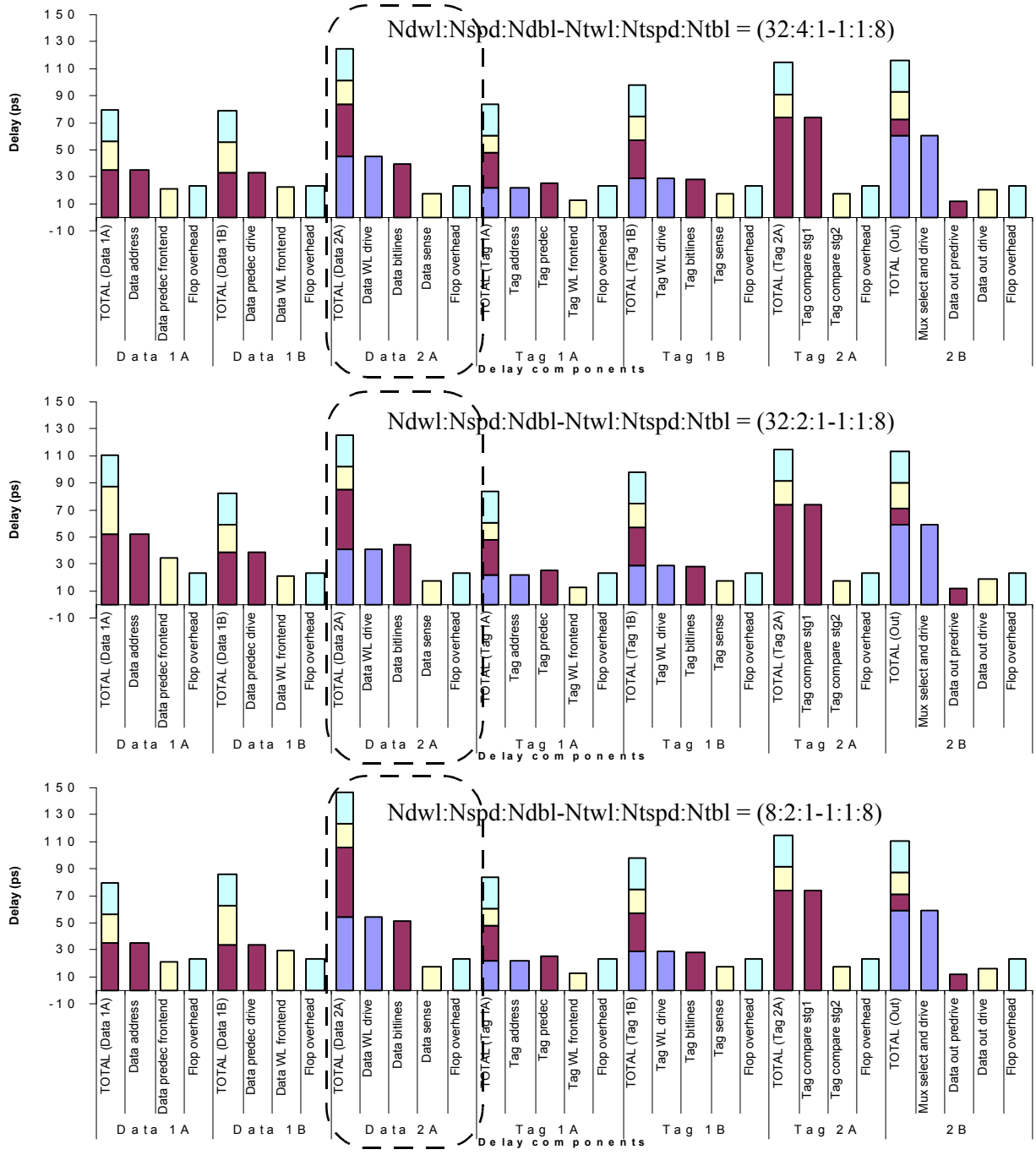


Figure 5-14: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 65nm cache. These plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the dashed .rectangles denote the critical path delay of the cache that sets its clock period.

delay, especially since the inputs are typically being discharged by very weak transistors in the memory cell such that the transition delays would be much greater than the propagation delay of the sense amplifier.

5.3.3 45nm 64kB 4-way detailed breakdowns

Power breakdown. The detailed power breakdown of a 45nm 64kB 4-way cache is shown in Figure 5-15. The power trends observed in the two previous technology nodes now do not hold true for the 45nm node. Specifically, the trend that power decreases as the number of subarray decreases (mostly because of the decreased power dissipation in the pipeline overhead) now does not hold true, as the number of subarrays for the MaxPower implementation to the MaxPower implementation now goes from 8 to 16 back to 4 subarrays (for the data). Looking at the MaxPower implementation, we see that the relatively small number of subarrays have resulted in a small pipeline overhead, but this is more than made up for by the larger power (mostly dynamic) in the other cache components like the data bitline, sense amplifiers and data output circuits. For the MidPower-MidDelay implementation, the number of subarrays goes up resulting in more than doubling the pipeline power dissipation, but the dynamic power of most other cache components have decreased resulting in a net decrease in power. But with the MinPower implementation, the number of subarrays goes down even further than for the MaxPower implementation, but the pipeline power dissipation does not go as low, while still having relatively low power dissipation for the other cache components. This emphasizes that the subarraying technique used by CACTI, eCACTI and myCACTI creates different types of subarrays depending on whether the wordline (Ndwl) or the bitline (Ndbl) is partitioned. The same holds true for partitioning the tag array.

Again, this kind of information can be useful during cache design. Although total power and delay are the immediate criteria used by the optimization algorithms, the cache designer is still in charge of balancing out the different options when choosing between different design tradeoffs. In cases where it is hard

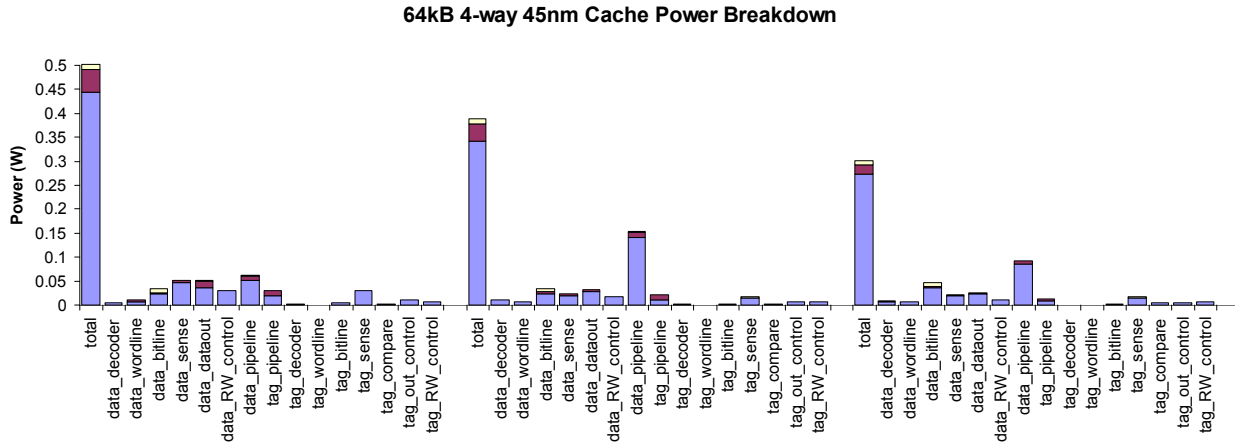


Figure 5-15: Detailed power breakdown for a 64kB 4-way cache for the 45nm technology node.

to objectively decide whether a small gain in one criteria and a small loss in a second criteria is better compared to the reverse with a small loss in the first criteria but also a small gain in the second. In this types of instances, a secondary factor that can come in is the ease of redesigning the cache to gain substantial improvemnt. With this in mind, it might not be advisable to choose the MaxPower implementation as it may be harder to improve later on because of the distributed nature of power. A better decision might be to choose the MidPower-MidDelay implementation and its better delay compared to the MaxDelay-MinPower implementation and accepting the substantial power increase because further focused design effort on just the pipeline overhead circuits would yield significant gains in power.

This outlines potential design flow in the use of myCACTI as part of a complete custom cache design tool methodology, where it can be used to produce a first-pass pareto optimal implementation possibilities (and maybe ones that are close to being pareto optimal) and then studying those implementations in detail to determine whether a focused design effort on one or two components will yield substantial gains enough. With this approach, it may be possible to identify the implementations like MinDelay-MaxPower where it is possible to do some custom optimizations to get lower power, resulting in an implementation that has the minimum delay possible while having reasonable power dissipation. The same holds true for the possibility of finding MaxDelay-MinPower implementations where it may be reasonable to improve on the delay through focused effort.

Delay breakdown. The delay numbers for a 45nm 64kB 4-way cache are shown in Figure 5-16. Although the 45nm delay trend for the data wordline driver-bitline-sense amp stage is similar to the 90nm and 65nm nodes, data out driver now exhibits more delay for the MinDelay-MaxPower stage, setting the critical path. All other stages exhibit a similar (but not the same) fluctuations as were observed for the previous technology nodes.

We can also see in the figure that the stage with the next slowest critical paths are now much closer to the prevailing critical path, especially for the two outside points. This means that trying to improve the cache clock period will require focused effort on more than one part of the cache to produce substantial gains in delay.

Figure 5-17 shows the detailed breakdown of the delay of each pipeline stage, again showing the critical path stage for each implementation. It now shows that the stage 2B setting the cache clock period for the MinDelay-MaxPower implementation instead of Data_2A. For the stage 2B, we see that the largest component of the delay is due to driving the multiplexing drivers from the result of the tag compare and multiplexer selectors, with a small delay that is only slightly larger than the latching overhead for actually driving the data from the cache internals to its periphery.

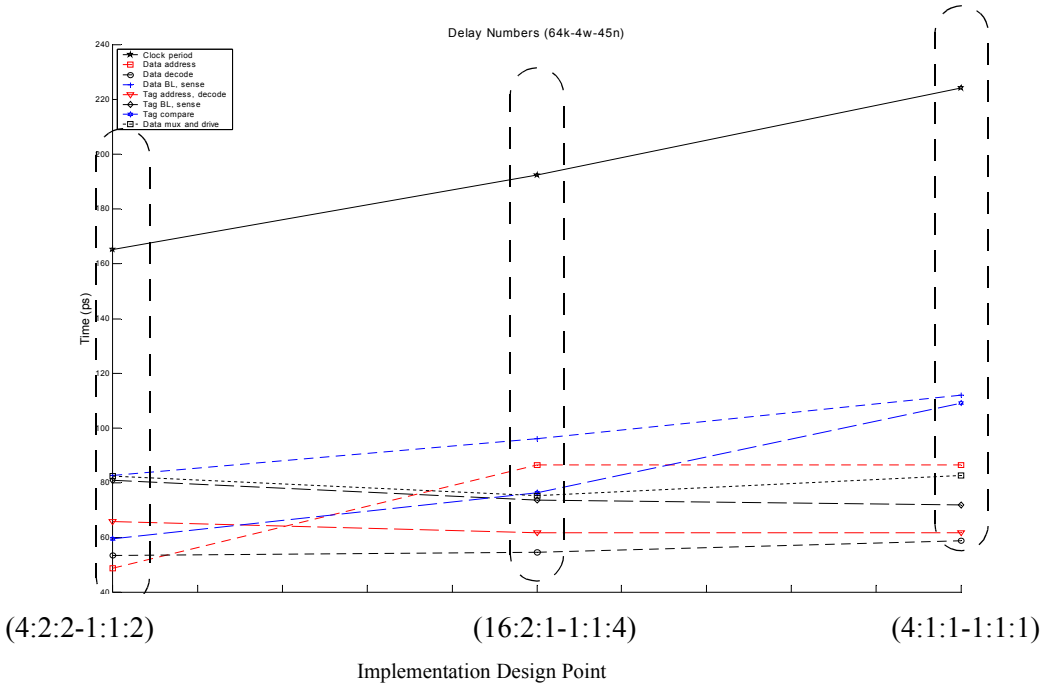


Figure 5-16: Delay numbers for three pareto optimal implementations of a 64kB 4-way 45nm cache. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power), the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the pareto curve.

5.3.4 32nm 64kB 4-way detailed breakdowns

Power breakdown. The powre breakdown of a 32nm 64kB 4-way cache is shown in Figure 5-18. Like the 45nm node, the subarray power trend that was observable in 90nm and 65nm also does not hold. What is interesting to observe is that even though the number of memory cells in the cache stays the same assuming a fixed cache size. We observe that for the max power implementation, the subthreshold leakage components in

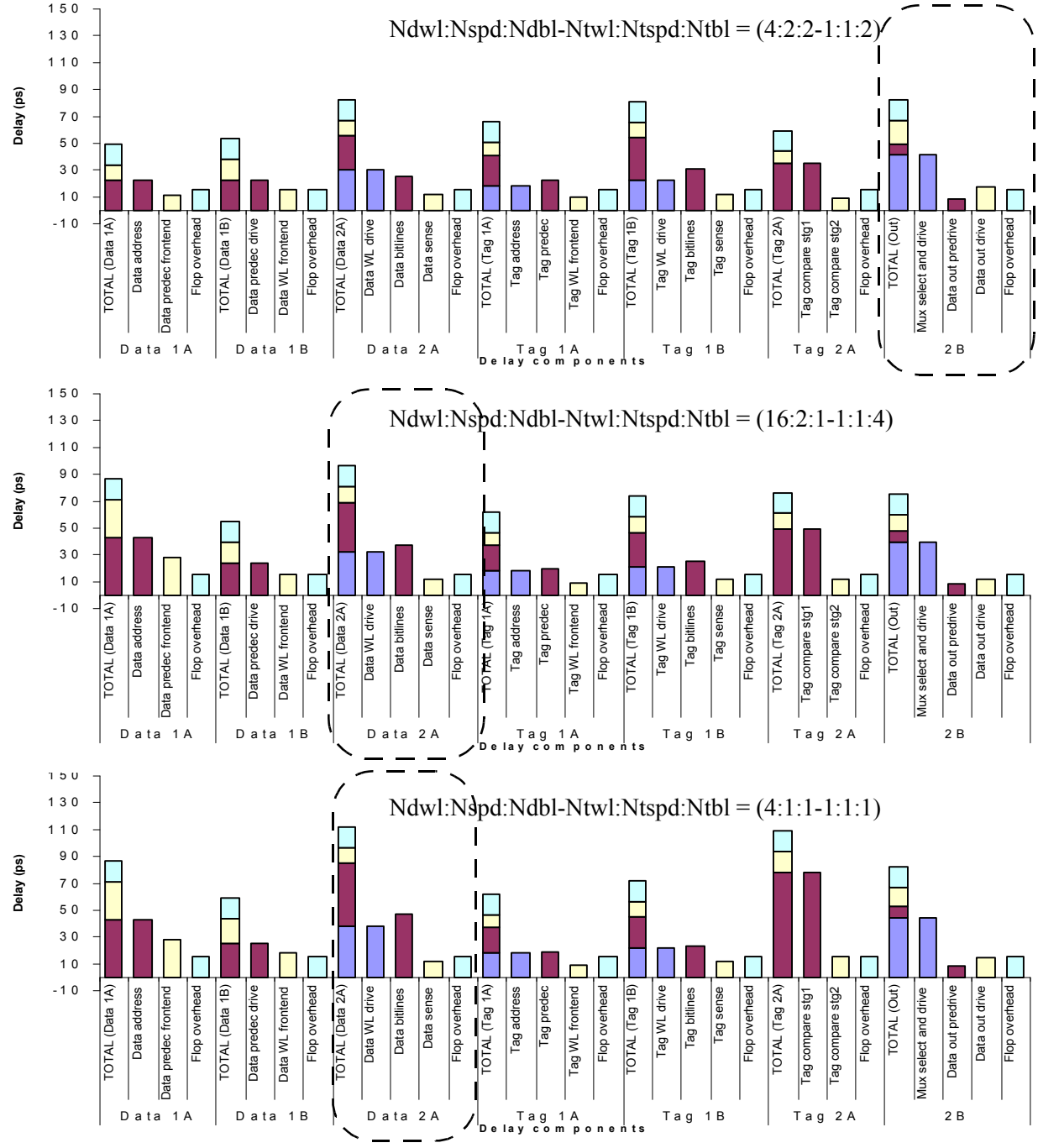


Figure 5-17: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 45nm cache. These plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the dashed .rectangles denote the critical path delay of the cache that sets its clock period.

the data_dataout, data_pipeline and tag_pipeline for the MaxPower-MinDelay implementation are much more significant compared to the other two implementations even though the cache size stays the same. This change can be explained by the changing number of circuits along with the sizes of the devices inside these circuits to account for the specific loading given a particular implementation.

Unlike the previous nodes where the pipeline overhead is typically the dominant component of power in the design, its contribution is notably less here. Although itis still still significant for all three implementations, it is not overly dominant unlike before. - With regards to the complete cache design flow that was earlier suggested, these particular results are not particular useable for the discussed flow, as there are no absolutely dominant component of cache power where design effort can be focused on to target power improvement. In this situation, the flow can still be made to produce more optimal results by expanding the detailed analysis to not just these three pareto optimal configurations, but to all the points in the pareto optimal curves as well as points that are close to it.

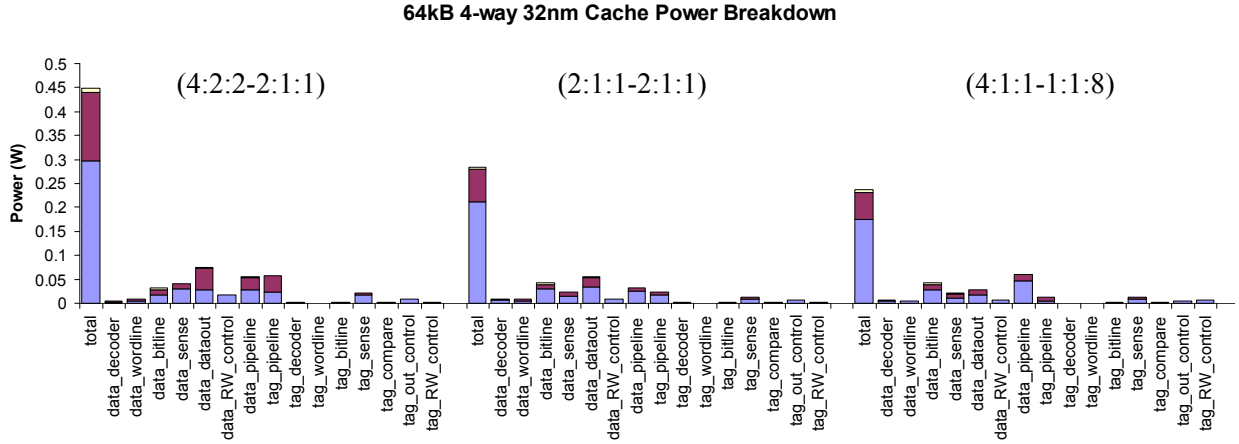


Figure 5-18: Detailed power breakdown for a 64kB 4-way cache for the 32nm technology node.

Delay breakdown. The delay numbers for a 32nm 64kB 4-way cache are shown in Figure 5-19. For the 32nm node, the critical stages are now different for the three implementations. Although the behavior of many of the stages are still largely the same (e.g. the data wordline driver-bitline-sense still shows a monotonic increase), jumps in some of the stages result in a change of behavior. For example, the sudden jump in delay of the tag compare stage for the MaxDelay-MinPower implementation results in a critical path that is significantly larger than the previous critical path.

Figure 5-20 shows the detailed breakdown of the delay of each pipeline stage, again showing which stage sets the cache clock period for each implementation. For the 32nm node, different stages now set the critical path of the cache for the different implementations.

5.3.5 Detailed power and delay breakdown summary

This section analyzed a specific cache configuration, specifically a 64kB 4-way cache, in detail for the 90nm, 65nm, 45nm and 32nm node. Detailed power and delay breakdowns were shown for three pareto optimal implementation for each node to show the range of possibilities when implementing the cache. Specifically, we have given detailed power and delay breakdowns for the implementations with minimum delay and maximum power (MinDelay-MaxPower), the implementation with maximum delay and minimum power (MaxDelay-MinPower), and an in-between optimal point (MidDelay-MidPower) to provide users with a choice depending on their own particular requirements. The default optimization of CACTI and eCACTI is

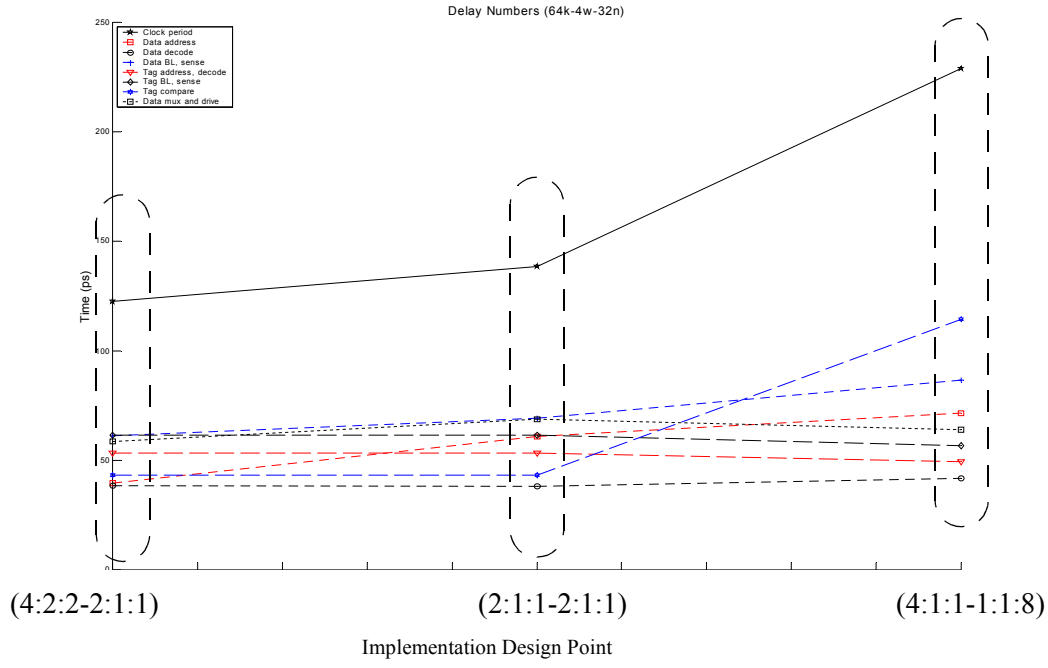


Figure 5-19: Delay numbers for three pareto optimal implementations of a 64kB 4-way 32nm cache. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power), the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the pareto curve.

to use weight all criteria equally, while myCACTI has changed this by recognizing that in typical microprocessor designs, a target frequency has to be met and all stages have to follow the specified clock period. With this in mind, the myCACTI optimization eliminates delay from the weighing optimization, and as long as an implementation passes a minimum clock period, the delay is ignored in further calculations. This

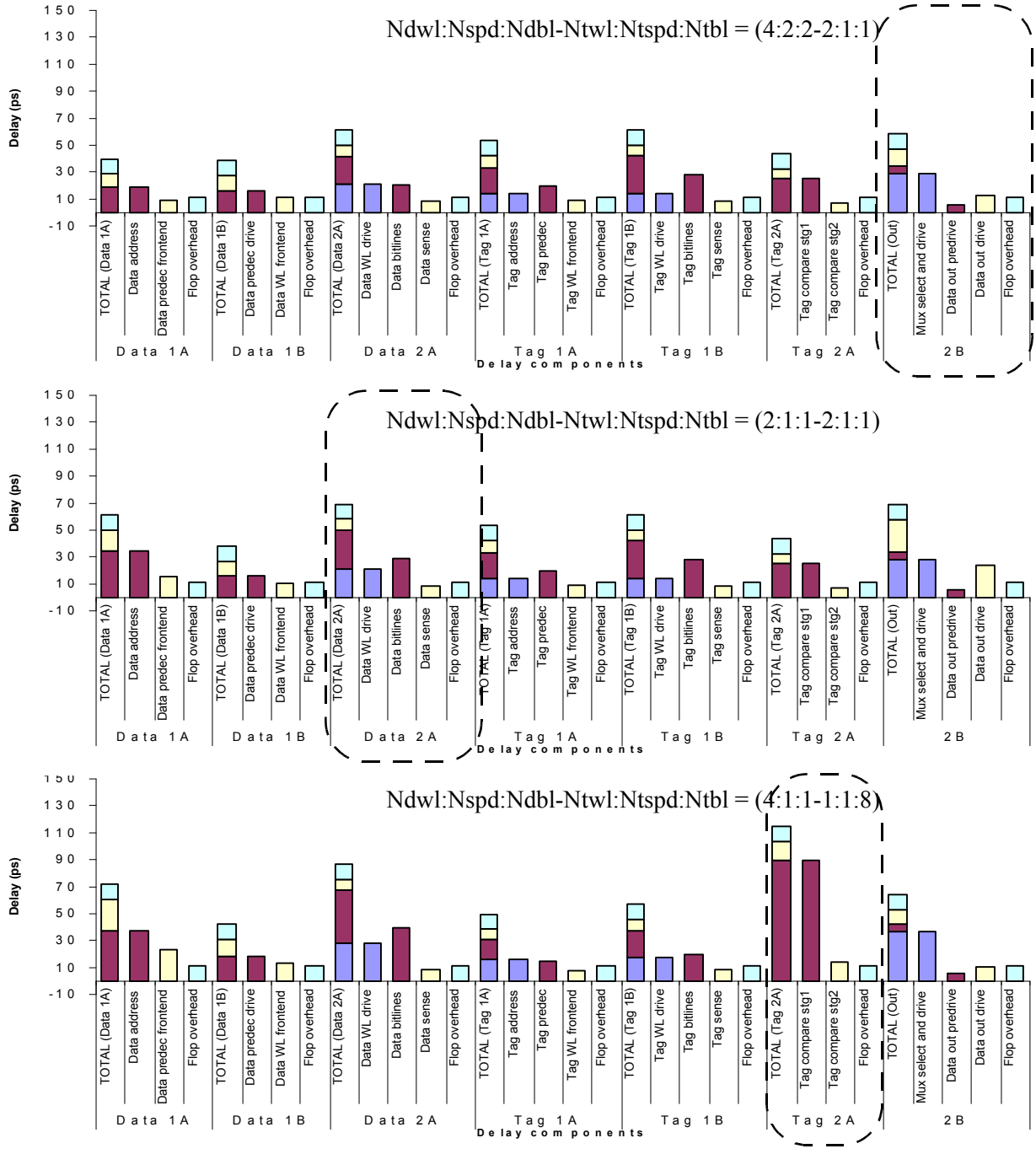


Figure 5-20: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 32nm cache. These plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the dashed .rectangles denote the critical path delay of the cache that sets its clock period.

section provides a third approach where the pareto optimal configurations are shown to the user and analyzed in detail.

One important comment that was explained in the section is the myCACTI's enabling of a complete custom cache design flow where myCACTI is used to isolate candidate points (most likely the ones on or close to the pareto optimal curves) as a first pass in choosing which design to implement. The second pass then involves analyzing some of these points in detail to examine the various tradeoffs involved in power and delay. Since myCACTI enables detailed analysis of the breakdown of power and delay, it is possible to point out implementations where improvements in a single area of a cache (say due to a focused design effort on that area) can yield significant gains and may result in a point that is clearly much more optimal than any other point in the pareto curve. One example is the possibility of finding a MaxDelay-MinPower implementation that has most of its delay accounted for by a single stage and where it is reasonable to expect that focused design effort on that stage will yield gains in delay. In this manner, we now have an implementation with the minimum power, but with an improved delay that may be significantly smaller than the original maximum delay.

5.4 Cache design technique comparisons

The two previous subsections have discussed cache design space explorations using myCACTI default settings (footnote: these are settings that we consider to be the typical choices for a cache implemented in early nanometer technology like 90nm). The first subsection demonstrated design space explorations for a wide range of cache configurations including caches with sizes from 8kB to 128kB, associativities from direct-mapped to 16-way, and process technologies of 90nm, 65nm, 45nm and 32nm. After the design space exploration, pareto optimal curves for each specific configuration (consisting of a given cache size, associativity and techology node) were given and analyzed. The second subsection then explored cache behavior in terms of cache read-hit power and delay in more detail by assuming a given cache size and associativity (64kB 4-way), and then providing data and analyses of the detailed power and delay breakdown for the different components and pipeline stages of the cache.

This subsection will perform a detailed study similar to the second subsection, but instead of using myCACTI defaults, three implementation parameters will be varied, namely the use of single-Vt or dual-Vt transistors, the use of static or dynamic decoding, and the use of low-swing differential or full-swing single-ended sense amplification. Pareto curves and detailed power and delay breakdowns are given for each of the eight possible combinations of the three parameters. Note that to reduce the amount of data, the studies use a single cache configuration, namely a 32nm 64kB 4-way cache.

5.4.1 Pareto optimal curves

The pareto optimal curves for each of the eight possible implementations are shown in Figure 5-21. For clarity, the implementation parameter sextet are omitted for each point in order to show all eight curves concurrently. The specific implementation parameters are given in later plots.

From these pareto curves, of the three implementations settings (transistor Vt, decode implementation and sense amp implementation), the parameter that has the largest effect on cache power is the transistor Vt, with implementations using a dual-Vt process dissipating significantly less power than their single-Vt counterparts. This is followed by the use of differential sense amplification, which results in a slightly lower power dissipation than implementations using full-swing single-ended sense amplification. Lastly, the use of static and dynamic decoding is observed to have minimal effect on power dissipation.

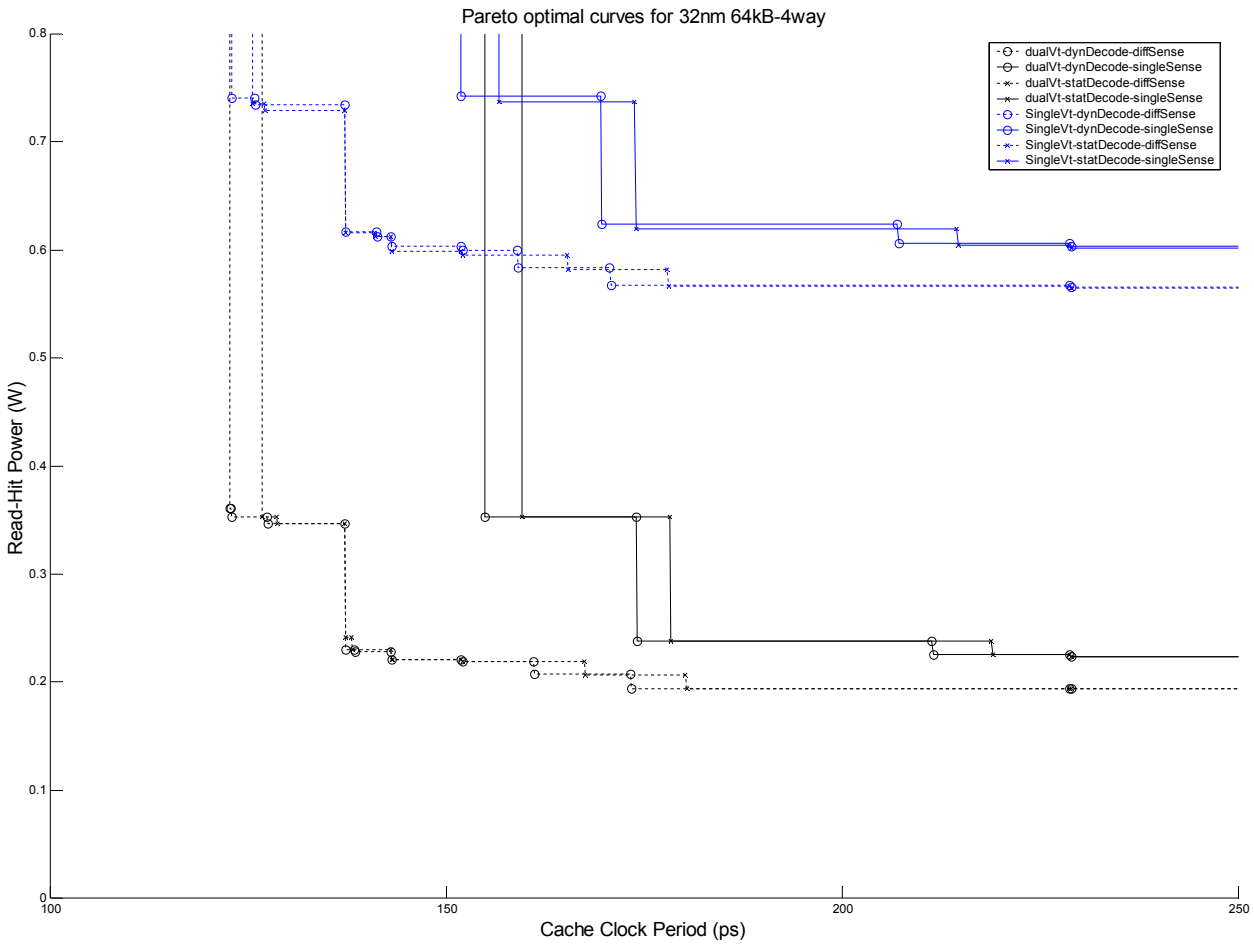


Figure 5-21: Pareto optimal curves for a 32nm 64kB 4-way cache. Eight pareto curves are shown for each possible permutation of the three parameters: Transistor threshold voltage, decode implementation and sense amplifier implementation.

We can also see from the pareto curves that, in terms of resulting cache clock period, the parameter that makes the biggest difference is the choice of sense amplification, with low-swing differential sensing potentially resulting in significantly less delays than full-swing single-ended sensing[1]. The choice of decoding implementation follows the choice of sensing with regards to the effect in the cache clock period, with the choice for transistor threshold voltage having the least difference among the three (although the effect on delay, in this case, is still significant and hardly negligible).

Figure 5-22 and Figure 5-23 simply repeat some of the data shown in the previous figure, where dual-Vt implementations are grouped into Figure 5-22 and single-Vt implementations are grouped into Figure 5-23

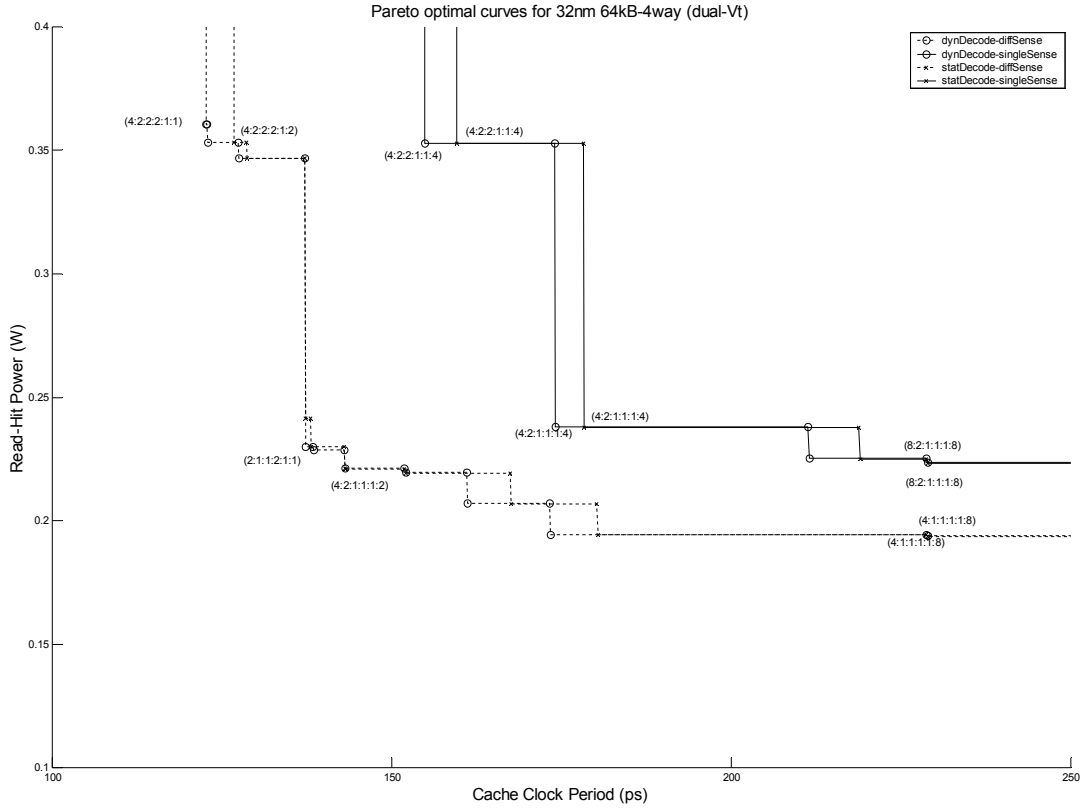


Figure 5-22: Pareto optimal curves for a dual-Vt 32nm 64kB 4-way cache. Four pareto curves are shown for each possible decode and sensing implementations. This plot simply repeats the main pareto plot showing the eight pareto curves, but focuses on the pareto curves for dual-Vt processes. Note that the y-axis is different, with limits from 0.1W to 0.4W.

1. Previous discussions regarding full-swing single-ended sensing still apply, where a significant part of the advantage of differential sensing may disappear once we perform design margining in order to account for noise events and PVT variations.

in order to show the points with more clarity and to allow room for labeling the three representative pareto optimal points that will be discussed in detail.

Finally, the different representative pareto optimal implementations are summarized in Table 5-2

Table 5-2: Representative Pareto Optimal Implementations for 32nm 64kB 4-way caches

Vt	Decode Implementation	Sense Amp Implementation	Minimum Delay (Maximum Power)	Medium Delay (Medium Power)	Maximum Delay (Minimum Power)
Dual	Dynamic	Differential	4:2:2-2:1:1	2:1:1-2:1:1	4:1:1-1:1:8
Dual	Dynamic	Single-ended	4:2:2-1:1:4	8:2:1-1:1:4	8:2:1-1:1:8
Dual	Static	Differential	4:2:2-2:1:2	4:2:1-1:1:2	4:1:1-1:1:8

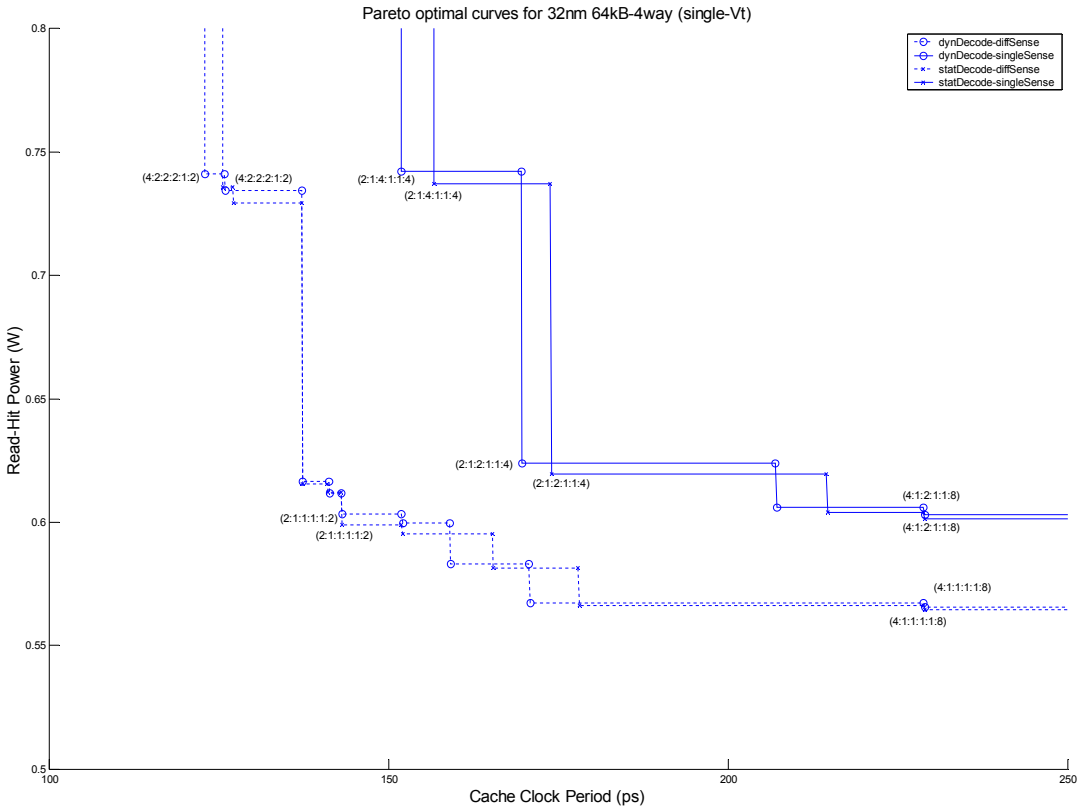


Figure 5-23: Pareto optimal curves for a single-Vt 32nm 64kB 4-way cache. Four pareto curves are shown for each possible decode and sensing implementations. This plot simply repeats the main pareto plot showing the eight pareto curves, but focuses on the pareto curves for single-Vt processes. Note that the y-axis is different, with limits from 0.5W to 0.8W.

Table 5-2: Representative Pareto Optimal Implementations for 32nm 64kB 4-way caches

Vt	Decode Implementation	Sense Amp Implementation	Minimum Delay (Maximum Power)	Medium Delay (Medium Power)	Maximum Delay (Minimum Power)
Dual	Static	Single-ended	4:2:2-1:1:4	8:2:1-1:1:4	8:2:1-1:1:8
Single	Dynamic	Differential	4:2:2-2:1:2	2:1:1-1:1:2	4:1:1-1:1:8
Single	Dynamic	Single-ended	2:1:4-1:1:4	2:1:2-1:1:4	4:1:2-1:1:8
Single	Static	Differential	4:2:2-2:1:2	2:1:1-1:1:2	4:1:1-1:1:8
Single	Static	Single-ended	2:1:4-1:1:4	2:1:2-1:1:4	4:1:2-1:1:8

5.4.2 Dual-Vt, Dynamic-Decode and Differential-Sense Implementation

Power breakdown. This implementation uses a dual-Vt process (with the low-Vt being designated for all the devices in the critical path except for the devices in the memory cell which are all HVT), dynamic decoding and low-swing differential sensing. The power numbers and analyses are exactly the same as the ones presented in the previous section for the 32nm 64kB 4-way cache, and they are repeated in Figure 5-24.

It is important to note that even with the use of HVT transistors, the leakage at 32nm is significant enough to account for roughly half of the power dissipation for the MaxPower-MinDelay process. Surprisingly, most of this leakage power is not due to the bitlines, but instead are dissipated in the data output

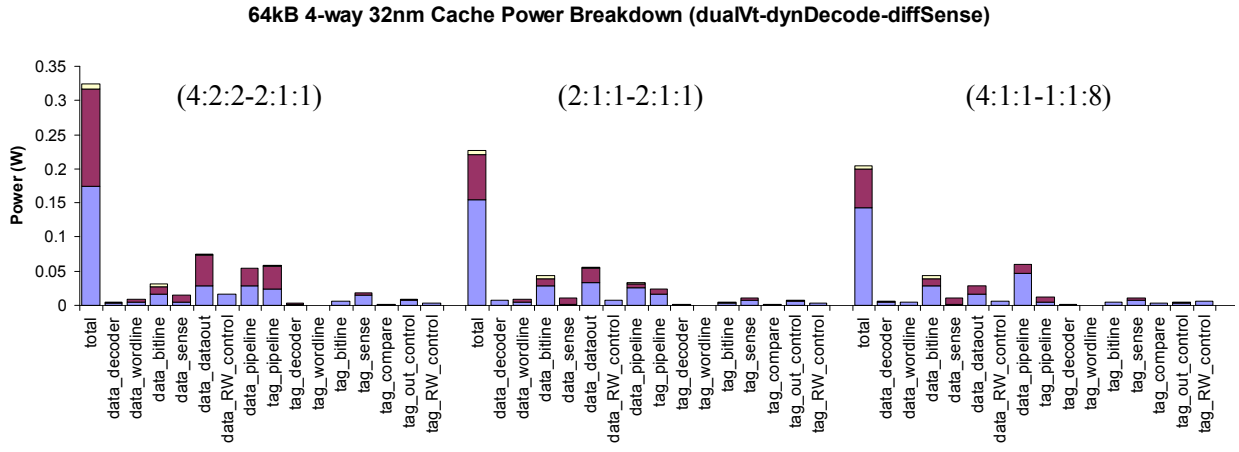


Figure 5-24: Detailed power breakdown for a 64kB 4-way cache for the 32nm technology node.

driver circuits and the pipeline overhead circuits. This makes sense, as these circuits both use LVT transistors. This again shows the potential use for the detailed breakdowns provided by myCACTI. With the availability of delay and power breakdowns, we can look for potential implementations that have significant subthreshold leakage power being dissipated by circuits that are not in the critical path. If some circuits match this requirement, we can potentially convert any LVT transistors in that circuit to HVT, resulting in a decrease in subthreshold leakage power. Even though the delay of that particular block may be degraded, the fact that it is not the critical path for that particular implementation means that th cache clock period stays the same. We essentially save on power without sacrificing anything. One example in this case is the MidDelay-MidPower implementation, where we can see that the data output driver stage dissipates significant subthreshold leakage power and it is not part of the critical path (although it is very close to being one). Substituting HVT transistors for LVT transistors in the data output driver will result in some power savings, while degrading the data output driver delay. In this particular case where the delay of the data output driver stage is close to the critical path delay, further degradation will most probably result in degrading the critical path delay, resulting in degradation of the cache clock period in order to save on power. Although not a clear design win, this is a valid tradeoff that the designer can use.

We will attempt to identify better examples later on of using the power and delay breakdowns provided by myCACTI to identify clear design wins that produce delay or power savings without degrading anything.

Delay breakdown. Like the power breakdown, the delay breakdown is exactly the same as the 32nm 64kB 4-way configuration studied in the previous subsection, showing the different implementations having different stages as their critical paths. These numbers are repeated in Figure 5-25 and Figure 5-26.

5.4.3 Dual-Vt, Dynamic-Decode, Single-ended-Sense Implementation

Power breakdown. Figure 5-27 shows the power breakdown for this implementation. This implementation uses a dual-Vt process, dynamic decoding, and full-swing single-ended sense amplification. With the use of full-swing single-ended sensing, we can see that the dynamic power dissipated in the bitlines is increased significantly for all three implementations to the point that bitline dynamic power is now the biggest contributor to the total dynamic power. At this technology node, though, the increase in bitline dynamic power is typically small compared to the subthreshold leakage power dissipated in the entire cache such that the net fractional difference in using full-swing single-ended sense amplification is not as big as one would expect after requiring the bitline to be completely discharged instead of discharging it enough to develop a relatively small voltage differential (which is what is done for low-swing differential sensing).

For this implementation, since the only change compared to the default myCACTI setting is the sense-amp implementation, the subthreshold leakage power for all components and all other components are left unchanged.

Delay breakdown. The delay numbers and the breakdown for each stage are shown in Figure 5-28 and Figure 5-29. With the use of full-swing single-ended sensing and the need to completely discharge the bitlines,

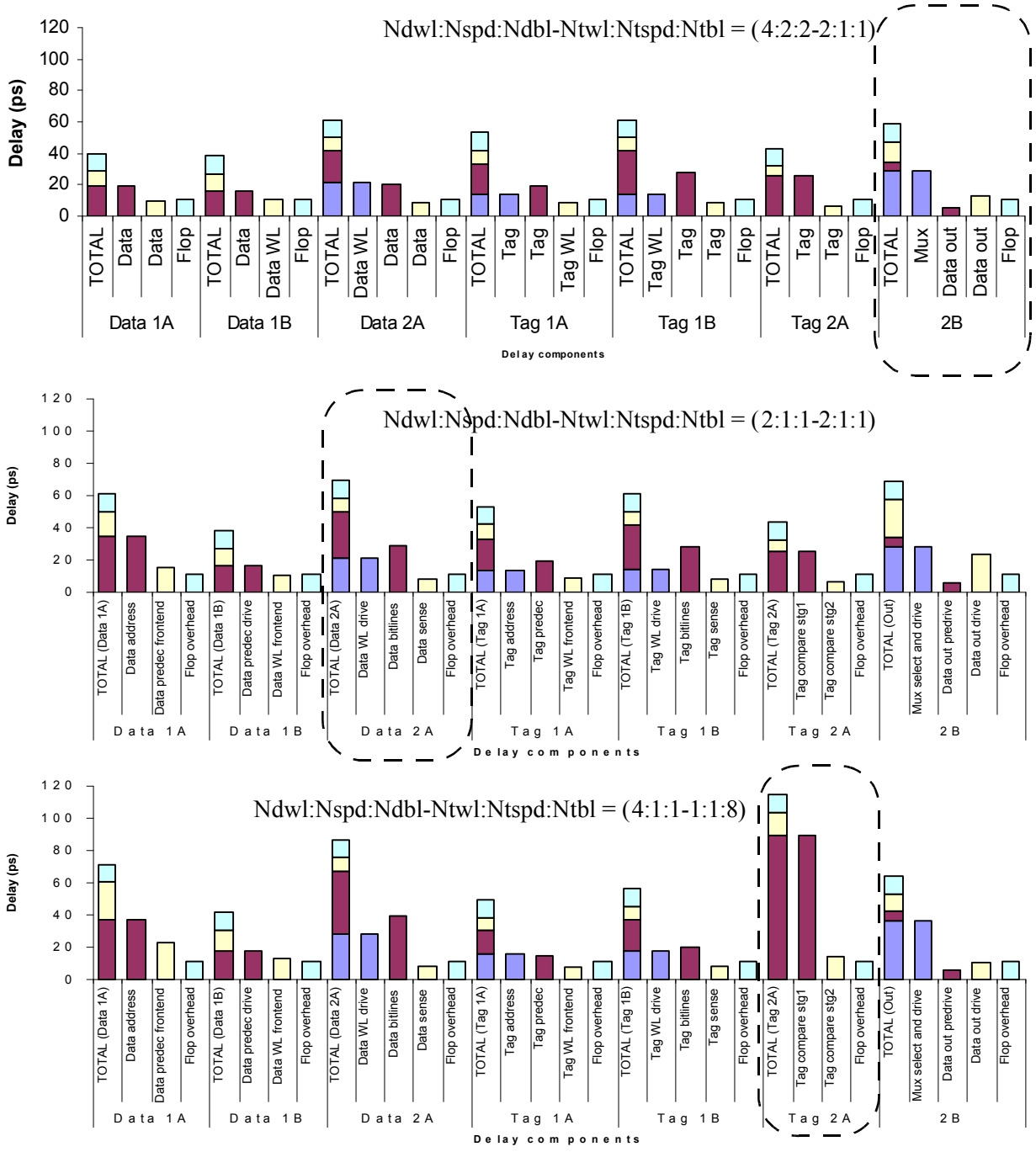


Figure 5-25: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 32nm cache. These plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the dashed .rectangles denote the critical path delay of the cache that sets its clock period.

the bitline delay has increased significantly, and only the suboptimal sizing of the comparator stage results in the bitline stage being the critical path for only two out of the three implementations studied. Optimizing the comparator through further custom design will be significantly easier than optimizing the bitline delay, as the consequences will be less widely spread for any tag comparator changes compared to bitline changes.

The MinDelay-MaxPower implementation is now a good example of the optimization that was discuss ificant

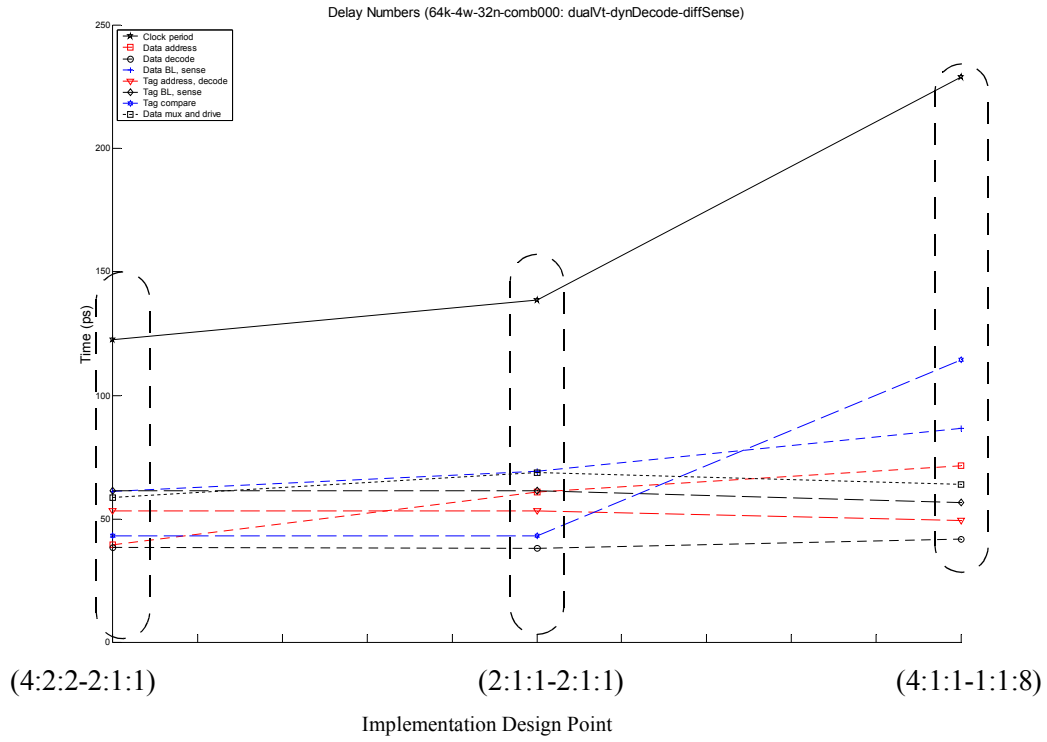


Figure 5-26: Delay numbers for three pareto optimal implementations of a 64kB 4-way 32nm cache with dual-Vt transistors, dynamic decoding and differential sensing. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power) the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the pareto curve.

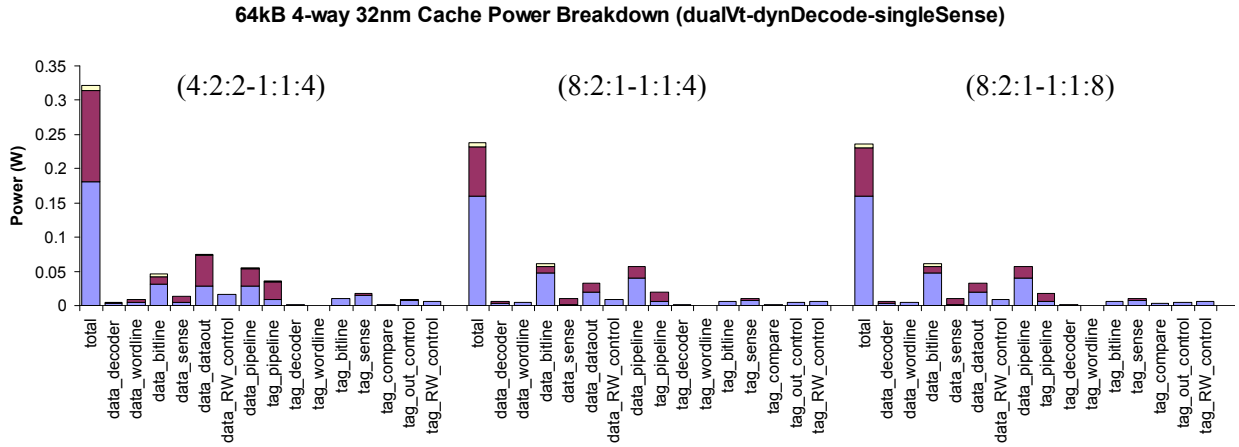


Figure 5-27: Detailed power breakdown for a 64kB 4-way cache for the 32nm technology node.

subthreshold leakage power is dissipated by the data output driver stage, and this stage has significant slack with respect to the cache clock period set by the data wordline driver-bitline-sense amp stage. Changing the output driver transistors from LVT to HVT will enable a significant reduction of the subthreshold leakage power without degrading the cache clock period since the degradation in the data output driver delay is essentially inconsequential as there is enough slack to absorb the delay increase. By doing this, this MinDelay-MaxPower optimal implementation will have a significantly smaller power while maintaining its minimum delay status, resulting a much more optimal implementation.

Looking at the delay breakdowns in Figure xx, we can readily see the significant portion of the delay attributed by the bitlines (both the data and the tag bitlines). These two delays, along with the tag compare delay, are the three largest delay from any cache component. But as earlier mentioned, it would be significantly easier to improve the tag comparator delay compared to the bitline delay as the tradeoffs involved are much easier for the tag comparator as the effects are largely localized and limited to a significantly smaller area or fraction of the cache compared to the bitlines.

5.4.4 Dual-Vt, Static-Decode, Differential-Sense Implementation

Power breakdown. The power breakdown for this implementation is shown in Figure 5-30. This implementation uses a dual-Vt process, static decoding and low-swing differential sense amplification. The

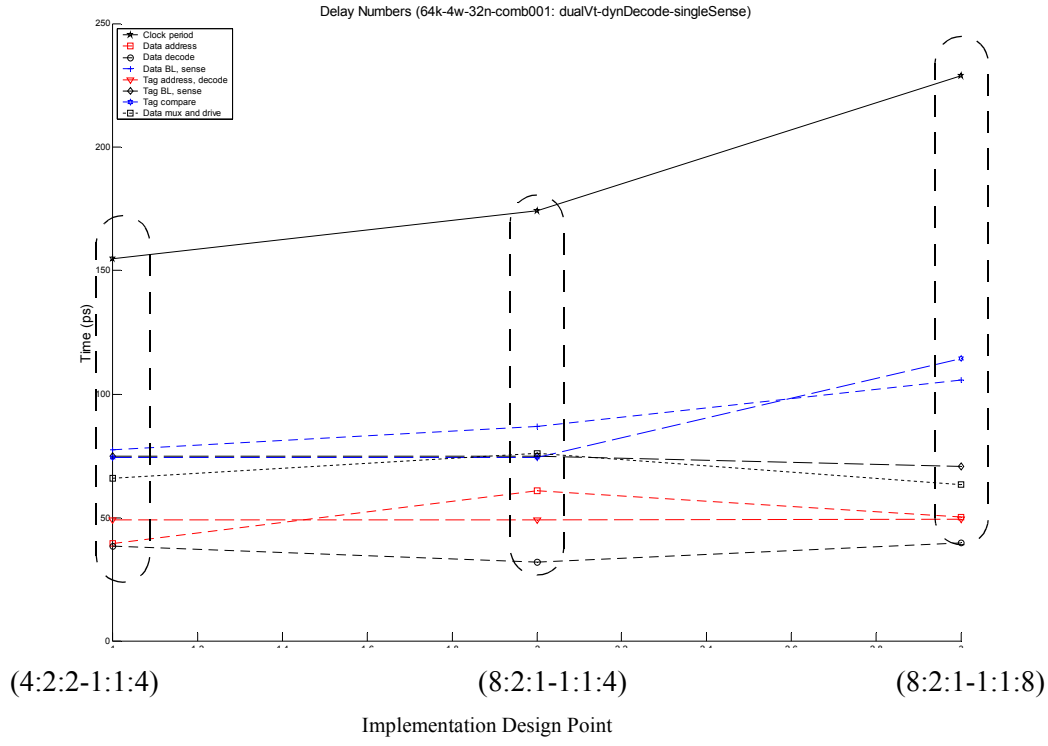


Figure 5-28: Delay numbers for three pareto optimal implementations of a 64kB 4-way 32nm cache with dual-Vt transistors, dynamic decoding and single-ended sensing. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power), the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the paretocurve.

behavior of this implementation compared to the default myCACTI implementation is largely the same, as the power dissipated by the static decoder is largely the same as the dynamic decoder.

Delay breakdown. The delay numbers and detailed breakdown are shown in Figure 5-31 and Figure 5-32. Compared to power, the use of static decoding typically has a greater effect on cache clock period as long as the critical path delay is a stage that contains part of the decoder. In this case, both the MinDelay-MaxPower

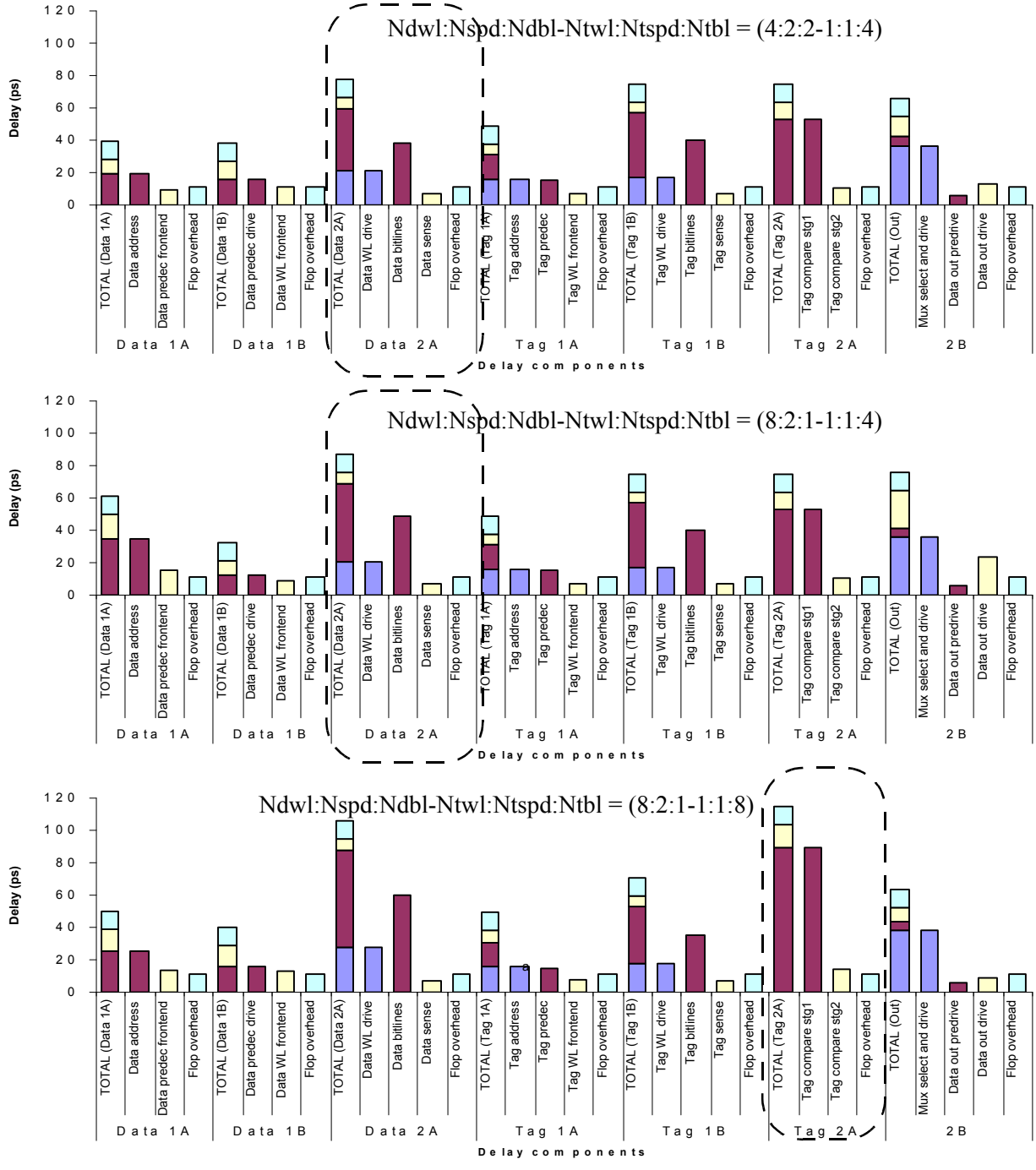


Figure 5-29: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 32nm cache. These plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the dashed .rectangles denote the critical path delay of the cache that sets its clock period.

and MaxDelay-MinPower both have the data wordline-bitline-sense amp stage as the critical paths, where the wordline driver is implemented as static drivers instead of skewed dynamic drivers, resulting in a larger delay. Even though the delay for the decoder in the MidDelay-MidPower stage may have increased, since the critical stage is that data output driver stage, the cache clock period stays the same as the one for the default myCACTI implementation.

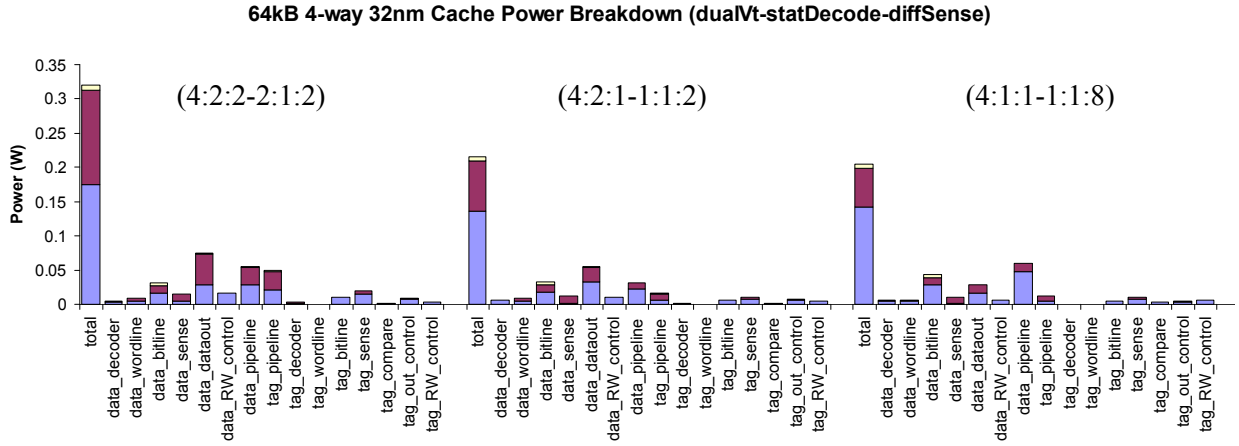


Figure 5-30: Detailed power breakdown for a 64kB 4-way cache for the 32nm technology node.

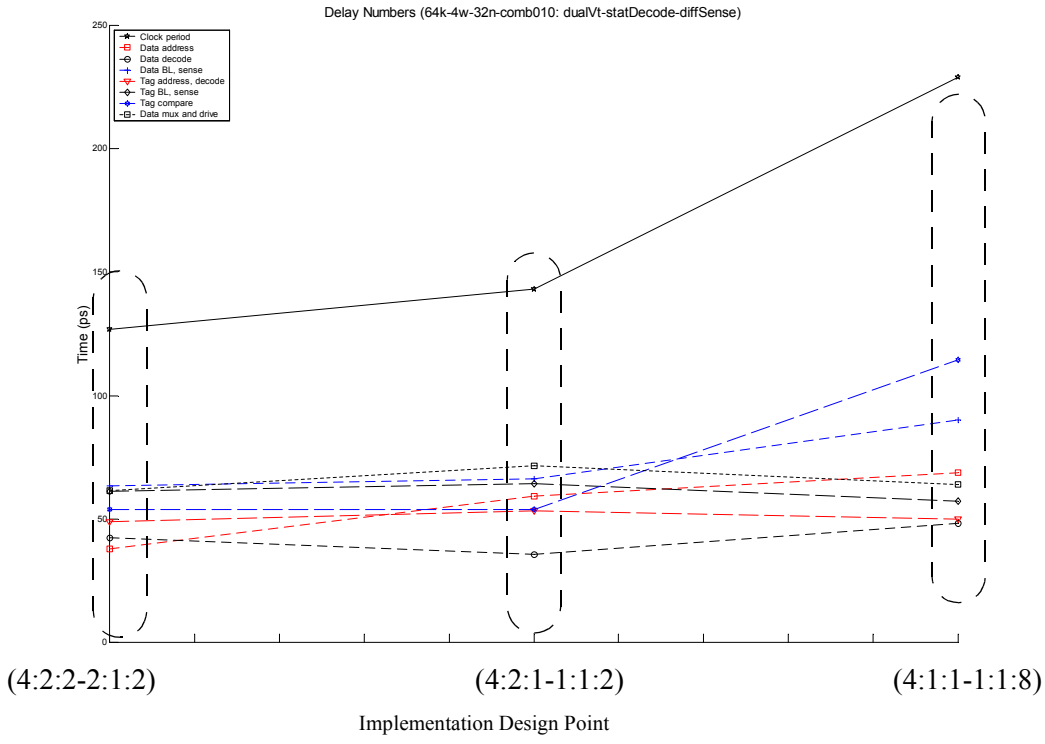


Figure 5-31: Delay numbers for three pareto optimal implementations of a 64kB 4-way 32nm cache with dual-Vt transistors, static decoding and differential sensing. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power) the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the pareto curve.

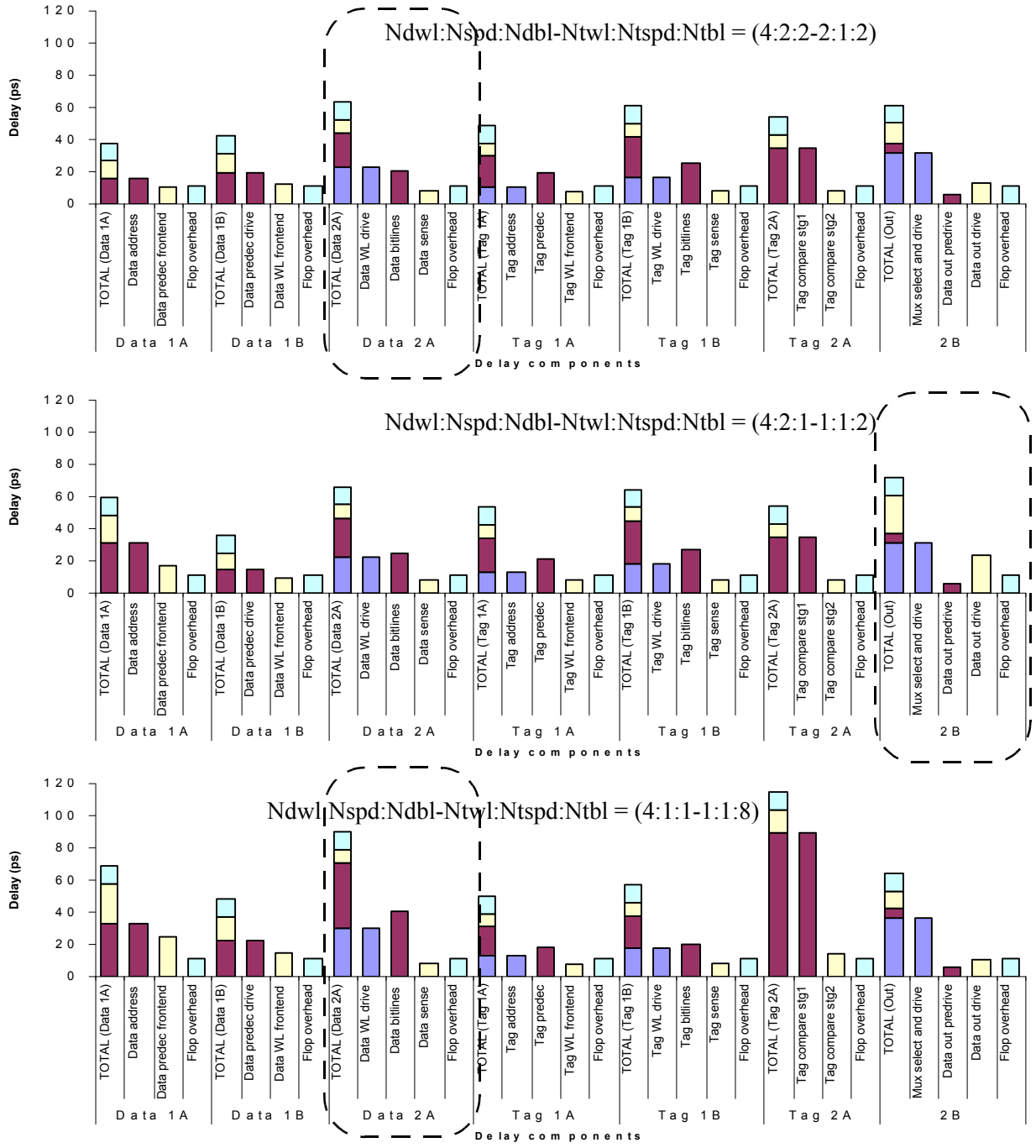


Figure 5-32: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 32nm cache. Th plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the .rectangles denote the critical path delay of the cache that sets its clock period.

5.4.5 Dual-Vt, Static-Decode, Single-ended-Sense Implementation

Power breakdown. The power breakdown for this implementation is shown in Figure 5-33. This implementation uses a dual-Vt process and as opposed to the myCACTI defaults, static decoding and full-swing single-ended sensing. Similar to before, we can see that the amount of dynamic power dissipated in the bitline is significantly increased compared to the myCACTI default. Also as before, the power in the decoder is roughly the same as for the myCACTI default, again showing that the power dissipated by static CMOS decoders compared to the dynamic decoders that we model are largely the same.

Delay breakdown. The delay numbers and detailed breakdown are shown in Figure 5-34 and Figure 5-35. Compared to the myCACTI default setting, this implementation has the worst delay, as it implements the worst possible choice of the three settings with respect to delay. Specifically, single-Vt is faster than dual-Vt, dynamic decoding is faster than static decoding, and low-swing differential sensing is faster than full-swing single-ended sensing.

The data wordline driver-bitline-sense stage again constitutes two of the three critical path delays of the circuit, and has only a small slack in the third one such that it is almost the critical path also. Looking at the delay breakdowns, we can see that the bitline delays are again very substantial because of the need for the weak memory cell driver and access transistor to completely discharge the bitlines. In addition, the wordline driver delay part of the stage is also degraded by using static decoding.

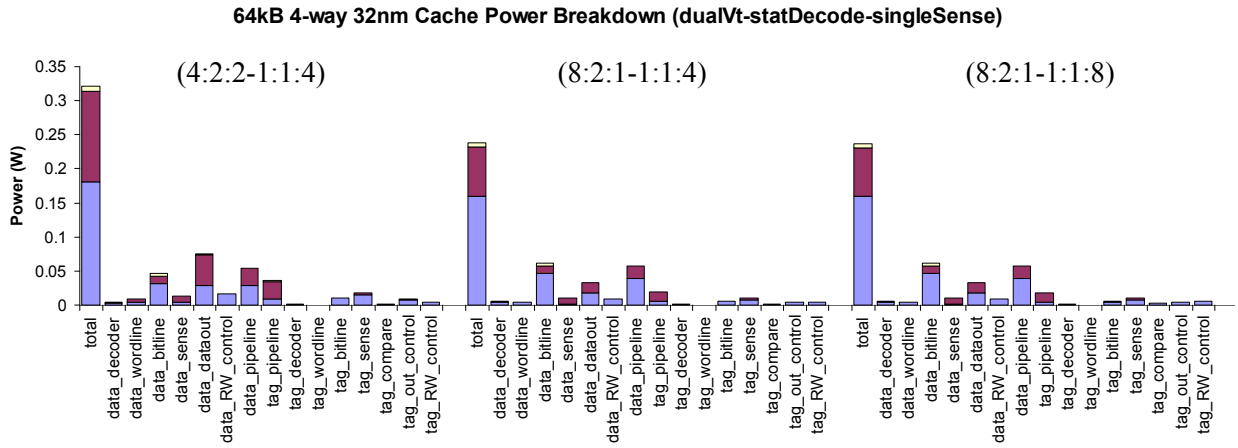


Figure 5-33: Detailed power breakdown for a 64kB 4-way cache for the 32nm technology node.

5.4.6 Single-Vt, Dynamic-Decode, Differential-Sense Implementation

Power breakdown. The detailed power breakdown for this implementation is shown in Figure 5-36. This implementation uses a single-Vt process as opposed to the dual-Vt process used by the myCACTI default. The two other parameters are the same, with the use of dynamic decoding and low-swing differential sensing.

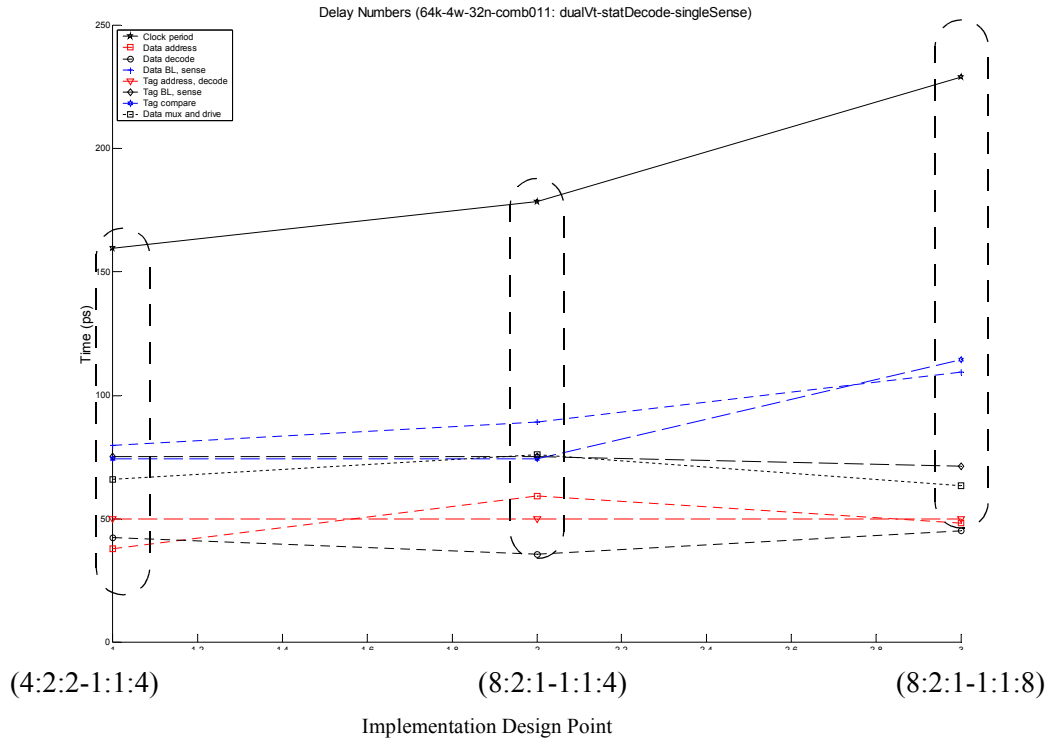


Figure 5-34: Delay numbers for three pareto optimal implementations of a 64kB 4-way 32nm cache with dual-Vt transistors, static decoding and single-ended sensing. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power) the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the pareto curve.

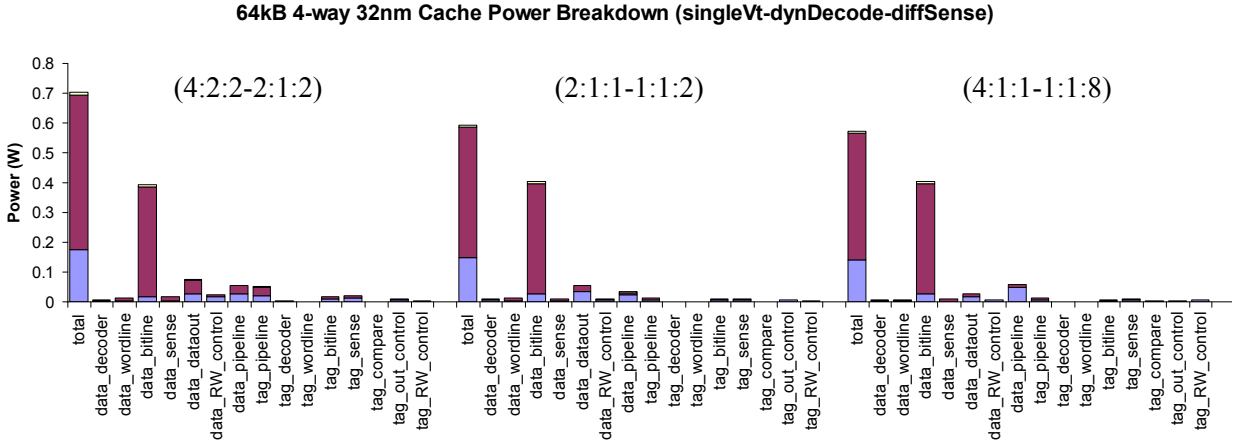


Figure 5-36: Detailed power breakdown for a 64kB 4-way cache for the 32nm technology node.

The biggest change that we can observe is that use of pure single-Vt transistors (assumed to be LVT) drastically increases the total power dissipation of the implementations, with most of the increase due to the increase in the subthreshold leakage power dissipated in the bitlines. It is worthwhie to note that percentage-wise, the power variation of the three implementations are much less compared to the myCACTI default. This is also due to the increased subthreshold leakage power dissipated in the bitlines, as this is something that is

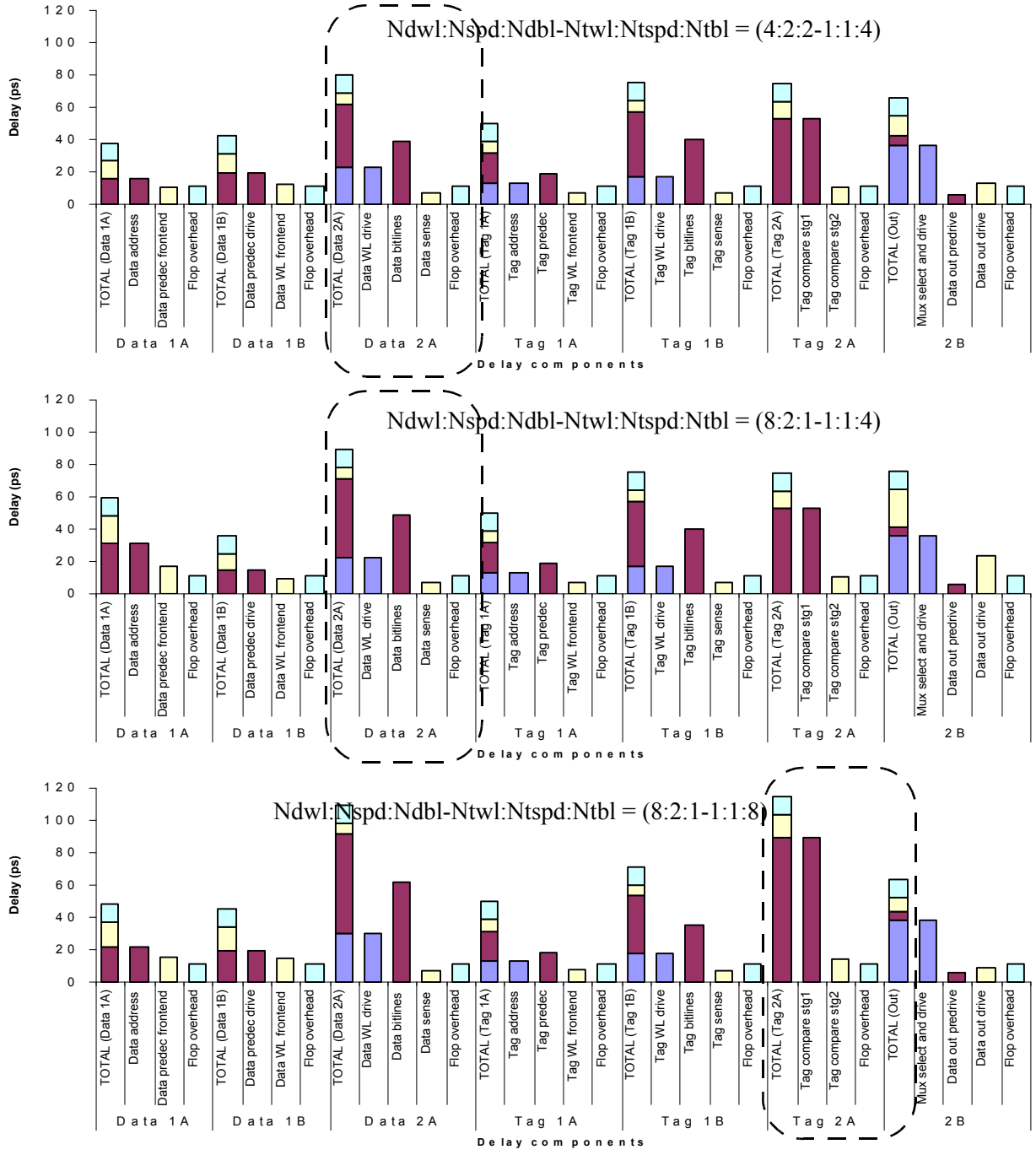


Figure 5-35: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 32nm cache. Th plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the .rectangles denote the critical path delay of the cache that sets its clock period.

determined only by the cache size, and is not affected by varying the cache implementation parameters that the CACTI, eCACTI and myCACTI tools vary. This can be seen from the power breakdown, where we can see visually that the subthreshold leakage component of the bitline power is the same for the MinDelay-MaxPower, MidDelay-MidPower and MaxDelay-MinPower implementations.

Delay breakdown. The delay numbers and detailed breakdown are shown in Figure 5-37 and Figure 5-38. With the use of single-Vt transistors, the bitline delay is sped up to the point where none of the three stages have it as the critical stage. Instead, two of the implementations now have the data output stage as the critical path, with the other one being the tag compare stage.

One interesting observation is that this particular implementation using single-Vt transistors, dynamic decoding and full-swing differential sense amplifiers should result in the fastest configurations, this is not true for the three representative pareto optimal implementations that we show here in detail. This is explained by the fact that since the bitlines are not included in the critical path for all three representative implementations, the cache clock period is largely the same as the dual-Vt implementation. This is important, as it demonstrates that a dual-Vt process can yield essentially the same clock speed as a single-Vt process, but with a much smaller power dissipation.

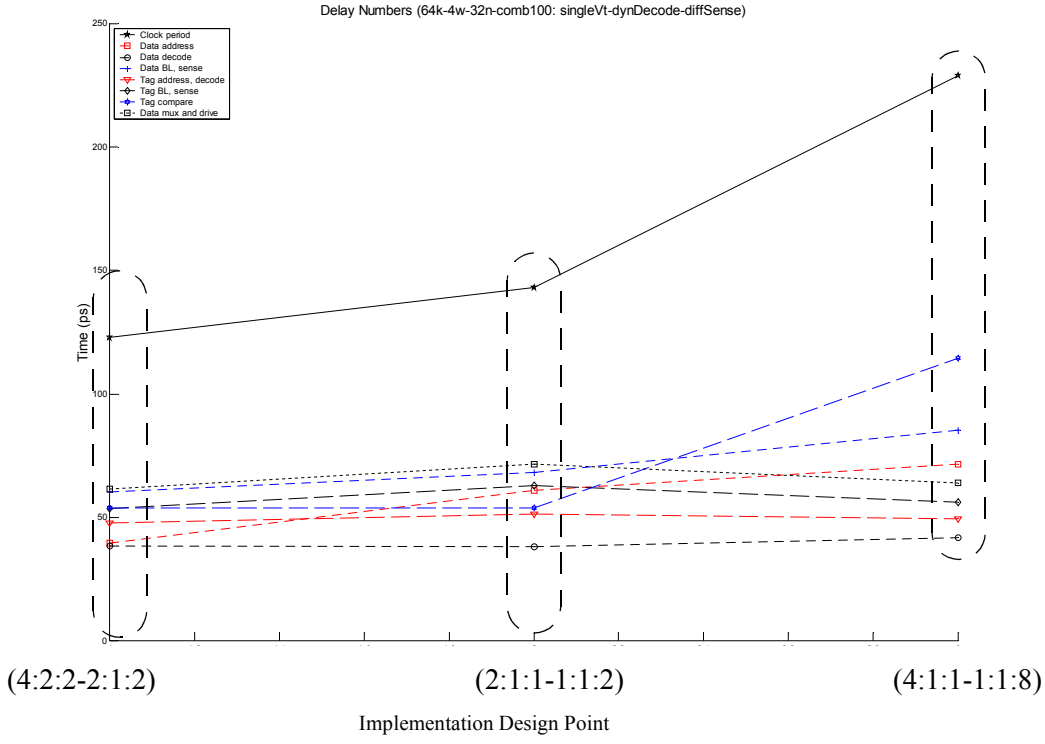


Figure 5-37: Delay numbers for three pareto optimal implementations of a 64kB 4-way 32nm cache with single-Vt transistors, dynamic decoding and differential sensing. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power) the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the pareto curve.

5.4.7 Single-Vt, Dynamic-Decode, Single-ended-Sense Implementation

Power breakdown. The detailed power breakdown for this implementation is shown in Figure 5-39. This implementation uses a single-Vt process as well as full-swing single-ended sensing as opposed to a dual-Vt process and low-swing differential sensing used by the myCACTI defaults. The other parameter is the same, with the use of dynamic decoding.

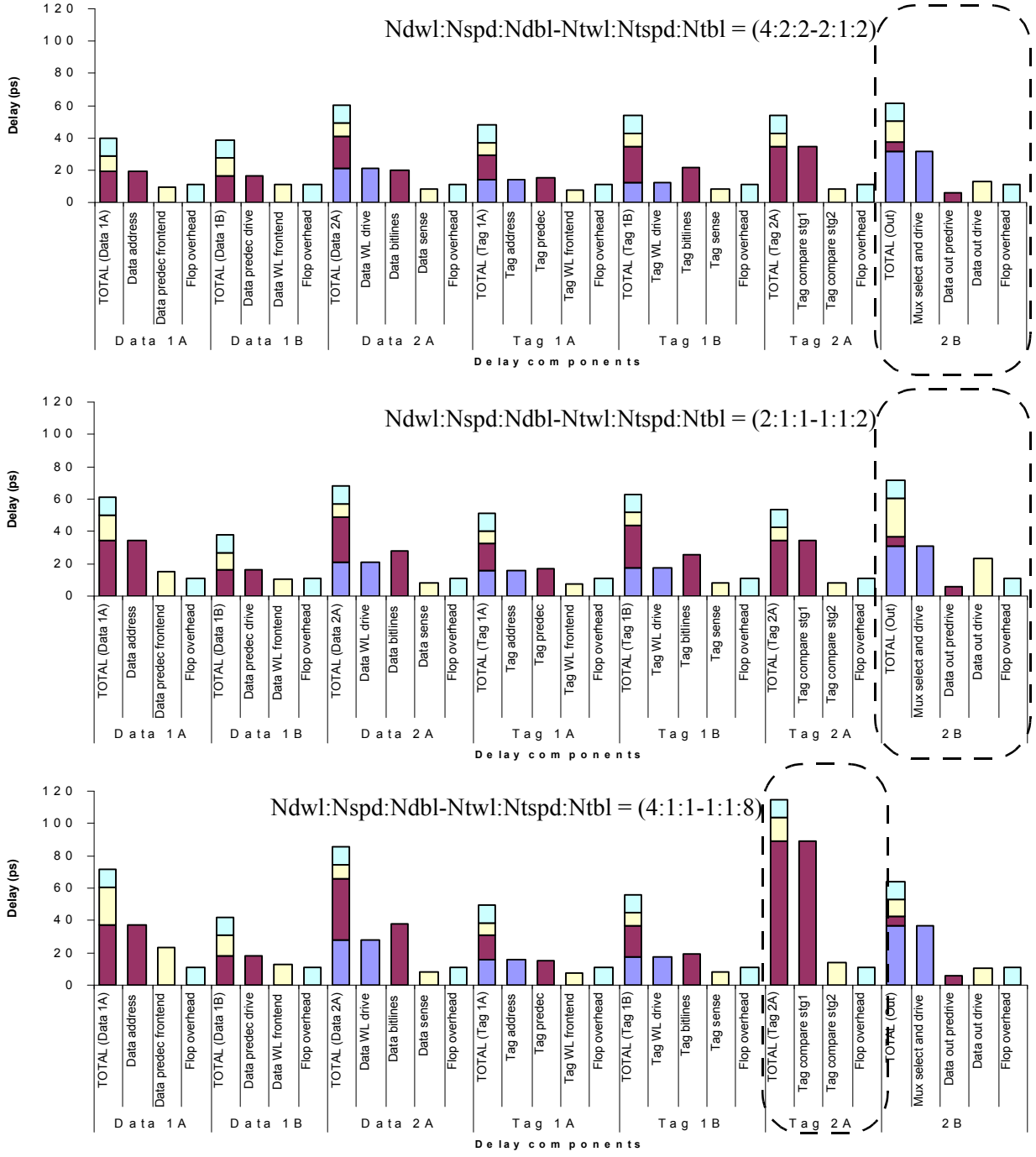


Figure 5-38: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 32nm cache. Th plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the .rectangles denote the critical path delay of the cache that sets its clock period.

We observe that even though the use of full-swing single-ended sensing has increased the dynamic power being dissipated in the bitlines, this dynamic power increase is still significantly less compared to the increase in bitline subthreshold leakage power due to the use of single-Vt transistors in the entire cache.

Delay breakdown. The delay numbers and detailed breakdown for each stage are shown in Figure 5-40 and Figure 5-41. This time, with the use of single-ended sensing, two out of the three representative pareto

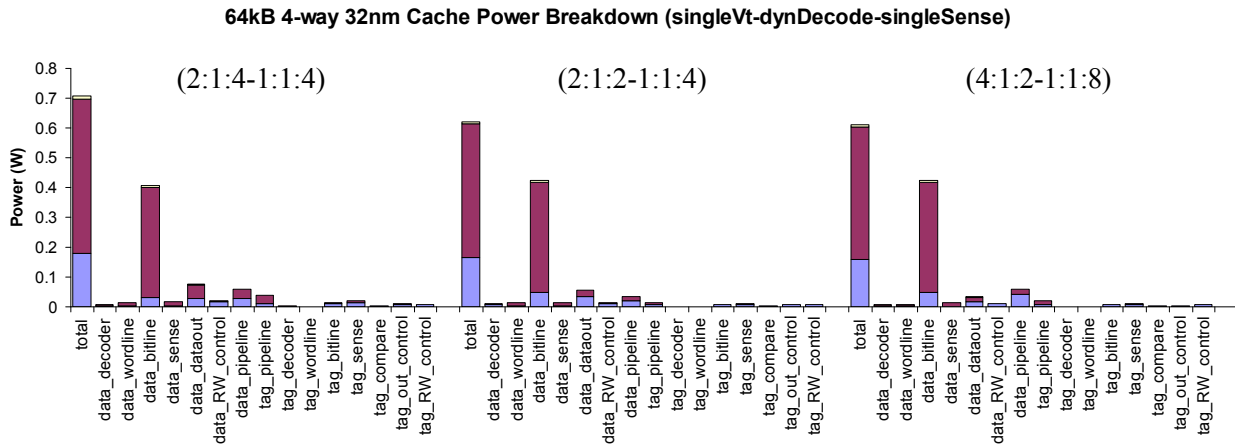


Figure 5-39: Detailed power breakdown for a 64kB 4-way cache for the 32nm technology node.

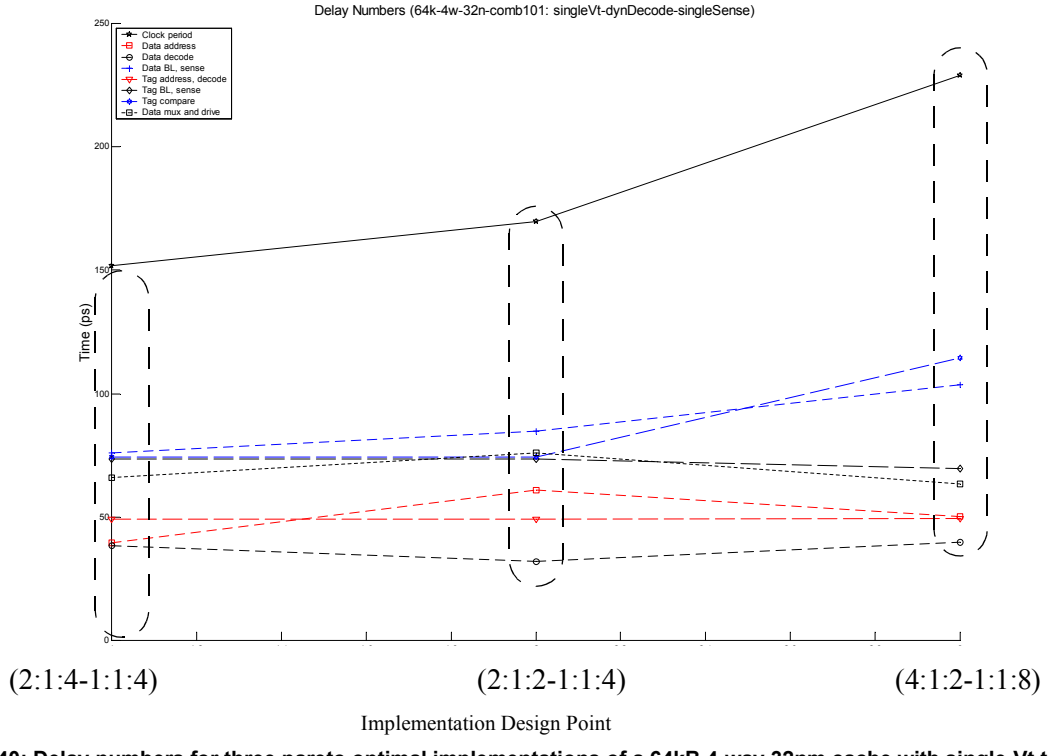


Figure 5-40: Delay numbers for three pareto optimal implementations of a 64kB 4-way 32nm cache with single-Vt transistors, dynamic decoding and single-ended sensing. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power), the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the paretocurve.

optimal implementations have the wordline driver-bitline-sense amp stage (Data_2A) as their critical stage, with the other one being the tag compare stage. Although use of single-ended sensing has increased the delay compared to low-swing differential sensing, this is offset by the better driving capability of the LVT memory cell driver and access transistor, such that it is easier to discharge the bitline completely compared to an HVT memory cell.

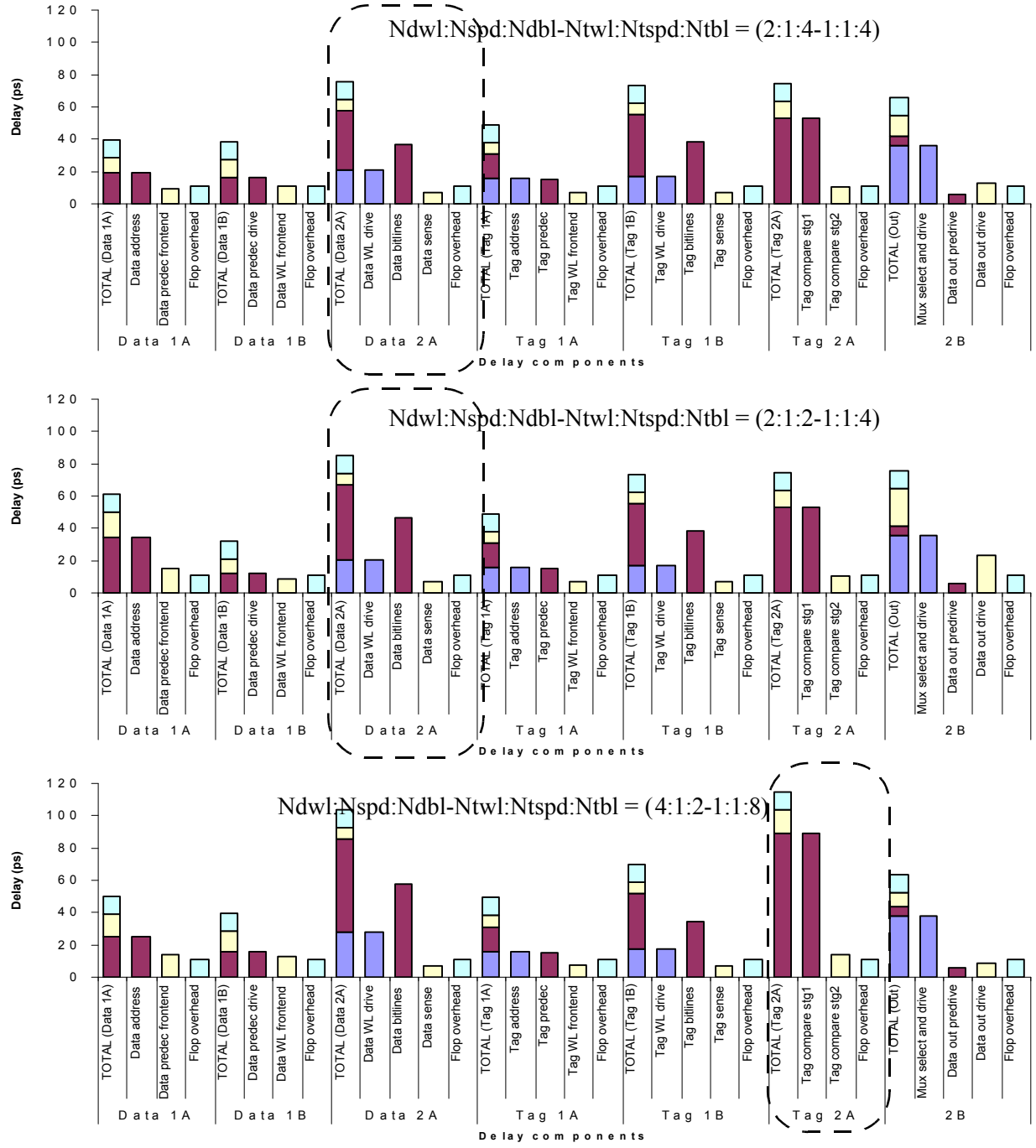


Figure 5-41: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 32nm cache. Th plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the .rectangles denote the critical path delay of the cache that sets its clock period.

5.4.8 Single-Vt, Static-Decode, Differential-Sense Implementation

Power breakdown. The detailed power breakdown is shown in Figure 5-42. This implementation uses a single-Vt process as well as static decoding as opposed to a dual-Vt process and dynamic decoding used by the myCACTI default configuration. The other parameter is the same, with the use of low-swing differential sensing.

Again, similar to the last implementation, the increase in subthreshold leakage power dissipated by the bitlines completely dominates any other increase in power (whether dynamic, subthreshold or gate leakage) in any other cache component. Since the only other place where the power can potentially change is the decoder, and since we know that the power dissipation in the decoder is roughly the same for both static and dynamic decoding, the differences are even much less such that virtually the entire difference in power is accounted for by the difference in the subthreshold leakage power dissipated in the bitlines.

Delay breakdown. The delay numbers and detailed breakdown for each stage are given in Figure 5-43 and Figure 5-44. With this implementation, we expect the use of single-Vt transistor to slightly speed up the bitlines, but we also expect some degradation in the decoder (with the greatest effect in the wordline driver). From the delay breakdowns and and the main pareto plot shown in the early part of the subsection, we see that a lot of times, the final cache clock period will remain roughly the same, as there exists a significant potential that the critical path does not contain the bitlines or any other stage that contains part of the decoder. In this particular case, only one of the three representative pareto optimal configuration contained the bitline and/or part of the decoder (in this csae, the wordline driver-bitline-sense amp stage or stage Data_2A).

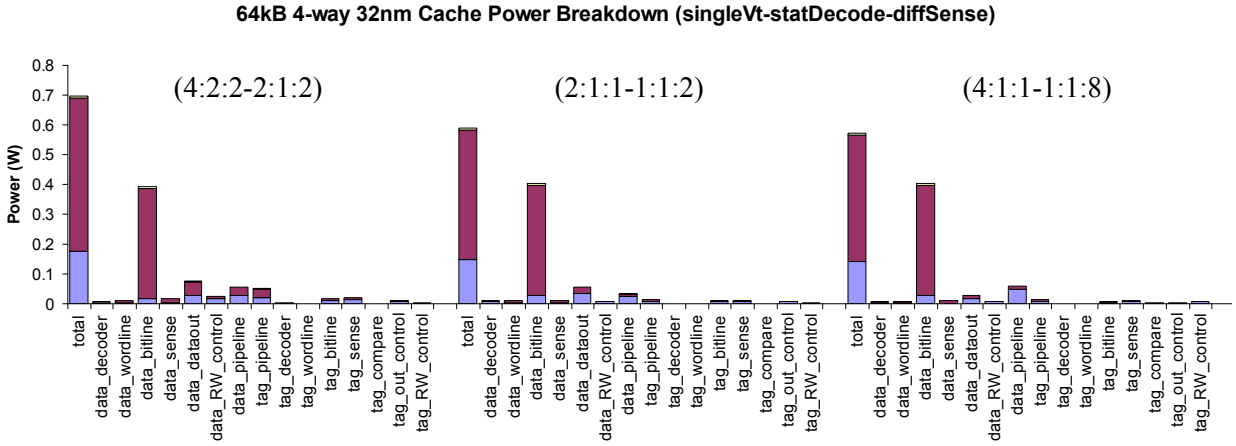


Figure 5-42: Detailed power breakdown for a 64kB 4-way cache for the 32nm technology node.

5.4.9 Single-Vt, Static-Decode, Single-ended-Sense Implementation

Power breakdown. The detailed power breakdown for this implementation is shown in Figure 5-45. This implementation uses a single-Vt process, as well as static decoding and full-swing single-ended sensing as

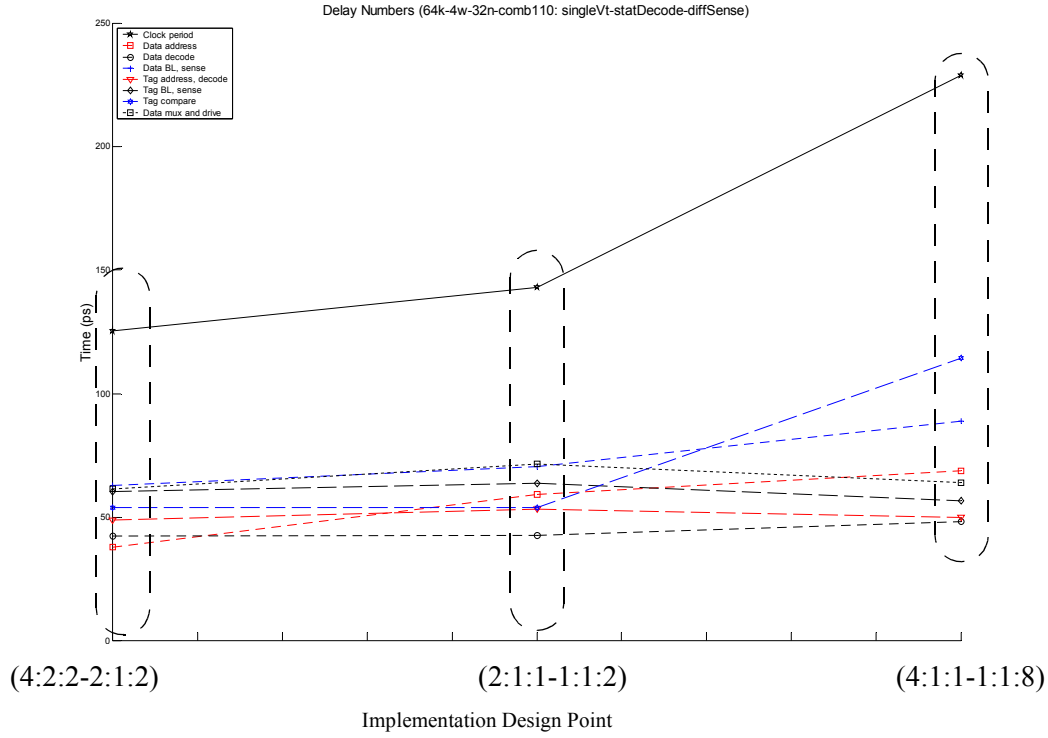


Figure 5-43: Delay numbers for three pareto optimal implementations of a 64kB 4-way 32nm cache with single-Vt transistors, static decoding and differential sensing. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power) the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the pareto curve.

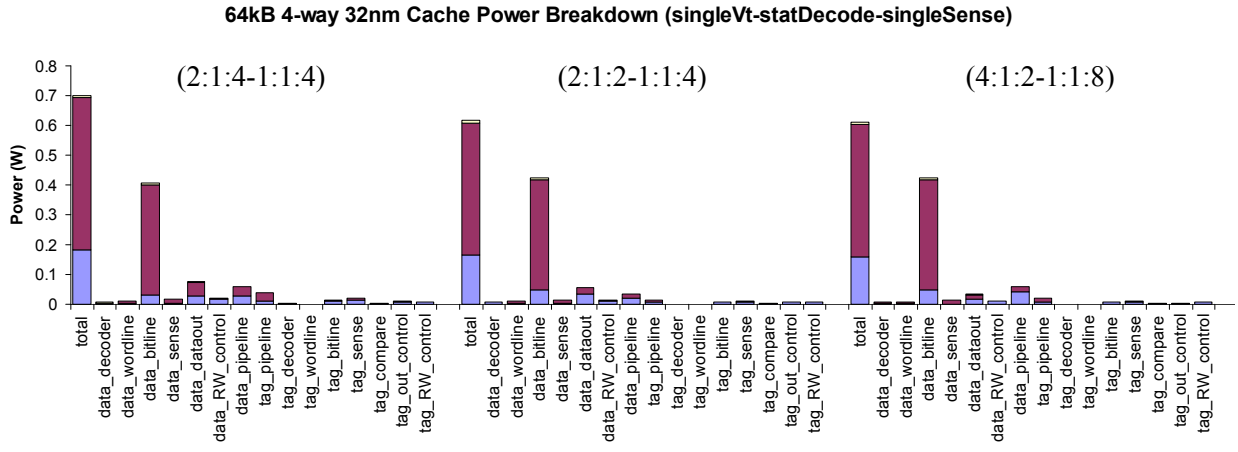


Figure 5-45: Detailed power breakdown for a 64kB 4-way cache for the 32nm technology node.

opposed to a dual-Vt process, dynamic decoding and low-swing differential sensing used by the myCACTI default configuration.

We see that this implementation results in significant power, and most of this is dissipated in the bitlines as the use of single-Vt transistor results in significant bitline subthreshold leakage power dissipation while the use of full-swing single-ended sensing results in an increase in dynamic power due to the need to

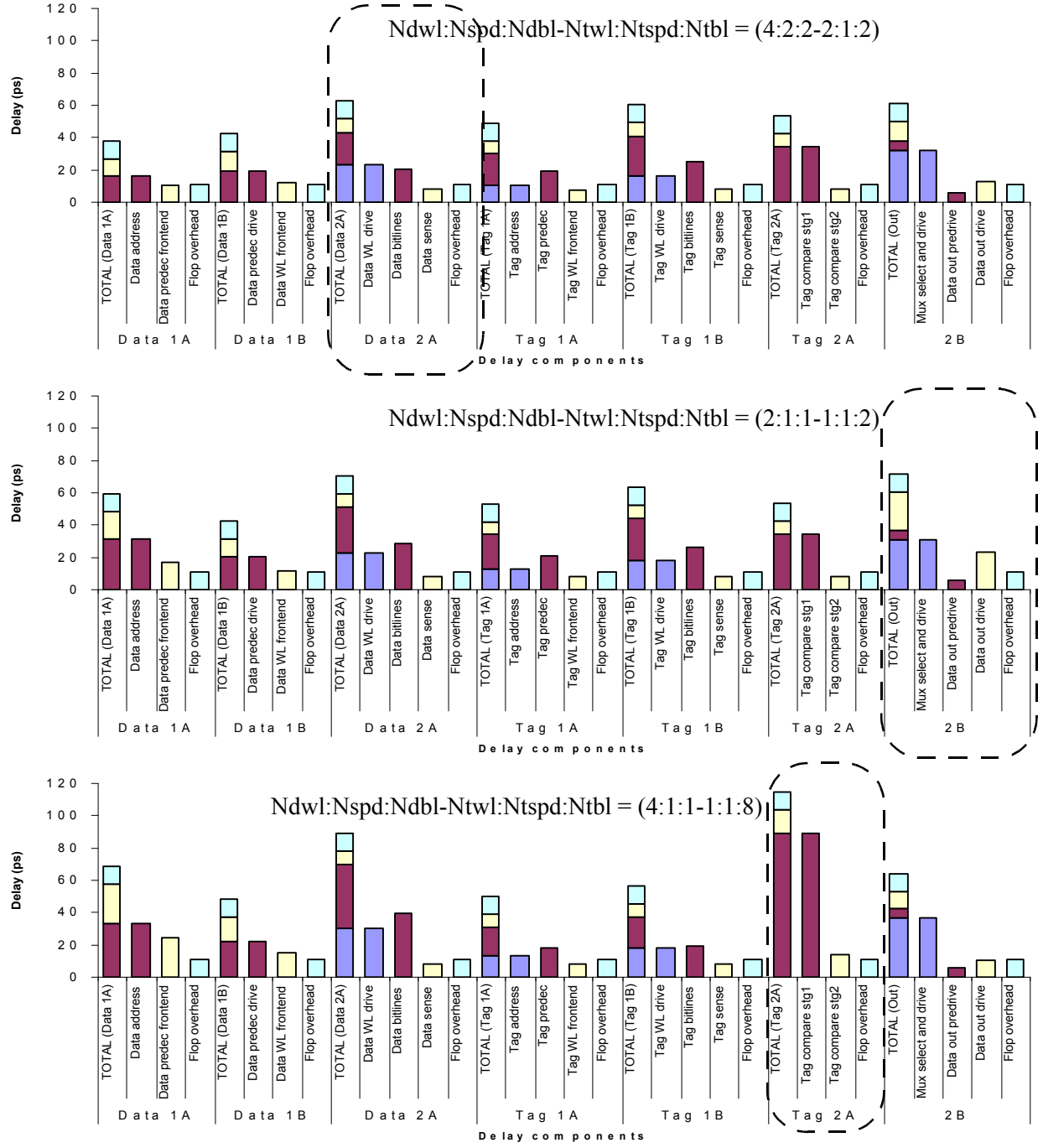


Figure 5-44: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 32nm cache. Th plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the .rectangles denote the critical path delay of the cache that sets its clock period.

fully discharge the bitlines to ground. It can easily be seen, though, that the second effect is much less compared to the subthreshold leakage power in the bitlines.

Delay breakdown. The delay numbers and their detailed breakdown are shown in Figure 5-46 and Figure 5-47. This implementation results in a significant delay in the bitline stage, as static decoding results in a slower wordline while full-swing single-ended sensing results in a slow bitline due to the need to fully discharge it. Although this is offset somewhat by the strong drive capability of the memory cell due to the single-Vt process (and hence, stronger memory cell driver and access transistors), the delay in the stage containing the bitlines is still substantial. In this case, two of the three representative pareto optimal implementations have this stage (stage Data_2A) as the critical stage, with the data bitline delay being a significant component of the total stage delay.

5.4.10 Summary

In this subsection, we have used myCACTI to perform detailed design space explorations on different permutations of dual-Vt/single-Vt fabrication process technology, static or dynamic decoding, and low-swing differential or full-swing single ended sense amplification for a 32nm 64kB 4-way cache. - Pareto optimal curves were shown for each of the eight possible permutations of the three cache design parameters. We have demonstrated that in terms of power, the parameter that has the most effect is the use of a single-Vt or dual-Vt

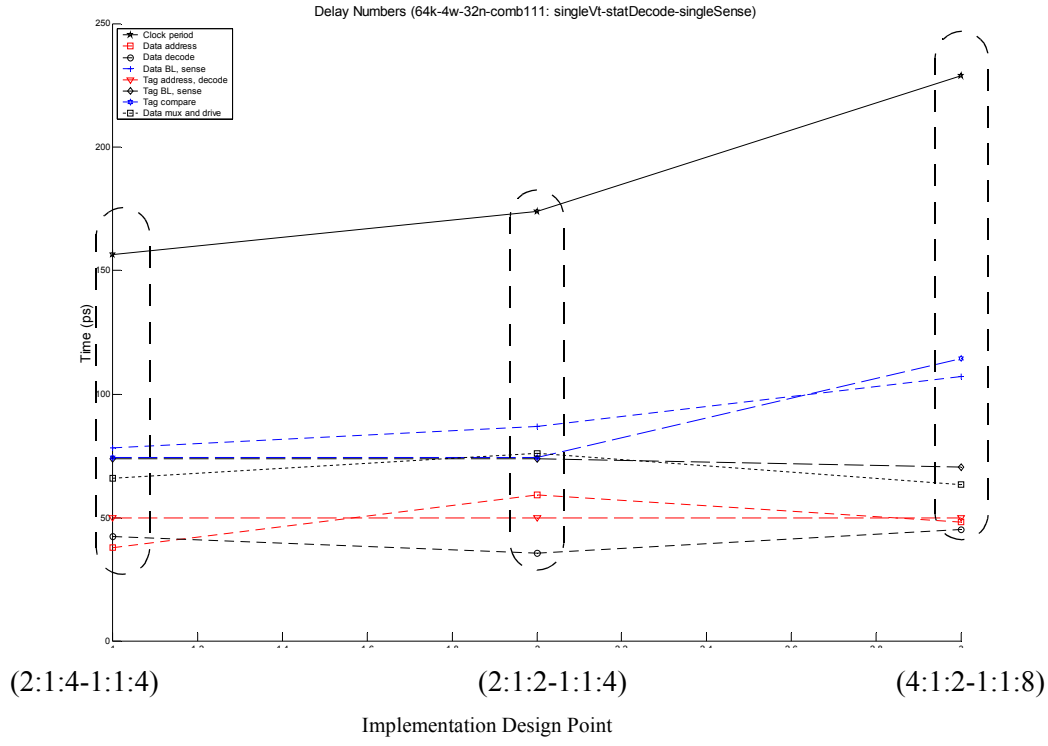


Figure 5-46: Delay numbers for three pareto optimal implementations of a 64kB 4-way 32nm cache with single-Vt transistors, static decoding and single-ended sensing. Each implementation is a pareto optimal design point signified by the cache implementation parameter sextet Ndwl:Nspd:Ndbl:Ntwl:Ntspd:Ntbl. The first point has the fastest clock period (and highest power) the third point the slowest clock period (but lowest power), while the middle point is taken from the knee of the pareto curve.

process, with dual-Vt implementations resulting in significantly lower power dissipation compared to single-Vt implementations. This is followed by the differential or single-ended sensing, with implementations using single-ended sensing having higher total power (due to increased dynamic power in the bitlines) compared to low-swing differential sensing. Finally, the static or dynamic decode parameter was shown to hve minimal effect on power as seen in the pareto optimal plots.

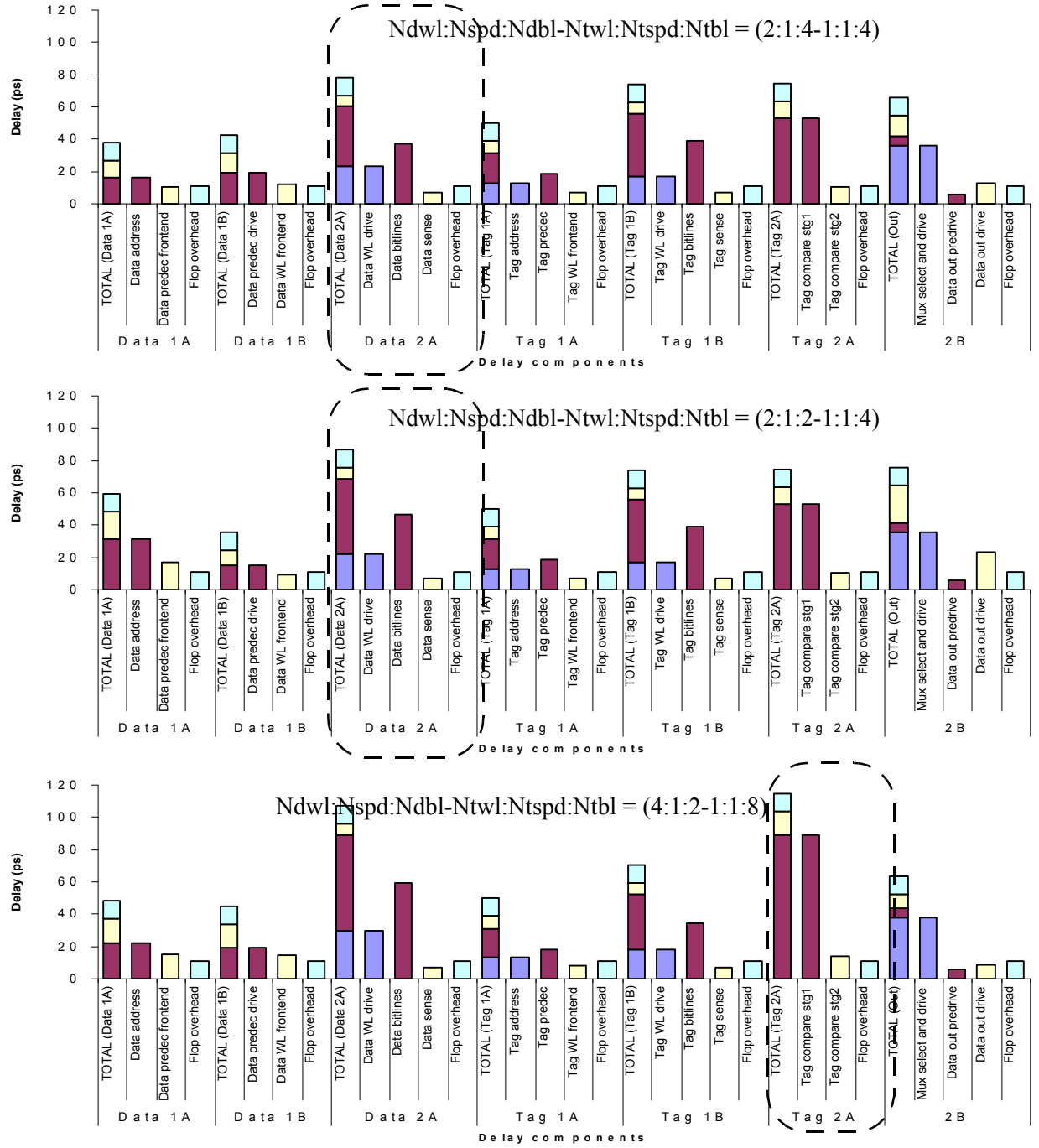


Figure 5-47: Detailed delay breakdown for three pareto optimal implementations of a 64kB 4-way 32nm cache. Th plots give detailed breakdowns of the pipeline stage delays shown in the previous figure. The stages encircled in the .rectangles denote the critical path delay of the cache that sets its clock period.

With respect to the cache clock period, the parameter that resulted in the biggest difference was the choice of single-ended or differential sensing, with single-ended sensing being significantly lower than implementations using differential sensing. This is followed by the choice of decode logic, with implementations using static decode having typically slower implementations than ones using dynamic decode. Finally, implementations using a dual-Vt process may result in a slightly slower delay than a single-Vt implementation. With respect to timing, it is important to stress that even though the choice of parameters may change the delay of one or more parts of the cache, this delay does not necessarily result in a degradation of the cache clock period. If it does or not is dependent on whether the degraded stage is actually the critical path of the cache. Any degradation in delay of a stage that has enough slack to absorb it will not degrade the critical path delay of the cache. This is not the same for power, where the increase in power of one stage adds to total power regardless of whether or not that particular stage contains the critical path. Of course, the opposite argument is the same, where savings in power in a particular stage will always affect total power.

From these studies, we conclude that it is very important to study the different parameter permutations in detail in order to explore all the tradeoffs involved and account for the fact that the model still is not complete, and never will be simply because of the impractical number of variables involved in the design. One simple example is the concept of design complexity, where it is often desirable to have a static implementation of a circuit instead of dynamic in order to avoid all the design problems (e.g. noise susceptibility, clocking complexity, verification complexity) of dynamic logic. Of course, use of static logic will typically result in a slower circuit. With proper exploration of the design space, we can identify specific implementations that have good power and delay behavior but at the same time does not contain any part of the decoder in the critical path. This way, it may be possible to implement the decoder purely with static gates, incurring the additional delay in the decoder stages but, as long as enough slack remains in those stages, not degrading the cache clock period even by a little amount. This results in a win-win design situation as we get the power behaviour we want, with less design complexity and with the same clock frequency.

One important observation that we emphasize is that it seems to be really important to use a dual-Vt process instead of a single-Vt one. Implementations using a dual-Vt process will have significantly smaller power dissipation compared to single-Vt implementations with little delay degradation. For implementations that don't contain the bitlines in their critical paths, there is actually no delay degradation, as the real critical path will have all LVT transistors while having HVT transistors for the non-critical path such that significant power is saved without degrading the delay in the critical path whatsoever. One thing that this does not account for is that dual-Vt processes are typically more expensive than a single-Vt process because of the additional masks involved. Of course, with the current industry prioritization of power dissipation, this is often an acceptable cost. In any case, shifting from LVT to HVT in the design process is typically simple, as it simply involves additional layout masks to implement the higher threshold voltages. In other words, the mask

needed for an HVT implementation is typically a superset of the masks for an LVT design, with most of the masks unchanged. This way, as long as the design is verified for both pure-LVT and hybrid LVT and HVT transistors, different variants of the same design can easily be created resulting in different performance and power dissipation.

Finally, we emphasize the potential design benefits of having accurate cache delay and power breakdowns when exploring the design space. With these data, it is easier to explore design tradeoffs because the designer can make well-informed decisions based on the numbers given by the tools.

5.5 Conclusion

This chapter has demonstrated the usefulness of myCACTI as a tool in cache design. We have demonstrated how we can generate pareto optimal configurations for various cache configurations with different cache sizes, associativities and technology nodes to provide designers with significantly more flexibility and design choices than CACTI or eCACTI. Aside from the genration of pareto optimal curves, we have also shown the detailed read-hit power and delay breakdown of a specific cache configuration (64kB 4-way). With the availability of the pareto optimal points (as well as design points close to the pareto optimal curve), along with the detailed delay and power breakdown, we have demonstrated a possible cache design flow that makes it possible to systematicaly identify implementations that can be optimized even further with a reasonable amount of focused design effort. This allows the designer another level of flexibility to come up with a cache design that is suited to the desired requirements. Finally, we have also shown design space explorations using the different process and circuit parameters that myCACTI supports, specifically the choice of a single-Vt or dual-Vt process, choice of static or dynamic decoding, and choice of low-swing differential or full-swing single-ended sensing. myCACTI supports a few more possible options, but the scope of this study was intentionally limited to these three options for clarity. We have shown how using one technique or option over another results in a change in either power or delay or both. Lastly, we have mentioned how this can also be used for the cache design flow we have mentioned, where one can systematically identify potential candidates for optimization more easily with the availability of detailed information along with the pareto optimal curves.

CHAPTER 6 Gate leakage characterization

6.1 INTRODUCTION

The ITRS 2001 roadmap projected an exponential increase in gate leakage currents with continuing technology scaling such that it was predicted to exceed the subthreshold leakage current in the deep nanometer technology nodes. As such, major research effort has been exerted to study gate leakage reduction techniques in an attempt to address the drastic increase in gate leakage tunneling currents.

But with the more recent versions of the ITRS, the scaling of the oxide thickness has been made less aggressive. Consequently, although gate leakage current still significantly increases on a per unit area basis, the projected rate of increase has been made much less aggressive compared to previous projections.

The gate oxide thickness, though, is just one, albeit major, factor in determining gate leakage. We will show that some of these factors tend to cause a decrease in gate leakage currents such that the overall effect is a net decrease of gate leakage current from one technology node to the next. Specifically, using SPICE BSIM4 equations, we have seen that the combination of oxide thickness decrease (increasing gate leakage), Vdd downscaling (decreasing gate leakage) and device size scaling (decreasing gate leakage) actually results in a net decrease in total gate leakage tunneling current, making gate leakage much less of a problem than previously expected. In addition, more widespread use of high-Vt transistors in an attempt to minimize subthreshold leakage currents also contribute to a decrease in gate leakage currents, reducing the magnitude of the problem even further.

6.2 Background

Gate leakage tunneling currents are caused whenever voltages are applied to the terminals of a transistor, producing an electric field across the gate oxide that facilitates the production of tunneling currents. With increasing thinner gate oxides as a result of technology scaling, this leakage tunneling current is projected to increase significantly from one technology node to the next.

Figure 6-1 shows the different components of gate leakage tunneling currents in a MOSFET. Although the figure shows five arrows representing the different gate leakage current paths, the five different currents can be grouped into the channel tunneling current, the junction overlap tunneling current, and the bulk tunneling current. These gate leakage tunneling mechanisms can be modeled using SPICE BSIM4 equations (included in the appendix).

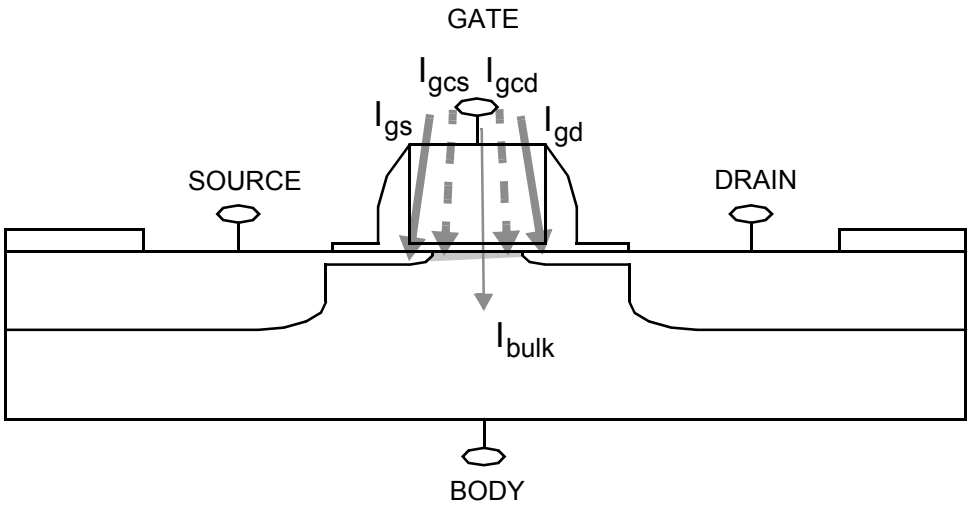


Figure 6-1: MOSFET Gate leakage tunneling currents. The five different gate leakage tunneling currents in a MOSFET are shown. These five can be categorized into three groups: the channel tunneling current, the source/drain junction overlap tunneling current, and the bulk tunneling current.

6.3 EXPERIMENTAL METHODOLOGY

In this study, we use the SPICE BSIM4 equations shown previously, along with predictive nanometer SPICE models, to study the different factors affecting the gate leakage tunneling currents. Specifically, we use predictive models for 130nm, 90nm, 65nm, 45nm, and 32nm for the study. Table 1 shows the values for frequency and supply voltages that were used for this study.

Table 1: VDD and frequency used for each tech node.

	250nm	180nm	130nm	90nm	65nm	45nm	32nm
VDD (V)	2.0	1.8	1.6	1.4	1.3	1.2	1.1

We implement the SPICE BSIM4 equations in Matlab to study the different factors involved in determining gate leakage current.

We also use two different sets of predictive models. One set of models use aggressive scaling of the gate oxide thickness, reflecting the ITRS2001 projections, while the second set of models use a more moderate scaling, reflecting ITRS2003/ITRS2005 projections. Results labeled with aggressive scaling are generated using the first set of predictive models, while results labeled with moderate scaling are generated using the second set.

Lastly, although we consider the bulk tunneling currents in our experiments, we have seen that these are typically insignificant compared to the channel and junction overlap tunneling currents and hence, omit any mention of the bulk tunneling currents for conciseness.

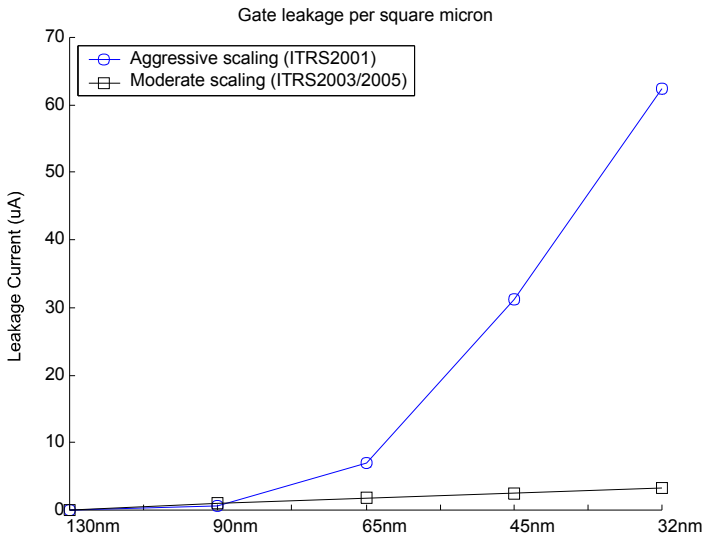


Figure 6-2: Gate leakage current per unit area. The plot shows the total gate leakage current per square micron for both aggressive and moderate scaling as a function of technology node.

6.4 Main gate leakage factors

6.4.1 Gate leakage current big picture

Figure 6-2 shows the total gate leakage current per unit area (um^2) using ggressive and moderate scaling plotted against technology node. It can be easily seen that while the gate leakage current increases significantly using the aggressively scaled SPICE models, the rate of increase as seen using the moderately scaled SPICE models is drastically less, resulting in a much slower increase in gate leakage current per unit area. If we now take into account the fact that the actual transistor device sizes used will typically be decreasing from one technology to the next (e.g. a simple process shrink), the total area where gate leakage causing mechanisms are present will also be smaller.

Figure 6-3 shows the total gate leakage current in a 4X-strength inverter of size L=Lmin and W=Wmin (where Wmin = 3 x Lmin), where Lmin is directly determined by the technology node. Although we are now directly comparing inverters with different total area, the comparison is still valid because the inverters have the same functionality across all the technology nodes. For example, when an existing design implemented in 90nm undergoes a straight-up process shrink to produce a 65nm design, a 4X-strength inverter in 65nm will have the exact same functionality in the design as a 4X-strength inverter in 90nm, even though the 65nm inverter's device area is less than the 90nm inverter because of device scaling.

It can be easily seen that for aggressively scaled SPICE models, the gate leakage current flowing across a 4X-strength inverter is still increasing. But when we use moderately scaled SPICE models, we see that the total gate leakage current across a 4X inverter actually decreases from one technology generation to

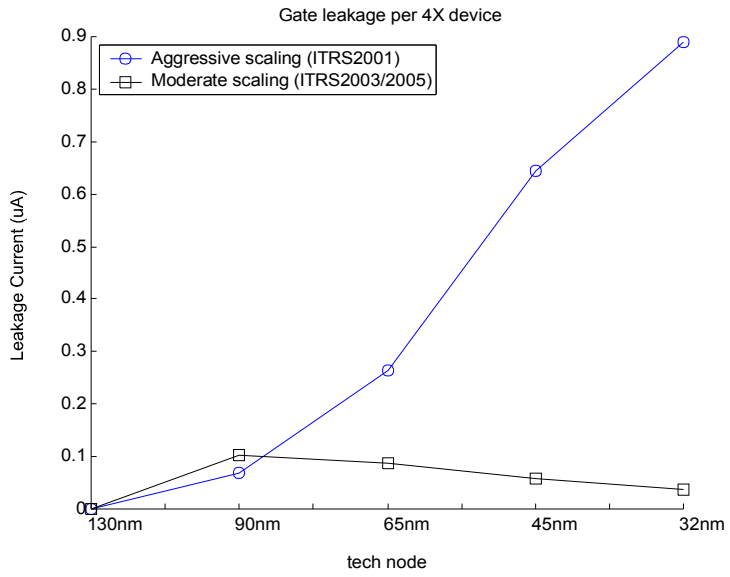


Figure 6-3: Gate leakage current per 4X inverter. The plot shows the total gate leakage tunneling current for a given 4X inverter for both aggressive and moderate scaling as a function of technology node.

the next. This is a very important observation that shows that the gate leakage current across a typical device in a design may actually decrease with technology scaling.

The previous results show the intertwined effects of the different factors involved in technology scaling. As such, it is important to isoltage the effects of individual factors. The next two sections try to explore some of these factors in more detail.

6.4.2 Vdd scaling

In this section, we isolate effect of Vdd scaling on the gate leakage current. Figure 6-4 shows multiple plots (one for each technology node) showing the effect of varying Vdd on the total gate leakage current per unit area (using both aggressive and moderate scaling).

It can be easily seen that Vdd has an exponential effect on the gate leakage current. This is a very important result since Vdd typically undergoes a downscaling across technology nodes. This exponential relationship of gate leakage and supply voltage will serve to significantly offset the effect of the expnential relationship of gate leakage with gate oxide thickness. Figure 6-5 shows similar plots and results for the gate leakage currents across a 4X inverter. Finally, Figure 6-6 demonstrates that the effect of Vdd scaling is even more significant when we consider that power is the product of voltage and current, such that Vdd scaling contributes to the decrease in gate leakage power both in the exponential reduction effect in the gate leakage, and the direct reduction in the supply voltage.

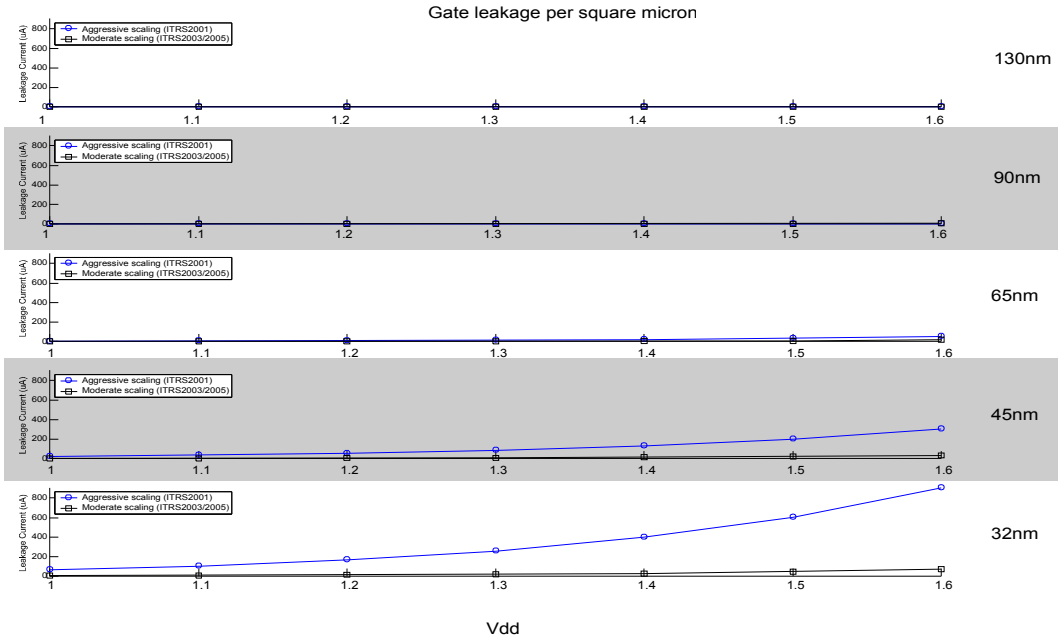


Figure 6-4: Vdd scaling (per square micron). The effect of Vdd scaling on gate leakage current per unit area is shown on the plots for different technology nodes for both aggressive and moderate scaling.

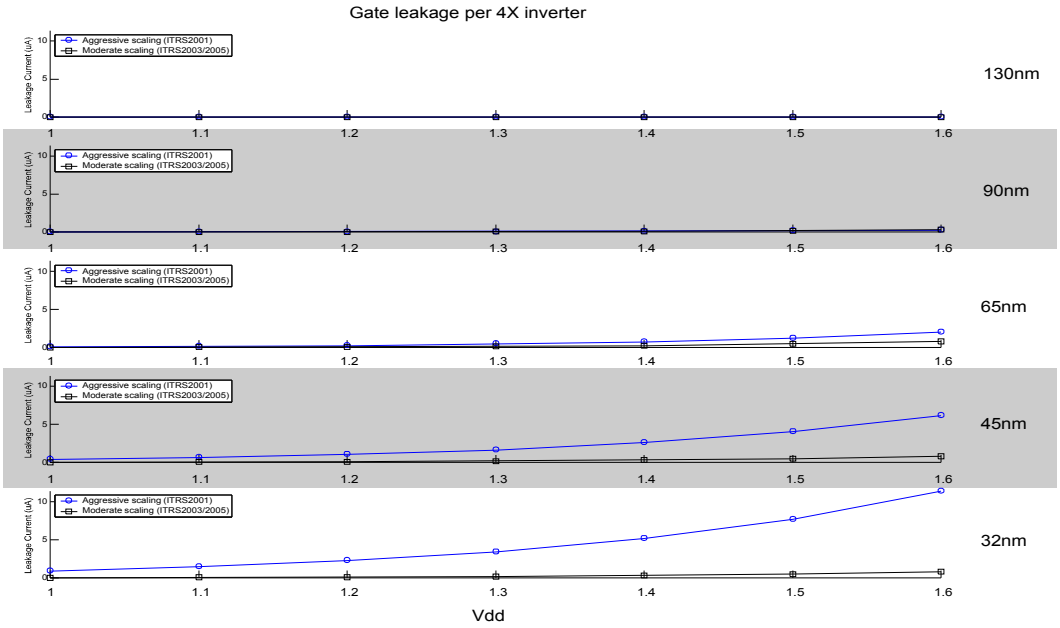


Figure 6-5: Vdd scaling (per 4X device). The effect of Vdd scaling on gate leakage current per 4X inverter is shown on the plots for different technology nodes for both aggressive and moderate scaling.

6.4.3 Process parameter scaling

In this section, we isolate the effect of the scaling of the process parameters on the gate leakage current. Although we are studying the scaling of all process parameters, we expect most of the effect is due to the scaling of the oxide thickness parameters.

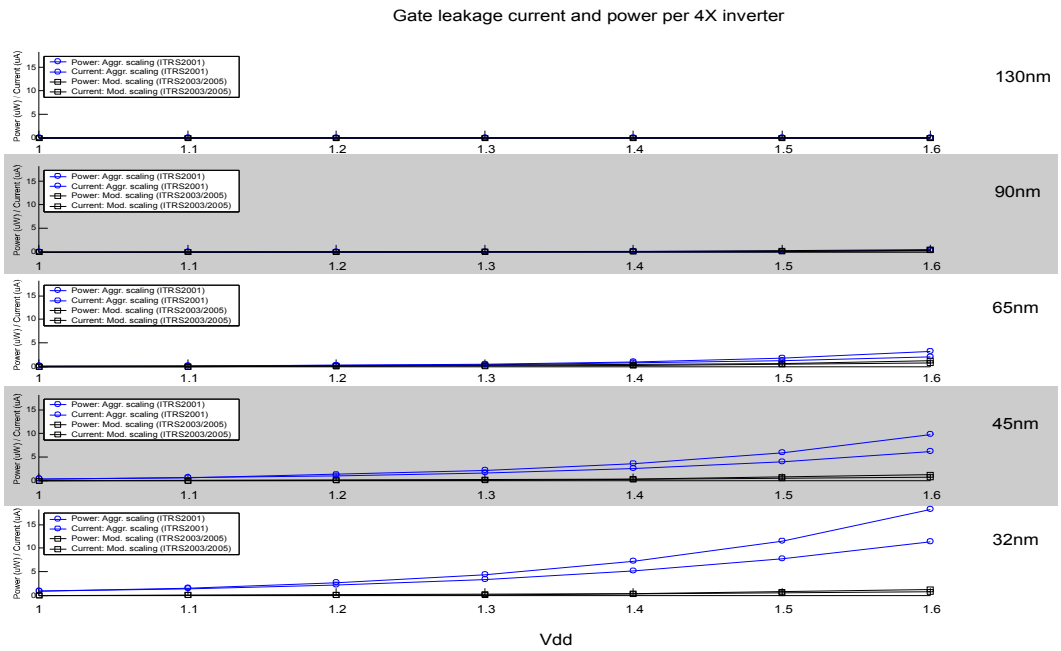


Figure 6-6: Vdd scaling (Power and Current). The effect of Vdd scaling on gate leakage current and power per 4X inverter is shown on the plots for different technology nodes for both aggressive and moderate scaling.

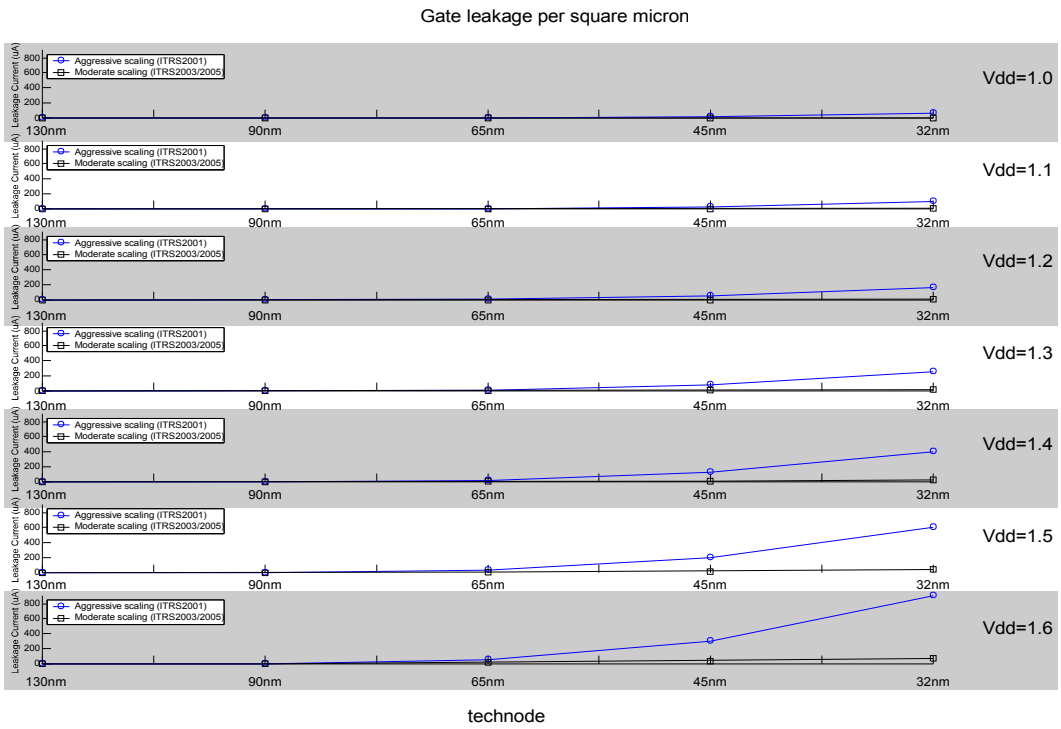


Figure 6-7: Process scaling (per unit area). The effect of scaling the process parameters on gate leakage current per unit area is shown for different values of supply voltage for both aggressive and moderate scaling.

Figure 6-7 shows multiple plots (for different values of supply voltage) showing the effect of scaling process technology parameters (as indicated by the technology node) on the total gate leakage current per unit area (using both aggressive and moderate scaling). It can be seen that for a fixed supply voltage, the gate

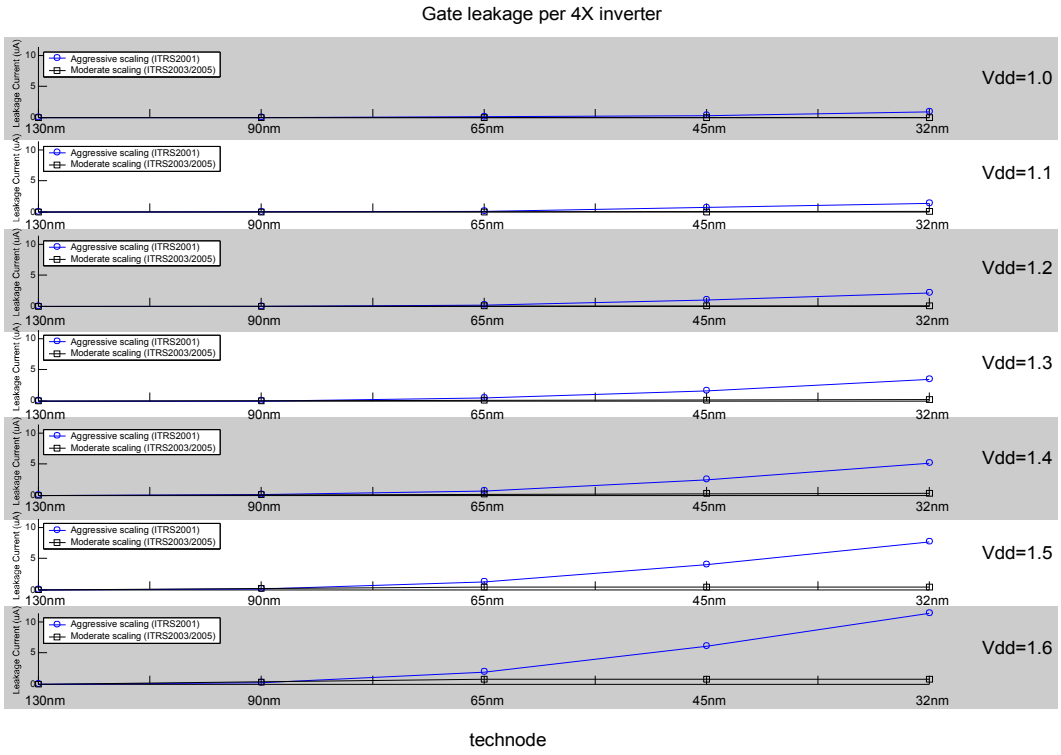


Figure 6-8: Process scaling (per 4X inverter area). The effect of scaling the process parameters on gate leakage current per 4X inverter is shown for different values of supply voltage for both aggressive and moderate scaling.

leakage current exponentially increases with the technology node for both aggressive and moderate scaling. When we consider typical devices, though, we see that device size scaling serves to dampen the rate of increase of the gate leakage current, as seen in Figure 6-8 where the rate of increase for the 45nm and 32nm nodes now seem to be simply linear.

6.4.4 Putting it all together

The factors studied in the previous two sections attempted to isolate the different contributors to the gate leakage current during technology scaling. We have observed the following effects:

- Vdd-scaling: downscaling of the supply voltage causes an exponential reduction in gate leakage current.
- Device-size scaling: device sizes typically decrease from one technology to the next, reducing gate leakage current.
- Process parameter scaling: technology scaling effects on the fabrication process parameters causes an exponential increase in gate leakage current, mostly caused by the thinning gate oxide.

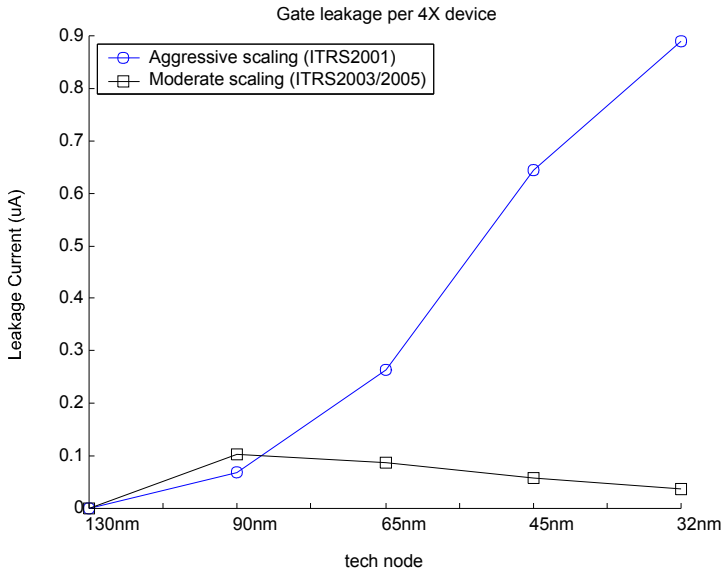


Figure 6-9: Gate leakage current for a 4X inverter. The plot shows the total gate leakage tunneling current for a given 4X inverter for both aggressive and moderate scaling as a function of technology node. (repeated from Figure 6-3)

Although projections about the effect of gate leakage previously concentrated on the oxide scaling effect (part of the third), we have shown (in Figure and repeated in Figure 6-9) that when we also account for the decreasing supply voltage and device size, the total gate leakage actually decreases.

6.4.5 Circuit example

In this section, we study different circuit examples and try to apply the knowledge gained in the previous sections.

We consider the 2-kB array shown in Figure 6-10, where we analyze the gate leakage behavior of the memory array, and the array decoder. In addition, we consider two types of decoders for this memory array. The first decoder we consider uses a simple static CMOS decoder implementation,while the second decoder is implemented using SCL-DRCMOS techniques.

The results are shown in Figure 6-11. The solid lines show the gate leakage using the aggressively scaled SPICE models, while the dotted lines show the gate leakage using the moderately scaled SPICE models. Consistent with the previous results, the gate leakage currents using aggressive scaling are larger than that for moderate scaling except for the 90nm node.

We can see from the results that a static decoder will have the largest gate leakage current, followed by the memory array, with the SCL DRCMOS decoder having the least leakage current. This shows that the SCL DRCMOS decoder has superior gate leakage current behavior compared to the static CMOS implementation, and would probably be the preferred implementation if we used aggressive scaling. When

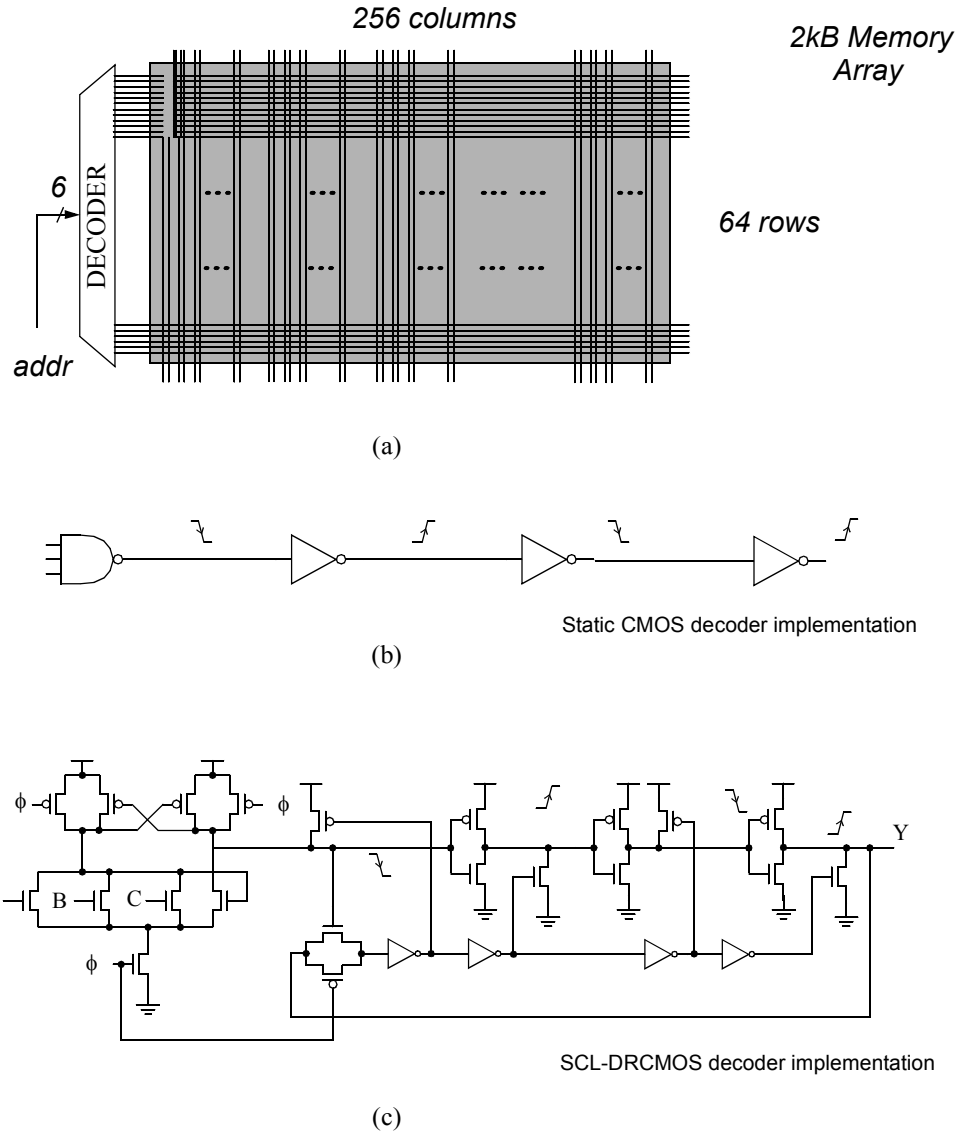


Figure 6-10: A 2-kB memory array. (a) The high-level block diagram showing the decoder and the actual memory array. (b) A static CMOS implementation of the decoder. (c) An SCL-DRCMOS implementation of the decoder

using moderate scaling, though, the advantages of the SCL-DRCMOS lessens such that its complexity (due to its dynamic nature) might make it less attractive overall compared to the static implementation.

6.5 Breakdown of gate leakage current

The gate leakage current values presented in previous sections all refer to the total gate leakage current, which is the sum of the channel tunneling current, the source/drain junction overlap tunneling current, and the bulk tunneling current (which we omit for conciseness). It is also useful to explore the breakdown of gate leakage current into these components.

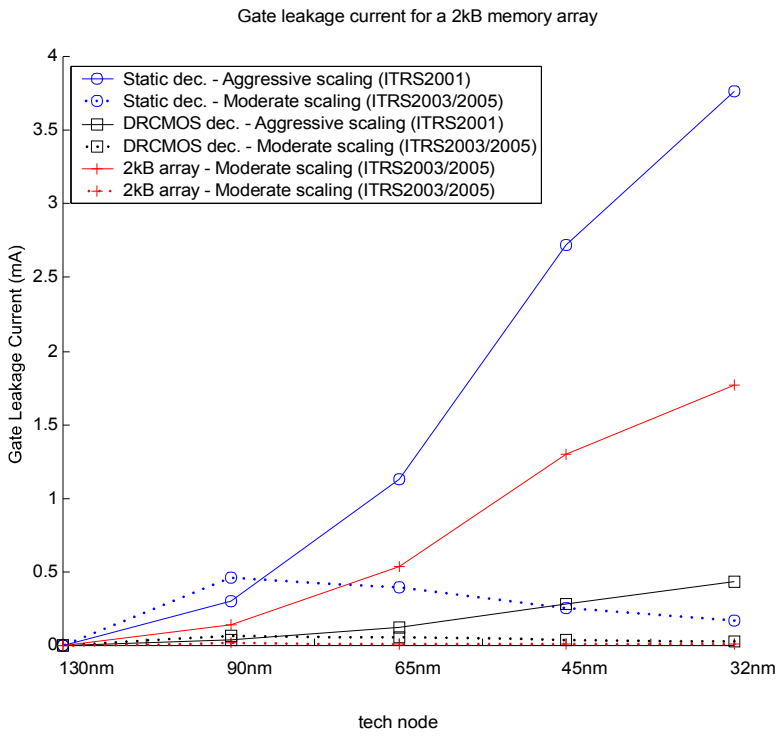


Figure 6-11: Gate leakage currents for a 2kB memory array. The total gate leakage currents for a 2kB memory array and for two types of decoder implementations (static CMOS and SCL-DRCMOS) are shown as a function of technology node for both aggressive and moderate scaling.

Figure 6-12 shows the breakdown of gate leakage current using aggressive scaling on a per unit area and per 4X inverter basis. It shows that the junction tunneling current is very significant to the point that it actually becomes greater than the channel tunneling current at the 32nm node. Figure 6-13 shows the breakdown of gate leakage current using moderate scaling on a per unit area and per 4X inverter basis. In contrast to the aggressive scaling results, the junction tunneling current is now significantly less than the channel tunneling current.

6.6 Threshold voltage

The previous sections show results that use low voltage threshold transistors. In state of the art microprocessors, different threshold voltages are typically used in an attempt to reduce the subthreshold leakage currents. Typically, low-Vt transistors are used mainly for speed-critical devices that are in the procesor's critical path, in which case the additional power dissipation due to the higher subthreshold leakage of the device is an acceptable tradeoff in exchange for faster operation. High-Vt transistors, on the other hand, are used for non-critical devices where speed loss is acceptable in exchange for lower power dissipation.

With changing Vt, the gate leakage current behavior also changes. Figure 6-14 shows the gate leakage current on a per unit area basis as a function of relative Vt (relative to an LVT transistor) for different

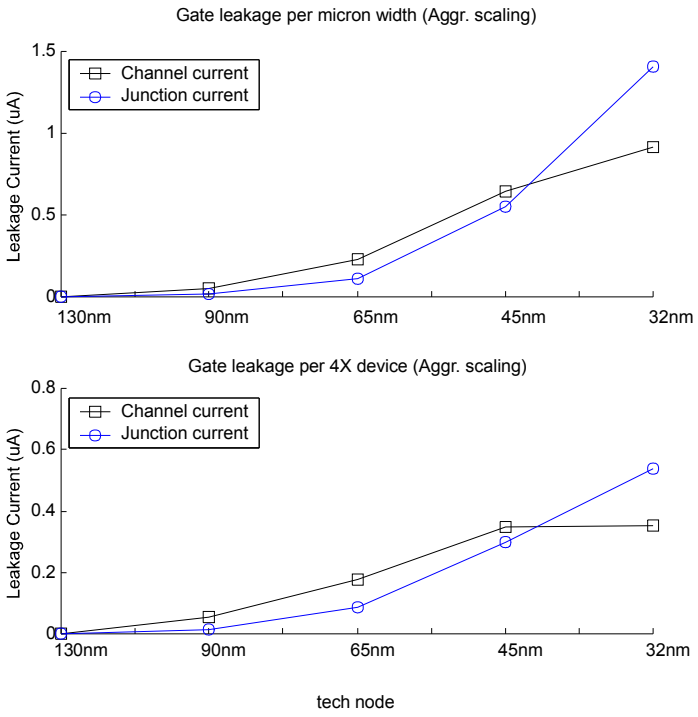


Figure 6-12: Gate leakage current breakdown (aggressive scaling). The breakdown of total gate leakage current (on a per unit area and per 4X inverter basis) into the channel current and the junction overlap current are shown as a function of technologynode using aggressive scaling.

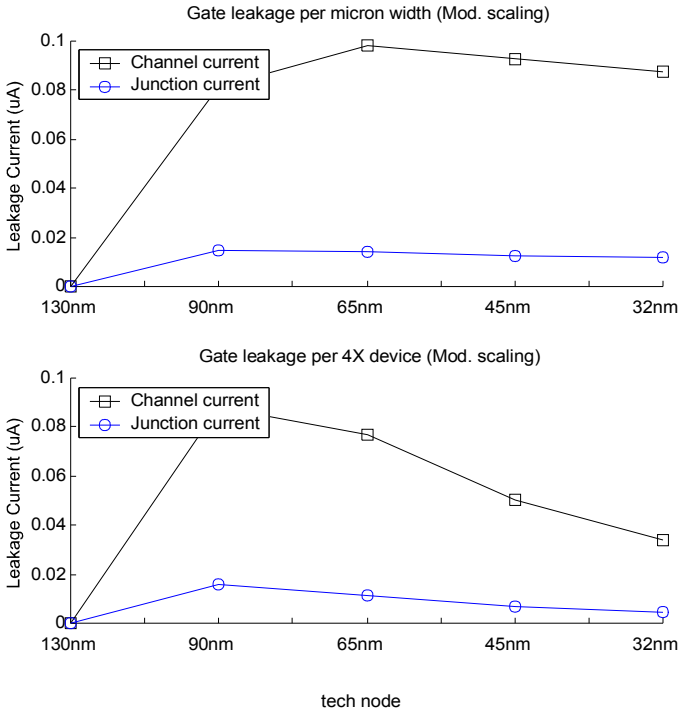


Figure 6-13: Gate leakage current breakdown (moderate scaling). The breakdown of total gate leakage current (on a per unit area and per 4X inverter basis) into the channel current and the junction overlap current are shown as a function of technologynode using aggressive scaling.

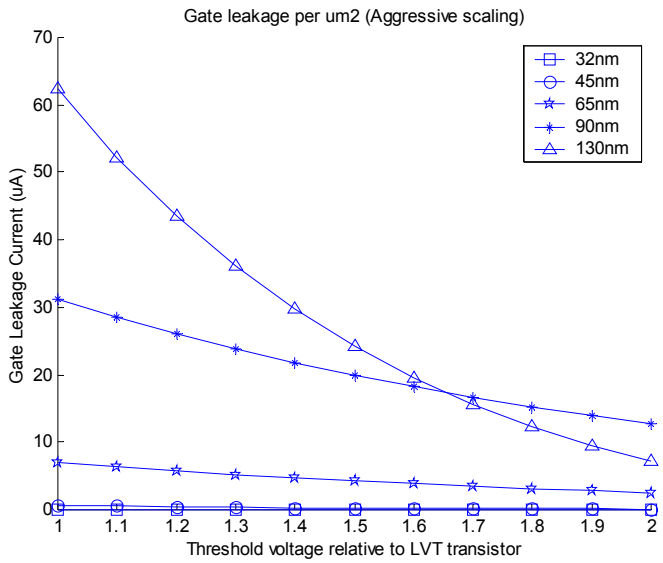


Figure 6-14: Gate leakage vs. threshold voltage . The effect of increasing the transistor threshold voltage on the gate leakage current per unit area is shown for different technologies using aggressive scaling.

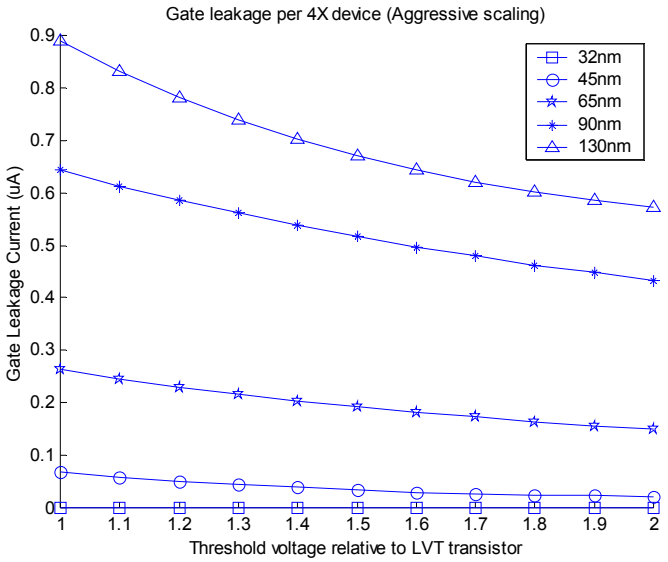


Figure 6-15: Gate leakage vs. threshold voltage . The effect of increasing the transistor threshold voltage on the gate leakag current per 4X inverter is shown for different technologies using aggressive scaling.

technology nodes (using aggressive scaling). Figure 6-15 shows the gate leakage current on a per 4X inverter basis as a function of relative Vt (relative to an LVT transistor) for different technology nodes (using aggressive scaling). We can see that higher threshold voltages result in smaller gate leakage currents, mostly due to the reduction in the gate overdrive voltage that induces the conducting channel in the MOSFET. This means that higher threshold voltage transistors will have less gate leakage than their lower threshold counterparts.

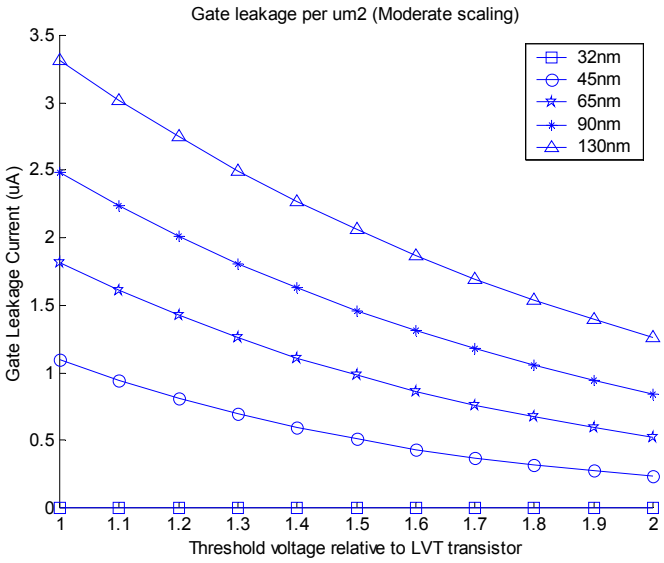


Figure 6-16: Gate leakage vs. threshold voltage . The effect of increasing the transistor threshold voltage on the gate leakage current per unit area is shown for different technologies using moderate scaling.

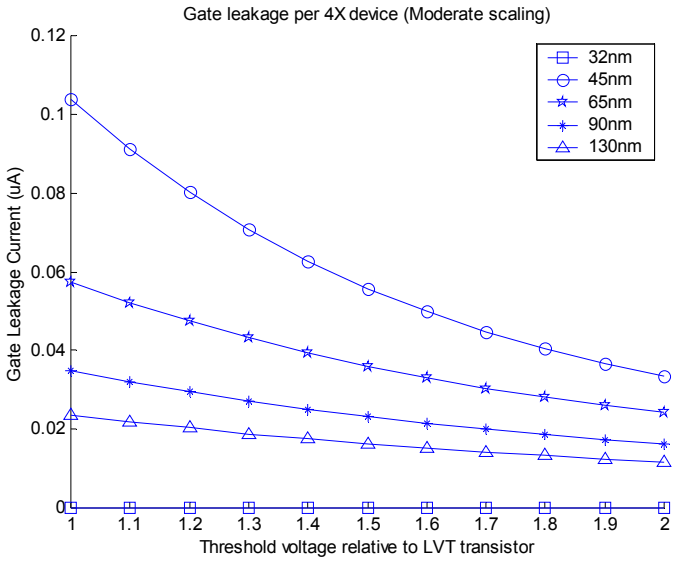


Figure 6-17: Gate leakage vs. threshold voltage . The effect of increasing the transistor threshold voltage on the gate leakag current per 4X inverter is shown for different technologies using moderate scaling.

Figure 6-16 and Figure 6-17 show corresponding plots for moderate scaling. The results are mostly similar, except for the faster rate of decrease of gate leakage current as a factor of threshold voltage. This is due to our earlier observation that the channel tunneling current is significantly greater than junction tunneling current for the moderate scaling results. Since the same does not hold with the aggressive scaling results, a reduction in the threshold voltage that results only in the reduction of the channel tunneling current (the junction tunneling current is not affected whatsoever by the threshold voltage), the decrease in gate leakage is much faster for moderate scaling than for aggressive scaling.

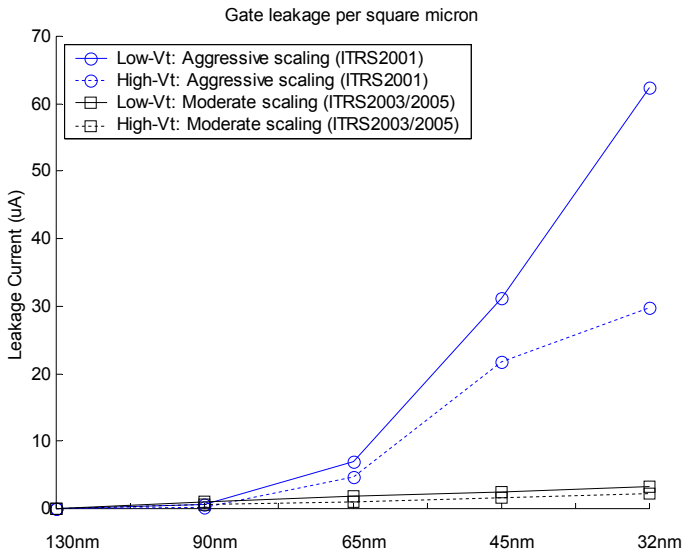


Figure 6-18: Gate leakage vs. threshold voltage . Figure 6-2 is recreated using an LVT (the original) and superimposing the results of using HVT devices (with HVT being twice LVT). This shows the effect of using HVT over LVT transistors on gate leakage current per unit area.

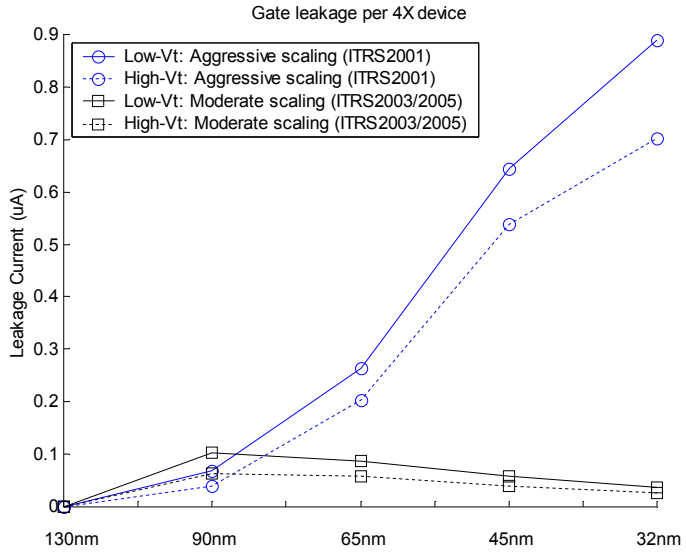


Figure 6-19: Gate leakage vs. threshold voltage . Figure 6-3 is recreated using an LVT (the original) and superimposing th results of using HVT devices (with HVT being twice LVT). This shows the effect of using HVT over LVT transistors on gate leakag current per 4X inverter.

Figure 6-18 and Figure 6-19 modifies Figure 6-2 and Figure 6-3 and shows the total gate leakage current on a per unit area and per 4X inverter basis. The figures show both the original curves (shown as solid lines) denoting runs using low-Vt transistors, and the corresponding curves (shown as dotted lines), when high-Vt transistors are used (with HVT set to twice the LVT). We can easily see that use of high threshold voltage transistors lower result in a lowering of gate leakage currents with respect to that of low threshold voltage transistors.

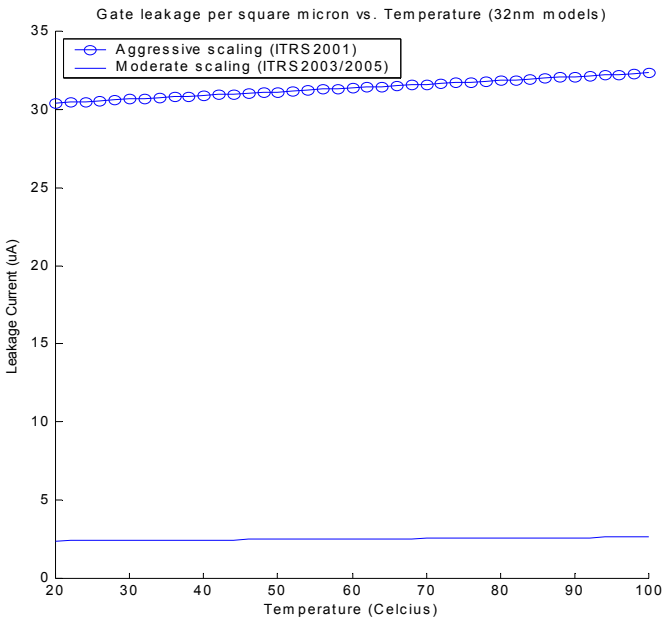


Figure 6-20: Temperature effects on gate leakage current. The effect of temperature is shown for both aggressive and moderate scaling using a 32nm process technology.

6.7 Temperature effects

Figure 6-20 shows the gate leakage current as a function of temperature for both aggressive and moderate scaling. The figure shows that sweeping the temperature from 20 C to 100 C does not result in significant changes in the temperature. We therefore conclude that the gate leakage current only has a weak dependence on temperature in that a significant rise in temperature only results in a nominal rise in gate leakage current. This is seen from the gate leakage current equations in the earlier sections, where the temperature has minimal influence on the gate leakage equations.

6.8 Discussion

In the previous experiments, we have demonstrated that gate leakage is affected by four factors: the gate oxide thickness, the supply voltage, the transistor device dimensions, and the transistor threshold voltage.

We have shown that when the aggressively scaled SPICE models are used (roughly representing ITRS2001 projections), the scaling of the gate oxide thickness is such that it more than makes up for any decrease in gate leakage current caused by the other factors. As a result, we see that the gate leakage current increases significantly from one technology generation to the next, both in terms of a per unit area basis (accounting for oxide and Vdd scaling and HVT transistors), and on a per 4X-strength inverter basis (accounting for all four factors).

We have also shown that when the moderately scaled SPICE models are used (roughly representing updated ITRS2003 and ITRS2005 projections), the scaling of the gate oxide thickness is slowed down such that the rate of increase of gate leakage current on a per unit area is slowed down drastically such that the gate leakage current for the deep nanometer nodes are now much less than that predicted using the aggressively scaled SPICE models. In addition, when looking at gate leakage current on a per device basis (e.g. a 4X inverter), we see that the value of gate leakage current from one technology generation to the next actually decreases.

This is a very important observation, as it demonstrates that as long as gate oxide thicknesses are not scaled as aggressively as originally projected in ITRS2001, other factors contribute to a reduction in gate leakage such that the problem doesn't escalate and actually gets reduced from one technology generation to the next.

Finally, it is also important to emphasize that all of our simulations have not used high-K gate dielectric technology in an effort to reduce gate leakage. The ITRS has stated that one of the biggest process challenge is the continued reduction of the gate oxide thickness. The ITRS2005 also states that currently, high-K dielectrics are still a problem because no suitable material has been found that can be practically integrated with the current front-end-of-line process while resulting in a high-quality dielectric. As such, the projections show the use of silicon oxynitride (which is not a high-K material) as a dielectric and slow down the rate of decrease of oxide thickness, at the expense of losing some control over the transistor operation (in the form of a poorly-scaled Cox or oxide capacitance which is essential to controlling device operation). The ITRS2005 mentions that, fortunately, this loss of control and resulting sacrifice in speed is more than made up for by enhancements in transistor channel mobility[1] obtained from technologies like SiGe and strained silicon, resulting in a net increase in transistor speed (which is one of the things we want when doing technology scaling).

The main point here is that even without using high-K dielectrics, we have shown that gate leakage currents may actually decrease with technology scaling as long as the oxide thicknesses are not aggressively scaled. When high-K dielectrics are eventually developed, this allows increasing the oxide thickness while retaining control over device operation, reducing the gate leakage current problem even further.

1. It does need to be mentioned that by improving mobility of the transistor as a way to offset the lost performance gains of using moderately scaled oxides, it will potentially result in the increase of subthreshold leakage, which is also dependent on the channel mobility.

6.9 Conclusion

We have demonstrated that although previously projected aggressive scaling of the gate oxide thickness results in drastic increases in gate leakage tunneling currents, current projections use a more moderate scaling of the gate oxide thickness. As such, the increase in gate leakage current due to decreasing oxide thickness is offset by a reduction in the gate leakage current due to Vdd and device scaling, along with more widespread use of high threshold voltage transistors, and that the net effect of all four factors actually result in a decrease in gate leakage current from one technology generation to the next.

CHAPTER 7 Conclusion

7.1 Summary and Contributions

This dissertation first provides a detailed tutorial of prevailing techniques for cache and SRAM design to be used as a background for understanding the subtleties of current cache design tools. Details on the state of the art of these cache design tools were then provided by discussing in detail the design tools CACTI and eCACTI. The major limitations of these two tools were then discussed, proving that although these tools are currently the best tools available in the academic setting, they will be insufficient for use in designing pipelined caches, especially for nanometer process technologies.

A new and enhanced cache design tool is then introduced, which we call myCACTI. Our new design tool advances the state of the art in cache design tools that use analytical modeling by including, among many other things, the following improvements:

- Use of SPICE BSIM4.0 equations to accurately characterize device behavior for nanometer process technologies. In contrast, CACTI and, to a major extent, eCACTI simply use hardcoded parameters derived for an obsolete 0.80μm process technology.
- The modeling of a typical explicitly-pipelined cache, which accounts for all the overhead in pipelining that will be present in virtually all industry-level microprocessor caches. In contrast, CACTI and eCACTI model wave-pipelined cache, something that is not representative of commercial designs.
- Inclusion of more optimal variable stage dynamic logic circuits for the decode hierarchy that provides the tool mor
- e flexibility in finding optimal implementations. In contrast, both CACTI and eCACTI model a fixed-stage static CMOS decode hierarchy, significantly limiting the optimization search.
- Inclusion of an accurate model and per-process numbers for a typical BEOL-stack that is representative of nanometer processes. The significance of this is made even more important given the tremendous effect of interconnect parasitics on a cache's behavior.
- Inclusion of a gate leakage tunneling current model for power dissipation.
- Inclusion of a very realistic interconnect model that is representative of the interconnects in a nanometer cache. In contrast, both CACTI and eCACTI have an unrealistic model of the

interconnect as they assume the use of interconnect with a single characteristic no matter where it is used in the cache.

We show that even after a major limitation of CACTI and eCACTI in the address decode hierarchy is fixed (i.e. by making the number of stages in the decode hierarchy variable and requiring that the first stage drive strength is of a reasonable number so as not to cause additional delay before the cache), the CACTI/ eCACTI numbers will still deviate by a reasonable amount from what it should be, as modeled by myCACTI.

We then use myCACTI to conduct detailed studies of a big portion of the cache design space, producing pareto optimal curves of the best implementations for every configuration or point in the design space. Very detailed studies of a small part of the design space (i.e. 64kB 4-way) are then done, producing detailed power and delay breakdowns for these caches, demonstrating how myCACTI can produce these useful information. Among the general observations that were seen are the following:

- The pipeline power dissipation overhead is very significant and that is typically dominates the total power.
- Interesting non-monotonic behavior with respect to delay and power dissipation for caches with different associativities exist, such that we can conclude that some optimal implementations are definitely superior than others.
- The power dissipation due to gate leakage tunneling current is surprisingly not as significant as initially expected.

Finally, detailed studies of gate leakage tunneling current was done to determine why static power dissipation due to gate leakage currents were relatively insignificant even at the very deep nanometer technology nodes like 32nm. We find that even though the parameters in the SPICE models that we use still result in an increasing gate leakage tunneling current per unit area, this increase has drastically slowed down compared to the increase previously projected. Consequently, when Vdd-scaling and device-size scaling are also accounted for, we find that the gate leakage current for a device of an arbitrary drive strength actually decreases from one generation to the next.

7.2 Limitations

Besides the limitations already put forth in the section discussing myCACTI in detail, it must also be emphasized that all the study results were generated using a given set of predictive SPICE BSIM4.0 nanometer transistor models for 130nm, 90nm, 65nm, 45nm and 32nm [PTM2006]. Hence, the analysis and conclusions put forth in this dissertation are necessarily attached to the SPICE models used. Use of different SPICE models most probably will result in different numbers, although the degree of difference would depend

on how the degree of the change, and especially on whether scaling trends are similar or not. Nevertheless, one of myCACTI's strengths, as has been emphasized before, is its ability to use any SPICE BSIM4.0 model that can be given by a user and produce results based on the given SPICE deck. This is a major enhancement over CACTI and eCACTI, which are both dependent on hardcoded parameters that can only be reliably modified by a user with great difficulty.

7.3 Related Work

myCACTI was initially based on the infrastructure of CACTI and eCACTI, and it advances the state of the art by implementing major enhancements and improvements over these two cache design tools. Before CACTI and eCACTI both also borrowed from previous work on analytical modeling for cache design, specifically [Wada1992] and [Mulder1991]. In addition, other analytical models exist that model caches to some extent [Kamble1997, Hanson2001, Skadron2003, Butts2000, Mamidipaka2004b, Evans1995], although these models are not as extensive and complete as those of CACTI and eCACTI and, consequently, myCACTI.

7.4 Future Work

Finally, we finish this disseration by enumerating possible additional enhancements that could be incorporated into myCACTI to improve its capabilities:

- Rewrite the helper scripts that convert and compute the SPICE BSIM4.0 parameters into the file technology.c which is then recompiled into the final executable. These scripts are mainly Matlab and perl scripts that were implemented as such because of the easier development environment for those platforms.
- Expose other operating parameters during the simulation to the user to add more flexibility. Examples of these parameters are the interconnect layer assignment for different signals in the cache, and the dielectric constant of the BEOL ILD.
- Improve the accuracy of the calculation using SPICE BSIM4.0 equations by also including the computation for second-order effects. The appendix shows the parts where myCACTI deviates from the published SPICE BSIM4.0 equations.
- The inclusion of an area model for pipelined nanometer caches. Currently, myCACTI only supports modeling of the cache delay and power dissipation. The current infrastructure for computing the area in CACTI and eCACTI can be utilized by updating it based on real and practical values for pipelined nanometer caches.

- For simulations using full-swing single-ended sensing, the static power dissipation due to sub-threshold leakage currents are skewed towards pessimism. It is being assumed that every bit-line is leaking in the same way as they leak for a low-swing differential sensing scheme. This is slightly inaccurate, as fully-discharged bitlines should not really be dissipating bitline static power, as there is no more charge in the bitline. Of course, there will still be some sort of bit-line leakage from the precharge devices to the active memory cells, but this should be an order of magnitude smaller because of the stacking effect due to the series PMOS and NMOS transistors.
- Inclusion of more optimal circuits after the sense-amplifier, including the output drivers and the tag comparators. Specifically, a better non-linear sizing algorithm should probably be implemented for the tag comparator, as we find that this typically proves to be a critical path simply because the transistors in the tag comparator are slow if a simple linear scaling algorithm is used to size them.
- Currently, myCACTI does not model the use of gated-VDD or gated-GND circuit, as these techniques are still in the prototype stages and are not yet widely accepted in commercial designs. myCACTI can be modified in a straightforward way to support these design techniques.
- myCACTI assumes a particular pipeline timing diagram as well as a specific pipeline latch implementation. To make the operation more flexible, multiple implementations can be supported, although this does require addition of revised analytical models specific to the particular implementation being considered.

CHAPTER 8 References

[Agarwal2003] A.Agarwal and K.Roy, "A noise tolerant cache design to reduce gate and subthreshold leakage in the nanometer region," ISLPED 2003.

[Amrutur1994] B.Amrutur and M.Horowitz, "Techniques to reduce power in fast wide memories," in Dig. Tech. Papers 1994 Symp. Low Power Electronics, 1994, pp. 92-93.

[Amrutur1998] B.Amrutur and M.Horowitz, "A replica technique for wordline and sense control in low-power SRAMs," IEEE J. Solid State Circuits, vol.33, pp.1208-1219, Aug.1998.

[Amrutur2000] B.Amrutur and M.Horowitz, "Speed and power scaling of SRAMs," IEEE J. Solid-State Circuits, vol.35, no.2, Feb 2000.

[Amrutur2001] B. Amrutur and M. Horowitz, "Fast low-power decoders for RAMs," IEEE J. Solid State Circuits, vol.36, no.10, pp.1506-1515, Oct.2001.

[Amrutur2001] B.Amrutur and M.Horowitz, "Fast low-power decoders for RAMs," IEEE JSSC, vol.36, no.10, Oct 2001.

[Anis2003] Mohab Anis, "Subthreshold leakage current: challenges and solutions," ICM 2003.

[Borkar2004] S.Borkar, "Circuit techniques for subthreshold leakage avoidance, control and tolerance," IEDM 2004.

[Burleson2004]W.P.Burleson et al., "Wave-pipelining: A tutorial and research survey," IEEE Trans. VLSI Systems, vol.6, no.5, Sep 1998.

[Butts2000] J. Adam Butts and Gurindar Sohi, "A static power model for architects," Proceedings of the 33rd MICRO (MICRO-33), December 2000.

[Cao2000] Y.Cao et al., "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," Proc. of CICC, pp.201-204, 2000.

[Chandrakasan1996] A.Chandrakasan, et al., "Design considerations and tools vor low voltage digital system design," in Proceedings of the 33rd Design Automation Conference, June 1996.

[Chappell1991] T.Chappell et al., "A 2-ns cycle, 3.8-ns access 512-kb CMOS ECL SRAM with a fully pipelined architecture," IEEE J.Solid State Circuits, vol.26, pp.1577-1585, No.v 1991.

[Chen1998] Z.Chen, M.Johnson, L.Wei and K.Roy, "Estimation of standby leakage power in CMOS circuits considering accurate modeling of transistor stacks," in Proc. Int.Symp.Low Power Electronics and Design, 1998.

[Cherkauer1995] B.S.Cherkauer and E.G.Friedman, "A unified design methodology for CMOS tapered buffers," IEEE J. Solid State Circuits, vol.3, pp.99-111, Mar. 1995.

[Clark2004] L.Clark, et al., "Managing standby and active mode leakage power in deep submicron design," in Proceedings of 2004 ISLPED.

[Doyle2002] B.Doyle, et al., "Transistor elements for 30nm physical gate lengths and beyond," Intel Technology Journal, Vol.6, Page(s): 42-54, May 2002.

[Evans1995] R.Evans and P.Franzon, Energy consumption modeling and optimization for SRAM's," IEEE JSSSC, May 1995.

[Gronowski1996] P.Gronowski et al., "A 433-MHz 64-b quad-ussue RISC microprocessor, "JSSC, vol.31, no.11, Nov.1996, pp.1687-1696.

[Gowan1998] M.Gowan, L.Biro and D.Jackson, "Power considerations in the design of the Alpha 21264 Microprocessor," 35th DAC.

[Halter1997] J.P.Halter and F.Najm, ,"A gate-level leakage power reduction method for ultra-low-power CMOS circuits," in Proc. IEEE Custom Integrated Circuits Conf., 1997.

[Hanson2001] H.Hanson, S.Keckler and D.Burger, “Static energy reduction techniques in microprocessor caches,” Tech Report TR2001-18, Computer Architecture and TEchnology Laboratory, Department of Computer Sciences, University of Texas, Austin.

[Hamzaoglu2002] F.Hamzaoglu and M.R.Stan, “Circuit-level techniques to control gate-leakage for sub-100nm CMOS,” ISLPED August 2002.

[Heald1993] R.Heald and J.Hoist, “A 6-ns cycle 256-kb cache memory and memory management unit,” IEEE J.Solid State Circuits, vol.28, pp.1078-1083, Nov.1993.

[Hirose1990] T.Hirose, et.al., “A 20-ns 4-Mb CMOS SRAM with hierarchical word decoding architecture,” IEEE J. Solid-State Circuits, vol. SC-25, no.5., pp.1068-1074, Oct.1990.

[Intel2006] Intel White Paper, “Increasing data center density while driving down power and cooling costs.”

[ITRS2003] "International Technology Roadmap for Semiconductors 2003 Edition," Semiconductor Industry Association, http://public.itrs.net.

[ITRS2005] "International Technology Roadmap for Semiconductors 2005 Edition," Semiconductor Industry Association, http://public.itrs.net.

[Jaeger1975] R.C. Jaeger, “Comments on “An optimized output stage for MOS integrated circuits”,” IEEE J.Solid State Circuits, vol. SC-10, pp.185-186, June 1975.

[Ghose1998] K. Ghose and M.Kamble, “Energy efficient cache organizations for superscalar processors,” Tech Report, SUNY.

[Kamble1997] M.Kamble and K.Ghose, “Analytical energy dissipation models for low power caches,” in Proc. 1997 ISLPED, Aug.1997.

[Kim2004] N.S.Kim, K.Flautner, D.Blaauw and T.Mudge, “Circuit and microarchitectural techniques for reducing cache leakage power,” IEEE Trans. on VLSI Systems, Feb.2004.

[Kursun2004] V.Kursun and E.G. Friedman, “Energy efficient dual threshold voltage dynamic circuits employing sleep switches to minimize subthreshold leakage.” ISCAS 2004.

[List1986] F.J.List, “The static noise margin of SRAM cells,” in ESSCIRC Dig. Tech. Papers, Sept.1986, pp. 16-18.

[Liu2001] W.Liu, “Mosfet Models for SPICE Simulation, including BSIM3V3 and BSIM4,” John Wiley & Sons, Copyright 2001.

[Lohstroh1983] J.Lohstroh, E.Seevinck, and J.de Groot, “Worst-case static noise margin criteria for logic circuits and their mathematical equivalence,” IEEE J. Solid-State Circuits, vol. SC-18, no.6, pp. 803-807, Dec. 1983.

[Mai1998] K. Mai et al., “Low-Power SRAM design using half-swing pulse-mode techniques,” IEEE J.Solid-State Circuits, vol.33, no.11, November 1998, pp.1659-1670.

[Mamidipaka2004] M.Mamidipaka and N.Dutt, “eCACTI: An Enhanced Power Estimation Model for On-chip Caches". Center for Embedded Computer Systems, Technical Report TR 04-28, Oct. 2004.

[Mamidipaka2004b] M.Mamidipaka et al., “Analytical models for leakage power estimation of memory array structures,” CODES+ISSS2004.

[Minato1987] O. Minato, et al., “A 42 ns 1Mb CMOS SRAM,” in ISSCC Dig. Tech. Papers, Feb 1987, pp.260-261.

[Montanaro1996] J.Montanaro et al., “A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor, JSSC, vol.31 no.11, Nov. 1996, pp.1703-1714.

[Mudge2004] N.S.Kim, K.Flautner, D.Blaauw and T.Mudge, “Single-Vdd and single-Vt super-drowsy techniques for low-leakage high-performance instruction caches,” in ISLPED 2004.

[Mulder1991] J.Mulder, N.Quach and M.Flynn, “An area model for on-chip memories and its application,” IEEE JSSC 1991.

[Mutoh1995] S.Mutoh, et al., “1V power supply high-speed digital circuit technology with multithreshold-voltage CMOS,” IEEE Journ. Solid State Circuits, Aug. 1995.

[Nakamura1997] K. Nakamura, “A 500-MHz 4-Mb CMOS pipeline-burst cache SRAM with point-to-point noise reduction coding I/O,” IEEE J. solid-State Circuits, vol.32, no.11, November 1997, pp.1758-1765.

[Nambu1998] H.Nambu, et.al., “A 1.8-ns access, 550MHz, 4.5-Mb CMOS SRAM,” IEEE J. Solid-State Circuits, vol.33, no.11, pp.1650-1658, Nov. 1998.

[Noda1998] K.Noda et al., “A 1.9 um2 loadless CMOS four-transistor SRAM cell in a 0.18um2 logic technology,” in IEDM Tech.Dig., 1998, pp.847-850.

[Osada2001] K.Osada et al., “Universal-Vdd 0.65-2.0V 32-kB cache using a voltage-adapted timing-generation scheme and a lithographically symmetrical cell,” IEEE J.Solid-State Circuits, vol.36 no.111, November 2001, pp.1738-1744.

[Park1998] H.C.Park et al., “A 833-Mb/s 2.5-V 4-Mb double-date-rate SRAM,” in 1998 IEEE Int. Solid State Circuits Conf. Dig. Tech. Papers, pp.356-357.

[Powell2000] M.Powell, S. Yang, B.Falsafi, K.Roy and T.N. Vijaykumar, “Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories,” ISLPED 2000.

[PTM] http://www.eas.asu.edu/~ptm

[PTM2006] Predictive technology model. http://www.eas.asu.edu/~ptm

[Rao2003a] R.M.Rao, J.L.Burns and R.B.Brown, “Circuit techniques for gate and subthreshold leakage minimization in future CMOS technologies,” ESSCIRC '03. Proceedings of the 29th European Solid-State Circuits Conference, 2003.

[Rao2003b] R.M.Rao, J.L.Burns and R.B.Brown, “Circuit techniques for gate and subthreshold leakage minimization in future CMOS technologies,” ESSCIRC '03. Proceedings of the 29th European Solid-State Circuits Conference, 2003.

[Reinman2000] G.Reinman and N.Jouppi, “CACTI 2.0: An integrated cache timing and power model,” WRL Research Report 2000/7, Feb.2000.

[Riedlinger2002] Riedlinger, R., Grutkowski, T., “The high-bandwidth 256 kB 2nd level cache on an Itanium microprocessor,” ISSCC 2002.

[|Sasaki1990] K. Sasaki et al, “A 23-ns 4-Mb CMOS SRAM with 0.2-uA standby current,” IEEE J. Solid State Circuits, vol.25 no.5, pp.1075-1081, Oct.1990.

[Sasaki1988] K.Sasaki, et.al., “A 15-ns 1-Mbit CMOS SRAM,” IEEE J. Solid-State Circuits, vol 23, no.5., pp.1067-1072, Oct.1988.

[Sasaki1989] K. Sasaki et al., “A 9-ns 1-Mbit CMOS SRAM,” IEEE J. Solid State Circuits, vol.24, no.5, pp.1219-1225, Oct.1989.

[Schuster1986] S. Schuster et al., “A 15-ns CMOS 64K RAM,” IEEE J. Solid-State Circuits, vol. SC-21, no.5, October 1986, pp.704-712.

[Shivakumar2001] P.Shivakumar and N.Jouppi , “CACTI 3.0: An integrated cache timing, power and area model,” WRL Research Report 2001/2, Aug.2001.

[Skadron2003] K.Skadron et al., “Temperature-aware microarchitecture,” ISCA 2003.

[Sutherland1991] I.E. Sutherland and R.F.Sproull, “Logical effort: designing for speed on the back of an envelope,” Advanced Res. VLSI, pp. 1-16, 1991.

[Sutherland1999] I.E. Sutherland, et.al., Logical Effort: Designing fast CMOS circuits, 1st ed. San Mateo, CA: Morgan Kauffman, 1999.

[TechReport2006] The Tech Report. http://techreport.com/cpu/.

[Uchiyama1991] K.Uchiyama et al., “Design of a second-level cache chip for shared-bus multimicroprocessor systems,” IEEE J. Solid State Circuits, vol.26, no.4, pp.566-571, Apr. 1991.

[Weiss2002] D.Weiss, J.Wuu and V.Chin, "The on-chip 3-MB subarray-based third-level cache of an Itanium microprocessor," in Journ. Solid-State Circuits, Nov.2002.[Anis2003] Mohab Anis, “Subthreshold leakage current: challenges and solutions,” ICM 2003.

[Weiss2002] Weiss, D., Wuu, J.J., Chin, V., “An on-chip 3MB subarray-based 3rd levl cache on an Itanium microprocessor,” ISSCC 2002.

[Wilton1996] S.J.Wilton and N.P.Jouppi, “CACTI: An enhanced cache access and cycle time model,” IEEE JSSC, vol.31, no.5, May 1996.

[Yang2003] Se-Hyun Yang and Babak Falsafi, “Near-optimal precharging in high-performance nanoscale CMOS caches,” Proc. 36th Int. Symp. on Microarchitecture, 2003.

[Ye1998] Y.Ye, S.Borkar and V.De, “A new technique for standby leakage reduction in high-performance circuits,” in Symp.VLSI Circuits Dig.Tech.Papers, 1998.

[Yeo2000] Y.C.Yeo et al., “Direct tunneling gate leakage current in transistorswith ultrathin nsilicon nitride gate dielectric,” IEEE Electron Device Letters, Nov. 2000.

[Yoshimoto1983] M. Yoshimoto, et.al., “A divided word-line structure in the static RAM and its application to a 64K full CMOS RAM,” IEEE J. Solid-State Circuit, vol. SC-18, no.5, pp.479-485, Oct. 1983.

Appendix A. SPICE BSIM4.0 Equations

(Eq. G-1)

$$L_{eff} = L - 2 \cdot \mathrm{LINT} - 2 \cdot \Delta L_{\mathrm{geometry}};$$

(Eq. G-8)

$$L_{eff,\mathrm{CV}} = L - 2 \cdot \mathrm{DLC} - 2 \cdot \Delta L_{\mathrm{geometry,CV}};$$

(Eq. G-10)

$$W_{eff,\mathrm{CJ}} = \frac{W}{\mathrm{NF}} - 2 \cdot \mathrm{DWJ} - 2 \cdot \Delta W_{\mathrm{geometry,CV}};$$

(Eq. G-13)

$$q = 1.6 \times 10^{-19}.$$

(Eq. G-14a)

$$\epsilon_s = 8.85 \times 10^{-12} \cdot 11.7055.$$

(Eq. G-14b

$$\epsilon_{ox} = 8.85 \times 10^{-12} \cdot \mathrm{EPSROX}.$$

(Eq. G-15)

$$n_i = 1.45 \times 10^{10} \cdot \frac{\mathrm{TNOM}}{300.15} \cdot \sqrt{\frac{\mathrm{TNOM}}{300.15}} \cdot \exp\left[21.5565981 - \frac{(\mathrm{TNOM})}{2 \cdot k(\mathrm{TNOM} + 273.15)}\right]$$

(Eq. G-16)

$$k = 1.3806226 \times 10^{-23}.$$

(Eq. G-17)

$$2\phi_f = \frac{k[\mathrm{TNOM} + 273.15]}{q} \ln\left(\frac{\mathrm{NDEP}}{n_i}\right) + 0.4 + \mathrm{PHIN}.$$

(Eq. G-20)

$$C'_{ox,\mathrm{e}} = \frac{\epsilon_{ox}}{\mathrm{TOXE}}.$$

(Eq. G-21)

$$C'_{ox,\mathrm{p}} = \frac{\epsilon_{ox}}{\mathrm{TOXP}}.$$

(Eq. G-22)

$$C_{ox,\mathrm{e}} = W_{eff,\mathrm{CV}} L_{eff,\mathrm{CV}} \times C'_{ox,\mathrm{e}} \times \mathrm{NF}.$$

(Eq. G-23)

$$T = \begin{cases} \mathrm{TNOM} + 273.15 & \text{if } T_{device} \text{ is not given;} \\ T_{device} + 273.15 & \text{if } T_{device} \text{ is given.} \end{cases}$$

(Eq. G-25)

$$V_{FB} = \delta_{NP} \cdot \mathrm{VTHO} - 2\phi_f - \mathrm{K1}\sqrt{2\phi_f},$$

(Eq. G-26)

$$V_{BS,\mathrm{eff}} = v_{bc} + \frac{(V_{BS} - v_{bc} - \delta_1) + \sqrt{(V_{BS} - v_{bc} - \delta_1)^2 - 4\delta_1 v_{bc}}}{2}; \qquad \delta_1 = 0.001,$$

(Eq. G-27)

$$V_x = 0.9 \times \left(2\phi_f - \frac{\mathrm{K1}^2}{4\,\mathrm{K2}^2}\right),$$

(Eq. G-31)

$$\begin{aligned} V_T = {} & \mathrm{VTHO} + \delta_{NP}(\Delta V_{T,\mathrm{body\_effect}} - \Delta V_{T,\mathrm{charge\_sharing}} - \Delta V_{T,\mathrm{DIBL}} \\ & + \Delta V_{T,\mathrm{reverse\_short\_channel}} + \Delta V_{T,\mathrm{narrow\_width}} + \Delta V_{T,\mathrm{small,size}} \\ & - \Delta V_{T,\mathrm{pocket\_implant}}). \end{aligned}$$

(Eq. G-32)

$$\begin{aligned} \Delta V_{T,\mathrm{body\_effect}} = {} & \left[\mathrm{K1}\frac{\mathrm{TOXE}}{\mathrm{TOXM}}\sqrt{2\phi_f - V_{BS,\mathrm{eff}}} - \mathrm{K1}\cdot\sqrt{2\phi_f}\right] \times \sqrt{1 + \frac{\mathrm{LPEB}}{L_{eff}}} \\ & - \mathrm{K2}\frac{\mathrm{TOXE}}{\mathrm{TOXM}} V_{BS,\mathrm{eff}}. \end{aligned}$$

(Eq. G-33)

$$\Delta V_{T,\mathrm{charage\_sharing}} = \mathrm{DVT0}\frac{0.5}{\cosh(\mathrm{DVT1} \times L_{eff}/L_t) - 1}(V_{bi} - 2\phi_f).$$

(Eq. G-34)

$$\begin{aligned} \Delta V_{T\,\mathrm{DIBL}} = {} & \left[\exp\left(-\mathrm{DSUB}\frac{L_{eff}}{2L_{t0}}\right) + 2\exp\left(-\mathrm{DSUB}\frac{L_{eff}}{L_{t0}}\right)\right] \\ & \times (\mathrm{ETA0} + \mathrm{ETAB} \cdot V_{BS,\mathrm{eff}}) \times V_{DS} \end{aligned}$$

(Eq. G-44)

$$Tmp = (\mathrm{ETA0} + \mathrm{ETAB} \cdot V_{BS,\mathrm{eff}}).$$

(Eq. G-45)

$$\Delta V_{T,\mathrm{DIBL}} = \begin{cases} \left[\exp\left(-\mathrm{DSUB}\dfrac{L_{eff}}{2L_{t0}}\right) + 2\exp\left(-\mathrm{DSUB}\dfrac{L_{eff}}{L_{t0}}\right)\right] V_{DS} \times Tmp \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if } Tmp \geq 10^{-4}; \\ \left[\exp\left(-\mathrm{DSUB}\dfrac{L_{eff}}{2L_{t0}}\right) + 2\left(-\mathrm{DSUB}\dfrac{L_{eff}}{L_{t0}}\right)\right] V_{DS} \times \dfrac{2\times 10^{-4} - Tmp}{3 - 2\times 10^{4}\cdot Tmp} \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if } Tmp < 10^{-4}. \end{cases}$$

(Eq. G-35)

$$\Delta V_{T,\mathrm{reverse\_shortchannel}} = \mathrm{K1}\cdot\frac{\mathrm{TOXE}}{\mathrm{TOXM}}\cdot\left(\sqrt{1+\frac{\mathrm{LPE0}}{L_{eff}}}-1\right)\sqrt{2\phi_f}.$$

(Eq. G-36)

$$\Delta V_{T,\mathrm{narrow\_width}} = (\mathrm{K3} + \mathrm{K3B}\cdot V_{BS,\mathrm{eff}})\frac{\mathrm{TOXE}}{W_{eff}+\mathrm{W0}}2\phi_f.$$

(Eq. G-37)

$$\Delta V_{T,\mathrm{small\_size}} = \mathrm{DVT0W}\left[\exp\left(-\mathrm{DVT1W}\frac{W_{eff}L_{eff}}{2L_{tw}}\right) + 2\exp\left(-\mathrm{DVT1W}\frac{W_{eff}L_{eff}}{L_{tw}}\right)\right](V_{bi}-2\phi_f).$$

(Eq. G-38)

$$V_{bi} = \frac{k[\mathrm{TNOM}+273.15]}{q}\ln\left(\frac{\mathrm{NDEP}\cdot\mathrm{NSD}}{n_i^2}\right).$$

(Eq. G-39)

$$L_{t0} = \sqrt{\frac{\epsilon_s X_{dep,0}}{C'_{ox,\mathrm{e}}}}.$$

(Eq. G-42)

$$X_{dep} = \sqrt{\frac{2\epsilon_s(2\phi_f - V_{BS,\mathrm{eff}})}{q\,\mathrm{NDEP}}}.$$

(Eq. G-43)

$$X_{dep,0} = \sqrt{\frac{2\epsilon_s(2\phi_f)}{q\,\mathrm{NDEP}}}.$$

(Eq. G-46)

$$V_{poly} = \frac{q \in_s \text{NGATE} C_{oxe}'^2 \cdot 10^6}{2} \left[\sqrt{1 + \frac{2(V_{GS} - V_{FB} - 2\phi_f)}{q \in_s \text{NGATE} C_{oxe}'^2 \cdot 10^6}} - 1 \right]^2.$$

(Eq. G47-)

$$V_{poly,\text{eff}} = 1.12 - \tfrac{1}{2}(1.12 - V_{poly} - \delta + \sqrt{(1.12 - V_{poly} - \delta)^2 + 4 \cdot \delta \cdot 1.12})$$

(δ fixed at 0.05).

(Eq. G-48)

$$V_{GS,\text{eff}} = V_{GS} - V_{poly,\text{eff}}.$$

(Eq. G-54)

$$n = 1 + \text{FACTOR} \cdot \frac{C_{dep}'}{C_{ox,e}'} + \frac{\text{CIT}}{C_{ox,e}'} + \frac{\text{CDSC} + \text{CDSCD} \cdot V_{DS} + \text{CDSCB} \cdot V_{BS,\text{eff}}}{C_{ox,e}'} \times \frac{0.5}{\cosh(\text{DVT1} \times L_{eff}/L_t) - 1}$$

(Eq. G-55)

$$Tmp = \text{NFACTOR} \cdot \frac{C_{dep}'}{C_{ox,e}'} + \frac{\text{CIT}}{C_{ox,e}'} + \frac{\text{CDSC} + \text{CDSCD} \cdot V_{DS} + \text{CDSCB} \cdot V_{BS,\text{eff}}}{C_{ox,e}'} \times \frac{0.5}{\cosh(\text{DVT1} \times L_{eff}/L_t) - 1}.$$

(Eq. G-56)

$$n = \begin{cases} 1 + Tmp & \text{if } Tmp \geq -0.5; \\ \dfrac{1 + 3 \cdot Tmp}{3 + 8 \cdot Tmp} & \text{if } Tmp < -0.5. \end{cases}$$

(Eq. G-57)

$$\Delta V_{T,\text{pocket\_implant}} = n \frac{kT}{q} \ln \left[\frac{L_{eff}}{L_{eff} + \text{DVTP0}(1 + \exp(-\text{DVTP1} \cdot V_{DS}))} \right].$$

(Eq. G-59)

$$Tmp = \left[\text{UA} \left(\frac{V_{GST,\text{eff}} + 2 \cdot V_T}{\text{TOXE}} \right) + \text{UB} \left(\frac{V_{GST,\text{eff}} + 2 \cdot V_T}{\text{TOXE}} \right)^2 \right] (1 + \text{UC} \cdot V_{BS,\text{eff}}).$$

(Eq. G-62)

$$\mu_{eff} = \frac{\mathrm{U0}}{Denominator},$$

(Eq. G-63)

$$Denominator = \begin{cases} 1 + Tmp & if\ Tmp \geq -0.8; \\ \dfrac{0.6 + Tmp}{7 + 10 \cdot Tmp} & if\ Tmp < -0.8. \end{cases}$$

(Eq. G-66)

$$R_{DS} = \begin{cases} \dfrac{\mathrm{RDSWMIN} + \mathrm{RDSW} \times \frac{1}{2}\left[Tmp + \sqrt{Tmp^2 + 0.01}\right]}{\left(10^6 \times W_{eff,\mathrm{CJ}}\right)^{\mathrm{WR}}} & if\ \mathrm{RDSMOD} \neq 1; \\ 0 & if\ \mathrm{RDSMOD} = 1; \end{cases}$$

(Eq. G-67)

$$Tmp = \frac{1}{1 + \mathrm{PRWG} \cdot V_{GST,\mathrm{eff}}} + \mathrm{PRWB}\left(\sqrt{2\phi_f - V_{BS,\mathrm{eff}}} - \sqrt{2\phi_f}\right).$$

(Eq. G-68)

$$A_{Bulk} = \left\{1 - \frac{dV_{T,\mathrm{Long\_L}}}{dV_{BS,\mathrm{eff}}} \times \left[\frac{\mathrm{A0} \cdot L_{eff}}{L_{eff} + 2\sqrt{\mathrm{XJ} \cdot X_{dep}}} \times \left(1 - \mathrm{AGS} \cdot V_{GST,\mathrm{eff}}\left(\frac{L_{eff}}{L_{eff} + 2\sqrt{\mathrm{XJ} \cdot X_{dep}}}\right)^2\right) + \frac{\mathrm{B0}}{W_{eff} + \mathrm{B1}}\right]\right\} \times \frac{1}{1 + \mathrm{KETA} \cdot V_{BS,\mathrm{eff}}}$$

(Eq. G-69)

$$\frac{dV_{T,\mathrm{long\_L}}}{dV_{BS,\mathrm{eff}}} = \sqrt{1 + \frac{\mathrm{LPEB}}{L_{eff}}} \times \frac{\mathrm{K1}}{2\sqrt{2\phi_f - V_{BS,\mathrm{eff}}}} \frac{\mathrm{TOXE}}{\mathrm{TOXM}} + \mathrm{K2}\frac{\mathrm{TOXE}}{\mathrm{TOXM}} - \mathrm{K3} \times \frac{\mathrm{TOXE}}{W_{eff} + \mathrm{W0}} 2\phi_f.$$

(Eq. G-70)

$$A_{Bulk,\mathrm{tmp}} = \left\{1 - \frac{dV_{T,\mathrm{long\_L}}}{dV_{BS,\mathrm{eff}}} \times \left[\frac{\mathrm{A0} \cdot L_{eff}}{L_{eff} + 2\sqrt{\mathrm{XJ} \cdot X_{dep}}} \times \left(1 - \mathrm{AGS} \cdot V_{GST,\mathrm{eff}}\left(\frac{L_{eff}}{L_{eff} + 2\sqrt{\mathrm{XJ} \cdot X_{dep}}}\right)^2\right) + \frac{\mathrm{B0}}{W_{eff} + \mathrm{B1}}\right]\right\}.$$

(Eq. G-71)

$$A_{Bulk} = \begin{cases} A_{Bulk,\mathrm{tmp}} \times \dfrac{1}{1 + \mathrm{KETA} \cdot V_{BS,\mathrm{eff}}} & \text{if } A_{Bulk,\mathrm{tmp}} \geq 0.1; \\ \dfrac{0.2 - A_{Bulk,\mathrm{tmp}}}{3 - 20 \cdot A_{Bulk,\mathrm{tmp}}} \times \dfrac{1}{1 + \mathrm{KETA} \cdot V_{BS,\mathrm{eff}}} & \text{if } A_{Bulk,\mathrm{tmp}} < 0.1. \end{cases}$$

(Eq. G-73)

$$\varepsilon_{sat} = \frac{2 \cdot \mathrm{VSAT}}{\mu_{\mathrm{eff}}}.$$

(Eq. G-74)

If $R_{DS} = 0$, then

$$V_{DS,\mathrm{sat}} = \frac{\varepsilon_{sat} L_{eff} \cdot (V_{GST,\mathrm{eff}} + 2kT/q)}{A_{bulk} \varepsilon_{sat} L_{eff} + (V_{GST,\mathrm{eff}} + 2kT/q)}.$$

(Eq. G-75)

If $R_{DS} \neq 0$, then

$$V_{DS,\mathrm{sat}} = \frac{-b - \sqrt{b^2 - 4ac}}{2a};$$

(Eq. G-76)

$$a = A_{bulk}^2 W_{eff} \mathrm{VSAT} C'_{ox,e} R_{DS} + \left(\frac{1}{\lambda} - 1\right) A_{bulk};$$

(Eq. G-77)

$$b = -\left[\left(V_{GST,\mathrm{eff}} + \frac{2kT}{q}\right) \cdot \left(\frac{2}{\lambda} - 1\right) + A_{bulk} \varepsilon_{sat} L_{eff} + 3A_{bulk}\left(V_{GST,\mathrm{eff}} + \frac{2kT}{q}\right) W_{eff} \mathrm{VSAT} C'_{ox,e} R_{DS}\right];$$

(Eq. G-78)

$$c = \left(V_{GST,\mathrm{eff}} + \frac{2kT}{q}\right) \varepsilon_{sat} L_{eff} + 2\left(V_{GST,\mathrm{eff}} + \frac{2kT}{q}\right)^2 W_{eff} \mathrm{VSAT} C'_{ox,e} R_{DS};$$

(Eq. G-79)

If user-inputted `A2` < 0.01, then `A2` is set to 0.01.

(Eq. G-80)

If user-inputted `A2` > 1, then `A2` is set to 1, and `A1` is set to 0.

(Eq. G-81)

$$Tmp = (1 - \mathrm{A2}) - \frac{1}{2}\left(1 - \mathrm{A2} - \mathrm{A1} \cdot V_{GST,\mathrm{eff}} - \delta + \sqrt{(1 - \mathrm{A2} - \mathrm{A1} \cdot V_{GST,\mathrm{eff}} - \delta)^2 + 4 \cdot \delta \cdot (1 - \mathrm{A2})}\right) \quad \delta = 0.001.$$

(Eq. G-82)

$$\lambda = \mathrm{A2} + Tmp.$$

(Eq. G-83)

$$Tmp = \mathrm{A2} - \frac{1}{2}\left(\mathrm{A2} - \mathrm{A1}' \cdot V_{GST,\mathrm{eff}} - \delta + \sqrt{(\mathrm{A2} - \mathrm{A1}' \cdot V_{GST,\mathrm{eff}} - \delta)^2 + 4\delta \mathrm{A2}}\right) \qquad \delta = 0.001,$$

(Eq. G-84)

$$\lambda = \mathrm{A2} - Tmp.$$

(Eq. G-85)

$$V_{DS,\mathrm{eff}} = V_{DS,\mathrm{sat}} - \tfrac{1}{2}\left(V_{Ds,\mathrm{sat}} - V_{DS} - \mathrm{DELTA} + \sqrt{(V_{Ds,\mathrm{sat}} - V_{DS} - \mathrm{DELTA})^2 + 4 \cdot \mathrm{DELTA} \cdot V_{DS,\mathrm{sat}}}\right).$$

(Eq. G-86)

$$C'_{ox,\mathrm{IV}} = \frac{\epsilon_{ox}}{\mathrm{TOXP}} // \frac{\epsilon_s}{X_{DC,\mathrm{IV}}},$$

(Eq. G-87)

$$X_{DC,\mathrm{IV}} = \begin{cases} \dfrac{1.9 \times 10^{-9}}{\left[1 + \dfrac{V_{GST,\mathrm{eff}} + 4(\mathrm{VTH0} - V_{FB} - 2\phi_f)}{2 \times 10^8 \cdot \mathrm{TOXP}}\right]^{0.7}} & \text{if } (\mathrm{VTH0} - V_{FB} - 2\phi_f) \geq 0; \\ \dfrac{1.9 \times 10^{-9}}{\left[1 + \dfrac{V_{GST,\mathrm{eff}}}{2 \times 10^8 \cdot \mathrm{TOXP}}\right]^{0.7}} & \text{if } (\mathrm{VTH0} - V_{FB} - 2\phi_f) < 0. \end{cases}$$

(Eq. G-88)

$$I_{DS} = \frac{I_{DS,0}}{1 + \dfrac{R_{DS} I_{DS,0}}{V_{Ds,\text{eff}}}} \left(1 + \frac{1}{C_{clm}} \ln \frac{V_A}{V_{A,\text{sat}}}\right) \times \left(1 + \frac{V_{DS} - V_{DS,\text{eff}}}{V_{A,\text{DIBL}}}\right) \times \left(1 + \frac{V_{DS} - V_{DS,\text{eff}}}{V_{A,\text{DITS}}}\right) \times \left(1 + \frac{V_{DS} - V_{DS,\text{eff}}}{V_{A,\text{SCBE}}}\right) \times \text{NF},$$

(Eq. G-89)

$$I_{DS,0} = \frac{W_{eff} \mu_{eff} C'_{ox,\text{IV}} V_{GST,\text{eff}}}{L_{eff}[1 + V_{DS,\text{eff}}/(\varepsilon_{sat} L_{eff})]} \left[1 - \frac{A_{bulk} V_{DS,\text{eff}}}{2(V_{GST,\text{eff}} + 2kT/q)}\right] V_{DS,\text{eff}}.$$

(Eq. G-90)

$$V_A = V_{A,\text{sat}} + V_{A,\text{CLM}}.$$

(Eq. G-91)

$$V_{A,\text{sat}} = \frac{\varepsilon_{sat} L_{eff} + V_{DS,\text{sat}} + 2R_{DS}\text{VSAT}C'_{ox,e} W_{eff} V_{GST,\text{eff}}}{2/\lambda - 1 + R_{DS}\text{VSAT}C'_{ox,e} W_{eff} A_{bulk}} \times \left[1 - \frac{A_{bulk} V_{DS,\text{sat}}}{2(V_{GST,\text{eff}} + 2kT/q)}\right].$$

(Eq. G-92)

$$F_p = \begin{cases} \left(1 + \text{FPROUT} \cdot \dfrac{\sqrt{L_{eff}}}{V_{GST,\text{eff}} + 2kT/q}\right)^{-1} & \text{if FPROUT} > 0; \\ 1 & \text{if FPROUT} \le 0. \end{cases}$$

(Eq. G-93)

$$F_{VG} = \begin{cases} 1 + \dfrac{\text{PVAG}\, V_{GST,\text{eff}}}{\varepsilon_{sat} L_{eff}} & \text{if } \dfrac{\text{PVAG} V_{GST,\text{eff}}}{\varepsilon_{sat} L_{eff}} > -0.9; \\ \dfrac{0.8 + \text{PVAG} \cdot V_{GST,\text{eff}}/(\varepsilon_{sat} L_{eff})}{17 + 20 \cdot \text{PVAG} \cdot V_{GST,\text{eff}}/(\varepsilon_{sat} L_{eff})} & \text{if otherwise.} \end{cases}$$

(Eq. G-94)

$$Tmp = \frac{F_p}{\text{PCLM}\, L_{itl}} F_{VG} \times \left(1 + \frac{R_{DS} I_{DS,0}}{V_{DS,\text{eff}}}\right) \times \left(L_{eff} + \frac{V_{DS,\text{sat}}}{\varepsilon_{sat}}\right).$$

(Eq. G-95)

$$C_{clm} = \begin{cases} Tmp & \text{if PCLM} > 0 \text{ and } (V_{DS} - V_{DS,\text{eff}}) > 10^{-10}; \\ 5.834617425 \times 10^{14} & \text{if otherwise.} \end{cases}$$

(Eq. G-96)

$$V_{A,\mathrm{CLM}} = \begin{cases} C_{clm}(V_{DS} - V_{DS,\mathrm{eff}}) & \text{if PCLM} > 0 \text{ and } (V_{DS} - V_{DS,\mathrm{eff}}) > 10^{-10} \\ 5.834617425 \times 10^{14} & \text{if otherwise} \end{cases}$$

(Eq. G-97)

$$\theta_{rout} = \mathrm{PDIBLC1} \times \left[\exp\left(-\mathrm{DROUT}\frac{L_{eff}}{2L_{t0}}\right) + 2\exp\left(-\mathrm{DROUT}\frac{L_{eff}}{L_{to}}\right)\right] + \mathrm{PDIBLC2}.$$

(Eq. G-98)

$$V_{A,\mathrm{DIBL}} = \begin{cases} \dfrac{(V_{GST,\mathrm{eff}} + 2kT/q)}{\theta_{rout}(1 + \mathrm{PDIBLCB} \cdot V_{BS,\mathrm{eff}})} \times F_{VG} & \\ \left[1 - \dfrac{A_{bulk}V_{DS,sat}}{A_{bulk}V_{DS,sat} + V_{GST,eff} + 2kT/q}\right] & \text{if } \theta_{rout} \geq 0; \\ 5.834617425 \times 10^{14} & \text{if } \theta_{rout} \leq 0. \end{cases}$$

(Eq. G-100)

$$V_{A,\mathrm{DITS}} = \begin{cases} \dfrac{F_P}{\mathrm{PDITS}}[1 + (1 + \mathrm{PDITSL} \cdot L_{eff}) \times \exp(\mathrm{PDITSD} \cdot V_{DS})] & \\ & \text{if PDITS} > 0; \\ 5.834617425 \times 10^{14} & \text{if otherwise.} \end{cases}$$

(Eq. G-101)

$$V_{A,\mathrm{SCBE}} = \begin{cases} \dfrac{L_{eff}}{\mathrm{PSCBE2}}\exp\left(\dfrac{\mathrm{PSCBE1}L_{itl}}{V_{DS} - V_{DS,\mathrm{eff}}}\right) & \text{if PSCBE2} > 0; \\ 5.834617425 \times 10^{14} & \text{if otherwise.} \end{cases}$$

(Eq. G-102)

$$L_{itl} = \sqrt{\frac{\epsilon_s}{\epsilon_{ox}}} \times \mathrm{TOXE} \cdot \mathrm{XJ}.$$

(Eq. G-248)

$$I_{gidl} = \mathrm{NF} \times \mathrm{AGIDL} \cdot W_{eff,\mathrm{CJ}}\left[\frac{V_{DS} - V_{GS,\mathrm{eff}} - \mathrm{EGIDL}}{3 \cdot \mathrm{TOXE}}\right] \times \exp\left(\frac{-3 \cdot \mathrm{TOXE} \times \mathrm{BGIDL}}{V_{DS} - V_{GS,\mathrm{eff}} - \mathrm{EGIDL}}\right) \times \frac{V_{DB}^3}{\mathrm{CGIDL} + V_{DB}^3}.$$

(Eq. G-249)

$$V_{FB,CV} = V_{fb,zb} = V_T(V_{GS} = V_{DS} = V_{BS} = 0) - 2\phi_f - \mathrm{K1}\sqrt{2\phi_f}. \qquad \text{(G-249)}$$

(Eq. G-250)

$$\sqrt{\phi_s, \mathrm{dep}} = \begin{cases} \left[\sqrt{\frac{1}{4}\left(\mathrm{K1}\frac{\mathrm{TOXE}}{\mathrm{TOXM}}\right)^2 + Tmp} - \frac{1}{2}\left(\mathrm{K1}\frac{\mathrm{TOXE}}{\mathrm{TOXM}}\right)\right] \\ Tmp/\left(\mathrm{K1}\frac{\mathrm{TOXE}}{\mathrm{TOXM}}\right) \end{cases}$$

(Eq. G-251)

$$Tmp = V_{GS,\mathrm{eff}} - V_{FB,\mathrm{effCV}} - V_{BS,\mathrm{eff}} - V_{GST,\mathrm{eff}}.$$

(Eq. G-252)

$$V_{GB,\mathrm{eff\_tunnel}} = V_{GS,\mathrm{eff}} - V_{BS,\mathrm{eff}}.$$

(Eq. G-254

$$V_{ox,\mathrm{acc}} = V_{FB,CV} - V_{FB,\mathrm{eff\_tunnel}};$$

(Eq. G-255)

$$V_{ox,\mathrm{depinv}} = \left(\mathrm{K1}\frac{\mathrm{TOXE}}{\mathrm{TOXM}}\right)\sqrt{\phi_{s,\mathrm{dep}}} + V_{GST,\mathrm{eff}}.$$

(Eq. G-256)

$$I_{gb,\mathrm{tunnel}} = \begin{cases} I_{gb,\mathrm{acc}} + I_{gb,\mathrm{inv}} & \text{if IGBMOD} \neq 0; \\ 0 & \text{if IGBMOD} = 0; \end{cases} \qquad \text{(G-256)}$$

(Eq. G-250)

$$\begin{aligned} I_{gb,\mathrm{acc}} = {} & \mathrm{NF}W_{eff}L_{eff}\frac{\mathrm{A1}}{\mathrm{TOXE}^2}\left(\frac{\mathrm{TOXREF}}{\mathrm{TOXE}}\right)^{\mathrm{NTOX}} V_{GB,\mathrm{eff\_tunnel}} \\ & \times \mathrm{NIGBACC}\frac{kT}{q}\ln\left[1 + \exp\left(-\frac{q(V_{GB,\mathrm{eff\_tunnel}} - V_{FB,CV})}{kT \cdot \mathrm{NIGBACC}}\right)\right] \\ & \times \exp[-\mathrm{B1} \cdot \mathrm{TOXE}(\mathrm{AIGBACC} - \mathrm{BIGBACC}V_{ox,\mathrm{acc}}) \\ & \cdot (1 + \mathrm{CIGBACC}\ V_{ox,\mathrm{acc}})]. \end{aligned}$$

(Eq. G-251)

$$\mathrm{A1} = 4.97232 \times 10^{-7}\frac{\mathrm{A}}{\mathrm{V}^2}; \qquad \mathrm{B1} = 7.45669 \times 10^{11}\frac{\mathrm{g}^{1/2}}{\mathrm{F}^{1/2} \cdot \mathrm{s}}.$$

(Eq. G-252)

$$I_{gb,\text{inv}} = \frac{\text{NF}\, W_{eff} L_{eff} \text{A2}}{\text{TOXE}^2}\left(\frac{\text{TOXREF}}{\text{TOXE}}\right)^{\text{NTOX}} V_{GB,\text{eff\_tunnel}} \times \text{NIGBINV}\frac{kT}{q}\ln\left[1+\exp\left(-\frac{q(V_{ox,\text{depinv}} - \text{EIGBINV})}{kT\cdot\text{NIGBINV}}\right)\right] \times \exp[-\text{B2}\cdot\text{TOXE}(\text{AIGBINV} - \text{BIGBINV}V_{ox,\text{depinv}}) \cdot (1+\text{CIGBINV}V_{ox,\text{depinv}})].$$

(Eq. G-253)

$$\text{A2} = 3.75956\times10^{-7}\frac{\text{A}}{\text{V}^2}; \qquad \text{B2} = 9.82222\times10^{11}\frac{\text{g}^{1/2}}{\text{F}^{1/2}\cdot\text{s}}.$$

(Eq. G-254)

$$I_{gcs,\text{tunnel}} = \begin{cases} I_{gc}\times\dfrac{-1+\text{PIGCD}\cdot V_{DS}+\exp(-\text{PIGCD}\cdot V_{DS})+10^{-4}}{(\text{PIGCD}\cdot V_{DS})^2+2\times10^{-4}} & \text{if IGCMOD} \neq 0; \\ 0 & \text{if IGCMOD} = 0. \end{cases}$$

(Eq. G-255)

$$I_{gcd,\text{tunnel}} = \begin{cases} I_{gc}\times\dfrac{1-(\text{PIGCD}\cdot V_{DS}+1)\cdot\exp(-\text{PIGCD}\cdot V_{DS})+10^{-4}}{(\text{PIGCD}\cdot V_{DS})^2+2\times10^{-4}} & \text{if IGCMOD} \neq 0; \\ 0 & \text{if IGCMOD} = 0. \end{cases}$$

(Eq. G-256)

$$I_{gs,\text{tunnel}} = \begin{cases} I_{gs} & \text{if IGCMOD} \neq 0; \\ 0 & \text{if IGCMOD} = 0. \end{cases}$$

(Eq. G-257)

$$I_{gd,\text{tunnel}} = \begin{cases} I_{gd} & \text{if IGCMOD} \neq 0; \\ 0 & \text{if IGCMOD} = 0; \end{cases}$$

(Eq. G-258)

$$I_{gc} = \text{NF}\, W_{eff} L_{eff}\frac{\text{A3}}{\text{TOXE}^2}\left(\frac{\text{TOXREF}}{\text{TOXE}}\right)^{\text{NTOX}} V_{GS,\text{eff}} \times \text{NIGC}\frac{kT}{q}\ln\left[1+\exp\left(+\frac{q(V_{GS,\text{eff}} - \text{VTH0})}{kT\cdot\text{NIGC}}\right)\right] \times \exp[-\text{B3}\cdot\text{TOXE}(\text{AIGC} - \text{BIGC}V_{ox,\text{depinv}})\cdot(1+\text{CIGC}V_{ox,\text{depinv}})].$$

(Eq. G-259)

$$I_{gs} = \mathrm{NF}\, W_{eff}\, \mathrm{DLCIG} \frac{\mathrm{A3}}{(\mathrm{TOXE} \cdot \mathrm{POXEDGE})^2} \left(\frac{\mathrm{TOXREF}}{\mathrm{TOXE} \cdot \mathrm{POXEDGE}}\right)^{\mathrm{NTOX}} V_{GS} \times V'_{GS}$$
$$\times \exp[-\mathrm{B3} \cdot \mathrm{TOXE} \cdot \mathrm{POXEDGE}(\mathrm{AIGSD} - \mathrm{BIGSD}\, V'_{GS})$$
$$\cdot (1 + \mathrm{CIGSD} V'_{GS})].$$

(Eq. G-260)

$$V'_{GS} = \sqrt{(V_{GS} - V_{FB,\mathrm{SD}})^2 + 10^{-4}}.$$

(Eq. G-261)

$$I_{gd} = \mathrm{NF} W_{eff}\, \mathrm{DLCIG} \frac{\mathrm{A3}}{(\mathrm{TOXE} \cdot \mathrm{POXEDGE})^2} \left(\frac{\mathrm{TOXREF}}{\mathrm{TOXE} \cdot \mathrm{POXEDGE}}\right)^{\mathrm{NTOX}} V_{GD} \times V'_{GD}$$
$$\times \exp[-\mathrm{B3} \cdot \mathrm{TOXE} \cdot \mathrm{POXEDGE}(\mathrm{AIGSD} - \mathrm{BIGSD}\, V'_{GD})$$
$$\cdot (1 + \mathrm{CIGSD}\, V'_{GD})]$$

(Eq. G-262)

$$V'_{GD} = \sqrt{(V_{GD} - V_{FB,\mathrm{SD}})^2 + 10^{-4}}.$$

(Eq. G-263)

$$\mathrm{A3} = \begin{cases} 4.97232 \times 10^{-7}\ (\mathrm{nmos}) \\ 3.42537 \times 10^{-7}\ (\mathrm{pmos}) \end{cases} \frac{\mathrm{A}}{\mathrm{V}^2};$$

$$\mathrm{B3} = \begin{cases} 7.45669 \times 10^{11}\ (\mathrm{nmos}) \\ 1.16645 \times 10^{12}\ (\mathrm{pmos}) \end{cases} \frac{\mathrm{g}^{1/2}}{F^{1/2} \cdot \mathrm{s}}.$$

Energy/Power Breakdown of Pipelined Nanometer Caches (90nm/65nm/45nm/32nm)

Samuel Rodriguez and Bruce Jacob

Electrical and Computer Engineering Department
University of Maryland, College Park
{samvr,blj}@eng.umd.edu

ABSTRACT

As transistors continue to scale down into the nanometer regime, device leakage currents are becoming the dominant cause of power dissipation in nanometer caches, making it essential to model these leakage effects properly. Moreover, typical microprocessor caches are pipelined to keep up with the speed of the processor, and the effects of pipelining overhead need to be properly accounted for.

In this paper, we present a detailed study of pipelined nanometer caches with detailed energy/power dissipation breakdowns showing where and how the power is dissipated within a nanometer cache. We explore a three-dimensional pipelined cache design space that includes cache size (16kB to 512kB), cache associativity (direct-mapped to 16-way) and process technology (90nm, 65nm, 45nm and 32nm).

Among our findings, we show that cache bitline leakage is increasingly becoming the dominant cause of power dissipation in nanometer technology nodes. We show that subthreshold leakage is the main cause of static power dissipation, and that gate leakage is, surprisingly, not a significant contributor to total cache power, even for 32nm caches. We also show that accounting for cache pipelining overhead is necessary, as power dissipated by the pipeline elements is a significant part of cache power.

Categories and Subject Descriptors

B.3.2 [Memory Structures] : Cache memories.

General Terms

Design, Performance.

Keywords

Cache design, nanometer design, pipelined caches.

1. INTRODUCTION

Power dissipation has become a top priority for today's microprocessors. What was previously a concern mainly for mobile devices has also become of paramount importance for general-purpose and even high-performance microprocessors, especially with the recent industry emphasis on processor "Performance-per-Watt."

As transistors become steadily smaller with improving fabrication process technologies, device leakage currents are expected to significantly contribute to processor power dissipation [7]. Cache power dissipation has typically been significant, but increasing static power will have a greater impact on the cache since most of its transistors are inactive (dissipating no dynamic power, only static) during any given access. It is therefore essential to properly account for these leakage effects during the cache design process.

ISLPED'06, October 4–6, 2006, Tegernsee, Germany.

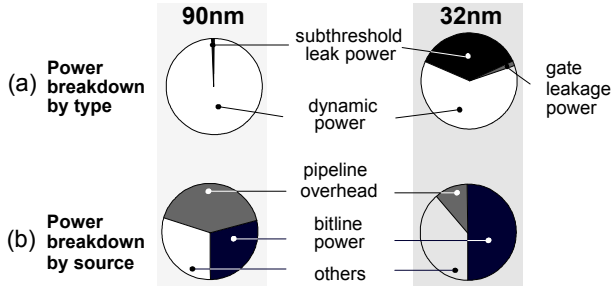


Figure 1: Power breakdowns of a 64kB-4way cache. (a) Dynamic and static power dissipation components for a 64kB-4W cache in 90nm and 32nm, (b) Major components of power dissipation for a 64kB-4way 90nm and 32nm pipelined cache

Moreover, microprocessor caches are obviously not designed to exist in a vacuum – they exist to complement the processor by hiding the relatively long latencies of the lower level memory hierarchy. As such, typical caches (specifically level-1 caches and often level-2) are pipelined and clocked at the same frequency as the core. Explicit pipelining of the cache will involve a non-trivial increase in access-time (because of added flop delays) and power dissipation (from the latch elements and the resulting additional clock power), and these effects must be accounted for properly, something which is not currently done by existing publicly available cache-design tools.

In this paper, we use analytical modeling of cache operation combined with nanometer BSIM3v3/BSIM4 SPICE models [5,13] to analyze the behavior of various cache configurations. We break down cache energy/power dissipation to show how much energy/power each individual part of a cache consumes, and what fraction can be attributed to dynamic (switching) and static (subthreshold leakage and gate tunneling) currents. We explore a three-dimensional cache design space by studying caches with different sizes (16kB to 512kB), associativities (direct-mapped to 16-way) and process technologies (90nm, 65nm, 45nm and 32nm).

Among our findings, we show that cache bitline leakage is increasingly becoming the dominant cause of power dissipation in nanometer technology nodes. We show that subthreshold leakage is the main cause of static power dissipation, but surprisingly, gate leakage tunneling currents do not become a significant contributor to total cache power even for the deep nanometer nodes. We also show that accounting for cache pipelining overhead is necessary, as power dissipated by the pipeline elements are a significant part of cache power.

2. BACKGROUND

2.1 Leakage

Dynamic power is dissipated by a circuit whenever transistors switch to change the voltage in a particular node. In the process, energy is consumed by charging (or discharging) the node's parasitic capacitive loads and, to a lesser degree, by possible short-circuit currents that flow during the small but finite time window when the transistor pull-up and pulldown networks are fully or partially turned "on," providing a low-impedance path from supply to ground.

On the other hand, static power is dissipated by leakage currents that flow even when the device is inactive. Different leakage mecha-

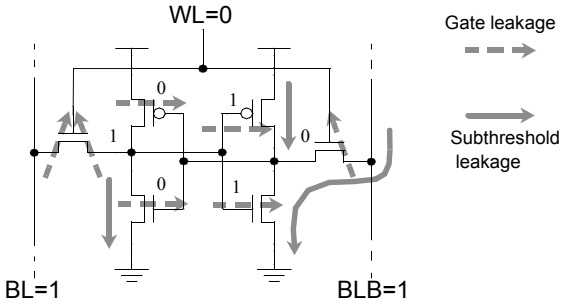


Figure 2: Memory cell leakage currents. An inactive six-transistor memory cell (6TMC) showing the subthreshold leakage and gate leakage currents flowing across the devices

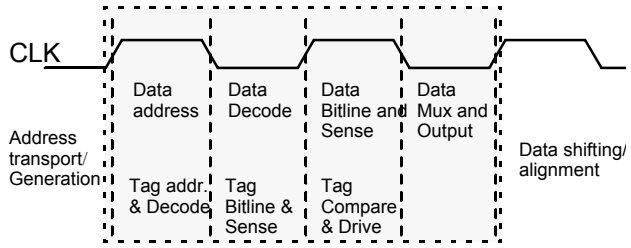


Figure 3: Cache pipeline diagram. The shaded region shows the part of the pipeline that we model

nisms exist for MOS transistors [2], but the two most important ones are lumped into the subthreshold leakage current and the gate leakage current. The subthreshold leakage current has been extensively studied [2,3,6,18,22]. It is mainly caused by the generational reduction in the transistor threshold voltage to compensate for device speed loss when scaling down the supply voltage, with the consequence of exponentially increasing subthreshold leakage current. The gate leakage current has been less extensively studied because of its relatively smaller value compared to subthreshold leakage for older technologies, but is increasingly receiving more attention [10,17,18,23] as it is expected to be comparable to the subthreshold leakage current in the deep nanometer nodes. Gate leakage currents flow whenever voltages are applied to transistor terminals, producing an electric field across gate oxides that are getting thinner (20 to less than 10 angstroms) resulting in significant leakage currents due to quantum tunneling effects.

Figure 2 shows a typical 6-transistor memory cell (6TMC) and the typical leakage currents involved for the memory cell idle state (i.e. wordline is off, one storage node is "0" and the other is "1"). Although a sizeable number of transistors in a cache are active for any given access, the vast majority of memory cells are in this inactive state, dissipating static power. Cache designers currently account for subthreshold leakage current (most problematic of which is the bitline leakage since this not only affects power, but also circuit timing and hence functionality), but not much attention is given to gate leakage in the publicly available cache design tools like CACTI [25,19,20] and eCACTI [14]. For better accuracy, this paper also considers the gate leakage currents as shown in the 6TMC diagram.

2.2. Pipelined Caches

To keep up with the speed of a fast microprocessor core while providing sufficiently large storage capacities, caches are pipelined to subdivide the various delays in the cache into different stages, allowing each individual stage to fit into the core's small clock period. Figure 3 shows a typical pipeline diagram for a cache. The given timing diagram shows operations being performed in both phases of the clock. Figure 4 shows a possible implementation of a pipeline latch that easily facilitates this phase-based operation.

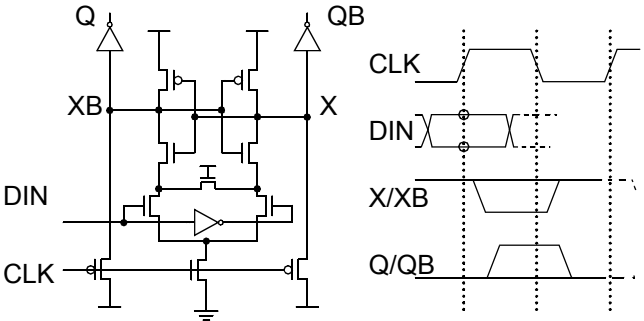


Figure 4: Pipeline latch. An example of a pipeline latch that can be used to implement the phase-based operations in the cache [8][15]. Also shown is the latch's timing diagram

The major publicly-available cache analysis tools CACTI and eCACTI use implicit pipelining through wave pipelining, relying on regularity of the delay of the different cache stages to separate signals continuously being shoved through the cache instead of using explicit pipeline state elements. Unfortunately, this is not representative of modern designs, since cache wave-pipelining is not being used by contemporary microprocessors [21,24]. Although wave pipelining has been shown to work in silicon prototypes [4], it is not ideally suited for high-speed microprocessor caches targeted for volume production which have to operate with significant process-voltage-temperature (PVT) variations. PVT variations in a wave-pipelined cache cause delay imbalances which, in the worst case, lead to signal races that are not possible to fix by lowering the clock frequency. Hence, the risk for non-functional silicon is increased, resulting in unattractive yields. In addition, wave-pipelining does not inherently support latch-based design-for-test (DFT) techniques that are critical in the debug and test of a microprocessor, reducing yields even further. On the contrary, it is easy to integrate DFT "scan" elements inside pipeline latches that allow their state to be either observed or controlled (preferably both). This ability facilitates debugging of microprocessor circuitry resulting in reduced test times that directly translate to significant cost savings. Although not immediately obvious at first, these reasons make it virtually necessary to implement explicit pipelining for high-volume microprocessors caches.

3. EXPERIMENTAL METHODOLOGY

Although initially based on CACTI and eCACTI tools, the analysis tool we have developed bears little resemblance to its predecesors. In its current form, it is, to the best of our knowledge, the most detailed and most realistic (i.e. similar to realistically implementable high-performance caches) cache design space explorer publicly available.

The main improvements of our tool compared to CACTI/eCACTI are the following:

- More optimal decode topologies [1] and circuits [16] along with more realistic device sizing ensures that cache inputs present a reasonable load to the preceeding pipeline stage (at most four times the load of a 1X strength inverter).[1]
- Accurate modeling of explicit cache pipelining to account for delay and energy overhead in pipelined caches.
- Use of BSIM3v3/BSIM4 SPICE models and equations to perform calculation of transistor drive strengths, transistor RC parasitics, subthreshold leakage and gate leakage. Simulation characteristics for each tech node are now more accurate.[2]
- More accurate RC interconnect parasitics by using local, intermediate and global interconnects (as opposed to using the

1. Both CACTI and eCACTI produce designs with impractically large first-stage inverters at the cache inputs. This shifts some of the burden of driving the cache decode hierarchy to the circuits preceding the cache, resulting in overly optimistic delay and power numbers.

same wire characteristics for all interconnects, as is being done by CACTI and eCACTI), and accurate analytical modeling of these structures. In addition, a realistic BEOL-stack[3] was used for each technology node.

An important note to make when discussing dynamic and static power and/or energy is that it is only possible to combine the two into a single measurement by assuming a specific frequency. Dynamic power inherently describes how much energy is consumed in a single switching event, and an assumption of how often that event happens is necessary to convert the energy value into power dissipation (hence the activity factor and frequency components in standard power equations). On the other hand, static leakage inherently describes the amount of current flow, and hence the instantaneous power, at any given time. Converting this into energy requires an assumption of the amount of time the current is flowing. Table 1 shows the values for frequency and supply voltages that were used for this study, where the values are chosen from historical [26] and projected [11,12] data.

Table I. VDD and frequency used for each tech node (T=50 C)

	250nm	180nm	130nm	90nm	65nm	45nm	32nm
VDD	2.0 V	1.8 V	1.6 V	1.4 V	1.3 V	1.2 V	1.1 V
Freq. (GHz)	0.8 GHz	1.5GHz	1.8GHz	2.4GHz	2.6GHz	2.8GHz	3.0GHz

4. RESULTS AND DISCUSSIONS

4.1. Dynamic and static power

Figure 5 shows the power dissipation of the different cache configurations as a function of process technology. Each column of plots represents a specific process technology, while each row represents a specific cache size. Each plot shows the dynamic, subthreshold leakage, gate leakage, and total power as a function of associativity for the given cache size and technology node.

The most basic observation here is that total power is dominated by the dynamic power in the larger technology nodes but is dominated by static power in the deep nanometer nodes (with the exception of very highly-associative small to medium caches).

This can be seen from the plots for the 90nm and 65nm nodes, where the dynamic power comprises the majority of the total power. In the 45nm node, the subthreshold leakage power is significant

2. For delay and dynamic power computations, CACTI and eCACTI use hardcoded numbers based on 0.80um technology and use linear scaling to translate power and delay numbers to the desired technology.
3. The Back-End-Of-Line stack refers to the fabrication steps performed after the creation of the active components. Use of a realistic BEOL-stack results in more accurate modeling of the interconnect. In CACTI and eCACTI, a single interconnect characteristic was assumed.

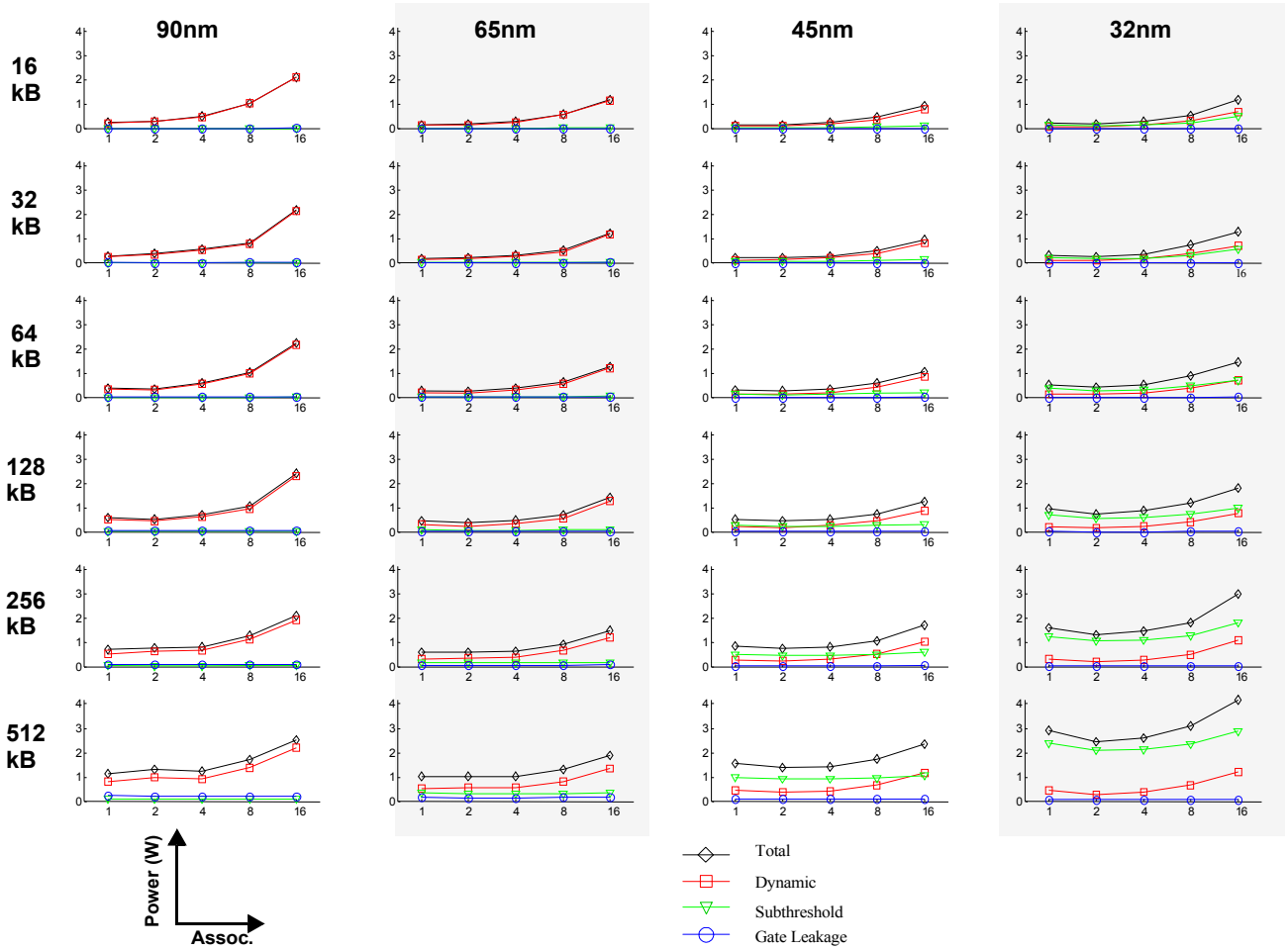


Figure 5: Power consumption vs. technology node of different cache configurations. The plots show how total power consumption is broken down into dynamic power and static power (due to subthreshold leakage and gate leakage). Each column of plots represents a single tech node, while a single row represents a specific cache size. Within each plot, the power dissipation is shown in the y-axis, with increasing associativitesrepresented in the x-axis.

enough that it becomes the dominant component for some configurations. For small caches of any associativity, dynamic power typically dominates because there are fewer leaking devices that contribute to subthreshold leakage power. But as cache sizes increase, the number of idle transistors that dissipate subthreshold leakage power also increases, making the subthreshold leakage power the dominant component of total power except for the configurations with high associativity (which requires more operations to be done in parallel, resulting in dissipation of more dynamic power). In the 32nm node, the subthreshold leakage power is already comparable to the dynamic power even for small caches (of any associativity), and starts to become really dominant as cache sizes increase (even at high associativities where we expect the cache to burn more dynamic power).

A surprising result that can be seen across all the plots is the relatively small contribution of gate leakage power to the total power. This is surprising given all the attention that gate leakage current is receiving and how it is expected to be one of the dominant leakage mechanisms. It turns out that this is a result of many factors, including the less aggressive gate oxide thickness scaling in the more recent ITRS roadmaps [11,12], and that Vdd scaling and device size scaling tend to decrease the value of the gate tunneling current. The net effect of a slightly thinner gate oxide (increasing tunneling), slightly lower supply voltage (decreasing tunneling), and smaller devices (decreasing tunneling) from one generation to the next might actually result in an effective decrease in gate leakage.

Some of the plots in Figure 5 (e.g. the 256k and 512k caches for the 45nm and 32nm node) exhibit a sort of "saddle" shape, which shows that increasing cache associativity from direct-mapped to 2-way or 4-way does not automatically cause an increase in power dissipation, as the internal organization may allow a more power optimal implementation of set-associative caches compared to direct-mapped caches, especially for medium to large-sized caches.

A final observation for Figure 5 is that technology scaling is capable both of increasing or decreasing the total power of a cache. Which direction the power goes depends on which power component is dominant for a particular cache organization. Caches with dominant dynamic power (e.g. small caches/highly associative caches) will enjoy a decrease in cache power as dynamic power decreases because of the net decrease in the CV^2f function, while caches with dominant static power (e.g. large caches/medium-sized low-associativy caches) will suffer from an increase in cache power as the static power increases.

4.2. Detailed power breakdown

Figure 6 shows a detailed breakdown of cache power for three different cache configurations (16kB-4W, 32kB-4W, 64kB-4W). Each plot represents a single configuration implemented in the four nanometer nodes. For each of these nodes, total cache power is shown, along with its detailed breakdown. Power dissipation for each component, as shown by each vertical bar in the plot, is also broken down into its dynamic, subthreshold leakage, and gate leakage components.

The first notable point here is that for these cache configurations, going from 90nm to 65nm results in power decrease; going from 65nm to 45nm results in a smaller power decrease; finally, going from 45nm to 32nm causes a significant power increase. Again, these observations are caused by the intertwined decrease and increase of dynamic and subthreshold leakage power, respectively, as we go from one technology node to the next.

A second point is that for most of the plots, it can be seen that the two main contributors to cache power are the bitlines and the pipeline overhead (with the data_dataout component, which accounts for the power in driving the output data buses, also significant for some configurations). Pipeline overhead, shown in the plots as data_pipe_ovhd and tag_pipe_ovhd, accounts for the power dissipated in the pipeline latches along with the associated clock tree circuitry.

This is an important observation as it shows that the overhead associated with pipelining the cache is often a very significant component of total power, and that most of this power is typically dynamic. This can be easily seen in Figure 7, which shows the fraction of total power dissipated by pipeline overhead for all the configurations studied. This is especially noteworthy since we model aggressive clock gating where only latches that need to be activated actually see a clock signal. Consequently, we expect the pipeline overhead to be even more significant for circuits that use less aggressive clock-gating mechanisms. Lastly, it should be noted that since most of the power in the pipeline overhead is dynamic, it should decrease for smaller technology nodes, as seen in Figure 5.

Another point of importance is that although dynamic power in general tends to decrease because of the net effect of increased frequency but decreased supply voltage and device capacitances, this is the case only for circuits with device-dominated loading (i.e. gate and diffusion capacitances). Circuits with wire-dominated loading may actually see dynamic power go up as we go from one generation to the next, since a wire capacitance decrease due to shorter lengths and slightly smaller coupling area will typically be offset by the shorter distances between wires that cause significantly increased capacitances. This can be seen in Figure 6, where the dynamic power for the data_dataout component increases for all three configurations, mostly since the load for data_dataout is wire-dominated, as it involves only a few high-impedance drivers driving a long wire across the entire cache.

The power breakdown of representative cache configurations as shown in Figure 6 shows that most of the power in a pipelined, nanometer cache is dissipated in the bitlines, the pipeline overhead, and possibly the data output drivers. To get a better view of the design space, we can lump together similar components in order to show the entire space. In Figure 8, we subdivide total power into five categories -- bitline power, pipeline overhead power, decoder power, data processing power (data/tag sensing, tag comparison, data output muxing and driving), and lump all remaining blocks into a single component labeled "others." Results for the 65nm and 32nm nodes for all the configurations are then plotted together.

It is interesting to see that while the power dissipation due to the bitlines monotonically increases as cache size increases for both techology nodes, the power due to the other components does not necessarily do so. For instance, the pipeline power of the 65nm 128k-16 way and 256kB-16 way caches, where the pipeline power is significantly reduced from one configuration to the next. The reason here is that the bitline power mainly depends on the cache size (assuming static power is the dominant cause of bitline power) and not the cache implementation. The other power components, on the other hand, greatly depend on the particular implementation, so their values may noticeably fluctuate from one implementation to the next, especially if there exist implementations that cause an increase in the power of one component but results in a significant power improvement in

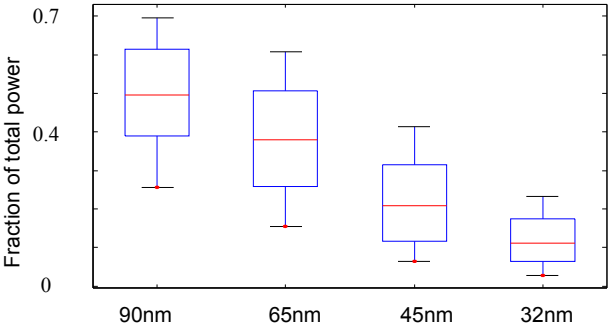


Figure 7: Pipeline overhead contribution to total cache power. Distribution of data showing the fraction of total power attributed to pipeline overhead for all cache configurations, where each plot shows min, max, median, and quartile information

some other part of the cache. It is important to realize that the optimization scheme that we used optimized total power, and not component power. As long as a specific implementation has a better power characteristic compared to another, relative values of specific cache components were not considered important.

It is easily seen from the different cache configurations in Figure 8 that that pipeline power (as represented by pipe_ovhd) will typically be a non-trivial contributor to the total cache power. Any analysis that fails to account for this pipelining overhead and instead assumes that caches can be trivially interfaced to a microprocessor core will be inaccurate. Although the figure also shows that the contribution of the pipeline overhead to total power is reduced from 65nm to 32nm compared to the contribution of the "dataproc" component (mainly because of the different effect of technology to the dynamic power of the two components), it should be noted that our study modeled conservative scaling of the dielectric constants of the interlevel dielectrics (i.e. we did not model aggressively scaled low-K interconnects). If wire delay becomes especially problematic in future generations and more aggressive measures are taken to improve the interconnect performance, we can expect the relative contribution of the pipeline overhead to the total power to increase in significance.

A final observation that we want to point out from Figure 8 is that increasing cache associativity may not necessarily result in a significant increase in total power, especially for the larger caches at 65nm and below. The power for these configurations are typically dominated by the bitline leakage, reducing the effect of any increase in power due to higher associativity. In some cases, the total power dissipation can actually decrease with an increase in associativity, as a

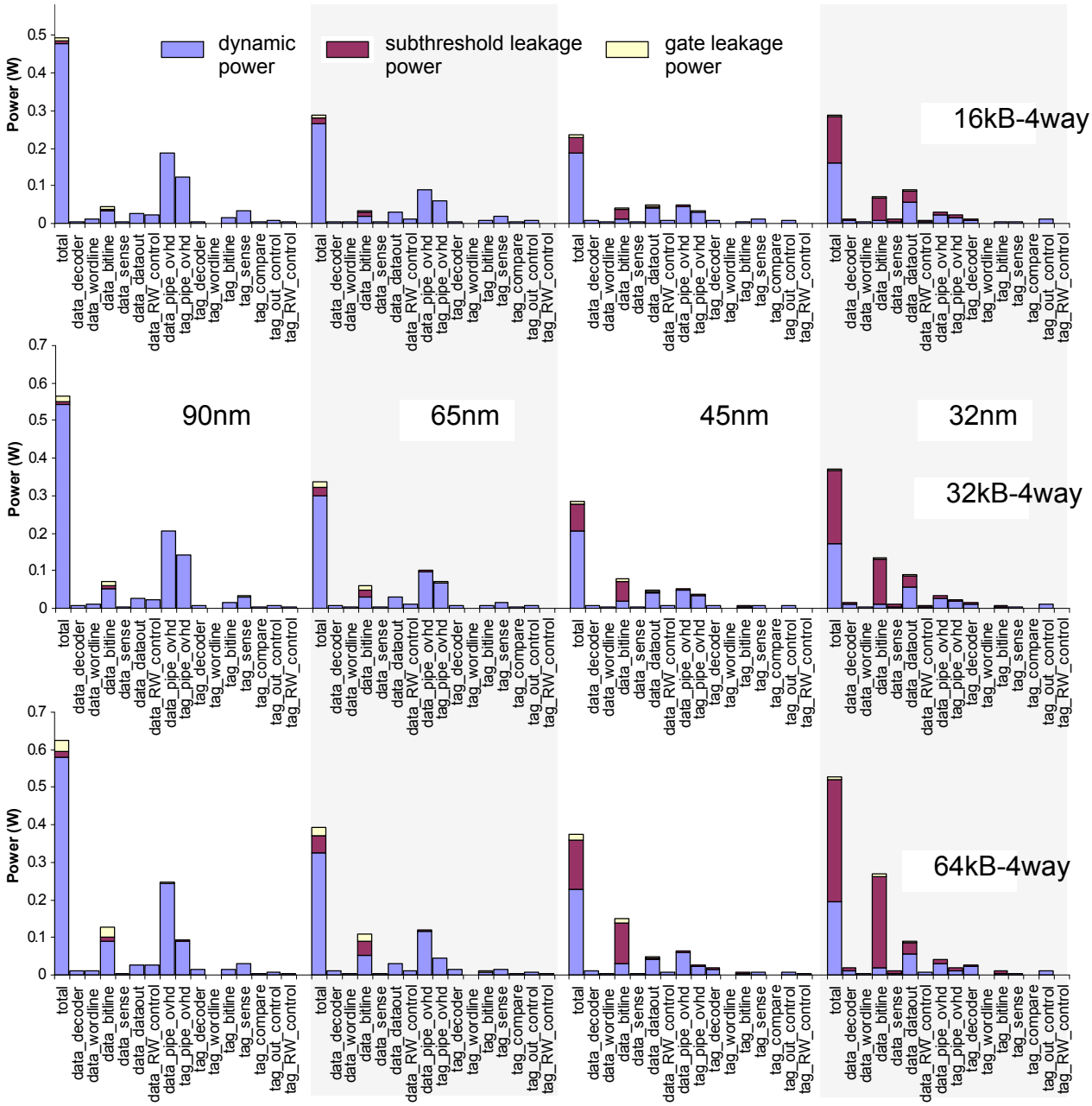


Figure 6: Detailed cache power breakdown. Detailed power breakdown for three different cache configurations showing the amount of dynamic, subthreshold leakage and gate leakage power being consumed by different parts of the cache. Each plot shows four sets of valuescorresponding to different technology nodes. (Note that all three plots use the same scale)

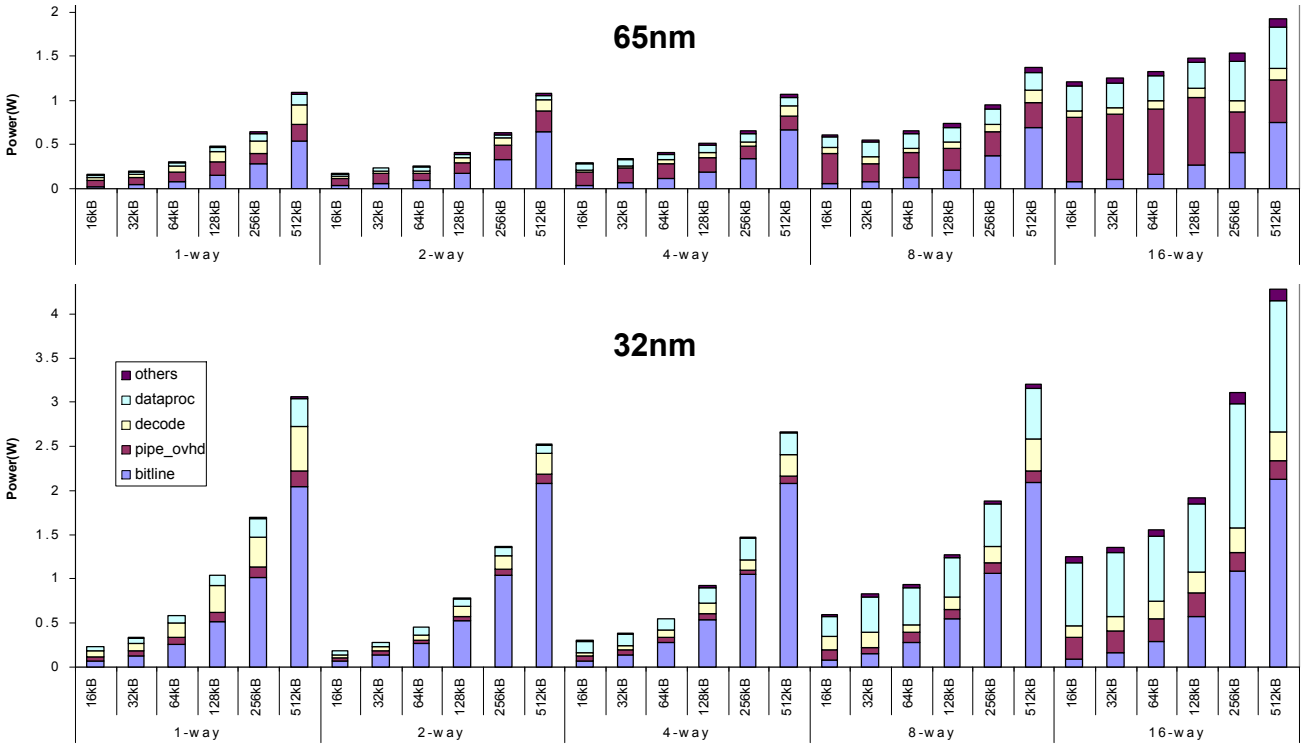


Figure 8: Power contributions of different cache parts. Power breakdown showing major cache power contributors for the 65nm and 32nm technology nodes for different cache sizes and associativities. (Note that the y-axis for both plots use the same scale)

particular configuration is able to balance the overall static and dynamic power for all the cache components. This knowledge will be useful in enabling large, high-associativity caches suitable for use in L2 caches (and higher).

5. CONCLUSION

We have presented a detailed power breakdown of the different nanometer pipelined cache configurations in a three-dimensional design space consisting of different cache sizes, associativities and process technologies.

We have shown that the power of nanometer caches will continue to increase, and that this increase will be primarily driven by static power due to subthreshold leakage currents. In addition, we saw that static power due to gate leakage tunneling currents does not contribute significantly to the cache power, even in the deep nanometer technology nodes.

Lastly, we have argued that practical microprocessor caches virtually require the use of pipeline latches to facilitate designing PVT variation-tolerant caches and debug and test-friendly designs. But in using explicit pipelining, accurate cache analysis requires accounting for the overhead introduced by these pipeline latches, as these represent a significant portion of the cache's power dissipated (in some corner cases being the dominant cause of power).

6. REFERENCES

[1] B.Amrutur and M.Horowitz, "Fast low-power decoders for RAMs," IEEE JSSC, vol.36, no.10, Oct 2001.

[2] Mohab Anis, "Subthreshold leakage current: challenges and solutions," ICM 2003.

[3] S.Borkar, "Circuit techniques for subthreshold leakage avoidance, control and tolerance," IEDM 2004.

[4] W.P.Burleson et al., "Wave-pipelining: A tutorial and research survey," IEEE Trans. VLSI Systems, vol.6, no.5, Sep 1998.

[5] Y.Cao et al., "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," Proc. of CICC, pp.201-204, 2000.

[6] Z.Chen, M.Johnson, L.Wei and K.Roy, "Estimation of standby leakage power in CMOS circuits considering accurate modeling of transistor stacks," in Proc. Int.Symp.Low Power Electronics and Design, 1998.

[7] B.Doyle, et al., "Transistor elements for 30nm physical gate lengths and beyond," Intel Technology Journal, Vol.6, Page(s): 42-54, May 2002.

[8] P.Gronowski et al., "A 433-MHz 64-b quad-ussue RISC microprocessor, "JSSC, vol.31, no.11, Nov.1996, pp.1687-1696.

[9] J.P.Halter and F.Najm, ,"A gate-level leakage power reduction method for ultra-low-power CMOS circuits," in Proc. IEEE Custom Integrated Circuits Conf., 1997.

[10] F.Hamzaoglu and M.R.Stan, "Circuit-level techniques to control gate-leakage for sub-100nm CMOS," ISLPED August 2002.

[11] "International Technology Roadmap for Semiconductors 2005 Edition," Semiconductor Industry Association, http://public.itrs.net.

[12] "International Technology Roadmap for Semiconductors 2003 Edition," Semiconductor Industry Association, http://public.itrs.net.

[13] Predictive technology model. http://www.eas.asu.edu/~ptm

[14] M.Mamidipaka and N.Dutt, "eCACTI: An enchanced power estimation model for on-chip caches," Center for Embedded Computer Systems, Technical Report TR 04-28, Oct. 2004.

[15] J.Montanaro et al., "A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor, JSSC, vol.31 no.11, Nov. 1996, pp.1703-1714.

[16] H.Nambu, et.al., "A 1.8-ns access, 550MHz, 4.5-Mb CMOS SRAM," IEEE J. Solid-State Circuits, vol.33, no.11, pp.1650-1658, Nov. 1998.

[17] R.Rao, J.L. Burns and R.B.Brown, "Circuit techniques for gate and subthreshold leakage minimization in future CMOS technologies," , 2003. ESSCIRC '03..

[18] R.M.Rao, J.L.Burns and R.B.Brown, "Circuit techniques for gate and subthreshold leakage minimization in future CMOS technologies," ESSCIRC '03. Proceedings of the 29th European Solid-State Circuits Conference, 2003.

[19] G.Reinman and N.Jouppi, "CACTI 2.0: An integrated cache timing and power model," WRL Research Report 2000/7, Feb.2000.

[20] P.Shivakumar and N.Jouppi , "CACTI 3.0: An integrated cache timing, power and area model," WRL Research Report 2001/2, Aug.2001.

[21] Riedlinger, R., Grutkowski, T., "The high-bandwidth 256 kB 2nd level cache on an Itanium microprocessor," ISSCC 2002.

[22] Y.Ye, S.Borkar and V.De, "A new technique for standby leakage reduction in high-performance circuits," in Symp.VLSI Circuits Dig.Tech.Papers, 1998.

[23] Y.C.Yeo et al., "Direct tunneling gate leakage current in transistorswith ultrathin nsilicon nitride gate dielectric," IEEE Electron Device Letters, Nov. 2000.

[24] Weiss, D., Wuu, J.J., Chin, V., "An on-chip 3MB subarray-based 3rd levl cache on an Itanium microprocessor," ISSCC 2002.

[25] S.J.Wilton and N.P.Jouppi, "CACTI: An enhanced cache access and cycle time model," IEEE JSSC, vol.31, no.5, May 1996.

[26] The Tech Report. http://techreport.com/cpu/.