Sleepy Stack: a New Approach to Low Power VLSI Logic and Memory

A Thesis
Presented to
The Academic Faculty

by

Jun Cheol Park

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Electrical and Computer Engineering
Georgia Institute of Technology
August 2005

Sleepy Stack: a New Approach to Low Power VLSI Logic and Memory

Approved by:

Professor Vincent J. Mooney III, Advisor
School of Electrical and
Computer Engineering
Georgia Institute of Technology

Professor John Dorsey
School of Electrical and
Computer Engineering
Georgia Institute of Technology

Professor Abhijit Chatterjee
School of Electrical and
Computer Engineering
Georgia Institute of Technology

Professor Michael Niemier
College of Computing
Georgia Institute of Technology

Professor Paul E. Hasler
School of Electrical and
Computer Engineering
Georgia Institute of Technology

Date Approved: June 24, 2005

To my wife, Jaemin Lee,

for her love, support and sacrifices.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to everyone who made this thesis possible. Most of all, I would like to thank my advisor, Professor Vincent J. Mooney III, for his guidance of my research and his patience during my Ph.D. study at Georgia Tech. With his knowledge and experience, he has guided me to successfully achieve my research objective.

Second, I would like to thank my thesis committee members, Professor Abhijit Chatterjee, Professor Paul E. Hasler, Professor John Dorsey, and Professor Michael Niemier for their critical evaluation and valuable suggestions. Especially Professor Chatterjee and Professor Hasler served me as reading committee members and gave me invaluable feedback for my thesis.

Third, I would like to thank all my colleagues in the Hardware/Software Codesign Group and the COPAC group for their support and friendship. Especially, Kiran Puttaswamy, Sudarshan K. Srinivasan, Pinar Korkmaz, and Philipp Pfeiffenberger helped me to make valuable results during my Ph.D. at Georgia Tech.

Last, but most importantly, I would like to specially thank to my wife Jaemin Lee for her love and support during my Ph.D. She encouraged me to finish my Ph.D. research.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS OR ABBREVIATIONS

6-T 6 Transistor.

ABC-MTCMOS Auto-Backgate-Controlled Multi-Threshold CMOS.

ACC Asymmetric-Cell Cache.

BTB Branch Target Buffer.

CMOS Complementary Metal Oxide Semiconductor.

DIBL Drain Induced Barrier Lowering.

EDR Energy-Delay Ratio.

FBB Forward-Body Bias.

ITRS International Technology Roadmap for Semiconductors.

LPPC Low-Power Pipelined Cache.

MTCMOS Multi-Threshold-voltage CMOS.

RBB Reverse-Body Bias.

RTL Register Transfer Level.

SRAM Static Random Access Memory.

VLSI Very Large Scale Integration.

ZBB Zero-Body Bias.

SUMMARY

The main objective of this thesis is to provide new low power solutions for Very Large Scale Integration (VLSI) designers. Especially, we focus on leakage power reduction. Although leakage power was negligible at 0.18μ technology and above, in nanoscale technology, such as 0.07μ, leakage power is almost equal to dynamic power consumption.

In this thesis, we present a novel circuit structure we call "sleepy stack." The sleepy stack structure dramatically reduces leakage. The sleepy stack is a combination of two well-known low-leakage techniques which are the forced stack technique and the sleep transistor technique. The sleepy transistor technique can achieve ultra-low leakage power consumption, but loses logic state during sleep mode. Meanwhile, the forced stack technique saves leakage power consumption by stacking transistors and retains logic state. However, the forced stack technique cannot use high-V_{th} without incurring a dramatic delay increase ($>$6.2X); however, if only low-V_{th} transistors are used, the leakage power savings of this technique are small. By combining two prior techniques, however, the sleepy stack technique can achieve (i) ultra-low leakage power consumption while (ii) saving state. One of the main advantages of the sleepy stack technique is a use of high-V_{th} transistors for key places. Utilizing high-V_{th} transistors, the sleepy stack technique can achieve around 200X leakage power reduction with similar delay as the forced stack technique using low-V_{th} transistors (when applied to generic logic circuits targeting 0.07μ technology). Since the sleepy stack technique comes with area and delay overhead compared to a conventional Complementary Metal Oxide Semiconductor (CMOS) technique, the sleepy stack technique can be applicable to a design that requires ultra-low leakage power consumption with quick response time and is able to pay the associated area and delay cost.

One of the advantages of the sleepy stack technique is saving state. Therefore, the sleepy stack technique can be applicable memory design, i.e., Static Random Access Memory (SRAM). When we apply the sleepy stack to SRAM cell design, we can observe new Pareto points which have not been presented prior to the research in this thesis. Although the sleepy stack incurs some delay and area overheads, the sleepy stack SRAM cell can achieve ultra-low leakage power consumption while suppressing two main leakage paths in an SRAM cell. When compared to a high-V_{th} SRAM cell, which is the best prior state-saving SRAM cell, the sleepy stack SRAM cell achieves 5.13$\sim$ 2.77X greater leakage reduction with 32$\sim$ 19% delay increase (at 110^o using 0.07μ technology). Alternatively, by increasing sleepy stack transistor widths, the sleepy stack SRAM cell can achieve approximately the same delay as high-V_{th} SRAM yet achieves 2.49$\sim$ 2.26X leakage reduction over high-V_{th} SRAM. Unfortunately, there is a cost of 140% area increase.

Along with the sleepy stack structure, we propose a architectural level low power technique we name Low-Power Pipelined Cache (LPPC). Originally, a pipelined cache was proposed to reduce cache pipeline stage delay and thus improve processor performance. However, we use the reduced cache delay to reduce power consumption. Our strategy is to pipeline caches without changing cycle time; instead, we lower cache supply voltage to save power consumption. Although we may increase the depth of a pipelined cache to achieve large power reduction, total energy savings associated with increasing pipeline depth may limited due to the pipelining penalties and lower limit of supply voltage. We obtain an optimal depth of the pipelined cache in our experimental configuration targeting an embedded processor; the result we find is that a two stage pipelined cache achieves maximum energy reduction (70% cache power savings).

The sleepy stack structure achieves ultra-low leakage power consumption with some area and delay overheads. To reduce the delay overhead while achieving low-leakage power, we combine LPPC and the sleepy stack. When targeting 0.07μ technology, the sleepy stack pipelined cache achieves 17X leakage power reduction with 4% execution

cycle increase and 31% active power increase compared to conventional SRAM. Using a pipelined cache, we can hide most of the delay overhead induced by the sleepy stack, which is 33%. The result indicates that total energy is saved when the device spends at least three times as much time in sleep mode as in active mode.

In summary, this thesis presents heretofore unexplored methods for low-power VLSI design. In particular, the sleepy stack approach provides what may be the best solution for VLSI designers concerned about the twin problems of low static power and maintenance of VLSI logic state during sleep mode. For such a two-headed problem, the sleepy stack approach can provide two orders of magnitude (100X) or more static power reduction over the best prior approach; however, there is a cost – potentially quite small – in terms of delay increase and area overhead. In short, sleepy stack principles provide heretofore unknown Pareto points for consideration in VLSI design.

CHAPTER I

INTRODUCTION

Power consumption is one of the top concerns of Very Large Scale Integration (VLSI) circuit design, for which Complementary Metal Oxide Semiconductor (CMOS) is the primary technology. Today's focus on low power is not only because of the recent growing demands of mobile applications. Even before the mobile era, power consumption has been a fundamental problem. To solve the power dissipation problem, many researchers have proposed different ideas from the device level to the architectural level and above. However, there is no universal way to avoid tradeoffs between power, delay and area, and thus designers are required to choose appropriate techniques that satisfy application and product needs.

Power consumption of CMOS consists of dynamic and static components. Dynamic power is consumed when transistors are switching, and static power is consumed regardless of transistor switching. Dynamic power consumption was previously (at 0.18μ technology and above) the single largest concern for low-power chip designers since dynamic power accounted for 90% or more of the total chip power. Therefore, many previously proposed techniques, such as voltage and frequency scaling, focused on dynamic power reduction. However, as the feature size shrinks, e.g., to 0.09μ and 0.065μ, static power has become a great challenge for current and future technologies. Based on the International Technology Roadmap for Semiconductors (ITRS) [30], Kim et al. report that subthreshold leakage power dissipation of a chip may exceed dynamic power dissipation at the 65nm feature size [38].

One of the main reasons causing the leakage power increase is increase of subthreshold leakage power. When technology feature size scales down, supply voltage and threshold voltage also scale down. Subthreshold leakage power increases exponentially as threshold

voltage decreases. Furthermore, the structure of the short channel device lowers the threshold voltage even lower. In addition to subthreshold leakage, another contributor to leakage power is gate-oxide leakage power due to the tunneling current through the gate-oxide insulator. Since gate-oxide thickness will be reduced as the technology decreases, in nanoscale technology, gate-oxide leakage power may be comparable to subthreshold leakage power if not handled properly. However, we assume other techniques will address gate-oxide leakage; for example, high-k dielectric gate insulators may provide a solution to reduce gate-leakage [38]. Therefore, this thesis focuses on reducing subthresholod leakage power consumption.

In this dissertation, we provide novel circuit structure named "sleepy stack" as a new remedy for designers in terms of static power. The sleepy stack has a novel structure that combines the advantages of two major prior approaches, the sleep transistor technique and the forced stack technique. However, unlike the sleep transistor technique, the sleepy stack technique retains the original state; furthermore, unlike the forced stack technique, the sleepy stack technique can utilize high-V_{th} to achieve more than two orders of magnitude leakage power reduction compared to the forced stack. Unfortunately, the sleepy stack technique comes with delay and area overheads. Therefore, the sleepy stack technique provides new Pareto points to designers who require ultra-low leakage power consumption and are willing to pay some area and delay cost. In this thesis, we explore the basic structure of the sleepy stack. Also, we study various sleepy stack circuits including generic logic circuits and memory. We discuss the advantages and disadvantages of the sleepy stack and a technique to reduce the delay overhead.

Along with the sleepy stack structure, we introduce in this dissertation an architectural level power reduction technique. One of the most effective dynamic power reduction techniques is lowering the supply voltage of CMOS transistors because the power consumption of CMOS transistors increases quadratically proportional to the supply voltage. However, lowering the supply voltage incurs an increase in transistor switching delays. Therefore,

designing CMOS circuits typically necessitates tradeoffs between performance (in terms of delay) and power consumption. Although CMOS circuits are governed by such tradeoffs, it is possible for a system to selectively lower supply voltage without compromising performance at the architectural level. The strategy is to lower supply voltage for circuits in non-critical paths while maintaining supply voltage for circuits in critical path(s); thus, available surplus slack in non-critical paths are removed. In our particular case, we observe that by pipelining the caches, we can obtain large surplus slack, which allows for large dynamic power savings. This new low-power cache technique is named Low-Power Pipelined Cache (LPPC). By apply sleepy stack instead of lowering supply voltage, LPPC can be used to save leakage power consumption.

1.1 Problem Statement

This research work addresses new low power approaches for Very Large Scale Integration (VLSI) logic and memory. Power dissipation is one of the major concerns when designing a VLSI system. Until recently, dynamic power was the only concern. However, as the technology feature size shrinks, static power, which was negligible before, becomes an issue as important as dynamic power. Since static power increases dramatically (indeed, even exponentially) in nanoscale silicon VLSI technology, the importance of reducing leakage power consumption cannot be overstressed. A well-known previous technique called the sleep transistor technique cuts off V_{dd} and/or Gnd connections of transistors to save leakage power consumption. However, when transistors are allowed to float, a system may have to wait a long time to reliably restore lost state and thus may experience seriously degraded performance. Therefore, retaining state is crucial for a system that requires fast response even while in an inactive state. Our research provides new VLSI techniques that achieve ultra-low leakage power consumption while maintaining logic state, and thus can be used for a system with long inactive times but a fast response time requirement.

1.2 Contributions

In this dissertation, we present two low-power techniques, (i) the sleepy stack technique for static power reduction and (ii) the low-power pipelined cache (LPPC) for dynamic and/or static power reduction. The two techniques provide new weapons to designers whose primary concern is low power.

The following items are the main contributions of this research:

- **Design of sleepy stack for logic circuits.** We develop a novel low-leakage technique we call "sleepy stack." The sleepy stack technique is applied to generic logic circuits, and we achieve between two and three orders of magnitude leakage power reduction compared to the best prior state saving technique we could find (namely, the forced stack technique).

- **Design of a sleepy stack SRAM cell.** Static Random Access Memory (SRAM) is a power hungry component in a VLSI chip. Therefore, we apply the sleepy stack technique to SRAM design. Since the sleepy stack technique comes with area and delay penalties, we explore many possible sleepy stack SRAM cell combinations and provide new Pareto points that can be used by designers who want extremely low leakage power consumption (and are willing to pay a cost of some area and/or delay increase).

- **Design of a novel low power pipelined cache (LPPC).** We design a novel low-power pipelined cache (LPPC). LPPC applies pipelining to a cache while lowering supply voltage of a cache to reduce dynamic power or using sleepy stack SRAM to reduce leakage power. We also optimize the number of cache pipeline stages using a specific architecture that we explore.

- **Design of novel sleepy stack pipelined cache.** We design a novel sleepy stack pipelined cache. We utilize the LPPC to save leakage power consumption by using

sleepy stack SRAM. Although the sleepy stack SRAM shows some delay overhead, by using the LPPC we can hide the delay overhead, and thus we can achieve very low leakage cache power consumption with small performance overhead.

1.3 Thesis Organization

The thesis is organized into eleven chapters.

CHAPTER I: INTRODUCTION. This chapter introduces power consumption issues in VLSI. This chapter also summarizes the contributions of this thesis. Finally, this chapter explains organization of the thesis.

CHAPTER II: MOTIVATION. This chapter addresses our motivation for this research.

CHAPTER III: NOTATION AND BACKGROUND. This chapter explains important notation and background used throughout this dissertation.

CHAPTER IV: PREVIOUS WORK. This chapter describes previous work in power reduction research and explains key differences between our solutions and previous work.

CHAPTER V: SLEEPY STACK STRUCTURE. This chapter introduces the novel sleepy stack leakage reduction technique. First, the structure of the sleepy stack is described followed by a detailed explanation of sleepy stack operation. An analytical delay model of the sleepy stack is derived and compared to the forced stack technique using an inverter circuit.

CHAPTER VI: APPLYING SLEEPY STACK. This chapter explores various applications of the sleepy stack approach. The applications/uses include generic logic circuits and memory circuits. For each application/use of the sleepy stack, comparisons

with the best known prior low-leakage techniques are carried out using benchmark circuits. We explain the experimental methodology used.

CHAPTER VII: SLEEPY STACK EXPERIMENTAL RESULTS. This chapter discusses the experimental results from various applications of the sleepy stack approach. The sleepy stack technique is empirically compared to well-known previous approaches. The comparisons are assessed in terms of area, dynamic power, static power and area while changing numerous VLSI and CMOS circuit parameters.

CHAPTER VIII: LOW-POWER PIPELINED CACHE (LPPC) ARCHITECTURE. This chapter introduces our new dynamic power reduction technique "low-power pipelined cache" and explains low power mechanism of the LPPC. The pipelining techniques for LPPC are discussed, and lastly, this chapter explores pipelining penalties and the solutions of the LPPC.

CHAPTER IX: LOW-POWER PIPELINED CACHE (LPPC) EXPERIMENTAL SETUP AND RESULTS. This chapter covers experimental methodology and LPPC results. In the experimental methodology, processor and cache models are explained followed by the architectural configurations and benchmarks used to evaluate the LPPC approach. Finally, the chapter provides experimental results for the low-power pipelined cache architecture compared to a non-pipelined cache architecture as well as a low-voltage pipelined cache architecture.

CHAPTER X: SLEEPY STACK PIPELINED CACHE. This chapter discusses combining the sleepy stack technique and the low-power pipelined cache technique. The proposed cache structure is explained and explored in terms of performance and power.

CHAPTER XI: CONCLUSION. This chapter summarizes the major accomplishments of this thesis.

CHAPTER II

MOTIVATION

Historically, in the 1980's CMOS technology took over the mainstream of VLSI design because CMOS consumes far less power than its predecessors (nMOS, bipolar, etc.). Although this advantage still holds, power dissipation of CMOS has nonetheless become a problem.

For a long time, dynamic power accounted for more than 90% (typically, over 99%) of total chip power, and thus was frequently used as the metric for total power consumption for technologies 0.18μ and above. However, as technology scales down to tens of nanometers, leakage power becomes as important as dynamic power. Therefore, many ideas have been proposed to tackle the leakage power problem. Although cutting off transistors from power rails, e.g., using the sleep transistor technique, is one of the possible solutions, losing state during inactive mode incurs long wake-up time and thus may not be appropriate for a system that requires fast response times.

To provide a motivational scenario to illustrate the possible impact of this thesis, let us compare the impact of static (leakage) power consumption in the context of a cell phone example. We assume that in general, the cell phone we consider is always on (i.e., 24 hours a day). However, the actual usage time of the cell phone is very limited. If we assume a 500 minute calling plan with 500 minutes total used per month, the cell phone is active only 1.15% $(500min/(30days * 24hours * 60min))$ of the total on-time. This means that during rest – 98.85% of the time – the cell phone is non-active; however, due to static power consumption, during rest (standby) the cell phone still consumes energy and reduces battery life. In technology such as 0.07μ, the impact of leakage power is huge.

Let us consider an energy consumption scenario of a cell phone predicted based on

Table 1: Power and area results from Chapter 7

	Best prior work that saves state (forced stack)			Our approach (sleepy stack)		
	Active power (W)	Leakeage power (W)	Area (u^2)	Active power (W)	Leakeage power (W)	Area (u^2)
4 Inverters	1.25E-06	9.81E-10	5.97E+00	1.09E-06	4.56E-12	9.03E+00
512B SRAM	5.22E-04	5.39E-06	2.00E+01	5.80E-04	3.24E-07	3.66E+01

Table 2: Energy consumption scenario of a cell phone (0.07μ)

	Best prior work that saves state (forced stack)				Our approach (sleepy stack)			
	Active power (W)	Leakeage power (W)	Area (u^2)	Energy (J) (Month)	Active power (W)	Leakeage power (W)	Area (u^2)	Energy (J) (Month)
Processor logic circuits	1.38E-01	1.02E-01	6.61E+05	2.65E+05	1.47E-01	5.74E-04	1.21E+06	5.87E+03
32KB SRAM	5.54E-03	4.15E-02	6.61E+05	1.06E+05	6.09E-03	2.44E-03	1.21E+06	6.44E+03
Total	1.43E-01	1.43E-01	1.32E+06	3.72E+05	1.53E-01	3.01E-03	2.42E+06	1.23E+04

our experimental results which will be presented in Chapter 7. Specifically, Table 1 shows some specific results from Chapter 7 for 0.07μ technology at 25^oC. Table 2 shows our hypothetical energy consumption scenario. We compare two different techniques which are the forced stack technique (best prior work that saves state) and the sleepy stack technique (our approach).

First, we assume a single chip containing an embedded processor core in 0.07μ technology. The chip largely consists of logic circuits and a 32KB SRAM; note that we exclude I/Os and the pad frame. Furthermore, we only consider here the digital chip; i.e., the liquid crystal display, Radio Frequency (RF) circuitry, etc., are all ignored.

Second, we assume that SRAM and logic circuits each occupy half of the digital chip area, respectively. We estimate 32KB SRAM area based on SRAM cell area which we will present in Chapter 7 – note that in all cases we exclude test, e.g., our SRAM does not include Built-In Self Test (BIST). The forced stack 32KB SRAM area is $6.61\text{x}10^5\mu^2$, and the sleepy stack SRAM area is $1.21\text{x}10^6\mu^2$. Then we estimate that the processor logic gates occupy the same amount of area as the 32KB SRAM as shown in the area columns of Table 2.

Third, we also assume that at 0.07μ technology leakage power consumption is as much as active power consumption when we use the forced stack technique. We multiply forced stack leakage values from Table 1 by a factor (specifically, 939), so that forced stack leakage power becomes the same as forced stack active power, i.e., 143mW. Then we apply the same factor (939) to the sleepy stack leakage power from Table 1, resulting in sleepy stack leakage power of 3.01mW. In other words, while Table 1 is based on Berkeley Predicted Technology Model (BPTM) [7], we instead assume a scenario where leakage power equals active power (which is, we believe, a hypothetical situation we may possibly see in the future.)

Now, recalling that our cell phone is active 500 minutes per month and thus inactive 42700 minutes per month, we calculate forced stack digital chip energy per month as follows:

$$Energy1 = 143mW * (500 * 60sec) + 143mW * (42700 * 60sec) \quad (1)$$

$$= 37.2KJ \quad (2)$$

Similarly, we calculate sleepy stack digital chip energy per month as follows:

$$Energy2 = 153mW * (500 * 60sec) + 3.01mW * (42700 * 60sec) \quad (3)$$

$$= 1.23KJ \quad (4)$$

The result predicts that the ultra-low leakage power technology, i.e., sleepy stack, saves 30X total energy consumption compared to the best prior work, i.e., forced stack. Therefore, potentially, the ultra-low leakage power technique can extend by 30X the cell phone battery life in this motivational example. There is a cost for this 30X savings, however: note that the overall are increases 83% (from $1.32mm^2$ to $2.42mm^2$ – see Table 2).

Although there already exist many low-leakage techniques, the best prior low-leakage technique in terms of leakage power reduction, the sleep transistor technique, loses logic state during sleep mode. Therefore, the sleep transistor technique requires non-negligible

time to wake-up the device from the sleep mode. If we consider an emergency calling situation to use cell phone, this wake-up time may not be acceptable. Therefore, an ultra-low-leakage technique that can save state even in non-active mode can be quite important in nanoscale technology VLSI.

In this dissertation, we use circuit as well as architectural techniques to reduce leakage power consumption. Especially, our technique can retain logic state and thus fast response time can be achieved even during non-active mode. The technique can be applicable to generic logic circuits as well memory, i.e., SRAM, since our technique can retain state.

In this chapter, some motivation for the importance of this research is provided. In the next chapter, we explain expressions, notation and background important for this thesis.

CHAPTER III

NOTATION AND BACKGOUND

In this chapter, we explain important notation and VLSI background used in this dissertation. First, we introduce subthreshold leakage power consumption on which our research focuses. Next, we explain the background underlying a particular leakage power model able to explain the stack effect, which is an important leakage reduction factor in our research. We then explain the body-bias effect. Furthermore, we explain subthreshold leakage power consumption of a conventional 6 Transistor (6-T) SRAM cell. Finally, we explain switching power and delay tradeoffs of CMOS circuits.

3.1 Leakage power

In this section, we explain notation and background relevant to leakage power consumption.

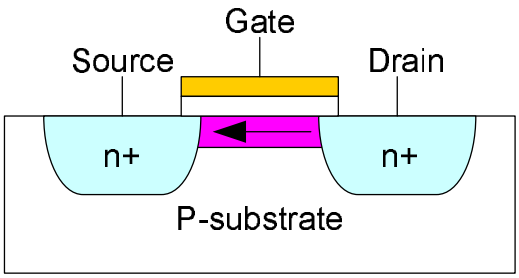


Figure 1: Subthreshold leakage of an nFET

Although dynamic power is dominant for technologies at 0.18μ and above, leakage (static) power consumption becomes another dominant factor for 0.13μ and below. One of the main contributors to static power consumption in CMOS is subthreshold leakage current shown in Figure 1, i.e., the drain to source current when the gate voltage is smaller than the transistor threshold voltage. Since subthreshold current increases exponentially as the threshold voltage decreases, nanoscale technologies with scaled down threshold voltages

will severely suffer from subthreshold leakage power consumption.

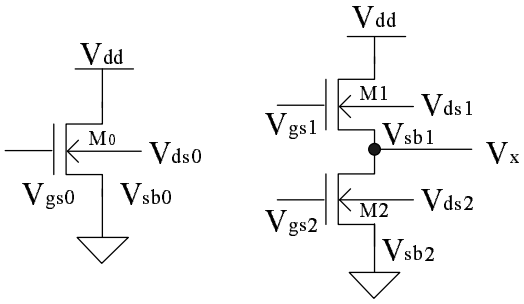


Figure 2: (a) A single transistor (left) and (b) stacked transistors (right)

Subthreshold leakage can be reduced by stacking transistors, i.e., taking advantage of the so-called "stack effect" [34]. The stack effect occurs when two or more stacked transistors are turned off together; the result is reduced leakage power consumption.

Let us explain an important stack effect leakage reduction model. The model we explain here is based on the leakage models in [34] and [46]. For a turned off single transistor shown in Figure 2(a), leakage current (I_{sub0}) can be expressed as follows:

$$I_{sub0} = Ae^{\frac{1}{nV_\theta}(V_{gs0}-V_{th0}-\gamma V_{sb0}+\eta V_{ds0})}(1-e^{-V_{ds0}/V_\theta}) \quad (5)$$

$$= Ae^{\frac{1}{nV_\theta}(-V_{th0}+\eta V_{dd})} \quad (6)$$

where $A = \mu_0 C_{ox}(W/L_{eff})V_\theta^2 e^{1.8}$, n is the subthreshold swing coefficient, and V_θ is the thermal voltage. V_{gs0}, V_{th0}, V_{bs0} and V_{ds0} are the gate-to-source voltage, the zero-bias threshold voltage, the base-to-source voltage and the drain-to-source voltage, respectively. γ is the body-bias effect coefficient, and η is the Drain Induced Barrier Lowering (DIBL) coefficient. μ is zero-bias mobility, C_{ox} is the gate-oxide capacitance, W is the width of the transistor, and L_{eff} is the effective channel length [60]. (Note that throughout this thesis we assume $\mu_n = 2\mu_p$, i.e., nMOS carrier mobility is twice pMOS carrier mobility. Also note that we use a W/L ratio based on a actual transistor size, in which way a W/L ratio properly characterizes circuit models used in this thesis.) We assume $1 >> e^{-V_{ds0}/V_\theta}$.

Let us assume that the two stacked transistors (M_1 and M_2) in Figure 2(b) are turned off. We also assume that the transistor width of each of M_1 and M_2 is the same as the transistor width of M_0 ($W_{M0} = W_{M1} = W_{M2}$). Two leakage currents I_{sub1} of the transistor M_1 and I_{sub2} of the transistor M_2 can be expressed as follows:

$$\begin{aligned} I_{sub1} &= Ae^{\frac{1}{nV_\theta}(V_{gs1}-V_{th0}-\gamma V_{sb1}+\eta V_{ds1})}(1-e^{-V_{ds1}/V_\theta}) && (7)\\ &= Ae^{\frac{1}{nV_\theta}(-V_x-V_{th0}-\gamma V_x+\eta(V_{dd}-V_x))} && (8)\\ I_{sub2} &= Ae^{\frac{1}{nV_\theta}(V_{gs2}-V_{th0}-\gamma V_{sb2}+\eta V_{ds2})}(1-e^{-V_{ds2}/V_\theta}) && (9)\\ &= Ae^{\frac{1}{nV_\theta}(-V_{th0}+\eta V_x)}(1-e^{-V_x/V_\theta}). && (10) \end{aligned}$$

where V_x is the voltage at the node between $M1$ and $M2$, and we assume $1 >> e^{-V_{ds1}/V_\theta}$.

Now consider leakage current reduction between I_{sub0} and $I_{sub1}(= I_{sub2})$. The reduction factor X can be expressed as follows:

$$X = \frac{I_{sub0}}{I_{sub1}} = \frac{Ae^{\frac{1}{nV_\theta}(-V_{th0}+\eta V_{dd})}}{Ae^{\frac{1}{nV_\theta}(-V_x-V_{th0}-\gamma V_x+\eta(V_{dd}-V_x))}} = e^{\frac{V_x}{nV_\theta}(1+\gamma+\eta)} \quad (11)$$

V_x in Equation 11 can be derived by letting $I_{sub1} = I_{sub2}$ and by solving the following equation:

$$1 = e^{\frac{1}{nV_\theta}(\eta V_{dd}-V_x(1+2\eta+\gamma))} + e^{-V_x/V_\theta} \quad (12)$$

If all the parameters are known, we can calculate stack effect leakage power reduction using Equations 11 and 12. As an example, we consider leakage model parameter values targeting 0.5μ technology in Table 3 obtained from [34]. From Equation 12, we calculate

Table 3: Leakage model parameters (0.5μ tech)

Parameter	Value
V_{dd}	1V
V_{th}	0.2V
n (subthreshold slope coeffcient)	1.5
η (DIBL coeffcient)	0.05V/V
γ (body-bias effect coeffcient)	0.24V/V

$V_x = 0.0443V$, and from Equation 11, we obtain leakage reduction factor $X = 4.188$.

Although the reduction is 4.188X at 0.5μ technology, the reduction increases at nanoscale technology because η increases as technology feature size shrinks.

Threshold voltage of a CMOS transistor can be controlled using body bias. In general, we apply V_{dd} to the body (e.g., an n-well or n-tub) of pMOS and apply Gnd to a body (e.g., p-well or p-substrate) of nMOS. This condition, in which source voltage and body voltage of a transistor are the same, is called Zero-Body Bias (ZBB). Threshold voltage at ZBB is called ZBB threshold voltage. When body voltage is lower than source voltage by biasing negative voltage to body, this condition is called Reverse-Body Bias (RBB). Alternatively, when body voltage is higher than source voltage by biasing positive voltage to body, this condition is called Forward-Body Bias (FBB). When RBB is applied to a transistor, threshold voltage increases, and when FBB applied to a transistor, threshold voltage decreases. This phenomenon is called body-bias effect, and this is frequently used to control threshold voltage dynamically [72].

In this section, Section 3.1, we explained subthreshold leakage power consumption, the stack effect, and body-bias effects which can alter subthreshold leakage power consumption. In the next section, we explain leakage current of an SRAM cell.

3.2 SRAM cell leakage paths

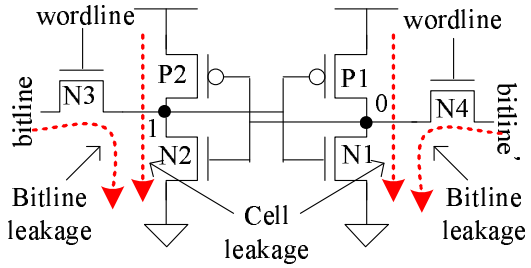


Figure 3: SRAM cell leakage paths

In this section, we explain the major subthreshold leakage components in a 6-T SRAM cell. The subthreshold leakage current in an SRAM cell is typically categorized into two kinds [37] as shown in Figure 3: (i) cell leakage current that flows from V_{dd} to Gnd internal

to the cell and (ii) `bitline` leakage current that flows from `bitline` (or `bitline'`) to Gnd.

Although an SRAM cell has two `bitline` leakage paths, the `bitline` leakage current and `bitline'` leakage current differ according to the value stored in the SRAM bit. If an SRAM cell holds '1' as shown in Figure 3, the `bitline` leakage current passing through N3 and N2 is effectively suppressed due to two reasons. First, after precharging `bitline` and `bitline'` both to '1,' the source voltage and the drain voltage of N3 are the same, and thus potentially no current flows through N3. Second, two stacked and turned off transistors (N2 and N3) induce the stack effect. Meanwhile, for this case where the SRAM bit holds value '1,' a large `bitline'` leakage current flows passing through N4 and N1. If, on the other hand, the SRAM cell holds '0,' a large `bitline` leakage current flows while `bitline'` leakage current is suppressed. Our results in Section 7.2.3 indicates that `bitline` leakage accounts for approximately 35% of SRAM cell leakage power consumption.

In this section, Section 3.2, we explain the two major types of leakage paths in an SRAM cell (cell leakage and `bitline` leakage). In next section, we explain tradeoffs between switching power and delay.

3.3 Switching power and delay tradeoffs

In this section, we explain tradeoffs between switching power and delay.

In CMOS, power consumption consists of leakage power and dynamic power – note that dynamic power includes both switching power and short-circuit power. Switching power is consumed when a gate charges its output load capacitance, and short-circuit power is consumed when a pull-up network and a pull-down network are on together for an instant while transistors are turning on and off. For 0.18μ channel lengths and above, leakage power is very small compared to dynamic power. Furthermore, short-circuit power is also less than 10% of the dynamic power for a typical CMOS design, and the ratio between

dynamic power and short-circuit power does not change as long as the ratio between supply voltage and threshold voltage remains the same [50]. Since, for 0.18μ and above, short-circuit power and leakage power are relatively small compared to switching power, CMOS power consumption of a particular CMOS gate under consideration can be represented by the following switching power ($P_{switching}$) equation for 0.18μ and above:

$$P_{switching} = p_t C_L V_{dd}^2 f \tag{13}$$

where C_L, V_{dd}, and f denote the load capacitance of a CMOS gate, the supply voltage and the clock frequency, respectively [15]. Notation p_t denotes the switching ratio of a gate output; this switching ratio represents the number of times the particular gate's output changes from Gnd to V_{dd} per second – please note that when output capacitance discharges from V_{dd} to Gnd, switching power is not consumed because power from V_{dd} is not used (e.g., discharging to Gnd does not consume battery power). The switching ratio varies according to the input vectors and benchmark programs, and thus an average value of each benchmark may be used as a switching ratio.

Equation 13 shows that lowering V_{dd} decreases CMOS switching power consumption quadratically. However, this power reduction unfortunately entails an increase in the gate delay in a CMOS circuit as shown in following approximated equation:

$$T_d \propto \frac{V_{dd}}{(V_{dd} - V_{th})^\alpha} \tag{14}$$

where T_d, V_{th}, and α denote the gate delay in a CMOS circuit, the threshold voltage and velocity saturation index of a transistor, respectively. It is well-known that while α has values close to 2 for above 2.0μ, for 0.25μ α is between 1.3 and 1.5, and for below 0.1μ α is close to 1 [35], [59]. However, instead of scaling down a α value along with the technology feature size, CMOS technology may take a constant α value to avoid the hot-carrier related problem [59]. A constant α value could be accomplished by changing V_{th} because α is a function of gate-source voltage [8]. If we scale down V_{dd}, switching power in Equation 13 decreases, while the gate delay in Equation 14 increases. Therefore, CMOS

circuit speed can be traded with switching power consumption as shown in Equations 13 and 14.

When there exist tradeoffs between multiple criteria, e.g., power and delay, we may say one design is better than another design in specific criteria. The point of design space is called a Pareto point if there is no point with one or more inferior objective [40].

In this thesis we estimate leakage power consumption by measuring static power when transistors are not switching. Furthermore, we estimate active power consumption by measuring power when transistors are switching. This active power include dynamic power consumption and leakage power consumption.

In this chapter we explained important notation and VLSI background used in this thesis. In the next section, we explain previous low-power research related to our research.

CHAPTER IV

PREVIOUS WORK

In this chapter, we review important prior work that is closely related to our research. Furthermore, the previous work is compared to our research. First we explore the prior work targeting leakage power reduction, and then we finish by studying previous techniques relevant to low-power pipelined caches.

4.1 Static Power Reduction VLSI Research

In this section, we discuss previous low-power techniques that primarily target reducing leakage power consumption of CMOS circuits. Techniques for leakage power reduction can be grouped into two categories: (i) state-saving techniques where circuit state (present value) is retained and (ii) state-destructive techniques where the current Boolean output value of the circuit might be lost [38]. A state-saving technique has an advantage over a state-destructive technique in that with a state-saving technique the circuitry can immediately resume operation at a point much later in time without having to somehow regenerate state. We characterize each low-leakage technique according to this criterion. We study low-leakage techniques for generic logic circuits followed by low-leakage SRAM designs separately.

4.1.1 Static Power Reduction Research for Generic Logic Circuits

This section explains previously proposed low-leakage techniques for generic logic circuits. As introduced, previously proposed work can be divided into techniques that either (i) save state or (ii) destroy state. Although our research focuses on techniques which save state, we also review the state-destructive techniques for the purposes of comparison.

4.1.1.1 Sleep transistor

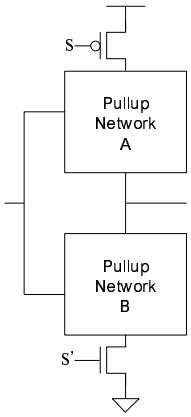


Figure 4: Sleep transistor technique

State-destructive techniques cut off transistor (pull-up or pull-down or both) networks from supply voltage or ground using sleep transistors [45]. These types of techniques are also called gated-V_{dd} and gated-Gnd (note that a gated clock is generally used for dynamic power reduction). Motoh et al. propose a technique they call Multi-Threshold-Voltage CMOS (MTCMOS) [45], which adds high-V_{th} sleep transistors between pull-up networks and V_{dd} and between pull-down networks and ground as shown in Figure 4 while logic circuits use low-V_{th} transistors in order to maintain fast logic switching speeds. The sleep transistors are turned off when the logic circuits are not in use. By isolating the logic networks using sleep transistors, the sleep transistor technique dramatically reduces leakage power during sleep mode. However, the additional sleep transistors increase area and delay. Furthermore, the pull-up and pull-down networks will have floating values and thus will lose state during sleep mode. These floating values significantly impact the wake-up time and energy of the sleep technique due to the requirement to recharge transistors which lost state during sleep (this issue is nontrivial, especially for registers and flip-flops).

Comparison with prior work using sleep transistors

The sleep transistor technique and the sleepy stack technique both achieve roughly the same (100X or more) static power savings over conventional CMOS. However, unlike the

sleep transistor technique, the sleepy stack technique saves logic state during low leakage mode (sleep mode), and this is a significant advantage over the state-destructive sleep transistor technique. The sleep transistor technique requires non-negligible power consumption to restore lost state. Further, the wake-up time of the sleep transistor technique is significant, while the sleepy stack technique needs only a very small extra wake-up time (a few clock cycles).

4.1.1.2 Zigzag

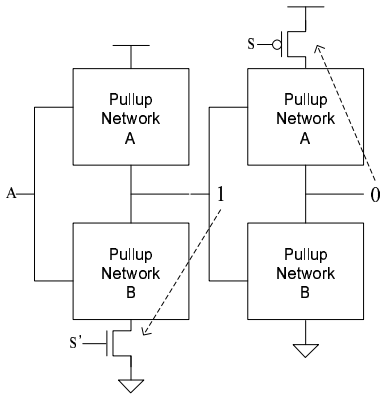


Figure 5: Zigzag technique

To reduce the wake-up cost of the sleep transistor technique, the zigzag technique is introduced [41]. The zigzag technique reduces the wake-up overhead by choosing a particular circuit state (e.g., corresponding to a "reset") and then, for the exact circuit state chosen, turning off the pull-down network for each gate whose output is high while conversely turning off the pull-up network for each gate whose output is low. For example, the zigzag technique in Figure 5 assume that the input 'A' is asserted such that the output values result as shown in the figure. If the output is '1,' then a pull-down sleep transistor is applied; if the output is '0,' then a pull-up sleep transistor is applied. By applying, prior to going to sleep, the particular input pattern chosen prior to chip fabrication, the zigzag technique can prevent floating. Although the zigzag technique retains the particular state chosen prior to chip fabrication, any other arbitrary state during regular operation is lost in

power-down mode.

Comparison with prior work using zigzag

Although the zigzag technique can reduce wake-up cost, the zigzag technique still loses state. Thus, any particular state (from prior to going to sleep) which is needed upon wakeup must be regenerated somehow. Also, the zigzag technique may need extra circuitry to generate a specific input vector (in case reset values are not used for the sleep mode input vector).

4.1.1.3 Forced stack

Another technique to reduce leakage power is transistor stacking. Transistor stacking exploits the stack effect explained in Chapter 3; the stack effect results in substantial subthreshold leakage current reduction when two or more stacked transistors are turned off together.

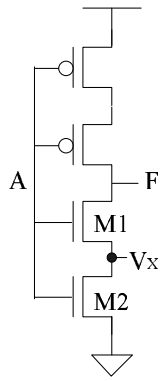


Figure 6: Forced stack inverter

Example 1: The stack effect can be understood from the forced stack inverter example shown in Figure 6. Unlike a generic CMOS inverter, this forced stack inverter consists of two pull-up transistors and two pull-down transistors. All inputs share the same input 'A.' If $A = 0$, then both transistors M1 and M2 are turned off. Due to the internal resistance of M2, the intermediate node voltage V_x is higher than Gnd. The positive potential of V_x results in a negative gate-source voltage (V_{gs}) for $M1$ and negative source-base voltage (V_{sb}) for $M1$. Furthermore, $M1$ has a reduced

drain-source voltage (V_{ds}), which degrades the Drain Induced Barrier Lowering (DIBL) effect [18]. All three effects together change the leakage reduction factor X in Equation 11 (see Chapter 3), reducing leakage current by an order of magnitude for today's channel lengths (0.18μ, 0.13μ, 0.10μ and 0.07μ) [7]. □

Narendra et al. study the effectiveness of the stack effect including effects from increasing the channel length [47]. Since forced stacking of what previously was a single transistor increases delay, Johnson et al. propose an algorithm that finds circuit input vectors that maximize stacked transistors of existing complex logic [32].

Comparison with prior work using the forced stack approach

Compared to the forced stack technique, the sleepy stack technique potentially achieves more power savings (e.g., 100X compared with 10X for the stack effect) because the sleepy stack can use high-V_{th} transistors in key places. The forced stack technique cannot use high-V_{th} transistors without dramatic delay increase (larger than 5X delay increase compared to conventional CMOS).

4.1.2 Static Power Reduction Research for SRAM

In this section, we discuss state-of-the-art low-power memory techniques, especially SRAM and cache techniques on which our research focuses.

4.1.2.1 High-V_{th} SRAM Cell

One easy way to reduce leakage power consumption is by adopting high-V_{th} transistors for all SRAM cell transistors. This solution is simple but incurs delay increase (our experiments indicate that doubling V_{th} for all six transistors increases delay by 2.5X using 0.07μ technology).

Comparison with prior work using high-V_{th} SRAM cells

Compared to the high-V_{th} SRAM cell, the sleepy stack SRAM cell achieves 2.5X leakage power reduction with the same delay (see Section 7.2.4). Alternatively, the sleepy

stack SRAM cell achieves 5.13X leakage power reduction with 32% delay increase when compared with high-V_{th} SRAM (again, see Section 7.2.4).

4.1.2.2 Asymmetric-Cell Cache (ACC)

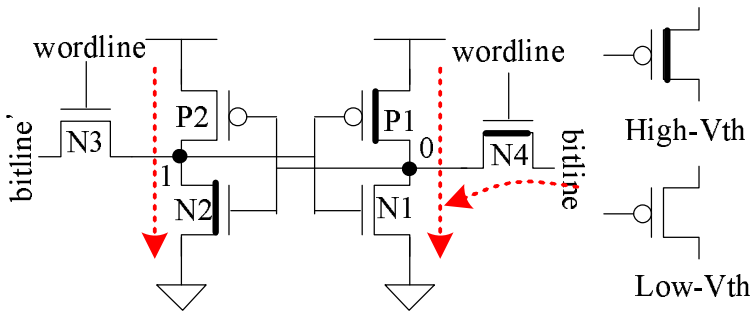


Figure 7: Asymmetric SRAM cell

Azizi et al. observe that in normal programs, most of the bits in a cache are zeros. Therefore, Azizi et al. propose an Asymmetric-Cell Cache (ACC), which partially applies high-V_{th} transistors in an SRAM cell to save leakage power if the SRAM cell is in the zero state [5]. Figure 7 shows an asymmetric SRAM cell. If the cell stores a '0,' then the transistors P1 and N2 dissipate cell leakage while transistor N4 dissipates `bitline` leakage power. Therefore, if we use high-V_{th} transistors for P1, N2 and N4, we can reduce leakage power as long as the cell stores '0.' Azizi et al. also propose a new sense amplifier to offset performance degradation due to the high-V_{th} transistors. The new sense amplifier may induce area increase since the transistor count of the new sense amplifier is 55% larger than a conventional sense amplifier.

Comparison with prior work using dual-V_{th} cells

The Asymmetric-Cell Cache leakage power savings are quite limited in case of a benchmark which fills SRAM with mostly non-zero values. Compared to the Asymmetric-Cell Cache, the sleepy stack achieves large leakage reduction regardless of stored values, i.e., the sleepy stack technique saves leakage power in cases with benchmarks with a large number of '1' values. Moreover, even in case with benchmarks with a large number of '0' values,

ACC achieves 40X leakage reduction while our sleepy stack achieves leakage reduction of 100X or more.

4.1.2.3 ABC-MTCMOS

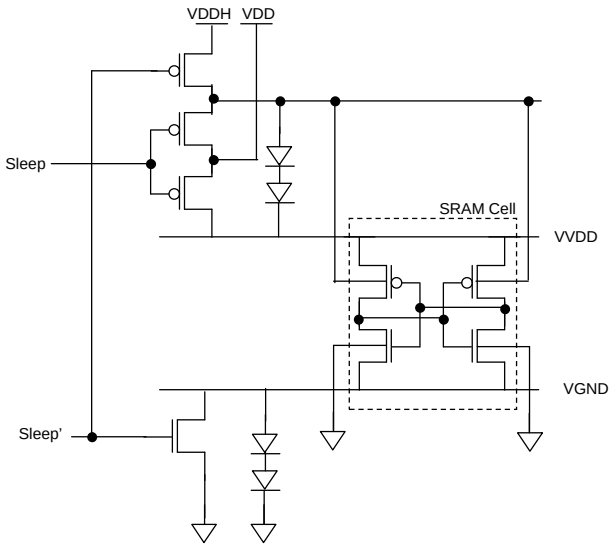


Figure 8: ABC-MTCMOS

Nii et al. propose a low-leakage SRAM design named Auto-Backgate-Controlled Multi-Threshold CMOS (ABC-MTCMOS) [49] based on the conventional MTCMOS technique explained in Section 4.1.1.1. Unlike straightforward MTCMOS, in which logic circuits float and thus potentially lose state during sleep mode, ABC-MTCMOS shown in Figure 8 uses Reverse-Body Bias (RBB) to reduce leakage power consumption and thus maintains state (no floating).

Let us take a specific example from [49], which targets 0.35μ technology. During active mode, Sleep='0' and Sleep'='1' are asserted, and thus the SRAM cell is connected to V_{dd}, which is $1V$, and Gnd. However, during sleep mode, Sleep='1' and Sleep'='0' are asserted; then, the body of the pFET of the SRAM cell is connected to V_{ddh}, which is $3.3V$. Furthermore, during sleep mode the V_{ddh} connection supplies power to VV_{dd} through two diodes. The voltage drop across a diode is $0.5V$ (assumed by Nii et al. [49]), and thus in sleep mode the source voltage of the pFETs is $2.3V$. Therefore, this structure forms reverse

source-body bias for pFETs. This reverse bias increases pFET V_{th} within the SRAM cell. Meanwhile, in sleep mode, $VGnd$ is connected to Gnd using two diodes. The the voltage drop across the two diodes is $1V$, and thus $VGnd = 1V$. Therefore, the nFETs of the SRAM cell also experience reverse source-body bias, which increases nFET V_{th} within the SRAM cell.

During active mode, ABC-MTCMOS use Zero-Body Bias (ZBB), and thus ABC-MTCMOS maintains performance. Meanwhile, during sleep mode, ABC-MTCMOS applies RBB for both pFETs and nFETs. RBB increases threshold voltage without losing logic state. This increased threshold voltage reduces leakage power consumption during sleep mode.

Comparison with prior work using ABC-MTCMOS

The ABC-MTCMOS technique requires an additional supply voltage throughout the whole SRAM cell array. More importantly, since the ABC-MTCMOS technique needs to charge large wells (e.g., larger n-wells), ABC-MTCMOS requires significant transition time and power consumption, which the sleepy stack does not need. Further, a large electric field across the transistors may affect reliability [26]. Finally, ABC-MTCMOS achieves similar leakage power reduction as our sleepy stack technique (1000X over conventional CMOS).

4.1.2.4 Gated-V_{dd} and gated-Gnd SRAM cell

Sleep transistors explained in Section 4.1.1.1 can also be used for SRAM cell design. Using sleep transistors, the gated-V_{dd} SRAM cell blocks pull-up networks from the V_{dd} rail (pMOS gated-V_{dd}) and/or blocks pull-down networks from the Gnd rail (nMOS gated-V_{dd}) [54]. The gated-V_{dd} SRAM cell achieves low-leakage power consumption from both the stack effect and high-V_{th} sleep transistors. However, this gated-V_{dd} SRAM cell [54] loses state when the sleep transistors are turned off. To overcome this problem, Powell et al. propose the Dynamically ResIzable instruction cache (DRI i-cache), an integration of

circuit and architecture techniques [54]. Considering an SRAM cell, Powell et al. study the effect of nMOS gated-V_{dd} and pMOS gated-V_{dd} in terms of energy, speed and area. (Please note that Powell et al. use the term "nMOS gated-V_{dd}" to indicate placing a sleep transistor between the pull-down network and Gnd; similarly, Powell et al. use the term "pMOS gated-V_{dd}" to indicate placing a sleep transistor between the pull-up network and V_{dd}.) Figure 9 shows an SRAM cell with nMOS gated-V_{dd}. The SRAM cell with nMOS gated-V_{dd} can suppress two kinds of leakage paths: (i) cell leakage paths and (ii) `bitline` leakage paths (see Section 3.2). Note that a pMOS gated-V_{dd} SRAM cell can be implemented with small area since the transistors in the pull-up network of an SRAM cell uses smaller transistor widths than the widths of transistors in the pull-down network. However, pMOS gated-V_{dd} cannot suppress the `bitline` leakage paths.

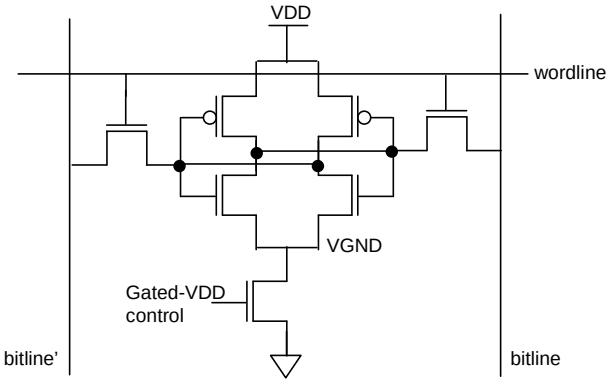


Figure 9: SRAM cell with nMOS gated-V_{dd}

Although the conventional nMOS gated-V_{dd} technique lose data, a nMOS gated-V_{dd} technique with carefully sized transistors can retain the original data. Agarwal et al. study the data retention capability of nMOS gated-V_{dd} technique and propose Data Retention Gated-Ground Cache (DRG-Cache) [1]. When the nMOS gated-V_{dd} transistor in Figure 9 is turned off, the $VGnd$ node is charged up. If the $VGnd$ is not high enough to change the stored value, the cell will retain its value. Agarwal et al. study various retaining conditions including temperature, V_{th}, and gate size. However, since the $VGnd$ node does not hold value '0' firmly, the DRG-Cache design is vulnerable, and even a small induced charge

may change the stored value [21].

Comparison with prior work using gated-V_{dd}

Since SRAM is a memory circuit and prefers to maintain state even during sleep mode, the state-saving of the sleepy stack has a great advantage over the gated-V_{dd} technique. The sleepy stack does not require complex architectural technique, which the DRI cache uses, to alleviate performance penalty induced by the lost values. Compared to the DRG-Cache, the sleepy stack technique is safer in soft errors. If we consider that even some conventional caches adopts soft error recovery system [55], techniques that increase soft error could limit the usage.

4.1.2.5 Drowsy Cache

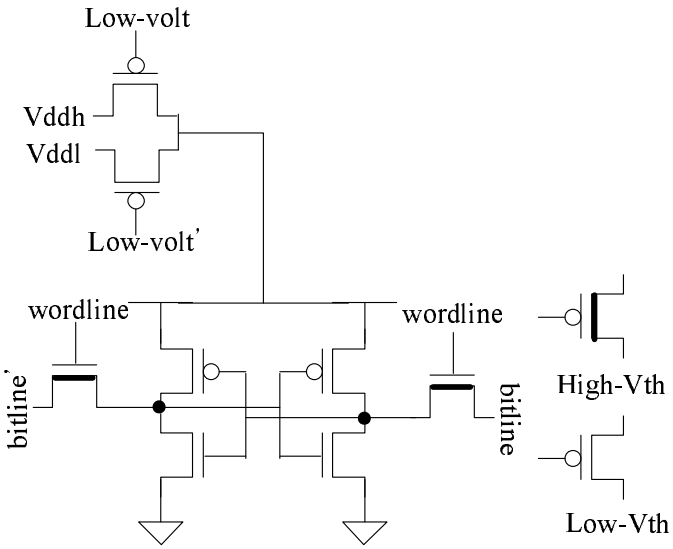


Figure 10: Drowsy cache

Flautner et al. propose the "drowsy cache" technique that switches V_{dd} dynamically [23]. The basic idea behind the drowsy cache is that the V_{dd} voltage value required to maintain state is only 1.5xV_{th}; e.g., in 0.07μ, if $V_{dd} = 1V$ and $V_{th} = 0.2V$, then the voltage required to maintain state is $0.3V$. Thus, the drowsy cache uses V_{ddl}, which is 1.5xV_{th}, during drowsy mode; during active mode, the drowsy cache uses $V_{ddh} (= V_{dd})$ as shown Figure 10. For short-channel devices such as 0.07μ channel length devices, leakage power increases

when drain voltage increases because increased drain voltage lowers the potential barrier at the channel region, thereby increasing subthreshold leakage current. This phenomenon is called Drain Induced Barrier Lowering (DIBL). Inversely, by lowering the supply voltage of a short-channel device, subthreshold leakage current can be suppressed. The drowsy cache lowers the supply voltage during drowsy mode and suppresses leakage current using DIBL. To control `bitline` leakage, the drowsy cache adopts high-V_{th} `wordline` transistors. The drowsy cache technique can retain stored data at a leakage power reduction of 86% or less (approximately 7X or less).

Comparison with prior work using drowsy cache

The drowsy cache reduces leakage power mainly using the DIBL effect, and thus the leakage reduction of the drowsy cache is limited and much smaller than the sleepy stack technique (approximately 7X compared to 100X or more).

4.1.2.6 Body Biasing

Unlike the drowsy cache, which scales V_{dd} dynamically, some techniques scale V_{th} dynamically using body-bias (–e.g., we have already seen ABC-MTCMOS in Section 4.1.2.3). Kim et al. propose Reverse Body-Biasing (RBB) SRAM, which applies negative voltage to a body (i.e., p-well in a deep n-well process) [37]. In deep n-well technology, each p-well is formed in an n-well, and thus p-wells are separated from the p-substrate. Using this technology, Kim et al. avoid having to charge up large p-substrate area. RBB SRAM adopts low-V_{th} and applies Zero Body-Biasing (ZBB) during active mode while scaling up V_{th} using RBB during sleep mode. By use of an approach similar to ABC-MTCMOS, Kim et al. implement RBB for the nMOS transistors of SRAM cells. When RBB is applied to pMOS, the leakage reduction achieved is relatively small, area overhead is large, and charging and discharging the p-well consumes large amount of extra energy (Kim et al. only consider deep n-well technology, e.g., twin-tub is not considered). However, the RBB SRAM technique still has large latency and overhead due to body bias transition, which charges and

discharges large p-well capacitance. Kim et al. also propose Forward Body-Biasing (FBB) SRAM. FBB applies positive voltage to the body of nFETs. FBB SRAM adopts high-V_{th} transistors and uses FBB during active mode to achieve high speed while using ZBB during sleep mode to suppress leakage current [36]. To overcome charging/discharging large p-substrate, the FBB SRAM technique divides SRAM into subarrays by adopting deep n-well technology, which isolates p-wells from p-substrate using n-wells.

Comparison with prior work using body biasing

The body biasing techniques typically require charging large substrate or deep n-well technology, which the sleepy stack does not need. The body biasing technique achieves up to 72% leakage reduction while our sleepy stack achieves leakage reduction of 100X or more.

Until now, we explained previous low-leakage techniques that can be applicable to generic logic circuit and/or memory. The low-leakage techniques can be categorized into two, (i) state saving, in which out research falls, and (ii) state destructive. In the next section, we explain previous work that is related to power reduction using voltage scaling and pipelining, approaches which our research also utilizes.

4.2 Power Reduction Research Using Voltage Scaling and Pipelining Caches

Our Low-Power Pipelined Cache (LPPC), which is one of our contributions (see Chapter 8), uses pipelining and voltage scaling to save dynamic power and/or static power. In this section, we discuss prior work that reduces power consumption primarily using either (i) static voltage scaling (i.e., voltage values are set at design time and never change during circuit operation) or (ii) pipelined caches.

4.2.1 HYPER-LP

Chandrakasan et al. study parallelization and pipelining techniques to save power consumption [15]. Parallelization and pipelining increase throughput by means of circuit duplication and circuit pipelining. Chandrakasan et al. lower supply voltage of circuits to save power consumption. Lowering supply voltage increases circuit delay. Chandrakasan et al. lower supply voltage until the throughput returns to the throughput before parallelization and/or pipelining. Since switching power consumption is quadratically proportional to supply voltage, this method potentially achieves large power savings while maintaining throughput. However, duplication and/or pipelining circuits have some drawbacks. The parallelization technique incurs significant area increase due to duplicated circuits and wires connecting them, resulting in extra power consumption. The pipelining technique also increases area due to latches between pipeline stages (the latches are used to store intermediate signals).

Although there exist some limitations to using parallelization and pipelining, the parallelization and pipelining techniques have driven power optimized high-level synthesis. Chandrakasan et al. introduce an automated high-level synthesis system, HYPER-LP [14], which explores concurrency in circuits to reduce the delay of the critical path in circuits by means of loop unrolling or pipelining. The reduced delay enables circuits to operate at a lower V_{dd}, and thus the lowered V_{dd} increases gate delay while reducing power as well. In high-level synthesis techniques [40], the concurrency exploration is typically used to reduce a critical path to enhance throughput. On the other hand, the high-level synthesis transformations for power optimization typically tradeoff between switching capacitance and voltage, while maintaining throughput [13].

Comparison with prior work using HYPER-LP

HYPER-LP mainly focuses on dynamic power reduction of logic circuits and does not consider power reduction of SRAM. Meanwhile, our LPPC focuses on cache power reduction.

4.2.2 Multiple V_{dd} and V_{th} Optimization

High-level synthesis based on voltage scaling can be extended to circuits with multiple supply voltages. A multiple voltage supply system can, for example, assign a low supply voltage (V_{ddl}) to non-critical paths while assigning a high supply voltage (V_{ddh}) to critical paths. The voltage level of each operation unit (each collection of logic circuits) is decided so that the power is reduced while preserving timing constraints: this is called the Multiple-Voltage Scheduling (MVS) [16]. Raje et al. propose a behavioral level MVS algorithm that uses a data flow graph to abstract a system; thus, an algorithm can be applied to minimize power consumption at the system or chip level [56].

In a multiple-V_{dd} system, the co-existence of multiple voltages in circuits potentially induces two problems. One is extra wiring needed to properly supply multiple V_{dd} values, potentially causing large area overhead. The other problem is placement of level converters. If a V_{ddl} gate drives the input of a V_{ddh} gate, the voltage level of the output of the V_{ddl} gate is not high enough to drive the input of the V_{ddh} gate; thus, if no level converter is used, the incompletely cut-off pMOS transistor of the V_{ddh} gate may incur static current flowing from V_{ddh} to ground (Gnd). This phenomenon can be prevented by placing a level converter that shifts the voltage level of the V_{ddl} gate output to V_{ddh}. These two problems are potentially serious for V_{dd} optimization because many extra wires and level converters may be required. Therefore, Johnson and Chang tackle the MVS problem with the consideration of level converters [16], [33].

While [16], [33] and [56] focus on solutions within high-level synthesis frameworks, Usami and Horowitz propose clustered voltage scaling, which handles level converter overhead in gate placement. Clustered voltage scaling minimizes the number of level converters by clustering gates having the same supply voltage and placing V_{ddh} gate clusters before V_{ddl} gate clusters if possible [69]. Usami et al. also tackle the placement problem of wires carrying different voltages by placing V_{ddh} and V_{ddl} wires row-by-row [70]. In placing gates using different supply voltages, the easier way is to place V_{ddh} and V_{ddl} gates in two

separate areas, which is called area-by-area placement. However, area-by-area placement requires long interconnections between V_{ddh} and V_{ddl} cells. The row-by-row scheme first places cells without considering voltages and then chooses the voltage level of each V_{dd} wire based on the majority of cells. The cells in a row of a different V_{dd} value (e.g., V_{ddl}) are relocated to the nearest row where cells use the same V_{dd} (e.g., V_{ddl}).

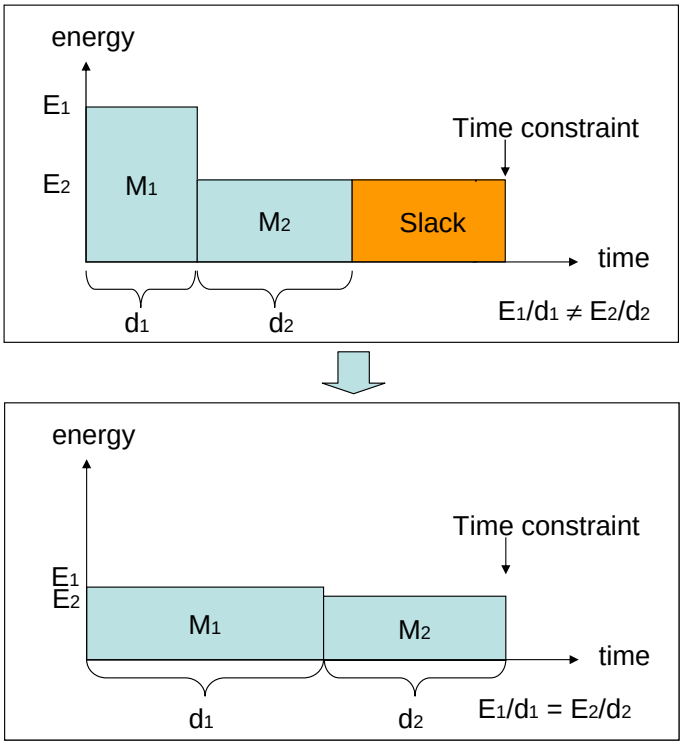


Figure 11: Before (up) and after (down) applying EDR paradigm

Choi et al. propose a new energy minimization metric they name the Energy-Delay Ratio (EDR) paradigm [19]. The EDR paradigm claims that total energy consumption is minimized when the energy-delay ratios of each module are the same. Choi et al. target 0.25μ technology in which switching energy dominates. In Figure 11, module $M1$ has energy $E1$ and delay $d1$ while module $M2$ has energy $E2$ and delay $d2$; the top box in Figure 11 shows the situation prior to applying the EDR paradigm. Since $d1 + d2$ is less than the time constraint, we can distribute the slack, the time difference between the time constraint and circuit delay, to achieve power reduction by means of lowering supply voltage.

The EDR is the metric that can be used to distribute slack to achieve minimum total energy consumption as shown in the bottom box in Figure 11. The EDR paradigm is used to minimize dynamic power by configuring V_{dd}, V_{th} and transistor width.

The techniques mentioned above are primarily concerned with dynamic power. As leakage power increases with shrinking feature size, researchers consider multiple V_{th} values as a solution. Wei et al. propose dual-V_{th} circuit optimization, which applies lower V_{th} for transistors in critical paths and higher V_{th} for non-critical paths [73]. Roy et al. extend the problem to multiple V_{dd} and V_{th} optimization to control both dynamic and static power concurrently [58]. Choi et al. develop a post-layout power optimization algorithm including V_{dd}, V_{th}, and transistor width based on the EDR paradigm [20]. Diril et al. also propose V_{dd} and V_{th} assignment algorithm based on the EDR paradigm to reduce dynamic power as well as static power [22]. Although Diril et al. ignore the level converter issue, Srivastava et al. propose a multiple V_{dd}, multiple V_{th} assignment algorithm which considers the level converter issue [63].

Comparison with prior work using multiple V_{dd} and V_{th} optimization

Multiple V_{dd} and V_{th} optimization techniques discussed in this section mainly focus on power reduction of logic circuits. Meanwhile, our LPPC focuses on power reduction of SRAM. Although multiple V_{dd} and V_{th} optimization techniques change V_{th} without changing circuit structure to reduce static power, the techniques discussed in Section 4.1.1 change circuit structures as well as V_{th}; as a result, while prior V_{th} optimization techniques cannot save much power for gates on the critical path, our sleepy stack approach for logic circuits can potentially apply to the critical path as well.

4.2.3 Low-Power Pipelined Cache

Chappell el al. broke down a cache into multiple segments and pipelined the segments [17]. Due to the increased parallelism, this pipelined cache technique can reduce cache access delay thus improving performance. However, Chappell el al. only address performance,

and no power reduction technique is mentioned [17].

Agarwal et al. proposed high bandwidth pipelined cache by breaking down caches and placing latches [2]. Although Agarwal et al. give some energy results, the primary concern of their work is not to reduce power but to improve performance.

Gunadi et al. proposed a bit-slice-pipelined cache to reduce power consumption by enabling the row decoder only for the necessary subbank [24]. Pipelining is used to offset the performance degradation due to bit slicing. However, this technique only addresses dynamic power reduction targeting 0.18μ technology in which leakage power is not dominant.

Comparison with prior work relevant to low-power pipelined caches

Prior pipelined caches mentioned above mainly focus on performance improvement. Although Gunadi et al. address power reduction by enabling only the requested subbank, they do not use voltage scaling and do not mention leakage power at all. Although Gunadi's pipelined cache saves larger dynamic power than our LPPC, we are using static voltage scaling which can be used together with Gunadi's pipelined cache. Meanwhile, our LPPC can be applicable to static power reduction for SRAM.

4.3 Summary

In this chapter, we discussed previous work related to the sleepy stack and LPPC techniques. Sleepy stack generic logic circuits achieve more power static savings (typically, 10X or more) than the forced stack approach while saving exact logic state (the sleep transistor technique and the zigzag technique have roughly equal static power reduction as sleepy stack but unfortunately lose original logic state). Our sleepy stack SRAM cell can achieve more power savings than high-V_{th} SRAM cell, ACC and drowsy cache. Furthermore, the sleepy stack SRAM does not require large transition time and transition power consumption unlike ABC-MTCMOS, and the sleepy stack SRAM does not require any special CMOS process, which body biasing techniques typically require. Finally, there is

no known prior pipelined cache technique proposed to save leakage power consumption while our LPPC achieves leakage power reduction using cache pipelining, and can even be combined with sleepy stack as we will show toward the end of this thesis (Chapter 10).

In the next section, we will discuss the sleepy stack structure and sleepy stack operation.

CHAPTER V

SLEEPY STACK STRUCTURE

In this chapter, we introduce our new leakage power reduction technique we name "sleepy stack." The sleepy stack technique has a combined structure of the forced stack technique and the sleep transistor technique. However, unlike the sleep transistor technique, the sleepy stack technique retains the exact logic state; and, unlike the forced stack technique, the sleepy stack technique can utilize high-V_{th} transistors without 5X (or greater) delay penalties. Therefore, far better than any prior approach known to this thesis author, the sleepy stack technique can achieve ultra-low leakage power consumption while saving state.

We first explain the structure of the sleepy stack technique using an inverter. Then we describe the details of sleepy stack operation in active mode and sleep mode. The advantages of the sleepy stack technique over the forced stack technique and the sleep transistor technique are explored. Finally, we derive a first order delay model that compares the sleepy stack technique to the forced stack technique analytically.

5.1 Sleepy stack approach

In this section, we explain our sleepy stack structure comparing to the forced stack technique and the sleep transistor technique. The details of the sleepy stack inverter are described as an example. Two operation modes, active mode and sleep mode, of the sleepy stack technique are explored.

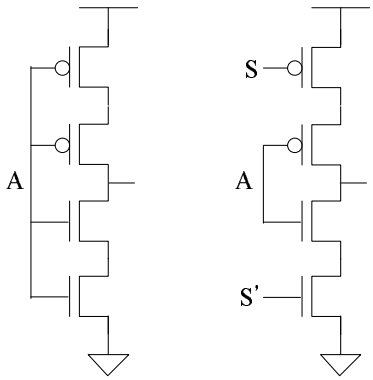


Figure 12: (a) Forced stack inverter (left) and (b) Sleep transistor inverter (right)

5.1.1 Sleepy stack structure

The sleepy stack structure has a combined structure of the forced stack and the sleep transistor techniques. Although we covered those two techniques in Section 4.1.1, we explain the forced stack and the sleep transistor inverters here for the purposes of comparison with a sleepy stack inverter. Figure 12(a) depicts a forced stack inverter, and Figure 12(b) depicts a sleep transistor inverter. The forced stack inverter breaks existing transistors into two transistors and forces a stack structure to take advantage of the stack effect; this is shown in Figure 12(a). Meanwhile, the sleep transistor inverter shown in Figure 12(b) isolates existing logic networks using sleep transistors. The stack structure in Figure 12(b) saves leakage power consumption during sleep mode. This sleep transistor technique frequently uses high-V_{th} sleep transistors (the transistors controlled by S and S') to achieve larger leakage power reduction.

The sleepy stack technique has a structure merging the forced stack technique and the sleep transistor technique. Figure 13 shows a sleepy stack inverter. The sleepy stack technique divides existing transistors into two transistors each typically with the same width W_1 half the size of the original single transistor's width W_0 (i.e., $W_1 = W_0/2$), thus maintaining equivalent input capacitance. The sleepy stack inverter in Figure 13(a) uses $W/L = 3$ for the pull-up transistors and $W/L = 1.5$ for the pull-down transistors, while a conventional inverter with the same input capacitance would use $W/L = 6$ for the pull-up

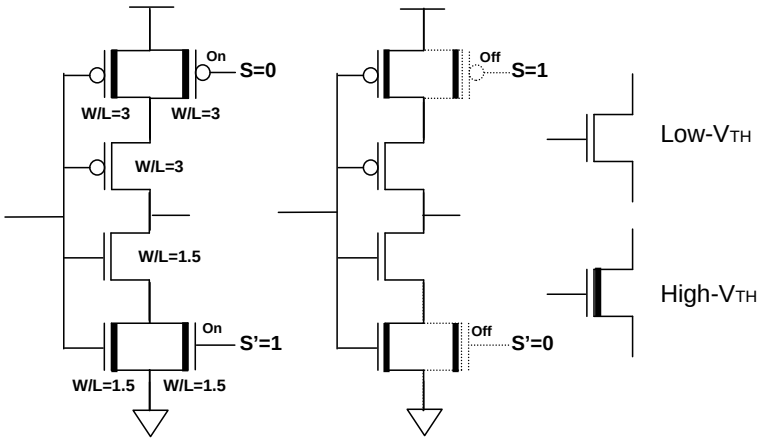


Figure 13: (a) Sleepy stack active mode (left) and (b) sleep mode (right)

transistor and $W/L = 3$ for the pull-down transistor (assuming $\mu_n = 2\mu_p$). Then sleep transistors are added in parallel to one of the transistors in each set of two stacked transistors. We use half size transistor width of the original transistor (i.e., we use $W_0/2$) for the sleep transistor width of the sleepy stack. Although we use $W_0/2$ for the width of the sleep transistor, changing the sleep transistor width may provide additional tradeoffs between delay, power and area. However, in this thesis we mainly focus on applying the sleepy stack structure with $W_2/2$ sleep transistor widths to generic logic circuits and SRAM while varying technology feature size, threshold voltage and temperature – although for SRAM, where transistor width is particularly critical and high-impact, we do vary all SRAM transistor widths (including sleep transistors).

5.1.2 Sleepy stack operation

Now we explain how the sleepy stack works during active mode and during sleep mode. Also, we explain leakage power saving using the sleepy stack structure.

The sleep transistors of the sleepy stack operate similar to the sleep transistors used in the sleep transistor technique in which sleep transistors are turned on during active mode and turned off during sleep mode. Figure 13 depicts the sleepy stack operation using a sleepy stack inverter. During active mode (Figure 13(a)), $S = 0$ and $S' = 1$ are asserted, and thus all sleep transistors are turned on. This sleepy stack structure can potentially

reduce circuit delay in two ways. First, since the sleep transistors are always on during active mode, the sleepy stack structure achieves faster switching time than the forced stack structure; specifically, in Figure 13(a), at each sleep transistor drain, the voltage value connected to the sleep transistor source is always ready and available at the sleep transistor drain, and thus current flow is immediately available to the low-V_{th} transistors connected to the gate output regardless of the status of each transistor in parallel to the sleep transistors. Furthermore, we can use high-V_{th} transistors (which are slow but 1000X or so less leaky), for the sleep transistors and the transistors parallel to the sleep transistors (see Figure 13) without incurring large delay increase.

During sleep mode (Figure 13(b)), $S = 1$ and $S' = 0$ are asserted, and so both of the sleep transistors are turned off. Although the sleep transistors are turned off, the sleepy stack structure maintains exact logic state. The leakage reduction of the sleepy stack structure occurs in two ways. First, leakage power is suppressed by high-V_{th} transistors, which are applied to the sleep transistors and the transistors parallel to the sleep transistors. Second, two stacked and turned off transistors induce the stack effect, which also suppresses leakage power consumption. By combining these two effects, the sleepy stack structure achieves ultra-low leakage power consumption during sleep mode while retaining exact logic state. The price for this, however, is increased area.

We will derive an analytical delay model of the sleepy stack inverter and compare the sleepy stack technique to the forced stack inverter in the next section. This analytical comparison can be skipped if desired. The detailed experimental methodology and the results will be presented in Chapter 6.

5.2 Analytical comparison of sleepy stack inverter vs. forced stack inverter

In this section, the analytical delay model of a sleepy stack inverter are explained and compared to a forced stack inverter, the best prior state-saving leakage reduction technique

we could find.

5.2.1 Delay model

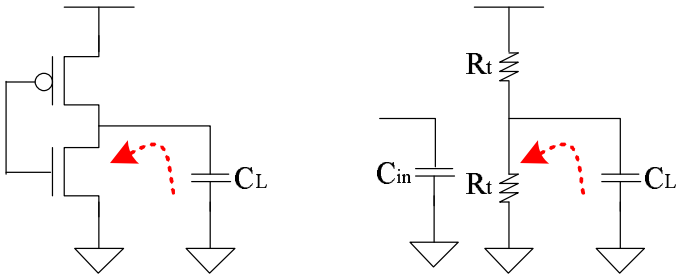


Figure 14: (a) Inverter logic circuit (left) and (b) RC equivalent circuit (right)

Generally the transistor delay of a conventional inverter shown in Figure 14 can be expressed using the following equation:

$$T_{d0} = C_L R_t, \tag{15}$$

where C_L is the load capacitance and R_t is the transistor resistance. C_{in} in Figure 14(b) indicates input capacitance. Although the non-saturation mode equation is complicated, we can predict the adequate first-order gate delay from Equation 15 [15].

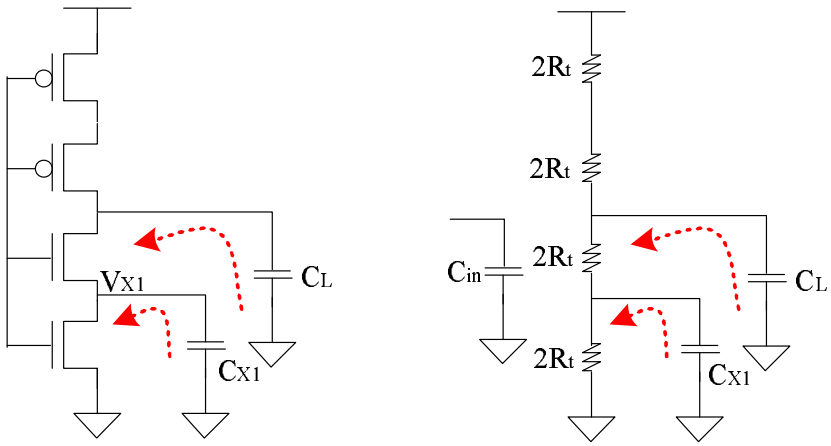


Figure 15: (a) Forced stack technique inverter (left) and (b) RC equivalent circuit (right)

Now we derive the delay of the inverter with the forced stack technique shown in Figure 15. Since we assume that we break each existing transistor into two half sized transistors (see Section 5.1.1), the resistance of each transistor of the forced stack technique

is doubled, i.e., $2R_t$, compared to the standard inverter; furthermore, in this way we can maintain input capacitance equal to Figure 14(b). In Figure 15, C_{x1} is internal node capacitance between the two pull-down transistors. Using the Elmore equation [72], we can express the delay of the forced stack inverter as follows:

$$T_{d1} = (2R_t + 2R_t)C_L + 2R_tC_{x1} \quad (16)$$

$$= 4R_tC_L + 2R_tC_{x1}. \quad (17)$$

Similarly, we can depict the sleepy stack inverter and its RC equivalent circuit as shown in Figure 16. Two extra sleep transistors are added and each sleep transistor has a resistance of $2R_t$ (as discussed in Section 5.1.1, please note that increasing sleep transistor width reduces the sleep transistor resistance further – however, let us continue with the approach of Section 5.1). The internal node capacitance is C_{x2}.

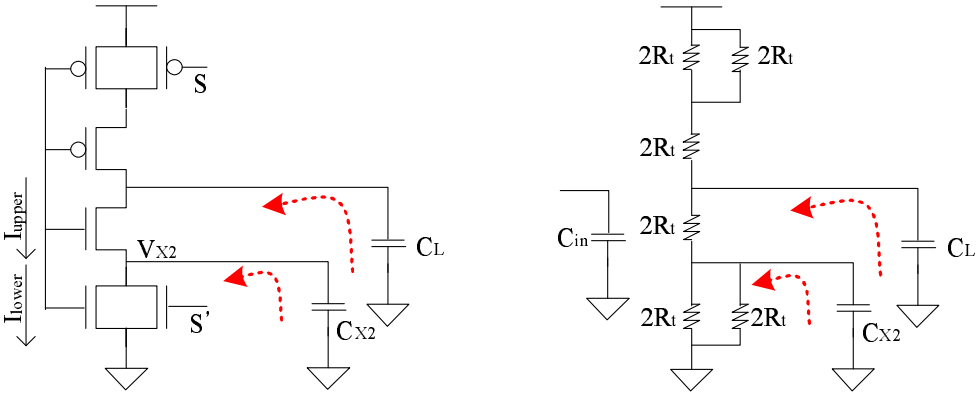


Figure 16: (a) Sleepy stack technique inverter (left) and (b) RC equivalent circuit (right)

Using the Elmore equation, we can derive the transistor delay of the sleepy stack inverter as follows:

$$T_{d2} = (2R_t + R_t)C_L + R_tC_{x2} \quad (18)$$

$$= 3R_tC_L + R_tC_{x2}. \quad (19)$$

We assume that the internal node capacitance C_{x2} is 50% larger than C_{x1} because C_{x2} is the capacitance from three transistors connected while C_{x1} is the capacitance from two

transistors connected. Then

$$R_t C_{x2} = 1.5 R_t C_{x1} \quad (20)$$

$$T_{d2} = 3 R_t C_L + 1.5 R_t C_{x1} = \frac{3}{4} T_{d1}. \quad (21)$$

Therefore, T_{d2} is 25% faster than T_{d1} if we use the same V_{dd} and V_{th} for the forced stack inverter and the sleepy stack inverter. Alternatively, we may increase V_{th} of the sleepy stack inverter and make the delay of the sleepy stack inverter and the delay of the forced stack inverter the same.

Let us take an example. Using Equation 14 (see Section 3.3), the delay of the forced stack (T_{d1}) and the delay of the sleepy stack (T_{d2}) can be expressed as follows:

$$T_{d1} = K_1 \frac{V_{dd}}{(V_{dd} - V_{th1})^\alpha} \quad (22)$$

$$T_{d2} = K_2 \frac{V_{dd}}{(V_{dd} - V_{th2})^\alpha} \quad (23)$$

where K_1 and K_2 are delay coefficients of the forced stack inverter and the sleepy stack inverter, respectively. When the threshold voltage of the forced stack V_{th1} is the same as the threshold voltage of the sleepy stack V_{th2}, we calculate $K_2 = 0.75 K_1$ from Equation 21. If we assume that $\alpha = 1.3$, $V_{dd} = 1V$, and $V_{th} = 0.25V$, we can make T_{d1} equal to T_{d2} by applying $V_{th2} = 0.423$, which is 69% higher than the V_{th1} of the forced stack inverter. This higher V_{th} can potentially result in large leakage power reduction (e.g., 10X).

5.3 Summary

In this chapter, we introduced the sleepy stack technique for leakage power reduction. By combining the forced stack technique and the sleep transistor technique, the sleepy stack can achieve smaller transistor delay than the forced stack technique while retaining state unlike the sleep transistor technique. The main advantage of the sleepy stack approach is the ability to use high-V_{th} for both the sleep transistors and the transistors in parallel with the sleep transistors. The increased threshold voltage transistors of the sleepy stack

technique potentially brings much larger (>10X) leakage power reduction than the forced stack technique while achieving the same transistor delay. From the analytical model of the sleepy stack inverter, we observe that the sleepy stack inverter can reduce delay by 25%, which alternatively can be used to increase V_{th} by 69%. Using this increased threshold voltage, the sleepy stack inverter can potentially achieve a large (e.g., 10X) leakage power reduction compared to the forced stack inverter.

In this chapter, we explained the sleepy stack structure and sleepy stack operation. We also described a first order delay model of the sleepy stack (please note that all power and delay results reported are based, however, on HSPICE). In the next chapter we apply the sleepy stack structure to generic logic circuits and to SRAM, explaining in detail our methodology.

CHAPTER VI

APPLYING SLEEPY STACK

In chapter 5, we explained our new low-leakage “sleepy stack” structure. Furthermore, we explore various circuit applications of the sleepy stack technique. We largely categorize the applications into two kinds: generic logic circuits and memory circuits, i.e., SRAM. The generic logic circuits – including inverter, NAND, NOR and full adder gates – are implemented using state-saving as well as state-destructive low-leakage techniques for empirical evaluation. We will explain our detailed experimental methodology. We then explore various implementations of the sleepy stack SRAM cell and explain the experimental methodology we apply to the SRAM cell.

6.1 Applying sleepy stack to logic circuits

In this section, we first explain target benchmark circuits focusing on generic logic to evaluate our sleepy stack technique. Then we explain low-leakage techniques we consider for purposes of comparison. Although the basic ideas of the compared techniques have been covered in Section 4.1, this section will give detailed structure with transistor sizing for each prior technique to be compared to our sleepy stack approach. Finally, we explain experimental methodology that we used to compare our technique to the previous techniques we consider.

6.1.1 Benchmark circuits

We apply the sleepy stack technique to various generic logic circuits to show that the sleepy stack technique is applicable to general logic design. We choose three benchmark circuits: (i) a chain of 4 inverters, (ii) a 4:1 multiplexer and (iii) a 4-bit adder.

6.1.1.1 A chain of 4 inverters

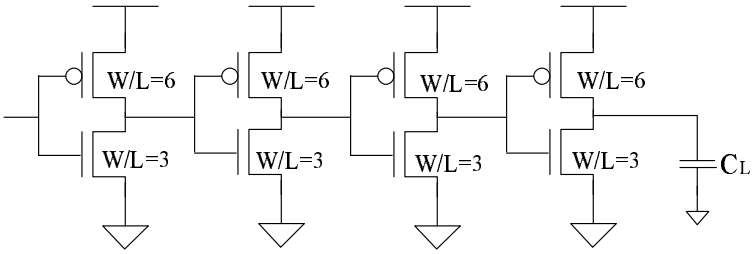


Figure 17: Chain of 4 inverters

A chain of 4 inverters shown in Figure 17 is chosen because an inverter is one of the most basic CMOS circuits and is typically used to study circuit characteristics. We size each transistor of the inverter to have equal rise and fall times in each stage. Instead of using the minimum possible size of the transistor in a given technology, we use $W/L = 6$ for PMOS and $W/L = 3$ for the NMOS. Please refer to Figure 55 in Appendix A for a layout of the chain of 4 inverters in TSMC 0.18μ technology using the widths shown in Figure 17; note that in Figure 55 all pMOS transistors have $W = 1.08\mu$ and $L = 0.18\mu$ while all nMOS transistors have $W = 0.54\mu$ and $L = 0.18\mu$.

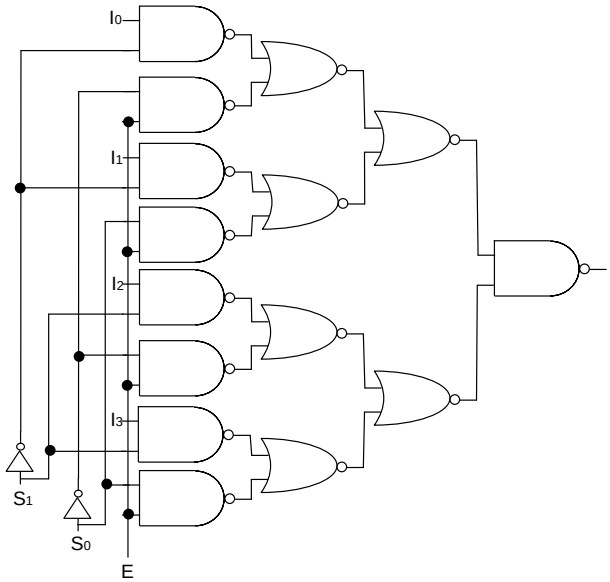


Figure 18: 4:1 multiplexer

6.1.1.2 4:1 multiplexer

A possible implementation of a 4:1 multiplexer is shown in Figure 18, in which I_0-I_3 are input signals, S_1 and S_2 are selection signals, and E is an enable signal. The multiplexer consists of inverter, 2-input NAND and 2-input NOR gates. All gates are size to have rise and fall times equal to an inverter with PMOS $W/L = 6$ and NMOS $W/L = 3$. Although the 4:1 multiplexer shown in Figure 18 is not the most efficient way to implement a 4:1 multiplexer, we use the design of Figure 18 to show that the sleepy stack can be applicable to a combination of (a logic network of) typical CMOS gates. Please refer to Figures 65 and 66 in Appendix B for NAND and NOR layouts used in this 4:1 multiplexer.

6.1.1.3 4-bit adder

By use of the one bit full adder shown in Figure 19, we implement a 4-bit adder. A full adder is an example of a typical complex CMOS gate. In Figure 19, a and b are two inputs

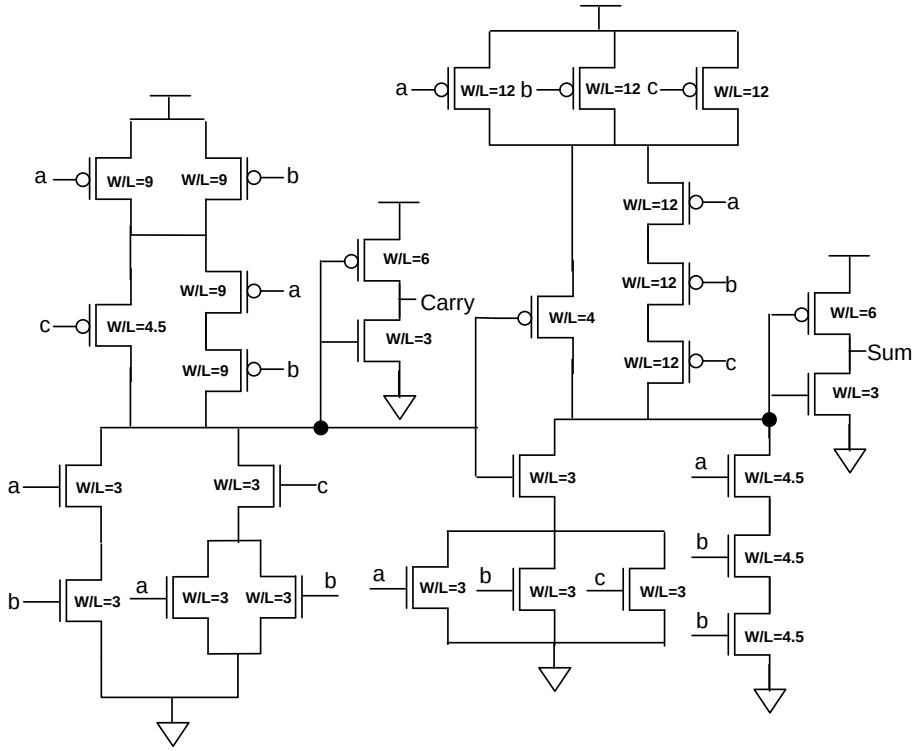


Figure 19: 1-bit full adder

and c is a carry input. $Carry$ and Sum are outputs. The transistor sizing of the full adder is noted in Figure 19. Please refer to Figure 60 in Appendix C for the full adder layout we use.

These three benchmark circuits (chain of 4 inverters, 4:1 multiplexer and 4-bit adder) designed in a conventional CMOS structure are used as our base case. In the next section we explain the low-leakage techniques we compare to our sleepy stack technique. These three benchmark circuits are also implemented using the low-leakage techniques explained in the next section, Section 6.1.2.

6.1.2 Prior low-leakage techniques considered for comparison purposes

The sleepy stack technique is compared to a conventional CMOS approach, which is our base case, and three other well-known previous approaches, i.e., the forced stack, sleep and zigzag techniques explained in Section 4.1.1. Furthermore, we additionally consider a high-V_{th} technique where all transistors are made to be high-V_{th} (this is another typical state-of-the-art approach to reduce leakage power consumption).

6.1.2.1 Base case and high-V_{th}

In this thesis we use the phrase "base case" to refer to the conventional CMOS technique shown in Figure 20 and described in a classic textbook by Weste and Eshraghian [74]. Figure 20 shows a pull-up network and a pull-down network using as few transistors as

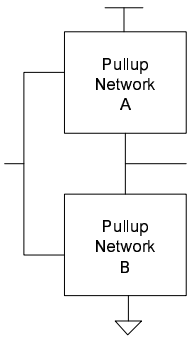


Figure 20: Base case

possible to implement the Boolean logic function desired. The base case of a chain of 4 inverters is sized as explained in Section 6.1.1.1. The base case of a 4:1 multiplexer is sized as explained in Section 6.1.1.2. The base case of a 4-bit adder is sized as explained in Section 6.1.1.3.

For the high-V_{th} technique, we use the same circuit structure and transistor widths as the base case for the benchmark circuit, but we use high-V_{th} transistors instead of the low-V_{th} transistors used in the base case.

6.1.2.2 Sleepy stack technique

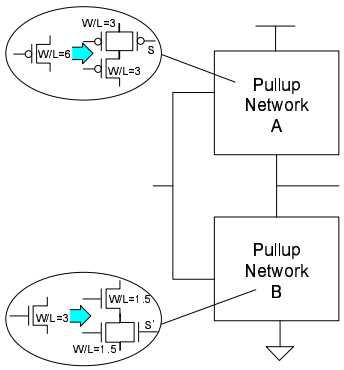


Figure 21: Sleepy stack

Figure 21 shows the sleepy stack technique applied a conventional CMOS design. When we apply the sleepy stack technique, we replace each existing transistor with two half sized transistors and add one extra sleep transistor as shown in Figure 21. If dual-V_{th} values are available, high-V_{th} transistors are used for sleep transistors and transistors that are parallel to the sleep transistors.

6.1.2.3 Forced stack technique

Figure 22 shows the forced stack technique, which forces a stack structure by breaking down an existing transistor into two half size transistors. When we apply the forced stack technique, we replace each existing transistor with two half sized transistors as shown in Figure 22.

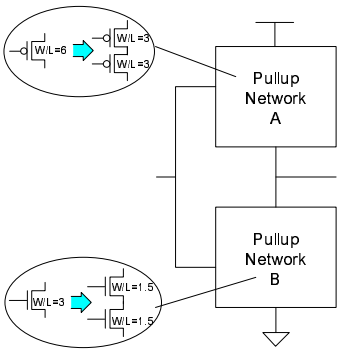


Figure 22: Forced stack

6.1.2.4 Sleep transistor technique

The sleep transistor technique shown in Figure 23 uses sleep transistors between both V_{dd} and the pull-up network as well as between Gnd and the pull-down network. Generally, the width/length (W/L) ratio is sized based on a trade-off between area, leakage reduction and delay. For simplicity, we size the sleep transistor to the size of the largest transistor in the network (pull-up or pull-down) connected to the sleep transistor. The size noted in Figure 23 shows an example when the sleep transistors are applied to one of the inverters from Figure 17. The pMOS and nMOS sleep transistors have $W/L = 6$ and $W/L = 3$, respectively, because the size of the pull-up and pull-down transistors in Figure 17 are $W/L = 6$ and $W/L = 3$, respectively. If dual-V_{th} values are available, high-V_{th} transistors are used for sleep transistors.

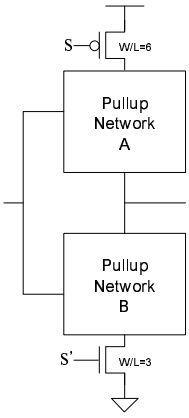


Figure 23: Sleep

6.1.2.5 Zigzag technique

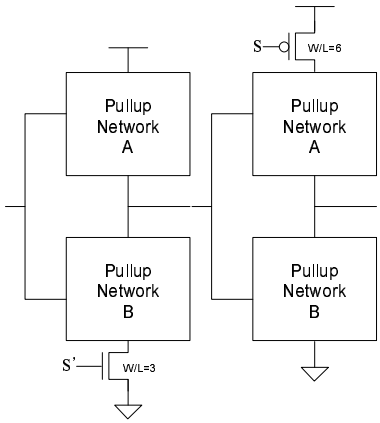


Figure 24: Zigzag

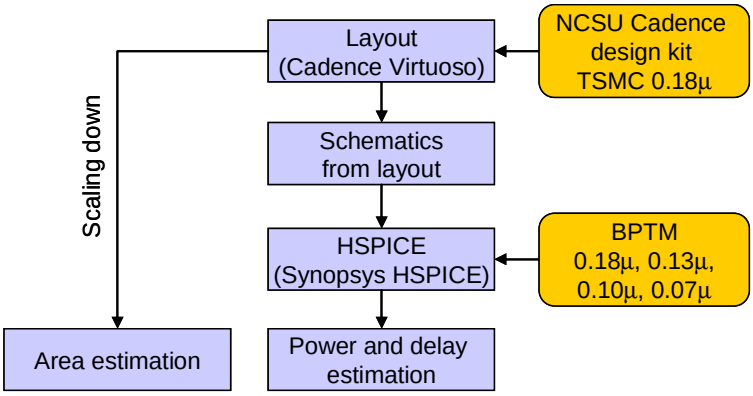
The zigzag technique in Figure 24 uses one sleep transistor in each logic stage either in the pull-up or pull-down network according a particular input pattern. In this thesis, we use an input vector that can achieve the lowest possible leakage power consumption. Then, we either assign a sleep transistor to the pull-down network if the output is '1' or else assign a sleep transistor to the pull-up network if the output is '0.' For Figure 24, we assume that the output of the first stage is '1' and the output of the second stage is '0' when minimum leakage inputs are asserted. Therefore, we apply a pull-down sleep transistor for the first stage and a pull-up sleep transistor for the second stage. Similar to the sleep transistor technique, we size the sleep transistors to the size of the largest transistor in the network (pull-up or pull-down) connected to the sleep transistor. The transistor sizing in Figure 24 shows an example where the zigzag technique is applied to two inverters from Figure 17. If dual-V_{th} values are available, high-V_{th} transistors are use for the sleep transistors.

The low-leakage techniques explained in this section, Section 6.1.2, are implemented using the three benchmark circuits described in Section 6.1.1. In the next section, we explain our experimental methodology.

6.1.3 Experimental methodology

The implemented circuits are simulated to measure delay, power, and area. For power measurement, we consider both dynamic power and static power. We first explain experimental infrastructure, and then we explain detailed measurement methodology.

6.1.3.1 Simulation setup



Tech.	0.07µ	0.10µ	0.13µ	0.18µ
V_{DD}	0.8V	1.0V	1.3V	1.8V

Figure 25: Experimental methodology

We use an empirical methodology to evaluate the five techniques which are the base case, zigzag, sleep, stack and sleepy stack techniques. Each benchmark circuit implemented using each of the five techniques is evaluated in terms of delay, dynamic power, static power and area. Our experimental procedure, which is shown in Figure 25, is as follows. We first design each target benchmark circuit with each specific technique using Cadence Virtuoso, a custom layout tool [11], and the North Carolina State University (NCSU) Cadence design kit targeting TSMC 0.18μ technology [48]. When we design a circuit using Cadence Virtuoso, we implement schematics as well as layouts. Then we extract schematics from layout to obtain transistor circuit netlists. The extracted netlists are fed into the HSPICE simulation to estimate delay and power of the target benchmark

designed with a specific technique; we use Synopsys HSPICE [65].

We use TSMC 0.18μ parameters obtained from MOSIS [66], and we also use the Berkeley Predictive Technology Model (BPTM) parameters for the technologies below 0.18μ in order to estimate the changes in power and delay as technology shrinks [7, 12]. The chosen technologies, i.e., 0.07μ, 0.10μ, 0.13μ and 0.18μ, use supply voltages of 0.8V, 1.0V, 1.3V and 1.8V, respectively. We assume that only a single supply voltage is used in the chip designs we target. We do consider both single- and dual-V_{th} technology for the sleep, zigzag and sleepy stack techniques. For the forced stack technique, we apply high-V_{th} to one of the stacked transistor while fixing the technology to 0.07μ to observe delay and leakage variations (we find that high-V_{th} causes dramatic – greater than 5X – delay increase with the forced stack technique – see Section 7.1.2). For the logic circuits, we set all high-V_{th} transistors to have 2.0 times higher V_{th} than the V_{th} of a normal transistor (low-V_{th}).

6.1.3.2 Delay

We measure the worst case propagation delay of each benchmark. Input vectors and input and output triggers are chosen to measure delay across a given circuit's critical path. The propagation delay is measured between the trigger input edge reaching 50% of the supply voltage value and the circuit output edge reaching 50% of the supply voltage value. Input waveforms have a 4ns period (i.e., a 250 MHz rate) and rise and fall times of 100ps.

For the chain of 4 inverters, we measure two different propagation delay values: one when an input goes high and another when an input goes low. We take the larger value as the worst case propagation delay of the chain of 4 inverters.

For the 4:1 multiplexer, we measure the worst case propagation delay of the path S_1-Inv-NAND-NOR-NOR-NAND-output shown in Figure 26 (note that several other paths exist with equal delay). We measure this critical path delay when the output changes from '0' to '1.' To generate this signal transition, we pick initial input values as $I_0 = 1$, $S_1 = 1$, $E = 1$, $I_1 = 0$, $I_2 = 0$, $I_3 = 0$, and $S_0 = 0$ as shown in Figure 27; the result is that the

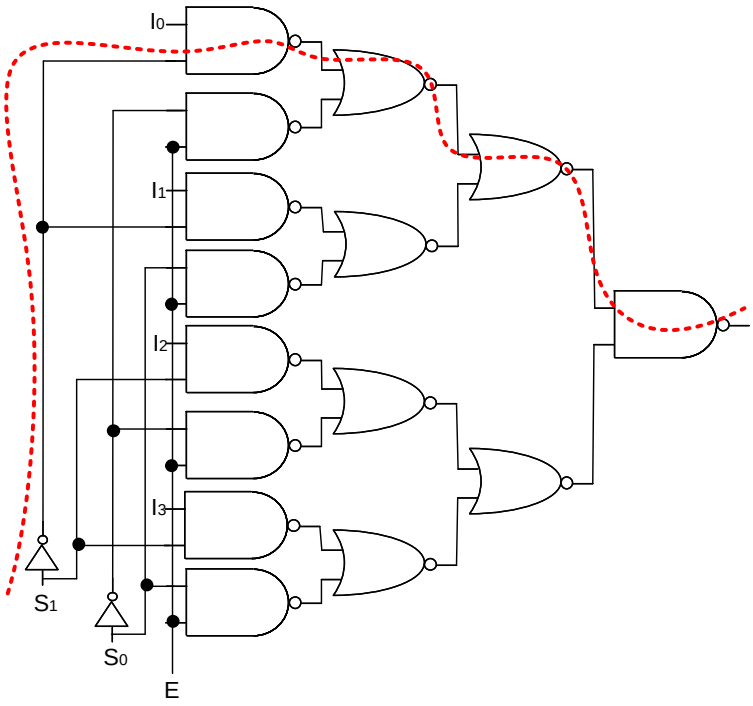


Figure 26: 4:1 multiplexer critical path

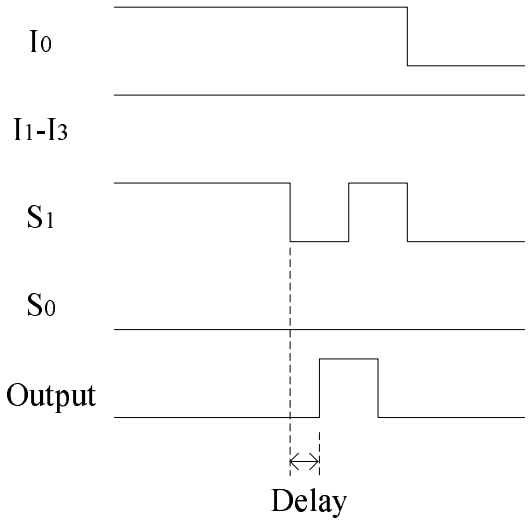


Figure 27: Input/output waveform for 4:1 multiplexer

output is equal to '0.' Then we set $S_1 = 0$ to make the output equal to '1.' We measure the propagation delay between the falling edge of S_1 and the rising edge of the output.

We form a 4-bit adder as shown in Figure 28 using four 1-bit full adders all of which are identical in size. The critical path of our 4-bit adder is the path $B_0 - C_{out0} - C_{in1} -$

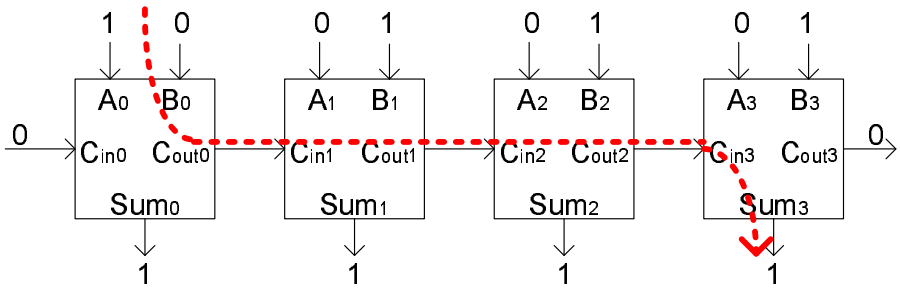


Figure 28: Inputs of 4-bit adder for critical path delay measurement

$C_{out1} - C_{in2} - C_{out2} - C_{in3} - C_{out0} - C_{in1} - Sum_3$. To measure the worst case propagation delay, we initially force input signals as shown in Figure 28. Then we assert $B_0 = 1$ and measure the delay from B_0 to Sum_3.

6.1.3.3 Active power

Active power is measured by asserting semi-random input vectors and calculating the average power dissipation during this time. Input vectors are chosen so that a large number of possible input combinations are included in the set. We take the average power dissipation reported by HSPICE as our estimate of active power consumption. This active power includes dynamic power as well as static power during the time we measure. However, we do not subtract out static power consumption to calculate pure dynamic power consumption; instead, we use this power consumption as active power consumption. All sleep transistors are turned on when we measure active power for the sleep, zigzag and sleepy stack techniques.

We measure the active power of the chain of 4 inverters by asserting '1' and '0' repeatedly. For the 4:1 multiplexer, the input vectors are chosen to represent a sample of possible inputs, with a change of at least four of the seven input bits at every input change. For the 4-bit adder, we assert input vectors covering every possible input. The waveform in Figure 29 shows input vectors asserted for each one bit adder, where the input vector changes in every 4ns. Please note that we use the same signal timing while we scaling technology from 0.18μ to 0.07μ. We do not customize signal timing to each particular technology

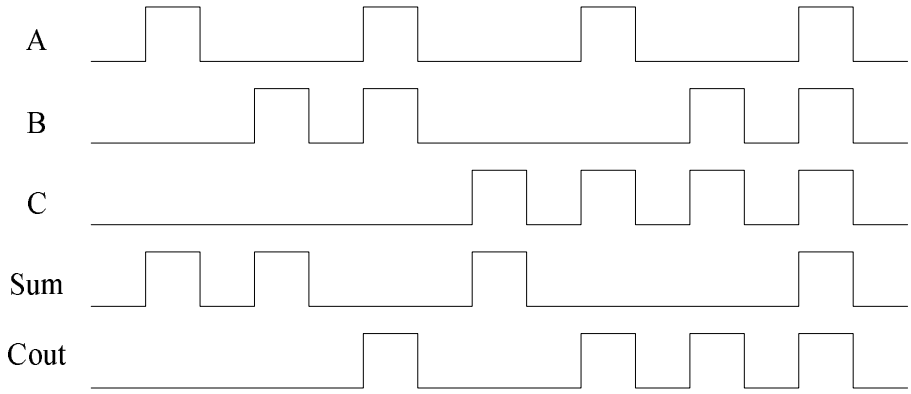


Figure 29: Waveforms of 1-bit adder for dynamic power measurement

(e.g., 0.13μ) because in this way we can observe the effect of technology scaling on a fixed clock. However, we are aware that reducing cycle time along with technology feature size is possible and may reveal additional insights and tradeoffs.

6.1.3.4 Static power

We also use HSPICE to measure static power consumption. Since static power varies according to input state, we consider either a full combination of input vectors or subset of possible input combinations. When we measure static power, we first assert an input vector and measure power consumption after signals become stable (e.g., after 30ns). Each measured static power consumption over 30ns is averaged to derive static power consumption of each benchmark.

For the chain of 4 inverters, we consider two input vectors '1' and '0.' For the 4:1 multiplexer, we choose eight input vectors out of 128 possible input combinations. The chosen input combinations are shown in Table 4. For the 4-bit adder, all eight possible input vectors of a full adder are considered for leakage power measurement.

The sleep transistors of the sleep, zigzag, and sleepy stack techniques are turned off during sleep mode in which we measure the leakage power consumption. For the zigzag technique, we take the lowest static power dissipation instead of averaging each measured power result for each input tested; in short, we assume that the zigzag technique applied

Table 4: Input sets for a 4:1 multiplexer static power measurement

I_0	I_1	I_2	I_3	S_0	S_1	E
0	0	0	0	0	0	0
1	0	0	0	0	0	0
1	0	0	0	0	0	1
1	1	0	0	0	0	1
1	1	0	1	0	1	1
1	1	0	1	1	1	1
1	1	0	1	1	1	0
1	1	1	1	1	1	1

an input vector that achieves the lowest possible leakage power by analyzing circuitry as explained in Section 4.1.1.2 and Section 6.1.2.5.

6.1.3.5 Area

The area of the 0.18μ technology version of each target circuit in a particular design style (e.g., zigzag) is measured using layout. For a chain of 4 inverters and a 4-bit adder, we directly measure from an actual full layout we did for each (see Appendices A and B). For a 4:1 multiplexer, we directly measure the area of the gates used (i.e., NAND, NOR and INV – see Appendix C) and estimate total area. Although the gates used to build the 4:1 multiplexer, i.e., NAND, NOR, and INV, have different heights, we assume that all gates have identical height to use the same V_{dd} and Gnd rails. Therefore, we estimate area of the 4:1 multiplexer by multiplying the height of the tallest gate and the sum of all gate widths. For example, if we use an INV (width=0.5μ, height=1μ), a NAND (width=0.5μ, height=1.2μ) and a NOR (width=0.5μ, height=1.4μ), then the area is 2.9μ^2.

Area when utilizing technologies below 0.18μ is estimated by scaling the area of each benchmark layout for each particular design style where TSMC 0.18μ technology is taken as a starting point. We add 10% area overhead in order to consider non-linear scaling layers, e.g., a particular metal layer. For example, if an area of 100μm^2 is measured for a particular layout in the TSMC 0.18μ process, we estimate the area for a 0.13μ process to be $100\mu m^2 \times (0.13^2/0.18^2) \times 1.1$. To estimate area of layouts using 0.13μ, 0.10μ and 0.07μ

technologies, we do not take into account extra area needed to wire gates (even though needed, e.g., to connect the gates comprising the 4:1 multiplexer or the 1-bit adders into 4 bits)), but the absence of a wiring penalty equally affects all techniques considered (i.e, base case, sleep, zigzag, forced stack and sleepy stack).

6.1.3.6 Experiments

We perform two different experiments. We first compare the sleepy stack to the base case and three well-known techniques, i.e., sleep, zigzag, and forced stack, while scaling transistor technologies. For this experiment, we use all three benchmark circuits explained in Section 6.1.1.

Second, we compare the sleepy stack technique only to the state-saving techniques, i.e., the forced stack technique and the high-V_{th} technique. At this time we consider various V_{th} values, various transistor widths and two different temperatures. We exclusively use a chain of 4 inverters for this experiment. For the base case, we vary V_{th} of all transistors. For the forced stack technique, we vary V_{th} of transistors connected to either V_{dd} or Gnd. For the sleepy stack technique, we vary V_{th} of sleep transistors and transistors in parallel with the sleep transistors. We use the "Delvto" option of HSPICE to change V_{th}. We also consider two different temperatures because leakage power is highly dependent on temperature. The two temperatures are 25^oC and 110^oC. The experimental results of the generic logic circuits will be presented in Section 7.1.

In this section, Section 6.1, we explained the application of sleepy stack to generic logic circuits. We further explained our experimental methodology. In the next section, we explain sleepy stack SRAM and associated experimental methodology.

6.2 Applying sleepy stack to SRAM

We apply sleepy stack principles to SRAM cell design. We consider four different versions of a sleepy stack SRAM cell to observe delay, area, and power tradeoffs. For purposes of

comparison, we also consider two other state-saving low-leakage SRAM techniques, which are the high-V_{th} SRAM cell and the forced stack SRAM cell.

6.2.1 Sleepy stack SRAM structure

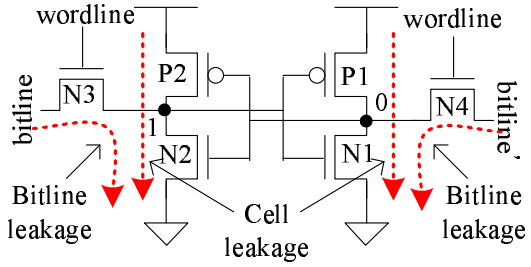


Figure 30: SRAM cell and leakage paths

We design an SRAM cell based on the sleepy stack technique. The conventional 6-T SRAM cell consists of two coupled inverters and two `wordline` pass transistors as shown in Figure 3, repeated here as Figure 30 for convenience. Since the sleepy stack technique can be applied to each transistor separately, the six transistors can be changed individually. However, to balance current flow (failure to do so potentially increases the risk of soft errors [21]), a symmetric design approach is used.

Table 5: Sleepy stack technique on a SRAM cell

Combinations	cell leakage reduction	bitline leakage reduction
Pull-down (PD) sleepy stack	medium	low
Pull-down (PD), wordline (WL) sleepy stack	medium	high
Pull-up (PU), pull-down (PD) sleepy stack	high	low
Pull-up (PU), pull-down (PD), wordline (WL) sleepy stack	high	hign

We already discussed the two main types of subthreshold leakage currents in a 6-T SRAM cell in Chapter 3. It is very important when applying the sleepy stack technique to consider the various leakage paths in the SRAM cell. To address the effect of the sleepy stack technique properly, we consider four combinations of the sleepy stack SRAM cell as shown in Table 5. In Table 5, “Pull-Down (PD) sleepy stack” means that the sleepy stack

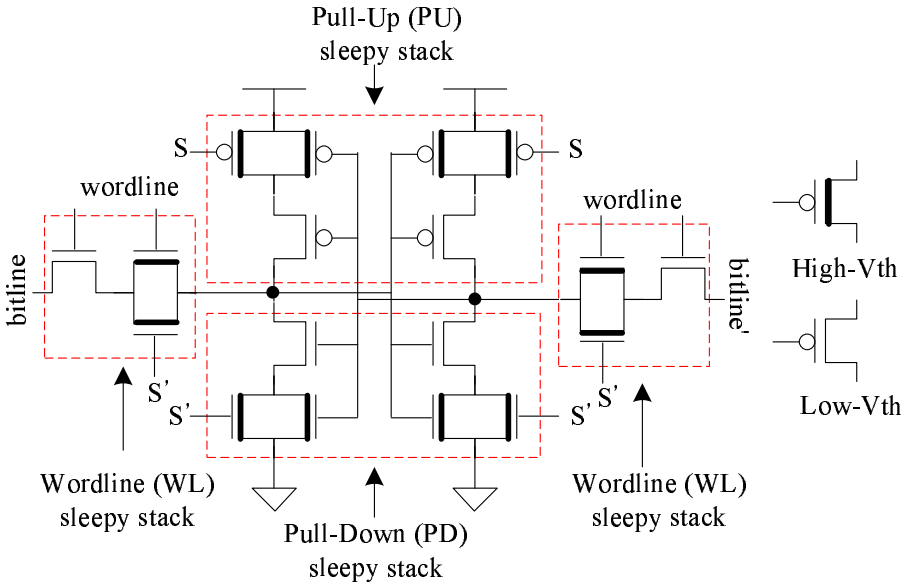


Figure 31: Sleepy stack SRAM cell

technique is only applied to the pull-down transistors of an SRAM cell as indicated in the bottom dashed box in Figure 31. “Pull-Down (PD), `wordline` (WL) sleepy stack” means that the sleepy stack technique is applied to the pull-down transistors as well as `wordline` transistors. Similarly, “Pull-Up (PU), Pull-Down (PD) sleepy stack” means that the sleepy stack technique is applied to the pull-up transistors and the pull-down transistors (but **not** to the `wordline` transistors) of an SRAM cell. Finally, “Pull-Up (PU), Pull-Down (PD), `wordline` (WL) sleepy stack” means that the sleepy stack technique is applied to all the transistors in an SRAM cell.

The PD sleepy stack can suppress some part of the cell leakage. Meanwhile, the PU, PD sleepy stack can suppress the majority of the cell leakage. However, without applying the sleepy stack technique to the `wordline` (WL) transistors, `bitline` leakage cannot be significantly suppressed. Although lying in the `bitline` leakage path, the pull-down sleepy stack is not effective to suppress both `bitline` leakage paths because one of the pull-down sleepy stacks is always on. Therefore, to suppress subthreshold leakage current in a SRAM cell fully, the PU, PD and WL sleepy stack approach needs to be considered as shown in Figure 31.

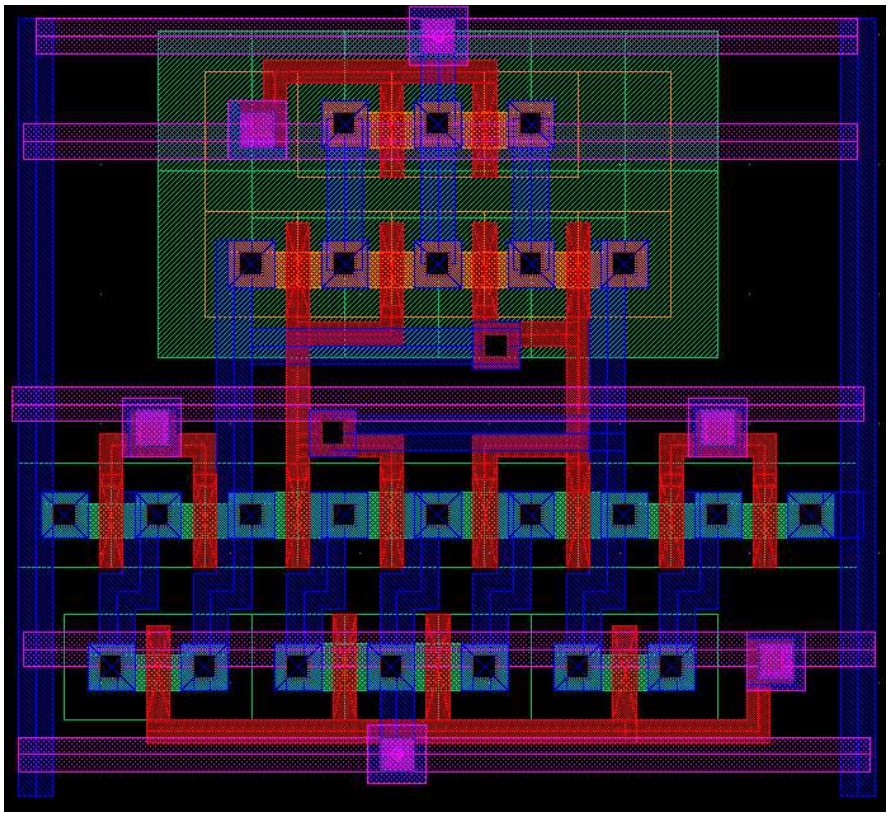


Figure 32: Sleepy stack SRAM cell layout

The sleepy stack SRAM cell design results in area increase because of the increase in the number of transistors. However, we halve the transistor width of a conventional SRAM cell to make a sleepy stack SRAM cell and thus the area increase of the sleepy stack SRAM cell not necessarily directly proportional to the number of transistors. Halving transistor width is possible when halved transistor width is larger than a minimum transistor width. Unlike the conventional 6-T SRAM cell, the sleepy stack SRAM cell requires the routing of one or two extra wires for the sleep control signal. We only use metal 1 and metal 2 layers for routing as shown in Figure 32 because we assume metal layers above metal 2 are reserved for global routing. Further, the sleepy stack SRAM cell is designed to abut easily.

6.2.2 Methodology

To evaluate the sleepy stack SRAM cell, we compare our technique to (i) using high-V_{th} transistors as direct replacements for low-V_{th} transistors (thus maintaining only 6 transistors in an SRAM cell) and (ii) the forced stack technique [47]; we choose these techniques because these two techniques are state saving techniques without high risk of soft error [21]. Although Asymmetric-Cell SRAM explained in Section 4.1.2.2 is also a state-saving SRAM cell design, we do not consider Asymmetric-Cell SRAM because we assume that our SRAM cells are filled equally with '1s' and '0s.' This is not the condition that ACC prefers, and thus leakage power savings of ACC are smaller than the high-V_{th} SRAM cell, which uses high-V_{th} for all six transistors.

We first layout SRAM cells of each technique, i.e., the conventional 6-T SRAM cell, the forced stack SRAM cell and the sleepy stack SRAM cell. Instead of starting from scratch, we use the CACTI model for the SRAM structure and transistor sizing [57]. We use NCSU Cadence design kit targeting TSMC 0.18μ technology [48]. By scaling down the 0.18μ layout, we obtain 0.07μ technology transistor level HSPICE schematics [53], and we design a 64x64bit SRAM cell array. Please refer to Appendix D for layouts of these various SRAM cells with different techniques in TSMC 0.18μ technology.

We estimate area directly from our custom layout (see Appendix D) using TSMC 0.18μ technology and scale to 0.13μ, 0.10μ and 0.07μ using the same approach (with 10% araa penalty) as described in Section 6.1.3.5. We are aware this is not exact, hence the word "estimate." We also assume the area of the SRAM cell with high-V_{th} technique is the same as low-V_{th}. This assumption is reasonable because high-V_{th} can be implemented by changing gate oxide thickness, and this almost does not affect area at all. We estimate dynamic power, static power and read time of the SRAM cell using HSPICE simulation with Berkeley Predictive Technology Model targeting 0.07μ technology [7]. The read time is measured from the time when an enabled `wordline` reaches 10% of the V_{dd} voltage to the time when either `bitline` or `bitline'` drops from 100% of the precharged voltage

to 90% of the precharged voltage value while the other remains high. Therefore, one of the `bitline` signal remains at V_{dd}, and the other is 0.9xV_{dd}. This 10% voltage difference between `bitline` and `bitline'` is typically enough for a sense amplifier to detect the stored cell value [6]. Dynamic power of the SRAM array is measured during the read operation with cycle time of 4ns. Static power of the SRAM cell is measured by turning off sleep transistors if applicable. To avoid leakage power measurement biased by a majority of '1' versus '0' (or vice-versa) values, half of the cells are randomly set to '0,' with the remaining half of the cells set to '1.'

We compare the sleepy stack SRAM cell to the conventional 6-T SRAM cell, high-V_{th} 6-T SRAM cell and forced stack SRAM cell. For the "high-V_{th}" technique and the forced stack technique, we consider the same technique combinations we applied to the sleepy stack SRAM cell – see Table 5 in Section 6.2.1.

Table 6: Applied SRAM techniques

	Technique	
Case1	Low-Vth Std	Conventional 6T SRAM
Case2	PD high-Vth	High-Vth applied to PD
Case3	PD, WL high-Vth	High-Vth applied to PD, WL
Case4	PU, PD high-Vth	High-Vth applied to PU, PD
Case5	PU, PD, WL high-Vth	High-Vth applied to PU, PD, WL
Case6	PD stack	Stack applied to PD
Case7	PD, WL stack	Stack applied to PD, WL
Case8	PU, PD stack	Stack applied to PU, PD
Case9	PU, PD, WL stack	Stack applied to PU, PD, WL
Case10	PD sleepy stack	Sleepy stack applied to PD
Case11	PD, WL sleepy stack	Sleepy stack applied to PD, WL
Case12	PU, PD sleepy stack	Sleepy stack applied to PU, PD
Case13	PU, PD, WL sleepy stack	Sleepy stack applied to PU, PD, WL

To properly observe the techniques, we compare 13 different cases as shown in Table 6. Case1 is the conventional 6-T SRAM cell, which is our base case. Cases 2, 3, 4 and 5 are 6-T SRAM cells using the high-V_{th} technique. PD high-V_{th} is the high-V_{th} technique applied only to the pull-down transistors. PD, WL high-V_{th} is the high-V_{th} technique applied

to the pull-down transistors as well as to the `wordline` transistors. PU, PD high-V_{th} is the high-V_{th} technique applied to the pull-up and pull-down transistors. PU, PD, WL high-V_{th} is the high-V_{th} technique applied to all the SRAM transistors. Cases 6, 7, 8 and 9 are 6-T SRAM cells with the forced stack technique [47]. PD stack is the forced stack technique applied only to the pull-down transistors. PD, WL stack is the forced stack technique applied to the pull-down transistors as well as to the `wordline` transistors. PU, PD stack is the forced stack technique applied to the pull-up and pull-down transistors. PU, PD, WL stack is the forced stack technique applied to all the SRAM transistors. Please note that we do not apply high-V_{th} to the forced stack technique because the forced stack SRAM with high-V_{th} incurs more than 2X delay increase. Cases 10, 11, 12 and 13 are the four sleepy stack SRAM cell approaches as listed in Table 5. For the sleepy stack, high-V_{th} is applied only to the sleep transistors and the transistors parallel to the sleep transistors as shown in Figure 31.

The experimental results regarding SRAM leakage results will be presented in Section 7.2.

6.3 Summary

In this chapter, we explore two applications of the sleepy stack approach: sleepy stack logic circuits and sleepy stack SRAM. For the sleepy stack logic circuit evaluation, we take a chain of 4 inverters, a 4:1 multiplexer and a 4-bit adder as benchmark circuits. Then we compare six different techniques, i.e., base case, high-V_{th}, sleep, zigzag, forced stack, and sleepy stack. We use HSPICE for performance evaluation, i.e., power consumption and delay. We estimate area using layout. Then we explain the sleepy stack SRAM cell and two other techniques, i.e., the high-V_{th} SRAM cell and the forced stack SRAM cell. Furthermore, we explain detailed experimental methodology.

In the next section, we will explain experimental results using the methodology explain in this section.

CHAPTER VII

SLEEPY STACK EXPERIMENTAL RESULTS

We compare the sleepy stack technique to a number of key, well-known low-leakage techniques. We first explore the experimental results for general logic circuits. Then we explore the experimental results for SRAM cell design.

7.1 Experimental results for general logic circuits

In this section, we explain the experimental results for generic logic circuits. We utilize the three logic designs presented in Section 6.1.

7.1.1 Impact of technology scaling

First we explore the impact of technology scaling. Figures 33, 34 and 35 show the experimental results for the chain of 4 inverters (see Section 6.1.1.1), 4:1 multiplexer (see Section 6.1.1.2), and 4-bit adder (see Section 6.1.1.3). Figures 33, 34 and 35 show results from 0.18μ to 0.07μ. We considered five different techniques: base case (standard low-V_{th} CMOS), forced stack, sleep, zigzag, and sleepy stack. Please note that in Figures 33, 34 and 35, a '*' next to a technique name means that the technique was implemented utilizing high-V_{th} transistors appropriately.

We can observe from Figures 33(a), 34(a) and 35(a) that static power increases as technology feature size shrinks. We can also observe from Figures 33(b), 34(b) and 35(b) that dynamic power decreases as technology feature size shrinks . Finally, we can observe from Figures 33(c), 34(c) and 35(c) that propagation delay decreases as technology feature size shrinks.

Let us focus on the single V_{th} 0.07μ technology implementation of each benchmark

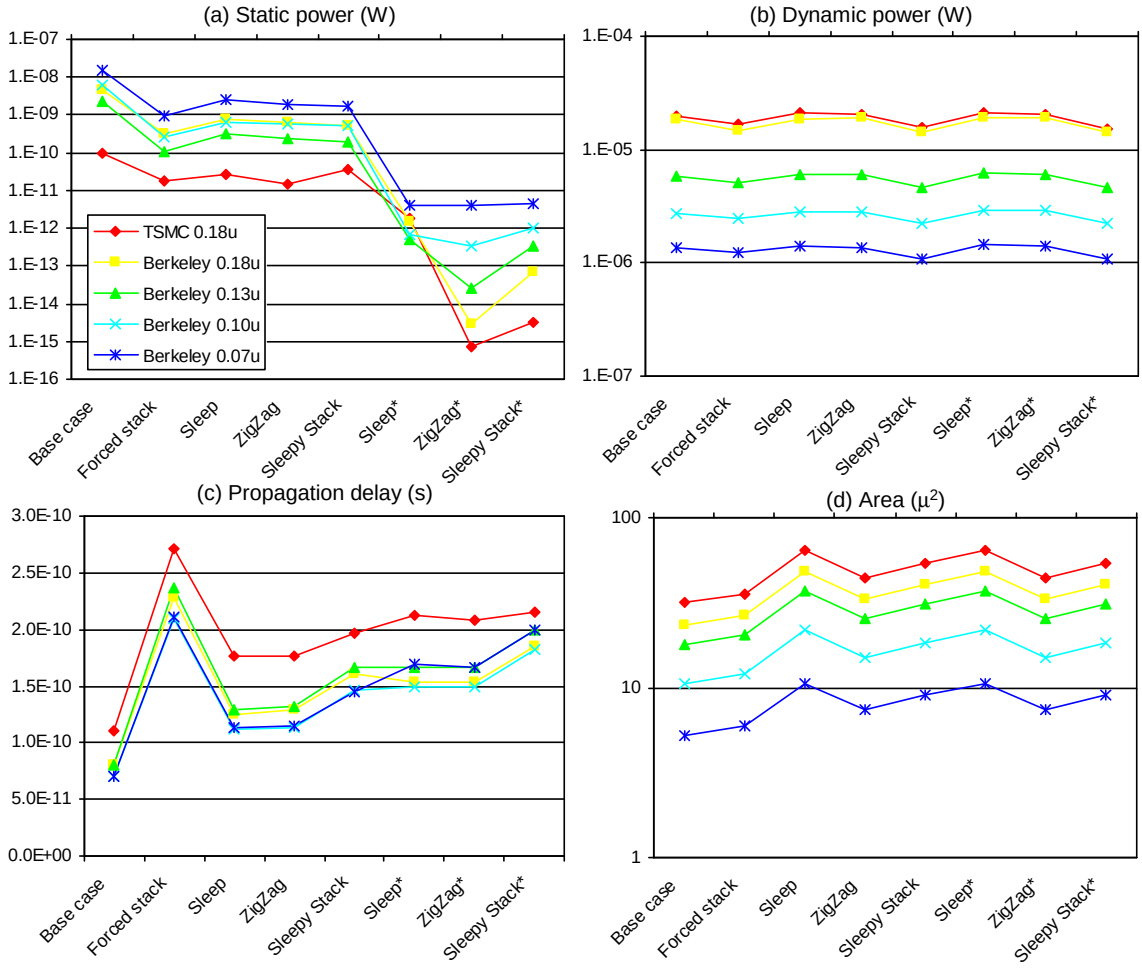


Figure 33: Results for a chain of 4 inverters (*dual V_{th})

Table 7: Results for a chain of 4 inverters (0.07μ)

A chain of 4 inverters	Propagation delay (s)	Static Power (W)	Dynamic Power (W)	Area (μ2)
Base case	7.05E-11	1.57E-08	1.34E-06	5.23
Forced stack	2.11E-10	9.81E-10	1.25E-06	5.97
Sleep	1.13E-10	2.45E-09	1.39E-06	10.67
ZigZag	1.15E-10	1.96E-09	1.34E-06	7.39
Sleepy Stack	1.45E-10	1.69E-09	1.08E-06	9.03
Sleep (dual Vth)	1.69E-10	4.12E-12	1.46E-06	10.67
ZigZag (dual Vth)	1.67E-10	4.07E-12	1.39E-06	7.39
Sleepy Stack (dual Vth)	1.99E-10	4.56E-12	1.09E-06	9.03

shown in Tables 7, 8 and 9: we see that our sleepy stack approach with single-V_{th} results in leakage power roughly equivalent to the other three leakage-reduction approaches, i.e., forced stack, sleep and zigzag when each uses single-V_{th} technology. Compared to the sleep and zigzag approaches, which do not save state, the sleepy stack approach results in

up to 68% delay increase and up to 138% area increase. Furthermore, compared to the forced stack approach, which saves state, the sleepy stack approach results in up to 118% area increase, but the sleepy stack is up to 31% faster. Thus, we recommend the sleepy stack approach with single-V_{th} when state-preservation is needed, dual-V_{th} is not available, the speedup over forced stack is important and the area penalty for sleepy stack is acceptable.

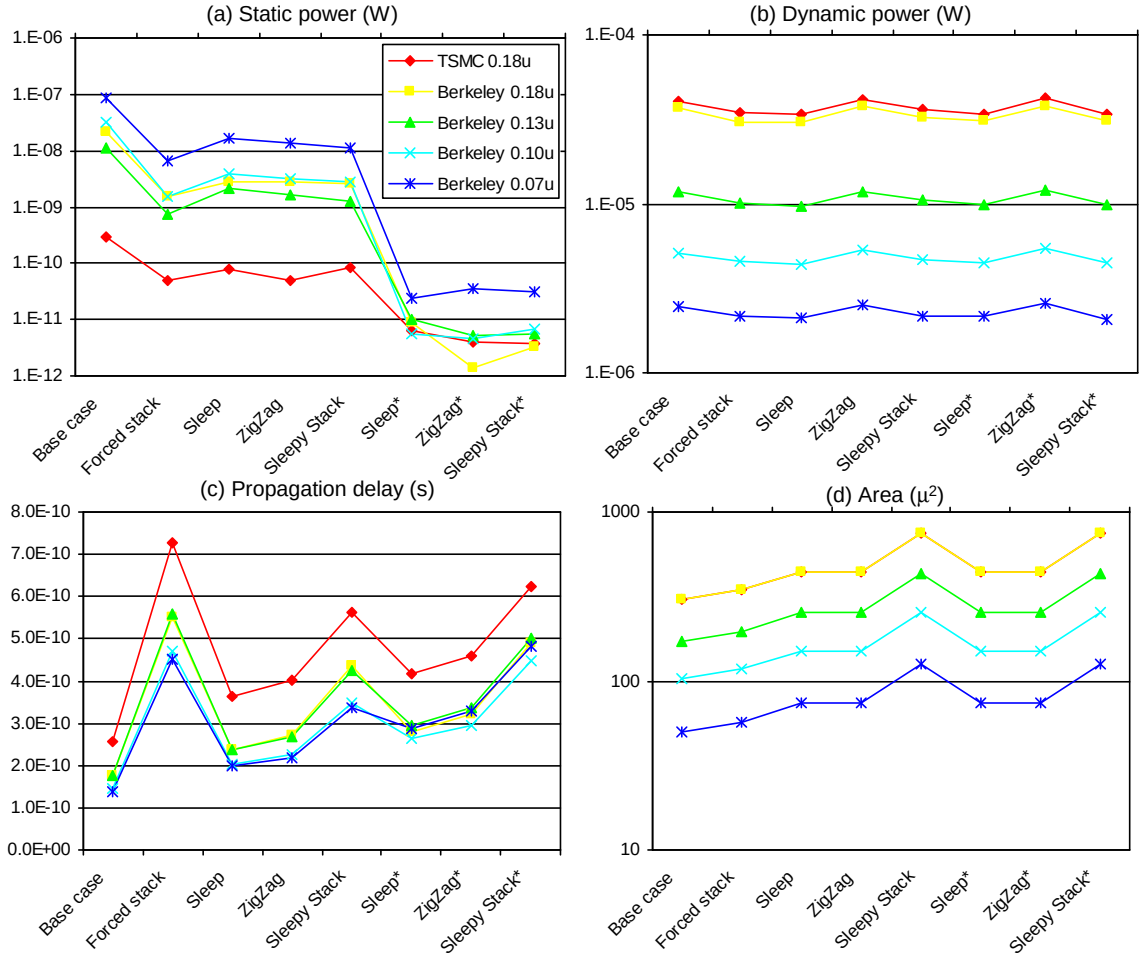


Figure 34: Results for a 4:1 multiplexers (*dual V_{th})

In addition to single-V_{th} technology, the zigzag, sleep and sleepy stack approaches are also implemented using dual-V_{th} technology in which high-V_{th} transistors are used as explained in Sections 6.1.2.2, 6.1.2.4 and 6.1.2.5. Compared to the sleep and zigzag approaches with using dual-V_{th} technology, the sleepy stack approach can save state. This is a main advantage of the sleepy stack over the sleep and zigzag techniques.

Table 8: Results for a 4:1 multiplexers (0.07μ)

4:1 multiplexer	Propagation delay (s)	Static Power (W)	Dynamic Power (W)	Area (µ2)
Base case	1.39E-10	8.57E-08	2.49E-06	50.17
Forced stack	4.52E-10	6.46E-09	2.14E-06	57.40
Sleep	1.99E-10	1.65E-08	2.10E-06	74.11
ZigZag	2.17E-10	1.36E-08	2.54E-06	74.36
Sleepy Stack	3.35E-10	1.09E-08	2.18E-06	125.33
Sleep (dual Vth)	2.87E-10	2.41E-11	2.15E-06	74.11
ZigZag (dual Vth)	3.28E-10	3.62E-11	2.59E-06	74.36
Sleepy Stack (dual Vth)	4.84E-10	3.20E-11	2.09E-06	125.33

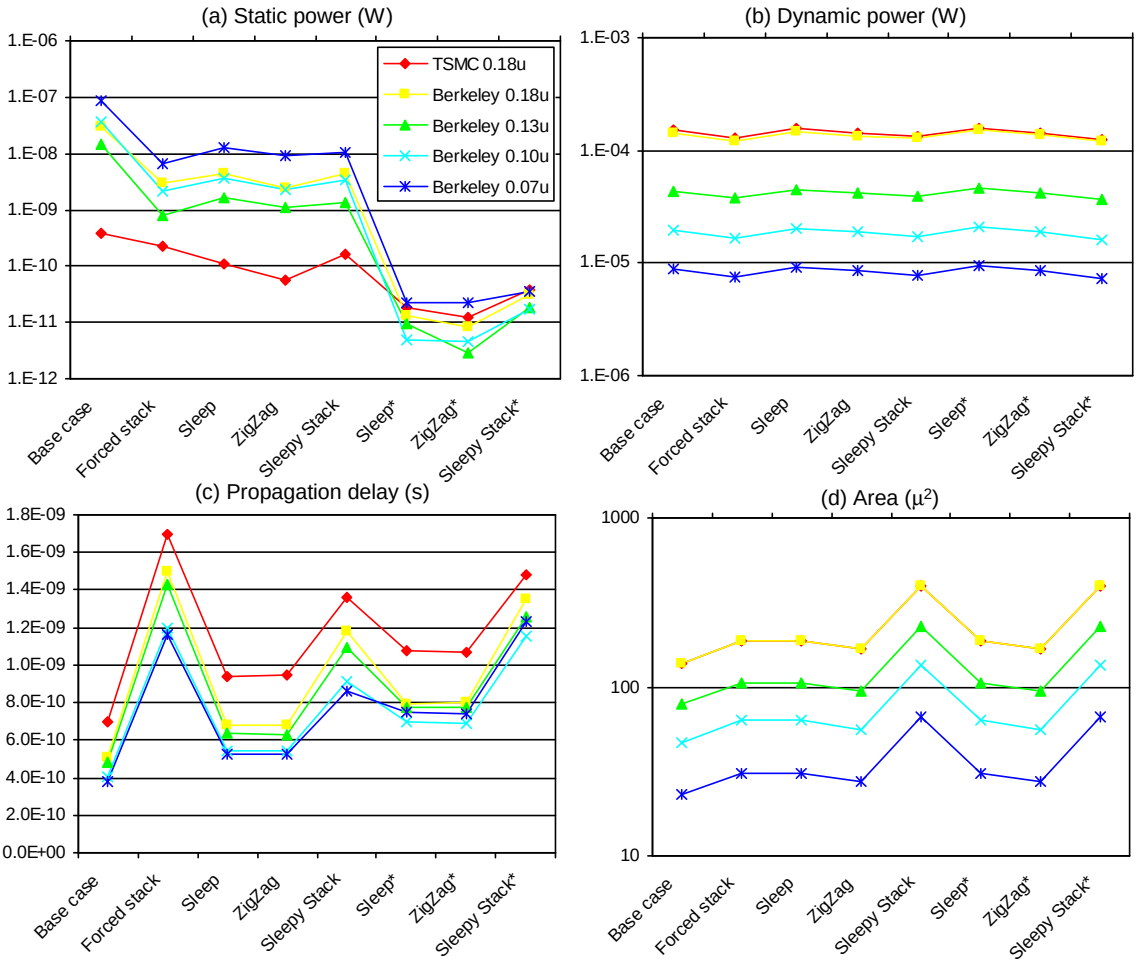


Figure 35: Results for a 4-bit adders (*dual V_{th})

Let us compare in 0.07μ technology the state-saving techniques, which are the base case with single V_{th}, forced stack with single V_{th} and sleepy stack with dual-V_{th}, highlighted as shaded rows in Tables 7, 8 and 9. The results from a chain of 4 inverters in Table 7 shows that the sleepy stack achieves 3440X leakage reduction over the base case. Furthermore,

Table 9: Results for a 4-bit adders (0.07μ)

4-bit adder	Propagation delay (s)	Static Power (W)	Dynamic Power (W)	Area (µ2)
Base case	3.76E-10	8.87E-08	8.81E-06	22.96
Forced stack	1.16E-09	6.77E-09	7.63E-06	30.94
Sleep	5.24E-10	1.24E-08	9.03E-06	30.94
ZigZag	5.24E-10	9.09E-09	8.44E-06	27.62
Sleepy Stack	8.65E-10	1.07E-08	7.70E-06	65.88
Sleep (dual Vth)	7.48E-10	2.23E-11	9.41E-06	30.94
ZigZag (dual Vth)	7.43E-10	2.19E-11	8.53E-06	27.62
Sleepy Stack (dual Vth)	1.23E-09	3.56E-11	7.26E-06	65.88

the sleepy stack achieves 215X leakage power reduction over the forced stack while reducing delay by 6% and increasing area by 51%. The results from a 4:1 multiplexer in Table 8 shows that the sleepy stack achieves 2680X leakage reduction over the base case. Compared to the forced stack, the sleepy stack achieves 202X leakage power reduction over the forced stack while increasing delay by 7% and increasing area by 118%. Finally, the results from a 4-bit adder in Table 9 shows that the sleepy stack achieves 2490X leakage reduction over the base case. Compared to the forced stack, Table 9 shows that the sleepy stack achieves 190X leakage power reduction over the forced stack while increasing delay by 6% and increasing area by 113%.

In short, our sleepy stack technique achieves up to 215X leakage power reduction with up to 7% delay overhead. Not surprisingly, the sleepy stack approach has 51$\sim$118% larger area as compared to the forced stack approach. Therefore, our sleepy stack approach with dual-V_{th} can be used where state-preservation and ultra-low leakage power consumption are needed and are judged to be worth the area overhead.

7.1.2 Impact of V_{th}

Choosing the right V_{th} value of the sleepy stack technique is very important in terms of delay and power consumption. Therefore, using a chain of 4 inverters with 0.07μ technology, we compare dynamic power, leakage power and delay of the state-saving techniques, i.e., base case (conventional CMOS technique), forced stack and sleepy stack, while varying V_{th}. We vary V_{th} of transistors as follows: all the transistors in the base case; one of

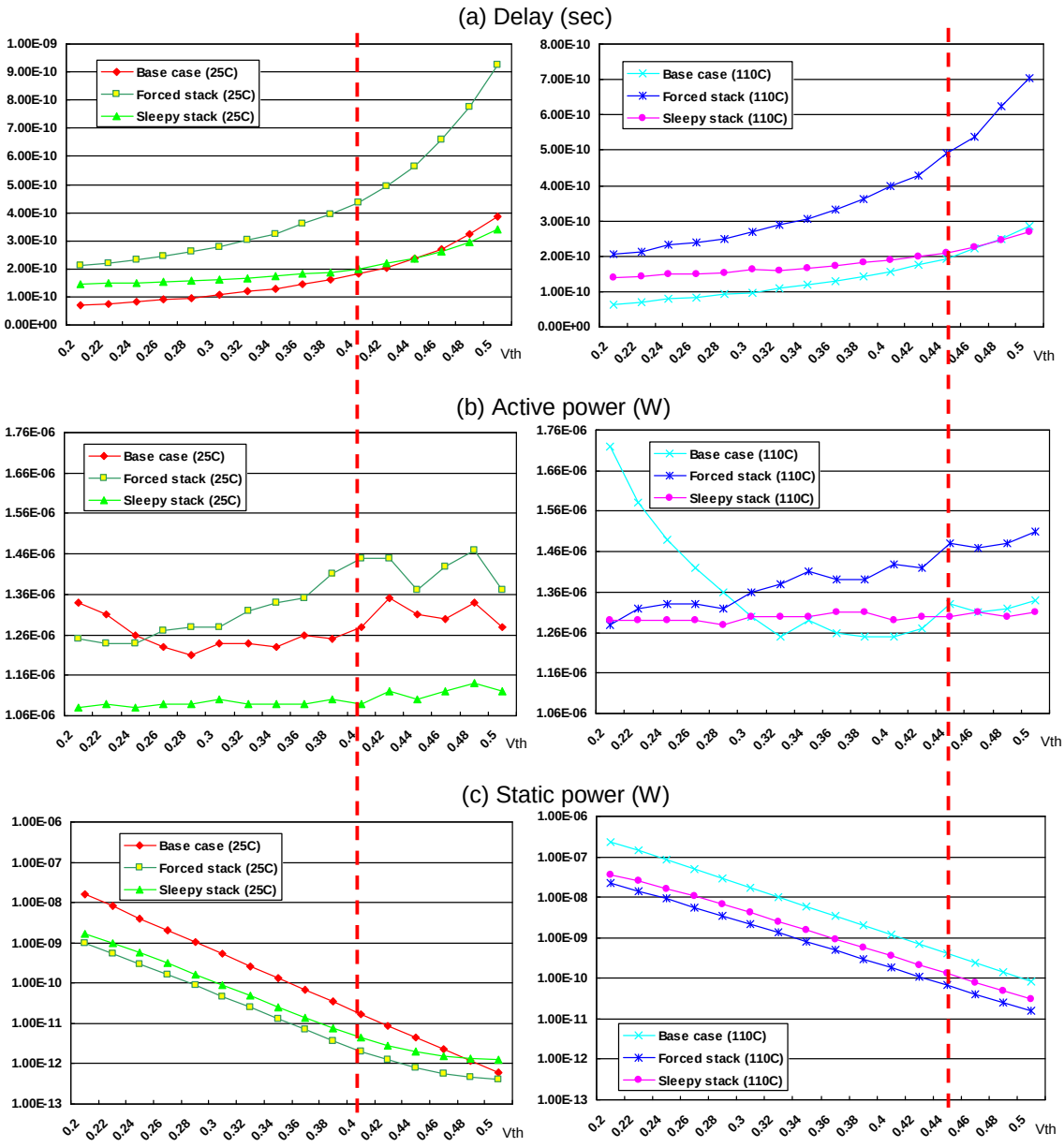


Figure 36: Results from a chain of 4 inverters while varying V_{th}

stacked transistors in the forced stack case; and the sleep transistors plus transistors parallel to the sleep transistors in the sleepy stack case. Figure 36 shows the measured results while varying V_{th}. From Figure 36(a), we can see the forced stack inverter increases delay dramatically as V_{th} increases (e.g., with 2xV_{th}, 6.2X delay the over base case). The base case also shows relatively large variation compared to the sleepy stack technique as V_{th} changes. While varying V_{th} of the sleepy stack, we can find V_{th} of the sleepy stack that achieves the

same delay with the forced stack with $V_{th} = 0.2V$, and dotted lines in Figure 36(a) indicate the V_{th} values found. At 25^oC, the sleepy stack with $V_{th} = 0.4V$ has almost exactly the same delay as the forced stack with $V_{th} = 0.2V$. Also, at 110^oC, the sleepy stack with $V_{th} = 0.42V$ has exactly the same delay as the forced stack with $V_{th} = 0.2V$.

If we compare the sleepy stack technique to the base case with $V_{th} = 0.2V$, the sleepy stack technique achieves 1000X-3400X leakage power reduction according to the temperature. Although the base case with higher V_{th} achieves better leakage reduction over the forced stack technique, the sleepy stack technique achieves 2X leakage power reduction over the base case with higher V_{th}. From Figure 36(b), we can observe that the base case with $V_{th} = 0.2V$ consumes unacceptable active power consumption when the temperature is 110^oC. This is because large leakage power consumption of the base case severely hurts active power consumption. This result emphasizes the importance of the leakage power reduction techniques in nanoscale technology.

7.1.3 Impact of transistor width

The sleepy stack technique comes with some area overhead. Therefore, we explore the impact of transistor width variation using three state-saving techniques, i.e., base case (conventional CMOS), forced stack and sleepy stack. Although increasing transistor width reduces gate internal resistance, the increased transistor width increases gate input capacitance. Therefore, we need to carefully size transistor width to reduce overall delay. We set V_{th} of the base case and the sleepy stack technique to $V_{th} = 0.4V$ while using $V_{th} = 0.2V$ for the forced stack technique since the forced stack technique with high-V_{th} increases delay dramatically as observed in Figure 37(a). We set the temperature to 25^oC. Also, we use $3C_{inv}$ as a load capacitance. The results show that inverter chain delay decreases as transistor width increases. However, delay reduction saturates due to the increased gate input capacitance. In Figure 37(a), initially the delay of the base case and the sleepy stack

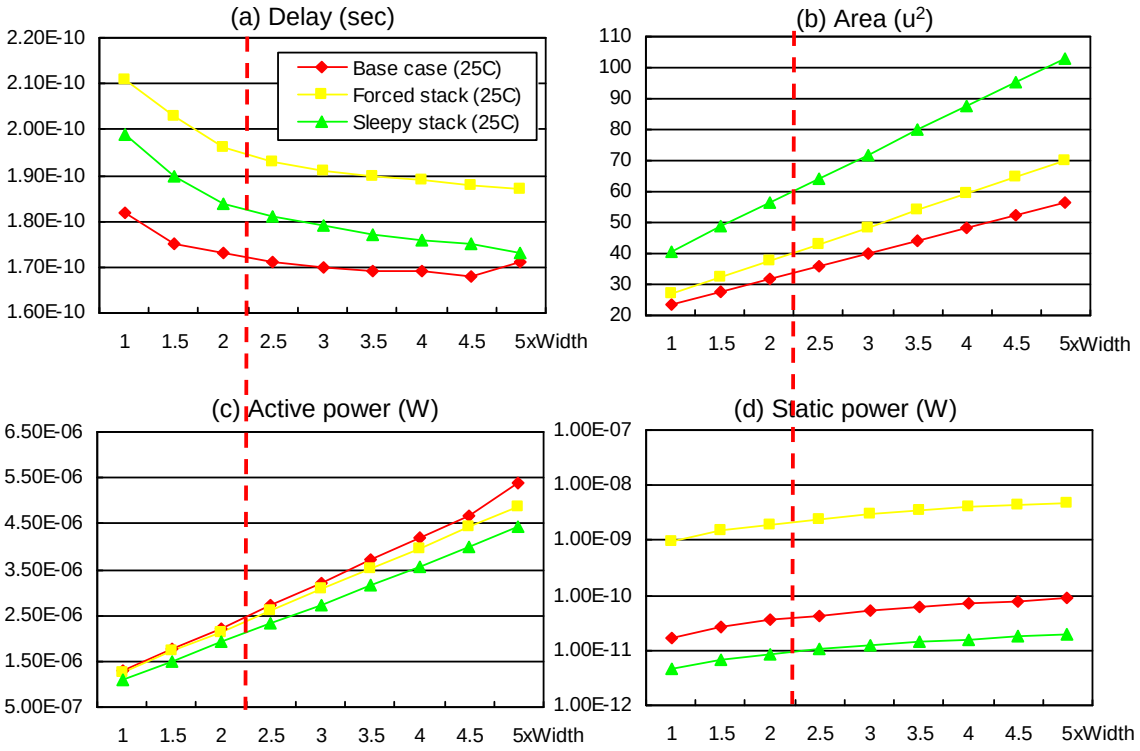


Figure 37: Results from a chain of 4 inverters while varying width

inverter are different. However, as transistor width increases, sleepy stack shows noticeable delay reduction, and the sleepy stack and the base case achieve similar delay using 5X transistor width. From Figure 37(b), the sleepy stack inverter with 1xWidth is 72% and 51% larger than the base case and the forced stack, respectively. Since the sleepy stack technique comes with some area penalties, we find an area of the forced stack technique that has the same area of sleepy stack technique by increasing transistor width. The forced stack inverter has similar areas with the sleepy stack when 2X transistor widths are applied. The forced stack with 2X width shows almost similar delay with the sleepy stack technique, but the forced stack shows 430X larger leakage power consumption than the sleepy stack technique.

We presented the advantage of the sleepy stack technique used in generic logic circuits. In next section, we explain the experimental results from SRAM.

7.2 Experimental results for SRAM

In this section, we explore the experimental results for the different sleepy stack SRAM cell variations. Unlike the generic circuit experimental comparisons in Section 7.1, here, we only consider state-saving techniques, i.e., high-V_{th} and forced stack, in addition to the sleepy stack SRAM. Also, for SRAM, we consider one more factor, the static noise margin, which represents the noise immunity of SRAM.

7.2.1 Area

Table 10: Layout area

	Technique	Height(u)	Width(u)	Area(u^2)	Normalized area
Case1	Low-Vth Std	3.825	4.500	17.213	1.00
Case2	PD high-Vth	3.825	4.500	17.213	1.00
Case3	PD, WL high-Vth	3.825	4.500	17.213	1.00
Case4	PU, PD high-Vth	3.825	4.500	17.213	1.00
Case5	PU, PD, WL high-Vth	3.825	4.500	17.213	1.00
Case6	PD stack	3.465	4.680	16.216	0.94
Case7	PD, WL stack	3.465	5.760	19.958	1.16
Case8	PU, PD stack	3.285	4.680	15.374	0.89
Case9	PU, PD, WL stack	3.465	5.760	19.958	1.16
Case10	PD sleepy stack	4.545	5.040	22.907	1.33
Case11	PD, WL sleepy stack	4.455	6.705	29.871	1.74
Case12	PU, PD sleepy stack	5.760	5.040	29.030	1.69
Case13	PU, PD, WL sleepy stack	5.535	6.615	36.614	2.13

Table 10 shows the area of each technique. Please note that SRAM cell area can be reduced further by using minimum size transistors, but reducing transistor size increases cell read time. Some SRAM cells with the forced stack technique show smaller area even compared to the base case. The reason is that divided transistors can enable a particularly squeezed design. The sleepy stack technique increases area by between 33% and 113%. The added sleep transistors are a bottleneck to reduce the size of the sleepy stack SRAM cells. Further, wiring the sleep control signals (an overhead we do not consider in Table 10) makes the design more complicated.

7.2.2 Cell read time

Table 11: Cell read time

	Technique	Delay (sec)						Normalized delay					
		25°C			110°C			25°C			110°C		
		1xVth	1.5xVth	2xVth	1xVth	1.5xVth	2xVth	1xVth	1.5xVth	2xVth	1xVth	1.5xVth	2xVth
Case1	Low-Vth Std	1.04E-10	N/A		1.05E-10	N/A		1.000	N/A		1.000	N/A	
Case2	PD high-Vth	N/A	1.06E-10	1.08E-10	N/A	1.07E-10	1.11E-10	N/A	1.022	1.043	N/A	1.020	1.061
Case3	PD, WL high-Vth		1.16E-10	1.33E-10		1.17E-10	1.32E-10		1.111	1.280		1.117	1.262
Case4	PU, PD high-Vth		1.06E-10	1.10E-10		1.07E-10	1.10E-10		1.022	1.055		1.020	1.048
Case5	PU, PD, WL high-Vth		1.15E-10	1.33E-10		1.16E-10	1.32E-10		1.111	1.277		1.110	1.259
Case6	PD stack	1.42E-10	N/A		1.41E-10	N/A		1.368	N/A		1.345	N/A	
Case7	PD, WL stack	1.71E-10			1.76E-10			1.647			1.682		
Case8	PU, PD stack	1.40E-10			1.40E-10			1.348			1.341		
Case9	PU, PD, WL stack	1.77E-10			1.75E-10			1.704			1.678		
Case10	PD sleepy stack	N/A	1.33E-10	1.36E-10	N/A	1.32E-10	1.31E-10	N/A	1.276	1.307	N/A	1.263	1.254
Case11	PD, WL sleepy stack		1.52E-10	1.61E-10		1.50E-10	1.62E-10		1.458	1.551		1.435	1.546
Case12	PU, PD sleepy stack		1.33E-10	1.36E-10		1.35E-10	1.38E-10		1.275	1.306		1.287	1.319
Case13	PU, PD, WL sleepy stack		1.51E-10	1.67E-10		1.52E-10	1.57E-10		1.456	1.605		1.450	1.504

Although SRAM cell read time changes slightly as temperature changes, the impact of temperature on the cell read time is quite small. However, the impact of threshold voltage is large. We apply 1.5xV_{th} and 2xV_{th} for the high-V_{th} technique and the sleepy stack technique. As shown in Table 11, the delay penalty of the forced stack technique is between 35% and 70% compared to the standard 6-T SRAM cell. This is one of the primary reasons that the forced stack technique cannot use high-V_{th} transistors without incurring dramatic delay increase (e.g., 2X or more delay penalty is observed using either 1.5xV_{th} or 2xV_{th}).

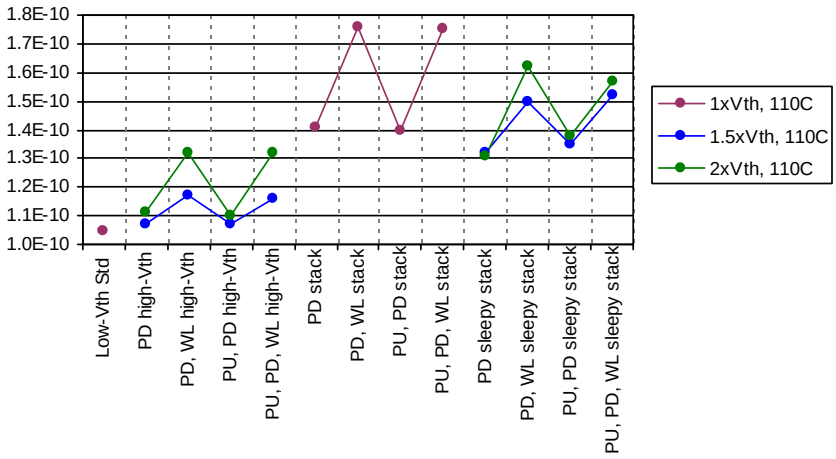


Figure 38: Worst case (at 110^oC) cell read time comparison

Let us focus on the worst case condition, i.e., at 110^oC shown in Figure 38. Among

the three low-leakage techniques, the sleepy stack technique is the second best in terms of cell read time. The PU, PD, WL high-V_{th} with 2xV_{th} is 16% faster than the PU, PD, WL sleepy stack with 2xV_{th} at 110^o. Since we are aware that area and delay are critical factors when designing SRAM, we will explore area and delay impact using tradeoffs in Section 7.2.4. However, let us first discuss leakage reduction (i.e., without yet focusing on tradeoffs, which will be the focus of Section 7.2.4).

7.2.3 Leakage power

We measure leakage power while changing threshold voltage and temperature because the impact of threshold voltage and temperature on leakage power is significant. Table 12 shows leakage power consumption with two high-V_{th} values, 1.5xV_{th} and 2xV_{th}, and two temperatures, 25^oC and 110^oC, where Case1 and the cases using the forced stack technique (Cases 6, 7, 8 and 9) are not affected by changing V_{th} because these use only low-V_{th}.

Table 12: Leakage power

	Technique	Leakage power (W)						Normalized leakage power					
		25°C			110°C			25°C			110°C		
		1xVth	1.5xVth	2xVth	1xVth	1.5xVth	2xVth	1xVth	1.5xVth	2xVth	1xVth	1.5xVth	2xVth
Case1	Low-Vth Std	9.71E-05	N/A		1.25E-03	N/A		1.0000	N/A		1.0000	N/A	
Case2	PD high-Vth	N/A	5.31E-05	5.12E-05	N/A	7.16E-04	6.65E-04	N/A	0.5466	0.5274	N/A	0.5711	0.5305
Case3	PD, WL high-Vth		2.01E-05	1.69E-05		3.20E-04	2.33E-04		0.2071	0.1736		0.2555	0.1860
Case4	PU, PD high-Vth		3.68E-05	3.45E-05		5.04E-04	4.42E-04		0.3785	0.3552		0.4022	0.3522
Case5	PU, PD, WL high-Vth		3.79E-06	1.38E-07		1.07E-04	8.19E-06		0.0391	0.0014		0.0857	0.0065
Case6	PD stack	5.38E-05	N/A		7.07E-04	N/A		0.5541	N/A		0.5641	N/A	
Case7	PD, WL stack	2.15E-05			3.20E-04			0.2213			0.2554		
Case8	PU, PD stack	3.75E-05			4.95E-04			0.3862			0.3950		
Case9	PU, PD, WL stack	5.39E-06			1.04E-04			0.0555			0.0832		
Case10	PD sleepy stack	N/A	5.18E-05	5.16E-05	N/A	6.62E-04	6.51E-04	N/A	0.5331	0.5315	N/A	0.5282	0.5192
Case11	PD, WL sleepy stack		1.80E-05	1.77E-05		2.45E-04	2.28E-04		0.1852	0.1827		0.1955	0.1820
Case12	PU, PD sleepy stack		3.54E-05	3.52E-05		4.43E-04	4.31E-04		0.3646	0.3630		0.3534	0.3439
Case13	PU, PD, WL sleepy stack		1.62E-06	3.24E-07		2.09E-05	2.95E-06		0.0167	0.0033		0.0167	0.0024

7.2.3.1 Results at 25^oC

Our results at 25^oC show that Case5 is the best with 2xV_{th} and Case13 is the best with 1.5xV_{th}. Specially, at 1.5xV_{th}, Case5 and Case13 achieve 25X and 60X leakage reduction over Case1, respectively. However, the leakage reduction comes with delay increase. The delay penalty is 11% and 45%, respectively, compared to Case1.

7.2.3.2 *Results at 110^oC*

Absolute power consumption numbers at 110^oC show more than 10X increase of leakage power consumption compared to the results at 25^oC. This could be a serious problem for SRAM because SRAM often resides next to a microprocessor whose temperature is high.

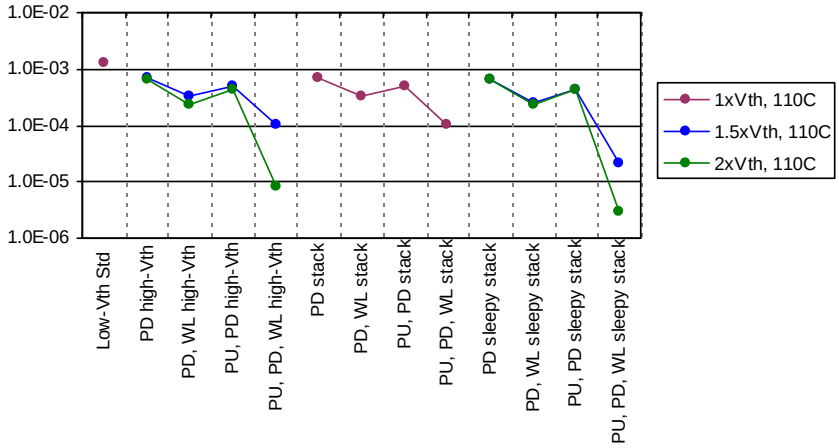


Figure 39: Worst case (at 110^oC) leakage comparison

At 110^oC, the sleepy stack technique shows the best result in both 1.5xV_{th} and 2xV_{th} even compared to the high-V_{th} technique (see Figure 39. The leakage performance degradation under high temperature is very noticeable with the high-V_{th} technique and the forced stack technique. For example, at 25^oC the high-V_{th} technique with 1.5xV_{th} (Case5) and the forced stack technique (Case9) show around 96% leakage reduction. However, at 110^oC the same techniques show around 91% of leakage power reduction compared to Case1. Only the sleepy stack technique achieves superior leakage power reduction; after increasing temperature, the sleepy stack SRAM shows 5.1X and 4.8X reductions compared to Case5 and Case9, respectively, with 1.5xV_{th}.

When the low-leakage techniques are applied only to the pull-up and pull-down transistors, leakage power reduction is at most 65% (2xV_{th}, 110^oC) because `bitline` leakage cannot be suppressed. The remaining 35% of leakage power can be suppressed by applying low-leakage techniques to `wordline` transistors. This implies that `bitline` leakage power addresses around 35% of SRAM cell leakage power consumption. This trend is be

observed for all three techniques compared, i.e., high-V_{th}, forced stack, sleepy stack.

7.2.4 Tradeoffs in low-leakage techniques

Although the sleepy stack technique shows superior results in terms of leakage power, we need to explore area, delay and power together because the sleepy stack technique comes with non-negligible area and delay penalties. To be compared with the high-V_{th} technique at the same cell read time, we consider four more cases other than shown in Table 6. We increase all transistors placed in the `wordline` and all transistors placed in pull-down of the sleepy stack SRAM. Then, for the sleepy stack technique, we find new transistor widths of `wordline` transistors and pull-down transistors, which result in delay approximately equal to the delay of the 6-T high-V_{th} case, i.e., Case5. The new cases are marked with '*' (Cases 10*, 11*, 12*, 13*). The results are shown in Table 13 and Table 14. To enhance readability of tradeoffs, each table is sorted by leakage power. Although we compared four different simulation conditions, we take the condition with 1.5xV_{th} at 110^oC and 2xV_{th} at 110^oC as important representative technology points at which to compare the tradeoffs between techniques. We choose 110^oC because generally SRAM operates at a high temperature and also because high temperature is the "worst case."

Table 13: Tradeoffs (1.5xV_{th}, 110^oC)

	Technique	Leakage power (W)	Delay (sec)	Area (u^2)	Normalized leakage power	Normalized delay	Normalized area
Case1	Low-Vth Std	1.254E-03	1.05E-10	17.21	1.000	1.000	1.000
Case2	PD high-Vth	7.159E-04	1.07E-10	17.21	0.571	1.020	1.000
Case6	PD stack	7.071E-04	1.41E-10	16.22	0.564	1.345	0.942
Case10*	PD sleepy stack*	6.744E-04	1.15E-10	25.17	0.538	1.102	1.463
Case10	PD sleepy stack	6.621E-04	1.32E-10	22.91	0.528	1.263	1.331
Case4	PU, PD high-Vth	5.042E-04	1.07E-10	17.21	0.402	1.020	1.000
Case8	PU, PD stack	4.952E-04	1.40E-10	15.37	0.395	1.341	0.893
Case12*	PU, PD sleepy stack*	4.532E-04	1.15E-10	31.30	0.362	1.103	1.818
Case12	PU, PD sleepy stack	4.430E-04	1.35E-10	29.03	0.353	1.287	1.687
Case3	PD, WL high-Vth	3.203E-04	1.17E-10	17.21	0.256	1.117	1.000
Case7	PD, WL stack	3.202E-04	1.76E-10	19.96	0.255	1.682	1.159
Case11*	PD, WL sleepy stack*	2.721E-04	1.16E-10	34.40	0.217	1.111	1.998
Case11	PD, WL sleepy stack	2.451E-04	1.50E-10	29.87	0.196	1.435	1.735
Case5	PU, PD, WL high-Vth	1.074E-04	1.16E-10	17.21	0.086	1.110	1.000
Case9	PU, PD, WL stack	1.043E-04	1.75E-10	19.96	0.083	1.678	1.159
Case13*	PU, PD, WL sleepy stack*	4.308E-05	1.16E-10	41.12	0.034	1.112	2.389
Case13	PU, PD, WL sleepy stack	2.093E-05	1.52E-10	36.61	0.017	1.450	2.127

Table 14: Tradeoffs (2.0xV_{th}, 110^oC)

	Technique	Static (W)	Delay (sec)	Area (u^2)	Normalized leakage	Normalized delay	Normalized area
Case1	Low-Vth Std	1.25E-03	1.05E-10	17.21	1.000	1.000	1.000
Case6	PD stack	7.07E-04	1.41E-10	16.22	0.564	1.345	0.942
Case2	PD high-Vth	6.65E-04	1.11E-10	17.21	0.530	1.061	1.000
Case10	PD sleepy stack	6.51E-04	1.31E-10	22.91	0.519	1.254	1.331
Case10*	PD sleepy stack*	6.51E-04	1.31E-10	22.91	0.519	1.254	1.331
Case8	PU, PD stack	4.95E-04	1.40E-10	15.37	0.395	1.341	0.893
Case4	PU, PD high-Vth	4.42E-04	1.10E-10	17.21	0.352	1.048	1.000
Case12*	PU, PD sleepy stack*	4.31E-04	1.33E-10	29.48	0.344	1.270	1.713
Case12	PU, PD sleepy stack	4.31E-04	1.38E-10	29.03	0.344	1.319	1.687
Case7	PD, WL stack	3.20E-04	1.76E-10	19.96	0.255	1.682	1.159
Case3	PD, WL high-Vth	2.33E-04	1.32E-10	17.21	0.186	1.262	1.000
Case11*	PD, WL sleepy stack*	2.29E-04	1.30E-10	32.28	0.183	1.239	1.876
Case11	PD, WL sleepy stack	2.28E-04	1.62E-10	29.87	0.182	1.546	1.735
Case9	PU, PD, WL stack	1.04E-04	1.75E-10	19.96	0.083	1.678	1.159
Case5	PU, PD, WL high-Vth	8.19E-06	1.32E-10	17.21	0.007	1.259	1.000
Case13*	PU, PD, WL sleepy stack*	3.62E-06	1.32E-10	38.78	0.003	1.265	2.253
Case13	PU, PD, WL sleepy stack	2.95E-06	1.57E-10	36.61	0.002	1.504	2.127

In Table 13 and Table 14, we observe seven and six Pareto points, respectively, which are in shaded rows, considering three variables of leakage, delay, and area. For both results, Case13 shows the lowest possible leakage, 2.7$\sim$5.1X smaller than the leakage of any of the prior approaches considered; however, there is a corresponding delay and area penalty. Alternatively, Case13* shows the same delay (within 0.2%) as Case5 and 2.26$\sim$2.5X leakage reduction over Case5. In short, this paper presents new, previously unknown Pareto points at the low-leakage end of the spectrum.

Please note that we do not vary sleep transistor width (e.g., we do not increase width even more) or transistor width of the in parallel transistor (e.g., we do not decrease to minimum the width of the transistor in parallel with the sleep transistor). Such additional optimizations can reduce delay further at a potential cost of increased area. However, we have nonetheless performed a broad search of the design space to capture important characteristics; of course, the design space is exponential and can be further explored!

Table 15: Active power

	Technique	Active power (W)						Normalized active power					
		25°C			110°C			25°C			110°C		
		1xVth	1.5xVth	2xVth	1xVth	1.5xVth	2xVth	1xVth	1.5xVth	2xVth	1xVth	1.5xVth	2xVth
Case1	Low-Vth Std	8.19E-04	N/A		2.04E-03	N/A		1.000	N/A		1.000	N/A	
Case2	PD high-Vth	N/A	7.67E-04	7.48E-04	N/A	1.48E-03	1.41E-03	N/A	0.936	0.913	N/A	0.724	0.691
Case3	PD, WL high-Vth		7.02E-04	6.78E-04		1.26E-03	9.75E-04		0.858	0.829		0.618	0.478
Case4	PU, PD high-Vth		7.60E-04	7.31E-04		1.17E-03	1.19E-03		0.928	0.893		0.572	0.582
Case5	PU, PD, WL high-Vth		6.86E-04	6.89E-04		8.82E-04	7.50E-04		0.838	0.842		0.432	0.368
Case6	PD stack	7.58E-04	N/A		1.37E-03	N/A		0.926	N/A		0.669	N/A	
Case7	PD, WL stack	5.45E-04			8.12E-04			0.665			0.398		
Case8	PU, PD stack	7.41E-04			1.22E-03			0.905			0.596		
Case9	PU, PD, WL stack	5.22E-04			5.97E-04			0.637			0.293		
Case10	PD sleepy stack	N/A	8.03E-04	8.03E-04	N/A	1.65E-03	1.66E-03	N/A	0.981	0.981	N/A	0.807	0.811
Case11	PD, WL sleepy stack		6.32E-04	5.87E-04		1.20E-03	1.22E-03		0.773	0.717		0.586	0.600
Case12	PU, PD sleepy stack		7.87E-04	8.23E-04		1.60E-03	1.63E-03		0.961	1.005		0.786	0.797
Case13	PU, PD, WL sleepy stack		5.89E-04	5.80E-04		1.20E-03	1.11E-03		0.719	0.708		0.588	0.546

7.2.5 Active power

Table 15 shows power consumption during read operations. The active power consumption includes dynamic power used to charge and discharge SRAM cells plus leakage power consumption. At 25^oC leakage power is less than 20% of the active power in case of the standard low-V_{th} SRAM cell in 0.07μ technology according to BPTM [7]. However, leakage power increases 10X as the temperature changes to 110^oC although active power increases 3X. At 110^oC, leakage power is more than half of the active power from our simulation results. Therefore, without an effective leakage power reduction technique, total power consumption – even in active mode – is affected significantly.

7.2.6 Static noise margin

Table 16: Static noise margin

	Technique	Static noise margin (V)	
		Active mode	Sleep mode
Case1	Low-Vth Std	0.299	N/A
Case10	PD sleepy stack	0.317	0.362
Case11	PD, WL sleepy stack	0.324	0.363
Case12	PU, PD sleepy stack	0.299	0.384
Case13	PU, PD, WL sleepy stack	0.299	0.384

Changing the SRAM cell structure may change the static noise immunity of the SRAM cell. Thus, we measure the Static Noise Margin (SNM) of the sleepy stack SRAM cell

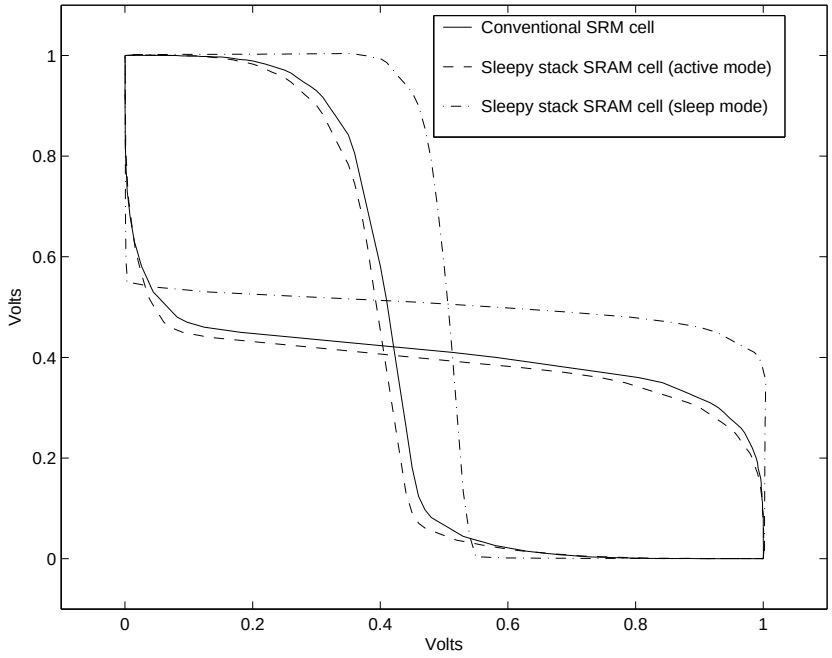


Figure 40: Static noise margin analysis

and the conventional 6-T SRAM cell using the butterfly plots shown in Figure 40. The SNM is defined by the size of the maximum nested square in a butterfly plot. The SNM of the sleepy stack SRAM cell is measured twice in active mode and sleep mode, and results are shown in Table 16. The SNM of the sleepy stack SRAM cell in active mode is $0.299V$ and almost exactly the same as the SNM of a conventional SRAM cell; the SNM of a conventional SRAM cell is $0.299V$. Although we do not perform a process variation analysis, we expect that the high SNM of the sleepy stack SRAM cell makes the technique as immune to process variations as a conventional SRAM cell.

7.3 Summary

We compared the sleepy stack technique to existing techniques in terms of delay, dynamic power, leakage power and area. The empirical analysis in Section 7.1.2 shows that we can increase V_{th} up to 2.1X while matching the delay to the forced stack technique. The increased V_{th} directly affects leakage power consumption. We apply the sleepy stack technique to a chain of 4 inverters, a 4:1 multiplexer and a 4-bit adder, achieving up to

200X leakage power reduction compared to the forced stack technique with 50$\sim$120% area penalty.

We also apply the sleepy stack technique to SRAM cell design. The sleepy stack SRAM cell achieves 2.5X leakage power reduction compared to the high-V_{th} SRAM cell when delay is matched at 110^oC with 1.5xV_{th} and 2xV_{th}. Although the sleepy stack SRAM cell comes with some area and delay penalty, the sleepy stack performs the best for a system which requires the lowest possible leakage power consumption while saving state.

In the next chapter, we will explain our low power pipelined cache (LPPC).

CHAPTER VIII

LOW-POWER PIPELINED CACHE (LPPC) ARCHITECTURE

In this chapter, we present our Low Power Pipelined Cache (LPPC) for dynamic power reduction. LPPC uses a pipelined cache, a cache technique widely used to enhance cache performance by means of pipelining. Unlike earlier pipelined cache techniques, we use cache pipelining to save power consumption of a cache by lowering supply voltage. We explore two pipelining techniques – standard (latch) pipelining and wave pipelining – and compare advantages and disadvantages. Finally, we discuss potential disadvantages of the LPPC. Pipelining a cache may come with processor pipelining penalties. We explore possible penalties and appropriate solutions.

8.1 Background of a Pipelined Cache

Although pipelining a cache is briefly described in Section 4.2.3, in this section we explain much more detail about the structure and operation of a pipelined cache. These details become quite important later when we describe the basic architecture of our experimental setup.

As explained in Section 4.2.3, a cache can be pipelined to increase cache access bandwidth. This technique is called the pipelined cache technique. The pipelined cache technique was introduced initially to improve pipelined processor performance (clock speed).

Figure 41 depicts non-pipelined and pipelined cache architectures. Figure 41(a) shows a 5-stage processor pipeline with non-pipelined caches. The five stages are Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), MEMory Access (MEM), and Write Back (WB). The IF and MEM stages access an Instruction cache (I-cache) and a Data

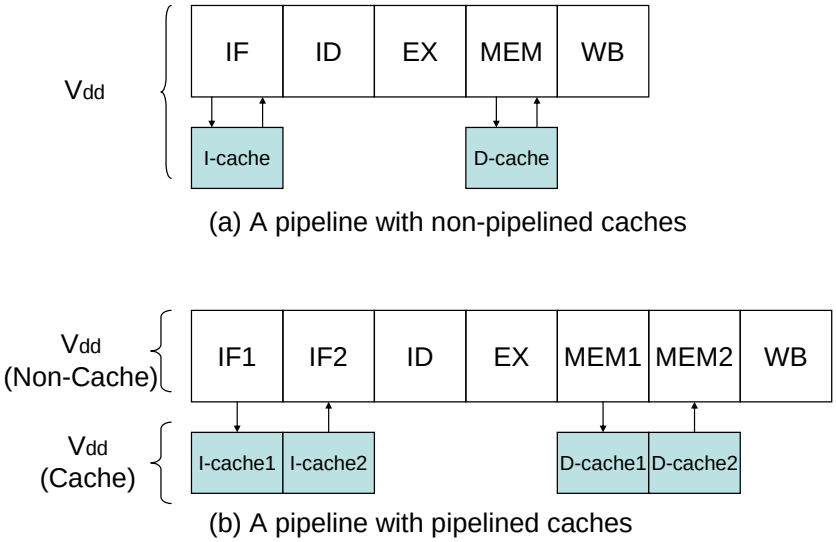


Figure 41: Non-pipelined and pipelined cache architectures

cache (D-cache), respectively. If the access time of either cache is larger than the delay of any other stage (i.e., ID, EX or WB), the cycle time of the processor must be increased to accommodate the cache access time, thus resulting in performance degradation. One prominent solution to this problem is to pipeline one or both caches. Figure 41(b) shows a case where the cache access pipeline stages (IF and MEM) are both broken into two stages and pipelined. The IF stage turns into IF1 and IF2, and the MEM stage turns into MEM1 and MEM2. The I-cache and D-cache are pipelined corresponding to the pipeline stages, forming I-cache1–I-cache2 and D-cache1–Dcache2, respectively. Thus, a smaller processor cycle time can be achieved, potentially resulting in performance improvement.

Since a processor pipeline deepens as a pipelined cache deepens, cache pipelining may incur increased branch delays and load delays due to control hazards and data hazards. A branch delay occurs when a branch instruction is inserted into a pipeline and a branch prediction scheme is not used; thus, the following instructions have to wait until the branch target is resolved, which is a control hazard. In a pipelined processor as shown in Figure 41, the number of waiting cycles, called branch delay cycles, will increase if the IF stage uses a pipelined I-cache. For example, in Figure 41(a), the branch target is resolved in EX and thus the following instruction, if no branch prediction scheme is used, needs to wait for

two extra cycles. If the IF stage is expanded as shown in Figure 41(b), the next instruction (after the branch) needs to wait for three extra cycles.

When a load instruction is fetched and fed into a pipeline such as Figure 41(a), subsequent instructions which are dependent on the load instruction cannot proceed to the EX stage until the load instruction fetches data: this is a data hazard due to a load delay. If added, an extra MEM2 stage for a pipelined D-cache induces extra load delay penalties. For example, if a load instruction is in MEM in Figure 41(a), the dependent instruction in the ID stage needs to wait, causing a one cycle delay. If the MEM stage is expanded as shown in Figure 41(b), the load instruction fetches data at the end of MEM2, and thus the next dependent instruction may have to wait for up to two cycles. A pipelined cache architecture reduces clock cycle time yet at the same time increases branch and load delay penalties. Therefore, there apparently exists an optimal depth of pipeline cache stages that can maximize performance under particular conditions [52].

Although the pipelined cache technique is introduced to improve performance, alternatively we use the reduced delay to trade off for reduced power consumption. In the next section, we explain how we use the pipelined cache to save power consumption.

8.2 Low-Power Pipelined Cache Architecture

In this section, we explain the basic idea of the low-power pipelined cache. Furthermore, we study the issues related to cache pipelining, such as implementation of pipelining and pipelining penalties.

8.2.1 Low-power pipelined cache energy savings

Our motivation for an LPPC mainly comes from the EDR paradigm discussed in Section 4.2.2. If we apply the EDR paradigm when designing a pipelined processor, minimum power consumption is achieved when the ratio of energy and delay of each stage are the same. We assume that each pipeline stage is area already optimized and thus no surplus

slack exists. Our strategy is to add slack by splitting power hungry stages into two or more as needed. Such a split stage will have surplus slack. Since caches are power hungry modules in a processor, we provide surplus slack to cache access stages (i.e., IF and MEM) using pipelined caches.

We assume for our base case a pipelined processor that has 5-stages, a non-pipelined I-cache and a non-pipelined D-cache as shown in Figure 41(a). We further assume that each stage of the datapath is optimized to a given cycle time restriction. If we now pipeline the caches without altering the existing cycle time, the pipelined caches will have surplus slack, the difference between the cycle time and the delay of the particular stage. Then, we can use the slack to lower the supply voltage of caches for power reduction purposes. Therefore, in Figure 41(b), $V_{dd}(Cache)$ will use a lower supply voltage V_{ddl} while $V_{dd}(Non-Cache)$ remains the same. We can lower the cache supply voltage in Figure 41(b) as long as the delay of each pipelined cache stage is less than the existing cycle time. Since we achieve power savings from lowering the V_{dd} for the caches, we expect that the power savings of a processor in our proposed architecture is dependent on the ratio of cache power consumption over total processor power consumption. This means that a processor with power hungry caches will benefit more from our LPPC.

Example 2: Let us take an example of saving energy with a pipelined cache architecture as shown in Figure 42, in which V_{dd} is supply voltage; C_L is load capacitance; f is frequency; $E.T.$ is execution time; and E is energy consumption. In Figure 42(a), the base case has 4.3 ns cycle time since we find that cache delay is 4.3 ns with $V_{dd} = 2.25V$. Then, the clock frequency f is 233MHz, and we assume $E.T. = 1sec$. Therefore, E of the base case becomes 0.589mJ. In Figure 42(b), the pipelined cache for high performance halves cycle time by splitting the stage and thus double frequency (466 MHz) as well as half $E.T.$ (0.5 sec) are achieved (energy consumption remains the same). Instead our pipelined cache for low power consumption in Figure 42(c) maintains the cycle time of the base case after splitting the stage into two; thus, there exists 2.15 ns extra slack.

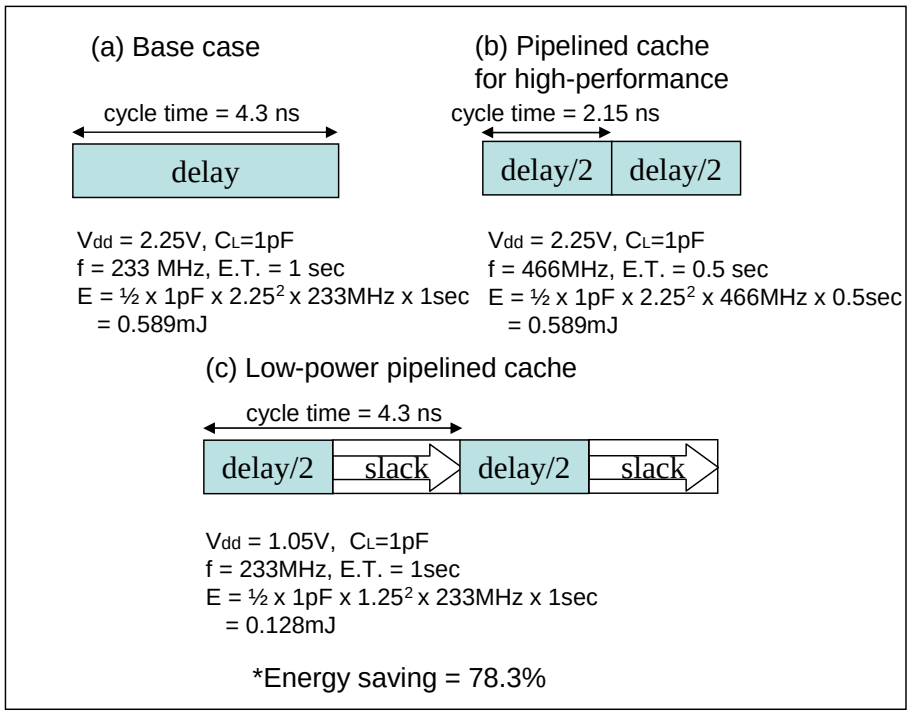


Figure 42: Illustration of energy of a pipelined cache

Therefore, now we can use the slack to lower the supply voltage as far as possible until the delay increases so much that it equals the initial cycle time. In this example, V_{dd} could be lowered to 1.05V so that the pipelined cache for low power would achieve 78.3% energy saving compared to the base case. □

Ideally, we can save significant power using the low-power pipelined cache. However, two issues need to be explored properly when we use the pipelined cache for the power reduction purposes. The following two sections (Sections 8.2.2 and 8.2.3) covers the two pipelining issues.

8.2.2 Pipelining techniques for LPPC

We observe significant power savings in Example 1. The power savings vary according to the method used to implement such a pipelined cache structure. Two typical ways are

as follows. One way is to place latches between pipeline stages, which is called conventional pipelining. Modern processors use this method for datapath pipelining. The main advantage of conventional pipelining is ease of implementing pipeline structures since the latches between pipeline stages are synchronized to a common clock. However, latches and additional clock distribution consume extra power. Furthermore, a cache structure may not pipeline evenly since modules such as `bitlines` cannot be pipelined [2]. The uneven distribution of delay restricts lowering the supply voltage of an LPPC. The other pipelining implementation uses existing gates as a virtual storage element instead of intermediate latches: this is called wave-pipelining (or virtual pipelining). Since wave-pipelining does not use latches, extra power consumption due to latches and clock distribution may be saved. Furthermore, we can distribute delay evenly over pipeline stages, although the number of waves in a stage is restricted by the delay variation of logic paths in a pipeline stage. In short, we may wave-pipeline caches to be able to use a supply voltage lower than latch-pipelined caches. Proper timing (for all paths) and testing are not easy problems in designing a wave-pipelined processor. However, since the simplicity of SRAM structure makes wave-pipelining easier, wave-pipelining is well suited for designing SRAM [10]. For example, UltraSPARC-IV uses a wave-pipelined SRAM design targeting 90nm technology [68], and Hitachi designed a 300-MHz, 4-Mbit wave-pipelined CMOS SRAM [31]. As a result, while a pipelined processor must use latches to split the IF and MEM stages (in-between IF1–IF2 and MEM1–MEM2 in Figure 41(b)), we compare two different cache pipelining methods (i.e., wave-pipelining and latch-pipelining) to implement pipelined caches (I-cache1–I-cache2 and D-cache1–D-cache2 in Figure 41(b)).

8.2.3 Pipelining penalties and solutions for LPPC

Another issue with pipelining a cache is pipeline penalties due to branch delay and load delay as described in Section 8.1. These two penalties should be handled properly in a pipelined cache processor, otherwise the processor may lose performance as the number

of pipelined cache stages increases. General techniques that minimize control hazards and data hazards in a pipelined processor can also be applied to a pipelined cache architecture. However, since some of the techniques are not adequate for low power processor design, we re-evaluate techniques that tackle control hazards and data hazards in term of power dissipation before choosing suitable techniques for our LPPC.

Control hazards have been mostly tackled with instruction speculation techniques, which predict the next instruction before a branch target is resolved. In these techniques, if a branch target is mispredicted, some instructions are unnecessarily fetched and thus must be wiped out. A deeper pipelining of the I-cache may cause a higher number of such penalties as explained in Section 8.1. The instruction speculation techniques can be classified as hardware based or software based techniques. A typical hardware technique uses a Branch Target Buffer (BTB). A BTB is a kind of cache that stores target branch addresses accessed previously. The software based speculation technique typically uses compilers to fill branch delay cycles with useful instructions if possible. Apparently, hardware based techniques consume additional power, which is less than 5% in the case of an Intel Pentium Pro processor [9]. However, a BTB shows better branch prediction accuracy [51], which is important not only from a performance point of view but also from an energy point of view because a processor with an inaccurate branch prediction scheme would consume extra energy due to flushed instructions when a misprediction occurs. In addition, the compiler based technique uses instruction replication; thus, the expanded code causes extra power consumption in an I-cache [51]. Therefore, BTB is widely used for modern pipelined processors (e.g., Intel XScale, Intel Pentium 4, and IBM PowerPC all use BTBs).

Data hazards have been also tackled using hardware based or software based techniques. A typical hardware technique is an out-of-order execution method used by a superscalar architecture, while software based techniques rearrange instructions to separate a load instruction as far as possible from its dependent instructions. Although a hardware

based technique shows better results, an out-of-order execution method requires a complicated structure. Furthermore, the extra hardware may cause cycle time to increase [51].

For our LPPC, while we adopt a BTB (a hardware technique) to minimize control hazards, and we utilize instruction scheduling of a compiler (a software based technique) to reduce data hazards. We save cache power consumption by adopting LPPC and lowering the supply voltage of caches although latches in-between the IF stage (IF1–IF2) and the MEM stage (MEM1–MEM2) still consume extra power. A deeper pipelined I-cache also dissipates extra power used to fetch unnecessary instructions when a branch is mispredicted. The lengthened execution time due to a deeper pipeline also increases leakage power. We measure execution time and power/energy consumption to find out an optimum LPPC depth for our target benchmarks and VLSI technology. We estimate dynamic power and energy dissipation targeting 0.25μ technology, and the result will be presented in the next section (please note that in Chapter 10, we combine LPPC and sleepy stack SRAM to save static power consumption).

8.3 Summary

In this chapter, we introduced Low-Power Pipelined Cache (LPPC), a new low-power technique. Previously, a pipelined cache is used to reduce cache access delay and thus reduce cycle time. Instead, LPPC uses the reduced cache access time to lower supply voltage. By using the same cycle time before and after cache pipelining, a pipelined cache can generate excess slack, which we can use to save power consumption. We discuss two cache pipelining techniques: one uses buffers between pipelined stages, while the other, wave-pipelining, use existing gates as virtual storage. Although wave-pipelining is not easy to implement, it has advantages over the buffer-pipelined cache. Furthermore, thanks to the regular structure of a cache, implementing wave-pipelining in a cache is relatively easy (compared to the rest of the microprocessor) and widely used. The two pipelining techniques will be compared in terms of delay in next chapter. We finally considered possible

pipelining penalties. Branch and load delays are explored. Our study shows that a BTB and a compiler optimization are prominent solutions for the branch delay and the load delay, respectively.

In the next chapter, we will explain the experimental results of LPPC.

CHAPTER IX

LOW-POWER PIPELINED CACHE (LPPC) EXPERIMENTAL SETUP AND RESULTS

In this chapter, we evaluate LPPC in terms of performance and power consumption. We setup an architectural infrastructure which can vary the number of pipeline stages as well as vary the supply voltage. Then we explain the cache delay model which is used to obtain resulting delaying due to lowering the supply voltage value of the cache. Finally, we provide experimental results that show the effectiveness of LPPC.

9.1 Experimental Setup

The LPPC experimental setup can measure execution cycles and power consumption of benchmarks while varying the number of pipelined cache stages and cache supply voltage. We use both a cycle accurate high level processor simulator as well as a Register Transfer Level (RTL) processor model to achieve both simulation speed and accuracy. We target an ARM-like architecture to verify our LPPC because the ARM architecture has been widely used in the embedded systems area [4][29]. We model a pipelined cache architecture by changing the target pipeline structure and buffer (latch) power consumption in-between the split IF and MEM stages. We choose V_{ddl} for a low voltage pipelined cache according to the cycle time of a pipelined cache that we measure. LPPC is evaluated using benchmarks selected to target embedded systems.

9.1.1 Processor Model

This section explains the processor model used to measure performance and power consumption of our low-power pipelined cache architecture. We describe the detailed experimental infrastructure and power modeling and measurement methodology.

9.1.1.1 Simplescalar/ARM+Wattch

We use Simplescalar/ARM for the performance evaluation of our target processor [61]. Simplescalar/ARM is a widely used processor model with a cycle accurate performance evaluation tool targeting an ARM processor. Since Simplescalar/ARM uses non-pipelined caches, we modified Simplescalar/ARM to simulate pipelined cache performance and power consumption; consequently, the modified Simplescalar/ARM increases branch and load delays as the pipelined I-cache and D-cache, respectively, deepen. However, these penalties can be reduced by using techniques explained in Section 8.2.3, which we propose.

Since Simplescalar/ARM did not have a power estimation tool at the time we downloaded it to perform this research, we integrated Simplescalar/ARM and the power model of Wattch, which is a widely used power estimation tool based on Simplescalar [9]. We further modified Simplescalar/ARM to calculate global clock power consumption according to the pipeline depth.

9.1.1.2 Buffer power model using MARS

We then added a buffer power consumption model using a synthesis based power estimation method, which is depicted in the left branch of Figure 43. Note that buffers inserted between split pipeline stages consume extra power as explained in Section 8.2.2. The core of the synthesis based power estimation method is MARS, which is a cycle-accurate Verilog model of a 5-stage ARM-like processor obtained from the University of Michigan [67]. The five stages of MARS are typical stages in the DLX architecture, i.e., IF, ID, EX, MEM and WB [27]. MARS has a non-pipelined I-cache and a non-pipelined D-cache.

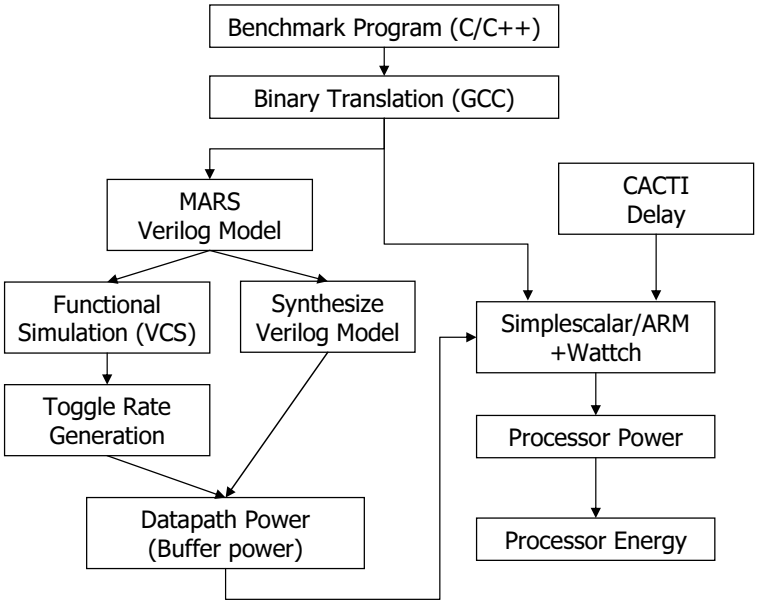


Figure 43: Performance and power simulation infrastructure

MARS uses a Backward-Taken Forward Not-taken (BTFN) branch prediction scheme to hide branch delay penalties. We use the original MARS for case (i) a non-pipelined cache architecture in Figure 41(a) in Section 8.1.

We modified MARS to obtain case (ii) described in Figure 41(b): a processor with two-stage pipelined instruction and data caches. For case (ii), we first modified the branch prediction scheme as explained in Section 8.2.3. We found that BTFN in MARS is not adequate for the pipelined I-cache because BTFN in MARS predicts the next instruction in the ID stage (not in the IF stage), where a datapath recognizes a branch instruction. (Please note that although implementing BTFN that predicts a next instruction in the IF stage is possible, we did not put in the extra effort to do so. Instead, we use a BTB which performs better than BTFN.) Although the BTFN can hide some of the branch delay penalties, BTFN still has non-zero branch delay penalties even when the prediction is correct. Unlike BTFN in MARS, since a Branch Target Buffer (BTB) enables a processor to predict the next instruction during the IF stage, we can potentially have a zero branch penalty if the BTB predicts always correctly [27]. As a result, we added a 128-entry BTB, which we use for Simplescalar/Wattch simulations, to MARS. The BTB has branch instruction addresses,

branch target addresses and a 2-bit counter for branch prediction [62]. Once the BTB mispredicts, the BTB table is updated while the pipeline wipes out mispredicted instructions from the pipeline.

Next, we expanded the pipeline of MARS. One extra stage (IF2) is inserted between IF (changed to IF1) and ID for the pipelined I-cache using a buffer. Another extra stage (MEM1) is inserted between EX and MEM (changed to MEM2) for the pipelined D-cache. Then, we modified branch control logic such that it calculates its target Program Counter (PC) value according to the newly added IF2 stage. In addition, we added one more data forwarding path from the MEM1 stage to the EX stage on top of the existing data forwarding paths (EX-to-EX and MEM2-to-EX).

Now, we explain the simulation procedure to extract buffer power consumption using MARS. First, we compile benchmarks using GNU-gcc ARM cross compiler version 2.95.2. Three benchmarks are used for buffer power modeling: (i) SORT_INT, a sorting program for integers; (ii) MATMUL, a matrix multiplication program; and (iii) FACTORIAL, a factorial calculation program. Each benchmark is compiled to relocatable ARM assembly code using the aforementioned GNU-gcc ARM cross compiler, then a binary executable targeted toward MARS is generated using a GNU cross-assembler. Second, we translate the binary into an ASCII format, called Verilog HeX (VHX), which is a suitable input format for MARS. Third, for each benchmark, the switching activities (i.e., p_t for each wire in the processor – see Chapter 3) of MARS and cache statistics (specifically, numbers of I-cache and D-cache accesses) are collected through Synopsys VCS simulation [65].

We use a synthesis based methodology to develop a power model for submodules belonging to the datapath. Submodules consist of the fetch unit, decode unit, register file, arithmetic logic unit, D-cache access unit and write-back unit. The synthesis infrastructure employs two software tools from Synopsys Inc.: Design Compiler and Power Compiler [65]. Design Compiler generates a gate level netlist from the hardware description of the submodules given as Verilog RTL description. The netlist is generated using the TSMC

0.25μ library from LEDA systems [39]. The technology details include features such as the transistor width, transistor length, gate capacitance, drain capacitance, rise time and fall time of each transistor. During the synthesis process, maximum delay is set to 10 ns, and maximum area is fixed to infinity in order to achieve the fastest implementation. In our case, the modules are synthesized to operate at 100 MHz (i.e., 10 ns cycle time). Power Compiler is used to estimate the power of the processor core. Switching activities collected from VCS simulation are fed to Power Compiler as input. Power Compiler reports the dynamic power dissipation of the technology chosen (i.e., 0.25μ technology).

We obtain buffer power from Power Compiler and translate the buffer power into average energy consumed per buffer access. We use the calculated buffer access energy to calculate buffer power consumption when we measure pipelined cache processor power using Simplescalar/ARM+Wattch. Simplescalar/ARM+Wattch calculates buffer power consumption by multiplying the buffer access energy by the number of buffer accesses. Furthermore, we scale the buffer access energy according to the V_{dd} values used in Simplescalar/ARM+Wattch.

9.1.1.3 Power overheads

In multiple voltage supply circuits, the effect of level converters needs to be considered. Usami et al. reported that the level converters employed in a particular media processor number more than 5000 and consume 8% of total processor power [70]. Unlike [70], our LPPC needs level converters in the connections between I-cache2–IF2 and D-cache2–MEM2 in Figure 41(b). If we consider the width of the data bus, i.e., 32 bits, the required number level converters in LPPC are $1\sim2\%$ of the number used in Usami's media processor. Since the power consumption of our level converters is very small, we do not consider the power overhead of level converters in our power and energy evaluations. In addition, our calculations do not include the extra overhead of multiple supply voltage generation

since we assume that the board already has the multiple supply voltages needed. For example, the "Skiff" Personal Server Board from HP/Compaq [28] has 2 Volt, 3.3 Volt and 5 Volt power supplies. In addition, the StrongARM SA-110 processor operates at 3.3V for the I/O interface and in-between 1.65V and 2.0V for the processor core [29]. Therefore, it is reasonable to assume that our system has at least three supply voltages readily available and that the processor is capable of operating at dual voltages.

We explained the processor performance and power measuring methodology in this section. In next section, we present the cache delay model, which calculates the cache voltage level of our low-power pipelined cache.

9.1.2 Cache Delay Model

LPPC saves power consumption by lowering a cache V_{dd}, which is chosen as the lowest voltage level that can achieve the same smaller cycle time than the cycle time before pipelining in a given pipeline depth. To estimate the cycle time of a cache as V_{dd} scales down, we use CACTI cache model version 2.0 [57], which is a well-known cache model that integrates timing and power estimation. CACTI 2.0 has a detailed model of the wire and transistor structure of on-chip memories and provides very detailed capacitance values for each circuit component which are verified by HSPICE. CACTI can also measure the cycle time of a wave-pipelined cache.

Table 17: Cache configuration parameters

Descriptions	Parameters
Cache size	32KB
Block size	32 bytes
Associativity	4-way
Number of tag word line	1
Number of tag bit line	2
Number of data word line	2
Number of data bit line	2
Number of input/output port	1

Table 17 shows parameters used in our CACTI simulation. The original CACTI model

optimizes the number of wordlines and bitlines in a given cache size and CMOS technology. To compare the cycle time in our CACTI simulation without changing the number of wordlines and bitlines, we fixed the number of wordlines and bitlines to the optimized values that are given by CACTI with 32KB cache size, 32 byte block size, 4-way associativity and 0.25μ CMOS technology. The original CACTI model uses an RC based timing model. Instead, we use delay model Equation 14 in Chapter 3, so that we can evaluate the impact of supply voltage scaling in a give CMOS technology.

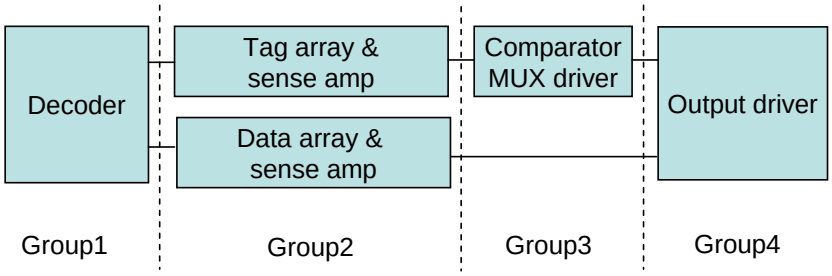


Figure 44: Latch-based cache pipelining

We apply the two different pipelined cache implementation schemes mentioned in Section 8.2.2. One is latch-pipelining. First, we break CACTI into four different functional groups considering the delay of each functional group as shown in Figure 44. The four groups would be pipeline stages for a 4-stage pipelined cache. Then, we decide 2-stage, 3-stage and 4-stage pipelined cache structures by merging adjacent groups such that the delay of each stage is as even as possible. The delay overhead due to latches is considered using delay values from [64]. For the wave-pipelined cache, we collect cache access time reported by CACTI while varying the number of wave pipeline stages.

Using LPPC, we lower the supply voltage instead of reducing the cycle time as shown in Figure 42. This means that we can choose the lowest V_{dd} for a pipelined cache while the cycle time of a pipelined cache is equal to or smaller than the cycle time of a non-pipelined cache. To obtain the lowest V_{dd} for LPPC, we measure the delay variations of CACTI by varying supply voltages and depth in a pipelined cache as shown in Figure 45. We assume that caches in a processor function properly to as low as 0.7V (please note that, however,

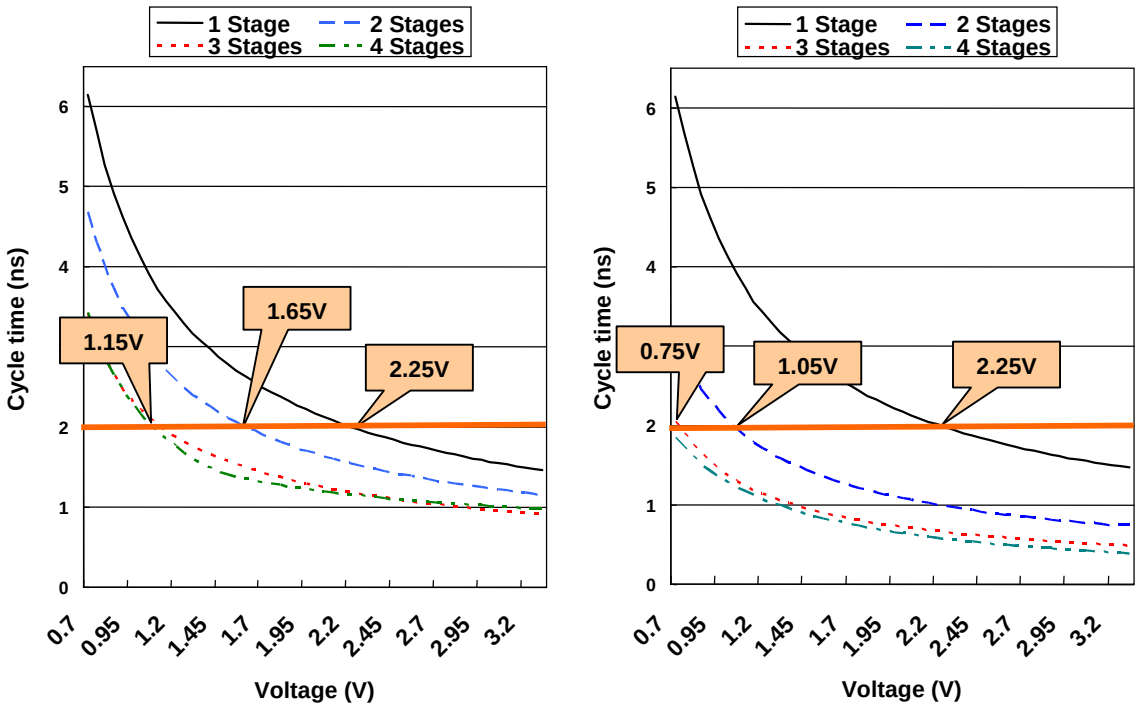


Figure 45: Delay of a latch-pipelined(left) and wave-pipelined(right) cache

as did not design sense amplifiers for each supply voltage). A lower supply voltage for a pipelined cache can be chosen from this graph. The following example shows how to select the V_{dd} of LPPC using Figure 45.

Example 2: Consider a processor with a non-pipelined cache of 2.25V supply voltage using the cache configuration parameters in Table 17. The delay of the cache is 1.97 ns from Figure 45. The non-pipelined cache is replaced with a multi-stage pipelined cache. If LPPC adopts a 2-stage latch-pipelined cache, we can lower the V_{dd} of the 2-stage pipelined cache to as low as 1.65V without increasing the cycle time. If LPPC adopts a 2-stage wave-pipelined cache, we can use 1.05V V_{dd}. Furthermore, we can use a 0.75V V_{dd} if we use a 3-stage wave-pipelined cache instead of a non-pipelined cache with a 2.25V V_{dd}. This lowered supply voltage directly impacts on the energy savings of a cache. As such, the 2-stage and 3-stage wave-pipelined cache can save 78% and 89% of energy consumed in a cache, respectively. □

The delay of a cache is not reduced linearly as the pipeline deepens as shown in Figure 45. This sub-linear delay reduction is caused by uneven pipelining in the case of a

latch-pipelined cache. In the case of a wave-pipelined cache, delay reduction from 1-stage to 2-stage is different from the reduction from 2-stage to 3-stage because the wave pipelining of CACTI is restricted by delay of each block in Table 44; thus, the delay time of the CACTI cache model cannot be smaller than each logic stage delay. We do not discuss latch-pipelined caches in the later sections of this chapter since a wave-pipelined cache outperforms a latch-pipelined cache as shown in Figure 45.

9.1.3 Architecture Configurations and Benchmarks

The Simplescalar/ARM+Wattch configuration is modeled after the Intel StrongARM microarchitecture. Since StrongARM does not use a branch target buffer (BTB), the BTB configuration follows the Intel XScale configurations [25]. The memory system consists of two levels. The first level is L1 caches (one instruction cache and one data cache), and the second level is off-chip main memory. We assume that benchmark programs fit into the main memory. The detail configuration is shown in Table 18.

Table 18: Simplescalar configurations

Execution type	In-order
Fetch queue	2
Branch predictor	128 entry BTB, 8K bimodal
Block Fetch & Decode width	1
Functional Units	1 int ALU, 1 FP ALU 1 FP mult
L1 I-cache	32KB, 4-way
L1 D-cache	32KB, 4-way
L2 cache	None
Memory bus width	4
Memory latency	12
Clock speed	233MHz
V_{dd} (Core)	2.25V
V_{dd} (Cache)	2.25V, 1.05V, 0.75V

We use a TSMC 0.25μ technology library from LEDA systems to estimate buffer power as explained at the beginning of this section. Therefore, Simplescalar/ARM+Wattch and

CACTI are scaled to 0.25μ technology. A clock speed of 233 MHz is used since it is the fastest clock speed used in StrongARM-110 [29]. Simplescalar/ARM+Wattch uses a 2.25V supply voltage when Simplescalar/ARM+Wattch is targeted 0.25μ technology [9]. Therefore, the supply voltage for the datapath (without caches) of the processor is fixed to 2.25V. The cache uses different voltages according to the depth of the pipelined cache stages. The 1-stage cache uses the same supply voltage as the datapath, 2.25V. The 2-stage, 3-stage and 4-stage pipelined caches use 1.05V, 0.75V and 0.7V respectively, which, as shown in Figure 45, are the lowest voltages that have equal or smaller delay time than a 1-stage (i.e., unpipelined) cache.

Table 19: Benchmarks

Name	Category	Characteristics
DIJKSTRA	Network	calculating the shortest path between every pairs in a tree structure using Dijkstra algorithm
DJPEG	Consumer	decompressing images using JPEG algorithm
GSM	Telecom	decoding a voice using the Global Standard for Mobile (GSM) standards
QSORT	Automotive and industrial control	sorting large array of strings with quick sort algorithm
SHA	Security	producing 160-bits message digest for a given input
STRINGSEARCH	Office	searching for given words using a case insensitive algorithm
MPEG2DEC	N/A	decoding MPEG2 video file

We evaluate our LPPC with benchmarks from embedded systems domains, which include the MiBench benchmark suit [25] and the software MPEG2 decoder v1.1 from MPEG Software Simulation Group [44]. MiBench is a set of benchmarks targeting embedded system performance evaluations. MiBench classifies embedded system applications into six categories. We chose one benchmark from each category. The MPEG2 decoder decodes an MPEG2 video file of 170(width)x128(height)x3(frames). Table 19 shows the benchmarks and functionalities that we chose.

A compiler optimization is used to minimize data hazards as explained in Section 8.2.3.

Each benchmark is compiled using ARM-LINUX-GCC cross compiler version 2.95.2 with *-fschedule-insns2* and *-O3* options. The *-fschedule-insns2* option combined with *-O3* reorders instructions to avoid stalls due to unavailable data if possible [71]. These two options optimize instructions to reduce load delay penalties.

The LPPC model is simulated using the benchmark programs explained in this section. The next section studies the experimental results in terms of performance and power.

9.2 Experimental Results

Our proposed Low Voltage Pipeline Cache (LPPC) saves power and energy by lowering the V_{dd} of caches and maintains system throughput by pipelining the caches as described in Section 8.2. We evaluate the power efficiency of our LPPC in comparison with the non-pipelined cache architecture using the experimental environment and benchmarks from embedded system domains described in Section 9.1. The two architectures are compared in terms of execution cycles, power consumption and energy consumption. We take the *clock gate mode 3* from Simplescalar/ARM+Wattch for power and energy estimations [9].

9.2.1 Performance results

In this section, we compare execution cycles of non-pipelined cache and pipelined cache architectures. We use 2-stage, 3-stage and 4-stage pipelined caches for both the I-cache and the D-cache of the baseline architecture; we always keep both caches at equal pipeline depth (e.g., both a 3-stage pipelined I-cache and a 3-stage pipelined D-cache). The performance simulation results are shown in Table 20. We simulate seven different benchmark programs from the embedded system domain. The 1-stage cache result in column 1 is the result of the non-pipelined cache. Each pipelined cache result has four columns: the column “cycles” indicates execution cycles used in running each benchmark on the processor pipeline associated with a particular cache; the column “Total” means total cycle increase compared to non-pipelined cache architecture; and the column “Icache” and “Dcache” each

refer to increase in execution cycles due to the pipelined I-cache and the pipelined D-cache, respectively.

From the results, we can notice that using a pipelined cache is not free because increasing pipelining depth increases pipelining penalties even though we use techniques to hide the pipelining penalties. The result shows 4.14%, 11.53% and 14.42% of execution cycle increment for 2-stage, 3-stage and 4-stage pipelined cache processor, respectively, on average. The performance penalty due to increased pipelining shows different results according to the benchmarks. Some benchmarks such as the GSM and SHA benchmarks show increments less than 2% with a 2-stage pipelined cache. In contrast, the DIJKSTRA benchmark increases execution cycles more than 9% due to the increased pipeline depth. More detailed explanations can be found by examining the “Icache” and “Dcache” columns. From the average for a 2-stage pipelined cache, we can see that the D-cache accounts for 77% of the total increase in number of clock cycles needed to execute the benchmarks. Even though the DIJKSTRA increases 9.4% due to the 2-stage pipelined caches, the majority of the cycle increase is due to the pipelined D-cache. The pipelined I-cache increases execution cycles only 1.13% with the DIJKSTRA benchmark. The STRINGSEARCH benchmark

Table 20: Execution cycles

	1-stage cache				2-stage pipelined cache			
		Increase (%)				Increase (%)		
Benchmark	cycles	Total	Icache	Dcache	cycles	Total	Icache	Dcache
DIJKSTRA	100,437,638	N/A	N/A	N/A	109,881,681	9.40	1.13	8.27
DJPEG	10,734,606	N/A	N/A	N/A	11,380,700	6.02	0.34	5.68
GSM	21,522,735	N/A	N/A	N/A	21,859,916	1.57	0.46	1.11
MPEG2DEC	28,461,724	N/A	N/A	N/A	29,398,489	3.29	0.28	3.01
QSORT	90,206,190	N/A	N/A	N/A	93,280,904	3.41	1.91	1.50
SHA	17,533,248	N/A	N/A	N/A	17,600,241	0.38	0.34	0.04
STRINGSEARCH	6,356,925	N/A	N/A	N/A	6,668,413	4.90	2.13	2.77
Average						4.14	0.94	3.20
	3-stage pipelined cache				4-stage pipelined cache			
		Increase (%)				Increase (%)		
Benchmark	cycles	Total	Icache	Dcache	cycles	Total	Icache	Dcache
DIJKSTRA	127,199,801	26.65	2.38	24.26	141,488,147	30.11	5.11	25.00
DJPEG	12,243,399	14.06	0.74	13.32	13,091,913	15.41	1.84	13.57
GSM	23,078,322	7.23	1.03	6.19	24,173,369	11.08	1.59	9.49
MPEG2DEC	31,741,379	11.52	1.27	10.25	33,969,889	15.87	2.37	13.50
QSORT	97,111,275	7.65	4.10	3.55	100,787,822	10.08	6.74	3.34
SHA	18,014,403	2.74	0.74	2.00	18,413,569	4.98	1.14	3.84
STRINGSEARCH	7,048,747	10.88	4.58	6.31	7,409,552	13.40	6.94	6.46
Average		11.53	2.12	9.41		14.42	3.68	10.74

has the largest influence from the pipelined I-cache increasing, 2.13% of execution cycle. The negative impact of the pipelined I-cache is small or negligible for six other benchmarks while the impact of the pipelined D-cache shows a huge variation.

9.2.2 Impact of instruction distribution on performance

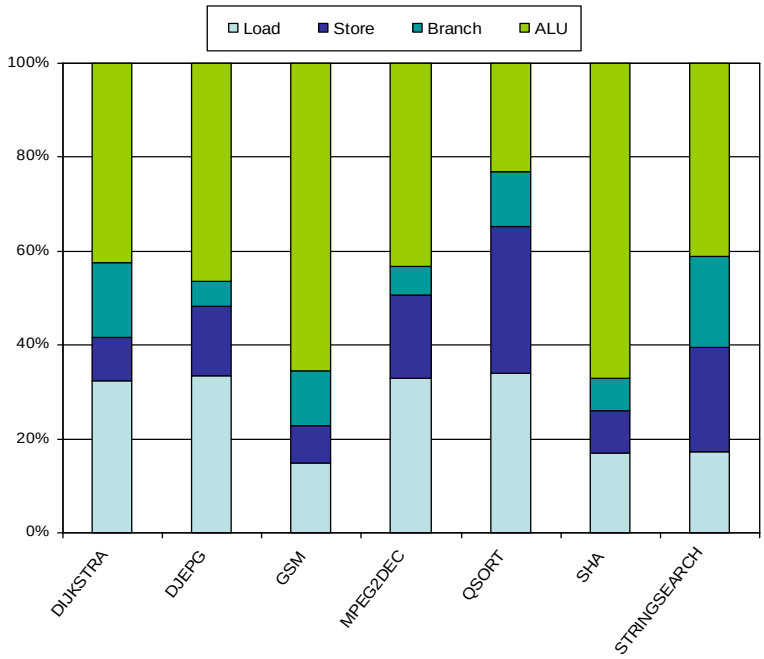


Figure 46: Dynamic instruction distribution

We analyze dynamic instruction distribution to explore pipelining overhead of each benchmark. The instructions are classified in four categories–branch, store, load and ALU instructions. The branch category represents flow control instructions. The store and load categories contain memory store and load instructions, respectively. The ALU category includes floating point and integer ALU operations. Figure 46 shows the dynamic instruction distribution of each benchmark.

Our architectural setup hides branch penalties using a BTB as explained in Section 8.2.3. However, a large number of branch instructions in a benchmark potentially incur higher chances of branch misprediction, which leads to increased branch penalties. We can observe this relationship by comparing Figure 46 and Table 20. For example, the DJEPG,

MPEG2DEC and SHA benchmarks have smaller numbers of branch instructions and thus show smaller increases in execution cycles due to the pipelined I-cache. Meanwhile, the GSM benchmark increases small number of cycles although having relative large amount of branch instructions. We observe a similar result with the DIJKSTRA benchmark, which has small penalty cycles due to a pipelined I-cache than the QSORT benchmark in spite of the large number of branch instructions. Therefore, we may say a BTB is well suited for the GSM and DIJKSTRA benchmarks.

Similar to the branch instructions, the number of load instructions affects the execution cycles increase due to the pipelined D-cache. The GSM and SHA benchmarks experience a small performance loss due to a pipelined D-cache thanks to the small number of load instructions in GSM and SHA. The QSORT benchmark has small performance overhead even though QSORT contains a large percentage of load instructions; DIJKSTRA and DJPEG benchmarks, on the other hand, show relatively large performance loss with a percentage of load instruction similar to QSORT. Compared to the number of branch instructions, the given pipelined cache architecture experiences relatively large performance loss. This result confirms that the static scheduling technique we use is not as effective as the dynamic scheduling we consider. However, we use static scheduling to hide load use delays because dynamic scheduling such as found in a superscalar architecture requires significant power budget and may not be adequate for embedded systems domains that we evaluate.

9.2.3 Cache power results

Now we analyze power consumption of the different pipelined cache architectures. We measure power consumption of the whole processor model: the entire microprocessor as well as each pipelined cache used by the microprocessor.

Figure 47 shows normalized cache power consumption of each benchmark with various pipelined cache stages. By lowering the supply voltage from 2.25V to 1.05V, the 2-stage pipelined cache achieves a 70% cache power reduction on average. The 3-stage and 4-stage

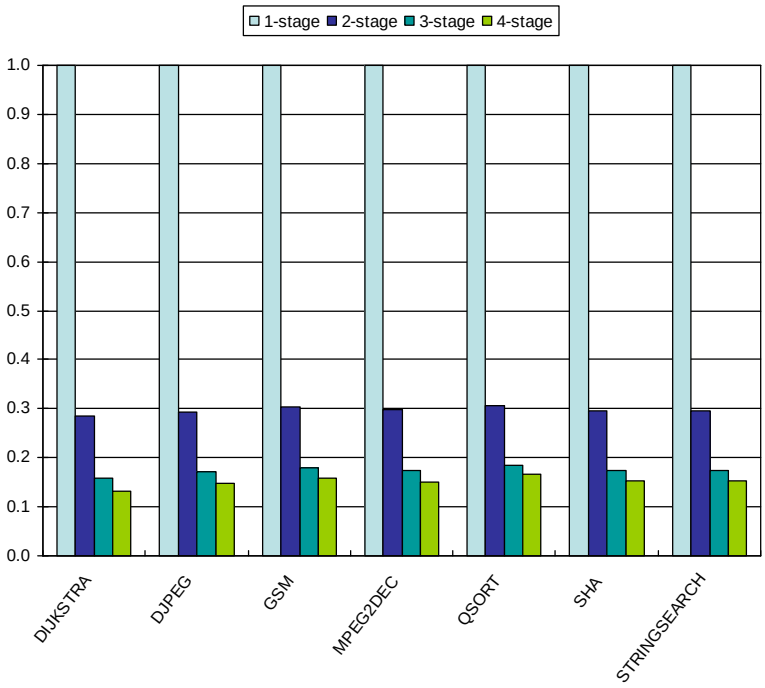
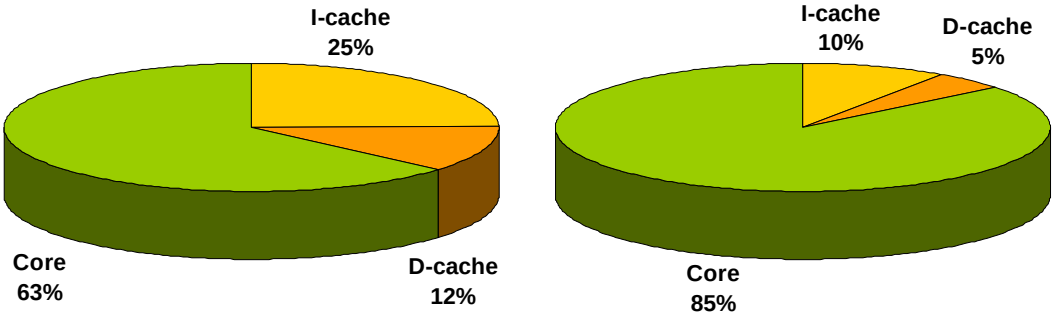


Figure 47: Normalized cache power consumption according to the cache pipeline stage

pipelined caches save 82% and 85% in average cache power, respectively. Power savings increments achieved using 3- and 4-stage pipelined caches are relatively small compared with the initial power savings of the 2-stage pipelined cache because we can lower only an incrementally small additional amount from the supply voltage: 0.3V and 0.05V for 3- and 4-stage pipelined caches, respectively. The power savings from the pipelined cache are limited by the supply voltage reduction.



(a) Non-pipelined cache processor (b) 2-stage pipelined cache processor

Figure 48: Processor power distribution

The reduction in total power consumption of the pipelined cache processor depends

on the contribution of the cache power consumption out of the total processor power consumption. The Simplescalar/ARM+Wattch processor model shows that 37% of the total processor power is consumed by the non-pipelined caches as shown in Figure 48(a). This proportion is reasonable if we consider that cache power of a StrongARM 110 processor comprises 43% of the total chip power [42]. Figure 48(b) is the power breakdown of the 2-stage pipelined cache processor. Thanks to the 70% cache power reduction, the 2-stage pipelined caches comprises only 15% total processor power consumption.

9.2.4 Processor power results

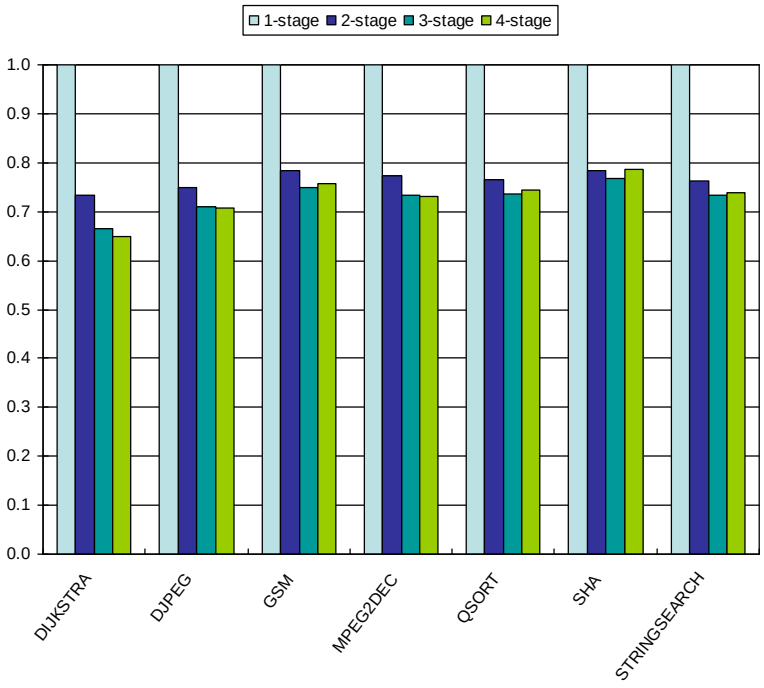


Figure 49: Normalized processor power consumption according to the cache pipeline stage

Figure 49 show normalized total processor power consumption. The 2-, 3- and 4-stage pipelined cache processors achieve 23.55%, 27.21% and 26.21% average power savings, respectively. Although the 4-stage pipelined cache saves 3% more cache power consumption compared to the 3-stage pipelined cache, the 4-stage pipelined cache processors consumes more power than 3-stage pipelined cache processor because pipelining overhead is larger than power reduction. Therefore, a 4-stage pipelined cache is not as power efficient

as a 3-stage nor as performance efficient as 3-stage in the given architecture we consider.

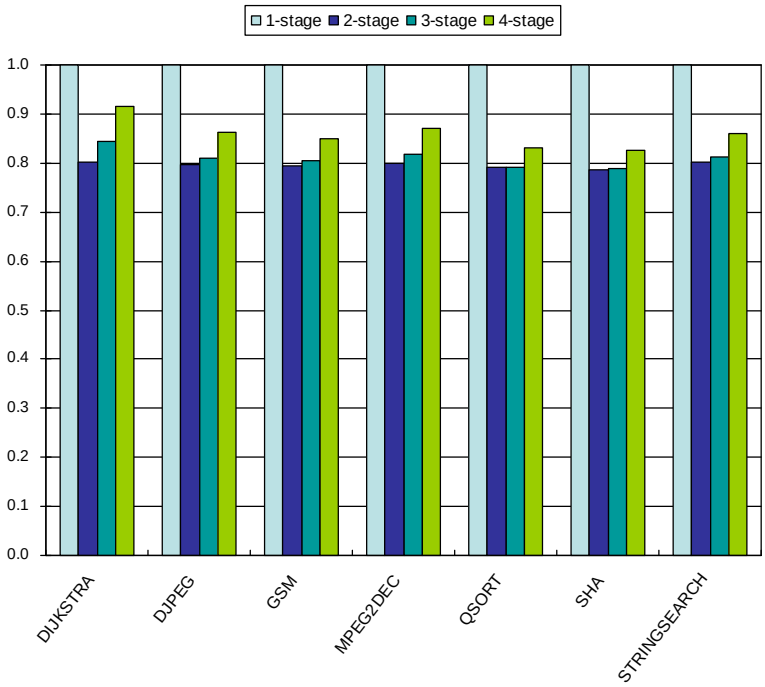


Figure 50: Normalized Processor energy consumption according to the cache pipeline stage

The power savings of the benchmarks do not represent the full effect of the pipelined cache correctly because the performance loss not accounted for in the power savings numbers. For example, the DIJKSTRA benchmark gains the largest power savings yet also experiences the largest increase in execution cycles. Therefore, energy savings are more important than power savings for understanding the results. Figure 50 shows the normalized energy consumption of each benchmark with different cache stages. The 2-stage pipelined cache processor achieves 20.43% average energy saving. However, the 3-stage pipelined cache saves only 19.03%. In short, we find that each extra pipeline stage (beyond two) in the caches results in reduced energy savings when compared to the 2-stage case. The 2-stage pipelined cache is the most energy effecient.

Although our proposed LPPC achieves 20.43% average energy savings with 4.14% average execution cycle increase, since some embedded systems must maintain exact execution time constraints, we investigate eliminating the performance loss due to execlusively

Benchmark	Non-pipelined cache architecture				
	Cycle time (nS)	Vdd (datapath)	Vdd (cache)	Power (W)	
DIJKSTRA	4.29	2.25	2.25	3.573	
DJPEG	4.29	2.25	2.25	3.709	
GSM	4.29	2.25	2.25	3.669	
MPEG2DEC	4.29	2.25	2.25	3.571	
QSORT	4.29	2.25	2.25	3.255	
SHA	4.29	2.25	2.25	3.833	
STRINGSEARCH	4.29	2.25	2.25	3.426	
Average				3.577	
Benchmark	**2-stage low power pipelined cache architecture**				
	Cycle time (nS)	**Vdd (datapath)**	**Vdd (cache)**	**Power (W)**	**% Total power savings**
DIJKSTRA	3.92	2.49	1.15	3.185	10.87
DJPEG	4.05	2.40	1.15	3.190	14.00
GSM	4.23	2.29	1.15	3.034	17.30
MPEG2DEC	4.16	2.33	1.15	3.012	15.67
QSORT	4.15	2.33	1.15	2.727	16.23
SHA	4.28	2.26	1.15	3.110	18.87
STRINGSEARCH	4.09	2.37	1.15	2.937	14.27
Average				3.028	15.32

Figure 51: Pipelined cache adjusted power consumption

scaling voltage. To maintain execution time, we reduce the cycle time (and thus increase the supply voltage over our current results). When we scale up required supply voltages for the non-cache circuits using Equation 14, in which α is set to 1.5 [43] , we choose adjusted V_{dd} for caches using Figure 45 and calculate power consumption using Equation 13 as well. Table 51 shows (i) the cycle time needed to maintain the throughput, (ii) required V_{dd} to speed up non-cache and (iii) requiring V_{dd} to speed up the cache and (iv) resulting power consumption. As a result, the adjusted 2-stage LPPC in Table 51 achieves the same execution time as our initial 1-stage cache architecture. Although the increased supply voltage increases power consumption (i.e., 5.11%), LPPC achieves 15.32% of average power saving without performance loss.

9.3 Summary

In this chapter, first we explained our experimental setup to evaluate LPPC. We use Simplescalar/ARM for the performance estimation of a processor. We added Wattch to Simplescalar/ARM to estimate power consumption of Simplescalar/ARM. We also modified

Simplescalar/ARM+Wattch so we can evaluate architectures each with a different number of cache stages. To model the power consumption of buffers between broken pipeline stages, we use the MARS Verilog processor model. We increase the number of pipeline stage in MARS; then, using a synthesis based power estimation, we model the buffer power and feed the power model into Simplescalar/ARM+Wattch.

Accurate cache delay modeling is very important because we decide the supply voltage of the cache based on cache delay. We modify CACTI to obtain cache delay while changing cache supply voltage. We considered conventional pipelined caches as well as wave-pipelined caches and find that the wave-pipelined cache performs best; thus, we consider only wave-pipelined caches for our experiments.

We evaluate LPPC by comparing to a base non-pipelined cache architecture. We consider 2-, 3- and 4-stage pipelined caches. While using a processor configuration targeting an embedded processor, we find that 2-stage LPPC shows the best results in terms of power. The 2-stage LPPC achieves 23.55% power reduction and 20.43% energy reduction.

In the next chapter, we will explain a sleepy stack pipelined cache, which combines the sleepy stack SRAM and LPPC.

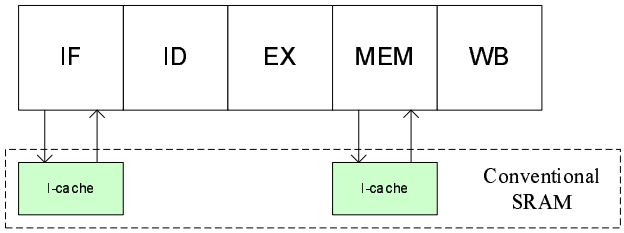
CHAPTER X

SLEEPY STACK PIPELINED CACHE

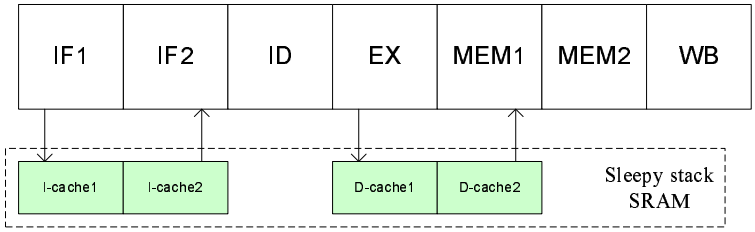
In this chapter, we combine our two low-power techniques, which are the sleepy stack SRAM and Low Power Pipelined Cache (LPPC) explained in Chapters 5 and 8. The sleepy stack technique achieves ultra-low leakage power consumption. However, increase of delay is a bottleneck to use the sleepy stack technique in a system that should maintain performance. To overcome this bottleneck, we apply LPPC to the sleepy stack SRAM, and thus we achieve low-leakage power consumption while maintaining performance.

10.1 Approach

In this section, we explain our design approach to combine the sleepy stack SRAM and LPPC.



(a) A pipeline with non-pipelined caches



(b) A pipeline with sleepy stack pipelined caches

Figure 52: Non-pipelined and sleepy stack pipelined cache architectures

Figure 52 depicts our sleepy stack pipelined cache compared to a conventional non-pipelined cache. The base case in Figure 52(a) has two non-pipelined caches, which are leaky but fast conventional SRAM. Meanwhile, our approach in Figure 52(b) applies sleepy stack pipelined SRAM to the processor pipeline. The sleepy stack SRAM explained in Section 6.2 increases delay compared to the conventional cache, and thus the cache may not be accessed in one cycle unlike the conventional cache. To overcome this problem, we pipeline the sleepy stack SRAM. This approach is similar to LPPC in Chapter 8. However, at this time, we apply the sleepy stack technique instead of lowering supply voltage. Therefore, the processor pipeline is expanded along the pipelined cache. The decreased cache access delay can allow the pipeline in Figure 52(b) to achieve the same cycle time as Figure 52(a) even though the delay of a cache in Figure 52(b) is larger than the delay of a cache in Figure 52(a). In Chapter 9, a wide variety of processor/cache pipeline structures (number of stages, etc.) are explored, and Figure 52 seems to provide a high-impact tradeoff; thus, we focus exclusively on Figure 52's tradeoff in this chapter.

We discussed issues related to pipelining penalties and pipeline implementation in Chapter 8. Similar to LPPC in Chapter 8, we use a Branch Target Buffer (BTB) to tackle branch delay and a compiler technique to tackle load delay. We also use wave-pipelining to implement cache pipelining. However, at the circuit level, at which we are to implement a sleepy stack pipelined cache, the wave pipelining design should satisfy two timing constraints [10]. First, the number of wave pipelining stages N needs to satisfy:

$$\frac{T_{max}}{T_{clk}} < N < \frac{T_{min}}{T_{clk}} + 1, \tag{24}$$

where T_{max} is the maximum delay, T_{min} is the minimum delay, and T_{clk} is the cycle time. Second, another constraint is that next earliest wave should not arrive at a gate input until the latest wave has propagated through. We will address these two issues when we implement the sleepy stack pipelined cache.

In the next section, we explain the detailed design methodology of the sleepy stack pipelined cache described in this section.

10.2 Design methodology

In this section, we explain the design methodology of our proposed low-power pipelined cache using sleepy stack SRAM. We first design our SRAM, then estimate delay and power consumption. The estimated values are fed into the pipelined processor simulation model to estimate power consumption using benchmark programs.

10.2.1 Sleepy stack SRAM

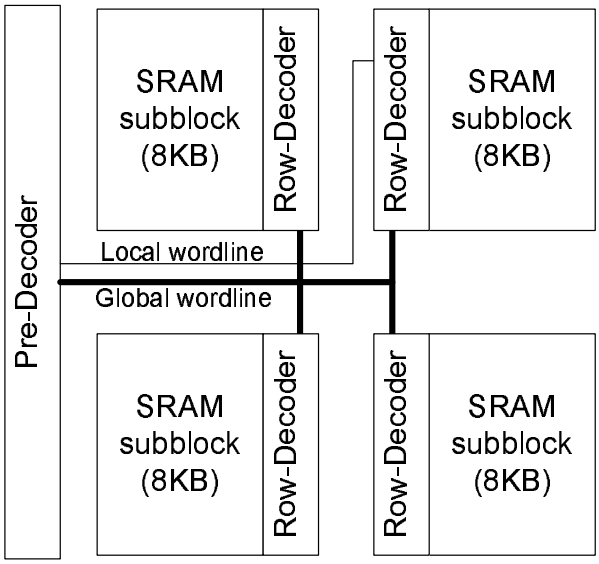


Figure 53: SRAM structure

We design 32KB SRAM with four 8KB subblocks as shown in Figure 53. The SRAM mainly consists of the decoder and the SRAM core. The decoder enables one of the SRAM rows according the address input. The decoder further consists of the pre-decoder and the row-decoder. Each subblock is selected using a global `wordline`, and a row of a subblock is selected using a local `wordline` from the pre-decoder (please note that there are more than one local `wordline` wires in Figure 53 even though only one wire is shown for simplicity). The row-decoder enables one row of the selected subblock by combining the global `wordline` and local `wordline` [3]. The line size of our subblock is 32B, and each subblock has 256 rows. A column of a subblock consists of precharge logic, SRAM cells,

and column select logic. Eight columns share a sense amplifier, write enable logic, and data input logic. The column select logic selects one column out of eight columns. This column select method is generally used in SRAM design to save area and power consumption of the sense amplifier and the write enable logic. Instead of starting from scratch, we use the CACTI model for the SRAM structure and transistor sizing [57], and scale down to the target process, i.e., 0.07μ.

We first design a base case SRAM that uses conventional CMOS techniques. Then we design sleepy stack SRAM by applying sleepy stack to the conventional SRAM. In the decoder, we do not apply sleepy stack to the global `wordline` decoder driver because degradation of its driving power degrades performance significantly. Also, in the SRAM cell array, we apply the sleepy stack only to the SRAM cell, i.e., the precharge logic, column select logic, and write enable logic are intact. Although the intact components also consume considerable amounts of leakage power, changing all components may reduce performance significantly.

Also, we consider low-V_{dd} LPPC for the purposes of comparison. In nanoscale technology, low-V_{dd} LPPC can save dynamic power as well as static power since low-V_{dd} degrades the Drain Induced Barrier Lowering (DIBL) effect. We use a supply voltage value of $0.7V$ for a low-V_{dd} LPPC.

To evaluate the SRAM design, we mainly use a simulation based methodology utilizing HSPICE. We design the SRAM (targeting 0.07μ technology) using gate-level netlists in HSPICE with which we can use the 0.07μ Berkeley Predictive Technology Model [7]. The SRAM models are simulated to measure active power, static power and SRAM access time. We use $1.0V$ as supply voltage of the base case. The active power is measured while accessing SRAM cells. We also derive per-access energy consumption by dividing measured power consumption by maximum clock frequency. The static power is measured while stopping all input transitions. During static power measurement, we turn off sleep transistors of the sleepy stack SRAM (i.e., we assume we are in sleep mode). To avoid

leakage power measurement biased by a majority of '1' versus '0' (or vice-versa) values, half of the cells are randomly set to '0,' with the remaining half of the cells set to '1.' SRAM propagation delay is also measured along the critical path from the input address to the sense amplifier output driver.

10.2.2 Pipelined cache model

We use modified Simplescalar/ARM [61], which we use in Chapter 9, for the performance evaluation of our target processor. Therefore, the modified Simplescalar/ARM processor has increased branch and load delays (i.e., proportional to the pipeline depth of the I-cache and D-cache, respectively). However, these penalties are reduced by a Branch Target Buffer (BTB) and compiler optimization explained in Section 8.2.3. The Simplescalar/ARM configuration is modeled after the Intel StrongARM as well as Intel XScale microarchitectures similar to the Low-Power Pipelined Cache (LPPC) experiment in Chapter 8. However, at this time we use direct mapped caches instead of 4-way caches to achieve fast cache access time. The memory system consists of two levels. The first level is L1 caches (one instruction cache and one data cache), and the second level is off-chip main memory. We assume that benchmark programs fit into the main memory. We also use the seven benchmark programs used in Table 19 in Section 9.1.3. The detailed configuration is shown in Table 21.

We assume that the critical path of the base case pipeline in Figure 52 (a) lies in the cache access stages, i.e., IF and MEM, and other stages are optimized based on the feasible cycle time of IF and MEM stages. This assumption is reasonable because caches are often a bottleneck to reduce cycle time, and a pipelined cache is typically introduced to overcome this bottleneck. A clock speed of 833 MHz in 0.07μ technology is used since it is the fastest clock speed that can be achieved from the SRAM that we design. Furthermore, we use the before and after cache cases shown in Figure 52. We measure execution cycle time using the non-pipelined cache processor and the sleepy stack pipelined cache processor.

Table 21: Simplescalar configurations for sleepy stack SRAM

Execution type	In-order
Fetch queue	2
Branch predictor	128 entry BTB, 8K bimodal
Block Fetch & Decode width	1
Functional Units	1 integer ALU, 1 FP ALU 1 FP mult
L1 I-cache	32KB, 1-way
L1 D-cache	32KB, 1-way
L2 cache	None
Memory bus width	4
Memory latency	12
Clock speed	833MHz
V_{dd} (Cache)	1.0V, 0.7V

We feed measured per-access energy of the SRAM into Simplescalar/ARM, then SRAM power consumption is estimated by combining the per-access energy and the cache access ratio.

We design the sleepy stack pipelined SRAM, the base case SRAM, and the low-V_{dd} LPPC described in this section. The three SRAM designs are simulated and compared in terms of power and performance in the next section.

10.3 Results

In this section, we compare the three SRAM designs explained in Section 10.2. We first study power consumption of the three designs. Then we explore the performance results at the architectural level.

10.3.1 SRAM power consumption

Table 22 shows HSPICE simulation results of SRAM in 0.07μ technology using BPTM [7] (note that Figures 54(a) and 54(b) show the same results with graphs). We measure active power, static power and delay. The sleepy stack SRAM achieves 17X static power reduction compared to the base case SRAM. Although the sleepy stack technique shows more than

Table 22: SRAM power consumption

		Active power (W)	Static power (W)
Basecase	Decoder	4.78E-03	4.15E-04
	SRAM	5.28E-02	4.78E-03
	Total	5.76E-02	5.20E-03
Low-Vdd pipelined cache	Decoder	2.07E-03	1.18E-04
	SRAM	2.17E-02	1.66E-03
	Total	2.38E-02	1.78E-03
Sleepy stack pipelined cache	Decoder	4.35E-03	5.84E-05
	SRAM	7.38E-02	2.48E-04
	Total	7.82E-02	3.07E-04

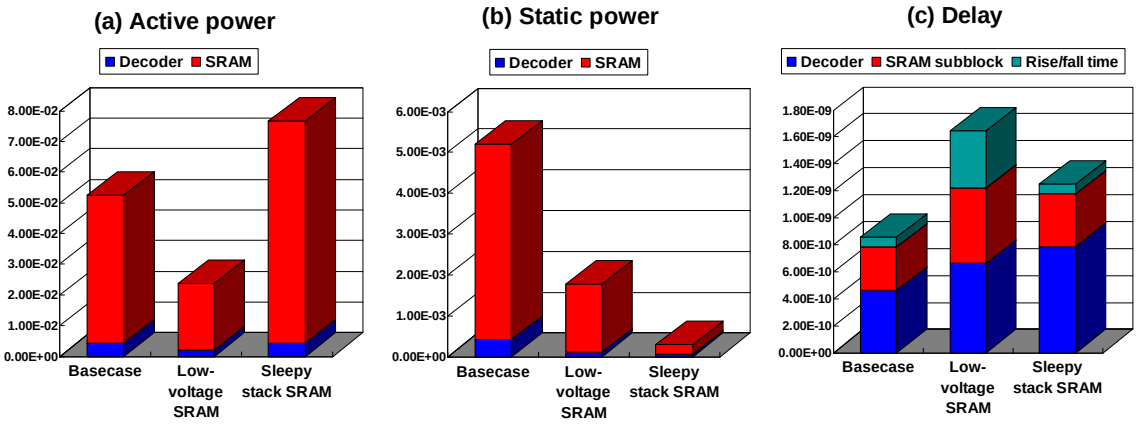


Figure 54: SRAM performance comparison

two orders of magnitude leakage power reduction for generic logic circuits and SRAM cells in Chapter 7, the overall SRAM leakage reduction is smaller due to the circuits (i.e., global `wordline` driver, sense amplifiers, and write enable logic) using conventional CMOS in both cases (before and after); only the SRAM cells are changed in SRAM subblocks and every gate except the global `wordline` driver in the decoder. Meanwhile, the sleepy stack SRAM shows a 36% active power increase. Delayed switching of the sleepy stack SRAM cell increases the sense amplifier and write enable circuitry unstable time, which increases short-circuit power during switching. Low-V_{dd} LPPC reduces active power by 59% and static power by 3X. The active power reduction of low-V_{dd} LPPC is an advantage over the sleepy stack SRAM. However, compared to the sleepy stack SRAM, leakage power

reduction of low-V_{dd} LPPC is small.

Table 23: SRAM delay

	Delay (ns)				
	Decoder	SRAM subblock	Rise/fall time	SRAM external	Total
Basecase	4.61E-10	3.24E-10	7.10E-11	3.44E-10	1.20E-09
Low-Vdd pipelined cache	6.61E-10	5.61E-10	4.25E-10	3.44E-10	1.99E-09
Sleepy stack pipelined cache (longest path)	7.89E-10	3.90E-10	7.17E-11	3.44E-10	1.59E-09
Sleepy stack pipelined cache (shortest path)	7.86E-10	3.61E-10	9.50E-11	3.44E-10	1.59E-09

Table 23 shows SRAM delay (note that Figure 54(c) shows the same results with graphs). Decoder delay is from the address input to the `wordline` driver output. SRAM subblock delay is from `wordline` select to the sense amplifier output driver. Rise/fall time is switching delay of the sense amplifier driver. Therefore, SRAM internal delay consists of decoder delay, SRAM subblock delay, and rise/fall time. We assume that the required cycle time ($1.2ns$) includes SRAM internal delay as well as SRAM external delay, e.g., address calculation circuitry delay in IF and MEM stages. The SRAM internal delay of the base case is $0.856ns$, and thus SRAM external delay is $1.20ns - 0.856ns = 0.344ns$; note that in our comparison, the "SRAM external delay" is common for both the base case and the sleepy stack SRAM. Based on this assumption, we now calculate the longest path delay of the sleepy stack SRAM. The longest path delay of the sleepy stack SRAM passes though the global `wordline` decoder, and thus the longest path delay of the sleepy stack SRAM is $1.59ns$ as shown in Table 23. Meanwhile the shortest path delay passing through local `wordline` is $1.59ns$. From Equation 24, we can find that possible number of wave pipelining stages is two. Although the sleepy stack SRAM increases internal delay by 46%,

we can potentially achieve $1.2n$ cycle time by pipelining the sleepy stack SRAM. The increased delay of low-V_{dd} LPPC, which is 92%, is also hidden using pipelining. We assume the SRAM design satisfies the second wave-pipelining constraint in Section 10.1 because SRAM inputs are balanced and easy to control [10].

10.3.2 Pipelined cache performance

Table 24: Pipelined cache active power per benchmark

Technique	Benchmark	Execution Cycles	Active cache power (W)		
			ICache	DCache	Total
Base case	CJPEG	45,689,751	3.73E-02	1.36E-02	5.10E-02
	DIJKSTRA	101,478,228	3.58E-02	1.47E-02	5.05E-02
	DJPEG	11,060,173	3.48E-02	1.67E-02	5.14E-02
	GSM_ENC	21,551,339	4.09E-02	9.25E-03	5.02E-02
	MPEG2DEC1	29,280,102	3.18E-02	1.58E-02	4.77E-02
	QSORT	91,810,948	2.76E-02	1.47E-02	4.23E-02
	SHA	17,533,451	4.31E-02	1.11E-02	5.42E-02
	STRINGSEARCH	6,983,598	3.08E-02	1.16E-02	4.24E-02
Average			3.53E-02	1.34E-02	4.87E-02
Low-voltage pipelined cache	CJPEG	47,300,053	1.49E-02	5.45E-03	2.04E-02
	DIJKSTRA	110,896,835	1.36E-02	5.55E-03	1.91E-02
	DJPEG	11,703,670	1.36E-02	6.51E-03	2.01E-02
	GSM_ENC	21,888,764	1.67E-02	3.77E-03	2.04E-02
	QSORT	30,212,689	1.28E-02	6.35E-03	1.91E-02
	MPEG2DEC1	94,873,589	1.10E-02	5.87E-03	1.69E-02
	SHA	17,600,173	1.78E-02	4.57E-03	2.23E-02
	STRINGSEARCH	7,287,217	1.22E-02	4.61E-03	1.68E-02
Average			1.41E-02	5.33E-03	1.94E-02
Sleepy stack pipelined	CJPEG	47,300,053	4.90E-02	1.79E-02	6.69E-02
	DIJKSTRA	110,896,835	4.45E-02	1.82E-02	6.27E-02
	DJPEG	11,703,670	4.46E-02	2.14E-02	6.60E-02
	GSM_ENC	21,888,764	5.47E-02	1.24E-02	6.71E-02
	QSORT	30,212,689	4.19E-02	2.08E-02	6.27E-02
	MPEG2DEC1	94,873,589	3.62E-02	1.93E-02	5.54E-02
	SHA	17,600,173	5.83E-02	1.50E-02	7.33E-02
	STRINGSEARCH	7,287,217	4.00E-02	1.51E-02	5.51E-02
Average			4.61E-02	1.75E-02	6.37E-02

We measure execution cycles and active power consumption of the conventional non-pipelined cache and the sleepy stack two-stage pipelined cache using our simulation environment. From the results of the seven different benchmarks, the result shows that the

pipelined cache increases cycle time by 4% on average compared to the non-pipelined cache; and the average I-cache and D-cache miss rates are 0.18% and 0.98%, respectively. Although the cycle time of the sleepy stack is increased, this is remarkable improvement if we consider the delay overhead of the sleepy stack SRAM (46%).

Unfortunately, the sleepy stack SRAM increases active power by an average of 31%, and this is consistent with the result in Section 10.3.1. Since the active power increase is non-negligible, we need to consider the tradeoffs between active power overhead and static power savings. The active power overhead is 15.0mW, and reduced leakage power saving from the two sleepy stack SRAMs (I-cache and D-cache) is 9.78mW. Therefore, if the sleep mode is 3 times larger than active mode, overall energy saving of the sleepy stack SRAM is larger than the conventional SRAM with a small execution cycle overhead. Compared to low-V_{dd} LPPC, the active power overhead of the sleepy stack pipelined cache is 44.3mW, and leakage power saving is 2.94mW. Therefore, if the sleep mode is 3 times larger than active mode, overall energy saving of the sleepy stack SRAM is larger than the conventional SRAM with a small execution cycle overhead. If we recall the cell phone calling time scenario in Chapter 2, our sleepy stack SRAM has an edge over the base case as well as low-V_{dd} LPPC.

Due to the additional transistors and complex structure, the sleepy stack SRAM incurs some area overhead as explained in Chapter 7. However, since our sleepy stack SRAM has some parts that use conventional style CMOS design, e.g., a decoder global `wordline` driver, precharge logic, sense amplifier, write enable logic, we estimate that overall area overhead is less than 100% (i.e., the sleepy stack SRAM is less than 2X the original area). (Please note that we assume the area overhead is paid for in term of increased chip cost; i.e., we do not halve the L1 cache sizes in this sleepy stack case.)

Clearly, the specific quantitative results presented here will vary as more accurate 0.07μ models become available and more accurate processor modeling/design is done. Nevertheless, we expect the high level trend/tradeoff identified in this paper to hold true in general:

sleepy stack pipelined caches provide much lower standby (leakage) power consumption at a cost of increased active power consumption and area.

10.4 Summary

In this Chapter we introduce a new low-leakage technique by combining two low power techniques we have pioneered: sleepy stack and LPPC. The sleepy stack pipelined SRAM can achieve low-leakage power consumption while almost maintaining the architectural performance with a potentially small 4% cycle time increase. The leakage power reduction is more than 17X compared to the conventional non-pipelined cache SRAM design and more than 5.6X Compared to low-V_{dd} LPPC. Although our combined technique incurs 31% dynamic power increase, our technique can be well applied a system that has short operation time (long sleep time) on average.

CHAPTER XI

CONCLUSION

In nanometer scale CMOS technology, subthreshold leakage power is compatible to dynamic power consumption, and thus handling leakage power is a great challenge. In this dissertation, we present a new circuit structure named "sleepy stack" to tackle the leakage problem. The sleepy stack has a combined structure of two well-known low-leakage techniques, which are the forced stack and sleep transistor techniques. However, unlike the forced stack technique, the sleepy stack technique can utilize high-V_{th} transistors without incurring large delay overhead. Also, unlike the sleep transistor technique, the sleepy stack technique can retain exact logic state while achieving similar leakage power savings. In short, our sleepy stack structure achieves ultra-low leakage power consumption while retaining state.

Since the sleepy stack technique can retain logic state, we can use the sleepy stack technique for both generic logic circuits as well as memory, i.e., SRAM. When applied to generic logic circuits, the sleepy stack technique achieves up to 200X leakage reduction compared the forced stack technique with -6%$\sim$7% delay variations and 51%$\sim$118% area overhead. When applied to SRAM, the sleepy stack SRAM cell with 1.5xV_{th} achieves 5X leakage reduction with 32% delay overhead compared to the best prior approach, a high-V_{th} SRAM cell. Alternatively, the sleepy stack SRAM cell achieves 2.49X leakage reduction with the same delay as the high-V_{th} SRAM cell. As such, the sleepy stack SRAM cell provides new Pareto points which were not known before.

We also propose a new low power architectural technique called Low-Power Pipelined Cache (LPPC). Our LPPC provides a new way to save power consumption of a cache with small performance overhead. By pipelining a cache into multiple stages, we provide

extra slack. Although a conventional pipelined cache uses this extra slack to reduce cache access time, alternatively, we use this slack to lower cache supply voltage and thus achieve cache power savings. Using a specific processor performance and power evaluation setup for LPPC, we observed that 2-stage pipelined cache achieves the best results among two, three, and four stage pipelined caches. We achieve 70% of cache power savings (20.43% of processor energy saving) with 4.14% of average execution cycle increase. Although evaluated with an embedded processor model, LPPC can be applicable to generic pipeline processors with caches.

Finally, we combine the two proposed low power cache techniques, sleepy stack SRAM and LPPC. Although sleepy stack SRAM achieves ultra-low leakage power consumption, increase of delay could be a bottleneck for a system that does not allow any delay increase. For our target system sleepy stack pipelined SRAM achieves 17X leakage power reduction while increasing execution cycle by 4% on average. Although this combined technique increases 33% active power consumption, this technique is well suited for products whose usage results in most of the time spent in sleep mode.

In conclusion, we have explored a high-impact and heavily researched area: low-power VLSI design. Two major discoveries - sleepy stack and LPPC - have been shown to have significant impact. For systems spending a large percentage of time in sleep mode yet requiring ultra-fast wakeup, sleepy stack provides the best solution currently known in VLSI design, typically resulting in approximately two orders of magnitude less leakage power over the best of all prior known state-saving VLSI design approaches.

APPENDIX A

CHAIN OF FOUR INVERTERS LAYOUT

In Appendices A, B, C, and D, we present layouts we design in this thesis. The layouts include logic circuits explained in Section 6.1 and SRAM cells explained in Section 6.2. We design target benchmark circuits using Cadence Virtuoso, a custom layout tool [11], and the North Carolina State University (NCSU) Cadence design kit targeting TSMC 0.18μ technology [48].

First, we present logic circuit layouts using five different techniques, i.e., base case, sleep, zigzag, forced stack, and sleepy stack. Second, we present SRAM cell design using three different techniques, i.e., base case, forced stack and sleepy stack.

Now we present chain of 4-inverters layouts in the rest of this appendix (Appendix A.

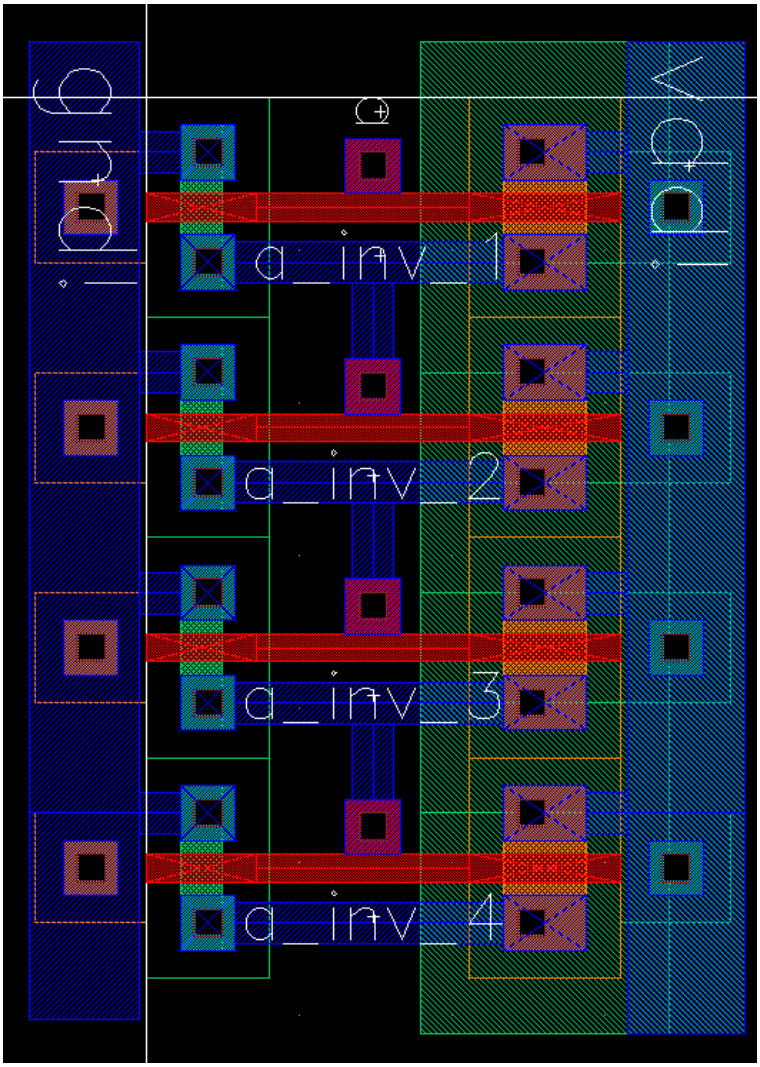


Figure 55: Base case 4 inverters

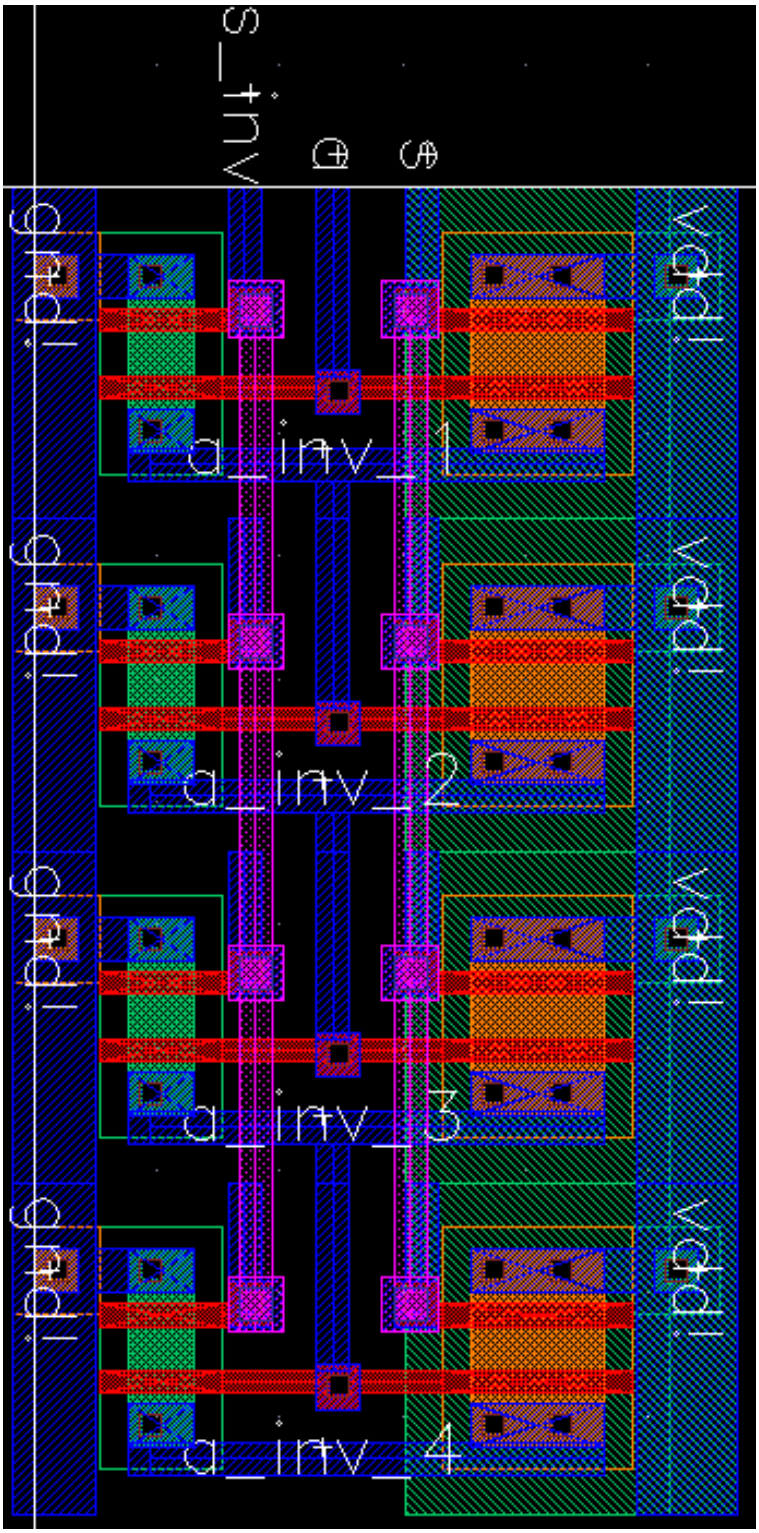


Figure 56: Sleep approach 4 inverters

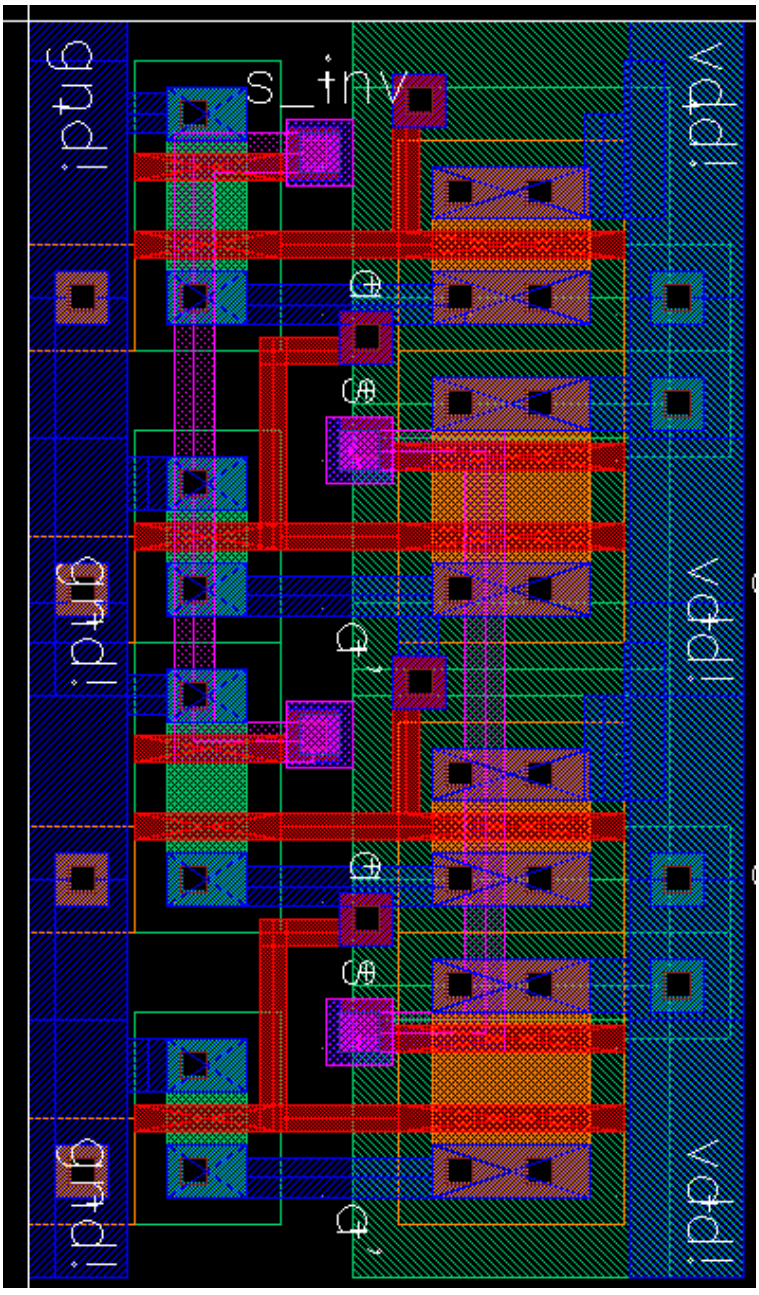


Figure 57: Zigzag approach 4 inverters

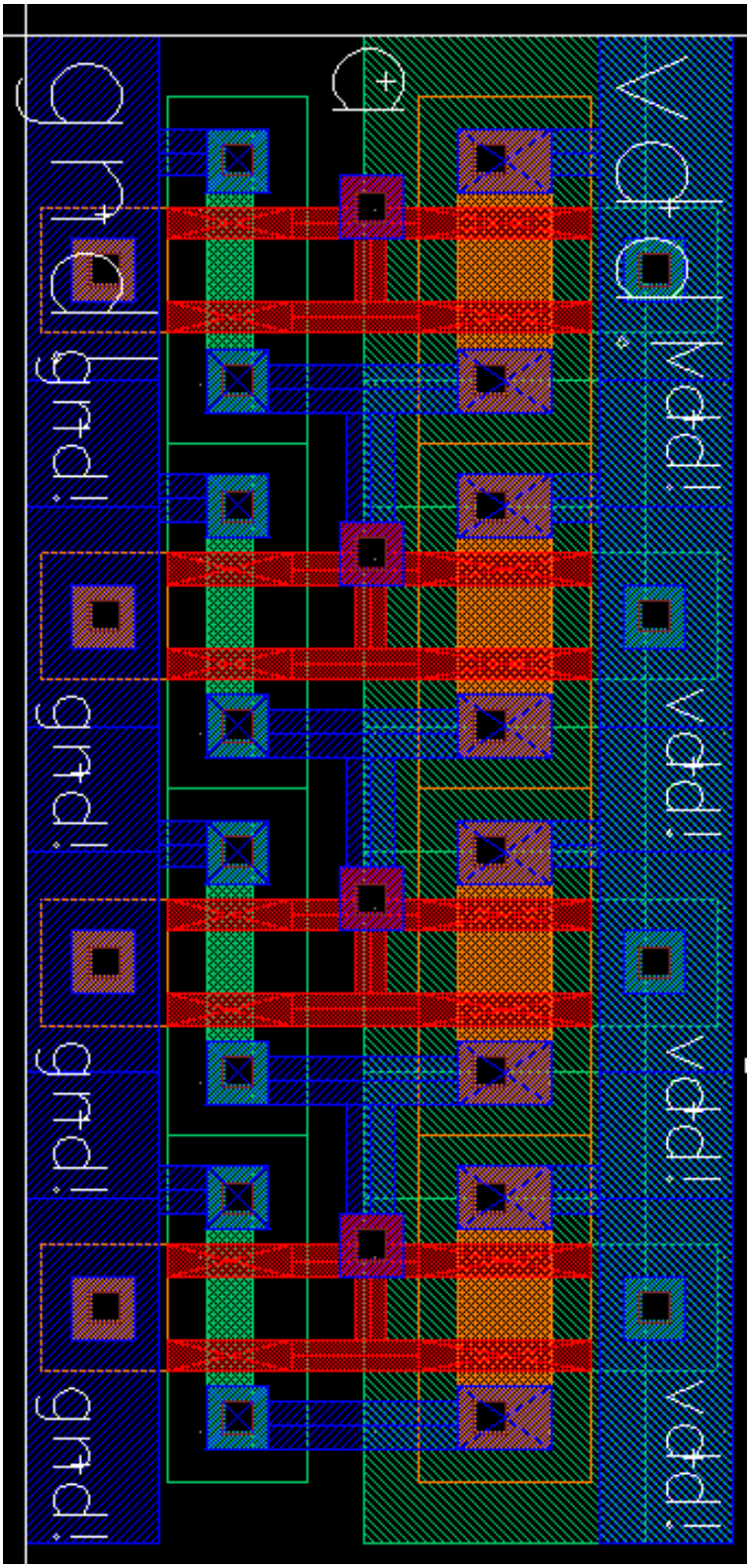


Figure 58: Forced stack approach 4 inverters

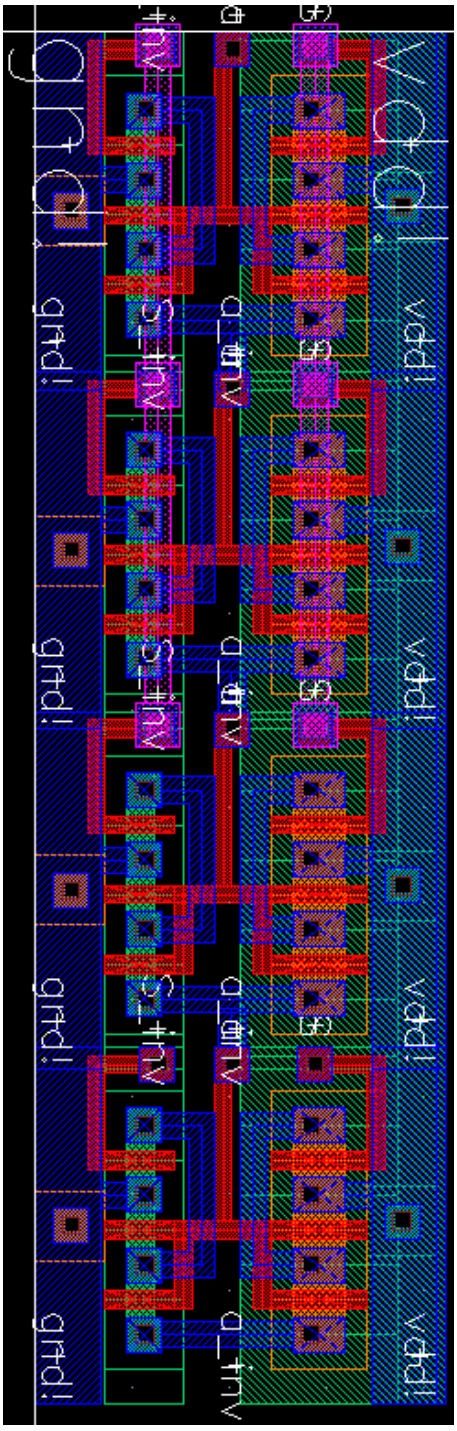


Figure 59: Sleepy stack approach 4 inverters

APPENDIX B

FULL ADDER LAYOUT

In this appendix, we present 1-bit full adder layouts using five different techniques, i.e., base case, sleep, zigzag, forced stack and sleepy stack.

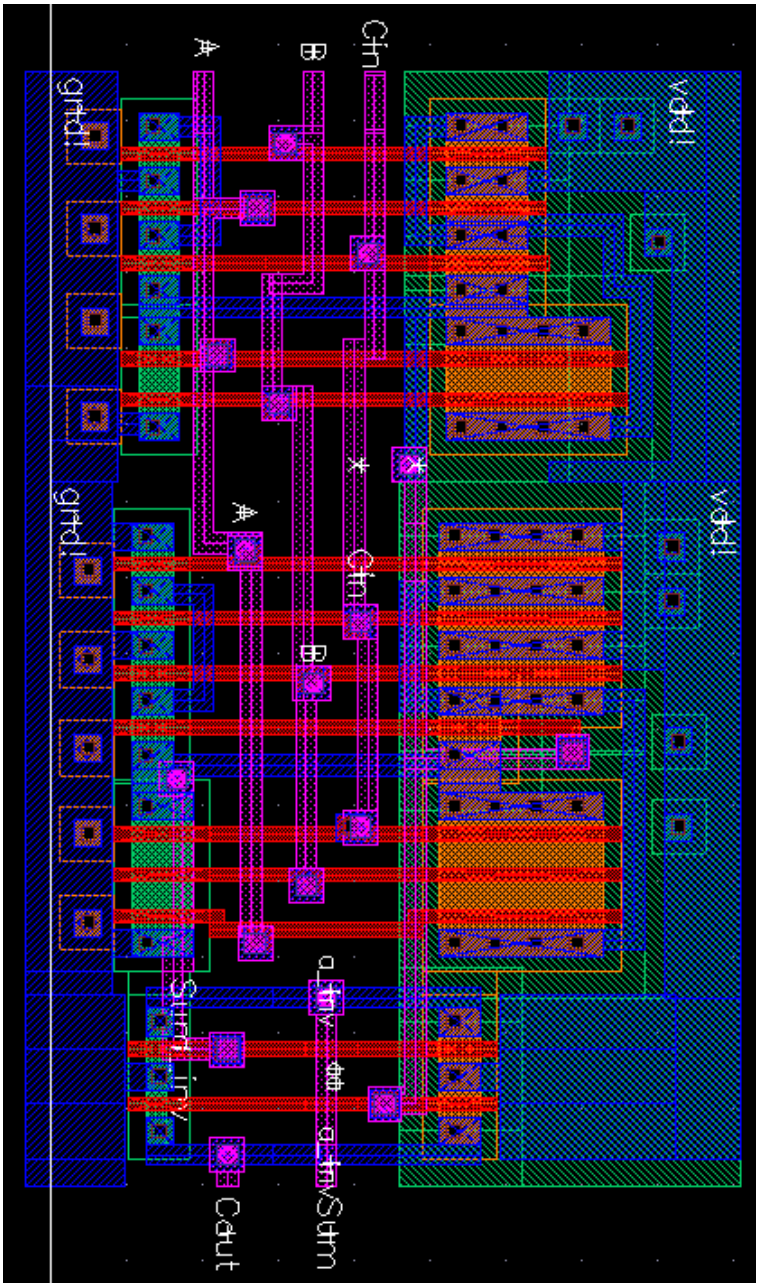


Figure 60: Base case full adder

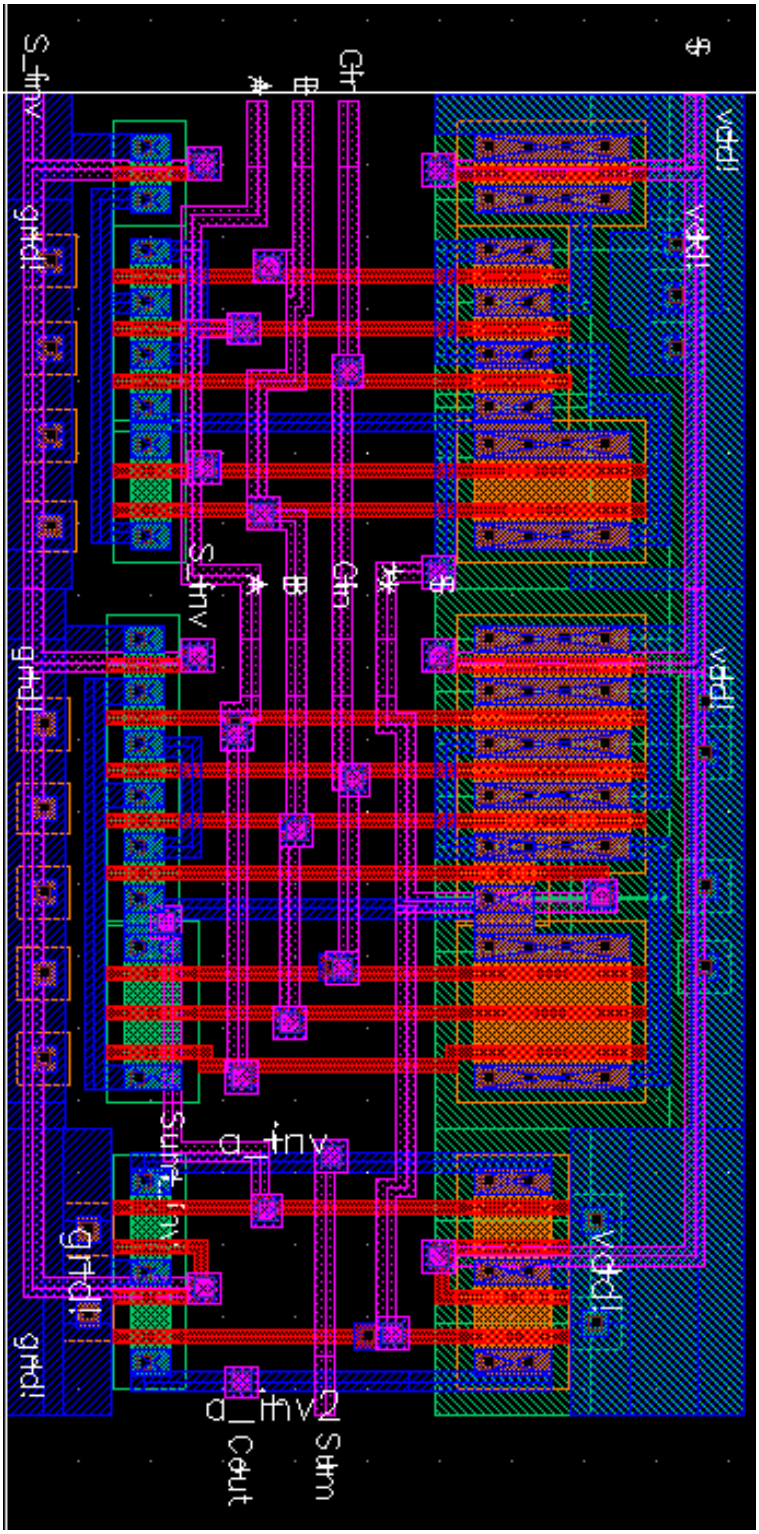


Figure 61: Sleep approach full adder

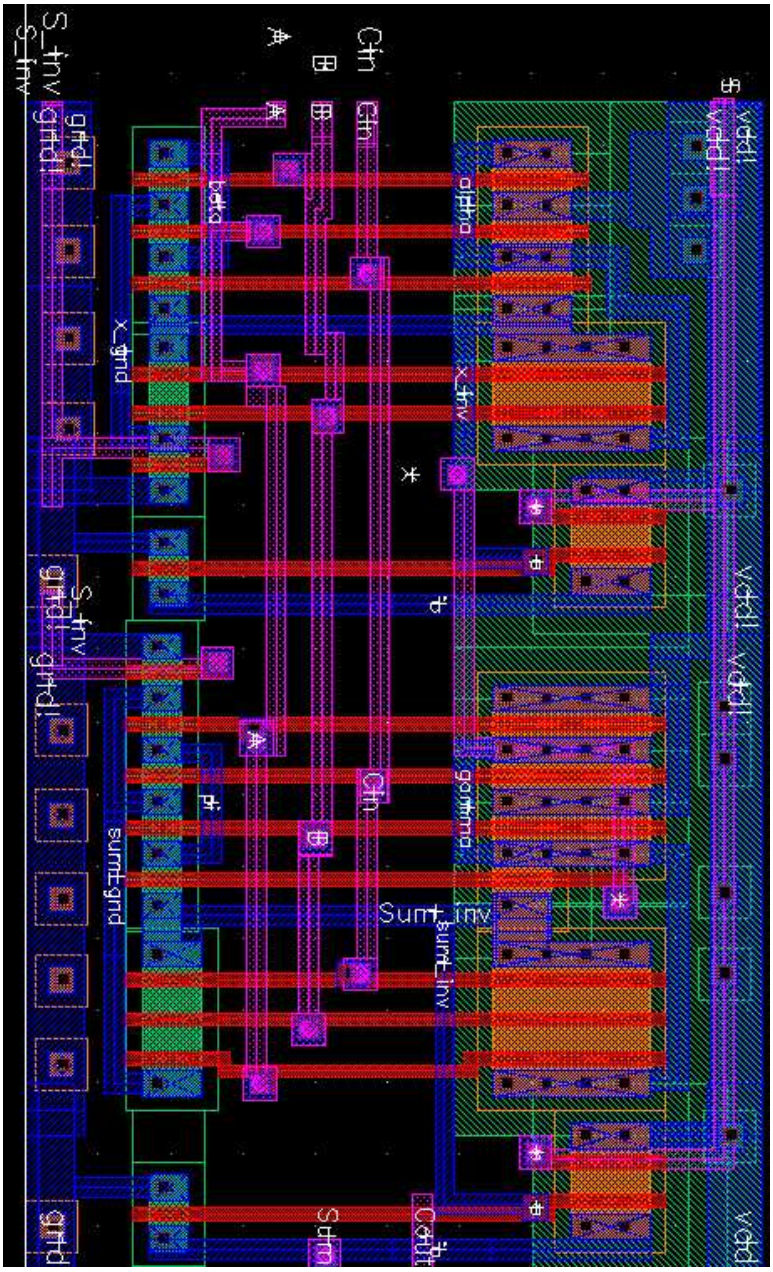


Figure 62: Zigzag approach full adder

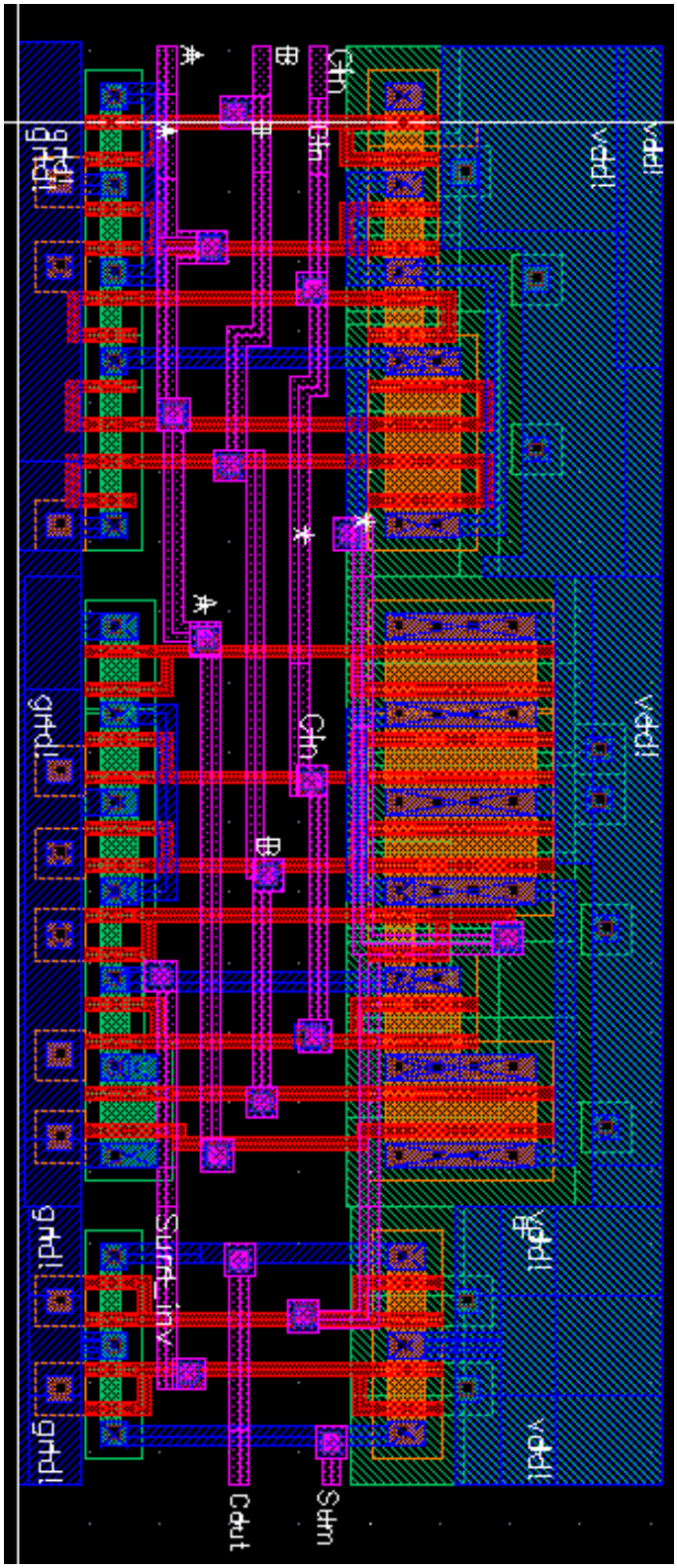


Figure 63: Forced stack approach full adder

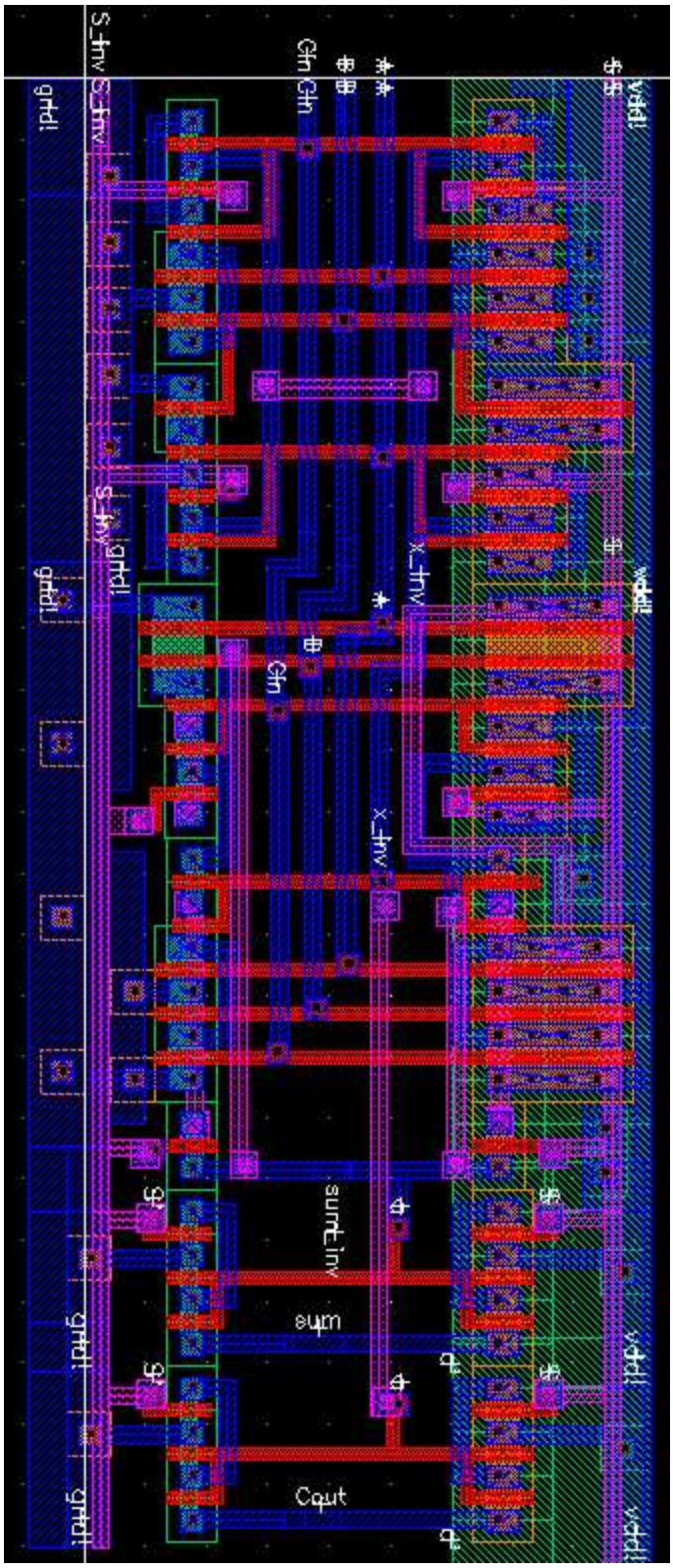


Figure 64: Sleepy stack approach full adder

APPENDIX C

NAND AND NOR LAYOUT

In this appendix, we present NAND and NOR layouts, which are parts of a 4:1 multiplexer, using five different techniques, i.e., base case, sleep, zigzag, forced stack and sleepy stack.

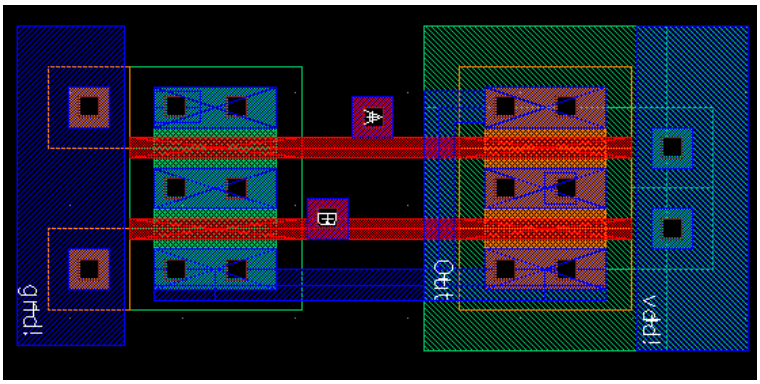


Figure 65: Base case NAND

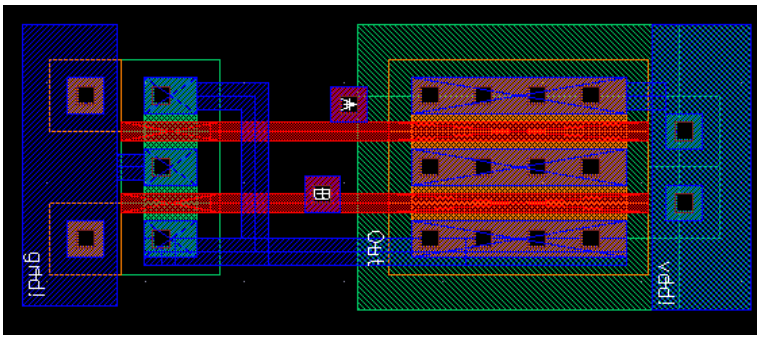


Figure 66: Base case NOR

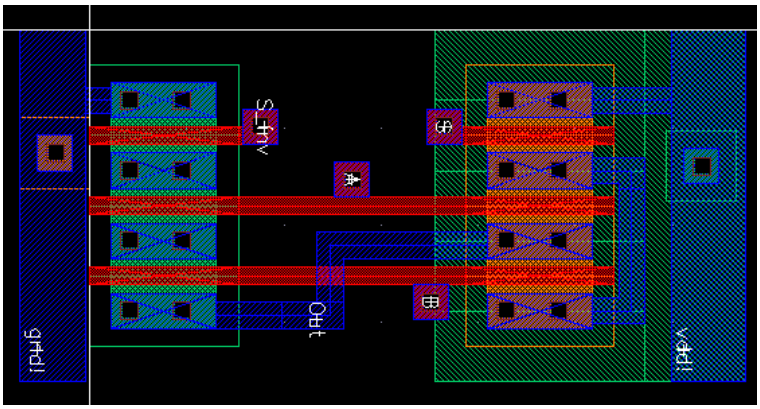


Figure 67: Sleep approach NAND

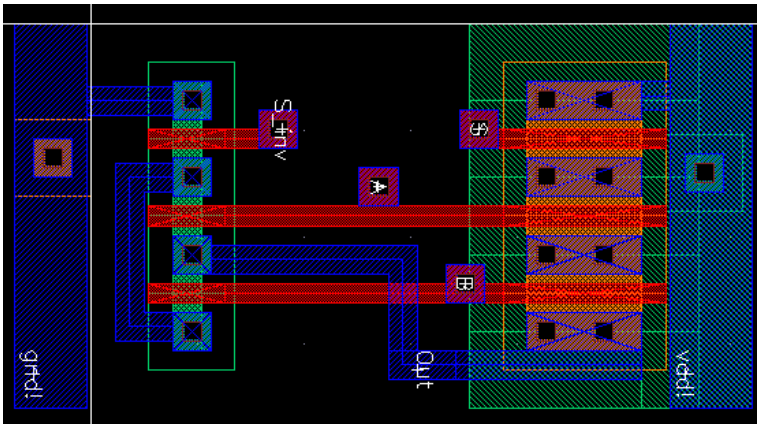


Figure 68: Sleep approach NOR

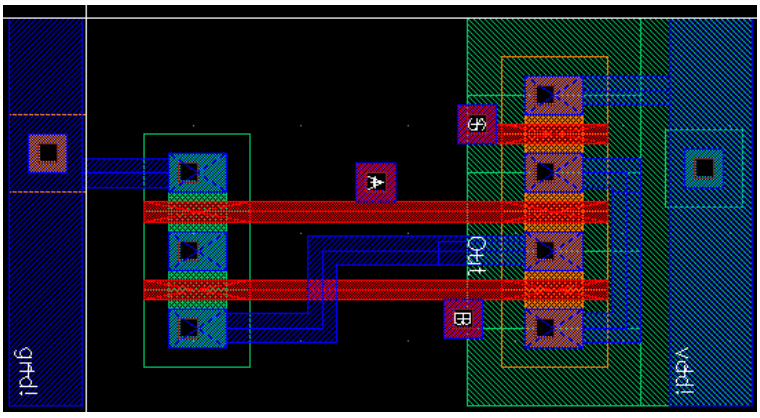


Figure 69: Zigzag approach NAND with pull-up sleep

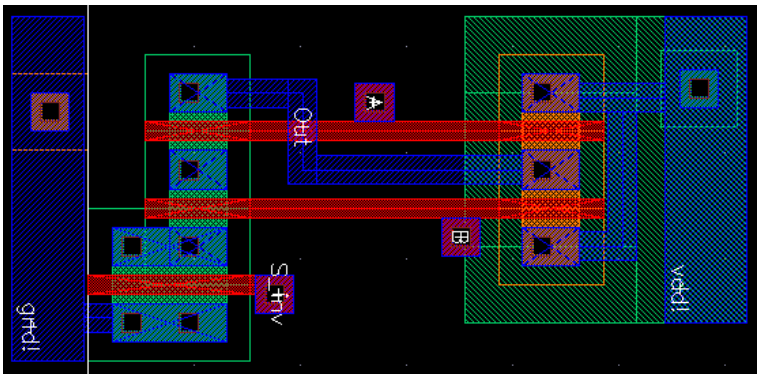


Figure 70: Zigzag approach NAND with pull-down sleep

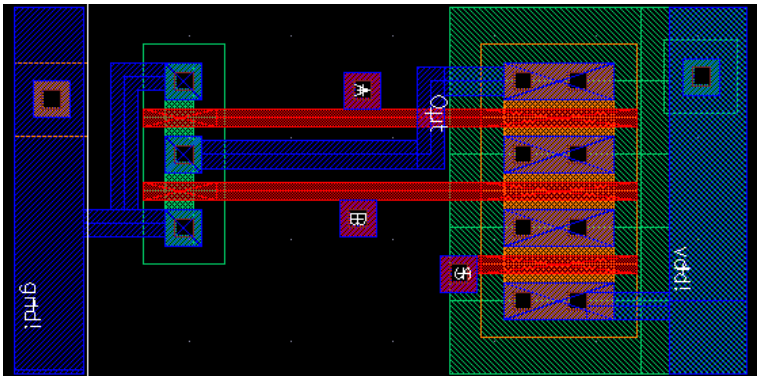


Figure 71: Zigzag approach NOR with pull-up sleep

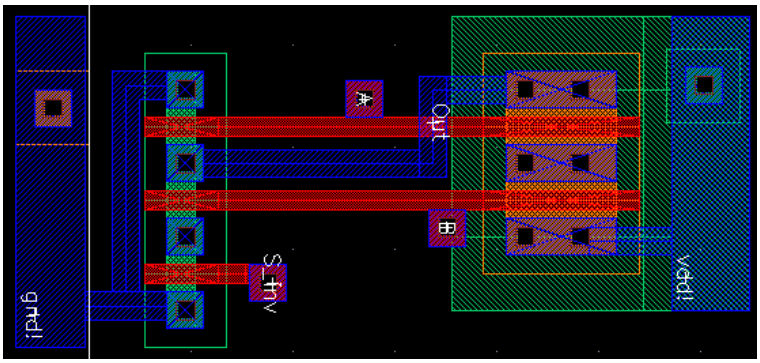


Figure 72: Zigzag approach NOR with pull-down sleep

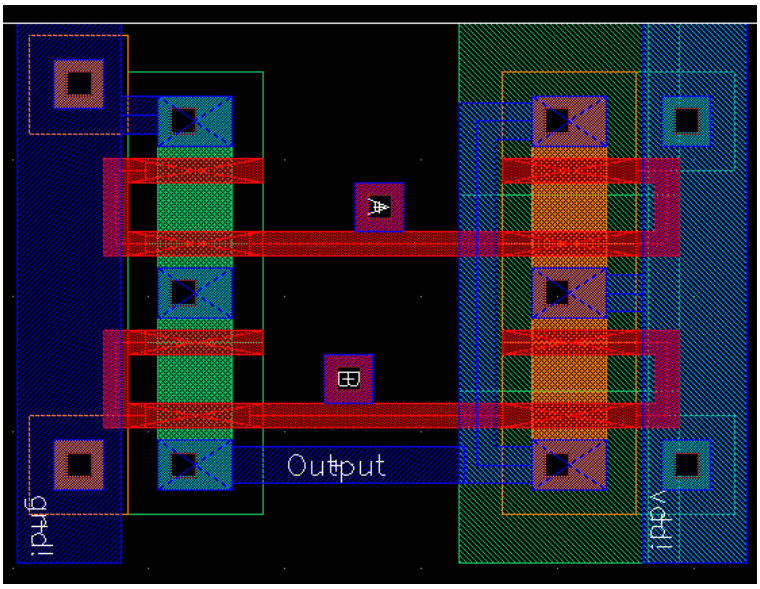


Figure 73: Forced stack approach NAND

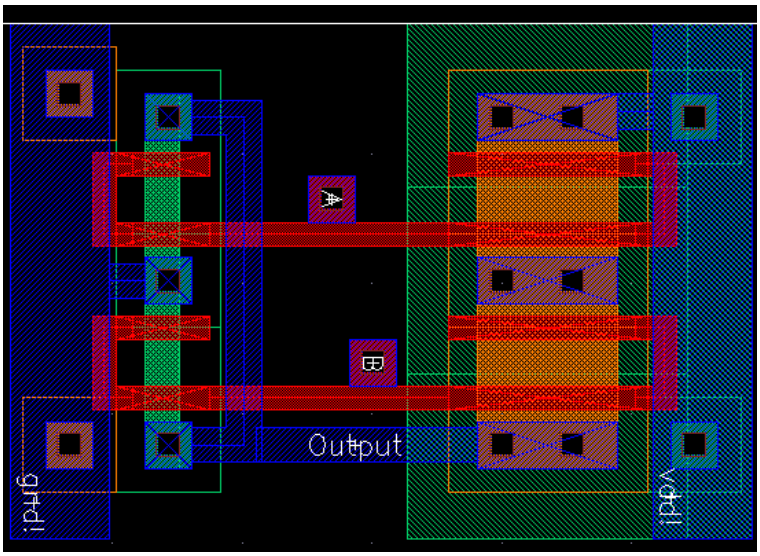


Figure 74: Forced stack approach NOR

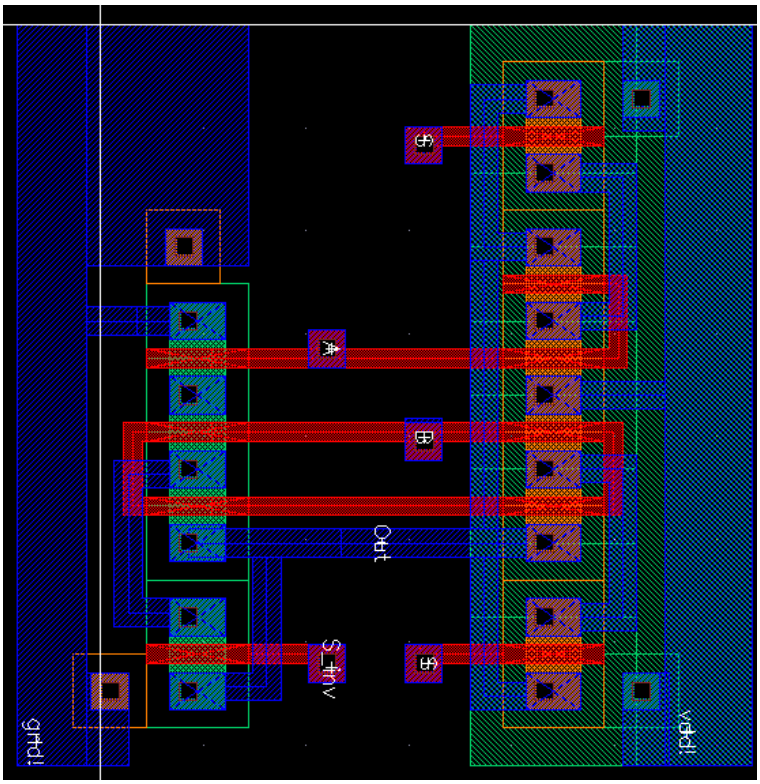


Figure 75: Sleepy stack approach NOR

APPENDIX D

SRAM CELL LAYOUT

In this appendix, we present SRAM cell layout using three different techniques, i.e., base case, forced stack and sleepy stack. For the forced stack and the sleepy stack SRAM cells, we consider four different SRAM cell structures for each technique. The result is we show PD forced stack SRAM cell; PD, WL forced stack SRAM cell; PU, PD forced stack SRAM cell; PU, PD, WL forced stack SRAM cell; PD sleepy stack SRAM cell; PD, WL sleepy stack SRAM cell; PU, PD sleepy stack SRAM cell; and PU, PD, WL sleepy stack SRAM cell.

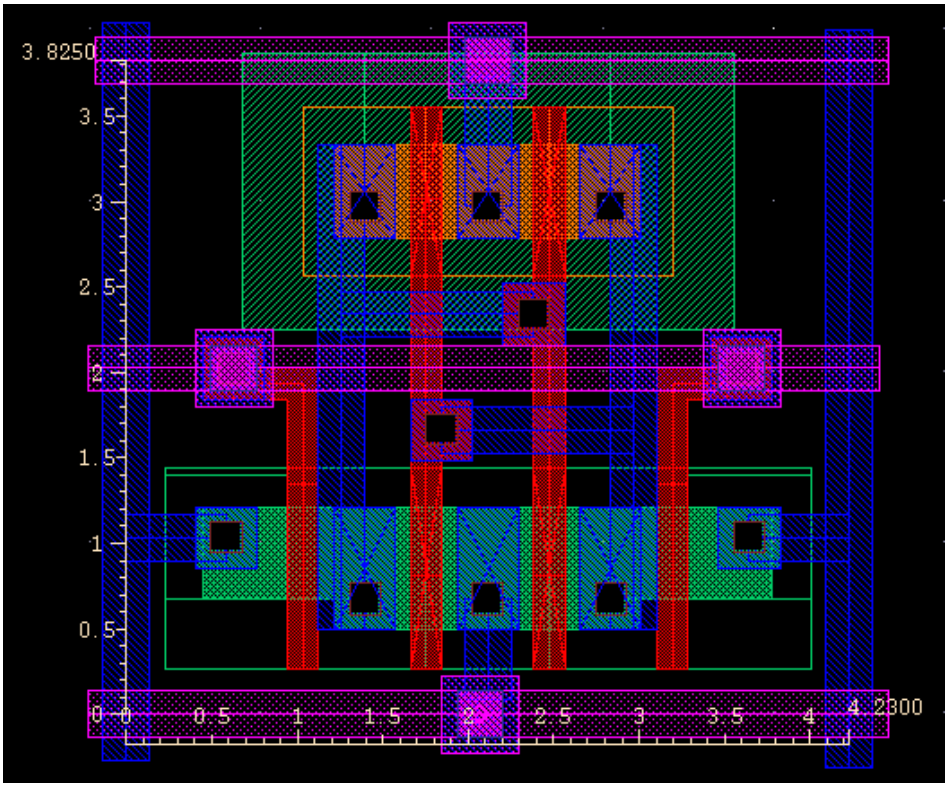


Figure 76: 6-T conventional SRAM cell

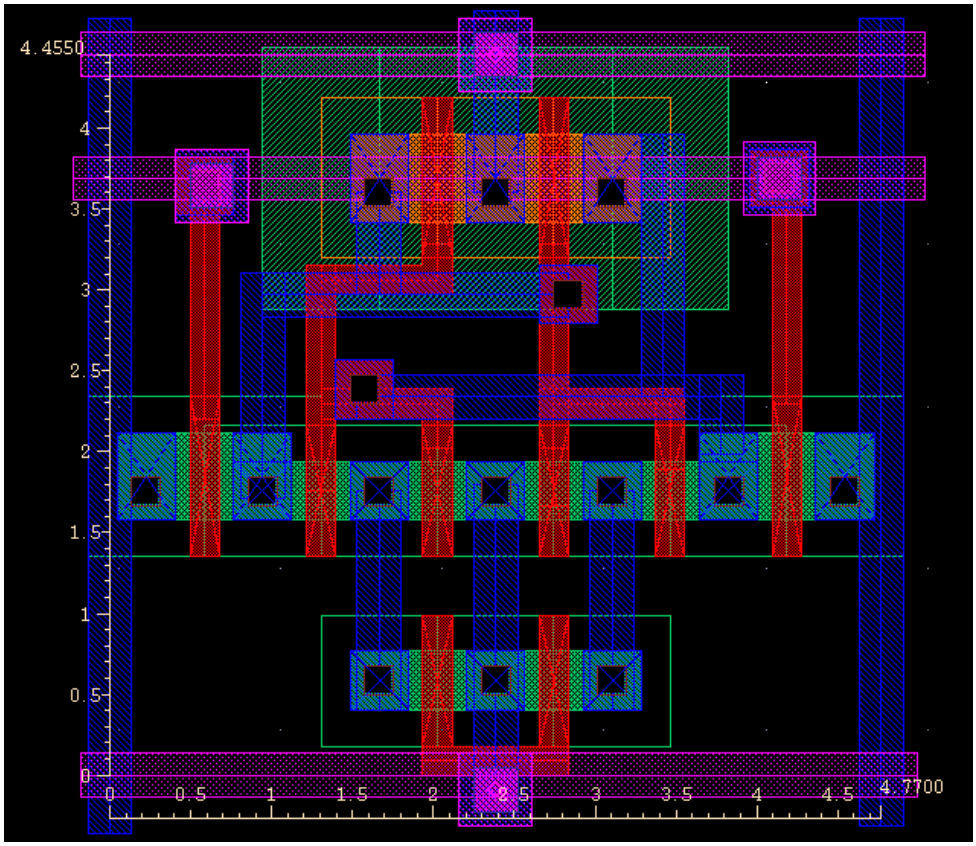


Figure 77: PD sleepy stack SRAM cell

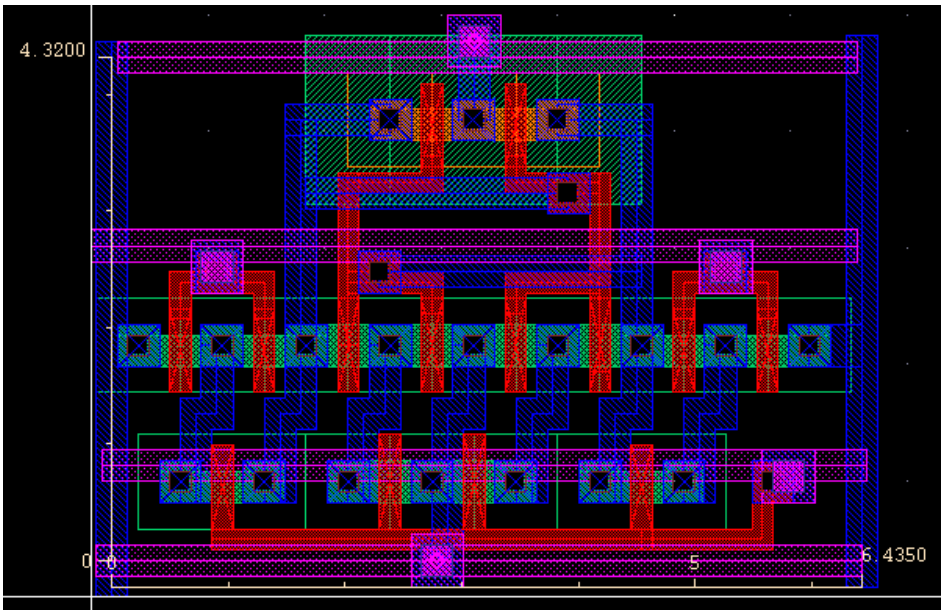


Figure 78: PD, WL sleepy stack SRAM cell

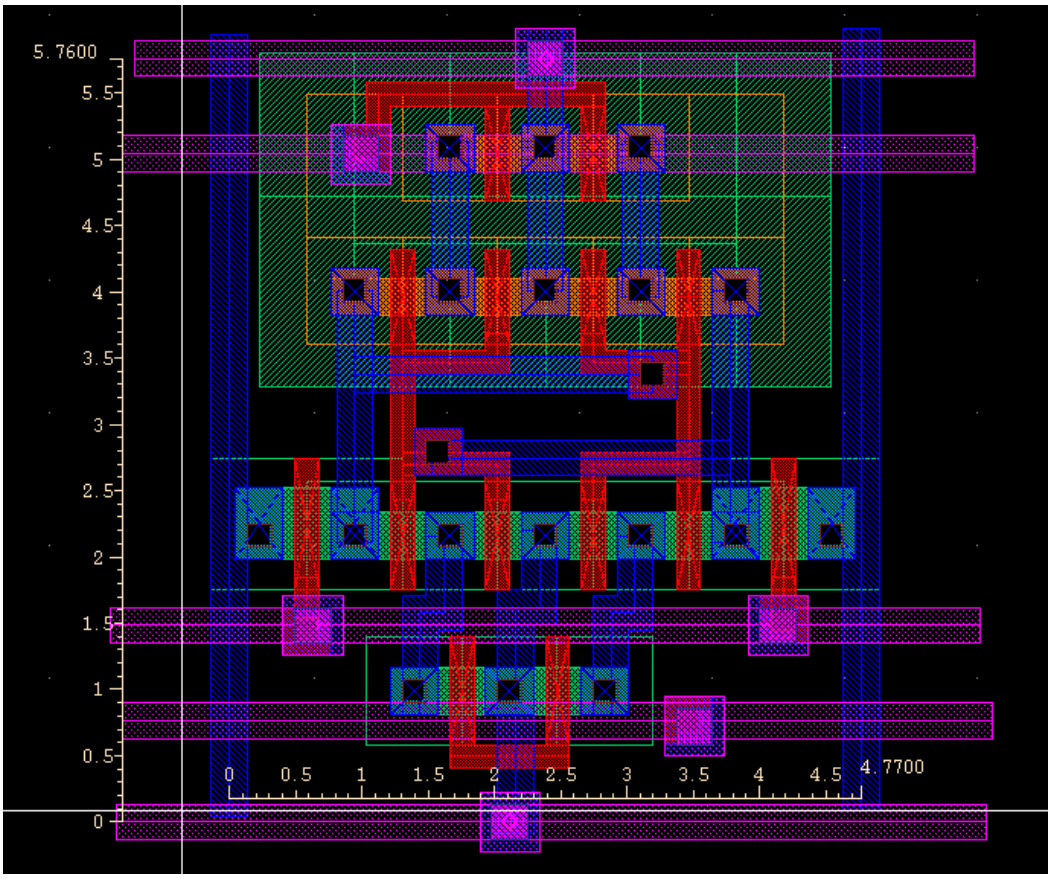


Figure 79: PU, PD sleepy stack SRAM cell

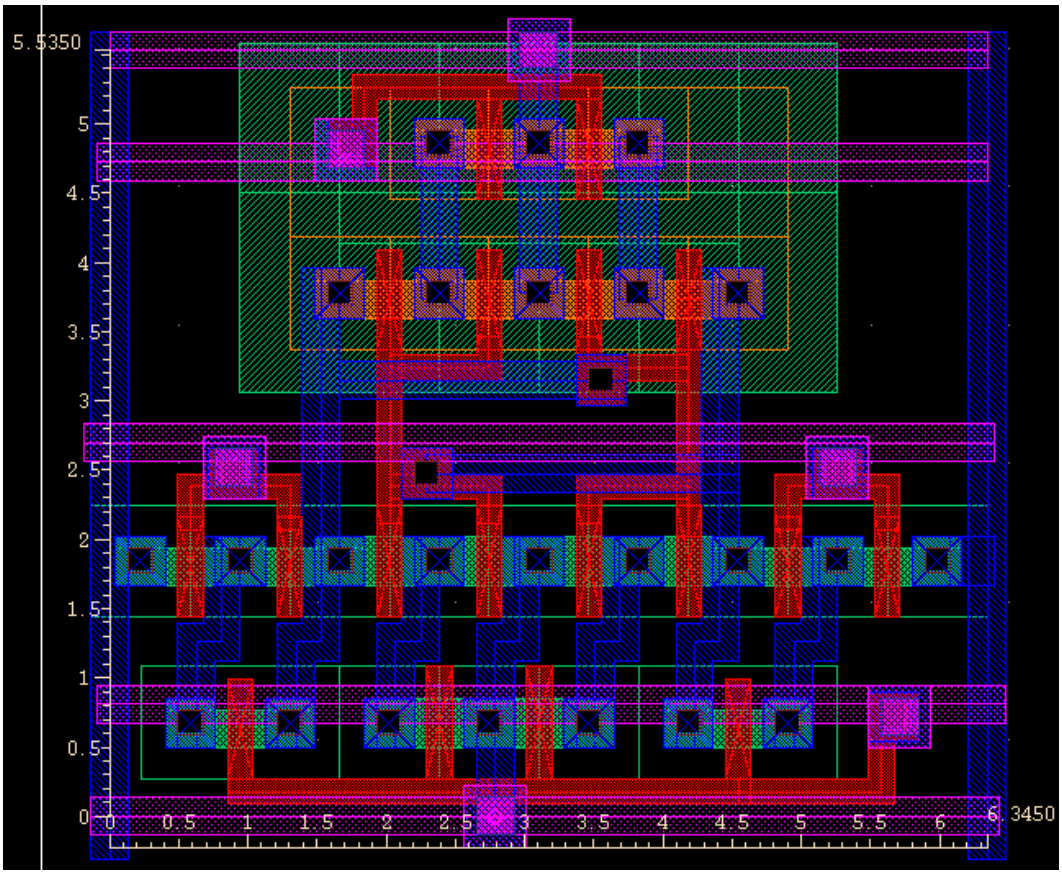


Figure 80: PU, PD, WL sleepy stack SRAM cell

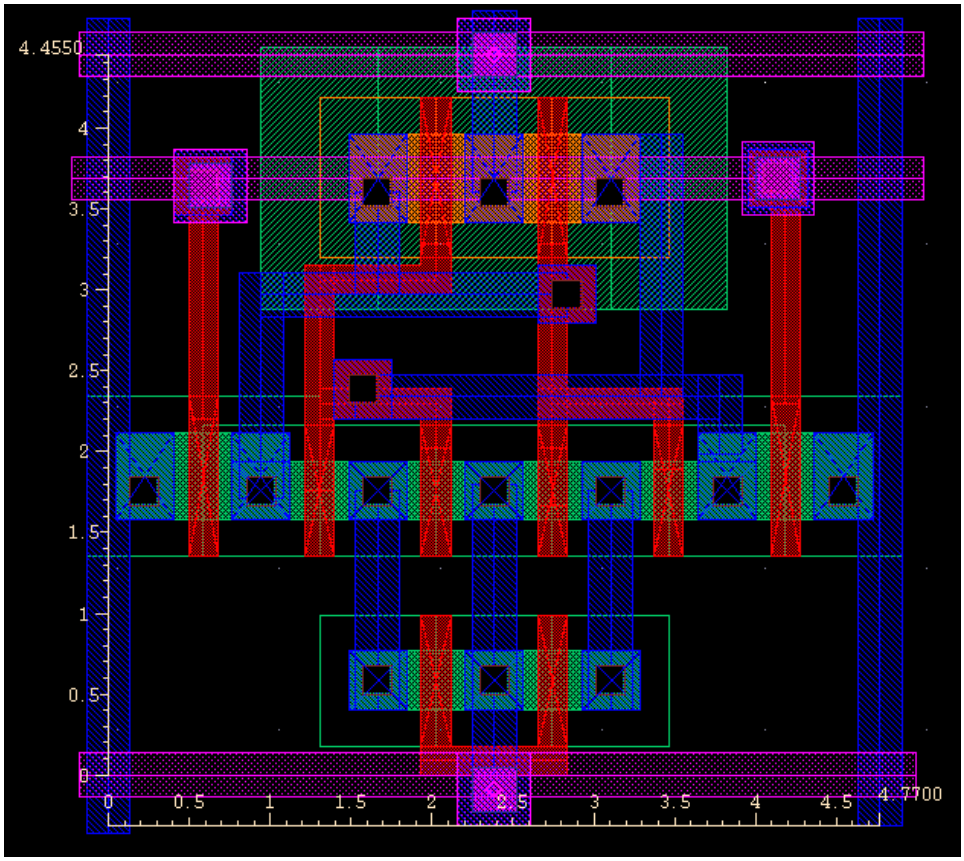


Figure 81: PD forced stack SRAM cell

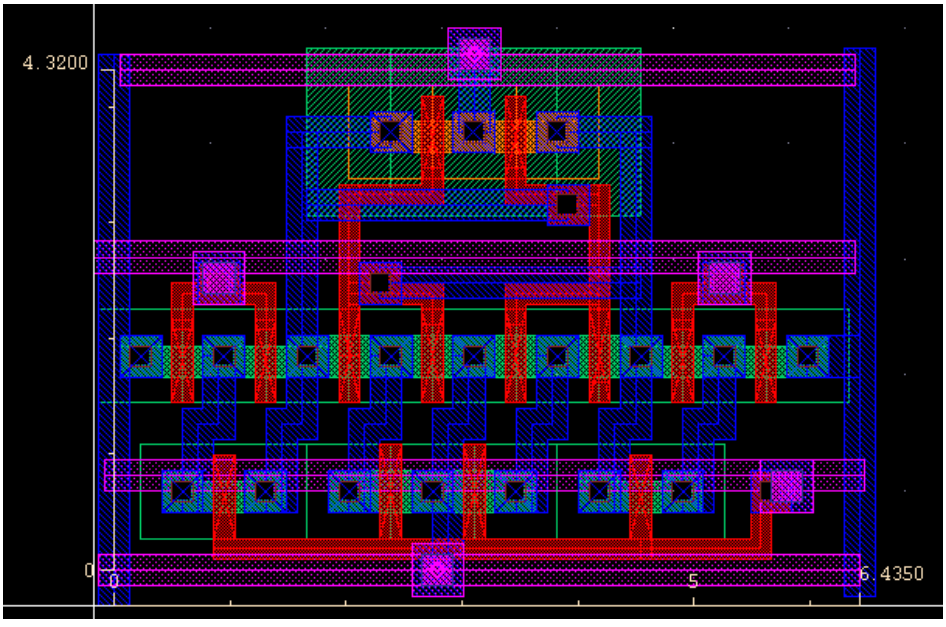


Figure 82: PD, WL forced stack SRAM cell

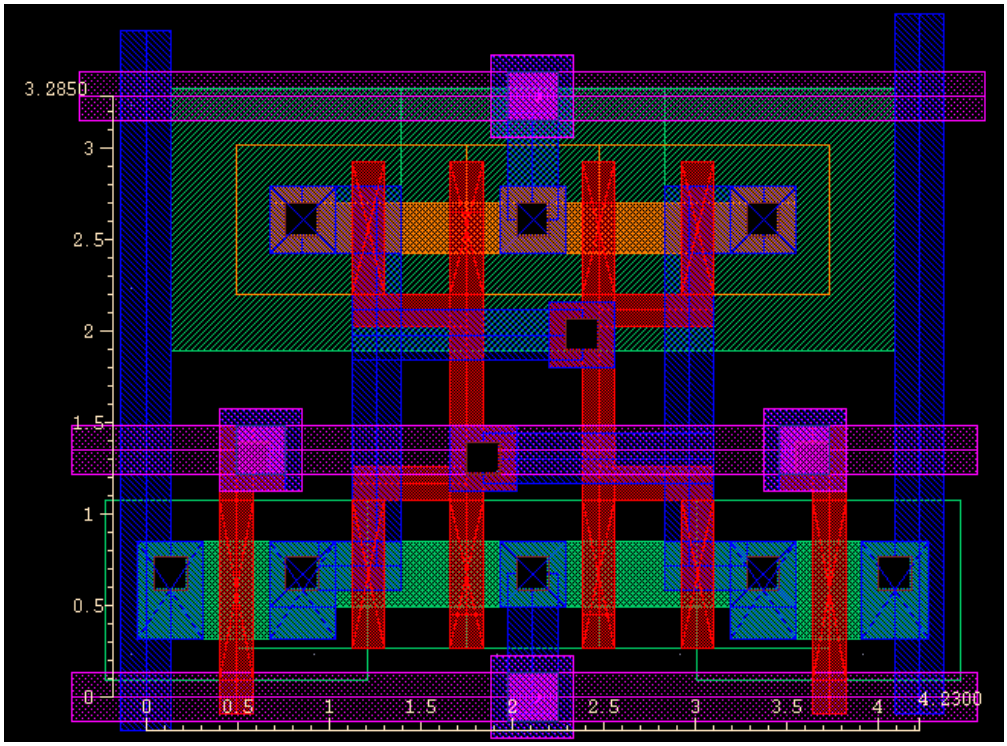


Figure 83: PU, PD forced stack SRAM cell

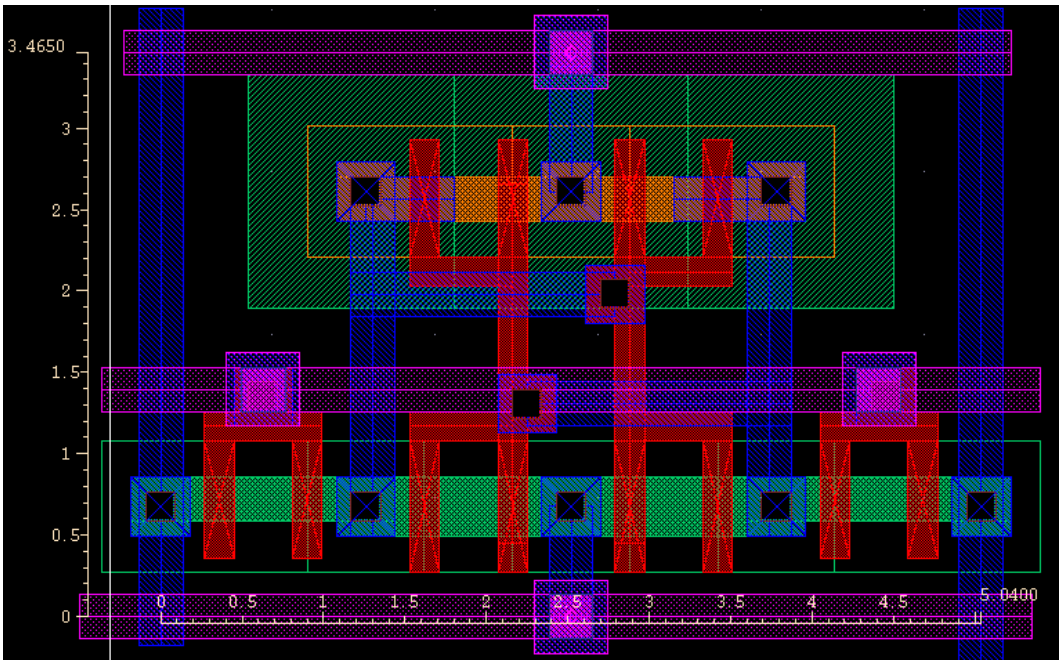


Figure 84: PU, PD, WL forced stack SRAM cell

REFERENCES

[1] AGARWAL, A., HAI, L., and ROY, K., "DRG-Cache: A Data Retention Gated-Ground Cache for Low Power," *Proceedings of the Design Automation Conference*, pp. 473 – 478, June 2002.

[2] AGARWAL, A., ROY, K., and VIJAYKUMAR, T. N., "Exploring High Bandwidth Pipelined Cache Architecture for Scaled Technology," *Proceedings of the Design, Automation and Test in Europe*, pp. 778–783, March 2003.

[3] AMRUTUR, B. S. and HOROWITZ, M. A., "Fast Low-Power Decoders for RAMs," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 10, pp. 1506–1515, October 2001.

[4] ARM Ltd. [Online]. Available http://www.arm.com.

[5] AZIZI, N., MOSHOVOS, A., and NAJM, F., "Low-Leakage Asymmetric-Cell SRAM," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 48–51, August 2002.

[6] AZIZI, N., MOSHOVOS, A., and NAJM, F., "Low-Leakage Asymmetric-Cell SRAM," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 48–51, August 2002.

[7] Berkeley Predictive Technology Model (BPTM). [Online]. Available http://www-device.eecs.berkeley.edu/~ptm/.

[8] BOWMAN, K. A., AUSTIN, B. L., EBLE, J. C., TANG, X., and MEINDL, J. D., "A Physical Alpha-Power Law MOSFET Model," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 10, pp. 1410–1414, October 1999.

[9] BROOKS, D., TIWARI, V., and MARTONOSI, M., "Wattch: A Framework for Architectural Level Power Analysis and Optimizations," *Proceedings of the International Symposium on Computer Architecture*, June 2000.

[10] BURLESON, W. P., CIESIELSKI, M., KLASS, F., and LIU, W., "Wave-pipelining: A Tutorial and Research Survey," *IEEE Transaction on VLSI Systems*, vol. 6, no. 3, pp. 464–474, September 1998.

[11] Cadence Design Systems. [Online]. Available http://www.cadence.com.

[12] CAO, Y., SATO, T., SYLVESTER, D., ORSHANSKY, M., and HU, C., "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 201–204, June 2000.

[13] CHANDRAKASAN, A. P., POTKONJAK, M., MEHRA, R., RABAEY, J., and BRODERSEN, R. W., "Optimizing Power Using Transformations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 1, pp. 12–31, January 1995.

[14] CHANDRAKASAN, A. P., POTKONJAK, M., RABAEY, J., and BRODERSEN, R. W., "HYPER-LP: A System for Power Minimization Using Architectural Transformations," *Proceedings of the International Conference on Computer Aided Design*, pp. 300–303, November 1992.

[15] CHANDRAKASAN, A. P., SHENG, S., and BRODERSEN, R. W., "Low-Power CMOS Digital Design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, April 1992.

[16] CHANG, J.-M. and PEDRAM, M., "Energy Minimization Using Multiple Supply Voltages," *IEEE Transactions on VLSI Systems*, vol. 5, no. 4, pp. 436–443, December 1997.

[17] CHAPPELL, T., CHAPPELL, B., SCHUSTER, S., ALLAN, J., KLEPNER, S., JOSHI, R., and FRANCH, R., "A 2-ns Cycle, 3.8-ns Access 512-kb CMOS ECL SRAM with a Fully Pipelined Architecture," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 11, pp. 1577–1585, 1991.

[18] CHEN, Z., JOHNSON, M., WEI, L., and ROY, K., "Estimation of Standby Leakage Power in CMOS Circuits Considering Accurate Modeling of Transistor Stacks," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 239–244, August 1998.

[19] CHOI, K. and CHATTERJEE, A., "PA-ZSA (Power Aware Zero Slack Algorithm): A Graph Based Timing Analysis for Ultra Low-Power CMOS VLSI," *Proceedings of the Power and Timing Modeling, Optimization and Simulation*, pp. 178–187, October 2002.

[20] CHOI, K.-W. and CHATTERJEE, A., "UDSM (Ultra-Deep Sub-Micron)-Aware Post-Layout Power Optimization for Ultra Low-Power CMOS VLSI," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 72–77, August 2003.

[21] DEGALAHAL, V., VIJAYKRISHNAN, N., and IRWIN, M., "Analyzing soft errors in leakage optimized SRAM design," *IEEE International Conference on VLSI Design*, pp. 227–233, January 2003.

[22] DIRIL, A. U., DHILLON, Y. S., CHOI, K., and CHATTERJEE, A., "An O(N) Supply Voltage Assignment Algorithm for Low-Energy Serially Connected CMOS Modules and a Heuristic Extension to Acyclic Data Flow Graphs," *IEEE Computer Society Annual Symposium on VLSI*, pp. 173–179, February 2003.

[23] FLAUTNER, K., KIM, N. S., MARTIN, S., BLAAUW, D., and MUDGE, T., "Drowsy Caches: Simple Techniques for Reducing Leakage Power," *Proceedings of the International Symposium on Computer Architecture*, pp. 148–157, May 2002.

[24] GUNADI, E. and LIPASTI, M. H., "Cache Pipelining with Partial Operand Knowledge," *Workshop on Complexity-Effective Design*, June 2004.

[25] GUTHAUS, M., RINGENBERG, J., ERNST, D., AUSTIN, T., MUDGE, T., and BROWN, R., "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *Proceedings of the IEEE 4th Annual Workshop on Workload Characterization*, pp. 3–14, December 2001.

[26] HANSON, H., HRISHIKESH, M. S., AGARWAL, V., KECKLER, S. W., and BURGER, D., "Static energy reduction techniques for microprocessor caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, pp. 303–313, June 2003.

[27] HENNESSY, J. and PATTERSON, D., *Computer Architecture: A Quantitative Approach Second Edition*. San Francisco, California: Morgan Kaufmann, 1996.

[28] HP Cambridge Research Laboratory Personal Server Project. [Online]. Available http://crl.research.compaq.com/projects/personalserver/personal-server-spec.html/.

[29] Intel, SA-110 Processor. [Online].
Available http://www.intel.com/design/pca/applicationsprocessors/strong/sa110.htm.

[30] International Technology Roadmap for Semiconductors by Semiconductor Industry Association, 2002. [Online]. Available http://public.itrs.net.

[31] ISHIBASHI, K., KOMIYAJI, K., TOYOSHIMA, H., MINAMI, M., OOKI, N., ISHIDA, H., YAMANAKA, T., NAGANO, F., and NISHIDA, T., "A 300 MHz 4-Mb Wave-pipeline CMOS SRAM Using a Multi-Phase PLL," *IEEE International Solid-State Circuits Conference*, pp. 308–309, February 1995.

[32] JOHNSON, M., SOMASEKHAR, D., CHIOU, L.-Y., and ROY, K., "Leakage Control with Efficient Use of Transistor Stacks in Single Threshold CMOS," *IEEE Transactions on VLSI Systems*, vol. 10, no. 1, pp. 1–5, February 2002.

[33] JOHNSON, M. C. and ROY, K., "Datapath Scheduling with Multiple Supply Voltages and Level Converters," *ACM Transactions on Design Automation of Electronic Systems*, vol. 2, no. 3, pp. 227–248, July 1997.

[34] JOHNSON, M. C., SOMASEKHAR, D., and ROY, K., "Models and Algorithms for Bounds on Leakage in CMOS Circuits," *IEEE Transactions on Computer Aided Design on Integrated Circuits and Systems*, vol. 18, no. 6, pp. 714–725, June 1999.

[35] KHELLAH, M. M. and ELMASRY, M. I., "Power Minimization of High-Performance Submicron CMOS Circuits Using a Dual-Vdd Dual-Vth (DVDV) Approach," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 106–108, 1999.

[36] KIM, C., KIM, J.-J., MUKHOPADHYAY, S., and ROY, K., "A Forward Body-Biased Low-Leakage SRAM Cache: Device and Architecture Considerations," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 6–9, August 2003.

[37] KIM, C. and ROY, K., "Dynamic Vt SRAM: a Leakage Tolerant Cache Memory for Low Voltage Microprocessors," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 251–254, August 2002.

[38] KIM, N., AUSTIN, T., BAAUW, D., MUDGE, T., FLAUTNER, K., HU, J., IRWIN, M., KANDEMIR, M., and NARAYANAN, V., "Leakage Current: Moore's Law Meets Static Power," *IEEE Computer*, vol. 36, pp. 68–75, December 2003.

[39] LEDA Systems Inc. [Online]. Available http://www.ledasys.com.

[40] MICHELI, G. D., *Synthesis and Optimization of Digital Circuits*. USA: McGraw-Hill Inc., 1994.

[41] MIN, K.-S., KAWAGUCHI, H., and SAKURAI, T., "Zigzag Super Cut-off CMOS (ZSCCMOS) Block Activation with Self-Adaptive Voltage Level Controller: An Alternative to Clock-gating Scheme in Leakage Dominant Era," *IEEE International Solid-State Circuits Conference*, vol. 1, pp. 400–401, February 2003.

[42] MONTANARO, J., WITEK, R., ANNE, K., BLACK, A., COOPER, E., DOBBERPUHL, D., DONAHUE, P., ENO, J., HOEPPNER, W., KRUCKEMYER, D., LEE, T., LIN, P., MADDEN, L., MURRAY, D., PEARCE, M., SANTHANAM, S., SNYDER, K., STEHPANY, R., and THIERAUF, S., "A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1703–1714, November 1996.

[43] MOYER, B., "Low-Power Design for Embedded Processors," *Proceedings of the IEEE*, vol. 89, no. 11, pp. 1576–1587, November 2001.

[44] MPEG-2 Encoder / Decoder, Version 1.1 by MPEG Software Simulation Group. [Online]. Available http://www.mpeg.org/MPEG/MSSG/.

[45] MUTOH, S., DOUSEKI, T., MATSUYA, Y., AOKI, T., SHIGEMATSU, S., and YAMADA, J., "1-V Power Supply High-speed Digital Circuit Technology with Multithreshold-Voltage CMOS," *IEEE Journal of Solis-State Circuits*, vol. 30, no. 8, pp. 847–854, August 1995.

[46] NARENDRA, S., DE, V., BORKAR, S., ANTONIADIS, D. A., and CHANDRAKASAN, A. P., "Full-Chip Subthreshold Leakage Power Prediction and Reduction Techniques for Sub-0.18um CMOS," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 2, pp. 501–510, February 2004.

[47] NARENDRA, S., S. BORKAR, V. D., ANTONIADIS, D., and CHANDRAKASAN, A., "Scaling of Stack Effect and its Application for Leakage Reduction," *Proceedings of*

the International Symposium on Low Power Electronics and Design, pp. 195–200, August 2001.

[48] NC State University Cadence Tool Information. [Online]. Available http://www.cadence.ncsu.edu.

[49] NII, K., MAKINO, H., TUJIHASHI, Y., MORISHIMA, C., HAYAKAWA, Y., NUNOGAMI, H., ARAKAWA, T., and HAMANO, H., "A Low Power SRAM Using Auto-Backgate-Controlled MT-CMOS," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 293–298, August 1998.

[50] NOSE, K. and SAKURAI, T., "Analysis and Future Trend of Short-Circuit Power," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 9, pp. 1023–1030, September 2000.

[51] OLUKOTUN, K., MUDGE, T., and BROWN, R., "Performance Optimization of Pipelined Primary Caches," *Proceedings of the International Symposium on Computer Architecture*, pp. 181–190, May 1992.

[52] OLUKOTUN, K., MUDGE, T., and BROWN, R., "Multilevel Optimization of Pipelined Caches," *IEEE Transactions on Computers*, vol. 46, no. 10, pp. 1093–1997, October 1997.

[53] PFEIFFENBERGER, P., PARK, J., and MOONEY, V., "Some Layouts Using the Sleepy Stack Approach," Tech. Rep. GIT-CC-04-05, Georgia Institute of Technology, June 2004.

[54] POWELL, M., YANG, S.-H., FALSAFI, B., ROY, K., and VIJAYKUMAR, T. N., "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-submicron Cache Memories," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 90–95, July 2000.

[55] QUACH, N., "High Availability and Reliability in the Itanium Processor," *IEEE Micro*, vol. 16, pp. 61–69, September 2000.

[56] RAJE, S. and SARRAFZADEH, M., "Variable Voltage Scheduling," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 9–14, August 1995.

[57] Reinman, G. and Jouppi, N., CACTI 2.0: An integrated cache timing and power model. [Online]. Available http://www.research.compaq.com/wrl/people/jouppi/CACTI.html.

[58] ROY, K., WEI, L., and CHEN, Z., "Multiple-Vdd multiple-Vth CMOS (MVCMOS) for low power applications," *IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 366–370, June 1998.

[59] SAKURAI, T. and NEWTON, A. R., "Alpha-Power Law MOSFET Model and Its Application to CMOS Inverter Delay and Other Formulas," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 584–593, April 1990.

[60] SHEU, B., SCHARFETTER, D., KO, P.-K., and JENG, M.-C., "BSIM: Berkeley short-channel IGFET model for MOS transistors," *IEEE Journal of Solid-State Circuits*, vol. 22, pp. 558–566, August 1987.

[61] SimpleScalar. [Online]. Available http://www.simplescalar.com/.

[62] SMITH, J. E., "A Study of Branch Prediction Strategies," *Proceedings of the International Symposium Computer Architecture*, pp. 135–148, 1981.

[63] SRIVASTAVA, A. and SYLVESTER, D., "Minimizing Total Power by Simultaneous Vdd/Vth Assignment," *IEEE Transactions on Computer Aided Design on Integrated Circuits and Systems*, vol. 23, pp. 665–677, May 2004.

[64] STOJANOVIC, V., OKLOBDZIJA, V. G., and BAJWA, R., "A Unified Approach in the Analysis of Latches and Flip-Flops for Low-Power System," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 227–232, August 1998.

[65] Synopsys Inc. [Online]. Available http://www.synopsys.com.

[66] The MOSIS Service. [Online]. Available http://www.mosis.org.

[67] The Simplescalar-Arm power modeling project. [Online]. Available http://www.eecs.umich.edu/˜jringenb/power.

[68] UltraSparc IV Processor Architecture Overview. [Online]. Available http://www.sun.com/.

[69] USAMI, K. and HOROWITZ, M., "Clustered Voltage Scaling Technique for Low-Power Design," *Proceedings of the International Symposium on Low Power Electronics and Design*, pp. 3–8, April 1995.

[70] USAMI, K., IGARASHI, M., MINAMI, F., ISHIKAWA, T., KANZAWA, M., ICHIDA, M., and NOGAMI, K., "Automated Low-Power Technique Exploiting Multiple Supply Voltages Applies to a Media Processor," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 3, pp. 463–472, March 1998.

[71] Using the GNU Compiler Collection. [Online]. Available http://gcc.gnu.org/onlinedocs.

[72] UYEMURA, J. P., *CMOS Logic Circuit Design Second Edition*. Norwell, Massachusetts USA: Kluwer Academic Publishers, 1999.

[73] WEI, L., CHEN, Z., JOHNSON, M., ROY, K., and DE, V., "Design and Optimization of Low Voltage High Performance Dual Threshold CMOS Circuits," *Proceedings of the Design Automation Conference*, pp. 489–494, June 1998.

[74] WESTE, N. and ESHRAGHIAN, K., *Principles of CMOS VLSI Design*. Santa Clara, California: Addison Wesley, 1992.

PUBLICATIONS

This dissertation is based on and/or related to the work and results presented in the following publications:

[1] **J. C. Park** and V. J. Mooney, "Pareto Points in SRAM Design Using the Sleepy Stack Approach," *IFIP International Conference on Very Large Scale Integration* (IFIP VLSI-SOC'05), October 2005.

[2] **J. C. Park**, V. J. Mooney and P. Pfeiffenberger, "Sleepy Stack Reduction in Leakage Power," *Proceedings of the International Workshop on Power and Timing Modeling, Optimization and Simulation* (PATMOS'04), pp.148-158, September 2004.

[3] P. Pfeiffenberger, **J. C. Park** and V. J. Mooney, "Some Layouts Using the Sleepy Stack Approach," Technical Report GIT-CC-04-05, Georgia Institute of Technology, June 2004, [Online] Available http://www.cc.gatech.edu/tech_reports/index.04.html.

[4] A. Balasundaram, A. Pereira, **J. C. Park** and V. J. Mooney, "Golay and Wavelet Error Control Codes in VLSI," *Proceedings of the Asia and South Pacific Design Automation Conference* (ASPDAC'04), pp. 563-564, January 2004.

[5] A. Balasundaram, A. Pereira, **J. C. Park** and V. J. Mooney, "Golay and Wavelet Error Control Codes in VLSI," Technical Report GIT-CC-03-33, Georgia Institute of Technology, December 2003,
[Online] Available http://www.cc.gatech.edu/tech_reports/index.03.html

[6] **J. C. Park**, V. J. Mooney and S. K. Srinivasan, "Combining Data Remapping and Voltage/Frequency Scaling of Second Level Memory for Energy Reduction in Embedded Systems," *Microelectronics Journal*, 34(11), pp. 1019-1024, November 2003.

[7] **J. C. Park**, V. J. Mooney, K. Palem and K. W. Choi, "Energy Minimization of a Pipelined Processor using a Low Voltage Pipelined Cache," *Conference Record of the 36th Asilomar Conference on Signals, Systems and Computers* (ASILOMAR'02), pp. 67-73, November 2002.

[8] K. Puttaswamy, K. W. Choi, **J. C. Park**, V. J. Mooney, A. Chatterjee and P. Ellervee, "System Level Power-Performance Trade-Offs in Embedded Systems Using Voltage

and Frequency Scaling of Off-Chip Buses and Memory," *Proceedings of the International Symposium on System Synthesis* (ISSS'02), pp. 225-230, October 2002.

[9] S. K. Srinivasan, **J. C. Park** and V. J. Mooney, "'Combining Data Remapping and Voltage/Frequency Scaling of Second Level Memory for Energy Reduction in Embedded Systems," *Proceedings of the International Workshop on Embedded System Codesign* (ESCODES'02), pp. 57-62, September 2002.

[10] K. Puttaswamy, L. N. Chakrapani, K. W. Choi, Y. S. Dhillon, U. Diril, P. Korkmaz, K. K. Lee, **J. C. Park**, A. Chatterjee, P. Ellervee, V. J. Mooney, K. Palem and W. F. Wong, "Power-Performance Trade-Offs in Second Level Memory Used by an ARM-Like RISC Architecture," in the book Power Aware Computing, edited by Rami Melhem, University of Pittsburgh, PA, USA and Robert Graybill, DARPA/ITO, Arlington, VA, USA, published by Kluwer Academic/Plenum Publishers, pp. 211-224, May 2002.