DESIGN TECHNIQUES AND TRADEOFFS OF

FINFET SRAM MEMORIES

by

MICHAEL ALLEN TURI

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

MAY 2013

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of MICHAEL ALLEN TURI find it satisfactory and recommend that it be accepted.

____
José G. Delgado-Frias, Ph.D., Chair

____
Deukhyoun Heo, Ph.D.

____
Partha P. Pande, Ph.D.

ACKNOWLEDGEMENTS

The research in this dissertation has been supported by the Boeing Centennial Endowed Chair, 2011 IEEE Circuits and Systems Society Pre-doctoral Scholarship #2, School of Electrical Engineering and Computer Science (EECS), and Washington State University (WSU). I wish to extend many thanks for this funding which has made this work possible. This research was conducted in the High Performance Computer Systems (HiPerCopS) group at WSU under the direction of Dr. José Delgado-Frias. I am tremendously appreciative of the guidance and encouragement provided by Dr. Delgado-Frias; his involvement and assistance greatly influenced the research presented in this dissertation. I also appreciate the help and suggestions from the other members of my dissertation committee: Dr. Partha Pande and Dr. Deuk Heo. I also wish to thank Dr. Jabu Nyathi for his help during the early stages of my research and I want to thank Al Guyer for his help with the Linux systems used for this research and for bailing me out when I was having system or network problems.

I greatly appreciated the help, camaraderie, and motivation provided by the HiPerCopS group's graduate students. I wish to thank Jason Van Dyken and Zhe (Nick) Zhang for their friendship and help during the bulk of my research. I thank Johnathan Cree for his friendship and help during my degree and at our internship at IHP-Microelectronics. I also wish to thank Mitch Myjak and Danny Blum for their help during the early stages of my research and my career search. I thank Gina Sprint, Daniel Iparraguire, and Steve Wang for their camaraderie during my final year. I also want to thank students in EECS for their friendship and help: Vic Valgenti, Tao Yang, Sou Sarkar, Sujay Deb, Turbo Majumder, Ding Ma, Xinmin Yu, Kevin Chang, Tao Yang, Kylan Robinson, and Guillermo Ramirez-Conejo. I especially thank Lorie Mochel for her support, motivation, and assistance during my time at WSU. I also want to thank many other friends and students for their help and friendship during my time at WSU for both my B.S. and Ph.D.

I wish to extend a special thank you to my parents, Ray and Donna, my brother Steve, and my relatives for their love, support, and encouragement throughout my life; I could not have been successful in my educational pursuits without this support. I also wish to thank my girlfriend Elizabeth Edwards for her love, support, encouragement, and providing more balance to my life.

DESIGN TECHNIQUES AND TRADEOFFS OF

FINFET SRAM MEMORIES

Abstract

by Michael Allen Turi, Ph.D.
Washington State University
May 2013

Chair: José G. Delgado-Frias

Nine novel eight-transistor (8T) FinFET SRAM cell schemes using different shorted gate (SG) or low power (LP) FinFET configurations are studied and evaluated comprehensively in terms of leakage current, delay, read and write energy dissipation, energy delay product (EDP), and static noise margin. Comparisons to conventional 6T SRAM schemes reveal that the 8T SRAM schemes perform better, especially for a 32-bit by 1024-word (32×1024) array, since leakage current can be reduced by low-power schemes that reverse-bias the inverter transistors' back gates without adversely impacting read speed or read static noise margin. FinFETs provide significantly lower leakage current and higher on-current than bulk-CMOS transistors and allow the 8T FinFET SRAM schemes to greatly outperform 8T 32 nm CMOS SRAM cells. 8T SRAM configuration choices further affect these performance metrics. Reverse-biasing the inverter FinFETs' back gates can reduce leakage current by 2-97%. Additionally, FinFET SRAM cells designed for low-leakage are more effective than header and/or footer transistors added to a cell to reduce leakage current. For a 32×1024 array, read delay is dominant and can be minimized by using SG configuration for the read transistors. These two performance metrics are chiefly responsible for determining the energy consumption of a SRAM array. Reverse-biasing the inverter FinFETs' back gates also minimizes leakage current and EDP variation due to parameter, voltage, and temperature (PVT) variations. The 8T LP_INV, low-power inverters, scheme uses these configurations and is the best-performing FinFET SRAM scheme at a 1 V V_{DD}, and in particular the 8T LP_INV1.2 cell has 60% less EDP than the conventional 8T SG FinFET SRAM scheme and performs best under PVT variations. Similar relative performance is observed for the cells at 0.6 V near-threshold operation; most cells yield reduced EDP and the 8T LP_INV1.2 cell still performs best. However, these cells suffer increased changes in performance due to PVT variations and increased delay. The LP_INV1.2 cell is also the best-performing FinFET SRAM cell at near-threshold operation; however, the 8T LP_SGR cell has less performance variation for PVT variations. The tradeoff is that the LP_SGR cell has a slightly higher average EDP than the LP_INV1.2 cell.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

For the past four decades CMOS scaling has offered improved performance from one technology node to the next. This in turn has brought smaller and faster digital systems. However, future bulk CMOS scaling faces considerable challenges due to material and process technology limits [1]. According to the 2011 International Technology Roadmap for Semiconductors (ITRS) [2], obstacles to the increased scaling of bulk CMOS include short-channel effects, sub-threshold leakage, gate-dielectric leakage, and device-to-device variations. These obstacles affect circuit and system reliability. The aforementioned challenges will become more prominent as CMOS scaling approaches atomic and quantum-mechanical physics boundaries [3]. Efforts to extend silicon scaling through innovations in materials and device structure continue.

FinFETs, which are double-gate field-effect transistors, are able to overcome these scaling obstacles [2] [4]. One of the most important features of FinFETs is that the front and back gates may be made independent and biased to control the current and the device threshold voltage [5]. This ability to control threshold voltage variations offers a temporary means to manage the challenge of standby power dissipation. FinFET is considered a promising technology that can impact the immediate future due to its high-performance, low leakage power consumption, reduced susceptibility to process variations, and ease of manufacture using current processes [1]. Gate lengths of 10 nm and below will be achievable with FinFETs. These features make FinFETs a strong candidate to bridge the technology gap between mainstream bulk CMOS and non-Silicon devices, such as carbon nanotubes.

FinFETs can be a replacement for bulk-CMOS transistors in many different designs. Its low leakage/standby power property makes FinFETs a desirable option for memory sub-systems. Memory modules are widely used in most digital and computer systems. Leakage power is very important in memory cells since most memory applications access only one or very few memory rows at a given time. The great majority of memory cells draw only leakage power. In microprocessors, the clock network power consumption due to memory devices (i.e., cache memory, register, and pipeline registers) accounts for 51% of the total power [6]. The application of FinFET technology to memories can save significant power. Optimizations can also be made to other components of memory modules, such as the address decoder and I/O buffers, to obtain power savings or a faster system.

1.1 FinFET Technology

The following subsections include an introduction to the FinFET technology used in this research. A key benefit of using FinFETs is the ability to configure the back gates of the devices to provide greater speed or greater leakage control; this design flexibility is described in the next subsection. Afterwards, the FinFET technology model used in this research is presented as well as the scripts used to aid simulating and gathering simulation results.

1.1.1 FinFET Back-Gate Biasing Strategies

FinFETs suffer from less leakage current since their geometry allows for better control over the channel. FinFETs may be substituted into a former bulk-CMOS design by merely shorting the front- and back-gates together during device fabrication to allow only one gate connection per FinFET. This transistor configuration is often called shorted gate (SG) and is illustrated in Figure 1.1(a). However, the FinFET's gates may be made independent (also referred to as a MIGFET, or multiple independent gate FET) to allow for separate control over the front- and back-gates [5]. An independent back gate enables greater design flexibility and allows novel biasing or control schemes to increase device speed or reduce power consumption. In the low-power (LP) operating mode, presented in Figure 1.1(b), a reverse-bias on the back gate greatly reduces leakage current (I_{off}). Having a back-gate bias of -0.2V on an n-type FinFET reduces I_{off} by 90% of a comparable SG FinFET [7]. A reverse-bias negatively affects the on current (I_{on}) of the transistor. I_{on} reduces by about 60% in the LP mode when the reverse-bias is set to -0.2V for an n-type transistor [7]. Smaller reverse-biases have similar effects on I_{off} and I_{on}, but to a lesser extent [7]. These properties make FinFETs promising devices for SRAM memories. Back-gate biases for LP-mode transistors can be statically driven by a voltage generator circuit [8]. Dynamic driving of back-gate biases is more difficult and requires more complex voltage generator circuitry, but could lead to more optimal performance of the biased system. The back-gate can also be tied to another input, leading to independent-gate (IG) operating mode, pictured in Figure 1.1(c).

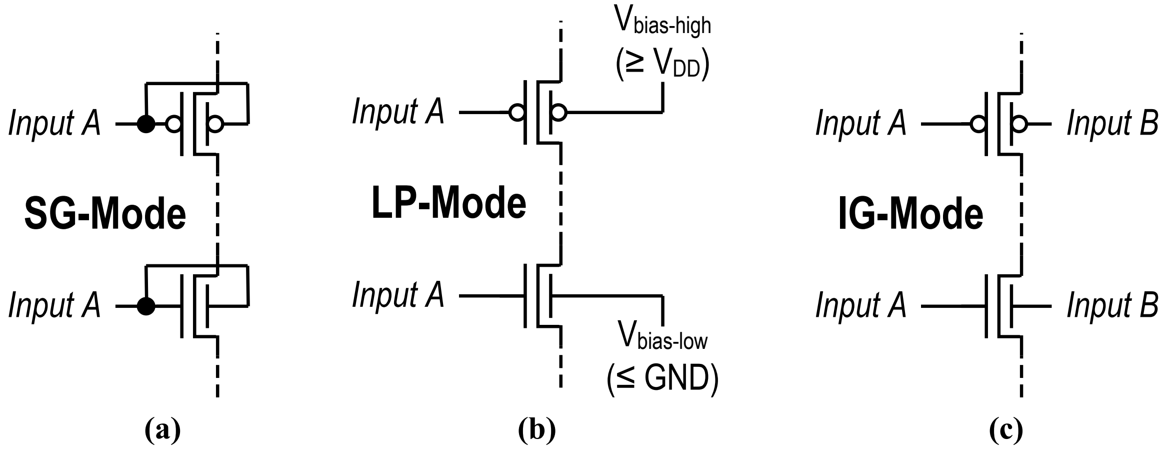


Figure 1.1. FinFET configuration modes: (a) Shorted-gate (SG) mode FinFETs (b) Low-power (LP) mode FinFETs (c) Independent-gate (IG) mode FinFETs

1.1.2 FinFET Technology Model

For this research, University of Florida's Spice3-UFDG (Linux version 3.7) was used to model n- and p-type fully-depleted (FD) silicon-on-insulator (SOI) FinFETs. Spice3-UFDG is a physics-based model calibrated to follow predicted results from Synopsys MEDICI and measured results from symmetrical double-gate FinFETs fabricated at Motorola [9] [10]. Other commonly-used FinFET simulation models available to the research community are the predictive technology model (PTM) [11] and BSIM-CMG/BSIM-IMG [12]. Table 1.1 shows the values of FinFET parameters used in this research. The L_G, T_{ox}, and R_{SD} values are from the "High-performance Logic Technology Requirements" in the 2007 ITRS Process Integration, Devices, and Structures report [2].

Table 1.1. FinFET device parameters

Parameter	Value
N-Channel Surface Orientation	<110>
Gate length (L_G)	30 nm
Gate to source/drain underlap (L_{SD})	12 nm
Fin height (H_{fin})	75 nm
Fin thickness (T_{SI})	15 nm
Oxide thickness (T_{ox})	1.2 nm
Gate thickness (T_G)	20 nm
Gate work function (Φ_G)	4.4 eV (n-type) 4.8 eV (p-type)
Low-field mobility for thick T_{SI} (μ0)	565 cm^2/(V-s) (n-type)
	250 cm^2/(V-s) (p-type)
Fin body doping (N_{Body})	10^{15} cm^{-3}
Source/drain doping (N_{DS})	10^{20} cm^{-3}
Source/drain resistance (R_{SD})	170 Ω-μm
Supply voltage (V_{DD})	1 V

The following figures show the current-voltage (I-V) characteristics of this particular FinFET technology sizing. Figure 1.2 and Figure 1.3 show the I-V characteristics for n- and p-type SG-mode FinFETs, respectively. Figure 1.4 and Figure 1.5 show the I-V characteristics for n- and p-type LP-mode FinFETs biased at 0 V and 1 V, respectively. Figure 1.6 and Figure 1.7 show the I-V characteristics for n- and p-type LP-mode FinFETs biased at -0.2 V and 1.2 V, respectively.

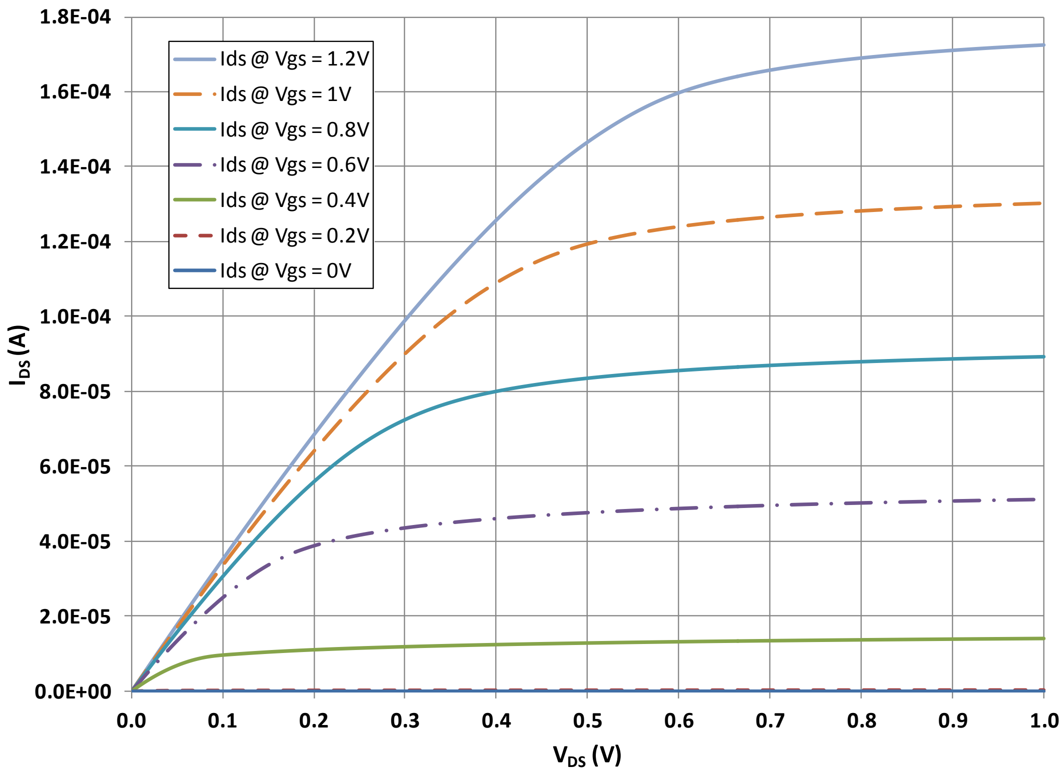


Figure 1.2. I-V curve of an n-type SG-mode FinFET

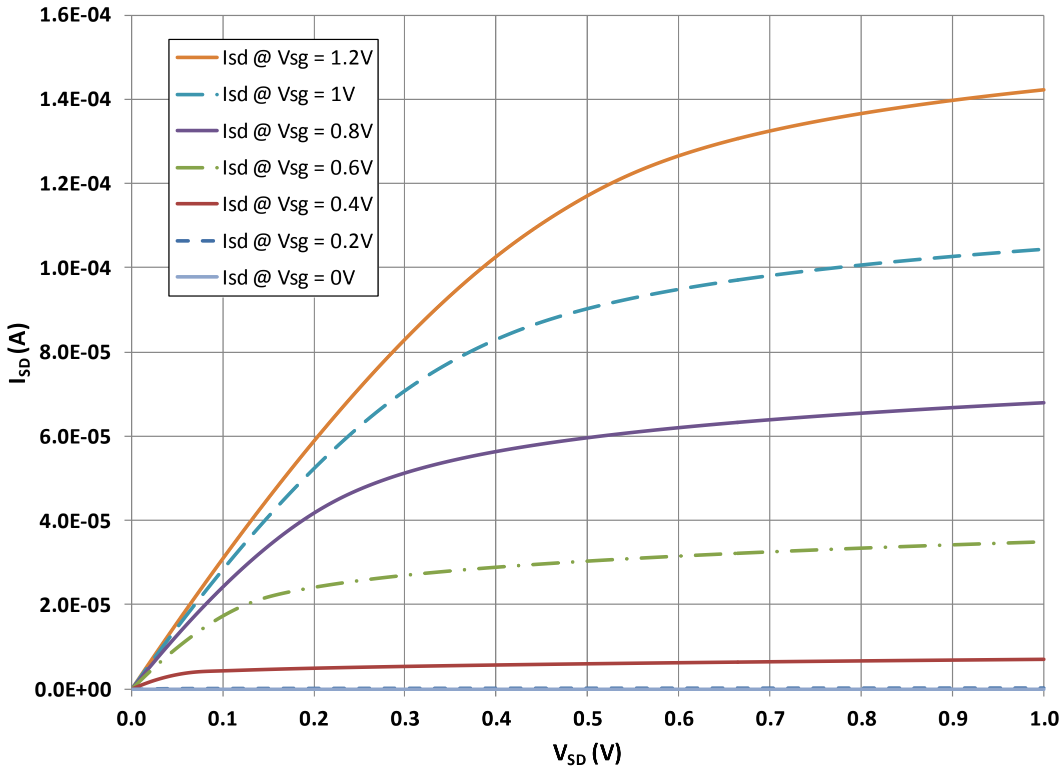


Figure 1.3. I-V curve of a p-type SG-mode FinFET

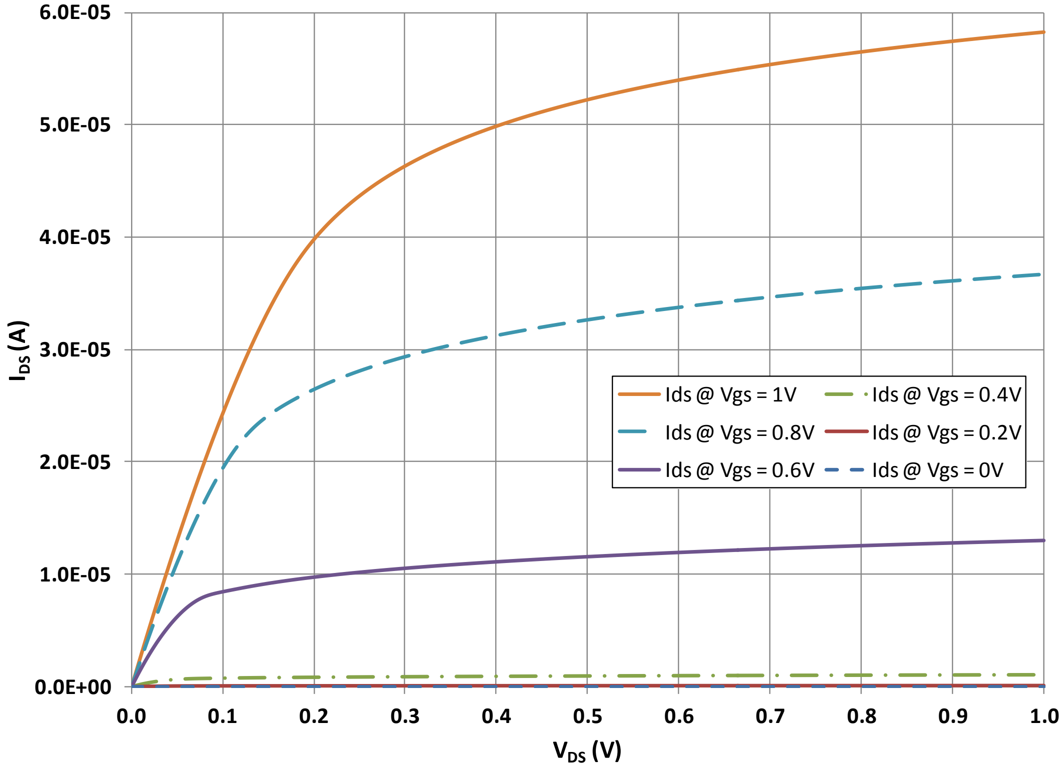


Figure 1.4. I-V curve of an n-type LP-mode FinFET with a 0 V back-gate bias

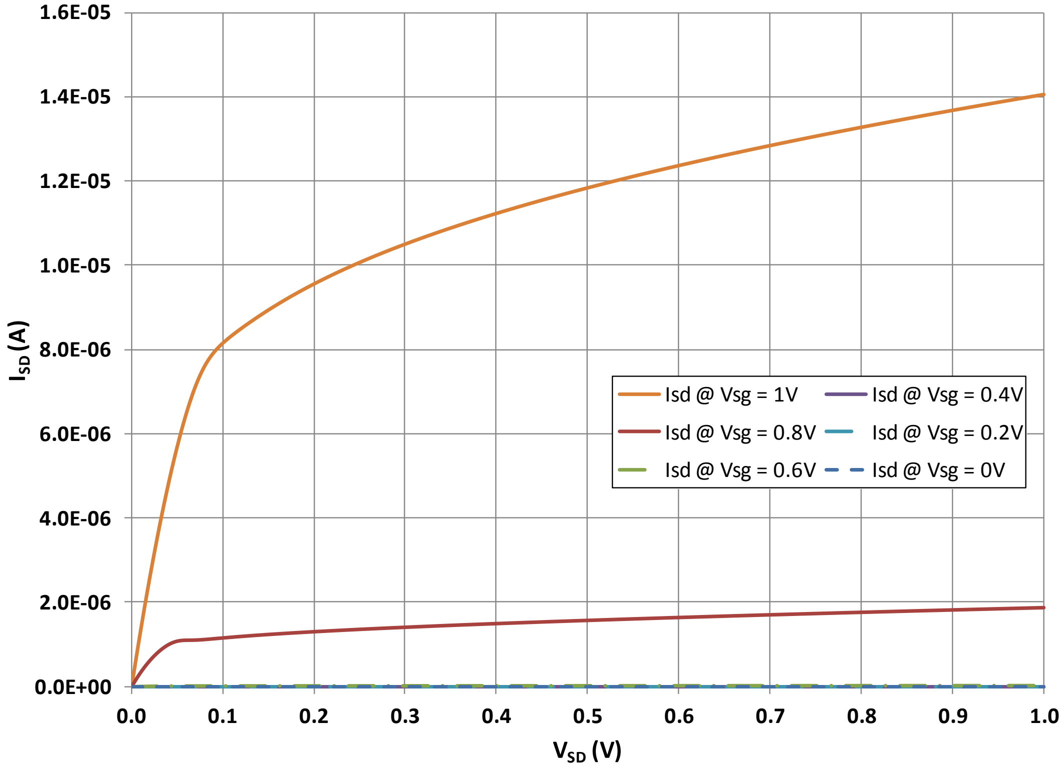


Figure 1.5. I-V curve of a p-type LP-mode FinFET with a 1 V back-gate bias

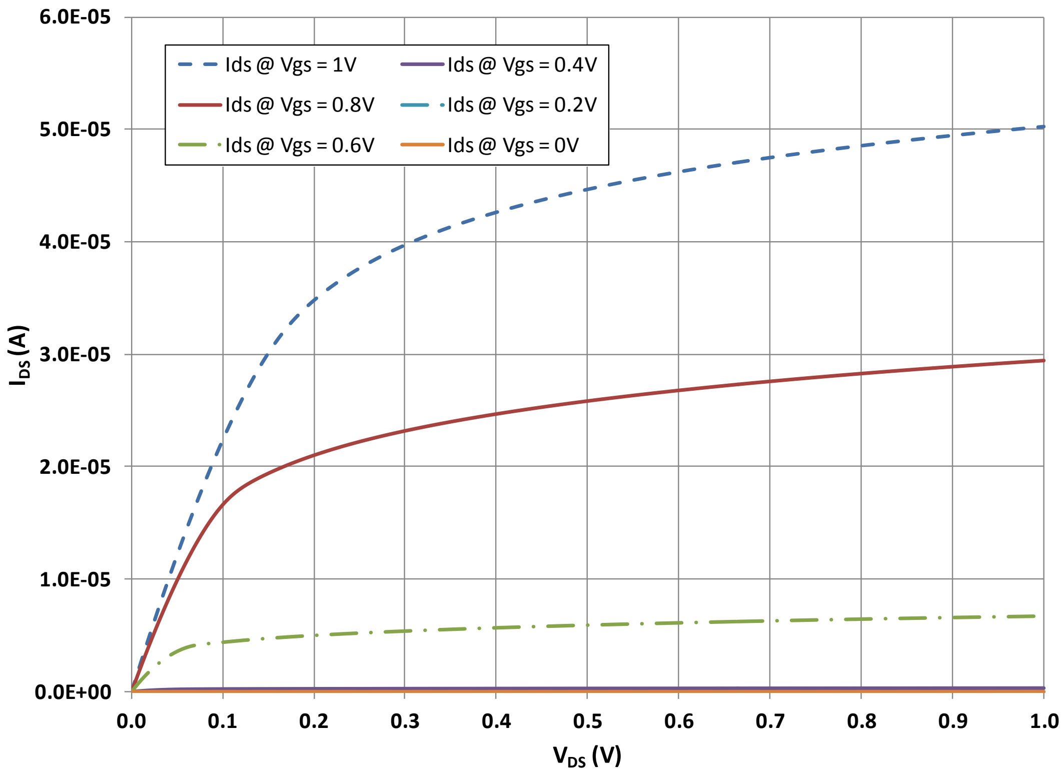


Figure 1.6. I-V curve of an n-Type LP-mode FinFET with a -0.2 V back-gate bias

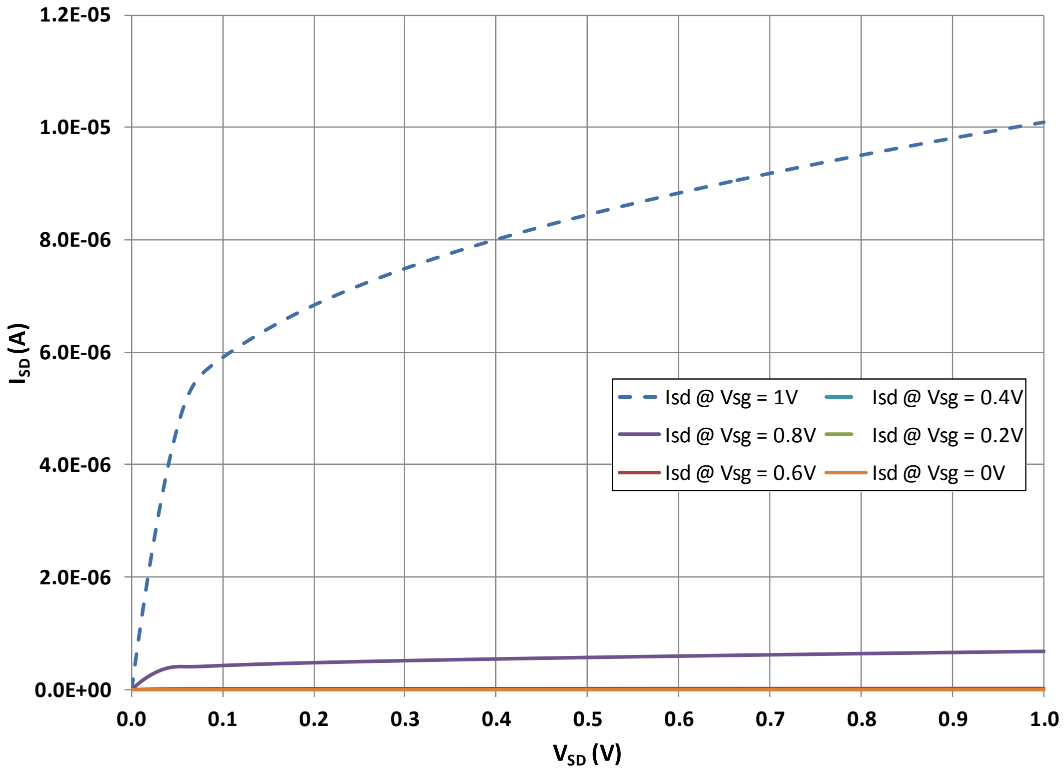


Figure 1.7. I-V curve of a p-type LP-mode FinFET with a 1.2 V (V_{DD}+0.2V) back-gate bias

Table 1.2 shows the I_{on} and I_{off} characteristics of single-fin n- and p-type FinFETs in LP-configuration. This table shows the current characteristics for varied values of back-gate biases, including biases which reverse-bias and forward-bias the back gate. A small forward-bias on the back gate enables a slight linear increase in I_{on} while I_{off} exponentially increases, however, a small reverse-bias on the back gate generates a slight linear decrease in I_{on} while I_{off} exponentially decreases—this creates a better I_{on}:I_{off} ratio.

Table 1.2. LP-mode FinFET I_{on} and I_{off} currents for various back-gate bias voltages

N-Type FinFET			P-Type FinFET		
Back-Gate Bias (V)	*Current*		*Back-Gate Bias (V)*	*Current*	
	I_{on} (μA)	*I_{off} (pA)*		*I_{on} (μA)*	*I_{off} (pA)*
-0.4	42.6	1	*1.4*	28.9	0.3
-0.3	46.3	3	*1.3*	31.7	1
-0.2	50.3	18	*1.2*	34.7	4
-0.1	54.2	72	*1.1*	37.7	15
0.0	58.2	387	*1.0*	40.8	78
0.1	62.3	2658	*0.9*	44.0	564
0.2	66.1	14824	*0.8*	47.1	3772
0.3	69.7	22667	*0.7*	49.6	8532

For more model characterization, the following figures show the current-voltage inverter characteristic curves of this particular FinFET technology sizing. Figure 1.8 and Figure 1.9 show the curves for SG-mode inverters for different input voltages and different n-type-to-p-type (n:p) ratios, respectively. Figure 1.10 and Figure 1.11 show the curves for different input voltages for LP-mode inverters with an n-bias of 0 V and a p-bias of 1 V and for LP-mode inverters with a n-bias of -0.2 V and a p-bias of 1.2 V, respectively, and Figure 1.12 and Figure 1.13 show the curves for different n:p ratios of these two LP-mode inverters. Figure 1.14 shows the character curves for LP-mode inverters using four combinations of back-gate biases.

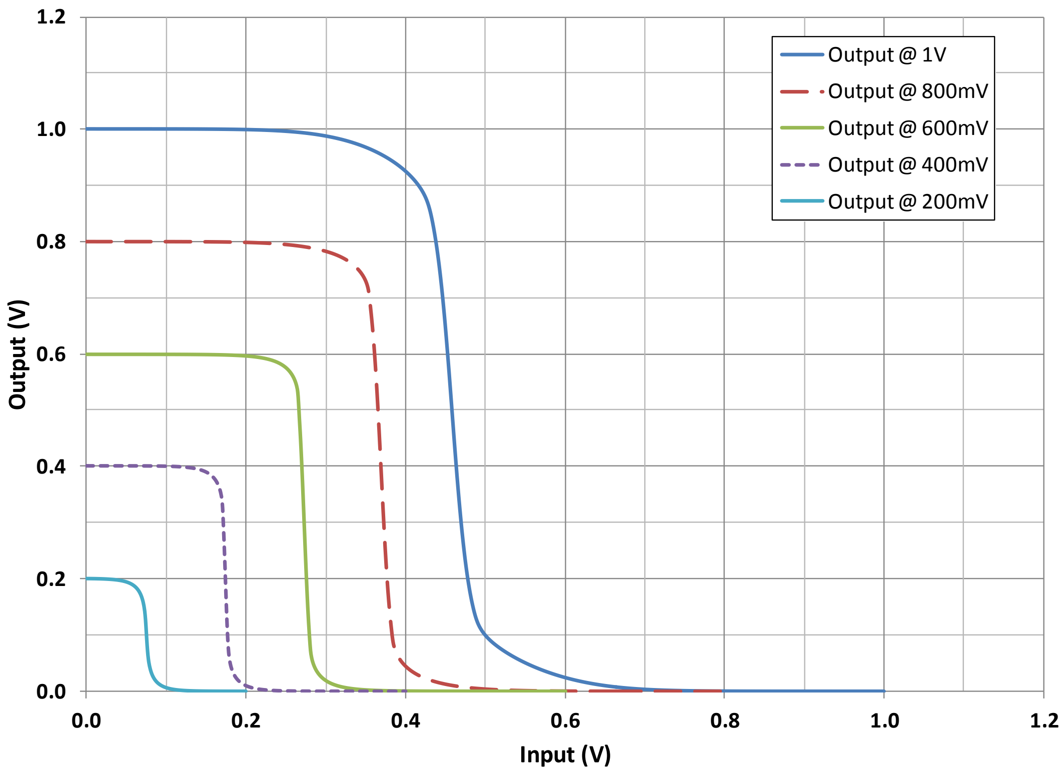


Figure 1.8. Characteristic curves of a SG-mode FinFET inverter at different input voltages

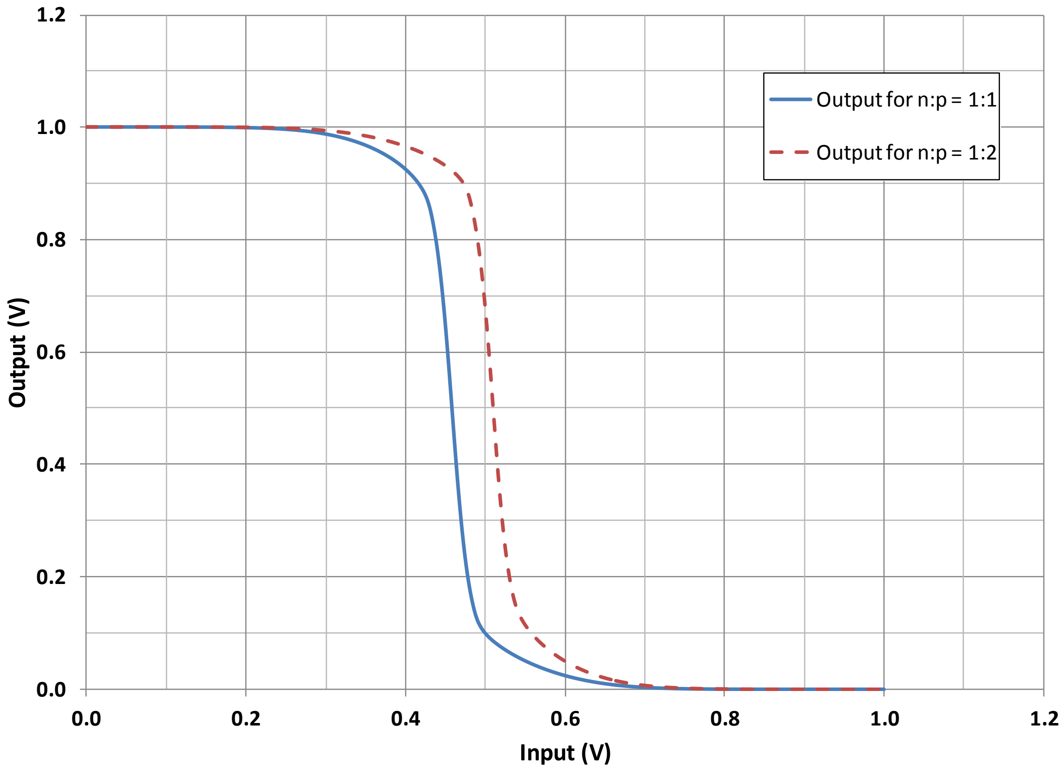


Figure 1.9. Characteristic curves of a SG-mode FinFET inverter at different n:p ratios

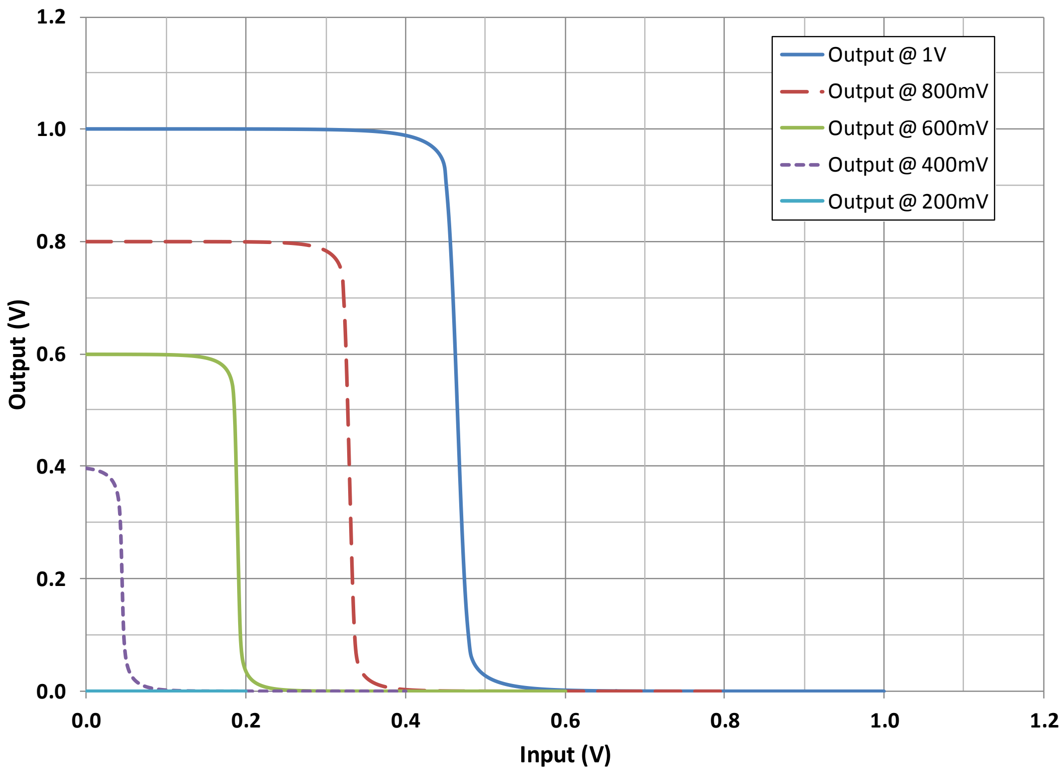


Figure 1.10. Characteristic curves of a LP-mode FinFET inverter using 0 V and 1 V biases

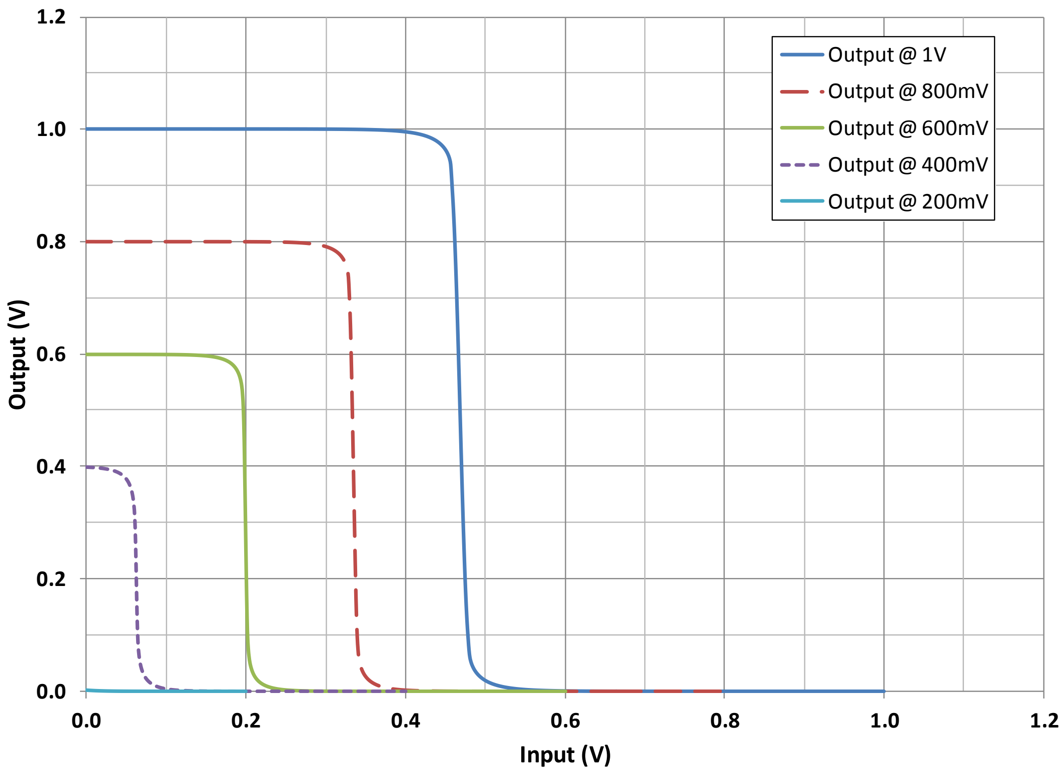


Figure 1.11. Characteristic curves of a LP-mode FinFET inverter using -0.2 V/1.2 V biases

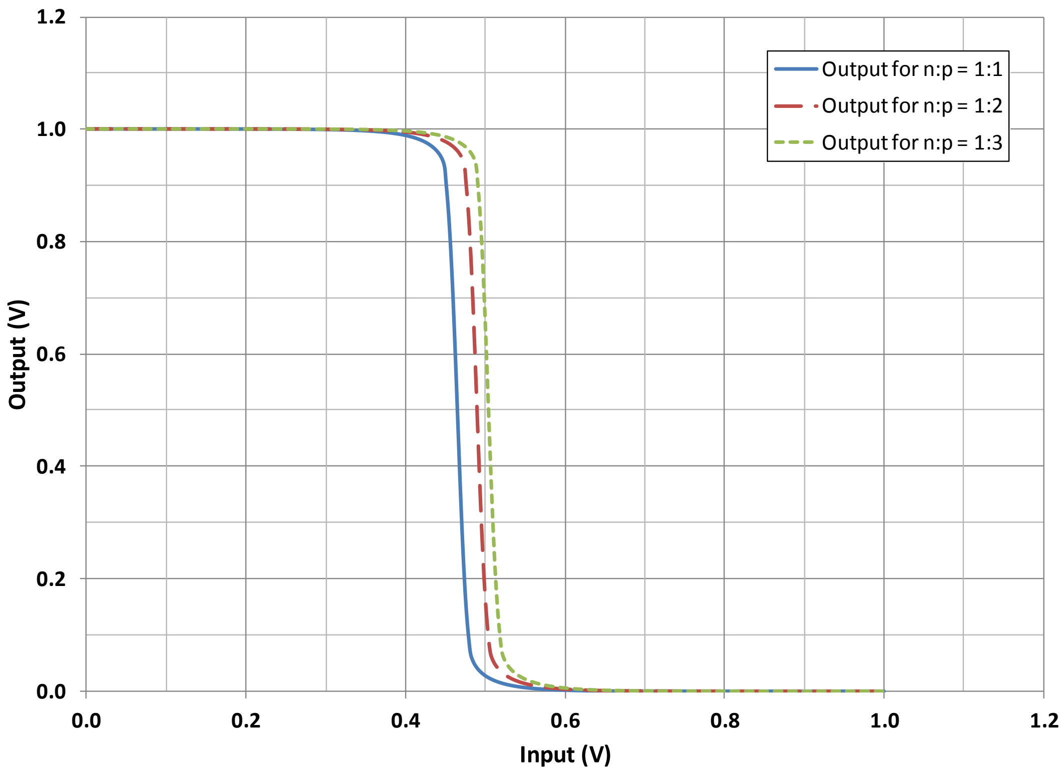


Figure 1.12. Curves of LP-mode FinFET inverter n:p ratios using 0 V/1 V biases

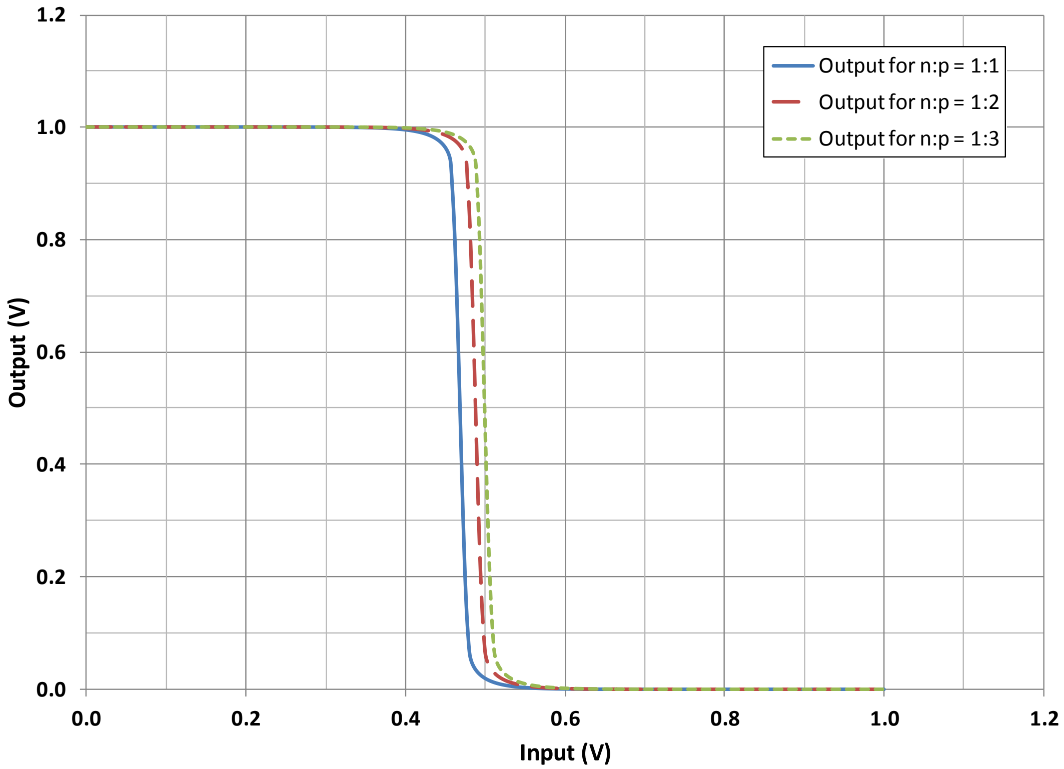


Figure 1.13. Curves of LP-mode FinFET inverter n:p ratios using -0.2 V/1.2 V biases

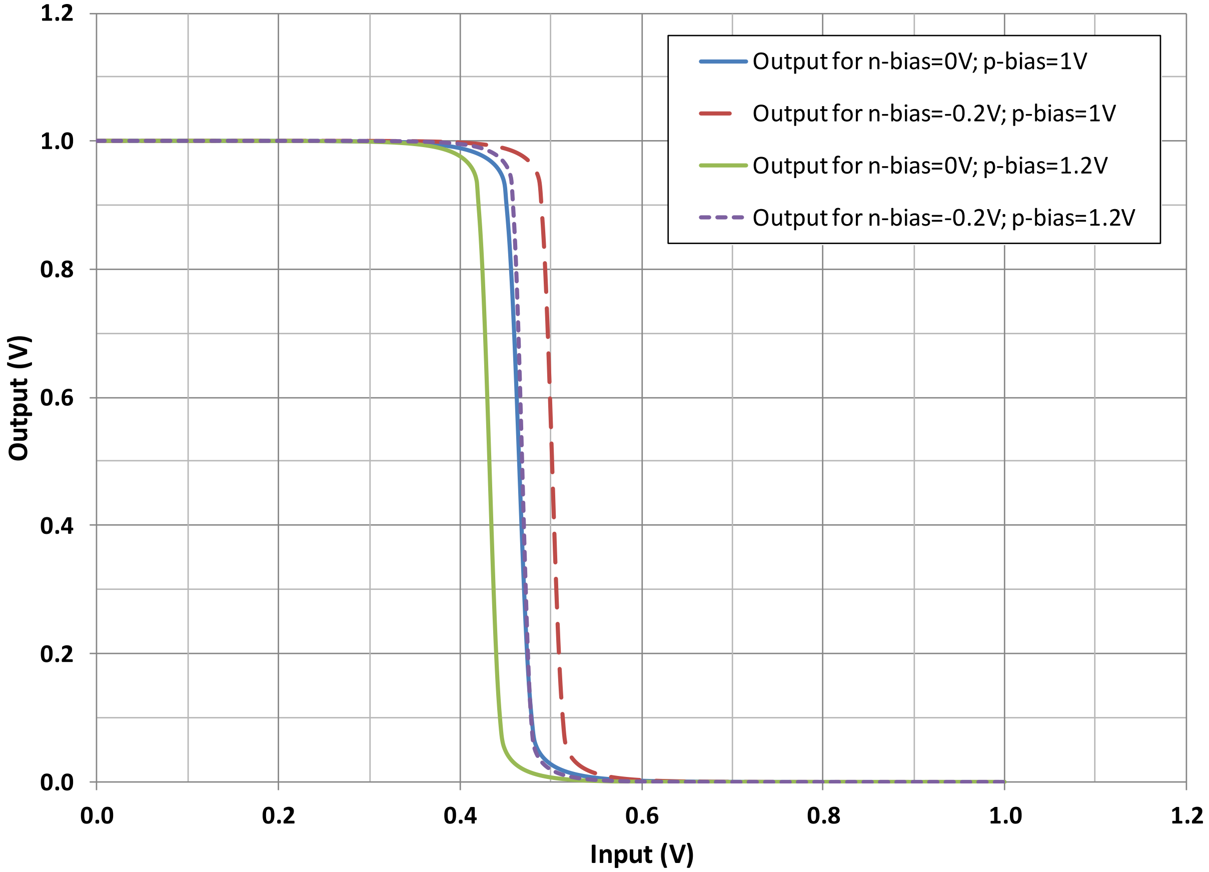


Figure 1.14. Characteristic curves of LP-mode FinFET inverter using various biases

In addition, as discussed in Section 1.1.1, back-gate biasing strategies can reduce FinFET leakage current and thus, SRAM power consumption. Table 1.3 and Table 1.4 show the leakage values for n- and p-type FinFETs, respectively, in SG and LP configurations. A single fin n-type FinFET has approximately 5X more leakage current than a single fin p-type FinFET; specifically 4.89X if in LP configuration with biases at -0.2 V for n-type and 1.2 V (V_{DD} + 0.2 V) for p-type, 4.95X if in LP configuration with biases at 0 V for n-type and 1 V VDD for p-type, and 5.53X if in SG configuration. For FinFETs with multiple fins, the leakage is directly proportional to the number of fins, e.g. compared to a single fin n-type SG FinFET, a FinFET with two fins has twice the amount of leakage (1 nA vs. 0.5 nA) and a FinFET with three fins has three times the amount of leakage (1.5 nA vs. 0.5 nA).

Table 1.3. N-type FinFET leakage current for SG and LP configurations (pA)

N-Type Config.	Number of Fins			Comp. vs. t
	1	*2*	*3*	
-	17.6	35.3	52.9	0.04
0	387.2	774.4	1161.7	0.77
t	501.4	1002.7	1504.1	--

Table 1.4. P-type FinFET leakage current for SG and LP configurations (pA)

P-Type Config.	Leakage (pA)	Comp. vs. t
+	3.6	0.04
vdd	78.2	0.86
t	90.6	--

1.1.3 Simulation Environment

In addition to the simulator, numerous scripts were written in Perl and tcl to aid the simulation and data gathering for this research. The primary scripts are described and included in Appendix A.

1.2 SRAM Memories

A block diagram of a typical memory sub-system is provided in Figure 1.15. In this figure, the address bits are A_{n-1}-A_0, the word-lines are d_{m-1}-d_0, and the bit-lines are D_{n-1}-D_0. The memory array is made up of memory cells. In this research, SRAM memories are studied, and so two common types of SRAM cells are analyzed: six-transistor (6T) and eight-transistor (8T) SRAM cells. These cells will be explained and studied in more depth throughout this dissertation. The memory array relies on a memory address decoder for its word-line inputs and input-output (I/O) buffers for its bit-line I/Os; these other circuits are the memory peripheral circuitry. Address decoders translate an address into the correct word-lines to be enabled or disabled. For more information about address decoders, Appendix B includes early research into memory address decoders using FinFETs and bulk-CMOS. I/O buffers drive values to be written onto the bit-lines for the memory address and read values from the bit-lines for a read operation. The later operation is more-specifically performed by bit-line sense-amplifiers. While this research did not examine FinFET bit-line sense-amplifiers, there is some research available about the application of FinFETs to this circuitry [13].

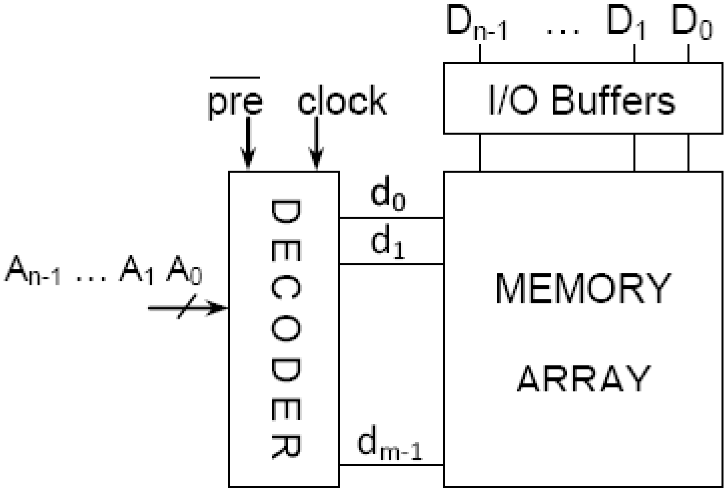


Figure 1.15. A block diagram view of a typical memory sub-system

Chapter 2

A Background into 6T SRAM Cells

The six-transistor (6T) static memory cell, shown in Figure 2.1, is widely accepted as the standard memory cell. It is designed to achieve fast read times with the inclusion of sense amplifiers. The standard 6T cell requires that a logic value and its inverse be placed on the bit lines during a write operation. The word line (WL) is raised to logic 1 and the logic levels on the bit lines are passed into the cross-coupled inverter pair. Reading from the memory cell entails pre-charging the bit lines and then asserting logic 1 on the word line. The complexity of this cell is in arriving at the appropriate device sizes for proper functionality. Device sizing for a CMOS-based cell is driven primarily by area and functional operation constraints; sizing must be carefully performed to maintain a stored value, enable the cell to push a stored value to the bit lines with reasonable speed for a read operation, and to correctly transfer and overwrite new logic values into the cell for a write operation.

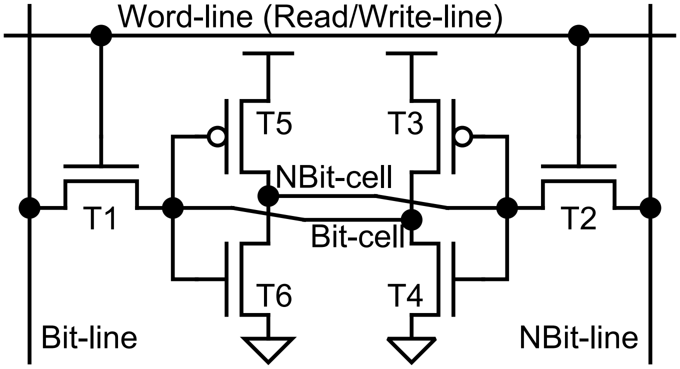


Figure 2.1. A 6T SRAM cell

2.1 6T FinFET SRAM Design Options

The 6T and eight-transistor (8T) SRAM cells examined in this paper are symmetrical, therefore if a 6T cell or an 8T cell's storage node and write path are to be symmetric and have similar performance for a stored "1" or "0," then each pair of transistors must have the same configuration: the Pass transistors (T1 and T2), n-type inverter (Inv-N) transistors (T4 and T6), and p-type inverter (Inv-P) transistors (T3 and T5). Table 2.1 presents all eight possible design options for a 6T FinFET cell, or the six-transistor storage node or write path for an 8T FinFET cell, when choosing between SG-mode or LP-mode for each type of transistor. This was used when choosing the 6T schemes to be studied, displayed in Table 2.2.

Table 2.1. FinFET SRAM design options for 6T cell or 8T storage node/write path

Type	**Transistor Configurations**		
	Pass Tran. T1, T2	***Inv. N-Tran. T4, T6***	***Inv. P-Tran. T3, T5***
I	SG	SG	SG
II	SG	SG	LP
III	SG	LP	SG
IV	SG	LP	LP
V	LP	SG	SG
VI	LP	SG	LP
VII	LP	LP	SG
VIII	LP	LP	LP

Table 2.2. FinFET 6T SRAM scheme configuration summary

Scheme	Storage Node Type	Pass Tran. T1, T2	Inv-N Tran. T4, T6	Inv-P Tran. T3, T5
SG	I	SG / 1	SG / 1	SG / 1
LP	VIII	LP / 1	LP / 1	LP / 1
LP_INV	IV	SG / 1	LP / 1	LP / 1
LP_INVN	III	SG / 1	LP / 1	SG / 1
LP_INVP	II	SG / 1	SG / 1	LP / 1
PGFB [15]	--	IG-C / 1	SG / 1	SG / 1
PUWG [15]	--	IG-C / 1	SG / 1	LP-WWL / 1
IG1 [14]	--	LP / 1	SG / 1	LP / 1
IG2 [14] [a]	--	SG / 1	SG / 1	LP / 1
Type5	V	LP / 2	SG / 1	SG / 1
Type6	VI	LP / 1	SG / 1	LP / 1
Type7	VII	LP / 2	LP / 2	SG / 1

* FinFET fin counts are labeled with the configuration mode, SG, LP, IG-C (independent gate mode with back gate tied to cell value), and LP-WWL (low-power using a write-word-line), for each transistor type

[a] IG2 [14] uses SG-mode p-type FinFETs for Pass transistors T1, T2

2.2 6T FinFET SRAM Design Schemes

Twelve 6T FinFET SRAM schemes (with back-gate bias variants) have been simulated. The 6T schemes examined in this paper are double-ended write, double-ended read, and have one read/write-line (word-line). Four schemes are from other research. The Pass Gate Feedback (PGFB) and Pull-Up Write Gating (PUWG) (with write word-line swing of 0V to V_{DD}=1V) schemes are from [15]. These two schemes connect the back gates of the pass transistors to the contents of the cell. The PUWG scheme also adds an additional write word-line which biases the back gates of the p-type inverter transistors to V_{DD} when it is active during a write operation. The Independent-Gate 1 (IG1) and Independent-Gate 2 (IG2) schemes are from [14]. The IG1 scheme is a Type VI SRAM scheme, but the IG2 scheme uses SG-mode p-type transistors for the pass-transistors T1 and T2. Therefore in addition, the word-line is generated to be active-low for this scheme. The scheme configurations and fin counts were chosen for the best energy-delay product (EDP), assuming 80% read and 20% write operations, for the 32×1024 array.

2.3 Simulation Setup

In addition to the loading from the other words on the Bit and NBit lines, 4X-sized inverters are added to simulate the input capacitance of a pipeline stage's flip-flop. The tri-state inverters of the write driver and the precharge transistors are 3X-sized. The read delay is the propagation delay from 50% of the read signal to when the Bit (or NBit) line discharges from V_{DD} to 50% of V_{DD}. This is a conservative estimate of the sense margin required for a sense-amplifier to sense the value of the bit for reading. The write delay is the propagation delay from 50% of the write signal (with the Bit and NBit lines already set with the write value) to when the cell's internal Bit and NBit nodes are within 10% of their final values; e.g. for a write "1" operation, the value of the cell's internal Bit node must be at least 90% of V_{DD} and the value of the cell's internal NBit node must be at most 10% of V_{DD}.

2.4 6T FinFET SRAM Performance Results

Table 2.3 and Table 2.4 present the performance results of the 6T FinFET SRAM schemes for the 16×16 and 32×1024 arrays, respectively. These results were obtained via simulation and contain performance data for the dynamic performance (read and write operations), leakage current, and noise margins which will be examined in the following subsections. The last subsection will summarize the performance of the 6T schemes as a whole.

Table 2.3. FinFET 6T SRAM 16×16 array simulation results summary

Scheme	Configuration^{+}			# Fins	Leakage / Cell (nA)	SNM (mV)	RSNM (mV)	16 bits × 16 words Array					
	Pass	Inv-N	Inv-P					Delay (ps)		Energy (fJ)			Ave* EDP (ps×fJ)
								Rd	*Wr*	*Rd*	*Wr*	*Ave**	
SG	t	t	t	6	0.59	360	124	4.5	5.6	*0.52*	*3.84*	1.19	6.59
LP	0	0	vdd	6	0.47	430	209	9.1	6.4	*0.24*	*2.25*	0.64	5.80
LP	0	0	+	6	0.39	414	190	9.1	6.1	*0.24*	*1.87*	0.56	5.08
LP	-	-	vdd	6	0.10	443	241	10.6	6.9	*0.25*	*2.34*	0.67	7.05
LP	-	-	+	6	0.02	442	222	10.6	6.7	*0.24*	*1.92*	0.58	6.14
LP_INV	t	0	vdd	6	0.47	430	29	6.3	3.4	*0.34*	*1.17*	0.50	3.19
LP_INV	t	0	+	6	0.39	414	45	6.3	3.5	*0.30*	*1.05*	0.45	2.84
LP_INV	t	-	vdd	6	0.10	443	23	7.0	3.3	*0.37*	*1.13*	0.52	3.64
LP_INV	t	-	+	6	0.02	442	37	7.0	3.5	*0.33*	*1.08*	0.48	3.35
LP_INVN	t	0	t	6	0.48	391	80	6.1	4.2	*0.68*	*2.46*	1.04	6.28
LP_INVN	t	-	t	6	0.11	378	81	6.7	4.1	*0.75*	*2.30*	1.06	7.07
LP_INVP	t	t	vdd	6	0.58	343	73	4.6	5.1	*0.28*	*1.54*	0.53	2.68
LP_INVP	t	t	+	6	0.51	325	56	4.6	5.2	*0.27*	*1.29*	0.47	2.45
PGFB [15]	IG-C	t	t	6	0.59	360	194	7.3	11.6	*0.42*	*10.17*	2.37	27.61
PUWG [15]	IG-C	t	wwl	6	0.50	302	78	7.3	7.6	*0.35*	*3.10*	0.90	6.80
IG1 [14]	0	t	vdd	6	0.58	343	207	8.1	9.4	*0.20*	*3.47*	0.85	7.97
IG2 [14]	p-t	t	vdd	6	0.58	343	99	6.6	5.8	*0.18*	*1.63*	0.47	3.10
Type5	2 x 0	t	t	8	0.59	360	200	6.1	7.4	*0.45*	*6.26*	1.61	11.95
Type6	0	t	+	6	0.51	325	188	8.1	9.0	*0.19*	*2.82*	0.71	6.42
Type7	2 x -	2 x -	t	10	0.13	398	272	6.1	8.2	*0.48*	*5.71*	1.52	12.51

* Average energy is calculated as 80% of the read energy plus 20% of the write energy. The average energy-delay product (EDP) is the product of the average energy and the delay.

$^{+}$ Config Key: **t** = tied/SG; **0** = LP: 0 V bias; **-** = LP: -0.2 V bias; **vdd** = LP: V_{DD}=1 V bias; **+** = LP: 1.2 V bias; **IG-C** = independent gate: back gate tied to internal cell net; **wwl** = write word line with 0 to V_{DD}=1 V swing; **p-t** = p-type FinFET in SG; **2 x -** = 2 fins per tran. LP: 0V bias

Table 2.4. FinFET 6T SRAM 32×1024 array simulation results summary

Scheme	Configuration+			# Fins	Leakage / Cell (nA)	SNM (mV)	RSNM (mV)	32 bits × 1024 words Array					
	Pass	Inv-N	Inv-P					Delay (ps)		Energy (fJ)			Ave* EDP (ps×fJ)
								Rd	*Wr*	*Rd*	*Wr*	*Ave**	
SG	t	t	t	6	0.59	360	124	32	5	*2.4*	*9.3*	3.8	120
LP	0	0	vdd	6	0.47	430	209	71	6	*1.6*	*5.6*	2.4	170
LP	0	0	+	6	0.39	414	190	71	5	*1.4*	*4.7*	2.1	146
LP	-	-	vdd	6	0.10	443	241	84	6	*0.8*	*5*	1.6	135
LP	-	-	+	6	0.02	442	222	84	6	*0.5*	*3.9*	1.2	103
LP_INV	t	0	vdd	6	0.47	430	29	49	3	*2.3*	*3.5*	2.5	123
LP_INV	t	0	+	6	0.39	414	45	49	3	*2.1*	*3.1*	2.3	113
LP_INV	t	-	vdd	6	0.10	443	23	56	3	*1.3*	*2.9*	1.6	89
LP_INV	t	-	+	6	0.02	442	37	56	3	*1.1*	*2.4*	1.3	75
LP_INVN	t	0	t	6	0.48	391	80	48	3	*3*	*6.5*	3.7	181
LP_INVN	t	-	t	6	0.11	378	81	55	3	*2.2*	*5.5*	2.9	157
LP_INVP	t	t	vdd	6	0.58	343	73	32	5	*1.9*	*4.4*	2.4	76
LP_INVP	t	t	+	6	0.51	325	56	32	5	*1.8*	*3.8*	2.2	69
PGFB [15]	IG-C	t	t	6	0.59	360	194	55	8	*2*	*16.2*	4.8	267
PUWG [15]	IG-C	t	wwl	6	0.50	302	78	55	7	*1.7*	*7.8*	2.9	161
IG1 [14]	0	t	vdd	6	0.58	343	207	62	9	*1.6*	*8.7*	3	188
IG2 [14]	p-t	t	vdd	6	0.58	343	99	54	5	*1.9*	*5.5*	2.7	144
Type5	2 x 0	t	t	8	0.59	360	200	77	6	*2.8*	*13.4*	4.9	375
Type6	0	t	+	6	0.51	325	188	62	9	*1.5*	*7.3*	2.6	163
Type7	2 x -	2 x -	t	10	0.13	398	272	79	7	*1.3*	*11.7*	3.4	267

* Average energy is calculated as 80% of the read energy plus 20% of the write energy. The average energy-delay product (EDP) is the product of the average energy and the delay.

[+] Config Key: **t** = tied/SG; **0** = LP: 0 V bias; **-** = LP: -0.2 V bias; **vdd** = LP: V_{DD}=1 V bias;
+ = LP: 1.2 V bias; **IG-C** = independent gate: back gate tied to internal cell net;
wwl = write word line with 0 to V_{DD}=1 V swing; **p-t** = p-type FinFET in SG;
2 x - = 2 fins per tran. LP: 0V bias

2.4.1 6T FinFET SRAM Dynamic Performance

Of the twelve simulated 6T SRAM schemes, the LP_INVP scheme has the lowest delay, 5.1 and 5.2 ps for the 16×16 array and 32 ps for the 32×1024 array, and lowest average EDP, 2.68 and 2.45 ps×fJ for the 16×16 array and 76 and 69 ps×fJ for the 32×1024 array, for both array sizes. LP_INVP is up to 8.9% faster than the SG scheme (5.6 ps) for the 16×16 array and is equally as fast for the 32×1024 array. The average EDP of the SG scheme is as much as 2.6X and 1.7X larger than that of LP_INVP for the 16×16 and 32×1024 array sizes, respectively. LP_INVP uses SG configuration for the pass transistors and n-type inverter FinFETs to achieve a faster read time, similar to the SG scheme. For LP_INVP, using an additional voltage of 1.2 V to reverse-bias the p-type inverter FinFETs provides additional EDP savings compared to using the 1 V V_{DD}, however, this additional voltage must be routed to each SRAM cell. The LP_INV scheme also performs well, but is slower than the LP_INVP scheme by 34-75% (LP_INV's delay is as high as 7.0 ps for the 16×16 array and 56 ps for the 32×1024 array) and does not provide any savings in terms of fewer additional voltages required. The Type5, Type7, and PGFB [15] schemes have the highest average EDPs. For both array sizes, these schemes have the highest write energies which cause them to have the highest average energies, and these schemes also have among the highest delays. For the 32×1024 array, the average EDPs of these schemes are 3.1X (375 ps×fJ for Type5) and 2.2X (267 ps×fJ for Type7 and PGFB [15]) greater than the SG scheme's EDP of 120 ps×fJ.

2.4.2 6T FinFET SRAM Leakage Current

There are four major leakage currents in the 6T SRAM as shown in Figure 2.2. The most commonly reported are the inverters' leakage currents, as shown in Table 2.3 and Table 2.4. These currents flow through transistors T3 and T4 for inverter 1 (in Figure 2.2, T4 is off but subthreshold current flows through the transistor). Inverter 2 formed by transistors T5 and T6 has T5 off. These two leakage currents are usually referred as static current (or static power). In this paper these two currents are called I_{inv1} and I_{inv2}. In addition to the leakage of the memory cell inverters, leakage is present when a write or read operation occurs to a different SRAM cell in the column. Memory cells that are not selected for a write or read operation draw current from (or to) the drivers. Figure 2.2 shows a SRAM cell with a “0” stored (inverter 1 and 2 output 1 and 0, respectively). If a “1” is being written (Bit and NBit are 1 and 0, respectively) there is a leakage current from the driver of the Bit line through transistor T1; this leakage current is called I_{T1}. There is another leakage current from the cell to the driver of the NBit line through T2; this leakage current is called I_{T2}. When a read occurs and the cell is not selected, at precharge time both Bit and NBit lines are set to “1.” Using the example of Figure 2.2, current I_{T1} is the same as when there is a write while there is a much smaller current through transistor T2.

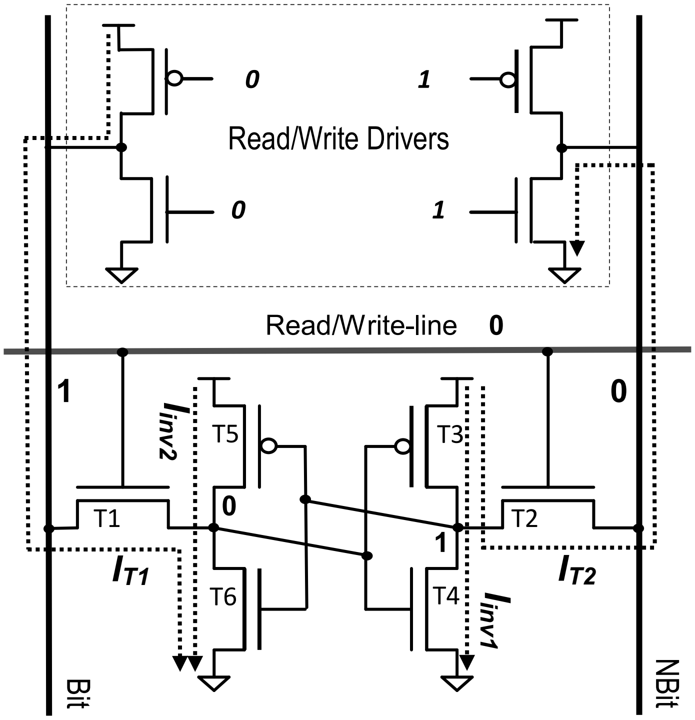


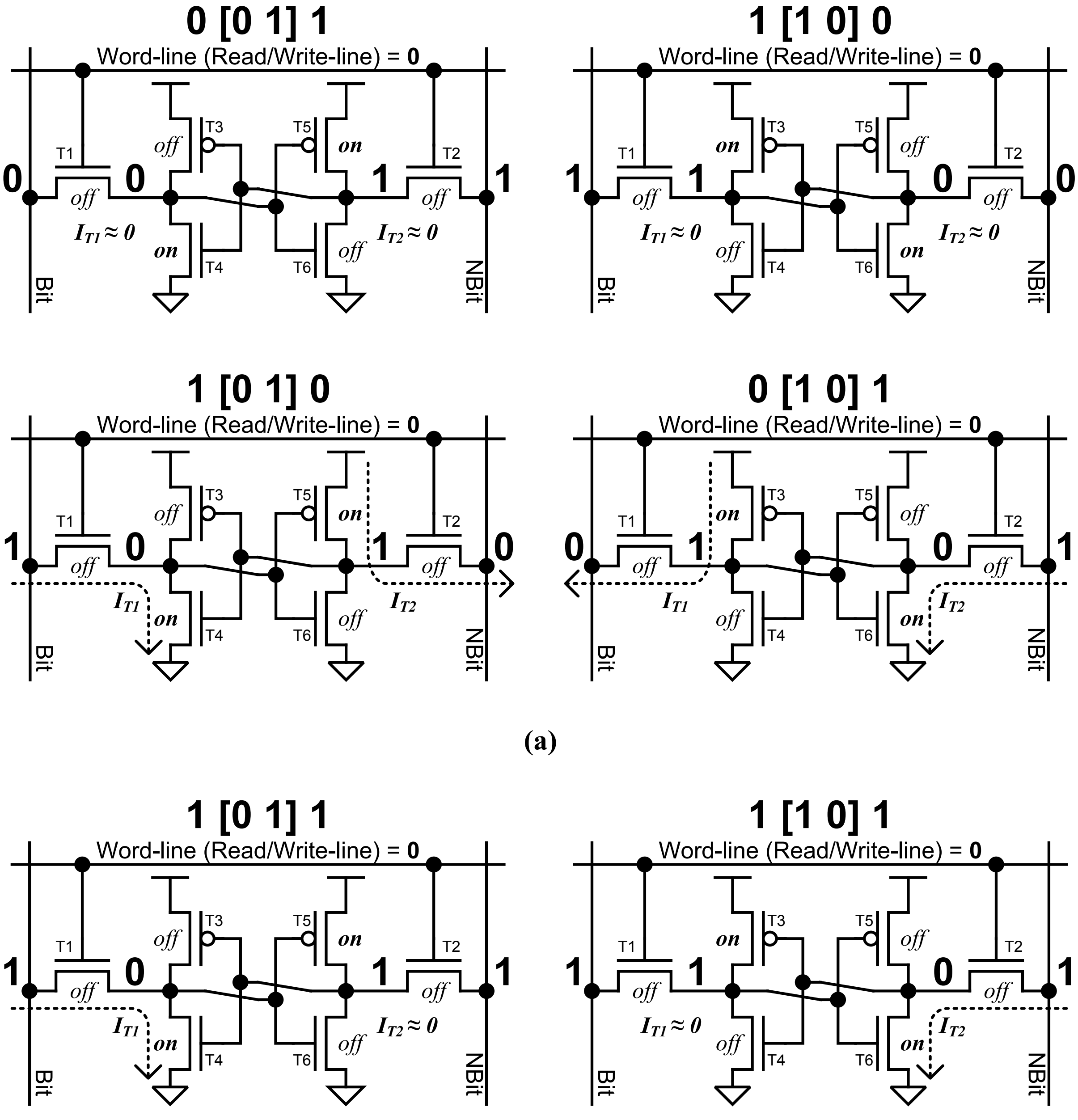
Figure 2.2. 6T SRAM cell leakage currents under a write operation to another cell in the column

Table 2.5 shows the leakage currents for the read and write operations for the 6T SG scheme. This table diagrams the status of the cell’s value and the Bit and NBit values using the notation: **Bit [Inv2_output Inv1_output] NBit**. The example shown in Figure 2.2 is the status

1 [0 1] 0. There is much symmetry in the leakage currents for each 6T scheme. There are three distinct leakage cases: write operation with the bitlines and cell having the same values (0 [0 1] 1 and 1 [1 0] 0), write operation with the bitlines and the cell having opposite values (0 [1 0] 1 and 1 [0 1] 0), and the read operation (1 [0 1] 1 and 1 [1 0] 1); each of these cases are diagramed in Figure 2.3. The I_{T1} and I_{T2} leakage currents are maximized when the value in the cell is opposite that of the Bit or NBit line and this also maximizes the total leakage. The total leakage current for the three cases are presented in Table 2.6 for each 6T scheme.

Table 2.5. FinFET SG 6T scheme simulation results of read/write leakage (nA)

Leakage Current	**Write Condition**				**Read Condition**	
	0 [0 1] 1	***1 [0 1] 0***	***0 [1 0] 1***	***1 [1 0] 0***	***1 [0 1] 1***	***1 [1 0] 1***
I_{inv1}	0.50	0.50	0.09	0.09	0.50	0.09
I_{inv2}	0.09	0.09	0.50	0.50	0.09	0.50
I_{T1}	0.00	0.50	0.50	0.00	0.50	0.00
I_{T2}	0.00	0.50	0.50	0.00	0.00	0.50
$\mathbf{I_{TOTAL}}$	**0.59**	**1.59**	**1.59**	**0.59**	**1.09**	**1.09**



(a)

(b)

Figure 2.3. 6T SRAM cell write (a) and read (b) leakage currents

Table 2.6. FinFET 6T SRAM simulation results of read/write leakage totals (nA)

Scheme	**Configuration**			**# Fins**	***Write 0 [0 1] 1***	***Write 1 [0 1] 0***	***Read 1 [0 1] 1***
	Pass	**Inv-N**	**Inv-P**				
SG	t	t	t	6	0.59	1.59	1.09
LP	0	0	vdd	6	0.47	1.24	0.85
LP	0	0	+	6	0.39	1.17	0.78
LP	-	-	vdd	6	0.10	0.13	0.11
LP	-	-	+	6	0.02	0.06	0.04
LP_INV	t	0	vdd	6	0.47	1.47	0.97
LP_INV	t	0	+	6	0.39	1.39	0.89
LP_INV	t	-	vdd	6	0.10	1.10	0.60
LP_INV	t	-	+	6	0.02	1.02	0.52
LP_INVN	t	0	t	6	0.48	1.48	0.98
LP_INVN	t	-	t	6	0.11	1.11	0.61
LP_INVP	t	t	vdd	6	0.58	1.58	1.08
LP_INVP	t	t	+	6	0.51	1.51	1.01
PGFB [15]	IG-C	t	t	6	0.59	0.98	0.98
PUWG [15]	IG-C	t	wwl	6	0.50	0.89	0.89
IG1 [14]	0	t	vdd	6	0.58	1.35	0.97
IG2 [14]	p-t	t	vdd	6	0.58	0.76	0.67
Type5	2 x 0	t	t	8	0.59	2.14	1.37
Type6	0	t	+	6	0.51	1.28	0.89
Type7	2 x -	2 x -	t	10	0.13	0.20	0.16

The schemes with cross-coupled transistors T3-T6 in LP configuration have the lowest leakage current. Due to this, the LP and LP_INV schemes have the lowest leakage of 0.02 nA; the 0.59 nA leakage of the SG scheme is 29.5X higher. In addition, read and write leakage current is minimized when T1 and T2 are in LP configuration and minimum-sized. Because of this, the LP scheme has the lowest read and write leakage current with its average write leakage and read leakage both equaling 0.04 nA. This is 27X less than the 1.09 nA average write leakage and read leakage of the SG scheme. Type7 also has low write and read leakage, 0.17 nA and

0.16 nA respectively, and both values are at least 6.4X less than the SG scheme due to reverse-biasing the pass transistors T1 and T2.

2.4.3 6T FinFET SRAM Noise Margins

Static noise margins (SNM) and read static noise margins (RSNM) of the 6T SRAM schemes are presented in Table 2.3 and Table 2.4. The RSNM is measured similar to the SNM, but with the word-line active and the pass-transistors of the 6T cells (T1 and T2) attempting to discharge one of the Bit/NBit lines for a read operation. The SNMs and RSNMs were measured using the "maximum squares" simulation method presented in [16]. Each cell's characteristic curves, also referred to as the butterfly-curve, for the write static noise margin were also analyzed to ensure no write errors exist [17].

The SNM depends on the configuration of the cross-coupled inverter transistors T3-T6. The LP and LP_INV schemes have the greatest SNMs of over 440 mV for configurations which reverse-bias the back gates of T4 and T6, the Inv-N transistors. These SNMs are 22% larger than the SG scheme's SNM of 360 mV. The Inv-P transistors (T3 and T5) must be strong relative to the Inv-N transistors (T4 and T6) in order to overcome leakage current from T4 and T6 [18]; this can be overdone however, as LP_INVN scheme has a lower SNM than these three schemes. A proper ratio must be in place for the Inv-P transistors (T3 and T5) to overcome leakage from the Inv-N transistors (T4 and T6) and vice versa. The worst-performing schemes, PUWG [15], LP_INVP, and Type6, use SG configuration for T4 and T6 while using LP configuration for T3 and T5, leading to a low SNM. PUWG [15] has a SNM of 302 mV which is 16% less than the SG scheme and LP_INVP and Type6 have SNMs of 325 mV which are 9.7% less than the SG scheme.

The RSNM depends on the configuration of the cross-coupled inverters' transistors (T3-T6) and the configuration of the Pass transistors (T1 and T2). For a high RSNM, the Pass transistors should be configured to be weaker than the cross-coupled transistors and additionally, as for the SNM, T3 and T5 should also be made strong relative to T4 and T6. Type7 and the LP schemes do this and have the highest RSNMs of 272 mV and up to 241 mV, respectively. These RSNMs are 2.1X and 1.9X larger than the SG scheme's RSNM of 124 mV. Schemes which use SG configuration for the Pass transistors and LP configuration for the cross-coupled inverters have lower RSNMs. In particular, the LP_INV, LP_INVP, PUWG [15], LP_INVN, and IG2 [14] schemes do this and have RSNMs lower than the SG scheme. The lowest RSNM of the LP_INV scheme, 23 mV, is only 18% of the SG scheme's RSNM. IG2 [14] has a larger RSNM of 99 mV, but this is only 79% of the SG scheme's RSNM. Since the RSNMs of these schemes are low, an accidental overwrite during a read operation could occur and this is a serious detriment to using these schemes in a 6T SRAM array.

2.4.4 6T FinFET SRAM Overall Performance Summary

Compared to the 6T SG scheme, the LP_INV, LP_INVN, LP_INVP, PUWG [15], and IG2 [14] schemes have smaller RSNMs; therefore, these schemes are error-prone during a read operation. The PGFB [15], IG1 [14], Type5, Type6, and Type7 schemes have a larger average EDP than the SG scheme for the 32×1024 array. In fact, the 6T LP scheme, with a -0.2 V back gate (BG) bias for the Pass and Inv-N transistors and a 1.2 V BG-bias for the Inv-P transistors, is the only 6T scheme that has a lower EDP than the 6T SG scheme for the larger 32×1024 6T SRAM array and possesses a RSNM larger than the 6T SG scheme. Of the 6T SRAM cells, only this LP cell and the SG cell will be examined further in this dissertation.

Chapter 3

8T SRAM Cells

An eight-transistor (8T) memory cell is shown in Figure 3.1. This 8T SRAM cell has a similar structure as a 6T cell with two additional transistors that decouple read and write operations. The read operation is performed by setting the read-word line to logic 1; the additional transistors (T7 and T8) discharge the RBit line that has been precharged before the Read-line is set. The 8T basic cell provides a more orthogonal design; read and write operations are performed by different transistors. Since the read operation does not affect the contents of the cell (the two back to back inverters), the worst-case static noise margin is simply that for two cross-coupled inverters. For a 6T cell, on the other hand, the worst-case static noise margin occurs in the read condition.

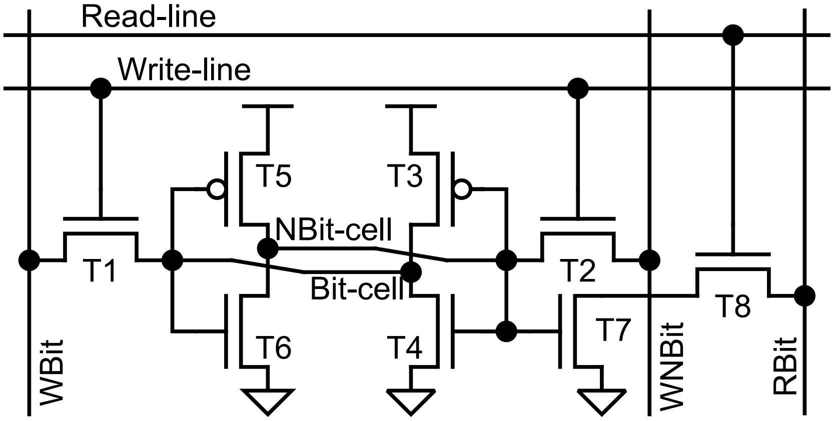


Figure 3.1. An 8T SRAM cell

3.1 8T FinFET SRAM Design Options

The 6T portion of the 8T cell, the storage node and write path, can be designed similar to a 6T cell; Table 2.1 highlights the design options for transistors T1-T6. In addition to the 6T portion of the cell, an 8T SRAM cell has a dedicated read port comprised of transistors T7 and T8 (the Read transistors). Table 3.1 presents all four possible design options for the read path for an 8T FinFET cell when choosing between shorted-gate (SG) mode or low-power (LP) mode for each transistor. The choice of 8T read path design was first analyzed in [19], and Table 3.2 and Table 3.3 present the read and write delays, energies, and energy-delay-products (EDPs) for 16-bit by 16-word (16×16) and 32-bit by 1024-word (32×1024) SRAM arrays, respectively. These 8T SRAM arrays use minimum-sized cells with T1-T6 in SG-mode.

Table 3.1. FinFET SRAM design options for 8T read path

Config. Type	**Transistor Configurations**	
	Read Tran. T7 (Cell Data Value at Gate)	***Read Tran. T8 (Read Line at Gate)***
SG	SG	SG
Read Line Biased	SG	LP
Data Biased	LP	SG
LP	LP	LP

Table 3.2. Results of read path design options for 8T FinFET SRAM 16×16 array

Config. Type	**Config.**		**16 bits × 16 words Array**					
	T7 (Cell Data)	**T8 (Read Line)**	**Read Delay (ps)**	**Read Energy (fJ)**	**Read EDP (fJ×ps)**	**Write Delay (ps)**	**Write Energy (fJ)**	**Write EDP (fJ×ps)**
SG	t	t	4.4	0.24	1.06	6.3	4.44	27.97
Read Line Biased	t	0	7.7	0.06	0.46	6.3	4.90	30.87
Read Line Biased	t	-	8.8	0.05	0.44	6.3	4.98	31.37
Data Biased	0	t	6.3	0.08	0.50	6.0	4.40	26.40
Data Biased	-	t	7.0	0.09	0.63	6.0	4.57	27.42
LP	0	0	8.8	0.04	0.35	6.0	4.81	28.86
LP	0	-	9.8	0.04	0.39	6.0	4.87	29.22
LP	-	0	9.2	0.05	0.46	6.0	4.82	28.92
LP	-	-	10.3	0.04	0.41	6.0	4.88	29.28

Table 3.3. Results of read path design options for 8T FinFET SRAM 32×1024 array

Config. Type	Config.		32 bits × 1024 words Array					
	T7 (Cell Data)	T8 (Read Line)	Read Delay (ps)	Read Energy (fJ)	Read EDP (fJ×ps)	Write Delay (ps)	Write Energy (fJ)	Write EDP (fJ×ps)
SG	t	t	32	0.9	28.8	6	10.6	63.6
Read Line Biased	t	0	61	1.3	79.3	6	11.0	66.0
Read Line Biased	t	-	71	1.5	106.5	6	11.2	67.2
Data Biased	0	t	48	1.1	52.8	5	10.1	50.5
Data Biased	-	t	55	1.3	71.5	5	10.1	50.5
LP	0	0	70	1.5	105.0	5	10.5	52.5
LP	0	-	80	1.6	128.0	5	10.7	53.5
LP	-	0	74	1.5	111.0	5	10.5	52.5
LP	-	-	84	1.7	142.8	5	10.7	53.5

The two series transistors of the read path are the only transistors involved in the read operation, thus the transistor with a lower current drive capability will limit the read speed by becoming a bottleneck when discharging the RBit line. For optimal read speed, and optimal read EDP for larger array sizes, both of these transistors should be equally-sized and identically configured in SG-mode.

The swing of the read-line can also be adjusted to obtain faster read times. Table 3.4 shows percent comparisons between normal (0 V to V_{DD} = 1 V) read line swing to larger read-line swings for 16×16 and 32×1024 SG-mode SRAM arrays. These 8T SRAM arrays use minimum-sized cells with all transistors in SG-mode. Read delay decreases by 17 to 18% when the read-line is at 1.2 V, or V_{DD} + 0.2 V, when active. This increases read energy significantly, by 43%, for the 16×16 array, but much less, by 3 to 4%, for the 32×1024 array. More energy is required to generate a higher read-line swing, but this can reduce the read speed and read EDP of a large SRAM array.

Table 3.4. Comparisons of normal to larger read-line swings for 8T FinFET SRAM arrays

Read-Line Swing	16 bits × 16 words Array		32 bits × 1024 words Array	
	Read Delay (%)	Read Energy (%)	Read Delay (%)	Read Energy (%)
0V to 1.2V	-17	43	-18	3
-0.2V to 1V	1	-2	1	1
-0.2V to 1.2V	-17	43	-17	4

3.2 8T FinFET SRAM Design Schemes

Nine 8T FinFET SRAM schemes (with back-gate bias and read-line swing variants) have been simulated. The configurations of these schemes are displayed in Table 3.5. The 8T schemes examined in this paper are double-ended write, single-ended read, and have one write-line and one read-line. As with the 6T cells, the scheme configurations and fin counts were chosen for the best energy-delay product, assuming 80% read and 20% write operations, for the 32×1024 array.

Table 3.5. FinFET 8T SRAM scheme configuration summary

Scheme	Storage Node Type	Pass Tran. T1, T2	Inv-N Tran. T4, T6	Inv-P Tran. T3, T5	Read Tran. T7, T8
SG	I	SG / 3	SG / 1	SG / 1	SG / 1
LP	VIII	LP / 3	LP / 1	LP / 1	LP / 1
LP_SGR	VIII	LP / 1	LP / 1	LP / 1	SG / 1
LP_INV	IV	SG / 1	LP / 1	LP / 1	SG / 1
LP_INVN	III	SG / 1	LP / 1	SG / 1	SG / 1
LP_INVP	II	SG / 3	SG / 1	LP / 1	SG / 1
Type5	V	LP / 3	SG / 1	SG / 1	SG / 1
Type6	VI	LP / 3	SG / 1	LP / 1	SG / 1
Type7	VII	LP / 3	LP / 1	SG / 1	SG / 1

*FinFET fin counts are labeled with the configuration mode (SG or LP) for each transistor type

3.3 Simulation Setup

The simulation setup for the 8T cells is similar to the setup for the 6T cells. In addition to the loading from the other words on the WBit, WNBit, and RBit lines, 4X-sized inverters are added to simulate the input capacitance of a pipeline stage's flip-flop. The tri-state inverters of the write driver and the precharge transistors are 3X-sized. The read delay is the propagation delay from 50% of the read signal to when the RBit line discharges from V_{DD} to 50% of V_{DD}. This is a conservative estimate of the sense margin required for a sense-amplifier to sense the value of the bit for reading. The write delay is the propagation delay from 50% of the write signal (with the WBit and WNBit lines already set with the write value) to when the cell's internal Bit and NBit nodes are within 10% of their final values; e.g. for a write "1" operation, the value of the cell's internal Bit node must be at least 90% of V_{DD} and the value of the cell's internal NBit node must be at most 10% of V_{DD}.

3.4 8T FinFET SRAM Performance Results

Table 3.6 and Table 3.7 present the performance results of the 8T FinFET SRAM schemes for the 16×16 and 32×1024 arrays, respectively. These results were obtained via simulation and contain performance data for the read operation, write operation, leakage current, and noise margins which will be examined in the following subsections. The last subsection will summarize the performance of the 8T schemes as a whole.

Table 3.6. FinFET 8T SRAM 16×16 array simulation results summary

Scheme	**Configuration[+]**				**# Fins**	**Leakage / Cell (nA)**	**SNM (mV)**	**16 bits × 16 words Array**					
	Pass	**Inv-N**	**Inv-P**	**Read**				**Delay (ps)**		**Energy (fJ)**			**Ave* EDP (ps×fJ)**
								Rd	***Wr***	***Rd***	***Wr***	***Ave****	
SG	3 x t	t	t	t	12	0.59	360	4.4	4.1	*0.11*	*2.62*	0.61	2.71
SG	3 x t	t	t	t_{0+}	12	0.59	360	3.7	4.1	*0.27*	*2.49*	0.71	2.91
SG	3 x t	t	t	t_{-vdd}	12	0.59	360	4.5	4.1	*0.11*	*2.63*	0.61	2.73
SG	3 x t	t	t	t_{-+}	12	0.59	360	3.7	4.1	*0.27*	*2.49*	0.71	2.90
LP	3 x 0	0	vdd	0	12	0.47	430	8.8	5.3	*0.04*	*1.98*	0.43	3.78
LP	3 x 0	0	+	0	12	0.39	414	8.8	5.5	*0.04*	*1.73*	0.38	3.34
LP	3 x -	-	vdd	-	12	0.10	443	10.2	5.3	*0.04*	*2.04*	0.44	4.52
LP	3 x -	-	+	-	12	0.02	442	10.2	5.5	*0.04*	*1.78*	0.39	3.98
LP_SGR	0	0	vdd	t	8	0.47	430	4.4	8.4	*0.27*	*2.59*	0.73	6.17
LP_SGR	0	0	+	t	8	0.39	414	4.4	8.4	*0.27*	*2.21*	0.66	5.52
LP_SGR	-	-	vdd	t	8	0.10	443	4.4	9.0	*0.28*	*2.64*	0.75	6.72
LP_SGR	-	-	+	t	8	0.02	442	4.4	9.0	*0.27*	*2.22*	0.66	5.97
LP_INV	t	0	vdd	t	8	0.47	430	4.4	4.9	*0.21*	*1.28*	0.43	2.10
LP_INV	t	0	+	t	8	0.39	414	4.4	5.0	*0.21*	*1.12*	0.40	1.96
LP_INV	t	-	vdd	t	8	0.10	443	4.4	4.9	*0.21*	*1.28*	0.43	2.08
LP_INV	t	-	+	t	8	0.02	442	4.4	5.0	*0.21*	*1.12*	0.39	1.95
LP_INVN	t	0	t	t	8	0.48	391	4.4	4.6	*0.21*	*2.44*	0.66	3.00
LP_INVN	t	-	t	t	8	0.11	378	4.4	4.4	*0.11*	*2.25*	0.54	2.39
LP_INVP	3 x t	t	vdd	t	12	0.58	343	4.4	4.8	*0.21*	*1.29*	0.43	2.05
LP_INVP	3 x t	t	+	t	12	0.50	325	4.4	5.1	*0.21*	*1.15*	0.40	2.02
Type5	3 x 0	t	t	t	12	0.59	360	4.4	7.5	*0.26*	*6.33*	1.47	10.95
Type6	3 x 0	t	+	t	12	0.51	325	4.4	8.2	*0.25*	*2.50*	0.70	5.74
Type7	3 x -	-	t	t	12	0.11	378	4.4	6.3	*0.24*	*4.45*	1.08	6.79

* The average energy is calculated as 80% of the read energy plus 20% of the write energy

[+] Config Key (SG or LP config): **t** = tied/SG; **t_{-+}** = SG: -0.2 to 1.2 V swing; **0** = LP: 0V bias; **-** = LP: -0.2 V bias; **vdd** = LP: V_{DD}=1 V bias; + = LP: 1.2 V bias; **3 x t** = 3 fins per tran. SG

Table 3.7. FinFET 8T SRAM 32×1024 array simulation results summary

Scheme	Configuration^{+}				# Fins	Leakage / Cell (nA)	SNM (mV)	32 bits × 1024 words Array					
	Pass	Inv-N	Inv-P	Read				Delay (ps)		Energy (fJ)			Ave* EDP (ps×fJ)
								Rd	*Wr*	*Rd*	*Wr*	*Ave**	
SG	3 x t	t	t	t	12	0.59	360	32	3	*0.9*	*6.9*	2.1	66
SG	3 x t	t	t	t_{0+}	12	0.59	360	26	3	*0.9*	*6.8*	2.1	54
SG	3 x t	t	t	$t_{-\,vdd}$	12	0.59	360	32	3	*0.9*	*6.9*	2.1	67
SG	3 x t	t	t	t_{-+}	12	0.59	360	26	3	*0.9*	*6.8*	2.1	55
LP	3 x 0	0	vdd	0	12	0.47	430	70	4	*1.2*	*5.3*	2.0	139
LP	3 x 0	0	+	0	12	0.39	414	70	5	*1.0*	*4.5*	1.7	119
LP	3 x -	-	vdd	-	12	0.10	443	84	4	*0.3*	*4.6*	1.2	99
LP	3 x -	-	+	-	12	0.02	442	84	5	*0.1*	*3.8*	0.9	73
LP_SGR	0	0	vdd	t	8	0.47	430	32	7	*0.8*	*6.2*	1.8	58
LP_SGR	0	0	+	t	8	0.39	414	32	8	*0.7*	*5.3*	1.6	50
LP_SGR	-	-	vdd	t	8	0.10	443	32	8	*0.4*	*5.9*	1.5	47
LP_SGR	-	-	+	t	8	0.02	442	32	8	*0.3*	*5.0*	1.2	39
LP_INV	t	0	vdd	t	8	0.47	430	32	4	*0.8*	*3.7*	1.3	42
LP_INV	t	0	+	t	8	0.39	414	32	5	*0.7*	*3.3*	1.2	38
LP_INV	t	-	vdd	t	8	0.10	443	32	4	*0.4*	*3.2*	0.9	30
LP_INV	t	-	+	t	8	0.02	442	32	5	*0.3*	*3.0*	0.8	26
LP_INVN	t	0	t	t	8	0.48	391	32	4	*0.8*	*7.0*	2.0	63
LP_INVN	t	-	t	t	8	0.11	378	32	4	*0.4*	*6.0*	1.5	48
LP_INVP	3 x t	t	vdd	t	12	0.58	343	32	4	*0.9*	*4.0*	1.5	47
LP_INVP	3 x t	t	+	t	12	0.50	325	32	4	*0.8*	*3.7*	1.4	43
Type5	3 x 0	t	t	t	12	0.59	360	32	6	*0.9*	*12.6*	3.2	102
Type6	3 x 0	t	+	t	12	0.51	325	32	7	*0.8*	*5.9*	1.8	57
Type7	3 x -	-	t	t	12	0.11	378	32	5	*0.4*	*9.7*	2.2	71

* The average energy is calculated as 80% of the read energy plus 20% of the write energy

$^{+}$ Config Key (SG or LP config): **t** = tied/SG; **t**$_{-+}$= SG: -0.2 to 1.2 V swing; **0** = LP: 0V bias;
- = LP: -0.2 V bias; **vdd** = LP: V_{DD}=1 V bias; **+** = LP: 1.2 V bias; **3 x t** = 3 fins per tran. SG

3.4.1 8T FinFET SRAM Read Operation

In the 8T SRAM cell, the read delay depends on the biasing and voltage swings of transistors T7 and T8. Table 3.8 shows configurations for transistors T7 and T8 and the delays for the 16×16 and 32×1024 arrays. In this table, delays are compared to the SG-mode with read-line swing only up to V_{DD} = 1 V. For both array sizes, the best speed is obtained when T7 and

T8 are in SG configuration with a read-line swing up to V_{DD} + 0.2 V; delays are 3.7 ps for the 16×16 array and 26 ps for the 32×1024 array. The next best delay is when these transistors are in SG configuration with a read-line swing only up to V_{DD}; delays are 4.4 and 32 ps for the small and large arrays. This grouping includes a configuration with -0.2 V to V_{DD} read-line swing which has no effect on delay (in the 16×16 array, this slightly increases delay due to larger swing from -0.2 V). The worst read delays occur when T7 and T8 are in LP configuration with their back gates biased to 0 V (8.8 and 70 ps) and -0.2 V (10.2 and 84 ps). Using SG configuration for T7 and T8 maximizes the current driving capacity of the read path and thus minimizes the read delay.

Table 3.8. T7 and T8 configurations and read delay

T7, T8 Configuration Description	**Read Config.**	**Delay (ps) & Comp***	
		16 × 16	***32 × 1024***
Shorted gate w/ swing to V_{DD}+0.2V	t_{-+} / t_{0+}	3.7 (0.8)	26 (0.8)
Shorted gate	t / t_{-vdd}	4.4-4.5 (1.0)	32 (1.0)
Low power $V_{backgate}$ = 0V	0	8.8 (2.0)	70 (2.2)
Low power $V_{backgate}$ = -0.2V	-	10.2 (2.3)	84 (2.6)

* In parentheses, the delay is compared to shorted gate configuration with regular swing

The read energy is mainly determined by the configuration of transistors T7 and T8. Table 3.9 shows maximum delays and read energy for the large array (32×1024). When the back gates of T7 and T8 are reverse-biased at -0.2 V, this requires the least energy of 2.6 aJ (69% less energy than when the back gates are tied to front gates). When the back gates are set to 0 V (GND), energy increases slightly to 2.8 aJ. A more significant increase to 7.9-8.7 aJ is seen when the front and back gates are tied in SG configuration. Using a larger read-line swing requires the most energy of 12.6-12.7 aJ which is 52% larger than the energy required for 0-to-

V_{DD} read-line swing. The read delay is minimized when SG configuration is used for the read transistors T7 and T8, but the larger current drive capability of these transistors causes these schemes to have larger read energies.

Table 3.9. T7 and T8 configurations and read energy of 32×1024 array

T7, T8 Configuration	Read Delay (ps)	Read Energy (aJ) & Comp*
-	84	2.6 (0.31)
0	70	2.8 (0.34)
$t / t_{-\,\mathrm{vdd}}$	32	7.9-8.7 (1.00)
t_{-+} / t_{0+}	26	12.6-12.7 (1.52)

* In parentheses, energy is compared to shorted gate configuration with regular swing

3.4.2 8T FinFET SRAM Write Operation

Write delay is determined by the relative current driving strength of the pass transistors (T1 and T2) over the inverter transistors (T3-T6). Having larger current drive capability for the T1 and T2 transistors leads to shorter write delays. Table 3.10 shows the write delays for both array sizes in ascending order. The fastest write delays are 4.1 and 3 ps (16×16 and 32×1024 arrays, respectively) for the SG configuration where there are three fins per pass transistors FinFETs and the inverter transistors are minimum-sized. Using LP configuration to reverse-bias the back gates to weaken the Inv-N transistors helps to shorten delays as shown in the table with the LP_INVN schemes (the second and third entries). However, biasing the Inv-P transistor back gates increases the write delays since these transistors are needed to set a logic “1” in the inverter; this is shown in the table with the LP_INVP schemes (the fourth and seventh entries). The LP_SGR scheme uses minimum-sized transistors in LP configuration for T1-T6 which presents the longest delays due to low current driving capabilities of the pass transistors which are also coupled with low driving capabilities of the inverters. Write delays for this scheme are 8.4 and 9 ps for the 16×16 array and 7 and 8 ps for the 32×1024 array.

Table 3.10. T1-T6 configurations and write delay

Sample Scheme	**Transistor Configuration**			**Delay (ps) & Comp***	
	Pass	***Inv-N***	***Inv-P***	***16 × 16***	***32 × 1024***
SG	3 x t	t	t	4.1 (1.00)	3 (1.00)
LP_INVN	t	-	t	4.4 (1.07)	4 (1.33)
LP_INVN	t	0	t	4.6 (1.12)	4 (1.33)
LP_INVP	3 x t	t	vdd	4.8 (1.17)	4 (1.33)
LP_INV	t	0 / -	vdd	4.9 (1.20)	4 (1.33)
LP_INV	t	0 / -	+	5.0 (1.22)	5 (1.67)
LP_INVP	3 x t	t	+	5.1 (1.24)	4 (1.33)
LP	3 x 0	0 / -	vdd	5.3 (1.29)	4 (1.33)
LP	3 x 0/-	0 / -	+	5.5 (1.34)	5 (1.67)
Type7	3 x -	-	t	6.3 (1.54)	5 (1.67)
Type5	3 x 0	t	t	7.5 (1.83)	6 (2.00)
Type6	3 x 0	t	+	8.2 (2.00)	7 (2.33)
LP_SGR	0	0	vdd / +	8.4 (2.05)	7-8 (2.33)
LP_SGR	-	-	vdd / +	9.0 (2.20)	8 (2.67)

* In parentheses, the delay is compared to the SG scheme.

The cell write energy is determined by the pass transistors, Inv-N, and Inv-P transistors. Table 3.11 shows maximum delays and write energy for the large array (32×1024). The least write energy of 72-74 aJ is obtained when the Inv-P transistors' back gates are reversed biased to V_{DD} + 0.2V and the pass transistors are in SG configuration (51% less energy then when the back gates are tied to front gates). From the table, it can be observed that energy has a closer dependency on the setting of the Inv-P transistors' back gates; these transistors limit the short-circuit current of the inverters. When the pass transistors are in SG configuration, they have higher current driving capability and provide faster write speed which minimizes the time the cross-coupled inverters dissipate switching/short-circuit current during a write operation. Using LP configuration and reverse-biasing the cross-coupled inverter transistors, especially the Inv-P transistors, also helps limit the switching/short-circuit current to reduce write energy.

Table 3.11. T1-T6 configurations and write energy of 32×1024 array

Sample Scheme	**Transistor Configuration**			**Write Delay (ps)**	**Write Energy (aJ) & Comp***
	Pass	***Inv-N***	***Inv-P***		
LP_INV	t / 3 x t	- / 0 / t	+	4-5	72-74 (0.49)
LP_INV	t / 3 x t	- / 0 / t	vdd	4	78-82 (0.53)
LP	3 x 0/-	0 / -	+	5	92-93 (0.62)
LP	3 x 0/-	0 / -	vdd	4	103-109 (0.71)
LP_SGR	0/-	0/-	+	8	131-136 (0.89)
LP_INVN	t	-	t	4	146 (0.97)
Type6	3 x 0	t	+	7	148 (0.99)
SG	3 x t	t	t	3	150 (1.00)
LP_SGR	0 / t	0	vdd / t	4-7	153-156 (1.03)
LP_SGR	-	-	vdd	8	160 (1.07)
Type7	3 x -	-	t	5	256 (1.71)
Type5	3 x 0	t	t	6	333 (2.22)

* In parentheses, the energy is compared to the SG scheme.

3.4.3 8T FinFET SRAM Leakage Current

There are five major leakage currents in the 8T SRAM as shown in Figure 3.2. As mentioned earlier the 6T SRAM cell has the same structure, but with just six transistors, T1-T6. The static current, I_{inv1} and I_{inv2}, are most commonly reported as the cross-coupled inverters' leakage currents. This 8T SRAM leakage is also similar to the 6T SRAM leakage in that memory cells that are not selected for a write or read operation draw leakage current from (or to) the drivers. Figure 3.2 shows a SRAM cell with a "0" stored (inverter 1 and 2 output 1 and 0, respectively). If a "1" is being written (WBit and WNBit are 1 and 0, respectively) there is a leakage current from the driver of the WBit line through transistor T1; this leakage current is called I_{T1}. There is another leakage current from the cell to the driver of the WNBit line through T2; this leakage current is called I_{T2}.

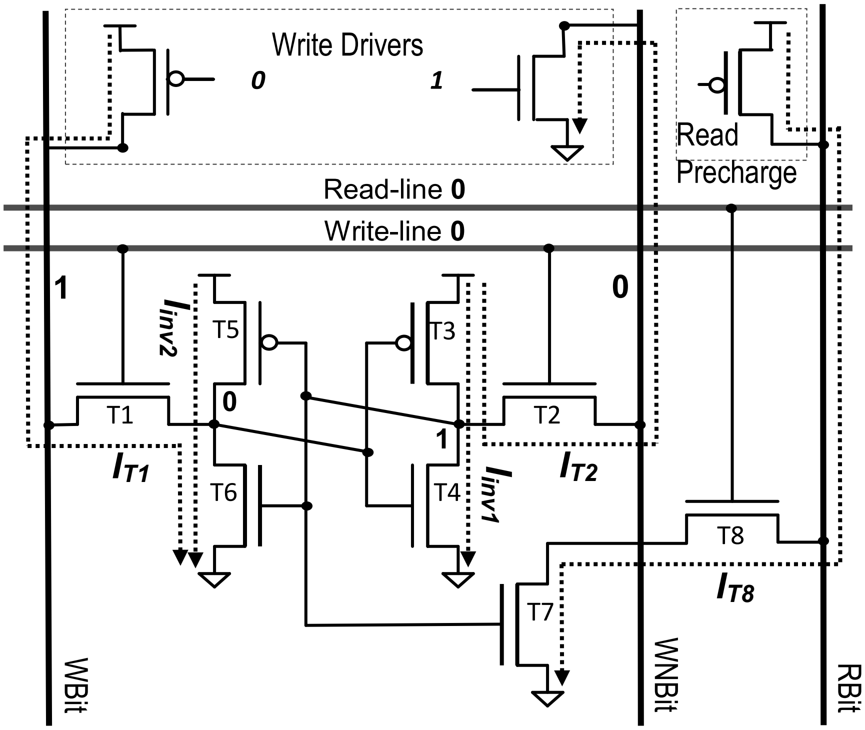


Figure 3.2. 8T SRAM cell leakage currents under write and read operations on other cells in the column

When a read occurs in a 6T SRAM array, at precharge time both Bit and NBit lines are set to “1” and leakage currents I_{T1} and I_{T2} are present. However, the 8T SRAM cell has an independent read port as shown in Figure 3.2. In addition to the 6T cell leakage currents (I_{inv1}, I_{inv2}, I_{T1}, and I_{T2}) there is a leakage current from the read precharge circuitry through transistor T8 to the SRAM cell. This leakage current is called I_{T8} and shown in Figure 3.2. If a “0” is stored in the SRAM cell transistor T7 is ‘on’; V_{DS} across T8 is approximately V_{DD}. On the other hand, if a “1” is stored in the cell transistor T7 is ‘off’; thus, there are two transistors in series that are ‘off’ (T7 and T8) presenting a larger resistance to current I_{T8}.

Table 3.12 shows simulation results of the leakage currents for the read and write operations for the 8T SG scheme. This table diagrams the status of the cell’s value and the WBit and WNBit values using the notation: **WBit [Inv2_output Inv1_output] WNBit**. The example shown in Figure 3.2 is the status 1 [0 1] 0. There is much symmetry in the leakage currents for each 8T scheme. There are four distinct leakage cases: write operation with the bitlines and cell having the same values (0 [0 1] 1 and 1 [1 0] 0), write operation with the bitlines and the cell having opposite values (0 [1 0] 1 and 1 [0 1] 0), a read operation with a stored logic “0” ([0 1]), and the read operation with a stored logic “1” ([1 0]). The write leakage cases are identical to a 6T SRAM cell and are displayed in Figure 2.3 while the read leakage cases are shown in Figure 3.3. The I_{T1} and I_{T2} leakage currents are maximized when the value in the cell is opposite that of the WBit or WNBit line and this also maximizes the total leakage. The total leakage current for the four cases are presented in Table 3.13 for each 8T scheme. The three types of leakage, cross-coupled inverter leakage, write leakage, and read leakage, are examined in greater detail in the following subsections.

Table 3.12. FinFET SG (normal read-line swing) 8T scheme read/write leakage (nA)

Leakage Current	Write Condition				Read Condition	
	0 [0 1] 1	*1 [0 1] 0*	*0 [1 0] 1*	*1 [1 0] 0*	*[0 1]*	*[1 0]*
I_{inv1}	0.50	0.50	0.09	0.09	0.50	0.09
I_{inv2}	0.09	0.09	0.50	0.50	0.09	0.50
I_{T1}	0.00	1.50	1.50	0.00	0.75*	0.75*
I_{T2}	0.00	1.50	1.50	0.00	0.75*	0.75*
I_{T8}^{+}	--	--	--	--	0.50	0.19
I_{TOTAL}	**0.59**	**3.60**	**3.60**	**0.59**	**2.60**	**2.29**

[+] When RBit is not precharged to V_{DD} in the write condition, I_{T8} leakage is negligible.

* Average values of I_{T1} and I_{T2} leakage are used for the read condition. These leakage currents are present in the read condition and depend on WBit and WNBit lines' status.

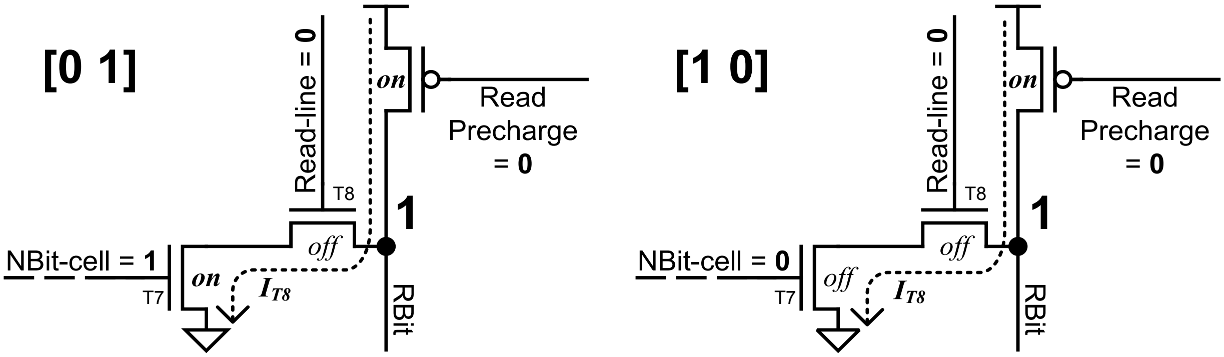


Figure 3.3. 8T SRAM cell read leakage currents

Table 3.13. FinFET 8T SRAM simulation results of read/write leakage totals (nA)

Scheme	Configuration				# Fins	*Write 0 [0 1] 1*	*Write 1 [0 1] 0*	*Read [0 1]*	*Read [1 0]*
	Pass	**Inv-N**	**Inv-P**	**Read**					
SG	3 x t	t	t	t	12	0.59	3.60	2.60	2.29
SG	3 x t	t	t	t_{0+}	12	0.59	3.60	2.60	2.29
SG	3 x t	t	t	t_{-vdd}	12	0.59	3.60	2.10	2.10
SG	3 x t	t	t	t_{-+}	12	0.59	3.60	2.10	2.10
LP	3 x 0	0	vdd	0	12	0.47	2.79	2.01	1.78
LP	3 x 0	0	+	0	12	0.39	2.71	1.94	1.70
LP	3 x -	-	vdd	-	12	0.10	0.20	0.17	0.16
LP	3 x -	-	+	-	12	0.02	0.13	0.09	0.08
LP_SGR	0	0	vdd	t	8	0.47	1.24	1.35	1.04
LP_SGR	0	0	+	t	8	0.39	1.17	1.28	0.97
LP_SGR	-	-	vdd	t	8	0.10	0.13	0.62	0.31
LP_SGR	-	-	+	t	8	0.02	0.06	0.54	0.23
LP_INV	t	0	vdd	t	8	0.47	1.47	1.47	1.16
LP_INV	t	0	+	t	8	0.39	1.39	1.39	1.08
LP_INV	t	-	vdd	t	8	0.10	1.10	1.10	0.79
LP_INV	t	-	+	t	8	0.02	1.02	1.02	0.72
LP_INVN	t	0	t	t	8	0.48	1.48	1.48	1.17
LP_INVN	t	-	t	t	8	0.11	1.11	1.11	0.80
LP_INVP	3 x t	t	vdd	t	12	0.58	3.59	2.58	2.28
LP_INVP	3 x t	t	+	t	12	0.51	3.51	2.51	2.20
Type5	3 x 0	t	t	t	12	0.59	2.92	2.25	1.95
Type6	3 x 0	t	+	t	12	0.51	2.83	2.17	1.86
Type7	3 x -	-	t	t	12	0.11	0.21	0.66	0.35

3.4.3.1 Cross-Coupled Inverter Leakage

The leakage presented in Table 3.6 and Table 3.7, and summarized with the configurations of transistors T3-T6 in Table 3.14, is the leakage current from the supplies of the cross-coupled inverters, namely I_{inv1} and I_{inv2}. Schemes, such as LP, LP_SGR, and LP_INV, which reverse-bias the back-gates of the cross-coupled inverter transistors, T3-T6, minimize these leakage currents. Having the n-type and p-type FinFET back gates biased to -0.2V and V_{DD} + 0.2 V, respectively, yields the smallest leakage (0.02 nA) which is just 3% of the largest leakage of the SG scheme (0.59 nA). The n-type FinFETs have roughly 5X more leakage than p-type FinFETs, thus the Inv-N, T4 and T6, configuration sets the order of the leakage currents reported in Table 3.14. Inv-N with -0.2 V (-) back-gate bias has the lowest leakage currents, this is followed by the Inv-N with 0 V (0) back-gate bias, and lastly the Inv-N with the front and back gates tied (t) together in SG configuration (no back-gate bias) is listed. For a given Inv-N configuration, the Inv-P configuration reduces leakage in the following order: V_{DD} + 0.2 V (+), V_{DD}, and t. Table 3.14 includes comparisons, reported in parentheses, which are based on the SG scheme, which has the largest leakage current of the schemes reported.

Table 3.14. T3-T6 configurations and cross-coupled inverter leakage

Sample Scheme	**Configuration**		**Leakage (nA) & Comp***
	Inv-N	***Inv-P***	
LP	-	+	0.02 (0.03)
LP	-	vdd	0.10 (0.17)
LP_INVN	-	t	0.11 (0.19)
LP	0	+	0.39 (0.66)
LP	0	vdd	0.47 (0.80)
LP_INVN	0	t	0.48 (0.81)
LP_INVP	t	+	0.50-0.51 (0.86)
LP_INVP	t	vdd	0.58 (0.98)
SG	t	t	0.59 (1.00)

* In parentheses, the leakage is compared to the SG scheme.

3.4.3.2 Write Leakage

The write leakage presented in Table 3.13, and summarized with transistor configurations in Table 3.15, is the sum of I_{inv1} and I_{inv2} plus the I_{T1} and I_{T2} leakage currents from the write drivers through the transmission gate transistors T1 and T2 of an idle cell. This occurs during a write operation to a row of SRAM cells. All cells on other rows share the WBit and WNBit lines with the written cells, so as the write drivers set the bit lines to the write value, these I_{T1} and I_{T2} leakage currents occur. The write leakage is smaller when the WBit and WNBit values are identical to the stored value within an idle cell. In this case, the write leakage is equal to the cross-coupled inverter leakage since the transmission gate leakage is negligible. It is observed that the LP, LP_SGR, and Type7 schemes, which have the lowest I_{inv1}, I_{inv2}, I_{T1}, and I_{T2} leakage currents, also have the lowest write leakage. LP_SGR has the lowest write leakage (0.02 nA and 0.06 nA), just 3% of the largest write leakage of the SG scheme (0.59 nA and 3.60 nA), since the back gates of transistors T1-T6 are reverse-biased and minimum-sized. As with the cross-coupled inverter leakage, the configuration of Inv-N, the n-type transistors T4 and T6, sets the order of the write leakage with -0.2V (-) back-gate bias having the lowest write leakage currents, this is followed by the Inv-N with 0V (0) back-gate bias, and lastly the Inv-N with the front and back gates tied (t) together in SG configuration (no back-gate bias) is listed. For a given Inv-N configuration, the pass transistors' configuration (in the order: -0.2V (-), 0V (0), and tied/SG (t)) reduces leakage followed by the configuration of Inv-P in the following order: V_{DD} + 0.2 V (+), V_{DD}, and t. Table 3.15 includes comparisons, reported in parentheses, which are based on the SG scheme, which has the largest write leakage current of the schemes reported.

Table 3.15. Transistor configurations and write leakage

Sample Scheme	Transistor Configuration			Leakage (nA) & Comp*	
	Pass	*Inv-N*	*Inv-P*	*0 [0 1] 1*	*1 [0 1] 0*
LP	- / 3 x -	-	+	0.02 (0.03)	0.06-0.13 (0.03)
LP_SGR	-	-	vdd	0.10 (0.17)	0.13 (0.04)
LP	3 x -	-	vdd / t	0.10-0.11 (0.18)	0.20-0.21 (0.06)
LP_INV	t	-	+	0.02 (0.03)	1.02 (0.28)
LP_INV	t	-	vdd / t	0.10-0.11 (0.18)	1.10-1.11 (0.31)
LP_SGR	0	0	+ / vdd	0.39-0.47 (0.73)	1.17-1.24 (0.33)
LP_INV	t	0	+ / vdd / t	0.39-0.48 (0.74)	1.39-1.48 (0.40)
LP	3 x 0	0	+ / vdd	0.39-0.47 (0.73)	2.71-2.79 (0.76)
Type6	3 x 0	t	+ / t	0.51-0.59 (0.93)	2.83-2.92 (0.80)
LP_INVP	3 x t	t	+ / vdd	0.51-0.58 (0.92)	3.51-3.59 (0.99)
SG	3 x t	t	t	0.59 (1.00)	3.60 (1.00)

* In parentheses, the leakage is compared to the SG scheme.

3.4.3.3 Read Leakage

The read leakage presented in Table 3.13, and summarized with transistor configurations in Table 3.16, is the sum of I_{inv1} and I_{inv2}, plus the average values of I_{T1} and I_{T2} for a write value of "0" and "1" on the WBit and WNBit lines since the capacitive loads of these lines will maintain the write value for multiple cycles after a write operation, plus the I_{T8} leakage current from the read precharge transistor through the two series read transistors T7 and T8 of an idle cell. This occurs during a read operation to a row of SRAM cells. All cells on other rows share the RBit lines with the written cells, so as the read precharge transistors set the RBit lines to V_{DD}, the I_{T8} leakage currents occur. If the idle cell's stored value is logic "0," then T7 will be "on" while T8 is "off" and the leakage current I_{T8} is equal to the leakage of one FinFET, T8. However, if the idle cell's stored value is logic "1," then both T7 and T8 are "off" and the leakage current I_{T8} is that of the two series n-type FinFETs. In simulation we have found this leakage value to be approximately 60% less than the leakage of one n-type FinFET. It is observed that the LP, LP_SGR, and Type7 schemes, which have the lowest I_{inv1}, I_{inv2}, I_{T1}, and I_{T2} leakage currents, also have the lowest read leakage. LP has the lowest read leakage (0.09 nA and 0.08 nA), just 3% of the largest read leakage of the SG scheme (2.60 nA and 2.29 nA), since the back gates of all eight transistors are reverse-biased. Once again, the configuration of Inv-N, the n-type transistors T4 and T6, sets the order of the read leakage with -0.2 V (-) back-gate bias having the lowest write leakage currents, this is followed by the Inv-N with 0 V (0) back-gate bias, and lastly the Inv-N with the front and back gates tied (t) together in SG configuration (no back-gate bias) is listed. For a given Inv-N configuration, the ordering for increasing read leakage is not as distinct since all eight transistors contribute to read leakage. From Table 3.12 it can be observed that the contributions of the currents other than I_{T8} outweigh the contribution of the I_{T8} leakage current; I_{T8} is not the majority of the total read leakage for the SG scheme. Table 3.16 includes comparisons, reported in parentheses, which are based on the SG scheme, which has the largest read leakage current of the schemes reported.

Table 3.16. Transistor configurations and read leakage

Sample Scheme	**Transistor Configuration**				**Leakage (nA) & Comp***	
	Pass	***Inv-N***	***Inv-P***	***Read***	***[0 1]***	***[1 0]***
LP	3 x -	-	+ / vdd	-	0.09-0.17 (0.05)	0.08-0.16 (0.05)
LP_SGR	- / 3 x -	-	+ / vdd	t	0.54-0.66 (0.23)	0.23-0.35 (0.13)
LP_INV	t	-	+ / vdd / t	t	1.02-1.11 (0.41)	0.72-0.80 (0.33)
LP_SGR	0	0	+ / vdd	t	1.28-1.35 (0.51)	0.97-1.04 (0.44)
LP_INV	t	0	+ / vdd / t	t	1.39-1.48 (0.55)	1.08-1.17 (0.49)
LP	3 x 0	0	+ / vdd	0	1.94-2.01 (0.76)	1.70-1.78 (0.76)
SG	3 x t	t	t	$t_{-\,vdd}$ / $t_{-\,0}$	2.10 (0.81)	2.10 (0.92)
Type6	3 x 0	t	+ / t	t	2.17-2.25 (0.85)	1.86-1.95 (0.83)
LP_INVP	3 x t	t	+ / vdd	t	2.51-2.58 (0.98)	2.20-2.28 (0.98)
SG	3 x t	t	t	t / $t_{0\,+}$	2.60 (1.00)	2.29 (1.00)

* In parentheses, the leakage is compared to the SG scheme.

3.4.4 8T FinFET SRAM Noise Margins

Static noise margins (SNM) of the 8T SRAM schemes are presented in Table 3.6 and Table 3.7. The SNMs were measured using the "maximum squares" simulation method presented in [16]. In addition, each cell's characteristic curves, also referred to as the butterfly-curve, for the write static noise margin were also analyzed to ensure no write errors exist [17]. The read static noise margin (RSNM) of the 8T cells is equivalent to the SNM. This is because the value on the RBit line will not affect the cell's contents, unlike 6T SRAM cells. Nonetheless, during a read operation, if more noise than the SNM is present, then the cell can be flipped. This is major advantage for the 8T SRAM cells since for 6T SRAM cells the RSNM is less than the SNM.

The SNM depends on the configuration of the cross-coupled inverter transistors T3-T6. The LP, LP_SGR, and LP_INV schemes have the greatest SNMs of over 440 mV (over 22% better than the 360 mV SNM of the SG scheme) for configurations which reverse-bias the back gates of T4 and T6, the Inv-N transistors. To achieve a high SNM, the Inv-P transistors (T3 and T5) must be strong relative to the Inv-N transistors (T4 and T6) in order to overcome leakage current from T4 and T6 [18]. This is apparent in the configuration summary provided by Table 3.17 which shows the SNM as a result of the configurations of the cross-coupled inverter transistors T3-T6 and the number of fins for the Inv-N FinFETs T4 and T6. This can be overdone however, as the LP_INVN scheme has an 11-14% lower SNM of 378-391 mV than these three schemes. A proper ratio must be in place for the Inv-P transistors (T3 and T5) to overcome leakage from the Inv-N transistors (T4 and T6) and vice versa. The worst-performing schemes, LP_INVP and Type6, have SNMs of 325 mV (10% worse than the SG scheme) and use SG configuration for T4 and T6 while using LP configuration for T3 and T5, leading to a low SNM.

Table 3.17. T3-T6 configurations and SNM

Sample Scheme	**Configuration**		**SNM (mV) & Comp.**
	Inv-N	***Inv-P***	
LP	-	vdd	443 (1.23)
LP	-	+	442 (1.23)
LP	0	vdd	430 (1.19)
LP	0	+	414 (1.15)
LP_INVN	0	t	391 (1.09)
LP_INVN	-	t	378 (1.05)
SG	t	t	360 (1.00)
LP_INVP	t	vdd	343 (0.95)
LP_INVP	t	+	325 (0.90)

* In parentheses, the SNM is compared to the SG scheme.

3.4.5 8T FinFET SRAM Overall Performance Summary

The LP_SGR and LP_INV 8T schemes have the two lowest EDPs for a 32×1024 8T SRAM array and have higher SNMs than the SG scheme. In particular, the 8T LP_SGR scheme with a -0.2 V back gate (BG) bias for the Pass and Inv-N transistors and a 1.2 V BG-bias for the Inv-P transistors, the 8T LP_INV scheme with a -0.2 V BG-bias for the Inv-N transistors and a 1.2 V BG-bias for the Inv-P transistors, and the 8T LP_INV scheme with a -0.2 V BG-bias for the Inv-N transistors and a 1 V BG-bias for the Inv-P transistors have among the lowest average EDPs for the larger 32×1024 8T SRAM array and possess SNMs larger than 440 mV. Of the 8T SRAM cells, only these three cells and the SG cell, with normal 0 V to 1 V read-line swing, will be examined further in this dissertation.

3.5 8T FinFET SRAM Comparisons to 6T FinFET and 8T CMOS

This section will provide comparisons of the 8T FinFET SRAM schemes' performance results to that of the 6T FinFET SRAM schemes and an 8T CMOS SRAM cell. The dynamic performance (read and write operations), leakage current, and noise margins will be examined in the following subsections. The last subsection will summarize the comparisons as a whole.

3.5.1 Dynamic Performance Comparisons to 6T FinFET Cells

Overall, the 8T SRAM cells outperform the 6T SRAM cells. The orthogonal read and write operations allow the FinFET 8T SRAM cells to be more optimally sized; the 6T portion of the cell can be sized for minimal leakage and the read transistors can be sized for a fast read time. The 8T SRAM cells unanimously outperform the 6T SRAM cells for both the 16×16 and 32×1024 array sizes. LP_INVP, the best-performing 6T scheme in terms of average EDP for both array sizes (as low as 2.45 ps×fJ for the 16×16 array and 69 ps×fJ for the 32×1024 array), has 17-23% (2.02 ps×fJ and 2.05 ps×fJ for a 16×16 array) and 37-38% (43 and 47 ps×fJ for a 32×1024 array) less average EDP as an 8T scheme.

The 8T SRAM cells have lower delays due to the dedicated read transistors, T7 and T8, which can be optimally configured in SG configuration for the fastest read time. The 6T SRAM cell cannot be optimally sized for the read operation since the write operation and leakage current will be negatively affected. The write times of the 8T SRAM cell are also shorter since the T1-T6 can be optimally sized for the write operation. As an example the fastest 6T scheme, the SG scheme, with maximum delays of 5.6 ps for the 16×16 array and 32 ps for the 32×1024 array is 21% faster for the 16×16 array (4.4 ps) and equally as fast for the 32×1024 array as an 8T scheme.

3.5.2 Dynamic Performance Comparisons to 32 nm CMOS 8T Cell

The 8T FinFET SRAM cells unanimously outperform the 32nm bulk-CMOS 8T cell. The performance results of a 16×16 array of the CMOS 8T cell is provided in Table 3.18. The CMOS 8T cell has an average EDP of 55.06 ps×fJ, which is 5X larger than the EDP of 10.95 ps×fJ for Type5, the worst-performing 8T FinFET cell in terms of average EDP for this smaller array. The slowest LP scheme's maximum delay of 10.2 ps is 35% faster than the 8T CMOS cell's delay of 15.9 ps and the 8T CMOS cell's leakage current of 5.99 nA is 10X more than the highest leakage of 0.59 nA for the SG and Type5 schemes. FinFETs provide lower leakage current and higher on-current than bulk-CMOS and this contributes greatly to the 8T FinFET schemes' outperformance of the 32 nm CMOS 8T cell.

Table 3.18. 32 nm CMOS 8T SRAM 16×16 array simulation results

Leak. / Cell (nA)	**Delay (ps)**		**Energy (fJ)**			**Ave+ EDP (ps×fJ)**	**SNM (mV)**
	Rd	***Wr***	***Rd***	***Wr***	***Ave+***		
5.99	15.9	14.6	*4.15*	*0.71*	3.46	55.06	289

* CMOS 8T SRAM transistor widths: Pass Tran. (T1, T2) = 5 × 16 nm;
Inv-N Tran. (T4, T6) = 2 × 16 nm; Inv-P Tran. (T3, T5) = 2 × 16 nm;
Read Tran. (T7, T8) = 8 × 16 nm

+ The average energy is calculated as 80% of the read energy plus 20% of the write energy

3.5.3 Leakage Current Comparisons to 6T FinFET Cells

The leakage currents of the 8T SRAM cells are very similar to that of the 6T cells. In particular, the cross-coupled inverter leakage and write leakage is the same for 6T and 8T cells with the same scheme and configuration. This is because the 8T cell is a 6T cell plus two additional read transistors. The cross-coupled inverter currents, I_{inv1} and I_{inv2}, are smallest for the LP, LP_INV, LP_INVN, and Type7 6T SRAM schemes, similar to the 8T SRAM cells. The LP and Type7 6T SRAM schemes also have the smallest read and write leakage currents, similar to the 8T SRAM schemes. The read leakage currents are slightly larger for the 8T SRAM schemes due to the independent read path. As shown in Figure 3.2, the total read leakage current of the 8T cells includes an additional current, I_{T8}, compared to the 6T cells. As an example, the 6T SG scheme has 1.09 nA of read leakage which is 52-58% less the 2.60 nA and 2.29 nA read leakages of the 8T SG scheme with normal read-line swing.

3.5.4 Leakage Current Comparisons to 32 nm CMOS 8T Cell

The leakage current results of the CMOS 8T cell are provided in Table 3.18 and Table 3.19; Table 3.18 includes the cross-coupled inverter leakage and Table 3.19 includes the read and write leakage currents. When compared to the 8T bulk-CMOS SRAM cell, the 8T FinFET SG scheme with normal read-line swing has 10X less cross-coupled inverter leakage, 4.0-5.3X less read leakage, and 2.7-10X less write leakage. This is due to the FinFETs having better I_{off} compared to bulk-CMOS. Additionally, the FinFET schemes which are in LP-configuration reduce their leakage current even further and vastly outperform the bulk-CMOS SRAM cell in terms of leakage currents. The bulk-CMOS cell has as much as 299X more cross-coupled inverter leakage than the LP, LP_SGR, and LP_INV schemes, as much as 114-154X more read leakage than the LP scheme, and as much as 163-300X more write leakage than the LP_SGR scheme.

Table 3.19. 32 nm CMOS 8T SRAM simulation results of read/write leakage totals (nA)

Write ***0 [0 1] 1***	***Write*** ***1 [0 1] 0***	***Read*** ***[0 1]***	***Read*** ***[1 0]***
6.00	9.79	13.86	9.14

* CMOS 8T SRAM transistor widths: Pass Tran. (T1, T2) = 5 × 16 nm; Inv-N Tran. (T4, T6) = 2 × 16 nm; Inv-P Tran. (T3, T5) = 2 × 16 nm; Read Tran. (T7, T8) = 8 × 16 nm

[+] The average energy is calculated as 80% of the read energy plus 20% of the write energy

3.5.5 Noise Margin Comparisons to 6T FinFET Cells

Since the 8T SRAM cell is comprised of a 6T SRAM cell and two read transistors, the SNMs of the 8T schemes are equal to that of the 6T cells for the same configurations of the cross-coupled inverter n-type and p-type FinFETs (T3-T6). The most noticeable difference, however, between the 8T and 6T cells is the lower RSNM for the 6T cells. Due to the 8T SRAM cell design, their contents are unaffected by the value on the RBit line during a read operation. Thus their RSNM is equal to their SNM. As discussed in the Appendix, the RSNM, compared to the SNM, for the 6T cells is between 40% lower for IG1 [14] and 95% lower for LP_INV. This is significant as the 8T SRAM cells are much less prone to a bit flip during a read operation. The orthogonal design of the 8T SRAM cells is an advantage over 6T SRAM cells in terms of the RSNM.

3.5.6 Noise Margin Comparisons to 32 nm CMOS 8T Cell

The SNM results of the CMOS 8T cell are provided in Table 3.18. All of the 8T FinFET cells have a larger SNM than the 32 nm bulk-CMOS 8T cell. The lowest 8T FinFET SNM belongs to a LP_INVP cell and a Type6 cell, however, this SNM is 12% larger than the 8T CMOS cell's SNM. The lower leakage current of the FinFET n- and p-type inverter transistors contributes to the larger SNM for almost all configurations of the n- and p-type inverter transistors.

Chapter 4

FinFET SRAM under Process Voltage Temperature Variations

The performance of 32×1024 arrays for the SG and LP 6T SRAM schemes and the SG, LP_SGR, and LP_INV (with -0.2 V Inv-N BG biasing and both 1 V and 1.2 V Inv-P BG biasing, henceforth abbreviated as LP_INV1 and LP_INV1.2, respectively) 8T SRAM schemes are examined and under the effects of process/parameter, voltage, and temperature (PVT) variations. The LP 6T scheme is the only 6T scheme that has a lower EDP than the SG 6T scheme for a 32×1024 6T SRAM array and possesses a RSNM larger than the SG 6T scheme. The LP_SGR and LP_INV 8T schemes have the three of the four lowest EDPs, the LP_INV scheme with 0 V Inv-N BG biasing and 1.2 V BG biasing has the third-lowest EDP (but higher than the LP_INV1 and LP_INV1.2 cells), for a 32×1024 8T SRAM array and the SG scheme is the conventional 8T SRAM scheme.

4.1 Performance under Process/Parameter Variations

The performance of the six cells are examined and compared using Monte Carlo simulations of process/parameter variations. Parameter variations are simulated using a quasi-Monte Carlo (QMC) analysis [20]. We use QMC samples for Monte Carlo simulation to achieve a good spread of data points in fewer simulations than with completely random samples. We used QMC and created sets of zero-mean and unit-standard deviation Sobol points to allow for completely independent assignment of varied values for all varied device parameters. These points were weighted by each parameter's mean and standard deviation: $X = \mu + x * \sigma$. Similar to other research which analyzes effects of parameter variations on FinFETs, each parameter's standard deviation is estimated to be $3\sigma = 10\% * \mu$ [18]; except for the standard deviation of Φ_G, the gate work function, for the front and back gates which is estimated to be $3\sigma = 50\ mV$ [21]. From Table 1.1, the following parameters were varied: L_G, H_{fin}, T_{SI}, N_{Body}, N_{DS}, R_{SD}, and T_{ox}, T_G, Φ_G for both the front and back gates. We obtained the results of 1000 simulations with varied parameters.

Table 4.1 shows the results of parameter variation simulations for the six examined cells. The next four subsections will analyze the following performances of the cells: read operation, write operation, leakage current, and noise margin. These subsections are followed by a concluding subsection analyzing the overall performance of the cells.

Table 4.1. FinFET SRAM parameter variation simulation results

	Scheme *Pass; Inv-N; Inv-P; Read*	Leakage / Cell (nA)	32 bits × 1024 words Array						SNM (mV)	RSNM (mV)
			Delay (ps)		Energy (fJ)			Ave* EDP (ps×fJ)		
			Rd	*Wr*	*Rd*	*Wr*	Ave*			
6T	SG	1.58	34	6	5.0	12.1	6.4	765	361	198
	t; t; t	*15.67*	*3*	*18*	*41.8*	*39.2*	*40.3*	*17318*	*16*	*14*
	LP	0.02	87	6	0.6	3.9	1.2	107	433	211
	-; -; +	*0.01*	*3*	*0*	*0.0*	*0.2*	*0.1*	*7*	*17*	*15*
8T	SG	1.13	33	4	1.5	8.5	2.9	116	361	
	3 x t; t; t; t	*11.47*	*2*	*8*	*12.4*	*19.8*	*12.8*	*677*	*11*	
	LP_SGR	0.02	33	8	0.3	5.0	1.2	41	434	
	-; -; +; t;	*0.01*	*4*	*1*	*0.1*	*0.3*	*0.1*	*9*	*7*	
	LP_INV1	0.10	33	5	0.4	3.4	1.0	33	435	
	t; -; vdd; t	*0.04*	*2*	*0*	*0.2*	*0.3*	*0.2*	*6*	*6*	
	LP_INV1.2	0.02	33	5	0.3	3.1	0.8	28	434	
	t; -; +; t	*0.01*	*2*	*1*	*0.1*	*0.6*	*0.1*	*4*	*7*	

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

4.1.1 Read Operation

The mean read delays of each 6T and 8T cell are no more than 7% longer than the nominal read delay values. The standard deviations of the read delays are all below 13% of the respective means. 8T LP_SGR has the largest standard deviation of 4 ps, or 12% of its delay of 33 ps. 6T LP has the smallest standard deviation of 3 ps, or 3.4% of its delay of 87 ps. The read delays are very similar to the nominal values with low variation for all schemes; greater variations are seen in the read energy.

The 6T and 8T SG schemes see the largest deviations in mean read energy when compared to their nominal values. The 6T SG scheme's mean read energy of 5.0 fJ is 2.1X greater than its nominal value of 2.4 fJ. The 8T SG scheme's mean read energy of 1.5 fJ is 1.7X greater than its nominal value of 0.9 fJ. These two schemes also have large standard deviations for the read energy. The 6T SG scheme has a standard deviation that is 8.4X greater than its mean while the 8T SG scheme has a standard deviation that is 8.3X greater than its mean. Conversely, the 8T LP_SGR, 8T LP_INV1, and 8T LP_INV1.2 cells' mean read energies are identical to their nominal values and the 6T LP scheme's mean read energy of 0.6 fJ is only 20% greater than its nominal value of 0.5 fJ. The standard deviations for these schemes are no greater than 50% of the mean read energies. The schemes using LP configuration see smaller standard deviations for read energy. Since the SG schemes use SG-configured FinFETs which leads to more leakage current during a read operation, it sees larger variations in read energy due to parameter variations than the other schemes which use LP-configured FinFETs.

4.1.2 Write Operation

The mean write delays of 6T LP, 8T LP_SGR, and 8T LP_INV1.2 cells are identical to their respective nominal values, while the 8T LP_INV1 cell's mean write delay is 1 ps longer (a

20% increase) than its nominal write delay. The standard deviations of the write delays of these schemes are no more than 20% of their respective means. The mean write delay of the 6T SG scheme is 1 ps longer (a 20% increase) and the mean write delay of the 8T SG scheme is 1 ps longer (a 33% increase) than their respective nominal write delays. The standard deviation of the write delay of the 6T SG scheme is triple its mean and the standard deviation of the 8T SG scheme is twice its mean. For both SG schemes, the strength of the cross-coupled inverter transistors, all in SG configuration, vary greatly due to parameter variations; this requires varying lengths of time to complete a write depending on the difficultly to overwrite the cell. The other schemes all use LP configuration for the cross-coupled inverter transistors and these transistors, while affected by parameter variations, usually remain weaker than the SG-configured Pass transistors and cause less variation in write delay.

The mean write energies of 6T LP and 8T LP_SGR are identical to their nominal values, while the mean write energy of 8T LP_INV1 is only 0.2 fJ larger (a 6.3% increase) and the mean write energy of 8T LP_INV1.2 is only 0.1 fJ larger (a 3.3% increase). The standard deviations of the write energies of these schemes are no more than 20% of their respective means. The 6T SG scheme’s mean write energy of 12.1 fJ is 30% larger than its nominal value of 9.3 fJ. The mean write energy of 8.5 fJ for the 8T SG scheme is 23% larger than its nominal value of 6.9 fJ. The standard deviation of the write energy of the 6T SG scheme is 3.2X its mean and the standard deviation of the write energy of the 8T SG scheme is 2.3X its mean. These results are similar to the results for the write delay. Greater variations in write delay cause greater variations in write energy. In addition, the varying difficulty, due to parameter variations, to write to the cell will increase the variation in power required for the write operation; this in turn increases the variation of the write energy.

4.1.3 Leakage Current

Overall, leakage current is most affected by parameter variations. The cross-coupled inverter leakage per cell means for the 6T LP, 8T LP_SGR, 8T LP_INV1, and 8T LP_INV1.2 cells are identical to their respective nominal values. These schemes, however, have standard deviations equal to at most 50% of their means, but since their mean leakage per cell is low (at 0.10 nA for 8T LP_INV1 and at 0.02 nA for 6T LP, 8T LP_SGR, and 8T LP_INV1.2), this is a small absolute variation. The 6T SG scheme's mean leakage per cell of 1.58 nA is 2.7X its nominal value of 0.59 nA. The 8T SG scheme's mean leakage per cell of 1.13 nA is 1.9X its nominal value of 0.59 nA. The standard deviation of the leakage per cell of the 6T SG scheme is 9.9X its mean and the standard deviation of the leakage per cell of the 8T SG scheme is 10.2X its mean. The SG schemes use SG cross-coupled inverter transistors which exhibit greater leakage current. Parameter variations cause this larger leakage current to have larger amounts of variation than the smaller leakage currents of the other four cells which use LP-configured cross-coupled inverter transistors. These large variations in leakage current per cell for the 6T SG and 8T SG schemes cause greater variation in their read and write energies, in addition to greater variation in their average energy and EDP.

The simulated effects of parameter variations on read and write leakage is displayed in Table 4.2. All six varied cells have average read and write leakage totals less than or equal to their nominal values (the 8T LP_SGR scheme has up to 9% less read leakage current while 6T LP scheme has identical read and write leakage totals). As a percentage of its mean, the 8T SG scheme has the smallest standard deviations of 43% and 18% of its mean for write 0 [0 1] 1 and write 1 [0 1] 0 leakage, respectively. Also, as a percentage of its mean, the 8T SG scheme has the smallest standard deviations of 19% of its mean for both read leakage conditions. However, both the 6T and 8T SG schemes have significantly greater read and write leakage than the other schemes which use LP configurations and reverse-bias the FinFET back gates. Overall, the smallest values for total read/write leakage and variation occur for the 6T LP scheme.

Table 4.2. Parameter variation results of FinFET SRAM read/write leakage totals (nA)

	Scheme *Pass; Inv-N; Inv-P; Read*	***Write 0 [0 1] 1***	***Write 1 [0 1] 0***	***Read 1 [0 1] 1***	
6T	SG	0.56	1.50	1.03	
	t; t; t	*0.24*	*0.42*	*0.33*	
	LP	0.02	0.06	0.04	
	-; -; +	*0.01*	*0.02*	*0.01*	
	Scheme *Pass; Inv-N; Inv-P; Read*	***Write 0 [0 1] 1***	***Write 1 [0 1] 0***	***Read [0 1]***	***Read [1 0]***
8T	SG	0.56	3.39	2.45	2.14
	3 x t; t; t; t	*0.24*	*0.62*	*0.46*	*0.40*
	LP_SGR	0.02	0.06	0.52	0.21
	-; -; +; t;	*0.01*	*0.02*	*0.24*	*0.07*
	LP_INV1	0.10	1.05	1.04	0.74
	t; -; vdd; t	*0.04*	*0.35*	*0.29*	*0.19*
	LP_INV1.2	0.02	0.97	0.97	0.66
	t; -; +; t	*0.01*	*0.33*	*0.29*	*0.18*

4.1.4 Noise Margins

The mean SNM of 361 mV for the 6T SG and 8T SG schemes is 1 mV larger than their nominal value of 360 mV. The mean SNM of 433 mV for the 6T LP scheme is 9 mV (2.0%) less than its nominal value of 442 mV. The mean SNM of 434 mV for the 8T LP_SGR and 8T LP_INV1.2 cells are 8 mV (1.8%) less than their nominal value of 442 mV and the mean SNM of 435 mV for the 8T LP_INV1 cell is also 8 mV (1.8%) less than its nominal value of 443 mV.

The 8T LP_INV1 cell has the smallest standard deviation of 6 mV, which is 1.4% of its mean, while the 8T LP_SGR and 8T LP_INV schemes have a standard deviation of 7 mV, which is 1.6% of their means, and the 6T LP scheme has a standard deviation of 17 mV, which is 3.9% of its mean. The 8T SG scheme has a standard deviation of 11 mV, which is 3.0% of its mean, while the 6T SG scheme has a standard deviation of 16 mV, which is 4.4% of its mean. The 8T schemes have slightly less variation in SNM due to the extra capacitance from the gates of read transistor T7 on the NBit-cell node. This extra capacitance is used to hold the stored bit in the cell and increases the difficultly to flip the stored bit. The 6T SG and 8T SG schemes also have slightly more SNM variation due to the larger exhibited leakage current of the SG-configured cross-coupled inverter transistors as explained previously.

The mean RSNM of 198 mV for the 6T SG scheme is 60% larger than its nominal value of 124 mV. The mean RSNM of 211 mV for the 6T LP scheme is 5% smaller than its nominal value of 222 mV. The standard deviations of the RSNM are similar for the 6T SG and LP schemes; both are 7.1% of their means. It is surprising to see the large increase in mean RSNM for the 6T SG scheme; however, it is possible to see an increase in RSNM if, due to parameter variations, the leakage current of the pass transistor connected to the side holding a logic “0” is reduced. It must also be noted, however, that the nominal RSNM for the 6T SG scheme is more than five standard deviations less than the mean RSNM value from the parameter variation simulations.

4.1.5 Overall Performance

Overall, there is minimal variation in delays for all six examined cells due to parameter variations. Cross-coupled inverter leakage current per cell is chiefly affected by the parameter variations. The 6T SG and 8T SG schemes see large variations in leakage current, and this causes large variations in energy and EDP. The mean average energy for the 6T LP, 8T LP_SGR, and 8T LP_INV1.2 cells are identical to their nominal values and their standard deviations are all less than 10% of their means, while the mean average energy for the 8T LP_INV1 cell is only 0.1 fJ larger (an 11.1% increase) and has a standard deviation of 20% of its mean. The 6T SG scheme has a mean average energy of 6.4 fJ, which is 68% than its nominal value of 3.8 fJ; its standard deviation is 6.3X more than its mean. The 8T SG scheme has a mean average energy of 2.9 fJ, which is 38% more than its nominal value of 2.1 fJ; its standard deviation is 4.4X more than its mean.

The mean average EDP for the 6T LP, 8T LP_SGR, and 8T LP_INV1.2 cells are no larger than 8% of their nominal values, while the 8T LP_INV1 cell has a 10% larger mean average EDP than its nominal average EDP. The 6T LP scheme has the smallest standard deviation of 6.5% of its mean average EDP, while the 8T LP_SGR, 8T LP_INV1, and 8T LP_INV1.2 standard deviations are 22%, 18%, and 14% of their mean average EDP, respectively. The mean average EDP of 765 ps×fJ for the 6T SG scheme is 6.4X more than its nominal value of 120 ps×fJ; its standard deviation is 22.6X more than its mean. The mean average EDP of 116 ps×fJ for the 8T SG scheme is 76% more than its nominal value of 66 ps×fJ; its standard deviation is 5.8X more than its mean. The schemes which use LP configuration for the cross-coupled inverter transistors have significantly less variation in average energy and average EDP than the SG schemes. Of the six cells examined using parameter variation simulations, the 8T LP_INV1.2 cell has the lowest average EDP for a 32×1024 array and the second-largest SNM (only the 8T LP_INV1 cell has a 1 mV larger SNM).

4.2 Performance under Supply Voltage Variations

For supply voltage (V_{DD}) variation simulations, the supply voltage was studied for 900, 950, 1000, 1050, and 1100 mV (or -10% to +10% of nominal V_{DD} = 1 V) for the six cells. Table 4.3 shows the simulation results of V_{DD} variations on the delays, energies, and average EDP of the 32×1024 arrays. Table 4.4 displays the simulation results of supply voltage variations on the leakage current, SNMs, and RSNMs of the SRAM cells. Table 4.5 summarizes the simulation results of V_{DD} variations on the read and write leakage current totals.

Table 4.3. Supply voltage variation results of FinFET SRAM delay, energy, and EDP

	Scheme *Pass; Inv-N; Inv-P; Read*	V_{DD}	**32 bits × 1024 words Array**											
			Delay (ps)				**Energy (fJ)**						**Ave* EDP (ps×fJ)**	
			Rd	*Comp*	Wr	*Comp*	Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*	Val	*Comp*
6T	SG t; t; t	-10%	34	*1.06*	5	*1*	1.8	*0.75*	6.8	*0.73*	2.8	*0.74*	95	*0.79*
		-5%	33	*1.03*	5	*1*	2.0	*0.83*	8.0	*0.86*	3.2	*0.84*	106	*0.88*
		Nom.	32	--	5	--	2.4	--	9.3	--	3.8	--	120	--
		+5%	31	*0.97*	4	*0.80*	2.9	*1.21*	10.6	*1.14*	4.4	*1.16*	137	*1.14*
		+10%	30	*0.94*	5	*1*	3.6	*1.50*	12.5	*1.34*	5.4	*1.42*	163	*1.36*
	LP -; -; +	-10%	99	*1.18*	7	*1.17*	0.4	*0.80*	2.7	*0.69*	0.9	*0.75*	86	*0.83*
		-5%	91	*1.08*	6	*1*	0.5	*1*	3.3	*0.85*	1.0	*0.83*	94	*0.91*
		Nom.	84	--	6	--	0.5	--	3.9	--	1.2	--	103	--
		+5%	79	*0.94*	6	*1*	0.6	*1.20*	4.7	*1.21*	1.4	*1.17*	114	*1.11*
		+10%	75	*0.89*	6	*1*	0.7	*1.40*	5.4	*1.38*	1.6	*1.33*	123	*1.19*
8T	SG 3 x t; t; t; t	-10%	34	*1.06*	3	*1*	0.8	*0.89*	5.6	*0.81*	1.7	*0.81*	58	*0.88*
		-5%	33	*1.03*	3	*1*	0.8	*0.89*	6.1	*0.88*	1.9	*0.90*	61	*0.92*
		Nom.	32	--	3	--	0.9	--	6.9	--	2.1	--	66	--
		+5%	31	*0.97*	3	*1*	0.9	*1*	7.8	*1.13*	2.3	*1.10*	71	*1.08*
		+10%	30	*0.94*	3	*1*	1.0	*1.11*	8.7	*1.26*	2.5	*1.19*	76	*1.15*
	LP_SGR -; -; +; t	-10%	34	*1.06*	9	*1.13*	0.2	*0.67*	3.5	*0.70*	0.9	*0.75*	30	*0.77*
		-5%	33	*1.03*	8	*1*	0.3	*1*	4.2	*0.84*	1.1	*0.92*	34	*0.87*
		Nom.	32	--	8	--	0.3	--	5.0	--	1.2	--	39	--
		+5%	31	*0.97*	8	*1*	0.3	*1*	5.9	*1.18*	1.4	*1.17*	44	*1.13*
		+10%	30	*0.94*	8	*1*	0.4	*1.33*	6.8	*1.36*	1.7	*1.42*	50	*1.28*
	LP_INV1 t; -; vdd; t	-10%	34	*1.06*	5	*1.25*	0.3	*0.75*	2.5	*0.78*	0.7	*0.78*	25	*0.83*
		-5%	33	*1.03*	5	*1.25*	0.3	*0.75*	2.8	*0.88*	0.8	*0.89*	27	*0.90*
		Nom.	32	--	4	--	0.4	--	3.2	--	0.9	--	30	--
		+5%	31	*0.97*	4	*1*	0.4	*1*	3.7	*1.16*	1.1	*1.22*	33	*1.10*
		+10%	30	*0.94*	4	*1*	0.5	*1.25*	4.3	*1.34*	1.2	*1.33*	37	*1.23*
	LP_INV1.2 t; -; +; t	-10%	34	*1.06*	5	*1*	0.2	*0.67*	2.2	*0.73*	0.6	*0.75*	21	*0.81*
		-5%	33	*1.03*	5	*1*	0.3	*1*	2.6	*0.87*	0.7	*0.88*	23	*0.88*
		Nom.	32	--	5	--	0.3	--	3.0	--	0.8	--	26	--
		+5%	31	*0.97*	4	*0.80*	0.3	*1*	3.3	*1.10*	0.9	*1.13*	28	*1.08*
		+10%	30	*0.94*	4	*0.80*	0.4	*1.33*	3.7	*1.23*	1.0	*1.25*	31	*1.19*

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

Table 4.4. Supply voltage variation results of FinFET SRAM leakage and noise margins

	Scheme *Pass; Inv-N; Inv-P; Read*	V_{DD}	**Leakage / Cell (nA)**		**SNM (mV)**		**RSNM (mV)**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*
6T	SG t; t; t	-10%	0.569	*0.96*	344	*0.96*	124	*1*
		-5%	0.580	*0.98*	352	*0.98*	124	*1*
		Nom.	0.592	--	360	--	124	--
		+5%	0.604	*1.02*	367	*1.02*	123	*0.99*
		+10%	0.615	*1.04*	373	*1.04*	123	*0.99*
	LP -; -; +	-10%	0.020	*0.95*	397	*0.90*	202	*0.91*
		-5%	0.021	*1*	419	*0.95*	213	*0.96*
		Nom.	0.021	--	442	--	222	--
		+5%	0.022	*1.05*	463	*1.05*	231	*1.04*
		+10%	0.022	*1.05*	485	*1.10*	239	*1.08*
8T	SG 3 x t; t; t; t	-10%	0.568	*0.96*	344	*0.96*		
		-5%	0.580	*0.98*	352	*0.98*		
		Nom.	0.592	--	360	--		
		+5%	0.604	*1.02*	367	*1.02*		
		+10%	0.615	*1.04*	373	*1.04*		
	LP_SGR -; -; +; t	-10%	0.020	*0.95*	397	*0.90*		
		-5%	0.021	*1*	419	*0.95*		
		Nom.	0.021	--	442	--		
		+5%	0.022	*1.05*	463	*1.05*		
		+10%	0.022	*1.05*	485	*1.10*		
	LP_INV1 t; -; vdd; t	-10%	0.092	*0.96*	400	*0.90*		
		-5%	0.094	*0.98*	422	*0.95*		
		Nom.	0.096	--	443	--		
		+5%	0.098	*1.02*	464	*1.05*		
		+10%	0.100	*1.04*	484	*1.09*		
	LP_INV1.2 t; -; +; t	-10%	0.020	*0.95*	397	*0.90*		
		-5%	0.021	*1*	419	*0.95*		
		Nom.	0.021	--	442	--		
		+5%	0.022	*1.05*	463	*1.05*		
		+10%	0.022	*1.05*	485	*1.10*		

Table 4.5. Supply voltage variations on FinFET SRAM read/write leakage totals (nA)

	Scheme *Pass; Inv-N; Inv-P; Read*	**V_{DD}**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read 1 [0 1] 1**			
			Val	*Comp*	Val	*Comp*	Val	*Comp*		
6T	SG t; t; t	-10%	0.569	*0.96*	1.532	*0.96*	1.050	*0.96*		
		-5%	0.580	*0.98*	1.563	*0.98*	1.072	*0.98*		
		Nom.	0.592	--	1.595	--	1.093	--		
		+5%	0.604	*1.02*	1.626	*1.02*	1.115	*1.02*		
		+10%	0.615	*1.04*	1.658	*1.04*	1.137	*1.04*		
	LP -; -; +	-10%	0.020	*0.95*	0.054	*0.95*	0.037	*0.95*		
		-5%	0.021	*1*	0.056	*0.98*	0.038	*0.97*		
		Nom.	0.021	--	0.057	--	0.039	--		
		+5%	0.022	*1.05*	0.058	*1.02*	0.040	*1.03*		
		+10%	0.022	*1.05*	0.059	*1.04*	0.040	*1.03*		
	Scheme *Pass; Inv-N; Inv-P; Read*	**V_{DD}**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read [0 1]**		**Read [1 0]**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*	Val	*Comp*
8T	SG 3 x t; t; t; t	-10%	0.569	*0.96*	3.458	*0.96*	2.495	*0.96*	2.202	*0.96*
		-5%	0.580	*0.98*	3.529	*0.98*	2.546	*0.98*	2.245	*0.98*
		Nom.	0.592	--	3.600	--	2.597	--	2.288	--
		+5%	0.604	*1.02*	3.671	*1.02*	2.649	*1.02*	2.331	*1.02*
		+10%	0.615	*1.04*	3.742	*1.04*	2.700	*1.04*	2.374	*1.04*
	LP_SGR -; -; +; t	-10%	0.020	*0.95*	0.054	*0.95*	0.519	*0.96*	0.227	*0.98*
		-5%	0.021	*1*	0.056	*0.98*	0.530	*0.98*	0.229	*0.99*
		Nom.	0.021	--	0.057	--	0.540	--	0.231	--
		+5%	0.022	*1.05*	0.058	*1.02*	0.551	*1.02*	0.233	*1.01*
		+10%	0.022	*1.05*	0.059	*1.04*	0.561	*1.04*	0.235	*1.02*
	LP_INV1 t; -; vdd; t	-10%	0.092	*0.96*	1.055	*0.96*	1.055	*0.96*	0.763	*0.97*
		-5%	0.094	*0.98*	1.077	*0.98*	1.077	*0.98*	0.776	*0.98*
		Nom.	0.096	--	1.099	--	1.099	--	0.789	--
		+5%	0.098	*1.02*	1.120	*1.02*	1.120	*1.02*	0.803	*1.02*
		+10%	0.100	*1.04*	1.142	*1.04*	1.142	*1.04*	0.816	*1.03*
	LP_INV1.2 t; -; +; t	-10%	0.020	*0.95*	0.984	*0.96*	0.984	*0.96*	0.691	*0.97*
		-5%	0.021	*1*	1.004	*0.98*	1.004	*0.98*	0.703	*0.98*
		Nom.	0.021	--	1.024	--	1.024	--	0.715	--
		+5%	0.022	*1.05*	1.044	*1.02*	1.044	*1.02*	0.727	*1.02*
		+10%	0.022	*1.05*	1.064	*1.04*	1.064	*1.04*	0.738	*1.03*

As expected, the read delays of the schemes decrease with increasing V_{DD}. The 8T schemes, as well as the 6T SG scheme, are affected identically, but only by ±6%, by changes in supply voltage since the read path involves two series SG-mode FinFETs. The 6T LP scheme is more greatly affected, by -11 to +18%, by changes in V_{DD} due to the use of LP-mode Pass transistors. The 8T SG scheme sees the least change in read energy, by ±11%, than the other schemes which see comparable changes in read energy due to variations in supply voltage. This is due, in large part, by similarities amongst the schemes in leakage current variations due to changes in V_{DD}.

The write delays for each scheme only slightly vary with changes in V_{DD}; these delays are already minimal compared to the read delay and vary by at most 1 ps. The schemes also see comparable variations in write energy with the 6T LP scheme seeing the greatest variation between -31% and +38%. Again, the similarities in leakage current variations yield similar variations in write energy for the schemes.

Leakage current is mildly affected by supply voltage variations. All six cells see a 4-5% reduction in leakage current due to a 10% reduction in V_{DD} and the cells see a 4-5% increase in leakage current due to a 10% increase in V_{DD}. This is also very similar for the read and write leakage. As a whole, the schemes leak within 5% of their nominal values for all read and write leakage scenarios.

The SNMs of 6T and 8T SG schemes vary within 4% of their nominal values. The other schemes use LP configuration for their Inv-N and Inv-P FinFETs whose on-currents are more greatly affected by changes in supply voltage. The SNMs of the other schemes vary within 10% of their nominal values. The 6T SG scheme's RSNM is only decreased by 1 mV for higher values of V_{DD}; this is less variation than this scheme saw for its SNM. The 6T LP scheme sees similar variation in its RSNM, -9% to +8%, as its SNM; this is again due to its use of LP-mode for its cross-coupled inverter FinFETs.

Overall, the 8T SG scheme has the least variation in average EDP, -12% to +15%, followed by the 6T LP cell at -17% to +19%, 8T LP_INV1.2 cell at ±19%, and the 8T LP_INV1 cell at -17% to 23%. As a whole, however, the 6T SG scheme, with average EDP variations between -21% and +36%, and the 8T LP_SGR scheme, with variations in average EDP between -23% and +28%, are still comparable to the other studied schemes. This is because all six cells see similar variations in leakage current due to changes in supply voltage. The 8T LP_ INV1.2 cell has smallest average EDP with low variation and the second-highest SNM, to the 8T LP_INV1 cell, with lowest variation at each value of V_{DD}.

4.3 Performance under Bias Voltage Variations

For bias voltage variation simulations, the reverse-bias value was studied for 18-22 mV (or -10% to +10% of the 0.2 V used throughout this research) for the four cells (6T LP, 8T LP_SGR, 8T LP_INV1, and 8T LP_INV1.2) which use LP-mode FinFETs. The n-type FinFET reverse-biases are -0.22 V to -0.18 V and the p-type FinFET reverse-biases are 1.18 V to 1.22 V. Table 4.6 shows the simulation results of bias voltage variations on the delays, energies, and average EDP of the 32×1024 arrays. Table 4.7 displays the simulation results of bias voltage variations on the leakage current, SNMs, and RSNMs of the SRAM cells. Table 4.8 summarizes the simulation results of bias voltage variations on the read and write leakage current totals.

Table 4.6. Bias voltage variation results of FinFET SRAM delay, energy, and EDP

	Scheme *Pass; Inv-N; Inv-P; Read*	**Bias**	**32 bits × 1024 words Array**											
			Delay (ps)				**Energy (fJ)**						**Ave* EDP (ps×fJ)**	
			Rd	*Comp*	Wr	*Comp*	Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.	32	--	5	--	2.4	--	9.3	--	3.8	--	120	--
	LP -; -; +	-10%	83	*0.99*	6	*1*	0.6	*1.20*	4.0	*1.03*	1.3	*1.08*	104	*1.01*
		-5%	84	*1*	6	*1*	0.6	*1.20*	4.0	*1.03*	1.2	*1*	103	*1*
		Nom.	84	--	6	--	0.5	--	3.9	--	1.2	--	103	--
		+5%	85	*1.01*	6	*1*	0.5	*1*	3.9	*1*	1.2	*1*	103	*1*
		+10%	86	*1.02*	6	*1*	0.5	*1*	3.9	*1*	1.2	*1*	102	*0.99*
8T	SG 3 x t; t; t; t	Nom.	32	--	3	--	0.9	--	6.9	--	2.1	--	66	--
	LP_SGR -; -; +; t	-10%	32	*1*	8	*1*	0.3	*1*	5.1	*1.02*	1.3	*1.08*	40	*1.03*
		-5%	32	*1*	8	*1*	0.3	*1*	5.0	*1*	1.2	*1*	39	*1*
		Nom.	32	--	8	--	0.3	--	5.0	--	1.2	--	39	--
		+5%	32	*1*	8	*1*	0.3	*1*	5.0	*1*	1.2	*1*	39	*1*
		+10%	32	*1*	8	*1*	0.3	*1*	5.0	*1*	1.2	*1*	39	*1*
	LP_INV1 t; -; vdd; t	-10%	32	*1*	4	*1*	0.4	*1*	3.3	*1.03*	1.0	*1.11*	30	*1*
		-5%	32	*1*	4	*1*	0.4	*1*	3.3	*1.03*	1.0	*1.11*	30	*1*
		Nom.	32	--	4	--	0.4	--	3.2	--	0.9	--	30	--
		+5%	32	*1*	4	*1*	0.4	*1*	3.2	*1*	0.9	*1*	30	*1*
		+10%	32	*1*	4	*1*	0.4	*1*	3.3	*1.03*	1.0	*1.11*	30	*1*
	LP_INV1.2 t; -; +; t	-10%	32	*1*	5	*1*	0.3	*1*	3.0	*1*	0.8	*1*	27	*1.04*
		-5%	32	*1*	5	*1*	0.3	*1*	3.0	*1*	0.8	*1*	26	*1*
		Nom.	32	--	5	--	0.3	--	3.0	--	0.8	--	26	--
		+5%	32	*1*	5	*1*	0.3	*1*	3.0	*1*	0.8	*1*	26	*1*
		+10%	32	*1*	5	*1*	0.3	*1*	3.0	*1*	0.8	*1*	26	*1*

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

Table 4.7. Bias voltage variation results of FinFET SRAM leakage and noise margins

	Scheme *Pass; Inv-N; Inv-P; Read*	**Bias**	**Leakage / Cell (nA)**		**SNM (mV)**		**RSNM (mV)**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.	0.592	--	360	--	124	--
	LP -; -; +	-10%	0.028	*1.33*	441	*1*	221	*1*
		-5%	0.024	*1.14*	441	*1*	222	*1*
		Nom.	0.021	--	442	--	222	--
		+5%	0.019	*0.90*	442	*1*	223	*1*
		+10%	0.016	*0.76*	443	*1*	223	*1*
8T	SG 3 x t; t; t; t	Nom.	0.592	--	360	--		
	LP_SGR -; -; +; t	-10%	0.028	*1.33*	441	*1*		
		-5%	0.024	*1.14*	441	*1*		
		Nom.	0.021	--	442	--		
		+5%	0.019	*0.90*	442	*1*		
		+10%	0.016	*0.76*	443	*1*		
	LP_INV1 t; -; vdd; t	-10%	0.101	*1.05*	443	*1*		
		-5%	0.098	*1.02*	443	*1*		
		Nom.	0.096	--	443	--		
		+5%	0.094	*0.98*	444	*1*		
		+10%	0.092	*0.96*	444	*1*		
	LP_INV1.2 t; -; +; t	-10%	0.028	*1.33*	441	*1*		
		-5%	0.024	*1.14*	441	*1*		
		Nom.	0.021	--	442	--		
		+5%	0.019	*0.90*	442	*1*		
		+10%	0.016	*0.76*	443	*1*		

Table 4.8. Bias voltage variation results of FinFET SRAM read/write leakage totals (nA)

	Scheme *Pass; Inv-N; Inv-P; Read*	**Bias**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read 1 [0 1] 1**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.	0.592	--	1.595	--	1.093	--
	LP -; -; +	-10%	0.028	*1.33*	0.074	*1.30*	0.051	*1.31*
		-5%	0.024	*1.14*	0.065	*1.14*	0.045	*1.15*
		Nom.	0.021	--	0.057	--	0.039	--
		+5%	0.019	*0.90*	0.049	*0.86*	0.034	*0.87*
		+10%	0.016	*0.76*	0.043	*0.75*	0.030	*0.77*

	Scheme *Pass; Inv-N; Inv-P; Read*	**Bias**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read [0 1]**		**Read [1 0]**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*	Val	*Comp*
8T	SG 3 x t; t; t; t	Nom.	0.592	--	3.600	--	2.597	--	2.288	--
	LP_SGR -; -; +; t	-10%	0.028	*1.33*	0.074	*1.30*	0.552	*1.02*	0.243	*1.05*
		-5%	0.024	*1.14*	0.065	*1.14*	0.546	*1.01*	0.237	*1.03*
		Nom.	0.021	--	0.057	--	0.540	--	0.231	--
		+5%	0.019	*0.90*	0.049	*0.86*	0.535	*0.99*	0.226	*0.98*
		+10%	0.016	*0.76*	0.043	*0.75*	0.531	*0.98*	0.222	*0.96*
	LP_INV1 t; -; vdd; t	-10%	0.101	*1.05*	1.104	*1*	1.104	*1*	0.795	*1.01*
		-5%	0.098	*1.02*	1.101	*1*	1.101	*1*	0.792	*1*
		Nom.	0.096	--	1.099	--	1.099	--	0.789	--
		+5%	0.094	*0.98*	1.096	*1*	1.096	*1*	0.787	*1*
		+10%	0.092	*0.96*	1.094	*1*	1.094	*1*	0.785	*0.99*
	LP_INV1.2 t; -; +; t	-10%	0.028	*1.33*	1.031	*1.01*	1.031	*1.01*	0.721	*1.01*
		-5%	0.024	*1.14*	1.027	*1*	1.027	*1*	0.718	*1*
		Nom.	0.021	--	1.024	--	1.024	--	0.715	--
		+5%	0.019	*0.90*	1.021	*1*	1.021	*1*	0.712	*1*
		+10%	0.016	*0.76*	1.019	*1*	1.019	*1*	0.710	*0.99*

The read delays of the 8T schemes are not affected by changes in bias voltages; this is expected since they use identical SG read transistor configurations. However, the 6T LP scheme is only minimally affected by changes in bias voltage as its read delay only varies by 3 ps. Since the affect on read delay is negligible, even the variations in leakage current do not cause variations in read energy. The 6T LP scheme only sees an increase of 0.1 fJ of read energy due to its slight increase in read delay and leakage current at smaller bias voltage values. The write delays for each scheme are not affected by bias voltage variations, and only slight increases in write energy are due to increases in leakage current at smaller bias voltage values.

Leakage current is most affected by bias voltage variations. At smaller bias voltage levels, the leakage current increases due to less reverse-biasing of LP-mode FinFETs, and vice versa. The 6T LP, 8T LP_SGR, and 8T LP_INV1.2 cells, which use a nominal -0.2 V BG bias for the Inv-N FinFETs and a nominal 1.2 V BG bias for the Inv-P FinFETs, see ±33% variation in leakage current. The 8T LP_INV1 cell, which uses a nominal -0.2 V BG bias for the Inv-N FinFETs and ties the back gates of the Inv-P FinFETs to V_{DD}, has leakage current variations within 5% of its nominal value due to less reliance on bias voltages. For read and write leakage, similar results are seen for the write 0 [0 1] 1 leakage. In fact, the 6T LP scheme sees similar variation in write and read leakage and the 8T LP_SGR scheme sees similar variation in its write 1 [0 1] 0 leakage; this is due to variations in the leakage current of their LP-configured Pass transistors. The 8T LP_INV1 and 8T LP_INV1.2 cells have similar and negligible variation in their write 1 [0 1] 0 leakage, but have larger write 1 [0 1] 0 leakage values, due to their SG-configured Pass transistors whose leakage does not vary due to bias voltage variations. Ideally, there should be no variation in read leakage for the 8T cells because SG-mode Read FinFETs are used; however, the read leakage of the 8T schemes sees only minimal variation due to slight variations of simultaneously-occurring write leakage caused by leftover values on the WBit and WNBit lines. Somewhat surprisingly, the variations in leakage current due to bias voltage variations do not cause variations in the SNMs or RSNMs for the cells. The ratios of leakage current of the Inv-N and Inv-P are minimally affected since both n-type and p-type FinFET BG biases either fall closer to ground and V_{DD} or rise farther from ground and V_{DD}.

Overall, variations in BG bias voltage negligibly affect the average EDP and noise margins of the four cells which use reverse-biased LP-mode FinFETs. At most, the 6T LP scheme sees a 2 ps×fJ variation in average EDP. The 8T LP_ INV1 cell sees the smallest variations in leakage current since only its Inv-N transistors use a bias voltage (the back gates of its Inv-P FinFETs are tied to V_{DD}). The other cells see ±33% variation in leakage current, but do not affect their noise margins and negligible affect their active energies and EDP.

4.4 Performance under Temperature Variations

For temperature variation simulations, the ambient temperature was studied for 0, 27, 50, 75, and 100°C for the six cells. Table 4.9 shows the simulation results of temperature variations on the delays, energies, and average EDP of the 32×1024 arrays. Table 4.10 displays the simulation results of temperature variations on the leakage current, SNMs, and RSNMs of the SRAM cells. Table 4.11 summarizes the simulation results of temperature variations on the read and write leakage current totals.

Table 4.9. Temperature variation results of FinFET SRAM delay, energy, and EDP

	Scheme *Pass; Inv-N;* *Inv-P; Read*	**Temp. (°C)**	**32 bits × 1024 words Array**											
			Delay (ps)				**Energy (fJ)**						**Ave* EDP (ps×fJ)**	
			Rd	*Comp*	Wr	*Comp*	Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*	Val	*Comp*
6T	SG t; t; t	0	31	*0.97*	5	*1*	1.5	*0.63*	9.2	*0.99*	3.0	*0.79*	92	*0.77*
		27	32	--	5	--	2.4	--	9.3	--	3.8	--	120	--
		50	35	*1.09*	5	*1*	4.1	*1.71*	10.5	*1.13*	5.4	*1.42*	187	*1.56*
		75	40	*1.25*	5	*1*	8.4	*3.50*	13.9	*1.49*	9.5	*2.50*	377	*3.14*
		100	45	*1.41*	6	*1.20*	17.5	*7.29*	21.9	*2.35*	18.4	*4.84*	827	*6.89*
	LP -; -; +	0	79	*0.94*	6	*1*	0.5	*1*	4.0	*1.03*	1.2	*1*	94	*0.91*
		27	84	--	6	--	0.5	--	3.9	--	1.2	--	103	--
		50	89	*1.06*	7	*1.17*	0.7	*1.40*	4.1	*1.05*	1.4	*1.17*	123	*1.19*
		75	94	*1.12*	7	*1.17*	1.2	*2.40*	4.6	*1.18*	1.9	*1.58*	178	*1.73*
		100	99	*1.18*	7	*1.17*	2.6	*5.20*	6.1	*1.56*	3.3	*2.75*	329	*3.19*
8T	SG 3 x t; t; t; t	0	30	*0.94*	3	*1*	0.4	*0.44*	6.4	*0.93*	1.6	*0.76*	48	*0.73*
		27	32	--	3	--	0.9	--	6.9	--	2.1	--	66	--
		50	36	*1.13*	3	*1*	2.2	*2.44*	8.3	*1.20*	3.4	*1.62*	121	*1.83*
		75	43	*1.34*	3	*1*	6.1	*6.78*	12.0	*1.74*	7.3	*3.48*	314	*4.76*
		100	51	*1.59*	3	*1*	15.4	*17.11*	21.5	*3.12*	16.6	*7.90*	845	*12.80*
	LP_SGR -; -; +; t	0	30	*0.94*	7	*0.88*	0.3	*1*	5.0	*1*	1.2	*1*	37	*0.95*
		27	32	--	8	--	0.3	--	5.0	--	1.2	--	39	--
		50	36	*1.13*	9	*1.13*	0.4	*1.33*	5.1	*1.02*	1.3	*1.08*	47	*1.21*
		75	43	*1.34*	9	*1.13*	0.6	*2.00*	5.3	*1.06*	1.6	*1.33*	68	*1.74*
		100	51	*1.59*	10	*1.25*	1.4	*4.67*	6.2	*1.24*	2.4	*2.00*	120	*3.08*
	LP_INV1 t; -; vdd; t	0	30	*0.94*	4	*1*	0.3	*0.75*	3.5	*1.09*	0.9	*1*	28	*0.93*
		27	32	--	4	--	0.4	--	3.2	--	0.9	--	30	--
		50	36	*1.13*	5	*1.25*	0.6	*1.50*	3.5	*1.09*	1.2	*1.33*	43	*1.43*
		75	43	*1.34*	5	*1.25*	1.4	*3.50*	4.2	*1.31*	2.0	*2.22*	86	*2.87*
		100	51	*1.59*	5	*1.25*	3.6	*9.00*	6.4	*2.00*	4.2	*4.67*	212	*7.07*
	LP_INV1.2 t; -; +; t	0	30	*0.94*	5	*1*	0.3	*1*	3.1	*1.03*	0.8	*1*	25	*0.96*
		27	32	--	5	--	0.3	--	3.0	--	0.8	--	26	--
		50	36	*1.13*	5	*1*	0.4	*1.33*	3.0	*1*	0.9	*1.13*	32	*1.23*
		75	43	*1.34*	5	*1*	0.6	*2.00*	3.1	*1.03*	1.1	*1.38*	48	*1.85*
		100	51	*1.59*	5	*1*	1.4	*4.67*	3.8	*1.27*	1.9	*2.38*	96	*3.69*

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

Table 4.10. Temperature variation results of FinFET SRAM leakage and noise margins

	Scheme *Pass; Inv-N; Inv-P; Read*	**Temp. (°C)**	**Leakage / Cell (nA)**		**SNM (mV)**		**RSNM (mV)**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*
6T	SG t; t; t	0	0.144	*0.24*	370	*1.03*	142	*1.15*
		27	0.592	--	360	--	124	--
		50	1.626	*2.75*	352	*0.98*	113	*0.91*
		75	4.116	*6.95*	345	*0.96*	104	*0.84*
		100	9.082	*15.34*	338	*0.94*	97	*0.78*
	LP -; -; +	0	0.004	*0.19*	448	*1.01*	228	*1.03*
		27	0.021	--	442	--	222	--
		50	0.074	*3.52*	436	*0.99*	218	*0.98*
		75	0.241	*11.48*	431	*0.98*	213	*0.96*
		100	0.671	*31.95*	425	*0.96*	209	*0.94*
8T	SG 3 x t; t; t; t	0	0.144	*0.24*	370	*1.03*		
		27	0.592	--	360	--		
		50	1.626	*2.75*	352	*0.98*		
		75	4.116	*6.95*	345	*0.96*		
		100	9.082	*15.34*	338	*0.94*		
	LP_SGR -; -; +; t	0	0.004	*0.19*	448	*1.01*		
		27	0.021	--	442	--		
		50	0.074	*3.52*	436	*0.99*		
		75	0.241	*11.48*	431	*0.98*		
		100	0.672	*32.00*	425	*0.96*		
	LP_INV1 t; -; vdd; t	0	0.021	*0.22*	450	*1.02*		
		27	0.096	--	443	--		
		50	0.289	*3.01*	438	*0.99*		
		75	0.815	*8.49*	432	*0.98*		
		100	1.995	*20.78*	426	*0.96*		
	LP_INV1.2 t; -; +; t	0	0.004	*0.19*	448	*1.01*		
		27	0.021	--	442	--		
		50	0.074	*3.52*	436	*0.99*		
		75	0.241	*11.48*	431	*0.98*		
		100	0.672	*32.00*	425	*0.96*		

Table 4.11. Temperature variation results of FinFET SRAM read/write leakage totals (nA)

	Scheme *Pass; Inv-N; Inv-P; Read*	**Temp. (°C)**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read 1 [0 1] 1**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*
6T	SG t; t; t	0	0.144	*0.24*	0.392	*0.25*	0.268	*0.25*
		27	0.592	--	1.595	--	1.093	--
		50	1.626	*2.75*	4.350	*2.73*	2.988	*2.73*
		75	4.114	*6.95*	10.920	*6.85*	7.517	*6.88*
		100	9.073	*15.33*	23.885	*14.97*	16.480	*15.08*
	LP -; -; +	0	0.004	*0.19*	0.010	*0.18*	0.007	*0.18*
		27	0.021	--	0.057	--	0.039	--
		50	0.074	*3.52*	0.196	*3.44*	0.135	*3.46*
		75	0.241	*11.48*	0.632	*11.09*	0.437	*11.21*
		100	0.671	*31.95*	1.753	*30.75*	1.212	*31.08*

	Scheme *Pass; Inv-N; Inv-P; Read*	**Temp. (°C)**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read [0 1]**		**Read [1 0]**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*	Val	*Comp*
8T	SG 3 x t; t; t; t	0	0.144	*0.24*	0.886	*0.25*	0.639	*0.25*	0.561	*0.25*
		27	0.592	--	3.600	--	2.597	--	2.288	--
		50	1.626	*2.75*	9.798	*2.72*	7.074	*2.72*	6.254	*2.73*
		75	4.115	*6.95*	24.530	*6.81*	17.724	*6.82*	15.725	*6.87*
		100	9.075	*15.33*	53.498	*14.86*	38.689	*14.90*	34.416	*15.04*
	LP_SGR -; -; +; t	0	0.004	*0.19*	0.010	*0.18*	0.131	*0.24*	0.053	*0.23*
		27	0.021	--	0.057	--	0.540	--	0.231	--
		50	0.074	*3.52*	0.196	*3.44*	1.497	*2.77*	0.677	*2.93*
		75	0.241	*11.48*	0.632	*11.09*	3.839	*7.11*	1.839	*7.96*
		100	0.671	*31.95*	1.753	*30.75*	8.614	*15.95*	4.341	*18.79*
	LP_INV1 t; -; vdd; t	0	0.021	*0.22*	0.268	*0.24*	0.268	*0.24*	0.191	*0.24*
		27	0.096	--	1.099	--	1.099	--	0.789	--
		50	0.289	*3.01*	3.013	*2.74*	3.013	*2.74*	2.193	*2.78*
		75	0.815	*8.49*	7.621	*6.93*	7.620	*6.93*	5.621	*7.12*
		100	1.995	*20.78*	16.806	*15.29*	16.803	*15.29*	12.530	*15.88*
	LP_INV1.2 t; -; +; t	0	0.004	*0.19*	0.251	*0.25*	0.251	*0.25*	0.173	*0.24*
		27	0.021	--	1.024	--	1.024	--	0.715	--
		50	0.074	*3.52*	2.798	*2.73*	2.798	*2.73*	1.978	*2.77*
		75	0.241	*11.48*	7.047	*6.88*	7.046	*6.88*	5.047	*7.06*
		100	0.672	*32.00*	15.483	*15.12*	15.480	*15.12*	11.207	*15.67*

The read delays of the schemes increase with increasing temperature; this is expected due to increasing resistivity with temperature. The 8T schemes are affected identically by temperature since they use the same SG read transistor configuration. It is interesting that the read delay for the 6T SG scheme experiences less of an increase, 41%, than the 8T schemes, 59%, for an ambient temperature of 100°C. However, the 6T SG scheme encounters a greater increase of 7.29X in read energy compared to the 8T LP_SGR and LP_INV1.2 cells' increase of 4.67X. This is due to decreased leakage current of the 8T LP_SGR and LP_INV1.2 schemes. The 8T LP_INV1 cell's read energy increases by 9.00X; this is greater than the 8T LP_SGR and LP_INV1.2 schemes due to increased leakage current at 100°C. The 8T SG scheme has comparable leakage current to the 6T SG scheme and sees a larger increase of 17.11X in read energy at 100°C. The 6T LP scheme has low leakage current and experiences a low increase of 5.2X, yet this increase is more than the 8T LP_SGR and LP_INV1.2 schemes due to a greater increase in its read delay at higher temperatures.

The write delays for each scheme only slightly vary with temperature increase; these delays are already minimal compared to the read delay and vary by at most 3 ps for the LP_SGR scheme. The schemes with lower leakage, namely the 6T LP, 8T LP_SGR, and 8T LP_INV1.2 cells, see less increase in write energy at 100°C (24-56%) than the 8T LP_INV1 cell (2.00X), 6T SG cell (2.35X), and 8T SG cell (3.12X).

Leakage current is most affected by temperature variations. At 100°C an increase of 15.34X is observed for the 6T and 8T SG schemes, an increase of 20.78X occurs for the 8T LP_INV1 cell, and increases of approximately 32X are witnessed for the 6T LP and 8T LP_SGR and LP_INV1.2 cells. The overall leakage of the low-power 6T LP, 8T LP_SGR, and 8T LP_INV1.2 schemes is less than one-tenth of the leakage of the SG schemes. For read and write leakage, similar results are seen for the write 0 [0 1] 1 leakage. For write 1 [0 1] 0 leakage, a roughly 15X increase at 100°C is seen for schemes with SG-configured pass transistors while a roughly 31X increase is observed for schemes with LP pass-transistors. For the 6T SG and LP schemes, the read 1 [0 1] 1 leakage increases comparably to their respective write leakage increases. For example, the write and read leakage of the 6T SG scheme increases by approximately 15X at 100°C and the 6T LP scheme's write and read leakage increases by approximately 31X at 100°C. For the 8T schemes, the read leakage, [0 1] and [1 0] cases, increases 15-19X at 100°C due to the SG-configured read transistors used by each scheme.

The SNMs of the six examined cells all vary within 6% of their nominal 27°C values. At higher temperatures, the noise margins decrease due to increased leakage current. Temperature causes greater variation for the RSNM for the 6T schemes. The 6T LP scheme's RSNM still varies within 6% of its nominal value, however, the 6T SG scheme's RSNM varies by 22% of its nominal value. This amount of variation would be a detriment to data fidelity under the effects of moderate changes in temperature.

Overall, the 6T LP scheme and the 8T LP_SGR and LP_ INV1.2 cells have the least variation in average EDP; a 3-4X increase at 100°C is witnessed for these schemes. The 6T SG scheme has an increase of 6.89X, the 8T LP_INV1 cell has an increase of 7.07X, and the 8T SG scheme has an increase of 12.80X in average EDP at 100°C. These schemes have larger leakage, leading to increased EDP and increased variation in EDP, and the 8T SG scheme has increased write 1 [0 1] 0 leakage leading to increased variation in EDP. The 8T LP_ INV1.2 cell has smallest average EDP with low variation and the second-highest SNM, to the 8T LP_INV1 cell, with lowest variation at each temperature.

4.5 Summary of FinFET SRAM PVT Variations

Overall, of the six examined cells, the 8T LP_INV1.2 cell performs the best under PVT variations. The 8T LP_INV1.2 cell has the lowest mean average EDP for a 32×1024 array and the second-largest mean SNM (only the 8T LP_INV1 cell has a 1 mV larger SNM) due to parameter variations. Bias voltage variations negligibly affect the average EDP and do not affect the noise margins of the four cells which use reverse-biased LP-mode FinFETs. The 8T LP_ INV1.2 cell has smallest average EDP with low variation and the second-highest SNM, to the 8T LP_INV1 cell, with lowest variation due to variations in supply voltage and temperature.

Chapter 5

FinFET SRAM Low-Leakage Modifications

Leakage current is very important in SRAM memories. At a given instant in a memory, the great majority of cells are not accessed, but draw leakage power. For a digital system, the leakage power of embedded memories can use a substantial portion of its power budget. A method to reduce leakage power in SRAM memories is through the use of header and/or footer transistors [22] [23]. Header transistors are p-type transistors which separate V_{DD} from the SRAM cells and supply them with a virtual V_{DD}. During normal read, write, and hold operation the header transistor is enabled. When in standby or sleep-mode, where the contents of the cells are disposable, the header transistors are disabled and create more resistance along the cross-coupled inverter leakage path. Footer transistors are n-type transistors which separate ground from the SRAM cells and supply them with a virtual ground. During normal read, write, and hold operation the footer transistor is enabled. When in sleep-mode the footer transistors are disabled and create more resistance along the cross-coupled inverter leakage path. A header transistor, footer transistor, or both can be used to limit leakage current. Header and footer transistors can also be used per SRAM cell or shared amongst two or more SRAM cells. Figure 5.1 displays an 8T SRAM cell with a p-type header transistor and an n-type footer transistor.

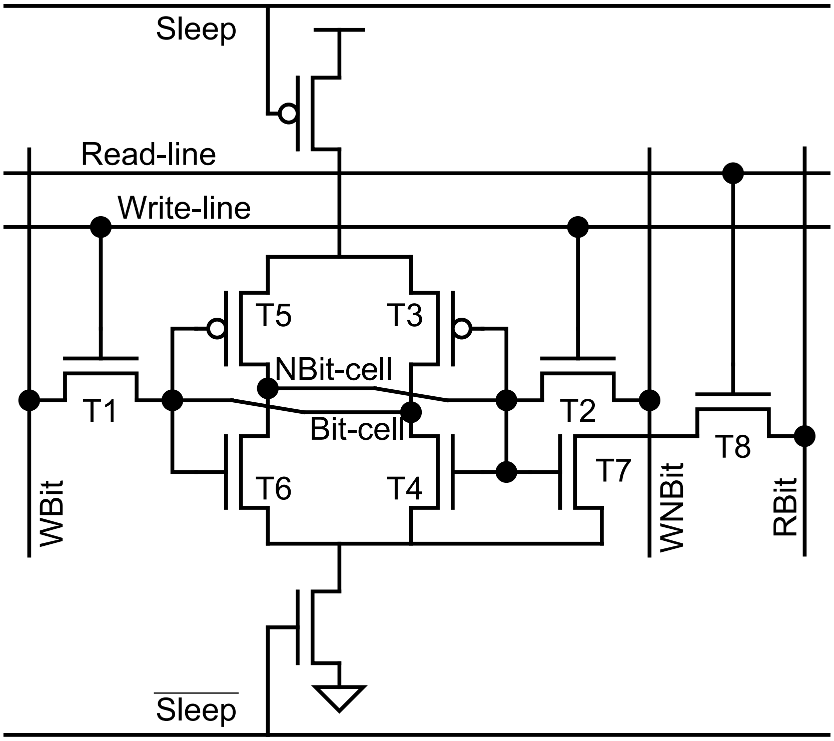


Figure 5.1. A header and a footer transistor on an 8T SRAM cell

The use of header and footer FinFETs has been examined for the six FinFET SRAM cells in this chapter. In addition to the choice of using header, footer, or both FinFETs, the header and footer FinFETs can be in SG-configuration, LP-configuration with a 0 V n-bias and a 1 V p-bias, or LP-configuration with a -0.2 V n-bias and a 1.2 V (V_{DD} + 0.2 V) p-bias. The following sections present the performance results when using header and footer FinFETs per SRAM cell, per two SRAM cells, and per four SRAM cells.

5.1 Header/Footer FinFETs per Cell

Table 5.1 shows the best leakage, delay, and average EDP results of using header and footer FinFETs per SRAM cell and Table 5.2 shows the static noise margin (SNM) and read static noise margin (RSNM) results of using header and footer FinFETs per SRAM cell.

Appendix C includes more results of lower-performing cells. If there are a small number of 32×1024 arrays in the memory, then the percent comparisons of the 32×1024 array average EDP are approximate to the percent comparisons of the average EDP for the entire SRAM memory. On the other hand, if there are a large number of 32×1024 arrays in the memory, then the percent comparisons of the leakage EDP per array in sleep-mode are approximate to the percent comparisons of the average EDP of the entire SRAM memory.

Overall, the best performing cells often use only a header transistor per SRAM cell. The leakage EDP per array in sleep-mode is reduced by up to 92% for the 6T SG and 8T SG cells (using header and footer transistors), 79% for the 8T LP_INV1 cell (using header transistors), 26% for the 6T LP cell (using header and footer transistors), and 10% for the 8T LP_SGR and 8T LP_INV1.2 cells (using header transistors). The cells with higher leakage, due to less reverse-biasing of the cross-coupled inverter FinFETs, benefit the most from header and footer transistors. However, there are tradeoffs of using header and footer transistors to reduce leakage current. The additional transistors increase the area requirement of the SRAM and require a sleep-mode signal, which in turn requires additional area and additional energy to generate. The delays of the SRAM array also increase for the read and write operations, causing the average EDP of the 32×1024 arrays to often increase; this limits the effectiveness of header and footer transistors for small SRAM memories with few arrays. Additionally, most cells see a reduction in SNM and RSNM with the inclusion of these transistors; up to a 7% reduction in SNM and up to a 71% in RSNM is observed.

Table 5.1. Leakage, delay, and EDP results of header/footer FinFETs per SRAM cell

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP+	32 bits × 1024 words Array					
										Delay (ps)				Ave* EDP (ps×fJ)	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	5	--	120	--
		Header	t	0.592	0.536	*0.91*	18.0	*0.91*	*1*	32	*1*	5	*1*	98	*0.82*
		Footer	t	0.592	0.283	*0.48*	18.0	*0.90*	*35*	44	*1.38*	4	*0.80*	184	*1.53*
		Hdr + Ftr	t	0.592	0.227	*0.38*	14.4	*0.72*	*9*	44	*1.38*	5	*1*	162	*1.35*
		Header	vdd	0.592	0.533	*0.90*	17.9	*0.90*	*1*	32	*1*	6	*1.20*	88	*0.73*
		Header	+	0.592	0.504	*0.85*	16.9	*0.85*	*1*	32	*1*	6	*1.20*	86	*0.72*
		Hdr + Ftr	+ / -	0.592	0.014	*0.02*	1.7	*0.08*	*9*	60	*1.88*	6	*1.20*	260	*2.17*
	LP -; -; +	Nom.		0.021	0.021	--	4.9	--		84	--	6	--	103	--
		Header	t	0.021	0.021	*1*	4.9	*1*	*1*	84	*1*	6	*1*	93	*0.90*
		Header	vdd	0.021	0.021	*1*	4.9	*1*	*1*	84	*1*	7	*1.17*	88	*0.85*
		Header	+	0.021	0.019	*0.90*	4.4	*0.90*	*1*	84	*1*	7	*1.17*	87	*0.84*
		Hdr + Ftr	+ / -	0.021	0.008	*0.38*	3.6	*0.74*	*35*	117	*1.39*	7	*1.17*	145	*1.41*
8T	SG 3 x t; t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	3	--	66	--
		Hdr + Ftr	t	0.592	0.227	*0.38*	14.4	*0.72*	*10*	44	*1.38*	4	*1.33*	115	*1.74*
		Header	vdd	0.592	0.533	*0.90*	17.9	*0.90*	*1*	32	*1*	5	*1.67*	51	*0.77*
		Header	+	0.592	0.504	*0.85*	16.9	*0.85*	*1*	32	*1*	6	*2.00*	50	*0.76*
		Footer	-	0.591	0.102	*0.17*	11.6	*0.59*	*20*	59	*1.84*	3	*1*	219	*3.32*
		Hdr + Ftr	+ / -	0.591	0.014	*0.02*	1.6	*0.08*	*9*	59	*1.84*	6	*2.00*	201	*3.05*
	LP_SGR -; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	8	--	39	--
		Header	vdd	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	10	*1.25*	33	*0.85*
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	32	*1*	10	*1.25*	33	*0.85*
	LP_INV1 t; -; vdd; t	Nom.		0.096	0.096	--	3.2	--		32	--	4	--	30	--
		Header	vdd	0.096	0.048	*0.50*	1.6	*0.50*	*1*	32	*1*	6	*1.50*	26	*0.87*
		Header	+	0.096	0.020	*0.21*	0.7	*0.21*	*1*	32	*1*	6	*1.50*	26	*0.87*
	LP_INV1.2 t; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	5	--	26	--
		Header	t	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	5	*1*	23	*0.88*
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	32	*1*	6	*1.20*	23	*0.88*

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

+ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table 5.2. Noise margin results of header/footer FinFETs per SRAM cell

	Scheme *Pass; Inv-N; Inv-P; Read*	**Leak. Tran.**	**Leak. Tran. Config.**	**Leak. / Cell (nA)**	**Leakage / Cell in Sleep-Mode (nA)**		**Leakage EDP / Array in Sleep-Mode (ps×fJ)**		**# Arrays for Break-Even EDP^{+}**	**SNM (mV)**		**RSNM (mV)**	
				Val	Val	*Comp*	Val	*Comp*		Val	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.592	0.592	--	19.9	--		360	--	124	--
		Header	t	0.592	0.536	*0.91*	18.0	*0.91*	*1*	343	*0.95*	87	*0.70*
		Footer	t	0.592	0.283	*0.48*	18.0	*0.90*	*35*	357	*0.99*	36	*0.29*
		Hdr + Ftr	t	0.592	0.227	*0.38*	14.4	*0.72*	*9*	343	*0.95*	65	*0.52*
		Header	vdd	0.592	0.533	*0.90*	17.9	*0.90*	*1*	336	*0.93*	74	*0.60*
		Header	+	0.592	0.504	*0.85*	16.9	*0.85*	*1*	334	*0.93*	71	*0.57*
		Hdr + Ftr	+ / -	0.592	0.014	*0.02*	1.7	*0.08*	*9*	335	*0.93*	131	*1.06*
	LP -; -; +	Nom.		0.021	0.021	--	4.9	--		442	--	222	--
		Header	t	0.021	0.021	*1*	4.9	*1*	*1*	439	*0.99*	220	*0.99*
		Header	vdd	0.021	0.021	*1*	4.9	*1*	*1*	438	*0.99*	218	*0.98*
		Header	+	0.021	0.019	*0.90*	4.4	*0.90*	*1*	438	*0.99*	218	*0.98*
		Hdr + Ftr	+ / -	0.021	0.008	*0.38*	3.6	*0.74*	*35*	438	*0.99*	175	*0.79*
8T	SG 3 x t; t; t; t	Nom.		0.592	0.592	--	19.9	--		360	--		
		Hdr + Ftr	t	0.592	0.227	*0.38*	14.4	*0.72*	*10*	343	*0.95*		
		Header	vdd	0.592	0.533	*0.90*	17.9	*0.90*	*1*	336	*0.93*		
		Header	+	0.592	0.504	*0.85*	16.9	*0.85*	*1*	334	*0.93*		
		Footer	-	0.591	0.102	*0.17*	11.6	*0.59*	*20*	356	*0.99*		
		Hdr + Ftr	+ / -	0.591	0.014	*0.02*	1.6	*0.08*	*9*	335	*0.93*		
	LP_SGR -; -; +; t	Nom.		0.021	0.021	--	0.7	--		442	--		
		Header	vdd	0.021	0.021	*1*	0.7	*1*	*1*	438	*0.99*		
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	438	*0.99*		
	LP_INV1 t; -; vdd; t	Nom.		0.096	0.096	--	3.2	--		443	--		
		Header	vdd	0.096	0.048	*0.50*	1.6	*0.50*	*1*	441	*1*		
		Header	+	0.096	0.020	*0.21*	0.7	*0.21*	*1*	440	*0.99*		
	LP_INV1.2 t; -; +; t	Nom.		0.021	0.021	--	0.7	--		442	--		
		Header	t	0.021	0.021	*1*	0.7	*1*	*1*	439	*0.99*		
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	438	*0.99*		

$^{+}$ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

5.2 Header/Footer FinFETs per Two Cells

Table 5.3 shows the best leakage, delay, and average EDP results of using header and footer FinFETs per two SRAM cells. The SNM and RSNM results are identical to the cells using header and footer transistors per one SRAM cell. Appendix C includes more results of lower-performing cells.

Similar to the previous section, the best performing cells often use only a header transistor per two SRAM cells. The leakage EDP per array in sleep-mode is reduced by up to

24% for the 6T SG and 8T SG cells (using header transistors), 80% for the 8T LP_INV1 cell (using header transistors), 16% for the 6T LP cell (using header and footer transistors), and 10% for the 8T LP_SGR and 8T LP_INV1.2 cells (using header transistors). These are slightly less percent reductions compared to using header and footer FinFETs per cell, but this is due to the increased read and write delays of the cells due to sharing a header or footer transistor. In addition, while the additional transistors increase the area requirement of the SRAM, sharing header and footer transistors between two cells halves the number of additional FinFETs required.

Table 5.3. Leakage, delay, and EDP results of header/footer FinFETs per two SRAM cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP$^+$	32 bits × 1024 words Array					
										Delay (ps)				Ave* EDP (ps×fJ)	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	5	--	120	--
		Hdr + Ftr	t	0.592	0.141	*0.24*	15.0	*0.76*	*26*	57	*1.78*	6	*1.20*	238	*1.98*
		Header	vdd	0.592	0.520	*0.88*	17.4	*0.88*	*1*	32	*1*	8	*1.60*	79	*0.66*
		Header	+	0.592	0.503	*0.85*	16.9	*0.85*	*1*	32	*1*	8	*1.60*	77	*0.64*
	LP -; -; +	Nom.		0.021	0.021	--	4.9	--		84	--	6	--	103	--
		Header	vdd	0.021	0.021	*1*	4.9	*1*	*1*	84	*1*	8	*1.33*	80	*0.78*
		Header	+	0.021	0.019	*0.90*	4.4	*0.90*	*1*	84	*1*	8	*1.33*	79	*0.77*
		Hdr + Ftr	+ / -	0.021	0.005	*0.24*	4.1	*0.84*	*130*	158	*1.88*	8	*1.33*	201	*1.95*
8T	SG 3 x t; t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	3	--	66	--
		Hdr + Ftr	t	0.591	0.141	*0.24*	14.5	*0.73*	*23*	56	*1.75*	6	*2.00*	182	*2.76*
		Header	vdd	0.592	0.520	*0.88*	17.4	*0.88*	*1*	32	*1*	7	*2.33*	47	*0.71*
		Header	+	0.592	0.503	*0.85*	16.9	*0.85*	*1*	32	*1*	8	*2.67*	46	*0.70*
	LP_SGR -; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	8	--	39	--
		Header	vdd	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	13	*1.63*	30	*0.77*
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	32	*1*	13	*1.63*	30	*0.77*
	LP_INV1 t; -; vdd; t	Nom.		0.096	0.096	--	3.2	--		32	--	4	--	30	--
		Header	vdd	0.096	0.036	*0.38*	1.2	*0.38*	*1*	32	*1*	7	*1.75*	25	*0.83*
		Header	+	0.096	0.019	*0.20*	0.6	*0.20*	*1*	32	*1*	8	*2.00*	25	*0.83*
	LP_INV1.2 t; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	5	--	26	--
		Header	vdd	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	7	*1.40*	22	*0.85*
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	32	*1*	8	*1.60*	21	*0.81*

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

$^+$ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

5.3 Header/Footer FinFETs per Four Cells

Table 5.4 shows the best leakage, delay, and average EDP results of using header/footer FinFETs per four SRAM cells. SNM/RSNM results are identical to cells using header and footer transistors per one SRAM cell. Appendix C includes more results of lower-performing cells.

Similar to the previous sections, the best performing cells often use only a header transistor per four SRAM cells. The leakage EDP per array in sleep-mode is reduced by up to 74% for the 6T SG cell (using header and footer transistors), 20% for the 8T SG cell (using header transistors), 81% for the 8T LP_INV1 cell (using header transistors), 14% for the 6T LP, 8T LP_SGR, and 8T LP_INV1.2 cells (using header transistors). These are similar percent reductions compared to using header and footer FinFETs per two cells. In addition, while the additional transistors increase the area requirement of the SRAM, sharing header and footer transistors between four cells quarters the number of additional FinFETs required.

Table 5.4. Leakage, delay, and EDP results of header/footer FinFETs per four SRAM cells

	Scheme *Pass; Inv-N;* *Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP^{+}	32 bits × 1024 words Array					
										Delay (ps)				Ave* EDP (ps×fJ)	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	5	--	120	--
		Header	vdd	0.592	0.512	*0.86*	17.2	*0.86*	*1*	32	*1*	11	*2.20*	62	*0.52*
		Header	+	0.592	0.502	*0.85*	16.8	*0.85*	*1*	32	*1*	12	*2.40*	66	*0.55*
		Hdr + Ftr	+ / -	0.592	0.004	*0.01*	5.2	*0.26*	*104*	200	*6.25*	12	*2.40*	1614	*13.45*
	LP -; -; +	Nom.		0.021	0.021	--	4.9	--		84	--	6	--	103	--
		Header	vdd	0.021	0.020	*0.95*	4.6	*0.95*	*1*	84	*1*	10	*1.67*	75	*0.73*
		Header	+	0.021	0.018	*0.86*	4.2	*0.86*	*1*	84	*1*	11	*1.83*	74	*0.72*
8T	SG 3 x t; t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	3	--	66	--
		Header	vdd	0.592	0.512	*0.86*	17.2	*0.86*	*1*	32	*1*	11	*3.67*	43	*0.65*
		Header	+	0.592	0.502	*0.85*	15.8	*0.80*	*1*	31	*1*	12	*4.00*	42	*0.64*
	LP_SGR -; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	8	--	39	--
		Header	vdd	0.021	0.020	*0.95*	0.7	*1*	*1*	32	*1*	19	*2.38*	28	*0.72*
		Header	+	0.021	0.018	*0.86*	0.6	*0.86*	*1*	32	*1*	21	*2.63*	27	*0.69*
	LP_INV1 t; -; vdd; t	Nom.		0.096	0.096	--	3.2	--		32	--	4	--	30	--
		Header	vdd	0.096	0.028	*0.29*	0.9	*0.29*	*1*	32	*1*	11	*2.75*	23	*0.77*
		Header	+	0.096	0.018	*0.19*	0.6	*0.19*	*1*	32	*1*	12	*3.00*	23	*0.77*
	LP_INV1.2 t; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	5	--	26	--
		Header	vdd	0.021	0.020	*0.95*	0.7	*1*	*1*	32	*1*	11	*2.20*	20	*0.77*
		Header	+	0.021	0.018	*0.86*	0.6	*0.86*	*1*	32	*1*	12	*2.40*	20	*0.77*

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

$^{+}$ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

5.4 Summary of SRAM Usage of Header/Footer FinFETs

Overall, the cells with higher leakage benefit the most from header and footer transistors since they see the largest reductions in leakage current, however, a more efficient method to reduce leakage power is to use SRAM cells, such as the LP cells, which reverse-bias the cross-coupled inverter FinFETs in order to reduce leakage. The only case when using header or footer transistors saves more leakage EDP than LP cells is when using LP-mode header and footer transistors per cell with 1.2 V and -0.2 V biases for the 6T SG cell; this saves 91% of leakage EDP (compared to 75% savings for the 6T LP cell) but has worse active EDP (2.17X more than nominal 6T SG cell), lower SNM (7% worse than the nominal 6T SG cell), and lower RSNM than the 6T LP cell. All LP 8T cells yield less leakage EDP than using header or footer FinFETs for the 8T SG cell.

Sharing header and footer transistors amongst multiple SRAM cells can lead to more leakage current savings, but also causes slower read and write speeds; this eliminate the effectiveness of some cells if the increase in active operation EDP far outpaces the reduction in leakage current. Another drawback of header and footer transistors is that they increase the area requirement of the SRAM memories; for designs requiring minimal die footprints, an increase in memory area may not be practical. Additionally, most cells see reduced noise margins with the inclusion of header and footer FinFETs; this can create memory fidelity issues for applications that require SRAMs with high resistance to noise.

Chapter 6

Near-Threshold FinFET SRAM Operation

Near-threshold operation of SRAMs can be used as a method to obtain additional savings in energy and leakage current. In [24], Dreslinski et. al. propose circuit operation in a region near the threshold voltage of the devices. Compared to full-V_{DD} operation, operating in a near-threshold region can save about 10X energy while only suffering from an approximate 10X increased delay [24]. Compared to near-threshold operation, operating in a sub-threshold region can save about 2X energy but suffers from an approximate 50-100X increased delay [24]. Thus, near-threshold operation can deliver a majority of the energy savings offered by sub-threshold operation, but with less of an increase in delay.

Figure 6.1 shows the relationship between delay, average energy (using 80% read and 20% write energy), and V_{DD} value for a 32×1024 FinFET 8T SRAM array using the SG scheme with normal 0 V to 1 V read-line swing. As the supply voltage decreases, the delays increase and the average energy decreases until the supply voltage enters the sub-threshold region. In the figure, once V_{DD} drops below 0.5 V, the average energy increases due to the substantial increases of the read delay (the worst-case delay). The average energy delay product (EDP) is superimposed on the graph of Figure 6.1. The optimal operating point, in terms of average EDP, is at the minimum of the average EDP curve on the plot. For this scheme, this occurs near 0.6 V.

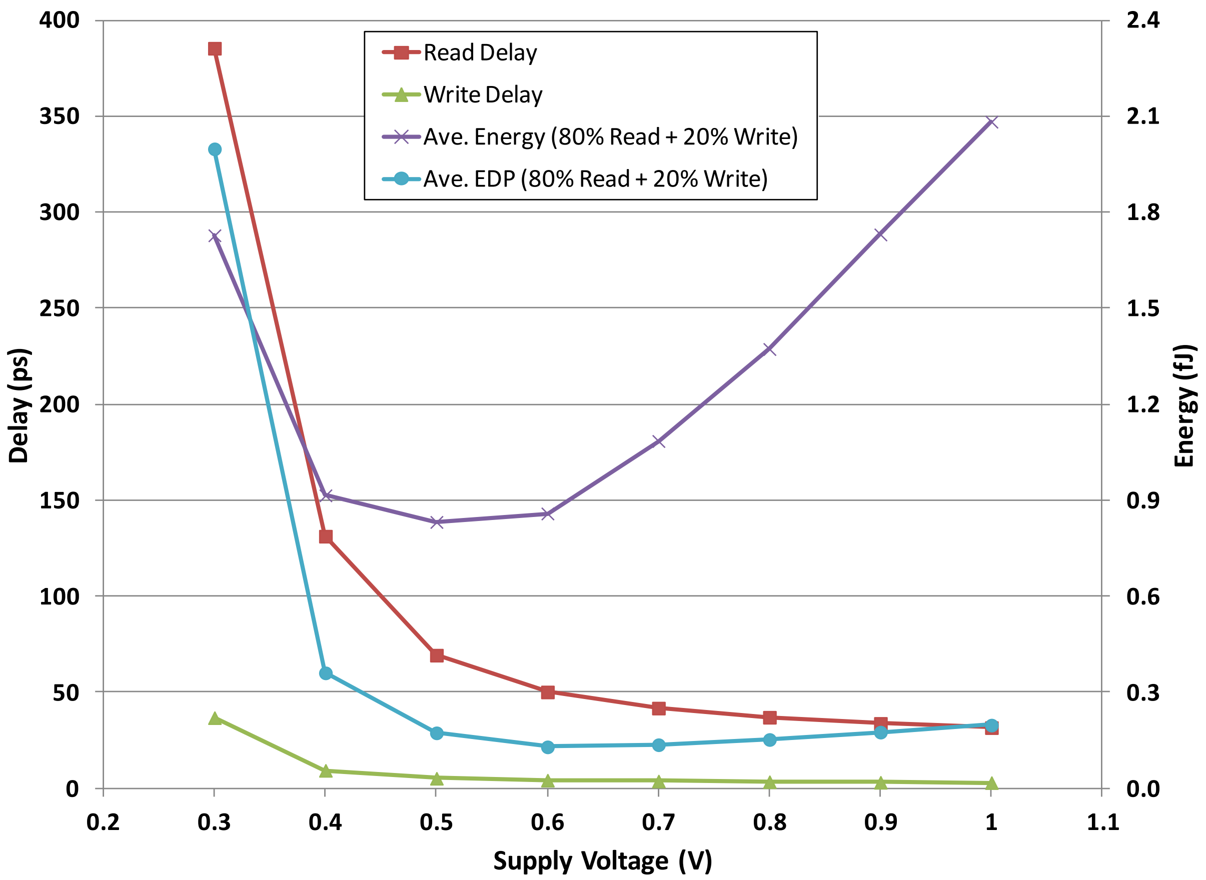


Figure 6.1. Delay, average energy, and average EDP versus V_{DD} value of 8T SG SRAM cell

However, there are barriers to near-threshold operation; three potential barriers are discussed by [24], these are: performance loss, increased parameter variation, and increased functional failure. A downside to using near-threshold operation, and similarly a problem for sub-threshold operation, for SRAMs is that the static noise margins (SNMs) and read static noise margins (RSNMs) will be smaller for smaller V_{DD} values; this can lead to increased SRAM failure and lower yield. In addition, write-ability or write static noise margin (WSNM), can also suffer due to less current drive from the Pass transistors T1 and T2. Another issue for near-threshold operation of FinFET SRAM cells is the impact of parameter variations and supply voltage variations. The performance and noise margins of the SRAMs are more affected by these variations when operating at a low supply voltage.

There is much research available about sub-threshold circuits, including SRAMs, however, there is much less research available about near-threshold SRAMs. A few researchers have presented work on near-threshold SRAM performance while taking yield and noise margins into consideration [25] [26] [27]. FinFETs may outperform planar devices for near-threshold and sub-threshold operation due to having a better I_{on}/I_{off} ratio even at these lower supply voltages [28] [29]. In [29], bulk-FinFETs are used but SOI FinFETs may also continue to outperform planar devices at lower values of V_{DD}. There is some research available on sub-threshold FinFET SRAMs. 6T SRAM cells are studied in [30] and 10T Schmitt Trigger SRAM cells are studied in [31]. Both take advantage of FinFET SG and LP transistor configurations for improved performance and noise margins; this is a promising result since this can also be done for near-threshold operation. Prior to this study, no research has been performed on near-threshold FinFET 8T SRAM cells.

In this chapter, near-threshold operation of FinFET 6T and 8T SRAMs are studied. The following subsections present the near-threshold operation SRAM performance results, speed enhancements, PVT variations, and low-leakage modifications.

6.1 SRAM Performance Results

All 6T SRAM cells from Chapter 2 and all 8T SRAM cells from Chapter 3 were simulated using decreased values of V_{DD}. Figure 6.2 shows the average EDP (for a 32×1024 FinFET 6T SRAM array) for all of the 6T SRAM cells across decreasing values of the supply voltage. For a majority of the schemes, the optimal value of V_{DD} to minimize EDP is near 0.8 V; however, for the better-performing (in terms of EDP) schemes such as 6T SG and 6T LP_INVP, supply voltages as low as 0.6 V minimize EDP.

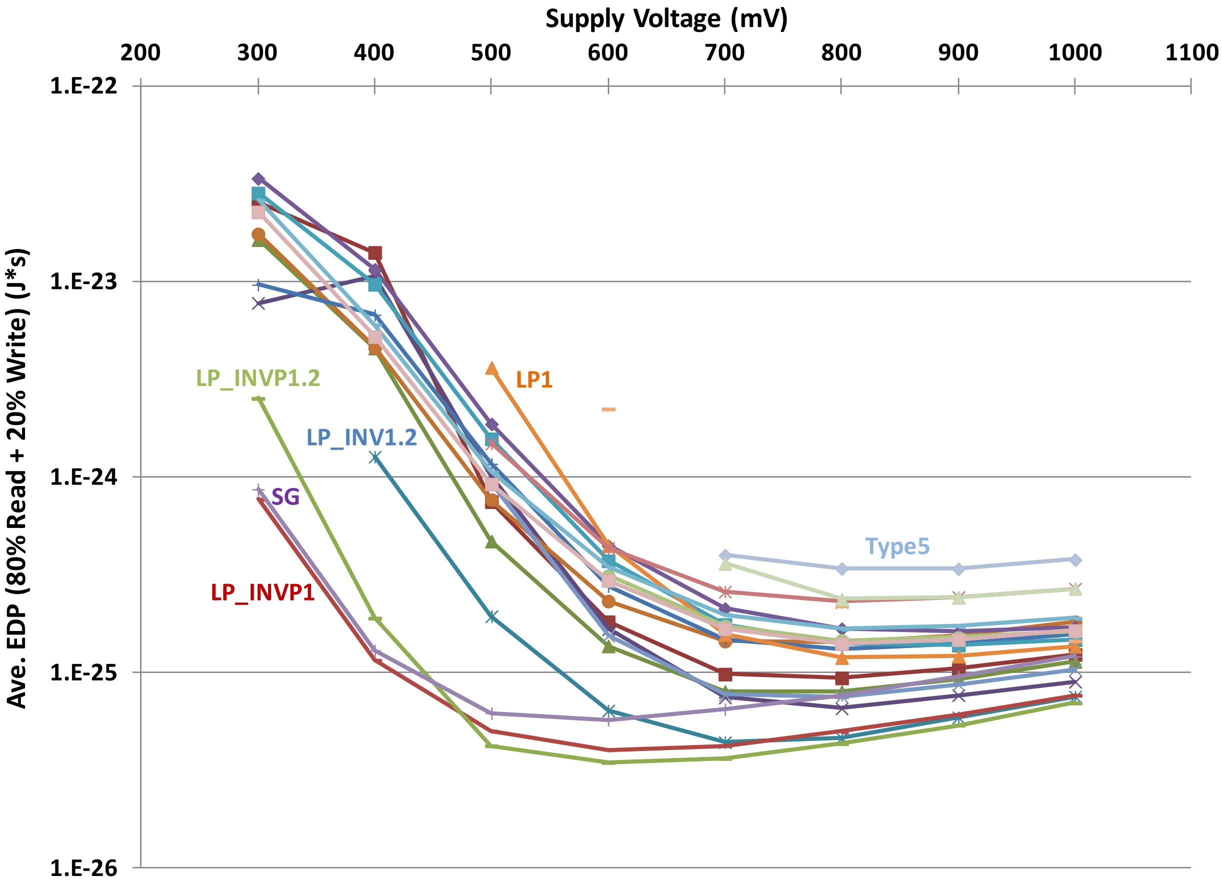


Figure 6.2. Average EDP versus V_{DD} value for 6T SRAM cells

Noise margins must also be considered for near-threshold operation; Figure 6.3 and Figure 6.4 show the SNMs and RSNMs, respectively, for all of the 6T SRAM cells across decreasing values of the supply voltage. The SNM decreases, as a percentage of V_{DD}, as the supply voltage decreases for all of the 6T SRAM cells but the 6T SG, 6T Type5, 6T PGFB [15], and 6T PUWG [15] schemes. The behavior of the 6T SRAM cells' RSNMs is much more erratic. The RSNMs, as a percentage of V_{DD}, of most 6T schemes either decreased slightly with decreasing supply voltage or roughly maintained its nominal value. The largest changes in RSNM/V_{DD} is seen for the 6T Type5 (8%), 6T Type6 (8%), 6T IG2 [14] (8%), 6T LP with 0 V Inv-N back gate (BG) bias and 1.2 V Inv-P BG bias (13%), 6T IG1 [14] (8%), and 6T PUWG [15] (19%) cells.

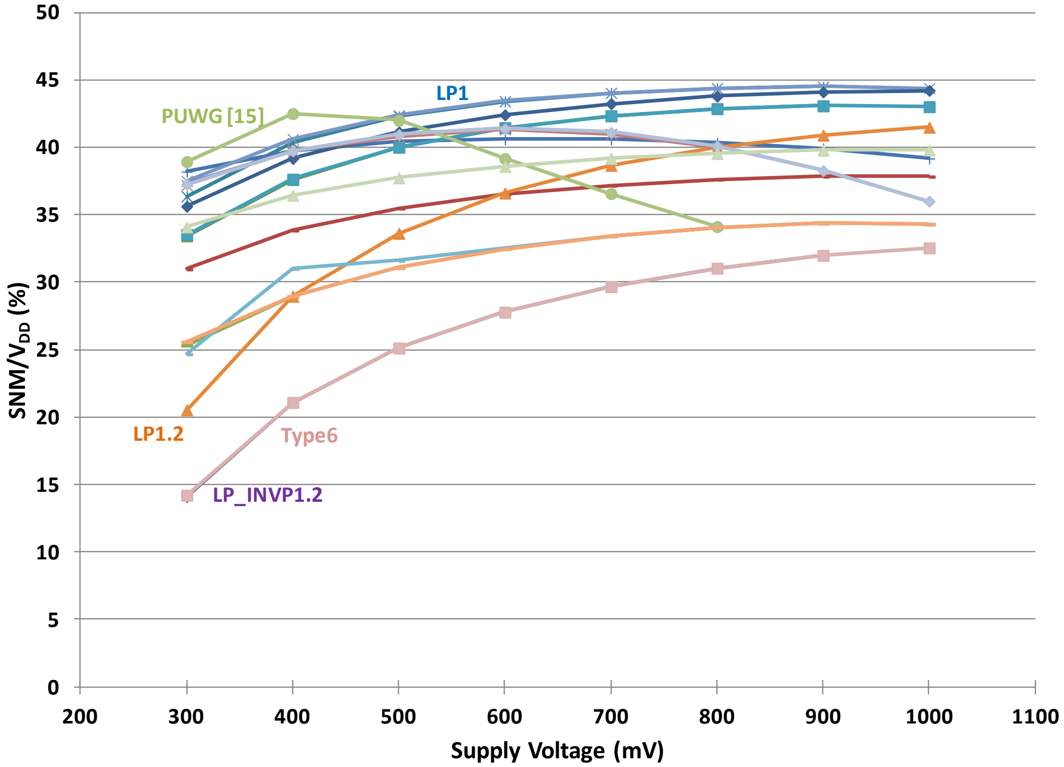


Figure 6.3. SNM versus V_{DD} value for 6T SRAM cells

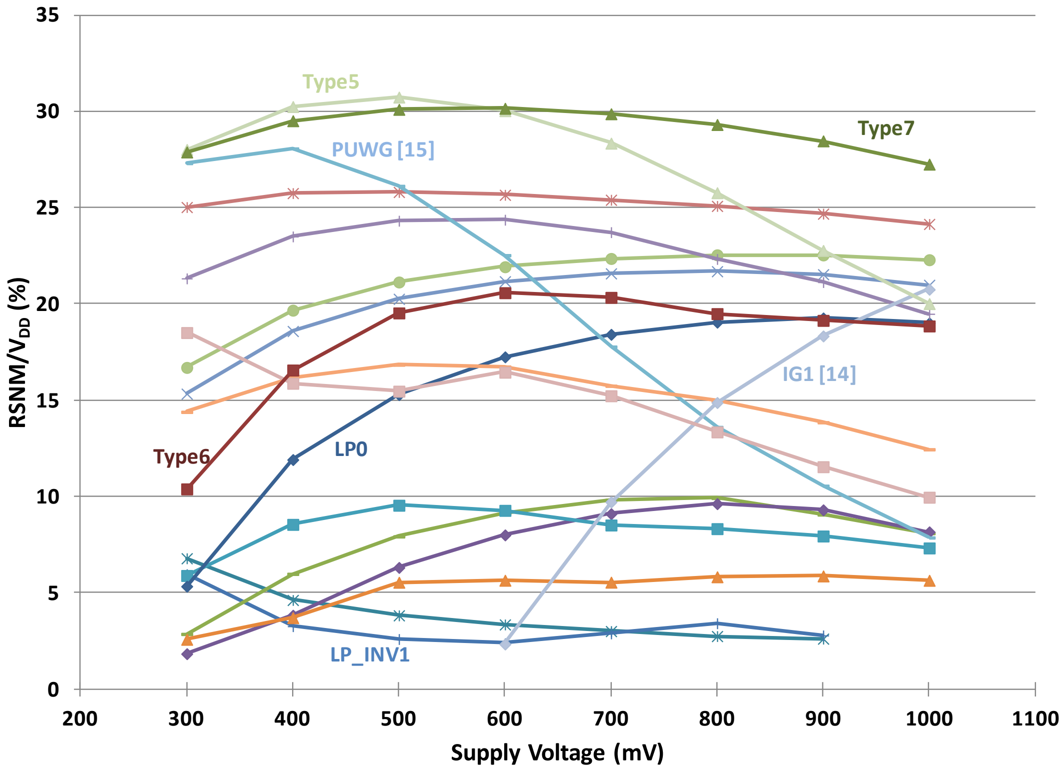


Figure 6.4. RSNM versus V_{DD} value for 6T SRAM cells

Figure 6.5 shows the average EDP (for a 32×1024 FinFET 8T SRAM array) for all of the 8T SRAM cells across decreasing values of the supply voltage. The optimal supply voltage, to minimize EDP, ranges between 0.5 V to 0.6 V for these schemes. The optimal supply voltage can be set as low as 0.5 V for the better-performing (in terms of EDP) schemes that either use a -0.2 V BG bias for the Inv-N FinFETs and SG-configure the Pass FinFETs (the LP_INV and LP_INVN schemes) or use boost the logic "1" value of the read-line to 1.2 V. Read-line boosting, a method to increase the read speed, will be explored further in the next subsection.

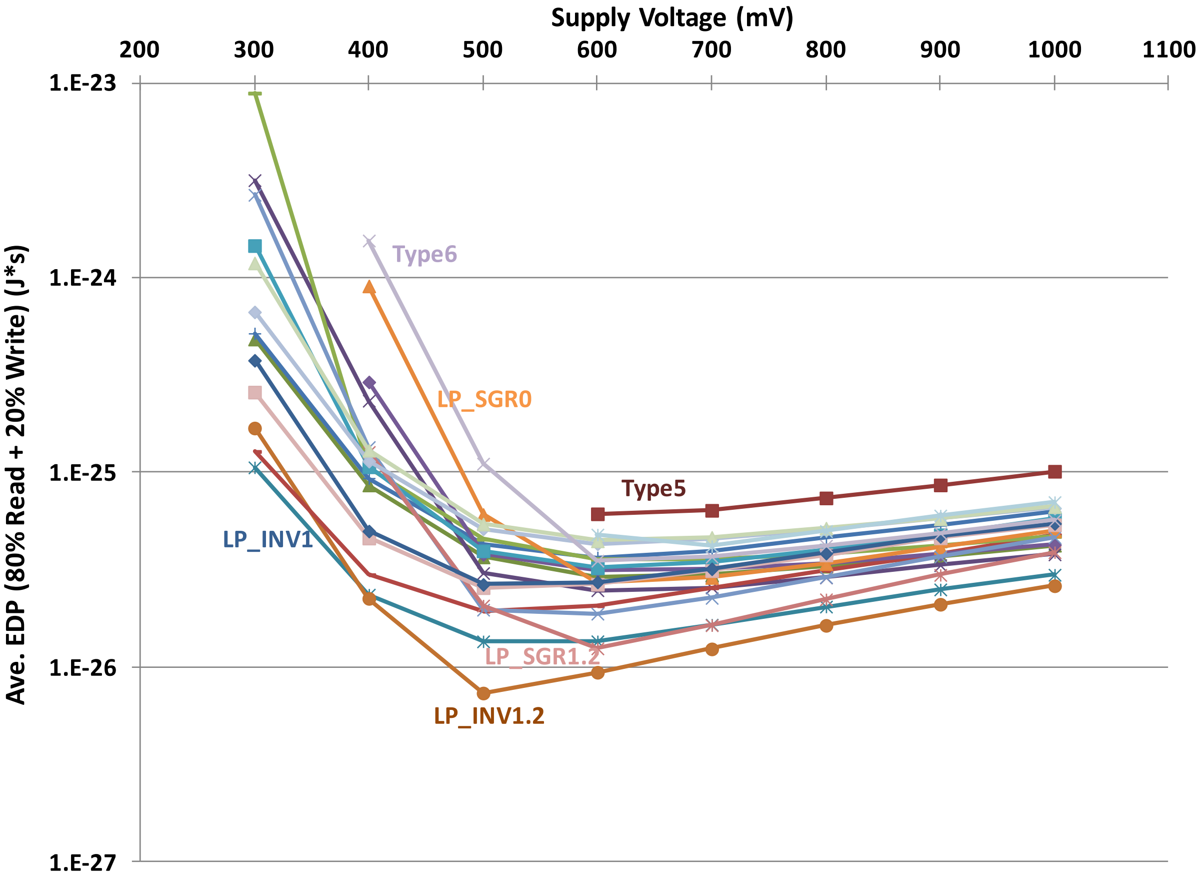


Figure 6.5. Average EDP versus V_{DD} value for 8T SRAM cells

Figure 6.6 shows the SNMs for all of the 8T SRAM cells across decreasing values of the supply voltage. This figure is very similar to the 6T SRAM SNMs shown in Figure 6.3 since a majority of the 8T SRAM cells share the same cross-coupled inverter configurations. Similar to the 6T SRAM cells, the SNM decreases, as a percentage of V_{DD}, as the supply voltage decreases for all of the 8T SRAM cells but the 8T SG and 8T Type5 schemes.

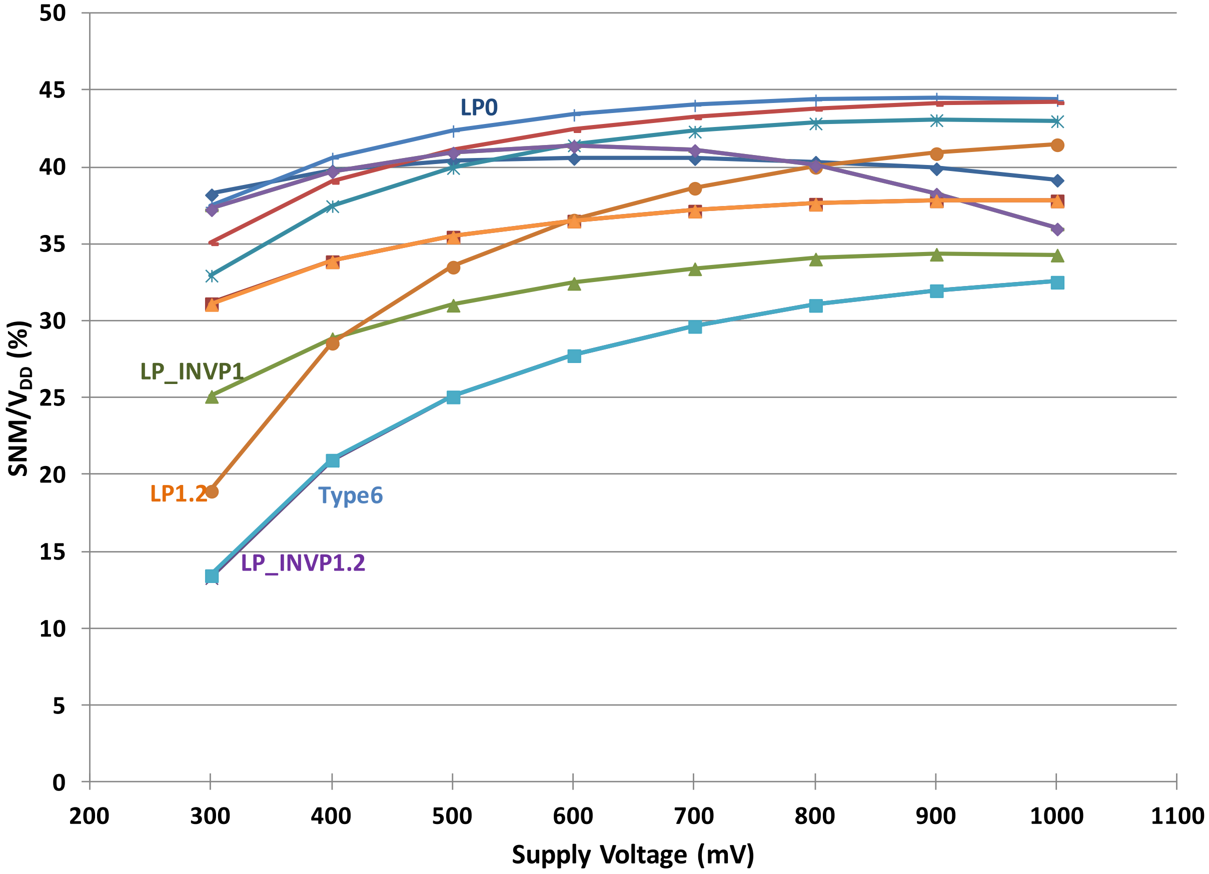


Figure 6.6. SNM versus V_{DD} value for 8T SRAM cells

The overall best six SRAM cells, previously examined in greater detail in Chapter 4 and Chapter 5, are the 6T SG, 6T LP, 8T SG, 8T LP_SGR, 8T_LP_INV1, and 8T LP_INV1.2 cells. These are also the best-performing SRAM cells for near-threshold operation in terms of average

EDP for a 32×1024 FinFET SRAM array, SNM, and RSNM. The average EDPs for these cells when decreasing the supply voltage are shown in Figure 6.7. Overall, the optimal supply voltage chosen for these six cells is 0.6 V; this is the optimal V_{DD} for the 6T SG, 8T SG, and 8T LP_INV1 cells. The optimal supply voltages for the 8T LP_SGR and 8T LP_INV1.2 cells are 0.55 V and 0.5 V, respectively, while the optimal V_{DD} for the 6T LP cell is 0.8 V. A near-threshold operating voltage of 0.6 V was chosen since the average EDP only increases by 16% for the 8T LP_SGR cell (from 11 ps×fJ to 13 ps×fJ) and by 35% for the 8T LP_INV1.2 cell (from 7 ps×fJ to 9 ps×fJ). The 6T LP cell has a lower average EDP than the 6T SG cell only for supply voltages of 1, 0.9, and 0.8 V; the 6T SG cell is more optimal for near-threshold operation.

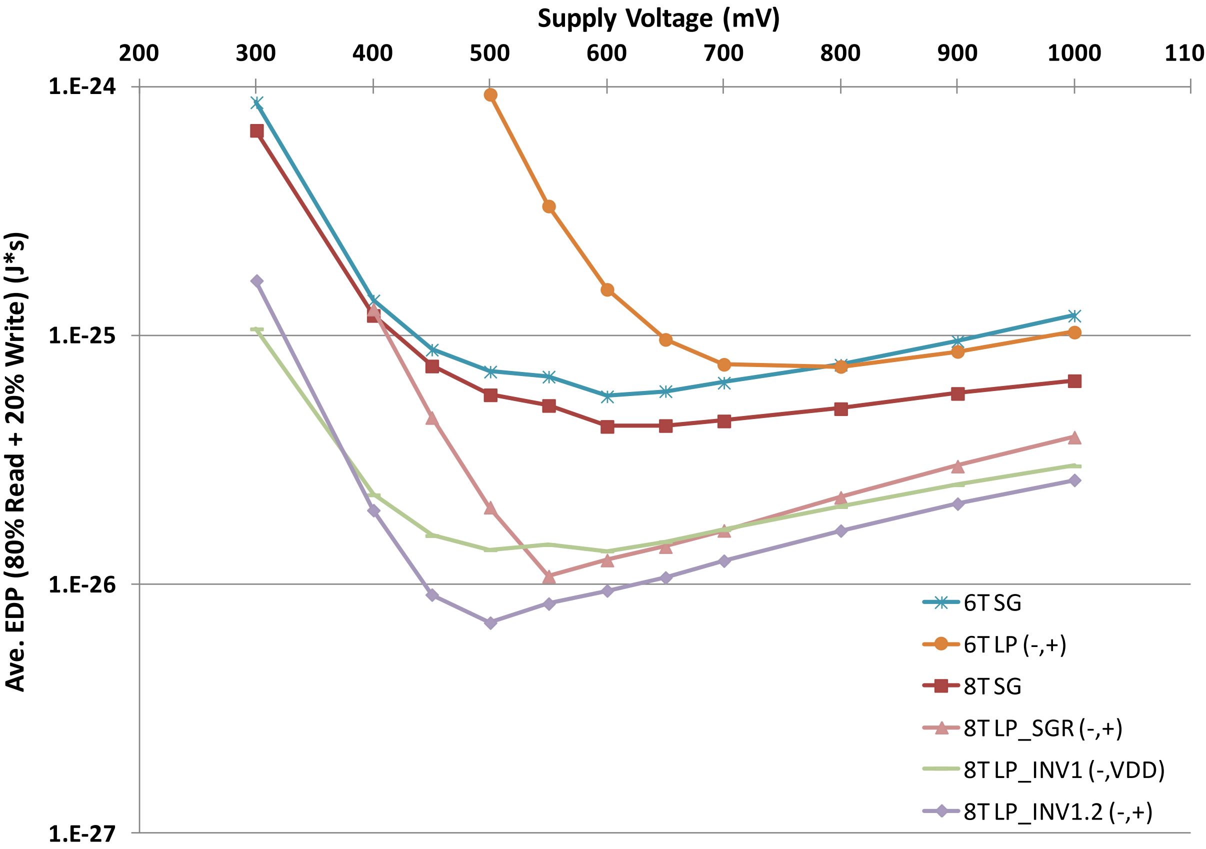


Figure 6.7. Average EDP versus V_{DD} value for the best six SRAM cells

The following two tables provide greater detail about the FinFET SRAM cells' performance at V_{DD} = 0.6 V; Table 6.1 displays the delays, energies, and average EDPs of the cells and Table 6.2 presents the leakage currents, SNMs, and RSNMs of the cells. At this supply voltage value, 8T LP_SGR experiences the greatest reduction of 67% in average EDP. In fact, all but the 6T LP cell sees average EDP reductions between 35% and 67%; the 6T LP cell's average EDP increases by 49%. The other performance metrics presented in Table 6.1 and Table 6.2 will be discussed throughout this subsection.

Table 6.1. Near-threshold operation results of FinFET SRAM delay, energy, and EDP

	Scheme *Pass; Inv-N; Inv-P; Read*	V_{DD}	**32 bits × 1024 words Array**											
			Delay (ps)				**Energy (fJ)**						**Ave* EDP (ps×fJ)**	
			Rd	*Comp*	Wr	*Comp*	Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*	Val	*Comp*
6T	SG	1 V	32	--	5	--	2.4	--	9.3	--	3.8	--	120	--
	t; t; t	0.6 V	51	*1.59*	5	*1*	0.9	*0.38*	2.1	*0.23*	1.1	*0.29*	57	*0.48*
	LP	1 V	84	--	6	--	0.5	--	3.9	--	1.2	--	103	--
	-; -; +	0.6 V	463	*5.51*	20	*3.33*	0.2	*0.40*	0.8	*0.21*	0.3	*0.25*	153	*1.49*
8T	SG	1 V	32	--	3	--	0.9	--	6.9	--	2.1	--	66	--
	3 x t; t; t; t	0.6 V	50	*1.56*	4	*1.33*	0.6	*0.67*	2.0	*0.29*	0.9	*0.43*	43	*0.65*
	LP_SGR	1 V	32	--	8	--	0.3	--	5.0	--	1.2	--	39	--
	-; -; +; t	0.6 V	50	*1.56*	32	*4.00*	0.1	*0.33*	0.9	*0.18*	0.3	*0.25*	13	*0.33*
	LP_INV1	1 V	32	--	4	--	0.4	--	3.2	--	0.9	--	30	--
	t; -; vdd; t	0.6 V	50	*1.56*	10	*2.50*	0.1	*0.25*	0.8	*0.25*	0.3	*0.33*	14	*0.47*
	LP_INV1.2	1 V	32	--	5	--	0.3	--	3.0	--	0.8	--	26	--
	t; -; +; t	0.6 V	50	*1.56*	18	*3.60*	0.1	*0.33*	0.6	*0.20*	0.2	*0.25*	9	*0.35*

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

Table 6.2. Near-threshold operation results of FinFET SRAM leakage and noise margins

	Scheme *Pass; Inv-N; Inv-P; Read*	V_{DD}	**Leakage / Cell (nA)**		**SNM (mV)**				**RSNM (mV)**			
			Val	*Comp*	Val	*Comp*	% V_{DD}	*Comp*	Val	*Comp*	% V_{DD}	*Comp*
6T	SG	1 V	0.592	--	360	--	36.0	--	124	--	12.4	--
	t; t; t	0.6 V	0.499	*0.84*	248	*0.69*	41.3	*1.15*	100	*0.81*	16.7	*1.34*
	LP	1 V	0.021	--	442	--	44.2	--	222	--	22.2	--
	-; -; +	0.6 V	0.018	*0.86*	254	*0.57*	42.3	*0.96*	131	*0.59*	21.8	*0.98*
8T	SG	1 V	0.592	--	360	--	36.0	--				
	3 x t; t; t; t	0.6 V	0.498	*0.84*	248	*0.69*	41.3	*1.15*				
	LP_SGR	1 V	0.021	--	442	--	44.2	--				
	-; -; +; t	0.6 V	0.018	*0.86*	254	*0.57*	42.3	*0.96*				
	LP_INV1	1 V	0.096	--	443	--	44.3	--				
	t; -; vdd; t	0.6 V	0.081	*0.84*	260	*0.59*	43.3	*0.98*				
	LP_INV1.2	1 V	0.021	--	442	--	44.2	--				
	t; -; +; t	0.6 V	0.018	*0.86*	254	*0.57*	42.3	*0.96*				

The reason that the 6T LP cell's average EDP rapidly increases for supply voltages below 0.8 V is that its read delay greatly increases for these values of V_{DD}. At the 0.6 V supply voltage, the 6T LP cell's read delay increases by 5.5X. Figure 6.8 shows the read and write delays for the six cells for decreasing supply voltages. The "8T Read" curve shows the read delay for all four 8T cells since each cell uses the same SG configuration for the Read transistors T7 and T8. The maximum delays for the other five cells do not see large increases until the supply voltage drops below 0.6 V; at 0.6 V these cells only experience read delay increases of 56-59% due to their use of SG-configured FinFETs along their read paths for maximum I_{on} and speed. Write delay is still not the dominant delay at this voltage level, however, compared to the read delay, greater increases in the write delay occurred for the 8T LP_SGR, 8T LP_INV1, and 8T LP_INV1.2 cells. As mentioned previously, word-line and/or read-line boosting can be used to decrease delay and enable lower operating supply voltages for near-threshold operation; the performance of these speed enhancements will be presented in the next subsection.

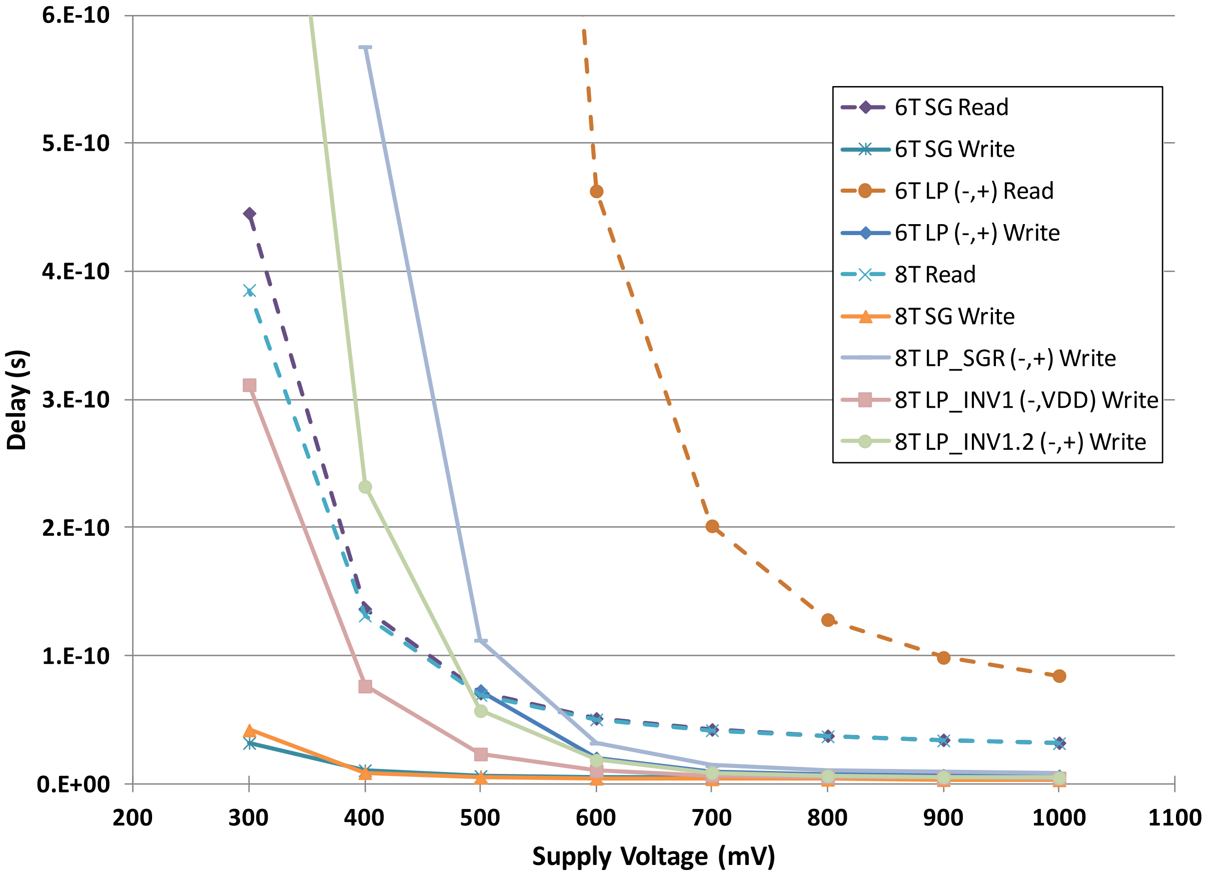


Figure 6.8. Read and write delays versus V_{DD} value for the best six SRAM cells

Figure 6.9 shows the leakage current of the six cells for decreasing supply voltages. As shown previously by the effect of supply voltage variations on leakage current presented Table 4.4, leakage current linearly decreases as the supply voltage decreases. The percentage decrease per reduction in supply voltage is similar for the cells; at 0.6 V V_{DD}, a 24% reduction is observed for the 6T LP, 8T LP_SGR, and 8T LP_INV1.2 cells while a 26% reduction is observed for the 6T SG, 8T SG, and 8T LP_INV1 cells. Due to their higher nominal leakage currents, the 6T SG and 8T SG cells see the largest absolute reductions in leakage current due to decreased supply voltages.

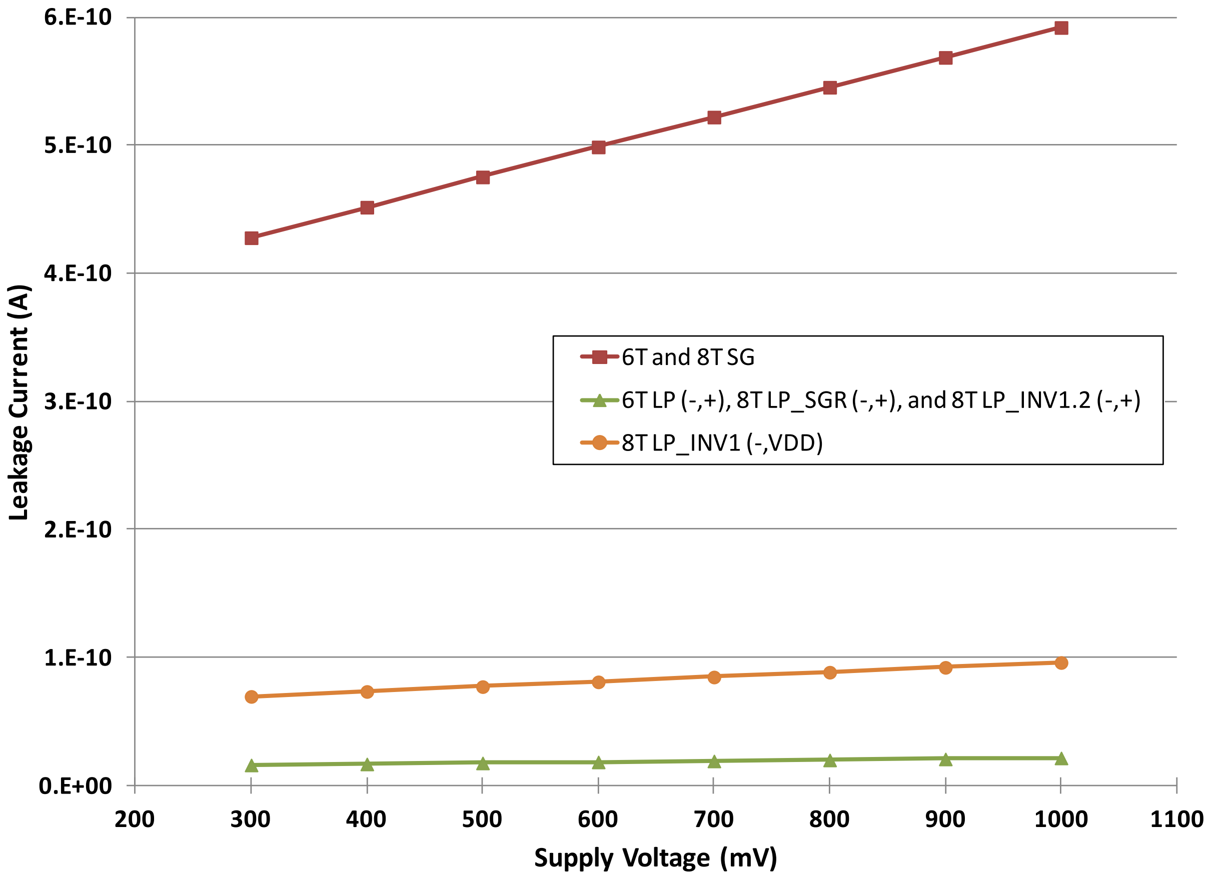


Figure 6.9. Leakage current versus V_{DD} value for the best six SRAM cells

Table 6.3 presents the read and write leakage current totals of the six cells for decreasing supply voltages. The comparisons between full-V_{DD} operation at 1 V and near-threshold operation at 0.6 V of the cross-coupled inverter leakage presented by Table 6.2 and Figure 6.9 and the read and write leakage currents are very similar. The read and write leakage currents at 0.6 V for each cell is between 14% and 16% less than the nominal read and write leakage currents at a supply voltage of 1 V, except for the read [0 1] current of the 8T cells which is slightly less reduced at 8-15% less than the nominal read [0 1] current. Overall, this read and write leakage behavior is expected, since all read and write leakage currents were similarly affected by the supply voltage variations presented by Table 4.5.

Table 6.3. Near-threshold operation results of FinFET SRAM read and write leakage

	Scheme *Pass; Inv-N;* *Inv-P; Read*	**V_{DD}**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read 1 [0 1] 1**			
			Val	*Comp*	Val	*Comp*	Val	*Comp*		
6T	SG	1 V	0.592	--	1.595	--	1.093	--		
	t; t; t	0.6 V	0.499	*0.84*	1.344	*0.84*	0.921	*0.84*		
	LP	1 V	0.021	--	0.057	--	0.039	--		
	-; -; +	0.6 V	0.018	*0.86*	0.048	*0.84*	0.033	*0.85*		
	Scheme *Pass; Inv-N;* *Inv-P; Read*	**V_{DD}**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read [0 1]**		**Read [1 0]**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*	Val	*Comp*
8T	SG	1 V	0.592	--	3.600	--	2.597	--	2.288	--
	3 x t; t; t; t	0.6 V	0.499	*0.84*	3.034	*0.84*	2.189	*0.84*	1.945	*0.85*
	LP_SGR	1 V	0.021	--	0.057	--	0.540	--	0.231	--
	-; -; +; t	0.6 V	0.018	*0.86*	0.048	*0.84*	0.456	*0.84*	0.212	*0.92*
	LP_INV1	1 V	0.096	--	1.099	--	1.099	--	0.789	--
	t; -; vdd; t	0.6 V	0.081	*0.84*	0.926	*0.84*	0.926	*0.84*	0.682	*0.86*
	LP_INV1.2	1 V	0.021	--	1.024	--	1.024	--	0.715	--
	t; -; +; t	0.6 V	0.018	*0.86*	0.863	*0.84*	0.863	*0.84*	0.619	*0.87*

Figure 6.10 shows the SNMs and RSNMs, as a percentage of V_{DD}, of the six cells for decreasing supply voltages. As expected, as V_{DD} decreases, the noise margins decrease; at the 0.6 V V_{DD}, the SNMs decrease by 31-43% and the RSNMs decrease by 19% for 6T SG and 41% for 6T LP. It is interesting that as V_{DD} decreases, the 6T SG and 8T SG cells see at first an increase, then a decrease in the values relative to V_{DD} (SNM/V_{DD} and RSNM/V_{DD}) of their noise margins—all other cells experience a reduction in SNM/V_{DD} and RSNM/V_{DD} as the supply voltage decreases. In particular, at the 0.6 V supply voltage, the 6T and 8T SG cells see a 15% increase in SNM/V_{DD} while the other cells witness a 2-4% reduction. Additionally, at 0.6 V, the 6T SG cell has a 34% increase in RSNM/V_{DD} while the 6T LP cell has a 2% decrease. These differences in SNM/V_{DD} and RSNM/V_{DD} are due to the differences in Inv-N and Inv-P I_{on} and I_{off} at the near-threshold operating voltage of 0.6 V.

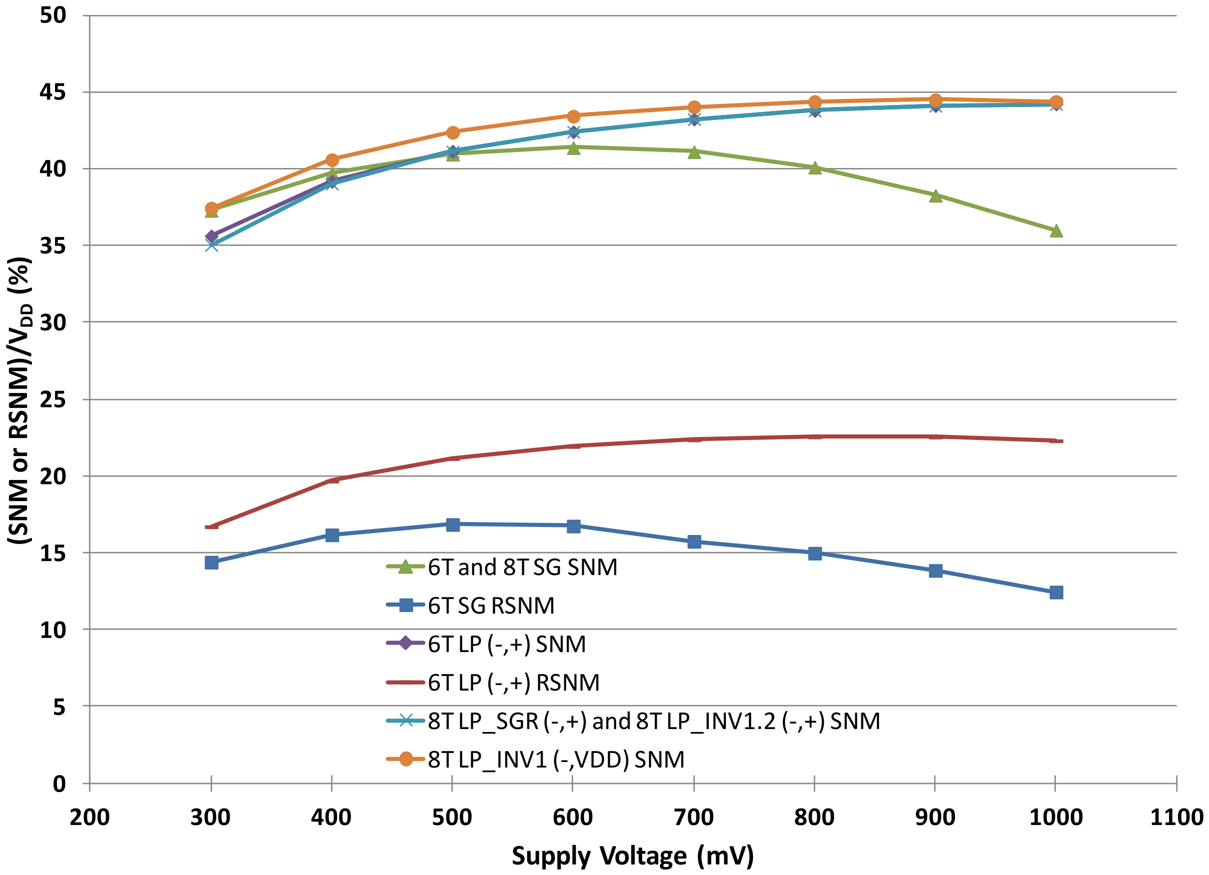


Figure 6.10. Noise margins versus V_{DD} value for the best six SRAM cells

Overall, of the six examined cells, the 8T LP_INV1.2 cell has the lowest leakage, lowest delay, lowest average EDP for a 32×1024 array, and the second-largest SNM (the 8T LP_INV1 cell has a 6 mV larger SNM) when operating at the near-threshold operation voltage of 0.6 V. If noise margins are the highest design priority, a higher SNM (a 2.4% increase from 254 mV to 260 mV) plus one less BG bias voltage can obtained by using the 8T LP_INV1 cell which has the third lowest average EDP for a 32×1024 array (a 55% increase from 9 ps×fJ to 14 ps×fJ).

6.2 SRAM Speed Enhancements

It is apparent from Figure 6.1 and Figure 6.8 that delay is the largest barrier for near-threshold operation. There are a couple techniques that can improve read and write speed, especially for operation below the nominal supply voltage. For 6T SRAM cells, the word-line can be boosted; for this research, instead of the word-line signal transitioning between 0 V and V_{DD}, the word-line is boosted to transition between 0 V and V_{DD} + 0.2 V. Boosting the word-line improves 6T SRAM read and write speeds. The word-line/write-line can also be boosted for 8T SRAM cells to improve the write speed, but the read speed can be improved by boosting the read-line; for this research, instead of the read-line signal transitioning between 0 V and V_{DD}, the read-line is boosted to transition between 0 V and V_{DD} + 0.2 V. The delay, energy, and EDP performance results for the six cells for near-threshold operation at 0.6 V are summarized by Table 6.4 and the RSNM results for the 6T SG and 6T LP cells are displayed by Table 6.5. The following two subsections will further analyze the speed improvements provided by boosting for the 6T and 8T SRAM cells, respectively.

Table 6.4. Near-threshold operation results with boosting of delay, energy, and EDP

	Scheme *Pass; Inv-N; Inv-P; Read*	**Boosting**$^+$	**32 bits × 1024 words Array**											
			Delay (ps)				**Energy (fJ)**						**Ave* EDP (ps×fJ)**	
			Rd	*Comp*	Wr	*Comp*	Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.	51	--	5	--	0.9	--	2.1	--	1.1	--	57	--
		WL	36	*0.71*	3	*0.60*	1.1	*1.22*	1.5	*0.71*	1.2	*1.09*	41	*0.72*
	LP -; -; +	Nom.	463	--	20	--	0.2	--	0.8	--	0.3	--	153	--
		WL	280	*0.60*	11	*0.55*	0.2	*1*	0.6	*0.75*	0.3	*1*	85	*0.56*
8T	SG 3 x t; t; t; t	Nom.	50	--	4	--	0.6	--	2.0	--	0.9	--	43	--
		WL	50	*1*	3	*0.75*	0.6	*1*	1.5	*0.75*	0.7	*0.78*	37	*0.86*
		RdLn	35	*0.70*	4	*1*	0.5	*0.83*	1.9	*0.95*	0.8	*0.89*	27	*0.63*
		WL+RdLn	35	*0.70*	3	*0.75*	0.5	*0.83*	1.3	*0.65*	0.6	*0.67*	22	*0.51*
	LP_SGR -; -; +; t	Nom.	50	--	32	--	0.1	--	0.9	--	0.3	--	13	--
		WL	50	*1*	22	*0.69*	0.1	*1*	0.7	*0.78*	0.2	*0.67*	10	*0.77*
		RdLn	35	*0.70*	32	*1*	0.1	*1*	0.9	*1*	0.3	*1*	10	*0.77*
		WL+RdLn	35	*0.70*	22	*0.69*	0.1	*1*	0.7	*0.78*	0.2	*0.67*	9	*0.69*
	LP_INV1 t; -; vdd; t	Nom.	50	--	10	--	0.1	--	0.8	--	0.3	--	14	--
		WL	50	*1*	5	*0.50*	0.1	*1*	0.4	*0.50*	0.2	*0.67*	10	*0.71*
		RdLn	35	*0.70*	10	*1*	0.2	*2.00*	0.7	*0.88*	0.3	*1*	10	*0.71*
		WL+RdLn	35	*0.70*	5	*0.50*	0.2	*2.00*	0.4	*0.50*	0.2	*0.67*	8	*0.57*
	LP_INV1.2 t; -; +; t	Nom.	50	--	18	--	0.1	--	0.6	--	0.2	--	9	--
		WL	50	*1*	6	*0.33*	0.1	*1*	0.3	*0.50*	0.1	*0.50*	6	*0.67*
		RdLn	35	*0.70*	18	*1*	0.1	*1*	0.6	*1*	0.2	*1*	8	*0.89*
		WL+RdLn	35	*0.70*	6	*0.33*	0.1	*1*	0.3	*0.50*	0.2	*1*	5	*0.56*

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.
$^+$ Boosting Key: **Nom.** = Nominal (no boosting); **WL** = Word-line boosting;
RdLn = Read-line boosting; **WL+RdLn** = Word-line and Read-line boosting

Table 6.5. Near-threshold operation results with boosting of FinFET 6T SRAM RSNMs

	Scheme *Pass; Inv-N; Inv-P; Read*	**Boosting**$^+$	**RSNM (mV)**		
			Val	*Comp*	% V_{DD}
6T	SG t; t; t	Nom.	100	--	16.7
		WL	11	*0.11*	1.8
	LP -; -; +	Nom.	131	--	21.8
		WL	45	*0.34*	7.5

$^+$ Boosting Key: **Nom.** = Nominal (no boosting);
WL = Word-line boosting; **RdLn** = Read-line boosting;
WL+RdLn = Word-line and Read-line boosting

6.2.1 Word-line Boosting for 6T SRAMs

Using word-line boosting for the 6T LP and 6T SG cells leads to 29-45% reductions in read and write delays. The maximum delay for both cells is still the read delay, but this is reduced by 29% for the 6T SG cell and 40% for the 6T LP cell. Figure 6.11 shows the read and write delays of these two cells for decreasing values of supply voltage. These reductions in delay also lead to reductions in EDP. The 6T SG cell sees a 28% reduction in average EDP and the 6T LP cell sees a 44% reduction in average EDP. Figure 6.12 shows the EDPs of these two cells for decreasing values of supply voltage. However, the tradeoff for using word-line boosting, other than increased power required by the word-line drivers and boosting circuitry, is a reduced RSNM. When the word-line is enabled for a read operation, there is always a risk in accidentally overwriting the cell; the RSNM measures this risk. When the word-line is boosted, the Pass transistors are more strongly turned on and this risk increases. Table 6.5 shows that the RSNM is greatly reduced for the 6T cells; the 6T SG cell witnesses an 89% reduction and the 6T LP cell witnesses a 66% reduction at V_{DD} = 0.6 V. The resultant RSNMs are both below 10% of V_{DD}. Figure 6.13 shows the RSNM/V_{DD} of these two cells for decreasing values of supply voltage. Even at full-V_{DD} operation, the 6T SG cell has a 63% reduction and the 6T LP cell has a 37% reduction in RSNM at the 1 V supply voltage. As the supply voltage decreases, each cell's RSNM falls precipitously. The resultant values of RSNM/V_{DD} are problematic for near-threshold operation as noise becomes much more likely to corrupt these 6T cells during a read operation.

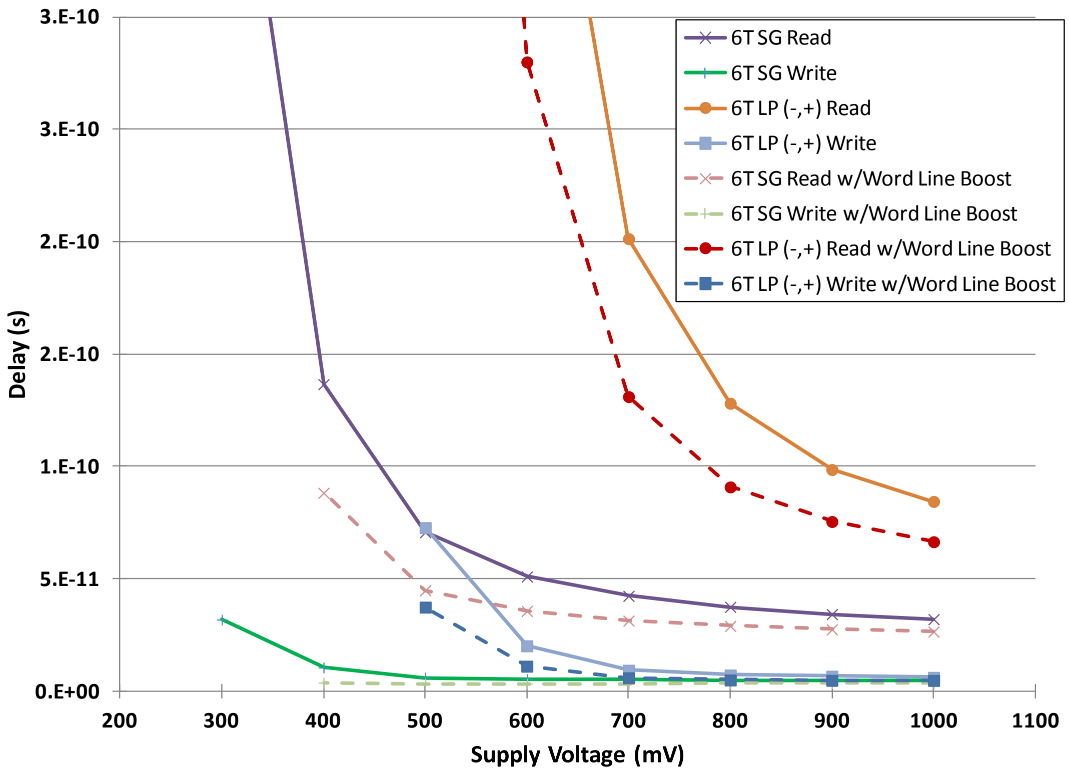


Figure 6.11. Delays versus V_{DD} value for the 6T SRAM cells using word-line boosting

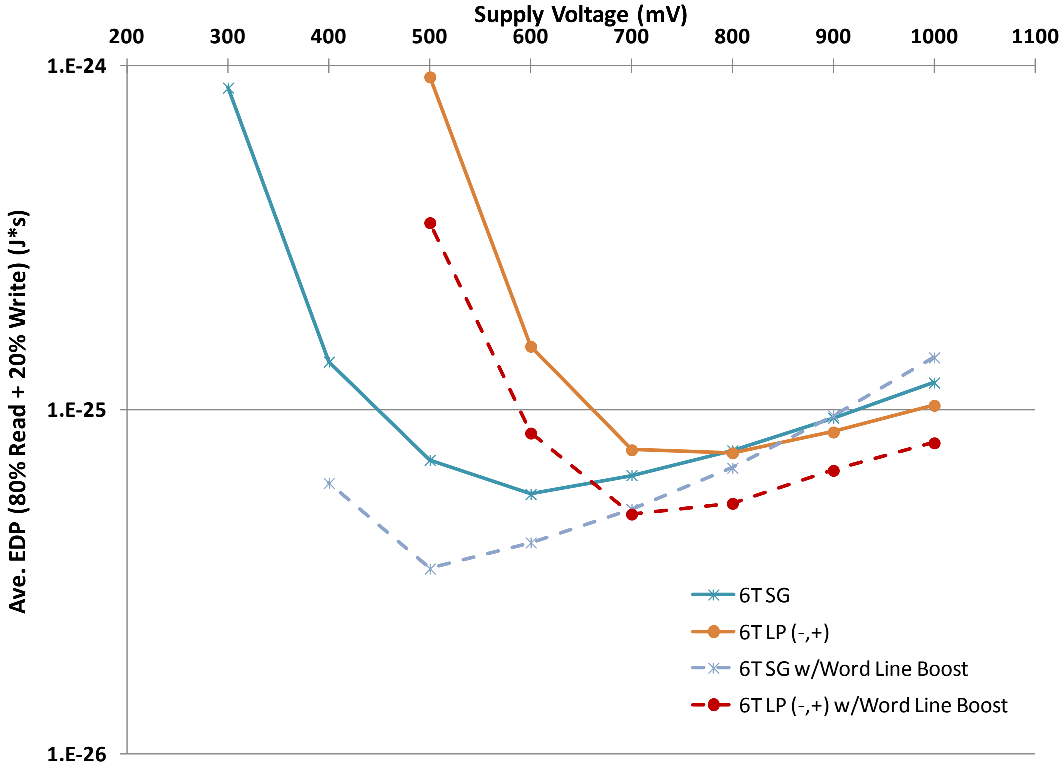


Figure 6.12. EDP versus V_{DD} value for the 6T SRAM cells using word-line boosting

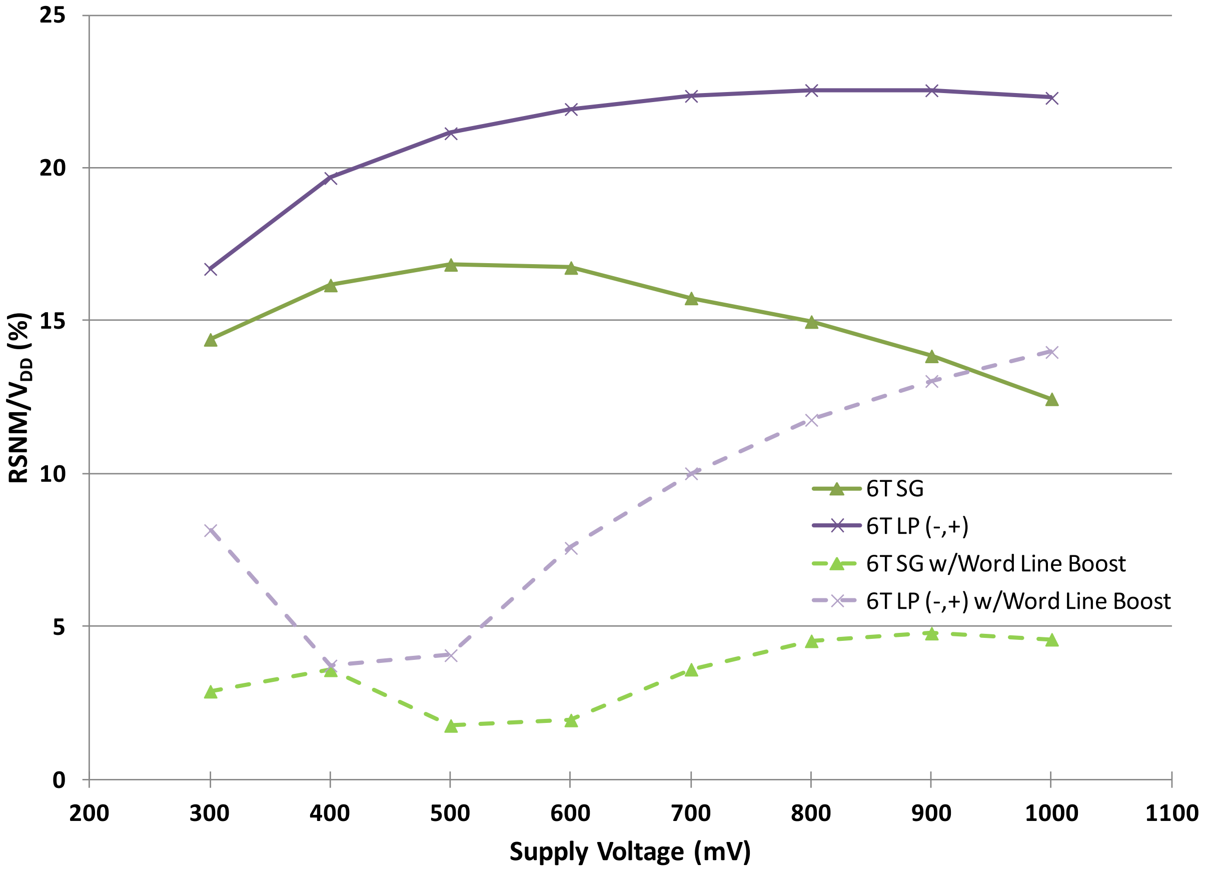


Figure 6.13. RSNM versus V_{DD} value for the 6T SRAM cells using word-line boosting

6.2.2 Word-line/Write-line and Read-line Boosting for 8T SRAMs

Using word-line/write-line and/or read-line boosting for the four 8T SRAM cells leads to 25-67% reductions in read and write delays. The maximum delay for all cells is still the read delay, but this is reduced by 30% using read-line boosting. Figure 6.14 shows the read and write delays of the 8T cells for decreasing values of supply voltage. These reductions in delay also lead to reductions in EDP. The 8T SG cells undergo 14-49% reductions in average EDP. Figure 6.15 shows the EDPs of the 8T cells for decreasing values of supply voltage.

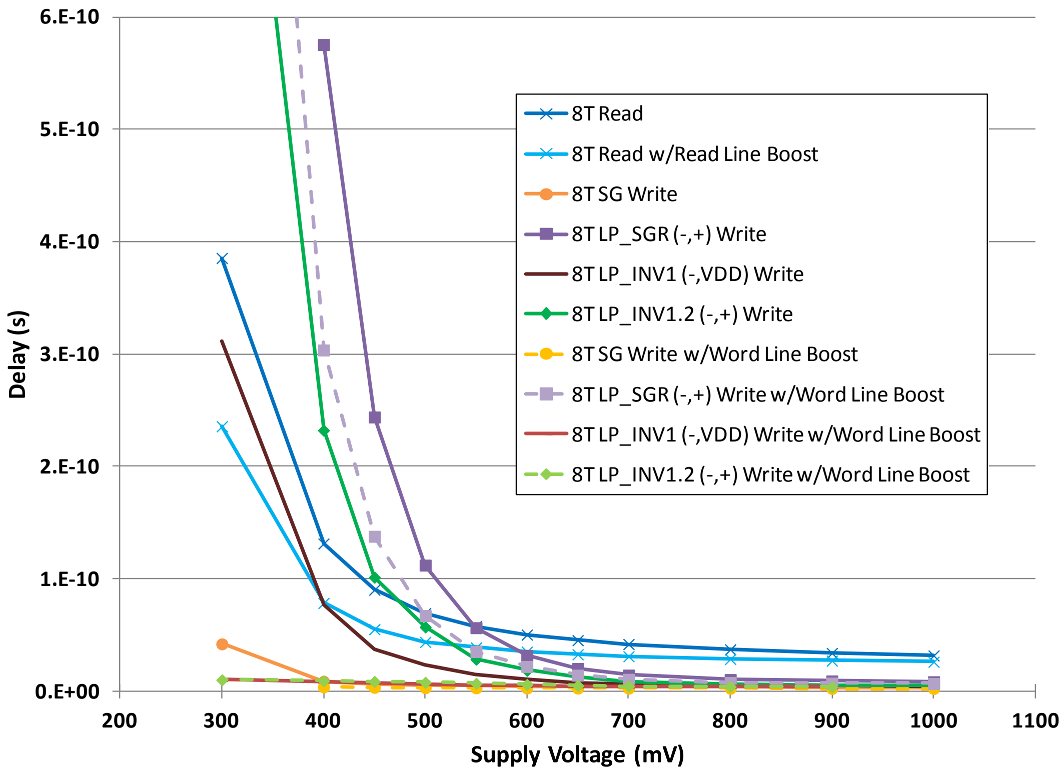


Figure 6.14. 8T delays versus V_{DD} value using word-line and read-line boosting

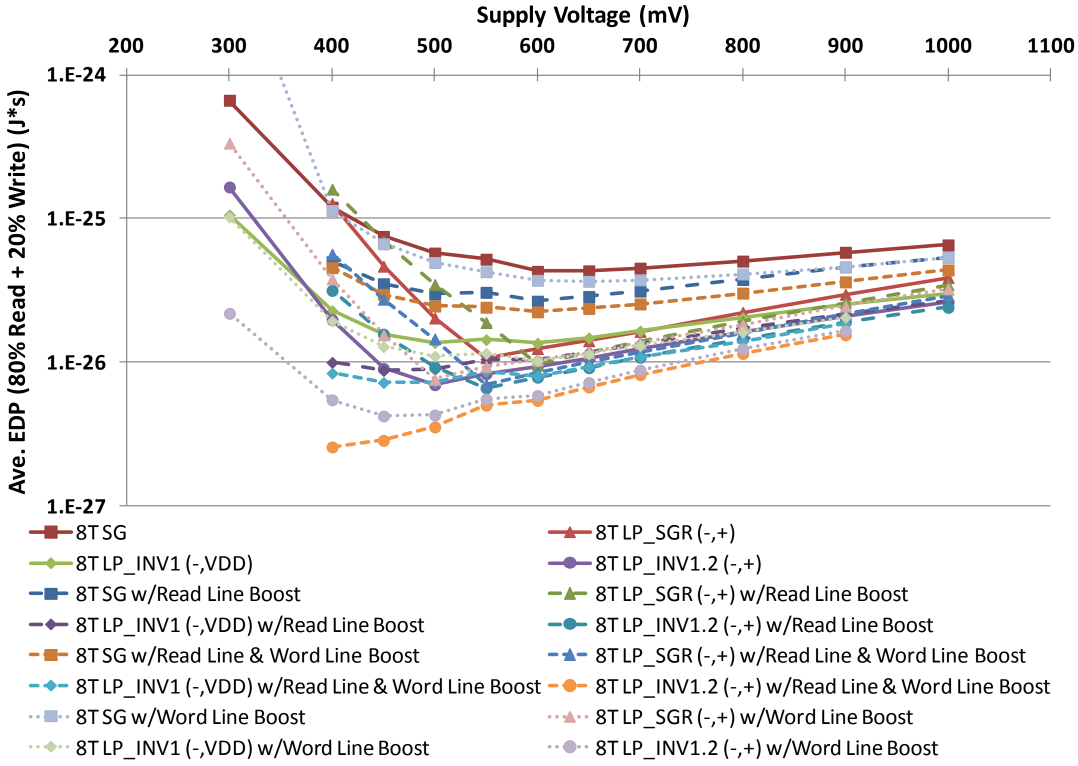


Figure 6.15. 8T EDP versus V_{DD} value using word-line and read-line boosting

While read-line boosting reduces the maximum delay of the 8T cells, word-line and read-line boosting contribute to reductions in average EDP. Only the 8T SG cell experiences more of a reduction in average EDP (37%) due to read-line boosting than word-line boosting (14%). The 8T LP_SGR and 8T LP_INV1 cells see equal reductions of 23% and 29%, respectively, and the 8T LP_INV1.2 cell only experiences an 11% reduction due to read-line boosting and sees a 33% reduction due to word-line boosting; this is because these three cells already have low read energies at 1 fJ and the word-line boosting reduces their higher write energies which in turn plays a part in the reduction of their average energy and EDP. Boosting both the read-line and the word-line reduces the 8T cells' average EDP by 31-49%. These are substantial savings, but are tempered with more power required for read-line and word-line drivers and boosting circuitry. However, unlike the two 6T SRAM cells, word-line and read-line boosting do not decrease noise margins for the 8T cells, therefore better delays, energies, and EDP can be obtained without reducing noise margin performance.

6.3 Process, Voltage, and Temperature Variations

In this section, the performance of 32×1024 arrays for the 6T SG SRAM cell and the 8T SG, LP_SGR, LP_INV1, and LP_INV1.2 SRAM cells are examined and under the effects of process/parameter, voltage, and temperature (PVT) variations for near-threshold operation at V_{DD} = 0.6 V. The 6T LP cell is no longer considered since it has a lower average EDP than the 6T SG cell only for supply voltages of 1, 0.9, and 0.8 V; the 6T SG cell is more optimal for near-threshold operation scheme since it has a lower average EDP at V_{DD} = 0.6 V. Word-line boosting improved the 6T LP cell's speed and average EDP, however, using word-line boosting for both 6T cells causes their RSNMs to become too small.

6.3.1 Process/Parameter Variations

The near-threshold performance of the five cells are examined and compared using the same quasi-Monte Carlo (QMC) analysis used in Section 4.1 [20]. Table 6.6 shows the near-threshold performance results of parameter variation simulations on the 32×1024 arrays. Table 6.7 summarizes the simulated effects of parameter variations on read and write leakage current totals during near-threshold operation.

Table 6.6. Near-threshold FinFET SRAM parameter variation simulation results

	Scheme *Pass; Inv-N; Inv-P; Read*	**Leakage / Cell (nA)**	**32 bits × 1024 words Array**						**SNM (mV)**		**RSNM (mV)**	
			Delay (ps)		**Energy (fJ)**			**Ave* EDP (ps×fJ)**				
			Rd	***Wr***	***Rd***	***Wr***	***Ave****		**Val**	**% V_{DD}**	**Val**	**% V_{DD}**
6T	SG	0.47	58	6	1.2	2.7	1.5	115	243	40.5	95	15.8
	t; t; t	*0.20*	*9*	*9*	*5.6*	*11.1*	*5.1*	*88*	*9*	*1.5*	*18*	*3.0*
8T	SG	0.48	54	6	0.6	3.1	1.1	109	244	40.7		
	3 x t; t; t; t	*0.30*	*13*	*18*	*1.2*	*20.2*	*4.2*	*1196*	*15*	*2.5*		
	LP_SGR	0.03	54	34	0.1	0.9	0.3	14	246	41.0		
	-; -; +; t;	*0.12*	*6*	*6*	*0.1*	*0.1*	*0.1*	*7*	*8*	*1.3*		
	LP_INV1	0.09	54	10	0.2	0.8	0.3	16	251	41.8		
	t; -; vdd; t	*0.04*	*6*	*2*	*0.1*	*0.1*	*0.1*	*8*	*6*	*1.0*		
	LP_INV1.2	0.03	54	19	0.1	0.6	0.2	12	246	41.0		
	t; -; +; t	*0.15*	*5*	*4*	*0.4*	*0.2*	*0.3*	*24*	*8*	*1.3*		

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

Table 6.7. Near-threshold parameter variation results of read/write leakage totals (nA)

	Scheme *Pass; Inv-N; Inv-P; Read*	***Write 0 [0 1] 1***	***Write 1 [0 1] 0***	***Read 1 [0 1] 1***	
6T	SG	0.47	1.25	0.88	
	t; t; t	*0.21*	*0.34*	*0.30*	
	Scheme *Pass; Inv-N; Inv-P; Read*	***Write 0 [0 1] 1***	***Write 1 [0 1] 0***	***Read [0 1]***	***Read [1 0]***
8T	SG	0.47	2.91	2.09	1.84
	3 x t; t; t; t	*0.20*	*0.45*	*0.35*	*0.29*
	LP_SGR	0.02	0.05	0.43	0.19
	-; -; +; t;	*0.01*	*0.02*	*0.19*	*0.06*
	LP_INV1	0.08	0.88	0.87	0.63
	t; -; vdd; t	*0.04*	*0.29*	*0.24*	*0.16*
	LP_INV1.2	0.02	0.81	0.81	0.57
	t; -; +; t	*0.01*	*0.28*	*0.24*	*0.15*

The mean read delays of the 6T SG cell is 14% longer than its nominal read delay while the mean read delays of the 8T cells are 8% longer than their nominal read delays. The standard deviations of the read delays are all below 13% of the respective means. The 8T SG cell has the largest standard deviation of 13 ps, or 24% of its delay of 54 ps. The 6T SG cell's standard deviation is 16% of its mean. As a percentage of their means, the other 8T cells have lower standard deviations: 11% for the LP_SGR and LP_INV1 cells and 9% for the LP_INV1.2 cell. The read delays are similar to the nominal values, but with higher variation than what was observed for V_{DD} = 1 V. The mean read energy of the 6T SG cell is 33% greater than its nominal value of 0.9 fJ. The mean read energies of the 8T cells are identical to their nominal values except for 8T LP_INV1, which only increases by 0.1 fJ. However, the standard deviations of the read energy are high for all five cells. The standard deviation of the 6T SG cell's read energy is 4.7X greater than its mean. The cells standard deviations are no more than 2X greater than their respective means, except for the 8T LP_INV1.2 cell which is 4X greater. However, this is due to its low mean read energy of 0.1 fJ, since small fluctuations can lead to large percent changes.

The mean write delays of the five cells are within 2 ps of their respective nominal values. The standard deviations of the write delays of 8T cells which use LP-configured FinFETs are no more than 21% of their respective means. The standard deviation of the write delay of the 6T SG scheme is 1.5X greater than its mean and the standard deviation of the 8T SG scheme is 3.0X greater than its mean. These variations are larger than was seen for operation at the 1 V supply voltage, however, similar to the parameter variations for V_{DD} = 1 V, the standard deviations of the SG cells are significantly larger than that for the LP cells. Similar versions of these results are also seen for the write energies, however, the mean write energies of 6T SG and 8T SG cells are 29% and 55% larger than their nominal values, respectively. The mean write energies of the 8T LP_SGR, LP_INV1, and LP_INV1.2 cells are identical to their nominal values. The standard deviations of the write energies of 8T cells which use LP-configured FinFETs are no more than 33% of their respective means. The standard deviation of the write delay of the 6T SG scheme is 4.1X greater than its mean and the standard deviation of the 8T SG scheme is 6.5X greater than its mean; this is due to the large standard deviations of the write delays for these schemes.

The means of the cross-coupled inverter leakage currents per cell for all five cells are all very similar to their respective nominal values (within 33%). Somewhat surprisingly, the standard deviations of the cross-coupled leakage current are lower for the 6T and 8T SG cells than what was observed for V_{DD} = 1 V; these cells have standard deviations of no more than 63% of their mean leakage values. This is similar for the LP_INV1 cell, whose standard deviation is

44% of its mean, but the LP_SGR and LP_INV1.2 cells have standard deviations of 4.0X and 5.0X of their means, respectively. The large variations in leakage current per cell for these cells are a byproduct of their low mean leakage values of 0.03 nA; small variations can cause large percentage changes in the leakage current. Additionally, all five varied cells have average read and write leakage totals less than their nominal values and see similar variations as for the cross-coupled inverter leakage current.

The mean SNMs of the five cells are up to 4% less, for the LP_INV1 cell, than their nominal values. The standard deviations, as a percentage of the mean SNM, are also small for these cells, at no more than 6% for the 8T SG cell. Both of these values are slightly greater than what was seen for V_{DD} = 1 V operation; the mean SNMs were no more than 2% less than their nominal values and the standard deviations were no more than 4.4% of their means. The mean RSNM of the 6T SG cell is 5% less than its nominal value of 100 mV; at V_{DD} = 1 V operation, the RSNM increased. The standard deviation of the 6T SG RSNM is 19% of it mean; greater than the 7.1% seen for a 1 V supply voltage.

The mean average EDP increased for each of the five cells compared to their nominal values; the 6T SG cell's mean average EDP is 2.0X greater, the 8T SG cell's is 2.5X greater, the 8T LP_SGR cell's is 7.7% greater, the 8T LP_INV1 cell's is 14.2% greater, and the 8T LP_INV1.2 cell's is 33% greater. The 8T SG cell has the largest standard deviation of 11.0X its mean and the LP_INV1.2 cell has a standard deviation of 2.0X its mean. The other cells, however, have standard deviations of no more than 77% of their respective means. The cells which use LP configuration and reverse-bias the cross-coupled inverter transistors have significantly less variation in average energy and average EDP than the other cells. Of the five cells examined using parameter variation simulations, the 8T LP_INV1.2 cell has the lowest average EDP for a 32×1024 array and the second-largest SNM. However, the 8T LP_INV1.2 cell also has higher variation in average EDP than the 8T LP_SGR and 8T LP_INV1 cells; these cells have 2 ps×fJ and 4 ps×fJ greater average EDP, respectively, and the 8T LP_INV1 cell has a 5 mV higher SNM.

6.3.2 Supply Voltage Variations

For near-threshold supply voltage variation simulations, the supply voltage was studied for 540, 570, 600, 630, and 660 mV (or -10% to +10% of V_{DD} = 1 V) for the five cells. Table 6.8 shows the simulation results of near-threshold V_{DD} variations on the delays, energies, and average EDP of the 32×1024 arrays. Table 6.9 displays the simulation results of near-threshold supply voltage variations on the leakage current, SNMs, and RSNMs of the SRAM cells. Table 6.10 summarizes the simulation results of near-threshold V_{DD} variations on the read and write leakage current totals.

Table 6.8. Near-threshold V_{DD} variation results of FinFET SRAM delay, energy, and EDP

	Scheme *Pass; Inv-N; Inv-P; Read*	**V_{DD}**	**32 bits × 1024 words Array**											
			Delay (ps)				**Energy (fJ)**						**Ave* EDP (ps×fJ)**	
			Rd	*Comp*	Wr	*Comp*	Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*	Val	*Comp*
6T	SG t; t; t	-10%	61	*1.20*	5	*1*	0.8	*0.89*	1.7	*0.81*	0.9	*0.82*	57	*1*
		-5%	55	*1.08*	5	*1*	0.8	*0.89*	1.9	*0.90*	1.0	*0.91*	57	*1*
		Nom.	51	--	5	--	0.9	--	2.1	--	1.1	--	57	--
		+5%	48	*0.94*	5	*1*	0.9	*1*	2.5	*1.19*	1.2	*1.09*	58	*1.02*
		+10%	45	*0.88*	5	*1*	1.0	*1.11*	2.8	*1.33*	1.3	*1.18*	61	*1.07*
8T	SG 3 x t; t; t; t	-10%	60	*1.20*	5	*1.25*	0.6	*1*	1.6	*0.80*	0.8	*0.89*	46	*1.07*
		-5%	54	*1.08*	4	*1*	0.6	*1*	1.8	*0.90*	0.8	*0.89*	44	*1.02*
		Nom.	50	--	4	--	0.6	--	2.0	--	0.9	--	43	--
		+5%	47	*0.94*	4	*1*	0.6	*1*	2.3	*1.15*	0.9	*1*	43	*1*
		+10%	44	*0.88*	4	*1*	0.6	*1*	2.6	*1.30*	1.0	*1.11*	44	*1.02*
	LP_SGR -; -; +; t	-10%	59	*1.18*	64	*2.00*	0.1	*1*	0.6	*0.67*	0.2	*0.67*	13	*1*
		-5%	54	*1.08*	44	*1.38*	0.1	*1*	0.8	*0.89*	0.2	*0.67*	12	*0.92*
		Nom.	50	--	32	--	0.1	--	0.9	--	0.3	--	13	--
		+5%	47	*0.94*	24	*0.75*	0.1	*1*	1.1	*1.22*	0.3	*1*	13	*1*
		+10%	44	*0.88*	18	*0.56*	0.1	*1*	1.2	*1.33*	0.3	*1*	14	*1.08*
	LP_INV1 t; -; vdd; t	-10%	60	*1.20*	16	*1.60*	0.1	*1*	0.6	*0.75*	0.2	*0.67*	13	*0.93*
		-5%	54	*1.08*	12	*1.20*	0.1	*1*	0.7	*0.88*	0.2	*0.67*	13	*0.93*
		Nom.	50	--	10	--	0.1	--	0.8	--	0.3	--	14	--
		+5%	47	*0.94*	8	*0.80*	0.2	*2.00*	0.9	*1.13*	0.3	*1*	14	*1*
		+10%	44	*0.88*	7	*0.70*	0.2	*2.00*	1.0	*1.25*	0.3	*1*	15	*1.07*
	LP_INV1.2 t; -; +; t	-10%	59	*1.18*	35	*1.94*	0.1	*1*	0.4	*0.67*	0.1	*0.50*	8	*0.89*
		-5%	54	*1.08*	25	*1.39*	0.1	*1*	0.5	*0.83*	0.2	*1*	9	*1*
		Nom.	50	--	18	--	0.1	--	0.6	--	0.2	--	9	--
		+5%	47	*0.94*	14	*0.78*	0.1	*1*	0.7	*1.17*	0.2	*1*	10	*1.11*
		+10%	44	*0.88*	11	*0.61*	0.1	*1*	0.8	*1.33*	0.2	*1*	11	*1.22*

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

Table 6.9. Near-threshold V_{DD} variation results of leakage and noise margins

	Scheme *Pass; Inv-N; Inv-P; Read*	V_{DD}	**Leakage / Cell (nA)**		**SNM (mV)**				**RSNM (mV)**			
			Val	*Comp*	Val	*Comp*	% V_{DD}	*Comp*	Val	*Comp*	% V_{DD}	*Comp*
6T	SG t; t; t	-10%	0.485	*0.97*	222	*0.90*	41.1	*0.99*	91	*0.91*	16.9	*1.01*
		-5%	0.492	*0.99*	235	*0.95*	41.2	*1*	96	*0.96*	16.8	*1.01*
		Nom.	0.499	--	248	--	41.3	--	100	--	16.7	--
		+5%	0.506	*1.01*	260	*1.05*	41.3	*1*	104	*1.04*	16.5	*0.99*
		+10%	0.513	*1.03*	272	*1.10*	41.2	*1*	107	*1.07*	16.2	*0.97*
8T	SG 3 x t; t; t; t	-10%	0.484	*0.97*	222	*0.90*	41.1	*0.99*				
		-5%	0.491	*0.99*	235	*0.95*	41.2	*1*				
		Nom.	0.498	--	248	--	41.3	--				
		+5%	0.506	*1.02*	260	*1.05*	41.3	*1*				
		+10%	0.513	*1.03*	272	*1.10*	41.2	*1*				
	LP_SGR -; -; +; t	-10%	0.018	*1*	225	*0.89*	41.7	*0.98*				
		-5%	0.018	*1*	239	*0.94*	41.9	*0.99*				
		Nom.	0.018	--	254	--	42.3	--				
		+5%	0.018	*1*	268	*1.06*	42.5	*1*				
		+10%	0.019	*1.06*	283	*1.11*	42.9	*1.01*				
	LP_INV1 t; -; vdd; t	-10%	0.078	*0.96*	231	*0.89*	42.8	*0.99*				
		-5%	0.080	*0.99*	246	*0.95*	43.2	*1*				
		Nom.	0.081	--	260	--	43.3	--				
		+5%	0.082	*1.01*	274	*1.05*	43.5	*1*				
		+10%	0.083	*1.02*	289	*1.11*	43.8	*1.01*				
	LP_INV1.2 t; -; +; t	-10%	0.018	*1*	225	*0.89*	41.7	*0.98*				
		-5%	0.018	*1*	239	*0.94*	41.9	*0.99*				
		Nom.	0.018	--	254	--	42.3	--				
		+5%	0.018	*1*	268	*1.06*	42.5	*1*				
		+10%	0.019	*1.06*	283	*1.11*	42.9	*1.01*				

Table 6.10. Near-threshold V_{DD} variations on read/write leakage totals (nA)

	Scheme *Pass; Inv-N; Inv-P; Read*	V_{DD}	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read 1 [0 1] 1**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*
6T	SG t; t; t	-10%	0.485	*0.97*	1.306	*0.97*	0.895	*0.97*
		-5%	0.492	*0.99*	1.325	*0.99*	0.908	*0.99*
		Nom.	0.499	--	1.344	--	0.921	--
		+5%	0.506	*1.01*	1.362	*1.01*	0.934	*1.01*
		+10%	0.513	*1.03*	1.381	*1.03*	0.947	*1.03*

	Scheme *Pass; Inv-N; Inv-P; Read*	V_{DD}	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read [0 1]**		**Read [1 0]**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*	Val	*Comp*
8T	SG 3 x t; t; t; t	-10%	0.485	*0.97*	2.948	*0.97*	2.127	*0.97*	1.893	*0.97*
		-5%	0.492	*0.99*	2.991	*0.99*	2.158	*0.99*	1.919	*0.99*
		Nom.	0.499	--	3.034	--	2.189	--	1.945	--
		+5%	0.506	*1.01*	3.076	*1.01*	2.219	*1.01*	1.971	*1.01*
		+10%	0.513	*1.03*	3.119	*1.03*	2.250	*1.03*	1.997	*1.03*
	LP_SGR -; -; +; t	-10%	0.018	*1*	0.047	*0.98*	0.443	*0.97*	0.209	*0.99*
		-5%	0.018	*1*	0.048	*1*	0.449	*0.98*	0.210	*0.99*
		Nom.	0.018	--	0.048	--	0.456	--	0.212	--
		+5%	0.018	*1*	0.049	*1.02*	0.462	*1.01*	0.214	*1.01*
		+10%	0.019	*1.06*	0.050	*1.04*	0.468	*1.03*	0.215	*1.01*
	LP_INV1 t; -; vdd; t	-10%	0.079	*0.98*	0.900	*0.97*	0.900	*0.97*	0.666	*0.98*
		-5%	0.080	*0.99*	0.913	*0.99*	0.913	*0.99*	0.674	*0.99*
		Nom.	0.081	--	0.926	--	0.926	--	0.682	--
		+5%	0.082	*1.01*	0.939	*1.01*	0.939	*1.01*	0.690	*1.01*
		+10%	0.083	*1.02*	0.952	*1.03*	0.952	*1.03*	0.698	*1.02*
	LP_INV1.2 t; -; +; t	-10%	0.018	*1*	0.839	*0.97*	0.839	*0.97*	0.605	*0.98*
		-5%	0.018	*1*	0.851	*0.99*	0.851	*0.99*	0.612	*0.99*
		Nom.	0.018	--	0.863	--	0.863	--	0.619	--
		+5%	0.018	*1*	0.875	*1.01*	0.875	*1.01*	0.627	*1.01*
		+10%	0.019	*1.06*	0.887	*1.03*	0.887	*1.03*	0.634	*1.02*

The read delays of the schemes decrease with increasing V_{DD}. The five cells have similar read paths, via two SG-configured FinFETs, and thus see similar changes in read delay of -12% to +20%. Only the 6T SG cell and 8T LP_INV1 cell experiences changes in read energy, but these changes are no more than 0.1 fJ.

The write delays of the SG cells minimally vary with changes in V_{DD}, but the other cells, which have higher nominal write delays, see larger variations in write delays. The 8T LP_SGR cell has the largest nominal write delay and experiences the largest variation in write delay between -44% and +2X. The cells have more variation in write energy due to larger nominal values, and, as expected, as V_{DD} increases so too does the write energy. All cells see a similar increase in write energy to +25-33% for a +10% change in the supply voltage, but the LP cells experience greater decreases in write energy to as much as -33% for a -10% change in V_{DD}. The LP cells see greater variation in write energy due to supply voltage variations at near-threshold operation.

As for V_{DD} = 1 V, leakage current is mildly affected by supply voltage variations. All five cells see a 0-4% reduction in leakage current due to a 10% reduction in V_{DD} and the cells see a 2-6% increase in leakage current due to a 10% increase in V_{DD}. This is also very similar for the read and write leakage. As a whole, the schemes leak within 6% of their nominal values for all read and write leakage scenarios.

The SNMs of the cells vary within 11% of their nominal values; this is greater variation than what was observed at V_{DD} =1 V for the SG cells. For the 6T SG cell, its RSNM varies between -9% to +7%; this is more than the 1 mV variation it experiences at V_{DD} = 1 V.

Overall the SG cells have the least variation in average EDP with at most a 7% increase. The 8T LP_SGR and 8T LP_INV1 cells have average EDP variations of ±8% while the 8T LP_INV1.2 cell has the greatest variation between -11% and 22%. The 8T LP_ INV1.2 cell has smallest average EDP with variation between -11% and 22% and the second-highest SNM at each value of V_{DD}. The 8T LP_SGR and 8T LP_INV1 cells have slightly higher average EDP, but with variations only of ±8%, and the LP_INV1 cell has the highest average EDP.

6.3.3 Bias Voltage Variations

For near-threshold bias voltage variation simulations, the reverse-bias value was studied for 18-22 mV (or -10% to +10% of the 0.2 V used throughout this research) for the three cells (8T LP_SGR, 8T LP_INV1, and 8T LP_INV1.2) which use LP-mode FinFETs. The n-type FinFET reverse-biases are -0.22 V to -0.18 V and the p-type FinFET reverse-biases are 1.18 V to 1.22 V. Table 6.11 shows the simulation results of near-threshold bias voltage variations on the delays, energies, and average EDP of the 32×1024 arrays. Table 6.12 displays the simulation results of near-threshold bias voltage variations on the leakage current, SNMs, and RSNMs of the SRAM cells. Table 6.13 summarizes the simulation results of near-threshold bias voltage variations on the read and write leakage current totals.

Table 6.11. Near-threshold bias voltage variation results of delay, energy, and EDP

	Scheme *Pass; Inv-N; Inv-P; Read*	**Bias**	**32 bits × 1024 words Array**											
			Delay (ps)				**Energy (fJ)**						**Ave* EDP (ps×fJ)**	
			Rd	*Comp*	Wr	*Comp*	Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.	51	--	5	--	0.9	--	2.1	--	1.1	--	57	--
8T	SG 3 x t; t; t; t	Nom.	50	--	4	--	0.6	--	2.0	--	0.9	--	43	--
	LP_SGR -; -; +; t	-10%	50	*1*	29	*0.91*	0.1	*1*	0.9	*1*	0.3	*1*	13	*1*
		-5%	50	*1*	30	*0.94*	0.1	*1*	0.9	*1*	0.3	*1*	13	*1*
		Nom.	50	--	32	--	0.1	--	0.9	--	0.3	--	13	--
		+5%	50	*1*	33	*1.03*	0.1	*1*	0.9	*1*	0.2	*0.67*	12	*0.92*
		+10%	50	*1*	34	*1.06*	0.1	*1*	0.9	*1*	0.2	*0.67*	12	*0.92*
	LP_INV1 t; -; vdd; t	-10%	50	*1*	10	*1*	0.2	*2.00*	0.8	*1*	0.3	*1*	14	*1*
		-5%	50	*1*	10	*1*	0.1	*1*	0.8	*1*	0.3	*1*	14	*1*
		Nom.	50	--	10	--	0.1	--	0.8	--	0.3	--	14	--
		+5%	50	*1*	10	*1*	0.1	*1*	0.8	*1*	0.3	*1*	13	*0.93*
		+10%	50	*1*	10	*1*	0.1	*1*	0.8	*1*	0.3	*1*	13	*0.93*
	LP_INV1.2 t; -; +; t	-10%	50	*1*	17	*0.94*	0.1	*1*	0.6	*1*	0.2	*1*	10	*1.11*
		-5%	50	*1*	18	*1*	0.1	*1*	0.6	*1*	0.2	*1*	10	*1.11*
		Nom.	50	--	18	--	0.1	--	0.6	--	0.2	--	9	--
		+5%	50	*1*	19	*1.06*	0.1	*1*	0.6	*1*	0.2	*1*	9	*1*
		+10%	50	*1*	20	*1.11*	0.1	*1*	0.6	*1*	0.2	*1*	9	*1*

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

Table 6.12. Near-threshold bias voltage variation results of leakage and noise margins

	Scheme *Pass; Inv-N; Inv-P; Read*	**Bias**	**Leakage / Cell (nA)**		**SNM (mV)**			**RSNM (mV)**	
			Val	*Comp*	Val	*Comp*	% V_{DD}	Val	% V_{DD}
6T	SG t; t; t	Nom.	0.499	--	248	--	41.3	100	*16.7*
8T	SG 3 x t; t; t; t	Nom.	0.498	--	248	--	41.3		
	LP_SGR -; -; +; t	-10%	0.024	*1.33*	254	*1*	42.3		
		-5%	0.021	*1.17*	254	*1*	42.3		
		Nom.	0.018	--	254	--	42.3		
		+5%	0.016	*0.89*	254	*1*	42.3		
		+10%	0.014	*0.78*	254	*1*	42.3		
	LP_INV1 t; -; vdd; t	-10%	0.085	*1.05*	260	*1*	43.3		
		-5%	0.083	*1.02*	260	*1*	43.3		
		Nom.	0.081	--	260	--	43.3		
		+5%	0.079	*0.98*	260	*1*	43.3		
		+10%	0.077	*0.95*	259	*1*	43.2		
	LP_INV1.2 t; -; +; t	-10%	0.024	*1.33*	254	*1*	42.3		
		-5%	0.021	*1.17*	254	*1*	42.3		
		Nom.	0.018	--	254	--	42.3		
		+5%	0.016	*0.89*	254	*1*	42.3		
		+10%	0.014	*0.78*	254	*1*	42.3		

Table 6.13. Near-threshold bias voltage variation results of read/write leakage totals (nA)

	Scheme *Pass; Inv-N; Inv-P; Read*	**Bias**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read 1 [0 1] 1**			
			Val	*Comp*	Val	*Comp*	Val	*Comp*		
6T	SG t; t; t	Nom.	0.499	--	1.344	--	0.921	--		
	Scheme *Pass; Inv-N; Inv-P; Read*	**Bias**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read [0 1]**		**Read [1 0]**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*	Val	*Comp*
8T	SG 3 x t; t; t; t	Nom.	0.499	--	3.034	--	2.189	--	1.945	--
	LP_SGR -; -; +; t	-10%	0.024	*1.33*	0.063	*1.31*	0.466	*1.02*	0.222	*1.05*
		-5%	0.021	*1.17*	0.055	*1.15*	0.460	*1.01*	0.217	*1.02*
		Nom.	0.018	--	0.048	--	0.456	--	0.212	--
		+5%	0.016	*0.89*	0.042	*0.88*	0.452	*0.99*	0.208	*0.98*
		+10%	0.014	*0.78*	0.037	*0.77*	0.448	*0.98*	0.204	*0.96*
	LP_INV1 t; -; vdd; t	-10%	0.085	*1.05*	0.930	*1*	0.930	*1*	0.687	*1.01*
		-5%	0.083	*1.02*	0.928	*1*	0.928	*1*	0.684	*1*
		Nom.	0.081	--	0.926	--	0.926	--	0.682	--
		+5%	0.079	*0.98*	0.924	*1*	0.924	*1*	0.680	*1*
		+10%	0.077	*0.95*	0.922	*1*	0.922	*1*	0.679	*1*
	LP_INV1.2 t; -; +; t	-10%	0.024	*1.33*	0.869	*1.01*	0.869	*1.01*	0.625	*1.01*
		-5%	0.021	*1.17*	0.866	*1*	0.866	*1*	0.622	*1*
		Nom.	0.018	--	0.863	--	0.863	--	0.619	--
		+5%	0.016	*0.89*	0.861	*1*	0.861	*1*	0.617	*1*
		+10%	0.014	*0.78*	0.859	*1*	0.859	*1*	0.615	*0.99*

The read delays of the 8T schemes are not affected by changes in bias voltages; this is expected since they use identical SG read transistor configurations. Since the affect on read delay is negligible, the variations in leakage current cause minimal variations in read energy; no more than a 0.1 fJ difference is observed. Only the write delays of the 8T LP_SGR and 8T LP_INV1.2 cells are affected by bias voltage variations; these differ between -9% and +11%. These variations are more than the negligible changes seen at the 1 V supply voltage. The write energy, however, is not influenced by bias voltage variations.

Leakage current is most affected by bias voltage variations. The 8T LP_SGR and 8T LP_INV1.2 cells, which use a -0.2 V BG bias for Inv-N FinFETs and a V_{DD} + 0.2 V, or 0.8 V, BG bias for Inv-P FinFETs, see variations between -22% to +33%; this is slightly less than the ±33% variation at V_{DD} = 1 V. The 8T LP_INV1 cell, which uses a nominal -0.2 V BG bias for the Inv-N FinFETs and ties the back gates of the Inv-P FinFETs to V_{DD} = 0.6 V, has leakage current variations within 5% of its nominal value due to less reliance on bias voltages; this is identical to bias voltage variation results at V_{DD} = 1 V. For read and write leakage, similar results are seen for the write 0 [0 1] 1 leakage. The 8T LP_SGR cell also has similar variation in its write 1 [0 1] 0 leakage; as mentioned before, this is due to variations in the leakage current of its LP-configured Pass transistors. Also similar to the bias voltage results for a 1 V supply voltage is that the 8T LP_INV1 and 8T LP_INV1.2 cells have similar and negligible variation in their write 1 [0 1] 0 leakage, but have larger write 1 [0 1] 0 leakage values, due to their SG-configured Pass transistors whose leakage does not vary due to bias voltage variations. Ideally, there should be no variation in read leakage for the 8T cells because SG-mode Read FinFETs are used; however, the read leakage of the 8T schemes sees only minimal variation due to slight variations of simultaneously-occurring write leakage caused by leftover values on the WBit and WNBit lines.

As for V_{DD} = 1 V, the variations in leakage current due to bias voltage variations do not cause variations in the SNMs or RSNMs for the cells. The ratios of leakage current of the Inv-N and Inv-P are minimally affected since both n-type and p-type FinFET BG biases either fall closer to ground and V_{DD} or rise farther from ground and V_{DD}.

Overall, variations in BG bias voltage negligibly affect the average EDP and noise margins of the three 8T cells which use reverse-biased LP-mode FinFETs. At most, the 8T

LP_SGR, LP_INV1, and LP_INV1.2 cells see a 1 ps×fJ variation in average EDP. As for a supply voltage of 1 V, the 8T LP_ INV1 cell sees the smallest variations in leakage current since only its Inv-N transistors use a bias voltage (the back gates of its Inv-P FinFETs are tied to V_{DD}). The 8T LP_SGR and LP_INV1.2 cells see between a -22% and +33% variation in leakage current, but this does not affect their noise margins and only negligibly affects their active energies and average EDP.

6.3.4 Temperature Variations

For near-threshold temperature variation simulations, the ambient temperature was studied for 0, 27, 50, 75, and 100°C for the five cells. Table 6.14 shows the simulation results of near-threshold temperature variations on the delays, energies, and average EDP of the 32×1024 arrays. Table 6.15 displays the simulation results of near-threshold temperature variations on the leakage current, SNMs, and RSNMs of the SRAM cells. Table 6.16 summarizes the simulation results of near-threshold temperature variations on the read and write leakage current totals.

Table 6.14. Near-threshold temperature variation results of delay, energy, and EDP

	Scheme *Pass; Inv-N;* *Inv-P; Read*	**Temp. (°C)**	**32 bits × 1024 words Array**											
			Delay (ps)				**Energy (fJ)**						**Ave* EDP (ps×fJ)**	
			Rd	*Comp*	Wr	*Comp*	Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*	Val	*Comp*
6T	SG t; t; t	0	49	*0.96*	5	*1*	0.5	*0.56*	1.8	*0.86*	0.7	*0.64*	35	*0.61*
		27	51	--	5	--	0.9	--	2.1	--	1.1	--	57	--
		50	54	*1.06*	6	*1.20*	1.9	*2.11*	3.2	*1.52*	2.1	*1.91*	114	*2.00*
		75	59	*1.16*	6	*1.20*	4.6	*5.11*	5.9	*2.81*	4.9	*4.45*	287	*5.04*
		100	64	*1.25*	7	*1.40*	10.5	*11.67*	11.8	*5.62*	10.8	*9.82*	686	*12.04*
8T	SG 3 x t; t; t; t	0	48	*0.96*	4	*1*	0.2	*0.33*	1.7	*0.85*	0.5	*0.56*	24	*0.56*
		27	50	--	4	--	0.6	--	2.0	--	0.9	--	43	--
		50	54	*1.08*	5	*1.25*	1.5	*2.50*	3.0	*1.50*	1.8	*2.00*	99	*2.30*
		75	61	*1.22*	5	*1.25*	4.4	*7.33*	5.8	*2.90*	4.7	*5.22*	283	*6.58*
		100	66	*1.32*	5	*1.25*	10.4	*17.33*	11.9	*5.95*	10.7	*11.89*	701	*16.30*
	LP_SGR -; -; +; t	0	48	*0.96*	33	*1.03*	0.1	*1*	0.9	*1*	0.2	*1*	11	*0.85*
		27	50	--	32	--	0.1	--	0.9	--	0.3	--	13	--
		50	54	*1.08*	31	*0.97*	0.1	*1*	1.0	*1.11*	0.3	*1.00*	17	*1.31*
		75	61	*1.22*	30	*0.94*	0.3	*3.00*	1.2	*1.33*	0.5	*1.67*	30	*2.31*
		100	66	*1.32*	30	*0.94*	0.8	*8.00*	1.8	*2.00*	1.0	*3.33*	67	*5.15*
	LP_INV1 t; -; vdd; t	0	48	*0.96*	10	*1*	0.1	*1*	0.7	*0.88*	0.2	*1*	10	*0.71*
		27	50	--	10	--	0.1	--	0.8	--	0.3	--	14	--
		50	54	*1.08*	10	*1*	0.3	*3.00*	0.9	*1.13*	0.5	*1.67*	24	*1.71*
		75	61	*1.22*	10	*1*	0.9	*9.00*	1.5	*1.88*	1.0	*3.33*	64	*4.57*
		100	66	*1.32*	10	*1*	2.3	*23.00*	3.0	*3.75*	2.5	*8.33*	163	*11.64*
	LP_INV1.2 t; -; +; t	0	48	*0.96*	20	*1.11*	0.1	*1*	0.6	*1*	0.2	*1*	8	*0.89*
		27	50	--	18	--	0.1	--	0.6	--	0.2	--	9	--
		50	54	*1.08*	17	*0.94*	0.1	*1*	0.7	*1.17*	0.2	*1*	13	*1.44*
		75	61	*1.22*	16	*0.89*	0.3	*3.00*	0.8	*1.33*	0.4	*2.00*	26	*2.89*
		100	66	*1.32*	15	*0.83*	0.8	*8.00*	1.4	*2.33*	0.9	*4.50*	62	*6.89*

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

Table 6.15. Near-threshold temperature variation results of leakage and noise margins

	Scheme *Pass; Inv-N; Inv-P; Read*	**Temp. (°C)**	**Leakage / Cell (nA)**		**SNM (mV)**			**RSNM (mV)**		
			Val	*Comp*	Val	*Comp*	% V_{DD}	Val	*Comp*	% V_{DD}
6T	SG t; t; t	0	0.119	*0.24*	253	*1.02*	42.2	107	*1.07*	17.8
		27	0.499	--	248	--	41.3	100	--	16.7
		50	1.395	*2.80*	243	*0.98*	40.5	95	*0.95*	15.8
		75	3.588	*7.19*	238	*0.96*	39.7	91	*0.91*	15.2
		100	8.026	*16.08*	233	*0.94*	38.8	86	*0.86*	14.3
8T	SG 3 x t; t; t; t	0	0.119	*0.24*	253	*1.02*	42.2			
		27	0.498	--	248	--	41.3			
		50	1.395	*2.80*	243	*0.98*	40.5			
		75	3.585	*7.20*	238	*0.96*	39.7			
		100	7.987	*16.04*	233	*0.94*	38.8			
	LP_SGR -; -; +; t	0	0.003	*0.17*	258	*1.02*	43.0			
		27	0.018	--	254	--	42.3			
		50	0.064	*3.56*	250	*0.98*	41.7			
		75	0.210	*11.67*	246	*0.97*	41.0			
		100	0.590	*32.78*	242	*0.95*	40.3			
	LP_INV1 t; -; vdd; t	0	0.017	*0.21*	264	*1.02*	44.0			
		27	0.081	--	260	--	43.3			
		50	0.246	*3.04*	256	*0.98*	42.7			
		75	0.707	*8.73*	252	*0.97*	42.0			
		100	1.752	*21.63*	248	*0.95*	41.3			
	LP_INV1.2 t; -; +; t	0	0.003	*0.17*	258	*1.02*	43.0			
		27	0.018	--	254	--	42.3			
		50	0.064	*3.56*	250	*0.98*	41.7			
		75	0.210	*11.67*	246	*0.97*	41.0			
		100	0.591	*32.83*	242	*0.95*	40.3			

Table 6.16. Near-threshold temperature variation results of read/write leakage totals (nA)

	Scheme *Pass; Inv-N; Inv-P; Read*	**Temp. (°C)**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read 1 [0 1] 1**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*
6T	SG t; t; t	0	0.119	*0.24*	0.324	*0.24*	0.222	*0.24*
		27	0.499	--	1.344	--	0.921	--
		50	1.395	*2.80*	3.736	*2.78*	2.566	*2.79*
		75	3.583	*7.18*	9.516	*7.08*	6.550	*7.11*
		100	7.977	*15.99*	21.010	*15.63*	14.494	*15.74*

	Scheme *Pass; Inv-N; Inv-P; Read*	**Temp. (°C)**	**Write 0 [0 1] 1**		**Write 1 [0 1] 0**		**Read [0 1]**		**Read [1 0]**	
			Val	*Comp*	Val	*Comp*	Val	*Comp*	Val	*Comp*
8T	SG 3 x t; t; t; t	0	0.119	*0.24*	0.733	*0.24*	0.528	*0.24*	0.468	*0.24*
		27	0.499	--	3.034	--	2.189	--	1.945	--
		50	1.395	*2.80*	8.416	*2.77*	6.076	*2.78*	5.413	*2.78*
		75	3.583	*7.18*	21.380	*7.05*	15.448	*7.06*	13.801	*7.10*
		100	7.978	*15.99*	47.070	*15.51*	34.038	*15.55*	30.477	*15.67*
	LP_SGR -; -; +; t	0	0.003	*0.17*	0.009	*0.19*	0.108	*0.24*	0.048	*0.23*
		27	0.018	--	0.048	--	0.456	--	0.212	--
		50	0.064	*3.56*	0.169	*3.52*	1.287	*2.82*	0.623	*2.94*
		75	0.209	*11.61*	0.550	*11.46*	3.346	*7.34*	1.699	*8.01*
		100	0.589	*32.72*	1.539	*32.06*	7.578	*16.62*	4.017	*18.95*
	LP_INV1 t; -; vdd; t	0	0.018	*0.22*	0.222	*0.24*	0.222	*0.24*	0.162	*0.24*
		27	0.081	--	0.926	--	0.926	--	0.682	--
		50	0.246	*3.04*	2.587	*2.79*	2.587	*2.79*	1.924	*2.82*
		75	0.707	*8.73*	6.640	*7.17*	6.640	*7.17*	4.993	*7.32*
		100	1.751	*21.62*	14.784	*15.97*	14.781	*15.96*	11.221	*16.45*
	LP_INV1.2 t; -; +; t	0	0.003	*0.17*	0.208	*0.24*	0.208	*0.24*	0.148	*0.24*
		27	0.018	--	0.863	--	0.863	--	0.619	--
		50	0.064	*3.56*	2.404	*2.79*	2.404	*2.79*	1.741	*2.81*
		75	0.210	*11.67*	6.143	*7.12*	6.142	*7.12*	4.495	*7.26*
		100	0.590	*32.78*	13.624	*15.79*	13.621	*15.78*	10.060	*16.25*

The read delays of the schemes increase with increasing temperature; the 6T SG cell's read delay increases up to +25% and the 8T cells increase up to +32%. This is because of the similar read paths for these cells. The increase in read delay is less than what was observed for the cells at V_{DD} = 1 V; the 6T SG scheme experienced a +41% increase and the 8T cells had a

+59% increase for an ambient temperature of 100°C. The cells experience larger increases in read energy at the 0.6 V supply voltage than at V_{DD} = 1 V; the 6T SG scheme encounters a greater increase of 11.67X (versus 7.29X for 1 V V_{DD}), the 8T SG scheme has an increase of 17.33X (versus 17.11X), the 8T LP_SGR and LP_INV1.2 cells have an increase of 8.00X (versus 4.67X), and the 8T LP_INV1 cell's read energy increases by 23.00X (versus 9.00X) at 100°C.

The write delays for each scheme only slightly vary with temperature increase; these delays are already minimal compared to the read delay and vary by at most 2 ps for the 8T LP_INV1.2 scheme. The cells also see greater increases in write energy at near-threshold operation; the LP cells see an increase in write energy at 100°C of at most 3.75X (versus 24-56% for V_{DD} = 1 V) and the SG cells see an increase of at most 5.95X (versus at most 3.12X).

Leakage current is most affected by temperature variations. At 100°C an increase of approximately 16X is observed for the 6T and 8T SG schemes, an increase of 21.63X occurs for the 8T LP_INV1 cell, and increases of approximately 33X are witnessed for the 8T LP_SGR and LP_INV1.2 cells. These increases are only slightly more than what was observed at the nominal 1 V supply voltage. For read and write leakage, similar results are seen for the write 0 [0 1] 1 leakage. For write 1 [0 1] 0 leakage, a roughly 16X increase at 100°C is seen for schemes with SG-configured pass transistors while a roughly 32X increase is observed for the 8T LP_SGR cell. The 6T SG cell's read 1 [0 1] 1 leakage increases comparably to its write leakage increases. For the 8T schemes, the read leakage, [0 1] and [1 0] cases, increases 15-19X at 100°C due to the SG-configured read transistors used by each scheme. This behavior is also very similar to what was observed for V_{DD} = 1 V.

Additionally, as for a 1 V supply voltage, at the near-threshold operating voltage of 0.6 V the SNMs of the five examined cells all vary within 6% of their nominal 27°C values. Temperature causes greater variation for the 6T SG cell's RSNM; the cell's RSNM varies by 14% of its nominal value, however, this is less than the 22% variation experienced at V_{DD} = 1 V.

Overall, the 6T LP scheme and the 8T LP_SGR and LP_ INV1.2 cells have the least variation in average EDP; a 5-7X increase at 100°C is witnessed for these schemes. However, this is greater than the 3-4X increase seen at V_{DD} = 1 V. The 6T SG scheme has an increase of 12.0X (versus 6.89X at 1 V V_{DD}), the 8T LP_INV1 cell has an increase of 11.6X (versus 7.07X), and the 8T SG scheme has an increase of 11.9X (versus 12.80X) in average EDP at 100°C. The 8T LP_ INV1.2 cell has smallest average EDP with second-lowest variation, to the 8T LP_SGR cell, and the second-highest SNM, to the 8T LP_INV1 cell.

6.3.5 Summary of PVT Variations

Overall, of the five examined cells, the 8T LP_INV1.2 cell has the lowest average EDP under PVT variations. The 8T LP_SGR cell boasts the lowest variation in average EDP due to PVT variations with only a minor increase in EDP. The 8T LP_INV1 cell also has low variation in average EDP due to parameter and voltage variations, but experiences greater variations for changes in temperature; however, this cell does have the highest SNM of the five studied cells. In particular, the 8T LP_INV1.2 cell has the lowest average EDP for a 32×1024 array and the second-largest SNM due to parameter variations, however, the 8T LP_INV1.2 cell also has higher variation in average EDP than the 8T LP_SGR and 8T LP_INV1 cells; these cells have 2 ps×fJ and 4 ps×fJ greater average EDP, respectively, and the 8T LP_INV1 cell has a 5 mV higher SNM. The 8T LP_ INV1.2 cell has smallest average EDP with variation between -11% and 22% and the second-highest SNM for variations in V_{DD}, however, the 8T LP_SGR and 8T

LP_INV1 cells have slightly higher average EDP, but with variations only of ±8%, and the LP_INV1 cell has the highest average EDP. Overall, variations in BG bias voltage negligibly affect the average EDP and noise margins of the three 8T cells which use reverse-biased LP-mode FinFETs. The 8T LP_ INV1.2 cell has smallest average EDP with second-lowest variation, to the 8T LP_SGR cell, and the second-highest SNM, to the 8T LP_INV1 cell, due to variations in temperature.

6.4 Low-Leakage Modifications: Header/Footer FinFETs

The use of header and footer FinFETs has been examined at near-threshold operation for the five FinFET SRAM cells. In addition to the choice of using header, footer, or both FinFETs, the header and footer FinFETs can be in SG-configuration, LP-configuration with a 0 V n-bias and a 1 V p-bias, or LP-configuration with a -0.2 V n-bias and a 0.8 V (V_{DD} + 0.2 V) p-bias.

Table 6.17 shows the best leakage, delay, and average EDP near-threshold results of using header and footer FinFETs per SRAM cell and Table 6.18 shows the SNM and RSNM near-threshold results of using header and footer FinFETs per SRAM cell. Appendix C includes more results of lower-performing cells. As explained in Chapter 5, if there are a small number of 32×1024 arrays in the memory, then the percent comparisons of the 32×1024 array average EDP are approximate to the percent comparisons of the average EDP for the entire SRAM memory. On the other hand, if there are a large number of 32×1024 arrays in the memory, then the percent comparisons of the leakage EDP per array in sleep-mode are approximate to the percent comparisons of the average EDP of the entire SRAM memory.

Overall, the best performing cells often use only a header transistor per SRAM cell. The leakage EDP per array in sleep-mode is reduced by up to 44-45% for the 6T SG and 8T SG cells (using header and footer transistors), 81% for the 8T LP_INV1 cell (using header transistors), and 17% for the 8T LP_SGR and 8T LP_INV1.2 cells (using header transistors). This is less than what was observed for V_{DD} = 1 V for all but the 8T LP_SGR and 8T LP_INV1.2 cells. Additionally, most cells see a reduction in SNM and RSNM with the inclusion of these transistors; up to a 32% reduction in SNM and up to a 65% in RSNM is observed. The reduction in SNM is larger than the maximum 7% reduction observed at the 1 V supply voltage.

Table 6.17. Leakage, delay, and EDP near-threshold results of headers/footers per cell

	Scheme *Pass; Inv-N; Inv-P; Read*	**Leak. Tran.**	**Leak. Tran. Config.**	**Leak. / Cell (nA)**	**Leakage / Cell in Sleep-Mode (nA)**		**Leakage EDP / Array in Sleep-Mode (ps×fJ)**		**# Arrays for Break-Even EDP$^+$**	**32 bits × 1024 words Array**					
										Delay (ps)				**Ave* EDP (ps×fJ)**	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.499	0.499	--	25.5	--		51	--	5	--	57	--
		Header	t	0.499	0.455	*0.91*	23.3	*0.91*	*1*	51	*1*	6	*1.20*	52	*0.91*
		Hdr + Ftr	t	0.499	0.211	*0.42*	21.5	*0.84*	*8*	72	*1.41*	6	*1.20*	99	*1.74*
		Header	vdd	0.499	0.423	*0.85*	21.6	*0.85*	*1*	51	*1*	41	*8.20*	44	*0.77*
		Header	+	0.499	0.423	*0.85*	20.8	*0.81*	*13*	50	*0.98*	107	*21.40*	151	*2.65*
		Hdr + Ftr	+ / -	0.499	0.011	*0.02*	14.1	*0.55*	*43*	255	*5.00*	108	*21.60*	842	*14.77*
8T	SG 3 x t; t; t; t	Nom.		0.498	0.498	--	24.5	--		50	--	4	--	43	--
		Header	t	0.498	0.455	*0.91*	22.4	*0.91*	*1*	50	*1*	6	*1.50*	40	*0.93*
		Header	vdd	0.498	0.423	*0.85*	20.8	*0.85*	*1*	50	*1*	54	*13.50*	40	*0.93*
		Hdr + Ftr	+ / -	0.497	0.012	*0.02*	13.6	*0.56*	*38*	240	*4.80*	131	*32.75*	703	*16.35*
	LP_SGR -; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	32	--	13	--
		Header	t	0.018	0.018	*1*	0.9	*1*	*1*	50	*1*	35	*1.09*	12	*0.92*
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*34*	50	*1*	77	*2.41*	21	*1.62*
	LP_INV1 t; -; vdd; t	Nom.		0.081	0.081	--	4.0	--		50	--	10	--	14	--
		Header	vdd	0.081	0.015	*0.19*	0.7	*0.19*	*1*	50	*1*	41	*4.10*	11	*0.79*
	LP_INV1.2 t; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	18	--	9	--
		Header	vdd	0.018	0.015	*1*	0.7	*0.83*	*1*	50	*1*	43	*2.39*	8	*0.89*

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

[+] Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table 6.18. Noise margin near-threshold results of headers/footers per cell

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP[+]	SNM (mV)		RSNM (mV)	
				Val	Val	*Comp*	Val	*Comp*		Val	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.499	0.499	--	25.5	--		248	--	100	--
		Header	t	0.499	0.455	*0.91*	23.3	*0.91*	*1*	243	*0.98*	94	*0.94*
		Hdr + Ftr	t	0.499	0.211	*0.42*	21.5	*0.84*	*8*	243	*0.98*	51	*0.51*
		Header	vdd	0.499	0.423	*0.85*	21.6	*0.85*	*1*	205	*0.83*	61	*0.61*
		Header	+	0.499	0.423	*0.85*	20.8	*0.81*	*13*	168	*0.68*	35	*0.35*
		Hdr + Ftr	+ / -	0.499	0.011	*0.02*	14.1	*0.55*	*43*	168	*0.68*	119	*1.19*
8T	SG 3 x t; t; t; t	Nom.		0.498	0.498	--	24.5	--		248	--		
		Header	t	0.498	0.455	*0.91*	22.4	*0.91*	*1*	243	*0.98*		
		Header	vdd	0.498	0.423	*0.85*	20.8	*0.85*	*1*	205	*0.83*		
		Hdr + Ftr	+ / -	0.497	0.012	*0.02*	13.6	*0.56*	*38*	168	*0.68*		
	LP_SGR -; -; +; t	Nom.		0.018	0.018	--	0.9	--		254	--		
		Header	t	0.018	0.018	*1*	0.9	*1*	*1*	254	*1*		
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*34*	253	*1*		
	LP_INV1 t; -; vdd; t	Nom.		0.081	0.081	--	4.0	--		260	--		
		Header	vdd	0.081	0.015	*0.19*	0.7	*0.19*	*1*	260	*1*		
	LP_INV1.2 t; -; +; t	Nom.		0.018	0.018	--	0.9	--		254	--		
		Header	vdd	0.018	0.015	*1*	0.7	*0.83*	*1*	253	*1*		

[+] Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table 6.19 shows the best leakage, delay, and average EDP near-threshold results of using header and footer FinFETs per two SRAM cells. The SNM and RSNM near-threshold results are identical to the cells using header and footer transistors per one SRAM cell. Appendix C includes more results of lower-performing cells.

Similar to the previous section, the best performing cells use only a header transistor per two SRAM cells. The leakage EDP per array in sleep-mode is reduced by up to 19% for the 6T SG cell (using header transistors), 15% for the 8T SG cell (using header transistors), 81% for the 8T LP_INV1 cell (using header transistors), and 17% for the 8T LP_SGR and 8T LP_INV1.2 cells (using header transistors). The percent reductions of the 8T cells are slightly smaller compared to using header and footer FinFETs per cell, and are slightly smaller than the reductions seen for a supply voltage of 1 V; The other cell's percent reductions are similar.

Table 6.19. Leakage, delay, EDP near-threshold results of headers/footers per two cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP⁺	32 bits × 1024 words Array					
										Delay (ps)				Ave* EDP (ps×fJ)	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.499	0.499	--	25.5	--		51	--	5	--	57	--
		Header	t	0.499	0.443	*0.89*	22.7	*0.89*	*1*	51	*1*	7	*1.40*	50	*0.88*
		Header	vdd	0.499	0.423	*0.85*	20.8	*0.81*	*4*	50	*0.98*	71	*14.20*	73	*1.28*
		Header	+	0.499	0.423	*0.85*	20.8	*0.81*	*43*	50	*0.98*	179	*35.80*	382	*6.70*
8T	SG 3 x t; t; t; t	Nom.		0.498	0.498	--	24.5	--		50	--	4	--	43	--
		Header	t	0.498	0.443	*0.89*	21.8	*0.89*	*1*	50	*1*	8	*2.00*	38	*0.88*
		Header	vdd	0.498	0.423	*0.85*	20.8	*0.85*	*11*	50	*1*	91	*22.75*	100	*2.33*
		Header	+	0.496	0.423	*0.85*	20.8	*0.85*	*67*	50	*1*	205	*51.25*	448	*10.42*
	LP_SGR -; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	32	--	13	--
		Header	t	0.018	0.018	*1*	0.9	*1*	*1*	50	*1*	38	*1.19*	12	*0.92*
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*136*	50	*1*	137	*4.28*	46	*3.54*
	LP_INV1 t; -; vdd; t	Nom.		0.081	0.081	--	4.0	--		50	--	10	--	14	--
		Header	t	0.081	0.035	*0.43*	1.7	*0.43*	*1*	50	*1*	14	*1.40*	13	*0.93*
		Header	vdd	0.081	0.015	*0.19*	0.7	*0.19*	*3*	50	*1*	71	*7.10*	20	*1.43*
	LP_INV1.2 t; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	18	--	9	--
		Header	t	0.018	0.018	*1*	0.9	*1*	*1*	50	*1*	22	*1.22*	9	*1*
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*18*	50	*1*	70	*3.89*	13	*1.44*

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

⁺ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table 6.20 shows the best leakage, delay, and average EDP near-threshold results of using header and footer FinFETs per four SRAM cells. The SNM and RSNM near-threshold results are identical to the cells using header and footer transistors per one SRAM cell. Appendix C includes more results of lower-performing cells.

Similar to the previous sections, the best performing cells often use only a header transistor per four SRAM cells. The leakage EDP per array in sleep-mode is reduced by up to 22% for the 6T SG cell (using header transistors), 15% for the 8T SG cell (using header transistors), 82% for the 8T LP_INV1 cell (using header transistors), 17% for the 8T LP_SGR cell (using header transistors), and 20% for the 8T LP_INV1.2 cell (using header transistors). Similar to the header and footer transistors per two SRAM cells, the percent reductions of the 8T cells are slightly smaller compared to using header and footer FinFETs per cell, and are slightly smaller than the reductions seen for a supply voltage of 1 V; The other cell's percent reductions are similar.

Table 6.20. Leakage, delay, EDP near-threshold results of headers/footers per four cells

	Scheme *Pass; Inv-N; Inv-P; Read*	**Leak. Tran.**	**Leak. Tran. Config.**	**Leak. / Cell (nA)**	**Leakage / Cell in Sleep-Mode (nA)**		**Leakage EDP / Array in Sleep-Mode (ps×fJ)**		**# Arrays for Break-Even EDP$^+$**	**32 bits × 1024 words Array**					
										Delay (ps)				**Ave* EDP (ps×fJ)**	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.499	0.499	--	25.5	--		51	--	5	--	57	--
		Header	t	0.499	0.434	*0.87*	22.2	*0.87*	*1*	51	*1*	9	*1.80*	48	*0.84*
		Header	vdd	0.499	0.423	*0.85*	20.8	*0.81*	*19*	50	*0.98*	123	*24.60*	192	*3.37*
		Header	+	0.501	0.423	*0.85*	20.0	*0.78*	*98*	49	*0.96*	294	*58.80*	949	*16.65*
8T	SG 3 x t; t; t; t	Nom.		0.498	0.498	--	24.5	--		50	--	4	--	43	--
		Header	t	0.498	0.434	*0.87*	21.3	*0.87*	*1*	50	*1*	12	*3.00*	38	*0.88*
		Header	vdd	0.498	0.423	*0.85*	20.8	*0.85*	*35*	50	*1*	149	*37.25*	248	*5.77*
	LP_SGR -; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	32	--	13	--
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*343*	50	*1*	255	*7.97*	97	*7.46*
	LP_INV1 t; -; vdd; t	Nom.		0.081	0.081	--	4.0	--		50	--	10	--	14	--
		Header	t	0.081	0.027	*0.33*	1.3	*0.33*	*1*	50	*1*	17	*1.70*	13	*0.93*
		Header	vdd	0.081	0.015	*0.19*	0.7	*0.19*	*7*	50	*1*	122	*12.20*	45	*3.21*
		Header	+	0.079	0.015	*0.19*	0.7	*0.18*	*30*	49	*0.98*	283	*28.30*	167	*11.93*
	LP_INV1.2 t; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	18	--	9	--
		Header	t	0.018	0.018	*1*	0.9	*1*	*1*	50	*1*	26	*1.44*	9	*1*
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*67*	50	*1*	118	*6.56*	25	*2.78*
		Header	+	0.018	0.015	*0.83*	0.7	*0.80*	*192*	49	*0.98*	271	*15.06*	65	*7.22*

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

$^+$ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Overall, compared to the 1 V supply voltage, header and footer FinFETs are slightly less effective at near-threshold operation for reducing the leakage of SRAM cells. The 6T and 8T SG cells have the most leakage of the five examined cells, and saw reduced percent reductions of leakage EDP per array in sleep-mode at V_{DD} = 0.6 V compared to V_{DD} = 1 V. The 8T LP cells witnessed similar percent reductions in leakage EDP per array in sleep-mode at the 0.6 V supply voltage compared to the 1 V supply voltage, but these reductions are smaller since these cells already reduce leakage current due to reverse-biasing of the cross-coupled inverter FinFETs. Since, for this 0.6 V operating voltage as well as for a V_{DD} of 1 V, all LP 8T cells yield less leakage EDP than using header or footer FinFETs for the 8T SG cell, it is more efficient, in terms of area and leakage current, to instead use the LP cells for low-leakage FinFET SRAMs for both near-threshold and full-V_{DD} operation.

Chapter 7

Conclusions

Nine novel 8T FinFET SRAM cell schemes and numerous configurations of these schemes have been comprehensively studied. The effects of LP and SG FinFET configurations and parameter, voltage, and temperature (PVT) variations on performance metrics are shown. Some of most important results of this study include:

- ***Reverse-biasing the cross-coupled inverter FinFETs' back gates reduces leakage current by 2-97%, depending on the biasing approach, versus using shorted-gate (SG) configuration:*** The largest reduction is obtained when the n- and p-back gates are biased to -0.2 V and V_{DD} + 0.2 V, respectively. This also minimizes the magnitude of leakage current variations due to parameter or temperature variations, which in turn reduces variations in average energy and EDP.
- ***Reverse-biasing the cross-coupled inverter FinFETs' back gates also enables over a 22% higher SNM than using SG configuration:*** The highest SNM is obtained when the n- and p-back gates are biased to -0.2 V and V_{DD}, respectively.
- ***Reverse-biasing the cross-coupled inverter FinFETs' back gates and SG-configuring the pass transistors also reduces write energy by 47-51%:*** The largest reduction is obtained when the n- and p-back gates are biased to -0.2 V and V_{DD} + 0.2 V, respectively.
- ***For a 32×1024 array, read delay is dominant:*** This delay can be mitigated by using SG-configured read transistors.

- ***8T FinFET schemes outperform both 6T FinFET and 8T CMOS cells:*** LP_INVP, the best-performing 6T scheme in terms of average EDP for both array sizes, has 17-23% and 37-38% less average EDP as an 8T scheme for the 16×16 and 32×1024 array sizes, respectively. The 8T CMOS cell has a 5X larger average EDP than the worst-performing 8T FinFET cell, the slowest 8T FinFET cell's maximum delay is 35% faster than the 8T CMOS cell's delay, and the 8T CMOS cell's leakage current is 10X more than the highest 8T FinFET cell leakage of 0.59 nA.

- ***The 6T and 8T FinFET SRAM cell delays are not greatly affected by parameter variations:*** The standard deviations of these cell delays are all below 13% of their respective means.

- ***Temperature variation greatly affects leakage current, causing increases of up to 32X:*** This increase in leakage causes an increase in average EDP of up to 4X for schemes which reverse-bias the cross-coupled inverter FinFETs' n- and p-back gates to -0.2 V and V_{DD} + 0.2 V, respectively, and up to 13X for schemes which do not use reverse-biasing.

- ***Similar relative performance is observed for the cells at near-threshold operation:*** When operating at a supply voltage of 0.6 V, a majority of the cells see a reduction in EDP and the best-performing cell, 8T LP_INV1.2, still performs the best. However, the cells see increased changes in performance due to PVT variations and increased delay. The increased delay can best be mitigated by using read-line boosting; however, word-line/write-line and read-line boosting can reduce EDP.

- ***FinFET SRAM cells designed for low-leakage are more effective than header and/or footer transistors added to a cell to reduce leakage current:*** At both 1 V and 0.6 V supply voltages, SRAM cells which use LP-configured FinFETs have larger percentage savings in leakage current versus SG-configured SRAM cells than do FinFET SRAM cells with header and/or footer FinFETs added to reduce leakage. Using LP FinFET SRAM schemes is more advantageous since minimal area and delay penalties are incurred versus SG FinFET SRAM schemes.

- ***The application of FinFETs to memory address decoders also provides substantial improvement in performance:*** Compared to a minimum-sized low power scheme, up to a 60% increase in speed and a 99.7% reduction in static/inactive power can be obtained with only a 12% increase in dynamic power by using alternative FinFET configurations for a NOR address decoder. This novel work and its results are presented in Appendix B in order to not interfere with the flow of the dissertation.

The 8T LP_INV scheme is the best-performing FinFET SRAM scheme at full-V_{DD} (1 V supply voltage) operation. In particular, the LP_INV1.2 cell has an EDP up to 60% less than the conventional 8T SG FinFET SRAM scheme and an EDP up to 62% less than the best-performing 6T FinFET SRAM scheme for a 32 bit by 1024 word array. This cell also performs best under PVT variations at this supply voltage. The LP_INV1.2 cell is also the best-performing FinFET SRAM cell at near-threshold operation with V_{DD} = 0.6 V, however, the 8T LP_SGR cell performs better under PVT variations at 0.6 V V_{DD}; at this voltage, the LP_SGR cell has less performance variation for parameter and supply voltage variations, and a similar change in performance for temperature variations. The tradeoff is that the LP_SGR cell has a slightly higher average EDP than the LP_INV1.2 cell. The results presented by this research can be weighed and assist choosing of a SRAM cell for a high-performance and/or low-power FinFET-based memory.

7.1 Contributions

This research includes a number of contributions concerning SRAM design using FinFET technology, including the following:

- ***This research is the first to conduct a comprehensive study of 8T SRAM design using FinFETs:*** A number of FinFET configurations have been proposed, evaluated, and analyzed; tradeoffs among the different figures of merit, such as delay, power, leakage current, and static noise margin, have been identified and evaluated. The FinFET-based 8T SRAM design configurations include: SG and LP FinFET, BG bias voltage, number of FinFETs per transistor. In this study was a thorough investigation for the best configurations that provide high speed, low leakage current, low energy, low EDP, high SNM, and/or high RSNM. The changes in performance have been analyzed for the SRAM cells under process, voltage, and temperature (PVT) variations to identify resilient design configurations.
- ***A thorough investigation of the effect of FinFET back-gate bias voltage variability on SRAM performance:*** Typically for PVT variations, only the supply voltage is varied for voltage variations. However, some FinFET SRAM cells which use LP-configured FinFETs require BG bias voltages which are also susceptible to variations voltage values. For the best-performing FinFET SRAM cells which require BG bias voltages, these voltages were varied for the n-type and p-type FinFETs to discover the impact on cell performance.
- ***Novel insights of header and footer FinFETs for low-leakage SRAM cells:*** Header and footer transistors can be added to decrease the leakage current of SRAM cells. Research

on the use of header and footer transistors for FinFET SRAMs is extremely limited, so this research provides new insights of leakage current savings of using header and footer transistors and compared them to the reduced leakage current obtained by using LP-configured FinFET SRAM cells.

- ***Study of near-threshold operation of FinFET SRAMs:*** The best-performing FinFET SRAM cells at the nominal 1 V supply voltage were further examined at a V_{DD} of 0.6 V. This near-threshold operation can be used to save substantial energy with a tradeoff of moderately longer delays and lower values of noise margins. To improve speed, boosting of the 6T SRAM word-line and 8T SRAM word-line/write-line and read-line were examined. PVT variations were also studied at this lower supply voltage to identify the configurations of more-resilient FinFET SRAM cells. Also for near-threshold operation, the use of header and footer FinFETs for low-leakage SRAM cells was analyzed.
- ***First, novel design of NOR address decoders using FinFETs:*** The novel design of a peripheral component of a SRAM memory sub-system, the memory address decoder, was studied using FinFETs and is presented in Appendix B. SG- and LP-configured FinFETs, with varying input signal swing from the address lines, were used in the designs of NOR address decoders to identify the best configurations to use for highest speed, lowest power, and lowest leakage current. The effect of parameter, bias voltage, and temperature variations were studied for the best-performing schemes to identify resilient design attributes.

7.2 Future Work

Future research on FinFET SRAMs can delve into multiple areas, including the following:

- ***Analyzing the thermal performance of FinFET SRAMs:*** FinFETs can be more-densely packed in an area footprint than bulk-CMOS transistors. Because of this, and compounded since FinFETs are often manufactured as a silicon-on-insulator (SOI) device, FinFETs can suffer from significant self-heating. Self-heating also generates heat dissipation to other FinFETs in the circuit and increases leakage current, delay, energy, and changes in performance due to PVT variations. Electro-thermal co-simulation can be used to examine the impact of self-heating on the entire FinFET SRAM array. This research has made an introductory, exploratory study of FinFET SRAM thermal performance in Appendix D; however, a more-detailed effort with a more-accurate simulation framework should examine this in greater detail.
- ***Studying FinFET SRAM performance using a future, scaled FinFET technology sizing:*** The results obtained for this research used a gate length (L_G) of 30 nm and a fin height (H_{fin}) of 75 nm for the FinFET technology. In Section B.1 of Appendix B, a future, scaled technology sizing with L_G = 13 nm and H_{fin} = 30 nm was used for the FinFET NOR decoders. Preliminary results of this research using this smaller technology were obtained, however, the performance of the SRAM arrays was significantly worse for the smaller technology compared to the larger technology used in this research. If proper values for the other parameters can be found or calculated, then it would be interesting to explore the performance of FinFET SRAMs at this smaller technology size since the effects of BG biasing and PVT variations would be greater.

- ***Design of FinFET SRAM peripheral circuitry:*** 6T and 8T SRAM cells were studied in this research using FinFET technology. While Section B.1 of Appendix B discusses the design of NOR address decoders using FinFETs, other memory address designs, including those in Section B.2 of Appendix B realized using bulk-CMOS transistors, can use FinFETs to obtain savings in delay, leakage, and energy. In addition, while some research is available [13], a more in-depth look at read sense-amplifier design using FinFETs can also be explored. Examining the use of FinFET technology in both memory address decoders and read sense-amplifiers can reveal the potential benefits of using FinFETs in the design of a complete SRAM sub-system.

- ***Design of SRAM sub-system communication architecture and control logic:*** In conjunction with the previous topic, the design of a complete SRAM sub-system must also examine the control logic needed for the required signals and the communication architecture of the sub-system. If adaptive BG biasing is used for the FinFET SRAMs, then the control logic and communication architecture become increasingly complex. However, the additional circuitry and power may be offset by additional savings from a finely-controlled FinFET SRAM memory. Other research in [32] uses adaptive BG biasing in a FinFET-based SRAM sub-system, however, it uses a simpler BG biasing strategy of only BG biasing the Pass transistors. This requires design and analysis at the system level in order to efficiently apply FinFET SRAMs to other applications, such as a medium-grained reconfigurable architecture for digital signal processing whose basic elements are comprised of small SRAM arrays [33].

References

[1] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur and H.-S. P. Wong, "Device scaling limits of Si MOSFETs and their application dependencies," *Proc. of the IEEE,* vol. 89, no. 3, pp. 259-288, Mar. 2001.

[2] "International Technology Roadmap for Semiconductors," [Online]. Available: www.itrs.net.

[3] T.-C. Chen, "Overcoming research challenges for CMOS scaling: Industry directions," in *Proc. Int. Conf. on Solid-State and IC Technology,* pp. 4-7, Oct. 2006.

[4] T.-J. King, "FinFETs for nanoscale CMOS digital integrated circuits," in *Proc. Int. Conf. Computer-Aided Design,* pp. 207-210, Nov. 2005.

[5] J.-P. Colinge, "The SOI MOSFET: From single gate to multigate," in *FinFETs and Other Multi-Gate Transistors*, 1st ed., J.-P. Colinge, Ed., New York, Springer, 2008, pp. 1-48.

[6] D. E. Duarte, N. Vijaykrishnan and M. J. Irwin, "A clock power model to evaluate impact of architectural and technology optimizations," *IEEE Trans. VLSI Systems,* vol. 10, no. 6, pp. 844-855, Dec. 2002.

[7] A. Muttreja, N. Agarwal and N. K. Jha, "CMOS logic design with independent-gate FinFETs," in *Proc. IEEE Int. Conf. on Computer Design,* pp. 560-567, Oct. 2007.

[8] C.-Y. Lee and N. K. Jha, "FinFET-based dynamic power management of on-chip interconnection networks through adaptive back-gate biasing," in *Proc. IEEE Int. Conf. on Computer Design,* pp. 350-357, Oct. 2009.

[9] J. G. Fossum, L. Ge, M.-H. Chiang, V. P. Trivedi, M. M. Chowdhury, L. Mathew, G. O. Workman and B.-Y. Nguyen, "A process/physics-based compact model for nonclassical CMOS device and circuit design," *Solid-State Electronics,* vol. 48, no. 6, pp. 919-926, Jun. 2004.

[10] J. G. Fossum, V. P. Trivedi, M. M. Chowdhury, S.-H. Kim and W. Zhang, "Recent upgrades and applications of UFDG," in *Proc. 2006 NSTI Nanotech. Conf. (Workshop on Compact Modeling),* pp. 674-679, May 2006.

[11] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm design exploration," in *Proc. Int. Symp. Quality of Electronic Design,* pp. 585-590, May 2006, http://www.eas.asu.edu/~ptm.

[12] M. V. Dunga, C.-H. Lin, A. M. Niknejad and C. Hu, "BSIM-CMG: A compact model for multi-gate transistors," in *FinFETs and Other Multi-Gate Transistors*, 1st ed., J.-P. Coligne, Ed., New York, Springer, 2008, pp. 113-153.

[13] M.-L. Fan, V. P.-H. Hu, Y.-N. Chen, P. Su and C.-T. Chuang, "Variability analysis of sense-amplifier for FinFET subthreshold SRAM applications," *IEEE Tran. Circuits Syst. II, Exp. Briefs,* vol. 59, no. 12, pp. 878-882, Dec. 2012.

[14] S. A. Tawfik and V. Kursun, "Compact FinFET memory circuits with p-type data access transistors for low leakage and robust operation," in *Proc. Int. Symp. Quality Electronic Design,* pp. 855-860, Mar. 2008.

[15] A. Carlson, Z. Guo, S. Balasubramanian, R. Zlatanovici, K. Liu T-J and B. Nikolic, "SRAM read/write margin enhancements using FinFETs," *IEEE Trans. VLSI Systems,* vol. 18, no. 6, pp. 887-900, Jun. 2010.

[16] E. Seevinck, F. J. List and J. Lohstroh, "Static-noise margin analysis of MOS SRAM cells," *IEEE J. Solid-State Circuits,* vol. SC-22, no. 5, pp. 748-754, Oct. 1987.

[17] A. Bhavnagarwala, S. Kosonocky, C. Radens, K. Stawiasz, R. Mann, Q. Ye and K. Chin, "Fluctuation limits and scaling opportunities for CMOS SRAM cells," in *Proc. IEEE Int. Electron Devices Meeting,* pp. 659-662, Dec. 2005.

[18] Z. Guo, S. Balasubramanian, R. Zlatanovici, T.-J. King and B. Nikolic, "FinFET-based SRAM design," in *Proc. IEEE Int. Symp. on Low Power Electronics and Design,* pp. 2-7, Aug. 2005.

[19] J. G. Delgado-Frias, Z. Zhang and M. A. Turi, "Low-power SRAM cell design for FinFET and CNTFET technologies," in *Proc. 1st IEEE Workshop of Low Power SoC in Int. Green Computing Conf.,* pp. 547-553, Aug. 2010.

[20] A. Singhee and R. A. Rutenbar, "From finance to flip flops: A study of fast quasi-Monte Carlo methods from computational finance applied to statistical circuit analysis," in *Proc. IEEE Int. Symp. on Quality Electronic Design,* pp. 685-692, Mar. 2007.

[21] S. A. Tawfik and V. K. Kursun, "Work-function engineering for reduced power and higher integration density: An alternative to sizing for stability in FinFET memory circuits," in *Proc. IEEE Int. Symp. Circuits Syst.,* pp. 788-791, May 2008.

[22] J. Rabaey, A. Chandrakasan and B. Nikolic, Digital Integrated Circuits: A Design Perspective, Upper Saddle River, NJ: Prentice Hall, 2003.

[23] K. A. Blomster, "Schemes for reducing power and delay in SRAMs," M.S. thesis, Washington State University, Aug. 2006.

[24] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester and T. Mudge, "Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits," *Proc. of the IEEE,* vol. 98, no. 2, pp. 253-266, Feb. 2010.

[25] G. Chen, D. Sylvester, D. Blaauw and T. Mudge, "Yield-driven near-threshold SRAM design," *IEEE Trans. VLSI Systems,* vol. 18, no. 11, pp. 1590-1598, Nov. 2010.

[26] V. P.-H. Hu, M.-L. Fan, P. Su and C.-T. Chuang, "Threshold voltage design of UTB SOI SRAM with improved stability/variability for ultra-low voltage near subthreshold operation," *IEEE Trans. Nanotechnology,* to appear.

[27] J. Mezhibovsky, A. Teman and A. Fish, "Low voltage SRAMs and the scalability of the 9T supply feedback SRAM," in *Proc. IEEE Int. SoC Conf.,* pp. 136-141, Sept. 2011.

[28] C. Hu, "Thin-body FinFET as scalable low voltage transistor (Keynote)," in *Proc. IEEE Int. Symp. VLSI Tech., Syst., and Applications,* Apr. 2012.

[29] F. Crupi, M. Alioto, J. Franco, P. Magnone, M. Togo, N. Horiguchi and G. Groeseneken, "Understanding the basic advantages of bulk FinFETs for sub- and near-threshold logic circuits from device measurements," *IEEE Trans. Circuits Syst. II, Exp. Briefs,* vol. 59, no. 7, pp. 439-442, Jul. 2012.

[30] M.-L. Fan, Y.-S. Wu, V. P.-H. Hu, P. Su and C.-T. Chuang, "Investigation of cell stability and write ability of FinFET subthreshold SRAM using analytical SNM model," *IEEE Trans. Electron Devices,* vol. 57, no. 6, pp. 1375-1381, Jun. 2010.

[31] C.-Y. Hsieh, F. M-L, V. P.-H. Hu, P. Su and C.-T. Chuang, "Independently-controlled-gate FinFET Schmitt trigger sub-threshold SRAMs," *IEEE Trans. VLSI Systems,* vol. 20, no. 7, pp. 1201-1210, Jul. 2012.

[32] S.-I. O'uchi, K. Endo, M. Masahara, K. Sakamoto, Y. Liu, T. Matsukawa, T. Sekigawa, H. Koike and E. Suzuki, "Flex-pass-gate SRAM for static noise margin enhancement using FinFET-based technology," *Solid-State Electronics,* vol. 52, no. 11, p. 1694–1702, Nov. 2008.

[33] M. J. Myjak, "A medium-grain reconfigurable architecture for digital signal processing," Ph.D. dissertation, Washington State University, May 2006.

[34] B. S. Amrutur and M. A. Horowitz, "Fast low-power decoders for RAMs," *IEEE J. Solid-State Circuits,* vol. 36, no. 10, pp. 1506-1515, Oct. 2001.

[35] K. W. Mai, T. Mori, B. S. Amrutur, R. Ho, B. Wilburn, M. A. Horowitz, I. Fukushi, T. Izawa and S. Mitarai, "Low-power SRAM design using half-swing pulse-mode techniques," *IEEE J. Solid-State Circuits,* vol. 33, no. 11, pp. 1659-1671, Nov. 1998.

[36] M. A. Turi and J. G. Delgado-Frias, "High-performance low-power selective precharge schemes for address decoders," *IEEE Trans. Circuits Syst. II, Exp. Briefs,* vol. 55, no. 9, pp. 917-921, Sept. 2008.

[37] M. A. Turi and J. G. Delgado-Frias, "High-performance low-power AND and sense-amp address decoders with selective precharging," in *Proc. IEEE Int. Symp. Circuits Syst.,* pp. 1464-1467, May 2008.

[38] C. A. Zukowski and S.-Y. Wang, "Use of selective precharge for low-power content-addressable memories," in *Proc. IEEE Int. Symp. Circuits Syst.,* pp. 1788-1791, Jun. 1997.

[39] M. A. Turi, J. G. Delgado-Frias and N. K. Jha, "Low-power FinFET design schemes for NOR address decoders," in *Proc. IEEE Int. Symp. VLSI Design, Automation, and Test,* pp. 74-77, Apr. 2010.

[40] M. A. Turi and J. G. Delgado-Frias, "Decreasing energy consumption in address decoders by means of selective precharge schemes," *Microelectronics Journal,* vol. 40, no. 11, pp. 1590-1600, Nov. 2009.

[41] S. Rodriguez and B. Jacob, "Energy/power breakdown of pipelined nanometer caches (90nm/65nm/45nm/32nm)," in *Proc. IEEE Int. Symp. Low Power Electronics and Design,* pp. 25-30, Oct. 2006.

[42] D. Schmitt-Landsiedel, B. Hoppe, G. Neuendorf, M. Wurm and J. Winnerl, "Pipelined architecture for fast CMOS buffer RAM's," *IEEE J. Solid-State Circuits,* vol. 25, no. 3, pp. 741-747, Jun. 1990.

[43] M. A. Turi and J. G. Delgado-Frias, "Reducing power in memory decoders by means of selective precharge schemes," in *Proc. IEEE Int. Midwest Symp. Circuits Syst.,* pp. 956-959, Aug. 2007.

[44] B. Swahn and S. Hassoun, "Electro-thermal analysis of multi-fin devices," *IEEE Trans. VLSI Systems,* vol. 16, no. 7, pp. 816-829, Jul. 2008.

[45] E. Pop, R. Dutton and K. Goodson, "Thermal analysis of ultra-thin body device scaling," in *Proc. IEEE Int. Electron Devices Meeting,* pp. 36.6.1-36.6.4, Dec. 2003.

[46] M. Fulde, J. P. Engelstädter, G. Knoblinger and D. Schmitt-Landsiedel, "Analog circuits using FinFETs: benefits in speed-accuracy-power trade-off and simulation of parasitic effects," *Advances in Radio Science,* vol. 5, pp. 285-290, 2007.

[47] University of Florida, SOI Group, "UFDG MOSFET model (Linux ver. 3.7) user's guide," Sept. 2009.

Appendix A

Simulation Scripts

This appendix includes a sample of the primary Perl and tcl scripts used for simulation and data collection for the FinFET SRAM research. These scripts include:

- *run_ufdg.pl:* A Perl script to more easily run a UFDG FinFET simulation
- *netgen_ufdg.pl:* A Perl script to clean-up a netlist (remove comments, blank lines, insert .include files, and make .param substitutions) and assist the "init_batch_ufdg.pl" script with variations of .param variables
- *mkout_ufdg.pl:* A Perl script to convert the UFDG simulator's ".o" output to something usable by other programs (e.g. waveform viewers, MS Excel, etc.)
- *init_batch_ufdg.pl:* A Perl script to initialize Spice decks for a Monte Carlo, corner, or parameter sweep simulation
- *batchexec_ufdg.pl:* A Perl script to simulate the Spice decks set up by init_vary_ufdg.pl
- *meas_ezwave.pl:* A Perl script to call a tcl script to make measurements from Mentor Graphics EZWave waveform viewer
- *meas_stub.tcl:* A tcl script example/template of making measurements from Mentor Graphics EZWave waveform viewer

A.1 run_ufdg.pl

```
#!/usr/bin/perl -w
use strict;
use warnings;

my $Version_Date = 'Apr. 15, 2012';
my $Script_Name = 'run_ufdg.pl';
```

```
# Requires scripts:
my $Mkout_Ufdg = 'mkout_ufdg.pl';
my $Netgen_Ufdg = 'netgen_ufdg.pl -zi';

# Constants:
my $Fmt_Input = '.i';
my $Fmt_Output = '.o';
my $Fmt_Raw = '.raw';

my $Ufdg_3_7_Spice = 'ngspice3.ufdg-3.7';
my $Ufdg_3_6_Spice = 'spice3.ufdg-3.6';

my $Timer_Tick_Amt = 10; # Number of seconds per 'tick'
my $Simlen_Filename = 'run_ufdg_simlength.csv';

# Signal (interrupt) handler for SIGUSR1 (#10):
my $simtime = 0;
# Procedure: get_simtime_handler
# Summary:
#   Returns the simulation duration upon exiting
sub get_simtime_handler {
  exit $simtime;
}
$SIG{'USR1'} = \&get_simtime_handler;

# Signal (interrupt) handler for SIGTERM (#15):
my $ufdg_pid;
my $timer_pid = -1;
# Procedure: clean_term_handler
# Summary:
#   If the terminate signal is given, also kills each simulation child process (ufdg
                           and timer)
sub clean_term_handler {
  kill 'TERM', $ufdg_pid if defined $ufdg_pid;
  kill 'TERM', $timer_pid if defined $timer_pid and $timer_pid > 0;
  exit 128+15;
}
$SIG{'TERM'} = \&clean_term_handler;

# Usage:
my $Usage = ''.
'Usage: '.$Script_Name."  [OPTIONS]  ufdg_deck_name\n".
"Options:\n".
    "\t-n  Let simulator (instead of ".$Netgen_Ufdg.") insert include files and make
                           .param substitutions\n".
    "\t-o  Run old version of UFDG (Solaris ver. 3.6)\n".
    "\t-r  Generate ".$Fmt_Raw." output file for simulation output (".$Mkout_Ufdg."
                           will not run; only errors/warnings in .o file)\n".
    "\tFor Timer:\n".
    "\t\t-#  Run with timer length=\"#\" sec; kill UFDG if timer expires\n".
    "\t\t-t  Run with no timer\n".
    "\t\t-p  Print (append) the simulation duration to file
                           \"".$Simlen_Filename."\"\n".
    "\tFor ".$Mkout_Ufdg." (by default, all output formats are generated):\n".
    "\t\t-m  Do not create additional output formats (do not call ".$Mkout_Ufdg.")\n".
    "\t\t-c  Create .csv output for Mentor Graphics EZwave\n".
    "\t\t-s  Create .sti output for Mentor Graphics EZWave\n".
    "\t\t-x  Create .txt output for Synopsys CosmosScope\n";
my $Min_Arg_Num = 1;

# run_ufdg.pl
#
```

```
# Created by Mike Turi
# Washington State University
# School of Electrical Engineering & Computer Science
# High Performance Computer Systems (HiPerCopS) Group
#
# Note: Code is semi-tested; may fail under certain conditions
#
# This is a Perl script used to more-easily run the
# University of Flordia Double-Gate (UFDG) MOSFET model
# (e.g. FinFETs).  Only the spice file's deck name needs to
# be passed as an argument.  This can also generate the .raw
# output file for the Nutmeg simulator to use (but if .raw file
# is generated, then the .o file has only error/warning info and
# mkout_ufdg.pl will not run).  A timer is also supplied to
# display the simulation's run-time.  It can also be used to kill
# the simulation if it runs too long.  This works for UFDG Linux
# ver. 3.7 and UFDG Solaris ver. 3.6 (but unsure [10/2010] if
# .raw file creation works in UFDG 3.6).
#
# Usage: Provide the UFDG deck name and any additional options.
#        Creates "ufdg_deck_name.o" (plus .csv and/or .txt output
#        formats from mkout_ufdg.pl) from "ufdg_deck_name.i".

print "**$Script_Name version date: $Version_Date**\n";
die $Usage if scalar @ARGV < 1;
my $argnum = scalar @ARGV;
die "Error: Not enough parameters\n\n$Usage" if $argnum < $Min_Arg_Num;

my $ufdg_spice = $Ufdg_3_7_Spice; # Use UFDG Linux ver. 3.7 by default

my $i = 0;
my $arg = shift;
my $op_sim_sub;
my $op_make_raw;
my $timer_val = 0;
my $op_print_simlen;
my $op_no_mkout;
my $mkout_opts;
while( $arg =~ m/^-/ ) {
  die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
  if( $arg =~ m/^-(\S+)/ ) { # Found options
    my $argmatch = $1;
    if( $argmatch =~ m/n/i ) { # Let simulator insert include files and make .param
                          substitutions
      $op_sim_sub = 1;
    }
    if( $argmatch =~ m/o/i ) { # Use old UFDG version (Solaris ver. 3.6)
      $ufdg_spice = $Ufdg_3_6_Spice;
    }
    if( $argmatch =~ m/r/i ) { # Generate .raw output file
      $op_make_raw = 1;
    }
    if( $argmatch =~ m/(\d+)/ ) { # Set timer to numerical value
      $timer_val = $1;
    }
    if( $argmatch =~ m/t/i ) { # Run with no timer
      $timer_val = -1 unless defined $op_print_simlen;
    }
    if( $argmatch =~ m/p/i ) { # Print the simulation duration to file
      $op_print_simlen = 1;
      $timer_val = 0 if $timer_val < 0;
    }
    if( $argmatch =~ m/m/i ) { # Do not run mkout_ufdg.pl (do not create additional
                           output formats)
      $op_no_mkout = 1;
    }
    if( $argmatch =~ m/c/i ) {
      $mkout_opts = (defined $mkout_opts) ? $mkout_opts.'c' : 'c'; # Create .csv
                           output for Mentor Graphics EZwave
    }
    if( $argmatch =~ m/s/i ) {
      $mkout_opts = (defined $mkout_opts) ? $mkout_opts.'s' : 's'; # Create .sti
                           output for Mentor Graphics EZwave
    }
    if( $argmatch =~ m/x/i ) {
      $mkout_opts = (defined $mkout_opts) ? $mkout_opts.'x' : 'x'; # Create .txt
                           output for Synopsys CosmosScope
    }
  }
  $arg = shift;
}
my $deckname = $arg;

# --- Check input file and delete output files ---
my $infile = $deckname.$Fmt_Input;
my $outfile = $deckname.$Fmt_Output;
my $rawfile = $deckname.$Fmt_Raw;
# Delete output files if it already exists; sometimes UFDG output appends
system( ('rm', '-f', $outfile) ) == 0 or die 'Error: "rm -f ', $outfile, '" somehow
                           failed';
system( ('rm', '-f', $rawfile) ) == 0 or die 'Error: "rm -f ', $rawfile, '" somehow
                           failed';
die 'Error: Cannot find "', $infile, '"' unless -e $infile;

# --- First fork(); the child is UFDG ---
$ufdg_pid = fork;
die "Error: Fork of ufdg spice failed" unless defined $ufdg_pid;
if( $ufdg_pid == 0 ) { # The child
  # Do I/O redirection manually
  close( STDIN );
  if( defined $op_sim_sub ) {
    open STDIN, "<$infile" or die 'Error: Cannot redirect UFDG input to "', $infile,
                           '"';
  } else {
    open STDIN, "$Netgen_Ufdg $infile |" or die 'Error: Cannot redirect UFDG input to
                           "', $Netgen_Ufdg, ' ', $infile, ' |"';
  }
  close( STDOUT );
  open STDOUT, ">$outfile" or die 'Error: Cannot redirect UFDG output to "', $outfile,
                           '"';
  #chmod the output file for write? (used to be required for ver. 3.6)
  #setenv setup for UFDG? (used to be required for ver. 3.6)

  # Add rawfile option if needed, then exec
  $ufdg_spice = $ufdg_spice.' -r '.$rawfile if defined $op_make_raw;
  exec $ufdg_spice or die 'Error: Could not exec "', $ufdg_spice, '"';
}

# --- Second fork(); the child is the timer ---
$timer_pid = fork if $timer_val >= 0;
die 'Error: Fork of timer failed' unless defined $timer_pid;
if( $timer_pid == 0 ) { # The child
  $simtime = 1;
  if( $timer_val > 0 ) { # Run timer with alarm
    my $ticks = $timer_val / $Timer_Tick_Amt;
    my $rem = $timer_val % $Timer_Tick_Amt;
    while( $simtime <= $ticks ) {
      sleep $Timer_Tick_Amt;
      print $Timer_Tick_Amt*$simtime++, "s elapsed...\n";
    }
    if( $rem > 0 ) {
      sleep $rem;
      print $timer_val, "s elapsed...\n";
    }
  } else { # Just display timer and run forever
    while(1) {
      sleep $Timer_Tick_Amt;
      print $Timer_Tick_Amt*$simtime++, "s elapsed...\n";
    }
  }
  exit $timer_val;
}

# --- This (parent) process will wait for ufdg or timer to die ---
my $pid;
do {
  $pid = wait;
} while( $pid != $ufdg_pid and $pid != $timer_pid );
my $retval = $? >> 8;

if( defined $op_print_simlen ) {
  open OUTAPP, ">>$Simlen_Filename" or die 'Error: Cannot open ', $Simlen_Filename, '
                           for append due to', $!;
}

# --- Figure out if ufdg or timer died first; kill other child ---
# --- Also, if timer died first, kill ufdg and exit; sim was not completed ---
my $simlength;
if( $pid == $ufdg_pid ) {
  print 'ufdg "', $deckname, '" died first (status=', $retval, ")--killing timer\n";
  if( $timer_val >= 0 ) { # Do a kill -SIGUSR1 "timer" and get sim length (if there is
                           a timer)
    kill 10, $timer_pid;
    do {
      $pid = wait;
    } while( $pid != $timer_pid );
    $simlength = $? >> 8;
    print $deckname, ' ran for approx. ', $simlength*$Timer_Tick_Amt, " seconds\n";
    if( defined $op_print_simlen ) {
      print OUTAPP $deckname, ',', $simlength*$Timer_Tick_Amt, "\n";
      close OUTAPP;
    }
  }
} else {
  print 'timer died first--killing ufdg "', $deckname, "\"\n";
  kill 9, $ufdg_pid; # Do a kill - 9 "ufdg"
  print $deckname, ' ran for ', $timer_val, "+ seconds\n";
  if( defined $op_print_simlen ) {
    print OUTAPP $deckname, ',', $timer_val, "+\n";
    close OUTAPP;
  }
  die 'Error: UFDG timeout; simulation was killed';
}

# --- If ufdg completed without error, make the output file(s) ---
if( $retval != 0 ) {
  print 'Warning: UFDG returned with value=', $retval, "\n";
  exit $retval;
```

```
}

# TODO: Append error messages from ".o" file to run_ufdg output
# my @spice_errors = "grep -i error $deckname.o |";
# #my @spice_errors = ('grep', '-i', 'error', "$deckname.o");
# #my $foo = system( @spice_errors );
# #print $foo;
# print @spice_errors;

exit $retval if defined $op_make_raw or defined $op_no_mkout; # Do not run
                          mkout_ufdg.pl (cannot run mkout_ufdg.pl using raw file at
                          this time)

if( defined $mkout_opts ) {
  my @mkout = ($Mkout_Ufdg, '-'.$mkout_opts, $deckname);
  system( @mkout ) == 0 or die 'Error: "', $Mkout_Ufdg, ' -', $mkout_opts, ' ',
                          $deckname, '" failed';
} else {
  my @mkout = ($Mkout_Ufdg, $deckname);
  system( @mkout ) == 0 or die 'Error: "', $Mkout_Ufdg, ' ', $deckname, '" failed';
}

exit $retval;
```

A.2 netgen_ufdg.pl

```
#!/usr/bin/perl -w
use strict;
use warnings;

my $Version_Date = 'Apr. 6, 2012';
my $Script_Name = 'netgen_ufdg.pl';

# Constants:
my $Placeholder_Prefix = 'T'; # Must be a Spice Prefix (so the simulator won't error
                          when doing a listing expand)
my $Init_Placeholder = 1000000;

# Usage:
my $Usage = ''.
"Usage: $Script_Name  [OPTIONS]\n".
"Options:\n".
    "\t-d           Print parameter details to STDOUT and substitute placeholders for
                          parameter expressions\n".
    "\t               STDOUT format:\n".
    "\t                 param1=value1\n".
    "\t                 param2=value2\n".
    "\t                 ".$Init_Placeholder.$Placeholder_Prefix.'=param_exp1 (using
                          "'.$Placeholder_Prefix."\" prefix for placeholder)\n".
    "\t                 ".($Init_Placeholder+1).$Placeholder_Prefix."=param_exp2\n".
    "\t                 * Comment (beginning of netlist if no \"-s\" option used)\n".
    "\t-i filename  Use \"filename\" for input netlist instead of STDIN\n".
    "\t-o filename  Use \"filename\" for netlist output (with/without STDOUT output--
                          look at -s)\n".
    "\t-p           Do not preform parameter substitutions on netlist\n".
    "\t-s           Do not print netlist output to STDOUT (use -o to print to file
                          instead)\n".
    "\t-z           Suppress output to STDERR (only show output for errors)\n";
my $Min_Arg_Num = 0;
```

```
# Procedure: rm_comments
# Summary:
#   Removes all commented lines in a spice netlist (1st line, blank or whitespace only
                           lines,
#   and lines beginning with the comment character "*" or whitespace).  Note: Comments
                           with
#   ";" are ignored and inline comments (e.g. vdd 1 0 dc 1V * Supply Voltage) are
                           ignored.
# Parameter: A reference to an array holding a spice netlist
sub rm_comments {
  my $aref = shift;
  shift @$aref; # Remove first line of the netlist (this is always treated as a
                           comment)
  @$aref = grep !/^\*/, @$aref; # Remove comment lines
  @$aref = grep !/^\s*$/, @$aref; # Remove blank lines (and lines with only
                           whitespace)
  @$aref = grep !/^\s+/, @$aref; # Remove lines beginning with a whitespace (oddly,
                           ngspice ignores these lines anyway)
}

# netgen_ufdg.pl
#
# Created by Mike Turi
# Washington State University
# School of Electrical Engineering & Computer Science
# High Performance Computer Systems (HiPerCopS) Group
#
# Note: Code is semi-tested; may fail under certain conditions
#
# This is a Perl script used to remove comments, remove whitespace, insert .include
                           files, and
# substitute .param variables into a netlist prior to running a simulation.  This
                           script is
# called by run_ufdg.pl and init_vary_ufdg.pl.  I have seen the University of Flordia
                           Double-Gate
# (UFDG) MOSFET (FinFET) model's Spice3 program, ngspice, incorrectly handle .param
                           variable
# substitutions (results would vary depending on whitespace and comments).  After this
                           script is
# run, ngspice will only need to substitute for subcircuits and compute math
                           expressions (where
# .param variables had been replaced by this script).
#
# Usage: By default this script takes an input script through STDIN and outputs the
                           resultant
# script to STDOUT.  Options allow the input and output to come from or go to files.
                           Parameter
# substitutions can be disabled, and details about .param variables can be printed.
                           Parameter
# details are printed to STDOUT, and are made up of sets of parameter names and
                           values, then are
# followed by sets of placeholder names and expressions.  If parameter substitution is
                           on, then
# placeholders are substituted into the netlist, which if output to STDOUT, will
                           appear directly
# following the sets of placeholder names and expressions.

print STDERR "**$Script_Name version date: $Version_Date**\n";
my $argnum = scalar @ARGV;

my $i = 0;
my $arg = shift;
my $op_param_details;
my $op_in_fname;
my $op_out_fname;
my $op_no_param_sub;
my $op_out_no_stdout;
my $op_out_no_stderr;
while( defined $arg and $arg =~ m/^-/ ) {
  if( $arg =~ m/^-(\S+)/ ) { # Found options
    my $argmatch = $1;

    # Check single letter options first
    if( $argmatch =~ m/d/i ) {
      $op_param_details = 1; # Print details about .param variables (.param name and
                           value plus placeholders)
    }
    if( $argmatch =~ m/p/i ) {
      $op_no_param_sub = 1; # Do not preform parameter substitutions on netlist
    }
    if( $argmatch =~ m/s/i ) {
      $op_out_no_stdout = 1; # Do not print output to STDOUT
    }
    if( $argmatch =~ m/z/i ) {
      $op_out_no_stderr = 1; # Do not print output to STDERR (unless there is actually
                           an error)
    }

    # Check options which require another argument
    if( $argmatch =~ m/i/i ) {
      $op_in_fname = shift; # Use this file instead of STDIN for input
    }
    if( $argmatch =~ m/o/i ) {
      $op_out_fname = shift; # Print output to this file (may be in addition to
                           STDOUT)
    }
  }
  $arg = shift;
}

print STDERR $Usage unless defined $op_out_no_stderr; # Print usage unless output to
                           STDERR is suppressed

##########
my @spice; # The spice netlist which will be modified
##########

if( defined $op_in_fname ) {
  open IN, "<$op_in_fname" or die 'Error: Cannot open ', $op_in_fname, ' due to: ',
                           $!;
  @spice = <IN>;
  die 'Error: ', $op_in_fname, ' is an empty file containing no netlist' unless scalar
                           @spice > 0;
} else {
  open IN, "<-" or die 'Error: Cannot open STDIN due to ', $!;
  @spice = <IN>;
  die 'Error: STDIN contains no data to read' unless scalar @spice > 0;
}
close IN;

##########
my $netlist_title = $spice[0]; # Save the netlist title to use for the resultant
                           netlist
##########

rm_comments \@spice; # Remove comments in the netlist
```

```
##########
my %params; # A hash of .param variable names (as key) and the respective values
my @params_ordered; # (for op_param_details) An array of the .param variable names (in
                           the order declared in the netlist); use instead of "keys
                           %params" to maintain order
##########

for( $i = 0; $i < scalar @spice; $i++ ) {
  if( $spice[$i] =~ m/^\.include\s+(\S+)/i ) { # Found an include statement: open the
                           include file and splice it into the netlist
    open IN, "<$1" or die 'Error: Cannot open ', $1, ' due to: ', $!;
    my @include = <IN>;
    close IN;

    rm_comments \@include; # Remove comments in the include file before splicing it
                           into the netlist
    splice @spice, $i, 1, @include;
    redo; # This line number ($i) is now the first line of the spliced in include file
  } elsif( $spice[$i] =~ m/^\.param\s+(\w+)\s*=\s*\{(.+)\}/i ) { # Found a param
                           definition which required other parameter(s)
    my $name = lc $1;
    my $expression = $2;
    if( defined $op_param_details ) { # Don't substitute for the expression, just
                           getting details now
      $params{$name} = '{'.$expression.'}'; # Add this .param name and value (an
                           expression) to the hash
      push @params_ordered, $name; # Add param name in ordered array
    } elsif( !defined $op_no_param_sub ) { # Now we'll substitute for the expression;
                           No reason to do this if $op_no_param_sub is chosen
      # Split the spice expression by words (.param variable names), and keep the non-
                           words (i.e. math operators)
      my @exp_parts = split /(\W)/, $expression;
      foreach (@exp_parts) {
        while((my $param_name, my $val) = each %params) {
          if( m/\w+/ ) { s/^$param_name$/$val/i; } # Substitute the value if a .param
                           variable is found
        }
      }
      $expression = join '', @exp_parts; # Combine the parts to form the expression
                           again
      $params{$name} = '('.$expression.')'; # Add parentheses to maintain order of
                           operations, and place in the hash
        # Note: Let .param X=5 and .param Y={1+1}; If another expression .param
                           PROD={X*Y}, then {5*1+1} is different than {5*(1+1)}
    }
    if( !defined $op_no_param_sub ) {
      splice @spice, $i, 1; # Remove the definition if we are substituting for .param
                           variables
      redo; # This line number ($i) is now the line directly after the .param
                           definition
    }
  } elsif( $spice[$i] =~ m/^\.param\s+(\w+)\s*=\s*(\S+)/i ) { # Found a simple param
                           definition (.param "name" = "value")
    $params{lc $1} = $2; # Add this .param name and value to the hash
    push @params_ordered, $1; # Add param name in ordered array
    if( !defined $op_no_param_sub ) {
      splice @spice, $i, 1; # Remove the definition if we are substituting for .param
                           variables
      redo; # This line number ($i) is now the line directly after the .param
                           definition
    }
  }
```

```
}

if( defined $op_param_details ) { # If printing .param details, first print the .param
                           names and the respective values
  foreach my $param_name (@params_ordered) {
    print $param_name, '=', $params{$param_name}, "\n";
  }
}

##########
my $spice_line; # A line from the spice netlist
my @spice_line_parts; # The $spice_line split up around the curly braces {} so
                           parameter substitution or details can be done
my $spice_exp_ref; # A reference to an expression in curly braces {} involving
                           possible .param variables
my @spice_exp_parts; # The $spice_exp_ref split in order to do substitution of .param
                           variables in the expression
my $placeholder = $Init_Placeholder; # The placeholder value, beginning at the initial
                           value
##########

foreach $spice_line (@spice) {
  if( $spice_line =~ m/\{.+\}/ ) {
    # Split the spice line by { and } chars. (char. class).  Keep { and } if not
                           substituting with placeholders (op_param_details)
    #   so Spice3 can calculate the math expression (placeholders substitute for the
                           entire expression).
    @spice_line_parts = defined $op_param_details ? split /[{}]/, $spice_line : split
                           /([{}])/, $spice_line;

    # Value of $i, given line "m1 d fg s bg psg l={l} w={h} m={np}"
    #  op_param_details (placeholders):  #0:"m1 d fg s bg psg l="  #1:"l"  #2:" w="
                           #3:"h"  #4:" m="  #5:"np"  #6:"\n"
    #  Normal (make substitution):       #0:"m1 d fg s bg psg l="  #1:"{"  #2:"l"
                           #3:"}"  #4:" w="  #5:"{"   #6:"h" ...
    $i = defined $op_param_details ? 1 : 2; # Location of first expression (see above)

    while( $i < scalar @spice_line_parts ) { # This is a little ugly
      $spice_exp_ref = \$spice_line_parts[$i]; # Reference the spice expression so $i
                           can be incremented
      $i += defined $op_param_details ? 2 : 4; # Location of next expression, if it
                           exists (see above)
      if( defined $op_param_details ) {
        print $placeholder, $Placeholder_Prefix, '=', ${$spice_exp_ref}, "\n"; # Print
                           the placeholder and the expression it represents
        ${$spice_exp_ref} = $placeholder.$Placeholder_Prefix unless defined
                           $op_no_param_sub; # If .param substitutions ok, then
                           substitute in the placeholder
        $placeholder++;
        next; # If getting details, don't continue farther in the loop--no need to
                           substitute for .param variables in the expression
      }
      next if defined $op_no_param_sub; # If not substituting for .param variables,
                           don't continue farther in the loop

      # Split the spice expression by words (.param variable names), and keep the non-
                           words (i.e. math operators)
      @spice_exp_parts = split /(\W)/, ${$spice_exp_ref};
      foreach (@spice_exp_parts) {
        while((my $param_name, my $val) = each %params) {
          if( m/\w+/ ) { s/^$param_name$/$val/i; } # Substitute the value if a .param
                           variable is found
        }
      }
      ${$spice_exp_ref} = join '', @spice_exp_parts; # Replace the original expression
                           with numerals and math operators, Spice3 can compute this
                           at runtime
    }
    $spice_line = join '', @spice_line_parts; # Replace the original spice line of the
                           netlist with this line with substituted expressions
  }
}

# One more substitution to do, ngspice doesn't like expressions on the .OPTION line
                           for some reason.
# Have Perl try to compute the expression if it hasn't been replaced by a placeholder
                           (getting details) and if .param variables are being
                           substituted
if( !defined $op_param_details and !defined $op_no_param_sub ) {
  foreach (@spice) {
    if( m/^\.OPTION/i ) { # Found the option line
      my @option_parts = split /[{}]/; # Split the line by the curly braces {}, but
                           don't keep them.
      my $answer;
      for( $i=1; $i < scalar @option_parts; $i+=2 ) { # Every other part will be an
                           expression (e.g. .OPTION ABSTOL=1E-6 TEMP={tempc} FOO={b*b-
                           4*a*c}\n)
        # Not sure if eval will calculate the answer, best to avoid expressions in the
                           .option line
        eval { $answer = $option_parts[$i]; }; # Try to evaluate the expression
                           (hopefully Spice3 suffixes are not used (i.e. n=1e-9, p=1e-
                           12, etc.))
        die 'Error: Cannot compute expression {', $option_parts[$i], '} on .option
                           line due to: ', $@ if $@; # Not sure if eval will actually
                           generate an error though
        $option_parts[$i] = $answer; # Replace the expression with the answer
      }
      $_ = join '', @option_parts; # Replace the .OPTION line with the updated version
    }
  }
}

if( length $netlist_title > 0 and $netlist_title =~ m/\S/ ) { # Use the original
                           netlist's title, if available, or make a new title
  $netlist_title = '* '.$netlist_title unless $netlist_title =~ m/^\*/;
} else {
  $netlist_title = "* This netlist generated by $Script_Name\n";
}

if( defined $op_out_fname ) { # If the user requested, print the netlist to the output
                           file
  open OUT, ">$op_out_fname" or die 'Error: Cannot open ', $op_out_fname, ' due to: ',
                           $!;
  print OUT $netlist_title;
  print OUT @spice;
  print OUT "\n";
  close OUT;
}

if( !defined $op_out_no_stdout ) { # Unless the user wanted no STDOUT, print the
                           netlist to STDOUT
  print $netlist_title;
  print @spice;
  print "\n";
}
```

A.3 mkout_ufdg.pl

```
#!/usr/bin/perl -w
use strict;
use warnings;

my $Version_Date = 'Apr. 15, 2012';
my $Script_Name = 'mkout_ufdg.pl';

my $Fmt_Output = '.o';

my $Ezwave_Csv_Out_Format = '.csv';
my $Ezwave_Sti_Out_Format = '.sti';
my $Cscope_Txt_Out_Format = '.txt';

# Usage:
my $Usage = ''.
"Usage: $Script_Name  [OPTIONS]  ufdg_deck_name\n".
"Options: (by default, all output formats are generated)\n".
    "\t-c  Create .csv output for Mentor Graphics EZwave\n".
    "\t-s  Create .sti output for Mentor Graphics EZwave\n".
    "\t-x  Create .txt output for Synopsys CosmosScope\n";
my $Min_Arg_Num = 1;

# mkout_ufdg.pl
#
# Created by Mike Turi
# Washington State University
# School of Electrical Engineering & Computer Science
# High Performance Computer Systems (HiPerCopS) Group
#
# Note: Code is semi-tested; may fail under certain conditions
#
# This is a Perl script used to convert the output from the
# University of Flordia Double-Gate (UFDG) MOSFET model
# (e.g. FinFETs) to more usable formats.  The script converts
# the Spice3 output from the model to ".csv" and ".sti" files
# for use with Mentor Graphics EZWave and a ".txt" file for use
# with Synopsys CosmosScope.
#
# Usage: Provide the UFDG deck name, and the "deck_name.o"
# output file will be converted into "deck_name.csv",
# "deck_name.sti", and "deck_name.txt".
#
# Note: If the signal name is too long, UFDG will truncate the
# signal name when creating the ".o" output file--this will
# cause an error for this script.  It appears the maximum signal
# name size is 15 characters (including v() or #branch).

print "**$Script_Name version date: $Version_Date**\n";
die $Usage if scalar @ARGV == 0;
my $argnum = scalar @ARGV;
die "Error: Not enough parameters\n\n$Usage" if $argnum < $Min_Arg_Num;

my $i = 0;
my $arg = shift;
my $op_csv_ezwave;
my $op_sti_ezwave;
my $op_txt_cscope;
my $user_def_output;
while( $arg =~ m/^-/ ) {
  die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
  if( $arg =~ m/^-(\S+)/ ) { # Found options
    my $argmatch = $1;
    if( $argmatch =~ m/c/i ) {
      $op_csv_ezwave = 1; # Create .csv output for Mentor Graphics EZwave
      $user_def_output = 1;
    }
    if( $argmatch =~ m/s/i ) {
      $op_sti_ezwave = 1; # Create .sti output for Mentor Graphics EZwave
      $user_def_output = 1;
    }
    if( $argmatch =~ m/x/i ) {
      $op_txt_cscope = 1; # Create .txt output for Synopsys CosmosScope
      $user_def_output = 1;
    }
  }
  $arg = shift;
}
my $deckname = $arg;

# Don't generate output for filetypes that the user doesn't define (unless none are
                          defined, then generate all)
my $no_csv_ezwave;
if( !defined $op_csv_ezwave and defined $user_def_output ) {
  $no_csv_ezwave = 1;
}
my $no_sti_ezwave;
if( !defined $op_sti_ezwave and defined $user_def_output ) {
  $no_sti_ezwave = 1;
}
my $no_txt_cscope;
if( !defined $op_txt_cscope and defined $user_def_output ) {
  $no_txt_cscope = 1;
}

# --- Find the signal names and prepare for reading data ---
my @sig_names;
my @sig_hdrs;
my @fhandles;
$i = 0;
# Open the UFDG output file
my $outname = $deckname.$Fmt_Output;
open IN, "<$outname" or die 'Error: Cannot open ', $outname, ' due to: ', $!;
my $match;
while( <IN> ) {
  $match = 0;
  if( m/^Index/i ) { # A signal header is found
    foreach my $hdr (@sig_hdrs) {
      $match = 1 if $_ eq $hdr;
    }
    # Only extract signal names from unique signal headers (sig. headers repeat
                          throughout the file)
    if( $match == 0 ) {
      my @tmpsigs = split; # Tokenizes $_ (header) to get signals
      # Ignore "Index" (1st column) and only add "Time"/"Sweep"/etc. (2nd column) once
      push @sig_names, ((scalar @sig_hdrs == 0) ? splice @tmpsigs, 1 : splice
                          @tmpsigs, 2);
      push @sig_hdrs, $_; # Add this signal header to the list
      my $dashes = <IN>; # Grab line of dashes "----"
      # Open a file handle to begin reading from Index 0 of this data
      # Multiple file handles are used for multiple pieces of data in different
                          locations in the file
      open $fhandles[$i], "<$outname" or die 'Error: Cannot open ', $outname, ' due
                          to: ', $!;
      seek $fhandles[$i++], (tell IN), 0; # (tell IN) is the location of Index 0 for
                           this data
    }
  }
}
close IN;

die 'Error: No plotted signals found (is DC op point only present in ', $outname, '?)'
                           if scalar @sig_names == 0;

# --- Initialize the new output files with the signal names ---
my $cs_txt_name = $deckname.$Cscope_Txt_Out_Format;
open CSCOPE_TXT, ">$cs_txt_name" or die 'Error: Cannot create CosmosScope ',
                           $Cscope_Txt_Out_Format, ' output due to ', $! unless
                           defined $no_txt_cscope;
my $ez_csv_name = $deckname.$Ezwave_Csv_Out_Format;
open EZWAVE_CSV, ">$ez_csv_name" or die 'Error: Cannot create EZWave ',
                           $Ezwave_Csv_Out_Format, ' output due to ', $! unless
                           defined $no_csv_ezwave;
print EZWAVE_CSV ".HEADER\n..NAMES\n" unless defined $no_csv_ezwave;
my @ezwave_csv_names;
my @ezwave_sti_names;
my %sig_type; # The type of signal, voltage (V) or current (I), for a given signal
                           name (for EZWave .sti output)
my $sim_type;
$i = 0;
foreach (@sig_names) {
  if( $i++ != 0 ) { # Delimeter beteen signal names
    print CSCOPE_TXT '  ' unless defined $no_txt_cscope;
    print EZWAVE_CSV ','  unless defined $no_csv_ezwave;
  }
  if( m/^v\((\w+)\)$/i ) { # Found a voltage signal
    $sig_type{$1} = 'V';
    push @ezwave_sti_names, $1;
    print CSCOPE_TXT "$_`V"  unless defined $no_txt_cscope;
    print EZWAVE_CSV "V($1)" unless defined $no_csv_ezwave;
    push @ezwave_csv_names, "V($1)";
  } elsif( m/^(\w+)\#branch$/i ) { # Found a current signal
    $sig_type{$1} = 'I';
    push @ezwave_sti_names, $1;
    print CSCOPE_TXT "i($1)`A" unless defined $no_txt_cscope;
    print EZWAVE_CSV "I($1)"   unless defined $no_csv_ezwave;
    push @ezwave_csv_names, "I($1)";
  } elsif( m/^time$/i ) { # Found "time" => a TRAN analysis
    $sim_type = 'TRAN';
    print CSCOPE_TXT 't`s'  unless defined $no_txt_cscope;
    print EZWAVE_CSV 'Time' unless defined $no_csv_ezwave;
    push @ezwave_csv_names, 'Time';
  } elsif( m/^v-sweep$/i ) { # Found "v-sweep" => a DC analysis (ngspice)
    $sim_type = 'DC';
    print CSCOPE_TXT 'VOLTS`V' unless defined $no_txt_cscope;
    print EZWAVE_CSV 'Voltage' unless defined $no_csv_ezwave;
    push @ezwave_csv_names, 'Voltage';
  } elsif( m/^i-sweep$/i ) { # Found "i-sweep" => a DC analysis (ngspice)
    $sim_type = 'DC';
    print CSCOPE_TXT 'AMPS`A'  unless defined $no_txt_cscope;
    print EZWAVE_CSV 'Current' unless defined $no_csv_ezwave;
    push @ezwave_csv_names, 'Current';
  } elsif( m/^sweep$/i ) { # Found "sweep" => a DC analysis (spice3f5)
    $sim_type = 'DC';
    print CSCOPE_TXT 'Sweep'  unless defined $no_txt_cscope;
    print EZWAVE_CSV 'Sweep' unless defined $no_csv_ezwave;
    push @ezwave_csv_names, 'Sweep';
  } else {
    die 'Cannot understand input signal: ', $_, '(was the signal name too long and
                           truncated?)';
  }
}
print CSCOPE_TXT "\n" unless defined $no_txt_cscope;

# Output the plotted signal names
print 'Plotted signals for "', $deckname, '" are: ', (join ' ', @ezwave_csv_names),
                           "\n";

# More fomatting for EZWave follows:
if( !defined $no_csv_ezwave ) {
  print EZWAVE_CSV "\n..UNITS\n";
  $i = 0;
  foreach (@ezwave_csv_names) {
    if( $i++ != 0 ) { # EZWave .csv is comma deliminated
      print EZWAVE_CSV ',';
    }
    if( m/^V\(/ ) { # Voltage signal measured in Volts
      print EZWAVE_CSV 'Voltage(V)';
    } elsif( m/^I\(/ ) { # Current signal measured in Amps
      print EZWAVE_CSV 'Current(A)';
    } elsif( $_ eq 'Time' ) { # Time is measured in seconds
      print EZWAVE_CSV 's';
    } elsif( $_ eq 'Voltage' ) { # Voltage sweep meas. in V (ngspice)
      print EZWAVE_CSV 'V';
    } elsif( $_ eq 'Current' ) { # Current sweep meas. in A (ngspice)
      print EZWAVE_CSV 'A';
    } elsif( $_ eq 'Sweep' ) { # An old sweep (spice3f5); don't know units
      print EZWAVE_CSV ' ';
    } else {
      die 'Corrupted EZWave input signal for .csv output: ', $_;
    }
  }
  # All data types are doubles
  print EZWAVE_CSV "\n..DATATYPES\n";
  for( $i = 1; $i < scalar @ezwave_csv_names; $i++ ) {
    print EZWAVE_CSV 'double,';
  }
  print EZWAVE_CSV "double\n..WAVEFORM_TYPES\n"; # Also prints last 'double'
  $i = 0;
  foreach (@ezwave_csv_names) {
    if( $i++ != 0 ) {
      print EZWAVE_CSV ',';
    }
    if( m/^[IV]\(/ ) { # All voltage/current signals are 'analog' type
      print EZWAVE_CSV 'analog';
    }
  }
  print EZWAVE_CSV "\n..AXIS_SPACING\nlinear\n..FOLDER_NAME\n$sim_type\n.DATA\n";
}

# --- Print the numeric data to each new output file ---
my @xaxis_vals; # The x-axis (time, sweep, etc.) values for EZWave .sti output
my %sig_vals; # The space-delimited values for a given signal name (for EZWave .sti
                           output)
my $signum; # Keeps track of what piece of data goes with which signal for EZWave .sti
                           output
$i = 0; # The current "Index" value that the file descriptors are at in the .o file
my $fnum; # Keeps track of how many file descriptors have been looked at for each
                           "Index" value
while(1) {
  $fnum = 0;
  $signum = 0;
  foreach my $fh (@fhandles) {
    my $data_line;
    while( <$fh> ) { # Loop until data is found (bypasses duplicate signal headers)
      if( m/^\d/ ) { # Found an index value (always positive)
        $data_line = $_;
        last;
      }
    }
    goto THE_END unless defined $data_line;

    # Tokenize line to get each data value
    my @data = split ' ', $data_line;
    # If the index from the output file ($data[0]) is not equal
    #   to the index $i, then the end of the data has been reached.
    #   Unless the file is corrupt, $data[0] will equal 0 again when
    #   the next set of signals are reached, at this point, all data
    #   has been processed.
    goto THE_END if $data[0] != $i;
    if( $fnum++ == 0 ) { # Print "Time"/"Sweep"/etc. only once
      push @xaxis_vals, $data[1] unless defined $no_sti_ezwave;
      print CSCOPE_TXT $data[1] unless defined $no_txt_cscope;
      print EZWAVE_CSV $data[1] unless defined $no_csv_ezwave;
    }
    foreach (splice @data, 2) {
      my $signame = $ezwave_sti_names[$signum++];
      $sig_vals{$signame} = (exists $sig_vals{$signame} ? $sig_vals{$signame}.' '.$_ :
                           $_) unless $no_sti_ezwave;
      # Print the data value with appropriate delimiter
      print CSCOPE_TXT "  $_" unless defined $no_txt_cscope;
      print EZWAVE_CSV ",$_"  unless defined $no_csv_ezwave;
    }
  }
  print CSCOPE_TXT "\n" unless defined $no_txt_cscope;
  print EZWAVE_CSV "\n" unless defined $no_csv_ezwave;
  $i++;
}

# Close all files when done (we'll do EZWave .sti output next)
THE_END: foreach (@fhandles) {
  close $_;
}
close CSCOPE_TXT unless defined $no_txt_cscope;
close EZWAVE_CSV unless defined $no_csv_ezwave;

exit if defined $no_sti_ezwave;
# Aha, but there's more...print output for EZWave .sti output if necessary

my $ez_sti_name = $deckname.$Ezwave_Sti_Out_Format;
open EZWAVE_STI, ">$ez_sti_name" or die 'Error: Cannot create EZWave ',
                           $Ezwave_Sti_Out_Format, ' output due to ', $!;

foreach (@ezwave_sti_names) {
  print EZWAVE_STI "\n", $sig_type{$_}, $_, ' ', $_, " 0 PWL (\n"; # E.g., Vbit bit 0
                           PWL (
  if( !exists $sig_vals{$_} ) {
    close EZWAVE_STI;
    die 'Internal Error: Data values of "', $_, '" not captured for EZWave ',
                           $Ezwave_Sti_Out_Format, ' output';
  }
  my @data_vals = split ' ', $sig_vals{$_};
  if( scalar @xaxis_vals != scalar @data_vals ) {
    close EZWAVE_STI;
    die 'Internal Error: Number of x-axis values is not equal to the number of data
                         values of "', $_, '" for EZWave ', $Ezwave_Sti_Out_Format,
                         ' output';
  }
  for( $i = 0; $i < scalar @xaxis_vals; $i++ ) {
    print EZWAVE_STI '+ ', $xaxis_vals[$i], ' ', $data_vals[$i], "\n";
  }
  print EZWAVE_STI "+ )\n";
}

close EZWAVE_STI;
```

A.4 init_batch_ufdg.pl

```
#!/usr/bin/perl -w
use strict;
use warnings;

my $Version_Date = 'May. 4, 2012';
my $Script_Name = 'init_batch_ufdg.pl';

# Constants:
my $Fmt_Input = '.i';
my $Default_Std_Dev_Factor = 30; # std_dev = mean/30 (3*std_dev = mean/10)

my $Ufdg_3_7_Spice = 'ngspice3.ufdg-3.7'; # Name of UFDG Spice executable
my $CMU_QMC_Exec = 'gennorm'; # Name of Carnegie Mellon Quasi-Monte Carlo executable
my $Default_QMC_Seq = 'S'; # Default sequence-type for CMU QMC (Sobol sequence)

my $Netlist_Offset = 1; # The title (1st comment line) is on line #0, line 1 begins
                          code for the netlist

my $Sweep_Sum_Prefix = 'sweepsum_'; # The file prefix for the sweep summary file
my $Sweep_Sum_Filetype = '.csv'; # The filetype for the sweep summary file
my $Sweep_Table_Spacing = 15; # The printf spacing for the sweep summary table (for
                          display to stdout)

my $Qmc_Sum_Prefix = 'qmcsum_'; # The file prefix for the qmc summary file
my $Qmc_Sum_Filetype = '.csv'; # The filetype for the qmc summary file

# Requires script:
my $Netgen_Ufdg = 'netgen_ufdg.pl';

# Corner names: np (e.g. fs-->n=fast; p=slow)
my @Corner_Names = ( 'ff', 'fs', 'sf', 'ss' );
my $Corner_Count = scalar @Corner_Names;
# Mult. factors for n- and p-type FinFETs (1=+n std. dev; -1=-n std. dev)
my @Corner_N_Factors = ( 1, 1, -1, -1 );
my @Corner_P_Factors = ( 1, -1, 1, -1 );

# Usage:
my $Usage = ''.
'Usage: '.$Script_Name."  [OPTIONS]  ufdg_deck_name  init_batch_def_file  [#sims]\n".
"Options\n".
    "\t-c N  Run N number of corners (+/- N standard deviations, e.g. N=3)\n".
    "\tFor CMU's QMC point generation:\n".
    "\t-l    Use \"Latin hypercube sampling (LHS)\"\n".
    "\t-r    Use \"linear congruential pseudo-random number generator\"\n".
    "\t-s    Use \"Sobol sequence\" (default)\n";
```

```
my $Min_Arg_Num = 2;

# Procedure: expand_sweep_expression
# Summary:
#   Expands a sweep expression by returning (1st) a comma delimited string of values
#   represented by the sweep expression and returning (2nd) a count of how many
#   values are in the comma delimited string
# Parameter: 1st param = sweep expression to expand; 2nd param = descriptive name
#            to be used in error messages if expansion of sweep expression fails
sub expand_sweep_expression {
  my $sw_expression = shift;
  my $sw_errorname = shift;

  my $sw_vals; # 1st return value, a comma delimited list of the sweep values
                           represented by the expression
  my $sw_vals_cnt = 1; # 2nd return value, the number (count) of sweep values in the
                           comma delimited list
  if( $sw_expression =~ m/^\s*=\s*(\S+\s*,.+)/ ) { # E.g. 5e-9, ...
    $sw_vals = $1; # Sweep values are already provided as a comma delimited list
    chomp $sw_vals;
    my @split_vals = split /[ ,]+/, $sw_vals;
    $sw_vals_cnt = scalar @split_vals;
  } elsif( $sw_expression =~ m/^\s*=\s*(\S+)\s*:\s*(\S+)\s*:\s*(\S+)/i ) { # E.g. 2e-
                           9:0.5e-9:4e-9 (equals 2e-9, 2.5e-9, 3e-9. 3.5e-9, 4e-9)
    my $s_beg = $1;
    my $s_end = $3;
    my $s_inc = $2; # Note: Increment can be positive or negative (for loop
                           compensates for this) [E.g. 4e-9:-0.5e-9:2e-9 (equals 4e-9,
                           3.5e-9, 3e-9. 2.5e-9, 2e-9)]
    $sw_vals = $s_beg;
    die 'Error: Increment value of 0 found for sweeping ', $sw_errorname if $s_inc ==
                           0;
    die 'Error: Illegal start:increment:finish for sweeping ', $sw_errorname if
                           (($s_beg < $s_end and $s_inc < 0) or ($s_beg > $s_end and
                           $s_inc > 0));
    # Use start:increment:finish to make comma delimited list of values
    for( my $j = $s_beg+$s_inc; ($j <= $s_end and $s_inc > 0) or ($j >= $s_end and
                           $s_inc < 0); $j+=$s_inc ) {
      $sw_vals = $sw_vals.','.$j;
      $sw_vals_cnt++;
    }
  } elsif( $sw_expression =~
                           m/^\s*=\s*(\S+)\s*;\s*(\S+)\s*%\s*:\s*(\S+)\s*%\s*:\s*(\S+)
                           \s*%/ ) { # E.g. 0.2; -10%:5%:10% (equals 0.18, 0.19, 0.2,
                           0.21, 0.22)
    my $s_ave = $1;
    my $s_beg = $2/100;
    my $s_end = $4/100;
    my $s_inc = $3/100; # Note: Increment can be positive or negative (for loop
                           compensates for this) [E.g. 0.2; 10%:-5%:-10% (equals 0.22,
                           0.21, 0.2, 0.19, 0.18)]
    $sw_vals = $s_ave + $s_ave * $s_beg;
    die 'Error: Increment value of 0 found for sweeping ', $sw_errorname if $s_inc ==
                           0;
    die 'Error: Illegal start:increment:finish for sweeping ', $sw_errorname if
                           (($s_beg < $s_end and $s_inc < 0) or ($s_beg > $s_end and
                           $s_inc > 0));
    # Use start:increment:finish to make comma delimited list of values
    for( my $j = $s_beg+$s_inc; ($j <= $s_end and $s_inc > 0) or ($j >= $s_end and
                           $s_inc < 0); $j+=$s_inc ) {
      $sw_vals = $sw_vals.','.($s_ave + $s_ave * $j);
      $sw_vals_cnt++;
    }
  } else {
    die 'Error: Could not understand sweeping "', $sw_expression, '" for ',
                           $sw_errorname;
  }
  die 'Error: Zero sweep values found in "', $sw_expression, '" for ', $sw_errorname
                           unless $sw_vals_cnt > 0;
  return ($sw_vals, $sw_vals_cnt);
}

# init_batch_ufdg.pl
#
# Created by Mike Turi
# Washington State University
# School of Electrical Engineering & Computer Science
# High Performance Computer Systems (HiPerCopS) Group
#
# Note: Code is semi-tested; may fail under certain conditions
#
# This is by far the most complex of this suite of scripts.
# This is a Perl script used to initialize a directory of
# spice netlists with varied (e.g. for Monte Carlo simulations) or
# sweeped transistor parameters, supply voltages, or options (such
# as temperature).  Afterwards, batchexec_ufdg.pl can be used to
# run all files and a measurement script can be used to summarize
# results of interest.  This script also supports corner simulations.
#
# Usage: Provide the UFDG deck name and the filename of the
#        init_batch_def_file (the syntax of this is described in the
#        README-pvt_variations.txt file).  The output from the script
#        will summarize any varied or swept parameters or variables in
#        the netlist.  Also, a summary table for swept params/vars will
#        be output and/or a summary listing of varied params/vars will
#        be output.  Note, this script requires that the path to the
#        Carnegie Mellon University quasi-Monte Carlo executable
#        (CMU QMC) be added to the PATH variable prior to running.  For
#        more information, refer to the README-pvt_variations.txt file.

print "**$Script_Name version date: $Version_Date**\n";
die $Usage if scalar @ARGV < 1;
my $argnum = scalar @ARGV;
die "Error: Not enough parameters\n\n$Usage" if $argnum < $Min_Arg_Num;

my $i = 0;
my $arg = shift;
my $op_corners;
my $qmc_seq = $Default_QMC_Seq; # The sequence type to use for CMU's QMC
while( $arg =~ m/^-/ ) {
  die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
  if( $arg =~ m/^-(\S+)/ ) { # Found options
     my $argmatch = $1;

    # Check single letter options first
    if( $argmatch =~ m/([lrs])/i ) {
      $qmc_seq = uc $1; # Method to use for CMU's QMC point generation
    }

    # Check options which require another argument
    if( $argmatch =~ m/c/i ) {
      $op_corners = shift; # Set to number of standard deviations to use
      die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
    }
  }
  $arg = shift;
```

```
}
my $deckname = $arg;
my $init_defs_file = shift;
my $sim_num = defined $op_corners ? $Corner_Count : shift;

print 'Note: The default std_dev = mean/', $Default_Std_Dev_Factor, ' (3*std_dev =
                           mean/', $Default_Std_Dev_Factor/3, ")\n\n";

##########
my @sweep_titles; # Holds the names/titles (defined by the user in the
                           init_batch_def_file) for the directories to be created for
                           a swept netlist

my %sweep_level_to_names; # Holds the model:param (or variable:param) combination(s)
                           for a given sweep level number (or "default")
my %sweep_name_to_level; # Holds the sweep level number (or "default") for a given
                           model:param (or variable:param) combination
my %sweep_level_steps; # Holds the number of steps for a given sweep level number (or
                           "default")

my %sweep_params; # Holds the names of swept transistor parameters (as a space
                           delimited string) for a given model name
my %sw_param_vals; # Holds the sweep values (comma delimited) for a given model:param
                           combination

my @sweep_vars; # Holds the names of swept .param variables
my %sw_var_vals; # Holds the sweep values (comma delimited) for a given .param
                           variable name

my %sweep_name_to_val_list; # Holds the sweep values (comma delimited) for a given
                           model:param (or variable:param) combination
                             # This is a combination of the %sw_param_vals and
                           %sw_var_vals hashes

my @vary_vars; # Holds the names of varied .param variables
my %vy_var_means; # Holds the mean for a given .param variable name
my %vy_var_stddevs; # Holds the standard deviation for a given .param variable name

my %vary_params; # Holds the names of varied transistor parameters (as a space
                           delimited string) for a given model name
my %vy_param_means; # Holds the mean for a given model:param combination
my %vy_param_stddevs; # Holds the standard deviation for a given model:param
                           combination
##########

# Open the init batch definitions file and parse for .param and model variables to
                           vary and their means and standard deviations
open DEFS, "<$init_defs_file" or die 'Error: Cannot open ', $init_defs_file, ' due to:
                           ', $!;
$i = 0;
while( <DEFS> ) {
  $i++;
  if( m/^\s*VARY\s+/i ) { # Found start of a VARY block
    VARY_LOOP: while( <DEFS> ) {
      $i++;
      if( m/^\s*END\s+VARY\s+/i ) { last; } # Found end of a VARY block
      if( m/^\s*(\w+)\.(\w+)(.*)/i ) { # Found a parameter or model
        my $type = lc $1; # param or model
        my $name = lc $2; # param_name or model_name
        my $rest = $3;

        if( $type eq 'param' or $type eq 'params' ) { # Place in ".param variable"
                           vary variables
          push @vary_vars, $name; # Add to list of .param variables to vary
          if( $rest =~ m/^\s*=\s*(\S+)\s*;\s*stddev\s*=\s*(\S+)/i ) {
            $vy_var_means{$name} = $1; # Use provided mean
            $vy_var_stddevs{$name} = $2; # Use provided standard deviation
          } elsif( $rest =~ m/^\s*=\s*(\S+)/ ) {
            $vy_var_means{$name} = $1; # Use provided mean
            $vy_var_stddevs{$name} = 'default'; # Use defualt standard deviation
          } else {
            $vy_var_means{$name} = 'file'; # Use value from file as mean
            $vy_var_stddevs{$name} = 'file'; # Use value from file as standard
                          deviation
          }
        } elsif( $type eq 'model' or $type eq 'models' ) { # Place in "param" vary
                          variables for variations in model parameters
          while( <DEFS> ) {
            $i++;
            if( m/^\s*END\s+VARY\s+/i ) { last VARY_LOOP; } # Found end of a VARY
                          block
            if( m/^\s*\w+\.\w+/ ) {  # Found new model, option, or voltage
              # "Put" this back in <DEFS> stream, so it can be properly refound
              seek DEFS, -1*length, 1;
              last;
            }
            if( m/^\s*\+?\s*(\w+)(.*)/ ) {
              my $param = lc $1;
              my $rest = $2;
              # Add to list of this model's params to vary
              $vary_params{$name} = exists $vary_params{$name} ? $vary_params{$name}.'
                          '.$param : $param;
              if( $rest =~ m/^\s*=\s*(\S+)\s*;\s*stddev\s*=\s*(\S+)/i ) {
                $vy_param_means{$name.':'.$param} = $1; # Use provided mean
                $vy_param_stddevs{$name.':'.$param} = $2; # Use provided standard
                          deviation
              } elsif( $rest =~ m/^\s*=\s*(\S+)/ ) {
                $vy_param_means{$name.':'.$param} = $1; # Use provided mean
                $vy_param_stddevs{$name.':'.$param} = 'default'; # Use default
                          standard deviation
              } else {
                $vy_param_means{$name.':'.$param} = 'file'; # Use value from file as
                          mean
                $vy_param_stddevs{$name.':'.$param} = 'file'; # Use value from file as
                          standard deviation
              }
            }
          }
        } else {
          print 'Warning: Type "', $type, '" from ', $init_defs_file, " was not
                          understood and was ignored\n\n";
        }
      }
    }
  } elsif( m/^\s*SWEEP\s+/i ) { # Found start of a SWEEP block
    if( m/^\s*SWEEP\s+(NAME|TITLE)S?\s+(.*)/i ) { # Actually just found the titles for
                          the swept simulation
      push @sweep_titles, split /[ ,]+/, $2;
      next;
    }
    my $level = 'default';
    $level = $1 if m/^\s*SWEEP\s+LEVEL\s*=\s*(\d+)/; # Grab the sweep level if it is
                          defined, else just use "default"
    my $lvl_steps;
    SWEEP_LOOP: while( <DEFS> ) {
      $i++;
```

```
if( m/^\s*END\s+SWEEP\s+/i ) { last; } # Found end of a SWEEP block
if( m/^\s*(\w+)\.(\w+)(.*)/i ) { # Found a parameter or model
  my $type = lc $1; # param or model
  my $name = lc $2; # param_name or model_name
  my $rest = $3;

  if( $type eq 'param' or $type eq 'params' ) { # Place in ".param variable"
                     sweep variables
    # Note: It would be nice to both sweep and vary a parameter (e.g. vbias is
                     swept to 0.4, 0.2, and 0V and then
    #   Monte Carlo simulations vary each vbias level: 0.4V +/- x%, and 0.2V +/-
                     x%, and 0V +/- x% groups of sims)
    die 'Error: Cannot both sweep and vary .param variable: ', $name if exists
                     $vy_var_means{$name};

    push @sweep_vars, $name; # Add to list of .param variables to sweep
    ($sw_var_vals{$name}, $lvl_steps) = expand_sweep_expression $rest, '.param
                     variable: '.$name; # Get the sweep values (expand from def.
                     file)
    $sweep_name_to_val_list{'variable'.':'.$name} = $sw_var_vals{$name};

    die 'Error: Number of sweep steps do not agree (',
                     $sweep_level_steps{$level}, ' and ', $lvl_steps, ') for
                     sweep level ', $level
      if (exists $sweep_level_steps{$level} and $lvl_steps !=
                     $sweep_level_steps{$level});
    $sweep_level_steps{$level} = $lvl_steps; # Set the number of steps for this
                     sweep level
    # Add this .param variable to the list of variables/parameters swept at this
                     sweep level
    $sweep_level_to_names{$level} = exists $sweep_level_to_names{$level} ?
                     $sweep_level_to_names{$level}.' '.'variable'.':'.$name :
                     'variable'.':'.$name;
    $sweep_name_to_level{'variable:'.$name} = $level;

  } elsif( $type eq 'model' or $type eq 'models' ) { # Place in "param" sweep
                     variables for sweeping model parameters
    while( <DEFS> ) {
      $i++;
      if( m/^\s*END\s+SWEEP\s+/i ) { last SWEEP_LOOP; } # Found end of a SWEEP
                     block
      if( m/^\s*\w+\.\w+/i ) {  # Found new model, option, or voltage
        # "Put" this back in <DEFS> stream, so it can be properly refound
        seek DEFS, -1*length, 1;
        last;
      }
      if( m/^\s*\+?\s*(\w+)(.*)/i ) {
        my $param = lc $1;
        my $rest = $2;

        # Note: It would be nice to both sweep and vary a parameter (e.g. vbias
                     is swept to 0.4, 0.2, and 0V and then
        #   Monte Carlo simulations vary each vbias level: 0.4V +/- x%, and 0.2V
                     +/- x%, and 0V +/- x% groups of sims)
        die 'Error: Cannot both sweep and vary parameter: ', $name, ':', $param
                     if exists $vy_param_means{$name.':'.$param};

        $sweep_params{$name} = exists $sweep_params{$name} ?
                     $sweep_params{$name}.' '.$param : $param; # Add to list of
                     this model's params to sweep
        ($sw_param_vals{$name.':'.$param}, $lvl_steps) = expand_sweep_expression
                     $rest, 'model '.$name.'\'s parameter '.$param; # Get the
                     sweep values (expand from def. file)
```

```
              $sweep_name_to_val_list{$name.':'.$param} =
                           $sw_param_vals{$name.':'.$param};

              die 'Error: Number of sweep steps do not agree (',
                           $sweep_level_steps{$level}, ' and ', $lvl_steps, ') for
                           sweep level ', $level
                if (exists $sweep_level_steps{$level} and $lvl_steps !=
                           $sweep_level_steps{$level});
              $sweep_level_steps{$level} = $lvl_steps; # Set the number of steps for
                           this sweep level
              # Add this model:param to the list of variables/parameters swept at this
                           sweep level
              $sweep_level_to_names{$level} = exists $sweep_level_to_names{$level} ?
                           $sweep_level_to_names{$level}.' '.$name.':'.$param :
                           $name.':'.$param;
              $sweep_name_to_level{$name.':'.$param} = $level;
            }
          }
        } else {
          print 'Warning: Type "', $type, '" from ', $init_defs_file, " was not
                           understood and was ignored\n\n";
        }
      }
    }
  }
}
close DEFS;

die 'Error: Sweep block missing level (if using levels of hierarchy, all sweep blocks
                           must have a level value)'
  if (exists $sweep_level_to_names{'default'} and scalar (keys %sweep_level_to_names)
                           > 1);

# Create the directory for the varied simulations to go (remove prior existing
                           directory if neccessary)
my $deckfile = $deckname.$Fmt_Input;
die 'Error: "', $deckfile, '" does not exist' unless -e $deckfile;
my $dirname = './batch_'.$deckname.'/';
my $dirtext = 'batch_'.$deckname;
system( ('rm', '-rf', $dirname) ) == 0 or die 'Error: "rm -rf ', $dirname, '" somehow
                           failed';
mkdir $dirname or die 'Error: Cannot "mkdir ', $dirname, '" due to: ', $!;

# Call netgen_ufdg.pl to remove comments and insert include files
#  If .param variables will be varied, then netgen_ufdg.pl will retrieve parameter
                           details and substitute parameters with placeholders
#  If .param variables will not be varied, then netgen_ufdg.pl will substitute the
                           proper values for the parameters
my $deckfile0 = $dirname.'0'.$Fmt_Input;
# @spice will hold the netlist (modified by netgen_ufdg.pl)
my @spice = (scalar @vary_vars > 0 or scalar @sweep_vars > 0) ? `$Netgen_Ufdg -zdi
                           $deckfile -o $deckfile0` : `$Netgen_Ufdg -zi $deckfile -o
                           $deckfile0`;
die 'Error: "', $Netgen_Ufdg, '" script failed, aborting initialization' unless $? ==
                           0;
print "\n";

##########
my %spice_params; # Holds the value (from the netlist) for a given .param variable
                           name
my @spice_params_ordered; # An array of the .param variable names (in the order
                           declared in the netlist); use instead of "keys
                           %spice_params" to maintain order
```

```
                          # Must maintain order because there can be .param variable
                           dependencies that must be resolved in proper order (e.g.
                           .param a = {10*b})
my %spice_placeholders; # Holds the expression (of one or more .param variables) for a
                           given placeholder (generated by netgen_ufdg.pl)
##########

if( scalar @vary_vars > 0 or scalar @sweep_vars > 0 ) { # If .param variables are
                           swept or varied, must deal with the placeholders that
                           netgen_ufdg.pl inserted

  my $detail = shift @spice; # Using shift since .param variable and placeholder
                           information is not part of the netlist and should be
                           removed
  chomp $detail;
  while( $detail =~ m/^([a-z]\w*)=(.+)$/i ) { # First set of lines from netgen_ufdg.pl
                           contain: param1=value1
    $spice_params{lc $1} = $2;
    push @spice_params_ordered, lc $1; # Place this .param variable name in the order
                           declared in the netlist
    $detail = shift @spice;
    chomp $detail;
  }
  while( $detail =~ m/^(\d\S*)=(.+)$/i ) { # Second set of lines from netgen_ufdg.pl
                           contain: placeholder1=param_exp1 (e.g. 1000000T=100p*simlen
                           or 1000001T=h)
    $spice_placeholders{$1} = $2;
    $detail = shift @spice;
    chomp $detail;
  }
  while( (substr $detail, 0, 1) !~ m/^\*/ ) { # Check to see if the title of the spice
                           netlist is next (after the placeholder info from
                           netgen_ufdg.pl)
    print 'Internal Warning: "', $detail, "\" was expected to be title of spice
                           netlist; It will be ignored\n";
    $detail = shift @spice;
    chomp $detail;
  }
  unshift @spice, $detail."\n"; # Put back the title of the spice netlist so that
                           @spice contains the entire netlist

  my @rm_vars;
  foreach my $pvar (@vary_vars, @sweep_vars) {
    my $usage = 0; # Flag to check if each .param variable swept or varied is actually
                           used in the netlist
    foreach (grep /^\{.*\}$/, values %spice_params) { # Checking if .param variable is
                           used in the definition of another .param variable (e.g.
                           .param a = {10*b})
      if( m/^\{$pvar\}$/i or m/\W$pvar\W/i ) {
        $usage = 1;
        last;
      }
    }
    if( $usage < 1 ) {
      foreach (values %spice_placeholders) { # Checking if .param variable is used in
                           the netlist by looking through the placeholders
        if( m/^$pvar$/i or m/\W$pvar\W/i ) {
          $usage = 1;
          last;
        }
      }
    }
    if( $usage < 1 ) { # .param variable is not used in the netlist
```

```
      print 'Warning: .param "', $pvar, "\" in init batch def. file was not found in
                           netlist, so it will not be swept or varied\n";
      push @rm_vars, $pvar; # Add this .param variable name to the list for deletion

    } else { # Check if .param variable was declared in the netlist (print a warning
                           if was not, but can continue with the sweep or vary)
      my $declared = 0;
      foreach (keys %spice_params) {
        if( $_ eq $pvar ) {
          $declared = 1;
          last;
        }
      }
      if( $declared < 1 ) {
        print 'Warning: .param "', $pvar, "\" in init batch def. file not declared in
                           netlist, but will still be swept or varied\n";
        $spice_params{$pvar} = 0;
      }
    }
  }

  foreach (@rm_vars) { # Some .param variable names are to be removed (if scalar
                           @rm_vars > 0) since they are unused in the netlist
    for( $i = 0; $i < scalar @vary_vars; $i++ ) {
      if( $_ eq $vary_vars[$i] ) {
        splice @vary_vars, $i, 1;
        last;
      }
    }
    for( $i = 0; $i < scalar @sweep_vars; $i++ ) {
      if( $_ eq $sweep_vars[$i] ) {
        splice @sweep_vars, $i, 1;
        last;
      }
    }
  }
}


# Grab the option, print, and save lines from the original netlist since
# ngspice's "listing expand" omits these in its output
# If a line is not present (e.g. no .SAVE line), then grep returns an empty string ''
my @grep_lines = grep /^\.option/i, @spice; # Format .OPTION ABSTOL=1e-6 ...
my $option_line = scalar @grep_lines < 1 ? "" : $grep_lines[0];
chomp $option_line;
@grep_lines = grep /^\.print/i, @spice; # Format .PRINT TRAN v(foo) ...
my $print_line = scalar @grep_lines < 1 ? "" : $grep_lines[0];
chomp $print_line;
@grep_lines = grep /^\.save/i, @spice; # Format .SAVE v(foo) ...
my $save_line = scalar @grep_lines < 1 ? "" : $grep_lines[0];
chomp $save_line;
print "\nOption line: ", $option_line, "\n";
print 'Print line:  ', $print_line, "\n";
print 'Save line:   ', $save_line, "\n";

# Use ngspice to expand the netlist (expand subcircuits prior to duplicating models
                           for varied transistors within subcircuit)
# Save the resultant netlist as 0.i in the simulation directory (will be overwritten
                           later)
system( "echo \"listing expand > $deckfile0\nquit\" | $Ufdg_3_7_Spice -i $deckfile0" )
                           == 0
  or die 'Error: Listing expansion through ', $Ufdg_3_7_Spice, ' failed';
```

```
# Open the resultant netlist
open IN, "<$deckfile0" or die 'Error: Cannot open ', $deckfile0, ' due to: ', $!;
@spice = <IN>;
close IN;

# Remove spacing before * for first line's comment
$spice[0] =~ s/^\s*\*/\*/;
# Remove silly line numbers from "listing expand" operation (e.g. "   17 : minvp...")
foreach (@spice) {
  s/^\s*\d+\s*:\s*//;
}

##########
my %model_linenum; # Holds the line number of the declaration for a given model
my %model_fettype; # Holds the fet type (nmos or pmos) for a given model
my %model_inline_param; # Holds if the model:param combination declared inline (with
                           each instance of a transistor, e.g. l & w) for a given
                           model:param
##########

# Look through the netlist for the declarations of models with varied parameters
for( $i = 0; $spice[$i] !~ m/^.end/i and $i < scalar @spice; $i++ ) {
  my $model;
  my $param;
  if( $spice[$i] =~ m/^\.MODEL\s+([\w.]+)\s+([NP]MOS)/i ) { # Format .MODEL NSG NMOS
                           LEVEL=17
    $model = lc $1;
    $model_fettype{$model} = lc $2;
    $model_linenum{$model} = $i;
    if( exists $vary_params{$model} ) { # There are varied parameters for this model
      foreach (split ' ', $vary_params{$model}) {
        if( $spice[$i] !~ m/\s+$_\s*=\s*\S+\s*/i ) { # Is each parameter defined in
                           the model declaration?
          # This model:param combination must be declared inline
          $model_inline_param{$model.':'.$_} = 1; # Or could maybe use:
                           $model_inline_param{$model} = (exists
                           $model_inline_param{$model}) ?
                           $model_inline_param{$model}.' '.$_ : $_;
        }
      }
    }
    if( exists $sweep_params{$model} ) { # There are swept parameters for this model
      foreach (split ' ', $sweep_params{$model}) {
        if( $spice[$i] !~ m/\s+$_\s*=\s*\S+\s*/i ) { # Is each parameter defined in
                           the model declaration?
          # This model:param combination must be declared inline
          $model_inline_param{$model.':'.$_} = 1; # Or could maybe use:
                           $model_inline_param{$model} = (exists
                           $model_inline_param{$model}) ?
                           $model_inline_param{$model}.' '.$_ : $_;
        }
      }
    }
  }
}
# Overwrite the ".END" in the netlist (will be appending to file)
$spice[$i] = "\n";
# Append the .OPTION line
push @spice, $option_line."\n" unless length $option_line == 0;
# Append the .PRINT line
push @spice, $print_line."\n" unless length $print_line == 0;
# Append the .SAVE line
push @spice, $save_line."\n" unless length $save_line == 0;
push @spice, "\n";

##########
my %data_column; # Holds the column number for a given model:param (or variable:param)
                           combination
my $col = 0; # Column count for the number of data columns required for this netlist
##########

print "\nSummary of Spice Deck:\n";

print "For .param variables\n";
print "  Sweep:\n";
print "    No .param variables to sweep\n" if scalar @sweep_vars < 1;
foreach (sort @sweep_vars) { # If .param variables are swept
  my $sw_lvl = $sweep_name_to_level{'variable'.':'.$_};
  print '    ', $_, ': ', $sw_var_vals{$_},
        ";\n      (level = ", $sw_lvl, '; # steps = ', $sweep_level_steps{$sw_lvl},
                           ")\n";
  # Print the .param variable name, its values (comma delimited), the sweep level, and
                           how many sweep level steps
}
print "  Vary:\n";
print "    No .param variables to vary\n" if scalar @vary_vars < 1;
foreach (sort @vary_vars) { # If .param variables are varied
  print "    Warning: There is no support to vary .param variables for corner
                           simulations\n",
        "             (.param values will remain unchanged from the netlist)\n" if
                           defined $op_corners;
  last if defined $op_corners;

  # Count this variable in the number of required data columns
  $data_column{'variable:'.$_} = $col++; # Use "variable" as the fetname (it's ok,
                           fetnames begin with m)

  # Print variable summary (its name, mean, and standard deviation)
  print '    ', $_, ': mean=', $vy_var_means{$_}, '; std_dev=', $vy_var_stddevs{$_},
                           "\n";
}

# Print model parameter summary
foreach (sort keys %model_linenum) {
  my $model = $_;
  print 'For ', $model_fettype{$model}, ' model "', $model, "\"\n";
  print "  Sweep:\n";
  if( exists $sweep_params{$model} ) {
    foreach (split ' ', $sweep_params{$model}) {
      my $mpkey = $model.':'.$_;
      my $sw_lvl = $sweep_name_to_level{$mpkey};
      print '    ', $_, ': ', $sw_param_vals{$mpkey}, ';   inst. ', exists
                           $model_inline_param{$mpkey} ? 'inline' : 'in model',
            "\n      (level = ", $sw_lvl, '; # steps = ', $sweep_level_steps{$sw_lvl},
                           ")\n";
      # Print the parameter name, its values (comma delimited), where it was
                           instantiated, the sweep level, and how many sweep level
                           steps
    }
  } else {
    print "    No parameters to sweep\n";
  }
  print "  Vary:\n";
  if( exists $vary_params{$model} ) {
    foreach (split ' ', $vary_params{$model}) {
      my $mpkey = $model.':'.$_;
      print '    ', $_, ': mean=', $vy_param_means{$mpkey}, '; std_dev=',
                           $vy_param_stddevs{$mpkey},
            ';   inst. ', exists $model_inline_param{$mpkey} ? 'inline' : 'in model',
                           "\n";
      # Print the parameter name, its mean, its standard deviation, and where it was
                           instantiated
    }
  } else {
    print "    No parameters to vary\n";
  }
}

##########
my %sweep_vals; # A space delimited list of all values to be used for a given
                           model:param name or variable:param name (for .param
                           variables)
                # Note, this will include duplicate values for a hierarchical sweep
                           (all values for all sweep combinations are included)
my $sw_num_combos = 1; # The number of sweep combinations (also equal to the number of
                           directories required for sweep)
                       # This is equal to the number of sweep steps for a single sweep
                           level defined, but it is the
                       # product of all sweep steps for a hierarchical sweep (multiple
                           sweep levels)
##########

# This is a bit complicated to find all of the sweep levels.  Here is an example and
                           how the code modifies it:
#   Sweep level = 1: "vdd_val" has values (0.9, 1.0, 1.1)
#   Sweep level = 2: "l" has values (17e-9, 30e-9) and "h" has values (30e-9, 75e-9)

if( scalar (keys %sweep_level_steps) > 0 ) { # If sweeping values (at least one sweep
                           level--even "default"), must figure out sweep combinations
  my @sw_levels = keys %sweep_level_steps;
  @sw_levels = sort {$b <=> $a} @sw_levels unless exists
                           $sweep_level_steps{'default'}; # Sort numerically
                           descending

  foreach my $name (split ' ', $sweep_level_to_names{$sw_levels[0]}) { # Get
                           variable:param and model:param names for the lowest level
                           sweep
    $sweep_vals{$name} = join ' ', (split /[ ,]+/, $sweep_name_to_val_list{$name}); #
                           Get the values for each param name and space delimit them
  }
  my $num_copies = $sweep_level_steps{$sw_levels[0]}; # The number of copies higher
                           sweep levels will need of each value

  # Example: %sweep_vals now has:
  #   $sweep_vals{"vdd_val"} = undefined (no key "vdd_val" yet)
  #   $sweep_vals{"l"} = "17e-9 30e-9"
  #   $sweep_vals{"h"} = "30e-9 75e-9"

  for( $i = 1; $i < scalar @sw_levels; $i++ ) { # Go to the next lowest level sweep
                           (go one sweep level up each iteration)
    foreach my $name (split ' ', $sweep_level_to_names{$sw_levels[$i]}) { # Get the
                           variable:param and model:param names for this sweep level
      $sweep_vals{$name} = '';
      foreach (split /[ ,]+/, $sweep_name_to_val_list{$name}) { # Get the values for
                           each param name
        $sweep_vals{$name} .= ($_.' ') x $num_copies; # Must make multiple copies of
                           each value in order to correctly pair with lower sweep
                           level values
      }
    }

    # Example: %sweep_vals now has:
    #   $sweep_vals{"vdd_val"} = "0.9 0.9 1.0 1.0 1.1 1.1"
    #   $sweep_vals{"l"} = "17e-9 30e-9"
    #   $sweep_vals{"h"} = "30e-9 75e-9"

    for( my $j = 0; $j < $i; $j++ ) { # Go to the lower sweep levels again
      foreach my $name (split ' ', $sweep_level_to_names{$sw_levels[$j]}) { # Get the
                           variable:param and model:param names for this lower sweep
                           level
        # Must make multiple copies of each set of values in order to correctly pair
                           with the lower sweep level values
        $sweep_vals{$name} = ($sweep_vals{$name}.' ') x
                           $sweep_level_steps{$sw_levels[$i]};
      }
    }

    # Example: %sweep_vals now has:
    #   $sweep_vals{"vdd_val"} = " 0.9   0.9   1.0   1.0   1.1   1.1"
    #   $sweep_vals{"l"} =       "17e-9 30e-9 17e-9 30e-9 17e-9 30e-9"
    #   $sweep_vals{"h"} =       "30e-9 75e-9 30e-9 75e-9 30e-9 75e-9"
    #
    # Repeat for higher orders of sweep hierarchy

    $num_copies *= $sweep_level_steps{$sw_levels[$i]}; # Update the number of copies
                           required for the next higher sweep level
  }

  foreach (values %sweep_level_steps) { # Obtain number of sweep combinations
    $sw_num_combos *= $_;
  }

  # Print the sweep summary to file and as a table to stdout
  open SWEEP_SUM, ">$Sweep_Sum_Prefix$dirtext$Sweep_Sum_Filetype"
       or die "Error: Cannot create $Sweep_Sum_Prefix$dirtext$Sweep_Sum_Filetype for
                           output due to ", $!;

  print "\nSummary of Sweep:\n";
  foreach (('Name', sort keys %sweep_vals)) { # Print the model:param and
                           variable:param names
    printf '%*s  ', $Sweep_Table_Spacing, $_;
    print SWEEP_SUM $_, ',';
  }
  print "\n";
  print SWEEP_SUM "\n";
  my $dash = (('-' x $Sweep_Table_Spacing).'  ');
  print $dash x (1 + scalar keys %sweep_vals); # Make some dashes for clarity
  print "\n";
  for( $i = 0; $i < $sw_num_combos; $i++ ) {
    if( scalar @sweep_titles > $i ) { # Print the sweep title (if there is one) in the
                           first column
      printf '%*s  ', $Sweep_Table_Spacing, $sweep_titles[$i];
      print SWEEP_SUM $sweep_titles[$i], ',';
      $sweep_titles[$i] .= '/' unless $sweep_titles[$i] =~ m/\/$/;
    } else { # Use a number if there are not enough sweep titles for the sweep
                           combinations
      printf '%*d  ', $Sweep_Table_Spacing, $i+1;
      print SWEEP_SUM $i+1, ',';
      $sweep_titles[$i] = sprintf '%d/', $i+1;
    }
```

```
    mkdir $dirname.$sweep_titles[$i] or die 'Error: Cannot "mkdir ',
                           $dirname.$sweep_titles[$i], '" due to: ', $!; # Create
                           directory from the sweep title
    foreach (sort keys %sweep_vals) { # Print the correct value (for the sweep name)
                           for each model:param and variable:param name
      my @vals = split ' ', $sweep_vals{$_};
      print ' ' x ($Sweep_Table_Spacing-(length $vals[$i])); # Place spacing for
                           stdout table
      print $vals[$i], '  ';
      print SWEEP_SUM $vals[$i], ',';
    }
    print "\n";
    print SWEEP_SUM "\n";
  }

  close SWEEP_SUM;

  print "\nWarning: There are ", scalar @sweep_titles, ' sweep titles for ',
                           $sw_num_combos, ' sweep combinations; ',
        "Sweep simulations may not be named as anticipated\n" unless scalar
                           @sweep_titles == $sw_num_combos;
}

##########
my %var_linenum; # Holds the line number for the declaration of a given .param
                           variable
##########

if( scalar @vary_vars > 0 or scalar @sweep_vars > 0 ) { # If .param variables are
                           varied or swept, place their declaration back into the
                           netlist
  my @param_def_code; # Will become all of the .param declarations
  foreach my $var (@spice_params_ordered) {
    if( exists $vy_var_means{$var} ) { # Place varied .param variable declaration
                           (defined as its mean) and keep track of its line number
      $var_linenum{$var} = $Netlist_Offset + scalar @param_def_code;
      push @param_def_code, '.param '.$var.' = '.$vy_var_means{$var}."\n";
    } elsif( exists $sw_var_vals{$var} ) { # Place swept .param variable declaration
                           (defined as its 1st swept value) and keep track of its line
                           number
      $var_linenum{$var} = $Netlist_Offset + scalar @param_def_code;
      my @all_vals = split /[ ,]/, $sw_var_vals{$var};
      push @param_def_code, '.param '.$var.' = '.$all_vals[0]."\n";
    } else { # Place non-varied and non-swept .param variable declaration (defined as
                           its appropriate value)
      push @param_def_code, '.param '.$var.' = '.$spice_params{$var}."\n";
    }
  }
  splice @spice, $Netlist_Offset, 0, @param_def_code; # Add the .param declarations to
                           the spice netlist
  foreach (keys %model_linenum) { # Must now correct the line numbers that were
                           previously stored since @param_def_code was inserted at the
                           top of the netlist
                                  # Only the model line numbers were affected
    $model_linenum{$_} += scalar @param_def_code;
  }

  foreach (@spice) {
    while((my $placeholder, my $expression) = each %spice_placeholders) { # Look
                           through the spice netlist, and replace placeholders with
                           the proper .param variable expressions
      s/$placeholder/\{$expression\}/ig;
    }
  }
}

##########
my %fet_linenum; # Holds the line number for the declaration of a given fet
my %fet_model; # Holds the model name for the declaration of a given fet
my %model_fets; # Holds the fet names (as a space delimited string) for a given model
                           name
##########

for( $i = 0; $i < scalar @spice; $i++ ) {
  if( $spice[$i] =~ m/^(m[\w.]*)\s+[\w.]+\s+[\w.]+\s+[\w.]+\s+[\w.]+\s+([\w.]+)/i ) {
    # Found a transistor
    my $fetname = lc $1;
    my $model = lc $2;
    my $made_phys_copy_model = 0;
    die 'Error: ', $fetname, ' (line #', $i+1, ') refers to nonexistant model "',
                           $model, "\"\n" if not exists $model_linenum{$model};

    if( exists $vary_params{$model} ) { # This transistor has varied parameters

      if( $spice[$i] =~ m/\s+m\s*=\s*(\d+)/i and $1 > 1 ) { # This transistor has
                           multiple fins/fingers (or FinFETs in parallel)
        my $nfingers = $1;
        $spice[$i] =~ s/\s+m\s*=\s*\d+/ m=1/i; # Change the transistor to only one
                           finger
        splice @spice, $i+1, 0, ($spice[$i]) x ($nfingers-1); # Add identical copies
                           of this transistor to equal the finger count
        my $j;
        for( $j = 1; $j <= $nfingers; $j++ ) {
          $spice[$i+$j-1] =~ s/^$fetname/$fetname.$j/i; # Append the finger number to
                           each transistor (e.g. mp.1, mp.2, ... mp.nfingers)
        }
        $fetname .= '.1'; # Update the fetname for the FinFET being analyzed in the
                           loop (the first finger of the transistor)
      }

      foreach (split ' ', $vary_params{$model}) { # Iterate through all parameters
                           that are varied in this model
        # Count this fetname:param in the number of required data columns
        $data_column{$fetname.':'.$_} = $col++;
        if( exists $model_inline_param{$model.':'.$_} ) { # This parameter is declared
                           inline
          # Append the parameter (with mean) on line if not already there
          if( $spice[$i] !~ m/\s+$_\s*=\s*\S+\s+/i ) {
            chomp $spice[$i];
            $spice[$i] = $spice[$i].' '.$_.'='.$vy_param_means{$model.':'.$_}."\n";
          }
        } elsif( $made_phys_copy_model == 0 ) { # This parameter is declared in the
                           model declaration (and the model dec. must be duplicated)
          # Make a physical copy of the model and name the copy "fetname" (this is
                           only done once)
          $made_phys_copy_model = 1;
          my $fet_model_linenum = scalar @spice;
          # Duplicate the spice model
          push @spice, $spice[$model_linenum{$model}];
          # Change the fet's model name to $fetname
          die 'Error: The model name ', $model, ' was matched multiple times (net and
                           transistor names cannot be equal to model name) on spice
                           line ',
            $i+1, ":\n", $spice[$i], "\n" if ($spice[$i] =~ s/\s+$model\s+/ $fetname
                           /ig) > 1; # Note: "s/ / /" returns number of substitutions
                           made
          # Change the model's name to $fetname
          $spice[$fet_model_linenum] =~ s/^\.MODEL\s+$model\s+/\.MODEL $fetname /i;
          # Update hashes (duplicate entries for this duplication of the model)
          $model_linenum{$fetname} = $fet_model_linenum;
          $model_fettype{$fetname} = $model_fettype{$model};
          $vary_params{$fetname} = $vary_params{$model};
          my @inline_keys = keys %model_inline_param;
          foreach (@inline_keys) {
            if( m/^$model:(\w+)$/i ) {
              $model_inline_param{$fetname.':'.$1} =
                           $model_inline_param{$model.':'.$1};
            }
          }
          if( exists $sweep_params{$model} ) {
            my @sw_modelparam = keys %sweep_vals;
            foreach (@sw_modelparam) {
              if( m/^$model:(\w+)$/i ) {
                $sweep_vals{$fetname.':'.$1} = $sweep_vals{$model.':'.$1};
              }
            }
          }
        }
      }
    }

    # Keep track of the models and line numbers for each transistor, plus the names of
                           the transistors for each model
    #  (do this here since the fetname could be modified if the transistor has
                           multiple fins/fingers)
    $fet_linenum{$fetname} = $i;
    $fet_model{$fetname} = $model;
    $model_fets{$model} = exists $model_fets{$model} ? $model_fets{$model}.'
                           '.$fetname : $fetname;
  }
}

push @spice, "\n";
push @spice, ".END\n"; # Appending .END at end of netlist
push @spice, "\n";

##########
my $req_col; # The number of data columns required for the netlist
my %r_data_column; # Holds the model:param (or option:param) combination for a given
                           column number
##########

print "\nSummary of data columns for Parameter Variations:\n";
%r_data_column = reverse %data_column;
for( $i = 0; exists $r_data_column{$i}; $i++ ) {
  print 'Column ', $i+1, ' --> ', $r_data_column{$i}, "\n";
}
print "\n";

$req_col = $i; # $i holds the required number of columns from the for loop

# Output the modified netlist to 0.i in the simulation directory (it is functionally
                           equivalent to the original netlist)
open OUT, ">$deckfile0" or die 'Error: Cannot open ', $deckfile0, ' due to: ', $!;
print OUT @spice;
close OUT;

if( scalar (keys %sweep_vals) > 0 ) { # Values are swept, so generate netlists for
                           each sweep combination
  for( my $sweep_iter = 0; $sweep_iter < $sw_num_combos; $sweep_iter++ ) {
    while((my $sw_name, my $sw_lst) = each %sweep_vals) {
      my @sw_vals = split ' ', $sw_lst;

      die 'Internal error: Cannot extract fetname:param from ', $sw_name unless
                           $sw_name =~ m/([\w.]+):(\w+)/i;
      my $model = $1;
      my $param = $2;
      if( $model eq 'variable' ) { # Sweeps a .param variable
        die 'Internal error: line number missing for .param variable: ', $param unless
                           exists $var_linenum{$param};
        $spice[$var_linenum{$param}] =~ s/\s+$param\s*=\s*\S+/
                           $param=$sw_vals[$sweep_iter]/i;

      } else { # This is a model:param being swept
        if( !exists $model_inline_param{$sw_name} ) { # Normal fet parameter --
                           defined in .MODEL
          # Note: model name for a fet parameter defined in .MODEL is renamed to be
                           the same as fetname
          die 'Internal error: line number missing for model: ', $model unless exists
                           $model_linenum{$model};
          $spice[$model_linenum{$model}] =~ s/\s+$param\s*=\s*\S+/
                           $param=$sw_vals[$sweep_iter]/i;

        } else { # Inline fet parameter
          die 'Internal error: fetnames missing for model ', $model unless exists
                           $model_fets{$model};
          foreach my $fet_name (split ' ', $model_fets{$model}) {
            die 'Internal error: line number missing for fet: ', $fet_name unless
                           exists $fet_linenum{$fet_name};
            $spice[$fet_linenum{$fet_name}] =~ s/\s+$param\s*=\s*\S+/
                           $param=$sw_vals[$sweep_iter]/i;
          }
        }
      }
    }

    my $deckfile_sw0 = $dirname.$sweep_titles[$sweep_iter].'0'.$Fmt_Input; # Make a
                           0.i for each sweep combination
    open OUT, ">$deckfile_sw0" or die 'Error: Cannot open ', $deckfile_sw0, ' due to:
                           ', $!;
    print OUT @spice;
    close OUT;
  }
}

##########
my $qmc_line; # The QMC command line to execute
my $qmc_worked = 0; # A flag to see if CMU QMC generated values; it may not if #sims
                           (aka #rows) is too small for the number of columns required
                           (aka dimensions)
##########

if( !defined $op_corners ) { # Run the CMU QMC program to obtain data columns for
                           sweep or Monte Carlo initialization (if not doing corner
                           simulations)
  die "Number of Monte Carlo simulations not included in arguments, quitting...\n"
                           unless defined $sim_num and $sim_num > 0;
  $qmc_line = join ' ', $CMU_QMC_Exec, $qmc_seq, $req_col, $sim_num, 1;
  print 'Using "', $qmc_line, "\" to generate values for Monte Carlo
                           simulations...\n";
  open VALUES, "$qmc_line |" or die 'Error: "', $qmc_line, '" failed';
}
```

```
print "Generating netlists...\n";
##########
my $copy_num = 0; # The number of the netlist the loop is currently generating
my @vals; # An array of the randomized values for the current netlist being generated
my $val_col; # The size of @vals (the number of columns the randomized number set has)
##########

# Print the QMC values summary to file
open QMC_SUM, ">$Qmc_Sum_Prefix$dirtext$Qmc_Sum_Filetype" or die "Error: Cannot create
                           $Qmc_Sum_Prefix$dirtext$Qmc_Sum_Filetype for output due to
                           ", $!;

# This gets a little messy for corner simulations:
# Do the loop for generating corner netlists based on the corner constants, but do not
                           read from <VALUES>
goto LBEGIN if defined $op_corners;

while( <VALUES> ) {
  next unless m/^\d+/ or m/^-/; # Discards any line(s) of text (e.g. a first line of
                           "Sobol")

  # Obtain a line of randomized values from the randomized number set (don't do if
                           $op_corners is selected)
  @vals = split;
  $val_col = scalar @vals;
  if( $val_col < $req_col ) {
    close VALUES unless defined $op_corners;
    close QMC_SUM;
    die 'Internal error: Not enough columns (need ', $req_col, ', has ', $val_col, '
                           [value set is: ', $_, ']) in randomized number set for
                           sim#', $copy_num+1;
  } # Incremented copy_num, or else Internal Error message would be one off


  LBEGIN: $copy_num++;
  $qmc_worked = 1; # QMC must have worked, since we entered the while loop

  if( $copy_num == 1 ) { # This is the first varied netlist, so must print column
                           headings for the QMC summary file
    print QMC_SUM 'file number,';
    for( $i = 0; exists $r_data_column{$i}; $i++ ) {
      print QMC_SUM 'value ', $r_data_column{$i}, ','; # The final (weighted) value of
                           the variable:param or fetname:param pair
      print QMC_SUM 'location ', $r_data_column{$i}, ','; # The location of the final
                           value relative to the mean (it's the QMC value from file--
                           zero mean, unit std. dev.)
    }
  }
  print QMC_SUM "\n", $copy_num, ','; # Print the file number for the new row in the
                           QMC summary file

  # Cycle through every column of the randomized number set, and appropriately set the
                           variable it corresponds to
  for( $i = 0; exists $r_data_column{$i}; $i++ ) {
    my $final_val; # The final (weighted) value of the variable:param or fetname:param
                           pair
    my $qmc_val; # The location of the final value relative to the mean (it's the QMC
                           value from file--zero mean, unit std. dev.)

    if( $r_data_column{$i} !~ m/([\w.]+):(\w+)/i ) {
      close VALUES unless defined $op_corners;
      close QMC_SUM;
      die 'Internal error: Cannot extract fetname:param from ', $r_data_column{$i};
    }
    my $fetname = $1;
    my $param = $2;
    if( $fetname eq 'variable' ) { # Modifies a .param variable
      if( !exists $vy_var_means{$param} or !exists $vy_var_stddevs{$param} ) {
        close VALUES unless defined $op_corners;
        close QMC_SUM;
        die 'Internal error: mean or std_dev missing for .param variable:', $param;
      }
      if( !exists $var_linenum{$param} ) {
        close VALUES unless defined $op_corners;
        close QMC_SUM;
        die 'Internal error: line number missing for .param variable: ', $param;
      }
      $qmc_val = $vals[$i]; # Note: .param variables are not placed into the
                           data_columns hash for corner simulations, so no risk of
                           corner sims here
      if( $vy_var_means{$param} eq 'file' ) { # Directly substitute randomized number
                           from randomized number set
        $final_val = $qmc_val;
      } elsif( $vy_var_stddevs{$param} eq 'default' ) { # Scale random number from
                           random number set with mean and default std. dev.
        $final_val = $vy_var_means{$param} + $qmc_val * $vy_var_means{$param} /
                           $Default_Std_Dev_Factor;
      } else { # Scale random number from random number set with mean and specified
                           std. dev.
        $final_val = $vy_var_means{$param} + $qmc_val * $vy_var_stddevs{$param};
      }
      $spice[$var_linenum{$param}] =~ s/\s+$param\s*=\s*\S+/ $param=$final_val/i;
    } else { # Modifies a fet parameter
      if( !exists $fet_model{$fetname} ) {
        close VALUES unless defined $op_corners;
        close QMC_SUM;
        die 'Internal error: The model for "', $fetname, '" could not be found';
      }
      my $orig_model = $fet_model{$fetname};
      if( !exists $vy_param_means{$orig_model.':'.$param} or !exists
                           $vy_param_stddevs{$orig_model.':'.$param} ) {
        close VALUES unless defined $op_corners;
        close QMC_SUM;
        die 'Internal error: mean or std_dev missing for ', $orig_model, ':', $param;
      }

      if( !exists $model_inline_param{$orig_model.':'.$param} ) { # Normal fet
                           parameter -- defined in .MODEL
        # Note: model name for a fet parameter defined in .MODEL is renamed to be the
                           same as fetname
        if( !exists $model_linenum{$fetname} ) {
          close VALUES unless defined $op_corners;
          close QMC_SUM;
          die 'Internal error: line number missing for model: ', $fetname;
        }
        # Set the corner value "cval" to the number of specified standard deviations
                           with the +/- factor
        my $cval = ($model_fettype{$orig_model} eq 'nmos' ?
                           $op_corners*$Corner_N_Factors[$copy_num-1] :
                           $op_corners*$Corner_P_Factors[$copy_num-1]) if defined
                           $op_corners;
        $qmc_val = defined $op_corners ? $cval : $vals[$i];
        if( $vy_param_means{$orig_model.':'.$param} eq 'file' ) { # Directly
                           substitute randomized number from randomized number set
          close QMC_SUM if defined $op_corners;
```

```
        die 'Error: Param. value from file makes no sense for corner test' if
                         defined $op_corners;
        $final_val = $qmc_val;
      } elsif( $vy_param_stddevs{$orig_model.':'.$param} eq 'default' ) { # Scale
                         random number from random number set with mean and default
                         std. dev.
        $final_val = $vy_param_means{$orig_model.':'.$param} + $qmc_val *
                         $vy_param_means{$orig_model.':'.$param} /
                         $Default_Std_Dev_Factor;
      } else { # Scale random number from random number set with mean and specified
                         std. dev.
        $final_val = $vy_param_means{$orig_model.':'.$param} + $qmc_val *
                         $vy_param_stddevs{$orig_model.':'.$param};
      }
      $spice[$model_linenum{$fetname}] =~ s/\s+$param\s*=\s*\S+/
                         $param=$final_val/i;
    } else { # Inline fet parameter
      if( !exists $fet_linenum{$fetname} ) {
        close VALUES unless defined $op_corners;
        close QMC_SUM;
        die 'Internal error: line number missing for fet: ', $fetname;
      }
      # Set the corner value "cval" to the number of specified standard deviations
                         with the +/- factor
      my $cval = ($model_fettype{$orig_model} eq 'nmos' ?
                         $op_corners*$Corner_N_Factors[$copy_num-1] :
                         $op_corners*$Corner_P_Factors[$copy_num-1]) if defined
                         $op_corners;
      $qmc_val = defined $op_corners ? $cval : $vals[$i];
      if( $vy_param_means{$orig_model.':'.$param} eq 'file' ) { # Directly
                         substitute randomized number from randomized number set
        close QMC_SUM if defined $op_corners;
        die 'Error: Param. value from file makes no sense for corner test' if
                         defined $op_corners;
        $final_val = $qmc_val;
      } elsif( $vy_param_stddevs{$orig_model.':'.$param} eq 'default' ) { # Scale
                         random number from random number set with mean and default
                         std. dev.
        $final_val = $vy_param_means{$orig_model.':'.$param} + $qmc_val *
                         $vy_param_means{$orig_model.':'.$param} /
                         $Default_Std_Dev_Factor;
      } else { # Scale random number from random number set with mean and specified
                         std. dev.
        $final_val = $vy_param_means{$orig_model.':'.$param} + $qmc_val *
                         $vy_param_stddevs{$orig_model.':'.$param};
      }
      $spice[$fet_linenum{$fetname}] =~ s/\s+$param\s*=\s*\S+/ $param=$final_val/i;
    }
  }
  print QMC_SUM $final_val, ',', $qmc_val, ','; # Print the final value and QMC
                         value of this variable:param or fetname:param pair to the
                         QMC summary file
}

if( scalar (keys %sweep_vals) > 0 ) { # Values are swept, so generate netlists for
                         each sweep combination
  for( my $sweep_iter = 0; $sweep_iter < $sw_num_combos; $sweep_iter++ ) {
    while((my $sw_name, my $sw_lst) = each %sweep_vals) {
      my @sw_vals = split ' ', $sw_lst;
      if( $sw_name !~ m/([\w.]+):(\w+)/i ) {
        close VALUES unless defined $op_corners;
        close QMC_SUM;
        die 'Internal error: Cannot extract fetname:param from ', $sw_name;
      }
      my $model = $1;
      my $param = $2;
      if( $model eq 'variable' ) { # Sweeps a .param variable
        if( !exists $var_linenum{$param} ) {
          close VALUES unless defined $op_corners;
          close QMC_SUM;
          die 'Internal error: line number missing for .param variable: ', $param;
        }
        $spice[$var_linenum{$param}] =~ s/\s+$param\s*=\s*\S+/
                         $param=$sw_vals[$sweep_iter]/i;
      } else { # This is a model:param being swept
        if( !exists $model_inline_param{$sw_name} ) { # Normal fet parameter --
                         defined in .MODEL
          # Note: model name for a fet parameter defined in .MODEL is renamed to be
                         the same as fetname
          if( !exists $model_linenum{$model} ) {
            close VALUES unless defined $op_corners;
            close QMC_SUM;
            die 'Internal error: line number missing for model: ', $model;
          }
          $spice[$model_linenum{$model}] =~ s/\s+$param\s*=\s*\S+/
                         $param=$sw_vals[$sweep_iter]/i;
        } else { # Inline fet parameter
          if( !exists $model_fets{$model} ) {
            close VALUES unless defined $op_corners;
            close QMC_SUM;
            die 'Internal error: fetnames missing for model ', $model;
          }
          foreach my $fet_name (split ' ', $model_fets{$model}) {
            if( !exists $fet_linenum{$fet_name} ) {
              close VALUES unless defined $op_corners;
              close QMC_SUM;
              die 'Internal error: line number missing for fet: ', $fet_name;
            }
            $spice[$fet_linenum{$fet_name}] =~ s/\s+$param\s*=\s*\S+/
                         $param=$sw_vals[$sweep_iter]/i;
          }
        }
      }
    }

    # Write varied netlist into the varied simulation directory
    # Name is taken from the corner names if corner simulations are being generated
    # else, the name is a numeral (the value of "copy_num")
    my $deckfile_n = defined $op_corners ?
                         $dirname.$sweep_titles[$sweep_iter].$Corner_Names[$copy_num
                         -1].$Fmt_Input :
                         $dirname.$sweep_titles[$sweep_iter].$copy_num.$Fmt_Input;
    open OUT, ">$deckfile_n" or die 'Error: Cannot open ', $deckfile_n, ' due to: ',
                         $!;
    print OUT @spice;
    close OUT;
  }
} else { # Not sweeping any values, so just print the varied netlist to file (once)
  # Write varied netlist into the varied simulation directory
  # Name is taken from the corner names if corner simulations are being generated
  # else, the name is a numeral (the value of "copy_num")
  my $deckfile_n = defined $op_corners ? $dirname.$Corner_Names[$copy_num-
                         1].$Fmt_Input : $dirname.$copy_num.$Fmt_Input;
  open OUT, ">$deckfile_n" or die 'Error: Cannot open ', $deckfile_n, ' due to: ',
                         $!;
  print OUT @spice;
```

```
    close OUT;
  }

  goto LEXIT if $copy_num >= $sim_num; # Finished with netlist generation
  goto LBEGIN if defined $op_corners; # Loop back to top if doing corner sims
}

LEXIT: print QMC_SUM "\n";

close QMC_SUM;

if( !defined $op_corners ) { # Corner simulations did not use QMC, so no need to close
                           pipe or error check
  close VALUES;
  die "Error: CMU QMC ($qmc_line) failed to generate values for Monte Carlo
                           simulations (#sims may be too low)" unless $qmc_worked !=
                           0;
}

print 'Warning: File terminated prematurely, only ', $copy_num, " files created\n" if
                           $copy_num < $sim_num;
```

A.5 batchexec_ufdg.pl

```
#!/usr/bin/perl -w
use strict;
use warnings;

my $Version_Date = 'Sept. 11, 2012';
my $Script_Name = 'batchexec_ufdg.pl';

# Requires script:
my $Run_Ufdg = 'run_ufdg.pl';

# Constants:
my $Default_Run_Opts = 't';
my $Failures_Output_Prefix = 'fails_';
my $Failures_Output_Filetype = '.txt';
my $Netlist_Filetype = 'i';
my $Username = $ENV{USER} || $ENV{LOGNAME};
my $User_Email = defined $Username ? $Username.'@eecs.wsu.edu' : 'UNDEFINED';

# Signal (interrupt) handler for SIGUSR1 (#10):
my $num_complete = 0;
my $num_total = 0;
my $last_finishtime = "None";
# Procedure: update_handler
# Summary:
#   Prints the current progress of the script
sub update_handler {
  print 'Update from ', $Script_Name, ': ', $num_complete, ' of ', $num_total, "
                           simulations are complete\n",
        'Last sim finished at ', $last_finishtime, "\n\n";
  $SIG{'USR1'} = \&update_handler;
}
$SIG{'USR1'} = \&update_handler;

# Signal (interrupt) handler for SIGTERM (#15):
my %children;
# Procedure: clean_term_handler
# Summary:
#   If the terminate signal is given, also kills each simulation child process
sub clean_term_handler {
  kill 'TERM', keys %children;
  exit 128+15;
}
$SIG{'TERM'} = \&clean_term_handler;

# Usage:
my $Usage = ''.
'Usage: '.$Script_Name."  [OPTIONS]  directory_name\n".
"Options:\n".
    "\t-0           Do not simulate the nominal netlist (0.i)\n".
    "\t-b N         Begin batch simulations with the N'th netlist in the
                          directory\n".
    "\t-e N         End batch simulations with the N'th netlist in the directory\n".
    "\t-l N         Limit the number of simulations to N; once N sims finish without
                          failures, kill all other busy sims\n".
    "\t-m           Send an email to \"".$User_Email."\" upon \"".$Script_Name."\"
                          completion\n".
    "\t-p N         At a maximum, only run N simulations in parallel (for process
                          friendliness on a shared server)\n".
    "\t-r run_opts   Use \"run_opts\" as the options for \"".$Run_Ufdg."\" [cmd will
                          be: ".$Run_Ufdg." -run_opts]\n".
    "\t               Note: By default, the -t option (no timer) is used.  The
                          \"run_opts\" will be used instead\n".
    "\t-s           Execute all netlists in subdirectories of \"directory_name\"\n".
    "\t-x exec_name  Batch execute \"exec_name\" instead of \"".$Run_Ufdg."\"\n";
my $Min_Arg_Num = 1;

# batchexec_ufdg.pl
#
# Created by Mike Turi
# Washington State University
# School of Electrical Engineering & Computer Science
# High Performance Computer Systems (HiPerCopS) Group
#
# Note: Code is semi-tested; may fail under certain conditions
#
# This is a Perl script used to run a set of simulations from a
# directory of netlists.  Often used in tandem with init_batch_ufdg.pl.
#
# Usage: Provide the directory name, and the netlists in the
# directory will be simulated using UFDG.  The run_ufdg.pl script will
# be used unless a different executable name is specified with the
# -x option.
print "**$Script_Name version date: $Version_Date**\n";
die $Usage if scalar @ARGV < 1;
my $argnum = scalar @ARGV;
die "Error: Not enough parameters\n\n$Usage" if $argnum < $Min_Arg_Num;

my $start_time = localtime;
my $calling_cmd_line = join ' ', $Script_Name, @ARGV;
my $i = 0;
my $arg = shift;
my $op_rm0;
my $op_begfile;
my $op_endfile;
my $op_sim_limit;
my $op_email;
my $op_max_parallel;
my $run_opts = $Default_Run_Opts;
my $op_subdirs;
my $exec_name;
while( $arg =~ m/^-/ ) {
  die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
  if( $arg =~ m/^-(\S+)/ ) { # Found options
    my $argmatch = $1;

    # Check single letter options first
    if( $argmatch =~ m/0/i ) {
      $op_rm0 = 1; # Do not simulate 0.i (if it exists)
    }
    if( $argmatch =~ m/m/i and defined $Username ) {
      $op_email = 1; # Send an email upon script completion (if email address is
                           defined)
    }
    if( $argmatch =~ m/s/i ) { # Execute netlists in subdirectories
      $op_subdirs = 1;
    }

    # Check options which require another argument
    if( $argmatch =~ m/b/i ) { # Begin batch simulations with netlist #N
      $op_begfile = shift;
      die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
      $op_begfile--; # Change notation from 1..#files to 0..#files-1
    }
    if( $argmatch =~ m/e/i ) { # End batch simulations with netlist #N
      $op_endfile = shift;
      die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
      $op_endfile--; # Change notation from 1..#files to 0..#files-1
    }
    if( $argmatch =~ m/l/i ) { # Use a limit of N sims
      $op_sim_limit = shift;
      die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
    }
    if( $argmatch =~ m/p/i ) { # Only run N sims in parallel
      $op_max_parallel = shift;
      die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
    }
    if( $argmatch =~ m/r/i ) { # Use the run_opts as the options for run_ufdg.pl
      $run_opts = shift;
      die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
    }
    if( $argmatch =~ m/x/i ) { # Batch execute "exec_name"
      $exec_name = shift;
      die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
    }
  }
  $arg = shift;
}
my $deck_dir = $arg;
if( $deck_dir !~ m/\/$/ ) { $deck_dir .= '/'; } # Add the trailing slash if it doesn't
                           exist

$exec_name = $Run_Ufdg.' -'.$run_opts unless defined $exec_name;

die 'Error: Cannot find directory "', $deck_dir, '"' unless -d $deck_dir;
opendir D, $deck_dir;
# Get a list of all spice input files (.i files) from the deck directory
#   Note: If traversing subdirectories, then @files will have entire paths, else,
                           @files only contain the basenames
#         This is done so numerical sorting is possible if simulating within the top
                           directory
```

```
my @files_with_suffix = defined $op_subdirs ? `find $deck_dir -name 
                           \\*\\.$Netlist_Filetype` : grep /\.$Netlist_Filetype$/i, 
                           readdir D;
closedir D;
# Remove the ".i" suffix from the end of each file (with map and substr)
my @files = map { substr $_, 0, (rindex $_, '.') } @files_with_suffix;
# If files have character names, then sort alphabetically, else, sort numerical names 
                           numerically
@files = (join '', @files) =~ m/[a-z]/i ? sort @files : sort { $a <=> $b} @files;

if( defined $op_rm0 ) { # Ignore the nominal netlist (0.i) when simulating a directory
  if( defined $op_subdirs ) {
    my $deckname0 = $deck_dir.'0';
    @files = grep !/^$deckname0$/, @files; # Subdirs are traversed, so must remove 
                           main directory's 0.i from list of files
  } else {
    @files = grep !/^0$/, @files; # No subdirs traversed, so can just remove "0" from 
                           list of files
  }
}

my $op_begfile_print;
my $op_endfile_print;
if( defined $op_begfile and defined $op_endfile ) { # Use defined beginning/ending 
                           boundaries from user if defined
  $op_begfile = 0 unless $op_begfile >= 0 and $op_begfile < scalar @files;
  $op_endfile = (scalar @files)-1 unless $op_endfile >= 0 and $op_endfile < scalar 
                           @files and $op_endfile >= $op_begfile;
  $op_begfile_print = ($op_begfile+1).' 
                           ('.$files[$op_begfile].'.'.$Netlist_Filetype.')';
  $op_endfile_print = ($op_endfile+1).' 
                           ('.$files[$op_endfile].'.'.$Netlist_Filetype.')';
  @files = @files[$op_begfile..$op_endfile];
} elsif( defined $op_begfile ) {
  $op_begfile = 0 unless $op_begfile >= 0 and $op_begfile < scalar @files;
  $op_begfile_print = ($op_begfile+1).' 
                           ('.$files[$op_begfile].'.'.$Netlist_Filetype.')';
  $op_endfile_print = 'END ('.$files[-1].'.'.$Netlist_Filetype.')';
  @files = @files[$op_begfile..(scalar @files)-1];
} elsif( defined $op_endfile ) {
  $op_endfile = (scalar @files)-1 unless $op_endfile >= 0 and $op_endfile < scalar 
                           @files;
  $op_begfile_print = 'BEGIN ('.$files[0].'.'.$Netlist_Filetype.')';
  $op_endfile_print = ($op_endfile+1).' 
                           ('.$files[$op_endfile].'.'.$Netlist_Filetype.')';
  @files = @files[0..$op_endfile];
}

$num_total = scalar @files;
die 'Error: No spice netlists found in directory "', $deck_dir, '"' unless $num_total 
                           > 0;

if( defined $op_begfile or defined $op_endfile ) {
  print 'Files ', $op_begfile_print, ' through ', $op_endfile_print, " will be 
                           simulated\n";
  sleep 2;
}

my $child_pid;
my $pid;
my $cur_parallel = 0;
my $retval;
my @failures;
```

```
foreach (@files) {
  if( !(defined $op_max_parallel) or $cur_parallel < $op_max_parallel ) { # Spawn more
                           child simulations
    $child_pid = fork();
    die "Error: Fork in $Script_Name failed" unless defined $child_pid;
    # Keep track of the children pid numbers (hash value is filename, which is
                           reported if simulation failure occurs)
    $children{$child_pid} = $_;
    $cur_parallel++;
    if( !$child_pid ) { # The child executes the executable (by default: run_ufdg.pl)
      my $full_deck_path = defined $op_subdirs ? $_ : $deck_dir.$_;
      exec "$exec_name $full_deck_path" or die "Error: Couldn't exec \"$exec_name
                           $full_deck_path\"";
    }
  } else { # Must wait for a child simulation to finish before starting any more
    do {
      $pid = wait();
    } while( !exists $children{$pid} );
    $retval = $? >> 8;
    if( $retval != 0 ) { # This child simulation died due to an error
      push @failures, $children{$pid};
    }
    delete $children{$pid};
    $cur_parallel--;
    $num_complete++;
    $last_finishtime = `date`;
    if( defined $op_sim_limit and $num_complete >= ($op_sim_limit+(scalar @failures))
                           ) { # Limit has been reached, end all other simulations
      kill 'TERM', keys %children;
      $cur_parallel = 0; # All children should be dead (do not want to enter the while
                           loop below)
      last;
    }
    redo;
  }
}

while( $cur_parallel > 0 ) { # Wait if child simulations are still running
  do {
    $pid = wait();
  } while( !exists $children{$pid} );
  $retval = $? >> 8;
  if( $retval != 0 ) { # This child simulation died due to an error
    push @failures, $children{$pid};
  }
  delete $children{$pid};
  $cur_parallel--;
  $num_complete++;
  $last_finishtime = `date`;
  if( defined $op_sim_limit and $num_complete >= ($op_sim_limit+(scalar @failures)) )
                           { # Limit has been reached, end all other simulations
    kill 'TERM', keys %children;
    last;
  }
}

if( scalar @failures > 0 ) { # Some simulation failures occurred, record this
  my @deck_basename = split /\//, $deck_dir; # The directory name (basename of the
                           pathname) is at the end of the array
  my $failures_outfile = $Failures_Output_Prefix.$deck_basename[-
                           1].$Failures_Output_Filetype;
  open OUT, ">$failures_outfile" or die 'Error: Cannot open ', $failures_outfile, '
                           due to: ', $!;
  my $failure_list = join "\n", @failures;
  print OUT "$failure_list\n";
  close OUT;
}

if( defined $op_email ) {
  open MAIL, "|/usr/sbin/sendmail -t";
  print MAIL 'To: ', $User_Email, "\n";
  print MAIL 'From: ', $User_Email, "\n";
  print MAIL 'Subject: ', $Script_Name, ' finished (', $calling_cmd_line, ' from ',
                           $start_time, '); ', $num_complete, ' sims complete; ',
                           scalar @failures, " sims failed\n\n";
  close MAIL;
}
```

A.6 meas_ezwave.pl

```
#!/usr/bin/perl -w
use strict;
use warnings;

my $Script_Name = 'meas_ezwave.pl';
my $Version_Date = 'Apr. 24, 2012';

# Constants:
my $Default_Fmt_Ezwave = '.csv';
my $Meas_Sum_Prefix = 'msum_';
my $Meas_Sum_Filetype = '.csv';
my $Stat_Sum_Prefix = 'msum_';
my $Stat_Sum_Filetype = '.txt';

my $Works_Keyword = 'works';
my $Flag_Keyword = 'flag';
my $Filename_Keyword = 'filename';

# Usage:
my $Usage = ''.
"Usage: $Script_Name  deckname_OR_directory_name  tcl_script_name\n".
"Options:\n".
    "\t-0      Ignore the nominal simulation (e.g. 0.csv) when computing overall
                           statistics\n".
    "\t-b N    Begin batch simulations with the N'th netlist in the directory\n".
    "\t-e N    End batch simulations with the N'th netlist in the directory\n".
    "\t-f fmt  Use another file format \"fmt\" (e.g. \"sti\") for input to EZWave\n".
    "\t-l N    Use a limit of the first N results for statistics (also omit failed
                           sims from the results used)\n".
    "\t-r      Randomize the simulation results (useful if using a limit of the
                           results for statistics)\n".
    "\t-s      Measure all netlists in subdirectories of \"directory_name\"\n";
my $Min_Arg_Num = 2;

# Procedure: sum
# Summary:
#   Computes the sum for a passed array
# Parameter: An array of numbers to sum
sub sum {
  my $s = 0;
  foreach (@_) {
    $s += $_;
  }
  return $s
}

# Procedure: mean
# Summary:
#  Computes the mean (average) for a passed array
# Parameter: An array of numbers to be averaged
sub mean {
  return (sum @_)/(scalar @_);
}

# Procedure: std
# Summary:
#   Computes the standard deviation for a passed array
# Parameter: An array of numbers to be used in finding the standard deviation
sub std {
  my $sq_err = 0;
  my $ave = mean @_;
  foreach (@_) {
    $sq_err += ($_ - $ave)**2;
  }
  return sqrt ($sq_err/((scalar @_)-1));
}

# meas_ezwave.pl
# Created by Mike Turi
# WSU EECS: HiPerCopS
# Note: Code is semi-tested; may fail under certain conditions
#
# This is a Perl script used to work with a TCL script to gather measurement data from
# Mentor Graphics EZWave.  The script expects the TCL script to use STDIN for a
# carriage returned list of pathnames (to .csv/.sti files) to measure (only one
                          pathname
# if only measuring one file).  The script expects the TCL script to use STDOUT in the
# following way: first line to STDOUT contains a pipe "|" delimited string of the
# measurement names/titles, the second line to STDOUT contains a pipe "|" delimited
# string of measurement data values (1st value on the line corresponds to the 1st
# measurment name, 2nd value corresponds to the 2nd measurment name, etc.), and
# additional lines (3rd and on) to STDOUT contain pipe "|" delimited strings of
# measurement data values for additional pathnames (in order the pathnames were
# given to the TCL script).  If taking measurments for one file, two lines of STDOUT
# are generated, the measurement names and the measurement data values.  If taking
# measurements for N files, N+1 lines of STDOUT are generated, the measurement names
# and N lines of measurement data values.
#
# Usage: Provide the filename (without the .xxx suffix--e.g. .csv or .sti) or a
# directory name, and the name of the TCL script to run.

print "**$Script_Name version date: $Version_Date**\n";
die $Usage if scalar @ARGV < 1;
my $argnum = scalar @ARGV;
die "Error: Not enough parameters\n\n$Usage" if $argnum < $Min_Arg_Num;

my $i = 0;
my $arg = shift;
my $op_rm0;
my $op_begfile;
my $op_endfile;
my $op_limit;
my $op_rand;
my $op_subdirs;
my $fmt_ezwave = $Default_Fmt_Ezwave;
while( $arg =~ m/^-/ ) {
  die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
  if( $arg =~ m/^-(\S+)/ ) { # Found options
    my $argmatch = $1;

    # Check single letter options first
    if( $argmatch =~ m/0/i ) {
      $op_rm0 = 1; # Ignore 0.csv or 0.sti (if it exists) when computing overall
                           statistics
    }
    if( $argmatch =~ m/r/i ) { # Randomize the order of the sim results
      $op_rand = 1;
    }
    if( $argmatch =~ m/s/i ) { # Measure netlists in subdirectories
      $op_subdirs = 1;
    }

    # Check options which require another argument
    if( $argmatch =~ m/b/i ) { # Begin measurements with simulation #N
      $op_begfile = shift;
      die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
      $op_begfile--; # Change notation from 1..#files to 0..#files-1
    }
    if( $argmatch =~ m/e/i ) { # End measurements with simulation #N
      $op_endfile = shift;
      die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
      $op_endfile--; # Change notation from 1..#files to 0..#files-1
    }
    if( $argmatch =~ m/f/i ) { # Use another file format "fmt" instead
      $fmt_ezwave = shift;
      die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
      $fmt_ezwave = '.'.$fmt_ezwave unless $fmt_ezwave =~ m/^\./;
    }
    if( $argmatch =~ m/l/i ) { # Use a limit of N for including simulation results in
                           statistics
      $op_limit = shift;
      die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
    }
  }
  $arg = shift;
}

my $deckname = $arg;
my $tcl_name = shift;
die 'Error: Cannot find tcl script "', $tcl_name, '"' unless -e $tcl_name;

my @files;
if( -d $deckname ) { # $deckname is actually the name of a directory with simulations
  if( $deckname !~ m/\/$/ ) { $deckname .= '/'; } # Add the trailing slash if it
                           doesn't exist
  opendir D, $deckname;
  # Get a list of all converted spice output files (e.g. .csv files) from the deck
                           directory (grep /(.csv)$/i, readdir D)
  #   Note: If traversing subdirectories, then @files will have entire paths, else,
                           @files only contain the basenames
  #         This is done so numerical sorting is possible if simulating within the top
                           directory
  my @files_with_suffix = defined $op_subdirs ? `find $deckname -name \\*$fmt_ezwave`
                           : grep /($fmt_ezwave)$/i, readdir D;
  closedir D;
  # Remove the ".csv" or ".sti" from the end of each file (with map and substr)
  @files = map { substr $_, 0, (rindex $_, '.') } @files_with_suffix;
  # If files have character names, then sort alphabetically, else, sort numerical
                           names numerically
  @files = (join '', @files) =~ m/[a-z]/i ? sort @files : sort { $a <=> $b} @files;

  if( defined $op_rm0 ) { # Ignore the nominal simulation (0.csv or 0.sti) when
                           simulating a directory
    if( defined $op_subdirs ) {
      my $deckname0 = $deckname.'0';
      @files = grep !/^$deckname0$/, @files; # Subdirs are traversed, so must remove
                           main directory's 0.csv/0.sti from list of files
    } else {
      @files = grep !/^0$/, @files; # No subdirs traversed, so can just remove "0"
                           from list of files
    }
  }

  my $op_begfile_print;
  my $op_endfile_print;
  if( defined $op_begfile and defined $op_endfile ) { # Use defined beginning/ending
                           boundaries from user if defined
    $op_begfile = 0 unless $op_begfile >= 0 and $op_begfile < scalar @files;
    $op_endfile = (scalar @files)-1 unless $op_endfile >= 0 and $op_endfile < scalar
                           @files and $op_endfile >= $op_begfile;
    $op_begfile_print = ($op_begfile+1).' ('.$files[$op_begfile].$fmt_ezwave.')';
    $op_endfile_print = ($op_endfile+1).' ('.$files[$op_endfile].$fmt_ezwave.')';
    @files = @files[$op_begfile..$op_endfile];
  } elsif( defined $op_begfile ) {
    $op_begfile = 0 unless $op_begfile >= 0 and $op_begfile < scalar @files;
    $op_begfile_print = ($op_begfile+1).' ('.$files[$op_begfile].$fmt_ezwave.')';
    $op_endfile_print = 'END ('.$files[-1].$fmt_ezwave.')';
    @files = @files[$op_begfile..(scalar @files)-1];
  } elsif( defined $op_endfile ) {
    $op_endfile = (scalar @files)-1 unless $op_endfile >= 0 and $op_endfile < scalar
                           @files;
    $op_begfile_print = 'BEGIN ('.$files[0].$fmt_ezwave.')';
    $op_endfile_print = ($op_endfile+1).' ('.$files[$op_endfile].$fmt_ezwave.')';
    @files = @files[0..$op_endfile];
  }

  die 'Error: No ', $fmt_ezwave, ' files found in directory "', $deckname, '"' unless
                           scalar @files > 0;

  if( defined $op_begfile or defined $op_endfile ) {
    print 'Files ', $op_begfile_print, ' through ', $op_endfile_print, " will be
                           measured\n";
    sleep 2;
  }

  if( !defined $op_subdirs ) { # Using the "find" command for subdirectories already
                           includes directory names in the pathname
    my $tmp = join '|', @files;  # Ugly way to add directory names prior to the
                           filenames (assumes pipe "|" isn't in the file name)
    $tmp =~ s/\|/\|$deckname/g;
    $tmp = $deckname.$tmp;
    @files = split /\|/, $tmp;
  }
} elsif( -r $deckname.$fmt_ezwave ) { # $deckname is the name of one netlist's .csv or
                           .sti file
  push @files, $deckname;
} else {
  die 'Error: Cannot find directory "', $deckname, '" or read from "', $deckname,
                           $fmt_ezwave, '"';
}

my $echo_names = join "\n", @files;
```

```
open EZWAVE, "echo \"$echo_names\" | run_wdb_server -do $tcl_name |" or die 'Error:
                           Cannot run EZWave with ', $tcl_name;

my $data_names_str = <EZWAVE>; # Measurement names are on the first line of TCL
                           output, data values are on subsequent lines
my @data_names = split /[,\n]/, $data_names_str;
my @tcl_data = <EZWAVE>;
close EZWAVE;

# Determine the indicies of each "special" measurement
my $works_index = -1;
my $flag_index = -1;
my $filename_index = -1;
for( $i = 0; $i < scalar @data_names; $i++ ) {
  if( $data_names[$i] eq $Works_Keyword ) {
    $works_index = $i;
  } elsif( $data_names[$i] eq $Flag_Keyword ) {
    $flag_index = $i;
  } elsif( $data_names[$i] eq $Filename_Keyword ) {
    $filename_index = $i;
  }
}

my @size_errors; # Error in size; either (data_lines != #_of_files --or--
                           #_of_measurements_in_a_line != #_of_measurement_names)
my @works_errors; # Errors from "works" variable in TCL measurement
my @flag_warnings; # Warnings from "flag" variable in TCL measurement

if( scalar @tcl_data != scalar @files ) { # Too few lines of data for number of lines
                           simulated
  die 'Error: Failure in EZWave, not all data gathered (', scalar @tcl_data, ' lines
                           of data for ', scalar @files, " files)" unless @files > 1;
  push @size_errors, 'Error: Failure in EZWave, not all data gathered ('.scalar
                           @tcl_data.' lines of data for '.scalar @files." files)";
}

if( defined $op_rand ) { # Randomize the order of the sim results
  for( $i = 0; $i < scalar @tcl_data; $i++ ) {
    my $j = rand scalar @tcl_data;
    @tcl_data[$i, $j] = @tcl_data[$j, $i];
  }
}

for( $i = 0; $i < scalar @tcl_data; $i++ ) {
  my @file_data = split /[,\n]/, $tcl_data[$i]; # All measurements for one file, now
                           as an array
  my $this_name = $filename_index >= 0 ? $file_data[$filename_index] : '#'.$i; # This
                           file's name
  my $limit = scalar @data_names; # How many pieces of measurement data this file has
                           (assume all pieces of data for now)

  if( scalar @file_data != scalar @data_names ) { # Too few measurements in this line
                           for number of measurements expected (# of measurement
                           names)
    push @size_errors, 'Error for file "'.$this_name.'": Has '.scalar @file_data.'
                           data measurements (Expected '.scalar @data_names.')';
    if( scalar @file_data < scalar @data_names ) {
      chomp $tcl_data[$i];
      $tcl_data[$i] .= ",size mismatch error\n"; # Place error in first missing
                           measurement (causes this file to be ignored in overall
                           stats and shows up in .csv file)
      $limit = scalar @file_data; # This file has fewer pieces of data, so change for-
                           loop limit accordingly
    }
  }

  for( my $j = 0; $j < $limit; $j++ ) { # Print the measurment name and value to
                           stdout
    print $data_names[$j], ': ', $file_data[$j], "\n";
  }

  # Capture (for later printing) the 'works' TCL error and 'flag' TCL warning if they
                           exist, are contained in this file's data, and are non-zero
  if( $works_index >= 0 and $works_index < $limit and ($file_data[$works_index] =~
                           m/error/i or $file_data[$works_index] != 0) ) {
    push @works_errors, 'Error for file "'.$this_name.'": Works is non-zero
                           ('.$file_data[$works_index].')';
  }
  if( $flag_index  >= 0 and $flag_index  < $limit and $file_data[$flag_index]  != 0 )
                           {
    push @flag_warnings, 'Warning for file "'.$this_name.'": Flag is non-zero
                           ('.$file_data[$flag_index].')';
  }
}

if( scalar @files > 1 ) {
  my @deck_basename = split /\//, $deckname; # The directory name (basename of the
                           pathname) is at the end of the array
  open MEAS_SUM, ">$Meas_Sum_Prefix$deck_basename[-1]$Meas_Sum_Filetype"
       or die "Error: Cannot create $Meas_Sum_Prefix$deck_basename[-
                           1]$Meas_Sum_Filetype for output due to ", $!;

  # Print the first row to the .csv file (sim_name and the names of all measurements)
  print MEAS_SUM $data_names_str;

  # Print the row of measurements for each simulation
  foreach (@tcl_data) {
    print MEAS_SUM;
  }
  print MEAS_SUM "\n";
  close MEAS_SUM;

  open STAT_SUM, ">$Stat_Sum_Prefix$deck_basename[-1]$Stat_Sum_Filetype"
       or die "Error: Cannot open $Stat_Sum_Prefix$deck_basename[-1]$Stat_Sum_Filetype
                           due to: ", $!;

  # Removing failed simulations from the overall statistics
  for( $i = 0; $i < scalar @tcl_data; $i++ ) {
    if( $tcl_data[$i] =~ m/error/i ) {

      my @file_data = split /[,\n]/, $tcl_data[$i]; # Let's pinpoint where the error
                           occurred
      my $this_name = $filename_index >= 0 ? $file_data[$filename_index] : '#'.$i; #
                           This file's name
      for( my $j = 0; $j < scalar @file_data; $j++ ) {
        if( $file_data[$j] =~ m/error/i ) {
          print 'Error for file "', $this_name, '": Measurement "', $data_names[$j],
                           "\" failed\n";
          print STAT_SUM 'Error for file "', $this_name, '": Measurement "',
                           $data_names[$j], "\" failed\n";
        }
      }

      splice @tcl_data, $i, 1; # Remove the failed simulation ($i now indexes the next
                           simulation
      $i--; # Compensate for the for-loop incrementing $i on the next pass
    }
  }
  my $complete_sims = $i; # Also == scalar @tcl_data
  print "\n";

  my $count;
  if( $complete_sims < 2 ) {
    print STAT_SUM "Skipping statistical summary (too few simulations successfully
                           completed without errors)\n";
    print "Skipping statistical summary (too few simulations successfully completed
                           without errors)\n";
  } else {
    # Print the means and standard deviations for each measurement to a summary file
    my $m;
    my $s;
    for( $i = 0; $i < scalar @data_names; $i++ ) { # Not using a foreach since $i is
                           used to find correct data in @tcl_data

      next if $data_names[$i] eq $Works_Keyword; # Don't do mean & std. dev. for works
                           entry
      next if $data_names[$i] eq $Flag_Keyword; # Don't do mean & std. dev. for flag
                           entry
      next if $data_names[$i] eq $Filename_Keyword; # Don't do mean & std. dev. for
                           filename entry
      my @meas_data;
      $count = 0;
      foreach (@tcl_data) {
        if( defined $op_limit and ++$count > $op_limit ) { last; }
        my @file_data = split /[,\n]/;
        push @meas_data, $file_data[$i]; # @meas_data contains all data for a given
                           measurement
      }
      $m = mean @meas_data;
      $s = std @meas_data;
      print STAT_SUM $data_names[$i], ":\nmean and stddev:\n", $m, "\n", $s, "\n";
      print $data_names[$i], ":\nmean and stddev:\n", $m, "\n", $s, "\n";
    }
  }

  print STAT_SUM $complete_sims, " simulations completed without errors\n";
  print $complete_sims, " simulations completed without errors\n";
  print STAT_SUM 'Statistics computed for ', $count-1, " simulations\n";
  print 'Statistics computed for ', $count-1, " simulations\n";

  foreach (@size_errors, @works_errors, @flag_warnings) { # Now print all of the
                           errors and warnings
    print STAT_SUM $_, "\n";
    print $_, "\n";
  }

  close STAT_SUM;

} else {
  # Only one file was simulated, print the data values again for easy copy-paste, then
                           print errors or warnings
  print "\n";
  my @file_data = split /[,\n]/, $tcl_data[0]; # All measurements for one file, now as
                           an array
  foreach (@file_data) {
    print $_, "\n";
  }
  print "\n";
  foreach (@size_errors, @works_errors, @flag_warnings) { # Now print all of the
                           errors and warnings
    print $_, "\n";
  }
}
```

A.7 meas_stub.tcl

```
# meas_stub.tcl
set Scriptname "meas_stub.tcl"
set Version_Date "Jun. 1, 2012"

puts stderr "**$Scriptname version date: $Version_Date**\n"

# Expect pathname(s) of deck(s) from stdin (usually by "echo deckpath |")
# Carriage returned list of pathnames for more than one: path1\npath2\npath3...
set deckpath [list]
set deckname [list]

while { [eof stdin] != 1 } {
  set x [gets stdin]

  if { [regexp "^\W*$" $x] } { # If there is a blank line, assume end of stdin
    break
  }

  lappend deckpath $x
  lappend deckname [lindex [split $x "/"] end]
}

# Measurement Settings:
# ---------------------
# Trigger when Read causes Bit/Nbit discharge to X volts
set RD_LOW_TRIG 0.5
set ANOTHER_MEAS_TRIG 0.9
set meas_setting_names [list "rd_low_trig" "another_trig"]
set meas_setting_values [list $RD_LOW_TRIG $ANOTHER_MEAS_TRIG]

# Constants:
# ----------
# 1/2 Risetime (RT) or Falltime (FT)
set HALF_RTFT 1e-12
set NET_B "bit"
set V_CELL "vddcell"
set V_SRC "vdd"

# Netlist Variables
set VAR_SIMLEN "vsimlen"
set VAR_ARRAY_SIZE "varray_size"
set net_var_names [list "sim length" "array size"]

# Loop Variables
set name_count 0
append name_print [join $meas_setting_names ","] ",filename,works"
set data_lst [list]

foreach path $deckpath name $deckname {

  set dataout $meas_setting_values
  set names $meas_setting_names
```

```
lappend names "filename"
lappend dataout $path

if { [file isfile "$path.csv"] == 0 } {
  puts stderr "Error ($Scriptname): File \"$path.csv\" not found"
  lappend dataout "error - could not open"
  lappend data_lst [join $dataout ","]
  dataset close
  continue
}

dataset open "$path.csv"
# ------------------------------------------------------------------
# begin measurement section

# Example: Using a voltage source to pass a variable value to the TCL script
set simlen [wfc "yval(wf(\"<$name/TRAN>V($VAR_SIMLEN)\"), 0)"]
set array_size [wfc "yval(wf(\"<$name/TRAN>V($VAR_ARRAY_SIZE)\"), 0)"]
set net_var_values [list $simlen $array_size]

# Locations:
# ----------
set LOC_RD0_BEG [expr 100e-12*$simlen+$HALF_RTFT]
set LOC_RD0_END [expr 155e-12*$simlen]

# For Errors - do some tests and set errors to a meaningful non-zero value for an
                         error
set rd0_ok [wfc "yval(wf(\"<$name/TRAN>V($NET_B)\"), $LOC_RD0_END)"]

set works ""
if { $rd0_ok > $RD_LOW_TRIG } {
  append works "Read0 Failed ($rd0_ok); "
}
lappend names "works"
if { $works ne "" } {
  lappend dataout "error = $works"
  lappend data_lst [join $dataout ","]
  dataset close
  continue
} else {
  lappend dataout 0
}

# Suggested to add netlist variable names and values now after it has been verified
                         that the simulation worked
set names [concat $names $net_var_names]
set dataout [concat $dataout $net_var_values]

# Example: Using yval to get a leakage value
set leakage [wfc "abs(yval(wf(\"<$name/TRAN>I($V_SRC)\"), $LOC_LK))"]
lappend names "leakage value" "double leakage"
lappend dataout $leakage
lappend dataout [expr {$leakage * 2}]

# Example: Using xval to get a delay
set rd_time [wfc "xval(wf(\"<$name/TRAN>V($NET_B)\"), $RD_LOW_TRIG, $LOC_RD0_BEG,
                         $LOC_RD0_END) - $LOC_RD0_BEG"]
# Caution: There could be 0, 1, or multiple x-values for a given y-interval --> must
                         check result before using
set testlen [llength $rd_time]
if { $testlen == 1 } {
  # No error: one crossing found
```

```
    lappend dataout $rd_time
  } elseif { $testlen < 1 } {
    # An error code: no crossings found
    lappend dataout "error $testlen"
  } else {
    # An error code: multiple crossings found
    lappend dataout "error $testlen"
  }
  lappend names "rd_time"
  # If $rd_time is used elsewhere in the script (e.g. see below) may want to abort the
                           rest of the script for this file
  if { $testlen != 1 } {
    lappend data_lst [join $dataout ","]
    # Important!  Must close dataset anytime "continue" is used, or else the same
                           file's dataset will be used for all subsequent measurements
                           in the foreach loop
    dataset close
    continue
  }

  # Example: Using avg to get an average current value
  set ave_current [wfc "avg(abs(wf(\"<$name/TRAN>I($V_CELL)\")), $LOC_RD0_BEG,
                           $LOC_RD0_BEG + $rd_time)"]
  lappend names "rd cell current"
  lappend dataout $ave_current

  # For Warnings -- do some tests and set flag to a meaningful non-zero value for a
                           warning
  set flag ""
  if { $ave_current > 10e-9 } {
    append flag "Ave. Current ($ave_current)) > 10nA; "
  }
  lappend names "flag"
  if { $flag eq "" } {
    set flag 0
  }
  lappend dataout $flag

  # end measurement section
  # -----------------------------------------------------------------
  if { [llength $names] > $name_count } {
    set name_count [llength $names]
    set name_print [join $names ","]
  }
  lappend data_lst [join $dataout ","]
  dataset close
}

puts $name_print
foreach i $data_lst {
  puts $i
}

exit
```

Appendix B

FinFET and CMOS Address Decoders

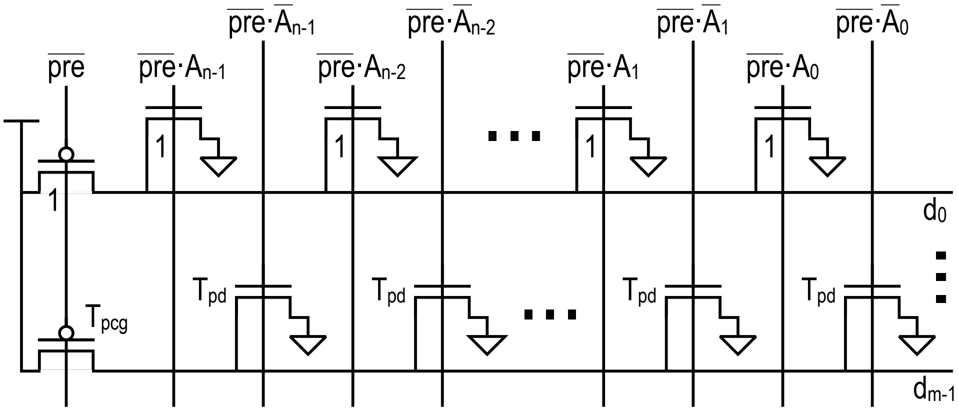
It is important to examine the power consumption of address decoders since this circuitry is always active in a memory sub-system. The majority of memory cells are often inactive since usually only one word is accessed per clock cycle. This in turn makes it very important to limit leakage power in memory cells. However, address decoders must continuously drive word-lines in order to enable one word of memory cells and disable the rest in a memory array. It is especially the case for smaller memories that a reduction in address decoder power consumption can significantly reduce the power of the overall memory sub-system.

While there has been a significant amount of research performed to reduce the power consumption of memory cells and of the memory read operation, there has not been much research on optimization of address decoders for high-speed and low-power performance. There has been some focus on optimally scaling decoders for larger memories by use of a pre-decoder [34]. Special gate-families, such as a half-swing pulse-mode gate family, can be used to design the pre-decoders to provide significant power savings when driving intermediate control signals to the local decoders [35].

Address decoders often function by precharging all word-lines and discharging all lines except for the addressed word-line. Address decoders can also be made more selective by precharging fewer word-lines based on the address. More selective decoders can save power by not needlessly charging and discharging all but the addressed word-line [36] [37]. Selective precharge has also been studied in other applications, such as to more-selectively precharge the match-lines in a content-addressable memory [38].

B.1 FinFET NOR Decoder

This work was originally presented in [39]. The NOR decoder uses one p-type transistor per word-line for precharging and several n-type transistors per word-line for discharging [22]. Figure B.1 shows the schematic of the NOR decoder in bulk-CMOS. In this figure, the variable *n* represents the number of address bits and the variable *m* indicates the number of address locations. This decoder precharges all of the word-lines and discharges all lines except for the addressed word-line.



$\overline{pre}$
$\overline{pre}\cdot A_{n-1}$
$\overline{pre}\cdot\overline{A}_{n-1}$
$\overline{pre}\cdot A_{n-2}$
$\overline{pre}\cdot\overline{A}_{n-2}$
$\overline{pre}\cdot A_1$
$\overline{pre}\cdot\overline{A}_1$
$\overline{pre}\cdot A_0$
$\overline{pre}\cdot\overline{A}_0$
1
1
1
1
1
d_0
T_{pcg}
T_{pd}
T_{pd}
T_{pd}
T_{pd}
d_{m-1}

Figure B.1. The schematic of the conventional NOR decoder [22]

In addition to the decoder circuitry, the transistors that provide the decoder's functionality by charging and discharging the word-lines, the NOR decoder also requires peripheral circuitry. The peripheral circuitry includes the inverters and logic gates that generate the signals required to control the decoder circuitry; the NOR decoder requires an inverter for the precharge signal to the T_{pcg} transistors as well as eight NOR gates that control the T_{pd} transistors to eliminate short-circuit currents when precharging so power is greatly reduced. The peripheral circuitry consumes less than a third of the power of the NOR decoder [36]. It should be pointed out that the peripheral circuitry will also consume power depending on the address transition and the amount of circuitry used.

Table B.1 and Figure B.2 present six different FinFET-based design schemes of the NOR decoder. These design schemes differ in the front and back gates either being shorted or independent and differ in terms of the input-signal swing. All schemes are minimally-sized (one fin per p-type FinFET and one fin per n-type FinFET), except for the 2X-LP scheme (two fins per p- and n-type FinFET), and have equally-sized p- and n-type FinFETs. Back-gate biases for the LP schemes and high input swing for the SG schemes can be statically driven by a voltage generator circuit [8].

Table B.1. FinFET-based NOR decoder design schemes

Scheme	**Type**	**Inputs to Gate**		***Input Swing***
		Front	***Back***	
Shorted-Gate (SG)	p	prechg	tied	$0 \rightarrow V_{DD}$
	n	A_i	tied	$0 \rightarrow V_{DD}$
Low-Power (LP) (*min-LP* & *2X-LP*)	p	prechg	V_{high}	$0 \rightarrow V_{DD}$
	n	A_i	V_{low}	$0 \rightarrow V_{DD}$
SG-High Pchg/Adr Swing	p	prechg	tied	$0 \rightarrow V_{high}$
	n	A_i	tied	$V_{low} \rightarrow V_{DD}$
SG-High Pchg Swing	p	prechg	tied	$0 \rightarrow V_{high}$
	n	A_i	tied	$0 \rightarrow V_{DD}$
SG-High Adr Swing	p	prechg	tied	$0 \rightarrow V_{DD}$
	n	A_i	tied	$V_{low} \rightarrow V_{DD}$
SG-Max Input Swing	p	prechg	tied	$V_{low} \rightarrow V_{high}$
	n	A_i	tied	$V_{low} \rightarrow V_{high}$

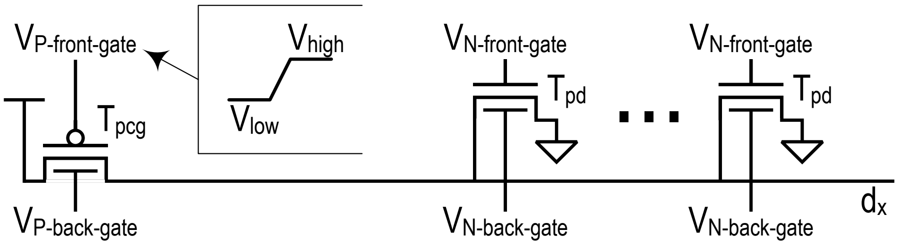


Figure B.2. A word-line of a generic FinFET NOR decoder scheme

B.1.1 Simulation Setup

For this study, Spice3-UFDG (Linux version 3.7) was used to model n- and p-type FinFETs. Spice3-UFDG is a physics-based model which is calibrated to follow predicted results from Synopsys MEDICI and measured results from symmetrical double-gate FinFETs fabricated at Motorola [9] [10]. Table B.2 highlights the chosen parameter values for this research.

Table B.2. FinFET device parameters for NOR address decoders

Parameter	Value
N-Channel Surface Orientation	<110>
Gate length (L_G)	13 nm
Gate to source/drain	2.4 nm (n-type)
underlap (L_{SD})	3.2 nm (p-type)
Fin height (H_{fin})	30 nm
Fin thickness (T_{SI})	7 nm
Oxide thickness (T_{ox})	1 nm
Gate thickness (T_G)	50 nm
Gate work function (Φ_G)	4.47 eV (n-type)
	4.75 eV (p-type)
Low-field mobility for thick T_{SI} (μ_0)	565 cm^2/V-s (n-type)
	250 cm^2/V-s (p-type)
Fin body doping (N_{Body})	10^{15} cm^{-3}
Source/drain doping (N_{DS})	10^{20} cm^{-3}
Source/drain resistance (R_{SD})	170 Ω-μm
Supply voltage (V_{DD})	1 V

In simulations we used a 100 ps (10 GHz) period for the precharge signal so as the decoder is active, practically all current consumption is dynamic and little due to leakage. Table B.3 presents the performance results for the schemes. These results are divided into two categories: dynamic and static/inactive performance.

Table B.3. The performance of the NOR decoder design schemes

Scheme			Shorted-Gate (SG)		min-LP (min-sized)		2X-LP (2xp & 2xn)		SG-High Pchg/Adr Swing		SG-High Pchg Swing		SG-High Adr Swing		SG-Max Input Swing	
Parameter			0.2V	0.4V	0.2V	0.4V	0.2V	0.4V	0.2V	0.4V	0.2V	0.4V	0.2V	0.4V	0.2V	0.4V
I_{dyn} (µA)	1 Decode-Line	Min	0.26	0.26	0.12	0.10	0.24	0.22	0.31	0.34	0.31	0.34	0.30	0.32	0.37	0.48
		Max	3.42	3.42	2.87	2.90	3.56	3.53	3.38	3.31	3.36	3.33	3.42	3.43	3.51	3.87
		Ave	2.91	2.91	2.66	2.66	3.32	3.28	2.97	3.02	2.95	2.97	2.97	2.96	3.17	3.33
	16 Decode-Lines	Ave	46.64	46.64	42.50	42.48	53.08	52.50	47.48	48.36	47.14	47.54	47.45	47.36	50.74	53.27
		Comp%	*9.73*	*9.78*	--	--	*24.90*	*23.58*	*11.70*	*13.83*	*10.92*	*11.91*	*11.64*	*11.48*	*19.37*	*25.39*
I_{lkg} (pA) 16 Decode-Lines		Ave	198.09	198.09	5.71	0.25	11.41	0.51	0.40	0.0008	0.40	0.0008	198.09	198.09	0.40	0.0017
		Comp%	*3371*	*77611*	--	--	*100.00*	*99.98*	*-92.96*	*-99.69*	*-92.96*	*-99.69*	*3371*	*77611*	*-92.95*	*-99.32*
Precharge Delay (ps)		Max	4.04	4.04	8.71	12.28	6.22	7.83	4.55	4.97	4.52	4.93	4.05	4.07	3.51	3.08
		Comp%	*-53.59*	*-67.08*	--	--	*-28.61*	*-36.26*	*-47.74*	*-59.55*	*-48.14*	*-59.86*	*-53.56*	*-66.88*	*-59.75*	*-74.92*
Discharge Delay (ps)		Max	2.67	2.67	6.87	8.83	5.16	6.13	2.69	2.49	2.48	1.93	2.98	2.97	2.43	2.23
		Comp%	-61.07	-69.72	--	--	-24.88	-30.59	-60.83	-71.84	-63.95	-78.16	-56.66	-66.33	-64.54	-74.75
16 Dec-Lines I_{dyn}*Max Delay (µA*ps) [10^{-16}]		Ave	1.89	1.89	3.70	5.22	3.30	4.11	2.16	2.40	2.13	2.34	1.92	1.93	1.78	1.64
		Comp%	-49.07	-63.86	--	--	-10.83	-21.23	-41.62	-53.96	-42.48	-55.08	-48.16	-63.07	-51.95	-68.55
16 Dec-Lines I_{lkg}*Max Delay (pA*ps) [10^{-24}]		Ave	801.12	801.12	49.72	3.13	70.99	3.99	1.83	0.00	1.81	0.00	801.58	806.03	1.41	0.01
		Comp%	1511.	25484	--	--	42.78	27.47	-96.32	-99.88	-96.35	-99.88	1512	25641	-97.16	-99.83

B.1.2 Dynamic Performance: Delay and Current

The reported worst-case precharge delay usually occurs when the word-line is precharged and all four T_{pd} transistors were previously active. The worst-case discharge delay usually occurs when only one T_{pd} transistor discharges the word-line. For all schemes, the maximum delay is the precharge delay since p-type FinFETs have lower current drive than n-type FinFETs. The schemes' delays are compared to the min-LP scheme. Percentage is calculated as:

$$100 \times (Scheme\_delay - minLP\_delay) / minLP\_delay$$

The LP schemes are the most affected by the reverse bias as the precharge delays increase by 40% for the min-LP scheme and by 25% for the 2X-LP scheme with a reverse-bias increase from 0.2 V to 0.4 V. The other schemes have smaller increases in delay due to high input swing except for the SG (which does not use reverse bias) and SG-Maximum Input Swing (that has a small decrease in delay) schemes which also provide the shortest delays.

The minimum, maximum, and average dynamic current consumption are presented for one word-line and the average dynamic current consumption is presented for all sixteen word-lines of the 4-to-16 decoder. A word-line consumes minimum current when it is discharged from being previously selected/addressed since the word-line is already at V_{DD} when the precharge stage begins and thus does not need to precharge. Usually, a word-line consumes maximum current when it is fully precharged and selected/addressed. The current consumption of the entire 4-to-16 decoder is compared to the min-LP scheme. The percentage is calculated as:

$$100 \times (Scheme\_current - minLP\_current) / minLP\_current$$

All of the SG schemes, except the SG-Maximum Input Swing scheme, dissipate approximately the same amount of current. When the input swing increases from 0.2 V to 0.4 V, these schemes generally dissipate slightly more current. This increase in input swing (reverse bias for LP schemes) reduces the current consumption of the min-LP and 2X-LP schemes. The min-LP scheme consumes the least current of all schemes. The 2X-LP scheme significantly reduces the delay of the min-LP scheme, however, it dissipates significantly more current. A sixteen word-line current-delay product (I_{dyn} × Max Delay) shows the best speed and power performance is seen by the SG schemes.

B.1.3 Static/Inactive Performance: Leakage Current

In scaled designs, many 4-to-16 decoders may be used to create a larger decoder for larger memories. In these designs, only one 4-to-16 decoder should be active, the decoder that has an addressed word-line, while all other 4-to-16 decoders should be inactive. However, inactive decoders will still consume power due to leakage currents.

FinFETs offer lower leakage due to better control of the channel. Leakage can be significantly reduced if the back-gates of the decoder’s FinFETs are reverse-biased. Reverse-biasing the back-gates with 0.4 V yields less leakage current than using a reverse-bias of 0.2 V or shorting the front and back gates together. In particular, the SG-High Precharge Swing, SG-High Precharge/Address Swing, and SG-Maximum Input Swing schemes consume the least leakage current. This is due to the reverse-biasing of the precharge p-type FinFET, T_{pcg}, since p-type FinFETs leak more current than n-type FinFETs. Reverse-biasing the precharge p-type FinFET directly results in substantially less leakage since one n-type pull-down is controlled by the most-significant address bits (MSBs) and is always active in order to keep the word-line unaddressed.

B.1.4 Overall Performance

There is no clear best-choice for both dynamic and static/inactive performance. If a system only uses a few, frequently-accessed small memories then the min-LP and SG schemes would likely be good choices since they offer the lowest dynamic power dissipation and leakage power will be negligible. However, if a system requires large memories and many address decoders, then the SG-High Precharge Swing, SG-High Precharge/Address Swing, and SG-Maximum Input Swing schemes may be better choices for lowest power due to the reduction of leakage power without much sacrifice of dynamic power or delay. Additionally, in terms of area, while FinFETs are much smaller than traditional bulk-CMOS transistors, SG schemes have a slight area advantage over LP schemes due to having a much easier wiring layout. Creating a shorted-gate FinFET also requires one less step in the fabrication process, while an independent-gate FinFET requires more area to be wired and must obey design rules about minimum distance between fins.

For scaled performance and process, voltage, and temperature (PVT) variations, we have analyzed the SG, min-LP, and SG-High Precharge Swing schemes. The SG and min-LP schemes are base schemes to compare against the SG-High Precharge Swing scheme, which has a good balance of dynamic current, leakage current, and speed. Plus, this scheme is better than the SG-High Precharge/Address Swing and SG-Maximum Input Swing schemes since it uses one less voltage level and one less voltage generator.

To analyze the scaled performance for 6-to-64 and 8-to-256 decoders, additional T_{pd} n-type FinFETs are added on each word-line to accommodate the additional address bits. The dynamic current increases per word-line as increased capacitance and more short-circuit currents are present with either two or four additional pull-down transistors on each word-line. This increase is greater when moving from four to six address bits (a 10-13% increase) than moving from six to eight address bits (about a 5% increase). However, each scheme's leakage current per word-line is the same for four, six, and eight address bits.

B.1.5 Performance under PVT Variations

Process/parameter, voltage, and temperature (PVT) variations are simulated using a quasi-Monte Carlo (QMC) analysis [20]. We use QMC samples for Monte Carlo simulation to achieve a good spread of data points in fewer simulations than with completely random samples. We used QMC and created 1000 sets of zero-mean and unit-standard deviation Sobol points to allow for completely independent assignment of varied values for all varied device parameters. These points were weighted by each parameter's mean and standard deviation: $X = \mu + x * \sigma$. Similar to other research which analyzes effects of parameter variations on FinFETs, each parameter's standard deviation is estimated to be $3\sigma = 10\% * \mu$ [18]. All parameters in Table B.2 were varied, except for L_{SD}, T_{ox}, Φ_G, and V_{DD} due to time-consuming simulations or convergence issues. Parameters were varied over a temperature range from 0°C to 100°C and over reverse-bias voltage variations of ±5% and ±10% (e.g., a 0.2 V + 5% bias voltage becomes -0.21 V and 1.21 V) at 27°C. Varying the reverse-bias voltage simulates the impact of process variations on accompanying voltage generator circuitry [8].

The three schemes function correctly under PVT variations. Table B.4 summarizes the performance variations. For all schemes' parameters, reverse-bias variation has less of an effect than temperature variation. Temperature and reverse-bias variations have a linear effect on dynamic current and maximum delay, and have an exponential effect on leakage current. Temperature has a small effect on dynamic current (no greater than a 6.7% increase) and a moderate effect on delay (at most a 19.8% increase); but, as anticipated, leakage current is greatly affected by temperature. SG leakage current is least affected by temperature variation, but still sees a factor of 13 increase in leakage (at 100°C) compared to its nominal average at 27°C. Reverse-bias variations have the largest effect on the ±0.4 V-biased min-LP scheme; an 8.4% speedup is seen for a decrease in bias voltage and a 9.9% slowdown is seen for an increase in bias voltage due to constant reverse-biasing regardless of the input signal. Reverse-bias variations affect leakage current the most as more leakage is seen at lower bias voltages and vice versa.

Table B.4. NOR address decoder performance variations due to PVT variations

Parameters and Variations			SG		min-LP				SG-High Precharge Swing			
					0.2V		*0.4V*		*0.2V*		*0.4V*	
			Ave	*%Err**	Ave	*%Err**	Ave	*%Err**	Ave	*%Err**	Ave	*%Err**
Dynamic Current (µA)	*Temp (°C)*	0°	46.4	*-1.1*	42.4	*-0.3*	42.2	*-0.5*	46.5	*-0.9*	46.7	*-1.1*
		27°	46.9	--	42.5	--	42.4	--	46.9	--	47.3	--
		50°	47.6	*1.5*	42.7	*0.4*	42.6	*0.4*	47.7	*1.6*	47.7	*1.0*
		75°	48.5	*3.4*	42.9	*1.1*	42.8	*0.9*	49.0	*4.5*	49.4	*4.4*
		100°	49.4	*5.3*	43.4	*2.1*	43.2	*1.7*	50.1	*6.7*	50.2	*6.2*
	Bias	-10%	--	--	42.5	*-1.9X10^{-3}*	42.4	*-28X10^{-3}*	46.9	*-28X10^{-3}*	47.2	*-120X10^{-3}*
		-5%	--	--	42.5	*0.5X10^{-3}*	42.4	*-3.8X10^{-3}*	46.9	*-21X10^{-3}*	47.2	*-66X10^{-3}*
		5%	--	--	42.5	*1.3X10^{-3}*	42.4	*-34X10^{-3}*	46.9	*26X10^{-3}*	47.3	*55X10^{-3}*
		10%	--	--	42.5	*-8.2X10^{-3}*	42.4	*-81X10^{-3}*	47.0	*46X10^{-3}*	47.3	*98X10^{-3}*
Leakage Current (pA)	*Temp (°C)*	0°	60	*-73*	1.2	*-82*	0.04	*-87*	0.07	*-85*	0.1X10^{-3}	*-89*
		27°	223	--	6.7	--	0.31	--	0.49	--	1.0X10^{-3}	--
		50°	584	*161*	23	*241*	1.34	*335*	1.95	*302*	6.2X10^{-3}	*505*
		75°	1447	*547*	73	*984*	5.34	*1642*	7.28	*1398*	35X10^{-3}	*3279*
		100°	3157	*1312*	199	*2879*	17.9	*5725*	23.1	*4645*	160X10^{-3}	*15126*
	Bias	-10%	--	--	9.2	*37.0*	0.56	*83.0*	0.90	*85.2*	3.5X10^{-3}	*240.1*
		-5%	--	--	7.8	*17.0*	0.41	*35.2*	0.66	*36.1*	1.9X10^{-3}	*84.1*
		5%	--	--	5.7	*-14.5*	0.23	*-25.9*	0.36	*-26.5*	0.6X10^{-3}	*-45.4*
		10%	--	--	4.9	*-26.9*	0.17	*-45.0*	0.26	*-46.0*	0.3X10^{-3}	*-69.8*
Maximum Delay (ps)	*Temp (°C)*	0°	8.7	*-2.3*	8.7	*-2.3*	11.7	*-5.5*	4.34	*-4.0*	4.86	*-1.0*
		27°	8.9	--	8.9	--	12.3	--	4.52	--	4.91	--
		50°	9.5	*5.8*	9.5	*5.8*	13.0	*5.7*	4.56	*1.0*	4.95	*0.9*
		75°	9.8	*10.0*	9.8	*10.0*	13.9	*12.8*	4.64	*2.7*	4.97	*1.3*
		100°	10.4	*16.9*	10.4	*16.9*	14.8	*19.8*	4.72	*4.5*	5.05	*2.8*
	Bias	-10%	--	--	8.8	*-1.6*	11.3	*-8.4*	4.48	*-0.9*	4.83	*-1.7*
		-5%	--	--	8.9	*-0.9*	11.8	*-4.4*	4.50	*-0.4*	4.86	*-0.9*
		5%	--	--	9.0	*1.0*	12.9	*4.7*	4.54	*0.4*	4.95	*0.9*
		10%	--	--	9.1	*2.0*	13.6	*9.9*	4.56	*0.9*	4.99	*1.7*

**%Err* = (Mean – Nominal Mean @ 27°C) / (Nominal Mean @ 27°C) * 100%

B.1.6 Conclusions

Six FinFET-based schemes for the NOR address decoder were presented which differ in front- and back-gate connections and input signal swing. The most promising scheme is the SG-High Precharge Swing scheme, which has the following features:

- *Delay is 48.1% (for 0.2 V reverse-bias) and 59.9% (for 0.4 V reverse-bias) less than the min-LP scheme:* The SG-High Precharge Swing scheme shorts the front and back gates of the FinFETs for increased speed.

- *Static/inactive power is 93.0% (for 0.2 V reverse-bias) and 99.7% (for 0.4 V reverse-bias) less than the min-LP scheme:* The SG-High Precharge Swing scheme reverse biases the front and back gates of the p-type precharge FinFET for greater reduction of leakage current. In addition, leakage power is reduced by over 510 times from 0.2 V to 0.4 V reverse-bias in SG-High Precharge Swing.

- *Dynamic power is only 10.9% (for 0.2 V reverse-bias) and 11.9% (for 0.4 V reverse-bias) more than the min-LP scheme:* The min-LP scheme reduces short-circuit currents more than the SG-High Precharge Swing scheme due to higher transistor impedances by continuously reverse-biasing the back-gates of FinFETs.

- *Temperature has an exponential effect on leakage current and small effect on dynamic current:* The SG-High Prechange Swing scheme has the largest effect; but, its leakage current is still lower than the other schemes.

There are tradeoffs involved in each performance metric. The min-LP and SG schemes have the lowest dynamic power, the SG-High Precharge Swing and SG-High Precharge/Address Swing schemes have the lowest leakage power, and the SG-Maximum Input Swing and SG schemes have the shortest delays. This information can be weighed and can assist in the choice of an address decoder when trying to design an optimal memory module for a system.

B.2 CMOS Memory Address Decoders

This work was originally presented in [40]. Four decoders were designed, simulated, and analyzed using a 65 nm CMOS technology. The next four subsections will discuss the designs and operation of the conventional NOR and the novel AND-NOR, Sense-Amp, and AND decoding schemes.

B.2.1 The Conventional NOR Decoder

The NOR decoder uses one p-type transistor per word-line for precharging and several n-type transistors per word-line for discharging [22]. Figure B.1 shows the schematic of the NOR decoder; for this research, all transistors are minimum-sized. The variables *m* and *n* are used in all decoding scheme schematics where *n* represents the number of address bits and *m* indicates the number of address locations. This decoder precharges all of the word-lines and discharges all lines except for the addressed word-line. This leads to high power consumption as all but one word-line are needlessly charged and discharged.

In addition to the decoder circuitry, the transistors that provide the decoder's functionality by charging and discharging the word-lines, the NOR decoder also requires peripheral circuitry. Peripheral circuitry includes the inverters and logic gates that generate the signals required to control the decoder circuitry. The NOR decoder's peripheral circuitry includes an inverter for the precharge signal to the T_{pcg} transistors as well as eight NOR gates that control the T_{pd} transistors to eliminate current dissipation when precharging so power consumption is greatly reduced.

This decoder is designed using a 65 nm CMOS technology and has a minimum delay of 120 ps. This decoder was simulated using a 120 ps (8.33 GHz) precharge-evaluate cycle based off the precharge signal with equal stage lengths of 60 ps. In the precharge stage the word-lines begin to charge. The NOR decoder uses T_{pcg} on each word line to nonselectively precharge all word-lines [22]. This scheme's current consumption is directly dependent on the precharge stage length; a shorter precharge stage yields lower current consumption as the T_{pcg} transistors are active for a shorter amount of time. In the evaluate stage the T_{pd} pull-downs discharge all non-addressed word-lines using all address bits. Figure B.3 shows a simulation of a word-line of the NOR decoder.

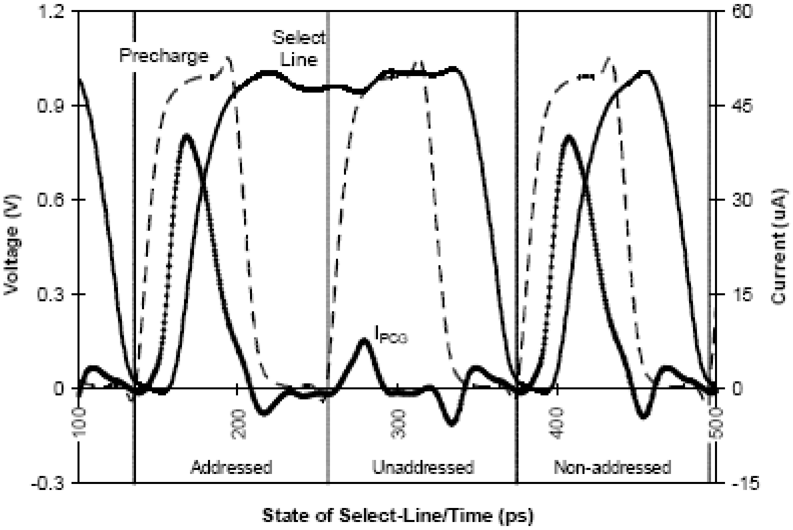


Figure B.3. A simulation of the NOR decoder

The current consumption of a NOR decoder word-line is measured as the current drawn by T_{pcg}. Each T_{pcg} on each word-line draws current since no selective precharging is used. For a 4-to-16 decoder, as was designed and tested in this study, 15 of the 16 word-lines (lines not previously addressed) draw current when they are precharged from ground to V_{DD}. The previously addressed line remains charged at V_{DD}, however, and requires significantly less current. Every word-line also dissipates short-circuit current when entering the precharge stage as shown by Figure B.4. This dissipation's magnitude is dictated by the address; as more T_{pd} transistors become active, the lower path resistance results in a greater current. The total current consumed by the conventional decoding scheme is the sum of the decoder current of each word-line, e.g. I_{PCG}, and the current drawn by the peripheral circuitry. Peripheral current consumption depends on the address transition and the amount of peripheral circuitry in the scheme.

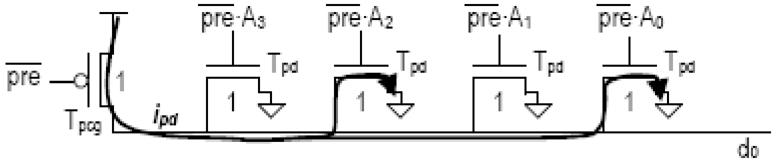

$\overline{pre}\cdot A_3$
$\overline{pre}\cdot A_2$
$\overline{pre}\cdot A_1$
$\overline{pre}\cdot A_0$
$\overline{pre}$
T_{pcg}
T_{pd}
T_{pd}
T_{pd}
T_{pd}
i_{pd}
1
1
1
1
1
d_0

Figure B.4. The short-circuit currents of the NOR decoder [22]

B.2.2 The AND-NOR Decoder

The novel AND-NOR decoder is a selective approach which only charges and discharges one-quarter of the word-lines each cycle. The most significant bits of the address are used to select the portion of the decoder that will be charged. This design choice benefits from the principle of locality as changes in the least-significant bits (LSBs) of the address are expected to be more frequent than changes in most-significant bits (MSBs) of the address. The AND-NOR decoder design is also similar to the basic selective precharge approach of [38] to precharge the match lines in a content-addressable memory. The basic form of the AND-NOR decoding scheme is shown in Figure B.5.

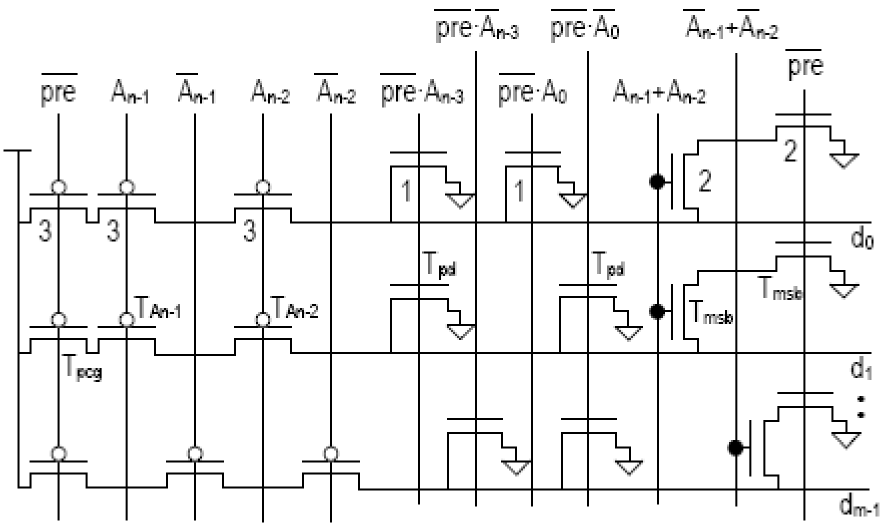


Figure B.5. The schematic of the AND-NOR decoder

This decoder has a minimum delay of 122 ps and is simulated using a 122 ps (8.20 GHz) precharge-evaluate cycle with equal stage lengths of 61 ps. This is the same cycle, albeit at a slightly slower frequency, as the NOR decoder. A simulation of the AND-NOR decoder's operation is provided in Figure B.6. In the first cycle of the simulation, the word-line is charged and addressed. During the second cycle, the word-line is discharged and unaddressed. The word-line is charged and discharged in the third cycle when it has identical MSBs as the addressed word-line (selected MSBs). Lastly in the fourth cycle the word-line is not charged when it has different MSBs than the address (non-selected MSBs). This case highlights the energy savings by not unnecessarily charging and discharging non-addressed word-lines.

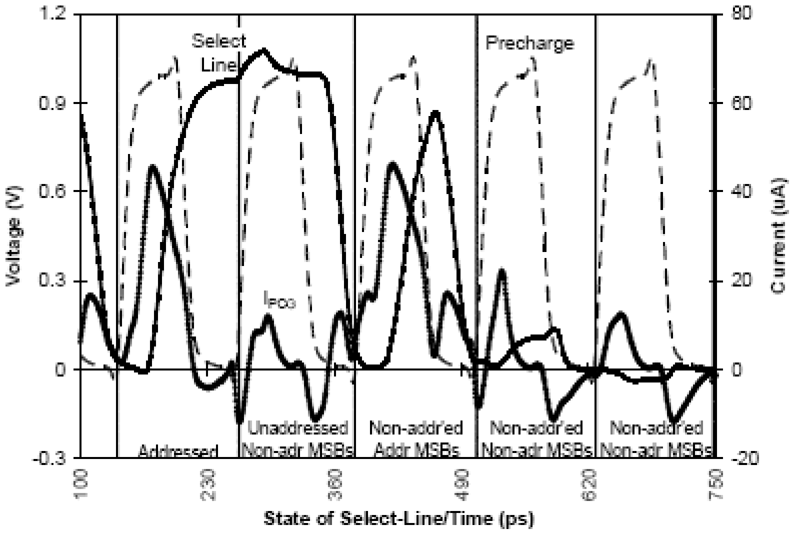


Figure B.6. A simulation of the AND-NOR decoder

In the precharge stage the AND-NOR decoder uses T_{pcg}, T_{An-1}, and T_{An-2} on each word-line to selectively precharge the lines based on the two address MSBs. This requires only one-quarter of the word-lines, the quarter containing the requested address, to be precharged each cycle. In the evaluate stage the T_{pd} transistors discharge all non-addressed word-lines. Figure B.7 shows short-circuit current dissipation between the precharge and evaluate stages. The AND-NOR decoder uses the two address LSBs to discharge and must also be able to discharge via the MSBs with T_{msb} if only the address MSBs change as shown by Figure B.8.

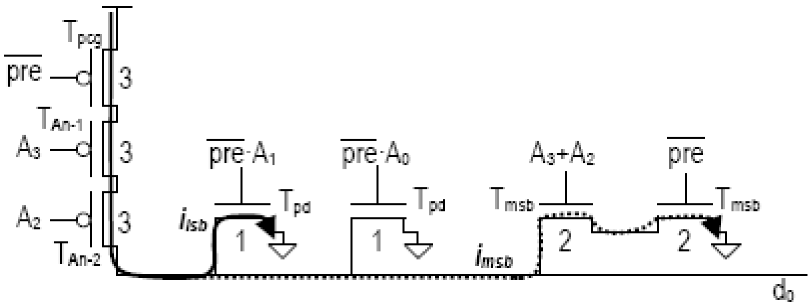


Figure B.7. The short-circuit currents of the AND-NOR decoder when the address has the same MSBs (for word-line d_0, A_3 is 0 and A_2 is 0)

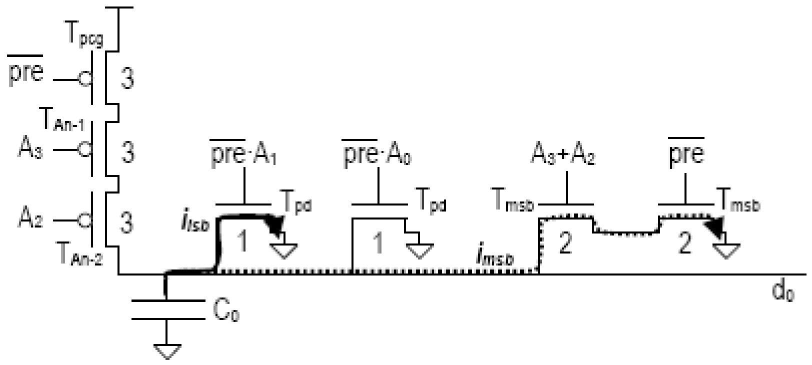


Figure B.8. The pull-down currents of a previously addressed word-line of the AND-NOR decoder when the address has different MSBs (for word-line d_0, A_3 and A_2 are not both 0)

In terms of peripheral circuitry, the AND-NOR decoder requires four NOR gates to control all T_{pd} transistors to eliminate current dissipation in the precharge stage. The AND-NOR decoder also requires four NAND gates to control the T_{msb} transistors and a strong inverter for precharge as the T_{pcg} and T_{msb} transistors are a large load for the inverted signal.

B.2.3 The Sense-Amp Decoder

The Sense-Amp decoder senses when the word-line is being charged and only allows for the addressed line to be fully charged. This novel scheme is shown in Figure B.9. Similar to the AND-NOR decoder, only one-quarter of the word-lines will charge, however, the word-lines need not be fully charged during the precharge stage since the sense-amplifier on the addressed word-line will charge the line up to V_{DD}. Therefore this decoder can have a shorter precharge stage length and will consume less power than the AND-NOR scheme since only the addressed word-line is fully precharged.

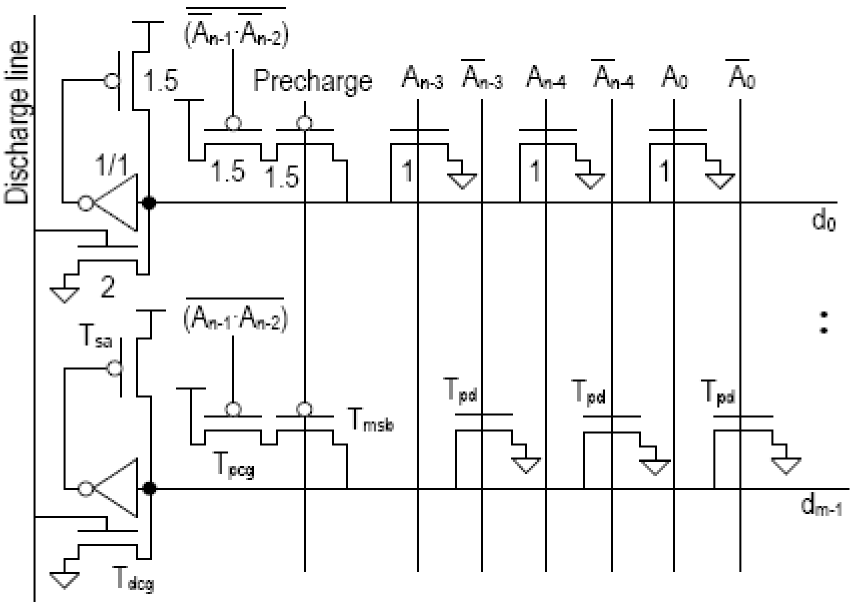


Figure B.9. The schematic of the Sense-Amp decoder

The Sense-Amp decoder has a minimum delay of 114 ps and is simulated using a 114 ps (8.77 GHz) discharge-precharge-evaluate cycle based off the precharge and discharge signals with equal stage lengths of 38 ps. In the first third of the cycle all word-lines are discharged. At the second third of the cycle one-quarter of the word-lines are precharged. During the final third of the cycle the sense-amplifier on the addressed word-line will charge the addressed line up to V_{DD} while the other charged non-addressed lines will discharge.

Figure B.10 and Figure B.11 illustrate the short-circuit currents of the Sense-Amp decoder and a diagram of the operation of this decoder is provided in Figure B.12. During the first cycle of the simulation the word-line is charged and selected. In the second cycle of the simulation the word-line is discharged and deselected. The third cycle illustrates the word-line being charged and discharged when it has identical MSBs as the accessed address. The fourth cycle shows the word-line not being charged when it has different MSBs than the addressed word-line.

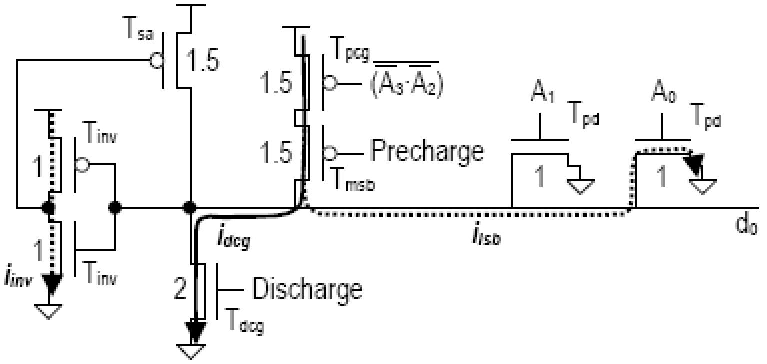


Figure B.10. The short-circuit currents of the Sense-Amp decoder when the word-line was not previously addressed

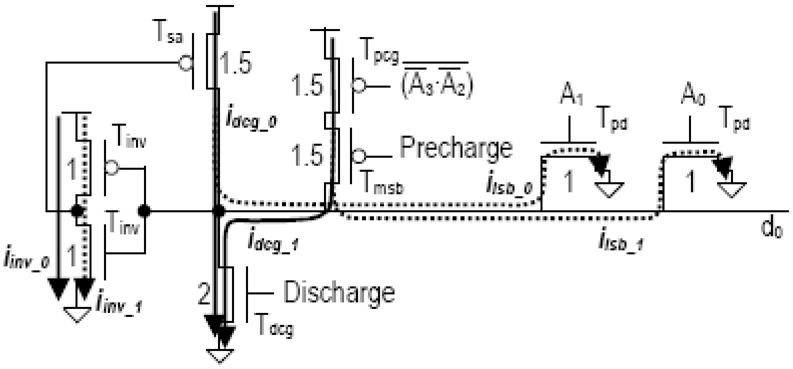


Figure B.11. The short-circuit currents of the Sense-Amp decoder when the word-line was previously addressed

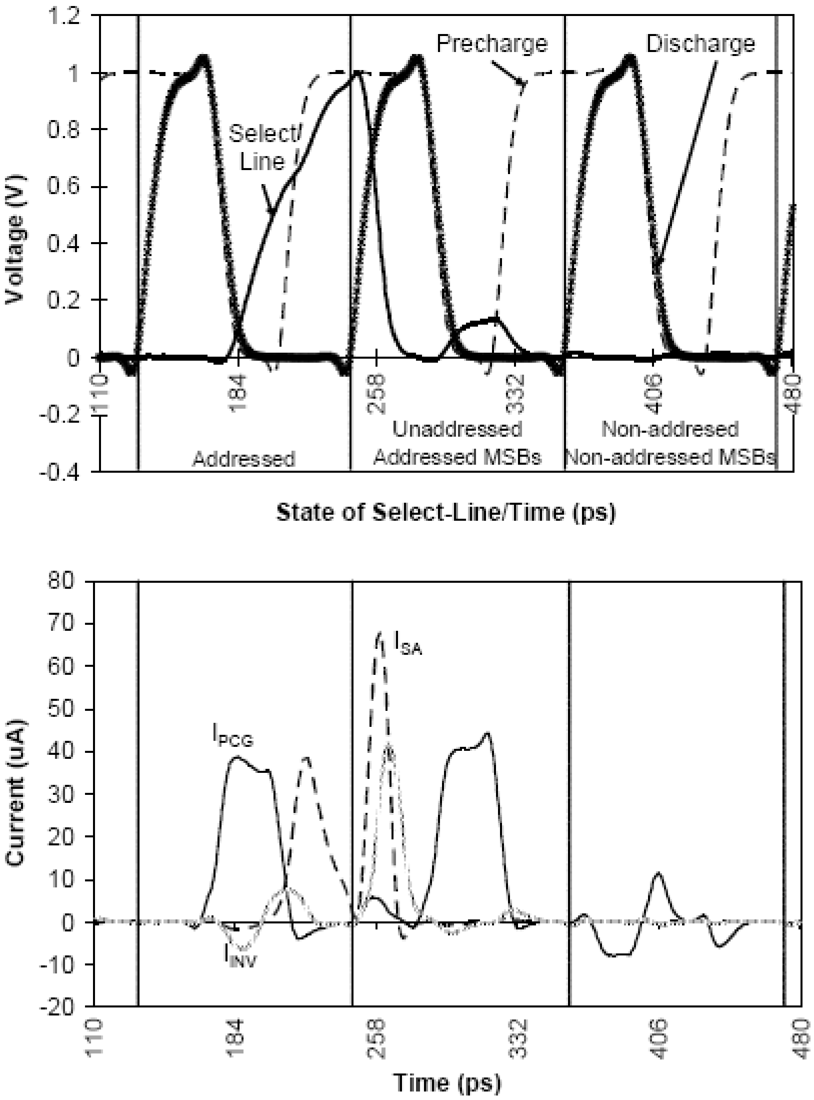


Figure B.12. A simulation of the Sense-Amp decoder

In the discharge stage T_{dcg} grounds all word-lines and only the previously addressed word-line is pulled from V_{DD} to ground. The precharge stage is very short and lines with selected MSBs are only partially precharged (at less than V_{DD}). The design uses T_{pcg} and T_{msb} on each line for selective precharging. In the evaluate stage the addressed line activates its sense-amplifier to charge the line to V_{DD} while all other word-line sense-amplifiers remain inactive and the lines discharge.

The Sense-Amp decoder requires four NAND gates to control the T_{msb} transistors. Instead of using more peripheral NOR gates, the address lines directly control the T_{pd} transistors resulting in current dissipation. Since the precharge stage is short (38 ps), the current dissipation is limited. Additionally, strong inverters are not needed for the precharge and discharge signals since the signals are not inverted.

There are three currents which are pulled from the supply: the precharge pull-up current at T_{pcg} (I_{PCG}), the sense-amplifier controlled pull-up current at T_{sa} (I_{SA}), and the sense-amplifier current at T_{inv} (I_{INV}). These currents are consumed at specific times during each discharge-precharge-evaluate cycle due to switching and short-circuits. Figure B.10 illustrates the short-circuit currents of a decoder word-line not previously addressed. In this situation, the short-circuit current, i_{dcg}, is always dissipated when a word-line enters the precharge stage with selected MSBs. While in the precharge stage the three non-addressed word-lines which have selected MSBs dissipate current as shown by i_{lsb}. The addressed word-line dissipates the short-circuit current, i_{inv}, as the sense-amplifier's output changes to make T_{sa} active. Figure B.11 displays the short-circuit currents of a previously addressed decoder word-line. As the discharge stage is entered, the charged line dissipates i_{dcg_0} until it is pulled down close enough to ground that the sense-amplifier sets T_{sa} inactive. Additionally if the LSBs of the address have changed, i_{lsb_0} will also be dissipated, resulting in a faster pull-down of the word-line. When the line's sense-amplifier changes its output, the short-circuit current i_{inv_0}, is dissipated. Afterwards, the short-circuit current, i_{dcg_1}, is always dissipated if the word-line enters the precharge stage with selected MSBs. While in the precharge stage, if this line is one of the three unaddressed word-lines which have selected MSBs, it will dissipate current as shown by i_{lsb_1}. If this word-line is readdressed, the short-circuit current, i_{inv_1}, is dissipated as the sense-amplifier's output activates T_{sa}.

B.2.4 The AND Decoder

The AND decoder is a simpler design which uses negative logic via a NAND gate per word-line to produce inverted, intermediate word-line signals. Each intermediate signal is then inverted to drive each word-line or the inverter can be incorporated into a buffer or pipeline register to drive the corresponding word-line. Figure B.13 shows the novel AND decoder schematic. This decoder does not use any clocking signals, therefore the delay of this decoder is due to the worst-case delay of the AND gate. This decoder can operate, and is simulated, at a minimum delay of 111 ps (9.01 GHz).

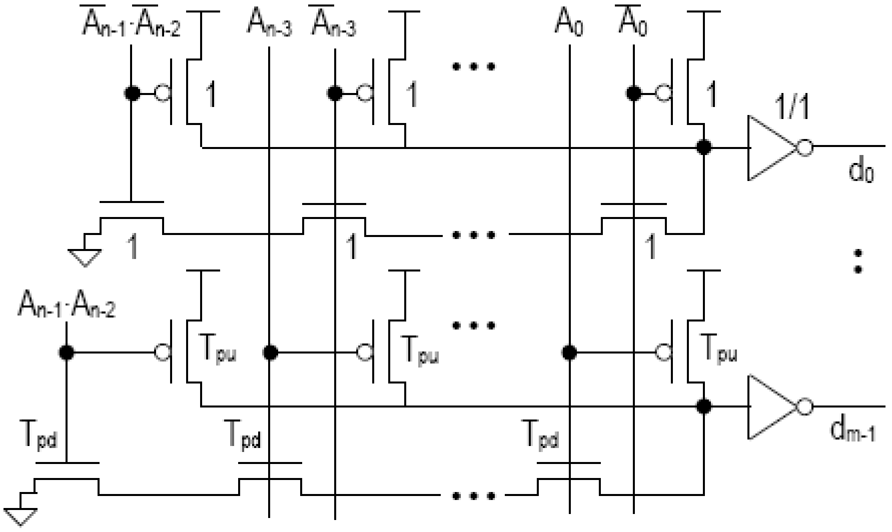


Figure B.13. The schematic of the AND decoder

If an AND decoder word-line is addressed, all of the T_{pd} transistors will be active and the intermediate word-line signal will be discharged. This causes the inverter to charge the line to signal it was selected. If the word-line is deselected or not addressed, then at least one T_{pd} transistor will be inactive and at least one T_{pu} transistor will be active. The intermediate word-line signal is charged to V_{DD} and the word-line output is discharged signifying the line is not selected. Figure B.14 shows a simulation of a word-line of the AND decoder. From the simulation it is observed that current is drawn from the supply by each T_{pu} (I_{MSBs}, I_{A0}, and I_{A1}) and the inverter (I_{INV}). The greatest current consumption is seen for changes in many address bits due to short-circuit and switching currents throughout the design, including the peripheral circuitry.

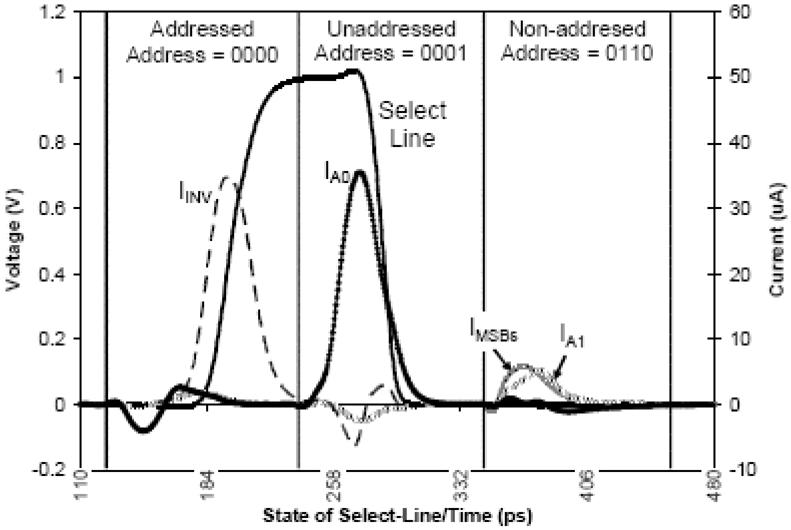


Figure B.14. A simulation of the AND decoder

The AND decoder uses four NOR gates to create the “and” expressions of the MSBs, e.g. $A_{n-1} \cdot A_{n-2}$. Four inverters also invert both the address MSBs for use by these NOR gates and address LSBs for use by the T_{pd} and T_{pu} transistors. Since no clocking signals are used, the AND decoder does not need to invert a clock and thus, similar to the Sense-Amp decoder, saves an inverter over the NOR decoding scheme.

The current consumption of an AND decoder word-line is measured as the sum of the currents drawn by the T_{pu} transistors plus the current drawn by the word-line inverter. The T_{pu} currents are only drawn when a word-line is deselected. Thus for any decoder only one word-line consumes this current for a change in address bits. The inverter current is consumed by two word-lines per change of address due to one line being deselected and another being selected. In addition, address transitions cause multiple word-lines to dissipate short-circuit currents; a change of more address bits cause more word-lines to experience short-circuit currents. The AND decoder is the most selective scheme as the unaddressed word-lines are neither fully nor partially precharged unlike the NOR, AND-NOR, and Sense-Amp decoding schemes.

B.2.5 Simulation Environment

The Cadence Virtuoso Schematic Editor with a 65 nm CMOS technology was used to create the NOR, AND-NOR, Sense-Amp, and the AND 4-to-16 decoders. Loading was simulated on the word-lines by adding inverters with four times the minimum p- and n-type transistor widths on each line. This simulates sufficient loading in order to drive a pipeline register as used in high-performance pipelined memories [41] [42]. Interconnects are not taken into account in these schematics, however, the distances between transistors in the design and pipeline registers is assumed to be small, which greatly reduces the impact of the physical characteristics of interconnects.

Spectre was used to perform extensive circuit simulations in a 65 nm CMOS technology using pre-layout data. The average current consumption for one period of each decoder's cycle was calculated for significant address transitions; these transitions are listed in first column of Table B.5. Note that the total current consumed by each decoding scheme is not limited to only the decoder current, e.g. I_{PCG} for the word-lines of the NOR and AND-NOR decoders, but the current drawn by the peripheral circuitry must also be considered. The transistor counts of all four schemes are shown in Table B.6. The table separates the scheme circuitry into two categories: decoder circuitry and peripheral circuitry. Decoder circuitry includes the transistors that provide the functionality of each decoder, namely charging and discharging word-lines. The peripheral circuitry includes the inverters and gates required to control the decoder circuitry. The consumed peripheral current depends on the address transition and the amount of peripheral circuitry in the scheme. The current consumption of the decoder and peripheral circuitry was computed separately to provide better insight on the consumption patterns of each decoding scheme.

Table B.5. Current consumption of 4-to-16 decoding schemes

Address Transition	Sch.	Ave. Current (μA)			Scheme Current Percentages		Delay (ps)	Energy		EDP[b] (fJxps)
		Dec.	*Periph.*	*Total*	*Dec. (%)*	*Periph. (%)*		*Total (fJ)*	*% Less than NOR[a]*	
0000→0000 *no change*	NOR	181	121	302	60	40	120	36	—	4348
	A-N	104	149	253	41	59	122	31	15	3772
	S-A	72	8	80	90	10	114	9	75	1043
	AND	1	4	5	19	81	111	1	99	56
0000→0011 *change LSBs*	NOR	177	127	305	58	42	120	37	—	4389
	A-N	104	165	270	39	61	122	33	10	4012
	S-A	71	9	80	89	11	114	9	75	1044
	AND	75	6	80	93	7	111	9	76	988
0011→0000 *change LSBs*	NOR	180	136	316	57	43	120	38	—	4548
	A-N	105	172	277	38	62	122	34	11	4121
	S-A	72	35	107	67	33	114	12	68	1385
	AND	50	50	100	50	50	111	11	71	1231
0000→1100 *change MSBs*	NOR	177	126	303	59	41	120	36	—	4360
	A-N	146	178	324	45	55	122	40	-9	4825
	S-A	80	34	114	70	30	114	13	64	1487
	AND	23	31	53	42	58	111	6	84	659
1100→0000 *change MSBs*	NOR	180	132	312	58	42	120	37	—	4489
	A-N	139	219	357	39	61	122	44	-17	5319
	S-A	79	42	120	66	34	114	14	63	1566
	AND	36	48	84	43	57	111	9	75	1037
0000→1111 *change all bits same dir.*	NOR	175	139	314	56	44	120	38	—	4527
	A-N	144	199	343	42	58	122	42	-11	5109
	S-A	84	36	120	70	30	114	14	64	1555
	AND	62	33	95	66	34	111	11	72	1169
1111→0000 *change all bits same dir.*	NOR	178	153	332	54	46	120	40	—	4777
	A-N	139	250	389	36	64	122	47	-19	5788
	S-A	80	68	148	54	46	114	17	58	1918
	AND	43	94	137	31	69	111	15	62	1686
0011→1100 *change all bits diff. dir.*	NOR	176	142	318	55	45	120	38	—	4574
	A-N	145	195	339	43	57	122	41	-9	5051
	S-A	80	57	138	58	42	114	16	59	1788
	AND	61	74	135	45	55	111	15	61	1663
1100→0011 *change all bits diff. dir.*	NOR	176	140	316	56	44	120	38	—	4549
	A-N	137	236	373	37	63	122	46	-20	5555
	S-A	74	41	115	64	36	114	13	65	1496
	AND	80	48	128	63	37	111	14	63	1575
Average	NOR	178	135	313	57	43	120	38	—	4507
	A-N	129	196	325	40	60	122	40	-6	4839
	S-A	77	37	114	68	32	114	13	66	1476
	AND	48	43	91	53	47	111	10	73	1118
Minimum	NOR	175	121	302	58	40	120	36	—	4348
	A-N	104	149	253	41	59	122	31	-20	3772
	S-A	71	8	80	89	10	114	9	58	1043
	AND	1	4	5	19	81	111	1	61	56
Maximum	NOR	181	153	332	55	46	120	40	—	4777
	A-N	146	250	389	38	64	122	47	15	5788
	S-A	84	68	148	57	46	114	17	75	1918
	AND	80	94	137	59	69	111	15	99	1686

[a] The energy percent differences are calculated using: *[(NOR_Val – Dec_Val)/NOR_Val] * 100%*

[b] EDP: Energy-Delay Product

Table B.6. 4-to-16 decoder transistor counts

Dec.	Decoder Circuitry			Peripheral Circuitry			Total
	Pchg	Dchg	S-A* or Inv.$^{+}$	Inv.	Col. NOR	Col. NAND	
NOR	1 X 16	4 X 16	–	2 X 5	4 X 8	–	122
A-N	3 X 16	4 X 16	–	2 X 5	4 X 4	4 X 4	154
S-A	2 X 16	2 X 16	4 X 16*	2 X 4	–	4 X 4	152
AND	3 X 16	3 X 16	2 X 16^{+}	2 X 4	4 X 4	–	152

The average power dissipation is calculated by multiplying the decoder and peripheral current consumption by the supply voltage whereas in [43] it was calculated by multiplying the total current by each word-line voltage. This research uses a 1.0 V supply voltage with the 65 nm CMOS technology. The decoder energy dissipation is computed by multiplying the average power dissipation by the decoder delay.

B.2.6 Decoding Scheme Performance

All four decoding schemes are compared with regard to decoder current consumption, peripheral current consumption, and total current, power, and energy consumption. The simulated data for all of these metrics is provided in Table B.5, which includes data of the average current consumption and energy dissipation without parameter variations; simulations are performed using pre-layout circuits.

B.2.6.1 Decoder Current Consumption

It can be observed from Table B.5 that the selective precharge schemes outperform the nonselective NOR decoder in terms of average decoder current for the listed address transitions. This is due to the AND-NOR and Sense-Amp decoders precharging only one-quarter of the word-lines and the AND decoder only charging the addressed word-line. The NOR decoder consumes decoder current uniformly for all address transitions since the design gives no selective preference to MSBs or LSBs during precharging or discharging. The AND-NOR decoder consumes decoder current uniformly for address LSB transitions. The AND-NOR decoder consumes more decoder current for address MSB transitions; especially when the address MSBs transition 00→11. This is due to the switching delay of the peripheral circuitry. As the address MSBs transition, glitches can occur on the word-lines as they are temporarily precharged due to the difference in delay between an inverted and non-inverted MSB input.

Both the Sense-Amp and AND decoders consume noticeably less decoder current the NOR and AND-NOR decoders for all of the address transitions. While the Sense-Amp decoder is equally selective when precharging as the AND-NOR decoder, it is less impacted by precharge glitching caused by switching delay in the peripheral circuitry. Though the control inputs to the T_{pcg} transistors are more delayed in the Sense-Amp decoder due to the use of peripheral NAND gates for combining inverted and non-inverted address MSBs, the precharge stage is not the first stage of the decoding cycle and these delays have little effect on the word-lines. The discharge stage occurs first during the decoding cycle, and while the word-lines are discharging, the NAND outputs which control the T_{pcg} transistors become stable. Nonetheless, similar to the AND-NOR decoder, the Sense-Amp decoder consumes the most current when the address MSBs transition 00→11.

The AND decoder only charges one word-line during the decode cycle and consumes the least amount of decoder current for a majority of the address transitions. The AND decoder consumes the most decoder current when the address LSBs transition 00→11. It is observed that the AND decoder consumes more decoder current than the Sense-Amp decoder for address transitions 0000→0011 and 1100→0011. This is due to glitching of the control inputs to the T_{pd}

transistors caused by delays in the inverted and non-inverted address LSBs. These delays can cause temporary short-circuit currents if all of the T_{pd} transistors are temporarily active and at least one T_{pu} transistor is temporarily active due to this peripheral circuitry delay.

B.2.6.2 Peripheral Current Consumption

The NOR decoder also displays approximately uniform peripheral current consumption with greatest consumption occurring when all of the address bits transition 1111→0000. This uniformity is caused by the four NOR gates which, regardless of the address transition, switch their outputs 0→1 when the evaluate stage is entered. The AND-NOR decoder requires the most peripheral current and for changes in the address MSBs the AND-NOR decoder consumes the most total current. This is due to the large amount of peripheral circuitry in the design: two NAND gates switch output 0→1 when entering the evaluate stage to control the T_{pd} transistors, one NAND switches output 0→1 to control T_{msb} if a change in MSBs occur, and there is a large amount of switching current required by the precharge signal inverter due to the high output load. The large amount of peripheral circuitry actually consumes more current than the decoder circuitry.

The Sense-Amp and the AND decoders have significantly less peripheral circuitry than the NOR and AND-NOR decoders and realize large energy gains here. The Sense-Amp decoder uses the least amount of peripheral circuitry compared to the other two decoders and consumes the least peripheral current. At most, one NAND gate switches its output 0→1 if a change in address MSBs occurs. The large reduction in peripheral circuitry reduces the impact of this circuitry on the overall Sense-Amp current consumption. The AND decoder requires similar peripheral circuitry to the Sense-Amp decoder, with the exception that NOR gates are used instead of NAND gates to control one pair of T_{pu} and T_{pd} transistors on one-quarter of the word-lines. At most, one NOR gate switches its output 0→1 if a change in address MSBs occurs, however, the p- type transistors in the NOR gate must be sized greater in order to sufficiently drive a load twice that of the Sense-Amp decoder's NAND gates. For most of the address transitions, the AND decoder consumes more peripheral current than decoder current, but this is mostly due to its very low decoder current consumption.

B.2.6.3 Total Current, Power, and Energy

Not all of the selective decoding schemes outperform the NOR decoder in terms of average total current and average power (the supply voltage for each scheme is 1.0 V). Figure B.15 shows that NOR and AND-NOR decoders have similar current consumptions. It should be noticed that the NOR decoder has a very consistent current consumption for all address transitions. The AND-NOR decoder consumes more total current, power, and energy than the NOR decoder for address MSB transitions. On the other end, the AND decoder has the lowest current consumption in all transitions except one (1100→0011). Figure B.16 summarizes the energy dissipation and delays of all four decoding schemes.

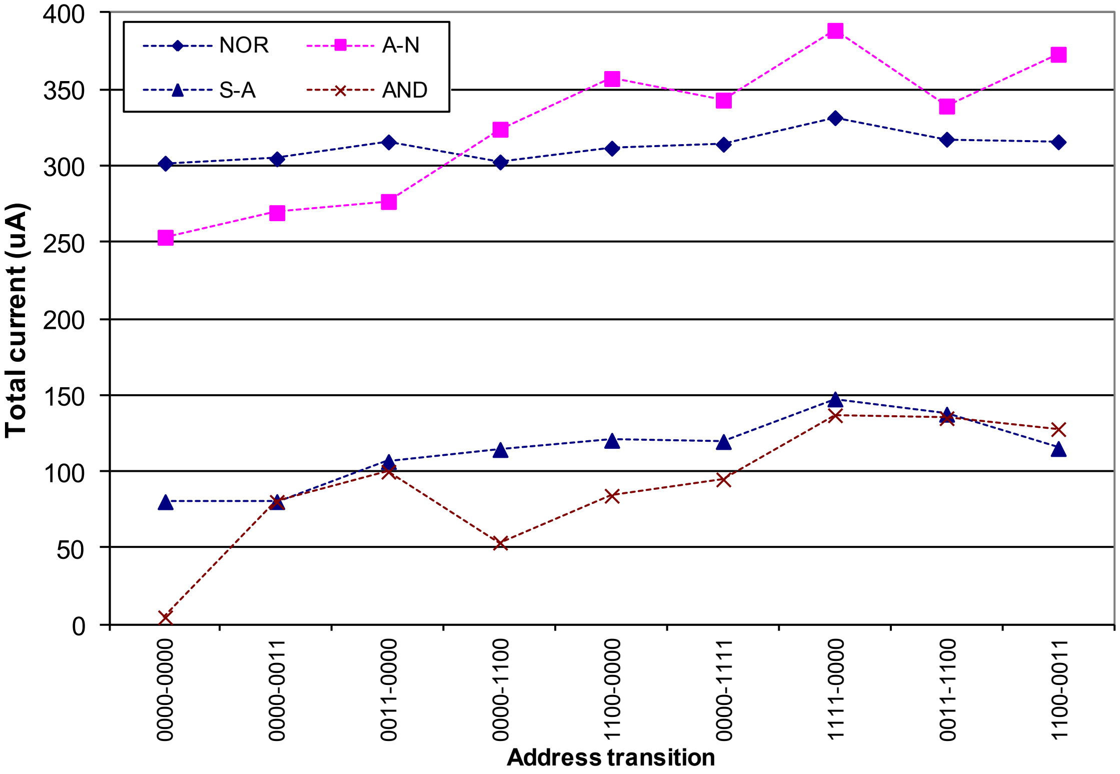


Figure B.15. The total current consumption under different address transitions for each decoder

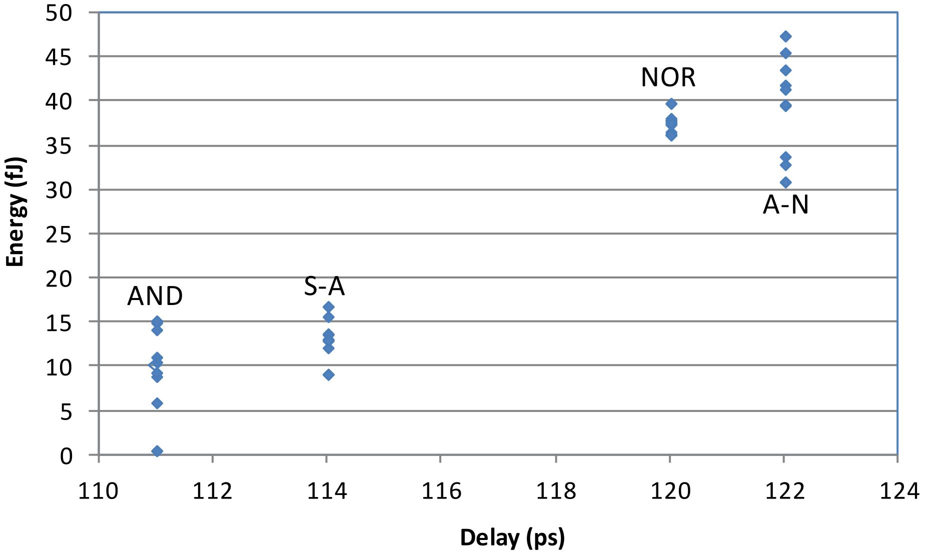


Figure B.16. A graph showing the delays and address transition energy dissipation for each decoder

B.2.7 Performance with Parameter Variations

In this section the impact of parameter variations on delays and energies of the four decoders is quantitatively evaluated. In the circuit simulations, parameters of the n- and p-type transistors, polysilicon, active region, and gate oxide are varied to determine each decoder's susceptibility to process variations. Each corner is in the form of N-type – P-type transistor speed. These corners impact the oxide thickness, threshold voltage, resistivity, electron/carrier mobilities, junction capacitance, and diode current to simulate fast or slow transistors. The corners that impact the polysilicon, active region, and gate oxide speeds manipulate their capacitances for high or low speed.

Table B.7 shows the performance of the decoders in terms of delay and energy under different parameter variations. All decoders operate correctly for each corner tested. Energy consumption varies within approximately 4% of the nominal energy consumption for all schemes, but the decoder delays are most affected by parameter variations; this can observed in Figure B.17. The Sense-Amp decoder has the greatest decrease in delay (24%) for fast transistor speeds. On the other hand the AND decoder has the largest increase in delay (28%) for slow transistor speeds. It should be pointed out that different corner variations tend to affect the decoder schemes in a similar way with only few exceptions. For instance, the decoder delays deviate from the nominal by 20% to 24% in the fast-fast corner and from -25% to -28% in the slow-slow corner as shown in Table B.7. The only exception in this trend is the slow-fast corner where the AND-NOR decoder has an increase in delay; this is due to the larger number of n-type transistors in series. The AND and Sense-Amp decoders perform best in terms of energy; the worst energies for these decoders are 16 and 18 fJ, respectively. These energy consumptions are significantly smaller than the other two decoder energies.

Table B.7. Summary of parameter variation effects on 4-to-16 decoders

Corner ***speed N – speed P***	**Sch.**	**Delay (ps)**	**Delay diff. Nom (%)**[a]	**Ave. Energy (fJ)**	**Ave. Energy diff. Nom (%)**[a]	**Worst-case 1111→0000 Energy (fJ)**
Fast-Fast[b] FETs and com. param.	NOR	96	20	38	-1	40
	A-N	98	20	38	3	47
	S-A	87	24	13	0	17
	AND	87	22	10	0	15
Fast-Fast	NOR	110	8	37	1	39
	A-N	110	10	39	1	47
	S-A	99	13	13	2	16
	AND	99	11	10	1	15
Slow-Fast	NOR	118	2	39	-3	41
	A-N	124	-2	41	-3	49
	S-A	105	8	13	-4	18
	AND	107	4	10	-2	15
Nominal *Typ.-Typ.*	NOR	120	–	38	–	40
	A-N	122	–	40	–	47
	S-A	114	–	13	–	17
	AND	111	–	10	–	15
Fast-Slow	NOR	122	-2	36	3	39
	A-N	128	-5	39	2	47
	S-A	120	-5	12	4	16
	AND	117	-5	10	2	15
Slow-Slow	NOR	132	-10	38	-2	40
	A-N	134	-10	40	-1	48
	S-A	129	-13	13	-3	17
	AND	125	-13	10	-1	15
Slow-Slow[b] FETs and com. param.	NOR	150	-25	38	-2	40
	A-N	152	-25	41	-4	49
	S-A	144	-26	14	-4	18
	AND	142	-28	10	-3	16

[a] The delay and energy percent differences are calculated using the following formula:

*[(Nominal_Val – Corner_Val)/Nominal_Val] * 100%*

[b] In addition to modifying the n- and p-type transistor parameters, these fast-fast and slow-slow corners also influence the speeds of the parameters common to both transistor types (the polysilicon, active region, and gate oxide) by manipulating their capacitances.

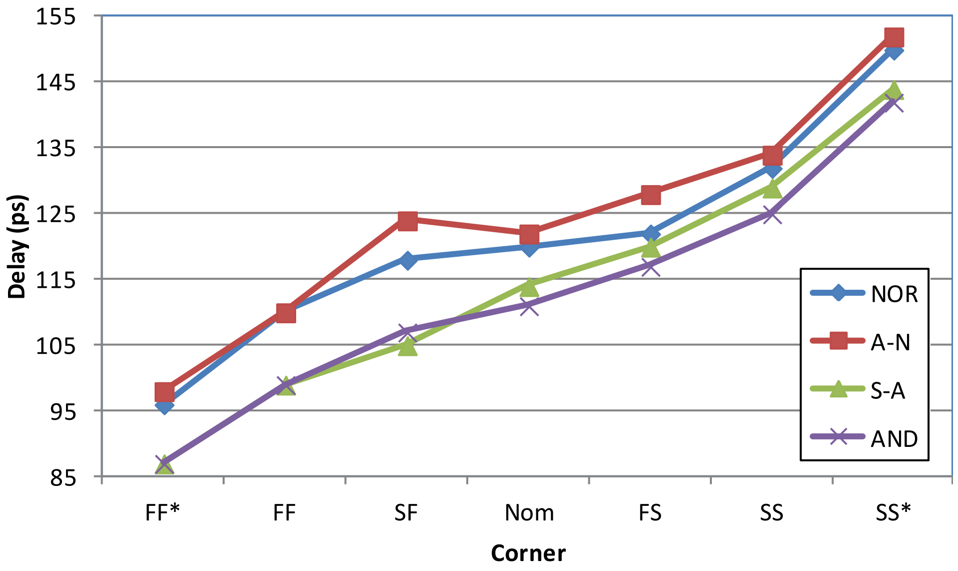


Figure B.17. A graph showing the impact of parameter variations (via corner simulations) on decoder delay. The FF* and SS* corners influence the speeds of parameters common to both n- and p-type transistors as explained in footnote b of Table B.7.

B.2.8 Decoder Scalability

To examine the scalability of the NOR, AND-NOR, Sense-Amp, and AND decoders, the 4-to-16 decoders are scaled to 8-to-256 decoders. The 8-to-256 decoder designs use a pre-decoder which decodes address bits 7-4 and sixteen 4-to-16 decoders which operate using address bits 3-0. The pre-decoder activates only one 4-to-16 decoder depending on bits 7-4 of the address by acting as the precharge signal for all sixteen 4-to-16 decoders. For the NOR, AND-NOR, and Sense-Amp 8-to-256 designs, the pre-decoder consists of two two-input NAND gates that combine address bits 7-6 and 5-4 and are "OR'ed" with the precharge signal. Note that the designs of the NOR and AND-NOR decoders required modification since the decoders are more selectively precharged depending on address bits 7-4. If an address change in only these bits occur, then a previously addressed word-line will not discharge unless additional pull-down circuitry is included. An additional n-type pull-down transistor is added to each word-line and each transistor is controlled by an "OR'ing" of address bits 7-4 to alleviate this issue. Two scaled designs of the AND decoder are presented: a four-input AND design (AND4) requiring the additional input to be controlled by the "AND'ing" of address bits 7-4 and a five-input AND design (AND5) requiring the two additional inputs to be controlled by address bit 7 "AND'ed" with bit 6 and bit 5 "AND'ed" with bit 4.

A summary of the delays and energy dissipations of these designs are presented in Table B.8. The AND4 design performs best as it has the least delay and lowest minimum and maximum energies. It is interesting to study the distribution of energy consumption between the pre-decoder, sixteen 4-to-16 decoders, and the accompanying sixteen sets of peripheral logic. The pre-decoders for the NOR and AND-NOR schemes consume the least percentage of total energy, but this is confounded with overall higher current consumption of the decoder and peripheral circuitry for these schemes compared to that of the Sense-Amp and AND decoders. The Sense-Amp pre-decoder consumes a high percentage of total energy due to fact that two control signals must be driven to a large load: all sixteen 4-to-16 decoders. The AND4 and AND5 pre-decoders consume less of a percentage of total energy since no global precharge or discharge signals must be driven to the decoders; only the "AND'ed" 7-4 bits must be driven to all decoders.

Table B.8. 8-to-256 decoder delay and energy summary

Dec.	Delay (ps)	Energy (fJ)		Energy Distribution Percentages (%)		
		Min.	Max.	Pre-Decoder	16 4-to-16 Decoders	16 sets of Peripheral Circuitry
NOR	230	400	460	21	41	39
A-N	160	670	690	14	58	29
S-A	150	370	410	66	18	17
AND4	130	170	190	24	35	41
AND5	130	170	300	37	30	33

Other pre-decoder schemes could be used to scale the 4-to-16 decoders. Multi-stage designs, where the pre-decoder can be a 4-to-16 decoder, could be used as another hierarchical design technique. Or larger decoders could be designed using MSBs for selective precharging. For example, if a 32 or 64-bit address is used, various numbers of MSBs could be used for selective precharging. Using the two MSBs can still result in only 25% of the word-lines to precharge, but using additional MSBs can result in greater potential energy savings. Using three or four MSBs can result in only 12.5% or 6.25% of the word-lines to precharge, respectively. However, there are limiting returns when using additional MSBs. For each additional MSB used for selective precharging, additional peripheral circuitry is required to control the decoder circuitry and the energy consumed by this additional peripheral circuitry can outweigh the energy reduction gained by more-selective precharging.

B.2.9 Conclusions

Three novel decoding schemes have been presented and compared to the conventional NOR decoder. Of these three, the AND and Sense-Amp decoders are shown to have less delay and energy dissipation than the conventional NOR decoder. These two schemes feature the following comparisons with the NOR decoder.

- *The AND decoder consumes between 61% and 99% less energy (73% on average) and the Sense-Amp decoder consumes between 58% and 75% less energy (66% on average) than the NOR decoder:* The proposed schemes save energy by using selective precharging. The AND decoder is most selective and charges only the addressed word-line and the Sense-Amp decoder precharges only a quarter of the word-lines. The NOR decoder uses a great amount of energy since it precharges all word-lines and discharges all lines except for the addressed word-line.

- *The AND decoder is 7.5% and the Sense-Amp decoder is 5.0% faster than the NOR decoder:* Both schemes quickly charge the addressed word-line and discharge all other lines thus enabling these decoders to be operated at higher frequencies. The AND decoder quickly charges the addressed word-line using a complementary CMOS design. The Sense-Amp quickly charges the addressed word-line by using a short precharge stage and a sense-amplifier during the evaluate stage, thus charging the addressed word-line for two-thirds of the clock period.

The energy and delay savings do not come without tradeoffs though. The NOR decoder has a very simple implementation which uses fewer transistors than the proposed methods. For systems that can devote only minimal area to address decoding, the NOR decoder may be a better choice if the proposed decoders are inadequate. However, the proposed decoding schemes are good choices for high-performance low-power designs due to the greatly decreased energy dissipation and reduced delay and these schemes could be used in conjunction with pre-decoders for scaling to large memory address decoders [34].

Appendix C

Complete Data for FinFET SRAM Low-Leakage Modifications

This appendix contains tables containing complete data for FinFET SRAM low-leakage modifications using header and footer transistors. This data was gathered for both a full-V_{DD} of 1 V and near-threshold operation at 0.6 V. Grayed-out rows indicate that convergence errors did not allow for the particular cell's performance data to be gathered.

Table C.1. 6T leakage, delay, and EDP 1 V V_{DD} results of headers/footers per cell

	Scheme *Pass; Inv-N; Inv-P; Read*	**Leak. Tran.**	**Leak. Tran. Config.**	**Leak. / Cell (nA)**	**Leakage / Cell in Sleep-Mode (nA)**		**Leakage EDP / Array in Sleep-Mode (ps×fJ)**		**# Arrays for Break-Even EDP**[+]	**32 bits × 1024 words Array**					
										Delay (ps)				**Ave* EDP (ps×fJ)**	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	5	--	120	--
		Header	t	0.592	0.536	*0.91*	18.0	*0.91*	*1*	32	*1*	5	*1*	98	*0.82*
		Footer	t	0.592	0.283	*0.48*	18.0	*0.90*	*35*	44	*1.38*	4	*0.80*	184	*1.53*
		Hdr + Ftr	t	0.592	0.227	*0.38*	14.4	*0.72*	*9*	44	*1.38*	5	*1*	162	*1.35*
		Header	vdd	0.592	0.533	*0.90*	17.9	*0.90*	*1*	32	*1*	6	*1.20*	88	*0.73*
		Footer	0	0.592	0.253	*0.43*	24.2	*1.22*	--	54	*1.69*	4	*0.80*	253	*2.11*
		Hdr + Ftr	vdd / 0	0.592	0.194	*0.33*	18.5	*0.93*	*78*	54	*1.69*	6	*1.20*	221	*1.84*
		Header	+	0.592	0.504	*0.85*	16.9	*0.85*	*1*	32	*1*	6	*1.20*	86	*0.72*
		Footer	-	0.592	0.156	*0.26*	18.4	*0.93*	*122*	60	*1.88*	4	*0.80*	296	*2.47*
		Hdr + Ftr	+ / -	0.592	0.014	*0.02*	1.7	*0.08*	*9*	60	*1.88*	6	*1.20*	260	*2.17*
	LP -; -; +	Nom.		0.021	0.021	--	4.9	--		84	--	6	--	103	--
		Header	t	0.021	0.021	*1*	4.9	*1*	*1*	84	*1*	6	*1*	93	*0.90*
		Footer	t	0.021	0.020	*0.95*	7.5	*1.55*	--	107	*1.27*	6	*1*	143	*1.39*
		Hdr + Ftr	t	0.021	0.020	*0.95*	7.4	*1.52*	--	106	*1.26*	6	*1*	133	*1.29*
		Header	vdd	0.021	0.021	*1*	4.9	*1*	*1*	84	*1*	7	*1.17*	88	*0.85*
		Footer	0	0.021	0.020	*0.95*	8.7	*1.79*	--	115	*1.37*	6	*1*	158	*1.53*
		Hdr + Ftr	vdd / 0	0.021	0.020	*0.95*	8.5	*1.75*	--	114	*1.36*	7	*1.17*	141	*1.37*
		Header	+	0.021	0.019	*0.90*	4.4	*0.90*	*1*	84	*1*	7	*1.17*	87	*0.84*
		Footer	-	0.021	0.011	*0.52*	4.9	*1.02*	--	117	*1.39*	6	*1*	163	*1.58*
		Hdr + Ftr	+ / -	0.021	0.008	*0.38*	3.6	*0.74*	*35*	117	*1.39*	7	*1.17*	145	*1.41*

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

[+] Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table C.2. 6T noise margin and energy 1 V V_{DD} results of headers/footers per cell

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	# Arrays for Break-Even EDP$^+$	32 bits × 1024 words Array						SNM (mV)		RSNM (mV)	
					Energy (fJ)									
					Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*	Val	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.			2.4	--	9.3	--	3.8	--	360	--	124	--
		Header	t	*1*	2.3	*0.96*	6.3	*0.68*	3.1	*0.82*	343	*0.95*	87	*0.70*
		Footer	t	*35*	3.3	*1.38*	7.7	*0.83*	4.2	*1.11*	357	*0.99*	36	*0.29*
		Hdr + Ftr	t	*9*	3.1	*1.29*	5.8	*0.62*	3.7	*0.97*	343	*0.95*	65	*0.52*
		Header	vdd	*1*	2.2	*0.92*	5.1	*0.55*	2.8	*0.74*	336	*0.93*	74	*0.60*
		Footer	0	--	3.9	*1.63*	7.8	*0.84*	4.7	*1.24*	356	*0.99*	63	*0.51*
		Hdr + Ftr	vdd / 0	*78*	3.8	*1.58*	5.4	*0.58*	4.1	*1.08*	336	*0.93*	118	*0.95*
		Header	+	*1*	2.2	*0.92*	4.9	*0.53*	2.7	*0.71*	334	*0.93*	71	*0.57*
		Footer	-	*122*	4.2	*1.75*	7.9	*0.85*	4.9	*1.29*	356	*0.99*	69	*0.56*
		Hdr + Ftr	+ / -	*9*	4.1	*1.71*	5.3	*0.57*	4.3	*1.13*	335	*0.93*	131	*1.06*
	LP -; -; +	Nom.			0.5	--	3.9	--	1.2	--	442	--	222	--
		Header	t	*1*	0.6	*1.20*	3.3	*0.85*	1.1	*0.92*	439	*0.99*	220	*0.99*
		Footer	t	--	0.7	*1.40*	3.8	*0.97*	1.4	*1.17*	442	*1*	189	*0.85*
		Hdr + Ftr	t	--	0.8	*1.60*	3.2	*0.82*	1.3	*1.08*	439	*0.99*	186	*0.84*
		Header	vdd	*1*	0.5	*1*	3.1	*0.79*	1.1	*0.92*	438	*0.99*	218	*0.98*
		Footer	0	--	0.8	*1.60*	3.8	*0.97*	1.4	*1.17*	442	*1*	182	*0.82*
		Hdr + Ftr	vdd / 0	--	0.8	*1.60*	3.1	*0.79*	1.2	*1*	439	*0.99*	177	*0.80*
		Header	+	*1*	0.5	*1*	3.1	*0.79*	1.0	*0.83*	438	*0.99*	218	*0.98*
		Footer	-	--	0.8	*1.60*	3.7	*0.95*	1.4	*1.17*	442	*1*	179	*0.81*
		Hdr + Ftr	+ / -	*35*	0.8	*1.60*	3.0	*0.77*	1.2	*1*	438	*0.99*	175	*0.79*

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

$^+$ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table C.3. 8T leakage, delay, and EDP 1 V V_{DD} results of headers/footers per cell

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP+	32 bits × 1024 words Array					
										Delay (ps)				Ave* EDP (ps×fJ)	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
8T	SG 3 x t; t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	3	--	66	--
		Header	t	0.592	0.536	*0.91*	18.0	*0.91*	*1*	32	*1*	4	*1.33*	54	*0.82*
		Footer	t	0.592	0.283	*0.48*	18.0	*0.90*	*33*	44	*1.38*	3	*1*	127	*1.92*
		Hdr + Ftr	t	0.592	0.227	*0.38*	14.4	*0.72*	*10*	44	*1.38*	4	*1.33*	115	*1.74*
		Header	vdd	0.592	0.533	*0.90*	17.9	*0.90*	*1*	32	*1*	5	*1.67*	51	*0.77*
		Footer	0	0.591	0.253	*0.43*	23.3	*1.17*	--	53	*1.66*	3	*1*	184	*2.79*
		Hdr + Ftr	vdd / 0	0.591	0.194	*0.33*	17.9	*0.90*	*52*	53	*1.66*	6	*2.00*	168	*2.55*
		Header	+	0.592	0.504	*0.85*	16.9	*0.85*	*1*	32	*1*	6	*2.00*	50	*0.76*
		Footer	-	0.591	0.102	*0.17*	11.6	*0.59*	*20*	59	*1.84*	3	*1*	219	*3.32*
		Hdr + Ftr	+ / -	0.591	0.014	*0.02*	1.6	*0.08*	*9*	59	*1.84*	6	*2.00*	201	*3.05*
	LP_SGR -; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	8	--	39	--
		Header	t	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	9	*1.13*	34	*0.87*
		Footer	t	0.021	0.020	*0.95*	1.3	*1.80*	--	44	*1.38*	8	*1*	72	*1.85*
		Hdr + Ftr	t	0.021	0.020	*0.95*	1.3	*1.80*	--	44	*1.38*	9	*1.13*	67	*1.72*
		Header	vdd	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	10	*1.25*	33	*0.85*
		Footer	0	0.021	0.020	*0.95*	1.8	*2.61*	--	53	*1.66*	8	*1*	101	*2.59*
		Hdr + Ftr	vdd / 0	0.021	0.020	*0.95*	1.8	*2.61*	--	53	*1.66*	10	*1.25*	94	*2.41*
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	32	*1*	10	*1.25*	33	*0.85*
		Footer	-	0.021	0.011	*0.52*	1.3	*1.78*	--	59	*1.84*	8	*1*	117	*3.00*
		Hdr + Ftr	+ / -	0.021	0.008	*0.38*	0.9	*1.30*	--	59	*1.84*	10	*1.25*	109	*2.79*
	LP_INV1 t; -; vdd; t	Nom.		0.096	0.096	--	3.2	--		32	--	4	--	30	--
		Header	t	0.096	0.050	*0.52*	1.7	*0.52*	*1*	32	*1*	5	*1.25*	27	*0.90*
		Footer	t	0.096	0.095	*0.99*	6.0	*1.87*	--	44	*1.38*	4	*1*	61	*2.03*
		Hdr + Ftr	t	0.096	0.049	*0.51*	3.1	*0.97*	*259*	44	*1.38*	5	*1.25*	59	*1.97*
		Header	vdd	0.096	0.048	*0.50*	1.6	*0.50*	*1*	32	*1*	6	*1.50*	26	*0.87*
		Footer	0	0.096	0.095	*0.99*	8.7	*2.71*	--	53	*1.66*	4	*1*	88	*2.93*
		Hdr + Ftr	vdd / 0	0.096	0.046	*0.48*	4.2	*1.31*	--	53	*1.66*	6	*1.50*	85	*2.83*
		Header	+	0.096	0.020	*0.21*	0.7	*0.21*	*1*	32	*1*	6	*1.50*	26	*0.87*
		Footer	-	0.096	0.085	*0.89*	9.7	*3.01*	--	59	*1.84*	4	*1*	105	*3.50*
		Hdr + Ftr	+ / -	0.096	0.009	*0.09*	1.0	*0.32*	*33*	59	*1.84*	6	*1.50*	100	*3.33*
	LP_INV1.2 t; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	5	--	26	--
		Header	t	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	5	*1*	23	*0.88*
		Footer	t	0.021	0.020	*0.95*	1.3	*1.80*	--	44	*1.38*	5	*1*	55	*2.12*
		Hdr + Ftr	t	0.021	0.020	*0.95*	1.3	*1.80*	--	44	*1.38*	5	*1*	52	*2.00*
		Header	vdd	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	6	*1.20*	23	*0.88*
		Footer	0	0.021	0.020	*0.95*	1.8	*2.61*	--	53	*1.66*	5	*1*	80	*3.08*
		Hdr + Ftr	vdd / 0	0.021	0.020	*0.95*	1.8	*2.61*	--	53	*1.66*	6	*1.20*	76	*2.92*
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	32	*1*	6	*1.20*	23	*0.88*
		Footer	-	0.021	0.011	*0.52*	1.3	*1.78*	--	59	*1.84*	5	*1*	94	*3.62*
		Hdr + Ftr	+ / -	0.021	0.008	*0.38*	0.9	*1.30*	--	59	*1.84*	6	*1.20*	90	*3.46*

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

+ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table C.4. 8T SNM and energy 1 V V_{DD} results of headers/footers per cell

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	# Arrays for Break-Even EDP$^+$	32 bits × 1024 words Array						SNM (mV)	
					Energy (fJ)							
					Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*	Val	*Comp*
8T	SG 3 x t; t; t; t	Nom.			0.9	--	6.9	--	2.1	--	360	--
		Header	t	*1*	0.9	*1*	5.1	*0.74*	1.7	*0.81*	343	*0.95*
		Footer	t	*33*	2.0	*2.22*	6.6	*0.96*	2.9	*1.38*	357	*0.99*
		Hdr + Ftr	t	*10*	1.9	*2.11*	5.4	*0.78*	2.6	*1.24*	343	*0.95*
		Header	vdd	*1*	0.9	*1*	4.6	*0.67*	1.6	*0.76*	336	*0.93*
		Footer	0	--	2.6	*2.89*	6.8	*0.99*	3.5	*1.67*	356	*0.99*
		Hdr + Ftr	vdd / 0	*52*	2.6	*2.89*	5.3	*0.77*	3.2	*1.52*	336	*0.93*
		Header	+	*1*	0.8	*0.89*	4.5	*0.65*	1.6	*0.76*	334	*0.93*
		Footer	-	*20*	2.9	*3.22*	6.9	*1*	3.7	*1.76*	356	*0.99*
		Hdr + Ftr	+ / -	*9*	2.9	*3.22*	5.4	*0.78*	3.4	*1.62*	335	*0.93*
	LP_SGR -; -; +; t	Nom.			0.3	--	5.0	--	1.2	--	442	--
		Header	t	*1*	0.3	*1*	4.3	*0.86*	1.1	*0.92*	439	*0.99*
		Footer	t	--	0.8	*2.67*	4.9	*0.98*	1.7	*1.42*	442	*1*
		Hdr + Ftr	t	--	0.8	*2.67*	4.3	*0.86*	1.5	*1.25*	439	*0.99*
		Header	vdd	*1*	0.3	*1*	4.2	*0.84*	1.0	*0.83*	438	*0.99*
		Footer	0	--	1.1	*3.67*	4.9	*0.98*	1.9	*1.58*	442	*1*
		Hdr + Ftr	vdd / 0	--	1.2	*4.00*	4.2	*0.84*	1.8	*1.50*	439	*0.99*
		Header	+	*1*	0.3	*1*	4.1	*0.82*	1.0	*0.83*	438	*0.99*
		Footer	-	--	1.3	*4.33*	4.9	*0.98*	2.0	*1.67*	442	*1*
		Hdr + Ftr	+ / -	--	1.3	*4.33*	4.1	*0.82*	1.9	*1.58*	438	*0.99*
	LP_INV1 t; -; vdd; t	Nom.			0.4	--	3.2	--	0.9	--	443	--
		Header	t	*1*	0.3	*0.75*	2.9	*0.91*	0.9	*1*	441	*1*
		Footer	t	--	1.0	*2.50*	3.1	*0.97*	1.4	*1.56*	441	*1*
		Hdr + Ftr	t	*259*	0.9	*2.25*	2.9	*0.91*	1.3	*1.44*	439	*0.99*
		Header	vdd	*1*	0.3	*0.75*	2.8	*0.88*	0.8	*0.89*	441	*1*
		Footer	0	--	1.3	*3.25*	3.1	*0.97*	1.7	*1.89*	440	*0.99*
		Hdr + Ftr	vdd / 0	--	1.3	*3.25*	2.8	*0.88*	1.6	*1.78*	438	*0.99*
		Header	+	*1*	0.3	*0.75*	2.8	*0.88*	0.8	*0.89*	440	*0.99*
		Footer	-	--	1.4	*3.50*	3.2	*1*	1.8	*2.00*	440	*0.99*
		Hdr + Ftr	+ / -	*33*	1.4	*3.50*	2.8	*0.88*	1.7	*1.89*	438	*0.99*
	LP_INV1.2 t; -; +; t	Nom.			0.3	--	3.0	--	0.8	--	442	--
		Header	t	*1*	0.3	*1*	2.6	*0.87*	0.7	*0.88*	439	*0.99*
		Footer	t	--	0.8	*2.67*	2.9	*0.97*	1.3	*1.63*	442	*1*
		Hdr + Ftr	t	--	0.8	*2.67*	2.6	*0.87*	1.2	*1.50*	439	*0.99*
		Header	vdd	*1*	0.3	*1*	2.6	*0.87*	0.7	*0.88*	438	*0.99*
		Footer	0	--	1.1	*3.67*	2.9	*0.97*	1.5	*1.88*	442	*1*
		Hdr + Ftr	vdd / 0	--	1.2	*4.00*	2.6	*0.87*	1.4	*1.75*	439	*0.99*
		Header	+	*1*	0.3	*1*	2.6	*0.87*	0.7	*0.88*	438	*0.99*
		Footer	-	--	1.3	*4.33*	2.9	*0.97*	1.6	*2.00*	442	*1*
		Hdr + Ftr	+ / -	--	1.3	*4.33*	2.6	*0.87*	1.5	*1.88*	438	*0.99*

Table C.5. Leakage, delay, and EDP 0.6 V V_{DD} results of headers/footers per cell

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP$^+$	32 bits × 1024 words Array					
										Delay (ps)				Ave* EDP (ps×fJ)	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.499	0.499	--	25.5	--		51	--	5	--	57	--
		Header	t	0.499	0.455	*0.91*	23.3	*0.91*	*1*	51	*1*	6	*1.20*	52	*0.91*
		Footer	t	0.499	0.255	*0.51*	26.0	*1.02*	--	72	*1.41*	5	*1*	105	*1.84*
		Hdr + Ftr	t	0.499	0.211	*0.42*	21.5	*0.84*	*8*	72	*1.41*	6	*1.20*	99	*1.74*
		Header	vdd	0.499	0.423	*0.85*	21.6	*0.85*	*1*	51	*1*	41	*8.20*	44	*0.77*
		Footer	0	0.499	0.229	*0.46*	90.8	*3.56*	--	142	*2.78*	5	*1*	318	*5.58*
		Hdr + Ftr	vdd / 0	0.499	0.153	*0.31*	59.8	*2.34*	--	141	*2.76*	41	*8.20*	301	*5.28*
		Header	+	0.499	0.423	*0.85*	20.8	*0.81*	*13*	50	*0.98*	107	*21.40*	151	*2.65*
		Footer	-	0.499	0.133	*0.27*	171.4	*6.72*	--	256	*5.02*	5	*1.00*	872	*15.30*
		Hdr + Ftr	+ / -	0.499	0.011	*0.02*	14.1	*0.55*	*43*	255	*5.00*	108	*21.60*	842	*14.77*
8T	SG 3 x t; t; t; t	Nom.		0.498	0.498	--	24.5	--		50	--	4	--	43	--
		Header	t	0.498	0.455	*0.91*	22.4	*0.91*	*1*	50	*1*	6	*1.50*	40	*0.93*
		Footer	t	0.498	0.255	*0.51*	25.3	*1.03*	--	71	*1.42*	4	*1*	84	*1.95*
		Hdr + Ftr	t	0.498	0.211	*0.42*	20.9	*0.85*	*8*	71	*1.42*	6	*1.50*	81	*1.88*
		Header	vdd	0.498	0.423	*0.85*	20.8	*0.85*	*1*	50	*1*	54	*13.50*	40	*0.93*
		Footer	0	0.498	0.229	*0.46*	83.3	*3.40*	--	136	*2.72*	4	*1*	270	*6.28*
		Hdr + Ftr	vdd / 0	0.498	0.153	*0.31*	55.6	*2.27*	--	136	*2.72*	54	*13.50*	255	*5.93*
		Header	+	0.498	0.423	*0.85*	20.8	*0.85*	*26*	50	*1*	129	*32.25*	192	*4.47*
		Footer	-	0.498	0.088	*0.18*	99.7	*4.07*	--	240	*4.80*	4	*1*	736	*17.12*
		Hdr + Ftr	+ / -	0.497	0.012	*0.02*	13.6	*0.56*	*38*	240	*4.80*	131	*32.75*	703	*16.35*
	LP_SGR -; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	32	--	13	--
		Header	t	0.018	0.018	*1*	0.9	*1*	*1*	50	*1*	35	*1.09*	12	*0.92*
		Footer	t	0.018	0.017	*0.94*	1.7	*1.90*	--	71	*1.42*	32	*1*	23	*1.77*
		Hdr + Ftr	t	0.018	0.017	*0.94*	1.7	*1.90*	--	71	*1.42*	35	*1.09*	23	*1.77*
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*34*	50	*1*	77	*2.41*	21	*1.62*
		Footer	0	0.018	0.017	*0.94*	6.3	*7.09*	--	137	*2.74*	32	*1*	56	*4.31*
		Hdr + Ftr	vdd / 0	0.018	0.014	*0.78*	5.2	*5.84*	--	137	*2.74*	77	*2.41*	55	*4.23*
		Header	+	0.018	0.015	*0.83*	0.7	*0.83*	*258*	50	*1*	204	*6.38*	76	*5.85*
		Footer	-	0.018	0.010	*0.56*	11.6	*13.12*	--	243	*4.86*	31	*0.97*	115	*8.85*
		Hdr + Ftr	+ / -	0.018	0.007	*0.39*	8.1	*9.19*	--	243	*4.86*	203	*6.34*	110	*8.46*
	LP_INV1 t; -; vdd; t	Nom.		0.081	0.081	--	4.0	--		50	--	10	--	14	--
		Header	t	0.081	0.045	*0.56*	2.2	*0.56*	*1*	50	*1*	12	*1.20*	13	*0.93*
		Footer	t	0.081	0.080	*0.99*	7.9	*1.99*	--	71	*1.42*	10	*1*	26	*1.86*
		Hdr + Ftr	t	0.081	0.045	*0.56*	4.5	*1.12*	--	71	*1.42*	12	*1.20*	26	*1.86*
		Header	vdd	0.081	0.015	*0.19*	0.7	*0.19*	*1*	50	*1*	41	*4.10*	11	*0.79*
		Footer	0	0.081	0.080	*0.99*	29.5	*7.41*	--	137	*2.74*	10	*1*	75	*5.36*
		Hdr + Ftr	vdd / 0	0.081	0.014	*0.17*	5.2	*1.30*	--	137	*2.74*	42	*4.20*	73	*5.21*
		Header	+	0.081	0.015	*0.19*	0.7	*0.19*	*5*	50	*1*	103	*10.30*	35	*2.50*
		Footer	-	0.081	0.072	*0.89*	84.3	*21.17*	--	244	*4.88*	10	*1*	180	*12.86*
		Hdr + Ftr	+ / -	0.081	0.007	*0.09*	8.1	*2.04*	--	243	*4.86*	104	*10.40*	176	*12.57*
	LP_INV1.2 t; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	18	--	9	--
		Header	t	0.018	0.018	*1*	0.9	*1*	*1*	50	*1*	20	*1.11*	9	*1*
		Footer	t	0.018	0.017	*0.94*	1.7	*1.90*	--	71	*1.42*	18	*1*	18	*2.00*
		Hdr + Ftr	t	0.018	0.017	*0.94*	1.7	*1.90*	--	71	*1.42*	21	*1.17*	18	*2.00*
		Header	vdd	0.018	0.015	*1*	0.7	*0.83*	*1*	50	*1*	43	*2.39*	8	*0.89*
		Footer	0	0.018	0.017	*0.94*	6.3	*7.09*	--	137	*2.74*	18	*1*	47	*5.22*
		Hdr + Ftr	vdd / 0	0.018	0.014	*0.78*	5.2	*5.84*	--	137	*2.74*	44	*2.44*	46	*5.11*
		Header	+	0.018	0.015	*0.83*	0.7	*0.83*	*46*	50	*1*	98	*5.44*	20	*2.22*
		Footer	-	0.018	0.010	*0.56*	11.6	*13.12*	--	243	*4.86*	18	*1*	99	*11.00*
		Hdr + Ftr	+ / -	0.018	0.007	*0.39*	8.1	*9.19*	--	243	*4.86*	99	*5.50*	96	*10.67*

Table C.6. Noise margin and energy 0.6 V V_{DD} results of headers/footers per cell

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	# Arrays for Break-Even EDP^{+}	32 bits × 1024 words Array						SNM (mV)		RSNM (mV)	
					Energy (fJ)									
					Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*	Val	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.			0.9	--	2.1	--	1.1	--	248	--	100	--
		Header	t	1	0.8	0.89	1.8	0.86	1.0	0.91	243	0.98	94	0.94
		Footer	t	--	1.2	1.33	2.3	1.10	1.4	1.27	247	1	57	0.57
		Hdr + Ftr	t	8	1.2	1.33	1.9	0.90	1.4	1.27	243	0.98	51	0.51
		Header	vdd	1	0.8	0.89	1.2	0.57	0.9	0.82	205	0.83	61	0.61
		Footer	0	--	2.1	2.33	2.9	1.38	2.2	2.00	247	1	15	0.15
		Hdr + Ftr	vdd / 0	--	2.1	2.33	2.2	1.05	2.1	1.91	206	0.83	54	0.54
		Header	+	13	1.4	1.56	1.5	0.71	1.4	1.27	168	0.68	35	0.35
		Footer	-	--	3.3	3.67	4.0	1.90	3.4	3.09	246	0.99	31	0.31
		Hdr + Ftr	+ / -	43	3.3	3.67	3.2	1.52	3.3	3.00	168	0.68	119	1.19
8T	SG 3 x t; t; t; t	Nom.			0.6	--	2.0	--	0.9	--	248	--		
		Header	t	1	0.6	1	1.7	0.85	0.8	0.89	243	0.98		
		Footer	t	--	0.9	1.50	2.2	1.10	1.2	1.33	247	1		
		Hdr + Ftr	t	8	0.9	1.50	1.9	0.95	1.1	1.22	243	0.98		
		Header	vdd	1	0.6	1	1.2	0.60	0.8	0.89	205	0.83		
		Footer	0	--	1.8	3.00	2.8	1.40	2.0	2.22	247	1		
		Hdr + Ftr	vdd / 0	--	1.8	3.00	2.3	1.15	1.9	2.11	206	0.83		
		Header	+	26	1.4	2.33	1.8	0.90	1.5	1.67	168	0.68		
		Footer	-	--	2.9	4.83	3.8	1.90	3.1	3.44	246	0.99		
		Hdr + Ftr	+ / -	38	2.9	4.83	3.1	1.55	2.9	3.22	168	0.68		
	LP_SGR -; -; +; t	Nom.			0.1	--	0.9	--	0.3	--	254	--		
		Header	t	1	0.1	1	0.9	1	0.2	0.67	254	1		
		Footer	t	--	0.2	2.00	0.9	1	0.3	1	254	1		
		Hdr + Ftr	t	--	0.2	2.00	0.9	1	0.3	1	254	1		
		Header	vdd	34	0.2	2.00	0.8	0.89	0.3	1	253	1		
		Footer	0	--	0.3	3.00	1.0	1.11	0.4	1.33	254	1		
		Hdr + Ftr	vdd / 0	--	0.3	3.00	0.9	1	0.4	1.33	253	1		
		Header	+	258	0.3	3.00	0.8	0.89	0.4	1.33	252	0.99		
		Footer	-	--	0.3	3.00	1.0	1.11	0.5	1.67	254	1		
		Hdr + Ftr	+ / -	--	0.4	4.00	0.9	1	0.5	1.67	252	0.99		
	LP_INV1 t; -; vdd; t	Nom.			0.1	--	0.8	--	0.3	--	260	--		
		Header	t	1	0.1	1	0.7	0.88	0.3	1	260	1		
		Footer	t	--	0.3	3.00	0.8	1	0.4	1.33	260	1		
		Hdr + Ftr	t	--	0.3	3.00	0.8	1	0.4	1.33	260	1		
		Header	vdd	1	0.1	1	0.6	0.75	0.2	0.67	260	1		
		Footer	0	--	0.5	5.00	0.9	1.13	0.5	1.67	260	1		
		Hdr + Ftr	vdd / 0	--	0.5	5.00	0.8	1	0.5	1.67	259	1		
		Header	+	5	0.3	3.00	0.6	0.75	0.3	1	259	1		
		Footer	-	--	0.6	6.00	1.1	1.38	0.7	2.33	260	1		
		Hdr + Ftr	+ / -	--	0.7	7.00	0.9	1.13	0.7	2.33	258	0.99		
	LP_INV1.2 t; -; +; t	Nom.			0.1	--	0.6	--	0.2	--	254	--		
		Header	t	1	0.1	1	0.6	1	0.2	1	254	1		
		Footer	t	--	0.2	2.00	0.6	1	0.3	1.50	254	1		
		Hdr + Ftr	t	--	0.2	2.00	0.6	1	0.3	1.50	254	1		
		Header	vdd	1	0.1	1	0.5	0.83	0.2	1	253	1		
		Footer	0	--	0.3	3.00	0.6	1	0.3	1.50	254	1		
		Hdr + Ftr	vdd / 0	--	0.3	3.00	0.6	1	0.3	1.50	253	1		
		Header	+	46	0.2	2.00	0.4	0.67	0.2	1	252	0.99		
		Footer	-	--	0.3	3.00	0.7	1.17	0.4	2.00	254	1		
		Hdr + Ftr	+ / -	--	0.3	3.00	0.6	1.00	0.4	2.00	252	0.99		

Table C.7. 6T leakage, delay, and EDP 1 V V_{DD} results of headers/footers per two cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP^{+}	32 bits × 1024 words Array					
										Delay (ps)				Ave* EDP (ps×fJ)	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	5	--	120	--
		Header	t	0.592	0.523	*0.88*	17.5	*0.88*	*1*	32	*1*	6	*1.20*	84	*0.70*
		Footer	t	0.592	0.210	*0.35*	23.1	*1.17*	--	58	*1.81*	4	*0.80*	274	*2.28*
		Hdr + Ftr	t	0.592	0.141	*0.24*	15.0	*0.76*	*26*	57	*1.78*	6	*1.20*	238	*1.98*
		Header	vdd	0.592	0.520	*0.88*	17.4	*0.88*	*1*	32	*1*	8	*1.60*	79	*0.66*
		Footer	0	0.592	0.188	*0.32*	53.3	*2.68*	--	93	*2.91*	4	*0.80*	566	*4.72*
		Hdr + Ftr	vdd / 0	0.592	0.117	*0.20*	32.4	*1.63*	--	92	*2.88*	8	*1.60*	509	*4.24*
		Header	+	0.592	0.503	*0.85*	16.9	*0.85*	*1*	32	*1*	8	*1.60*	77	*0.64*
		Footer	-	0.592	0.155	*0.26*	57.1	*2.87*	--	106	*3.31*	5	*1*	688	*5.73*
		Hdr + Ftr	+ / -												
	LP -; -; +	Nom.		0.021	0.021	--	4.9	--		84	--	6	--	103	--
		Header	t	0.021	0.021	*1*	4.9	*1*	*1*	84	*1*	7	*1.17*	85	*0.83*
		Footer	t												
		Hdr + Ftr	t	0.021	0.019	*0.90*	10.4	*2.13*	--	129	*1.54*	7	*1.17*	161	*1.56*
		Header	vdd	0.021	0.021	*1*	4.9	*1*	*1*	84	*1*	8	*1.33*	80	*0.78*
		Footer	0	0.021	0.019	*0.90*	14.0	*2.89*	--	150	*1.79*	6	*1*	222	*2.16*
		Hdr + Ftr	vdd / 0	0.021	0.018	*0.86*	13.3	*2.73*	--	150	*1.79*	7	*1.17*	189	*1.83*
		Header	+	0.021	0.019	*0.90*	4.4	*0.90*	*1*	84	*1*	8	*1.33*	79	*0.77*
		Footer	-	0.021	0.008	*0.38*	6.5	*1.35*	--	158	*1.88*	6	*1*	243	*2.36*
		Hdr + Ftr	+ / -	0.021	0.005	*0.24*	4.1	*0.84*	*130*	158	*1.88*	8	*1.33*	201	*1.95*

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

$^{+}$ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table C.8. 6T energy 1 V V_{DD} results of headers/footers per two cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP^{+}	32 bits × 1024 words Array					
										Energy (fJ)					
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*
6T	SG t; t; t	Nom.		0.592	0.592	--	19.9	--		2.4	--	9.3	--	3.8	--
		Header	t	0.592	0.523	*0.88*	17.5	*0.88*	*1*	2.1	*0.88*	4.8	*0.52*	2.6	*0.68*
		Footer	t	0.592	0.210	*0.35*	23.1	*1.17*	--	4.0	*1.67*	7.7	*0.83*	4.8	*1.26*
		Hdr + Ftr	t	0.592	0.141	*0.24*	15.0	*0.76*	*26*	3.9	*1.63*	5.2	*0.56*	4.1	*1.08*
		Header	vdd	0.592	0.520	*0.88*	17.4	*0.88*	*1*	2.0	*0.83*	4.3	*0.46*	2.5	*0.66*
		Footer	0	0.592	0.188	*0.32*	53.3	*2.68*	--	5.5	*2.29*	8.4	*0.90*	6.1	*1.61*
		Hdr + Ftr	vdd / 0	0.592	0.117	*0.20*	32.4	*1.63*	--	5.5	*2.29*	5.5	*0.59*	5.5	*1.45*
		Header	+	0.592	0.503	*0.85*	16.9	*0.85*	*1*	2.0	*0.83*	4.1	*0.44*	2.4	*0.63*
		Footer	-	0.592	0.155	*0.26*	57.1	*2.87*	--	6.0	*2.50*	8.6	*0.92*	6.5	*1.71*
		Hdr + Ftr	+ / -												
	LP -; -; +	Nom.		0.021	0.021	--	4.9	--		0.5	--	3.9	--	1.2	--
		Header	t	0.021	0.021	*1*	4.9	*1*	*1*	0.5	*1*	2.9	*0.74*	1.0	*0.83*
		Footer	t												
		Hdr + Ftr	t	0.021	0.019	*0.90*	10.4	*2.13*	--	0.8	*1.60*	2.9	*0.74*	1.2	*1*
		Header	vdd	0.021	0.021	*1*	4.9	*1*	*1*	0.5	*1*	2.7	*0.69*	0.9	*0.75*
		Footer	0	0.021	0.019	*0.90*	14.0	*2.89*	--	0.9	*1.80*	3.7	*0.95*	1.5	*1.25*
		Hdr + Ftr	vdd / 0	0.021	0.018	*0.86*	13.3	*2.73*	--	0.9	*1.80*	2.7	*0.69*	1.3	*1.08*
		Header	+	0.021	0.019	*0.90*	4.4	*0.90*	*1*	0.5	*1*	2.7	*0.69*	0.9	*0.75*
		Footer	-	0.021	0.008	*0.38*	6.5	*1.35*	--	1.0	*2.00*	3.7	*0.95*	1.5	*1.25*
		Hdr + Ftr	+ / -	0.021	0.005	*0.24*	4.1	*0.84*	*130*	0.9	*1.80*	2.7	*0.69*	1.3	*1.08*

Table C.9. 8T leakage, delay, and EDP 1 V V_{DD} results of headers/footers per two cells

	Scheme *Pass; Inv-N; Inv-P; Read*	**Leak. Tran.**	**Leak. Tran. Config.**	**Leak. / Cell (nA)**	**Leakage / Cell in Sleep-Mode (nA)**		**Leakage EDP / Array in Sleep-Mode (ps×fJ)**		**# Arrays for Break-Even EDP**$^{+}$	**32 bits × 1024 words Array**					
										Delay (ps)				**Ave* EDP (ps×fJ)**	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
8T	SG 3 x t; t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	3	--	66	--
		Header	t	0.592	0.523	*0.88*	17.5	*0.88*	*1*	32	*1*	5	*1.67*	49	*0.74*
		Footer	t	0.591	0.210	*0.35*	22.4	*1.13*	--	57	*1.78*	3	*1*	200	*3.03*
		Hdr + Ftr	t	0.591	0.141	*0.24*	14.5	*0.73*	*23*	56	*1.75*	6	*2.00*	182	*2.76*
		Header	vdd	0.592	0.520	*0.88*	17.4	*0.88*	*1*	32	*1*	7	*2.33*	47	*0.71*
		Footer	0	0.591	0.188	*0.32*	49.9	*2.51*	--	90	*2.81*	3	*1*	431	*6.53*
		Hdr + Ftr	vdd / 0	0.591	0.117	*0.20*	30.4	*1.53*	--	89	*2.78*	8	*2.67*	399	*6.05*
		Header	+	0.592	0.503	*0.85*	16.9	*0.85*	*1*	32	*1*	8	*2.67*	46	*0.70*
		Footer	-	0.591	0.096	*0.16*	33.4	*1.68*	--	103	*3.22*	3	*1*	529	*8.02*
		Hdr + Ftr	+ / -												
	LP_SGR -; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	8	--	39	--
		Header	t	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	11	*1.38*	32	*0.82*
		Footer	t												
		Hdr + Ftr	t	0.021	0.019	*0.90*	2.0	*2.87*	--	57	*1.78*	11	*1.38*	100	*2.56*
		Header	vdd	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	13	*1.63*	30	*0.77*
		Footer	0												
		Hdr + Ftr	vdd / 0	0.021	0.018	*0.86*	4.8	*6.78*	--	90	*2.81*	12	*1.50*	186	*4.77*
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	32	*1*	13	*1.63*	30	*0.77*
		Footer	-												
		Hdr + Ftr	+ / -	0.021	0.005	*0.24*	1.8	*2.51*	--	104	*3.25*	13	*1.63*	217	*5.56*
	LP_INV1 t; -; vdd; t	Nom.		0.096	0.096	--	3.2	--		32	--	4	--	30	--
		Header	t	0.096	0.038	*0.40*	1.3	*0.40*	*1*	32	*1*	6	*1.50*	25	*0.83*
		Footer	t	0.096	0.094	*0.98*	10.0	*3.11*	--	57	*1.78*	4	*1*	96	*3.20*
		Hdr + Ftr	t	0.096	0.036	*0.38*	3.8	*1.19*	--	57	*1.78*	6	*1.50*	91	*3.03*
		Header	vdd	0.096	0.036	*0.38*	1.2	*0.38*	*1*	32	*1*	7	*1.75*	25	*0.83*
		Footer	0	0.096	0.094	*0.98*	25.5	*7.92*	--	91	*2.84*	4	*1*	191	*6.37*
		Hdr + Ftr	vdd / 0	0.096	0.034	*0.35*	9.0	*2.80*	--	90	*2.81*	8	*2.00*	184	*6.13*
		Header	+	0.096	0.019	*0.20*	0.6	*0.20*	*1*	32	*1*	8	*2.00*	25	*0.83*
		Footer	-	0.096	0.083	*0.86*	29.4	*9.13*	--	104	*3.25*	4	*1*	227	*7.57*
		Hdr + Ftr	+ / -	0.096	0.006	*0.06*	2.1	*0.66*	*174*	104	*3.25*	9	*2.25*	219	*7.30*
	LP_INV1.2 t; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	5	--	26	--
		Header	t	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	6	*1.20*	22	*0.85*
		Footer	t	0.021	0.019	*0.90*	2.0	*2.87*	--	57	*1.78*	5	*1*	87	*3.35*
		Hdr + Ftr	t	0.021	0.019	*0.90*	2.0	*2.87*	--	57	*1.78*	7	*1.40*	82	*3.15*
		Header	vdd	0.021	0.021	*1*	0.7	*1*	*1*	32	*1*	7	*1.40*	22	*0.85*
		Footer	0	0.021	0.019	*0.90*	5.2	*7.32*	--	91	*2.84*	5	*1*	165	*6.35*
		Hdr + Ftr	vdd / 0	0.021	0.018	*0.86*	4.8	*6.78*	--	90	*2.81*	8	*1.60*	159	*6.12*
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	32	*1*	8	*1.60*	21	*0.81*
		Footer	-	0.021	0.008	*0.38*	2.8	*4.02*	--	104	*3.25*	5	*1*	193	*7.42*
		Hdr + Ftr	+ / -	0.021	0.005	*0.24*	1.8	*2.51*	--	104	*3.25*	9	*1.80*	187	*7.19*

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

$^{+}$ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table C.10. 8T energy 1 V V_{DD} results of headers/footers per two cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP+	32 bits × 1024 words Array					
										Energy (fJ)					
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*
8T	SG 3 x t; t; t; t	Nom.		0.592	0.592	--	19.9	--		0.9	--	6.9	--	2.1	--
		Header	t	0.592	0.523	*0.88*	17.5	*0.88*	*1*	0.8	*0.89*	4.4	*0.64*	1.6	*0.76*
		Footer	t	0.591	0.210	*0.35*	22.4	*1.13*	--	2.7	*3.00*	6.7	*0.97*	3.5	*1.67*
		Hdr + Ftr	t	0.591	0.141	*0.24*	14.5	*0.73*	*23*	2.7	*3.00*	5.2	*0.75*	3.2	*1.52*
		Header	vdd	0.592	0.520	*0.88*	17.4	*0.88*	*1*	0.8	*0.89*	4.1	*0.59*	1.4	*0.67*
		Footer	0	0.591	0.188	*0.32*	49.9	*2.51*	--	4.2	*4.67*	7.3	*1.06*	4.8	*2.29*
		Hdr + Ftr	vdd / 0	0.591	0.117	*0.20*	30.4	*1.53*	--	4.2	*4.67*	5.5	*0.80*	4.5	*2.14*
		Header	+	0.592	0.503	*0.85*	16.9	*0.85*	*1*	0.8	*0.89*	4.0	*0.58*	1.5	*1*
		Footer	-	0.591	0.096	*0.16*	33.4	*1.68*	--	4.6	*5.11*	7.6	*1.10*	5.2	*2.48*
		Hdr + Ftr	+ / -												
	LP_SGR -; -; +; t	Nom.		0.021	0.021	--	0.7	--		0.3	--	5.0	--	1.2	--
		Header	t	0.021	0.021	*1*	0.7	*1*	*1*	0.3	*1*	4.0	*0.80*	1.0	*0.83*
		Footer	t												
		Hdr + Ftr	t	0.021	0.019	*0.90*	2.0	*2.87*	--	1.2	*4.00*	4.0	*0.80*	1.8	*1.50*
		Header	vdd	0.021	0.021	*1*	0.7	*1*	*1*	0.3	*1*	3.8	*0.76*	1.0	*0.83*
		Footer	0												
		Hdr + Ftr	vdd / 0	0.021	0.018	*0.86*	4.8	*6.78*	--	1.6	*5.33*	3.9	*0.78*	2.1	*1.75*
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	0.3	*1*	3.8	*0.76*	1.0	*0.83*
		Footer	-												
		Hdr + Ftr	+ / -	0.021	0.005	*0.24*	1.8	*2.51*	--	1.7	*5.67*	3.9	*0.78*	2.1	*1.75*
	LP_INV1 t; -; vdd; t	Nom.		0.096	0.096	--	3.2	--		0.4	--	3.2	--	0.9	--
		Header	t	0.096	0.038	*0.40*	1.3	*0.40*	*1*	0.3	*0.75*	2.7	*0.84*	0.8	*0.89*
		Footer	t	0.096	0.094	*0.98*	10.0	*3.11*	--	1.3	*3.25*	3.2	*1*	1.7	*1.89*
		Hdr + Ftr	t	0.096	0.036	*0.38*	3.8	*1.19*	--	1.3	*3.25*	2.8	*0.88*	1.6	*1.78*
		Header	vdd	0.096	0.036	*0.38*	1.2	*0.38*	*1*	0.3	*0.75*	2.6	*0.81*	0.8	*0.89*
		Footer	0	0.096	0.094	*0.98*	25.5	*7.92*	--	1.8	*4.50*	3.3	*1.03*	2.1	*2.33*
		Hdr + Ftr	vdd / 0	0.096	0.034	*0.35*	9.0	*2.80*	--	1.9	*4.75*	2.8	*0.88*	2.0	*2.22*
		Header	+	0.096	0.019	*0.20*	0.6	*0.20*	*1*	0.3	*0.75*	2.6	*0.81*	0.8	*0.89*
		Footer	-	0.096	0.083	*0.86*	29.4	*9.13*	--	1.9	*4.75*	3.4	*1.06*	2.2	*2.44*
		Hdr + Ftr	+ / -	0.096	0.006	*0.06*	2.1	*0.66*	*174*	2.0	*5.00*	2.8	*0.88*	2.1	*2.33*
	LP_INV1.2 t; -; +; t	Nom.		0.021	0.021	--	0.7	--		0.3	--	3.0	--	0.8	--
		Header	t	0.021	0.021	*1*	0.7	*1*	*1*	0.3	*1*	2.5	*0.83*	0.7	*0.88*
		Footer	t	0.021	0.019	*0.90*	2.0	*2.87*	--	1.2	*4.00*	2.9	*0.97*	1.5	*1.88*
		Hdr + Ftr	t	0.021	0.019	*0.90*	2.0	*2.87*	--	1.2	*4.00*	2.5	*0.83*	1.4	*1.75*
		Header	vdd	0.021	0.021	*1*	0.7	*1*	*1*	0.3	*1*	2.4	*0.80*	0.7	*0.88*
		Footer	0	0.021	0.019	*0.90*	5.2	*7.32*	--	1.6	*5.33*	2.9	*0.97*	1.8	*2.25*
		Hdr + Ftr	vdd / 0	0.021	0.018	*0.86*	4.8	*6.78*	--	1.6	*5.33*	2.4	*0.80*	1.8	*2.25*
		Header	+	0.021	0.019	*0.90*	0.6	*0.90*	*1*	0.3	*1*	2.4	*0.80*	0.7	*0.88*
		Footer	-	0.021	0.008	*0.38*	2.8	*4.02*	--	1.6	*5.33*	2.9	*0.97*	1.9	*2.38*
		Hdr + Ftr	+ / -	0.021	0.005	*0.24*	1.8	*2.51*	--	1.7	*5.67*	2.4	*0.80*	1.8	*2.25*

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

+ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table C.11. Leakage, delay, and EDP 0.6 V V_{DD} results of headers/footers per two cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP⁺	32 bits × 1024 words Array					
										Delay (ps)				Ave* EDP (ps×fJ)	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.499	0.499	--	25.5	--		51	--	5	--	57	--
		Header	t	0.499	0.443	*0.89*	22.7	*0.89*	*1*	51	*1*	7	*1.40*	50	*0.88*
		Footer	t	0.499	0.190	*0.38*	34.4	*1.35*	--	96	*1.88*	5	*1*	165	*2.89*
		Hdr + Ftr	t	0.499	0.134	*0.27*	24.3	*0.95*	*50*	96	*1.88*	7	*1.40*	157	*2.75*
		Header	vdd	0.499	0.423	*0.85*	20.8	*0.81*	*4*	50	*0.98*	71	*14.20*	73	*1.28*
		Footer	0	0.499	0.170	*0.34*	227.7	*8.92*	--	261	*5.12*	5	*1*	906	*15.89*
		Hdr + Ftr	vdd / 0	0.499	0.094	*0.19*	125.9	*4.93*	--	261	*5.12*	72	*14.40*	879	*15.42*
		Header	+	0.499	0.423	*0.85*	20.8	*0.81*	*43*	50	*0.98*	179	*35.80*	382	*6.70*
		Footer	-	0.499	0.132	*0.26*	590.5	*23.14*	--	477	*9.35*	5	*1*	2680	*47.02*
		Hdr + Ftr	+ / -	0.498	0.006	*0.01*	26.7	*1.05*	--	476	*9.33*	181	*36.20*	2628	*46.11*
8T	SG 3 x t; t; t; t	Nom.		0.498	0.498	--	24.5	--		50	--	4	--	43	--
		Header	t	0.498	0.443	*0.89*	21.8	*0.89*	*1*	50	*1*	8	*2.00*	38	*0.88*
		Footer	t	0.498	0.190	*0.38*	32.3	*1.32*	--	93	*1.86*	4	*1*	138	*3.21*
		Hdr + Ftr	t	0.498	0.134	*0.27*	22.8	*0.93*	*33*	93	*1.86*	8	*2.00*	132	*3.07*
		Header	vdd	0.498	0.423	*0.85*	20.8	*0.85*	*11*	50	*1*	91	*22.75*	100	*2.33*
		Footer	0	0.497	0.170	*0.34*	200.6	*8.20*	--	245	*4.90*	4	*1*	763	*17.74*
		Hdr + Ftr	vdd / 0	0.497	0.094	*0.19*	110.9	*4.53*	--	245	*4.90*	93	*23.25*	734	*17.07*
		Header	+	0.496	0.423	*0.85*	20.8	*0.85*	*67*	50	*1*	205	*51.25*	448	*10.42*
		Footer	-	0.497	0.082	*0.16*	302.3	*12.35*	--	433	*8.66*	4	*1*	2161	*50.26*
		Hdr + Ftr	+ / -	0.493	0.006	*0.01*	22.0	*0.90*	*498*	432	*8.64*	209	*52.25*	2079	*48.35*
	LP_SGR -; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	32	--	13	--
		Header	t	0.018	0.018	*1*	0.9	*1*	*1*	50	*1*	38	*1.19*	12	*0.92*
		Footer	t	0.018	0.017	*0.94*	2.9	*3.27*	--	93	*1.86*	32	*1*	34	*2.62*
		Hdr + Ftr	t	0.018	0.016	*0.89*	2.7	*3.08*	--	93	*1.86*	38	*1.19*	33	*2.54*
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*136*	50	*1*	137	*4.28*	46	*3.54*
		Footer	0	0.018	0.016	*0.89*	19.3	*21.87*	--	248	*4.96*	31	*0.97*	118	*9.08*
		Hdr + Ftr	vdd / 0	0.018	0.013	*0.72*	15.7	*17.77*	--	248	*4.96*	137	*4.28*	116	*8.92*
		Header	+	0.018	0.015	*0.83*	0.7	*0.83*	*592*	50	*1*	382	*11.94*	158	*12.15*
		Footer	-	0.018	0.007	*0.39*	26.5	*29.98*	--	439	*8.78*	31	*0.97*	247	*19.00*
		Hdr + Ftr	+ / -	0.018	0.004	*0.22*	15.2	*17.13*	--	439	*8.78*	381	*11.91*	235	*18.08*
	LP_INV1 t; -; vdd; t	Nom.		0.081	0.081	--	4.0	--		50	--	10	--	14	--
		Header	t	0.081	0.035	*0.43*	1.7	*0.43*	*1*	50	*1*	14	*1.40*	13	*0.93*
		Footer	t	0.081	0.079	*0.98*	13.7	*3.45*	--	94	*1.88*	10	*1*	41	*2.93*
		Hdr + Ftr	t	0.081	0.033	*0.41*	5.6	*1.41*	--	93	*1.86*	14	*1.40*	41	*2.93*
		Header	vdd	0.081	0.015	*0.19*	0.7	*0.19*	*3*	50	*1*	71	*7.10*	20	*1.43*
		Footer	0	0.081	0.079	*0.98*	96.3	*24.19*	--	249	*4.98*	10	*1*	186	*13.29*
		Hdr + Ftr	vdd / 0	0.081	0.013	*0.16*	15.7	*3.95*	--	248	*4.96*	72	*7.20*	182	*13.00*
		Header	+	0.081	0.015	*0.19*	0.7	*0.19*	*14*	50	*1*	173	*17.30*	79	*5.64*
		Footer	-	0.080	0.070	*0.86*	266.4	*66.92*	--	440	*8.80*	9	*0.90*	471	*33.64*
		Hdr + Ftr	+ / -	0.080	0.004	*0.05*	15.2	*3.81*	--	439	*8.78*	176	*17.60*	457	*32.64*
	LP_INV1.2 t; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	18	--	9	--
		Header	t	0.018	0.018	*1*	0.9	*1*	*1*	50	*1*	22	*1.22*	9	*1*
		Footer	t	0.018	0.017	*0.94*	2.9	*3.27*	--	93	*1.86*	18	*1*	28	*3.11*
		Hdr + Ftr	t	0.018	0.016	*0.89*	2.7	*3.08*	--	93	*1.86*	23	*1.28*	27	*3.00*
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*18*	50	*1*	70	*3.89*	13	*1.44*
		Footer	0	0.018	0.016	*0.89*	19.3	*21.87*	--	248	*4.96*	18	*1*	101	*11.22*
		Hdr + Ftr	vdd / 0	0.018	0.013	*0.72*	15.7	*17.77*	--	248	*4.96*	71	*3.94*	99	*11.00*
		Header	+	0.018	0.015	*0.83*	0.7	*0.83*	*124*	50	*1*	166	*9.22*	39	*4.33*
		Footer	-	0.018	0.007	*0.39*	26.5	*29.98*	--	439	*8.78*	19	*1.06*	217	*24.11*
		Hdr + Ftr	+ / -	0.018	0.004	*0.22*	15.2	*17.13*	--	439	*8.78*	168	*9.33*	207	*23.00*

Table C.12. Energy 0.6 V V_{DD} results of headers/footers per two cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP$^+$	32 bits × 1024 words Array					
										Energy (fJ)					
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*
6T	SG t; t; t	Nom.		0.499	0.499	--	25.5	--		0.9	--	2.1	--	1.1	--
		Header	t	0.499	0.443	0.89	22.7	0.89	1	0.8	0.89	1.6	0.76	1.0	0.91
		Footer	t	0.499	0.190	0.38	34.4	1.35	--	1.5	1.67	2.4	1.14	1.7	1.55
		Hdr + Ftr	t	0.499	0.134	0.27	24.3	0.95	50	1.5	1.67	2.0	0.95	1.6	1.45
		Header	vdd	0.499	0.423	0.85	20.8	0.81	4	1.0	1.11	1.2	0.57	1.0	0.91
		Footer	0	0.499	0.170	0.34	227.7	8.92	--	3.3	3.67	4.0	1.90	3.5	3.18
		Hdr + Ftr	vdd / 0	0.499	0.094	0.19	125.9	4.93	--	3.4	3.78	3.3	1.57	3.4	3.09
		Header	+	0.499	0.423	0.85	20.8	0.81	43	2.1	2.33	2.1	1	2.1	1.91
		Footer	-	0.499	0.132	0.26	590.5	23.14	--	5.5	6.11	6.1	2.90	5.6	5.09
		Hdr + Ftr	+ / -	0.498	0.006	0.01	26.7	1.05	--	5.6	6.22	5.3	2.52	5.5	5.00
8T	SG 3 x t; t; t; t	Nom.		0.498	0.498	--	24.5	--		0.6	--	2.0	--	0.9	--
		Header	t	0.498	0.443	0.89	21.8	0.89	1	0.6	1	1.6	0.80	0.8	0.89
		Footer	t	0.498	0.190	0.38	32.3	1.32	--	1.2	2.00	2.4	1.20	1.5	1.67
		Hdr + Ftr	t	0.498	0.134	0.27	22.8	0.93	33	1.3	2.17	2.0	1	1.4	1.56
		Header	vdd	0.498	0.423	0.85	20.8	0.85	11	1.0	1.67	1.5	0.75	1.1	1.22
		Footer	0	0.497	0.170	0.34	200.6	8.20	--	2.9	4.83	3.8	1.90	3.1	3.44
		Hdr + Ftr	vdd / 0	0.497	0.094	0.19	110.9	4.53	--	2.9	4.83	3.2	1.60	3.0	3.33
		Header	+	0.496	0.423	0.85	20.8	0.85	67	2.1	3.50	2.4	1.20	2.2	2.44
		Footer	-	0.497	0.082	0.16	302.3	12.35	--	4.8	8.00	5.6	2.80	5.0	5.56
		Hdr + Ftr	+ / -	0.493	0.006	0.01	22.0	0.90	498	4.8	8.00	4.8	2.40	4.8	5.33
	LP_SGR -; -; +; t	Nom.		0.018	0.018	--	0.9	--		0.1	--	0.9	--	0.3	--
		Header	t	0.018	0.018	1	0.9	1	1	0.1	1	0.9	1	0.2	0.67
		Footer	t	0.018	0.017	0.94	2.9	3.27	--	0.2	2.00	0.9	1	0.4	1.33
		Hdr + Ftr	t	0.018	0.016	0.89	2.7	3.08	--	0.2	2.00	0.9	1	0.4	1.33
		Header	vdd	0.018	0.015	0.83	0.7	0.83	136	0.2	2.00	0.8	0.89	0.3	1
		Footer	0	0.018	0.016	0.89	19.3	21.87	--	0.3	3.00	1.0	1.11	0.5	1.67
		Hdr + Ftr	vdd / 0	0.018	0.013	0.72	15.7	17.77	--	0.4	4.00	0.9	1	0.5	1.67
		Header	+	0.018	0.015	0.83	0.7	0.83	592	0.3	3.00	0.8	0.89	0.4	1.33
		Footer	-	0.018	0.007	0.39	26.5	29.98	--	0.4	4.00	1.1	1.22	0.6	2.00
		Hdr + Ftr	+ / -	0.018	0.004	0.22	15.2	17.13	--	0.4	4.00	0.9	1	0.5	1.67
	LP_INV1 t; -; vdd; t	Nom.		0.081	0.081	--	4.0	--		0.1	--	0.8	--	0.3	--
		Header	t	0.081	0.035	0.43	1.7	0.43	1	0.1	1	0.7	0.88	0.3	1
		Footer	t	0.081	0.079	0.98	13.7	3.45	--	0.3	3.00	0.8	1	0.4	1.33
		Hdr + Ftr	t	0.081	0.033	0.41	5.6	1.41	--	0.3	3.00	0.8	1	0.4	1.33
		Header	vdd	0.081	0.015	0.19	0.7	0.19	3	0.2	2.00	0.5	0.63	0.3	1
		Footer	0	0.081	0.079	0.98	96.3	24.19	--	0.7	7.00	1.1	1.38	0.7	2.33
		Hdr + Ftr	vdd / 0	0.081	0.013	0.16	15.7	3.95	--	0.7	7.00	0.9	1.13	0.7	2.33
		Header	+	0.081	0.015	0.19	0.7	0.19	14	0.4	4.00	0.6	0.75	0.5	1.67
		Footer	-	0.080	0.070	0.86	266.4	66.92	--	1.0	10.00	1.4	1.75	1.1	3.67
		Hdr + Ftr	+ / -	0.080	0.004	0.05	15.2	3.81	--	1.0	10.00	1.2	1.50	1.0	3.33
	LP_INV1.2 t; -; +; t	Nom.		0.018	0.018	--	0.9	--		0.1	--	0.6	--	0.2	--
		Header	t	0.018	0.018	1	0.9	1	1	0.1	1	0.6	1	0.2	1
		Footer	t	0.018	0.017	0.94	2.9	3.27	--	0.2	2.00	0.6	1	0.3	1.50
		Hdr + Ftr	t	0.018	0.016	0.89	2.7	3.08	--	0.2	2.00	0.6	1	0.3	1.50
		Header	vdd	0.018	0.015	0.83	0.7	0.83	18	0.1	1	0.4	0.67	0.2	1
		Footer	0	0.018	0.016	0.89	19.3	21.87	--	0.3	3.00	0.7	1.17	0.4	2.00
		Hdr + Ftr	vdd / 0	0.018	0.013	0.72	15.7	17.77	--	0.4	4.00	0.6	1	0.4	2.00
		Header	+	0.018	0.015	0.83	0.7	0.83	124	0.2	2.00	0.4	0.67	0.2	1
		Footer	-	0.018	0.007	0.39	26.5	29.98	--	0.4	4.00	0.7	1.17	0.5	2.50
		Hdr + Ftr	+ / -	0.018	0.004	0.22	15.2	17.13	--	0.4	4.00	0.6	1	0.5	2.50

Table C.13. 6T leakage, delay, and EDP 1 V V_{DD} results of headers/footers per four cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP+	32 bits × 1024 words Array					
										Delay (ps)				Ave* EDP (ps×fJ)	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	5	--	120	--
		Header	t	0.592	0.514	*0.87*	17.2	*0.87*	*1*	32	*1*	8	*1.60*	75	*0.63*
		Footer	t	0.592	0.160	*0.27*	39.7	*2.00*	--	87	*2.72*	5	*1*	512	*4.27*
		Hdr + Ftr	t	0.592	0.080	*0.14*	19.8	*1*	--	87	*2.72*	8	*1.60*	460	*3.83*
		Header	vdd	0.592	0.512	*0.86*	17.2	*0.86*	*1*	32	*1*	11	*2.20*	62	*0.52*
		Footer	0	0.592	0.158	*0.27*	156.7	*7.89*	--	174	*5.44*	5	*1*	1395	*11.63*
		Hdr + Ftr	vdd / 0	0.592	0.065	*0.11*	63.7	*3.21*	--	173	*5.41*	11	*2.20*	1290	*10.75*
		Header	+	0.592	0.502	*0.85*	16.8	*0.85*	*1*	32	*1*	12	*2.40*	66	*0.55*
		Footer	-	0.592	0.154	*0.26*	203.9	*10.26*	--	201	*6.28*	5	*1*	1727	*14.39*
		Hdr + Ftr	+ / -	0.592	0.004	*0.01*	5.2	*0.26*	*104*	200	*6.25*	12	*2.40*	1614	*13.45*
	LP -; -; +	Nom.		0.021	0.021	--	4.9	--		84	--	6	--	103	--
		Header	t	0.021	0.020	*0.95*	4.6	*0.95*	*1*	84	*1*	8	*1.33*	78	*0.76*
		Footer	t	0.021	0.018	*0.86*	18.3	*3.76*	--	176	*2.10*	5	*0.83*	275	*2.67*
		Hdr + Ftr	t	0.021	0.017	*0.81*	17.3	*3.55*	--	176	*2.10*	8	*1.33*	229	*2.22*
		Header	vdd	0.021	0.020	*0.95*	4.6	*0.95*	*1*	84	*1*	10	*1.67*	75	*0.73*
		Footer	0	0.021	0.017	*0.81*	30.5	*6.28*	--	234	*2.79*	5	*0.83*	386	*3.75*
		Hdr + Ftr	vdd / 0	0.021	0.016	*0.76*	28.5	*5.86*	--	233	*2.77*	10	*1.67*	324	*3.15*
		Header	+	0.021	0.018	*0.86*	4.2	*0.86*	*1*	84	*1*	11	*1.83*	74	*0.72*
		Footer	-	0.021	0.006	*0.29*	13.4	*2.76*	--	261	*3.11*	5	*0.83*	440	*4.27*
		Hdr + Ftr	+ / -	0.021	0.003	*0.14*	6.7	*1.38*	--	261	*3.11*	11	*1.83*	372	*3.61*

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

+ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table C.14. 6T energy 1 V V_{DD} results of headers/footers per four cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP+	32 bits × 1024 words Array					
										Energy (fJ)					
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*
6T	SG t; t; t	Nom.		0.592	0.592	--	19.9	--		2.4	--	9.3	--	3.8	--
		Header	t	0.592	0.514	*0.87*	17.2	*0.87*	*1*	1.9	*0.79*	4.2	*0.45*	2.4	*0.63*
		Footer	t	0.592	0.160	*0.27*	39.7	*2.00*	--	5.3	*2.21*	8.1	*0.87*	5.9	*1.55*
		Hdr + Ftr	t	0.592	0.080	*0.14*	19.8	*1*	--	5.3	*2.21*	5.3	*0.57*	5.3	*1.39*
		Header	vdd	0.592	0.512	*0.86*	17.2	*0.86*	*1*	1.8	*0.75*	3.5	*0.38*	2.1	*0.55*
		Footer	0	0.592	0.158	*0.27*	156.7	*7.89*	--	7.6	*3.17*	9.7	*1.04*	8.0	*2.11*
		Hdr + Ftr	vdd / 0	0.592	0.065	*0.11*	63.7	*3.21*	--	7.7	*3.21*	6.4	*0.69*	7.4	*1.95*
		Header	+	0.592	0.502	*0.85*	16.8	*0.85*	*1*	1.7	*0.71*	3.4	*0.37*	2.1	*0.55*
		Footer	-	0.592	0.154	*0.26*	203.9	*10.26*	--	8.2	*3.42*	10.2	*1.10*	8.6	*2.26*
		Hdr + Ftr	+ / -	0.592	0.004	*0.01*	5.2	*0.26*	*104*	8.4	*3.50*	6.8	*0.73*	8.1	*2.13*
	LP -; -; +	Nom.		0.021	0.021	--	4.9	--		0.5	--	3.9	--	1.2	--
		Header	t	0.021	0.020	*0.95*	4.6	*0.95*	*1*	0.5	*1*	2.6	*0.67*	0.9	*0.75*
		Footer	t	0.021	0.018	*0.86*	18.3	*3.76*	--	1.0	*2.00*	3.6	*0.92*	1.6	*1.33*
		Hdr + Ftr	t	0.021	0.017	*0.81*	17.3	*3.55*	--	1.0	*2.00*	2.6	*0.67*	1.3	*1.08*
		Header	vdd	0.021	0.020	*0.95*	4.6	*0.95*	*1*	0.5	*1*	2.4	*0.62*	0.9	*0.75*
		Footer	0	0.021	0.017	*0.81*	30.5	*6.28*	--	1.2	*2.40*	3.6	*0.92*	1.7	*1.42*
		Hdr + Ftr	vdd / 0	0.021	0.016	*0.76*	28.5	*5.86*	--	1.1	*2.20*	2.5	*0.64*	1.4	*1.17*
		Header	+	0.021	0.018	*0.86*	4.2	*0.86*	*1*	0.5	*1*	2.3	*0.59*	0.9	*0.75*
		Footer	-	0.021	0.006	*0.29*	13.4	*2.76*	--	1.2	*2.40*	3.7	*0.95*	1.7	*1.42*
		Hdr + Ftr	+ / -	0.021	0.003	*0.14*	6.7	*1.38*	--	1.2	*2.40*	2.5	*0.64*	1.4	*1.17*

Table C.15. 8T leakage, delay, and EDP 1 V V_{DD} results of headers/footers per four cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP+	32 bits × 1024 words Array					
										Delay (ps)				Ave* EDP (ps×fJ)	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
8T	SG 3 x t; t; t; t	Nom.		0.592	0.592	--	19.9	--		32	--	3	--	66	--
		Header	t	0.592	0.514	*0.87*	17.2	*0.87*	*1*	32	*1*	8	*2.67*	46	*0.70*
		Footer	t	0.591	0.159	*0.27*	37.6	*1.90*	--	85	*2.66*	3	*1*	390	*5.91*
		Hdr + Ftr	t	0.591	0.080	*0.14*	18.5	*0.93*	*214*	84	*2.63*	9	*3.00*	357	*5.41*
		Header	vdd	0.592	0.512	*0.86*	17.2	*0.86*	*1*	32	*1*	11	*3.67*	43	*0.65*
		Footer	0	0.590	0.145	*0.24*	130.9	*6.59*	--	166	*5.19*	3	*1*	1107	*16.77*
		Hdr + Ftr	vdd / 0												
		Header	+	0.592	0.502	*0.85*	15.8	*0.80*	*1*	31	*1*	12	*4.00*	42	*0.64*
		Footer	-	0.590	0.094	*0.16*	112.4	*5.66*	--	191	*5.97*	3	*1*	1380	*20.91*
		Hdr + Ftr	+ / -												
	LP_SGR -; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	8	--	39	--
		Header	t	0.021	0.020	*0.95*	0.7	*1*	*1*	32	*1*	13	*1.63*	29	*0.74*
		Footer	t	0.021	0.018	*0.86*	4.4	*6.19*	--	86	*2.69*	8	*1*	185	*4.74*
		Hdr + Ftr	t	0.021	0.017	*0.81*	4.0	*5.71*	--	85	*2.66*	13	*1.63*	170	*4.36*
		Header	vdd	0.021	0.020	*0.95*	0.7	*1*	*1*	32	*1*	19	*2.38*	28	*0.72*
		Footer	0	0.021	0.017	*0.81*	15.7	*22.31*	--	168	*5.25*	8	*1*	398	*10.21*
		Hdr + Ftr	vdd / 0	0.021	0.016	*0.76*	14.8	*21.00*	--	168	*5.25*	19	*2.38*	368	*9.44*
		Header	+	0.021	0.018	*0.86*	0.6	*0.86*	*1*	32	*1*	21	*2.63*	27	*0.69*
		Footer	-	0.021	0.006	*0.29*	7.4	*10.50*	--	194	*6.06*	8	*1*	463	*11.87*
		Hdr + Ftr	+ / -	0.021	0.003	*0.14*	3.7	*5.25*	--	194	*6.06*	21	*2.63*	430	*11.03*
	LP_INV1 t; -; vdd; t	Nom.		0.096	0.096	--	3.2	--		32	--	4	--	30	--
		Header	t	0.096	0.030	*0.31*	1.0	*0.31*	*1*	32	*1*	8	*2.00*	24	*0.80*
		Footer	t	0.096	0.093	*0.97*	22.5	*7.00*	--	86	*2.69*	4	*1*	176	*5.87*
		Hdr + Ftr	t	0.096	0.026	*0.27*	6.2	*1.91*	--	85	*2.66*	9	*2.25*	168	*5.60*
		Header	vdd	0.096	0.028	*0.29*	0.9	*0.29*	*1*	32	*1*	11	*2.75*	23	*0.77*
		Footer	0	0.096	0.092	*0.96*	85.1	*26.41*	--	168	*5.25*	4	*1*	414	*13.80*
		Hdr + Ftr	vdd / 0	0.095	0.024	*0.25*	22.2	*6.89*	--	168	*5.25*	13	*3.25*	404	*13.47*
		Header	+	0.096	0.018	*0.19*	0.6	*0.19*	*1*	32	*1*	12	*3.00*	23	*0.77*
		Footer	-	0.095	0.081	*0.84*	99.9	*31.01*	--	194	*6.06*	4	*1*	496	*16.53*
		Hdr + Ftr	+ / -												
	LP_INV1.2 t; -; +; t	Nom.		0.021	0.021	--	0.7	--		32	--	5	--	26	--
		Header	t	0.021	0.020	*0.95*	0.7	*1*	*1*	32	*1*	8	*1.60*	21	*0.81*
		Footer	t	0.021	0.018	*0.86*	4.3	*6.05*	--	85	*2.66*	5	*1*	152	*5.85*
		Hdr + Ftr	t	0.021	0.017	*0.81*	4.0	*5.71*	--	85	*2.66*	9	*1.80*	146	*5.62*
		Header	vdd	0.021	0.020	*0.95*	0.7	*1*	*1*	32	*1*	11	*2.20*	20	*0.77*
		Footer	0	0.021	0.017	*0.81*	15.7	*22.31*	--	168	*5.25*	5	*1*	334	*12.85*
		Hdr + Ftr	vdd / 0	0.021	0.016	*0.76*	14.8	*21.00*	--	168	*5.25*	13	*2.60*	325	*12.50*
		Header	+	0.021	0.018	*0.86*	0.6	*0.86*	*1*	32	*1*	12	*2.40*	20	*0.77*
		Footer	-												
		Hdr + Ftr	+ / -												

* The average EDP is calculated as 80% of the read EDP plus 20% of the write EDP.

+ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table C.16. 8T energy 1 V V_{DD} results of headers/footers per four cells

	Scheme *Pass; Inv-N; Inv-P; Read*	**Leak. Tran.**	**Leak. Tran. Config.**	**Leak. / Cell (nA)**	**Leakage / Cell in Sleep-Mode (nA)**		**Leakage EDP / Array in Sleep-Mode (ps×fJ)**		**# Arrays for Break-Even EDP$^+$**	**32 bits × 1024 words Array**					
										Energy (fJ)					
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*
8T	SG 3 x t; t; t; t	Nom.		0.592	0.592	--	19.9	--		0.9	--	6.9	--	2.1	--
		Header	t	0.592	0.514	*0.87*	17.2	*0.87*	*1*	0.8	*0.89*	3.9	*0.57*	1.5	*0.71*
		Footer	t	0.591	0.159	*0.27*	37.6	*1.90*	--	3.9	*4.33*	7.2	*1.04*	4.6	*2.19*
		Hdr + Ftr	t	0.591	0.080	*0.14*	18.5	*0.93*	*214*	4.0	*4.44*	5.3	*0.77*	4.2	*2.00*
		Header	vdd	0.592	0.512	*0.86*	17.2	*0.86*	*1*	0.8	*0.89*	3.5	*0.51*	1.4	*0.67*
		Footer	0	0.590	0.145	*0.24*	130.9	*6.59*	--	6.2	*6.89*	8.7	*1.26*	6.7	*3.19*
		Hdr + Ftr	vdd / 0												
		Header	+	0.592	0.502	*0.85*	15.8	*0.80*	*1*	0.8	*0.89*	3.4	*0.49*	1.3	*0.62*
		Footer	-	0.590	0.094	*0.16*	112.4	*5.66*	--	6.7	*7.44*	9.2	*1.33*	7.2	*3.43*
		Hdr + Ftr	+ / -												
	LP_SGR -; -; +; t	Nom.		0.021	0.021	--	0.7	--		0.3	--	5.0	--	1.2	--
		Header	t	0.021	0.020	*0.95*	0.7	*1*	*1*	0.3	*1*	3.7	*0.74*	0.9	*0.75*
		Footer	t	0.021	0.018	*0.86*	4.4	*6.19*	--	1.5	*5.00*	4.8	*0.96*	2.2	*1.83*
		Hdr + Ftr	t	0.021	0.017	*0.81*	4.0	*5.71*	--	1.5	*5.00*	3.8	*0.76*	2.0	*1.67*
		Header	vdd	0.021	0.020	*0.95*	0.7	*1*	*1*	0.2	*0.67*	3.4	*0.68*	0.9	*0.75*
		Footer	0	0.021	0.017	*0.81*	15.7	*22.31*	--	1.7	*5.67*	4.8	*0.96*	2.4	*2.00*
		Hdr + Ftr	vdd / 0	0.021	0.016	*0.76*	14.8	*21.00*	--	1.8	*6.00*	3.6	*0.72*	2.2	*1.83*
		Header	+	0.021	0.018	*0.86*	0.6	*0.86*	*1*	0.2	*0.67*	3.4	*0.68*	0.9	*0.75*
		Footer	-	0.021	0.006	*0.29*	7.4	*10.50*	--	1.8	*6.00*	4.9	*0.98*	2.4	*2.00*
		Hdr + Ftr	+ / -	0.021	0.003	*0.14*	3.7	*5.25*	--	1.9	*6.33*	3.6	*0.72*	2.2	*1.83*
	LP_INV1 t; -; vdd; t	Nom.		0.096	0.096	--	3.2	--		0.4	--	3.2	--	0.9	--
		Header	t	0.096	0.030	*0.31*	1.0	*0.31*	*1*	0.3	*0.75*	2.5	*0.78*	0.8	*0.89*
		Footer	t	0.096	0.093	*0.97*	22.5	*7.00*	--	1.8	*4.50*	3.3	*1.03*	2.1	*2.33*
		Hdr + Ftr	t	0.096	0.026	*0.27*	6.2	*1.91*	--	1.8	*4.50*	2.7	*0.84*	2.0	*2.22*
		Header	vdd	0.096	0.028	*0.29*	0.9	*0.29*	*1*	0.3	*0.75*	2.4	*0.75*	0.7	*0.78*
		Footer	0	0.096	0.092	*0.96*	85.1	*26.41*	--	2.2	*5.50*	3.6	*1.13*	2.5	*2.78*
		Hdr + Ftr	vdd / 0	0.095	0.024	*0.25*	22.2	*6.89*	--	2.3	*5.75*	2.9	*0.91*	2.4	*2.67*
		Header	+	0.096	0.018	*0.19*	0.6	*0.19*	*1*	0.3	*0.75*	2.3	*0.72*	0.7	*0.78*
		Footer	-	0.095	0.081	*0.84*	99.9	*31.01*	--	2.3	*5.75*	3.7	*1.16*	2.6	*2.89*
		Hdr + Ftr	+ / -												
	LP_INV1.2 t; -; +; t	Nom.		0.021	0.021	--	0.7	--		0.3	--	3.0	--	0.8	--
		Header	t	0.021	0.020	*0.95*	0.7	*1*	*1*	0.3	*1*	2.3	*0.77*	0.7	*0.88*
		Footer	t	0.021	0.018	*0.86*	4.3	*6.05*	--	1.5	*5.00*	2.9	*0.97*	1.8	*2.25*
		Hdr + Ftr	t	0.021	0.017	*0.81*	4.0	*5.71*	--	1.5	*5.00*	2.4	*0.80*	1.7	*2.13*
		Header	vdd	0.021	0.020	*0.95*	0.7	*1*	*1*	0.2	*0.67*	2.2	*0.73*	0.6	*0.75*
		Footer	0	0.021	0.017	*0.81*	15.7	*22.31*	--	1.7	*5.67*	3.0	*1*	2.0	*2.50*
		Hdr + Ftr	vdd / 0	0.021	0.016	*0.76*	14.8	*21.00*	--	1.8	*6.00*	2.4	*0.80*	1.9	*2.38*
		Header	+	0.021	0.018	*0.86*	0.6	*0.86*	*1*	0.2	*0.67*	2.2	*0.73*	0.6	*0.75*
		Footer	-												
		Hdr + Ftr	+ / -												

* The average energy is calculated as 80% of the read energy plus 20% of the write energy.

$^+$ Number of 32×1024 arrays "*N*" required for the total EDP (for one active array and *N-1* in sleep-mode) to be less than or equal to the EDP of *N* arrays of the nominal cell.

Table C.17. Leakage, delay, and EDP 0.6 V V_{DD} results of headers/footers per four cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP^{+}	32 bits × 1024 words Array					
										Delay (ps)				Ave* EDP (ps×fJ)	
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Val	*Comp*
6T	SG t; t; t	Nom.		0.499	0.499	--	25.5	--		51	--	5	--	57	--
		Header	t	0.499	0.434	*0.87*	22.2	*0.87*	*1*	51	*1*	9	*1.80*	48	*0.84*
		Footer	t	0.499	0.142	*0.28*	61.2	*2.40*	--	148	*2.90*	5	*1*	339	*5.95*
		Hdr + Ftr	t	0.499	0.078	*0.16*	33.1	*1.30*	--	147	*2.88*	9	*1.80*	328	*5.75*
		Header	vdd	0.499	0.423	*0.85*	20.8	*0.81*	*19*	50	*0.98*	123	*24.60*	192	*3.37*
		Footer	0	0.499	0.134	*0.27*	622.3	*24.39*	--	486	*9.53*	5	*1*	2778	*48.74*
		Hdr + Ftr	vdd / 0	0.499	0.053	*0.11*	245.1	*9.61*	--	485	*9.51*	125	*25.00*	2734	*47.96*
		Header	+	0.501	0.423	*0.85*	20.0	*0.78*	*98*	49	*0.96*	294	*58.80*	949	*16.65*
		Footer	-												
		Hdr + Ftr	+ / -												
8T	SG 3 x t; t; t; t	Nom.		0.498	0.498	--	24.5	--		50	--	4	--	43	--
		Header	t	0.498	0.434	*0.87*	21.3	*0.87*	*1*	50	*1*	12	*3.00*	38	*0.88*
		Footer	t	0.497	0.142	*0.29*	56.3	*2.30*	--	142	*2.84*	4	*1*	288	*6.70*
		Hdr + Ftr	t	0.497	0.078	*0.16*	30.9	*1.26*	--	142	*2.84*	12	*3.00*	279	*6.49*
		Header	vdd	0.498	0.423	*0.85*	20.8	*0.85*	*35*	50	*1*	149	*37.25*	248	*5.77*
		Footer	0	0.496	0.129	*0.26*	493.3	*20.15*	--	441	*8.82*	4	*1*	2234	*51.95*
		Hdr + Ftr	vdd / 0												
		Header	+	0.469	0.423	*0.85*	20.8	*0.85*	*149*	50	*1*	31	*7.75*	948	*22.05*
		Footer	-												
		Hdr + Ftr	+ / -												
	LP_SGR -; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	32	--	13	--
		Header	t												
		Footer	t	0.018	0.016	*0.89*	6.3	*7.17*	--	142	*2.84*	31	*0.97*	58	*4.46*
		Hdr + Ftr	t												
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*343*	50	*1*	255	*7.97*	97	*7.46*
		Footer	0	0.018	0.015	*0.83*	58.9	*66.60*	--	447	*8.94*	31	*0.97*	253	*19.46*
		Hdr + Ftr	vdd / 0	0.018	0.012	*0.67*	47.1	*53.28*	--	447	*8.94*	254	*7.94*	248	*19.08*
		Header	+												
		Footer	-												
		Hdr + Ftr	+ / -												
	LP_INV1 t; -; vdd; t	Nom.		0.081	0.081	--	4.0	--		50	--	10	--	14	--
		Header	t	0.081	0.027	*0.33*	1.3	*0.33*	*1*	50	*1*	17	*1.70*	13	*0.93*
		Footer	t	0.081	0.078	*0.96*	31.4	*7.88*	--	143	*2.86*	10	*1*	79	*5.64*
		Hdr + Ftr	t	0.081	0.024	*0.30*	9.6	*2.42*	--	143	*2.86*	17	*1.70*	79	*5.64*
		Header	vdd	0.081	0.015	*0.19*	0.7	*0.19*	*7*	50	*1*	122	*12.20*	45	*3.21*
		Footer	0	0.080	0.078	*0.96*	307.8	*77.31*	--	448	*8.96*	9	*0.90*	485	*34.64*
		Hdr + Ftr	vdd / 0	0.080	0.012	*0.15*	47.4	*11.89*	--	448	*8.96*	125	*12.50*	473	*33.79*
		Header	+	0.079	0.015	*0.19*	0.7	*0.18*	*30*	49	*0.98*	283	*28.30*	167	*11.93*
		Footer	-												
		Hdr + Ftr	+ / -												
	LP_INV1.2 t; -; +; t	Nom.		0.018	0.018	--	0.9	--		50	--	18	--	9	--
		Header	t	0.018	0.018	*1*	0.9	*1*	*1*	50	*1*	26	*1.44*	9	*1*
		Footer	t	0.018	0.016	*0.89*	6.3	*7.17*	--	142	*2.84*	18	*1*	49	*5.44*
		Hdr + Ftr	t	0.018	0.015	*0.83*	5.9	*6.72*	--	142	*2.84*	26	*1.44*	49	*5.44*
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*67*	50	*1*	118	*6.56*	25	*2.78*
		Footer	0	0.018	0.015	*0.83*	58.9	*66.60*	--	447	*8.94*	19	*1.06*	222	*24.67*
		Hdr + Ftr	vdd / 0	0.018	0.012	*0.67*	47.1	*53.28*	--	447	*8.94*	121	*6.72*	213	*23.67*
		Header	+	0.018	0.015	*0.83*	0.7	*0.80*	*192*	49	*0.98*	271	*15.06*	65	*7.22*
		Footer	-												
		Hdr + Ftr	+ / -												

Table C.18. Energy 0.6 V V_{DD} results of headers/footers per four cells

	Scheme *Pass; Inv-N; Inv-P; Read*	Leak. Tran.	Leak. Tran. Config.	Leak. / Cell (nA)	Leakage / Cell in Sleep-Mode (nA)		Leakage EDP / Array in Sleep-Mode (ps×fJ)		# Arrays for Break-Even EDP+	32 bits × 1024 words Array					
										Energy (fJ)					
				Val	Val	*Comp*	Val	*Comp*		Rd	*Comp*	Wr	*Comp*	Ave*	*Comp*
6T	SG t; t; t	Nom.		0.499	0.499	--	25.5	--		0.9	--	2.1	--	1.1	--
		Header	t	0.499	0.434	*0.87*	22.2	*0.87*	*1*	0.8	*0.89*	1.5	*0.71*	1.0	*0.91*
		Footer	t	0.499	0.142	*0.28*	61.2	*2.40*	--	2.1	*2.33*	2.9	*1.38*	2.3	*2.09*
		Hdr + Ftr	t	0.499	0.078	*0.16*	33.1	*1.30*	--	2.2	*2.44*	2.4	*1.14*	2.2	*2.00*
		Header	vdd	0.499	0.423	*0.85*	20.8	*0.81*	*19*	1.5	*1.67*	1.6	*0.76*	1.6	*1.45*
		Footer	0	0.499	0.134	*0.27*	622.3	*24.39*	--	5.6	*6.22*	6.2	*2.95*	5.7	*5.18*
		Hdr + Ftr	vdd / 0	0.499	0.053	*0.11*	245.1	*9.61*	--	5.7	*6.33*	5.4	*2.57*	5.6	*5.09*
		Header	+	0.501	0.423	*0.85*	20.0	*0.78*	*98*	3.2	*3.56*	3.2	*1.52*	3.2	*2.91*
		Footer	-												
		Hdr + Ftr	+ / -												
8T	SG 3 x t; t; t; t	Nom.		0.498	0.498	--	24.5	--		0.6	--	2.0	--	0.9	--
		Header	t	0.498	0.434	*0.87*	21.3	*0.87*	*1*	0.6	*1*	1.6	*0.80*	0.8	*0.89*
		Footer	t	0.497	0.142	*0.29*	56.3	*2.30*	--	1.8	*3.00*	2.8	*1.40*	2.0	*2.22*
		Hdr + Ftr	t	0.497	0.078	*0.16*	30.9	*1.26*	--	1.9	*3.17*	2.5	*1.25*	2.0	*2.22*
		Header	vdd	0.498	0.423	*0.85*	20.8	*0.85*	*35*	1.6	*2.67*	2.0	*1*	1.7	*1.89*
		Footer	0	0.496	0.129	*0.26*	493.3	*20.15*	--	4.9	*8.17*	5.7	*2.85*	5.1	*5.67*
		Hdr + Ftr	vdd / 0												
		Header	+	0.469	0.423	*0.85*	20.8	*0.85*	*149*	3.0	*5.00*	3.2	*1.60*	3.0	*3.33*
		Footer	-												
		Hdr + Ftr	+ / -												
	LP_SGR -; -; +; t	Nom.		0.018	0.018	--	0.9	--		0.1	--	0.9	--	0.3	--
		Header	t												
		Footer	t	0.018	0.016	*0.89*	6.3	*7.17*	--	0.3	*3.00*	1.0	*1.11*	0.4	*1.33*
		Hdr + Ftr	t												
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*343*	0.3	*3.00*	0.8	*0.89*	0.4	*1.33*
		Footer	0	0.018	0.015	*0.83*	58.9	*66.60*	--	0.4	*4.00*	1.1	*1.22*	0.6	*2.00*
		Hdr + Ftr	vdd / 0	0.018	0.012	*0.67*	47.1	*53.28*	--	0.5	*5.00*	1.0	*1.11*	0.6	*2.00*
		Header	+												
		Footer	-												
		Hdr + Ftr	+ / -												
	LP_INV1 t; -; vdd; t	Nom.		0.081	0.081	--	4.0	--		0.1	--	0.8	--	0.3	--
		Header	t	0.081	0.027	*0.33*	1.3	*0.33*	*1*	0.1	*1*	0.7	*0.88*	0.3	*1*
		Footer	t	0.081	0.078	*0.96*	31.4	*7.88*	--	0.5	*5.00*	0.9	*1.13*	0.6	*2.00*
		Hdr + Ftr	t	0.081	0.024	*0.30*	9.6	*2.42*	--	0.5	*5.00*	0.8	*1*	0.6	*2.00*
		Header	vdd	0.081	0.015	*0.19*	0.7	*0.19*	*7*	0.3	*3.00*	0.6	*0.75*	0.4	*1.33*
		Footer	0	0.080	0.078	*0.96*	307.8	*77.31*	--	1.0	*10.00*	1.4	*1.75*	1.1	*3.67*
		Hdr + Ftr	vdd / 0	0.080	0.012	*0.15*	47.4	*11.89*	--	1.0	*10.00*	1.2	*1.50*	1.1	*3.67*
		Header	+	0.079	0.015	*0.19*	0.7	*0.18*	*30*	0.6	*6.00*	0.7	*0.88*	0.6	*2.00*
		Footer	-												
		Hdr + Ftr	+ / -												
	LP_INV1.2 t; -; +; t	Nom.		0.018	0.018	--	0.9	--		0.1	--	0.6	--	0.2	--
		Header	t	0.018	0.018	*1*	0.9	*1*	*1*	0.1	*1*	0.6	*1*	0.2	*1*
		Footer	t	0.018	0.016	*0.89*	6.3	*7.17*	--	0.3	*3.00*	0.6	*1*	0.3	*1.50*
		Hdr + Ftr	t	0.018	0.015	*0.83*	5.9	*6.72*	--	0.3	*3.00*	0.6	*1*	0.3	*1.50*
		Header	vdd	0.018	0.015	*0.83*	0.7	*0.83*	*67*	0.2	*2.00*	0.4	*0.67*	0.2	*1*
		Footer	0	0.018	0.015	*0.83*	58.9	*66.60*	--	0.4	*4.00*	0.7	*1.17*	0.5	*2.50*
		Hdr + Ftr	vdd / 0	0.018	0.012	*0.67*	47.1	*53.28*	--	0.4	*4.00*	0.6	*1*	0.5	*2.50*
		Header	+	0.018	0.015	*0.83*	0.7	*0.80*	*192*	0.2	*2.00*	0.4	*0.67*	0.2	*1*
		Footer	-												
		Hdr + Ftr	+ / -												

Appendix D

An Exploration into FinFET SRAM Thermal Performance

A potential downside to FinFET devices is they suffer from problems with self-heating. FinFETs are often manufactured as a silicon-on-insulator (SOI) device which increases its thermal isolation, thus reducing the ability of the devices to dissipate heat through the substrate. The geometry of FinFETs is also an obstacle in dealing with self-heating. The vertical structure of FinFETs allows the devices to be more closely packed, reducing area over planar devices but also reducing heat dissipation. A FinFET with many fins experiences an increased temperature for the innermost fins as these are thermally insulated by the surrounding fins [44]. As any device's temperature increases, the operational properties of the device also changes.

At the system level, hotter devices lead to a number of issues. Power dissipation increases, especially leakage power which is troublesome for SRAMs. Delay also increases, thus overall energy consumption is greatly increased. Over time, hot devices may wear out and fail, leading to irreversible system failure unless redundancy is in place. The following subsections present the simulation environment and the simulation summary for an attempt to quantify the thermal performance of the FinFET 8T SG SRAM cell.

D.1 Electro-Thermal Co-Simulation Environment

FinFET device temperatures can be calculated from the device current and the thermal resistances and capacitances of the FinFETs by using one of a few available models [44] [45] [46] [46]. In this exploration, a simplification of the self-heating equivalent circuit's transfer function from [46] was used to calculate the change in FinFET temperature; this equation is:

$$\Delta T = I_{device} * V_{DD} * R_{th} * e^{-\Delta t/(R_{th} * C_{th})}$$

where ΔT is the change in FinFET temperature, Δt is the length of time or time-step length, I_{device} is the average current through a FinFET during Δt, V_{DD} is the supply voltage, R_{th} is the thermal resistance, and C_{th} is the thermal capacitance. The values of R_{th} and C_{th} are 5000 K/W and 1×10^{-12} W*s/K, respectively, and were obtained from [47].

Exploratory electro-thermal co-simulation was performed by substituting the effects of increased device temperature back into the electrical simulation for a piecewise transient simulation. Due to the limited functionality of the simulator, the global temperature of the SRAM array is updated to reflect the maximum temperature increase experienced by a FinFET in the circuit. A time-step was chosen to determine how often to evaluate the device temperatures for the electrical transient analysis. At each evaluation, the global temperature was updated to simulate the impact of increasing device temperature, and the transient analysis was then allowed to resume. This is a relatively simple, albeit lower accuracy setup. It may still suffice as an estimation however, as only one SRAM cell is simulated at a time and the device self-heating may be similar for the FinFETs in such a small circuit. The following subsection includes the Perl script used to run the thermal simulations.

D.1.1 run_therm_ufdg.pl Perl Script for Thermal FinFET Simulations

```
#!/usr/bin/perl -w
use strict;
use warnings;

my $Version_Date = 'Mar. 23, 2013';
my $Script_Name = 'run_therm_ufdg.pl';

# Requires script:
my $Run_Ufdg = 'run_ufdg.pl';

# Constants:
my $Default_Run_Opts = '-c';
my $Netlist_Suffix = '.i';
my $Tcl_Name = 'meas_8t_thermal.tcl';
my $Rtherm = 5e3;
my $Ctherm = 1e-12;

# Usage:
my $Usage = ''.
'Usage: '.$Script_Name."  deck_name\n";
my $Min_Arg_Num = 1;

###

print "**$Script_Name version date: $Version_Date**\n";
die $Usage if scalar @ARGV < 1;
my $argnum = scalar @ARGV;
die "Error: Not enough parameters\n\n$Usage" if $argnum < $Min_Arg_Num;

my $i = 0;
my $arg = shift;
while( $arg =~ m/^-/ ) {
  die "Error: Not enough parameters\n\n$Usage" if $argnum < (++$i + $Min_Arg_Num);
  if( $arg =~ m/^-(\S+)/ ) { # Found options
    my $argmatch = $1;

    # Check single letter options first

    # Check options which require another argument
  }
  $arg = shift;
}
my $deckname = $arg;

open IN, "<$deckname$Netlist_Suffix" or die 'Error: Cannot open ',
             $deckname.$Netlist_Suffix, ' due to: ', $!;
my @netlist = <IN>;
close IN;

my @grep_lines = grep /thermsimlen/i, @netlist;
my $thermsimlen = 0;
if( $grep_lines[0] =~ m/\.param\s+thermsimlen\s*=\s*(\d+)/i ) {
  $thermsimlen = $1;
}
@grep_lines = grep /thermstep/i, @netlist;
my $thermstep = 0;
if( $grep_lines[0] =~ m/\.param\s+thermstep\s*=\s*(\d+)/i ) {
  $thermstep = $1;
}
@grep_lines = grep /tempc/i, @netlist;
my $temp = 0;
if( $grep_lines[0] =~ m/\.param\s+tempc\s*=\s*(\d+)/i ) {
  $temp = $1;
}
@grep_lines = grep /vdd_val/i, @netlist;
my $vdd_val = 0;
if( $grep_lines[0] =~ m/\.param\s+vdd_val\s*=\s*(\d+)/i ) {
  $vdd_val = $1;
}

my $simbegt;
my $simendt;
my $num_thermsteps = $thermsimlen / $thermstep;
my $tempinc;
for( $i = 0; $i < $num_thermsteps-1; $i++ ) {
  $simbegt = $i * $thermstep;
  foreach (@netlist) {
    s/\.param\s+simbegt\s*=\s*\d+/.param simbegt = $simbegt/i;
  }
  $simendt = ($i+1) * $thermstep;
  foreach (@netlist) {
    s/\.param\s+simendt\s*=\s*\d+/.param simendt = $simendt/i;
  }

  open OUT, ">$deckname$i$Netlist_Suffix" or die 'Error: Cannot open ',
            $deckname.$i.$Netlist_Suffix, ' due to: ', $!;
  print OUT @netlist;
  close OUT;
  system( ($Run_Ufdg, $Default_Run_Opts, $deckname.$i) ) == 0 or die "Error:
            \"$Run_Ufdg $Default_Run_Opts $deckname$i\" failed";

  open EZWAVE, "echo \"$deckname$i\" | run_wdb_server -do $Tcl_Name |" or die 'Error:
            Cannot run EZWave with ', $Tcl_Name;
  my $data_names_str = <EZWAVE>; # Measurement names are on the first line of TCL
            output, data values are on subsequent lines
  #my @data_names = split /[,\n]/, $data_names_str;
  my $tcl_data = <EZWAVE>;
  close EZWAVE;
  my @data = split /,/, $tcl_data;
  print $i, '-->', $data[-1], "\n";

  $tempinc = $vdd_val*$data[-1]*$Rtherm*exp(-1*$thermstep*1e-12/($Rtherm*$Ctherm));
  print $i, '-->', $tempinc, "\n";
  $temp += $tempinc;
  foreach (@netlist) {
    s/\.param\s+tempc\s*=\s*\d+\.?\d*/.param tempc = $temp/i;
  }
}

$simbegt = $i * $thermstep;
foreach (@netlist) {
  s/\.param\s+simbegt\s*=\s*\d+/.param simbegt = $simbegt/i;
}
$simendt = $thermsimlen;
foreach (@netlist) {
  s/\.param\s+simendt\s*=\s*\d+/.param simendt = $simendt/i;
}

open OUT, ">$deckname$i$Netlist_Suffix" or die 'Error: Cannot open ',
            $deckname.$i.$Netlist_Suffix, ' due to: ', $!;
print OUT @netlist;
close OUT;

system( ($Run_Ufdg, $Default_Run_Opts, $deckname.$i) ) == 0 or die "Error: \"$Run_Ufdg
            $Default_Run_Opts $deckname$i\" failed";

open EZWAVE, "echo \"$deckname$i\" | run_wdb_server -do $Tcl_Name |" or die 'Error:
            Cannot run EZWave with ', $Tcl_Name;
my $data_names_str = <EZWAVE>; # Measurement names are on the first line of TCL
            output, data values are on subsequent lines
#my @data_names = split /[,\n]/, $data_names_str;
my $tcl_data = <EZWAVE>;
close EZWAVE;
my @data = split /,/, $tcl_data;
print $i, '-->', $data[-1], "\n";

$tempinc = $vdd_val*$data[-1]*$Rtherm*exp(-1*($simendt-$simbegt)*1e-
            12/($Rtherm*$Ctherm));
print $i, '-->', $tempinc, "\n";
$temp += $tempinc;
```

D.2 Simulation Summary

Thermal simulations were attempted for the 8T SRAM SG cell. At a V_{DD} of 1 V and a nominal ambient temperature of 27°C, a time-step of 50 ps was used for the simulation to reevaluate and update the global temperature of a transient analysis which included read and write operations to one cell. At each time-step, a change in temperature of between 3 and 4 μK was calculated. However, unfortunately, this change in temperature was too small, by trial-and-error it seems that at least a 100 μK change in temperature is required, to register any change with the simulator—the simulator rounded this temperature to the original value, e.g. for a simulation beginning at 27°C, the first time-step would reevaluate and update the global temperature to 27.000004°C, however, the simulator then rounds this temperature back to 27°C effectively negating the attempt at electro-thermal simulation.

If the changes in temperature at each time-step could be preserved, then over the course of the simulation a larger overall change in temperature would be witnessed and the performance of the SRAM array could be noticeably affected. However, for this to happen in a future research project, a different simulator or electro-thermal simulation environment must be used. More advanced Spice simulators, such as hSpice, have the capability to accommodate this, as well as device simulators.

Appendix E

Publications

The following subsections list the published journal and conference publications produced thus far while working on the research presented in this dissertation.

E.1 Journal Publications

1. M. A. Turi and J. G. Delgado-Frias, “Decreasing energy consumption in address decoders by means of selective precharge schemes,” *Microelectronics Journal*, vol. 40, no. 11, pp. 1590-1600, Nov. 2009.
   (*Invited Paper* for *Special Issue: Digital and Mixed-Signal Circuits and Systems* from MWSCAS 2007 publication)

2. M. A. Turi and J. G. Delgado-Frias, “High-performance low-power selective precharge schemes for address decoders,” *IEEE Transactions on Circuits and Systems II, Express Briefs*, vol. 55, no. 9, pp. 917-921, Sept. 2008.

E.2 Conference Publications

1. C. M. Gerik, M. A. Turi, and J. G. Delgado-Frias, “FinFET 3T and 3T1D dynamic RAM cells,” in *Proceedings of 55th IEEE International Midwest Symposium on Circuits and Systems*, pp. 454-457, Aug. 5-8, 2012, Boise, ID.

2. Z. Zhang, M. A. Turi, and J. G. Delgado-Frias, “SRAM leakage in CMOS, FinFET, and CNTFET technologies,” in *Proceedings of 22nd ACM Great Lakes Symposium on VLSI*, pp. 267-270, May 3-4, 2012, Salt Lake City, UT.

3. M. A. Turi and J. G. Delgado-Frias, “Performance-power tradeoffs of 8T FinFET SRAM cells,” in *Proceedings of 54th IEEE International Midwest Symposium on Circuits and Systems*, Aug. 7-10, 2011, Seoul, South Korea.

4. J. G. Delgado-Frias, Z. Zhang, and M. A. Turi, “Low-power SRAM cell design for FinFET and CNTFET technologies,” in *Proceedings of 1st IEEE Workshop of Low Power SoC in International Green Computing Conference*, pp. 547-553, Aug. 18, 2010, Chicago, IL.

5. M. A. Turi, J. G. Delgado-Frias, and N. K. Jha, “Low-power FinFET design schemes for NOR address decoders,” in *Proceedings of 2010 IEEE International Symposium on VLSI Design, Automation, and Test*, pp. 74-77, Apr. 26-29, 2010, Hsinchu, Taiwan.

6. M. A. Turi and J. G. Delgado-Frias, “High-performance low-power AND and sense-amp address decoders with selective precharging,” in *Proceedings of 2008 IEEE International Symposium on Circuits and Systems*, pp. 1464-1467, May 18-21, 2008, Seattle, WA.

7. M. A. Turi and J. G. Delgado-Frias, “Reducing power in memory decoders by means of selective precharge schemes,” in *Proceedings of 50th IEEE International Midwest Symposium on Circuits and Systems*, pp. 956-959, Aug. 5-8, 2007, Montréal, Canada.