University of Kentucky

UKnowledge

Theses and Dissertations--Electrical and Computer Engineering

Electrical and Computer Engineering

2015

A METHODOLOGY OF SPICE SIMULATION TO EXTRACT SRAM SETUP AND HOLD TIMING PARAMETERS BASED ON DFF DELAY DEGRADATION

Xiaowei Zhang
University of Kentucky, wakenaway@hotmail.com

Right click to open a feedback form in a new tab to let us know how this document benefits you.

Recommended Citation

Zhang, Xiaowei, "A METHODOLOGY OF SPICE SIMULATION TO EXTRACT SRAM SETUP AND HOLD TIMING PARAMETERS BASED ON DFF DELAY DEGRADATION" (2015). *Theses and Dissertations--Electrical and Computer Engineering*. 75.
https://uknowledge.uky.edu/ece_etds/75

This Master's Thesis is brought to you for free and open access by the Electrical and Computer Engineering at UKnowledge. It has been accepted for inclusion in Theses and Dissertations--Electrical and Computer Engineering by an authorized administrator of UKnowledge. For more information, please contact UKnowledge@lsv.uky.edu.

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student’s advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student’s thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Xiaowei Zhang, Student

Dr. Joseph A. Elias, Major Professor

Dr. Caicheng Lu, Director of Graduate Studies

A METHODOLOGY OF SPICE SIMULATION
TO EXTRACT SRAM SETUP AND HOLD TIMING PARAMETERS
BASED ON DFF DELAY DEGRADATION

THESIS

A Thesis Submitted in Partial Fulfillment of the
Requirements for the degree of Master of Science in Electrical Engineering
in the College of Engineering
at the University of Kentucky

By

Xiaowei Zhang
Lexington, Kentucky

Directors: Dr. Joseph A. Elias, Adjunct Professor of Department of Electrical and Computer Engineering
Dr. Zhi D. Chen, Professor of Department of Electrical and Computer Engineering

Lexington, Kentucky
2015

ABSTRACT OF THESIS

A METHODOLOGY OF SPICE SIMULATION TO EXTRACT SRAM SETUP AND HOLD TIMING PARAMETERS BASED ON DFF DELAY DEGRADATION

SRAM is a significant component in high speed computer design, which serves mainly as high speed storage elements like register files in microprocessors, or the interface like multiple-level caches between high speed processing elements and low speed peripherals. One method to design the SRAM is to use commercial memory compiler. Such compiler can generate different density/speed SRAM designs with single/dual/multiple ports to fulfill design purpose. There are discrepancy of the SRAM timing parameters between extracted layout netlist SPICE simulation vs. equation-based Liberty file (.lib) by a commercial memory compiler. This compiler takes spec values as its input and uses them as the starting points to generate the timing tables/matrices in the .lib. Originally large spec values are given to guarantee design correctness. While such spec values are usually too pessimistic when comparing with the results from extracted layout SPICE simulation, which serves as the "golden" rule. Besides, there is no margin information built-in such .lib generated by this compiler.

A new methodology is proposed to get accurate spec values for the input of this compiler to generate more realistic matrices in .lib, which will benefit during the integration of the SRAM IP and timing analysis.

KEYWORDS: SRAM, Timing Parameters, SPICE, Liberty File, DFF

Xiaowei Zhang

05/25/2015

A METHODOLOGY OF SPICE SIMULATION
TO EXTRACT SRAM SETUP AND HOLD TIMING PARAMETERS
BASED ON DFF DELAY DEGRADATION

By
Xiaowei Zhang

Dr. Joseph A. Elias

Director of Thesis

Dr. Zhi D. Chen

Director of Thesis

Dr. Caicheng Lu

Director of Graduate Studies

05/25/2015

Acknowledgments

I would like to thank my thesis advisor, Dr. Joseph A. Elias, for all the guidance and help I have received from him. He was always patient and willing to answer my questions, work with me to figure out the solutions. Besides, he always pointed out the right direction to me about my research. This thesis would be impossible without his extensive knowledge and innovative ideas in the VLSI field.

I would also like to thank Dr. Zhi D. Chen and Dr. Himanshu Thapliyal for serving as committee members, and for the insightful guidance I have received from them.

Last but not least, I would like to express my deepest gratitude to my parents, for the endless love and support I have always been with since I was born.

Table of Contents

Acknowledgments iii
Table of Contents iv
List of Figures vii
List of Tables IX
Chapter 1 Introduction 1
Chapter 2 Literature Review 4
2.1 0.35um Technology Node 4
2.2 0.25um Technology Node 5
2.3 0.18um Technology Node 7
2.4 0.15um Technology Node 8
2.5 0.13um Technology Node 8
Chapter 2 DFF Metastability 10
2.1 D Flip-Flop 10
2.2 Setup and Hold Times of DFF 11
2.3 Static Timing Analysis (STA) of DFF 11
2.4 Metastability 12
Chapter 3 A Semiconductor Firm's SRAM Design 14
3.1 Introduction to A Vendor's Memory Compiler 14
3.2 Design Automation Using Script Languages 16
Chapter 4 Data Input Setup Time (tSDI) 17
4.1 Equation 17
4.2 Schematic 18
4.3 Logic of WE Signal 21
4.4 Stimulus Waveforms 21
4.5 Methodology 23
4.6 Optimization (PassFail vs. Dichotomy) 23
4.7 General Procedures (Vary PVTs) 24
4.8 Results 24
4.8.1 Rising Polarity 25
4.8.2 Falling Polarity 27
Chapter 5 Data Input Hold Time (tHDI) 28
5.1 Equation 28
5.2 Schematic 29
5.3 Stimulus Waveforms 29
5.4 Methodology 30

5.5 Results 31
5.5.1 Rising Polarity 32
5.5.2 Falling Polarity 34
Chapter 6 Data Writing Delay (tWR) 35
6.1 Equation 35
6.2 Schematic 36
6.3 Results 36
6.4 Validation 37
Chapter 7 Read/Write Setup Time (tSRWB) 37
7.1 Equation 37
7.2 Schematic 38
7.3 Pre-charge Latch 39
7.4 Delay Degradations of Normal DFF and Pre-charge Latch 40
7.5 Individual Simulation of Pre-charge Latch without Other Circuits 41
7.5.1 Individual Simulation vs. Extracted Layout Simulation 41
7.5.2 Varying Output Load Capacitance 42
7.5.3 Varying the W/L of PMOS I76 and I77 43
7.5.4 Varying the Power Supply Voltage V_{DD} 44
7.5.5 Varying the Process 44
7.5.6 Varying the PMOS Model of Output Inverters 45
7.5.7 Schematic vs. Extracted Layout Simulation 47
7.5.8 Different Data Input Polarities 48
7.5.9 Tweak of the Inverter on the Data Input Path 48
7.5.10 A Proposed Improvement of the Inverter on the Data Input Path . 50
7.5.11 Different Versions of the Modified Pre-charge Latch with Pull-down Path 52
7.5.12 Final Top-level Layout of the SRAM 54
7.5.12 Simulation Results of Different Versions of the Modified Layouts 55
7.5.13 The Effect of M Factor on the Delay Degradation Pattern 57
7.6 Stimulus Waveforms 57
7.7 Methodology 58
7.8 Results 58
7.8.1 Default Layout (Rising Polarity) 59
7.8.2 Modified Layout Version 3 (Rising Polarity) 61
7.8.3 Default Layout (Falling Polarity) 63
7.8.4 Modified Layout Version 3 (Falling Polarity) 65
Chapter 8 Data Reading Delay (tRD) 67
8.1 Equation 67

8.2 Schematic 67
8.3 Results 68
8.4 Validating 68
Chapter 9 Final Results 69
Appendix A: Generic Perl Script for Individual DFF Simulation 72
Appendix B: Generic Ruby Script for Individual DFF Simulation 83
References 94
Vita 97
Education 97

List of Figures

Figure 1. A Typical 6T SRAM Cell Configuration 2
Figure 2. A Typical 4T SRAM Cell Configuration 3
Figure 3. Trends in Device Count/Chip and Feature Size of MOS Device 3
Figure 4. SRAM Bit-cell and Minimum-supply-voltage Scaling 4
Figure 5. 10T Cell Using Extra Low-V_{th} NMOS to Accelerate Readout Operations 5
Figure 6. Concept of the HBLSA-SRAM 6
Figure 7. Schematic of a Two-Stage Sense-Amp 7
Figure 8. Equivalent Circuit of a LL4T SRAM Cell and Node Voltages in a Stand-by Cycle . 8
Figure 9. Circuit Diagram of DFF[23] 10
Figure 10. Timing Definition of Setup/Hold Time[23] 11
Figure 11. DFF Environment in a Digital System[24] 11
Figure 12. The Metastability Window[28] 13
Figure 13. Definition of Setup and Hold Times[24] 14
Figure 14. Schematic of the Underlying DFF 17
Figure 15. Schematic of tSDI 18
Figure 16. Schematic from DI to DO 19
Figure 17. Schematic of RBK Block 20
Figure 18. Schematic of RDATA Block 20
Figure 19. Waveforms Indicates Isolation of Delay Degradation 21
Figure 20. Schematic of WE Signal 21
Figure 21. Stimulus Waveforms of tSDI Simulation (SS/1.35V/-40 ℃, Rising) 22
Figure 22. tSDI_spec Simulation Results (Rising) with Varying PVTs 25
Figure 23. tSDI_spec Simulation Results (Falling) with Varying PVTs 27
Figure 24. Schematic of tHDI 29
Figure 25. Stimulus Waveforms of tHDI Simulation (SS/1.35V/-40 ℃, Rising) 30
Figure 26. tHDI_sim Simulation Results (Rising) with Varying PVTs 32
Figure 27. tHDI_sim Simulation Results (Falling) with Varying PVTs 34
Figure 28. Schematic of tWR 36
Figure 29. tWR_spec Simulation Results with Varying Temperature and V_{DD} 37
Figure 30. Schematic of tSRWB 38
Figure 31. Schematic of Normal DFF 39
Figure 32. Schematic of Pre-charge Latch 39
Figure 33. Comparison Between Normal DFF and Pre-charge Latch. 40
Figure 34. (a) Individual Simulation (b)(c) Extracted Layout Simulation 41
Figure 35. Pre-charge Latch Simulation Results with Varying Output Load Capacitance (a)(c) Absolute Value (b)(d) Percentage Value 42
Figure 36. Pre-charge Latch Simulation Results with Varying I76/I77 Width (a) Absolute Value (b) Percentage Value 43
Figure 37. Pre-charge Latch Simulation Results with Varying Vdd (a) Absolute Value (b) Percentage Value 44
Figure 38. Pre-charge Latch Simulation Results with Varying Process (a) Absolute Value (b) Percentage Value 44
Figure 39. Delay Degradation Patterns for Different PMOS Models 45
Figure 40. Waveforms for Different PMOS Models 46
Figure 41. Schematic vs. Extracted Layout Simulation (a) Rising (b) Falling 47
Figure 42. Simulation Results of Different Data Input Polarities (a) Schematic (b) Extracted Layout 48
Figure 43. Simulation Results of Tweaking the Inverter 49
Figure 44. Rising/Falling Simulation Results without the Input Inverter 49
Figure 45. A_N Waveforms with Unchanged Netlist and Netlist without the Inverter 50
Figure 46. Portion of Pre-charge Latch Schematic Shows the Added Pull-down Path 51
Figure 47. Default Layout 52

Figure 48. Modified Layout Version 1 52
Figure 49. Modified Layout Version 2 52
Figure 50. Modified Layout Version 3 53
Figure 51. Final Top-level Layout 54
Figure 52. Zoom-in Layout Shows the Improved Pre-charge Type Latch with Pull-down Path 54
Figure 53. Simulation Results of the Default Layout 55
Figure 54. Simulation Results of the Version 1 55
Figure 55. Simulation Results of the Version 2 56
Figure 56. Simulation Results of the Version 3 56
Figure 57. Simulation Results of Rising/Falling Delay Degradation Patterns with Different M Factors 57
Figure 58. Stimulus Waveforms of tSRWB Simulation (SS/1.35V/-40 ℃, Rising) 58
Figure 59. tSRWB Simulation Results of Default Layout (Rising) with Varying PVTs 59
Figure 60. tSRWB Simulation Results of Modified Layout Version 3 (Rising) with Varying PVTs 61
Figure 61. tSRWB Simulation Results of Default Layout (Falling) with Varying PVTs 63
Figure 62. tSRWB Simulation Results of Modified Layout Version 3 (Falling) with Varying PVTs 65
Figure 63. Schematic of tRD 68
Figure 64. tRD Simulation Results with Varying Temperature and V_{DD} 68

List of Tables

Table 1. Design Summary of MTCMOS SRAM 5
Table 2. Design Summary of HBLSA-SRAM 7
Table 3. Design Summary of DDR CMOS 8
Table 4. Comparison of Different SRAM Designs 9
Table 5. Some Optimistic Values in The .lib 16
Table 6. tHDI_sim Guardband for Different Process Corners 28
Table 7. tWR_spec Values for Different Processes 35
Table 8. Different PMOS Models in Tech Library 45
Table 9. Different Configurations of the Modified Layouts 53
Table 10. tRD_spec Values for Different Processes 67
Table 11. Final Results 69

Chapter 1 Introduction

SRAM is a kind of memory which uses bistable latching circuitry to store binary bit values (logic 0 or 1). Unlike the Dynamic RAM (DRAM) used, like as discrete main memories in PC, SRAM doesn't require periodic refresh to keep the stored bit values. The back-to-back inverters in the SRAM cell keep reinforcing each other as long as the SRAM cell is powered. On the other hand, SRAM is volatile, which means it will lose the stored bit values if the power goes off[1].

Comparing to other kinds of volatile memories (e.g. DRAM), SRAM is fast and expensive, which limits its applications in high capacity, low cost areas. Because of its high performance (e.g. low access time), SRAM is widely utilized as cache memory in microprocessors or microcontrollers (MCUs)[2]. Modern microprocessors have at least two-level caches built in the chip, which serve as an interface between high speed processing elements and low speed peripherals[1]. Besides, SRAM exists in some application specific integrated circuit (ASIC) designs where burst transfers are needed[3].

Except for integrating in System on Chip (SoC), SRAM is also found in many embedded systems used in industrial subsystems, automotive electronics, and etc[4, 5]. Even in many consumer products like digital cameras, cell phones, SRAM can be found, for example, as LCD screen buffers[6].

For timing aspect, there are two different kinds of SRAM: synchronous or asynchronous. The operation of the synchronous SRAM is controlled by the clock edge(s). All operations happen on the clock edge(s). On the other hand, the asynchronous SRAM has no clock input, the data input/output are controlled by address transition.

One of the key elements of the SRAM design is the SRAM cell design. There are different configurations of SRAM cell, which consist of different number of transistors. The typical configuration is 6-transistor (6T) SRAM cell shown in Figure 1[7]:

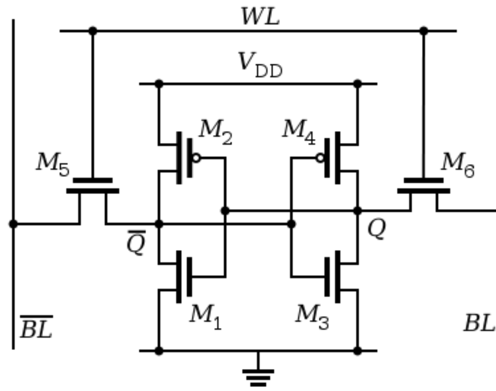

Figure 1. A Typical 6T SRAM Cell Configuration

It can be seen that the transistors M_1 and M_2, M_3 and M_4 form two cross-couple inverters (back-to-back) so that the bit values stored in the Q and Q bar are kept refreshing as long as these two inverters are connected to V_{DD} and GND. The M_5 and M_6 are the access transistors, which serve as the connections between the SRAM cell and the bitlines (BL and BL bar). Both M_5 and M_6 are controlled by the wordline (WL), and if the WL=1, both access transistors are open and the SRAM cell is connected to the bitlines. The SRAM works in reading/writing states. If WL=0, both access transistors are closed and the SRAM cell is isolated. The SRAM works in idle state.

In reading state, suppose a logic 1 (V_{DD}) is stored in the SRAM cell before reading out. The Q is logic 1 and Q bar is logic 0. Before accessing to the SRAM cell, both bitlines are pre-charged to logic 1. Then the WL signal is asserted, which turns on the access transistors M_5 and M_6. Since Q=1 which turns on M_1, the BL bar is discharged through M_5 and M_1 while BL is clamped to V_{DD} for a short period time (a short pulse of WL signal). Once BL and BL bar have enough difference to be amplified by the sense amplifier (sense-amp), the WL signal is off and both access transistor are turned off so that the stored bit value won't be compromised. Depending on which bitline is lower, this small voltage swing will be amplified to full swing by the sense-amp and asserted to output bus.

In writing state, suppose a logic 1 is written into the SRAM cell. The write driver will charge BL to be logic 1, and BL bar to be logic 0. Then the WL signal is asserted and both access transistors are turned on. The Q is connected to BL, which will be charged to logic 1 because the write driver has stronger drive strength than the transistor M_3 and M_4. The same case for Q bar. After that, the WL signal is off and the SRAM cell can keep refreshing the written bit value.

If not in neither reading nor writing states, the SRAM cell is in idle state, where WL=0 turns off both access transistors. The SRAM cell is isolated from outside.

There are many other configurations of the SRAM cell (4T, 8T, 10T, etc.)[8, 9]. Usually the less transistors, the smaller area the SRAM cell will be. A smaller SRAM cell usually results in higher density. One example of the 4T SRAM cell is shown in Figure 2[10]:

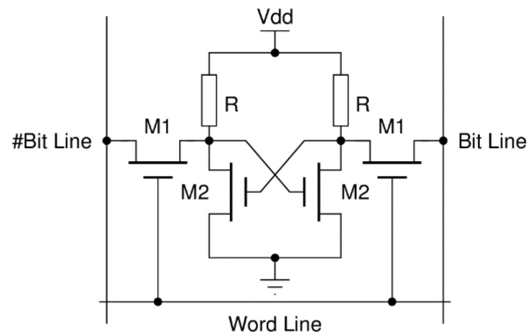

Figure 2. A Typical 4T SRAM Cell Configuration

It can be seen that the two PMOS in the cross-coupled inverters are replaced by polysilicon resistors R, which has higher demand for the process because these two polysilicon resistors have to be small but have large values.

The size of a SRAM is associated with the numbers of address lines and data lines. m address lines means there are 2^m words in this SRAM. And n data lines means each word has n bits, in other words, it is n bit word. So if a SRAM has 11 address lines as well as 8 data lines, the size of this SRAM is 2K x 8bit.

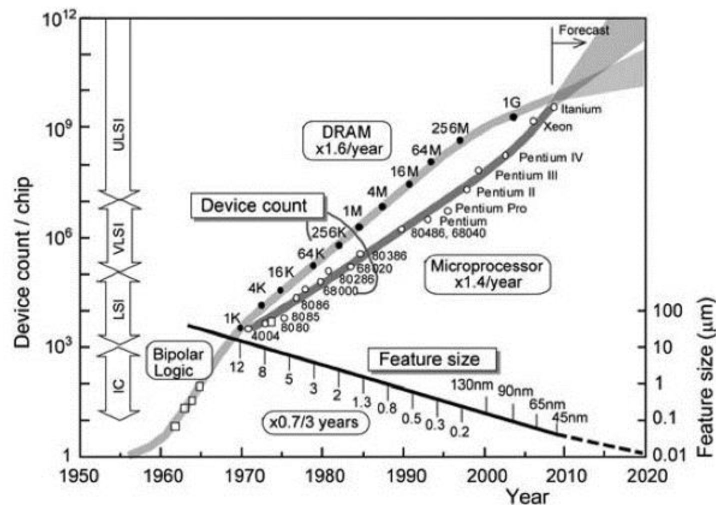

Figure 3. Trends in Device Count/Chip and Feature Size of MOS Device

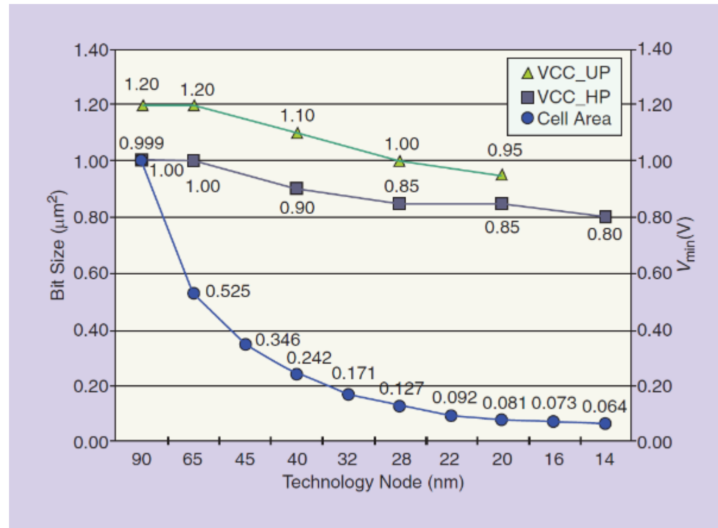

Figure 4. SRAM Bit-cell and Minimum-supply-voltage Scaling

The Figure 3 shows technology node (feature size) trends in semiconductor industry[11], which is getting smaller every year following the Moore's Law. In Figure 4, it can been seen that the finest technology node for SRAM is 14nm now[12]. Both V_{cc} and Bit size are decreasing alongside with the smaller technology nodes.

Chapter 2 Literature Review

2.1 0.35um Technology Node

Shibata et al. proposed a 1V 100MHz MTCMOS SRAM design[13]. In this design, the authors used 0.35um (effective channel length 0.17um) MultiThreshold-voltage CMOS (MTCMOS)/Separation by IMplantation of OXygen (SIMOX) process to fabricate an 8K x 16bit SRAM, which could reach 100MHz working frequency with 1V V_{DD}. In order to reduce the large bitline delay, the low V_{th} transistors were used for logic gates to gain high performance. On the other hand, high V_{th} transistors were used to cut off the sub-threshold leakage current path so that the low power operation could be achieved. A latch type sense-amp was used in this design. In order to increase the working frequency, the authors proposed a pseudo-two-stage pipeline architecture, which featured a sensing delay. For the SRAM cell design, they proposed a 10T SRAM cell configuration (shown in Figure 5), which was 33% larger than conventional 6T SRAM cell. The cell size is 11.2um x 2.8um under their 0.35um MTCMOS/SIMOX process. The cycle time at the worst power supply condition (1V) is 9ns, and the clock access time at single fan-in load is 3.5ns. The summary of their design is shown in Table 1:

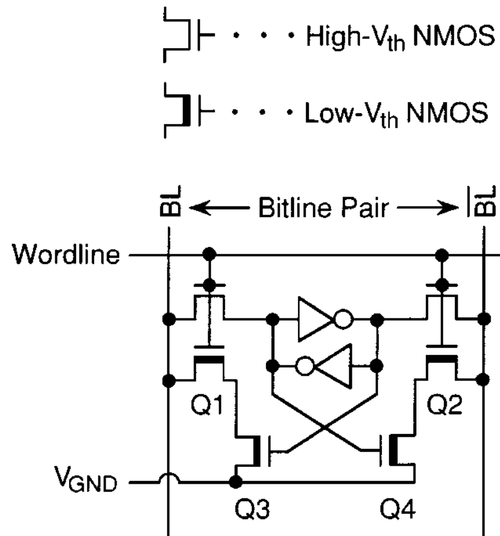

Figure 5. 10T Cell Using Extra Low-V_{th} NMOS to Accelerate Readout Operations

Table 1. Design Summary of MTCMOS SRAM

Chip Size	1.6mm x 3.2mm = 5.12mm^2
SRAM Cell Size	11.2um x 2.8um = 31.36um^2
Organization	8K x 16bit
Minimum Cycle Time (1V)	9ns
Power Dissipation (1.2V, 100MHz)	
Stand by	0.2uW
Read	13.2mW
Write	15.4mW

2.2 0.25um Technology Node

B. D. Yang et al. proposed a low power SRAM design with hierarchical bitlines and local sense-amps (HBLSA-SRAM)[14]. In order to reduce the power dissipation and increase the speed, this HBLSA-SRAM reduced both capacitance and write voltage swing of bitlines by implementing a bitline and sub-bitlines with local sense-amps. The key idea was to apply a low voltage swing (V_{DD}/10=2.5V/10=0.25V) to the high capacitive bitlines and apply a full voltage swing to the low capacitive sub-bitlines. An 8K x 32bit SRAM was fabricated with 0.25um CMOS technology, which consumed 26mW read power and 28mW write power at 253MHz with 2.5V power supply. Unlike read with a small voltage swing in the bitlines, conventional SRAM consumed more power during write cycle due to the full voltage swing in bitlines and

data bus, which both had high capacitance. In order to reduce the voltage swing when write, a hierarchical bitline consisted of a bitline and several sub-bitlines were implemented so that the voltage swing on the bitline was small (kept the same as the voltage swing when read), and only the sub-bitline of the cell accessed connected to the bitline (controlled by a global word line signal GWL bar) had full voltage swing. Once the small voltage swing was transferred to the sub-bitline, a local sense-amp would amplify it to a full voltage swing. Due to the low capacitance of the sub-bitline, the power dissipation of the entire two-stage operation was less than conventional write with a full voltage swing to the bitline. The concept of this HBLSA-SRAM is shown in Figure 6:

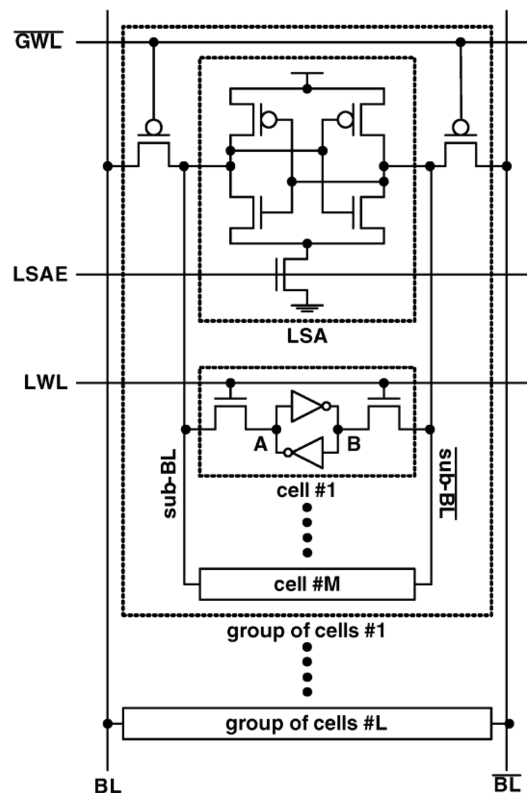

Figure 6. Concept of the HBLSA-SRAM

They used conventional 6T SRAM cell, and two PMOS and a local sense-amp were added to each sub-bitline, which increased the length of bitlines but area overhead was small. They fabricated two SRAM: one was a conventional SRAM, the other was the HBLSA-SRAM, which used the same 0.25um technology. The comparison results showed the HBLSA-SRAM had 18% speed overhead with 8% area overhead, partially because of the 9% longer bitlines. As for the power dissipation, the HBLSA-SRAM saved 34% of the write power of the conventional SRAM, and they had the same read power dissipation. The summary of the HBLSA-SRAM design is shown in Table 2:

Table 2. Design Summary of HBLSA-SRAM

Chip Size	3.26mm x 1.88mm = 6.13mm^2
Organization	8K x 32bit
Supply Voltage	2.5V
Frequency	220MHz
Power Dissipation (200MHz)	
Read	28mW
Write	26mW

2.3 0.18um Technology Node

A. Kawasumi et al. proposed a 18Mbit (1M x 18bit) 1.8V 900MHz DDR CMOS SRAM design with power reduction techniques[15]. The technology node was 4-metal 0.25um with gate length 0.18um. The final SRAM cell size was 2.25um x 2.35um, which leaded to an 11.2mm x 19.0mm chip size. The key design in their SRAM cell was the implementation of two-stage sense-amps in order to reduce the read data bus capacitance, which is shown in Figure 7. A current sense-amp was used for the first stage, which had less dependence on the bitline capacitance. Then a second stage sense-amp was implemented to drive the data bus, which was shared with two first stages so that the number of the second stage sense-amps could be reduced. In their design, the read data bus capacitance was reduced 40%, the active current for sensing was decreased by 33%, and the sensing delay was reduced by 9.6%. The authors declared that this sense-amp configuration was faster than conventional latch type sense-amp.

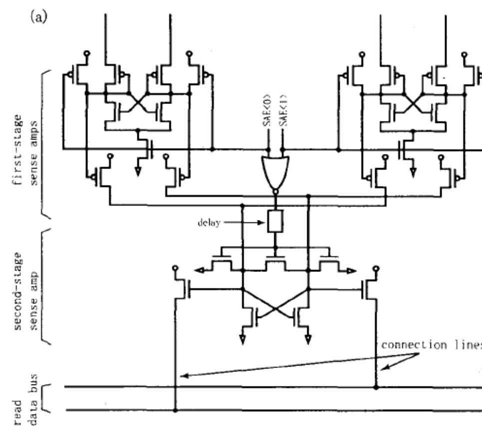

Figure 7. Schematic of a Two-Stage Sense-Amp

Table 3. Design Summary of DDR CMOS

Chip Size	11.2mm x 19.0mm = 212.8mm^2
SRAM Cell Size	2.25um x 2.35um = 5.2875um^2
Organization	1M x 18bit, 512K x 36bit
Supply Voltage	1.8V
Frequency (25 ℃)	900MHz
Power Dissipation (667MHz)	
Read	1.1W
Write	1.3W

2.4 0.15um Technology Node

J. H. Jang et al. proposed a 2.05um^2 (1.3um x 1.58um) CMOS SRAM cell with 0.15um single gate CMOS technology[16]. Their technology had 0.15um for NMOS and 0.17um for PMOS. The final 16Mbit SRAM had a size of 54.13mm^2.

2.5 0.13um Technology Node

S. Masuoka et al. proposed a loadless 4T SRAM cell design (0.99um^2 area: 0.80um x 1.24um) with 0.13um generation CMOS technology[17]. This SRAM cell provided high stable operation at 1.2V from -40 ℃ to 125 ℃. The key design was the loadless 4T (LL4T) SRAM cell, which was shown in Figure 8.

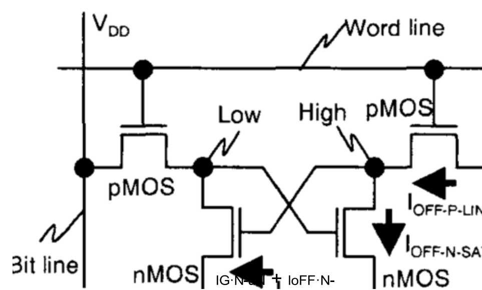

Figure 8. Equivalent Circuit of a LL4T SRAM Cell and Node Voltages in a Stand-by Cycle

This LL4T SRAM cell size was 50-65% of a conventional 6T SRAM cell, which had advantage to reduce the SRAM layout area. Besides, unlike the typical 4T SRAM cell shown in Figure 2, this LL4T SRAM cell didn't require the pull-up resistors, which usually resulted in a challenge for the process. This 0.13um technology node had a 0.12um gate length.

There were many other SRAM designs with various technologies. D. K. Nelson et al. proposed a SOI SRAM design with 0.15um technology node, which had 3-5ns access time under 5ns clock period[18]. Another 4Mbit 1.8V SOI CMOS SRAM (6T SRAM cell configuration) was implemented with 0.2um bulk CMOS process by K. Cox et al. The cell size was 3.77um^2[19]. F. Ootsuka et al. introduced a high density, high performance SRAM design for large scale SoC application under 0.13um CMOS technology with 0.2um gate length[20]. The 6T SRAM cell size was 0.8um x 3.2um = 1.92um^2. Under the same generation process, W. Kong et al. introduced a 6T SRAM cell of 1.87um^2[21]. The comparison of different SRAM designs is shown in Table 4:

Table 4. Comparison of Different SRAM Designs

SRAM Design	Designers	Technology Node (um)	Working Frequency (MHz)	VDD (V)	SRAM Cell Size
MTCMOS SRAM	Shibata et al.	0.35	111	1.00	11.2um x 2.8um = 31.36um^2
HBLSA-SRAM	B. D. Yang et al.	0.25	250	2.50	
SOI CMOS SRAM	K. Cox et al.	0.20		1.80	3.77 um^2
DDR CMOS SRAM	A. Kawasumi et al.	0.18	900	1.80	2.25um x 2.35um = 5.2875um^2
SRAM Cell with Single Gate CMOS Technology	J. H. Jang et al.	0.15			1.3um x 1.58um = 2.05um^2
SOI SRAM	D. K. Nelson et al.	0.15	200		
A Semiconductor Firm's Design		0.15	83.33	1.35	1.2um x 1.58um = 1.896um^2
Loadless LL4T SRAM Cell	S. Masuoka et al.	0.13		1.20	0.80um x 1.24um = 0.99um^2
High Density/Performa nce SRAM	F. Ootsuka et al.	0.13			0.8um x 3.2um = 1.92um^2
6T SRAM Cell	W. Kong et al.	0.13			1.87um^2

R. Castagnetti et al. investigated the effect of different chip level route techniques in order to get high performance SRAM design[22]. The specific route techniques they investigated by fabricating a 6T SRAM cell with 0.18/0.13um technology involved metal 2 (M2) and metal 3

(M3) layers. There were two options for routing: use M2 for horizontal WL and M3 for vertical bitlines and V_{DD} and GND; or use M2 for bitlines and V_{DD} and M3 for WL and GND. What they found was the capacitance of the bitlines dominated the performance of the SRAM cell, and using M2 for the bitlines had 25% bitline capacitance reduction. Besides, the M3 for WL and GND provided good shield for M2 from M4 and above, which leaded to an unrestricted M4 routing. The option of M2 for the bitlines was superior to the other option.

Chapter 2 DFF Metastability

The entire research is about to extract the timing parameters of the SRAM design. Since for the synchronous SRAM, all input signals are captured by the underlying DFFs in the external logic of the SRAM synchronized by the clock, extracting the behaviors of these underlying DFFs, especially setup and hold times, is a method to estimate the setup and hold times of the entire SRAM design.

2.1 D Flip-Flop

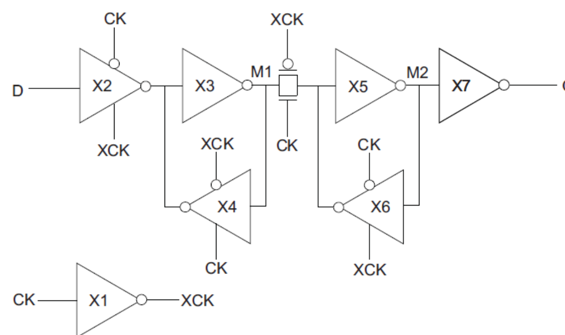

Figure 9. Circuit Diagram of DFF[23]

Figure 9 shows a typical configuration of a master-slave DFF. The master latch consists of the back-to-back inverters X3 and X4, which is controlled by CLK, the same as the slave latch. These two latches are separated by a transmission gate (TG) controlled by CLK. When CLK=0, TG is closed so that both latches are isolated with each other. The X2 is open when CLK=0, so that the data appears on the input D can transmit to node M1. At the same time, X6 is also open controlled by the CLK, then X5 and X6 will enforce each other to hold the previous value Q to the output port. When CLK=1, the TG is open and X2 is closed, so that no more new value can transmit to the DFF, and whatever logic value in node M1 will pass the TG to arrive to X5, and X7, eventually to Q. The CLK will also open X4 and close X6 so that only the master latch has the back-to-back inverters to hold the value.

2.2 Setup and Hold Times of DFF

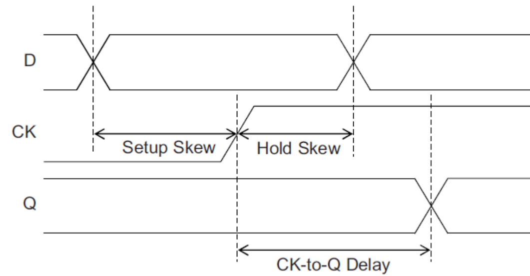

Figure 10. Timing Definition of Setup/Hold Time[23]

For synchronous DFF, the setup time is the minimum amount of time the input data D of the DFF should be stable before the clock CLK trigger edge arrives, so that the data can be reliably sampled and caught by the DFF. The hold time is the minimum amount of time the input data D of the DFF should hold after the clock CLK trigger edge arrives, so that the data can be reliably sampled. The third timing vale is the propagation delay, which measures the delay from the CLK trigger edge to the actual change on its output Q.[23] All three timing parameters of a DFF are shown in Figure 10. If either setup or hold time isn't satisfied, the DFF will enter a state call metastability.

2.3 Static Timing Analysis (STA) of DFF

The typical connection between DFFs is shown below:

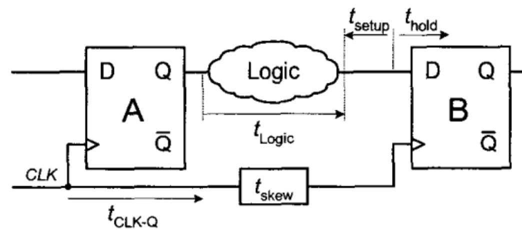

Figure 11. DFF Environment in a Digital System[24]

As shown in Figure 11, the setup and hold should satisfy two equations respectively.[24, 25]

$$t_{CLK\text{-}Q} + t_{setup} \leq T - t_{Logic} - t_{skew}$$

$$t_{CLK\text{-}Q} - t_{hold} \geq t_{skew} - t_{Logic}$$

Equation 1

In Equation 1:

- $t_{CLK\text{-}Q}$ is the propagation delay of the DFF.
- t_{setup} is the setup time of the DFF.
- t_{hold} is the hold time of the DFF.
- T is the clock period.
- t_{Logic} is the delay through the combinational logic between launch and capture DFFs.
- t_{skew} is the delay difference of the clock tree root to the CLK port of the launch and capture DFFs.

In STA of DFF, the worst setup slack ($Slack_{setup}$) and hold slack ($Slack_{hold}$) are calculated by the STA tools by reading the design netlist, cell library and clock period. The setup and hold slacks are defined in Equation 2:

$$Slack_{setup} = T - t_{Logic} - t_{skew} - t_{CLK\text{-}Q} - t_{setup}$$

$$Slack_{hold} = t_{CLK\text{-}Q} - t_{hold} - t_{skew} + t_{Logic}$$

Equation 2

In order to meet the timing requirements of the DFFs in a digital system, or achieving timing closure, the slacks of all datapath should be calculated and positive or 0. If a slack is negative, it's said to be "violated". If a setup slack $Slack_{setup}$ is violated, the circuit can operate correctly by increasing the clock period T, in other words, in lower clock frequency. If a hold slack is violated, the circuit won't function correctly until delay elements are inserted into the short datapaths in the combinational logic between the launch and capture DFFs.[25]

2.4 Metastability

Metastability is a phenomenon where a bi-stable output enters an unstable third state and becomes an intermediate level between logic 0 and 1.[26] DFF is subject to such metastability, when two inputs (D and CLK in our case) are changing at about the same time. The result is the output might behave unpredictably, taking much more time than nominal to settle to one state or the other. As CMOS technology scales, PVT variations and increasing clock frequency all contribute to the possibility of the metastability failure.[27] Such metastability can cause severe problem like corruption of data. This metastability can't be eliminated entirely, because when the D and CLK is closer and closer, the DFF is forced to decide which comes first. No matter how fast the circuit is, there's always a possibility these two input signals are so close to each other than the DFF can't detect which happens first. But as long as the setup and hold times are satisfied, the metastability in DFF can be avoided. So using pre-defined metastability windows to measure the setup and hold times of DFF is a more practical method instead of

looking for the values of setup and hold times that cause the DFF to fail to operate, because a DFF will malfunction long before it starts to completely fail. The metastability window is shown below in:

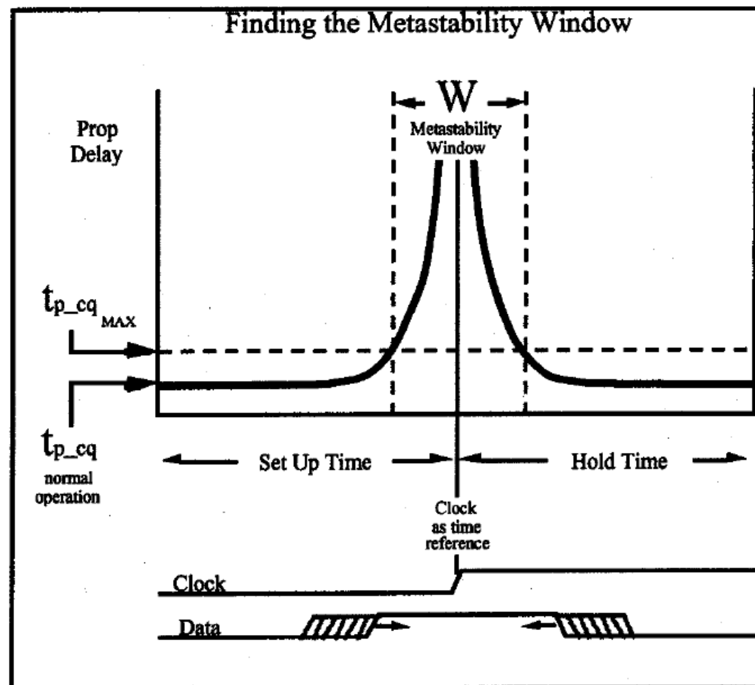

Figure 12. The Metastability Window[28]

The metastability window can be determined by extract the propagation delay t_{CLK-Q} when D is shifting closer to CLK from both direction.[28] First, the nominal value of the propagation delay t_{CLK-Q} can be obtained by extracting under normal operation of the DFF. Then when the D is moving closer to CLK, the propagation delay t_{CLK-Q} will increase exponentially.[26] When the propagation delay t_{CLK-Q} reaches a pre-defined value (normally 10% larger than the nominal value), the DFF is considered to enter metastability. So the edges of metastability window can be consider to be setup and hold times. By reproducing such curves, we can accurately extract the setup and hold timing parameters of a DFF under different PVTs.

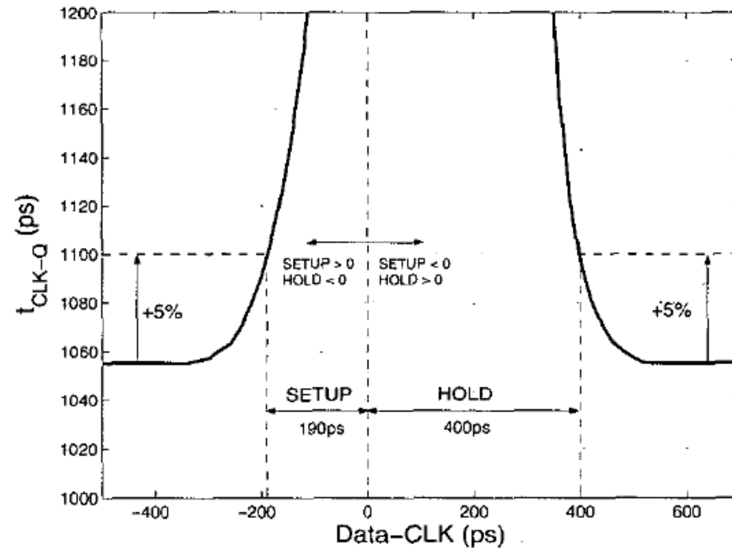

Figure 13. Definition of Setup and Hold Times[24]

Figure 13 is an example from 0.25um process, it can be seen that the setup time t_{setup} is 190ps, allowing 5% propagation delay increase (1100ps) comparing with the nominal value (1050ps). The same case for hold time (t_{hold} = 400ps for 5% delay degradation). If a smaller setup time is allowed, e.g. 120ps, which still guarantee the correct functionality of the DFF, this will lead invalid timing analysis because of the dramatically increasing propagation delay $t_{CLK\text{-}Q}$, which will probably lead a negative setup slack $Slack_{setup}$ unless a large clock period T is used. In that case, this choice of small setup time results in a longer critical path and a slower clock frequency.

Chapter 3 A Semiconductor Firm's SRAM Design

3.1 Introduction to A Vendor's Memory Compiler

This semiconductor firm's SRAM design is generated by a vendor's memory compiler with 0.15um technology node. This compiler provides flexibility that the user can choose different numbers of words as well as how many bits one word has. Except for some common choices like 16, 32 or 64-bit for a word, arbitrary bits design is also supported.

Besides, the user can determine the height/width ratio of the physical layout so that the generated layout can have different shapes/outlines to fit different requirements. It can become extremely high with few bitlines and many word lines. Or conversely, an extremely wide layout is possible with many bitlines and few word lines.

There are many PVT (Process Voltage Temperature) conditions associated with this design. For the process, one of the FF (NMOS fast, PMOS fast), TT (NMOS typical, PMOS typical) or SS (NMOS slow, PMOS slow) can be chosen depending on the technology process. The

voltage range is from 1.35V to 1.95V depending on the peripherals, like power supply design. As for the temperature, this SRAM is required to function correctly from -40 ℃ to 150 ℃.

Since a large numbers of volume and arbitrary bitwidth are supported by this compiler, there can be huge amount of the final generated layouts. Besides, even for a fixed choice, the height/width ratio can be also adjusted. When considering the PVT variations, the actual choices could be hundreds of thousands of combinations.

The user needs to know all the characteristics of the design before actual processing, like timing constraints, power constraints, etc. A classic way to get such information is from simulation. A full circuit simulation can provide some of these characteristics, while the cost is high, since a single runtime might take minutes or hours. Multiple simulations may be required to extract all information needed. In addition, there are literally hundreds of thousands of combinations of bitwidth, height/width ratios and PVTs, so it is impossible to simulate every single one of them to get information associated with this very combination, which potential customer might be interested in. Besides, the time from designing a new product to the market is getting shorter, which makes this full circuit simulation impractical.

The compiler has a different method to come up with all the required parameters associated with different design combinations. This method is equation-based and will dramatically reduce the simulation time. Once the compiler has the values of all variables for different blocks of the entire circuit, it can come up with the overall characteristics by adding them together according to pre-defined equations. The compiler takes basic simulation results of each block as inputs, then it can handle all the variations (e.g. different PVTs, signal slew rate, output load capacitance) the user might want to use. Such method can give the user a confident margin and estimation of the performance of actual chip, and once it complies all the requirements, the final product will be in that range.

But there is a disadvantage to use this equation-based method, which is too conservative (and too pessimistic) for most PVT conditions. On the other hand, the .lib for some PVT conditions (e.g. data writing delay (tWR) under FF/1.60V/150 ℃ and FF/1.95V/-40 ℃) is optimistic comparing with the results we gather from the extracted layout SPICE simulation. There is always a trade-off between reliability and performance. If the user want to have very small data input setup time (tSDI) under FF/1.95V/-40 ℃, e.g. 0.300ns, there might be no the .lib value which is smaller (0.7ns in the .lib across all PVT conditions). In such case, this method will mislead the user that such requirement is impractical. But in fact, our extracted layout SPICE simulation method shows the tSDI under FF/1.95V/-40 ℃ is 0.050ns, which satisfies the user's requirement. Besides, the compiler doesn't provide information about how much the margin

will be before the circuit starts to fail. For example, for the setup time, the margin could be relatively small for the slow circuit (SS/1.35V/-40 ℃), but it could be fairly large for the fast circuit (FF/1.95V/-40 ℃). In addition, the user might want to know the exact margin built-in. Sometimes it is not necessary to have so much margin built-in because higher performance could be achieved with a little margin sacrifice.

There is another problem embedded in this equation-based method that not every parameter value in the .lib is pessimistic, there are some which is optimistic instead. For example, tWR we simulate for FF/1.60V/150 ℃ is 1.633ns, but in the .lib, it is 0.500ns (shown in Table 5). To tell from our results, it is 3X larger in reality than the .lib. Except for tWR, we find the data reading delay (tRD) has the same issue under FF/1.60V/150 ℃ and FF/1.95V/-40 ℃. There might be more values which are optimistic somehow. In this case, it can't be guaranteed that when the .lib satisfies all the user's requirements, the final product will do the same.

Table 5. Some Optimistic Values in The .lib

PVT	Layout	Param	Polarity	.lib (ns)	Simulation (ns)
FF/1.60V/150 ℃	Default	tWR	Rising	0.500	1.633
FF/1.95V/-40 ℃	Default	tWR	Rising	0.500	1.075
FF/1.60V/150 ℃	Default	tRD	Falling	0.500	2.054
FF/1.95V/-40 ℃	Default	tRD	Falling	0.500	1.318

So our goal is to reproduce the spec values for all the parameters in the .lib. Since the spec values are the major part of these values, adding some variation from other terms depending on the equations, once we determine the spec values, we can generate more realistic matrices for all of them, which guarantee the circuit will not fail as long as it satisfies all the user's requirement. Besides, the information of the actual built-in margin will be also available.

3.2 Design Automation Using Script Languages

Since the methodology is associated with a lot of fully extracted layout simulations for different PVTs using SPICE simulator Eldo, many iterations of the simulation take much time to reach a conclusion. In order to automate the entire simulation flow (let the computer to automatically initialize the simulations and collect the data after completion) and minimize the human intervention during simulation, a script is written by the user in both Perl and Ruby to expedite each iteration, the source code is included in the appendices. Thanks to the script, the user can focus on interpreting the extracted data by computer instead of tweaking the simulation input

files. Such large amount of simulations couldn't be possible without the script taking care of many steps in the background.

The basic idea of the script is to read the configuration files written by the user, understanding the parameters for each iteration. Then the script will do pattern matching to modify the template input file of the simulator Eldo. After that the script will invoke the Eldo to run the simulation and wait for the completion, then start another run with the new parameters set. Once all the iterations are finished, the script will do the pattern matching of the output files of Eldo, extracting the results the user is interested, generating a CSV (Comma-Separated Values) file for human to post-process.

Chapter 4 Data Input Setup Time (tSDI)

4.1 Equation

In the equation-based method, the tSDI is composed of three individual terms, *T_DI_del_ts_r/f_a*, *tSDI_spec* and *T_CLKIO_del_ts_a*. The *T_DI_del_ts_r/f_a* is the delay from top-level data input bus (DI) least significant bit (MSB) DI<0> to an internal node "N2" (the middle point between the master and the slave latches) of the underlying DFF of LSB in the datapath, which is shown in Figure 14:

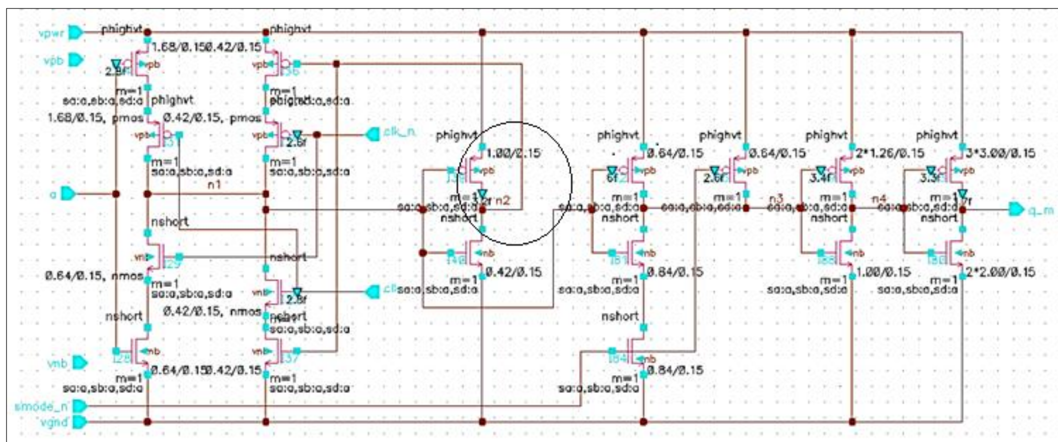

Figure 14. Schematic of the Underlying DFF

The *T_CLKIO_del_ts_a* is the delay from the top-level clock pin (CLKin) to the local clock pin (CLK_LOC_N) of the underlying DFF of LSB.

tSDI_spec is the actual central point of the matrix in the .lib. The compiler takes the *tSDI_spec* as an input which the user specifies before it constructs the matrix. It uses the *tSDI_spec* as the starting point and both *T_DI_del_ts_r/f_a* and *T_CLKIO_del_ts_a* act as variations depending

on different output load capacitance and input signal slew rate. We think such *tSDI_spec* value (same as other spec values) are achieved from ASIM run before. The .lib uses 0.7ns across all PVT conditions.

$$\mathit{tSDI_rr_ar} = \mathit{T_DI_del_ts_r_a} + \mathit{tSDI_spec} - \mathit{T_CLKIO_del_ts_a}$$

$$\mathit{tSDI_rf_ar} = \mathit{T_DI_del_ts_f_a} + \mathit{tSDI_spec} - \mathit{T_CLKIO_del_ts_a}$$

Equation 3

In Equation 3:

- *T_DI_del_ts_r_a* is the delay from DI<0> to an internal node "N2" of the underlying DFF when DI<0> is from logic 0 to 1.
- *T_DI_del_ts_f_a* is the delay from DI<0> to an internal node "N2" of the underlying DFF when DI<0> is from logic 1 to 0.
- *tSDI_spec* is the input value the user specifies when running compiler, which serves as the central point of the generated matrix.
- *T_CLKIO_del_ts_a* is the delay from CLKin to CLK_LOC_N of the underlying DFF of LSB.

4.2 Schematic

The Figure 15 shows the schematic of tSDI, from which it can be seen that there are two input signals, DI<0> and CLKin. The actual clock pin of the underlying DFF, CLK_LOC_N, is connected to CLKin through some delay. The compiler takes the two delays shown in Figure 15 as parameters to vary from the tSDI_spec to generate the 5x5 matrix.

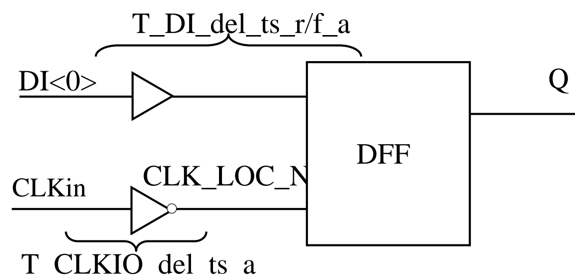

Figure 15. Schematic of tSDI

In order to reproduce tSDI_spec value equal of 0.700ns in the .lib, the worst case PVT condition (SS/1.35V/-40 ℃) is chosen. The clock period (tCYC) has to be increased from 8ns to 12ns so

that circuit can work correctly. Since circuit is slower than typical PVT condition (TT/1.80V/25 ℃), the default tCYC=8ns is not suitable anymore.

The critical point where circuit starts to fail is 0.420ns, and the reason is the underlying DFF can't catch the valid DI signal anymore. The underlying DFF shows metastability called delay degradation (DD). The delay degradation is the smaller time between input and the clock (T_{setup}) is, the larger the propagation time between clock and output ($T_{propagation}$) is than nominal value (computed when there is enough time between input and the clock). When T_{setup}=0.500ns, the delay degradation is almost 9.82% already, shown in Figure 22 (a).

The data output bus (DO) does not show any delay degradation, in other words, the delay degradation from the underlying DFF does not pass through to the final output DO. There is an internal node named DO_I_N (the white circle shown in Figure 16 and Figure 17), which is located before the output buffer. DO_I_N is connected to the negative output DINREG_N (the white circle shown in Figure 16) of the underlying DFF, but gated by WE (write enable) and BITEN (bit enable) (the white circles shown in Figure 16 and Figure 17). Since WE arrives very late comparing with DINREG_N (about 3ns after DINREG_N arrives), so that even though DINREG_N shows delay degradation due to the previous DFF and shifts about 0.700ns, as long as DINREG_N is valid before WE arrives, DO_I_N will start to toggle right after WE enables the transistors and DINREG_N will pass through those two transistors to DO_I_N. In this case, our delay degradation measurement can't be conducted between the top-level ports CLKin and DO<0> because the logic mentioned before filters such delay shifting due to the DFF. The schematic from DI to DO is shown in Figure 16:

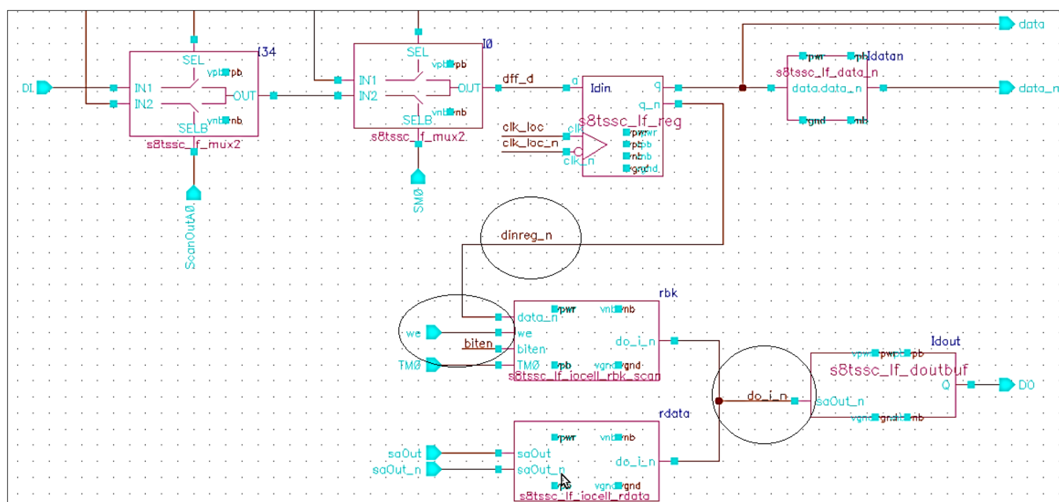

Figure 16. Schematic from DI to DO

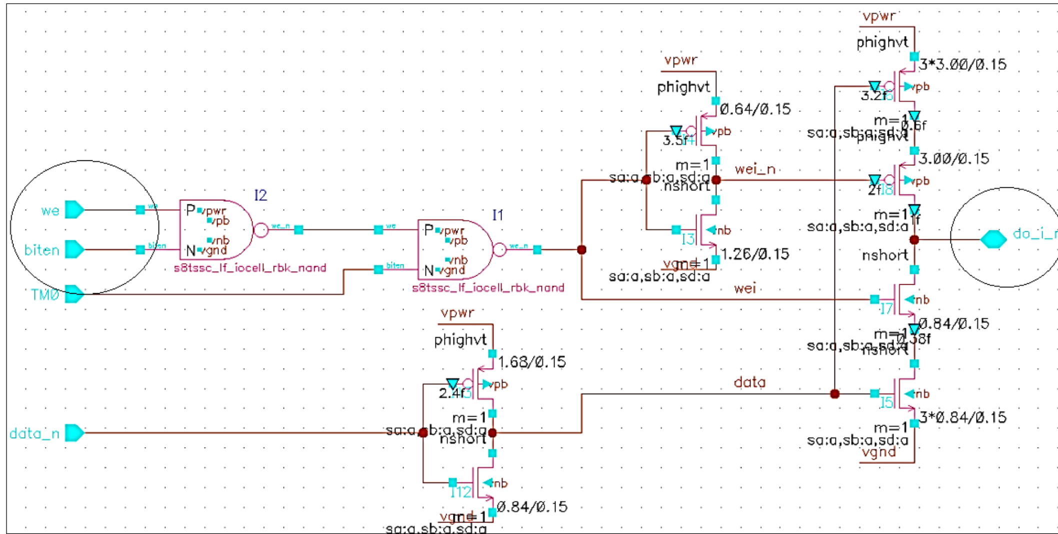

Figure 17. Schematic of RBK Block

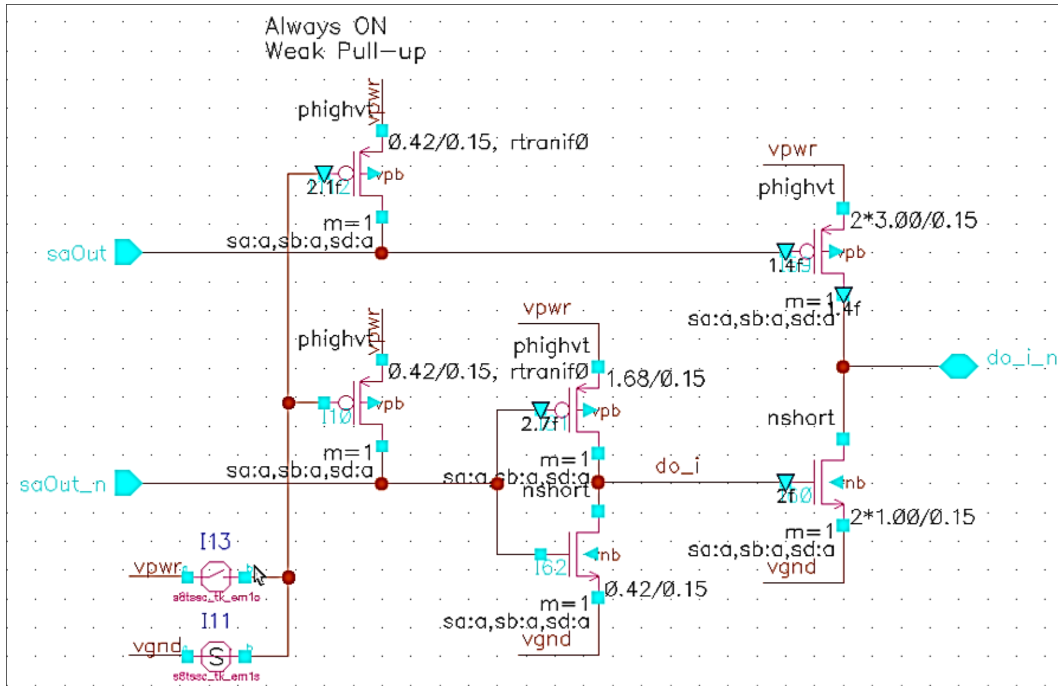

Figure 18. Schematic of RDATA Block

As shown below in Figure 19, the black and green curves are DINREG_N signals from different T_{setup} (2ns vs. 0.440ns), and there is observable 354ps delay indicating there is delay degradation from the underlying DFF. While WE and DO_I_N overlaps, which indicates the toggle of DO_I_N is triggered by the toggle of WE and the delay degradation shown in DINREG_N does not pass through to DO_I_N. That is the reason such delay degradation could not be observed from the final output DO<0>.

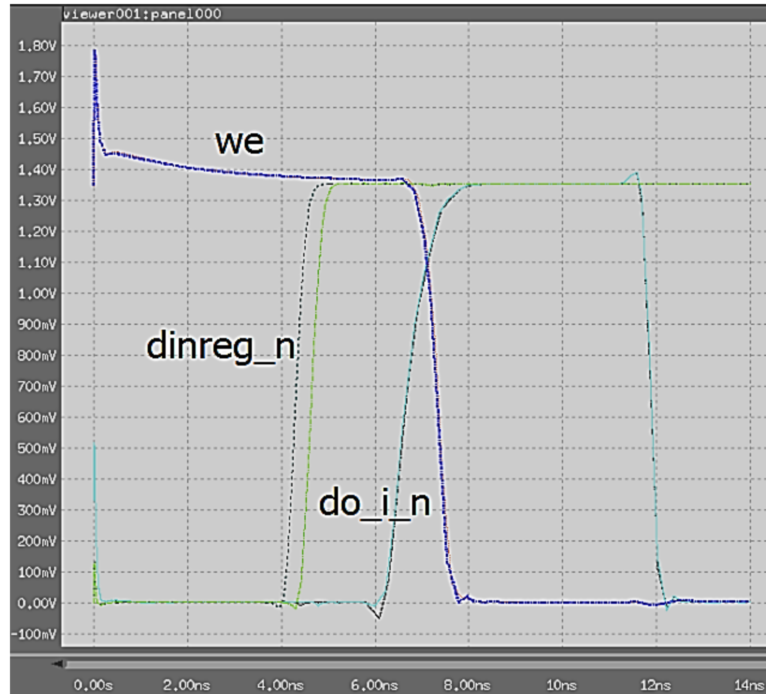

Figure 19. Waveforms Indicates Isolation of Delay Degradation

4.3 Logic of WE Signal

For WE signal, it is the logic output of three input signals, CLKin, R_WB and WLOFF (always logic 0 in normal operation), the Figure 20 shows the logic diagram, and the blue rectangles represents combination logic delay:

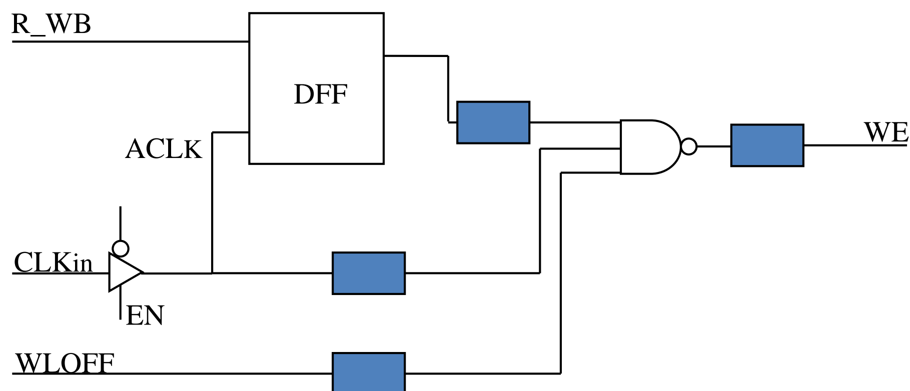

Figure 20. Schematic of WE Signal

4.4 Stimulus Waveforms

There are different top-level signals need be stimulated in order to get tSDI_spec: data input bus (DI), address input bus (AD), chip enable (EN), bit enable (BEN), read write bar (R_WB) and clock (CLKin). Except for simulation of chip enable setup time (tSEN), EN will be the first

to be active (logic 1). Since the circuit needs time to initialize after EN goes high, there will be a read cycle without doing anything dedicated to that. There is a feature called "write-through" that in write cycle, the data written into the SRAM will appear on data output bus (DO) after some delay, which is required by modern cache design, when the microprocessor wants to write data to the cache, it can write the same data to the memory behind the cache simultaneously. Thus it is hard to distinguish whether writing is successful with only one write cycle simulation. Besides, if a writing logic 1 is to be tested (tSDI rising polarity), a logic 0 should be guaranteed to be written into the SRAM bitcell before the writing logic 1 happens. Same case for writing logic 0 (tSDI falling polarity). So two write cycles will be used, which will be the second and third clock cycles, write logic 0 then logic 1 for tSDI rising polarity or write logic 1 then logic 0 (shown in Figure 21). In this case, if the internal SRAM bit flips (shown in Figure 21), it is assured that the write logic 1/0 is successful. Then the T_{setup} can be reduced till the internal SRAM bit does not flip any more. In general, whether the internal SRAM bit flips will be the indication of whether the circuit works correctly or not. Because the worst case for setup time is the slowest circuit, and CLK_LOC_N is slower than CLK_LOC, the CLK_LOC_N is chosen in the setup time analysis.

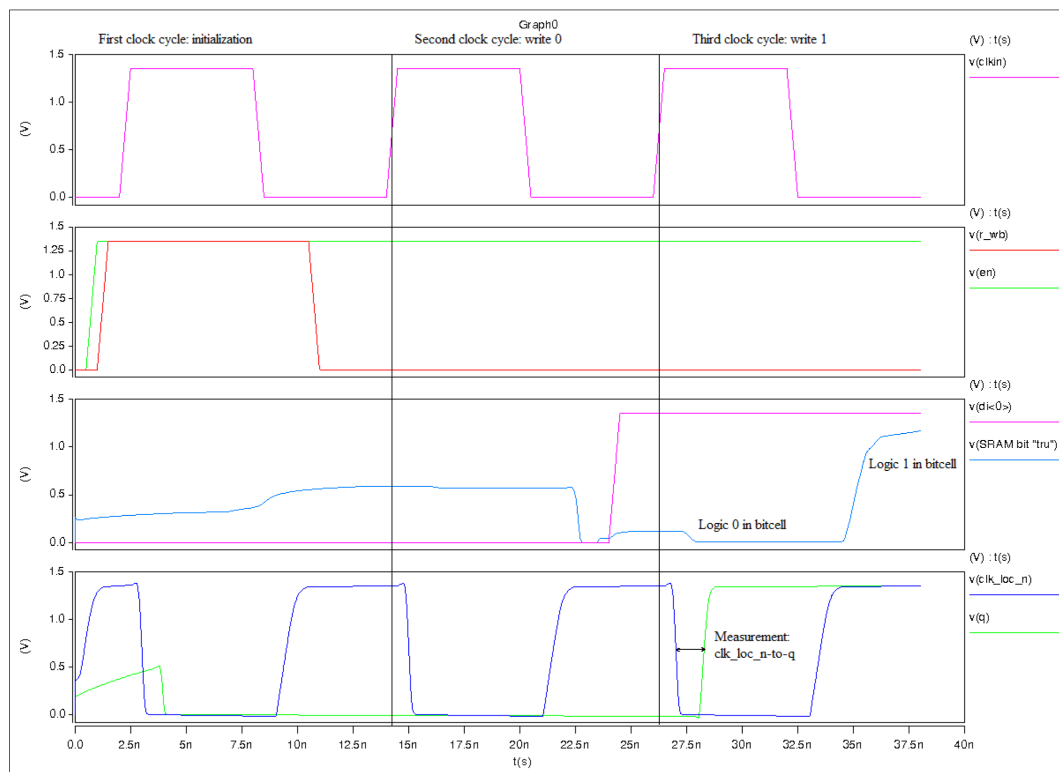

Figure 21. Stimulus Waveforms of tSDI Simulation (SS/1.35V/-40 ℃, Rising)

4.5 Methodology

The compiler uses a pre-defined tSDI_spec across all PVT conditions to be the central point of all matrices. Since the worst PVT condition for a setup time is SS/1.35V/-40 ℃, the 0.700ns of tSDI_spec should represent the margin which the compiler uses in this worst case. If such margin is kept unchanged for all PVT conditions, all tSDI_spec values associated with those different PVTs can be generated instead of using the only, most conservative one for all cases. In this way, the compiler could generate a more realistic, more balanced (reliability vs. performance) tSDI matrix for each PVT condition.

Based on the simulation of SS/1.35V/-40 ℃, the nominal delay from the underlying DFF CLK_LOC_N to DATA is 1.259ns (T_{setup}=2ns). When T_{setup}=0.700ns, which matches the tSDI_spec, the simulated delay is 1.286ns. The margin is 1.78%. Then this margin could be used in other PVT conditions to determine the tSDI_spec associated. All tSDI_spec values associated with the rest of PVT conditions can be achieved when 1.78% delay degradation happens.

4.6 Optimization (PassFail vs. Dichotomy)

The Eldo simulator provides an optimization method to automatically extract object by varying parameters in given range. The basic algorithm is bisectional scan with tolerance specified by the user. Since Eldo can't work on any range, in other words, if there is a point where Eldo can't extract the measurement, it will give error message and exit. So there is a dedicated PassFail (P/F) method running before the actual bisectional scan to provide the simulator a valid parameter range.

The P/F method doesn't care about the starting point. It will try to get as close as possible to the critical point where circuit starts to fail (the simulator can't extract the measurement any more).

Dichotomy method is purely bisectional scan. There are three options the user can specify, minimal and maximal value (provided by P/F) and starting point. The simulator assumes the measurement curve will be monotonic, the Dichotomy will start with the starting point and one end. The user can specify with how much tolerance the simulator will consider to stop comparing with last step by adjusting tol_relpar value in Eldo option. Smaller tol_relpar indicates higher accuracy and longer simulation time.

4.7 General Procedures (Vary PVTs)

First, the P/F method need be run to get the valid range of T_{setup} to simulate the delay from CLK_LOC_N to DATA of underlying DFF. The upper bound could choose 2ns to get the nominal delay. The lower bound could choose 0 in order to avoid missing the actual critical point where circuit starts to fail. For the first time of P/F optimization, the accuracy of Eldo simulation could be relaxed (by increasing tol_relpar value to 0.1, the default value is 0.001) so that the optimization will not take too long. Once it finishes, it will give the delay at the critical point, if it is larger than the margin the user want to use, this P/F optimization is enough because the desired point will be between the upper bound and critical point. If it is not, a more accurate, less relaxed P/F optimization might need to be run because the current critical point is too conservative. Several P/F optimizations might need to be iterated to get the reasonable critical point.

Once the P/F method gives the valid range of T_{setup}, the Dichotomy method could be utilized to find where delay is 1.78% larger than the nominal value (could choose different margin depending on the design). The Dichotomy method will do bisectional scan to get as close as possible to the T_{setup} point where delay is 1.78% larger. The Dichotomy method should use the same/higher accuracy as the last P/F optimization.

Once the T_{setup} where delay is 1.78% larger than nominal is given by optimizations, it will be the central point of tSDI matrix of this very PVT condition, tSDI_spec. When tol_relpar=0.1, it will give the user 1E-11 accuracy. When tol_relpar=0.01, it will give the user 1E-13 accuracy. The feasible low accuracy will be done by specifying tol_relpar=0.1, while feasible high accuracy will be tol_relpar=0.05.

4.8 Results

4.8.1 Rising Polarity

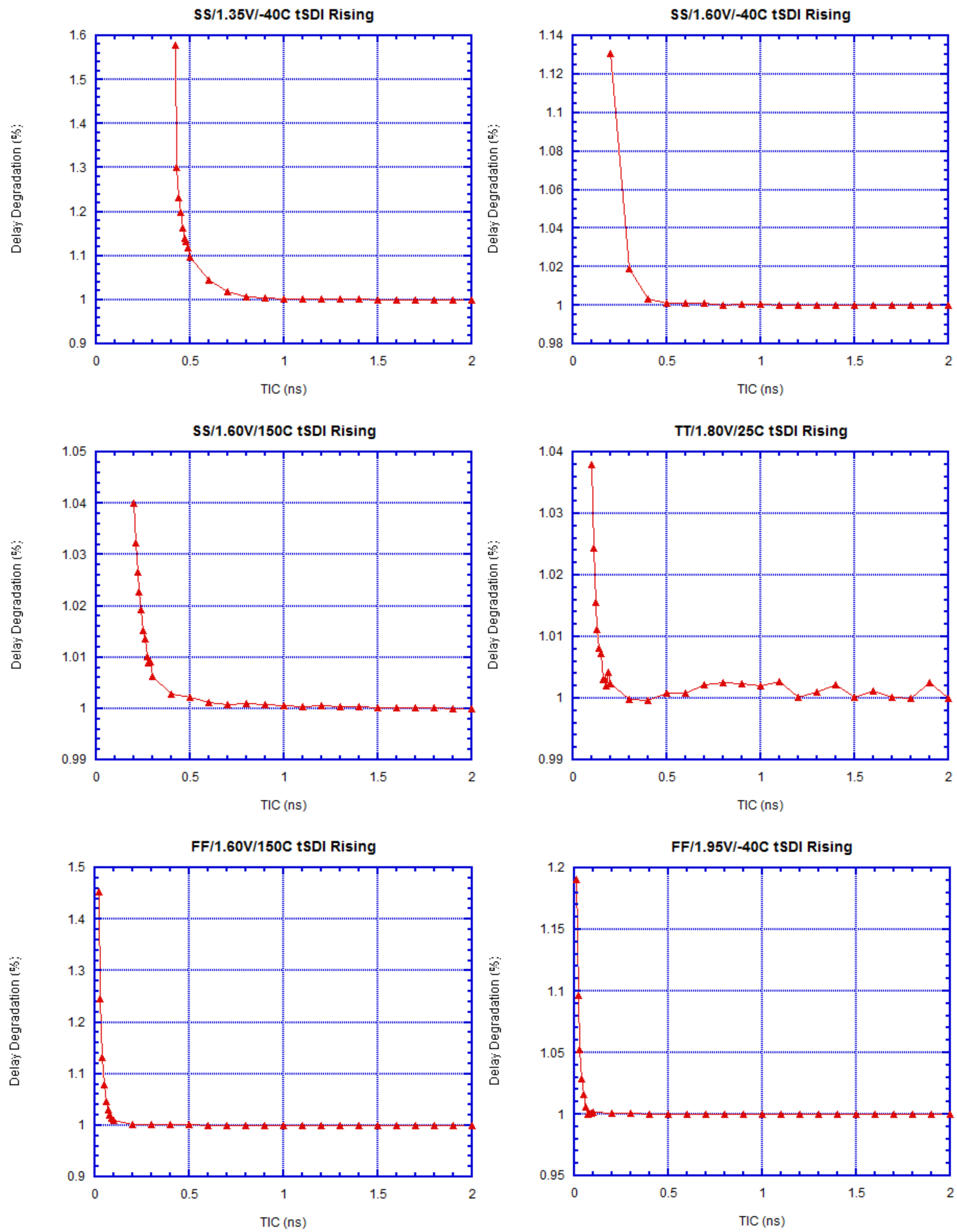

Figure 22. tSDI_spec Simulation Results (Rising) with Varying PVTs

It can be seen that there is some small fluctuation (<1%) from T_{setup}=2ns (where the nominal delay is calculated) for TT/1.80V/25 ℃ to where the delay degradation starts to appear.

According to the methodology, for TT/1.80V/25 ℃, when T_{setup}=0.120ns, the delay from the underlying DFF CLK_LOC_N to DATA is 1.78% larger than the nominal value. Comparing with the tSDI_spec = 0.700ns used in this PVT condition, the simulated central point of tSDI matrix is 4X smaller, which guarantees much smaller setup time (better performance) with reasonable 1.78% margin.

Another example for FF/1.95V/-40 ℃. Applying the 1.78% margin, the tSDI_spec for FF/1.95V/-40 ℃ is 0.050ns. Again, it is very smaller comparing with the default tSDI_spec the compiler uses, which gives the user better estimation of how fast the circuit could go before failure starts. One thing need be notified is that the delay degradation curve is very sharp once showing up. The 1.78% point is on the very edge of the cliff, which is not a suitable point for operation. If there is a little variation of the T_{setup}, the circuit will probably fail. There might be an independent margin of these timing parameters acting like design guardband, within which these parameters could have a small perturbation safely without going into the catastrophic failure. We has 0.200ns design guardband. Once it added to the simulated tSDI_spec, a better estimation of tSDI_spec could be 0.050+0.200=0.250ns.

4.8.2 Falling Polarity

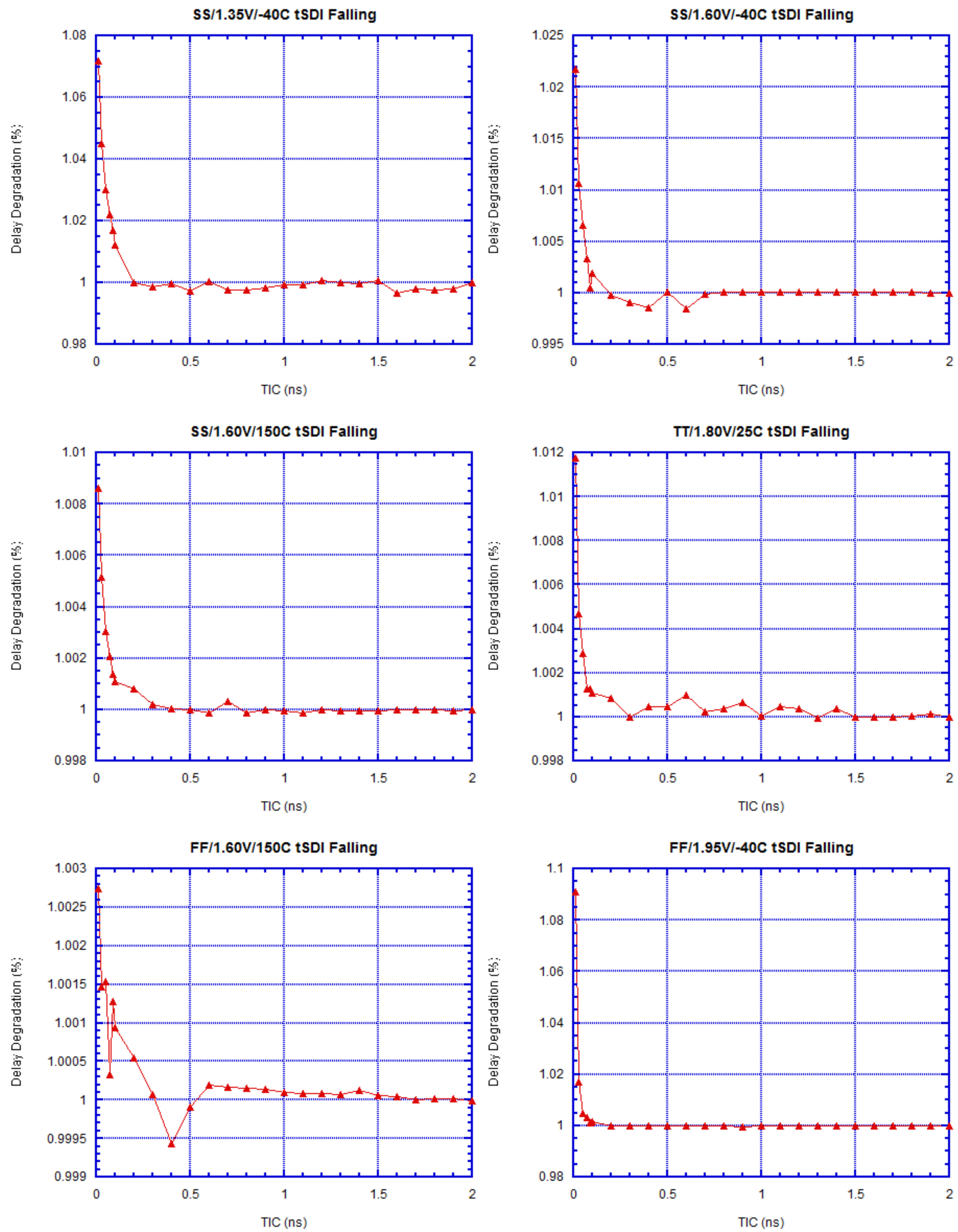

Figure 23. tSDI_spec Simulation Results (Falling) with Varying PVTs

For the falling polarity, it can be seen that, for SS/1.35V/-40°C, there isn't any delay degradation when T_{setup}=0.700ns. So instead applying the same delay degradation percentage through all PVTs, we pick SS/1.35V/-40 ℃ as a reference, then extract where the catastrophic failure happens (the Eldo can't extract the CLK_LOC_N to DATA delay). The difference between the T_{setup} where catastrophic failure happens and the 0.700ns is assumed to be the design guardband. For SS/1.35V/-40 ℃, the catastrophic failure point is T_{setup}=0, since the tSDI_spec in .lib is 0.700ns, the design guardband is 0.700-0=0.700ns, which is maintained through all other PVTs. The extracted layout simulation results for all PVTs have the same catastrophic failure point 0ns, which leads to the same simulated tSDI_spec=0.700ns for tSDI falling polarity.

Chapter 5 Data Input Hold Time (tHDI)

5.1 Equation

In the equation-based method, the tHDI is composed of three individual terms, T_DI_del_th_r/f_a, tHDI_sim and T_CLKIO_del_th_a. The T_DI_del_th_r/f_a is the delay from top-level data input bus (DI) most significant bit (MSB) DI<15> to an internal node "N2" (the middle point between the master and the slave latches) of the underlying DFF of MSB in the datapath, which is shown in Figure 14.

The *T_CLKIO_del_th_a* is the delay from the top-level clock pin (CLKin) to the local clock pin (CLK_LOC) of the underlying DFF of MSB.

tHDI_sim is a design guardbanded simulation value to be used for certain PVT. We has three different *tHDI_sim* for different process corners, which is shown in:

Table 6. tHDI_sim Guardband for Different Process Corners

PVT	tHDI_sim (ns)
SS/1.60V/150 ℃	0.980
SS/1.60V/-40 ℃	0.980
TT/1.80V/25 ℃	0.540
FF/1.60V/150 ℃	0.560
FF/1.95V/-40 ℃	0.560

The .lib uses 0 assumption for hold time across all PVTs, then add the associated guardband for different PVTs to generate the central point of the matrices. For example, for SS corner, regardless the voltage and temperature, all central points are 0.980ns. Same case for TT and FF.

$$tHDI_rr_ar = -T_DI_del_th_r_a + tHDI_sim + T_CLKIO_del_th_a$$

$$tHDI_rf_ar = -T_DI_del_th_f_a + tHDI_sim + T_CLKIO_del_th_a$$

Equation 4

In Equation 4:

- *T_DI_del_th_r_a* is the delay from DI<15> to an internal node "N2" of the underlying DFF when DI<15> is from logic 0 to 1.
- *T_DI_del_th_f_a* is the delay from DI<15> to an internal node "N2" of the underlying DFF when DI<15> is from logic 1 to 0.
- *tHDI_sim* is the guardband value the user specifies when running compiler.
- *T_CLKIO_del_th_a* is the delay from CLKin to CLK_LOC of the underlying DFF of MSB.

5.2 Schematic

The Figure 24 shows the schematic of tHDI, from which it can be seen that there are two input signals, DI<15> and CLKin. The actual clock pin of the underlying DFF, CLK_LOC, is connected to CLKin through some delay. The compiler takes the two delays shown in Figure 24 as parameters to vary from the 0 + tHDI_sim to generate the 5x5 matrix.

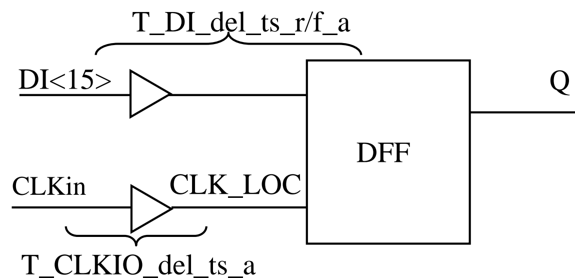

Figure 24. Schematic of tHDI

5.3 Stimulus Waveforms

There are different top-level signals need be stimulated in order to get tHDI_sim: DI, AD, EN, BEN, R_WB and CLKin. Except for simulation of tSEN, EN will be the first to be active (logic

1). Since the circuit needs time to initialize after EN goes high, there will be a read cycle without doing anything dedicated to that. Like the simulation of tSDI, there are two consecutive write cycles needed to make sure when we test if a logic 0/1 is written in the SRAM, a complementary logic 1/0 is already in the SRAM bitcell. So two write cycles will be used, which will be the second and third clock cycles, write logic 1 then logic 0 for tHDI rising polarity or write logic 0 then logic 1 (shown in). In this case, if the internal SRAM bit flips (shown in), it is assured that the write logic 0/1 is successful. Since the hold time of the underlying DFF needs to be extracted, the DI<15> will be toggled shortly after the CLKin, then the delay from CLKin to DI<15> is the T_{hold} for tHDI simulation. The T_{hold} can be reduced so that the hold time of the data after the trigger of clock is smaller and smaller till the internal SRAM bit does not flip any more, which indicates the hold time of the underlying DFF isn't satisfied anymore. In general, whether the internal SRAM bit flips will be the indication of whether the circuit works correctly or not. Because the worst case for hold time is the fastest circuit, and CLK_LOC_N is slower than CLK_LOC, the CLK_LOC is chosen in the hold time analysis.

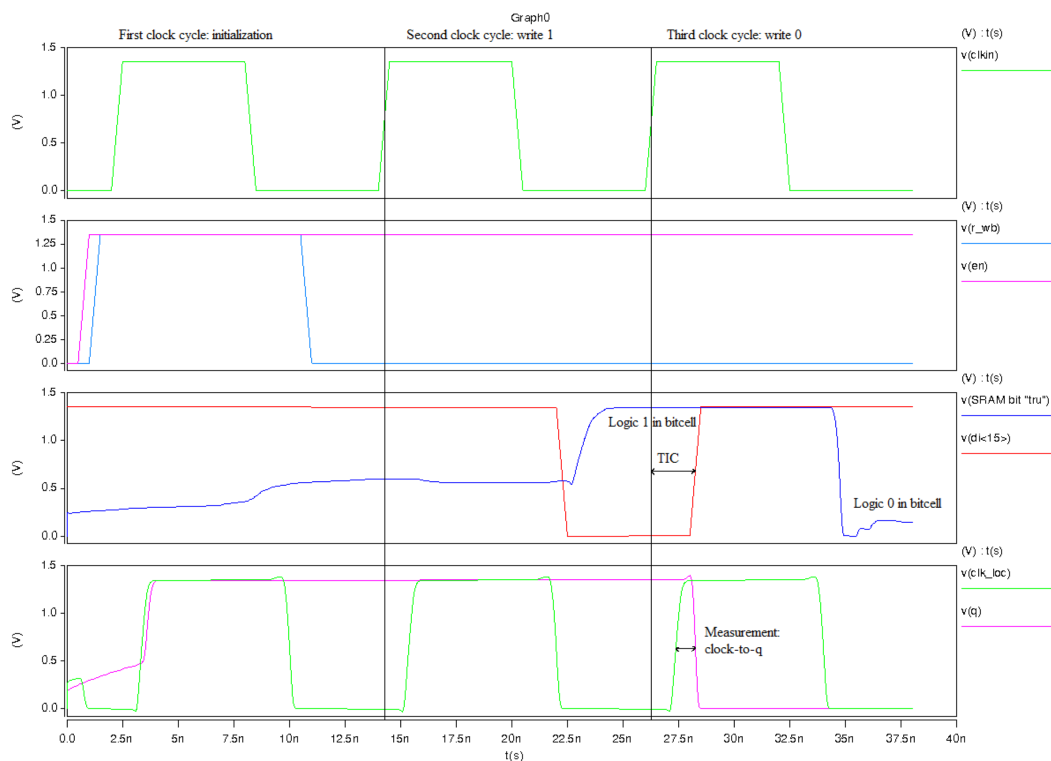

Figure 25. Stimulus Waveforms of tHDI Simulation (SS/1.35V/-40 ℃, Rising)

5.4 Methodology

The compiler uses a user-specified tHDI_sim for different process corners to be the central point of all matrices. Since the worst PVT condition for a hold time is FF/1.95V/-40 ℃, the

0.560ns of tHDI_sim should represent the guardband which the compiler uses in this worst case. If such guardband is kept unchanged for all PVT conditions, all tHDI_sim values associated with those different PVTs can be generated by adding this guardband to the actual simulated catastrophic failure points. In this way, the compiler could generate a more realistic, more balanced (reliability vs. performance) tHDI matrix for each PVT condition.

Based on the simulation of FF/1.95V/-40 ℃, the catastrophic failure point is 0.030ns. Since 0.560ns is used in the .lib, the actual guardband needed to be maintained is 0.560-0.030=0.530ns. Then this 0.530ns guardband should be kept unchanged across all other PVTs when adding to the catastrophic failure points associated with those PVTs.

5.5 Results

5.5.1 Rising Polarity

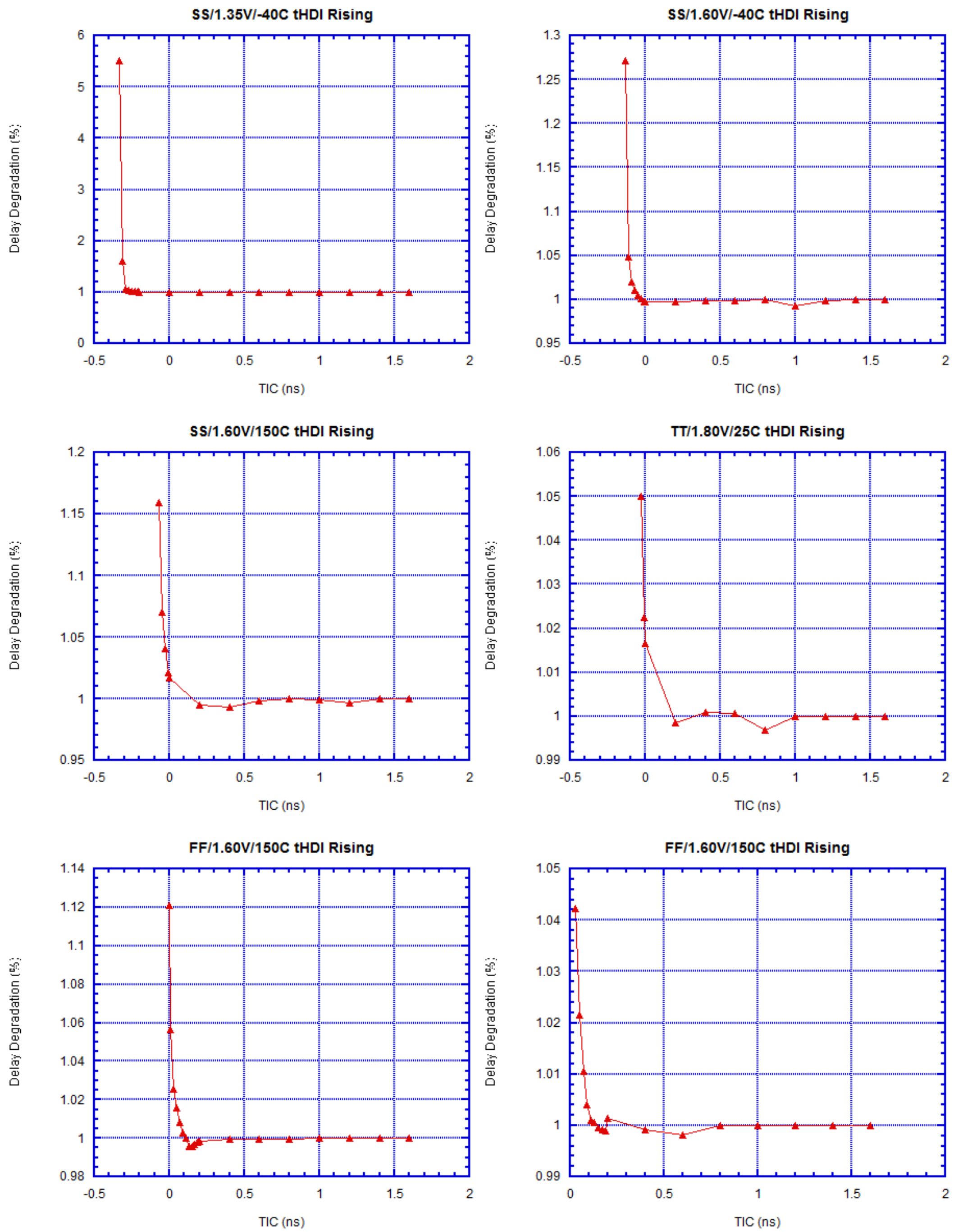

Figure 26. tHDI_sim Simulation Results (Rising) with Varying PVTs

Regardless of the 0.530ns guardband, the simulations across different PVTs show the actual catastrophic failure points are very close to 0, even negative values for hold time. Since the faster the circuit is, the worse the situation for hold time, it can be seen that for the slowest circuit, SS/1.35V/-40 ℃, its hold time catastrophic failure point is almost -0.300ns. With the circuit faster and faster, this catastrophic failure point actually shifts right, which is consistent with the assumption that the faster circuit is, the larger its hold time will be.

5.5.2 Falling Polarity

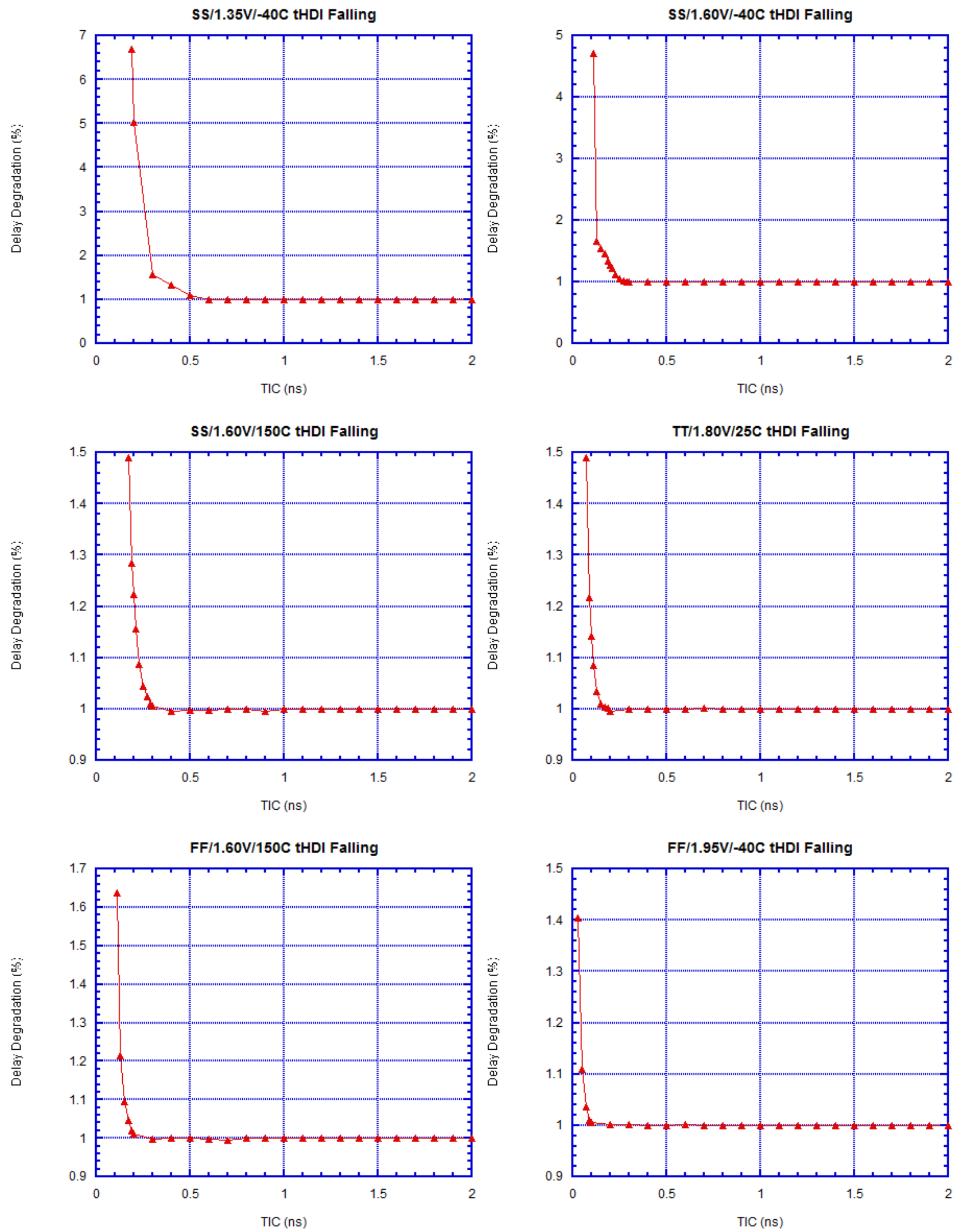

Figure 27. tHDI_sim Simulation Results (Falling) with Varying PVTs

Unlike the rising polarity, the simulations for different PVTs show that faster circuit has slightly smaller hold time because the catastrophic points are more on the left. Such phenomenon might result from the simulator accuracy, or there is some other mechanism to cause the slower circuit to fail earlier. But even though there is slight difference between faster and slower circuits, such difference isn't as large as what we see in rising polarity. Generally for falling polarity, the user could consider all PVTs have a uniform hold time, which is around 0.650ns after adding the guardband (0.530ns) extracted from FF/1.95V/-40 ℃.

Chapter 6 Data Writing Delay (tWR)

6.1 Equation

Similar with the tSDI, the tWR also has three terms, two from the subcircuit delay measurements and one spec value. The *T_CLKCTL_del_r_a* is the delay from top-level CLKin to local clock CLK_LOC which triggers the underlying DFF of LSB. The *T_DO_del_r/f_a* is the delay from DO_I_N to top-level DO<15>. Unlike the tSDI using same tSDI_spec (0.7ns) across all PVT conditions, the tWR_spec has three different values (minimal, typical and maximal). The tWR_spec has variations across process, in other words, the compiler uses the minimal value for FF, the typical value for TT and maximal value for SS.

$$tWR_rr_ar = T_CLKCTL_del_r_a + tWR_spec + T_DO_del_r_a$$

$$tWR_rf_ar = T_CLKCTL_del_r_a + tWR_spec + T_DO_del_f_a$$

Equation 5

In Equation 5:

- *T_CLKCTL_del_r_a* is the delay from top-level CLKin to local CLK_LOC which triggers the underlying DFF of LSB.
- *T_DO_del_r/f_a* is the delay from DO_I_N to top-level DO<15>.
- *tWR_spec* has three different values for minimal, typical and maximal conditions.

Table 7. tWR_spec Values for Different Processes

Min (FF)	Typ (TT)	Max (SS)
0.500ns	1.930ns	3.920ns

6.2 Schematic

The Figure 28 shows the brief schematic of tWR. It can be seen that the equation-based method is literally adding all the major delays of the path from CLKin to DO. The *T_CLKCTL_del_r_a* counts the delay of clock signal, and *T_DO_del_r/f_a* counts the delay of output buffer (the blue rectangle between DO_I_N and DO<15>). We assume the delays for the rest parts is included in the tWR_spec and will not change with different output load capacitances and signal slew rate.

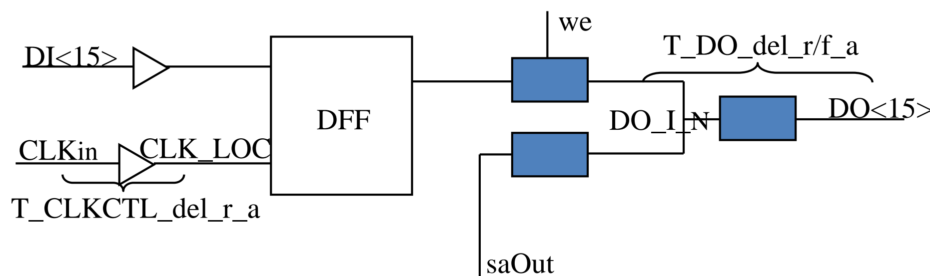

Figure 28. Schematic of tWR

6.3 Results

For direct measurement of tWR_spec, the delay from CLKin to DO<15> in write cycle is considered to be tWR_spec. Different simulated tWR_spec for different PVT conditions are shown in Table 11.

For the rising polarity, the maximal value of tWR_spec which the compiler uses is based on SS/1.60V/-40 ℃ or SS/1.60V/150 ℃ (depending on which is larger), but the slowest condition of all cases is SS/1.35V/-40 ℃. So it is reasonable the simulated tWR_spec of SS/1.35V/-40 ℃ is larger than the maximal value in the raw data file. On the other hand, the simulated tWR_spec for SS/1.60V/-40 ℃ is 3.211ns, for SS/1.60V/150 ℃ is 3.213ns, both are smaller than 3.920ns as expected. Same case for TT/1.80V/25 ℃. The minimal value of tWR_spec is based on FF/1.60V/150 ℃ or FF/1.95V/-40 ℃ (depending on which is smaller). But with these two PVT conditions, the simulated tWR_spec values (1.633ns for FF/1.60V/150 ℃ and 1.075ns for FF/1.95V/-40 ℃) are larger than 0.500ns shown in Table 11. Same case for the falling polarity.

6.4 Validation

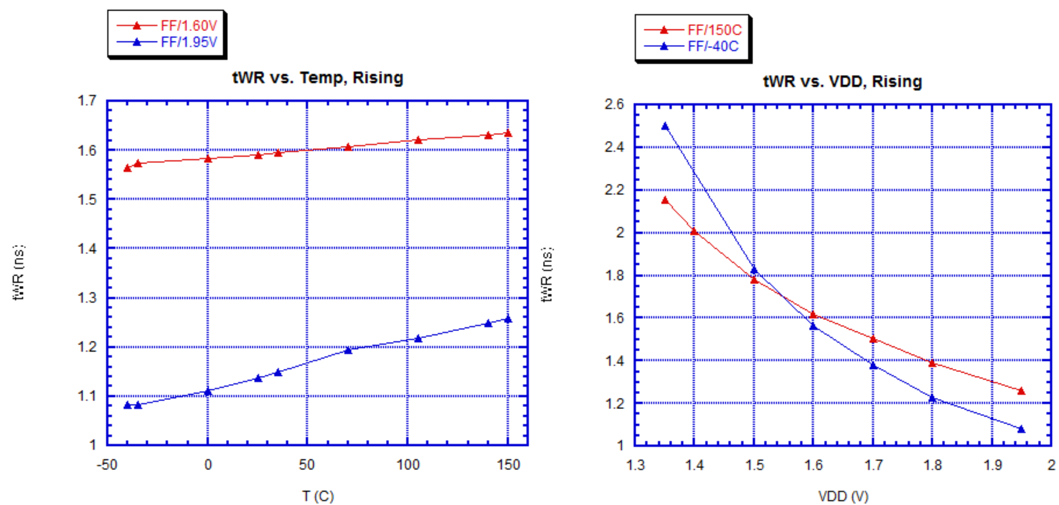

Figure 29. tWR_spec Simulation Results with Varying Temperature and V_{DD}

At first we assumes the .lib is very pessimistic, which means our extracted layout simulation results should be larger than the values in the .lib. But it turns out that some values are optimistic instead. The tWR_spec for FF/1.95V/-40 ℃ is about 2X larger than the .lib values (shown in Table 11). In order to have a sanity check to prove the methodology is correct, for both FF/1.60V/150 ℃ and FF/1.95/-40 ℃, we vary one of the temperature (T) and voltage (V_{DD}) keep the other one intact. The simulated curves are as expected, that higher temperature means more delay because the circuit is slower (Figure 29(a)). Higher V_{DD} indicates faster circuit (Figure 29(b)).

One interesting phenomenon is that when V_{DD} is relatively small (V_{DD} < 1.5V), increasing the temperature will actually increase the speed of the circuit, which is because the threshold voltage V_t of the MOSFETs is lower with temperature increasing. The lower threshold voltage V_t will compensate the negative effect resulting from lower mobility in higher temperature, and finally overcome it and make the circuit faster, which can be seen from Figure 29(b). When V_{DD} is smaller than 1.5V, the circuit at 150 ℃ has smaller data writing delay than the circuit at -40 ℃.

Chapter 7 Read/Write Setup Time (tSRWB)

7.1 Equation

The tSRWB also has three terms in the equation. Except for the tSRWB_spec, the rest two are delays measured from subcircuits. The *T_RWB_del_ts_r_a* is the delay from top-level R_WB

to an internal node “A_N” (the invert of input A) of the underlying DFF (the very left white circle shown in Figure 32). The *T_CLKCTL_del_ts_a* is the delay from top-level CLKin to local clock CLKEN. The compiler has a fixed tSRWB_spec (0.5ns) across all PVT conditions.

$$tSWRB = T_RWB_del_ts_r_a + tSRWB_spec - T_CLKCTL_del_ts_a$$

Equation 6

In Equation 6:

- *T_RWB_del_ts_r_a* is the delay from top-level R_WB to an internal node “A_N” of the underlying DFF.
- *T_CLKCTL_del_ts_a* is the delay from top-level CLKin to local clock CLKEN.
- *tSRWB_spec* has a value of 0.5ns across all PVT conditions.

7.2 Schematic

The Figure 30 shows the schematic of the tSRWB. It can be seen that the local clock ACLK which triggers the pre-charge latch of R_WB is gated by the EN_M, which is the registered signal of the EN. There are two different type of input registers: the normal DFF used in EN path and pre-charge latch in R_WB path. This pre-charge latch exhibits a unique delay degradation pattern different from the normal DFF, and that is reason we investigate it more and do individual simulation of this type of latch without other circuits.

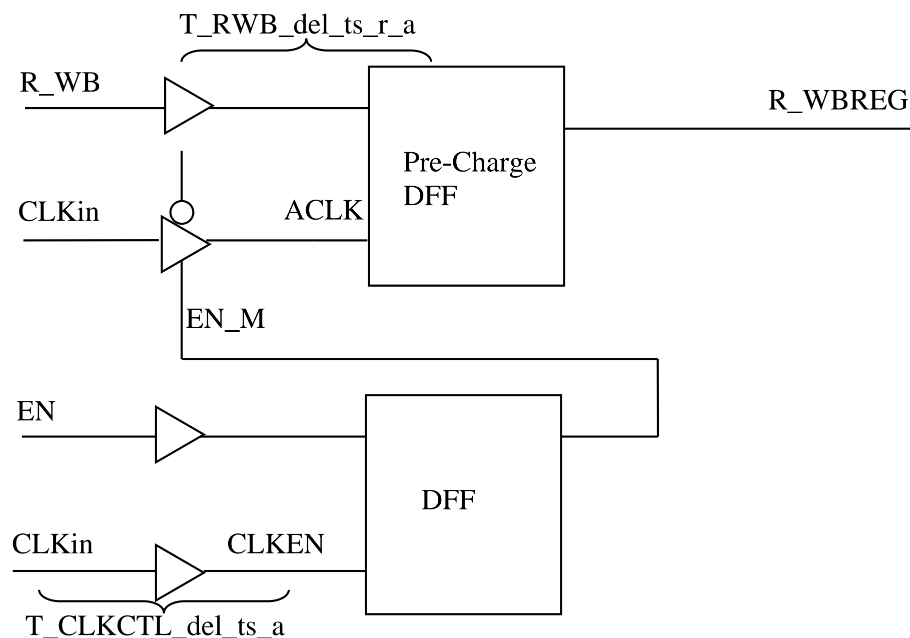

Figure 30. Schematic of tSRWB

7.3 Pre-charge Latch

According to design document, the input register used in R_WB signal path is an improved one. The normal input registers used for DI and EN are normal DFFs shown in Figure 31.

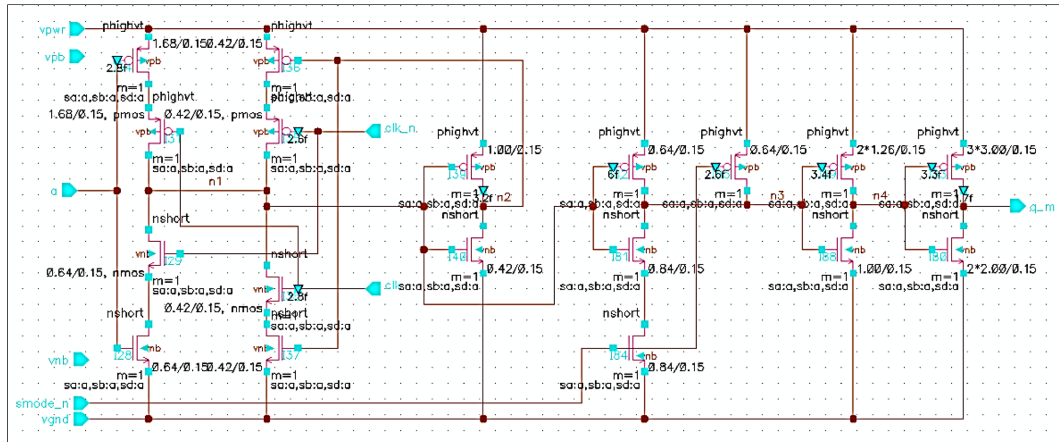

Figure 31. Schematic of Normal DFF

While for those input registers used for AD and R_WB, they are pre-charge latch shown in Figure 32.

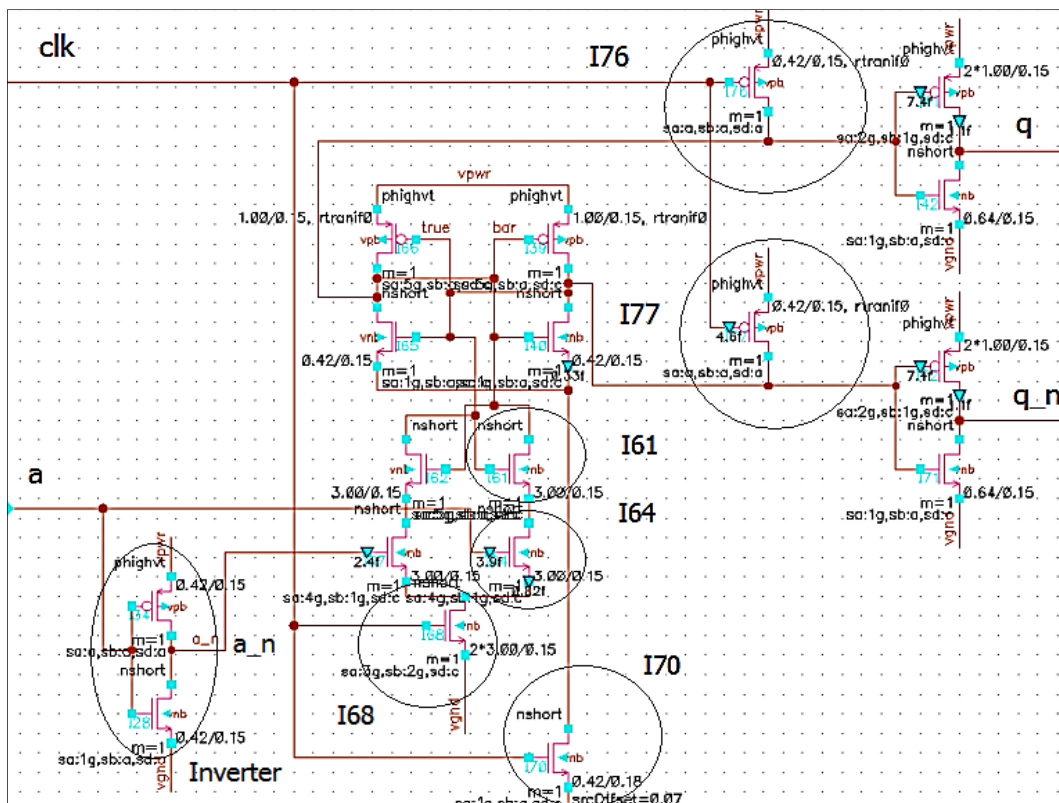

Figure 32. Schematic of Pre-charge Latch

When the CLK=0, it will open the PMOS I76 and I77 and shut down the NMOS I68, which will clamp the internal nodes “TRU” and “BAR” to be V_{DD} all the time. Once the CLK flips to 1, it will open the NMOS I68, and release the clamping. Now if the data input of the latch, A, is 0, the “TRU” node will be discharged to 0. On the other hand, if A=1, the “BAR” node will be discharge to 0. It works as a latch with level sensitivity of CLK.

We keep the same methodology as the tSDI_spec simulation does. But the tSRWB_spec simulation exhibits quite different delay degradation pattern (shown in Figure 34(c)(d)). It can be seen that the delay degradation curve is not monotonically increasing as expected when T_{setup} decreases. Especially for SS/1.35V/-40℃, there is a range where the delay increases to a maxima, then decreases to a certain level, then increases again. Seen from the curve, it is like a hill.

Another problem we find in this methodology when applying to the tSRWB_spec simulation is the results are extremely small than the .lib. In Table 11, it can be seen that for SS/1.60V/150℃, the tSRWB_spec we simulate (0.01ns) is 50X smaller than the value in the .lib (0.5ns). Even though we assume that the .lib is somewhat pessimistic, but such huge difference leads us to investigate more about this pre-charge latch used in R_WB signal path. We do individual simulation of such latch to show it exhibits quite different delay degradation pattern from the normal DFF used in DI and EN.

7.4 Delay Degradations of Normal DFF and Pre-charge Latch

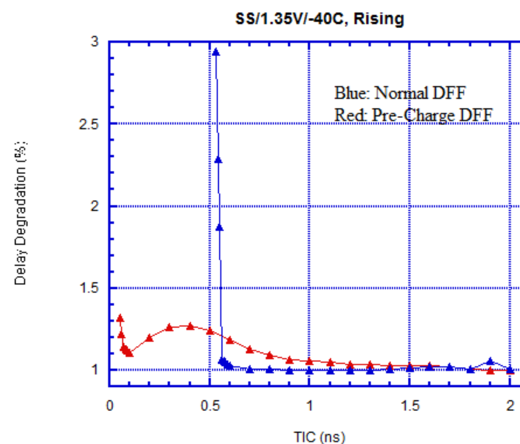

Figure 33. Comparison Between Normal DFF and Pre-charge Latch.

7.5 Individual Simulation of Pre-charge Latch without Other Circuits

7.5.1 Individual Simulation vs. Extracted Layout Simulation

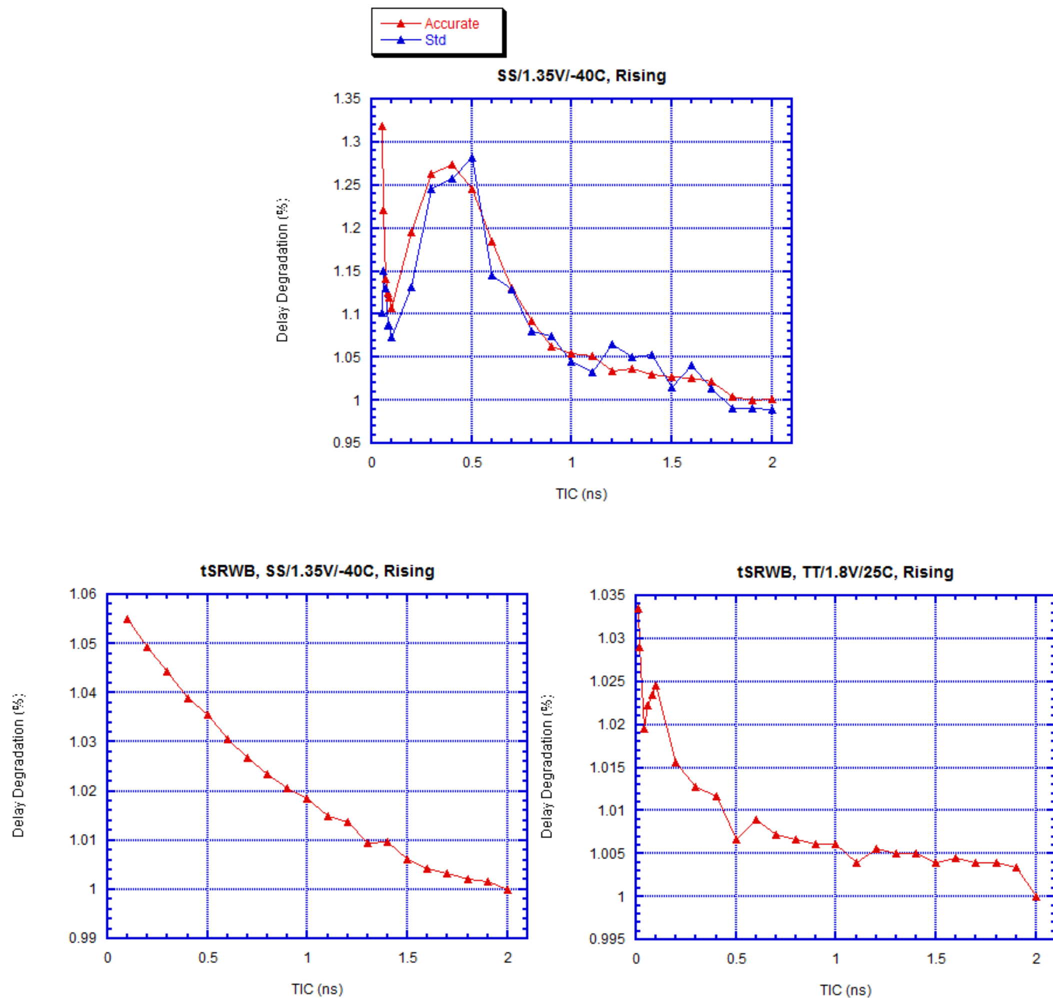

Figure 34. (a) Individual Simulation (b)(c) Extracted Layout Simulation

Comparing the red and blue curves in Figure 34(a), we can see the tuning factor play a significant role in simulation. With higher accuracy setting (.option tuning=accurate), the red curve is smoother with less unexpected spikes (e.g. the blue curve at T_{setup}=1.5ns). More obvious is in Figure 34(b). The blue curve has a large downward spike at T_{setup}=0.5ns. The low accuracy setting of the simulator could introduce some amount of noise into the results we have before. We think the actual value might not change too much, but the pattern is somehow changed by adding some unexpected spikes.

Comparing Figure 34(b) and (c), the pre-charge latch has different delay degradation patterns under SS/1.35V/-40 ℃ and TT/1.80V/25 ℃ respectively. There is a large hill at T_{setup}=0.5ns

with height of more than 25% delay degradation in SS/1.35V/-40 ℃. But in TT/1.80V/25 ℃, the general trend of curves is monotonically increasing with T_{setup} decreasing.

If comparing with Figure 34(b) and (c), we could see the pre-charge latch behaves worse individually than it with entire circuit. The Figure 34(c) and (d) are done with entire circuit. But if considering the same T_{setup}=0.5ns for Figure 34(b) and (c), for the entire circuit simulation, it only gives us 3.5% delay degradation, which should be used across all PVT simulations. While the individual simulation gives us more than 25% delay degradation at T_{setup}=0.5ns, which should be considered as catastrophic failure.

7.5.2 Varying Output Load Capacitance

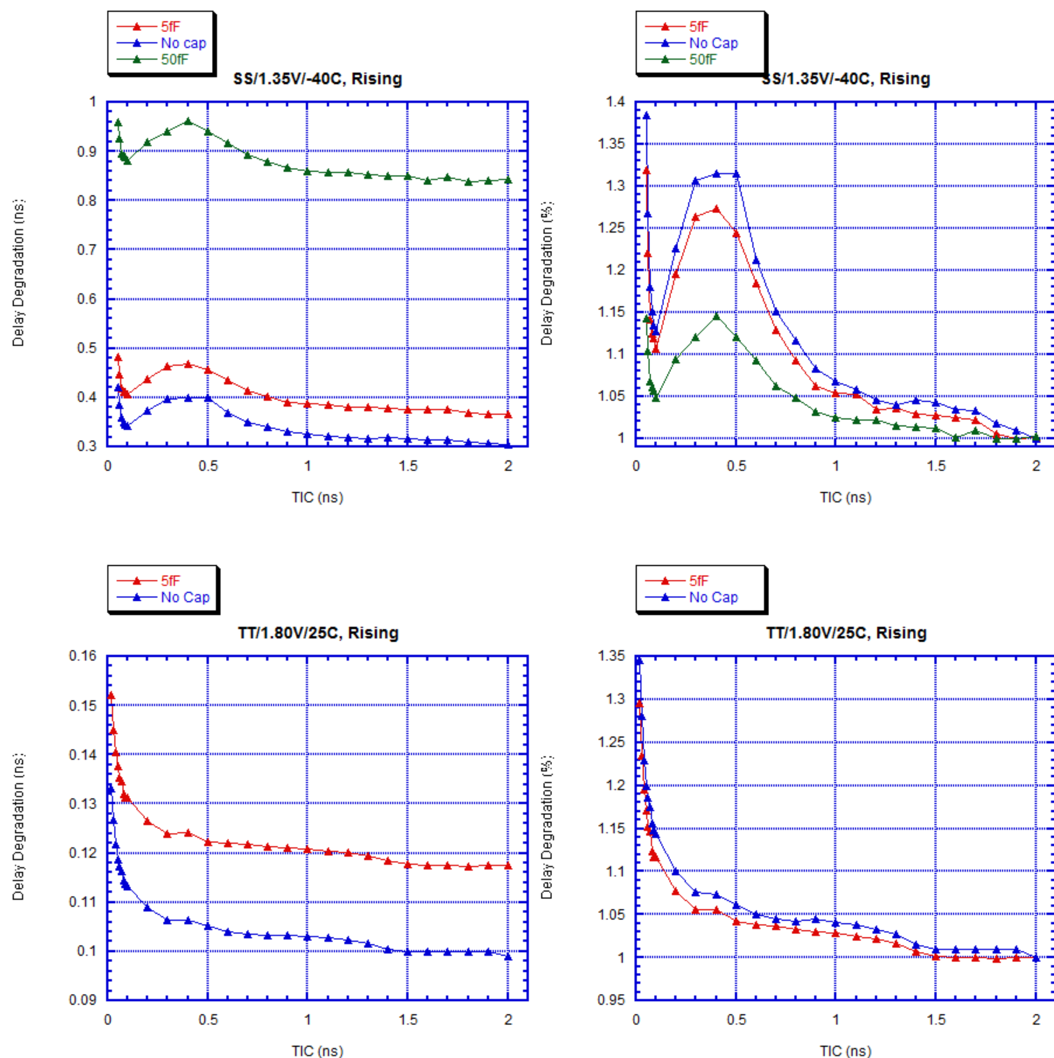

Figure 35. Pre-charge Latch Simulation Results with Varying Output Load Capacitance (a)(c) Absolute Value (b)(d) Percentage Value

From Figure 35, we can tell different output capacitance will result in different delay. But from Figure 35(a) and (c), the basic patterns are the same. Besides, from Figure 35(b) and (d), the percentage of delay degradation along with decreasing T_{setup} doesn't change too much. Even without output load capacitance, this pre-charge latch under SS/1.35V/-40 ℃ still shows more than 25% delay degradation at T_{setup}=0.5ns. And the blue curve is actually above the red curve (with 5fF output load capacitance), which means without output capacitance has worse delay degradation distortion.

7.5.3 Varying the W/L of PMOS I76 and I77

From Figure 32, we could see PMOS I76 and I77 provide the pre-charging path for the "TRU" and "BAR" nodes. When CLK=0, both PMOS are turned on and "TRU" and "BAR" are clamped to V_{DD}. The drive strength of these PMOS determine how fast the two nodes (with other nodes like the drain of I64, and capacitance associated with) are pre-charging. Larger W/L ratio can offer larger drive strength, larger charging current, which will reduce the time for these node to be pre-charged to a certain voltage. We want to know if the drive strength of these two PMOS, or the relative strength between these two and NMOS I70 can affect the shape or height of the abnormal hill in the delay degradation pattern found in simulation.

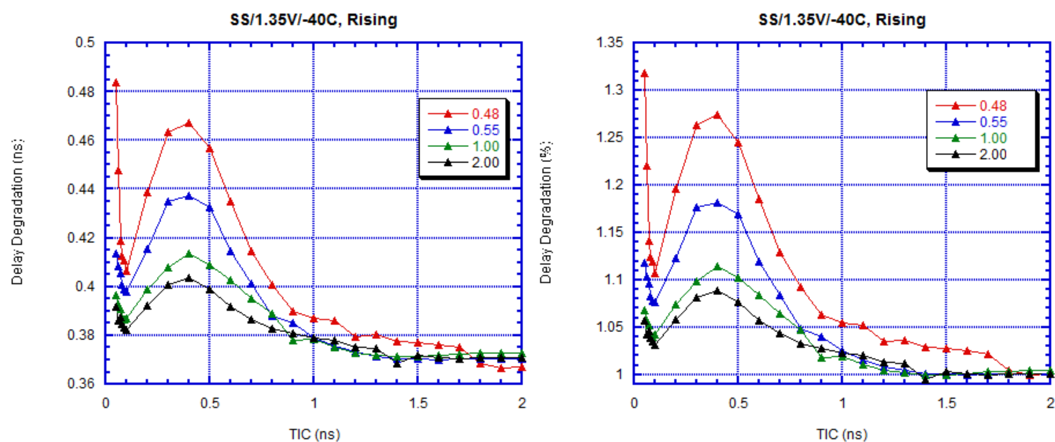

Figure 36. Pre-charge Latch Simulation Results with Varying I76/I77 Width (a) Absolute Value (b) Percentage Value

As we can see in Figure 36, increasing the width of both PMOS (I76 and I77) can help reducing the height of the hill between T_{setup}=0.1ns and 0.5ns. While the pattern shape keeps the similar.

7.5.4 Varying the Power Supply Voltage V_{DD}

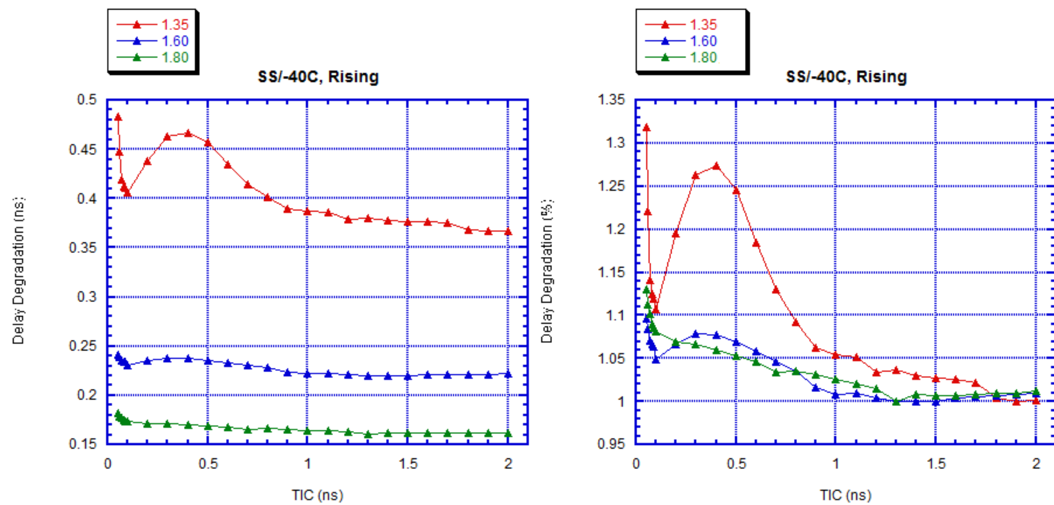

Figure 37. Pre-charge Latch Simulation Results with Varying Vdd (a) Absolute Value (b) Percentage Value

It can be seen that increasing the V_{DD} can greatly reduce the height of hill. Besides, for V_{DD}=1.8V, the abnormal hill disappears, and the entire delay degradation pattern comes back to the normal fashion.

7.5.5 Varying the Process

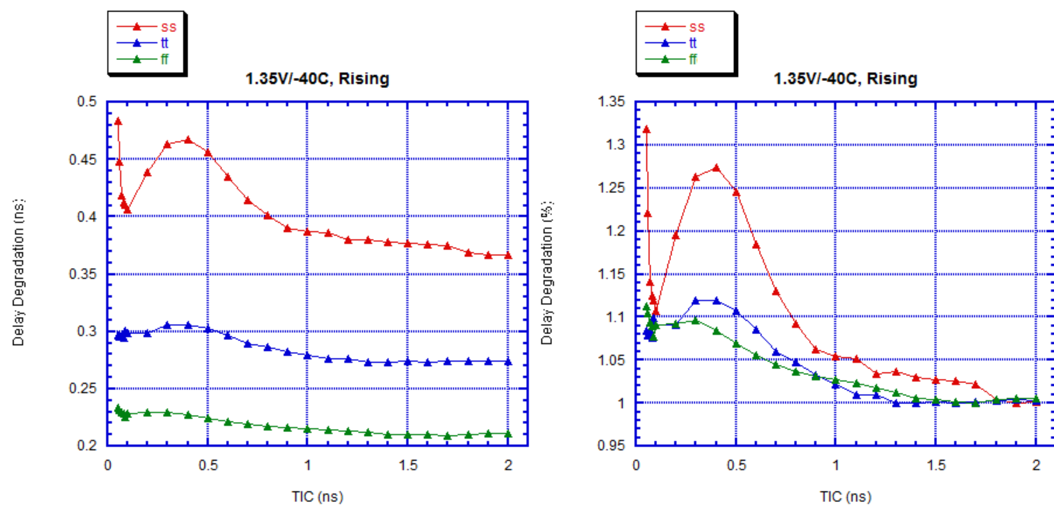

Figure 38. Pre-charge Latch Simulation Results with Varying Process (a) Absolute Value (b) Percentage Value

Similar with increasing the V_{DD}, using fast corners FF can reduce the height of hill. It can be still seen a little bit hill for TT corner, but there is none for the FF corner. The assumption, which still needs to be proven, is that the abnormal pattern (hill) could be dampened or eliminated with lower threshold voltage (V_{th}) of MOS, fast device or higher power supply (V_{DD}).

7.5.6 Varying the PMOS Model of Output Inverters

There are several PMOS models available in the tech library. The presumption is that with lower V_{th} PMOS of output inverters, it can dampen the hill in delay degradation pattern. The reason is, with lower V_{th}, the inverters will flip earlier than those with higher V_{th}. The different PMOS models with different V_{th} are shown in Table 8:

Table 8. Different PMOS Models in Tech Library

Model	Vth (mV)	W/L (um)
phighvt	942	2 x 1.65/0.15
plowvt	602	2 x 3.00/0.35
pshort	790	2 x 1.65/0.15

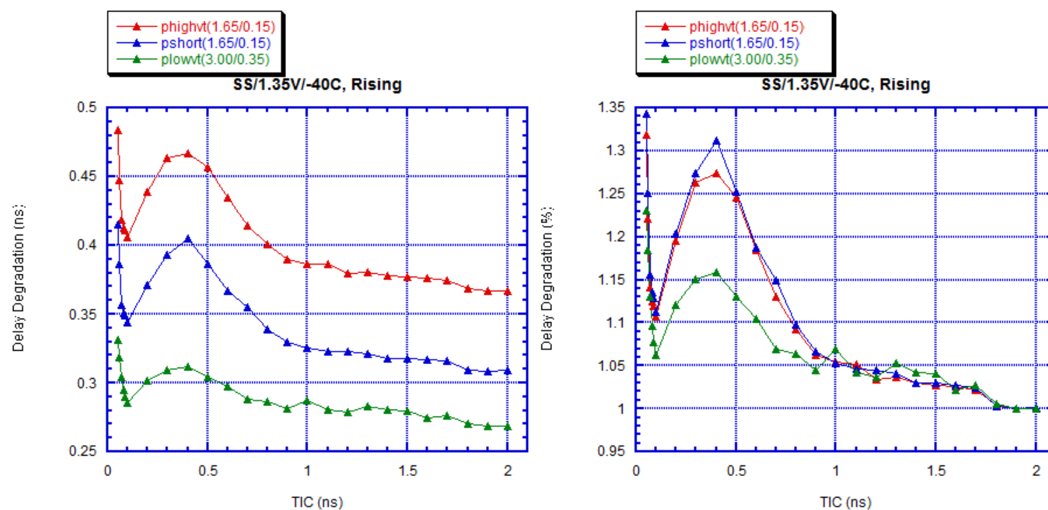

Figure 39. Delay Degradation Patterns for Different PMOS Models

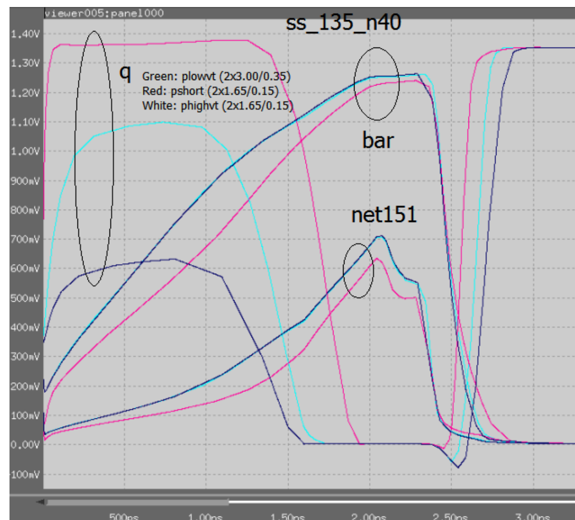

Figure 40. Waveforms for Different PMOS Models

By changing the PMOS model of the inverters (I41 and I72) from phighvt (W/L=2 x 1.62/0.15) to plowvt (W/L=2 x 3.00/0.35), the hill in delay degradation is damped a lot. But it can observed that the "TRU" and "BAR" have no longer enough pre-charging current to be charged close to V_{DD}. So the width (from 0.42/0.15 to 0.55/0.15) of PMOS of both pre-charge path (I76 and I77) is increased to provide enough pre-charging current before clock arrives. One thing noticed is that by changing from phighvt to plowvt, the inverter actually flips earlier than before. In the pre-charge period (before CLK arrives), the output Q will rise higher, from less than $0.5V_{DD}$ to V_{DD}. Considering the next stage is a gating for CLK, as long as the CLK keeps 0, it will not be a problem. According to the design document, only the logic when CLK is active (=1) is considered.

Changing from phighvt to pshort has similar effect. But it doesn't require increasing the drive strength of PMOS of the pre-charge paths.

7.5.7 Schematic vs. Extracted Layout Simulation

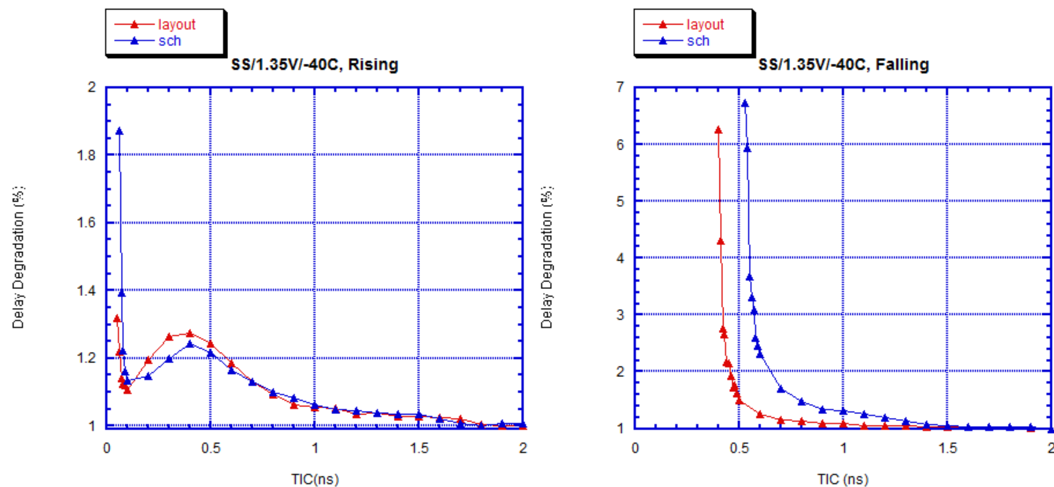

Figure 41. Schematic vs. Extracted Layout Simulation (a) Rising (b) Falling

From Figure 41(a) it can be seen that both schematic and extracted layout simulations show this non-monotonic delay degradation pattern with data input rising. These two curves are close to each other till T_{setup}<0.1ns. After T_{setup}<0.1ns, the schematic simulation shows larger increasing rate.

While from Figure 41 (b), this pre-charge type latch demonstrates monotonic delay degradation pattern, which is similar as the normal DFF does in Figure 33 (b). It is unexpected that even schematic simulation shows this asymmetry because in the simulation with only transistors (schematic netlist), both paths ("TRU" and "BAR") have identical transistor parameters (e.g. W/L, model type, V_{th}, etc.). The only difference in the schematic is there is an extra inverter to generate the reciprocal input signal A_N by taking the A as input, which is shown in Figure 32.

Another thing from Figure 41 (b) is the schematic simulation is worse than extracted layout simulation. The blue curve (schematic) is above the red curve (layout), and when T_{setup}=0.5ns (which is the tSRWB_spec value in the .lib), the extracted layout simulation gives us 40% delay degradation while the circuit fails before T_{setup} reaches 0.5ns in the schematic simulation.

7.5.8 Different Data Input Polarities

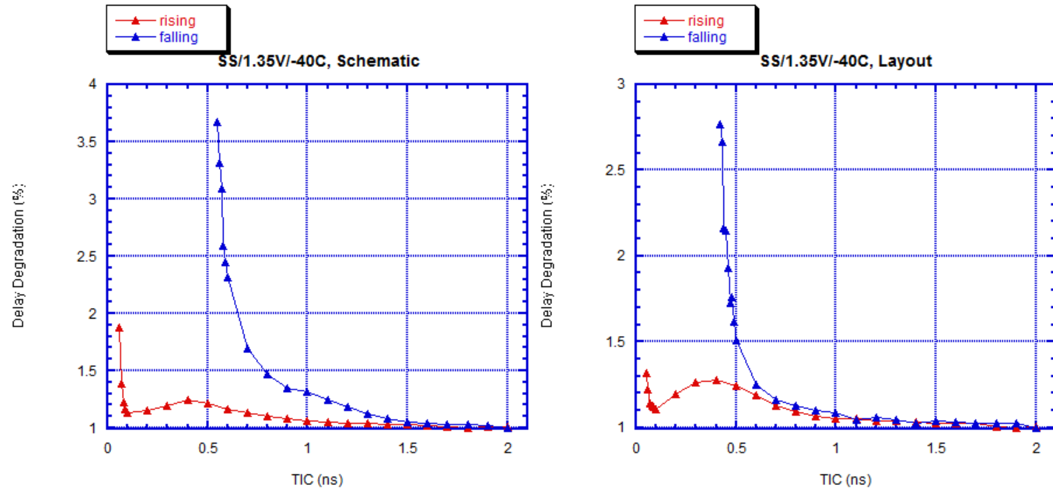

Figure 42. Simulation Results of Different Data Input Polarities (a) Schematic (b) Extracted Layout

From Figure 42 it can be seen that both schematic and extracted layout simulations shows different delay degradation patterns for different data input polarities (rising vs. falling). This pre-charge latch favors the rising data input signal because the red curve in Figure 42 shows much smaller delay degradation than the blue curve does, which means faster propagation. The presumption, which still needs to be proven, is the inverter on the input side causes this asymmetry because, for schematic, the rest logic paths are symmetric. In order to answer this question, we tweak the MOS in this inverter by changing the drive strength, V_{th}, etc. to see if it actually affects this non-monotonic patterns and asymmetric response.

7.5.9 Tweak of the Inverter on the Data Input Path

The idea is since the inverter on the data input side is the only asymmetric part in the entire schematic, this non-monotonic patterns showing only in data input rising polarity should result from it. By tweaking the W/L of either the NMOS or PMOS in this inverter, or completely removing this subcircuit, we could have a better understanding its effect on the non-monotonic pattern.

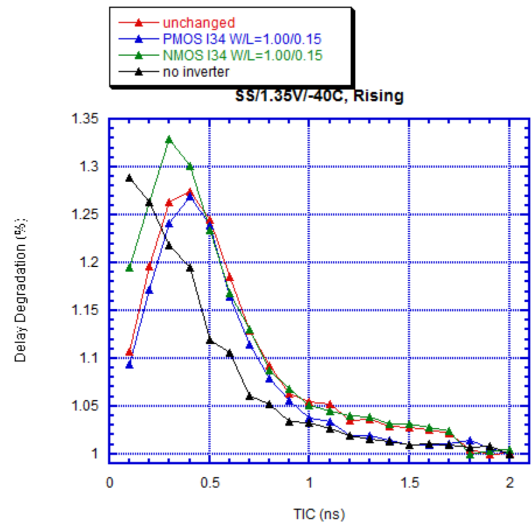

Figure 43. Simulation Results of Tweaking the Inverter

It can be proven from Figure 43 that this non-monotonic pattern results from the asymmetry caused by this inverter. If completely eliminating the inverter (apply stimulus directly on the output of this inverter “A_N”), this non-monotonic pattern disappears.

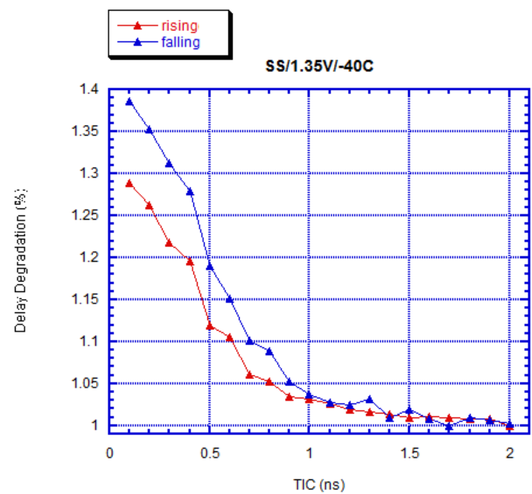

Figure 44. Rising/Falling Simulation Results without the Input Inverter

From Figure 44 it can be seen that both rising and falling are monotonic. One thing needs to be noticed is that there is still observable asymmetry from the curves, falling has larger delay degradation than rising does. Another thing is the shapes of both curves change comparing with Figure 42(b). The reason might be the clock is always positive edge sensitive, which might introduce this asymmetry.

7.5.10 A Proposed Improvement of the Inverter on the Data Input Path

By comparing the waveforms from the simulations with both the unchanged netlist and netlist without the inverter, we propose the reason causing the non-monotonic pattern is the delay from the asymmetric existing of the inverter on the data input path.

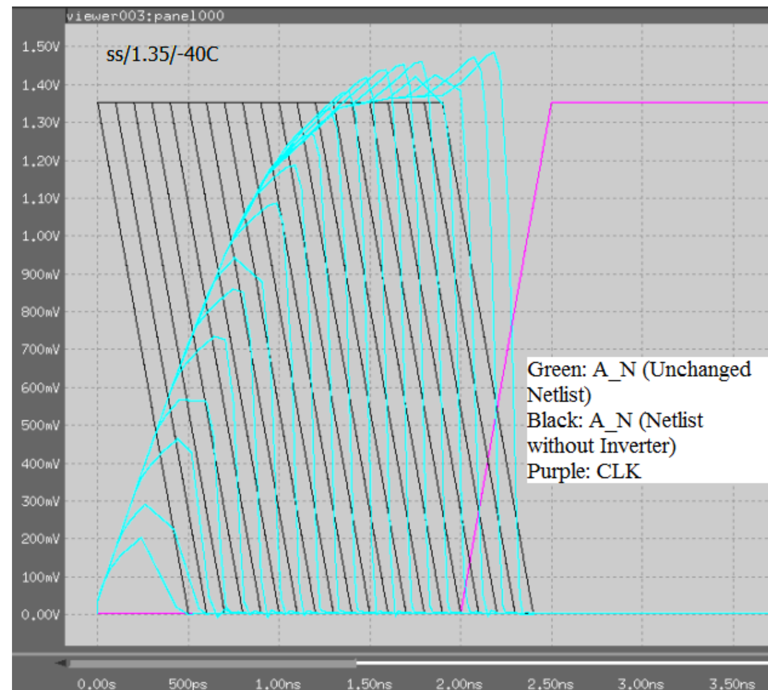

Figure 45. A_N Waveforms with Unchanged Netlist and Netlist without the Inverter

From Figure 45, it can be seen that the waveforms of the actual A_N generated by the inverter is quite different from the waveforms directly forced in the simulation with the netlist without the inverter. So we think the non-monotonic is caused by the delay introduced by the inverter. With this delay, the actual A_N signal can't drop to logic 0 before the CLK becomes active when T_{setup} is small enough. If there is much setup time (T_{setup} is large enough), in other words, input A toggles early enough before the CLK toggles, the inverted signal A_N could have enough time drop from logic 1 to 0. When the input A is more and more close to the CLK, considering the delay introduced by the inverter, the A_N will be high enough to be considered logic 1 when the CLK is active. In this case, both A and A_N are logic 1 when the latch evaluates the input, which turns on both discharge paths and results in a temporary speed up.

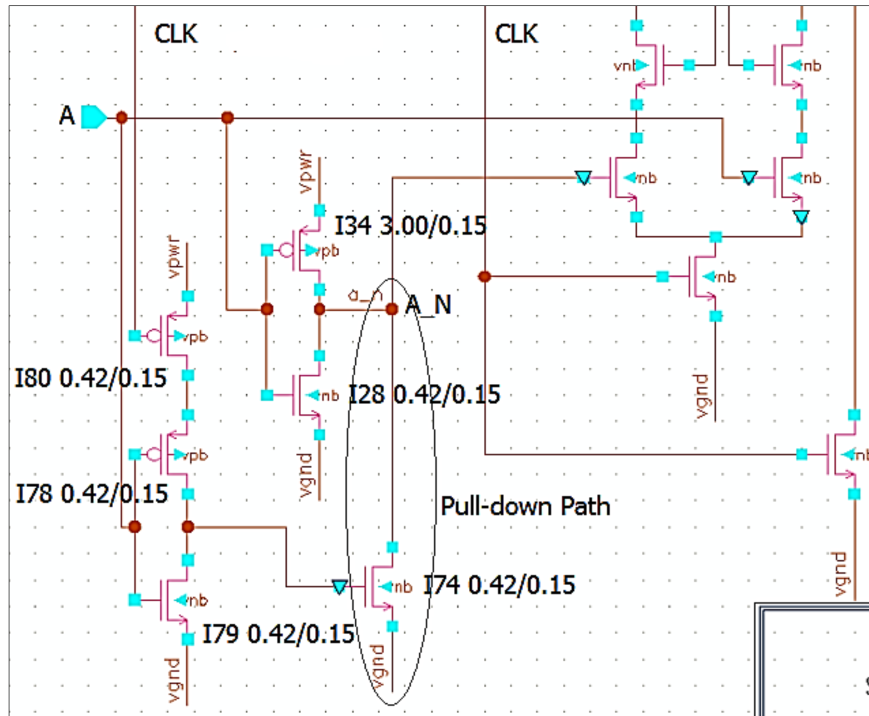

Figure 46. Portion of Pre-charge Latch Schematic Shows the Added Pull-down Path

In order to eliminate this non-monotonic pattern, we try to compensate the delay introduced by the inverter. A pull-down path (shown in Figure 46) is added to the A_N node to pre-discharge the value of A_N to logic 0 so that it doesn't need to wait for the effective input A to arrive. This pull-down path is control by the logic value of input A and CLK so that it will be only turned on when A is logic 0 and CLK is inactive. For the rising polarity scenario (input signal A toggles from logic 0 to 1), this pull-down path turns on for a while then shuts off. For the falling polarity scenario (input signal A toggles from logic 1 to 0), this pull-down path shuts off for a while then turns on, and after a short time, it will be turned off again because CLK is active. Because we tweak the drive strength of the NMOS I74 (W/L=0.42/0.15) used in this pull-down path very weak comparing with the PMOS I34 (W/L=3.00/0.15) in the inverter, this pull-down path can't affect the output logic of the inverter (shown in Figure 46).

7.5.11 Different Versions of the Modified Pre-charge Latch with Pull-down Path

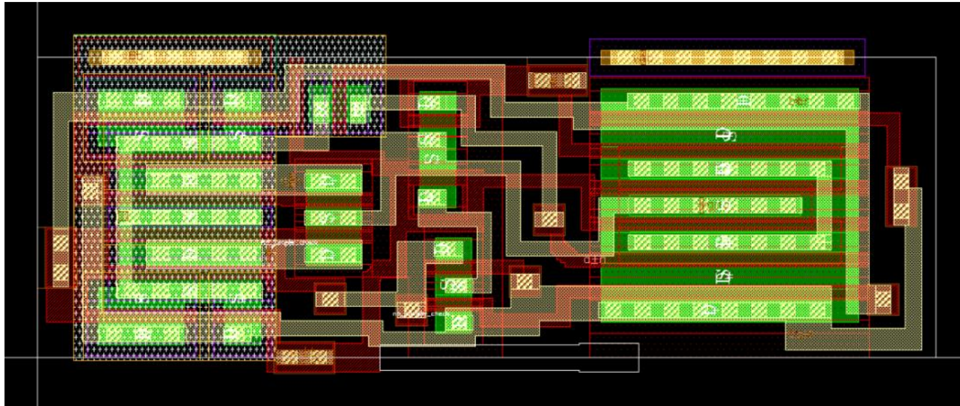

Figure 47. Default Layout

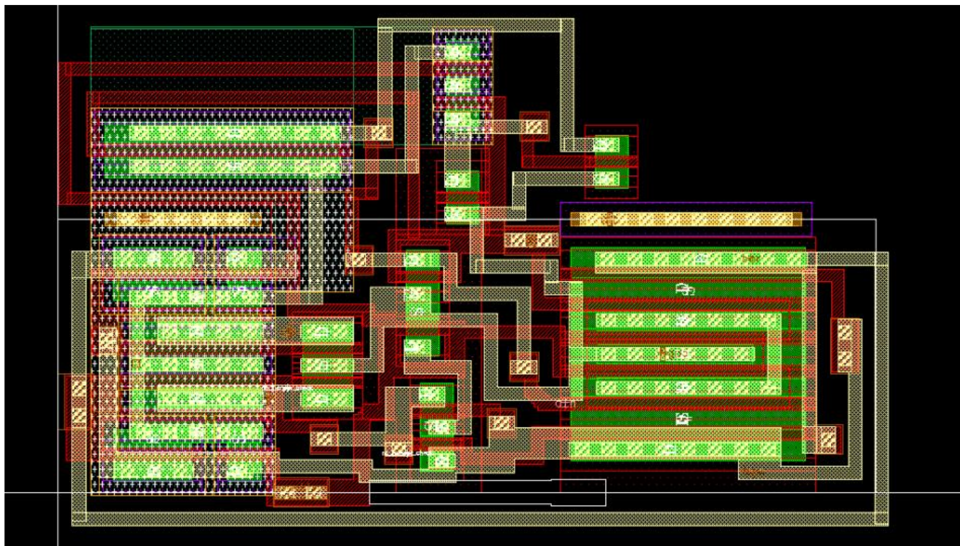

Figure 48. Modified Layout Version 1

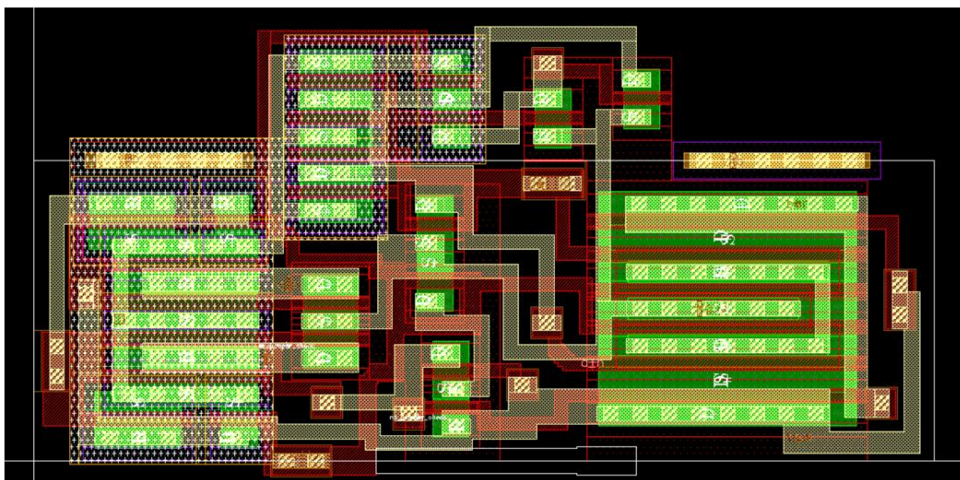

Figure 49. Modified Layout Version 2

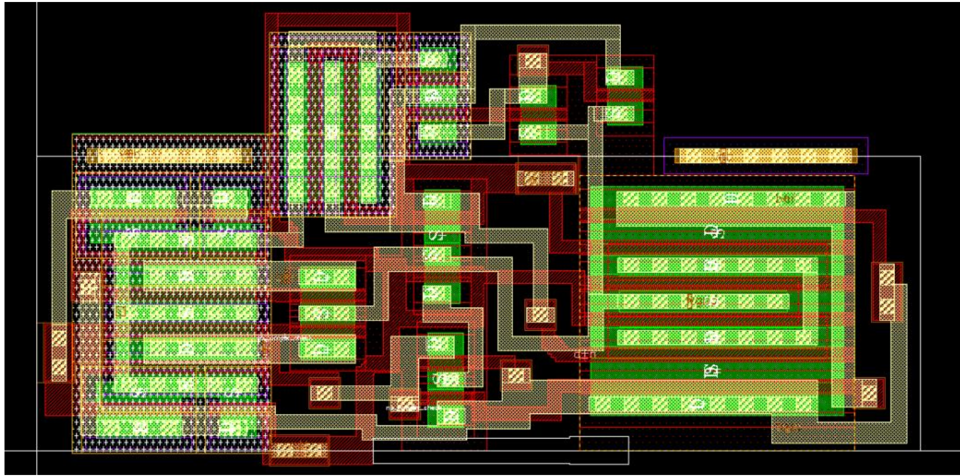

Figure 50. Modified Layout Version 3

Table 9. Different Configurations of the Modified Layouts

	Default	Version 1	Version 2	Version 3
Size	10.20um x 3.93um = 40.04um^2	10.63um x 6.49um = 68.90um^2	10.20um x 5.22um = 53.19um^2	10.2um x 5.34um = 54.62um^2
With Pull-down Path	No	Yes	Yes	Yes
PMOS I34 W/L (um)	0.42/0.15	3.00/0.15	4x0.84/0.15	2x1.65/0.15
M Factor of PMOS I34	1	1	4	2

Based on the default layout, according to the pull-down path design (shown in Figure 46), 4 more transistors needed to be added to the existing layout. Besides that, the W/L of the PMOS I34 (shown in Figure 46) needed to be increased. The version 1 was the first modified design, which confirmed the design correction without taking layout area into consideration. The increased area for version 1 was 69.8%.

Since large area made the version 1 very difficult to fit into the default SRAM layout, much effort was made to shrink the layout. The version 2 was based on the version 1, in order to save area, the M factor of the PMOS I34 was increased, which ended up with 4x0.84/0.15 from 3.00/0.15. The equivalent W/L is larger (4x0.84/0.15 = 3.36/0.15). The simulation results showed this large M factor (leads to different V_{th}) actually affected the falling behavior a lot (discussed in next section), which made the version 2 impractical.

The version 3 was proposed based on the version 2 with decreasing the high M factor from 4 to 2. The W/L of the PMOS I34 was 2x1.65/0.15. The simulation results showed good trade-off between M factor (the low M factor, the better rising/falling behaviors) and small area.

7.5.12 Final Top-level Layout of the SRAM

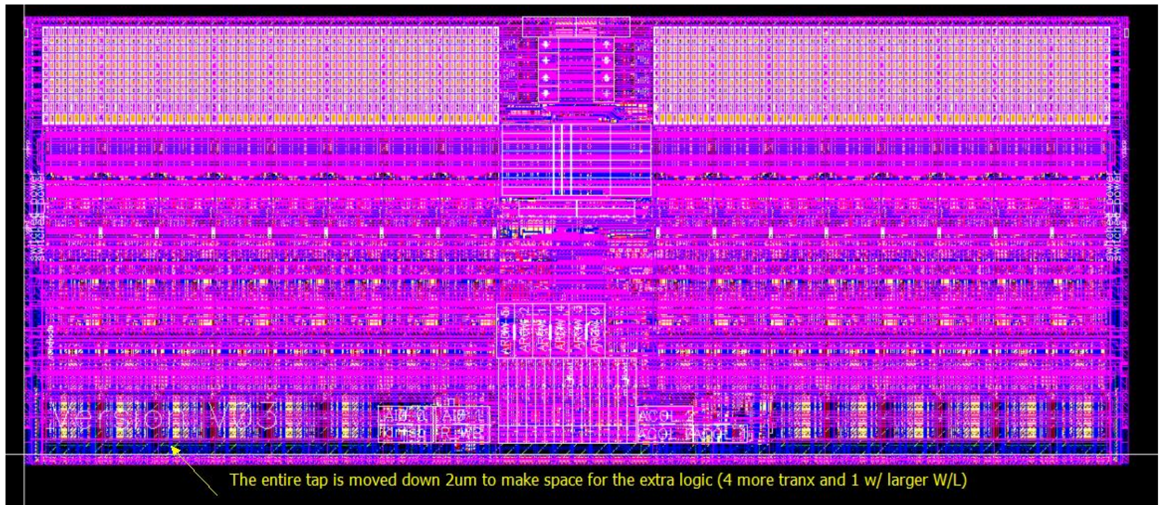

Figure 51. Final Top-level Layout

In order to make space for the extra logic (4 more transistors and 1 PMOS with increased W/L), the entire ring was moved down 2um. The modified top-level layout is 2.365% larger than the default one.

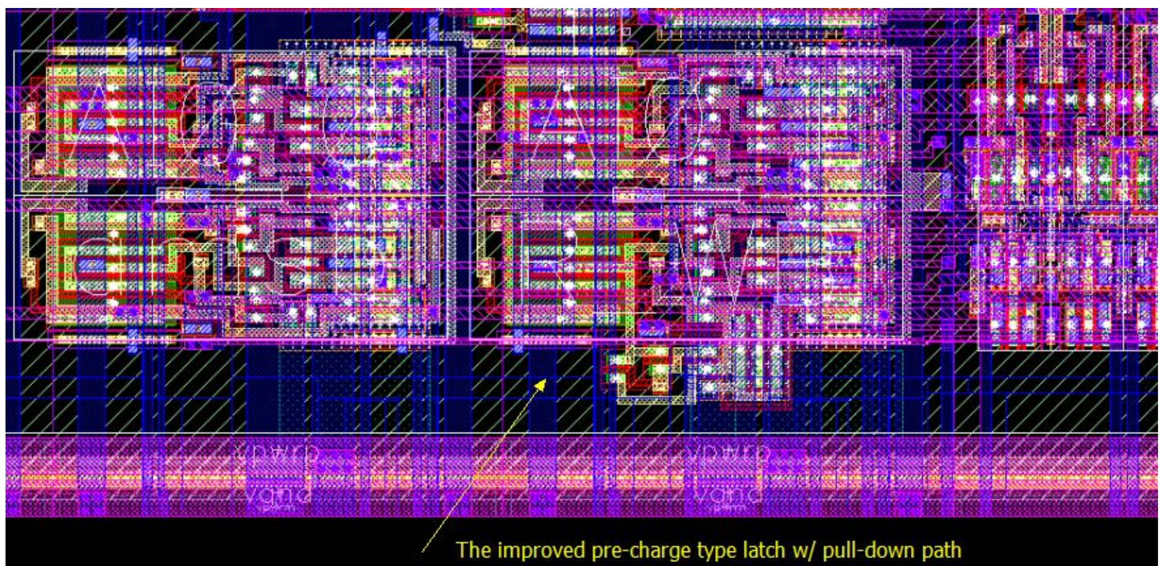

Figure 52. Zoom-in Layout Shows the Improved Pre-charge Type Latch with Pull-down Path

It can be seen that except for the extra space, the added logic of the pull-down path didn't affect any re-route of the default layout, which preserved the hierarchy instantiation.

7.5.12 Simulation Results of Different Versions of the Modified Layouts

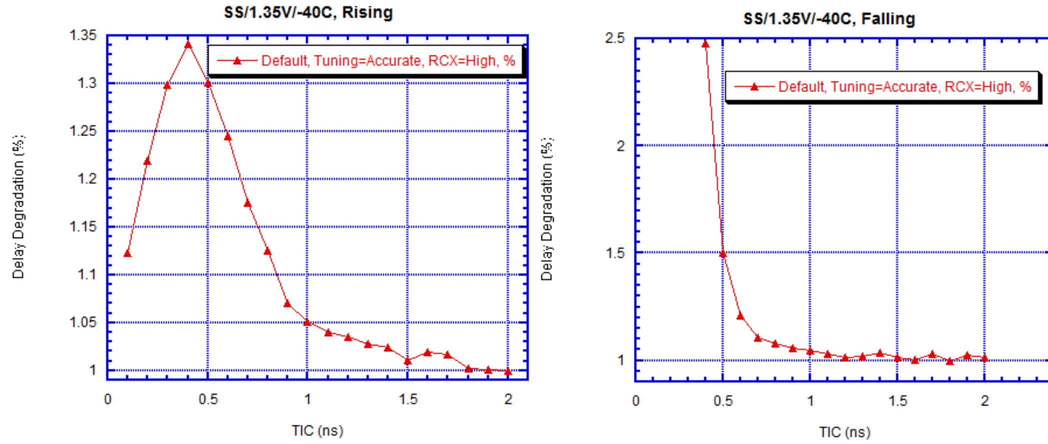

Figure 53. Simulation Results of the Default Layout

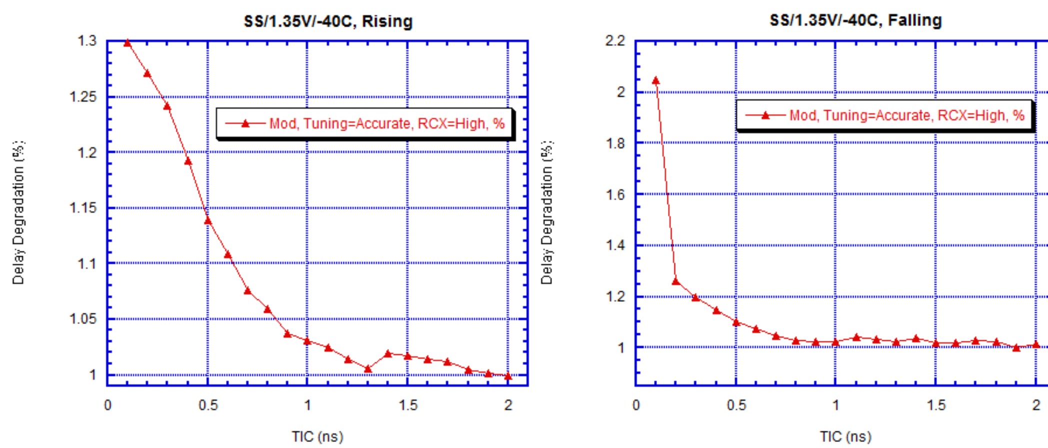

Figure 54. Simulation Results of the Version 1

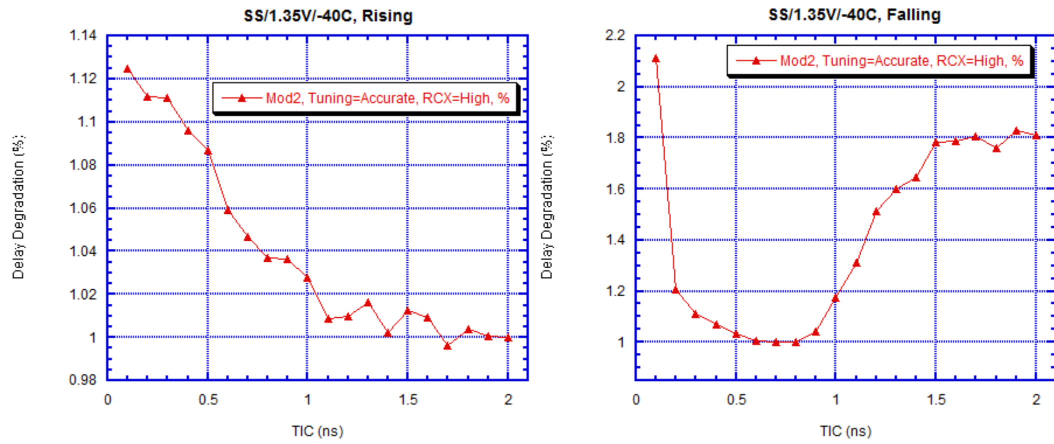

Figure 55. Simulation Results of the Version 2

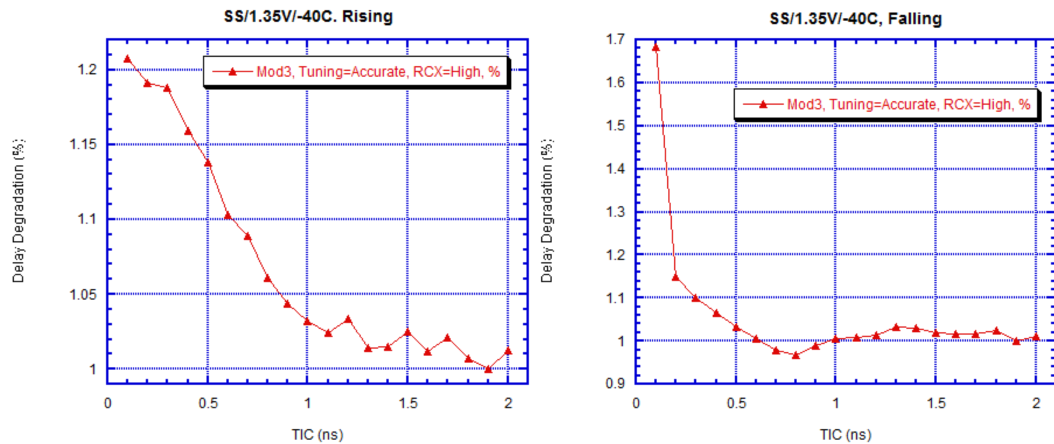

Figure 56. Simulation Results of the Version 3

It can be seen that the simulation result of the rising edge of the default layout had a large bump shown in the delay degradation pattern. The delay degradation pattern for the falling edge was monotonically increasing.

In order to eliminate the large bump shown in the rising edge delay degradation pattern, the pull-down path was added to the default design. From the simulation results of the version 1, it can be seen that the bump was eliminated, and both rising and falling edge delay degradation patterns were monotonically increasing.

In order to save area, the version 2 was based on the version 1. But with large M factor (4), the simulation results of the version 2 showed distortion for the falling edge. The M factor had large effect (different V_{th}) on the falling edge delay degradation pattern, which is shown in next section.

The version 3 was a good trade-off between M factor and small area. With relative small M factor (2), the version 3 kept similar delay degradation patterns (rising/falling) as those of the version 1 with smaller layout area. This design was chosen to be integrated into the SRAM layout, which is shown in Figure 51 Figure 52.

7.5.13 The Effect of M Factor on the Delay Degradation Pattern

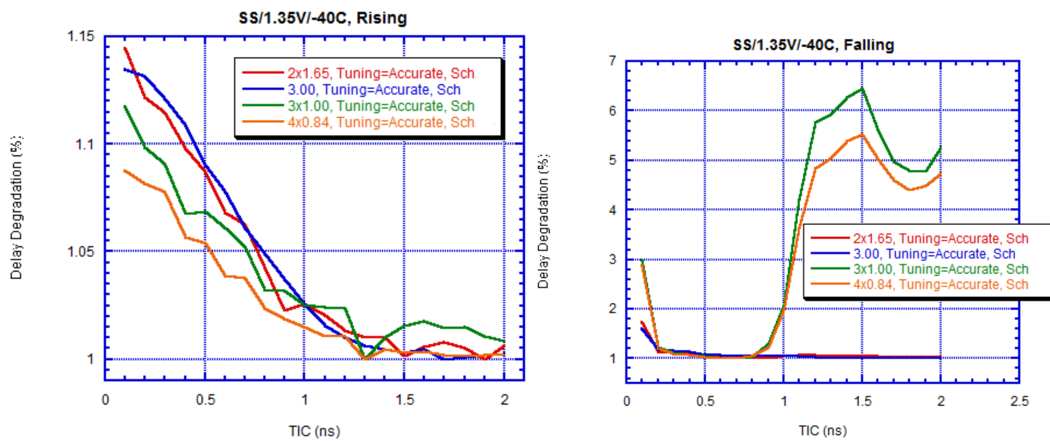

Figure 57. Simulation Results of Rising/Falling Delay Degradation Patterns with Different M Factors

It can be seen that different M factor (the equivalent W/L were similar, around 3.00/0.15) had minor effect on the rising edge delay degradation patterns. But it had huge effect on the falling edge delay degradation patterns. The falling edge delay degradation patterns with large M factor (3/4) had distortion. Even though the drive strength PMOS I34 is kept similar, different M factor results in different V_{th}, which leads to unexpected behavior (distortion) of the pre-charge type latch.

7.6 Stimulus Waveforms

EN will be the first to be active. Since the circuit needs time to initialize after EN goes high, there will be a read cycle without doing anything dedicated to that. There is a feature called “feed-through” that in write cycle, the data written into the SRAM will appear on DO after some delay. Thus it is hard to distinguish whether reading is successful if trying reading the same data right after writing. So two write cycles will be used, which will be the second and third clock cycles, write 0 at address 111111 then write 1 at 000000. After that, in the fourth clock cycle, the simulator will try to read 0 from address 111111 (shown in Figure 58). In this case, if the output DO flips (shown in Figure 58), it is assured that the read 0 at 111111 is successful. Then the Tsetup can be reduced till the output DO does not flip any more (stay in

high). In general, whether the output DO flips will be the indication of whether the circuit works correctly or not.

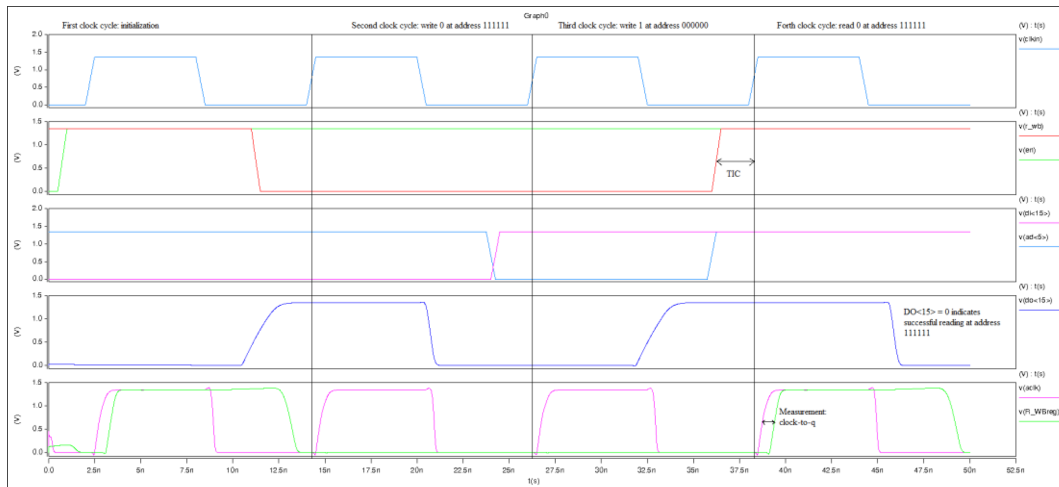

Figure 58. Stimulus Waveforms of tSRWB Simulation (SS/1.35V/-40 ℃, Rising)

7.7 Methodology

To use the same approach as tSDI_spec, the T_{setup}=2ns point is simulated to get the nominal delay from the underlying DFF ACLK to R_WBREG with SS/1.35V/-40 ℃. The T_{setup}=0.5ns (the same as tSRWB_spec in the .lib) will give the delay at that point. Then a margin of 3.5% can be achieved, which will be used in the rest to get the tSRWB_spec associated with these PVT conditions.

7.8 Results

7.8.1 Default Layout (Rising Polarity)

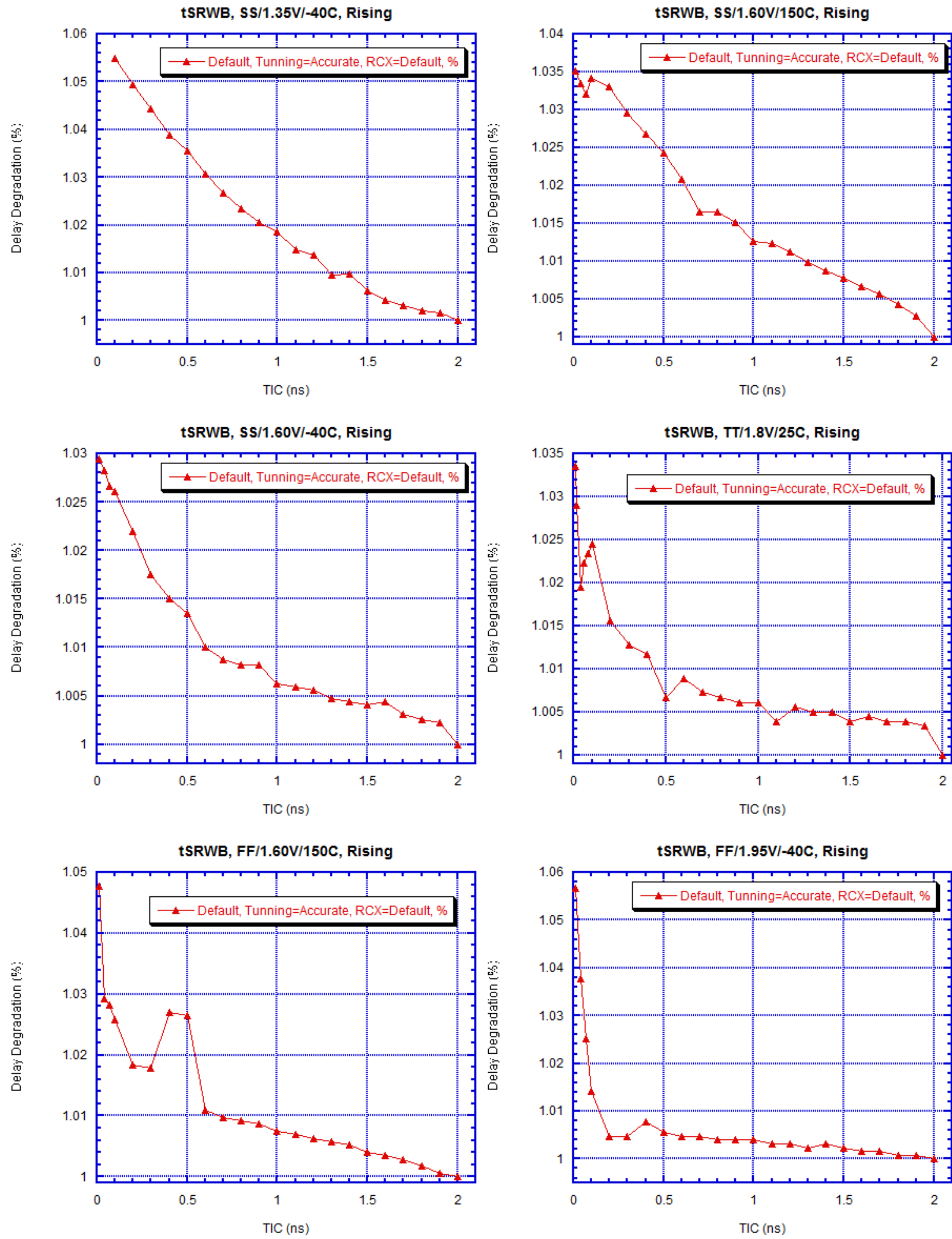

Figure 59. tSRWB Simulation Results of Default Layout (Rising) with Varying PVTs

According to the tSRWB_spec in the .lib, when T_{setup}=0.5ns, for SS/1.35V/-40 ℃, the delay degradation is 3.6%. When applying 3.6% delay degradation to other PVTs, the new tSRWB_spec values are extracted, which are shown in Table 11. The simulation results of tSRWB aren't quite consistent with those of tSDI, that tSRWB_spec values for SS and TT are smaller than those for FF, which is the reason why the underlying pre-charge latch is studied, and modified to eliminate the non-monotonic delay degradation pattern in the individual simulations. We believe such non-monotonic pattern results from the imbalance of the pre-charge latch circuit.

Besides, the non-monotonic delay degradation pattern shown in individual simulations disappears in full circuit simulations for SS corner. But for TT and FF corners, the non-monotonic hills can be still seen in the curves.

7.8.2 Modified Layout Version 3 (Rising Polarity)

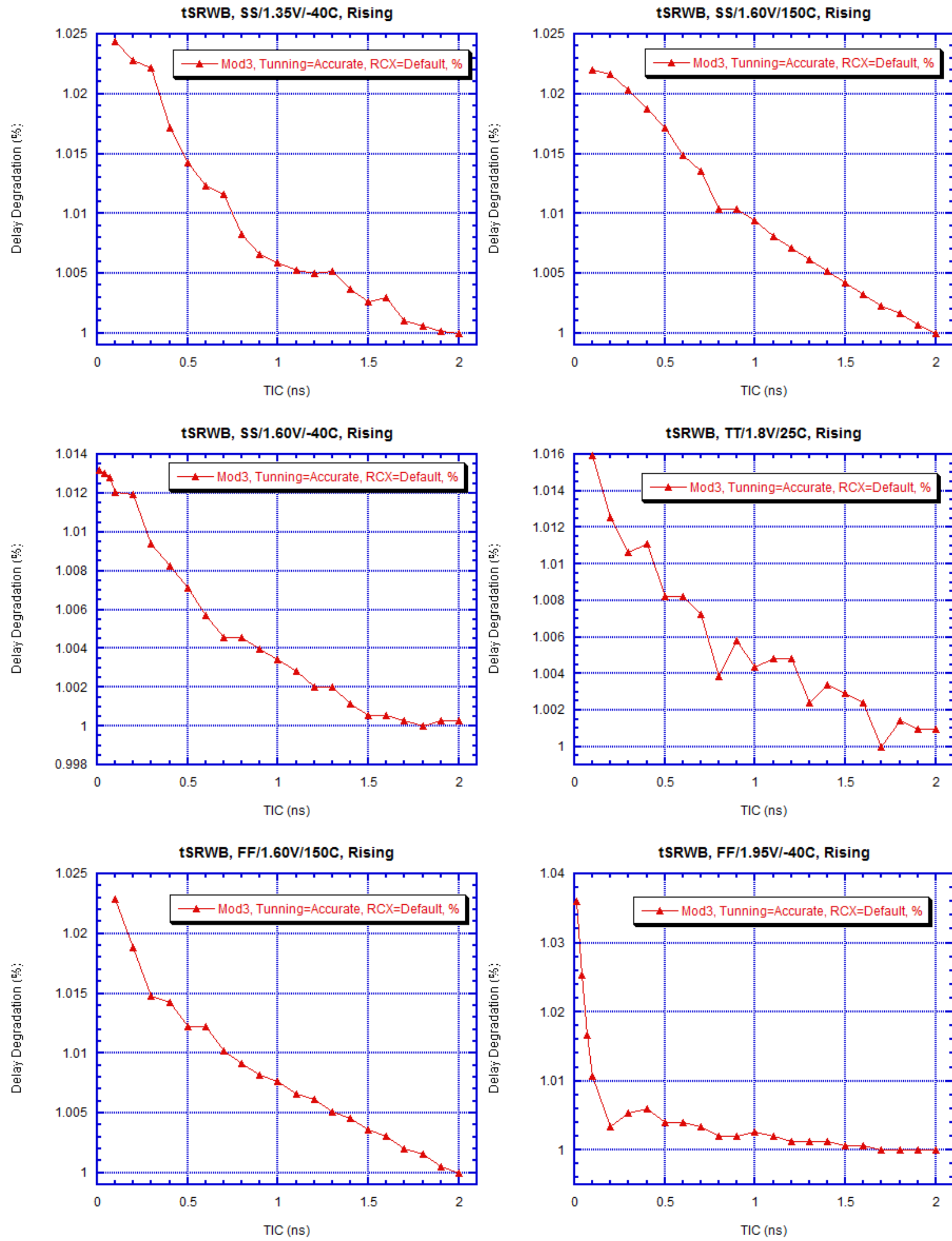

Figure 60. tSRWB Simulation Results of Modified Layout Version 3 (Rising) with Varying PVTs

The modified layout version 3 is designed to eliminate the non-monotonic delay degradation pattern shown in the simulations of the pre-charge latch. The individual simulations, which are shown in previous section (Figure 53 vs Figure 56), confirm the added pull-down path can eliminate the non-monotonic pattern resulting from the imbalance of the input paths (A and A_N). The full circuit simulations shown in Figure 60 indicate such non-monotone, which still exist in TT and FF corners of the default layout (Figure 59 (d)(e)), almost disappear (only a small hill for FF/1.95V/-40 ℃) with the modified layout version 3.

7.8.3 Default Layout (Falling Polarity)

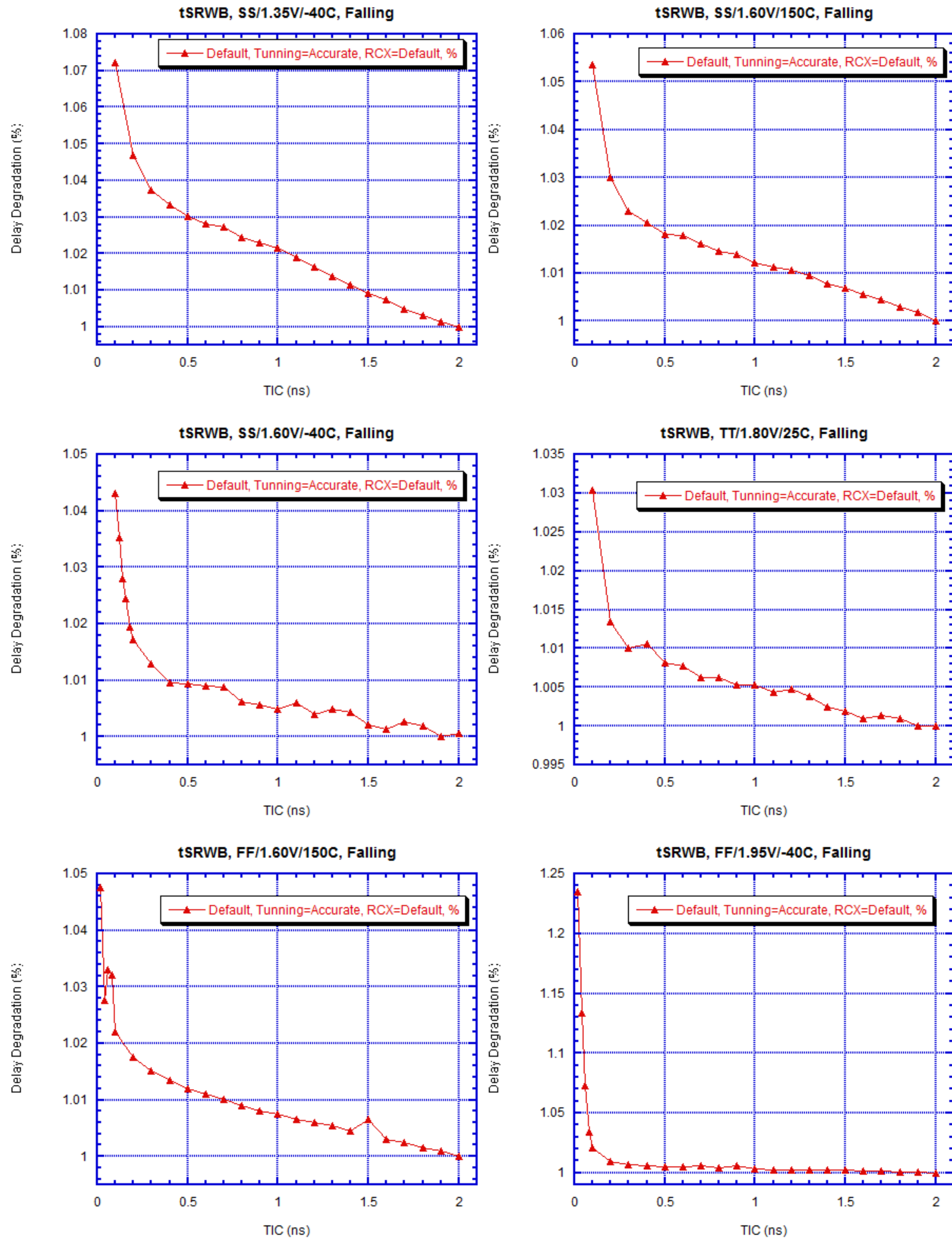

Figure 61. tSRWB Simulation Results of Default Layout (Falling) with Varying PVTs

For the falling polarity, the simulations of the default layout show ideal monotonic delay degradation pattern, with which the tSRWB_spec values associated with each PVT can be easily extracted. For SS/1.35V/-40 ℃, the tSRWB_spec used in the .lib is 0.5ns. When applying T_{setup}=0.5ns, it gives us 3% delay degradation, which are used in other PVTs to get each tSRWB_spec. The simulation results (shown in Table 11) are consistent with the assumption that faster circuit has smaller tSRWB_spec value.

7.8.4 Modified Layout Version 3 (Falling Polarity)

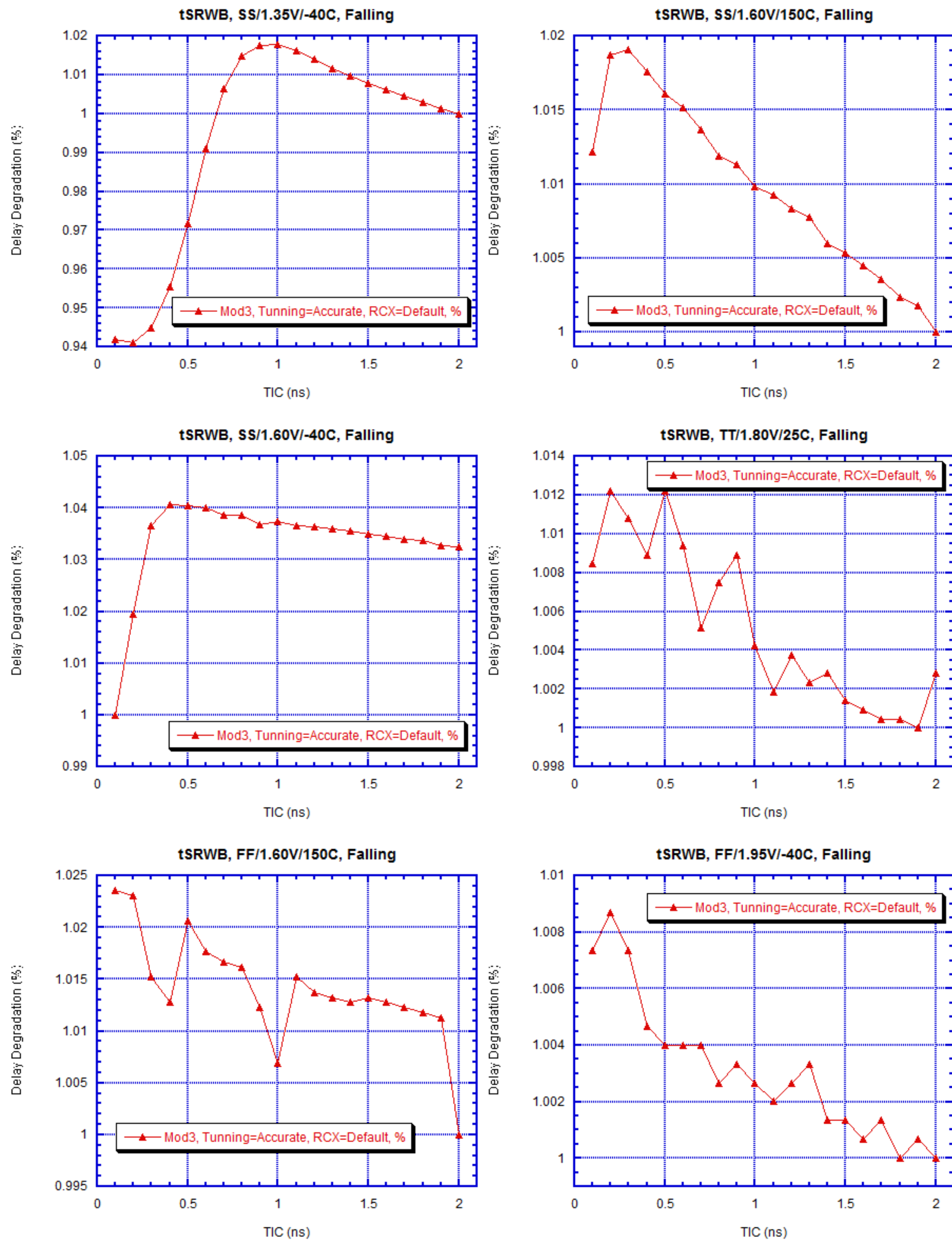

Figure 62. tSRWB Simulation Results of Modified Layout Version 3 (Falling) with Varying PVTs

Even through the modified layout version 3 works well for the rising polarity, the simulation results for the falling polarity are abnormal. Especially for SS corner, the delay degradation patterns are falling instead of rising when T_{setup} gets smaller. Even for TT and FF corners, although the general trend is increasing when T_{setup} gets smaller, there're a lot of fluctuation which doesn't show in the simulations of the default layout. Our modified layout version 3 somehow doesn't work well for the falling polarity, preventing further use before fixing this issue.

Chapter 8 Data Reading Delay (tRD)

8.1 Equation

Similar with the tWR, the tRD also has three terms, two from the subcircuit delay measurement and one spec value. The *T_CLKCTL_del_r_a* is the delay from top-level CLKin to local clock CLK_LOC which triggers the underlying DFF. The *T_DO_del_r/f_a* is the delay from DO_I_N to top-level DO. Like tWR, the *tRD_spec* has three different values (minimal, typical and maximal). The *tRD_spec* has variation across process, in other words, the compiler uses the minimal value for FF, the typical value for TT and maximal value for SS.

$$tRD_rr_ar = T_CLKCTL_del_r_a + tRD_spec + T_DO_del_r_a$$

$$tRD_rf_ar = T_CLKCTL_del_r_a + tRD_spec + T_DO_del_f_a$$

Equation 7

In Equation 7:

- *T_CLKCTL_del_r_a* is the delay from top-level CLKin to local CLK_LOC which triggers the underlying DFF.
- *T_DO_del_r/f_a* is the delay from DO_I_N to top-level DO.
- *tRD_spec* has three different values for minimal, typical and maximal conditions.

Table 10. tRD_spec Values for Different Processes

Min (FF)	Typ (TT)	Max (SS)
0.500ns	2.330ns	4.620ns

8.2 Schematic

The Figure 63 shows the brief schematic of tRD. It can be seen that the equation-based method is literally adding all the major delays of the path from CLKin to DO. The *T_CLKCTL_del_r_a* counts the delay of clock signal, and *T_DO_del_r_a* counts the delay of output buffer (the blue rectangle between DO_I_N and DO). We assume the delays for the rest parts is included in the tRD_spec and will not change with different output load capacitances and signal ramp time.

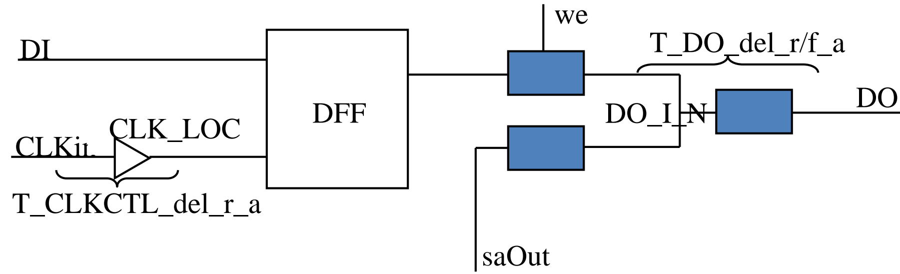

Figure 63. Schematic of tRD

8.3 Results

For direct measurement of tRD_spec, the delay from CLKin to DO in read cycle is considered to be tRD_spec. Different simulated tRD_spec for different PVT conditions are shown in Table 11.

For the rising polarity, similar like tWR_spec, since the maximum value of tRD_spec is based on SS/1.60V/-40 ℃ or SS/1.60V/150 ℃ (depending on which one is larger), it is reasonable that simulated tRD_spec of SS/1.35V/-40 ℃ is larger than the maximum value here. But for the minimal value, which should be based on FF/1.60V/150 ℃ or FF/1.95V/-40 ℃ (depending on which one is smaller), the simulated values are larger than 0.5ns unexpectedly. Same case for the falling polarity.

8.4 Validating

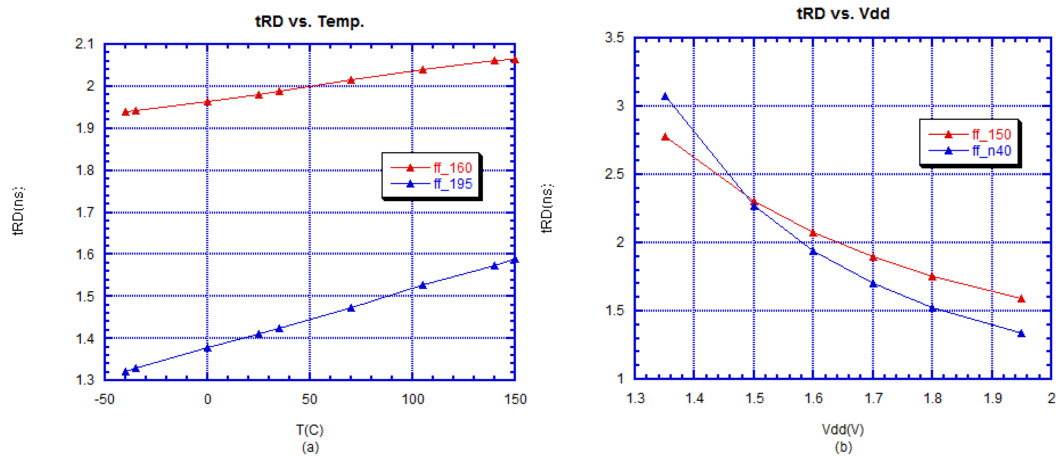

Figure 64. tRD Simulation Results with Varying Temperature and V_{DD}

The tRD_spec for FF/1.95V/-40 ℃ is 2.6X larger than the .lib values (shown in Table 11). In order to have a sanity check to prove the methodology is correct, for both FF/1.60V/150 ℃ and FF/1.95V/-40 ℃, we vary one of the temperature (T) and voltage (V_{DD}) keep the other one intact.

The simulated curves are as expected, that higher temperature means more delay because the circuit is slower (Figure 64(a)). Higher V_{DD} indicates faster circuit (Figure 64(b)).

Chapter 9 Final Results

Table 11. Final Results

PVT	Layout	Param	Polarity	.lib (ns)	Simulation (ns)
SS/1.35V/-40 ℃	Default	tSDI	Rising	0.700	0.700
SS/1.60V/-40 ℃	Default	tSDI	Rising	0.700	0.300
SS/1.60V/150 ℃	Default	tSDI	Rising	0.700	0.250
TT/1.80V/25 ℃	Default	tSDI	Rising	0.700	0.120
FF/1.60V/150 ℃	Default	tSDI	Rising	0.700	0.090
FF/1.95V/-40 ℃	Default	tSDI	Rising	0.700	0.050
SS/1.35V/-40 ℃	Default	tSDI	Falling	0.700	0.700
SS/1.60V/-40 ℃	Default	tSDI	Falling	0.700	0.700
SS/1.60V/150 ℃	Default	tSDI	Falling	0.700	0.600
TT/1.80V/25 ℃	Default	tSDI	Falling	0.700	0.700
FF/1.60V/150 ℃	Default	tSDI	Falling	0.700	0.700
FF/1.95V/-40 ℃	Default	tSDI	Falling	0.700	0.700
SS/1.35V/-40 ℃	Default	tHDI	Rising	0.980	0.200
SS/1.60V/-40 ℃	Default	tHDI	Rising	0.980	0.400
SS/1.60V/150 ℃	Default	tHDI	Rising	0.980	0.460
TT/1.80V/25 ℃	Default	tHDI	Rising	0.540	0.500
FF/1.60V/150 ℃	Default	tHDI	Rising	0.560	0.530
FF/1.95V/-40 ℃	Default	tHDI	Rising	0.560	0.560
SS/1.35V/-40 ℃	Default	tHDI	Falling	0.980	0.720
SS/1.60V/-40 ℃	Default	tHDI	Falling	0.980	0.640
SS/1.60V/150 ℃	Default	tHDI	Falling	0.980	0.700
TT/1.80V/25 ℃	Default	tHDI	Falling	0.540	0.600
FF/1.60V/150 ℃	Default	tHDI	Falling	0.560	0.640
FF/1.95V/-40 ℃	Default	tHDI	Falling	0.560	0.560
SS/1.35V/-40 ℃	Default	tWR	Rising	4.040	6.563

SS/1.60V/-40 ℃	Default	tWR	Rising	4.040	3.211
SS/1.60V/150 ℃	Default	tWR	Rising	4.040	3.213
TT/1.80V/25 ℃	Default	tWR	Rising	1.960	1.698
FF/1.60V/150 ℃	Default	tWR	Rising	0.500	1.633
FF/1.95V/-40 ℃	Default	tWR	Rising	0.500	1.075
SS/1.35V/-40 ℃	Default	tWR	Falling	4.040	6.581
SS/1.60V/-40 ℃	Default	tWR	Falling	4.040	3.136
SS/1.60V/150 ℃	Default	tWR	Falling	4.040	3.128
TT/1.80V/25 ℃	Default	tWR	Falling	1.960	1.653
FF/1.60V/150 ℃	Default	tWR	Falling	0.500	1.622
FF/1.95V/-40 ℃	Default	tWR	Falling	0.500	1.040
SS/1.35V/-40 ℃	Default	tSRWB	Rising	0.500	0.500
SS/1.60V/150 ℃	Default	tSRWB	Rising	0.500	0.010
SS/1.60V/-40 ℃	Default	tSRWB	Rising	0.500	0.010
TT/1.80V/25 ℃	Default	tSRWB	Rising	0.500	0.010
FF/1.60V/150 ℃	Default	tSRWB	Rising	0.500	0.030
FF/1.95V/-40 ℃	Default	tSRWB	Rising	0.500	0.050
SS/1.35V/-40 ℃	Mod Ver 3	tSRWB	Rising	0.500	0.500
SS/1.60V/150 ℃	Mod Ver 3	tSRWB	Rising	0.500	0.650
SS/1.60V/-40 ℃	Mod Ver 3	tSRWB	Rising	0.500	0.010
TT/1.80V/25 ℃	Mod Ver 3	tSRWB	Rising	0.500	0.150
FF/1.60V/150 ℃	Mod Ver 3	tSRWB	Rising	0.500	0.400
FF/1.95V/-40 ℃	Mod Ver 3	tSRWB	Rising	0.500	0.080
SS/1.35V/-40 ℃	Default	tSRWB	Falling	0.500	0.500
SS/1.60V/150 ℃	Default	tSRWB	Falling	0.500	0.200
SS/1.60V/-40 ℃	Default	tSRWB	Falling	0.500	0.130
TT/1.80V/25 ℃	Default	tSRWB	Falling	0.500	0.100
FF/1.60V/150 ℃	Default	tSRWB	Falling	0.500	0.090
FF/1.95V/-40 ℃	Default	tSRWB	Falling	0.500	0.080
SS/1.35V/-40 ℃	Mod Ver 3	tSRWB	Falling	0.500	0.500
SS/1.60V/150 ℃	Mod Ver 3	tSRWB	Falling	0.500	Can’t Measure

SS/1.60V/-40 ℃	Mod Ver 3	tSRWB	Falling	0.500	Can't Measure
TT/1.80V/25 ℃	Mod Ver 3	tSRWB	Falling	0.500	Can't Measure
FF/1.60V/150 ℃	Mod Ver 3	tSRWB	Falling	0.500	Can't Measure
FF/1.95V/-40 ℃	Mod Ver 3	tSRWB	Falling	0.500	Can't Measure
SS/1.35V/-40 ℃	Default	tRD	Falling	4.740	7.636
SS/1.60V/150 ℃	Default	tRD	Falling	4.740	3.924
SS/1.60V/-40 ℃	Default	tRD	Falling	4.740	3.866
TT/1.80V/25 ℃	Default	tRD	Falling	2.360	2.082
FF/1.60V/150 ℃	Default	tRD	Falling	0.500	2.054
FF/1.95V/-40 ℃	Default	tRD	Falling	0.500	1.318
SS/1.35V/-40 ℃	Mod Ver 3	tRD	Falling	4.740	7.733
SS/1.60V/150 ℃	Mod Ver 3	tRD	Falling	4.740	3.923
SS/1.60V/-40 ℃	Mod Ver 3	tRD	Falling	4.740	3.866
TT/1.80V/25 ℃	Mod Ver 3	tRD	Falling	2.360	2.082
FF/1.60V/150 ℃	Mod Ver 3	tRD	Falling	0.500	2.052
FF/1.95V/-40 ℃	Mod Ver 3	tRD	Falling	0.500	1.317

The simulations for tSDI_spec, tHDI_sim, tWR and tRD meet our expectation that such methodology can provide an insight for the SRAM design by extracting more realistic central points of the timing matrices in .lib when the commercial memory compiler uses them as the starting point to generate entire timing table. The simulations of tSRWB_spec are abnormal mainly because the register used in the R_WB datapath is a kind of pre-charge type latch which exhibits non-monotonic delay degradation pattern. By adding a pull-down path to the existing design, such non-monotonic pattern can be mitigated, but the falling polarity scenario is worse than before. In general, the original design is a better trade-off between both polarities.

Appendix A: Generic Perl Script for Individual DFF Simulation

```
#!/usr/local/bin/perl
# Developer: Xiaowei Zhang

use 5.008;
use warnings;
use diagnostics;
use strict;
use Switch;

if(@ARGV != 1){
    &syntax;
}

my $input_cor = shift;
#my $input_cir = shift;
my $counter = 0;
#my $line = <STDIN>;
#chomp($line);

open CORNER, '<', $input_cor or die "Cannot open $input_cor: $!";
my @line_cor = <CORNER>;
close CORNER;

foreach(@line_cor){
    my $proc = 'tt';
    my $vdd = '1.8';
    my $temp = '25';
    my $load = '5f';
    my $ramp = '0.5n';
    my $tuning = 'accurate';
    my $step = '0.001n';
    my $end = '12n';

    my $nl = 's8tssc_lf_dff.nl';
    my $nl_flag = 0;

    my $subckt = 's8tssc_lf_dff';
    my $ports = 'vgnd vpwr vnb clk q q_n vpb a';
    my $a = 'd';

    my $clk = 'clk';
    my $clk_flag = 0;

    my $clk_n = 'clk_n';
    my $clk_n_flag = 0;

    my $q = 'q';

    my $tmod = 'test';
    my $tm_val = 'vdd';
    my $tm_flag = 0;

    my $reset = 'reset';
    my $reset_val = 'vdd';
    my $reset_flag = 0;

    my $set = 'set';
    my $set_val = 'vdd';
    my $set_flag = 0;

    my $start = '2n';
    my $stop = '0.1n';
    my $incr = '0.1n';

    my $modes = 'linear';
```

```
my $md_flag = 0;

my $type = 'setup';
my $delta = '2n';
my $opmod = 'passfail';
my $tol = 0.01;
my $opstart = '1n';
my $lower = '0.1n';
my $upper = '2n';
my $perc = 1.05;
my $nom = '0.12n';
my $goal = '0.13n';
my $output_cir = 'dff.cir';

$counter++;
chomp;
if(/\A\*.*/i or /\A\#.*/i){
    next;
}

s/\s+//g;
my @param_pair = split /;/;
foreach(@param_pair){
    chomp;
    my $key;
    my $value;
    ($key, $value) = split /=/;
    chomp($key);
    chomp($value);

    switch($key){
        case 'proc'{
            $proc = $value;
            print "Process: $proc\n";
        }
        case 'vdd'{
            $vdd = $value;
            print "Vdd = $vdd\n";
        }
        case 'temp'{
            $temp = $value;
            print "Temp = $temp\n";
        }
        case 'load'{
            $load = $value;
            print "Cap load = $load\n";
        }
        case 'ramp'{
            $ramp = $value;
            print "Ramp rate = $ramp\n";
        }
        case 'tuning'{
            $tuning = $value;
            print "Tuning factor: $tuning\n";
        }
        case 'step'{
            $step = $value;
            print "Sim stepwise = $step\n";
        }
        case 'end'{
            $end = $value;
            print "Sim length = $end\n";
        }
        case 'netlist'{
            $nl = $value;
            print "Netlist file: $nl\n";
        }
        case 'input'{
```

```
        $a = $value;
    }
    case 'clk'{
        $clk = $value;
        $clk_flag = 1;
    }
    case 'clk_n'{
        $clk_n = $value;
        $clk_n_flag = 1;
    }
    case 'output'{
        $q = $value;
    }
    case 'tmod'{
        $tmod = $value;
        $tm_flag = 1;
    }
    case 'tmod_val'{
        $tm_val = $value;
    }
    case 'reset'{
        $reset = $value;
        $reset_flag = 1;
    }
    case 'reset_val'{
        $reset_val = $value;
    }
    case 'set'{
        $set = $value;
        $set_flag = 1;
    }
    case 'set_val'{
        $set_val = $value;
    }
    case 'mode'{
        $modes = $value;
        print "Sim mode: $modes\n";
    }
    case 'type'{
        $type = $value;
        print "Sim Type: $type\n";
    }
    case 'delta'{
        $delta = $value;
        print "Delay from clk to output = $delta\n";
    }
    case 'start'{
        $start = $value;
        print "Linear sweep start = $start\n";
    }
    case 'stop'{
        $stop = $value;
        print "Linear sweep stop = $stop\n";
    }
    case 'incr'{
        $incr = $value;
        print "Linear sweep increment = $incr\n";
    }
    case 'opmod'{
        $opmod = $value;
        print "Optimization mode: $opmod\n";
    }
    case 'tol'{
        $tol = $value;
        print "Optimization tolerance = $tol\n";
    }
    case 'opstart'{
        $opstart = $value;
```

```
            print "Optimization start = $opstart\n";
        }
        case 'lower'{
            $lower = $value;
            print "Optimization lower band = $lower\n";
        }
        case 'upper'{
            $upper = $value;
            print "Optimization upper band = $upper\n";
        }
        case 'perc'{
            $perc = $value;
            print "Delay degradation percentage = $perc\n";
        }
        case 'nom'{
            $nom = $value;
            print "Nominal delay = $nom\n";
        }
        else{}
    }
}

open NETLIST, '<', $nl or die "Cannot open $nl: $!";
my @line_nl = <NETLIST>;
close NETLIST;
while(@line_nl){
    $_ = shift @line_nl;
    chomp;
    if(/.*subckt\s*(\w*)\s*(.*)/i){
        $nl_flag = 1;
        $subckt = $1;
        $ports = $2;
        #LOOP:
        $_ = shift @line_nl;
        chomp;
        #if(/\A\+(.*)/i){
        while(/\A\+(.*)/i){
            $ports = "$ports"." "."$1";
            #goto LOOP;
            $_ = shift @line_nl;
            chomp;
        }
    }
}
if($nl_flag == 0){
    die "Can't find subckt in netlist $nl\n";
}

$_ = $ports;
if(/\A$a\s.*/i or /.*\s$a\s.*/i or /.*\s$a\z/i){
    print "Input port name: $a\n";
}else{
    die "Can't find input port $a in netlist $nl\n";
}
if($clk_flag == 1){
    if(/\A$clk\s.*/i or /.*\s$clk\s.*/i or /.*\s$clk\z/i){
        print "Clk port name: $clk\n";
    }else{
        die "Can't find clk port $clk in netlist $nl\n";
    }
}
if($clk_n_flag == 1){
    if(/\A$clk_n\s.*/i or /.*\s$clk_n\s.*/i or /.*\s$clk_n\z/i){
        print "Clk_n port name: $clk_n\n";
    }else{
        die "Can't find clk_n port $clk_n in netlist $nl\n";
    }
}
```

```
    if($clk_flag == 0 and $clk_n_flag == 0){
        die "Can't find clk port in netlist $nl\n";
    }
    if(/\A$q\s.*/i or /.*\s$q\s.*/i or /.*\s$q\z/i){
        print "Output port name: $q\n";
    }else{
        die "Can't find output port $q in netlist $nl\n";
    }
    if($tm_flag == 1){
        if(/\A$tmod\s.*/i or /.*\s$tmod\s.*/i or /.*\s$tmod\z/i){
            print "Test mode port name: $tmod\n";
        }else{
            die "Can't find test mode port $tmod in netlist $nl\n";
        }
    }
    if($reset_flag == 1){
        if(/\A$reset\s.*/i or /.*\s$reset\s.*/i or /.*\s$reset\z/i){
            print "Reset port name: $reset\n";
        }else{
            die "Can't find reset port $reset in netlist $nl\n";
        }
    }
    if($set_flag == 1){
        if(/\A$set\s.*/i or /.*\s$set\s.*/i or /.*\s$set\z/i){
            print "Set port name: $set\n";
        }else{
            die "Can't find set port $set in netlist $nl\n";
        }
    }

    if($modes eq 'single'){
        $output_cir =
"$proc"."_"."$vdd"."_"."$temp"."_"."$subckt"."_"."$modes"."_"."$type"."_"."$
delta";
        $md_flag = 1;
    }elsif($modes eq 'linear'){
        $output_cir =
"$proc"."_"."$vdd"."_"."$temp"."_"."$subckt"."_"."$modes"."_"."$type"."_"."$
start"."_"."$stop"."_"."$incr";
        $md_flag = 2;
    }else{
        if($opmod ne 'passfail'){
            $output_cir =
"$proc"."_"."$vdd"."_"."$temp"."_"."$subckt"."_"."$modes"."_"."$opmod"."_"."
$tol"."_"."$opstart"."_"."$lower"."_"."$upper"."_"."$perc"."_"."$nom";
            $md_flag = 3;
        }else{
            $output_cir =
"$proc"."_"."$vdd"."_"."$temp"."_"."$subckt"."_"."$modes"."_"."$opmod"."_"."
$tol"."_"."$lower"."_"."$upper";
            $md_flag = 4;
        }
    }

    $nom =~ s/(\d+)\w+/$1/i;
    $goal = $perc * $nom;
    $goal = "$goal"."n";

    #open CIR, '<', $input_cir or die "Cannot open $input_cir: $!";
    #my @line_cir = <CIR>;
    my @line_cir = split /\n/, &eldoTemplate;
    #close CIR;
    open OUTPUT, '>', "$output_cir.cir" or die "Cannot open $output_cir:
$!";

    foreach (@line_cir){
        chomp;
        if($proc eq 'ff'){
```

```
    s/\*?(.*include.*ff.*.cor)/$1/i;
    s/\*?(.*include.*tt.*.cor)/\*$1/i;
    s/\*?(.*include.*ss.*.cor)/\*$1/i;
}elsif($proc eq 'tt'){
    s/\*?(.*include.*ff.*.cor)/\*$1/i;
    s/\*?(.*include.*tt.*.cor)/$1/i;
    s/\*?(.*include.*ss.*.cor)/\*$1/i;
}else{
    s/\*?(.*include.*ff.*.cor)/\*$1/i;
    s/\*?(.*include.*tt.*.cor)/\*$1/i;
    s/\*?(.*include.*ss.*.cor)/$1/i;
}
s/(.*param.*vdd.*=).*/$1$vdd/i;
s/(.*param\s+t\s*=).*/$1$temp/i;
s/(.*param.*capVal.*=).*/$1$load/i;
s/(.*param.*slope.*=).*/$1$ramp/i;
s/(.*option.*tuning.*=).*/$1$tuning/i;
s/(.*param.*xStep.*=).*/$1$step/i;
s/(.*param.*xEnd.*=).*/$1$end/i;
s/(.*include.*)s8tssc_lf_dff\.nl/$1$nl/i;
s/(.*X1\s).*/$1$ports $subckt/i;
s/X1\.a/X1\.$a/i;
if($clk_flag == 1){
    s/\*?(.*vin1.*)/$1/i;
    s/(.*vin1.*X1.*)clk(.*)/$1$clk$2/i;
}else{
    s/\*?(.*vin1.*)/\*$1/i;
}
if($clk_n_flag == 1){
    s/\*?(.*vin3.*)/$1/i;
    s/(.*vin3.*X1.*)clk_n(.*)/$1$clk_n$2/i;
}else{
    s/\*?(.*vin3.*)/\*$1/i;
}
if($clk_flag == 1){
    s/(.*tpduu.*X1.)clk(.*)/$1$clk$2/i;
}elsif($clk_n_flag == 1){
    s/(.*tpd)u(.*X1.)clk(.*)/$1d$2$clk_n$3/i;
}else{
    die "Neither clk nor clk_n port is specified\n";
}
s/X1\.q/X1\.$q/i;
if($tm_flag == 1){
    if($tm_val eq 'vdd'){
        s/\*?(.*vvpwr3.*)/$1/i;
        s/(.*vvpwr3\s)smode_n(.*)/$1$tmod$2/i;
    }elsif($tm_val eq 'gnd'){
        s/\*?(.*vvgnd12.*)/$1/i;
        s/(.*vvgnd12\s)smode_n(.*)/$1$tmod$2/i;
    }else{
        die "Incorrect test mode port value\n";
    }
}
if($reset_flag == 1){
    if($reset_val eq 'vdd'){
        s/\*?(.*vvpwr4.*)/$1/i;
        s/(.*vvpwr4\s)resetb(.*)/$1$reset$2/i;
    }elsif($reset_val eq 'gnd'){
        s/\*?(.*vvgnd13.*)/$1/i;
        s/(.*vvgnd13\s)resetb(.*)/$1$reset$2/i;
    }else{
        die "Incorrect reset port value\n";
    }
}
if($set_flag == 1){
    if($set_val eq 'vdd'){
        s/\*?(.*vvpwr5.*)/$1/i;
        s/(.*vvpwr5\s)setb(.*)/$1$set$2/i;
```

```
    }elsif($set_val eq 'gnd'){
        s/\*?(.*vvgnd14.*)/$1/i;
        s/(.*vvgnd14\s)setb(.*)/$1$set$2/i;
    }else{
        die "Incorrect set port value\n";
    }
}
s/(.*param.*xStart.*=).*/$1$start/i;
s/(.*param.*xStop.*=).*/$1$stop/i;
s/(.*param.*xIncr.*=).*/$1$incr/i;
s/(.*tran.*delta\s).*/$1$start $stop -$incr/i;
if($modes eq 'single'){
    if($type eq 'setup'){
        s/\*?(.*vin2.*0 vdd.*10n).(delta.*)/$1-$2/i;
        s/\*?(.*vin2.*vdd 0.*)/*$1/i
    }elsif($type eq 'hold'){
        s/\*?(.*vin2.*vdd 0.*10n).(delta.*)/$1+$2/i;
        s/\*?(.*vin2.*0 vdd.*)/*$1/i
    }else{
        die "Incompatible sim type";
    }
    s/(.*param.*delta.*=).*/$1$delta/i;
    s/\*?(.*tran.*uic\z)/$1/i;
    s/\*?(.*tran.*delta.*)/\*$1/i;
    s/\*?(.*optimize)/\*$1/i;
    s/\*?(.*method.*)/\*$1/i;
    s/\*?(.*tol_rel.*)/\*$1/i;
    s/\*?(.*paramopt.*)/\*$1/i;
    s/\*?(.*goal.*)/\*$1/i;
}elsif($modes eq 'linear'){
    if($type eq 'setup'){
        s/\*?(.*vin2.*0 vdd.*10n).(delta.*)/$1-$2/i;
        s/\*?(.*vin2.*vdd 0.*)/*$1/i
    }elsif($type eq 'hold'){
        s/\*?(.*vin2.*vdd 0.*10n).(delta.*)/$1+$2/i;
        s/\*?(.*vin2.*0 vdd.*)/*$1/i
    }else{
        die "Incompatible sim type";
    }
    s/\*?(.*tran.*uic\z)/\*$1/i;
    s/\*?(.*tran.*delta.*)/$1/i;
    s/\*?(.*optimize)/\*$1/i;
    s/\*?(.*method.*)/\*$1/i;
    s/\*?(.*tol_rel.*)/\*$1/i;
    s/\*?(.*paramopt.*)/\*$1/i;
    s/\*?(.*goal.*)/\*$1/i;
}else{
    s/\*?(.*tran.*uic\z)/$1/i;
    s/\*?(.*tran.*delta.*)/\*$1/i;
    s/\*?(.*optimize)/$1/i;
    if($opmod eq 'passfail'){
        s/\*?(.*method.*=.*passfail)/$1/i;
        s/\*?(.*method.*=.*dichotomy)/\*$1/i;
        s/\*?(.*method.*=.*secant)/\*$1/i;
    }elsif($opmod eq 'dichotomy'){
        s/\*?(.*method.*=.*passfail)/\*$1/i;
        s/\*?(.*method.*=.*dichotomy)/$1/i;
        s/\*?(.*method.*=.*secant)/\*$1/i;
        s/\*?(.*goal.*=).*/$1$goal/i;
    }else{
        s/\*?(.*method.*=.*passfail)/\*$1/i;
        s/\*?(.*method.*=.*dichotomy)/\*$1/i;
        s/\*?(.*method.*=.*secant)/$1/i;
        s/\*?(.*goal.*=).*/$1$goal/i;
    }
    s/\*?(.*tol_relpar.*=).*/$1$tol/i;
    s/\*?(.*paramopt.*=).*/$1\($opstart,$lower,$upper\)/i;
}
```

```
        print OUTPUT "$_\n";
    }
    close OUTPUT;

    my $sim_cmd = "eldo $output_cir.cir -queue -noconf";
    print "$sim_cmd\n";
    system "mkdir -p WA/$input_cor/$counter/eldo";
    system "$sim_cmd > ./WA/$input_cor/$counter/$output_cir.log 2>&1";
    if($? != 0){
        exit $?;
    }

    open LOG, '<', "./WA/$input_cor/$counter/$output_cir.log" or die "Can't
open $output_cir.log: $!";
    my @line_log = <LOG>;
    close LOG;
    open CSV, '>>', "./WA/$input_cor/$counter/$output_cir.csv" or die "Can't
open $output_cir.csv: $!";
    my $nom_delay = 0;
    while(@line_log){
        $_ = shift @line_log;
        chomp;
        if($md_flag == 1 or $md_flag == 2){
            if(/.*value.*of.*parameter.*/i){
                s/.*value.*of.*parameter\s*\w*\s*is\s*(.*)/$1/i;
                print CSV "$_".",\t";
            }elsif(/.*clk2q.*=.*/i){
                s/.*clk2q.*=\s*(.*)\s+Sec/$1/i;
                #Works on cobb instead of wildcat
                if($nom_delay == 0){
                    $nom_delay = $_;
                }
                my $dd = $_ / $nom_delay;
                print CSV "$_,\t$dd\n";
            }elsif(/.*clk2q cannot.*/i){
                print CSV "Can't be measured,\tCan't be measured\n";
            }
        }else{
            if(/.*\*\*\* OPTIMIZATION \*\*\*.*/i){
                until(/.*\*\*\*>MESSAGE SUMMARY.*/i){
                    #unless(/\A\s+\z/i or /\A\n\z/i){
                        print CSV "$_\n";
                    #}
                    $_ = shift @line_log;
                    chomp;
                }
            }
        }
    }
    system "mv $output_cir*.* ./WA/$input_cor/$counter/eldo";
}
close CSV;
exit 0;

sub eldoTemplate{
    my $template = "* DFF Sim

.option brief probe
.notrc
*.option strict
.option nomod
.option printlg=10000
.option compat
.option post=1
.option ingold=1
.option numdgt=10
.option gmin=1.0e-18
.option gmindc=1.0e-18
```

```
.option nojwdb
.option tuning=accurate
.option interp=1

*.include /tools/cadflow/t/4.4/s8p-5r/models/ff.cor
.include /tools/cadflow/t/4.4/s8p-5r/models/tt.cor
*.include /tools/cadflow/t/4.4/s8p-5r/models/ss.cor

*.include /tools/cadflow/t/4.4/s8p-5r/models/hrlc.cor
.include /tools/cadflow/t/4.4/s8p-5r/models/trtc.cor
*.include /tools/cadflow/t/4.4/s8p-5r/models/lrhc.cor

*.include /tools/cadflow/t/4.4/s8p-5r/models/ffcell.cor
.include /tools/cadflow/t/4.4/s8p-5r/models/ttcell.cor
*.include /tools/cadflow/t/4.4/s8p-5r/models/sscell.cor

.include ./s8tssc_lf_dff.nl

X1 vgnd vpwr vnb clk q q_n vpb a s8tssc_lf_dff

.param capVal=5f
.param t=-40
.param delta=2n
.param vdd=1.35
.param period=6n
.param slope=0.5n
.param xStep=0.001n
.param xEnd=20n
.param xStart=2n
.param xStop=0.1n
.param xIncr=0.1n
.temp t

cL0 q       0 capVal
cL1 q_n     0 capVal

vvpwr1  vpb     0 vdd
vvpwr2  vpwr        0 vdd
*vvpwr3 smode_n     0 vdd
*vvpwr4 resetb      0 vdd
*vvpwr5 setb        0 vdd

vvgnd10     vgnd    0 0
vvgnd11     vnb 0 0
*vvgnd12    smode_n 0 0
*vvgnd13    resetb  0 0
*vvgnd14    setb    0 0

vin1    X1.clk 0    pulse(0 vdd 10n slope slope 'period-slope' '2*period')
vin2    X1.a 0      pulse(0 vdd '10n-delta' slope slope 100n 200n)
*vin2   X1.a 0      pulse(vdd 0 '10n-delta' slope slope 100n 200n)
*vin3   X1.clk_n 0  pulse(vdd 0 10n slope slope 'period-slope' '2*period')

.plot tran
+v(X1.clk)
+v(X1.a)
+v(X1.q)
+v(X1.q_n)
+v(X1.true)
+v(X1.bar)

+isub(X1.vpwr)
+isub(X1.vpb)

+power
+ix(X1.7)

.tran xStep xEnd uic
```

```
*.tran xStep xEnd uic sweep delta 0.1n 2n 0.1n

*.optimize
*+method=passfail
*+method=dichotomy
*+method=secant

*+tol_relpar=0.1
*+tol_reltarg=0.01

*.paramopt delta=(1n,0.1n,2n)

.extract
+tran
+label=clk2q
+tpduu(v(X1.clk),v(X1.q),vth='0.5*vdd',after=0)
*+goal=0.215n

.end";

    $template;
}

sub syntax{
    print "extract.pl <corner file>\n";
    print "corner file structure\n";
    print "|_Process, <proc>\n";
    print "|_Voltage, <vdd>\n";
    print "|_Temp, <temp>\n";
    print "|_Cap load, <load>\n";
    print "|_Ramp rate, <ramp>\n";
    print "|_Tuning factor, <tuning>\n";
    print "|_Sim stepwise, <step>\n";
    print "|_Sim length, <end>\n";
    print "|_netlist, <nl>\n";
    print "|_Input port, <input>\n";
    print "|_Clk port, <clk>\n";
    print "|_Clk_n port, <clk_n>\n";
    print "|_Output port, <output>\n";
    print "|_Test mode port, <tmod>\n";
    print "|     |_Test mode port value, <tmod_val>\n";
    print "|_Reset port, <reset>\n";
    print "|     |_Reset port value, <reset_val>\n";
    print "|_Set port, <set>\n";
    print "|     |_Set port value, <set_val>\n";
    print "|_Sim mode, <mode>\n";
    print "      |_Single sim\n";
    print "      |    |_Sim type, <type>\n";
    print "      |    |_Single sim delta, <delta>\n";
    print "      |_Linear sweep\n";
    print "      |    |_Sim type, <type>\n";
    print "      |    |_Linear sweep start, <start>\n";
    print "      |    |_Linear sweep stop, <stop>\n";
    print "      |    |_Linear sweep increment, <incr>\n";
    print "      |_Optimization\n";
    print "           |_Optimization mode, <opmod>\n";
    print "           |_Optimization tolerance, <tol>\n";
    print "           |_Optimization start point, <opstart>\n";
    print "           |_Lower band, <lower>\n";
    print "           |_upper band, <upper>\n";
    print "           |_Dichotomy/secant\n";
    print "                     |_Delay degradation percentage, <perc>\n";
    print "                     |_Nominal delay, <nom>\n";
    print "Example:\n";
    print "proc=tt; vdd=1.8; temp=25; load=5f; ramp=0.5n; tuning=accurate;
step=0.01n; end=12n; netlist=s8tssc_lf_dff.nl; input=a; clk=clk; output=q;
mode=optimization; opmod=dichotomy; tol=0.05; opstart=1n; lower=0.1n;
upper=2n; perc=1.05; nom=0.1190n\n";
```

```
    die "Eldo sim abort";
}
```

Appendix B: Generic Ruby Script for Individual DFF Simulation

```
#!/usr/local/bin/ruby
# Developer: Xiaowei Zhang

def main
    syntax if ARGV.length == 0
    cor_name = ARGV[0]
    counter = 0

    cor_file = File.open(cor_name, "r")
    until cor_file.eof do
        line_cor = cor_file.gets.strip.chomp
        counter += 1

        proc = "tt"
        vdd = "1.8"
        temp = "25"
        load = "5f"
        ramp = "0.5n"
        tuning = "accurate"
        step = "0.001n"
        sim_end = "20n"
        nl = Attr.new("s8tssc_lf_dff.nl", 0)
        subckt = "s8tssc_lf_dff"
        ports = "vgnd vpwr vnb clk q q_n vpb a"
        a = "d"
        clk = Port.new("clk", 0)
        clk_n = Port.new("clk_n", 0)
        q = "q"
        tmod = Port.new("test", 0, "vdd")
        reset = Port.new("reset", 0, "vdd")
        set = Port.new("set", 0, "vdd")
        start = "2n"
        stop = "0.1n"
        incr = "0.1n"
        modes = Attr.new("linear", 0)
        type = "setup"
        # Need more work to distinguish master-slave & pre-charge type FFs
        # Pre-charge type FFs need to have two output q & q_cmp
        # polarity = "rising"
        delta = "2n"
        opmod = "passfail"
        tol = 0.01
        opstart = "1n"
        lower = "0.1n"
        upper = "2n"
        perc = 1.05
        nom = "0.12n"
        goal = "0.13n"
        output_cir = "dff.cir"

        # Never use match block, use "if" instead
        if /\A(\*|\#).*/i.match(line_cor)
            next
        end

        line_cor.gsub!(/\s+/i, "")
        param_pair = line_cor.split(/;/)
        param_pair.each do |param|
            #param_hash = Hash.new
            (key, val) = param.split(/=/)

            case key
                when "proc"
                    proc = val
                    puts "Process: #{proc}"
```

```
when "vdd"
    vdd = val
    puts "Vdd = #{vdd}"
when "temp"
    temp = val
    puts "Temp = #{temp}"
when "load"
    load = val
    puts "Cap load = #{load}"
when "ramp"
    ramp = val
    puts "Slew rate = #{ramp}"
when "tuning"
    tuning = val
    puts "Tuning factor: #{tuning}"
when "step"
    step = val
    puts "Sim stepwise = #{step}"
when "end"
    sim_end = val
    puts "Sim length = #{sim_end}"
when "netlist"
    nl.name = val
    puts "Netlist file: #{nl.name}"
when "input"
    a = val
when "clk"
    clk.name = val
    clk.flag = 1
when "clk_n"
    clk_n.name = val
    clk_n.flag = 1
when "output"
    q = val
when "tmod"
    tmod.name = val
    tmod.flag = 1
when "tmod_val"
    tmod.val = val
when "reset"
    reset.name = val
    reset.flag = 1
when "reset_val"
    reset.val = val
when "set"
    set.name = val
    set.flag = 1
when "set_val"
    set.val = val
when "mode"
    modes.name = val
    puts "Sim mode: #{modes.name}"
when "type"
    type = val
    puts "Sim Type: #{type}"
# when "polarity"
    # polarity = val
    # puts "Input polarity: #{polarity}"
when "delta"
    delta = val
    puts "Delay from clk to output = #{delta}"
when "start"
    start = val
    puts "Linear sweep start = #{start}"
when "stop"
    stop = val
    puts "Linear sweep stop = #{stop}"
when "incr"
```

```
                incr = val
                puts "Linear sweep increment = #{incr}"
            when "opmod"
                opmod = val
                puts "Optimization mode: #{opmod}"
            when "tol"
                tol = val
                puts "Optimization tolerance = #{tol}"
            when "opstart"
                opstart = val
                puts "Optimization start = #{opstart}"
            when "lower"
                lower = val
                puts "Optimization lower band = #{lower}"
            when "upper"
                upper = val
                puts "Optimization upper band = #{upper}"
            when "perc"
                perc = val
                puts "Delay degradation percentage = #{perc}"
            when "nom"
                nom = val
                puts "Nominal delay = #{nom}"
            else
                puts "Incompatible param"
                exit
        end
    end

    nl_file = File.open(nl.name, "r")
    until nl_file.eof do
        line_nl = nl_file.gets.strip.chomp
        if /.*subckt\s*(\w*)\s*(.*)/i.match(line_nl)
            nl.flag = 1
            subckt = $1
            ports = $2
            line_nl = nl_file.gets.strip.chomp
            while /\A\+(.*)/i.match(line_nl) do
                ports = ports + " " + $1
                line_nl = nl_file.gets.strip.chomp
            end
        end
    end
    if nl.flag == 0
        puts "Can't find subckt in netlist #{nl.name}"
        exit
    end

    # "or" or "|" doesn't work well in Ruby unlike in Perl, use
Regexp.union instead
    if Regexp.union(/\A#{a}\s.*/i, /.*\s#{a}\s.*/i,
/.*\s#{a}\z/i).match(ports)
        puts "Input port name: #{a}"
    else
        puts "Can't find input port #{a} in netlist #{nl.name}"
        exit
    end
    if clk.flag == 1
        if Regexp.union(/\A#{clk.name}\s.*/i, /.*\s#{clk.name}\s.*/i,
/.*\s#{clk.name}\z/i).match(ports)
            puts "Clk port name: #{clk.name}"
        else
            puts "Can't find clk port #{clk.name} in netlist #{nl.name}"
            exit
        end
    end
    if clk_n.flag == 1
```

```
            if Regexp.union(/\A#{clk_n.name}\s.*/i,
/.*\s#{clk_n.name}\s.*/i, /.*\s#{clk_n.name}\z/i).match(ports)
                puts "Clk port name: #{clk_n.name}"
            else
                puts "Can't find clk port #{clk_n.name} in netlist
#{nl.name}"
                exit
            end
        end
        if clk.flag == 0 && clk_n.flag == 0
            puts "Can't find clk port in netlist #{nl.name}"
        end
        if Regexp.union(/\A#[6]\s.*/i, /.*\s#[6]\s.*/i,
/.*\s#[6]\z/i).match(ports)
            puts "Input port name: #[6]"
        else
            puts "Can't find input port #[6] in netlist #{nl.name}"
            exit
        end
        if tmod.flag == 1
            if Regexp.union(/\A#{tmod.name}\s.*/i, /.*\s#{tmod.name}\s.*/i,
/.*\s#{tmod.name}\z/i).match(ports)
                puts "Clk port name: #{tmod.name}"
            else
                puts "Can't find test port #{tmod.name} in netlist
#{nl.name}"
                exit
            end
        end
        if reset.flag == 1
            if Regexp.union(/\A#{reset.name}\s.*/i,
/.*\s#{reset.name}\s.*/i, /.*\s#{reset.name}\z/i).match(ports)
                puts "Clk port name: #{reset.name}"
            else
                puts "Can't find reset port #{reset.name} in netlist
#{nl.name}"
                exit
            end
        end
        if set.flag == 1
            if Regexp.union(/\A#{set.name}\s.*/i, /.*\s#{set.name}\s.*/i,
/.*\s#{set.name}\z/i).match(ports)
                puts "Clk port name: #{set.name}"
            else
                puts "Can't find set port #{set.name} in netlist #{nl.name}"
                exit
            end
        end

        case modes.name
            when "single"
                output_cir = proc + "_" + vdd + "_" + temp + "_" + subckt +
"_" + modes.name + "_" + type + "_" + delta
                modes.flag = 1
            when "linear"
                output_cir = proc + "_" + vdd + "_" + temp + "_" + subckt +
"_" + modes.name + "_" + type + "_" + start + "_" + stop + "_" + incr
                modes.flag = 2
            when "optimization"
                if opmod == dichotomy
                    output_cir = proc + "_" + vdd + "_" + temp + "_" +
subckt + "_" + modes.name + "_" + opmod + "_" + tol + "_" + opstart + "_" +
lower + "_" + upper + "_" + perc + "_" + nom
                    modes.flag = 3
                elsif opmode == passfail
                    output_cir = proc + "_" + vdd + "_" + temp + "_" +
subckt + "_" + modes.name + "_" + opmod + "_" + tol + "_" + lower + "_" +
upper
```

```
                modes.flag = 4
            else
                puts "Incompatible optimization mode"
                exit
            end
        else
            puts "Incompatible sim mode"
            exit
    end

    # Use \1 instead of $1, $1 can only be used in block for back-
reference
    # Use '' instead of ""
    # If it is a double-quoted string, both back-references must be
preceded by an additional backslash.
    # However, within replacement the special match variables, such as
&$, will not refer to the current match.
    nom.sub!(/(.*)n/i, '\1')
    goal = perc * nom.to_f
    goal = goal.to_s + "n"

    line_cir = EldoTemplate::TEMPLATE.split(/\n/)
    output = File.open("#{output_cir}.cir", "w")

    line_cir.each do |line|
        line.chomp
        case proc
            when "ff"
                line.sub!(/\*?(.*include.*ff.*.cor)/i, '\1')
                line.sub!(/\*?(.*include.*tt.*.cor)/i, '*' + '\1')
                line.sub!(/\*?(.*include.*ss.*.cor)/i, '*' + '\1')
            when "tt"
                line.sub!(/\*?(.*include.*ff.*.cor)/i, '*' + '\1')
                line.sub!(/\*?(.*include.*tt.*.cor)/i, '\1')
                line.sub!(/\*?(.*include.*ss.*.cor)/i, '*' + '\1')
            when "ss"
                line.sub!(/\*?(.*include.*ff.*.cor)/i, '*' + '\1')
                line.sub!(/\*?(.*include.*tt.*.cor)/i, '*' + '\1')
                line.sub!(/\*?(.*include.*ss.*.cor)/i, '\1')
            else
                puts "Incompatible process"
                exit
        end
        line.sub!(/(.*param.*vdd.*=).*/i, '\1' + vdd)
        line.sub!(/(.*param\s+t\s*=).*/i, '\1' + temp)
        line.sub!(/(.*param.*capVal.*=).*/i, '\1' + load)
        line.sub!(/(.*param.*slope.*=).*/i, '\1' + ramp)
        line.sub!(/(.*option.*tuning.*=).*/i, '\1' + tuning)
        line.sub!(/(.*param.*xStep.*=).*/i, '\1' + step)
        line.sub!(/(.*param.*xEnd.*=).*/i, '\1' + sim_end)
        line.sub!(/(.*include.*)s8tssc_lf_dff\.nl/i, '\1' + nl.name)
        line.sub!(/(.*X1\s).*/i, '\1' + ports + ' ' + subckt)
        line.sub!(/X1\.a/i, "X1." + a)
        if clk.flag == 1
            line.sub!(/\*?(.*vin1.*)/i, '\1')
            line.sub!(/(.*vin1.*X1.*)clk(.*)/i, '\1' + clk.name + '\2')
        else
            line.sub!(/\*?(.*vin1.*)/i, '*' + '\1')
        end
        if clk_n.flag == 1
            line.sub!(/\*?(.*vin3.*)/i, '\1')
            line.sub!(/(.*vin3.*X1.*)clk_n(.*)/i, '\1' + clk_n.name +
'\2')
        else
            line.sub!(/\*?(.*vin3.*)/i, '*' + '\1')
        end
        # If both clk and clk_n are present, use clk in "extract"
        if clk.flag == 1
```

```
                line.sub!(/(.*tpduu.*X1.)clk(.*)/i, '\1' + clk.name + '\2')
            elsif clk_n.flag == 1
                line.sub!(/(.*tpd)u(.*X1.)clk(.*)/i, '\1' + 'd' + '\2' +
clk_n.name + '\3')
            else
                puts "Neither clk nor clk_n port is specified"
                exit
            end
            line.sub!(/X1\.q/i, "X1." + q)
            if tmod.flag == 1
                if tmod.val == "vdd"
                    line.sub!(/\*?(.*vvpwr3\s)smode_n(.*)/i, '\1' +
tmod.name + '\2')
                elsif tmod.val == "gnd"
                    line.sub!(/\*?(.*vvgnd12\s)smode_n(.*)/i, '\1' +
tmod.name + '\2')
                else
                    puts "Incorrect test mode port value"
                    exit
                end
            else
                line.sub!(/\*?(.*vvpwr3.*)/i, '*' + '\1')
                line.sub!(/\*?(.*vvgnd12.*)/i, '*' + '\1')
            end
            if reset.flag == 1
                if reset.val == "vdd"
                    line.sub!(/\*?(.*vvpwr4\s)resetb(.*)/i, '\1' +
reset.name + '\2')
                elsif reset.val == "gnd"
                    line.sub!(/\*?(.*vvgnd13\s)resetb(.*)/i, '\1' +
reset.name + '\2')
                else
                    puts "Incorrect reset mode port value"
                    exit
                end
            else
                line.sub!(/\*?(.*vvpwr4.*)/i, '*' + '\1')
                line.sub!(/\*?(.*vvgnd13.*)/i, '*' + '\1')
            end
            if set.flag == 1
                if set.val == "vdd"
                    line.sub!(/\*?(.*vvpwr5\s)setb(.*)/i, '\1' + set.name +
'\2')

                elsif set.val == "gnd"
                    line.sub!(/\*?(.*vvgnd14\s)setb(.*)/i, '\1' + set.name +
'\2')

                else
                    puts "Incorrect set mode port value"
                    exit
                end
            else
                line.sub!(/\*?(.*vvpwr5.*)/i, '*' + '\1')
                line.sub!(/\*?(.*vvgnd14.*)/i, '*' + '\1')
            end
            line.sub!(/(.*param.*xStart.*=).*/i, '\1' + start)
            line.sub!(/(.*param.*xStop.*=).*/i, '\1' + stop)
            line.sub!(/(.*param.*xIncr.*=).*/i, '\1' + incr)
            line.sub!(/(.*tran.*delta\s).*/i, '\1' + start + ' ' + stop + '
-' + incr)
            case modes.name
                when "single"
                    # Add sim timing param type & polarity
                    if type == "setup"
                        line.sub!(/\*?(.*vin2.*0 vdd.*10n).(delta.*)/i, '\1'
+ '-' + '\2')
                        line.sub!(/\*?(.*vin2.*vdd 0.*)/i, '*' + '\1')
                        # line.sub!(/\*?(.*tpd.*clk.*q)/i, '\1')
                        # line.sub!(/\*?(.*tpd.*q.*clk)/i, '*' + '\1')
```

```
                    elsif type == "hold"
                        line.sub!(/\*?(.*vin2.*vdd 0.*10n).(delta.*)/i, '\1'
+ '+' + '\2')

                        line.sub!(/\*?(.*vin2.*0 vdd.*)/i, '*' + '\1')
                    else
                        puts "Incompatible sim type"
                        exit
                    end
                    # if polarity == "rising"
                        # line.sub!(/\*?(.*vin2.*0 vdd.*)/i, '\1')
                        # line.sub!(/\*?(.*vin2.*vdd 0.*)/i, '*' + '\1')
                        # line.sub!(/(.*tpd)..(.*clk.*q)/i, '\1' + 'uu' +
'\2')

                        # line.sub!(/(.*tpd)..(.*q.*clk)/i, '\1' + 'uu' +
'\2')

                    # elsif polarity == "falling"
                        # line.sub!(/\*?(.*vin2.*0 vdd.*)/i, '*' + '\1')
                        # line.sub!(/\*?(.*vin2.*vdd 0.*)/i, '\1')
                        # line.sub!(/(.*tpd)..(.*clk.*q)/i, '\1' + 'ud' +
'\2')

                        # line.sub!(/(.*tpd)..(.*q.*clk)/i, '\1' + 'du' +
'\2')

                    # else
                        # puts "Incompatible input polarity"
                        # exit
                    # end
                    line.sub!(/(.*param.*delta.*=).*/i, '\1' + delta)
                    line.sub!(/\*?(.*tran.*uic\z)/i, '\1')
                    line.sub!(/\*?(.*tran.*delta.*)/i, '*' + '\1')
                    line.sub!(/\*?(.*optimize)/i, '*' + '\1')
                    line.sub!(/\*?(.*method.*)/i, '*' + '\1')
                    line.sub!(/\*?(.*tol_rel.*)/i, '*' + '\1')
                    line.sub!(/\*?(.*paramopt.*)/i, '*' + '\1')
                    line.sub!(/\*?(.*goal.*)/i, '*' + '\1')
                when "linear"
                    if type == "setup"
                        line.sub!(/\*?(.*vin2.*0 vdd.*10n).(delta.*)/i, '\1'
+ '-' + '\2')

                        line.sub!(/\*?(.*vin2.*vdd 0.*)/i, '*' + '\1')
                        # line.sub!(/\*?(.*tpd.*clk.*q)/i, '\1')
                        # line.sub!(/\*?(.*tpd.*q.*clk)/i, '*' + '\1')
                    elsif type == "hold"
                        line.sub!(/\*?(.*vin2.*vdd 0.*10n).(delta.*)/i, '\1'
+ '+' + '\2')

                        line.sub!(/\*?(.*vin2.*0 vdd.*)/i, '*' + '\1')
                    else
                        puts "Incompatible sim type"
                        exit
                    end
                    # if polarity == "rising"
                        # line.sub!(/\*?(.*vin2.*0 vdd.*)/i, '\1')
                        # line.sub!(/\*?(.*vin2.*vdd 0.*)/i, '*' + '\1')
                        # line.sub!(/(.*tpd)..(.*clk.*q)/i, '\1' + 'uu' +
'\2')

                        # line.sub!(/(.*tpd)..(.*q.*clk)/i, '\1' + 'uu' +
'\2')

                    # elsif polarity == "falling"
                        # line.sub!(/\*?(.*vin2.*0 vdd.*)/i, '*' + '\1')
                        # line.sub!(/\*?(.*vin2.*vdd 0.*)/i, '\1')
                        # line.sub!(/(.*tpd)..(.*clk.*q)/i, '\1' + 'ud' +
'\2')

                        # line.sub!(/(.*tpd)..(.*q.*clk)/i, '\1' + 'du' +
'\2')

                    # else
                        # puts "Incompatible input polarity"
                        # exit
                    # end
                    line.sub!(/\*?(.*tran.*uic\z)/i, '*' + '\1')
```

```
                    line.sub!(/\*?(.*tran.*delta.*)/i, '\1')
                    line.sub!(/\*?(.*optimize)/i, '*' + '\1')
                    line.sub!(/\*?(.*method.*)/i, '*' + '\1')
                    line.sub!(/\*?(.*tol_rel.*)/i, '*' + '\1')
                    line.sub!(/\*?(.*paramopt.*)/i, '*' + '\1')
                    line.sub!(/\*?(.*goal.*)/i, '*' + '\1')
                when "optimization"
                    line.sub!(/\*?(.*tran.*uic\z)/i, '\1')
                    line.sub!(/\*?(.*tran.*delta.*)/i, '*' + '\1')
                    line.sub!(/\*?(.*optimize)/i, '\1')
                    case opmod
                        when "passfail"
                            line.sub!(/\*?(.*method.*=.*passfail)/i, '\1')
                            line.sub!(/\*?(.*method.*=.*dichotomy)/i, '*' +
'\1')

                            line.sub!(/\*?(.*method.*=.*secant)/i, '*' +
'\1')

                            line.sub!(/\*?(.*goal.*=.*)/i, '*' + '\1')
                        when "dichotomy"
                            line.sub!(/\*?(.*method.*=.*passfail)/i, '*' +
'\1')

                            line.sub!(/\*?(.*method.*=.*dichotomy)/i, '\1')
                            line.sub!(/\*?(.*method.*=.*secant)/i, '*' +
'\1')

                            line.sub!(/\*?(.*goal.*=).*/i, '\1' + goal)
                        when "secant"
                            line.sub!(/\*?(.*method.*=.*passfail)/i, '*' +
'\1')

                            line.sub!(/\*?(.*method.*=.*dichotomy)/i, '*' +
'\1')

                            line.sub!(/\*?(.*method.*=.*secant)/i, '\1')
                            line.sub!(/\*?(.*goal.*=).*/i, '\1' + goal)
                        else
                            puts "Incompatible optimization mode"
                            exit
                    end
                    line.sub!(/\*?(.*tol_relpar.*=).*/i, '\1' + tol)
                    line.sub!(/\*?(.*paramopt.*=).*/i, '\1' + '(' + opstart
+ ', ' + lower + ', ' + upper + ')')
                else
                    puts "Incompatible sim mode"
                    exit
            end
            output.printf("#{line}\n")
        end
        output.close

        sim_cmd = "eldo #{output_cir}.cir -queue -noconf"
        puts sim_cmd
        system("mkdir -p WA/#{cor_name}/#{counter}/eldo")
        system("#{sim_cmd} > ./WA/#{cor_name}/#{counter}/#{output_cir}.log
2>&1")

        log = File.open("./WA/#{cor_name}/#{counter}/#{output_cir}.log",
"r")

        csv = File.open("./WA/#{cor_name}/#{counter}/#{output_cir}.csv",
"w")

        nom_delay = 0
        until log.eof do
            line_log = log.gets.strip.chomp
            if modes.flag == 1 || modes.flag == 2
                if /.*value.*of.*parameter.*/i.match(line_log)
                    if
line_log.sub!(/.*value.*of.*parameter\s*\w*\s*is\s*(.*)/i, '\1')
                        csv.printf("#{$1},\t")
                    end
                elsif /.*clk2q.*=.*/i.match(line_log)
                    if line_log.sub!(/.*clk2q.*=\s*(.*)\s+Sec/i, '\1')
```

```
                        nom_delay = $1.to_f if nom_delay == 0
                        dd = $1.to_f / nom_delay
                        csv.printf("#{$1},\t#{dd}\n")
                    end
                elsif /.*clk2q cannot.*/i.match(line_log)
                    csv.printf("Can't be measured,\tCan't be measured\n")
                end
            else
                if /.*\*\*\* OPTIMIZATION \*\*\*.*/i.match(line_log)
                    until /.*\*\*\*>MESSAGE SUMMARY.*/i.match(line_log)
                        csv.printf("#{line_log}\n")
                        line_log = log.gets.strip.chomp
                    end
                end
            end
        end
        system("mv #{output_cir}*.* ./WA/#{cor_name}/#{counter}/eldo")
        log.close
        csv.close
    end
end

class Attr
    attr_accessor :name
    attr_accessor :flag

    def initialize(name, flag = 0)
        @name = name
        @flag = flag
    end
end

class Port
    attr_accessor :name
    attr_accessor :flag
    attr_accessor :val

    def initialize(name, flag = 0, val = "vdd")
        @name = name
        @flag = flag
        @val = val
    end
end

module EldoTemplate
    TEMPLATE = "* DFF Sim

.option brief probe
.notrc
*.option strict
.option nomod
.option printlg=10000
.option compat
.option post=1
.option ingold=1
.option numdgt=10
.option gmin=1.0e-18
.option gmindc=1.0e-18
.option nojwdb
.option tuning=accurate
.option interp=1

*.include /tools/cadflow/t/4.4/s8p-5r/models/ff.cor
.include /tools/cadflow/t/4.4/s8p-5r/models/tt.cor
*.include /tools/cadflow/t/4.4/s8p-5r/models/ss.cor

*.include /tools/cadflow/t/4.4/s8p-5r/models/hrlc.cor
.include /tools/cadflow/t/4.4/s8p-5r/models/trtc.cor
```

```
*.include /tools/cadflow/t/4.4/s8p-5r/models/lrhc.cor

*.include /tools/cadflow/t/4.4/s8p-5r/models/ffcell.cor
.include /tools/cadflow/t/4.4/s8p-5r/models/ttcell.cor
*.include /tools/cadflow/t/4.4/s8p-5r/models/sscell.cor

.include ./s8tssc_lf_dff.nl

X1 vgnd vpwr vnb clk q q_n vpb a s8tssc_lf_dff

.param capVal=5f
.param t=-40
.param delta=2n
.param vdd=1.35
.param period=6n
.param slope=0.5n
.param xStep=0.001n
.param xEnd=20n
.param xStart=2n
.param xStop=0.1n
.param xIncr=0.1n
.temp t

cL0 q       0 capVal
cL1 q_n     0 capVal

vvpwr1  vpb     0 vdd
vvpwr2  vpwr        0 vdd
*vvpwr3 smode_n     0 vdd
*vvpwr4 resetb      0 vdd
*vvpwr5 setb        0 vdd

vvgnd10     vgnd    0 0
vvgnd11     vnb 0 0
*vvgnd12    smode_n 0 0
*vvgnd13    resetb  0 0
*vvgnd14    setb    0 0

vin1    X1.clk 0    pulse(0 vdd 10n slope slope 'period-slope' '2*period')
vin2    X1.a 0      pulse(0 vdd '10n-delta' slope slope 100n 200n)
*vin2   X1.a 0      pulse(vdd 0 '10n-delta' slope slope 100n 200n)
*vin3   X1.clk_n 0  pulse(vdd 0 10n slope slope 'period-slope' '2*period')

.plot tran
+v(X1.clk)
+v(X1.a)
+v(X1.q)
+v(X1.q_n)
+v(X1.true)
+v(X1.bar)

+isub(X1.vpwr)
+isub(X1.vpb)

+power
+ix(X1.7)

.tran xStep xEnd uic
*.tran xStep xEnd uic sweep delta 0.1n 2n 0.1n

*.optimize
*+method=passfail
*+method=dichotomy
*+method=secant

*+tol_relpar=0.1
*+tol_reltarg=0.01
```

```
*.paramopt delta=(1n,0.1n,2n)

.extract
+tran
+label=clk2q
+tpduu(v(X1.clk),v(X1.q),vth='0.5*vdd',after=0)
*+goal=0.215n

.end";
end

def syntax
    puts "extract.rb <corner file>"
    puts "corner file structure:"
    puts "|_Process, <proc>"
    puts "|_Voltage, <vdd>"
    puts "|_Temp, <temp>"
    puts "|_Cap load, <load>"
    puts "|_Ramp rate, <ramp>"
    puts "|_Tuning factor, <tuning>"
    puts "|_Sim stepwise, <step>"
    puts "|_Sim length, <end>"
    puts "|_netlist, <nl>"
    puts "|_Input port, <input>"
    puts "|_Clk port, <clk>"
    puts "|_Clk_n port, <clk_n>"
    puts "|_Output port, <output>"
    puts "|_Test mode port, <tmod>"
    puts "| |_Test mode port value, <tmod_val>"
    puts "|_Reset port, <reset>"
    puts "| |_Reset port value, <reset_val>"
    puts "|_Set port, <set>"
    puts "| |_Set port value, <set_val>"
    puts "|_Sim mode, <mode>"
    puts "      |_Single sim"
    puts "      |   |_Sim type, <type>"
    puts "      |   |_Single sim delta, <delta>"
    puts "      |_Linear sweep"
    puts "      |   |_Sim type, <type>"
    puts "      |   |_Linear sweep start, <start>"
    puts "      |   |_Linear sweep stop, <stop>"
    puts "      |   |_Linear sweep increment, <incr>"
    puts "      |_Optimization"
    puts "          |_Optimization mode, <opmod>"
    puts "          |_Optimization tolerance, <tol>"
    puts "          |_Optimization start point, <opstart>"
    puts "          |_Lower band, <lower>"
    puts "          |_upper band, <upper>"
    puts "          |_Dichotomy/secant"
    puts "                  |_Delay degradation percentage, <perc>"
    puts "                  |_Nominal delay, <nom>"
    puts "Example:\n";
    puts "proc=tt; vdd=1.8; temp=25; load=5f; ramp=0.5n; tuning=accurate;
step=0.01n; end=12n; netlist=s8tssc_lf_dff.nl; input=a; clk=clk; output=q;
mode=optimization; opmod=dichotomy; tol=0.05; opstart=1n; lower=0.1n;
upper=2n; perc=1.05; nom=0.1190n"
    puts "Eldo sim abort"
    exit
end

main
```

References

[1] J. Wawrzynek, A. Krste, and J. Lazzaro, "UC Berkely CS250 VLSI Systems Design Lecture 8: Memory," ed.

[2] "University of Washington CSE471: Main Memory," ed.

[3] IDP. *What is Pipeline Burst SRAM?* Available: https://www.idp.net/sysinfo/motherboards.asp#5

[4] M. Barr, "Memory types," *Embedded Systems Programming,* vol. 14, pp. 103-104, 2001.

[5] I. S. S. Inc. *Selecting the Right ISSI Industrial Grade Memory*. Available: http://www.issi.com/WW/pdf/Ind-Memory.pdf

[6] B. L. Star Sung, Jacques Baudier (TITC). *Display Driver with on-chip frame buffer and a scalable image compression engine*. Available: http://www.design-reuse.com/articles/33274/display-driver-with-on-chip-frame-buffer-and-a-scalable-image-compression-engine.html

[7] Q. Chen, H. Mahmoodi, S. Bhunia, and K. Roy, "Efficient testing of SRAM with optimized march sequences and a novel DFT technique for emerging failures due to process variations," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 13, pp. 1286-1295, 2005.

[8] Wikipedia. *Static random-access memory*. Available: http://en.wikipedia.org/wiki/Static_random-access_memory

[9] J. P. Kulkarni, K. Kim, and K. Roy, "A 160 mV robust schmitt trigger based subthreshold SRAM," *Solid-State Circuits, IEEE Journal of,* vol. 42, pp. 2303-2313, 2007.

[10] R. P. Preston, "14 REGISTER FILES AND CACHES," *Design of High-Performance Microprocessor Circuits,* p. 285, 2001.

[11] H. Sunami, *Dimension Increase in Metal-Oxide-Semiconductor Memories and Transistors*: INTECH Open Access Publisher, 2010.

[12] S. Narenda, L. Fujino, and K. Smith, "Through the Looking Glass Continued (III): Update to Trends in Solid-State Circuits and Systems from ISSCC 2014 [ISSCC Trends]," *Solid-State Circuits Magazine, IEEE,* vol. 6, pp. 49-53, 2014.

[13] N. Shibata, H. Morimura, and M. Harada, "1-V 100-MHz embedded SRAM techniques for battery-operated MTCMOS/SIMOX ASICs," *Solid-State Circuits, IEEE Journal of,* vol. 35, pp. 1396-1407, 2000.

[14] B.-D. Yang and L.-S. Kim, "A low-power SRAM using hierarchical bit line and local sense amplifiers," *Solid-State Circuits, IEEE Journal of,* vol. 40, pp. 1366-1376, 2005.

[15] A. Kawasumi, A. Suzuki, H. Hatada, Y. Takeyama, O. Hirabayashi, Y. Kameda*, et al.*, "a 1.8 v 18 mb ddr cmos sram with power reduction techniques," in *VLSI Circuits, 2000. Digest of Technical Papers. 2000 Symposium on*, 2000, pp. 72-73.

[16] J. Jang, H. Kim, H. Baek, J. Na, K. Lee, D. Seo*, et al.*, "A 2.05 um/sup 2/full CMOS ultra-low power SRAM cell with 0.15 nm generation single gate CMOS technology," in *Electron Devices Meeting, 2000. IEDM'00. Technical Digest. International*, 2000, pp. 579-582.

[17] S. Masuoka, K. Noda, S. Ito, K. Matsui, K. Imai, N. Yasuzato*, et al.*, "A 0.99-/spl mu/m/sup 2/loadless four-transistor SRAM cell in 0.13-/spl mu/m generation CMOS technology," in *VLSI Technology, 2000. Digest of Technical Papers. 2000 Symposium on*, 2000, pp. 164-165.

[18] D. K. Nelson, H. Liu, K. Golke, and A. Kohli, "150nm SOI embedded SRAMs with very low SER," in *SOI Conference, 2005. Proceedings. 2005 IEEE International*, 2005, pp. 188-190.

[19] K. Cox, J. Scott, S. Bishop, M. Bhat, B. Nettleton, D. Pan*, et al.*, "A partially depleted 1.8 V SOI CMOS SRAM technology featuring a 3.77/spl mu/m/sup 2/cell," in *VLSI Technology, 2000. Digest of Technical Papers. 2000 Symposium on*, 2000, pp. 170-171.

[20] F. Ootsuka, S. Wakahara, K. Ichinose, A. Honzawa, S. Wada, H. Sato*, et al.*, "A highly dense, high-performance 130 nm node CMOS technology for large scale system-on-a-chip applications," in *Electron Devices Meeting, 2000. IEDM'00. Technical Digest. International*, 2000, pp. 575-578.

[21] W. Kong, R. Venkatraman, R. Castagnetti, F. Duan, and S. Ramesh, "High-density and high-performance 6T-SRAM for system-on-chip in 130 nm CMOS technology," in *VLSI Technology, 2001. Digest of Technical Papers. 2001 Symposium on*, 2001, pp. 105-106.

[22] R. Castagnetti, R. Venkatraman, B. Bartz, C. Monzel, T. Briscoe, A. Teene*, et al.*, "A high-performance SRAM technology with reduced chip-level routing congestion for SoC," in *Quality of Electronic Design, 2005. ISQED 2005. Sixth International Symposium on*, 2005, pp. 193-196.

[23] T. Okumura and M. Hashimoto, "Setup time, hold time and clock-to-Q delay computation under dynamic supply noise," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences,* vol. 94, pp. 1948-1953, 2011.

[24] D. Markovic, B. Nikolic, and R. Brodersen, "Analysis and design of low-energy flip-flops," in *Proceedings of the 2001 international symposium on Low power electronics and design*, 2001, pp. 52-55.

[25] H. Abrishami, S. Hatami, and M. Pedram, "Design and Multi-Corner Optimization of the Energy-Delay Product of CMOS Flip-Flops under the NBTI Effect."

[26] D. Li, P. Chuang, and M. Sachdev, "Comparative analysis and study of metastability on high-performance flip-flops," in *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, 2010, pp. 853-860.

[27] D. Rennie, D. Li, M. Sachdev, B. L. Bhuva, S. Jagannathan, S. Wen*, et al.*, "Performance, metastability, and soft-error robustness trade-offs for flip-flops in 40 nm CMOS," *Circuits and Systems I: Regular Papers, IEEE Transactions on,* vol. 59, pp. 1626-1634, 2012.

[28] C. Foley, "Characterizing metastability," in *Advanced Research in Asynchronous Circuits and Systems, 1996. Proceedings., Second International Symposium on*, 1996, pp. 175-184.

Vita

Xiaowei Zhang was born in Leshan, Sichuan, P.R. China.

Education

August, 2012 — Present

MSEE

Dept. of Electrical and Computer Engineering, University of Kentucky

August, 2007 — May, 2011

BSEE

School of Microelectronics and Solid-state Electronics, University of Electronic Science and Technology of China