2012

Characterization and modeling of digital circuits using the ferroelectric transistor

Michael Cody Mitchell

CHARACTERIZATION AND MODELING OF DIGITAL CIRCUITS USING THE FERROELECTRIC TRANSISTOR

by

MICHAEL CODY MITCHELL

A THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in
The Department of Electrical and Computer Engineering
to
The School of Graduate Studies
of
The University of Alabama in Huntsville

HUNTSVILLE, ALABAMA

2012

In presenting this thesis in partial fulfillment of the requirements for a master's degree from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department or the Dean of the School of Graduate Studies. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Michael Cody Mitchell 8/20/2012

(Student Signature) (Date)

THESIS APPROVAL FORM

Submitted by Michael Cody Mitchell in partial fulfillment of the requirements for the degree of Master of Science in Engineering with an option in Electrical Engineering and accepted on behalf of the faculty of the School of Graduate Studies by the thesis committee.

We, the undersigned members of the Graduate Faculty of The University of Alabama in Huntsville, certify that we have advised and/or supervised the candidate on the work described in this thesis. We further certify that we have reviewed the thesis manuscript and approve it in partial fulfillment of the requirements for the degree of Master of Science in Engineering.

August 20, 2012
Committee Chair
(Date)

8/20/2012

8/20/2012

8/24/12 Department Chair

08/31/12 College Dean

9/28/12 Graduate Dean

ABSTRACT

The School of Graduate Studies
The University of Alabama in Huntsville

Degree Master of Science in Engineering College/Dept. Engineering/Electrical & Computer Engineering
Name of Candidate Michael Cody Mitchell
Title Characterization and Modeling of Digital Circuits Using the Ferroelectric Transistor

This thesis provides a detailed characterization of the usage of the metal-ferroelectric-semiconductor field-effect transistor (MFSFET) in digital circuits. The properties of the ferroelectric material and the unique behavior resulting from this are compared to the general behavior of the metal-oxide-semiconductor field-effect transistor (MOSFET). The behavior of the MFSFET is characterized in the inverter, static random access memory (SRAM), and dynamic random access memory (DRAM) circuits. The unique behavior of each circuit is examined to determine the advantages of utilizing MFSFETs to replace MOSFETs. Finally, an empirical model of the MFSFET is discussed, detailing a method of predicting the drain current of the transistor; the model also includes simulation of the inverter and SRAM circuits based on the MFSFET.

Abstract Approval:
Committee Chair Jat Deen Ho August 20, 2012
Department Chair Robert Tinker 8/21/12
Graduate Dean Rhonda Kay Gaede 9/28/12

ACKNOWLEDGMENTS

The research presented in this thesis is not based solely on my efforts, but was only possible due to the help and guidance provided by mentors, co-workers, and friends. I want to give special thanks to Dr. Fat D. Ho for his guidance, wisdom, and encouragement throughout my graduate studies and research activities. I am also grateful to my other committee members who have provided their time to review this thesis.

Next, I want to thank Crystal Laws McCartney of the UAH Graduate School for her help during lab experiments and discussions concerning this thesis.

I want to acknowledge the efforts of Mitchell Hunt of the UAH Graduate School for his guidance in lab experiments as well. His wealth of knowledge and insight into the research being conducted have proven to be very useful during countless experiments.

I also want to thank Todd MacLeod of NASA for his help during my research. Mr. MacLeod effectively served as a second advisor to me, and his affiliation with NASA has made many of the experiments possible through the use of specialized equipment.

I would also like to acknowledge Dr. Rana Sayyah for her assistance at various points of my research, specifically in modeling the ferroelectric transistor.

Thanks are also owed to Joe Evans of Radiant Technologies for supplying the transistors used in these experiments.

I want to thank God for the patience and determination to continue my graduate studies. Finally, I thank my family for their encouragement of my education, as well as my employer, Mentor Graphics, for supporting me as I further my education.

TABLE OF CONTENTS

Page

LIST OF FIGURES ix

LIST OF TABLES xv

LIST OF SYMBOLS xvi

Chapter

I. INTRODUCTION 1

II. FERROELECTRIC THEORY 3

A. Properties of Ferroelectric Materials 3

B. Ferroelectric Transistors 8

C. Ferroelectric Memories and Devices 18

III. THE INVERTER CIRCUIT 19

A. Inverter Theory 20

a. Resistive-Load Inverters 23

b. Active-Load Inverters 24

B. Ferroelectric Inverter Testing 24

a. Static Characteristics 24

b. Switching Characteristics 37

C. Contrasts to MOSFET-based Inverters 40

D. Analysis 44

IV. THE FERROELECTRIC STATIC RANDOM ACCESS MEMORY CELL 45

A. Effects of the Ferroelectric Transistor 46

a. Logic Levels and Static Characteristics 47
b. Current Measurements 52
c. Switching Characteristics 59
B. Current Contrasts to MOSFET-based SRAM circuits 61
C. Analysis 64
V. THE FERROELECTRIC DYNAMIC RANDOM ACCESS MEMORY CIRCUIT 65
A. The 3T DRAM Circuit 65
a. Test Circuit 66
b. Experimental Data 69
c. Analysis 83
B. The 1T-1C DRAM Cell 83
a. Test Circuit 84
b. Experimental Data 85
c. Analysis 90
VI. MODELING THE FERROELECTRIC TRANSISTOR 92
A. Background of Existing Models 92
B. Development of a New Model 95
C. General Model Execution 99
D. Simulating the Standalone Ferroelectric Transistor 100
E. Simulating the Resistive-Load Inverter 106
F. Simulating the SRAM Circuit 113
G. Performance of the Model 115

H. Accuracy of Model 116

I. Analysis.....119

VII. Conclusion 121

APPENDIX: C++ CODE FOR MFSFET AND CIRCUIT MODEL.....124

REFERENCES 177

LIST OF FIGURES

Figure Page

2.1 Typical relationship between applied electric field and dielectric and paraelectric polarization 4

2.2 Typical relationship between applied electric field and ferroelectric polarization 5

2.3 PZT molecule in neutral, positive, and negative polarization states..... 6

2.4 Orientation of domains for no polarization (top), positive polarization (bottom), and negative polarization (bottom) 7

2.5 Structure of an n-channel MOSFET 8

2.6 Sample MOSFET current for varying gate voltages..... 10

2.7 MFSFET structure 11

2.8 Charges associated with positive gate voltage applied to the MFSFET 12

2.9 Charges associated with removal of positive gate voltage from MFSFET 13

2.10 Charges associated with negative gate voltage applied to the MFSFET 14

2.11 Charges associated with removal of negative gate voltage from MFSFET..... 14

2.12 Test circuit for transistor characterization 15

2.13 Drain current hysteresis for ND1 transistor 15

2.14 Measured gate polarization of ND1 transistor 16

2.15 Measured gate polarization of ND1 transistor after 6 V pulse 17

2.16 Measured gate polarization of ND1 transistor after -6 V pulse 18

3.1 Inverter circuit layouts – resistive load (left) and PMOS load (right) 19

3.2 VTC of an ideal inverter 21

3.3 Inverter VTC with critical points..... 21

3.4 ND7 inverter VTC for V_{DD} = 4 V, R_L = 5 kΩ and R_L = 15 kΩ 25

3.5 ND7 inverter output voltage hysteresis after +4 V poling voltage 27

3.6 ND7 inverter output voltage hysteresis after -4 V poling voltage 28

3.7 ND7 inverter current hysteresis after +4 V poling voltage 29

3.8 ND7 inverter current hysteresis after -4 V poling voltage..... 29

3.9 ND1 inverter output voltage with 5 kΩ load 30

3.10 ND1 inverter output voltage for 150 kΩ and 200 kΩ load resistances..... 31

3.11 ND7 inverter output voltage vs. load resistance for 4 V square wave.....32

3.12 ND7 inverter V_{OL} and V_{OH} values for varying input frequency 32

3.13 ND7 inverter output for negative input square waves 33

3.14 ND7 inverter output for positive input square waves 34

3.15 ND7 inverter full output voltage hysteresis, part 1 of 2 35

3.16 ND7 inverter full output voltage hysteresis, part 2 of 2 35

3.17 ND1 inverter full output voltage hysteresis, part 1 of 2 36

3.18 ND1 inverter full output voltage hysteresis, part 2 of 2 36

3.19 ND7 inverter rise and fall times for varying input frequency.....38

3.20 ND7 inverter rise and fall times for varying load resistance 39

3.21 ND7 inverter rise and fall times for varying supply voltage 40

3.22 ND7 inverter rise and fall times for varying input signal amplitude 40

3.23 MOSFET inverter full output voltage hysteresis, part 1 of 2 41

3.24 MOSFET inverter full output voltage hysteresis, part 2 of 2 41

3.25 All inverter full output voltage characteristics for R_L = 5 kΩ 42

3.26 All inverter full output voltage characteristics for R_L = 105 kΩ 43

3.27 All inverter full output voltage characteristics for PMOS load 43

4.1 SRAM circuit layouts – resistive load (left) and PMOS load (right) 45

4.2 Write operations of MOSFET-based SRAM 47

4.3 Write operations of SRAM with MFSFET at M2 49

4.4 Negative write operation for SRAM with MFSFETs 50

4.5 SRAM node voltages and current for V_{DD} = 2 V and $R_1 = R_2 = 100$ kΩ 53

4.6 SRAM M2 transistor current for varied supply voltages 54

4.7 Time-based representation of current in ND1 SRAM for varying V_{DD} 55

4.8 ND1 SRAM with square wave input after positive and negative poling pulses 56

4.9 SRAM 58D1 transistor drain currents for 0 V to 4 V after 6 V pulse 57

4.10 SRAM ND1 transistor drain currents for 0 V to 4 V after -6 V pulse 58

4.11 MOSFET SRAM Node V2 voltage for 0 V to 4 V square wave input range with 100 kΩ load resistance 62

4.12 ND1 SRAM Node V2 voltage for 0 V to 4 V square wave input range with 100 kΩ load resistance 62

4.13 MOSFET SRAM current for 0 V to 4 V triangle wave input range with 100 kΩ load resistance 63

4.14 ND1 SRAM current for 0 V to 4 V triangle wave input range with 100 kΩ load resistance 63

5.1 Structure of the 3T DRAM circuit 66

5.2 Test circuit for 3T DRAM cell 67

5.3 Active elements during DRAM write operation 67

5.4 Active elements during DRAM read operation 68

5.5 MOSFET 3T DRAM with write operation to store “1” 70

5.6 MOSFET 3T DRAM with read operation to read “1” 71

5.7 MOSFET 3T DRAM with write operation to store “0” 72

5.8 MOSFET 3T DRAM with read operation to read “0” 72

5.9 Mixed MFSFET/MOSFET 3T DRAM with write operation to store “1” 73

5.10 Mixed MFSFET/MOSFET 3T DRAM with read operation to read “1” 74

5.11 Mixed MFSFET/MOSFET 3T DRAM with write operation to store “0” 74

5.12 Mixed MFSFET/MOSFET 3T DRAM with read operation to read “0” 75

5.13 Mixed MFSFET/MOSFET 3T DRAM with write operation to store “1” including negative voltages at WWL.....76

5.14 Mixed MFSFET/MOSFET 3T DRAM with read operation to read “1” including negative voltages at WWL.....76

5.15 Mixed MFSFET/MOSFET 3T DRAM with write operation to store “0” including negative voltages at WWL.....77

5.16 Mixed MFSFET/MOSFET 3T DRAM with read operation to read “0” including negative voltages at WWL.....78

5.17 MFSFET 3T DRAM with write operation to store “1” 78

5.18 MFSFET 3T DRAM with read operation to read “1” 79

5.19 MFSFET 3T DRAM with write operation to store “0” 80

5.20 MFSFET 3T DRAM with read operation to read “0” 80

5.21 MFSFET 3T DRAM with write operation to store “1” including negative voltages at WWL.....81

5.22 MFSFET 3T DRAM with read operation to read “1” including negative voltages at WWL.....81

5.23 MFSFET 3T DRAM with write operation to store “0” including negative voltages at WWL.....82

5.24 MFSFET 3T DRAM with read operation to read “0” including negative voltages at WWL.....82

5.25 Structure of the 1T-1C DRAM circuit.....84

5.26 Test circuit for the 1T-1C DRAM cell 85

5.27 Reading a “1” from a MOSFET 1T-1C DRAM 86

5.28 Reading a “0” from a MOSFET 1T-1C DRAM 86

5.29 Reading a “1” from an ND1 MFSFET 1T-1C DRAM 87

5.30 Reading a “0” from an ND1 MFSFET 1T-1C DRAM 87

5.31 Reading a “1” from an ND1 MFSFET 1T-1C DRAM using negative gate voltages 88

5.32 Reading a “1” from an ND7 MFSFET 1T-1C DRAM using -4 V and 0 V gate voltages 89

5.33 Reading a “0” from an ND7 MFSFET 1T-1C DRAM using -4 V and 0 V gate voltages 90

6.1 Hyperbolic tangent hysteresis loop 93

6.2 Sample Fermi-Dirac distributions 96

6.3 Measured and modeled current hysteresis loop for ND1 transistor 97

6.4 Modeled vs. measured ND1 current for V_{DS} = 1 V 103

6.5 Modeled vs. measured ND1 current for V_{DS} = 2 V 103

6.6 Modeled vs. measured ND1 current for V_{DS} = 3 V 104

6.7 Modeled vs. measured ND1 current for V_{DS} = 4 V 104

6.8 Modeled vs. measured ND1 current for V_{DS} = 5 V 105

6.9 Modeled vs. measured ND1 current for V_{DS} = 5 V, minor loop 106

6.10 Modeled vs. measured ND7 current for V_{DS} = 1 V 106

6.11 Behavior of inverter V_{OH} for varying V_{DD} 108

6.12 Behavior of inverter V_{OL} for varying V_{DD} 108

6.13 Behavior of inverter V_{OH} for varying load resistance 109

6.14 Behavior of inverter V_{OL} for varying load resistance 109

6.15 Modeled output voltage of ND1 inverter for V_{DD} = 4 V, R_L = 105 kΩ 112

6.16 Modeled drain current of ND1 inverter for V_{DD} = 4 V, R_L = 105 kΩ 112

6.17 Modeled output voltage of ND1 inverter for V_{DD} = 5 V, R_L = 5 kΩ 113

6.18 Modeled drain current of ND1 SRAM for V_{DD} = 4 V, R_L = 100 kΩ (transistor M2) 114

6.19 Model performance relative to input dataset size 116

6.20 Percent difference for measured and modeled current in MFSFET with V_{DS} = 3 V 117

6.21 Percent difference for measured and modeled current in MFSFET inverter with V_{DD} = 4 V, R_L = 105 kΩ 118

6.22 Percent difference for measured and modeled current in MFSFET with V_{DD} = 4 V, R_L = 100 kΩ (transistor M2) 119

LIST OF TABLES

Table		Page
2.1	Curie temperatures for 2- μm PZT films and ceramics	7
2.2	Radiant MFSFET channel dimensions	12
3.1	ND7 inverter output voltage for varied V_{DD} and R_L	26
3.2	ND7 inverter output voltages at high resistance	26
4.1	Retention time analysis of SRAM for -4 V input	51
4.2	Retention times for negative applied voltages	52
4.3	Resistive load SRAM timing characteristics	59
4.4	PMOS load SRAM timing characteristics	60
5.1	Logic values of 3T DRAM elements during read operation	69
6.1	Decision logic of the MFSFET model based on gate voltage	102
6.2	V_{OH} for the MFSFET inverter based on varied supply and load values	107
6.3	V_{OL} for the MFSFET inverter based on varied supply and load values	107
6.4	Coefficients for curve-fitting cubic equation	110
6.5	Performance of model for varying number of inputs	115

LIST OF SYMBOLS

C_{OX}	Oxide capacitance
C_S	Storage capacitance of DRAM
E	Energy
E_C	Coercive electric field
E_f	Fermi level for an electron shell
I_D	Drain current
I_D^+	Drain current for increasing gate voltage
I_D^-	Drain current for decreasing gate voltage
I_{lower}	Drain current lower bound, dynamic parameter of model
I_{max}	Drain current maximum bound
I_{min}	Drain current minimum bound
I_{upper}	Drain current upper bound, dynamic parameter of model
L	Channel length
P	Polarization
P_R	Remnant polarization
P_S	Ferroelectric saturation polarization
R_L	Load resistance
T	Temperature
T_C	Curie point, or Curie temperature
V	Voltage
V_B	Substrate voltage

V_C Coercive voltage

V_D Drain voltage

V_{DD} Power supply voltage

V_{DS} Drain to source voltage

V_G Gate voltage

V_{GS} Gate to source voltage

V_{in} Inverter input voltage

V_{IH} Inverter high input voltage

V_{IL} Inverter low input voltage

V_{offset} Offset voltage, dynamic parameter of model

V_{OH} Inverter high output voltage

V_{OL} Inverter low output voltage

V_{out} Inverter output voltage

V_{PP} Peak-to-peak voltage

V_S Source voltage

V_{SF} Scaling factor of model

V_T Threshold voltage of a transistor

V_{th} Inverter threshold voltage

V_X Internal node voltage of 3T DRAM

V_Z Offset/shift voltage of model

W Channel width

$f(E)$ Probability of an electron at energy level E

i_D Drain current, including small signals

k	Boltzmann constant
v_{DS}	Drain to source voltage, including small signals
v_{GS}	Gate to source voltage, including small signals
ε	Dielectric permittivity
ε_0	Permittivity of free space
λ	Channel length modulation parameter
μ	Electron mobility
τ	Timing parameter/measurement
χ_e	Electrical susceptibility

CHAPTER I

INTRODUCTION

Ferroelectricity is the property by which a material's crystals are in a polarized state in the absence of an applied electric field, and that polarization may be changed by the application of an external electric field [1]. This leads to a hysteresis effect, which means that, unlike devices whose output values depend solely on the input values, ferroelectric devices produce outputs that depend on the input values, as well as the previous state of the device.

In this thesis, the characteristics of certain circuits involving metal-ferroelectric-semiconductor field-effect transistors (MFSFETs) will be presented, with comparisons made to their counterparts based on metal-oxide-semiconductor field-effect transistors (MOSFETs). Following this introductory chapter, Chapter II will discuss the theory of ferroelectricity and its effects on devices such as the transistor. Chapter III will focus on the inverter logic circuit, and the impact of replacing the traditional MOSFET with a ferroelectric transistor. Similarly, Chapter IV will present research related to the static random access memory (SRAM) cell utilizing the ferroelectric transistor in various configurations. In Chapter V, experimental data will be presented showing the effects of replacing one or more MOSFETs with MFSFETs in two variations of the dynamic random access memory (DRAM) circuit: the three-transistor as well as the one-transistor,

one-capacitor DRAM circuits. Chapter VI will detail the efforts of modeling the MFSFET, inverter, and SRAM circuit. Finally, Chapter VII will present a conclusion and statement of potential future research.

CHAPTER II

FERROELECTRIC THEORY

The theory of ferroelectric phenomena has been known for a long time. It was first discovered as a property of Rochelle salt in 1921 by Joseph Valasek. Rochelle salt was chosen as the subject of Valasek's research because it was already known to show piezoelectricity, and it also revealed properties such as pyroelectricity and optical activity. According to Valasek, factors that prompted research involving Rochelle salt were its strange electro-optical properties; electromechanical properties stronger than those of quartz; properties dependent not only on temperature and electric field, but also on the previous history of electrical and mechanical treatment; and military needs for sensitive electromechanical transducers [2].

Rochelle salt was one of the crystals tested by the Curie brothers, Jacques and Pierre, who discovered the property of piezoelectricity in 1880 [3]. Piezoelectricity is the property by which an applied mechanical stress induces an electric charge in a material. Pyroelectricity is the property by which a temperature change induces an electric field in a material. Ferroelectric materials are by nature both piezoelectric and pyroelectric [4].

A. Properties of Ferroelectric Materials

The distinguishing property of ferroelectricity is the hysteresis loop seen in a plot of polarization of the material with respect to varying applied electric fields. There is no

linear or direct relationship between an applied electric field and the resulting polarization; instead, the polarization not only depends on the input, which would be the applied field, but also on the current and previous states of the material [5].

To understand why hysteresis makes ferroelectric polarization unique, it is important to see how dielectric and paraelectric materials behave. Dielectric polarization varies linearly with the applied field. Figure 2.1 shows an example of this type of polarization. In fact, the polarization of most dielectric materials can be defined as seen in (2.1), where χ_e is the electrical susceptibility of the material, ε_o is the permittivity of free space, and $\vec{E}$ is the applied electric field [5].

$$\vec{P} = \chi_e \varepsilon_o \vec{E} \tag{2.1}$$

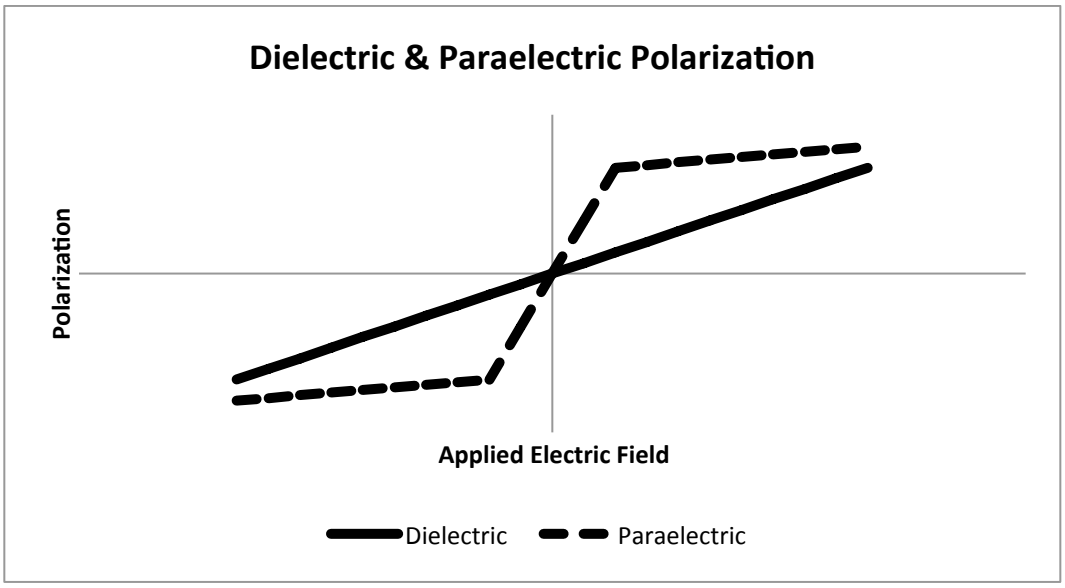

Figure 2.1 Typical relationship between applied electric field and dielectric and paraelectric polarization

Paraelectric materials show a nonlinear relationship between the applied field and the polarization. Even though it is nonlinear, the relationship shows that the polarization is a function of the applied field, as demonstrated in Figure 2.1.

Ferroelectric polarization, however, cannot simply be described in terms of the applied electric field. Figure 2.2 shows an example of ferroelectric polarization and some of the critical points on such a plot.

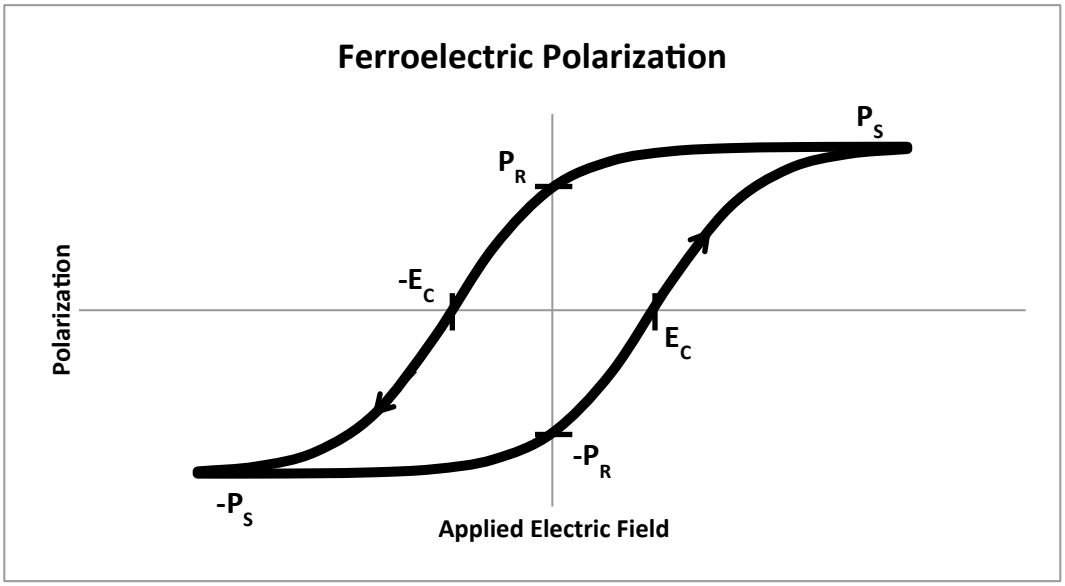

Figure 2.2 Typical relationship between applied electric field and ferroelectric polarization

The maximum and minimum polarization values that are found for the more extreme applied electric fields are identified as P_S and $-P_S$, the saturation polarization. When the applied electric field is removed, the value of the polarization when positively polarized is P_R, the remanent polarization; likewise, the value of the polarization is $-P_R$ when the material is negatively polarized and then the field is removed. The coercive field values,

E_C and $-E_C$, are the applied electric field values at which the material begins changing its polarization from negative to positive or positive to negative [6].

Ferroelectric materials have a polarization that does not disappear when the external field is removed. Instead, the external field must reach the coercive voltage, or its negative value if already positively polarized, in order to remove the polarization from the material. The reasons behind this can be seen by examining a lead zirconate titanate (PZT) molecule, with as example shown in Figure 2.3; PZT is a widely used ferroelectric material, and is used in the transistors that are characterized in this thesis [7], [8].

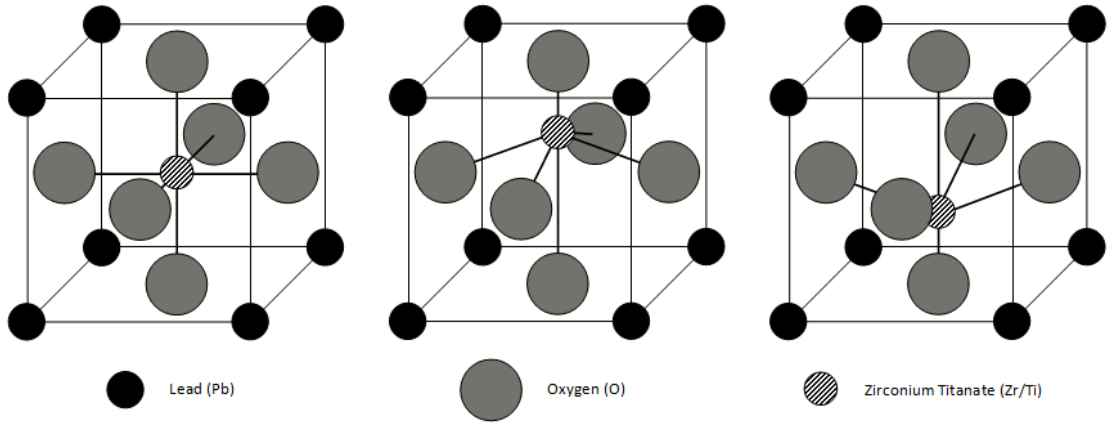

Figure 2.3 PZT molecule in neutral, positive, and negative polarization states

Figure 2.3 shows a single PZT molecule in its neutral state, with the zirconium titanate at the center of the molecule, followed by its representation when polarized with positive and negative fields. One should pay close attention to the positioning of the center ZrTi atom as it moves upward or downward based on the applied external field. A bulk crystal is composed of regions with varying directions of polarization, known as ferroelectric domains [9]. Figure 2.4 shows a simplistic example of how these domains

are oriented when the material is not polarized, positively polarized, and negatively polarized.

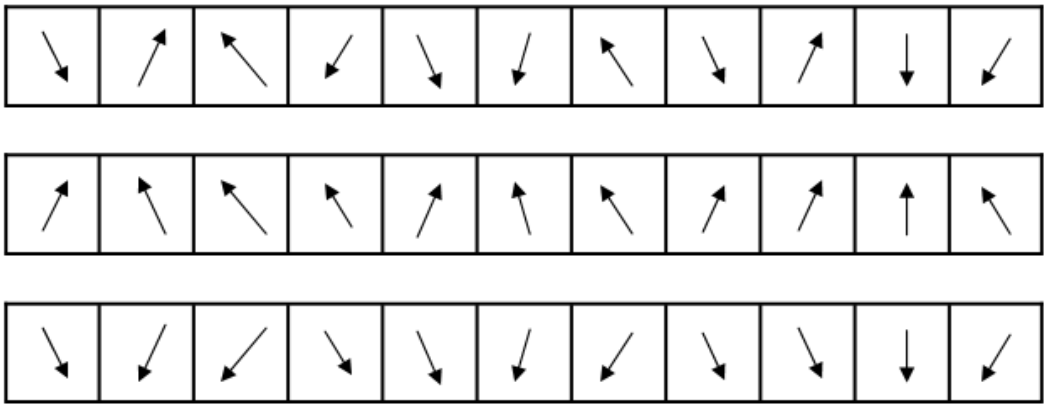

Figure 2.4 Orientation of domains for no polarization (top), positive polarization (center), and negative polarization (bottom)

Ferroelectric behavior is only shown below a critical temperature called the Curie point, T_C. When the crystals of the ferroelectric material are heated above the Curie point, the material begins to show paraelectric behavior. An interesting characteristic of Rochelle salt is that it displays ferroelectric behavior between the temperatures -18°C and 23°C [9]. The Curie point for PZT can depend on the Zr/Ti ratio. The Curie point for 20/80 PZT, used in the MFSFETs in this thesis, is found to be 450°C [10]. Some other example Curie temperatures for PZT are shown in Table 2.1 for 2-μm-thick PZT films and ceramics [11].

Table 2.1 Curie temperatures for 2- μm PZT films and ceramics [11]

	PZT 40/60	PZT 45/55	PZT 52/48	PZT 60/40
Films	470°C	435°C	420°C	370°C
Ceramics	419°C	407°C	399°C	366°C

B. Ferroelectric Transistors

Transistors are four-terminal devices that use the voltage between two terminals, the gate and source, to control the current flowing through a third terminal, which is called the drain. These devices are often used in amplifier circuits and as voltage-controlled switches. In particular, the MOSFET is a widely used device in integrated circuit design due to its small size, simple manufacturing process, and low power requirements [12]. The structure of a typical n-channel MOSFET can be seen in Figure 2.5 [13].

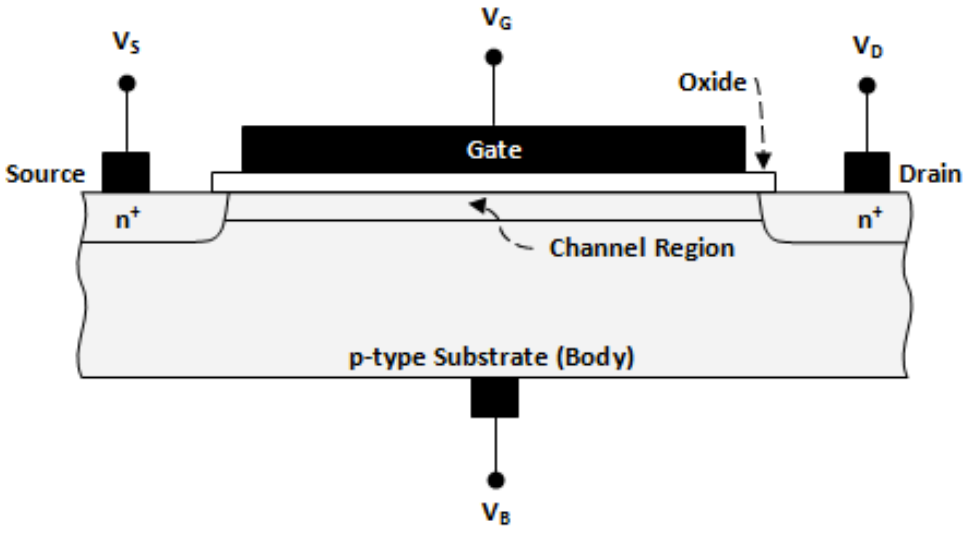

Figure 2.5 Structure of an n-channel MOSFET [13]

As a voltage is applied to the gate terminal, electrons begin to collect in the area denoted as the channel region. When a voltage of sufficient magnitude, known as the threshold voltage, or V_T, is applied to the gate, a channel is formed between the drain and source regions of the transistor. In an n-channel MOSFET, this channel is formed by repelling holes in the substrate just under the gate and pushing them further into the

substrate, and also by the attraction of electrons from the source and drain regions into the channel region. When voltages are applied to the drain and source terminals, current begins to flow from the drain to the source through the channel that was formed. The voltage v_{GS} between the gate and source is used to control the current i_D flowing into the drain of the transistor [12].

There are three regions of operation for the MOSFET: cutoff, triode, and saturation. In its cutoff mode, in which the gate-source voltage v_{GS} has not yet reached the threshold voltage value V_T, no current flows through the transistor in an ideal case; there is a level of subthreshold current that can be calculated for this region of operation, but this is typically neglected in generic discussion of MOSFET operation. In the triode region, sometimes called the linear region, $v_{DS} < v_{GS} - V_T$; as v_{DS} changes in this region, the current also changes because the channel resistance increases with v_{DS}. Once v_{DS} reaches or exceeds $v_{GS} - V_T$, the channel becomes pinched off and there is almost no effect on the drain current as v_{DS} changes past this point. To be thorough, it must be pointed out that there is some impact in increasing v_{DS} once the transistor is in saturation because the channel length will be shortened. This effect is known as channel-length modulation and is considered in the drain current equation with the parameter λ. Equations (2.2) and (2.3) describe the calculation of the drain current for the triode and saturation modes of operation, respectively, for the n-channel MOSFET [12].

$$i_D = \mu_n C_{OX} \frac{W}{L}\left[(v_{GS} - V_T)v_{DS} - \frac{1}{2}v_{DS}^2\right] \tag{2.2}$$

$$i_D = \frac{1}{2}\mu_n C_{OX} \frac{W}{L}(v_{GS} - V_T)^2(1 + \lambda v_{DS}) \tag{2.3}$$

A sample plot of these current equations, for arbitrary parameters, is shown in Figure 2.6 for five distinct values of v_{DS} as the gate voltage v_{GS} is varied. What is not

obvious in the plot is that the input voltages ranged not only from -6 V to 6 V, but also back to -6 V again in order to show that the past and present states of the transistor have no bearing on the drain current.

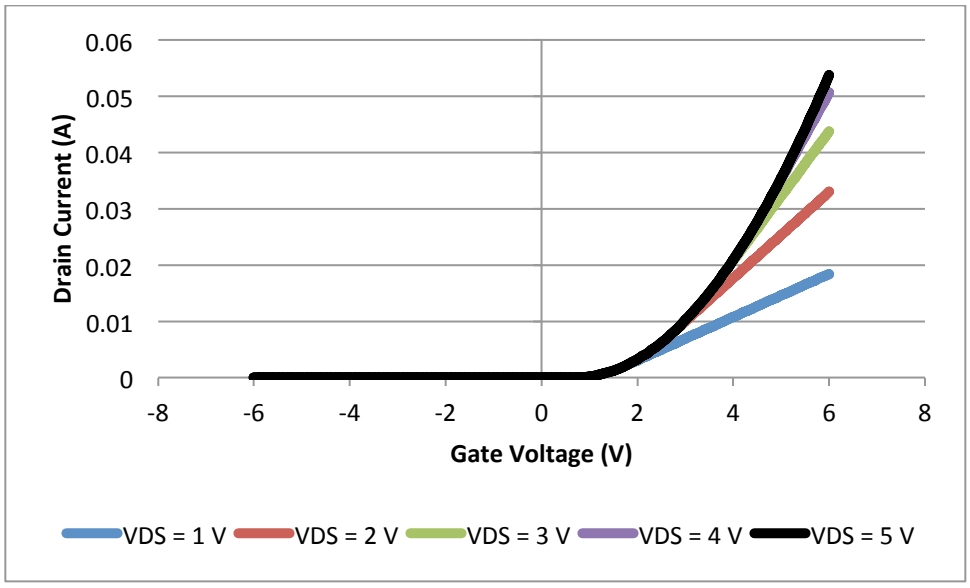

Figure 2.6 Sample MOSFET current for varying gate voltages

There are two types of MOSFETs to be considered in understanding MOSFET operation: the enhancement-mode MOSFET and the depletion-mode MOSFET. Most of the theory presented previously in this chapter applies primarily to the enhancement-mode MOSFET. The depletion-mode MOSFET is similar, with one key difference: it can conduct current at $v_{GS} = 0$ V. In the enhancement-mode transistor, a channel is induced by the application of a gate voltage, and the threshold voltage at which a current is conducted is positive, commonly a value such as 0.7 V. The depletion-mode transistor, however, has a channel implanted and can conduct current with a 0 V gate voltage

applied. There is a negative threshold voltage at which the transistor ceases to conduct current if the gate voltage is decreased past this point [12].

The previous equations show that the MOSFET drain current can be calculated at any given time, based on the voltages applied at each terminal and the parameters of the transistor. The ferroelectric transistor (MFSFET), however, cannot be analyzed by simply using equations based on terminal voltages, due to the polarization effects described earlier in this chapter.

The MFSFETs used in the experiments presented in this thesis were provided by Joe Evans of Radiant Technologies, Inc. in Albuquerque, New Mexico. These transistors are thin-ferroelectric-film-transistors using PZT as the ferroelectric gate material. Figure 2.7 provides a diagram of the transistor structure [14].

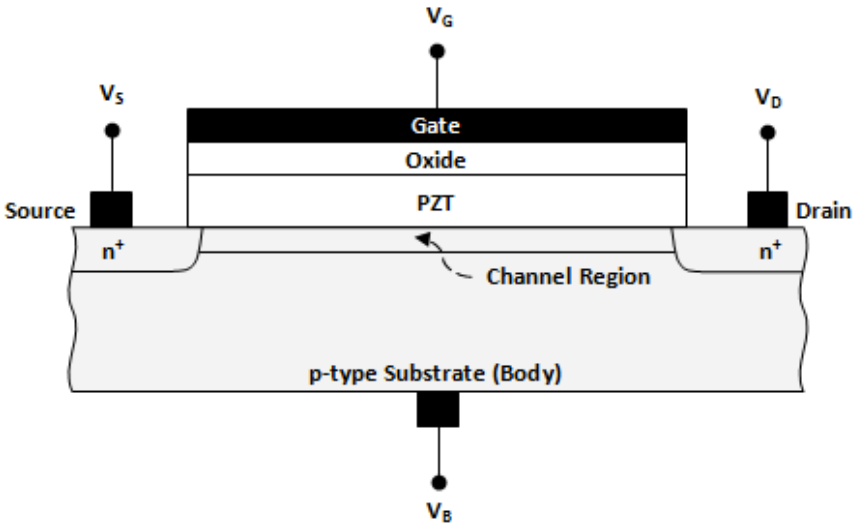

Figure 2.7 MFSFET structure [14]

For the transistors provided, the PZT thickness was 3500 Å of 20/80 PZT and the indium oxide layer was 200 Å. The gate contained 1500 Å-thick platinum, while the platinum

drain and source were 1000 Å. The channel dimensions varied among the two types of transistors used, shown in Table 2.2 [8].

Table 2.2 Radiant MFSFET channel dimensions

Device	Channel Width	Channel Length
ND1	10 µm	10 µm
ND7	400 µm	4 µm

When a positive voltage is applied to the gate of the MFSFET, the polarization in the ferroelectric layer is aligned such that the negative charges are located at the oxide-ferroelectric interface and positive charges at the ferroelectric-semiconductor interface. Negative charges from the surface of the semiconductor are attracted into the ferroelectric layer. Once this gate voltage is removed, the negative charges that were attracted into the ferroelectric layer will remain due to remanent polarization, and a positive charge will be seen at the semiconductor surface. This can be considered as an off state for the transistor, and is similar to the behavior of an enhancement-mode MOSFET [14]. Figures 2.8 and 2.9 show the application and removal of the positive voltage at the gate.

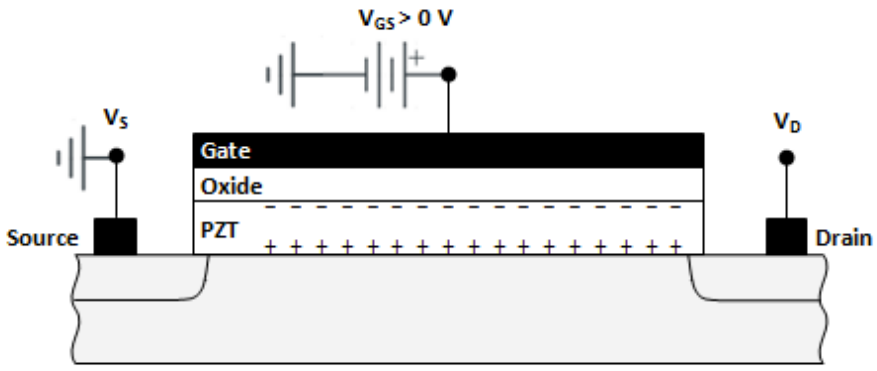

Figure 2.8 Charges associated with positive gate voltage applied to the MFSFET [14]

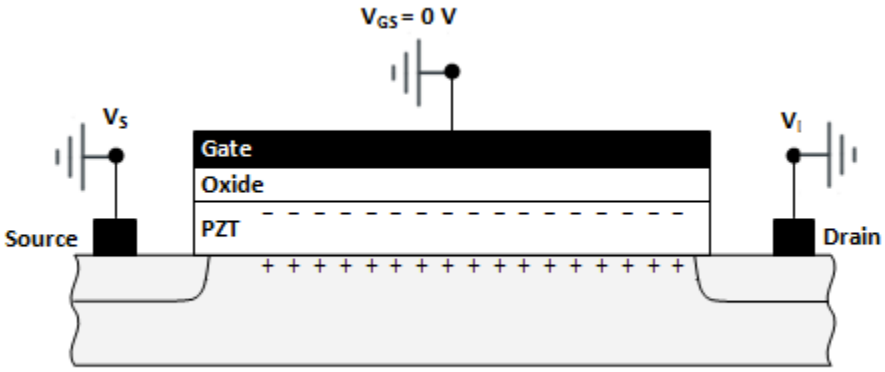

Figure 2.9 Charges associated with removal of positive gate voltage from MFSFET [14]

When applying negative voltages to the gate of the MFSFET, positive charges are seen at the oxide-ferroelectric interface and negative charges at the ferroelectric-semiconductor interface. This attracts positive charges from the semiconductor surface into the ferroelectric layer. When the negative gate voltage is removed, the positive charges remain in the ferroelectric layer, bound to the ferroelectric domains, and a negative charge is seen at the semiconductor surface. This negative charge forms a conducting electron channel so that current can flow from the drain to the source. The transistor is considered on at the point [14]. Figures 2.10 and 2.11 show the application and removal of the negative voltage at the gate.

To characterize and understand the operation of the MFSFETs, current and polarization measurements were taken using a Radiant Technologies Precision Premier II ferroelectric test system and its included Vision software. Figure 2.12 shows a typical setup used to characterize a single transistor. The I2C port was held at a constant voltage and was typically used for the drain. The Drive port was varied during the tests, making

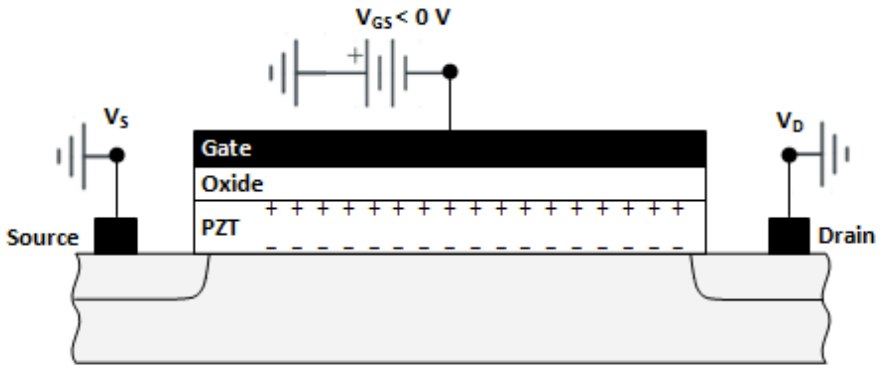

Figure 2.10 Charges associated with negative gate voltage applied to the MFSFET [14]

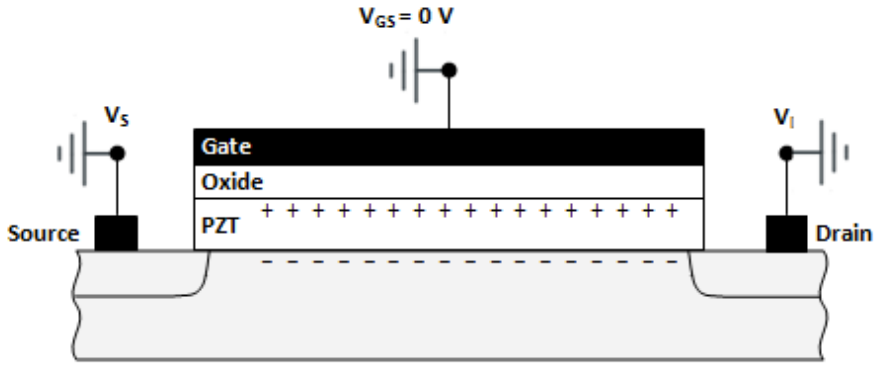

Figure 2.11 Charges associated with removal of negative gate voltage from MFSFET [14]

it suitable for the gate connection. The Return port measured the current from the source of the transistor. During testing, it was important to note the response of the transistor after providing a poling voltage on the gate; this poling action was performed by setting the drain, substrate, and source voltages to 0 V, and then applying a sufficiently positive or negative voltage, such as 6 V, to the gate for a small amount of time in order to switch the polarization in the ferroelectric layer of the transistor.

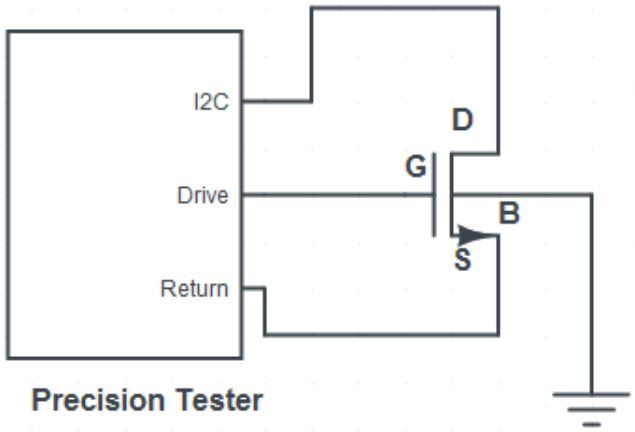

Figure 2.12 Test circuit for transistor characterization

Figure 2.13 shows a characteristic drain current hysteresis for the ND1 MFSFET for various discrete values of v_{DS}. For each loop, the input voltages provided on the gate began with 0 V, increased to 6 V, decreased to -6 V, and increased again to 6 V to finish the series. The input voltage was incremented by 0.1 V for each measurement.

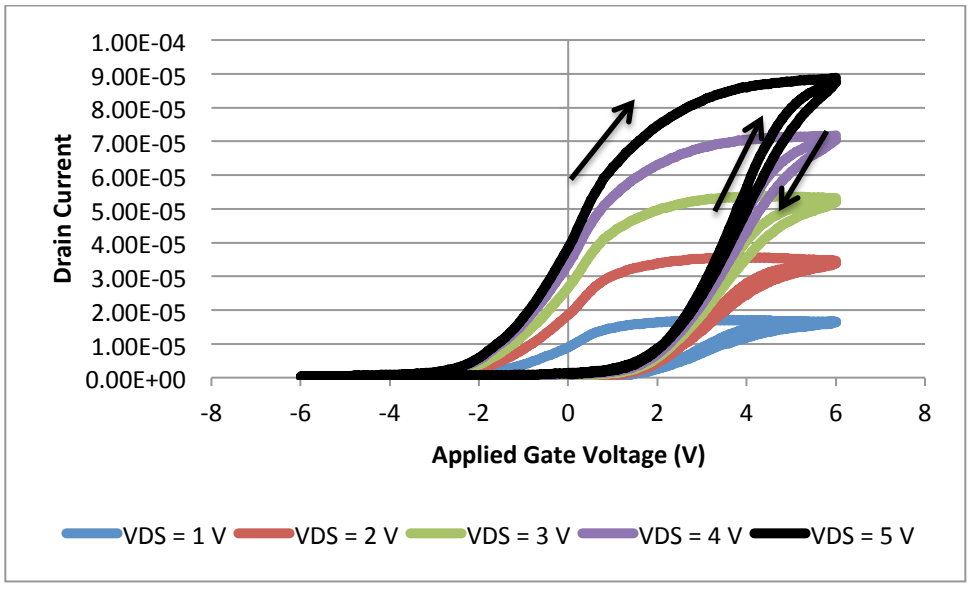

Figure 2.13 Drain current hysteresis for ND1 transistor

Upon observation of the figure, it is shown that the trip from 6 V to -6 V followed much the same path as the initial series beginning with 0 V. This is because the Precision tester was configured to provide a 6 V poling pulse to the gate before testing. Once the input reached a negative value sufficient to switch the polarization in the MFSFET, a much lower voltage was required to turn on the transistor and allow current to flow. In this way, the MFSFET displayed properties of behaving as both a depletion-mode and enhancement-mode transistor: depletion-mode if negatively pulsed, and enhancement-mode if positively pulsed [15].

The Precision tester was also used to obtain a plot of the gate polarization in the ND1 transistor, shown in Figure 2.14, for a full loop of gate voltage values ranging between -7 V and 7 V. In contrast with the ideal polarization plot shown in Figure 2.2, the actual polarization of this transistor displayed a less distinct behavior in terms of identifying critical electric field and polarization values.

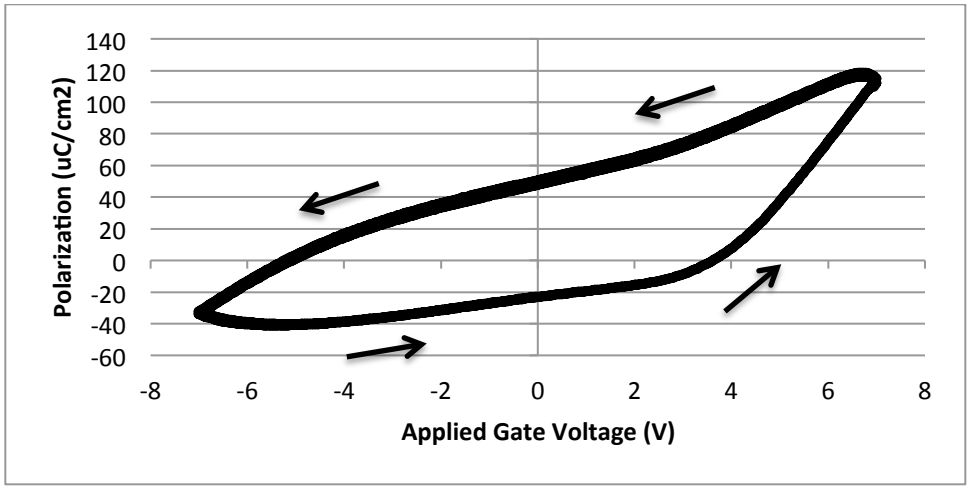

Figure 2.14 Measured gate polarization of ND1 transistor

Figure 2.15 shows the polarization of the gate of the ND1 transistor in a case where a 6 V positive poling pulse was applied to the gate, and then a series of gate voltages were applied. This range started with 0 V, increased to 6 V, decreased to -6 V, and then returned to 0 V again. The initial reading was 0, and the polarization increased as the gate voltage was increased. Once reaching the top gate voltage, the polarization decreased with the gate voltage as it approached -6 V, and increased slightly again as it approached 0 V for the final gate voltage application.

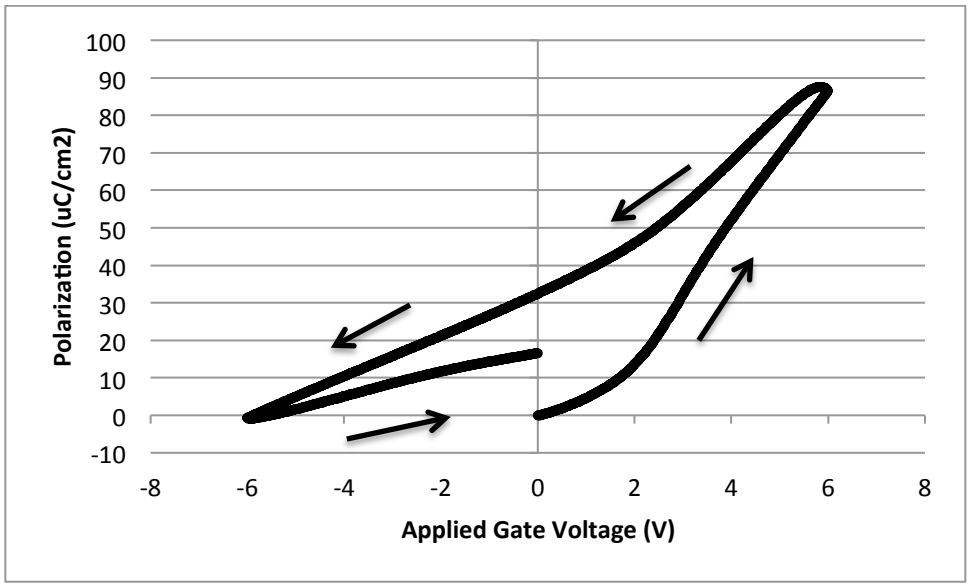

Figure 2.15 Measured gate polarization of ND1 transistor after 6 V pulse

Figure 2.16 shows a scenario similar to Figure 2.15, except that a -6 V poling pulse was used initially, and then the inputs ranged from 0 V to -6 V, -6 V to 6 V, and then 6 V to 0 V. The initial reading for this test was also 0, indicating a default initial value for the tester, since it is unlikely that the transistor had exactly no polarization before beginning each test.

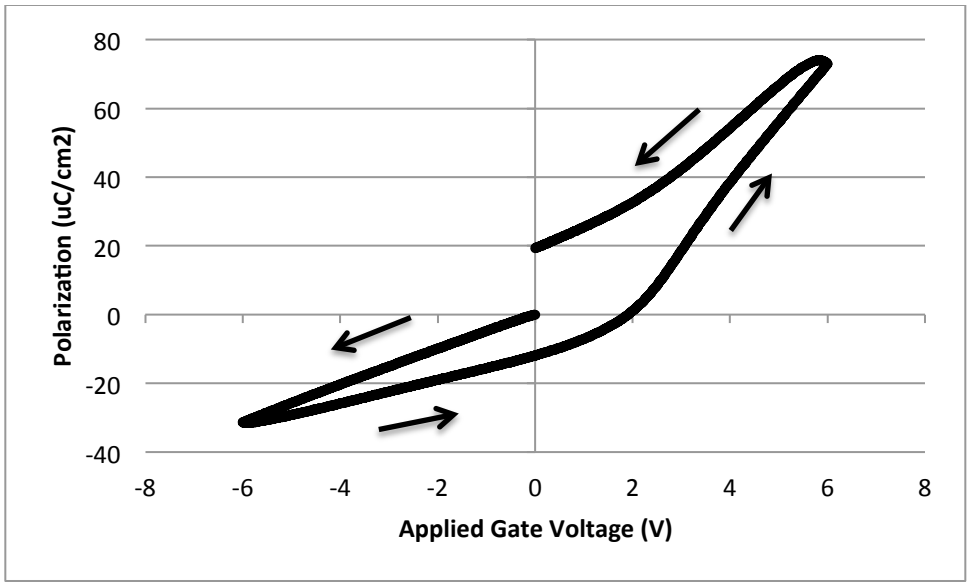

Figure 2.16 Measured gate polarization of ND1 transistor after -6 V pulse

C. Ferroelectric Memories and Devices

Ferroelectric materials have been widely investigated for usage in memories. Because of the bistable polarization states of such devices, much research has been conducted into applications as binary memory devices [16], [17]. Utilizing the effects of the polarization of the ferroelectric materials included in these devices, it becomes possible to create memory circuits capable of nonvolatile storage so that the loss of power does not result in the loss of data storage in the memory. This thesis will explore this, particularly in the SRAM cell in Chapter IV. In addition to ferroelectric transistors, ferroelectric capacitors are available and used as well [18]. These capacitors can be utilized in a DRAM circuit to create nonvolatile memory [16], but Chapter V of this thesis will actually investigate the effect of using ferroelectric transistors rather than capacitors in that particular circuit.

CHAPTER III

THE INVERTER CIRCUIT

The inverter is a fundamental building block of digital logic gates and circuits. While the ferroelectric transistor in digital circuits has typically been limited to usage in memory applications, its presence in the inverter circuit provides unique behavior due to the hysteresis effect present in the ferroelectric material in the transistor. The inverter, shown in Figure 3.1, appears in two common forms: a resistive load or a PMOS transistor load, both with an NMOS driver transistor. In addition to these forms, it is also possible for the load to be an NMOS transistor [19].

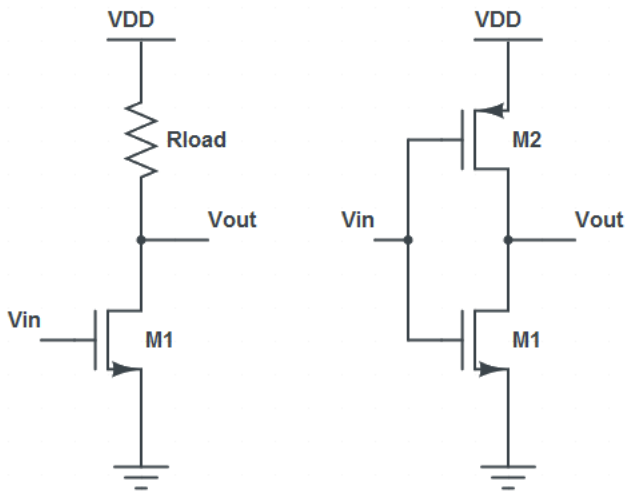

Figure 3.1 Inverter circuit layouts – resistive load (left) and PMOS load (right)

In this chapter, some generic concepts of inverters are presented, followed by a presentation of data collected from inverter circuits based on a ferroelectric transistor. Comparisons are then made to equivalent inverters containing a traditional MOSFET. Because the resistive-load inverter was the focus of research involving the ferroelectric transistor, the concepts presented here for the inverter are primarily composed of resistive-load inverter characteristics.

A. Inverter Theory

In digital logic, Boolean logic values are represented by certain voltages understood to be high or low. A "0" logic level is likely represented by a low voltage like 0 V, while a logic value of "1" is represented by a high voltage, such as the V_{DD} value for a circuit. An inverter circuit takes an input voltage and outputs a voltage that is opposite in logic; for example, applying the V_{DD} voltage value to the input terminal of an inverter should produce an output of approximately 0 V, and applying 0 V at the input terminal should produce an output of approximately V_{DD}.

In an ideal inverter circuit, a threshold voltage that is equal to one half of the supply voltage would be the input voltage at which the output transitions from one logic level to the other, as shown in Figure 3.2 [19]. However, an actual inverter does not necessarily display these ideal characteristics, and instead may contain multiple points along the voltage transfer characteristic (VTC) plot that define voltage ranges to represent the input and output logic levels. Figure 3.3 shows a realistic VTC of an inverter, with these distinct points identified.

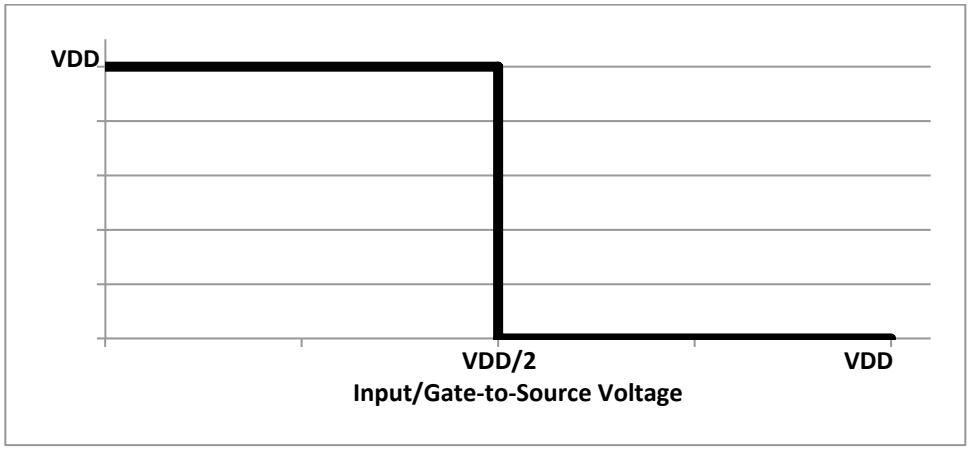

Figure 3.2 VTC of an ideal inverter

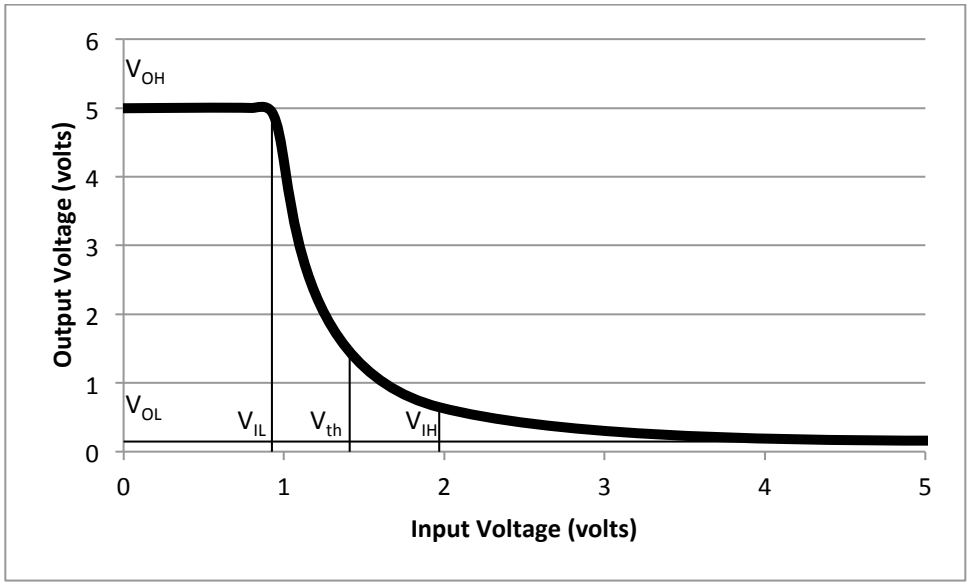

Figure 3.3 Inverter VTC with critical points

In Figure 3.3, there are five specific points identified in the VTC: V_{OL}, V_{OH}, V_{IL}, V_{IH}, and V_{th}. The output low voltage, V_{OL}, is the output voltage of the circuit when the input voltage is sufficiently high to cause the transistor to conduct current and allow only

a minimal voltage to be dropped across the transistor. The output high voltage, V_{OH}, is the output voltage seen at low input voltages, when the transistor is in cutoff mode, no current flows in the circuit, and most of the supply voltage is seen at the drain of the transistor. When the input voltage of the circuit is at or below the input low voltage, V_{IL}, the input is considered to be a logic "0" and the output is at or around V_{OH}. The input high voltage, V_{IH}, is the minimum input voltage that represents a logic "1" value. The final point of interest on the plot is the inverter threshold voltage, V_{th}, which is the point on the characteristic where the input and output voltages are equal [19].

Concerning the switching characteristics of an inverter, there are further key voltages and critical time values that characterize an inverter circuit [19]. The voltage $V_{50\%}$ is defined in (3.1).

$$V_{50\%} = \frac{1}{2}(V_{OL} + V_{OH}). \tag{3.1}$$

This voltage is important in determining the τ_{PLH} and τ_{PLH} time delay values of the circuit; τ_{PLH} is defined as the time required for the output voltage to rise from V_{OL} to $V_{50\%}$, and τ_{PHL} is the time required for the output to drop from V_{OH} to $V_{50\%}$ [19]. The $V_{10\%}$ and $V_{90\%}$ voltages are defined in (3.2) and (3.3).

$$V_{10\%} = V_{OL} + 0.1 \times (V_{OH} - V_{OL}) \tag{3.2}$$

$$V_{90\%} = V_{OL} + 0.9 \times (V_{OH} - V_{OL}) \tag{3.3}$$

Knowing the $V_{10\%}$ and $V_{90\%}$ values, the τ_{fall} and τ_{rise} times may be determined; τ_{fall} is the amount of time it takes for the output voltage to fall from $V_{90\%}$ to $V_{10\%}$, and τ_{rise} is the amount of time it takes for the output voltage to rise from $V_{10\%}$ to $V_{90\%}$ [19].

a. Resistive-Load Inverters

In a resistive-load inverter, the current flowing through the resistor is equal to the current through the driver transistor, since these are the only two elements in the circuit, other than supply voltages, and they are connected in series. Using this, one can calculate the previously described critical voltages that define the thresholds for logic level interpretation in the circuit.

The expected values of the critical voltages for the resistive-load inverter, as derived in [19], are listed in (3.4) through (3.9).

$$V_{OH} = V_{DD} \tag{3.4}$$

$$V_{OL} = V_{DD} - V_{T0} + \frac{1}{k_n R_L} - \sqrt{\left(V_{DD} - V_{T0} + \frac{1}{k_n R_L}\right)^2 - \frac{2V_{DD}}{k_n R_L}} \tag{3.5}$$

$$V_{IL} = V_{T0} + \frac{1}{k_n R_L} \tag{3.6}$$

$$V_{out}(V_{in} = V_{IL}) = V_{DD} - \frac{1}{2k_n R_L} \tag{3.7}$$

$$V_{IH} = V_{T0} + \sqrt{\frac{8}{3}\frac{V_{DD}}{k_n R_L}} - \frac{1}{k_n R_L} \tag{3.8}$$

$$V_{out}(V_{in} = V_{IH}) = \sqrt{\frac{2}{3}\frac{V_{DD}}{k_n R_L}} \tag{3.9}$$

The threshold voltage of the resistive-load inverter, V_{th}, is not derived in [19]. Because the threshold voltage is the point at which V_{in} is equal to V_{out}, which also means that V_{GS} is equal to V_{DS}, the transistor operates in saturation mode at this point since $V_{out} > V_{in} - V_{T0}$. The current of the resistor is set equal to the current of the transistor, and V_{th} is substituted into the current equations in place of V_{in} and V_{out} to find the threshold voltage as shown in (3.10).

$$V_{th} = \left(V_{T0} - \frac{1}{k_n R_L}\right) + \left(\frac{1}{k_n R_L}\right)\sqrt{1 + 2k_n R_L (V_{DD} - V_{T0})} \tag{3.10}$$

b. Active-Load Inverters

The resistive-load inverter presents a simple circuit that is useful for characterizing the behavior of an inverter, but [19] makes it clear that it is not the optimal configuration for a circuit in practical use due to performance and fabrication concerns. The silicon area required for a resistive load to be fabricated is typically more than that required for fabricating an active load. Also, circuits with an active load can be designed to have better performance than circuits with a resistive load. Traditional n-channel MOSFETs can be used as loads in the enhancement-load and depletion-load NMOS inverters. However, an attractive design for the inverter is to use NMOS and PMOS transistors together in a Complementary MOS (CMOS) structure, where the NMOS transistor pulls the output down for a high input and the PMOS transistor pulls the output up for a low input; such a design results in a VTC that comes closer to resembling an ideal inverter, and it also results in negligible power dissipation since only one transistor is on at a time and essentially no current flows through the circuit [19].

B. Ferroelectric Inverter Testing

The experimental results of testing the inverter focus on the static aspects of the circuit, such as the current and the output voltages for varying input parameters. A brief analysis of the switching characteristics is presented, giving insight to the performance of the ferroelectric inverter.

a. Static Characteristics

A ferroelectric inverter was constructed by replacing the traditional MOSFET driver transistor with a ferroelectric transistor. Before investigating the unique aspects of the circuit due to the presence of the ferroelectric material, it was important to verify that

the circuit would function as a basic inverter as well. Figure 3.4 shows a case where V_{DD} was set to 4 V, and inputs ranging from 0 V to 6 V were applied to a resistive load inverter with loads of 5 kΩ and 15 kΩ. Most results shown below, unless otherwise stated, were collected when using the ND7 MFSFET in the inverter circuit.

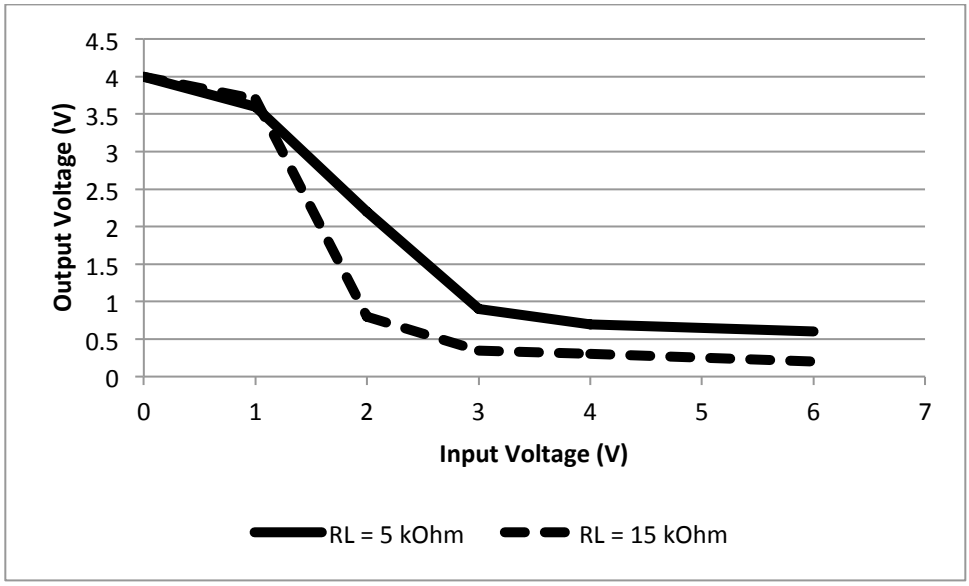

Figure 3.4 ND7 inverter VTC for V_{DD} = 4 V, R_L = 5 kΩ and R_L = 15 kΩ

In addition to this set of data, Table 3.1 also includes measurements taken when V_{DD} was set to 2 V and 3 V. Load resistances of 5 kΩ and 15 kΩ produced optimal results for the inverter circuit using an ND7 MFSFET. Measurements taken show that while high load resistances yielded near-zero V_{OL} values, they also produced V_{OH} values much lower than desired. In Table 3.2, the output voltages for the ferroelectric inverter are given when starting with a high, 6 V, input voltage and decreasing to a 0 V input. While the V_{OL} values for all of the listed loaded resistances were desirable, the V_{OH} values were not acceptable given that the supply voltage was 4 V.

Table 3.1 ND7 inverter output voltage for varied V_{DD} and R_L

V_{DD} (V)	V_{GS} (V)	Output voltage (V), R_L = 5 kΩ	Output voltage (V), R_L = 15 kΩ
2	0	2	1.8
2	1	1.7	1.2
2	2	0.75	0.3
2	3	0.4	0.2
2	4	0.4	0.1
2	6	0.3	0.1
3	0	2.9	2.7
3	1	2.6	1.6
3	2	1.2	0.4
3	3	0.6	0.2
3	4	0.5	0.2
3	6	0.5	0.2
4	0	4	4
4	1	3.6	3.7
4	2	2.2	0.8
4	3	0.9	0.35
4	4	0.7	0.3
4	6	0.6	0.2

Table 3.2 ND7 inverter output voltages at high resistance

V_{DD} (V)	V_{GS} (V)	Output Voltage (V) for Varying Load Resistances					
		40 kΩ	70 kΩ	100 kΩ	200 kΩ	400 kΩ	800 kΩ
4	6	0.1	0	0	0	0	0
4	4	0.1	0	0	0	0	0
4	3	0.1	0	0	0	0	0
4	2	0.3	0.1	0.1	0	0	0
4	1	2	1.2	0.4	0.3	0.2	0
4	0	3.5	2.9	2.8	2	1	0.6

Once it was proven that a ferroelectric transistor could function normally in an inverter circuit, it was important to study any behavior that distinguished it from the traditional MOSFET. Figure 3.4 shows voltage values that only transitioned in one

direction, starting at 6 V and decreasing in 1 V increments until reaching 0 V. However, providing a series of input voltages to the circuit that both increased and decreased showed the hysteresis effects of the ferroelectric material. Figures 3.5 and 3.6 show an example of this, as the input voltage was varied from 0 V to 4 V and decreased again to 0 V, in 0.2 V increments; the supply voltage was set to 4 V and the load resistance was 5 kΩ. The difference between the two figures, however, is that in Figure 3.5 a 4 V poling voltage was applied to the circuit before starting the test, and in Figure 3.6 a -4 V poling voltage was applied first. The figures show that once the input reached a higher value, the ferroelectric material caused a charge to be retained such that an input voltage resulted in an output of higher magnitude than the same input previously produced. The hysteresis curve is evident in the figures.

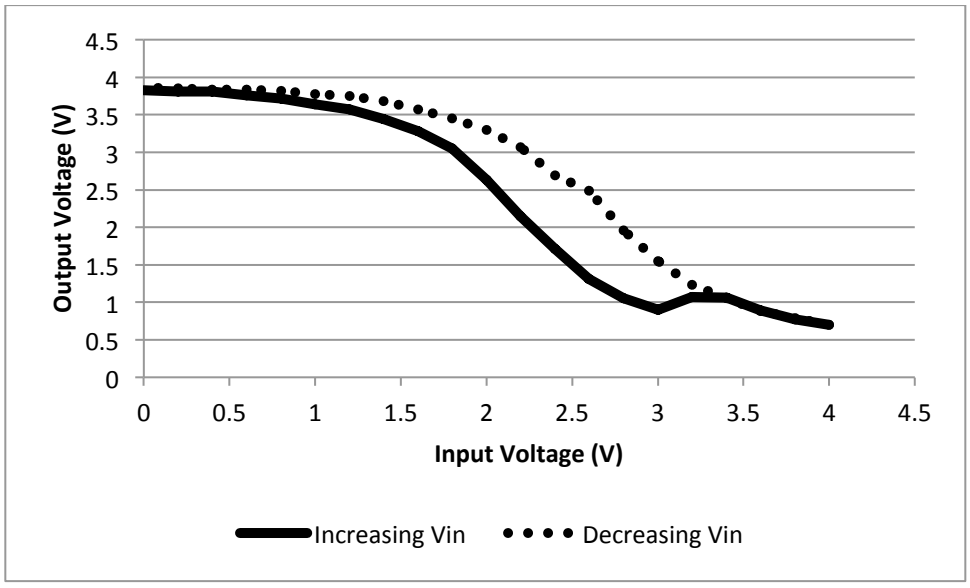

Figure 3.5 ND7 inverter output voltage hysteresis after +4 V poling voltage

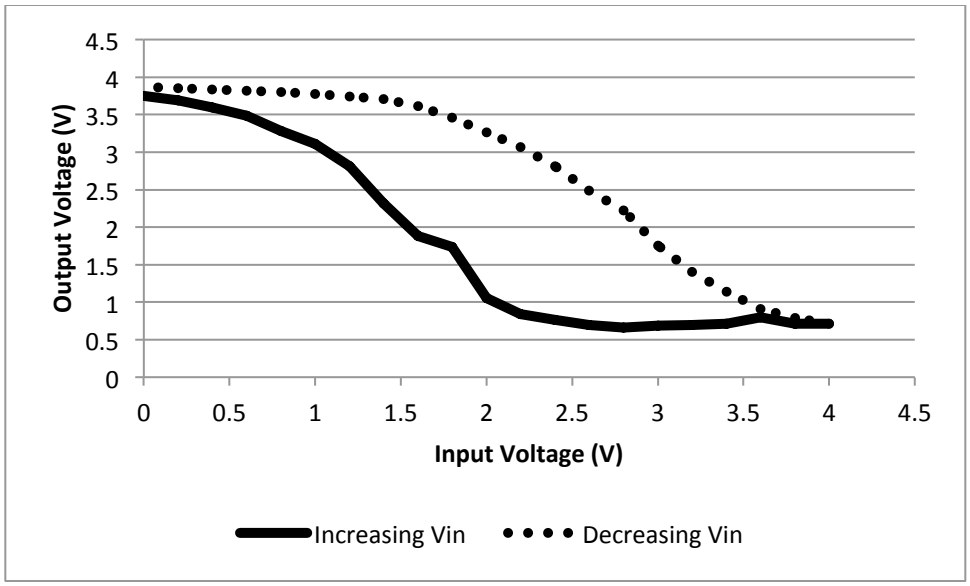

Figure 3.6 ND7 inverter output voltage hysteresis after -4 V poling voltage

This effect can also be seen in plots of the drain current through the transistor, which is calculated from the output voltage, the supply voltage, and the load resistance. Similar to Figures 3.5 and 3.6, Figures 3.7 and 3.8 show the current for the input voltage sequence after each poling voltage. The figures show that approaching a 4 V input voltage from 0 V resulted in a higher current than the trip back to a 0 V input. However, Figure 3.8 shows the impact that the -4 V poling voltage had by leading to a higher drain current for initial low voltage inputs. This is a result of the negative polarization reducing the threshold voltage of the transistor. Once the input voltage reached 4 V and was then decreased, the current plots followed the same approximate pattern of values regardless of the initial poling voltage used prior to the experiment since the magnitude of the positive voltage effectively switched the polarization in the ferroelectric layer.

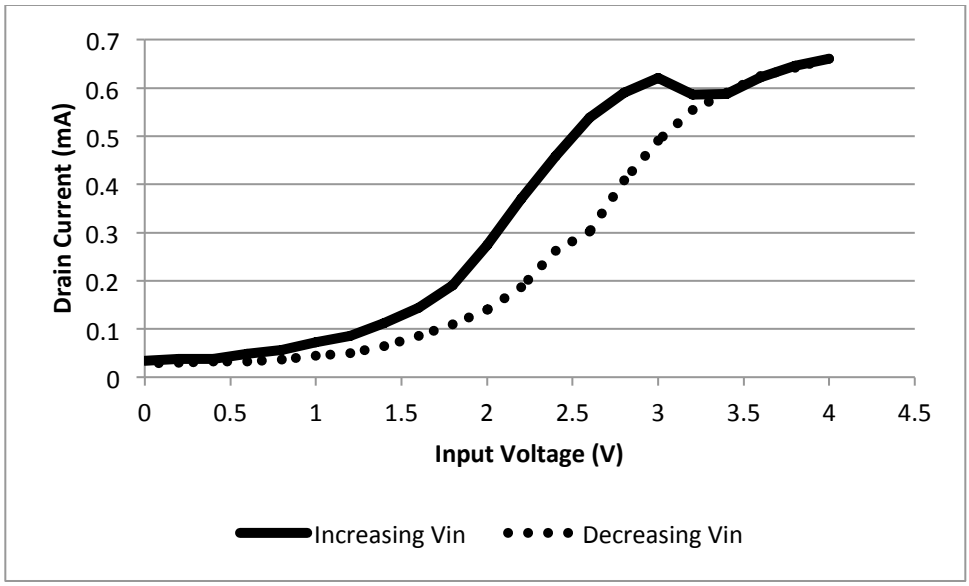

Figure 3.7 ND7 inverter current hysteresis after +4 V poling voltage

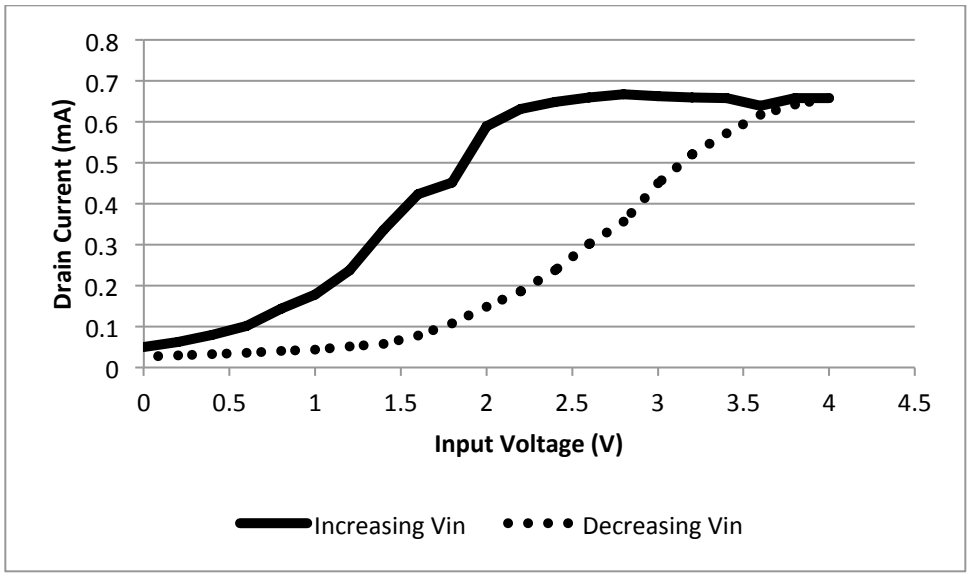

Figure 3.8 ND7 inverter current hysteresis after -4 V poling voltage

Using a load resistance of around 5 kΩ was sufficient to produce a usable inverter with the ND7 MFSFET, but this did not hold true when instead using an ND1 MFSFET. Due to the difference in channel aspect ratio between the two MFSFETs, shown in

Table 2.2, the ND1 transistor produced an overall lower magnitude of drain current. Figure 3.9 shows the case where the ND1 transistor was used in an inverter circuit with the supply voltage set to 4 V and the load resistance set to 5 kΩ. Rather than showing an explicit poling pulse applied to the circuit, a full range of negative and positive input voltages were used to see a more complete hysteresis effect.

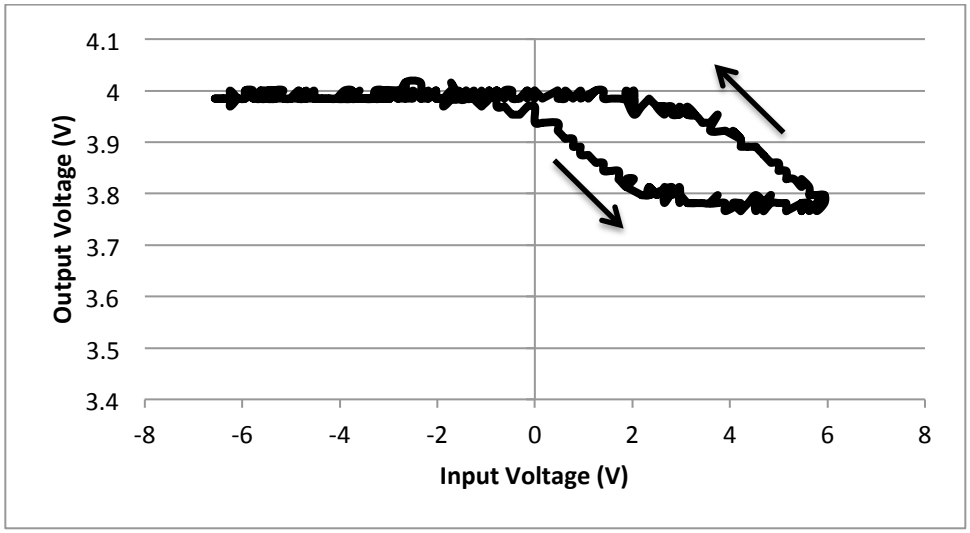

Figure 3.9 ND1 inverter output voltage with 5 kΩ load

The output voltage of the inverter could only reach a minimum of around 3.8 V, resulting in a circuit that did not actually produce inverter behavior. Once the load resistance was increased, however, the ND1 MFSFET became more usable in an inverter circuit. In Figure 3.10, the same circuit using a supply voltage of 4 V was tested with load resistances of 150 kΩ and 200 kΩ, and a more acceptable inverter was observed, though the output range was still smaller than what would typically be desired in an inverter.

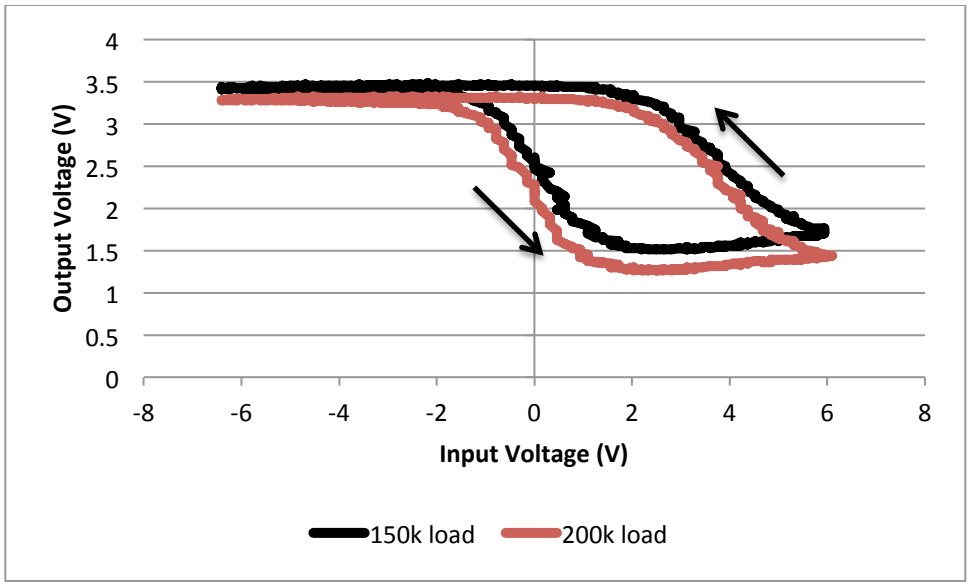

Figure 3.10 ND1 inverter output voltage for 150 kΩ and 200 kΩ load resistances

In addition to the varying voltage values that were applied and shown in the previous figures, square wave inputs were also tested to understand the effects of using waveforms of high and low voltages that might be seen in digital circuits. The impact of varying the load resistance and the input frequency of the waveform were specifically tested.

With a supply voltage of $V_{DD} = 4$ V, an input square wave of 4 V (peak-to-peak) was applied to the inverter circuit at varying resistances. The input wave had a 2 V offset so that it ranged from 0 V to 4 V rather than -2 V to 2 V. Because a square wave input was used rather than a constant voltage value, it was possible to immediately see the V_{OH} and V_{OL} values of the circuit by just observing the output waveform. The outputs supported the findings shown in Table 3.2, as higher resistances produced near-zero V_{OL} values, with the side effect of lower V_{OH} values than would be desired. Figure 3.11 shows the effect of the varied resistances on the output voltage.

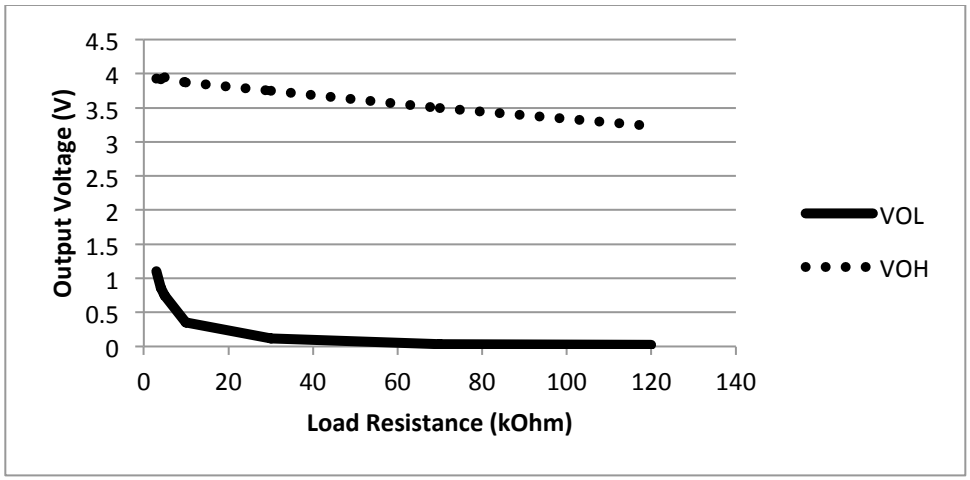

Figure 3.11 ND7 inverter output voltage vs. load resistance for 4 V square wave

Varying the input frequency had no significant impact on the behavior of the inverter. The supply voltage was 4 V, the input voltage was a square wave of 0 V to 4 V (peak-to-peak), and the load resistance was 5 kΩ. The results, shown in Figure 3.12, confirm the expected behavior of the circuit: the input frequency should not directly impact the output voltage of the circuit as long as the waveform is not transitioning so quickly that the circuit is unable to respond to the applied voltage level.

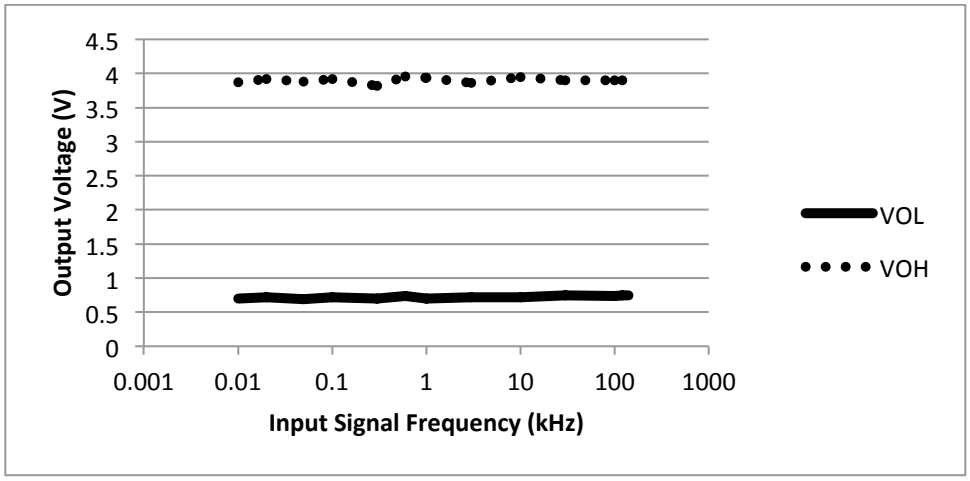

Figure 3.12 ND7 inverter V_{OL} and V_{OH} values for varying input frequency

The input voltage magnitude and offsets were varied such that the square wave ranged from 0 V to a positive V_{in} value, as well as a less traditional negative V_{in} value to 0 V. In all cases, a larger magnitude for V_{in}, whether it involved negative or positive voltages, produced more reliable inverter behavior than smaller input voltage magnitudes. This can be attributed to the fact that the smaller magnitude of waveforms such as 0 V to 1 V was insufficient to turn on the transistor. Figures 3.13 and 3.14 show the measurements taken when using negative and positive voltages, respectively. The input frequency was set to 1 kHz and the load resistance was 10 kΩ.

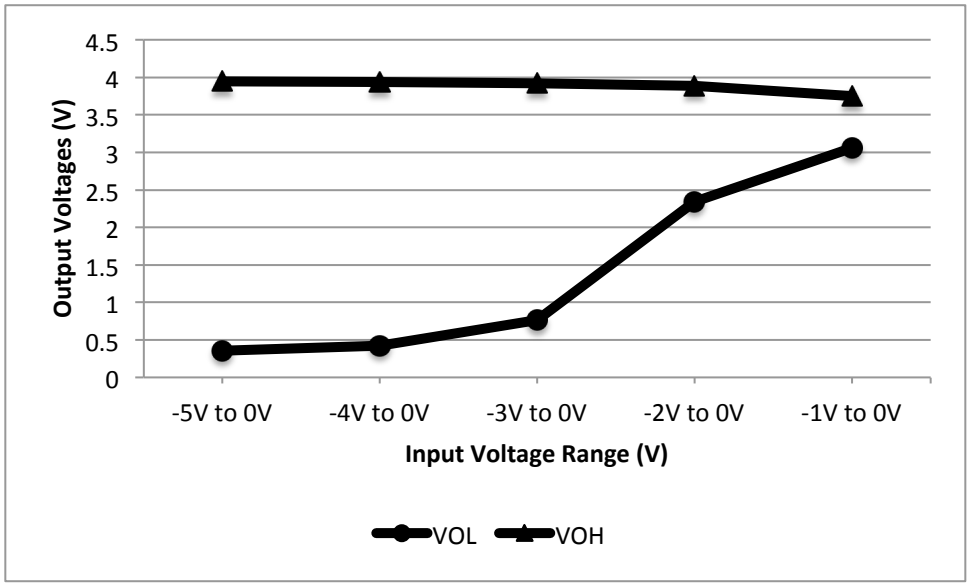

Figure 3.13 ND7 inverter output for negative input square waves [20]

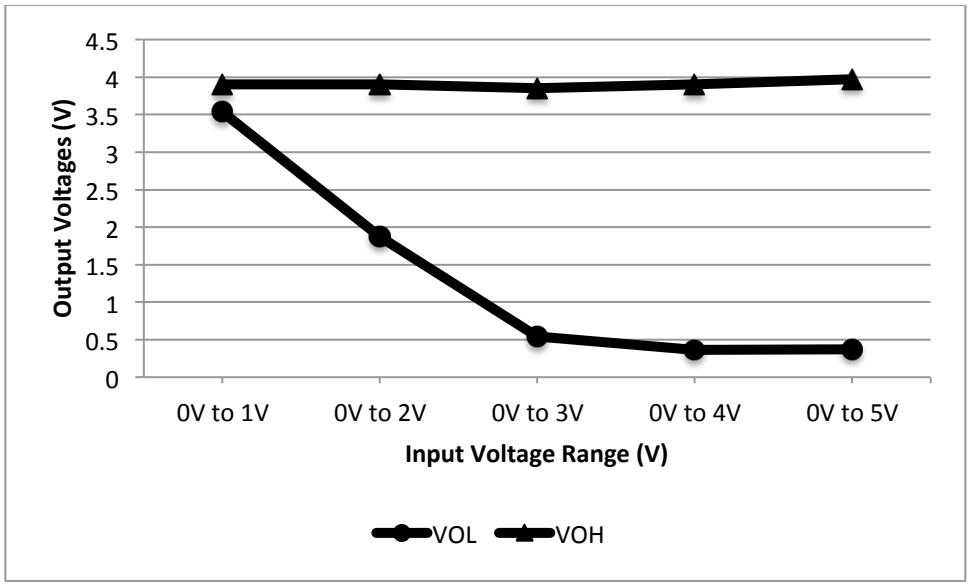

Figure 3.14 ND7 inverter output for positive input square waves

To see the full impact of varying the input voltage from a negative value to a positive value, the next figures show a summary of results for various load resistances and even a p-channel MOSFET load for each transistor over a range of inputs from -6 V to 6 V, while holding V_{DD} to 4 V. Figures 3.15 and 3.16 display the output voltages for the ND7 MFSFET. As suggested earlier, the ND7 MFSFET made a usable inverter for low resistances around 5kΩ and 15 kΩ. When the resistance was increased beyond 105 kΩ, the curvature of the voltage characteristic began to develop a straight-line slope for the increasing current as the gate voltage was increased. Of great interest is the behavior when the load resistance was replaced with a Zetex ZVP4105A p-channel MOSFET, which resulted in near-ideal output voltages for an inverter, at the cost of losing any hysteresis effect provided by the MFSFET. This can be attributed to the threshold voltage needed to turn on the PMOS transistor before it would conduct current,

in conjunction with the general capability of the PMOS transistor to provide better performance than a passive load resistor.

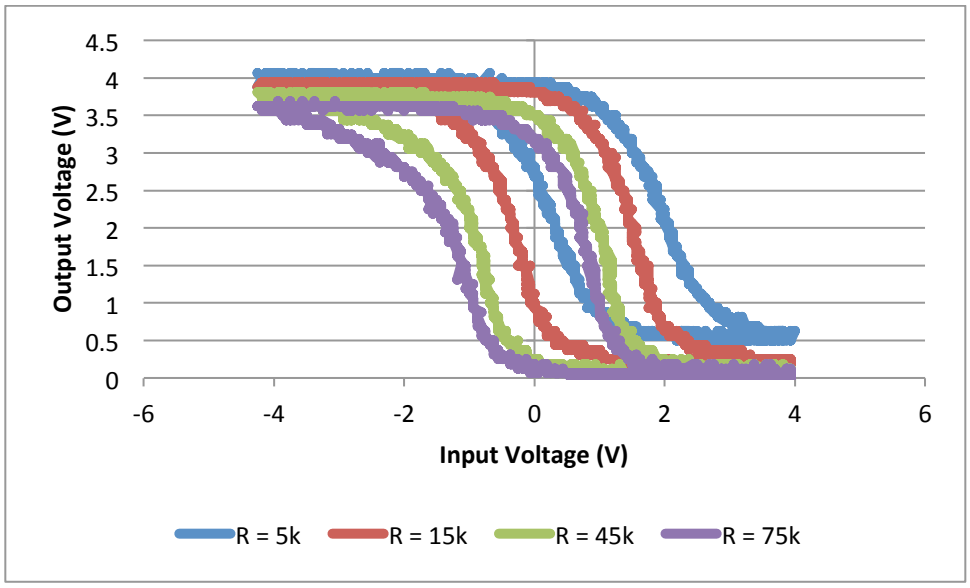

Figure 3.15 ND7 inverter full output voltage hysteresis, part 1 of 2

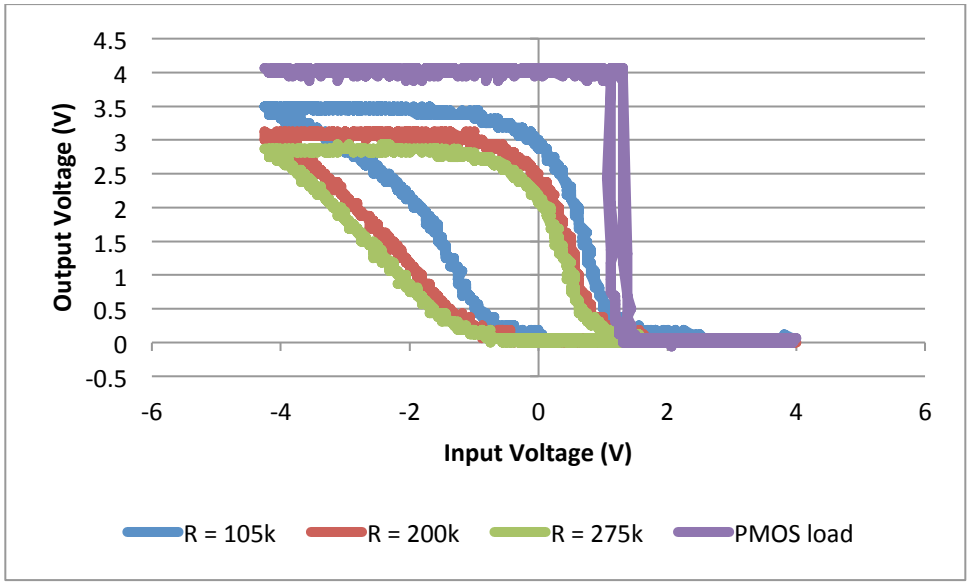

Figure 3.16 ND7 inverter full output voltage hysteresis, part 2 of 2

Figures 3.17 and 3.18 show the output voltages for the ND1 inverter for the same loads as the ND7 inverter.

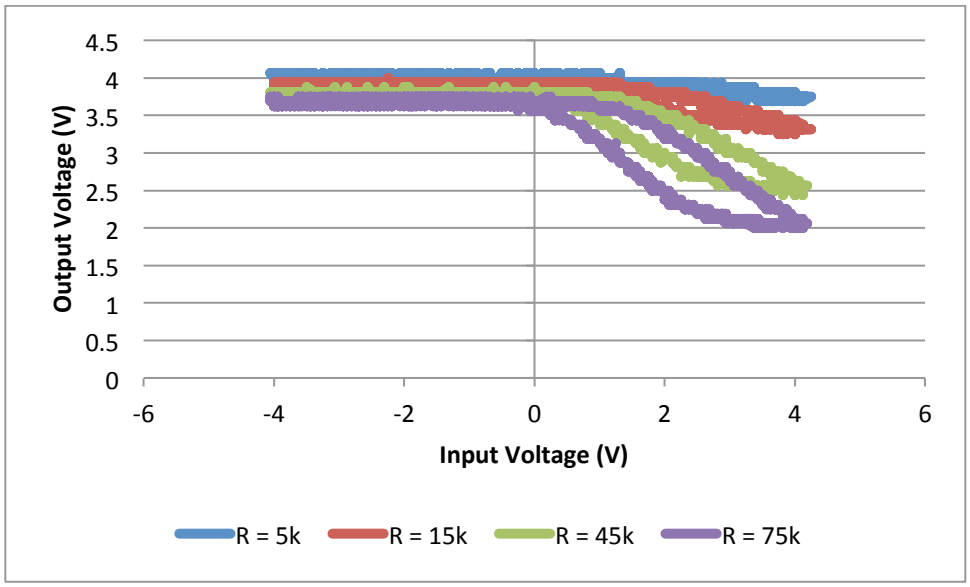

Figure 3.17 ND1 inverter full output voltage hysteresis, part 1 of 2

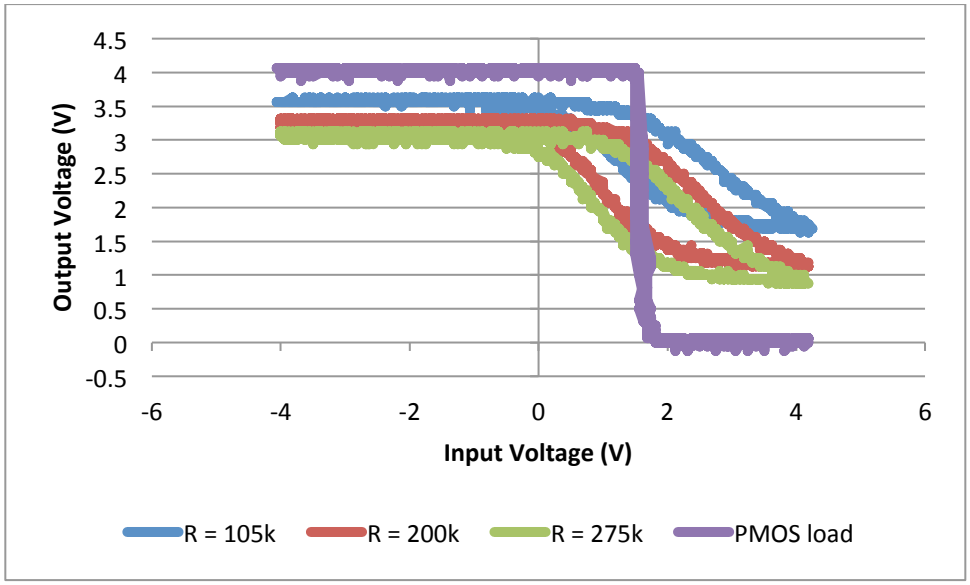

Figure 3.18 ND1 inverter full output voltage hysteresis, part 2 of 2

The ND1 MFSFET could not successfully act as an inverter until the load resistance was increased to around 200 kΩ. Similar to the ND7 inverter circuit, using a PMOS load resulted in an inverter producing near-ideal output voltages, but without hysteresis.

b. Switching Characteristics

Though passive-load inverters are typically undesirable for performance reasons [19], it was useful to investigate the switching characteristics of the MFSFET resistive-load inverter. In addition, the results shown in Figures 3.16 and 3.18 show that the PMOS load negated the hysteresis effects of the MFSFET, making resistive loads more useful to examine the unique properties of the MFSFET. Experiments were conducted to find the switching times—τ_{fall}, τ_{rise}, τ_{PLH}, and τ_{PHL}—in various configurations of this MFSFET inverter. Key configuration differences that were investigated were varying input signal frequencies, load resistances, and supply and input voltages.

The input signal used for testing the switching characteristics of the inverter was usually a square-wave input. Figure 3.19 shows the case where the input frequency was varied. The supply voltage was set to 4 V, the load resistance was 5 kΩ, and the input signal was 4 V peak-to-peak with a 2 V offset, making the signal range from 0 V to 4 V. The plot shows that there was some variation at each measurement, but there is no trend to suggest that the rise and fall times of the circuit had a dependence on the input frequency. This result is reasonable, since the input frequency should have only affected the amount of time elapsed after a fall or rise operation until the next one took place, and not the speed of a specific fall or rise operation directly [21].

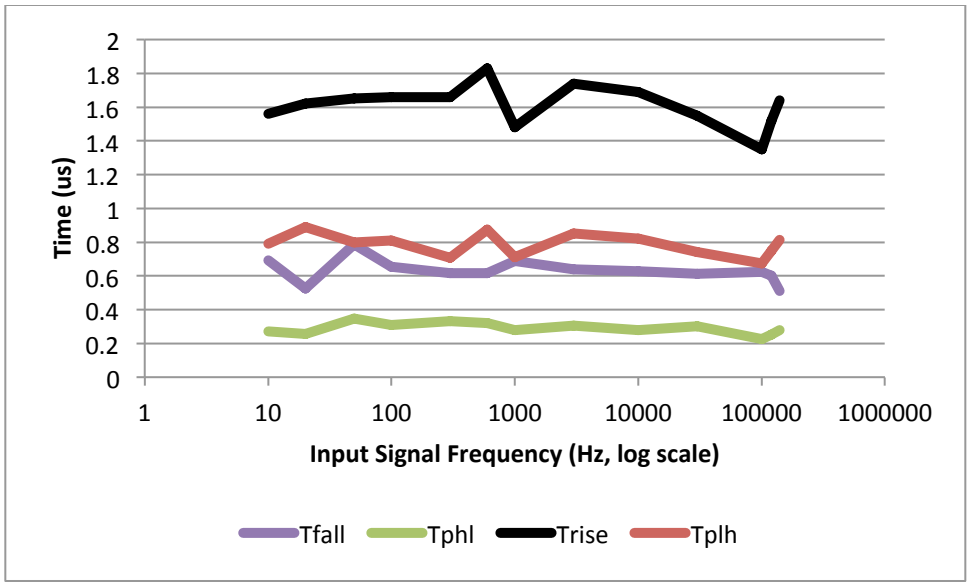

Figure 3.19 ND7 inverter rise and fall times for varying input frequency [21]

Just as Figure 3.11 displayed that varying the load resistance value had an impact on the output voltages observed in the inverter, Figure 3.20 shows that it also impacted the performance of the circuit. The input signal was again set to be a square wave of 0 V to 4 V, this time fixed at a frequency of 1 kHz; the supply voltage was held at 4 V. The most obvious effect shown was that the rise-related time delays were significantly increased as the load resistance was increased [21]. This would not be unique to the MFSFET inverter, but would apply to any resistive-load inverter.

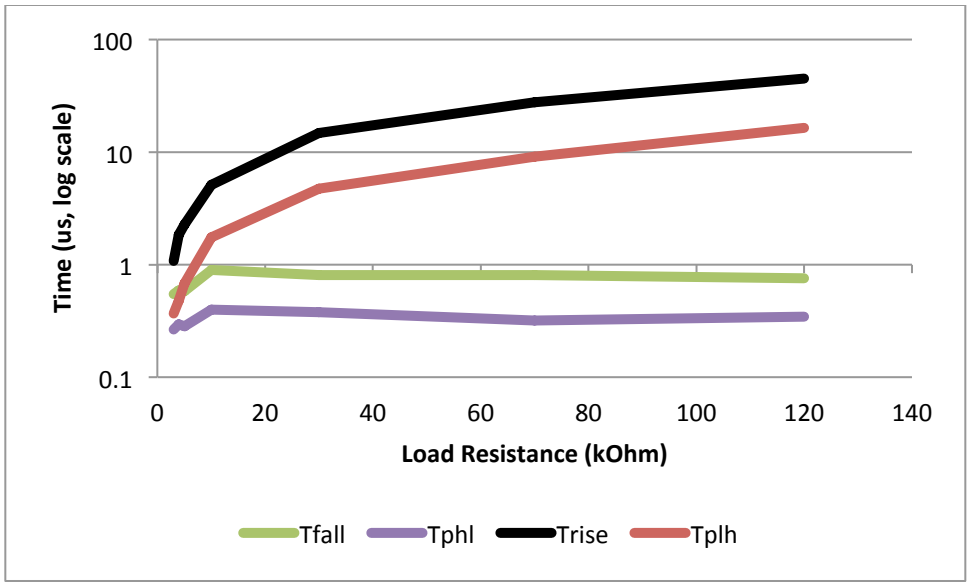

Figure 3.20 ND7 inverter rise and fall times for varying load resistance

The performance of the inverter circuit, however, was not significantly impacted when the supply voltage was the only varied parameter. In Figure 3.21, the provided data shows that there was no overwhelming effect on the output time delays when varying the supply voltage, although there was variance in the rise time but with no obvious trend. The input voltage was a 0 V to 4 V square wave at a frequency of 1 kHz, with a load resistance of 10 kΩ.

Further tests were conducted, holding the supply voltage constant at 4 V, load resistance at 10 kΩ, and input voltage frequency at 1 kHz. However, the input voltage amplitude was varied such that the square wave ranged from 0 V to a specified peak amplitude, as shown on the x-axis of Figure 3.22. In these tests, the input signal amplitude did impact the performance of the inverter. Specifically, the fall-related times demonstrated improvement as the input amplitude increased, while the rise-related times increased.

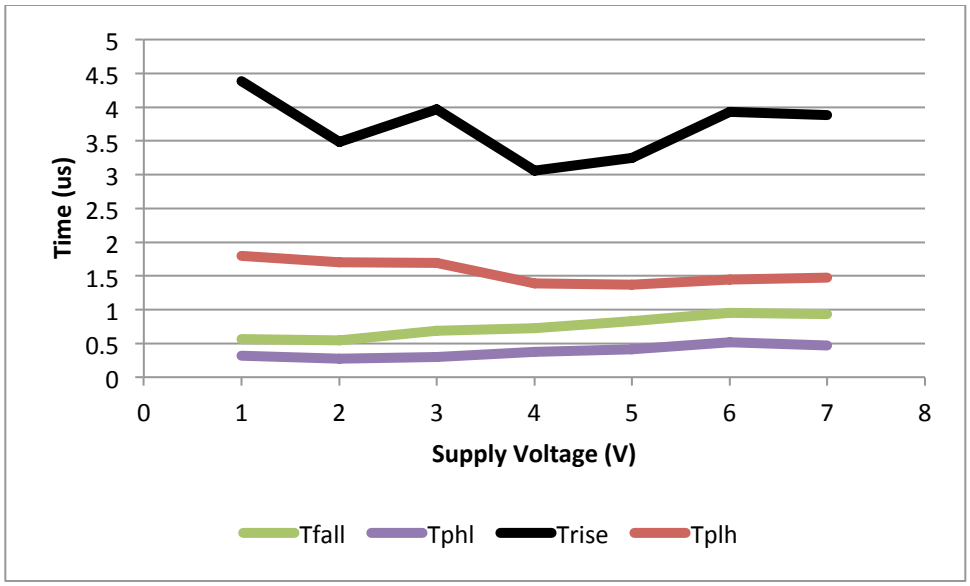

Figure 3.21 ND7 inverter rise and fall times for varying supply voltage

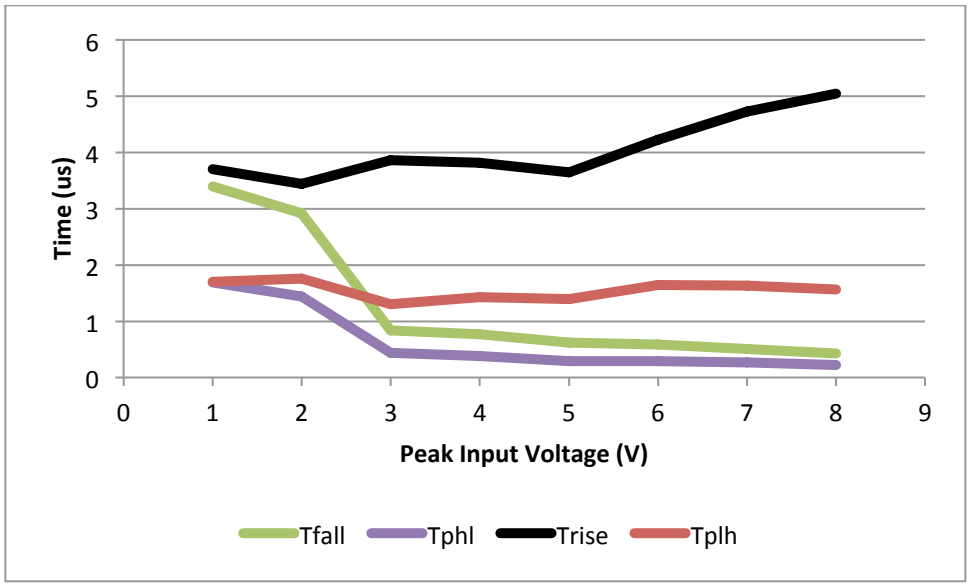

Figure 3.22 ND7 inverter rise and fall times for varying input signal amplitude

C. Contrasts to MOSFET-based Inverters

The MOSFET-based inverter, unlike the MFSFET variant, shows no hysteresis since it is based solely on the current node voltages applied to the transistor. Figure 3.23

and Figure 3.24 show output voltage characteristics for a MOSFET inverter, where the NMOS driver transistor was a Sanyo 2SK669 MOSFET. The loads tested and shown in these figures are the same as those in Figures 3.15 through 3.18, providing a comparison of the MOSFET behavior to that of the ND7 and ND1 MFSFETs.

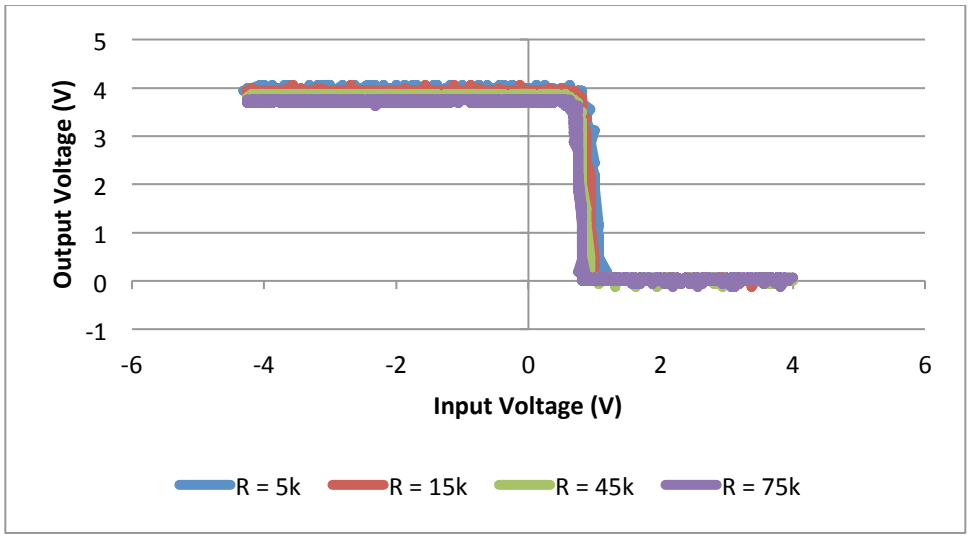

Figure 3.23 MOSFET inverter full output voltage hysteresis, part 1 of 2

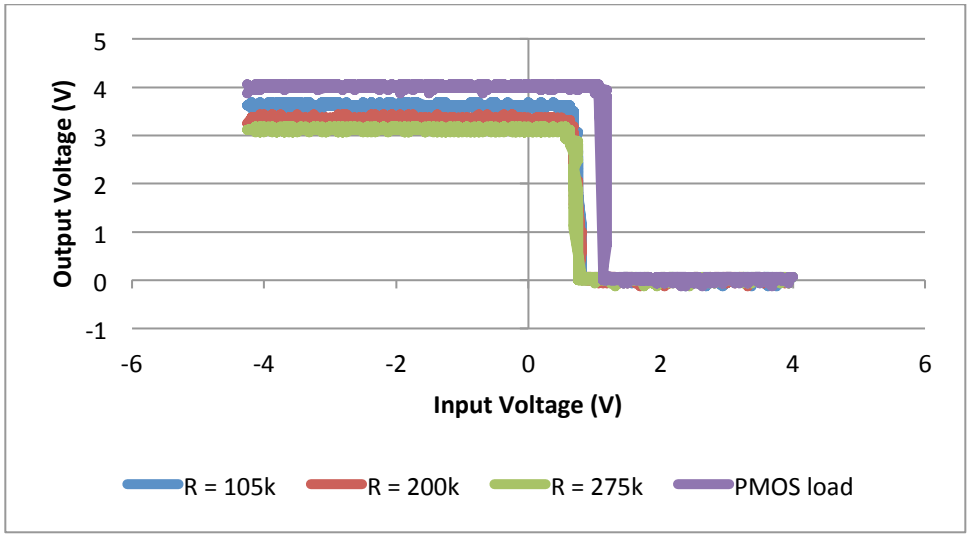

Figure 3.24 MOSFET inverter full output voltage hysteresis, part 2 of 2

Most of the load resistances produced suitable inverters, with the only disadvantages seen in resistances higher than 100 kΩ where the V_{OH} value decreased. As expected, the PMOS load transistor also produced a good inverter.

It is useful to be able to see a direct comparison of the MOSFET and MFSFET inverters within the same plots. In Figure 3.25, all three inverter output voltage characteristics are plotted for the 5 kΩ load resistor. This shows the advantages of the ND7 MFSFET and disadvantages of the ND1 MFSFET in comparison with the Sanyo 2SK669 MOSFET.

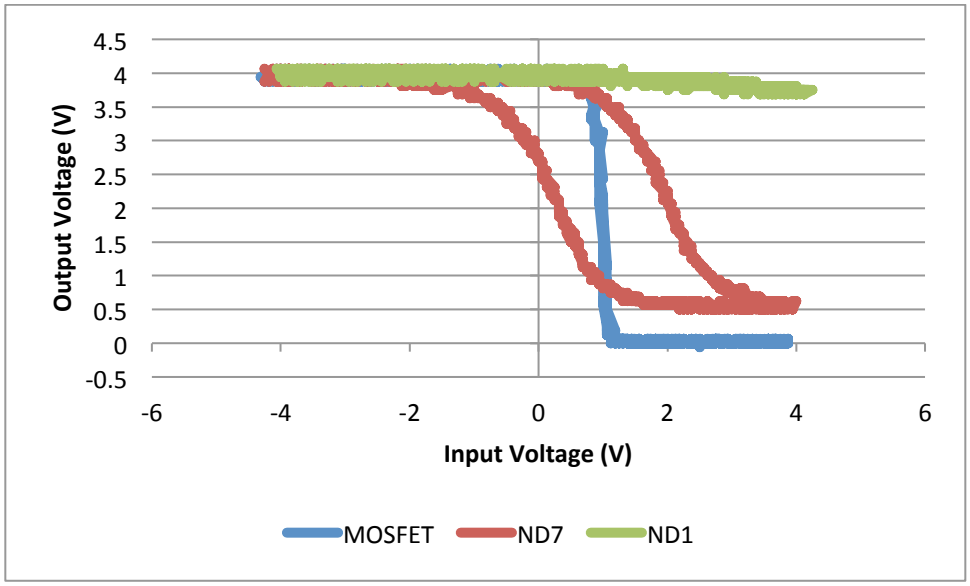

Figure 3.25 All inverter full output voltage characteristics for R_L = 5 kΩ

Another plot that shows the progression of the ND1 MFSFET into a good inverter with the ND7 MFSFET beginning to lose part of its curvature is presented in Figure 3.26. The ND1 and ND7 transistors appeared to have occupied completely different portions of the voltage plot, with the MOSFET overlapping both.

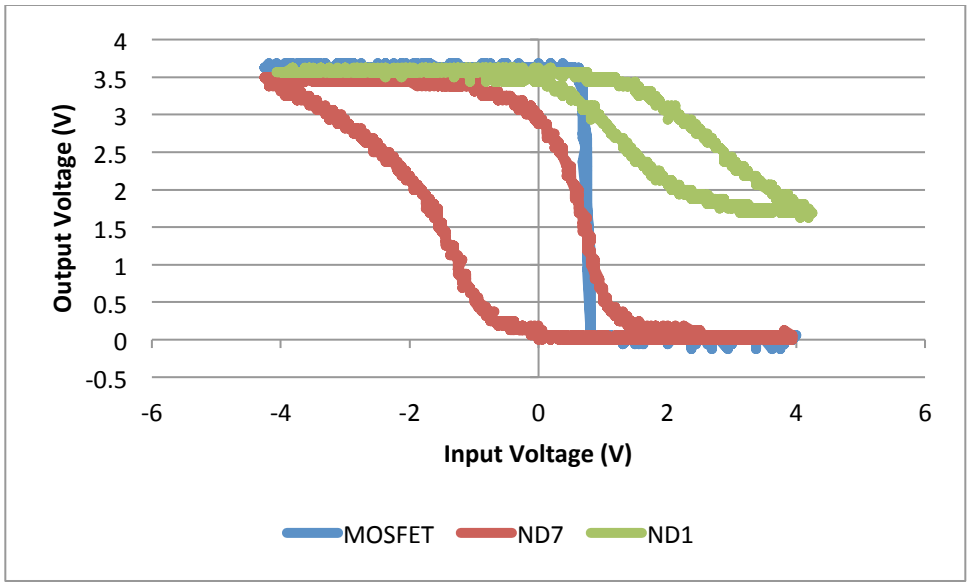

Figure 3.26 All inverter full output voltage characteristics for R_L = 105 kΩ

In Figure 3.27, the usage of a PMOS load device is shown to allow the MFSFETs to behave almost identically to the MOSFET, producing output voltages lacking hysteresis, and differing only in the threshold voltage of the circuit.

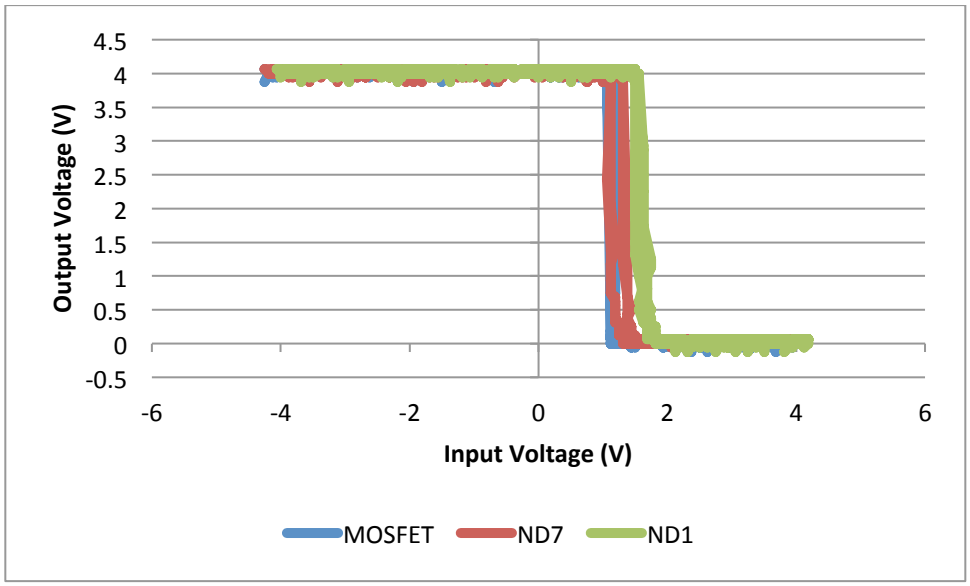

Figure 3.27 All inverter full output voltage characteristics for PMOS load

D. Analysis

The experimental data collected suggests that replacing the MOSFET in a simple inverter circuit makes it possible to create a more complex circuit when a ferroelectric transistor is used instead. In order for this to be useful, however, it must be possible for the circuit to reproduce the basic inverter behavior produced by the MOSFET-based circuit; this is possible, as shown in Figure 3.4 where two different resistive loads were utilized to create a circuit with clear inverter characteristics. Figures 3.5 through 3.8 also show inverter behavior, but further demonstrate the hysteresis effects that the presence of the ferroelectric material had in the circuit. The history of the input voltages influenced the VTC, and these figures show two voltage curves forming the hysteresis loop that, while unique, did retain inverter-like properties.

Though a simple circuit, the inverter is useful for research into digital circuits with MFSFETs. One advantage to studying the inverter and characterizing its behavior is the application of this same research to more complex circuits, such as the SRAM cell, which is just a set of two latched inverters. In addition, the results in this chapter show a device that essentially acts as two different devices in one package, with its behavior altered by the application of specific voltages on the gate of the MFSFET. The possibilities for utilizing this include creating a device whose behavior can be changed at runtime based on a voltage pulse. Applying poling voltages can also allow for devices with varied power and current requirements.

CHAPTER IV

THE FERROELECTRIC STATIC RANDOM ACCESS MEMORY CELL

Building on the digital logic inverter leads to the SRAM cell. The structure of the SRAM cell is essentially that of two inverters, with a latch connecting the drain of each transistor to the gate of its counterpart. Similar to the inverter, two common forms of the SRAM cell are shown in Figure 4.1, where load devices are either resistors or PMOS transistors, while the driver transistors are NMOS transistors. Another circuit layout, though not studied here, consists of the loads being depletion-mode NMOS transistors, with each load transistor's gate connected to its own source [19].

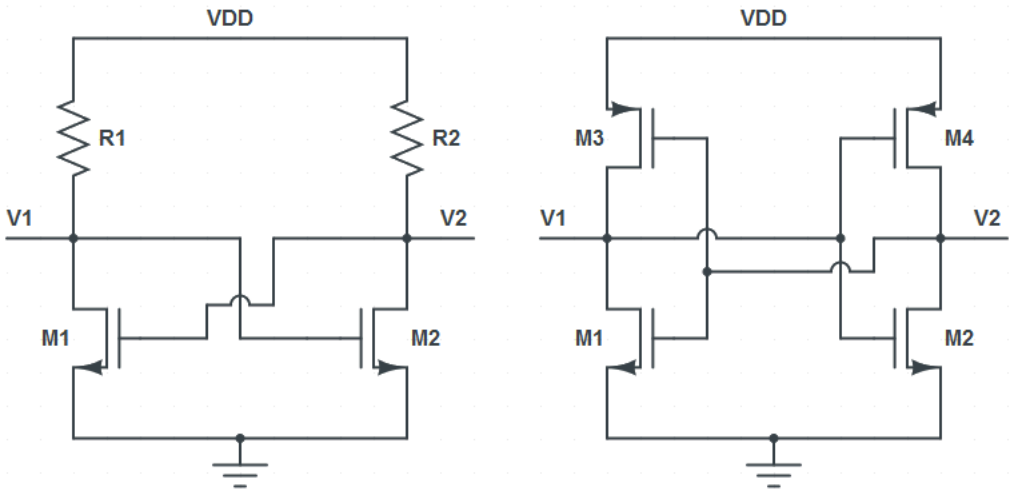

Figure 4.1 SRAM circuit layouts – resistive load (left) and PMOS load (right)

Much of the behavior of the SRAM circuit directly relates to the theory of the inverter circuit because the SRAM is a set of two latched inverters. Because the two inverters are latched, with the output of one driving the input of the other, the SRAM cell contains two values, allowing it to store a desired value and its complement. Replacing one or both of the NMOS transistors with an MFSFET not only produces an SRAM cell capable of operation as if it only contained traditional MOSFETs, but produces a unique SRAM cell capable of nonvolatile memory storage.

A. Effects of the Ferroelectric Transistor

While studying the SRAM cell, various circuit configurations were used, and therefore different device types are shown in the following sections of this chapter. The ND7 MFSFETs were heavily used in early static and retention characterization of the circuit; later current analysis with the Radiant Precision Premier was focused primarily on the newer ND1 MFSFETs. However, the results given are applicable to any of the Radiant transistors, with consideration for the current capabilities and channel dimensions of each one. Use of traditional MOSFETs included the Sanyo 2SK669 n-channel enhancement-mode MOSFET, Zetex ZVP4105A p-channel enhancement-mode MOSFET, and National Semiconductor CD4007 integrated circuit containing both n- and p-channel enhancement-mode MOSFETs.

To simplify the testing process, the access transistors that are typically connected to the bit lines of the cell were not included here, as the design process requires careful sizing of transistor devices and this could present issues due to the presence of the MFSFETs. Voltages were applied to the drain of the left side transistor, which was connected to the gate of the right transistor; this node is referred to in this thesis as V1,

while the equivalent node on the other side of the circuit is referred to as V2, as designated in Figure 4.1.

a. Logic Levels and Static Characteristics

As with the characterization of the inverter circuit, it was important to ensure that the SRAM circuit containing MFSFETs could, at a minimum, behave in a manner similar to a typical SRAM circuit. To verify this, experiments were conducted both with and without MFSFETs to characterize the basic behavior of the cell. In Figure 4.2, the write operations and results are shown from an SRAM cell containing the 2SK669 transistors as the driver transistors and the ZVP4105A p-channel transistors as the loads, with V_{DD} set to 4 V.

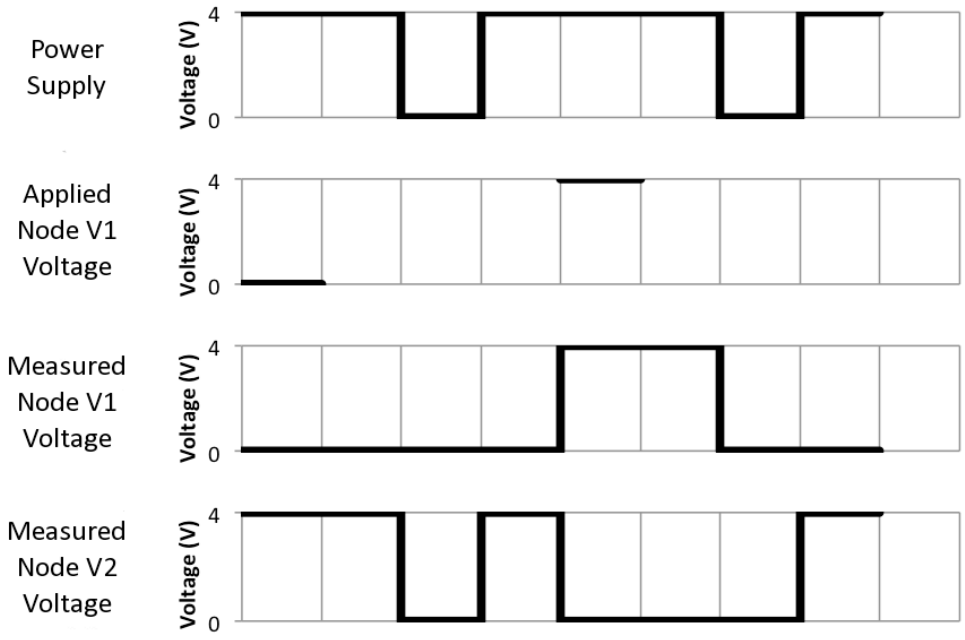

Figure 4.2 Write operations of MOSFET-based SRAM

When a voltage of 0 V was directly applied to node V1, this was considered to be writing a logic "0" into the cell, and applying a voltage of 4 V was considered writing a logic "1" value. With no voltage directly applied to V1, the value read at V1 was interpreted as the bit value contained by the memory cell. In Figure 4.2, it is shown that a logic "0" or "1" could be successfully written to the cell, and that value retained at node V1 even after being removed. However, once the supply voltage V_{DD} was removed from the circuit by setting it to 0 V, any logic "1" value was lost, and node V1 read 0 V when V_{DD} was restored; likewise, node V2 read 4 V when V_{DD} was restored.

It was necessary to observe how the circuit behaved when replacing one or both of the driver transistors with MFSFETs. The left side driver transistor was replaced with an ND7 MFSFET, and the steps shown in Figure 4.2 were repeated, but the observed results were the same as the SRAM circuit without MFSFETs. The test was repeated again, with both driver transistors replaced with ND7 MFSFETs, and the results remained the same as those recorded in Figure 4.2.

Another test was conducted where only the right side driver transistor, M2, was replaced with a MFSFET, and the left side driver was a MOSFET; in this case, as shown in Figure 4.3, a unique result was shown in that the ferroelectric properties of the transistor caused a voltage of 4 V to be seen at node V1 after removing and restoring the supply voltage for both cases of writing a logic "0" or "1" value.

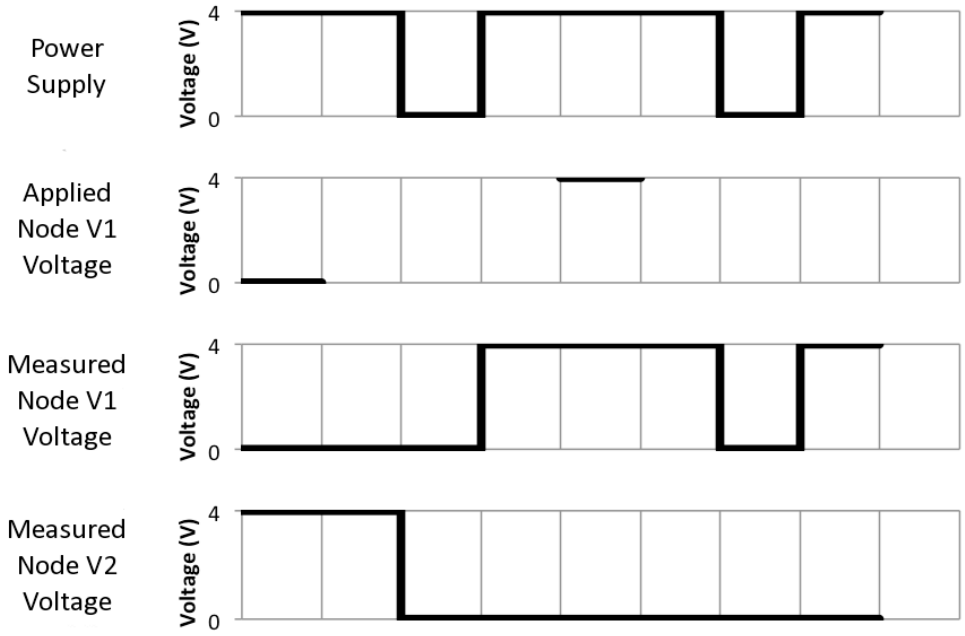

Figure 4.3 Write operations of SRAM with MFSFET at M2

Because of the unique properties of the MFSFET, the range of inputs was extended to also include negative voltages. This produced behavior that allowed the SRAM to behave as a nonvolatile memory cell. Both driver transistors were replaced with ND7 MFSFETs, still using PMOS transistors as loads, and a series of steps was taken similar to those shown in Figures 4.2 and 4.3. In Figure 4.4, these steps and their observed results are shown.

Based on the observations made in Figure 4.4, one can see that applying a negative voltage to the SRAM cell acted as a method to write a nonvolatile logic "1" value. Once the power was removed and then restored, the voltage at node V1 was measured at 4 V, the same as if a logic "1" had been applied with a positive 4 V pulse, but without removing the supply power. One important note is that this required cycling the supply power in order to actually see this stored value, as node V1 only read 0 V

immediately after applying and removing the -4 V input. Further tests were conducted to see how long the cell could retain the stored value.

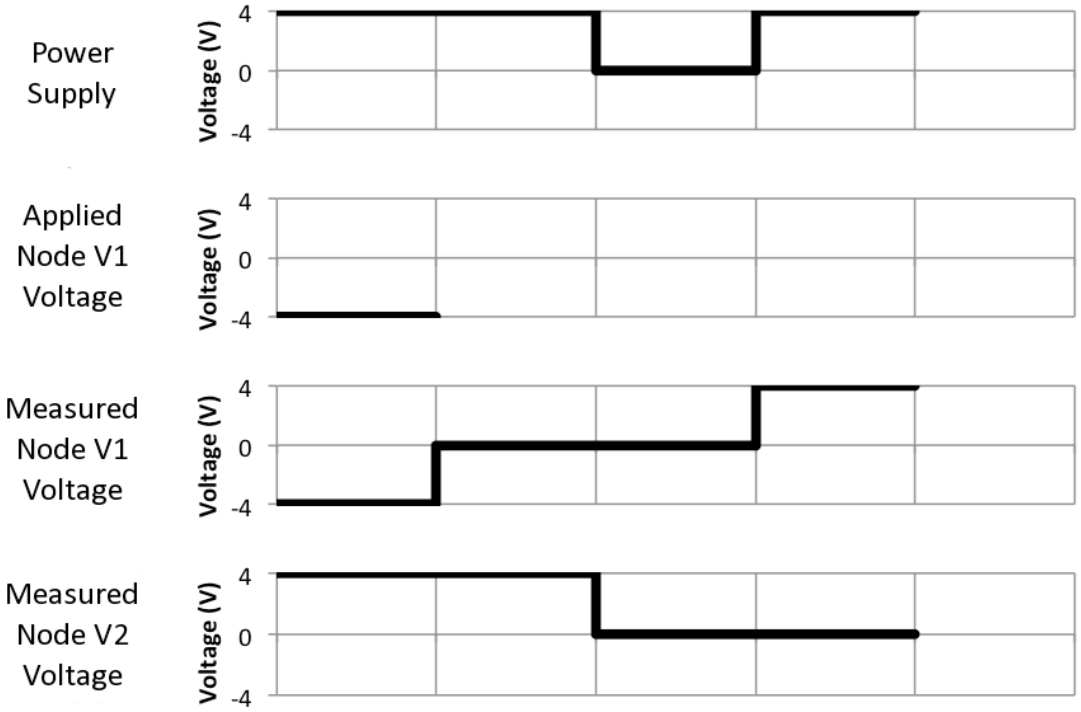

Figure 4.4 Negative write operation for SRAM with MFSFETs

As shown in Table 4.1, the SRAM cell was able to retain a logic "1" value for both measurements of 10 minutes and 30 minutes when a -4 V input voltage was applied to node V1. The written "1" was not retained for the period of 71 hours, shown at the bottom of the table. Also worthy of mention is the additional test that was conducted immediately following the reading taken after 30 minutes of the supply voltage being removed; when the supply voltage was reapplied and node V1 still retained the stored "1" value, the supply voltage was again removed and immediately reapplied, with 0 V showing at node V1. This shows that while the SRAM did show nonvolatile properties,

the reading of the cell by cycling the power again destroyed the stored value. Later test results will show estimated retention times of the written "1" value, based on the magnitude of the applied voltage.

Table 4.1 Retention time analysis of SRAM for -4 V input

Operation	V_{DD}	Voltage applied at V1	Voltage measured at V1	Voltage measured at V2
Apply -4 V	4 V	-4 V	-4 V	4 V
Remove input, keep power	4 V	(not applied)	0 V	4 V
Remove power (for 10 minutes)	0 V	(not applied)	0 V	0 V
Reapply power, read node voltages	4 V	(not applied)	4 V	0 V
Apply -4 V	4 V	-4 V	-4 V	4 V
Remove input, keep power	4 V	(not applied)	0 V	4 V
Remove power (for 30 minutes)	0 V	(not applied)	0 V	0 V
Reapply power, read node voltages	4 V	(not applied)	4 V	0 V
Remove power	0 V	(not applied)	0 V	0 V
Reapply power, read node voltages	4 V	(not applied)	0 V	4 V
Apply -4 V	4 V	-4 V	-4 V	4 V
Remove input, keep power	4 V	(not applied)	0 V	4 V
Remove power (for 71 hours)	0 V	(not applied)	0 V	0 V
Reapply power, read node voltages	4 V	(not applied)	0 V	4 V

The SRAM testing shown prior to this consisted of input voltages of 4 V, whether positive or negative. Voltages of varying magnitudes were applied to determine the range of useful inputs. Starting at -4 V, which was already shown to display nonvolatile

behavior, the applied voltage was incrementally increased and tested. When the supply voltage was removed, it was immediately reapplied, rather than waiting for a specified amount of time. The threshold of useful voltages was found to be at -1.2 V, as this input could produce a logic "1" upon cycling the supply voltage off and back on. Measurements taken with -1 V and -1.1 V inputs showed no signs of non-volatility.

Testing showed that higher magnitudes of negative voltages applied to node V1 resulted in longer retention times of the written value, with -5 V producing a retention time of over 20 hours. Table 4.2 lists a summary of the retention times seen for various applied voltages.

Table 4.2 Retention times for negative applied voltages

Applied Voltage	Approximate retention time
-1.2 V	25 seconds
-1.3 V	5 minutes
-1.4 V	15 minutes
-1.5 V	20 minutes
-2 V	4 hours
-5 V	20 hours

b. Current Measurements

Current measurements for the SRAM circuit were primarily focused on the resistive load variation of the circuit, in order to simplify analysis. Two methods of measurement were used: the Precision Premier tester was used to directly measure the current in one or both sides of the circuit, or an oscilloscope was used to measure the voltage at the V2 node that could then be translated into a current calculation. It should

be noted that the results below focus on circuits built with the ND1 transistor rather than the ND7 transistor.

Because the SRAM circuit is a set of two latched inverters, analysis of the gate and drain voltages of the M2 driver transistor is almost identical to analysis of the input and output voltage of the inverter. Applying a voltage at node V1 should show an inverted voltage value at node V2, and this was the case for the SRAM configuration described in Figure 4.5. This particular SRAM configuration used load resistances of 100 kΩ with V_{DD} = 2 V and the node V1 voltage as a triangle wave measuring 12 V_{pp} with no offset voltage at 1 Hz.

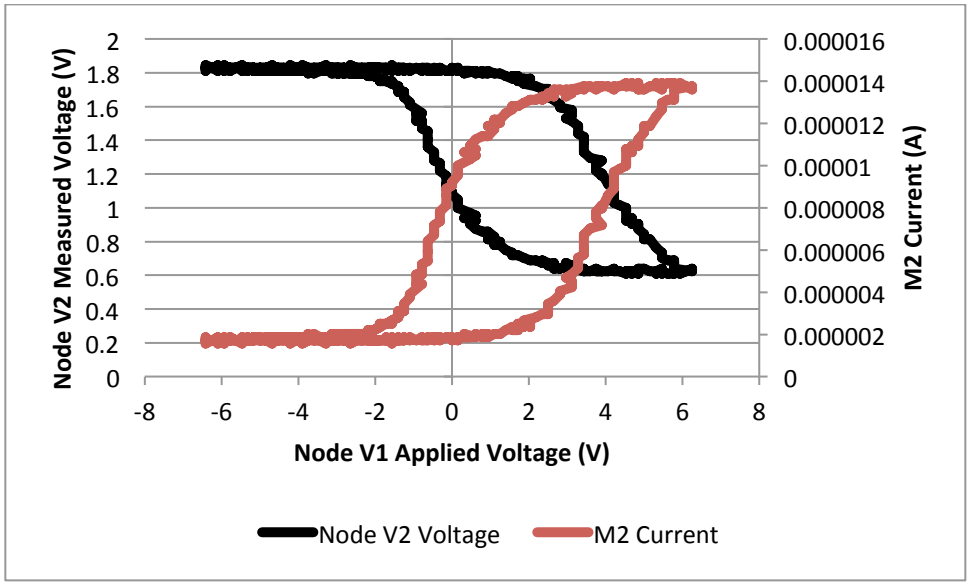

Figure 4.5 SRAM node voltages and current for V_{DD} = 2 V and $R_1 = R_2 = 100$ kΩ

The resulting node V2 voltage is not unexpected, as it follows the same pattern as the MFSFET inverter, generating an inverted output with hysteresis. The current in the M2 transistor, calculated by considering the supply voltage, load resistance, and node V2

voltage, is also similar to the characteristic current hysteresis curves shown earlier for the standalone MFSFET device and for the MFSFET inverter. More interesting results, however, were found when increasing the supply voltage beyond 2 V up to 6 V. Figure 4.6 shows the M2 transistor drain current for 5 distinct values of V_{DD}, using the same load resistance and input signal at node V1. There is a noticeable difference seen for negative gate voltages on M2, with current flowing through the transistor as the applied gate voltage became more negative.

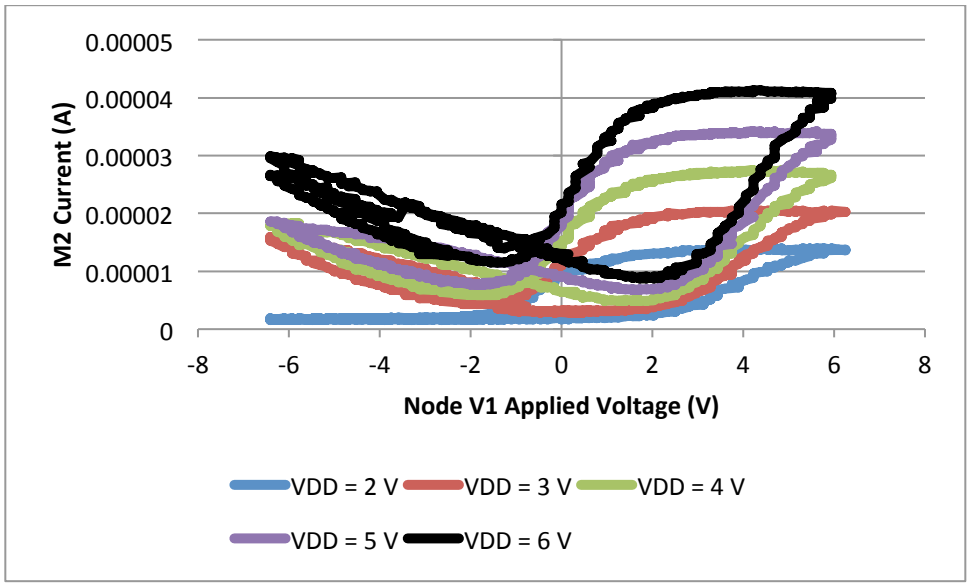

Figure 4.6 SRAM M2 transistor current for varied supply voltages

The effect shown for negative voltages on the gate of M2 was not seen in the inverter circuit for negative input voltages; this suggests that the combination of negatively polarizing the gate on M2 and the presence of the opposing inverter is the cause of the increased current on the negative input range. Because node V1 is the connection of the drain of M1 with the gate of M2, applying negative voltages to V1

caused the drain of M1 to become lower than the source of M1, effectively reversing the terminals.

Another view of this unique effect is shown in Figure 4.7. Rather than showing the hysteresis loop as the node V1 voltage was varied, it instead shows a time-varying plot of the node V1 voltage and the resultant current through M2 for V_{DD} of 2 V and 6 V. While the current for the V_{DD} = 2 V remained flat for most of the negative voltage input range, there are sharp points showing drastic increases in the current that directly correspond to the extreme minimum values of the applied voltage at V1 for the case where V_{DD} = 6 V.

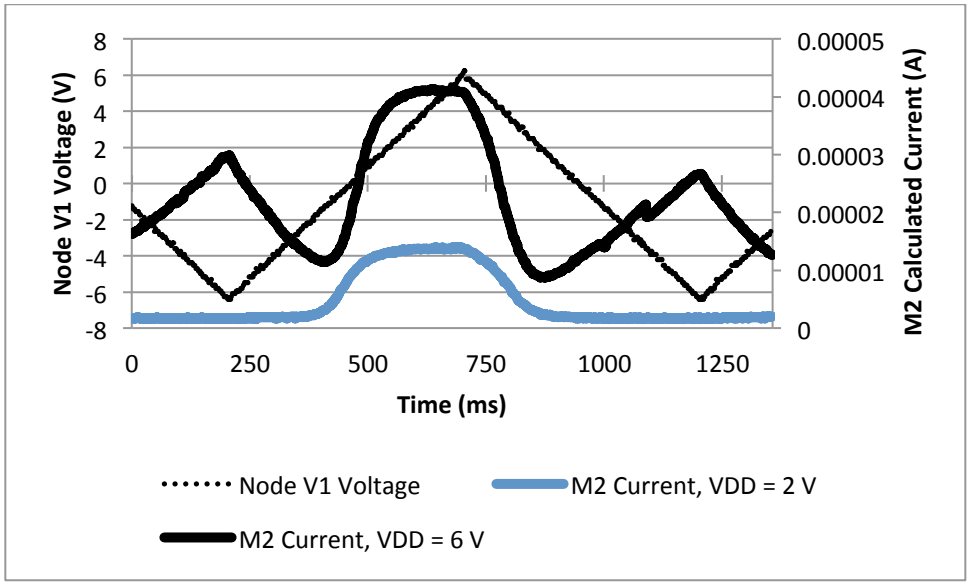

Figure 4.7 Time-based representation of current in ND1 SRAM for varying V_{DD}

Figure 4.8 shows the case where positive and negative poling pulses of 6 V were applied prior to testing; during the test, V_{DD} was set to 4 V, the load resistance was 100 kΩ, and the input square wave was 4 V_{pp} with 2 V offset at 2 kHz. The overlap of

the two current plots shows that no significant difference was made on the circuit after time had elapsed since the poling pulse had been applied. This is due to the fact a square wave consisting only of 0 V and 4 V was used, preventing any negative polarization. Any effects of the poling pulse would likely only be seen just as the function generator was connected to the circuit and would be negligible after the voltage had cycled between 0 V and 4 V.

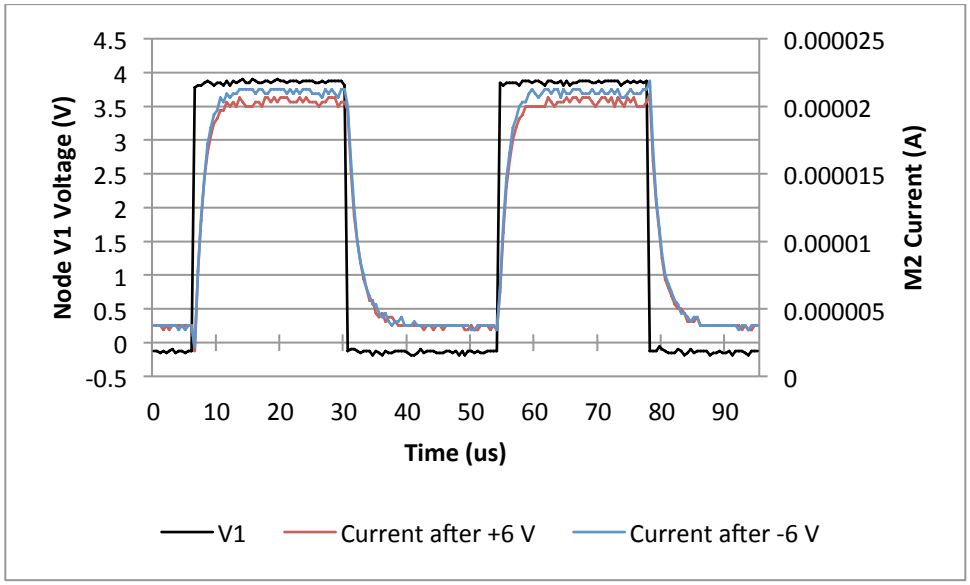

Figure 4.8 ND1 SRAM with square wave input after positive and negative poling pulses

The Precision Premier tester was used to measure the current flowing through each side of the SRAM circuit, as well as the current combined from both sides. Using the Precision tester's pulse voltage before testing was not sufficient to polarize the transistors, however, because this pulse was only applied to the M2 transistor since it was the transistor whose gate was connected to the tester. To address this, a testing process was followed where both MFSFETs were removed from the SRAM circuit, poled with a

voltage applied to the gate by a separate power supply, and then reinserted into the SRAM circuit. In Figure 4.9, the currents found in each side of the circuit are plotted as a function of the applied gate voltage, with the transistors having had a 6 V poling voltage pulse applied before the test. The load devices were 100 kΩ resistors, and the input voltage started at 0 V, was increased incrementally to 4 V, and then returned in the same manner back to 0 V. The value of V_{DD} was 1 V.

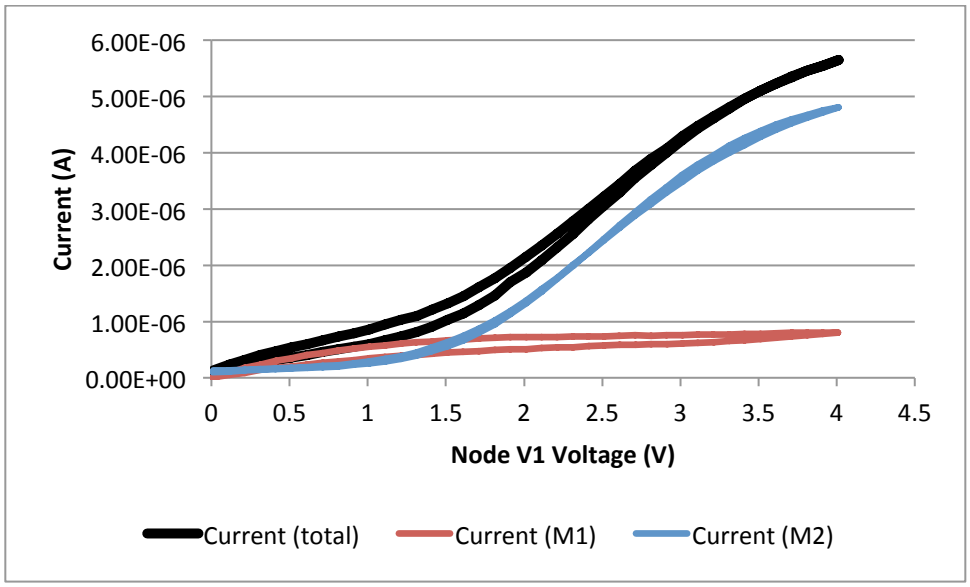

Figure 4.9 SRAM ND1 transistor drain currents for 0 V to 4 V after 6 V pulse

To obtain the data in Figure 4.9, three separate tests were actually conducted. In one test, only the current through the left side of the SRAM circuit was measured, and a second test was conducted to measure the current in the right side assuming the same input data. The Return connection of the Precision tester was attached to the source terminal of the transistor whose current was being measured. The final test measured the current flowing from both branches of the SRAM circuit, merged together at the sources

of the transistors. The shape of the current plots in Figure 4.9 show that the sum of each transistor's current was equivalent to the total current measured in the circuit. It should be noted that because three separate tests were conducted to obtain this data, the poling voltage was reapplied to each transistor between each test.

The series of tests shown in Figure 4.9 were repeated with an identical circuit setup, but with the poling voltage set to -6 V before each test rather than 6 V. These results, shown in Figure 4.10, indicate that the negative poling voltage introduced larger currents into the circuit, which is consistent with the behavior of an individual MFSFET, as it will show a larger current for a given input if poled with a negative voltage rather than a positive voltage.

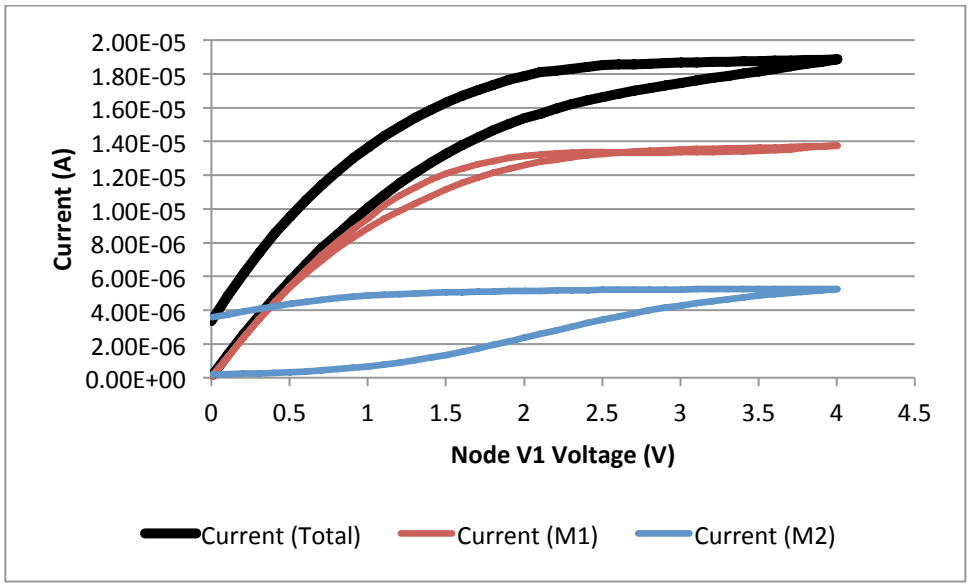

Figure 4.10 SRAM ND1 transistor drain currents for 0 V to 4 V after -6 V pulse

c. Switching Characteristics

The SRAM cell did prove to be an interesting circuit for investigating the unique aspects of the MFSFET, due to its current characteristics and the potential for a nonvolatile memory. There are also interesting aspects of the switching characteristics of the circuit, measuring the behavior of the circuit in response to an applied voltage.

A simple comparison between MOSFET- and MFSFET-based SRAM circuits is shown in Table 4.3, where V_{DD} = 4 V and a 100 kΩ resistive load was used in both circuits, with an additional test on the MFSFET-based SRAM using 275 kΩ. In these tests, the ND1 MFSFET was used, along with the CD4007 chip containing both NMOS and PMOS transistors. The fall time of the MOSFET-based circuit is significantly better than the MFSFET-based circuits. Even with the large difference in resistive loads for the MFSFET testing, the fall time in both circuit setups remained close. The rise time, however, was more than twice as slow for the 275 kΩ resistive load setup than it was for the 100 kΩ resistive load, which is not surprising given the change in resistance. The MOSFET-based circuit actually performed slightly slower in rise time than the equivalent MFSFET-based circuit using the 100 kΩ load, but the difference was not significant.

Table 4.3 Resistive load SRAM timing characteristics

Transistor	Resistive load	τ_{fall} (s)	τ_{rise} (s)
MOSFET	100 kΩ	7.50E-07	3.40E-05
ND1 MFSFET	100 kΩ	2.10E-05	2.80E-05
ND1 MFSFET	275 kΩ	2.60E-05	8.00E-05

Further tests, detailed in Table 4.4, were conducted that not only replaced the resistive loads with PMOS loads, but also explored the effect of poling pulses on the

MFSFET-based SRAM cell for varied V_{DD} values. These tests also used the ND1 MFSFET and the CD4007 for the MOSFET circuit, and used the PMOS transistors found on the CD4007 chip in all cases.

Table 4.4 PMOS load SRAM timing characteristics

Transistor	V_{DD} (V)	Square Wave Applied at V1 (V)	Polarization Pulse (V)	τ_{fall} (s)	τ_{rise} (s)
MOSFET	2	0 to 2 V	none	1.20E-06	3.25E-06
MOSFET	3	0 to 3 V	none	4.30E-07	7.10E-07
MOSFET	4	0 to 4 V	none	2.80E-07	3.90E-07
ND1 MFSFET	2	0 to 2 V	+6 V	2.11E-04	4.00E-06
ND1 MFSFET	3	0 to 3 V	+6 V	7.30E-05	1.50E-06
ND1 MFSFET	4	0 to 4 V	+6 V	5.50E-05	1.50E-06
ND1 MFSFET	2	0 to 2 V	-6 V	5.30E-05	4.00E-06
ND1 MFSFET	3	0 to 3 V	-6 V	5.20E-05	1.50E-06
ND1 MFSFET	4	0 to 4 V	-6 V	5.10E-05	1.50E-06

Just as the resistive load MOSFET-based SRAM performed better in fall time than the MFSFET-based circuits did, the PMOS load variations of the SRAM behave in the same manner. However, these results show that a negative poling voltage applied to the MFSFETs prior to testing can decrease the fall time if V_{DD} and the magnitude of the applied square wave are small enough. When the size of the square wave was only 2 V, there was not enough of a voltage to fully switch the polarization of the transistor, meaning that the effects of the -6 V poling pulse remained present in the circuit. In this case, the application of a negative poling pulse yielded a fall time of 53 μs and a positive poling pulse yielded a fall time of 211 us. The circuit performed four times faster when using a 2 V square wave along with negative pulses. The effect was not as large when the magnitude of the square wave increased to 4 V, with the negative pulse only resulting

in a 51 μs fall time in comparison to the 55 μs seen after the positive pulse. This shows that the 4 V square wave was great enough to influence the polarization of the transistor and negate most of the effects of the negative poling pulse.

B. Contrasts to MOSFET-based SRAM circuits

Differences between MOSFET-based and MFSFET-based SRAM circuits are not unlike those of the inverter. The circuit with MFSFETs can behave identically to the one with MOSFETs for positive applied voltages, and create different variations when negative voltages are applied at node V1. The results below detail the differences between the MOSFET and MFSFET in the SRAM circuit with regard to current and static characteristics, but differences in switching performance were mentioned in the previous section.

The MOSFET-based SRAM cell was tested using the Sanyo 2SK669 n-channel MOSFET, V_{DD} = 4 V, R_L = 100 kΩ, and the input signal in the form of a 0 V to 4 V square wave. The voltage at V2 acted in manner opposite to node V1. Node V2 had a maximum value of just less than 4 V, at approximately 3.6 V. The waveforms of nodes V1 and V2 for the MOSFET SRAM are shown in Figure 4.11.

The SRAM based on MFSFETs displayed similar behavior, except for the level of the voltages values not being as distinct as would be desired. The same issues observed for a similar inverter circuit were also noticed here; the V_{OL} value was not very low when using a 100 kΩ load with the ND1 MFSFET. This result is shown in the waveform in Figure 4.12.

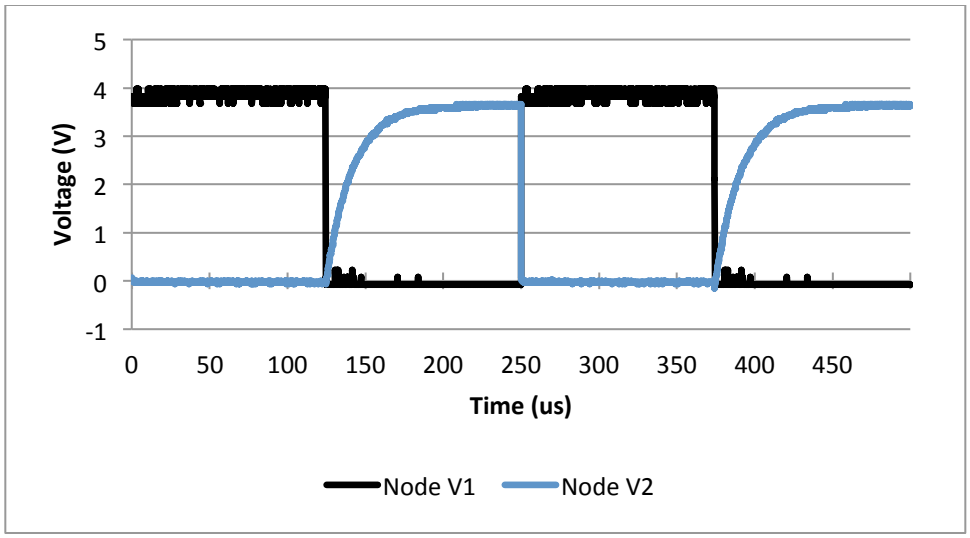

Figure 4.11 MOSFET SRAM Node V2 voltage for 0 V to 4 V square wave input range with 100 kΩ load resistance

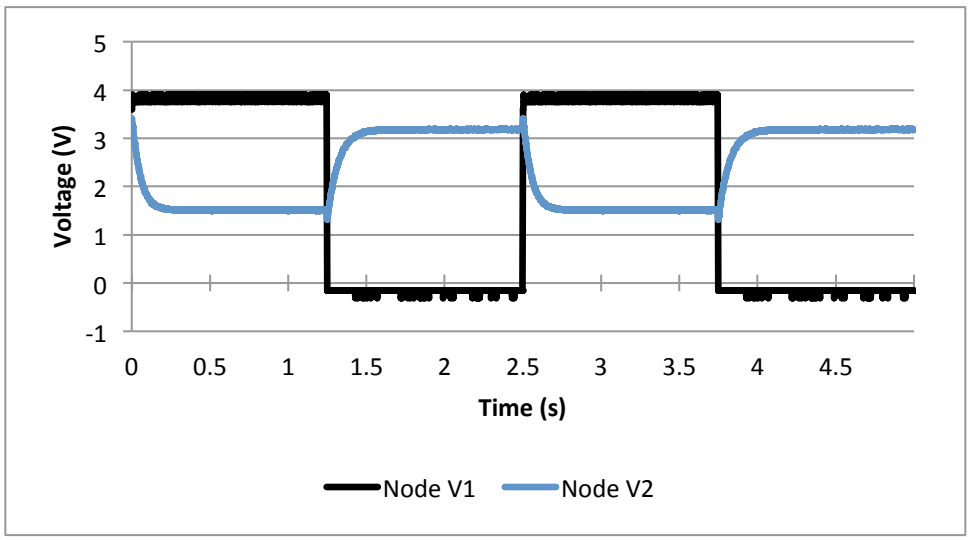

Figure 4.12 ND1 SRAM Node V2 voltage for 0 V to 4 V square wave input range with 100 kΩ load resistance

In addition, the current found in the MOSFET SRAM displayed no hysteresis, while the MSFET SRAM did. Figures 4.13 and 4.14 present the current found in the

MOSFET and ND1 SRAM circuits when using V_{DD} = 1 V, R_L = 100 kΩ, and inputs varying between 0 V and 4 V with a triangle wave rather than a square wave.

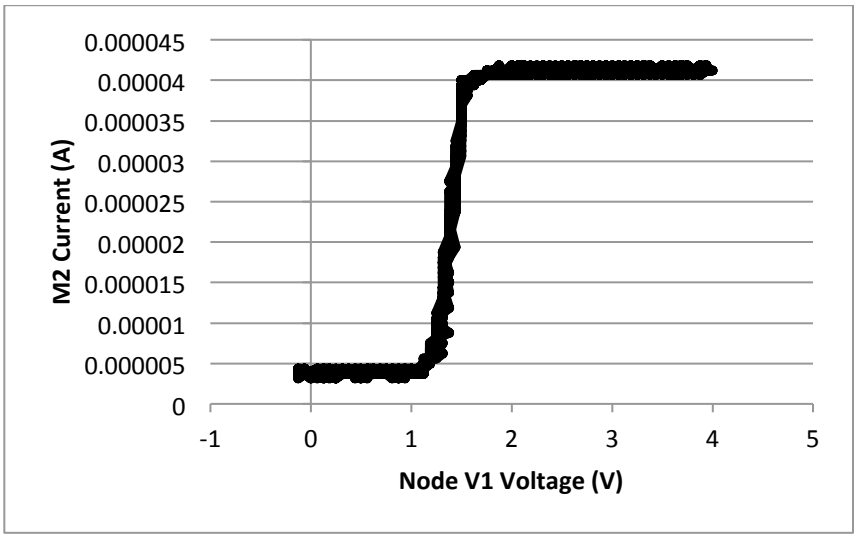

Figure 4.13 MOSFET SRAM current for 0 V to 4 V triangle wave input range with 100 kΩ load resistance

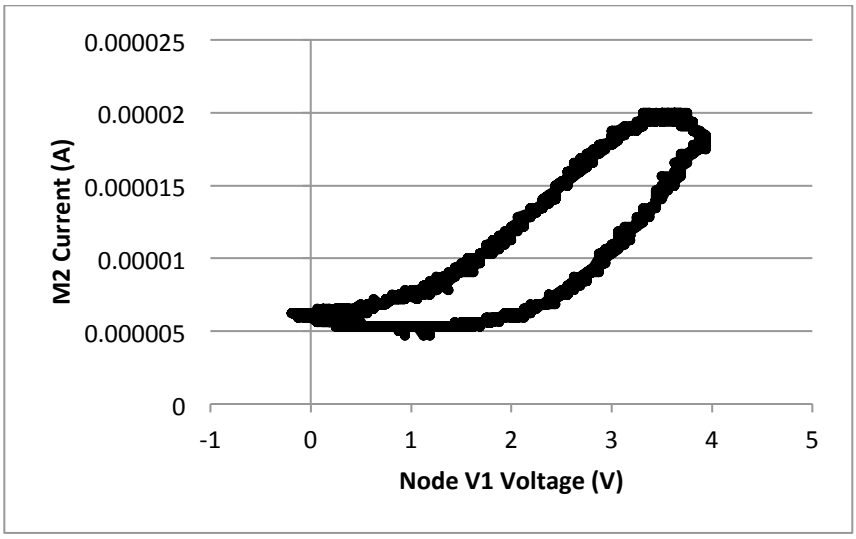

Figure 4.14 ND1 SRAM current for 0 V to 4 V triangle wave input range with 100 kΩ load resistance

C. Analysis

The SRAM circuit provided many opportunities to investigate the unique aspects of the MFSFET. Like the MFSFET in a standalone configuration or in an inverter circuit, the current measured through the transistors displayed hysteresis when varying between a loop of applied voltage values at node V1. The primary useful feature of the MFSFET SRAM, however, was the ability to use the circuit as a nonvolatile memory cell. This would be useful in case of power loss to the circuit, but also could be used to specifically design a circuit in order to take advantage of these characteristics. The power supply was not required to be turned on for the circuit in order to apply a negative voltage to write a value, meaning that power would only be required for reading. While negative voltages like -4 V are not typically seen in digital logic, these could be used along with one or both values of 0 V or 4 V in order to create a logic scheme used for writing to the SRAM cell. Because the cell stores both a value and its complement, the definition of a polarity being assigned to a logic level is left to the designer to determine a point of reference.

CHAPTER V

THE FERROELECTRIC DYNAMIC RANDOM ACCESS MEMORY CIRCUIT

In addition the SRAM, another type of volatile memory circuit is the dynamic random access memory (DRAM) cell. The DRAM removes the load devices found in the SRAM, which are used to prevent charge loss caused by leakage. Instead, charge losses are addressed by a refresh operation consisting of reading the cell contents and rewriting them back into the cell. These memory circuits rely on the usage of a storage capacitor to hold the charge written in the cell [22]. This chapter presents experimental results of both a three-transistor circuit and a one-transistor, one-capacitor (1T-1C) circuit.

A. The 3T DRAM Circuit

The 3T DRAM circuit, shown in Figure 5.1, is a DRAM cell consisting of only three transistors and a capacitor, a simpler circuit than the SRAM cell that contains four transistors in its core cell, and even six when considering the access transistors attached to the bit lines. The SRAM cell contains both the written value and its complement, due to its latch structure of inverters. The DRAM cell, however, stores only the written value. In this circuit, the M1 transistor behaves as the writing transistor, M2 acts as the storage transistor, and M3 is the reading transistor [22].

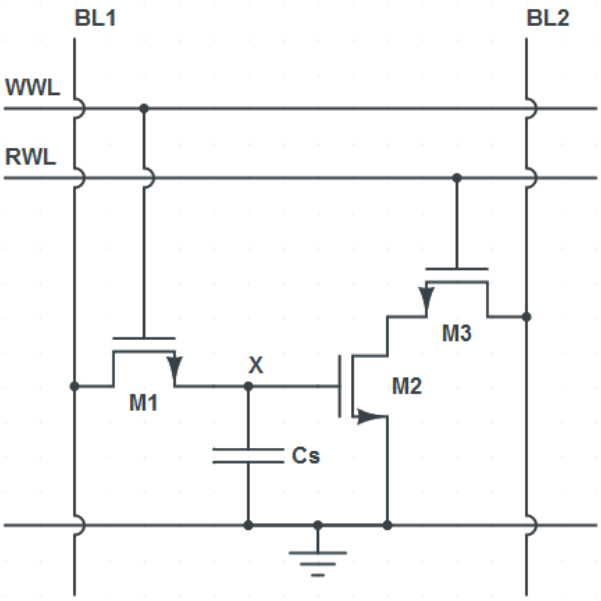

Figure 5.1 Structure of the 3T DRAM circuit [22]

a. Test Circuit

Testing the DRAM required creating a circuit that would be able to sense the output on the BL2 bit line. Common suggestions are to precharge the bit line to V_{DD} or $(V_{DD} - V_T)$, or to clamp the bit line to V_{DD} with a load device such as an NMOS transistor in saturation or a PMOS transistor that has been grounded. The precharge method is generally preferred because the use of a load device would require careful transistor sizing [22]. In these experiments, the DRAM was connected to a V_{DD} through a load resistor, as shown in Figure 5.2.

While writing to the DRAM circuit, the M1 transistor and the capacitor are the primary elements of concern. This portion of the circuit has been redrawn and shown in Figure 5.3 to emphasize the events of the write operation.

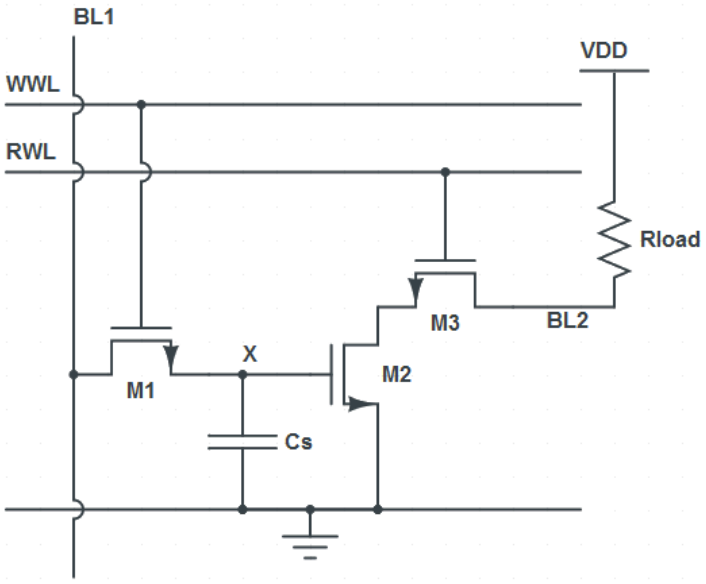

Figure 5.2 Test circuit for 3T DRAM cell

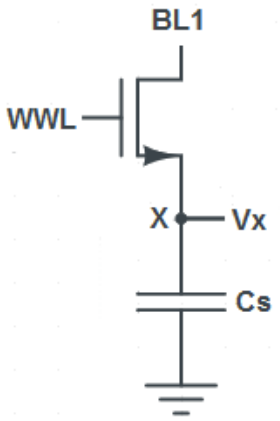

Figure 5.3 Active elements during DRAM write operation

The value to be written to the cell should be placed on the bit line BL1, which is the drain of M1. The write word line (WWL), the gate of M1, is set high when a write operation takes place and set low to stop writing. Raising the WWL signal turns on M1

to complete the connection between BL1 and node X. The value measured at node X, which will be referred to as V_X when speaking of its voltage, will be equal to $(V_{DD} - V_T)$ when a "1" is written to the cell, and will be 0 V when a "0" is written to the cell. This value of V_X is then stored on the capacitor C_S, which also drives the gate of M2 [22].

The read operation of the DRAM uses transistors M2 and M3 to move the value stored at X to the bit line BL2. The gate of M2 is connected to the storage capacitor, and M2 is turned on when a high logic voltage value is stored in C_S. The drain of M2 is connected to the source of M3, and the drain of M3 is BL2, which is connected to V_{DD} through a load resistor. When the read word line (RWL), the gate of M3, is set high, M3 is turned on and completes the connection between M2 and BL2 [22]. This portion of the DRAM circuit, shown in Figure 5.4, can be visualized as a NAND gate with inputs V_X and RWL and output BL2.

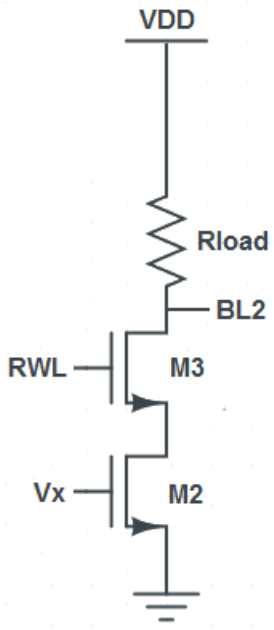

Figure 5.4 Active elements during DRAM read operation

When V_X is high, the cell contains a logic "1" value, but this can only be read at BL2 if RWL is also high. Because BL2 is connected to V_{DD} through the load resistor, the value measured at BL2 is actually the inverse of the value stored in the cell. Table 5.1 shows the possible logic values measured at BL2 depending on the stored value in the DRAM cell and the value of RWL; this mimics the truth table of a NAND gate.

Table 5.1 Logic values of 3T DRAM elements during read operation

RWL	V_X	BL2
0	0	1
0	1	1
1	0	1
1	1	0

b. Experimental Data

Three variations of the 3T DRAM were tested: only MOSFETs in the circuit, only M1 replaced with an MFSFET, and all transistors as MFSFETs. Experiments were initially conducted using only MOSFETs in order to establish a baseline for the results. In all cases when MOSFETs were present, the Sanyo 2SK669 n-channel MOSFET was used. The ND1 MFSFET was used when replacing one or more MOSFETs.

In these experiments, regardless of the type of transistor being used, a common set of steps was followed. The RWL signal was held high at all times so that the output could be easily observed with respect to changes at the input. The WWL signal was controlled by a 0 V to 4 V square wave input at 0.5 Hz, and the BL1 signal was controlled by a square wave input at 1 Hz, but otherwise synchronized with the WWL

transitions. This allowed for both "0" and "1" values to be written to the cell during a cycle where WWL was high, and one of the values would be stored as WWL went low and the write operation ceased. Experiments were alternated to allow both values to be stored in the cell and the circuit's behavior to be observed fully with these actions. In half of the tests of the MFSFET-based circuits, WWL was set to be a square wave of -4 V to 4 V instead, in order to see the effect of poling the MFSFET with a negative voltage.

Figures 5.5 through 5.8 show the write and read operations of the 3T DRAM utilizing only MOSFETs. The load resistor connecting BL2 to V_{DD} had a value of 10 kΩ, and the storage capacitor had a capacitance of 10 μF. In Figure 5.5, the write operation shows that while the WWL signal was high, a "0" was written and then a "1" was written. Because the WWL was lowered as the BL1 signal was lowered, the "1" value was stored to the cell. The V_X voltage shows that the "1" value was stored, and that voltage slowly decreased as the capacitor discharged. The maximum value of V_X was less than 4 V because $V_X = (V_{DD} - V_T)$ when a "1" is written to the circuit.

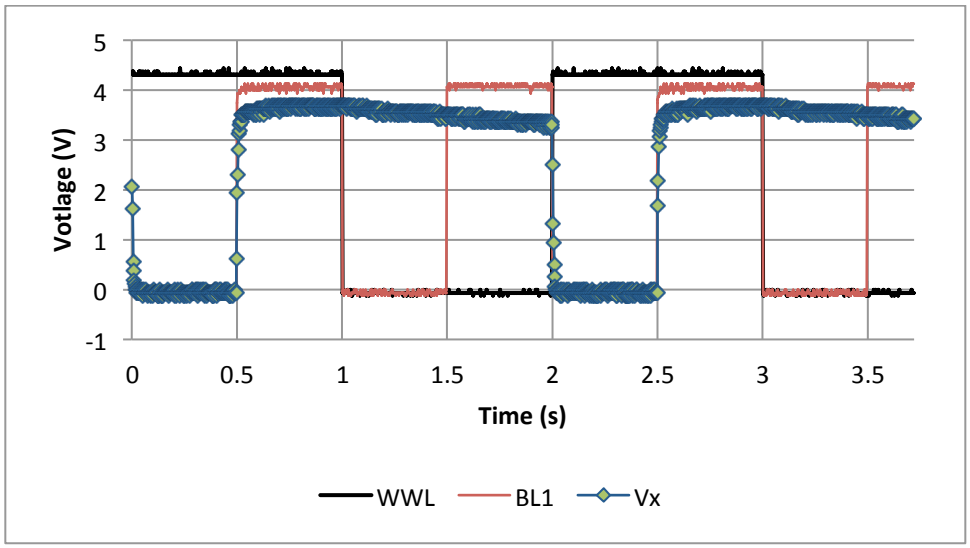

Figure 5.5 MOSFET 3T DRAM with write operation to store "1"

Figure 5.6 shows the reading of the circuit. The BL2 signal was high when the V_X value was low, and the RWL signal was always high. Therefore, this reflected the NAND-like nature of the read operation of the circuit. BL2 was always low except for the brief periods where a "0" was being written to the cell, showing that the output was correctly read since BL2 should be the inverse of the stored value.

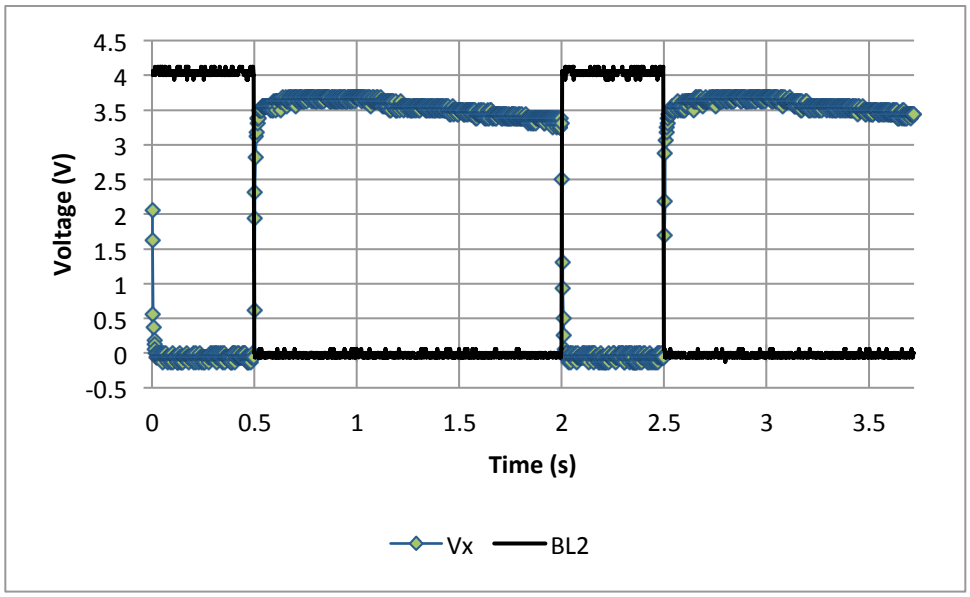

Figure 5.6 MOSFET 3T DRAM with read operation to read "1"

Figure 5.7 shows the operation of writing a "1," writing a "0," and then storing the "0" for the all-MOSFET DRAM, the opposite of the set of operations shown in Figure 5.5. It shows that the V_X signal dropped to 0 V when a "0" was written to the cell, and remained at 0 V since there was no charge on the capacitor. Figure 5.8 shows the read operation, and similar to Figure 5.6, shows that the reading was successful. BL2 remained high most of the time, meaning that a "0" was being read from the cell, except for the brief periods when a "1" was written.

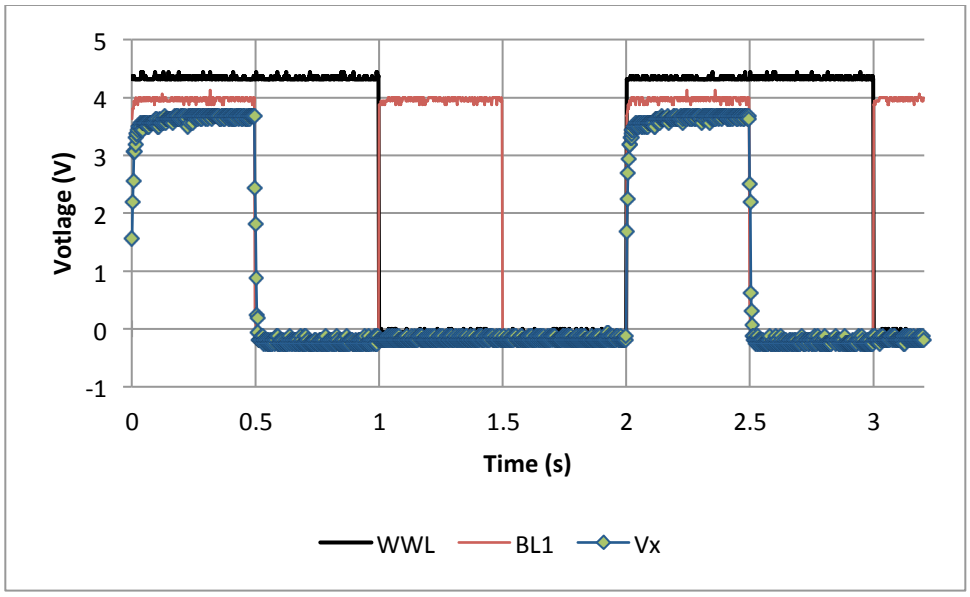

Figure 5.7 MOSFET 3T DRAM with write operation to store "0"

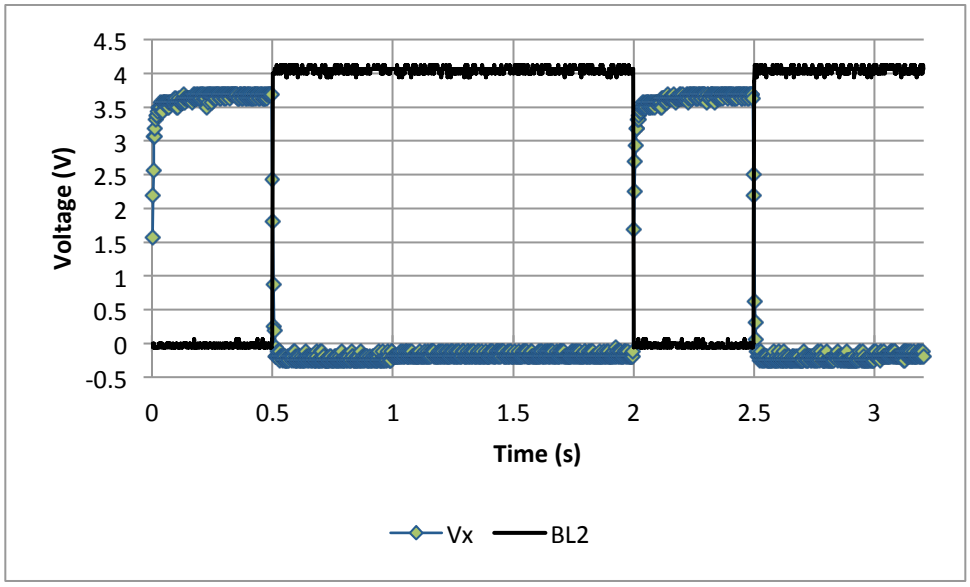

Figure 5.8 MOSFET 3T DRAM with read operation to read "0"

Figures 5.9 through 5.16 detail the operation of the 3T DRAM when replacing the write transistor, M1, with an ND1 MFSFET. The other MOSFETs remained in place. The storage capacitor was changed to 0.1 µF while configuring the circuit, but another

value could be chosen as needed. The load resistor connecting BL2 to V_{DD} had a value of 10 kΩ. Figure 5.9 shows the operation of writing a "0," writing a "1," and storing the "1." The V_X value did not reach the same maximum value as shown when a MOSFET was used for writing, but did increase during write "1" operations. This can be attributed to an increased threshold voltage in the MFSFET due to the positive gate voltage polarization. The V_X voltage value did not retain this level for long, and discharged quickly after the WWL signal was lowered; this should be solved by choosing a different capacitor. The read operation shown in Figure 5.10 reflects what one would expect in terms of BL2 remaining low when a "1" should have been stored. However, because the V_X value dropped quickly, one would expect BL2 to actually only be low if both RWL and V_X were high. In this particular case, V_X dropped, but BL2 remained low as well rather than going high.

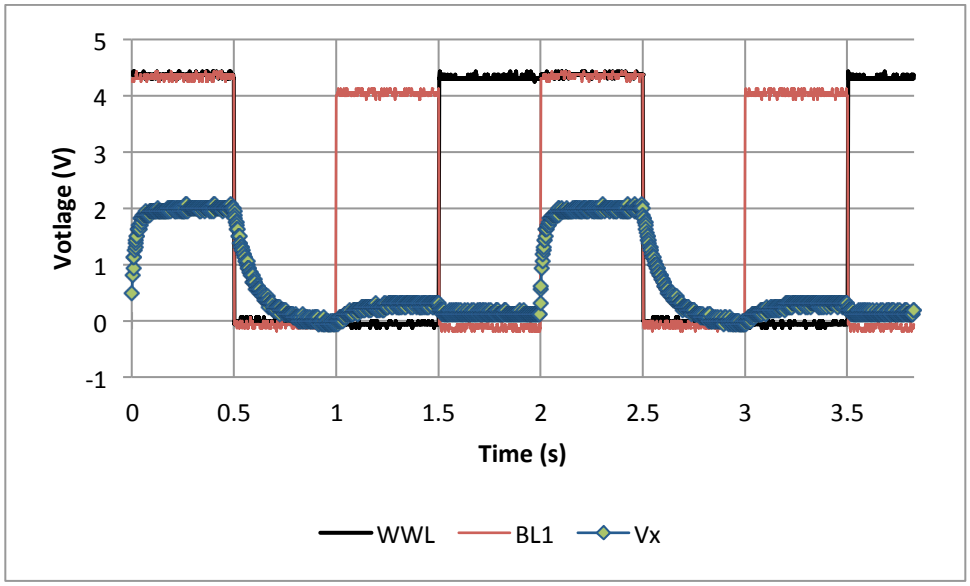

Figure 5.9 Mixed MFSFET/MOSFET 3T DRAM with write operation to store "1"

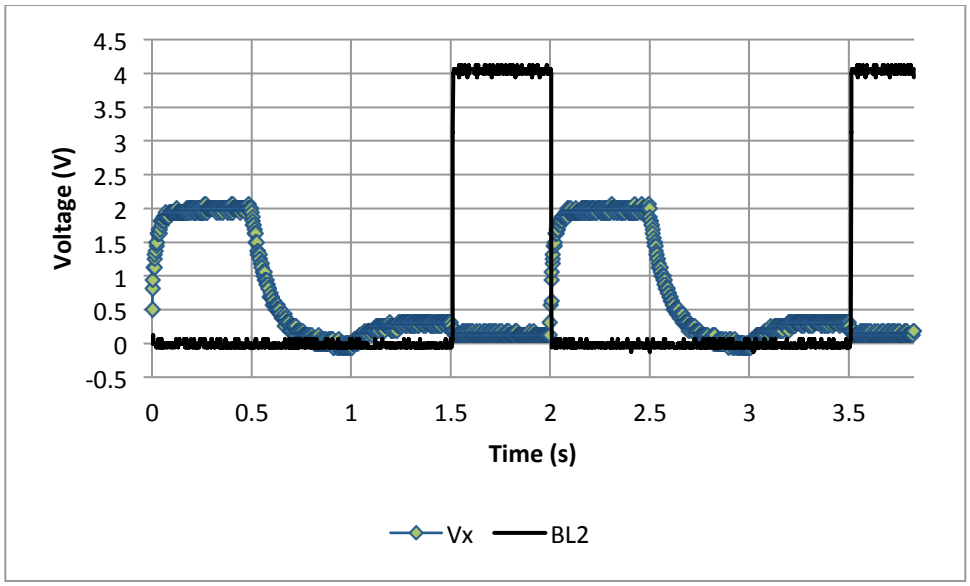

Figure 5.10 Mixed MFSFET/MOSFET 3T DRAM with read operation to read "1"

The same variation of the previous mixed-transistor DRAM was tested again with a write sequence of writing "1," writing "0," and storing "0." The write operation is shown in Figure 5.11.

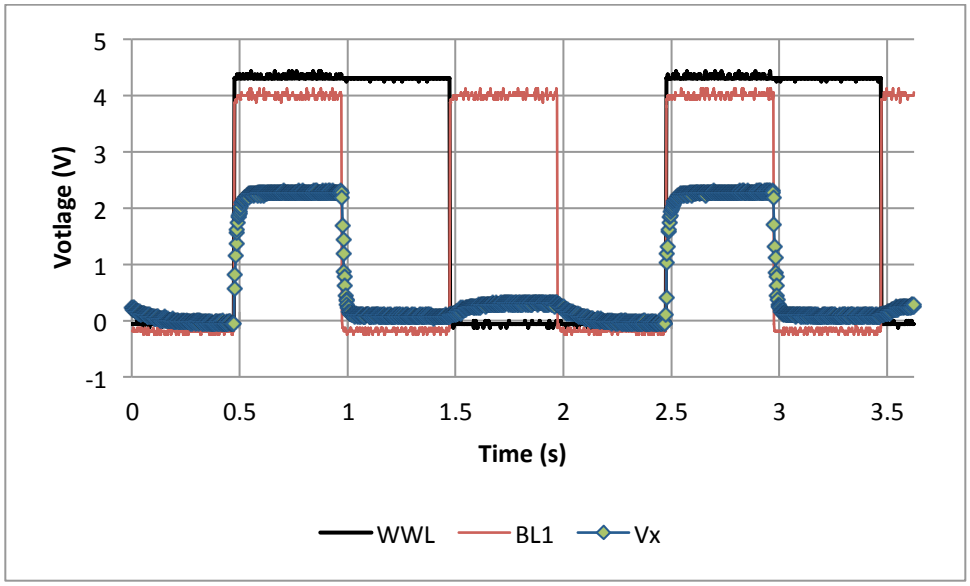

Figure 5.11 Mixed MFSFET/MOSFET 3T DRAM with write operation to store "0"

It varied little from the previous circuit, with the maximum value of V_X only reaching approximately 2 V when a "1" value was being written to the cell. In Figure 5.12, the output at BL2 was mostly expected as well, except that there was an unexpected decrease in the BL2 value when BL1 transitioned from high to low, but WWL was low and therefore no value should have been written to the cell.

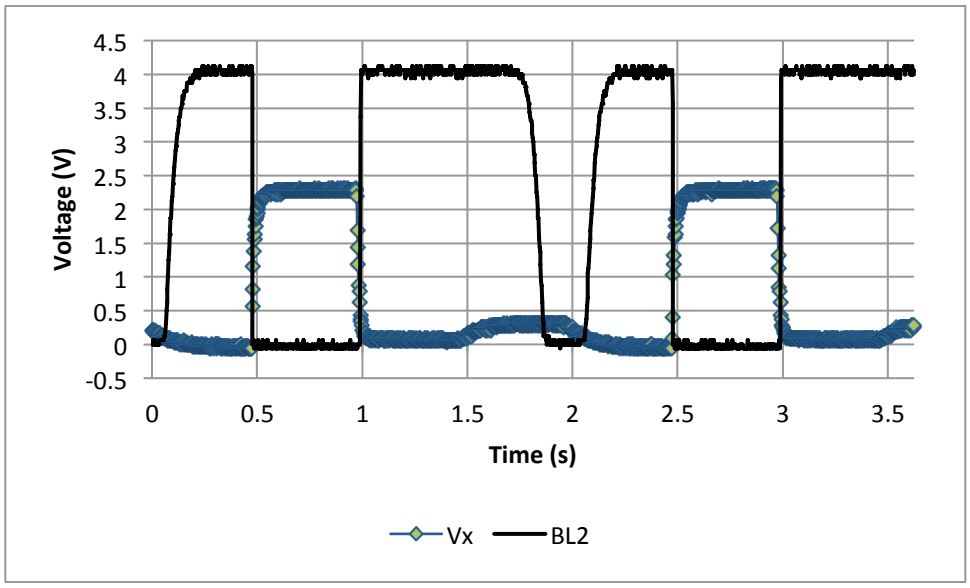

Figure 5.12 Mixed MFSFET/MOSFET 3T DRAM with read operation to read "0"

The mixed-transistor DRAM was tested again, with the WWL set to be a square wave ranging from -4 V to 4 V instead of 0 V to 4 V. During the operation to store a "1," shown in Figure 5.13, no different effects were seen that had not already been demonstrated in Figure 5.9 when only 0 V to 4 V was allowed on WWL for the same write operation. The result of the read operation in Figure 5.14 was already very similar to the read operation in Figure 5.10; this is not surprising, since the transistors on the read

side of the circuit had not been replaced, but it could be possible for variations in V_X to affect the read operation if any were present.

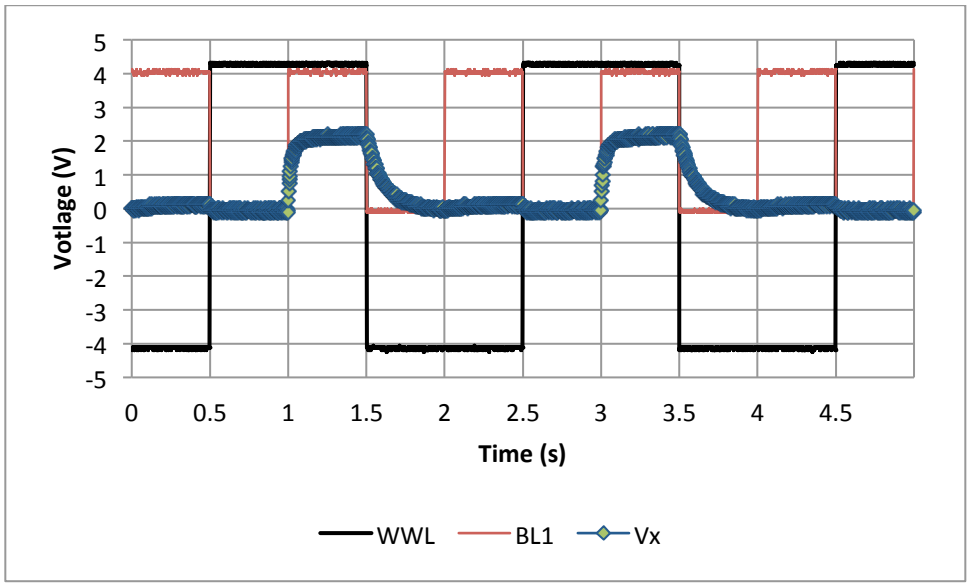

Figure 5.13 Mixed MFSFET/MOSFET 3T DRAM with write operation to store "1" including negative voltages at WWL

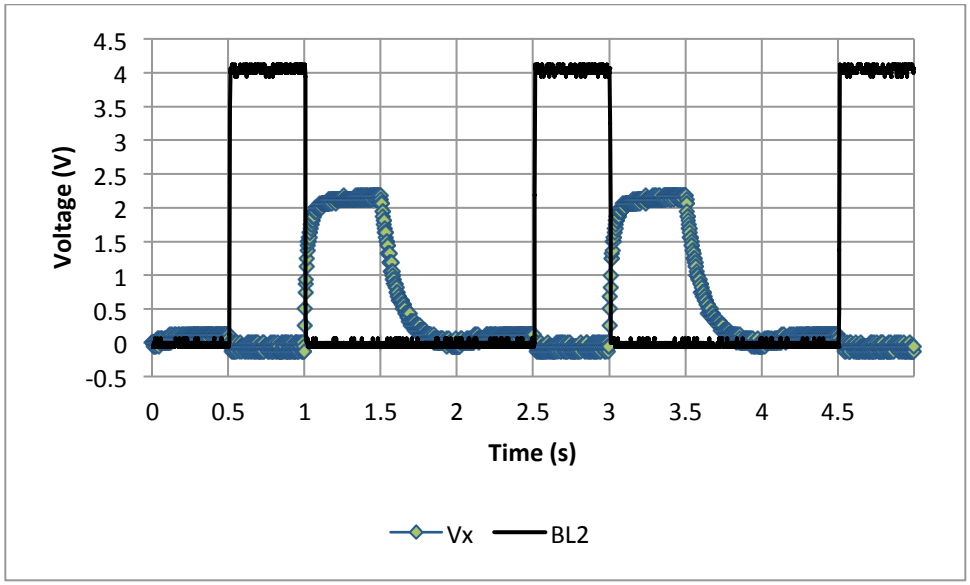

Figure 5.14 Mixed MFSFET/MOSFET 3T DRAM with read operation to read "1" including negative voltages at WWL

The most noticeable change due to the application of negative voltages to WWL was seen during the operation to store a "0." The V_X level increased to a maximum of more than 3 V rather than only 2 V, which can be attributed to a decreased threshold voltage resulting from a rapid change from -4 V to 4 V on WWL. This is shown in Figure 5.15. The read operation in Figure 5.16 also displayed correct behavior, since the output BL2 was only low during the time when V_X was around 3 V, and was high for the remainder of the read process.

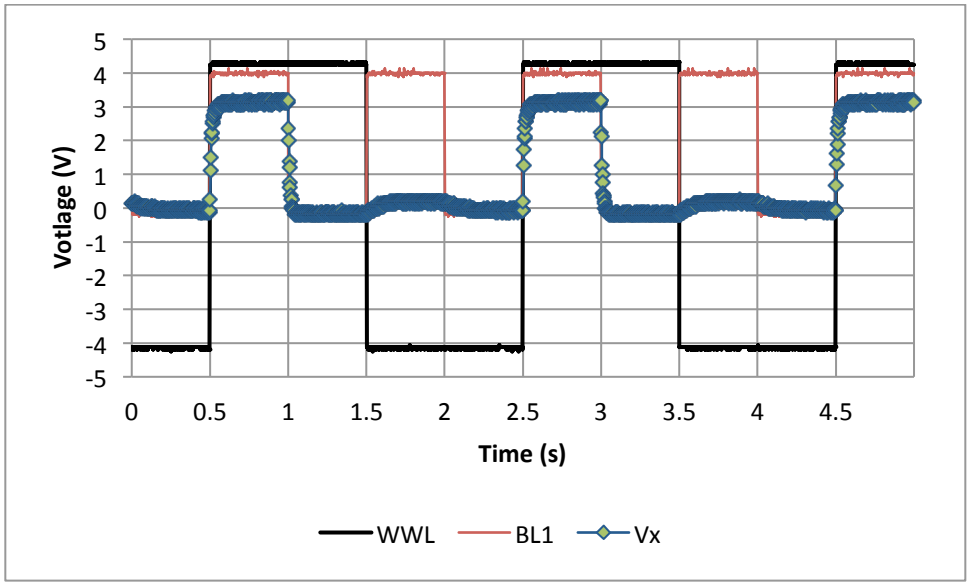

Figure 5.15 Mixed MFSFET/MOSFET 3T DRAM with write operation to store "0" including negative voltages at WWL

The final configuration of the 3T DRAM was an all-MFSFET configuration utilizing the ND1 transistor. The load resistor was changed to 250 kΩ to accommodate the lower current capabilities associated with the ND1 transistor that would be connected to BL2. The storage capacitor remained at 0.1 μF for these tests. Figure 5.17 displays the write operation of the circuit when storing a "1" to the cell, with results not unlike

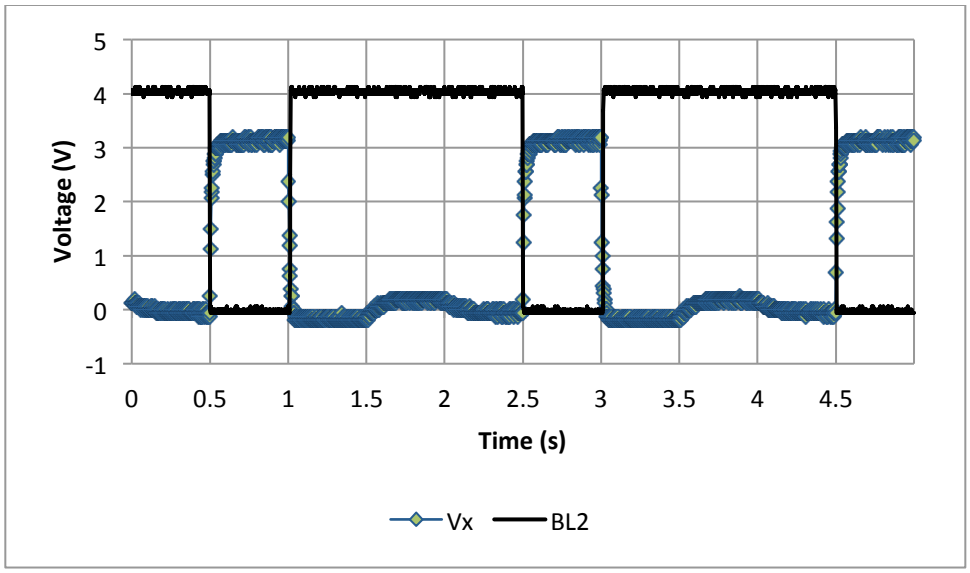

Figure 5.16 Mixed MFSFET/MOSFET 3T DRAM with read operation to read "0" including negative voltages at WWL

those seen in Figure 5.9 when only M1 was an MFSFET. Figure 5.18, however, shows a drastically different result when reading from the circuit after all transistors had been replaced with MFSFETs.

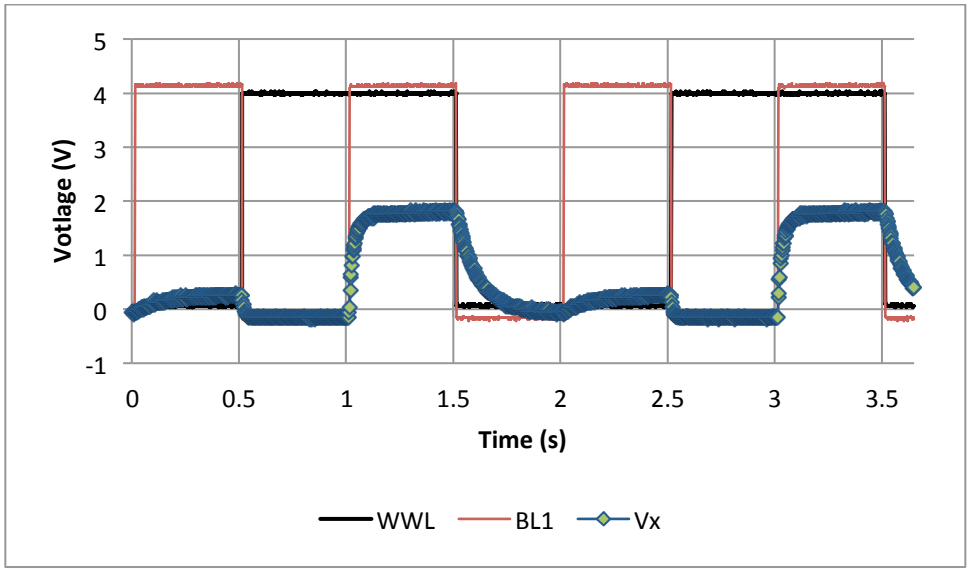

Figure 5.17 MFSFET 3T DRAM with write operation to store "1"

Even though the BL2 value did not approach the low voltage values seen in previous configurations of this circuit, it would still be possible to sense that the bit line had dropped and amplify the difference in order to determine what value should be read from the circuit.

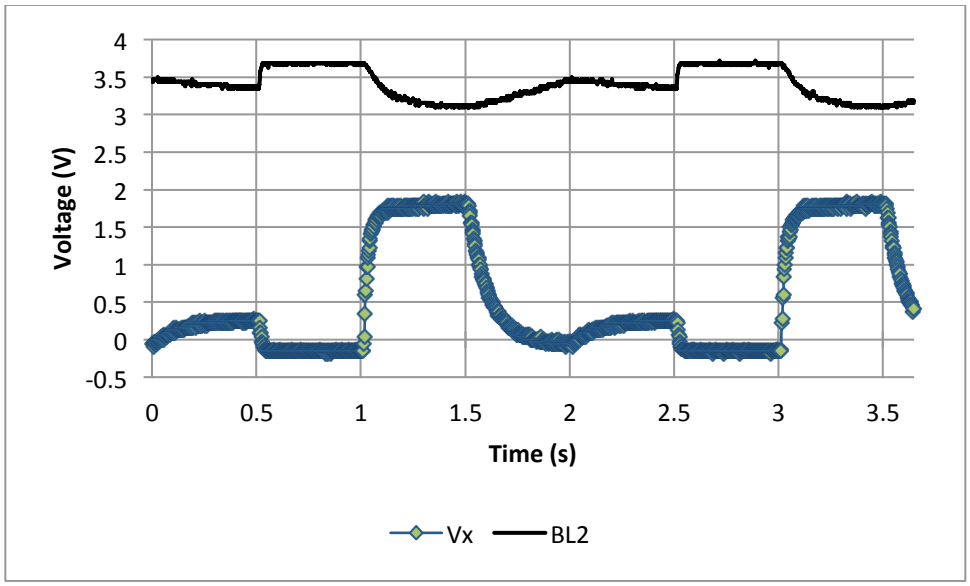

Figure 5.18 MFSFET 3T DRAM with read operation to read "1"

Figure 5.19 again shows a similar result when storing a "0" as Figure 5.11 did for the mixed-transistor configuration. Figure 5.20 presents another situation where the output on BL2 did not reach the desired voltage values, but the difference in output would be able to be amplified and sensed to determine the expected value.

No surprises were found while storing a "1" when the WWL was allowed to vary between -4 V and 4 V, with the write operation shown in Figure 5.21. The read operation in Figure 5.22 continued to show the same pattern as the other variations of the circuit

using all MFSFETs; the output was high when a "0" was written, and the lower voltage seen otherwise could be interpreted as the bit line dropping to indicate reading a "1."

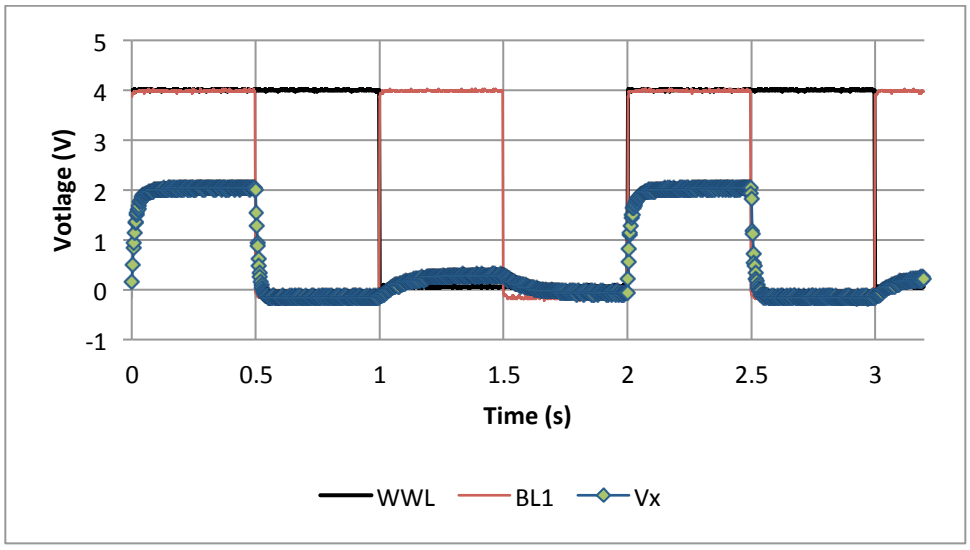

Figure 5.19 MFSFET 3T DRAM with write operation to store "0"

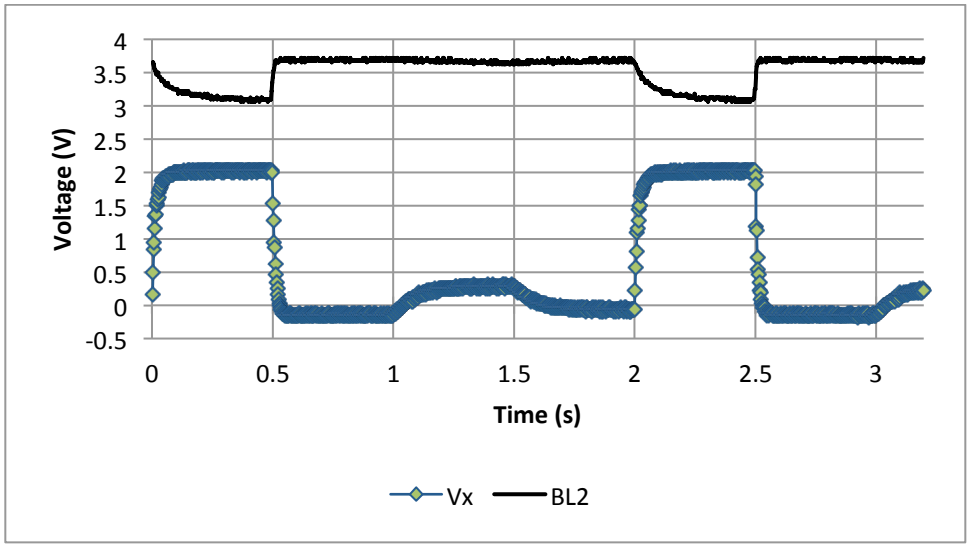

Figure 5.20 MFSFET 3T DRAM with read operation to read "0"

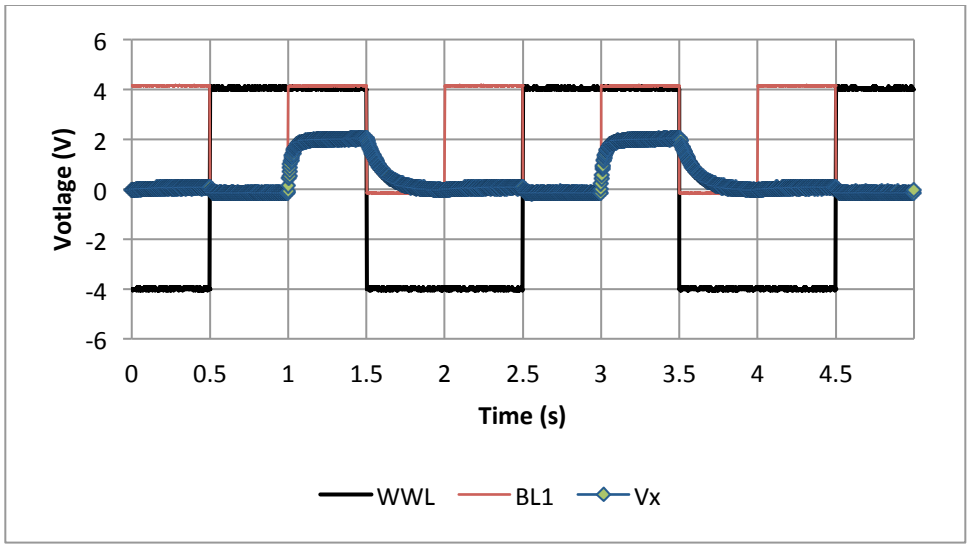

Figure 5.21 MFSFET 3T DRAM with write operation to store "1" including negative voltages at WWL

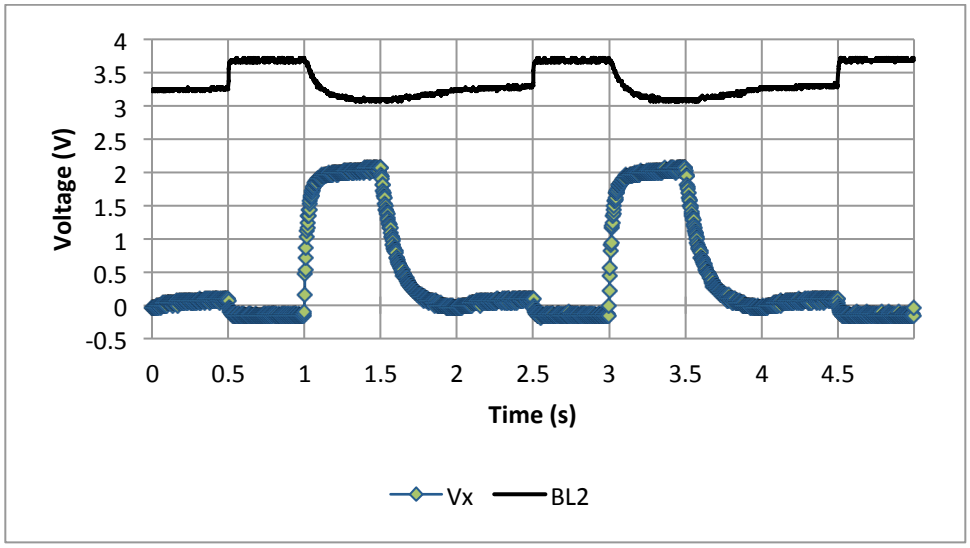

Figure 5.22 MFSFET 3T DRAM with read operation to read "1" including negative voltages at WWL

The final test of the DRAM, with writing shown in Figure 5.23 and reading in Figure 5.24, did mimic the results of Figure 5.15 where the negative gate voltage at

WWL allowed V_X to increase more than usual. The output remained less than ideal, but usable due to the clear distinction of changing values in the cell.

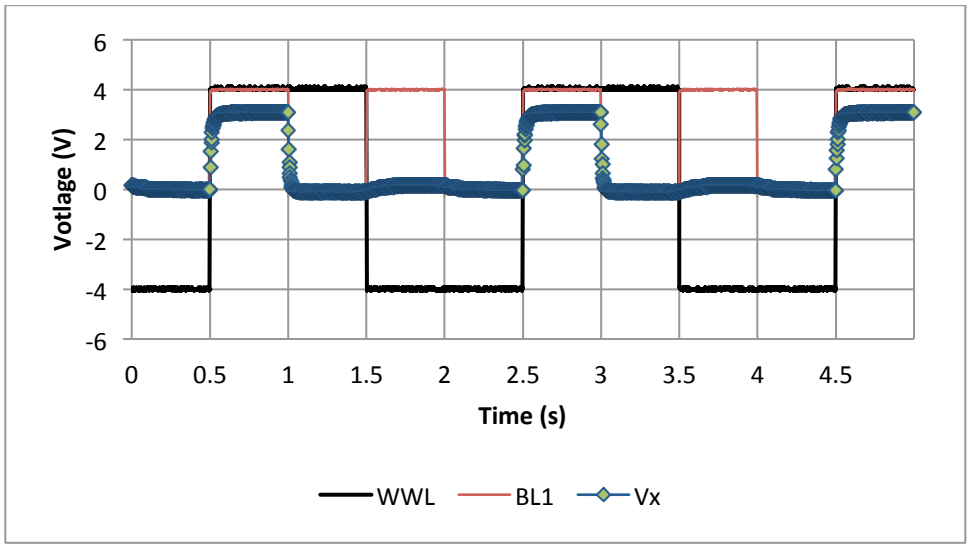

Figure 5.23 MFSFET 3T DRAM with write operation to store "0" including negative voltages at WWL

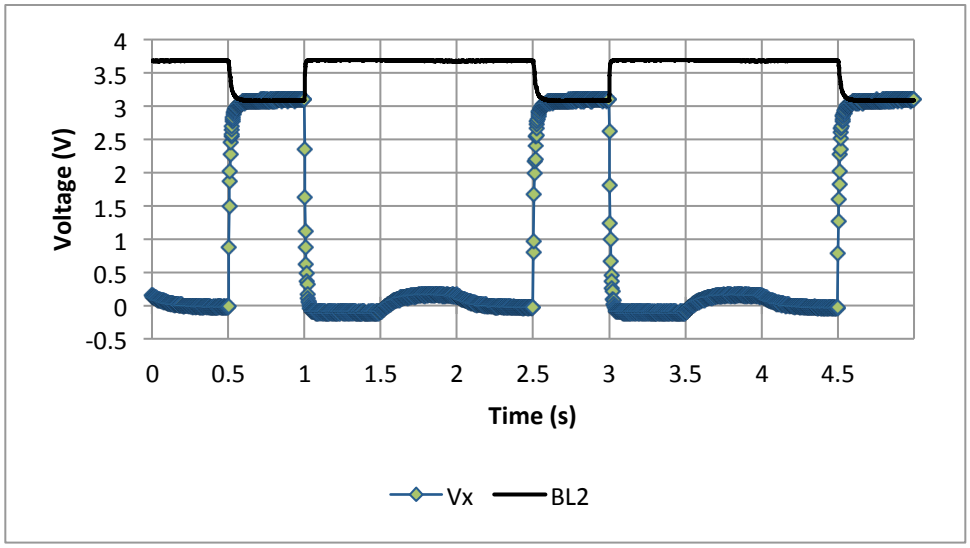

Figure 5.24 MFSFET 3T DRAM with read operation to read "0" including negative voltages at WWL

c. Analysis

The results presented from these experiments show that the MFSFET can successfully be used in the 3T DRAM circuit. There were three interesting points demonstrated in these results. First, the circuits using an MFSFET for the write operation exhibited an effect of the V_X value increasing with changes to BL1, even when WWL was set low and should have prevented any modifications to the cell contents; this suggests that the polarization in the transistor may have allowed to continue to pass a small amount of current from the drain to source even in an expected off state. Second, the capacitor value chosen for the circuit using an MFSFET did not appear to be ideal, and so this could be adjusted for more desirable results. Finally, the value seen on the bit line BL2 showed less ideal behavior when using an MFSFET at the read transistor location, but the value varied enough to indicate the correct reading of the stored value; further experimentation could be conducted by altering the RWL signal to also include negative voltages to see the impact of the varied polarization on that transistor. Also, the load device is another point of experimentation that could allow more distinct transitions, such as by utilizing a p-channel MOSFET.

B. The 1T-1C DRAM Cell

The 1T-1C DRAM cell further simplifies the 3T DRAM cell by eliminating explicit reading and writing lines in favor of a single word line that controls access to the cell's contents. This also allows the cell to operate with only one bit line rather than specific bit lines for both read and write operations. This smaller cell layout is shown in Figure 5.25 [22].

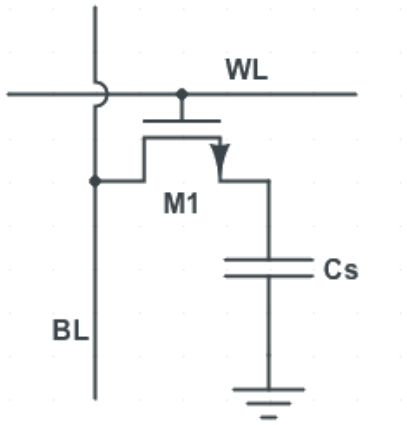

Figure 5.25 Structure of the 1T-1C DRAM circuit

a. Test Circuit

The 1T-1C DRAM cell also required additional circuitry in order to write to and read from the cell. The test circuit used in these experiments is shown in Figure 5.26. To write to the cell, it was necessary to apply V_{DD} or 0 V to the bit line BL, representing a "1" or "0" logic level, while raising the word line WL to a V_{DD}. To read from the cell, the word line WL was first lowered to 0 V, and then the bit line was precharged to $V_{DD}/2$; the word line WL was then raised to V_{DD} after precharging was complete, so that the bit line value could increase if a "1" had been stored or decrease if a "0" had been stored.

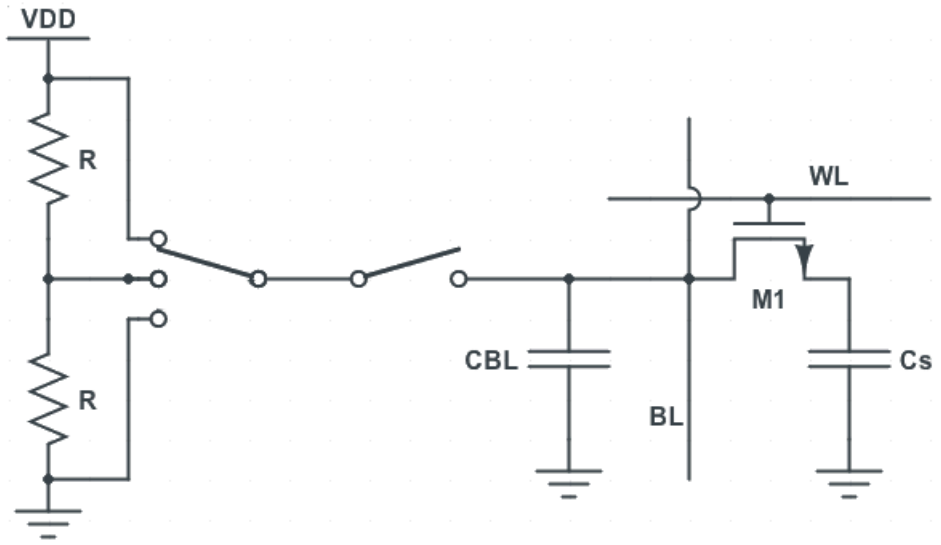

Figure 5.26 Test circuit for the 1T-1C DRAM cell

b. Experimental Data

As with the 3T DRAM, experiments were first conducted with only a MOSFET being used in order to establish a baseline behavior for the circuit; the Sanyo 2SK669 n-channel MOSFET was used. For all experiments, including those with the MFSFET, the data collection focused on the response of the circuit during the read process, since the write process only consisted of forcing the drain of the transistor to 0 V or V_{DD} while the WL signal was raised. Figure 5.27 and Figure 5.28 show the read processes for reading a "1" and "0" value, respectively, from the MOSFET-based DRAM circuit. The precharge operation is clearly marked in each figure; this was the point of the experiment where WL was lowered to 0 V and the bit line was precharged to $V_{DD}/2$. The read operation is also noted on each figure. In Figure 5.27, the circuit successfully read that a "1" had been stored in the cell because the voltage on the bit line capacitor increased above its precharged value. Likewise, Figure 5.28 shows that the cell's stored "0" value was

successfully read because the voltage on the bit line capacitor dropped from its precharged value to 0 V when the WL signal was raised.

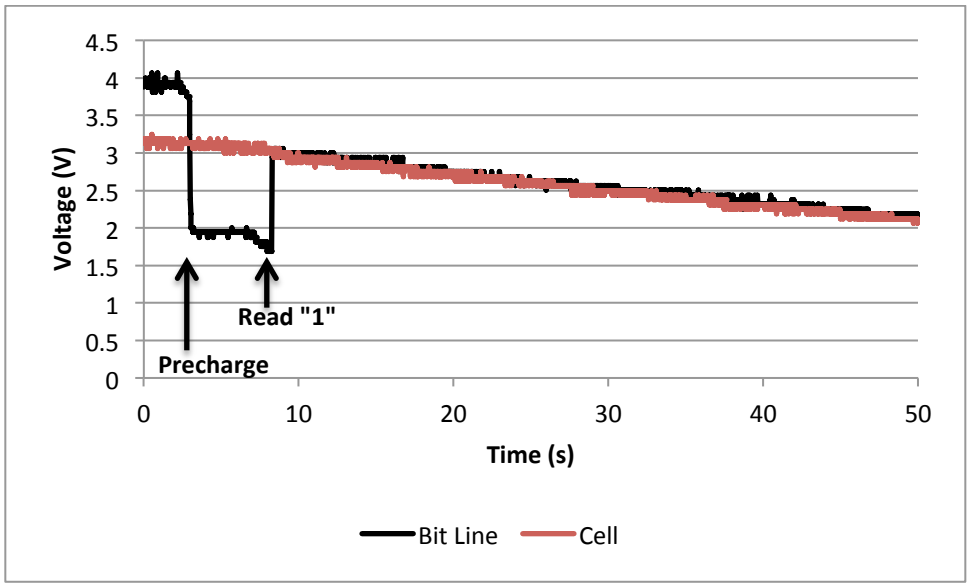

Figure 5.27 Reading a “1” from a MOSFET 1T-1C DRAM

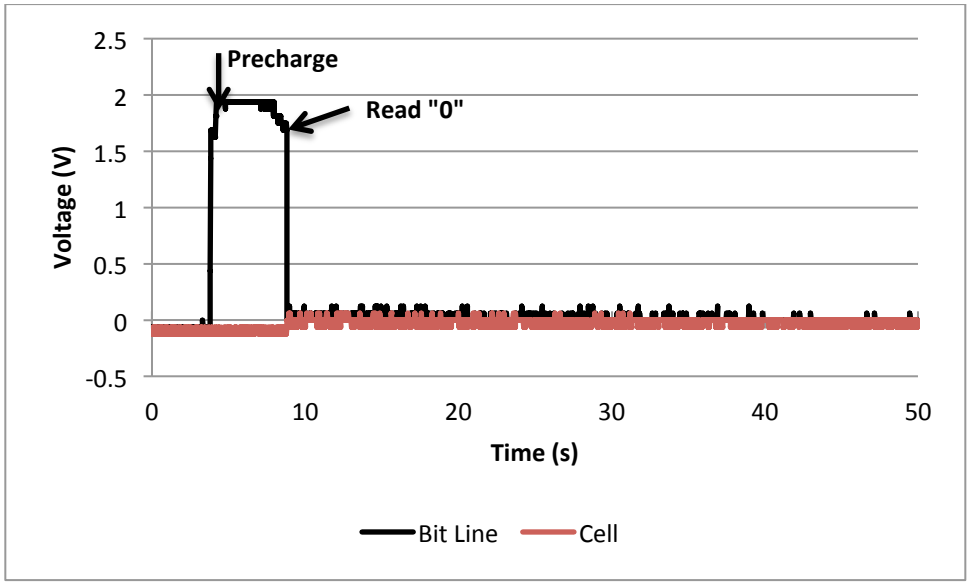

Figure 5.28 Reading a “0” from a MOSFET 1T-1C DRAM

The MOSFET was replaced with an ND1 MFSFET, and Figures 5.29 and 5.30 show that this variation of the circuit was also able to successfully read the value stored in the cell since the bit line voltage increased or decreased from its precharged value as expected. What was unexpected, however, was that the bit line voltage and internal capacitor's voltage began increasing after reading a "0" value; while there was an initial drop in the bit line voltage to indicate that a "0" was being read, the voltage on both capacitors began increasing and eventually settled to a value of approximately $V_{DD}/2$.

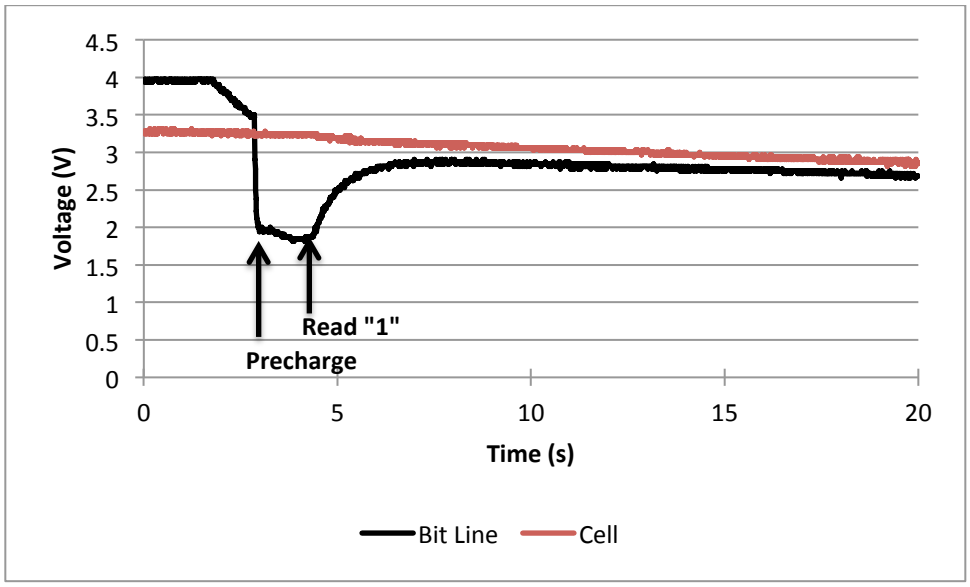

Figure 5.29 Reading a "1" from an ND1 MFSFET 1T-1C DRAM

Negative voltages were applied to the word line WL in order to change the polarization on the gate of the transistor. However, no obvious effect was seen by doing this. In Figure 5.31, the word line WL was toggled between 4 V and -4 V rather than the typical 4 V and 0 V values that were used to turn the transistor on and off. The read

operation showed similar results to the read operation in Figure 5.29 where only 4 V and 0 V were used.

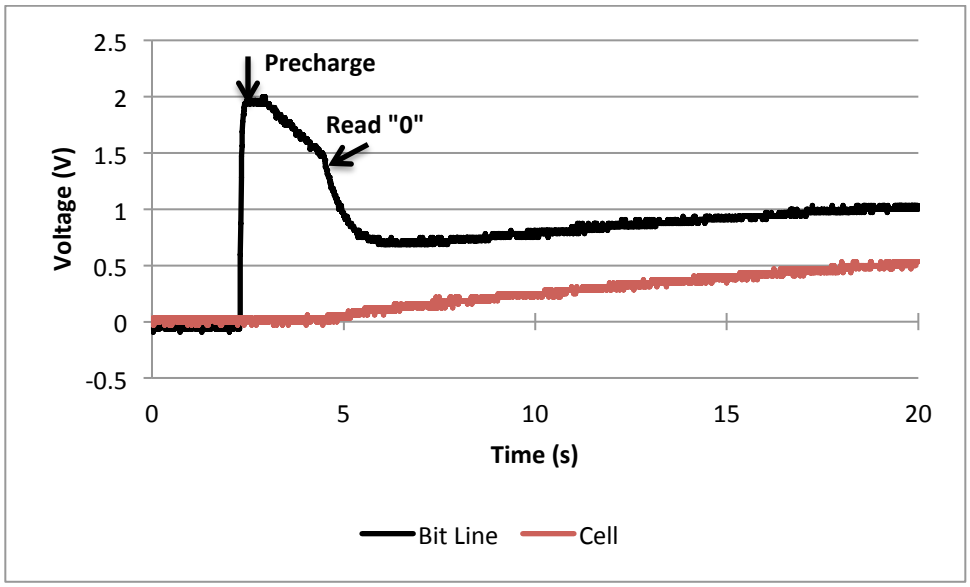

Figure 5.30 Reading a “0” from an ND1 MFSFET 1T-1C DRAM

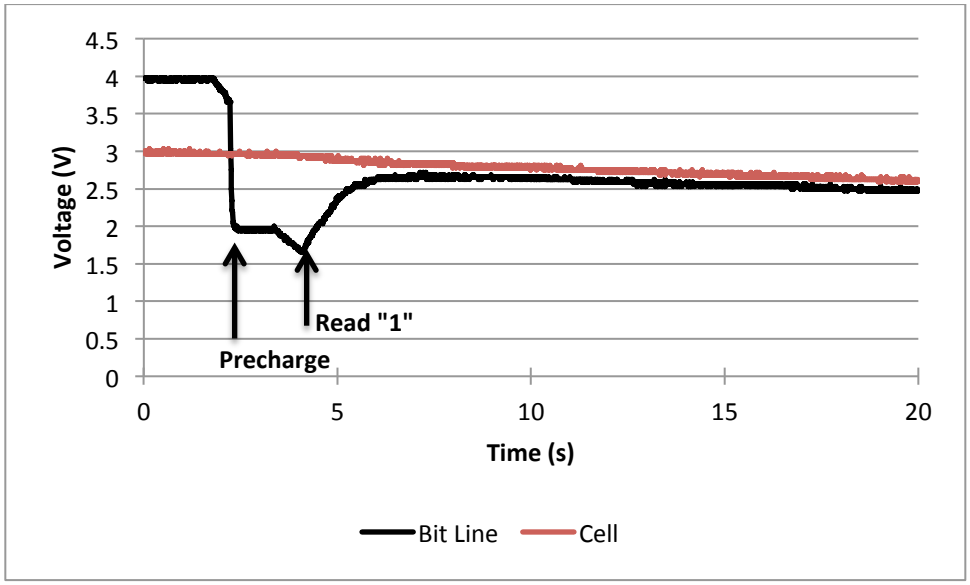

Figure 5.31 Reading a “1” from an ND1 MFSFET 1T-1C DRAM using negative gate voltages

The DRAM circuit performed similarly if the ND7 MFSFET was used rather than the ND1 MFSFET. While testing the ND7 transistor, another observation was made regarding negative voltages being applied at the gate. Typical test operations consisted of setting the gate to 4 V while writing, setting it to 0 V while precharging the bit line capacitor, and setting it to 4 V again in order to turn the transistor on and read its stored value. However, this process was altered to use 4 V while writing, -4 V while precharging the bit line, and 0 V while reading; in this case, the transistor was able to turn on at 0 V and allow the stored value to be read and placed on the bit line. Figure 5.32 and Figure 5.33 show the operations for reading a "1" and "0," respectively, and it was observed that the bit line did increase or decrease from its precharged value to successfully read the expected value when the gate voltage was changed from -4 V to 0 V.

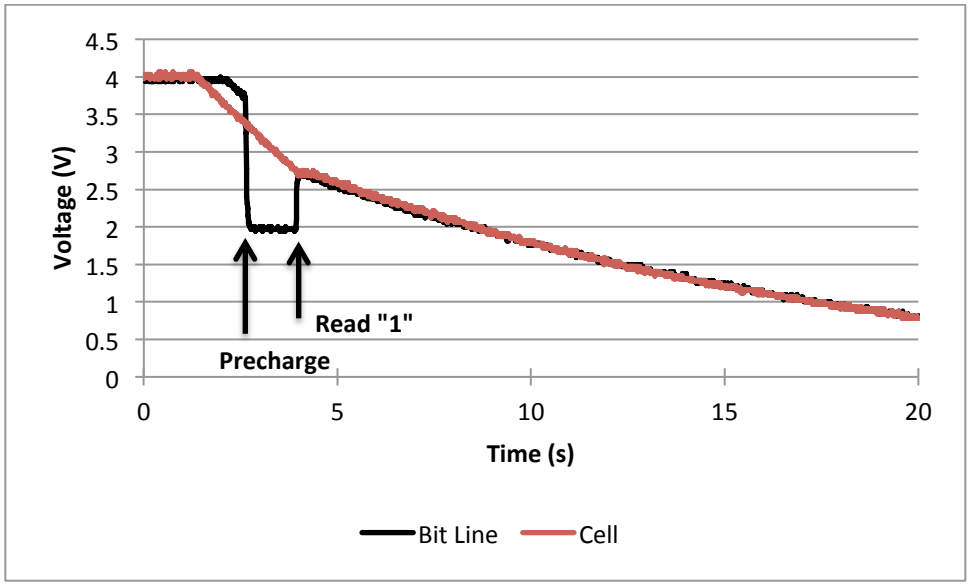

Figure 5.32 Reading a "1" from an ND7 MFSFET 1T-1C DRAM using -4 V and 0 V gate voltages

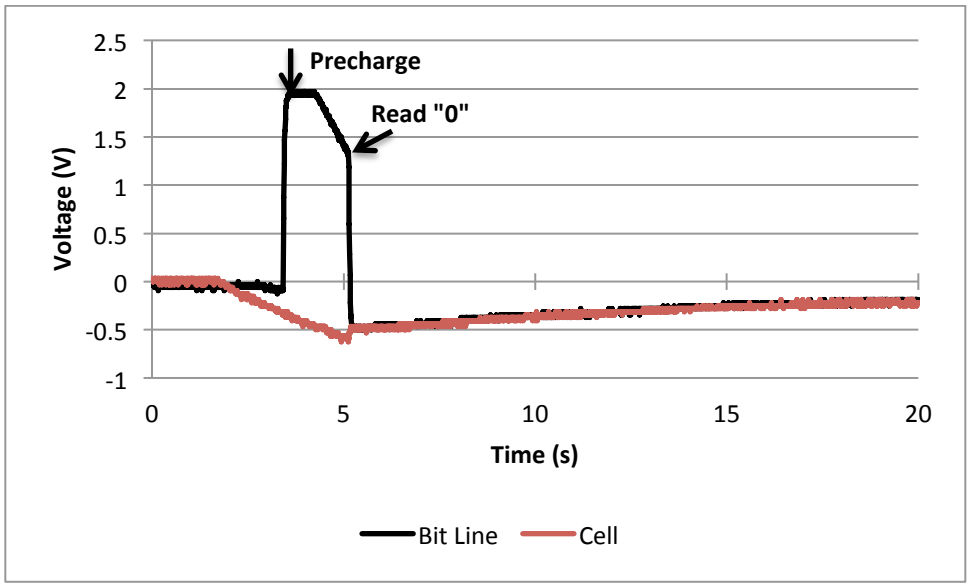

Figure 5.33 Reading a "0" from an ND7 MFSFET 1T-1C DRAM using -4 V and 0 V gate voltages

c. Analysis

Similar to the 3T DRAM circuit, the 1T-1C DRAM could also be successfully created using an MFSFET in place of a MOSFET. The circuit did exhibit odd behavior where the voltages on both capacitors would converge to around $V_{DD}/2$ after an operation where a "0" was read, but the bit line did initially decrease from its $V_{DD}/2$ precharge value in order to indicate that a "0" had been read. The use of negative voltages on the gate of the single transistor also showed that reading the cell contents was possible by setting the WL to 0 V during the read operation if WL had previously been set to -4 V during the time that the bit line was being precharged. Using this multi-tiered system of gate voltages at WL could potentially allow a circuit designer to alter their power requirements for such a memory cell or allow different variations of the memory circuit without changing the hardware.

Others have investigated the usage of the MFSFET in the DRAM circuit as a method for removing the requirement of the storage capacitor. While the research conducted so far indicates that the MFSFET in this role may not successfully produce a retention time long enough to be considered nonvolatile, the retention is more than adequate to modify the DRAM to remove the storage capacitor and retain a value for a longer period [23].

CHAPTER VI

MODELING THE FERROELECTRIC TRANSISTOR

The behavior of a traditional MOSFET can be described with a series of equations that can accurately predict current and voltage characteristics of the transistor. The presence of a hysteresis effect in a MFSFET makes it difficult to predict the behavior of the transistor. With a MOSFET, knowing the voltages at the drain, gate, source, and substrate, as well as knowing the general characteristics of the device, make it possible to use one or more equations to find the current flowing through the device. Knowing such inputs and characteristics of a MFSFET is not sufficient to determine its current, as previous inputs will have an impact as well. Therefore, it is necessary to use modeling techniques in order to predict the current in a MFSFET.

A. Background of Existing Models

Popular models often reference research by Miller and McWhorter [24] in their attempts to predict the impact that the hysteresis effect will have on the device. Such models rely on variations of the hyperbolic tangent function, the shape of which can be manipulated into a shape similar to a hysteresis curve; Figure 6.1 shows an arbitrary hyperbolic tangent curve formed with two different offsets in order to create a hysteresis loop.

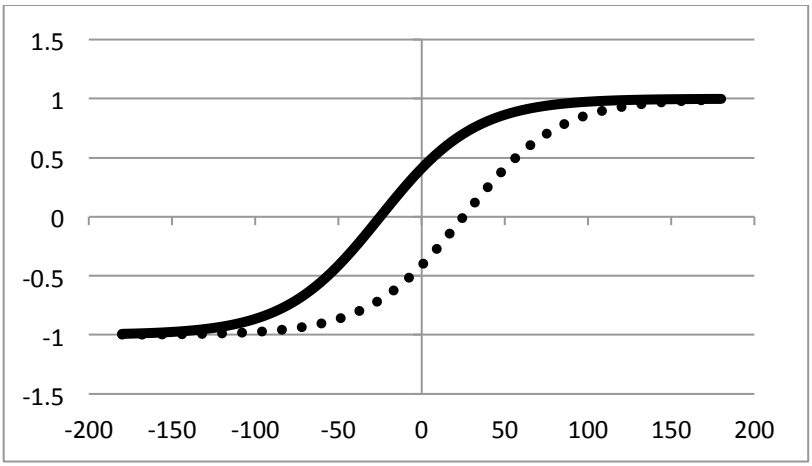

Figure 6.1 Hyperbolic tangent hysteresis loop

Of particular interest is their determination of a few equations that are used to calculate the dipole polarization in a ferroelectric material, shown below in (6.1) through (6.3) [25].

$$P_{sat}^{+}(V) = P_0(1 + \alpha V)\left[\tanh\left(\frac{V - V_C}{2V_0}\right)\right] \tag{6.1}$$

$$P_{sat}^{-}(V) = P_0(1 + \alpha V)\left[\tanh\left(\frac{V + V_C}{2V_0}\right)\right] \tag{6.2}$$

$$V_0 = V_C\left(\frac{1+\frac{P_R}{P_S}}{1-\frac{P_R}{P_S}}\right)^{-1} \tag{6.3}$$

In these equations, the variables are: P_{sat}, the saturated polarization function; P_S, the hysteretic saturation polarization value; P_R, the remnant polarization value; V_C, the coercive voltage value; V, the applied voltage; and α, a field-effect parameter. Bailey further manipulated these equations to find (6.4) for active-mode polarization values and (6.5) for remnant-mode polarization values [25].

$$P(V) = \frac{1}{2}\left[P_S + P_{sat}^{\pm}(V)\right] \tag{6.4}$$

$$P(V) = \frac{1}{2}\left[P_S - P_{sat}^{\pm}(V)\right] \tag{6.5}$$

Bailey presented a model to simulate the current flowing through a ferroelectric transistor [17], [25]. His model was a C++ program that contained a defined range of input voltage values for the gate voltage of the transistor, and allowed the user to customize the parameters of the transistor with an input file. For each input value, the input voltage value range was iterated multiple times, rather than a single time. The results of the last iteration were used as the results of the simulation, considering these values to be the modeled active and remanent mode currents of the transistor.

Bailey's model contained three distinct elements: the simulation driver, the transistor being simulated, and the fictional partitions that make up the ferroelectric layer for modeling purposes. MacLeod and Ho [17] explain that the reason for considering the ferroelectric layer as a series of partitions is that each area of the ferroelectric material will be under different conditions. For example, the semiconductor surface potential would vary from the source to the drain. The program also had the capability of providing one of two types of simulations: a hysteresis mode that investigated the impact of a varying input voltage on the transistor current, as well as a retention mode that would model the effect of an applied and removed input voltage over a period of time; the mode being used needed to be determined at compile time rather than at run time.

Further modifications were made to Bailey's model that allowed the transistor to be simulated as part of an SRAM cell circuit rather than as an individual element. The SRAM cell was constructed as having a traditional n-channel MOSFET as the driver transistor on the left side of the circuit, a ferroelectric transistor as the driver on the right side, and resistors as loads on both sides. Numerical bisection methods were used to iteratively determine the current in the circuit. Once a voltage was applied to the gate of

the ferroelectric transistor, the voltage at the node between this transistor and its load resistor was changed until a current solution could be found through the resistor and transistor. This drain voltage was then used as the gate voltage to the MOSFET on the left side of the circuit, repeating the same bisection method to find a drain voltage that made the currents through the resistor and MOSFET equal.

B. Development of a New Model

The initial goal of the model presented in this thesis was to use Bailey's model as a basis for a more complete model that could be expanded to simulate additional digital circuits including MFSFETs. However, due to the differences in behavior seen even in transistors of the same type, as well as the lack of some parameters of the transistors, an empirical model was created. The simulation program was written with the goal of providing for easier expansion of features and simulation circuits. A key objective in rewriting the program was to make the code more modular so that devices could be visualized in blocks and connected together into more complex circuits by creating new instances of these blocks and defining the behavior of how they would interact with one another. Bailey's program modeled the behavior of a ferroelectric transistor connected to no other components or an SRAM cell with the ferroelectric transistor. Rewriting this model allowed a program that provided choices of which devices and circuits to simulate, building on the ideas of the previous models.

The general current hysteresis loop of the ND1 transistor is shown in Figure 2.13. Figure 6.1 shows that the hyperbolic tangent function is useful for creating such a hysteresis loop, but another option is to use the Fermi-Dirac function. The Fermi-Dirac distribution function is a function that determines the probability that an energy state is

filled by an electron at a given energy E [26]. The equation for the function is shown in (6.6), and Figure 6.2 shows a sample plot of the function for two different arbitrary values of kT. When considering the Fermi-Dirac function as the basis for this model, it is important to note that it is merely the shape of the function that is of interest, with no actual application of the probability being used in the modeling of the transistor.

$$f_D(E) = \frac{1}{1+e^{\frac{E-E_F}{kT}}} \tag{6.6}$$

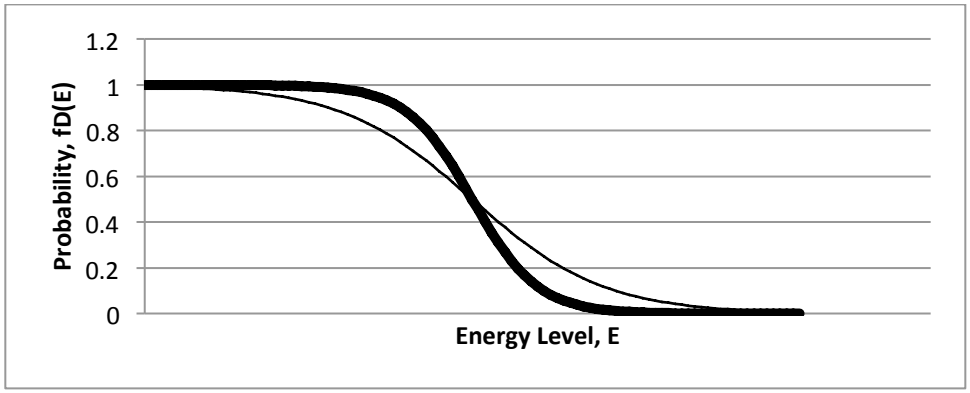

Figure 6.2 Sample Fermi-Dirac distributions

The full current hysteresis loops for the ND1 transistor, shown in Figure 2.13, were observed to note the characteristics of the current curve. Note that the ND1 transistor was primarily used for modeling, rather than the ND7 transistor, but the model was also used to simulate the ND7 transistor just to verify that the same principles applied. Equations (6.7) and (6.8) were developed to describe the shape of the current curve for increasing and decreasing V_{GS} values, respectively.

$$I_D^+ = I_{max}\left[1 - \frac{1}{1+\exp\left(\frac{V_{GS}+V_C-V_Z}{V_{SF}}\right)}\right] \tag{6.7}$$

$$I_D^- = I_{max}\left[1 - \frac{1}{1+\exp\left(\frac{V_{GS}-V_C-V_Z}{V_{SF}}\right)}\right] \tag{6.8}$$

The drain current of the transistor, calculated as I_D^+ or I_D^- depending on the direction of the gate voltage, is a scaled value of the maximum current, I_{max}, which can actually vary per transistor as seen in laboratory experiments. The maximum current corresponds to the highest value where the current begins to flatten in Figure 2.13. The value V_C is the coercive voltage of the transistor, found by measuring the width of the current hysteresis loop and dividing in half. The variables V_Z and V_{SF} are additional values used to fit the equations to the measured current of the transistor; V_Z, set to 1.6, is a shifting factor used to move the entire plot to the right, and V_{SF}, set to 0.73, is not a true voltage but rather a scaling factor used to adjust the slope of the curves. Figure 6.3 shows a comparison of measured current data from the ND1 transistor with modeled current values found with (6.7) and (6.8), using a range of gate voltages from -6 V to 6 V and back down to -6 V in 0.1 V increments.

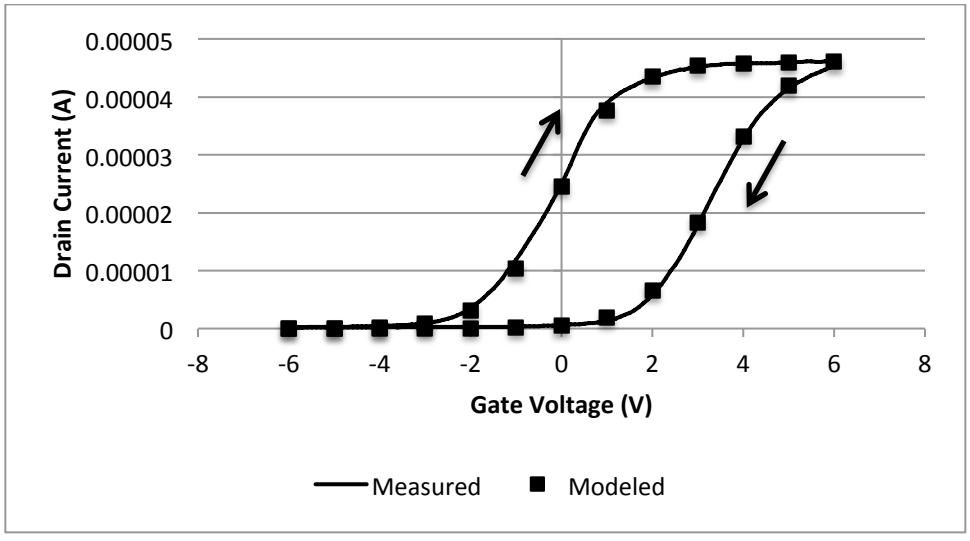

Figure 6.3 Measured and modeled current hysteresis loop for ND1 transistor

While the model current equations produce an acceptable replica of the measured current data, it is not complete because it assumes that only full loops ranging from gate voltages such as -6 V to 6 V will be used. Therefore, calculating inner loops of the current would require modified versions of these equations, found in (6.9) and (6.10).

$$I_D^+ = I_{max}\left[1 - \frac{1-\frac{I_{lower}}{I_{max}}}{1+\exp\left(\frac{V_{GS}+V_C-V_Z+V_{offset}}{V_{SF}}\right)}\right] \tag{6.9}$$

$$I_D^- = I_{upper}\left[1 - \frac{1-\frac{I_{min}}{I_{upper}}}{1+\exp\left(\frac{V_{GS}-V_C-V_Z+V_{offset}}{V_{SF}}\right)}\right] \tag{6.10}$$

The modified equations present no new user-specified values, but consist of variables that are changed dynamically by the modeling program in order to manipulate or shift the current curves based on the state of the transistor. The I_{upper} variable is used to create an imaginary curve that extends above I_{max} in order to create sharp decreases for cases where the gate voltages from an increasing to decreasing direction without fully polarizing the transistor in the positive direction. The I_{min} value is typically held to 0 for transistor-only simulations, but is used in more complex circuits to assert a non-zero minimum current in the circuit, based on collected data for inverter testing. Similar to I_{min}, I_{lower} is used for shaping the bottom of the curve, with emphasis on maintaining an appropriate minimum current for minor loops in the hysteresis. The only voltage term added in the equations, V_{offset}, is used to shift the current curve in the positive direction for cases where the gate voltage begins at a non-polarizing voltage or changes from decreasing to increasing direction without having become low enough to negatively polarize the transistor. Testing showed that in cases where the initial gate voltage was 0 V and increased to 6 V followed by being decreased back to 0 V, the current curve

retraced itself as the gate voltage decreased. The modified equations (6.9) and (6.10) account for cases such as this. Specific examples are shown later in this chapter.

C. General Model Execution

The model created is a multi-platform, command-line application that simulates three circuit types: the standalone MFSFET, the resistive load inverter with an MFSFET driver transistor, and the resistive load SRAM cell with MFSFET driver transistors. Each of these circuits is discussed in further detail later in this chapter. The application reads an input text file, where each line of the file contains a gate voltage to be applied to the MFSFET in the circuit, and provides an output text file of comma-separated values that include the original input gate voltage and, depending on the circuit being simulated, drain currents or drain voltages of the MFSFETs involved.

Regardless of the type of circuit being simulated, the application follows some basic steps on every execution. As the program begins, it executes a function, ProcessArguments(), that checks that command line parameters are provided as expected. There are three expected arguments: the circuit type to be simulated, the path to the text file containing input voltages, and the path to the file that will contain any resulting output values. After verifying this, the input and output files are opened for reading and writing, respectively, so that the application has access to them. The application then calls DevicePrefs::ReadDevicePrefs() to open and load a preferences file, which should be called deviceprefs.ini and located in the same directory as the executable. This file contains parameters for the MFSFET, as well general information about static node voltages that are not varied for a particular simulation, such as V_{DD} and V_S in an inverter circuit. The specified circuit is then simulated, the input and output files are closed, and

the application exits. If the program detects errors along the way, particularly in file operations, it exits with an appropriate return status.

D. Simulating the Standalone Ferroelectric Transistor

To understand how each circuit is modeled by the simulation program, it is necessary to understand how the ferroelectric transistor itself is modeled. The device is written as a C++ class so that it appears as a black box, except for functions that allow access to read from and write to the terminals of the transistor, as well as functions to initialize the device or update its state and polarization.

Before the device is used, it is initialized with its Init() function, which uses values from the deviceprefs.ini file and other predefined values to set up the parameters of the transistor, such as the coercive voltage, maximum drain current, and channel dimensions. In order to establish the expected behavior of the transistor, the maximum current for V_{DS} = 3 V should be measured and specified as the "normcurrent" parameter in the device preferences file. The current is then scaled accordingly during program execution for other V_{DS} values, since the maximum current was seen to be linear for varying V_{DS}, as shown in Figure 2.13. Then the drain, source, and substrate voltages are set; these values are also specified in the deviceprefs.ini file. Next, a loop is executed while each line of the voltage input file is read. For each input voltage value specified, the voltage is applied to the gate of the transistor, the RecalcCurrent() function of the transistor is called, and the active mode current is determined.

There is a special case where, instead of specifying a numerical gate voltage input, the user can instead specify that a "ppulse" or "npulse" be applied to the gate, for positive and negative poling voltages, respectively. In this special case, the magnitude of

the pulse is read from the device preferences file using the "pulsevoltage" parameter, and multiple steps are taken automatically to ensure that the transistor is poled appropriately without forcing the user to manage this process. The input voltage and transistor drain current are then written to the output file.

The most complex logic of simulating the MFSFET takes place in the transistor's RecalcCurrent() function. This function contains a series of conditional statements used to determine the correct value to use for the V_{offset} variable. These statements are written in such a way to base this decision on three conditions: the increasing or decreasing direction of the gate voltage with respect to the last gate voltage value, the increasing or decreasing direction of the gate voltage prior to the newest gate voltage being applied, and the location of the gate voltage within three ranges of inputs. The first two conditions are considered so that the model can determine if the gate voltages are continuing an increasing or decreasing trend, or if the gate voltage has changed direction and the model therefore needs to adjust the dynamic parameters of (6.9) and (6.10). The third condition is monitored in order to determine if the gate voltage is either positive or negative enough to induce a certain polarization state, or if it falls within a range where the transistor is more sensitive to the actual value of the applied gate voltage. This range is bounded by the values $(-V_C - V_Z)$ and $(V_C - V_Z)$. These bounds were selected based on observation of collected data. The various conditions that must be considered are listed in Table 6.1.

Table 6.1 Decision logic of the MFSFET model based on gate voltage

Previous V_{GS} Direction	Current V_{GS} Direction	Range of V_{GS}	V_{offset} Modifications
decreasing	decreasing	any value	no change
decreasing	increasing	$V_{GS} \leq (-V_C - V_Z)$	$V_{offset} = 0$ V
decreasing	increasing	$(-V_C - V_Z) < V_{GS} \leq (V_C - V_Z)$	$V_{offset} = -2V_C + abs(V_{GS}) - abs(V_C - V_Z)$
decreasing	increasing	$V_{GS} > (V_C - V_Z)$	$V_{offset} = -2V_C$
increasing	decreasing	$V_{GS} \leq (V_C - V_Z)$	no change
increasing	decreasing	$V_{GS} > (V_C - V_Z)$	$V_{offset} = 0$ V
increasing	increasing	any value	no change

After determining the V_{offset} value, the CurrentCurve() function of the transistor is called from within the RecalcCurrent() function. This function applies (6.9) or (6.10) based on its provided arguments.

The figures beginning with Figure 6.4 and ending with Figure 6.8 display modeled current generated by this program in comparison with equivalent measured data for the standalone ND1 transistor for five different drain-to-source voltage values. In each case, V_{GS} was varied by beginning at 0 V, increasing in 0.1 V increments to 6 V, decreasing incrementally to -6 V, and increasing again incrementally to 6 V. To reduce confusion on the plots and only focus on the outermost loops of the current, the initial data from 0 V to 6 V was omitted, so that the data begins with 6 V, decreases to -6 V, and increases again to 6 V.

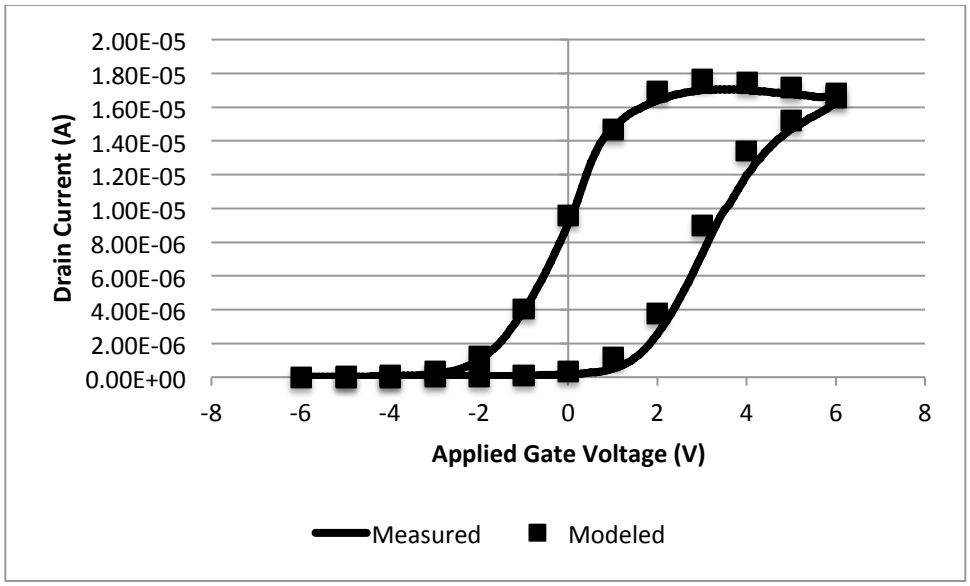

Figure 6.4 Modeled vs. measured ND1 current for V_{DS} = 1 V

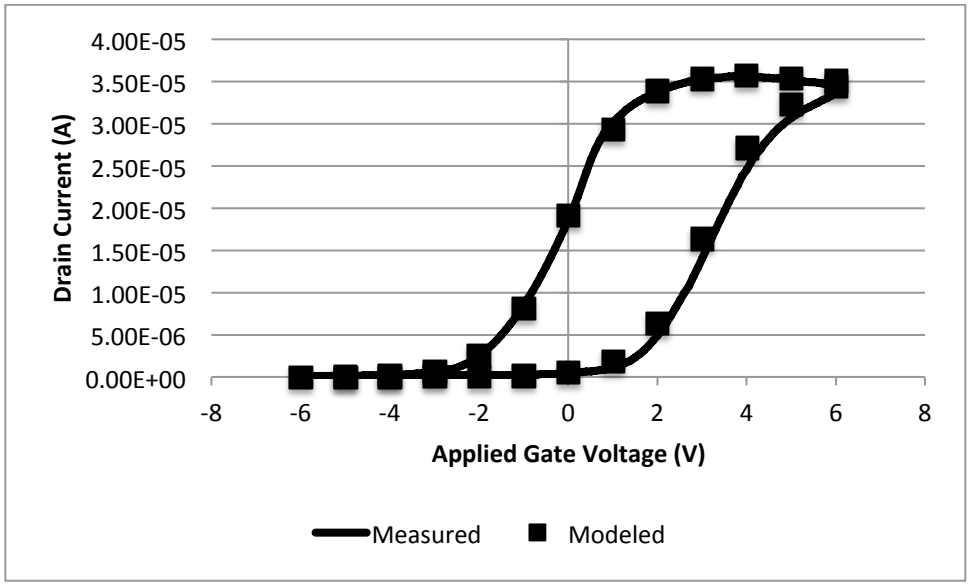

Figure 6.5 Modeled vs. measured ND1 current for V_{DS} = 2 V

The maximum current for a transistor at V_{DS} = 3 V is used as a baseline current for each transistor, so the modeled data should match the measured data at V_{DS} = 3 V more closely than at other drain-to-source voltages. This is demonstrated in Figure 6.6.

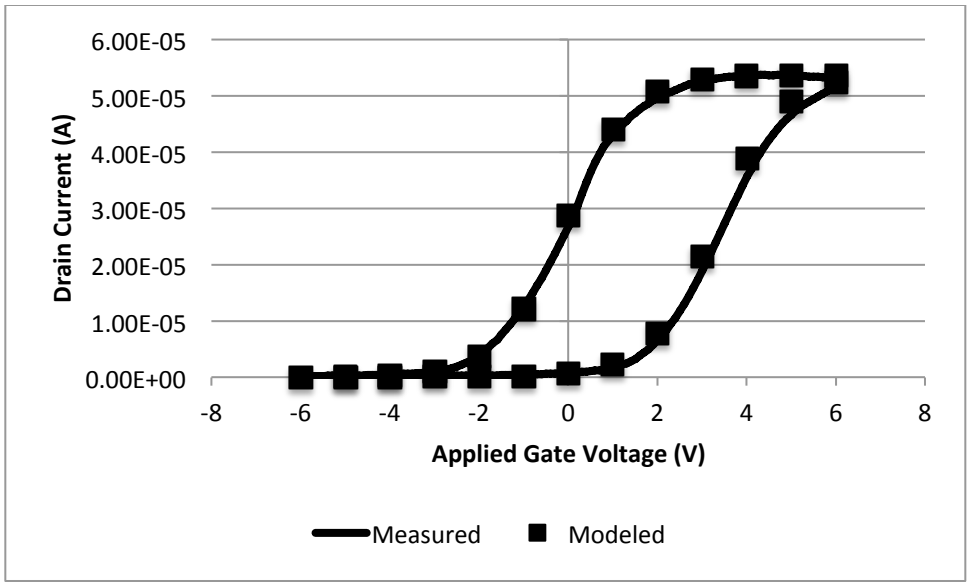

Figure 6.6 Modeled vs. measured ND1 current for V_{DS} = 3 V

In Figures 6.7 and 6.8, the modeled transistor current still follows the overall shape of the measured current at higher drain-to-source voltages, but shows a slight loss of detail at the transition points around V_{GS} = 1 V.

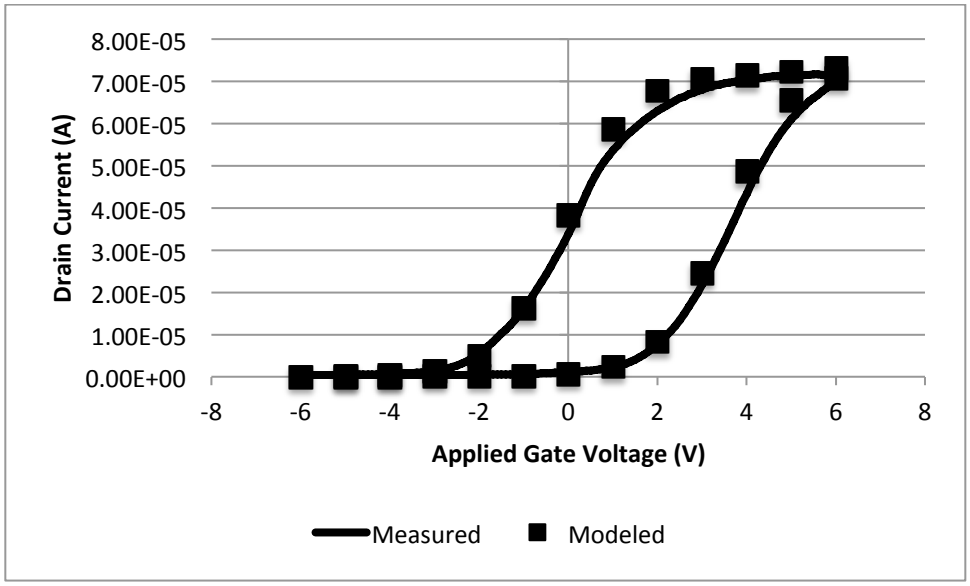

Figure 6.7 Modeled vs. measured ND1 current for V_{DS} = 4 V

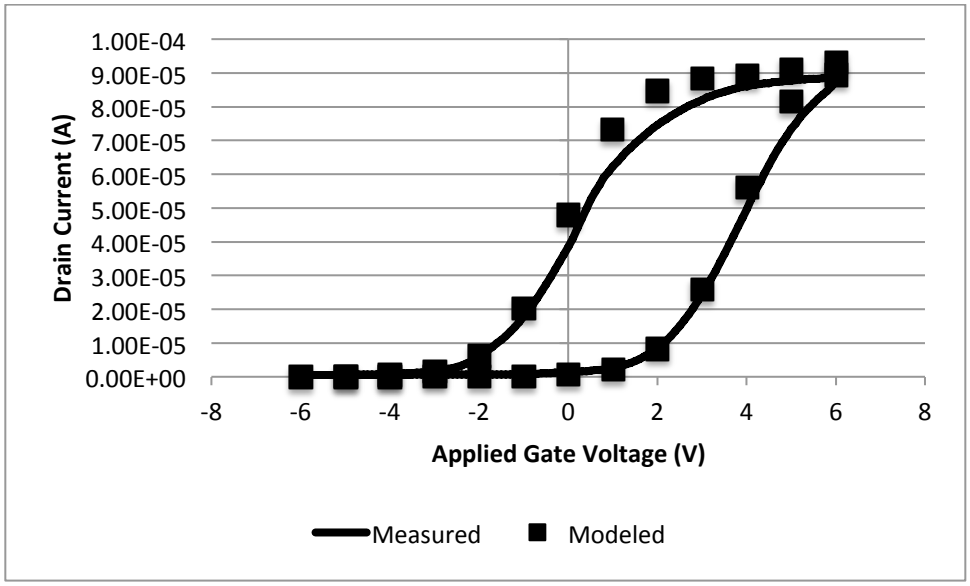

Figure 6.8 Modeled vs. measured ND1 current for V_{DS} = 5 V

For the 0 V to 6 V initial input range, the measured data would not completely lie on the outer hysteresis loop, meaning that the model's minor loop calculations would be needed in order to successfully model the current for this condition. Figure 6.9 shows the resulting loop from this input range, displaying that the model calculations hold close to the measured data.

The model was also tested with the ND7 transistor and produced a less precise current curve, while still approximating the same general shape. As shown in Figure 6.10, the current appeared to increase linearly as it became more negative or positive, but the Fermi-Dirac-based equations used in the model did not account for these specific changes.

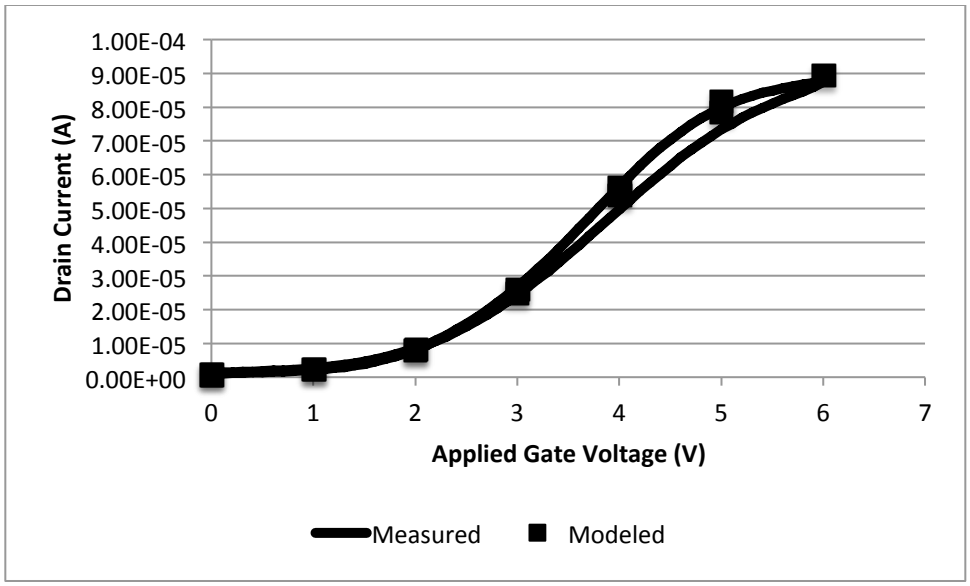

Figure 6.9 Modeled vs. measured ND1 current for V_{DS} = 5 V, minor loop

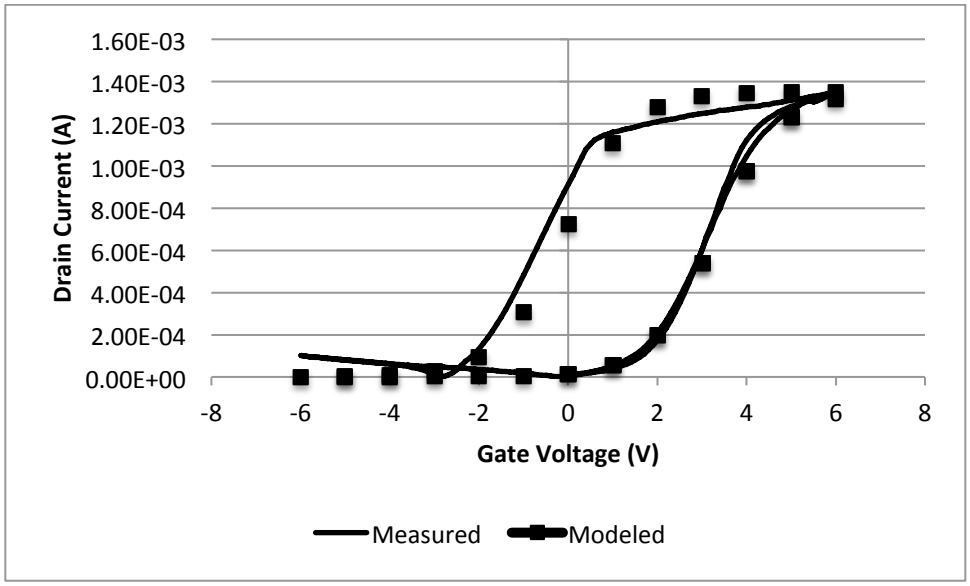

Figure 6.10 Modeled vs. measured ND7 current for V_{DS} = 1 V

E. Simulating the Resistive-Load Inverter

Being able to define and reproduce the behavior of the MFSFET makes it possible to simulate other circuits that include an MFSFET. A simple example of this is the

resistive-load inverter. Due to the introduction of the resistor into the circuit, additional unknown characteristics are introduced as well. The key concern of this circuit was determining a way to properly handle the drain voltage of the MFSFET, which is also the lower node of the resistor. A series of inverters was tested, varying both the V_{DD} supply voltage and the load resistance, to determine a pattern for predicting the minimum and maximum drain voltages that would be seen for the MFSFET. The high-logic output voltages, V_{OH}, for the inverter are shown in Table 6.2, followed by the low-logic output voltages, V_{OL}, in Table 6.3.

Table 6.2 V_{OH} for the MFSFET inverter based on varied supply and load values

V_{DD} (V)	R_L = 5 kΩ	R_L = 35 kΩ	R_L = 75 kΩ	R_L = 105 kΩ	R_L = 150 kΩ	R_L = 200 kΩ
2	2.031	n/a	1.906	1.844	1.765	1.719
3	3.047	2.906	2.828	n/a	2.672	2.546
4	4.016	3.875	3.813	3.657	3.484	3.329
5	5.016	4.859	4.656	4.563	4.406	4.203
6	n/a	5.828	5.578	n/a	5.204	4.985

Table 6.3 V_{OL} for the MFSFET inverter based on varied supply and load values

V_{DD} (V)	R_L = 5 kΩ	R_L = 35 kΩ	R_L = 75 kΩ	R_L = 105 kΩ	R_L = 150 kΩ	R_L = 200 kΩ
2	1.906	n/a	1.110	0.828	0.812	0.688
3	2.828	2.094	1.656	n/a	1.156	0.984
4	3.766	2.719	2.141	1.641	1.500	1.250
5	4.688	3.391	2.578	1.875	1.828	1.532
6	n/a	4.047	3.031	n/a	2.125	1.766

The effects of varying only one parameter, supply voltage or load resistance, rather than both usually resulted in linear changes in the output voltage with respect to

that parameter. Figures 6.11 and 6.12 show the linear relationship observed for V_{OH} and V_{OL} while varying V_{DD}.

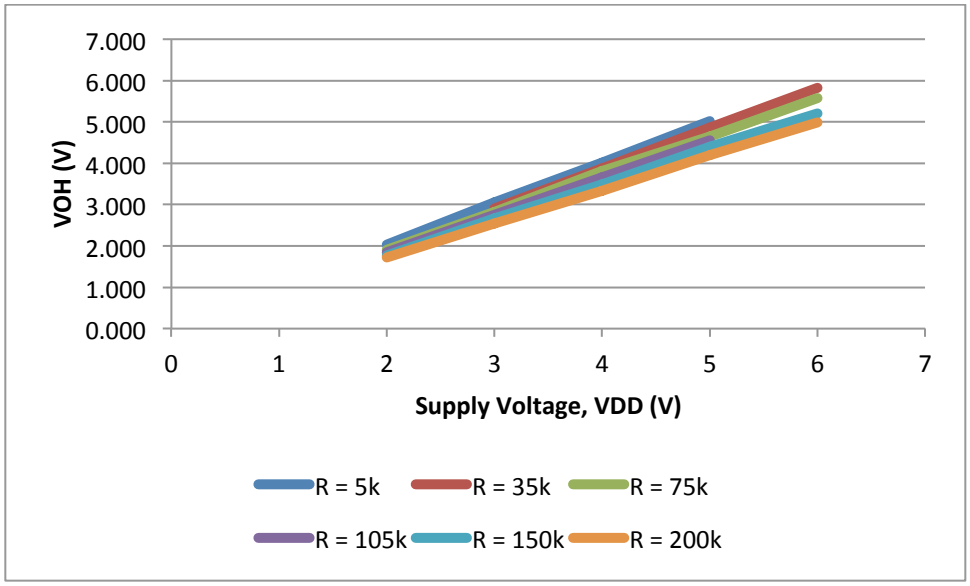

Figure 6.11 Behavior of inverter V_{OH} for varying V_{DD}

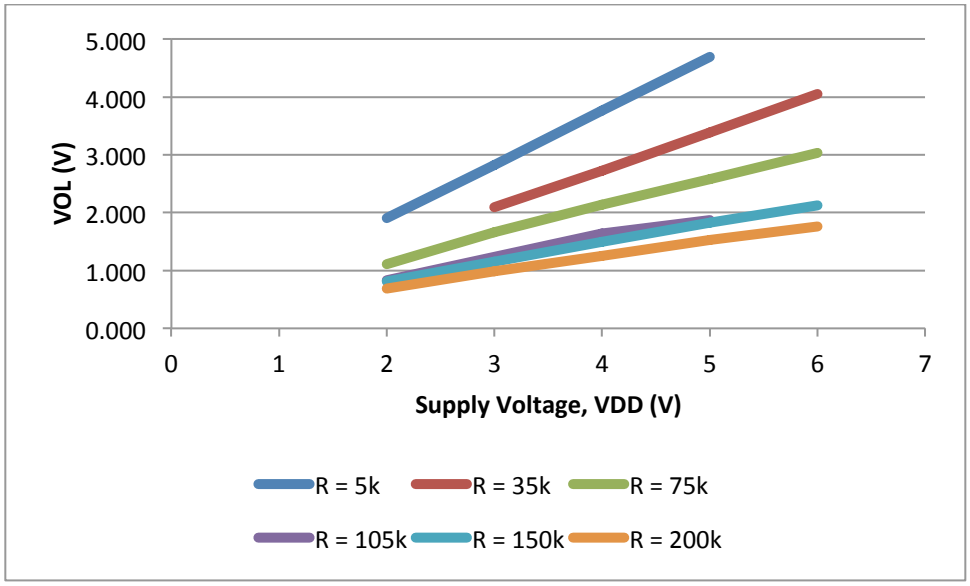

Figure 6.12 Behavior of inverter V_{OL} for varying V_{DD}

The relationship between V_{OH} and the load resistance of the inverter were also found to be linear, but V_{OL} showed less linear behavior at lower resistances. These are shown in Figures 6.13 and 6.14.

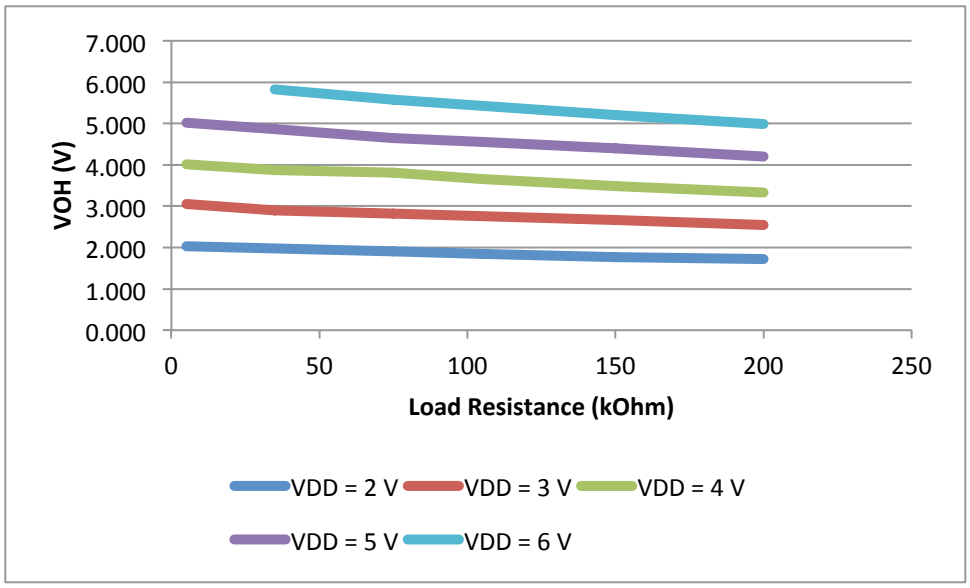

Figure 6.13 Behavior of inverter V_{OH} for varying load resistance

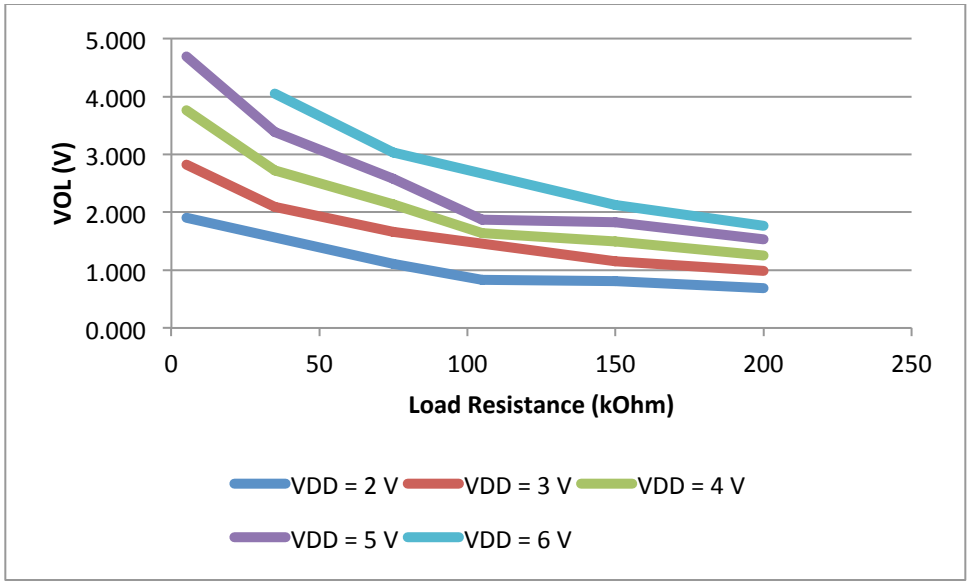

Figure 6.14 Behavior of inverter V_{OL} for varying load resistance

This calculation is complicated, however, by considering both parameters simultaneously. The data in Tables 6.2 and 6.3 were provided to a curve-fitting algorithm [27] that generated a cubic equation in form of (6.11).

$$z = a + bx + cy + dx^2 + fy^2 + gx^3 + hy^3 + ixy + jx^2y + kxy^2 \quad (6.11)$$

The x parameter represented the load resistance R_L, the y parameter represented the supply voltage V_{DD}, and the z output variable represented the predicted V_{OH} or V_{OL} value. A table of the coefficients calculated by the algorithm is shown in Table 6.4.

Table 6.4 Coefficients for curve-fitting cubic equation (6.11)

Coefficient	V_{OH}	V_{OL}
a	9.50E-02	1.29E-01
b	-3.81E-07	-1.00E-05
c	9.51E-01	9.44E-01
d	5.01E-12	1.49E-10
f	1.44E-02	1.32E-02
g	-1.71E-17	-4.86E-16
h	-1.35E-03	-3.10E-03
i	-9.48E-07	-7.49E-06
j	8.31E-13	1.83E-11
k	-1.30E-08	9.91E-08

To start simulating the inverter, it has an Init() function that must be called, and its only argument is the value of the resistance used for the load, which is found in the device preferences file. This function calls the Init() function of the MFSFET so that its parameters can also be initialized. This also tells the transistor that the circuit being used is an inverter and that the V_C and V_X voltages of the transistor should be altered to certain values to better reflect empirical results. After initializing the inverter and its member

transistor, the supply voltage V_{DD} and source voltage V_S are set on the circuit based on the values in the device preferences file. The final step before applying gate voltages is that the Init2() function of the inverter is called; this function uses (6.11) and the coefficients in Table 5.4 to determine V_{OH} and V_{OL} of the inverter circuit, and applies these voltages to determine the minimum and maximum current values that will be seen through the transistor.

As with the standalone MFSFET, the inverter can accept input values of "ppulse" or "npulse" to apply a positive or negative poling pulse to the gate of the MFSFET, or it can accept a numerical value to be applied as a gate voltage. The Update() function of the inverter is then called, which uses (6.9) and (6.10) to determine the current through the transistor, and then combines this with the load resistance and supply voltage to calculate the output voltage of the inverter circuit. The simulator outputs the applied gate voltage, drain current, and output voltage of the circuit to each line of the output file.

In comparing the modeled data to the measured data, one can see that the model accurately described the behavior of the resistive-load inverter when using the ND1 transistor. Figure 6.15 shows a VTC of the inverter for the case where V_{DD} was set to 4 V, the load resistance was 105 kΩ, and the input signal was a triangle wave of 12 V_{PP} with a 0 V offset, effectively creating a -6 V to 6 V input range.

The effect of the high and low output voltages shown in Tables 6.2 and 6.3 is apparent in Figure 6.15, and (6.11) helped to accurate predict these voltage levels. The current for the same circuit is shown in Figure 6.16, showing only minimal variation in the measured and modeled current when reaching the positive saturation voltages.

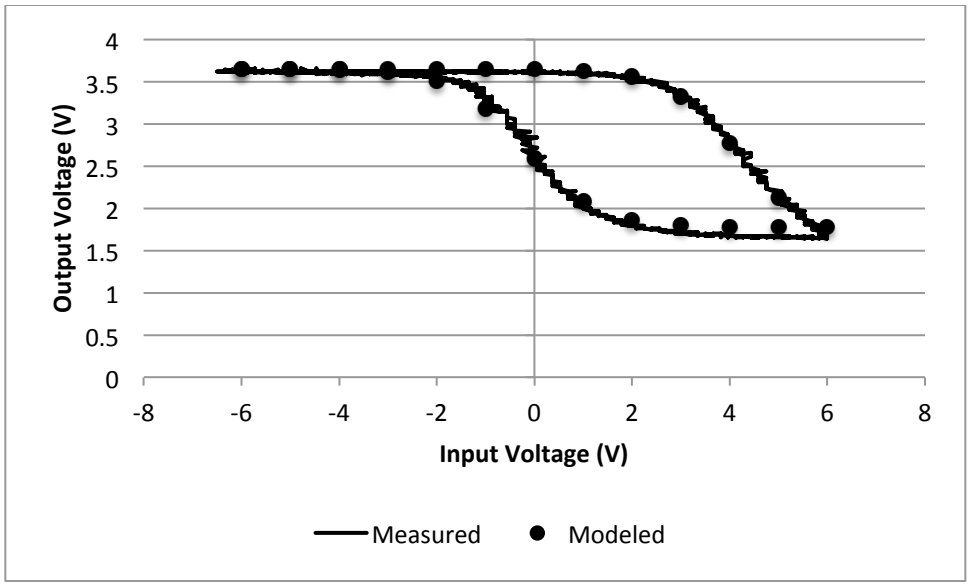

Figure 6.15 Modeled output voltage of ND1 inverter for V_{DD} = 4 V, R_L = 105 kΩ

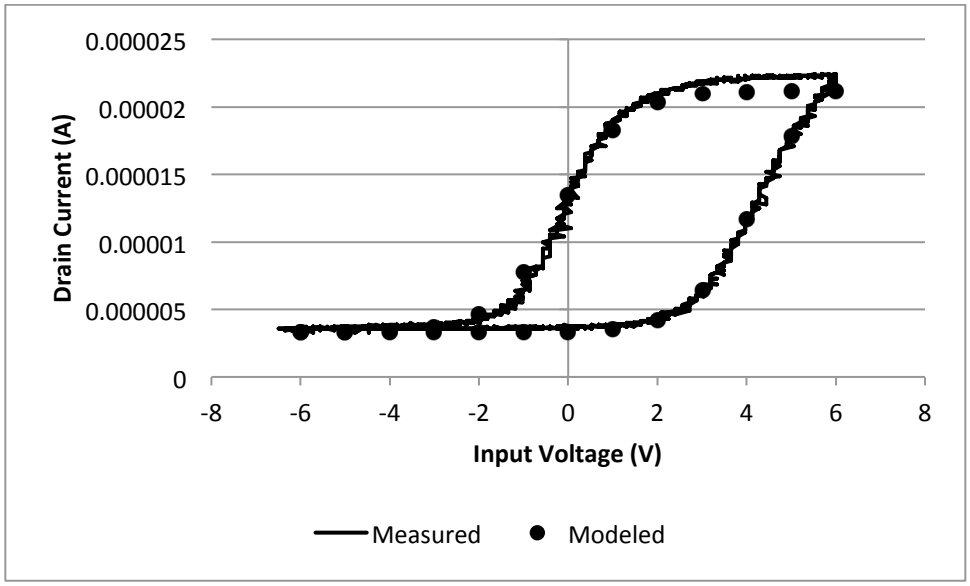

Figure 6.16 Modeled drain current of ND1 inverter for V_{DD} = 4 V, R_L = 105 kΩ

Using a load resistance of only 5 kΩ did not result in a usable inverter for the ND1 transistor, but the model could still accurately predict this non-ideal behavior, as shown in Figure 6.17.

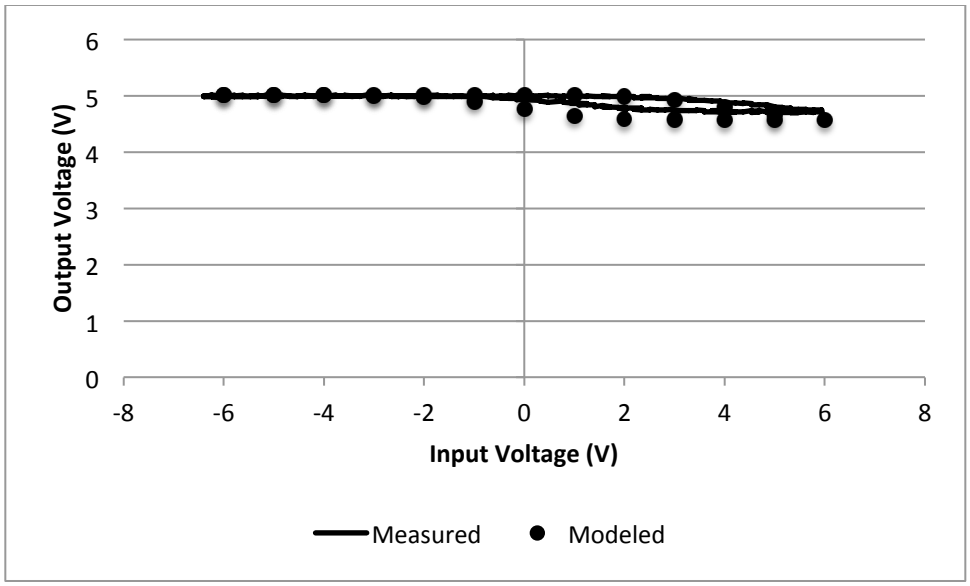

Figure 6.17 Modeled output voltage of ND1 inverter for V_{DD} = 5 V, R_L = 5 kΩ

F. Simulating the SRAM Circuit

The SRAM circuit posed a larger challenge for modeling than the standalone MFSFET or the resistive load inverter, due to the presence of two ferroelectric transistors. For simplicity, and relative comparison to the inverter, a resistive load implementation of the SRAM cell is used here.

The general steps of the simulation of the SRAM are similar to the inverter at a high level. Initialization steps, with the SRAM's Init() and Init2() functions, are executed in order to setup the transistor objects and establish minimum and maximum current values, just as what was required for the inverter. The node voltages are also established. Then the program loops through all input voltage values and pulse commands found in the input text file, each time setting the V1 node voltage to the input voltage, calling the Update() function of the SRAM, and then using the IDLeft() and IDRight() functions of

the SRAM to return the current values found in each side of the SRAM cell. The input voltage and the two calculated current values are printed to the output file.

Inside the SRAM circuit's Update() function, the same steps are taken as with the inverter. The input voltage is applied to the gate of the right side transistor, and the current through that transistor is then calculated. Using this current and the load resistance value, the drain voltage of the transistor is found, and then applied to the left side transistor's gate. The current through the left side of the circuit is then calculated.

The modeled current for the right side of the SRAM circuit shows similarities to the experimental data measured, but those similarities are limited. Figure 6.18 shows this comparison.

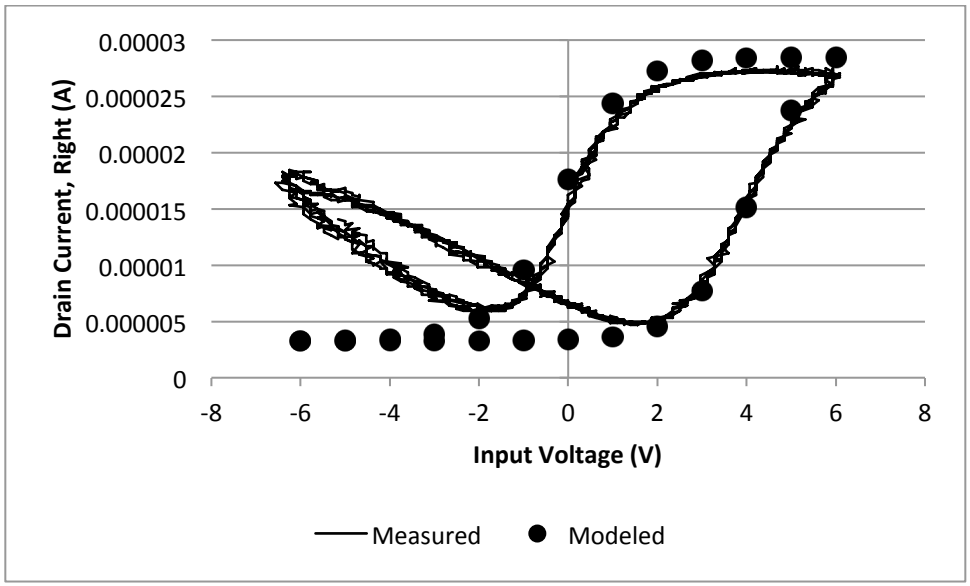

Figure 6.18 Modeled drain current of ND1 SRAM for V_{DD} = 4 V, R_L = 100 kΩ (transistor M2)

The model does not accurately reflect the current seen in the right side of the circuit for negative applied voltages, as it expects the current to eventually approach zero as it goes

more negative; Figure 4.6 shows, however, that the current does not always approach zero. For positive input voltages, however, the model can produce an accurate representation of the current. In order to see the hysteresis effect on the current, the input voltage was a 12 V_{PP} triangle wave at 1 Hz with 0 V offset; the load resistance was 100 kΩ, and V_{DD} was set to 4 V.

G. Performance of the Model

The performance of the model was measured for a number of inputs increasing by an order of 10 for each of the three circuit types, from 1000 input voltages to 100 million inputs. The measurements were taken on an Apple MacBook Pro laptop computer with a quad-core 2 GHz Intel Core i7 processor and 8 GB of memory. The results show that for a set of input values sized at 100,000, the model could generate results in less than one second. Even for an extremely large set of inputs ranging up to 100 million, results could be obtained in only a few minutes.

Table 6.5 Performance of model for varying number of inputs

Inputs	MFSFET	Inverter	SRAM
1.0E+03	0.01s	0.01s	0.01s
1.0E+04	0.02s	0.03s	0.03s
1.0E+05	0.17s	0.31s	0.23s
1.0E+06	1.69s	2.21s	2.39s
1.0E+07	16.56s	21.21s	22.41s
1.0E+08	159.67s	219.33s	237.33s

The execution time for each circuit of the model is consistent with the fact that the standalone MFSFET is the most basic of the circuits, with the inverter adding a small layer of complexity, and the SRAM further extending that complexity. Overall,

performance shows a linear relation with the number of inputs provided, as demonstrated in Figure 6.19, which is sensible given that there is not much overhead per circuit in the model, and most of the processing takes place for a given input voltage value. Increasing the number of input values will therefore increase the amount of work required at the same rate, since each input has to be processed separately.

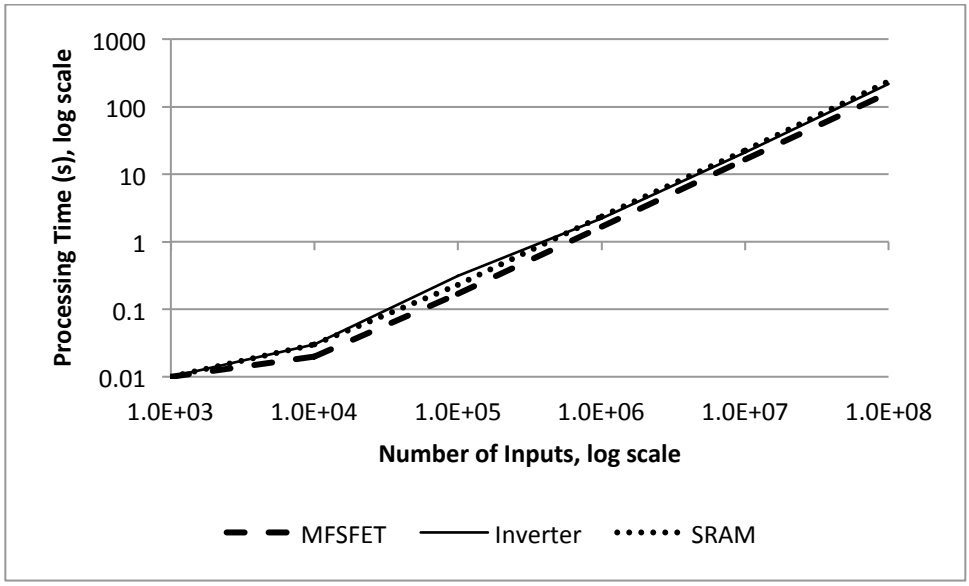

Figure 6.19 Model performance relative to input dataset size

H. Accuracy of Model

The accuracy of the model was examined for selected examples of modeled data presented previously in this chapter. To measure this accuracy, the percent difference equation shown in (6.12) was applied to samples of measured and modeled current data for the MFSFET, inverter, and SRAM.

$$\%_{diff} = 2\times\left(\frac{|I_{measured}-I_{modeled}|}{|I_{measured}|+|I_{modeled}|}\right) \quad (6.12)$$

For the MFSFET in a standalone configuration, Figure 6.20 shows the percent difference when VDS was set to 3 V and the gate voltage varied between -6 V and 6 V in a complete loop. The green data points on the figure indicate that period during which the transistor was turned on resulted in accurate modeling calculations, with a difference of less than or equal to 15%. The red data points indicate a greater difference, but are misleading at a first glance. The large percent differences observed were at times when the transistor was in an off state; at these times, the magnitude of the current was near-zero, so small that any variation would result in a large percent difference. For the favorable measurements of 15% difference or less, the average difference was only 4%.

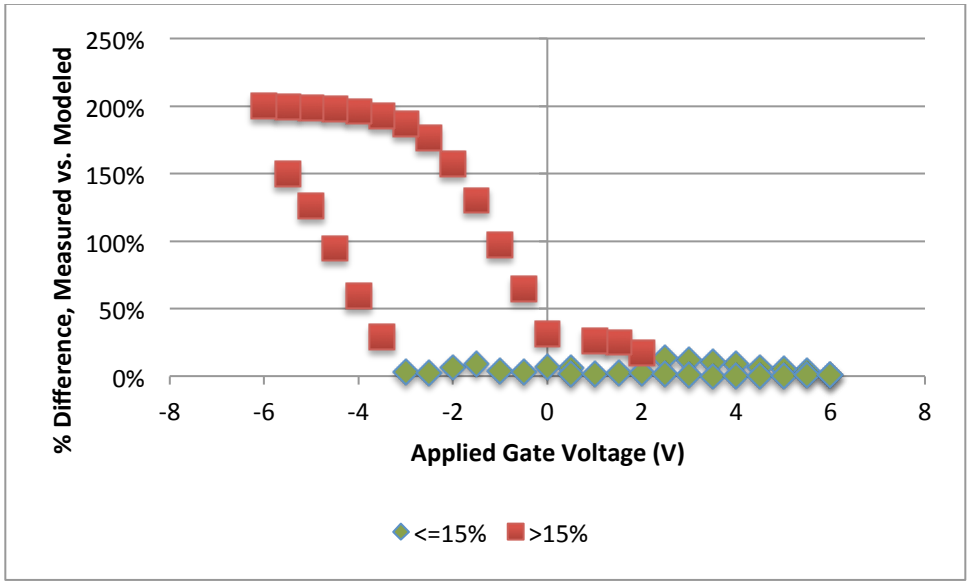

Figure 6.20 Percent difference for measured and modeled current in MFSFET with V_{DS} = 3 V

The modeled data of the inverter was compared to the measured data in the same way as the standalone MFSFET, with positive results observed. For the resistive load inverter with V_{DD} = 4 V and R_L = 105 kΩ, there were only a few areas of concern where

the percent difference exceeded 15%. These points occurred as the transistor was turning on and current was only beginning to flow through the circuit. The remaining results were favorable, with slightly higher differences shown for the off state of the transistor at negative input voltages when compared with the on state at positive input voltages. The average difference for the results at 15% or less was found to be 6%. These results are shown in Figure 6.21.

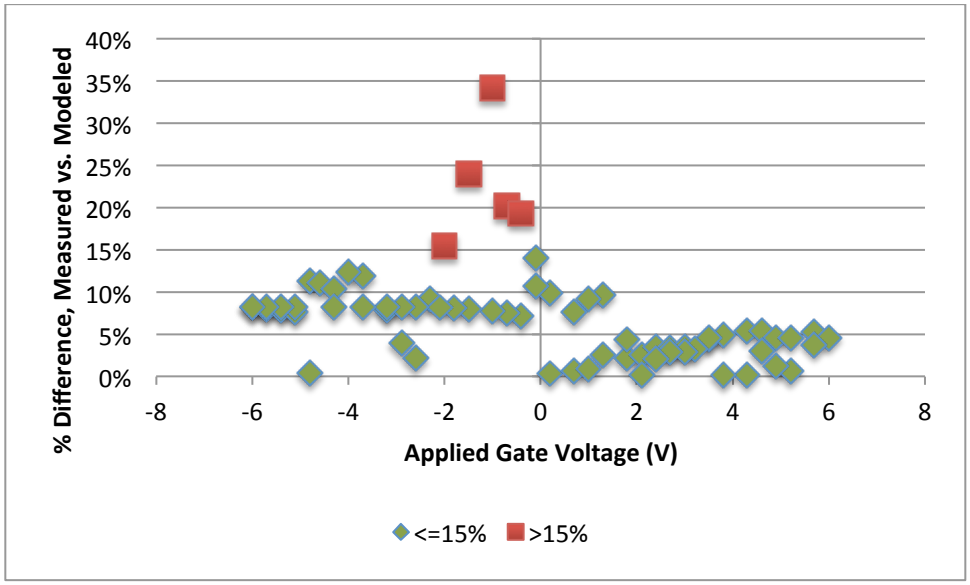

Figure 6.21 Percent difference for measured and modeled current in MFSFET inverter with V_{DD} = 4 V, R_L = 105 kΩ

Figure 6.22 presents percent difference data for the MFSFET-based SRAM circuit with V_{DD} = 4 V and R_L = 100 kΩ. The current through transistor M2 was the subject of observation. The percent difference found for the on state of the transistor shows that the model approximation of the current is suitable. The average percent difference for data points at or below 15% was 6%. For the higher differences, there are two potential causes. The first potential explanation for the large difference at negative voltages is that

the transistor may have behaved similar to the standalone MFSFET in Figure 6.20, where a small variation in extremely small currents was detected as a large percent difference. The second, and more likely, cause of differences between measured and modeled data is the current noticed in Figure 6.18 as the gate voltage on M2 became increasingly negative.

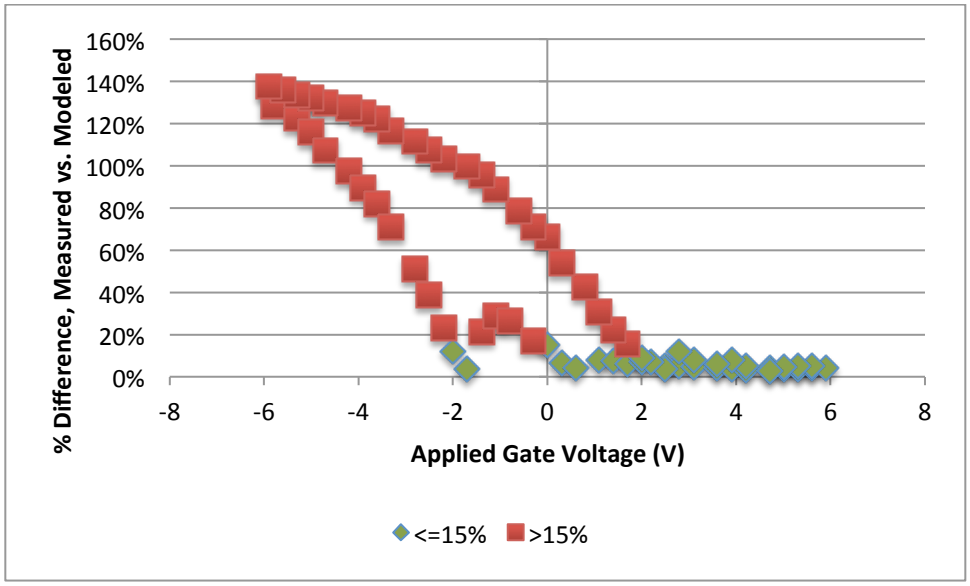

Figure 6.22 Percent difference for measured and modeled current in MFSFET with V_{DD} = 4 V, R_L = 100 kΩ (transistor M2)

I. Analysis

The model provided here was shown to be accurate for the available transistors, but highly sensitive to the current capabilities of each transistor tested. However, this is not a limitation of the model, but rather an acknowledgement that even multiple transistors of the same type, such as two separate ND1 transistors, did not appear to perform in the same manner. For example, in testing, one ND1 transistor was shown to operate at a maximum current of approximately 46 μA, while another ND1 transistor

operated at just over 60 μA, for a fixed V_{DS} value. Because of the differences seen in the devices that were tested, this model required a level of flexibility in order to adapt to these differences.

While the strong point of this model is that it is flexible in order to accommodate the transistors, its weakness lies in the fact that it needed to be based on empirical data in order to simulate the behavior of the transistors. Ideally, this model should be capable of reproducing the behavior of the transistor based on the physics of the device, and then extended with minimal effort to simulate other circuits such as the inverter or SRAM cell. In this case, the behavior of the transistor alone required empirical data in order to establish that the current followed a path similar to the shape of the Fermi-Dirac function. In addition, adding the inverter to the model required additional empirical data to understand the limits of the V_{OH} and V_{OL} values and modify the current measurements to respect these limits. Most of all, the inability to simulate the SRAM circuit for negative applied voltages is a limitation that could be investigated in further research.

While the model is empirical in nature, it was shown here to have successfully reproduced the general behavior of the MFSFET and associated digital circuits over a reasonable range of inputs. The only modifications that should be needed by a user at runtime are to establish external voltages and load resistances, which would vary depending on the desired circuit anyway, as well as the current capabilities of the transistors being used since this was shown to vary per device.

CHAPTER VII

CONCLUSION

The work presented in this thesis demonstrates the unique characteristics of the ferroelectric transistor and how its use in digital circuits has the potential for more interesting behavior that can be altered with the application of poling voltages to its gate terminal. Basic theoretical concepts of the inverter, SRAM, and DRAM circuits were presented, and the principles of the ferroelectric transistor were explained along with its differences from the traditional MOSFET. Experiments conducted demonstrate that circuits utilizing a ferroelectric transistor can behave similar to their non-ferroelectric equivalents, but that extending the range of applied voltages of such circuits creates the possibility for a different kind of circuit without changing the circuit layout or configuration. The hysteresis effect that the MFSFET possesses has been the subject of research into ferroelectric memories, and the results provided here show that it has the capability for providing the basis of a nonvolatile memory circuit. The MFSFET has a clear advantage over the MOSFET when considering that it can be used in a circuit and have its behavior changed by simply applying a particular poling voltage to its gate terminal; this creates the possibly for different circuits using the same hardware. Additional advantages include the potential for nonvolatile memory and reduced power

consumption when writing to an SRAM cell, as well as radiation hardness for space applications.

A programmatic model of the ferroelectric transistor and its use in the inverter and SRAM circuits was presented here. The model is based on empirical data and can quite accurately describe the behavior of the transistor alone. Further data collected allow the model to also provide a reasonable approximation of the behavior of the inverter and SRAM circuits. Due to presence of two ferroelectric transistors in the SRAM circuit, modeling becomes more difficult and should be further investigated to understand how each transistor affects the operation of its counterpart. The model is written in C++, and allows easy access since it can be compiled on a variety of operating system platforms, including Windows, OS X, and Linux. It also provides easy use due to the fact that it can be run from a command line environment and has preferences and input files that can be modified by any text editor without requiring the program to be recompiled for such changes.

Despite the age of ferroelectricity and its long-term usage in memory circuits, there is still much room for research into the operation of the MFSFET and its potential uses. Future research should include investigations into more digital circuits. Research should also be conducted on the topic of reconfigurable logic and what role the ferroelectric transistor could play in such circuits.

Further modeling efforts of the transistor and its associated circuits should strive to more accurately describe the behavior of the transistor and better understand its parameters. While the model presented here has shown success in effectively predicting the transistor's behavior, it was based on empirical data collected over a long period of

time. The ideal model should be analytical in nature, rather than empirical, and be applicable to the transistor regardless of the circuit to which it belongs. Alternatively, it could focus on recreating the measured gate polarization and applying this behavior to better calculate the threshold voltage and drain current. Also, this model focused on the current and voltage characteristics of the transistor with no regard to time; it would be ideal to be able to incorporate the timing aspects of the device into the model as well. Additionally, the consideration of active loads, such as traditional MOSFETs, can be investigated for inclusion into the model. Finally, an interactive model capable of accepting input and generating output in a graphical interface would be ideal, removing the need for text file input and allowing better processing of the data within the application itself rather than requiring the user to process the data separately once it has been collected.

APPENDIX

C++ CODE FOR MFSFET AND CIRCUIT MODEL

```
// *** simapp.cpp ***

// ****************************************************************
// MFSFET Device and Circuit Simulator
// Cody Mitchell, Master's Thesis Research
// Advisor: Dr. Fat Duen Ho
// Fall 2012
//
// Main program
// Purpose of this code: This is the main simulation driver for
//     the application. This code starts the program, processes
//     any command-line arguments, and creates the appropriate
//     objects for the MFSFET or its associated circuit. The
//     program then reads all voltages from the input file and
//     passes them to the appropriate functions of other classes
//     for further processing in simulating the desired circuit.
// ****************************************************************

#include <iostream>
#include <fstream>
#include <string>
#include <stdlib.h>
#include <cmath>
#include "Utilities.h"
#include "nMFSInverter.h"
#include "nMFS_SRAM_RLOAD.h"
#include "DevicePrefs.h"

// Define constants
#define MAX_LINE_SIZE 256
```

```
// Exit error codes
#define ERR_ARGCOUNT                    1
#define ERR_UNKNOWNMODE                 2
#define ERR_INPUTFILE_NOT_FOUND         3
#define ERR_OUTPUTFILE_NOT_FOUND        4
#define ERR_DEVICEPREFSFILE             5

bool ProcessArguments(int, char*[], std::string&, std::string&, std::string&);
void RunMFSInverter(std::ifstream&, std::ofstream&);
void RunSRAM1(std::ifstream&, std::ofstream&);
void RunMFSFET(std::ifstream&, std::ofstream&);

int main(int argc, char* argv[])
{
   // Declare necessary variables
   std::string runMode = "";
   std::string inFilename = "";
   std::string outFilename = "";
   std::ifstream iFile;
   std::ofstream oFile;

   // Process command line arguments
   if(ProcessArguments(argc, argv, runMode, inFilename, outFilename) != true)
      return ERR_ARGCOUNT; // Exit with error

   // Open input file and return an error if it could not be opened
   iFile.open(inFilename.c_str());
   if (!iFile)
      return ERR_INPUTFILE_NOT_FOUND;

   // Open output file and return an error if it could not be opened
   oFile.open(outFilename.c_str());
   if(!oFile)
   {
      // If the input file was opened, but the output file could not
      // be opened, close the input file cleanly before continuing
      iFile.close();
      return ERR_OUTPUTFILE_NOT_FOUND;
   }

   // Return error if the device preferences file could not be found or opened
   if(!(DevicePrefs::ReadDevicePrefs()))
      return ERR_DEVICEPREFSFILE;
```

```
   // Run appropriate simulation, depending on runMode value
   if (runMode == "mfsinverter")
      RunMFSInverter(iFile,oFile);
   else if (runMode == "sram1")
      RunSRAM1(iFile,oFile);
   else if (runMode == "mfsfet")
      RunMFSFET(iFile,oFile);
   else
      return ERR_UNKNOWNMODE;

   // Close input/output files
   iFile.close();
   oFile.close();

   // Exit successfully
   return 0;
}

// ****************************************************************
// ProcessArguments
//
// This function evaluates the provided command line arguments
//
// Input: # of arguments, argument values, circuit type,
//      input file reference, output file reference
// Return: Boolean
//      True - There were errors
//      False - There were no errors
// ****************************************************************
bool ProcessArguments(int progArgsCnt, char* progArgs[], std::string &mode,
std::string &inputfile, std::string &outputfile)
{
   // Check if we even have the correct number of arguments to the application
   if (progArgsCnt != 4)
   {
      // Print usage information
      std::cout << "\nUsage: simapp <mode> <inputfile> <outputfile>\n";
      std::cout << "Note: \"mode\" option is case sensitive\n";
      std::cout << "Valid options for \"mode\": mfsfet, mosinverter, mfsinverter,
         sram1\n\n";
      return false;
   }

   mode = progArgs[1];
   inputfile = progArgs[2];
   outputfile = progArgs[3];
```

```
   return true;
}

// ****************************************************************
// RunMFSInverter
//
// This function drives the simulation of the MFSFET-based inverter
//
// Input: input and output file references
// Return: none
// ****************************************************************
void RunMFSInverter(std::ifstream& fileIn, std::ofstream& fileOut)
{
   // Initialize variables to default values or values from
   // the deviceprefs.ini file
   double vin = 0.0;
   double ids = 0.0;
   double vout = 0.0;
   std::string voltageLine = "";
   double paramRload =
         DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_mfsinverter["rload"]);
   double paramVtop =
         DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_mfsinverter["vdd"]);
   double paramVs =
         DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_mfsinverter["vs"]);

   // Create the inverter object
   nMFSInverter myInverter;

   // Initialize the inverter and start applying node voltages
   myInverter.Init(paramRload);
   myInverter.Vtop(paramVtop);
   myInverter.VS(paramVs);
   myInverter.Init2();

   // Print header in output file
   fileOut << "vin,vout,ids\n";

   // Loop through all contents of input file
   while(!fileIn.eof())
   {
      // Read in the input/gate voltage
      fileIn >> voltageLine;
```

```
        // Ignore blank lines
        if (voltageLine != "")
        {
           // Apply a positive pulse
           if (voltageLine == "ppulse")
           {
              myInverter.PulsePositive();
              fileOut << "positivepulse,novout,nocurrent\n";
           }
           // Apply a negative pulse
           else if (voltageLine == "npulse")
           {
              myInverter.PulseNegative();
              fileOut << "negativepulse,novout,nocurrent\n";
           }
           // Apply an explicit input/gate voltage
           else
           {
              // Convert string to double (input voltage)
              vin = atof(voltageLine.c_str());

              // Apply input voltage to inverter
              myInverter.Vin(vin);

              // Udpate the inverter, recalculate the current
              myInverter.Update();

              // Retrieve the current and output voltage
              // of the inverter
              ids = myInverter.ID();
              vout = myInverter.Vout();

              // Print input voltage, output voltage, and drain
              // current to output file
              fileOut << vin << "," << vout << "," << ids << "\n";
           }
        }
     }
  }
```

```
// ****************************************************************
// RunSRAM1
//
// This function drives the simulation of the MFSFET-based SRAM
// with resistive loads.
//
// Input: input and output file references
// Return: none
// ****************************************************************
void RunSRAM1(std::ifstream& fileIn, std::ofstream& fileOut)
{
   // Initialize variables to default values or values from
   // the deviceprefs.ini file
   double vin = 0.0;
   double idsLeft = 0.0;
   double idsRight = 0.0;
   double vout = 0.0;
   std::string voltageLine = "";
   int maxTimePts = 25;
   double t, t1, t2;
   t2 = 0.1;
   double paramRload =
         DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_sram1["rload"]);
   double paramVtop =
         DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_sram1["vdd"]);
   double paramVs = DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_sram1["vs"]);

   // Create SRAM circuit object
   nMFS_SRAM_RLOAD mySRAM;

   // Initialize the SRAM and start applying node voltages
   mySRAM.Init(paramRload);
   mySRAM.Vtop(paramVtop);
   mySRAM.VS(paramVs);
   mySRAM.Init2();

   // Print header in output file
   fileOut << "vin,idsLeft,idsRight\n";

   // Loop through all contents of input file
   while(!fileIn.eof())
   {
      // Read in the next node 1 voltage
      fileIn >> voltageLine;

      // Ignore blank lines
```

```
        if (voltageLine != "")
        {
           // Apply a positive pulse
           if (voltageLine == "ppulse")
           {
              mySRAM.PulsePositive();
              fileOut << "positivepulse,nocurrent,nocurrent\n";
           }

           // Apply a negative pulse
           else if (voltageLine == "npulse")
           {
              mySRAM.PulseNegative();
              fileOut << "negativepulse,nocurrent,nocurrent\n";
           }
           // Apply an explicit input/gate voltage
           else
           {
              // Convert string to double (input voltage)
              vin = atof(voltageLine.c_str());

              // Apply node 1 voltage to SRAM
              mySRAM.V1(vin);

              // Recalculate the current through each of the
              // SRAM's MFSFETs and retrieve those values
              mySRAM.Update();
              idsLeft = mySRAM.IDLeft();
              idsRight = mySRAM.IDRight();

              // Print input voltage and both transistor drain
              // currents to output file
              fileOut << vin << "," << idsLeft << "," << idsRight << "\n";
           }
        }
     }
  }
```

```
// ****************************************************************
// RunMFSFET
//
// This function drives the simulation of the standalone MFSFET.
//
// Input: input and output file references
// Return: none
// ****************************************************************
void RunMFSFET(std::ifstream& fileIn, std::ofstream& fileOut)
{
   // Initialize variables to default values or values from
   // the deviceprefs.ini file
   double vin = 0.0;
   double ida = 0.0;
   std::string voltageLine = "";
   double paramVs = DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_mfsfet["vs"]);
   double paramVd = DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_mfsfet["vd"]);
   double paramVb = DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_mfsfet["vb"]);

   // Create the MFSFET object
   nMFSFET myFET;

   // Initialize the MFSFET and start applying node voltages
   myFET.Init("standalone");
   myFET.VD(paramVd);
   myFET.VS(paramVs);
   myFET.VB(paramVb);

   // Print header in output file
   fileOut << "vin,current\n";

   // Loop through all contents of input file
   while(!fileIn.eof())
   {
      // Read in the next gate voltage
      fileIn >> voltageLine;

      // Ignore blank lines
      if (voltageLine != "")
      {
         // Apply a positive pulse
         if (voltageLine == "ppulse")
         {
            myFET.PulsePositive();
            fileOut << "positivepulse,nocurrent\n";
         }
```

```
            // Apply a negative pulse
            else if (voltageLine == "npulse")
            {
               myFET.PulseNegative();
               fileOut << "negativepulse,nocurrent\n";
            }
            // Apply an explicit input/gate voltage
            else
            {
               // Convert string to double (input voltage)
               vin = atof(voltageLine.c_str());

               // Apply gate voltage to MFSFET
               myFET.VG(vin);

               // Recalculate drain current of MFSFET
               myFET.RecalcCurrent();

               // Retrieve drain current value
               ida = myFET.ID();

               // Print gate voltage and drain current
               // to output file
               fileOut << vin << "," << ida << "\n";
            }
         }
      }
   }
```

```
// *** nMFSFET.h ***

// ********************************************************************
// MFSFET Device and Circuit Simulator
// Cody Mitchell, Master's Thesis Research
// Advisor: Dr. Fat Duen Ho
// Fall 2012
//
// Class: nMFSFET
// Purpose of class: This class is used to simulate the MFSFET.
// ********************************************************************

#ifndef _NMFSFET_H_
#define _NMFSFET_H_

#include <vector>
#include "DevicePrefs.h"

class nMFSFET
{
public:

   // Enumerated values
   enum Polarization_t { ACTIVE, REMANENT };
   enum MfsfetType { ND1, ND7 };

   // Constructor and destructor
   nMFSFET();
   ~nMFSFET();

   // "Get" functions for MFSFET voltages
   double VDB();
   double VSB();
   double VGB();
   double VDS();
   double VGS();
   double VD();
   double VB();
   double VS();
   double VG();

   // "Get" function for MFSFET drain current
   double ID();

   // "Set" functions for MFSFET node voltages
   void VD(double);
```

```
    void VB(double);
    void VS(double);
    void VG(double);

    // Initialize the MFSFET
    void Init(std::string);

    // Set the maximum or minimum current of MFSFET
    // to a default or specified value
    void SetMaxCurrent();
    void SetMaxCurrent(double);
    void SetMinCurrent();
    void SetMinCurrent(double);

    // Get functions for min, max, and baseline current
    double GetBaselineCurrent();
    double GetMaxCurrent();
    double GetMinCurrent();

    // Recalculate the MFSFET drain current
    void RecalcCurrent();

    // Provide a positive or negative pulse on the MFSFET gate
    void PulsePositive();
    void PulseNegative();


private:

    // Curve fit method to determine drain current of MFSFET
    double CurrentCurve(bool, double, double, double, double);

    // Member variables
    double vb;          // Substrate voltage
    double w;           // Channel width
    double l;           // Channel length
    double vc;          // Coercive voltage
    double current;     // Drain current of transistor
    double vg;          // Gate/input voltage of transistor
    double vd;          // Drain voltage of transistor
    double vs;          // Source voltage of transistor
    double vt;          // Threshold voltage
    double basecurrent; // Baseline current used for scaling in
                        // VOH/VOL calculations
    double vsf;         // Scale-factor for determining curvature
                        // of polarization curve
```

```
    double vz;          // Voltage offset factor to shift current
                        // hysteresis loop to match empirical data
    double iMax;        // Maximum drain current
    double vgsLast;     // Previous gate-to-source voltage
    double iLast;       // Previous drain current
    int vDirection;     // Direction (increasing or decreasing)
                        // of gate-to-source voltage
    double iUpper;      // Upper bound of drain current, dynamically
                        // calculated during simulation
    double iLower;      // Lower bound of drain current, dynamically
                        // calculated during simulation
    double iMin;        // Minimum possible drain current
    double vInOffset;   // Offset voltage to shift drain current,
                        // dynamically calculated during simulation

    double vPulseVoltage;       // Magnitude of voltage used for
                                // poling pulses
    std::string circuitType;    // Type of circuit to which the
                                // MFSFET belongs
    MfsfetType mfsfetType;      // ND1 or ND7 transistor

};

#endif //_NMFSFET_H_
```

```
// *** nMFSFET.cpp ***

// ********************************************************************
// MFSFET Device and Circuit Simulator
// Cody Mitchell, Master's Thesis Research
// Advisor: Dr. Fat Duen Ho
// Fall 2012
//
// Class: nMFSFET
// Purpose of class: This class is used to simulate the MFSFET.
// ********************************************************************

#include "nMFSFET.h"
#include <cmath>
#include <vector>
#include "Utilities.h"

// Constructor and destructor of class
nMFSFET::nMFSFET() {}
nMFSFET::~nMFSFET() {}

// ********************************************************************
// nMFSFET::VDB()
//
// Input: none
// Return: drain-to-substrate voltage (calculated from nodes)
// ********************************************************************
double nMFSFET::VDB() { return vd - vb; }

// ********************************************************************
// nMFSFET::VSB()
//
// Input: none
// Return: source-to-substrate voltage (calculated from nodes)
// ********************************************************************
double nMFSFET::VSB() { return vs - vb; }

// ********************************************************************
// nMFSFET::VGB()
//
// Input: none
// Return: gate-to-substrate voltage (calculated from nodes)
// ********************************************************************
double nMFSFET::VGB() { return vg - vb; }
```

```
// ***************************************************************
// nMFSFET::VDS()
//
// Input: none
// Return: drain-to-source voltage (calculated from nodes)
// ***************************************************************
double nMFSFET::VDS() { return vd - vs; }

// ***************************************************************
// nMFSFET::VGS()
//
// Input: none
// Return: gate-to-source voltage (calculated from nodes)
// ***************************************************************
double nMFSFET::VGS() { return vg - vs; }

// ***************************************************************
// nMFSFET::ID()
//
// Input: none
// Return: drain current
// ***************************************************************
double nMFSFET::ID() { return current; }

// ***************************************************************
// nMFSFET::VD()
//
// Input: none
// Return: drain voltage
// ***************************************************************
double nMFSFET::VD() { return vd; }

// ***************************************************************
// nMFSFET::VB()
//
// Input: none
// Return: substrate voltage
// ***************************************************************
double nMFSFET::VB() { return vb; }
```

```
// ****************************************************************
// nMFSFET::VS()
//
// Input: none
// Return: source voltage
// ****************************************************************
double nMFSFET::VS() { return vs; }

// ****************************************************************
// nMFSFET::VG()
//
// Input: none
// Return: gate voltage
// ****************************************************************
double nMFSFET::VG() { return vg; }


// ****************************************************************
// nMFSFET::VD()
//
// Input: drain voltage of MFSFET
// Return: none
// ****************************************************************
void nMFSFET::VD(double x) { vd = x; }

// ****************************************************************
// nMFSFET::VB()
//
// Input: substrate voltage of MFSFET
// Return: none
// ****************************************************************
void nMFSFET::VB(double x) { vb = x; }

// ****************************************************************
// nMFSFET::VS()
//
// Input: source voltage of MFSFET
// Return: none
// ****************************************************************
void nMFSFET::VS(double x) { vs = x; }
```

```
// ************************************************************
// nMFSFET::VG()
//
// This function sets the gate voltage of the MFSFET, and creates
// a backup value of the voltage in order to determine directional
// trends.
//
// Input: gate voltage of MFSFET
// Return: none
// ************************************************************
void nMFSFET::VG(double x)
{
   // Backup the current VGS value
   vgsLast = VGS();

   // Set the gate voltage with the newly applied value
   vg = x;
}

// ************************************************************
// nMFSFET::Init()
//
// This function sets default values of MFSFET parameters, many
// based on the deviceprefs.ini file.
//
// Input: circuit type
// Return: none
// ************************************************************
void nMFSFET::Init(std::string circuit)
{
   // Initialize parameters of FET
   if (circuit == "")
      circuitType = "standalone";
   else
      circuitType = circuit;


   // Read device parameters from deviceprefs.ini
   w = DevicePrefs::GetDoubleValue(DevicePrefs::DPrefs_mfsfet["w"]);
   l = DevicePrefs::GetDoubleValue(DevicePrefs::DPrefs_mfsfet["l"]);
   vc = DevicePrefs::GetDoubleValue(DevicePrefs::DPrefs_mfsfet["vc"]);
   vPulseVoltage =
         DevicePrefs::GetDoubleValue(DevicePrefs::DPrefs_mfsfet["pulsevoltage"]);

   // Set type of transistor to ND1 or ND7
   std::string devtype = DevicePrefs::DPrefs_mfsfet["type"];
```

```
if (    ( devtype.compare("nd7") == 0 ) || ( devtype.compare("Nd7") == 0 ) ||
        ( devtype.compare("nD7") == 0 ) || ( devtype.compare("ND7") == 0 ) )
{
   mfsfetType = nMFSFET::ND7;
}
else
{
   mfsfetType = nMFSFET::ND1;
}

// Initialize current and node voltages
current = 0;
vb = 0;
vd = 0;
vg = 0;
vs = 0;
basecurrent = 4.6e-5;   // Used for VOL and VOH calculations

// Initialize scaling and shift factors, and directional
// tracking variables
vsf = 0.73;
vz = 1.6;
vgsLast = 0;
iLast = 0;
vDirection = 0;
vInOffset = 0;

// Alter a few parameters if this MFSFET is part of a larger circuit
// If circuit type is not recognized, assume a standalone MFSFET
if (circuitType == "mfsinverter")
{
   vc = vc + 0.5;
   vz = 2.0;
}
else if (circuitType == "sram1")
{
   vc = vc + 0.5;
   vz = 2.0;
}
else //if (circuitType == "standalone") - Default is "standalone" MFSFET
{
   vd = DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_mfsfet["vd"]);
   vb = DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_mfsfet["vb"]);
   vs = DevicePrefs::GetDoubleValue(DevicePrefs::CPrefs_mfsfet["vs"]);
   vc = vc + (vd-3)/10;
   vz = vz + (vd-3)/10;
```

```
    }

    // Call functions to set minimum and maximum current values
    // based on parameters provided so far
    SetMaxCurrent();
    SetMinCurrent();
}

// ****************************************************************
// nMFSFET::SetMaxCurrent()
//
// This function sets the maximum current of the MFSFET. This is
// based on the normcurrent value, which is maximum current of the
// MFSFET when VDS is equal to 3.
//
// Input: none
// Return: none
// ****************************************************************
void nMFSFET::SetMaxCurrent()
{
    iMax = DevicePrefs::GetDoubleValue(DevicePrefs::DPrefs_mfsfet["normcurrent"]);
    if (circuitType == "standalone")
        iMax = iMax * vd/3;
    iUpper = iMax;
}

// ****************************************************************
// nMFSFET::SetMaxCurrent()
//
// This function sets the maximum current of the MFSFET to a
// specified value.
//
// Input: max current
// Return: none
// ****************************************************************
void nMFSFET::SetMaxCurrent(double x)
{
    iMax = x;
    iUpper = x;
}
```

```
// ****************************************************************
// nMFSFET::SetMinCurrent()
//
// This function sets the minimum current of the MFSFET.
//
// Input: none
// Return: none
// ****************************************************************
void nMFSFET::SetMinCurrent()
{
   iMin = 0;
   iLower = iMin;
}

// ****************************************************************
// nMFSFET::SetMinCurrent()
//
// This function sets the minimum current of the MFSFET to a
// specified value.
//
// Input: minimum current
// Return: none
// ****************************************************************
void nMFSFET::SetMinCurrent(double x)
{
   iMin = x;
   iLower = x;
}

// ****************************************************************
// nMFSFET::GetMaxCurrent()
//
// This function returns the maximum current of the MFSFET to
// the caller.
//
// Input: none
// Return: maximum current
// ****************************************************************
double nMFSFET::GetMaxCurrent()
{
   return iMax;
}

// ****************************************************************
// nMFSFET::GetMinCurrent()
//
```

```
// This function returns the minimum current of the MFSFET to
// the caller.
//
// Input: none
// Return: minimum current
// ********************************************************************
double nMFSFET::GetMinCurrent()
{
   return iMin;
}

// ********************************************************************
// nMFSFET::GetBaselineCurrent()
//
// This function returns the baseline current of the MFSFET that
// was used to establish the behavior of the current when the MFSFET
// is used in an inverter or SRAM cell.
//
// Input: none
// Return: baseline current
// ********************************************************************
double nMFSFET::GetBaselineCurrent()
{
   return basecurrent;
}

// ********************************************************************
// nMFSFET::RecalcCurrent()
//
// This function determines the appropriate steps to recalculate
// the current through the MFSFET. Any modifications to the offset
// voltage are done here. The CurrentCurve() function is called to
// determine the appropriate shape of the current hysteresis curve.
//
// Input: none
// Return: none
// ********************************************************************
void nMFSFET::RecalcCurrent()
{
   // Reset the previous current to the present one before
   // recalculating this value
   iLast = current;

   // Calculate offset voltage
   // The offsets are based on empirical data collected from
   // testing and observing the behavior of an MFSFET's drain current.
```

```
if ( VGS() == vgsLast )
{
   vInOffset = vInOffset;
}
else if ( VGS() > vgsLast )
{
   if (vDirection > 0)
   {
      vInOffset = vInOffset;
   }
   else
   {
      if (( VGS() <= (vc - vz) ) && ( VGS() > (-vc - vz) ))
      {
         vInOffset = -2 * vc + (abs(VGS()) - abs(vc - vz));
      }
      else if ( VGS() <= ( -vc - vz ) )
      {
         vInOffset = 0;
      }
      else
      {
         vInOffset = -2 * vc;
      }
   }
}
else // VGS() < vgsLast
{
   if (vDirection > 0)
   {
      if ( VGS() > (vc - vz) )
      {
         vInOffset = 0;
      }
      else
      {
         vInOffset = vInOffset;
      }
   }
   else if (vDirection < 0)
   {
      vInOffset = vInOffset;
   }
   else
   {
```

```
        if (( VGS() <= (vc - vz) ) && ( VGS() > (-vc - vz) ))
        {
           vInOffset = -2 * vc + (abs(VGS()) - abs(vc - vz));
        }
        else if ( VGS() <= ( -vc - vz ) )
        {
           vInOffset = 0;
        }
        else
        {
           vInOffset = -2 * vc;
        }
     }
  }

  // Recalculate the current with the CurrentCurve() function
  // Determine our bounds for the CurrentCurve() function
  // according to the direction of the gate voltage change
  if (VGS() > vgsLast)
  {
     if (vDirection != 0)
     {
        current = CurrentCurve(true,iLower,iMax,VGS(),vInOffset);
     }
     else
     {
        iLower = iLast;
        current = CurrentCurve(true,iLower,iMax,VGS(),vInOffset);
     }
     vDirection = 1;
  }
  else if (VGS() < vgsLast)
  {
     vInOffset = 0;

     if (vDirection > 0)
     {
        iUpper = (iLast - ((iMin)/(exp((vgsLast-vc-vz)/(vsf))+1)))/(1 - (1/(exp((vgsLast-
                 vc-vz)/(vsf))+1)));
        current = CurrentCurve(false,iMin,iUpper,VGS(),vInOffset);
     }
     else
     {
        current = CurrentCurve(false,iMin,iUpper,VGS(),vInOffset);
     }
     vDirection = -1;
```

```
    }
    else
    {
        // If the gate voltage has not changed,
        // the current should not change
        current = iLast;
    }

}

// ****************************************************************
// nMFSFET::CurrentCurve()
//
// This function determines the appropriate steps to recalculate
// the current through the MFSFET. Any modifications to the offset
// voltage are done here. The CurrentCurve() function is called to
// determine the appropriate shape of the current hysteresis curve.
//
// Input: direction of gate voltage, min/max current bounds,
//      input/gate voltage, offset voltage
// Return: none
// ****************************************************************
double nMFSFET::CurrentCurve(bool increasing, double iL, double iU, double vIn,
double vOffset)
{
    // Variable "increasing" is true when the input voltage is higher than the previous
    // input voltage, meaning that the input is on an increasing curve
    double retVal;
    double deltaVal;

    // If testing the MFSFET alone and the gate voltage is
    // sufficiently high, the slope of the current curve
    // should be altered by setting deltaVal
    if ((vIn >= 2*vc) && (circuitType == "standalone") && (mfsfetType ==
            nMFSFET::ND1))
        deltaVal = (vd-3)*abs(2*vc-vIn)*iMax*0.01;

    else
        deltaVal = 0;

    // Determine current value based on direction of gate voltage
    if (increasing)
    {
        // If vin is increasing
        retVal = iMax * (1-(1-iL/iMax) / (exp((vIn+vc-vz+vOffset)/(vsf))+1)) + deltaVal;
    }
```

```
    else
    {
        // If vin is decreasing
        retVal = iU * (1-(1-iMin/iU)/(exp((vIn-vc-vz+vOffset)/(vsf))+1)) + deltaVal;
    }

    return retVal;

}

// ****************************************************************
// nMFSFET::PulsePositive()
//
// This function uses the pulse voltage in the deviceprefs.ini file
// to apply a positive pulse to the gate of the MFSFET. Additional
// steps are taken to recalculate the current, apply a slightly
// smaller voltage, and recalculate the current again. This allows
// the direction of the gate voltage to be updated correctly since
// this is likely the highest voltage value that will be applied
// to the gate.
//
// Input: none
// Return: none
// ****************************************************************
void nMFSFET::PulsePositive()
{
    double vb1, vd1, vs1;
    VG(vPulseVoltage);
    RecalcCurrent();
    VG(vPulseVoltage - 0.001);
    RecalcCurrent();
}

// ****************************************************************
// nMFSFET::PulseNegative()
//
// This function uses the pulse voltage in the deviceprefs.ini file
// to apply a negative pulse to the gate of the MFSFET. Additional
// steps are taken to recalculate the current, apply a slightly
// larger voltage, and recalculate the current again. This allows
// the direction of the gate voltage to be updated correctly since
// this is likely the lowest voltage value that will be applied
// to the gate.
//
// Input: none
// Return: none
```

```
// ************************************************************
void nMFSFET::PulseNegative()
{
   double vb1, vd1, vs1;
   VG(-vPulseVoltage);
   RecalcCurrent();
   VG(-vPulseVoltage + 0.001);
   RecalcCurrent();
}
```

```
// *** nMFSInverter.h ***

// ********************************************************************
// MFSFET Device and Circuit Simulator
// Cody Mitchell, Master's Thesis Research
// Advisor: Dr. Fat Duen Ho
// Fall 2012
//
// Class: nMFSInverter
// Purpose of class: This class is used to simulate the resistive-
//      load inverter based on the MFSFET.
// ********************************************************************

#ifndef _NMFSINVERTER_H
#define _NMFSINVERTER_H

#include "nMFSFET.h"

class nMFSInverter
{
public:

   // Constructor and destructor
   nMFSInverter();
   ~nMFSInverter();

   // Set/get functions for Vout (drain)
   double Vout();
   void Vout(double);

   // Set/get functions for Vin (gate)
   double Vin();
   void Vin(double);

   // Set/get functions for VS (source)
   double VS();
   void VS(double);

   // Set/get functions for Vtop (VDD)
   double Vtop();
   void Vtop(double);

   // Get the drain current
   double ID();
```

```
    // Update the parameters of the circuit to calculate
    // the current and output voltage
    void Update();

    // Initialization functions for the inverter to set parameters
    void Init(double);
    void Init2();

    // Provide a positive or negative pulse on the MFSFET gate
    void PulsePositive();
    void PulseNegative();

private:
    double vtop;                // Voltage applied to top of resistor
    double vout;                // Output voltage of inverter
    double vin;                 // Input voltage of inverter (gate of transistor)
    double vs;                  // Source voltage of transistor
    nMFSFET theTransistor;      // Instance of the nMFSFET being used
    double resistance;          // Value of the resistor in the inverter
    double current;             // Current through the inverter

    // Find expected VOH or VOL based on resistance and VDD
    double GetVOH(double,double);
    double GetVOL(double,double);
};

#endif //_NMFSINVERTER_H
```

```
// *** nMFSInverter.cpp ***

// ********************************************************************
// MFSFET Device and Circuit Simulator
// Cody Mitchell, Master's Thesis Research
// Advisor: Dr. Fat Duen Ho
// Fall 2012
//
// Class: nMFSInverter
// Purpose of class: This class is used to simulate the resistive-
//      load inverter based on the MFSFET.
// ********************************************************************

#include "nMFSInverter.h"
#include "Utilities.h"
#include <iostream>

// Default constructor and destructor
nMFSInverter::nMFSInverter() {}
nMFSInverter::~nMFSInverter() {}

// ********************************************************************
// nMFSInverter::Init
//
// This function initializes the inverter circuit by setting the
// resistance of the load and initializing default values of the
// MFSFET by calling its own Init() function.
//
// Input: value of load resistance
// Return: none
// ********************************************************************
void nMFSInverter::Init(double r)
{
   resistance = r;
   theTransistor.Init("mfsinverter");
}
```

```
// ************************************************************
// nMFSInverter::Init2
//
// This function initializes the inverter circuit by determining
// the minimum and maximum expected current based on the supply
// voltage (VDD) and the load resistance.
//
// Input: none
// Return: none
// ************************************************************
void nMFSInverter::Init2()
{
   // Calculate minimum and maximum expected current values
   double iLow = (vtop-GetVOH(vtop,resistance))/resistance;
   double iHigh = (vtop-GetVOL(vtop,resistance))/resistance;

   // Set minimum and maximum current values of MFSFET
   theTransistor.SetMaxCurrent(iHigh);
   theTransistor.SetMinCurrent(iLow);
}

// ************************************************************
// nMFSInverter::Vout
//
// Input: none
// Return: Vout (drain voltage of transistor)
// ************************************************************
double nMFSInverter::Vout()
{
   return vout;
}

// ************************************************************
// nMFSInverter::Vin
//
// Input: none
// Return: Vin (gate voltage of transistor)
// ************************************************************
double nMFSInverter::Vin()
{
   return vin;
}
```

```
// ***************************************************************
// nMFSInverter::VS
//
// Input: none
// Return: VS (source voltage of transistor)
// ***************************************************************
double nMFSInverter::VS()
{
   return vs;
}

// ***************************************************************
// nMFSInverter::Vtop
//
// Input: none
// Return: Vtop (VDD, supply voltage of inverter)
// ***************************************************************
double nMFSInverter::Vtop()
{
   return vtop;
}

// ***************************************************************
// nMFSInverter::Vout
//
// Input: double value to become new value of Vout
//      (drain voltage of transistor)
// Return: none
// ***************************************************************
void nMFSInverter::Vout(double x)
{
   vout = x;
}

// ***************************************************************
// nMFSInverter::Vin
//
// Input: double value to become new value of Vin
//      (gate voltage of transistor)
// Return: none
// ***************************************************************
void nMFSInverter::Vin(double x)
{
   vin = x;
}
```

```
// ***************************************************************
// nMFSInverter::VS
//
// Input: double value to become new value of VS
//      (source voltage of transistor)
// Return: none
// ***************************************************************
void nMFSInverter::VS(double x)
{
   vs = x;
}

// ***************************************************************
// nMFSInverter::Vtop
//
// Input: double value to become new value of Vtop
//      (VDD, supply voltage of inverter circuit)
// Return: none
// ***************************************************************
void nMFSInverter::Vtop(double x)
{
   vtop = x;
}

// ***************************************************************
// nMFSInverter::ID
//
// Input: none
// Return: ID (drain current of transistor, or total current
//      through circuit)
// ***************************************************************
double nMFSInverter::ID()
{
   return current;
}

// ***************************************************************
// nMFSInverter::Update
//
// This function uses the node voltages applied to the inverter
// circuit and applies them to the MFSFET as needed. The current is
// then calculated through the transistor.
//
// Input: none
// Return: none
// ***************************************************************
```

```
void nMFSInverter::Update()
{
   double ir = 0;         // Current through resistor
   double it = 0;         // Current through transistor
   double vtmp = 0;    // Temporary voltage
   double itmp = 0;     // Temporary current calculation

   // Update node voltages of transistor
   theTransistor.VB(vs);
   theTransistor.VS(vs);
   theTransistor.VG(vin);
   vtmp = vtop/2;
   theTransistor.VD(vtmp);

   // Recalculate the current of the transistor
   theTransistor.RecalcCurrent();
   it = theTransistor.ID();
   ir = it;

   // Recalculate output voltage of inverter
   vtmp = vtop - resistance * it;
   current = it;
   vout = vtmp;
   theTransistor.VD(vtmp);
}

// ****************************************************************
// nMFSInverter::PulsePositive
//
// This function calls the PulsePositive() function of the MFSFET
// in order to apply a specific pulse/poling voltage to the
// transistor and take any needed steps to determine the increasing
// or decreasing direction of the gate voltage without the user
// needing to explicitly input multiple values to achieve the same
// result.
//
// Input: none
// Return: none
// ****************************************************************
void nMFSInverter::PulsePositive()
{
   theTransistor.PulsePositive();
}
```

```
// ****************************************************************
// nMFSInverter::PulseNegative
//
// This function calls the PulseNegative() function of the MFSFET
// in order to apply a specific pulse/poling voltage to the
// transistor and take any needed steps to determine the increasing
// or decreasing direction of the gate voltage without the user
// needing to explicitly input multiple values to achieve the same
// result.
//
// Input: none
// Return: none
// ****************************************************************
void nMFSInverter::PulseNegative()
{
   theTransistor.PulseNegative();
}

// ****************************************************************
// nMFSInverter::GetVOH
//
// This function uses a curve-fitting cubic equation to determine
// the expected VOH (high output voltage) of the circuit based on
// the supply voltage and the load resistance.
//
// Input: vdd (supply voltage VDD), rload (value of load resistance)
// Return: VOH (highest expected output voltage of inverter)
// ****************************************************************
double nMFSInverter::GetVOH(double vdd, double rload)
{
   // Use a cubic curve-fitting equation to find the expected value of VOH
   // based on a given VDD and load resistance
   double a, b, c, d, f, g, h, i, j, k, voh;
   a=9.50E-02;
   b=-3.81E-07;
   c=9.51E-01;
   d=5.01E-12;
   f=1.44E-02;
   g=-1.71E-17;
   h=-1.35E-03;
   i=-9.48E-07;
   j=8.31E-13;
   k=-1.30E-08;
```

```
voh=a+(b*rload)+(c*vdd)+(d*rload*rload)+(f*vdd*vdd)+(g*rload*rload*rload)+(h*vdd
*vdd*vdd)+(i*rload*vdd)+(j*rload*rload*vdd)+(k*rload*vdd*vdd);

   return voh;
}

// ***************************************************************
// nMFSInverter::GetVOL
//
// This function uses a curve-fitting cubic equation to determine
// the expected VOL (low output voltage) of the circuit based on
// the supply voltage and the load resistance.
//
// Input: vdd (supply voltage VDD), rload (value of load resistance)
// Return: VOL (lowest expected output voltage of inverter)
// ***************************************************************
double nMFSInverter::GetVOL(double vdd, double rload)
{
   // Use a cubic curve-fitting equation to find the expected value of VOL
   // based on a given VDD and load resistance
   double a, b, c, d, f, g, h, i, j, k, vol;
   a=1.29E-01;
   b=-1.00E-05;
   c=9.44E-01;
   d=1.49E-10;
   f=1.32E-02;
   g=-4.86E-16;
   h=-3.10E-03;
   i=-7.49E-06;
   j=1.83E-11;
   k=9.91E-08;


vol=a+(b*rload)+(c*vdd)+(d*rload*rload)+(f*vdd*vdd)+(g*rload*rload*rload)+(h*vdd*
vdd*vdd)+(i*rload*vdd)+(j*rload*rload*vdd)+(k*rload*vdd*vdd);

   return vol;
}
```

```
// *** nMFS_SRAM_RLOAD.h ***

// ********************************************************************
// MFSFET Device and Circuit Simulator
// Cody Mitchell, Master's Thesis Research
// Advisor: Dr. Fat Duen Ho
// Fall 2012
//
// Class: nMFS_SRAM_RLOAD
// Purpose of class: This class is used to simulate the resistive-
//      load SRAM cell based on the MFSFET.
// ********************************************************************

#ifndef _NMFS_SRAM_RLOAD_H_
#define _NMFS_SRAM_RLOAD_H_

#include "nMFSFET.h"

class nMFS_SRAM_RLOAD
{
public:

   // Constructor and destructor
   nMFS_SRAM_RLOAD();
   ~nMFS_SRAM_RLOAD();

   // Initialization functions for SRAM cell
   void Init(double);
   void Init2();

   // "Get" functions for node voltages
   double V2();
   double V1();
   double VS();
   double Vtop();

   // "Set" functions for node voltages
   void V2(double);
   void V1(double);
   void VS(double);
   void Vtop(double);

   // "Get" functions for current in each side of circuit
   double IDLeft();
   double IDRight();
```

```
    // Function to update state of circuit
    void Update();

    // Provide a positive or negative pulse on the MFSFET gate
    void PulsePositive();
    void PulseNegative();

private:
    double vtop;        // Voltage applied to top of resistors
    double v2;          // Voltage at node 2 (drain of right MFSFET)
    double v1;          // Voltage at node 1 (drain of left MFSFET)
    double vs;          // Source voltage of transistors
    nMFSFET m1;         // The left side MFSFET transistor object
    nMFSFET m2;         // The right side MFSFET transistor object
    double res;         // Value of the load resistors in the SRAM cell
    double idLeft;      // Current through the left side of the SRAM cell
    double idRight;     // Current through the right side of the SRAM cell

    // Find expected VOH or VOL based on resistance and VDD
    double GetVOH(double,double);
    double GetVOL(double,double);
};

#endif //_NMFS_SRAM_RLOAD_H_
```

```
// *** nMFS_SRAM_RLOAD.cpp ***

// ********************************************************************
// MFSFET Device and Circuit Simulator
// Cody Mitchell, Master's Thesis Research
// Advisor: Dr. Fat Duen Ho
// Fall 2012
//
// Class: nMFS_SRAM_RLOAD
// Purpose of class: This class is used to simulate the resistive-
//      load SRAM cell based on the MFSFET.
// ********************************************************************

#include "nMFS_SRAM_RLOAD.h"
#include <cmath>

// Default constructor and destructor
nMFS_SRAM_RLOAD::nMFS_SRAM_RLOAD() {}
nMFS_SRAM_RLOAD::~nMFS_SRAM_RLOAD() {}


// ********************************************************************
// nMFS_SRAM_RLOAD::Init
//
// This function initializes the SRAM circuit by setting the
// resistance of the loads and initializing default values of the
// MFSFET by calling its own Init() function.
//
// Input: value of load resistance
// Return: none
// ********************************************************************
void nMFS_SRAM_RLOAD::Init(double resLoad)
{
   // Set load resistance value (same for both loads)
   res = resLoad;

   // Initialize each of the MFSFETs
   m1.Init("sram1");
   m2.Init("sram1");

   // Set the initial node voltages for each transistor
   m1.VB(0.0);
   m1.VS(0.0);
   m1.VD(0.0);
   m1.VG(0.0);
   m2.VB(0.0);
```

```
   m2.VS(0.0);
   m2.VD(0.0);
   m2.VG(0.0);
}

// ****************************************************************
// nMFS_SRAM_RLOAD::Init2
//
// This function initializes the SRAM circuit by determining
// the minimum and maximum expected current based on the supply
// voltage (VDD) and the load resistances.
//
// Input: none
// Return: none
// ****************************************************************
void nMFS_SRAM_RLOAD::Init2()
{
   // Determine how to scale the max current of each MFSFET
   // based on the baseline current that was initially used
   // to characterize the MFSFET
   double scale1 = m1.GetMaxCurrent() / m1.GetBaselineCurrent();
   double scale2 = m2.GetMaxCurrent() / m2.GetBaselineCurrent();

   // Determine minimum and maximum current values for MFSFETs
   // based on VOH and VOL expected values
   double iLow = (vtop-GetVOH(vtop,res))/res;
   double iHigh = (vtop-GetVOL(vtop,res))/res;

   // Set minimum and maximum current values for MFSFETs
   m1.SetMaxCurrent(iHigh * scale1);
   m2.SetMaxCurrent(iHigh * scale2);
   m1.SetMinCurrent(iLow);
   m2.SetMinCurrent(iLow);
}

// ****************************************************************
// nMFS_SRAM_RLOAD::V2
//
// Input: none
// Return: V2 (drain voltage of right-side transistor)
// ****************************************************************
double nMFS_SRAM_RLOAD::V2()          { return v2; }
```

```
// ************************************************************
// nMFS_SRAM_RLOAD::V1
//
// Input: none
// Return: V1 (drain voltage of left-side transistor)
// ************************************************************
double nMFS_SRAM_RLOAD::V1()          { return v1; }

// ************************************************************
// nMFS_SRAM_RLOAD::VS
//
// Input: none
// Return: VS (source voltage common to both transistors)
// ************************************************************
double nMFS_SRAM_RLOAD::VS()          { return vs; }

// ************************************************************
// nMFS_SRAM_RLOAD::Vtop
//
// Input: none
// Return: Vtop (VDD, top-most voltage, power supply to resistors)
// ************************************************************
double nMFS_SRAM_RLOAD::Vtop()        { return vtop; }

// ************************************************************
// nMFS_SRAM_RLOAD::V2
//
// Input: V2 (drain of right-side transistor, gate of left)
// Return: none
// ************************************************************
void nMFS_SRAM_RLOAD::V2(double x)    { v2 = x; }

// ************************************************************
// nMFS_SRAM_RLOAD::V1
//
// Input: V1 (drain of left-side transistor, gate of right)
// Return: none
// ************************************************************
void nMFS_SRAM_RLOAD::V1(double x)    { v1 = x; }

// ************************************************************
// nMFS_SRAM_RLOAD::VS
//
// Input: VS (source voltage common to both transistors)
// Return: none
// ************************************************************
```

```
void nMFS_SRAM_RLOAD::VS(double x)      { vs = x; }

// ********************************************************************
// nMFS_SRAM_RLOAD::Vtop
//
// Input: Vtop (VDD, top-most voltage, power supply to resistors)
// Return: none
// ********************************************************************
void nMFS_SRAM_RLOAD::Vtop(double x)    { vtop = x; }

// ********************************************************************
// nMFS_SRAM_RLOAD::IDLeft
//
// Input: none
// Return: drain current of left-side transistor
// ********************************************************************
double nMFS_SRAM_RLOAD::IDLeft()        { return idLeft; }

// ********************************************************************
// nMFS_SRAM_RLOAD::IDRight
//
// Input: none
// Return: drain current of right-side transistor
// ********************************************************************
double nMFS_SRAM_RLOAD::IDRight()       { return idRight; }


// ********************************************************************
// nMFS_SRAM_RLOAD::Update
//
// This function updates the node voltages of the SRAM circuit and
// triggers the recalculation of current through each MFSFET.
//
// Input: none
// Return: none
// ********************************************************************
void nMFS_SRAM_RLOAD::Update()
{
   double IdL, IdR, VDR;
   double itmp = 0;
   double vtmp = 0;

   // Update node voltages of transistors
   m1.VB(vs);
   m1.VS(vs);
   m2.VB(vs);
```

```
    m2.VS(vs);
    m2.VG(v1);
    m1.VD(v1);
    vtmp = vtop/2;
    m2.VD(vtmp);

    // Calculate current in right-side transistor
    m2.RecalcCurrent();
    IdR = m2.ID();

    // Recalculate drain voltage on right-side transistor
    vtmp = vtop - res * IdR;
    VDR = vtmp;
    m1.VG(VDR);

    // Calculate current in left-side transistor
    m1.RecalcCurrent();
    IdL = m1.ID();

    // Set current values
    idLeft = IdL;
    idRight = IdR;
}


// ****************************************************************
// nMFS_SRAM_RLOAD:PulsePositive
//
// This function calls the PulsePositive() function of each MFSFET
// in order to apply a specific pulse/poling voltage to the
// transistor and take any needed steps to determine the increasing
// or decreasing direction of the gate voltage without the user
// needing to explicitly input multiple values to achieve the same
// result.
//
// Input: none
// Return: none
// ****************************************************************
void nMFS_SRAM_RLOAD::PulsePositive()
{
    m1.PulsePositive();
    m2.PulsePositive();
}
```

```
// ****************************************************************
// nMFS_SRAM_RLOAD:PulseNegative
//
// This function calls the PulseNegative() function of each MFSFET
// in order to apply a specific pulse/poling voltage to the
// transistor and take any needed steps to determine the increasing
// or decreasing direction of the gate voltage without the user
// needing to explicitly input multiple values to achieve the same
// result.
//
// Input: none
// Return: none
// ****************************************************************
void nMFS_SRAM_RLOAD::PulseNegative()
{
   m1.PulseNegative();
   m2.PulseNegative();
}

// ****************************************************************
// nMFS_SRAM_RLOAD::GetVOH
//
// This function uses a curve-fitting cubic equation to determine
// the expected VOH (high output voltage) of the circuit based on
// the supply voltage and the load resistance.
//
// Input: vdd (supply voltage VDD), rload (value of load resistance)
// Return: VOH (highest expected drain voltage of right-side MFSFET)
// ****************************************************************
double nMFS_SRAM_RLOAD::GetVOH(double vdd, double rload)
{
   // Use a cubic curve-fitting equation to find the expected value of VOH
   // based on a given VDD and load resistance
   double a, b, c, d, f, g, h, i, j, k, voh;
   a=9.50E-02;
   b=-3.81E-07;
   c=9.51E-01;
   d=5.01E-12;
   f=1.44E-02;
   g=-1.71E-17;
   h=-1.35E-03;
   i=-9.48E-07;
   j=8.31E-13;
   k=-1.30E-08;
```

```
voh=a+(b*rload)+(c*vdd)+(d*rload*rload)+(f*vdd*vdd)+(g*rload*rload*rload)+(h*vdd
*vdd*vdd)+(i*rload*vdd)+(j*rload*rload*vdd)+(k*rload*vdd*vdd);

   return voh;
}

// ****************************************************************
// nMFS_SRAM_RLOAD::GetVOL
//
// This function uses a curve-fitting cubic equation to determine
// the expected VOL (low output voltage) of the circuit based on
// the supply voltage and the load resistance.
//
// Input: vdd (supply voltage VDD), rload (value of load resistance)
// Return: VOL (lowest expected drain voltage of left-side MFSFET)
// ****************************************************************
double nMFS_SRAM_RLOAD::GetVOL(double vdd, double rload)
{
   // Use a cubic curve-fitting equation to find the expected value of VOL
   // based on a given VDD and load resistance
   double a, b, c, d, f, g, h, i, j, k, vol;
   a=1.29E-01;
   b=-1.00E-05;
   c=9.44E-01;
   d=1.49E-10;
   f=1.32E-02;
   g=-4.86E-16;
   h=-3.10E-03;
   i=-7.49E-06;
   j=1.83E-11;
   k=9.91E-08;


vol=a+(b*rload)+(c*vdd)+(d*rload*rload)+(f*vdd*vdd)+(g*rload*rload*rload)+(h*vdd*
vdd*vdd)+(i*rload*vdd)+(j*rload*rload*vdd)+(k*rload*vdd*vdd);

   return vol;
}
```

```
// *** DevicePrefs.h ***

// ********************************************************************
// MFSFET Device and Circuit Simulator
// Cody Mitchell, Master's Thesis Research
// Advisor: Dr. Fat Duen Ho
// Fall 2012
//
// Class: DevicePrefs, header file
// Purpose of class: This class is used to read and interpret a
//      device preferences file (deviceprefs.ini).
// ********************************************************************

#ifndef _DEVICEPREFS_H_
#define _DEVICEPREFS_H_

#include <map>
#include <vector>
#include <string>

class DevicePrefs
{
public:
   // Core function of class, read in the device prefs file
   static bool ReadDevicePrefs();

   // Functions to get numerical equivalents of string-
   // formatted values
   static double GetDoubleValue(std::string);
   static int GetIntValue(std::string);

   // Maps that contain property-value pairs for circuits
   // and devices
   static std::map<std::string,std::string> CPrefs_mfsinverter;
   static std::map<std::string,std::string> CPrefs_sram1;
   static std::map<std::string,std::string> CPrefs_mfsfet;
   static std::map<std::string,std::string> DPrefs_mfsfet;

private:

   // Function used to split property=value string into separate elements
   static std::vector<std::string> SplitString(std::string);
};


#endif //DEVICEPREFS_H_
```

```
// *** DevicePrefs.cpp ***

// ********************************************************************
// MFSFET Device and Circuit Simulator
// Cody Mitchell, Master's Thesis Research
// Advisor: Dr. Fat Duen Ho
// Fall 2012
//
// Class: DevicePrefs
// Purpose of class: This class is used to read and interpret a
//      device preferences file (deviceprefs.ini).
// ********************************************************************

#include "DevicePrefs.h"
#include <cmath>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include <map>

// Create maps to contain properties for each circuit and device type
std::map<std::string,std::string> DevicePrefs::CPrefs_mfsinverter;
std::map<std::string,std::string> DevicePrefs::CPrefs_sram1;
std::map<std::string,std::string> DevicePrefs::CPrefs_mfsfet;
std::map<std::string,std::string> DevicePrefs::DPrefs_mfsfet;

// ********************************************************************
// DevicePrefs::ReadDevicePrefs
//
// This function reads in deviceprefs.ini to determine the device
// and circuit settings to be simulated.
//
// Input: (none)
// Return: Boolean
//      True - no problems encountered while reading the prefs file
//      False - problems encountered while reading the prefs file
// ********************************************************************
bool DevicePrefs::ReadDevicePrefs()
{
   // Set up initial variables
   std::string inFilename = "deviceprefs.ini";
   std::ifstream iFile;
   std::string thisTextLine = "";
   std::string category = "";
```

```
bool flagReadOk = true;
size_t searchBeginBracket, searchEndBracket;
std::vector<std::string> fileFields;
std::map<std::string,std::string> * mapPointer;

// Try to open the device prefs file. Return "false" to indicate
// an error if the file doesn't open
iFile.open(inFilename.c_str());
if (!iFile)
   return false;

// Loop through the device prefs file while it's open and no
// errors have been found
while(!iFile.eof() && flagReadOk)
{
   // Read the next line of text
   iFile >> thisTextLine;
   if ((thisTextLine != "") && !iFile.eof())
   {
      // Determine if line contains a category name for a circuit or device
      // Check to see if line contains a beginning and ending bracket
      searchBeginBracket = thisTextLine.find("[");
      searchEndBracket = thisTextLine.find("]");


      if ((searchBeginBracket != std::string::npos) && (searchEndBracket !=
      std::string::npos))
      {
         // Line contains a beginning and ending bracket

         // Check to see actual positions of beginning and ending brackets in line
         if ((int(searchBeginBracket) != 0) || (int(searchEndBracket) !=
               (thisTextLine.length()-1)))
         {
            // This line does not contain a valid category name
            flagReadOk = false;
         }
         else
         {
            // This line contains a valid category name
            // Determine while circuit/device this is
            category = thisTextLine;
            if (category == "[circuit_mfsinverter]")
               mapPointer = &DevicePrefs::CPrefs_mfsinverter;
            else if (category == "[circuit_sram1]")
               mapPointer = &DevicePrefs::CPrefs_sram1;
```

```
                else if (category == "[circuit_mfsfet]")
                    mapPointer = &DevicePrefs::CPrefs_mfsfet;
                else if (category == "[device_mfsfet]")
                    mapPointer = &DevicePrefs::DPrefs_mfsfet;
                else
                    flagReadOk = false; // We've encountered an invalid header/category
            }
        }
        else
        {
            // Line does not contain a beginning and ending bracket
            // Line must be a property assignment, so read the
            // property and its value (ex: property=value)
            fileFields = DevicePrefs::SplitString(thisTextLine);
            if (fileFields.size() != 2)
            {
                // If there are not 2 fields, the line is bad
                flagReadOk = false;
            }
            else
            {
                // Create a property in the map for this circuit/device
                // Assign the corresponding value to the property
                mapPointer->insert( make_pair(fileFields[0],fileFields[1]) );
            }
        }
    }
}

// Close the device prefs file
iFile.close();

if (!flagReadOk)
{
    // Let's let the user know if there was an error when reading
    // the device prefs file
    std::cout << "\n";
    std::cout << "ERROR: An error was detected while reading deviceprefs.ini.\n";
    std::cout << "Error found on line: " << thisTextLine << "\n";
    std::cout << "\n";
    return false;
}

    // If no errors until now, all has been successful
return true;
}
```

```
// ************************************************************
// DevicePrefs::SplitString
//
// This function takes a property assignment line (format:
// property=value), splits it on the = sign, and creates a 2-element
// array containing the property and the value.
//
// Input: String (format: property=value)
// Return: Array of property and value strings
// ************************************************************
std::vector<std::string> DevicePrefs::SplitString(std::string x)
{
   std::stringstream stream(x);
   std::string field;
   std::vector<std::string> y;

   // Split string on =, create 2-element array with property
   // and value
   while( getline(stream, field, '=') )
   {
      y.push_back(field);
   }

   return y;
}

// ************************************************************
// DevicePrefs::GetDoubleValue
//
// This function takes a string that actually contains a double,
// and returns the value in a double variable type.
//
// Input: String
// Return: Double version of the input string
// ************************************************************
double DevicePrefs::GetDoubleValue(std::string x)
{
   double myReturn = 0;
   myReturn = atof(x.c_str());
   return myReturn;
}
```

```
// ************************************************************
// DevicePrefs::GetIntValue
//
// This function takes a string that actually contains an integer,
// and returns the value in an integer variable type.
//
// Input: String
// Return: Integer version of the input string
// ************************************************************
int DevicePrefs::GetIntValue(std::string x)
{
   int myReturn = 0;
   myReturn = atoi(x.c_str());
   return myReturn;
}
```

```
// *** Utilities.h ***

// ********************************************************************
// MFSFET Device and Circuit Simulator
// Cody Mitchell, Master's Thesis Research
// Advisor: Dr. Fat Duen Ho
// Fall 2012
//
// Class: Utilities
// Purpose of class: This class is used to provide a few useful
//      functions that can be used independent of a particular
//      circuit or device.
// ********************************************************************

#ifndef _UTILITIES_H_
#define _UTILITIES_H_

#include <string>

class Utilities
{
public:
        // Get the difference between two double values
   static double GetDifference(double, double);

   // Determine if a particular value is within a desired
   // tolerance by comparing it to a baseline value
   static bool InTolerance(double, double);

   // Determine if a particular value is within the bounds of
   // two other values, and reset it to the nearest boundary
   // value if it is outside the range
   static double Bound(double,double,double);

   // Get the sign of a number and return as a positive or
   // negative multiple of 1
   static int Sign(double);

   // Convert a double value to a string
   static std::string DblToString(double);
};


#endif //_UTILITIES_H_
```

```
// *** Utilities.cpp ***

// ********************************************************************
// MFSFET Device and Circuit Simulator
// Cody Mitchell, Master's Thesis Research
// Advisor: Dr. Fat Duen Ho
// Fall 2012
//
// Class: Utilities
// Purpose of class: This class is used to provide a few useful
//      functions that can be used independent of a particular
//      circuit or device.
// ********************************************************************

#include "Utilities.h"
#include <cmath>
#include <sstream>

// ********************************************************************
// Utilities::GetDifference
//
// This function returns the absolute difference between two
// provided double values.
//
// Input: 2 double values
// Return: Double value (the difference)
// ********************************************************************
double Utilities::GetDifference(double x, double y)
{
   return fabs(x - y);
}

// ********************************************************************
// Utilities::InTolerance
//
// This function determines if a value is within a specified
// tolerance.
//
// Input: Value to be checked, Tolerance to verify
// Return: Boolean
//      True - "value" is less that the "tol" tolerance
//      False - "value" is not within the "tol" tolerance
// ********************************************************************
```

```
bool Utilities::InTolerance(double value, double tol)
{
   // If the value passed in is equal to or less than
   // the specified tolerance, return true since it
   // is "in tolerance"
   if (value <= tol)
      return true;
   else
      return false;
}

// ****************************************************************
// Utilities::Bound
//
// This function determines if a value is within a specified
// range of two other values. If the value is outside the range,
// the function returns the closest bounding value. Otherwise, the
// original value is returned.
//
// Input: Value to be checked, two bound values
// Return: New double value properly bounded in the correct range
// ****************************************************************
double Utilities::Bound(double var, double v1, double v2)
{
   if (v1 < v2)
   {
      if (var < v1) var = v1;
      else if (var > v2) var = v2;
   }
   else
   {
      if (var < v2) var = v2;
      else if (var > v1) var = v1;
   }

   return var;
}
```

```
// ****************************************************************
// Utilities::Sign
//
// This function determines the sign of a number and returns a
// negative or positive multiple of 1 based on that sign.
//
// Input: Double value to be checked
// Return: Integer multiple of 1 based on sign of value
//      -1 if value was negative
//      0 if value was 0
//      1 if value was positive
// ****************************************************************
int Utilities::Sign (double a) {
   if (a > 0) return 1;
   else if (a < 0) return -1;
   else return 0;
}


// ****************************************************************
// Utilities::DblToString
//
// This function converts a double value to a string.
//
// Input: Double value to be converted
// Return: String value
// ****************************************************************
std::string Utilities::DblToString(double x)
{
   std::ostringstream myStream;
   myStream << x;
   std::string newString = myStream.str();
   return newString;
}
```

REFERENCES

[1] M. Lines and A. Glass, *Principles and applications of ferroelectrics and related materials*. Oxford, UK: Clarendon Press, 1977.

[2] J. Fousek, "Joseph Valasek and the Discovery of Ferroelectricity," in *Proc. 9th IEEE Int. Symp. on Applicat. of Ferroelectrics*, University Park, Aug. 1994.

[3] A. Ballato, "Piezoelectricity: history and new thrusts," in *Ultrasonics Symposium, 1996. Proc. IEEE*, vol. 1, pp. 575-583, Nov. 1996.

[4] P. Moseley and J. Crocker, *Sensor Materials*. Bristol, UK: Institute of Physics Publishing, 1996.

[5] M. Sadiku, *Elements of Electromagnetics*, 3rd ed. New York: Oxford University Press, 2001.

[6] R. Waser, Ed., *Nanoelectronics and Information Technology: Advanced Electronic Materials and Novel Devices*, 3rd ed. Weinheim, Germany: Wiley-VCH, 2012.

[7] D. Damjanovic, "Ferroelectric, dielectric and piezoelectric properties of ferroelectric thin films and ceramics," *Reports on Progress in Physics*, vol. 61, no. 9, pp. 1267-1324, 1998.

[8] J. Evans, "Modeling Radiant Thin Ferroelectric Film Transistors," Albuquerque, NM, 2011.

[9] J. Srivastava, *Elements of Solid State Physics*, 2nd ed. New Delhi, India: Prentice-Hall of India, 2006.

[10] I. Misirlioglu, A. Vasiliev, S. Alpay, M. Aindow, and R. Ramesh, "Defect microstructures in epitaxial $PbZr_{0.2}Ti_{0.8}O_3$ films grown on (001) $SrTiO_3$ by pulsed laser deposition," *J. Mater. Sci.*, vol. 41, no. 3, pp. 697-707, 2006.

[11] R. Wolf and S. Trolier-McKinstry, "Temperature dependence of the piezoelectric response in lead zirconate titanate films," *J. Appl. Physics*, vol. 95, no. 3, pp. 1397-1406, Feb. 2004.

[12] A. Sedra and K. Smith, *Microelectronic Circuits*, 5th ed. New York: Oxford, 2004.

[13] Y. Tsividis, *Operation and Modeling of the MOS Transistor*, 2nd ed. New York: Oxford University Press, 1999.

[14] F. Ho, "Testing of Semiconductor/Ferroelectric NDRO Transistors," Final Report for the MFS NDRO Research Project, Huntsville, AL.

[15] R. Sayyah, "Characterization of the Ferroelectric Transistor and its Applications in Analog Circuits," M.S.E. thesis, Dept. of Elect. and Comput. Eng., Univ. of Alabama in Huntsville, AL, 2009.

[16] J. Scott, *Ferroelectric Memories*. Berlin, Germany: Springer-Verlag, 2000.

[17] T. MacLeod and F. Ho, "Ferroelectric Field Effect Transistor Model Using Partitioned Ferroelectric Layer and Partial Polarization," *Integrated Ferroelectrics*, vol. 64, no. 1, pp. 89-100, Jan. 2004.

[18] S. Miller, R. Nasby, J. Schwank, M. Rodgers, and P. Dressendorfer, "Device modeling of ferroelectric capacitors," *J. Appl. Physics*, vol. 68, no. 12, pp. 6463-6471, Dec. 1990.

[19] S. Kang and Y. Leblebici, *CMOS Digital Integrated Circuits: Analysis and Design*, 3rd ed. New York: McGraw-Hill, 2003.

[20] C. Mitchell, C. Laws, T. MacLeod, and F. Ho, "Static Characteristics of the Ferroelectric Transistor Inverter," *Integrated Ferroelectrics*, vol. 125, pp. 123-129, 2011.

[21] C. Laws, C. Mitchell, T. MacLeod, and F. Ho, "Switching Characteristics of Ferroelectric Transistor Inverters," *Integrated Ferroelectrics*, vol. 125, pp. 141-146, 2011.

[22] J. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, 2nd ed. Upper Saddle River: Pearson Education, Inc., 2003.

[23] T. Ma and J. Han, "Why is Nonvolatile Ferroelectricity Memory Field-Effect Transistor Still Elusive?," *IEEE Electron Device Lett.*, vol. 23, no. 7, pp. 386-388, July 2002.

[24] S. Miller and P. McWhorter, "Physics of the ferroelectric nonvolatile memory field effect transistor," *J. Appl. Physics*, vol. 72, no. 12, pp. 5999-6010, 1992.

[25] M. Bailey and F. Ho, "Metal-Ferroelectric-Semiconductor Field-Effect Transistor Modeling Using a Partitioned Ferroelectric Layer," *Integrated Ferroelectrics*, vol. 51, no. 1, pp. 19-37, 2003.

[26] R. Muller and T. Kamins, *Device Electronics for Integrated Circuits*, 3rd ed. New York: John Wiley & Sons, 2003.

[27] J. Phillips. *Online Curve Fitting and Surface Fitting at ZunZun.com*. [Online]. Available: http://www.zunzun.com. [Accessed: May 22, 2011].