ABSTRACT

COLEBAUGH, SARAH. Adaptation of a CMOS Reliability Simulation Model for the Open Model Interface. (Under the direction of Dr. William Rhett Davis.)

The HiSIM2 transistor model released by Hiroshima University can be used to predict the effects of hot carrier injection and bias temperature instability using a new charge trapping. However, this model has been written in Verilog-A which makes it difficult to use with most reliability simulators that are currently being used in industry. This paper goes through the steps that are needed to take in order to convert the HiSIM2 model from Verilog-A to C so that it may be used with the Open Model Interface and then incorporated into commercial reliability simulators.

Adaptation of a CMOS Reliability Simulation Model for the Open Model Interface

by
Sarah Colebaugh

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Electrical Engineering

Raleigh, North Carolina

2021

APPROVED BY:

_____	_____
Dr. William Rhett Davis Committee Chair	Dr. Yaoyao Jia
_____	
Dr. David Ricketts	

BIOGRAPHY

The author received her bachelor's degree in Electrical Engineering at North Carolina State University in 2019 and is on track to complete her master's degree by May 2021, also in Electrical Engineering at North Carolina State University.

ACKNOWLEDGEMENTS

I would like to thank Dr. Davis for his guidance and knowledge throughout this process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

The ability to model and predict the effects of stress on transistors has become increasingly important in the past several years as industry keeps pushing their demand for increased performance in integrated circuits. By understanding the relationship between stress on a circuit and its lifetime, the reliability of that circuit can be calculated and used during the design phase of a project, rather than discovering it while the device is already out in the field. This idea of incorporating reliability simulators into the schematic design software will allow engineers to quickly test a variety of stress cases on their circuit without having to make a build, therefore saving time, saving money, saving resources, and increasing the overall reliability of their product. While most reliability simulation models have been proprietary, one state-of-the-art model has recently emerged that is open source.

1.1 HiSIM2

The reliability (or "aging") simulation model that will be presented in this paper is included as part of the HiSIM2 (Hiroshima-university STARC IGFET Model) compact simulation model from Hiroshima University. It is the first complete surface-potential-based MOSFET model for circuit simulation based on the drift-diffusion approximation. [1] There are two types of aging that are used in the model, the hot electron induced aging (HC aging) and the bias temperature instability aging (N(P)BTI aging). Both models have some effects on nMOSFET and pMOSFET, but the HC aging is mostly responsible for nMOSFET and the N(P)BTI aging is mostly responsible for pMOSFET. "The origin of the HC aging is modeled as the trap-density increase, which is included precisely in the Poisson equation solved iteratively"

as shown in Equation 1.1.1. [1][4] “The hole trapping at the Si/oxide interface of the NBTI is considered and modeled as the threshold voltage shift” as shown in Equation 1.1.2. [1][5]

$$\nabla^2\varphi = -\frac{q}{\varepsilon_{si}}(p - n + N_D^+ - N_A^- - N_{trap}(t)) \tag{1.1.1}$$

$$\Delta V_{th,trap} = TRAPA \cdot \exp(TRAPB \cdot E_{ox}) \cdot \left[1 - \exp\left(-\frac{t_s}{TRAPBTI}\right)\right] \tag{1.1.2}$$

$$E_{ox} = \frac{V_{Gon} - \Delta V_{th,trap} - \varphi_s}{T_{ox}}$$

1.2 The Problem

While the HiSIM2 reliability model offers an accurate prediction for device lifetime based on the effects of stress on the circuit, it was not written in a way that can easily be used in most reliability simulators. Like most transistor models today, HiSIM2 was written in Verilog-A, and commercial SPICE-like simulators offer a compiled version of it for fast simulation without the need of Verilog-A. However, these simulators omit HiSIM2’s aging model, due to its incompatibility with their proprietary reliability-model formats. This “language barrier” prevents HiSIM2’s open reliability model from being incorporated into commercial reliability simulators and does not allow the industry to take full advantage of this valuable reliability model.

1.3 Background

1.3.1 Brief History on Reliability Simulation

The need for high performance technology has been rapidly increasing in the aerospace world and has caused a push for the improvement and transformation of commercial grade products. [8] However, devices that will be placed into aircraft are expected to last longer than just the few years that most commercial grade parts are rated for. [8] By pushing and changing the limits of these products to improve the performance, there is an increased risk of experiencing the negative effects of hot carrier injection and negative bias temperature instability. If the impact of these aging mechanisms are too great, there is an increased likelihood of system failure and a delay on any device production. In order to keep the aerospace industry (and any other technologically advanced industry) satisfied, analysis must be done on the reliability of all high performance integrated circuits that are being put into their devices.

Before 1985, there were only device level reliability models that existed and they did not provide insight on where the hotspots or points of failure existed inside the device. It was a group of students at UC Berkeley that made “the first attempt to simulate actual circuit degraded behavior from transient device degradation calculations which ultimately resulted in the (CAS) and (BERT) reliability simulators.” [6] The methodologies used in BERT soon became the unofficial standard for all circuit level reliability models. [6] Several other reliability simulators were created after the release of BERT and although they all followed the same flow, they implemented different proprietary aging models. “Creating simulator-dependent aging models are very costly and time consuming let alone the confusion caused by the different simulation results among these simulators even though all these proprietary aging models are created from

the same data." [7] This issue was the cause for the implementation of "a unified aging model through TSMC Model Interface (TMI) - the industry standard model interface for circuit simulators." [7]

.

1.3.2 Open Model Interface

The TSMC Model Interface was licensed to the Silicon Integration Initiative Compact Model Coalition (CMC) who then expanded it to release an interface known as the Open Model Interface (OMI) which supports SPICE compact model extensions and can be used as a common platform for reliability simulators. [2] It is a C-language application programming interface that allows engineers to simulate and analyze physical effects, such as aging, on their circuit and optimize their design. [2] As mentioned in section 1.2, the HiSIM2 model has a "language barrier" that is preventing it from being incorporated into commercial reliability simulators. OMI can solve this issue, but first the relevant lines of Verilog-A code must be extracted from the HiSIM2 model and converted into the C-language. Determining which sections of the code need to be converted while maintaining the same level of accuracy is the purpose of this study. That also includes rewriting the lines in C, debugging, simulating the C code, and providing proof that a near zero percent error is achieved.

Chapter 2 will show an analysis of the HiSIM2 model and determine which of the internal nodes are of the most interest for the aging models. What the nodes of interest represent will also be described in this chapter. Chapter 3 will explain what steps were taken for the conversion from Verilog-A to C-code, including the method of data extraction, and provide an analysis of the error for the output.

CHAPTER 2: ANALYSIS OF HiSIM2 AGING MODEL

2.1 HiSIM2 Aging Simulation Verification

Before starting to convert parts of the HiSIM2 model to be used in OMI, we ran tests to verify that we were using the model properly. We ran verification tests on a 3-stage ring oscillator as shown in the schematic in Figure 1. The basic operation of this circuit is that the 2 inverters connected in series with the NAND gate create a positive feedback loop and have the output oscillate between 0V and 2V. This circuit was chosen for our verification simulation, because an earlier private presentation by the HiSIM2 developers showed their expectations for how much the oscillator frequency should decrease for various degradation times. We used these results as the expected output during testing. This chapter presents these verification tests in detail and how we used them to determine the sections of Verilog-A code to extract into OMI-compatible C-code.

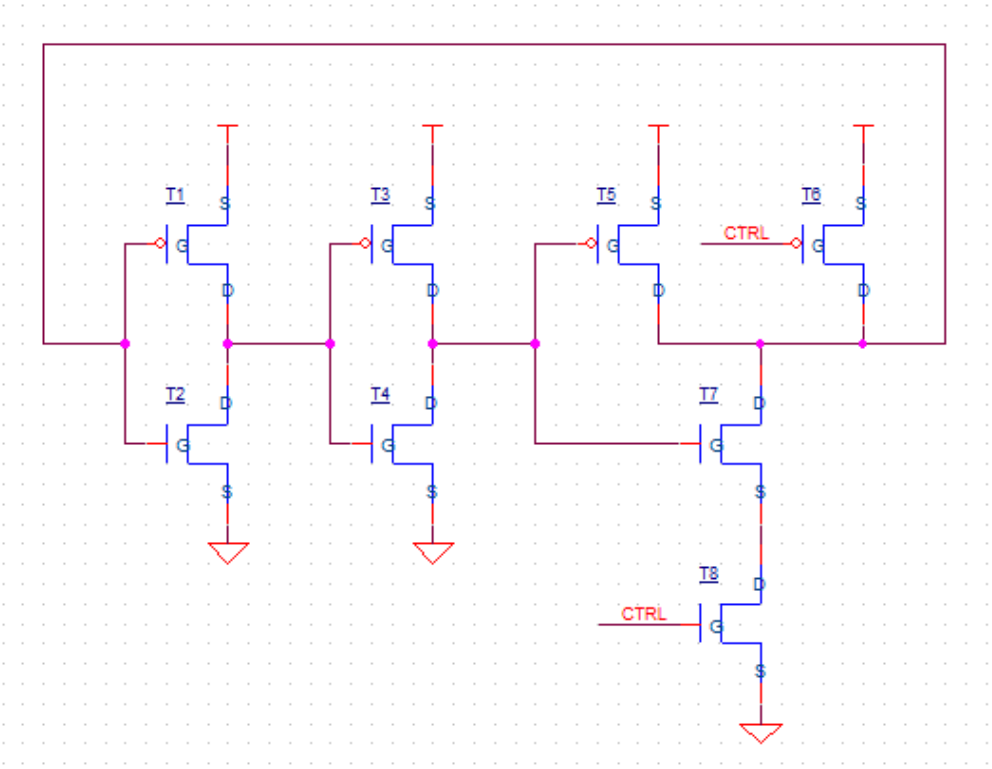

Figure 1: 3-Stage Ring Oscillator Schematic

Because we did not have access to the HiSIM2 developers' original circuit, we had to make educated guesses about the transistor parameters in Figure 1. We began with the Verilog-A default transistor parameters and made small changes to transistor parameters until the circuit showed degradation similar to the private HiSIM2 presentation. To make the transistors balanced in this circuit, PMOS transistors were given a width that was equivalent to twice the width of the NMOS transistor for their gate. Other parameters that were changed for each transistor can be seen in Table 1. The control input for this circuit was connected to the gate of T6 and T8 and was an input signal that had a linear rise from 0V to 2V in the first 1ns and then held high at 2V for the remaining simulation time. There was a 5ns delay before the signal began to rise where the input was held low for initialization of the circuit. A plot of this signal can be seen in Figure 2.

Table 1: 3-Stage Ring Oscillator Transistor Parameters

PARAMETER	VALUE			
TRANSISTOR TYPE	NMOS INVERTER	PMOS INVERTER	NMOS NAND GATE	PMOS NAND GATE
WIDTH	2.5 um	5 um	5 um	5 um
COISUB	1	-	1	-
SVDS	1.5	-	1.5	-
TRAPGCTIME2	1E9 s	-	1E9 s	-
TRAPA	-	0.3 V	-	0.3 V
TRAPB	-	4e-9 m/V	-	4e-9 m/V
TRAPBTI	-	1e8 s	-	1e8 s
CODEGSTEP	1	1	1	1

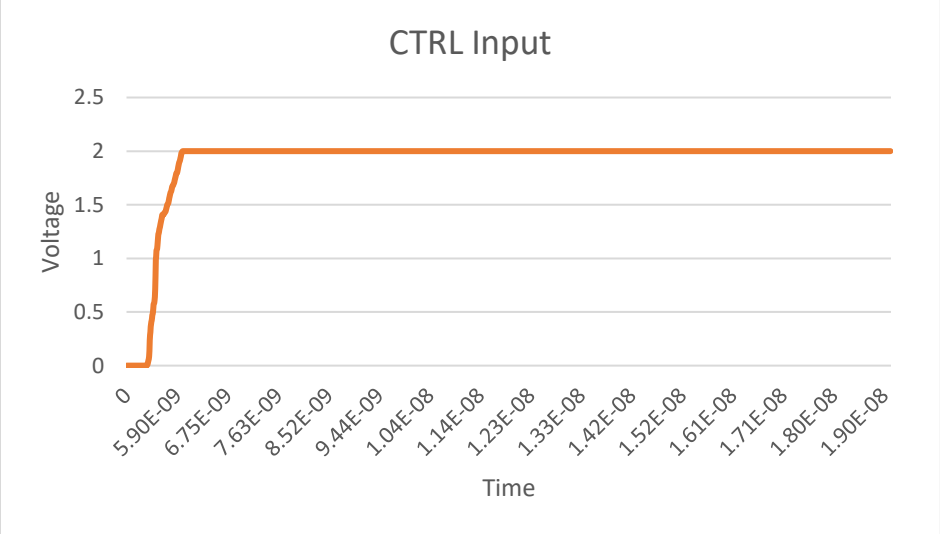

Figure 2: CTRL Input for 3-Stage Ring Oscillator

The first verification test was ran with a degradation time of 0. By probing at the drain of T6, a voltage plot was produced for the simulation time. The frequency was calculated from this plot by inverting its period. Four additional test were ran at degradation times of 1E3 seconds (~17 minutes), 1E5 seconds (~28hours), 1E7 seconds (~116 days), and 1E8 seconds (~3.17 years). For each of these times, the frequency was again calculated by inverting the period of the

output from the drain of T6. A plot of these five test points is shown in Figure 3 displaying a decrease in frequency as degradation time increases.

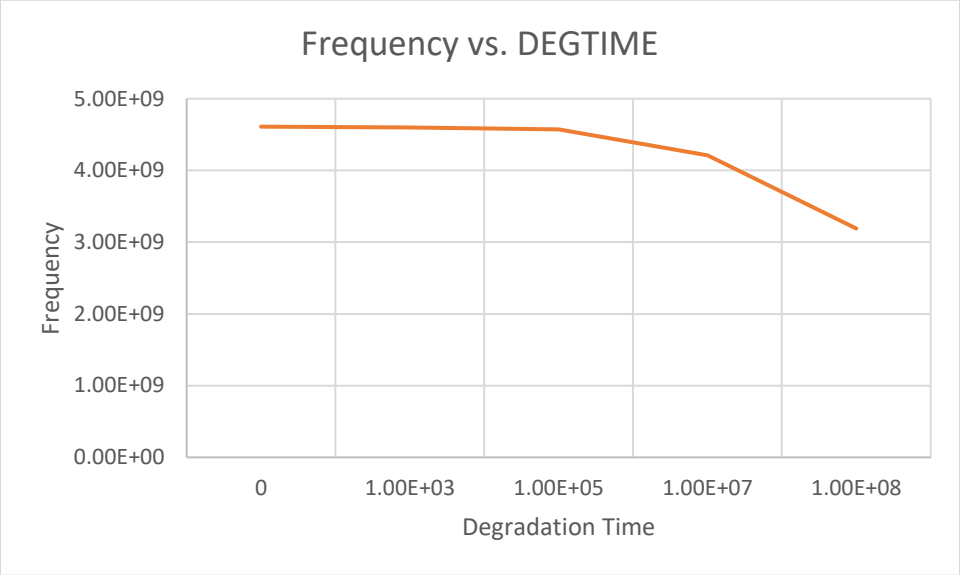

Figure 3: Frequency vs. Degradation Time, 3-Stage Ring Oscillator

For a degradation time of 0, the calculated frequency that the simulation produced was around 4.6 GHz. For a degradation time of 1E7 seconds, the calculated frequency decreased to about 4.16 GHz. By taking the percent difference between the two frequencies, we get a value of %10.05. Plots of the output from these verification test at degradation times 0 and 1E7 can be seen in Figure 4 and Figure 5. Although these values do not exactly match the values in the private HiSIM2 presentation, that presentation showed a similar curve to Figure 3 with a frequency decrease of 10.0% for a degradation time of 1E7. This fact leads us to the conclusion that our simulation is working as expected and can now be used for further studies. The formula used for percent difference can be seen in Equation 2.1.1.

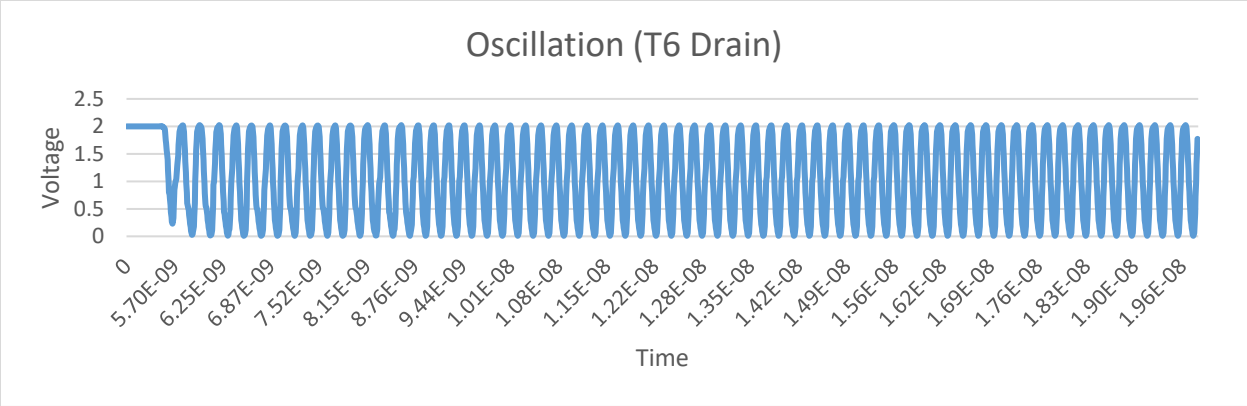

Figure 4: Output Oscillation of 3-Stage Ring Oscillator, DEGTIME = 0

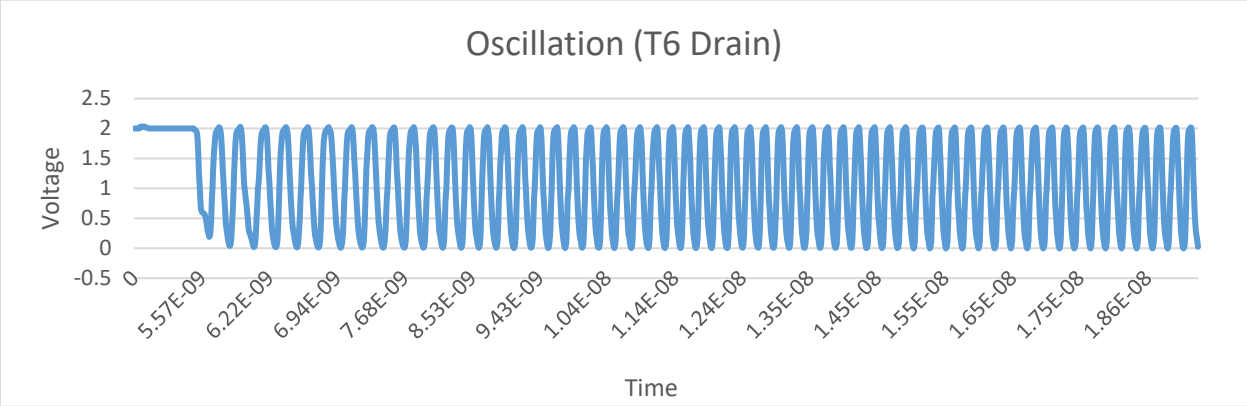

Figure 5: Output Oscillation of 3-Stage Ring Oscillator, DEGTIME = 1E7

$$\textit{Percent Difference} = (|a - b| / [(a + b) / 2]) * \%100 \qquad (2.1.1)$$

2.2 Determining the Nodes of Interest

After verifying the model operated properly, the next step was to determine which nodes were directly affected by a change in degradation time. The identifying behavior of a node that is directly affected would be a continuous increase in its voltage over time when the aging simulation is turned on. The reason why this is the identifying behavior is because the HiSIM2 model exhibits aging by having the affected node accumulate damage over time. This damage is

shown as voltage on the node. These internal nodes are used specifically for the aging integration and simulation. Some nodes represent different types of stress observed on the transistor. There are eight (8) aging node for the HiSIM2 model, which are easy to identify with the “ifdef AGING” directive around their declaration. They are named as idtag1, idtag3, vgc1, ve1, dVfb, idtag4, ps0, and vtraplx.

For this study, degradation time was set to 1E7 seconds and a simple inverter was used for the simulation. All transistor parameters from the ring oscillator were kept for the inverter. Each internal and external node voltage was monitored and the output was plotted against time. The first test was run with the aging simulation turned off to view what the node voltage behavior was without any stress. The second test was run with the aging models invoked. Based off the plots, it can be seen that the internal nodes idtag3 and idtag4 are directly affected by aging and display the identifying behavior mentioned in the paragraph above. Plots for these nodes, with and without aging, can be seen in Figure 7 through Figure 12. For reference, the input of the inverter is given in Figure 6.

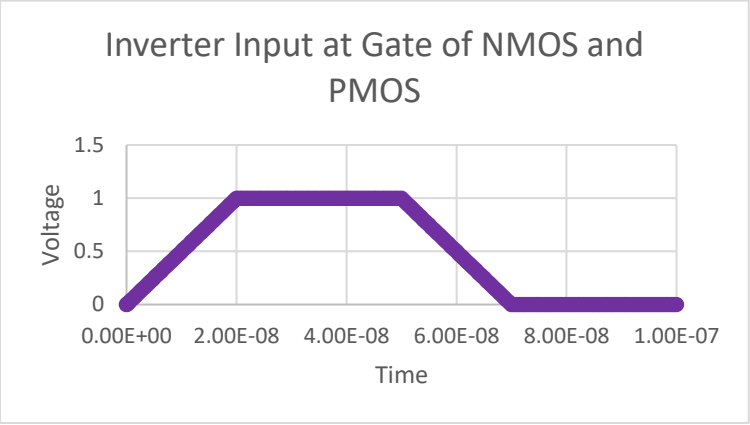

Figure 6: Inverter Input at Gate of NMOS and PMOS

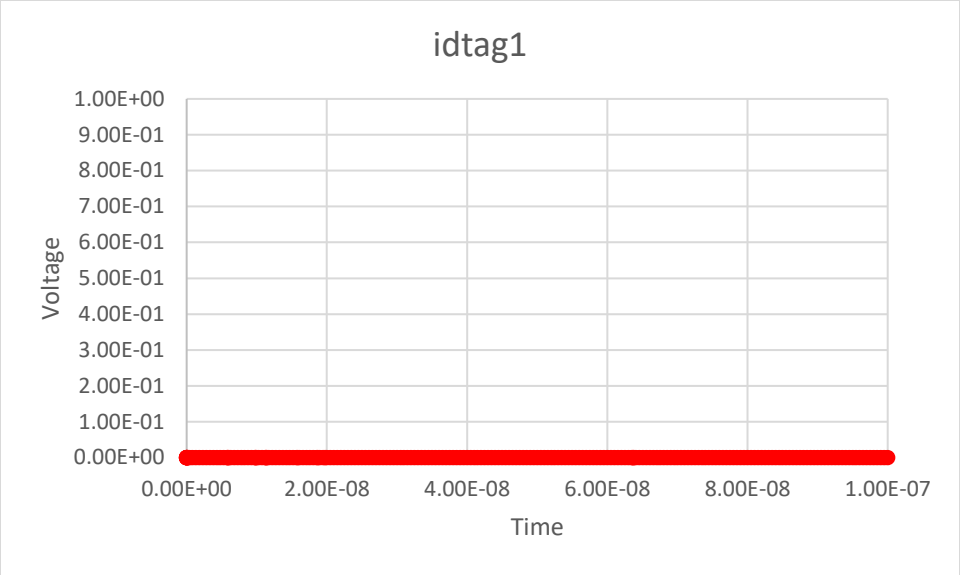

Figure 7: Voltage at Internal Node "idtag1", aging OFF

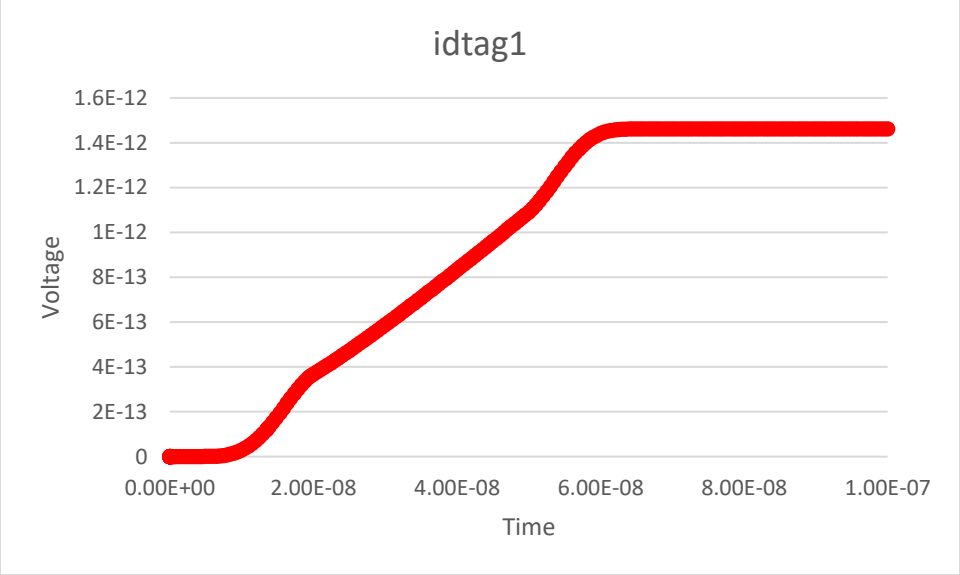

Figure 8: Voltage at internal Node "idtag1", aging ON

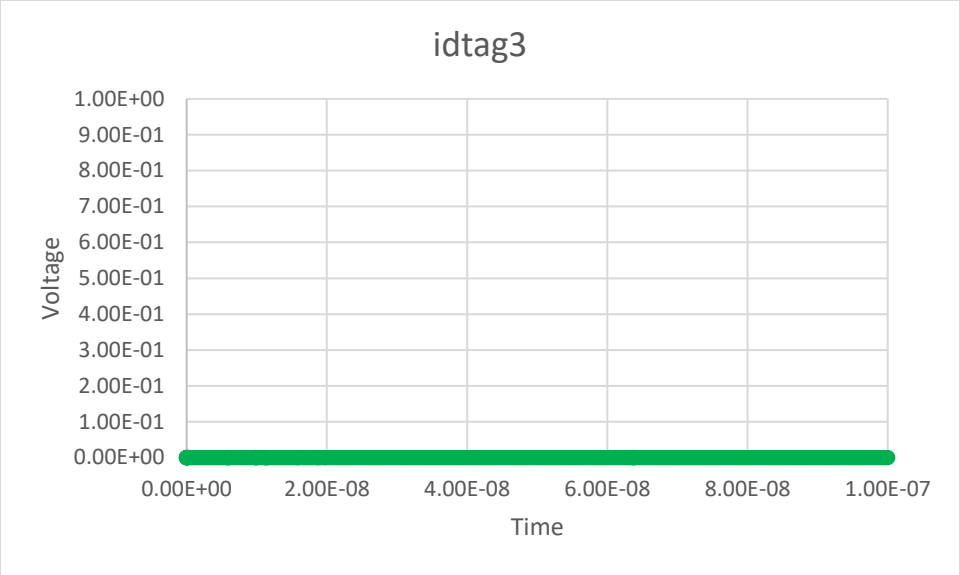

Figure 9: Voltage at Internal Node "idtag3", aging OFF

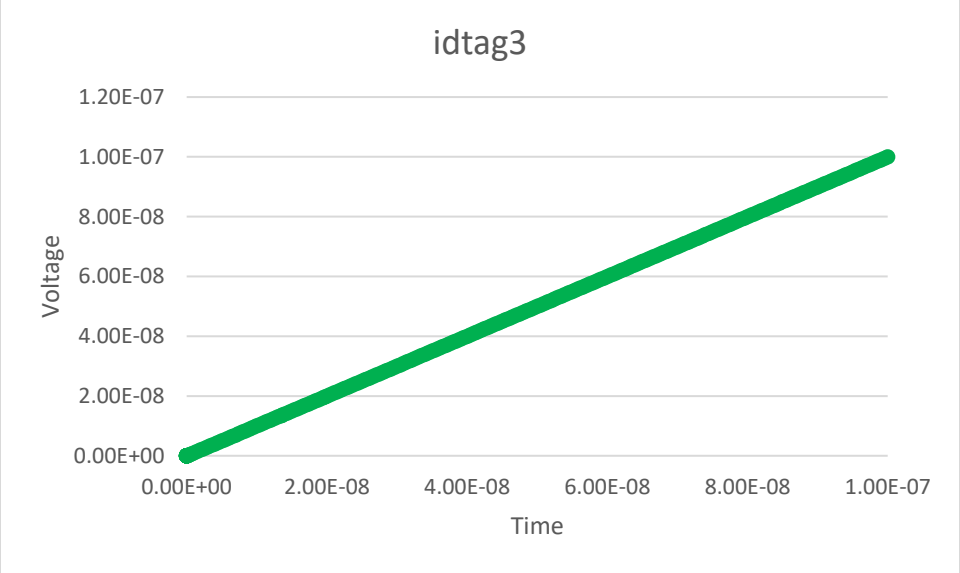

Figure 10: Voltage at Internal Node "idtag3", aging ON

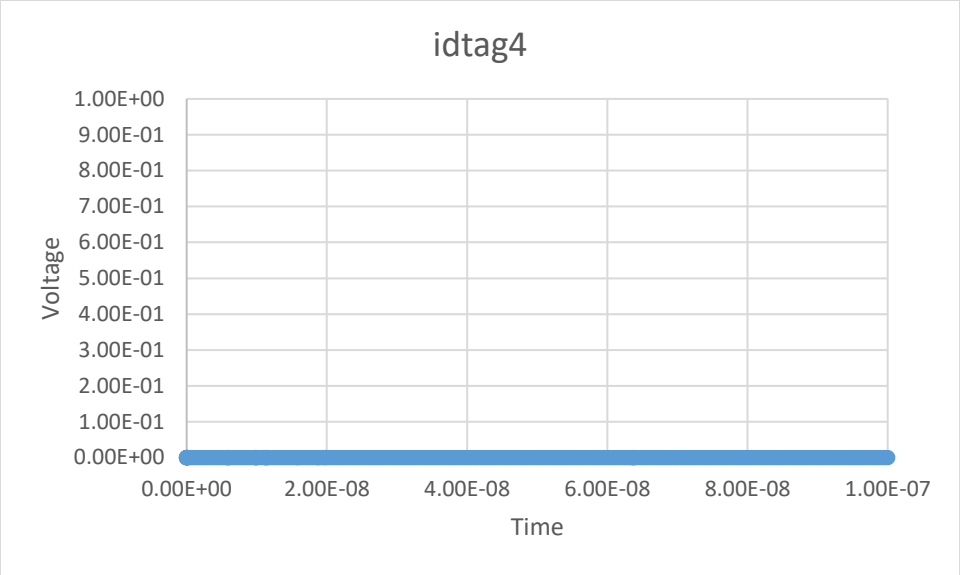

Figure 11: Voltage at Internal Node "idtag4", aging OFF

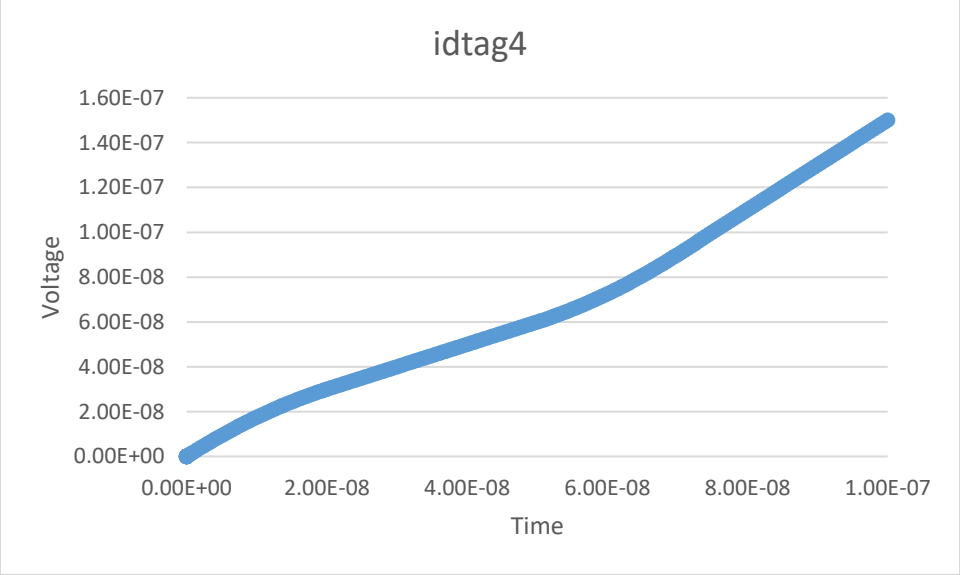

Figure 12: Voltage at Internal Node "idtag4", aging ON

2.3 Node Representation

It is important to understand what these 3 nodes of interest (idtag1, idtag3, and idtag4) represent. It was stated earlier in section 2.2 that the internal nodes all represent stress, but in this section the specifics of what each stress relates to will be explained. For idtag1, the stress being

modeled is related to the substrate current. The substrate current of a transistor is inversely related to the transistor's lifetime as shown in Figure 13. [3] Idtag3 and idtag4 both model stress related to the shift in the threshold voltage. Idtag3 tells if the transistor falls into cutoff mode (Vgs < Vth) or if the TRAPA density (the coefficient of existing interface trap density) drops below 0 and idtag4 gives the value of the voltage between the gate and the source. The stress related to the threshold voltage shift is part of the N(P)BTI aging mode whereas the stress related to the substrate current is part of the HC aging model.

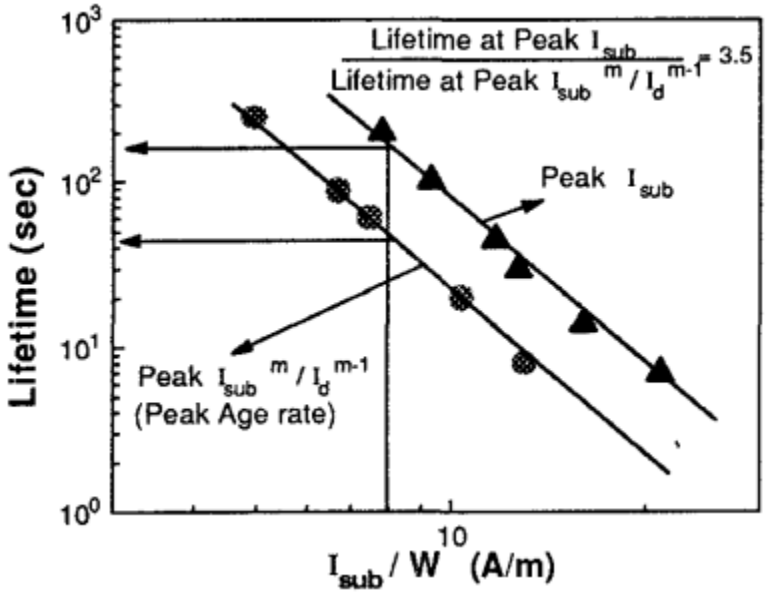

Figure 13: NMOS Lifetime vs. Substrate Current [3]

CHAPTER 3: CONVERTING VERILOG-A TO C

Now that the HiSIM2 model has been verified, we can begin the principal task of this study which is converting the Verilog-A code to C language. To avoid converting the entirety of the HiSIM2 model code, which is nearly 3,500 lines, we will focus on the portions of the code that are specific to the aging models and the internal nodes mentioned in section 2.2. Percent error will be calculated between the Verilog-A code and the C code to determine the accuracy of the conversion. The formula for percent error is given in Equation 3.0.1.

$$Percent\ Error = (|accepted - measured| / accepted) * \%100 \quad (3.0.1)$$

3.1 Conversion of the Code

The first step in the conversion process was to determine which blocks of the Verilog-A code needed to be converted to C to be used in OMI. At the end of the HiSIM2.va code there is a $fdisplay line showing which parameters are being written to the degradation file for aging simulation. For this line, the parameters of interest come from the code-block directly above it where the voltages for each internal node are defined for both the BTI model and the HC model. This block will serve as the root of the C conversion. However, since this block was picked from the end of the code, there are several parameters that need to be defined and equations to trace back through. The idea here was to start with the parameters that directly contributed to determining the node voltage and trace backwards until an operating point parameter (OPP) was reached. A list of associated OPPs would then be written to a text file. This text file would then be read by the C code and the parameters would be used to compute the value of each of the internal nodes. Any additional lines or sub-functions that were traced through when finding the

associated OPPs were added to the C code. A copy of the C code can be seen in the appendix of this paper.

3.2 Method of Data Extraction

Since we are not running the entirety of the HiSIM2 model, we must provide a way for the necessary data to get extracted from the simulator. We must also provide a way for data to be extracted from the C code so that it can be used when calculating percent error. The method for extracting data from the HiSIM2.va model and the C code were the same in theory. At each time step during the simulation, any necessary parameters or values were written to a text file along with their associated transistor's reference designator (i.e. instance name). Each entry was printed on a new line and would begin with that reference designator in order to allow easy parsing of values from the simulation output. For example, for the text file created from the C code each line printed the following values in order: reference designator, abstime, idtag voltage, contribution value, time step.

This text file was then be sent through a python script where it was parsed and converted to a .csv file. Opening that .csv file in Microsoft Excel then allowed for plots to be created and easily examine percent error, voltage values, and other outputs of the simulation. An example of the Python code used to parse and convert the text file can be seen in Figure 14.

Python Code Example for Parsing and Conversion to .csv

```
import csv, re

with open('idtag1.txt', 'r') as in_file:
        open('idtag1parse.txt', 'w').close()
```

```
f= open("idtag1parse.txt", "a+")
for line in in_file:
        m=re.search(r"T0.fet.*", line)
        if m:
                f.write(line)
        else:
                continue
f.close()
with open('idtag1parse.txt', 'r') as in_file:
        lines = (line.split() for line in in_file)
        with open('idtag1.csv', 'w') as out_file:
                writer = csv.writer(out_file)
                writer.writerows(lines)
```

Figure 14: Python Parsing Code Example

3.3 Diminishing Error

Now that we have compiled the C code, we want to validate that the code is correct. This can be done by calculating the percent error of the output when comparing the C code to the HiSIM2.va model. Focusing on the nodes of interest mentioned in section 2.2 (idtag1, idtag3, and idtag4), an integration function is performed for each node and then the voltage levels at that node are used for calculating percent error. For these three nodes, a small percent error is found at the output. This error is due to the differences in methods of integration between the two languages. Verilog-A specifies the idt() function identifier to indicate integration, but it does not specify which integration technique to use. For C, there is not a built-in time integrator operator so a method of integration must be chosen and explicitly written out. For simplicity's sake, the Backward Euler Method was selected. The formula for this method can be seen in Equation 3.3.1.

$$y_{k+1} = y_k + h*f(y_{k+1}, y_{k+1}) \quad (3.3.1)$$

h is the step size for the function

The percent error caused by the differences in integration methods is substantial (greater than 10%) for about the first 8 time steps. After that, it rapidly diminishes to a percent error that can be considered negligible (less than 0.5%). Plots of relative error for idtag1, idtag3, and idtag4 can been seen in Figure 18 to Figure 20.

Since the different integration methods make it difficult to determine the accuracy of the C code, it is important to focus on the contributions for each node. The HiSIM2 Verilog-A aging code uses the integration syntax "V(node) <+ idt(variable, 0)", where "<+" is the contribution operator. If the variables on the right-hand side of the contribution operator (i.e. the "contributions") match what is calculated in the HiSIM2.va code, then it can be assumed that, when using the same integration technique, the outputs would also match. A plot of relative error for the inputs to idtag1, idtag3, and idtag4 can be seen in Figure 15 to Figure 17. As shown in the figures, the percent error here is zero at all time steps with the exception of the two outlier data points in Figure 17 (an explanation for these outliers is given at the end of section 3.4.1). The zero percent error proves that the C code is just as accurate as the Verilog-A model and is ready for use with the OMI.

3.4 Analysis of Error

To determine the accuracy of the C code, certain outputs were printed to text files that were then converted to .csv files as shown in section 3.2. The outputs that were chosen for each idtag from the Verilog-A model were the abstime, the idtag value, and the contribution to the integration function. The outputs that were chosen for each idtag from the C code were the abstime, the idtag value, the contribution to the integration function, and the step size that was used for the integration function. These contributions to the integration functions were labeled T0 for idtag1, T2 for idtag3, and T3 for idtag4. A plot of relative error for each of the contributions can be seen in Figure 15 to Figure 17. A plot of relative error for each idtag value can be seen in Figure 18 to Figure 20.

3.4.1 Plot of Relative Error for Nodes of Interest

This section contains all output plots of relative error that were created during the comparison between the Verilog-A model and the C code. For these plots the red line will always represent the percent error. The other two lines show the value of the parameter of interest where green is the value found in the Verilog-A code and blue is the value found in the C code. Since there is a small, or zero, percent error between the two outputs, these last two lines will overlap with blue on top of green.

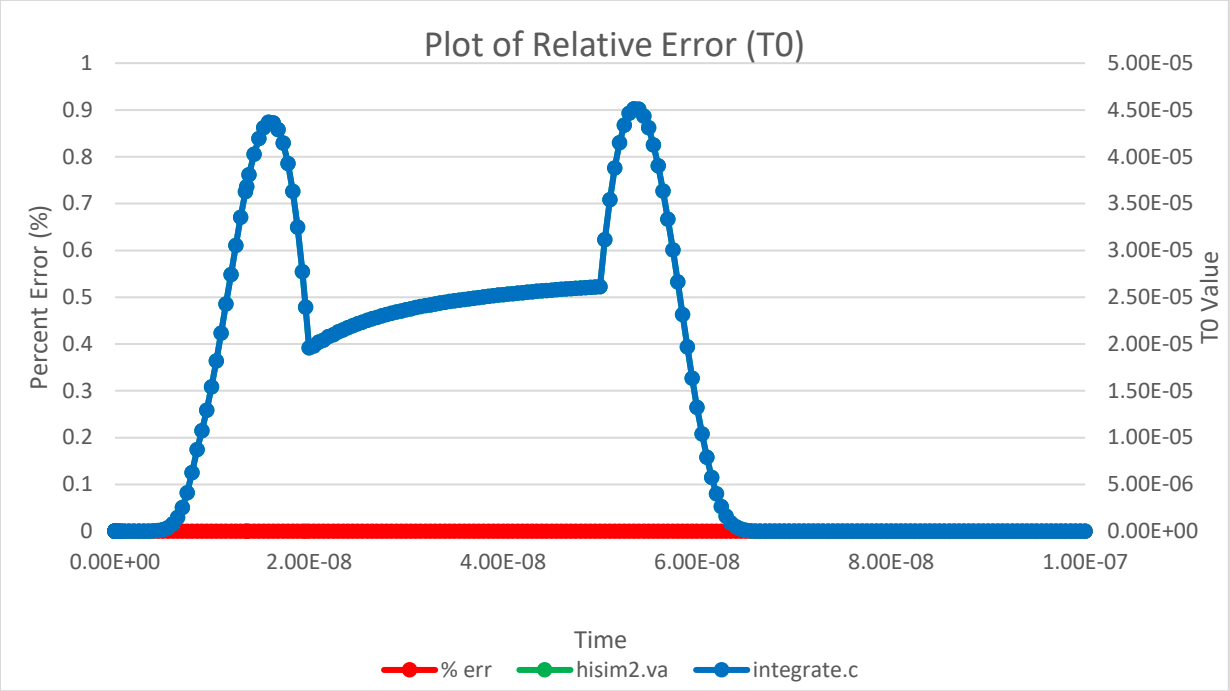

Figure 15: Plot of Relative Error for T0

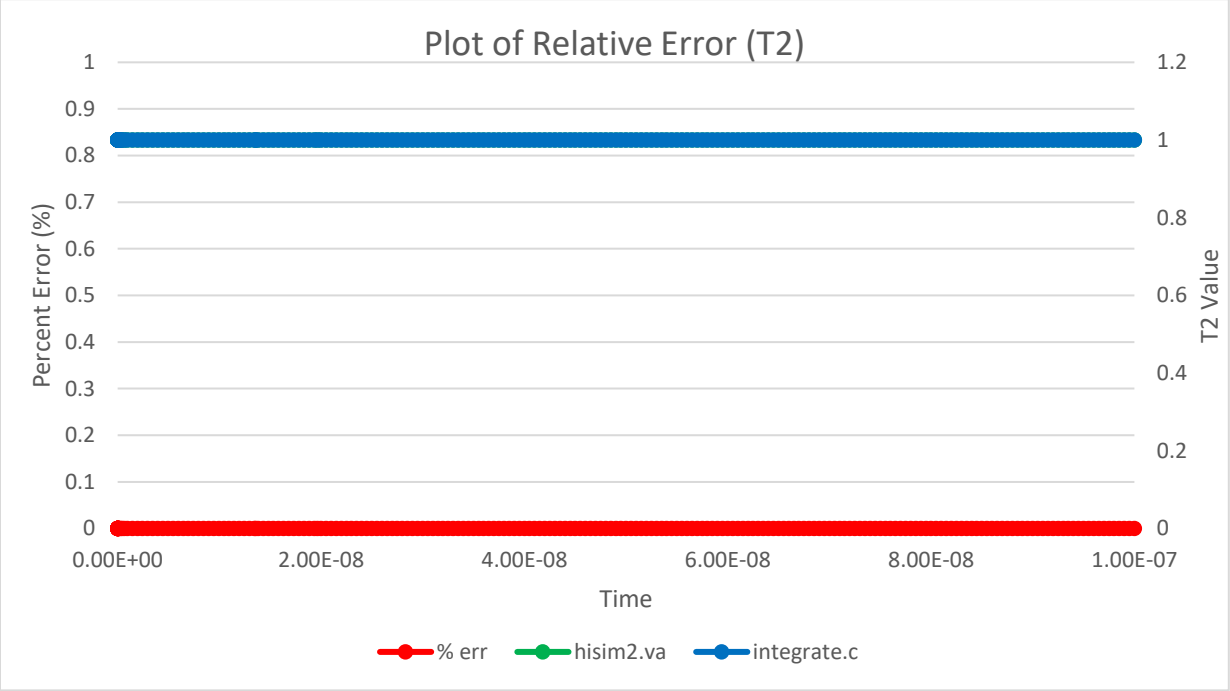

Figure 16: Plot of Relative Error for T2

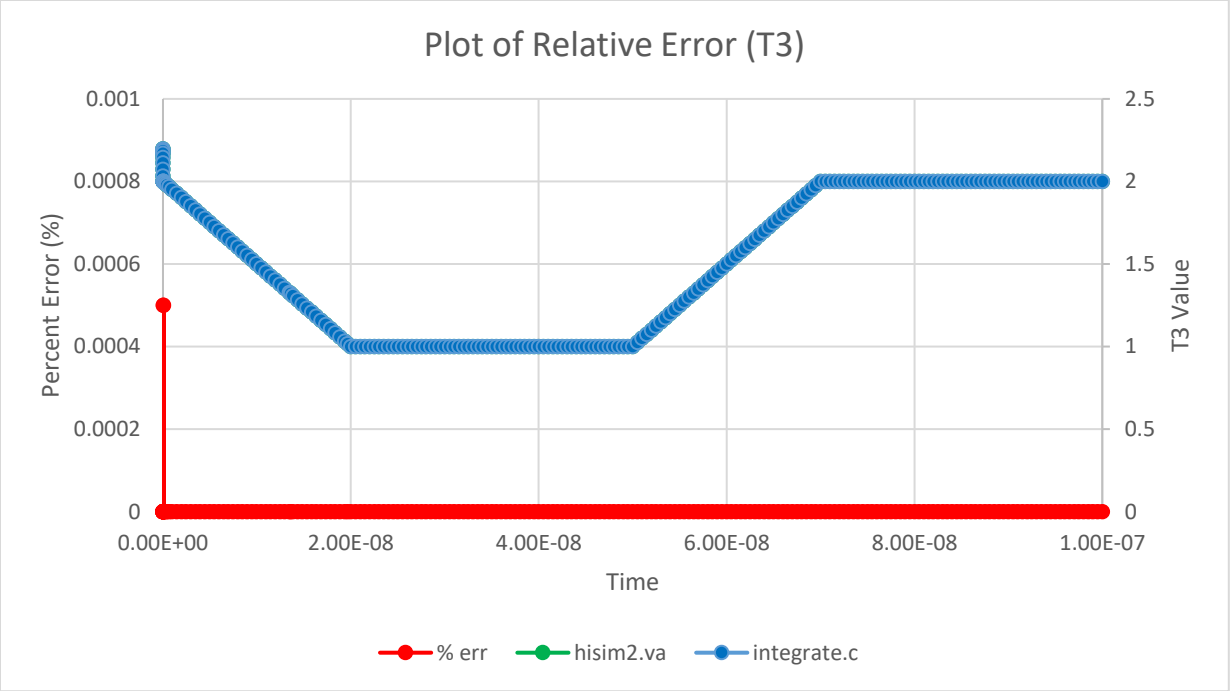

Figure 17: Plot of Relative Error for T3

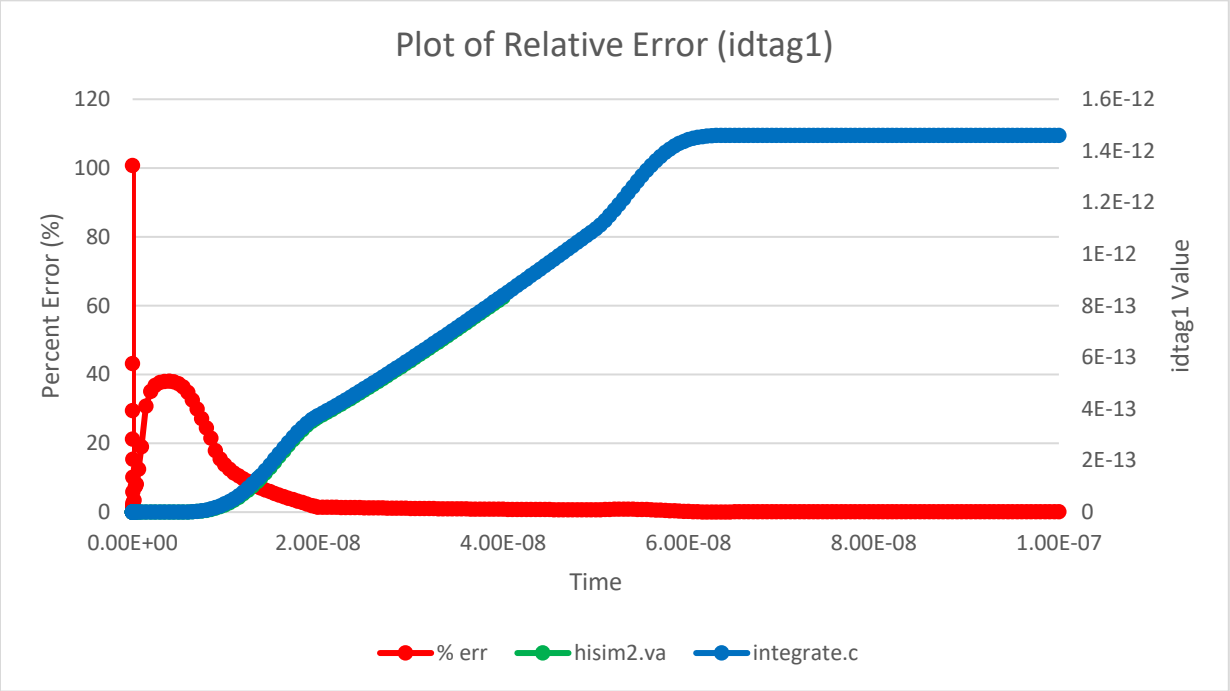

Figure 18: Plot of Relative Error for idtag1

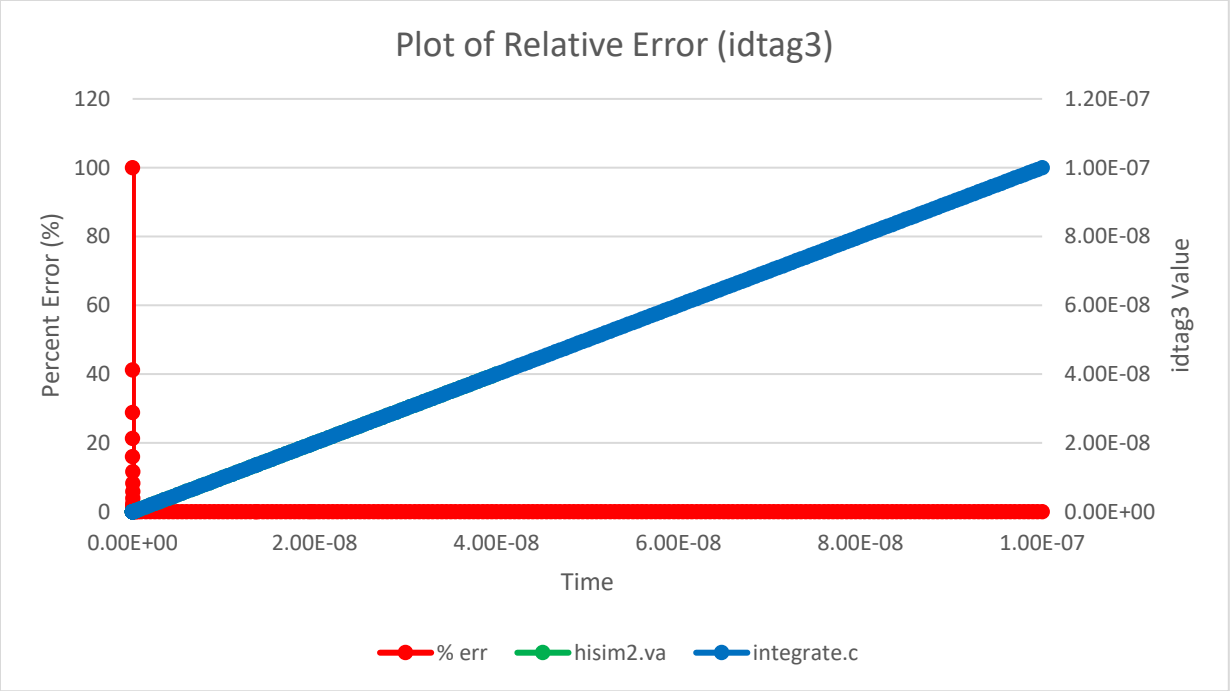

Figure 19: Plot of Relative Error for idtag3

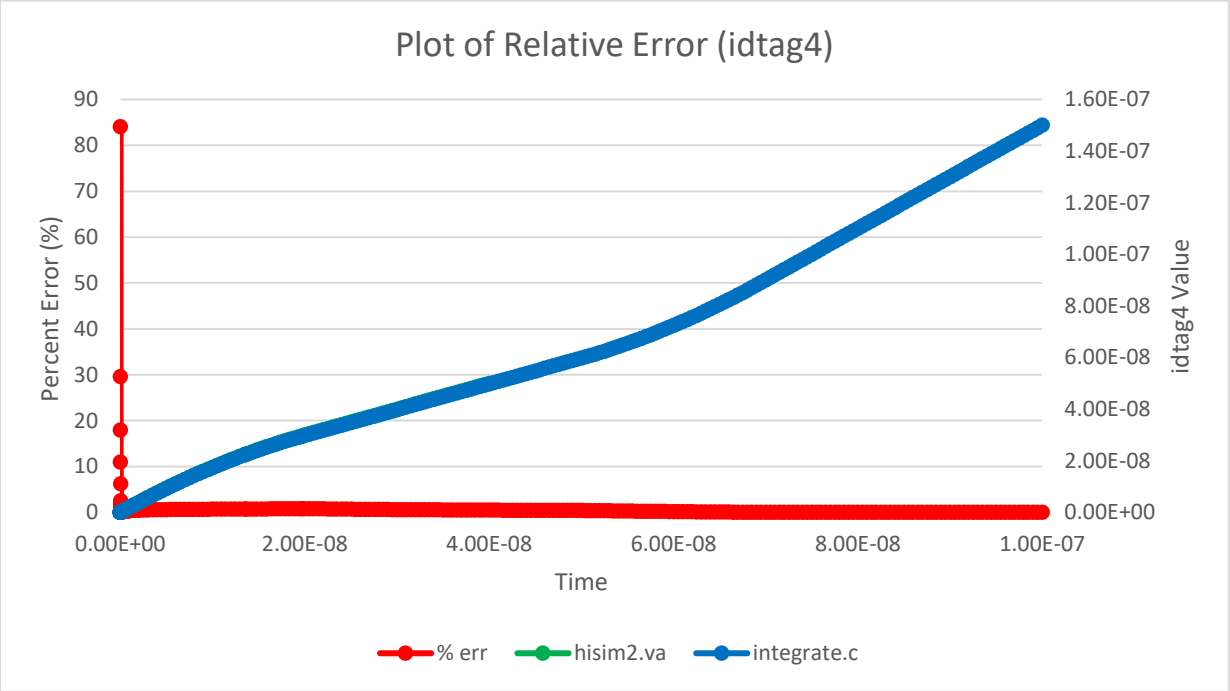

Figure 20: Plot of Relative Error for idtag4

Reasoning for Error on T3

Figures 15 and 16 show that the contributions for idtag1 and idtag3 in the Verilog-A simulation and extracted C code match with no error for the entirety of a 100ns simulation. However, there are two (2) data points on Figure 17 where the percent error is .0005%. This error is likely due to the fact that six significant digits were printed in the Verilog-A simulation output of the operating-point information that was used as input to our extracted C-code model. Because these values are double-precision floating-point types in the Verilog-A simulation, a small rounding error appears to have been introduced for these two points. Table 2 shows the exact values as printed by the Verilog-A simulation and our C-code in context with the previous and subsequent time steps. This small amount of overall error shows a high likelihood that our C-code has represented the Verilog-A code perfectly. Using more significant digits in the Verilog-A output would increase this certainty.

Table 2: T3 Percent Error Sample

time	T3 Value		percent error
	hisim2.va	integrate.c	
9.28E-12	1.99954	1.99954	0
1.21E-11	1.99939	1.9994	5.002E-04
1.70E-11	1.99915	1.99915	0
2.67E-11	1.99867	1.99867	0
4.47E-11	1.99779	1.9978	5.006E-04
7.89E-11	1.99611	1.99611	0

3.4.2 Accuracy for Other Internal Nodes

Although the remaining 5 internal nodes are not the main interest of this study, it is still important to examine their accuracy in the C code. The reason why they were not of interest is because they either maintained a consistent value during aging, or their voltage level did not continuously increase. It will be of interest in future studies to figure out why these nodes did not show a continuous rise indicating accumulating damage. For sake of time, a deeper dive was not taken for these 5 nodes to find a way to further lower their percent error. A plot of the node voltage and percent error was created for dVfb, vgc1, ve1, ps0, and vtraplx can be seen in Figure 21 to Figure 25. For these plots the red line will always represent the percent error. The other two lines show the node voltage where green is the value found in the Verilog-A code and blue is the value found in the C code. Since there is a small, or zero, percent error between the two values, these last two lines will overlap with blue on top of green or vise-versa.

Figure 21 shows a plot for the dVfb node. Examination of the Verilog-A code shows that it is related to the BTI model. We did not convert the Verilog-A equations for this node into C-code because its value is specified in the model as the operating point parameter DVTBTI which means that the calculations for this node may already be included as part of the HiSIM2 implementation in each commercial SPICE-like simulator. If so, then the OMI aging model would be able to reference this parameter rather than calculate its value. Discussion with the CMC HiSIM2 working group would be required to clarify the expectations for the dVfb equations.

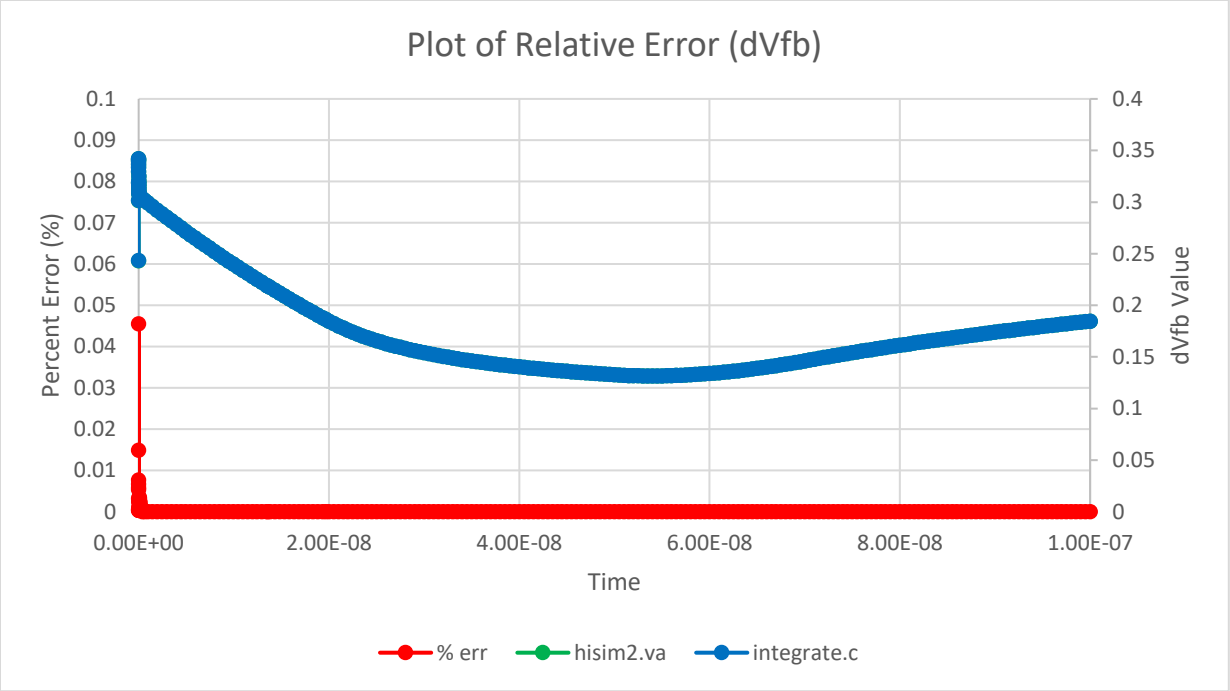

Figure 21: Plot of Relative Error for dVfb

Figure 22 and 23 show plots for the vgc1 and ve1 nodes. Examination of the Verilog-A code shows that they are related to the HC model. The contribution for these nodes is determined directly from the idtag1 node voltage. Cause for error on vgc1 and ve1 is idtag1 being a contribution to the equations that are used to calculate the voltage values for these nodes. As shown in Figure 18 idtag1 has a certain percent error, this error then propagates down to vgc1 and ve1.

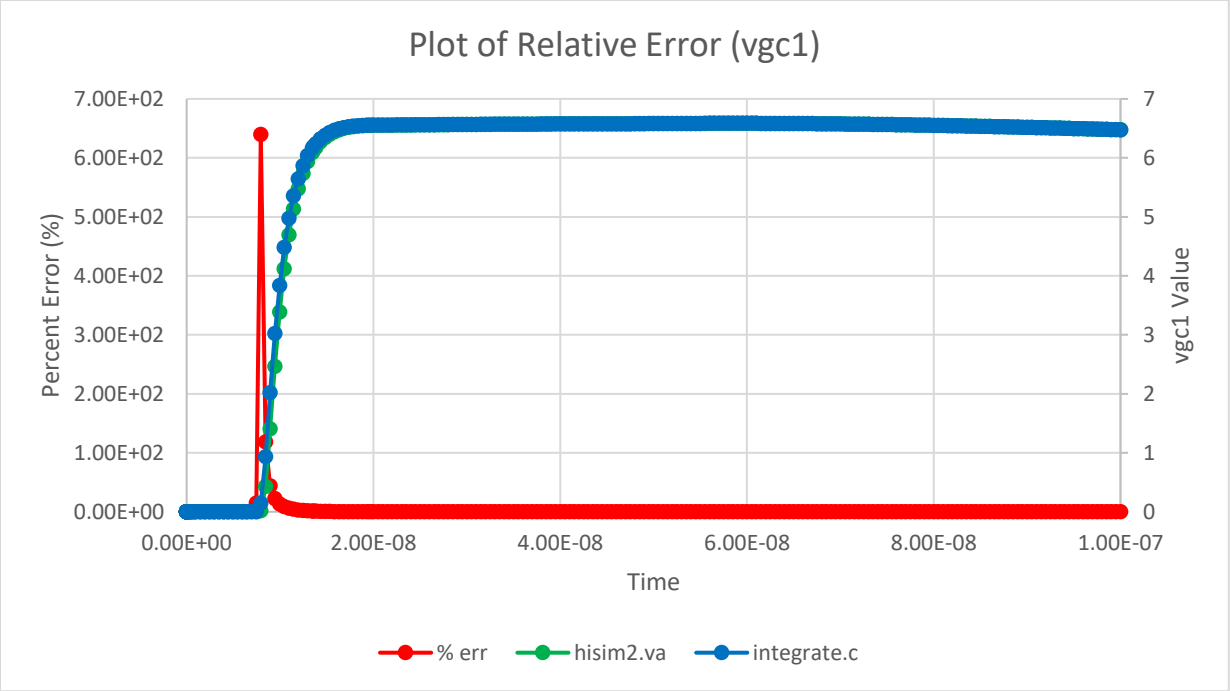

Figure 22: Plot of Relative Error for vgc1

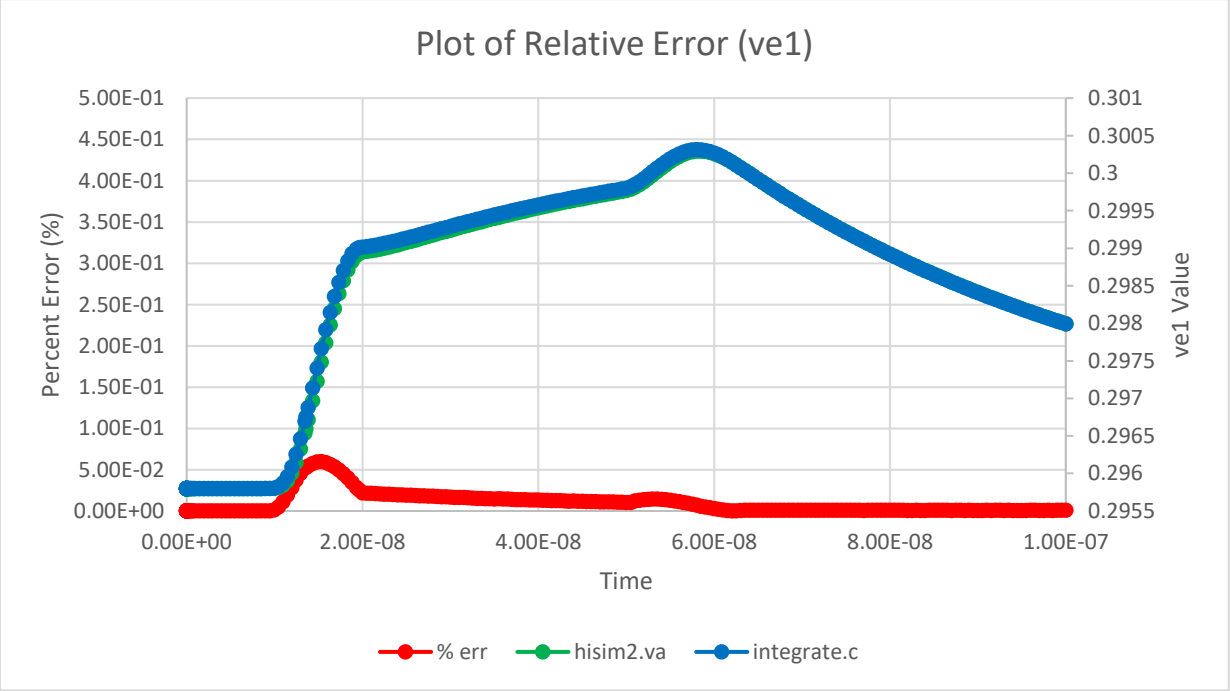

Figure 23: Plot of Relative Error for ve1

Figure 24 and Figure 25 show the percent error and node voltage for ps0 and vtraplx. By looking at the Verilog-A code, we can see that both of these nodes are part of the HC model. For these 2 nodes, the parameter CODEGSTEP had to change to 0 in order to get non-zero values for the nodes. Having CODEGSTEP set to 0 means that the model is set up for DC aging. Besides this error analysis for ps0 and vtraplx, CODEGSTEP was set to 1 (setup for circuit aging) for all other test and analyses.

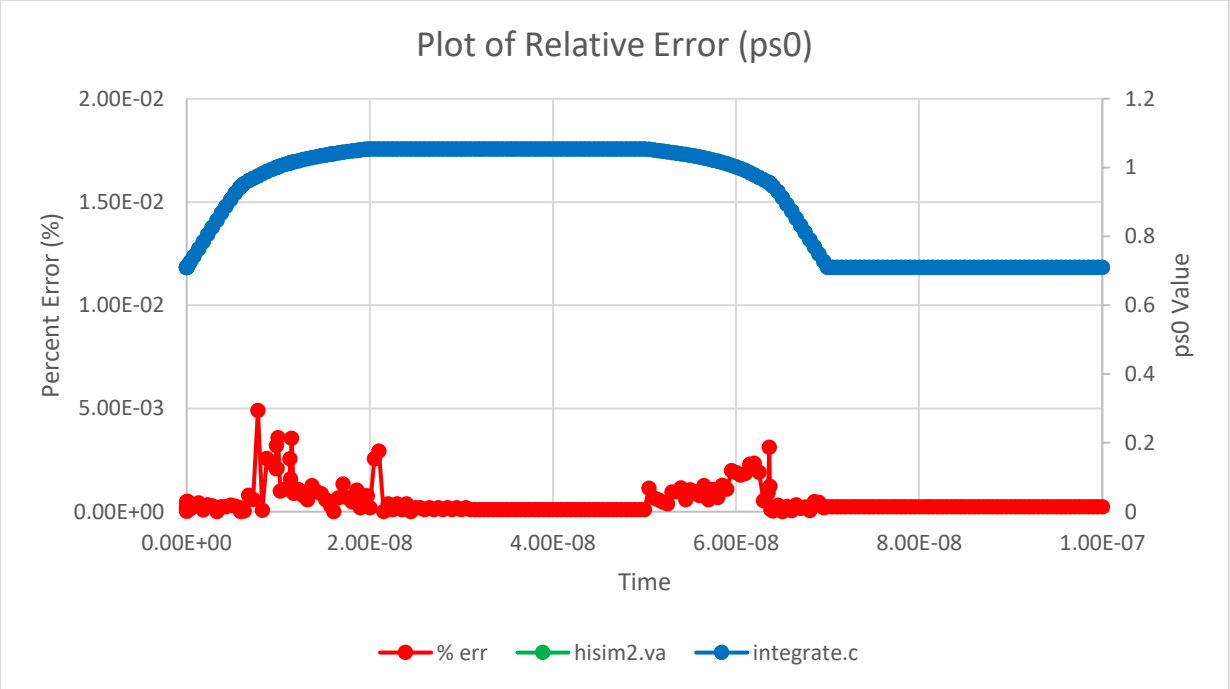

Figure 24: Plot of Relative Error for ps0

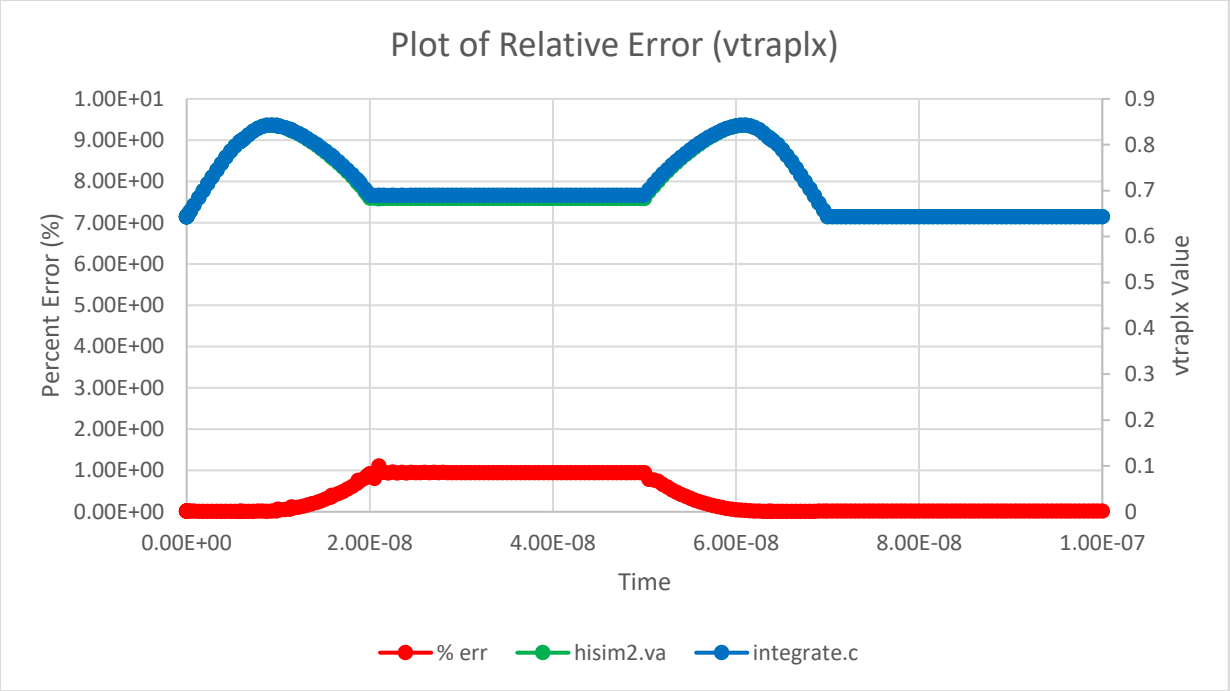

Figure 25: Plot of Relative Error for vtraplx

CHAPTER 4: CONCLUSION

Hiroshima University has developed a reliability model to predict the effects that stress from aging has on transistors. This type of reliability model is incredibly valuable for design engineers when testing out different designs for a circuit, especially circuits that are intended to have high performance. However, in order for the model to be used to its full potential it must be converted to a language that is compatible with modern reliability simulators. This can be accomplished by using the Open Model Interface (OMI) and converting parts of the model to C language. This study determined which parts of the Verilog-A code needed to be converted and then provided proof that the converted C code had nearly the same level of accuracy as the Verilog-A model.

REFERENCES

[1] M. Miura-Mattausch, T. Iizuka, H. Kikuchihara, H. Miyamoto, U. Feldmann, H. J. Mattausch, “HiSIM2 3.1.0 User’s Manual”, 2018

[2] Compact model coalition: Silicon integration initiative. (2021, February 23). Retrieved March 11, 2021, from https://si2.org/cmc/

[3] K. N. Quader, E. R. Minami, Wei-Jen Ko, P. K. Ko and Chenming Hu, "Hot-carrier-reliability design guidelines for CMOS logic circuits," in IEEE Journal of Solid-State Circuits, vol. 29, no. 3, pp. 253-262, March 1994, doi: 10.1109/4.278346.

[4] H. Tanoue, A. Tanaka, Y. Oodate, T. Nakahagi, C. Ma, M. Miyake, H. J. Mattausch, and M. MiuraMattausch, ”Universal Properties and Compact Modeling of Dynamic Hot-Electron Degradation in n-MOSFETs,” IEEE Int. Reliability Phys. Symp., pp. CM 4.1, April 2013.

[5] C. Ma, H. J. Mattausch, K. Matsuzawa, S. Yamaguchi, T. Hoshida, M. Imade, R. Koh, T. Arakawa, and M. Miura-Mattausch, ”Universal NBTI Compact Model for Circuit Aging Simulation under Any Stress Conditions,” IEEE Trans. Semiconductor Manufacturing, vol. 14, no. 3, pp. 818-825, 2014.

[6] P. M. Lee, "Compact modeling for simulation of circuit reliability: Historical and industrial perspectives," *2013 IEEE International Reliability Physics Symposium (IRPS)*, Monterey, CA, USA, 2013, pp. 2A.1.1-2A.1.6, doi: 10.1109/IRPS.2013.6531941

[7] M. Jeng, C. Hsiao, K. Su and C. Lin, "Circuit reliability simulation using TMI2," Proceedings of the IEEE 2013 Custom Integrated Circuits Conference, San Jose, CA, USA, 2013, pp. 1-7, doi: 10.1109/CICC.2013.6658491.

[8] M. Katoozi *et al*., "An age-aware library for reliability simulation of digital ICs," *2013 IEEE International Reliability Physics Symposium (IRPS)*, Monterey, CA, USA, 2013, pp. 3A.3.1-3A.3.5, doi: 10.1109/IRPS.2013.6531973.

APPENDIX

A.1 integrate.c code

```
/*
 * File:   integrate.c
 * Author: Sarah Colebaugh
 *
 * Created on November 23, 2020, 7:12 PM
 */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define pretime 1E-12
#define C2p72_GCTIME1     2.72/TRAPGCTIME1
#define C2p72_GCTIME1_dlt 2.72/TRAPGCTIME1*1e-3
#define C2p72_ESTIME1     2.72/TRAPESTIME1
#define C2p72_ESTIME1_dlt 2.72/TRAPESTIME1*1e-3
#define IsubT_min         2.72
#define IsubpW_min        1e-6

#define TRAPGCTIME1       10
#define TRAPESTIME1       10
#define TRAPGCTIME2       1E9
#define TRAPESTIME2       100
#define TRAPES1MAX        0.5
#define MKS_GC1MAX        5E19 / 1E-6  // TRAPGC1MAX / `C_cm2m_p3 ;
#define D_EXP_THRESHOLD   500.0                //exp(710)=inf
#define D_MAX_EXP         1.403592217853e+217  //exp(500.0)
#define C_UNIT_EXP_ARG    60.0                 //exp(60.0) may touch
the Verilog-A limit
#define C_UNIT_EXP_RES    1.14200738981568e+26
#define C_cm2m_p3   1E-6

#define cn_nc51      0.707106781186548     // sqrt(2)/2 */
#define cn_nc52     -0.117851130197758     // -sqrt(2)/12 */
#define cn_nc53      0.0178800506338833    // (187 - 112*sqrt(2))/1600
*/
#define cn_nc54     -0.00163730162779191   // (-131 + 88*sqrt(2))/4000
*/
```

```
38 #define cn_nc55      6.36964918866352e-5   // (1509-
   1040*sqrt(2))/600000 */
39 #define cn_im53      2.9693154855770998e-1
40 #define cn_im54     -7.0536542840097616e-2
41 #define cn_im55      6.1152888951331797e-3
42
43 void  Fn_calcGC1ES1(double gc1e1deg[], double isub, double DEGTIME);
44 double Fn_Sigmoid(double y, double x);
45 double Fn_DExp(double y, double x, double dx);
46 double Fn_SLtemp(double y, double x, double xmin, double delta);
47 double Fn_SU(double y, double x, double xmax, double delta, double dx);
48 double lexp(double x);
49 double Fn_SUtemp(double y , double x , double xmax , double delta );
50 double Fn_SZtemp(double y , double x , double delta );
51 double Fn_Sgn(double x);
52 double Fn_SZ(double y , double x , double delta , double dx );
53 double Fn_Max(double x, double y);
54 double Fn_Min(double x, double y);
55
56 void strip(char* head);
57
58 int main( ) {
59 FILE *fptr;
60 FILE *fp;
61
62 double dvalue;
63 double TRAPA;
64 double TRAPGC1MAX;
65 double TYPE, DVTBTI, vone, Vgpsp, Vdpsp, Isuba, abstime, DEGTIME,
   CODEG, CODEGSTEP;
66 double Vbs, Vgp, Vfb, nt0s, TTEMP;
67 double Vds, Vgpz, Vbsz, ntls;
68 double idtag1p, idtag1n, idtag3p, idtag3n, idtag4p, idtag4n, dVfb,
   vgc1, ve1, ps0, vtraplx;
69 double prevtime, previdtag1p, previdtag1n, previdtag3p, previdtag3n,
   previdtag4p, previdtag4n;
70 char *param, *value;
71 int i;
72
73 char RefDes[256];
74 char line[256];
75 int retval;
76
77 //removing old data from txt files
```

```
 78  fptr = fopen("idtag1c.txt","w"); fclose(fptr);
 79  fptr = fopen("idtag3c.txt","w"); fclose(fptr);
 80  fptr = fopen("idtag4c.txt","w"); fclose(fptr);
 81  fptr = fopen("dVfbc.txt","w"); fclose(fptr);
 82  fptr = fopen("vgc1c.txt","w"); fclose(fptr);
 83  fptr = fopen("ve1c.txt","w"); fclose(fptr);
 84  fptr = fopen("ps0c.txt","w"); fclose(fptr);
 85  fptr = fopen("vtraplxc.txt","w"); fclose(fptr);
 86
 87  fp = fopen("params.txt","r");
 88  if(!fp) printf("file not accessed\n");
 89  if(fp == NULL) perror("fopen()");
 90
 91  //storing parameter values in doubles
 92  while (fgets(line, sizeof(line), fp) != NULL){
 93      if(strlen(line)==0) continue;
 94      strip(line);
 95      param = strtok(line, "=");
 96      value = strtok(NULL, "=");
 97      retval = sscanf(value, "%lf", &dvalue);
 98      if (!(strcmp(param, "RefDes "))) strcpy(RefDes, value);
 99      if (!strcmp(param, "CODEG ")) CODEG=dvalue;
100      if (!strcmp(param, "CODEGSTEP ")) CODEGSTEP=dvalue;
101      if (!strcmp(param, "TYPE ")) TYPE=dvalue;
102      if (!(strcmp(param, "DVTBTI "))) DVTBTI=dvalue;
103      if (!(strcmp(param, "vone "))) vone=dvalue;
104      if (!(strcmp(param, "Vgpsp "))) Vgpsp=dvalue;
105      if (!(strcmp(param, "Vdpsp "))) Vdpsp=dvalue;
106      if (!(strcmp(param, "Isuba "))) Isuba=dvalue;
107      if (!(strcmp(param, "abstime "))) abstime=dvalue;
108      if (!(strcmp(param, "DEGTIME "))) DEGTIME = dvalue;
109      if (!strcmp(param, "NTRAP0 ")) nt0s=dvalue;
110      if (!strcmp(param, "temp ")) TTEMP=dvalue;
111      if (!strcmp(param, "Vgp ")) Vgp=dvalue;
112      if (!strcmp(param, "Vbs ")) Vbs=dvalue;
113      if (!strcmp(param, "Vfb ")) Vfb=dvalue;
114      if (!strcmp(param, "Vds ")) Vds=dvalue;
115      if (!strcmp(param, "Vgpz ")) Vgpz=dvalue;
116      if (!strcmp(param, "Vbsz ")) Vbsz=dvalue;
117      if (!strcmp(param, "NTRAPL ")) ntls=dvalue;
118     if(!(strcmp(param, "NEXT "))){
119  if(!(strcmp(RefDes, "T0.fet")) || !(strcmp(RefDes, "T9.fet"))){
    //selecting which transistors to examine
```

```
120 //goes through loop for each set of parameters. there is 1 set per
    timestamp per transistor
121
122 if (TYPE < 0){ //for PMOS
123
124 TRAPA = 3E-1;
125 TRAPGC1MAX = 0;
126
127 //setting initial values for integration function
128 if (abstime == 0){
129     fptr = fopen("prev.txt", "w");
130     fprintf(fptr,"prevtime =0");
131     fprintf(fptr,"\n previdtag1p =0");
132     fprintf(fptr,"\n previdtag1n =0");
133     fprintf(fptr,"\n previdtag3p =0");
134     fprintf(fptr,"\n previdtag3n =0");
135     fprintf(fptr,"\n previdtag4p =0");
136     fprintf(fptr,"\n previdtag4n =0");
137     fclose(fptr);
138 }
139
140 //reading previous integration values
141 fptr = fopen("prev.txt", "r");
142 while (fgets(line, sizeof(line), fptr) != NULL){
143     if(strlen(line)==0) continue;
144     strip(line);
145     param = strtok(line, "=");
146     value = strtok(NULL, "=");
147     if (value!=NULL){
148         if (!strcmp(param, "prevtime ")) sscanf(value, "%lf",
    &prevtime);
149         if (!strcmp(param, "previdtag1p ")) sscanf(value, "%lf",
    &previdtag1p);
150         if (!strcmp(param, "previdtag1n ")) sscanf(value, "%lf",
    &previdtag1n);
151         if (!strcmp(param, "previdtag3p ")) sscanf(value, "%lf",
    &previdtag3p);
152         if (!strcmp(param, "previdtag3n ")) sscanf(value, "%lf",
    &previdtag3n);
153         if (!strcmp(param, "previdtag4p ")) sscanf(value, "%lf",
    &previdtag4p);
154         if (!strcmp(param, "previdtag4n ")) sscanf(value, "%lf",
    &previdtag4n);
155     }
```

```
156 }
157 fclose(fptr);
158
159
160 } else { //for NMOS
161     TRAPA = 0;
162 TRAPGC1MAX = 5E19;
163 }
164
165 double step = abstime-prevtime;
166
167 double dVth_bti = DVTBTI;
168 double Vth = vone;
169
170 double ivds = TYPE * Vdpsp;
171 double Vgs = TYPE * Vgpsp;
172 if (ivds >= 0) Vgs = Vgs; //normal mode
173 else Vgs = Vgs - ivds; //reverse mode
174
175 double IsubA = Isuba;
176 double gc1;
177 double e1;
178 double Isub_trap;
179 double Ps0_prv ;
180 double Vtraplx_prv;
181
182 double T0, T2, T3;
183
184 //assumes ifdef AGING is true
185 if (TYPE < 0){ //TYPE = -1 for PMOS and TYPE = 1 for NMOS.
    // Using IfElse structure so that data is not accumulated for an
186 integration variable that does not belong to the transistor being
    evaluated.
    //Have an inverter for this simulation so only separating by transistor
187 type is sufficient.
188      T0 =  TYPE * Isuba ;
189      T0 = ( T0 < 0 || pretime > abstime ) ? 0.0 : T0 ;
190      idtag1p += T0*step; //idt( T0 , 0 );
191      T2 = ( Vgs > Vth  && TRAPA > 0 && pretime < abstime ) ? 1.0 : 0.0
    ;
192      T3 = ( Vgs > Vth  && TRAPA > 0 && pretime < abstime ) ? Vgs : 0.0
    ;
193      if ( TRAPA > 0 && CODEG == 1 && CODEGSTEP == 1 ) {  // BTI model
194        idtag3p += T2*step; //idt( T2 , 0 ); // On time
```

```
195         idtag4p += T3*step; //idt( T3 , 0 ); // Vgp
196       } else {
197         idtag3p += 0 ;
198         idtag4p += 0 ;
199       }
200   } else {
201       T0 =  TYPE * Isuba ;
202       T0 = ( T0 < 0 || pretime > abstime ) ? 0.0 : T0 ;
203       idtag1n += T0*step; //idt( T0 , 0 );
204       T2 = ( Vgs > Vth  && TRAPA > 0 && pretime < abstime ) ? 1.0 : 0.0
    ;
205       T3 = ( Vgs > Vth  && TRAPA > 0 && pretime < abstime ) ? Vgs : 0.0
    ;
206       if ( TRAPA > 0 && CODEG == 1 && CODEGSTEP == 1 ) {  // BTI model
207         idtag3n += T2*step; //idt( T2 , 0 ); // On time
208         idtag4n += T3*step; //idt( T3 , 0 ); // Vgp
209       } else {
210         idtag3n += 0 ;
211         idtag4n += 0 ;
212       }
213   }
214
215   //saving integration values for next loop
216       fptr = fopen("prev.txt", "w");
217       fprintf(fptr,"prevtime = %e", abstime);
218       fprintf(fptr,"\n previdtag1p = %e", idtag1p);
219       fprintf(fptr,"\n previdtag1n = %e", idtag1n);
220       fprintf(fptr,"\n previdtag3p = %e", idtag3p);
221       fprintf(fptr,"\n previdtag3n = %e", idtag3n);
222       fprintf(fptr,"\n previdtag4p = %e", idtag4p);
223       fprintf(fptr,"\n previdtag4n = %e", idtag4n);
224       fclose(fptr);
225
226       if ( CODEG == 1 && TRAPA > 0 ) {          // for BTI model
227         dVfb    = dVth_bti ;
228       } else {
229         dVfb    = 0 ;
230       }
231       if( CODEG == 1 && TRAPGC1MAX > 0 ) {      // for HC model
232   double gc1e1deg[2];
233   Isub_trap = idtag1n / ((abstime-pretime)+1E-50);
234   Fn_calcGC1ES1(gc1e1deg,Isub_trap,DEGTIME);
235   gc1 = gc1e1deg[0];
```

```
236 e1 = gc1e1deg[1];
237         vgc1    = gc1/1e24;
238         ve1     = e1;
239       } else {
240         vgc1    = 0 ;
241         ve1     = 0 ;
242       }
243
244       if( TRAPGC1MAX > 0 && CODEGSTEP == 0 ) { // for HC model
245
246 double beta  = 1.6021918e-19 / ((1.3806226e-23) * TTEMP);
247 double beta2 = beta*beta;
248 double beta_inv = 1.0 / beta ;
249 double T0, T1, T2, T3, T4, T5, T6, T9, T10, TX, TY, Ps0, dPs0;
250
251         //* Initial guess for Ps0.
252         //* Ps0_iniA: solution of subthreshold equation assuming zone-
D1/D2.
253         double EF_NSUBC  = 5E17  / C_cm2m_p3 ;
254         EF_NSUBC = Fn_SLtemp( EF_NSUBC , EF_NSUBC , (1e15 / C_cm2m_p3),
(0.01 / C_cm2m_p3)) ;
255
256         T2 = 1.0e0 ;// + ( NSUBCW / pow ( WG, NSUBCWP )) ; NSUBCW=0;
257         double MKS_NSUBCMAX  = (5E18) / C_cm2m_p3 ;
258         T3 = MKS_NSUBCMAX / EF_NSUBC ;
259         T1 = Fn_SUtemp( T1 , T2 , T3 , 0.01 ) ;
260         EF_NSUBC = EF_NSUBC * T1 ;
261
262         if( EF_NSUBC <= 0.0 ){
263           EF_NSUBC = 1e15 / C_cm2m_p3 ;
264         }
265
266         double MKS_NPEXT = (5.0E17) / C_cm2m_p3 ;
267   double Npexte = MKS_NPEXT;// * ( 1.0 + NPEXTW / pow( WG, NPEXTWP ) );
NPEXTW=0;
268         Npexte = Fn_SLtemp( Npexte , Npexte , (1e15 / C_cm2m_p3), (0.01
/ C_cm2m_p3))  ;
269
270 double Nsub = EF_NSUBC; //LP=0 --> (EF_NSUBC * (Lgate - LP) +  Nsubps
* LP) / Lgate
271         double Lgate = 200E-9  ; //L
272         T3 = 0.5e0 * Lgate;
273         T3 = Fn_SZtemp( T3 , T3 , (1e-8 / (1.0e2))) ;
274         T1 = 1.0E-50;// `Fn_Max(0.0e0, (1.0E-50) ) ;
```

```
275         T3 = T3 * T1 / ( T3 + T1 ) ;
276         Nsub = Nsub + T3 * (Npexte - EF_NSUBC) / Lgate ;
277
278 double cnst0 = sqrt ( 2.0 * (1.034943e-10) * (1.6021918e-19) * Nsub /
    beta ) ;
279 double c_eox = (8.8541878e-12) * 3.9 ; //`C_VAC * KAPPA
280 double Cox0 = c_eox / (3E-9) ;
281 double Cox_inv = 1.0 / Cox0 ;
282 T0 = cnst0 * cnst0 * Cox_inv ;
283 double cnstCoxi = T0 * Cox_inv ;
284
285 double fac1 = cnst0 * Cox_inv ;
286 double fac1p2 = fac1 * fac1 ;
287
288 double Tratio = TTEMP / (27.0 + 273.15) ;
289 double Nin = (1.04e+16) * pow(Tratio,(1.5e0));
290 T1 = Nin / Nsub ;
291 double cnst1 = T1 * T1 ;
292
293          TX = 1.0e0 + 4.0e0 * ( beta * ( Vgp - Vbs ) - 1.0e0 ) / (
    fac1p2 * beta2 ) ;
294 double epsm10 = 10.0e0 * 2.2204460492503131e-16 ;
295          TX = Fn_Max( TX , epsm10 ) ;
296          double Ps0_iniA = Vgp + fac1p2 * beta * 0.5 * ( 1.0e0 - sqrt(
    TX ) ) ;
297
298 double flg_pprv, Chi, Ps0_ini;
299
300              // use analytical value in subthreshold region. //
301              if( Vgs <( Vfb + Vth ) * 0.5 ) {
302                flg_pprv = 0 ;
303              }
304
305            if( flg_pprv == 0 ) {
306
307                //* Analytical initial guess.
308                //* Common part.
309
310                Chi = beta * ( Ps0_iniA - Vbs ) ;
311                if( Chi < 3.0) {
312                  //-----------------------------------*
313                  //* zone-D1/D2
314                  //* - Ps0_ini is the analytical solution of Qs=Qb0
    with
```

```
315                 //*   Qb0 being approximated to 3-degree polynomial.
316                 //*-----------------//
317                 TY = beta * ( Vgp - Vbs ) ;
318 double cn_nc3 = sqrt(2) / 108e0;
319                 T1 = 1.0e0 / ( cn_nc3 * beta * fac1 ) ;
320                 T2 = 81.0 + 3.0 * T1 ;
321                 T3 = -2916.0 - 81.0 * T1 + 27.0 * T1 * TY ;
322                 T4 = 1458.0 - 81.0 * ( 54.0 + T1 ) + 27.0 * T1 * TY ;
323                 T4 = T4 * T4 ;
324                 T5 = pow(( T3 + sqrt( 4 * T2 * T2 * T2 + T4) ), (1/3))
;
325                 TX = 3.0 - ( (1.259921049894873e+00) * T2 ) / ( 3.0 *
T5 ) + 1 / ( 3.0 * (1.259921049894873e+00) ) * T5 ;
326                 Ps0_iniA = TX * beta_inv + Vbs ;
327                 Ps0_ini = Ps0_iniA ;
328               } else if( Vgs <= Vth ) {
329                 //-----------------------------------*
330                 //* Weak inversion zone.
331                 //*-----------------//
332                 Ps0_ini = Ps0_iniA ;
333               } else {
334                 //-----------------------------------*
335                 //* Strong inversion zone.
336                 //* - Ps0_iniB : upper bound.
337                 //*-----------------//
338                 T1 = 1.0 / cnst1 / cnstCoxi ;
339                 T2 = T1 * Vgp * Vgp ;
340                 T3 = beta + 2.0 / Vgp ;
341                 double Ps0_iniB = log ( T2 + (1.0e-50)) / T3 ;
342                 Ps0_ini = Fn_SU( Ps0_ini , Ps0_iniA, Ps0_iniB, (8.0e-
4), T1) ;
343               }
344             }
345             TX = Vbs + (5.0e-13) / 2 ;
346             if ( Ps0_ini < TX ) Ps0_ini = TX ;
347
348       Ps0 = Ps0_ini ;
349             double Psl_lim = Ps0_iniA ;
350
351     double ftr0_qs, ftr0_qs_dPs0, ftrl_qs, ftrl_qs_dPsl;
352             ftr0_qs = 0.0 ; ftr0_qs_dPs0 = 0.0 ;
353             ftrl_qs = 0.0 ; ftrl_qs_dPsl = 0.0 ;
354
```

```
355
356             //-------------------------------------------------*
357             //* Calculation of Ps0. (beginning of Newton loop)
358             //* - Fs0 : Fs0 = 0 is the equation to be solved.
359             //* - dPs0 : correction value.
360             //*-----------------//
361             double exp_bVbs = exp( beta * Vbs ) ;
362             double cfs1 = cnst1 * exp_bVbs ;
363             double flg_conv = 0 ; double lp_s0;
364             for ( lp_s0 = 1 ; lp_s0 <= 40 + 1 ; lp_s0 = lp_s0 + 1 ) {
365
366
367               if( nt0s > 0.0 ) {
368                 T1 = ( Ps0 - Vbs ) * 0.5 ;
369                 T2 = beta / Nsub * nt0s ;
370                 ftr0_qs      = T2 * T1 ;
371                 ftr0_qs_dPs0 = T2 * 0.5 ;
372                 if( ftr0_qs < 0.0 ) {
373                     ftr0_qs = 0.0 ; ftr0_qs_dPs0 = 0.0 ;
374                 }
375               }
376
377
378
379 double fs01, fs01_dPs0, fs02, fs02_dPs0;
380               Chi = beta * ( Ps0 - Vbs ) ;
381               if( Chi < 5 ) {
382                 //-----------------------------------------*
383                 //* zone-D1/D2.  (Ps0)
384                 //* - Qb0 is approximated to 5-degree polynomial.
385                 //*-----------------//
386                 double fi = Chi * Chi * Chi * ( cn_im53 + Chi * (
cn_im54 + Chi * cn_im55 ) ) ;
387                 double fi_dChi = Chi * Chi * ( 3 * cn_im53 + Chi * ( 4
* cn_im54 + Chi * 5 * cn_im55 ) ) ;
388                 fs01 = cfs1 * fi * fi ;
389                 fs01_dPs0 = cfs1 * beta * 2 * fi * fi_dChi ;
390                 double fb = Chi * ( cn_nc51 + Chi * ( cn_nc52 + Chi *
( cn_nc53 + Chi * ( cn_nc54 + Chi * cn_nc55 ) ) ) ) ;
391                 double fb_dChi = cn_nc51 + Chi * ( 2 * cn_nc52 + Chi *
( 3 * cn_nc53 + Chi * ( 4 * cn_nc54 + Chi * 5 * cn_nc55 ) ) ) ;
392                 fs02 = sqrt( fb * fb + fs01 + ftr0_qs ) + (1.0e-50) ;
393                 fs02_dPs0 = ( beta * fb_dChi * 2 * fb + fs01_dPs0 +
ftr0_qs_dPs0 ) / ( fs02 + fs02 ) ;
```

```
394
395                 } else {
396                   //-----------------------------------------*
397                   //* zone-D3.  (Ps0)
398                   //*-----------------//
399                   if( Chi < 60 ) { // avoid exp_Chi to become extremely large //
400                     double exp_Chi = exp( Chi ) ;
401                     fs01 = cfs1 * ( exp_Chi - 1.0e0 ) ;
402                     fs01_dPs0 = cfs1 * beta * ( exp_Chi ) ;
403                   } else {
404                     double exp_bPs0 = exp( beta*Ps0 ) ;
405                     fs01 = cnst1 * ( exp_bPs0 - exp_bVbs ) ;
406                     fs01_dPs0 = cnst1 * beta * exp_bPs0 ;
407                   }
408
409                   fs02 = sqrt( Chi - 1.0 + fs01 + ftr0_qs ) ;
410                   fs02_dPs0 = ( beta + fs01_dPs0  + ftr0_qs_dPs0 ) / ( fs02 + fs02 ) ;
411
412                 } // end of if( Chi ... ) else block //
413                 double Fs0 = Vgp - Ps0 - fac1 * fs02 ;
414                 double Fs0_dPs0 = - 1.0e0 - fac1 * fs02_dPs0 ;
415                 if( flg_conv == 0 ) {// break
416                   dPs0 = - Fs0 / Fs0_dPs0 ;
417
418                   //-----------------------------------------*
419                   //* Update Ps0 .
420                   //* - clamped to Vbs if Ps0 < Vbs .
421                   //*-----------------//
422  double maxHolder = Fn_Max(1.0e0,abs(Ps0));
423                   double dPlim = 0.5*0.1*(1.0 + maxHolder) ;
424  double sign = Fn_Sgn( dPs0 );
425                   if ( abs( dPs0 ) > dPlim ) dPs0 = dPlim * sign ;
426                   Ps0 = Ps0 + dPs0 ;
427                   TX = Vbs + (5.0e-13) / 2 ;
428                   if ( Ps0 < TX ) Ps0 = TX ;
429                 } // flg_conv == 0
430               } // end of Ps0 Newton loop //
431
432 Ps0_prv = Ps0;
433
434 // Vdseff(begin) //
```

```
435   double LG = Lgate * (1.0e6)  ;
436          T1 = 10 * LG ;
437          double DDLTe = T1 * 10 / ( T1 + 10 ) + 0 + 1.0e-50 ;
438          double Toxe     = 3E-9 ; //TOX
439          double Cox      = c_eox / Toxe ;
440
441            double Vdsorg = Vds ;
442            double qnsub_esi = Nsub * (1.6021918e-19) * (1.034943e-10) ;
443            T2 = qnsub_esi / ( Cox * Cox ) ;
444            T5 = Vgpz - beta_inv - Vbsz ;
445            T1 = 1.0e0 + 2.0 / T2 * T5 ;
446            T9 = Fn_SZ( T9 , T1 , 0.05 , T9 );
447            T9 = T9 + 1.0e-50  ;
448            T3 = sqrt( T9 ) ;
449            T10 = Vgpz + T2 * ( 1.0e0 - T3 ) ;
450            T10 = Fn_SZ( T10 , T10 , 0.01 , T0 );
451            T10 = T10 + epsm10 ;
452            T1 = Vds / T10 ;
453            T2 = pow( T1 , DDLTe - 1.0 ) ;
454            T3 = 1.0 + T2 * T1 ;
455            T4 = pow( T3 , 1.0 / DDLTe - 1.0 ) ;
456            T6 = T4 * T3 ;
457            double Vdseff = Vds / T6 ;
458            Vds = Vdseff ;
459            // Vdseff(end) //
460
461 double exp_bVbsVds = exp( beta * ( Vbs - Vds ) ) ;
462 double Pds, Psl;
463 int start_of_loopl = 0, NNN = 0;
464
465            //-----------------------------------------------------*
466            //* Skip Psl calculation when Vds is very Small.
467            //*-----------------//
468            if( Vds <  0.0 ) {
469              Pds = 0.0 ;
470              Psl = Ps0 ;
471              start_of_loopl = 1 ;
472            }
473         if(start_of_loopl == 0) {
474
475            //-------------------------------------------------------------
   -*
476            //* Initial guess for Pds( = Psl - Ps0 ).
```

```
477             //*-----------------//
478         double Pds_ini, Pds_max;
479             if( flg_pprv == 0 ) {
480               //-------------------------------------------------*
481               //* Analytical initial guess.
482               //*-----------------//
483               Pds_max = Fn_Max( Psl_lim - Ps0 , 0.0e0 ) ;
484               Pds_ini = Fn_SU( Pds_ini , Vds, (1.0e0 + 0.3) * Pds_max,
3.0e-2, T1 ) ;
485               Pds_ini = Fn_Min( Pds_ini , Pds_max ) ;
486             }
487             if ( Pds_ini < 0.0 ) Pds_ini = 0.0 ;
488             else if ( Pds_ini > Vds ) Pds_ini = Vds ;
489
490             //-------------------------------------------------*
491             //* Assign initial guess.
492             //*-----------------//
493             if( NNN == 0 ) {
494               Pds = Pds_ini ;
495               Psl = Ps0 + Pds ;
496             } else {
497               // take solution from previous PS0_SCE_loop as initial
value //
498               Pds = Psl - Ps0 ;
499             }
500             TX = Vbs + (5.0e-13) / 2 ;
501             if ( Psl < TX ) Psl = TX ;
502
503             //-------------------------------------------------*
504             //* Calculation of Psl by solving Poisson eqn.
505             //* (beginning of Newton loop)
506             //* - Fsl : Fsl = 0 is the equation to be solved.
507             //* - dPsl : correction value.
508             //*-----------------//
509             flg_conv = 0 ;
510
511             //-------------------------------------------------*
512             //* start of Psl calculation. (label)
513             //*-----------------//
514           } if(start_of_loopl)  start_of_loopl = 0; // end of
start_of_loopl
515 // start_of_loopl:
516 double lp_sl;
```

```
517             for ( lp_sl = 1 ; lp_sl <= 40 + 1 ; lp_sl = lp_sl + 1 ) {
518
519               if( ntls > 0.0 ) {
520                 T1 = ( Psl - Vbs ) * 0.5 ;
521                 T2 = beta / Nsub * ntls ;
522                 ftrl_qs      = T2 * T1 ;
523                 ftrl_qs_dPsl = T2 * 0.5 ;
524                 if( ftrl_qs < 0.0 ) {
525                     ftrl_qs = 0.0 ; ftrl_qs_dPsl = 0.0 ;
526                 }
527               }
528
529
530               Chi = beta * ( Psl - Vbs ) ;
531 double fi, fi_dChi, fsl1, fsl1_dPsl, fb, fb_dChi;
532 double fsl2, fsl2_dPsl, Fsl, Fsl_dPsl, dPsl;
533               if( Chi  < 5 ) {
534                 //-----------------------------------------*
535                 //* zone-D2.  (Psl)
536                 //* - Qb0 is approximated to 5-degree polynomial.
537                 //*-----------------//
538                 fi = Chi * Chi * Chi * ( cn_im53 + Chi * ( cn_im54 + Chi
 * cn_im55 ) ) ;
539                 fi_dChi = Chi * Chi * ( 3 * cn_im53 + Chi * ( 4 *
 cn_im54 + Chi * 5 * cn_im55 ) ) ;
540                 cfs1 = cnst1 * exp_bVbsVds ;
541                 fsl1 = cfs1 * fi * fi ;
542                 fsl1_dPsl = cfs1 * beta * 2 * fi * fi_dChi ;
543                 fb = Chi * ( cn_nc51 + Chi * ( cn_nc52 + Chi * ( cn_nc53
 + Chi * ( cn_nc54 + Chi * cn_nc55 ) ) ) ) ;
544                 fb_dChi = cn_nc51 + Chi * ( 2 * cn_nc52 + Chi * ( 3 *
 cn_nc53 + Chi * ( 4 * cn_nc54 + Chi * 5 * cn_nc55 ) ) ) ;
545                 fsl2 = sqrt( fb * fb + fsl1 + ftrl_qs ) ;
546                 fsl2_dPsl = ( beta * fb_dChi * 2 * fb + fsl1_dPsl +
 ftrl_qs_dPsl ) / ( fsl2 + fsl2 ) ;
547               } else {
548                 //-----------------------------------------*
549                 //* zone-D3.  (Psl)
550                 //*-----------------//
551                 double Rho = beta * ( Psl - Vds ) ;
552 //              double exp_Rho = exp( Rho ) ;
553                 double exp_Rho = Fn_DExp( exp_Rho , Rho , T0) ;
554                 fsl1 = cnst1 * ( exp_Rho - exp_bVbsVds ) ;
555                 fsl1_dPsl = cnst1 * beta * ( exp_Rho ) ;
```

```
556                 double Xil = Chi - 1.0e0 ;
557
558                 fsl2 = sqrt( Xil + fsl1  + ftrl_qs ) ;
559                 fsl2_dPsl = ( beta + fsl1_dPsl + ftrl_qs_dPsl ) / ( fsl2
+ fsl2 ) ;
560               }
561               Fsl = Vgp - Psl - fac1 * fsl2 ;
562               Fsl_dPsl = - 1.0e0 - fac1 * fsl2_dPsl ;
563               if( flg_conv == 0 ) { // break
564                  dPsl = - Fsl / Fsl_dPsl ;
565
566                 //-----------------------------------------*
567                 //* Update Psl .
568                 //* - clamped to Vbs if Psl < Vbs .
569                 //*-----------------//
570 double maxHolder = Fn_Max(1.0e0,abs(Psl));
571                 double dPlim = 0.5*0.1*(1.0 + maxHolder) ;
572         double sign = Fn_Sgn( dPsl );
573                 if ( abs( dPsl ) > dPlim ) dPsl = dPlim * sign ;
574                 Psl = Psl + dPsl ;
575                 TX = Vbs + (5.0e-13) / 2 ;
576                 if ( Psl < TX ) Psl = TX ;
577
578                 //-----------------------------------------*
579                 //* Check convergence.
580                 //* NOTE: This condition may be too rigid.
581                 //*-----------------//
582                 if( abs( dPsl ) <= (5.0e-13) && abs( Fsl ) <= (1.0e-8) )
{
583                   flg_conv = 1 ;
584                 }
585               } // flg_conv==0
586             } // end of Psl Newton loop //
587
588             // Eliminate loop count to take account of the derivative
loop //
589             lp_sl = lp_sl - 1 ;
590
591             //-----------------------------------------*
592             //* Procedure for diverged case.
593             //*-----------------//
594             if( flg_conv == 0 ) {
595               printf( "*** warning(HiSIM2): %M Went Over Iteration
Maximum(Psl)\n" ) ;
```

```
596             }
597 double Xil, Xilp12, Xilp32;
598             if( Chi < 5 ) {
599 double fb;
600               //-----------------------------------------*
601               //* zone-D1/D2. (Psl)
602               //*-----------------//
603               Xil = fb * fb + epsm10 ;
604               Xilp12 = fb + epsm10 ;
605               Xilp32 = fb * fb * fb + epsm10 ;
606             } else {
607               //-----------------------------------------*
608               //* zone-D3. (Psl)
609               //*-----------------//
610               Xil = Chi - 1.0e0 ;
611               Xilp12 = sqrt( Xil ) ;
612               Xilp32 = Xil * Xilp12 ;
613             }
614
615             //-----------------------------------------------------*
616             //* Assign Pds.
617             //*-----------------//
618             Pds = Psl - Ps0 ;
619             if( Pds < 0.0 ) {
620               Pds = Pds + epsm10 ;
621               Psl = Pds + Ps0 ;
622             }
623
624 Vtraplx_prv = Vdseff + Ps0 - Psl;
625
626         ps0     = Ps0_prv ;
627         vtraplx = Vtraplx_prv ;
628       } else {
629         ps0     = 0 ;
630         vtraplx = 0 ;
631       }
632
633 fptr = fopen("idtag1c.txt","a");
634 fprintf(fptr, "\n %s", RefDes);
635 fprintf(fptr, " %e", abstime);
636 if(TYPE < 0) fprintf(fptr,"%e",idtag1p);
637 else fprintf(fptr, " %e", idtag1n);
638 fprintf(fptr, " %e", T0);
```

```
639  fprintf(fptr, " %e", step);
640  fclose (fptr);
641
642  fptr = fopen("idtag3c.txt","a");
643  fprintf(fptr, "\n %s", RefDes);
644  fprintf(fptr, " %e", abstime);
645  if(TYPE < 0) fprintf(fptr, " %e", idtag3p);
646  else fprintf(fptr, " %e", idtag3n);
647  fprintf(fptr, " %e", T2);
648  fprintf(fptr, " %e", step);
649  fclose (fptr);
650
651  fptr = fopen("idtag4c.txt","a");
652  fprintf(fptr, "\n %s", RefDes);
653  fprintf(fptr, " %e", abstime);
654  if(TYPE < 0) fprintf(fptr, " %e", idtag4p);
655  else fprintf(fptr, " %e", idtag4n);
656  fprintf(fptr, " %lf", T3);
657  fprintf(fptr, " %e", step);
658  fclose (fptr);
659
660  fptr = fopen("dVfbc.txt","a");
661  fprintf(fptr, "\n %s", RefDes);
662  fprintf(fptr, " %e", abstime);
663  fprintf(fptr, " %e", dVfb);
664  fclose (fptr);
665
666  fptr = fopen("vgc1c.txt","a");
667  fprintf(fptr, "\n %s", RefDes);
668  fprintf(fptr, " %e", abstime);
669  fprintf(fptr, " %e", vgc1);
670  fclose (fptr);
671
672  fptr = fopen("ve1c.txt","a");
673  fprintf(fptr, "\n %s", RefDes);
674  fprintf(fptr, " %e", abstime);
675  fprintf(fptr, " %e", ve1);
676  fclose (fptr);
677
678  fptr = fopen("ps0c.txt","a");
679  fprintf(fptr, "\n %s", RefDes);
680  fprintf(fptr, " %e", abstime);
681  fprintf(fptr, " %e", ps0);
```

```
682  fclose (fptr);
683
684  fptr = fopen("vtraplxc.txt","a");
685  fprintf(fptr, "\n %s", RefDes);
686  fprintf(fptr, " %e", abstime);
687  fprintf(fptr, " %e", vtraplx);
688  fclose (fptr);
689
690  }
691  }
692  }
693  fclose(fp);
694  }
695
696  void Fn_calcGC1ES1(double gc1e1deg[], double isub, double DEGTIME) {
697  double Wgate = (2.5E-6) / 1.0 + 0.0; //  W / NF  + XW ;
698  //double MKS_WL = 0; // WL / pow( `C_m2cm , WLN ) ; WL=0
699  double dW = 0; //XWD  +(MKS_WL / pow(Wgate + WLD, WLN)) ; XWD = 0
     MKS_WL = 0
700  double Weff    = Wgate - 2.0 * dW ; //equal to W, may to have this
     brought over in list of parameters
701  double gc1deg0   = (1E15) / (1E-6); //TRAPGC1 / `C_cm2m_p3 ;
702  double e1deg0    = 0.3; //TRAPES1 ;
703  double Isub_fac;
704  double T0, T4, T5, T6;
705  double gc1deg, e1deg;
706
707  double T2 = isub / Weff ;
708  Isub_fac = Fn_Sigmoid( Isub_fac , (T2-IsubpW_min)/IsubpW_min*15 );
709  T2 = Fn_SLtemp( T2 , T2 , C2p72_GCTIME1 , C2p72_GCTIME1_dlt );
710  double T3 = T2  * TRAPGCTIME1 ;
711  T4 = Fn_SLtemp( T4 , (T2 * DEGTIME) , IsubT_min , 1e-3 );
712  T5 = Fn_SLtemp( T5 , (T2 * TRAPGCTIME2) , IsubT_min , 1e-3 );
713  double gc_time  = log(T4) ;
714  double gc_time1 = log(T3) ;
715  double gc_time2 = log(T5) ;
716  double T8 = (gc_time - gc_time2)/gc_time1 ;
717  T5 = - 0.5 * T8 * T8 ;
718  T6 = Fn_DExp( T6 , T5 , T0 );
719  gc1deg = gc1deg0 + MKS_GC1MAX / gc_time1 * T6 * Isub_fac ;
720  gc1e1deg[0] = gc1deg;
721  T2 = isub / Weff ;
722  T2 = Fn_SLtemp( T2 , T2 , C2p72_ESTIME1 , C2p72_ESTIME1_dlt )  ;
```

```
723  T3 = T2  * TRAPESTIME1 ;
724  T5 = Fn_SLtemp( T5 , (T2 * TRAPESTIME2) , IsubT_min , 1e-3 )  ;
725  double e_time1 = log(T3) ;
726  double e_time2 = log(T5) ;
727  T8 = (gc_time - e_time2)/e_time1 ;
728  T5 = -0.5 * T8 * T8 ;
729  T6 = Fn_DExp( T6 , T5 , T0 );
730  e1deg = e1deg0 + TRAPES1MAX / e_time1 * T6 * Isub_fac ;
731  e1deg = Fn_SU( e1deg, e1deg, 1.0, 1e-2, T0 );
732  gc1e1deg[1] = e1deg;
733  }
734
735
736  double Fn_Sigmoid( double y , double x ) {
737  double z = lexp(-(x));
738      y = 1.0 / ( 1.0 + z );
739  return y;
740  }
741
742  double Fn_DExp( double y , double x , double dx) {
743    if ((x) >= D_EXP_THRESHOLD) {
744      y  = D_MAX_EXP * (1 + (x) - D_EXP_THRESHOLD);
745      dx = D_MAX_EXP ;
746    } else {
747      double TMF1 = (x);
748      y = 1.0;
749      while (TMF1 >= C_UNIT_EXP_ARG ) {
750        y = y * C_UNIT_EXP_RES;
751        TMF1 = TMF1 - C_UNIT_EXP_ARG;
752      }
753      y = y * exp(TMF1); dx = y;
754    }
755  return y;
756  }
757
758  double Fn_SLtemp( double y , double x , double xmin , double delta ) {
759      double TMF1 = ( x ) - ( xmin ) - ( delta ) ;
760      double TMF2 = 4.0 * ( xmin ) * ( delta ) ;
761      TMF2 = TMF2 > 0.0 ?  TMF2 : - ( TMF2 ) ;
762      TMF2 = sqrt ( TMF1 * TMF1 + TMF2 ) ;
763      y = ( xmin ) + 0.5 * ( TMF1 + TMF2 ) ;
764  return y;
765    }
```

```
766
767 double Fn_SU( double y , double x , double xmax , double delta , double
    dx ) {
768     double TMF1 = ( xmax ) - ( x ) - ( delta ) ;
769     double TMF2 = 4.0 * ( xmax ) * ( delta) ;
770     TMF2 = TMF2 > 0.0 ?  TMF2 : - ( TMF2 ) ;
771     TMF2 = sqrt ( TMF1 * TMF1 + TMF2 ) ;
772     dx = 0.5 * ( 1.0 + TMF1 / TMF2 ) ;
773     y = ( xmax ) - 0.5 * ( TMF1 + TMF2 ) ;
774 return y;
775  }
776
777 double lexp(double x){
778 double lexp;
779 if(x > 80.0){
780 lexp = 5.540622384e+34 * (1.0 + (x) - 80.0);
781 }
782 else if(x < -80.0){
783 lexp = 1.804851387e-35;
784 }
785 else{
786 lexp = exp(x);
787 }
788 return lexp;
789 }
790
791 double Fn_SUtemp( double y , double x , double xmax , double delta ){
792     double TMF1 = ( xmax ) - ( x ) - ( delta ) ;
793     double TMF2 = 4.0 * ( xmax ) * ( delta) ;
794     TMF2 = TMF2 > 0.0 ?  TMF2 : - ( TMF2 ) ;
795     TMF2 = sqrt ( TMF1 * TMF1 + TMF2 ) ;
796     y = ( xmax ) - 0.5 * ( TMF1 + TMF2 ) ;
797 return y;
798 }
799
800 double Fn_SZtemp(double  y , double x , double delta ) {
801     double TMF1 = sqrt ( ( x ) *  ( x ) + 4.0 * ( delta ) * ( delta) )
    ;
802     y = 0.5 * ( ( x ) + TMF1 ) ;
803 return y;
804 }
805
806 double Fn_Max(double x, double y){
```

```
807   double ans = ( (x) >= (y) ? (x) : (y) );
808   return ans;
809  }
810
811  double Fn_Min(double x, double y){
812   double ans = ( (x) <= (y) ? (x) : (y) );
813   return ans;
814  }
815
816  double Fn_Sgn(double x){
817     double sign = ( (x) >= 0  ?  (1) : (-1) );
818  return sign;
819  }
820
821  double Fn_SZ( double y , double x , double delta , double dx ) {
822      double TMF2 = sqrt ( ( x ) *  ( x ) + 4.0 * ( delta ) * ( delta) )
     ;
823      dx = 0.5 * ( 1.0 + ( x ) / TMF2 ) ;
824      y = 0.5 * ( ( x ) + TMF2 ) ;
825      if( y < 0.0 ) y=0.0;
826  }
827
828  void strip(char *head){
829      char *tail;
830      tail=head;
831      while(*tail!=(char)0){
832          tail++;
833      }
834      while (tail!=head && (*tail)==(char)0){
835          tail--;
836          if(*tail==(char)9) *tail=(char)0;
837          if(*tail==(char)10) *tail=(char)0;
838          if(*tail==(char)13) *tail=(char)0;
839          if(*tail==(char)32) *tail=(char)0;
840      }
841  }
```