Modified Bayesian Approach to Modeling Commercial Parts with TID Response Variability in Analog Systems

by

Michael Brandon Smith

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

May 14, 2021

Nashville, Tennessee

Approved:

Ronald D. Schrimpf, Ph.D.

Arthur F. Witulski, Ph.D.

To God, the Almighty,

in magnification of His creation that we are so blessed to study and inhabit,

and

To His son, Jesus, the Christ,

for His mercy that saves and sustains the children of God

ACKNOWLEDGEMENTS

This work is certainly a product of ample amounts of help from others. Firstly, I would like to thank Dr. Art Witulski for introducing me to the project as a mere undergraduate intern in the summer of 2016. His guidance then and over the years has helped transform this from a summer project into what it is today. I would also like to thank Dr. Ron Schrimpf for his advice and encouragement since starting in his lab in 2018. His leadership and expertise have been of great value to this work and my graduate career.

Secondly, there are several other professors and colleagues that greatly influenced this work. A huge thanks goes to Kaitlyn Ryder for helping me with experimentation. I could not have completed this without her help. I would also like to thank Dr. Sternberg, Dr. Kauppila, and Mike McCurdy for helping me organize my simulations and experiment and Nag for helping me with my understanding of Bayesian statistics. Additionally, I would like to thank the students of the RER group that have helped with the concepts and encouraged me along the way: Rebekah Austin, Drew Tonigan, Landen Ryder, and Rachel Brewer.

Finally, I would like to thank my numerous friends and family for the support they have shown along the way. This would not be possible without you all.

This work was supported by the Jet Propulsion Laboratory and the Defense Threat Reduction Agency.

TABLE OF CONTENTS

PAGE

ACKNOWLEDGEMENTS iii

LIST OF TABLES vi

LIST OF FIGURES vii

LIST OF ABBREVIATIONS/NOMECLATURE/SYMBOLS ix

I. Introduction 2

II. Background 4

A. Total Ionizing Dose Effects 4

B. Design Challenges of System Level TID Effects 5

C. Bayesian Modeling 6

D. Other Modeling Methods 6

III. Component TID Modeling 8

A. TID Induced Parameter Shift 8

B. Building IRF510 Radiation Model 9

C. Discussion 13

IV. Modified Bayesian Modeling of System 16

A. Analog System Design 16

B. System Level Parameters 16

C. Predicting System Level Response 18

V. Experimental Verification 21

A. Pair-Wise Transistor Swapping 21

B. Experimental Setup 22

C. Analysis of Results 24

VI. Conclusions 26

VII. References 27

VIII. Appendix A – Component Radiation Modeling Code 29

IX. Appendix B – Circuit Simulation Code 35

LIST OF TABLES

TABLE PAGE

Table 1 – System parameter summarization 18

Table 2 – Ranked threshold voltage shift of transistors after 10 krad(Si), resulting in assigned letter 23

Table 3 – List of experiment runs, the transistors used, and subsequent system gain 23

LIST OF FIGURES

FIGURE PAGE

Figure 1 – Radiation's effect on oxides in MOS structures, from [3] 4

Figure 2 – Differential amplifier schematic 5

Figure 3 – Expected parameter degradation curve, with mean in blue and red lines representing one standard deviation 9

Figure 4 – Probability distribution function for threshold voltages for increasing functions of dose (right-to-left). Generated distributions from experimental data. 10

Figure 5 – Probability distribution function of transcoductance parameter for increasing functions of dose(left-to-right). Generated distributions from experimental data. 10

Figure 6 – Theshold voltage model as a function of dose; average value (blue), standard deviation (yellow), and raw data (red) 12

Figure 7 – Data distributions (blue) with modeled distributions (red) for threshold voltage 12

Figure 8 – Two groupings of transistors (top group on left, bottom group on right) showing tighter distributions than combined; threshold voltage as function of dose and the composite of these two make the bimodal distribution in Figure 9. 13

Figure 9 – Overlay of two plots of Figure 8; shows the wide standard deviation of Figure 6 14

Figure 10 - Correlation between threshold voltage (VT) and transconductance parameter (K) for all data points taken during TID modeling. No significant correlation between the parameters is shown, so sampling from the model can be randomly sampled. 15

Figure 11 – Differential amplifier with current mirror source resistance 16

Figure 12 – V_{out} and derivative (gain), with definitions of output parameters 18

Figure 13 – Simulation results of system parameters, plotted in 3D. Both plots are same data set, just rotated by 90 degrees. 19

Figure 14 – Max Gain vs. Center Voltage for pre-rad (green crosses) and 10krad (blue dots) from SPICE simulation of differential amplifier. 20

Figure 15 – Circuit board used to perform system level tests..... 22

Figure 16 – Simulations (blue) with data overlayed in red. Data sets are identical, just rotated by 90 degrees. 25

LIST OF ABBREVIATIONS/NOMECLATURE/SYMBOLS

COTS – Commercial Off the Shelf

FWHM – Full Width Half Max

MCMC - Markov Chain Monte Carlo

MOSFET – Metal-Oxide-Semiconductor Field Effect Transistor

PCB – Printed Circuit Board

TID – Total Ionizing Dose

I. INTRODUCTION

When constructing a system suitable for radiation environments, it is important to predict the system level response to said radiation with a fair degree of accuracy. Much research has been done in the radiation effects community to refine this process and produce systems that are not only functional, but also reliable. This is typically done by using specified manufacturing processes, as well as redundant systems and specialized circuit design. However effective, specialized processes and additional parts are expensive, and, with access to commercial off the shelf (COTS) parts being widely available, cheaper and less process-controlled parts are being used.

In a system that uses COTS parts, the system response to radiation is much harder to predict than systems with radiation-hardened parts, given that COTS parts are much less controlled in their manufacturing: device parameters and material qualities can change from device-to-device and lot-to-lot. This presents a problem in modeling such a system, since even pre-radiation parameters can have some variability. As such a system is exposed to a level of total ionizing dose (TID) radiation, these parameters can increase in variability, ultimately allowing the system to have a wide range of potential outputs after being irradiated.

To demonstrate this effect, this document will showcase the IRF510 transistor in a simple analog circuit: a differential amplifier. These parts and circuit were chosen because they provide a simple platform for exploring the proposed methodology. A modified Bayesian approach is used to model the discrete device behavior as well as the system as a whole. In Chapter 2, relevant background information will be discussed. Chapter 3 will focus on modeling the TID response of an individual COTS part, incorporating said model into a simple differential amplifier system in

Chapter 4. Chapter 5 includes an experimental verification and compares the data to the simulations in Chapter 4. Finally, Chapter 6 discusses the conclusions from this work.

II. BACKGROUND

A. Total Ionizing Dose Effects

Radiation produces electron-hole pairs (ehp) in materials when the energy of such particles or photons exceeds a certain ionizing threshold [1] [2]. In metal-oxide-semiconductor field effect transistors (MOSFETs), this additional charge created in the oxide regions of the device can cause device degradation due to trapped charge at oxide-semiconductor interfaces, as seen in Figure 1 [3]. This trapped charge can create leakage paths in the device structure, as well as modify the needed charge on the gate to turn on and off the device.

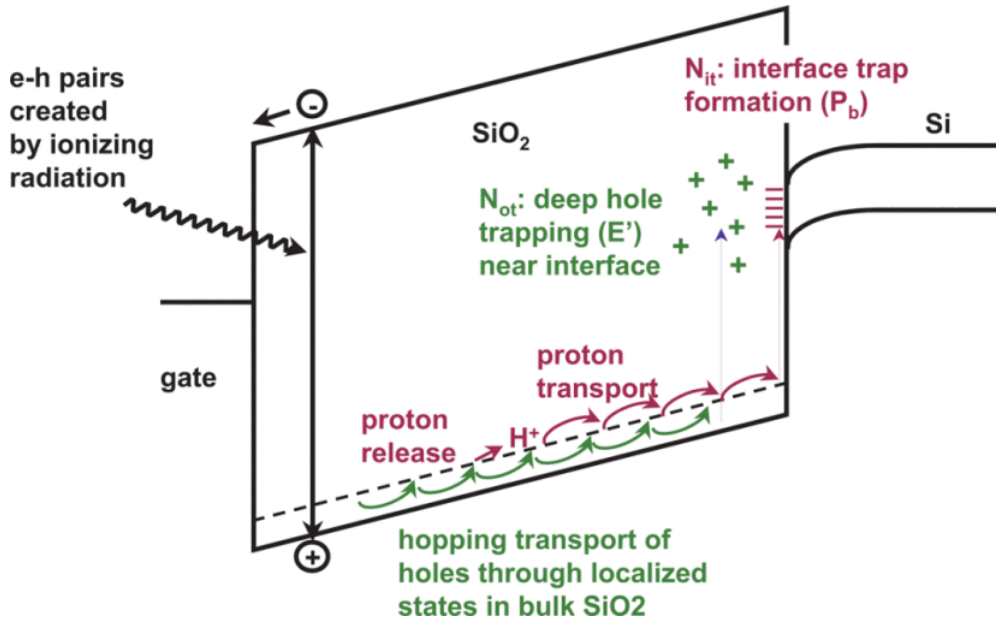

Figure 1 – Radiation's effect on oxides in MOS structures, from [3]

This device degradation leads to systems that were designed to operate in a specific manner to alter in functionality, whether switching to a different mode of operation or completely failing after a certain amount of ionizing radiation. This can prove catastrophic for larger electrical systems, and much of the research in radiation effects has been done to mitigate such effects.

B. Design Challenges of System Level TID Effects

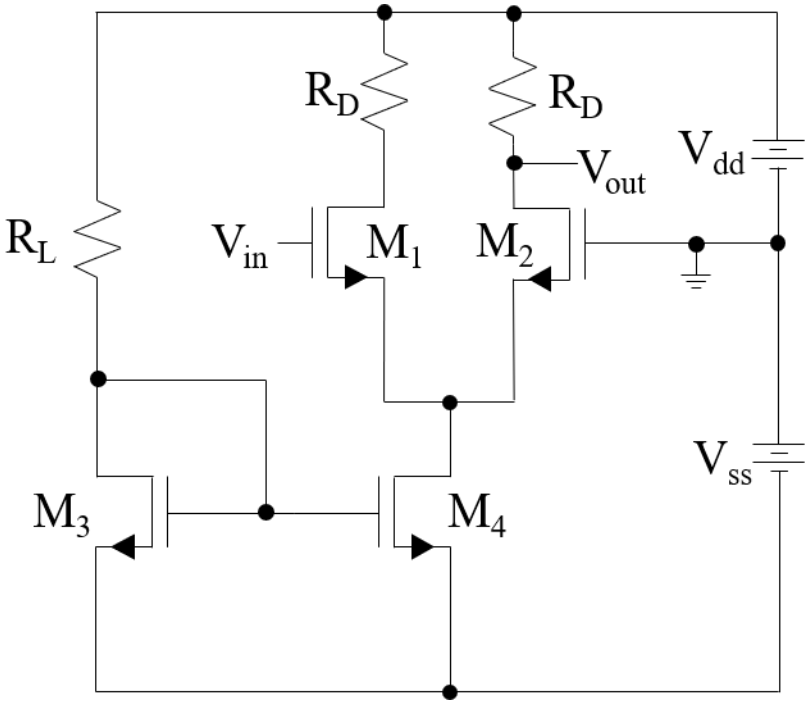

Figure 2 – Differential amplifier schematic

Given that MOSFETs degrade when they are irradiated, building a circuit such as that in Figure 2 suitable for high-dose environments proves to be a challenge. As shown in [4], the threshold matching of transistors M_1 and M_2 in the diff amp must be tracked as the devices degrade in order to maintain circuit functionality. This is because the gain function of the circuit depends on the difference in threshold voltage of M_1 and M_2. Additionally, it is important that the bias transistors of M_3 and M_4 also match in threshold voltage. These transistors set the bias current for the differential pair, and mismatch could lead to M_4 coming out of saturation.

The example given in this work is meant to show how one might mitigate radiation concerns when one does not have control over the manufacturing process of parts or cost of the project. As Ray Ladbury put it in [5], “system-level hardening provides one last line of defense

whereby a critical part with marginal radiation performance may still meet its system requirements." In other words, the methods in this work are not meant to be the first line of solutions to a system susceptible to radiation, but rather a process to determine what steps can be taken at a system level to mitigate the effects of radiation, particularly TID.

C. Bayesian Modeling

Techniques used to model parameter shifts in devices and systems have been researched for quite some time [6]. One tool that is common for modeling the variability in parameter shifts is Bayesian modeling [7], which is particularly useful for COTS parts due to their part-to-part and lot-to-lot variability [8] [9] [10] [11] [12] [13]. In a nutshell, this process involves taking a predicted distribution of outcomes and modifying that distribution with the discovery of new information. Research has been done to model systems using a Bayesian technique [14], but these models often rely on software such as PyMC [15]. This Python library builds a posterior distribution from a particular data set and allows the user to sample from said distribution using Markov Chain Monte Carlo (MCMC) techniques. The drawback to using a tool such as PyMC is user familiarity. This thesis provides a modified approach to simplify the technique in [4] by using commonplace statistical models that can be implemented in a coding language of choice.

D. Other Modeling Methods

As mentioned before, many methods have been derived to predict system performance at various confidence levels. One in particular from MIL-HDBK-814 utilizes a "K_{TL} factor" approach, where a factor K_{TL} is chosen based off the confidence level and sample size of the data set [16]. This factor is used to calculate a probability of survivability with a confidence level, with

the probability having an exponential relationship to K_{TL}. While this approach does not factor in confidence levels, utilizing such techniques would add to the analysis of any system.

III. COMPONENT TID MODELING

A. TID Induced Parameter Shift

For the purposes of this paper, the IRF510 power MOSFET will be modeled for use in a system. To build a radiation model for a discrete part, a population of parts must be irradiated to various levels, and the parameters of interest must be measured at each dose level. This is done to determine the degradation behavior of parameters as a function of dose. The two parameters that are modeled are threshold voltage V_T and the transconductance parameter K, as given in the 2-parameter MOSFET equations (Eqn. 1) [17].

$$I_D = \begin{cases} K\left((V_{GS} - V_T)V_{DS} - \frac{{V_{DS}}^2}{2}\right), V_{DS} < V_{GS} - V_T \\ \frac{K}{2}(V_{GS} - V_T)^2, V_{DS} \geq V_{GS} - V_T \end{cases} \quad (1)$$

This model was chosen due to its simplicity, and, since the IRF510 is a MOSFET, it should do a fair job of modeling its behavior.

As a MOSFETs is exposed to radiation, it is expected that the threshold voltage and transconductance parameter change as functions of dose. Due to manufacturing variations, this parameter degradation does not happen at the same rate in all devices. Instead, as the TID increases, the parameter of choice (such as the threshold voltage) is expected to spread out in a manner such as that in Figure 3. The blue line represents the average threshold voltage, while the dashed red lines represent one standard deviation on each side of the average. This sort of behavior is what makes COTS parts unpredictable in systems.

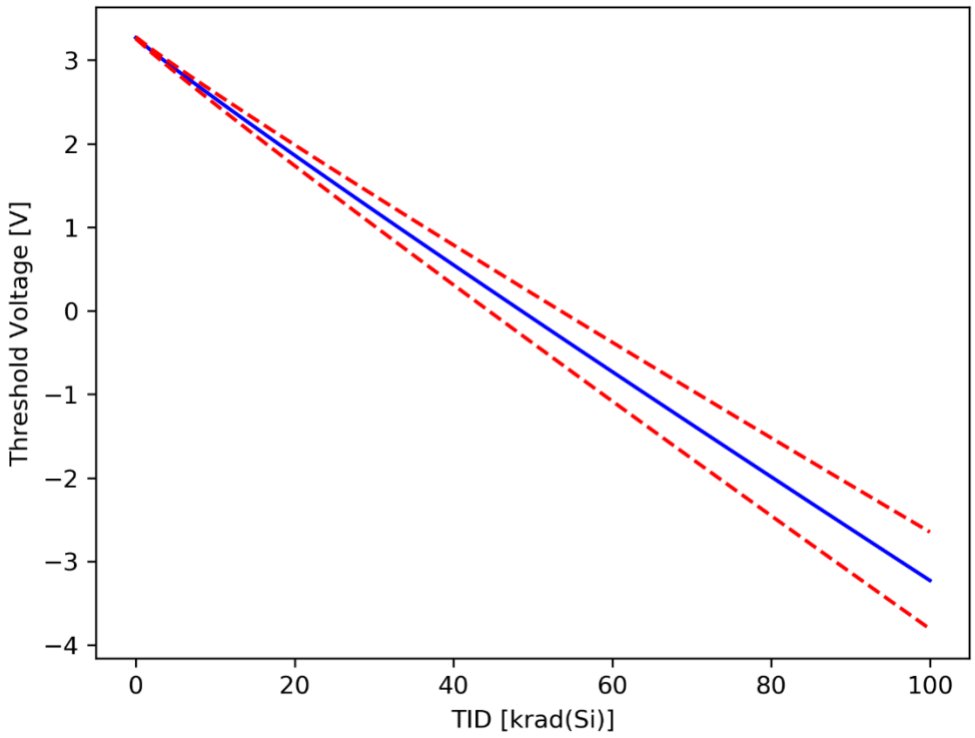

Figure 3 – Expected parameter degradation curve, with mean in blue and red lines representing one standard deviation

B. Building IRF510 Radiation Model

To determine the TID-induced parameter shifts in the IRF510, 12 devices were irradiated to several doses: 1, 3, 5, and 10 krad(Si). After each of these dose levels, I_D-V_G sweeps were performed so as to later extract the threshold voltage and transconductance parameter using methods described in [18], and the python code that was used to extract them is contained in Appendix A – Component Radiation Modeling Code. Each transistor was biased at 100 mV from drain to source and 5 V from gate to source during irradiation to emulate worst case irradiation conditions of the device being on [19].

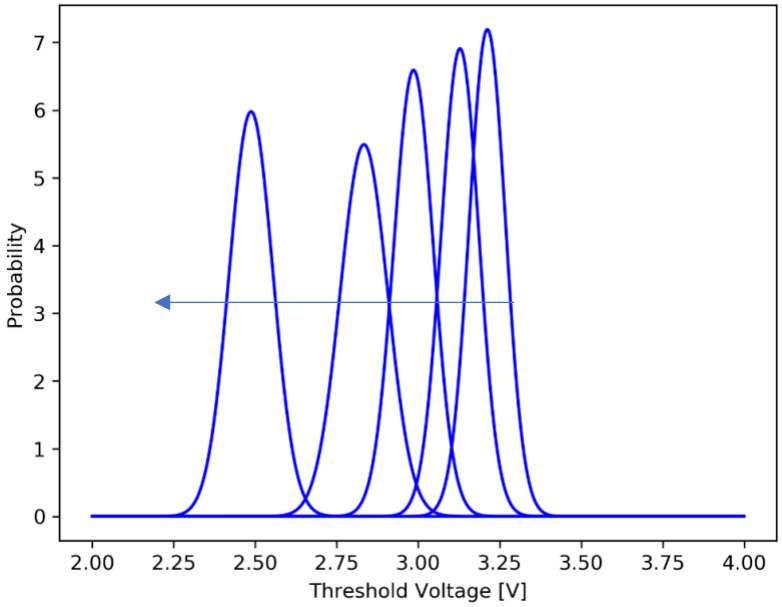

Figure 4 – Probability distribution function for threshold voltages for increasing functions of dose (right-to-left). Generated distributions from experimental data.

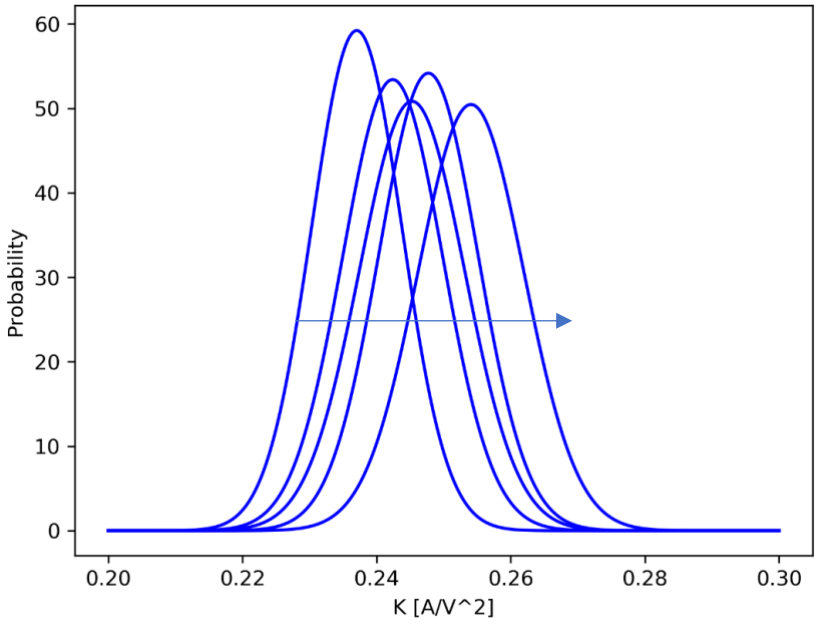

Figure 5 – Probability distribution function of transcoductance parameter for increasing functions of dose(left-to-right). Generated distributions from experimental data.

Figure 4 shows that the mean threshold voltage decreases as a function of dose, while Figure 5 shows that the mean transconductance parameter shifts by only a small amount as a function of TID. Since the threshold voltages of n-MOSFETs typically decrease with TID, a modeling method for parameters that decrease with time from [6] can be used to model the mean and is described by Eqn. 2.

$$\overline{V_T}(D) = \overline{V_{T0}}[1 - A_0 D^m] \tag{2}$$

where $\overline{V_T}(D)$ is the mean threshold voltage as a function of dose, D is dose, $\overline{V_{T0}}$ is the mean pre-irradiation threshold voltage, A_0 and m are constants. Taking the natural log of both sides of this equation and rearranging gives Eqn. 3.

$$\ln\left(\frac{\overline{V_{T0}} - \overline{V_T}(D)}{\overline{V_{T0}}}\right) = mln(D) + \ln(A_0) \tag{3}$$

which has the same form as a linear function:

$$y = mx + b \tag{4}$$

Performing a linear regression on the data using Eqn. 3, the constants *m* and A_0 can be calculated to give a generalized model for Eqn. 2. Note that the parameters were extracted using threshold voltage extraction methods in [18].

The model described above was coded in Python and is described in Appendix A – Component Radiation Modeling Code. This was done for both the transconductance parameter and the threshold voltage, with the standard deviations of each being fit with a linear regression. Figure 6 shows the model with the included data overlayed for the threshold voltage, and Figure 7 shows what the distributions look like for 0, 1, 3, 5, and 10 krad(Si) in this device.

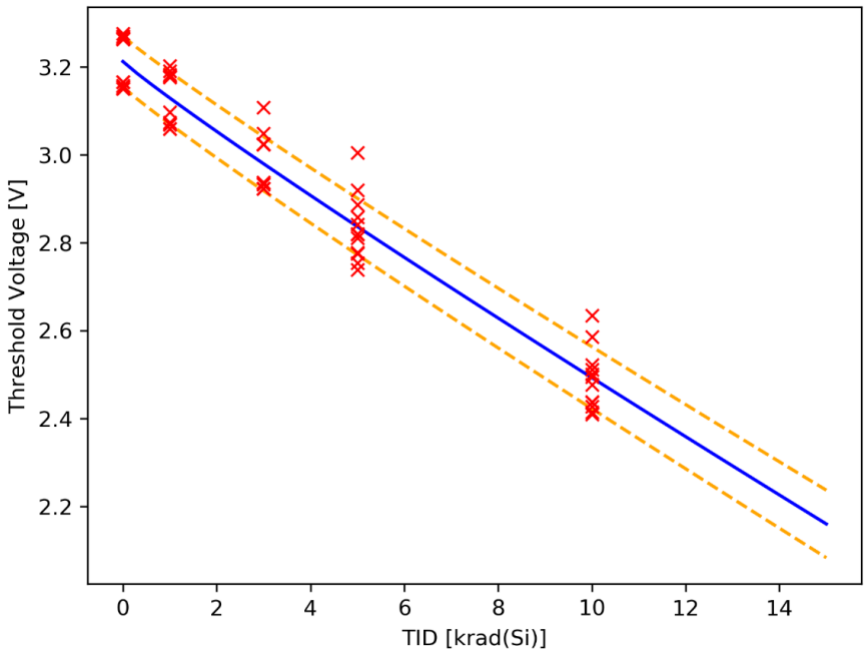

Figure 6 – Theshold voltage model as a function of dose; average value (blue), standard deviation (yellow), and raw data (red)

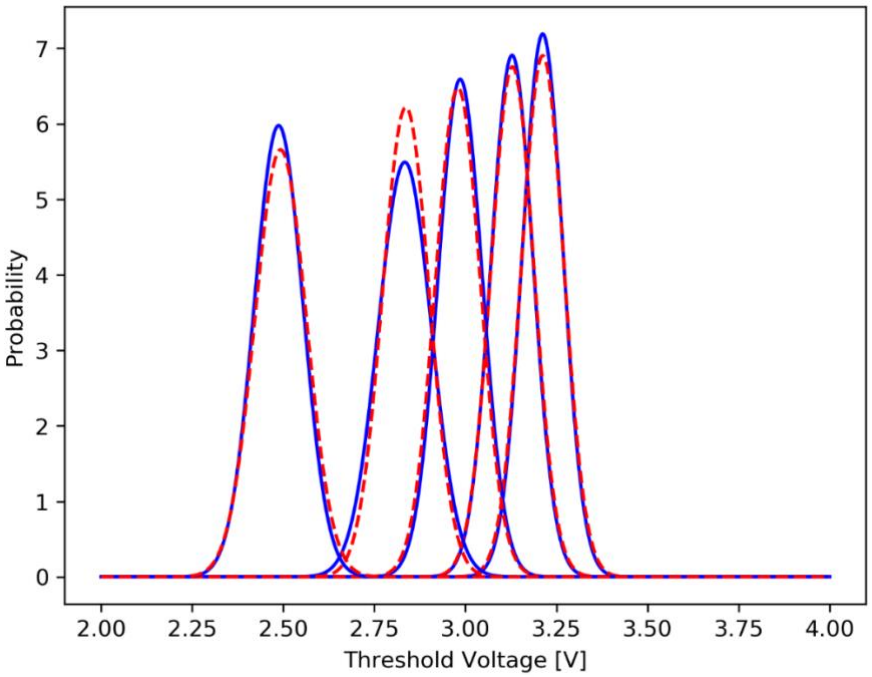

Figure 7 – Data distributions (blue) with modeled distributions (red) for threshold voltage

C. Discussion

Figure 6 shows that the pre-rad threshold voltages for the devices are clumped into two separate groups. This is probably due to the COTS parts that were used being from different lots; a problem that would not be as prevalent in more controlled manufacturing environments. Since these two groups are about 100 mV apart, the standard deviation at 0 rads(Si) is wider than what would normally be expected. If each group of transistors (labelled top and bottom for reference) is looked at individually, the model looks like the two graphs in Figure 8. Both of these plots show that these groups of 6 transistors each increase in variability for their threshold voltage as a function of dose.

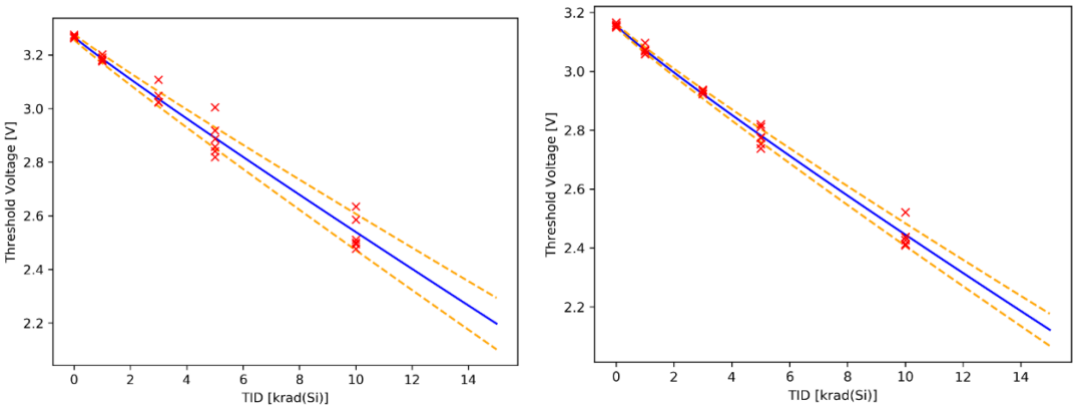

Figure 8 – Two groupings of transistors (top group on left, bottom group on right) showing tighter distributions than combined; threshold voltage as function of dose and the composite of these two make the bimodal distribution in Figure 9.

Since these two groups are combined together for the model used in Figure 6, the standard deviation does not seem to increase greatly as a function of dose. Overlaying the two plots of Figure 8, Figure 9 demonstrates that the two groups of transistors overlap in a bimodal distribution as a function of dose. This is behavior that might be expected from COTS parts, but to decrease

variability in the system, parts could be sorted before implementing them into an experimental setup or into the final system. For the purposes of this work, the bimodal distribution is used going forward for the threshold voltage, but the transconductance parameter is described using a single Gaussian due to the small magnitude of the shifts and for simplicity.

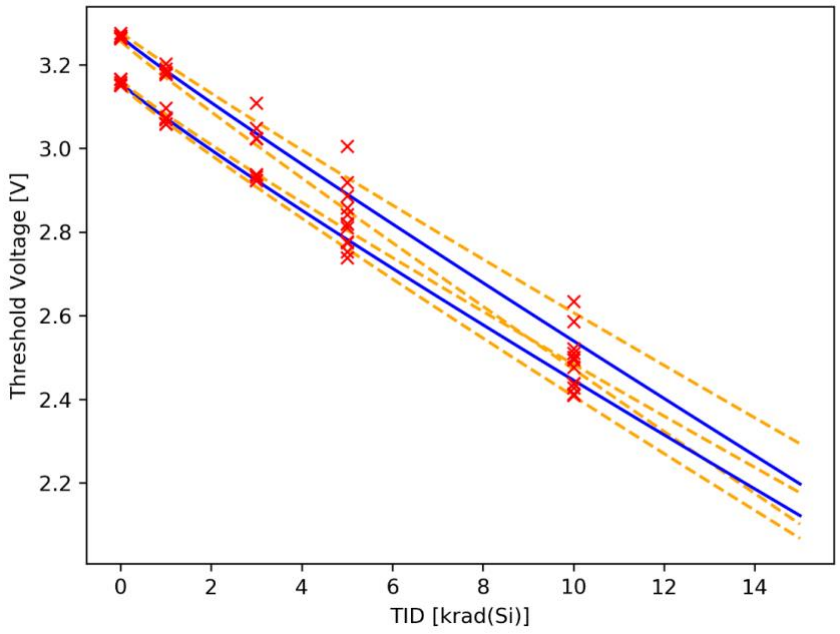

Figure 9 – Overlay of two plots of Figure 8; shows the wide standard deviation of Figure 6

One might note that the threshold voltage and transconductance parameter may be correlated rather than being independent. However, for the purposes of this work, the two are treated independently since the correlation as a function of TID is weak, as shown in Figure 10. It also might be noted that a Student-T distribution could be used in this circumstance rather than a Gaussian due to the smaller population size, but for the sake of simplicity, the Gaussian is used here. The user can adjust the distributions for parameters as needed to increase accuracy and confidence.

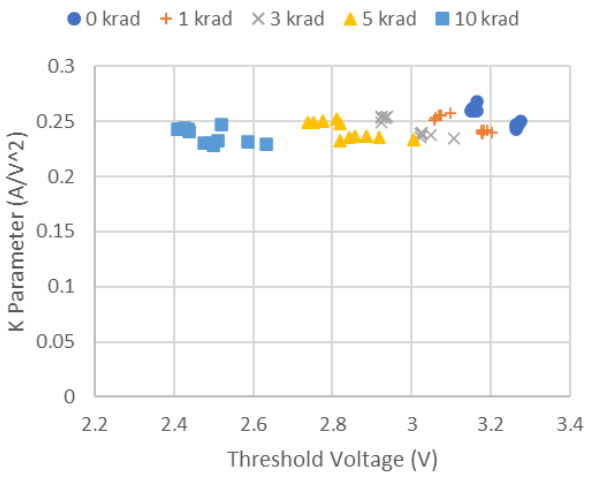

Figure 10 - Correlation between threshold voltage (VT) and transconductance parameter (K) for all data points taken during TID modeling. No significant correlation between the parameters is shown, so sampling from the model can be randomly sampled.

IV. MODIFIED BAYESIAN MODELING OF SYSTEM

A. Analog System Design

The system chosen to demonstrate this technique is a differential amplifier shown in Figure 2. Consisting of 4 IRF510 transistors, this system was chosen due to its simplicity. A common problem in this type of differential amplifier is mismatch between pairs of transistors [4]. This problem is especially prevalent when using COTS parts due to the variability in parameters. To mitigate this effect, a small resistance is placed at the source of the current source transistor, resulting in the final circuit design shown in Figure 11.

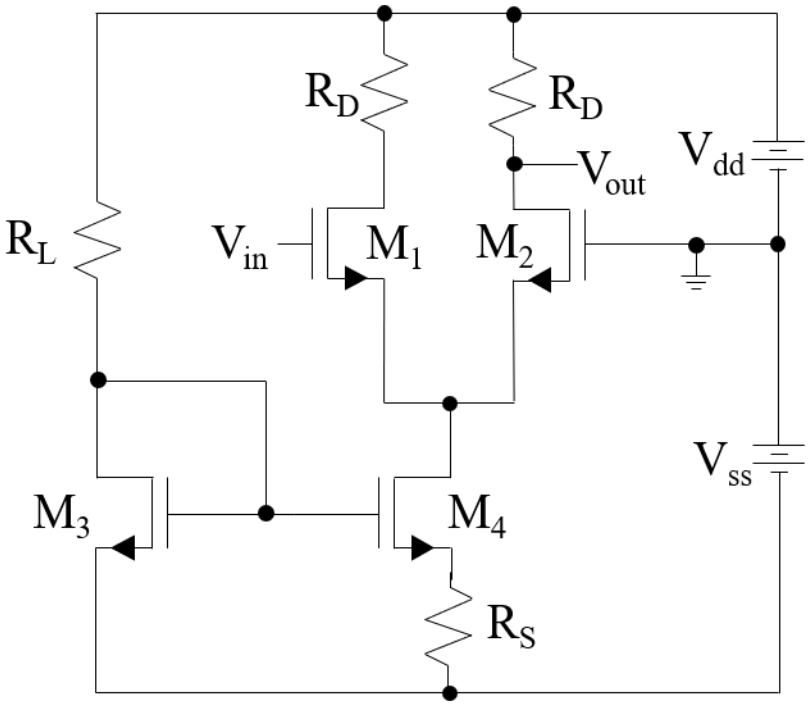

Figure 11 – Differential amplifier with current mirror source resistance

B. System Level Parameters

Three different system level parameters were chosen to represent the functionality of the differential amplifier in Figure 11. The primary output parameter is the maximum gain of the

system, which is traditionally calculated from the quotient of the output voltage and the input voltage. To make the measurements in this example easy, the DC gain was calculated by taking the derivative of the output voltage as the input voltage is swept from low to high. This allows the amplifier to start with transistor M_1 off and M_2 on, as the gate of M_1 starts with a low voltage. As the input voltage is swept to a high value, the operation of the amplifier reaches its equilibrium point at a certain voltage, determined by the difference in threshold voltages of M_1 and M_2. At this point, both transistors have the same amount of current flowing through them, and the gain is at its maximum. This input voltage value is labeled the "center voltage" for the purposes of this thesis. Also, at this point, the slope of the output voltage curve as a function of input should be the maximum, so the maximum value of the derivative of this function is labeled the "max gain". After this equilibrium point, the input voltage into the system increases, allowing M_2 to turn off and M_1 to turn on. As seen in Figure 12, the derivative of the output voltage with respect to the input voltage has a certain width, and the third and final parameter tracked is the "gain width", represented by the width of the peak curve at half the maximum, or the full width, half max (FWHM). This parameter represents the range of inputs centered around the center voltage that would produce an output of at least 50% the maximum gain. Table 1 summarizes these three parameters in a more concise way.

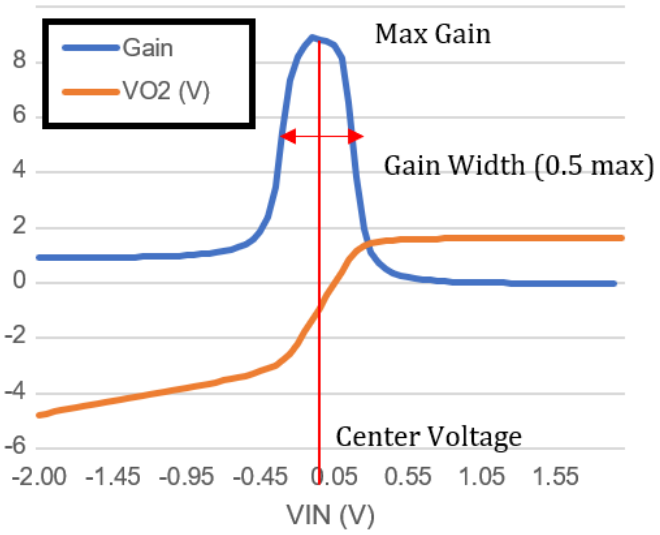

Figure 12 – V_{out} and derivative (gain), with definitions of output parameters

Table 1 – System parameter summarization

Parameter	Short Description	Functional Value
Center Voltage	Input voltage that provides maximum gain	$V_{in}\ such\ that\ g(V_{in}) = \max\left(\frac{dV_{out}}{dV_{in}}\right)$
Max Gain	Maximum value of the derivative of V_{out} with respect to V_{in}	$\max\left(\frac{dV_{out}}{dV_{in}}\right)$
Gain Width	FWHM of derivative of V_{out} with respect to V_{in}	$\lvert V_a - V_b \rvert$ for $V_a, V_b = index\left(\frac{1}{2}\max\left(\frac{dV_{out}}{dV_{in}}\right)\right)$

C. Predicting System Level Response

Taking the model developed in Chapter III for the IRF510, the system level response can be predicted simulations from a random sampling of the distributions of parameters. This process can be done for 4 transistor models (independent of each other) and used in a SPICE simulator to get an output response. Using NGSpice (see Appendix B – Circuit Simulation Code), a total of

1000 simulations was performed where 4 level 1 transistor models were built for the IRF510s and inserted into a SPICE circuit identical to that in Figure 11. The system parameters discussed in the previous section were tracked and Figure 13 shows the results.

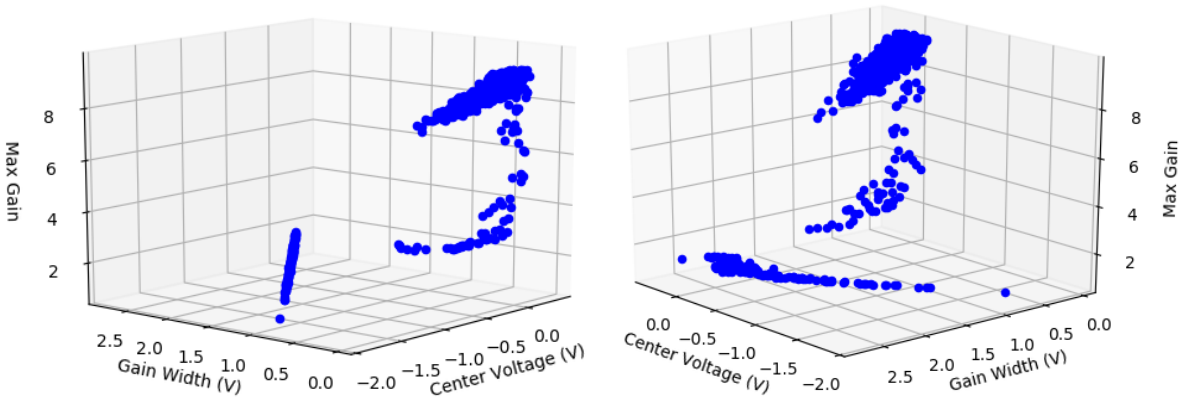

Figure 13 – Simulation results of system parameters, plotted in 3D. Both plots are same data set, just rotated by 90 degrees.

This system predicts that there are several modes of operation that occur as the devices are irradiated. The diff amp starts in normal operation, where the gain is closest to its maximum value (around a gain of 10), and the center voltage is close to 0. This implies that the transistors are matched well, and the current mirror is biasing the diff pair correctly. Another regime of operation predicted is that the gain decreases, but the center voltage stays around 0. This is most likely caused by one of the differential pair transistors coming out of saturation. Finally, the third mode of operation is the straight line with wide gain widths and low max gain. This is most likely caused by the current source to the diff pair coming out of saturation, causing the amplifier to fail completely.

For comparison, the same simulations were run using the distribution from 0 krad(Si). Figure 14 shows the Max Gain vs. the Center Voltage for both the pre-rad and 10 krad(Si)

simulations. Note that the pre-rad simulations, while all high in gain and relatively close to a center voltage of 0, do contain some variability due to the wider distribution of parameters from the COTS parts.

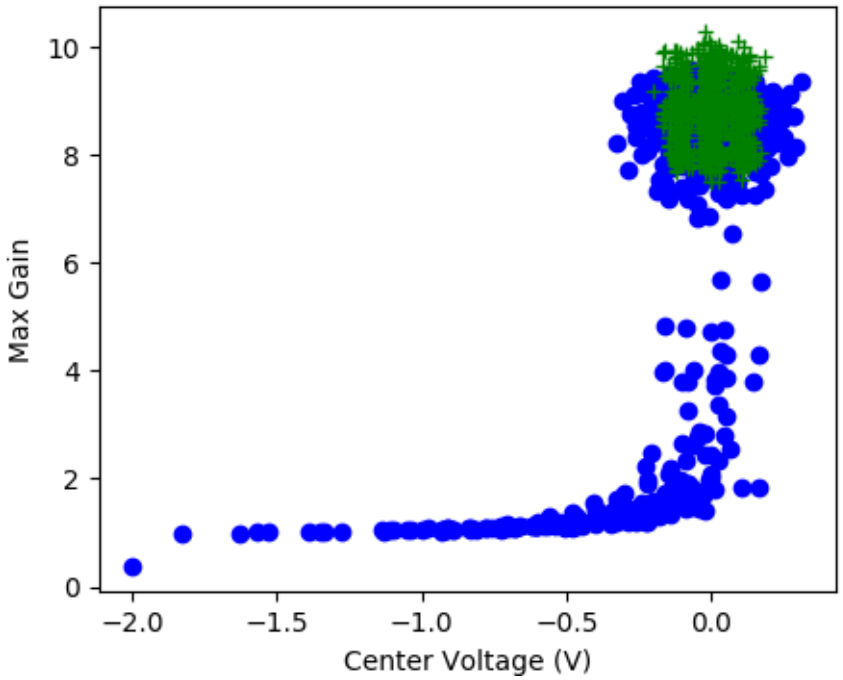

Figure 14 – Max Gain vs. Center Voltage for pre-rad (green crosses) and 10krad (blue dots) from SPICE simulation of differential amplifier.

V. EXPERIMENTAL VERIFICATION

To accurately verify the system's simulated response to dose, one approach might be to create a population of test circuits to then be irradiated and the output response observed. However, to have any confidence in these measurements, a large population of systems might need to be tested [20]. Testing a large population of some systems is impractical due to their cost. Simulations are not well suited to provide this confidence on their own, so some sort of experimental verification should be done to validate the simulations.

A. Pair-Wise Transistor Swapping

In the case of the example system (the differential amplifier), it would be unreasonable to build over 30 circuits to sample the output distribution. This would mean creating at least 30 PCBs and purchasing over 120 IRF510 transistors, adding on the cost to populate and test such boards. The following method utilizes the same 12 IRF510 transistors that were used in the development of the component model to test the system at 10 krad(Si).

Instead of producing a population of circuits, this method utilizes one (or a few) boards that have sockets provided for the transistors. This allows devices with different radiation induced parameter shifts to be swapped in and out, creating a large number of possible system outcomes. If the transistors are sorted by threshold voltage shift immediately after irradiation, the ranked transistors can be used to simulate small and large parameter shifts between components, as well as a distribution in between. In other words, the ranked transistors can be paired-up with other transistors of differing threshold voltage shift, and the system output can be observed.

The advantage of using such a method is having knowledge of how each device in the system has degraded when you get the output response. Theoretically, it would be possible to

obtain the same information out of fully constructed, non-modular systems, but the overhead of time and nodal access would be paid for all 30+ circuits.

B. Experimental Setup

Figure 15 shows the PCB used for experimentation. This board consisted of four three-pronged sockets for the IRF510 transistors, switches for purposes of biasing during irradiation, and a fan to keep all the devices cool during circuit operation. The source series resistor (R_S) was chosen to be 1 Ω, the differential resistors (R_D) were chosen to be 100 Ω, and the current biasing load (R_L) was chosen to be 56 Ω.

Figure 15 – Circuit board used to perform system level tests

The 12 transistors that were irradiated to 10 krad(Si) were ranked from least to greatest threshold voltage shift and assigned a letter, as shown in Table 2. These were biased at 100mV V_{DS} and 5V V_{GS} during irradiation, and control transistors 13-16 were biased the same amount of time.

Table 2 – Ranked threshold voltage shift of transistors after 10 krad(Si), resulting in assigned letter

Serial #	ΔV_T (V)	Assigned Letter	Serial #	ΔV_T (V)	Assigned Letter
1	0.717	K	7	0.741	E
2	0.634	S	8	0.787	A
3	0.678	M	9	0.764	C
4	0.648	P	10	0.739	H
5	0.715	L	11	0.772	B
6	0.734	J	12	0.751	D

Table 3 – List of experiment runs, the transistors used, and subsequent system gain

Run #	Transistors	Gain	Run #	Transistors	Gain	Run #	Transistors	Gain
1	A,P,13,14	8.90	21	A,J,B,K	6.64	41	14,E,15,16	2.36
2	P,A,13,14	8.81	22	J,A,K,B	8.21	42	14,A,15,16	2.47
3	C,L,13,14	8.69	23	J,A,B,S	7.60	43	A,P,B,16	1.92
4	L,C,13,14	8.73	24	A,J,S,B	1.69	44	P,A,B,16	1.86
5	E,J,13,14	9.02	25	J,E,D,M	8.58	45	P,A,S,16	1.76
6	J,E,13,14	8.96	26	E,J,M,D	8.55	46	A,P,S,16	1.77
7	A,J,13,14	8.81	27	L,C,H,K	8.71	47	E,J,H,16	1.79
8	J,A,13,14	8.85	28	C,L,K,H	9.18	48	J,E,H,16	1.71
9	15,16,B,S	7.94	29	P,A,B,K	7.18	49	L,S,A,D	8.91
10	15,16,S,B	10.12	30	A,P,K,B	8.22	50	B,L,K,C	8.36
11	15,16,D,M	8.73	31	A,P,S,B	2.58	51	E,A,J,S	2.16
12	15,16,M,D	10.44	32	P,A,B,S	7.72	52	B,H,M,A	3.32
13	15,16,H,K	9.46	33	C,J,H,M	2.15	53	14,15,16,13	8.89
14	15,16,K,H	10.00	34	J,C,M,H	0.99	54	15,16,13,14	9.31
15	15,16,B,K	10.58	35	E,P,B,K	7.29	55	15,16,13,14	9.17
16	15,16,K,B	8.63	36	P,E,K,B	8.49	56	16,15,14,13	9.13
17	A,P,B,S	7.64	37	13,14,B,16	1.99	57	B,D,S,M	9.20
18	P,A,S,B	2.58	38	13,14,H,16	1.91	58	D,A,B,S	7.55
19	C,L,D,M	8.40	39	13,14,M,16	1.89	59	S,A,L,K	8.67
20	L,C,M,D	8.80	40	14,L,15,16	2.37	-	-	-

Immediately after irradiation, the 16 transistors were placed in the sockets of the board (from left to right is M_1, M_2, M_3, and M_4, which is the order listed in the table) in various combinations, as described in Table 3.

C. Analysis of Results

Table 3 shows the maximum differential amplifier gain values for certain combinations of transistors used in the system. It is clear to see that the gain is the highest when the current bias transistors are closely matched, such in runs 1-8 (where control transistors are used). Certainly, this is not the only factor that determines gain magnitude, as seen in runs 9-16, where the gain is higher than in runs 1-8, but the current mirror transistors are not matched. This is due to the current biasing of the circuit in M_3 is larger when the threshold voltage of M_3 only changes a small amount, as seen in runs 20, 36, and 57. However this is not universally true, as the differential pair matching also plays a major role in gain, as seen in the difference between runs 30 and 31. Here the same transistors are used for M_1, M_2, and M_4, and the threshold voltage change is smaller in M_3 for run 31 than in run 30, yet the gain goes down. This is due to M_4 coming out of saturation, not allowing as much current to flow in the differential pair, and thus diminishing the gain.

The other system parameters were tracked, and the results are plotted in Figure 16 on top of the simulated data of the last section. Overall, the model does a good job predicting the different modes of operation, with one exception being the black, green, and cyan shapes with a center voltage of approximately 2 V. These test cases are caused by experimental runs with one irradiated and one control part in the differential pair, causing a large difference in threshold voltage between the two.

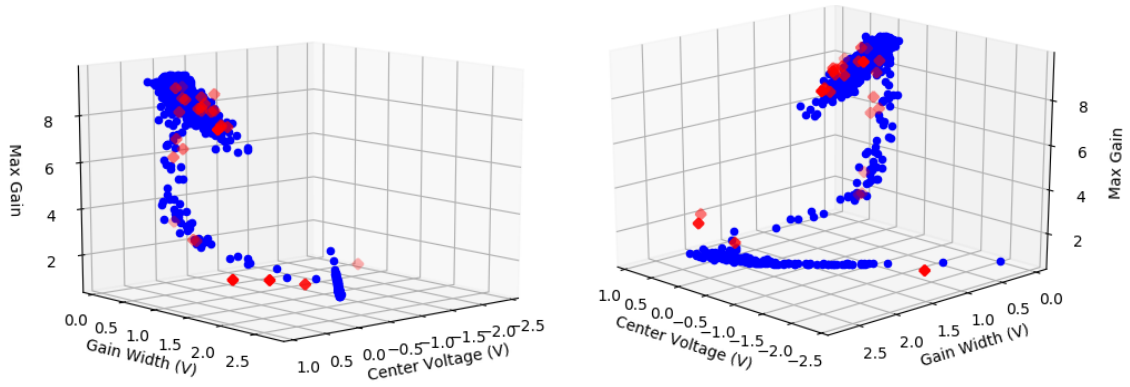

Figure 16 – Simulations (blue) with data overlayed in red. Data sets are identical, just rotated by 90 degrees.

VI. CONCLUSIONS

This work demonstrates a method using Bayesian analysis and Monte Carlo simulations to predict the TID response of a simple system. By using a small population of parts to generate a radiation degradation distribution for the transistors in the circuit without use of MCMC analysis, the TID response of the system can be predicted accurately without having to test a large population of system circuits. This predicted response is verified with a sampling of the system level responses by doing a novel pairwise experiment with the dedicated parts. The agreement between the experiment and the simulations is good, showing that the permutations of parts used in the experiment are a good representation of the circuit response for the part population.

While the specifics of this system and experiment are described in this document, the technique could be applied to other systems. For example, a sub-system that requires multiple operational amplifiers, such as an instrumentation amplifier, could be modeled by developing a TID model for the operational amplifier, then predicting the TID response of the system via simulation by sampling the model parameter distributions. Then a pairwise experiment with the irradiated microprocessors could be done to verify the results. This allows a larger sampling space of system outputs to be sampled without having to construct numerous system level circuits. Thus, this technique can be used to save both time and money in radiation testing at the system level.

VII. REFERENCES

[1] G. A. Ausman and F. B. McLean, "Electron-hole pair creation energy in SiO2," *Applied Physics Letters*, vol. 26, no. 4, pp. 173-175, 2008.

[2] F. Sholze, H. Rabus and G. Ulm, "Mean energy required to produce an electron-hole pair in silicon for photons of energies between 50 and 1500eV," *Journal of Applied Physics*, vol. 84, no. 5, p. 2926, 1998.

[3] J. R. Schwank, M. R. Shaneyfelt, D. M. Fleetwood, J. A. Felix, P. E. Dodd, P. Paillet and V. Ferlet-Cavrois, "Radiation Effects in MOS Oxides," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 1833-1853, 2008.

[4] A. F. Witulski, M. B. Smith, N. Mahadevan, A. L. Sternberg, C. Barnes, D. Sheldon, R. D. Schrimpf, G. Karsai and M. W. McCurdy, "Bayesian Modeling of TID Response Variation of COTS Power MOSFETs," in *2017 17th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Geneva, Switzerland, 2017.

[5] R. Ladbury, "Radiation Hardening at the System Level," in *IEEE NSREC Short Course*, Honolulu, 2007.

[6] J. W. McPherson, Reliability Physics and Engineering: Time-To-Failure Modeling, New York: Springer Science, 2010.

[7] J. Kruschke, Doing Bayesian Data Analysis, Elsevier, 2015.

[8] R. Ladbury, "Statistical Techniques for Analyzing Process or \textquotedblleftSimilarity\textquotedblright Data in TID Hardness Assurance," *IEEE Transactions on Nuclear Science,* 12 2010.

[9] R. Ladbury and M. J. Campola, "Statistical Modeling for Radiation Hardness Assurance: Toward Bigger Data," *IEEE Transactions on Nuclear Science,* vol. 62, p. 2141–2154, 10 2015.

[10] R. Ladbury and B. Triggs, "A Bayesian Approach for Total Ionizing Dose Hardness Assurance," *IEEE Transactions on Nuclear Science,* vol. 58, p. 3004–3010, 12 2011.

[11] R. Ladbury, J. L. Gorelick and S. S. McClure, "Statistical Model Selection for TID Hardness Assurance," *IEEE Transactions on Nuclear Science,* vol. 56, p. 3354–3360, 12 2009.

[12] R. Ladbury, J. L. Gorelick, M. A. Xapsos, T. O\textquotesingleConnor and S. Demosthenes, "A Bayesian Treatment of Risk for Radiation Hardness Assurance," in *2005 8th European Conference on Radiation and Its Effects on Components and Systems*, 2005.

[13] M. Pignol, F. Malou and C. Aicardi, "COTS in Space: Constraints, Limitations and Disruptive Capability," in *Radiation Effects on Integrated Circuits and Systems for Space Applications*, Springer International Publishing, 2019, p. 301–327.

[14] Z. J. Diggins, N. Mahadevan, E. B. Pitt, D. Herbison, R. M. Hood, G. Karsai, B. D. Sierawski, E. J. Barth, R. A. Reed, R. D. Schrimpf, R. A. Weller, M. L. Alles and A. F. Witulski, "Bayesian Inference Modeling of Total Ionizing Dose Effects on System Performance," *IEEE Transactions on Nuclear Science,* vol. 62, no. 6, p. 2517, 2015.

[15] A. Patil, D. Huard and C. J. Fonnesbeck, "PyMC: Bayesian Stochastic Modeling in Python," *Journal of Statistical Software,* vol. 35, no. 4, 2010.

[16] MIL-HDBK-814, Ionizing Dose and Neutron Hardness Assurance Guidelines for Microcircuits and Semiconductor Devices, 1994.

[17] A. S. Sedra and K. C. Smith, Microelectronic Circuits, New York: Oxford University Press, 2004.

[18] A. Ortiz-Conde, F. J. G. Sanchez, J. J. Liou, A. Cerdeira, M. Estrada and Y. Yue, "A review of recent MOSFET threshold voltage extraction methods," *Microelectronics Reliability,* vol. 42, p. 583–596, 4 2002.

[19] D. M. Fleetwood, "Total Ionizing Dose Effects in MOS and Low-Dose-Rate-Sensitive Linear-Bipolar Devices," *IEEE Transactions on Nuclear Science,* vol. 60, p. 1706–1730, 6 2013.

[20] "NIST/SEMATECH e-Handbook of Statistical Methods," 30 October 2013. [Online]. Available: https://doi.org/10.18434/M32189A . [Accessed 18 May 2020].

VIII. APPENDIX A – COMPONENT RADIATION MODELING CODE

```
import csv,scipy,math
from matplotlib import pyplot as plt
import numpy as np
from scipy import signal
from mpl_toolkits.mplot3d import Axes3D

datapath = '../../2020Data/'
```

```
      files =
['PreradVth_IDVG.csv','1kradVth_IDVG.csv','3kradVth_IDVG.csv','5kradVth_IDVG.
csv','10kradVth_IDVG.csv']
      DPI = 300
      top = [1,2,7,8,10,11]
      bottom = [0,3,4,5,6,9]

      def returnCSVfloat(file='',column=0,allData=True):
          with open(file) as csvfile:
              reader = csv.reader(csvfile)
              outdata = []
              for row in reader:
                  try:
                      d = float(row[column])
                      if allData:
                          for col in row:
                              c = float(col)
                      outdata.append(d)
                  except ValueError:
                      continue
          return outdata

      class data2020():

          def __init__(self):
              self.data = None
          #returns gaussian function of input x from average mu and standard
dev sig. normalized to have area=1 under curve
          def gaussian(self,x, mu, sig):
              func = np.exp(-np.power(x - mu, 2.) / (2 * np.power(sig, 2.)))
              return func/sig/np.sqrt(2*3.1415962)


          def writeCSV(file='untitled.csv',data=[]):
              with open(file,'w',newline='') as csvfile:
                  writer = csv.writer(csvfile,delimiter=',')
                  for row in data:
                      writer.writerow(row)

          def getVth(vgs,ids,vds=0.1):
              y1 = ids #signal.savgol_filter(ids,51,5)
              gm1 = np.diff(y1)/np.diff(vgs)
              xcoor = vgs[np.argmax(gm1)]
              ycoor = y1[np.argmax(gm1)]
              k = np.max(gm1)
              th = -(ycoor-xcoor*k)/k
              return th-vds/2,k/vds

          def extractVth2CSV(self):
              data = np.zeros((len(files),16,2))
              for f in range(len(files)):
                  for num in range(1,17):
                     filepath = str(datapath) + str(files[f])
                     vgate = returnCSVfloat(filepath,0)
                     idrain = returnCSVfloat(filepath,num)
```

```
                    th,k = self.getVth(vgate,idrain)
                    data[f,num-1] = [th,k]

self.writeCSV('../../2020Data/VthFromExtraction.csv',data[:,:,0])
            self.writeCSV('../../2020Data/KFromExtraction.csv',data[:,:,1])

        def primeData(self):
            self.data = np.zeros((len(files),16,2))
            for num in range(1,17):
                filepathTH = str(datapath) + 'VthFromExtraction.csv'
                filepathK = str(datapath) + 'KFromExtraction.csv'
                th = returnCSVfloat(filepathTH,num-1)
                k = returnCSVfloat(filepathK,num-1)
                for f in range(len(files)):
                    self.data[f,num-1,:] = [th[f],k[f]]

        # inputs dose in krad(Si) and outputs the avg, std, and initial
average for threshold voltage; data=0 means all, data=1 means top, data=2
means bottom used for function
        def VtModel(self,dose,data=0):
            self.primeData()
            tid = np.array([0,1,3,5,10])
            dataPoints = self.data[:,0:12,0]
            if(data==1):
                dataPoints = np.transpose(np.array([dataPoints[:,i] for i
in top]))
            elif(data==2):
                dataPoints = np.transpose(np.array([dataPoints[:,i] for i
in bottom]))
            vtavg = np.average(dataPoints,axis=1)
            vtstd = np.std(dataPoints,axis=1)
            s0avg = vtavg[0]
            mavg,bavg = np.polyfit(np.log(tid[1:]),np.log(1-
vtavg[1:]/s0avg),1)
            mstd,bstd = np.polyfit(tid,vtstd,1)
            return s0avg*(1-np.exp(bavg)*dose**mavg),mstd*dose+bstd,s0avg

        # inputs dose in krad(Si) and outputs the avg, std, and initial
average for k parameter; data=0 means all, data=1 means top, data=2 means
bottom used for function
        def KModel(self,dose,data=0):
            self.primeData()
            tid = np.array([0,1,3,5,10])
            dataPoints = self.data[:,0:12,1]
            if(data==1):
                dataPoints = np.transpose(np.array([dataPoints[:,i] for i
in top]))
            elif(data==2):
                dataPoints = np.transpose(np.array([dataPoints[:,i] for i
in bottom]))
            kavg = np.average(dataPoints,axis=1)
            kstd = np.std(dataPoints,axis=1)
            s0avg = kavg[0]
            mavg,bavg = np.polyfit(np.log(tid[1:]),np.log(1-
kavg[1:]/s0avg),1)
```

```
            mstd,bstd = np.polyfit(tid,kstd,1)
            return s0avg*(1-np.exp(bavg)*dose**mavg),mstd*dose+bstd,s0avg

        # plot=0 all data, plot=1 top data, plot=2, bot data
        def TIDvVtFigure(self,dist=False,plot=0):
            x = np.linspace(0,100)
            y,std,s0 = self.VtModel(x,plot)
            plt.figure()
            plt.plot(x,y,'b')
            if(dist):
                plt.plot(x,y+std,'--r')
                plt.plot(x,y-std,'--r')
            filename = 'TIDvVtFigure'
            if(plot==1): filename = filename + '_top'
            if(plot==2): filename = filename + '_bot'
            plt.xlabel('TID [krad(Si)]')
            plt.ylabel('Threshold Voltage [V]')
            plt.savefig(filename,dpi=DPI,bbox_inches='tight')

        def TIDvlogVtFigure(self):
            x = np.linspace(0,20)
            y,std,y0 = self.VtModel(x)
            plt.plot(np.log(x),np.log((y0-y)/y0))
            plt.show()
            plt.savefig('TIDvlogVtFigure',dpi=DPI,bbox_inches='tight')

        # plot=0 all data, plot=1 top data, plot=2, bot data
        def TIDvKFigure(self,dist=False,plot=0):
            x = np.linspace(0,100)
            y,std,s0 = self.KModel(x,plot)
            plt.figure()
            if(dist):
                plt.plot(x,y+std,'--r')
                plt.plot(x,y-std,'--r')
            plt.plot(x,y,'b')
            filename = 'TIDvKFigure'
            if(plot==1): filename = filename + '_top'
            if(plot==2): filename = filename + '_bot'
            plt.xlabel('TID [krad(Si)]')
            plt.ylabel('K [A/V^2]')
            plt.savefig(filename,dpi=DPI,bbox_inches='tight')

        # plot=0: model distribution, plot=1: data distribution, plot=2:
both
        def VthvProbFigure(self,plot=0):
            self.primeData()
            tid = [0,1,3,5,10]
            x = np.linspace(2,4,1000)
            plt.figure()
            filename = 'VthvProbFigure'
            if(plot==0): filename = filename + '_model'
            if(plot==1): filename = filename + '_data'
            for n in range(0,len(tid)):
                avg,std,s0 = self.VtModel(tid[n])
```

```
                y1 =
self.gaussian(x,np.average(self.data[n,0:12,0]),np.std(self.data[n,0:12,0]))
                y = self.gaussian(x,avg,std)
                if(plot==1 or plot==2):
                    plt.plot(x,y1,'b')
                if(plot==0 or plot==2):
                    plt.plot(x,y,'--r')
            plt.xlabel('Threshold Voltage [V]')
            plt.ylabel('Probability')
            plt.savefig(filename,dpi=DPI,bbox_inches='tight')

        # plot=0: model distribution, plot=1: data distribution, plot=2:
both
        def KvProbFigure(self,plot=0):
            self.primeData()
            tid = [0,1,3,5,10]
            x = np.linspace(0.2,0.3,1000)
            filename = 'KvProbFigure'
            plt.figure()
            if(plot==0): filename = filename + '_model'
            if(plot==1): filename = filename + '_data'
            for n in range(0,len(tid)):
                avg,std,s0 = self.KModel(tid[n])
                y1 =
self.gaussian(x,np.average(self.data[n,0:12,1]),np.std(self.data[n,0:12,1]))
                y = self.gaussian(x,avg,std)
                if(plot==1 or plot==2):
                    plt.plot(x,y1,'b')
                if(plot==0 or plot==2):
                    plt.plot(x,y,'--r')
            plt.xlabel('K [A/V^2]')
            plt.ylabel('Probability')
            plt.savefig(filename,dpi=DPI,bbox_inches='tight')

        # plot model with data; plot=0 is all data, plot=1 is top dist.,
plot = 2 is bottom, plot=3 is all data with top and bottom distributions
        def TIDvVt_WithDist_WithDataFigure(self,plot=0):
            self.primeData()
            tid = np.linspace(0,15)
            tid_data = [0,1,3,5,10]
            dataIndicies = range(12)
            fileName = 'TIDvVt_WithDist_WithDataFigure'
            vtavgTOP,vtstdTOP,s0 = self.VtModel(tid,1)
            vtavgBOT,vtstdBOT,s0 = self.VtModel(tid,2)
            plt.figure()
            if(plot==1):
                dataIndicies = top
                fileName = 'TIDvVt_WithDist_WithDataFigure_top'
                plt.plot(tid,vtavgTOP,'b')
                plt.plot(tid,vtavgTOP+vtstdTOP,'--',color='orange')
                plt.plot(tid,vtavgTOP-vtstdTOP,'--',color='orange')
            elif(plot==2):
                dataIndicies = bottom
                fileName = 'TIDvVt_WithDist_WithDataFigure_bottom'
                plt.plot(tid,vtavgBOT,'b')
```

```
                plt.plot(tid,vtavgBOT+vtstdBOT,'--',color='orange')
                plt.plot(tid,vtavgBOT-vtstdBOT,'--',color='orange')
            elif(plot==3):
                fileName = 'TIDvVt_WithDist_WithDataFigure_doubleDist'
                plt.plot(tid,vtavgTOP,'b')
                plt.plot(tid,vtavgTOP+vtstdTOP,'--',color='orange')
                plt.plot(tid,vtavgTOP-vtstdTOP,'--',color='orange')
                plt.plot(tid,vtavgBOT,'b')
                plt.plot(tid,vtavgBOT+vtstdBOT,'--',color='orange')
                plt.plot(tid,vtavgBOT-vtstdBOT,'--',color='orange')
            else:
                vtavg,vtstd,s0 = self.VtModel(tid,0)
                plt.plot(tid,vtavg,'b')
                plt.plot(tid,vtavg+vtstd,'--',color='orange')
                plt.plot(tid,vtavg-vtstd,'--',color='orange')
            for n in dataIndicies:
                plt.plot(tid_data,self.data[:,n,0],'xr')
            plt.xlabel('TID [krad(Si)]')
            plt.ylabel('Threshold Voltage [V]')
            plt.savefig(fileName,dpi=DPI,bbox_inches='tight')

    # plot model with data; plot=0 is all data, plot=1 is top dist., plot =
2 is bottom, plot=3 is all data with top and bottom distributions
        def TIDvK_WithDist_WithDataFigure(self,plot=0):
            self.primeData()
            tid = np.linspace(0,15)
            tid_data = [0,1,3,5,10]
            dataIndicies = range(12)
            fileName = 'TIDvK_WithDist_WithDataFigure'
            kavgTOP,kstdTOP,s0 = self.KModel(tid,1)
            kavgBOT,kstdBOT,s0 = self.KModel(tid,2)
            plt.figure()
            if(plot==1):
                dataIndicies = top
                fileName = fileName + '_top'
                plt.plot(tid,kavgTOP,'b')
                plt.plot(tid,kavgTOP+kstdTOP,'--',color='orange')
                plt.plot(tid,kavgTOP-kstdTOP,'--',color='orange')
            elif(plot==2):
                dataIndicies = bottom
                fileName = fileName + '_bot'
                plt.plot(tid,kavgBOT,'b')
                plt.plot(tid,kavgBOT+kstdBOT,'--',color='orange')
                plt.plot(tid,kavgBOT-kstdBOT,'--',color='orange')
            elif(plot==3):
                fileName = fileName + '_doubleDist'
                plt.plot(tid,kavgTOP,'b')
                plt.plot(tid,kavgTOP+kstdTOP,'--',color='orange')
                plt.plot(tid,kavgTOP-kstdTOP,'--',color='orange')
                plt.plot(tid,kavgBOT,'b')
                plt.plot(tid,kavgBOT+kstdBOT,'--',color='orange')
                plt.plot(tid,kavgBOT-kstdBOT,'--',color='orange')
            else:
                kavg,kstd,s0 = self.KModel(tid,0)
                plt.plot(tid,kavg,'b')
```

```
                plt.plot(tid,kavg+kstd,'--',color='orange')
                plt.plot(tid,kavg-kstd,'--',color='orange')
            for n in dataIndicies:
                plt.plot(tid_data,self.data[:,n,1],'xr')
            plt.xlabel('TID [krad(Si)]')
            plt.ylabel('K [A/V^2]')
            plt.savefig(fileName,dpi=DPI,bbox_inches='tight')

        # not finished
        def KvVtvTID3DFigure(self):
            self.primeData()

        # not finished
        def VtvTIDvP3DFigure(self):
            self.primeData()
            tid = np.linspace(0,10)
            x = np.linspace(2,4,500)
            xv,yv = np.meshgrid(x,tid)
            avgs,stds,s0s= self.VtModel(yv)
            prob3d = self.gaussian(xv,avgs,stds)
            fig = plt.figure()
            ax = fig.gca(projection='3d')
            ax.plot_surface(xv,yv,prob3d)
            plt.show()

    model = data2020()
    model.TIDvVt_WithDist_WithDataFigure(0)
    model.TIDvVt_WithDist_WithDataFigure(1)
    model.TIDvVt_WithDist_WithDataFigure(2)
    model.TIDvVt_WithDist_WithDataFigure(3)
    model.TIDvK_WithDist_WithDataFigure(0)  #not yet there
    model.TIDvK_WithDist_WithDataFigure(1)  #not yet there
    model.TIDvK_WithDist_WithDataFigure(2)  #not yet there
    model.TIDvK_WithDist_WithDataFigure(3)  #not yet there
    model.TIDvVtFigure(True,0)
    model.TIDvVtFigure(True,1)
    model.TIDvVtFigure(True,2)
    model.TIDvKFigure(True,0)               #not yet there
    model.TIDvKFigure(True,1)               #not yet there
    model.TIDvKFigure(True,2)               #not yet there
    model.VthvProbFigure(0)
    model.VthvProbFigure(1)
    model.VthvProbFigure(2)
    model.KvProbFigure(0)
    model.KvProbFigure(1)
    model.KvProbFigure(2)
```

IX. APPENDIX B – CIRCUIT SIMULATION CODE

```
#!/usr/bin/python

import subprocess,time,sys,os
import numpy as np
from matplotlib import pyplot as plt
```

```
import csv
from mpl_toolkits.mplot3d import Axes3D
import extractionTools as et

def make_model(kp,vth):
    sweepFile = open('idvg_sweep.cir','w')

    sweepFile.write('* IDVG\n')
    sweepFile.write('\n')
    sweepFile.write('.model Q18 NMOS (level=1 VT0=%s KP=%s)\n' % (th,kp))
    sweepFile.write('\n')
    sweepFile.write('M18 D18 G 0 0 Q18 \n')
    sweepFile.write('VD18 D18 0 0.1\n')
    sweepFile.write('VG G 0 0.0\n')
    sweepFile.write('\n')
    sweepFile.write('.dc VG 0 5 0.001\n')
    sweepFile.write('.print dc i(VD18)\n')
    sweepFile.write('\n')
    sweepFile.close()

def makeTransModel(kp,vth):
    return 'NMOS (level=1 VT0=%s KP=%s)' % (vth,kp)

def make_system(kp_array,th_array,rS):
    systemFile = open('system_circuit.cir','w')

    systemFile.write('*System Level Circuit\n')
    systemFile.write('.model Q1 ' + makeTransModel(kp_array[0],th_array[0]) +
'\n')
    systemFile.write('.model Q2 ' + makeTransModel(kp_array[1],th_array[1]) +
'\n')
    systemFile.write('.model Q3 ' + makeTransModel(kp_array[2],th_array[2]) +
'\n')
    systemFile.write('.model Q4 ' + makeTransModel(kp_array[3],th_array[3]) +
'\n')
    systemFile.write('\n')
    systemFile.write('M1 D1 VIN S1 S1 Q1\n')
    systemFile.write('M2 D2 0 S1 S1 Q2\n')
    systemFile.write('M3 D3 D3 VSS1 VSS1 Q3\n')
    systemFile.write('M4 S1 D3 S4 S4 Q4\n')
    systemFile.write('RS1 S4 VSS1 ' + str(rS) + '\n')
    systemFile.write('RL1 VDD1 D3 56\n')
    systemFile.write('RD1 VDD1 D1 100\n')
    systemFile.write('RD2 VDD1 D2 100\n')
    systemFile.write('V1 VDD1 0 5\n')
    systemFile.write('V2 VSS1 0 -8\n')
    systemFile.write('V3 VIN 0 0\n')
    systemFile.write('\n')
    systemFile.write('.dc V3 -2.5 2.5 0.01\n')
    systemFile.write('.print dc v(D1) v(D2)\n')
    systemFile.write('\n')
    systemFile.close()

def run_sim():
```

```
    subprocess.Popen('ngspice    -b    idvg_sweep.cir    |    grep    ^[0-9]    >
idvgSweeps',shell=True,stderr=subprocess.PIPE,stdout=subprocess.PIPE).wait()
    sweepFile = open('idvgSweeps','r')
    data = sweepFile.readlines()
    sweepFile.close()
    vg = []
    lev1 = []
    lev8 = []
    for i in range(len(data)):
        point = data[i].replace(',',' ')
        vg.append(float(point.split()[1].strip()))
        lev8.append(-float(point.split()[2].strip()))
    return vg,lev8

def runSysSim():
    subprocess.Popen('ngspice   -b   system_circuit.cir   |   grep   ^[0-9]   >
systemOutput',shell=True,stderr=subprocess.PIPE,stdout=subprocess.PIPE).wait(
)
    outputFile = open('systemOutput','r')
    data = outputFile.readlines()
    outputFile.close()
    vo1 = []
    vo2 = []
    vin = []
    for i in range(len(data)):
        point = data[i].replace(',',' ')
        vin.append(float(point.split()[1].strip()))
        vo1.append(float(point.split()[2].strip()))
        vo2.append(float(point.split()[3].strip()))
    return vin,vo1,vo2

def returnCSVfloat(file='',column=0,allData=True):
    with open(file) as csvfile:
        reader = csv.reader(csvfile)
        data = []
        for row in reader:
            try:
                d = float(row[column])
                if allData:
                    for col in row:
                        c = float(col)
                data.append(d)
            except ValueError:
                continue
    return data

#returns vthAvg,vthStd,kAvg,kStd for measured data
def radiationData(dose):
    if (dose==0):
        return 3.21221,0.05548,2.54066e-1,7.90584e-3
    elif (dose==1000):
        return 3.12817,0.05774,2.47716e-1,7.36553e-3
    elif (dose==3000):
        return 2.98569,0.06053,2.45275e-1,7.84776e-3
    elif (dose==5000):
```

```
        return 2.83355,0.07263,2.42396e-1,7.46991e-3
    elif (dose==10000):
        return 2.48724,0.06673,2.37034e-1,6.73769e-3
    else:
        print("not a specified data point")

def getSysData():
    sysX = returnCSVfloat('../../2020Data/SystemLevelTests_Rs1_VO2.csv',0)
    maxvals = []
    centervolt = []
    gainwidth = []
    for run in range(1,60):
        sysData                                                                    =
returnCSVfloat('../../2020Data/SystemLevelTests_Rs1_VO2.csv',run)
        dx = sysX[1] - sysX[0]
        gain = np.gradient(sysData)/dx
        maxvals.append(np.max(gain))
        centervolt.append(sysX[np.argmax(gain)])
        widthtotal = 0
        for x in range(len(sysX)):
            if gain[x] > 0.5*np.max(gain):
                widthtotal = widthtotal + dx
        gainwidth.append(widthtotal)
    return centervolt,maxvals,gainwidth

def runSamples():
    return

start = time.time()
#filepath = '../../2020Data/PreradVth_IDVG.csv'
#dataX = returnCSVfloat(filepath,0)
#dataY = returnCSVfloat(filepath,1)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
numpoints = 1000
irfModel = et.data2020()
dose = 10 #in krad
vtavg,vtstd,s0 = irfModel.VtModel(dose)
kavg,kstd,s0 = irfModel.KModel(dose)
for i in range(numpoints):
   kp = np.random.normal(kavg,kstd,4)
   th = np.random.normal(vtavg,vtstd,4)
   make_system(kp,th,1)
   VIN,VO1,VO2=runSysSim()
   dx=VIN[1]-VIN[0]
   gain=np.gradient(VO2)/dx
   maxgain=np.max(gain)
   centervolt=VIN[np.argmax(gain)]
   gainwidth = 0
   for x in range(len(VIN)):
       if gain[x] > 0.5*maxgain:
           gainwidth = gainwidth + dx
   ax.scatter(centervolt,gainwidth,maxgain,c='b',marker='o')
```

```
#the following section selects the data runs that are useful that were not cut
off early due to user error
syscenter,sysmax,syswidth = getSysData()
ax.scatter(syscenter[0:8],syswidth[0:8],sysmax[0:8],c='m',marker='o',linewidt
h=5)
ax.scatter(syscenter[8:16],syswidth[8:16],sysmax[8:16],c='y',marker='^',linew
idth=5)
ax.scatter(syscenter[16:36],syswidth[16:36],sysmax[16:36],c='r',marker='x',li
newidth=5)
ax.scatter(syscenter[36:39],syswidth[36:39],sysmax[36:39],c='k',marker='^',li
newidth=5)
ax.scatter(syscenter[39:42],syswidth[39:42],sysmax[39:42],c='g',marker='o',li
newidth=5)
ax.scatter(syscenter[42:48],syswidth[42:48],sysmax[42:48],c='c',marker='o',li
newidth=5)
ax.scatter(syscenter[48:52],syswidth[48:52],sysmax[48:52],c='r',marker='x',li
newidth=5)
ax.scatter(syscenter[52:56],syswidth[52:56],sysmax[52:56],c='b',marker='*',li
newidth=5)
ax.scatter(syscenter[56:59],syswidth[56:59],sysmax[56:59],c='r',marker='x',li
newidth=5)
#
ax1.scatter(syscenter[0:8],syswidth[0:8],sysmax[0:8],c='m',marker='o',linewid
th=5)
#
ax1.scatter(syscenter[8:16],syswidth[8:16],sysmax[8:16],c='y',marker='^',line
width=5)
#
ax1.scatter(syscenter[16:36],syswidth[16:36],sysmax[16:36],c='r',marker='x',l
inewidth=5)
#
ax1.scatter(syscenter[48:52],syswidth[48:52],sysmax[48:52],c='r',marker='x',l
inewidth=5)
#
ax1.scatter(syscenter[52:56],syswidth[52:56],sysmax[52:56],c='b',marker='*',l
inewidth=5)
#
ax1.scatter(syscenter[56:59],syswidth[56:59],sysmax[56:59],c='r',marker='x',l
inewidth=5)

ax.set_xlabel('Center Voltage (V)')
ax.set_ylabel('Gain Width (V)')
ax.set_zlabel('Max Gain')
# ax1.set_xlabel('Center Voltage (V)')
# ax1.set_ylabel('Gain Width (V)')
# ax1.set_zlabel('Max Gain')
stop = time.time()
print("Run time: " + str(stop-start))
plt.show()
```