Radiation Effects Screening Campaign Planning Algorithm and Test Platform for Modern Commercial Off the Shelf Electronics

by

Jesse Ward

A Thesis submitted to the Faculty of Graduate Studies of

The University of Manitoba

in partial fulfilment of the requirements of the degree of

MASTER OF SCIENCE

Department of Mechanical Engineering

University of Manitoba

Winnipeg

Copyright © 2023 by Jesse Ward

"*The cosmos is within us. We are made of star-stuff. We are a way for the universe to know itself.*"

Carl Sagan

Abstract

Significant growth in the space industry has occurred in the last decade, largely thanks to technical advancements that have reduced the cost of select subsystems. For instance, structural and mechanical systems have seen cost reductions through more efficient manufacturing methods, less expensive materials, and improved simulations. Further cost reductions can still be obtained in the domain of space electronics. This domain continues to experience high costs due to the manufacturing methods and qualification processes required to certify radiation-hardened electronics. Well-defined processes to approve inexpensive commercial electronics for use in space are needed if the broader space electronics industry is to adopt using them in future designs, thus reducing the costs of spaceflight. This research first conducted two radiation effects screening campaigns to study the viability of using commercial parts in low Earth orbit. A statistical algorithm to aid in planning screening campaigns for commercial electronics is also provided. Finally, this research presents a generic test bench platform which is capable of accommodating various part families to facilitate rapid, cost-effective radiation effects screening. The data obtained from the research provided evidence to suggest that utilization of commercial parts in space is feasible. The algorithm provides methodology to aid in planning screening campaigns such that sample sizes can be selected based on a variety of factors including part parameter control data. The test bench was developed to aid in conducting the two screening campaigns and provided a proof of concept that generic testing platforms utilizing plug-and-play functionality can be effective in enabling researchers to conduct cost-effective screening campaigns. Utilization of the screening algorithm and test bench platform will enable the space industry to effectively certify commercial off-the-shelf electronics for use in low Earth orbit.

Acknowledgements

Firstly, Thank you to my parents, brother, and grandmother, who have loved and supported me throughout all my endeavours. I hope to make you all proud of the man I have become and to do great things with the wisdom and help you have provided me throughout my life.

Thanks to my partner Nina for her love and support throughout my degree. Your kindness and patience helped me manage any stress along the way. I couldn't imagine a better person to have by my side.

A massive thank you to my advisor, Dr. Philip Ferguson. I still remember the day a few years ago during your Aerospace course when you offered the class to apply for a master's if we were interested. It truly changed my life and provided me with a foot in the door to the space industry. For that, I am grateful. Your help, wisdom, and patience helped me learn many things and participate in this fascinating research. Thank you once again.

I want to thank my committee members, Dr. Ian Jeffrey and Dr. Larry Liang, for taking the time to evaluate and critique my research. Your advice and insight helped steer me the right way, allowing me to deliver this thesis for publication.

Finally, thank you to all my friends, lab members and colleagues, especially Jayden McKoy, Aref Asgari and Jason Garr. I could not have finished this research without you. Your support developing the hardware for the test bench and the help provided while conducting the screening at TRIUMF was invaluable. I also want to thank everyone who helped with this document's edits, comments and feedback.

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Motivation

The space industry has seen significant growth in the last decade [1]. Much of this growth has been driven by technical advancements that have reduced the cost of select subsystems. For instance, structural and mechanical systems have seen cost reductions through more efficient manufacturing methods, less expensive materials, improved simulations, and the development of reusable rocketry. A recent example of this improvement includes the Falcon series of rockets by SpaceX, which features the reusability of the first stage of the rocket. [2]

Canada's flagship RADARSat series of spacecraft have experienced similar cost reductions. In 1995 the total cost (excluding launch) of the RADARSat-1 satellite was approximately 620 million dollars [3]. In 2019 the RADARSat Constellation Mission was completed for only 600 million dollars despite consisting of three satellites instead of only one [4].

The cost of space launches has reduced in large part thanks to the development of commercial space launch systems [2], however there is still room for further cost reductions in the domain of space electronics. The primary reasons for the high cost of space electronics is the exhaustive testing, analysis, and qualification that is often required to prove that the electronic components can survive the extreme environment of space [5]. Violent thermal cycles, mechanical vibration, material out-gassing, and atomic oxygen [6] all pose significant risks to delicate semiconductors and solder joints. The move towards

miniaturized electronics exacerbates these risks, due to the limited space between transistors increasing the likelihood of contact / shorting, requiring painstaking analysis and testing to confirm their suitability for the space environment [7]. However, while thermal, mechanical, and chemical stresses on electronics significantly affect the price of space-qualified electronics, the strongest cost driver is the space radiation environment [6].

The challenge associated with space radiation stems from how the effects of radiation can cause electronics to fail either due to ionization or from particles directly damaging the component through collision [7]. In space, there are a variety of particles that contribute to the total spectrum of radiation which includes energized protons and neutrons as well as heavy ions of various elements [7]. These particles come from the sun as well as galactic cosmic rays [7]. The particles collide with the electronics causing different types of damage: single event effects, total ionizing dose effects, and displacement damage [7]. Single event effects (SEEs) commonly affect digital parts and are caused by single energetic particles colliding with the components, sensitive volume N+ [8]. An illustration of this phenomenon is shown in figure 1.1. The N+ region contains a high concentration of electrons which produce a cascade of charge propagation when hit by an energetic particle [7]. This collision can cause either recoverable or unrecoverable errors depending on the type of component [8]. A common example of a SEE is bit flipping, whereby a single bit in memory is flipped from its proper on or off state [9]. The bit flipping phenomenon is generally considered to be a soft error and can be mitigated using error correcting code [10].

Total ionizing dose (TID) and displacement damage (DD) both result from a buildup of damage over time either through directly displacing atoms in the atomic lattice of which the electronics material is comprised of or charging and ionizing the base level of the component [11]. Both effects lead to degraded performance until the material

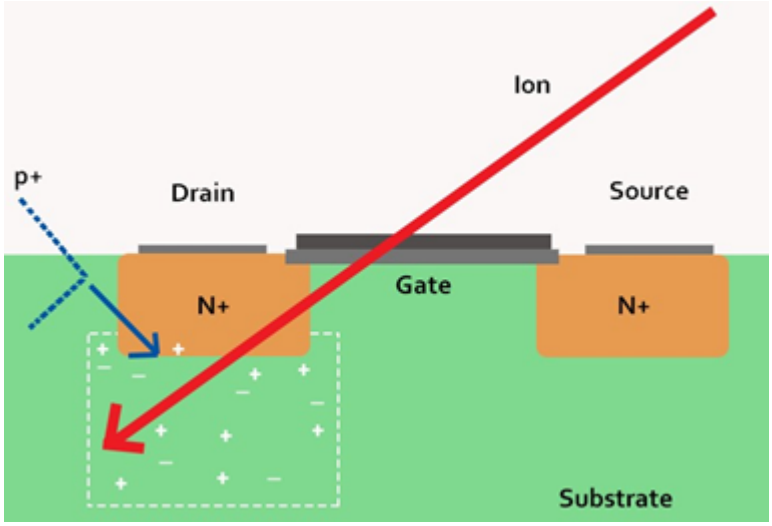

FIGURE 1.1: Single event effects involve ionizing particles passing through the sensitive volume of an active device. The particle may be a heavy ion (right), or a recoil ion produced by a proton (p+) hitting the material lattice (left) [8]

properties change enough to cease function [12]. Testing for these failures at the energy levels which reflect the expected environment is commonly done using cobalt-60 gamma sources, cyclotron proton beams, and neutron beams [12].

Mitigation for the space radiation environment is primarily done using radiation-hardened components, which are designed using different techniques that provide resistance to the different failure modes [13]. Specifically, these techniques include changing the materials used to manufacture the part and implementing redundant elements inside the device. However, using more resistant materials and implementing internal redundancies results in a more expensive part [14]. Even after a lengthy design process, these parts must undergo certification to be labelled as radiation-hardened. The certification process thoroughly tests the component for the different failure modes at various energy levels, often at a price of several thousand dollars per hour [5].

As a result, characterizing and certifying the radiation-hardness of a component is a significant development cost that is subsequently passed down to the consumer. Costs are

further compounded by small order quantities and lack of variety in options compared to the number of parts produced at the consumer level [15]. Consumer electronics utilizing mass production methods are known to have lower costs than batch-produced products [16]. These factors all contribute to radiation-hardened parts being more expensive by several orders of magnitude.

Commercial electronic components are inexpensive compared to radiation-hardened parts but offer other advantages. Notably, the processing speed of commercial parts is often faster. Commercial microprocessors, for example, have improved from single-core chips running at 10 MHz to powerful 16-core chips which run at speeds of 5 GHz [17]. Such improvements make computer systems more capable than those used in the past while being smaller and less expensive than their predecessors. Improvements such as processor speed have been accomplished by significantly changing the materials, architecture, and manufacturing methods over part generations [17].

There are historical reasons to exclude commercial parts in heavy radiation environments, including space. Early studies found that commercial parts became more susceptible to radiation-induced failure as the newer generation components shrunk in size [18]. However, there are upsides to smaller parts. Newer parts operate at the nanometer scale, necessitating higher quality inspection measures during manufacturing to ensure better lot uniformity and part qualities than ever before [19]. These components are also so small that employing redundant circuitry in all facets of the system is viable for mitigating radiation damage [7]. The RAID-1 architecture [20], which is a method of data mirroring, involves two or more drives reading and writing identical data. Using redundancy methods such as RAID-1 is one possibility that future space systems could now employ without a significant cost, volume, or mass increase [20].

CubeSats are a market which has already begun to take advantage of the lower costs and

better performance of commercial electronics. CubeSats are small, low-budget satellites that are now increasingly used to conduct science in low Earth orbit (LEO) [21]. Graduate-level CubeSat projects such as the University of Manitoba STARLab's Iris Mission require low-cost consumer electronics strictly due to budget and manufacturing constraints despite the higher risk of failure [22]. Regardless of the increased risk, dozens of such satellites have recently performed their missions without failure [21]. The success of these small satellites suggests that using commercial parts in developing low-cost next-generation satellite computer systems is possibly a viable option.

However, the viability of using COTS electronics in space goes along with the aforementioned increased risk of mission failure [22]. System reliability is of chief concern when operating in space [23]. A part failure cannot be repaired without incurring high costs. The Hubble telescope repair mission, for example, resulted in 700 million dollars of spending [24]. Redundant systems may now be more easily implemented using COTS electronics [20] but may not always be feasible to introduce due to budget constraints [23]. Engineers must be able to confidently ensure their design will operate reliably during its mission lifetime.

It is clear then that well-defined processes for consumer electronics part screening are needed if the broader space electronics industry is to adopt using commercial electronics within a larger scope. These parts must be tested to ensure they can survive in the low Earth orbit environment because systems reliability is a key component of successful space missions [23]. There are questions about how many parts must be tested to screen an entire purchase lot for quality control. What testing methods should be considered? To what extent can analysis help reduce the screening of the parts?

My research studied the viability of available radiation test facilities in conjunction with statistical conformance and sampling methods to screen a large variety of commercial

off-the-shelf (COTS) electronics. These components provided a modern characterization of radiation tolerance for the tested part families, which subsequently provided data which was used to develop an algorithm that aids in determining what tests are needed during the screening process for candidate components. A generic test bench platform was also created with hardware development help from Jayden McKoy and Jason Garr, which facilitates rapid, cost-effective part screening in addition to the algorithm. Updated radiation statistics and low-cost screening strategies for COTS electronics will provide an opportunity to reduce the cost of next-generation avionics used in low Earth orbit by several orders of magnitude.

1.2 Research Hypotheses and Contributions

My research screened a large variety of COTS electronics at the Tri-University Meson Facility (TRIUMF) and used the resultant data to develop a statistical algorithm which provides recommendations on what types of testing are needed for different part categories for future missions. My colleagues and I developed a test bench platform to aid in rapidly and cost-effectively screening the selected components. At the outset of this research, I hypothesized that due to the previously mentioned properties of commercial electronics, I could develop a comprehensive screening system that would improve the reliability of commercial electronics for low Earth orbit missions. Specifically, I hypothesized that:

1. Modern COTS electronics can be effectively utilized in large-scale, commercial satellites operating in low Earth orbit for missions with a duration of less than five years, without compromising reliability.

2. One can develop an algorithm which provides cost effective radiation screening and mitigation strategies for COTS electronics.

3. It is cost-effective to screen COTS electronics for use in space using a reconfigurable, and reusable test platform when compared to purchasing radiation-hardened components.

By verifying these hypotheses, I made the following contributions to space electronics:

- A modern radiation tolerance characterization for a select set of COTS electronics for use in low Earth orbit.

- An algorithm that suggests appropriate radiation screening sample sizes for different part families.

- A reconfigurable, and reusable testing platform developed alongside Jayden McKoy and Jason Garr, which is capable of cost effectively and rapidly testing a variety of candidate components for use in low Earth orbit.

1.3 Thesis Summary

This first chapter provides the motivation for this research, including an overview of the space electronics industry and its high costs. I discuss how radiation in space results in a substantial increase in electronic component cost and provide examples of commercial parts used in low-budget applications. I presented my hypotheses and contributions, which aim to address some questions about utilizing COTS electronics on spacecraft operating in low Earth orbit.

Chapter 2 reviews existing literature relevant to my hypotheses, including an overview of how the CubeSat market uses commercial electronics despite increased the risk of mission failure. I discuss how statistical algorithms can aid in designing screening studies. I also present information on how generic test bench platforms enable more efficient, cost-effective screening campaigns.

Chapter 3 provides a comprehensive overview of the methods I used in completing this thesis. I discuss the selection of various electronic part families and how the generic test bench facilitated rapid radiation tolerance screening. I provide an overview of the screening strategies employed for two testing campaigns and discuss developing an algorithm that aids in selecting screening campaign sample sizes for different part families.

Chapter 4 is dedicated to discussing the radiation tolerance data obtained from the two screening campaigns for each part and presents an overview of the algorithm I developed. Additionally, I provide a discussion on validating the algorithm using radiation effects data from prior studies and provide test bench performance results.

Chapter 5 summarizes the conclusion, research contributions, and recommendations for future work.

2 Literature Review and Additional Research

This chapter provides background information on the testing and use of COTS electronics within the space industry. A literature review of the current use and reliability of COTS electronics in space is provided as well as details regarding the use statistical methods to help plan and conduct screening campaigns. Additionally, the utilization of test bench platforms and how they can aid in performing future radiation tolerance screening is presented.

2.1 Usability of COTS electronics in space

Hypothesis one predicts that commercial electronics are suitable for use in low Earth orbit for mission durations of less than five years. The introduction mentioned that low-budget satellite platforms such as CubeSats provide anecdotal evidence suggesting this hypothesis is true. A significant question my research aims to address still exists that prevents the adoption of COTS electronics within the more significant space electronics industry. This question is one of parts reliability and how it can impact the success of a spacecraft mission. I discuss in this section some examples of commercial space electronics use, their reasons for being implemented despite reliability being a concern, and how mission duration plays a vital role in allowing COTS electronics to be used. I also provide an overview of how the space industry continuously publishes radiation tolerance testing data to aid engineers in reliably developing future spacecraft.

Low-budget satellite platforms have become an increasingly popular design in recent years amongst graduate labs focused on conducting scientific missions in low Earth orbit [21]. Private organizations such as Swarm Technologies [25] and Kepler Communications [26] are also using CubeSats to create fleets of satellites known as constellations that provide access to global high-speed internet. Future CubeSats will include the fleet built by various universities across Canada as part of the Canadian CubeSat Project [27]. A common characteristic of these satellites is commercial electronics being implemented in various electronic subsystems, whether at the component or system level.

Iris is one such CubeSat which implements COTS electronics [22]. Designed by the University of Manitoba, Iris is an upcoming satellite launching into low Earth orbit in 2023. The satellite will study the weathering of geological samples when exposed to the vacuum of space and has an expected mission duration of 1.3 years. Iris utilizes COTS electronics in many facets of the onboard computer systems. The sun sensors developed by York University utilize commercial MLX75306 Optical Arrays to detect sunlight [28]. The satellite's two MT9D111 cameras are also off-the-shelf components. The cameras are both 1-megapixel in resolution and are used to take photos of the satellite's various mineral samples [22]. The CubeSat's lower available budget necessitated using commercial components without performing radiation qualification tests. However, Iris itself, being inexpensive, allows it to become a real-world screening platform for the broader space industry [22].

The CanX-2 satellite is another example of utilizing a COTS component in space without modifications [29]. The GPS receiver used in this satellite was a NovAtel OEM4-G2L. This component determined orbit and performed radio occultation [29]. The satellite was launched in 2008 and operated nominally until as recently as 2017, with a mission duration of nine years [30].

Mission duration is an important parameter to consider when developing electronic systems that must withstand some level of radiation. The total ionizing dose accumulates over the lifetime of a spacecraft. Likewise, longer mission durations provide more opportunities for a stray energetic particle to strike sensitive electronics, inducing a single-event upset. In recent years, many companies have been launching large satellite constellations with shorter mission durations [31]. Constellations like SpaceX's Starlink routinely launch replacement satellites for the constellation on a routine basis as older ones retire [32]. Mission durations of five years are typical for satellite constellations operating within LEO, including Starlink [31]. My research uses this five-year duration when calculating hardness assurance rankings in the parts I tested.

Reliability is always a concern when subjecting electronics to radiation. The space industry combats this risk by conducting radiation tolerance screening campaigns and publishing the data for others. Electronics that pass testing often become publicly known and can be used on future spacecraft with matching mission criteria. This work is a collaborative effort, and government institutions such as the National Aeronautics and Space Administration (NASA) and the European Space Agency (ESA) publish compendiums of test data in both the IEEE Transactions on Nuclear Science journal and the radiation effects data workshop (REDW) of the IEEE Nuclear & Space Radiation Effects Conference (NSREC).

In recent years the publications have shifted towards data on COTS electronics. NASA's 2022 compendium of testing for candidate parts included many commercial components. The parts were screened for heavy ion-induced single-event effects at the Texas A&M University Cyclotron (TAMU). One of these parts was an industrial Wi-Fi transceiver produced by Doodle Labs. This component successfully survived the heavy ion testing up to a linear energy transfer (LET) of 37 MeV-cm^2/mg and was considered to have potential use in a future deep-space mission [33]. Metal-oxide-semiconductor field-effect transistor (MOSFET) testing was conducted in the same 2022 report and included automotive-grade AEC-Q100-certified parts. The BSS806NE N-Channel MOSFET was one such part that survived the single-event testing. Other MOSFETs, such as the SFC85N9051 experienced gate rupture at a LET of 37 MeV-cm^2/mg. [33]

The Jet Propulsion Lab (JPL) conducted an intensive TID screening campaign from 2003 to 2009. JPL screened 65 parts across a variety of part categories. Tested devices include the TC4423 gate driver, which survived without failure up to 50 kRad(Si). I corroborated the findings of this component in this thesis up to 25 kRad(Si). Some components, such as the AD823 op-amp, experienced catastrophic failure at 5.4 kRad(Si), making them unusable for space applications. Other op-amps experienced parameter degradation over 10 to 50 kRad(Si). Many of the parts remained within specification limits. [34]

Publishing data on 65 parts is excellent for future engineers. However, this data accounts for a tiny portion of the available commercial parts on the market. More data is needed to build a reliable understanding of how commercial components using modern manufacturing methods fail in response to radiation. My work contributes to this endeavor to reliably use COTS electronics in space by expanding the publicly available test data for various part families, including my most recent publication about automotive grade MOSFETs in the 2022 NSREC workshop [35] (the first time data on these MOSFETS had ever been published).

2.2 Statistical Methods

My second hypothesis predicted that one could develop an algorithm to aid in planning radiation screening campaigns. Such algorithms could help provide a rigorous process for making decisions regarding various aspects of a test plan. Some aspects of interest would include how many samples to test, which forms of testing would be needed, and how one could use past data to inform decisions about future testing. Engineers could optimize the number of test runs required and improve the overall efficiency of a test campaign by developing guidelines on how to rigorously screen COTS components for spaceflight to a desired level of confidence and reliability.

While not specifically related to radiation screening for space applications, other researchers have developed statistical test methodologies that help direct researchers in planning appropriate test campaigns. For example, one algorithm developed by researchers of the Computational Mechanics and Reliability Group [36] aims to aid in planning general qualification testing of electronics. This method utilizes a "self-organizing map" neural network to provide a similarity index between various commercial electronics of the same category. The similarity index scores how similar a new candidate part is to those already having test data available. This algorithm assumes that when a sufficient level of similarity is achieved between a new and existing part, the new candidate will likely exhibit similar performance in qualification testing. The study validated this assumption to be true for how parts respond to hot solder dip tests. Deciding whether one needs to perform qualification tests on the new components requires confidence that the algorithm's assumptions are valid for the type of qualification testing one wishes to perform.

An algorithm developed by S. Saxena and Co. [37] provides a method to determine the viability of conducting degradation testing on lithium-ion batteries. Accelerated testing saves time and reduces the costs of a screening campaign. However, careful selection of stress factors is required to ensure that the test suitably provides an understanding of how the battery would perform in the typical use case. This algorithm aids in identifying and ranking stress factors based on how they affect battery degradation. The stress factor rank list obtained for a particular battery allows one to plan a qualification test campaign with only the relevant top-ranked stress factors.

Medical studies are similar to qualification testing in that one must often balance the sample size and duration of testing against the budget and time available to perform the study. Sample size selection plays a critical role in influencing the conclusions one can make about the error, variability, and distribution of the data obtained from a study [38]. The actual value of these parameters is often unknown. Obtaining statistically significant results requires a method for sample size selection to ensure one does not make type I and type II errors while adequately determining the variance [38]. Type I errors reject a null hypothesis despite it being true, and type II errors fail to reject the null hypothesis because it is indeed false [39]. Methods developed by C. Beleites and Co. found that specifying acceptable confidence interval widths in conjunction with using estimates of variance based on prior data allowed one to determine test sample sizes related to biospectroscopy [40]. As the study notes, small sample size problems are common in many other fields [40]. Radiation effects testing is one such field that follows this trend, often due to the costs associated with renting out facilities for testing [5].

The radiation effects field stands to benefit from the development of test planning algorithms. Past literature shows that statistical algorithms are valuable in planning scientific studies and qualification tests. Such algorithms can aid in various ways, including selecting sample sizes, constraining what testing is needed, and even eliminating the need for testing in some scenarios. My research provides a sample size selection algorithm which aids in more efficiently and cost-effectively planning future screening campaigns.

2.3 Test Platforms for Electronics

A test bench is required to monitor any electronic item during reliability screening. Whether testing for degradation of solid-state drives due to radiation [41] or checking the power cycling capability of a MOSFET [42], one requires a specialized system capable of monitoring the electrical parameters of interest that are unique to the component or must spend money to manufacture and test a finished design. The system is usually connected to an external computer running software that controls the various test bench inputs. The computer will also log and sometimes analyze the data obtained from the part. A test bench may be highly specialized to a single component or a general design which accommodates many different parts.

Specialized test benches include the custom GBTX ASIC test bench developed for the Large Hadron Collider upgrade program. This test bench allows total dose and single-event upset screening of a custom radiation-hardened ASIC. The test bench features a high-speed socket allowing the user to swap the GBTX chips throughout the screening campaign easily. An Ethernet connection connects the board to the monitoring software's external computer. [43]

Another test bench platform developed by A. Barari, known as the Reconfigurable and Empirical Spacecraft Emulation Testbed (RESET), focuses on testing satellite control systems. Testing the functionality of such systems is similar to testing radiation effects concerning the challenges that must be faced to mimic the space environment accurately. This generalized test platform utilizes drone technology to allow CubeSats to test and evaluate their control systems in a manner which suitably mimics the micro-gravity environment that exists in orbit. This test bench platform is capable of evaluating new satellite control systems and is not specifically geared to test a single solution. [44]

Many existing test bed designs target various aspects of spacecraft testing, and specialized radiation screening systems exist to target a single component, including the design for testing the GBTX ASIC, as mentioned earlier. However, no generalized radiation test bench designs capable of screening many components have been published. Developing a generic testing platform is a way to reduce the costs associated with radiation effects screening campaigns by providing researchers with a reconfigurable test platform capable of handling any candidate component (within the considered part categories) needing radiation hardness certification. My research aided in accomplishing this goal by working with Jayden and Jason to define the requirements needed to design the platform. I also developed automated analysis software to configure and run the test bench once my colleagues had finished designing the hardware.

2.4 Additional Background

2.4.1 Radiation Environment

In this thesis, I considered the low earth orbit (LEO) environment. The standard definition for LEO is an Earth-centered orbit with an altitude of less than 2000 km [45]. The LEO region of space contains the most artificial objects in the Solar system and is likely to remain a popular destination for future space missions [45]. In fact, small-budget CubeSats [27] and large constellations of telecommunications satellites are both becoming a significant percentage of the spacecraft operating in LEO. These satellites stand to benefit from the reduced manufacturing cost gained from using commercial components.

The radioactive environment surrounding Earth is composed of three major groupings: solar particles, galactic cosmic rays, and trapped particles within the Van Allen belts [46]. These three radiation sources are primarily composed of protons and alpha particles, while a small percentage is composed of heavier elements [46]. The six following

heavy ions provide the primary source of heavy ion ionization: H, He, C, N, O, and Fe [46]. The energy levels of the particles can reach hundreds of MeV [7], rendering shielding irrelevant to onboard systems [6].

The Space Environment Information System (SPENVIS) can provide total dose estimates of the radiation environment around Earth for various hypothetical missions. For example, a SPENVIS simulation I conducted provided an estimate of the radiation dose for a LEO satellite mission. The simulated orbit was a circular 500 km altitude orbit with an inclination of 85°. The SHIELDOSE-2 module simulated the electronics being protected behind 3 mm of aluminum shielding [47]. I also performed an additional simulation for a theoretical geostationary Earth orbit (GEO) satellite. The simulated orbit was a circular 35,786 km altitude orbit with zero inclination, and the dose accrued by the electronics was simulated behind 3 mm of aluminum shielding.

The simulations had mission durations ranging from one to five years. The total dose over the mission lifetimes is shown in figures 2.1 and 2.2 for the LEO and GEO simulations, respectively. The results show that the LEO spacecraft would receive 6.5 kRad(Si) after five years. In contrast, the GEO spacecraft may receive up to 35 kRad(Si) after only one year; however, experimental results suggest that SPENVIS may overestimate the dose rate for a geostationary orbit [48].

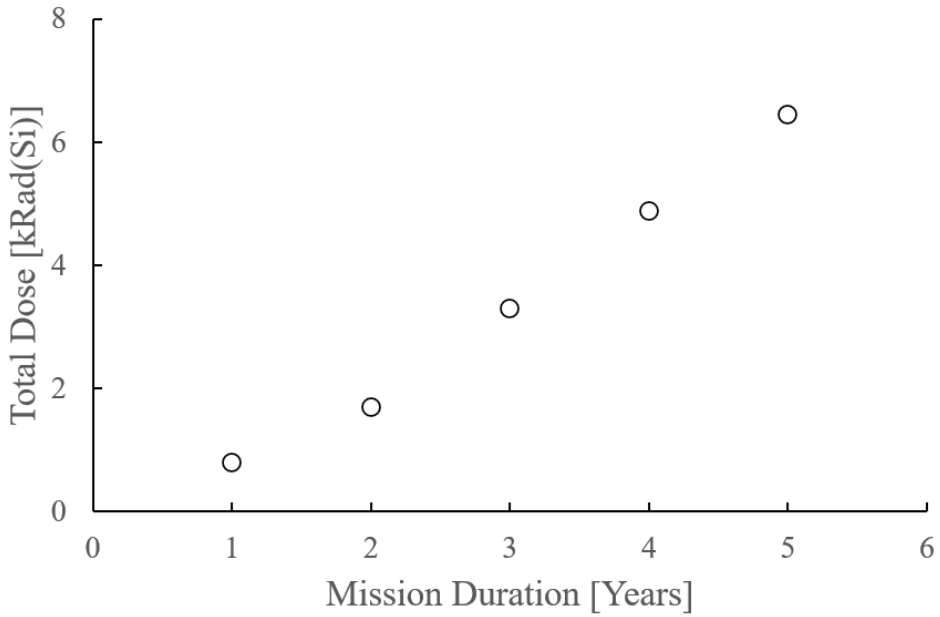

FIGURE 2.1: SPENVIS simulated total ionizing dose vs. mission duration for a 500 km LEO mission at 88° inclination behind 3 mm of aluminum shielding.

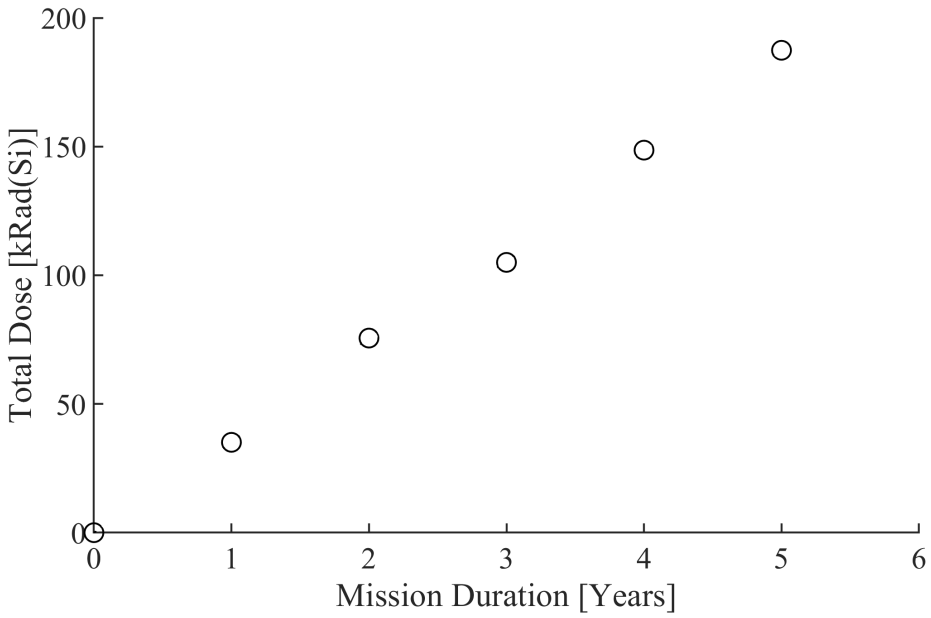

FIGURE 2.2: SPENVIS simulated total ionizing dose vs. mission duration for a GEO mission at zero inclination behind 3 mm of aluminum shielding.

The radiation in space can affect a component differently depending on the type of energized particle colliding with the spacecraft. Each heavy ion species has a specific mass associated with it, and ions with different speeds will have different kinematic energies to impart on a component during a collision due to differences in velocity. These differences contribute to how much damage is imparted on the crystal lattice comprising a component's internal circuitry. Linear energy transfer (LET) describes this damage. It is defined as the amount of energy ionized particle deposits into a material per unit distance. A higher LET, therefore, constitutes a higher energy level deposited into a component, resulting in a higher chance of damage. The mass and velocity of each heavy ion contribute to the resultant LET value and allow different elements to have the same value despite differences in mass. When considering how LET affects SEEs, it is possible to disregard the particular set of heavy ions present in the radiation environment and instead focus on the cumulative flux of particles regardless of the element type. This cumulative LET spectrum provides the flux of particles for different LET values.

The flux of each heavy ion present in the radiation environment can be simulated using SPENVIS. Each element has a different flux at different energies. A range of LET values is possible with different probabilities of colliding with a spacecraft. The cumulative LET spectrum can then be derived from the sum of each element's flux. Figure 2.3 provides the LET spectra plot for the LEO simulation I previously discussed. One can see from the plot that a significant drop off in cumulative particle flux occurs after 30 MeV-cm^2/mg. Often, components require a minimum LET threshold of 37 MeV-cm^2/mg to be classified as moderately tolerant to SEEs due to this known drop-off in flux [49]. Components are further classified as SEE immune when their minimum LET threshold lies above 100 MeV-cm^2/mg [50].

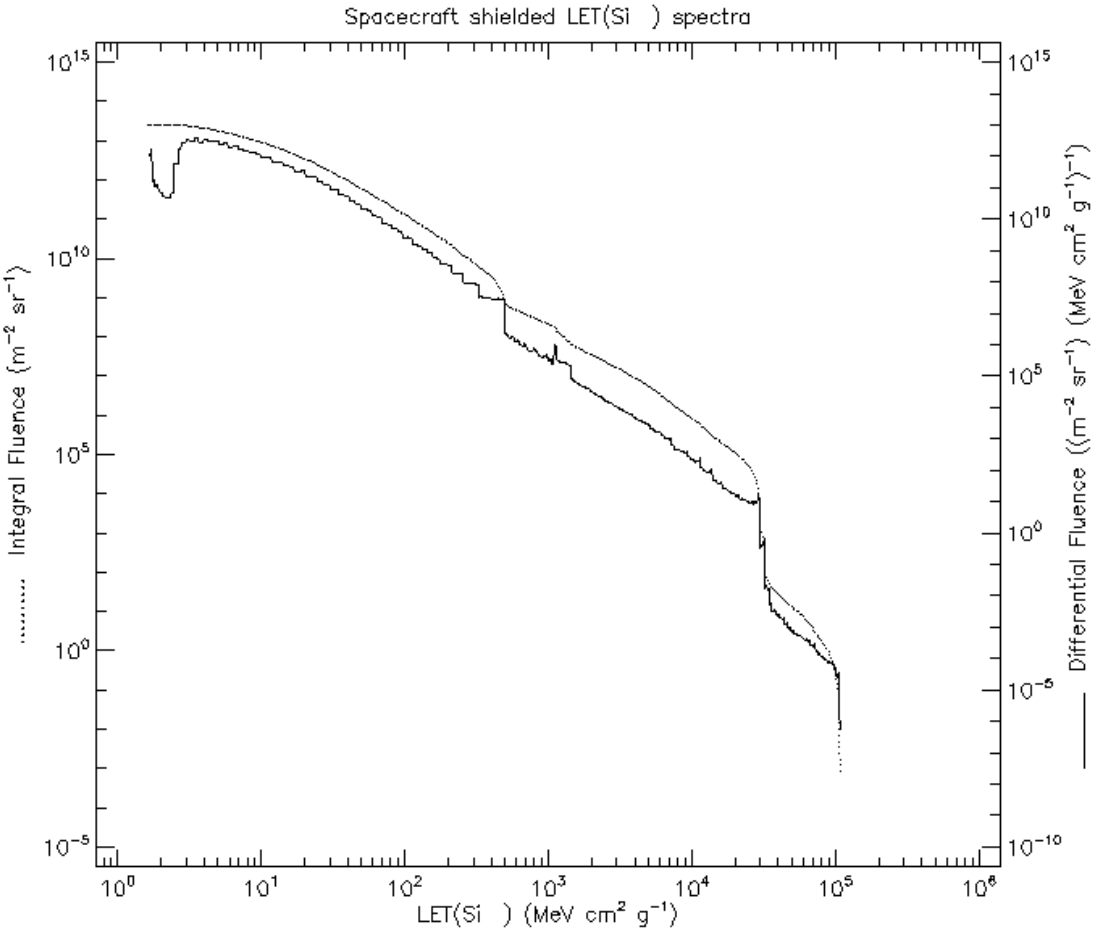

FIGURE 2.3: SPENVIS simulated LET spectra for a simulated mission operating in LEO.

2.4.2 Single Event Effects Testing

Tests that check the radiation tolerance of electronics are needed to certify the parts for use in various space environments [13]. Several tests exist to certify parts for use in the LEO regime. The tests include blasting the parts with heavy-ion beams that use one of the many energetic elements present in LEO [8]. Proton beam and neutron beam tests are also used depending on facility beam capabilities [8]. These beam tests generally check for rates of single-event effect failures.

Heavy Ion Beams

Heavy ion testing is the classic method of choice for screening single-event effects due to the higher LET that comes with using a more massive particle [8]. Heavier ions and higher energies transfer more energy into a part, and thus induce a higher likelihood of an upset occurring [46]. Heavy ion screening requires testing parts using a variety of different ions at a variety of different energy levels to determine the LET threshold for inducing upsets [46]. The higher mass of heavy ions limits the energy levels that can be obtained by the facilities that use them. Galactic cosmic rays are capable of reaching energy levels of up to 10 GeV [51] but the cyclotron institute at TAMU is only capable of providing beam energies of less than 50 MeV despite being one of the most commonly used facilities in North America. To address this limitation, testing standards require that parts be de-lidded prior to irradiation within a vacuum chamber, an expensive and risky process that involves removing a part's protective outer coating, exposing the internal circuitry inside [52].

Proton and Neutron Beams

Testing using a proton or neutron beam is another available single-event testing option. A method by E.L Peterson [53] known as the Figure of Merit (FOM) enables SEE evaluation using only a proton or neutron beam. This technique requires only one particle type and provides a characterization of the upset rate for a component in most orbits [54] for both protons and the heavy ion spectrum [55]. Using the FOM analysis technique may reduce the testing needed to produce a cost-effective screening process [55]. It may be sufficient to perform a proton beam test to characterize the single event effects [54].

Energy levels between 50 to 200 MeV are commonly used for LEO proton beam testing [55]. At 200 MeV a particle fluence of 10×10^{11} protons/cm^2 is acceptable to characterize a components SEE response [55] using the FOM technique. Other energy levels less than 200 MeV require increased particle fluence to obtain a statistically equivalent result for characterization [55]. The radiation source for such tests is usually a high-energy particle accelerator such as a cyclotron [46]. Tested components require mounting to an operating test bed to power and operate the device in either air or in a vacuum chamber [52]. The parts can be de-lidded for reduced test times [52], but this is not necessary as it is for heavy ion testing.

2.4.3 Total Ionizing Dose and Displacement Damage Testing

Other tests exist to measure total ionizing dose (TID) failure rates. These tests involve exposing a component to an ionizing radiation environment and then measuring the performance degradation over time [12]. Testing facilities will commonly employ either Cobalt-60 or Cesium-137 sources, which emit gamma rays at a relatively constant rate [12]. This testing method has validated many components for different space environments, including testing by S.M. Brylow to validate various Mars Observer Camera electronics using a Co-60 source [56]. Components will eventually fail after enough

radiation is absorbed. The amount of radiation absorbed before failure and the dose rate both play essential roles in the certification of a part [57]. Cobalt-60 sources can be used to conduct extremely low dose rate screening (ELDRS). This method of testing is needed in some electronics due to an increased risk of radiation failure when the dose rate is low.

Proton beam testing is another viable option for TID screening. This method has the additional benefit of facilitating combined TID and single-event screening campaigns, allowing a researcher to monitor the candidate parts for gradual degradation while also providing the opportunity for parts to catastrophically fail due to a single-event effects as discussed in section 2.4.2. Protons additionally test for displacement damage due to their characteristics as massive, highly penetrating particles [58]. Overall, proton testing and its ability to cover a variety of failure modes makes it a desirable choice for obtaining a more complete picture of how new commercial components may fail [58].

2.4.4 Test Facilities

Parts screening often uses a radiation test facility. Several facilities are available in Canada and the United States, providing the required energy to perform space certification testing. For example, TRIUMF (pictured in figures 2.4 and 2.5) is the premier particle accelerator in Canada [59]. The facility offers a variety of proton and neutron beams with hourly rates ranging from $850 to $950 as of this writing, depending on beam choice [60], allowing researchers to conduct both TID and SEE screening. For example, my research utilized beam line 2C (BL2C), which offers energies between 70 to 120 MeV [59]. Another particle accelerator is the TAMU cyclotron institute. The cyclotron offers various heavy-ion and proton beams capable of energies up to 60 MeV [61]. TAMU charges an hourly rate of $1130 per hour as of this writing.

FIGURE 2.4: Picture of TRIUMF taken during the first screening campaign.

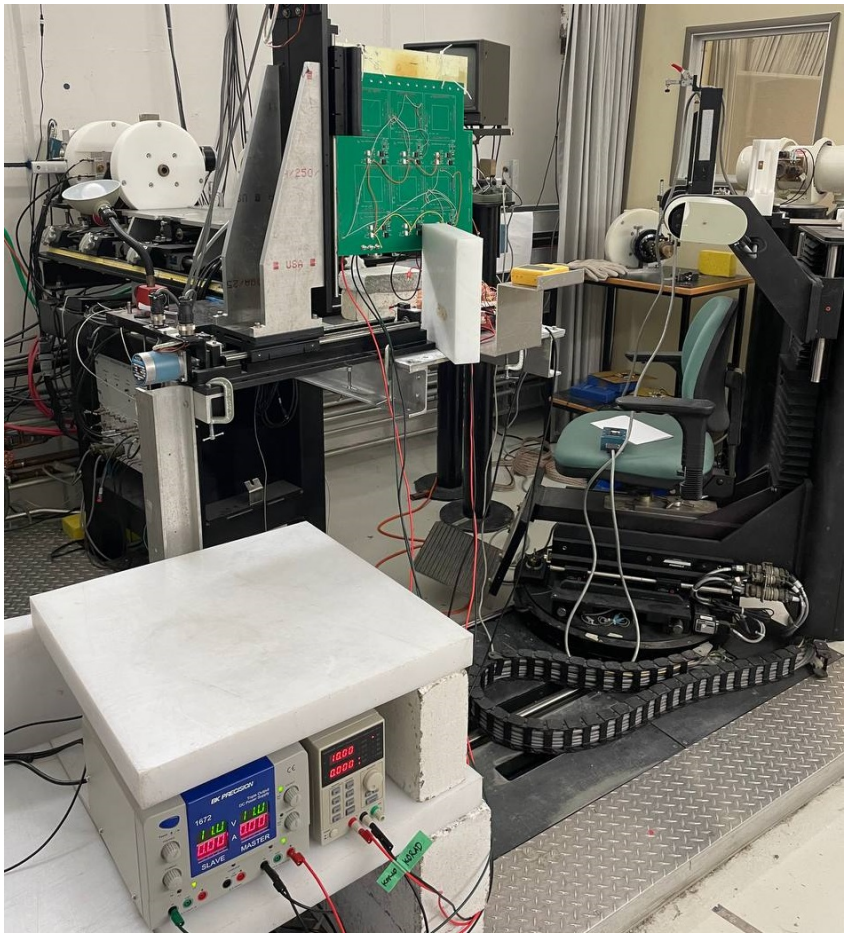

FIGURE 2.5: Picture of the test setup in the BL2C beam line room during the first screening campaign.

Other facilities, such as McMaster University or Sandia National Labs test facilities, can conduct TID screening, including ELDRS. McMaster University provides a Cobalt-60 facility, as shown in figure 2.6, which houses a 370 TBq source [62]. Robotic arms behind the one-meter thick lead and oil-lined glass windows are required to handle the screening setup [62]. Parts are exposed directly to the Cobalt-60 source and receive an accelerated radiation dose which quickly allows one to determine the dose before failure. Sandia National Labs [63] offers ELDRS testing at their Low-Dose-Rate Irradiation Facility using a Cesium-137 source. Dose rates as low as 1×10^{-6} Rad(Si)/s are

provided by the Sandia facility [63]. Facilities such as McMaster, Sandia, and the previously discussed particle accelerators offer a range of energy levels and are some of North America's most used places for radiation testing [59].

FIGURE 2.6: Picture of the Cobalt-60 test chamber at McMaster University [62].

2.4.5 Part Category Overview

Various electronic component types apply to the development of space systems and encompass two domains: analog and digital. MOSFETs and half-bridge drivers are analog components commonly used in spacecraft systems [64], while digital components such as RAM and flash memory are needed to store and process data [14]. Many manufacturers and part types are available for all components, which provides an opportunity to gather large amounts of data on each part type.

Different part types have different internal structures, which let them perform various tasks and thus require different monitoring characteristics during radiation screening. Different parts may be more susceptible to single-event effects, while other components

require testing for degradation due to the total ionizing dose. Additionally, each part family must have its standard I/O features noted so that the generic test bench platform can screen any component within a part family. This section will overview the various part families chosen for the test bench platform.

MOSFETs

Spacecraft use MOSFETs as electronic switches by controlling the input (gate) voltage. A MOSFET's standard I/O pins are the gate, source, and drain setup as shown in figure 2.7. Voltage at the MOSFET gate is toggled to overcome a specified gate threshold voltage when one wants current to flow between the source and drain. There are two types of MOSFETs: N-channel and P-channel. The N-channel MOSFETs toggle when the gate voltage changes from 0 V to some positive voltage. In contrast, the P-channel MOSFET will toggle when driven from 0 V to some negative voltage.

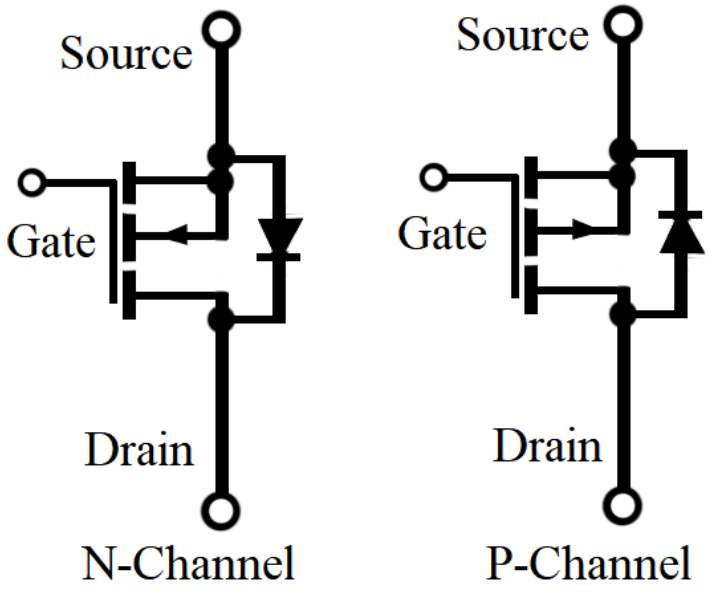

FIGURE 2.7: MOSFETs generally are composed of three I/O pins known as the Gate, Source, and Drain.

Testing conducted by J.M. Lauenstein [25] showed that MOSFETs are susceptible to single event gate rupture (SEGR) and single event burnout (SEB), failure modes which result in a catastrophic short circuiting of the component. The short circuit occurs between the drain and source pins during SEB while gate and source pins are shorted in the case of SEGR [65]. A spike in drain-source leakage current I_{DSS} or gate-leakage current I_{GSS} occurs once sufficient drain-source bias is reached to induce either failure mode, as demonstrated in testing guidelines developed by L. Scheick for the Jet Propulsion Laboratory [65].

MOSFETS also experience total ionization dose degradation in the form of a shift in gate threshold voltage [66]. Threshold voltage shift is tracked during radiation screening by performing gate stress testing at set dose increments. Gate stress testing involves gradually changing the gate voltage from zero to above the threshold while holding the drain to source voltage V_{DS} high. A sudden rise in the current response through the drain indicates the gate beginning to open. [66]

Half-Bridge Drivers

As mentioned prior, half-bridge gate drivers are a component which is often composed of MOSFETs internally. They are commonly used to flip the flow of current through an half-bridge circuit in a typical configuration shown in figure 2.8. The two MOSFETs of an external half-bridge circuit are controlled by the driver chip, enabling the circuit to effectively direct the flow of current through another component, typically a motor. These components commonly have two or more outputs labelled lower and upper gates. Lower gate outputs are driven high when the input pin is set low, while upper gates, conversely, are set high when the input is high. Table I lists the standard pins for these components.

TABLE I: Typical Half-Bridge Driver Pins

Symbol	**Name**
V_{CC}	Low-Side & Logic Supply
V_B	High-Side Supply
V_{in}	Logic Input
H_O	High-Side Driver Output
EN	Chip Enable
V_S	High-Side Supply Return
COM	Low-Side & Logic Return
L_O	Low-Side Driver Output

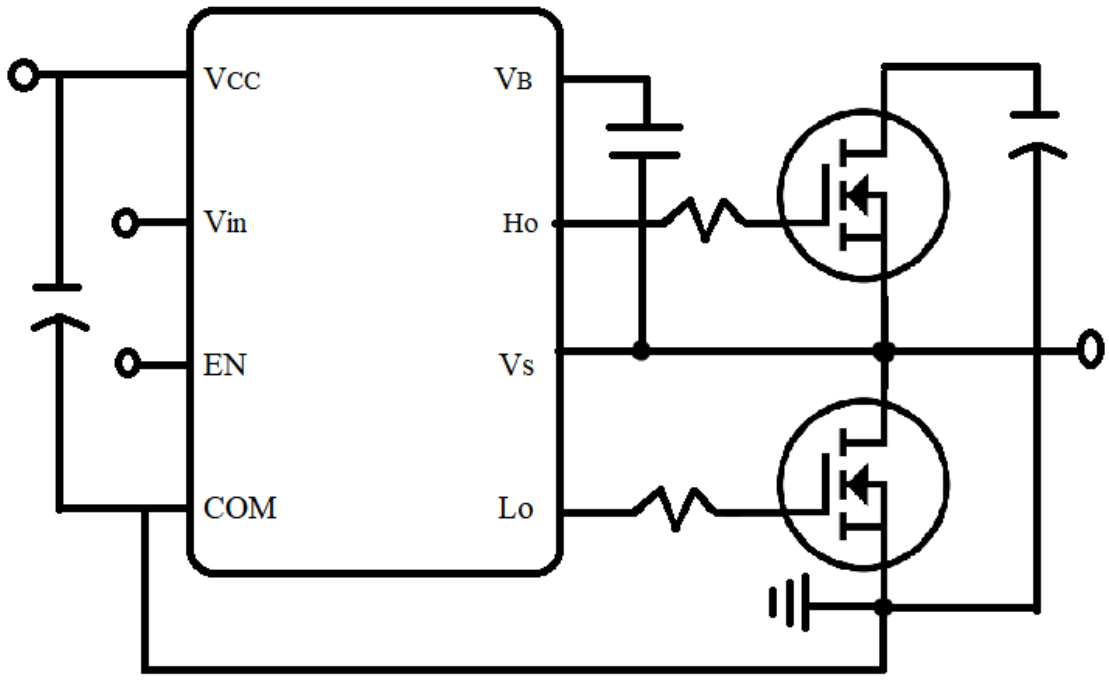

FIGURE 2.8: Typical half-bridge driver configuration.

Half-bridge drivers are susceptible to increasingly likely catastrophic failure as dose increases which manifests as one or more outputs no longer switching state in response to the input changes [67]. The outputs stuck in a high or low position can cause disastrous effects on a spacecraft mission. Examples range from a motor only being able to drive forward to an entire power bus suffering a catastrophic blow-through in the case of the driver outputs becoming stuck high, driving both MOSFETs to be on at the same time [67].

Instrumentation Amplifiers

Instrumentation amplifiers (in-amps) amplify the difference between two signals using a gain that may differ from part to part. Many common pins used by this part family can aid in generalizing a test bench design. Inputs include the voltage supply, ground, and two input voltages whose difference the device amplifies. A reference gain pin is used to set the amplifier gain, and an output pin is provided to output the amplified difference with respect to ground. Internally, an instrumentation amplifier usually consists of three operational amplifiers (op-amps) and is commonly setup according to the schematic shown in figure 2.9. The first two op-amps buffer the inputs while the third acts to produce the desired output.

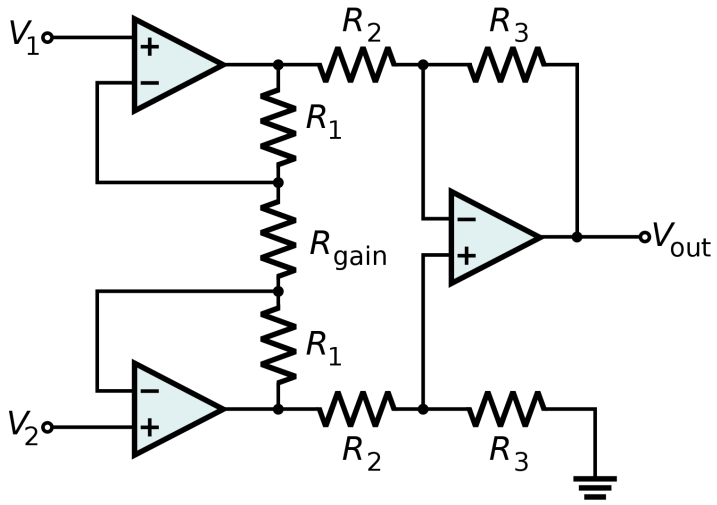

FIGURE 2.9: Typical instrumentation amplifier circuit [68].

Studies such as the one by J. Agapito and Co. [69] found the differential gain to be the main parameter to monitor during screening campaigns. The study results showed that differential gain has minimal degradation as the dose is accumulated until failure occurs. The differential gain will then suffer a sudden drop off after failure [69]. The input-offset voltage is an additional parameter of interest that affects the amplifier output. This parameter may shift by several millivolts as the dose accumulates, shifting the needed voltage across the input terminals to read zero at the output terminal and changing the output voltage [69]. Tracking failures in instrumentation amplifiers thus requires monitoring for shifts in output voltage from the desired value.

Crystal Oscillators

Crystal oscillators are commonly used on satellites to provide a consistent reference frequency for various timing circuits, such as GPS receivers [70]. These components are relatively simple and utilize three to four I/O pins. Inputs to the oscillator are the power supply and, in some cases, an enable pin. The outputs are the ground and frequency out pins.

Past commercial screening by M. Markgraf [70] indicated that current draw shift and frequency shift were both characteristics to monitor during TID screening. Oscillator frequency shifted by up to 2 ppm/kRad(Si) during the screening campaign. The current draw likewise saw shifts of up to 6%. At the same time, SEE tests done by M. Markgraf at the system level on GPS receivers did not experience failure in the oscillator itself [70]. Future screening campaigns focusing on shifts in frequency and current draw due to TID should then be the focus of a certification campaign for this part category.

Digital Memory

Commercial digital memory (RAM and Flash) is designed to store data composed of ones and zeros in the form of bits. Previous studies such as the commercial NAND flash memory screening done by T. Oldham [71] provide compelling evidence that digital memory is quite resistant to TID failure and that screening for SEEs is more critical. Previous DRAM memory screening conducted by R. Harboe-Sorensen of the European Space Agency [72] concluded that several of the 16 Mbit parts tested showed potential for space usage with significant improvement in SEE resistance over older 4 Mbit models. The main characteristic of such screening isflipped or stuck bits [72].

2.5 Chapter Two Summary

This chapter provided an overview of prior studies related to my three hypotheses. First, I provided a compelling reason for researching the feasibility of using COTS electronics in space by giving an overview of some projects that utilized COTS electronics. Next, I discussed statistical methods that aided researchers in planning their studies. I then discussed test bench platforms for stress and qualification testing, which provided insight into why a general test bench platform would be helpful for radiation effects engineering. The remainder of the chapter addressed various concepts related to radiation effects screening, including the radiation environment and how it causes component failure through total dose or single event effects. I also briefly discussed radiation simulation tools and testing facilities. Finally, I provided an overview of the part categories I studied for this thesis.

3 Existing Test Guidelines

This chapter provides an overview of industry guidelines and test methods that are commonly used to plan and conduct radiation test campaigns. First, I briefly discuss the military standards NASA uses, and the standards produced by the European Space Components Coordination (ESCC) for the ESA. Afterwards, I go into detail about various mathematical methods one uses when following the existing guidelines.

3.1 Government Standards and Guidelines

Engineers conducting radiation effects testing often follow published guidelines from government organizations such as NASA and the ESA. For example, MIL-STD-883 [73] and its associated test handbook MIL-HBD-814 [74] are commonly cited in radiation effects reports submitted to the NSREC workshop alongside ESCC standards 25100 [57], and 22900 [52]. The standards provide comprehensive test procedures for various failure modes, including those caused by radiation.

Figures 3.1 to 3.3 provide a visualization of how one could plan and conduct a radiation screening campaign according to the existing guidelines I've mentioned. Figure 3.1 first outlines the steps one follows while preparing to perform a test campaign. Afterwards, the flowcharts provided by figures 3.2 and 3.3 describe performing the test and analyzing the resultant data.

Important aspects of planning and conducting a new test campaign include determining the test parameters of a part candidate, purchasing at least ten samples and one control, determining the distribution your data is following, and choosing an appropriate design margin method (discussed in section 3.3) for testing. The ultimate purpose of the test campaign is to assign a hardness critical category (HCC) to the candidate part based on its resistance to radiation-induced failure. These hardness categories are discussed in section 3.5

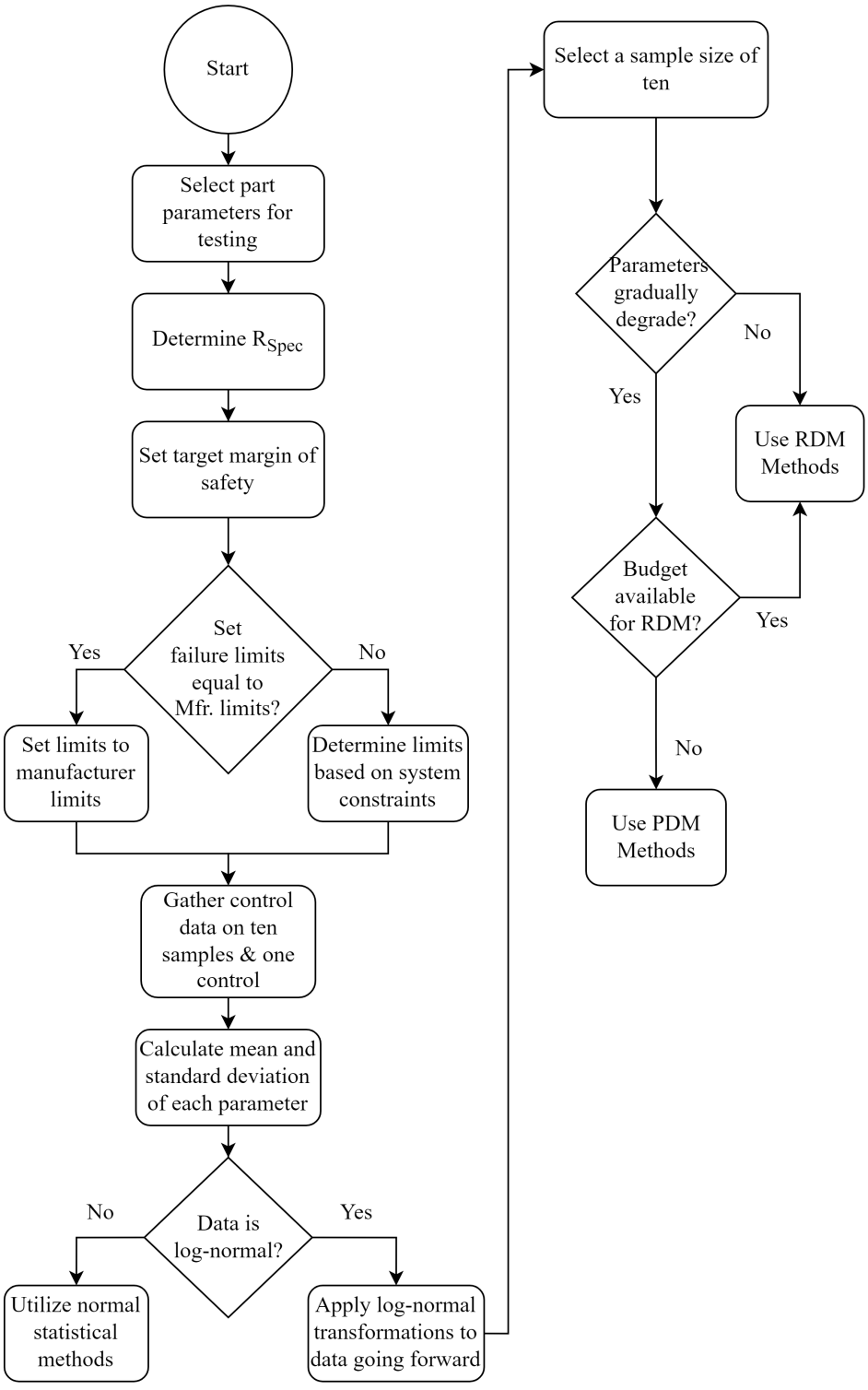

FIGURE 3.1: Screening method and sample size selection decision tree following industry guidelines.

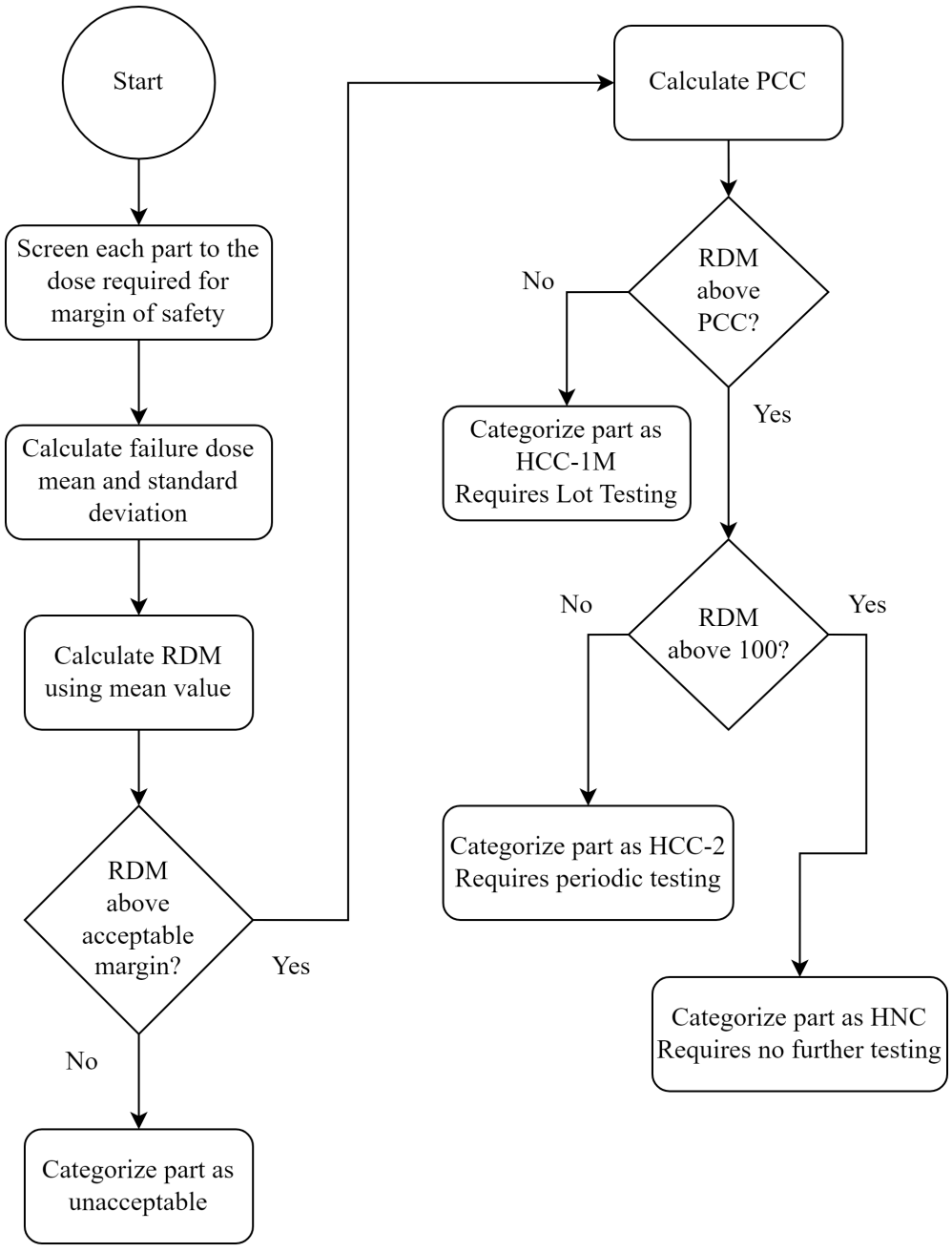

FIGURE 3.2: Radiation design margin (RDM) decision tree following industry guidelines.

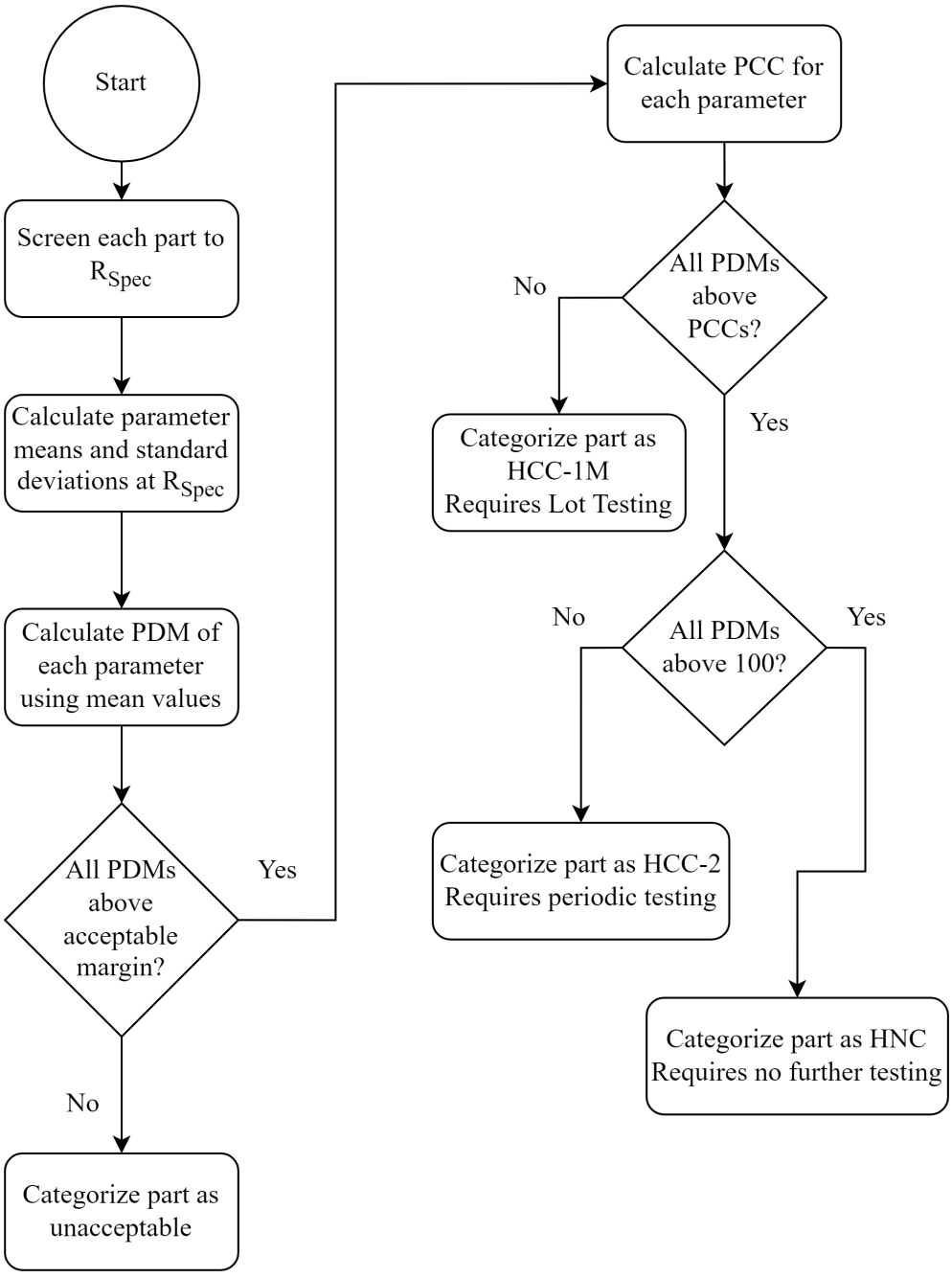

FIGURE 3.3: Parametric design margin (PDM) method decision tree following industry guidelines.

3.2 Log-Normal Distribution Statistics

According to published radiation effects guidelines [74], a common feature of the radiation-induced failure response is its tendency to follow either a normal or log-normal distribution. The log-normal distribution is defined such that the logarithm of the data for a parameter of interest follows a normal distribution. Log-normal distributions can use the same statistical methods commonly associated with normal distributions. However, applying the techniques requires data processing before performing statistical analysis. The logarithm of the data set is then used when applying statistical calculations.

In utilizing this transformation, it is useful here to define some parameters related to the log-normal distribution of the radiation test data for use throughout the remainder of this document. MIL-HBK-814 [74] provides the definitions for the various parameters. I first define the log-normal mean ($\overline{ln(x)}$), and log-normal standard deviation ($\sigma_{ln(x)}$) of any parameter with equations 3.1 and 3.2 where n is the number of samples and x_i is the i^{th} sample of a given data set.

$$\overline{ln(x)} = \frac{1}{n}\sum_{i=1}^{n} ln(x_i) \tag{3.1}$$

$$\sigma_{ln(x)} = \left(\frac{1}{n-1}\sum_{i=1}^{n}\left[ln(x_i) - \overline{ln(x)}\right]^2\right)^{1/2} \tag{3.2}$$

Using these two equations, one can substitute x_i to be defined as either the failure dose (R_{Fail}) or test parameters values (PAR_{Rad}) after accumulation of the target mission dose (R_{Spec}). Equations 3.3 through 3.6 detail the substitutions.

$$\overline{ln(R_{Fail})} = \frac{1}{n}\sum_{i=1}^{n} ln(R_{fail_i}) \tag{3.3}$$

$$\sigma_{ln(R_{Fail})} = \left(\frac{1}{n-1} \sum_{i=1}^{n} \left[ln(R_{fail_i}) - \overline{ln(R_{Fail})} \right]^2 \right)^{1/2} \tag{3.4}$$

$$\overline{ln(PAR_{Rad})} = \frac{1}{n} \sum_{i=1}^{n} ln(PAR_{Rad_i}) \tag{3.5}$$

$$\sigma_{ln(PAR_{Rad})} = \left(\frac{1}{n-1} \sum_{i=1}^{n} \left[ln(PAR_{Rad_i}) - \overline{ln(PAR_{Rad})} \right]^2 \right)^{1/2} \tag{3.6}$$

The geometric mean of the radiation-induced failure level ($\overline{R_{Fail}}$) and the geometric mean parameter value ($\overline{PAR_{Rad}}$) at R_{Spec} are obtained using the anti-log equations shown for R_{MF} and PAR_{Rad_M} in equations 3.7 and 3.8 respectively. As shown in figure 3.4, it is more appropriate to use the geometric mean when working with parameters known to follow a log-normal distribution [74], given that it more appropriately falls at the peak of probability in contrast with the more commonly used arithmetic mean, used when working with normal distributions.

$$\overline{R_{Fail}} = e^{\overline{ln(R_{Fail})}} \tag{3.7}$$

$$\overline{PAR_{Rad}} = e^{\overline{ln(PAR_{Rad})}} \tag{3.8}$$

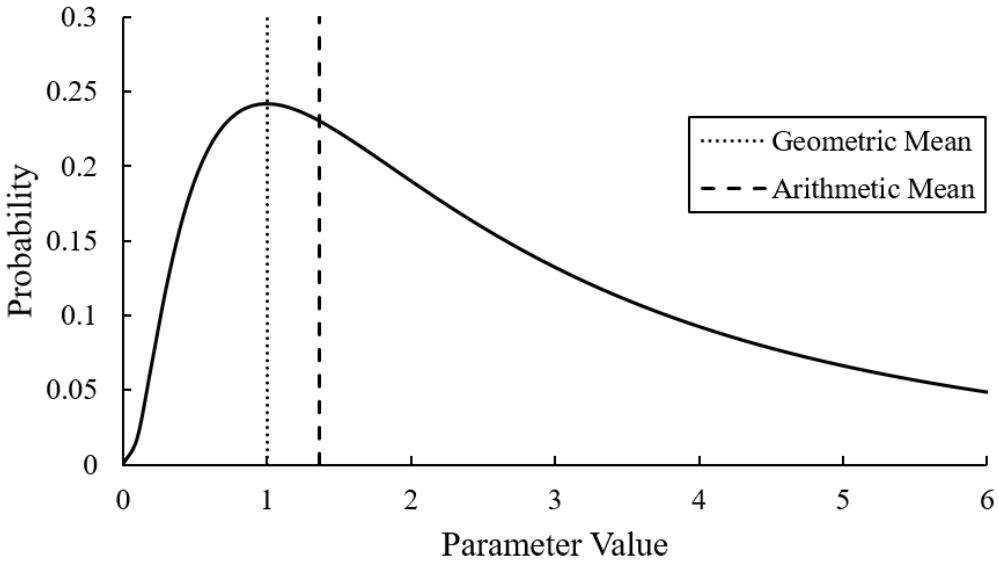

FIGURE 3.4: Log-Normal Distribution Visualization

3.3 Design Margins

Two methods to determine design margin exist. The most commonly used method defines the radiation design margin (*RDM*) as the ratio between R_{MF} to the total dose for the mission, as shown in equation 3.9 [74]. Another approach obtains a parametric design margin (*PDM*) when the critical part parameters are known to degrade with dose accumulation instead of experiencing sudden failure. The ratio uses $\overline{PAR_{Rad}}$ alongside the part parameter specification limit (PAR_{Fail}) [74]. The ratio is provided in equation 3.10 for parameter values that increase with radiation. The inverse ratio is used for values that decrease with radiation. A margin of safety of two to three is commonly selected as the target for a radiation screening campaign [74].

$$RDM = \frac{\overline{R_{Fail}}}{R_{Spec}} \tag{3.9}$$

$$PDM = \frac{PAR_{Fail}}{PAR_{Rad}} \tag{3.10}$$

An additional safety margin specifies whether lot acceptance testing is required for future part purchases after the part withstands radiation in the initial screening campaign. This factor is known as the part characterization criterion (*PCC*) [74]. Hardness assurance that eliminates lot testing is achieved when the design margin surpasses the criterion. *PCC* depends on the sample size used in the screening campaign and the log-normal standard deviation. Equation 3.11 provides the calculation of *PCC*. A one-sided tolerance limit factor (K_{TL}) is used in the equation and is variable, depending on the sample size. Tolerance limit tables for 90% confidence and 99% probability of survival are commonly employed however other values of confidence and probability are possible based on system requirements [75].

$$PCC = e^{\sigma_{ln(R_{Fail})} * K_{TL}} \tag{3.11}$$

A larger sample size and smaller standard deviation in the failure response both affect *PCC* by reducing it towards a value of one. This relationship makes it easier to eliminate the need for lot acceptance testing if a part's radiation-induced failure response shows minimal variability.

3.4 Sample Size Selection

Standards such as MIL-STD-883 [73] and ESCC 22900 [52] often recommend ten samples and one control when conducting radiation effects screening. The standards provide these sample size selections as general best practices.

3.5 Hardness Assurance Categorization

Existing test guidelines [74] provide a range of categories for radiation hardness. These hardness critical categories (HCCs) are assigned to parts after qualification screening to label what forms of testing are needed when purchasing the components in the future. Figure 3.5 visualizes the primary categorizations used during screening campaigns. Parts marked as unacceptable for use have a design margin falling under the minimum level following a qualification test. The HCC-1M category is assigned in cases where the part design margin is above the minimum acceptable limit but lower than the PCC value. Future purchases of the components require lot acceptance testing to certify flight hardware in line with standards such as MIL-STD-883 [73] or ESCC-25100 [52]. The HCC-2 category is assigned for components passing both the minimum design margin and PCC values. This category complies with the MIL-STD-883 [73] qualification requirements, and parts only require periodic testing to confirm the stability of manufacturing methods over time. The hardware non-critical (HNC) category is assigned in cases where the components design margin lies above 100 for a particular use case. These components require minimal to no testing after initial qualification.

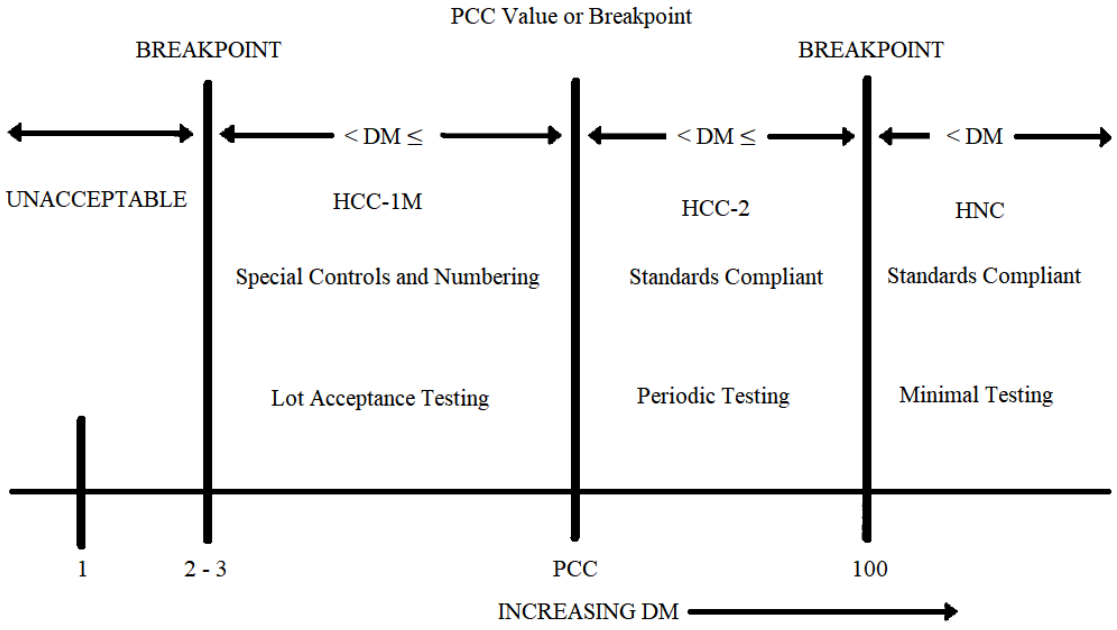

FIGURE 3.5: Hardness Critical Categories Diagram [74]

3.6 Tolerance Intervals

The interplay between sample size and standard deviation is seen through tolerance intervals. Tolerance intervals give confidence that future measurements of some parameter will fall within the given range a certain percentage of the time. The tolerance limit factor K_{TL} is used for one-sided or two-sided intervals depending on whether a parameter degrades in both directions. One-sided limits are typical due to parameters often degrading in one direction [75].

Equations 3.12 and 3.13 define upper and lower tolerance bounds using the tolerance limit factor. The variables X_{pc_L} and X_{pc_H} represent a chosen part characteristic's lower and upper bound values. The variables μ_{pc} and σ_{pc} represent the sample mean and standard deviation for a given part characteristic. Equation 3.14 describes the tolerance interval as a percentage of the mean.

$$X_{pc_H} = \mu_{pc} + K_{TL}\sigma_{pc} \tag{3.12}$$

$$X_{pc_L} = \mu_{pc} - K_{TL}\sigma_{pc} \tag{3.13}$$

$$X_{pc}^{\%} = 100 \times \frac{X_{pc}}{\mu_{pc}} \tag{3.14}$$

3.7 Paired T-Tests

Paired samples t-tests compare the means of two measurements taken from the same samples at different conditions [76]. For example, in my research, this would be the part's control data and the data obtained after irradiation. The test aims to determine whether a statistically significant difference between the two data sets is present. A t-score is calculated using equation 3.15, where μ_{Diff} represents the mean of differences between the two data sets, n represents the sample size, and σ_{Diff} is the standard deviation of the differences. The result must surpass a critical t-score value at a desired confidence value [76]. The critical t-scores used in my research were 4.604 for five samples and 3.250 for ten. I obtained these values from the standard t-score table [77] for a confidence level of 99%.

$$t = \frac{\mu_{Diff}}{\sigma_{Diff}/\sqrt{n}} \tag{3.15}$$

3.8 Chapter Three Summary

In this chapter I provided an overview of the industry test guidelines, followed by a description of the methods used when performing tests. These methods included log-normal statistics, tolerance intervals, hardness assurance categorizations, and different ways to apply design margins. I also provided flowcharts following the industry guidelines which I will later discuss modifying with new additions gained from my research in chapter 7.

4 Test Bench Architecture

In this section, I will discuss my design requirements of the test bench which were followed during development of the test bench hardware by other research colleagues. Next, I will briefly overview the design of the final revision of the test bench hardware, including its primary motherboard and the disposable device under testing (DUT) boards that slot into it. Finally, I discuss my automated analysis software, and the embedded systems which operate the test bench hardware.

4.1 Design Requirements

Three major design requirements were selected as targets for developing the test bench hardware. Each requirement is critical for ensuring the test bench is effective in facilitating future radiation effects screening campaigns for commercial electronics. The three design requirements are as follows:

4.1.1 Reconfigurable

The need for the test bench to be reconfigurable arose because this research planned to test various components across several part categories. Each part within the same category shared I/O pins; however, they had different operating levels. Reconfigurable operating ranges for the inputs to each component were also required. The test bench needed to accommodate components from various manufacturers within the same categories as this thesis considered.

4.1.2 Reusable

Reusability of the test bench was essential because it needed to test new components without requiring a redesign. Design requirements prevented the motherboard's circuitry from lying within the path of the beamline and ensured the motherboard itself would remain reusable for multiple irradiations.

4.1.3 Cost Effective

Cost-effectiveness was the final requirement that the previously discussed requirements help address. Testing various components would not be feasible if each part required a specialized testing platform. The screening campaign would have required an increased budget due to requiring multiple test bench designs to be purchased and additional beam time to be scheduled for each part category. Creating a reconfigurable and reusable testing platform would address both issues by allowing the test of every part category of interest simultaneously on the same equipment.

The test bench required the ability to use plug-and-play DUTs that housed the component types considered for this research project. At least four slots were required to minimize the testing time, employing a robotic table to reposition DUTs as needed under the beam without incurring a cooldown period. The minimized testing time further contributed to reducing the cost.

4.2 Hardware Overview

While not part of my research, my research colleagues' final design of the test bench hardware consisted of two major components, which I briefly summarize in this section for completeness: the primary motherboard and disposable DUT boards slotted into it. The hardware additionally contains two embedded systems written by myself. These

systems worked alongside my automated analysis software and provided an effective platform for conducting radiation effects screening campaigns. In the future, this test platform could be used for other test campaigns.

Motherboard

The motherboard illustrated in figure 4.1 acted as the primary module and provided the software with the tools needed to interface with the candidate parts during testing. The board had two plug-and-play slots for connecting analog DUT boards and two additional slots for digital DUT boards. Embedded systems running on PIC microcontrollers (MCUs) interpreted commands received from the analysis software through an Omega 1608G data acquisition module (DAQ). The analysis software received data from each analog component through the connected DAQ. Each analog slot accommodated the part types listed in table II.

TABLE II: Test Bench Analog Component Capabilities

Part Family	**Parts per Slot**
N-Channel MOSFET	1
P-Channel MOSFET	1
Crystal Oscillator	2
Half-Bridge Driver	2
Instrumentation Amplifier	2

The test bench handled digital DUT board operations in a different method from the analog slots. Embedded software, running on MCUs, interpreted commands received from the analysis software through USB serial connections. Each slot's MCU performed read and write operations on the candidate memory components. The test bench sent bytes

read from each part to the automated software for SEE analysis over the USB connection. Each slot could test the part categories provided in table III, including Serial Peripheral Interface (SPI) memory. The motherboard's capability to test synchronous dynamic random-access memory (SDRAM) and Secure Digital memory cards (SD cards) remained unused for my screening campaigns.

TABLE III: Test Bench Digital Component Capabilities

Part Family	**Parts per Slot**
SPI Memory	3
Quad-SPI Memory	1
SD Card	1
SDRAM	1

Two power supplies powered the test bench during testing. One supplied power to the motherboard and the various test components; another supplied drain voltages to the MOSFETs during testing. The second power supply's output for the drain voltages was controlled by my automated software through a USB serial connection.

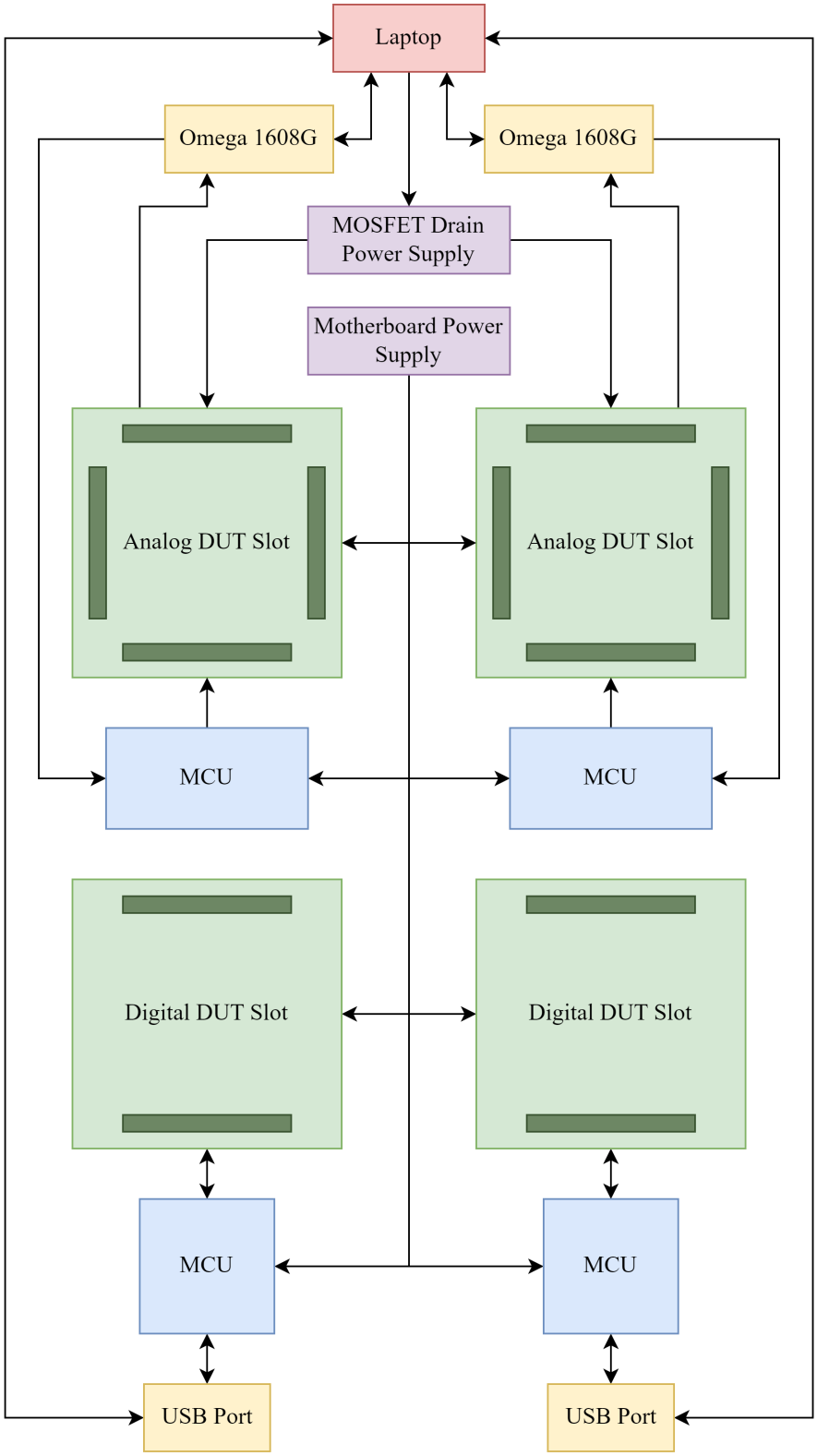

FIGURE 4.1: Motherboard Block Diagram

Disposable DUT Boards

Disposable DUT boards safeguarded the motherboard from radiation while supporting a reconfigurable setup. Analog DUT boards held the components listed in table II. My research colleagues designed two analog boards using the parts listed in table IV, described in this section for completeness.

TABLE IV: Analog DUT Board Components

Part Category	**DUT 1**	**DUT 2**
N-Channel MOSFET	BUK7M	DMNH6008
P-Channel MOSFET	BSS84A	SSM6J80x
Crystal Oscillators	7X-20 and 8W-13	AU-12 and AW-11
Half-Bridge Drivers	TC442 and TPS2822	DGD054x and ISL784x
Instrumentation Amplifiers	INA32x and AD524	LM741 and TSV911

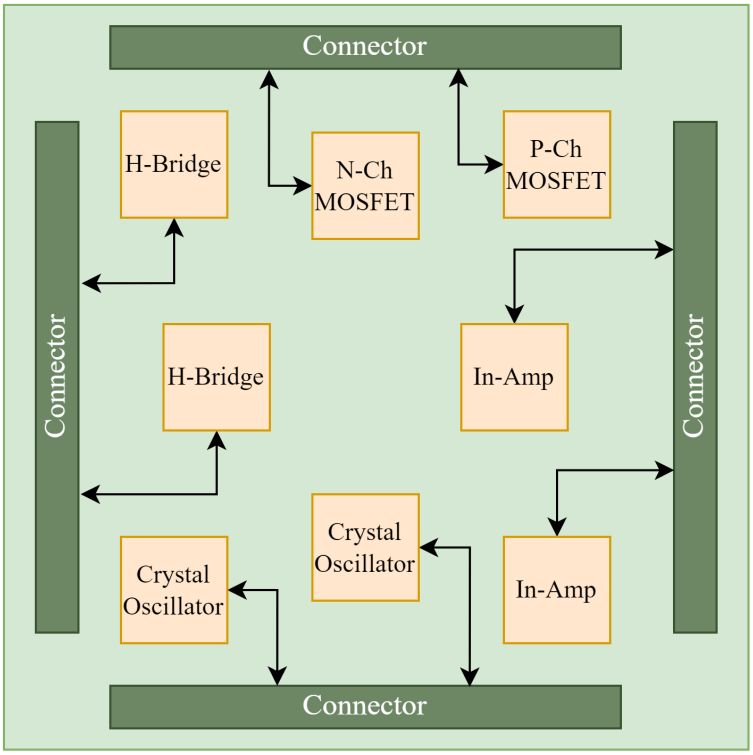

FIGURE 4.2: Analog DUT Board Block Diagram

The digital DUT boards accommodated the components provided in table III according to the block diagram shown in figure 4.3. My research colleagues designed a single DUT board for the second screening campaign using the components listed in table V.

TABLE V: Digital DUT Board Components

Part Category	**Digital DUT**	**Memory Type**
SPI Memory	MR25Hxx	MRAM
SPI Memory	MB85RS2	FRAM
SPI Memory	S25FL512	NOR Flash
Quad-SPI Memory	S25FL032	NOR Flash
SD Card	Unused	
SDRAM	Unused	

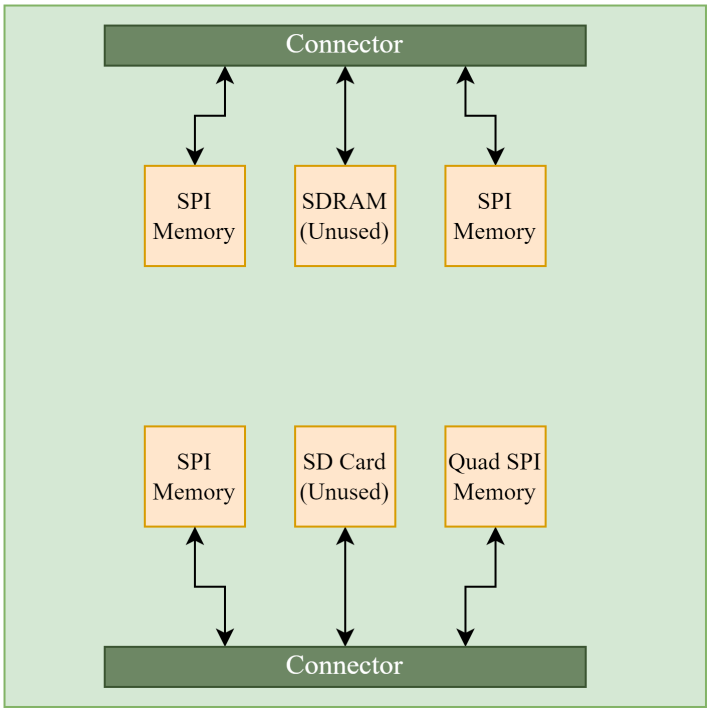

FIGURE 4.3: Digital DUT Board Block Diagram

Embedded Systems

MCUs for the analog slots interpreted commands from my automated analysis software following the logic in figure 4.4. The embedded software I wrote for the MCUs conducted either SEE or TID screening on the active analog slot depending on the signals received by my analysis software. The MCU initialized the part enable pins and kept the slot activation switch off until it received a command to begin testing. The MCU changed which MOSFET's current sensor was active depending on the mode of testing and was responsible for switching the half-bridge driver states. Finally, the MCU commanded DACs to swap the in-amps' input voltage continuously or activated MOSFET gate testing using the SPI bus connected to DAC potentiometers. Output test parameters went directly to the DAQ's analog input pins for analysis by the automated software. Listing B.6 provides my source code for the primary software loop.

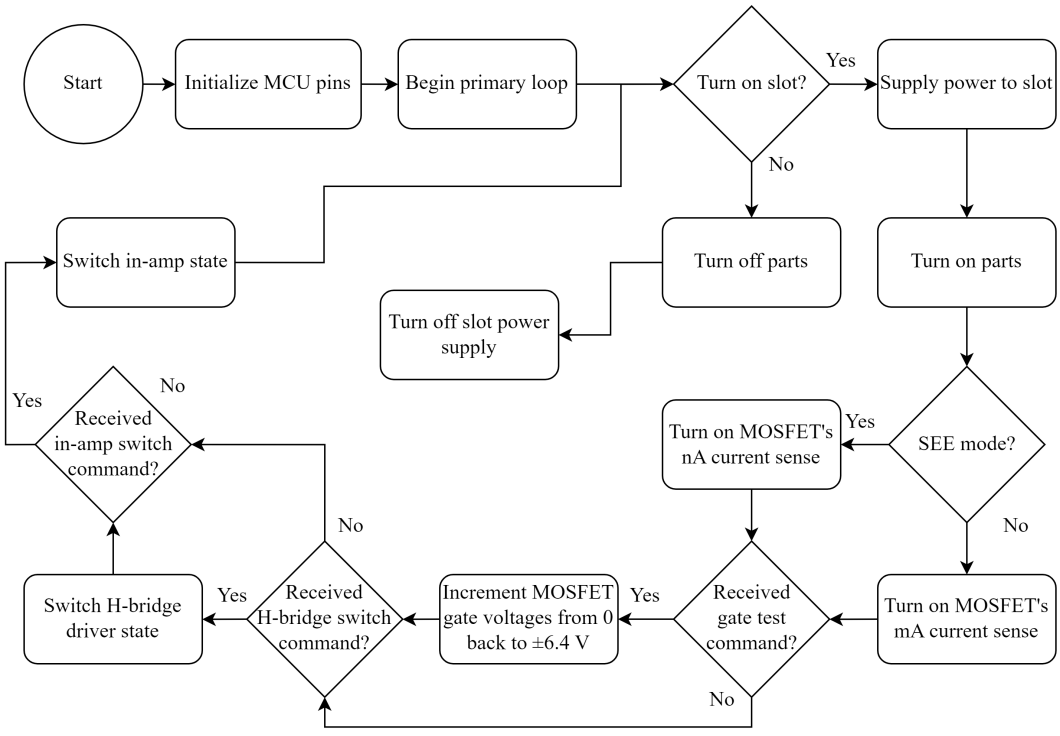

FIGURE 4.4: Analog Embedded System Flowchart

MCUs were again used to interpret commands from my automated software for the digital slots. The MCU received commands to conduct either static or dynamic memory testing through a serial USB connection between the laptop and the test bench. My embedded software running on the processor initialized and erased the memory chips on the DUT board, after which the test bench remained idle until a test command was received. Testing in either mode began upon receiving a command from the analysis software. Dynamic memory testing continuously transmitted byte data for analysis until commanded to stop. Static testing utilized a stop signal that told the test bench to read and send the data for each component after the components received the target fluence. Listing B.7 provides my source code for the primary software loop.

The MCU communicated with three of the memory components through an SPI bus. The fourth component communicated using quad-SPI. SD Card and SDRAM capability were present on the hardware but unused during my screening campaigns. Clock speeds for the digital testing were set to the highest possible value (48 MHz) that worked for every memory involved in the test campaign.

4.3 Software Overview

This section discusses the acquisition of data from the test bench and how it was analyzed by my monitoring software during the second screening campaign. I first discuss how my software interfaced with the test bench hardware. Afterwards, I provide an overview of my software's automated analysis functions for each category with flowcharts.

4.3.1 Automated Analysis Software

During the first screening campaign I saved raw test bench data from the DAQ using DasyLab and manually analyzed the data afterwards. Manually processing the large

volume of data required a significant amount of time so I developed an automated analysis program to aid me in screening parts during the second test campaign. The Python source code I wrote for the analysis software is available in listing B.1. The code used the MCC Universal Library for Windows plugin to communicate with the DAQ. The DAQ served as a bridge for communicating with the test bench during analog device testing. The embedded MCU processed command signals sent from the laptop, and the DAQ returned raw data from the parts for failure analysis. The software used multi-threading and allowed live analysis of each part category to happen in parallel. The software saved the processed data into XML files after the completion of a screening run while it also continuously saved raw test bench data into CSV files as a precaution against data loss at a sample rate of 100 Hz.

Several threads operated while testing analog components as shown in figure 4.5. The primary analog thread monitored input signals from the user interface and passed them to the test bench through the DAQ. Threads for each part category monitored the analog data received from the test bench while performing automated analysis. A visualization of the analysis software initialization is provided in figure 4.6.

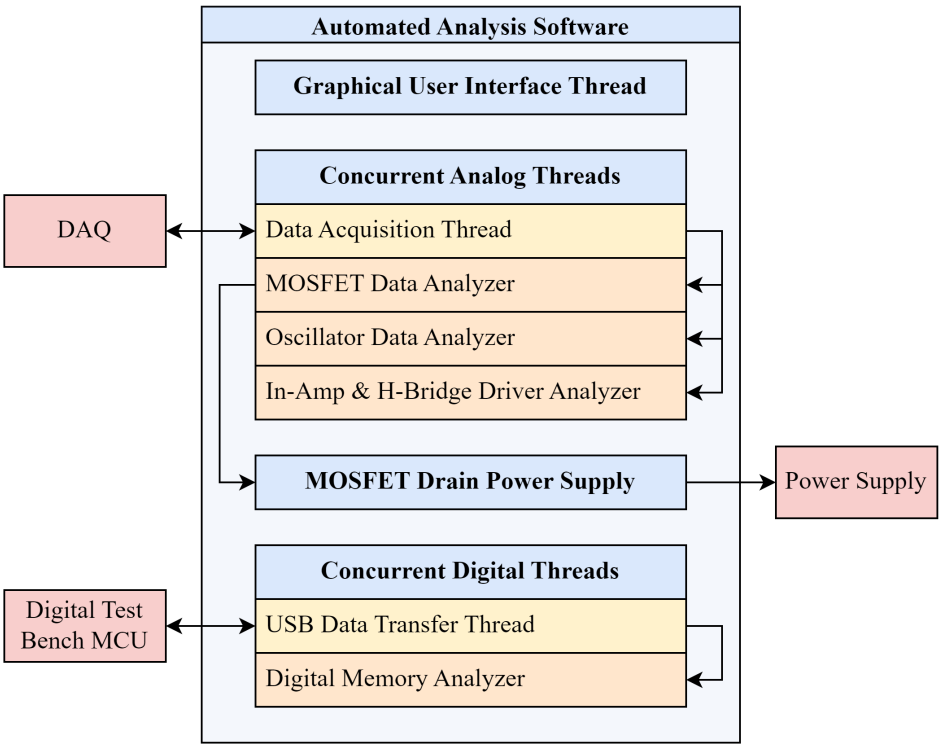

FIGURE 4.5: Automated Analysis Software Block Diagram

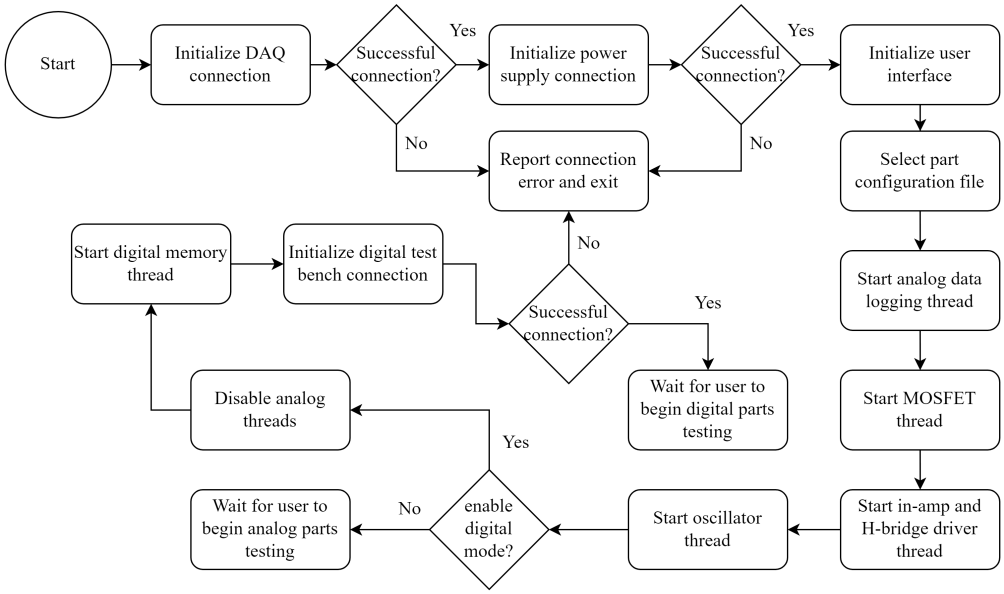

FIGURE 4.6: Automated Analysis Software Initialization Flowchart

The laptop directly controlled the digital memory slots of the test bench using a USB serial connection. No analog signals required a DAQ as an intermediary component. The analysis software sent command signals to an embedded ARM microprocessor, which then performed read and write operations on the various memory components. During read cycles, data present on each part was sent back to the analysis software to check for bit flips.

The analysis software was reconfigurable for different components through initialization files that declared the control data for each parameter of every tested part. These files enabled the test bench to act as a generic testing platform for each part category. Listings B.2 to B.4 define the part configurations used in screening campaign two. The code was compiled into an executable using PyInstaller. The code can be quickly run on other machines without the need for installing Python and the various dependencies by compiling the source code. Compiling the source code into an executable requires a specification file, as shown in listing B.5.

4.3.2 MOSFET Thread

Single Event Effect Screening

The *"FET_SEE_Processor"* function in line 830 of listing B.1 performed automated analysis of the MOSFETs during SEE screening. The process continuously monitored the incoming leakage current data and checks for values exceeding the manufacturer's limits. Next, the function incremented SEE counters for the present value of the drain voltage when an upset was detected. The function monitored gate and drain leakage currents as shown by the flowchart summarizing the process in figure 4.7.

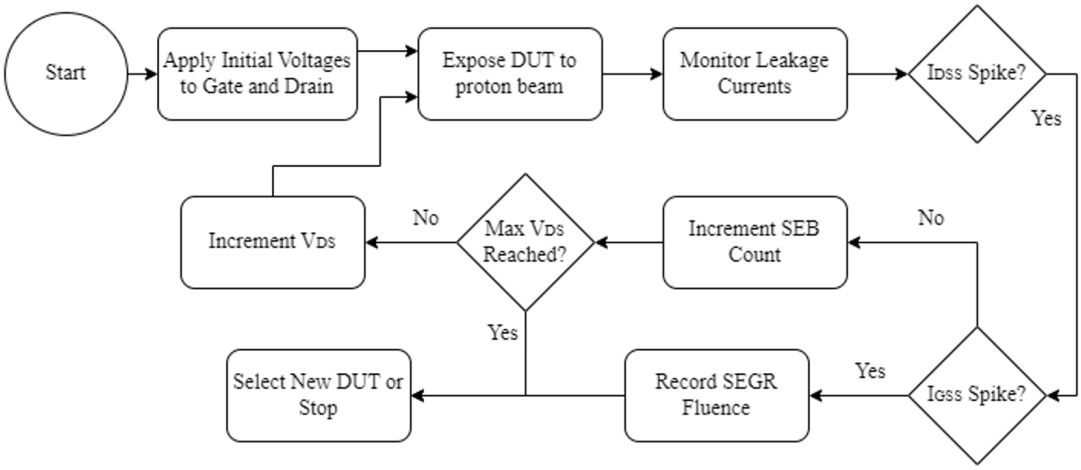

FIGURE 4.7: MOSFET Single Event Screening Flowchart

Total Ionizing Dose Screening

The *"FET_TID_Processor"* function in line 860 of listing B.1 performed automated analysis of the MOSFETs during total dose screening. The process first checked whether to perform a gate stress test. When performing this test, the function first collected the time, gate voltage, and current values for both MOSFETs in 0.2 V increments to the gate until the gate voltage increased to ± 6.4 V.

Next, the function calculated the threshold voltage of both MOSFETs by finding the point where the mean current exceeded the threshold current. The process finally plotted the current draw vs. gate voltage at a specific dose level using the Matplotlib library. Figure 4.8 presents the function's summarized data analysis flowchart.

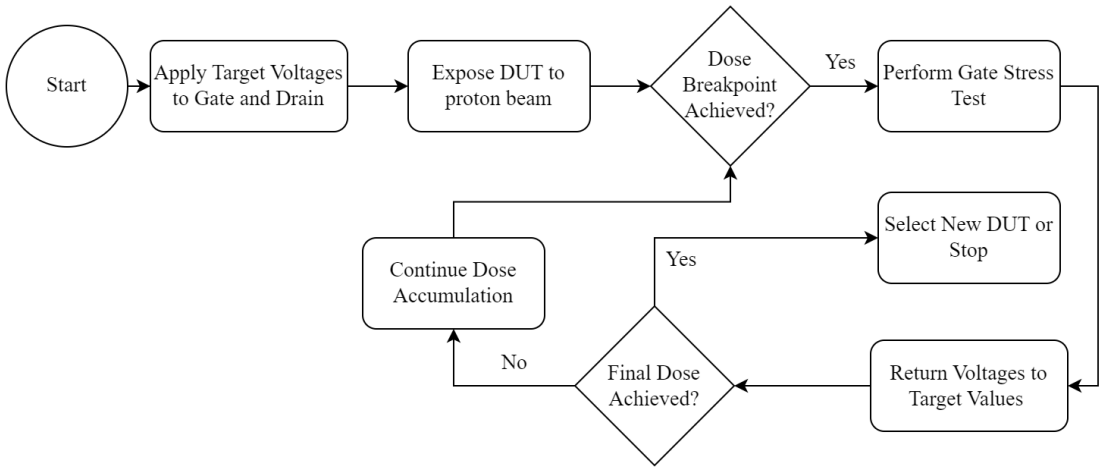

FIGURE 4.8: MOSFET Total Dose Screening Flowchart

4.3.3 Amplifier and Half-Bridge Driver Thread

Function *"AMP_TID_Processor"* monitored instrumentation amplifiers and half-bridge drivers as shown in line 770 of listing B.1. The function first commanded the test bench to set the components in a low state for ten seconds, followed by a high state, as shown in the flowchart in figure 4.9. The software logged the part as failed if its output voltage exceeded the manufacturer specified limits. The software monitored the instrumentation amplifiers for gradual degradation due to a gain or input-offset voltage shift. half-bridge drivers were checked for failure such that they failed to switch state when commanded.

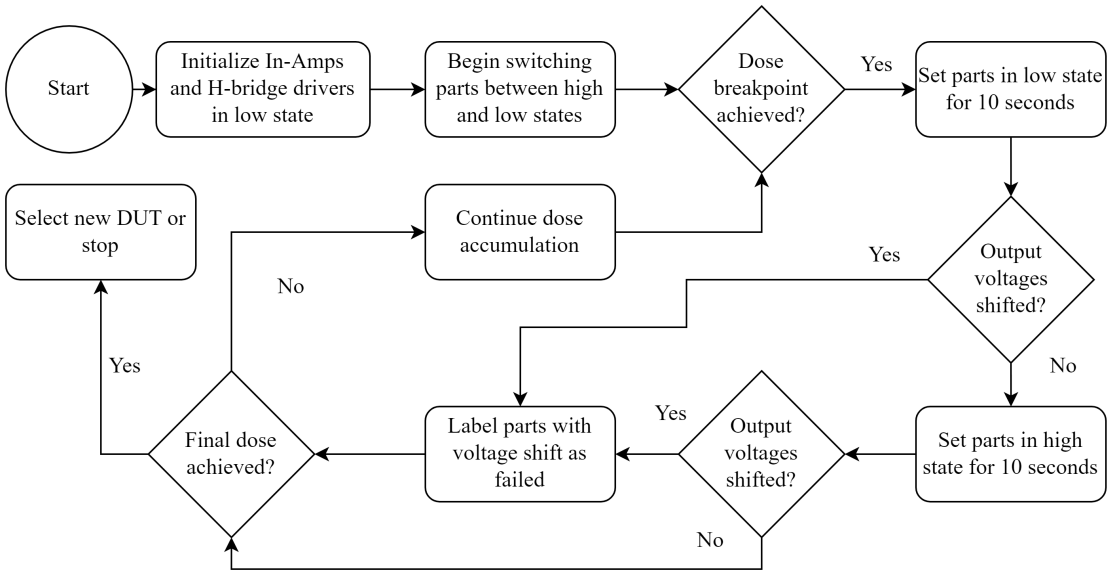

FIGURE 4.9: In-Amp and Half-Bridge Driver TID Screening Flowchart

4.3.4 Crystal Oscillator Thread

Crystal oscillator data used the DAQ's counter pins to determine the frequency by checking the count obtained in two seconds at specified dose levels. The current draw was monitored for any shifts during total dose testing. Function *"update_value_ctr"* as shown in line 715 of listing B.1 was used to analyze the data received by the test setup.

4.3.5 Digital Memory Thread

Function *"digital_analyze"* in line 1145 of listing B.1 was used by the test bench software to analyze and detect SEEs in the digital components. The raw data was passed to the software by the MCU on the test bench. A small header before the data indicated the current memory sector and selected chip. The header also included the operating mode of the test bench as a sanity check within the software. The Python struct library unpacked the bytes of data from the current sector and checked that each byte matched the 0x55 checkerboard byte. Bytes not matching the checkerboard value were logged as

upsets using a counter. Afterwards, the software calculated the upset cross-section at the target fluence using equation 4.1.

$$Cross-Section = \frac{Number\ of\ Upsets}{Particle\ Fluence\ [p/cm^2]} \tag{4.1}$$

4.4 Chapter Four Summary

This chapter discussed the test bench architecture used during my radiation effects screening campaigns. First, I overviewed the three design requirements: reconfigurability, reusability, and cost-effectiveness. The test bench's hardware consisted of four main components: the primary motherboard, two embedded systems, and two types of plug-and-play slots for DUT boards. The motherboard's slots accommodated two analog DUT boards and two digital DUT boards. Two power supplies powered the test bench during testing. After discussing the hardware, I provided an overview of the automated analysis software I developed to acquire and analyze data from the test bench. The software used multi-threading and interfaced with the test bench hardware through a DAQ or USB serial connection. The software was reconfigurable for different components through initialization files. I provided flowcharts that summarized the analysis functions for each part category.

5 Screening Campaign Methodology

This chapter will first provide an overview of the process leading to the selection of the part categories discussed earlier in section 2.4.5, followed by a list of the specific parts recommended for testing by Magellan Aerospace and the reasoning for their selection. The next discussion covers how SPENVIS simulations determined the target radiation dose margin of safety for the screening campaigns. An overview of the tested part parameters is provided, followed by an overview of each parameter's testing method. Next, I discuss how I gathered control data prior to testing to verify nominal test bench operations. Finally, I describe selecting sample sizes for the first screening campaign based on existing industry guidelines and discuss the feasibility of reducing sample size in the second screening round due to minimal part-to-part variance in tested electronics.

5.1 Part Categories

My first hypothesis predicts that using commercial electronics in LEO is feasible. To address this hypothesis, I began my research by investigating what part categories I could gather data for that would best address this prediction. I read the test data published in the NSREC workshop ([78], [79], [80], [81]) to understand better what parts are of interest to the broader space electronics industry. The data from the NSREC workshop showed that two domains requiring different test requirements existed: digital and analog parts. Digital components include flash memory, random access memory (RAM), and central processing units (CPUs). Analog parts include MOSFETs, gate drivers, amplifiers, crystal oscillators, and digital-analog converters (DACs).

I compiled this list and spoke with my industry advisor David Ross to determine which part categories would be feasible to test given that one of the requirements is developing a new generic testing platform that would address hypothesis three. CPUs and DACs were not considered due to concerns about the complexity needed to properly address these part categories.

The remaining part categories were further constrained to focus on automotive AEC-Q100 [82] graded components where possible. Parts manufactured at this grade employ stricter production methods and are certified to operate within a temperature range of -40°C to 150°C [82]. Stress tests including power temperature cycling, moisture sensitivity testing, and early life failure rate screening are conducted on the parts during production to ensure strict lot uniformity and operating temperature compliance. A single failure in any test results in discarding the entire production run [82].

A selection of parts relevant to Magellan Aerospace was provided prior to each campaign. Tables VI and VII list the selected parts. Many of the selected parts are automotive grade however the 2020 global semiconductor shortage resulted in the selection of some non-automotive grade alternatives. Some Components listed in table VI were unable to be tested during the first screening campaign due to several design issues present in the test bench hardware. These issues were later addressed and fixed in time for the second screening campaign.

TABLE VI: Parts Screened During Campaign One

Part Name	**Part Family**	**Automotive Grade**	**Tested**
BUK7M	N-Channel MOSFET	✓	✓
300NB06	N-Channel MOSFET	✓	✓
RSQ015	P-Channel MOSFET	✓	×
7X-20	Crystal Oscillator	×	✓
8W-13	Crystal Oscillator	×	✓
TC442	Half-Bridge Driver	×	✓
LTC706x	Half-Bridge Driver	✓	×
L99xx	Half-Bridge Driver	✓	×
INA333	Instrumentation Amplifier	✓	×
AD524	Instrumentation Amplifier	×	×

TABLE VII: Parts Screened During Campaign Two

Part Name	**Part Family**	**Automotive Grade**
BUK7M	N-Channel MOSFFET	✓
DMNH6008	N-Channel MOSFFET	✓
BSS84A	P-Channel MOSFET	✓
SSM6J80x	P-Channel MOSFET	✓
7X-20	Crystal Oscillator	×
8W-13	Crystal Oscillator	×
AU-12	Crystal Oscillator	×
AW-11	Crystal Oscillator	×
TC442	Half-Bridge Driver	×
TPS2822	Half-Bridge Driver	✓
DGD054x	Half-Bridge Driver	×
ISL784x	Half-Bridge Driver	✓
INA32x	Instrumentation Amplifier	×
AD524	Instrumentation Amplifier	×
LM741	Instrumentation Amplifier	×
TSV911	Instrumentation Amplifier	✓
MR25Hxx	Magnetoresistive RAM	✓
MB85RS2	Ferroelectric RAM	✓
S25FL512	SPI NOR Flash	✓
S25FL032	Quad-SPI NOR Flash	✓

5.2 Target Fluence and Total Dose

In this section, I provide an overview of the SPENVIS simulations I performed to aid in selecting a target level of total ionizing dose R_{Spec} for the screening campaigns. Additionally, I discuss choosing a fluence level to provide statistically significant proton upset rates. Finally, I discuss using the figure of merit [55] method to estimate heavy ion upset rates.

To conduct the screening campaigns, I first needed to determine what dose requirements would represent the LEO environment throughout a five-year mission. I used SPENVIS simulations to understand what level of total ionizing dose would be required. I simulated a representation of hypothetical mission environments for a CubeSat. Table VIII provides details on the two simulated trajectories. The simulated polar orbit is similar to that of high-speed satellite internet constellations like Starlink. In contrast, the other simulated orbit uses the orbital parameters of the International Space Station (ISS) because the station acts as the launch platform for many research CubeSats, including Iris.

TABLE VIII: Simulated Orbits in SPENVIS

Orbit Type	**Altitude [km]**	**Inclination**	**Dose [kRad(Si)]**
Polar	500	88°	6.4
ISS	422	52°	1.8

The SHIELDOSE-2 module simulated the total ionizing dose at various mission durations ranging from one to five years. Three millimetres of aluminum shielding was used for total dose calculations, the same structural thickness present on the Iris CubeSat. Figure 5.1 shows a trend of increasing total dose as missions progress. After five years in polar orbit, a total of 6.4 kRad(Si) of radiation is accumulated. The ISS orbit gains a smaller quantity of approximately 1.8 kRad(Si). Based on these results, I selected 25

kRad(Si) as the total dose screening requirement, allowing me to screen parts with a radiation design margin of at least 3.8.

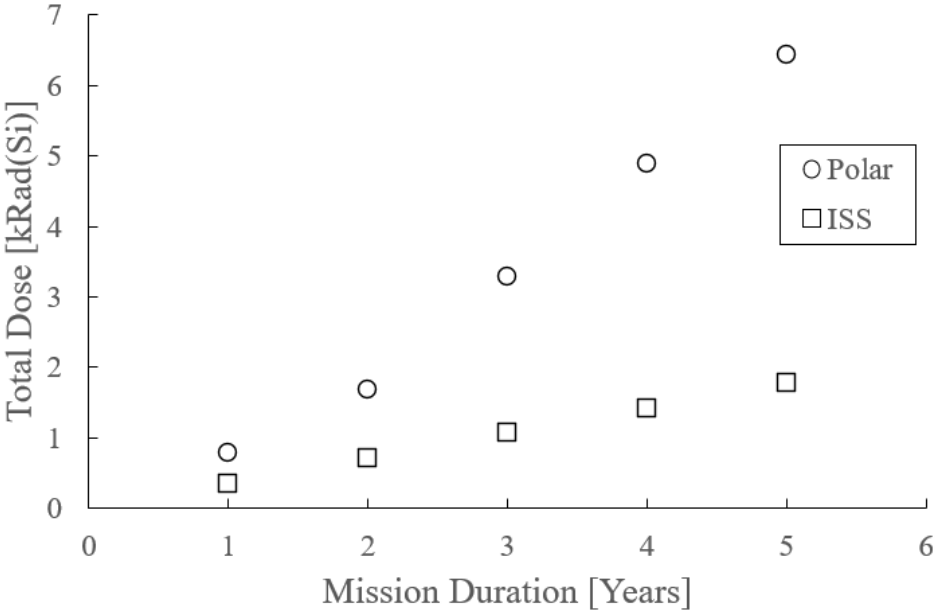

FIGURE 5.1: SPENVIS simulated total ionizing dose vs. mission duration for both simulated orbits and 3 mm aluminum shielding.

Test fluence for single event testing depends on the energy level of the beam when using protons to irradiate the component [55]. Research published by D. Hiemstra [55] established a test fluence of 10^{10} p/cm^2 is sufficient for LEO qualification testing when using 200 MeV protons. My screening campaigns increased the fluence level to 1.7×10^{10} p/cm^2 based on the scale figure provided for 100 MeV proton beams [55]. The proton beam is effective for testing with a LET threshold of 8 MeVcm^2/mg [83]. This LET threshold is less than the drop-off threshold of 37 MeV-cm^2/mg [49] for classification as moderately tolerant to SEEs as visualized in figure 5.2. However, the figure of merit method can be used to estimate SEE resistance at higher LET values in cases where the parts are not proton SEE immune [53]. Components showing immunity to proton-induced upsets would require heavy ion testing to determine the LET threshold accurately.

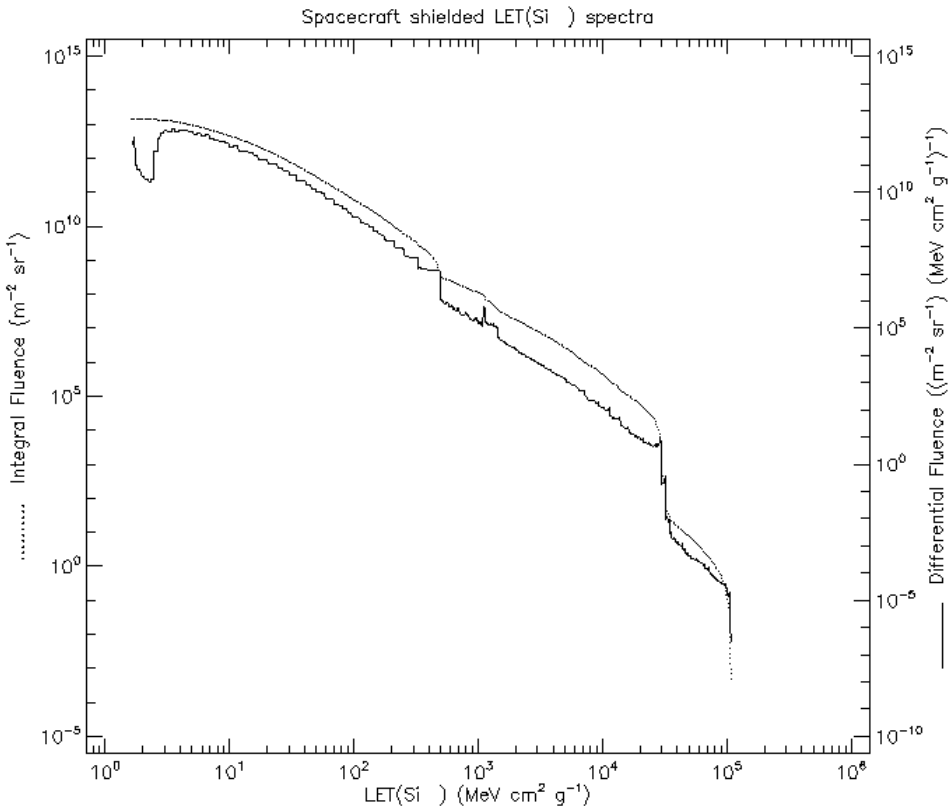

FIGURE 5.2: SPENVIS simulated LET spectra for the ISS orbit.

5.3 Part Parameters and Screening Methods

Section 2.4.5 provided an initial overview of the function and failure modes of each part category, and section 4.3 explained how my software gathered data on the test parameters during the second test campaign. This section provides a more detailed discussion of the test parameters. It outlines the methods I used to monitor each characteristic that may lead to radiation-induced failure during my screening campaigns. Standard features and failure modes are present within each part category, which enabled me to develop a generalized screening strategy that addressed hypotheses two and three. Analog DUT boards were first screened to a fluence of 1.7×10^{10} p/cm^2 using a beam flux of 8×10^6 p/cm^2/s to facilitate screening the MOSFETs for SEEs. This fluence was equivalent to about one kRad(Si) total dose. Afterwards, I irradiated the analog DUT boards at 9 Rad(Si)/s until a total dose of 25 kRad(Si) was achieved for TID screening. The digital DUT boards were screened to a fluence of 1.7×10^{10} p/cm^2 using a beam flux of 8×10^6 p/cm^2/s to facilitate SEE screening. A visualization of the screening campaign test strategy is provided in figure 5.3.

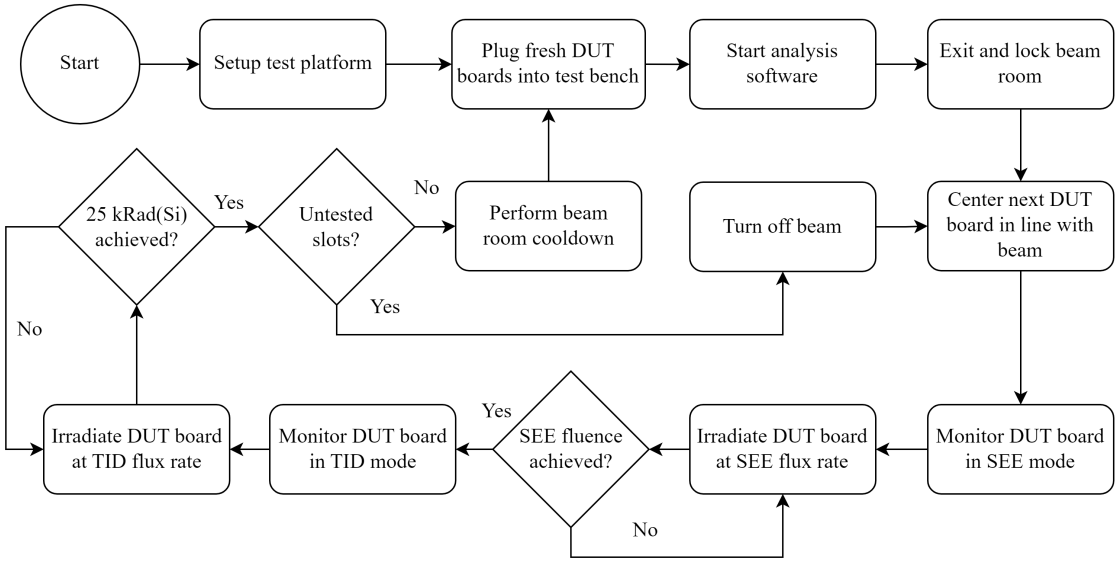

FIGURE 5.3: Flowchart followed for the screening campaigns.

5.3.1 MOSFET Parameters

The MOSFETs were first screened for SEE-induced failure. Surviving parts were tested afterwards for TID-induced failure. Each test monitored different parameters, as described below:

Single Event Effect Screening

During SEE screening, the test bench monitored both drain-source leakage current (I_{DSS}) and gate-source leakage current (I_{GSS}), while the MOSFETs were biased in the off-state. Each N-channel MOSFET had its gate-source voltage (V_{GS}) held at 0 V with the drain-source voltage (V_{DS}) initially set to 0 V. The V_{DS} was incremented by 3 V every 1.7×10^9 p/cm^2 up to a maximum V_{DS} of 30 V. A total dose of 1 kRad(Si) was accumulated during SEE testing, leaving 24 kRad(Si) remaining for TID screening. P-channel MOSFETs followed the same test procedure but instead used negative voltages. The gate leakage current was monitored at each V_{DS} step for a negative increase in I_{GSS} indicative of single-event gate rupture (SEGR). Afterwards, reverse current from the source pin would flow into the digital potentiometer (which was used to step the gate voltage), thus causing attempts to change the gate voltage to fail. Monitoring I_{DSS} was required to catch current spikes due to single-event burnout (SEB). A short circuit between the drain and source pins would present itself, leading to failure to turn off the MOSFET when the gate voltage decreased. The parts failed if SEGR or SEB occurred prior to total ionizing dose screening.

Total Ionizing Dose Screening

After completing the SEE screening described above, The MOSFETs were irradiated at a rate of 9 Rad(Si)/s during TID screening while biased on with V_{GS} set to 5 V and V_{DS} set to 30 V. Gate stress testing was performed every two kRad(Si) to monitor for

a shift in the gate threshold voltage (V_{Gth}). Part failure occurred when V_{Gth} exceeded manufacturer limits.

5.3.2 Half-Bridge Driver Parameters

Half-bridge driver failure manifested in a way that caused difficulties in determining whether the failure was a result of total dose degradation or due to the internal circuitry experiencing an SEE. In either case, the half-bridge driver would lose the ability to switch when commanded. I instead performed a combined failure test which monitored for parts breaking in general. Afterwards, I determined whether the failures were due to TID by checking if the DUTs failed consistently at the same amount of total dose. The first kRad(Si) was accumulated while the MOSFETs were screened for SEEs.

Combined Failure Screening

The half-bridge drivers were monitored for failure at either output pin as the total dose accumulated. Each part switched between its two operating states every ten seconds as the test progressed during the second screening campaign. The low-side and high-side driver output voltages were fed to the DAQ by the test bench. I declared a part as failed if it ceased to switch when commanded. Overheating issues were present in the first test bench hardware revision and necessitated changing the operating times to mitigate the heat. The first screening campaign operated the drivers in an on-state for one second followed by an off-state for 14 seconds, repeating this pattern for the full duration of the test.

5.3.3 Instrumentation Amplifier Parameters

The amplifiers were monitored for total dose failure up to 25 kRad(Si). The first kRad(Si) of dose was accumulated while the test bench conducted SEE testing on the MOSFETs.

Total Ionizing Dose Screening

The test bench hardware was setup such that a common gain would be chosen for the instrumentation amplifiers. The lowest common denominator of five was the gain chosen for the screened parts. The amplifiers were monitored by the analysis software for changes in the output voltage indicative of total dose failure due to degradation of either the gain or the input-offset voltage. The amplifiers high pin voltage was repeatedly toggled every ten seconds between 0.25 V and 1 V while the low pin voltage was held at 0 V. The amplifier output was expected to toggle between 1.25 V and 5 V.

5.3.4 Crystal Oscillator Parameters

The oscillators were tested for total ionizing dose effects up to 25 kRad(Si) during the screening campaigns. The first kRad(Si) was accumulated while the MOSFETs were screened for SEEs, similarly to the other part categories.

Total Ionizing Dose Screening

The test bench monitored the crystal oscillators for frequency shift during total ionizing dose screening. The second testing campaign also checked the components for changes in the current draw. Part failure occurred when part parameters exceeded manufacturer's stated limits.

5.3.5 Digital Memory Parameters

The digital memory components were tested separately from the analog components on different pair of slots on the test bench. These components were screened for bit flips and stuck bits due to single event effects.

Single Event Effect Screening

The screening of digital memory involved performing two types of SEE monitoring. A static memory test was conducted by writing a checkerboard pattern (0x55) to every byte in the memory. The proton beam would then irradiate the parts to a fluence of 1.7e10 p/cm^2, following which the memory would be read back for comparison and the upset cross-section logged. Dynamic memory testing was also conducted by continuously writing the pattern to the memory and reading it back until the desired fluence of 1.7e10 p/cm^2 was reached. The dynamic test enabled me to screen for permanently stuck bits. The results of the memory testing against a proton beam allowed for estimates of the heavy ion upset rates using the figure of merit method when components were not immune to proton-induced SEEs.

5.4 Control Data Analysis and Sample Size Selection

In this section, I will review the guidelines used in determining the sample sizes for both screening campaigns. I will first discuss how I analyzed the control data variance of each test parameter to verify nominal test bench operations while also checking for defective parts. Next, I discuss selecting sample sizes for the first screening campaign based on test guidelines used by NASA and the ESA. Finally, I discuss reducing the sample sizes in the second screening campaign due to observations of minimal variance in the first round results.

5.4.1 First Campaign

In the first screening campaign, I calculated the variance in each part parameter I planned to test by first gathering data for each component prior to radiation testing. Automotive

grade certification requires minimal lot variation, and I expected the control data to reflect that. I needed quantifiable data showing that modern electronics have minimal variation to use this property in developing additions to the screening plan guidelines. Table IX lists the standard deviation (σ_{pc}) and tolerance interval width as a percentage of the mean (μ_{pc}) for each tested part. This data represents part-to-part variation and the tolerance interval widths were calculated according to the methods provided in section 3.6. Tolerance interval calculations were done using a confidence of 90% and a probability of 99%. The data shows that the 300NB06 MOSFET has the largest variance among the selected parts.

TABLE IX: Part Variance in the First Screening Campaign

Part	**Most Variable Parameter**	μ_{pc}	σ_{pc}	**Tolerance Interval**
BUK7M	Gate Threshold [V]	3.12	7.02×10^{-2}	7.94%
300NB06	Gate Threshold [V]	2.74	1.24×10^{-1}	16%
7X-20	Frequency [Hz]	19999749	97	0.001%
8W-13	Frequency [Hz]	12999925	118	0.003%
TC442	High-Side On-State [V]	8.71	4.18×10^{-2}	1.69%

I selected manufacturer limits as failure bounds in the absence of requirements developed from a finished system. The first test campaign ignored the in-amps due to excessive signal noise, while all other components exhibit nominal behaviour within the manufacturer's stated variance. For sample size selection in the first screening campaign, I used the recommended ten DUT boards and one control as mentioned in section 3.4. I also procured four backup boards to mitigate risks of damage from either test bench verification tests or from shipping to TRIUMF.

5.4.2 Second Campaign

I used the data obtained from the first campaign to help plan the sample sizes for the second campaign. This round of screening needed to balance the available beam time so

I could equally test the three different DUT boards. Reduction in the total dose imparted onto the components would be necessary if testing required ten of each DUT board to be used (as discussed in chapter 3), compromising the total dose requirements.

I analyzed the data from the first screening campaign, checking the standard deviation and calculating the tolerance interval width of each component's failure dose. The tolerance interval was calculated around either R_{Fail} or PAR_{Rad} depending on which design margin was used using the methods provided in section 3.6. Next, I recalculated the results at a smaller sample size of five. I performed the five sample calculations several times, randomly selecting a different set each time. Each part showed similar tolerance margins at five samples compared with the initial ten sample data set, regardless of which random selection was chosen.

Figures 5.4 through 5.6 provide visualizations of the results for several part parameters. The reduction in sample size increased the tolerance bounds to accommodate the desired confidence and failure probability of 90% and 99% respectively; however, the minimal part variation obtained from using modern components resulted in the sample size playing an equally minimal role in the tolerance bound width. The tolerance bound width increased less than 10% in the worst-case. I deemed this an acceptable increase in failure tolerance bounds and used five samples and one control during the second screening campaign. Two additional backup boards were procured to once again mitigate risks of damage from either test bench verification tests or from shipping to TRIUMF.

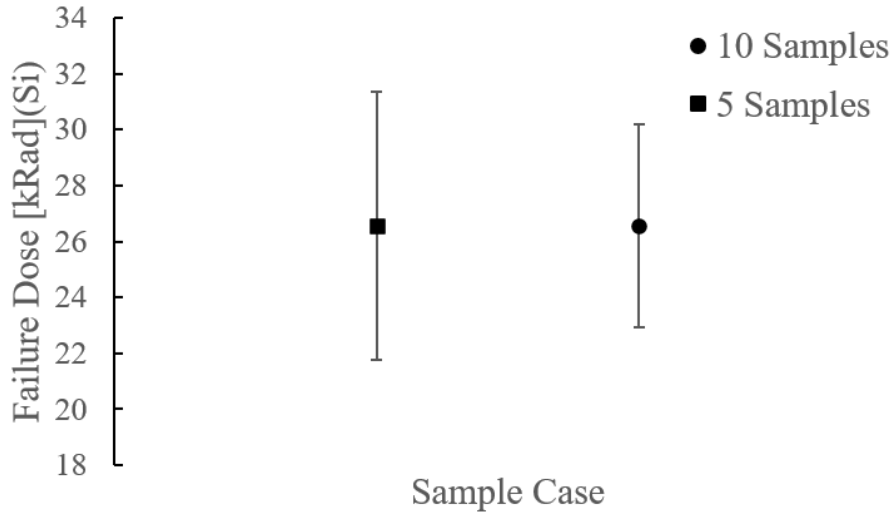

FIGURE 5.4: BUK7M TID failure tolerance intervals for five and ten samples. The five sample set is a random selection from the original ten samples.

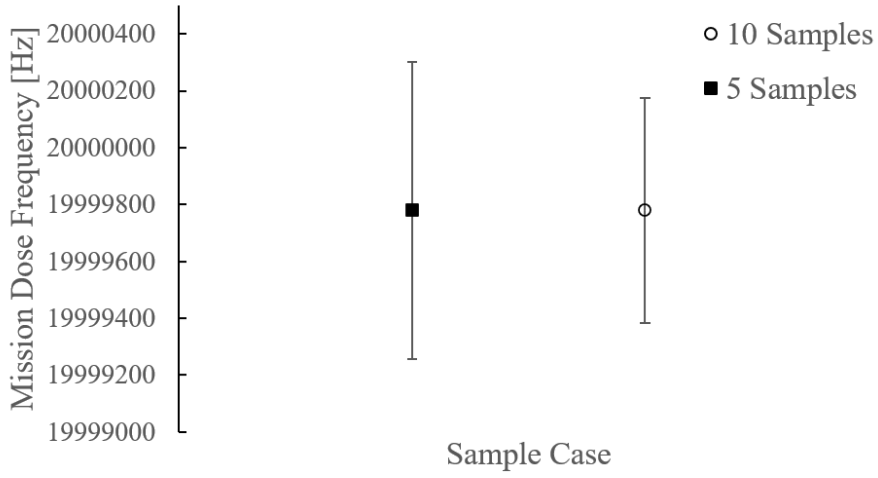

FIGURE 5.5: 7X-20 frequency shift tolerance intervals for five and ten samples. The five sample set is a random selection from the original ten samples.

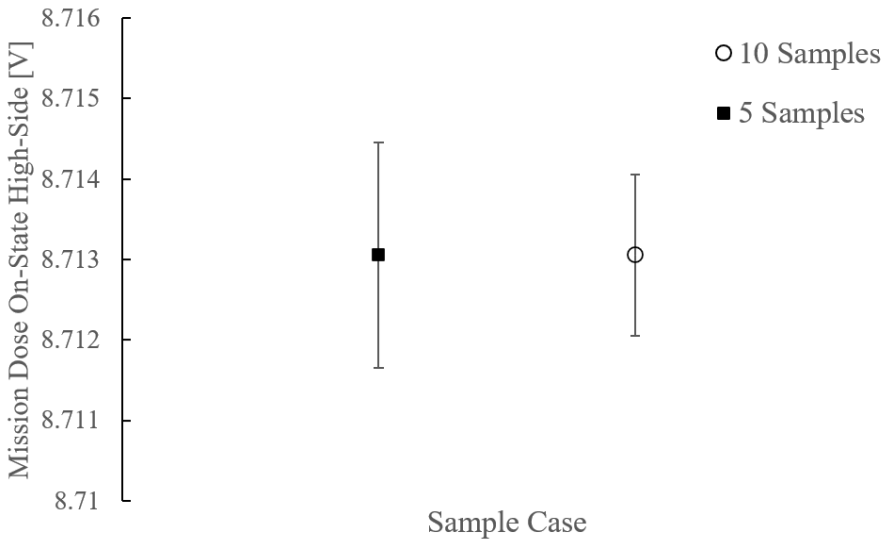

FIGURE 5.6: TC442 on-state high-side voltage shift tolerance intervals for five and ten samples. The five sample set is a random selection from the original ten samples.

5.5 Chapter Five Summary

This chapter explained how I planned and conducted my two screening campaigns. I first provided an overview of the part categories of interest to Magellan Aerospace. Afterwards, I discussed using SPENVIS to determine the target dose for the screening campaigns and provided an overview of the part parameters I used for testing. Next, I discussed verifying that the components were operating as expected before irradiation. Finally, I described reducing the sample size for my second campaign based on my first-round results exhibiting minimal part-to-part variance.

6 Test Data and Results

This chapter presents the test results of the commercials parts I irradiated across two screening campaigns. Both campaigns occurred at the TRIUMF facility in Vancouver. In both campaigns, the BL2C beam line was operated at the facility's standard test energy of 105 MeV. I brought one test bench and 15 DUT boards to the facility during the first campaign, four of which were backups, and one was a control. The second campaign instead brought eight of each type of DUT to the facility, including two backups and one control each. SEE screening used a particle flux of 8×10^6 p/cm^2/s until reaching the target fluence of 1.7×10^{10} p/cm^2. I chose the lowest flux that the beam line could provide to minimize the probability of simultaneous SEEs occurring. Afterwards, the parts were further irradiated at a dose rate of 9 Rad(Si)/s until the target dose of 25 kRad(Si) was achieved. This dose rate was the maximum value beam line BL2C could provide and was chosen to minimize the time scheduled at the facility.

In the first screening campaign, I monitored each component at the maximum sample rate the DAQ could provide (10,000 samples per second). I chose the top sampling speed to increase the likelihood of observing the shape of the SEE response prior to part failure. Later, data for TID screening were down-sampled to 1,000 samples per second using decimation to reduce file sizes because MATLAB and Excel could not process the original data files. The TID data were not compromised due to the decimation because part parameters gradually decayed across larger time scales than either sample rate. Data were automatically analyzed in the second screening campaign using my automated analysis software as described in section 4.3.

Several design issues in the first test bench design were present and addressed in time for the next revision used in the second screening campaign. Appendix A provides an overview of these issues, their impact on the first round data, and how I addressed each of them while performing analysis on the data.

6.1 MOSFETs

Three N-channel MOSFETs and two P-channel MOSFETs were screened across the two campaigns. Tables X and XI display the vendor-supplied electrical parameters for each component. The BUK7M[1] was screened in both campaigns. The BUK7M, 300NB06[1], and both P-channel MOSFETs were automotive-grade and AEC-Q101 certified. I monitored each component for SEGR, SEB and gate threshold shift following the methods described in section 5.3.1. Pre-radiation measurements were obtained for each sample before the screening campaigns for control purposes, verifying that the components operated within the expected parameters. The pre-radiation I_{DS} vs. V_{GS} curves are shown in figures 6.1 through 6.6.

Voltage regulator heating issues in the first test bench design reduced the maximum voltage supply to the MOSFET circuit from 5 V to 4 V. Test data for the BUK7M, and 300NB06 were not compromised, given that the threshold voltages of the MOSFETs were still below the maximum gate voltage.

[1]This section reuses content from my paper published in the REDW record [35] with permission from NSREC. See appendix C for permissions.

During the second round, I temporarily lost the remote desktop connection to the test bench computer, missing the 20 kRad(Si) and 22 kRad(Si) gate tests on the first sample of the second DUT board. The 24 kRad(Si) and 25 kRad(Si) gate tests could not be obtained for the third sample of the second DUT board due to a beam line issue which suspended testing during the second day at TRIUMF. The DMNH6008 and BSS84A MOSFETs had their calculations at these breakpoints adjusted accordingly to account for the loss of these gate tests.

TABLE X: N-Channel MOSFET Electrical Parameters ([84], [85], [86])

Symbol	**Parameter**	**BUK7M**	**300NB06**	**DMNH6008**
V_{DS}	Drain-Source Voltage	0 to 40 V	0 to 60 V	0 to 60 V
V_{Th}	Gate Threshold Voltage	2.4 to 3.6 V	1.8 to 3.8 V	2 to 4 V
I_{DSS}	Drain Leakage Current	0.01 to 1 µA	0.01 to 1 µA	0.01 to 1 µA
I_{GSS}	Gate Leakage Current	2 to 100 nA	2 to 100 nA	0 to 100 nA

TABLE XI: P-Channel MOSFET Electrical Parameters ([87], [88])

Symbol	**Parameter**	**BSS84A**	**SSM6J80x**
V_{DS}	Drain-Source Voltage	-60 to 0 V	-40 to 0 V
V_{Th}	Gate Threshold Voltage	-2.5 to -1.0 V	-2 to -0.8 V
I_{DSS}	Drain Leakage Current	-1 to 0 µA	-10 to 0 µA
I_{GSS}	Gate Leakage Current	-10 to 10 nA	-100 to 100 nA

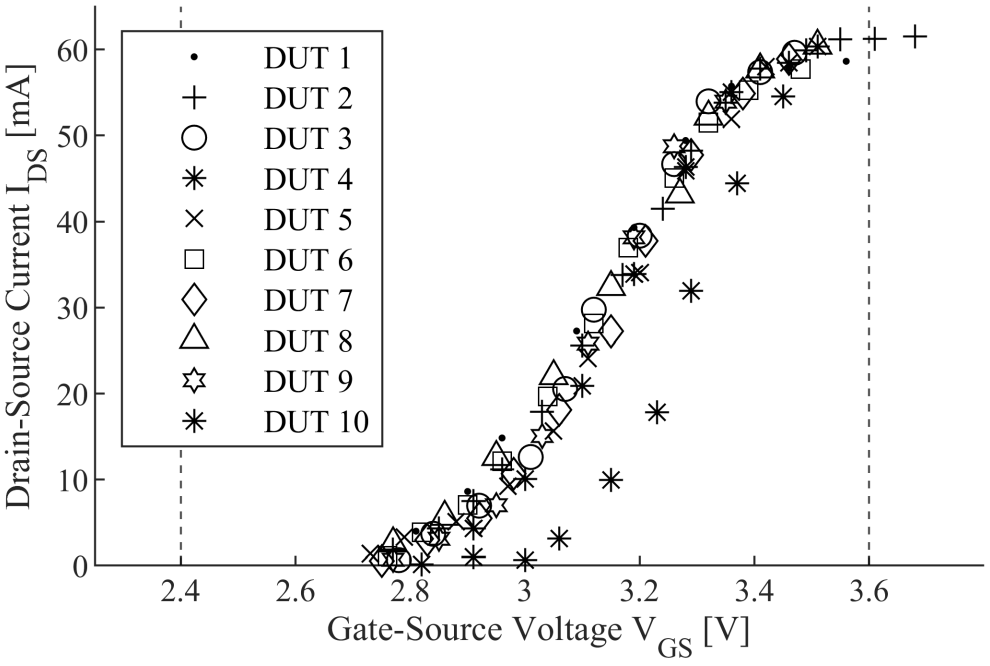

FIGURE 6.1: First round pre-radiation BUK7M $I_{DS} - V_{GS}$ plot for parts one through ten. Vertical lines indicate limits for V_{Th} as shown in Table X.

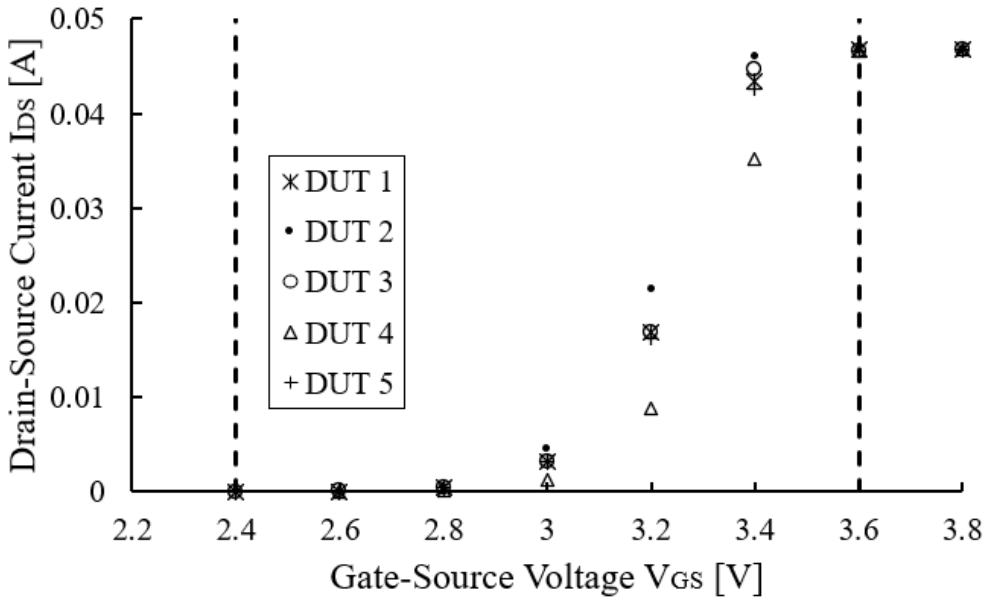

FIGURE 6.2: Second Round pre-radiation BUK7M $I_{DS} - V_{GS}$ plot for parts one through five.

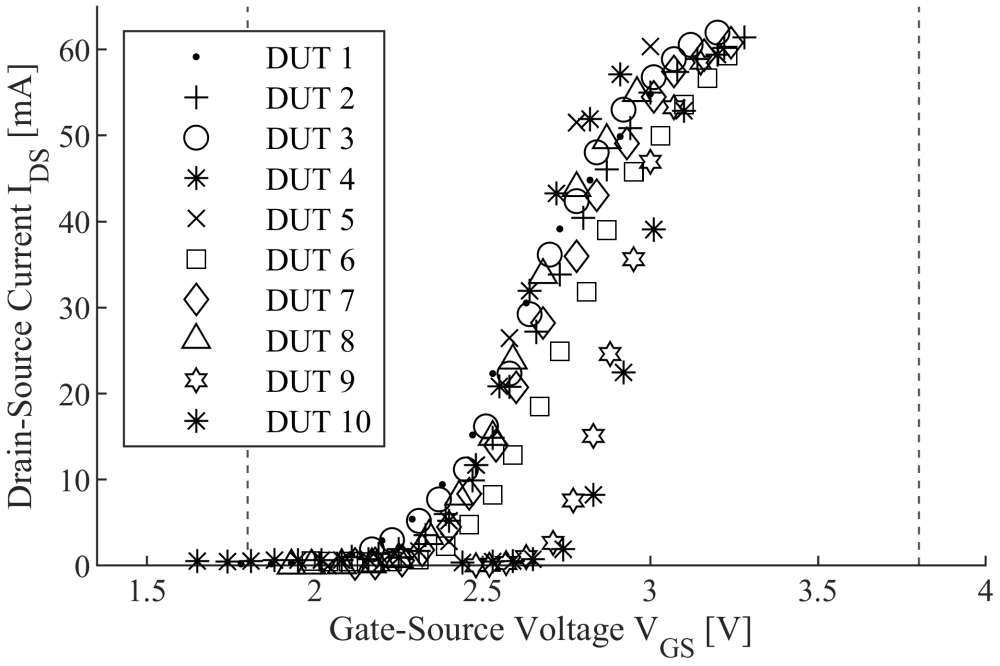

FIGURE 6.3: Pre-radiation 300NB06 $I_{DS} - V_{GS}$ plot for parts one through ten.

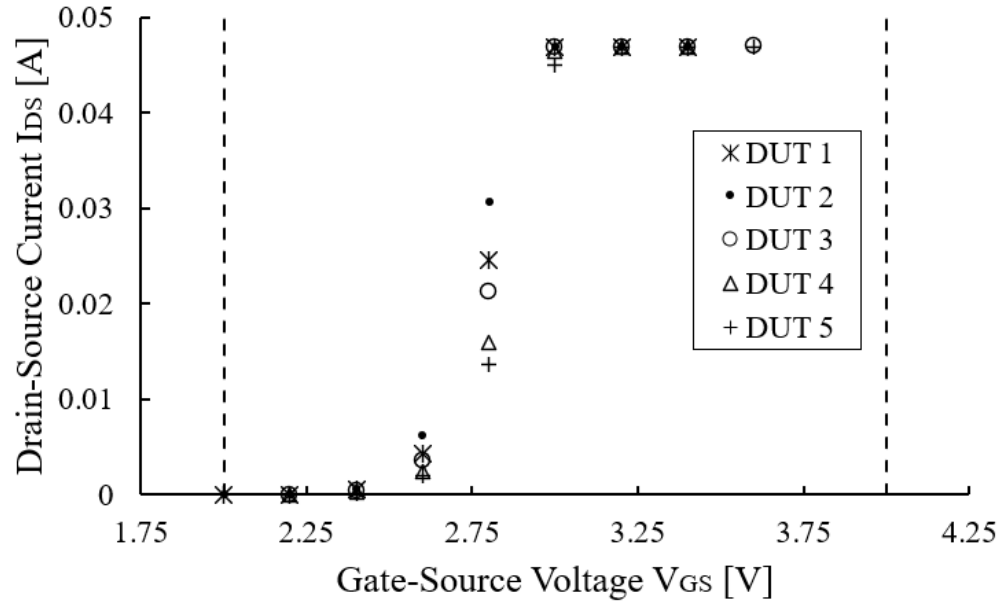

FIGURE 6.4: Pre-radiation DMNH6008 $I_{DS} - V_{GS}$ plot for parts one through ten.

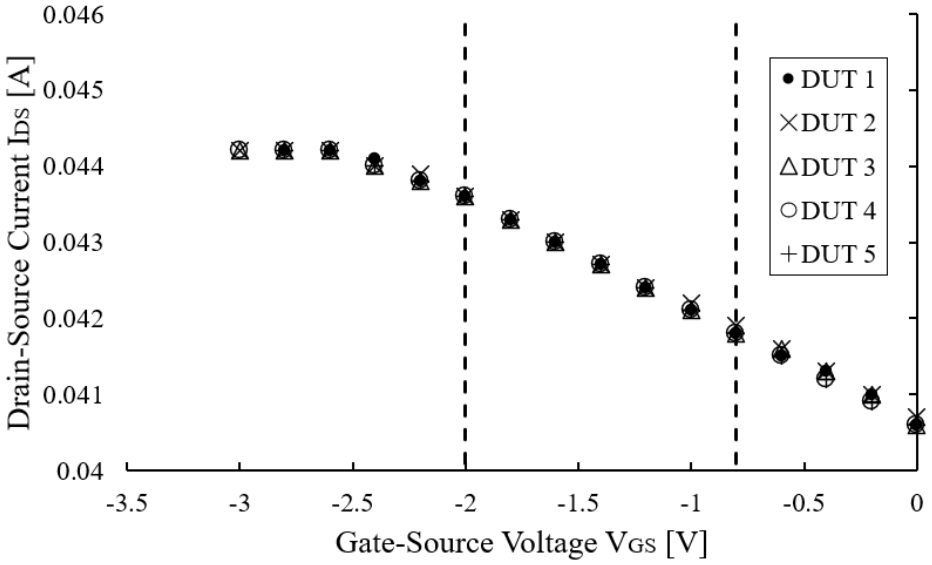

FIGURE 6.5: Pre-radiation BSS84A $I_{DS} - V_{GS}$ plot for parts one through five. Vertical lines indicate manufacturer limits for V_{Th} as shown in Table XI.

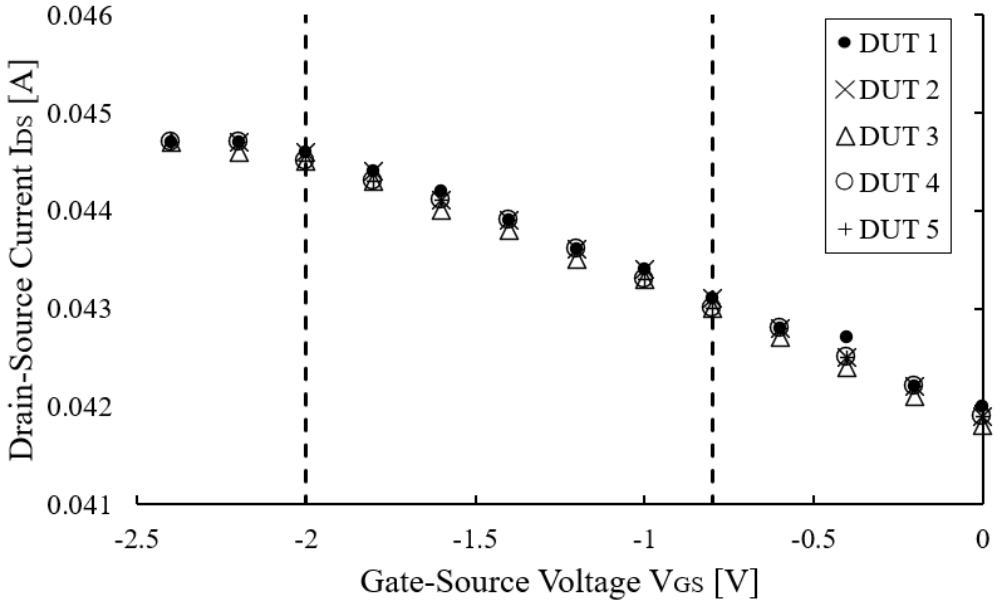

FIGURE 6.6: Pre-radiation SSM6J80x $I_{DS} - V_{GS}$ plot for parts one through five.

I_{DS} vs. V_{GS} characteristics from 0 to 25 kRad(Si) for a single sample of each component are provided in figures 6.7 through 6.12. The curves of each figure showed a correlation between threshold voltage and total ionizing dose. The gate threshold voltages were shown to decrease as the dose increased. Figures 6.13 through 6.18 show an approximately linear degradation in threshold voltage for each component. Table XII provides the estimated degradation rates for each component based on best-fit linear regressions.

TABLE XII: MOSFET V_{Th} Degradation Rates

Component	**Rate [V/kRad(Si)]**	R^2
BUK7M (1^{st} Round)	-0.0282	0.988
BUK7M (2^{nd} Round)	-0.0379	0.992
300NB06	-0.0485	0.996
DMNH6008	-0.0804	0.985
BSS84A	-0.0088	0.966
SSM6J80x	-0.0041	0.882

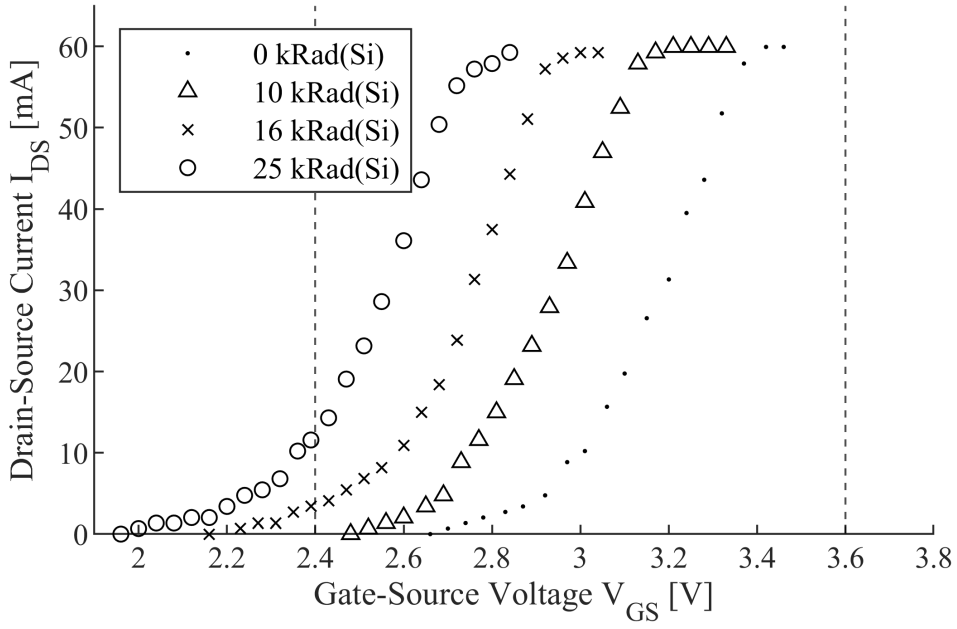

FIGURE 6.7: First Round BUK7M $I_{DS} - V_{GS}$ plot for part five, 0 to 25 kRad(Si). Vertical lines indicate limits for V_{Th} as shown in Table X.

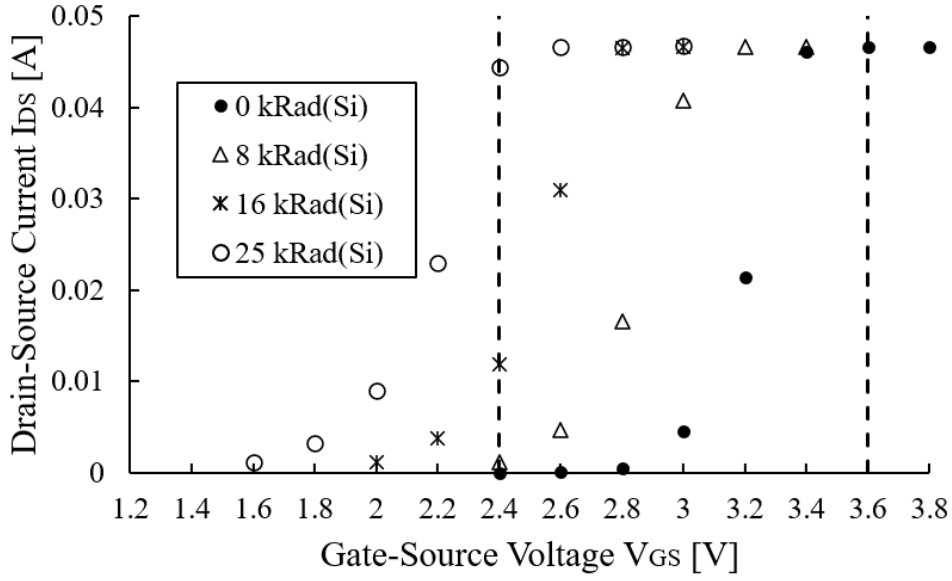

FIGURE 6.8: Second Round BUK7M $I_{DS} - V_{GS}$ plot for part two, 0 to 25 kRad(Si).

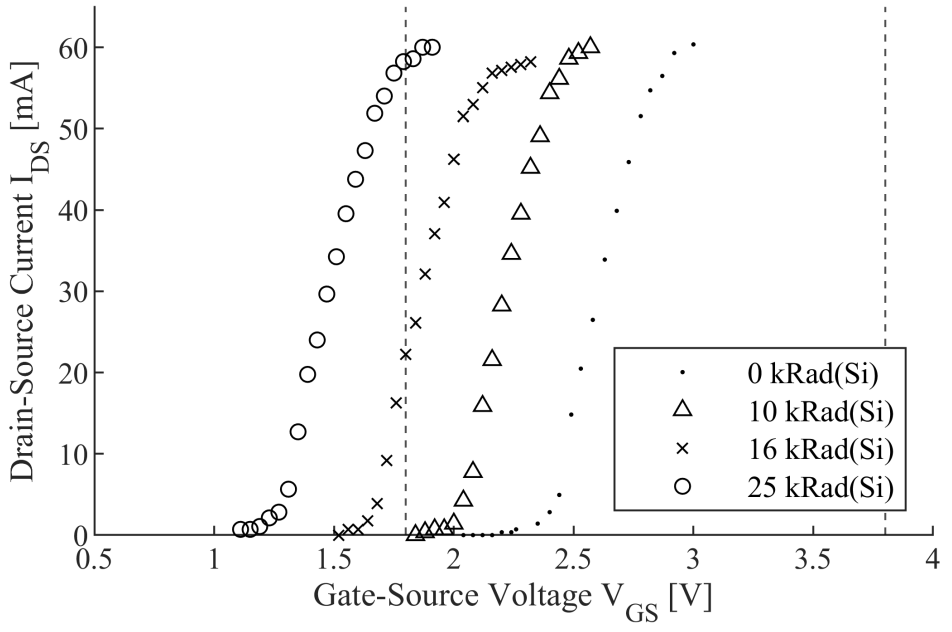

FIGURE 6.9: 300NB06 $I_{DS} - V_{GS}$ plot for part five, 0 to 25 kRad(Si).

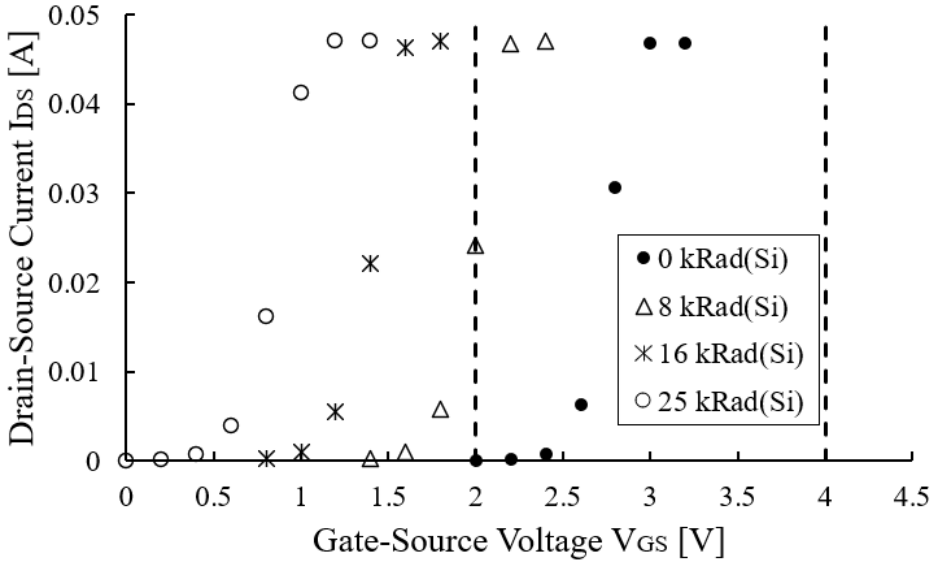

FIGURE 6.10: DMNH6008 $I_{DS} - V_{GS}$ plot for part two, 0 to 25 kRad(Si).

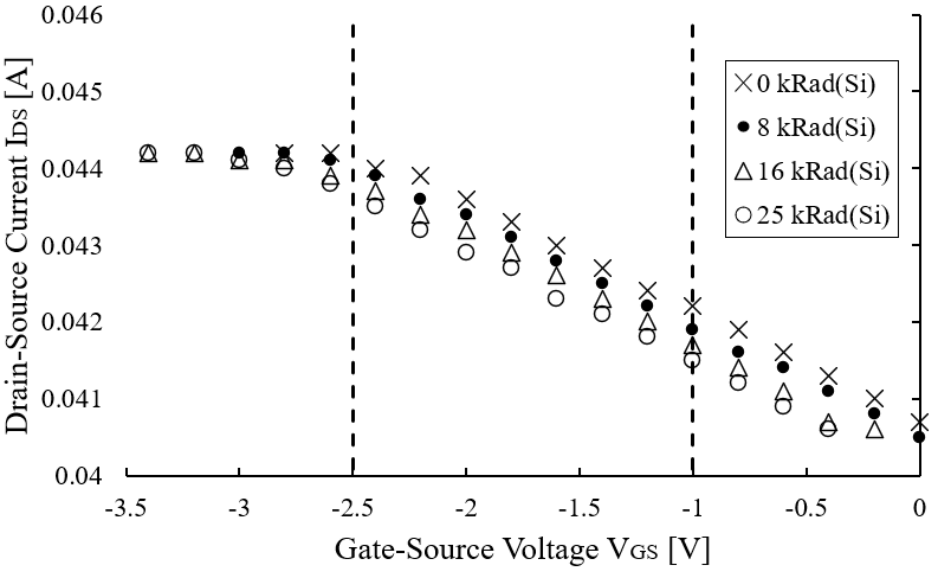

FIGURE 6.11: BSS84A $I_{DS} - V_{GS}$ plot for part two, 0 to 25 kRad(Si). Vertical lines indicate limits for V_{Th} as shown in Table XI.

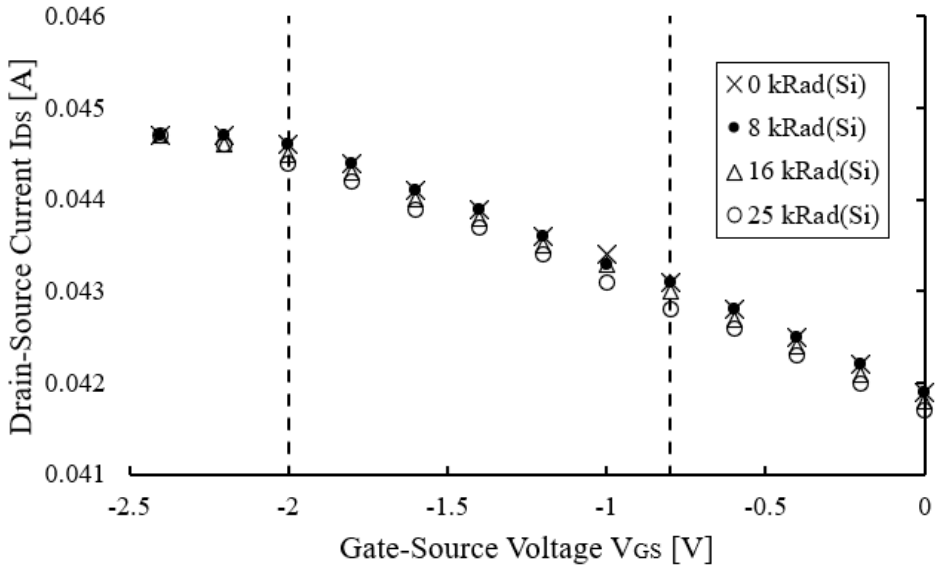

FIGURE 6.12: SSM6J80x $I_{DS} - V_{GS}$ plot for part two, 0 to 25 kRad(Si).

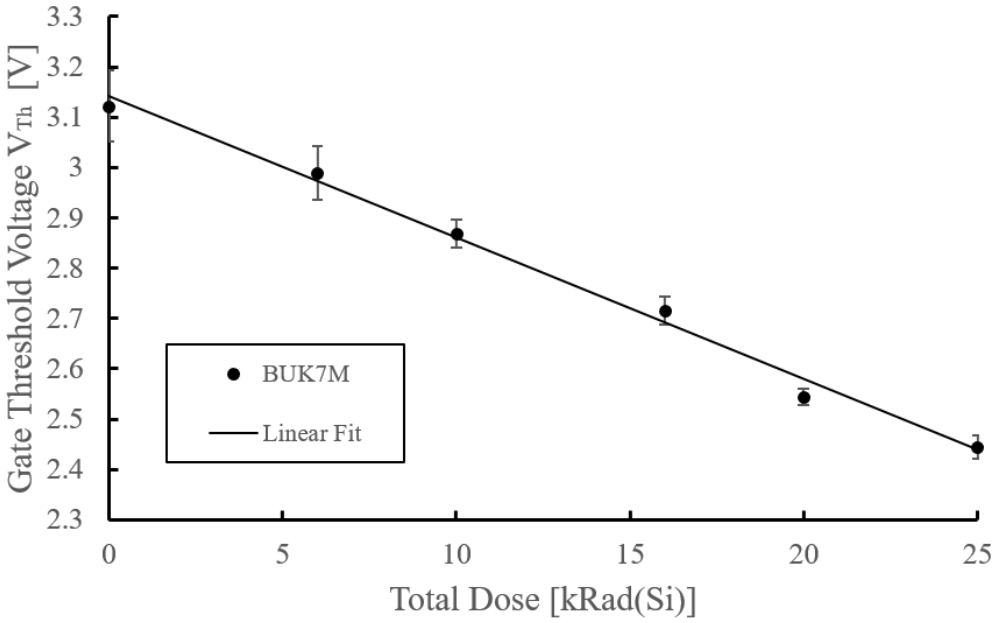

FIGURE 6.13: First Round BUK7M Gate threshold voltage V_{Th} vs. total ionizing dose. Error bars represent one standard deviation from the mean within the part-to-part variation.

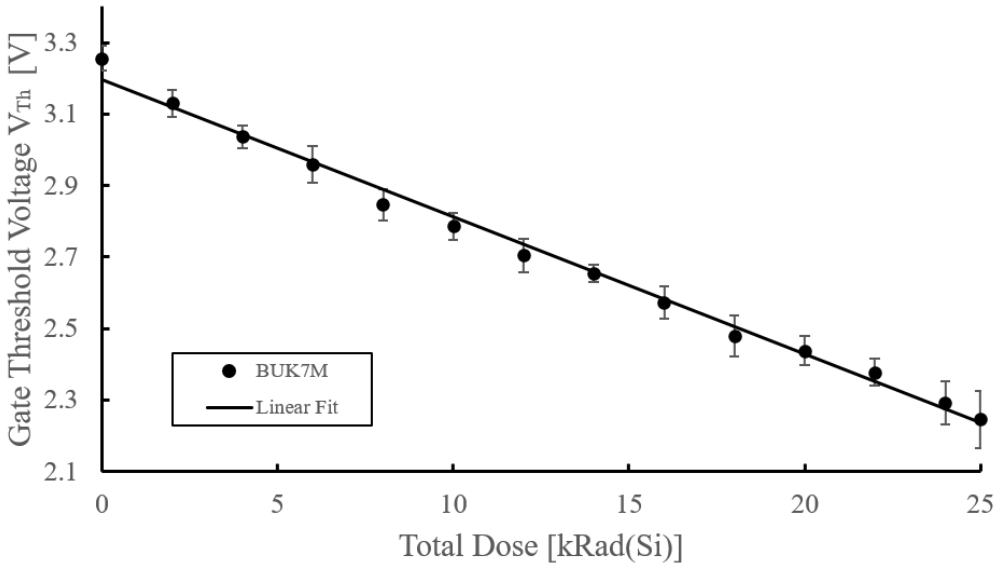

FIGURE 6.14: Second Round BUK7M Gate threshold voltage V_{Th} vs. total ionizing dose.

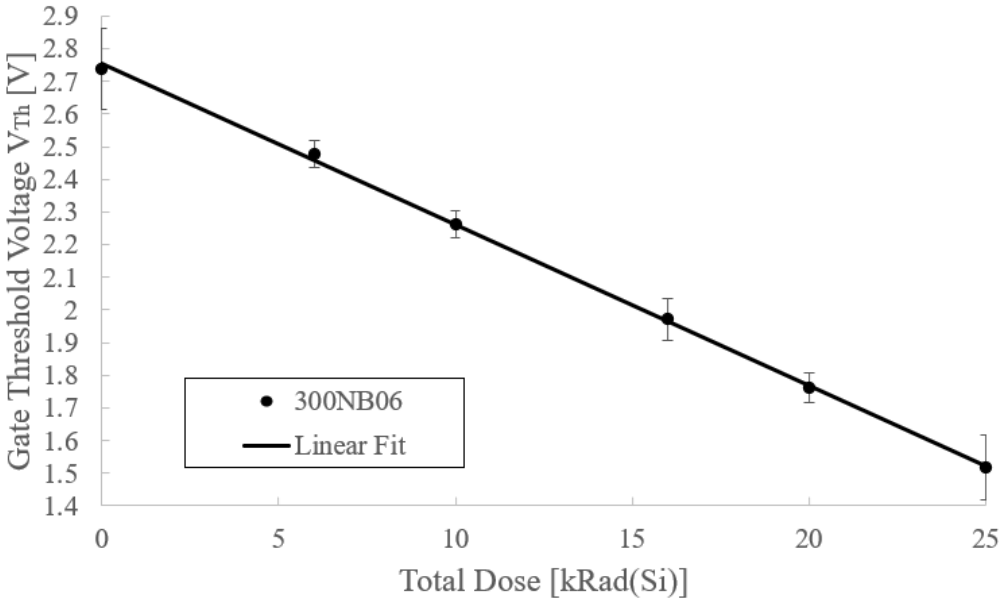

FIGURE 6.15: 300NB06 Gate threshold voltage V_{Th} vs. total ionizing dose.

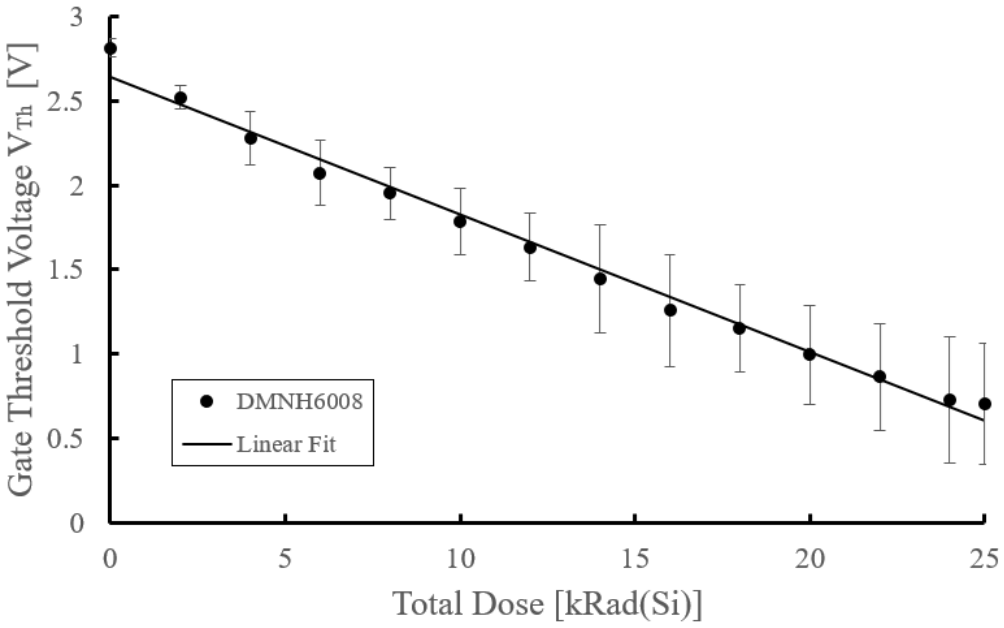

FIGURE 6.16: DMNH6008 Gate threshold voltage V_{Th} vs. total ionizing dose.

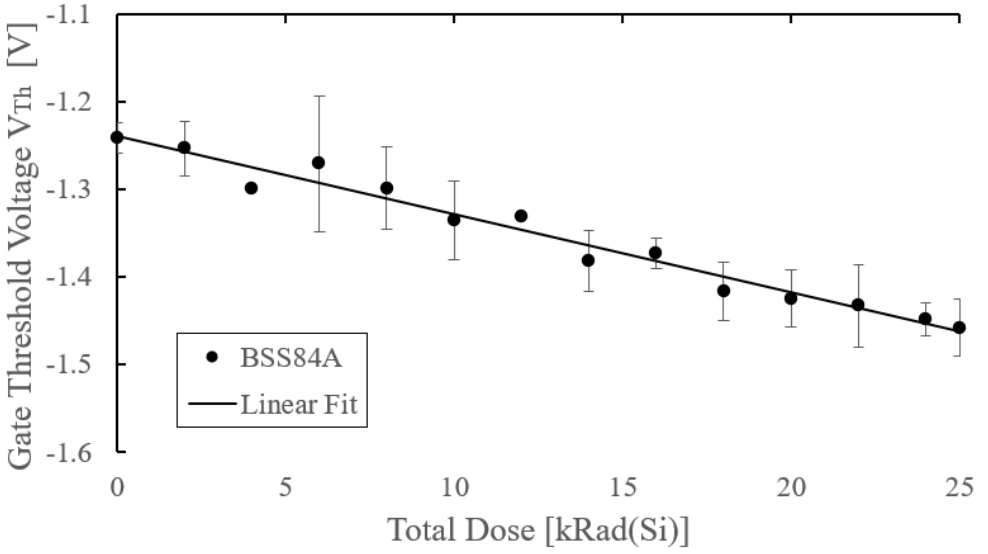

FIGURE 6.17: BSS84A Gate threshold voltage V_{Th} vs. total ionizing dose.

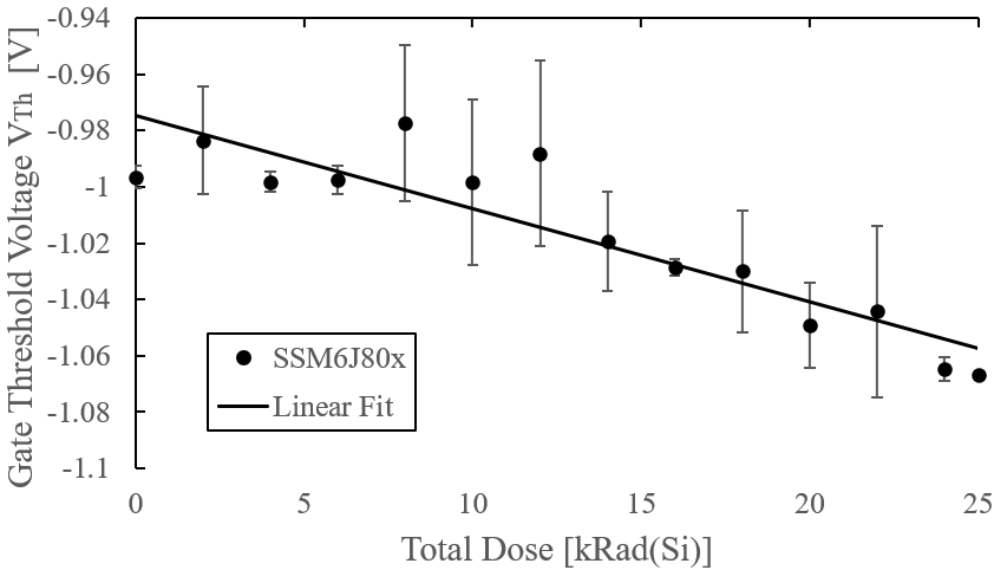

FIGURE 6.18: SSM6J80x Gate threshold voltage V_{Th} vs. total ionizing dose.

The test results of each component are listed in tables XIII and XIV. The failure calculations were calculated per the methods described in chapter 3. The one-sided tolerance limit factor K_{TL} equals 3.532 and 4.666 for ten and five samples, respectively, at a chosen probability of 99% and a confidence level of 90%. The DMNH6008 was the only MOSFET categorized as unacceptable for use. The other MOSFETs were classified as HCC-2 or HNC (for the hypothetical mission dose of 6.4 kRad(Si)). Two separate purchases of the BUK7M were tested six months apart, resulting in different failure doses. In either case, the radiation design margin remained above the minimum acceptable limit of three. The P-channel MOSFETs did not experience failure after irradiation and required hardness categorization using parametric design margins. In contrast, the N-channel MOSFETs did experience failure and used radiation design margins instead. Figures 6.19 through 6.24 provide each component's cumulative probability of failure distributions.

TABLE XIII: N-Channel MOSFET TID Results

Part	$\overline{R_{Fail}}$	$\sigma_{ln(R_{Fail})}$	**Samples**	K_{TL}	*PCC*	*RDM*	**Category**
BUK7M (1^{st})	26 kRad(Si)	0.258	10	3.532	2.489	4.1	HCC-2
300NB06	19 kRad(Si)	0.607	10	3.532	1.239	3.0	HCC-2
BUK7M (2^{nd})	21 kRad(Si)	0.050	5	4.666	1.260	3.2	HCC-2
DMNH6008	8 kRad(Si)	0.219	5	4.666	2.776	1.2	N/A

TABLE XIV: P-Channel MOSFET TID Results

Part	$\overline{PAR_{Rad}}$	$\sigma_{ln(PAR_{RAD})}$	**Samples**	K_{TL}	*PCC*	*PDM*	**Category**
BSS84A	-1.278 V	0.0773	5	4.666	1.434	34.5	HCC-2
SSM6J80x	-1.067 V	0.0050	5	4.666	1.024	1000	HNC

No SEB or SEGR were witnessed in each part as V_{DS} was incremented from 0 to 30 V. However, the threshold for seeing such effects may reside within the remaining untested V_{DS} ranges which was outside the scope of the de-rated range of each component. Additionally, the parts may be susceptible to SEEs under heavy-ion screening.

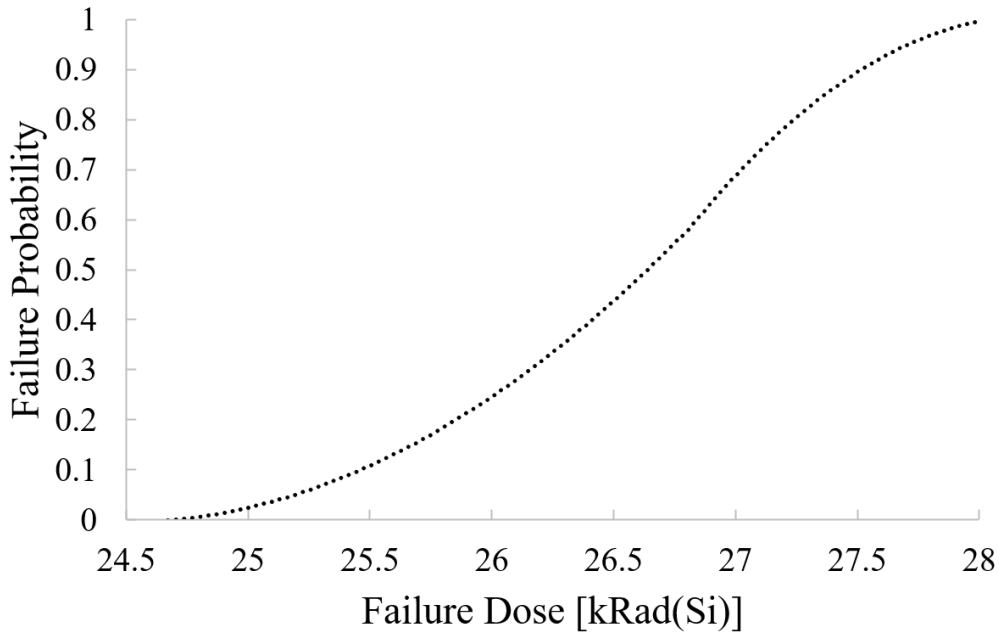

FIGURE 6.19: Cumulative probability distribution of failure dose for the BUK7M.

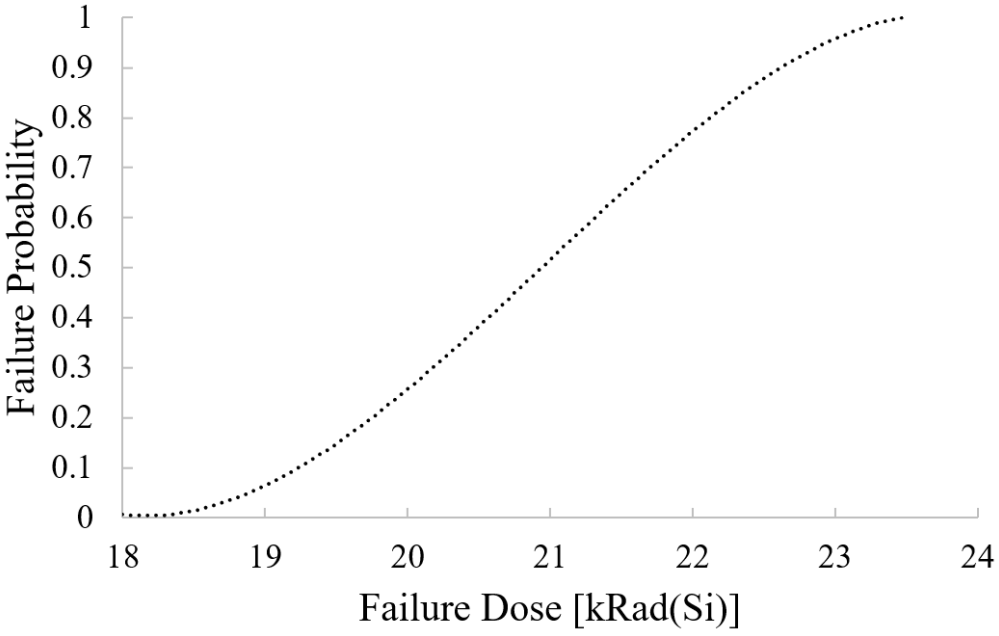

FIGURE 6.20: Cumulative probability distribution of failure dose for the BUK7M.

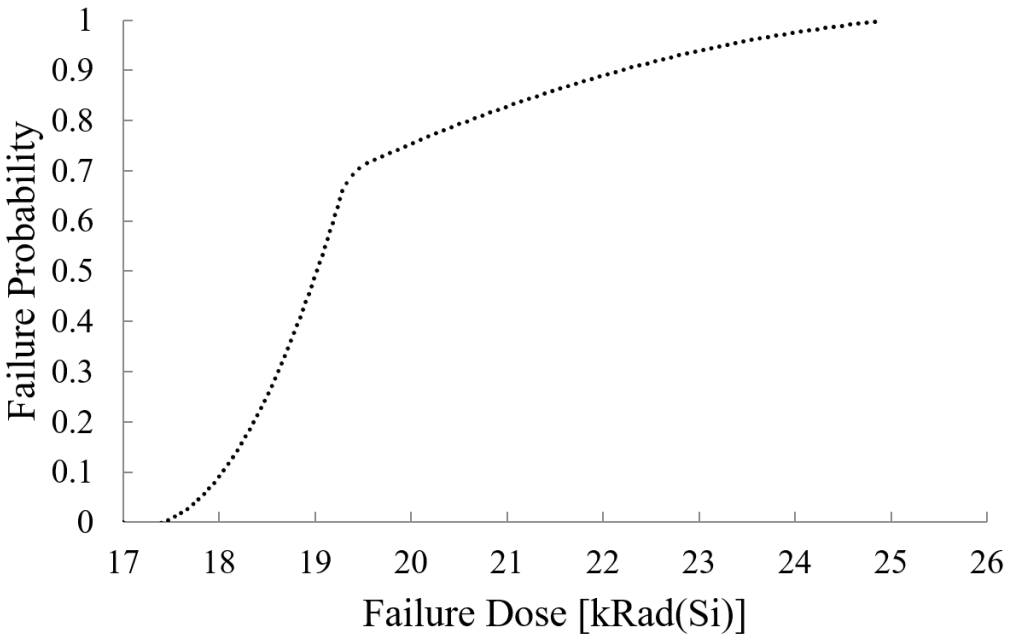

FIGURE 6.21: Cumulative probability distribution of failure dose for the 300NB06.

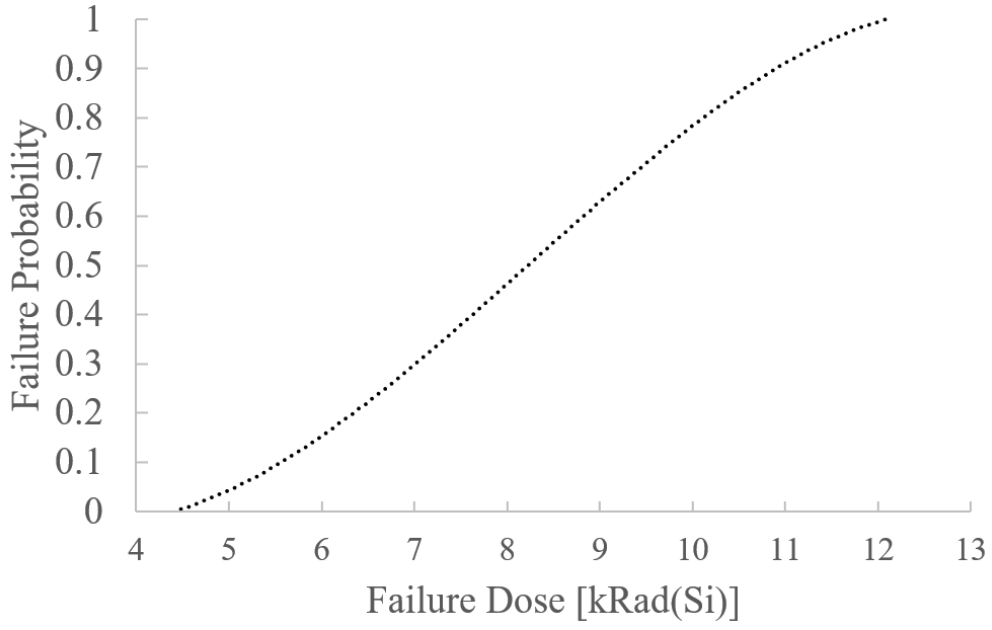

FIGURE 6.22: Cumulative probability distribution of failure dose for the DMNH6008.

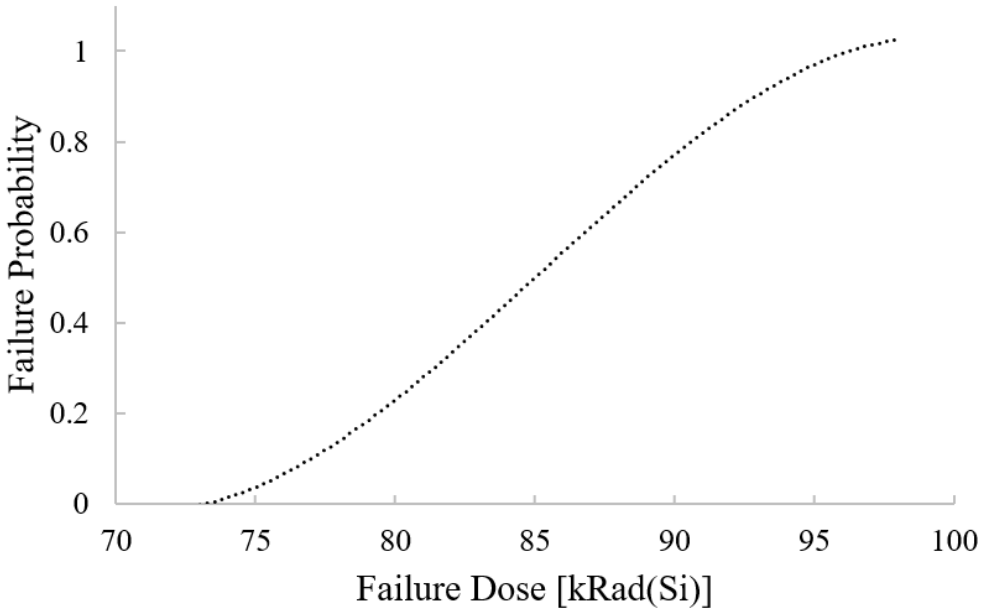

FIGURE 6.23: Cumulative probability distribution of failure dose for the BSS84A.

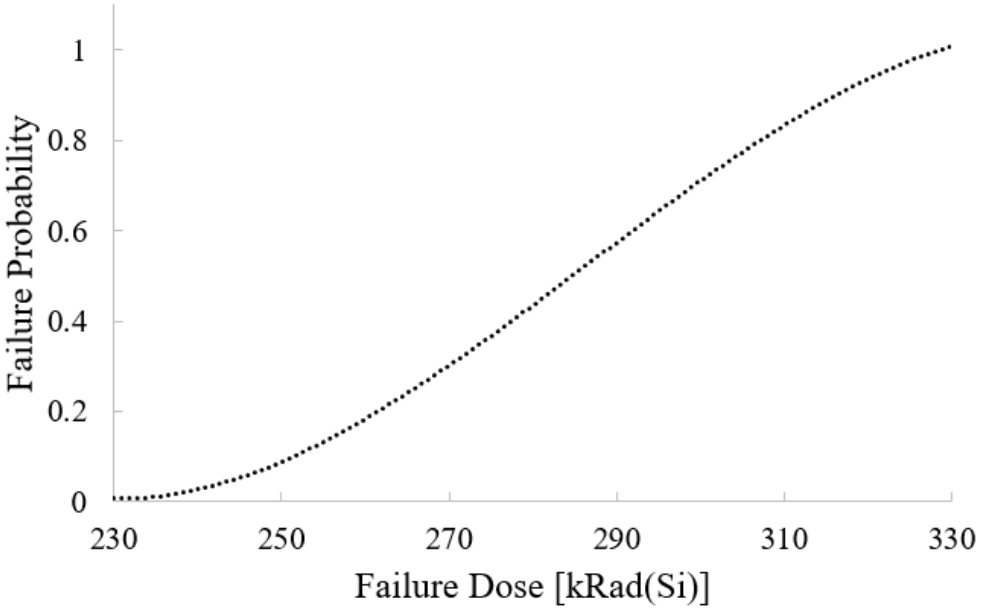

FIGURE 6.24: Cumulative probability distribution of failure dose for the SSM6J80x.

6.2 Half-Bridge Drivers

I screened four half-bridge drivers during the two test campaigns. Each component underwent combined failure screening per the methods described in section 5.3.2, and I performed failure analysis using the *RDM* procedures outlined in chapter 3. Table XV lists the vendor-supplied electrical parameters in relation to the input voltage received by the parts during the tests. The TC442 driver was screened in both test campaigns.

TABLE XV: Half-Bridge Driver Parameters ([89], [90], [91], [92])

Part	**Test Bench Supply** V_{CC}	**Off-State** V_{Off}	**On-State** V_{On}
TC442 (1^{st})	8.5 V	0 to 0.025 V	$\geq V_{CC} - 0.025$ V
TC442 (2^{nd})	8 V	0 to 0.025 V	$\geq V_{CC} - 0.025$ V
TPS2822	8 V	0 to 0.3 V	$V_{CC} \pm 0.3$ V
DGD054x	8 V	0 to 0.3 V	$V_{CC} \pm 0.3$ V
ISL784x	8 V	0 to 0.3 V	$V_{CC} \pm 0.3$ V

Pre-radiation control measurements are plotted in figures 6.25 through 6.34. Figure 6.29 indicated that the TPS2822 experienced systemic deviation from one of its expected outputs. During the second test campaign, the TPS2822's low-side output voltage readings remained at 2 V in the off-state, indicating a possible design issue on the DUT board. Additionally, the high-side outputs of the TPS2822, DGD054x and ISL784x components were powered through a bootstrap circuit and required pre-screening measurements to determine their expected ranges. Later, I centered the failure bounds around mean values of the control data. Table XVI lists the failure bounds for each component.

TABLE XVI: Half-Bridge Driver Failure Bounds

Part	**High-Side Off**	**High-Side On**	**Low-Side Off**	**Low-Side On**
TC442 (1^{st})	0 to 0.025 V	$\geq$ 8.475 V	0 to 0.025 V	$\geq$ 8.475 V
TC442 (2^{nd})	0 to 0.025 V	$\geq$ 7.953 V	0 to 0.025 V	$\geq$ 7.953 V
TPS2822	0 to 0.3 V	7.0 to 7.6 V	1.7 to 2.3 V	7.7 to 8.3 V
DGD054x	0 to 0.3 V	7.1 to 7.7 V	0 to 0.3 V	7.7 to 8.3 V
ISL784x	0 to 0.3 V	7.3 to 7.9 V	0 to 0.3 V	7.7 to 8.3 V

Samples six and nine for the first round of TC442 data had elevated off-state voltage readings, which lay outside the failure bounds of the rest of the samples. These two samples responded to radiation in the same manner as the eight other DUTs, despite starting with a different output voltage.

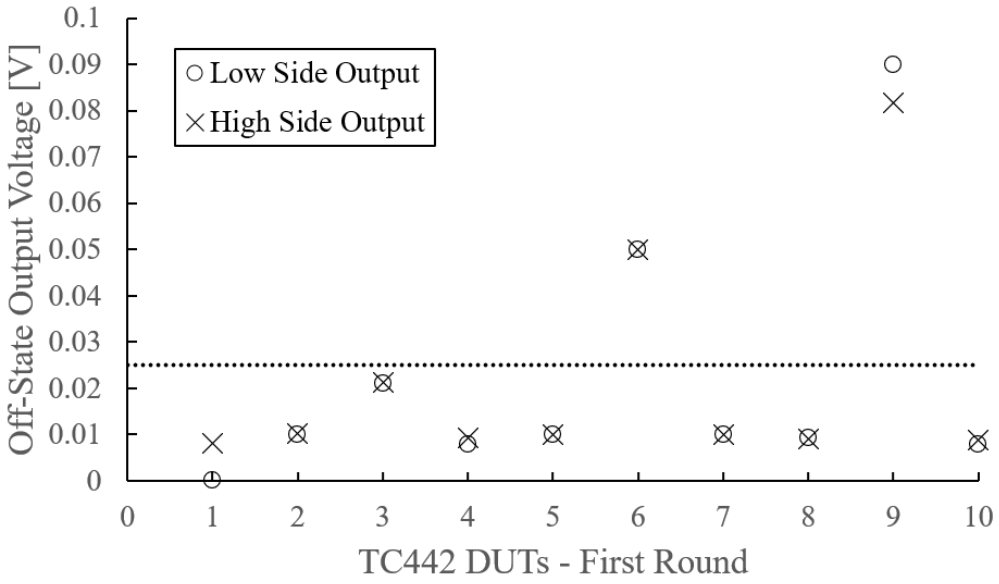

FIGURE 6.25: First Round TC442 Off-State Control Data. Horizontal lines indicate failure limits.

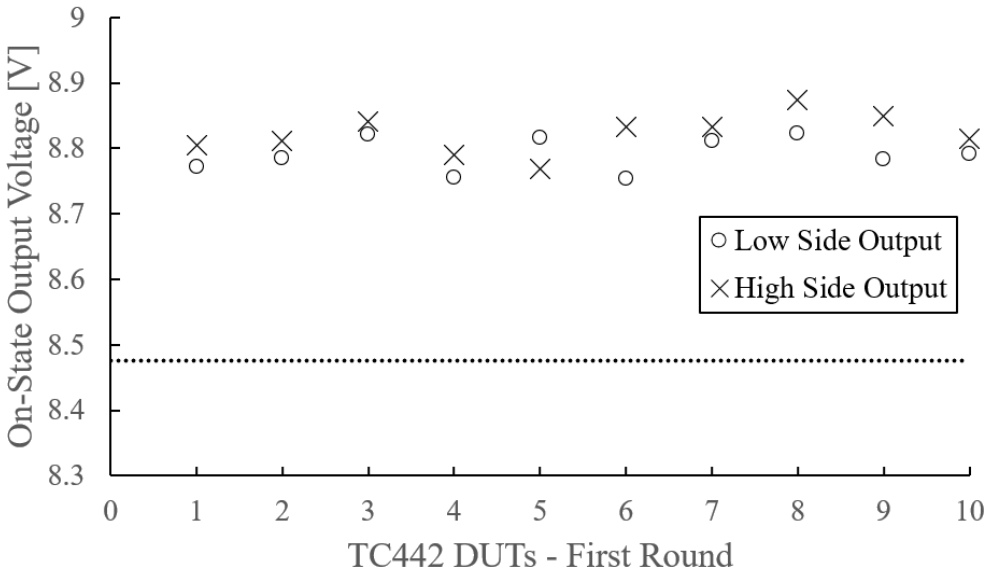

FIGURE 6.26: First Round TC442 On-State Control Data.

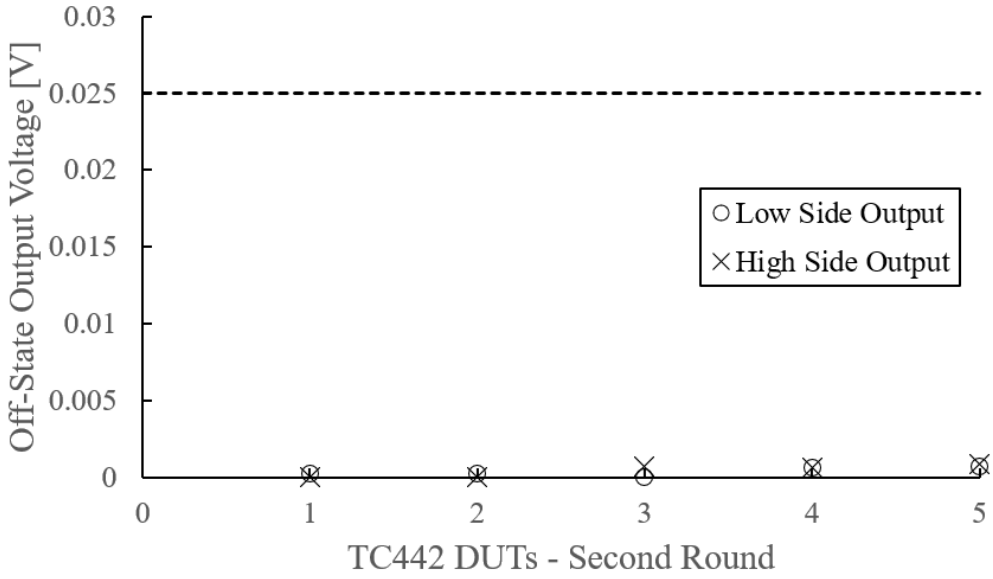

FIGURE 6.27: Second Round TC442 Off-State Control Data.

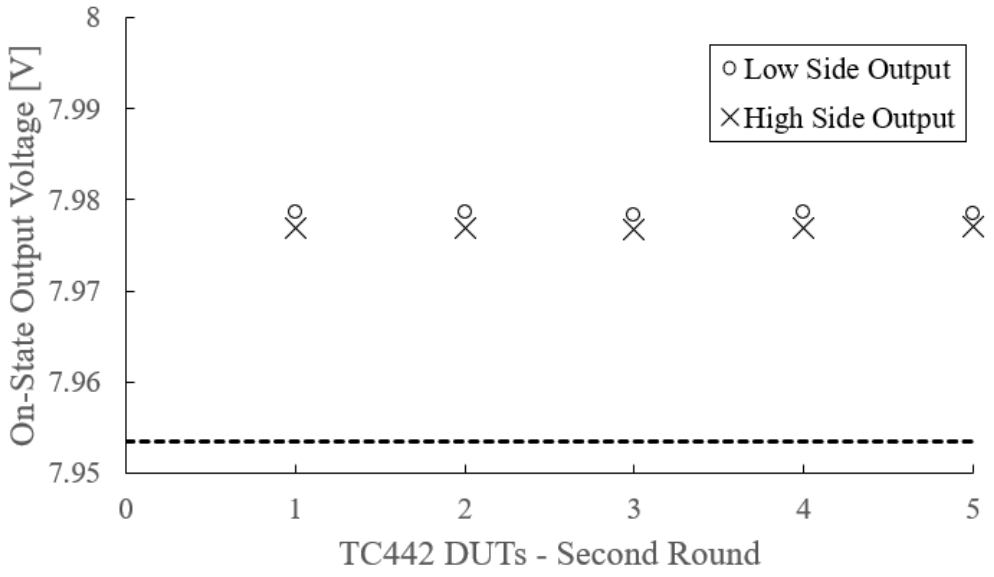

FIGURE 6.28: Second Round TC442 On-State Control Data.

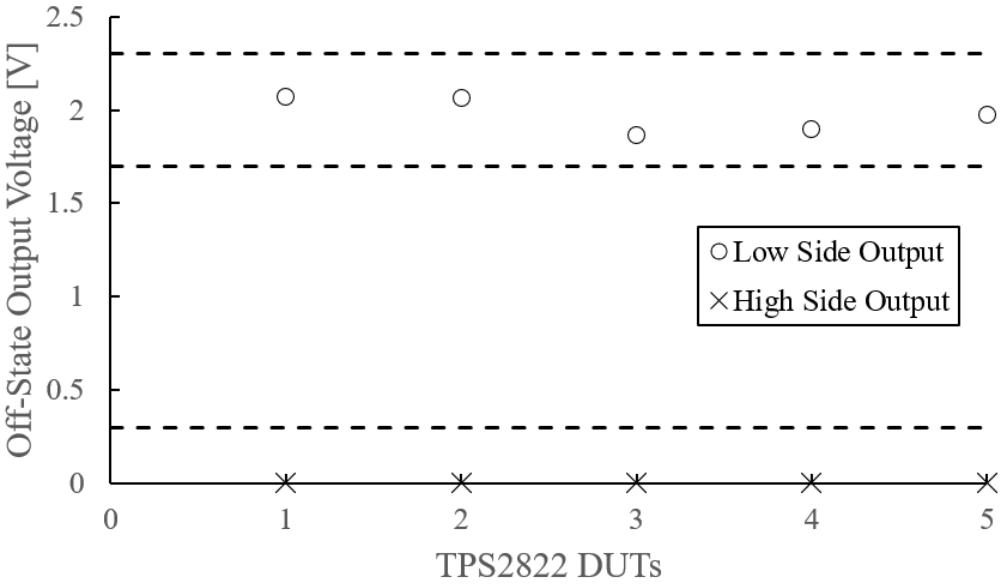

FIGURE 6.29: TPS2822 Off-State Control Data.

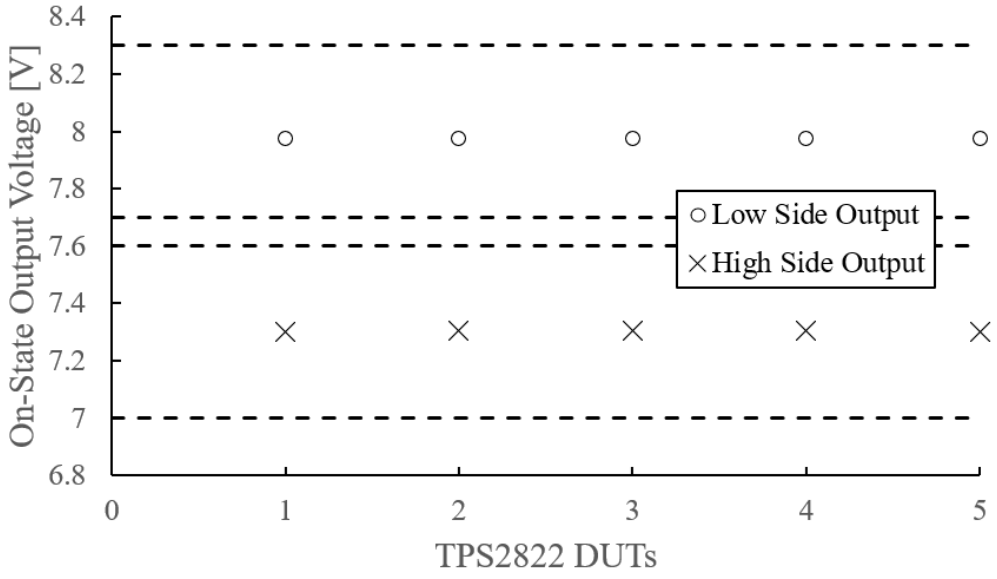

FIGURE 6.30: TPS2822 On-State Control Data.

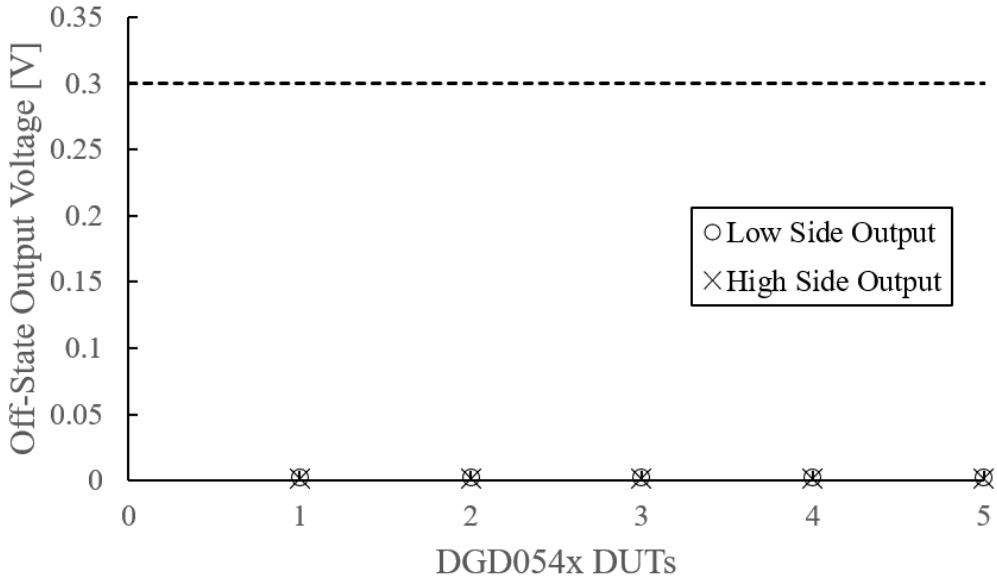

FIGURE 6.31: DGD054x Off-State Control Data.

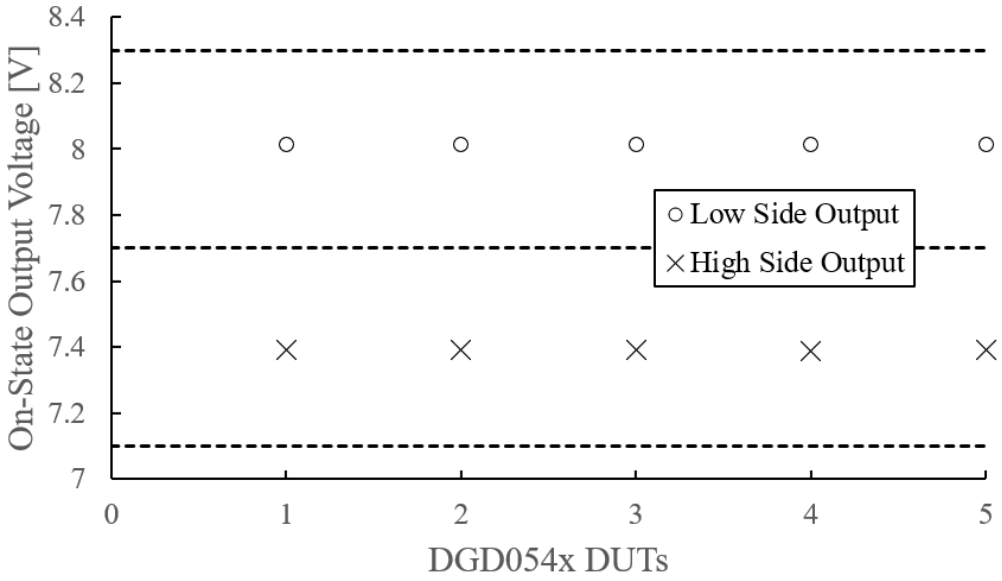

FIGURE 6.32: DGD054x On-State Control Data.

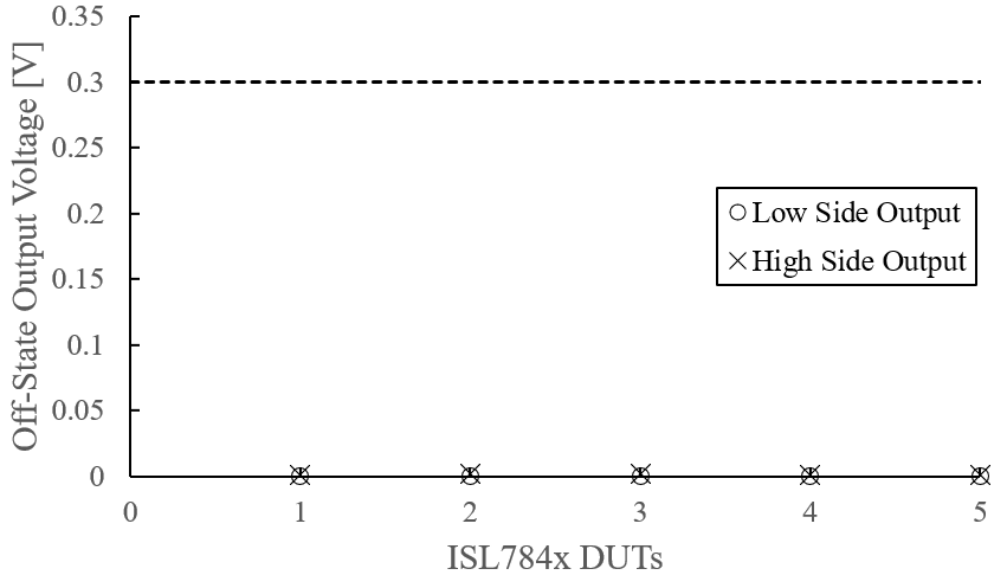

FIGURE 6.33: ISL784x Off-State Control Data.

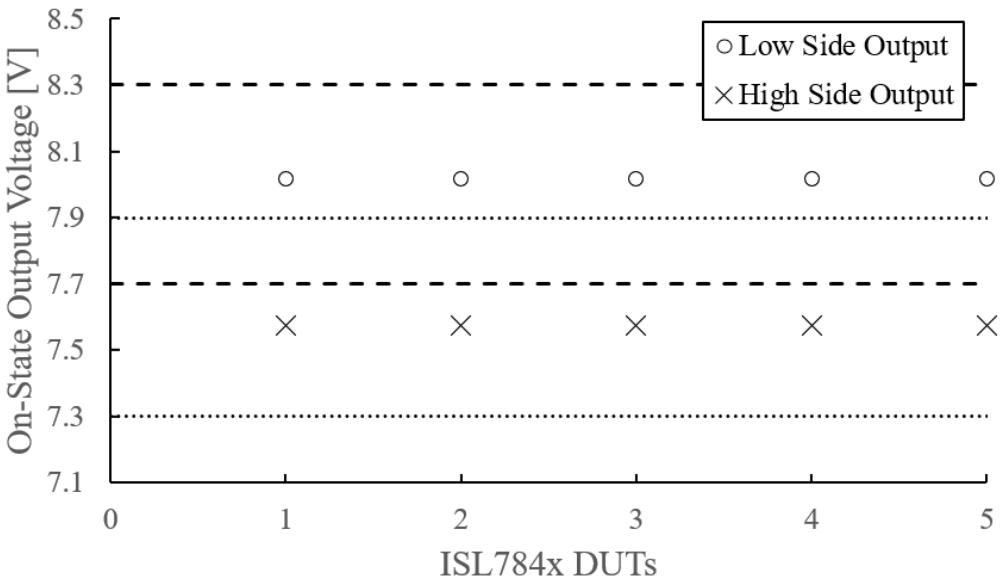

FIGURE 6.34: ISL784x On-State Control Data.

Radiation response plots for the components are provided in figures 6.35 through 6.48. The TC442 driver survived both screening campaigns, and my findings echoed the results published by JPL in 2009 [34]. Meanwhile, the TPS2822 driver experienced low-side failure in every sample. The low-side output pin consistently became stuck in an on-state at eight kRad(Si), indicative of TID failure. A cumulative probability failure plot for the TPS2822 is provided in figure 6.42. Table XVII lists the results for all the components. Every component except the TPS2822 driver was categorized as HCC-2. The actual *RDM* values were not obtained for the categorized components. However, the minimum bounds on the design margins were constrained above an acceptable level of 3.9 for a mission dose of 6.4 kRad(Si), given that the parts survived to 25 kRad(Si).

TABLE XVII: Half-Bridge Driver TID Results

Part	$\overline{R_{Fail}}$	$\sigma_{ln(R_{Fail})}$	**Samples**	K_{TL}	*PCC*	*RDM*	**Category**
TC442 (1^{st})	>25 kRad(Si)	-	10	3.532	1	>3.9	HCC-2
TC442 (2^{nd})	>25 kRad(Si)	-	5	4.666	1	>3.9	HCC-2
TPS2822	8 kRad(Si)	0.098	5	4.666	1.577	2.3	N/A
DGD054x	>25 kRad(Si)	-	5	4.666	1	>3.8	HCC-2
ISL784x	>25 kRad(Si)	-	5	4.666	1	>3.9	HCC-2

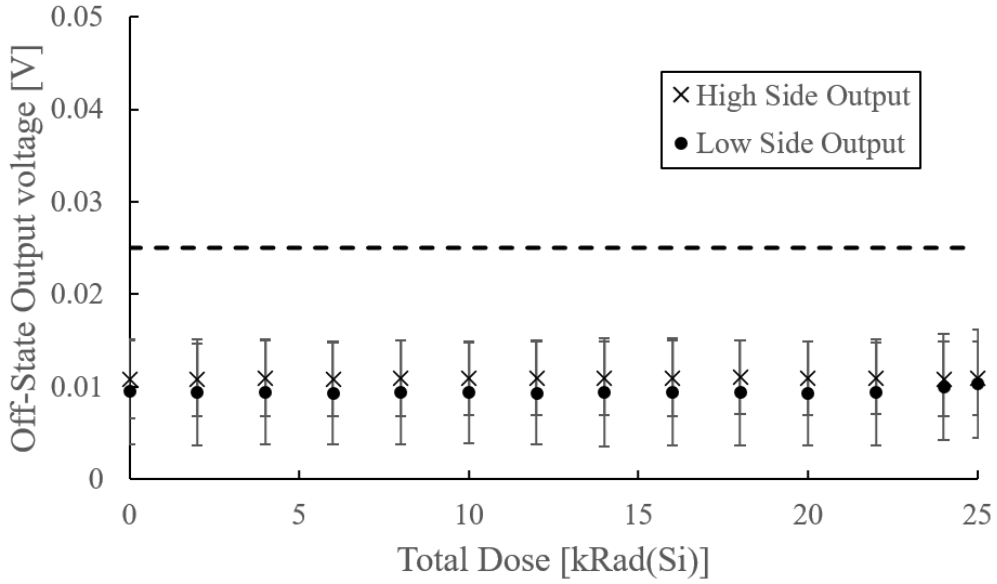

FIGURE 6.35: First Round TC442 Off-State Output Voltage vs. Total dose. Error bars represent one standard deviation from the mean within the part-to-part variation. Horizontal lines represent failure limits.

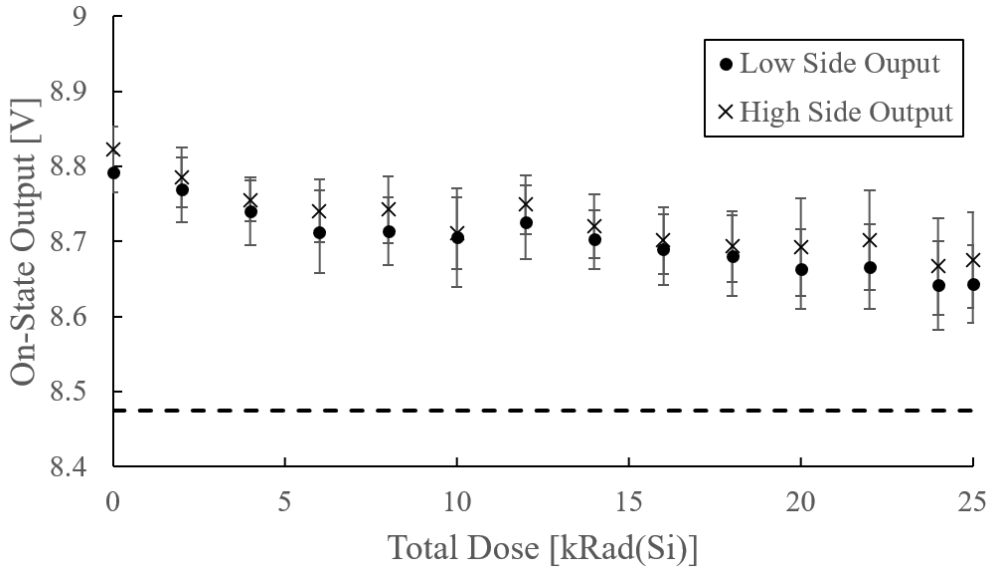

FIGURE 6.36: First Round TC442 On-State Output Voltage vs. Total Dose.

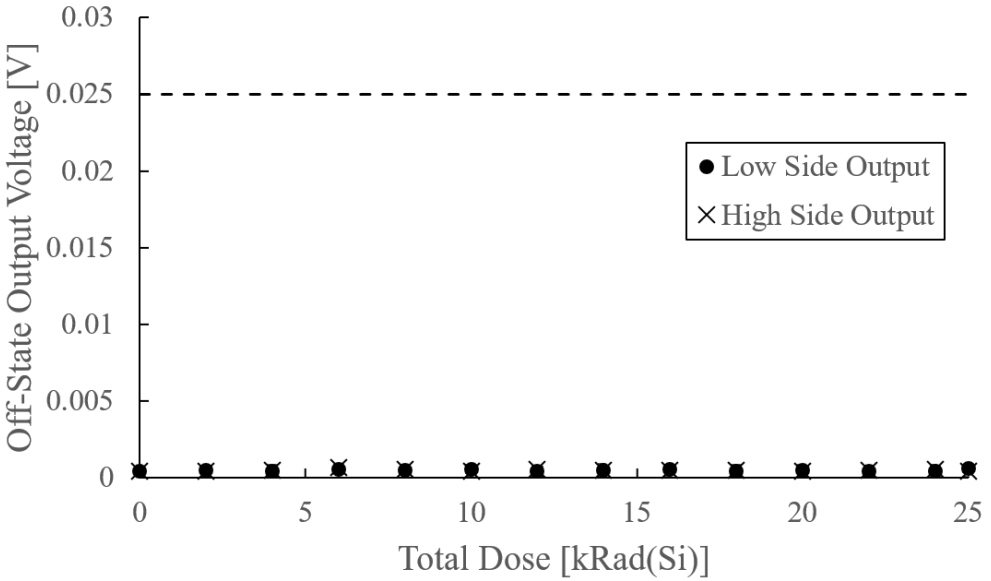

FIGURE 6.37: Second Round TC442 Off-State Output Voltage vs. Total Dose.

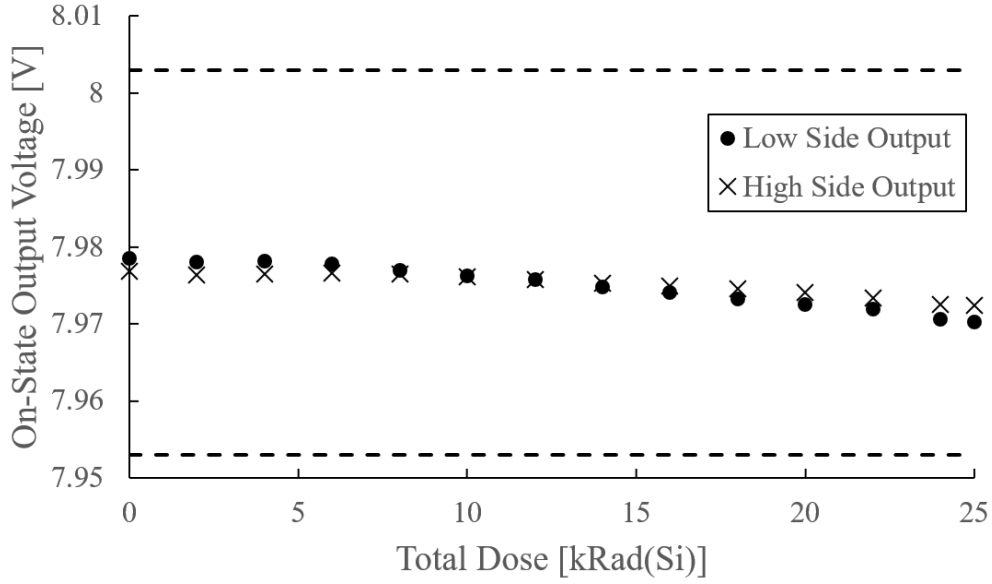

FIGURE 6.38: Second Round TC442 On-State Output Voltage vs. Total Dose.

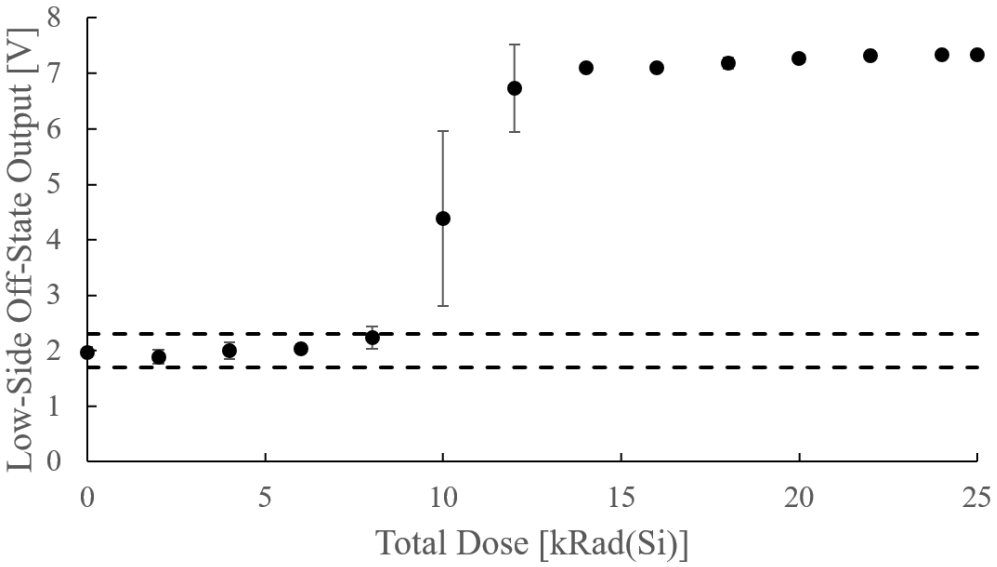

FIGURE 6.39: TPS2822 Off-State Output Voltage vs. Total Dose.

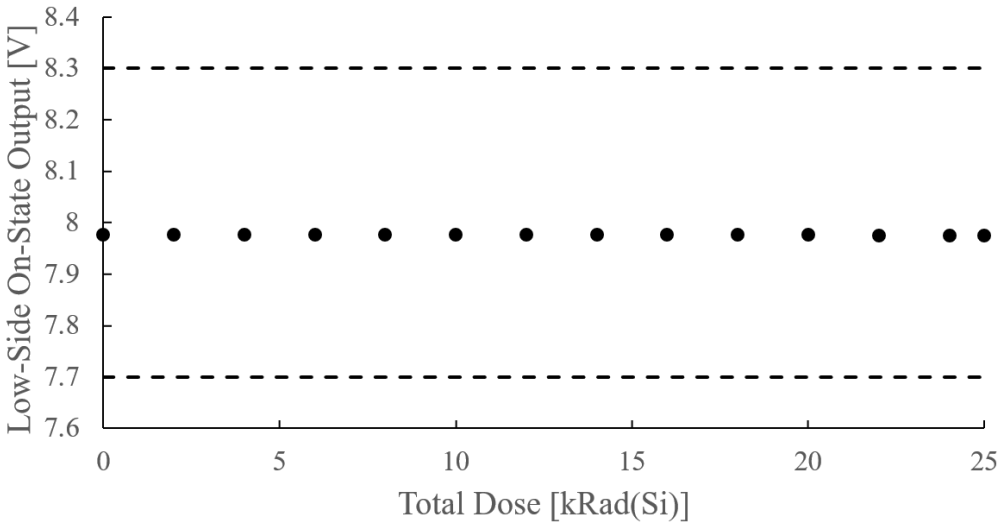

FIGURE 6.40: TPS2822 Low-Side On-State Output Voltage vs. Total Dose.

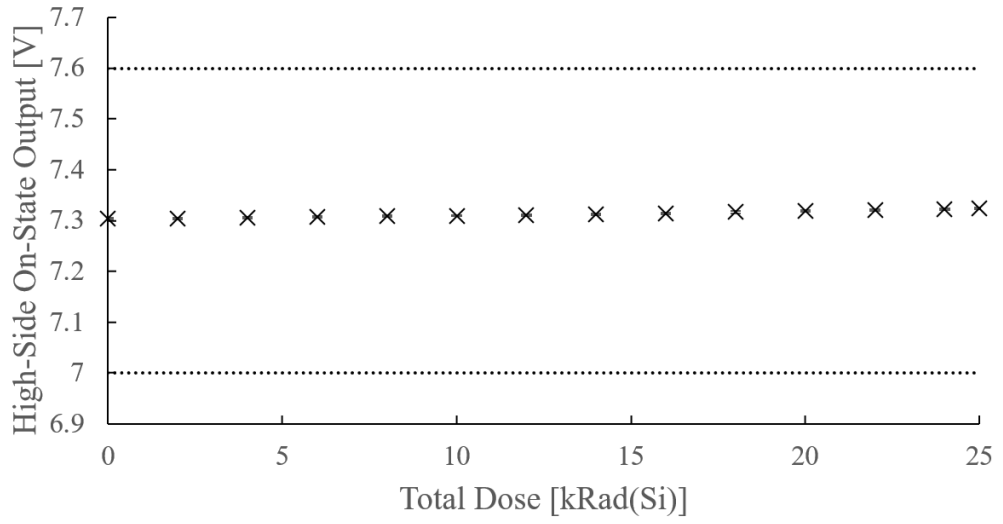

FIGURE 6.41: TPS2822 High-Side On-State Output Voltage vs. Total Dose.

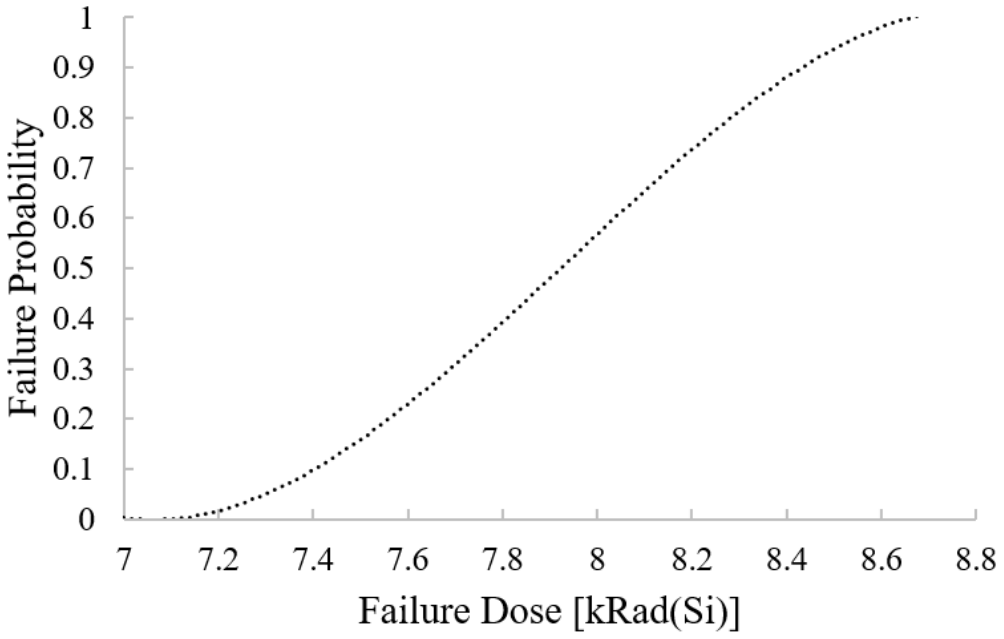

FIGURE 6.42: Cumulative probability of failure distribution for the TPS2822.

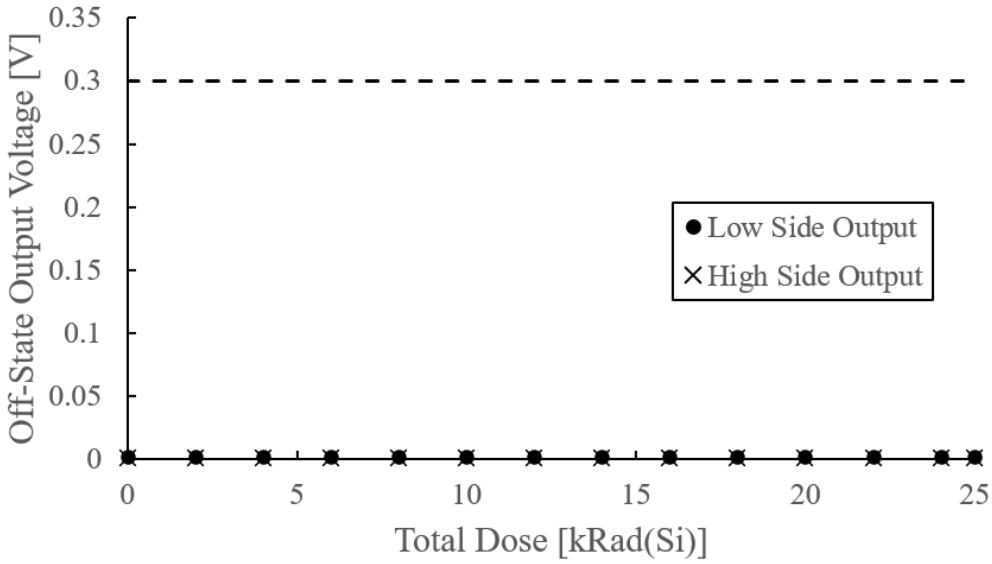

FIGURE 6.43: DGD054x Off-State Output Voltage vs. Total Dose.

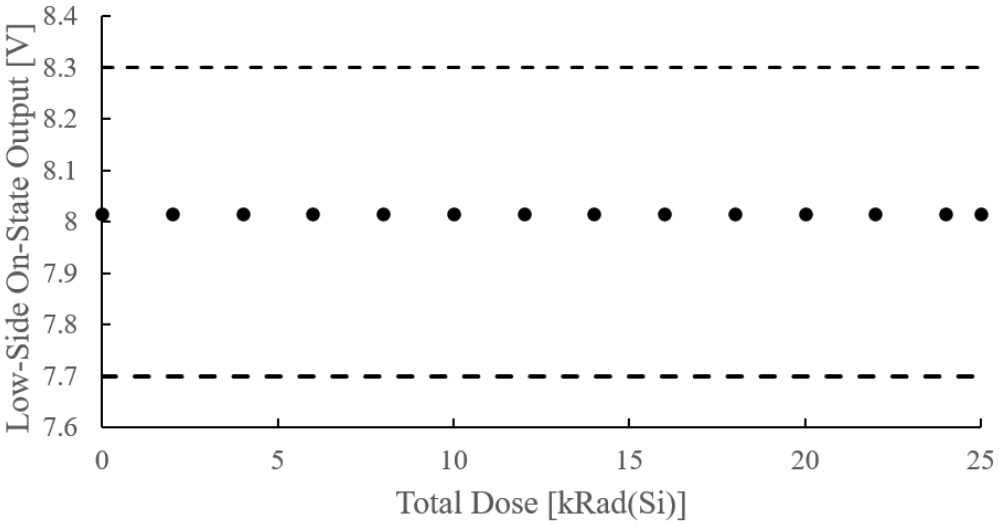

FIGURE 6.44: DGD054x Low-Side On-State Output Voltage vs. Total Dose.

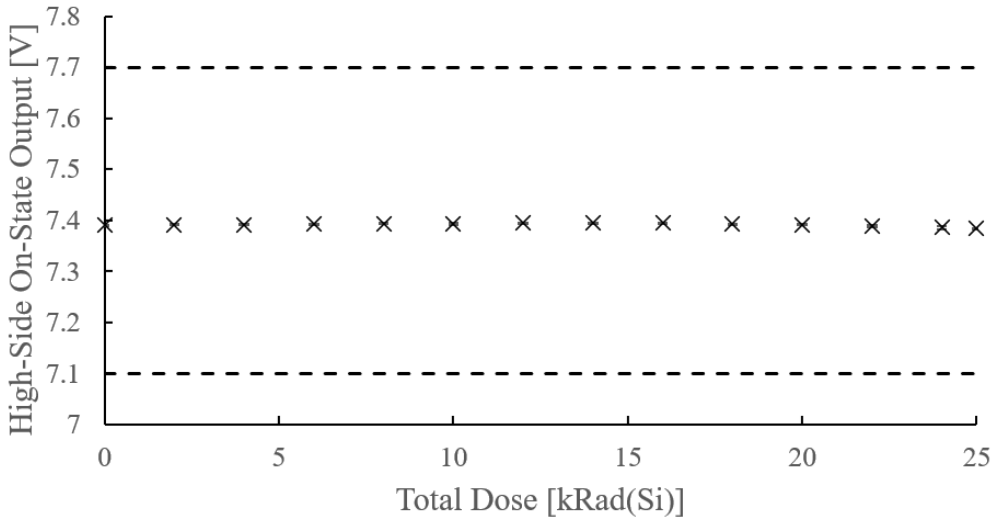

FIGURE 6.45: DGD054x High-Side On-State Output Voltage vs. Total Dose.

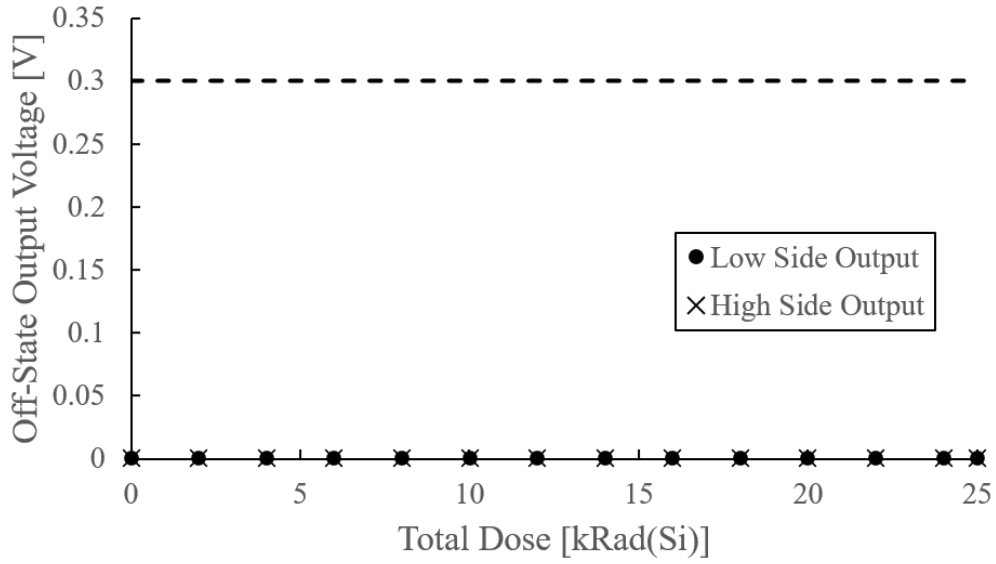

FIGURE 6.46: ISL784x Off-State Output Voltage vs. Total Dose.

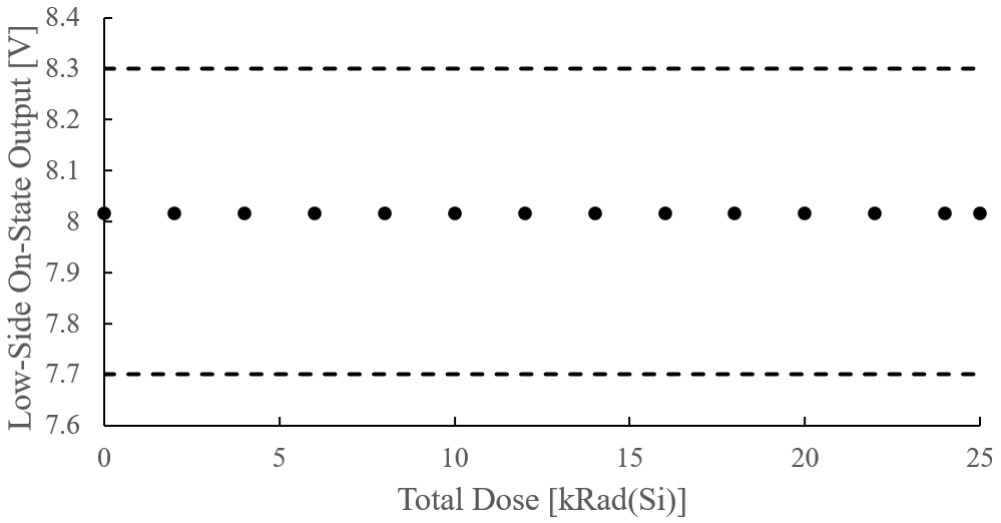

FIGURE 6.47: ISL784x Low-Side On-State Output Voltage vs. Total Dose.

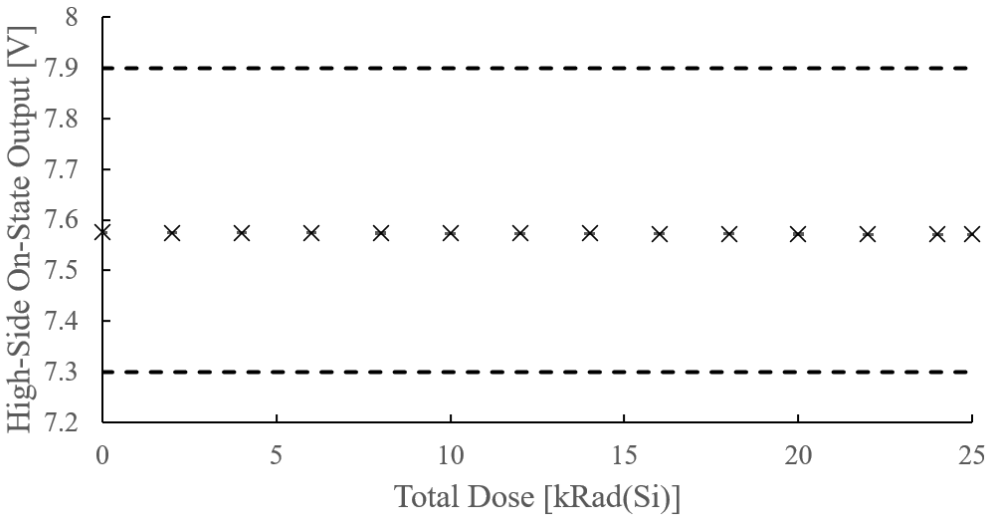

FIGURE 6.48: ISL784x High-Side On-State Output Voltage vs. Total Dose.

6.3 Instrumentation Amplifiers

I screened four amplifiers in the second test campaign following the methods detailed in section 5.3.3 and applied *PDM* analysis as described in chapter 3. Vendor-supplied electrical parameters for each part are listed in table XVIII alongside the part failure bounds. Gain failure bounds were set to at least 5% due to the tolerances of external components used to help set the gains, such as resistors. Pre-radiation control measurements obtained before testing were used to center the failure bounds. Figures 6.49 through 6.52 provide plots of the pre-radiation output values. The INA32x exhibited two separate groupings concerning the on-state output. The two groups had greater separation than the 5% bounds such that I had to give the groups their own failure ranges.

TABLE XVIII: Amplifier Electrical Parameters ([93], [94], [95], [96])

Part	**Gain**	**Off-State Failure Bounds**	**On-State Failure Bounds**
INA32x Case 1	$5 \pm 5\%$	1.19 to 1.32 V	4.20 to 4.64 V
INA32x Case 2	$5 \pm 5\%$	1.19 to 1.32 V	4.81 to 5.31 V
AD524	$5 \pm 20\%$	1.03 to 1.54 V	4.15 to 6.22 V
LM741	$5 \pm 5\%$	1.19 to 1.31 V	4.75 to 5.25 V
TSV911	$5 \pm 5\%$	1.19 to 1.31 V	4.75 to 5.25 V

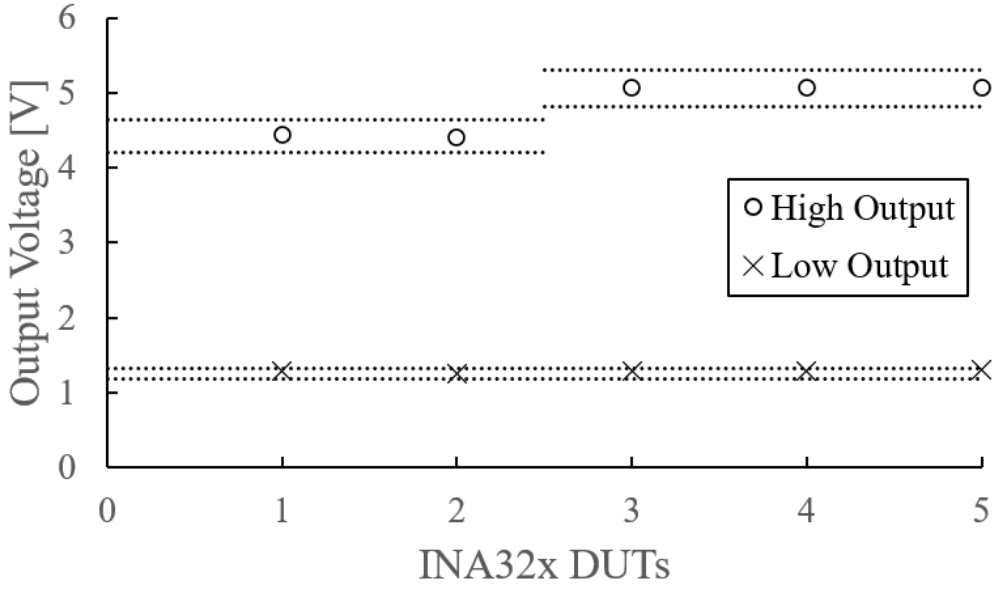

FIGURE 6.49: INA32x Control Data.

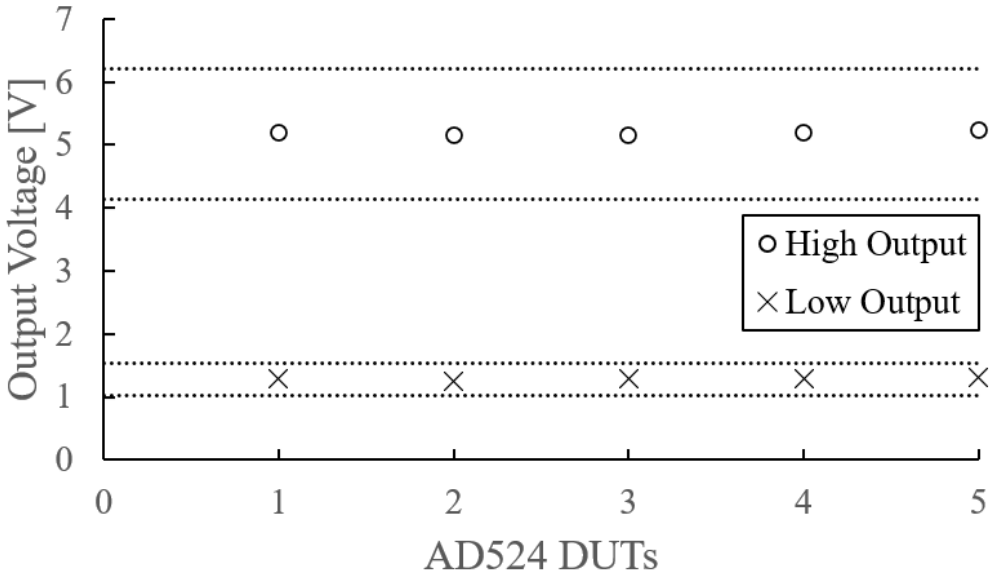

FIGURE 6.50: AD524 Control Data.

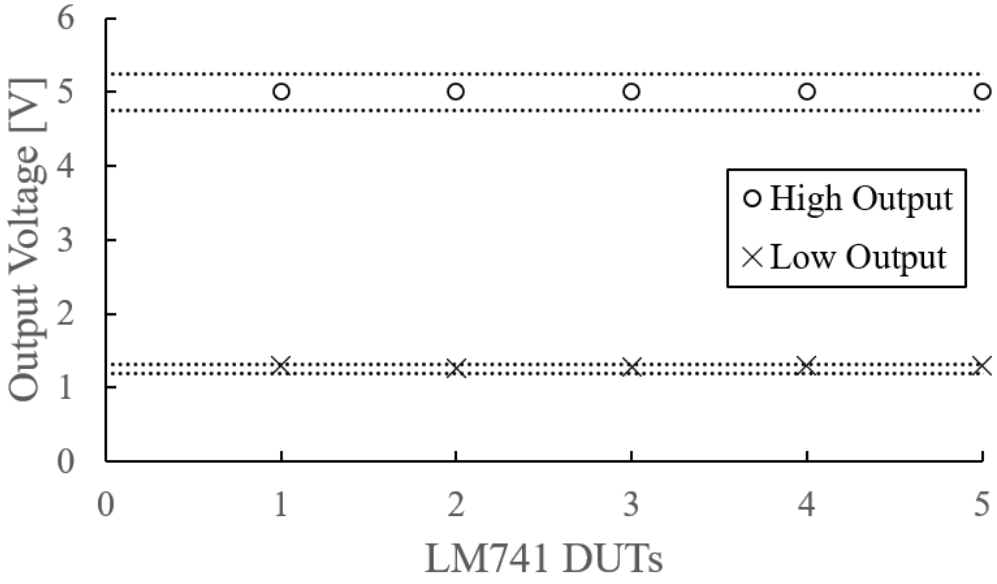

FIGURE 6.51: LM741 Control Data.

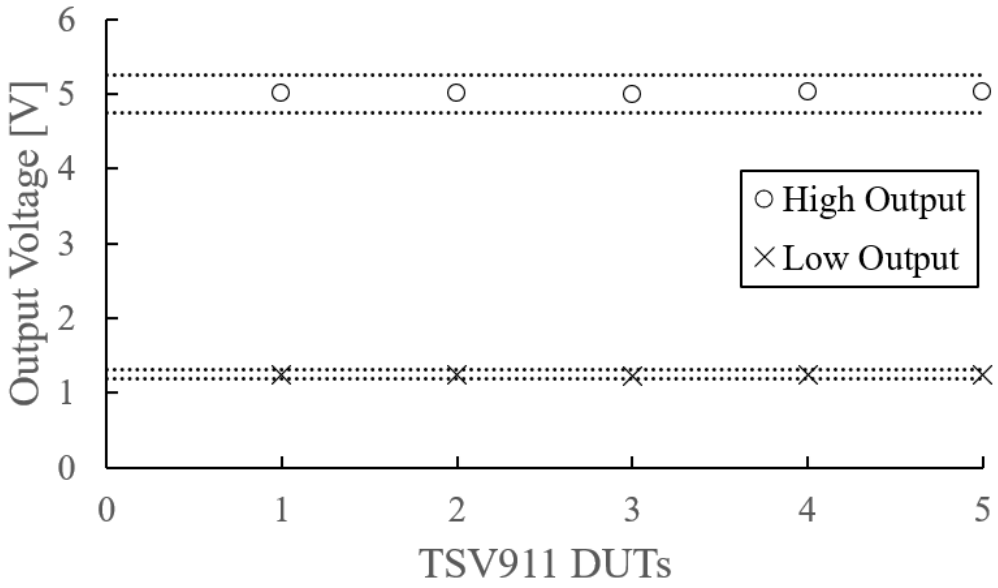

FIGURE 6.52: TSV911 Control Data.

Figures 6.53 through 6.60 provide the total dose plots for each part. I analyzed the test data according to the *PDM* method described in chapter 3 because the samples did not fail after 25 kRad(Si) while having test parameters known to degrade gradually. The INA32x was categorized as HCC-2, and the other components were categorized as HNC for the simulated five-year mission dose of 6.4 kRad(Si).

TABLE XIX: Instrumentation Amplifier Low Output TID Results

Part	$\overline{PAR_{Rad}}$	$\sigma_{ln(PAR_{RAD})}$	**Samples**	K_{TL}	*PCC*	*PDM*	**Category**
INA32x	1.244 V	0.0032	5	4.666	1.015	22.4	HCC-2
AD524	1.287 V	0.0172	5	4.666	1.083	815	HNC
LM741	1.246 V	0.0036	5	4.666	1.017	1382	HNC
TSV911	1.246 V	0.0095	5	4.666	1.045	114	HNC

TABLE XX: Instrumentation Amplifier High Output TID Results

Part	$\overline{PAR_{Rad}}$	$\sigma_{ln(PAR_{RAD})}$	**Samples**	K_{TL}	*PCC*	*PDM*	**Category**
INA32x	4.799 V	0.0200	5	4.666	1.098	10.4	HCC-2
AD524	5.186 V	0.0345	5	4.666	1.175	3685	HNC
LM741	5.007 V	0.0048	5	4.666	1.023	964	HNC
TSV911	5.014 V	0.0106	5	4.666	1.051	507	HNC

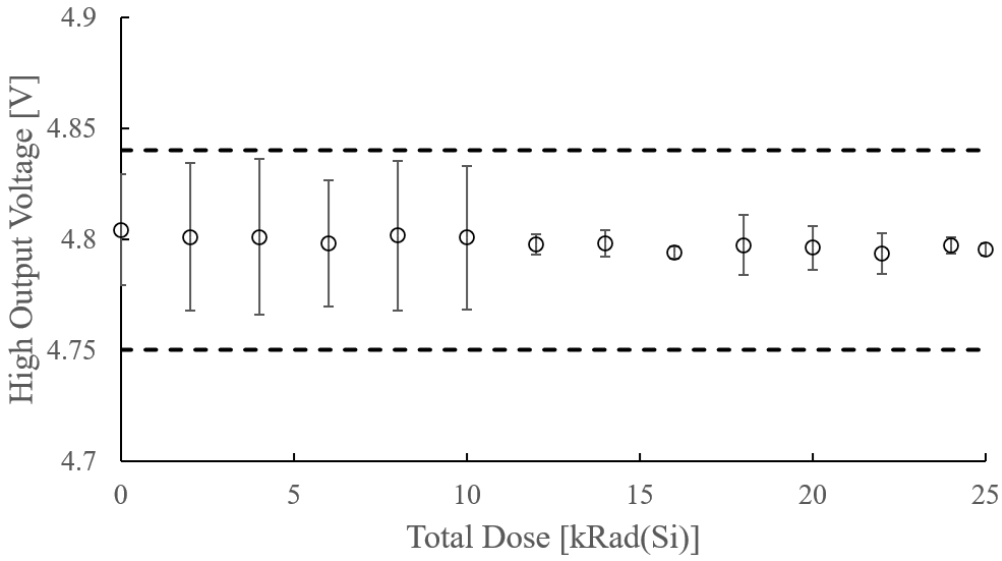

FIGURE 6.53: INA32x High Output Voltage vs. Total Dose. Horizontal lines indicate failure limits as shown in Table XVIII. Error bars represent one standard deviation from the mean within the part-to-part variation.

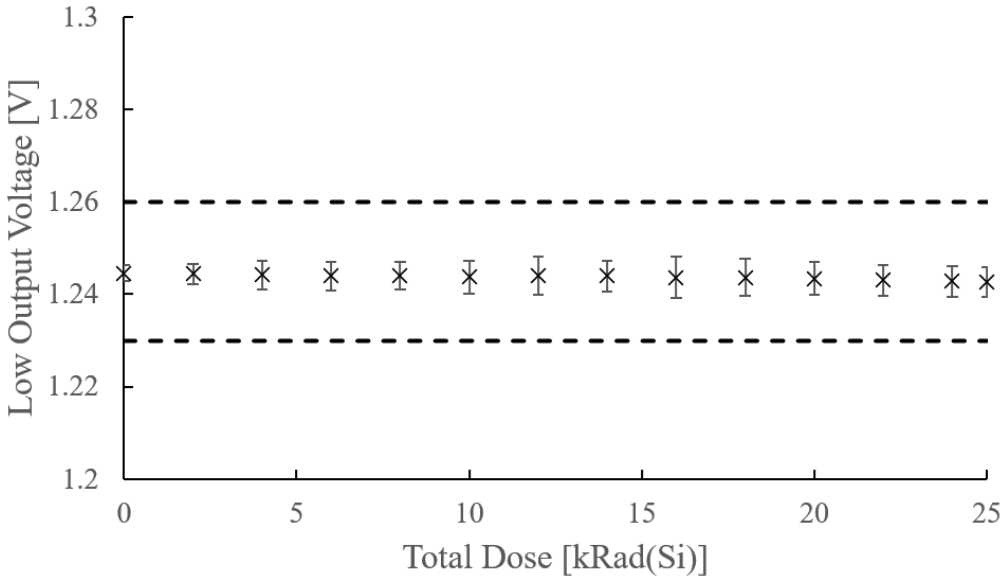

FIGURE 6.54: INA32x Low Output Voltage vs. Total Dose.

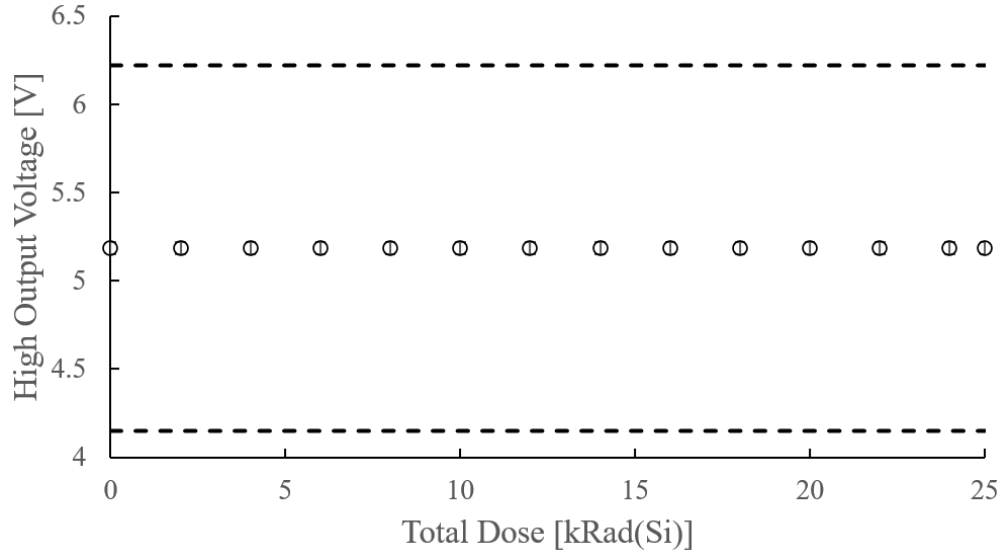

FIGURE 6.55: AD524 High Output Voltage vs. Total Dose.

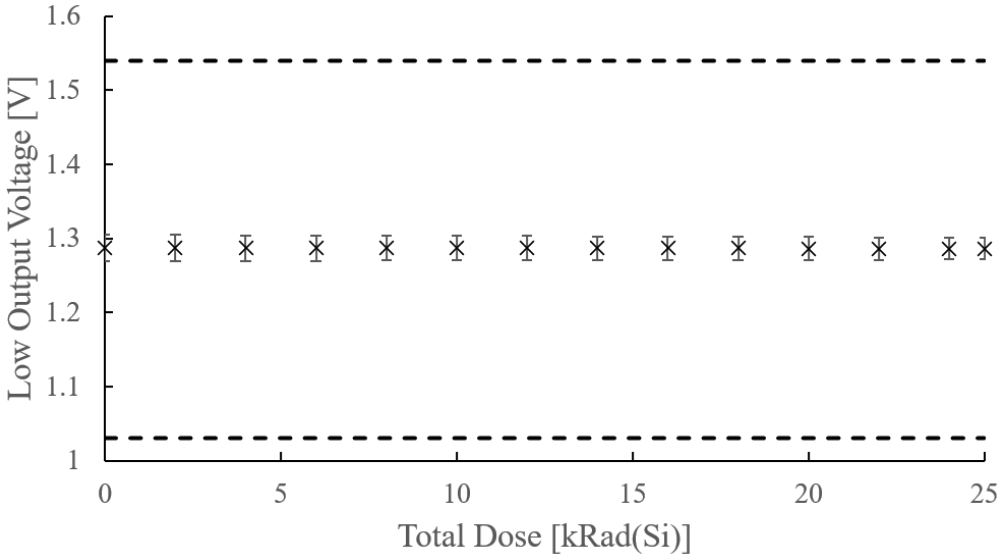

FIGURE 6.56: AD524 Low Output Voltage vs. Total Dose.

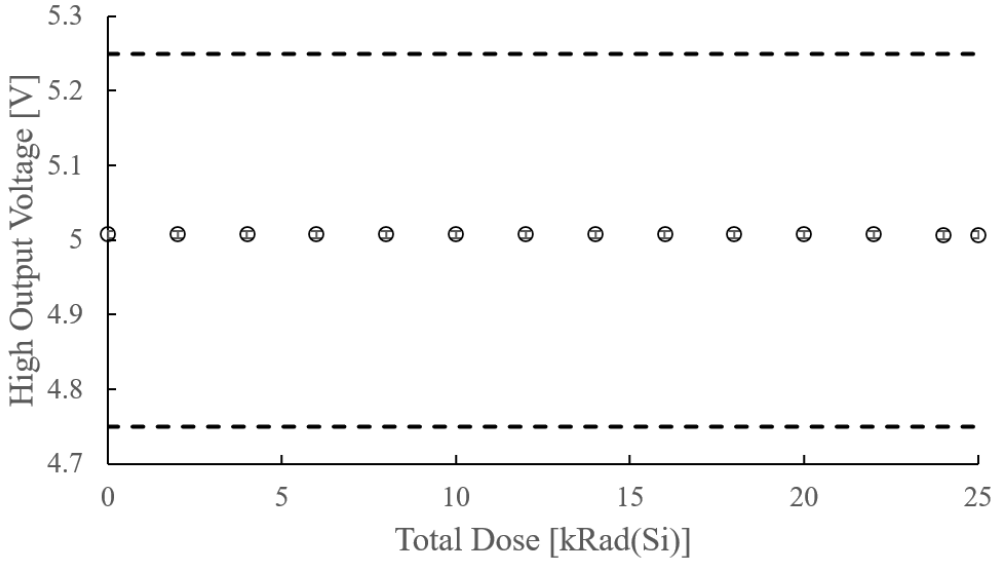

FIGURE 6.57: LM741 High Output Voltage vs. Total Dose.

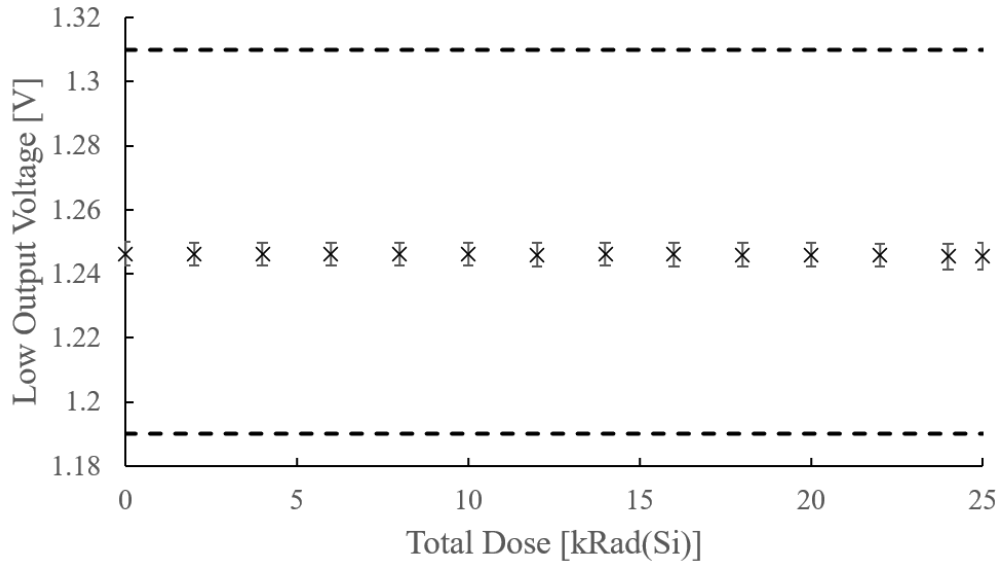

FIGURE 6.58: LM741 Low Output Voltage vs. Total Dose.

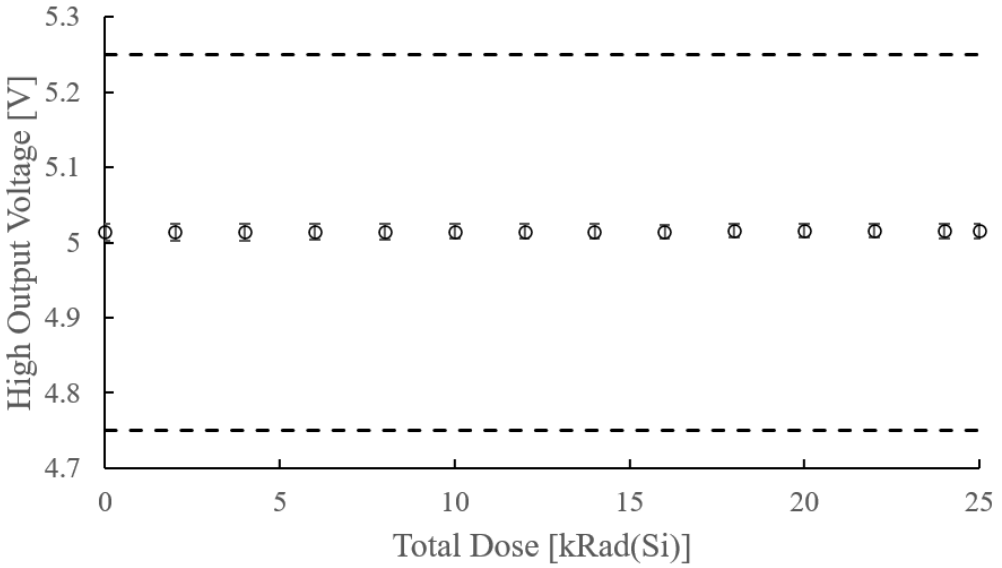

FIGURE 6.59: TSV911 High Output Voltage vs. Total Dose.

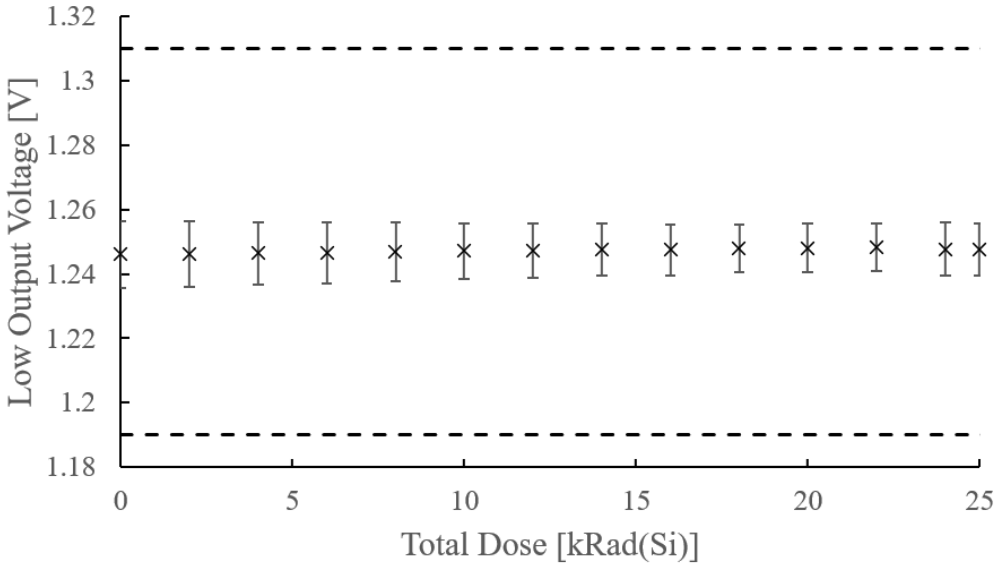

FIGURE 6.60: TSV911 Low Output Voltage vs. Total Dose.

6.4 Crystal Oscillators

I screened four crystal oscillators during the two test campaigns using the methods described in section 5.3.4. Table XXI displays the manufacturer limits for the part's test parameters, and figures 6.61 through 6.66 provide pre-radiation measurements for the four components. I tested the 7X-20 and 8W-13 in both test campaigns at different sample sizes.

TABLE XXI: Oscillator Electrical Parameters ([97], [98], [99], [100])

Parameter	**7X-20**	**8W-13**	**AU-12**	**AW-11**
Frequency	20 MHz	13 MHz	12.288 MHz	11.2896 MHz
Frequency Stability	±50 ppm	±50 ppm	±50 ppm	±50 ppm
Current Draw Limits	<10 mA	<10 mA	<6 mA	<15 mA
Frequency Limits	±1000 Hz	±650 Hz	±615 Hz	±565 Hz

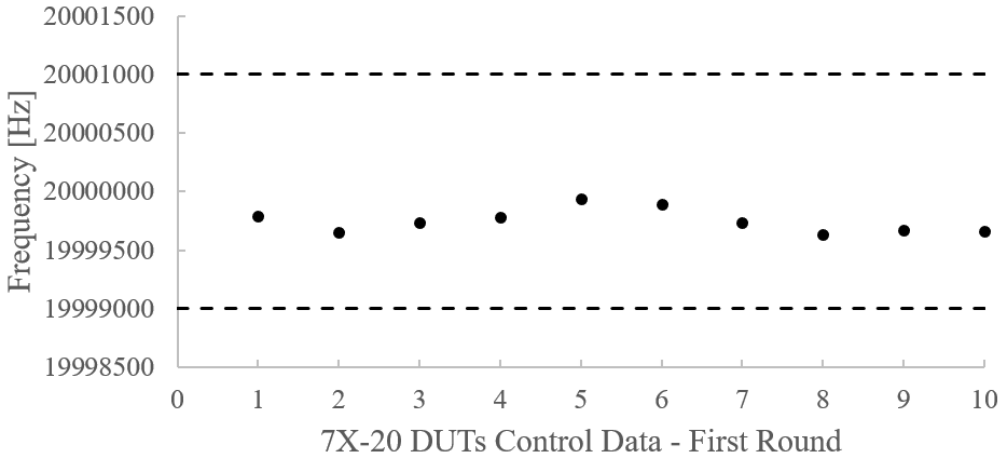

FIGURE 6.61: First Round 7X-20 Control Data.

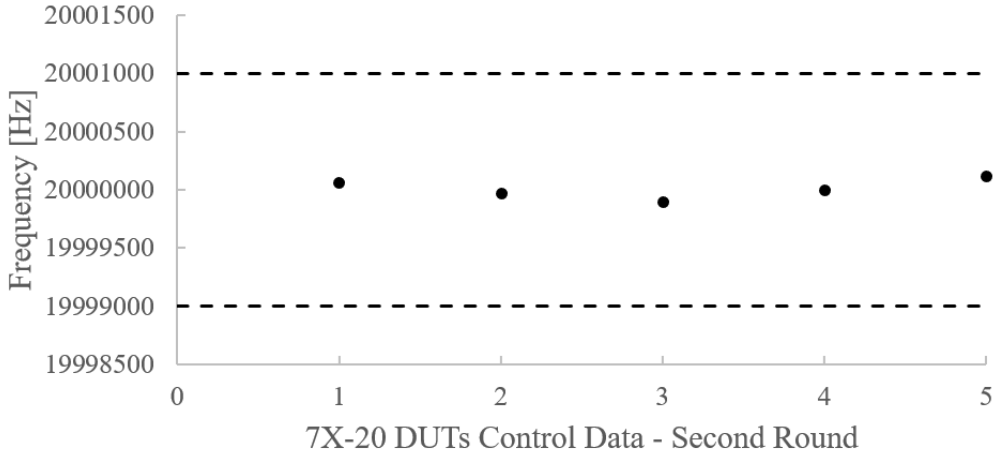

FIGURE 6.62: Second Round 7X-20 Control Data.

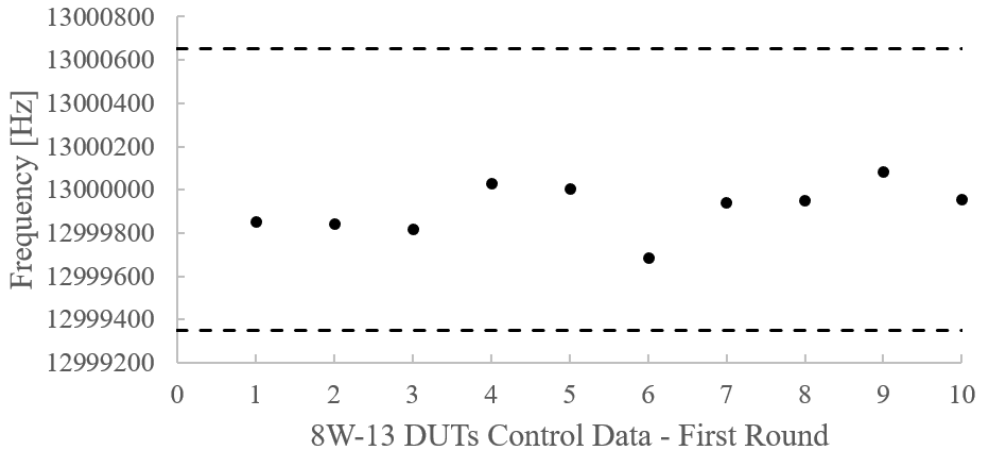

FIGURE 6.63: First Round 8W-13 Control Data.

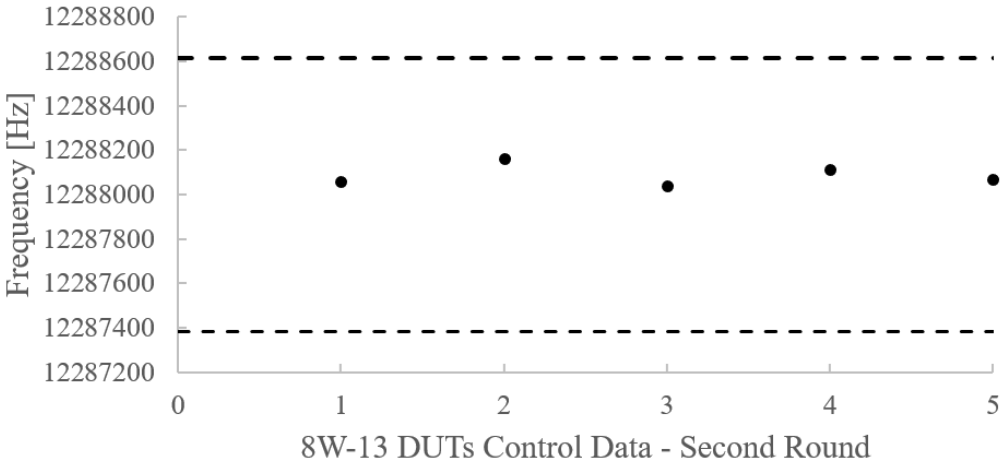

FIGURE 6.64: Second Round 8W-13 Control Data.

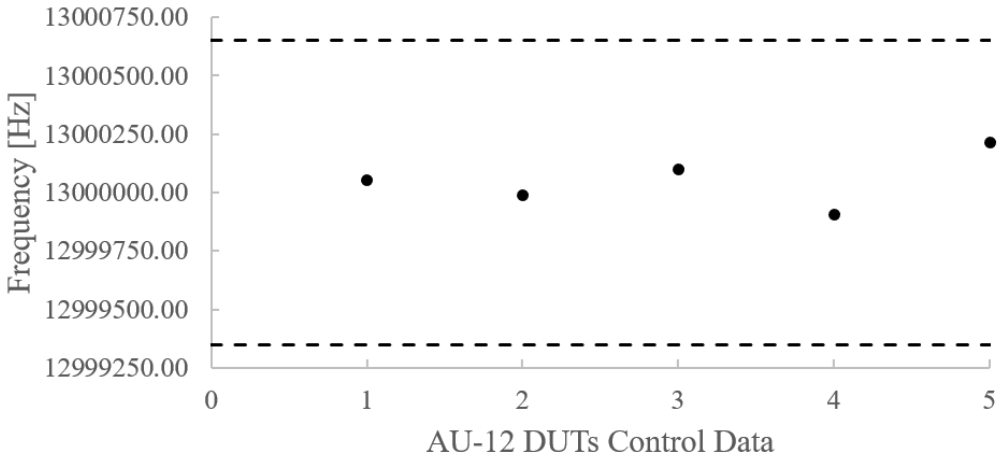

FIGURE 6.65: AU-12 Control Data.

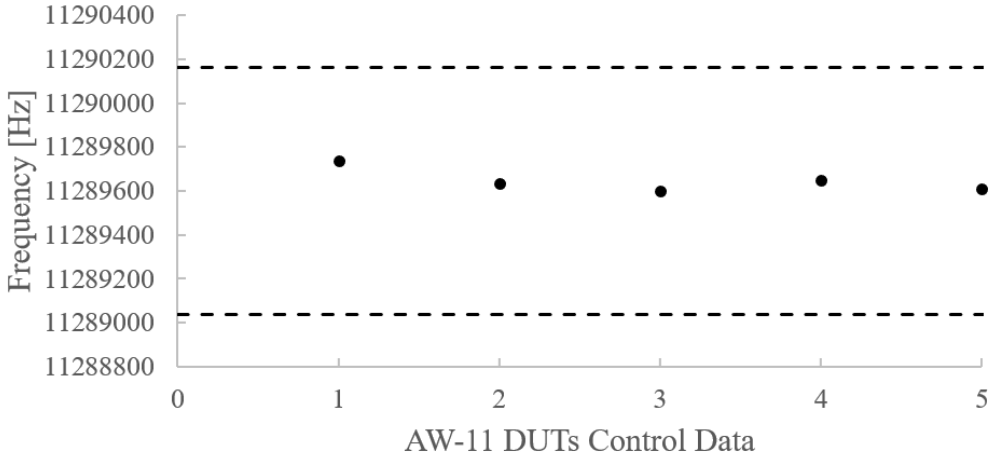

FIGURE 6.66: AW-11 Control Data.

The total dose response curves are provided in figures 6.67 through 6.76. I analyzed the data following the *PDM* methods described in chapter 3 because I expected the parts to gradually degrade as the dose increased. The results of each component are listed in tables XXII and XXIII. None of the parts failed during testing. However, a non-linear rise in the 7X-20's current draw was observed after 16 kRad(Si) but remained within manufacturer limits.

Shifts in oscillator frequency remained well within the boundaries of the frequency stability of each component. I found the frequency shift on the oscillators after radiation statistically insignificant using a paired t-test (as described in section 3.7) at 99% confidence as shown in table XXIV. Each part received HCC-2 hardness categorization.

TABLE XXII: Crystal Oscillator Frequency TID Results

Part	$\overline{PAR_{Rad}}$	$\sigma_{ln(PAR_{RAD})}$	**Samples**	K_{TL}	*PCC*	*PDM*	**Category**
7X-20 (1^{st})	19999784 Hz	5.10×10^{-6}	10	3.532	1	35.4	HCC-2
7X-20 (2^{nd})	20000043 Hz	4.66×10^{-6}	5	4.666	1	29.7	HCC-2
8W-13 (1^{st})	12999865 Hz	4.96×10^{-6}	10	3.532	1	10.7	HCC-2
8W-13 (2^{nd})	13000036 Hz	4.01×10^{-6}	5	4.666	1	39.2	HCC-2
AU-12	12288069 Hz	2.55×10^{-6}	5	4.666	1	35.4	HCC-2
AW-11	11289660 Hz	2.01×10^{-6}	5	4.666	1	32.9	HCC-2

TABLE XXIII: Crystal Oscillator Current TID Results

Part	$\overline{PAR_{Rad}}$	$\sigma_{ln(PAR_{RAD})}$	**Samples**	K_{TL}	*PCC*	*PDM*	**Category**
7X-20	5.692 mA	0.0090	5	4.666	1.04	1025	HNC
8W-13	5.228 mA	0.0209	5	4.666	1.10	85	HCC-2
AU-12	5.900 mA	0.0083	5	4.666	1.04	2716	HNC
AW-11	5.382 mA	0.0051	5	4.666	1.02	137	HNC

TABLE XXIV: Crystal Oscillator Frequency Paired T-Tests

Part	μ_{Diff}	σ_{Diff}	**Samples**	T_{Crit}	**Result**	**Significant**
7X-20 (1^{st})	-129 Hz	180 Hz	10	3.250	2.148	No
7X-20 (2^{nd})	-69 Hz	188 Hz	5	4.604	0.739	No
8W-13 (1^{st})	80 Hz	135 Hz	10	3.250	1.784	No
8W-13 (2^{nd})	-33 Hz	203 Hz	5	4.604	0.329	No
AU-12	35 Hz	62 Hz	5	4.604	1.140	No
AW-11	-18 Hz	186 Hz	5	4.604	0.198	No

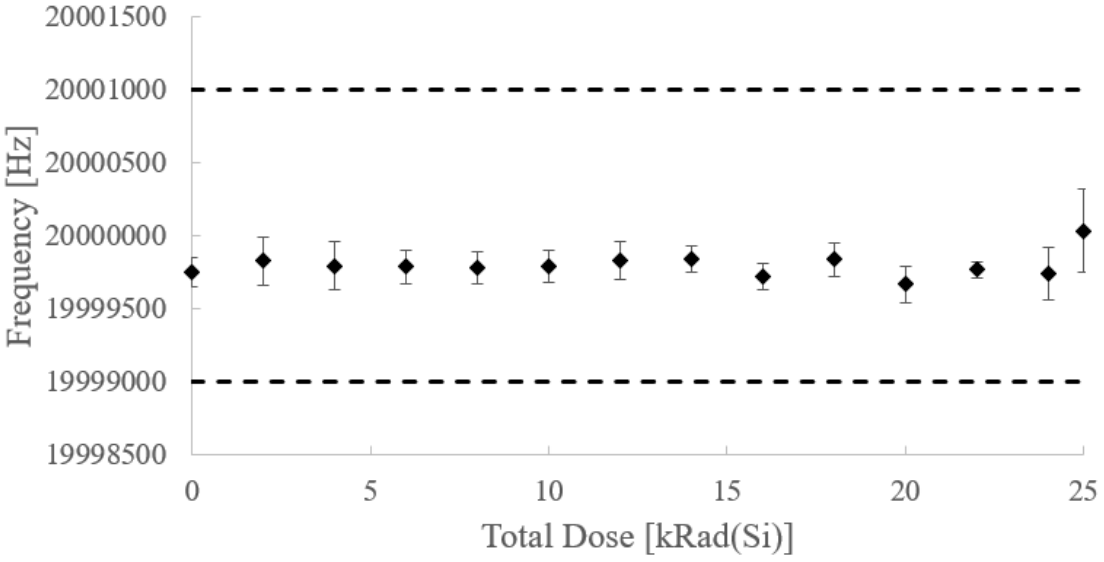

FIGURE 6.67: First Round 7X-20 Frequency vs. Total Dose. Error bars represent one standard deviation from the mean within part-to-part variation. Horizontal lines indicate failure limits.

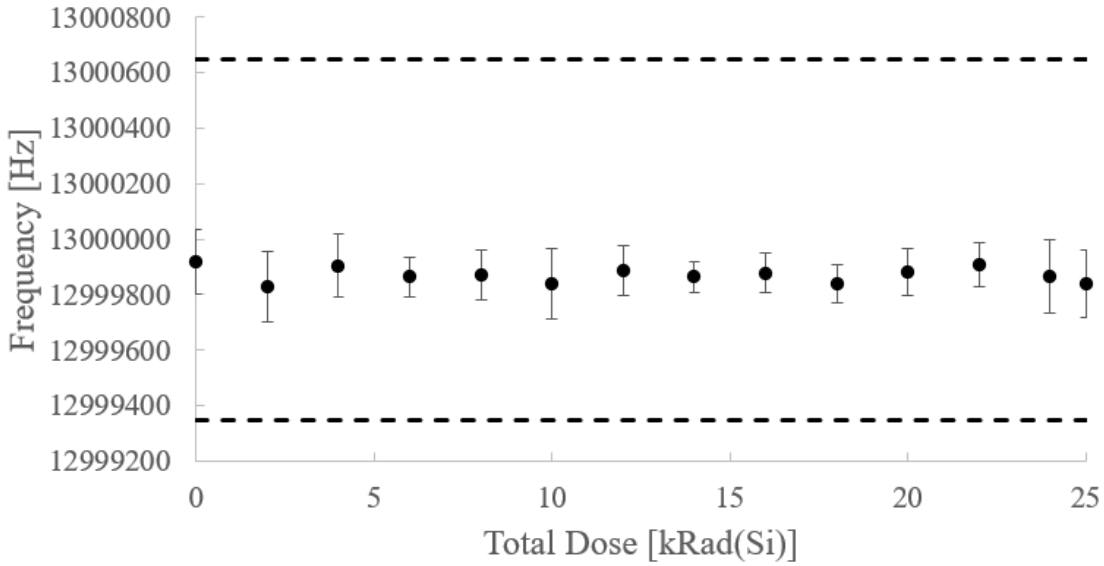

FIGURE 6.68: First Round 8W-13 Frequency vs. Total Dose.

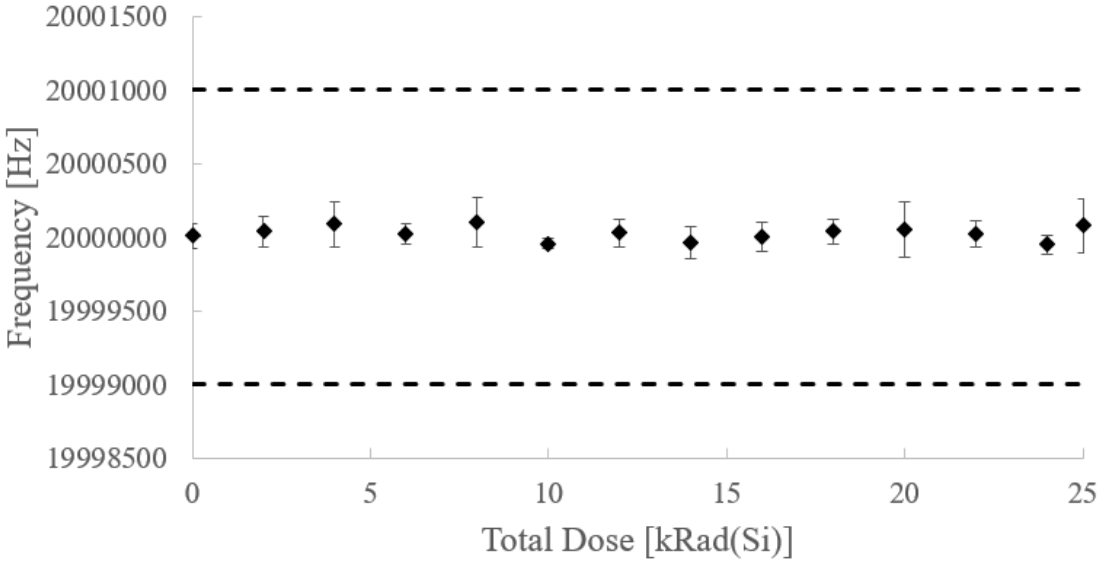

FIGURE 6.69: Second Round 7X-20 Frequency vs. Total Dose.

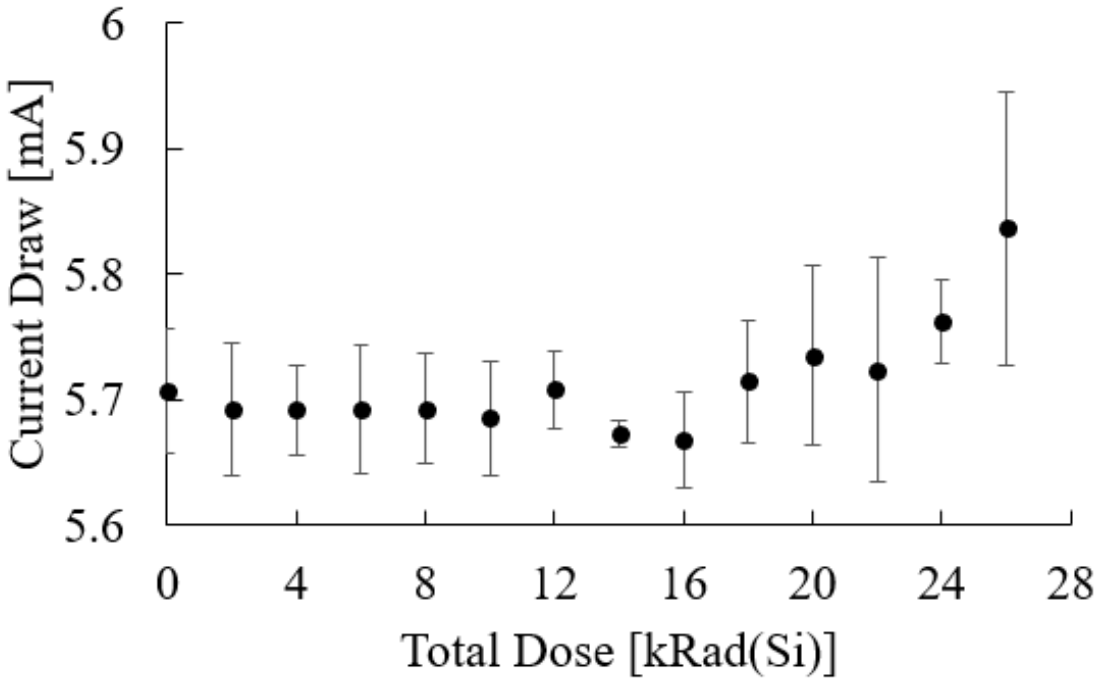

FIGURE 6.70: Second Round 7X-20 Current Draw vs. Total Dose. Failure limit lies at 10 mA.

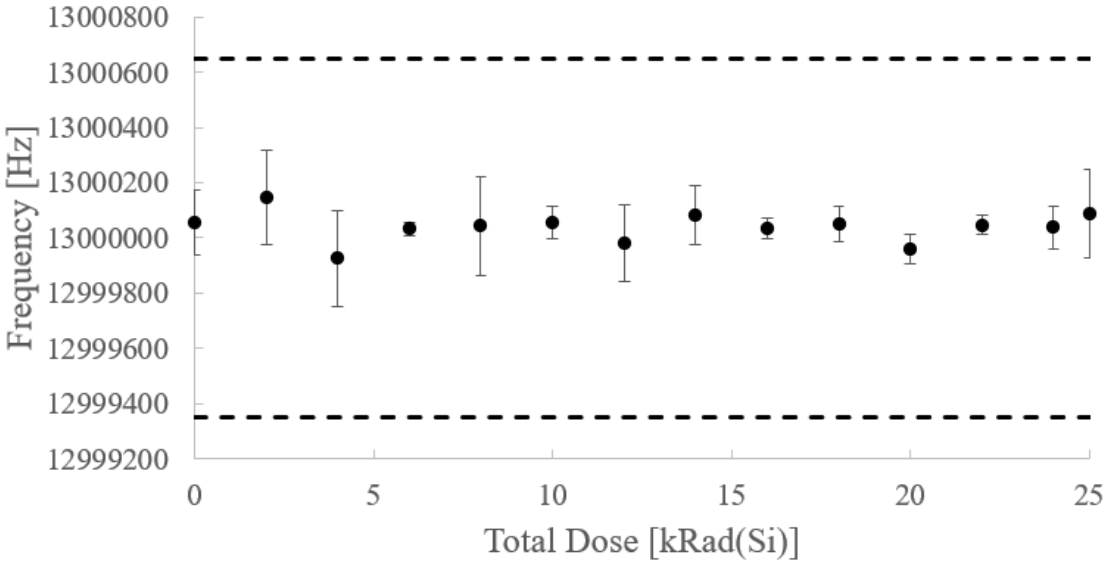

FIGURE 6.71: Second Round 8W-13 Frequency vs. Total Dose.

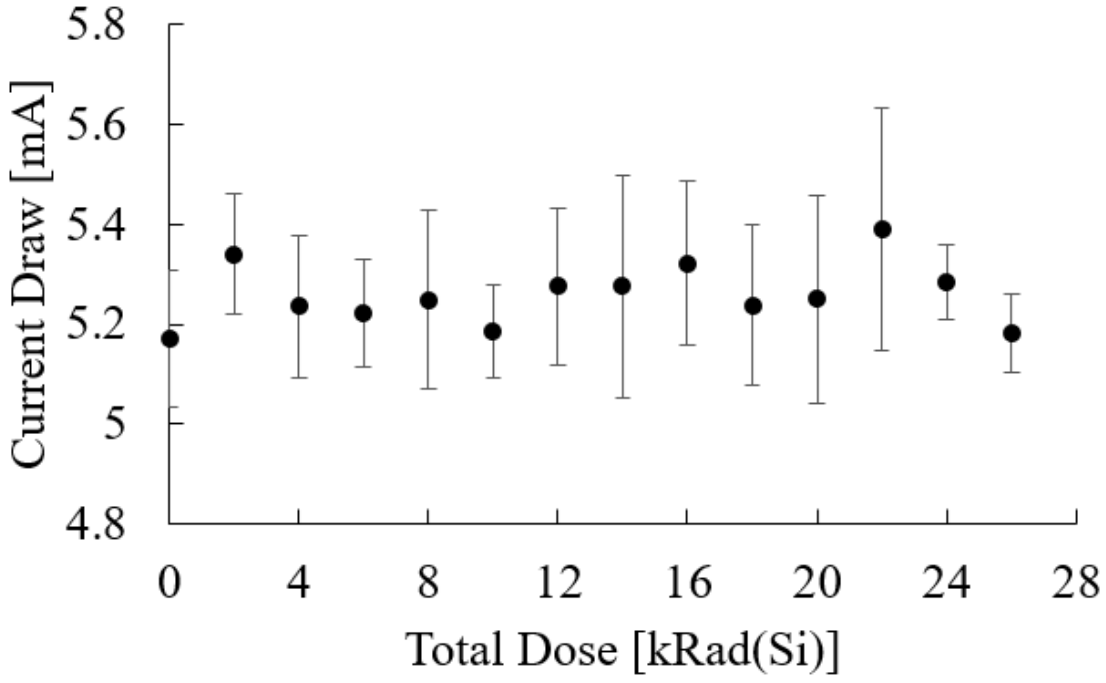

FIGURE 6.72: Second Round 8W-13 Current Draw vs. Total Dose. Failure limit lies at 10 mA.

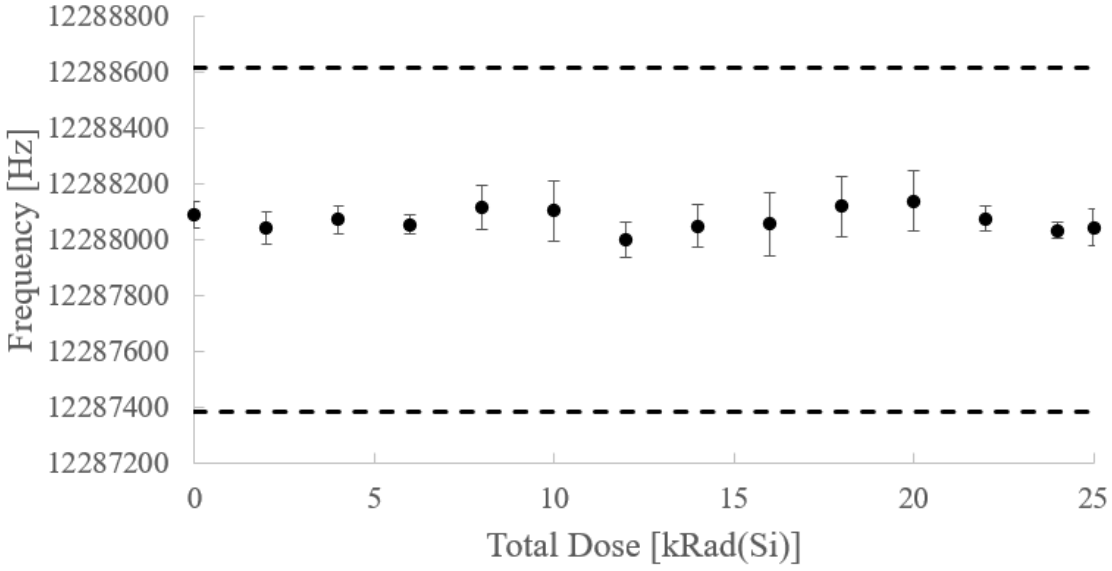

FIGURE 6.73: AU-12 Frequency vs. Total Dose.

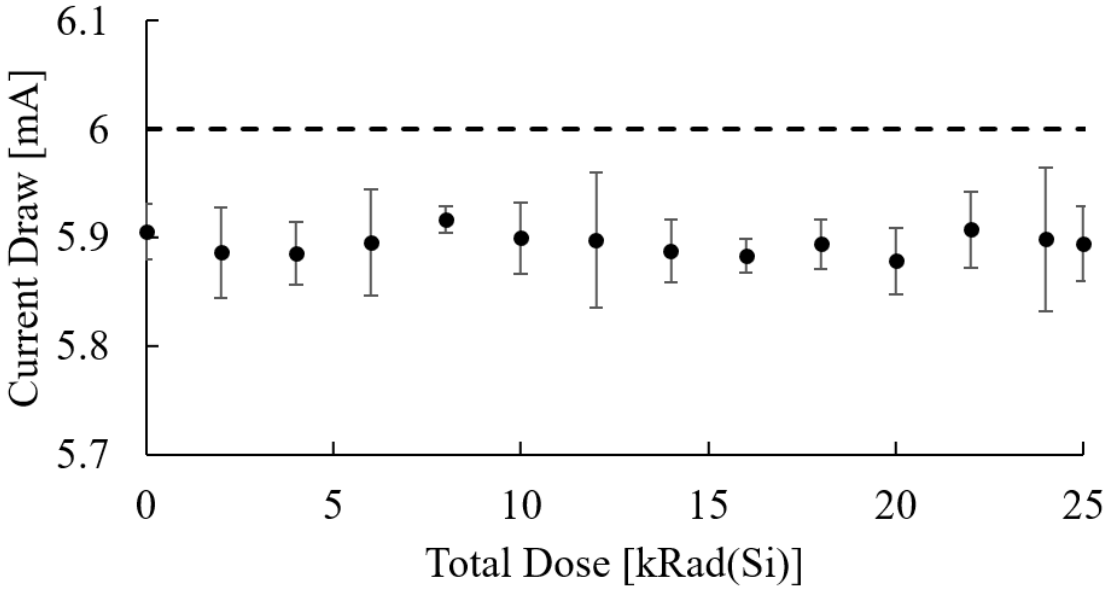

FIGURE 6.74: AU-12 Current Draw vs. Total Dose. Failure limit lies at 6 mA.

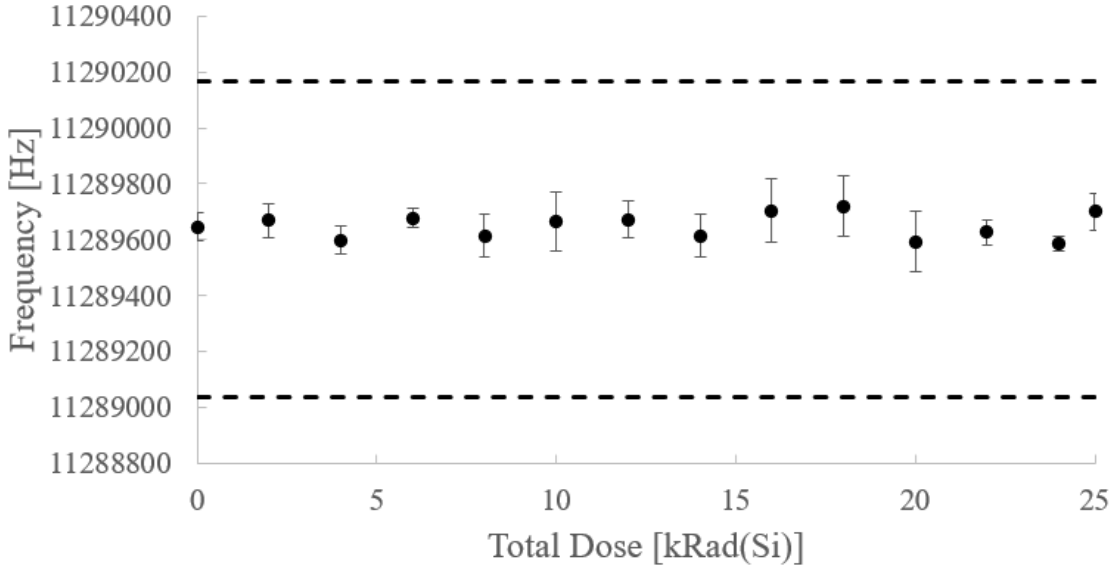

FIGURE 6.75: AW-11 Frequency vs. Total Dose.

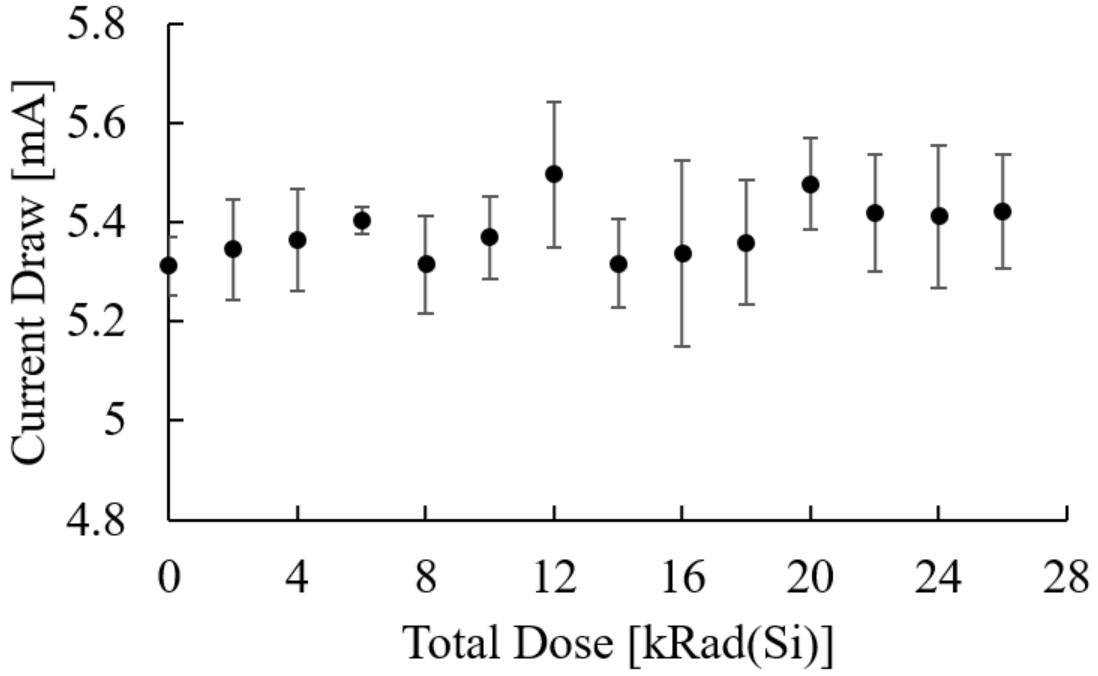

FIGURE 6.76: AW-11 Current Draw vs. Total Dose. Failure limit lies at 15 mA.

6.5 Digital Memory

I performed SEE testing on the four digital memory components listed in table XXV with the methods described in section 5.3.5. I screened five DUT boards and two backup boards using a 105 MeV proton beam. The screening campaign first conducted static testing on five DUT boards. Afterwards, enough scheduled time remained to perform two additional tests collecting data under the dynamic test configuration. The test bench successfully wrote the checkerboard 0x55 pattern to each memory component in both test cases. No SEE bit flip accumulation occurred on any samples during static testing, and no bits became stuck during dynamic testing. These findings complement the 2019 results published by the Aerospace Corporation for the MRAM component [101].

TABLE XXV: Tested Digital Components

Part Category	**Digital DUT**	**Memory Type**
SPI Memory	MR25Hxx	MRAM
SPI Memory	MB85RS2	FRAM
SPI Memory	S25FL512	NOR Flash
Quad-SPI Memory	S25FL032	NOR Flash

The test results indicated that each component had a LET threshold above 8 MeVcm^2/mg. The lack of any upsets with the proton beam prevented use of the figure of merit method to estimate the minimum LET threshold. Further testing using a heavy ion beam is needed to determine the LET threshold required to induce upsets within these parts.

7 Discussions and Recommendations

This chapter discusses the results of my research as it relates to my three hypotheses. I first summarize the test results and discuss commercial parts becoming a viable option for use in space. Next, I overview the platform accomplishing its three design goals: reusability, reconfigurability, and cost-effectiveness. Finally, I discuss my recommended additions to the current radiation screening guidelines as an updated decision tree algorithm.

7.1 Commercial Part Feasibility

To evaluate the feasibility of using commercial parts in space, I conducted two screening campaigns. The two campaigns tested 22 commercial components relevant to future spacecraft systems being designed by Magellan Aerospace. My screening campaign results showed that most of the commercial parts tested are suitable for use in the radiation environment of a five-year mission operating in low Earth orbit. The DMNH6008 N-channel MOSFET and the TPS2822 half-bridge driver were the only parts which failed during the two campaigns.

Overall, the high success rate of the screened parts suggested that using modern COTS components has become a viable option for designing future LEO space mission electronics. However, it is still important to carefully select and test candidate parts to ensure their suitability for the specific mission requirements. Testing continues to be needed to validate the use of commercial parts for missions, especially missions with more stringent radiation requirements than the five-year LEO mission considered for this research.

7.2 Test Platform Performance

This section discusses the performance of the test platform in terms of the three design goals detailed in section 4.1. First, I discuss the reusability of the test bench and how it survived multiple rounds of testing. Next, I overview how the test platform was reconfigurable thanks to using DUT boards and software initialization files. Finally, I discuss how using a multi-slot test platform is cost-effective thanks to reduced cooldown times and the previous design goals.

Reusability

The motherboard of the test platform withstood 17 screening rounds during the second test campaign. Ten of those screening rounds were in analog test mode, irradiating the components to 25 kRad(Si) each time. The motherboard survived a significant radiation dose, thanks to the design keeping critical electronic circuitry outside the beam line's path. The tested parts were populated onto disposable DUT boards and plugged into several slots on the motherboard. This setup facilitated a reusable screening setup while performing destructive testing.

Reconfigurability

The test platform was able to successfully be reconfigured to screen 21 candidate parts from various manufacturers. The system's reconfigurability was achieved through plug-and-play DUT boards and user-provided configuration files, which initialized the analysis software with the correct failure limits. Each DUT board design contained a unique complement of parts. The software initialization settings for each DUT board had the following part information: part name, part category, and failure limits for each test parameter. The configuration files also provided beam settings, including the total dose, fluence, and beam energy.

Cost Effectiveness

The test bench and automated analysis software proved to be a capable testing platform, successfully screening two different analog DUT boards with varying types of parts. I completed a more rapid second test campaign because the test bench had multiple slots which allowed the test setup to switch quickly between DUT boards. The test setup used a two-axis robotic centering arm to move new DUTs into the beam line area without needing a cooldown process. The current test bench design used four slots, two for analog parts and two for digital components.

The first test bench design was only able to utilize one slot due to issues in the design. This problem allowed for a quantitative comparison of needed screening time between a single board setup versus a test platform that utilizes multiple slots. The first screening campaign required a cooldown between each test. These cooldowns contributed an additional 20 to 30 minutes to each test. The average screening time was two hours per DUT board. In comparison, the second screening campaign averaged 1.15 hours per DUT board. A 30-minute cooldown was needed every four tests to swap out new DUTs. The tests took 1.28 hours per board when factoring in the cooldowns. The second test bench design reduced the average screening time by 36.5%. DUT boards plugging into a parent test bench additionally provided the benefit of not needing to unplug the power supply and computer setup between tests to swap to new parts.

The test platform's reconfigurability and reusability contributed to the platform being cost-effective. Data on various parts were obtainable thanks to the test setup's ability to be easily reconfigured, eliminating the need to design and purchase new test platforms when one wishes to screen a new part candidate. The test bench's reusability further contributed to its cost-effectiveness by eliminating the need to acquire multiple motherboards. Only the inexpensive DUT boards required multiple purchases. The test platform further reduced the cost associated with radiation effects screening by allowing

users to screen critical components isolated from a more extensive system. This isolation protected against purchasing an entire subsystem to destructively test the parts of interest.

Overall, the test platform's cost is laid out in table XXVI, which provides an overview of the recurrent and non-recurrent expenses associated with screening commercial parts at the TRIUMF facility at the time I performed this research. Commercial component screening campaign at five samples resulted in a recurrent cost of about $7,400 after an initial test platform purchase of about $3,450. In contrast, table XXVII lists the costs of purchasing radiation-hardened components from several manufacturers. While not comprehensive, these prices provide an overview of the magnitude of radiation-hardened part purchases. The screening campaign's goal is to qualify future lot purchases of a commercial component. The costs associated with this task were less expensive than simply buying a manufacturing lot of a radiation-hardened part.

TABLE XXVI: Test Platform Costs

Recurrent		**Non-Recurrent**	
Item	**Unit Cost**	**Item**	**Unit Cost**
Average DUT	$169	Motherboard	$1932
Screening Beam Time	$1312	Omega 1608G	$639
		Power Supply	$886
Recurrent Total	$1481	**Non-Recurrent Total**	$3457

TABLE XXVII: Radiation-Hardened Component Prices ([102], [103], [104], [105], [106])

Part Family	**Part Manufacturer**	**Unit Cost**	**Purchase Lot Quantity**
MOSFETs	International Rectifier	$683	1,000
Half-Bridge Drivers	Texas Instruments	$255	100
Crystal Oscillators	STMicroelectronics	$69	1,000
In-Amps	Texas Instruments	$242	100
Digital Memory	Infineon	$3,200	1,000

7.3 Test Plan Inefficiencies

My screening campaigns were planned according to the industry guidelines which ensure parts could be certified to MIL-STD-883. After conducting the campaigns, I identified that a couple of the guidelines could be improved based on modern electronics exhibiting minimal part-to-part variation. Firstly, the sample size selection guidelines recommend using ten or more DUTs as a best practice without providing methods to better tune the number of samples. Additionally, the parametric design margin uses a ratio between the parameter value at the mission dose and the value for failure. This ratio provides unusable design margin results for modern parts which have tighter bounds on manufacturer specifications. As part of my research, I developed additions to the current guidelines which provide an updated *PDM* equation and a new method for tuning sample sizes based on tolerance interval statistics. The remaining sections of this chapter discuss these proposed additions.

7.4 Tolerance Similarity Factor

Minimal part-to-part variance in the first screening campaign provided me with data which allowed me to be confident in running a second screening campaign with a reduced sample size. The second campaign's data showed equivalent results to the first round of testing. Modern COTS components have trended towards less variation in part characteristics due to the stricter manufacturing methods needed to accommodate their smaller sizes [107]. In particular, automotive-grade components have minimal variances to satisfy the necessary lot requirements to obtain certification. This reduction in variance between parts reduced the sample size's impact on tolerance interval calculations. My algorithm uses this property to reduce the number of samples in a screening campaign from the often-used ten plus control suggestion.

To aid in this endeavour, I propose a new tolerance similarity factor Ξ_T, which compares how similar the tolerance interval bounds are between two different sample sizes as shown in equation 7.1. The variable X_{pc_N} represents the tolerance bound for a part characteristic or failure dose at ten or more samples, and X_{pc_n} represents the tolerance bound for a part characteristic or failure dose at a reduced sample size.

$$\Xi_T = 100 \times \left(1 - \frac{|X_{pc_n} - X_{pc_N}|}{X_{pc_N}}\right) \tag{7.1}$$

With this factor, one can pick a similarity limit and select the smallest acceptable sample size that still provides a tolerance interval that closely matches what one would get using the larger, more conservative sample size. For example, I tested the BUK7M MOSFET with five samples in the second screening campaign and obtained statistically similar results as my first screening campaign, which used ten samples.

In the first screening campaign, the BUK7M's gate threshold voltage (V_{th}) degraded to a mean value of 2.98 V with a standard deviation of 0.052 V at the mission dose requirement of 6.5 kRad(Si). The second screening campaign showed similar degradation with a mean value of 2.96 V and a standard deviation of 0.051 V despite using five samples. Tolerance intervals for both sample size cases are visualized in figure 7.1. I set the confidence and probability values to 90% and 99%, respectively. In either case, the lower bound of the tolerance interval remained well above the manufacturer-specified limit of 2.4 V. The lower bounds were 2.80 V and 2.72 V for the ten and five samples, respectively. Calculating Ξ_T for this parameter provided a tolerance interval similarity of 97%. This similarity factor provides my algorithm with a tool to better determine small sample sizes now that modern electronics often exhibit minimal part-to-part variation, as shown with the BUK7M.

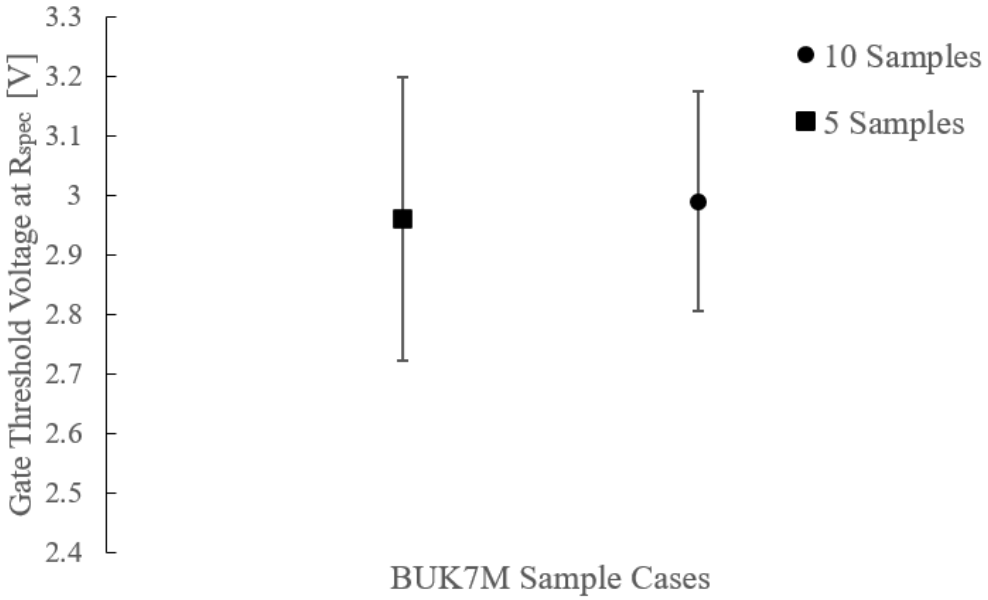

FIGURE 7.1: Tolerance intervals for BUK7M V_{th} at R_{Spec}. The first test campaign screened ten samples and the second campaign screened five.

7.5 Proposed Change to the Parametric Design Margin

The *PDM* equation, as provided in equation 3.10, is a simple ratio between a parameter's value at the mission dose and its failure limit. This ratio can be shown to provide unusable design margin results when tightening a part's specification limits. For example, a hypothetical component may have a current draw limit of 50 mA while operating at 6 mA normally. After irradiation to a specified dose, the current draw increases to 6.2 mA. Using the original design margin equation yields a *PDM* of eight.

In contrast, if this component's current draw limit was 10 mA, the *PDM* of even the original value before irradiation is less than two. I instead propose a new parametric design margin in equation 7.2. This equation instead compares the magnitude of the parametric shift from its initial value to the magnitude of the change needed to induce failure. This equation yields design margins that better reflect a part's radiation tolerance when working with small specification limits. In the provided example, The *PDM* for

the 50 mA and 10 mA cases are 220 and 20, respectively.

$$PDM = \frac{PAR_{Init} - PAR_{Fail}}{PAR_{Init} - \overline{PAR_{Rad}}} \tag{7.2}$$

7.6 Test Planning Algorithm

This section provides a complete description of my proposed test planning algorithm, which integrates my updated *PDM* equation, tolerance intervals, and my proposed tolerance similarity factor into the current planning procedure following industry guidelines. As mentioned in section 7.4, I used the data obtained during my first screening campaign and determined it would be feasible to reduce the sample size for the next round of testing. The variance between parts was minimal, producing similarly constrained tolerance intervals for part failure or parameter data even after I reduced the sample size in the second campaign. I utilized this finding to target a particular aspect of radiation effects test plans: The small sample selection procedures. My proposed additions to the test planning procedure provide a more comprehensive sample size selection algorithm than the commonly stated best practices of ten samples and one control.

My first significant addition implements my tolerance similarity factor. The new procedure performs tolerance interval calculations on the part variance control data before testing. Afterwards, a minimum acceptable Ξ_T is chosen. Next, I propose calculating the tolerance similarity between the best practice value and each sample size below it. The smallest sample size with an acceptable similarity score is then selected for the screening campaign. The flowchart provided in figure 7.2 visualizes these new additions to the planning procedure.

My additions to the *RDM* and *PDM* flowcharts propose using tolerance interval statistics to aid the hardness assurance categorization. After determining that the mean value for

radiation-induced failure or parameter shift provides an acceptable design margin result, I propose calculating a tolerance interval around the mean and checking that the interval bounds also give good design margin values. A part with good design margin results at its tolerance interval bounds can then be compared against the part characterization criterion to determine whether it requires lot testing. My final addition to the *PDM* procedure swaps out the old design margin equation for the one provided in equation 7.2. Visualizations of the updated flowcharts are provided in figures 7.3 and 7.4. Overall, the proposed updates provide researchers with a better tool to select small sample sizes in future radiation effects test plans when working with modern electronics whose part-to-part variances have significantly reduced in the last decade [107].

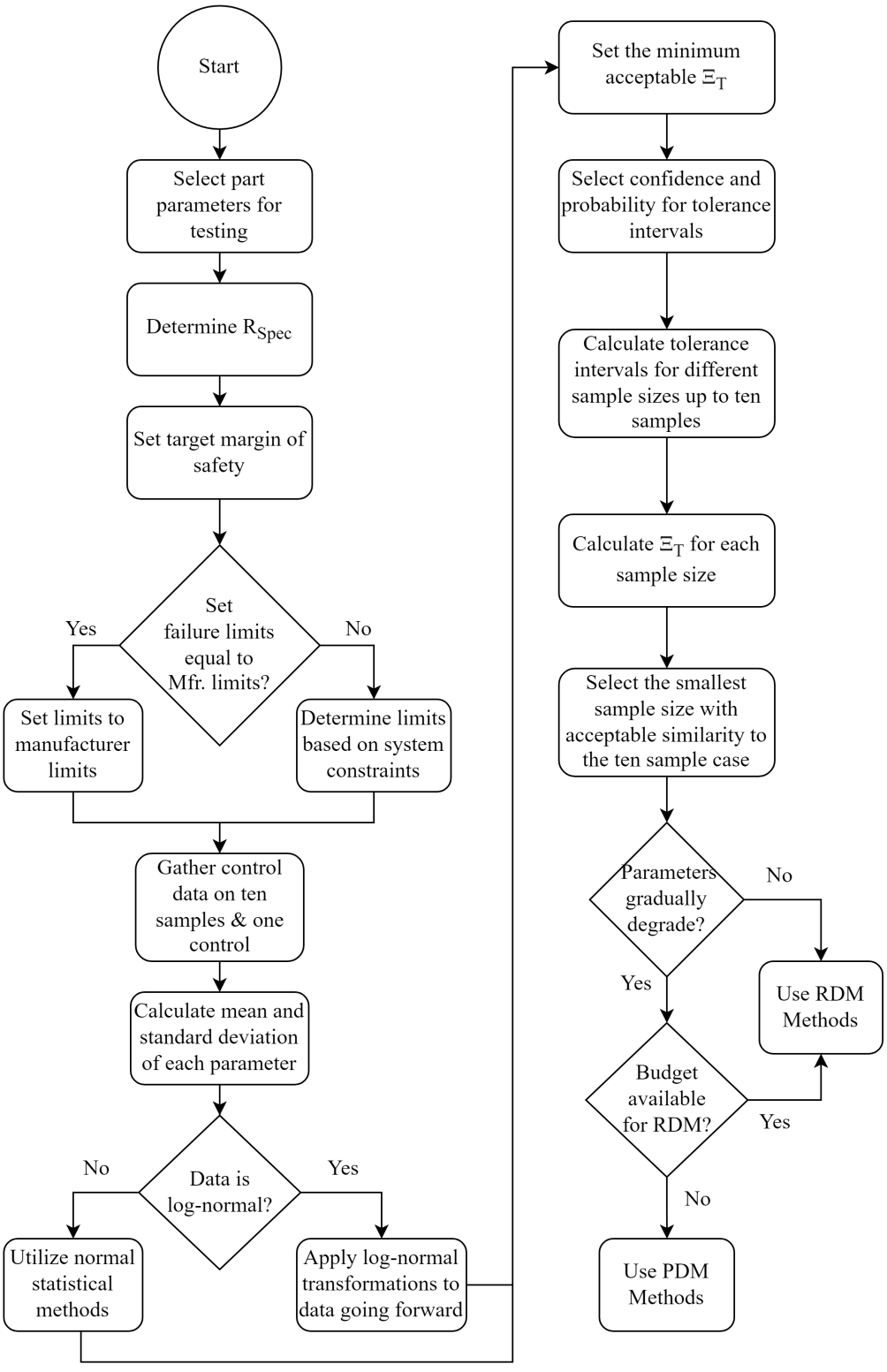

FIGURE 7.2: Screening Method and Sample Size Selection Decision Tree

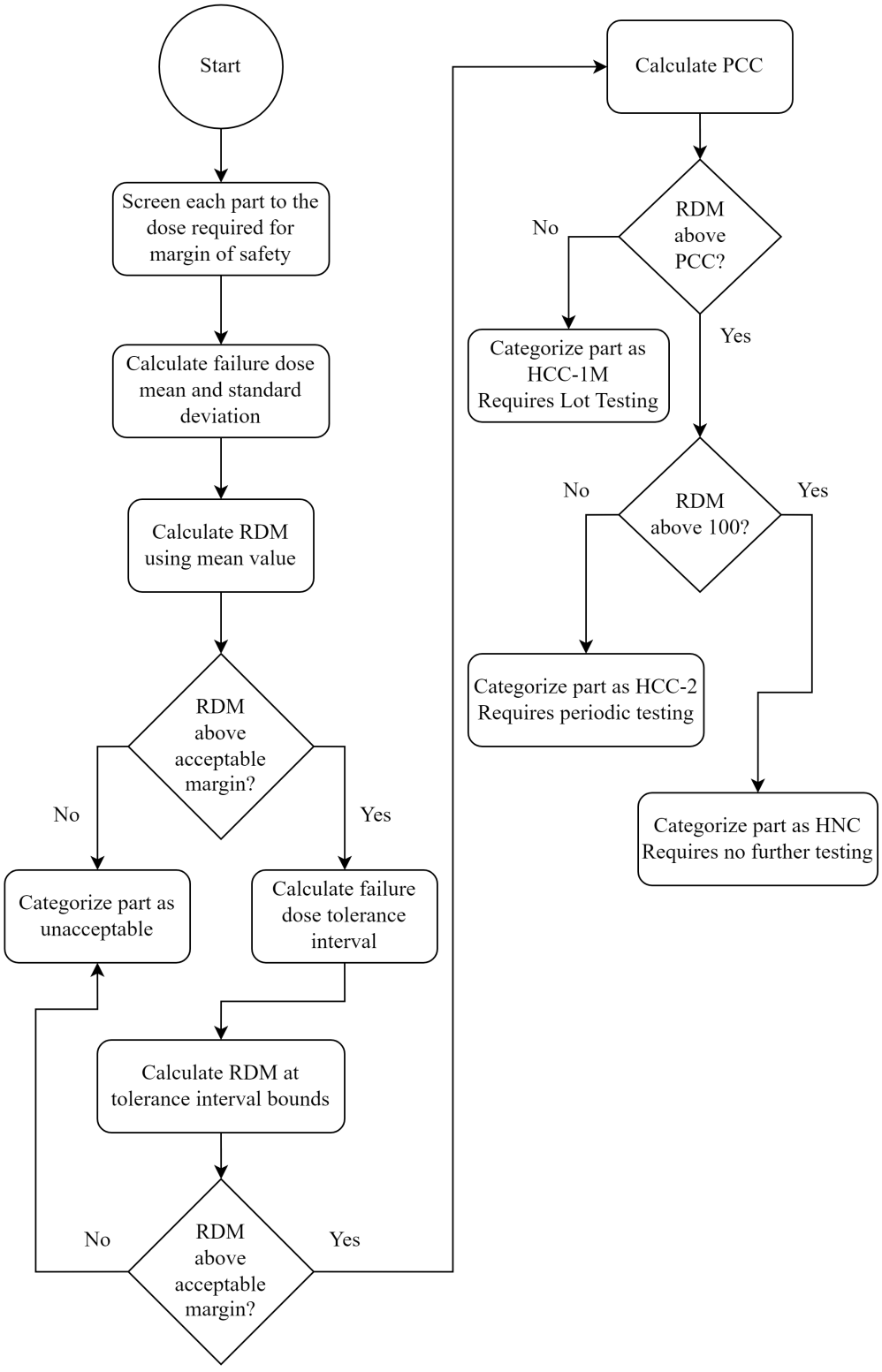

FIGURE 7.3: *RDM* Method Decision Tree

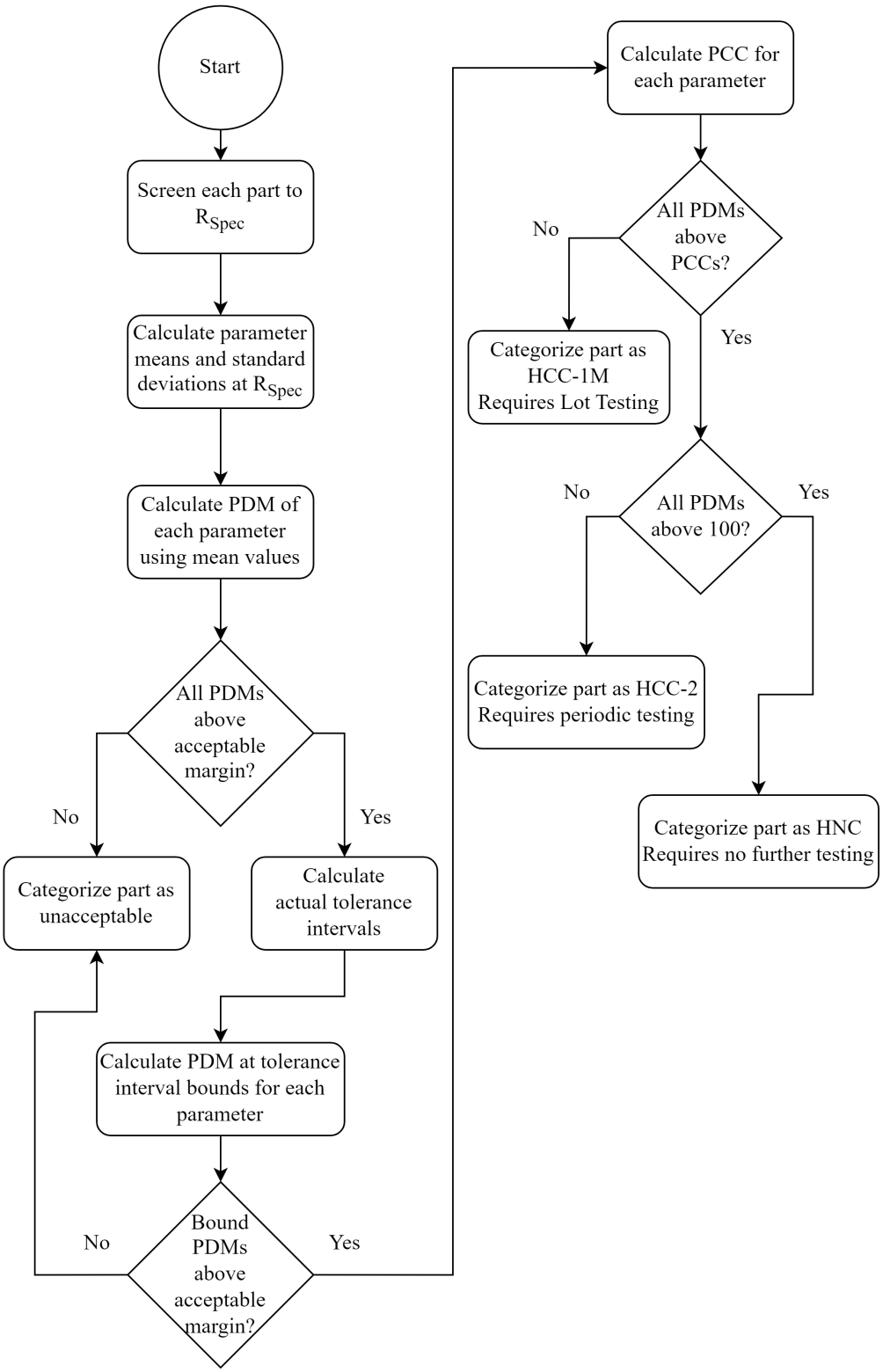

FIGURE 7.4: *PDM* Method Decision Tree

7.7 Chapter Summary

In this chapter, I discussed the results of my research as they related to my three hypotheses. I summarized the test results and discussed commercial parts becoming viable for use in LEO missions of fewer than five years. Next, I provided an overview of the test platform and how the test bench and analysis software achieved the three design goals (reusability, reconfigurability, and cost-effectiveness). Finally, I discussed my proposed additions to existing industry guidelines for radiation effects test planning. These additions included an updated *PDM* equation, tolerance interval statistics, and a more comprehensive small sample size selection algorithm.

8 Conclusion and Future Work

The overall objective of this research was to develop a statistical algorithm to aid in designing future radiation tolerance screening campaigns for commercial electronics. COTS electronics offer many potential benefits to the space industry, especially on missions operating in low Earth orbit, including better-operating speeds and lower costs. However, radiation hardness assurance testing is needed to ensure the components can be used in space. I proposed three hypotheses to address commercial electronic reliability in future space missions. The hypotheses were:

1. Modern COTS electronics can be effectively utilized in large-scale, commercial satellites operating in low Earth orbit for missions with a duration of less than five years, without compromising reliability.

2. One can develop an algorithm which provides cost effective radiation screening and mitigation strategies for COTS electronics.

3. It is cost-effective to screen COTS electronics for use in space using a reconfigurable, and reusable test platform when compared to purchasing radiation-hardened components.

By verifying these hypotheses, I made the following contributions to space electronics:

- A modern radiation tolerance characterization for a select set of COTS electronics for use in low Earth orbit.
- An algorithm that suggests appropriate radiation screening sample sizes for future test campaigns using tolerance interval statistics.
- A reconfigurable, and reusable testing platform developed alongside my research colleagues, which is capable of cost effectively and rapidly testing a variety of candidate components for use in low Earth orbit.

My research screened various COTS electronics at TRIUMF throughout two test campaigns. It used the resultant data to develop the statistical algorithm. My research colleagues and I developed a reconfigurable test platform capable of accommodating N-Channel MOSFETS, half-bridge drivers, crystal oscillators, P-Channel MOSFETS, instrumentation amplifiers, and digital memory. This test platform aided in conducting the screening campaigns. The study screened 21 different components.

The N-channel MOSFET gate threshold voltages degraded due to total ionizing dose, resulting in the components experiencing parametric failure. The BUK7M and 300NB06 had good TID resistance to be considered for use on spacecraft operating in low Earth orbit with mission durations of fewer than five years. The DMNH6008 was determined to be unacceptable for use in space. No proton-induced SEEs were observed in all three components. Determining the LET threshold to induce upsets requires heavy ion testing.

The half-bridge drivers and instrumentation amplifiers did not experience proton-induced upsets. They were generally resistant to total dose degradation. The TPS2822 gate driver was the only part from this set that experienced failure and is unacceptable for use in space. The other components did not degrade as the total dose increased and may be suitable for use on spacecraft operating in low Earth orbit.

Crystal oscillators showed no degradation in response to total ionizing dose screening. All the components remained within specification limits for frequency and current draw. The digital memory similarly had no observed proton-induced bit flips in any of the tested parts. The screening campaigns successfully show that commercial parts are becoming a viable option for use in low Earth orbit.

The screening campaign successfully tested a reconfigurable test platform capable of screening any component within these categories: N-Channel MOSFETS, half-bridge drivers, crystal oscillators, P-Channel MOSFETS, instrumentation amplifiers, and digital memory. The test bench featured four slots to accommodate disposable device under-testing (DUT) boards which contain the actual parts to be irradiated. Multiple slots on the board allowed for a more rapid testing campaign. The slots allowed the test bench to conduct four test runs without the beam needing to enter cooldown. A two-axis robotic arm was required to remotely center each slot in-line with the beam between test runs. The slots also allowed the user not to spend time unplugging any power supplies or computers between tests, further reducing the turnover time.

The test bench software facilitated live analysis of the candidate parts during the screening campaign. The software was capable of being reconfigured for different components from different manufacturers through the use of user-provided initialization files.

Issues with the first test bench revision resulted in only one available slot during the initial screening campaign. This issue, however, resulted in screening time data being available to quantitatively compare a single board test setup with the multi-slot capability of the final test bench design used during the second campaign. The second test bench design using multiple slots reduced the average screening time by 36.5% over the single slot setup.

Overall, the screening campaign successfully screened a variety of COTS electronics for use on a future low Earth orbit space mission. The campaigns proved that a reconfigurable test platform could facilitate reduced screening time requirements in future radiation effects screening campaigns. My proposed test plan algorithm provides the following additions to current industry guidelines:

- Tolerance interval statistics to aid in hardness assurance categorization.
- An updated parametric design margin equation.
- A more comprehensive small sample size selection procedure for modern electronics.

Future work could involve screening more commercial parts to continue building a better understanding of how they fail in response to radiation. Additional work can also include expanding the test platform to cover other part categories not considered in this research. Other part categories include bipolar junction transistors, optocouplers, digital-analog converters, FPGAs, and comparators. Future test bench designs could also expand the number of slots to allow more test runs to be completed before requiring beam cooldown. The lack of proton-induced SEEs indicates that heavy ion testing should be considered for future campaigns if one wants to determine the LET threshold of the tested components.

Bibliography

[1] V. Reddy, "The spacex effect," *New Space*, vol. 6, no. 2, pp. 125–134, 2018.

[2] H. W. Jones, "The recent large reduction in space launch cost," presented at the 48th International Conference on Environmental Systems, Albuquerque, New Mexico, USA, 2018.

[3] Canadian Space Agency. "Introduction to radarsat." (1999), [Online]. Available: http://ewh.ieee.org/reg/7/millennium/radarsat/radarsat.pdf (visited on 02/12/2021).

[4] P. de Selding. "Canadian radarsat constellation to get $374 million cash infusion." (2010), [Online]. Available: https://spacenews.com/canadian-radarsat-constellation-get-374-million-cash-infusion/ (visited on 02/17/2021).

[5] J. Plante and M. Sampson. "Cost impacts of upgrading electronic parts for use in nasa spaceflight systems." (2003), [Online]. Available: https://spacenews.com/canadian-radarsat-constellation-get-374-million-cash-infusion (visited on 01/08/2021).

[6] V. Pisacane, "The space environment and its effects on space systems," American Institute of Aeronautics and Astronautics, 2012.

[7] R. H. Maurer, M. E. Fraeman, M. N. Martin, and D. R. Roth, "Harsh environments: Space radiation," *Johns Hopkins APL technical digest*, vol. 28, no. 1, p. 17, 2008.

[8] S. Buchner, D. McMorrow, J. Melinger, and A. Camdbell, "Laboratory tests for single-event effects," *IEEE Transactions on Nuclear Science*, vol. 43, no. 2, pp. 678–686, 1996.

[9] C. R. Julien, B. J. LaMeres, and R. J. Weber, "An fpga-based radiation tolerant smallsat computer system," in *2017 IEEE Aerospace Conference*, IEEE, 2017, pp. 1–13.

[10] J. Li, P. Reviriego, L. Xiao, and H. Wu, "Protecting memories against soft errors: The case for customizable error correction codes," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 2, pp. 651–663, 2021.

[11] A. Privat, H. J. Barnaby, P. C. Adell, *et al.*, "Multiscale modeling of total ionizing dose effects in commercial-off-the-shelf parts in bipolar technologies," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 190–198, 2018.

[12] F. Ravotti, "Dosimetry techniques and radiation test facilities for total ionizing dose testing," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1440–1464, 2018.

[13] J. L. Barth, K. A. LaBel, and C. Poivey, "Radiation assurance for the space environment," in *2004 International Conference on Integrated Circuit Design and Technology (IEEE Cat. No. 04EX866)*, IEEE, 2004, pp. 323–333.

[14] D. Sinclair and J. Dyer, "Radiation effects and cots parts in smallsats," 2013.

[15] D. Alexander, K. Hunt, M. Owens, and J. Lyke, "Affordable rad-hard–an impossible dream?," 2008.

[16] M. J. Boyle. "How did mass production affect the price of consumer goods?" (2021), [Online]. Available: https://www.investopedia.com/ask/answers/050615/how-did-mass-production-affect-price-consumer-goods.asp (visited on 08/27/2021).

[17] N. Cumins. "Intel processors over the years." (2022), [Online]. Available: https://www.businessnewsdaily.com/10817-slideshow-intel-processors-over-the-years.html (visited on 06/29/2022).

[18] Military&Aerospace Electronics. "Radiation-hardened electronics designers face increasing difficulty with shrinking chip geometries." (2011), [Online]. Available: https://www.militaryaerospace.com/computers/article/16717354/radiationhardened-electronics-designers-face-increasing-difficulty-with-shrinking-chip-geometries (visited on 01/18/2021).

[19] A. Christou and W. Webb, *Reliability and Quality in Microelectronic Manufacturing*. Center for Risk and Reliability at University of Maryland, 2006, ISBN: 9781933904153. [Online]. Available: https://books.google.ca/books?id=BzolsLQsAWEC.

[20] Storage Networking Industry Association. "Common raid disk data format." (2009), [Online]. Available: https://www.snia.org/tech_activities/standards/curr_standards/ddf (visited on 03/05/2021).

[21] T. Villela, C. A. Costa, A. M. Brandão, F. T. Bueno, and R. Leonardi, "Towards the thousandth cubesat: A statistical overview," *International Journal of Aerospace Engineering*, vol. 2019, 2019.

[22] A. Yahyaabadi, M. Driedger, N. Turenne, *et al.*, "Manitobasat-1: A novel approach for technology advancement," *IEEE Potentials*, vol. 39, no. 4, pp. 17–23, 2020.

[23] E. B. Rice and S. J. Lev-Tov, "Optimized spacecraft fault protection for the wise mission," in *2008 IEEE Aerospace Conference*, 2008, pp. 1–8.

[24] Wired. "Hubble Space Telescope: 1990-2007." (2004), [Online]. Available: https://www.wired.com/2004/02/hubble-space-telescope-1990-2007/ (visited on 11/28/2022).

[25] M. Clark, "To 2030 and beyond," *Company Director*, vol. 36, no. 4, pp. 56–57, 2020.

[26] M. Mitry, "Routers in space: Kepler communications' cubesats will create an internet for other satellites," *IEEE Spectrum*, vol. 57, no. 2, pp. 38–43, 2020.

[27] Canadian Space Agency. "Canadian CubeSat Project teams." (2022), [Online]. Available: https://www.asc-csa.gc.ca/eng/satellites/cubesat/selected-teams-map.asp (visited on 11/21/2022).

[28] K. Bolshakov, "Digital sun sensor design for nanosatellite applications," M. Sci. thesis, York University, Toronto, Canada, 2020.

[29] E. Kahr, K. O'Keefe, and S. Skone, "Optimizing tracking and acquisition capabilities for the canx-2 nanosatellite's cots gps receiver in orbit," in *Proceedings of the 23rd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2010)*, 2010, pp. 2750–2760.

[30] European Space Agency. "eoPortal - CanX-2 (Canadian Advanced Nanosatellite eXperiment-2)." (2017), [Online]. Available: https://www.eoportal.org/satellite-missions/canx-2#spacecraft (visited on 11/21/2022).

[31] C. Daehnick, I. Klinghoffer, B. Maritz, and B. Wiseman, "Large leo satellite constellations: Will it be different this time?" *McKinsey & Company, https://www.mckinsey.com*, vol. 4, 2020.

[32] O. B. Osoro and E. J. Oughton, "A techno-economic framework for satellite networks applied to low earth orbit constellations: Assessing starlink, oneweb and kuiper," *IEEE Access*, vol. 9, pp. 141 611–141 625, 2021.

[33] J. M. Pritts, R. Gaza, C. R. Bailey, and K. V. Nguyen, "Compendium of current heavy ion single-event effects test results for candidate electronics for nasa johnson space center," in *2022 IEEE Radiation Effects Data Workshop (REDW)(in conjunction with 2022 NSREC)*, IEEE, 2022, pp. 1–5.

[34] A. J. Kenna, B. G. Rax, D. O. Thorbourn, R. D. Harris, and S. S. McClure, "Compendium of recent total ionizing dose test results conducted by the jet propulsion laboratory from 2003 through 2009," in *2009 IEEE Radiation Effects Data Workshop*, 2009, pp. 32–38.

[35] J. Ward, J. McKoy, I. Jeffrey, D. Ross, and P. Ferguson, "Radiation assessment of two automotive-grade n-channel mosfets," in *2022 IEEE Radiation Effects Data Workshop (REDW)(in conjunction with 2022 NSREC)*, IEEE, 2022, pp. 1–4.

[36] S. Stoyanov, C. Bailey, and G. Tourloukis, "Similarity approach for reducing qualification tests of electronic components," *Microelectronics Reliability*, vol. 67, pp. 111–119, 2016.

[37] S. Saxena, D. Roman, V. Robu, D. Flynn, and M. Pecht, "Battery stress factor ranking for accelerated degradation test planning using machine learning," *Energies*, vol. 14, no. 3, p. 723, 2021.

[38] A. L. Gould, "Planning and revising the sample size for a trial," *Statistics in medicine*, vol. 14, no. 9, pp. 1039–1051, 1995.

[39] A. Banerjee, U. Chitnis, S. Jadhav, J. Bhawalkar, and S. Chaudhury, "Hypothesis testing, type i and type ii errors," *Industrial psychiatry journal*, vol. 18, no. 2, p. 127, 2009.

[40] C. Beleites, U. Neugebauer, T. Bocklitz, C. Krafft, and J. Popp, "Sample size planning for classification models," *Analytica chimica acta*, vol. 760, pp. 25–33, 2013.

[41] V. A. Chepov, S. B. Shmakov, I. I. Shvetsov-Shiovsky, A. G. Petrov, and V. D. Kalashnikov, "Solid-state drives parameters control system for ionizing radiation tests," in *2021 International Siberian Conference on Control and Communications (SIBCON)*, 2021, pp. 1–4.

[42] S. Baba, A. Gieraltowski, M. Jasinski, F. Blaabjerg, A. S. Bahman, and M. Zelechowski, "Active power cycling test bench for sic power mosfets—principles, design, and implementation," *IEEE Transactions on Power Electronics*, vol. 36, no. 3, pp. 2661–2675, 2021.

[43] P. Leitao, S. Feger, D. Porret, *et al.*, "Test bench development for the radiation hard gbtx asic," *Journal of Instrumentation*, vol. 10, no. 01, p. C01038, 2015.

[44] A. Barari, R. Dion, I. Jeffrey, and P. Ferguson, "Testing satellite control systems with drones," *IEEE Potentials*, vol. 41, no. 1, pp. 6–13, 2021.

[45] European Space Agency. "Low earth orbit." (2020), [Online]. Available: https://www.esa.int/ESA_Multimedia/Images/2020/03/Low_Earth_orbit (visited on 08/01/2021).

[46] V. Gupta, "Analysis of single event radiation effects and fault mechanisms in sram, fram and nand flash: Application to the mtcube nanosatellite project," Ph.D. dissertation, Université Montpellier, 2017.

[47] S. M. Seltzer, "Updated calculations for routine space-shielding radiation dose estimates: Shieldose-2," *NIST Publication NISTIR 5477*, 1994.

[48] B. R. Bhat, N. Upadhyaya, and R. Kulkarni, "Total radiation dose at geostationary orbit," *IEEE transactions on nuclear science*, vol. 52, no. 2, pp. 530–534, 2005.

[49] K. A. LaBel, "Proton single event effects (see) guideline," *Submitted for publication on the NASA Electronic Parts and Packaging (NEPP) Program web site*, 2009.

[50] M. M. Rahman, D. Shankar, and S. Santra, "Analysis of radiation environment and its effect on spacecraft in different orbits," in *Conference: international Astronautical Congress (IAC2017)*, 2017, pp. 8073–8079.

[51] M. Looper, J. Mazur, J. Blake, *et al.*, "Long-term observations of galactic cosmic ray let spectra in lunar orbit by lro/crater," *Space Weather*, vol. 18, no. 12, 2020.

[52] European Space Components Coordination, *Single event effects test methods and guidelines*, ESCC Std. 25100 No. 2, 2014.

[53] E. Petersen, "The seu figure of merit and proton upset rate calculations," *IEEE Transactions on Nuclear Science*, vol. 45, no. 6, pp. 2550–2562, 1998.

[54] J. Barak, R. Reed, and K. LaBel, "On the figure of merit model for seu rate calculations," *IEEE Transactions on Nuclear Science*, vol. 46, no. 6, pp. 1504–1510, 1999.

[55] D. M. Hiemstra and E. W. Blackmore, "Let spectra of proton energy levels from 50 to 500 mev and their effectiveness for single event effects characterization of microelectronics," *IEEE Transactions on Nuclear Science*, vol. 50, no. 6, pp. 2245–2250, 2003.

[56] S. Brylow and T. Soulanille, "Low-cost quick-look radiation testing of electronic components for the mars observer camera," 1992.

[57] European Space Components Coordination, *Total dose steady-state irradiation test methods and guidelines*, ESCC Std. 22900 No. 5, 2016.

[58] S. Buchner, P. Marshall, S. Kniffin, and K. LaBel, "Proton test guideline development–lessons learned," *NASA/Goddard Space Flight Center, NEPP*, 2002.

[59] TRIUMF. "About triumf." (2020), [Online]. Available: https://www.triumf.ca/home/abouttriumf/about-us/overview-profile (visited on 12/04/2020).

[60] TRIUMF. "Beam availability and cost." (2020), [Online]. Available: https://www.triumf.ca/beam-availability (visited on 12/04/2020).

[61] Texas A&M University Cyclotron Institute. "Heavy ion testing." (2020), [Online]. Available: https://cyclotron.tamu.edu/ref/heavy_ions.html (visited on 12/16/2020).

[62] McMaster University. "Co-60 irradiations." (2020), [Online]. Available: https://nuclear.mcmaster.ca/products-services/co-60-irradiations/ (visited on 12/08/2020).

[63] Sandia National Laboratories. "Gamma irradiation facility and low-dose-rate irradiation facility." (2023), [Online]. Available: https://www.sandia.gov/research/gamma-irradiation-facility-and-low-dose-rate-irradiation-facility/ (visited on 02/06/2023).

[64] J.-M. Lauenstein, M. C. Casey, E. P. Wilcox, *et al.*, "Recent radiation test results for trench power mosfets," in *2017 IEEE Radiation Effects Data Workshop (REDW)*, IEEE, 2017, pp. 1–8.

[65] L. Scheick, *Testing guideline for segr of power mosfets*, Pasadena, Florida, 2008.

[66] R. Koga, S. Bielat, and J. George, "Charged particle induced degradation of trench type n-channel power mosfets," in *2014 IEEE Radiation Effects Data Workshop (REDW)*, 2014, pp. 1–7. DOI: 10.1109/REDW.2014.7004556.

[67] J. H. Warner, E. Faraci, C. Pham, A. Khachatrian, and D. McMorrow, "Set characterization of a high and low side gate driver (ric7s113) using pulsed laser and heavy ion testing," in *2022 IEEE Radiation Effects Data Workshop (REDW) (in conjunction with 2022 NSREC)*, 2022, pp. 1–9. DOI: 10.1109/REDW56037.2022.9921552.

[68] Wikimedia Foundation, *Instrumentation amplifier*, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Instrumentation_amplifier (visited on 01/16/2023).

[69] J. Agapito, J. Santos, F. Franco, *et al.*, "Radiation tests on commercial instrumentation amplifiers, analog switches & dac's," 2001.

[70] C. Renaudie, M. Markgraf, O. Montenbruck, and M. Garcia, "Radiation testing of commercial-off-the-shelf gps technology for use on low earth orbit satellites," in *2007 9th European Conference on Radiation and Its Effects on Components and Systems*, IEEE, 2007, pp. 1–8.

[71] T. R. Oldham, M. Friendlich, M. Carts, C. Seidleck, and K. A. LaBel, "Effect of radiation exposure on the endurance of commercial nand flash memory," *IEEE Transactions on Nuclear Science*, vol. 56, no. 6, pp. 3280–3284, 2009.

[72] R. Harboe-Sorensen, R. Muller, and S. Fraenkel, "Heavy ion, proton and co-60 radiation evaluation of 16 mbit dram memories for space application," in *Proceedings of 1995 IEEE Nuclear and Space Radiation Effects Conference (NSREC'95)*, IEEE, 1995, pp. 42–49.

[73] Department of Defense, *Department of defense test method for standard microcircuits*, MIL-STD-883L, 2019.

[74] Department of Defense, *Ionizing dose and neutron hardness assurance guidelines for microcircuits and semiconductor devices*, Military Handbook MIL-HDBK-814, 1994.

[75] I. Arimura and A. Namenson, "Hardness assurance statistical methodology for semiconductor devices," *IEEE Transactions on Nuclear Science*, vol. 30, no. 6, pp. 4322–4325, 1983.

[76] Kent State University. "Spss tutorials: Paired samples t test." (2023), [Online]. Available: https://libguides.library.kent.edu/spss/pairedsamplesttest (visited on 03/08/2023).

[77] San Jose State University. "T-score table." (2023), [Online]. Available: https://www.sjsu.edu/faculty/gerstman/StatPrimer/t-table.pdf (visited on 03/25/2023).

[78] S. C. Davis, A. D. Yarbrough, R. Koga, A. W. Wright, J. L. Taggart, and B. N. Davis, "The aerospace corporation's compendium of recent radiation testing results," in *2021 IEEE Radiation Effects Data Workshop (REDW)*, 2021, pp. 1–8.

[79] D. J. Cochran, S. P. Buchner, C. Poivey, *et al.*, "Compendium of current total ionizing dose results and displacement damage results for candidate spacecraft electronics for nasa," in *2007 IEEE Radiation Effects Data Workshop*, 2007, pp. 146–152.

[80] A. D. Topper, E. P. Wilcox, M. C. Casey, *et al.*, "Nasa goddard space flight center's compendium of recent total ionizing dose and displacement damage dose results," in *2018 IEEE Radiation Effects Data Workshop (REDW)*, 2018, pp. 1–7.

[81] L. N. Kessarinskiy, D. V. Boychenko, A. G. Petrov, *et al.*, "Compendium of tid comparative results under x-ray, gamma and linac irradiation," in *2014 IEEE Radiation Effects Data Workshop (REDW)*, 2014, pp. 1–3.

[82] Automotive Electronics Council, *Failure mechanism based stress test qualification for integrated circuits*, AEC - Q100 - Rev-H, 2014.

[83] P. O'Neill, G. Badhwar, and W. Culpepper, "Risk assessment for heavy ions of parts tested with protons," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 2311–2314, 1997.

[84] Nexperia. "Buk7m specifications." (2019), [Online]. Available: https://assets.nexperia.com.

[85] Taiwan Semiconductor. "300nb06 specifications." (2022), [Online]. Available: https://www.taiwansemi.com.

[86] Diodes Incorporated. "Dmnh6008 specifications." (2018), [Online]. Available: https://www.diodes.com.

[87] Rohm Semiconductor. "Bss84a specifications." (2020), [Online]. Available: https://fscdn.rohm.com.

[88] Toshiba. "Ssm6j80x specifications." (2019), [Online]. Available: https://www.mouser.ca.

[89] Microchip. "Tc442 specifications." (2012), [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21421E.pdf.

[90] Texas Instruments. "Tps2822 specifications." (2021), [Online]. Available: https://www.ti.com.

[91] Diodes Incorporated. "Dgd054x specifications." (2021), [Online]. Available: https://www.diodes.com.

[92] Renesas. "Isl784x specifications." (2018), [Online]. Available: https://www.renesas.com/us/en/document/dst/isl78424-isl78434-isl78444-datasheet.

[93] Texas Instruments. "Ina32x specifications." (2006), [Online]. Available: https://www.ti.com/lit/ds/sbos168d/sbos168d.pdf.

[94] Analog Devices. "Ad524 specifications." (2018), [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ad524.pdf.

[95] Texas Instruments. "Lm741 specifications." (2015), [Online]. Available: https://www.ti.com/lit/ds/symlink/lm741.pdf.

[96] STMicroelectronics. "Tsv911 specifications." (2022), [Online]. Available: https://www.st.com/en/amplifiers-and-comparators/tsv911.html.

[97] TXC. "7x-20 specifications." (2022), [Online]. Available: http://www.txccrystal.com/images/pdf/7x.pdf.

[98] TXC. "8w-13 specifications." (2022), [Online]. Available: http://www.txccrystal.com/images/pdf/8w-tight.pdf.

[99] TXC. "Au-12 specifications." (2022), [Online]. Available: http://www.txccrystal.com/images/pdf/au.pdf.

[100] TXC. "Aw-11 specifications." (2022), [Online]. Available: http://www.txccrystal.com/images/pdf/aw.pdf.

[101] S. C. Davis, D. J. Mabry, A. D. Yarbrough, *et al.*, "The aerospace corporation's compendium of recent radiation effect results," in *2020 IEEE Radiation Effects Data Workshop (in conjunction with 2020 NSREC)*, 2020, pp. 1–7.

[102] International Rectifier. "Ir's expanded family of radiation hardened logic level mosfets simplify design and increase reliability." (2008), [Online]. Available: https://www.irf.com/pressroom/pressreleases/nr080603.html (visited on 03/25/2023).

[103] Texas Instruments. "Gate drivers product selection - space." (2008), [Online]. Available: https://www.ti.com/power-management/gate-drivers/products.html#1498=Space& (visited on 03/25/2023).

[104] STMicroelectronics. "Rad-hard, crystal oscillator driver and divider." (2023), [Online]. Available: https://www.st.com/en/space-products/rhfosc04.html (visited on 03/25/2023).

[105] Texas Instruments. "Amplifiers product selection - space." (2023), [Online]. Available: https://www.ti.com/amplifier-circuit/products.html#1498=Space& (visited on 03/25/2023).

[106] Infineon. "Rad hard memories." (2023), [Online]. Available: https://www.infineon.com/cms/en/product/high-reliability/space/rad-hard-memories/ (visited on 03/25/2023).

[107] M. Mottonen, P. Belt, J. Harkonen, H. Haapasalo, and P. Kess, "Manufacturing process capability and specification limits," *The Open Industrial & Manufacturing Engineering Journal*, vol. 1, no. 1, 2008.

A First Round Test Bench Issues

This appendix provides an overview of the design issues present in the first test bench. These issues were ultimately addressed by the second test bench revision for the second screening campaign. However, the data obtained from the first round required analyzing and addressing the potential impact of each issue.

A.1 Half-Bridge Driver Thermal Issues

Initial startup testing of the test bench hardware while using one of the backup DUT boards resulted in the H-bridge drivers overheating. The H-bridge drivers ultimately burnt out When they were set to their high-side on-state for several seconds as initially planned. This unexpected behaviour required adjusting the state-change timing to mitigate this issue while still allowing me to test the H-bridge components. The components were instead screened for 14 seconds off, followed by one second on, enough time to see the H-bridge driver switch between low-side and high-side states, but with sufficient time to cool off before the next switch occurred.

A.2 MOSFET Power Supply Thermal Issues

Two independent MOSFET circuits were present in the test bench design, one of which experienced a consistent issue with overheating each time the test bench was powered on. This issue occurred thrice per DUT board, since the hardware was turned off twice

every test. The test hardware pauses (approximately one minute in duration each) occurred because the beam line would turn off every 10 kRad(Si) as a TRIUMF facility safety precaution and required me to call in for further irradiation each time to continue screening a sample.

The drain-source MOSFET circuit was driven by a power supply operating at 30 V. The MOSFET gate and current sense circuits were connected to another primary power supply working at 12 V alongside the rest of the test bench hardware. A series of voltage regulators lowered the supply voltage of the primary power supply to the various levels needed to operate the hardware. The overheating issue consistently occurred in the voltage regulators approximately five minutes after I turned on the test bench. Several symptoms of this issue presented themselves once the heating issue occurred and primarily affected the MOSFET circuitry. The thermal issue reduced the maximum voltage supply to the MOSFET gates by one volt due to a reduction in power to the gate potentiometers. This issue decreased the full range of the MOSFET gate test from 5 V to 4 V. This reduction did not affect the collected data because the MOSFET gate threshold voltages were still below the maximum voltage supplied to the gates. Figures A.1 and A.3 provide plots of some of the raw data for the gate voltage supply to the MOSFETs, providing a visualization of this symptom while showing that it was a repeatable and consistent issue.

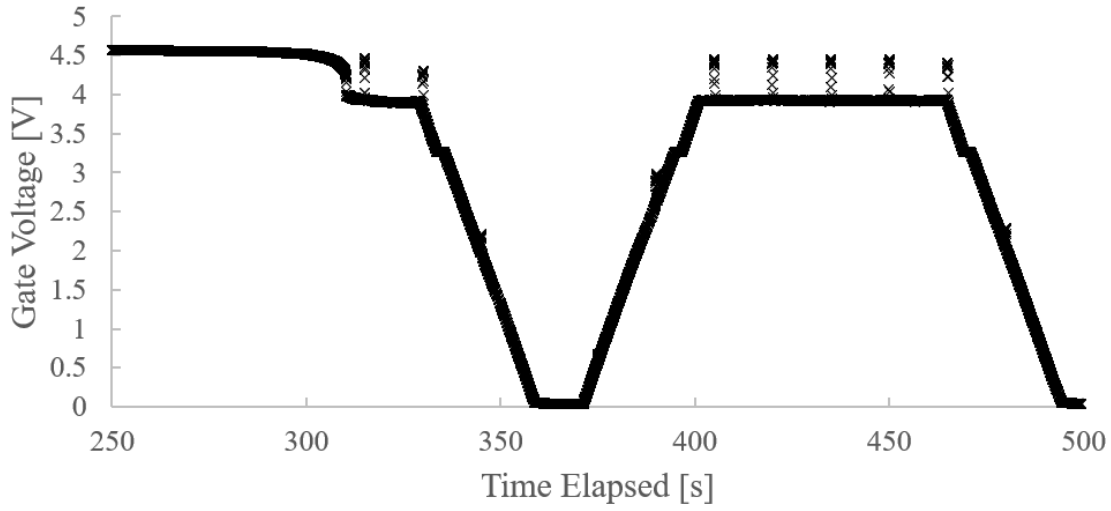

FIGURE A.1: DUT five's first gate voltage drop due to the voltage regulators overheating.

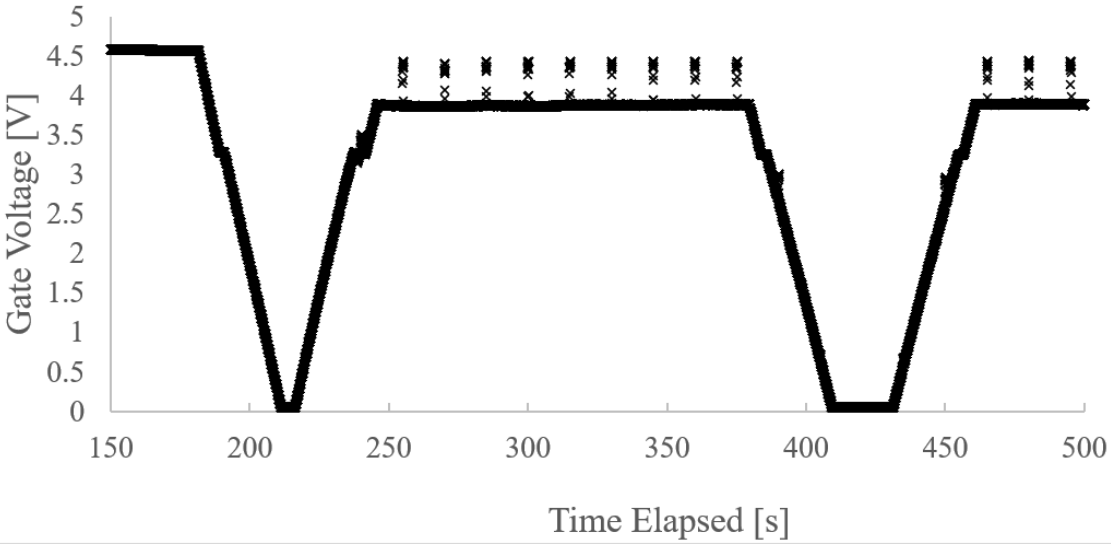

FIGURE A.2: DUT five's second gate voltage drop due to the voltage regulators overheating.

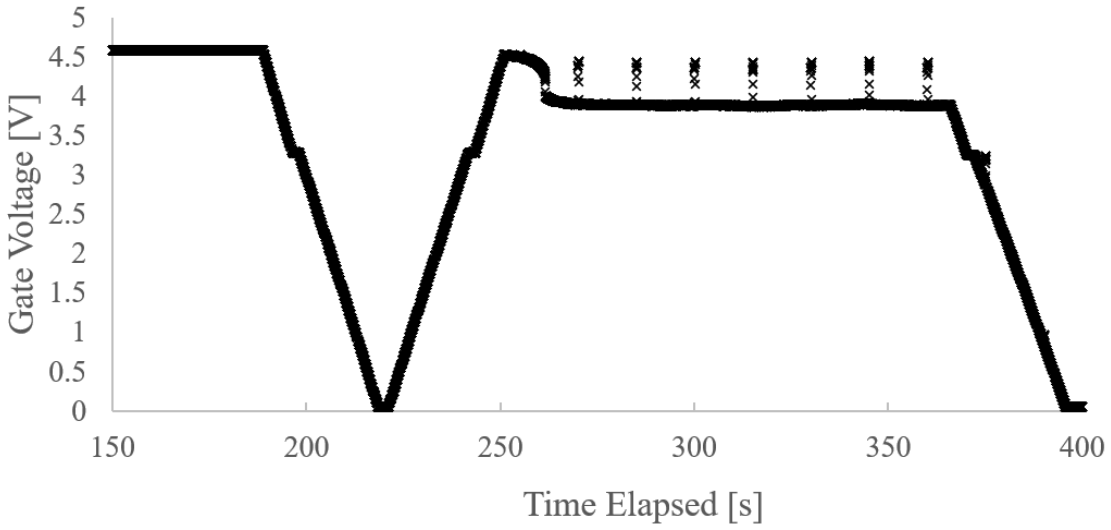

FIGURE A.3: DUT five's final gate voltage drop due to the voltage regulators overheating.

A.3 MOSFET Current Sense Leak

The next issue present in the design was a current leak between the MOSFET current sensors. Figure A.4 visualizes some of the gate stress test results I obtained for DUT seven from the perspective of the Q300NB06's current sense amplifier. Each gate test showed a two-step response to increased gate voltage, each step corresponding to one of the two MOSFETs on the DUT board. However, the BUK7M and Q300NB06 had different threshold voltages, which provided enough separation for their results to be analyzed, as indicated by the middle plateau shown in the figure. The Q300NB06 degraded faster than the BUK7M, further increasing the separation between the two MOSFETs as the tests progressed.

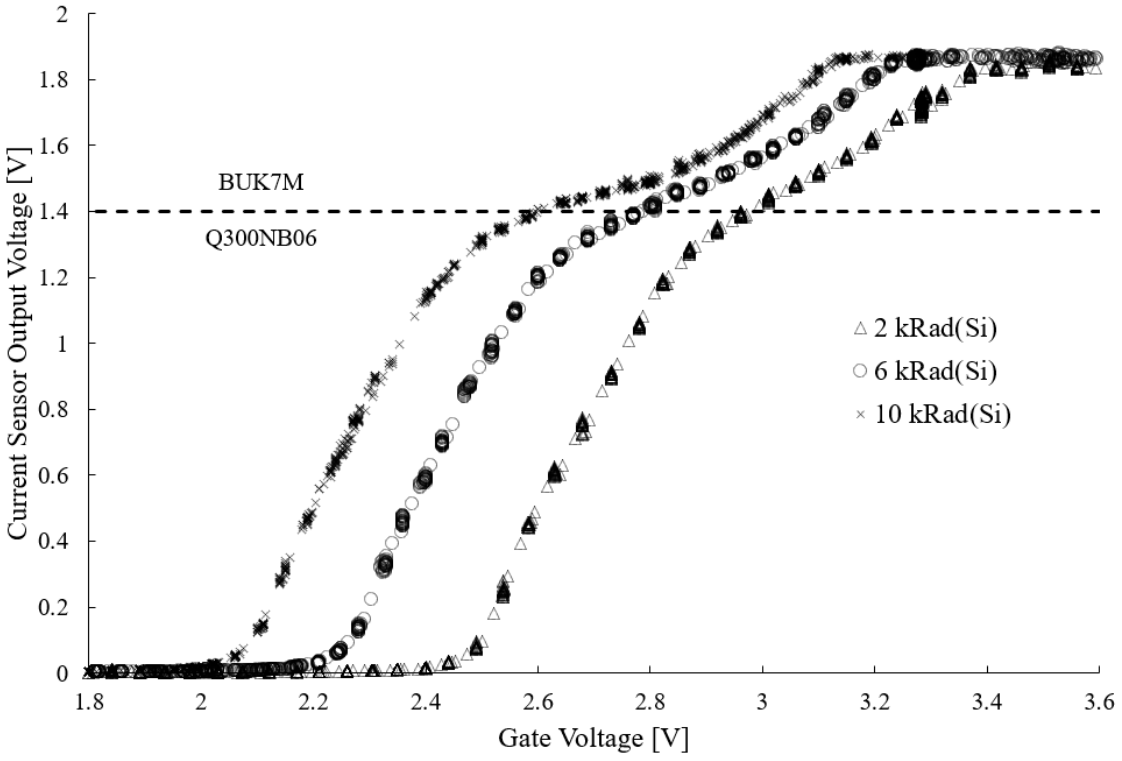

FIGURE A.4: Gate stress test data for DUT seven visualizing the MOSFET current leak.

This issue meant that while the current sense voltage output could determine if the MOSFETs were on and off, I no longer had a direct transformation factor for changing the voltage values into appropriate amperage results. Using a multimeter, I mitigated this issue by determining the amperage flowing through each MOSFET current sense circuit. Checking the amperage using the multimeter let me establish that the actual current flowing through MOSFETs during testing would switch from 0 mA when off to 60 mA when toggled on. I later mapped the high and low results of the current sense readings to this range during my analysis.

A.4 Half-Bridge Driver Interference

I encountered another issue with the MOSFET circuit after the power system began overheating in the form of H-bridge driver interference present on the gate voltages. The H-bridge drivers toggling on caused consistent voltage spikes to occur every 15 seconds as shown in figure A.5. However, this interference did not hinder my ability to determine the gate threshold at the various dose breakpoints. I excluded two seconds of data at each interference spike when calculating the MOSFET gate threshold values to ensure that the interference did not affect my calculations. Figure A.6 visualizes the data shown in figure A.5 after removing the interference.

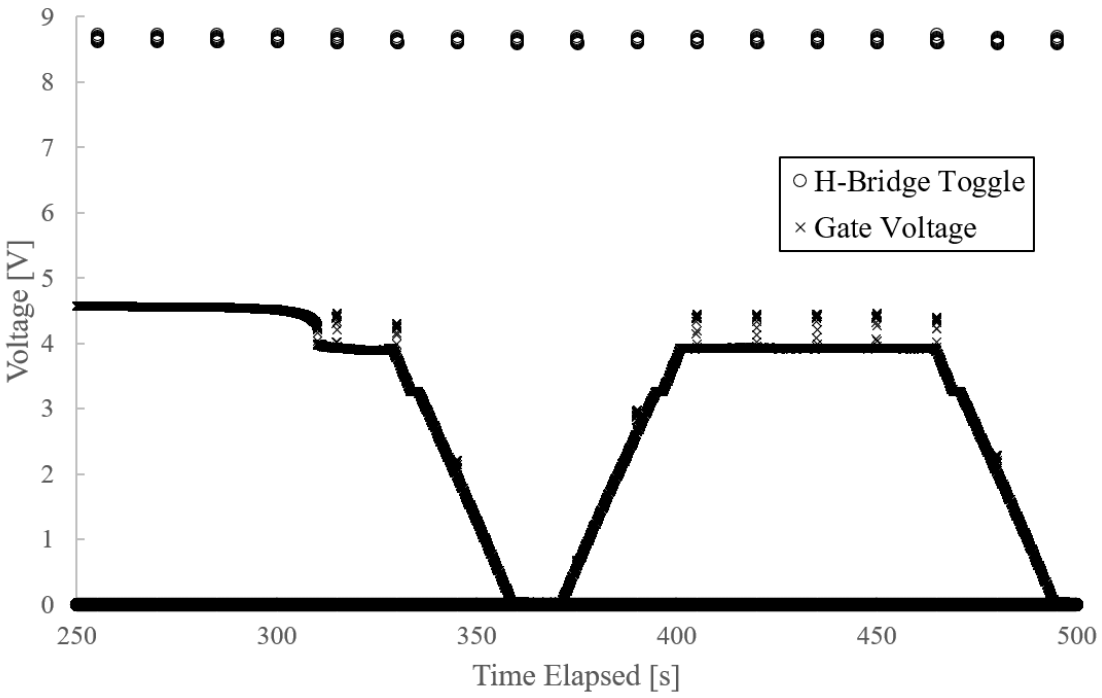

FIGURE A.5: MOSFET Gate voltage spikes in response to the H-bridge drivers being commanded to switch state. Data plotted from DUT five.

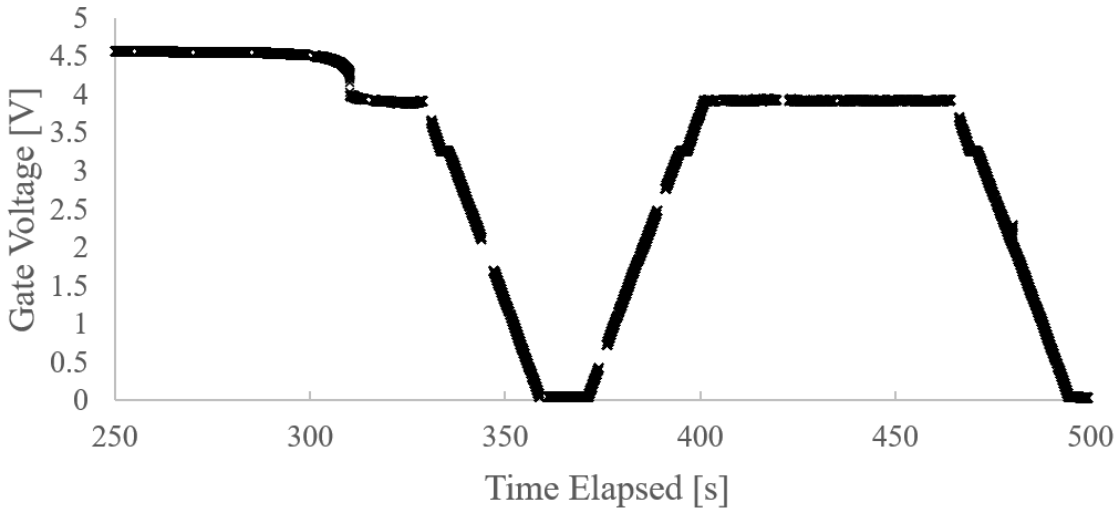

FIGURE A.6: MOSFET Gate voltage after removal of H-bridge interference. Data plotted from DUT five.

B Test Bench Software

B.1 Automated Analysis Program Source Code

```
"""
Test bench software developed by Jesse Ward for the purposes
of conducting radiation effect screening as part of a masters
thesis aimed towards obtaining a mechanical engineering degree
at the University of Manitoba
"""

from __future__ import absolute_import, division, print_function
from builtins import *
from tkinter import *
import tkinter as tk
from tkinter import IntVar
import numpy as np
from ctypes import c_double, cast, POINTER, addressof, sizeof
from time import sleep
import sys
import os
np.set_printoptions(suppress=True)
import multiprocessing
import threading
import time
import struct
import logging
import select
import serial
import matplotlib.pyplot as plt
from tkinter import filedialog as fd
import xml.etree.cElementTree as ET
from configparser import ConfigParser
from mcculw import ul
from mcculw.enums import DigitalIODirection, ScanOptions, FunctionType, Status
```

```
from mcculw.enums import InterfaceType, ErrorCode
from mcculw.ul import ULError
from mcculw.device_info import DaqDeviceInfo

try:
    from ui_examples_util import UIExample, show_ul_error
except ImportError:
    from .ui_examples_util import UIExample, show_ul_error

class TimerError(Exception):
    """A custom exception used to report errors in use of Timer class"""

class Timer:
    def __init__(self):
        self._start_time = None

    def start(self):
        """Start a new timer"""
        if self._start_time is not None:
            raise TimerError(f"Timer is running. Use .stop() to stop it")
        self._start_time = time.perf_counter()

    def stop(self):
        """Stop the timer, and report the elapsed time"""
        if self._start_time is None:
            raise TimerError(f"Timer is not running. Use .start() to start it")
        self.elapsed_time = time.perf_counter() - self._start_time
        self._start_time = None
        print(f"Elapsed time: {self.elapsed_time:0.4f} seconds")

    def poll(self):
        """Poll the timer, reports the elapsed time without stopping"""
        if self._start_time is None:
            raise TimerError(f"Timer is not running. use .start() to start it")
        self.polled_time = time.perf_counter() - self._start_time
        print(f"Elapsed time: {self.polled_time:0.4f} seconds")

class Application(UIExample):
    def create_widgets(self):
        """Large function used to initialize the GUI"""
```

```
        self.device_label = tk.Label(self)
        self.device_label.pack(fill=tk.NONE, anchor=tk.NW)
        self.device_label["text"] = ('Board Number ' + str(self.board_num)
                                     + ": " + self.device_info.product_name
                                     + " (" + self.device_info.unique_id + ")")
        main_frame = tk.Frame(self)
        main_frame.pack(fill=tk.X, anchor=tk.NW)
        curr_row = 0
        self.file_name_label = tk.Label(main_frame)
        self.file_name_label["text"] = ('File name: ')
        self.file_name_label.grid(row=curr_row, column=0, sticky=tk.W)
        self.file_name_in = tk.Entry(main_frame, width = 16)
        self.file_name_in.insert(0,"scan_data.csv")
        self.file_name_in.grid(row=curr_row, column=1, sticky=tk.W)
        self.dose_label = tk.Label(main_frame)
        self.dose_label["text"] = ('TID Dose: ')
        self.dose_label.grid(row=curr_row, column=2, sticky=tk.W)
        self.dose_in = tk.Entry(main_frame, width = 16)
        self.dose_in.insert(0,"0")
        self.dose_in.grid(row=curr_row, column=3, sticky=tk.W)
        self.dose_unit = tk.Label(main_frame)
        self.dose_unit["text"] = ('Rad')
        self.dose_unit.grid(row=curr_row, column=4, sticky=tk.W)
        self.fluence_label = tk.Label(main_frame)
        self.fluence_label["text"] = ('SEE Fluence: ')
        self.fluence_label.grid(row=curr_row+1, column=2, sticky=tk.W)
        self.fluence_in = tk.Entry(main_frame, width = 16)
        self.fluence_in.insert(0,"0")
        self.fluence_in.grid(row=curr_row+1, column=3, sticky=tk.W)
        self.fluence_unit = tk.Label(main_frame)
        self.fluence_unit["text"] = ('p/cm^2')
        self.fluence_unit.grid(row=curr_row+1, column=4, sticky=tk.W)
        self.beam_energy_label = tk.Label(main_frame)
        self.beam_energy_label["text"] = ('Beam Energy: ')
        self.beam_energy_label.grid(row=curr_row+2, column=2, sticky=tk.W)
        self.beam_energy_in = tk.Entry(main_frame,width = 16)
        self.beam_energy_in.insert(0,"0")
        self.beam_energy_in.grid(row=curr_row+2, column=3, sticky=tk.W)
        self.beam_unit = tk.Label(main_frame)
        self.beam_unit["text"] = ('MeV')
```

```
        self.beam_unit.grid(row=curr_row+2, column=4, sticky=tk.W)
        self.DUT_label = tk.Label(main_frame)
        self.DUT_label["text"] = ('DUT Board #: ')
        self.DUT_label.grid(row=curr_row+3, column=2, sticky=tk.W)
        self.DUT_num = tk.Entry(main_frame, width = 16)
        self.DUT_num.insert(0,"0")
        self.DUT_num.grid(row=curr_row+3, column=3, sticky=tk.W)
        curr_row +=1
        if self.ai_info.num_chans > 1:
            channel_vcmd = self.register(self.validate_channel_entry)
            low_channel_entry_label = tk.Label(main_frame)
            low_channel_entry_label["text"] = "Low Channel Number:"
            low_channel_entry_label.grid(
                row=curr_row, column=0, sticky=tk.W)
            self.low_channel_entry = tk.Spinbox(
                main_frame, from_=0,
                to=max(self.ai_info.num_chans - 1, 0),
                validate='key', validatecommand=(channel_vcmd, '%P'))
            self.low_channel_entry.grid(
                row=curr_row, column=1, sticky=tk.W)
            curr_row += 1
            high_channel_entry_label = tk.Label(main_frame)
            high_channel_entry_label["text"] = "High Channel Number:"
            high_channel_entry_label.grid(
                row=curr_row, column=0, sticky=tk.W)
            self.high_channel_entry = tk.Spinbox(
                main_frame, from_=0, validate='key',
                to=max(self.ai_info.num_chans - 1, 0),
                validatecommand=(channel_vcmd, '%P'))
            self.high_channel_entry.grid(
                row=curr_row, column=1, sticky=tk.W)
            initial_value = min(self.ai_info.num_chans - 1, 11)
            self.high_channel_entry.delete(0, tk.END)
            self.high_channel_entry.insert(0, str(initial_value))
            curr_row += 1

        vds_val_label = tk.Label(main_frame)
        vds_val_label["text"] = "VDS Voltage:"
        vds_val_label.grid(row=curr_row, column=0, sticky=tk.W)

```

```
        self.vds_val = tk.Spinbox(
            main_frame, from_=0, validate='key', to=30,
            values=(0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30))
        self.vds_val["command"] = self.vds_update
        self.vds_val.grid(row=curr_row, column=1, sticky=tk.W)
        curr_row += 1
        ctr_value_left_label = tk.Label(main_frame)
        ctr_value_left_label["text"] = "XTAL 1 Freq [Hz]:"
        ctr_value_left_label.grid(row=curr_row, column=0, sticky=tk.W)
        self.ctr_value_label = tk.Label(main_frame)
        self.ctr_value_label.grid(row=curr_row, column=1, sticky=tk.W)
        curr_row += 1
        ctr1_value_left_label = tk.Label(main_frame)
        ctr1_value_left_label["text"] = "XTAL 2 Freq [Hz]:"
        ctr1_value_left_label.grid(row=curr_row, column=0, sticky=tk.W)
        self.ctr1_value_label = tk.Label(main_frame)
        self.ctr1_value_label.grid(row=curr_row, column=1, sticky=tk.W)
        curr_row += 1
        bit_values_frame = tk.Frame(main_frame)
        bit_values_frame.grid(row=curr_row, column=0, padx=3, pady=3)
        label = tk.Label(bit_values_frame, text="Board Function:")
        label.grid(row=0, column=0, sticky=tk.W)
        label = tk.Label(bit_values_frame, text="State:")
        label.grid(row=1, column=0, sticky=tk.W)

        # Create Checkbutton controls for each bit
        self.bit_checkbutton_vars = []
        self.bit_checkbutton = [None,None,None,None,None,None,None,None]
        max_bit = min(self.port.num_bits, 8)
        D_labels = ["N/A","N/A","N/A","XTAL Invert EN","Gate Trigger","TID mode","Slot Chip Enable","Slot
            Power On"]
        for bit_num in range(0, max_bit):
            bit_label = tk.Label(bit_values_frame, text=str(D_labels[bit_num]))
            bit_label.grid(row=0, column=bit_num + 1)
            var = IntVar(value=0)
            self.bit_checkbutton[bit_num] = tk.Checkbutton(
                bit_values_frame, tristatevalue=-1, variable=var,
                borderwidth=0,
                command=lambda n=bit_num:
                self.bit_checkbutton_changed(n))
```

```
            self.bit_checkbutton[bit_num].grid(row=1, column=bit_num + 1, padx=(5, 0))
            self.bit_checkbutton_vars.append(var)
        curr_row += 1
        self.digital_group = tk.LabelFrame(
            self, text="Digital", padx=3, pady=3)
        self.digital_group.pack(fill=tk.X, anchor=tk.NW, padx=3, pady=3)
        self.digl = tk.Label(self.digital_group)
        self.digl["text"] = ('Command Text: ')
        self.digl.grid(row=0, column=0, sticky=tk.W)
        self.dig = tk.Entry(self.digital_group, width = 16)
        self.dig.grid(row=0, column=1, sticky=tk.W)
        self.dc = tk.Button(self.digital_group)
        self.dc["text"] = "SEND"
        self.dc["command"] = self.dsend
        self.dc.grid(row=0, column=2)
        self.results_group = tk.LabelFrame(
            self, text="Results", padx=3, pady=3)
        self.results_group.pack(fill=tk.X, anchor=tk.NW, padx=3, pady=3)
        self.data_frame = tk.Frame(self.results_group)
        self.data_frame.grid()
        button_frame = tk.Frame(self)
        button_frame.pack(fill=tk.X, side=tk.RIGHT, anchor=tk.SE)
        self.save_button = tk.Button(button_frame)
        self.save_button["text"] = "SAVE DATA"
        self.save_button["command"] = self.saverun
        self.save_button.grid(row=0, column=0, padx=3, pady=3)
        self.start_button = tk.Button(button_frame)
        self.start_button["text"] = "ANALOG START"
        self.start_button["command"] = self.start
        self.start_button.grid(row=0, column=1, padx=3, pady=3)
        self.dstart_button = tk.Button(button_frame)
        self.dstart_button["text"] = "DIGITAL START"
        self.dstart_button["command"] = self.dstart
        self.dstart_button.grid(row=0, column=2, padx=3, pady=3)
        self.quit_button = Button(button_frame)
        self.quit_button['text'] = 'QUIT'
        self.quit_button['fg'] = 'red'
        self.quit_button['command'] = self.quit
        self.quit_button.grid(row=0, column=3, padx=3, pady=3)
```

```
230     def __init__(self, master=None):
            """Initialization of variables and connection to the test bench"""
            super(Application, self).__init__(master)
            self._start_time = None
            self.use_device_detection = True
235         self.board_num = 0
            self.counter_num = 1
            self.running = False
            self.gate_counter = 0
            self.do_gate = 0
240         self.do_ctr = 0
            self.ctr_up = 0
            self.do_amp = 0
            self.do_hbd = 0
            self.mean_array = 0
245         self.mram_bytes_static = []
            self.mram_bytes_dynamic = []
            self.mram_bytes_dynamic_out = []
            self.mram_static_upsets = 0
            self.mram_dynamic_upsets = 0
250         self.mram_static_cross = 0
            self.mram_dynamic_cross= 0
            self.fram_bytes_static = []
            self.fram_bytes_dynamic = []
            self.fram_bytes_dynamic_out = []
255         self.fram_static_upsets = 0
            self.fram_dynamic_upsets = 0
            self.fram_static_cross = 0
            self.fram_dynamic_cross= 0
            self.sflash_bytes_static = []
260         self.sflash_bytes_dynamic = []
            self.sflash_bytes_dynamic_out = []
            self.sflash_static_upsets = 0
            self.sflash_dynamic_upsets = 0
            self.sflash_static_cross = 0
265         self.sflash_dynamic_cross= 0
            self.qflash_bytes_static = []
            self.qflash_bytes_dynamic = []
            self.qflash_bytes_dynamic_out = []
            self.qflash_static_upsets = 0
```

```
        self.qflash_dynamic_upsets = 0
        self.qflash_static_cross = 0
        self.qflash_dynamic_cross= 0

        global vds_ser
        vds_ser = serial.Serial('COM9', 9600, timeout=None, xonxoff=False, rtscts=False, dsrdtr=False)
        self.do_plot = False
        global function_bits
        function_bits = [0,0,0,0,0,0,0,0] #wip0, wip1, inamp_hbd_mode, xtal_inv, gate_Trigger, fet_mode,
            slot_en, slot_pwr

        self.nfet_gate_array = np.zeros((14,3,32)) #14 tests for 0:2:26 krad, [time,volt,current] and each
            voltage step 0:0.2:5
        self.nfet_gate_test_count = 0
        self.nfet_gate_th_array = np.zeros((14,1))
        self.pfet_gate_array = np.zeros((14,3,32))
        self.pfet_gate_test_count = 0
        self.pfet_gate_th_array = np.zeros((14,1))
        self.nfet_see_array = np.zeros((16,2)) # see counts per every 2 VDS 0:2:30, vdss, vgss
        self.pfet_see_array = np.zeros((16,2))
        self.hbd0_array = np.zeros((5,14)) #each 2 krad get low a low b high a high b pass
        self.hbd1_array = np.zeros((5,14))
        self.osc0_array = np.zeros((2,14)) #each krad get current, freq
        self.osc1_array = np.zeros((2,14))
        self.osc_test_count = 0
        self.inamp0_array = np.zeros((3,14)) #each 2 krad get low and high pass
        self.inamp1_array = np.zeros((3,14))
        self.amp_test_count = 0

        try:
            if self.use_device_detection:
                ul.ignore_instacal()
                devices = ul.get_daq_device_inventory(InterfaceType.ANY)
                if not devices:
                    raise ULError(ErrorCode.BADBOARD)

                # Add the first DAQ device to the UL with the specified board number
                ul.create_daq_device(self.board_num, devices[0])
            self.device_info = DaqDeviceInfo(self.board_num)
            dio_info = self.device_info.get_dio_info()
```

```
            self.ctr_info = self.device_info.get_ctr_info()
            self.port = next((port for port in dio_info.port_info
                              if port.supports_output), None)
            if self.port is not None:
                if self.port.is_port_configurable:
                    try:
                        ul.d_config_port(self.board_num, self.port.type,
                                         DigitalIODirection.OUT)
                    except ULError as e:
                        show_ul_error(e)
            if self.device_info.supports_analog_input and self.ctr_info.is_supported:
                self.ai_info = self.device_info.get_ai_info()
                ul.flash_led(self.board_num)
                ul.a_input_mode(self.board_num, 1)
                self.create_widgets()
            else:
                self.create_unsupported_widgets()
        except ULError:
            self.create_unsupported_widgets(True)
        print("read ini")
        config = ConfigParser()
        self.select_ini()
        config.read(self.configfile)
        self.dose_in.delete(0,END)
        self.fluence_in.delete(0,END)
        self.beam_energy_in.delete(0,END)
        self.DUT_num.delete(0,END)
        self.dose_in.insert(0,config.get("beam_setting","dose"))
        self.fluence_in.insert(0,config.get("beam_setting","fluence"))
        self.beam_energy_in.insert(0,config.get("beam_setting","energy"))
        self.DUT_num.insert(0,config.get("beam_setting","dut"))
        self.component_names = [config.get("parts","NFET"),config.get("parts","PFET"),config.get("parts","
            OSC0"),
        config.get("parts","OSC1"),config.get("parts","HBD0"),config.get("parts","HBD1"),config.get("parts","
            AMP0"),config.get("parts","AMP1")]
        self.nfet_base = [ config.getfloat("parts","NFET_th_min"), config.getfloat("parts","NFET_th_max")]
        self.pfet_base = [ config.getfloat("parts","PFET_th_min"), config.getfloat("parts","PFET_th_max")]
        self.osc0_base = [ config.getint("parts","OSC0_hz"), ]
        self.osc1_base = [ config.getint("parts","OSC1_hz"), ]
        self.amp0_base = [config.getfloat("parts","AMP0_low"),config.getfloat("parts","AMP0_hi"),]
```

```
        self.amp1_base = [config.getfloat("parts","AMP1_low"),config.getfloat("parts","AMP1_hi"),]
        self.hbd0_base = [config.getfloat("parts","HBD0_s0_A"),config.getfloat("parts","HBD0_s0_B"),config.
            getfloat("parts","HBD0_s1_A"),config.getfloat("parts","HBD0_s1_B"),]
        self.hbd1_base = [config.getfloat("parts","HBD1_s0_A"),config.getfloat("parts","HBD1_s0_B"),config.
            getfloat("parts","HBD1_s1_A"),config.getfloat("parts","HBD1_s1_B"),]
        self.mem_names = [config.get("parts","MRAM"),config.get("parts","FRAM"),config.get("parts","SFLASH"),
            config.get("parts","QFLASH")]
        self.poll()


    def select_ini(self):
        """Select a test bench configuration file"""
        filetypes = (
            ('config files', '*.ini'),
            ('All files', '*.*')
        )
        self.configfile = fd.askopenfilename(
            title='Open a file',
            initialdir=os.getcwd(),
            filetypes=filetypes)

    def start(self):
        """Starts the test bench in analog mode"""
        self.running = True
        global start_flag
        start_flag.value = True
        self.start_button["command"] = self.stop
        self.start_button["text"] = "ANALOG STOP"
        self._start_time = time.perf_counter()

    def poll_timer(self):
        """Simple timer"""
        self.polled_time = time.perf_counter() - self._start_time

    def dstart(self):
        """Starts the test bench in digital mode"""
        self.running = True
        global d_start_flag
        d_start_flag.value = True
        self.dstart_button["command"] = self.dstop
```

```
        self.dstart_button["text"] = "DIGITAL STOP"

    def stop(self):
        """Stops the test bench"""
        self.running = True
        global start_flag
        start_flag.value = False
        self.start_button["command"] = self.start
        self.start_button["text"] = "ANALOG START"

    def dstop(self):
        """Stops the test bench in digital mode"""
        self.running = True
        global d_start_flag
        d_start_flag.value = False
        self.dstart_button["command"] = self.dstart
        self.dstart_button["text"] = "DIGITAL START"

    def dsend(self):
        """Sends data from software to the digital mode microcontroller"""
        text = self.dig.get()
        global ser
        if len(text) > 0:
            ser.write(bytes(text, encoding='ascii'))
        else:
            ser.write(b'\x33')

    def vds_update(self):
        """Updates the MOSFET drain power supply to a new value"""
        global vds_ser
        text = f"VSET1:{self.vds_val.get()}"
        vds_ser.write(bytes(text, encoding='ascii'))

    def saverun(self):
        """Save screening data after run is finished"""
        for i in range(14):
            th = self.nfet_gate_th_array[i,0]
            print(th)
            print(self.nfet_base[0],self.nfet_base[1])
            if (th > self.nfet_base[0]) and (th < self.nfet_base[1]):
```

```
            self.nfetbroke = False
            self.nfetbrokeval = "N/A"
        else:
            self.nfetbroke = True
            self.nfetbrokeval = i*2
            break
for i in range(14):
    th = self.pfet_gate_th_array[i,0]
    print(th)
    print(self.pfet_base[0],self.pfet_base[1])
    if (th > self.pfet_base[0]) and (th < self.pfet_base[1]):
        self.pfetbroke = False
        self.pfetbrokeval = "N/A"
    else:
        self.pfetbroke = True
        self.pfetbrokeval = i*2
        break

#Large code block aimed at formatting the output XML for the save file
xml_root = ET.Element("TestBenchRoot")
xml_beam = ET.SubElement(xml_root,"RunData")
xml_dut = ET.SubElement(xml_beam, "DUT", name="DUT Board").text = self.DUT_num.get()
xml_dose = ET.SubElement(xml_beam, "TotalDose", name="Total Dose [rad]").text = self.dose_in.get()
xml_fluence = ET.SubElement(xml_beam, "SEEFluence", name="SEE Fluence [p/cm^2]").text = self.
    fluence_in.get()
xml_energy = ET.SubElement(xml_beam, "BeamEnergy", name="Beam Energy [MeV]").text = self.
    beam_energy_in.get()
if  float(self.DUT_num.get()) < 3: #XML format if the screening run was for analog parts
    xml_nfetdata = ET.SubElement(xml_root,"NFETData", name=self.component_names[0])
    base_nfetdata = ET.SubElement(xml_nfetdata,"Base_Data")
    nfet_th_min = ET.SubElement(base_nfetdata,"NFET_Vth_min").text = str(self.nfet_base[0])
    nfet_th_max = ET.SubElement(base_nfetdata,"NFET_Vth_max").text = str(self.nfet_base[1])
    tid_nfetdata = ET.SubElement(xml_nfetdata,"TID_Data")
    see_nfetdata = ET.SubElement(xml_nfetdata,"SEE_Data")
    nfet_krad = [0 for i in range(14)]
    nfet_th = [0 for i in range(14)]
    nfet_arr_curr = [0 for i in range(14)]
    nfet_arr_gate = [0 for i in range(14)]
    for i in range(14):
        nfet_krad[i] = ET.SubElement(tid_nfetdata,f"NFET_{i*2}krad")
```

```
            nfet_th[i] = ET.SubElement(nfet_krad[i],f"NFET_th{i*2}", name=f"Threshold Voltage {i*2} kRad"
                ).text = str(self.nfet_gate_th_array[i,0])
            nfet_arr_curr[i] = ET.SubElement(nfet_krad[i],"NFET_Drain_Array", unit="[A]").text = str(self
                .nfet_gate_array[i,2,0:])
            nfet_arr_gate[i] = ET.SubElement(nfet_krad[i],"NFET_Gate_Array", unit="[V]").text =  str(self
                .nfet_gate_array[i,1,0:])
        nfet_break = ET.SubElement(tid_nfetdata,"NFET_Broke").text = str(self.nfetbroke)
        nfet_break_dose = ET.SubElement(tid_nfetdata,"NFET_Break_Dose", unit="kRad").text = str(self.
            nfetbrokeval)

        nfet_sees = [0 for i in range(16)]
        nfet_seed = [0 for i in range(16)]
        nfet_seeg = [0 for i in range(16)]
        for i in range(16):
            nfet_sees[i] = ET.SubElement(see_nfetdata,f"NFET_{i*2}VDS")
            nfet_seed[i] = ET.SubElement(nfet_sees[i],"DRAIN_SEE_COUNT").text = str(self.nfet_see_array[i
                ,0])
            nfet_seeg[i] = ET.SubElement(nfet_sees[i],"GATE_SEE_COUNT").text = str(self.nfet_see_array[i
                ,1])

        xml_pfetdata = ET.SubElement(xml_root,"PFETData", name=self.component_names[1])
        base_pfetdata = ET.SubElement(xml_pfetdata,"Base_Data")
        pfet_th_min = ET.SubElement(base_pfetdata,"PFET_Vth_min").text = str(self.pfet_base[0])
        pfet_th_max = ET.SubElement(base_pfetdata,"PFET_Vth_max").text = str(self.pfet_base[1])
        tid_pfetdata = ET.SubElement(xml_pfetdata,"TID_Data")
        see_pfetdata = ET.SubElement(xml_pfetdata,"SEE_Data")
        pfet_krad = [0 for i in range(14)]
        pfet_th = [0 for i in range(14)]
        pfet_arr_curr = [0 for i in range(14)]
        pfet_arr_gate = [0 for i in range(14)]
        for i in range(14):
            pfet_krad[i] = ET.SubElement(tid_pfetdata,f"PFET_{i*2}krad")
            pfet_th[i] = ET.SubElement(pfet_krad[i],f"PFET_th{i*2}", name=f"Threshold Voltage {i*2} kRad"
                ).text = str(self.pfet_gate_th_array[i,0])
            pfet_arr_curr[i] = ET.SubElement(pfet_krad[i],"PFET_Drain_Array", unit="[A]").text = str(self
                .pfet_gate_array[i,2,0:])
            pfet_arr_gate[i] = ET.SubElement(pfet_krad[i],"PFET_Gate_Array", unit="[V]").text =  str(self
                .pfet_gate_array[i,1,0:])
        pfet_break = ET.SubElement(tid_pfetdata,"PFET_Broke").text = str(self.pfetbroke)
```

```
    pfet_break_dose = ET.SubElement(tid_pfetdata,"PFET_Break_Dose", unit="kRad").text = str(self.
        pfetbrokeval)

    pfet_sees = [0 for i in range(16)]
    pfet_seed = [0 for i in range(16)]
    pfet_seeg = [0 for i in range(16)]
    for i in range(16):
        pfet_sees[i] = ET.SubElement(see_pfetdata,f"PFET_{i*2}VDS")
        pfet_seed[i] = ET.SubElement(pfet_sees[i],"DRAIN_SEE_COUNT").text = str(self.pfet_see_array[i
            ,0])
        pfet_seeg[i] = ET.SubElement(pfet_sees[i],"GATE_SEE_COUNT").text = str(self.pfet_see_array[i
            ,1])


    xml_oscdata0 = ET.SubElement(xml_root,"OSCData0", name=self.component_names[2])
    base_oscdata0 = ET.SubElement(xml_oscdata0,"Base_Data")
    hz_oscdata0 = ET.SubElement(base_oscdata0,"OSC0_Hz").text = str(self.osc0_base[0])
    tid_oscdata0 = ET.SubElement(xml_oscdata0,"TID_Data")
    osc0_krad = [0 for i in range(14)]
    osc0_krad_hz = [0 for i in range(14)]
    osc0_krad_mA = [0 for i in range(14)]
    for i in range(14):
        osc0_krad[i] = ET.SubElement(tid_oscdata0,f"OSC0_{i*2}krad")
        osc0_krad_hz[i] = ET.SubElement(osc0_krad[i],"OSC0_Hz").text = str(self.osc0_array[1,i])
        osc0_krad_mA[i] = ET.SubElement(osc0_krad[i],"OSC0_Current", unit="[mA]").text = str(self.
            osc0_array[0,i])
    xml_oscdata1 = ET.SubElement(xml_root,"OSCData1", name=self.component_names[3])
    base_oscdata1 = ET.SubElement(xml_oscdata1,"Base_Data")
    hz_oscdata1 = ET.SubElement(base_oscdata1,"OSC1_Hz").text = str(self.osc1_base[0])
    tid_oscdata1 = ET.SubElement(xml_oscdata1,"TID_Data")
    osc1_krad = [0 for i in range(14)]
    osc1_krad_hz = [0 for i in range(14)]
    osc1_krad_mA = [0 for i in range(14)]
    for i in range(14):
        osc1_krad[i] = ET.SubElement(tid_oscdata1,f"OSC1_{i*2}krad")
        osc1_krad_hz[i] = ET.SubElement(osc1_krad[i],"OSC1_Hz").text = str(self.osc1_array[1,i])
        osc1_krad_mA[i] = ET.SubElement(osc1_krad[i],"OSC1_Current", unit="[mA]").text = str(self.
            osc1_array[0,i])
    xml_hbddata0 = ET.SubElement(xml_root,"HBDData0", name=self.component_names[4])
    base_hbddata0 = ET.SubElement(xml_hbddata0,"Base_Data")
```

```
        hbd0_loza = ET.SubElement(base_hbddata0,"HBD0_Low_Out_A").text = str(self.hbd0_base[0])
        hbd0_lozb = ET.SubElement(base_hbddata0,"HBD0_Low_Out_B").text = str(self.hbd0_base[1])
        hbd0_hiza = ET.SubElement(base_hbddata0,"HBD0_High_Out_A").text = str(self.hbd0_base[2])
        hbd0_hizb = ET.SubElement(base_hbddata0,"HBD0_High_Out_B").text = str(self.hbd0_base[3])
        TID_hbddata0 = ET.SubElement(xml_hbddata0,"TID_Data")
        hbd0_krad = [0 for i in range(14)]
        hbd0_broke = [0 for i in range(14)]
        hbd0_loloa = [0 for i in range(14)]
        hbd0_lolob = [0 for i in range(14)]
        hbd0_hihia = [0 for i in range(14)]
        hbd0_hihib = [0 for i in range(14)]
        for i in range(14):
            hbd0_krad[i] = ET.SubElement(TID_hbddata0,f"HBD0_{i*2}krad")
            hbd0_broke[i] = ET.SubElement(hbd0_krad[i],"HBD0_Working").text = str(bool(self.hbd0_array[4,
                i]))
            hbd0_loloa[i] = ET.SubElement(hbd0_krad[i],"HBD0_Low_Out_A").text = str(self.hbd0_array[0,i])
            hbd0_lolob[i] = ET.SubElement(hbd0_krad[i],"HBD0_Low_Out_B").text = str(self.hbd0_array[1,i])
            hbd0_hihia[i] = ET.SubElement(hbd0_krad[i],"HBD0_High_Out_A").text = str(self.hbd0_array[2,i
                ])
            hbd0_hihib[i] = ET.SubElement(hbd0_krad[i],"HBD0_High_Out_B").text = str(self.hbd0_array[3,i
                ])

        xml_hbddata1 = ET.SubElement(xml_root,"HBDData1", name=self.component_names[5])
        base_hbddata1 = ET.SubElement(xml_hbddata1,"Base_Data")
        hbd1_loza = ET.SubElement(base_hbddata1,"HBD0_Low_Out_A").text = str(self.hbd1_base[0])
        hbd1_lozb = ET.SubElement(base_hbddata1,"HBD0_Low_Out_B").text = str(self.hbd1_base[1])
        hbd1_hiza = ET.SubElement(base_hbddata1,"HBD0_High_Out_A").text = str(self.hbd1_base[2])
        hbd1_hizb = ET.SubElement(base_hbddata1,"HBD0_High_Out_B").text = str(self.hbd1_base[3])
        TID_hbddata1 = ET.SubElement(xml_hbddata1,"TID_Data")
        hbd1_krad = [0 for i in range(14)]
        hbd1_broke = [0 for i in range(14)]
        hbd1_loloa = [0 for i in range(14)]
        hbd1_lolob = [0 for i in range(14)]
        hbd1_hihia = [0 for i in range(14)]
        hbd1_hihib = [0 for i in range(14)]
        for i in range(14):
            hbd1_krad[i] = ET.SubElement(TID_hbddata1,f"HBD1_{i*2}krad")
            hbd1_broke[i] = ET.SubElement(hbd1_krad[i],"HBD1_Working").text = str(bool(self.hbd1_array[4,
                i]))
            hbd1_loloa[i] = ET.SubElement(hbd1_krad[i],"HBD1_Low_Out_A").text = str(self.hbd1_array[0,i])
```

```
        hbd1_lolob[i] = ET.SubElement(hbd1_krad[i],"HBD1_Low_Out_B").text = str(self.hbd1_array[1,i])
        hbd1_hihia[i] = ET.SubElement(hbd1_krad[i],"HBD1_High_Out_A").text = str(self.hbd1_array[2,i
            ])
        hbd1_hihib[i] = ET.SubElement(hbd1_krad[i],"HBD1_High_Out_B").text = str(self.hbd1_array[3,i
            ])
565
    xml_ampdata0 = ET.SubElement(xml_root,"AMPData0", name=self.component_names[6])
    base_ampdata0 = ET.SubElement(xml_ampdata0,"Base_Data")
    amp0_loz = ET.SubElement(base_ampdata0,"AMP0_Low_Out").text = str(self.amp0_base[0])
570 amp0_hiz = ET.SubElement(base_ampdata0,"AMP0_High_Out").text = str(self.amp0_base[1])
    TID_ampdata0 = ET.SubElement(xml_ampdata0,"TID_Data")
    amp0_krad = [0 for i in range(14)]
    amp0_broke = [0 for i in range(14)]
    amp0_hihi = [0 for i in range(14)]
575 amp0_lolo = [0 for i in range(14)]
    for i in range(14):
        amp0_krad[i] = ET.SubElement(TID_ampdata0,f"AMP0_{i*2}krad")
        amp0_broke[i] = ET.SubElement(amp0_krad[i],"AMP0_Working").text = str(bool(self.inamp0_array
            [2,i]))
        amp0_lolo[i] = ET.SubElement(amp0_krad[i],"AMP0_Low_Out").text = str(self.inamp0_array[0,i])
580     amp0_hihi[i] = ET.SubElement(amp0_krad[i],"AMP0_High_Out").text = str(self.inamp0_array[1,i])


    xml_ampdata1 = ET.SubElement(xml_root,"AMPData1", name=self.component_names[7])
    base_ampdata1 = ET.SubElement(xml_ampdata1,"Base_Data")
585 amp1_loz = ET.SubElement(base_ampdata1,"AMP1_Low_Out").text = str(self.amp1_base[0])
    amp1_hiz = ET.SubElement(base_ampdata1,"AMP1_High_Out").text = str(self.amp1_base[1])
    TID_ampdata1 = ET.SubElement(xml_ampdata1,"TID_Data")
    amp1_krad = [0 for i in range(14)]
    amp1_broke = [0 for i in range(14)]
590 amp1_hihi = [0 for i in range(14)]
    amp1_lolo = [0 for i in range(14)]
    for i in range(14):
        amp1_krad[i] = ET.SubElement(TID_ampdata1,f"AMP1_{i*2}krad")
        amp1_broke[i] = ET.SubElement(amp1_krad[i],"AMP1_Working").text = str(bool(self.inamp1_array
            [2,i]))
595     amp1_lolo[i] = ET.SubElement(amp1_krad[i],"AMP1_Low_Out").text = str(self.inamp1_array[0,i])
        amp1_hihi[i] = ET.SubElement(amp1_krad[i],"AMP1_High_Out").text = str(self.inamp1_array[1,i])
```

```
        else:
            #XML format if the screening run was for digital parts
            xml_mramdata = ET.SubElement(xml_root,"MRAMData", name=self.mem_names[0])
            see_mramdata = ET.SubElement(xml_mramdata,"SEE_Data")
            mram_cross_static = ET.SubElement(see_mramdata, "StaticCross", name="Static SEU Cross Section [cm
                ^2]").text = str(self.mram_static_cross )
            mram_num_static = ET.SubElement(see_mramdata, "StaticNum", name="Static SEU Count").text = str(
                self.mram_static_upsets)
            mram_cross_dym = ET.SubElement(see_mramdata, "DymCross", name="Dynamic SEU Cross Section [cm^2]")
                .text = str(self.mram_dynamic_cross)
            mram_num_dym = ET.SubElement(see_mramdata, "DymNum", name="Dynamic SEU Count").text = str(self.
                mram_dynamic_upsets)
            mram_static = ET.SubElement(see_mramdata, "StaticBytes").text = str(self.mram_bytes_static)
            mram_dynamic = ET.SubElement(see_mramdata, "DymBytes").text = str(self.mram_bytes_dynamic_out)


            xml_framdata = ET.SubElement(xml_root,"FRAMData", name=self.mem_names[1])
            see_framdata = ET.SubElement(xml_framdata,"SEE_Data")
            fram_cross_static = ET.SubElement(see_framdata, "StaticCross", name="Static SEU Cross Section [cm
                ^2]").text = str(self.fram_static_cross )
            fram_num_static = ET.SubElement(see_framdata, "StaticNum", name="Static SEU Count").text = str(
                self.fram_static_upsets)
            fram_cross_dym = ET.SubElement(see_framdata, "DymCross", name="Dynamic SEU Cross Section [cm^2]")
                .text = str(self.fram_dynamic_cross)
            fram_num_dym = ET.SubElement(see_framdata, "DymNum", name="Dynamic SEU Count").text = str(self.
                fram_dynamic_upsets)
            fram_static = ET.SubElement(see_framdata, "StaticBytes").text = str(self.fram_bytes_static)
            fram_dynamic = ET.SubElement(see_framdata, "DymBytes").text = str(self.fram_bytes_dynamic_out)

            xml_sfdata = ET.SubElement(xml_root,"FlashData", name=self.mem_names[2])
            see_sfdata = ET.SubElement(xml_sfdata,"SEE_Data")
            sf_cross_static = ET.SubElement(see_sfdata, "StaticCross", name="Static SEU Cross Section [cm^2]"
                ).text = str(self.sflash_static_cross )
            sf_num_static = ET.SubElement(see_sfdata, "StaticNum", name="Static SEU Count").text = str(self.
                sflash_static_upsets)
            sf_cross_dym = ET.SubElement(see_sfdata, "DymCross", name="Dynamic SEU Cross Section [cm^2]").
                text = str(self.sflash_dynamic_cross)
            sf_num_dym = ET.SubElement(see_sfdata, "DymNum", name="Dynamic SEU Count").text = str(self.
                sflash_dynamic_upsets)
            sf_static = ET.SubElement(see_sfdata, "StaticBytes").text = str(self.sflash_bytes_static)
```

```
            sf_dynamic = ET.SubElement(see_sfdata, "DymBytes").text = str(self.sflash_bytes_dynamic_out)

            xml_qfdata = ET.SubElement(xml_root,"QuadFlashData", name=self.mem_names[3])
            see_qfdata = ET.SubElement(xml_qfdata,"SEE_Data")
            qf_cross_static = ET.SubElement(see_qfdata, "StaticCross", name="Static SEU Cross Section [cm^2]"
                ).text = str(self.qflash_static_cross )
            qf_num_static = ET.SubElement(see_qfdata, "StaticNum", name="Static SEU Count").text = str(self.
                qflash_static_upsets)
            qf_cross_dym = ET.SubElement(see_qfdata, "DymCross", name="Dynamic SEU Cross Section [cm^2]").
                text = str(self.qflash_dynamic_cross)
            qf_num_dym = ET.SubElement(see_qfdata, "DymNum", name="Dynamic SEU Count").text = str(self.
                qflash_dynamic_upsets)
            qf_static = ET.SubElement(see_qfdata, "StaticBytes").text = str(self.qflash_bytes_static)
            qf_dynamic = ET.SubElement(see_qfdata, "DymBytes").text = str(self.qflash_bytes_dynamic_out)

        xmltree = ET.ElementTree(xml_root)
        ET.indent(xmltree, '    ')
        xmltree.write("TestBench Results.xml", encoding="utf-8", xml_declaration=True)

    def poll(self):
        """
        This method is required to allow the mainloop to receive keyboard
        interrupts when the frame does not have the focus
        """
        if self.do_plot == True:#Saving plots for MOSFET gate threshold without breaking threads
            plt.figure(self.nfet_gate_test_count+1)
            plt.plot(self.nfet_gate_array[self.nfet_gate_test_count,1,0:],self.nfet_gate_array[self.
                nfet_gate_test_count,2,0:],"k*")
            plt.plot(self.nfet_gate_th_array[self.nfet_gate_test_count,0],self.current_threshold,'ko')
            plt.ylabel("Drain Current [A]")
            plt.xlabel("Gate Voltage [V]")
            plt.text(3.10,self.current_threshold,"Gate Threshold")
            plt.axvline(self.nfet_gate_th_array[self.nfet_gate_test_count,0],ls='--')
            plt.title("NFET Gate Vs. Current - DUT "+self.DUT_num.get()+" - "+str(self.nfet_gate_test_count
                *2)+" kRad")
            plt.grid(True)
            plt.show(block=False)
            plt.savefig(f"NFET - {self.nfet_gate_test_count*2}kRad.png")
            plt.figure(self.pfet_gate_test_count+11)
```

```
            plt.plot(self.pfet_gate_array[self.pfet_gate_test_count,1,0:],self.pfet_gate_array[self.
                pfet_gate_test_count,2,0:],"k*")
            plt.plot(self.pfet_gate_th_array[self.pfet_gate_test_count,0],self.current_thresholdp,'ko')
            plt.ylabel("Drain Current [A]")
            plt.xlabel("Gate Voltage [V]")
            plt.text(-3,self.current_thresholdp,"Gate Threshold")
            plt.axvline(self.pfet_gate_th_array[self.pfet_gate_test_count,0],ls='--')
            plt.title("PFET Gate Vs. Current - DUT "+self.DUT_num.get()+" - "+str(self.pfet_gate_test_count
                *2)+" kRad")
            plt.grid(True)
            plt.show(block=False)
            plt.savefig(f"PFET - {self.pfet_gate_test_count*2}kRad.png")
            self.do_plot = False
        self.master.after(250, self.poll)

    def validate_channel_entry(self, p):
        """Validates number of DAQ Channels"""
        if p == '':
            return True
        try:
            value = int(p)
            if value < 0 or value > self.ai_info.num_chans - 1:
                return False
        except ValueError:
            return False

        return True

    def get_low_channel_num(self):
        """Sets the lower range of active DAQ channels"""
        if self.ai_info.num_chans == 1:
            return 0
        try:
            return int(self.low_channel_entry.get())
        except ValueError:
            return 0

    def get_high_channel_num(self):
        """Sets the higher range of active DAQ channels"""
        if self.ai_info.num_chans == 1:
```

```
            return 0
        try:
            return int(self.high_channel_entry.get())
700     except ValueError:
            return 0

    def bit_checkbutton_changed(self, bit_num):
        """Checks for new commands from the user"""
705     try:
            # Get the value from the checkbutton
            bit_value = self.bit_checkbutton_vars[bit_num].get()
            if bit_num != 2:
                global function_bits
710             function_bits[bit_num] = bit_value

                # Output the value to the board
                ul.d_bit_out(self.board_num, self.port.type, bit_num, bit_value)
        except ULError as e:
715         show_ul_error(e)

    def update_value_ctr(self):
        """Grabs latest crystal oscillator data from the DAQ and processes it"""
        ctr_channel = 0
720     ctr_channel1 = 1
        global function_bits

        try:
            # Get a value from the device
725         ul.c_clear(self.board_num, ctr_channel)
            start = time.perf_counter_ns()
            value = ul.c_in_32(self.board_num, ctr_channel)
            time.sleep(2)
            end = time.perf_counter_ns()
730         value2 = ul.c_in_32(self.board_num, ctr_channel)

            out = (value2 - value) / (end - start) * 1000000000

            ul.c_clear(self.board_num, ctr_channel1)
735         start = time.perf_counter_ns()
            value = ul.c_in_32(self.board_num, ctr_channel1)
```

```
            time.sleep(2)
            end = time.perf_counter_ns()
            value2 = ul.c_in_32(self.board_num, ctr_channel1)

            out2 = (value2 - value) / (end - start) * 1000000000




            if function_bits[4] == 1:
                print("counter")
                self.do_ctr = 1
            if self.do_ctr == 1:
                if self.ctr_up < 4:

                    self.osc0_array[1,self.osc_test_count] += out
                    self.osc1_array[1,self.osc_test_count] += out2
                    self.ctr_up += 1
                else:
                    self.osc0_array[1,self.osc_test_count] = round(self.osc0_array[1,self.osc_test_count]/4)
                    self.osc1_array[1,self.osc_test_count] = round(self.osc1_array[1,self.osc_test_count]/4)
                    self.osc0_array[0,self.osc_test_count] = self.mean_array[10]
                    self.osc1_array[0,self.osc_test_count] = self.mean_array[11]
                    self.osc_test_count += 1
                    self.ctr_up = 0
                    self.do_ctr = 0

            # Display the value
            self.ctr_value_label["text"] = str(round(out))
            self.ctr1_value_label["text"] = str(round(out2))

        except ULError as e:
            self.stop()
            show_ul_error(e)


    def AMP_TID_Processor(self):
        """Grabs latest in-amp and h-bridge data from the DAQ and processes it"""
        global function_bits
        if function_bits[4] == 1:
            self.do_amp = 1
```

```
        print("inamp & hbd")
    if self.do_amp == 1:
        ul.d_bit_out(self.board_num, self.port.type, 2, 0)
        sleep(10)
        self.inamp0_array[0,self.amp_test_count] = self.mean_array[2]
        self.inamp1_array[0,self.amp_test_count] = self.mean_array[3]



        self.hbd0_array[0,self.amp_test_count] = self.mean_array[4]
        self.hbd0_array[1,self.amp_test_count] = self.mean_array[5]

        self.hbd1_array[0,self.amp_test_count] = self.mean_array[6]
        self.hbd1_array[1,self.amp_test_count] = self.mean_array[7]


        ul.d_bit_out(self.board_num, self.port.type, 2, 1)
        sleep(10)
        self.inamp0_array[1,self.amp_test_count] = self.mean_array[2]
        self.inamp1_array[1,self.amp_test_count] = self.mean_array[3]

        self.hbd0_array[2,self.amp_test_count] = self.mean_array[4]
        self.hbd0_array[3,self.amp_test_count] = self.mean_array[5]

        self.hbd1_array[2,self.amp_test_count] = self.mean_array[6]
        self.hbd1_array[3,self.amp_test_count] = self.mean_array[7]

        if ( 1.05 >= self.inamp0_array[0,self.amp_test_count]/self.amp0_base[0] >= 0.95) and ( 1.05 >=
            self.inamp0_array[1,self.amp_test_count]/self.amp0_base[1] >= 0.95):
            self.inamp0_array[2,self.amp_test_count] = True

        if (1.05>=self.inamp1_array[0,self.amp_test_count]/self.amp1_base[0] >= 0.95) and (1.05>= self.
            inamp1_array[1,self.amp_test_count]/self.amp1_base[1] >= 0.95):
            self.inamp1_array[2,self.amp_test_count] = True

        if (1.05>=(self.hbd0_array[0,self.amp_test_count]+1)/(self.hbd0_base[0]+1) >=0.95) and (1.05>=(
            self.hbd0_array[1,self.amp_test_count]+1)/(self.hbd0_base[1]+1) >=0.95):
            if (1.05>=(self.hbd0_array[2,self.amp_test_count]+1)/(self.hbd0_base[2]+1) >=0.95) and
                (1.05>=(self.hbd0_array[3,self.amp_test_count]+1)/(self.hbd0_base[3]+1) >=0.95):
                self.hbd0_array[4,self.amp_test_count] = True
```

```
            if (1.05>=(self.hbd1_array[0,self.amp_test_count]+1)/(self.hbd1_base[0]+1) >=0.95) and (1.05>=(
                self.hbd1_array[1,self.amp_test_count]+1)/(self.hbd1_base[1]+1) >=0.95):
                if (1.05>=(self.hbd1_array[2,self.amp_test_count]+1)/(self.hbd1_base[2]+1) >=0.95) and
                    (1.05>=(self.hbd1_array[3,self.amp_test_count]+1)/(self.hbd1_base[3]+1) >=0.95):
                    self.hbd1_array[4,self.amp_test_count] = True

            print(self.hbd0_array[4,self.amp_test_count])
            print(self.hbd1_array[4,self.amp_test_count])
            self.amp_test_count +=1

            self.do_amp = 0
            sleep(5)
        else:
            ul.d_bit_out(self.board_num, self.port.type, 2, 0)
            sleep(1)
            ul.d_bit_out(self.board_num, self.port.type, 2, 1)
            sleep(1)




def FET_SEE_Processor(self):
    """Automatic analysis of MOSFET SEE data"""
    global function_bits
    print(self.mean_array[1])
    print(self.mean_array[0])
    print(self.mean_array[8])
    print(self.mean_array[9])
    if abs(float(self.mean_array[1])) > 12:
        #see occured in drain leakage NFET
        self.nfet_see_array[int(int(self.vds_val.get())/2),0] += 1
        print("NFET DRAIN SEES: "+str(self.nfet_see_array[int(int(self.vds_val.get())/2),0]))

    if abs(float(self.mean_array[0])) > 12:
        #see occured in drain leakage PFET
        self.pfet_see_array[int(int(self.vds_val.get())/2),0] += 1
        print("PFET DRAIN SEES: "+str(self.pfet_see_array[int(int(self.vds_val.get())/2),0]))

    if abs(float(self.mean_array[8])) > 5:
```

```
            #see occured in gate leakage NFET
            self.nfet_see_array[int(int(self.vds_val.get())/2),1] += 1
            print("NFET GATE SEES: "+str(self.nfet_see_array[int(int(self.vds_val.get())/2),1]))
        if abs(float(self.mean_array[9])) > 5:
            #see occured in gate leakage PFET
            self.pfet_see_array[int(int(self.vds_val.get())/2),1] += 1
            print("PFET GATE SEES: "+str(self.pfet_see_array[int(int(self.vds_val.get())/2),1]))



def FET_TID_Processor(self):
    """Automatic analysis of MOSFET TID data"""
    global function_bits
    if function_bits[4] == 1:
        self.do_gate = 1
        print("fet")
    if self.do_gate == 1:
        time.sleep(0.5)
        etime = time.perf_counter() - self._start_time
        self.nfet_gate_array[self.nfet_gate_test_count,0,self.gate_counter] = etime
        self.pfet_gate_array[self.pfet_gate_test_count,0,self.gate_counter] = etime
        self.nfet_gate_array[self.nfet_gate_test_count,1,self.gate_counter] = self.gate_counter*0.2
        self.pfet_gate_array[self.pfet_gate_test_count,1,self.gate_counter] = self.gate_counter*(-0.2)
        self.nfet_gate_array[self.nfet_gate_test_count,2,self.gate_counter] = round(self.mean_array[1],4)
        self.pfet_gate_array[self.pfet_gate_test_count,2,self.gate_counter] = round(self.mean_array[0],4)
        self.gate_counter += 1
        #code block to determine gate threshold voltage
        if self.gate_counter >= 32:
            self.current_threshold = (self.nfet_gate_array[self.nfet_gate_test_count,2,0]+self.
                nfet_gate_array[self.nfet_gate_test_count,2,31])/2
            self.current_thresholdp = (self.pfet_gate_array[self.pfet_gate_test_count,2,0]+self.
                pfet_gate_array[self.pfet_gate_test_count,2,31])/2
            self.do_gate = 0
            self.gate_counter = 0
            for i in range(32):
                if self.nfet_gate_array[self.nfet_gate_test_count,2,i]  >= self.current_threshold:
                    self.nfet_gate_th_array[self.nfet_gate_test_count,0] = self.nfet_gate_array[self.
                        nfet_gate_test_count,1,(i-1)] + \
```

```
                    (self.nfet_gate_array[self.nfet_gate_test_count,1,i]-self.nfet_gate_array[self.
                        nfet_gate_test_count,1,(i-1)])* \
                    (self.current_threshold-self.nfet_gate_array[self.nfet_gate_test_count,2,(i-1)])/(
                        self.nfet_gate_array[self.nfet_gate_test_count,2,i] \
                    -self.nfet_gate_array[self.nfet_gate_test_count,2,(i-1)])
                    print(self.nfet_gate_th_array[self.nfet_gate_test_count,0])
                    break


            for i in range(32):
                if self.pfet_gate_array[self.pfet_gate_test_count,2,i]  >= self.current_thresholdp:
                    self.pfet_gate_th_array[self.pfet_gate_test_count,0] = self.pfet_gate_array[self.
                        pfet_gate_test_count,1,(i-1)] + \
                    (self.pfet_gate_array[self.pfet_gate_test_count,1,i]-self.pfet_gate_array[self.
                        pfet_gate_test_count,1,(i-1)])* \
                    (self.current_thresholdp-self.pfet_gate_array[self.pfet_gate_test_count,2,(i-1)])/(
                        self.pfet_gate_array[self.pfet_gate_test_count,2,i] \
                    -self.pfet_gate_array[self.pfet_gate_test_count,2,(i-1)])
                    print(self.pfet_gate_th_array[self.pfet_gate_test_count,0])
                    break
            self.do_plot = True
            sleep(6)
            self.nfet_gate_test_count+=1
            self.pfet_gate_test_count+=1
        time.sleep(5)


def get_secs(self,plusser):
    global maxx
    global minn
    minn = maxx
    maxx += plusser
    return maxx, minn

def update_value(self):
    """Grabs latest analog data from DAQ"""
    low_chan = self.get_low_channel_num()
    high_chan = self.get_high_channel_num()
    global maxx
    global minn
```

```
        if low_chan > high_chan:
            messagebox.showerror(
                "Error",
                "Low Channel Number must be greater than or equal to High "
                "Channel Number")
            self.start_button["state"] = tk.NORMAL
            return

        rate = 100
        file_name = self.file_name_in.get()
        memhandle = None
        buffer_size_seconds = 2
        num_buffers_to_write = 2
        plusser = buffer_size_seconds*num_buffers_to_write
        maxx, minn = self.get_secs(plusser)
        secs = np.arange(minn,maxx,1/rate)
        num_channels = high_chan - low_chan + 1
        points_per_channel = max(rate * buffer_size_seconds, 10)
        if self.ai_info.packet_size != 1:
            packet_size = self.ai_info.packet_size
            remainder = points_per_channel % packet_size
            if remainder != 0:
                points_per_channel += packet_size - remainder
        ul_buffer_count = points_per_channel * num_channels

        # Write the UL buffer to the file num_buffers_to_write times.
        points_to_write = ul_buffer_count * num_buffers_to_write

        # When handling the buffer, we will read 1/10 of the buffer at a time
        write_chunk_size = int(ul_buffer_count / 10)
        ai_range = self.ai_info.supported_ranges[0]
        scan_options = (ScanOptions.BACKGROUND | ScanOptions.CONTINUOUS |
                        ScanOptions.SCALEDATA)

        # Allocate a buffer for the scan
        memhandle = ul.scaled_win_buf_alloc(ul_buffer_count)

        # Allocate an array of doubles temporary storage of the data
        self.write_chunk_array = (c_double * write_chunk_size)()
```

```
# Check if the buffer was successfully allocated
if not memhandle:
    raise Exception('Failed to allocate memory')
array = cast(memhandle, POINTER(c_double))
# Start the scan
ul.a_in_scan(
    self.board_num, low_chan, high_chan, ul_buffer_count,
    rate, ai_range, memhandle, scan_options)
status = Status.IDLE
# Wait for the scan to start fully
while status == Status.IDLE:
    status, _, _ = ul.get_status(self.board_num, FunctionType.AIFUNCTION)
with open(file_name, 'a+') as f:
    #print('Writing data to ' + file_name, end='')
    # Write a header to the file
    #print(minn)
    if minn == 0.0:
        f.write('Seconds,')
        for chan_num in range(low_chan, high_chan + 1):
            f.write('Channel ' + str(chan_num) + ',')
        f.write(u'\n')
    nnn=0
    # Start the write loop
    prev_count = 0
    prev_index = 0
    write_ch_num = low_chan
    while status != Status.IDLE:
        # Get the latest counts
        status, curr_count, _ = ul.get_status(self.board_num,
                                              FunctionType.AIFUNCTION)

        new_data_count = curr_count - prev_count

        # Check for a buffer overrun before copying the data, so
        # that no attempts are made to copy more than a full buffer
        # of data
        if new_data_count > ul_buffer_count:
            # Print an error and stop writing
            ul.stop_background(self.board_num, FunctionType.AIFUNCTION)
```

```
            print('A buffer overrun occurred')
            break

        # Check if a chunk is available

        if new_data_count > write_chunk_size:
            wrote_chunk = True
            # Copy the current data to a new array

            # Check if the data wraps around the end of the UL
            # buffer. Multiple copy operations will be required.
            if prev_index + write_chunk_size > ul_buffer_count - 1:
                first_chunk_size = ul_buffer_count - prev_index
                second_chunk_size = (
                    write_chunk_size - first_chunk_size)

                # Copy the first chunk of data to the
                # write_chunk_array
                ul.scaled_win_buf_to_array(
                    memhandle, self.write_chunk_array, prev_index,
                    first_chunk_size)

                # Create a pointer to the location in
                # write_chunk_array where we want to copy the
                # remaining data
                second_chunk_pointer = cast(addressof(self.write_chunk_array)
                                            + first_chunk_size
                                            * sizeof(c_double),
                                            POINTER(c_double))

                # Copy the second chunk of data to the
                # write_chunk_array
                ul.scaled_win_buf_to_array(
                    memhandle, second_chunk_pointer,
                    0, second_chunk_size)
            else:
                # Copy the data to the write_chunk_array
                ul.scaled_win_buf_to_array(
                    memhandle, self.write_chunk_array, prev_index,
                    write_chunk_size)
```

```
            # Check for a buffer overrun just after copying the data
            # from the UL buffer. This will ensure that the data was
            # not overwritten in the UL buffer before the copy was
            # completed. This should be done before writing to the
            # file, so that corrupt data does not end up in it.
            status, curr_count, _ = ul.get_status(
                self.board_num, FunctionType.AIFUNCTION)
            if curr_count - prev_count > ul_buffer_count:
                # Print an error and stop writing
                ul.stop_background(self.board_num, FunctionType.AIFUNCTION)
                print('A buffer overrun occurred')
                break
            self.display_values(self.write_chunk_array, write_chunk_size, low_chan, high_chan)



            if nnn ==0:
                f.write("{:.2f}".format(secs[nnn])+',')
            for i in range(write_chunk_size):
                f.write(str(self.write_chunk_array[i]) + ',')
                #print(write_chunk_array[i])
                write_ch_num += 1
                if write_ch_num == high_chan + 1:
                    write_ch_num = low_chan
                    f.write(u'\n')
                    nnn+=1
                    if nnn < len(secs):
                        f.write("{:.2f}".format(secs[nnn])+',')
        else:
            wrote_chunk = False

        if wrote_chunk:
            # Increment prev_count by the chunk size
            prev_count += write_chunk_size
            # Increment prev_index by the chunk size
            prev_index += write_chunk_size
            # Wrap prev_index to the size of the UL buffer
            prev_index %= ul_buffer_count

            if prev_count >= points_to_write:
```

```
                    break
                #print('.', end='')
            else:
                # Wait a short amount of time for more data to be
                # acquired.
                sleep(0.1)

    ul.stop_background(self.board_num, FunctionType.AIFUNCTION)


def display_values(self, array, total_count, low_chan, high_chan):
    """GUI code for displaying and updating live values of part parameters"""
    new_data_frame = tk.Frame(self.results_group)
    global function_bits
    channel_text = ["PFET [A]\n","NFET [A]\n","InAmp 1 [V]\n","InAmp2 [V]\n","HBD1 A [V]\n","HBD1 B [V]\n
        ","HBD2 A [V]\n","HBD2 B [V]\n","NFET Gate [uA]\n","PFET Gate [uA]\n","OSC0 [mA]\n","OSC1 [mA]\n"
        ]
    if function_bits[5] == 1:
        channel_text[0] = "PFET [A]\n"
        channel_text[1] = "NFET [A]\n"
    elif function_bits[5] == 0:
        channel_text[0] = "PFET Leak [uA]\n"
        channel_text[1] = "NFET Leak [uA]\n"
    # Add the headers
    chan_count = high_chan - low_chan + 1
    # Add (up to) the first 10 values and average them for each channel to the text
    chan_num = low_chan
    self.mean_array = np.arange(low_chan,high_chan+1,dtype=float)
    self.mean_array = self.mean_array * 0
    for data_index in range(0, min(chan_count * 10, total_count)):
        self.mean_array[chan_num] += array[data_index]
        chan_num = low_chan if chan_num == high_chan else chan_num + 1
    self.mean_array = self.mean_array/10

    #various conversions to change voltage readings to current readings
    self.mean_array[8] = self.mean_array[8]* 1000000 / (100*4700)
    self.mean_array[9] = self.mean_array[9]* 1000000 / (100*4700)

    self.mean_array[10] = self.mean_array[10] * 10 / 2.6
    self.mean_array[11] = self.mean_array[11] * 10 / 2.6
```

```
        if function_bits[5] == 1:
            self.mean_array[0] = self.mean_array[0] / (100*0.95)
            self.mean_array[1] = self.mean_array[1] / (100*0.95)
        elif function_bits[5] == 0:
            self.mean_array[0] = self.mean_array[0]* 1000000 / (100*4700)
            self.mean_array[1] = self.mean_array[1]* 1000000 / (100*4700)

        # Add the labels for each channel
        for chan_num in range(low_chan, high_chan + 1):
            chan_label = tk.Label(new_data_frame, justify=tk.LEFT, padx=3)
            chan_label["text"] = channel_text[chan_num - low_chan] + "{:.3f}".format(self.mean_array[chan_num
                -low_chan]) + "\n"
            chan_label.grid(row=0, column=chan_num - low_chan)
        self.data_frame.destroy()
        self.data_frame = new_data_frame
        self.data_frame.grid()


    def get_channel_num(self):
        if self.ai_info.num_chans == 1:
            return 0
        try:
            return int(self.channel_entry.get())
        except ValueError:
            return 0


    def digital_analyze(self, raw_data):
        """Automated analysis of digital memory"""
        end_sec = raw_data.find(b"byt")
        self.current_cs = int(raw_data[3:4])
        self.current_mode = raw_data[4:5].decode()
        self.curr_sector = int(raw_data[12:end_sec])
        #error catch for when test bench fails to send a packet
        try:
            self.curr_bytes = struct.unpack('128B',raw_data[(end_sec+6):-1])
        except:
            print("packet miss")
            lllen = len(raw_data[(end_sec+6):-1])
            self.curr_bytes = struct.unpack(f'{lllen}B',raw_data[(end_sec+6):-1])
```

```
        print(f"current cs: {self.current_cs}, current_sec: {self.curr_sector}")

        if self.current_mode == "d": #checking memory for upsets dynamically
            if self.current_cs == 0:
                self.mram_bytes_dynamic.extend(self.curr_bytes)
                print(self.mram_bytes_dynamic)
                if self.curr_sector >= 2:
                    for b in range(len(self.mram_bytes_dynamic)):
                        if self.mram_bytes_dynamic[b] != 85:
                            self.mram_dynamic_upsets += 1
                    self.mram_dynamic_cross = self.mram_dynamic_upsets  / float(self.fluence_in.get())
                    print(f"dynamic upset count: {self.mram_dynamic_upsets}")
                    print(f"dynamic upset cross section: {self.mram_dynamic_cross}")
                    self.mram_bytes_dynamic_out = self.mram_bytes_dynamic
                    self.mram_bytes_dynamic = None
                    self.mram_bytes_dynamic = []

            if self.current_cs == 1:
                self.fram_bytes_dynamic.extend(self.curr_bytes)
                print(self.fram_bytes_dynamic)

                if self.curr_sector >= 2:
                    for b in range(len(self.fram_bytes_dynamic)):
                        if self.fram_bytes_dynamic[b] != 85:
                            self.fram_dynamic_upsets += 1
                    self.fram_dynamic_cross = self.fram_dynamic_upsets  / float(self.fluence_in.get())
                    print(f"dynamic upset count: {self.fram_dynamic_upsets}")
                    print(f"dynamic upset cross section: {self.fram_dynamic_cross}")
                    self.fram_bytes_dynamic_out = self.fram_bytes_dynamic
                    self.fram_bytes_dynamic = None
                    self.fram_bytes_dynamic = []

            if self.current_cs == 2:
                self.sflash_bytes_dynamic.extend(self.curr_bytes)
                print(self.sflash_bytes_dynamic)

                if self.curr_sector >= 2:
                    for b in range(len(self.sflash_bytes_dynamic)):
                        if self.sflash_bytes_dynamic[b] != 85:
                            self.sflash_dynamic_upsets += 1
```

```
                self.sflash_dynamic_cross = self.sflash_dynamic_upsets  / float(self.fluence_in.get())
                print(f"dynamic upset count: {self.sflash_dynamic_upsets}")
                print(f"dynamic upset cross section: {self.sflash_dynamic_cross}")
                self.sflash_bytes_dynamic_out = self.sflash_bytes_dynamic
                self.sflash_bytes_dynamic = None
                self.sflash_bytes_dynamic = []

            if self.current_cs == 3:
                self.qflash_bytes_dynamic.extend(self.curr_bytes)
                print(self.qflash_bytes_dynamic)

                if self.curr_sector >= 2:
                    for b in range(len(self.qflash_bytes_dynamic)):
                        if self.qflash_bytes_dynamic[b] != 85:
                            self.qflash_dynamic_upsets += 1
                    self.qflash_dynamic_cross = self.qflash_dynamic_upsets  / float(self.fluence_in.get())
                    print(f"dynamic upset count: {self.qflash_dynamic_upsets}")
                    print(f"dynamic upset cross section: {self.qflash_dynamic_cross}")
                    self.qflash_bytes_dynamic_out = self.qflash_bytes_dynamic
                    self.qflash_bytes_dynamic = None
                    self.qflash_bytes_dynamic = []

        if self.current_mode == "s": #checking memory for upsets statically
            if self.current_cs == 0:
                self.mram_bytes_static.extend(self.curr_bytes)
                print(self.mram_bytes_static)
                if self.curr_sector >= 2:
                    for b in range(len(self.mram_bytes_static)):
                        if self.mram_bytes_static[b] != 85:
                            self.mram_static_upsets += 1
                    self.mram_static_cross = self.mram_static_upsets  / float(self.fluence_in.get())
                    print(f"static upset count: {self.mram_static_upsets}")
                    print(f"static upset cross section: {self.mram_static_cross}")

            if self.current_cs == 1:
                self.fram_bytes_static.extend(self.curr_bytes)
                print(self.fram_bytes_static)
                if self.curr_sector >= 2:
                    for b in range(len(self.fram_bytes_static)):
                        if self.fram_bytes_static[b] != 85:
```

```
                        self.fram_static_upsets += 1
                self.fram_static_cross = self.fram_static_upsets  / float(self.fluence_in.get())
                print(f"static upset count: {self.fram_static_upsets}")
                print(f"static upset cross section: {self.fram_static_cross}")

            if self.current_cs == 2:
                self.sflash_bytes_static.extend(self.curr_bytes)
                print(self.sflash_bytes_static)

                if self.curr_sector >= 2:
                    for b in range(len(self.sflash_bytes_static)):
                        if self.sflash_bytes_static[b] != 85:
                            self.sflash_static_upsets += 1
                    self.sflash_static_cross = self.sflash_static_upsets  / float(self.fluence_in.get())
                    print(f"static upset count: {self.sflash_static_upsets}")
                    print(f"static upset cross section: {self.sflash_static_cross}")

            if self.current_cs == 3:
                self.qflash_bytes_static.extend(self.curr_bytes)
                print(self.qflash_bytes_static)
                if self.curr_sector >= 2:
                    for b in range(len(self.qflash_bytes_static)):
                        if self.qflash_bytes_static[b] != 85:
                            self.qflash_static_upsets += 1
                    self.qflash_static_cross = self.qflash_static_upsets  / float(self.fluence_in.get())
                    print(f"static upset count: {self.qflash_static_upsets}")
                    print(f"static upset cross section: {self.qflash_static_cross}")


def worker_function(quit_flag, app_thing):
    """Background thread for exiting the program"""
    counter = 0
    while not quit_flag.value:
        counter += 1
        time.sleep(1.0)

def analog_function(running, appp):
    """Thread function to control analog part screening"""
    counter = 0
    global start_flag
```

```
    while not running.value:
        if start_flag.value == True:
            counter += 1
            appp.update_value()
            appp.poll_timer()
            time.sleep(.25)
        else:
            time.sleep(.25)
            counter = 0


def counter_function(running, appp):
    """Thread function to control oscillator screening"""
    global start_flag
    while not running.value:
        if start_flag.value == True:
            appp.update_value_ctr()
            time.sleep(.25)
        else:
            time.sleep(.25)


def digital_function(running, appp):
    """Thread function to control Digital memory screening"""
    counter = 0
    global d_start_flag
    global ser
    try:
        ser = serial.Serial('COM10', 115200, timeout=10, xonxoff=False, rtscts=False, dsrdtr=False)
    except:
        print("Digital DUT not connected!")
    while not running.value:
        if d_start_flag.value == False:
            time.sleep(.25)
            counter = 0
        else:
            print("timeout\n")
            try:
                data_raw = ser.readline()
            except:
                data_raw = b"no_data"
            print(data_raw)
```

```
            if data_raw[0:2].decode('utf-8','replace') == "CS":
                appp.digital_analyze(data_raw)
            counter += 1
            time.sleep(.1)

def FET_function(running, appp):
    """Thread function to control MOSFET part screening"""
    global start_flag
    global function_bits
    delay = True
    while not running.value:

        if start_flag.value == True:

            if delay == True:
                sleep(5)
                delay = False

            if function_bits[5] == 1:
                appp.FET_TID_Processor()
            else:
                appp.FET_SEE_Processor()
            time.sleep(.25)
        else:
            time.sleep(.25)


def AMP_function(running, appp):
    """Thread function to control In-Amp part screening"""
    global start_flag
    while not running.value:
        if start_flag.value == True:
            appp.AMP_TID_Processor()
            time.sleep(.25)
        else:
            time.sleep(.25)
            try:
                ul.d_bit_out(0, DigitalIODirection.OUT, 2, 0)
            except:
                print("not init")
```

```
1360 def HBD_function(running, appp):
        """Thread function to control H-Bridge Driver part screening"""
        global start_flag
        while not running.value:
            if start_flag.value == True:
1365            appp.HBD_TID_Processor()
                time.sleep(.25)
            else:
                time.sleep(.25)


1370
    #Initialization of software multi threading
    format = '%(levelname)s: %(filename)s: %(lineno)d: %(message)s'
    logging.basicConfig(level=logging.DEBUG, format=format)
    root = Tk()
1375 root.title('TestBench Control Software')
    app = Application(master=root)
    minn = 0.0
    maxx = 0.0
    function_bits = [0,0,0,0,0,0,0,0]
1380 ser = None
    do_plot = False
    quit_flag = multiprocessing.Value('i', int(False))
    start_flag = multiprocessing.Value('i', int(False))
    d_start_flag = multiprocessing.Value('i', int(False))
1385 worker_thread = threading.Thread(target=worker_function, args=(quit_flag,app.quit_button))
    analog_thread = threading.Thread(target=analog_function, args=(quit_flag,app))
    counter_thread = threading.Thread(target=counter_function, args=(quit_flag,app))
    digital_thread = threading.Thread(target=digital_function, args=(quit_flag,app))
    FET_thread = threading.Thread(target=FET_function, args=(quit_flag,app))
1390 AMP_thread = threading.Thread(target=AMP_function, args=(quit_flag,app))
    worker_thread.start()
    analog_thread.start()
    counter_thread.start()
    FET_thread.start()
1395 AMP_thread.start()
    digital_thread.start()
    logging.info("quit_flag.value = %s" % bool(quit_flag.value))
    try:
```

```
1400        app.mainloop()
    except KeyboardInterrupt:
        logging.info("Keyboard interrupt")
    quit_flag.value = True
    logging.info("quit_flag.value = %s" % bool(quit_flag.value))
1405 worker_thread.join()
```

LISTING B.1: TestBenchFinal.py

```
[beam_setting]
dose = 25000
fluence = 1.7e10
energy = 105
dut = 1.0

[parts]
NFET = BUK7M
NFET_th_min = 2.4
NFET_th_max = 3.6

PFET = BSS84A
PFET_th_min = -2.5
PFET_th_max = -1.0



OSC0 = 7X-20
OSC0_hz = 20000000

OSC1 = 8W-13
OSC1_hz = 13000000



HBD0 = TC442
HBD0_s0_A = 8.0
HBD0_s0_B = 0.0
HBD0_s1_A = 0.0
HBD0_s1_B = 8.0

HBD1 = TPS2822
HBD1_s0_A = 2.2
HBD1_s0_B = 8.0
HBD1_s1_A = 7.2
HBD1_s1_B = 8.0

AMP0 = AD524
AMP0_hi = 5.25
AMP0_low = 1.30
```

```
AMP1 = INA321
AMP1_hi = 5.0
AMP1_low = 1.25

MRAM = NA

FRAM = NA

SFLASH = NA

QFLASH = NA
```

LISTING B.2: DUT1 Config.ini

```
[beam_setting]
dose = 25000
fluence = 1.7e10
energy = 105
dut = 2.0

[parts]
NFET = DMNH6008
NFET_th_min = 2.0
NFET_th_max = 4.0

PFET = SSM6J808X
PFET_th_min = -2.0
PFET_th_max = -0.8


OSC0 = AU-12
OSC0_hz = 12288000

OSC1 = AW-11
OSC1_hz = 11289600


HBD0 = DGD054X
HBD0_s0_A = 0.0
HBD0_s0_B = 8.0
HBD0_s1_A = 7.35
HBD0_s1_B = 0.0

HBD1 = ISL784X
HBD1_s0_A = 0.0
HBD1_s0_B = 8.0
HBD1_s1_A = 7.55
HBD1_s1_B = 0.0


AMP0 = LM741
AMP0_hi = 5.00
```

```
AMP0_low = 1.25

AMP1 = TSV911
AMP1_hi = 5.0
AMP1_low = 1.25

MRAM = NA

FRAM = NA

SFLASH = NA

QFLASH = NA
```

LISTING B.3: DUT2 Config.ini

```
[beam_setting]
dose = 1500
fluence = 1.7e10
energy = 105
dut = 3.0

[parts]
NFET = NA
NFET_th_min = 0
NFET_th_max = 0

PFET = NA
PFET_th_min = 0
PFET_th_max = 0



OSC0 = NA
OSC0_hz = 0

OSC1 = NA
OSC1_hz = 0



HBD0 = NA
HBD0_s0_A = 0.0
HBD0_s0_B = 0.0
HBD0_s1_A = 0.0
HBD0_s1_B = 0.0

HBD1 = NA
HBD1_s0_A = 0.0
HBD1_s0_B = 0.0
HBD1_s1_A = 0.0
HBD1_s1_B = 0.0

AMP0 = NA
AMP0_hi = 0.0
AMP0_low = 0.0
```

```
AMP1 = NA
AMP1_hi = 0.0
AMP1_low = 0.0

MRAM = MR25HXX

FRAM = MB85RS2

SFLASH = S25FL512

QFLASH = S25FL032
```

LISTING B.4: Digital DUT Config.ini

```
# -*- mode: python ; coding: utf-8 -*-
block_cipher = None
a = Analysis(['testbenchfinal.py'],
    pathex=['E:\\New folder (2)\\Python','E:\\New folder (2)\\Python\\examples\\ui'],
    binaries=[],
    datas=[('E:\\New folder (2)\\Python\\examples\\ui\\logo.ico','.')],
    hiddenimports=['ui_examples_util',],
    hookspath=[],
    hooksconfig={},
    runtime_hooks=[],
    excludes=[],
    win_no_prefer_redirects=False,
    win_private_assemblies=False,
    cipher=block_cipher,
    noarchive=False,)
pyz = PYZ(a.pure, a.zipped_data, cipher=block_cipher)
exe = EXE(pyz,
    a.scripts,
    a.binaries,
    a.zipfiles,
    a.datas,
    [],
    name='testbenchfinal',
    debug=False,
    bootloader_ignore_signals=False,
    strip=False,
    upx=True,
    icon='E:\\New folder (2)\\Python\\examples\\ui\\logo.ico',
    upx_exclude=[],
    runtime_tmpdir=None,
    console=True,
    disable_windowed_traceback=False,
    argv_emulation=False,
    target_arch=None,
    codesign_identity=None,
    entitlements_file=None,)
```

LISTING B.5: TestBenchFinal.spec

B.2 Analog Micro-controller Source Code

```
#pragma config FOSC = INTRC_NOCLKOUT    // Oscillator Selection bits (XT oscillator)
#pragma config WDTE = OFF  // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = OFF // Power-up Timer Enable bit (PWRT disabled)
#pragma config BOREN = OFF // Brown-out Reset Enable bit (BOR disabled)
#pragma config LVP = OFF   // Low-Voltage In-Circuit Serial Programming Enable bit
#pragma config CPD = OFF   // Data EEPROM Memory Code Protection bit
#pragma config WRT = OFF   // Flash Program Memory Write Enable bits
#pragma config CP = OFF    // Flash Program Memory Code Protection bit

#include <xc.h>
#include "PIC16F887_SPI.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#define _XTAL_FREQ 15000000

#define FET_Reg_Sense_EN     PORTEbits.RE0 //leakage when low, regular when high
#define FET_Reg_Sense_TRI    TRISEbits.TRISE0 //set as digital port
#define FET_Reg_Sense_ANS    ANSELbits.ANS5 //disable analog port
#define FET_Gate_DIS         PORTCbits.RC0 //disables the gate when high for leakage monitor
#define FET_Gate_TRI         TRISCbits.TRISC0 //set as digital port
#define XTAL1_EN             PORTAbits.RA0
#define XTAL1_TRI            TRISAbits.TRISA0
#define XTAL1_ANS            ANSELbits.ANS0
#define XTAL2_EN             PORTAbits.RA1
#define XTAL2_TRI            TRISAbits.TRISA1
#define XTAL2_ANS            ANSELbits.ANS1
#define DAQ_port_init        TRISD //initializer for DAQ GPIO
#define Slot_EN              PORTDbits.RD1 //slot enable port
#define FET_mode             PORTDbits.RD2 //SEE when low, TID when high
#define Gate_test_trigger    PORTDbits.RD3 //when high, do a gate test
#define XTAL_EN_INVRT        PORTDbits.RD4 //DUT2 needs inverted enable signals
#define Inamp_HBD_state      PORTDbits.RD5 //bool for high or low on inamps and hbds
#define inamp_CS             PORTCbits.RC6 //chip select for inamps
#define inamp_CS_TRI         TRISCbits.TRISC6
#define inamp_LDAC           PORTCbits.RC7 //inamp LDAC
#define inamp_LDAC_TRI       TRISCbits.TRISC7
#define FET_CS               PORTCbits.RC1
```

```
#define FET_CS_TRI          TRISCbits.TRISC1
#define FET_LDAC            PORTCbits.RC2
#define FET_LDAC_TRI        TRISCbits.TRISC2
#define HBD1_INA            PORTAbits.RA5
#define HBD1_INA_TRI        TRISAbits.TRISA5
#define HBD1_INA_ANS        ANSELbits.ANS4
#define HBD1_INB            PORTAbits.RA6
#define HBD1_INB_TRI        TRISAbits.TRISA6
#define HBD1_EN             PORTAbits.RA7
#define HBD1_EN_TRI         TRISAbits.TRISA7
#define HBD2_INA            PORTBbits.RB0
#define HBD2_INA_TRI        TRISBbits.TRISB0
#define HBD2_INA_ANS        ANSELHbits.ANS12
#define HBD2_INB            PORTBbits.RB1
#define HBD2_INB_TRI        TRISBbits.TRISB1
#define HBD2_INB_ANS        ANSELHbits.ANS10
#define HBD2_EN             PORTBbits.RB2
#define HBD2_EN_TRI         TRISBbits.TRISB2
#define HBD2_EN_ANS         ANSELHbits.ANS8



void write_pot(unsigned int value, unsigned int chip_select);

void main()
{
   SPI_Initialize_Master();
    FET_Reg_Sense_ANS = 0;  //disable AN11 shared with RB5
    FET_Reg_Sense_TRI = 0;  //RB5 is a digital output
    FET_Reg_Sense_EN = 0;   //init the fet sense

    FET_Gate_TRI = 0;   //Enable RC0 as digital output
    FET_Gate_DIS = 0;   //init the gate disable

    XTAL1_ANS = 0;  //disable AN0 shared with RB5
    XTAL1_TRI = 0;  //Ra0 is a digital output
    XTAL1_EN = 0;   //init the xtal0
    XTAL2_ANS = 0;  //disable AN1 shared with RB5
    XTAL2_TRI = 0;  //RA1 is a digital output
    XTAL2_EN = 0;   //init the xtal1
```

```
    HBD1_INA_ANS = 0;
    HBD1_INA_TRI = 0;
    HBD1_INA     = 0;
    HBD1_INB_TRI = 0;
    HBD1_INB     = 0;
    HBD1_EN_TRI = 0;
    HBD1_EN      = 0;
    HBD2_INA_ANS = 0;
    HBD2_INA_TRI = 0;
    HBD2_INA     = 0;
    HBD2_INA_ANS = 0;
    HBD2_INB_TRI = 0;
    HBD2_INB     = 0;
    HBD2_EN_ANS = 0;
    HBD2_EN_TRI = 0;
    HBD2_EN      = 0;

    DAQ_port_init = 0b11111111; //initialize DOMI ports as inputs

    FET_CS_TRI = 0;
    FET_LDAC_TRI = 0;
    inamp_CS_TRI = 0;
    inamp_LDAC_TRI = 0;
    FET_CS = 1;
    inamp_CS = 1;
    FET_LDAC = 1;
    inamp_LDAC = 1;
    int temp = 0;
    int number = 0;
    int gate_test_count = 0;
    int do_gate = 0;
    int gate_lvls = 0;

   while(1)
   {
       if(Slot_EN == 1)//slot is on
       {
            //osc inverter for DUT
            if(XTAL_EN_INVRT == 1)
            {
```

```
        XTAL1_EN = 0;
        XTAL2_EN = 0;
    }
    else
    {
        XTAL1_EN = 1;
        XTAL2_EN = 1;
    }

    FET_LDAC = 0;
    if(FET_mode == 0)//SEE mode
    {
        FET_Reg_Sense_EN = 1;
        FET_Gate_DIS = 1;
        write_pot(0,0);
    }
    else //TID mode
    {
        //FET stuff
        FET_Reg_Sense_EN = 1; //turn on regular current sense
        FET_Gate_DIS = 0; //enable gate
        number = 6.25 /(2.5*0.000610500611); //set gate on
        write_pot(number,0);
        if(Gate_test_trigger == 1) //if gate test triggered, enable test
        {
            do_gate = 1;
        }
        if(do_gate == 1)
        {
            number = gate_lvls/(2.5*0.00610500611);
            write_pot(number,0);
            gate_lvls += 2;
            __delay_ms(1500);
            if (gate_lvls >= 64) //if gate test enabled then run through 26 steps 0 to 5 V
            {
                gate_lvls = 0;
                gate_test_count += 1;
                do_gate = 0; // turn off testing loop if not doing gate test
            }
        }
```

```
            //In amp stuff
            inamp_LDAC = 0;
            if(Inamp_HBD_state == 1)
            {
                number = 4 / 0.0012210012210012;
                write_pot(number,1);
            }
            else
            {
                number = 1 / 0.0012210012210012;
                write_pot(number,1);
            }

            //HBD Stuff
            HBD1_EN = 1;
            HBD2_EN = 1;
            if(Inamp_HBD_state == 1)
            {
                HBD1_INA = 1;
                HBD1_INB = 0;
                HBD2_INA = 1;
                HBD2_INB = 0;
            }
            else
            {
                HBD1_INA = 0;
                HBD1_INB = 1;
                HBD2_INA = 0;
                HBD2_INB = 1;
            }
        }
    }
    else //turn slot off
    {
        XTAL1_EN = 0;
        XTAL2_EN = 0;
        FET_Gate_DIS = 1;
        FET_Reg_Sense_EN = 0;
        write_pot(0,0);
        write_pot(0,1);
```

```
            __delay_us(100);
            FET_LDAC = 1;
            inamp_LDAC = 1;
        }
    }
}

void write_pot(unsigned int value, unsigned int chip_select)
{

    //Step Size = 2^n, Therefore 12bit 2^12 = 4096. 5V reference step will be 5/4095 = 0.0012210012210012V
    unsigned int container ;
    unsigned int MSB;
    unsigned int LSB;
    //Create container for value
    container = value;
    //Creating Dummy 8 bit. Dividing 256, upper 4 bits are captured in LSB
    LSB = 0000 1111*/
    LSB = container/256;
    //Sending the configuration with the 4 bit data. LSB = 0011 0000 OR 0000 1111. Result is 0011 1111
    LSB = (0x30) | LSB;
    //Container has the 21bit value. Extracting the lower 8 bits. Result is 1111 1111 which is MSB
    MSB = 0xFF & container;
    //Setup Chip selects. CS is low during data transmission.
    if (chip_select == 0){ FET_CS = 0; }
    else { inamp_CS = 0; }
    //Sending the 16bits data by dividing into two bytes
    SPI_Write(LSB);
    SPI_Write(MSB);
    //Turn off chip selects. CS is high.
    if (chip_select == 0) { FET_CS = 1; }
    else { inamp_CS = 1; }
}
```

LISTING B.6: PIC_Embedded_System.c

B.3 Digital Micro-controller Source Code

```
/*
Main function Source Code for Digital Embedded System
used on the radiation test bench
Written by Jesse Ward
 */
#include "atmel_start.h"
#include "usb_start.h"
#include <string.h>
#include "ext_mram.h"
#include <stdio.h>
#include "spi_nor_flash.h"

#if CONF_USBD_HS_SP
static uint8_t single_desc_bytes[] = {
    /* Device descriptors and Configuration descriptors list. */
    CDCD_ACM_HS_DESCES_LS_FS};
static uint8_t single_desc_bytes_hs[] = {
    /* Device descriptors and Configuration descriptors list. */
    CDCD_ACM_HS_DESCES_HS};
#define CDCD_ECHO_BUF_SIZ CONF_USB_CDCD_ACM_DATA_BULKIN_MAXPKSZ_HS
#else
static uint8_t single_desc_bytes[] = {
    /* Device descriptors and Configuration descriptors list. */
    CDCD_ACM_DESCES_LS_FS};
#define CDCD_ECHO_BUF_SIZ CONF_USB_CDCD_ACM_DATA_BULKIN_MAXPKSZ
#endif
static struct usbd_descriptors single_desc[]
    = {{single_desc_bytes, single_desc_bytes + sizeof(single_desc_bytes)}
#if CONF_USBD_HS_SP
        ,
        {single_desc_bytes_hs, single_desc_bytes_hs + sizeof(single_desc_bytes_hs)}
#endif
};
/** Buffers to receive and echo the communication bytes. */
static uint32_t usbd_cdc_buffer[CDCD_ECHO_BUF_SIZ / 4];
/** Ctrl endpoint buffer */
static uint8_t ctrl_buffer[64];
volatile bool cdcTransferRead = false;
volatile uint32_t cdcTransferReadLen;
```

```
volatile bool cdcTransferWrite = false;
volatile bool cdcConnected = false;
static bool cdcWriteDone(const uint8_t ep, const enum usb_xfer_code rc, const uint32_t count){
    cdcTransferWrite = false;
    return false;
}
static bool cdcReadDone(const uint8_t ep, const enum usb_xfer_code rc, const uint32_t count){
    cdcTransferReadLen = count;
    cdcTransferRead = false;
    return false;
}
static int32_t cdcRead(char* const buf, const uint16_t length){
    cdcTransferRead = true;
    if (cdcdf_acm_read((uint8_t*)buf, length) != USB_OK) {
        cdcTransferRead = false;
        return 0;
    }
    while(cdcTransferRead && cdcConnected);
    return (int32_t)cdcTransferReadLen;
}
static uint8_t outBuf[80];
static uint32_t outLen = 0;
static int32_t cdcWrite(char* buf, const uint16_t length){
    const char* end = buf + length;
    for (const char* p = buf; p < end && cdcConnected; ++p) {
        outBuf[outLen++] = *p;

        if (*p == '\n' || outLen==sizeof(outBuf)) {
            cdcTransferWrite = true;
            cdcdf_acm_write(outBuf, outLen);
            while(cdcTransferWrite && cdcConnected);
            outLen = 0;
        }
    }
    return length;
}

//brief Callback invoked when bulk OUT data received
static bool usb_device_cb_bulk_out(const uint8_t ep, const enum usb_xfer_code rc, const uint32_t count){
    cdcdf_acm_write((uint8_t *)usbd_cdc_buffer, count);
```

```
    /* No error. */
    return false;
}

//brief Callback invoked when bulk IN data received
static bool usb_device_cb_bulk_in(const uint8_t ep, const enum usb_xfer_code rc, const uint32_t count){
    /* Echo data. */
    cdcdf_acm_read((uint8_t *)usbd_cdc_buffer, sizeof(usbd_cdc_buffer));

    /* No error. */
    return false;
}

//brief Callback invoked when Line State Change
static bool usb_device_cb_state_c(usb_cdc_control_signal_t state){
    if (state.rs232.DTR) {
        /* Callbacks must be registered after endpoint allocation */
        //cdcdf_acm_register_callback(CDCDF_ACM_CB_READ, (FUNC_PTR)usb_device_cb_bulk_out);
        //cdcdf_acm_register_callback(CDCDF_ACM_CB_WRITE, (FUNC_PTR)usb_device_cb_bulk_in);
        ///* Start Rx */
        //cdcdf_acm_read((uint8_t *)usbd_cdc_buffer, sizeof(usbd_cdc_buffer));
        cdcdf_acm_register_callback(CDCDF_ACM_CB_READ, (FUNC_PTR)cdcReadDone);
        cdcdf_acm_register_callback(CDCDF_ACM_CB_WRITE, (FUNC_PTR)cdcWriteDone);
        cdcConnected = true;
    }
    else {
        cdcConnected = false;
    }
    /* No error. */
    return false;
}

//brief CDC ACM Init
void cdc_device_acm_init(void){
    /* usb stack init */
    usbdc_init(ctrl_buffer);
    /* usbdc_register_funcion inside */
    cdcdf_acm_init();
    usbdc_start(single_desc);
```

```
    usbdc_attach();
}

//Main Loop for digital system
void cdcd_acm_example(void){
    while (!cdcdf_acm_is_enabled()){
        // wait cdc acm to be installed
    };
    uint8_t wdata[128] = {0};//checkerboard pattern buffer
    uint8_t wdata2[128] = {0};//null pattern buffer
    int sslen;
    int cs;
    int i;
    int countlen = 0;
    bool dym_do = false;//dynamic test bool
    char static_ch0 = 'z';//command char for static start
    char static_ch1 = 's';//command char for static end
    char dym_ch = 'd';//command char for dynamic start
    char dym_fin = 'x';//command char for dynamic end
    char csout[4] = {0};
    char intout[6] = {0};
    for (i =0; i < 128; ++i){//fill the checkerboard buffer
        wdata[i] = 0x55;
    }
    cdcdf_acm_register_callback(CDCDF_ACM_CB_STATE_C, (FUNC_PTR)usb_device_cb_state_c);
    while (1){
        delay_ms(200);
        if (cdcConnected){
            while (cdcConnected){
                char input;
                if (cdcRead(&input,1) == 1){//initialize digital DUT setup
                    char incheck = input;
                    if (incheck == static_ch0){
                        for (cs = 0; cs < 4; ++cs){
                            countlen = 4096;
                            if (cs < 3){
                                for ( i = 0; i < countlen; ++i ){
                                    if (cs < 2){
                                        while(EXTMRAM_write(i*128,128,&wdata2[0],cs)==false);
                                        sprintf(intout,"%d",i);
```

```
                        sslen = sprintf(intout, "%d", i);
                        cdcWrite(&intout,sizeof(intout));
                        cdcWrite("\n",1);
                        delay_us(5);
                    }
                    while(EXTMRAM_write(i*128,128,&wdata[0],cs)== false);
                    sprintf(intout,"%d",i);
                    sslen = sprintf(intout, "%d", i);
                    cdcWrite(&intout,sizeof(intout));
                    cdcWrite("\n",1);
                    delay_us(5);
                    if ( i    == (countlen-1)){
                        cdcWrite("done\n",5);
                    }
                }
            }
            else{
                for (i = 0; i < countlen; ++i){
                    spi_nor_flash_write(SPI_NOR_FLASH_0,&wdata[0],(i+1025)*128,128);
                     delay_ms(2);
                    if ( i == (countlen-1))
                    {
                        cdcWrite("done\n",5);
                    }
                }
            }
        }
    }
    if (incheck == static_ch1){//do static test
        for (cs = 0; cs < 4; ++cs){
            countlen = 4096;
            if (cs < 3){
                for (i = 0; i < countlen; ++i){
                    uint8_t rdata[128] = {0};
                    while(EXTMRAM_read(i*128,128,&rdata[0],cs)==false);
                    sprintf(csout,"CS:%d",cs);
                    while(cdcWrite(&csout,4) == 0);
                    while(cdcWrite(&incheck,1) == 0);
                    while(cdcWrite("sector:",7) == 0);
                    sprintf(intout,"%d",i);
```

```
                    sslen = sprintf(intout, "%d", i);
                    while(cdcWrite(&intout,sslen) == 0);
                    while(cdcWrite("bytes:",6) == 0);
                    while(cdcWrite(&rdata[0],128) == 0);
                    while(cdcWrite("\n",1) == 0);
                    delay_ms(100);
                }
            }
            else{
                for (i = 0; i < countlen; ++i){
                    uint8_t rdata[128] = {0};
                    spi_nor_flash_read(SPI_NOR_FLASH_0,&rdata[0],(i+1025)*128,128);
                    delay_us(10);
                    sprintf(csout,"CS:%d",cs);
                    while(cdcWrite(&csout,4) == 0);
                    while(cdcWrite(&incheck,1) == 0);
                    while(cdcWrite("sector:",7) == 0);
                    sprintf(intout,"%d",i);
                    sslen = sprintf(intout, "%d", i);
                    while(cdcWrite(&intout,sslen) == 0);
                    while(cdcWrite("bytes:",6) == 0);
                    while(cdcWrite(&rdata[0],128) == 0);
                    while(cdcWrite("\n",1) == 0);
                    delay_ms(100);
                }
            }
        }
    }
    if (incheck == dym_ch){
        dym_do = true;
    }
    if (incheck == dym_fin){
        dym_do = false;
    }
    if(dym_do  == true){//do dynamic test
        while(1){
        for (cs = 0; cs < 4; ++cs){
            countlen = 4096;
            if (cs < 3){
                for ( i = 0; i < countlen; ++i ){
```

```
                    if (cs < 2){
                        while(EXTMRAM_write(i*128,128,&wdata2[0],cs)==false);
                        delay_us(5);
                    }
                    while(EXTMRAM_write(i*128,128,&wdata[0],cs)== false);
                    delay_us(5);
                }
            }
            else{
                for (i = 0; i < countlen; ++i){
                    spi_nor_flash_erase(SPI_NOR_FLASH_0,(i+1025)*128,128);
                    delay_us(10);
                    spi_nor_flash_write(SPI_NOR_FLASH_0,&wdata[0],i*128,128);
                    delay_us(5);
                }
            }
        }
        for (cs = 0; cs < 4; ++cs){
            countlen = 4096;
            if (cs < 3){
                for (i = 0; i < countlen; ++i){
                    uint8_t rdata[128] = {0};
                    while(EXTMRAM_read(i*128,128,&rdata[0],cs)==false);

                    sprintf(csout,"CS:%d",cs);
                    while(cdcWrite(&csout,4) == 0);
                    while(cdcWrite("dsector:",8) == 0);
                    sprintf(intout,"%d",i);
                    sslen = sprintf(intout, "%d", i);
                    while(cdcWrite(&intout,sslen) == 0);
                    while(cdcWrite("bytes:",6) == 0);
                    while(cdcWrite(&rdata[0],128) == 0);
                    while(cdcWrite("\n",1) == 0);
                    delay_ms(100);
                }
            }
            else{
                for (i = 0; i < countlen; ++i){
                    uint8_t rdata[128] = {0};
                    spi_nor_flash_read(SPI_NOR_FLASH_0,&rdata[0],(i+1025)*128,128);
```

```
                                    delay_us(10);
                                    sprintf(csout,"CS:%d",cs);
                                    while(cdcWrite(&csout,4) == 0);
                                    while(cdcWrite("dsector:",8) == 0);
                                    sprintf(intout,"%d",i);
                                    sslen = sprintf(intout, "%d", i);
                                    while(cdcWrite(&intout,sslen) == 0);
                                    while(cdcWrite("bytes:",6) == 0);
                                    while(cdcWrite(&rdata[0],128) == 0);
                                    while(cdcWrite("\n",1) == 0);
                                    delay_ms(100);
                                }
                            }
                        }

                        }
                    }
                }
            }
        }
    }
}
void usb_init(void)
{
    cdc_device_acm_init();
}
```

LISTING B.7: SAME70_Embedded_System.c

C Author Copyright Permission

This appendix provides the permission to re-use the contents of my previously published paper.

C.1 Nuclear & Space Radiation Effects Conference

From: Jesse Ward <wardj3@myumanitoba.ca>

Sent: Tuesday, November 29, 2022, 4:24 AM

To: Pascale Gouker <pgouker@ll.mit.edu>

Hello Pascale,

I'd like to request permission to re-use sections of my paper published in the 2022 NSREC radiation effects data workshop in my thesis document. Please let me know if this is okay to do. The paper is available here:

https://ieeexplore.ieee.org/document/9921622

Thanks for your time!

Regards, Jesse Ward

From: Pascale Gouker <pgouker@ll.mit.edu>

Sent: Tuesday, November 29, 2022, 9:41 AM

To: Jesse Ward <wardj3@myumanitoba.ca>

Hello Jesse,

This is not a problem to re-use the sections of your REDW paper as long as you make reference to it in your thesis manuscript.

Good luck with writing the thesis and let me know if you have any other questions about your paper.

Best,

Pascale