STUDY OF SINGLE EVENT UPSETS (SEUS) A SURVEY AND ANALYSIS

by

Sheldon Mark Foulds, B.ASc.,

Ghulam Ishaq Khan University, Topi, Swabi, Pakistan, 2009,

A project

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Engineering

in the Program of

Electrical and Computer Engineering.

Toronto, Ontario, Canada, 2013

AUTHOR'S DECLARATION

Abstract

Title: STUDY OF SINGLE EVENT UPSETS - A SURVEY AND ANALYSIS

Student Name: Sheldon Mark Foulds, B.ASc,

Ghulam Ishaq Khan University, Pakistan, 2009,

Program: Electrical and Computer Engineering

Name of University: Ryerson University

Over the last few years evolution in electronics technology has led to the shrinkage of electronic circuits. While this has led to the emergence of more powerful computing systems it has also caused a dramatic increase in the occurrence of soft errors and a steady climb in failure in time (FIT) rates. This problem is most prevalent in FPGA based systems which are highly susceptible to radiation induced errors. Depending upon the severity of the problem a number of methods exist to counter these effects including Triple Modular Redundancy (TMR), Error Control Coding (ECC), scrubbing systems etc. The following project presents a simulation of an FPGA based system that employs one of the popular error control code techniques called the Hamming Code. A resulting analysis shows that Hamming Code is able to mitigate the effects of single event upsets (SEUs) but suffers due to a number of limitations.

Keywords: Single Event Upsets (SEUs), Hamming Code, Radiation Induced Errors.

Acknowledgments

I would like to express my deep gratitude to my project supervisor Dr. Vadim Geurkov at Ryerson University for his sincere help and support and inspiring me to think creatively like a true engineer.

I wish to express my sincere gratitude to my course supervisor Dr. Fei Yuan at Ryerson University for his guidance and support over the course of my graduate studies.

I would like to appreciate my family who have sacrificed a lot to see me succeed to this level and finally to my best friend Zhiliang Xing for changing the way I think and live my life and my cousin Calvin for his MATLAB troubleshooting skills.

Table of Contents

List of Figures

Chapter 1

Introduction

In today’s world there is a high demand for reliable electronic systems in the aeronautics and marine industry. This need is especially evident in space programs which are mammoth undertakings and where the margin for error is very little. In such missions the need for reliable systems that are able to give precise and accurate results at all times cannot be stressed enough. One common aspect that affects the reliability of devices across these different sectors is radiation induced errors. In recent years radiation induced errors are becoming significant contributors to the FIT (Where FIT is the Failure-In Time, that is the number of errors in 10^9 hours) rate of electronic devices not only in space where they are most prominent but also recently at sea level.

This problem can be traced back to 1975 when Binder el [1] first spoke about anomalies occurring in space electronics due to the bombardment of on board flips flops by cosmic radiation particles. Binder attributed this to the ionization tracks of electron hole pairs that charged the base emitter capacitors of critical transistors. Soon as the capacitor acquired the right amount of charge it would cause the flip flop to change its value giving erroneous results. At the time analysis revealed that the relative failure rate was small (approximately one error in 4 years) and the particles causing these errors were 100 MeV Fe Atoms. Since these atoms are unable to penetrate the earth’s atmosphere the problem was considered as confined to space.

In 1978 however May and Woods of Intel provided the first evidence of electronics being affected at sea level by radiation from decaying Uranium and Thorium impurities in packaging material [2]. While not directly related to cosmic radiation in nature, it was to prove pivotal as it demonstrated the effects of radiation on systems at sea level. It was also the first time the term

“Soft Error” was used to describe errors induced by radiation. The paper by May and Woods went onto became famous across the semiconductor industry and kick started a trend in research of radiation induced errors at sea level. Today the terms soft errors and hard errors are commonly used in the field of digital system testing.

In the same year (1978) Ziegler of IBM [3] showed that similar to alpha particles, cosmic radiation could also contribute towards soft error rates. Ziegler proved that most cosmic radiation was blocked out in space due to the earth’s atmosphere and only particles with energies of 2 GeV or higher are able to actually to penetrate through. Once in the atmosphere these primary particles tend to collide with other particles present causing a cascade of nuclear reactions. It is only the sixth generation of these particles that actually manages to get through to devices at sea level. While the probability of the occurrence of such an event is extremely low; the possibility of its existence is very real and this in turn demands that precautions be taken to ensure the reliability of critical systems. The following figure demonstrates how primary particles lead to formation of secondary particles in the Earth’s atmosphere

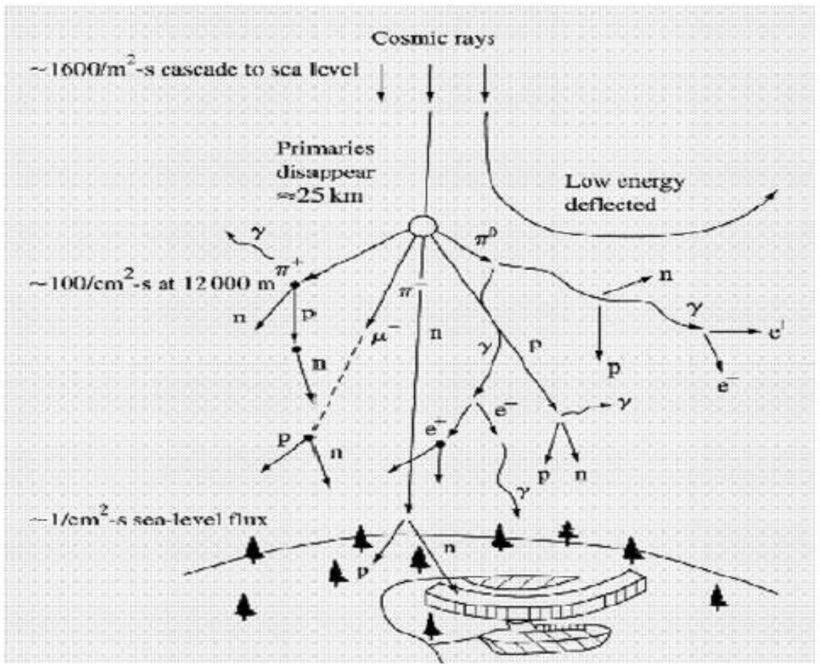


FIG 1.1: Cosmic Radiation causing cascade of particles [4]

From the above mentioned facts it is concluded that in terrestrial applications the two main sources of radiation induced errors are:

Neutrons: Generated when cosmic radiation interacts in the earth's atmosphere with other particles, causing the release of numerous secondary particles.[18]

Alpha Particles: Emitted by radioactive impurities that are present in low concentration in packaging, bonding and die materials.

These two sources of radiation work in different mechanisms. Alpha particles are highly charged in nature, thus when they penetrate into the ICs (integrated circuits) they are able to produce a number of electron hole pairs that may cause a bit flip if the charge is above the critical charge value of the transistor. Neutrons on the other hand being neutral are unable to cause any harm on their own. They instead interact with atoms present in the substrate material like Silicon and Oxygen. Collisions with these atoms cause the formation of numerous secondary charged particles that allow the same bit flip to occur. Neutrons colliding with atoms in the substrate material of transistors also tend to cause numerous material defects that may alter the behavior of a transistor entirely, thus they have a highly probability to damage electronic systems. When a sensitive node (for example the drain of a transistor) is in the proximity of the ionization track of these electrically charged particles, it collects a significant part of carriers (holes or electrons), resulting in a transient current pulse on this node. The effect of this pulse depends on the type of the cell to which this node belongs. If it is a storage cell such as those found in memory modules such as DRAM's and SRAMs then the outcome is different and if it is a simple logic cell such as those found on gates in logic components then the effect is completely different. Thus ensuring a reliable design demands that each section of the circuit be separately considered and these charged particles affect different parts of the circuit is the main focus of this research project.

The problem with investigating the effect of SEU's in electronic circuits is that the field covered by such a statement covers a vast domain. Electronic circuits tend to come in all kinds of shapes, forms and sizes ranging from simple combinational circuits to advanced sequential circuits with many memory elements. Memory elements form the majority portion of modern day electronic systems and prove to be the most vulnerable part of the circuit as well. Protection of memory modules in electronic systems is one of the foremost categories in digital system testing and design today, especially since FPGAs have been gaining prominence in recent years. Field Programmable Gate Arrays (FPGAs) are reconfigurable platforms on which electronic designs can be implemented via the use of design files stored in the SRAM memory cells on board. They are increasingly being used on space missions [15] and aeronautical applications where there is a need to conserve space and designers opt for platforms that can be reconfigured to implement different circuits on the same platform at different points in time. Thus the investigation of how radiation induced errors affect FPGA's is important.

Logic circuits on the other hand have a completely different mechanism when it comes to radiation induced errors. They are effectively two parts of a logic circuit, Sequential elements and Combinational elements. When radiation induced errors in logic circuits are observed; the majority of them tend to originate from the sequential parts of the logic circuit which consist of latches and flip flops. In fact the first soft error ever detected, which was reported by Binder, occurred in flip flops in space. Sequential elements, such as latches and flip-flops, are used in digital logic circuits for temporary data storage. Unlike SRAM and DRAM cells however all latches are not symmetric in nature and the charges stored by them may have different meanings as judged by the system.

Combinational parts of the circuit themselves are not directly responsible for the occurrence of soft errors. In fact combinational circuits actually work to counter the effects of soft errors and dramatically decrease the probability of their occurrence due to various masking effects which are described later in section V.

What’s disturbing is that today the electronic industry is entering a dangerous trend that tends to rapidly increase the probability of occurrence of radiation induced errors in electronic systems rather than trying to counter them. Three major trends that dominate the electronic industry today are

1. The shrinkage of transistor sizes to incorporate larger number of circuit elements onto single platforms; this is especially visible in FPGA systems.
2. The use of faster processors with higher frequencies.
3. The use of systems that have lower power consumption.

These trends in the industry have continued for a long time, typically driven by the demand in household computing systems that do not require protection from SEU. However when it comes to Military and Commercial Aviation systems or Spacecrafts that are constantly being exposed to radiation at high altitudes these trends make the system on board much more susceptible to radiation.

The shrinkage in transistor sizes has two effects on electronic systems. One it reduces the effective area of the circuit thus reducing the probability that it will be struck by incoming radiation particles. However this reduction is offset by the fact that miniaturization of the circuit is leading to a large densification of circuit elements. For example a smaller feature size means a larger number of transistors can be packed onto a single FPGA platform, thus when a radiation particle tends to strike FPGA boards they tend to cause a number of radiation induced errors. This is one of the main reasons why the study of SEUs is given less importance and there is a greater effort made towards the understanding of multiple bit errors (Popularly recognized as Multiple Bit Upsets or MBUs) which are more likely to occur as device feature sizes continue to shrink.

The use of faster frequency processors is also another concern for electronic systems. A higher frequency means that the system is utilizing a clock with a very short period. While this allows the system to work faster it also allows radiation induced errors particularly soft errors to penetrate easily into the system. When radiation particles collide with the elements in the electronic circuit they tend to produce a small voltage spike which is visible for only a short period of time. This voltage spikes value was easily overwritten in time by systems in the past. Now however because the frequency of clocks being used is much higher it ensures that the voltage spike occurring can be visible for a good portion of the clock thus increasing the probability that it could be captured by a memory element when it is sampling its input lines. Thus this trend contributes to the occurrence of soft errors in electronic systems that are synchronous in nature.

Finally the use of components with smaller power consumption is another trend that’s making radiation induced errors to occur more frequently. In order for radiation induced errors to cause errors in the system they must be able to overturn the values stored by transistors in the memory cell, for this they require to have sufficient energy to overcome the energy that is possessed by the transistor. In the past components were controlled by much greater power values that ensured they could not easily be disturbed by radiation induced errors. Now however electronic systems are more easily disturbed due to the shrinkage of components sizes that consume much less power.

It should be noted here that roughly 90% of SEUs passing through FPGA platforms bear no effect on the output of the system. Still their existence cannot be denied and their prevention is necessary.

1.1 Motivation

The motivation of this paper comes from the fact that there is growing evidence that the effects of radiation induced errors are becoming more and more prominent at sea level. With growing shrinkage in device size this trend is expected to continue. Thus there is a need investigate and develop methods that counter the effects of radiation induced errors but also tools that would make it easier to understand and further investigate how radiation induced errors affect FPGA based electronic systems. The objective of this paper was to conduct a study of the effect of single event upsets and how Hamming Codes (which is used to detect and correct single bit errors) can be used to counter their effects. This was done by designing a simulator that works as an isolated electronic system and uses Hamming codes to protect data that is being transmitted between an encoder and decoder circuit. Simulated SEUs are generated to disrupt different parts of the systems and observe how the system reacts under various conditions.

The contribution of this paper is as follows.

1. The development of a unique SEU simulator that employs Hamming code to protect data being transmitted between an encoder and decoder. The radiation induced SEUs are generated electronically to mirror the effect of bit flips in the system.

2. The study and analysis of how SEUs can affect different parts of electronic systems via simulations is conducted using the Hamming Code simulator developed in conjunction with this project. In particular the parts of the circuit that are studied are

    a. Data transmission lines and Data Registers

    b. Synchronizations lines

    c. Clock lines

3. The robustness of the Hamming Code is investigated with successive SEUs while a 16 second audio file in WAV format is transmitted in binary 8-bit words between the encoder and decoder in the presence of continuous SEUs in different bits of the codeword.

4. The limitations of the Hamming Code are also investigated with the simulation of a multiple bit upset (MBU) event and observing the behavior of the system in the event of such an occurrence.

The primary goal of the design project was to develop a system that can constantly correct SEUs on data transmission lines and in data registers as long as they occur within the limits of its processing capabilities i.e SEUs should not occur after the Hamming Code has checked certain data and used it to process a decoded word as this would defeat the purpose of the system. Further assumptions associated with the system are presented in section V along with simulation results.

The Hamming Code simulator designed for this project was completely designed in VHDL language using the XILINX ISE design Suite 14.1. To facilitate with the conversions of WAV format files into unsigned integer 8-bit format and back again MATLAB R2013a Student Version was used.

Chapter 2

Background

Radiation induced faults are commonly known as single event effects (SEE) have been recognized as a design issue for avionics systems as well as high reliability ground-based systems over the past few years. SEE disturbances are primarily caused when a single particle of radiation passes through an electronic device and deposits energy in sensitive areas of the device. While the majority of the radiation particles passing through the device have no effect, some of these particles that collide with lines in the circuit or logic or memory elements tend to produce a sudden spike in voltage. This spike known as a Single Event Transient (SET) value tends to propagate through the system before eventually dying down slowly. Usually this is not a problem in combinational circuits because it causes a momentary glitch and in instantly overwritten by data being followed. However if this value propagates forward onto a memory element and gets stored it can cause corruption of data in memory elements. This incorrect value stored can either lead to corruption of data leading to erroneous output or have catastrophic effects on system integrity (especially in FPGAs) depending on where it strikes. This value storage of an SET is defined as a Single Event Upset (SEU).

*A Single Event Upset (SEU) is defined by NASA as "radiation induced errors in microelectronic circuits caused when charged particles(usually from radiation belts or from cosmic rays) lose energy by ionizing the medium through which they pass, leaving behind a wake of electron-hole pairs"**[5]**. SEUs primarily occur in analog, digital and optical devices. They tend to be less destructive in nature and usually lead to minor corruption of data that can be ignored altogether or corrected easily. [5]*

Field programmable gate arrays (FPGA) are one of these electronic devices that tend to be most susceptible to SEUs. The presence of a large number of memory cells increases the probability of a registration of a faulty value in time.

The effects of radiation induced faults can be classified into three primary categories [6]

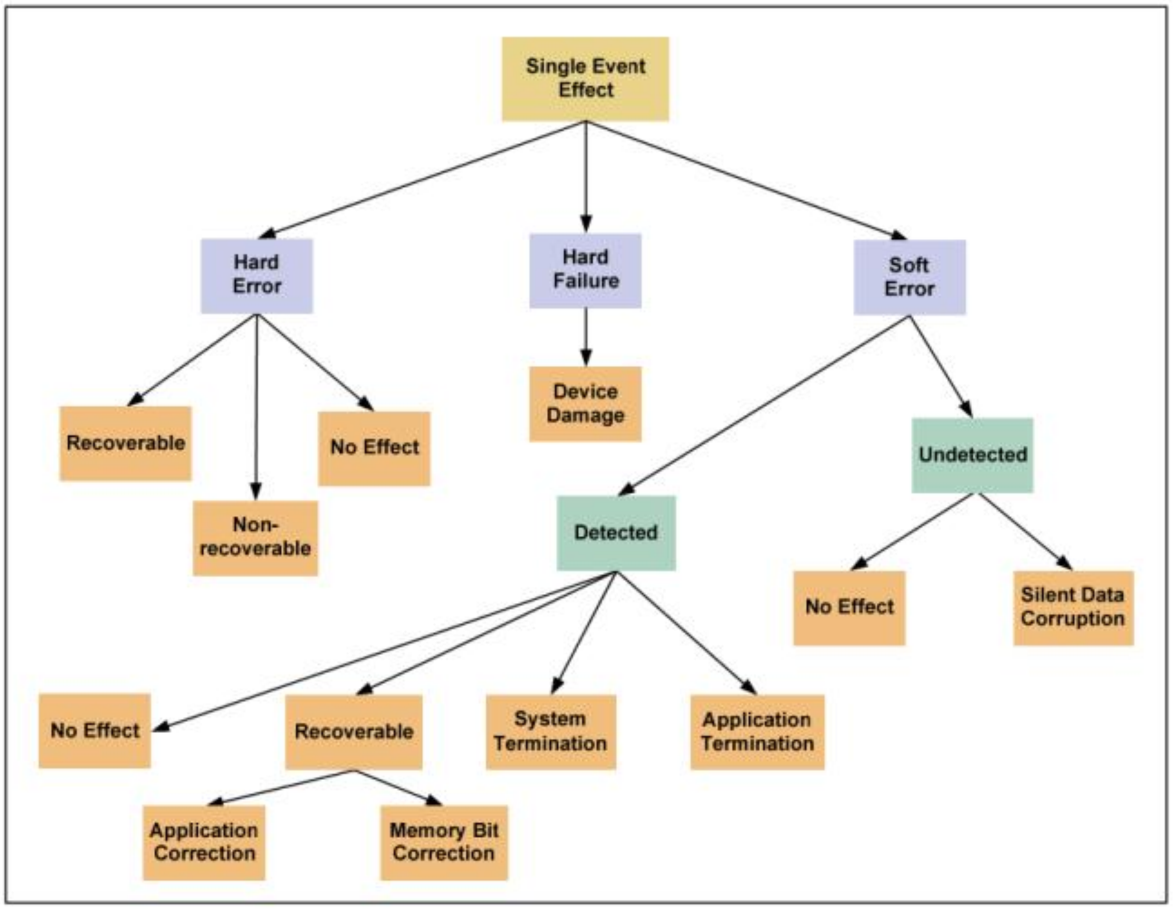


Fig 2.1: System Impact of Singe Event Effects [6]

Hard errors are errors that are not recoverable by a software reset and often require a complete shutdown of power and reset of the component to recover it for its operation. An example of this is when the configuration registers of an FPGA are faced with corruption, while this error does not physically or permanently damage the device; the system implemented by the configuration corrupted is faulty and may lead to a number of errors. The only way therefore to restore correct functionality is to reset the content of each configuration register and halt all operations until this is done.

In contrast a soft error is one in which the damage to the device is not permanent. Only a temporary change of state occurs in part of the device which can easily be recovered by resetting the component completely or simply applying correction codes to the data that was disrupted. Soft errors are therefore defined as non destructive and recoverable. Soft errors can be further divided into 3 categories. SETs defined earlier are temporary values that tend to propagate through the system; they usually are unable to cause any major disruption in the system unless they affect important synchronization signals like the clocking lines and represent momentary glitches that are overwritten in time. The second class of soft error known as SEU occurs when an SET arrives at a memory cell at a time when it is sampling input values. This causes the memory cell to accept faulty values as input, SEUs may also occur when radiation upsets data already stored in memory cells. The definition of an SEU is restricted in this paper as being able to only disrupt of a single bit of data. The final class of soft error is defined as the multiple bit upset or MBU. MBUs tend to occur when a burst of radiation hits a sensitive portion of the circuit affecting either multiple lines of data or multiple memory cells simultaneously causing large disruption in data words. In essence MBUs tend to be a collection of SEUs. Extensive information on how SEUs occur in Xilinx FPGAs can be found in [13][14]

The soft-error rate (SER) in devices is measured in FIT units, where 1 FIT is equal to one failure per billion device hours (i.e., one failure per 114,077 years). Typical SER values for electronic systems range between a few100 and about 100,000 FIT (i.e., roughly one soft error per year)[4]. Extensive product monitoring in the industry has shown that typical hard-error failure rates due to physical damage do not exceed values of more than 10 FIT[4] and typical values are usually lower. While the SER of 1Mbit of an SRAM cell are typically in the order of 1000 FIT[4]. SRAM cells happen to be the one of the most vulnerable parts of modern day electronic circuits and are usually the key component inside SRAM based FPGAs. While the error rate of 1Mbit of an SRAM cell is 1000 FIT, a larger collection of cells may have a much larger combined FIT rate.

Finally the last class of radiation induced error is termed as hardware failure which usually results in damage to the hardware and is the most extreme form of radiation induced errors that may occur. Not only does the effect causes a loss of function but requires that the device be repaired or replaced physically. An example of this is known as the Single Event Latch up effect or SEL. SELs occur when the capacitors in electronic systems are forced from a high impedance state to a low impendence state due to passing radiation particles. This causes a electric surge to pass through the system damaging sensitive components. More information on SELs can be found here [4]. It is interesting to note here that in the past few years the effect of SELs have disappeared from the electronic circuits due to the use of smaller voltages in the circuit design process. Fig 2.1 summarizes the effect of radiation induced faults. More knowledge on hard errors and hardware failures can be found in [6]

Since the primary focus of this research paper is SEUs in FPGAS, The study presented in this paper will focus on soft errors which are a more regularly occurring phenomenon in FPGAs but will restrict the discussion to only diagnosing and solving single bit errors. Soft errors are one of the most challenging errors to detect in the electronics industry because the value spike from a radiation induced error occurs only momentarily; once this value propagates to a memory element and is stored it virtually becomes part of the system making it untraceable. Only once erroneous results are detected at the output of the system can the fault be detected and by this time it’s too late because the fault may have accumulated over a multiple number of outputs requiring a complete recycling of all data in the system. Design engineers today are therefore striving to introduce new methodologies in design processes so they can counter these influence of these soft errors as soon as they occur and ensure system reliability overall.

Depending on where the soft error occurs there can be different effects on the system. If the data register or data line is affected it can lead to erroneous results in processing or storage of data. More serious problems however tend to occur in FPGA's when a configuration register is hit by an SEU. Configuration registers are extremely critical to an FPGA's ability to reconfigure itself and maintain its current configuration. Modern day real-time systems rely on the ability of the FPGA to partially reconfigure itself and implement process sensitive soft cores that allow faster execution of data. If one of the configuration registers is affected it leaves the FPGA unable to correctly implement the architecture required or permanently alters the functionality of the device already mapped. This type of error is harder to fix and requires the control register to be erased and rewritten to completely remove errors.

Figure 2.2 shown below explains the options available to a system upon the detection of an SEU. If the system does not have error detection or correction capabilities then the SEU propagates through the system unnoticed. If it affects the output result from our system it is known as silent data corruption (SDC). Error detection techniques such as parity comparison can be used to detect this error and provide a stop mechanism to avoid system corruption, the hamming code system employed by the project presented in section IV is an example of such a method. However detection alone is not sufficient in many cases. Errors that are detected but not corrected are defined as detected unrecoverable errors (DUE). Currently the industry specifies soft error rates on devices in terms of DUE and SDC.

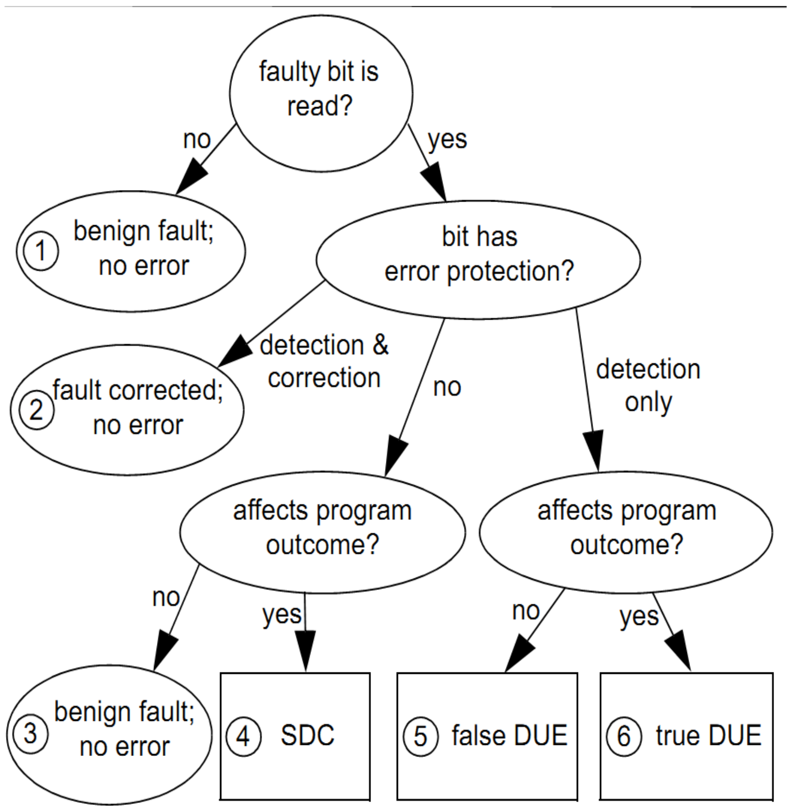


Fig 2.2: Classification of outcomes of a faulty bit [11]

Therefore in order to ensure reliability and reduce system fail rates it is important to have a combination of both detection and mitigation techniques. Mitigation techniques introduced in the next section have different ways of dealing with these soft errors. Some choose to mask the error through redundant hardware and work around them, while some tend to address them quickly and correct them to avoid future disruptions, other techniques provide simple detection that provide are able to immediately detect SEUs and can be coupled with scrubbing techniques to ensure the system data does not fall into silent corruption or lead to system failure, thus ensuring system reliability.

Chapter 3

Survey

With the shrinkage and scaling down of process technologies there has been a rapid rise in demand for systems that can demonstrate a higher level reliability than normal systems. The concept here is that the occurrence of a radiation induced error is completely based on probability. Whenever a system has to be designed; its design specifications must specify the acceptable level of tolerance for radiation induced errors. It is then up to the designers to simulate the behavior of the circuit in its intended environment of operation under lab tests. This is where they can determine exactly what level of mitigation is required and on which parts of the circuit. Generally implementing mitigation techniques over the entire circuit is impossible because each has its associated costs in processing time and space required. For example Triple Modular Redundancy (TMR) [7] attempts to replicate the module it is aiming to protect three times. Thus if TMR was to be used for an entire system such as the guidance system modules of a space craft it would consume a significant amount of area on board the space shuttle. Worse yet if no SBUs or MBUs occur the two extra systems are essentially redundant, consuming only extra power and taking space that could otherwise be provided to more essential systems. Thus in design of such systems only certain modules are chosen to be protected by mitigation techniques, the goal being ultimately to reduce the FIT rate of the device to a point where it is within acceptable standards.

Modern mitigation techniques come in various forms and can be employed at the component level or the system level. Component level mitigation techniques are usually measures implemented on individual components at the highest (deepest level) in a bid to reduce the overall FIT rate of overall systems. This may include improvements in the design process such as transistor sizing that provides radiation hardening [8]. In contrast system level mitigation techniques attempt to overlook individual parts of the system and implement mitigation techniques over the entire system; common system level mitigation techniques include scrubbing, voting, redundancy etc. It should be noted here that besides mitigation and correction techniques radiation hardening can also be achieved by use of special materials. For example Anti-Fuse FPGAs [16][17], These materials are able to counter the effects of radiation quite well but are expensive to manufacture, thus there is a need for cheaper alternatives that can be implemented faster on board electronic systems. Further overview of the different mitigation level techniques can be found here [6]. In the following section a number of popular mitigation techniques are presented briefly.

3.1 TMR (Triple Module Redundancy)

One of the basic techniques that have been proposed by researchers to counter the influence of soft errors is Triple module redundancy (TMR). While not exclusively used just for soft errors, TMR is often used in combination with other methods to provide a promising solution to avoid (not correct) soft errors caused by bit flips, especially in FPGA devices.

In its most basic form; The TMR attempts to avoid system errors by masking their existence through the use of additional redundant circuitry. In TMR, each system module along with every associated input and output line is replicated three times. Each model works exactly at the same time as the other. At the end of each module is a “detect and decide” mechanism that evaluates the outputs received from all three lines. Labeled as the “majority voter” this module is responsible for assessing the outputs of all three lines and determining which output will be forwarded to the primary output of the system. Thus two or more modules output the same answer then that answer is considered as the correct one and forwarded to the output. The idea here is that the SEU will at most cause a disruption in one processing line at a time; thus by conducting three separate redundant instantiations of the same operation the correct result from two of the modules is still attainable.

The idea sounds primitive and while the concept of implementing each system three times in full may not sound feasible it is actually quite reasonable on FPGA system if their new emerging ability to reconfigure parts of themselves is considered. Recent releases by Xilinx based FPGAs allow users to reconfigure parts of the FPGA at different points in time thus removing constraints to have a standard configuration at all times. Thus if a particular process running on an FPGA has five different sets of calculations to perform, all modules do not need to be configured onto the FPGA at the same time. Only modules that are needed at a particular moment in time maybe loaded and that too in triplicates, thus instituting the TMR mitigation system without having to take up the extra space. TMR is important for ensuring the reliability of modern day electronic systems, however its biggest weakness is that it is only a mitigation system and is unable to diagnose or correct radiation induced errors. Thus to make the system more robust and resistant to errors it is necessary to use TMR techniques in conjunction with other mitigation and correction techniques like scrubbing and ECC.

The TMR technique is not without fault though. As is common with many mitigation techniques TMR suffers from reliability issues inside itself as it too is part of the same circuit being exposed to radiation. Thus if an error was to occur in the majority voter module that judges the outputs of the two redundant lines and the original lines as well there is a high probability that an incorrect result would be generated. A number of solutions to improve the ability of the TMR design have been suggested recently including the use of minority voters in between circuit modules to monitor changes inside the redundant circuits rather than simply at the end of the circuit. Other ideas include the separation of the redundant modules into different columns of the FPGA so to have maximum separation, thus if the FPGA was to fall victim to localized radiation beams; only part of the circuit would be affected and would have a lower probability of affecting all three modules at the same time. Further reading regarding the improvements to the TMR system in the form of T-TMR maybe found here [9].

3.2 Scrubbing

Scrubbing is a vital tool that is usually used in conjunction with TMR or error control correction mechanisms to continually monitor the state of configuration registers on FPGAs. When FPGA systems are considered for protection against radiation induced errors and specifically SEUs they are two main types of information that must be considered in the systems. One is the regular that data being passed between modules inside the system currently implemented on the FPGA and one is the control data that determines how the FPGA is currently configured. The protection of control data is extremely important for the functioning of the FPGA. For example, consider that there is an electronic system currently implemented on board the FPGA platform along with TMR. This system can function successfully as long as the TMR configuration remains intact. However, if a radiation induced error was to occur in the configuration memory of the FPGA it would cause an unpredictable change in the systems configuration, thus disrupting the function of the entire system. The only way to fix such an error would be to rewrite the configuration of the FPGA so a fresh copy of the system can be re-implemented. This kind of error in the configuration memory is referred to as a Single Event Functional Interrupt or SEFI. To solve this problem the scrubbing mechanism was created to be used for protecting the configuration bit stream of FPGA systems. The scrubbing mechanism in its basic form requires three components to work correctly, a detection mechanism to detect SEUs (In either configuration bits or data bits in the modules), temporary fault tolerance to SEUs (E.g via a TRM Module) and finally a correcting mechanism to fix detected errors. The mechanism depends on keeping a fixed “Golden” copy of the systems configuration on a radiation hardened external memory device. The scrubbing mechanism then uses this golden copy of the data to rewrite the entire configuration of the FPGA based on what sort of solution method is being implemented. They are two basic kinds of scrubbing techniques utilized in the industry.

Blind scrubbing

Blind scrubbing [10] continuously writes valid configuration data back to the FPGA over existing data periodically. This process continues whether an upset is present or not, so even if an error is not present the current copy of the system is overwritten completely after specific interval of time. Since the golden copy of the system is essential here designers often retain the copy of this circuit not on the FPGA itself but on an external memory which is radiation hardened. This introduces extra hardware components such as external controllers and memory devices that add a small bit of complexity to the system. However this can be justified if the reliability of the circuit can be ensured. There is a major flaw in the blind scrubbing system. Since the system only rewrites that data of the configuration registers after a certain period of time; if an SEU was to occur in one of the configuration bit streams just after a rewrite, then the blind scrubber would never know and errors would propagate through the entire system without any detection. The only correction that would occur would be on the next blind scrub. This makes blind scrubbing less optimal for use in safety critical systems.

Read Back Scrubbing

The second kind of scrubbing technique used is read back scrubbing [10], this method involves constant monitoring of the configuration bit stream and its comparison to data stored in the golden copy of the system. If an error is detected at any time between the two copies; a rewrite of the configuration registers takes place. Since this requires constant reading of configuration and data bit streams, read back scrubbers tend to be harder to implement. Also the configuration bit stream of FPGAs tend to be enormous in size and allows errors to go unchecked for a short window of time while each control bit is read back and compared to the configuration bits in the golden copy. Despite this draw back Read Back Scrubbers tend to be much safer than blind scrubbers since they rewrite portions of the system only when an error is detected. Also if SEUs errors can be detected in specific control bits it may not be necessary to rewrite the configuration of the entire system like blind scrubbers do. If dynamic FPGAs are considered whose configuration can be changed during run time then if we know which control bits are affected by errors we can easily locate the affected module and call for a reconfiguration of that specific module instead of the whole system. This ensures system productivity and reliability even in the face of radiation induced errors.

It should be noted that for FPGA based systems scrubbing mechanisms are extremely essential. The fact that FPGAs are composed of SRAM based memory cell that are highly susceptible to radiation makes a technique like scrubbing invaluable to the correct operation of the system. Some designers prefer the use of TMR system in conjunction with simple scrubbing mechanisms to ensure that the system implemented on FPGA systems is reliable. In such cases read back or blind scrubbing is completely unnecessary. Only when the voter mechanism detects a constant error from a single source of the TMR, it can call for reconfiguration of the control bit stream of that particular module. This avoids the complexity of having to constantly monitor all configuration bits at all times and also does away with the inefficiency of blindly scrubbing the configuration registers.

3.3 Hamming Codes:

Hamming code belongs to a very important class of codes known as Error Control Codes ECC. Originally designed for the intention of protecting data from corruption during transmission between emitter and receiver ECC's like the Hamming code are gaining popularity in the field of system reliability especially when it comes to protection from SEUs and MBUs in memory cells in FPGAs and data transmission as well. ECC's are generally based on the concept of parity check bits which have been in use for a long time. Consider the “even parity check” ; If k bits of data are to be transmitted from one point in the system to another then the emitter adds a single bit to the sequence called the parity bit. What the even parity check encoder does is that it simply counts the number of 1’s in the k-bit data word and if there happens to be an even number of 1’s in the word the parity bit is set to 1, otherwise it is set to 0 to indicate an odd number of 1s. This proves valuable in indicating the occurrence of SEU’s in the system. ECC’s follow a similar concept, encoding data bits with redundant bits to form code words that can not only be used to indicate the occurrence of SEU’s and MBU’s but in many cases correct them as well. They are a number of different ECC’s that exist but the project presented in the next section of this paper makes use of the popular (12, 8) hamming code which is what this paper will focus on. The description of the Hamming code concept here is extended to provide more detail so it is easier to understand the makeup of the system described in the next section.

The (12, 8) hamming code is based on the concept that if a specific number of data bits must be protected from SEUs then they can be encoded with a fixed number of redundant bits that will allow the receiver system to decode where the error has occurred in the data. The number of check bits that would be required to encode a certain number of bits is given by the following formula

$$\mathbf{r \geq \log_2 (k + r + 1)} \tag{1}$$

Where **r** represents the minimum number of check bits that are required to encode **k** data bits. The Hamming code system designed for this project makes use of 8 data bits, thus by the formula given 4 check bits would be sufficient to correctly identify and correct all singe bit errors that may occur. Before data can be communicated (or corrected in case a faulty word is detected) from one module to another a code word needs to be generated that consists of both data bits and check bits arranged in a particular order. In this word there are 8 information or data bits identified as $\mathbf{D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0}$ and 4 check bits identified as $\mathbf{C_3, C_2, C_1}$ and $\mathbf{C_0}$. This will give us a code word of 12 bits in total identified as $\mathbf{B_{12}, B_{11}, B_{10}, B_9, B_8, B_7, B_6, B_5, B_4, B_3, B_2, B_1, B_0}$. The arrangement of data bits and check bits is as follows

$$\mathbf{B_{12}, B_{11}, B_{10}, B_9, B_8, B_7, B_6, B_5, B_4, B_3, B_2, B_1, B_0}$$

$$\mathbf{D_7, D_6, D_5, D_4, C_3, D_3, D_2, D_1, C_2, D_0, C_1, C_0}$$

The check bits are arranged inside the code word in such a way that they always occupy positions that are powers of two. From the distribution shown above it can be seen that check bit C_0 occupies position $B_2{}^0$, C_1 occupies position $B_2{}^1$, C_2 occupies position $B_2{}^2$ and finally C_3 occupies position $B_2{}^3$. With the positions in the word marked, the next step is to calculate the check bits which are embedded into the code words. The check bits are calculated using the following formula. The arrangement of bit positions is explained further in the section.

$$B_{2^i} = C_i = \sum_{j=1, j\neq 2^q}^{k+r} a_{i,j} B_j, \quad \forall i \in \{0, 1, \ldots, r-1\}, \qquad (2)\ [4]$$

Where j represents the position of the data bits inside the code word, it can take on any values except powers of 2 as these positions are occupied by check bits that are being determined. a_i represents a co-efficient that can take on binary values 0 or 1 and is responsible for activating the data bits associated with a particular check bit. These activated data bits are then Modulo 2 (XOR)ed together to give us the check bit value. For example if C_3 needs to be calculated then the formula translates to

$$\mathbf{C_3 = B_{12}.1 \oplus B_{11}.1 \oplus B_{10}.1 \oplus B_9.1 \oplus B_7.0 \oplus B_6.0 \oplus B_5.0 \oplus B_3.0}$$

It can be noticed from the above formula that because $C_3 = B_{2^3}$ and the binary representations of 12, 11, 10 and 9 contain the 2^3 they are included in the representation of the term C_3 while other terms 7, 6, 5 and 3 do not and so they are excluded. Using this clue a set of formulas can be constituted for computing the formulas of the check bits

$$\begin{aligned}
B_1 &= C_0 = B_3 \oplus B_5 \oplus B_7 \oplus B_9 \oplus B_{11} = D_0 \oplus D_1 \oplus D_3 \oplus D_4 \oplus D_6,\\
B_2 &= C_1 = B_3 \oplus B_6 \oplus B_7 \oplus B_{10} \oplus B_{11} = D_0 \oplus D_2 \oplus D_3 \oplus D_5 \oplus D_6,\\
B_4 &= C_2 = B_5 \oplus B_6 \oplus B_7 \oplus B_{12} = D_1 \oplus D_2 \oplus D_3 \oplus D_7,\\
B_8 &= C_3 = B_9 \oplus B_{10} \oplus B_{11} \oplus B_{12} = D_4 \oplus D_5 \oplus D_6 \oplus D_7.
\end{aligned} \qquad (3)\ [4]$$

With the check bits computed and the code word ready the system is now able to transmit the data. Once the data has been transmitted from one module to another it must determined if any errors occurred in the data during transmission for this purpose the syndrome must be calculated. A block diagram explaining the syndrome calculation process is show below.

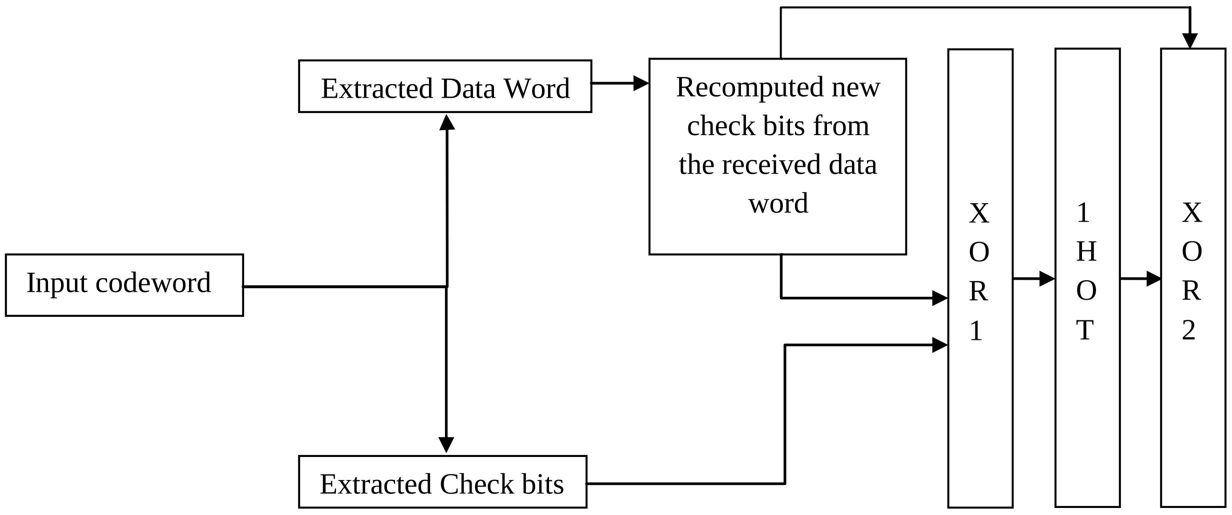


Fig 3.1: Overview of decoding process

At the end of transmission, the received word is split up into its constituent bits. The data bits are stored in one register and the check bits received are stored in a separate register. From the data bits received a new set of check bits are recomputed. These recomputed check bits are bit wise XORed with the check bits from the received word in the block labeled XOR1. If the received code word is labeled as B_{12}, B_{11}, B_{10}, B_9…..B_1 and the recomputed code words as B_{12}', B_{11}', B_{10}', B_9'…….B_1'. Then using equation 2 the syndrome can be described as

$$S_i = B'_{2^i} \oplus \sum_{j=1, j\neq 2^q}^{k+r} a_{i,j}B'_j = \sum_{j=1}^{k+r} a_{i,j}B'_j, \quad \forall i \in \{0, 1, \dots, r-1\}. \qquad (4)[4]$$

This process generates an r bit syndrome which indicates the position of any faulty bit in the received word. Now if there is no error in the received word then B_j = B_j' for each bit indicating an error free word. However if a single error exists in the received word then code word B_j is not equal to B_j' at every bit except the bit where the bit flip has occurred. At this position the value of the bit is B_m = B_m'$\oplus$ 1.Thus if the syndrome indicates the existence of an error a new 1 out of n word which is 12 bits long and contains all zeros except at the position indicated by the syndrome to have an error; at this position the bit is set to 1. This unique word is known as the 1 hot code and this is XORed with the received word at the XOR2 module in the figure to give us a corrected word.

The above description of the hamming code process is a simplified version of the mathematical process that occurs in the encoding and decoding process. The simplistic formulas presented above allow for the system to easily be translated into logic statements that can be implemented in VHDL. However the true mathematical process is slightly more rigorous.

The Hamming code relationships can truly be expressed in the form of matrices. For computing r check bits from a series of k data bits a generator matrix **G** is required. This generator matrix is always composed of a fixed number of elements arranged in an order; so that when any data word arranged in the form of a row matrix is multiplied with it, it generates a unique codeword for that particular data word. Since the generator matrix always remains the same the functions associated with the calculation of each check bit inside the code word always remains the same. Assume there is a row matrix D composed of eight data bits where D = [D_{k-1} ………..D_0] and from which a code word B is desired where B = [B_{n-1} …… B_0]. Then the generator matrix G can produce the n-bit codeword if the relation **B = DG** is used. **G** known as the generator matrix is composed of two different parts or matrices which are joined together to form a single matrix of values. One part is represented by the co-efficient's $a_{i,j}$ that were mentioned earlier and the other part is a identity matrix which is of size k.

$$\mathbf{G = [\ P \mid I_k\ ]}$$

An easy way to visualize this matrix is to start with a matrix Q. The columns of this matrix Q are made the binary representations of j which represents the position of each data bit in the codeword. The Q matrix is shown as follows

$$\begin{array}{cccccccc}
\mathbf{B_3} & \mathbf{B_5} & \mathbf{B_6} & \mathbf{B_7} & \mathbf{B_9} & \mathbf{B_{10}} & \mathbf{B_{11}} & \mathbf{B_{12}} \\
\left|\begin{matrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \end{matrix}\right. & \begin{matrix} \mathbf{1} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{0} \end{matrix} & \begin{matrix} \mathbf{0} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \end{matrix} & \begin{matrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \end{matrix} & \begin{matrix} \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \end{matrix} & \begin{matrix} \mathbf{0} \\ \mathbf{1} \\ \mathbf{0} \\ \mathbf{1} \end{matrix} & \begin{matrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \\ \mathbf{1} \end{matrix} & \left.\begin{matrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{1} \end{matrix}\right|
\end{array}$$

The rows are limited to four because that represents the number of check bits that are being utilized in the codeword. Thus each row and column has significance. The transpose of Q gives us the first part of the generator matrix or P. Next the identity matrix is attached to the P matrix. Since k = 8. The identity matrix used is $I_{8.}$ The completed generator matrix is shown below

$$\mathbf{G = [\,P \mid I_k\,] =} \left[\begin{array}{cccc|cccccccc}
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

The generator matrix G generates a unique codeword for any data word D arranged in a row matrix of [D_0…..D_7] the result of this is a 12-bit codeword which is arranged in the form

$$\mathbf{B = C_0\ C_1\ C_2\ C_3\ D_0\ D_1\ D_2\ D_3\ D_4\ D_5\ D_6\ D_7}$$

An important point to mention here is that after multiplication between a single column of the data word and a single column of the parity check matrix all non-zero bits must be modulo-2 XORed with each other. This operation holds true for both the encoding process and decoding process as well. From the above matrix it is also noted that the operations in equation 3 are verified since multiplication with a data word leaves only certain terms left behind. For coding purposes it is more convenient to use the formulas from equation 3 to calculate the check bits individually and rearrange the check bits and data bits to form the code word rather than doing complex matrix multiplications.

While this is the conventional notation of constructing a generator matrix, Nicolaidis[12] method of arranging the data and control bits inside the codeword to make the codeword easier to read requires the check bits to be shifted into certain locations. This slightly modifies the contents of the G which is shown below. The new generator matrix which is based on the distribution of bits described in equation 1 where all the check bits occupy positions of the power 2 (Nicolaidis' convention). Also the data bits are assembled in descending order.

$$\begin{array}{cccccccccccc}
\mathbf{D7} & \mathbf{D6} & \mathbf{D5} & \mathbf{D4} & \mathbf{C4} & \mathbf{D3} & \mathbf{D2} & \mathbf{D1} & \mathbf{C3} & \mathbf{D0} & \mathbf{C1} & \mathbf{C0} \\
\end{array}$$
$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}$$

With the codeword generated the next step involves checking to see whether the codeword received is error free; this is done by determining the syndrome of the received word. Calculation of the syndrome S requires the use of the transpose of a matrix H which is similar in many ways to the G matrix. Matrix H^T is defined as

$$H^T = \frac{I_{n-k}}{P}$$

Using the same P matrix generated earlier the H^T matrix is constructed. It should be noted that the unity matrix used in the formation of H^T is now I_{n-k} and not I_k used before. The resulting H^T matrix is shown below in its conventional method of arrangement.

$$H^T = \frac{I_{n-k}}{P} = \begin{array}{c|cccc|} C_0 & 1 & 0 & 0 & 0 \\ C_1 & 0 & 1 & 0 & 0 \\ C_2 & 0 & 0 & 1 & 0 \\ C_3 & 0 & 0 & 0 & 1 \\ \hline D_0 & 1 & 1 & 0 & 0 \\ D_1 & 1 & 0 & 1 & 1 \\ D_2 & 1 & 0 & 1 & 0 \\ D_3 & 1 & 0 & 0 & 1 \\ D_4 & 0 & 1 & 1 & 1 \\ D_5 & 0 & 1 & 1 & 0 \\ D_6 & 0 & 1 & 0 & 1 \\ D_7 & 0 & 0 & 1 & 1 \end{array}$$

When any 12-bit codeword is multiplied by the H^T matrix it outputs a row matrix which has 4 columns; this is regarded as a 4 bit word that indicates the location of any single bit error in the received word. If there is no error in the word then the row matrix is zero.

A very important relationship exists between the both the generator matrix G and the matrix H. As mentioned earlier the control bits and data bits inside the code word can be shifted depending upon the users choice (As is done in Nicolaidis' notation of the codeword) and each configuration will have a unique codeword. However, any changes in positioning the elements in G to modify the codeword also gives rise to changes in H^T. The relationship between generator matrix G and Matrix H^T is defined as

$$\mathbf{G\,H^T = 0}$$

The product of G and H^T must always be zero. For this purpose some manipulation is required when Nicolaidis' notation is used to rearrange the check bits inside the word. To match the rearrangement of the G matrix the new H^T matrix is shown below.

$$\mathbf{H^T} = \frac{\mathbf{I_{n-k}}}{\mathbf{P}} = \begin{array}{c|cccc|}
D_7 & 0 & 0 & 1 & 1 \\
D_6 & 1 & 1 & 0 & 1 \\
D_5 & 0 & 1 & 0 & 1 \\
D_4 & 1 & 0 & 0 & 1 \\
\hline
C_3 & 0 & 0 & 0 & 1 \\
D_3 & 1 & 1 & 1 & 0 \\
D_2 & 0 & 1 & 1 & 0 \\
D_1 & 1 & 0 & 1 & 0 \\
C_2 & 0 & 0 & 1 & 0 \\
D_0 & 1 & 1 & 0 & 0 \\
C_1 & 0 & 1 & 0 & 0 \\
C_0 & 1 & 0 & 0 & 0
\end{array}$$

It can be verified that both G and H^T in Nicolaidis' convention will result in zero when multiplied with each other will be equal to zero. This will also hold true for the conventional G and H as well. Only the first four rows of the G matrix can be multiplied in this instance with H^T since they're dimensions are different, but if they fulfill the equation then our matrices are correct and the coding and decoding processes will deliver correct results.

The Important rule to remember is that in G is that the single '1' columns are actually occupied by the data bits and in H are occupied by control bits. In both cases the matrix is written out to represent the position values of the codeword. The only manipulation required is that the check bits have to move to the exact location where they are located in the word in both G and H and the product of $GH^T = 0$ must hold true if the system is to work. Any shifts that are made in the G have corresponding shifts inside H. Thus foe example if C_0 and D_7 switch places in G (One of the changes between the conventional notation and Nicolaidis' method) then they also switch places in H^T. Subsequent changes follow the same set of rules

For simplification of the simulator designed for this project the use of large matrix constructs in VHDL was avoided and instead the simpler formula presented earlier in the section were used. These matrix notations were presented in this paper for the understanding of the user. The rules for the construction of the generator matrix and the matrix H^T remain the same and can be extended for use with larger size data words which require more code words as well. As long as the word satisfies the rules of formula (1) the matrices for them can be easily formed. In the next section the hamming code simulator is presented.

Chapter 4

The Hamming Code Simulator

For understanding the various concepts related to SEUs and a hamming code which can correct them; a simulator was designed to accompany the literature surveyed for this project. The simulator is composed of two modules, all designed as VHDL constructs and is completely synthesizable incase emulation of the system is required. Due to the time constraints associated with the undertaking, the emulation of the system was not possible on a FPGA platform. However, this can be left for future work. In order to conduct successful simulation and testing a number of functions outside VHDL libraries were needed for input files to be generated; these were done in MATLAB. The following section is arranged in a step by step guide to allow for easier understanding for future users of this simulator on how the entire process is run. To start a WAV format audio file that is recognized by MATLAB is used and the goal of the simulation is to pass the data from this WAV file through different SEU scenarios and reconvert this data back to the original WAV file successfully. For the purpose of comparison, the simulator outputs two sets of data. One that is reconstituted from a system that underwent continuous SEU's from simulated radiation and was able to correct itself using hamming codes; and the other from a system that underwent the same SEU's but had no correction mechanism and therefore suffered from a disruption in data. The following sections provide a step by step approach to use the simulator and explain the architecture of the Hamming Code simulator.

4.1 System Input

The system implemented in VHDL was based on encoding 8-bits of data with 4 check bits to form a 12-bit code work. Thus in order to make audio files work seamlessly with the system, any audio file that has to be run through the simulator requires to be converted into a data stream of 8 bits in binary format. For this purpose MATLAB functions are instituted to convert the audio file. Starting with an audio file of .WAV format (The standard audio format used by MATLAB); the file is read into MATLAB using the “wavread” command saving it a string of values in a user specified variable (or by default to a variable called “ans”) in native format (**wavread (filename, ‘native’);**). With the data saved to a variable, a new file must be opened to save the data to. New files must can be opened via the fopen command (**myoutput = " fopen('aitest_file.txt','wt');).** The command here opens a user specified file **aitest_file.txt** in write format. Next data is written to the file using **fprintf(myoutput,'%u\n',ans');** this command writes data to the file aitest.txt. The data here being integers (described by %u) each on a new line (described by \n) from the variable ans. Finally the file is closed using **fclose (myoutput);** which closes the file and saves our work.

This creates an array of integers between 0 and 255 (unsigned 8 bits) saved to a readme file with each entry separated by line. Conversion from the WAV format to 8-bit binary numbers yields an extremely large file, therefore the audio clip used for testing the system was limited to 16 seconds. Even then the data extracted from the sound file amounted to 755,872 words. Despite the large number of binary words the data is processed quickly by the hamming code system.

This integer file now is converted into binary format in order to make it easier for the hamming code system to use. For this purpose an integer to binary converter was designed in VHDL to create a mirror image of the data from the inter range 0 to 255 to 8-bit binary words. The code for this is presented in index(). Since the binary converter converts data and writes it in output to an external file, it cannot be synthesized and instead is designed to simply be simulated and output a text file of data. With the binary data file now created it can be forwarded to the hamming code simulator for processing.

4.2 The Hamming Code Simulator (Top Level)

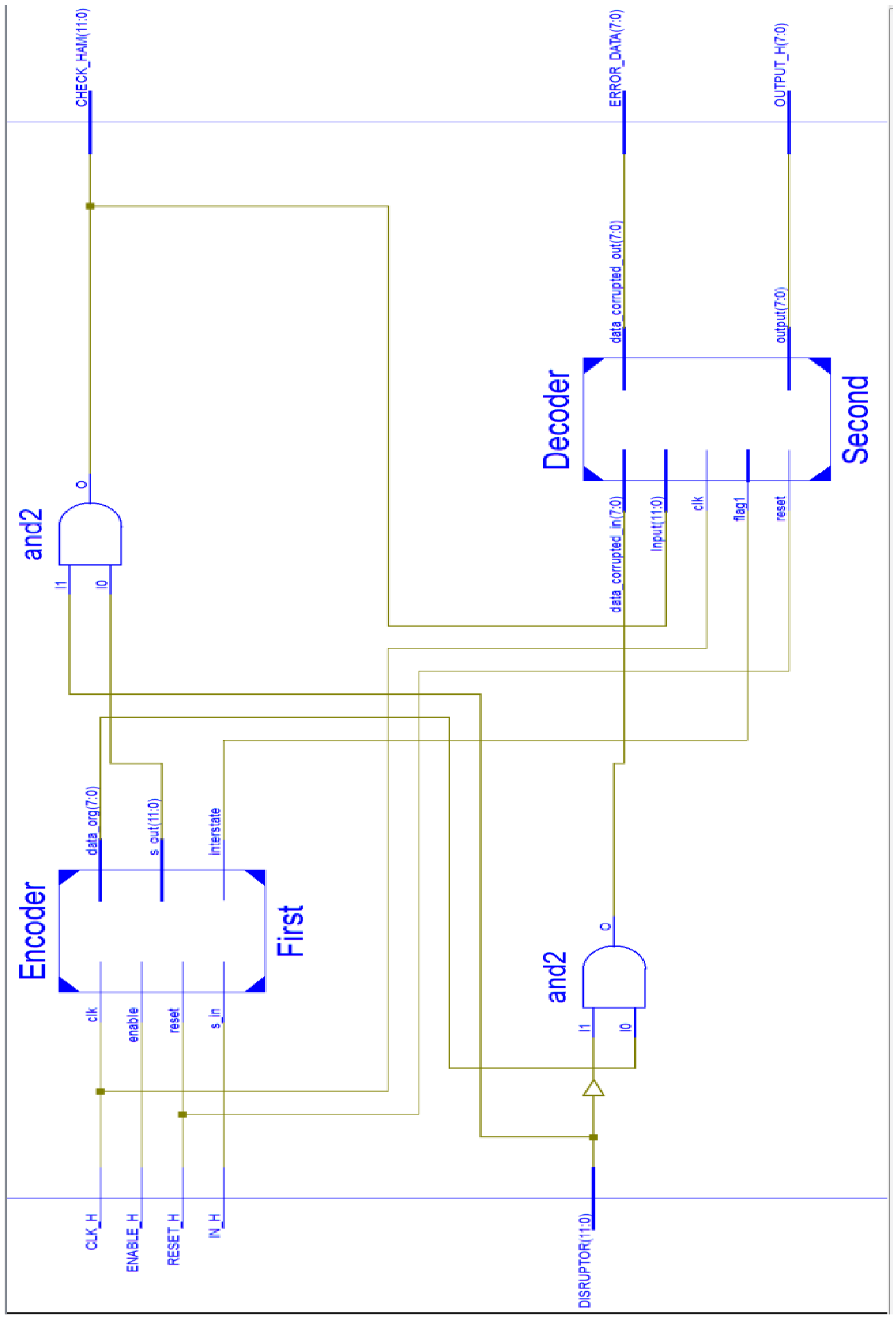


Figure 4.1: Top Level Diagram of Hamming Code Simulator

From the top level the Hamming code simulator shown in figure 4.1 is essentially made up of two modules. It consists of an encoder which takes in 8-bit binary data and outputs two different sets of data. One set of data is encoded with the hamming code check bits and is therefore a 12-bit output and the other is an 8-bit output which is un-coded and is supposed to mirror data that has no protection against single event upsets. The system was designed to let the data from both sources undergo the same processes, delays and exposure to simulated SEUs to see how they react under different conditions.

The output lines of the encoder are connected to a testing mechanism. Each bit line is connected to the input of an AND gate that allows control over the values passing in between the encoder and decoder. The default value of each of the other inputs of the AND gate are set to 1 so that data coming in is passed the same way out. However to cause a bit flip all that is required is to change the controlling value of the AND gate to zero. This will cause any incoming data from the encoder to be overwritten with a zero. While this may not be able to corrupt 0 value bits being transferred between the encoder and decoder it still serves in a useful way for testing the system. A block diagram of the top level is shown on the next page. These AND gates that are used for disrupting the values outputted by the encoder are labeled as "disruptors" in the diagram shown below.

Once the lines have passed through the disruptors they are passed on as input to the decoder where the decoding process begins. To keep check of how the SEU generation mechanism is affecting the codeword an output has been taken from the disruptor AND gates and used as an output for the module. The output is labeled as the "CHECK_HAM" line (shown in the diagram). The output was originally used to determine where the SEU landed on the parallel lines carrying data between the encoder and decoder so that the functioning of the hamming code correction mechanism could be corrected.

4.3 The Encoder

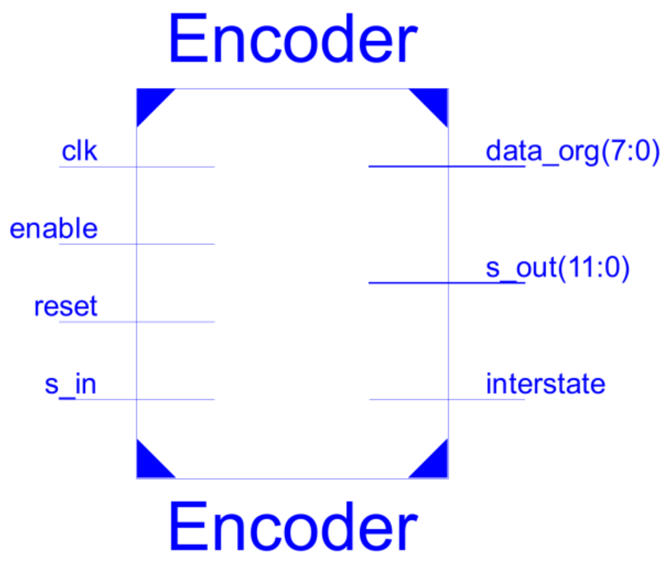


Figure 4.2: Block Diagram of Encoder Circuit

The encoder circuit in figure 4.2 is responsible for outputting two sets of data. One set of data contains 12-bit code words and the other 8-bit data words which carry no encoding. To make the system resemble other computer systems; data was fed serially into the system via the data line "s_in". Data is read serially from the text file outputted earlier on by the integer to bit converter using the test bench. The test bench is designed to read each line of the file one by one and send data to the input line one bit at a time, so it is important that the format of the file always remains the same and each 8-bit is located on a new line. A reasonable duration of time is allowed to give them system time to accept the data and appear in the internal registers and data lines so it is visible during the simulation. Each bit is then stored into internal data registers and once a line is completely read, the encoder makes use of the hamming code formula presented in equation (1) to generate check bits from the data bits read in and store those into control bit registers. Finally the codeword is formed using the Nicoladis' convention to arrange the bits into a specific order. This is then output in parallel via the "**s_out**" line. At the same time as this data that is stored in the data registers initially is output in 8-bit format to reflect the un-encoded data via the “**data_org**” line. One of the most critical lines coming out of the encoder is the “**interstate**” line. This line is essentially a flag that tells the system each time a new word is being sent out by the system. It is in effect a handshaking signal that allows the decoder to recognize that new data is coming forward and it must be ready to decode it. The signal is vital for the synchronization between the encoder and decoder to occur. The effect of an SEU on this line will further be investigated in section V of this paper when simulation results are presented. Further information on the internal operations of the encoder module can be found in the index section where the code is well commented.

4.4 Decoder

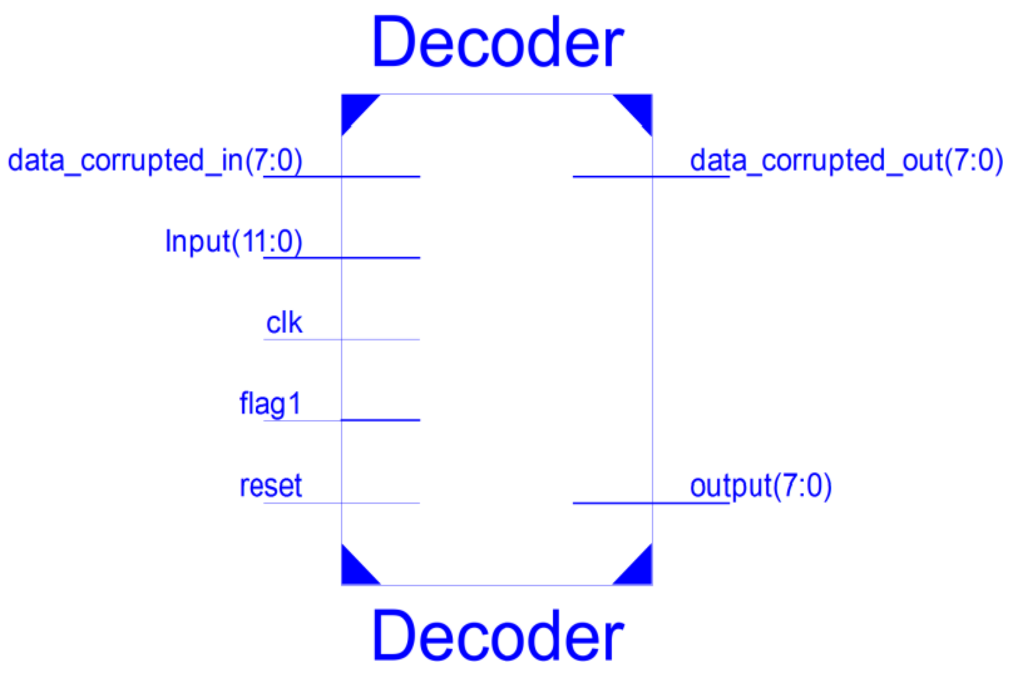


Figure 4.3: Block Diagram of Decoder Circuit

The decoder unit shown in figure 4.3 is quite similar to the encoder in its design. The un-coded data word is received via the “**data_corrupted_in**” input line; while the coded word is received via the "**input**" line shown in the diagram. "**Flag1**" represents the input for the "**interstate**" signal transmitted out by the encoder to signify that a new word is being transmitted to the decoder on the data lines. The decoder then begins to reconstitute the check bits from the data bits inside the code word and compares them to the check bits contained inside the code word received. If an SEU has occurred and the data is corrupted in a single bit then the decoder simply computes the location of the error using the syndrome and flips the bit over. Once the word has been corrected it passes the corrected data word out via the "**output**" line. At the same time the corrupted data word is made to wait until all necessary checks on the coded word is performed and both sets of data are output together. The corrupted data word is output through the "**data_corrupted_out**" line.

The decoder also has an additional function. While originally designed with the concept of outputting binary data; to facilitate the reconstruction of data back to the WAV format, additional lines of code were added to its function. Thus once the data has been received in binary format; the decoder converts both the corrupted data words and the corrected data words to integer format and writes them simultaneously to two separate text files. This is necessary because the import of files with the 8-bit binary numbers into MATLAB results in erroneous reading; MATLAB imports the data in double format, making it unable to recognize the bits as 8-bit binary words. So when the data is saved to an array it caused the values like "00000001" to be saved simply as "1" amounting in loss of data. After numerous unsuccessful attempts to convert the 8-bit binary data directly back into WAV format; the decoder code was modified to write both outputs to two different text files in integer format. The integer format follows the standard MATLAB convention of 8-bit unsigned numbers representation and gives us files in which integers can range only between 0 and 255 making the conversion back to WAV format much simpler.

4.5 Output

With the transmission, disruption and correction process complete, two sets of data are available in two different text files. The sure way to test if the data has undergone any corruption is to reconvert the data back to the WAV file it was originally extracted from and see how much the data has changed. To do this; simple functions from MATLAB were employed. Data is imported into any user defined variable using the **A = importdata ('decoder_out.txt');** command. This saves the values from the decoder_out.txt file which in the code was assigned to corrected data to a user defined value A. The data is saved as an array but its format is read by MATLAB as double. To correct this; the data type is changed from double to unsigned 8-bit integers format using the command **B = uint8(A);** this allows MATLAB to recognize the integers in the file as belonging to the 8-bit unsigned format without affecting any of the underlying data or causing any conversions. Finally the data can be converted back to the WAV format using the command **wavwrite (B, 44100, 8, 'decoded.wav');** Where B is the array containing the integers representing unsigned 8-bits, 44100 represents the sampling frequency, 8 represents the bit depth of the data and "decoded.wav" represents the name of the WAV file to which the data must be saved. This saves the data back into a file called decoded.wav in the work directory. The same process can be repeated to convert corrupt.txt the text file containing corrupted data back to a WAV file as well.

It was important to use the sound file for data comparison between corrupted and corrected data because graph plotting proved difficult due to the sheer volume of data. This made it impossible to compare different points in time and see how they had differed. Even if a small segment of the data could be chosen in both the data streams and compared it would not show much difference since large portions of both streams just use the value of 128 to signify silence and very little difference is observed until both sounds are played back together. The different simulation tests run on the hamming code simulator are shown in the next section.

Chapter 5

Simulations and Analysis

In this section the simulation results of a numbers of tests are presented. For the duration of this study a number of assumptions are made to help understand the concepts associated with SEU's. A list of these is presented below.

5.1 Assumptions:

1. SEUs are understood to occur randomly and occur for a short duration in very specific locations. To mirror this concept it is assumed that the SEU effect will last for less than a clock cycle (< 10ns). For the localized radiation effect the system disruption will not be directed over entire modules but single bits of registers and bit lines used for data transmission, since the objective of the simulations is to test for SEUs.

2. To mirror the disruption caused by SEUs, the bits values of data registers, flags or data lines are simply flipped to the opposite value for a short duration. In reality SEUs cause not only bits flips but may disrupt the value of registers so that they become undefined or unreadable, in the occurrence of this event in the simulation the system will be unable to calculate either check bits or correct other values since it uses very simplified combinational circuitry to perform its operations.

3. Testing is limited only to SEUs in registers, flags and data lines across the system and does not encompass testing for MBUs or SETs. A test is carried out to observe the effect of MBU on the Hamming code during the trials.

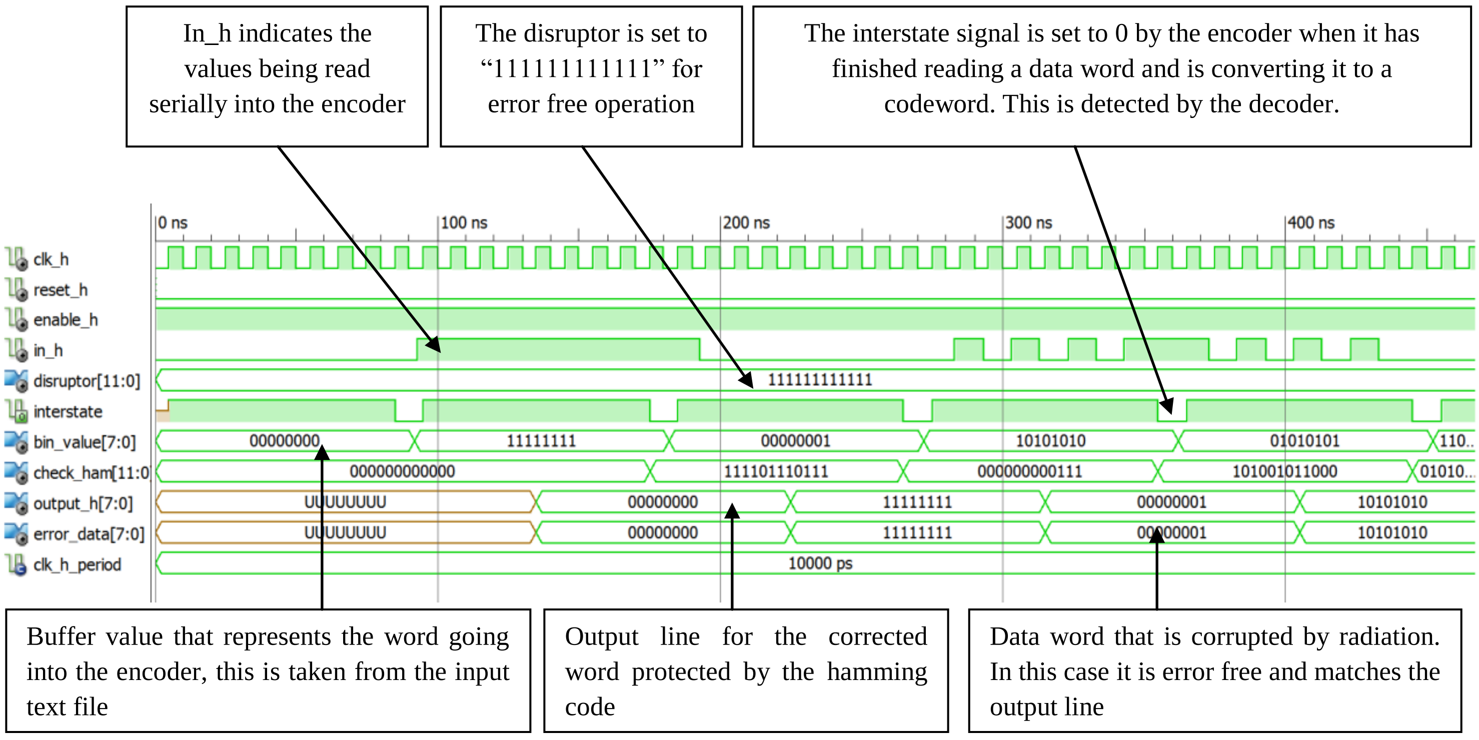
4. A critical assumption for the duration of testing is that while the system was designed for implementation on an FPGA. It is assumed that the configuration of the Hamming Code system cannot be disrupted by SEUs and only the data passing through the system can be

disrupted by radiation. In reality the control registers on board the FPGA platform are equally susceptible to radiation as the data registers. In the event of such an occurrence there is probability that a catastrophic failure will occur which would disrupt the configuration of the system and not allow it to perform its function. This is beyond the scope of the currently created simulator and requires further understanding to incorporate a scrubbing mechanism to rewrite the data held be the configuration registers and reset the system.

The assumptions outlined in the above section serve as a guideline to begin system testing. However after extensive testing it was discovered that some of these assumptions required modification to allow different phenomenon to be observed.

5.2 Simulation I: The Error Free System

To verify that the system is operating under normal parameters the first simulation carried out is for an error free system.



In_h indicates the values being read serially into the encoder

The disruptor is set to "111111111111" for error free operation

The interstate signal is set to 0 by the encoder when it has finished reading a data word and is converting it to a codeword. This is detected by the decoder.

Buffer value that represents the word going into the encoder, this is taken from the input text file

Output line for the corrected word protected by the hamming code

Data word that is corrupted by radiation. In this case it is error free and matches the output line

Figure 5.1: Simulation of Error Free System

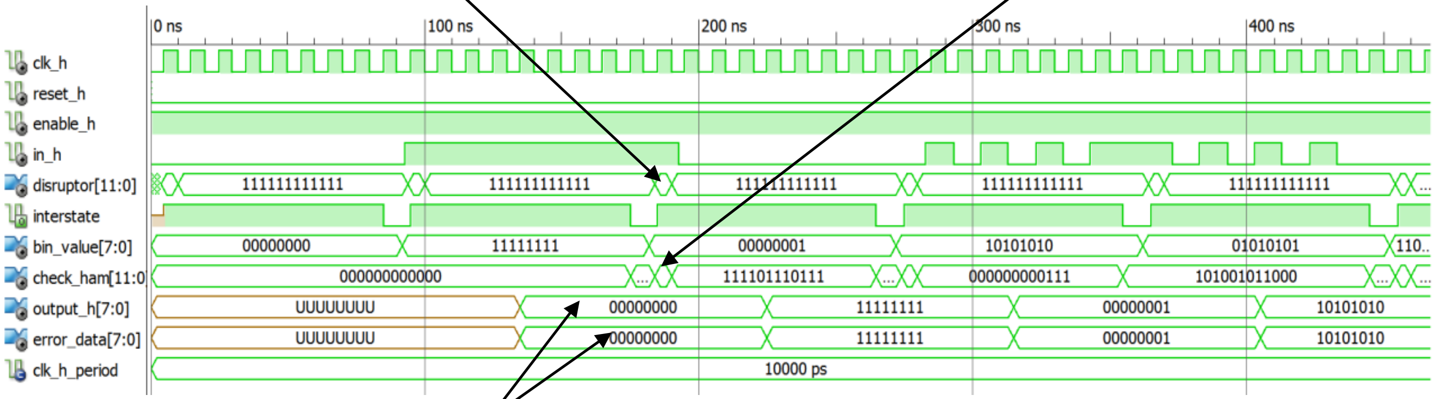
The above figure demonstrates how the system behaves in an error free environment Along with associated signals that will be observed during the trials.

<u>5.3 Simulation II: SEUs occurring on the transmission lines between encoder and decoder</u>

Next the occurrence of SEU occurring on the data lines is investigated. Following the assumption made earlier, an SEU is allowed to change the value of the disruptor for a period less that 10ns which is the standard clock cycle being used in the system. The result of the simulation is presented in the figure below.

The disruptor line is switched to a value of “111111111011” for 6ns and the same error occurs repeatedly through the simulation

The Hamming code detects the error and quickly modifies the codeword to indicate the error. However as soon as the error is gone it switches back



Even though a SEU has occurred and the system registered that it was struck, the values of both corrupted and corrected data remain the same.

Figure 5.2: Simulation of Short duration SEU

The result from this simulation is extremely important in understanding how SEUs affect electronic systems. In the figure it can be observed that an SEU was activated for 6ns and is visible on the disruptor line by the rapid change is value. This SEU is detected inside the system as the “**CHECK_HAM**” signal indicates that it switched values for a short duration, the

Hamming operation is a combinational circuit so as soon as the error disappears the Hamming word changes back to normal. Even though the error is registered in the system the data word that remains un-coded and should be susceptible to any radiation appears error free whereas it should have detected the SEU and its data bit D_0 should have switched from '1' to '0'. This is what is described as a masking effect produced by the circuit. Masking effects were described in [8] and are of three types

Timing masking*: The SEU arrives too soon or too late at the input of the storage elements to be captured. [8]*

This stems from the fact that Storage elements capture the value on their inputs at particular clock instances. Thus in order to be captured; the SET must either arrive at the storage elements input at that exact particular time to be captured or must exist for a period much greater than the clock cycle of the storage element in question.

Electrical masking*: The pulse is attenuated during its propagation such that its amplitude is too small for the SET to be captured.[8]*

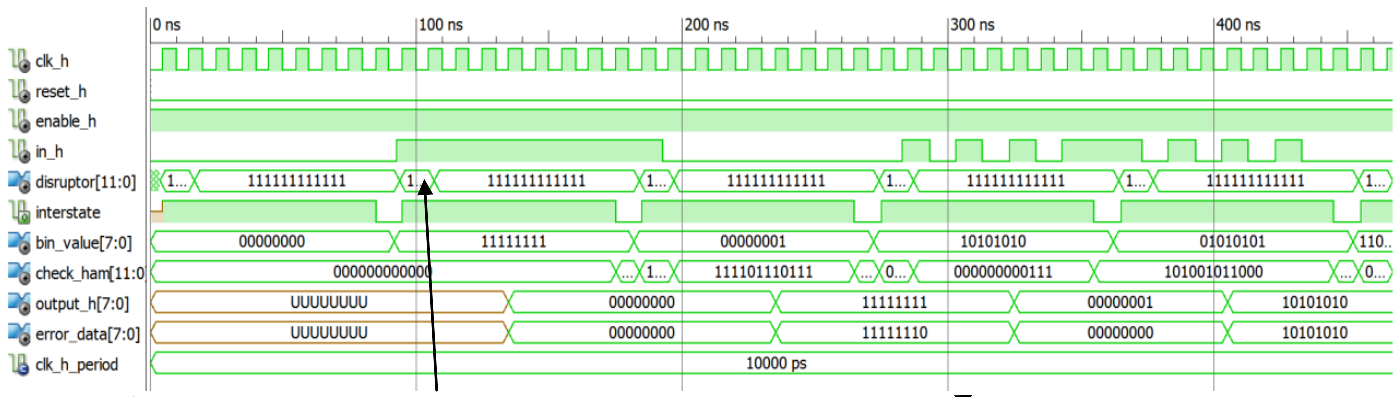
Electrical masking is related to the voltage pulse that is imparted into the circuit when a particle of radiation strikes the circuit. This transient pulse moves through the circuit and reduces in amplitude in time, until it is no longer big enough to be registered by the storage element as a valid value.

Logical masking*: The inputs of the combinational circuit are such that the logical path for the propagation of the SET is blocked.[8]*

The logical masking concept originates from a combinational circuits ability to mask errors through controlling values on different logic elements that negate the effects of transient pulses.

For example: if the circuit consists of a 2 input AND gate and one of the inputs already contains a "0" value. Then any SETs appearing at the other input of the AND gate get cancelled out.

In the case of the simulation being conducted here, the SEU is blocked out due to the occurrence of timing masking. The SEU duration is too small and occurs at a time when data is not being read by the decoder from the transmission lines. Also the simulator runs on clock signals that tend to use the positive clock edge of the signal to trigger events changes. Thus if the SEU does not extend across this positive clock edge or is not long enough to for the system to detect it; then it is masked out. In order to conduct further study it is necessary to amend the original assumption and increase the duration of the SEU to greater than ten nanoseconds (>10ns) this will ensure that the SEU will extend across a complete period and have a chance to be detected and not overwritten. In the next simulation the duration of the SEU is extended to 14ns. The simulation result is as follows. Even with the extended signal the SEU must still occur at a point where the decoder samples the values from the data lines, otherwise it is completely missed



again

It can be seen that the SEU occurs is not only longer but also occurs just when the data lines are being read by the decoder as well at the positive edge of the interstate

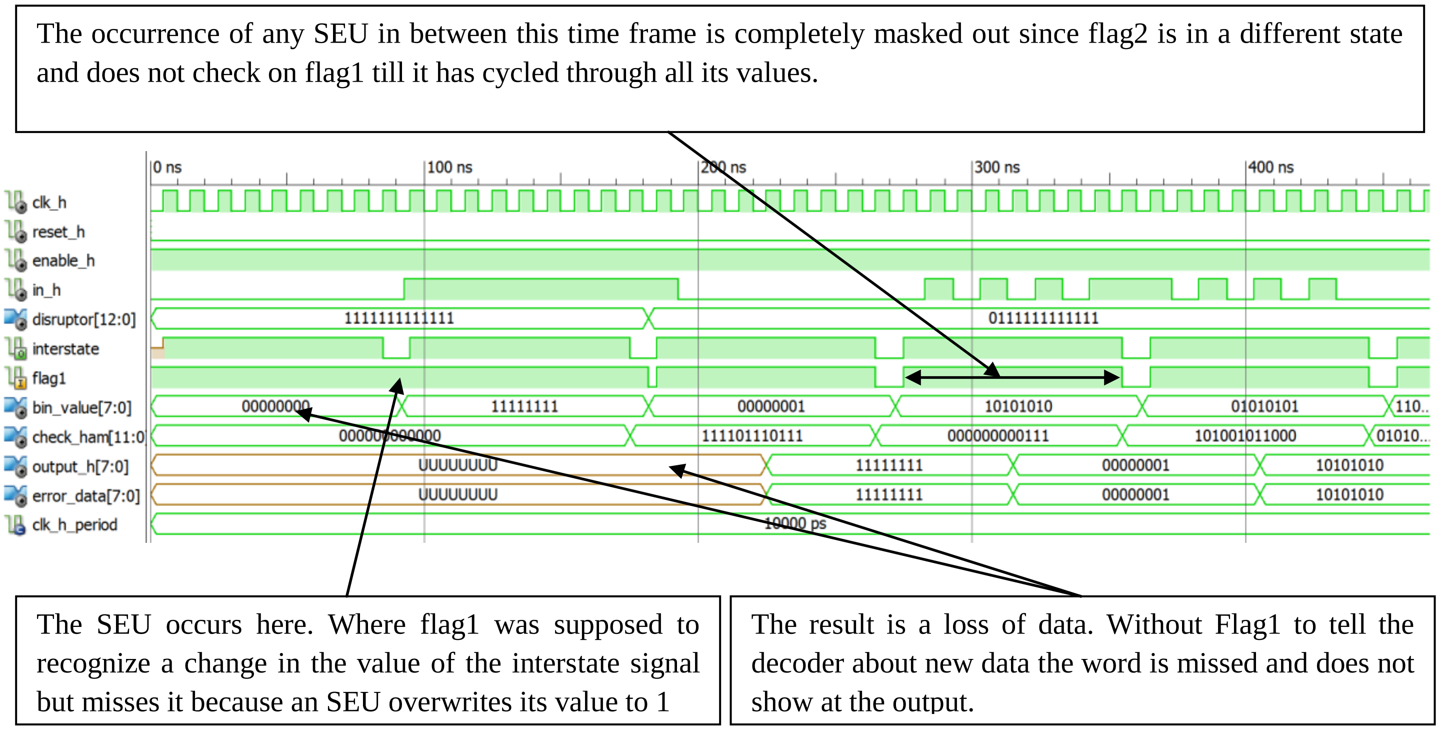
Upon extension of the duration of the SEU it is observed that the data line is indeed affected as its 0 bit flips from 1 to 0 showing that timing masking failed to mask the SEU.

Figure 5.3: Simulation of Longer Duration SEU

From the simulation results it is observed that the radiation pulse is long enough to actually extend over a clock cycle and is able to avoid the timing masking effect. The timing masking effect is common among synchronous systems. In the past when electronic system ran at lower frequencies and the clock cycles were much longer; SEUs had a much lower probability of affecting systems because the sampling of values had a longer duration and any disruptions were easily overwritten. This situation however is rapidly changing due to the use of high frequency components which make the occurrence of SEU more likely than ever before.

5.4 Simulation III: SEU occurring on Synchronization Signal lines

Next a simulation test is done to see how the occurrence of a bit flip in the synchronization signals affects the system. To do this a brief understanding of the decoder module is required. The decoder module uses two flags in order to recognize that state in which it exists. Flag1 signaled by the interstate line of the encoder tells the decoder that a new word is now available to the decoder for processing. This flag however does not start the decoding process. A second flag called "**Flag2"** is responsible for cycling the system through different states once **Flag1** indicates a '1' or that a fresh word is available. It has four states; State 0 which resets Flag 1 and reads the values of the input lines; state 1 that calculates the check bits, state 2 that compares control bits from both sources and if they are equal, forwards the data word to output in state 4 otherwise it calculates the syndrome and moves to state 3. In state 3 the syndrome is translated to a location for the corrupted bit. This corrupted bit is then flipped to correct the word. Finally state 4 is when the system sends the word to the output and resets **Flag2** to state 0. **Flag1** is only checked when the system is in state 0 of **Flag2**, this was done to avoid overwriting of data if the system was already processing a word and a new word became available before it could output the current word insider the decoder. Since **Flag2** is an internal signal we do not simulate it and assume the system is safe from SEUs. However **Flag1** is essential to the system and since its information is carried via external signals the effect of an SEU is investigated and presented here. The simulation result is presented below.


The occurrence of any SEU in between this time frame is completely masked out since flag2 is in a different state and does not check on flag1 till it has cycled through all its values.

The SEU occurs here. Where flag1 was supposed to recognize a change in the value of the interstate signal but misses it because an SEU overwrites its value to 1

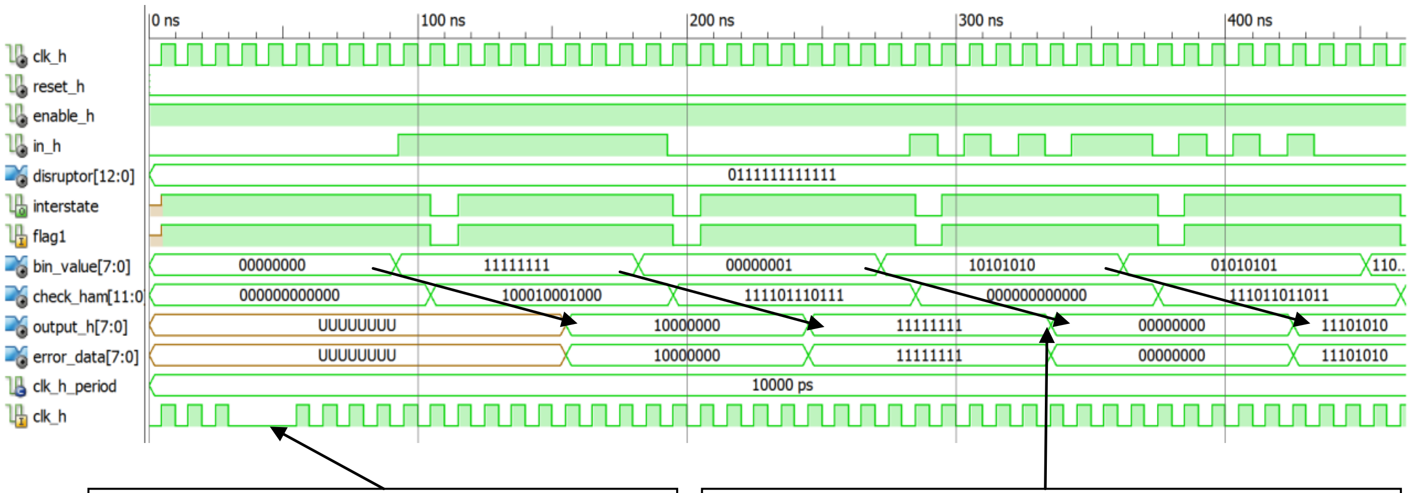
The result is a loss of data. Without Flag1 to tell the decoder about new data the word is missed and does not show at the output.

Figure 5.4: Simulation of SEU on Synchronization line

The result is as expected. Without the synchronization signal to indicate a change in value the decoder fails to recognize a new word and ignores the data thus resulting in data loss at the output. While this only occurs in data loss; if Flag2 inside the decoder was to be corrupted by an SEU the effects would vary considerably and lead to a range of possible outcomes from a minor delay due to a step back by the state counter to a decoder failure because the state counter is put in a state where the system cannot recognize what action to take. Thus protection of synchronization signals is crucial if the system is to operate. This is elaborated further by testing the effect of SEUs on the primary synchronization signal the clk.

5.5 Simulation IV: SEU occurring on the clock lines

The clock line is the most essential signal inside any electronic system. Not only is it necessary to trigger event changes, but it is crucial to ensure that different modules may be able to synchronize together. Many electronic systems make use of the clk edges to synchronize their data. This helps the system to understand the state which it exists in. In the simulator system presented here the clock is vital for the encoder and decoder to co-ordinate operations. The encoder is constantly reading data from the text file as if a real system was constantly sending it data which had to be read continuously and forwarded to the decoder. If either the decoder or the encoder lose track of the clk signal the result may prove catastrophic. The simulation of this test is presented below



Clock signal for the hamming code system. The blank position indicates the occurrence of an SEU as the natural gets overwritten by a zero

The disruption in the clocking signal values causes a number of problems as it can be seen that future outputs remain corrupted and not match input values.

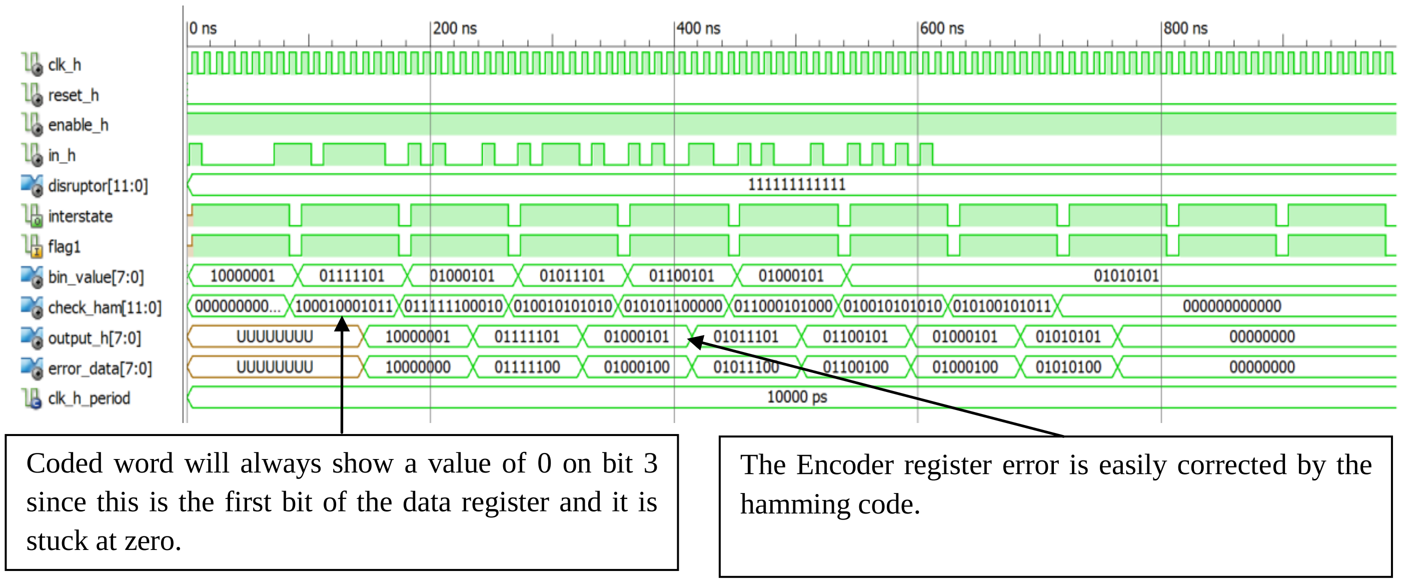
Figure 5.5: Simulation of SEU on clocking line

The following simulation shows just how dangerous an SEU can be if it hits a critical line within the system. Electronic systems vary in their design; some systems make use of a global clock for all modules while others make use of a localized clock for special modules. In the simulator design the serial input into the system is not controlled by the CLK_H signal used by the encoder and decoder. The inputs simply arrive after a clock period and wait for one clock period before they change. The entire encoder read in process is dependent upon this, so if the clock runs correctly the system has no problems. However the encoder and decoder depend on the incoming CLK_H signal to trigger their events and keep them synchronized with any external system. When an SEU strikes this line and causes the value to flip or overwrites it; it causes the system to lose its ability to trigger events correctly. So the encoder and decoder are essentially set off track in time; in the mean time serial inputs to the encoder keep arriving and changing since it is assumed that the system behind the encoder and decoder remain unaffected. Thus the result is that the output is constantly incorrect because the data being read into the system is being misread and out of synch. The Hamming code cannot correct this problem since the input to it is incorrect to begin with. The effect caused by the SEU here is no longer a soft error, because the error does not remain temporary and continues as long as data is being read in. This is therefore an example of a hard error because a reset of the system is needed to correct the synchronization for both the encoder and decoder. While this happens to only the encoder and decoder modules it is hard to imagine how much damage a disruption in the clk line would cause a real system with an extended number of modules as it propagated through the entire circuit. Simple software resets of every module would not fix the synchronization problem it would require a system wide reset to bring every module back to synchronize together.

There is also no easy way to correct such a problem, while it is possible to utilize correction mechanisms on data lines and control lines to fix erroneous signals there is no easy way to correct a clock signal as it serves as the main guideline for other systems to work with. Any delays or faults in the signal automatically propagate to the entire system. This is the exact reason why TMR is essential to clock lines to ensure that the probability of an error on this critical line is avoided at all costs

5.6 Simulation V: SEU occurring in data registers

SRAM based memory cells represent the most vulnerable component of an FPGA based system. For this reason it is important to conduct a simulation of how any disruption of memory cell values can affect our system. This simulation is expected to follow the same behavior as SEU occurring on the data lines. In order to be classified as an SEU any voltage pulse occurring on the data line must be registered incorrectly for a value by a register at the end if it is to exist in the system otherwise the value either propagates through the entire system until it dissipates and is no longer recognizable or is masked out by one of the masking effects prevalent in the system. For this simulation the value of a data register in the encoder data_reg(0) is permanently set to bit value 0, this is similar to have a stuck-at-0-fault or a physical error. While this does not encompass the dimensions of testing for SEUs alone it shows that the system is robust and can handle not only temporary disruptions to data registers but also permanent faults as long as they remain single bit faults. The simulation result is shown below.



Coded word will always show a value of 0 on bit 3 since this is the first bit of the data register and it is stuck at zero.

The Encoder register error is easily corrected by the hamming code.

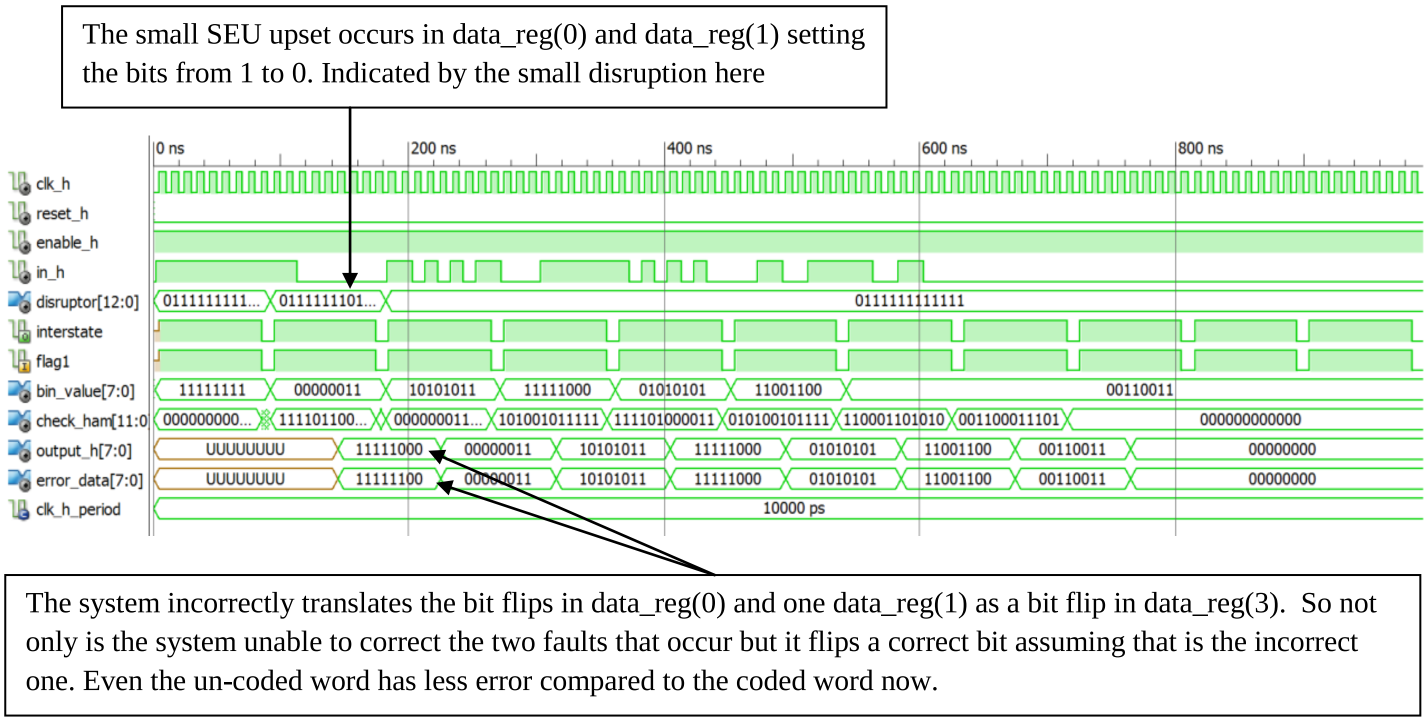
Figure 5.6: Simulation of SEU occurring in data registers

To observe fault occurrences, the first few inputs into the system were made words that had a 1 bit at the end of them. As can be seen in the above diagram the system recognizes the fault and is able to correct it with each output however the data line not protected constantly remains corrupted by the influence of the error.

An interesting result observed in the testing of data registers was that although the error can be corrected for data registers in the encoder circuit; However if an error occurs in a register in the decoder side it depends on what register and how far the checking process has resumed. For example if the check bits (recalculated and received) have just been recomputed in the decoder and are just being compared when an SEU strikes one of the registers holding the copy of the received word. In the event of such a case the SEU has already progressed too far up the line for even the Hamming code to detect and correct, In this case the error is unavoidable and leads to silent data corruption. Thus if the system is to be robust it is essential that the decoder circuits are radiation hardened to avoid errors.

5.7 Simulation V1: MBU effects on the Hamming Code Simulator

In the following simulation the limitations of the hamming code process are put to the test to see how the system behaves in the occurrence of MBU. Hamming codes were specifically designed with the ability to correct single bit errors. Their uniqueness lies in the fact that they are able to correctly diagnose which bit the fault has occurred in. When radiation strikes the data registers and bit flips occur in two different places inside the same word an MBU has occurred. The Hamming code system still recognizes the problem and attempts to correct it. The problem is because the error has occurred in two different places the syndrome inaccurately translates where the bit flip has occurred. This is shown in the simulation result below.


The small SEU upset occurs in data_reg(0) and data_reg(1) setting the bits from 1 to 0. Indicated by the small disruption here

The system incorrectly translates the bit flips in data_reg(0) and one data_reg(1) as a bit flip in data_reg(3). So not only is the system unable to correct the two faults that occur but it flips a correct bit assuming that is the incorrect one. Even the un-coded word has less error compared to the coded word now.

Figure 5.7: Simulation of an MBU

From the simulation above it is observed that the original input is “11111111” and bit flips occur in data_reg(0) and data_reg(1) making the word “11111100”. The SEU is generated after the hamming code word is already formed so the translation procedure should be able to detect the error. The problem is that when the syndrome is calculated because they are two errors in the data and the algorithm only expects a single error it incorrectly translates the location of the error to data_reg(3) and flips that value. So the system fails to not only diagnose the correct errors but it further adds to the error by flipping a completely different bit. This is a very big limitation of the Hamming code algorithm and reduces its effectiveness. A few solutions have been proposed to solve this problem. Nicolaidis in his book [4], talks about the use of an extra parity bit that can be used to solely monitor the existence of MBU. In the event of an MBU the extra parity would serve to indicate that an MBU has occurred but still be unable to diagnose the location of the two errors. For countering the effects of MBUs the Reed-Solomon code is most effective but is beyond the scope of this project. Further reading can be found in [4]

5.8 Simulation VII: Multiple SEU simulations

The last simulation conducted in the study of the Hamming code simulator is a stress test to see how extensively the system can deal with single event upsets. As mentioned earlier a 15 second audio clip was used to test the system extensively over 755,872 words. The system was able to accurately reproduce each word and reproduce the same sound clip at the end of the test even with the continuous occurrence of simulated SEUs in every transmitted word. The system however is able to only correct the SEUs that occur in the data lines and is unable to correct errors that occur on any synchronization signals like **"Flag1"** and the **clk** signals. As long as the errors occur on the data registers and they are SEUs only; the Hamming code system is able to correct them, provided they occur at a point in the translation process where the system is still able to detect them. The segment of the simulation result is presented below.

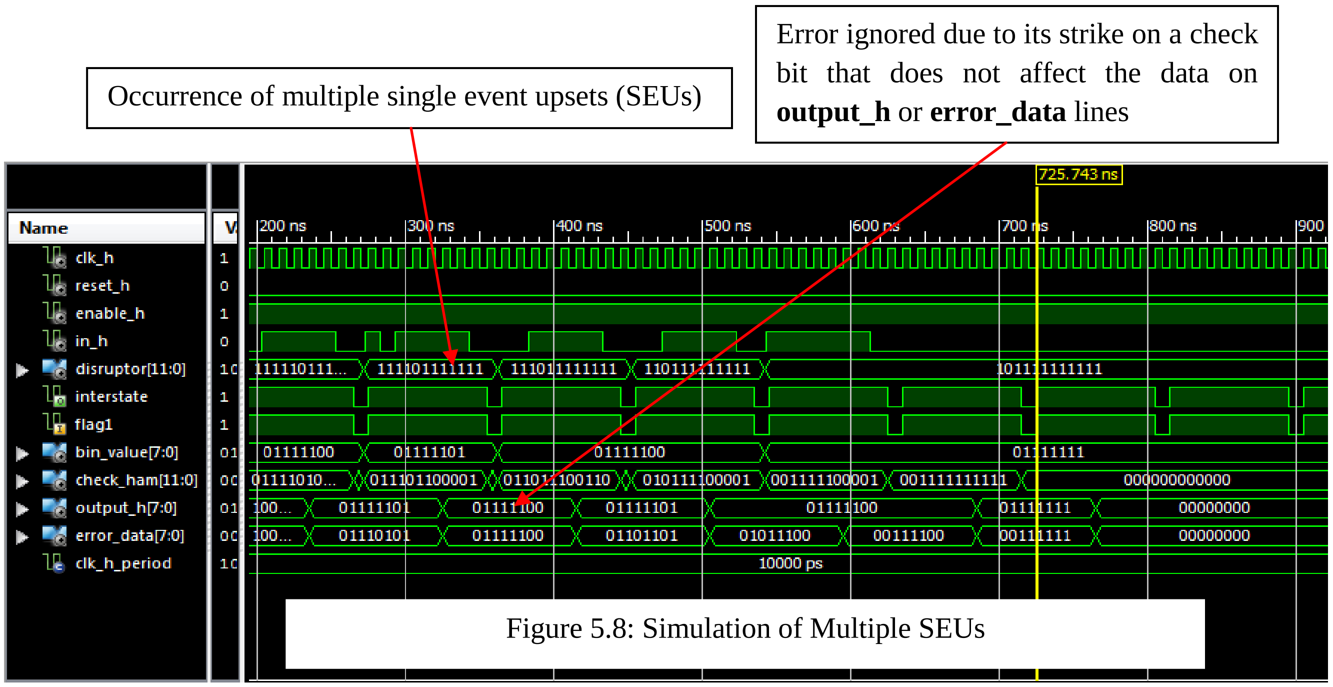


Figure 5.8: Simulation of Multiple SEUs

To make the system fairer, The SEUs were constantly aimed at the codeword being transmitted. The focus was to test how the hamming code reacts when data bits and check bits are both are affected by SEUs. While the hamming code is able to correctly identify if a check bit is struck and correct it as well, it does not make any difference to the output. Since SEUs are only being considered; as long as a data bit is not struck the data remains intact and the impact on the words if a check bit is flipped can be ignored. To be fair to the un-coded word (for the sake of comparison in the reconstructed audio clips) the exact same radiation effect was applied to the un-coded word as the coded word. Thus it is visible in the simulation above that at times the system seems unaffected although radiation strikes the hamming code word. This is because a check bit is being struck and this effect is ignored in the un-coded word and coded word both.

Chapter 6

Results

In the following section a summary of the results analyzed in the previous section are presented in a concise manner. One of the major advantages that the Hamming Code system possesses over solutions like TMR is that it adds very little complexity to the overall system, especially if its implementation is considered with FPGAs. The entire Hamming code system can be implemented with simple logic components, its calculation for the code word and decoding of the code word taking very few clock cycles. Thus it offers a much better alternative in terms of area and time overhead if we consider how much area and power the TMR method takes up in comparison. In the above simulations the effect of SEUs was tested on different parts of the circuit. A summary of results is presented below

Radiation Effect	Result
SEU occurring on data transmission line	Error either mitigated due to masking effects or corrected if detected by hamming code
SEU occurring on synchronization line	Data loss. As system is unable to detect availability of new word.
SEU occurring on Clock line	Hard Error. Clocking line disturbs the synchronization between the Hamming Code modules and external system. Extensive data corruption occurs.
SEU occurring in data registers	Error detected and corrected if it occurs within the limits of Hamming Code processing
MBU on data transmission lines	Error detected but incorrectly translated increasing error in the output.
Extensive SEU Test	System is able to correctly detect and correct 100% of errors occurring on data transmission lines and accurately reproduces WAV file used as input

The goal of the project was to design a Hamming Code simulator that is able to detect and correct SEUs on both data transmission lines and data registers inside the modules. This has been successfully achieved with a 100% accuracy rate. Also it has been proven through analysis that the Hamming Code can correct all single bit errors that occur on the data lines but is unable to correct MBUs and is not implementable on synchronization or clocking lines since these control signals are time dependent. For the synchronization flags and timing signals it is preferable to use the TMR system as it allows uninterrupted processing so long as the structure of the TMR system remains intact.

Chapter 7

Conclusion and Future Work

While radiation induced errors are only slightly beginning to influence systems at the ground level today, they are already prominent in aircrafts operating at higher altitudes and in spacecrafts outside the earth's atmosphere. A major concern of industrial practices today is that most of the electronics market focuses heavily on producing everyday commercial computing systems which are not affected by radiation induced errors that much. Thus these systems tend to continue on their trend to higher complexities and faster clock speeds with shrinking transistor sizes which make them much more susceptible to radiation. However they also represent the cutting edge of technology and contrary to popular belief they can perform much better than computer systems based on spacecrafts and airplanes. In comparison the aeronautics industry and space industry is beginning to lag behind in technology because their systems require much greater reliability in order to operate in their respective environments. Thus the design process in these systems often involves lengthy simulations and careful planning before the design process can even begin. One example of this is systems based on radiation hardening through transistor sizing which require mapping out the effects of radiation on different parts of the system and then evaluating the reduction in component sizing required to reduce the FIT rate. A proposal for this system is presented in [8]. This tends to not only be a costly venture but also a really lengthy venture as it take a lot of time to complete, thus by the time the system for one of these industries is fully developed and ready to be implemented it already lags far behind normal industry design practices and technology that is already available on the commercial market. A detailed discussion about this trend can be found in [12]

To make up for this gap in performance and reliability there is a need to introduce mitigation techniques that are able to counter the effects of radiation and provide the required high standards of reliability while not compromising on the performance of the system. They are a number of promising solutions now in development and there is a marked improvement in the mitigation techniques being brought forward as interest in this particular domain begins to rise. Originally basic techniques like TMR and Scrubbing allowed the capture of errors but were unable to correctly diagnose the problem, these methods therefore tended to incur heavy penalties when it came to area occupied and system performance. Today however we are moving forward to more promising techniques like the ECC hamming codes and current sensors that not only take up less space and power to perform their operations but are able to go one step further and accurately diagnose the location of the error.

Although the goal for such techniques remains delivering 100% reliability while consuming the least amount of power and space, this does not seem to be completely achievable at the moment, since all mitigation techniques prove to have some form of trade off or the other. For now though it seems radiation induced errors are a matter of probability, The FIT rates are low and their effects not so prominent. Thus simplistic mitigation techniques provide satisfactory results and can account for worst case scenarios. However with shrinking circuit size standards it won’t be long before FIT rates begin to strongly affect the performance and reliability of circuits. It is then that mitigation techniques will be needed most and their value truly recognized.

Conclusion

In the present paper a survey was conducted on the phenomenon of radiation induced errors pertaining especially to single event upsets. Three popular industry mitigation techniques TMR, Scrubbing and ECC (Hamming Codes) are surveyed and discussed.

To realize the true extent of the threat caused by SEUs; a Hamming code simulator was designed in VHDL to explore the different possible effects on the system caused by bit flips when

SEUs occur. In this regard data lines, clk lines, data registers and synchronization signals were tested to see how each affects the system. It was also shown how the hamming code can effectively correct an extensive number of SEU's provided that the errors remain single in nature and occur within boundary of its detection mechanism.

The study in this paper is limited due to time constraints as the field of digital system testing and radiation induced errors is extremely extensive in nature requiring familiarity with many aspects in Engineering Physics, Electronic systems and Mathematics and combining the concepts across all these different fields. Many concepts presented in this paper are actually fields or research within themselves and thus cannot be covered in extensive detail in a single paper. Thus this paper serves as a modest review and provides the simulator which substitutes as a useful educational tool for understanding the concepts of SEUs

<u>Future work</u>

- Emulation of the Hamming code simulator on an FPGA based system. This would allow the observation of the system with real delays between registers.
- The testing and simulation of how SEUs and MBUs affect the control registers of an FPGA is another important concept that can be explored and further extended.
- An advanced step would be to emulate the system in a real radiation environment where the system is bombarded by alpha and gamma radiation as is common in simulations in the electronics industry. It would be interesting to see how the software based hamming code system instituted on an FPGA platform would actually function when disturbed by radiation.
- The implementation of the Reed Solomon code algorithms that allows the diagnosis and correction of MBUs in the system is also another avenue that maybe explored.

- Finally this simulator is intended as an educational tool that allows for students to understand the concept of radiation induced errors. It can be extended to other concepts like stuck-at faults and error correction from noise during transmission of data as well.

Bibliography

- [1] D. Binder, E.C. Smith, and A.B. Holman, “*Satellite anomalies from galactic cosmic rays*”,IEEE Trans. Nucl. Sci., vol. NS-22, no. 6, pp. 2675–2680, 1975.

- [2] T.C. May and M.H. Woods, “*A new physical mechanism for soft errors in dynamic memories*”, in Proc. Int’l Rel. Phys. Symp. (IRPS), pp. 33–40, 1978.

- [3] J.F. Ziegler et al., “*IBM experiments in soft fails in computer electronics (1978–1994)*”, IBM, J. Res. Dev., vol. 40, no. 1, pp. 3–18, 1996

- [4] Michael Nicolaidis, “*Soft Errors in Modern Electronic Systems*”, Page 2,5,206 – 209, 222 - 224 Springer Science and Business Media, LLC 2011, Springer New York Heidelberg Dordrecht London

- [5] Baloch, S.; Arslan, T.; Stoica, A.; “*An Efficient Technique for Preventing Single Event Disruptions in Synchronous and Reconfigurable Architectures* “, Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on Publication Year: 2006 , Page(s): 292 - 295

- [6] Dominik, L.;” *System mitigation techniques for single event effects*”, Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th Publication Year: 2008 , Page(s): 5.C.2-1 - 5.C.2-12

- [7] P. K. Lala; ” *Self-Checking and Fault-Tolerant Digital Design*”. San Francisco,CA: Morgan Kaufman Publishers, 2001.

- [8] Quming Zhou; Mohanram, K. “*Cost-effective radiation hardening technique for combinational logic* “,Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on
Digital Object Identifier: 10.1109/ICCAD.2004.1382551
Publication Year: 2004 , Page(s): 100 – 106

- [9] Gericota, M.G.; Lemos, L.F.; Alves, G.R.; Ferreira, J.M.;
“*A Framework for Self-Healing Radiation-Tolerant Implementations on Reconfigurable FPGAs* “,Design and Diagnostics of Electronic Circuits and Systems, 2007. DDECS '07. IEEE Publication Year: 2007 , Page(s): 1 - 6

- [10] Heiner, J.; Sellers, B.; Wirthlin, M.; Kalb, J.;
“*FPGA partial reconfiguration via configuration scrubbing* “,Field Programmable Logic and Applications, 2009. FPL 2009. International Conference ,Publication Year: 2009 , Page(s): 99 - 104

- [11] Mukherjee, S.S.; Emer, J.; Reinhardt, S.K. “*The soft error problem: an architectural perspective*” High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on Digital Object Identifier: 10.1109/HPCA.2005.37

- [12] Chad Thibodeau, “*An Integrated Approach with COTS Creates Rad-Tolerant Single Board Computer for Space*”. COTS Journal Date Published December 2003.

- [13] Jameel Hussein and Gary Swift "*Mitigating Single-Event Upsets*" White Paper 7-Series FPGAs. WP395 (v1.0) April 9, 2012
- [14] Jet Propulsion Laboratory (JPL) California Institute of Technology (2012). "*Reports on radiation effects in Xilinx FPGAs"*. Available. http://www.ieee.org/documents/ieeecitationref.pdf
- [15] Bernard Bancelin and Nicolas Ganry "*Space Industry Looks toward On-the-Fly Reconfiguration"*. COTS Online journal September 2012. http://www.cotsjournalonline.com/articles/view/102932
- [16] Ken Orsquo "*Antifuse FPGA Technology: Best Option for Satellite Applications*". COTS Online Journal December 2003. http://www.cotsjournalonline.com/articles/view/100087
- [17] Courtney Howard "*Aerospace and defense systems integrators continue their reliance on radiation-hardened electronics"*. Military and Aerospace Electronics June 2012. http://www.militaryaerospace.com/articles/2012/06/aerospace-and-defense-systems-integrators-continue-their-reliance-on-radiation-hardened-electronics.html
- [18] Kenneth A. LaBel "*Natural Space Radiation Effects on Technology"*NASA/GSFC Radiation Effects & Analysis Home Page. Available Online. NASA/GSFC Radiation Effects & Analysis Home Page March 2009. http://radhome.gsfc.nasa.gov/radhome/Nat_Space_Rad_Tech.htm