Qualification methodology for ISO26262 certification of automotive SoC systems

by

Tiziano Fiorucci

(Under the Direction of Giorgio di Natale)

Abstract

his thesis proposes to set up a flow and a methodology of ISO26262 certification for system-type integrated circuits on a digital chip dedicated to driving. These circuits are generally composed of several Intellectual Properties, IPs, dedicated to different functions such as communication or processing of information from sensors (camera, lidar ...), real-time system, vision and imaging, system management (operating system), security. The ISO26262 methodology requires the extraction of a number of metrics related to the resilience of the system to single and multiple faults as well as the effectiveness of countermeasures (detection, reporting and correction of errors) and failure modes. The extraction of failure metrics from fault trees is a method known and documented in the literature. Nevertheless, its application has often been limited to macroscopic electromechanical systems such as a car, actuator or sensor chains. On the other hand, these methods are rarely applied in the field of automotive SoCs where the extraction of metrics is still largely manual (usually using a spreadsheet) and dependent on an expert, and where the verification of the effectiveness of countermeasures is best done by targeted fault injection on a few sub-parts of the complete system or irradiation under a particle beam. This thesis proposes to develop a reliability metrics extraction methodology based on fault injection per block as well as composition methods to obtain the metrics at the level of the complete system. The first part of the thesis will be devoted to the study of the bibliography on the construction of fault trees, the ISO26262 standard and the declination of the different reliability metrics in the case of a digital SoCs type system. The extraction of metrics at the block level will be based on 2 different methods, one analytical based on probabilities, the other experimental based on fault injection. The aim is not to develop new probability codes or fault injection tools but to develop a methodology to use them in

the context of a SoC to obtain the desired data. The second part of the thesis will concern the composition of the data obtained at the functional block level in order to obtain the ISO26262 metrics at the system level (SoC). It will be a matter of developing a composition method adapted in particular to the characteristics of SoCs (communicating system, performing calculations that must react in real time, ...) and to the fault models that characterize them or imposed by the ISO26262 standard. The third part of the thesis concerns the application of the developments described in the previous paragraph to an SoC-type system and the verification of the results obtained.

Index words: [PUT YOUR LIST OF INDEX WORDS HERE SEPERATED BY COMMAS]

Qualification methodology for ISO26262 certification of automotive SoC systems

by

Tiziano Fiorucci

B.S., University of Rome "Tor Vergata", 2018
B.Sc., University of Rome "Tor Vergata", 2016

A Dissertation Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the
Degree.

Doctor of Philosophy of Nano Electronics and Nano Technologies

Grenoble, France

2023

Qualification methodology for ISO26262 certification of automotive SoC systems

by

Tiziano Fiorucci

Major Professor:	Giorgio di Natale
Industrial Advisor:	Jean Marc Daveau
Committee:	Prof 1 Prof 2 Prof 3

Electronic Version Approved:

Ron Walcott
Dean of the Graduate School
The University of Georgia
Month Year

Dedication

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

THIS PAGE IS OPTIONAL

Acknowledgments

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

THIS PAGE IS OPTIONAL

Contents

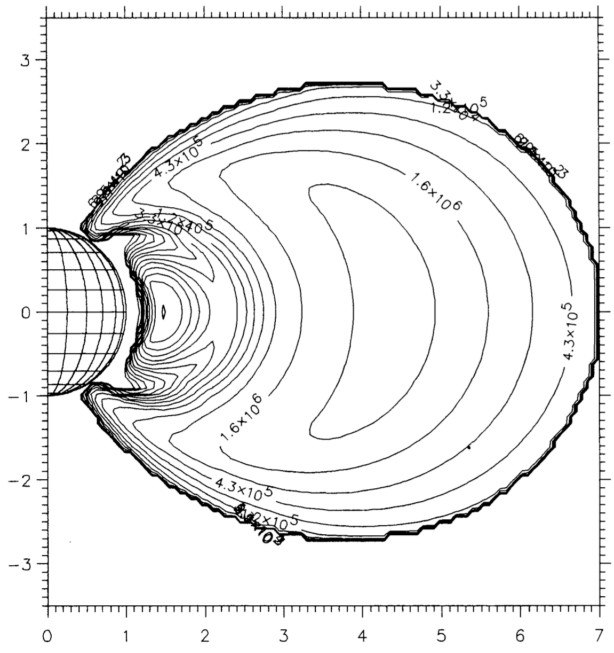
List of Figures

List of Tables

Chapter 1

Problem and its Background

Semiconductor devices and integrated circuits are nowadays operated in a number of hostile environments, therefore it worth to analyze all of them in order to determine what threat show up. Moreover in this chapter it will shown the most common effects on MOSFET based devices as well as the possible architectural solution of the state of the art



1.1 The Space Environment

The Earth and its immediate surroundings are protected by the atmosphere, which acts as a semi-permeable shield letting through light and heat while stopping radiation and UV's; because no such protection is available in space, human beings and electronics (onboard Earth orbiting satellites, space shuttles, space probes) must be able to cope with the resulting set of constraints. Based on several tens of years of this space era, a detailed analysis of the problems on satellites shows that the part due to the radiation environment is significant. It appears that the malfunctions are due to problems linked to the space environment (9 to 21%), electronic problems (6 to 16%), design problems (1 1 to 25%), quality problems (1 to 8%), other problems (11 to 33%) and problems that are still unexplained (19 to 53%) [4]. It is clear that the unexplained problems are either problems linked to the space environment, to the electronics, to the design, or otherwise but the information collected on the ground is generally not sufficient to define the origin of the problem. The space environment is largely responsible for about 20% of the anomalies occurring on satellites and a better knowledge of that environment could only increase the average lifetime of space vehicles.

So the study of the space environment and its couses started, in all its great variety of environments depending on different orbital levels and electromagnetic forces involved. The degradations and disturbances induced by space radiation in the materials and the electronic components are phenomena that have been studied for many years.[4] It resulted in a basic classification of damages, either for Humans and for Electronics that can be easly divided into two groups each:

For Electronics

1. **Cumulative** such as the degradation of thermal control coatings, optics and electronics and the erosion of materials;

2. **Sporadic** such as noises in the detectors and optics, single event effects in highly integrated electronic circuits and electrostatic discharges.

while for humans

1. **Immediate**, permanent or delayed non stochastic effects (destruction or modification of cells), the speed with which the symptoms appear and their seriousness increase in proportion to the exposure to the radiation;

2. **Stocastic** associated with the modifications to the cells whose probability of appearing in the long term increases in proportion to the irradiation (cancers, leukemia, (SET) Program. genetic effects).

1.1.1 Solar Activity

Sun is either a source and modulator of space radiation, its activity can be described using a cyclical model. Each cycle has approximately is 11 years long. In this time span the Sun has 7 years of maximum activity and 4 at its minimum, the transition is considered sharp even though it is indeed continuos. Moreover every 11 years cycle the Sun reverse its magnetic polarity, this leads to an actual 22 years period between two equal configurations.

Usually two main indicators are used to describe the solar activity:

1. F10.7 - 10.7 cm radiation flux
2. Sunspots count - The numbering of sunspot cycles began in 1749 and it is currently near the end of solar cycle 23. The record of F10.7 began part way through solar cycle 18 in the year 1947

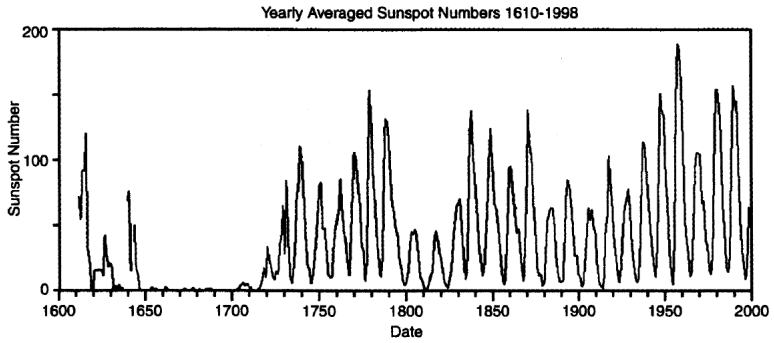


Figure 1.1: The observed record of yearly averaged sunspot numbers [4]

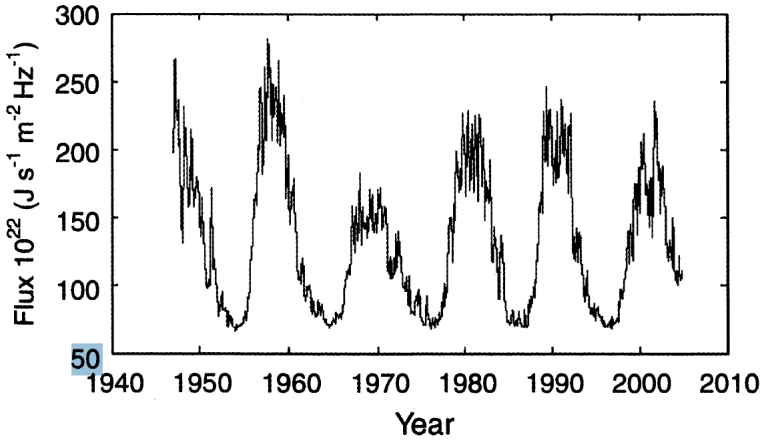


Figure 1.2: Measured values of solar 10.7 cm radio flux [4]

Large solar particle events are known to occur with greater frequency during the declining phase of solar maximum [3]. Trapped electron fluxes also tend to be higher during the declining phase [4]. Trapped proton fluxes in low earth orbit (LEO) reach their maximum during solar minimum but exactly when this peak is reached depends on the particular location [5]. Galactic cosmic ray fluxes are also at a maximum during solar minimum but in addition depend on the magnetic polarity of the sun [6].

1.1.2 Cosmic Rays

With Galactic Cosmic Rays (GCR) it is indended all those higly-charged particles that have been generated outside our solar system, even thought their precise origin is unknown, scientific community believes that Supernovas explosions may be the first source. Some general characteristics of GCR are listed in the following table

Hadron Compositoin	Energy	Flux	Radiation Effects	Metric
87% Protons 12% Alpha 1% Heavy Ions	up to 10^11 GeV	1 to 10 cm^{-2} s^{-1}	SEE	LET

But it is possible to have a deeper look at the relative abundances in 1.3.

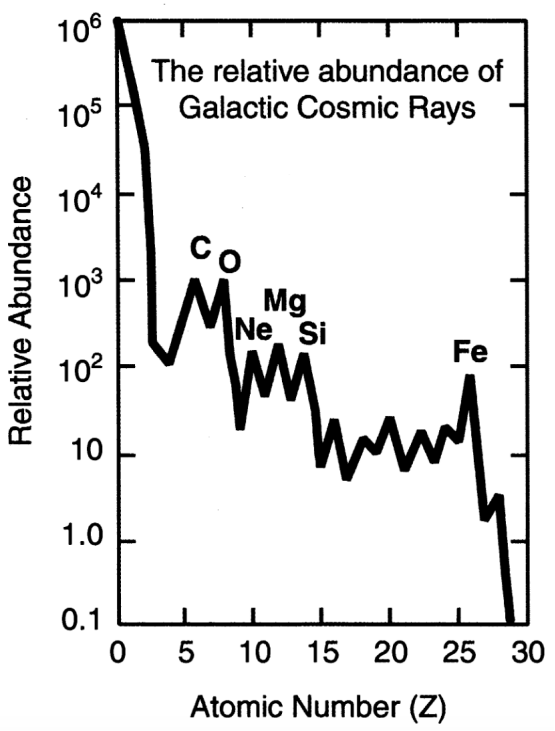


Figure 1.3: Abundances of GCR up through Z = 28

All the elements in the Periodic Table up to Uranium are present in GCR although there is a steep drop-off for atomic numbers higher than iron (Z=26). Their source isn't the only unknown feature of GCR. We saw that they can reach energies up to $10^{11} GeV$, but the couses behind such acceleration are still to be comprehended. On the other hand their flux is really small, limited to a few $cm^{-2}s^{-1}$. Typical GCR energies and fluxes are shown in Fig. 1.4.

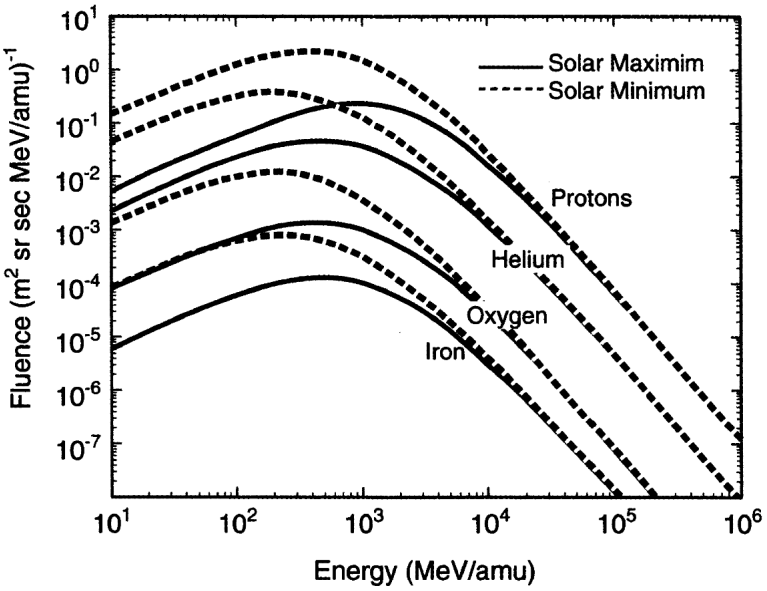


Figure 1.4: GCR energy spectra for protons, helium, oxygen and iron during solar maximum and solar minimum conditions

The peak around $1GeV$ is due to the moderation effect of the solar wind and solar magnetic field, still another expanation for the inverse proportionality of GCR Flux and solar activity. Speaking about the Radiation Effects that these

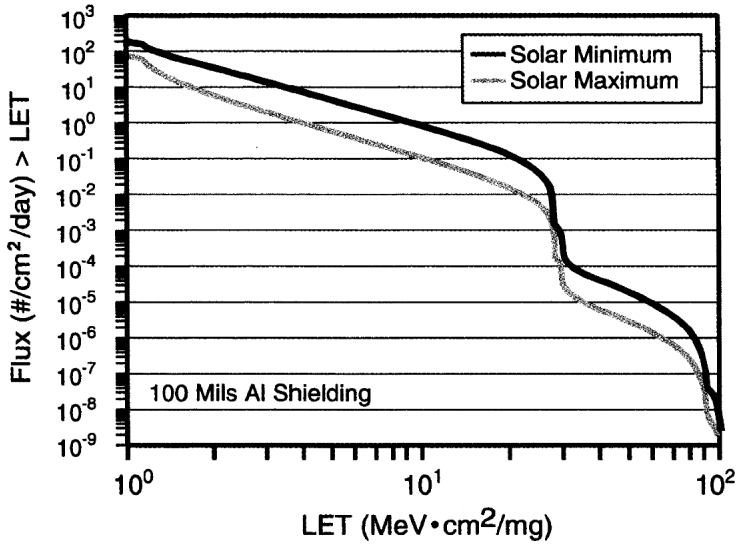


Figure 1.5: Integral LET spectra for GCR during solar maximum and solar minimum

GCR can cause, the main Result of impact is a Single Event Effects (SEE) and the metric to usually utilized to describe the heavy ion induced SEE is the Linear Energy Transfer (LET) which can be defined as the energy lost by the ionizing particle per unit of path lenght in the sensitive section of the device. So it is possible to convert Fig. 1.4 into the spectrum per LET, integrating we can see the difference between the minimum and maximum activity level of the Sun. In the following image all the contributes from all the elements, starting from protons up to uranium, have been considered. The ordinate gives the Flux of particles having a LET above the corresponding abscissa.

The LET Metric can be applied in GEO and Interplanetary missions, in absence of geomagnetic attenuation. For instance, due to basic interaction between charged particles and Earth magnetic field they tend to follow the geomagnetic lines and so parallel to the plant surface at the equator. Thus the energy is mostly deflected away. The Effect of the Geomagnetic field on the incident GCR-LET spectrum during solar minimum is discussed for various orbits in [12]

1.1.3 Radiation Belts

Earth is relatively well protected against external influences such as radiation coming from outern space. In these terms we can imagine the Earth Magnetosphere as the natural cavity in the interplanetary medium that serve to the cause. It is compressed on the solar side and highly extended on the anti-solar side. Poles represent the only space offered to the interplanetary particles to penetrate into the upper atmosphere. Meanwhile the charged particles close to Earth can be trapped by the magnetic field and form the Radiation Belts. As shown in Fig: 1.6 the radiation belts only occupy a limited internal region of the whole magnetosphere. Starting from the closest section to Earth it's possible to identify the upper atmosphere, constant over time. On the opposite end, we cannot really define a boundary due to its strong dependence on solar wind and magnetic field.

In the Earth Magnetosphere we can define the magnetic field as the sum of two contributes:

1. **Main Component** - This term is based on the convection motion in the core of the planet

2. **External Origin** - This includes all the permanent magnets of the terrestrial crust.

In a zero order approximation the field can be considered bipolar. However it is way more accurate to take an off-centered and tilted dipolar magnetic field

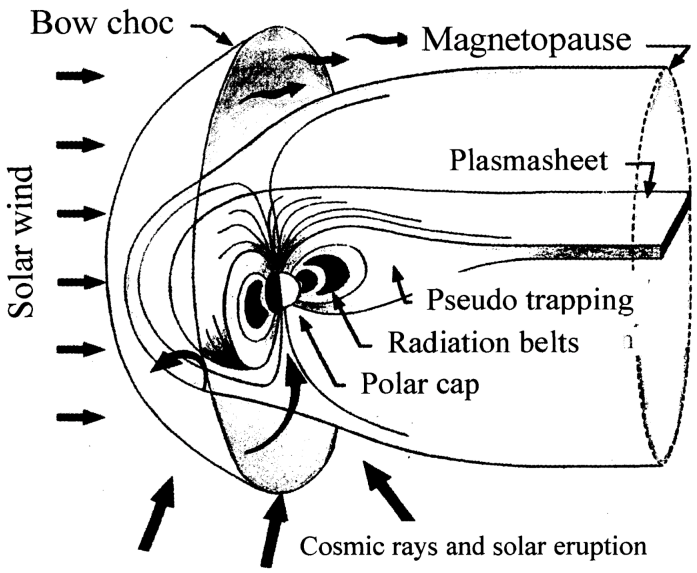


Figure 1.6: Magnetosphere with respect to Radiation Belts [4]

as approximation. This gives us a Dipole not centered in the center of Earth and having its axis not parallel to the earth one. This geometry leads to an anomaly in the magnetic field, a region in which the field is weaker, called the South Atlantic Anomaly, as shown in Fig. 1.7. It is important to observe that the magnetic field on Earth is evolving on a long term basis (secular drift), in particular the South Atlantic Anomaly, which is drifting south-eastwards. As the present time we note:

- a decrease in the intensity of $27\frac{nT}{year}$ (0.05 % a year) [4]
- a drift of the axis, resulting in a westward rotation of the southern end of the dipole (0.014deg a year) and an increase in the shift towards the West Pacific close to 3 km a year. [4]

Dynamics of the charged particles In order to better understand and describe the dynamics of the charged particle in the magnetosphere, we can define a reference system and its coordinates.

- r is the distance from the center of the dipole
- λ is the latitude with θ its colatitude ($\theta = \frac{\pi}{2} - \lambda$)
- ϕ its magnetic longitude

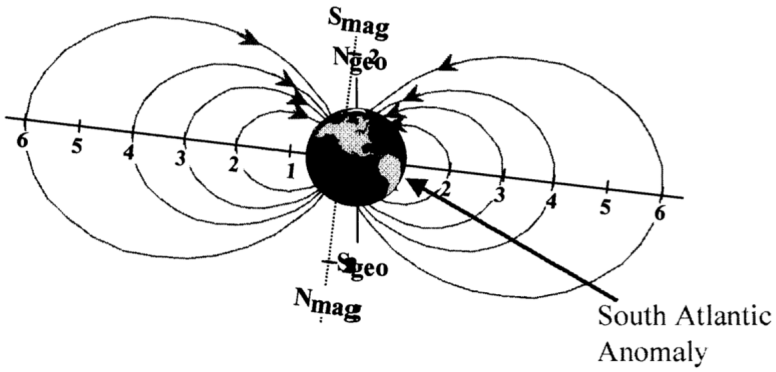


Figure 1.7: Dipolar magnetic field tilted and off-center with respect to Earth. [4]

Last we need to describe the Field lines or force line by the McIlwain parameter L, roughly equal to the distance from the center of the planet to the intersetion point of that force line with the magnetic equatorial plane. So a single point in the field is called B, modulus of the magnetic field.

All charged particles subject to an electromagnetic field will be subject to the Lorentz force

$$F = q(v \wedge B + E) \tag{1.1}$$

Under these conditions, the movement of the high-energy particles can be generally broken down into three basic periodic movements.

Gyration All the charged particles in a magnetic field will rotate around the line of field. This is called Gyration and we can define this movement

1. the Larmor radius $r_L = \frac{mv^2}{qB}$
2. the relativistic magnetic moment $\mu = \frac{mv^2}{2B}$

Bounce If a particle only has a component of its velocity parallel to the magnetic field, then it will move along the field lines. In their motion they keep the magnetic moment μ constant. Since the magnetic moment has to stay constant, while moving from the equator towards the poles, it is possible to notice a strongly increasing magnetic field. It is necessary that the perpendicular component of the speed should increase in order for p to remain constant.[4]

Drift in order to simplify the problem, we place ourselves on the magnetic equator. Since the magnetic field of the planets has a radial gradient, the gyration cannot take place in a constant Larmor radius. Indeed, the magnetic field along a gyration becomes stronger if the particle approaches the planet, the Larmor radius is then smaller and therefore the radius of the trajectory's curve is also smaller. The particle will thus be able to move away from the planet, the magnetic field will be weaker and therefore the Larmor radius and the radius of the trajectory's curve will be greater. The particle therefore does not go through a simple circle but along a more complex trajectory. This movement breaks down into a simple gyration (circular) and a rotation movement around the planet: this is the drift movement.

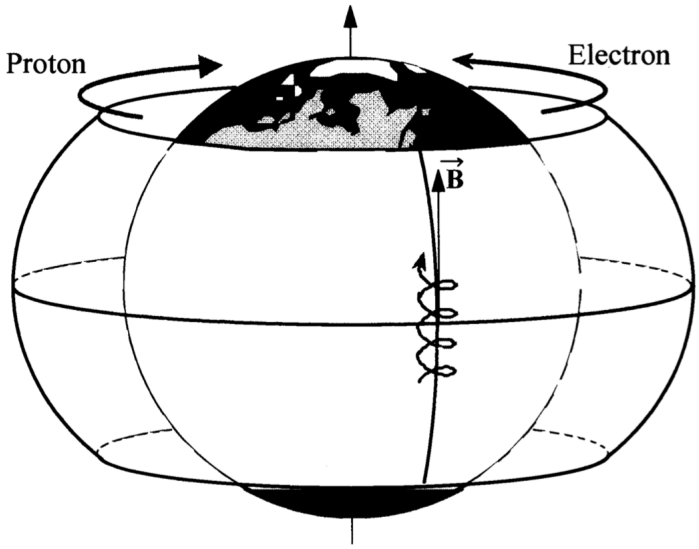


Figure 1.8: Composition of a charged particle's three periodic movements: gyration, bounce and drift. The particle then follows a torus surface called a drift shell. [4]

A charged particle submitted to these three basic [81] and periodic movements then moves through torus shaped surfaces around the Earth, which are commonly called drift shells Fig 1.8. The periods associated with each of these basic movements for a 3 MeV electron at L=3 are respectively 2.1410^{-4} s, 0.19 s and 504 s. The disparity between the periods is very great, a factor of the order of 1000 should be noted between each of them going from the gyration movement to the drift movement.

Due to the magnetic field in proximity of Earth make all the relativistic charged particles to remain trapped in a quasi periodic movement. These conditions are perfect to start an increasing high energy charged particles ensables,

creating the so called radiation Belts. Given the previously presented trajectories, the Radiation Belts assume a toroidal shape surrounding the Earth. The atmosphere is the lower bound while the outern limit is not well defined and may be time variable.

During the first space missions, J. Van Allen has discovered that mostly all the trapped particles are Protons and Electrons, having an Energy range between come KeV and hundreds of MeV. Below there is a representation of the Proton belt (Fig.1.9), pretty stable and constant in time, with energies from some MeV to hundreds of MeV.

On the other hand, the electron belt is more complex (Fig. 1.10) and has two maximums respectively corresponding to the internal and external zones: - the first one centered on L = 1.4 extends up to L = 2.8; the electron populations are relatively stable there and can reach maximum energy levels of the order of 10 or even 30 MeV; - the second one, centered on L = 5, extends from L = 2.8 to L = 10; the electron flows there are much more variable and the energy levels can be as high as 7 MeV.

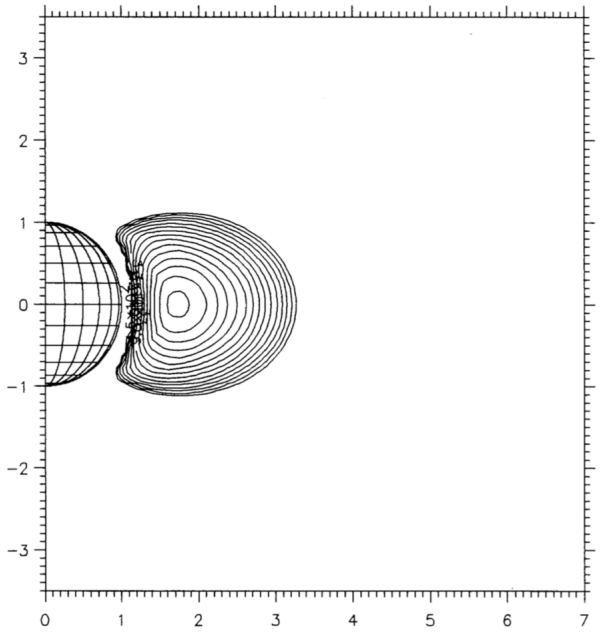


Figure 1.9: Proton radiation belt [4]

Dynamic of the radition Belts Starting from the 1990s the American satellite CRRES has shown te extreme dynamics of protons and electrons trapped in the radiation belts. In fact the population of these particles is strongly dependent on two factors:

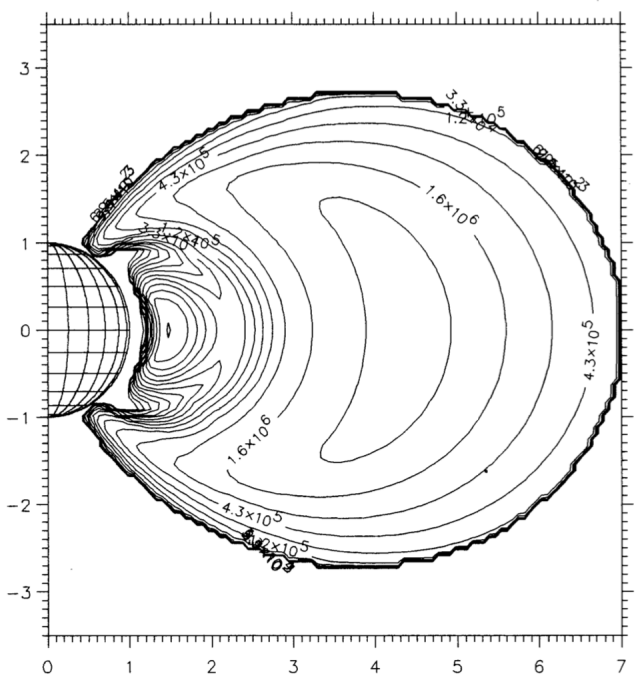


Figure 1.10: Electron radiation belt. [4]

1. The Sources - injections from the tail of the magnetosphere and creations by nuclear reactions

2. The Losses - Precipitations in the upper atmosphere or by charge exchange with particles from the Exosphere.

Dynamics on the scale of the Solar cycle-Protons The radiation belt of protons having high energies, more than 10 MeV varies slowly as function of the solar cycle, as shown in Fig:1.11. The flux levels oscillates around its maximum at the minimum activity of the Sun and vice versa. This is the actual result of two different phenomena, the absorbtion of the protons by the upper atmosphere and the modulation of CRAND (Cosmic Ray Albedo Neutron Decay) source. This balance is shown in Fig:1.11.

Dynamics on the scale of the Solar Cycle-Electrons As well as for the proton cycle, the electrons, especially in the geostationary orbit, follow a similar trend. Inversely proportional to the Sun activity cycle, as shown in Fig:1.12

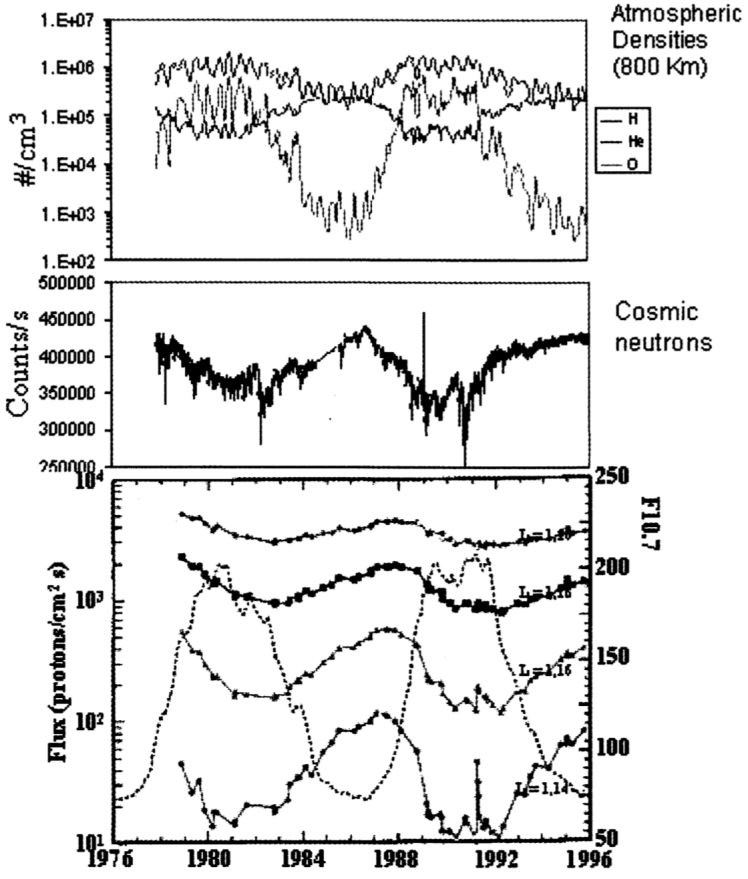


Figure 1.11: Changes in the proton fluxes at low altitudes (bottom), in the cosmic radiation (middle) and atmospheric densities (top) as a function of the solar cycle. [4]

1.2 The Earth Environment

Since the 1984, the existence and impact of atmospheric neutrons has been predicted. They can cause Single Event Upset in the electronics, the first actually measured was in 1992. After that several hundreds have been observed. As we can see in Fig. 1.13 Cosmic rays cover a large spectrum of energies, with a comparatively high flux in the 100 MeV to 10 GeV range and a peak around 500 MeV; cosmic particles collide with the nuclei of atoms making up the Earth atmosphere and initiate the so-called air showers, producing particles such as neutrons, protons, muons, pions, electrons and gamma-rays.

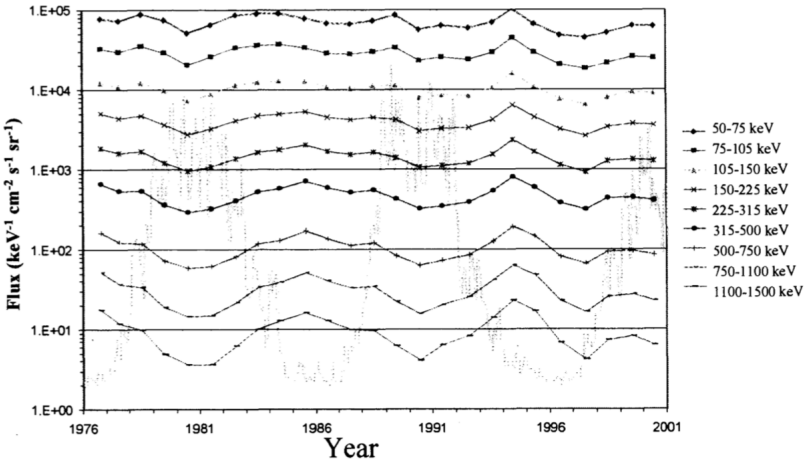


Figure 1.12: Electron fluxes at geostationary orbit as a function of the solar cycle. [4]

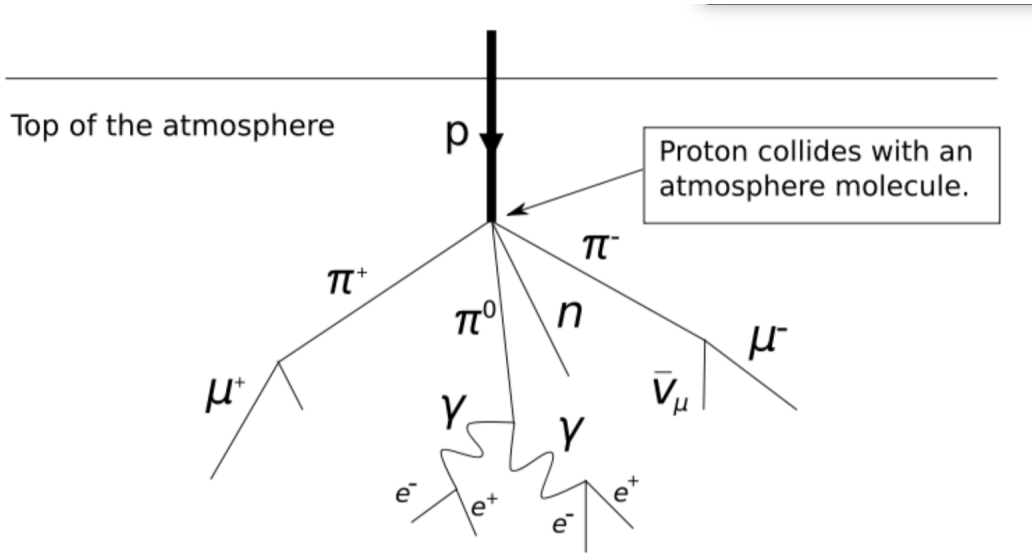


Figure 1.13: Example of Proton Scattering [19]

A deeper analysis of the particles at commercial flight level shows a great majority of neutrons, while protons play a minor role in the Single Event Upset at those altitudes.

Moreover, even though the scheme reported in Fig. 1.13 seems to be a top down shower, the neutron flux is, in fact, isotropic.

1.3 Military Environment

In case of an explosion occurring above the Earth atmosphere, two possible scenarios may be considered

- Aerospace systems operating at altitudes higher than 50-100 km will be directly submitted to the radiation emitted by the weapon; hard X-rays, gamma-rays and neutrons all have a significant impact on the electronic systems onboard satellites
- The main indirect effect to be considered is that related to trapping by the Earth magnetic field lines of electrons from fission debris, resulting in the formation of highly stable, artificial radiation belts which can deliver much higher radiation doses to satellites than natural belts; the first satellite failure due to radiation effects dates back to 1963, to the Starfish test (a 1.4 Mton thermonuclear bomb detonated at an altitude of 400 km); the test produced an intense radiation belt destroying seven satellites over seven months, primarily because of dose effects on their solar panels; the TELSTAR satellite, launched on July 10, 1962, broke down in February 1963

A nuclear explosion in the Earth atmosphere, besides a variety of mechanical effects, is responsible for different purely radioactive effects, which can be split into two main categories

- **Initial nuclear radiation (INR)** is that released within less than a minute after detonation; X-rays are quickly stopped (at about 500 m above sea) so that gamma-rays (responsible for ionizing dose effects) and neutrons remain
- **Residual nuclear radiation (RNR)** features several radiation sources; fission products, radioactivity of debris (neutron-activated weapon materials), that of unfissioned uranium and/or plutonium and activation of the environment; a further distinction can be made between local fallout, occurring less than 24 hours after explosion, with a resulting significant level of ground radiation, and worldwide fallout, which can take place at significant distances from the place of explosion

1.4 Radiation Effects on COTS Components

1.4.1 Basic Damage Mechanism in Semiconductor Devices

Even if we take into consideration the great diversity of particles and their interaction with different technologies, it is possible to distinguish just two main categories of damaging mechanism for semiconductor devices

1. Ionization Damage
2. Displacement Damage

Ionization Damage

The ionization takes place when energy deposited in a semiconductor or in insulating layers, chiefly SiO2, frees charge carriers (electron-hole pairs), which diffuse or drift to other locations where they may get trapped, leading to unintended concentrations of charge and parasitic fields; this kind of damage is the primary effect of exposure to X- and gamma-rays and charged particles; it affects mainly devices based on surface conduction (e.g. MOSFETs)[19]. Directly ionizing radiation consists of charged particles. Such particles include energetic electrons (sometimes called negatrons), positrons, protons, alpha particles, charged mesons, muons and heavy ions (ionized atoms). This type of ionizing radiation interacts with matter primarily through the Coulomb force, repelling or attracting electrons from atoms and molecules by virtue of their charges. Indirectly ionizing radiation consists of uncharged particles. The most common kinds of indirectly ionizing radiation are photons above 10 keV (x rays and gamma rays) and all neutrons. X-ray and gamma-ray photons interact with matter and cause ionization in at least three different ways:

- Lower-energy photons interact mostly via the photoelectric effect, in which the photon gives all of its energy to an electron, which then leaves the atom or molecule. The photon disappears.
- Intermediate-energy photons mostly interact through the Compton effect, in which the photon and an electron essentially collide as particles. The photon continues in a new direction with reduced energy while the released electron goes off with the remainder of the incoming energy (less the electron's binding energy to the atom or molecule).
- Pair production is possible only for photons with energy in excess of 1.02 MeV. However, near 1.02 MeV, the Compton effect still dominates: pair

production dominates at higher energies. The photon disappears and an electron-positron pair appears in its place (this occurs only in the vicinity of a nucleus because of conservation of momentum and energy considerations). The total kinetic energy of the electron-positron pair is equal to the energy of the photon less the sum of the rest-mass energies of the electron and positron (1.02 MeV). These energetic electrons and positrons then proceed as directly ionizing radiation. As it loses kinetic energy, a positron will eventually encounter an electron, and the particles will annihilate each other. Two (usually) 0.511 MeV photons are then emitted from the annihilation site at 180 degrees from each other. For a given photon any of these can occur, except that pair production is possible only for photons with energy greater than 1.022 MeV. The energy of the photon and the material with which it interacts determine which interaction is the most likely to occur[30].

About indirect ionization, we can say that light particles such as protons and neutrons could have not enough LET to cause upset, but they can cause nuclear reactions that in turn create heavier particles that can cause upsets by direct ionization. Secondary reaction products have much higher LET but shorter ranges and lower energies[31].

Displacement Damage

Incident radiation dislodges atoms from their lattice site, the resulting defects altering the electronic properties of the crystal; this is the primary mechanism of device degradation for high energy neutron irradiation, although a certain amount of atomic displacement may be determined by charged particles (including Compton secondary electrons); Displacement Damage mainly affects devices based on bulk conduction (e.g. BJTs, diodes, JFETs) [19]. Displacement damage occurs when sufficient energy is transferred from an incident energetic particle to a lattice atom to dislodge it from its normal location. Using Si as an example, the Si atom initially displaced by an incoming particle is known as the primary knock-on atom (PKA) or the primary recoil.The PKA, or recoil, carries a net charge that depends on its kinetic energy. Displacement damage occurs through the interaction of incident particles with Si atoms by any of the following three processes:

- Rutherford (i.e., Coulomb) scattering
- Nuclear elastic scattering
- Nuclear inelastic scattering

Once any of those basic interaction processes produces a PKA, that ion subsequently can introduce further displacement damage by Rutherford and nuclear scattering. Lattice defects are produced by PKAs and any later-generation energetic recoils that they create. When defects produced by incident radiation are relatively far apart, they are known as isolated, or point, defects. As an example, isolated defects are created by 1 MeV electrons incident on Si. Radiation-induced defects may also be created closely together and form local regions of disorder known as defect clusters. For example, incident 1 MeV neutrons produce both isolated and clustered defects in Si. In general, energetic particles incident on semiconductors create either isolated and clustered defects or solely isolated defects, depending on the mass and energy of the incident particles. Nearly all the effects of displacement damage on the electrical and optical properties of semiconductor materials and devices can be understood in terms of energy levels introduced in the bandgap. Those radiation-induced levels result in the following effects: recombination lifetime and diffusion length are reduced; generation lifetime decreases; majority and minority-carrier trapping increase; majority carrier concentration changes; thermal generation of electron-hole pairs is enhanced in the presence of a sufficiently high electric field; tunnelling at junctions is enabled. In addition, radiation-induced defects reduce the carrier mobility and can exhibit metastable configurations[**bib10**].

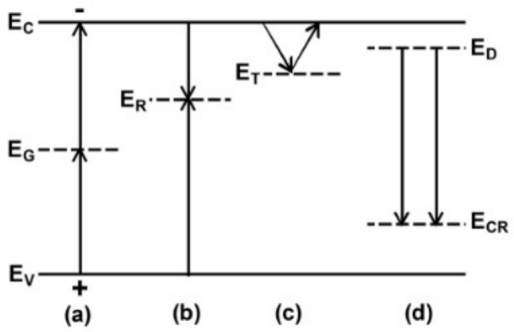


Figure 1.14: Radiaton-Induced energy

Figure 1.14 illustrates radiation-induced energy levels in the Si bandgap that give rise to the following processes: (a) enhanced thermal generation; (b) enhanced recombination; (c) enhanced temporary trapping; (d) reduced carrier concentration due, in this example, to the introduction of centers that compensate for donors (carrier removal).

1.4.2 Radiation Effects in Electronics

As previously stated, the effects of radiation on semiconductor devices can be divided into two broad classes. Previously the Total Dose effects have been explained, now the focus will be moved to Single Event Effects.

Single Event Effect

These effects are due to the deposition of charge by a single particle that goes through a sensitive region of the device. This can lead to a desctructive or non destructive damage of the device. Moreover it is possible to identify a couple of differences between Total Dose effects and Single Event Effects:

- Single Event Effects are Stocastical, while TID effects are cumulative and may occur after the device has been exposed to radiation for a long time
- TID are related to Long Term response, while SEE to Short term Response
- Only a very limited portion of the device is affected by SEE, while TID affects uniformly the entire device, this is due to the number of particles hitting the device and their distribution.
- While for TID the main figure is the drift of the main device parameters, concerning SEE the most important figure is the Rate of Occurrence.

We can so define this effects some of the SEE as "Soft" in case they do not induce any physical damage to the device, but just an information loss. Otherwise they are categorized as "hard" in case the impact of a heavy ion is followed by the rupture of the gate oxide. Here there is a list of the major Soft effects and Hard effects

Main Classes of Soft Effects

- Single Event Upset (SEU) - the corruption of a single bit in a memory array
- Multiple Bit Upset (MBU) - the corruption of multiple bits due to a single particle
- Single Event Transient (SET) - a transient signal induced by an ionizing particle in a combinatorial or analog part of a circuit

Main classes of hard effects

- Single Event Gate Rupture (SEGR) - rupture of gate oxide occurring especially in power MOSFETs
- Single Event Burnout (SEB) - burnout of a power device
- Single Event Latch-Up (SEL) - the activation of parasitic bipolar structures, leading to a sudden increase of the supply current

Usually in order to evaluate the occurrence of SEE the cross section of the device is used. It is defined as follow:

$$\sigma_{SEE} = \frac{NumberOfEvents}{ParticleFluence} \tag{1.2}$$

the Cross Sections varies as function of the LET of the particle hitting the device, observable only if higher than the threshold LET. The charged released by the particle hitting the transistor is collected via the so called Funneling mechanism. Most of the charge is sucked in at the struck junction through a deformation of the junction potential, while the remaining charge diffuses in the substrate and may be collected or not at the same junction[19]

The aim of these thesis is to focus on Soft Effects, in the next section the most common SEE will be analyzed in details.

Radiation Effects in MOSFETs

Single Event Upset It is obvious that in order to cause disturbance in any circuit the charge generated by a particle hitting the device must be in a sensitive node; in particular reverse biased PN junctions are the most affected by collected charge, having a larger deplation reagion and stronger electric field. Taking into account the case of an SRAM cell, may be the case of a particle hitting the Drain of the off NMOSFET. If that is the case the released charge is collected by the reverse-biased drain, the voltage at the struck node tends to decrease, turning the radiation-induced current in to a voltage transient. The curren decreases the potential at the node, and it may go as low as below the switching voltage, changing the initial state.

These effects are function of the LET of the impinging particle and the incident angle θ

Multiple Bit Upset Single events effects have become more complex to study as the new technologies are released. In particular the minimum lentgh that can be obtained while creating CMOS devices in lithography has gone below the

micron realm. Nowadays the size of the path of the hitting particle has become comparable to the size of modern chips. Therefore that in the past may have involved a single point in the circuit now involve multiple nodes and charge sharing may occur. It follows that the rate of occurrence of Multiple Bit Upset is strongly bound to rise as fabrication process evolve.

Radiation Induce Latch-Up A latch-up is a type of short circuit which can occur in an integrated circuit. More specifically it is the inadvertent creation of a low-impedance path between the power supply rails of a MOSFET circuit, triggering a parasitic structure which disrupts proper functioning of the part, possibly even leading to its destruction due to overcurrent. A power cycle is required to correct this situation.

Single event latch-up is a latch-up caused by a single event upset, typically heavy ions or protons from cosmic rays or solar flares. The parasitic

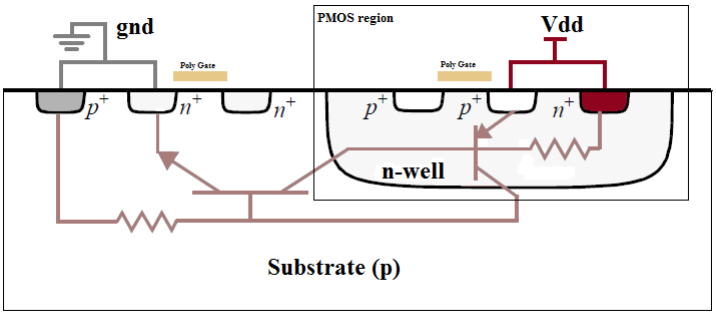


Figure 1.15: Latch-up Diagram

structure is usually equivalent to a thyristor (or SCR), a PNPN structure which acts as a PNP and an NPN transistor stacked next to each other. During a latch-up when one of the transistors is conducting, the other one begins conducting too. They both keep each other in saturation for as long as the structure is forward-biased and some current flows through it - which usually means until a power-down. The SCR parasitic structure is formed as a part of the totem-pole PMOS and NMOS transistor pair on the output drivers of the gates.

1.5 Types of redundant architectures

While there are various methods to implement redundant architectures, techniques and terminologies, the following section wants to represent the most common ones used in the industry.

1.5.1 Standby Redundancy

Standby redundancy, also known as Backup Redundancy is when you have an identical secondary unit to back up the primary unit. The secondary unit typically does not monitor the system, but is there just as a spare. The standby unit is not usually kept in sync with the primary unit, so it must reconcile its input and output signals on takeover of the Device Under Control (DUC). This approach does lend itself to give a "bump" on transfer, meaning the secondary may send control signals to the DUC that are not in sync with the last control signals that came from the primary unit. You also need a third party to be the watchdog, which monitors the system to decide when a switchover condition is met and command the system to switch control to the standby unit and a voter, which is the component that decides when to switch over and which unit is given control of the DUC. The system cost increase for this type of redundancy is usually about 2X or less depending on your software development costs. In Standby redundancy there are two basic types, Cold Standby and Hot Standby. [**bib12**]

Cold Standby Redundancy

In cold standby, the secondary unit is powered off, this is preserving the reliability of the unit. The drawback with respect to the hot standby is the longer downtime needed to switch from one unit to the secondary one. this makes it more challenging from the synchronization issues point of view.

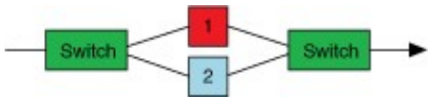


Figure 1.16: Cold Standby scheme

Hot Standby Redundancy

In hot standby instead, the secondary unit is always powered on and can eventually monitor the DUC. If the secondary unit is used as watchdog or voter to decide when to switch over, it is possible to eliminate the need for a third party unit to perform these operations. It is also possible to notice that some versions of the Hot standby are similar to the Dual Modular Redundancy (DMR) or Parallel Redundancy.

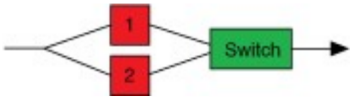


Figure 1.17: Hot Standby Schema

1.5.2 N-Modular Redundancy

N Modular Redundancy, also known as Parallel Redundancy, refers to the approach of having multiply units running in parallel. All units are highly synchronized and receive the same input information at the same time. Their output values are then compared and a voter decides which output values should be used. This model easily provides bumpless switchovers. This model typically has faster switchover times than Hot Standby models, thus the system availability is very high, but because all the units are powered up and actively engaged with the DUC, the system is at more risk of encountering a common mode failure across all the units.

Deciding which unit is correct can be challenging if you only have two units. Sometimes you just have to choose which one you are going to trust the most and it can get complicated. If you have more than two units the problem is simpler, usually the majority wins or the two that agree win. In N Modular Redundancy, there are three main typologies: Dual Modular Redundancy, Triple Modular Redundancy, and Quadruple Redundancy.

Dual Modular Redundancy

Dual Modular Redundancy or DMR uses two identical and so functional equivalent units, both of them able to control the DUC. The most challenging side of this configuration is the switching decision between the two units. Since both of them are monitoring the DUC there is the need for a routine in case of mismatch between the two units. In is possible to create a tiebraker ore even designate the second as default winner, assuming it is more trustworty than the primary unit. The average cost increase of a DMR system is about twice that of a non-redundant system, factoring in the cost of the additional hardware and the extra software development time.

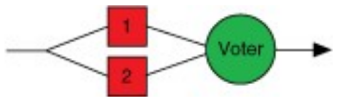


Figure 1.18: DMR scheme

Triple Modular Redundancy

Triple Modular Redundancy (TMR) uses three functionally equivalent units to provide redundant backup. This approach is very common in aerospace applications where the cost of failure is extremely high.

TMR is more reliable than DMR due to two main features. The most immediate is that there are two "standby" units instead of a single one. The second is that in TMR it common to see the so called diversity platforms or diversity programming techniques applied. in these techniques it is possible to notice the use of different hardware or software platforms.

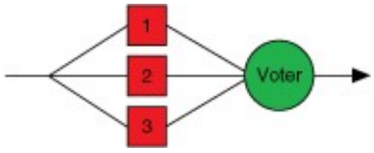


Figure 1.19: TMR Schema

Quadruple

Quadruple Modular Redundancy (QMR) is fundamentally similar to TMR but using four units instead of three to increase the reliability. The obvious drawback is the 4X increase in system cost.

1.5.3 1:N Redundancy

This design technique is used in case the system has a single backup for multiple modules and this backup is able to act as any of the single ones. This technique offers a redundancy at much lower costs than the others. This approach only works well when the primary units all have very similar functions, thus allowing the standby to back up any of the primary units if one of them fails.

Other drawbacks of this approach are the added complexity of deciding when to switch and of a switch matrix that can reroute the signals correctly and efficiently.

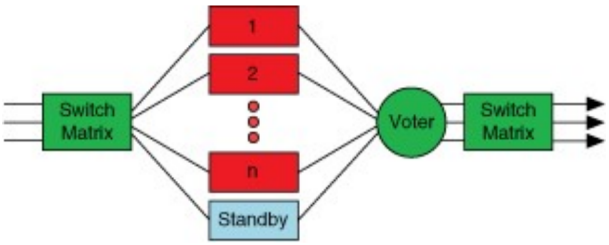


Figure 1.20: Example of Proton Scattering [19]

1.5.4 Redundancy Improves Reliability

Reliability is defined as the probability of not failing in a particular environment for a particular mission time. Reliability is a statistical probability and there are no absolutes or guarantees. The goal is to increase the odds of success as much as you can within reason.

The following equation is the most common to calculate reliability, and it assumes that the system has a constant failure rate λ

$$R(t) = e^{-\lambda t} \tag{1.3}$$

in which:

- $R(t)$ is the probability of success
- t is the mission time or the time the system has to execute without an outage
- λ is the constant failure rate over time (N Failures per hour)
- $\frac{1}{\lambda}$ is the MTTF or mean time to failure

In fact one way to calculate reliability is to take the probability equation and instead solve for the mean time to failure (MTTF) of the system:

$$R(t) = e^{-\lambda t} = e^{-\frac{t}{MTTF}} \tag{1.4}$$

solving for MTTF

$$MTTF = -\left(\frac{t}{\ln\left[R(t)\right]}\right) \tag{1.5}$$

For example, if your application had a mission time of 24 hours a day, 7 days a week, for one year (24/7/365) and you experienced a success rate of 90%:

$$MTTF = -\left(\frac{1yr}{\ln[.90]}\right) = 9.49years \tag{1.6}$$

in case redundancy has been added to the system we can, for istance, increase the success rate to 99% for the same mission time and therefore:

$$MTTF = -\left(\frac{1yr}{\ln[.99]}\right) = 99.50years \tag{1.7}$$

These equations effectively demonstrate the vast improvement in reliability that redundancy can bring to any system.

In particular, it is usefull for this thesis to better understand the real andvantages of TMR over the Simplex model. First we have to set some assumptions:

1. TMR only works if there are at least 2 working modules
2. R_m is the Reliability of the single module
3. R_v is the Reliability of the Voter

That said, it is possible to calculate the Reliability of a TMR system as follow:

$$R_{TMR} = R_v \sum_{i=2}^{3} \binom{3}{i} R_m^i (1 - R_m)^{3-i} \tag{1.8}$$

and so

$$R_v[R_m^3 + 3R_m^2(1 - R_m)] = R_v(3R_m^2 - 2R_m^3) \tag{1.9}$$

from which it is possible to evaluate the MTTF for the same system as:

$$MTTF_{TMR} = \int_0^\infty R_{TMR} dt = \int_0^\infty R_v(3R_m^2 - 2R_m^3) dt \tag{1.10}$$

$$\int_0^\infty e^{-\lambda_v t} \left(3e^{-2\lambda_m t} - 2e^{-3\lambda_m t}\right) dt = \frac{3}{2\lambda_m + \lambda_v} - \frac{3}{3\lambda_m + \lambda_v} \tag{1.11}$$

It is possible to neglect the failure rate of the voter since it is usually designed to be way lower than the module one.

Now comparing the $MTTF_{TMR}$ with the Simplex solution we obtain:

$$MTTF_{TMR} = \frac{3}{2\lambda_m} - \frac{2}{3\lambda_m} = \left(\frac{5}{6}\right)\left(\frac{1}{\lambda_m}\right) = \frac{5}{6} MTTF_{Simplex} \tag{1.12}$$

Chapter 2

Failure Mode and Effect Analysis FMEA

2.1 Introduction

2.1.1 General

FMEA and FMECA are important techniques for a reliability assurance programme.They can be applied to a wide range of problems which may occur in technical systems, and can be carried out in varying degrees of depth, or modified, to suit a particular purpose. The analysis is carried out in a limited way during the conception, planning, and definition phases and more fully in the design and development phase. It is however important to remember that the FMEA is only part of a reliability and maintainability programme which requires many different tasks and activities. FMEA is an inductive method of performing a qualitative system reliability or safety analysis from a low to a high level. A thorough understanding of the system under analysis is essential prior to undertaking FMEA. Functional diagrams and other system drawings are normally necessary for this understanding. Reliability block diagrams, fault trees and/or state diagrams are then usually derived from these in order to carry out the analysis. In many instances the block diagram descriptions and block diagram failure descriptions are included in the FMEA format. Separate diagrarns will be needed for the following:

1. The way in which different criteria for system faiulre are determined;
2. Degradation of function or reduction in assurance of function;
3. Alternative operational phases

2.1.2 Purpose of the Analysis

The reasons for undertaking FMEA (or FMECA) may include the following:

- to identify those failures which have unwanted effects on system operation, e.g. safety critical failures;
- to satisfy contractual conditions that an FMEA should be completed;
- where appropriate, to quantify the reliability and/or safety of the system;
- to allow improvements of the system's reliability and/or safety (e.g. by design or quality assurance action)
- to produce aids to fault diagnosis;
- to allow improvement of the system's maintâinability (by highlighting areas of risk or non-conformance for maintainability).

ln view of these reasons the objectives of an FMEA (or FMECA) may include the following:

1. a comprehensive identification and evaluation of all the unwanted effects within the defined boundaries of the system being analysed, and the sequences of events brought about by each identified item failure mode, from whatever cause, at various levels of the system's functional hierarchy;
2. the determination of the significance (or criticality) of each failure mode with respect to the system's correct function or performance and the impact on the reliability and/or safety of the process concerned;
3. a classification of identified failure modes according to relevant characteristics, including detectability, diagnosability, testability, item replaceability, compensating and operating provisions (repair, maintenance, logistics, etp.);
4. an estimation of measures of the significance and probability of failure.

2.1.3 Basic Principles of FMEA

The following concepts are essential to FMEA:

1. breakdown of the system into 'elements';

2. a diagram of the system's functional structure and identification of the various data which are needed to perform the FMEA;

3. the failure mode concept (a part may have several failure modes or a failure mode may involve several parts);

4. identification of new physical features or new requirements;

5. the criticality concept and the measure to be used (if criticality analysis is required).

Further, it is essential to specify the existing links between the FMEA (and the FMECA) and other qualitative (and quantitative) analytical methods within the overall reliability programme. Very few designs are wholly new. Most are to some extent developments of old designs. FMEA should use the information on existing systems and draw attention to the need for tests, etc. for the new parts.

Chapter 3

Hardware Contribution

3.1 Introduction

In the context of new applications for autonomous mobility, digital components a required to reach high levels of functional safety performances. This level of assurance is necessary to supply safely the computational power and advanced processing required by those applications. It is therefore necessary for digital SoCs safety engineers to be able to demonstrate thru advanced provable methods the achieved reliability of their system and counter measures.

Similarly to complex electomecanical systems, it is difficult to predict the failure modes of a complex SoC which exhibits an almost infinite state space (in the order of 2^n, n is the number of sequential elements, reaching ten's of thousands easily) and distributed-systems characteristics: numerous independents sub-systems operating and communicating asynchronously.

Digital systems are subject to two kind of errors, *permanent* which are created by destructive or aging effects, and *transient* [1] created by particle impacts such as thermal neutrons at ground level or solar wind in low and high earth orbits [39][28]. Permanent effects shows in the form of a permanent stuck to an electrical value ('0' or '1' logic value) and can occurs on any digital element (combinational logic, i.e. *logic gate* or sequential element, i.e. *flip-flops*). So do transient faults, also called *soft errors*, but with different, non-permanent, effects on logic or sequential elements. Transient faults on logic gates are called *Single Event Transient* (SET) [16] and are particularly dangerous on clock trees and reset trees (which distributes the clock and reset signals through the chip using trees of *buffers*) of SoCs as their effect, that has the form of a glitch, is to reset or desynchronize the sequencing of a sub-part of the system. Transient faults on sequential elements (memories or flip-flops) only invert the value of the element which will retain the faulty value until overwritten by a new value.

They are called *Single Event Upset* (SEU) and are the main cause of safety goals violations in digital SoCs [25].

However, digital system exhibit a natural resistance to soft errors and most of them have no functional effect while a small proportion of them ($\approx 10\%$ of them in a standard 5-stages processor [34, 26]) will lead to system execution failure. FMEDA analysis [22], targeting goals such as ISO26262 automotive safety norm [**iso26262**] certification will consist in quantifying those failure modes, proving the effectiveness of counter measures and absence of safety goals violation. An effective solution consists in submitting the system to faults, by simulation or under radiation beam, therefore stressing it and provoking intentionally dysfunctional behaviors. Those 'out of trajectory' behaviors can then be recorded, analysed and used for FMEDA analysis in the certification process. However, both methods are costly both in term of engineering setup needed and cost: fault injection of a full SoC requires a complex setup, test suite and costly hardware emulator while radiation test requires an acquisition system, a test setup and access to costly and constrained radiation facilities, Both have the disadvantage to require, the full SoC gate netlist (fault injection [38]) or silicon (irradiation [3]). Also, both methods can be classified as experimental as it is a verification 'by observation' of the resilience of the system to faults. No proof, except statistical confidence is made on the extracted faults metrics.

In this work, we aim to assess the capability of Model-Based Safety Assessment methods to build the dysfunctional model of a digital SoC from its subsystems and perform the currently hand-made FMEDA of the full system automatically. We expect the methods to be able to quantify globally the system safety metrics more accurately than with hand-made spreadsheets which only basically multiply probabilities. Automatic failure analysis such as fault trees extraction, fault sequences leading to unwanted events are also expected to be of great help during the certification process. The problem to solve is then to extract and build the required dysfunctional models of the different subsystems of the SoC and to properly expose the failure modes in the constructed models to be able to use existing model composition frameworks.

The document is organized as follow: we first present the system used as example and how fault injection is used to expose dysfunctional behaviors and extract a model. The chosen approach is then detailed reminding generic principles before explaining specific mechanisms put in place to model digital system. Finally, the document details fault injection campaign post-processing methods and obtained results. We compare composition results with fault injection performed on the full system used as a reference.

3.2 State-of-the-Art

3.2.1 Probabilistic Methods in Digital Systems Safety

Probabilistic methods [**5724504**] [**Torras**] have been developed to estimate propagation and masking rates of errors in gate netlists. Such approaches, restricted to combinational logic provide an helper to estimate certain metrics (λ_{spf}, i.e. *Dangerous Undetected* by a safety mechanism faults [**iso26262-acronyms**]) required in ISO26262, but are far from being able to provide metrics even at the sequential block level. Likewise, industrial formal proof tools [**jaspergold**] [**yeung-2018**] are able to compute such metrics by using formal methods.

Methods like FIDES [**fides**] [**FIDES_fault_tree**] targets Commercial Off-the-Shelf (COTS) based Electronic Control Unit (ECU), with components failure rates extracted from available reliability databases. It takes into account systematic or aging failures but not transient effects such as soft-errors.

3.2.2 Formal Methods in Digital Systems Safety

Formal methods [**Brayton1996VISAS**, **Brayton2010ABCAA**] are mostly used on unitary blocks or functionalities to prove assertions (i.e. properties) expressed in linear [**SVA**] or branching [**EMERSON1980**] timing logic. When applied to safety, it comes to proving absence of safety goals violations that are expressed as assertions on outputs in the presence of faults. Tools like [**jaspergold**][**yeung-2018**] are able to compute, given a nelist of logic gates and flip-flops and an initial state, the cone of influence of flip-flops or gates and whether a fault in such elements can propagate to a given output. Such structural analysis can perform *Out-of-Cone-of-Influence* (*COI*) fault analysis allowing to classify a fault as *safe* when it cannot reach a given output. Activation analysis determine whether a fault injected on a specific node can be activated. Propagatability analysis determine if an activated fault in a COI can propagate to a strobed output and detection analysis determines if a fault will (always) be propagated and detected at the checker output. Such analysis can reveal what logic is covered by a safety mechanism or not. However, no formal methods is able to address such safety properties at SoC level.

3.2.3 Altarica

Functional safety objective is to identify the most probable failure combinations leading to a feared event. Model-Based Safety Analysis performs safety analysis by building dysfunctional models for each block of the considered system

and using formal methods to combine and extract failure modes at the system level [**mortada-imbsa-2014**]. MBSA introduces the use of high level modeling languages dedicated to functional safety analysis [**Prosvirnova**] [**arnold**] [**bozzano-avocs-2010**]. It allows extending classical methods such as FMEA or fault trees. These languages help capture system dynamics and how failures propagate inside it. Moreover, models support structural modeling allowing identifying and locating induced effects of a failure inside the architecture.

Altarica Dataflow (Fig. 3.1) is an event-driven asynchronous language that implements discrete variables with a finite number of values, leading to a finite number of combinations of state values and propagated flows, allowing theoretically to cover the entire system state space. AltaRica Dataflow is at the core of several Reliability, Availability, Maintainability and Safety (RAMS) environments: Cecilia OCAS (Dassault Aviation), SimfiaNeo (Airbus Protect), and Safety Designer (Dassault Systèmes)

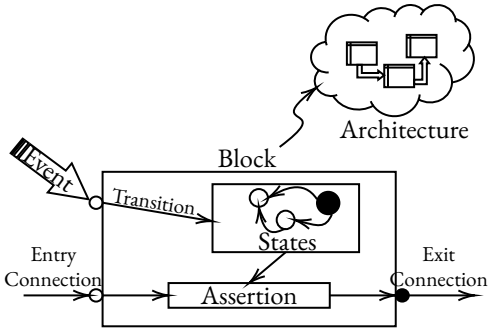


Figure 3.1: Altarica Dataflow Model

- *Variables*: AltaRica variables are discrete and represents an enumerated finite set of values called its *domain*. Variable definition inside its domain is free. The variable can represent for example functional modes, dysfunctional status, message types ....

  Inside MBSA models, state variables are generally used to represent dysfunctional status with a default value as nominal behavior and a value for each degraded mode reached from any failure mode. Flow variables are generally to describe the type of data exchanged between components. This type can represent a functional value (e.g. instruction value) or a dysfunctional value (e.g. message status). It depends of the model level of detail. As flows are used to propagate failures, they can be described either by sending a status or a faulty value.

- *Transitions*: Transitions describe possible states changing values. Transitions are guarded by a condition allowing the transition to become

fireable when true. A transition is associated with a triggering event and is fired when the event is triggered and the guard is true. In MBSA modeling, triggering events are used to represent failure modes. AltaRica allows to assign a probability law to an event, modeling the behavior of random failures or deterministic actions. The transition completion describes the effect of the failure mode on the component state. Guards can be enriched to restrict to describe conditional failures. For example, in a cold redundancy, some failures can't happen when the component is off.

- *Assertions*: Assertion is the mechanism used to set outputs values of a node. Output values are a function of input values and internal state values. Assertion can be interpreted as a logical function describing a truth table assigning outputs according to each combination of inputs and internal state values. Combinations are described through Boolean expressions and imperative programming constructs such as *if-then-else* or *case*.

  Assertions are used to propagate failures from a faulty component to other blocks. Fault injected on the internal state is propagated to its output and then to others blocks. Depending of the granularity level of the model, assertions are manipulating either functional values or states.

3.3 Modeling Digital Systems for MBSA

Digital systems, by essence, lend themselves well to finite state machines representation making the use of languages and formalism such as Altarica very suitable for their modelling. However, dysfunctional modeling requires extracting the faulty behaviour of the blocks composing the system. Such task is usually carried out by a Failure Mode and Effect Analysis (FMEA) to identify possible malfunction of the individual blocks. In digital system, such task can be performed automatically by simulation with fault injection[17] and possibly formal methods [**7333399**].

The main issue in modelling digital systems for MBSA is choosing the adequate level of abstraction avoiding a direct $1 \Leftrightarrow 1$ translation of *Hardware Description Languages* (HDL) modeling concepts into Altarica. When extracting a safety model from a digital block three points must be addressed:

- Structural hierarchy: Because Altarica support hierarchy [**PR14a**], translating hierarchy with adequate granularity can be straightforward, especially as natural design hierarchy is usually a good candidate.

- Behavioral modelling: Faulty behavioral aspects requires extraction of failure modes which can be performed manually, based on design knowledge or automatically using fault injection or formal approaches. Fault injection is well suited to such analysis, especially in the world of digital design which rely heavily on HDL simulators and digital fault injection driven by ISO26262 requirements. In this work we will exclusively focus on fault injection.

- Faults propagation: Blocks in a SoCs are usually connected though buses with well defined protocols and their failures modes (*unaligned access* ...) are known. The issue comes in the granularity of the modelling that, if too low will lead to too numerous events (1 HDL signal $\rightarrow$ 1 flow variable) while a too high abstraction may prevent catching of some protocol failures.

Fault injection campaigns are used to characterize the behavior of the system from its output pins point of view which are the 'vectors' for faults propagation between blocks. Also, knowing the functionality of each of these pins, it is possible to attach some possible consequences to the failure to such (group of) output(s). Such semantic labelling is, however, still manual and based on safety engineers knowledge and experience.

3.4 Methodology

On top of any explicit finite state machine or control code encoding the user specified behaviour, a more complete state exist that includes the totality of the signals belonging to the control path of a design, such as data states implicitly exposed in controls states, or implicitly coded control states. These signals compose a more complete and larger state machine exposing new states and transitions that are implicitly specified, for example resulting from Cartesian product of automatons. Those are, technically, the signals driving transitions conditions.

Combinations of these signals in those states can lead to a subtle set of fault states, difficult to identify from the HDL description as the encoding in this state machine is sparse due to correlations. Such argument comes from the fact that even for a small ($>\approx$50) number of flip-flops, the complete state space (2^{50}) cannot be traversed in a reasonable time. Therefore a non-negligible proportion of these states are what we call *illegal states* i.e. unreachable under normal behaviour, potentially leading to undesired and unspecified behavior when the block is exposed to those states though faults.

In order to build a failure model from a nominal behavioral *Register Transfer Model* (RTL) in Verilog or VHDL, behaviour of the system under faults must be analyzed and faulty behavior as well as failure modes must be extracted. We proceed using the following steps:

1. *Identification and Extraction of State Signals*: Starting from the functional description, the set of flip-flops, belonging to both the *control* and possibly *data*) paths composing what we name as the *state*, has to be identified and extracted. This set, composed by all the flip-flops composing the control path and possibly the datapath which maintain the control state of the block, correspond to possible fault injection sites.

2. *Testbench Setup* : A standalone testbench is set up with care given to coverage and testbench representability as the states traversed during this golden execution will serve as non-faulty reference behavior. Tools like *Incisive Metrics Coverage* (IMC) [**CDN**] or *Certitude* [32] can be used to assess testbench coverage. A first reference run is performed to allow extraction of golden functional states that will be used later in the process to be differentiated from non-functional ones under fault injection.

3. *Fault Injection Campaign*: Fault injection is the mean by which the misbehavior and faulty execution is exposed on purposes. Probes (i.e., observation points) are defined during the setup of the fault injection campaign. They are set on the outputs of all blocks in order to identify failures that propagates to other blocks. Probes monitor and compare the probed signal value at each clock cycle with the golden reference and report any difference. They have been set to stop simulation when a fault reaches an output of the design. This step is the core of our analysis aimed at extracting faulty behaviour, modes and effects though exploration of the faulty states by fault injection.

4. *Extraction of Faulty Behavior*: Once the faulty runs have completed, non-functional (i.e. *faulty*) states and behavior are extracted by subtracting functional (*golden*) states taken from the golden run state dictionary to the faulty run states, leaving only newly discovered faulty states and transitions.

5. *Construction of the Faulty Model*: The newly discovered states and transitions are used to augment the functional models with faulty behavior. Transitions from a functional to a non-functional state are labeled with the responsible faults so are states responsible for an incorrect output. This model serves as a base for the translation into the Altarica language.

Currently, the method is limited in the *effect* analysis of the FMEA. Effect such as *loss of power* cannot be attached automatically to a faulty state as it would requires an inference and abstraction process out the reach of the tool currently. Thus, such labelling is performed manually by attaching effects to outputs and then back-propagating them into the states and faults responsible for the given outputs corruptions.

3.4.1 Faulty Behavior Model Construction

Once the faulty behavior has been extracted from faulty runs, the faulty model can be constructed using graph analysis algorithms. The first step in the model construction is collapsing states that are not meaningful for the dysfunctional model. We proceed currently with the following rules:

- Any component (connected subgraph) comprising only legal states and legal transitions are collapsed into one single *functional* state.
- Legal states with illegal transitions or incorrect outputs (outputs values do differs from reference in these states) are kept and illegal transition probabilities are attached.
- Any component comprising only nodes not propagating any faults to outputs are collapsed into one single *faulty* state. Probabilities to enter this state can be extracted from transitions leading to the collapsed states.
- Faulty nodes propagating faults to outputs are kept and transitions probabilities are attached to allow computing incorrect outputs probabilities.
- Effect attached to output pins are back propagated in the state graph faulty states where output corruptions occurs.

However additional rules may be added like to remove faulty nodes and transitions from masked faults for example, especially those not leading to any latent faults (execution is correct with no faults propagated to outputs and internal state doesn't differs from reference one at some point, i.e. fault has *vanished*). We ultimately target discrete-time Markov chains [6] for our dysfunctional behaviour modelling (Fig. 3.2).

3.4.2 Completeness of Extraction

The main risk in state identification is to under or overestimate the state which would lead to uncovered faulty states (fault not injected in a flip-flop misidentified as not *control*) or over estimate the state leading to classification of what

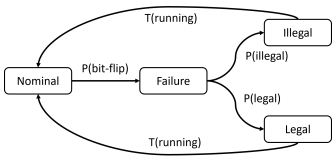


Figure 3.2: Altarica Dysfunctional Model

are, in fact, data state as control states. The latter can be easily identified as randomizing data in the golden or faulty state machine extraction step leads to an increasing number of states with the number of runs. On Fig. 3.3, a correct identification leads to a saturating number of states (green curves) while an incorrect one leads to a diverging number of states as the number of tests grows (red curve).

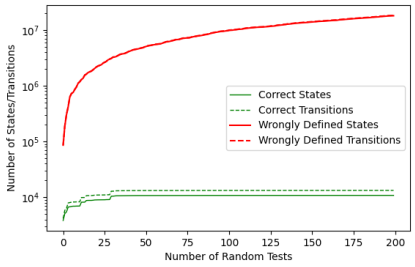


Figure 3.3: Completeness of the Extraction

3.4.3 Altarica Modeling

Such an automaton representation is adequate for Altarica modelling as described below. When performing translation to Altarica, two elements shall be extracted:

1. The internal state machine corresponding to failures.
2. The assertion part corresponding to the propagation failure probability from input to one or more output of element.

Base modelling must include at least four states (Fig. 3.6): a *nominal* state where no failure occurs, a *failure* state corresponding to a bit-flip error injection and an *illegal* state corresponding to propagation of the failure to one or more outputs. The *legal* state correspond to failures leading to legals transitions, without failure propagation to outputs.

Assertions on outputs are conditioned by the internal state machine and inputs of the block. Every time internal state machine is in the illegal state, outputs values are updated. In same way, if one input of the block is set in the failed state, outputs are updated. Probabilities to generate a faulty output or to propagate failures from inputs to outputs are extracted from fault injection campaigns (Table **??**). Currently, all illegal states are collapsed into a single one, but different non-functional states corresponding to different failure modes can be extracted as well, such as represented on Fig. 3.6 where two illegal states are identified whether or not a simulation timeout (10% of golden execution time) occurs. Criteria for refined dysfunctional automaton extraction are not yet addressed as well as construction and reduction rules from fault injection data for such an automaton.

3.5 Application: *I2C* to *AHB* bridge

In order to exercise the methodology presented in Section 4.3, we use a test case composed of 2 blocks: an *I2C* slave [21] connected to an *AHB* [11] bus master interface (Fig. 3.4). Commands (*read* or *write*) along with parameters (*address* and *data*) are received on the serial line and transformed into a series of AHB read and write transactions. Such a system, composed of two interconnected blocks, is humanely understandable so are its dysfunctional modes, while being complex enough to detail thoroughly the methodology.

The I2C slave, taken from [15], receives *read* or *write* commands followed by an address byte and an optional data byte. On an I2C read, the byte returned from the AHB read transaction is returned. chronogramm for the read and write sequences are represented on Fig. 3.5. The system is represented on Fig. 3.4. At both end of the system (I2C input and AHB output buses), I2C master and AHB slave Verification IPs (VIP) are attached to generate and verify correctness of I2C and AHB transactions.

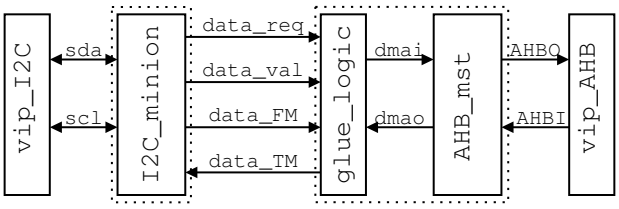


Figure 3.4: I2C to AHB System Block Diagram

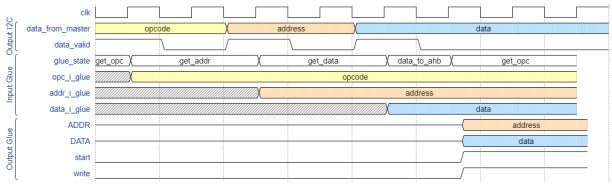


Figure 3.5: I2C/AHB System Model

Table 3.1: IMC Coverage Figures (%)

	I2C			AHB		
	cov.	tot.	overall	cov.	tot.	overall
Overall	352	410	93.8%	333	1057	61.3%
block	164	180	95.4%	51	68	85.55%
Expression	44	44	100%	7	17	41.18%
Toggle	112	148	75.68%	266	963	30.17%
FSM	32	38	83.36%	9	9	100%

3.5.1 I2C Block Modeling

The testbench is composed of a series of read and write random transactions. The coverage evaluation of the design has been carried out, results are presented in Table 3.1. Having considered the results of the coverage evaluations sufficient for the demonstration, application of the method presented in section 4.3 have been performed. The list of all injection sites, reported by Cadence *Xcelium Fault Simulator* (FSV) [**CDN**] fault injection tool are considered for state including ones containing data as the serial nature of the I2C protocol, which mixes control and data frames on the same signals thought time-multiplexing, doesn't allow differentiation. However, the small size of data considered (8-bit) only induce a low (256) superset of the real control states. All outputs are probed so that any mismatch with the reference run will stop the simulation and report the fault as *Detected*. State (flip-flop value, i.e. '0' or '1') is simply extracted at each clock cycle and printed in the simulation logs to be post-processed.

Fault injection traces are then processed following rules described in section 3.4.1 extracting transitions probabilities between the connected subgraphs:

- `Nominal 1` - Subgraph made of legal states only, part of the nominal execution.
- `Nominal 2` - Subgraph made of legal states only, part of the nominal execution.

- `Faulty 1` - Illegal state Subgraph, leading to a propagation of the fault to the output.
- `Faulty 2` - Illegal state Subgraph, leading to a simulation timeout.

The resulting model is represented on Fig. 3.6.

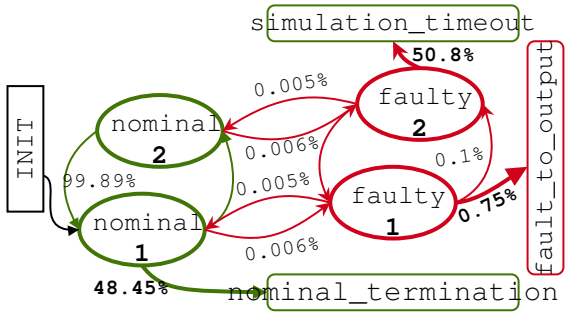


Figure 3.6: Extracted FSM for the I2C Block

3.5.2 AHB Block Modeling

The AHB bus interface is taken from the GRLIB [12] library with added custom logic to connect it to the master parallel interface of the I2C. The added logic comprise an interpreter for the command received by the I2C and the glue logic interface to the AHB master side. A verification IP is connected to the AHB slave interface side to respond to transactions and check protocol. Fig. 3.5 represents the translation of I2C signals into an AHB transaction by the system. Coverage for AHB block is low and can be explained as only a limited use of the AHB protocol is made:

1. only byte accesses are performed.
2. only single (SINGLE) non-sequential (NONSEQ) transfers are performed.
3. the VIP has not been programmed to insert HREADY wait states in the transaction.
4. the VIP has not be programmed to generate HRESP transaction response error.

The low coverage obtained here doesn't restrict the generality of the methodology but may prevents some failure modes to be identified in this specific case.

3.5.3 Complete System Test Case

The complete system is composed of both the I2C slave and AHB master along with VIPs at both ends. As previously mentioned, probes are placed on all outputs of the complete system, leaving this time, faults freely propagating internally between the I2C and the AHB without being reported by FSV nor the simulation to be stopped. The main difference of this testbench regarding the two standalone previous ones is that faults injected in one block will be able to propagate to the other one (I2C $\rightarrow$ AHB, for example) and back-propagate to the first block (AHB $\rightarrow$ I2C) as simulation will not be stopped when the fault will output from the first (i.e. I2C), and later second (i.e. AHB), block. Such "fault loop" (I2C $\circlearrowleft$ AHB or AHB $\circlearrowleft$ I2C) are expected to be the main possible source of faulty states differences between the standalone and full system faulty states extraction. However, as faults are injected on the inputs in both approach (standalone and full system), we expect to capture, at least a part of theses "fault loops" induced faulty states in the standalone extractions, if such case exist.

The AltaRica structural model architecture follows the natural hierarchy of the system. As shown on Fig. 3.7, the AltaRica models includes the exact same blocks with the same interconnections between blocks as the functional model. The main I2C and AHB modules are composed of two sub-elements, shown respectively in Fig. 3.8 and Fig. 3.9.

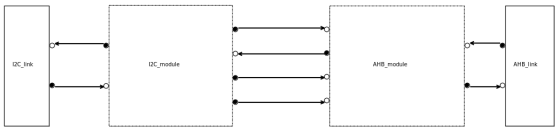


Figure 3.7: I2C to AHB System Model

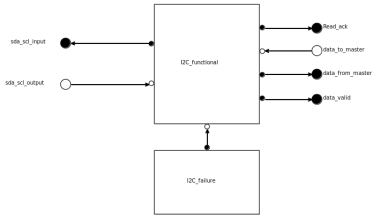


Figure 3.8: I2C Block Model

The first element is the *functional* state machine of the module. In case of internal or external fault, this state machine will dispatch the fault to the impacted outputs. This state machine only model the internal faults propagation and do not generate any random failure on its own. The second element is the

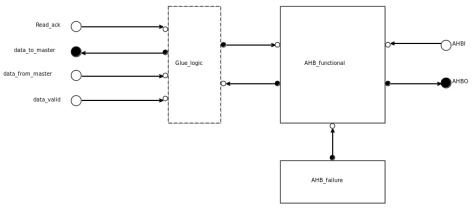


Figure 3.9: AHB Block Model

internal failure state machine. This state machine generate internal random failures and provide to the functional state machine the outputs impacted by it. In addition, the AHB module include an additional glue-logic block that converts I2C output signals to AHB bridge input signals. No internal failure are generated by this element. At both end of the I2C and AHB blocks, links module have been added to model the faults coming from outside of the system. To model the behavior of the I2C and AHB system, only standalone block fault injection test results have been used.

Depending on the methodology, two types of metrics can be extracted. The first one is the probability to propagate internal failure to one or more outputs of the system. From the test results, this probability has been extracted by considering all faults injected in the studied system. The probability to propagate internal faults to an output of the system is then equal to the ratio between the faults detected by the output probe and the total number of faults injected.

The second metrics is the probability to propagate a failure from an input of the system to one or more output of the system. For this metric, only fault injected on inputs have been taken into account. This probability is the ratio of the input faults leading to an erroneous output over the total number of input faults injected.

SimfiaNeo allows to perform Monte-Carlo simulation. In this type of simulation, a large number of failure scenarios are generated to assess the mean behavior of the system under random failure scenarios. The first possible assessment randomly injected one failure by failure scenario inside the system while the outputs are monitored. If at least one output triggers a faulty state, the error is accounted to have been propagated outside of the system. With this methodology, it's possible to estimate the probability to have a failure propagation from the I2C+AHB system to the I2C or AHB external signals. The second possible assessment randomly injected one failure by failure scenario in a link module and monitored the other link module. If the opposite link module triggers a faulty state, the error is accounted to have been propagated from one end to

Table 3.2: RTL Fault Injection Vs Altarica Model Failures

AHB Failure	I2C+AHB RTL	Altarica Model	I2C Failure	I2C+AHB RTL	Altarica Model
haddr	0.00551	0.00551	SDA	0.00338	0.00339
hwdata	0.00382	0.00382	Read req	0.00254	0.00255
hsize	0.00068	0.00068	Data	0.00580	0.00581
hbusreq	0.00026	0.00026	Data valid	0.00322	0.00323
hwrite	0.00022	0.00022			

another. With this methodology, it is possible to estimate the probability to have a failure propagation from AHB or I2C back to the other link.

3.6 Results

Result of composition obtained by SimfiaNeo are compared to fault injection performed on the full system with probes set only on external outputs of the system on table 3.2 for the I2C and AHB side signal. Because the system is simple and faults propagate only forward, it came to simple probability multiplication explaining exact matching of model and system fault injection. No back-propagating faults were observed.

3.7 Discussion and Future Work

In this work, we have proposed and experimented the use of Model-Based Safety Assessment on digital system for safety analysis. We have addressed the construction of dysfunctional model for digital system using simulation and have been able to build a simple, but functional dysfunctional model in Altarica. Ongoing work include automatic dysfunctional models reduction to more than one state and the application to a small RISCV SoC and software reliability[**iolts2022**].

Chapter 4

Footnotes and Sidenotes

4.1 Introduction

In the last years the density of integration in VLSI systems and microprocessors performances have continuously increased, thanks to the relentless technology scaling. Even though this trend can only continue on its path, several constraints may obstruct the way (power, energy, performance), in particular *reliability* (or cross-layer resilience) can become the more relevant. Hardware redundancy can be used to manage errors at the hardware architecture layer, and eventually even software implemented error detection and correction mechanisms can manage those errors that escalated from the lower layers of the stack [7] [14]. Overall, the goal is to determine the resilience of a particular system in determined conditions, meeting the requirements considering its sensitivity to hardware faults.

It is also true that software failures are not only caused by software implemented faults, as it has been shown [29] the propagation of hardware faults plays a central role, eventually catastrophic. Base on what literature reports on hardware faults evaluation reports [8] [9] it is possible to observe that the percentage of software failure that are caused by pure hardware faults average around 10% [18]. The most famous example is surely the crash of the Mars Polar Lander [2], which cause was established to be dependant on hardware faults resulting in software failure. In that case the lander was not able to settle the legs into their deployed position, which is an hardware fault, and the software gave a wrong order to turn off the engines in the air of Mars, which is a software fault. The system crashed and the entire mission failed.

This paper not only wants to furthermore analyse the behaviour of software failure due to hardware propagated fault but parallely to the main research [10] path that applies these new methodology to Hardware design in order to simplify the reliability assessment, the idea of applying the same method in the

scope of the assessment of the reliability of software has never been tested. In order to do so, there is the need to specify the main characteristic that Software Products have, fundamental to lay the basis for the described work. Every software can be divided into basic block, atomic chunks of software having the following proprieties:

- One entry point, meaning no code within it is the destination of a jump instruction anywhere in the program.
- One exit point, meaning only the last instruction can cause the program to begin executing code in a different basic block.

Under these circumstances, whenever the first instruction in a basic block is executed, the rest of the instructions are necessarily executed exactly once, in order. The code may be source code, assembly code, or some other sequence of instructions. More formally, a sequence of instructions forms a basic block if:

- The instruction in each position dominates, or always executes before, all those in later positions.
- No other instruction executes between two instructions in the sequence. This definition is more general than the intuitive one in some ways. For example, it allows unconditional jumps to labels not targeted by other jumps. This definition embodies the properties that make basic blocks easy to work with when constructing an algorithm.

The blocks to which control may transfer after reaching the end of a block are called that block's successors, while the blocks from which control may have come when entering a block are called that block's predecessors. The start of a basic block may be jumped to from more than one location. Laid these basis, if, as we'll show in this paper, the reliability metrics extracted for each basic block can be recomposed just knowing the sequence of block required to execute a precise operation, the need for a fault injection campaign on the entire software product doesn't stand anymore.

This paper is organised as follows: the current state of the art is summarised in section II; section III describes the proposed methodology, including its setup, the fault injection procedure and the re-composition of the results from each basic block; a test case is provided in Section IV, while Section V presents the obtained results and sketches some perspectives.

4.2 State of the Art

The rush to develop a methodology to assess the reliability and availability of electronic systems has speed up together with the increasing complexity of the microelectronic systems and the miniaturization of such devices. In particular an eye has been keep onto the propagation of faults throughout the entire stack of layers that compose the system as whole, starting from the technological layer all the way up to the software/application layer passing through hardware. In particular the extraction of reliability metrics for software has been the focus of a consistent thread of research [18][17][35] that aimed to verify:

1. whether the software respects the specification requirements,
2. the improvement of the software quality and,
3. *the reliability of the software*

Tools to verify the reliability of software, defined as the probability of the correct software performances for specific period of time on specific environments, have been already developed. In particular the SyRA [36] Cross-Layer Soft Error Resilience evaluation framework proposes a solid method to move from the industrial level Cross-Layer evaluation techniques that are still mainly guided by the sole experience of the designers [7]. These methods are all based on the use of fault injection tools, and they all produce satisfying results in their fields. Nevertheless they have limitation, the description of the Software Fault Models have always been based on the simulation of propagation from the hardware architecture up to software routines, assessing their impact in the correctness of the computation as in [27][37][24]. Moreover no attention has been given to the enormous effort that this type of campaign require, in terms of time, licences for tools and computational power, for an assessment that is limited to the hardware the application is running on and most importantly on the inputs the software receives to perform its calculation. This makes the assessment completely not re-usable in the future requiring a completely new set of campaigns.

Here the focus will be, instead, put on how the software computation reacts to the vulnerable hardware underneath and most importantly to the de development of a methodology like there are no other example in the related research, the possibility of decomposing the software products to abstract the single basic blocks and perform a reliability assessment on the single, apparently meaningless blocks to then recompose them obtaining the reliability assessment with a huge time and computational power advantage with respect to the existing methods.

4.3 Methodology

The Classical reliability assesment of Hardware as well as Software is Fault Injection driven. The extensive usage of commercial fault injection tools like the ones provided by **Cadence** [5] or **Synopsys** [33] guarantees the proper exploration of the behaviour of the DUT when subject to SEU or other types of faults. This allows the Verification Engineers to have an idea of the behavior of their design without the need to move onto practical testing in radiation environments, which require a dedicate setup [28] and an expensive and not widely available infrastructure.

These advantages come at two main costs, *time* and *Computational Power*, which are comsumed in great quantities by the above mentioned simulators. Attempts of Optimization and Parallelization have been put in practice before, but they are not tackling the bigger overhead that we need to take care of every time we simulate a design. Let us assume that, as shown in Fig:4.3 there is the need to test and entire Software Product composed of n basic blocks, this simulation will last as long as the time to initialize T_{init} plus the time of the checker/footer to be executed T_{foot} plus the sum of the duration of all basic blocks multiplied by their multiplicity through the program $m_n \cdot T_{bb_n}$. All multiplied by the number of runs that the simulator has to perform to achieve the desired number of injections I, resulting in:

$$T_{campaign} = I \cdot \left[T_{init} + T_{foot} + \sum_{0}^{N} m_n \cdot T_{bb_n} \right] \tag{4.1}$$

In which the entire program is executed every time entirely, the method proposed by this paper consist in a fragmented study of the basic blocks composing the software, extracting the same metrics that would be extracted by the same fault injection campaing on the whole Software. In this case, in the same way we did before, it is possible to calculate the time needed to carry out the fault injection campaign as we have defined it now, on separate basic blocks, each of them having their random initialization and checker to ensure functionality.

$$I \cdot \left[T_{init} + T_{foot} + \sum_{0}^{N} T_{bb_n} \right] \tag{4.2}$$

In this way we have drastically reduced the amount of time needed to perform the same amount of fault injections, just focusing on the single blocks. Moreover, the difference between the two previously calculated timings, will give us the benefit of studying the blocks singularly, as follow:

$$
\begin{aligned}
I \cdot \left[\sum_{0}^{N} m_n \cdot T_{bb_n} \right] - I \cdot \sum_{0}^{N} T_{bb_n} = \\
= I \cdot \left[\sum_{0}^{N} m_n T_{bb_n} - \sum_{0}^{N} T_{bb_n} \right] = \\
= I \cdot \sum_{0}^{N} T_{bb_n} \cdot (m_n - 1)
\end{aligned}
\tag{4.3}
$$

which means that we save the time needed for the execution of each basic block multiplied by its multiplicity, minus one that we still have to execute. Clearly this saved time increases with the length of the Software and therefore the multiplicity of the blocks. In particular, the length of the Fault injection campaign on the entire software is linear with respect to the increasing of multiplicity of the basic blocks, for example due to a larger data input, whereas the solution proposed in this paper is linear with respect to the overall number of unique basic blocks, which remain the same regardless of the data.

4.3.1 Setup

The first step towards the application of the method described in the previous section, is the identification and of the different basic block that compose the Software Product under analysis. This can be easily carried out automatically by a simple parser. Basic Block at Assembly level are easy to identify and parse thanks to their intrinsic definition of linear chunks of code. It is therefore trivial to identify in the code all those instructions that modify the flow of the program, tearing down the hypothesis of linearity that defines the blocks themselves. For instance, all the jumping and branching point define the end of a block, as well as the beginning of the following one. Labels in the code also identify starting point of basic block, as they are frequently arrival points for the above mentioned jump and branch operations.

Although having the set of basic blocks divided in single file may seem sufficient, there still the need to initialize all the resources that both the processor and the basic block itself need to run properly, as well as a control logic to ensure that the functionalities of the basic block are preserved (or not) throughout the course of the fault injection campaign. As Shown in fig:2 a **random** initialization is included in the header for the basic block, ensuring the non dependability of the reliability metrics extracted on the input data, together with a footer that checks the functionalities of the block itself. Notice that in in this case, contrary

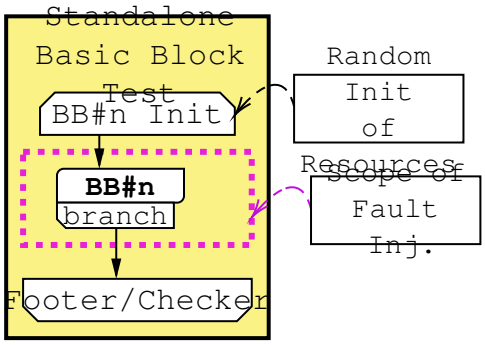


Figure 4.1: Block Diagram of the Entire SW Product

of what is done in the Hardware methodology, there is no physical probing of the circuit on which the program or the testbench is running. In this study only the functional aspect of the Software Product under test is observed.

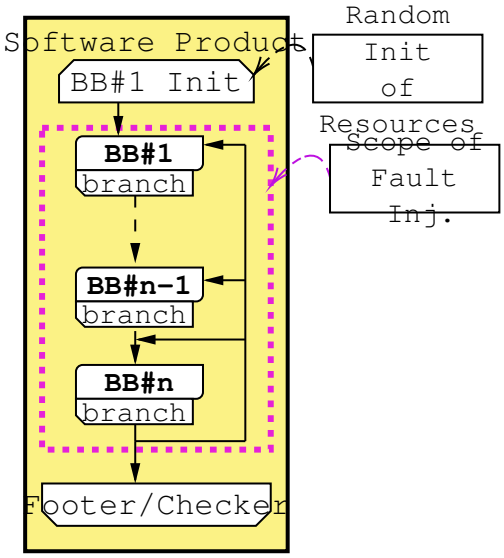


Figure 4.2: Block Diagram of the Entire SW Product

4.3.2 Fault Injection on Randomly Initialized Resources

Fault injection is the mean by which the misbehavior and faulty execution is provoked on purpose on digital systems. In the past, especially on hardware, fault injection was aimed to functionally verify the designs under test. Those DUT were analysed, their functions (data dependent) extracted and inputs

were selected in order to exercise those functions. Later on the fault injection had the role of determining whether those functions were preserved in cases of fault or how eventually they were modified. Today this is still the state of the art for software verification.

With time a second approach on hardware was presented, testing moved from functional to structural, where the integrity of the device is evaluated, regardless of the function (and therefore of data), verifying solely the implemented boolean function.

The methodology introduced in this paper presents the novelty of applying this structural approach to software. To abstract the basic block as much as possible from its link to data, **every resource utilized has been randomized** before each fault injection. The probability of failure and propagation probabilities are therefore extracted **independently** of their data input.

Probes (i.e., observation points) are defined during the setup of the fault injection campaign. It is the role of the Footer/Checker (out of the scope of the fault injection) to redirect the output of the block function into a reserved portion of the memory to be probed. Probes are set on those reserved memory location on all blocks, not probing the correctness of the data with respect to the golden run, but solely if the basic function included in that portion of code has been affected by the fault injection.

4.3.3 Re-Composition of the basic blocks

Once the fault injection campaigns are over, it is time to re-compose the information that have been extracted on the single blocks into a complete description of the original Software Product. To perform the re-composition there is the need run the software once and record the trace, this will allow us to know exactly the sequence in which the basic blocks have been executed during the nominal run.

We distinguish two main branches of the re-composition, the ones containing fault that *do not modity the program flow* and those that lead to a *modified program flow*

Not modified Program Flow

First we nee to define the probability of being executing a precise basic block in time during the execution of the program. Assuming a deterministic duration per executed instruction, without nested or hidden operation, we can define

the probability of executing BB_n as

$$P_{in-bb_n} = \frac{instructions - in - BB_n}{total - instruction - in - exe} \tag{4.4}$$

Next step is to define the probability of a fault happening in BB_n being able to become an error in the same block. This has been deduced from fault injection and must be differentiated per every register in which we inject faults and it represented as:

$$P^{A_m}_{gBB_n} \tag{4.5}$$

Last probability to define is the probability of a block to receive a wrong input and propagate it to its output. Defined as:

$$P^{A_m}_{pBB_n} \tag{4.6}$$

which is related to the "time of life" of the variables, defined as the number of basic block between the last time a variable has been read and the first time it gets overwritten.

Once these probabilities ha been defined we can describe the worst possible case, in which a fault is injected in BB_n and gets propagated throughout the whole program.

$$\begin{aligned} &P_{in-bb_n} * P^{A_m}_{gBB_n} * \left[P^{A_m}_{pBB_{n+1}} \dots P^{A_m}_{pBB_f}\right] + \\ &+ P_{in-bb_{n+1}} * P^{A_m}_{gBB_{n+1}} * \left[P^{A_m}_{pBB_{n+2}} \dots P^{A_m}_{pBB_f}\right] \dots \end{aligned} \tag{4.7}$$

which summarizes, per every register A_m as:

$$\sum_{n=0}^{N} P_{in-bb_n} * P^{A_m}_{gBB_n} * \left[\prod_{i=n}^{N} \left[P^{A_m}_{pBB_{i+1}}\right]\right] \tag{4.8}$$

$$P_{tot} = [P_{in-bb-x} * P_{p-bbx}] + [P_{p-bbx+1} * P_{p-bbx+1}] \dots \tag{4.9}$$

Modified Program Flow

Regarding the possibility of having a fault injected on a register while the program is executing a precise Basic Block that requires a branching operation at the end, we cannot consider them while recomposing the metrics as in the previous subsection.

These blocks contribute instead to the composition of a particular subset of runs (diverse behaviour of the program) which include all those runs in which the program simulation has reached the end in a time that differs from the nom-

inal one. In particular, it can be shortened due to a premature jump to the conclusive part of the program as well as delayed due to an incorrect loop that sends the machine into a non-necessary series of states from which it will eventually recover. In the case in which the machine would not be able to recover, we categorize those runs as Timeouts (when longer than 150% of nominal time). *It worth to point out that, due to the nature of the injections, which focus on the Register file, with one SEU per run, these cases are reduced to the minimum, if not nonexistent.* Give these assumptions, taking into account this second section of Basic Blocks, it is possible to assume that most, if not all of these runs will generate a failure in the functionalities of the program itself. therefore the recomposition, that was missing a good half of what was neded, now finds the missing cases in all those blocks that led to a modification in the flow.

In particular, considering the possibility that this blocks have not to propagate (to mask) a fault occurring in the course of their routine, the event of flow corruption has probability $1 - P_{masking}$, then the probability of these fault becoming a funcional error is 100% and it does not propagate. In this case the recomposition technique is slightly different than the previous section, as the case of a missed branch or jump leads directly to an error. So defined the multiplicity of the same critical block in the nominal sequence m, the probability of having a functinoal failure is described by:

$$P_{err} + P_{msk} * P_{err} + (P_{msk})^2 * P_{err} \cdots + (P_{msk})^m * P_{err} \tag{4.10}$$

Taking into accoun the approximation due to the algorithm intrinsic ability to recover from a flow error.

4.4 Test Case and Application

4.4.1 The Software

The Software of choice for the Proof of Concept of this methodology is the Bubble Sort Algoritm, in its Assembler for RISC-V Version. Bubble sort is an $O(n^2)$ sorting algorithm. A simple sorting algorithm that performs a one-way comparison of two adjacent records from the head to tail of the disordered part in each sort trip. Of course, the direction can also be the contrary, one-way comparison from the tail to head of the disordered part. This will form gradually an ordered table at the head of the disordered table, and the basic idea of the algorithm has no difference with the foregoing [23].

Table 4.1: Result of Fault Injection on basic blocks

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16		x31
bb_0	0	0	139	0	0	0	0	0	379	0	0	0	0	0	0	162	0	...	0
bb_1	0	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	...	0
bb_2	0	0	0	0	0	0	0	0	269	0	0	0	0	28	14	38	0	...	0
bb_3	0	0	0	0	0	0	0	0	500	0	0	0	0	16	131	244	0	...	0
bb_4	0	0	0	0	0	0	0	0	495	0	0	0	0	0	0	62	0	...	0
bb_5	0	0	0	0	0	0	0	0	380	0	0	0	0	0	0	0	0	...	0
bb_6	0	0	0	0	0	0	0	0	494	0	0	0	0	0	0	41	0	...	0
bb_7	0	0	0	0	0	0	0	0	466	0	0	0	0	0	0	0	0	...	0

Table 4.2: Comparison of Fault Injection data vs Recomposed data on Entire Software

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16		x31
FI	0	0	4	0	0	0	0	0	221	0	0	0	0	15	41	90	0	...	0
Reco	0	0	5	0	0	0	0	0	228	0	0	0	0	13	44	98	0	...	0

Table 4.3: Control Flow Driven Errors

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16		x31
FI	0	0	0	0	0	0	0	0	198	0	0	0	0	4	77	90	0	...	0

4.4.2 The Division in Basic Block

The processing of dividing the Software under test into basic block has been carried out automatically and returned 8 different blocks, together with the list of resources that each and every basic blocks utilizes during its own functions. After a Nominal run without faults of the entire software, it was possible to trace the transition between the different basic blocks throughout the whole execution. These information, summarized in the scheme below, will be the key to predict the behaviour of the program starting from the behaviour of the basic blocks themselves.

4.4.3 The Platform

The choice of the platform on which the program has been run and tested fell on the SCR1SOC. SCR1 is an open-source and free to use RISC-V compatible MCU-class core, designed and maintained by Syntacore. It is industry-grade and silicon-proven (including full-wafer production), works out of the box in

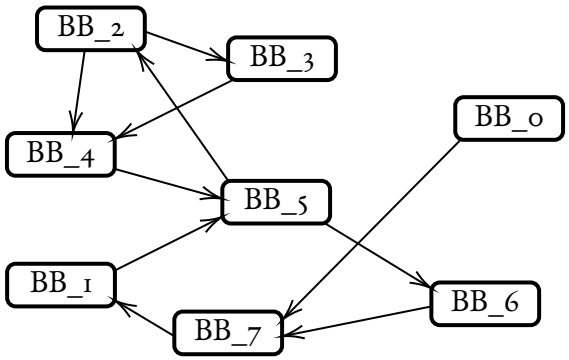


Figure 4.3: Block Diagram of the Entire SW Product

all major EDA flows and Verilator, and comes with extensive collateral and documentation []. This choice had mostly been driven by the larger and larger usage of these kind of RISCV based cores in the academic community. Any test based on these platforms is and added value to their development.

4.4.4 The Fault Injection Campaign

The next step in the methodology is fault injection. It is performed using Cadence fault injection tool *FSV* [5]. Once fault injection sites are automatically identified from the RTL description, fault injection is performed and 20 faults are injected per identified site using a custom pre-generated fault dictionary, including a random injection time. An in-house tool build on top of the *GSL* [13] has been developed for this purpose. Such number is statistically significant enough [20] without compromising fault injection campaign running time. Faults are injected on the integrity of the register file, mimic as well the possibility of faults propagated to the memory and back. Also, fault probes are set on non excerciced memory location to record which injected faults will cause a functional failure of the basic block. For each fault injection run, a logfile is generated which reports the outcome of the run, later a custom made parsing tool will recollect the data from this logfile and present the results to the re-composition tool.

4.5 Results and Future Work

The results of the recomposition are based on Table:1, which summarizes the result of the fault injection campaign that has been carried out on the single

basic blocks. Each entry of the table enumerates the number of functional error on caused by each register in the register file, keeping in mind that every bit in the register has been affected by 20 faults randomized in time, for a total of 640 faults per register. Once these table has been given to the recomposition tool, Table:2 is returned, including the benchmark fault injection campaign on the entire Software Product for validation of results together with the expected number of faults, calculated following the methodology described. Last, Table:3 Reports the number of Errors that have been caused by an error in the flow of the program, which can be extracted by an equivalent of table number 1 for flow errors caused by each register failing in each basic block and recomposed as in its dedicated section.

The last part of the methodology will be the focus of the work to come, as includes the implicit ability of the different algorithms to recover from flow errors, which understanding can lead to much more refined results.

Chapter 5

Enter Title Here

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Chapter 6

Conclusion

This is where your Conclusion will go

Appendix A

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Bibliography

[1] R. Baumann. "Radiation-induced soft errors in advanced semiconductor technologies". In: *Device and Materials Reliability, IEEE Transactions on* 5 (Oct. 2005), pp. 305–316.

[2] M. Blackburn et al. "Mars Polar Lander fault identification using model-based testing". In: *Eighth IEEE International Conference on Engineering of Complex Computer Systems, 2002. Proceedings.* 2002, pp. 163–169. DOI: 10.1109/ICECCS.2002.1181509.

[3] C. Bottoni et al. "Heavy ions test result on a 65nm Sparc-V8 radiation-hard microprocessor". In: *2014 IEEE International Reliability Physics Symposium*. 2014.

[4] Sebastien Bourdarie and Michael Xapsos. "The Near-Earth Space Radiation Environment". In: *IEEE Transactions on Nuclear Science* 55.4 (2008), pp. 1810–1832. DOI: 10.1109/TNS.2008.2001409.

[5] *Cadence*. https://www.cadence.com/en_US/home.html.

[6] Yen-Chi Chen. *Discrete-Time Markov Chain*. 2018. URL: http://faculty.washington.edu/yenchic/18A_stat516/Lec3_DTMC_p1.pdf.

[7] Eric Cheng et al. "CLEAR: Cross-layer exploration for architecting resilience: Combining hardware and software techniques to tolerate soft errors in processor cores". In: *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2016, pp. 1–6. DOI: 10.1145/2897937.2897996.

[8] Mojtaba Ebrahimi et al. "A fast, flexible, and easy-to-develop FPGA-based fault injection technique". In: *Microelectronics Reliability* 54.5 (2014), pp. 1000–1008. ISSN: 0026-2714. DOI: https://doi.org/10.1016/j.microrel.2014.01.002. URL: https://www.sciencedirect.com/science/article/pii/S0026271414000067.

[9] Mojtaba Ebrahimi et al. “Revisiting software-based soft error mitigation techniques via accurate error generation and propagation models”. In: *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2016, pp. 66–71. DOI: 10.1109/IOLTS.2016.7604674.

[10] Tiziano Fiorucci et al. “Automated Dysfunctional Model Extraction for Model Based Safety Assessment of Digital Systems”. In: *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2021, pp. 1–6. DOI: 10.1109/IOLTS52814.2021.9486705.

[11] P. Giridhar and P. Choudhury. “Design and Verification of AMBA AHB”. In: *2019 1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing Communication Engineering (ICATIECE)*. 2019, pp. 310–315. DOI: 10.1109/ICATIECE45860.2019.9063856.

[12] *GRLIB*. https://www.gaisler.com/products/grlib/grlib-gpl-2020.4-b4261.tar.gz.

[13] *GSL, GNU Scientific Library*. https://www.gnu.org/software/gsl/.

[14] Jörg Henkel et al. “Reliable on-chip systems in the nano-era: Lessons learnt and future trends”. In: *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2013, pp. 1–10. DOI: 10.1145/2463209.2488857.

[15] *I2c Minion Repository*. https://github.com/oetr/FPGA-I2C-Minion.

[16] Tanay Karnik, Peter Hazucha, and Jagdish Patel. “Characterization of soft errors caused by single event upsets in CMOS processes”. In: *IEEE Transactions on Dependable and Secure Computing* 1 (2004).

[17] Maha Kooli and Giorgio Di Natale. “A survey on simulation-based fault injection tools for complex systems”. In: *2014 9th IEEE International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*. 2014, pp. 1–6. DOI: 10.1109/DTIS.2014.6850649.

[18] Maha Kooli et al. “Computing reliability: On the differences between software testing and software fault injection techniques”. In: *Microprocessors and Microsystems: Embedded Hardware Design (MICPRO)* 50 (May 2017), pp. 102–112. DOI: 10.1016/j.micpro.2017.02.007. URL: https://hal-lirmm.ccsd.cnrs.fr/lirmm-01693156.

[19] L. Ratti. *Ionizing Radiation Effects in Electronic Devices and Circuits*. Accessed: 2018-12-06. 2017.

[20] R. Leveugle et al. "Statistical fault injection: Quantified error and confidence". In: *2009 Design, Automation Test in Europe Conference Exhibition*. 2009, pp. 502–506. DOI: 10.1109/DATE.2009.5090716.

[21] Jayant Mankar et al. "Review of I2C protocol". In: *International Journal of Research in Advent Technology* 2.1 (2014).

[22] R. Mariani, G. Boschi, and F. Colucci. "Using an innovative SoC-level FMEA methodology to design in compliance with IEC61508". In: *2007 Design, Automation Test in Europe Conference Exhibition*. 2007, pp. 1–6. DOI: 10.1109/DATE.2007.364641.

[23] Wang Min. "Analysis on Bubble Sort Algorithm Optimization". In: *2010 International Forum on Information Technology and Applications*. Vol. 1. 2010, pp. 208–211. DOI: 10.1109/IFITA.2010.9.

[24] S. Mirkhani, M. Lavasani, and Z. Navabi. "Hierarchical fault simulation using behavioral and gate level hardware models". In: *Proceedings of the 11th Asian Test Symposium, 2002. (ATS '02).* 2002, pp. 374–379. DOI: 10.1109/ATS.2002.1181740.

[25] S.S. Mukherjee, Joel Emer, and S.K. Reinhardt. "The Soft Error Problem: An Architectural Perspective". In: Mar. 2005.

[26] Shubhendu S. Mukherjee et al. "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor". In: 2003.

[27] Shubu Mukherjee. *Architecture Design for Soft Errors*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008. ISBN: 9780080558325.

[28] Marco Ottavi et al. "Setup and experimental results analysis of COTS Camera and SRAMs at the ISIS neutron facility". In: *2018 13th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS)*. 2018, pp. 1–4. DOI: 10.1109/DTIS.2018.8368564.

[29] Jinhee Park et al. "An Embedded Software Reliability Model with Consideration of Hardware Related Software Failures". In: *2012 IEEE Sixth International Conference on Software Security and Reliability*. 2012, pp. 207–214. DOI: 10.1109/SERE.2012.10.

[30] Robert N. Cherry. *Encyclopaedia of Occupational Health and Safety 4th Edition-Chapter 48.*

[31] S. di Mascio. *Low-Energy Proton Test of Safety-Critical Microcontrollers for Space Applications.*

[32] *Synopsys Certitude*. `https://www.synopsys.com/verification/simulation/certitude.html`.

[33] *Synopsys Zo1x*. `https://www.synopsys.com/verification/simulation/z01x-functional-safety.html`.

[34] Emmanuel Touloupis et al. "Study of the Effects of SEU-Induced Faults on a Pipeline Protected Microprocessor". In: *IEEE Transactions on Computers* 56.12 (2007).

[35] A. Vallero et al. "Cross-layer reliability evaluation, moving from the hardware architecture to the system level: A EU project overview". In: *Microprocessors and Microsystems* 39.8 (2015), pp. 1204–1214. ISSN: 0141-9331.

[36] A. Vallero et al. "SyRA: Early System Reliability Analysis for Cross-Layer Soft Errors Resilience in Memory Arrays of Microprocessor Systems". In: *IEEE Transactions on Computers* 68.5 (2019), pp. 765–783. DOI: 10.1109/TC.2018.2887225.

[37] N.J. Wang and S.J. Patel. "ReStore: Symptom-Based Soft Error Detection in Microprocessors". In: *IEEE Transactions on Dependable and Secure Computing* 3.3 (2006), pp. 188–201. DOI: 10.1109/TDSC.2006.40.

[38] Nicholas Wang, Aqeel Mahesri, and Sanjay Patel. "Examining ACE analysis reliability estimates using fault-injection". In: vol. 35. June 2007.

[39] James F. Ziegler and G. R. Srinivasan. "IBM Journal of Research and Development". In: *IBM J. Res. Dev.* 40.1 (1996), p. 2.