THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : EEATS - Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)
Spécialité : Nano électronique et Nano technologies
Unité de recherche : Techniques de l'Informatique et de la Microélectronique pour l'Architecture des systèmes intégrés

Méthodologie de qualification pour la certification ISO26262 des systèmes-sur-puce pour lautomobile

Qualification methodology for ISO26262 certification of automotive SoC systems

Présentée par :

Tiziano FIORUCCI

Direction de thèse :

Giorgio DI NATALE DIRECTEUR DE RECHERCHE, Université Grenoble Alpes	Directeur de thèse
Jean-Marc DAVEAU STMicroelectronics	Co-encadrant de thèse

Rapporteurs :

Alberto Bosio
PROFESSEUR DES UNIVERSITES, Université de Lyon
Stefano di Carlo
FULL PROFESSOR, Politecnico di Torino

Thèse soutenue publiquement le **6 juin 2023**, devant le jury composé de :

Giorgio DI NATALE DIRECTEUR DE RECHERCHE, Université Grenoble Alpes	Directeur de thèse
Vincent Beroulle PROFESSEUR DES UNIVERSITES, Grenoble INP	Examinateur
Pascal Benoit MAITRE DE CONFERENCES HDR, Universite' de Montpellier	Examinateur
Alberto Bosio PROFESSEUR DES UNIVERSITES, Université de Lyon	Rapporteur
Stefano di Carlo FULL PROFESSOR, Politecnico di Torino	Rapporteur

Invités :

Jean-Marc Daveau
INGENIEUR DOCTEUR,

Qualification methodology for ISO26262 certification of automotive SoC systems

by

Tiziano Fiorucci

(Under the Direction of Giorgio di Natale)

Abstract

This 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.

French Translation Cette thèse propose de mettre en place un flux et une méthodologie de certification ISO26262 pour les circuits intégrés de type système sur une puce numérique dédiée à la conduite. Ces circuits sont généralement composés de plusieurs propriétés intellectuelles, IP, dédiées à différentes fonctions telles que la communication ou le traitement d'informations provenant de capteurs (caméra, lidar...), le système en temps réel, la vision et l'imagerie, la gestion du système (système d'exploitation), la sécurité. La méthodologie ISO26262 nécessite l'extraction d'un certain nombre de métriques liées à la résilience du système face aux pannes simples et multiples, ainsi qu'à l'efficacité des contre-mesures (détection, signalement et correction des erreurs) et modes de défaillance. L'extraction des métriques d'échec à partir des arbres de défaillance est une méthode connue et documentée dans la littérature. Néanmoins, son application a souvent été limitée aux systèmes électromécaniques macroscopiques tels qu'une voiture, un actionneur ou des chaînes de capteurs. D'autre part, ces méthodes sont rarement appliquées dans le domaine des SoC automobiles où l'extraction des métriques est encore largement manuelle (généralement à l'aide d'un tableur) et dépendante d'un expert, et où la vérification de l'efficacité des contre-mesures est mieux effectuée par injection de fautes ciblée sur quelques sous-parties du système complet ou par irradiation sous un faisceau de particules. Cette thèse propose de développer une méthodologie d'extraction de métriques de fiabilité basée sur l'injection de fautes par bloc ainsi que des méthodes de composition pour obtenir les métriques au niveau du système complet. La première partie de la thèse sera consacrée à l'étude de la bibliographie sur la construction d'arbres de défaillance, la norme ISO26262 et la déclinaison des différentes métriques de fiabilité dans le cas d'un système SoCs numérique. L'extraction des métriques au niveau du bloc sera basée sur deux méthodes différentes, l'une analytique basée sur les probabilités, l'autre expérimentale basée sur l'injection de fautes. L'objectif n'est pas de développer de nouveaux codes de probabilité ou des outils d'injection de fautes, mais de développer une

méthodologie pour les utiliser dans le contexte d'un SoC afin d'obtenir les données souhaitées. La deuxième partie de la thèse concernera la composition des données obtenues au niveau du bloc fonctionnel afin d'obtenir les métriques ISO26262 au niveau du système (SoC). Il s'agira de développer une méthode de composition adaptée en particulier aux caractéristiques des SoCs (système de communication, effectuant des calculs devant réagir en temps réel, ...) et aux modèles de défaillance qui les caractérisent ou imposés par la norme ISO26262.

Index words: [FMEA, ISO26262, System on Chip, Reliability, Functional Safety]

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 Grenoble Alpes.

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:	Alberto Bosio Stefano di Carlo Pascal Benoit Vincent Berouille

Electronic Version Approved

I am Me.

Dedica

La decisiona di intraprendere un percorso di studi aggiuntivo a cio' che la maggior parte degli ingegneri intraprende e' qualcosa su cui si siflette molto. Sono anni tolti alla carriera, anni di instabilita' prolungata oltre a quello che normalmente ci si protrebbe aspettare. In fondo pero' la prolungazione dello status da studente piu' che per i normali anni del liceo, e per alcuni, gli anni dell'universita', non e' un piacere che apprezzano in molti, ed e' impossibile ignorare il fatto che se questa distinzione esiste, e se si sente di far part edi quel tipo di personalita', puo' solo che essere un peccato ignorare che esiste una comunita' di scienziati-ingegneri-medici-letterati che condivide l'amore per la propria materia allo stesso modo e merita quanto meno di essere esplorata e conosciuta, puntando eventualmentea unircisi e tentare di farne parte.

come vediamo iun questo file se si compila automaticamente

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

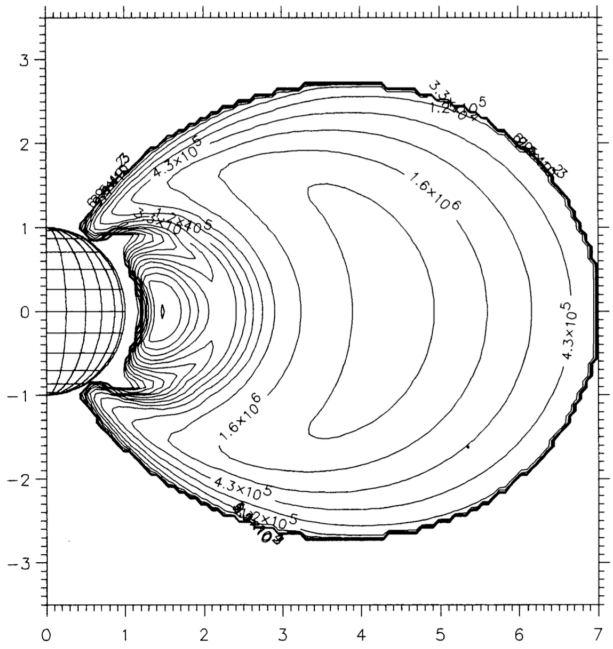
Part I

First Part - Introduction

Chapter 1

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%) [15]. 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 causes started, in all its great variety of environments depending on different orbital levels and electromagnetic forces involved. The degradation and disturbances induced by space radiation in the materials and the electronic components are phenomena that have been studied for many years.[15] It resulted in a basic classification of damages, either for Humans and for Electronics that can be easily 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. **Stochastic** 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 continuous. 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

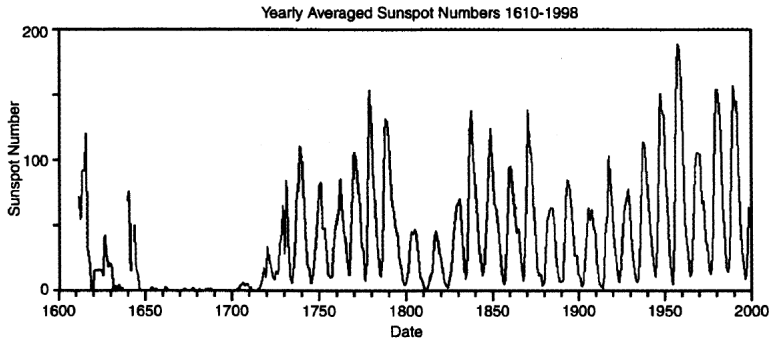


Figure 1.1: The observed record of yearly averaged sunspot numbers [15]

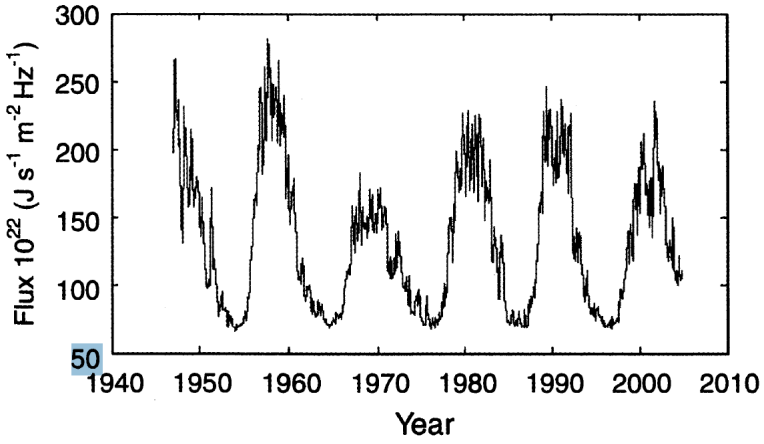


Figure 1.2: Measured values of solar 10.7 cm radio flux [15]

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 intended all those highly-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 Composition	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.

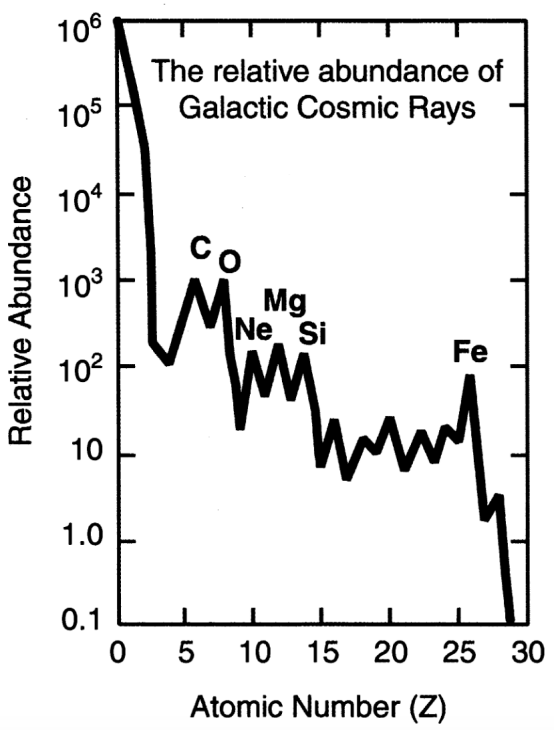


Figure 1.3: Abundances of GCR up through Z = 28

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 causes 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.

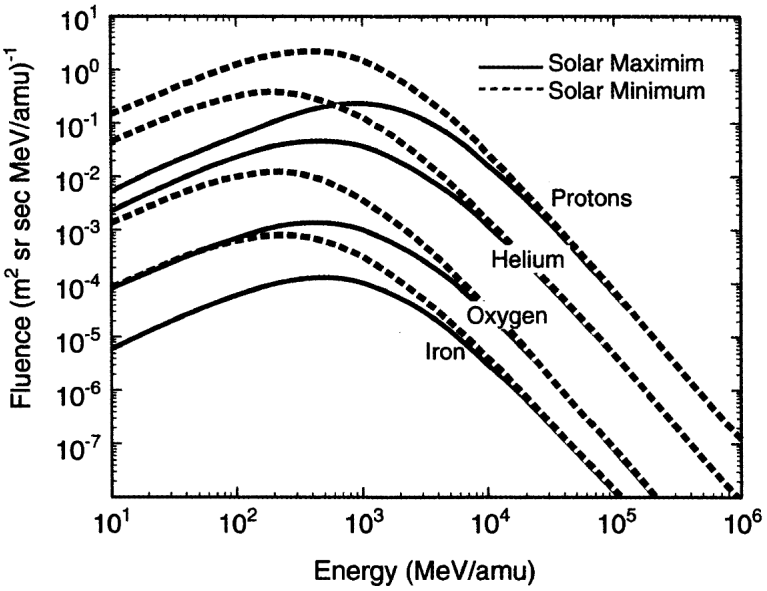


Figure 1.4: GCR energy spectra for protons, helium, oxygen and iron during solar maximum and solar minimum conditions

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

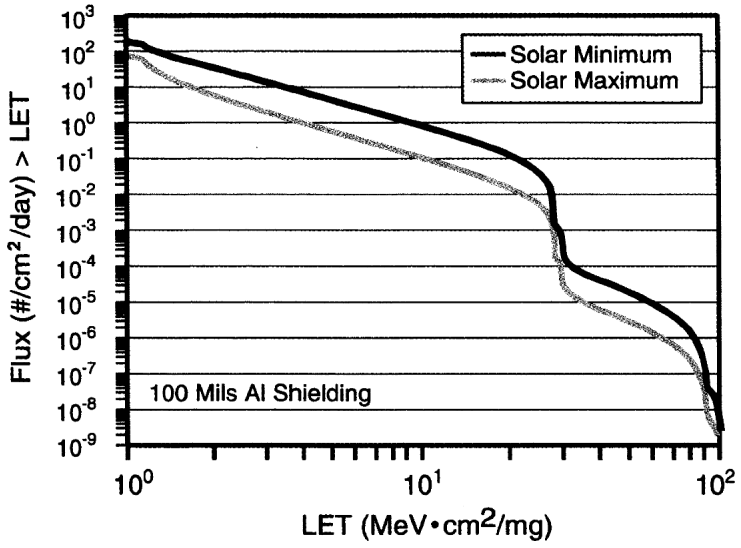


Figure 1.5: Integral LET spectra for GCR during solar maximum and solar minimum

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 length 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 outer 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

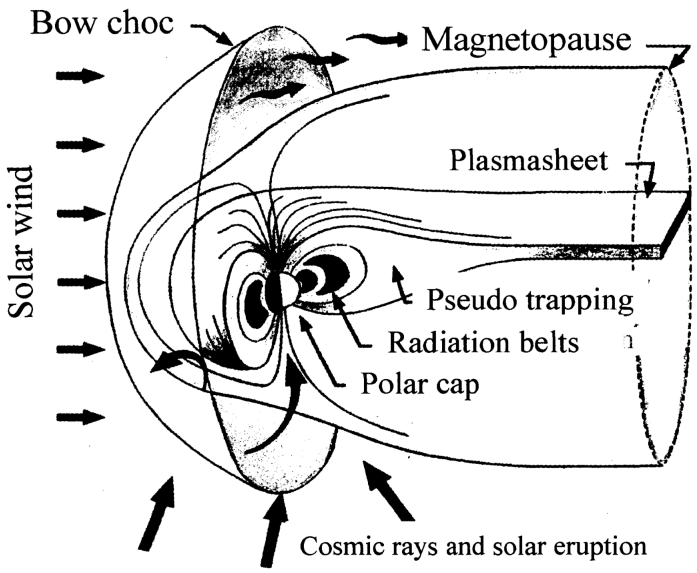


Figure 1.6: Magnetosphere with respect to Radiation Belts [15]

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) [15]
- 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. [15]

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

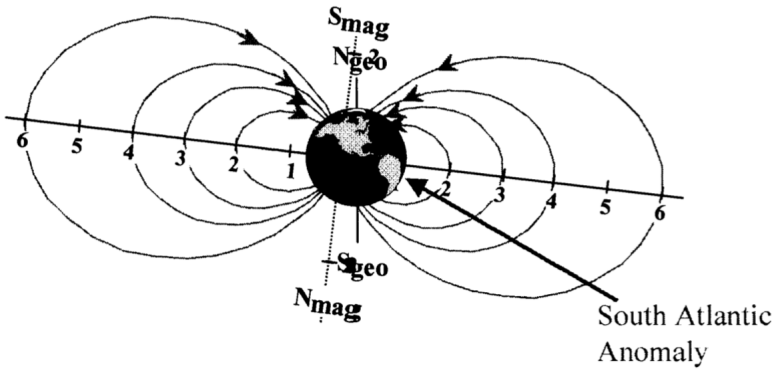


Figure 1.7: Dipolar magnetic field tilted and off-center with respect to Earth. [15]

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 intersection 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.[15]

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.

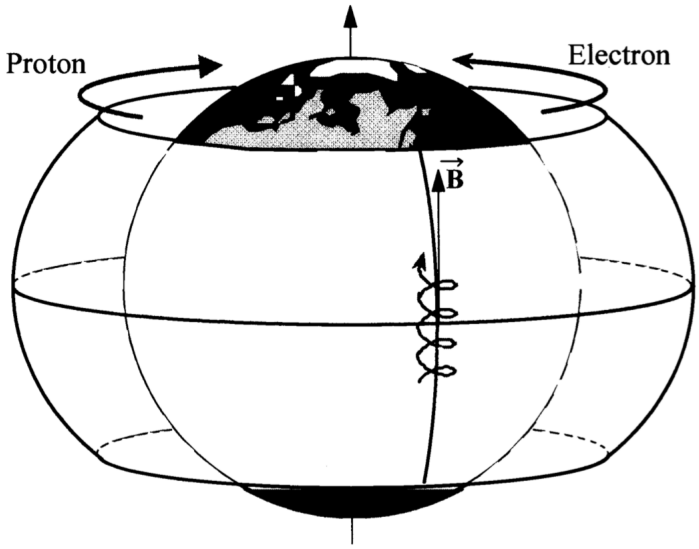


Figure 1.8: Composition of a charged particle's three periodic movements: gyration, bounce and drift. The particle then follows a torus surface called a drift shell. [15]

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 enables,

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 outer 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.

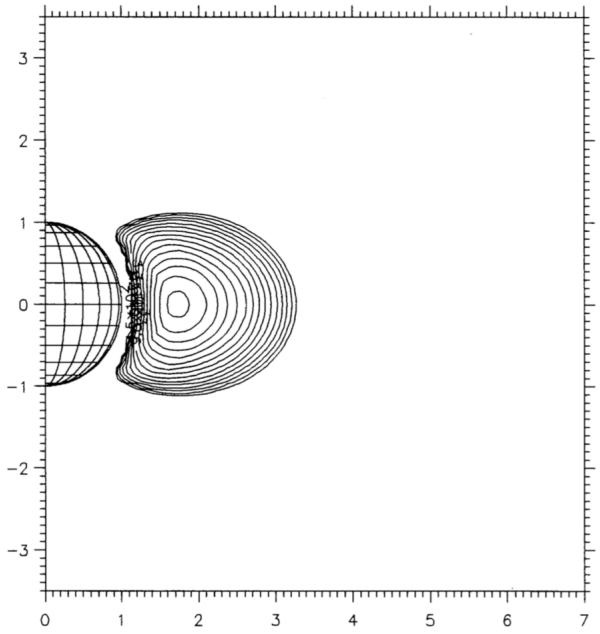


Figure 1.9: Proton radiation belt [15]

Dynamic of the radiation 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:

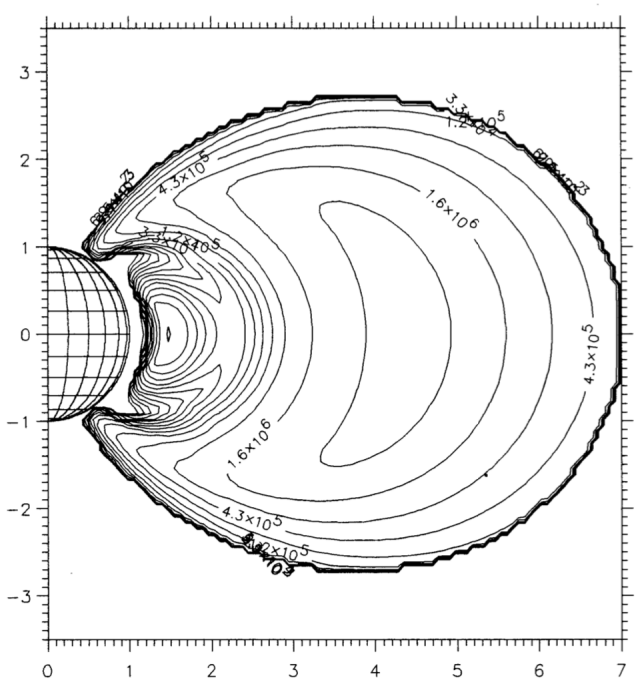


Figure 1.10: Electron radiation belt. [15]

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 absorption 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

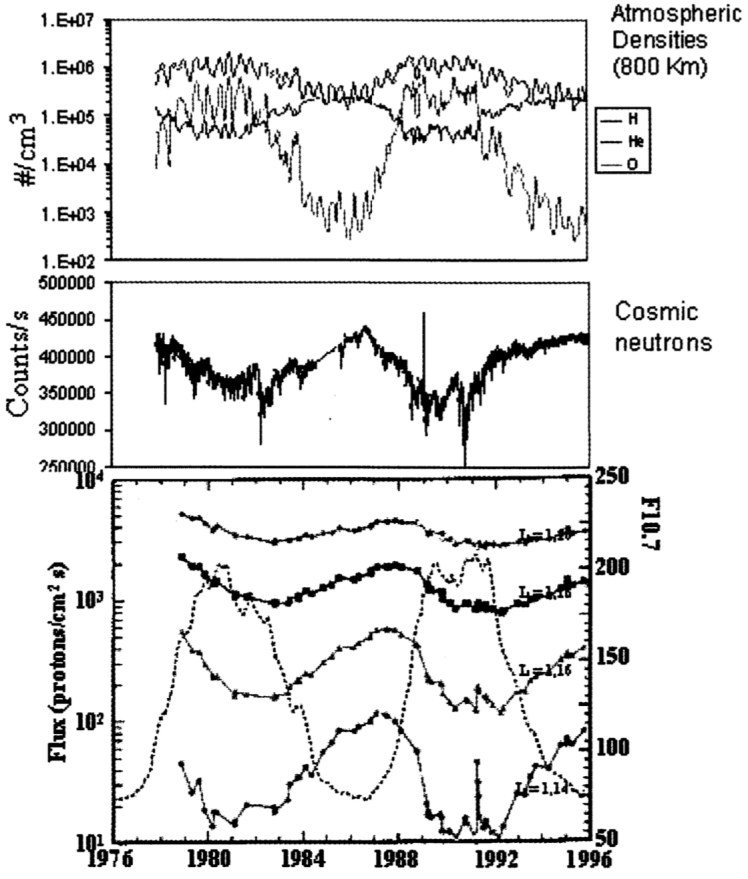


Figure 1.11: Changes in the proton fluxes at low altitudes (bottom), in the cosmic radiation (middle) and atmospheric densities (top) as a function of the solar cycle. [15]

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.

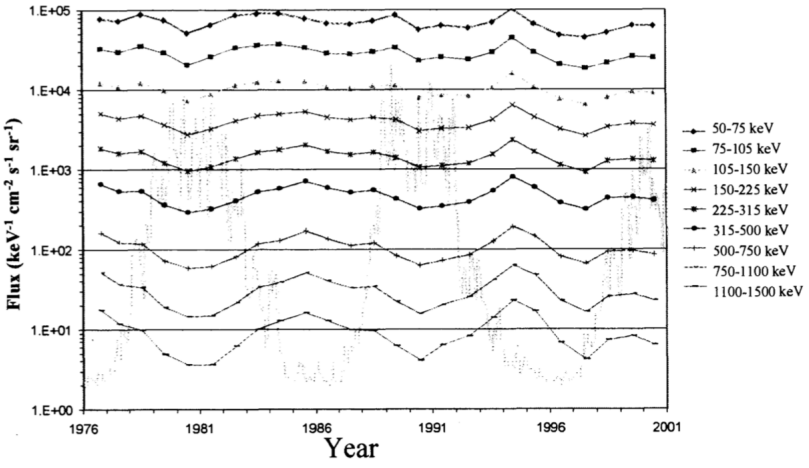


Figure 1.12: Electron fluxes at geostationary orbit as a function of the solar cycle. [15]

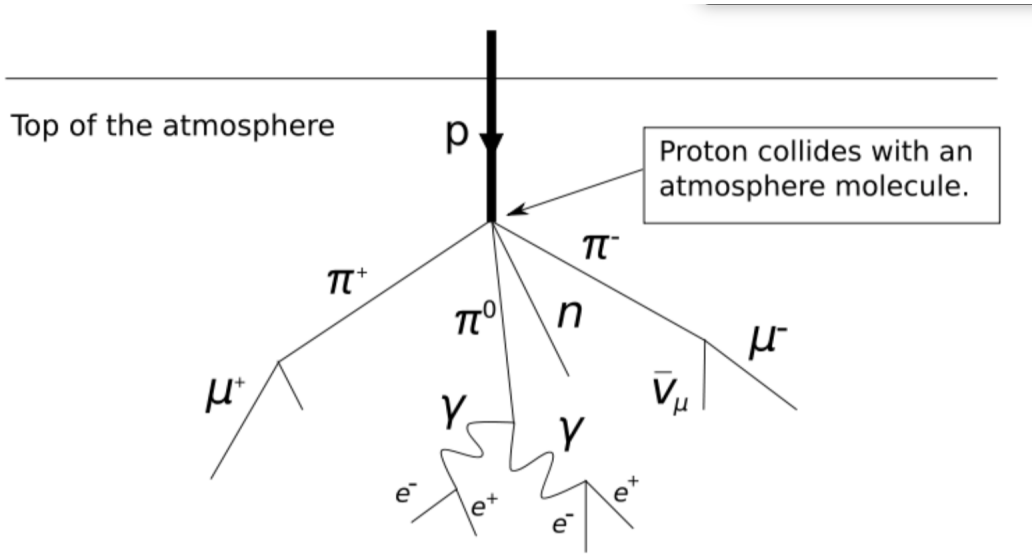


Figure 1.13: Example of Proton Scattering [65]

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)[65]. 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[83].

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[84].

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) [65]. 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; tunneling at junctions is enabled. In addition, radiation-induced defects reduce the carrier mobility and can exhibit meta-stable configurations[**bib10**].

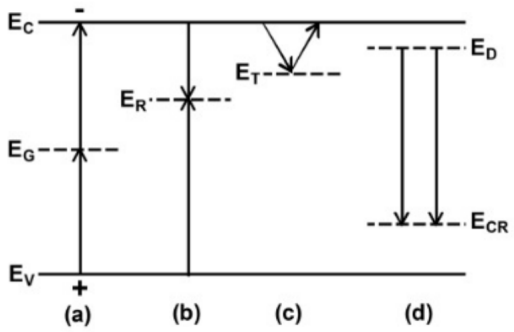


Figure 1.14: Radiation-Induced energy

Figure 1.14 illustrates radiation-induced energy levels in the Si band-gap 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 destructive 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 Stochastic, 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[65]

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 depletion region 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 current 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 length 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 over-current. 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

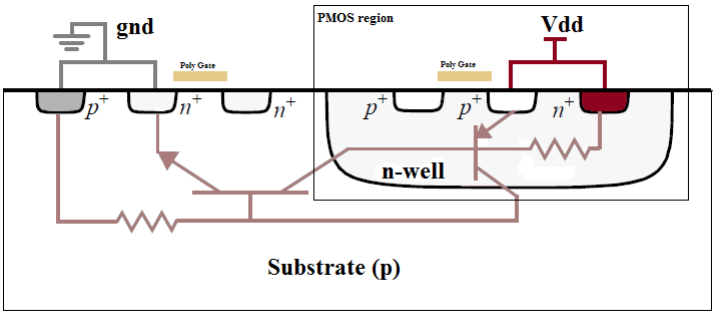


Figure 1.15: Latch-up Diagram

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.

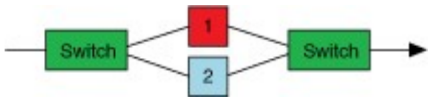


Figure 1.16: Cold Standby scheme

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.

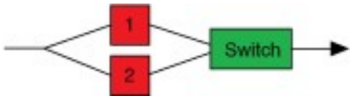


Figure 1.17: Hot Standby Schema

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 bump-less 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.

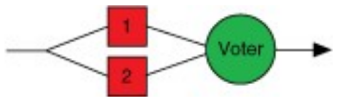


Figure 1.18: DMR scheme

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.

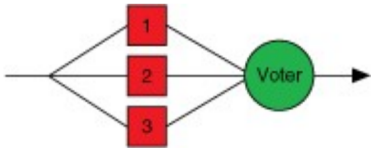


Figure 1.19: TMR Schema

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.

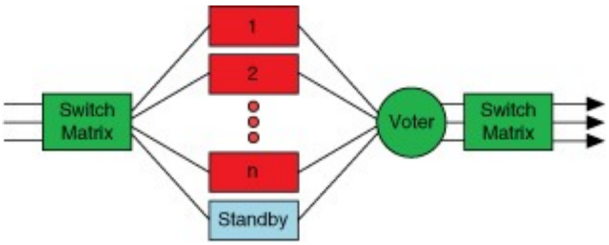


Figure 1.20: Example of Proton Scattering [65]

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 useful for this thesis to better understand the real advantages 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:

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

It is worth to underline that even though these methods seem to solve the problem of the harsh environments explained in this chapter, they're not the most efficient way of mitigating risks, as well as they are not enough to cover other types of failures. In particular, the assessment of the reliability is a much more complex matter, following precise international standards and procedures, as shown in the following chapters.

Cross layer reliability evaluation is also something that has been taken into account when performing the reliability assessment of certain systems, more information can be found at [32]

Chapter 2

ISO26262

2.1 introduction

The *ISO 26262* series of standards is the adaptation of *IEC 61508* series of standards to address the sector specific needs of electrical and/or electronic (E/E) systems within road vehicles. This adaptation applies to all activities during the safety limescale of safety-related systems comprised of electrical, electronic and software components. Safety is one of the key issues in the development of road vehicles. Development and integration of automotive functionalities strengthen the need for functional safety and the need to provide evidence that functional safety objectives are satisfied. With the trend of increasing technological complexity, software content and mechatronic implementation, there are increasing risks from systematic failures and random hardware failures, these being considered within the scope of functional safety. ISO 26262 series of standards includes guidance to mitigate these risks by providing appropriate requirements and processes.

To achieve functional safety, the ISO 26262 series of standards:

1. provides a reference for the automotive safety life cycle and supports the tailoring of the activities to be performed during the limescale phases, i.e., development, production, operation, service and decommissioning;

2. provides an automotive-specific risk-based approach to determine integrity levels [Automotive Safety Integrity Levels (ASILs)];

3. uses ASILs to specify which of the requirements of ISO 26262 are applicable to avoid unreasonable residual risk;

4. provides requirements for functional safety management, design, implementation, verification, validation and confirmation measures; and

5. provides requirements for relations between customers and suppliers.

The ISO 26262 series of standards is concerned with functional safety of E/E systems that is achieved through safety measures including safety mechanisms. It also provides a framework within which safety-related systems based on other technologies (e.g. mechanical, hydraulic and pneumatic) can be considered. The achievement of functional safety is influenced by the development process (including such activities as requirements specification, design, implementation

The current ISO 26262 standard consists of 10 parts or phases: Vocabulary (Part 1) [48], Management of Functional Safety (Part 2) [49], Concept Phase (Part 3) [50], Product Development at System Level (Part 4) [51], Hardware Level (Part 5) [52] and Software Level (Part 6) [53], Production and Operation (Part 7) [54], Supporting Processes (Part 8) [55], ASIL oriented and safety oriented analyses (Part 9) [56] and Guideline on ISO 26262 (Part 10) [57]. Based on Part 2, different phases of the product development lifecycle are assigned corresponding safety roles.

In order to categorize elements into Automotive Safety Integrity Levels (ASILs), it is essential to perform a comprehensive Hazard Analysis and Risk Assessment (HARA). A systematic approach involves the maintenance of a tabular record comprising all conceivable hazardous incidents. These incidents are subsequently categorized according to criteria such as: the likelihood of the event's manifestation, the extent of human intervention required to avert a potential accident upon its occurrence, and the probable gravity of the ensuing damage or harm.

Each ASIL not only establishes a methodology for error detection and management to minimize and accept residual risk, but also outlines validation procedures encompassing scrutiny and evaluation. A practical application of ISO 26262 is exemplified in [101] through the implementation of a "Fuel Level Display (FLD) system." The authors of [39] present a safety-focused process line-oriented methodological framework, which facilitates the analysis of similarities and disparities among various safety models, thereby enablin g the repurposing and development of an adaptable model.

2.2 Structure of the Standard

2.2.1 Vocabulary

One of the challenges in acquainting oneself with any new standard is becoming familiar with the specific jargon. ISO 26262 introduces some unique terms not

found in IEC 61508. However, many of these terms are at least approximately analogous to familiar IEC 61508 terms. Some of the key vocabulary additions include:

- **Automotive Safety Integrity Level (ASIL)** – One of four qualitative levels (A-D), notionally corresponding to an IEC 61508 high demand SIL, but not equivalent. Can be applied to a Safety Goal or an individual element.
- Functional safety requirements are specified hierarchically, from most general to most specific:
  - **Safety Goal** – Roughly equivalent to a risk assessment recommendation. Safety Goals are assigned ASILs, but these ASILs may be achieved by multiple systems. Roughly equivalent to the "IPL gap" in a LOPA.
  - **Functional Safety Concept** – Identifies the functional safety requirements necessary to achieve the safety goal (i.e., close the gap) and allocates to elements (i.e., functions). Similar to allocating SILs to SIFs to close a risk assessment gap.
  - **Functional Safety Requirement** – Specification of implementation-independent safety behavior. Sort of like the definition of the safe state and required actions without specifying hardware.
  - **Technical Safety Requirement** – Specifies detailed requirements for detection of faults, self-testing, responses to inputs, response times, etc.
  - **Hardware / Software Safety Requirements** – Specifies the detailed performance requirements and architecture requirements. Includes the probabilistic metrics.
- There is a hierarchy of equipment comprised of, from top to bottom:
  - **Item** – A system or array of systems that implements a function for the vehicle, e.g., the steering system. Similar to the IEC Equipment Under Control and the Control System. Items have Safety Goals with assigned ASILs.
  - **System** – A set of elements that relates at least a sensor, controller, and actuator.
  - **Element** – A collection of components that make up a system or part of a system.

  - **Component** – Logically and technically separable part made up of hardware parts (or software units).
  - **Hardware part** – A piece of hardware which cannot be subdivided.

- Faults and failures use a somewhat different nomenclature:
  - **Perceived Fault** – Detected by the driver (we might call this operator surveillance or response to alarm).
  - **Permanent Fault** – Fault that stays until repaired (the default in SIS).
  - **Residual Fault** – A "portion of a fault" that leads to system failure and is not covered by a safety mechanism. Similar to IEC 61508 Dangerous Undetected (DU) failure where some failures are detected by diagnostics and converted to Dangerous Detected (DD) failures.
  - **Single Point Fault** – A "portion of a fault" that leads to system failure and is not covered by a safety mechanism. Similar to an IEC 61508 DU failure where Hardware Fault Tolerance (HFT) = 0.
  - **Dual / Multiple Point Fault** – ISO doesnot use the term Hardware Fault Tolerance, but instead talks about single-, dual-, and multiple-point failures. Similar to a DU failure with HFT > 0.
  - **Safe Fault** – Equivalent to Safe Undetected (SU) and Safe Detected (SD) failures.
  - **Detected Fault** – Fault that is detected by a safety mechanism.

- **Dedicated measure** – There is not really a single term for this in IEC, but includes things to increase confidence in failure rates, like derating, separation, supplier monitoring, etc.

- **Safety Mechanism** – Technical solution to detect faults or control failures. Includes sensors, diagnostics, self-tests, etc.

- **Fault reaction time** – Time from detection of a fault to reaching the safe state. Roughly equivalent to SIF response time.

- **Fault Tolerant Time Interval** – Time from the fault to the hazardous event. Equivalent to Process Safety Time.

The above is not an exhaustive list, but it is sufficient to begin understanding the similarities and differences between the standards without being hopelessly lost in new jargon. (Now I know how the non-SIS folks feel when we start talking about SIFs, SIL, PFD, HFT, DU, DD, etc.)

Many familiar IEC 61508 terms are also recognizable in ISO 26262, including common cause failures, systematic failures, diagnostic coverage, proven-in-use, and validation, to name a few. Some important IEC 61508 terms (e.g., hardware fault tolerance, verification) are missing from the definitions list, but we will see later that these concepts are still present in the ISO standard.

2.2.2 Management of Functional Safety

Part 2 of the ISO standard delineates the process for managing functional safety and introduces the automotive safety lifecycle, as illustrated in the figure below (click to view a larger version).

A few quick observations on similarities and differences:

- Terms such as safety lifecycle, hazard analysis, validation, and functional safety assessment should be reassuringly familiar.
- Item definition can be considered similar to the process design. One must have some level of process design definition before performing the risk assessment.
- A key difference in the ISO standard is the distinction between production and operation, as it deals with mass-produced products (i.e., cars) rather than unique process plants.
- The boxes for Allocation to other technologies, Controllability, and External Measures may appear unusual until it becomes clear that, unlike IEC 61508, the ISO standard mandates a specific risk assessment process within the standard.
- Operation and Production planning are analogous to clause 16 in IEC 61511 but make the aforementioned distinction between production and operation.

The remainder of the document discusses other concepts familiar to IEC 61508 and IEC 61511 users, including safety culture, competence management, functional safety planning, and verification.

The terminology for verification processes deviates slightly from IEC, but the underlying concepts remain very similar. Part 2 outlines the following verification activities:

- Confirmation reviews are intended to check the compliance of selected work products with the corresponding requirements of ISO 26262.
- Functional safety audit evaluates the implementation of the processes required for functional safety activities.
- Functional safety assessment evaluates the functional safety achieved by the item.

Further detail on verification is provided in Part 2 Appendix D, which details a list of required verifications.

2.2.3 Concept Phase

Upon completing the development of the Functional Safety Concept in the third segment, the fourth segment outlines the necessary elements for the Technical Safety Requirements Specification (TSRS). This specification addresses various aspects, such as:

- Protective mechanisms, encompassing fault detection and signaling, steps to reach a secure condition, degradation reasoning, and tests to avoid hidden faults
- Reasoning for transitioning to a secure state (akin to Cause and Effect) and maintaining that state Temporal restrictions, including fault-tolerant time intervals and emergency operation periods

The TSRS clarifies the functional intricacies of the safety systems' responsibilities. As such, this is where ASIL Decomposition is presented, which involves dividing a Safety Goal's ASIL requirement into multiple lower ASILs assigned to distinct components. This process is comparable to using "stacked SIFs," or combining two SIL1 levels to achieve SIL2.

Moreover, this segment details specific strategies to manage both systematic and random hardware failures. This approach is fundamentally similar to the qualitative techniques and guidelines provided by IEC standards for managing systematic failures.

- **Severity classes** : The consequence of an anomaly is classified on a scale of $S0$ – $S3$ (Table 1)
- **Probability of Exposure** : The likelihood is classified on a scale of $E0$ – $E4$ (Table 2)

- **Controllability** : The likelihood that the driver can mitigate the anomaly is classified on a scale of $C0 - C3$. Similar to a conditional modifier or operator response to alarm. (Table 3)

It is important to note that ASILs are assigned to safety goals, which are approximately equivalent to HazOp or LOPA recommendations. This can be somewhat confusing since we will later observe that ASILs can be "decomposed" so that multiple elements can be combined to meet the ASIL requirement.

This section outlines the required content of the Functional Safety Concept, which is sometimes referred to as the initial SRS, i.e., the SRS encompassing the basic safety requirements without any design specifics. The section also introduces the Technical Safety Requirements Specification, roughly analogous to a detailed SRS as one might produce in a FEED package.

The hierarchical organization of safety requirements specifications in ISO 26262 is among their most innovative features. It has long been my belief that the SRS definition in IEC 61511 inadequately represented the evolving nature of the SRS as a set of documents that develops throughout the safety lifecycle. The ISO 26262 approach provides much-needed structure to the SRS development, and the process industries should take this approach into consideration.

2.2.4 Product development at the system level

Upon the development of the Functional Safety Concept in Part 3, Part 4 outlines the requirements for the Technical Safety Requirements Specification (TSRS). This specification addresses various aspects, such as:

- Safety mechanisms, encompassing fault detection and signaling, steps to reach a secure condition, degradation reasoning, and tests to avoid hidden faults
- Reasoning for transitioning to a secure state (akin to Cause and Effect) and maintaining that state
- Temporal restrictions, including fault-tolerant time intervals and emergency operation periods

The TSRS clarifies the functional intricacies of the safety systems' responsibilities. As such, this is where ASIL Decomposition is presented, which involves dividing a Safety Goal's ASIL requirement into multiple lower ASILs assigned to distinct components. This process is comparable to using "stacked SIFs," or combining two SIL1 levels to achieve SIL2.

Moreover, this segment details specific strategies to manage both systematic and random hardware failures. This approach is fundamentally similar to the qualitative techniques and guidelines provided by IEC standards for managing systematic failures.

2.3 V model and its Parts

In addition to the V-model, shown in Fig:2.2, the **ISO 26262** series of standards includes a variety of other concepts and processes, such as hazard analysis and risk assessment, functional safety requirements, and software development processes. These concepts and processes are used to ensure that the development of safety-related systems is carried out in a structured, systematic, and consistent manner.

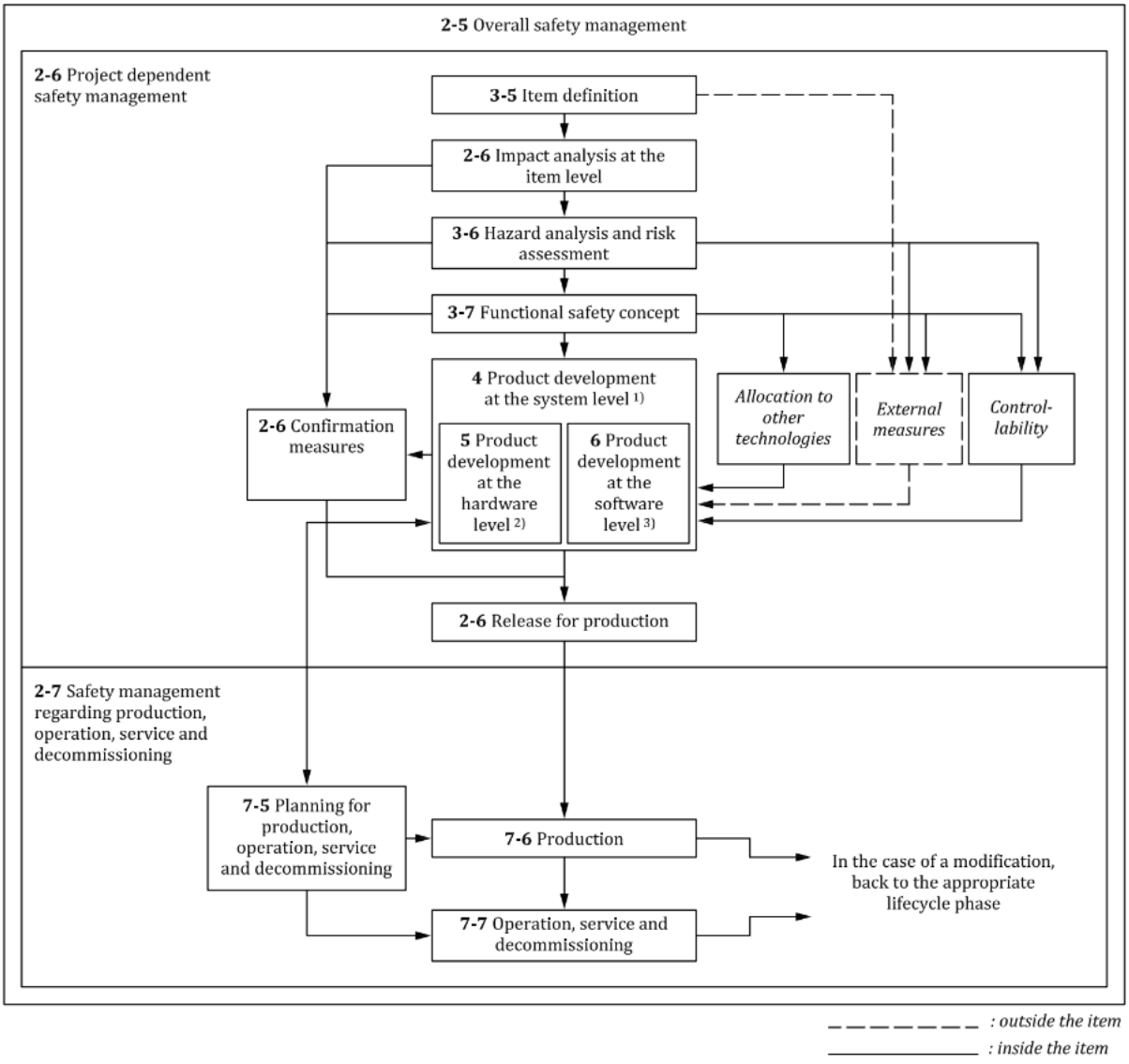


Figure 2.1: Safety Lifecycle Figure

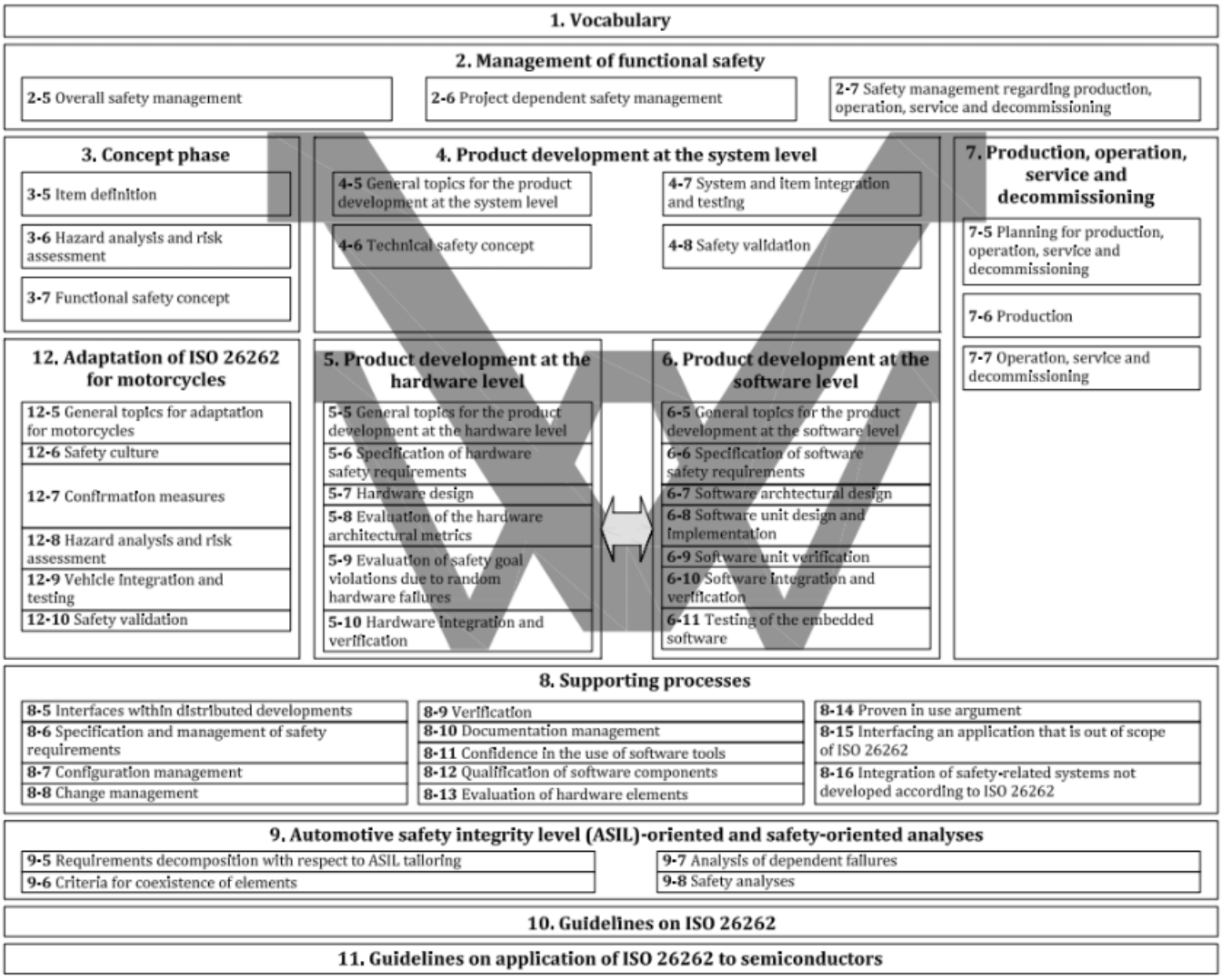


Figure 2.2: Overall structure of the **ISO 26262** series of standards

Overall, the **ISO 26262** series of standards plays a critical role in ensuring the safety and reliability of road vehicles. By providing guidance on the development of functional safety measures, it helps to minimize the risk of accidents and injuries caused by malfunctions in E/E systems.

2.4 Scope

This standard is intended to be applied to safety critical systems which may include more electrical and/or electronic [E/E] subsystems, usually installed in the vast majority of vehicles that are mass produced. This standard does not only address these E/E systems in special vehicles such could be the ones for disabled drivers. [1]

[1] Other dedicated application-specific safety standards exist and can complement ISO26262 series of standards or vice versa.

It is clear that all the systems and products open for commerce at the time of release of the this issue of the standard are exempted from complying to the standard itself. It does worth to notice that, even those systems are extempted, their modifications are not, and must comply to the latest issue of the standard.

In the exploration of the standard it will be possible to explore the different possible hazard causes by malfunctioning behaviour of safety related E/E systems, including interactions of these systems. It will not address hazards related to electric shock, fire, smoke, heat, radiation, toxicity, flammability, reactivity, corrosion, relead of energy or similar hazards, unless directly caused by malfunctioning behaviour of safety related E/E systems.

in particular this standard specifies the requirements for product development at the hardware level for automotive applications, including the following:

- General topics for the product development at the hardware level;
- Specification of hardware safety requirements;
- Hardware design;
- Evaluation of the hardware architectural metrics;
- Evaluation of safety goal violations due to random hardware failures; and
- Hardware integration and verification.

The requirements of this document for hardware elements are applicable to both non-programmable and programmable elements, such as ASIC, FPGA, and PLD, as specified in ISO 26262-10:2018 and ISO 26262-11:2018[2].

[2] Check annex A of the standard for further information

Other Section of interest The following section of the same standard are of interest of for this thesis:

- ISO 26262-1, Road vehicles - Functional safety - Part 1: Vocabulary [48]
- ISO 26262-2:2018, Road vehicles - Functional safety - Part 2: Management of functional safety [49]
- ISO 26262-4:2018, Road vehicles - Functional safety - Part 4: Product development at the system level [51]
- ISO 26262-6:2018, Road vehicles - Functional safety - Part 6: Product development at the software level [53]
- ISO 26262-7:2018, Road vehicles - Functional safety - Part 7: Production and operation [54]
- ISO 26262-8:2018, Road vehicles - Functional safety - Part 8: Supporting processes [55]
- ISO 26262-9:2018, Road vehicles - Functional safety - Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses [56]

  The missing part is the fifth, which it will be pictured in the course of this chapter.

2.5 Overview of the standard form the Automotive point of view

According to WHO "1.2 million people die each year on the world's roads, with millions more sustaining serious injuries and living with long-term adverse health consequences. Globally, road traffic crashes are a leading cause of death among young people, and the main cause of death among those aged 15–29 years [104]." Thus, road safety is a critical issue. IEC 61508 defines Safety as "the freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment [10]".

2.6 Main Examples of ASIL-D Chips

In order to understand the difficultied of certifing a device in the ASIL-D level, there is the need to first understand the metrics defined by the standard itself.

Part 5 of the standard is dedicated to the development of the hardware required to achieve safety goals, while the software aspect is covered in the subsequent part. In this section, the technical safety requirements developed in Part 4 are allocated to specific hardware and software designs. This process can be considered analogous to detailed engineering in a typical IEC 61511 project.

Although a comprehensive explanation is beyond the scope of this text, the ISO standard necessitates that the design takes into account several factors, including:

- Response to failures, including transient faults
- Diagnostic capabilities
- Consideration of fault detection times
- Expected failure rates of components
- Design verification requirements
- Consistency with the higher-level safety specifications

The hardware detailed design is documented in three primary deliverables:

1. Hardware Safety Requirements Specification
2. Hardware-software Interface Specification
3. Hardware Safety Requirements Verification Report

It is worth noting that the ISO standard does not provide extensive details on the hardware design process, and neither does IEC 61508. Consequently, this text will not delve further into this topic.

The latter sections of Part 5 address the quantitative verification of the hardware via various metrics, which will be the primary focus of this section.

Clause 8 of the ISO 26262 standard addresses the evaluation of hardware architectural metrics. These metrics are specifically designed to assess the effectiveness of the hardware architecture in managing random failures. It is important to distinguish this from the evaluation of random hardware failures (i.e., ASIL), which is covered in Clause 9.

Diagnostic coverage is defined similarly to its definition in the IEC standards. The estimation of diagnostic coverage must be based on failure rates from recognized industry sources, statistics derived from field returns or tests, or expert judgment.

This part of the standard defines two primary metrics:

1. Single-point Fault Metric – measures the robustness of the design concerning single-point and residual faults. A higher value is more desirable.

2. Latent-fault Metric – measures the robustness of the design concerning latent faults. A higher value is more desirable.

It is important to note that these metrics are only applicable to higher ASIL functions (i.e., B, C, or D).

Before discussing these metrics, it is beneficial to recall the taxonomy of faults/failures (from Part 1) utilized by ISO 26262, which differs from the IEC classification. The total failure rate λ can be decomposed into:

$$\lambda = \lambda_{SPF} + \lambda_{RF} + \lambda_{MPF} + \lambda_S \tag{2.1}$$

where:

- λ_{SPF}: Single Point Faults (i.e., a DU fault where there are no diagnostics)
- λ_{RF}: Residual Faults (i.e., a DU fault not covered by diagnostics)
- λ_{MPF}: Multiple Point Faults (i.e., a combination of independent SPFs)
- λ_S: Safe Faults

The single-point fault metric is defined as the sum of the multiple-point faults and the safe faults divided by the total failure rate, that is, the following ratio: $\sum(\lambda_{MPF} + \lambda_S)/\sum(\lambda)$

- Note: The name "single-point fault metric" may initially be confusing, as the single-point fault rate (λ_{SPF}) does not appear in the formula! However, the formula can equivalently be written as: $1 - \sum(\lambda_{SPF} + \lambda_{RF})/\sum(\lambda)$

This ratio bears a striking resemblance to the IEC 61508 concept of Safe Failure Fraction (SFF), with the notable exception that multiple-point faults are also considered "safe." The inclusion of multiple-point faults is a somewhat unusual approach, but it likely explains why the latent-fault metric is also calculated.

A quantitative target for the single-point fault metric is established by the standard based on the ASIL target:

	ASIL B	ASIL C	ASIL D
Single-point fault metric	≥90 %	≥97 %	≥99 %

Figure 2.3: Possible Source for the Derivation of the Target "Single Point of Failure" metric

By combining safe faults and multiple-point faults into the same metric, this metric exhibits a similar impact to the SFF-based hardware fault tolerance requirements in IEC 61508. In other words, if the single-point fault metric is too low, additional fault tolerance will convert those faults into multiple-point faults, thereby improving the metric.

Latent Fault metric The latent fault metric is defined as the sum of the multiple-point faults perceived by the driver or detected by diagnostics, in addition to the safe faults, divided by the total multiple-point and safe faults. The formula for this ratio is as follows:

$$\frac{\sum(\lambda_{MPF}(Per/Det) + \lambda_S)}{\sum(\lambda_{MPF} + \lambda_S)} \tag{2.2}$$

This concept bears a strong resemblance to the IEC 61508 diagnostic coverage notion, with the key difference being the inclusion of driver "perception" of, and response to, faults. This aspect is generally not considered in IEC 61508 (or IEC 61511) as most systems are dormant and activate on demand.

A quantitative target value for the latent-fault metric is established by the standard based on the ASIL target:

	ASIL B	ASIL C	ASIL D
Latent-fault metric	≥60 %	≥80 %	≥90 %

Figure 2.4: Possible Source of Derivation for the "Latent Fault" metric

2.6.1 Automotive Safety Integrity Level (ASIL)

Clause 9 introduces the concept of Automotive Safety Integrity Level (ASIL) and outlines the process for its evaluation. Similar to the IEC 61508 Safety Integrity Level (SIL), the objective of the ASIL calculation is to demonstrate that the probability of random failures is sufficiently low to meet the safety goal.

The ISO standard explicitly states that the evaluation is limited to random hardware failures, excluding systematic failures. As regular readers of my other blog already know, I strongly disagree, and believe that systematic failures should be both qualitatively and quantitatively considered wherever possible.

However, this digression aside, two options are provided in Part 5 for calculating ASIL:

1. Probabilistic Metric for random Hardware Failures (PMHF)
2. Individual evaluation of faults (i.e., cut set analysis)

Each approach will be briefly discussed in the following sections.

Probabilistic Metric for random Hardware Failures (PMHF) This method is akin to the IEC 61508 approach for a high demand or continuous SIF, where a target probability of failure per hour (PFH) is specified for each SIL level. The ASIL[3] targets are shown below:

[3]

ASIL	Random hardware failure target values
D	$<10^{-8}$ h^{-1}
C	$<10^{-7}$ h^{-1}
B	$<10^{-7}$ h^{-1}
NOTE The quantitative target values described in this table can be tailored as specified in 4.1 to fit specific uses of the item (e.g. if the item is able to violate the safety goal for durations longer than the typical use of a passenger car).	

Figure 2.5: Possible Source of Derivation for the "Latent Fault" metric

It is important to note that for ASIL:

- No quantitative target or calculations are required for ASIL A.
- The PFH targets for ASIL B and C are the same (though other requirements differ).
- The PFH values for ASIL do not exactly align with the targets for SIL. The ASIL targets are higher; for instance, ASIL B/C is equivalent to the PFH for SIL3, while ASIL D is equivalent to SIL4. (Although ASIL A does not have a quantitative target, it is presumably equivalent to SIL2 if the risk matrix is linear.)

Since most of us do not perform IEC 61508 continuous mode SIL calculations regularly, the continuous mode SIL table from IEC 61508 is provided below for reference:

CONTINUOUS MODE OR DEMAND MODE OF OPERATION	
Safety integrity level (SIL)	Average frequency of dangerous failures (failures per hour)
4	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-6}$ to $< 10^{-5}$

Figure 2.6: Possible Source of Derivation for the "Latent Fault" metric

The ISO 26262 probabilistic metric evaluation must be conducted quantitatively, although the standard does not prescribe a specific method. Quantitative fault tree analysis (FTA) is suggested as one possible approach. The standard does stipulate that the analysis must cover certain requirements, including the concept of "exposure duration," which is roughly equivalent to the MTTR considered in SIL calculations.

For higher ASIL C and D targets, the standard mandates that a single-point fault in a hardware part shall only be considered acceptable if "dedicated measures" are taken, where dedicated measures refer to steps that ensure a low failure rate, such as over-design, separation, sampling, etc. This concept bears some similarity to the proven-in-use concepts of IEC 61508.

Individual evaluation of faults (i.e., cut set analysis) The second option for evaluating ASIL is based on assessing faults individually. A simple flowchart is provided in the standard to describe the iterative design process. The advantage of this approach appears to be that a complex model (e.g., FTA) of the entire system is not required for the analysis. It is speculated (though not confirmed) that the downside is that this method may be more conservative.

This method introduces the concept of failure rate class for individual hardware parts. The failure rate class ranking for a hardware part failure rate is determined as follows:

- Class 1 = $< 10^{-10}$/hr
- Class 2 = $< 10^{-9}$/hr
- Class 3 = $< 10^{-8}$/hr
- etc. (where Class i = Class $i - 1 \times 10$)

In this method, each ASIL level has one or more failure rate classes that are permitted to be used, as shown below. There is one table each for (i) single-point faults (i.e., no diagnostics), (ii) residual faults, and (iii) dual-point faults, as shown below:

ASIL of the safety goal	Failure rate class
D	Failure rate class 1 + dedicated measures[a]
C	Failure rate class 2 + dedicated measures[a] or Failure rate class 1
B	Failure rate class 2 or Failure rate class 1
[a] The note in requirement 9.4.2.4 gives examples of dedicated measures.	

Figure 2.7: Target of Failure Rate Classes of Hardware Parts Regarding Single Point Faults

ASIL of the safety goal	Diagnostic coverage with respect to residual faults			
	≥99,9 %	≥99 %	≥90 %	<90 %
D	Failure rate class 4	Failure rate class 3	Failure rate class 2	Failure rate class 1 + dedicated measures[a]
C	Failure rate class 5	Failure rate class 4	Failure rate class 3	Failure rate class 2 + dedicated measures[a]
B	Failure rate class 5	Failure rate class 4	Failure rate class 3	Failure rate class 2
[a] The note in requirement 9.4.2.4 gives examples of dedicated measures.				

Figure 2.8: Mazimum Failure Rate Classes for a Given Diagnostic Coverage of the Hardware Part - Residual Fault

ASIL of safety goal	Diagnostic coverage with respect to latent faults		
	>= 99 %	>= 90 %	<90 %
D	Failure rate class 4	Failure rate class 3	Failure rate class 2
C	Failure rate class 5	Failure rate class 4	Failure rate class 3

Figure 2.9: Target of Failure Rate Classe and Coverage of Hardware Parts Regarding Dual Point Faultsg:asil-t-9

This method of component "classes" bears conceptual resemblance to the "parts counts" methods employed in the past, most notably in MIL-HDBK-217. This fact alone is sufficient to evoke skepticism regarding this approach, considering the military eventually abandoned MIL-HDBK-217 due to its extreme inaccuracy.

One might also question whether the failure class targets specified in ISO 26262 are so conservative as to render this approach impractical. A Class 1 part, as required for ASIL D, must exhibit a failure rate of less than 1×10^{-10}/hr. Expressed differently, the MTTF for that part must exceed 1,141,000 years. It would certainly be intriguing to know the source of these exceptional Class 1 devices for safety functions! This stipulation may subtly encourage hardware fault tolerance without explicitly mandating it.

2.7 wWhy it may not be enough

Standards for functional safety, such as ISO 26262 and IEC 61508, assess a system's safety based on the existence or absence of intolerable and unreasonable risk. Consequently, these standards overlook minimal acceptable risks, resulting in a lack of assurance of absolute safety. The potential for human error in risk assessment judgments, due to the nature of these standards, may have severe consequences. In [60], the authors explore moral concepts and challenges associated with functional safety, as well as prevalent misconceptions regarding risk perception. They utilize Kahneman's book, "Thinking, Fast and Slow" [29], as the foundation for examining irrational risk judgment.

Critical ethical issues related to functional safety and risk-based decision-making processes are as follows:

- Diverse judgments: Variability exists among individuals' assessments of risk levels.
- Vision Zero and Zero Tolerance: These government principles, targeting behaviors that cause harm, are not applied to functional safety.
- Wants versus Needs: Activities such as flying and driving are human desires rather than necessities, raising questions about the justification for accommodating such desires in light of associated risks.
- Business: The primary objective for creating safe systems should be safety, not profit, yet the pursuit of profit often drives the development of safer systems.

Additional ethical considerations encompass law, regulations, policies, evolution, innovation, and sustainability.

Addressing these issues is of paramount importance, particularly in the context of emerging autonomous systems. Current standards may be insufficient, and future work in this domain should investigate these topics further.

Chapter 3

Failure Mode and Effect Analysis FMEA

3.1 Introduction

3.1.1 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 diagrams will be needed for the following:

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

3.1.2 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 maintainability (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, etc.);
4. an estimation of measures of the significance and probability of failure.

3.1.3 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.

3.2 Procedure

3.2.1 General

The wide variation in complexity of system designs and applications may require the development of highly individualised FMEA procedures consistent with the information available. Traditionally, there have been wide variations in the manner in which FMEA is conducted and presented. However, the analysis is usually done in a standard manner and presented on a worksheet that contains a core of essential information which can be developed and extended to suit the particular system or project to which it is applied. A typical example of a worksheet is shown in Figure 1.

The procedure consists of the following four main stages:

1. Preparatory definition of the system including the design, functional, operational, maintenance, and environmental requirements;

2. Establishment of the basic principles and purposes of the FMEA and the form of its presentation;

3. Carrying out the FMEA using the appropriate worksheet designed according to (a) and (b);

4. Reporting of the complete analysis including any conclusions and recommendations made.

A more detailed consideration of the information needed is given in the following sections.

3.2.2 Preparation

At the commencement of an analysis, the following preparations should be made:

1. The analyst should have available the information listed in Section 2.4.2.2 to 2.4.2.7 that clearly defines the system to be analysed.

2. It will usually be necessary for the analyst to translate the information into some form of functional, hierarchical, or reliability block diagrams. An example of a functional diagram is shown in Figure 2. This diagram shows how the failure effects at the part level form the failure modes at the module level, the failure effects at the module level form the failure modes at the subsystem level, and so on. Such a representation of the system should explicitly identify the system's functional structure, the system boundary, and the inputs and outputs crossing that boundary. Further information is given in Section 2.4.2.8 to 2.4.2.10.

3.2.3 FMEA principles

The following principles should be applied:

1. Define clearly the purposes and uses of the FMEA as indicated in Section 2.1.2.

2. Establish and define the relationships with other forms of reliability analysis with which the FMEA may subsequently be integrated. (See Section 2.3.5.)

3. Define the scope of the FMEA in relation to the functional structure and hierarchical structure of the system as described by the block diagrams referred to in Section 2.4.2.10. It is essential to define the lowest level in the system's hierarchical structure at which the analysis will start. The guidance given in Sections 2.3.4, 2.4.1, and 2.4.2.8 is especially important for this task.

4. Define the format of the FMEA worksheet to suit the project requirements. The core information considered essential is as follows:

1. The name of the item in the system being analysed;

2. Function performed by the item;
3. Identification number of the item;
4. Failure modes of the item;
5. Failure causes;
6. Failure effects on the system;
7. Failure detection methods;
8. Compensating provisions;
9. Severity of effects;
10. Remarks.

Other information required for the particular system and project needs to be defined by the analyst according to the purposes of the

3.3 Analysis

It is worth to underline that, even though the scope of this thesis is to establish a new methodology for automated FMEA in both hardware and software systems, there is the absolute need to comply with what is today the state of the art for FMEA and the guidelines to be followed to bring the metrics extracted during the analysis to certification. It is for this reason that the following section will present the traditional method applied today to all the electromechanical systems under evaluation. Bare in mind that the totality of the procedure described below is man driven.

The usual requirement and purpose of an FMEA is to identify the effect of oJl failure modes of all constituent items at the lowest level in the system.Tb achieve this the worksheet should be used in the following manner:

1. Identify all items in the system or subsystem,each of which is to have its failure modes and effects analysed. The system of identification by name and number should be such that no item swill be omitted.
2. Select the first item for analysis and enter the item name and identification number in the appropriate columns of the worksheet.Determine the function of that item in the system and enter that on the worksheet.

3. Deduce all the possible failure modes of the item due to any possible cause and individually enter these modes on the worksheet
4. Postulate the most likely failure causes for each failure mode of the item and enter these on the worksheet. It will usually not be possible to consider all possible causes because the range is so vast, but the most significant with regard to the item, the failure mode and the application should be identified.
5. Deduce the effects of the failure on the subsystem and system, as determined by the scope of the FMEA
6. Complete the remaining columns of the worksheet for the first failure mode of the first item.
7. Repeat 3 to 5 for all failure modes of the first item
8. Repeat 2 to 6 for all other items

3.3.1 Multiple Stages

If the FMEA is to be done in stages that each relate to separate Levels in the system's hierarchical structure, the failure effects from the lower level become the failure modes at the next level up. The analysis should then proceed as follows.

1. Identify the lower level FMEAs that are appropriate for the next stage in the system FMEA according to the system's hierarchical structure defined by the block or functional diagrams (see 2.2,2(b)). Where appropriate also include items defined as being at the lowest level in that part of the system structure
2. Perform the FMEA for each failure of each item at this higtrer level in the system structures given in the previous section.
3. repeat the two above steps for any further higher levels in the system structure.

3.3.2 Worksheet recommendations

The last worksheet entry should give any pertinent remarks to clarify other entries. Possible future actions such as recommendations for design improvements may be recorded and then amplified in the report. This column may also include the following:

(a) any unusual conditions;

(b) effects of redundant element failures;

(c) recognition of specially critical design features;

(d) any remarks to amplify the entry;

(e) references to other entries for sequential failure analysis;

(f) significant maintenance requirements;

(g) dominant failure causes;

(h) dominant failure effects;

(i) decisions taken, e.g. at design review.

The report on the FMEA (or FMECA) may be included in a wider study or may stand alone. In neither case, the report should include a summary and a detailed record of the analysis and the block or functional diagrams which define the system structure. The report should also contain a list of the drawings (including issue status) on which the FMEA is based.

The summary should contain a brief description of the method of analysis and the level to which it was conducted, the assumptions and the ground rules. In addition, it should include listings of the following:

1. recommendations for the attention of designers, maintenance staff, planners, and users;

2. failures which, when initially occurring alone, result in serious effects;

3. failures which have no effect;

4. design changes which have already been incorporated as a result of the FMEA (or FMECA).

3.3.3 Report on Analysis

The report on Failure Modes and Effects Analysis (FMEA) or Failure Modes, Effects and Criticality Analysis (FMECA) can be either a standalone document or a part of a broader study. In either case, the report should contain a summary and a detailed record of the analysis along with block or functional diagrams

FMEA

Indenture level:					Design by:			Prepared by:				
Sheet no:					Item:			Approved by				
Mission phase:					Issue:			Date:				
Item ref.	Item description-function	Failure entry code	Failure mode	Possible failure causes	Symptom detected by	Local effect	Effect on unit output	Compensating provision against failure	Severity class	Failure rate (F/Mhr)	Data source	Recommendations and actions taken

Figure 3.1: Example of FMEA Table

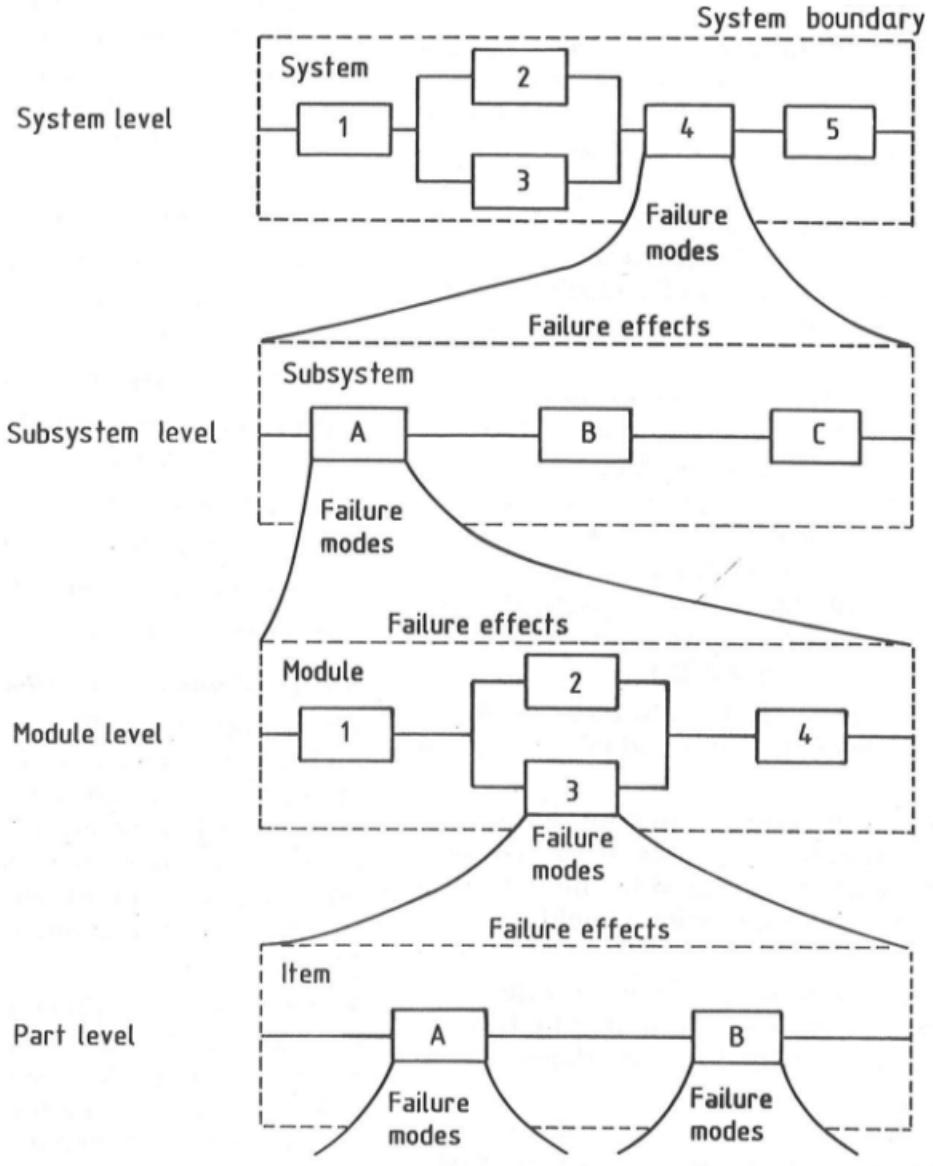


Figure 3.2: Criticality Grid

that define the structure of the system. Furthermore, the report should include a list of drawings (with issue status) on which the FMEA is based.

The summary section of the report should provide a brief explanation of the analysis method, the level to which it was carried out, the assumptions made, and the ground rules followed (see section 2.4.L). The summary should also include lists of the following:

1. Recommendations for designers, maintenance staff, planners, and users;
2. Failures that, when occurring alone, have significant consequences;
3. Failures that have no impact; and
4. Design changes that were made as a result of the FMEA (or FMECA).

3.4 Application

3.4.1 Field of application

FMEA is a method that is primarily adapted to the study of material and equipment failures and that can be applied to categories of systems based on different technologies (electrical, mechanical, hydraulic, etc.) and combinations of technologies. FMEA should also include the consideration of software and human performance where these are relevant to the reliability of the system. An FMEA can be a study for general use or it may be specific to particular pieces of equipment, to systems or to projects as a whole.

3.4.2 Application within a project

The user should determine how and for what purposes he uses FMEA within his own technical discipline. It may be used alone or to complement and support other methods of reliability analysis. The requirements for FMEA originate from the need to understand hardware behaviour and its implications for the operation of the system or equipment. The need for FMEA can vary widely from one project to another. FMEA is the principal reliability engineering activity in support of the design review concept (see 4.2.1.4 of BS 5760: Part 1: 1988) and should be put into use from the very first steps of system and subsystem design. FMEA is applicable to all levels of system design but is most appropriate for lower levels where large numbers of items are involved and/or there is functional complexity. Special training of personnel performing FMEA

is essential and they need the close collaboration of systems engineers and designers. The FMEA should be updated as the project progresses and as designs are modified. At the end of the project, FMEA is used to check the design and may be essential for demonstration of conformity of a designed system to the required standards, regulations, and user's requirements.

Information from the FMEA identifies priorities for statistical process control sampling and inspection tests during manufacture and installation and for qualification, approval, acceptance and start-up tests. It provides essential information for diagnostic and maintenance procedures for inclusion in handbooks.

In deciding on the extent and the way in which FMEA should be applied to an item or design, it is important to consider the specific purposes for which FMEA results are needed, the time phasing with other activities and the importance of establishing a predetermined degree of awareness and control over unwanted failure modes and effects. This leads to the planning of FMEA in qualitative terms at specified levels (system, subsystem, component, item) to relate to the iterative design and development process (see BS 5760: Part 1).

To ensure that it is effective, the place of FMEA should be clearly established in the reliability program, together with the time, manpower and other resources needed to make it effective. It is vital that FMEA is not abridged to save time and money. If time and money are short the FMEA should concentrate on those parts of the design which are new or are used in new ways. FMEA can be economically directed to areas identified as crucial by other methods of analysis, e.g. fault tree analysis (FTA).

3.4.3 Uses of FMEA

Some of the detailed applications and benefits of FMEA are listed below:

(a) to avoid costly modifications by the early identification of design deficiencies;

(b) to identify failures which, when they occur alone or in combination, have unacceptable or significant effects, and to determine the failure modes which may seriously affect the expected or required operation; [4]

[4] Such effects may include secondary failures.

(c) to determine the need for the following:

(1) redundancy;

(2) design improvement;

(3) more generous stress allowances (derating);

(4) screening of items;

(5) design of features that ensure that the system fails in a preferred failure mode, e.g. 'fail-safe' outcomes of failures;

(6) selection of alternative materials, parts, devices, and components;

(d) to identify serious failure consequences and hence the need for changes in design and/or operational rules;

(e) to provide the logic model required to evaluate the probability or rate of occurrence of anomalous operating conditions of the system in preparation for criticality analysis;

(f) to disclose safety hazard, and product liability problem areas, or non-compliance with regulatory requirements; **Note:** Frequently, separate studies will be required for safety, but overlap is inevitable and therefore cooperation is highly advisable.

(g) to ensure that the development test programme can detect potential failure modes;

(h) to focus upon key areas in which to concentrate quality control, inspection and manufacturing process controls;

(i) to assist in defining various aspects of the general maintenance strategy, such as:

(1) establishing the need for data recording and condition monitoring during testing, checking-out and use;

(2) provision of information for development of trouble-shooting guides;

(3) establishing maintenance cycles which anticipate and avoid wear-out failures;

(4) the selection of preventative or corrective maintenance schedules, facilities, equipment and staff;

(5) selection of built-in test equipment and suitable test points;

(j) to provide a systematic and rigorous approach to the study of the installation in which the system is embedded;

(k) to facilitate or support the determination of test criteria, test plans and diagnostic procedures, for example: performance testing, reliability testing;

(l) to identify parts and assemblies requiring worst case analysis (frequently required for failure modes involving parameter drifts);

(m) to support the design of fault isolation sequences and to support the planning for alternative modes of operation and reconfiguration;

(n) to facilitate communication between the following:

  (1) general and specialised engineers;

  (2) equipment manufacturer and his suppliers;

  (3) system user and the designer or manufacturer;

(o) to enhance the analyst's knowledge and understanding of the behaviour of the equipment studied;

(p) to provide designers with an understanding of the factors which influence the reliability of the system;

(q) to provide a final document that is proof of the fact that (and of the extent to which) care has been taken to ensure that the design will meet its specification in service. (This is especially important in the case of product liability

3.4.4 Limitations and Drawbacks

FMEA is extremely efficient when it is applied to the analysis of elements that cause a failure of the entire system or of a major function of the system. However, FMEA may be difficult and tedious for the case of complex systems that have multiple functions involving different sets of system components. This is because of the quantity of detailed system information that needs to be considered. This difficulty can be increased by the existence of a number of possible operating modes, as well as by consideration of the repair and maintenance policies. FMEA can be a laborious and inefficient process unless it is judiciously applied. The uses to which the results are to be put subsequently should be defined and FMEA should not be included in requirements specifications indiscriminately. Complications, misunderstandings and errors can occur when FMEA attempts to span several levels in a hierarchical structure if redundancy is applied in the system design. It is therefore preferable for an FMEA to be restricted to relating two levels only in the hierarchical structure. For example, it is a relatively straightforward task to identify failure modes of items and to determine their effects on the assembly. These effects then become the failure modes at the next level up, e.g. the module, and so on. However, successful

multi-level FMEAs are often carried out. FMEA is applicable to all levels of a system but is most appropriate to lower levels where large numbers of items are involved and/or there is functional complexity.

3.4.5 Relationships with Other Methods

FMEA (or FMECA) can be used alone. As a systematic inductive method of analysis, FMEA is most often used to complement other approaches, especially deductive ones. At the design stage, it is often difficult to decide whether the inductive or deductive approach is dominant, as both are combined in processes of thought and analysis. Where levels of risk are identified in industrial facilities and systems, the inductive approach is preferred and therefore FMEA is an essential design tool. However, it should be supplemented by other methods. This is particularly the case when problems need to be identified and solutions need to be found in situations where multiple failures and sequential effects need to be studied. The method used first will depend on the project programme.

During the initial phases of system design, which involve defining system functions, general structure, and subsystems, the successful performance of the system can be represented using a reliability block diagram or a fault tree that depicts a failure path. However, to assist in the creation of these diagrams for the system, it is recommended to apply the Failure Modes and Effects Analysis (FMEA) inductive process to the subsystems prior to their design. Under these circumstances, the FMEA approach cannot be a set procedure, but instead requires a thought process that is not easily expressed in a rigid tabular format. In general, for analysing a complex system that involves multiple functions, numerous items, and interrelations between these items, FMEA proves to be essential but not sufficient. Fault Tree Analysis (FTA) is a complementary deductive method that traces the low-level causes of a postulated high-level failure. Although the logical analysis can be, and sometimes is, used for purely qualitative analysis of fault sequences, it is typically a precursor to estimating the frequency of the postulated high-level failure. FTA focuses on the logic of coincident (or sequential) and alternative events causing undesirable consequences. The FMEA format is more descriptive. Both methods are useful in conducting a full analysis for safety and reliability in a complex system. However, if the system primarily relies on series logic, with few redundancies and few functions, then FIA may be an unnecessarily complicated way of presenting the logic and identifying the failure modes. In such cases, FMEA may suffice.

In other cases where FTA is preferred, it still needs to be enhanced with descriptions of the failure modes and effects. The main consideration in selecting

the method of analysis should depend on the particular requirements of the project, not only with regard to technical requirements but also timescale, cost, efficiency, and usage of the results. General guidelines are as follows:

1. FMEA is appropriate when comprehensive knowledge of the failure characteristics of an item is required.
2. FMEA is more appropriate for smaller systems, modules, or assemblies.
3. FMEA is an essential tool at the research and development or design stage when unacceptable effects of failures need to be identified, and solutions found.
4. FMEA can be necessary for items that are of innovative design so that their failure characteristics cannot be known from previous operational experience.
5. FMEA is usually more applicable to items having large numbers of components to be considered that are related by predominantly series failure logic.
6. FTA is generally more suitable for the analysis of single failure modes involving complex failure logic and redundancy. This would usually be so for the higher levels in the hierarchical structure of large systems or entire plants.
7. FTA can be used at the higher levels in the system structure early in the design stage and can help in identifying the need for detailed FMEA at lower levels during detailed design.

3.5 Supplementary Information

3.5.1 Establishment of Ground Rules

Levels of Analysis

It is important to determine the level in the system that will be used for the analysis. For example, systems can be broken down into subsystems, replaceable units, or individual components (see figure 2). Basic principles for selecting the system levels for analysis depend on the results desired and the availability of design information. The following guidelines are useful.

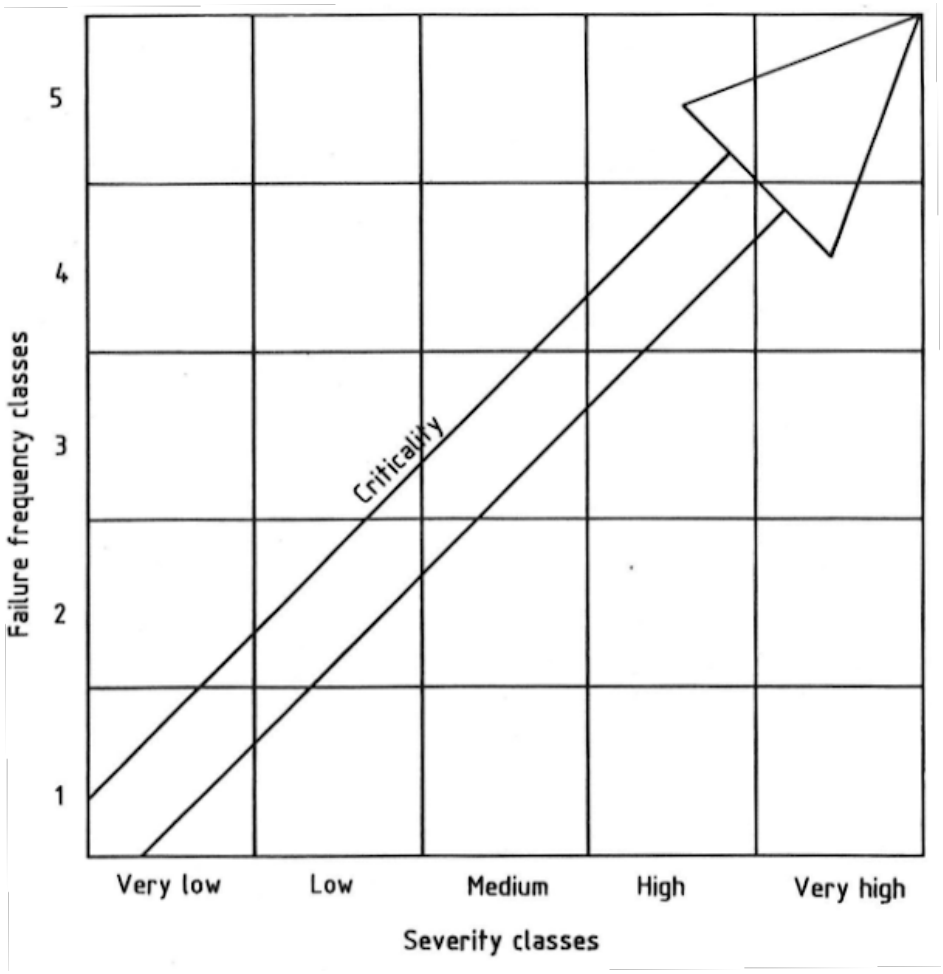


Figure 3.3: Criticality Grid

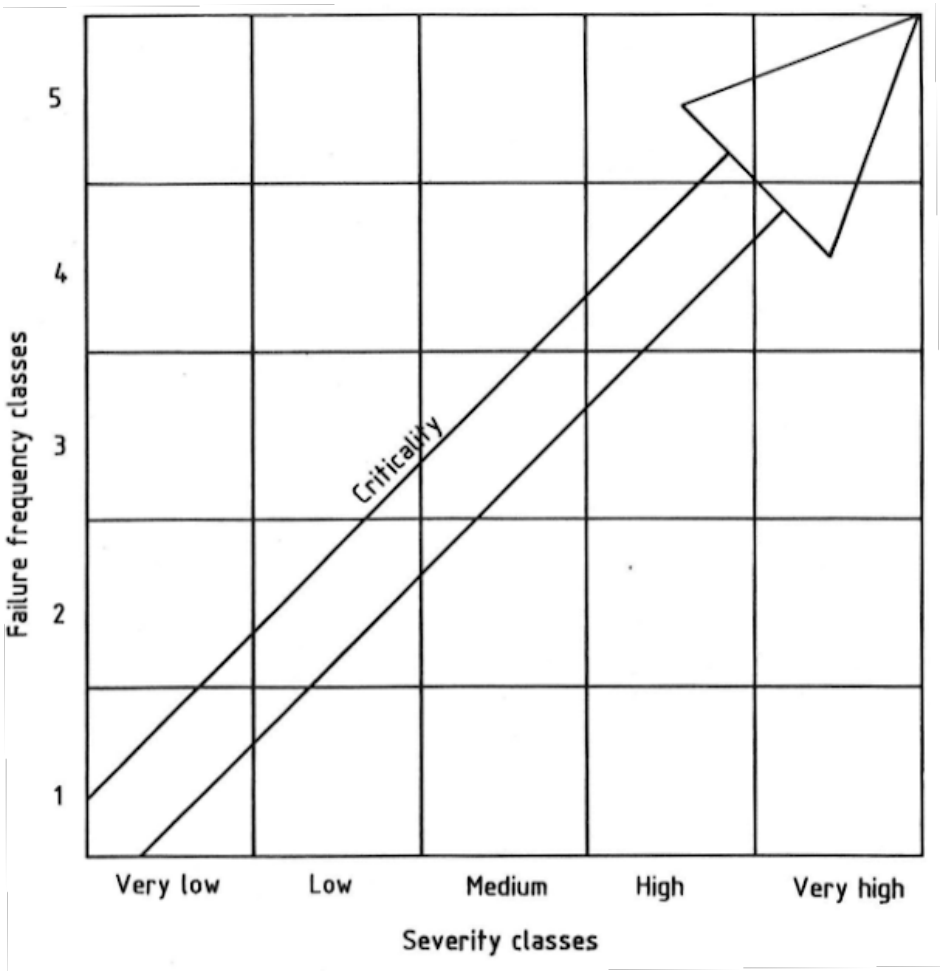


Figure 3.4: Criticality Grid

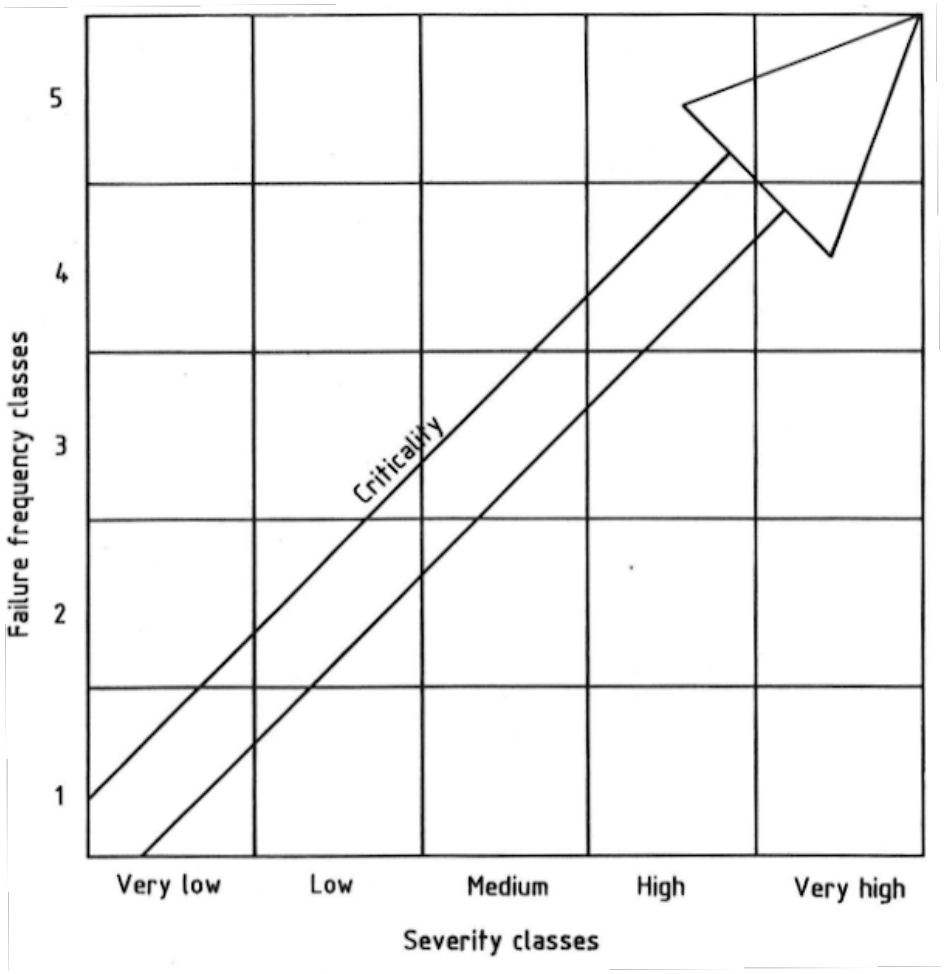


Figure 3.5: Criticality Grid

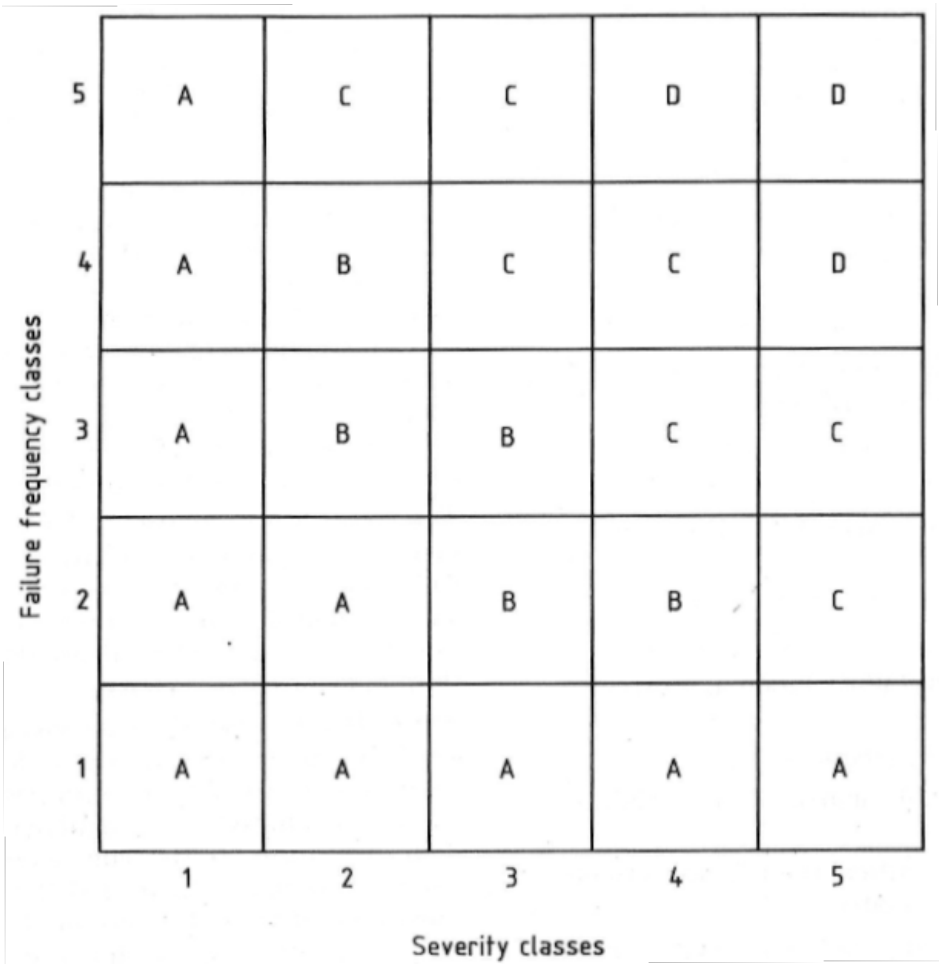


Figure 3.6: FMEA Criticality Matrix

1. The highest level within the system is selected from the design concept and specified output requirements.

2. The lowest level within the system at which the analysis is effective is that level for which information is available to establish definition and description of functions. The appropriate system level is influenced by previous experience. Less detailed analysis may be justified for a system based on a mature design, with good reliability, maintainability and safety record. Conversely, greater detail and a correspondingly lower system level is indicated for any newly designed system or a system with unknown reliability history.

3. The specified or intended maintenance and repair level may be a valuable guide in determining lower system levels.

3.5.2 Failure Modes

Successful operation of a given system is subject to the performance of certain critical system elements. The key to evaluation of system performance is the identification of critical elements. The procedures for identifying failure modes, their causes, and effects can be effectively enhanced by the preparation of a list of failure modes anticipated in the light of the following:

1. The usage of the system;

2. The particular system element involved;

3. The mode of operation;

4. The pertinent operational specifications;

5. The time constraints;

6. The environment.

In the FMEA, the definitions of failure modes, failure causes, and failure effects depend on the level of analysis. As the analysis progresses, the failure effects identified at the lower level may become failure modes at the higher level. Similarly, the failure modes at the lower level may become the failure causes at the higher level, and so on. A list of general failure modes is given in Table 1. Virtually every type of failure mode can be classified into one or more of these categories. However, these general failure mode categories are too broad in scope for definitive analysis; consequently, the list needs to be expanded to make the

categories more specific as shown in Table 2. Failure modes such as those listed in Table 2 can describe the failure of any system element in sufficiently specific terms. When used in conjunction with performance specifications governing the inputs and outputs on the reliability block diagram, all potential failure modes can be identified and described. It should be noted that a given failure mode may have several causes.

Table 3.1: Example set of general failure modes

Mode	Description
1	Failure during operation
2	Failure to operate at a prescribed time
3	Failure to cease operation at a prescribed time
4	Premature operation

It is important that evaluation of all items within the system boundaries at the lowest practicable level is undertaken to identify all potential failure modes. Investigation to determine possible failure causes and also failure effects on subsystem and system function can then be undertaken. Item or equipment suppliers should identify the potential item failure modes within their products. To assist this function, typical failure mode data can be sought from the following areas:

1. For new items, reference can be made to other items with similar function and structure and to the results of tests performed on them under appropriate stress levels.

2. For items in use, in-service records and failure data may be consulted.

3. Potential failure modes can be deduced from functional and physical parameters typical of the operation of the item.

It is important that item failure modes are not omitted for lack of data and that initial estimates are improved by test results and design progression. The FMEA should record the status of such estimates.

The identification of failure modes and, where necessary, the determination of remedial design actions, preventative quality assurance actions, or preventative maintenance actions is of prime importance. It is more important to identify and, if possible, design out modes than to know their rate of occurrence. When it is difficult to assign priorities, criticality analysis may be required.

Example of an expanded tist of failure modes			
1	Cracked/fractured	21	Binding/jamming
2	Distorted	22	Loose
3	Undersize	23	Incorrect Adjustment
4	Oversize	24	Seized
5	Fails to open	25	Worn
6	Fails to close	26	Sticking
7	Fail open	27	Overheated
8	Failed closed	28	False response
9	Internal leakage	29	Displaced
10	External Leakage	30	Delayed operation
11	Fails to stop	31	Burned
12	Fails to Start	32	Collapsed
13	Corroded	33	Overloaded
14	Contaminated	34	Omitted
15	Intermittent operation	35	Incorrect assembly
16	Open circuit	36	Scored
17	Short circuit	37	Noisy
18	Out of tolerance (drifted)	38	Arcing
19	Fails to operate	39	Unstable
20	Operates prematurely	40	Chafed
NOTE: This list in an example only. The modes contained in the list cannot be applied to all items and the list is not exhaustive			

3.5.3 Failure Causes

The possible causes associated with each possible failure mode should be identified and described. The causes of each failure mode are identified in order to estimate its probability of occurrence, to uncover secondary effects, and to devise recommended corrective action. Since a failure mode can have more than one cause, all potential independent causes for each failure mode need to be identified and described. The failure causes within the adjacent system levels should also be considered. The list given in Table B illustrates how a more specific definition of failure causes can be developed.

Table 3.2: Possible failure causes

Type	Examples
Specification	Omitted statements, Erroneous statements, Support system failure
Design	Misapplication, Design error, Design omission, Support equipment failure
Manufacture	Omitted action, Erroneous action, Procedural error, Manufacturing equipment
Installation	Omitted action, Erroneous action, Procedural error, Installation equipment fail
Operation	Omitted action, Erroneous action, Procedural error, Off-line equipment failure
Maintenance	Omitted action, Erroneous action, Procedural error, Maintenance equipment fa
Environment	Temperature, Humidity, Vibration, Corrosion, Uncontrollable forces, Fire, Floo

Note: This table is an example only. Different types and examples of failure causes may be required for different systems.

3.5.4 Common-Cause (Common Mode) Failures

In a reliability analysis, it is not sufficient to consider only random and independent failures. Some 'common-cause' (or 'common mode') failures (CCF) can occur that cause system performance degradation or failure through simultaneous deficiency in several system components, due to a single source such as design error or human error. A CCF is the result of an event that, because of logical dependencies, causes a coincidence of failure states in two or more components (excluding secondary failures caused by the effects of a primary failure).

CCFs can be analysed qualitatively using FMEA. As FMEA is a procedure to examine successively each failure mode and associated causes and also to identify all periodic tests, preventative maintenance measures, etc., it makes possible a study of all the causes which can induce potential CCF.

A check list developed from Table 3 may be used to identify in a detailed manner all possible causes which may induce CCF. A combination of several methods is useful in dealing with these failures: functional diversity, redundancies of different types, physical separation, tests, etc. Check lists, as above, may be used to examine the relevance and effectiveness of each method. The examination of preventative measures against CCF is usually considered to be outside the scope of FMEA, but this need not be the case.

3.5.5 Human Factors

Some systems have to be designed to cater for some human error, for example by providing mechanical interlocks on railway signals, and passwords for com-

puter usage or data retrieval. Where such provisions exist in a system, the effect of failure of the provisions will depend on the type of error. Some modes of human error should also be considered for an otherwise fault-free system, to check the effectiveness of the provisions. Although incomplete, even a partial listing of these modes is beneficial for the identification of design and procedural deficiencies; the identification of all possible forms of human error would probably be impossible.

Many CCFs involve human factors. For example, incorrect maintenance of similar items can negate redundancy. To avoid this, material diversity in redundant elements is often introduced.

3.5.6 Software Errors

Malfunctions due to software errors or inadequacies will have effects whose significance will be determined by both hardware and software design. The postulation of such errors or inadequacies and the analysis of their effects is possible only to a limited extent. The effects upon associated hardware of possible errors in software may be estimated and the provision of fall-back arrangements either in software or hardware is often suggested by such analysis.

3.5.7 Failure Detection Methods

The methods for detection of the failure mode should be described. Failure modes other than the one being considered which give rise to an identical manifestation should be analysed and listed. The need for separate detection of failure of redundant elements during operation should be considered.

3.5.8 Failure Effects

General

A failure effect is the consequence of a failure mode in terms of the operation, function or status of a system (see 1.2.1). A failure effect may be caused by one or more failure modes of one or more items.

The consequences of each failure mode on system element operation, function, or status need to be identified, evaluated and recorded. Maintenance, preventive or corrective action to be taken in each case should be identified.

MISSING TABLE PAG 17

MISSING TABLE PAG 18

3.6 1.9 Consequences of System Failure

A system FMEA can be conducted independently, without referencing a specific application, and later adapted for project use. This is applicable to small assemblies, which can be considered as generic components (e.g., electronic amplifiers, electric motors, mechanical valves). However, it is more common to develop a project-specific FMEA and consider the particular consequences of system failure. It might be necessary to categorise the effects of failure on the system based on the consequences, such as fail-safe, fail-danger, repairable failure, non-repairable failure, mission degraded, mission failed, and effects on individuals, groups, or society in general.

The need to connect an FMEA to the ultimate consequence of system failure depends on the project and the relationship between the FMEA and other forms of analysis, such as fault trees.

3.6.1 2.4.2 Information Required

2.4.2.1 General

Company management should be aware that the success of FMEA (and FMECA) relies on the unrestricted availability of all relevant information to analysts and the active cooperation of the designer. Information in categories listed from 2.4.2.2 to 2.4.2.11 needs to be obtained.

2.4.2.2 System Structure

Information on system structure should include the following items:

(a) Different system elements with their characteristics, performances, roles, and functions;

(b) Logical connections between elements;

(c) Redundancy level and nature of the redundancies;

(d) Position and importance of the system within the whole facility (if possible);

(e) Inputs and outputs of the system;

(f) Changes in system structure for varying operational modes.

Data related to functions, characteristics, and performances are required for all levels considered, up to the highest level.

2.4.2.3 System Initiation, Operation, Control, and Maintenance

The status of different operating conditions of the system should be specified, as well as the changes in the configuration or position of the system and its components during different operational phases. The minimum performances demanded of the system should be defined so that success and/or failure criteria can be clearly understood. Specific requirements, such as availability or safety, should be considered in terms of specified minimum levels of performance to be achieved and maximum levels of damage or harm to be accepted.

It is necessary to have accurate knowledge of:

(a) The duration of each task the system may be called upon to fulfil;

(b) The time interval between periodic tests;

(c) The time available for corrective action before serious consequences occur to the system;

(d) The entire facility, the environment, and/or the personnel, including interfaces and interactions with operators;

(e) Repair conditions, including corrective actions and the time, equipment, and/or personnel needed to achieve them;

(f) Operating procedures during system start-up, shut-down, and other operational transitions;

(g) Control during the operational phases;

(h) Preventative and/or corrective maintenance (see note);

(i) Procedures for routine testing, if employed.

Note: One of the uses of FMEA is to assist in the development of the maintenance strategy. However, if the latter has been pre-determined, information on maintenance facilities, equipment, and spares should be known for both preventative and corrective maintenance.

2.4.2.4 System Environment

The environmental conditions of the system should be specified, including ambient conditions and those created by other systems in the vicinity. The system should be delineated with respect to its relationships, dependencies, or interconnections with auxiliary or other systems and human interfaces.

At the design stage, these facts are usually not all known, and therefore approximations and assumptions will be needed. As the project progresses, the data will have to be augmented, and the FMEA modified to allow for new information or changed assumptions or approximations. Often, the FMEA will be helpful in defining the required conditions.

Note: The FMEA should be updated for each design review milestone (see BS 5760: Part 1).

2.4.2.5 Modelling

FMEA requires some modelling of the system, i.e., a logical representation of the relevant information on the system (reliability block diagram or fault tree, etc.; see 2.4.2.9 and 2.4.2.10). Some assumptions may be made about the nature of failure modes and the seriousness of their consequences. For example, in safety studies, conservative hypotheses may have to be formed concerning the impact of certain failures on the system unless or until better information becomes available.

3.6.2 2.4.2.6 Software

An FMEA conducted on the hardware of a complex system may have repercussions on the software in the system. Thus, decisions about effects, criticality and conditional probabilities resulting from the FMEA may be dependent upon the software elements and their nature, sequence and timing. When this is the case, the interrelationships between hardware and software need to be clearly identified because any subsequent alteration or improvement of the software may modify the FMEA and the assessments derived from it. Approval of software development and change may be conditional upon a revision of the FMEA and the related assessments, e.g. software logic may be altered to improve safety at the expense of operational reliability.

3.6.3 2.4.2.7 System boundary

The definition of the system boundary is more likely to be influenced by design, source of supply, or commercial criteria rather than the optimum requirements of the FMEA. However, where it is possible to define the boundaries to facilitate the system FMEA and its integration with other related studies in the reliability program, such action is preferable. This is especially so if the system is functionally complex with multiple interconnections between items within the boundary and multiple outputs crossing the boundary. In such cases, it could

be advantageous to define a study boundary from functional rather than hardware divisions to limit the number of input and output links to other systems. This would tend to reduce the number of system failure effects. Care should be taken to ensure that other systems or components outside the boundaries of the subject system are not forgotten, by explicitly stating that they are excluded from the particular study.

3.6.4 2.4.2.8 Definition of the system's functional structure

The analysis should be initiated by selecting the lowest level of interest (usually the part, circuit, or module level) at which sufficient information is available or at which it is judged that it needs to be obtained (by tests etc.) to ensure a reliable design. Thus new features of the design should be thoroughly investigated but old features under known stress levels can be incorporated into the analysis at a higher level. If the analysis starts at the lowest level, the various failure modes that can occur for each item at that level are tabulated. The corresponding failure effect for each, taken singly and in turn, is interpreted as a failure mode for consideration of the failure effect at the functional level immediately above. Successive iterations result in the identification of the failure effects, in relation to specific failure modes, at all necessary functional levels up to the system or highest level. The choice of breakdown level (which may vary for different areas of the system) requires a dependable and detailed knowledge of the failure modes of the elements. Apart from this requirement, it is neither possible nor desirable to set strict rules about the choice of the breakdown level. When quantitative results are required, the level chosen should be one at which it is possible to obtain adequate (and dependable) failure data on each failure mode or error mode, or to make reasonable identified assumptions of such failure rates. The analyst should investigate all aspects which might be important until satisfied they are not.

3.6.5 2.4.2.9 Representation of system structure

Symbolic representations of the system structure and operation, especially diagrams, can be used. Usually block diagrams are adopted, highlighting all the functions essential to the system. In the diagram, the blocks are linked together by lines which represent the inputs and outputs for each function. Usually, the nature of each function and each input needs to be precisely described. There may also be several diagrams to cover different phases of system operation. Generally, graphical presentations, including those closely related to analytical

methods, like fault trees or cause-consequence diagrams, contribute to a better understanding of a system, its structure and its operation.

3.6.6 Block Diagrams

Diagrams showing the functional elements of the system are necessary both for technical understanding of the functions and the subsequent analysis. The diagrams should display any series and redundant relationships among the elements and the functional inter-dependencies between them. This allows the functional failures to be tracked through the system. More than one diagram may be needed to display the alternative modes of system operation. Separate logic diagrams may be required for each operational mode. As a minimum, the block diagram should contain the following:

1. Breakdown of the system into major subsystems including functional relationships;
2. All appropriately labelled inputs and outputs and identification numbers by which each subsystem is consistently referenced;
3. All redundancies, alternative signal paths and other engineering features which provide 'fail-safe' measures.

3.6.7 Failure Significance and Compensating Provisions

The relative significance of the failure should be recorded on the worksheet. Also recorded on the worksheet should be the identification and evaluation of any design features at a given system level for other provisions to prevent or reduce the effect of the failure mode. Thus the worksheet should clearly show the true behaviour of the equipment in the presence of an internal malfunction. Other provisions against failure which need to be recorded on the worksheet include the following:

1. Redundant items that allow continued operation if one or more elements fail;
2. Alternative means of operation;
3. Monitoring or alarm devices;
4. Any other means of permitting effective operation or limiting damage.

During the design stage, the functional elements (hardware and software) of a piece of equipment may be rearranged or reconfigured to change its capability. Following this, the relevant failure modes should be re-examined before repeating the FMEA.

Chapter 4

Problem, Proposals and Contributions

The Failure Mode and Effects Analysis (FMEA) process is an essential tool for ensuring the safety and reliability of complex systems. It involves a systematic approach to identifying potential failure modes and their potential effects, as well as developing appropriate measures to prevent or mitigate these failures. However, despite its importance, the FMEA process is not without its challenges and limitations, especially in the context of certifying systems according to ISO26262.

One of the main challenges with the FMEA process is that it relies heavily on human expertise and experience. This means that the quality of the analysis can be highly dependent on the skills and knowledge of the individuals involved, which can lead to inconsistencies and errors. Additionally, the FMEA process can be time-consuming and resource-intensive, which can be problematic for organizations with limited budgets and schedules.

Another challenge with the FMEA process is that it can be difficult to capture and analyze all the potential failure modes and effects, especially in complex systems. This can lead to critical failure modes being missed or overlooked, which can have serious consequences for the safety and reliability of the system.

Moreover, the FMEA process is often conducted in isolation from other system development processes, which can lead to a lack of integration and coordination between different teams and departments. This can result in duplication of efforts, inconsistencies, and a failure to address all the potential failure modes and effects.

In addition to the challenges associated with the FMEA process, certifying a system according to ISO26262 also poses significant challenges. This standard

is designed to ensure the safety of electronic systems used in road vehicles, and it requires a rigorous and systematic approach to system development and testing.

One of the main challenges with certifying a system according to ISO26262 is that it requires a significant amount of documentation and evidence to be produced to demonstrate compliance with the standard. This can be time-consuming and resource-intensive, and it requires a high level of expertise and knowledge.

Furthermore, ISO26262 requires a high level of integration and coordination between different teams and departments involved in the system development process. This can be challenging in large organizations with complex development processes, and it can require significant investment in training and development.

In conclusion, the FMEA process and certifying a system according to ISO26262 both pose significant challenges and require a high level of expertise, knowledge, and resources. However, these challenges can be overcome through careful planning, coordination, and a commitment to quality and safety. By addressing these challenges, organizations can ensure that their systems are safe, reliable, and compliant with industry standards and regulations.

The automation of the Failure Modes and Effects Analysis (FMEA) process is crucial due to the increasing complexity of systems, the need for enhanced efficiency, and the demand for more reliable products in today's fast-paced, technologically-driven world. This becomes even more relevant with the introduction of the ISO 26262 standard, which establishes stringent guidelines for the functional safety of electrical and electronic systems in the automotive industry. As systems become more intricate, the possibility of failure modes and their subsequent effects multiply, making manual FMEA methods laborious, time-consuming, and prone to human error. By leveraging artificial intelligence and advanced algorithms, automating the FMEA process allows for the swift identification of potential failure modes and their effects, ensuring a thorough and accurate analysis that complies with the rigorous requirements set forth by the ISO 26262 standard.

Furthermore, the automated FMEA process can reduce the workload on engineers and other experts by allowing them to focus on higher-level tasks and decision-making, while the system handles the minutiae of the analysis. This translates to substantial time and cost savings for organizations, which is essential for meeting the ISO 26262's requirement of achieving functional safety at an acceptable cost. Additionally, as the world becomes increasingly reliant on technology, the importance of product reliability and safety cannot be overstated. Automated FMEA not only improves the overall quality of products but also

reduces the risk of catastrophic failures that can have severe consequences, both financially and in terms of human safety. This is particularly significant in the context of ISO 26262, as it aims to minimize the risk of systematic and random hardware failures in automotive systems.

In conclusion, automating the FMEA process is essential for the development of reliable, efficient, and safe products in an increasingly complex and competitive world. By adhering to the ISO 26262 standard, organizations can ensure that they are meeting the functional safety requirements for electrical and electronic systems in the automotive industry, and automated FMEA plays a crucial role in achieving this objective.

4.1 Manuscript Organization

The proposed Manuscript is divided in 3 main Parts, each of those divided in chapters, summarized as follow the First Part includes:

Background The need for a safety assessment finds its rationale in the ability of electronics to fail in both temporary or permanent way. This section analyses the most common reason of transient failure in electronics, the effects of harsh environments with a high level of radiation. The different radiations and radiation effects are analysed to understand the core problem.

ISO26262 In this chapter, an all-encompassing overview of the ISO 26262 standard will be provided, a critical regulation governing the functional safety of electrical and electronic systems within road vehicles. As the intricacy of automotive systems perpetually expands, the adoption of this global standard has emerged as indispensable for curbing hazards and guaranteeing vehicles' secure operation. The chapter intends to demystify the various components of ISO 26262, expounding its aims, framework, and approaches, while accentuating its paramount importance in the spheres of automotive engineering, manufacturing, and production. By imparting an exhaustive comprehension of this crucial standard, the chapter aspires to highlight the integral role of ISO 26262 in augmenting automotive security and facilitating the industry in attaining uniform functional safety execution across a diverse range of vehicular contexts.

Failure Mode and Effect Analysis In this chapter, an in-depth exploration of Failure Modes and Effects Analysis (FMEA) shall be conducted, elucidating the necessity of this robust and systematic methodology in identifying, prioritizing, and mitigating potential failure modes within products, processes, or

systems. Comprehending the FMEA process is of paramount importance, as it directly impacts the capacity to innovate it efficaciously. Through a meticulous examination of the FMEA procedure, the chapter shall elucidate each stage, thereby facilitating the harnessing of its potential for perpetual improvement and risk reduction. Moreover, the chapter shall underscore the significance of innovation within the FMEA process itself, enabling organizations to detect and address potential issues with unparalleled expediency and foresight, ultimately resulting in the development of more resilient and dependable products, processes, or systems.

The Second Part Includes

Basic Hardware Components This chapter discusses the challenges of ensuring safety and quality in digital systems, specifically in the context of System-on-a-Chip (SoC) and Intellectual Property (IP) development. The classical approach to safety assurance relies on complex and systematic methods, requiring a complete description of every block in the system, including the set of failure modes and consequences. However, this process is mostly man-driven, making it prone to errors and omissions. Model-Based Safety Assessment (MBSA) provides a more automated approach, but building failure models for each IP in a SoC is still a challenging task. This chapter proposes a new approach to address this challenge, using digital fault injection to extract non-functional behavior and build a failure model that can be used in a MBSA framework. The chapter also presents the methodology for this approach, its application to two examples, and its scalability for more complex systems. Overall, the proposed approach offers a more automated and reliable way to ensure safety and quality in digital systems.

Complex and microprocessor Based Components Software reliability has become a crucial factor to consider in software design due to the increase in the density of integration in VLSI systems and microprocessor performance. Hardware redundancy and software-implemented error detection and correction mechanisms can manage errors, but the propagation of hardware faults still plays a crucial role in software failures.

This chapter proposes a new methodology to simplify the reliability assessment in software design by applying the same techniques used for hardware design to software reliability assessment. The focus is on the characteristics of basic blocks in software products, and the proposed method aims to extract reliability metrics for each basic block that can be recomposed just knowing the sequence of block required to execute a precise operation.

Cern Use Case Last, this chapter presents the environment chosen for a first practical use case for the methodology developed. CERN and its laboratories provide a perfect example of what an harsh environment may mean and what it takes to design and develop electronics for such applications.

In particular, the ongoing trend of developing reprogrammable electronics for the detectors on LHC has triggered the need for a more profound reliability assessment and the exploration of new tecniques of evaluation of the systems under development.

At the time being the SoC "Picorvino" is being developed, based on the single pipeline stage RISC-V processor PicoRV. This has been the system of choice for the first use case of the new methodology explored.

Finally the Third Part includes

Conlusion and Prospectives In the conclusion chapter, we will provide a comprehensive analysis of the results obtained throughout the study and discuss any contradictions that may have emerged. This chapter aims to synthesize the key findings and insights, drawing connections between different aspects of the research. Furthermore, we will highlight the implications of the results, reflecting on their significance in the broader context of the field. By examining the outcomes and addressing potential contradictions, this chapter will offer a thorough understanding of the study's overall impact and contributions to the knowledge base.

Appendixes The appendixed report the details of the peripherals of PicoRVino SoC that have been designed at CERN.

4.2 Publications

Part II

Second Part - Scientific Contributions

Chapter 5

Basic Hardware Components

5.1 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 ageing effects, and *transient* [9] created by particle impacts such as thermal neutrons at ground level or solar wind in low and high earth orbits [106][78]. 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) [61] 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 [75].

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 [94, 76]) will lead to system execution failure. FMEDA analysis [70], targeting goals such as ISO26262 automotive safety norm [58] 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 behaviours. Those 'out of trajectory' behaviours 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 [100]) or silicon (irradiation [14]). 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 Failure Mode and Effects Analysis (FMEA) of a SoC or a single Intellectual Property (IP) is still mostly human based, prone to a series of errors and omissions inherent to the human reasoning, inference and encompassing process. The main hard point of applying MBSA methods to SoCs is building the failure models for each IP composing it. Because safety models are far from the specifications of digital IPs behavioral models, a first step towards the automa-

tion of the construction of such a failure model is the automated exploration of the faulty state-space and extraction of its safety relevant behavior.

Attempts have already been put in practice on rather simple systems, relying on a human based library of components describing possible malfunction [7] [102]. However, such models are far from the complete dysfunctional model specifications and only macroscopic failures are considered, thus not leaving space to unexplored combination values in registers that may lead to unexpected faulty states and behaviours. For example, a 0 to 4 counter, which needs a 3-bit state register, may go out of it functional range in case of a bit flip. The state "$1-0-1$", in Fig.5.1, is a non-explored and not explicitly declared state. It may occur due to a faulty combination of values in the flip-flops storing the state.

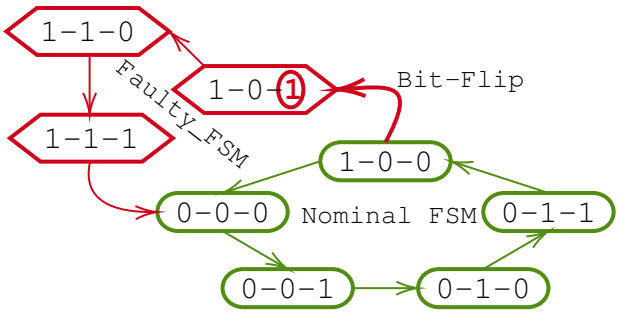


Figure 5.1: Non Explicit State Exploration Example

Exploring non-functional states in a digital system is far from human encompassing capabilities, even for systems with a small number of flip-flops. The approach proposed in this paper aims at filling the gap between digital IP behavioral model and generation of a dysfunctional model for FMEA evaluation, providing a model that can be used in a MBSA framework, thus, targeting safety assessment of the full system from its individual IPs. In this work we target tools based on the Altarica safety modeling language[80].

In the proposed approach, digital fault injection, in the form of bit-flip, stuck-at and transient faults, is used to extract the non-functional behavior of the studied digital block from its functional model. The extraction of those data allows building a failure model that includes the propagation of errors to and from inputs and outputs, thus enabling structural composition. We show that it is possible to extract a failure model in the form of a state machine describing the faulty behavior with scalable level of details and ensuring the fault propagation to (resp. from) outputs (resp. inputs).

The document is organised as follow: we first present the system used as example and how fault injection is used to expose dysfunctional behaviours and extract a model. The chosen approach is then detailed reminding generic princi-

ples 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.

5.2 State-of-the-Art

Several attempts towards fully automated FMEA are reported in literature. They can be divided into 3 categories based on the main idea that drove the approach to the problem: (1) Fault injection based, (2) manual-developed libraries approach, and (3) formal netlist verification methods. In the first category, the work in [70] aims to create primitives for a different standard [IEC61508], starting from a fault injection campaign and analyzing the results to evaluate the FMEA of a safety critical SoC, in order to evaluate the compliance to the standard. In the second category, the works in [7] and [102] have developed a framework for behavioral modeling a SoC (then being able to extract the FMEA from there) but starting from a library of elementary blocks, human written and prone to errors, which do not assure the complete FSM coverage for the blocks under test. In the last category, the works in [46][3][17][86] have tackled the problem from a different point of view, trying to formally verifying the netlist of a specific circuit and then build a translator from Verilog to CLU, the language utilized to verify control and mixed (data/control) paths. In [62], the behavior of system components are specified by UML (Unified Modelling Language) state machines determining intended/correct and undesired/faulty behaviors. The UML state machine description represents both nominal behavior of the component but also the failure modes though dedicated states (called *failure* states). The behavior of the component in each state is defined using the Object Constraint Language (OCL). The user then specify top nodes of the fault tree (state combinations at the system boundaries) and sequences of events composing the fault tree are computed and expanded. In [4] a reverse approach is followed were fault trees are converted and integrated into the *statechart* behavioural model of the system under evaluation. On top of the previously stated approaches, the most signficant for this work, it also worth to mention [96] and [85], where the problem of statistical forecasting of errors in microprocessors in ostile environment has been tackled even taking into account the whole stack of the execution. In this chapter the focus is on tools based on the Altarica language [80], which is a high level formal modeling language dedicated to safety analysis. It can be seen as a generalization of Petri nets for the behavioral part, and block diagrams for the structural part. It borrows

to Petri nets the notion of states, events and guarded transitions and to block diagrams, the notion of hierarchical descriptions and flows circulating through a network. Starting from such dysfunctional models, fault scenarios leading to a specified set of unexpected states can be computed and quantified to determine the probability of such behaviours. Automatic generation of reliability models such as fault tree, event tree, markov chain or monte-carlo simulation models for use in reliability assessment tools can be performed. Framework such as SimfiaNeo [68, 13], based on the Altarica language, belong to that category.

5.2.1 Probabilistic Methods in Digital Systems Safety

Probabilistic methods [92] [93] 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 [59]) required in ISO26262, but are far from being able to provide metrics even at the sequential block level. Likewise, industrial formal proof tools [30] [104] are able to compute such metrics by using formal methods.

Methods like FIDES [71] [103] 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 ageing failures but not transient effects such as soft-errors.

5.2.2 Formal Methods in Digital Systems Safety

Formal methods [18, 19] are mostly used on unitary blocks or functionalities to prove assertions (i.e. properties) expressed in linear [98] or branching [35] 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 [30][104] are able to compute, given a netlist 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.

5.2.3 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 [74]. MBSA introduces the use of high level modelling languages dedicated to functional safety analysis [81] [6] [16]. 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 modelling allowing identifying and locating induced effects of a failure inside the architecture.

Altarica Dataflow (Fig. **??**) 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)

- *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 behaviour 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 modelling, triggering events are used to represent failure modes. AltaRica allows to assign a probability law to an event, modelling the behaviour 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.

5.3 First Manual Application of FMEA on a SoC

In order to fully understand the challenges that performing the full FMEA process implies (as depicted in chapter MISSING REF), the decision of fully implement the process on a selected SoC has been taken. The SoC on choice has been the RISC-V based SoC developed in Microelectronics's, in order to have full observability and have all the testing suite and reports of irradiation available. Figure 5.2 shows the complexity of the entire system of choice $SCR1$, composed of two cores and several peripherals, each of those dedicated to a specific communication or testing function.

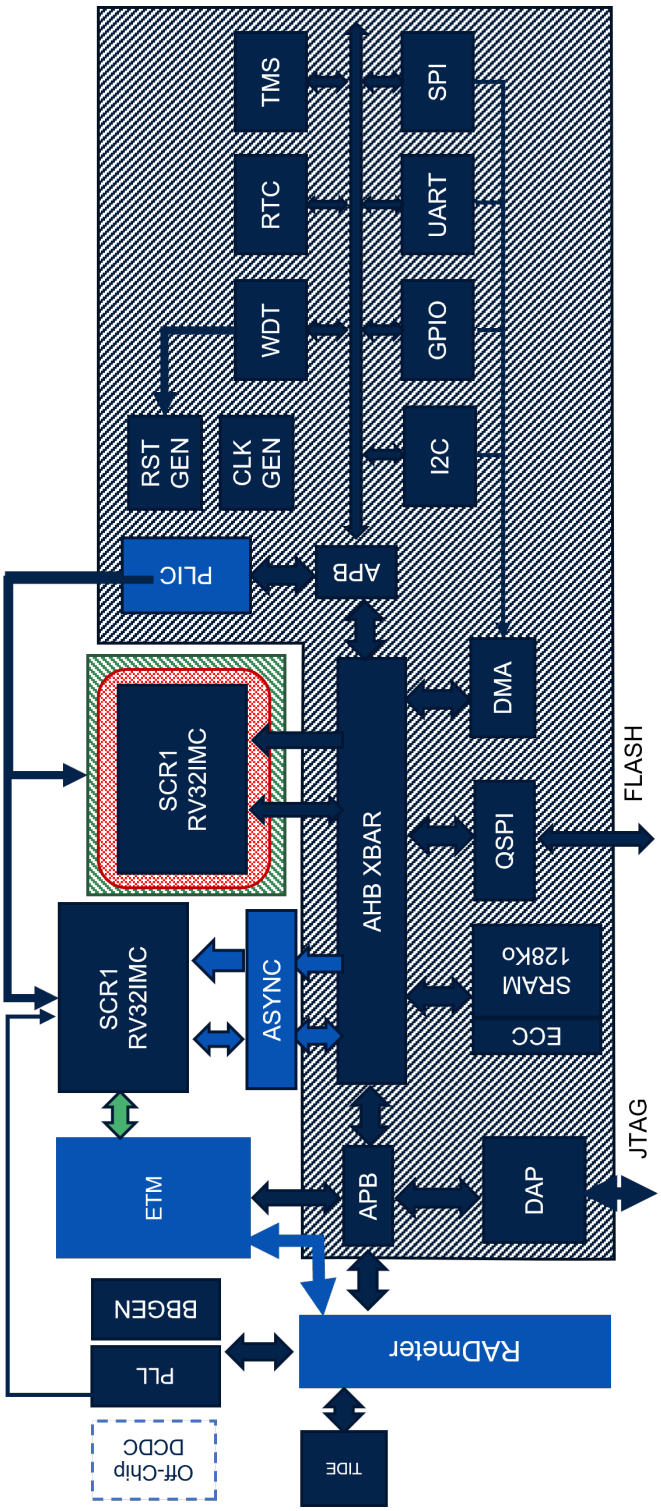


Figure 5.2: RTL Description of SCR1

It has been clear since the beginning that the greatest obstacle to overcome was the definition of the internal blocks of the cores, together with their inter-

connections. There is where the FMEA process has started. There was the need to identify one of the two twin cores and decompose it manually in order to obtain a complete definition to then analyse the possible failure modes. Figure5.3 shows the beginning of the process, dictated by the identification of all the connections to peripherals that are established with the core of choice.

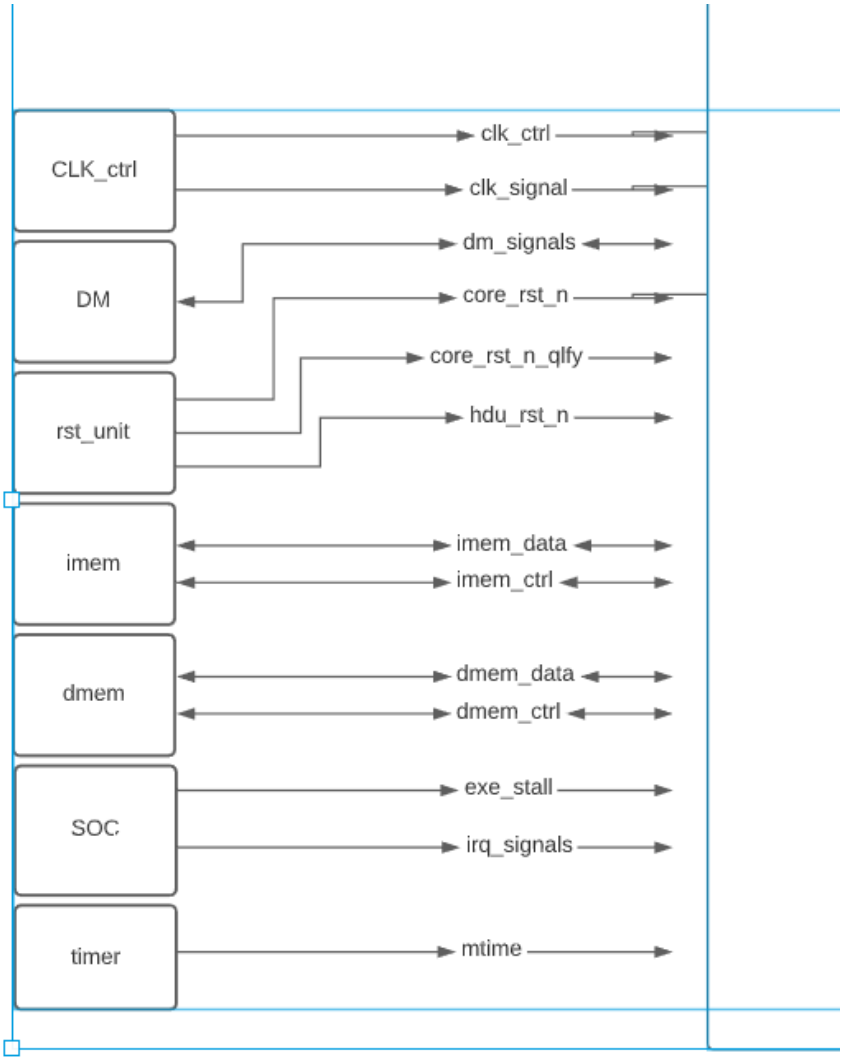


Figure 5.3: Inputs to the SoC

That being the starting point, the work has proceeded with the definition of all the internal signals to all the sub-blocks of which the core is composed. This is a rather time consuming operation to be carried out manually, and it will be shown, one of the most prone to introduce human errors. Nevertheless it has

been completed and the results are shown in Fig. 5.4, showing how complex the interconnections could be even in case of a high level description of the internal of the core.

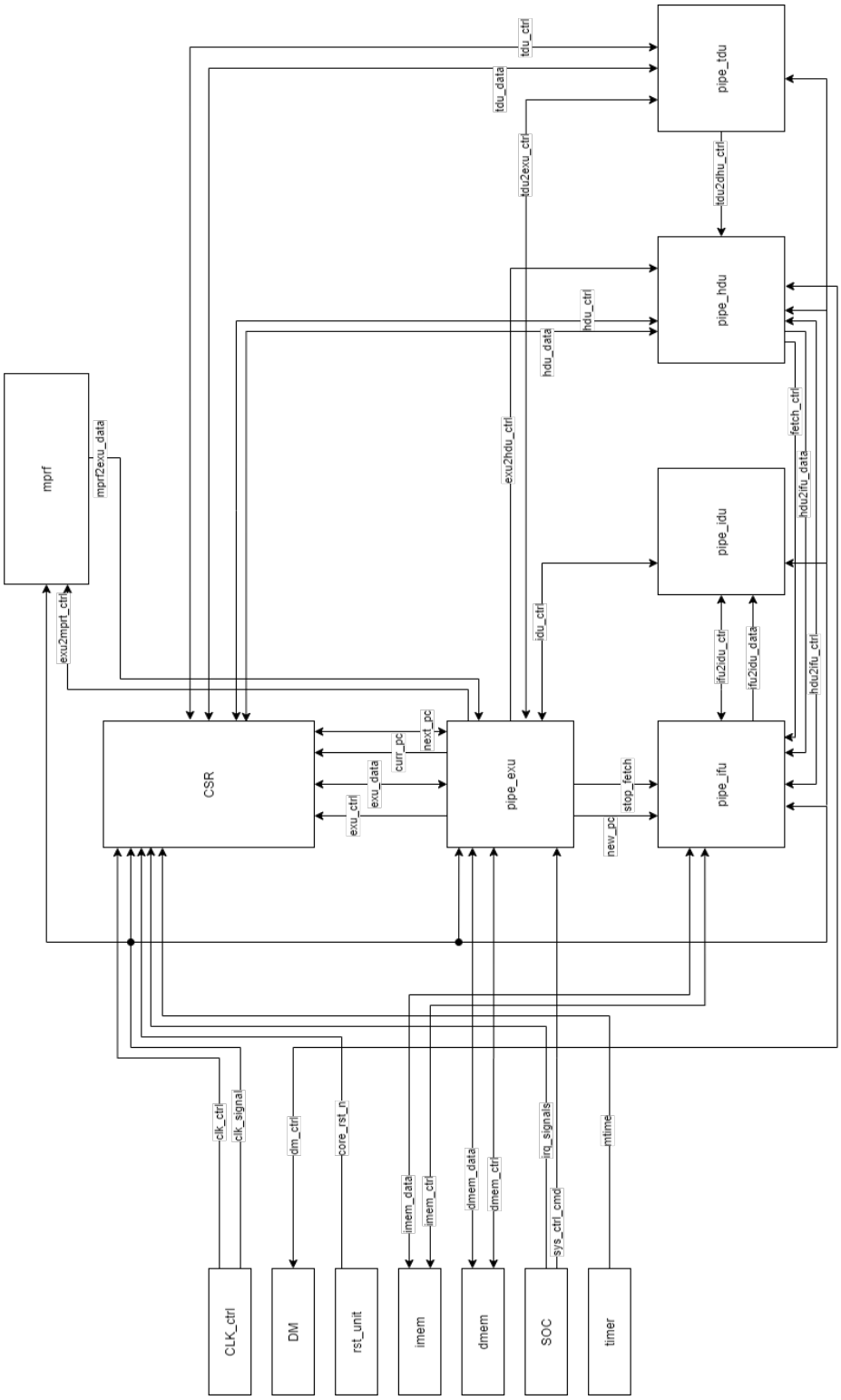
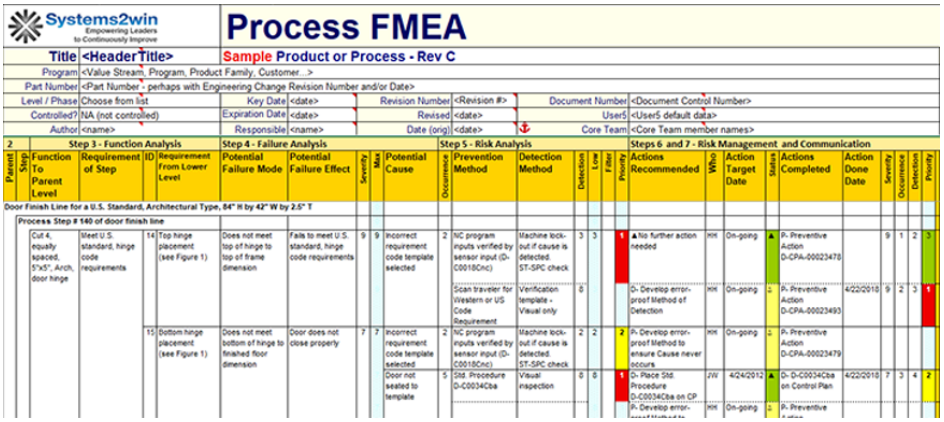


Figure 5.4: Detailed reconstruction of block model

Despite the completion of the internal signal definition, the complexity of the FMEA analysis procedure necessitated the use of the layout shown in Fig. 5.5 as a worksheet. This layout served as a guide for the FMEA analysis team, who were tasked with identifying potential failures within the system and assessing the severity of their impact. However, the complexity of the system, as evidenced by the intricate interconnections between sub-blocks, proved to be a significant obstacle for the team, and the procedure was ultimately halted due to its difficulty.



Systems2win
Empowering Leaders to Continuously Improve

Process FMEA

Title	<HeaderTitle>	Sample Product or Process - Rev C					
Program	<Value Stream, Program, Product Family, Customer...>						
Part Number	<Part Number - perhaps with Engineering Change Revision Number and/or Date>						
Level / Phase	Choose from list	Key Date	<date>	Revision Number	<Revision #>	Document Number	<Document Control Number>
Controlled?	NA (not controlled)	Expiration Date	<date>	Revised	<date>	User5	<User5 default data>
Author	<name>	Responsible	<name>	Date (orig)	<date>	Core Team	<Core Team member names>

2	Step 3 - Function Analysis				Step 4 - Failure Analysis					Step 5 - Risk Analysis					Steps 6 and 7 - Risk Management and Communication									
Parent Step	Function To Parent Level	Requirement of Step	ID	Requirement From Lower Level	Potential Failure Mode	Potential Failure Effect	Severity	Max	Potential Cause	Occurrence	Prevention Method	Detection Method	Detection	Low	Filter	Priority	Actions Recommended	Who	Action Target Date	Status	Actions Completed	Action Done Date	Severity	Occurrence	Detection	Priority
Door Finish Line for a U.S. Standard, Architectural Type, 84" H by 42" W by 2.5" T																									
Process Step # 140 of door finish line																									
	Cut 4, equally spaced, 5'x5', Arch, door hinge	Meet U.S. standard, hinge code requirements	14	Top hinge placement (see Figure 1)	Does not meet top of hinge to top of frame dimension	Fails to meet U.S. standard, hinge code requirements	9	9	Incorrect requirement code template selected	2	NC program inputs verified by sensor input (D-C0018Cnc)	Machine lock-out if cause is detected. ST-SPC check	3	3		1	▲No further action needed	HH	On-going	▲	P- Preventive Action D-CPA-00023478		9	1	2	3
											Scan traveler for Western or US Code Requirement	Verification template - Visual only	6				D- Develop error-proof Method of Detection	HH	On-going	▲	P- Preventive Action D-CPA-00023493	4/22/2010	9	2	3	1
			15	Bottom hinge placement (see Figure 1)	Does not meet bottom of hinge to finished floor dimension	Door does not close properly	7	7	Incorrect requirement code template selected	2	NC program inputs verified by sensor input (D-C0018Cnc)	Machine lock-out if cause is detected. ST-SPC check	2	2		2	P- Develop error-proof Method to ensure Cause never occurs	HH	On-going	▲	P- Preventive Action D-CPA-00023479					
									Door not sealed to template	5	Std. Procedure D-C0034Cba	Visual inspection	6	6		1	D- Place Std. Procedure D-C0034Cba on CP	JW	4/24/2012	▲	D- D-C0034Cba on Control Plan	4/22/2010	7	3	4	2
																	P- Develop error-	HH	On-going	▲	P- Preventive					

Figure 5.5: Classic FMEA Worksheet layout

Even with the aid of the layout, the team found that the manual identification of potential failures was too error-prone and time-consuming, necessitating a re-evaluation of the FMEA analysis process. A first detailed analysis of the possible failure modes and their gravity has been completed, and one extract can be seen in the Fig.5.6 Despite this setback, the layout shown in Fig. 5.4 remains a valuable tool for visualising the internal workings of the core and can aid in future system design and analysis efforts.

	PROBE	Definition	NOTES	Failure Mode	
	DATA Memory				
	tb_top.U_RV32SOC.U_SCR1.dmem_haddr	is the address bus, which size is defined by hsize	DATA	wrong data fetch	misaligned memory read
	tb_top.U_RV32SOC.U_SCR1.dmem_hburst	indicates the transaction burst type	3 bits?	wrong burst: sequential errors	
	tb_top.U_RV32SOC.U_SCR1.dmem_hmastlock	locked (atomic) transfer sequence		?	
	tb_top.U_RV32SOC.U_SCR1.dmem_hprot	provides protection information (I/D, user or handler)		?	
	tb_top.U_RV32SOC.U_SCR1.dmem_hsize	size of the transfer	Byte, halfword word	fetching a wrong sized data	trapping on unaligned address
	tb_top.U_RV32SOC.U_SCR1.dmem_htrans	current transfer type	idle, busy, nonseq, seq	Partial data fetching	
	tb_top.U_RV32SOC.U_SCR1.dmem_hwdata	data bus		Silent data corruption	
	tb_top.U_RV32SOC.U_SCR1.dmem_hwrite	transfer direction	\x1=R=0	unsaved data	hang waiting for data
	INSTRUCTION Memory				
	tb_top.U_RV32SOC.U_SCR1.imem_haddr	is the address bus, which size is defined by hsize	INSTRUCTION	Wrong instruction fetch	misaligned memory read
	tb_top.U_RV32SOC.U_SCR1.imem_hburst	indicates the transaction burst type		wrong burst: sequential errors	
	tb_top.U_RV32SOC.U_SCR1.imem_hmastlock	locked (atomic) transfer sequence		?	
	tb_top.U_RV32SOC.U_SCR1.imem_hprot	provides protection information (I/D, user or handler)		?	
	tb_top.U_RV32SOC.U_SCR1.imem_hsize	size of the transfer		fetching a wrong sized instruction	trapping on broken instruction
	tb_top.U_RV32SOC.U_SCR1.imem_htrans	current transfer type		partial instruction fetching	
	CSR				
	tb_top.U_RV32SOC.U_SCR1.U_core_top.Lpipe_top.Lpipe_csr.csr_i_data	microarchitecture register (Grep in the design)	^ code dependent ?^	?	
	tb_top.U_RV32SOC.U_SCR1.U_core_top.Lpipe_top.Lpipe_csr.csr_mstatus	mstatus register of the core		?	
	tb_top.U_RV32SOC.U_SCR1.U_core_top.Lpipe_top.Lpipe_csr.csr_mie	machine interrupt enable		?	
	tb_top.U_RV32SOC.U_SCR1.U_core_top.Lpipe_top.Lpipe_csr.csr_mtvec_base	machine trap vector table base address 64 byte alignment	MTVEC: Base (upper 26 bits)	?	
	tb_top.U_RV32SOC.U_SCR1.U_core_top.Lpipe_top.Lpipe_csr.csr_mtvec_mode	machine trap vector mode: exist the interrupt processing mode	MTVEC: Mode (0=direct, 1=vectored)	wrong interrupt mode (wrong routine	trapping on the wrong interrupt
	tb_top.U_RV32SOC.U_SCR1.U_core_top.Lpipe_top.Lpipe_csr.csr_mscratch	scratch register (GP)	The RISC-V CSR swap instructions can quickly save/restore values to the microarch register. Unlike the MIPS design, the OS can rely on holding a value in the mscratch register while the user context is running	Silent data corruption	
	tb_top.U_RV32SOC.U_SCR1.U_core_top.Lpipe_top.Lpipe_csr.csr_mcause_i	machine interrupt cause (identifies the interrupt nature)	Interrupts can be separated from other traps with a single branch on the sign of the mcause register value. A shift left can remove the interrupt bit and scale the exception codes to index into a trap vector table.		trapping on the wrong interrupt
	tb_top.U_RV32SOC.U_SCR1.U_core_top.Lpipe_top.Lpipe_csr.csr_mcause_ec	exception code			trapping on the wrong interrupt
	tb_top.U_RV32SOC.U_SCR1.U_core_top.Lpipe_top.Lpipe_csr.csr_mcause_ec_new	not user visible register -			trapping on the wrong interrupt

Figure 5.6: Classic FMEA Worksheet of SCR1

The difficulty encountered during the FMEA analysis procedure highlights the need to re-think the overall approach to system design and analysis. While the layout shown in Fig. 5.4 provides a useful tool for visualising system in-

terconnections, it also underscores the need for more automated and efficient methods of identifying potential failures. By relying on manual analysis, the FMEA procedure is susceptible to human error and can be prohibitively time-consuming for complex systems like the one depicted in the figure. Moving forward, it may be necessary to explore more sophisticated analysis methods, such as automated approaches, to ensure that potential failures are identified and addressed in a timely and accurate manner.

5.4 Modelling 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 modelling 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[63] and possibly formal methods [82].

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) modelling concepts into Altarica. When extracting a safety model from a digital block three points must be addressed:

- Structural hierarchy: Because Altarica support hierarchy [91], translating hierarchy with adequate granularity can be straightforward, especially as natural design hierarchy is usually a good candidate.

- Behavioural modelling: Faulty behavioural 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 characterise the behaviour 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.

5.5 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 behaviour when the block is exposed to those states though faults.

In order to build a failure model from a nominal behavioural *Register Transfer Model* (RTL) in Verilog or VHDL, behaviour of the system under faults must be analysed and faulty behaviour 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 footpath which maintain the control state of the block, correspond to possible fault injection sites as described in Fig.5.7.

2. *Testbench Setup* : A standalone testbench is set up with care given to coverage and testbench representability as the states traversed during this

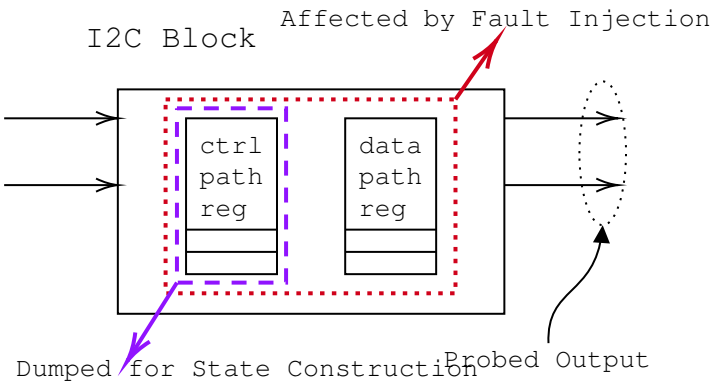


Figure 5.7: scheme of probes placement and affected/dumped registers

golden execution will serve as non-faulty reference behaviour. Tools like *Incisive Metrics Coverage* (IMC) [20] or *Certitude* [89] 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 misbehaviour 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 Behaviour*: Once the faulty runs have completed, non-functional (i.e. *faulty*) states and behaviour 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.

```
    # Opening LogFile
    with open (PATH, "r") as file_handler:
        tmp_state_path.clear()
        dect_flag = 0
        for line in file_handler:
            if LINE_IDENTIFIER  in line:
                raw_output = line.split(":")[1].strip() # Identify desired rows and isolate output
                tmp_state_path.append(raw_output)

          # Occurrencies
                if raw_output in occurrencies_dict:
                    occurrencies_dict[raw_output] += 1
                else:
                    occurrencies_dict[raw_output] = 1


                    if raw_output in occurrencies_dict_gold:
                        G.add_node(raw_output, color = '#03fc94')
                        node_color_map.append('green')
                        ## TRY
                        if dect_flag != 0:
                            LG.add_node(raw_output, color = 'g')
                    else:
                        G.add_node(raw_output, color = '#fc0303')
                        #print('RED STATE : ', raw_output)
                        node_color_map.append('red')
                        ## TRY
                        if dect_flag != 0:
                            LG.add_node(raw_output, color = "#ed33ff")
              # Transitions
                if counter > 0:
                    transition_key = previous_raw_output + ";" + raw_output
                    if transition_key in transitions_dict:
                        transitions_dict[transition_key] += 1
                    else:
                        transitions_dict[transition_key] = 1
                        if transition_key in transitions_dict_gold:
                            G.add_edge(previous_raw_output, raw_output, label = 'g')
                            if dect_flag != 0:
                                LG.add_edge(previous_raw_output, raw_output)
                        else:
                            G.add_edge(previous_raw_output, raw_output, label = 'ILLEGAL')
                            if dect_flag != 0:
                                LG.add_edge(previous_raw_output, raw_output, color = '#ed33ff', lable = "ILLEGAL")
                        #edge_color_map.append("black")
                previous_raw_output = raw_output
                counter += 1
```

5. *Construction of the Faulty Model*: The newly discovered states and transitions are used to augment the functional models with faulty behaviour. Transitions from a functional to a non-functional state are labelled 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.

5.5.1 Faulty Behaviour Model Construction

Once the faulty behaviour 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 subgroup) 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 [24] for our dysfunctional behaviour modelling (Fig. 5.8).

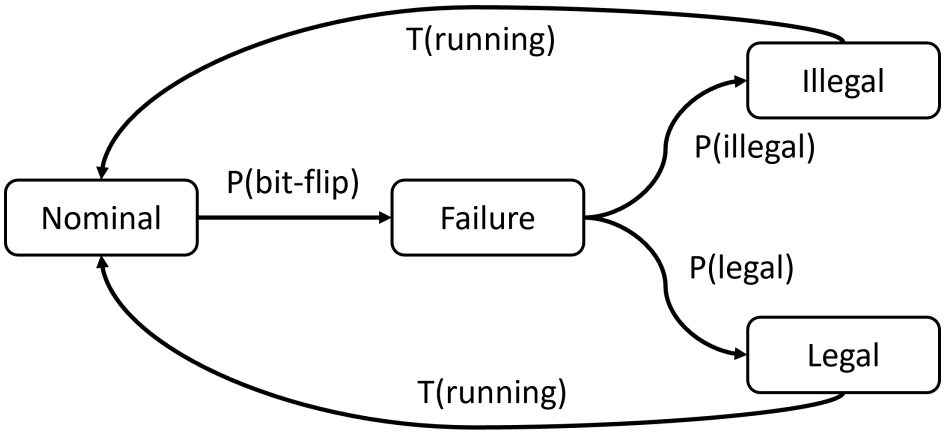


Figure 5.8: Altarica Dysfunctional Model

5.5.2 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 are, in fact, data state as control states. The latter can be easily identified as randomising data in the golden or faulty state machine extraction step leads to an increasing number of states with the number of runs. On Fig. 5.9, 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).

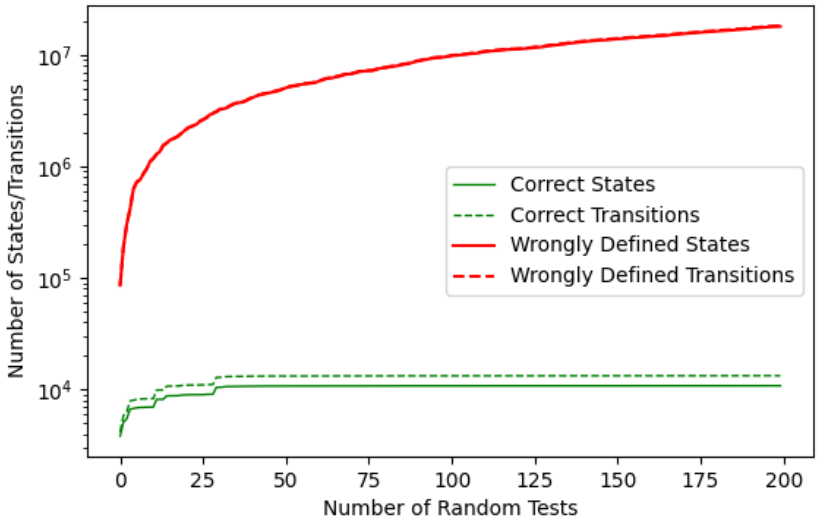


Figure 5.9: Completeness of the Extraction

The choice of the dummy example fell on a simple I2C to WB communicator, with the sole purpose of proving the mechanism of the methodology to me as smooth as advertised.

This prove has raised question on the importance of the proper definition of the state and whether or not the state space would have an upper bound much lower than the predicted 2^n states where n is the number of bits in the registers of the design. In figure number 5.10 is in fact possible to observe how a state defined by the totality of registers in the control path, contaminated by register of the data path, has a monotonically non-decreasing behaviour.

This can be fixed manually eliminating from the definition of the state all those signals carrying data, obtaining then the second graph in figure 5.11.

The same figure 5.11, show nonetheless the trend of the state and transition space in case of fault injection, in which one fault per run is injected in a random fault-able flop at a random time. Unfortunately, while it is possible to find an upper bound to the number of states and transitions in nominal conditions, the trend for the illegal ones results to be monotonically non-decreasing. The results are reported in figure 5.11 up to a thousand random tests, but a widely larger number of random tests have been carried out looking for the upper limit for that curve.

It can then be deduced that, while it is possible to affirm that with a reasonable ($\approx$25) number of random tests it is possible to explore the totality of the legal state a certain machine can find itself in, the coverage for the illegal state machine remains at "best effort".

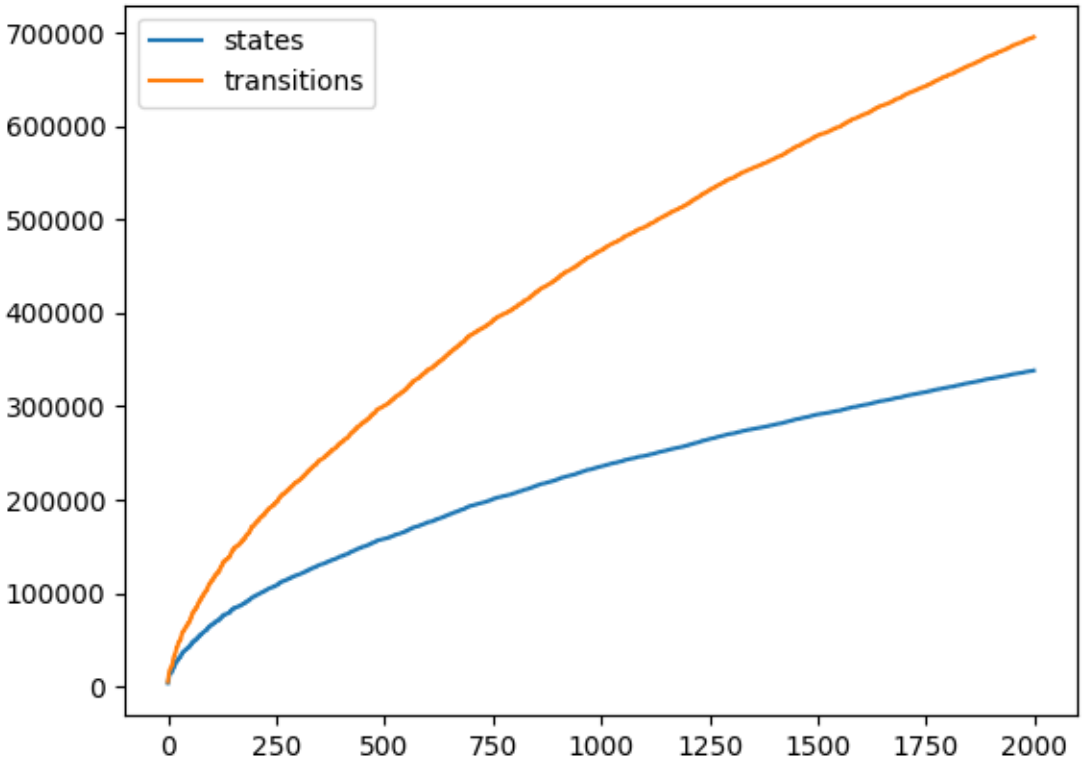


Figure 5.10: Data Contaminated Definition of the State

These consideration are not influencing the recomposition, since they only affirm the relevance of the study with respect to the ideal case.

5.5.3 Altarica Modelling

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.

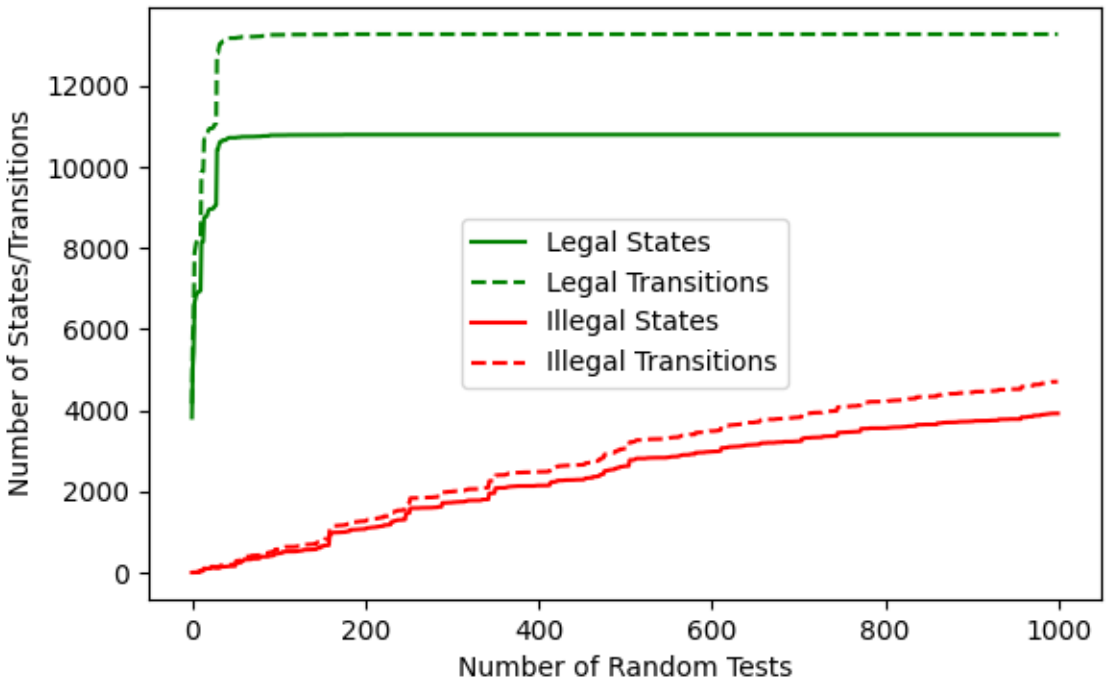


Figure 5.11: Pure Control Defined State

Base modelling must include at least four states (Fig. 5.18): 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 5.3). 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. 5.18 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.

5.6 Application: *I2C* to *AHB* bridge

In order to exercise the methodology presented in Section 6.3, we use a test case composed of 2 blocks: an *I2C* slave [69] connected to an *AHB* [40] bus master interface (Fig. 5.12). 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 [47], 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. 5.13. The system is represented on Fig. 5.12. 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.

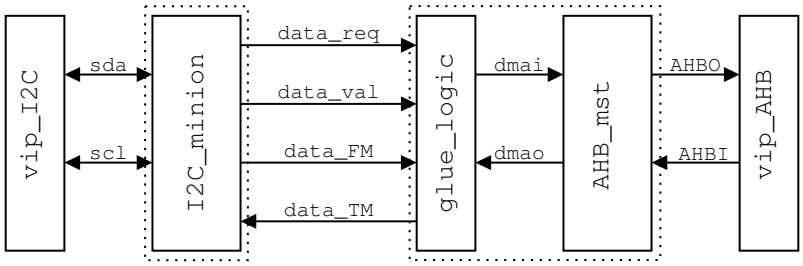


Figure 5.12: I2C to AHB System Block Diagram

Figure 5.13: I2C/AHB System Model

5.6.1 I2C Block Modelling

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 5.1. Having considered the results of the coverage evaluations sufficient for the demonstration, application of the method presented in section 6.3 have been performed. The list of all injection sites, reported by Cadence *Xcelium Fault Simulator* (FSV) [20] 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.

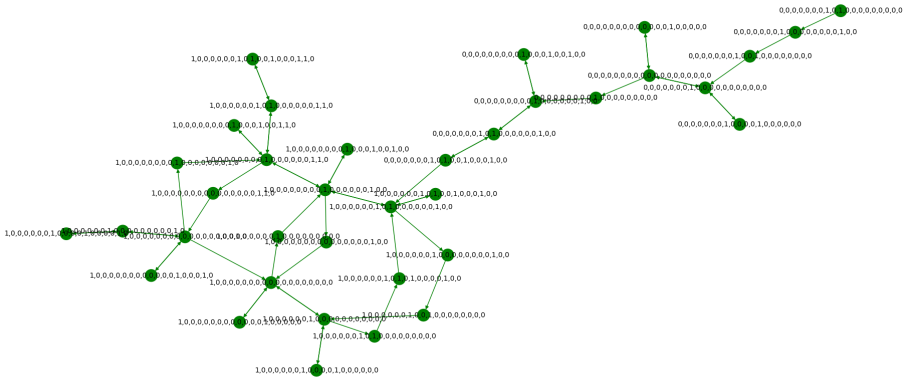


Figure 5.14: I2C Gold Automaton

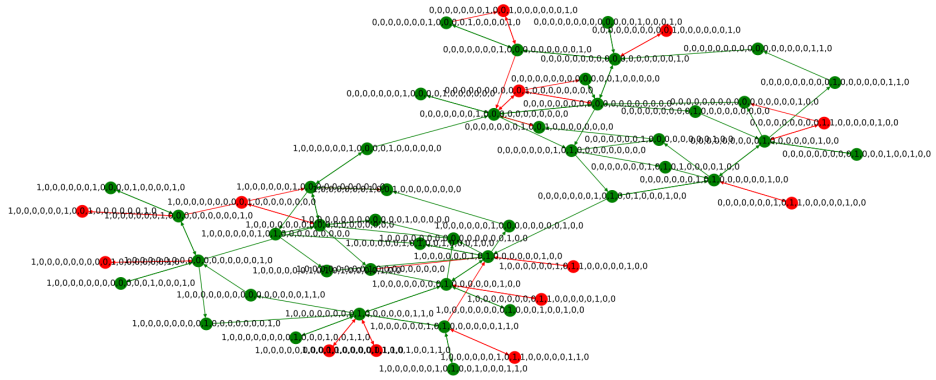


Figure 5.15: I2C Complete Automaton

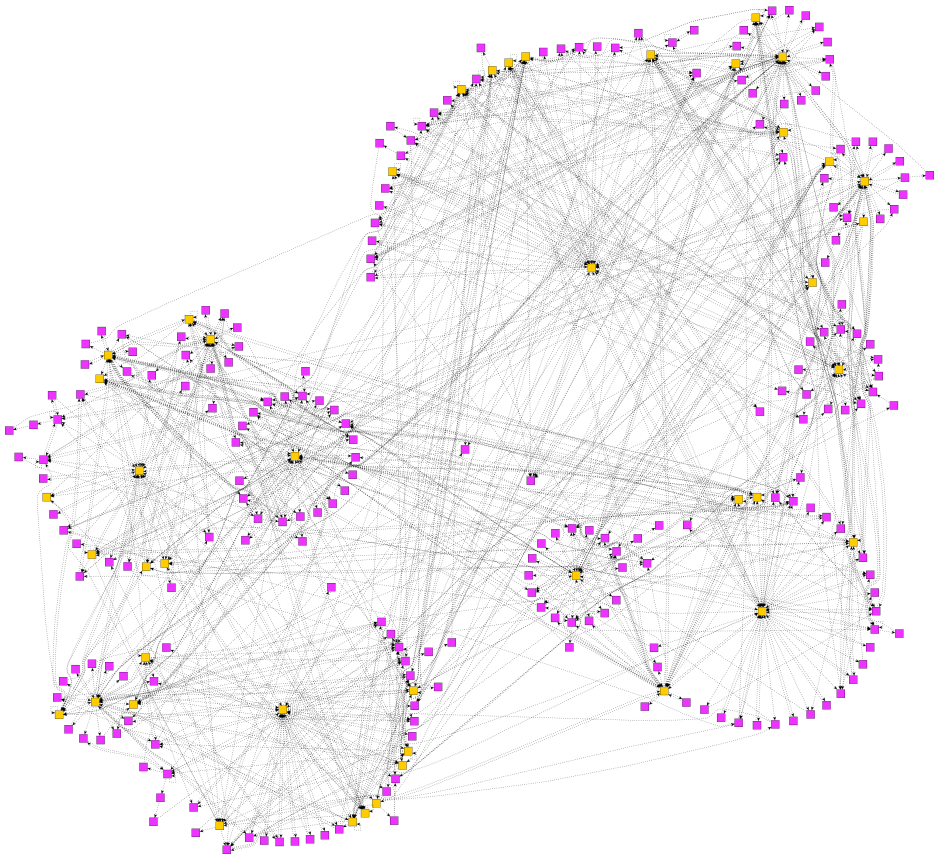


Figure 5.16: I2C Complete Automaton Clusterized

Table 5.1: 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%

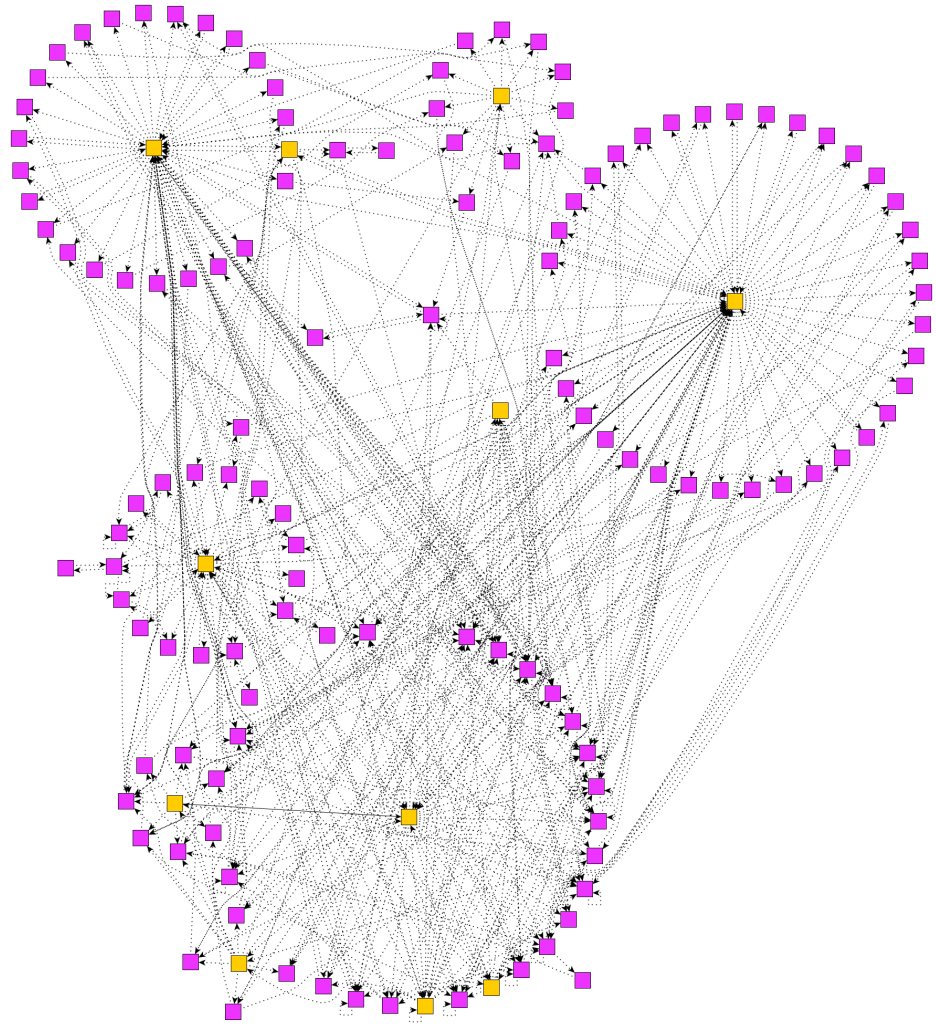


Figure 5.17: AHB Complete Automaton Clusterized

Fault injection traces are then processed following rules described in section 5.5.1 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. 5.18.

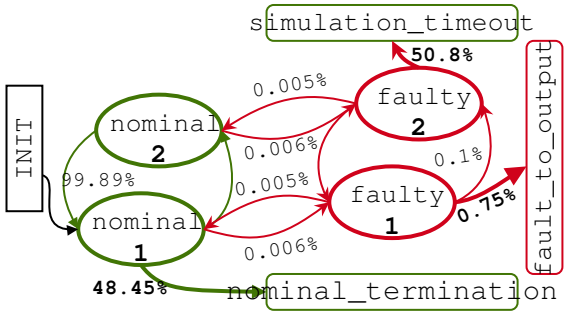


Figure 5.18: Extracted FSM for the I2C Block

5.6.2 AHB Block Modeling

The AHB bus interface is taken from the GRLIB [41] 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. 5.13 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.

5.6.3 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. 5.19, 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. 5.20 and Fig. 5.21.

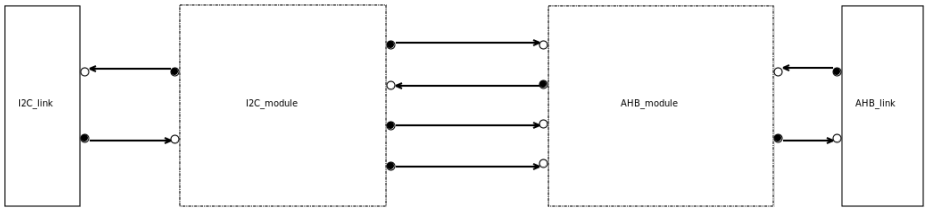


Figure 5.19: I2C to AHB System Model

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 *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

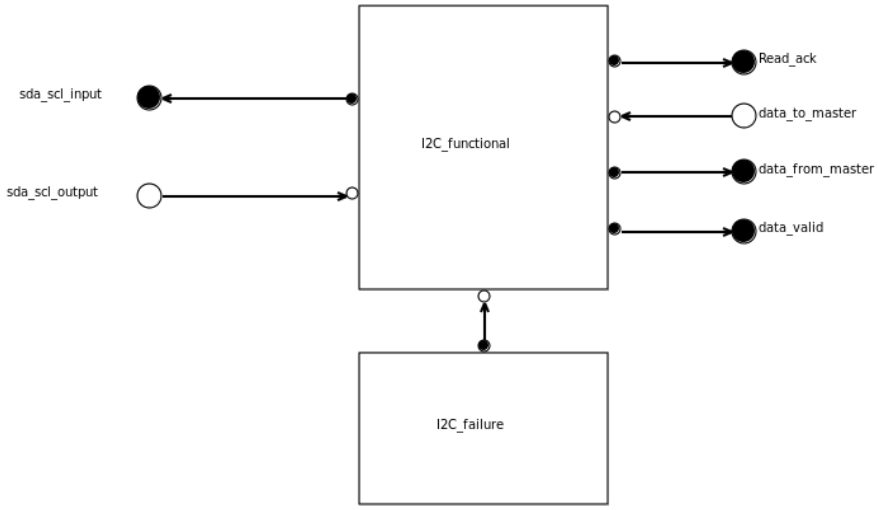


Figure 5.20: I2C Block Model

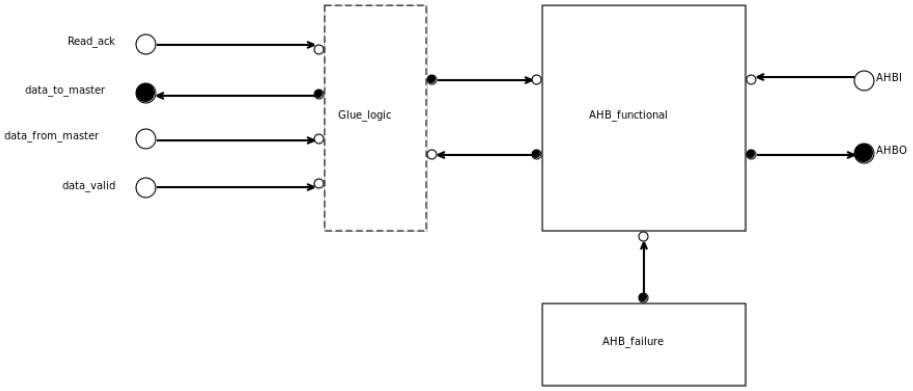


Figure 5.21: AHB Block Model

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 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.

5.7 Application

In this section, we apply the methodology presented in section 6.3 to the two blocks of our test case one after the other, constructing a failure model for each one.

5.7.1 Identification and Extraction of Faulty States

As stated in section 6.1, all digital systems can be represented as a finite state automaton, where the state is composed by all the flips-flops of the system, whether they maintain a control or data state. In this approach we consider, in a first approach, only control states with the following justification: faults (bit-flip) in datapath may not propagate in control states nor even create a faulty control state. Therefore faults in data states (that is flip-flops) will not lead to faulty behavior unless some data states are transformed directly into control states and encoding is sparse (some data states do not correspond to any control state and will therefore result in faulty state unless handled explicitly handled by a *default* case in the design).

On our example, analysis is performed at the RTL level, and the (signals composing the) state have been identified easily considering the small size of the

design. As the I2C protocol makes use of a serial line, data states and control states are merged when they can't be completely differentiated. On the AHB interface block, data states are excluded as no path exists from a data signal to a control signal (the reverse being obviously not true).

From a general perspective, analysis should be performed at the gate netlist level to ensure correct extraction of the control and data states with the drawback of slower fault simulation. Also data states can be pruned from the whole state using graph netlist forward (from data input) and backward (from data output) propagation algorithms. However, fault injection tools operating at the RTL level such as FSV [21] and ZOIX [90] do perform a pre-synthesis step to identify potential injection sites, that is flip-flops.

Once signals composing the state of a block have been identified, using a standalone testbench which can be derived from verification ones, a golden run is performed and golden states and transitions are recorded at each clock cycle. Such states and transitions are referred as *legal* composing the *non-faulty* or *golden* behaviour. Results for I2C and AHB blocks are reported in table **??**. The golden state automaton for the I2C is represented on figure **??**.

5.7.2 Fault Injection Campaign Setup

The next step in the methodology is fault injection. It is performed using Cadence fault injection tool *FSV* [21]. Once fault injection sites are automatically identified from the RTL description, fault injection is performed and 400 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* [42] has been developed for this purpose. Such number is statistically significant enough [67] without compromising fault injection campaign running time. Faults are also injected on inputs to take into effect of faults propagated from other blocks during composition. Also, fault probes are set on the outputs to record injected faults that will propagate to other blocks. For each fault injection run, states are recorded to identify new states and transitions that appear as a consequence of the injected faults. The new discovered states and transitions are referred as *illegal* or *faulty*.

In our modeling approach, we are interested only in states and transitions and we omit executions paths that are the ordered list of transitions traversed during a golden or fault run, even though it is available from the extracted data. The reason is that the faulty behaviour modelling strategy targeted, based on the Altarica language, doesn't requires such information. Thus only new faulty discovered states and transitions are extracted from execution runs and faults leading to an illegal execution path (list of traversed transitions) containing only

legal states and transitions will not be reported as a faulty behavior unless an incorrect output is reported during simulation.

In order to achieve this status during the nominal execution of the test-bench, the IP has been modified, adding dedicated code to write, into a log file, the state vector that includes all the selected control signals. This allows to have a real time, event driven, transition set in between the different combinations of the observed signals. The created log file will be crucial to the rest of the procedure, that will take it as input.

5.7.3 Faulty Behavior Extraction

Once the raw data have been dumped by the simulator in the log files (for a total of 12000 files equal to 400 faults $\times$ 30 flip-flops for the I2C and 42000 files equal to 400 faults $\times$ 105 flip-flops for the AHB), it is necessary to extract the values of the signals composing the state, records them and keep trace of all transitions. In order to complete this task, a python script using the *NetworkX* library [44] has been written to create a dictionary containing all the different states and their occurrence count, as well as another dictionary with all the transitions, to perform statistics and extract the faulty behavior model.

An example of raw log is given on listing **??**. States and transitions extraction from the logs is straightforward and requires only one pass per log file. An example of extracted fault behavior is represented on figure **??** for one flip-flop. Extracted automaton characteristics are reported on table **??**.

```
\caption{Raw log from I2C FSM extraction}

\label{lst:rawlog}
\begin{minted}{bash}
 --- Testing repeated reads ---
out:0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0
out:0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0
\end{minted}
```

Once the golden and faulty states and transitions have been extracted from the logs, the failure model can be built by collapsing states and transitions into the desired ones for the model. The raw faulty automaton (before node collapsing) for the I2C block is partially represented on figure **??** where legal states and transitions are represented in green and illegal ones in red.

5.8 Results on the $I2C$ to AHB System

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 5.2 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.

Table 5.2: RTL Fault Injection Vs Altarica Model Failures

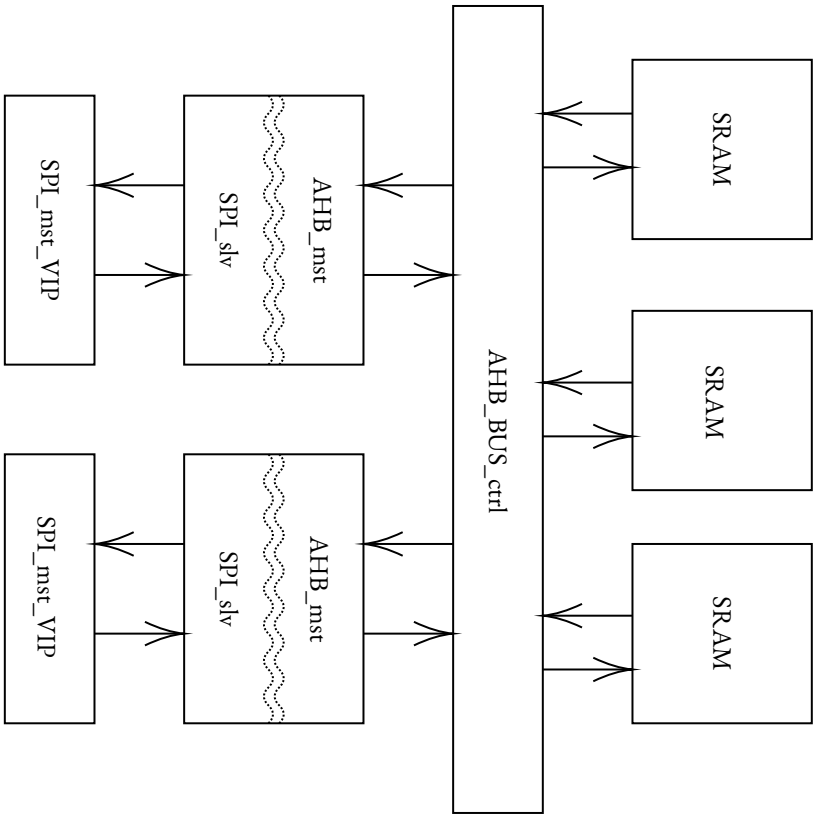
AHB Failure	I2C+AHB RTL	Altarica Model	I2C Failure	I2C+AHB RTL	Altarica Model
haddr	0.00551	0.00551	SDA	0.00338	0.00339
hwdata	0.00382	0.00382	Read req	0.00254	0.00255
hsize	0.00068	0.00068	Data	0.00580	0.00581
hbusreq	0.00026	0.00026	Data valid	0.00322	0.00323
hwrite	0.00022	0.00022			

Table 5.3: Block and Full System Fault Injection Results

		I2C	AHB	I2C + AHB	
Golden states (static)		44	11	44	11
Golden transitions (static)		90	20	89	19
Faulty states (static)		223	126	198	193
Faulty transitions (static)		1093	470	899	428
Injected faults (400 / FF)		11670	41200	52878	
Detected faults	golden	6840 (58.61%)	19285 (46.80%)	23840 (45.08%)	22089 (41.77%)
	faulty	19 (0.16%)	466 (1.31%)	219 (0.41%)	1970 (3.72%)
Detecting states	golden	44 (100%)	11 (100%)	44 (100%)	11 (100%)
	faulty	5 (1.87%)	13 (9.48%)	5 (2.06%)	70 (34.31%)

5.9 Second Proof of Concept 4BlocksSystem

In an effort to further augment the complexity of the proof of concept, an advanced and more intricate system has been meticulously designed and developed, as depicted in the subsequent figure. The decision to employ this particular system was made after careful consideration, ultimately opting for a system comprised of communicator Intellectual Properties (IPs) created by Gaissler. This choice ensures that the components utilized in the system have undergone a thorough verification and testing process, thereby establishing a high level of reliability and performance.



A closer examination of the system reveals that not only has the number of components constituting the system experienced a considerable increase, but the logical connections between the individual IPs have also become substantially more complex and sophisticated. This complexity is further exemplified by the presence of logical loops within the system, which add an additional layer of intricacy to the overall design. As a result, the task of recomposing the metrics becomes dramatically more challenging, as the simplistic case of the multiplication tree no longer proves to be a viable solution in this context.

The methodology adhered to in this advanced system remains consistent with the one meticulously delineated in the previous section. This methodology emphasizes a systematic and comprehensive approach to the design, development, and evaluation of the system, ensuring that each component and connection is thoroughly assessed and optimized for maximum efficiency and reliability. By maintaining this rigorous methodology, the proof of concept effectively demonstrates the scalability and adaptability of the approach, even when faced with systems characterized by higher levels of complexity and interconnectivity.

Additionally, the enhanced complexity of this system offers valuable insights into the challenges and potential obstacles that may arise when implementing the methodology in real-world scenarios, where systems often exhibit a wide

range of intricacies and nuances. Through the successful application of the methodology to this more complex proof of concept, the viability and efficacy of the approach are further reinforced, highlighting its potential for widespread adoption in the design and development of advanced digital systems.

5.10 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[38].

Chapter 6

Complex and μ-Processor Based Systems

6.1 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 [25] [45]. 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 [79] the propagation of hardware faults plays a central role, eventually catastrophic. Base on what literature reports on hardware faults evaluation reports [33] [34] it is possible to observe that the percentage of software failure that are caused by pure hardware faults average around 10% [64]. The most famous example is surely the crash of the Mars Polar Lander [11], 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 [37] 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.

6.2 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 [64][63][95] 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 [97] 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 [25]. 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 [77][99][73]. 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.

6.3 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** [21] or **Synopsys** [90] 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 [78] 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:6.3 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:

$$T_{campaign} = I \cdot \left[T_{init} + T_{foot} + \sum_{0}^{N} m_n \cdot T_{bb_n} \right] \tag{6.1}$$

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.

$$I \cdot \left[T_{init} + T_{foot} + \sum_{0}^{N} T_{bb_n} \right] \tag{6.2}$$

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:

$$
\begin{aligned}
I \cdot \left[\sum_{0}^{N} m_n \cdot T_{bb_n} \right] - I \cdot \sum_{0}^{N} T_{bb_n} = \\
= I \cdot \left[\sum_{0}^{N} m_n T_{bb_n} - \sum_{0}^{N} T_{bb_n} \right] = \\
= I \cdot \sum_{0}^{N} T_{bb_n} \cdot (m_n - 1)
\end{aligned}
\tag{6.3}
$$

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.

6.3.1 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

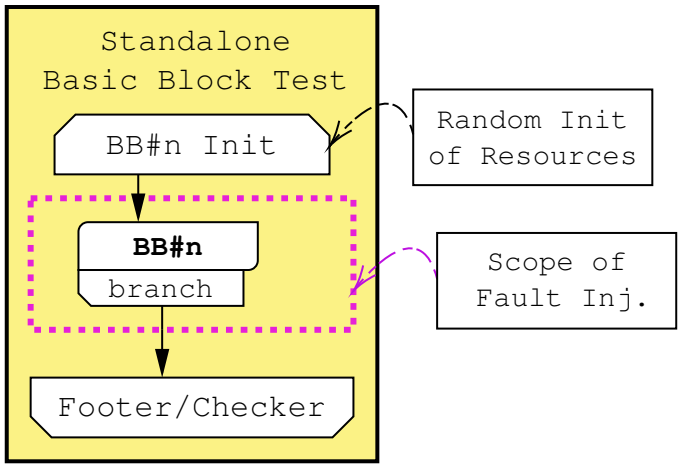


Figure 6.1: Block Diagram of the Entire SW Product

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.

6.3.2 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.

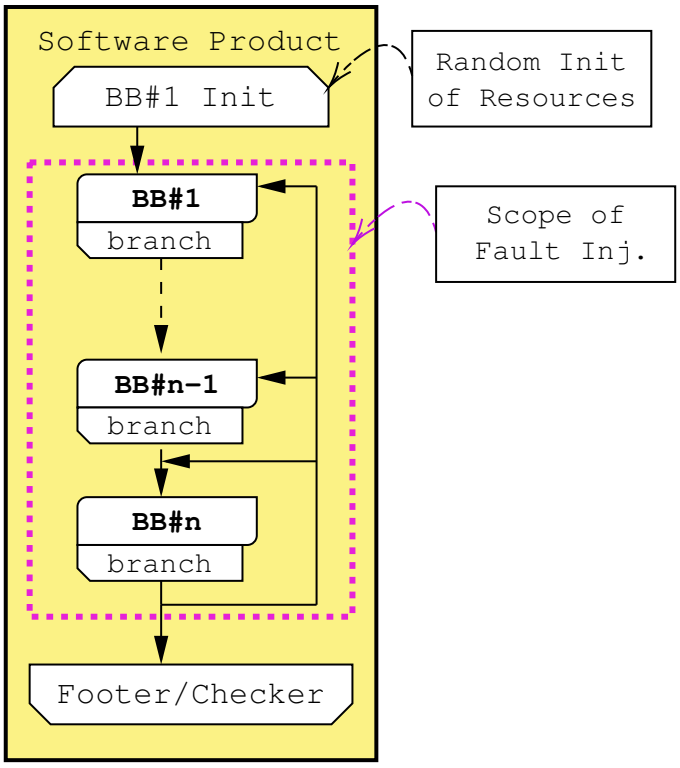


Figure 6.2: Block Diagram of the Entire SW Product

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.

6.3.3 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

$$P_{in-bb_n} = \frac{instructions - in - BB_n}{total - instruction - in - exe} \tag{6.4}$$

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:

$$P^{A_m}_{gBB_n} \tag{6.5}$$

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

$$P^{A_m}_{pBB_n} \tag{6.6}$$

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.

$$\begin{aligned} &P_{in-bb_n} * P^{A_m}_{gBB_n} * \left[P^{A_m}_{pBB_{n+1}} \dots P^{A_m}_{pBB_f}\right] + \\ &+ P_{in-bb_{n+1}} * P^{A_m}_{gBB_{n+1}} * \left[P^{A_m}_{pBB_{n+2}} \dots P^{A_m}_{pBB_f}\right] \dots \end{aligned} \tag{6.7}$$

which summarizes, per every register A_m as:

$$\sum_{n=0}^{N} P_{in-bb_n} * P^{A_m}_{gBB_n} * \left[\prod_{i=n}^{N} \left[P^{A_m}_{pBB_{i+1}}\right]\right] \tag{6.8}$$

$$P_{tot} = [P_{in-bb-x} * P_{p-bbx}] + [P_{p-bbx+1} * P_{p-bbx+1}] \dots \tag{6.9}$$

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 nominal 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:

$$P_{err} + P_{msk} * P_{err} + (P_{msk})^2 * P_{err} \cdots + (P_{msk})^m * P_{err} \tag{6.10}$$

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

6.4 Test Case and Application

6.4.1 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 [72].

Table 6.1: 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 6.2: 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 6.3: 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

6.4.2 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.

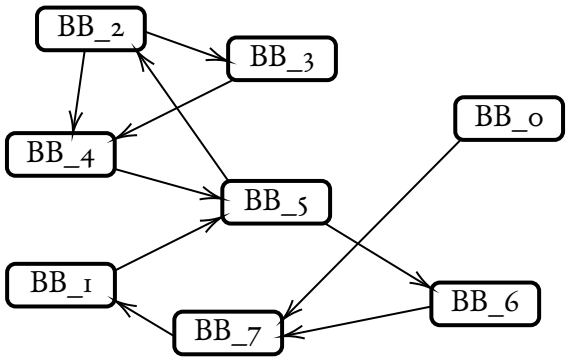


Figure 6.3: Block Diagram of the Entire SW Product

6.4.3 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 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.

6.4.4 The Fault Injection Campaign

The next step in the methodology is fault injection. It is performed using Cadence fault injection tool *FSV* [21]. 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* [42] has been developed for this purpose. Such number is statistically significant enough [67] 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 log-file 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.

6.5 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 7

CERN Use Case

In this chapter, a description of one of the LHC detectors at CERN is given, as to give a better understanding of the environment and setting in which the SoC is planned to function. Following this, a theoretical outline is given for radiation and its effects on CMOS electronics and how this can be mitigated by design choice. Following this, an method for extensive verification of individual block and system verification is presented. After this a description of the steps of physical implementation is given and finally the choice of CPU for the SoC is made, which is accompanied by a description of the state of the system at the beginning of the author's internship and the future plans for it.

7.1 LHC Detectors at CERN

There is a vast complex network of different accelerators and different detectors at CERN. The highest energy endpoint in this network is the LHC. The LHC is a $27\,\mathrm{km}$ counter-rotating accelerator. Using superconducting magnets, it is capable of accelerating protons up to a peak energy level of $7\,\mathrm{TeV}$, which results in a peak collision energy of $14\,\mathrm{TeV}$. To reach these energy levels, a network of several accelerators is used to initially accelerate the protons to $450\,\mathrm{GeV}$ before they are injected into the LHC. In the LHC the beams collide with a bunch spacing of $25\,\mathrm{ns}$ corresponding to a frequency of $40\,\mathrm{MHz}$ and is called the bunch-crossing (BX) rate. The entire network can be seen in figure 7.1. More details can be read in the article: LHC Machine [36].

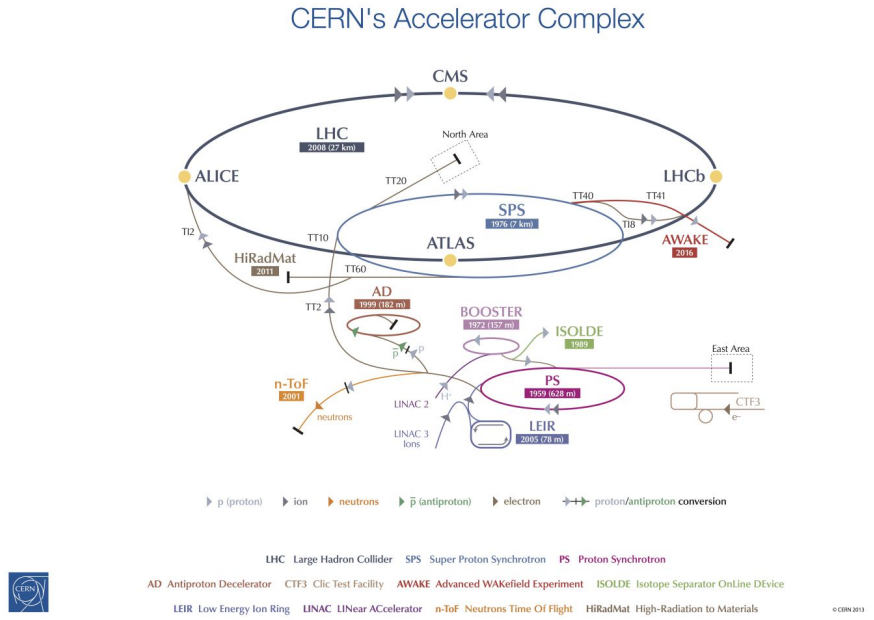


Figure 7.1: Shows the complete network of used accelerators at CERN (Courtesy of [43]).

There are four detectors placed on the LHC ring. These are A Large Ion Collider Experiment (ALICE), Compact Muon Solenoid (CMS), Large Hadron Collider Beauty (LCHb) and A Toroidal LHC ApparatuS (ATLAS). These 4 experiments can be split into 3 categories. CMS and ATLAS are general-purpose onion detectors, which means they try to detect all particles created by the collision. These two are built in different ways by different independent teams such that they can be used to verify the results of each other. ALICE is an experiment focused on the collision of heavier ions, e.g. lead ions. LCHb is a detector focused on only detecting the particles created by collision, which moves in the beam direction. This makes it possible for them to make a specialized detector better suited for detecting the particles in that specific direction. A majority of the particles created in this experiment are related to the beauty quark, which gives reason to its name. A detailed description of CMS, one of the general-purpose experiments, will now be given. This will lay the foundation for understanding the environment in which the electonics are expected to survie and give a reasoning to some of the design choices made later.

The CMS sits at one of the four collision points in LHC. Figure 7.2 shows the layout of the CMS detector. The particles generated in the collisions propagate radially, traversing the silicon tracker. The silicon tracker measures the particle trajectory and transverse momentum p_T. The silicon tracker is com-

posed of an all-silicon pixel and strip tracker [27]. Next are the electromagnetic calorimeter (ECAL) and the hadron calorimeter (HCAL). The calorimeters enable the evaluation of the particle energy. The ECAL uses lead tungstate scintillating crystals for this purpose [27]. Scintillating crystals emit photons when ionizing particles pass through them. The light is then detected by silicon avalanche photodiodes (APD) in the barrel region and vacuum phototriodes (VPT) in the endcap region. The APD makes use of the avalanche effect, where a single charged particle can knock multiple electrons out of their bond and thereby amplifying their electrical signature. The VPTs are single amplification stage photomultipliers. They have a photocathode at ground potential, a single dynode biased at $600\,\mathrm{V}$, and an anode biased at $800\,\mathrm{V}$. VPTs operate by a photon hitting the cathode which releases electrons. The released photoelectrons are accelerated towards the dynode, where each photoelectron releases multiple new photoelectrons. These are then accelerated towards the anode as it is at a higher potential. The anode then produces an amplified current.

After the ECAL, the particles enter a brass/scintillator HCAL [27]. Here the scintillation light is collected by wavelength-shifting (WLS) fibers embedded in the scintillator tiles. The WLS fibers emit multiple low-energy photons for each high-energy photon strike. This light is channeled to photodiodes which amplify the signal. The aforementioned components are encapsulated by a $3.8\,\mathrm{T}$ superconducting solenoid. Outside the superconducting solenoid, the iron return yoke with muon chambers is placed. The iron return yoke confines the magnetic field and stops all remaining particles except for muons and neutrinos. The muon system has 3 functions: muon identification, momentum measurement, and triggering. In the barrel, region detection is done using drift tubes while in the end-cap region it is done using cathode strip chambers. Both of these systems are completed by a dedicated trigger system of resistive plate chambers.

In total, the CMS detector has a diameter of $15\,\mathrm{m}$, a length of $28.7\,\mathrm{m}$, and weighs $14 \cdot 10^6\,\mathrm{kg}$.

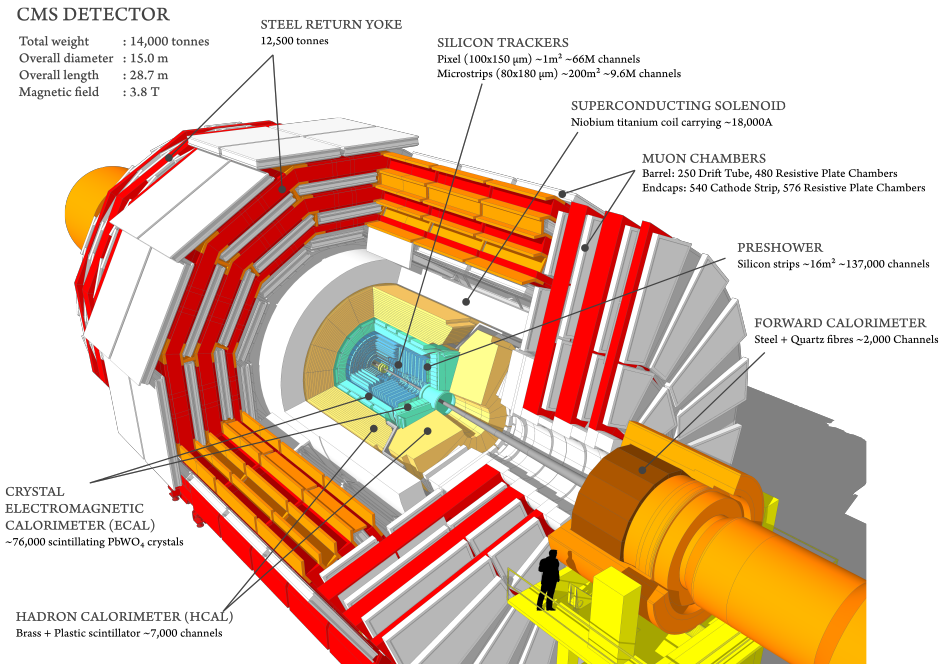


Figure 7.2: Shows the layout of the CMS detector [23].

The CMS is capable of detecting a wide range of particles using the collection of the data from each of its components. Different particles will follow a different path through the detector based on their charge, momentum, and trajectory. In figure 7.3 a path of the common particles can be seen. The superconducting solenoid enables the estimation of the charge and momentum of a particle. The charge can be determined by the bend direction of the trajectory because positively charged particles will bend opposite to negatively charged particles and neutral particles will not bend at all. The momentum can then be estimated by the degree of bending as faster-moving particles will bend less than slow-moving particles.

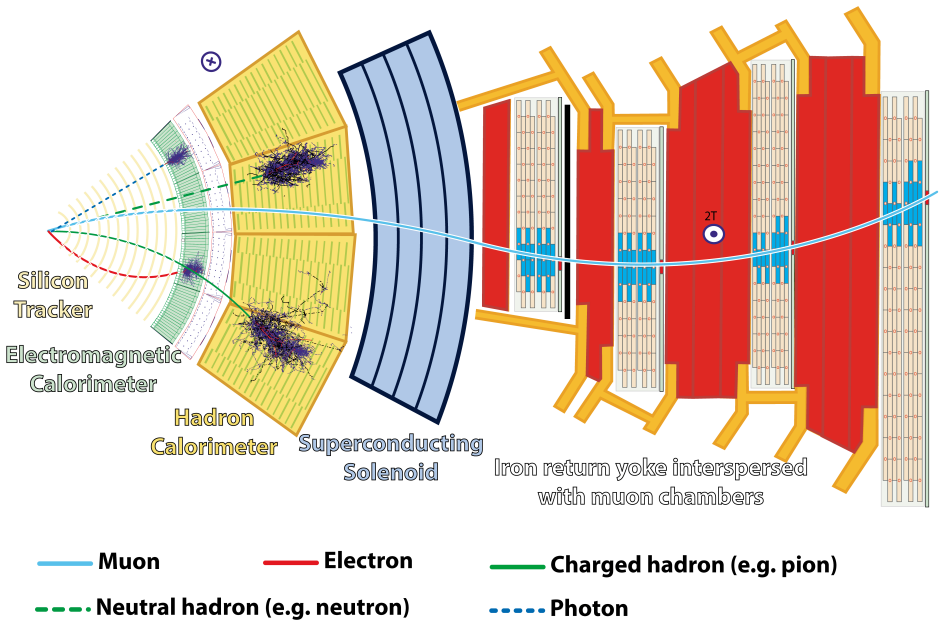


Figure 7.3: Shows the movement of different particles in the CMS detector [8].

The collision of charged particles in the LHC creates ionizing particles which over time will accumulate. The total ionizing dose (TID) expected after 10 years of operation has been simulated using FLUKA, a tool for monte carlo simulation of particle movement and their interactions. The expected dose decreases with distance from the collision point. A complete map of the expected TID for the detector can be seen in figure 7.4.

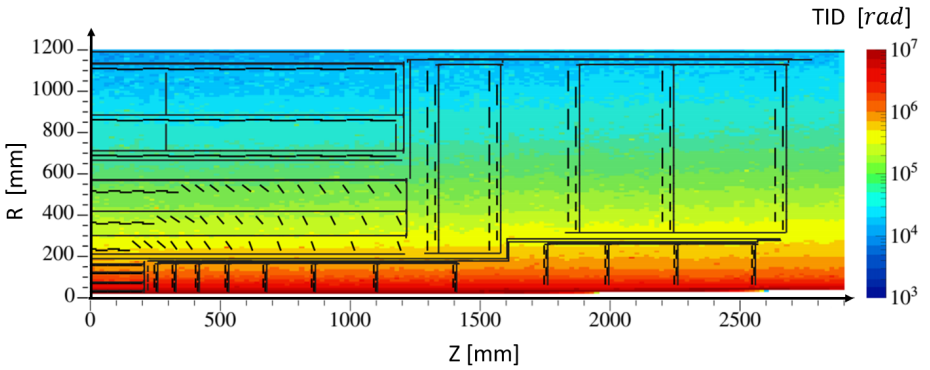


Figure 7.4: Shows expected total ionizing dose in Gy during a 10-year operation period of the CMS. This is simulated using FLUKA. [28]

The ionizing dose and charged particles passing through electronics can alter their behavior and affect the output in unwanted ways. These effects will now be discussed in detail.

7.2 Radiation Effects on CMOS Electronics

It is necessary to discuss and understand the effects of radiation on CMOS electronics due to the highly radioactive environment of the CERN accelerators. This understanding will lead to understanding the necessity and the methods for radiation hardening of the electronics. The radiation affects the CMOS in two distinguishable ways. In the form of cumulative damages and single-event effects (SEEs).

7.2.1 Cumulative Damages

Cumulative damages can be split into two subcategories. The first is non-ionizing processes, which come in the form of displacement of atoms in the lattice structure of the transistor. These are called displacement damages and are of little concern to CMOS technologies due to the high amount of doping [12]. The second is damages induced by ionizing doses, i.e. TID effects. MOS transistors can accumulate charges in the gate oxide, which creates a voltage difference on the gate and leads to unwanted biasing. This effect is dominant when the gate oxide is thick as it can contain a larger charge compared to the voltage threshold of the gate. Therefore, this effect decreases with smaller technologies as the gate oxide becomes thinner. The smaller technologies are instead dominated by effects like shallow trench isolation (STI) effects, which come in the form of radiation-induced drain-to-source leakage current and radiation-induced narrow channel effects (RINCE) [12]. Radiation-induced drain-to-source leakage current is caused by the accumulation of positive charges in the STI, which opens parasitic channels between the source and drain. This leads to an increase in leakage current. Positive charges are far more likely to be trapped due to electrons moving fast enough to leave the STI, while electron holes do not. However, over time electrons are attracted and enough electrons can be attracted to invert this effect. Therefore, initially, an increase in leakage current is observed, but as TID increases this effect reaches a peak and begins to invert. Since only positive charges are initially trapped, this effect does not increase the leakage current of pMOS. Instead, it repels the holes of the doped silicon increasing its threshold voltage and decreasing current flow. It is clear that increasing the length of the channel, decreases this effect as more charge is to be trapped before a channel can be opened. Therefore this effect is significant in smaller technologies with short gate lengths. However, this effect can be mitigated by using enclosed layout transistors (ELT), where the channel does not face the STI.

The other effect is RINCE, which is also due to the trapped charges in the STI. As positive charges are trapped in the STI, an electric field is created. This electric field leads to a decrease in threshold voltage for nMOS transistors and an increase in threshold voltage for pMOS transistors. However, as the width of the channel decrease, this effect becomes more dominant as the number of trapped charges does not change. This leads to a proportionally larger electric field, which signifies a dependency on channel width for this effect. For nMOS transistors, this effect is limited, as negative charges become trapped at the interface leading to the two canceling out similar to the inversion seen in radiation-induced drain-to-source leakage current at higher TID. However, for pMOS the trapped charges at the interface are also positive, leading to RINCE only increasing in potency with an increase in TID. As this effect is also due to charges trapped in the STI, it can be mitigated by the use of ELT [12]. However, these ELTs do use significantly more area compared to traditional designs.

7.2.2 Single-Event Effects

Single-Event effects can be split into two categories, permanent single-event effects, and single-event upsets (SEUs) (and transients (SETs)). A permanent single-event is the possible creation of parasitic transistor structures between two n-wells, i.e. between two transistors. This can potentially shorten VDD and ground, which can permanently damage the device. However, this effect is limited due to highly doped substrates and the use of STI between wells [22].

SEUs and transients are soft errors and not destructive to the die. Instead, they corrupt the information stored in digital logic circuits by flipping bits. SEUs become possible when the collected fraction of the charge liberated by an ionizing particle is larger than the electric charge stored on a sensitive node [22]. This critical charge scales with the gate area of the design. As the gate area decreases, the amount of stored charge representing a logical value of information decreases. In general, SEU sensitivity is increased by the scaling down of technology as node capacitance and the supply voltage are both scaled down as well [22].

A SET is an event, where a static combinatorial circuit is upset by a charged particle, leading to a glitch in the circuit. The time duration of SET is determined by the injected charge and the driving strength of the cell. If the output of this combinatorial circuit is sampled during the transient, a register can enter a metastable state, which can propagate through causing fatal errors [22].

7.3 Radiation-Tolerant Design

The goal of radiation-tolerant design is to limit the potential damage caused by radiation effects as described in section 7.2. Some of these effects can only be limited by a careful layout of the die or by the intrinsic properties of the chosen CMOS technology. However, SEUs can be reduced and mitigated by a combination of digital design. The methods for radiation hardening are usually dependent on redundancy either in space or in time.

A common method is the use of triple module redundancy (TMR). In this method, each memory element (i.e. register) and the corresponding combinatorial logic are instantiated three times. The outputs from these three registers are then passed to a voting system, which outputs the majority vote. The voting system itself is also triplicated to minimize the chance of a SET upset. If a single voter is used and is sampled during the SET, an error will occur in the system, thus leading to a single-point of failure. To avoid the build-up of errors in the 3 different paths, the feedback loop of a state machine should be taken from the voted result. An example of a TMR radiation hardening can be seen in figure 7.5.

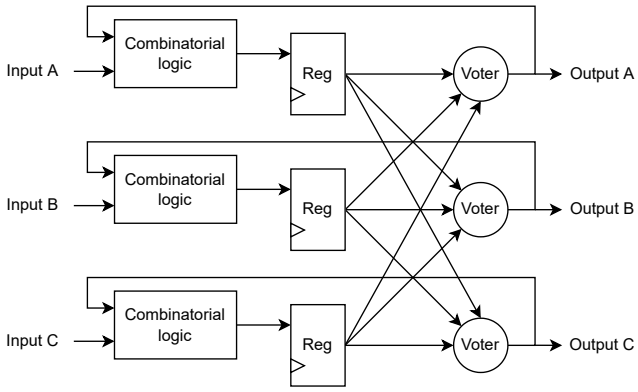


Figure 7.5: Shows a spatial radiation hardening technique using triple module redundancy.

This added redundancy would lose many of its radiation hardness benefits if the triplicated registers are placed close together on the chip die as this would increase the chance of an SEU happening on multiple registers from the same charged particle. Therefore the placement of these registers is restricted in the physical layout such that a minimum distance is enforced. From this, it is clear that TMR increases radiation hardness by using space and power consumption (increased due to increase in hardware) as a trade-off. This increase in power and area is not cheap as the heat generated needs to be transported away from

the detector and the space itself is limited inside the detector. Therefore to limit the disadvantages, the TMR is usually only done to the control path of a state machine. This is done as errors in the data path are limited in time, while an error in the control path can result in complete failure of the chip. At CERN, a tool has been developed for this method of radiation hardening named TMRG.

Another way of radiation hardening is the use of temporal spacing and is done by delaying the clock signal. The registers between combinatorial logic are triplicated, while the combinatorial logic itself, is not. Instead, the clock signal for each of the three registers is delayed, such that the SEU has a high statistical probability of having passed. This results in the SEU only affecting one register. It is then possible to use the same voting system as in TMR to achieve a corrected output. An example of this can be seen in figure 7.5.

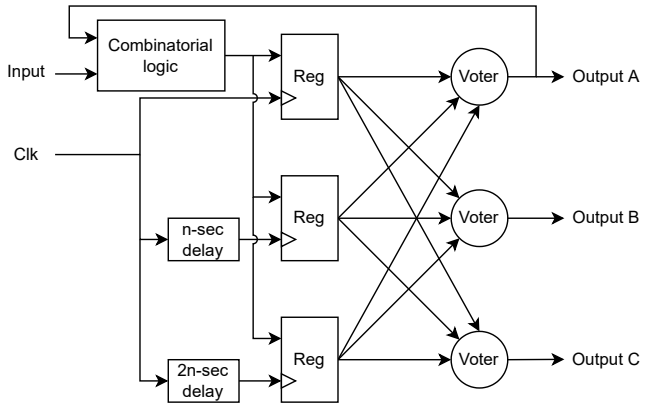


Figure 7.6: Shows a temporal radiation hardening technique where clock signals are delayed.

This method does not have the same minimum distance requirement as the TMR hardening technique as the registers sample at different timestamps. Instead, there is a temporal spacing requirement. This method does not require a triplication of the combinatorial logic and therefore saves on space and power consumption. However, it does make the timing analysis and closure difficult and this only gets more problematic as the frequency increases.

Even though temporal radiation hardening has multiple advantages in the form of space and power consumption, the TMR is chosen due to its simpler implementation. This is due to the problematic nature of timing closure for the temporal radiation hardening, but also due to the existence of an already-developed tool for performing TMR.

7.4 Universal Verification Methodology

The universal verification methodology (UVM) is an IEEE industry standard for the verification of design components [2]. It is developed by the Accellera group and its members. The goal is to create a modular, scalable and reusable generic verification environment. For these reasons, this methodology will be used for the verification of the SoC and its components. A short description of UVM will now be given.

UVM is based upon a hierarchy structure laid out by Accellera. This specifies guidelines for the creation of a verification environment and gives the support structure for this development. It does this by supplying a framework, for the designers to build on top of. This framework has the most general and essential features (Reporting, handshake mechanisms etc.), such that they do not need to be redeveloped for each project. This also ensures uniformity in testbench creation across many different work groups. The framework hierarchy is seen in figure 7.7 as it is laid out by Accellera [1]. Here the UVM agent can be expanded as it contains a sequencer, a driver, and a monitor. The expanded UVM agent can be seen in figure 7.8. Even though this is the recommended structure and should fit the most common use cases, the framework can be customized.

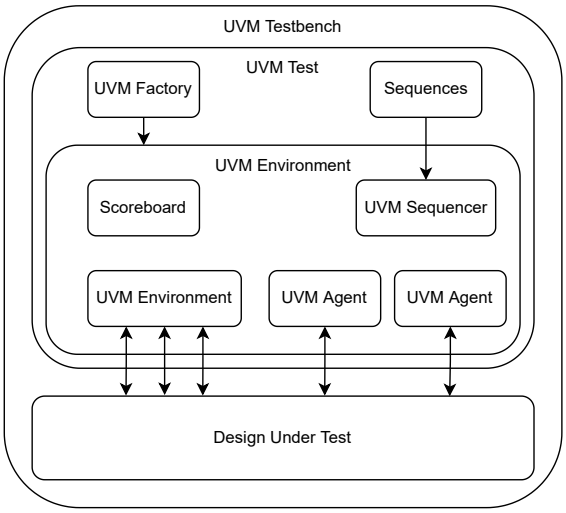


Figure 7.7: Shows the complete hierarchy of the UVM structure laid out by Accellera.

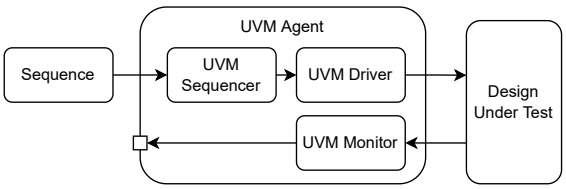


Figure 7.8: Shows the UVM components in a UVM agent and its connection to other components.

The communication between components is based on transaction-level modeling (TLM) and UVM items. The items are designed to fit the specific device under test (DUT). It contains the information necessary to create stimuli to the DUT and updating scoreboard and reference module. The item is sent between components using TLM.

Each of the hierarchy levels has a base class associated with them. It is on top of these base classes that the project-relevant components will be built. Each of these classes and their functionality is described below [1].

- **Testbench:** is the root class and container for all that needs to be simulated and tested. Typically this instantiates the DUT and the connections between the test and the DUT.

- **Test:** is the top-level UVM component. It has 3 main functions. To instantiate the test environment, configure the environment via a configuration database or factory overrides and apply stimulus to the DUT via the UVM sequences. This enables the designer to not have multiple instances of the same environment with different configurations for different test cases. Instead, the environment can be configured from this top-level UVM component to perform those test cases without repeating code.

- **Environment:** is a UVM component that instantiates and contains other reusable verification components such as agents, scoreboards, and other environments. It is also here the different components are connected and configured for default use.

- **Scoreboard:** is the verification component that compares the DUT to an implemented reference module. The scoreboard does this by receiving UVM items from the DUT via the UVM agent using TLM ports. It can then use the reference module as a predictor and compare that to the DUT.

- **Agent:** is a hierarchical component that contains other UVM components. These are typically a sequencer, a driver, and a monitor. The agent can either be active or inactive. An agent is active when it can drive stimuli to the DUT, which requires the use of a driver and sequencer. An inactive agent only contains a monitor.
  - **Sequencer:** controls the flow of sequence items from the multiple sequences to the driver, i.e. queues different sequence items according to a set of given parameters.
  - **Driver:** receives sequence items from the sequencer and converts them from transaction-level stimuli into pin-level stimuli for the DUT. For example, it can take a parallel data packet and transmit it via input pins to the DUT using a specified protocol.
  - **Monitor:** samples the DUT interface to convert data from pin-level stimuli into transaction-level stimuli. This data is then broadcast to the rest of the UVM testbench. The monitor can perform some levels of processing internally. For example, it can receive pin-level stimuli and decipher them according to a chosen protocol. So instead of broadcasting all pin-level activity, it first converts it into a specified UVM item and then broadcasts that using TLM.
- **Sequence:** makes up the core stimuli of the verification plan. A sequence can be made of multiple data items which can be used to create the scenarios for testing the DUT extensively. The items are eventually sent to a sequencer which will then queue it and send it to the driver. Multiple sequences can be connected to the same sequencer. The randomization tools in UVM are commonly used in creating data items such that there is a higher chance of discovering bugs. A sequence is not part of the component hierarchy. A sequence can contain other sequences, called a parent or virtual sequence.

A UVM environment can be developed either in SystemVerilog or SystemC, most commonly SystemVerilog. The SystemC variant is still under development but is in working condition. In this project, it has been chosen to use SystemC.

7.5 Physical Implementation of Digital ICs

The PicoRV32 SoC will be implemented in a physical design. Therefore, a description of the steps from a digital design to a physical implementation will

be given. The steps of physical implementation can in general be split into two categories: synthesis and implementation. Doing synthesis a Verilog design is converted from behavioral modeling into a gate-level description using only basic components supplied by a library. In the implementation, the gate-level description is converted into a physical implementation on a die with power distribution, a clock tree, and non-ideal components. Doing most steps, setup and hold conditions are checked for violations using different conditions. These are commonly referred to as corners. Corners are on-chip variations that alter the expected behavior of digital circuits and commonly include process, voltage, and temperature (PVT). The process variations are often summed and split into 3 categories slow (S), typical (T), and fast (F) for nMOS and pMOS transistors. 5 combinations (TT, SS, FF, SF, FS) of these corners can be evaluated to ensure that system works given variations and uncertainties. Setup and hold are tested in these corners, to ensure that the system behaves correctly even given on-chip variations. At CERN, radiation is included as an additional variation parameter. The TID effects are also evaluated in corners, using models developed at CERN for the behavior of chips doing radiation.

7.5.1 Synthesis

Synthesis is split into 3 steps: synthesis, mapping, and optimization. Doing synthesis a netlist is created from the supplied behavioral HDL model. For example, the always blocks in Verilog are converted into a basic set of components, e.g. flip-flop, AND-gate, NAND-gate. This step requires a list of constraints supplied by the designer. These constraints can be the clock period, output load capacitance, the maximum transition time of components, and setup/hold uncertainty on the clock. When this step is done, the Verilog code has been converted into a gate-level netlist that implements the same logic.

The next step is mapping. Doing mapping a gate-level netlist is converted from using generic components to using components given by a library called standard cells. This library contains detailed descriptions of common cells in an implementation technology. The technology used in this project is $28\,\mathrm{nm}$. Many libraries exist for the same technology, but with different characteristics, e.g. track number, cell width, and gate length. The number of tracks refers to the height of each cell. The cell width is the number of which each cell width has to be a multiple of. Three libraries will be used as the default for physical implementation. The libraries will be 9 tracks high, a multiple of $140\,\mu\mathrm{m}$ wide, and have a gate length of $35\,\mathrm{nm}$. The three libraries will have different threshold voltages. One will have a standard threshold, one will have an ultra-high threshold voltage and one will have a low-threshold voltage. The threshold

voltage controls the voltage needed for the transistor to switch state. This is controlled by the amount of dopping on the source and drain of the transistor. However, lowering the threshold voltage increases the leakage current of the transistor. Different threshold voltages are therefore a trade-off between speed and power consumption. By including multiple libraries, the algorithms are given more options for optimization, such that speed requirements can be met by using low-threshold transistors or power consumption can be reduced by using ultra-high threshold transistors. The libraries contain a description of the physical layout of the cell, a timing model describing the delay from input to output, and noise models. After mapping the gate-level netlist described using generic components has been converted into a gate-level netlist with standard cell components from a specific library.

After mapping, the last step of synthesis is optimization. Doing this step, different optimizations of the gate-level netlist are performed. This can be the removal of grounded circuits, removal of registers containing constants, or relocation of registers to reduce the amount needed. After all the steps have been completed estimates of certain parameters can be given for the digital design. This is things such as worst negative slack, power consumption, number of cells needed, and size of the complete design.

7.5.2 Implementation

Implementation is the next step after synthesis. The implementation converts the gate-level netlist into a physical layout on a die. The steps of this are floorplan, place, clock tree synthesis, route, design finishing, and verification.

Doing floorplanning, the die layout is designed. One of these variables is the die size. A rule of thumb is that the estimate at the end of synthesis should be 60% of the final size. However, this does not account for the size of the power rings. These are chosen to be $10\,\mu\text{m}$ wide for both ground and VDD. This results in a spacing of $25\,\mu\text{m}$ of every side from the edge to the area in which standard cells can be placed (Extra space is needed to allow spacing between GND and VDD rings). Power rings are used to ensure a uniform distribution of power to all parts of the chip and to reduce the effect of IR drops. As the height of a standard cell is given by the library, the die is divided into rows with that height. The top and bottom of these rows can then be connected to the power by having wires going across from side to side. This creates a power mesh that supplies all the rows of standard cells with VDD and ground connections. On large designs, these lines can become long. Therefore, to minimize IR drop vertical and horizontal larger stripes are used to connect these tracks at multiple points. In the floorplanning, the height of the die is also given in metal tracks.

For this project 9 metal (M1-M9) layers will be used. These do not have the same width. Instead, the top layer M1, has the highest routing width, M2 - M6 has the same width but is lower than M1, and M7-9 all decrease in routing width as the metal layer number goes up. M1 and M2 will be used for power distribution as these layers have the largest routing width and therefore the lowest resistance in the wires. M9 is used for the placement of standard cells and wiring. Usually, it is preferred to use the lower layers for wiring between standard cells as these signals do not draw significant current and the resistance is therefore not of concern. After this I/O ports are placed and if it is a top-level design, pads are also placed.

The next step is placement. Here the standard cells are placed in the rows. This is with the goal of achieving minimal congestion and the best foundation for timing closure. This is typically done by keeping connecting cells close together. The tool also has to do the placement with the minimum distance between triplicated registers.

The next step is clock tree synthesis. In this step, all the relevant components are connected to the clock signal. However, this is not a straightforward process. If the clock was just connected to all components without buffering, the clock edge would arrive at slightly different times depending on the placement of the component. Therefore the clock signal is continuously buffered to increase driving strength on long wires, but also to introduce delay in short wires, so that they arrive at nearly the same time, with the goal of not violating setup and hold time. This creates a tree-like structure. However, there is also a disadvantage to having all flip-flops latch at the same point in time, as this creates a large current spike which might cause IR drops and cause faults. Therefore if there is room in the setup and hold time, it is also an advantage to space out the latching in time, so that the current spike is limited.

After clock tree synthesis, the next step is routing. This is done after clock tree synthesis because swapping these two steps would mean that the clock tree would have suboptimal wiring due to possible high congestion by routing. In the routing step all the connections specified by the gate-level netlist are established. If these wires are long, buffers are inserted to increase driving strength.

The design has now been placed and routed. Power and clock distribution have been established and in theory, the physical implementation process is finished. However, a few steps still remain to verify the design and ensure manufacturing is possible. For example, metal is filled into empty areas of the die. Without this sinkholes would be created and uneven metal layers would be laid on top, this would only get worse as more and more layers are added if nothing is routed or placed in that region. Therefore, metal is placed to create an even

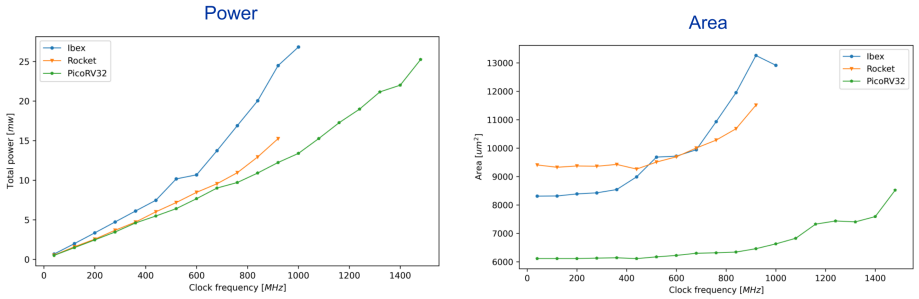
surface on which the metal can be laid. Next, design rule checks (DRC) and layout vs schematic (LVS) are performed. DRCs are manufacturing rules and safety protocols to ensure that the layout can actually be manufactured by the foundry. It can be simple things such as wire proximity checks, no short circuits, etc. However, there are also many complex rules. Things such as the area of a wire on a specific metal layer might be so big that it collects enough charge doing manufacturing that it destroys the connected component (Antenna effect). Another example is the concept of electromigration. When current is transported through a wire, it will slowly move material around. This effect increase with the amount of current. If this effect is too large, it means that some parts of the wire might become shallow and thereby increase resistance changing the behavior of a sensitive circuit. It can also short-circuit to a nearby wire if it gets too wide at places. It, therefore, checks if a wire is in danger of changing properties over a given lifetime. LVS checks whether the resulting layout at the end of the physical implementation is equivalent to the digital description from which it was derived.

With the physical implementation complete, we can generate an SDF file. This file contains all the propagation delays, clock arrivals, and such for the gate-level netlist. This can be used in combination with a testbench and the circuit can be verified to ensure that even given non-ideal clock signals and propagation delays, it still behaves as expected. Using this testbench and SDF file, a more accurate estimate of the switching activity of each transistor can be derived. This can be used in a dynamic power analysis to obtain a far more accurate estimation of power consumption. Before this, the tool just assumed that all transistor switches with a 20% probability on each clock edge. This is the final step of physical implementation. All of this can then be collected in a set of files to establish a library for the implemented block. This can now be manufactured or used as a component in a larger design.

7.6 The PicoRV32: System on Chip

The goal of this project is to make a simple demonstrator SoC, with a CPU, memory blocks, and standardized interconnect bus with a basic set of peripherals. This demonstrator chip will then be used to test whether it is possible to create a radiation-tolerant SoC with a reasonable amount of area and power usage. Computing power is not of concern as its use case is targeted toward control and monitoring. RISC-V is an open-source instruction set architecture (ISA) that perfectly fits this application. Many different open-source SoC foundations have been considered and 3 stood out as candidates. Those are the Rocket

Chip from UC Berkeley, the PicoSoC, and the Pulpissimo by ETH Zurich. The Rocket chip is a promising chip-building framework in Chisel. The PicoSoC is simple and written in pure Verilog, making it easier to understand. The Pulpissimo is made by ETH which has been collaborating with CERN before and therefore enables more direct communication with the designers. These three have been compared on their power and area usage during post-synthesis. The result of this can be seen in figure 7.9. Based on this comparison, the PicoSoC is chosen for the demonstrator chip.



(a) (b)

Figure 7.9: Shows the post-synthesis power and area comparison of an Ibex, Rocket, and PicoRV32 core.

The PicoRV32 SoC is based upon the RISC-V open-source CPU: PicoRV32. This core is meant to be used as a size-optimized auxiliary CPU in an FPGA or ASIC design. It does not have high computational power, but it is small and simple. Due to its simplicity, it is also easier to debug and develop extra features.

RISC-V is an open-source instruction set architecture (ISA). Its architecture is developed on Reduced Instruction Set Computer (RISC) principles. This is in contrast to the Complex Instruction Set Computer (CISC), to which the commonly known family of x86 ISAs belongs. The two design topologies differ because a CISC instruction often executes several lower-level instructions, while a RISC architecture does not. RISC-V employs a base set of the most needed instruction, while several extensions are available to expand the instruction set. The PicoRV32 is configurable [105]. Its instruction base can be based on either RV32I (32-bit, base integer instruction set) or RV32E (32-bit, base integer embedded instruction set). The RV32I is capable of having two extensions added: multiplication (M) and the compressed instruction set (C). There is also an optional built-in interrupts controller. However, this is not based upon the RISC-V standard and is instead custom-built for the PicoRV32. This design choice was made because the RISC-V interrupt handle was extensive

and comprehensive, so a simpler IRQ handling with less hardware overhead is available.

At the start of this project, several things were already implemented: The core, two buses, a bridge connecting the two busses, and a temporary memory block. A Native Memory Interface (NMI) connects the core and memory. The NMI is connected to a bridge that is connected to an Advanced Peripheral Bus (APB) interface. Peripherals will be connected to SoC using the APB interface. The state of the SoC at the beginning of this project is visualized in figure 7.10.

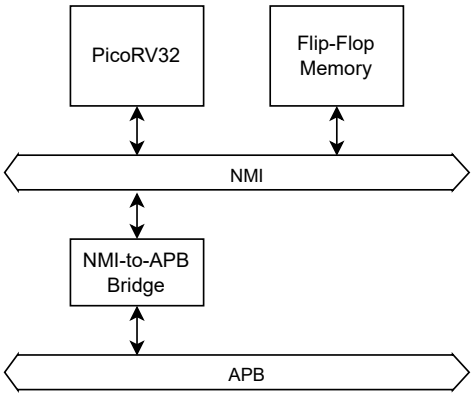


Figure 7.10: Shows the state of the PicoRV32 SoC at the beginning of this project.

A detailed description of the two interfaces used on the buses is now given.

7.6.1 Native Memory Interface

The NMI is an interface defined by the PicoRV32 [105]. It is a simple valid-ready interface bus. It requires five outputs from the PicoRV32 core and two outputs from the slave, which is receiving the transfer. These signals and their functions are described in table 7.1. This bus is only used to connect the most essential and critical blocks to the core, e.g., memory and bootloader.

Table 7.1: NMI signal descriptions

Signal	Source	Description
CLK	Clock source	Clock. The transfer is completed on the rising edge.
VALID	PicoRV32 core	Valid. The core uses the valid signal to initiate a transfer. All core outputs are stable while valid is high.
INSTR	PicoRV32 core	Instruction fetch. Used by the core to indicate if the memory transfer is an instruction fetch
READY	Slave interface	Ready. Asserted by the slave when the read data is available and used to acknowledge a write transfer.
ADDR	PicoRV32 core	Address. The core supplies the address which is used by the slave to read or write to the requested cell.
WDATA	PicoRV32 core	Write data. If a write transfer is being performed, the core supplies the data to be written using this bus
WSTRB	PicoRV32 core	Write strobe. If the write strobe is 0, it indicates a read transfer, while it being non-zero indicates a write operation. The write strobe signal is used to write specific bytes of the wdata. It is possible to write 32 bits, the upper 16 bits, the lower 16 bits, or 8 bits.
RDATA	Slave interface	Read data. In the case of read transfer, this is the data read from the specified address. When rdata is available, ready is asserted.

This bus is fully triplicated to achieve radiation hardening. This is decided due to it being critical infrastructure and also the length of this bus being limited since peripheral devices are not connected to this bus.

7.6.2 AMBA APB Interface

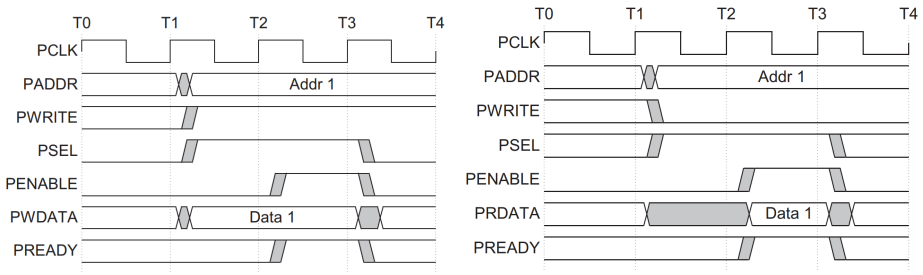
The APB is designed by ARM and is part of the Advanced Microcontroller Bus Architecture (AMBA) protocol family. This protocol has been chosen for the SoC since it is designed for minimal power consumption and reduced interface complexity [5], which aligns with the goals of the project. The list of signals in the protocol can be seen in figure 7.11.

Table 2-1 APB signal descriptions

Signal	Source	Description
PCLK	Clock source	Clock. The rising edge of **PCLK** times all transfers on the APB.
PRESETn	System bus equivalent	Reset. The APB reset signal is active LOW. This signal is normally connected directly to the system bus reset signal.
PADDR	APB bridge	Address. This is the APB address bus. It can be up to 32 bits wide and is driven by the peripheral bus bridge unit.
PPROT	APB bridge	Protection type. This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.
PSELx	APB bridge	Select. The APB bridge unit generates this signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required. There is a **PSELx** signal for each slave.
PENABLE	APB bridge	Enable. This signal indicates the second and subsequent cycles of an APB transfer.
PWRITE	APB bridge	Direction. This signal indicates an APB write access when HIGH and an APB read access when LOW.
PWDATA	APB bridge	Write data. This bus is driven by the peripheral bus bridge unit during write cycles when **PWRITE** is HIGH. This bus can be up to 32 bits wide.
PSTRB	APB bridge	Write strobes. This signal indicates which byte lanes to update during a write transfer. There is one write strobe for each eight bits of the write data bus. Therefore, **PSTRB[n]** corresponds to **PWDATA[(8n + 7):(8n)]**. Write strobes must not be active during a read transfer.
PREADY	Slave interface	Ready. The slave uses this signal to extend an APB transfer.
PRDATA	Slave interface	Read Data. The selected slave drives this bus during read cycles when **PWRITE** is LOW. This bus can be up to 32-bits wide.
PSLVERR	Slave interface	This signal indicates a transfer failure. APB peripherals are not required to support the **PSLVERR** pin. This is true for both existing and new APB peripheral designs. Where a peripheral does not include this pin then the appropriate input to the APB bridge is tied LOW.

Figure 7.11: Table of signals in the APB protocol (Courtesy of ARM [5]).

The protocol contains two individual buses for read and write operations. However, only one of these transfers can be executed at a time. The read and write transfer using the APB protocol with no wait cycles can be seen in figure 7.12.



(a) Write transfer (b) Read transfer

Figure 7.12: Shows the basic transfers of the APB protocol with no wait cycles (Courtesy of ARM [5]).

Wait cycles can be introduced if the slave does not assert the PREADY signal. However, the way the APB protocol is connected to the CPU will cause stalling of the entire CPU until the PREADY is asserted and the transfer is completed. A timeout could partially solve this by limiting the stalling period, but this does not fix the problem but only reduces it. Therefore it is chosen that the IP block as a general rule should always have PREADY asserted, as to ensure no stalling of the system. Instead in the case of a bad transfer due to no data available or similar situations, the PSLVERR signal is utilized and the IP block will then assert this signal to indicate that the transfer has failed. This introduces problems as this signal does not indicate why the transfer failed. However, it does remove the stalling. Therefore, this method is chosen.

The APB bus will not be triplicated as this will connect all peripherals to the core and the length can therefore be quite significant, which will result in a larger area used and more complex routing. Instead, an encoding approach is used. Every byte of the bus is encoded using Hamming codes, which are capable of single error correction and double error detection. This approach uses the same logic developed later for radiation-tolerant memories.

7.6.3 Design of the Peripherals

7.7 Timer

In this section, a timer IP block will be designed. The functionalities of the timer are inspired by the ST's timers [87]. The functionalities should cover a large number of general use cases. The timer will consist of two functional parts: the counter and multiple capture/compare registers. To describe the functionalities which need to be implemented in these two functional units, a list has been made which can be seen below.

- 16-bit up/down, on-the-fly adjustable auto reload counter capable of counting center-aligned modes (CMS).
- 16-bit prescaler on-the-fly adjustable.
- 4 independent capture/compare channels with on-the-fly update.
- Capture, compare equality, compare greater than, compare toggle & one-pulse mode for the capture/compare registers (CCR)
- Interrupt generation on update event, trigger event, input capture or output compare.

- Input capture filter and edge detector.

These features will now be elaborated to ensure a common understanding:

- Update event (UEV): Describes when the main counter inside the timer is reset, either from overflow or underflow.
- Auto-reload: Gives the attached counter the ability to reset to or on the value of this register, depending on the counter mode.
- On-the-fly adjustable: Normally the auto-reload value, the prescaler value, and the CCR value would be synchronized to update on a UEV, however using on-the-fly it is possible to update these values in the middle of counting progress.
- Center-aligned modes: refers to a operational mode of counter where it automatically switches between counting up and down every time it overflows/underflows. There are 4 different modes depending on which direction it should update the output.
- (Input) Capture: refers to an external event triggering a capture thereby saving the value of the counter into the CCR.
- (Output) Compare: refers to the comparison between the CCR value and the counter value, when a specified criterion is met, i.e the counter value is greater than or equal to the CCR value, then a specified output signal is asserted.
- One-pulse Mode: refers to the system creating a single pulse at a specified time and then ending it at another specified time and stopping either the counter itself or disabling the relevant CCR.

From this list of features, a figure is constructed showing the different high-level blocks of the timer, which is seen in 7.13.

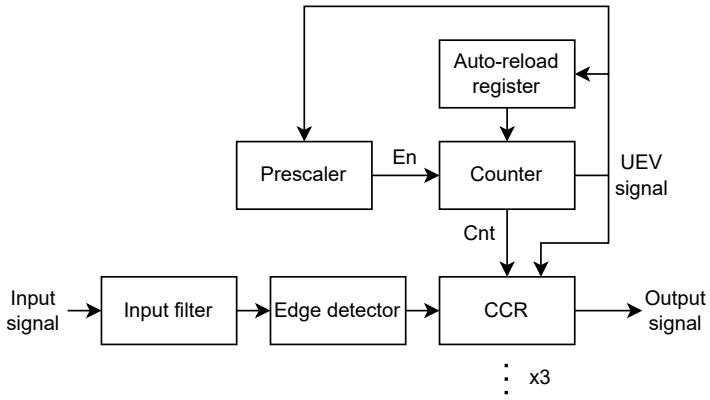


Figure 7.13: Shows the functional blocks of the timer. The CCR register is repeated 3 times, for a total of 4 CCR registers.

A list of control registers is used to program the timer to the intended application. A detailed description of all programmable registers and status register can be found in the RDL file for the timer in the GitHub repository.

The amount of capture/compare registers is potentially unlimited, however it is chosen to use 4 as a standard following the ST Microelectronics timer [87]. The development of the timer will be split into two: the timer part consisting of a counter and prescaler, and the capture/compare part.

7.7.1 Counter and Prescaler

The timer itself will have a 16 bit counter as its central component. This counter is a bidirectional, meaning it is capable of both counting up and counting down, which is controlled by a direction signal. Connected to this counter is 16 bit prescaler and a 16 bit auto-reload register. The prescaler is also a 16-bit counter, where the value of reset can be programmed from the APB interface. When the programmed value is reached the prescaler pulls a enable signal high for one clock cycle. The enable signal is sent to the counter, which then counts up/-down on the next positive clock edge. Using the prescaler as an enable signal instead of using it to lower the frequency of the system clock, means that no extra clock domains are created, making timing closure easier in the physical implementation. To program the reset value of prescaler, a shadow register is used to ensure that the updating of this value happens at the correct timing. A shadow register is an extra register inserted between the programmable register and the compare circuit used for comparing current count and reset value. Without this, the value would change instantly, the moment the programmable register was written. In some cases this might be preferable, but in other cases control of when this reset value is updated might be needed. This control is

implemented in two ways: on-the-fly update and using UEVs as triggers. On-the-fly refers to operation similar to that of using no shadow register resulting in the value being updated instantly, while using UEVs the update will happen when the main counter either overflows and/or underflows.

Besides a prescaler, a 16-bit auto-reload register is also connected to the counter. This auto-reload register controls what value the counter will reset at or reset to depending on the direction. When the counter is counting up it will reset at the value specified in the auto-reload register and when it is counting down it will reset at 0 and take the value of the auto-reload register. This auto-reload register also has a shadow register to control the timing of updating the value. It has the same possible operations as the prescaler, either on-the-fly or on UEVs. Doing CMS mode, the operation of the auto-reload register is different to ensure smooth transistions between counting up and down. For example when the counter is counting up, it will reset at the auto-reload value subtracted one. It then resets to the auto-reload value and inverts the direction bit. When it is counting down, it will reset to 0 and invert the direction bit when the counter has the value 1. This ensures that no value will be repeated in a cycle.

7.7.2 Capture/Compare Register

The CCR register is the functional unit that generates the output signals of the timer and also receives external signals. The CCR register can be configured to one of these two modes: capture or compare. These modes can be further subdivided or customized to fit the specific application. We will start by discussing the compare part.

The compare part of the CCR generates a output on a single line. This line will continuously switch between logic high and low depending on the operational mode of the CCR, the value to compare against, and the current value of the counter. These things enable the CCR to create a controllable output signal and even enables it to create PWM signals, which are commonly used in control of motors. The value switch the CCR uses to compare against the current counter value is stored in a programmable register equipped with a shadow register identical to that of the prescaler or auto-reload register, such that this compare value can be updated on-the-fly or during an UEV. The operational modes can be controlled by a programmable register and are as follows: deactivated, equal to, greater than, toggle, and one-pulse mode. In "equal to" the output goes high if the compare value is equal to the counter value. In "Greater than" the output goes high if the counter value is greater than the compare value. In "toggle" the output gets inverted every time the compare value is equal to the

counter value. In "one pulse mode" the output goes high when the compare value is equal to the counter value and goes low on a UEV. This also deactivates the CCR or the entire counter based on the value in the corresponding programmable register. One has to be aware of the fact that these comparison is not guaranteed to be at all times when the counter is in CMS mode. In this mode the comparison is either done only when counting up or when counting down or at all times. The increase the control of the output a controlled inverter is also added on the output, such that output signal can be programmed to be inverted or not.

In capture mode, an output is not generated. Instead the CCR waits for a edge on a input line and when this edge is received it saves the current value of the counter into the shadow register of the compare register (The same as used in compare mode). This can be used to generate an interrupt and the value be read through the APB interface. This can be used for a number of things, but most intuitively this can be used to measure the time between two events. To further control the capture mode, a programmable edge detector and a programmable input filter is added. The edge detector goes high when a rising edge or falling edge is detected. This is done by sampling the state of the input signal every clock cycle into a register and comparing this to the current input signal using an AND-gate. If one wish to detect a rising edge, the input of the AND-gate coming from the register should be inverted, while the direct input should not. This way, the AND-gate will go high if the last state of the input signal is low and the current input signal is high. This circuit can be seen in figure 7.14. A similar circuit can be made for a falling edge detector.

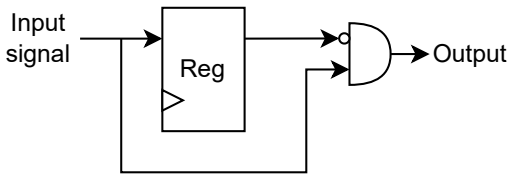


Figure 7.14: Shows a positive edge detector circuit. The output will go high when the previous state was 0 and current state is 1, thereby detecting a positive edge.

The input filter is made using a programmable 16-bit counter. This counter will count up if the current input value is equal to the last input value. If they are different it will reset to 0. If it reaches the value in the programmable register of counter it will output the value of the last state. This creates a programmable low-pass filter, which filters out the high frequency digital input signals. If the filter is not enabled, it will simply always pass the current state value.

Before the input signal reaches the input filter, it will have to go through a synchronizer. A synchronizer is needed as the input signal will come from an external source which might be asynchronous to the system clock and can have another clock frequency or clock phase. This is called clock domain crossing (CDC) [66]. Without a synchronizer this would mean that the input signal could be sampled at a time where setup or hold is violated and a metastable state is induced in the register. This metastable state could then propagate through the CCR and create a fatal error or malfunction. To avoid this a 2 flip-flop synchronizer is used. The is that this gives the input flip-flop time to exit the metastable state and the next flip-flop would then sample a stable flip-flop and thereby avoiding the propagation of the metastable state. However, if the clock frequency is high, the next flip-flop might still sample the first flip-flop in a metastable state. More flip-flops can be added to create a longer synchronizer and decrease the probability of a metastable state propagating. A 2 flip-flop synchronizer will be used here. The resulting circuit of combining the synchronizer and the input filter can be seen in figure 7.15. From this figure it is clear that this input filter and synchronizer adds a 4 cycle latency period. However, this delay is known and can therefore be subtracted if necessary.

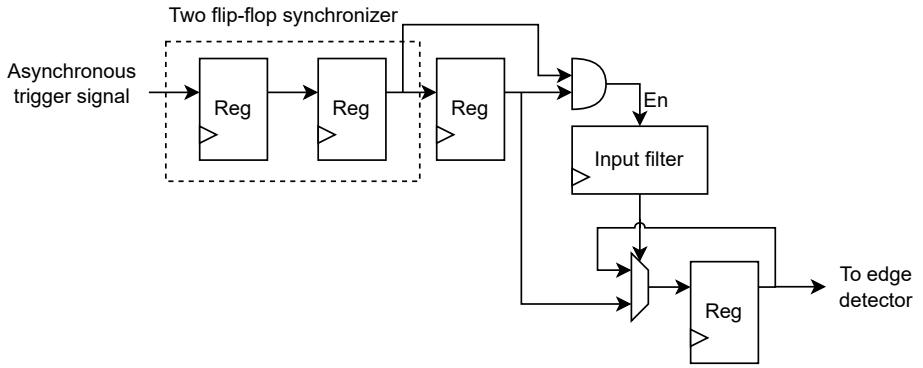


Figure 7.15: Shows the input capture stage with a 2 flip-flop synchronizer and a input filter.

7.7.3 UVM Verification

The timer has been continuously verified using the PicoRV32 core and C++ to ensure basic functionality. To achieve a more extensive verification of the timer a UVM environment is designed. This timer has a APB interface and its own interface containing the input capture signals and the output compare signals. The APB agent has already been designed and can be reused here. That leaves

a timer agent, a scoreboard with a reference module and test sequences to be designed.

The timer agent will monitor the compare output signals and drive the capture input signals. The capture input signals will be generated randomly with a certain probability of inverting every clock edge. This probability can be adjusted to achieve the wanted amount of capture triggers. This will be done by the UVM driver. The sequencer used will be the standard UVM sequencer as it will only be driven by a single test sequence and therefore there is no need for sequencing. The monitor will sample the output compare signals every clock cycle and sent them to the scoreboard.

The scoreboard samples the APB interface and timer interface using the corresponding agents. These samples are then used to update the reference module. The reference module proved to be a challenge to design. It was the goal to design a timer with mostly the functionality but using a high-level approach such that the probability of programming the same errors in the reference module as in the timer was lowered. Therefore, an approach using time stamps to calculate the current count and update the output compare was attempted. However, this approach started to fall apart as more of the features were added as the timing and updating of these were required to be quite accurate to achieve a behavior similar to that of the timer. It also ended up needing to update the outputs every clock cycle and thereby losing most of the efficiency gained by calculated the time between updates and then waiting. Therefore, an approach much closer to the actual timer was chosen, even though this increased the risk of programming the same errors twice and thereby lowering the probability of finding them. This approach was done in C++, with a while loop of counting up every rising edge was the central part. On this other features such as the prescaler, the auto-reload register, and the compare mode of the CCR was built. The capture mode was not implemented.

The test sequence written was designed using the APB interface. There was no need to design a test sequence for the capture input as the this mode has not been implemented in the reference module. The test sequence comprised of putting the four connected CCR register into the four compare modes: equal to, greater than, toggle, and one-pulse mode. The sequences then, at random intervals, reads the counter value and compares that to the reference module. At every clock cycle the compare output is compared to the reference module. Furthermore, the auto-reload register and the prescaler is also tested, by writing random values to these with random intervals.

The coverage report generated by this UVM environment is seen in appendix **??**. Following this, the timer was triplicated and the UVM test was repeated. An identical coverage report was generated testing the triplicated timer.

7.8 SPI Master

A serial peripheral interface (SPI) master is to be designed and connected to the CPU using the existing APB interface. SPI is a synchronous protocol, in comparison to UART, which is asynchronous. This is due to the master transmitting a clock signal, which synchronizes sampling and transmission between the slave and master. This clock signal will be labeled SCLK. Besides the clock signal, three other signals are present in the SPI protocol: Master Out Slave In (MISO), Master In Slave Out (MOSI), and Slave Select (SS) [88]. Slave select is a collection of wires, one for each slave, which goes to its respective slave and is pulled low, when that slave is to be communicated with. The protocol itself is quite simple and is executed by first pulling SS low for one of the connected slaves, then enabling the clock signal SCLK, which initiates the data transfer on the MISO and MOSI line starting with MSB. The exact timings of these signals can be seen in figure 7.16.

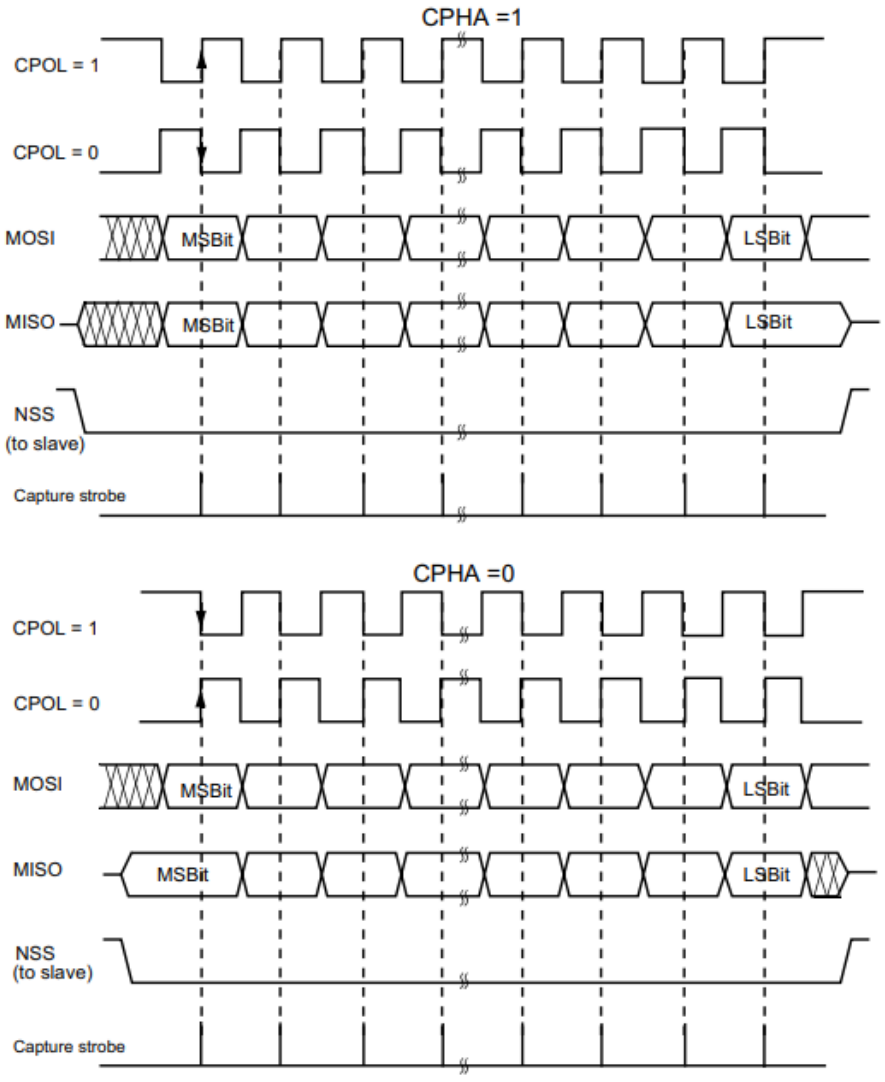


Figure 7.16: Table of signal timings in the SPI protocol for different values of CPHA and CPOL (Courtesy of STMicroelectronics [88]).

Due to the simple protocol, the essential hardware is equally simple. The main component is a shift register which will perform the transmission. Besides this two buffer registers is needed for holding the message and saving the message coming from the slave. A baud rate generator is also needed to create the SCLK signal, which is essentially a counter. Only a single shift register is in theory needed, as the slave also contains a shift register and these two can be connected in a ring connection using MOSI and MISO. An example of this ring connection can be seen in 7.17.

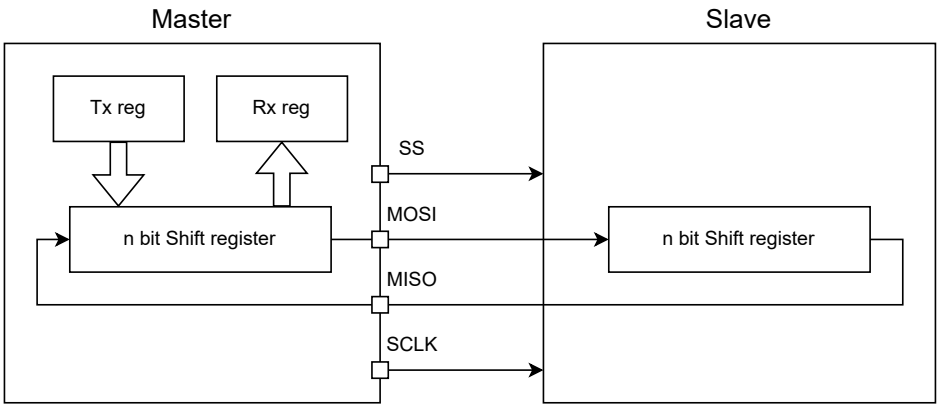


Figure 7.17: Shows the ring connection established by connecting MOSI and MISO to shift registers.

A few modifications and additions will be made to this setup to make it easier to use and to simplify the development. First is the addition of two FIFOs instead of a single transmission register and receiver register. This means that many messages can be written over a short time frame without the need to let the transmitter finish sending the previous message. The rx FIFO removes the necessity of reading the master peripheral each time a new message is received and limits the risk of losing data if new messages are received. Furthermore, even though it is possible to do it with one shift register in the master controller, it is chosen to do it using two shift registers, one for tx and one for rx. This is done to simplify the controller needed for executing the protocol. The extra cost in form of a shift register is deemed insignificant. For selecting the slave it is chosen to use a register for controlling which wire is pulled low. The resulting master block diagram can be seen in figure 7.18.

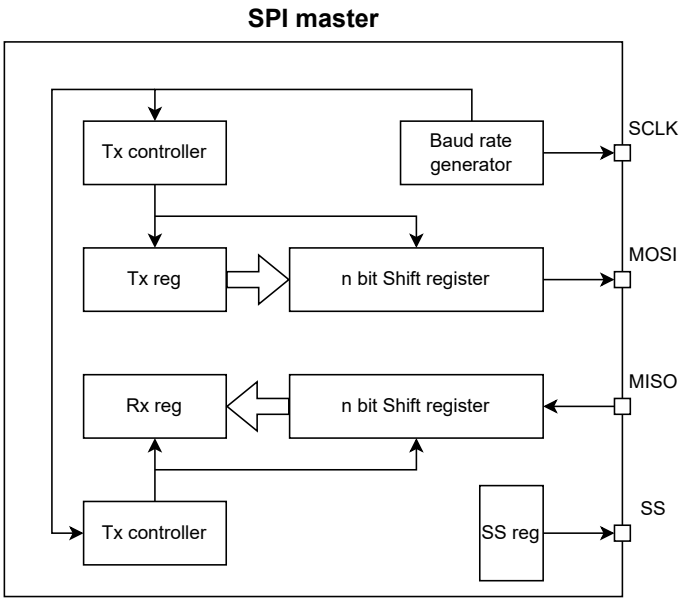


Figure 7.18: Shows functional block diagram of the SPI master that is to be developed. The hollow arrows symbolize a bus connection which is 32 bits wide.

With the functional units established for the system, it is time to discuss the programmable control registers and status registers for the system. The system should be adjustable to a high degree of freedom to ensure that it has a high amount of use cases. First, it should be possible to adjust the clock frequency of the system by controlling the baud rate generator. This will be done in the power of 2's, so that the clock is divided by 2, 4, 8, and so on. Up to and with 2^{16}. A minimum division of 2 is needed, since the tasks in the form of sampling and shifting, are performed in every period of the SCLK. This means that at least two rising edges from the system clock are needed since everything should update on the system clock and on rising edges. This allows a high range of possible frequencies for the system to run at. It is also worth mentioning here, that the baud rate generator will not generate a clock signal for the system, but will function as an enable signal for the rest of the system which still runs on the system clock. This avoids creating multiple clock domains.

It should also be possible to adjust the size of the data frames used in the communication, i.e. how many bits are sent between master and slave. This is chosen to be 8, 16, 24 or 32 bits. To this an enable signal is also added, such that the block will not start a transmission before it is intended. Two more control signals are added, which are commonly found in an SPI master, which is clock phase (CPHA) and clock polarity (CPOL) [88]. These two signals both affect how the SCLK signal is generated and how the shifting and sampling is

done. The effects can be seen in figure 7.16. These control signals make the SPI communication highly flexible and it is, therefore, able to communicate with a wide range of different systems.

All control signals from before are bundled into 1 programmable control register. In addition to this, a status register is added, containing the full and empty signals from the two FIFOs and a busy signal to indicate whether a transfer is in progress. To complete the list of registers available to the APB interface, a read and write register is added, which writes to the tx FIFO and reads from the rx FIFO. This list of registers is described extensively in RDL file for the SPI master in GitHub repository.

The programmable control registers and the functional units have been presented and development of the system can begin.

7.8.1 Development

The functional units will be developed in Verilog with a focus on behavioral development, i.e. no direct instantiation of adders, registers, etc. The development has been split into 5 groups: the APB interface, the FIFOs, the rx controller, the tx controller, and the baud rate generator. The FIFOs will be reused from the UART (see section 7.9).

TX & RX Controller

The tx and rx controllers for the shift registers are simple in design as the ticks needed for performing their respective action are generated by the baud rate generator. Therefore it only needs two states: idle and tx. In idle it simply keeps checking the value of tx_start and if it's true, it will load in the data available in the FIFO and set a variable t to 0 and set high a tx_data_read. The tx_start is the output of and gate, where the two input signals enable and tx_fifo_not_empty. The tx_data_read is set high for one clock cycle to indicate to the FIFO that the value has been read. The next state is the tx. In this state, it performs actions every time a tx_tick is received. Every time a tick is received, it checks how many times it has performed a shift operation. If it has done it x amount of times it sets a tx_done signal high and checks whether tx_start is high. If it is high it sets t to 0 and loads in new data and generates a high tx_data_read signal. This is done to enable continuous transmissions instead of resetting to the idle state between each transmission. However, if tx_start is not high, it goes back to the idle state. If t was not equal to x in the first place, it adds 1 to the value of t and performs a left shift operation on the data. The value of x is decided by the DFS register, which selects the size of the data frame for transmission. All of

this information is encapsulated in the ASM chart for the tx controller seen in figure 7.19.

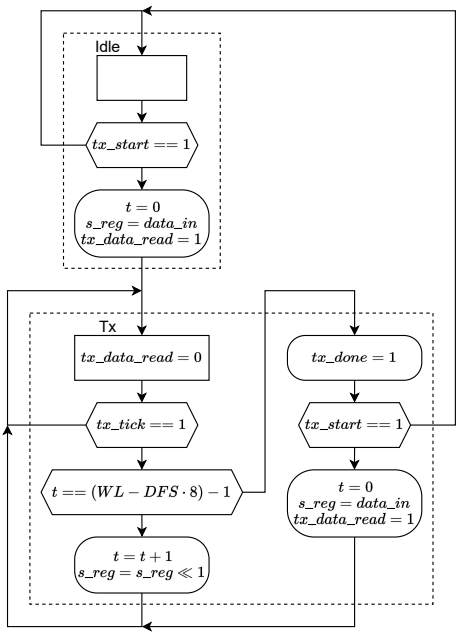


Figure 7.19: Shows the ASM chart for tx controller for the SPI master.

A nearly identical ASM chart can be made for the rx controller. Instead of shifting out data on the tx_tick, it samples data on the rx_tick into a shift register. When it has sampled that x amount of times, it saves it into the rx FIFO.

Baud Rate Generator

The task of the baud rate generator is to create the clock signal: SCLK. To do this it has to take the state of the enable signal, the state of the FIFOs, the desired clock frequency, and the CPOL and CPHA signals. Here, it is worth highlighting how the desired clock frequency and CPHA are incorporated into the system. The clock division can be achieved by a simple 16-bit counter from which the output can be taken from a single-bit position depending on the desired division. Therefore the SCLK can simply be generated by connecting all bit positions from the counter to a multiplexer, where the select bits are connected to the BR_DIV register. However, the output does need to be inverted, as the clock signal has to start with a change, i.e. rising edge in the case of no polarity changes. This is not achieved by the counter itself as it needs to count halfway before a change is made. To fix this, the output from the counter is inverted.

To achieve correct timing of sampling and shifting, the counter will be reset at the value $2^{BR\_DIV+1} - 1$, and either a tx tick or an rx tick will be produced. Which one depends on the value of CPHA. One is added to the exponent since the smallest divider is 2. At the halfway point of the count, the other tick will be produced. To incorporate CPHA fully, it will decide one of two states enter when coming from idle. These two states will differ on whether the clock is activated instantly or delayed by half a clock cycle. Thereby, the desired phase change is achieved.

With the details discussed and elaborated, an ASM chart is made as seen in figure 7.20, which encapsulates the operation of the baud rate generator. The Verilog code is written based on this ASM chart.

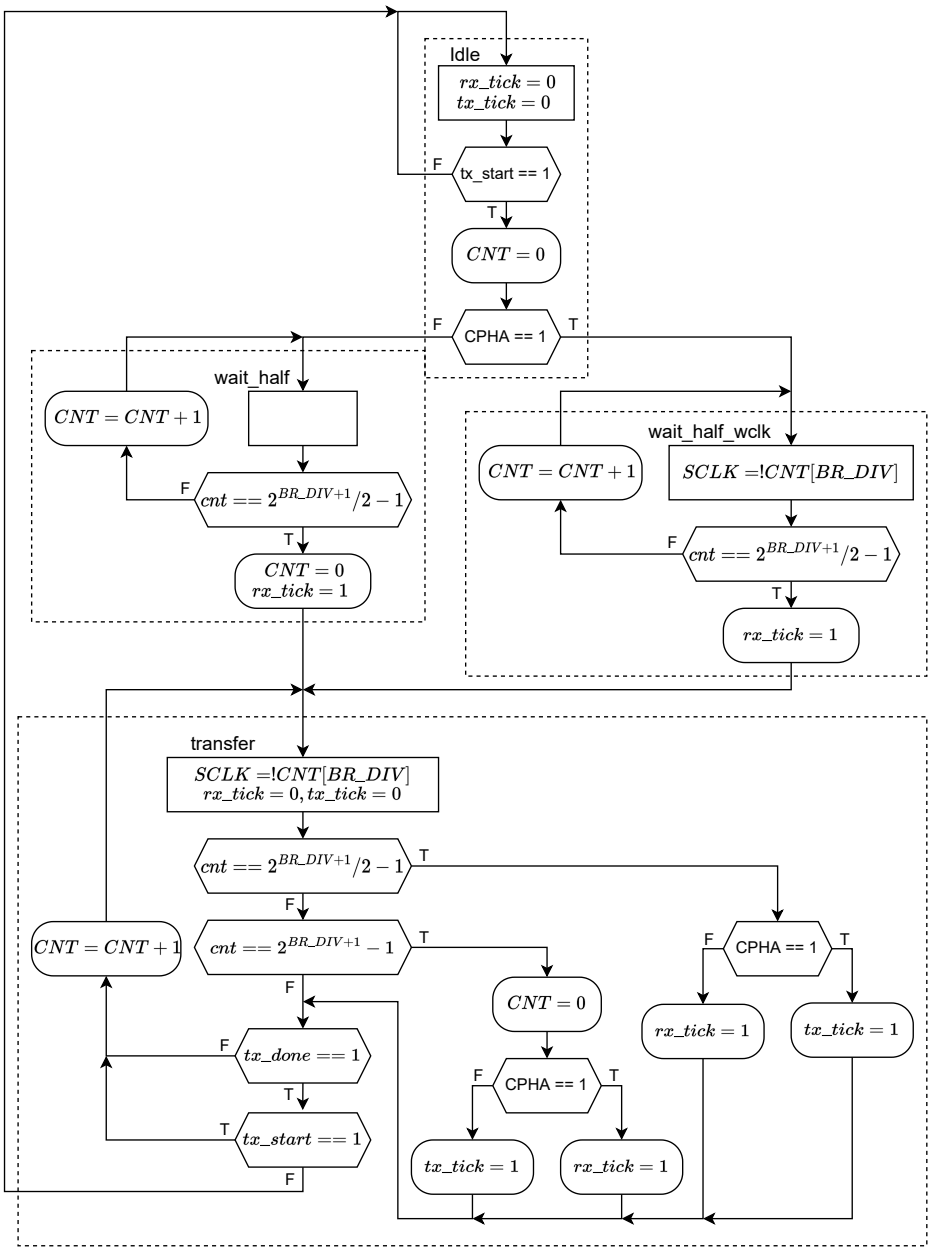


Figure 7.20: Shows the ASM chart for the baud rate generator in the SPI master.

7.8.2 UVM Verification

The verification of the SPI master follows the same procedure as the verification of the UART controller. Doing development it has been under constant simple verification using the PicoRV32 and C++, to ensure that basic functionality is achieved. Following this basic verification, a more extensive verification is set up using the UVM environment. For the UVM environment, an agent consisting

of a driver, sequencer and monitor, a scoreboard with reference module, and test sequences has been developed.

The UVM driver is designed to produce the signals expected from a connected slave, so it only generates the MISO signal. It does this by shifting out individual bits starting with MSB. The shifting occurs at a rising edge of the system clock. Which specific rising edge is calculated based on the data frame size, baud rate, CPOL, and CPHA. It was a priority to make the shifting independent of the SCLK signal, as using this signal to do the shifting would mean that an error on SCLK is more likely to go undiscovered. The monitor samples the MOSI signal, again, based on the system clock, data frame size, baud rate, CPHA, and CPOL being independent of the SCLK signal again. The sequencer used is the basic version supplied with the UVM environment.

The scoreboard samples the signals sent using the APB interface or the SPI interface. These signals are then compared to the reference module and if they are different an error is reported. The scoreboard has two coverage groups, an APB coverage group, and an SPI coverage group. The APB group covers the read and write operations performed through the APB interface. The SPI group covers the status of CPOL, CPHA, the data frame size, and the baud rate used. A cross-coverage group is made using these four signals, such that it can verify at what CPOL, CPHA, and data frame size a specific baud rate has been tested. The reference module consists of the internal registers of the SPI master based on the RDL file for the SPI master. Whenever a change to the registers is made, this is reported to the driver and monitor such that they can change settings as well to match what the DUT is doing.

The test sequences of the SPI master consist of writing data through the APB interface and the SPI slave interface. Random data is sent to the driver at all times using a UVM item. Every time an item is sent, the test sequence waits for this item to complete before a new item is created. This ensures that the slave will always respond with a known item when a transfer is initiated from the DUT. The main test sequence is through the APB interface. It will start the test by setting a baud rate. Then it will write multiple random messages to the MOSI FIFO and start transmission. Multiple messages are written to test the SPI master's capabilities for continuous transmission. When this is done it changes CPOL repeats. This is again repeated for CPHA and the 4 possible data frame sizes. When it has tested a specific baud rate, it moves on to the next baud rate and repeats.

The resulting coverage report generated by using the UVM environment for verification of the SPI master can be seen in appendix **??**. After verification

of the basic HDL code, the triplicated version of the SPI master was designed and tested again. An identical coverage report was generated.

7.9 UART Controller

In this section, a UART (Universal Asynchronous Receiver/Transmitter) controller will be developed. The UART controller will be connected to the CPU as a peripheral interface on the APB bus. The development of the UART controller is inspired by "FPGA Prototyping by Verilog Examples" [26].

A UART controller is asynchronous due to the fact that no clock signal is transmitted. Therefore it only makes use of 1 input and 1 output wire, rx and tx respectively. To achieve a successful transmission the protocol makes use of a common symbol frequency, commonly referred to as baud rate, and start, data and stop bits of predefined length. A parity bit can also be included for error detection. The common ranges for these values can be seen in figure 7.21. In this protocol the idle is logic level high, meaning that the start bit is created by pulling the signal line low, indicating that a transmission is starting. The LSB is transmitted first in the UART protocol. For this controller, it is decided to use a data frame size of 8 bits, no parity bit, and 1 stop bit. The parity bit is not utilized, as the rx and tx lines will be fully triplicated. To allow flexibility in the communication, the baud rate should be changeable on the fly, i.e. while the system is running.

Start bit (1 bit)	Data Frame (5 to 9 data bits)	Parity bit (0 to 1 bit)	Stop bit (1 to 2 bits)

Figure 7.21: Shows the common ranges for the length of different sections of the UART protocol [31].

The UART controller itself consists of 5 elements: A baud rate generator, a transmitter module, a receiver module, and two first-in, first-out (FIFO) buffers. The interconnection between these elements can be seen in figure 7.22.

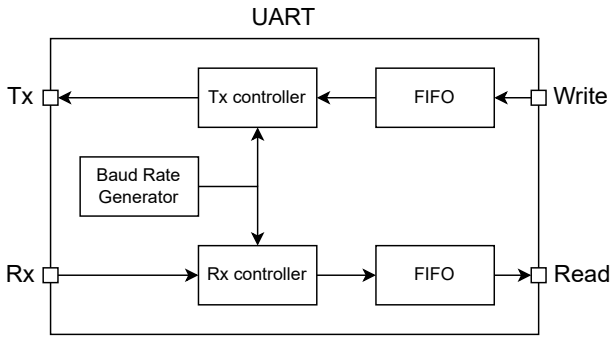


Figure 7.22: Block diagram of the UART components and their connections.

To control the system, programmable control register will be used and made accessible from the APB interface. The specific control registers have been chosen based on the components used in the controller and the common use case. As this is a fairly simple system in itself, only 3 programmable registers and a status register are implemented. These consist of writing and reading from the FIFO buffers in the system, setting the baud rate, and reading the status of the aforementioned FIFO buffers. This implementation of the control registers and their exact layout are summarised in the table 7.2.

Table 7.2: Shows the programmable register layout available to the APB interface for the UART controller. Addresses are given in hexadecimal.

Field name	**Address**	**Description**
status	0x00	Read only: Holds the status values for the rx and tx FIFOs and an additional busy signal
read_rx_data	0x04	Read only: reads the values currently in the rx FIFO. Returns an error if the FIFO is empty
write_tx_data	0x08	Write Only: Writes data to the tx FIFO, which begins transmission. Returns error if FIFO is full.
set_baudrate	0x12	Write and read: Sets the baud rate of the UART or reads the currently used baud rate.

The higher-level architecture has now been described and the next step is to begin the HDL implementation of each of the functional units.

7.9.1 Baud Rate Generator

The baud rate generator is used to generate the ticks for the rx/tx controller. This will be done by taking the system clock and dividing it by a specific amount

to reach the specified baud rate. This division is done using a mod-m counter and the m value can be calculated as seen in equation 7.1.

$$m = \frac{clk}{os \cdot baudrate} \tag{7.1}$$

Where:

m is the value which the counter should reset at	
clk is the frequency of the system clock	$[Hz]$
os is the oversampling rate	
$baudrate$ is the specified baud rate	$[\frac{bit}{s}]$

As the baud rate is programmable, a list of 8 possible baud rates has been chosen, which the baud rate generator is capable of generating. These baud rates are chosen on the premise of commonly used baud rates with a bias towards higher baud rates. They are as follows: 4800, 9600, 19200, 38400, 57600, 115200, 128000, and 256000. The choice is arbitrary and can always be changed by modifying a simple list in the Verilog source file. These baud rates, together with the clock frequency and oversampling rate, can be used to calculate the values for when to reset the counter (m). These are calculated at compile time using equation 7.1 and saved in an array. From these m values, the needed bit width for the counter and all other relevant signals in the baud rate generator. The baud rate can then be changed by programming the corresponding register to a different binary value. The m value will then be updated and the counter will be reset. Resetting the counter eliminates the risk of changing m to a lower value than the current count. This would result in the counter overflowing before normal operation would resume. This does not eliminate the risk of the baud rate being changed in the middle of a transmission. To eliminate this, the baud rate will not be changed while a busy signal is high. This busy signal will come from the rx and tx controllers and will indicate whether they are idle or not.

From this description, a functional diagram has been derived which can be seen in figure 7.23. This will be used as a baseline for the behavioral modeling in Verilog.

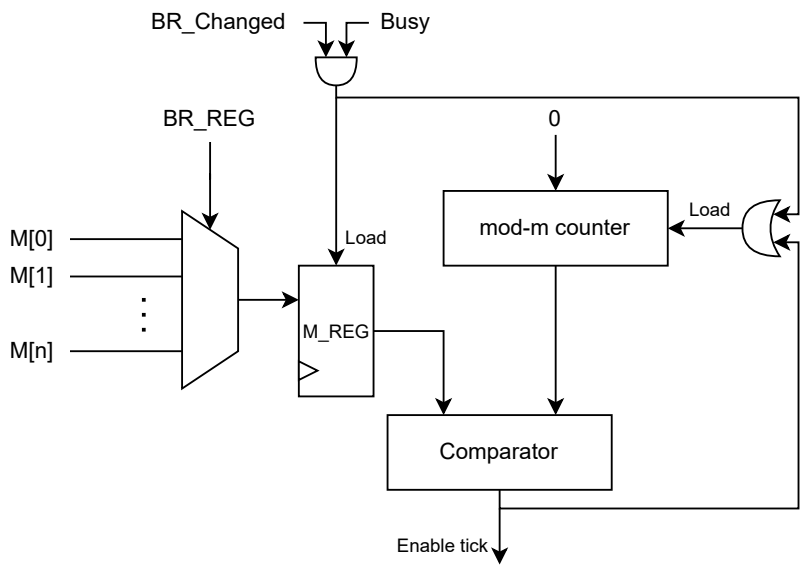


Figure 7.23: Shows a functional diagram for the baud rate generator. This is used to highlight the purposed functionality in an easy-to-understand way and therefore might not be the optimal solution.

7.9.2 First In First Out buffer

The first in first out (FIFO) buffer will for storing the received words and the words to transmit. This allows for a far more flexible usage of the UART, by not required to read the received words immediately and by allowing multiple words to be written to the UART from the CPU. The implementation will be based on a circular queue. This implementation illustrates the operation by taking a range of memory elements and forming them into a single. Two pointers will then be created, one for read operation and one for write operation. When one of these operations is performed the corresponding pointer will increase by one. This continues until the last memory element is reached, when an operation is then performed the pointer will reset back to 0 and start over again as in a circle. From the positions of the pointers, it is possible to derive the status of the FIFO, whether it is full or empty. It is also possible to rule out the illegal operation using the derived status, as the FIFO cannot be read from when empty and cannot be written to when full.

The FIFO will be parameterized using two variables. These will be word length, simply how large each memory element is, and the number of address bits, which will control how many memory elements are in the FIFO buffer. The number of address bits will be raised to the power of 2 to create a length

that corresponds to a whole number of bits. This ensures that the pointers will overflow by themselves without the need for a checker.

The FIFO can be described as a state machine with the states being given by the read and write pointers. The next state logic will be given by the control signals: read and write. These control what operations will be performed.

7.9.3 Tx/Rx Controller

The receiving and transmitting controllers are similar in operation and will therefore be presented together. These controllers will be parameterized using the chosen configuration of the UART protocol. Since parity will not be implemented, the parameters are the number of data bits, the number of stop bits, and oversampling rate.

4 states can be identified for both controllers. These correspond to the sections of the protocol shown in figure 7.21 in addition to an idle state. The states are then called: idle, start, data and stop. The idle contains a simple wait routine. For the transmitter this wait routine is broken when the tx FIFO buffer is not empty, communicated to the controller using a signal called tx_start. The next state is "start", where the transmitter will send a start bit in the form of a binary 0. This will last for an amount of enable ticks received from the baud rate generator corresponding to the oversampling rate, i.e. if the oversampling rate is 16, the controller will count 16 enable ticks from the baud rate generator before moving to the next state. In the next state, the data is sent, with LSB being sent first. Each data bit has the same duration as the transmitted start bit. This is done the number of times specified by the number of data bits given as a parameter. The last state is the stop state, where a 1 is transmitted for the duration specified by the number of stop bits. Each bit has the length specified by the oversampling rate. This operation is encapsulated in the ASM chart shown in figure 7.24.

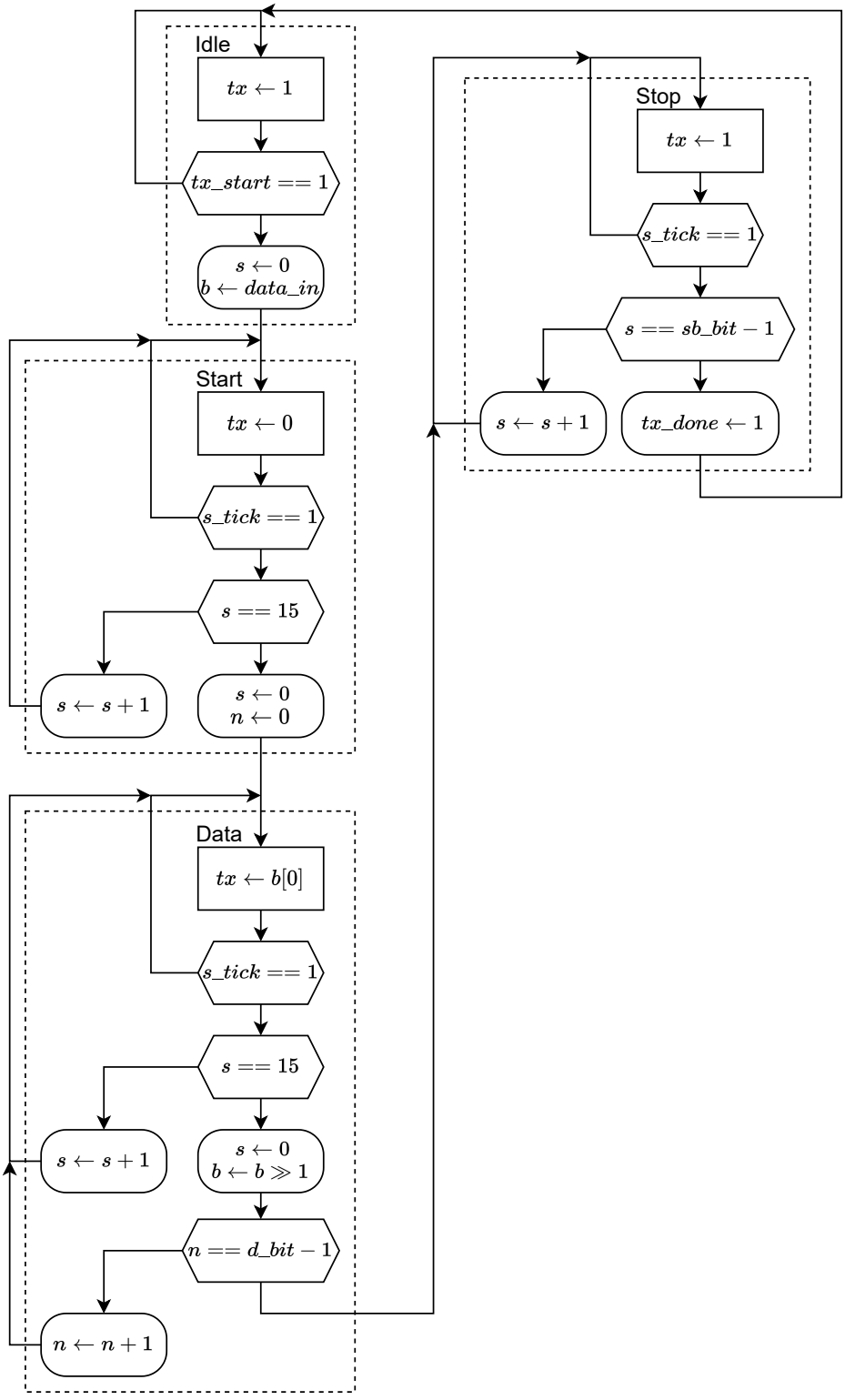


Figure 7.24: ASM chart for a UART transmitter without parity bit.

For the receiver, the ASM is nearly identical and will therefore not be shown. However, the differences will be highlighted here. For the rx controller, the idle state will be broken when the rx line goes low, corresponding to receiving the start bit. The start state will only count half of the enabled ticks specified by the oversampling rate. This is done such that the sampling of the rx line will be done in the middle of the data bit duration, ensuring that hold and setup times will not be violated.

7.9.4 Verification of UART Controller Design

The verification will be done in multiple steps. Critical functionality needed for basic operation will be tested using a Verilog testbench already developed which uses the PicoRV32 (CPU core). When basic functionality is ensured, a more elaborate testing environment will be designed using UVM.

By using the PicoRV32 for verification of basic functionality, the task is made easier by allowing for the use of the APB bridge attached to the PicoRV32 and allowing for the use of the c++ programming language. This makes writing the APB interface of the UART controller a one-line operation, while if one had used a Verilog testbench, this would require many lines of code. This does however rely on the correct operation of the CPU and the APB interface, but as these have both been verified before, this is not an obstacle. To provide stimuli to the output and monitor the output of the UART controller, a copy of the UART has been connected to the tx and rx lines. This copy of the UART does not make use of the APB interface as this is just an object used for testing purposes. This simplifies the production of the stimuli and monitoring of the outputs, as it can observe whether data is received correctly and what stimuli are applied, which is used for comparison. One has to be aware of the fact, that this test setup uses the UART itself to test whether the protocol is executed correctly, so errors can go undetected if they do not cause fatal errors in the operation. However, within the scope of basic functionality testing, this is deemed acceptable as this will be tested by the UVM environment. The c++ testbench is seen in code 7.1, while the relevant part of the Verilog testbench containing the test UART is seen in code 7.2.

```
1 #include <cstdint>
2
3 //Base address for the UART on the APB bridge
4 #define UART_BASE_ADDR 0x0A0C0000
5
6 int main(){
7     volatile uint32_t *uart_addr = (uint32_t *)
    UART_BASE_ADDR;
```

```
    uint32_t status_reg;
    uint32_t read_reg;
    *(uart_addr+3) = 0x05;
    *(uart_addr+2) = 0xDA;
    status_reg = *(uart_addr);
    *(uart_addr+2) = status_reg;
    while (status_reg == (int)2){
        status_reg = *(uart_addr);
    };
    *(uart_addr+2) = status_reg;
    read_reg = *(uart_addr+1);
    *(uart_addr+2) = read_reg;
    return 0;
}
```

Listing 7.1: C++ code for basic functionality testing of the UART controller

```
//uart test setup
reg rd_uart, wr_uart;
reg [7:0] tx_data;
reg [2:0] BR_REG;
wire rx_empty, rx_full, tx_full;
wire [7:0] rx_data;
uart_top test_uart (
  .clk(clk), .reset(!resetn),
  .rd_uart(rd_uart), .wr_uart(wr_uart),
  .tx_data(tx_data), .rx(uart_tx),
  .rx_empty(rx_empty), .rx_full(rx_full),
  .tx_full(tx_full), .rx_data(rx_data),
  .tx(uart_rx), .BR_REG(BR_REG)
);

initial begin
  rd_uart <= 1'b0;
  wr_uart <= 1'b0;
  BR_REG <= 3'b101;
  repeat(1000) @(posedge clk);
  tx_data <= 8'h4B;
  @(posedge clk);
  wr_uart <= 1'b1;
  @(posedge clk);
  wr_uart <= 1'b0;
end
```

Listing 7.2: Verilog code for basic functionality testing of the UART controller

With basic functionality verified, elaborate testing based on UVM will begin development. This development consists of designing the relevant agents with the corresponding monitor, sequencer, and driver, a UART package for sending

information between components, a model of the interface, a reference module, and a scoreboard with functional coverage. The UART will need two agents to connect to the test environment, an APB agent and a UART agent. The APB agent will monitor and provide stimuli for the APB interface, while the UART agent will monitor the tx line and provide stimuli on the rx line. Since an APB agent has been developed before, this can simply be reused without any modifications. It only needs to be instantiated and connected to the APB interface of the UART. This highlights some of the advantages of using UVM as this facilitates the reuse of components with its hierarchical structure.

It is worth mentioning here, that it is a goal to create a test environment that is distanced from the implementation in Verilog, i.e. by using a high-level description of certain parameters used in the functionality of the UART. If the test environment is designed using a low-level language approach similar to the use of Verilog, one has a higher risk of designing the same bugs and mistakes as in the Verilog code, especially when both design and verification are done by the same individual. Each of the mentioned components will now be designed individually, starting with infrastructure-related items and then from the bottom of the hierarchical structure.

UART Interface

The interface is extremely simple and only consists of the input wire rx and the output wire tx from the UART. There is no need to define an APB interface as this has already been done before in the development of the APB agent.

UART Item

The UART item is a class object, which will be used for packaging the relevant data for a UART transaction and sending it between UVM components. This item will have 3 variables: the data which has been sent, the type of transaction (received or transmitted), and whether the package is valid.

UART Sequencer

The UART sequencer is a part of the agent required for making it active, i.e. capable of driving stimuli to the device under test (DUT). In this test setup, the basic UVM sequencer will be used and therefore no development will be required. The sequencer becomes relevant when multiple sequences are sending packages to the same agent. In this case, it is the task of the sequencer to decide which packages have priority and will be sent to the driver first and in general what the order of packages will be. However, in this simple test environment

for the UART, it is not a plan to have multiple parallel running sequences and therefore no special scheduling scheme is necessary.

UART Driver

The task of the driver is to request an item from the sequencer, which will be a class object of the UART item and convert the parallel data into serial data by transmitting it using the UART protocol. This requires that it uses variables for the data length, stop length, parity bit, and the baud rate. All of these are required to be configured to match the configurations of the test UART. Even though parity is not implemented in the designed UART, it is included here, such that the UART has greater reusability in other UVM test environments. The oversampling rate is not a necessary variable here as the driver will not base the baud duration on a clock frequency. Instead, a high-level approach will be used, by simply calculating the baud duration as $1/baudrate$. This ensures a difference in the implementations but does result in the UVM environment not being cycle accurate to the DUT. This is due to the fact, that when the baud rate is used to calculate the count at which to reset the baud rate generator, it is rounded down or up as it can't count to fractional values. It does not result in critical differences in the baud duration, but it does mean the DUT and the test environment might differ for a few clock cycles after transmitting a message.

UART Monitor

The monitor operates nearly identically to the driver. Instead of driving signals to rx line of the DUT, it will sample the tx line. Identically to the Verilog implementation, the monitor will wait a half baud duration, when a start bit is encountered. Following that, it will wait for the full baud duration between sampling. This is done to sample in the middle of a baud. Due to the exact same reason as in the driver, the monitor will have slight differences in when a message is received and might not be accurate compared to the DUT in a few clock cycles following a transmission.

UART Agent

The UART agent is the container of the sequencer, driver, and monitor. One thing worth mentioning is that the agent will also have the variables for baud rate, data length, stop length, and parity. The agent will then pass these down to the components within the agent. This is done such the components of the agent can be configured by only configuring the agent.

Scoreboard and Reference Module

The scoreboard contains the reference module, the functional coverage, and the checking mechanisms to compare the reference module and the DUT. The scoreboard will contain subscribers for the APB and the UART interface. That is to say, it will receive the packages sent using both interfaces. These packages will then be used to update the reference module or check if the reference module and DUT are identical. Each time a package is received, it will sample the functional coverage, which will add one to the correct bin corresponding to the action performed. First, the functional coverage will be discussed and then the reference module and checking mechanisms.

The functional coverage, as the name implies, is a metric used to show how well the DUT has been tested on its different functionalities. This can be combined with code coverage to form a metric for how extensively the DUT has been tested. The functional coverage will be subdivided into two categories: APB and UART, corresponding to the two interfaces on the DUT. The UART coverage group will only be characterized by received messages and transmitted messages. The APB coverage will be divided into 3 categories: read, write and errors. The read-and-write category will contain all the programmable control registers of the DUT. The status registers have been divided further into all the possible combinations of status signals (Excluding the busy signal). As some of them are illegal, they have been marked this way, meaning that the success criteria become avoiding that state. The reading of the received messages (RX FIFO) together with the writing of messages to transmit (tx FIFO) has been divided into all the possible messages to send using 8 bits. This makes it possible to track what message has been sent. The error category is used to test whether correct errors are received when trying to write to a full tx FIFO or reading from an empty RX FIFO.

Next, is the design of the reference module. This reference module is used for comparison when testing the DUT. As the transmitting and receiving of the message are handled by the UART agent, the only things needed to be modelled by the reference module are the registers inside the DUT. The FIFO can be modelled by a std::queue in C++. Every time a message is received from the UART or APB agent or an APB interface operation is performed, the contents of these queues will be updated together with the status signals empty and full. The length of these queues will be equal to that of the FIFOs in the DUT. The busy status is excluded here as modelling this signal, would require a higher level of communication between the agent and the scoreboard. The development time of this system is not deemed necessary as the busy signal is not a critical feature. The rest of the registers can be modelled as simple variables.

The checking mechanisms are performed every time an APB read operation is performed or a message is received on the UART monitor. Every time a message is received on the UART monitor, the first item of the queue modelling the tx FIFO is popped (read and removed) and compared to the received message and the status variables are updated. When an APB read operation is performed, the comparison depends on which register was read. In the case of reading the rx FIFO, an operation equivalent to that of receiving a message on the UART monitor is performed on the respective queue object. When reading the status or baud rate register (BR_REG), the corresponding variable is used for comparison. The error signals are also checked doing APB read operations as these are subpart of the APB interface. If an error occurs, meaning that the DUT and reference module are not equivalent, it is reported to the user.

7.9.5 Test sequences

The UVM environment has now been designed and the development of test sequences (Stimuli for the DUT) can begin. The tests will be split into 3 categories: status test, error test and UART test. The status test will ensure that all states of the FIFOs are reached to ensure correct operation. The error test will ensure that the error signals are received from the APB interface at the correct times, i.e. reading from an empty FIFO and writing to a full FIFO. The UART test will write to tx FIFO using the APB interface and transmit a message using the UART driver using random data. The resulting coverage report from using these test sequences can be seen in appendix **??**.

7.10 Triplication

Triplication of the UART was done using the TMRG tool. For this tool to function all registers in the Verilog code would need altering. For each register, a wire was added named the same as the register, but with an added "Voted" to the end of it. The voted wire was then used everywhere the original register is used as an input. This allows the TMRG tool to triplicate those registers.

Doing triplication it was discovered that the TMRG tool does not allow for the Verilog function structure. This made the baud rate changing feature fall apart, as a function was used to calculate the count at which the counter should reset. Another approach was needed. Instead, Python was used to generate the mod-counter, the uart_top, and the apb_uart file based on a given set of baud rates. This is a lot more flexible than the original approach but does make the changing of the source code harder, as this now needs to be changed inside the

python template. The Python file uses two templates: one for a single baud rate and one for multiple baud rates. This python file is integrated into the compilation of the entire project, such that every time the project is compiled, the python file is used to generate the mod-m counter given a range of defined inputs.

Both the triplicated version with a single baud rate and one with multiple baud rates are tested using the UVM environment. The exact same coverage report is achieved in these tests and can be seen in appendix **??**.

7.11 Physical Implementation

The last step of development for the UART is the physical implementation. This follows the steps described in section 7.5. The die size is $110\ \mu\text{m}$ by $110\ \mu\text{m}$ including power rings. The I/O connections have been spaced equally around all four sides of the die. Dynamic power analysis using a testbench has been performed and IR drops have been analyzed. The overview of the results of this process can be seen in figure 7.26. A figure of the IR drop can be seen in 7.25. The design violates no hold or setup timings at $325\ \text{MHz}$, it has $88.35\ \%$ utilization of area, and uses $0.61\ \text{mW}$.

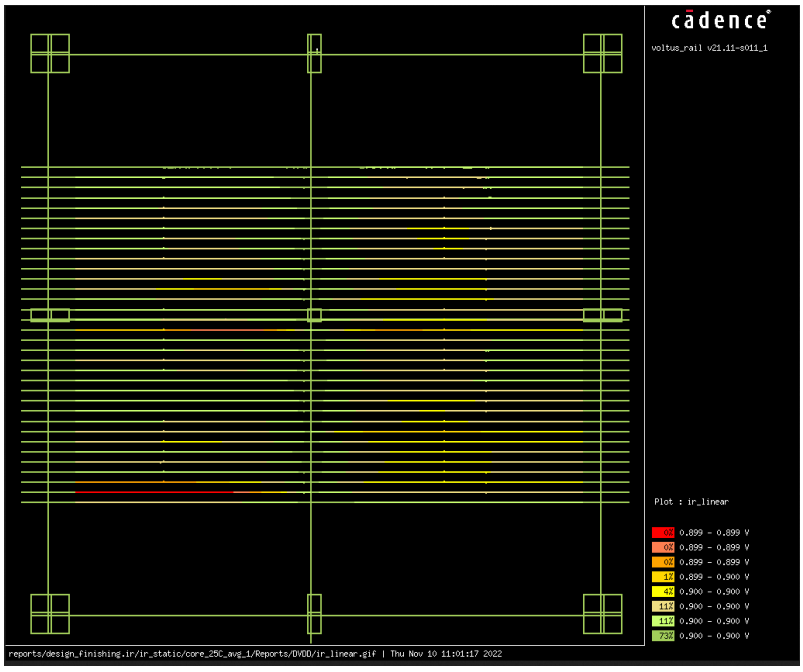


Figure 7.25: Shows the IR drop for the physical implementation of the UART controller. Even though some areas are red, the percentage drop is minimal.

▸ Filter Metrics

Snapshots	Setup (all)			Setup (reg2reg)			Hold (all)			Hold (reg2reg)			DRV			Clock		Design		DFT	Power				Congestion	
	WNS (ns)	TNS (ns)	FEPS	WNS (ns)	TNS (ns)	FEPS	WNS (ns)	TNS (ns)	FEPS	WNS (ns)	TNS (ns)	FEPS	Tran (ns)	Load (pF)	Fanout	Insts	Area (um^2)	Density (%)	Insts	Area (um^2)	Passing	Total (mW)	Leakage (mW)	Internal (mW)	Switching (mW)	Clock (mW)	Max	Total
syn_generic	1.839	0	0	2.622	0	0							0	0	0			0.00	5055	8281		0.68	0.03	0.62	0.04			
syn_map	0.208	0	0	0.299	0	0							12528	0	80			0.00	2833	2274		1.09	0.00	0.67	0.42			
syn_opt	0.012	0	0	0.012	0	0							0	0	80			0.00	2833	2273		1.07	0.00	0.66	0.41			
floorplan																		65.91	2833	2273								
place	0.019	0	0	0.088	0	0							-0	0	-354			65.81	2869	2270		0.74	0.00	0.54	0.20		0.00	0.00
cts	0.208	0	0	0.222	0	0	-0.001	-9	18	-0.001	-9	18	-0	-6	0	15	24.066	67.06	2959	2313		0.92	0.00	0.62	0.30	0.17	0.00	0.00
postcts	0.206	0	0	0.206	0	0	0.200	0	0	0.200	0	0	-0	-6	0	15	24.066	86.67	3698	2989		1.12	0.00	0.76	0.36	0.17	0.00	0.00
route	-0.129	-1	11	-0.129	-1	11	0.180	0	0	0.180	0	0	-0	-6	0	15	24.066	86.67	3698	2989		1.16	0.00	0.76	0.40	0.18	51.00	51.00
postroute	0.112	0	0	0.112	0	0	0.118	0	0	0.118	0	0	-0	-2	0	15	24.066	88.35	3834	3047		1.17	0.00	0.76	0.41	0.18	6.00	15.00
design_finishing	0.112	0	0	0.112	0	0	0.118	0	0	0.118	0	0	-0	-2	0	15	24.066	88.35	3834	3047		1.17	0.00	0.76	0.41	0.18	6.00	15.00
signoff (design_finishing)	0.146	0	0	0.186	0	0	0.146	0	0	0.146	0	0	-0	-2								1.16	0.00	0.76	0.40	0.18		
ir_dynamic																						0.61	0.00	0.47	0.14	0.18		

Figure 7.26: Shows the table of results relevant to the physical implementation.

Chapter 8

Conclusion

In conclusion, this PhD thesis may have not made significant and practical contributions to the field, but by addressing critical gaps in knowledge, developing innovative methodologies, and providing valuable insights paved the way for further research. The Field of Automatic Certification and safety assessment is rather open field, joinin the specialized conferences and symposiums it is clear that the community it is, yes willing, but not read to yield the responsability of such and analyis to an automated method. Nevertheless the rigorous theoretical framework presented in the thesis, which integrates and builds upon previous literature, serves as a solid foundation for the empirical analysis future PhD will continue, under different spotlights to enrich this research, until finally it will be the state of the art for the relibility assessment of electronic components. In any case, the extensive data collection, meticulous experimental design, and robust validation of the proposed models have allowed me to have a more comprehensive understanding of the subject matter, thus enhancing the existing body of knowledge and allowing a completlely different vision of the field.

The research conducted in this thesis has successfully adapted and extended the fault tree-based extraction of failure metrics, which has conventionally been applied to macroscopic electromechanical systems, to the realm of automotive SoCs. The current state-of-the-art in automotive SoCs involves largely manual metric extraction and expert dependence, whereas the verification of countermeasure effectiveness is typically performed through targeted fault injection on specific sub-parts of the system or irradiation under a particle beam.

The first part of the thesis involved an in-depth literature review of fault tree construction, the ISO26262 standard, and the derivation of different reliability metrics for digital SoC-type systems. The developed methodology extracts metrics at the block level using two different methods: an analytical method based on probabilities and an experimental method based on fault injection. The fo-

cus of this research was not to create new probability codes or fault injection tools, but rather to establish a methodology for employing these techniques in the context of an SoC to obtain the desired data.

The second part of the thesis addressed the composition of data obtained at the functional block level to derive the ISO26262 metrics at the system level (SoC). The developed composition method was tailored to the unique characteristics of SoCs, such as their communication systems, real-time calculations, and fault models imposed by the ISO26262 standard.

In the third part of the thesis, the developed methodology was applied to an SoC-type system, and the results were validated. This practical application of the research demonstrates the effectiveness of the proposed methodology in a real-world context, offering valuable insights and facilitating the ISO26262 certification process for automotive SoCs.

Through this research journey, the thesis has contributed to bridging the gap between traditional fault tree-based methods and their application to the automotive SoC domain. The findings have not only enhanced the existing literature but also provided practical implications for stakeholders, policymakers, and practitioners involved in the development and certification of automotive SoCs.

As with any research endeavor, this thesis has also identified areas for future exploration and improvement. Future researchers can build upon the foundations laid in this work to further refine the proposed methodology, expand its applicability to other domains, and develop innovative techniques for extracting and analyzing reliability metrics in complex systems.

In conclusion, this thesis marks a significant milestone in the field of automotive SoC reliability and ISO26262 certification. The rigorous theoretical framework, extensive empirical analysis, and robust validation of the proposed models have provided a comprehensive understanding of the subject matter, paving the way for further research and advancements. The interdisciplinary nature of the study has fostered collaborations and synergies among different domains, encouraging a more holistic approach to problem-solving. As the field continues to evolve, the knowledge generated by this research will serve as a stepping stone for further innovations and the development of more efficient and reliable automotive systems.

Throughout this research journey, several challenges were encountered and overcome, leading to a more profound appreciation for the complexities inherent in the chosen field. The interdisciplinary nature of the study has fostered collaborations and synergies among different domains, encouraging a more holistic approach to problem-solving. By embracing a multidisciplinary per-

spective, this thesis has successfully transcended the traditional boundaries of the field and provided innovative solutions to complex issues.

The findings of this research have also offered practical implications for stakeholders, policymakers, and practitioners. The policy recommendations and best practices derived from the results can be applied to various real-world contexts, contributing to more effective decision-making and the development of more sustainable and equitable systems. Moreover, the thesis has highlighted the importance of continuous dialogue between academia and industry, emphasizing the need for a collaborative approach to address the multifaceted challenges faced by our society.

In addition to the immediate contributions, this thesis has also identified several areas for future research. The findings presented here open up new avenues for exploration, inviting future researchers to delve deeper into the subject matter and build upon the foundations laid in this work. As the field continues to evolve, it is expected that the knowledge generated by this research will serve as a stepping stone for further innovations and advancements, leading to the discovery of new solutions and a better understanding of the complexities of our world.

Furthermore, the methodological advancements and novel approaches proposed in this thesis offer exciting opportunities for researchers in other domains. The cross-disciplinary applicability of these methods can be leveraged to tackle challenges in diverse areas, ultimately contributing to the growth of knowledge across fields.

Throughout the research process, the importance of ethical considerations and the impact of the study on society have been acknowledged and addressed. This thesis has demonstrated that the pursuit of knowledge must be accompanied by a commitment to responsible research practices, ensuring that the generated knowledge contributes positively to the well-being of all stakeholders involved.

This thesis is a testament to the power of collaboration, perseverance, and the relentless pursuit of knowledge in the service of a better understanding of our world. As I embark on the next chapter of my academic journey, I carry with me the lessons learned, the skills acquired, and the passion for discovery that has been nurtured throughout the course of my doctoral studies. I am committed to using my research as a catalyst for positive change and to continuing my lifelong pursuit of knowledge in the service of society.

Bibliography

[1] accellera. *UVM 1.2 User Guide*. 2015. URL: https://www.accellera.org/downloads/standards/uvm (visited on 10/11/2022).

[2] accellera. *Universal Verification Methodology (UVM) Working Group*. URL: https://www.accellera.org/activities/working-groups/uvm/ (visited on 10/11/2022).

[3] Z. S. Andraus and K. A. Sakallah. "Automatic abstraction and verification of verilog models". In: *Proceedings. 41st Design Automation Conference, 2004.* 2004, pp. 218–223. DOI: 10.1145/996566.996629.

[4] O. Ariss, D. Xu, and W. E. Wong. "Integrating Safety Analysis With Functional Modeling". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 41 (2011), pp. 610–624.

[5] ARM. *AMBA APB Protocol Specification*. 2010. URL: https://developer.arm.com/documentation/ihi0024/c/Introduction/About-the-APB-protocol.

[6] André Arnold et al. "The AltaRica Formalism for Describing Concurrent Systems". In: *Fundam. Inf.* 40.2–3 (1999).

[7] R. A. Austin et al. "Automatic Fault Tree Generation from Radiation-Induced Fault Models". In: *2020 Annual Reliability and Maintainability Symposium (RAMS)*. 2020, pp. 1–7. DOI: 10.1109/RAMS48030.2020.9153630.

[8] David Barney. "CMS Detector Slice". CMS Collection. 2016. URL: https://cds.cern.ch/record/2120661.

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

[10] Ron Bell. "Introduction to IEC 61508". In: *Acm international conference proceeding series*. Vol. 162. Citeseer. 2006, pp. 3–12.

[11] 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.

[12] Giulio Borghello. “Ionizing Radiation Effects in Nanoscale CMOS Technologies Exposed to Ultra-High Doses”. 2018.

[13] Xavier de Bossoreille, Mathilde Machin, and Laurent Sagaspe. “Un Nouvel Outil de Safety pour Maitriser la Complexité des Systèmes”. In: *"Maîtrise des risques et transformation numérique: opportunités et menaces*. Reims, France, Oct. 2018. URL: https://hal.archives-ouvertes.fr/hal-02063631.

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

[15] 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.

[16] Marco Bozzano et al. “Statistical fault injection: Quantified error and confidence”. In: *Proceedings of the 10th International Workshop on Automated Verification of Critical Systems*. 2010.

[17] B. A. Brady et al. “ATLAS: Automatic Term-level abstraction of RTL designs”. In: *Eighth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2010)*. 2010, pp. 31–40. DOI: 10.1109/MEMCOD.2010.5558624.

[18] Robert Brayton et al. “VIS : A System for Verification and Synthesis”. In: (Aug. 2000).

[19] Robert K. Brayton and Alan Mishchenko. “ABC: An Academic Industrial-Strength Verification Tool”. In: *CAV*. 2010.

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

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

[22] Alessandro Caratelli. “Research and development of an intelligent particle tracker detector electronic system”. 2019.

[23] CMS CERN. “Detector: About CMS”. In: (). URL: https://cms.cern/detector (visited on 12/08/2022).

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

[25] 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.

[26] Pong P. Chu. *FPGA Prototyping by Verilog Examples Xilinx Spartan-3 Version*. John Wiley & Sons, INC., 2008.

[27] CMS collaboration et al. "The CMS experiment at the CERN LHC". In: *Jinst* 3 (2008), S08004.

[28] CMS tracker collaboration et al. *The Phase-2 Upgrade of the CMS Tracker*. Tech. rep. CERN-LHCC-2017-009. CMS-TDR-014. Geneva: CERN, June 2017. URL: https://cds.cern.ch/record/2272264.

[29] Kahneman Daniel. *Thinking, fast and slow*. 2017.

[30] Keerthikumara Devarajegowda et al. "A Mutually-Exclusive Deployment of Formal and Simulation Techniques Using Proof-Core Analysis". In: Oct. 2017.

[31] Analog Devices. *UART: A Hardware Communication Protocol*. 2020. URL: https://www.analog.com/en/analog-dialogue/articles/uart-a-hardware-communication-protocol.html (visited on 09/16/2022).

[32] Giorgio Di Natale et al. *Cross-Layer Reliability of Computing Systems*. iet - the institution of engineering and technology, Jan. 2020, pp. 1–328. DOI: 10.1049/PBCS057E. URL: https://hal.science/hal-02986877.

[33] 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.

[34] 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.

[35] E. Allen Emerson. "Temporal and Modal Logic". In: *Handbook of Theoretical Computer Science*. Vol. B: Formal Models and Sematics. 1990.

[36] Lydon Evans and Philip Bryant. "LHC Machine". In: (2008). LHC description. URL: https://iopscience.iop.org/article/10.1088/1748-0221/3/08/S08001/pdf.

[37] 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.

[38] Tiziano Fiorucci et al. "Software Product Reliability Based on Basic-Block Metrics Recomposition". In: *IEEE International Symposium on On-Line Testing* (2023, To appear in).

[39] Barbara Gallina et al. "Modeling a Safety- and Automotive-Oriented Process Line to Enable Reuse and Flexible Process Derivation". In: *2014 IEEE 38th International Computer Software and Applications Conference Workshops*. 2014, pp. 504–509. DOI: 10.1109/COMPSACW.2014.84.

[40] 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.

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

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

[43] Julie Haffner. "The CERN accelerator complex. Complexe des accélérateurs du CERN". In: (2013). General Photo. URL: https://cds.cern.ch/record/1621894.

[44] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[45] 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.

[46] Y. Ho, A. Mishchenko, and R. Brayton. “Property directed reachability with word-level abstraction”. In: *2017 Formal Methods in Computer Aided Design (FMCAD)*. 2017, pp. 132–139. DOI: 10.23919/FMCAD.2017.8102251.

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

[48] ISO. *ISO 26262–1:2011 - Road vehicles - Functional safety - Part 1: Vocabulary.*

[49] ISO. *ISO 26262–2:2011 - Road vehicles - Functional safety - Part 2: Management of functional safety.*

[50] ISO. *ISO 26262–3:2011 - Road vehicles - Functional safety - Part 3: Concept phase.*

[51] ISO. *ISO 26262–4:2011 - Road vehicles - Functional safety - Part 4: Product development at the system level.*

[52] ISO. *ISO 26262–5:2011 - Road vehicles - Functional safety - Part 5: Product development at the hardware level.*

[53] ISO. *ISO 26262–6:2011 - Road vehicles - Functional safety - Part 6: Product development at the software level.*

[54] ISO. *ISO 26262–7:2011 - Road vehicles - Functional safety - Part 7: Production and operation.*

[55] ISO. *ISO 26262–8:2011 - Road vehicles - Functional safety - Part 8: Supporting processes.*

[56] ISO. *ISO 26262–9:2011 - Road vehicles - Functional safety - Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses.*

[57] ISO. *ISO 26262–10:2012 - Road vehicles - Functional safety - Part 10: Guideline on ISO 26262.*

[58] ISO26262. *Road vehicles – Functional safety*. 2018. URL: https://www.iso.org.

[59] ISO26262. *Acronyms and abbreviations*. URL: https://www.arteris.com/blog/top-35-iso-26262-acronyms-and-abbreviations/.

[60] Andreas Johnsen et al. "Risk-Based Decision-Making Fallacies: Why Present Functional Safety Standards are Not Enough". In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. 2017, pp. 153–160. DOI: 10.1109/ICSAW.2017.50.

[61] 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).

[62] Christof Kaukewitsch et al. "Automatic Generation of RAMS Analyses from Model-based Functional Descriptions using UML State Machines". In: *2020 Annual Reliability and Maintainability Symposium (RAMS)* (2020), pp. 1–6.

[63] 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.

[64] 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.

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

[66] Weng Fook Lee. *Learning from VLSI Design Experience*. Springer, 2019.

[67] 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.

[68] Mathilde Machin, Laurent Sagaspe, and Xavier de Bossoreille. *SimfiaNeo, Complex Systems, yet Simple Safety*. 2018.

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

[70] 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.

[71] J.J. Marin and R.W. Pollard. "Experience report on the FIDES reliability prediction method". In: *Annual Reliability and Maintainability Symposium, 2005. Proceedings.* 2005.

[72] 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.

[73] 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.

[74] Hala Mortada, Tatiana Prosvirnova, and Antoine Rauzy. "Safety Assessment of an Electrical System with AltaRica 3.0". In: *4th International Symposium on Model-Based Safety Assessment*. Munich, Germany, Oct. 2014.

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

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

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

[78] 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.

[79] 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.

[80] T. Prosvirnova. "AltaRica 3.0: Model-Based approach for Safety Analyses". Theses. Ecole Polytechnique, Nov. 2014. URL: https://pastel.archives-ouvertes.fr/tel-01119730.

[81] T. Prosvirnova. "AltaRica 3.0: a Model-Based approach for Safety Analyses". In: *Thèse de doctorat de l'ecole Polytechnique, 2014* ().

[82] Sergio Ramírez and Camilo Rocha. "Formal verification of safety properties for a cache coherence protocol". In: *2015 10th Computing Colombian Conference (10CCC)*. 2015.

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

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

[85] Alessandro Savino et al. "Statistical Reliability Estimation of Microprocessor-Based Systems". In: *IEEE Transactions on Computers* 61.11 (2012), pp. 1521–1534. DOI: 10.1109/TC.2011.188.

[86] S. A. Seshia and P. Subramanyan. "UCLID5: Integrating Modeling, Verification, Synthesis and Learning". In: *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*. 2018, pp. 1–10. DOI: 10.1109/MEMCOD.2018.8556946.

[87] STMicroelectronics. *RM0008 Reference Manual*. 2021. Chap. General-purpose timers (TIM2 to TIM5).

[88] STMicroelectronics. *RM0008 Reference Manual*. 2021. Chap. Serial Peripheral Interface (SPI).

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

[90] *Synopsys Z01x*. https://www.synopsys.com/verification/simulation/z01x-functional-safety.html.

[91] T. Prosvirnova and A. Rauzy. "The structural constructions of AltaRica 3.0". In: *Actes du congrès LambdaMu'19*. IMdR, Oct. 2014.

[92] Josep Torras Flaquer et al. "Handling reconvergent paths using conditional probabilities in combinatorial logic netlist reliability estimation". In: *2010 17th IEEE International Conference on Electronics, Circuits and Systems*. 2010.

[93] J. Torras-Flaquer. "Méthodes probabilistes pour l'estimation de la fiabilité dans la logique combinatoire". In: *Thèse de Doctorat de l'école doctorale TELECOM Paristech, 2011* ().

[94] 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).

[95] 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.

[96] A. Vallero et al. "Cross-layer system reliability assessment framework for hardware faults". In: *2016 IEEE International Test Conference (ITC)*. 2016, pp. 1–10. DOI: 10.1109/TEST.2016.7805863.

[97] 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.

[98] S. Venkataramanan, A. Kumari, and L. Piper. *System Verilog Assertions Handbook*. CreateSpace Independent Publishing Platform, 2015.

[99] 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.

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

[101] Jonas Westman and Mattias Nyberg. "A reference example on the specification of safety requirements using ISO 26262". In: *SAFECOMP 2013-Workshop DECS (ERCIM/EWICS Workshop on Dependable Embedded and Cyber-physical Systems) of the 32nd International Conference on Computer Safety, Reliability and Security*. 2013, NA.

[102] A. Witulski et al. "Development of a Flight-Program-Ready Radiation Model-Based Assurance Platform". In: *2020 IEEE Aerospace Conference*. 2020, pp. 1–8. DOI: 10.1109/AERO47225.2020.9172762.

[103] Nataliya Yakymets and Morayo Adedjouma. "Model-based Quantitative Fault Tree Analysis based on FIDES Reliability Prediction". In: *IEEE International Symposium on Software Reliability Engineering Workshops*. 2020.

[104] Ping Yeung, Doug Smith, and Abdelouahab Ayari. "Whose Fault Is It Formally? Formal Techniques for Optimizing ISO 26262 Fault Analysis". In: Feb. 2018.

[105] YosysHQ. *PicoRV32 Native Memory Interface*. URL: https://github.com/YosysHQ/picorv32#picorv32-native-memory-interface (visited on 12/02/2022).

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