Università degli Studi di Napoli Federico II

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Classe delle Lauree Magistrali in Ingegneria Elettronica, Classe n. LM-29

Corso di Laurea Magistrale in Ingegneria Elettronica

Thesis

FPGA - MCU design for the performance improvement of a space payload for radiation monitoring

Supervisor:
Ch.mo Prof. Pasquale Arpaia

Co-Supervisor:
Dr. Danzeca Salvatore

Candidate:
Zimmaro Alessandro
Matr. M61/499

Academic Year
2018/2019

Acknowledgments

First of all, I would like to express my deepest gratitude to Prof. Pasquale Arpaia, my University Supervisor, and to Dr. Salvatore Danzeca, my CERN Section Leader and Supervisor, for giving me this opportunity and for trusting in me. This experience was really important for me because, during these 6 months, I acquired further knowledge about the topics I studied during my University courses. I had the opportunity to find solutions to real problems and grow as an engineer thanks to their suggestions.

Many thanks also to my CERN Section colleagues.

Now, I switch to Italian to thank my family and my old friends.

Devo ringraziare la mia famiglia senza la quale tutto questo non sarebbe stato possibile: non solo per gli aiuti economici forniti, ma soprattuto per il loro supporto quando ne ho avuto bisogno. Ringrazio mio padre, fonte di ispirazione e miglioramento e mia madre, per il supporto e l'amore costante. Spero di essere riuscito a renderli fieri di me.

Ringrazio tutte le persone che mi sono state vicine in questi anni e che mi hanno sempre supportato, soprattutto nei periodi in cui alcuni esami o momenti della vita sembravano impossibili da superare

Un ringraziamento speciale voglio farlo ai miei migliori amici, Raffaele e Antonio, i quali sono rimasti sempre al mio fianco e sono certo che ci saranno sempre.

Abstract

Electronic devices working in a radiation environment are inevitably affected by radiation effects and for this reason, it is important to monitor them. The aim of this thesis is the development of the SpaceRadMonMx, an instrument able, thanks to the carefully selected sensors, to measure the Total Ionizing Dose (TID), the number of Single Event Upsets (SEU), and Single Event Latchups (SEL) that provide with sufficient information to characterize the space environment through the HEH fluence and the spectrum hardness factor. The SpaceRadMonMx system is based on the interface of an FPGA and a Microcontroller. The latter is used to acquire analog voltage from a RadFET through an ADC and send data acquired to the FPGA via SPI protocol. Instead, the FPGA manages the MCU and monitors SEL, SEU, and MBU events. At the same time, it transmits the data to the outside through the I2C protocol. The combination of these two devices allows the system to monitor radiation in the Space environment. In particular, in this thesis work, the firmware that allows the SpaceRadMonMx to manage the sensors and communicate data is described.

Introduction

In recent decades, the electronics world has been the object of strong growth and widespread thanks to which, the electronic devices have had a strong development in every type of application and with a wide range of costs. The COTS (commercial off-the-shelf) products can be used as an alternative to the ones developed internally by the company to reduce the costs of development and maintenance. Strong attention must be given to the work environments like space and particle accelerators where the radiation can lead the electronic devices to not work properly. If electronic devices must be used in a radioactive environment, the knowledge of radiation sources and effects is really important.

For the reasons expressed above, an introduction to radiation and different radiation environment is presented in the first part of the thesis. Particular attention is given to the CERN radiation environment where the thesis is developed and which is different from the Space one.

In a modern accelerator, electronic components and systems are exposed to a mixed radiation field which is made of photons, electrons, muons, charged and neutral hadrons with energies that can be very low (thermal)up to the TeV range. The electronic components object of this mixed field would be a place of Single Event Effects (SEEs), damage from Total Ionizing Dose (TID), and displacement damage (DD).

For this reason, these three effects will be all presented and described. Moreover, to mitigate SEEs, different techniques based on hardware and software exists and will be presented.

To monitor these radiation effects, the RadMon instrument was designed at CERN and used inside CERN facilities since 2013. The RadMon, thanks to the carefully selected sensors, measures the Total Ionizing Dose (TID), the number of Single Event Upsets (SEU), and Single Event Latchups (SEL) that provide sufficient information to characterize the space environment through the HEH fluence and the spectrum hardness factor.

The SpaceRadMonMx instrument is a CERN design based on the miniaturization of version 6 of the RadMon and the whose working system is based on interfacing between a microcontroller (MCU) and FPGA. For this reason, a state of art between interfacing these two devices will be presented in the second part.

Finally, the SpaceRadMonMx is presented and described in every single part, which is composed. In particular, firstly the SpaceRadMonMx detector system will be presented with all the

Verilog modules that are part of it. Then the microcontroller configuration and the module implemented in the FPGA that interacts with it, are described and subsequently, the flash freeze technology and its improvement on the system. Finally, the I2C communication and the implemented module that able the system to transmit the data to the outside are presented.

Part I
Background

Chapter 1

1 Radiation effects

The art of radiation effects, as defined into the Handbook of radiation effects (A. Holmes-Siedle) [1], is an interdisciplinary subject involving radiological physics, solid-state physics, and radiobiology. Radiology is the science of high-energy particles and photons and how they interact with matter

Three important discoveries let this science develop:

1) 1895: X-rays discovered by Rontgen
2) 1896: Radioactivity discovered by Becquerel
3) 1898: Radium discovered by Curies.

Only in the half of the 20th century, the radiation effects became an important topic due to the use of semiconductor technology in space and military fields. These two environments are dangerous for semiconductor devices because provide radiation exposure.

1.1 Radioactivity

Radioactivity is based on the fact that not all atomic nuclei are stable. The elements with an atomic number Z (the number of protons present in the nucleus of that specific element) between 1 and 92 exist naturally, while those from 93 to 106 are artificially produced. The elements with Z greater than 82 are radioactive because they modify their original structure by emitting sub atomic particles and gamma radiation to reach a stable configuration.

Negative beta particles are emitted by elements with high Z. The radioactivity is measured in Becquerel (Bq) and it is defined as the activity of a quantity of radioactive material in which one nucleus decays per second. A Bq is equivalent to disintegration in a second.

The decay rate of a nuclide can be defined in terms of its 'half-life' (1.1) that is the time necessary for the radioactivity to reach half of its original value.

$$t_{\frac{1}{2}} = \frac{\ln(2)}{\lambda} \qquad (1.1)$$

In equation (1.1) λ is the decay constant and depends on the particular nuclide.

1.2 Types of radiation

Ionizing radiation is radiation that carries sufficient energy to detach electrons from atoms or molecules, thus ionizing them.

It can be classified into 5 groups:

- Alpha radiation: consists of 2 protons and two neutrons bound together. It can travel only a few centimeters in the air and can be shielded with paper. It has an energy of about some MeV.
- Beta radiation: consists of electrons or positrons. It can travel more than Alpha particles and can be shielded with plexiglass or a few mm of aluminum. It has an energy that goes from a few keV up to a few MeV.
- Gamma radiation: consists of photons. It can travel up to a few hundred meters in the air and can be shielded with lead or concrete. It has an energy of about some MeV.
- X-ray radiation: consists of photons with a frequency lower than those of gamma radiation. The difference lies in the source of generation: it is produced in the electron cloud while gamma, is generated in the nucleus.
- Proton radiation: consists of protons (nuclei of hydrogenous atoms). It is difficult to deflect respect electrons because it has a mass 1800 times greater. It can travel for several centimeters in air and tens of micrometers in aluminum. This radiation is produced in the accelerator environment and it can be shielded with concrete.
- Neutron radiation: it consists of neutrons. The problem of a neutron is the charge: being neutral particles they are difficult to stop. Neutrons are classified according to their energy: thermal (<1 eV), intermediate, fast (>100 keV). The velocity of the neutrons can be reduced by hydrogenous material. Like protons, this radiation is really important because is produced by a particle accelerator. Moreover, this type of radiation is produced through the interaction between cosmic rays and the atmosphere and so, it exists at ground level. It can be shielded with concrete.

1.3 Units

Energy units. It is measured in Joule (J) (S.I.). In radiation, environment eV is preferred. One eV is defined as the energy gained by one electron in accelerating through a potential difference of 1 V. 1 eV is equal to 1.6 × 10−19 J. Energies in nuclear reactions are usually expressed in MeV or keV.

Flux. It is defined as the number of particles passing through a given area per unit of time. Since the radiation is isotropic, the area considered is a sphere with a cross-section of 1 cm^2. The unit of flux is $cm^{-2}s^{-1}$.

Fluence. It is the time integral f the flux. The unit is cm^{-2}.

Absorbed dose. It is the measure of the energy deposited in the matter by ionizing radiation per unit mass.

Gray (Gy). It is the standard unit of the absorbed ionizing-radiation dose, equivalent to one joule per kilogram ($J \cdot Kg^{-1}$).

Rad. It is the old unit of absorbed ionizing-radiation dose. 1 Rad is equivalent to 0.01 Gy.

Sievert (Sv). It is a derived unit of ionizing radiation dose. It is a measure of the health effect of low levels of ionizing radiation on the human body. The Sievert is used for radiation dose quantities such as equivalent dose and effective dose, which represent the risk of external radiation from sources outside the body, and committed dose which represents the risk of internal irradiation due to inhaled or ingested radioactive substances.

1.4 Radiation environments

Space, nuclear reactors, controlled fusion, and particle accelerators are typical environments where electronic devices are damaged by radiation.

1.4.1 The space environment

Different types of energetic particles compose the space environment: their energies go from some keV to GeV and beyond. These particles are either trapped by the Earth's magnetic field or are passing through the solar system. The main elements of the space environment are:

- *Trapped radiation*: It consists of charged particles - ions and electrons - trapped by the Earth's magnetic field. These particles form the radiation belt.
- *Cosmic rays*: They are high energy particles that move through space at nearly the speed of light. Most cosmic rays are made of high-energy protons. Within cosmic-rays, other

sub-atomic particles like neutrons, electrons, and neutrinos can be found. They originate from the sun, from outside of the solar system, and even from distant galaxies

- *Solar flares*: energetic protons are produced by solar eruptions. Alpha particles, heavy ions, and electrons are also produced but with a minor contribution.
- *Plasma*: a plasma made of electrons and protons with energies up to 100 keV pervades the space environment.
  Orbit and duration determine the radiation effects on satellites.

1.4.2 The CERN environment [2]

Space and the accelerator's environments are strongly different. The mixed field present in a modern accelerator consists of a wide spectrum of particles: charged and neutral hadrons (protons, pions, kaons, and neutrons), photons, electrons, and muons, ranging from very low (thermal) energies up to the TeV range. The complexity of the field is due to 3 different causes:

1) Primary particle collisions in the experimental areas.
2) Distributed beam losses around the machine.
3) Interaction between the residual gas and the beam inside the beam pipe.

At CERN, groups of engineers and physicists study the fundamental particles using the world's largest and most complex scientific instruments. By colliding the particles at close to the speed of light, the physicists obtain information about how the particles interact and the fundamental laws that govern the universe. To obtain this information, they use specific particle accelerators and detectors. Accelerators are machines that use the electromagnetic field to bring the particle beam to high energies before letting it collide with another or with a fixed target. These collisions are observed and recorded by the detectors.

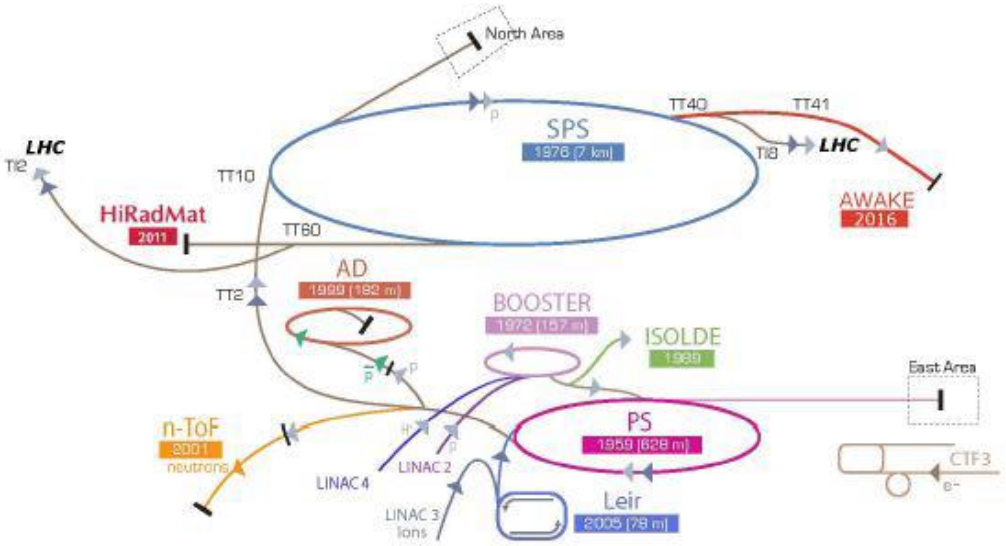


Figure 1.1: The CERN accelerator chain

In particular, the accelerator has a complex structure consisting of a chain of different machines, shown in Figure 1.1. Each of them increases the energy of the particle beam before letting it enter the next machine of the chain. In the LHC – the last element in this chain – particle beams are today accelerated up to 7 TeV. Most of the accelerators in the chain have their experimental halls where beams are used for experiments at lower energies. From the hydrogen gas is possible to obtain a proton source: electrons are excited using an electric field to obtain protons. Linac 2, the first element of the chain, increases the energy of the beam to 50 MeV. Then, it is injected into the Proton Synchrotron Booster (there, the energy arrives at 1.4 GeV) and subsequently in the Proton Synchrotron (PS) where the energy is increased to 25 GeV.

The next element in the chain is the Super Proton Synchrotron (SPS) where the beam energy reaches 450GeV. At this point, the protons are finally injected into the two beam pipes of the LHC. The two pipes differ only in the direction of circulation: in one the beam circulates anticlockwise while in the other pipe, it circulates clockwise.

The LHC ring is filled by it in 4 minutes and 20 seconds while it is necessary to wait 20 minutes to allow the protons to reach the maximum energy. The two beams are brought into collision inside four detectors – called ALICE, ATLAS, CMS, and LHCb – where the total energy at the collision point is equal to 14 TeV. Antiproton Decelerator and the Online Isotope Mass Separator (ISOLDE) are also part of the accelerator complex. The CERN Neutrinos to Gran Sasso (CNGS) project, the Compact Linear Collider test area (CLIC), and the neutron flight time structure (nTOF) are also fed by this system.

Protons are not the only particles accelerated: lead ions for the LHC are obtained from a source of vaporized lead and injected in the LINAC 3 before being collected and accelerated in the Low Energy Ion Ring (LEIR). After they follow the same path as the

protons and reach the maximum energy. In the various underground areas, the high radiation levels existing, represent a risk for the electronic devices that must work in these areas.

As to what regards the different locations of the accelerator in which electronic devices must operate, a first division can be made considering 3 zones:

1) The experiments where the devices use components designed to operate in harsh environments.
2) The injector line.
3) The LHC machine itself.

For the last two zones, three sub groups can be further defined:

i. the tunnel, in which the amount of electronics is minimized and if present, where equipment was initially tested for and validated against SEEs for the respective fluences.

ii. heavily-shielded areas (e.g, for the LHC close to interaction points and known as UJs).

iii. lightly-shielded areas, with less intense, harder spectra than their more protected counterparts (e.g. for the LHC the so-called RRs).

In the next Table, a definition of the quantities used to characterize the particle energy spectra is presented.

Quantity	Notation	Definition [2]
High Energy Hadron	HEH	Hadron flux above 20 MeV.
Intermediate neutron	n_{int}	Neutron in the 0.2-20 MeV energy range.
Equivalent Thermal Neutron Flux	thn_{eq}	Thermal Neutron contribution.
Risk Factor	R	Ratio between the equivalent thermal neutron flux and HEH.
Hardness Energy	$H_{0.x}$	Energies of particle spectra at which 10x % (for instance $H_{0.1}$ stands for 10 %) of the total HEH lie above

Table 1.1: Quantities defined to characterize particle spectra

In the next part, the critical CERN location will be presented and compared with other well-known radiation environments.

1.4.3 LHC-Machine Environment

In Figure 1.2, the yearly HEH flux, the typical particle compositions, the intermediate neutron contribution, the R-factor, and the hardness energy are reported for nominal work conditions (usually referred to as 50 fb^{-1} (inverse femtobarns, a unit proportional to the number of proton-proton interactions in the accelerator) of yearly integral luminosity at 7 TeV energy) and for the 3 different areas described previously (tunnel, the UJ, and the RR).

Critical Area	$HEH/cm^2/y$	Composition (%)				R	Hardness Energy (MeV)	
		n	p	$\pi^{\pm}$	n_{int}		$H_{0.1}$	$H_{0.01}$
UJ	$\sim 2.5 \cdot 10^9$	99	1	0	36	10.5	180	360
RR	$\sim 1.0 \cdot 10^9$	71	13	16	26	7.5	690	2.8 GeV
Tunnel	$\sim 6.0 \cdot 10^{11}$	45	18	37	19	1.2	1.8 GeV	5.7 GeV

Figure 1.2: Expected maximum nominal yearly HEH fluence for different LHC areas together with the respective typical HEH composition, R factor, and hardness energies [2].

1.4.4 LHC-Experiment Environment

In addition to the LHC accelerator locations relevant to electronic component operation, an example of representative experiment spectra is here shown for the largest CERN experiment, ATLAS. Figure 1.3 shows the SEE relevant parameters for two cylindrical detection surfaces in the ATLAS detector, both centered on the interaction point. The first (labeled as inner) is located at a radius of 20 cm from the beam line and extends over 3.0 m in the *z* (beam) direction, thus representative for the inner detector locations where only very limited electronic elements are situated. The second (labeled as outer) has a radius of 390 cm and a z dimension of 6.0 m and corresponds to locations where, though limited, more complex electronics can be found. It is to be noted that the location closest to the interaction point shows a harder HEH spectrum and a composition very rich in charged pions.

Critical Area	$HEH/cm^2/y$	Composition (%)				R	Hardness Energy (MeV)	
		n	p	$\pi^{\pm}$	n_{int}		$H_{0.1}$	$H_{0.01}$
Outer	$\sim 3.5 \cdot 10^9$	67	5	28	25	1.1	460	710
Inner	$\sim 1.0 \cdot 10^{12}$	4	7	89	1	2.0	3.8 GeV	12 GeV

Figure 1.3: Expected maximum nominal yearly HEH fluence for different LHC areas together with the respective typical HEH composition, R factor, and hardness energies [2].

1.4.5 LHC Injector Chain Environment

In Figure 1.4, the yearly HEH flux, their typical particle compositions, the intermediate neutron contribution, the R-factor, and the hardness energy are reported for nominal work conditions and the 3 different accelerators (LINAC 4, PS, and SPS).

Accelerator	HEH/cm^2/yr	Composition (%)				R	Hardness Energy (MeV)	
		n	p	$\pi^{\pm}$	n_{int}		$H_{0.1}$	$H_{0.01}$
LINAC 4	10^{10}	99	1	0	86	1.5	70	100
PS	10^{11}	61	17	22	19	4.9	900	2400
SPS	10^{12}	70	12	18	29	48.9	940	5100

Figure 1.4: Expected maximum nominal yearly HEH fluence for different LHC areas together with the respective typical HEH composition, R factor, and hardness energies [2].

1.4.6 Comparable Radiation Environment: Atmosphere

The interest in ground level and avionic radiation induced effects has become more and more important during recent years. With the scaling of electronic components, the sensitivity per integrated chip can develop significantly, reaching levels that can negatively impact the reliability of systems even for ground level applications. Therefore, comparing the accelerator environment with its atmospheric counterpart is highly interesting to underline the possible similarities and differences. For this purpose, Figure 1.5 shows the main characteristics of the atmospheric spectra for different altitudes.

Altitude	$HEH/cm^2/y$	Composition (%)				R	Hardness Energy (MeV)	
		n	p	$\pi^{\pm}$	n_{int}		$H_{0.1}$	$H_{0.01}$
350 m	$\sim 1.6 \cdot 10^5$	93	7	0	21	0.12	340	1.3 GeV
10 km	$\sim 1.7 \cdot 10^7$	82	18	0	18	0.08	920	5.0 GeV
20 km	$\sim 3.8 \cdot 10^7$	68	32	0	14	0.06	2.9 GeV	13 GeV

Figure 1.5: Expected nominal yearly HEH for different atmospheric Environments together with the respective typical HEH composition (in HEH percentage), R factor, and hardness energies. Different altitudes are included for the Geneva area, Switzerland.

The following general conclusions can be drawn when comparing these values to the accelerator environment cases:

(i). The flux levels are significantly below those encountered in critical accelerator areas.

(ii). The proportion of HEH is similar to the accelerator environment in the case of neutrons and protons (with different possible combinations), however, pions do not reach 1% even at the highest altitude considered (while they can be dominating in certain accelerator cases).

(iii). The intermediate neutron contribution is similar; however, the thermal neutron proportion (represented through the R-factor) is much lower than in the accelerator cases. It is to be noted that the thermal neutron flux for a given location strongly depends on the materials surrounding it, therefore for components inside buildings or airborne, the R-factor can be very different.

(iv). The spectra are of a hard character, with the hardness parameters increasing rapidly with altitude. HEHs at 20 km are roughly as energetic as in the LHC-tunnel environment.

1.4.7 Comparable Radiation Environment: Proton-belt Space

Of the relevant space environments for electronic component operation, the Earth's inner proton radiation belt is the context that most resembles that of an accelerator owing to the dominance of singly charged hadrons and thus having indirect ionization as the main source of SEE-induction. This environment is highly relevant for Low-Earth Orbits (LEO), including those of the International Space Station (ISS) and more generally Earth-observation missions. To compare this radiation environment with high-energy accelerators, two LEO orbit spectra examples (that of the International Space

Station and the sun-synchronous orbit of the Proba-II mission) are simulated. The main relevant characteristics of the spectra are shown in Figure 1.6. It is to be noted that in the case of the latter, protons are typically present together with other hadron species (notably neutrons and charged pions).

Mission	Orbit		HEH Fluence $(/cm^2/yr)$	$H_{0.1}$ (MeV)	$H_{0.01}$ (MeV)
	Altitude (km)	Inclination			
ISS	450	51.5^o	$7.3 \cdot 10^8$	250	530
Proba-II	800	98.3^o	$2.7 \cdot 10^9$	280	1.5 GeV

Figure 1.6: Information of two examples of trapped proton-belt spectra averaged over the orbit. Protons are the only relevant HEH.

The following conclusions can be extracted when comparing the proton belt environment with the accelerator context:

(i). The HEH fluxes in the proton belt are similar to those in critical LHC locations.

(ii). The accelerator field is typically composed of protons, charged pions, and neutrons, or of the latter in heavily shielded areas, whereas protons are the only relevant hadron for the inner belt case.

(iii). Spectra are generally harder (more energetic) in the accelerator case (except for the negligible contribution from Galactic Cosmic Ray (GCR) protons) whereas the low-energy interval (about 1 MeV), potentially relevant for direct ionization effects in deep sub-micron technologies, is more prominent in the case of the inner radiation belt.

1.5 Radiation effects on electronic devices

If radiation is applied to an electronic circuit, a permanent or a transient variation of its electric properties can be observed. In particular, the radiation can cause the device to fail.

The radiation effects can be divided into two groups: the cumulative effects (Total ionizing Dose (TID) and Displacement Damage (DD)) and single-particle effects (Single event effects).

1.5.1 Displacement Damage (DD)

The displacement damage is classified as a non-ionizing cumulative effect. Due to a particle strike, an atom can be displaced from its original lattice position.

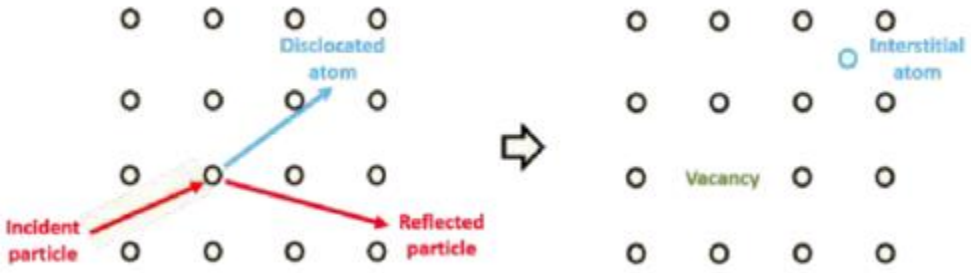


Figure 1.7: Effect of displacement damage

In an electronic device built on Si Substrate, to have a defect in Si lattice, the incident particle should transfer to the Si recoiled atom minimum energy of 21 eV.

If the recoil energy is more than 1-2 keV, the recoiled atom can dislocate other lattice atoms and create a region called a cluster. The cluster is a region with a high density of vacancy defects.

Protons and neutrons do not interact in the same way with the lattice atoms. Due to the lack of charge, neutrons interact with the atom by impact while the protons (charged particles) interact with it by Coulomb repulsion.

While protons and neutrons usually generate point defects for energies greater than 158 eV and clusters for energies greater than 35 keV, electrons need higher energies to create a vacancy defect (from 270 keV) and cluster (from 8 MeV). Gamma rays can also produce defects.

Due to the defects produced, new traps can be generated in the band gap of the semiconductor that influences the electrical performance of the device.

What can happen is shortly summarized in the following list.

- *Thermal generation of e-h pairs*: The defects are close to the midgap, which allows an electron in the valence band to pass in the conduction band, leaving a hole in the valence band.

This effect leads to an increase in the leakage current of a p-n junction, due to the carrier generation in the depletion region. This phenomenon is relevant for BJTs and particle detectors.

- *Recombination of the carriers*. The traps work as a point of recombination between the electrons in the conduction band and the holes in the valence band. The recombination of e-h pairs decreases the lifetime of the carriers, causing a degradation of the direct current of a p-n junction. Similarly, the recombination phenomenon is relevant also for BJT by degrading the gain.
- *Compensation:* The defects generated in the Si substrate can introduce some acceptor levels in the band gap, which causes a compensation of the donor atoms followed by a decrease of the Fermi level. In case of strong radiation levels, the substrate can invert its doping type, from n to p. The BJT technology is sensitive to the compensation process, as it increases the collector resistance.
- *Tunneling:* This mechanism happens only under strong electric fields. The traps in the energy gap help the electrons in the valence band to pass by tunneling to the conduction band.
- *Trapping of the carriers*. The new energy levels of the traps induced by DD are close to the valence band or the conduction band. The holes or electrons are temporarily trapped in these levels.

A parameter called Non Ionizing Energy Loss (NIEL) is used to quantify the DD produced by different particle types and energies. Due to the dependence of the magnitude of the displacement damage on the energy loss per unit of length and on the interaction of the particle with the lattice structure, the formula for NIEL is the following:

$$NIEL = \frac{1}{\rho}\frac{dE}{dx} \quad \left[\frac{MeV\ cm^2}{g}\right] \qquad (1.2)$$

Where ρ is the density of the material. For mixed radiation composed of different particles of energy E and Φ (E) fluence, the displacement damage Dose is defined as:

$$DDD = \int NIEL(E)\frac{d\Phi(\mathrm{E})}{dE}dE \qquad (1.3)$$

The DDD is usually referred to as the equivalent fluence (Φ_{1MeV}) of a monoenergetic 1 MeV neutrons beam. This equivalent quantity gives the equivalent fluence of 1 MeV neutron beam which generates the same DD effects on the device. The conversion from DDD to Φ_{1MeV} is carried out by dividing the DDD by the NIEL of the 1 MeV neutrons:

$$\Phi_{1MeV} = \int \frac{NIEL(E)}{NIEL_n(1MeV)} \frac{d\Phi(\mathrm{E})}{dE} dE \qquad (1.4)$$

1.5.2 Total Ionizing Dose (TID)

The other cumulative effect is the TID. It depends on the Coulomb interaction between particles and electrons of the lattice. This is the main difference with the DD, where the interaction is nuclear type. TID does not depend only on the energy and type of the incident particle but also on the density of the medium. For this reason, the TID is proportional to electron stopping power, better known as Linear Energy Transfer of the impinging particle:

$$LET = \frac{1}{\rho} \frac{dE}{dx}|_e \qquad \left[\frac{MeV\ cm^2}{g}\right] \qquad (1.5)$$

By definition, the dose is the energy per unit mass absorbed by the material when it is subjected to ionizing radiation and can be expressed in equation (1.6):

$$D = \frac{dE}{dm}|_e \qquad \left[\frac{MeV}{g}\right] \qquad (1.6)$$

The Dose is directly proportional to the fluence and depends on the LET of the impinging particles. For this reason, it is possible to write it as a function of the LET.

$$D = \int LET(E) \frac{d\Phi(\mathrm{E})}{dE} dE \qquad (1.7)$$

1.5.3 TID in a MOS structure

In a Metal-Oxide-Semiconductor structure, the most sensitive layer to the ionizing radiation is the Gate Oxide (SiO2).

When a particle passes through a MOS structure there are 2 effects:

- E-h pairs generation.
- Trap creation at the SiO2-Si interface.

After the generation, part of the generated pairs recombines while the remaining part is divided by the Electric Field. For Vgb>0, electrons move towards the Gate electrode while the holes move towards the interface SiO2-Si through a particular mechanism called hopping transport. During this process of migration and trapping of the holes in the oxide, ions of Hydrogenous are produced. This Ion can interact at the interface SiO2-Si generating traps.

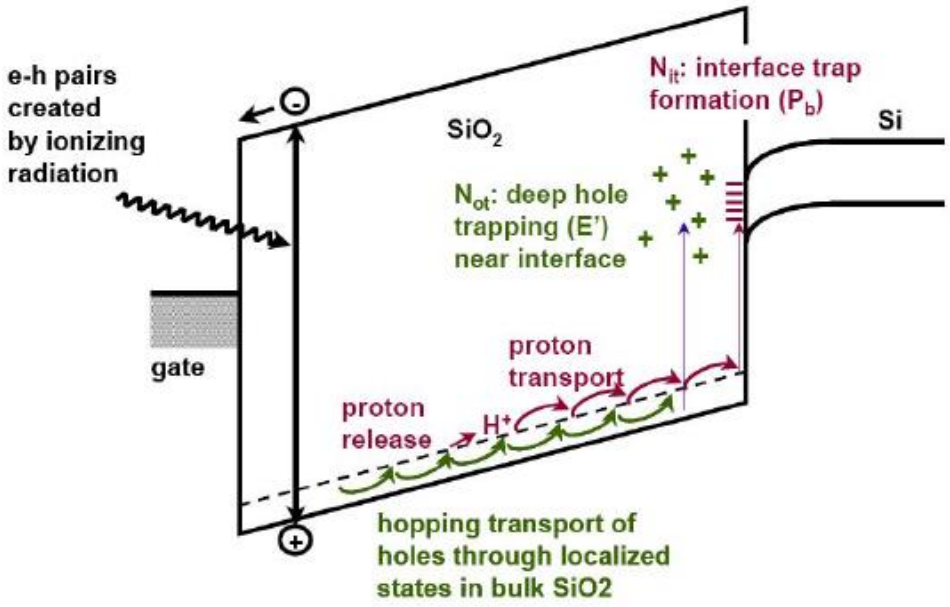


Figure 1.8: Effect of TID in a MOS structure

The charges of the traps depend on the MOS type (negative for NMOS and positive for PMOS)

The occupation of the interface traps is determined by Fermi level and leads to a shift of the threshold voltage, depending on device polarization and to degradation of timing parameters (i.e. careers lifetime).

$$\Delta V_{IT} = -\frac{\Delta Q_{IT}}{C_{OX}} \qquad (1.8)$$

Where ΔQ_{IT} is the trapped charge at the interface.

At the same time, the hole trapped in the oxide is more than the electrons because the mobility of the first one is lower.

The holes trapped into SiO2 lead to a negative variation ΔV_{OT} of the threshold voltage.

$$\Delta V_{OT} = -\frac{q}{C_{OX}} \Delta N_{OT} \qquad (1.9)$$

Where:

- q: electron charge
- C_{OX} : oxide capacitance per area unit
- ΔN_{OT}: trapped holes density into the oxide.

The voltage shift ΔV_T can be defined as:

$$\Delta V_T = \Delta V_{OT} + \Delta V_{IT} \qquad (1.10)$$

For an nMOS, when the dose is lower than about 1 kGy there is a negative shift of the threshold voltage since positive charge

trapped into the oxide is predominant. Instead, the threshold voltage starts to increase when the dose is greater than 1000 Gy because the negative charge at the interface contributes.

For a pMOS, there is a constant decrease of the threshold voltage. Therefore, while the N-mos up to a dose of about 1000 Gy will be hard to switch off, the pMOS will be hard to switch on.

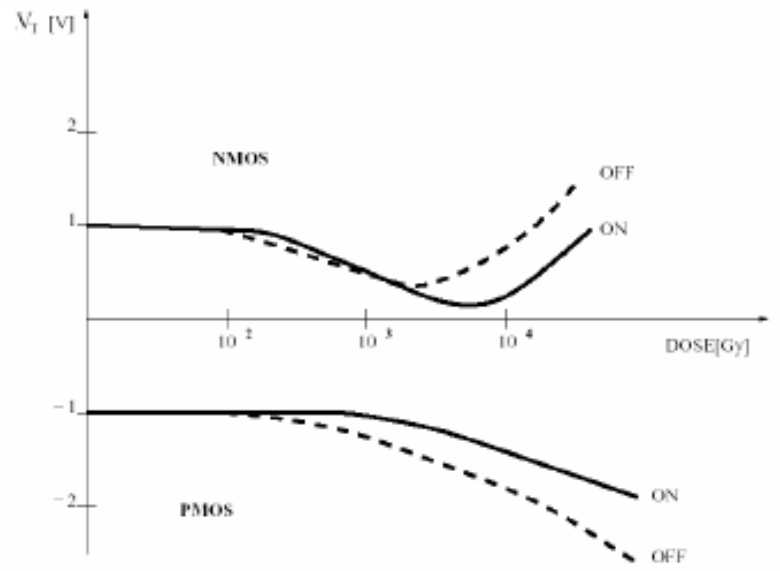


Figure 1.9: Effect of TID on Threshold Voltage for nMOS and pMOS device

After the ionizing irradiation, the holes are not trapped into SiO2 permanently. There is an effect called Annealing which helps neutralize the charge. The Annealing can be thermal or by tunnel effect and the neutralization time is in a range from a few ms up to years.

1.5.4 Single Event Effects (SEE)

Single Event Effect (SEE) is caused by a single, energetic particle, and can take on many forms. It can lead to different effects ranging from data corruption and transient disturbance to destructive effects. The following is a short list and description of some SEE type:

- Single Event Latch-Up (SEL): this condition occurs when a high current state is induced in the device by a charged particle. This event may be destructive for the device. It is typical of CMOS structures because it is caused by the activation of parasitic PNPN SCR that is normally in the off state. If one parasitic transistor is turned on, feeds the base of the other one. This positive feedback increases the current until the circuit fails or burns out.

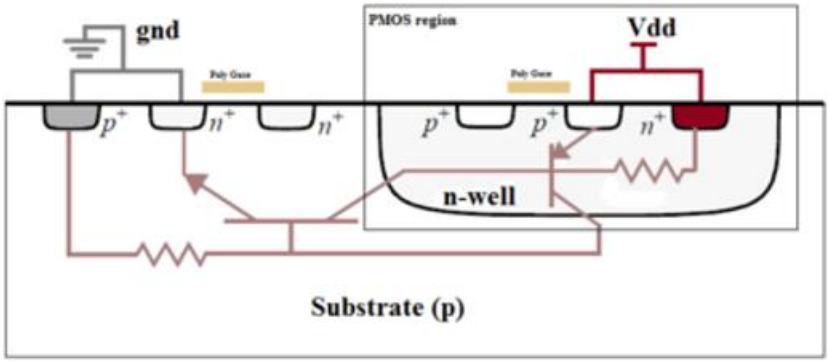


Figure 1.10: Effect of SEL on CMOS structure.

- Single Event Burnout (SEB): it destroys the device due to a high current state. This is typically found in power transistors. It can be associated with a second breakdown for power devices due to a critical current that turns on a parasitic BJT. In fact, due to a particle strike, an e-h pair can be generated in the depletion region between body and drain, and consequently is separated by its electric field generating, therefore, current. This current can be high enough to create a voltage drop in the body region representing the base of the parasitic BJT, consequently turning it on.
- Single Event Gate Rupture (SEGR): an ion through the gate (but avoiding the p-regions), generates a plasma filament through the n-epi layer that applies the drain potential to the gate oxide, damaging (increased gate leakage) or rupturing the gate oxide insulation (device destruction).
- Single Event Upset (SEU): When a charged particle strikes one of the sensitive nodes of a memory cell, such as a drain in an off state transistor, it generates a transient current pulse that can turn on the gate of the opposite transistor. The effect can produce an inversion in the stored value, in other words, a bit flip in the memory cell. Memory cells have two stable states, one that represents a stored '0' and one that represents a stored '1.' In each state, two transistors are turned on and two are turned off (sensitive nodes). A bit-flip in the memory element occurs when an energetic particle causes the state of the transistors in the circuit to reverse. This effect is called Single Event Upset (SEU) and it is one of the major concerns in digital circuits because usually memory cells are designed with very compact transistors that present high soft error sensitivity (low critical charge). The SEU phenomenon is illustrated in Figure 1.11.

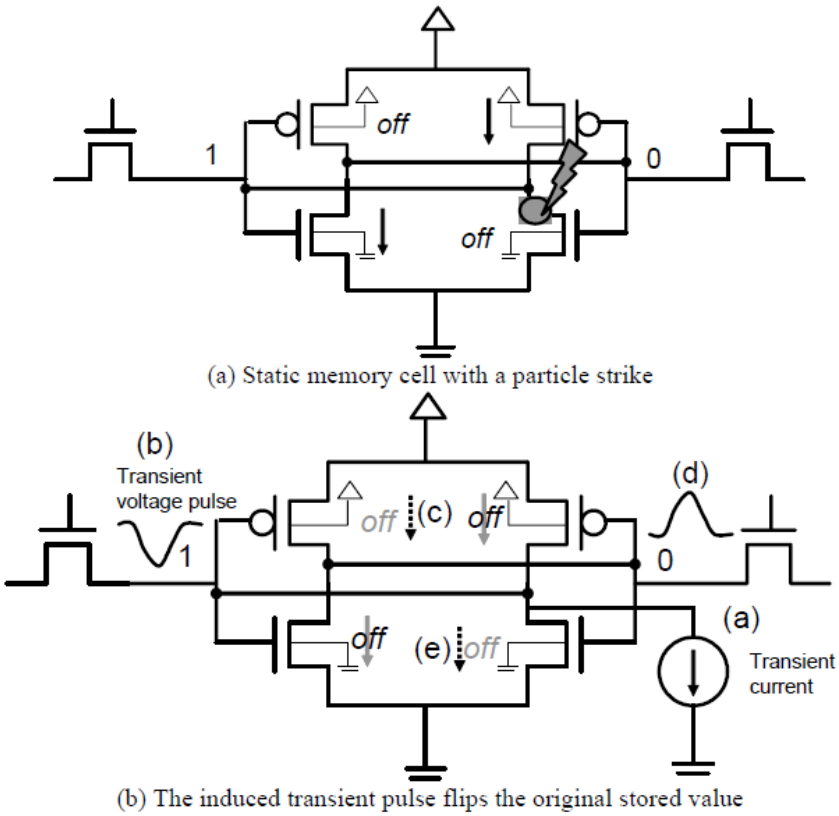


Figure 1.11: Single Event Upset (SEU) in a static memory cell

- Single Event Transient (SET): When a charged particle hits the combinational logic block, it also generates a transient current pulse. This phenomenon is called the single transient effect (SET).
  If the logic propagates the induced transient pulse, then the SET will eventually appear at the input of a latch, where it may be interpreted as a valid signal. This case is depicted in Figure 1.12.

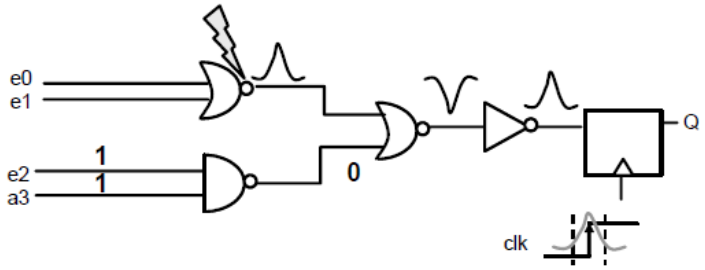


Figure 1.12: Single Event Transient (SET) in a combinational circuit

The cross-section (cm^2) represents the probability of a SEE occurs, and it is expressed, for a single device, in (1.12).

$$\sigma = \frac{N_{SEEs}}{Fluence} \quad [cm^2] \qquad (1.12)$$

1.6 SEE mitigation strategy for FPGA and MCU

Single Event Effects (SEEs) affecting COTS devices may have catastrophic effects if neglected, and for this reason, SEE mitigation techniques have to be employed.

In this chapter, some techniques against SEEs that can be implemented on FPGA or MCU will be presented. Firstly hardware solutions are discussed (Internal watchdog, External watchdog, Logic-level based techniques) and then, information based solutions.

1.6.1 Internal Watchdog

A watchdog timer is an electronic timer that is used to detect system error and recover it. In normal conditions, the system regularly resets the watchdog timer while in case of a hardware fault or program error, it cannot reset it and so the timer will elapse and generate a timeout signal. This signal is used to restore the system. Watchdog timers are commonly found in embedded systems and other computer-controlled equipment where humans cannot easily access the equipment or would be unable to react to faults on time. For example, space probes are not physically accessible to human operators and they would be permanently disabled in case of failure (if they do not have an automatic restore).

If an SEL affects the system, the Watchdog timer can restore the system avoiding a high current state. Moreover, if an SEU occurs in the internal register of a CPU, such as a program counter, leads the system in a loop situation, the Watchdog timer can restore normal operating condition.

1.6.2 External Watchdog

An external watchdog is an external circuit able to reset /wake up/control a connected system.

1.6.3 Logic-level based techniques

The logic-level based techniques are all faulted tolerant techniques that can be easily applied at the gate level to tolerate soft errors (SET in combinational circuits and SEU in sequential circuits). They can be applied directly in hardware description-level languages (Verilog). These techniques will be presented and discussed in the next section.

1.6.4 Hardware redundancy techniques

Hardware redundancy is provided by adding extra hardware into the design. It has always been successfully used to detect and vote out errors of the Logic.

The first approach is duplication with comparison (DWC): it consists to replicate the module and compare their outputs. If the outputs are different, it means that there was an error.

This technique can be used for both combinational and sequential logic to SET and SEU detection.

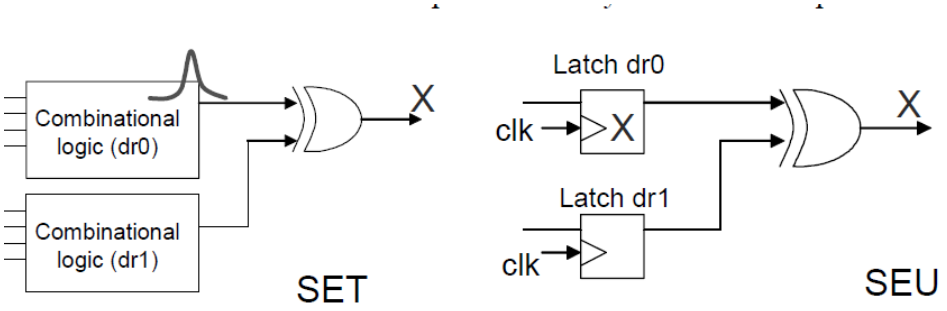


Figure 1.13: Duplication with comparison scheme

Comparators are the most important elements of this structure because they must be immune to errors. To make them less sensitive to upsets, they can be duplicated, or usually, they are designed with a larger transistor.

However, the main problem with this scheme is that it can only notify that an error occurs but cannot inform which module or piece of logic has the error and therefore, which is the correct output.

To detect and vote the correct output, the N-modular redundancy approach is used. The idea is really simple: use more than two identical modules and use their output as input for a vote circuit that will decide which one is the correct output.

In the N-modular redundancy, normally N is chosen equal to three and this approach is called triple modular redundancy (TMR).

Figure 1.14, illustrates this approach used for sequential elements: three flip flops stores the same output of a combinational logic block and a Majority voter (MAJ is the vote circuit) decided which output is the correct one. If an upset occurs, it is expected that at least two outputs of three are correct and so the vote must only count which output is repeated more times.

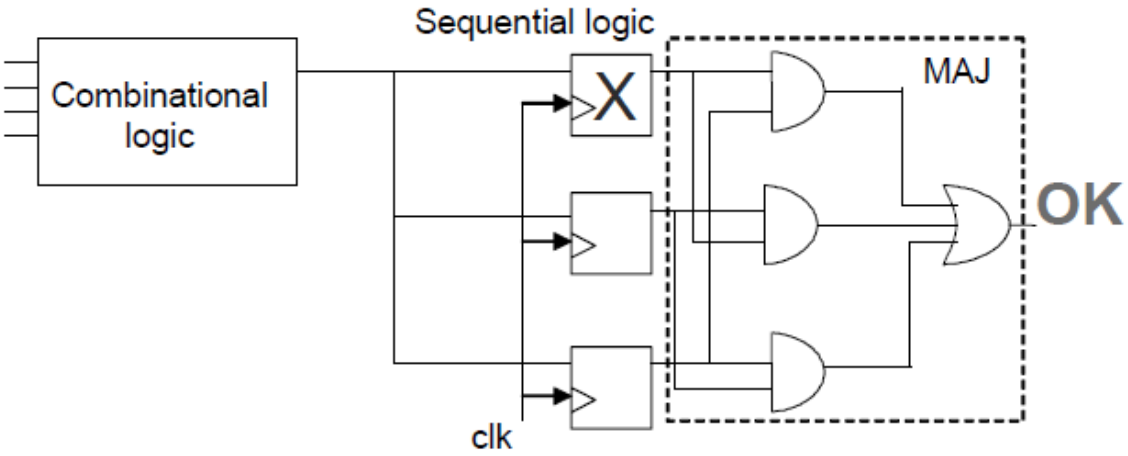


Figure 1.14: Triple Modular Redundancy (TMR) in the sequential logic and the majority voter (MAJ)

This approach presents two limitations:

1) If a SET occurs in the combination logic, it will not be detected because it will propagate through the three flip-flops (Figure 1.15).

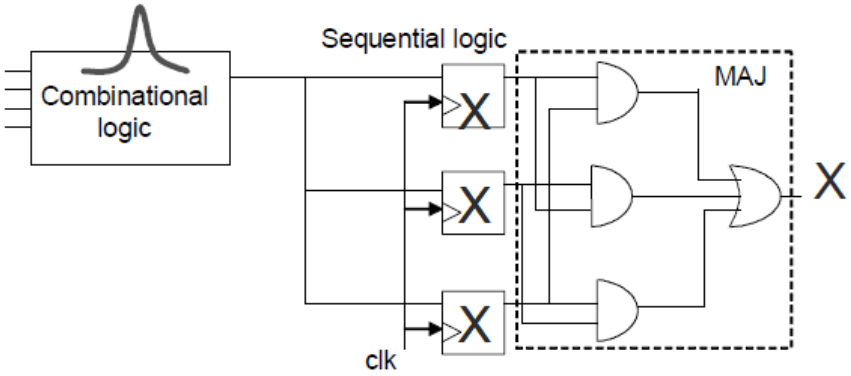


Figure 1.15: SET propagation in a TMR scheme in the sequential logic

2) The SET can occur also in the MAJ that is a combinatorial circuit (Figure 1.16).

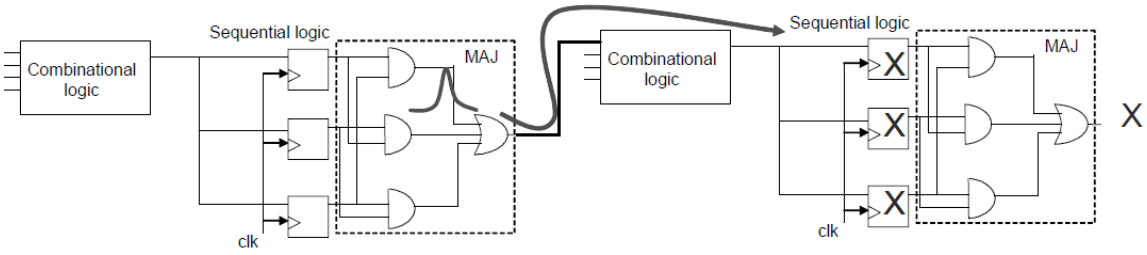


Figure 1.16: SET propagation in the majority voter (MAJ)

A solution can be to triplicate also the combinational logic and the MAJ. In this case, a 4th voter will be necessary to vote the correct output.

If a SET occurs in one of the combinational logic blocks, the SET may be captured by only one of the flip flops and the MAJ voter will be able to choose the correct output, as represented in Figure 1.17.

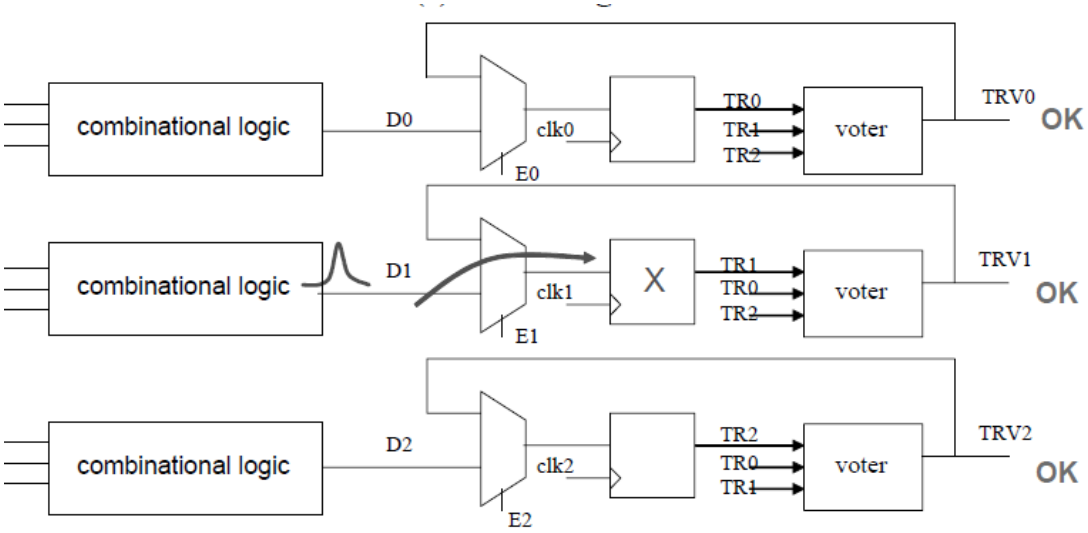


Figure 1.17: SET Mitigation in the logic

As it is possible to see there is a mux introduced in the structure: it is really important in the case of an SEU in the flip flop. If an SEU occurs in one of the flip-flops, the MAJ voter chooses the correct output, as shown in Figure 1.18 and at the next clock the correct output can be loaded to the flip-flop clearing the SEU.

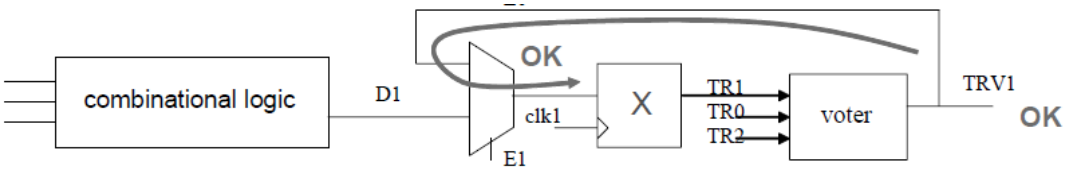


Figure 1.18: SEU Mitigation

However, there are always three limitations:

1) The 4TH voter, used to compare the 3 MAJ illustrated, can be affected by the SET problem.
2) This system does not protect against simultaneous double faults.
3) Power dissipation due to the amount of logic required.

1.6.5 Time redundancy techniques

In Time redundancy techniques solutions, the data are processed at a different time and allow the detection of a SET. The idea consists to use two different flip-flops: one controlled by the normal clock and one by a delayed clock. The delay must be chosen according to the larger SET time duration that can occur. Figure 1.19; illustrate the simplest way to implement this solution.

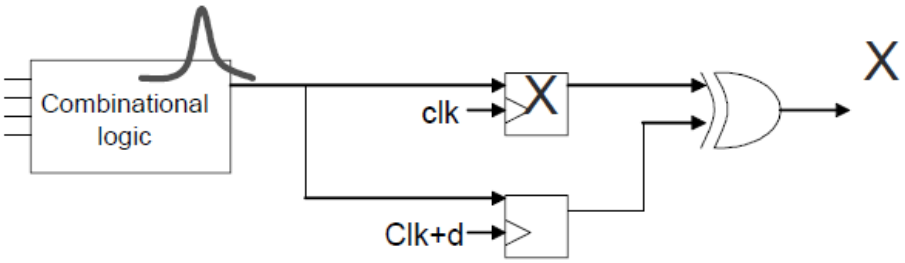


Figure 1.19: Time redundancy scheme for SET detection

Normally a full time redundancy is implemented: in this case, the output of the combinational logic is captured at 3 different moments where the clock edge of the second latch is shifted by the time delay d and the clock of the third latch is shifted by the time delay 2 d. It has two limitations:

1) SET in the last MAJ voter will not be detected.
2) This method is suitable only for SETs with a time duration not higher than 10% of the clock period.
3) Less area is required but it does not detect SEUs.

1.6.6 Information Redundancy

During the transfer of data or the saving in a memory unit, errors can occur. To detect and correct them, redundancy is introduced also into the data: this is called information redundancy.

The most important technique used for information redundancy is the Error Correcting Code (ECC) technique used to mitigate SEU in integrated circuit (IC) and based on information redundancy in memory structures. This technique can be used to detect and correct a single bit error (simplest implementation) to correct multi-bit errors. Figure 1.20 illustrates 8-bit data being written and read from a register. If an SEU occurs and there is no information redundancy, an error occurs but is not detected. If a parity bit is added to the stored data, an error can be detected when the parity bits do not match.

However, many applications require a complex ECC that not only detects the error but restores the original value. In these cases, an encoder is used to identify the error position and a decoder to correct it.

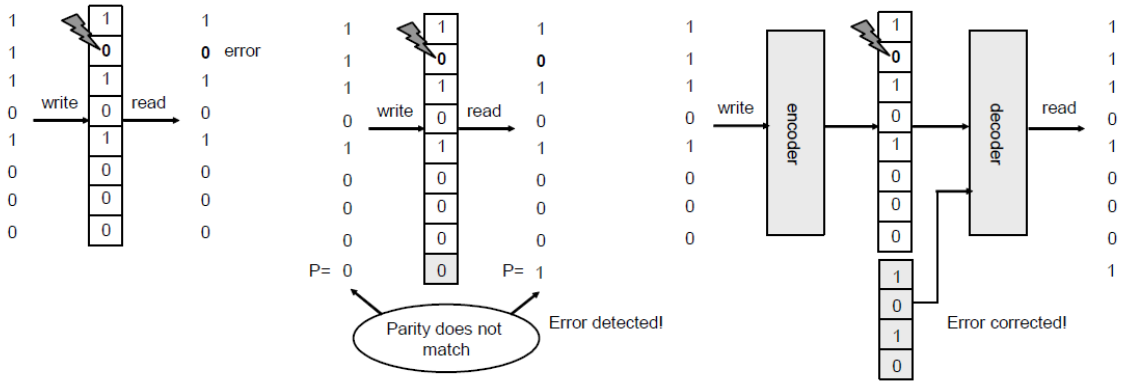


Figure 1.20: Error Correcting Code Principle

In the next section, redundant bits will be defined and two techniques of ECC will be presented.

Redundant bits

Redundant bits are extra bits that are generated from and added to the data, to detect and correct an error that occurred during the transfer or the saving of the data in memory. The number of redundant bits can be calculated using the following formula:

$$2^r = m + r + 1$$

Where r is the number of redundant bits and m is the number of bits of the original data

Parity bits

This technique consists to append to the data 1 bit that provides information on the number of 1s in the data (odd or even). This technique is able only to detect an error and has two versions:

1) Even parity bit: in this case the parity bit is set equal to 1 when the total number of 1s in the data is odd, making the total

count of occurrences of 1s in the new data (information data + parity bit) an even number. Otherwise, it is set equal to 0.

2) Odd parity bit: in this case the parity bit is set equal to 1 when the total number of 1s in the data is even, making the total count of occurrences of 1s in the new data an odd number. Otherwise, it is set equal to 0.

General Algorithm of Hamming code

The Hamming Code is simply the use of extra parity bit to allow the identification of an error.

1) Write the bit positions starting from 1 in binary form (1, 10, 11, 100, etc.),

2) All the bit positions that are a power of 2 are marked as parity Bits.

3) All the other bit positions are marked as data bits.

4) Each data bit is included in a unique set of parity bits, as determined by its bit position in binary form.

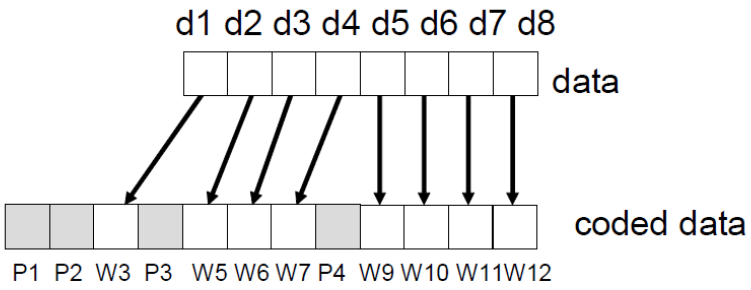
The first parity bit covers all the bits positions (of the coded data) whose binary representation includes 1 in the least significant position (1, 3, 5, 7, 9, etc...).

The second parity bit covers all the bits positions whose binary representation includes a 1 in the second position from the least significant bit (2, 3, 6, 7, 10, 11, etc...)

The fourth parity bit covers all the bits positions whose binary representation includes a 1 in the fourth position from the least significant bit (4, 7, 12, 15, 20, 23, etc…).

5) Since we check for even parity set a parity bit to 1 if the total number of ones in the positions it checks is odd.

6) Set a parity bit to 0 if the total number of ones in the positions it checks is even.



Encoder block: check bits generation

P1 = W3 **xor** W5 **xor** W7 **xor** W9 **xor** W11
P2 = W3 **xor** W6 **xor** W7 **xor** W10 **xor** W11
P3 = W5 **xor** W6 **xor** W7 **xor** W12
P4 = W9 **xor** W10 **xor** W11 **xor** W12

Figure 1.21: Parity bits in Hamming Code

Error detection and correction

Using a decoder and some XOR ports, it is possible to obtain new parity bits, if they are all equal to 0s no bit is changed. Instead, if one is changed, there will be 2 or more new parity bits different from 0.

Figure 1.22 illustrates how it is possible to detect the position of the bit that is changed.

Syndrome P1 = P1 **xor** W3 **xor** W5 **xor** W7 **xor** W9 **xor** W11
Syndrome P2 = P2 **xor** W3 **xor** W6 **xor** W7 **xor** W10 **xor** W11
Syndrome P3 = P3 **xor** W5 **xor** W6 **xor** W7 **xor** W12
Syndrome P4 = P4 **xor** W9 **xor** W10 **xor** W11 **xor** W12

Decoder block: mask generation

Syndrome P4P3P2P1	Mask P1P2W3…….W11W12
0000	no error
0001	**1**00000000000
0010	0**1**0000000000
0011	00**1**000000000
0100	000**1**00000000
0101	0000**1**0000000
0110	00000**1**000000
0111	000000**1**00000
1000	0000000**1**0000
1001	00000000**1**000
1010	000000000**1**00
1011	0000000000**1**0
1100	00000000000**1**

Figure 1.22: Hamming code check bits generation for an 8-bit word, 12-bit coded

Part II
State of the art

Chapter 2

2 Interfacing FPGAs and Microcontrollers [7]

As with many embedded designs, also our application requires an FPGA next to a microcontroller (MCU). The FPGA can be used to implement anything from glue logic to custom IP, to accelerators for computationally intensive algorithms. By taking on some of the processing tasks, FPGAs help to improve system performance, thereby freeing up the MCU from cycle-intensive tasks. FPGAs also provide excellent performance characteristics and lots of flexibility to accommodate changing standards.

There is two possible implementations of MCU plus FPGA design: putting a soft MCU core into the FPGA logic structure or using a standard product MCU with a discrete FPGA. The first solution can be expensive and this is especially true when a 32 bit ARM based core is used.

This solution cannot be implemented in our application because the MCU that we have decided to use, belongs to the 32 bit ARM based core family and so the second solution proposed was chosen.

Interfacing FPGAs and microcontrollers can be a true challenge because both are not designed to communicate with each other efficiently. The problem that can be encountered are listed below:

FPGA does not have any dedicated module to communicate with MCU and so it must be designed.

Second, transferring information between MCU and FPGA usually requires cycles on the MCU bus and hardware resource (PIO or EBI) used to make the transfer.

Since the communication between MCU and FPGA is asynchronous, special attention must be given to resynchronize the MCU to the FPGA clock domain.

In the FPGA, the module must be designed to avoid metastability problems.

This last problem can be solved using some techniques. The paper “Design of High-Speed Parallel Data Interface Based on ARM &

FPGA" (Daode Zhang) [8] is an interesting study, in which two techniques against this problem are explained.

The first solution is the Double trigger method that consists to use two or more flip-flops in cascade. The Figure below shows the principle of work.

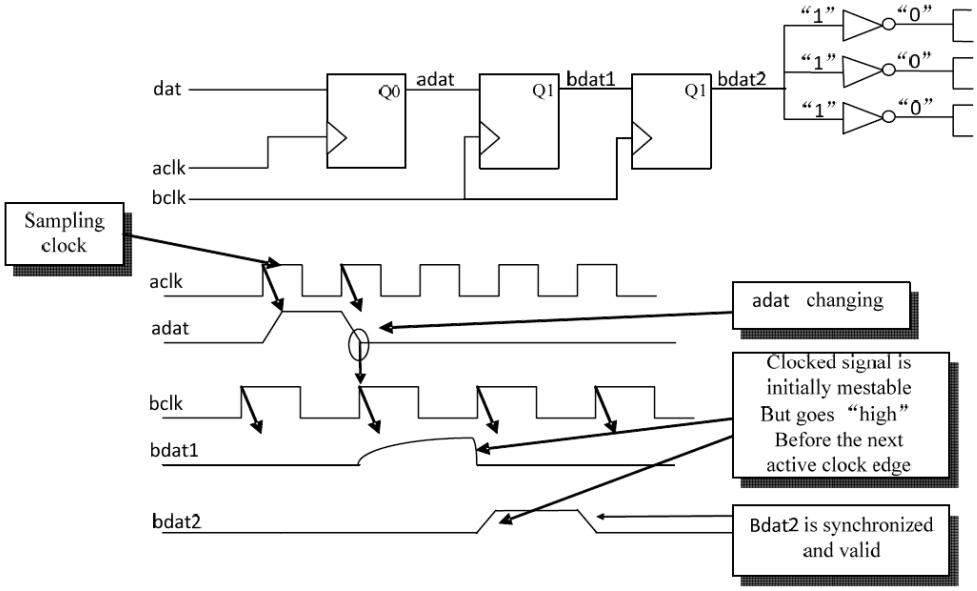


Figure 2.1: Diagram of Double trigger method

In Figure 2.1, the signal called adat does not respect hold time and the output of the flip-flop (bdat1) is metastable (Assume a value that is neither 0 nor 1). This represents a problem for the logic circuits that receive this signal as input because it can be seen as a 0 for some and as a 1 for others. It is a dangerous problem in the case of an FSM because it can take it in a not-allowed state.

There is a time, called resolution time, after which the metastable signal becomes stable but it has no relation with the input signal. When the metastability problem occurs, the flip-flop output assumes a value between 0 and the supply voltage (Vdd), since the asynchronous input signal does not respect Setup/Hold time. This value assumed by the flip-flop output determines the stable value assumed after the resolution time.

The double trigger technique can solve the metastability problem. In Figure 2.1, adat does not respect the hold time and produces a signal bdat that is metastable. It would be a problem but using another flip-flop, bdat has enough time to assume a stable value. At the next clock rising edge, bdat will respect the setup time and bdat2 will not suffer from metastability. This solution can effectively reduce the transmission of the metastable state.

However, the double trigger method cannot guarantee the stability of the second flip-flop output with absolute certainty.

The reason is that the Resolution time depends on the voltage value assumed by the metastable signal.

To reduce the possibility of failure, it is possible to use more than two flip-flops to increase the mean time between failures (MTBF). Hence, this method cannot get rid of the possibility to generate output error but it is a good technique to implement the synchronization state which requires little consumption of logic to avoid metastability.

The second method described proposes to solve metastability problems consist of using FIFO or DRAM. Writing asynchronous FIFO or DRAM is equivalent to put the data into a buffer and give them enough time to stay stable.

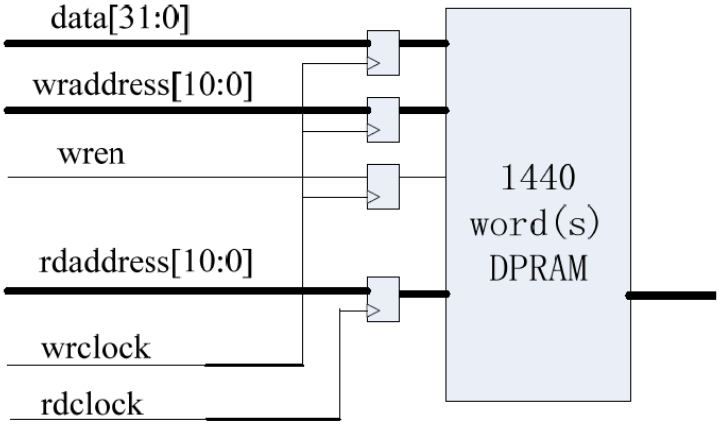


Figure 2.2: The internal structure of the DPRAM

This method is surely better than the previous one but it has two problems.

1) The system has to avoid reading and writing at the same time. In the particular case showed in the paper, the DRAM used has independent write and read data bus but the number of pins used increases.
2) The amount of logic used is greater than the one used in the previous case. This can be partially solved using FPGAs that have DRAM IP that use the least logical resources to get the most optimal performance.

The FPGA can be interfaced to the MCU in two different ways: programmable I/O (PIO) or external bus interface (EBI), if available;

The approach to use depends on the end application and the desired result.

2.1 PIO Interface

Belong to this family, all the interfaces that use programmable I/O to interface FPGA and MCU.

In particular, they can be divided into two sub-families:

- Serial Interface
- Address/data bus interface.

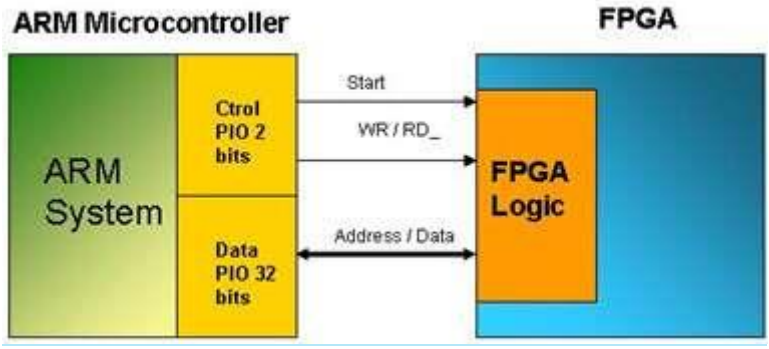


Figure 2.3: PIO interface to FPGA.

These two classes of interfaces represent opposites in terms of performance. Indeed, while the Address/data bus interface allows obtaining faster communication thanks to the parallel transmission but it needs more hardware resources (PIO). In addition, it is easier to implement in the FPGA because it does not require the implementation of a module that emulates a protocol.

On the other hand, a serial interface allows to use of a minus number of PIO but at the same time, the transmission is serial and so slower than the previous one described. Furthermore, it requires a module in FPGA that emulates the serial protocol chosen, and therefore, it requires the implementation of a complex FSM.

The choice between these two types of interface depends on the needs. An example where both the interfaces are implemented depending on the needs is the "8051 microcontroller to FPGA and ADC interface design for high speed parallel processing systems – Application in ultrasound scanners" (J. Jean Rossario Raj) [9]. This paper is a study on the different methods used for interfacing between the microcontroller and parallel processing devices such as FPGAs and data converters. The paper also compares and studies the best algorithm for different implementation conditions.

As is possible to see in Figure 2.4, in this application the Microcontroller interfaces with different devices, and to do this, it uses different types of interfacing according to the needs.

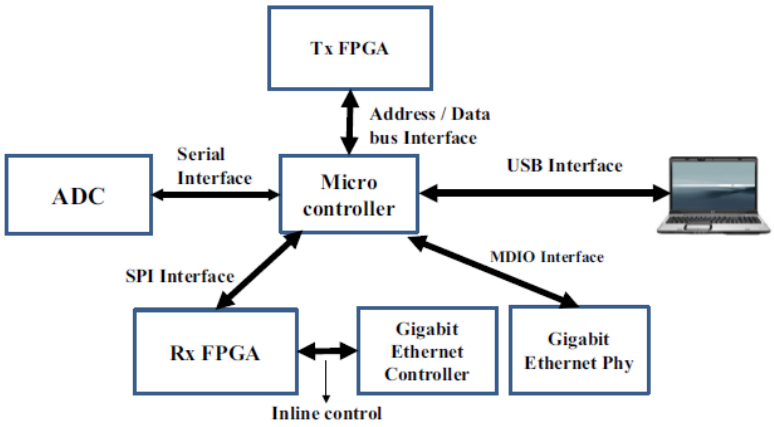


Figure 2.4: Block schematic of the experimental setup

We are interested in the two interfaces with FPGA and we will focus on them to understand the different types of implementation and how they work.

The microcontroller to transmit side FPGA (Tx-FPGA) interface was implemented using the address/data bus method as shown in the Figure above. This interface method was chosen because both the devices have a sufficient number of pins available and to make the program logic simple.

The interconnection between the FPGA and the microcontroller used 8-bit address bus, 8-bit data bus, read, write, and CS as shown in Figure 2.5.

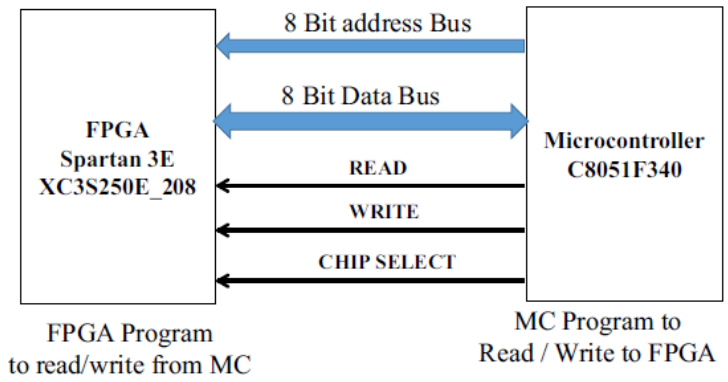


Figure 2.5 Block schematic of the microcontroller – FPGA address/data bus interfacing.

The address bus is simplex (From micro to FPGA) while the data bus is duplex. The low logic level in the read or write pin indicates that the proposed operation is read or write.

The flow chart of the microcontroller program for reading and write operations are shown in Figure 2.6:

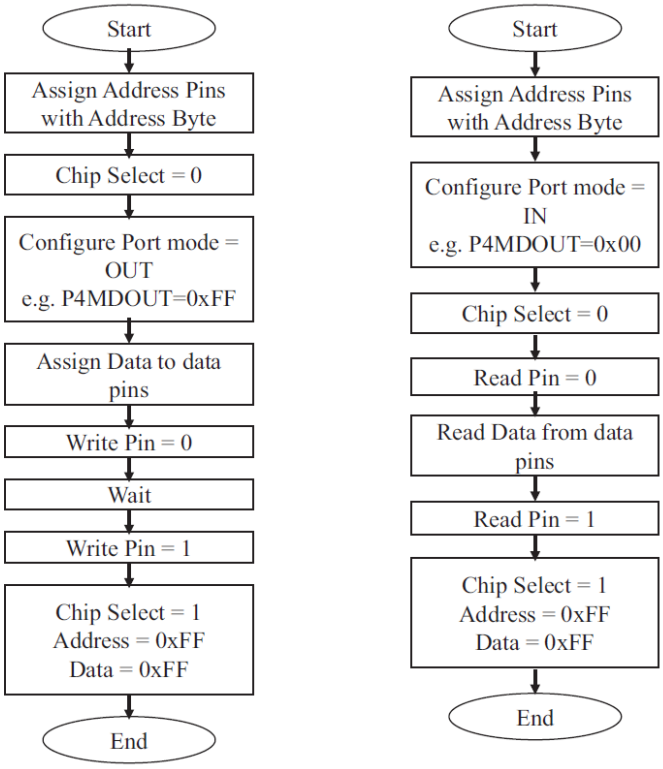


Figure 2.6: Flow chart of microcontroller firmware for address/data bus interfacing – Write and Read operations.

For both communications, there is a start and assignment of the address pins with the address data. Then, the chip selection of the device that we want to read or write is set low and the data pins are configured as an output (write) or input (read).

In the case of writing, the data we want to write is assigned to data pins, and the write pin is set equal to zero. A wait time is respected to send the Data assigned. After this operation, the write pin is set high and the communication ends.

In the case of reading, a pin of reading is set equal to zero and the Data pins are assigned to a variable called Read Data. After the operation of reading is complete, the read pin is set high and communication ends.

As it was said previously, a module in the FPGA is necessary to understand the commands coming from the microcontroller. For this reason, a VHDL module is implemented in the FPGA and a flow chart that explains how it works is presented in Figure 2.7.

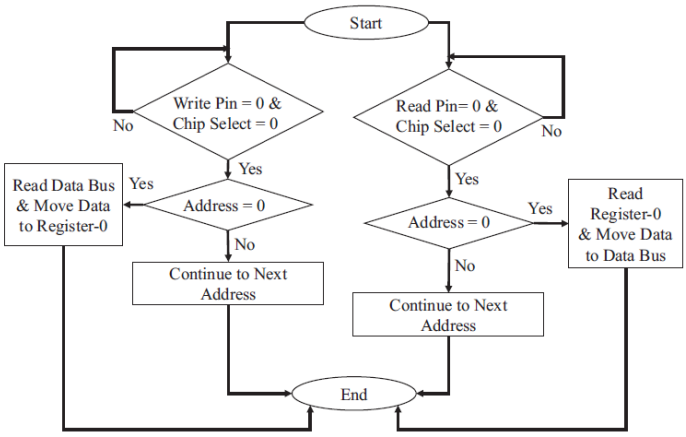


Figure 2.7: Flow chart of FPGA program for address/data bus interfacing.

After the start, two conditions are continuously checked. to understand if the microcontroller is asking for a read or write operation and if it wants to talk with this particular device (CS of the device equal to zero).

After this check, the Address sent is checked and compared with all the register addresses of the Memory on the FPGA, until it matches. It starts to compare from the address zero so the communication time is not always the same but depends on the selected address. When it matches, in the case of a reading operation, the system moves the data contained in the selected register to a data bus. Instead, in the case of a writing operation, it moves the data from Data Bus to the selected register.

On the receiving side, FPGA does not have enough IO pins available and therefore it is not possible to use the previous type of interface. Therefore, a serial interface that uses the SPI bus has been implemented.

The block schematic of the microcontroller to Rx-FPGA interface is presented in Figure 2.8.

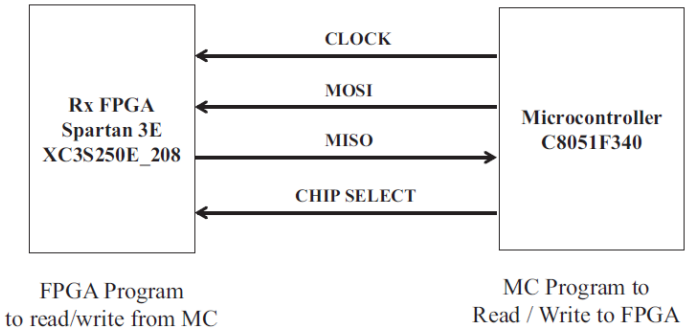


Figure 2.8: Block schematic of the microcontroller – FPGA SPI bus interfacing.

In this method, the microcontroller is acting as the master while the Rx-FPGA is the slave. The microcontroller program flowchart for SPI is given in Figure 2.9.

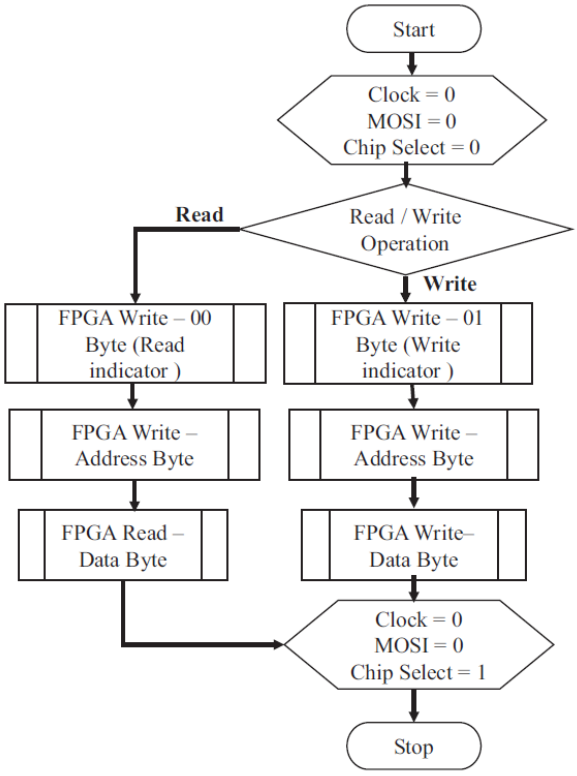


Figure 2.9: Flow chart of microcontroller firmware for SPI bus interfacing towards FPGA.

After the start, the clock line, the MOSI line, and chips select are set to zero. In the read/write (RW) operation, one byte of RW flag is written to the FPGA indicating the type of operation. Furthermore, in the writing operation, address and data bytes are written while in the reading operation, address byte is written and data are read from the MISO pins. At the end of the communication the clock line and the MOSI line are set low and the chips select is set high. The operation of the byte read and byte write are managed through two different subroutines described in Figure 2.10.

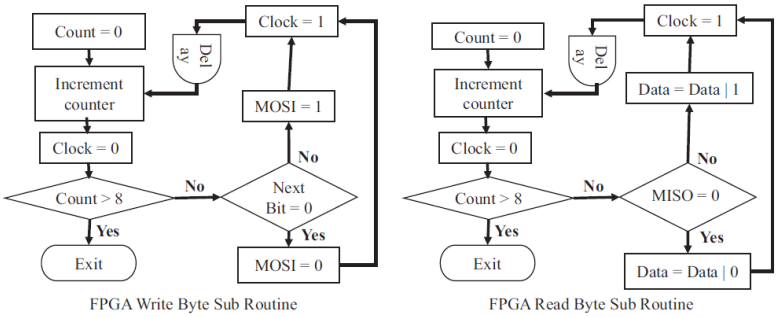


Figure 2.10: Flow chart of microcontroller firmware for SPI bus interfacing towards FPGA.

In these subroutines, the time duration for toggling of the clock pin is controlled by a delay. In each clock, one bit is moved to the MOSI pin during the write operation and a bit is read from the

MISO pin during the reading operation. This cycle is repeated for 8 bits each time one of the two subroutines is called.

Furthermore, in this case, a module in the FPGA is needed to understand the commands coming from the microcontroller. For this reason, a VHDL module is implemented in the FPGA.

When CS becomes active low, FPGA reads from MOSI pin bit by bit during the rising edge of the microcontroller clock. When FPGA completes reading one byte, the LSB provides information on the type of operation. In the case of reading another byte is acquired and saved in a logic vector for the temporary storage of the address value. FPGA moves the data from the register at the address location to the MISO pin bit by bit.

In the case of writing, 2 bytes are read (address and data). Then, FPGA writes the data value to the register at the address location.

2.2 Interfacing through the External Bus Interface (EIB)

Many 32-Bit microcontrollers have an EIB module, which is designed to transfer data between external devices and the memory controllers on ARM-based devices. These memory controllers are capable of managing different types of external memory and peripheral devices, such as SRAM, PROM, EPROM, EEPROM, flash, and SDRAM.

It can be used also to interface with the FPGA as long as the FPGA can handle the predefined memory interfaces. The SRAM is preferable for FPGA communication because it is simple to implement.

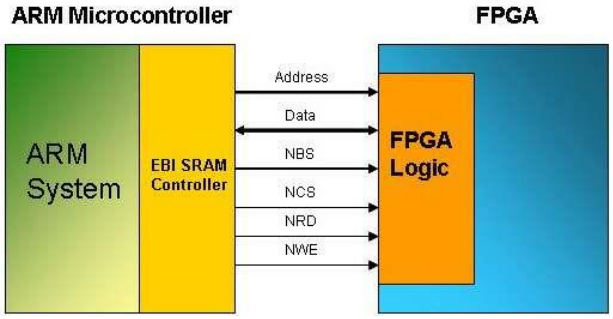


Figure 2.11: EBI-SMC interface.

As the PIO interface, a module that understands the SRAM timing and can produce a response back to the microcontroller must be implemented in the FPGA.

An EBI interface is faster than PIO because it has its own dedicated I/O and most of the signals are parallel. In this case, the FPGA can represent the bottleneck of the communication,

because if it is slow to respond, the EBI's speed advantage could be lost.

Part III
The SpaceRadMonMx

Chapter 3

3 The SpaceRadMonMx

The SpaceRadMonMx instrument is a CERN design based on the miniaturization of version 6 of the RadMon, which has been used inside the CERN irradiation facilities since 2013. The SpaceRadMonMx, thanks to the carefully selected sensors, measures the Total Ionizing Dose (TID), the number of Single Event Upsets (SEU), and Single Event Latchups (SEL) that provide sufficient information to characterize the space environment through the HEH fluence and the spectrum hardness factor.

3.1 SpaceRadMonMx structure

The SpaceRadMonMx is based on a system consisting of an FPGA and an MCU that manage all the sensors on the board. The structure can be summarized in Figure 3.1.

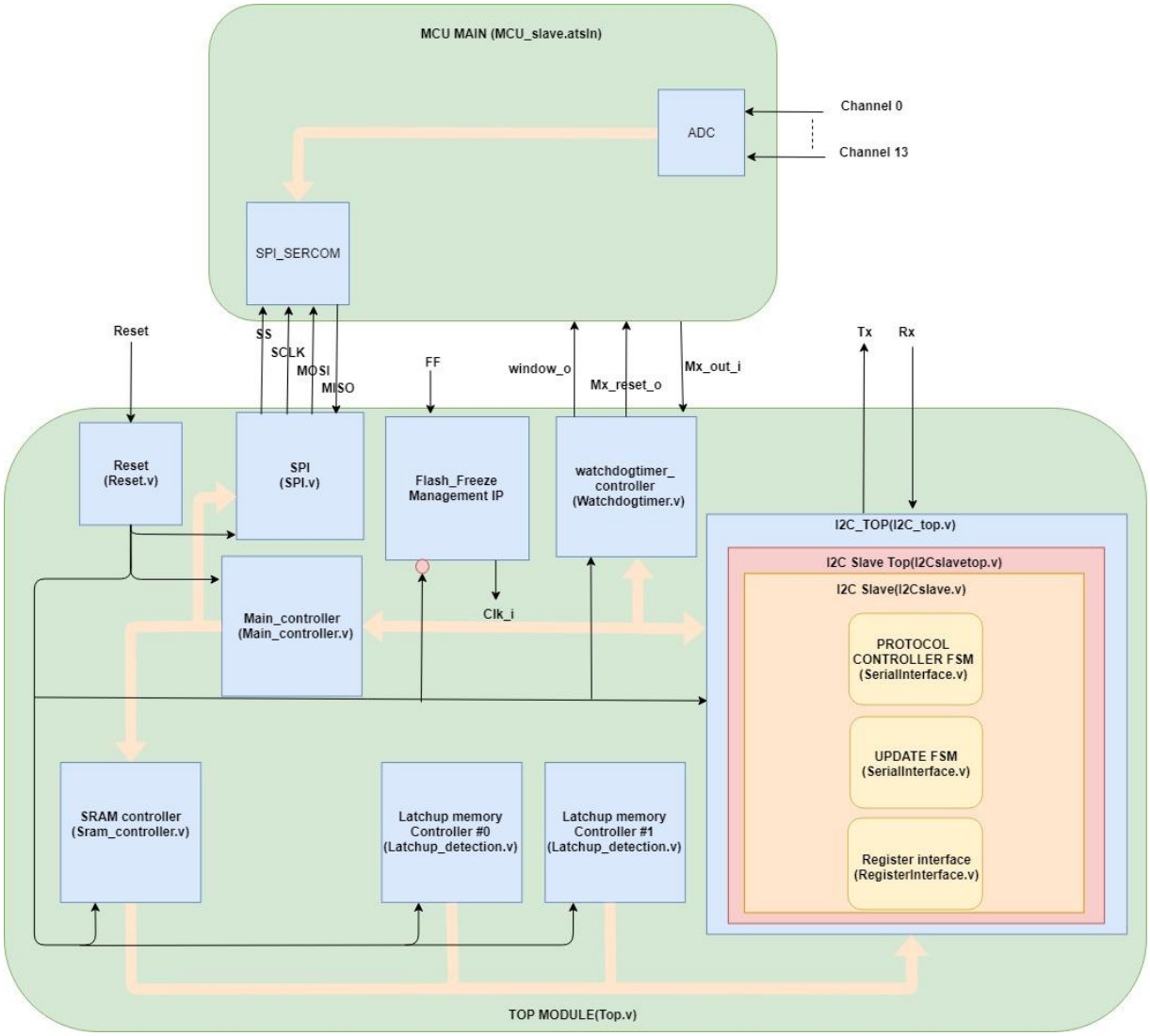


Figure 3.1: SpaceRadMonMx block diagram.

There are several blocks in the design that are interconnected, allowing the proper functionality of the SpaceRadMonMx. Compared to the old version of SpaceRadMon, the ADC was replaced with a Microcontroller (MCU). Communication with the Microcontroller is performed via SPI protocol. Communication with the external world is performed through an I2C interface. The FlashFreeze management IP manages the system's power consumption and allows it to enter in low power mode.

The Watchdog timer Controller manages the microcontroller and resets it in case of problems.

The other blocks are responsible for the control of the sensors on the board. More specifically, the SRAM controller and Latchup memory controllers are responsible for the acquisition of the data coming from the Cypress SRAM and two BSI SRAMs.

In the next sections of this thesis work, firstly the SpaceRadMonMx detector system will be presented with all the blocks that are part of it. Then, the microcontroller configuration and the module that interact with it, and the flash freeze technology. Finally, the I2C communication and the implemented modules allow the system to transmit the data to the outside.

3.2 The SpaceRadMonMx detector system

The SpaceRadMonMx system allows measuring the TID through Radfet, the number of Single Event Upsets (SEU) and Single Event Latchups (SEL) using a Cypress memory and two BSI memories.

3.2.1 TID-RadFET

RadFETs are p-channel MosFETs that are optimized for monitoring the TID on electronic devices. The RadFET dosimeters measure the build-up of oxide positive trapped charges in the gate silicon oxide layer of the transistor. This positive trapped charge is responsible for the threshold voltage shift (ΔVth) of the transistor, which is measured at a constant drain to source current (Ids). Therefore the growth of the voltage Vth is a function of the dose. The variation of the Vth at increasing dose determines the sensitivity of the RadFETs. The sensor reaches the saturation condition when the threshold voltage Vth does not vary anymore at increasing doses. The sensitivity and the saturation conditions depend on the biasing of the gate and the oxide thickness.

The sensitivity of the RadFETs is higher for thicker oxides, respectively thicker RadFETs also reach saturation at lower dose levels.

The ADC of the microcontroller reads the Vth of the Radfet.

3.2.2 Single Events Upset – SRAM memories

When an SRAM memory cell is exposed to ionizing radiation, the energy deposited by a single ionizing particle may be sufficient to change the state of the memory cell. The registers of these memories are initially written with all 0s (pattern 0) or 1s (pattern 1). After this initialization, every n minutes the memory is controlled. Supposing a pattern 0, those are the possible operation made for every register:

1) If the bit of the register is all 0s the next register is checked.
2) If one or more bits are not 0, the number of SEU is increased with the number of changed bits and the next register is checked.

Knowing the number of SEEs and the cross-section is possible to obtain the fluence.

The SRAM controller executes these reading and writing operations. In particular, the module is described in figure 3.2

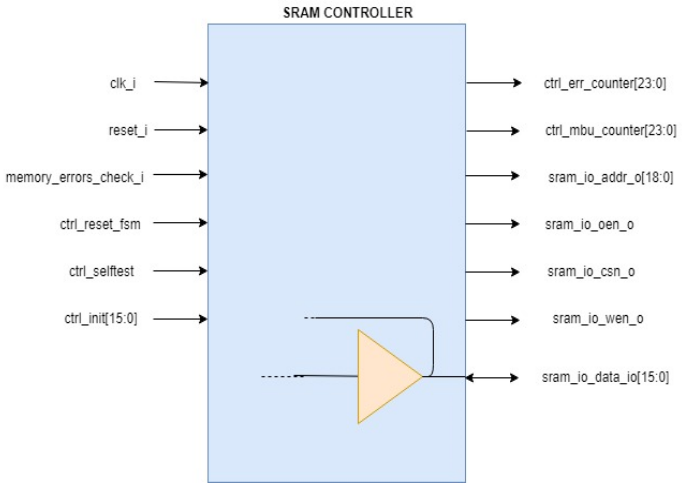


Figure 3.2: SRAM Controller block diagram.

It is made by a large finite state machine (FSM). This FSM is composed of two main states: one to write the registers with the desired value and one to read the number of MBUs and SEUs occurred. Each of these two states is divided further into sub-FSMs to drive properly the different control signals (output enable, chip enable, etc.). In Figure 3.3, the SRAM controller FSM is depicted.

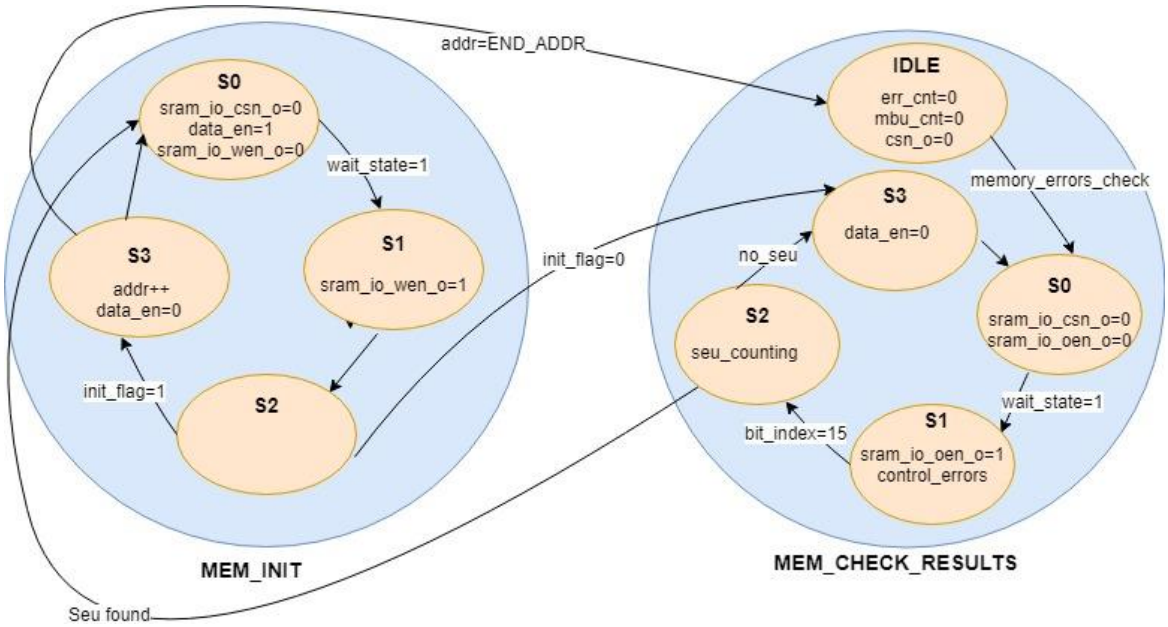


Figure 3.3: SRAM Finite State Machine

The reset state of the FSM is MEM_INIT-S0. In this state, the memory is enabled, setting the chip select (csn) low. Sram_io_csn_o, is the output of a flip-flop: it will be updated the next rising edge of the clock. The sram_data_o (the value that we want to write in the selected register) is written in the memory, setting the data to enable (data_en) high and write enable (sram_io_wen_o) low. A counter will change the state to MEM_INI-S1 when the counting value will be equal to WRITE_WS (1).

In MEM_INIT-S1, the sram_io_wen_o is disabled and the state changes to MEM_INIT-S2. In this state there is an important condition: if ctrl_init_flag is different from the reset value (1), the next state of the FSM will not be MEM_INIT-S3 but MEM_CHECK_RESULTS-S3. If this condition is not respected, the FSM reaches MEM_INIT-S3 where the input of the memory is set to Z (data_en = 0) and the address is increased by 1 to write in another register the next time. The next state will be MEM_INIT-S0 and the cycle will be repeated. This is true only the first time that the memory is written because after the first writing cycle there will be no more a continuous writing cycle.

When the address is equal to END_ADDRESS, the control of SEUs and MBUs will start changing the FSM state to MEM_CHECK_RESULTS-IDLE. The address is reset to the start value and ctrl_init_flag is set low. In MEM_CHECK_RESULTS-IDLE csn is set low and the counter of SEUs (seu_counter) and MBUs (mbu_counter) are reset to 0.

Only when memory_errors_check is changed by the main_controller, the FSM state becomes MEM_CHECK_RESULTS-S0.

In MEM_CHECK_RESULTS-S0 csn is set high and the SRAM output is enabled (sram_io_oen_o = 0). A counter (wait_state) will change the state to MEM_CHECK_RESULTS–S1 after the counting value is equal to READ_WS (1). When this condition is verified, the counters wait_state and wseu_counter are reset to 0, while the selected register of the memory is read.

In MEM_ CHECK_RESULTS–S1, the SRAM output is disabled. The 16 bits of the register (sram_data_i) are controlled (one for every rising edge of the clock) and wseu_counter is increased every time the bit is different from the expected one. When the 16th bit is controlled, the state becomes MEM_ CHECK_RESULTS–S2.

In MEM_CHECK_RESULTS–S2 the SEUs and MBUs are counted. If wseu_counter is more than 1, mbu_counter is increased by 1. At the same time, if wseu_counter is different from 0, seu_counter is increased of wseu_counter, and the state of the FSM changes to MEM_INIT-S0. This is an important passage of the FSM because, during it, only this register is written. Indeed, when MEM_INIT-S2 is reached, the condition on ctrl_init_flag is true and the FSM will reach the state of MEM_CHECK_RESULTS–S3 where data_en is set to 0 and the address is increased. If the end address is reached, ctrl_err_counter is updated adding to its old value, the one of seu_counter. It is the same for ctrl_mbu_counter. The seu_counter and mbu_counter are also reset after the saving.

The next state is another time MEM_ CHECK_RESULTS – IDLE and the cycle will restart only when memory_errors_check will be changed by the main_controller another time.

3.2.3 Single Events Latch-up – SRAM memories

The severe damage caused by latch-up events on CMOS devices is enough to compromise the result of a space mission or the functioning of equipment placed in high radiation areas, such as in high energy accelerators. To evaluate the SELs, the SpaceRadMonMx uses two brilliance SRAM components. In particular, a circuit of conditioning is made to control SEL events.

The FPGA controls an enable of a DC-DC converter by a module that supplies the BSI SRAM for a certain amount of time. The Icc current is monitored using the INA146 difference amplifier.

A low resistance value was chosen for R, to reduce the impact that the voltage drop over the resistor would have on the biasing of the component, and therefore on its latchup sensitivity. The output VLAT is compared with a voltage reference VREF using an LM124 operational amplifier in a voltage comparator configuration. The final output Vout is sent to the processing unit which records the latchup event.

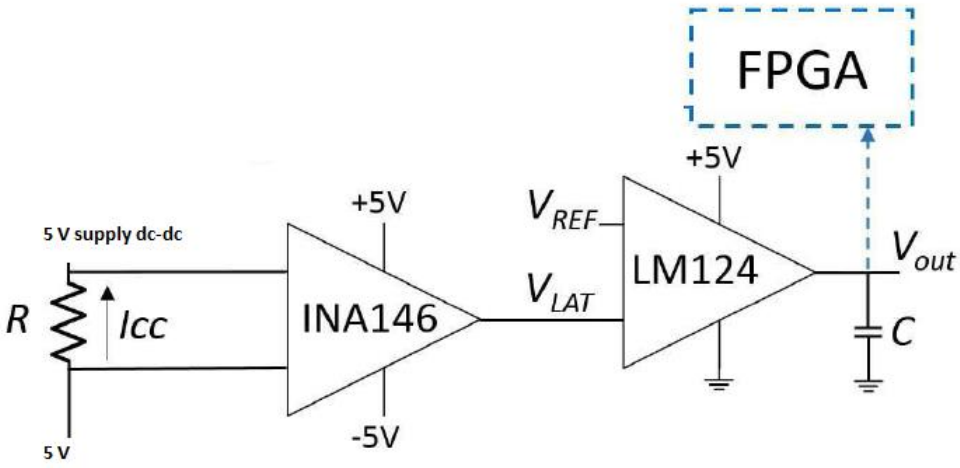


Figure 3.4: Detection Circuit for SELs.

These operations of enabling the DC-DC and output control are made by a module called Latchup Detection Controller.

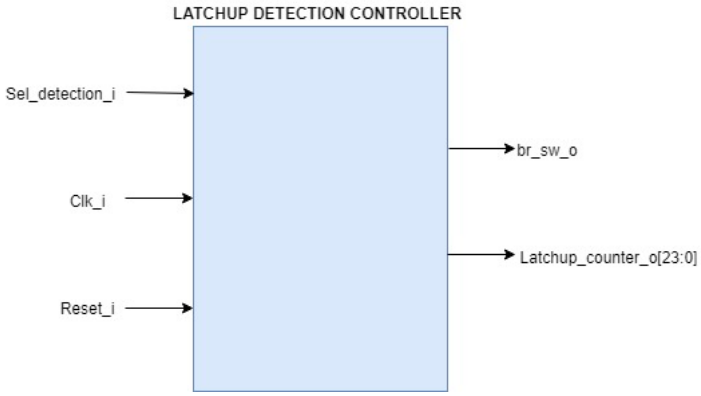


Figure 3.4: LATCHUP Detection Controller block diagram.

This module implements an FSM that manages the operations described.

This FSM is depicted in Figure 3.5.

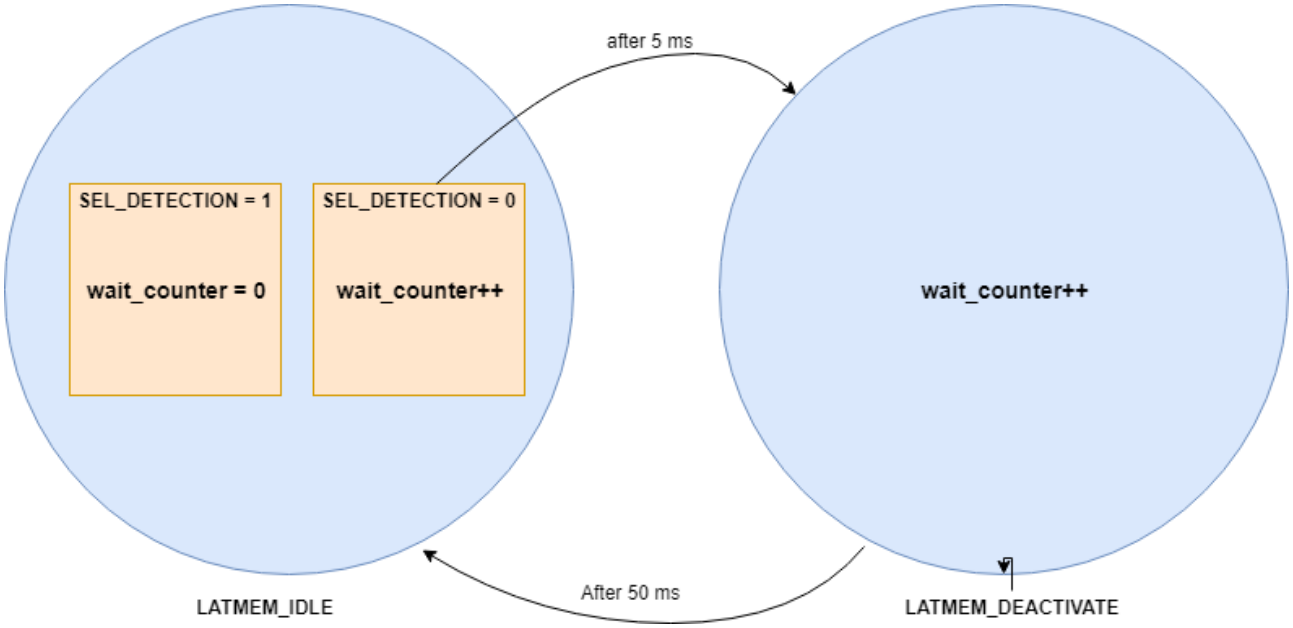


Figure 3.5: LATCHUP Detection Controller FSM.

The Latchup Detection Controller is functioning on a simple scheme that is controlled by an FSM of two states. The reset state is the LATMEM_IDLE.

In this state normally there is only a reset of a counter that time the operations and control on an input bit (SEL_detection_i) that designates the occurrence of a latch-up when it is pulled down. For as long as the SEL_detection_i signal is pulled down, a wait time counter (wait_counter) counts the number of clock cycles. When the number of clock cycles reaches the equivalent of the HOLD_TIME (after 5ms), it designates that a latch-up has been detected, A SEL counter is increased (latchup_counter_o), the memory is deactivated (br_sw_o Low) to recover it and the FSM reaches the LATMEM_DEACTIVATE state. The time counter is reset to 0.

If the counter does not reach the HOLD_TIME before the SEL_detection_i pin is asserted, the latch-up is not detected and the time counter is reset to 0.

In LATMEM_DEACTIVATE, the time counter starts counting, until it reaches the equivalent of CUT_TIME (after 50ms). During this period, the memory remains de-activated (powered down). After this period, the memory is re-activated, the time

counter reset to 0 and the FSM returns to the LATMEM_IDLE state where it waits for a latch-up to occur.

3.2.4 Main Controller

The main controller is a module responsible for the SPI start communication and the SRAM start reading.

It implements an FSM of 2 states:

1) Idle state: nothing happen till the power on of the system coming from the outside (I2C)
2) Nominal state: several counters manage the different sensors of the board.

The counter is described below.

SPI_counter: every second from the moment that the FSM has reached the NOMINAL state, it is reset to 0 and enables the SPI module that requires data from the MCU SLAVE.

Sram_counter: every 240 seconds from the moment that the FSM has reached the NOMINAL state, it is reset to 0, and sram_scan_o is set high. This bit allows the SRAM FSM to reach the MEM_CHECK_RESULTS-S0 state when is in the MEM_CHECK_RESULTS-IDLE state.

3.3 MCU configuration

To improve the SpaceRadMonMx instrument, the ADC has been replaced by a microcontroller that has its own ADC.

The reason for this replacement:

1) Low power.
2) The previous ADC has a reading with more bits than what was necessary.
3) ADC is radiation tolerant up to 400 Gy.

This improvement requires a new communication interface between FPGA and the microcontroller. There are several ways to interface these two devices. The SPI communication was chosen for three reasons:

1) Cheap.
2) The application does not need the data rate provided by a parallel interface like EBI.
3) Only 4 GPIOs are required.

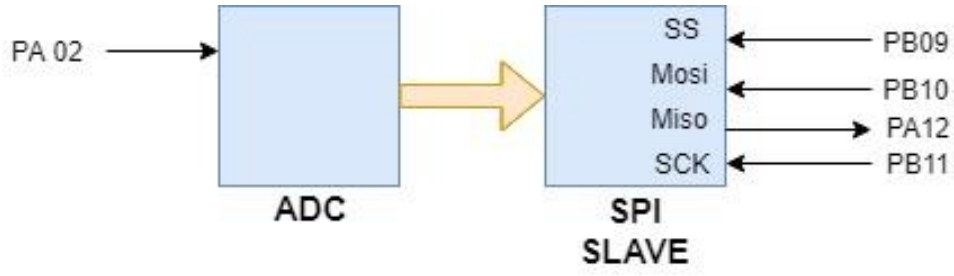


Figure 3.6: MCU system block diagram

In Figure 3.6, the system that we want to realize is schematized. As it is possible to see, the ADC acquires the analog input from a GPIO (PA02) and the converted data are sent to the FPGA via SPI.

3.3.1 SPI Configuration

The first operation performed was to configure the SPI and test it. Since the FPGA will request the data from the microcontroller, it must be configured as a slave. The configuration of the SPI slave is shown in Figure 3.7.

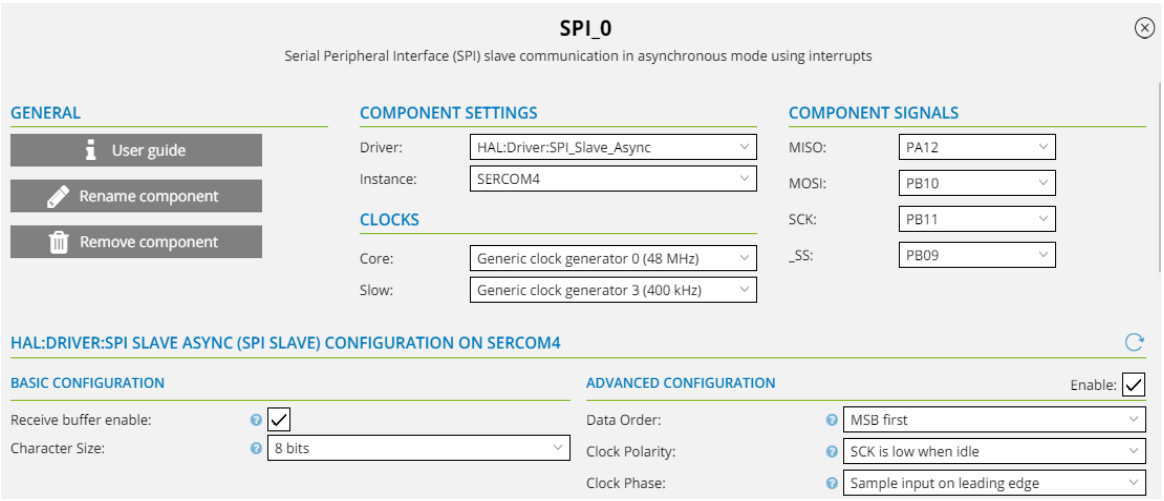


Figure 3.7: SPI slave configuration

For the configuration of the SPI slave, the 8-bits transmission was chosen and the MSB was set as the first sent. The address is not required in the communication. CPOL and CPHA have been set equal to 0 to have the sampling on the rising front edge of the clock and the values change on the MOSI and MISO lines when the clock is low. Therefore, the SPI has been configured to operate in mode 0 of Figure 3.8.

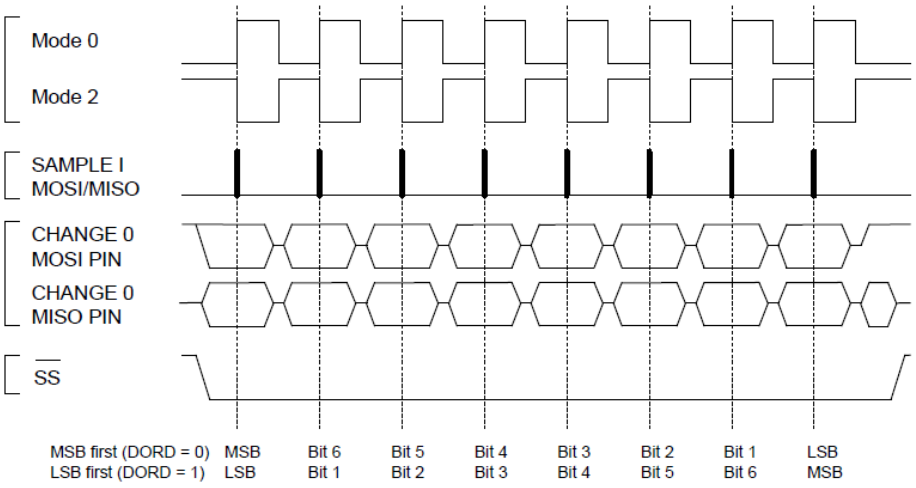


Figure 3.8: SPI Working Mode

3.3.2 SPI Test

To test the SPI slave, an SPI Master has been configured on another microcontroller. The chosen configuration is shown in Figure 3.9.

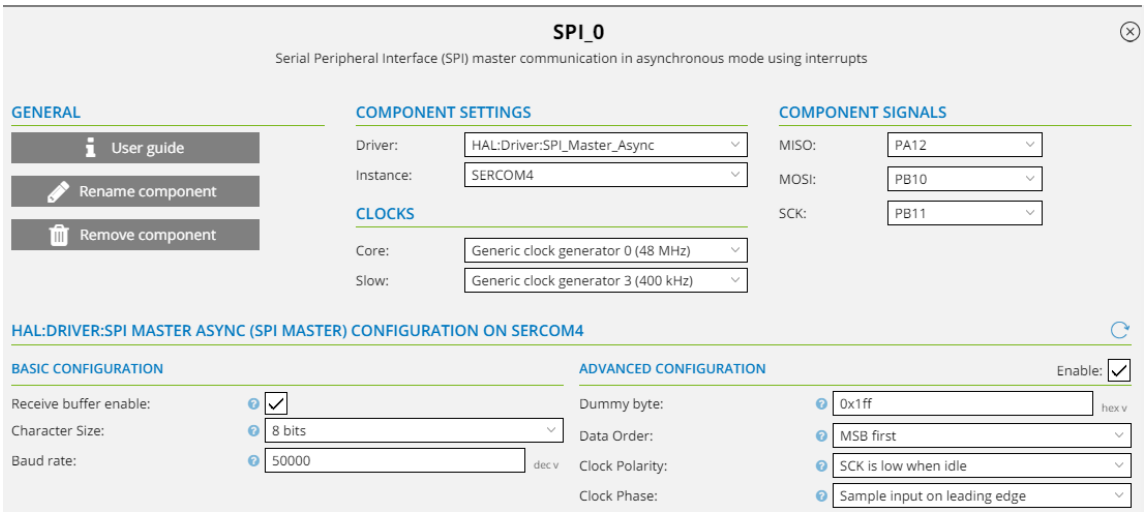


Figure 3.9: SPI Master configuration

The chosen configuration is coherent with that chosen on the SPI Slave.

For test reason, the UART was configured on the master to take the data received from the SPI transmission and send it to a Personal Computer (PC). The system created for test purposes is schematized in Figure 3.10.

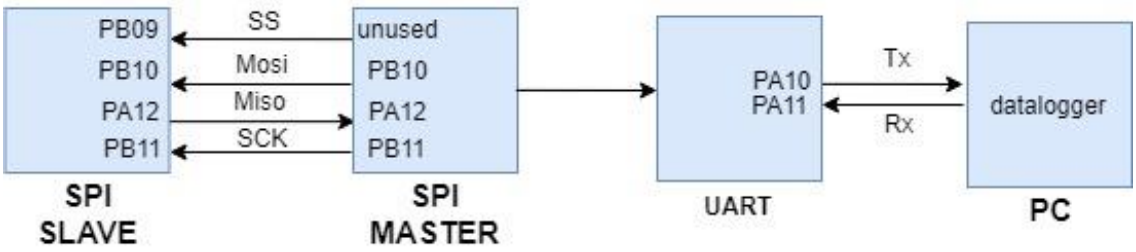


Figure 3.10: SPI Test system block diagram

Moreover, a buffer of constants was created on the slave and set as the output of the MISO lines in the case of SCK toggling by the master.

```
spi_buff_tx[6] = {0x22,0x33,0x44,0x55,0x66,0x00};
```

Figure 3.11: Buffer of constants used for test

The SPI master was configured in the firmware to request 7 bytes for each transmission. The reason for requesting an extra byte concerning the size of spi_buff_tx, is related to the buffer used by the microcontroller's SERCOM. This buffer contains a byte equal to zero before being filled with the first byte of spi_buff_tx and so, the first byte sent by the slave is 0. In the master firmware, this byte is discarded and the other 6 bytes are sent to the PC via the Tx line of the UART.

With this configuration, the system was tested with good results. They are shown in Figure 3.12.

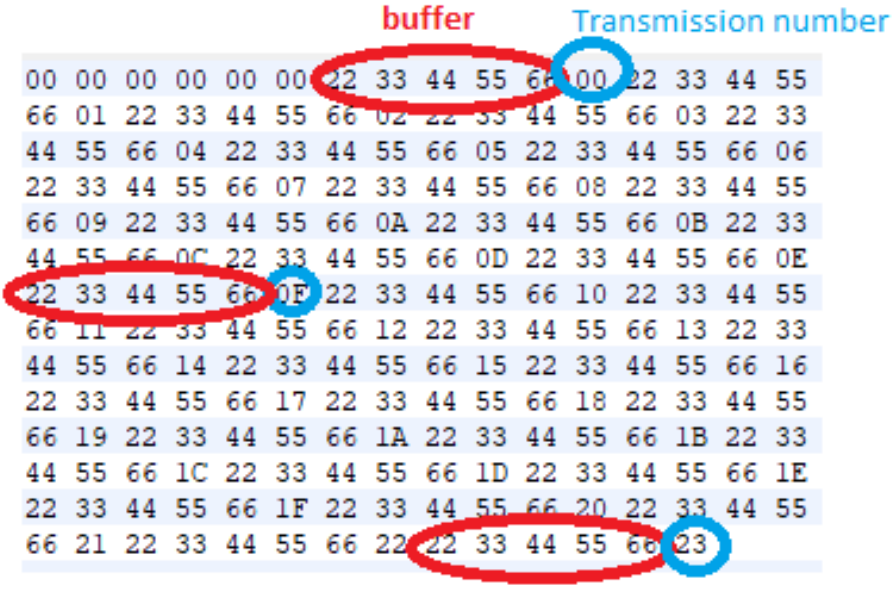


Figure 3.12: Capture of the Datalogger for SPI Test

As it is possible to see in Figure 3.12, the buffer of constants defined above is transmitted correctly (circled in red). To verify if the system continues to work properly in case of a change of spi_buff_tx and it is not repeating the first transmission, the last byte sent is increased by one for each transmission. It is possible to see that the last byte (circled in blue) is increased by one each time that a new transmission is made. It is also really important to know if some transmissions are lost.

Once that the SPI was tested, the ADC must be configured and characterize.

To characterize the ADC we need to know its sampling rate. This value is normally provided by the manufacturer and, on the datasheet of the microcontroller, there is a formula that gives it. The formula depends on the different working conditions: resolution, $frequency_{ADC}$, input acquisition mode (single-ended/differential), working mode (Free running / single-shot). So different working conditions were tested.

3.3.3 ADC Configuration

First, the ADC was tested in single-shot mode with the following setup.

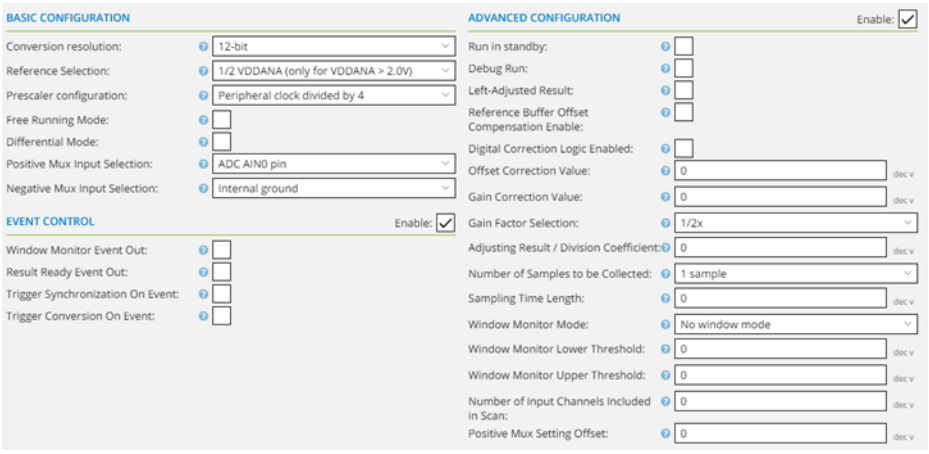


Figure 3.13: ADC configuration for single-shot mode

As it is possible to see in Figure 3.13, the resolution was chosen equal to 12 bit (max without the averaging mode) and the reference voltage equal to half of VDDANA (3.3 V). The gain factor was chosen equal to ½ because the voltages that we have to acquire are greater than ½ VDDANA. The acquisition mode is the default (Single-ended). The differential mode was not chosen because the input is always positive and we want to use the

maximum resolution. The differential mode uses a bit for the sign and so the max resolution is reduced to 11.

With this configuration, the ADC was tested and it was discovered that the formula provided by the manufacturer is not true. The sampling rate is that of the formula only if more than one conversion is carried out. The ADC can perform a single-shot conversion made of different acquisitions and provide as output an average value. In this case, the time between two consecutive conversions is that of the formula.

Therefore, the single-shot mode has been replaced with the free-running mode which is faster and more useful for our application. Thanks to this working method, it would no longer be necessary to ask the ADC to carry out a conversion because it will do it continuously. Unfortunately, the ADC with this work mode cannot be used without a DMA because it creates problems when is integrated with the SPI. The problem is related to some interrupts that are continuously called by the ADC when a conversion is done. These occur also when the MCU is serving the SPI's interrupt and they make the microcontroller not work properly.

For this reason, the DMA was introduced in the MCU configuration. The DMA saves the ADC outputs in a buffer of n elements. The DMA can transfer data between memories and peripherals and thus off-load these tasks from the CPU. In particular, it transfers a defined number of bytes and it performs this operation only when requested by the firmware. The DMA channel 0 was configured as shown in Figure 3.14.

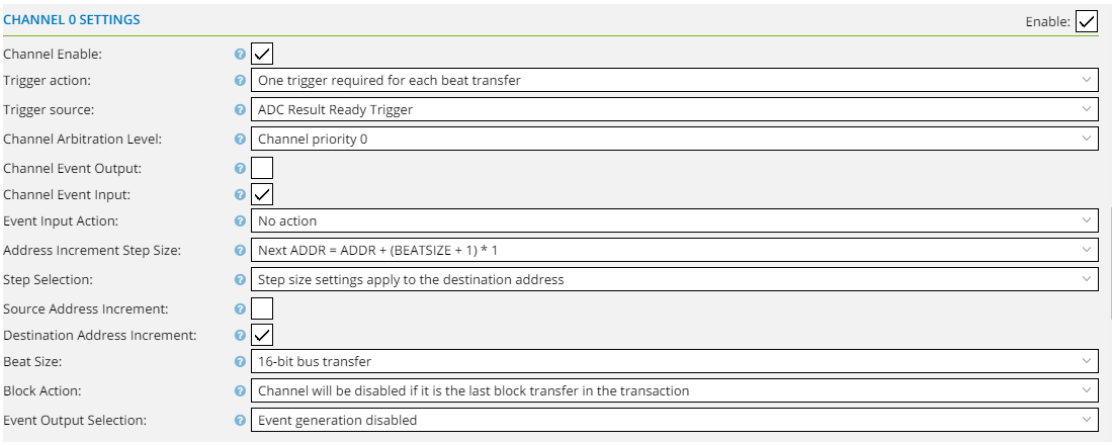


Figure 3.14: DMA channel 0 configuration

As is possible to see in Figure 3.14, channel 0 of the DMA was configured to get be trigger by the ADC Result Ready Trigger (RSRT). In particular, it transfers data of sixteen bits (the ADC result register is composed of 2 Bytes) and the destination address is increased by sixteen each time that there is a trigger event.

The ADC configuration was changed as showed in Figure 10.

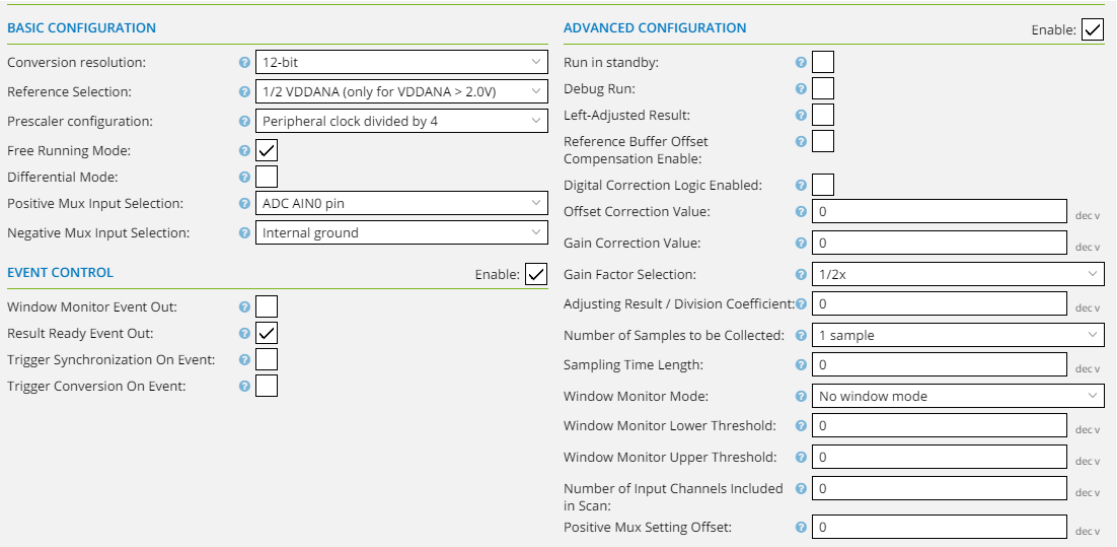


Figure 3.15: ADC configuration with DMA and free running mode

The Result Read Event Out was enabled so that the ADC can generate an event every time that a conversion is completed.

Thanks to these settings, the ADC and the SPI can work together and it is possible to measure the sampling rate.

3.3.4 Sampling Time Measure

As said, the manufacturer provides a formula that with these settings is the following:

$$Sampling\ time = \frac{7}{frequency_{ADC}} \tag{3.1}$$

The relation (3.1), considering a frequency of 250 kHz, provides a sampling rate equal to 28 µs.

To measure the sampling rate, three approaches were followed:

1) Using a Time;
2) Using a GPIO;
3) FFT analysis.

The first approach is to use a timer of the microcontroller that counts the time between two consecutive conversions. The second is to toggle the GPIO output every time whenever a conversion is completed. These two methods were implemented in the slave firmware. The structure of the firmware can be summarized in Figure 3.16.

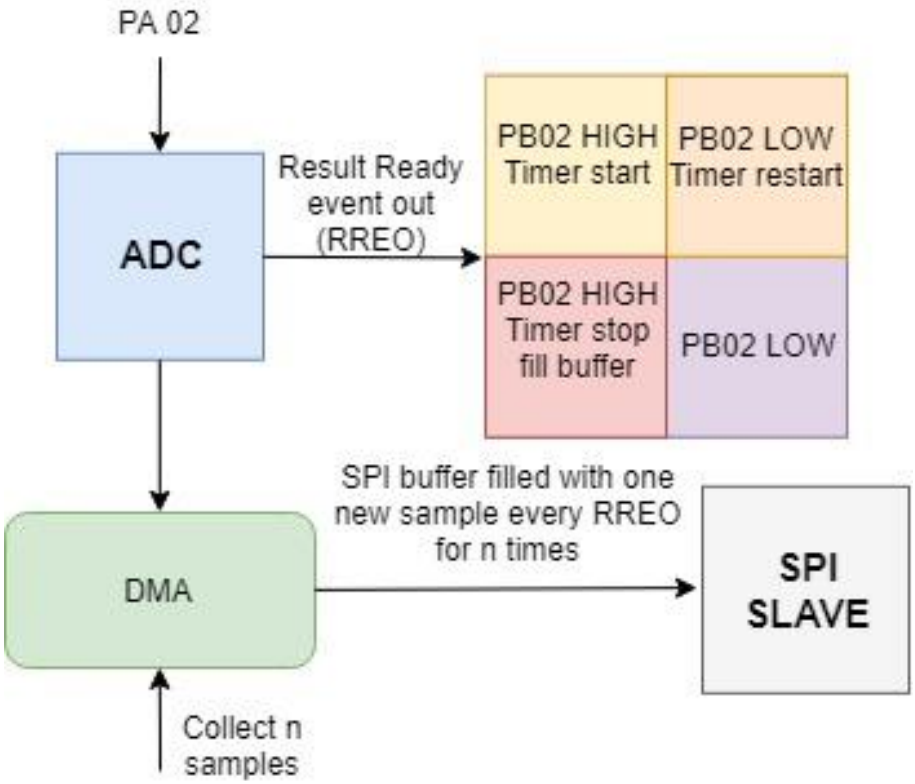


Figure 3.16: Sampling time monitoring system (Slave side)

As it is possible to see in Figure 3.16, every time there is the RREO one of 4 possible operations is carried out. The order of the operations is the following:

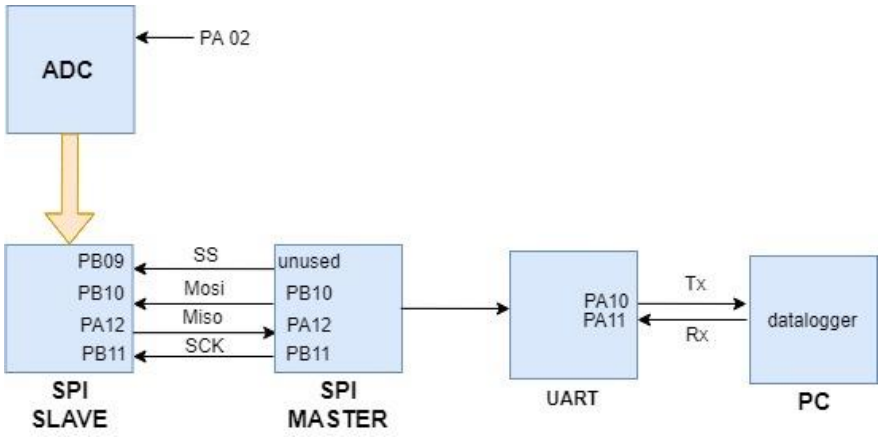
The GPIO selected (PB02) is set high and the timer count start

- PB02 is set low and the timer restart
- PB02 is set High and the timer stops
- PB02 is set low

Each time the interrupt occurs, only one operation is performed.

The timer count register must be synchronized before stopping the count or it will remain at the value zero. One was subtracted from the timer count because synchronization requires one clock cycle.

This count is added to the spi_buff_tx and it can be read using the previous system implemented for the SPI test and repeated in Figure 3.17.



`

Figure 3.17: Sampling time measure system block diagram

To monitor if the ADC is working properly, PA02 is connected to 3.3 V and one of the conversions is sent via SPI.

Figure 3.18 shows the data acquired by the data logger.

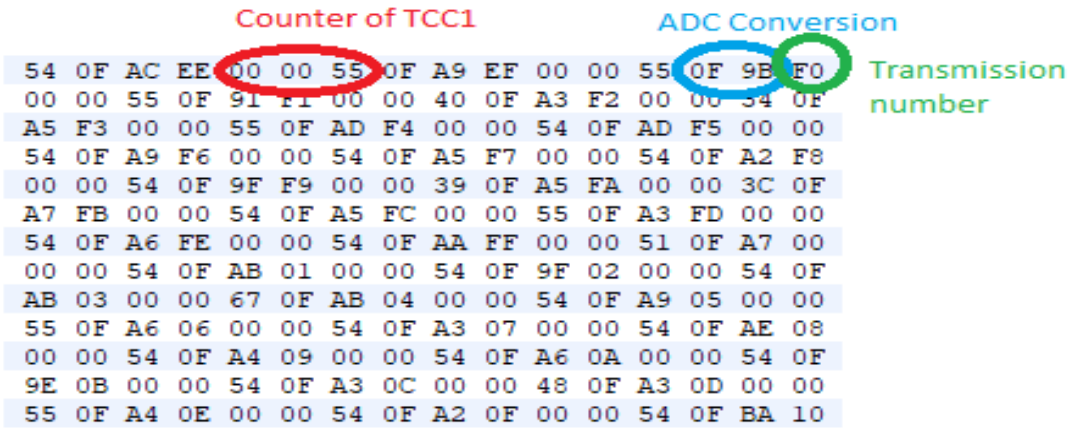


Figure 3.18: Capture of the Datalogger for sampling time measure

Also, in this case, SPI Master requires 7 bytes (the first is discarded). The first 3 bytes of the frame (circled in red) are the output of the counter. The fourth and fifth bytes are the ADC output and the last byte is the transmission number used to monitor that no transmission is lost.

Considering a population of 64 measures, the sample rate measured is shown in Table 3.1.

Parameter	**Value (µs)**
µ	28.12
σ	1.80

Table 3.1: Sampling rate measured using a timer

As is possible to see in Table 3.1, this measurement method provides a too high standard deviation but it gives us important information: the value provided by the manufacturer is only indicative.

As will be clear with the FFT method shown later, the manufacturer provides a sampling rate with a low resolution.

The second method described previously (GPIO), was measured with an oscilloscope. In Figure 3.19, it is possible to see the reconstructed wave of the GPIO output.

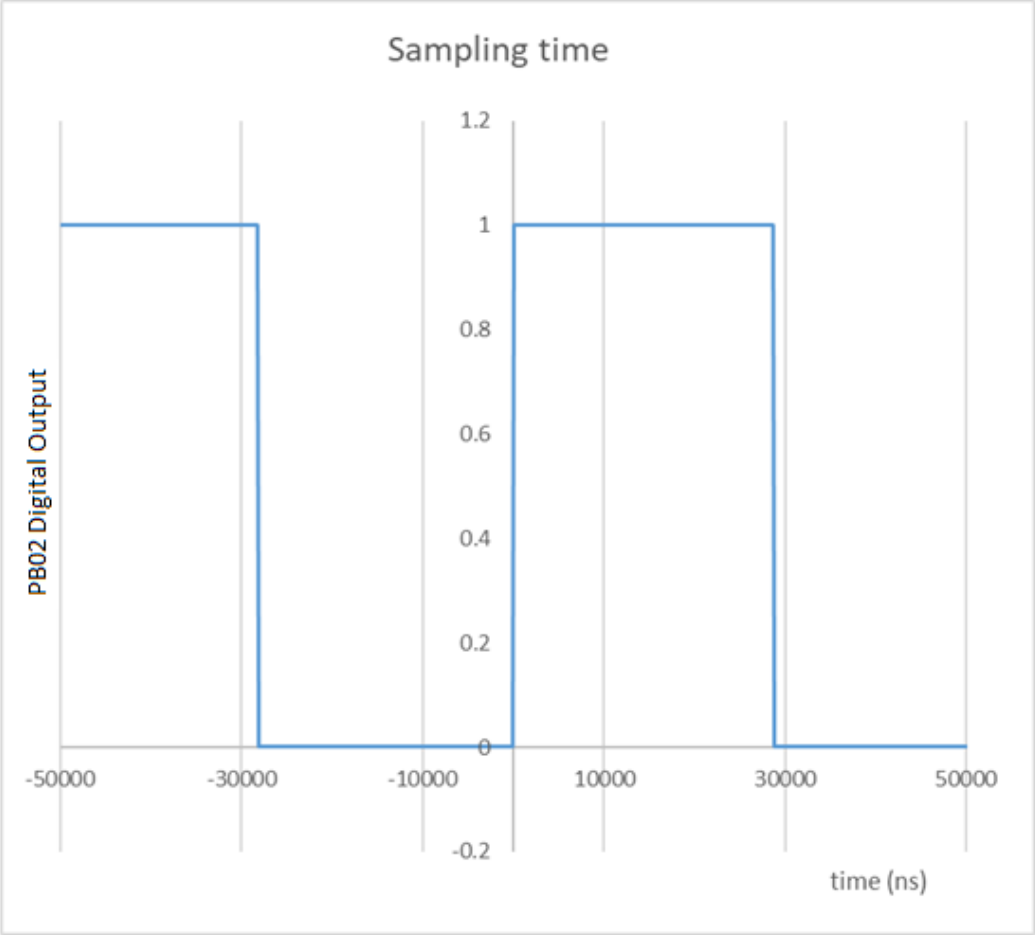


Figure 3.19: PB02 output wave

The high and low logic times of this wave were measured and saved. Considering a population of 64 measures, the sample rate measured is shown in Table 3.2.

Parameter	**Value (µs)**
µ	28.54
σ	0.32

Table 3.2: sampling rate measured using a GPIO

In this case, the standard deviation is lower than in the previous case but the range is already too big.

A third method was implemented to obtain a better measurement. It is based on coherent sampling.

If the sampling rate of the ADC is known, it's possible to choose an input sine with frequency N times lower than the sampling frequency.

$$\frac{f_s}{f_{in}} = \frac{N_{samples}}{M_{cycles}}; \qquad (3.2)$$

Considering $M_{cycles} = 1$ and assuming to know f_s, if a signal with an input frequency $N_{samples}$ times lower than the f_s , is chosen as input of the ADC and $N_{samples}$ consecutive samples are saved, a perfect period of the input sine will be obtained. Ideally (no noise on the signal, ENOB equal to 12), if an FFT of this population is performed, the spectrum obtained is made of only a component with amplitude different from zero at the input signal frequency. Otherwise, if the relation (3.2) is not respected or if f_s is not the real one, the output of the FFT will be made of more than one frequency component with amplitude different from zero.

In our case, the sampling rate is unknown and only an indicative value is known (obtained with the second method). It's possible to see if the FFT is close or far away from the ideal one, increasing and decreasing the frequency of the sine generated by the signal generator. In the first case, this means that we are close to the real sampling frequency while in the second we are moving far away from it. Furthermore, if we get closer, parameters like SINAD will increase.

The operation described above was carried out starting from the frequency measured by the second method. The conditions shown in Table 3.3 were chosen to evaluate f_{in} from the relation (1).

Parameter	**Value**
$\boldsymbol{N_{samples}}$	2048
$\boldsymbol{M_{cycles}}$	2
$\boldsymbol{f_{in}}$	1024*f_s

Table 3.3: Coherent sampling parameter

As suggested in the Microcontroller datasheet, the input signal was chosen with a maximum value and a minimum value lower than 95% and greater than 5% of the reference voltage respectively.

The offset and amplitude chosen are showed in Table 3.4

Parameter	**Value**
Wave type	Sine
Offset (V)	1.65
$\boldsymbol{V_{pp}}$ **(V)**	2.97
Max value (V)	3.135
Min value (V)	0.165

Table 3.4: Input signal parameter

A Matlab script was used to obtain the FFT. In particular, to reduce the noise floor, an average of the FFT was carried out. For this reason, 30x2048 samples were acquired for each tested input frequency. There are 2 different ways to obtain this average value:

Incoherent integration: It consists of carrying out the FFT of each population of 2048 samples and does the module of its output. Subsequently, the different FFT outputs with the same index are added up and divided by the number of acquired populations as shown in equation (3.3).

$$\mathrm{F_{incoh}(i)} = \frac{|\mathrm{F_1(i)}| + |\mathrm{F_2(i)}| + \ldots + |\mathrm{F_{29}(i)}| + |\mathrm{F_{30}(i)}|}{30}; \quad \mathrm{i} = 1, 2, \ldots, 2048. \qquad (3.3)$$

Coherent integration: It consists of carrying out the FFT of each population of 2048 samples and adding the complex outputs of the FFT with the same index without performing the module. The FFT coefficients obtained are divided for the number of acquired populations.

Equation (3.4) clarifies the operations.

$$F_{coh}(i) = \frac{F_{1\_real}(i) + \ldots + F_{30\_real}(i)}{30} + j\frac{F_{1\_imag}(i) + \ldots + F_{30\_imag}(i)}{30}; \; i = 1, 2, \ldots, 2048. \qquad (3.4)$$

Let's consider why the integration gain afforded by coherent averaging is more useful than the integration gain from incoherent averaging. An incoherent integration reduces the variation in the FFT's background noise because we are dealing with magnitude and so all FFT noise bin values are positive. Therefore, their averaged values will never be zero.

On the other hand, when a coherent integration is carried out, those complex noise bin values can be positive or negative. When we add them up, they can cancel each other and in this way, the background noise is reduced.

To have the best coherent integration, the input signal must have a fixed phase during every FFT. The phase is really important in the FFT output because the sign of each complex number will depend on this parameter. The noise components have a random initial phase, so using this FFT averaging method, the complex values of each FFT bin will cancel each other if they are related to a noise bin. As it is possible to understand, if the input signal is sampled at a random initial phase, the fundamental will also be reduced by this type of FFT averaging. A trigger is required and for this reason, an incoherent integration was preferred to the coherent one.

First of all, the incoherent integration was performed to find the mean value of the input frequency that is related through (1) with the sampling frequency. After finding it, the ADC ENOB was evaluated.

Executing the operations described previously, it was obtained the sampling rate shown in Table 3.5.

Parameter	**Value (μs)**
μ	28.571
σ	0.008

Table 3.5: sampling rate measured with FFT

Note that the resolution obtained is improved and the range has been reduced.

To demonstrate the validity of the work, Figure 3.20 shows the output of the FFT in 3 different cases.

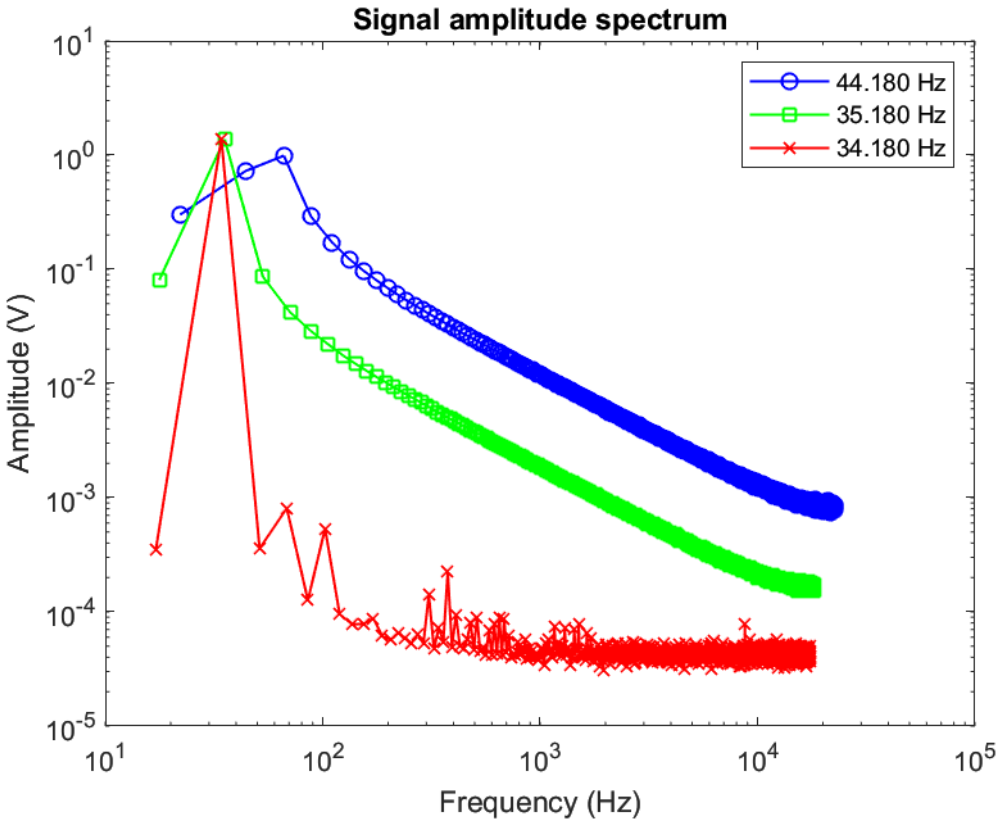


Figure 3.20: Signal Amplitude Spectrum

As it is possible to see in Figure 3.21, the more we get closer to the input frequency that is 1024 times lower than the sampling frequency, the more the spectrum is closer to that of the ideal FFT.

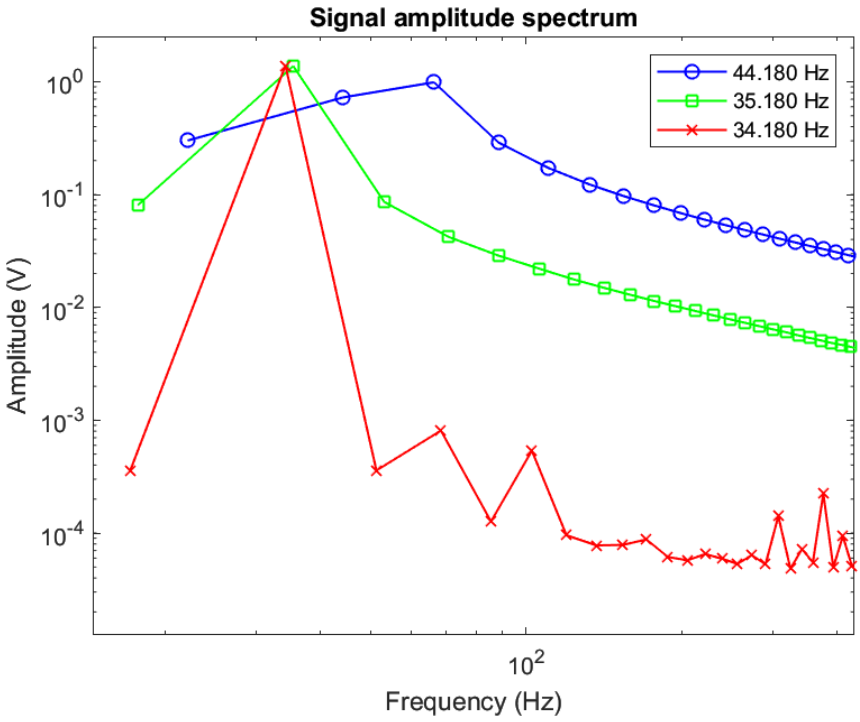


Figure 3.21: Signal Amplitude Spectrum. Focus on the lower frequency range.

The same is true for the peak of the fundamental which is greater when we are closer to the searched input frequency, as it is possible to see in Figure 3.22.

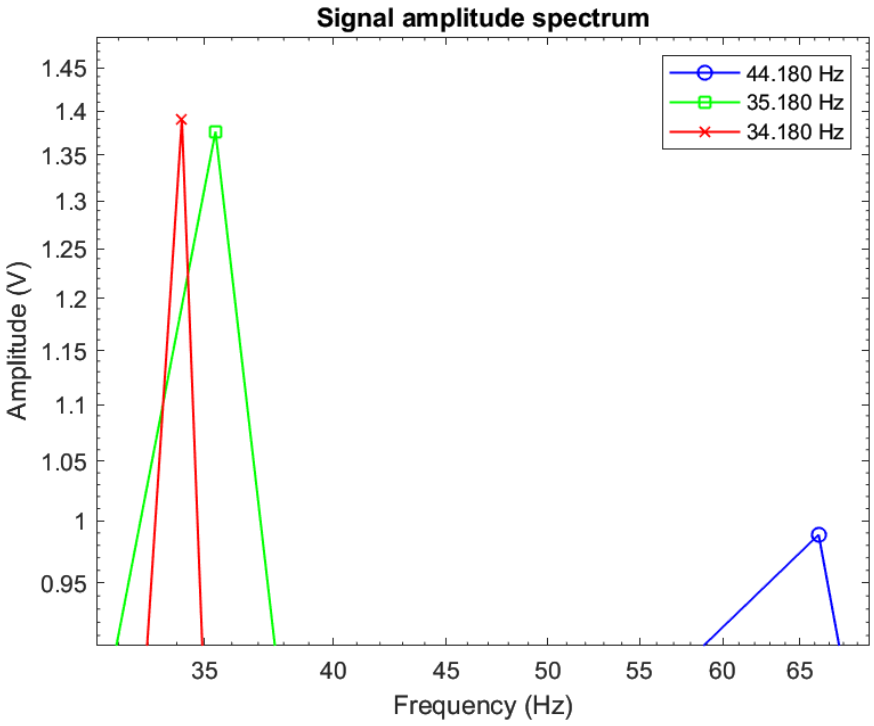


Figure 3.22: Signal Amplitude Spectrum. Focus on peak value.

Table 3.6 shows the results obtained for the 3 different input frequencies.

Input Freq. (Hz) / Parameter	34.180	35.180	44.180
Max Amplitude (V)	1.390	1.376	0.967
Frequency of the fundamental (Hz)	34.180	35.18	66.27
SINAD (dB)	57.72	20.01	0.71

Table 3.6; Parameter vs Input Frequency

We can affirm that the real value of the sampling frequency is contained in the range 34.180 ± 0.010 Hz because it is possible to prove that the SINAD will reduce if we consider signals with input frequency equal to 34.190 and 34.170. Therefore, the input frequency that is 1024 times lower than the sampling frequency must belong to this interval.

Figure 3.23 shows the FFT outputs for the other 3 different frequencies.

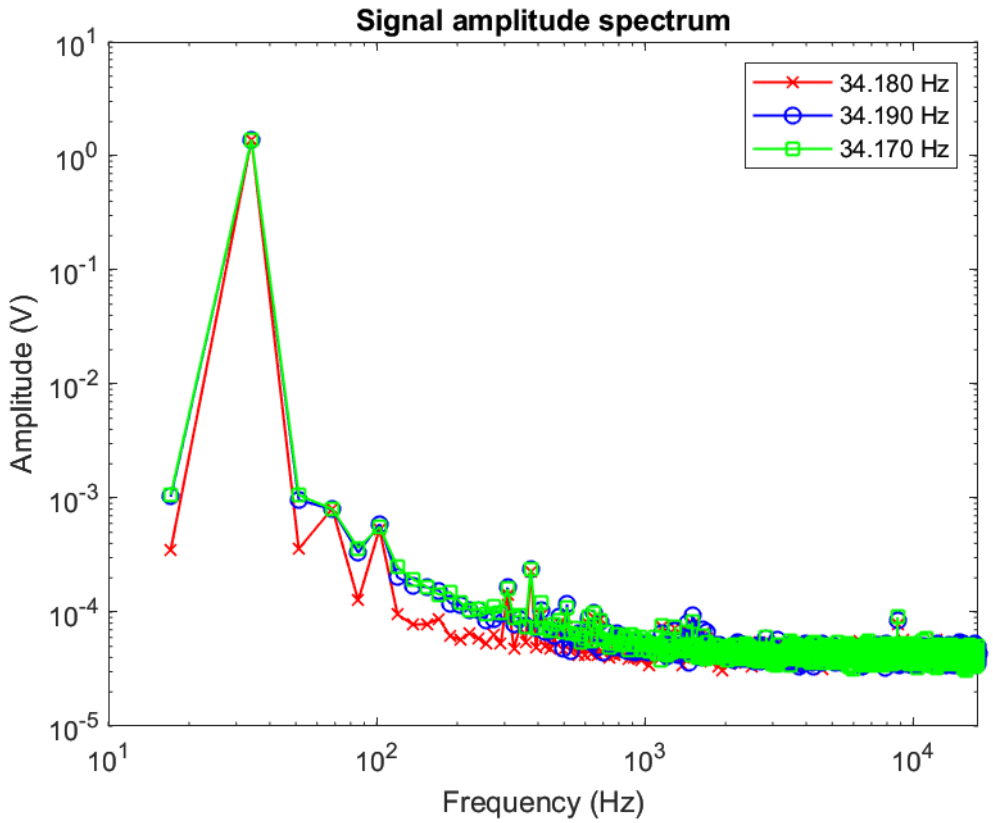


Figure 3.23: Signal Amplitude Spectrum

As it is possible to see in Figure 3.24, the more we get closer to the input frequency that is 1024 times lower than the sampling frequency, the more the spectrum is closer to that of the ideal FFT.

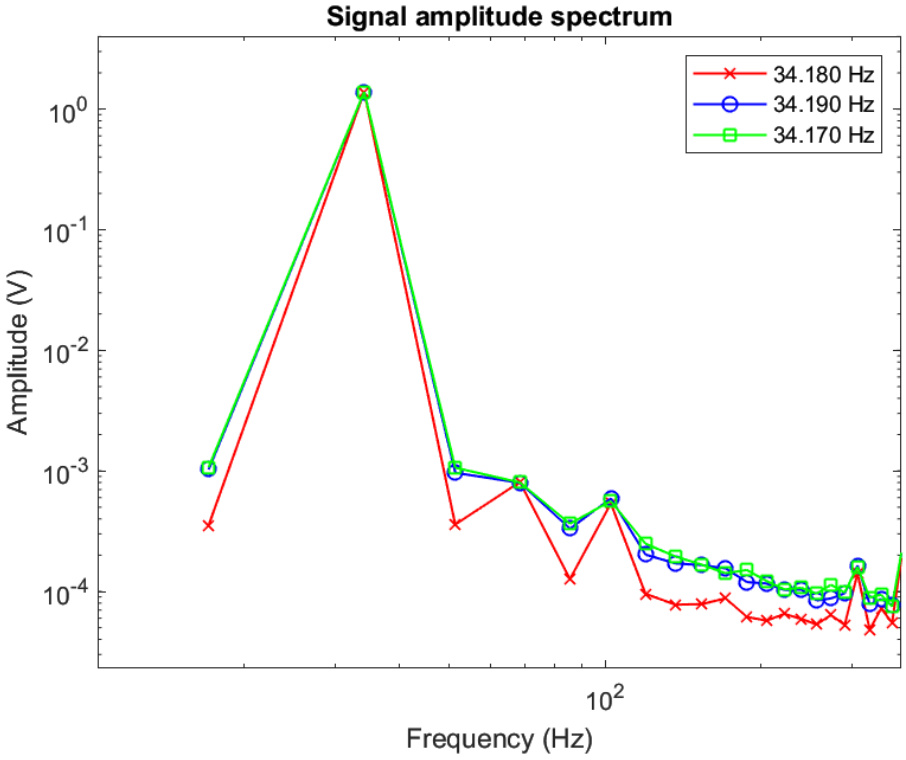


Figure 3.24: Signal Amplitude Spectrum. Focus on a lower frequency range

The same is true for the peak of the fundamental which is greater when we are closer to the searched input frequency, as it is possible to see in Figure 3.25.

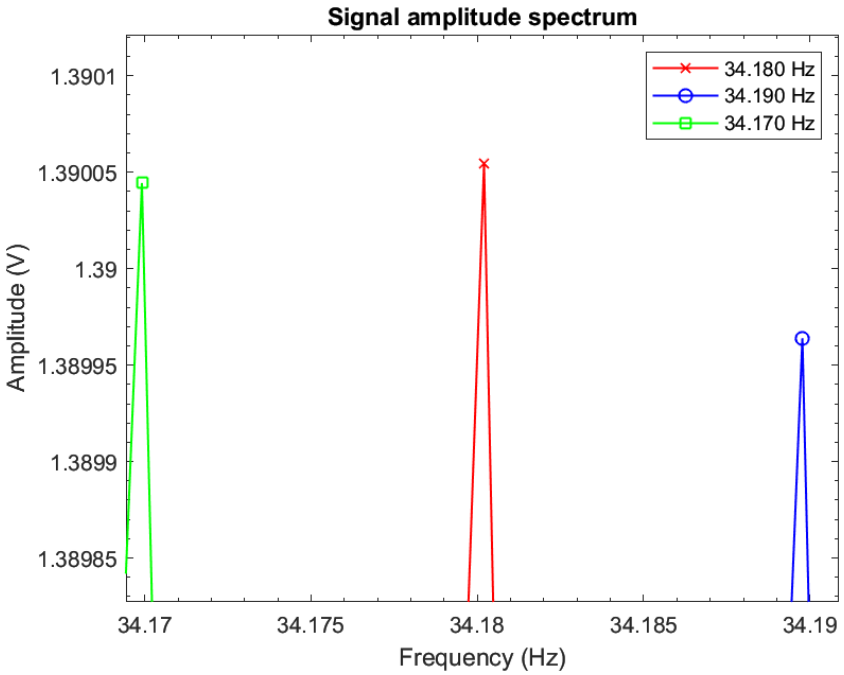


Figure 3.25: Signal Amplitude Spectrum. Focus on peak value.

Table 3.6 shows the results obtained for the 3 different input frequencies.

Input Freq. (Hz) / Parameter	34.180	34.190	34.170
max Amplitude (V)	1.390	1.390	1.390
Frequency of max Amplitude (Hz)	34.180	34.190	34.170
SINAD (dB)	57.72	55.57	55.29

Table 3.6: Parameter vs Input Frequency

The ADC sampling rate is showed in Table 3.7

Parameter	**Value (μs)**
μ	28.571
σ	0.008

Table 3.7: sampling rate measured with FFT

The ADC sampling frequency is showed in Table 3.8

Parameter	**Value (Hz)**
μ	35000.32
σ	10.25

Table 3.8: Sampling frequency measured with FFT

Once demonstrated that the input frequency which is 1024 times lower than the sampling frequency belongs to the interval 34.180 ± 0.010 Hz, the ENOB was evaluated with an input signal with a frequency of 34.180 Hz.

3.3.5 ENOB Measure

In Figure 3.26, it is possible to see the power spectrum obtained from the FFT of an input sine with a frequency of 34.180 Hz.

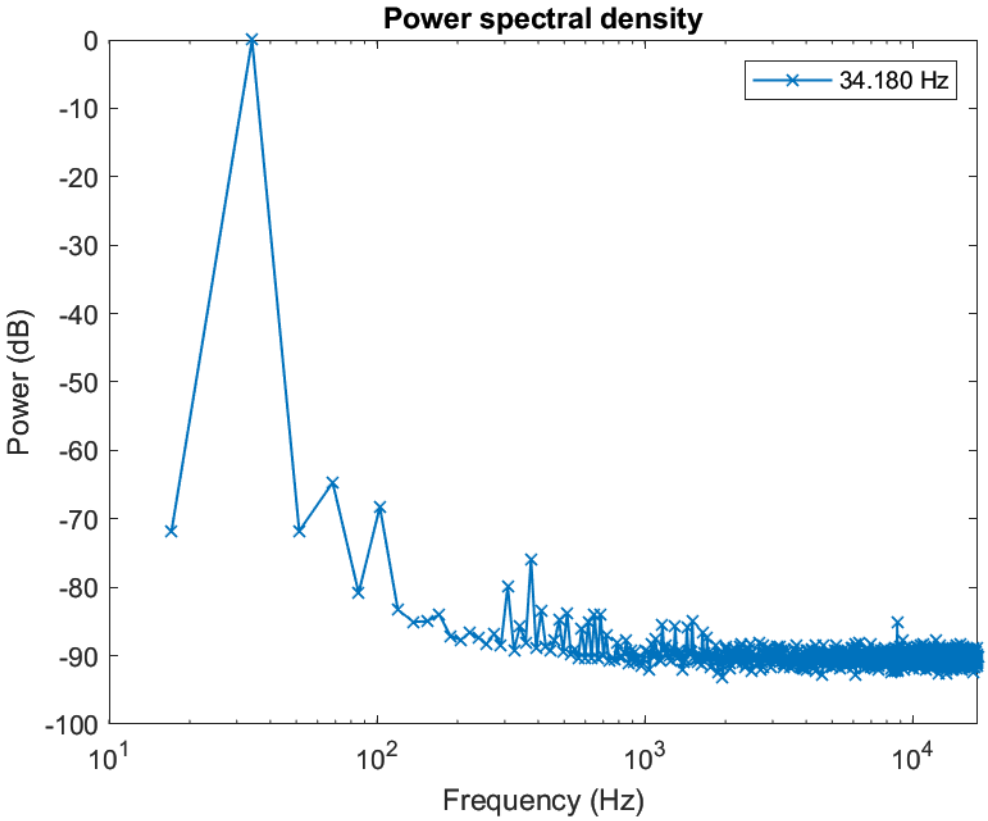


Figure 3.26: Power spectral density

From the SINAD, it is possible to obtain the ENOB thanks to the equation (3.5).

$$ENOB = \frac{SINAD - 1.76}{6.02} \qquad (3.5)$$

Table 3.9 shows the SINAD and the ENOB obtained.

Parameter	Incoherent integration
SINAD (dB)	57.72
ENOB	9.3

Table 3.9: Parameter vs Incoherent and Coherent integration

The datasheet of the MCU provides an ENOB tested for single-ended mode, with $frequency_{ADC}$ of 2.1 MHz and 1xgain.

His typical value is 9.5 while his is maximum 9.8. The minimum value is not provided.

The ENOB measured is not the same due to the noise introduced by the testing board and on the input signal. It is important to consider that to have this ENOB, it was necessary to add to the initial test configuration an RC filter with a cut-off frequency of 100Hz.

3.3.6 External watchdog timer

To avoid the malfunction of the MCU, an external watchdog timer has been implemented in the FPGA.

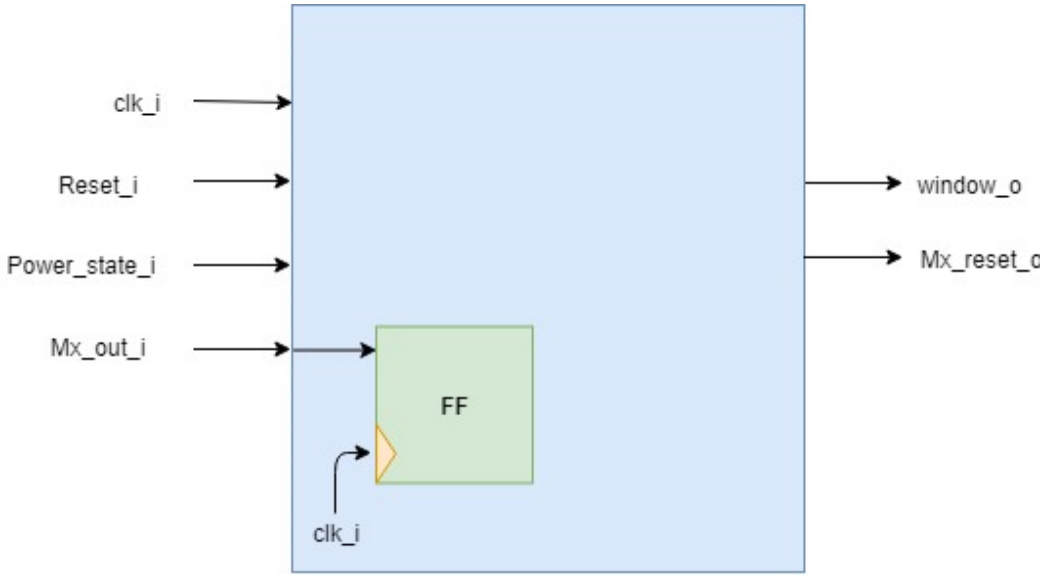


Figure 3.27: watchdog timer Controller block diagram

This module implements an FSM described in Figure 3.28.

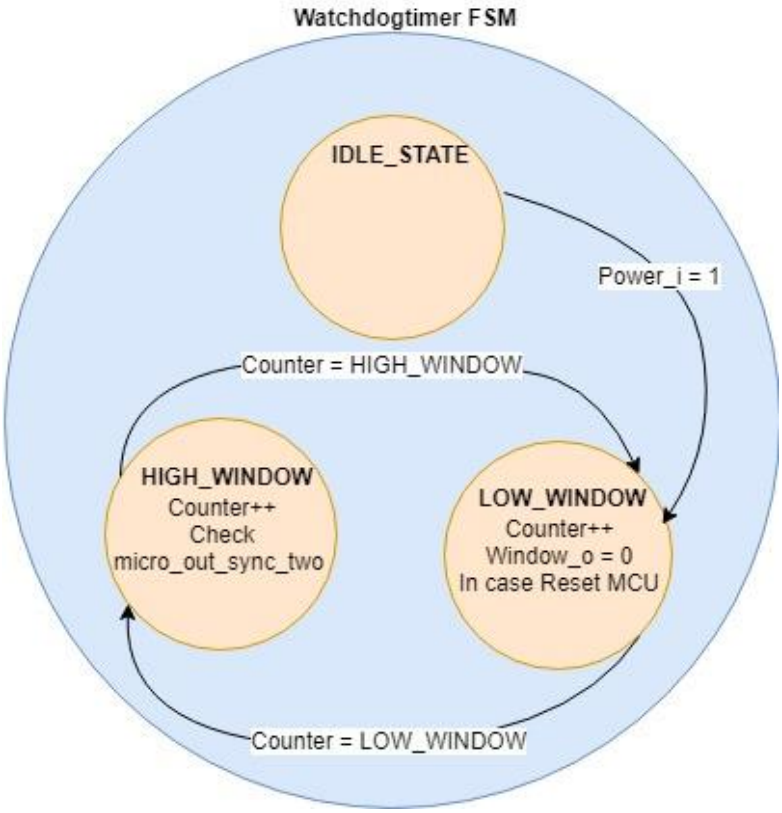


Figure 3.28 — Watchdog Timer Controller Finite State Machine

The Watchdog Timer Controller is functioning on a simple scheme that is controlled by an FSM of three states. The reset state is the IDLE_STATE.

In this state, the power_i signal is checked and when it becomes high, the FSM reaches the LOW WINDOW state.

In the LOW WINDOW state, window_o is set low, a counter is increased each rising clock edge and the signal no_reset is checked. If it is low, it means that a reset of the MCU is required and therefore, Mx_reset_o is set low and a counter (time_reset) is increased each rising edge of the clock. When time_reset reaches the value corresponding to 1 μs (8 clocks), Mx_reset is set high, time_reset is reset and no_reset is set high.

When the counter is equal to LOW_WINDOW_PERIOD, it is reset, no_reset is set low and the state of the FSM becomes HIGH_WINDOW.

In the High_window counter is increased each rising edge of the clock and window_o is set high. In particular, in this state

micro_out_sync_two (the synchronized signal coming from the micro) is check for each rising edge of the clock and if it is high, no_reset is set high. This check is performed until the counter is equal to HIGH_WINDOW_PERIOD. When this condition is verified, the counter is reset, the FSM reaches the LOW_WINDOW state. The cycle restarts.

3.3.7 SPI Master

As said in chapter II, to interface MCU and FPGA, it is necessary to implement a module in the FPGA that emulates the SPI protocol (Master Side).

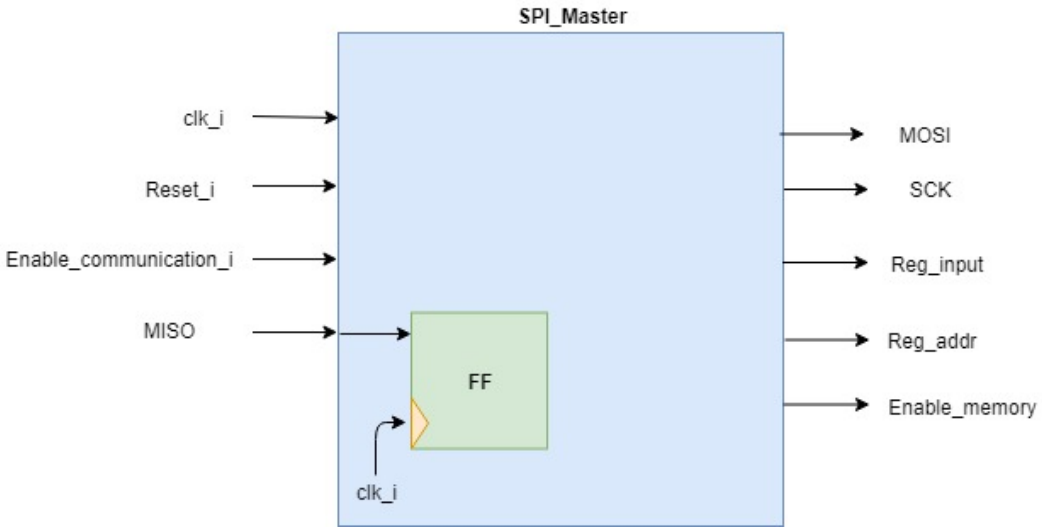


Figure 3.29: SPI Master block diagram

In particular, the SPI_Master is connected to memory through the signals Reg_input, Reg_Addr, and Enable_memory where the bytes read from the MCU are saved.

The SPI_Master module implements the FSM described in Figure 3.30.

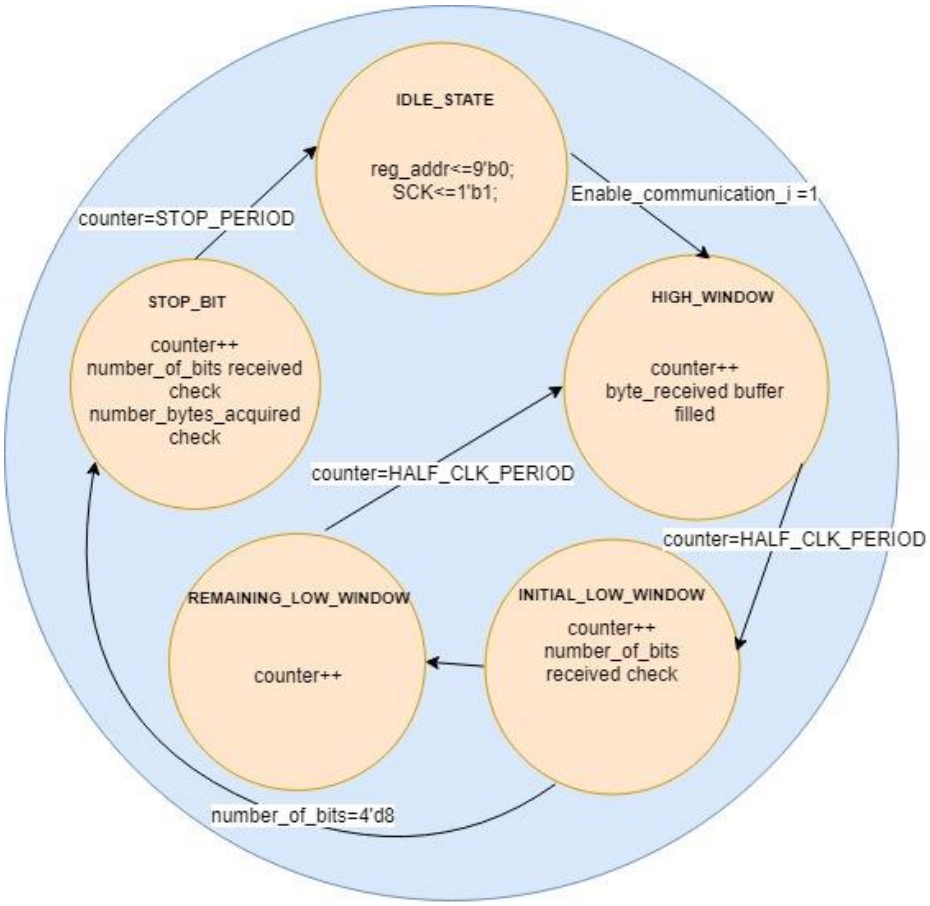


Figure 3.30: SPI Master FSM

The reset state of the FSM is the IDLE_STATE where SCK and Reg_addr are set low. Each rising edge of the clock, enable_communication_i (a signal that is set high every second by the Main_controller) is checked and when it is high, the FSM state becomes High_Window.

In this state, a counter is increased with each rising edge of the clock. After 1 μs, the byte received is shifted of one position on the left and the LSB is set equal to MISO_sync_two. This signal is the output of two flip-flops connected that receive MISO as input.

When the counter reaches the value of HALF_CLK_PARAMETER (Corresponding to 6 μs), it is reset, SCK is set low and the FSM reaches the INITIAL_LOW_WINDOW state.

There, the counter is increased by one. If the number of bits is equal to 8, the input of the register (reg_input) is set equal to byte

received, enable_memory is set high and number_bytes_acquired is increased and the FSM state is set equal to STOP_BIT. Otherwise, the FSM state is set equal to REMAINING_LOW_WINDOW.

In REMAINING_LOW_WINDOW, the counter is increased with each rising edge of the clock. When the counter reaches the value of HALF_CLK_PARAMETER (Corresponding to 6 µs), it is reset, SCK is set high and the FSM reaches the HIGH_WINDOW state.

In STOP_BIT, the counter is increased with each rising edge of the clock. If the number of bits is equal to 8, enable_memory is set low (disabling it), reg_addr increased by one, byte_received and number_of_bits reset to zero. If number_of_bytes_acquired is equal to NUMBER_OF_BYTES_REQUIRED, counter and number_of_bytes_acquired are reset and the FSM backs in IDLE_STATE. Otherwise, when the counter reaches the value of STOP_PERIOD (Corresponding to 12 µs), the counter is reset, SCK is set high and the FSM reaches the HIGH_WINDOW state.

3.3.7 SPI Test

The firmware presented previously was modified to have the SPI on the new pins

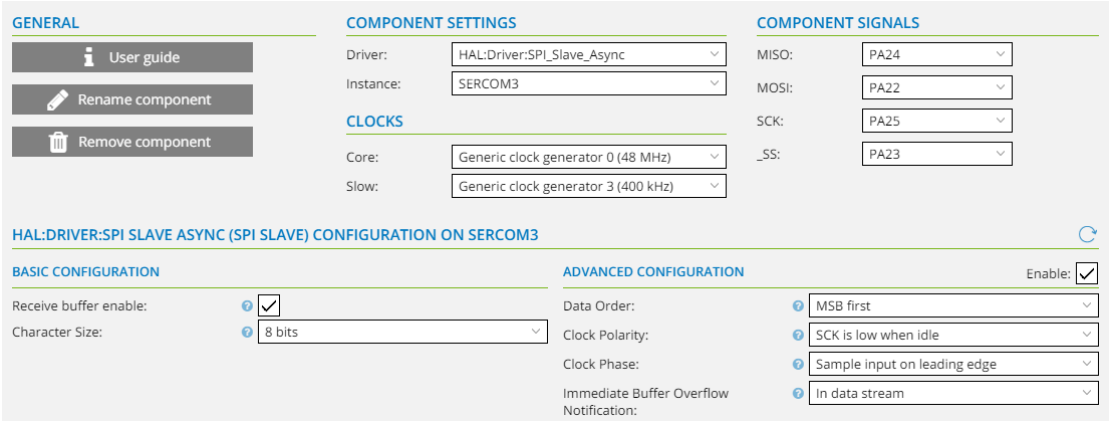


Figure 3.31: SPI Slave configuration

The firmware was modified to send 4 constants bytes (0x22, 0x33, 0x44, and 0x55). To check that no transmissions are lost, the 4th byte is increased by one each transmission. The data once sent through SPI, is sent from the FPGA to the outside via I2C protocol.

```
09:57:35.423 -> 2233          4492
09:57:35.423 -> 56
09:57:36.425 -> 2233          4493
09:57:36.425 -> 57
09:57:37.428 -> 2233          4494
09:57:37.428 -> 58
09:57:38.431 -> 2233          4495
09:57:38.431 -> 59
09:57:39.434 -> 2233          4496
09:57:39.434 -> 60
09:57:40.451 -> 2233          4497
09:57:40.451 -> 61
09:57:41.451 -> 2233          4498
09:57:41.451 -> 62
09:57:42.451 -> 2233          4499
09:57:42.451 -> 63
09:57:43.438 -> 2233          449A
09:57:43.491 -> 64
09:57:44.495 -> 2233          449B
09:57:44.495 -> 65
09:57:45.498 -> 2233          449C
09:57:45.498 -> 66
09:57:46.501 -> 2233          449D
09:57:46.501 -> 67
09:57:47.481 -> 2233          449E
09:57:47.481 -> 68
09:57:48.508 -> 2233          449F
09:57:48.508 -> 69
09:57:49.511 -> 2233          44A0
09:57:49.511 -> 70
```

Figure 3.32: SPI test

The result of the test is shown in Figure 3.32.

3.4 Flash Freeze Technology

Space application always requires systems with low power or that spend less energy than is possible (In particular when they are not used).

For this reason, the FPGA was changed with a new model that is low power and has the Flash Freeze technology.

The Flash Freeze technology allows the system to shut off dynamic power consumption instantaneously and retain all SRAM and register information.

IOs and Clock can still be driven without impact on power consumption.

To let the Flash Freeze technology work properly, an IP provided by MicroSemi was implemented.

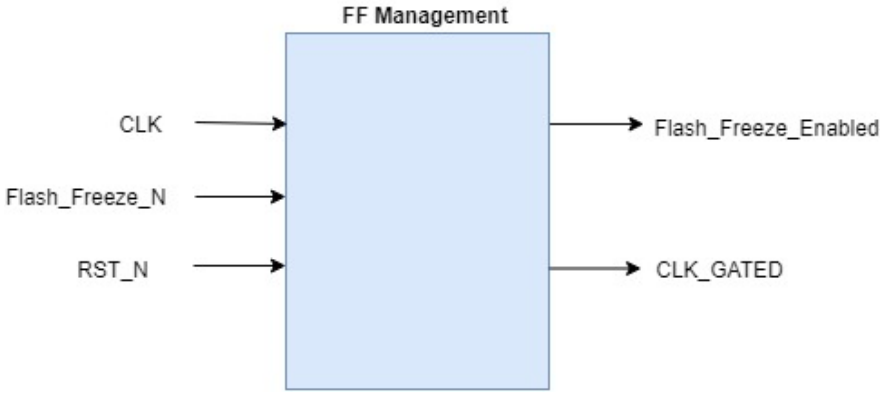


Figure 3.33: FlashFreeze Management IP block diagram

The Flash Freeze management IP has 2 modalities of working:

1) Flash_Freeze_N = 1: in this case, the CLK_GATED is equal to the input CLK.
2) Flash_Freeze_N = 0: in this case, the CLK_GATED is set low and instantaneously shut off dynamic power consumption. All the SRAM and register information is retained because the system continues to be supplied.

To allow this IP to work properly, the CLK_GATED must be connected to all the sequential modules in the system.

3.4.1 FlashFreeze improvements

In Figure 3.34, the power consumption with the old FPGA is showed.

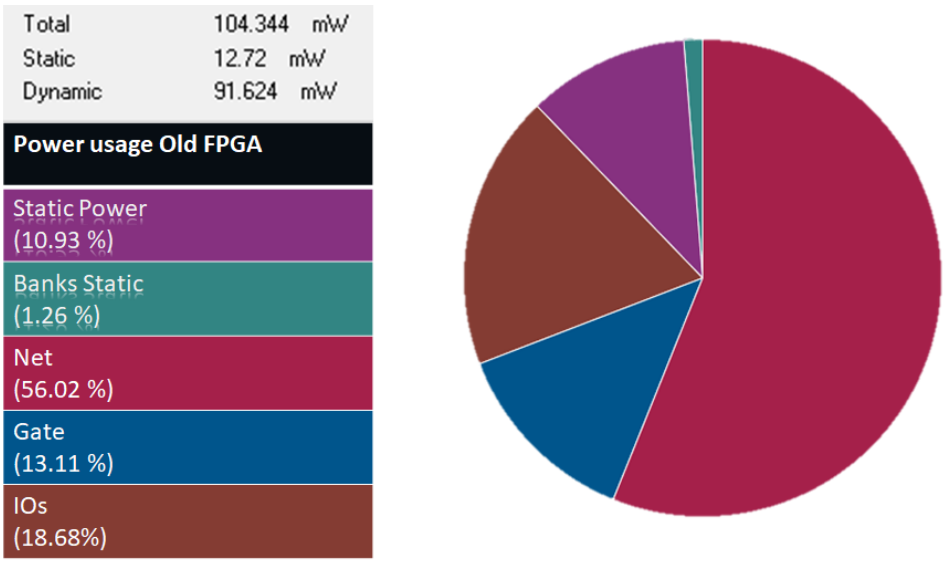


Figure 3.34: Power consumption of the old FPGA

In Figure 3.35, the power consumption with the new FPGA is shown. As it is possible to see, the total power consumption was reduced by 87.39 %.

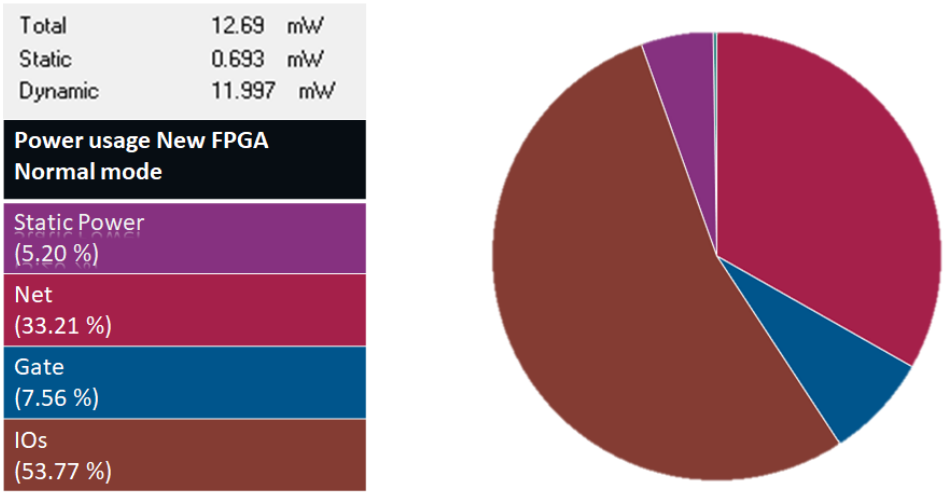


Figure 3.35: Power consumption of the new FPGA with FF disabled

Figure 3.36, shows the power consumption when Flash Freeze mode is enabled.

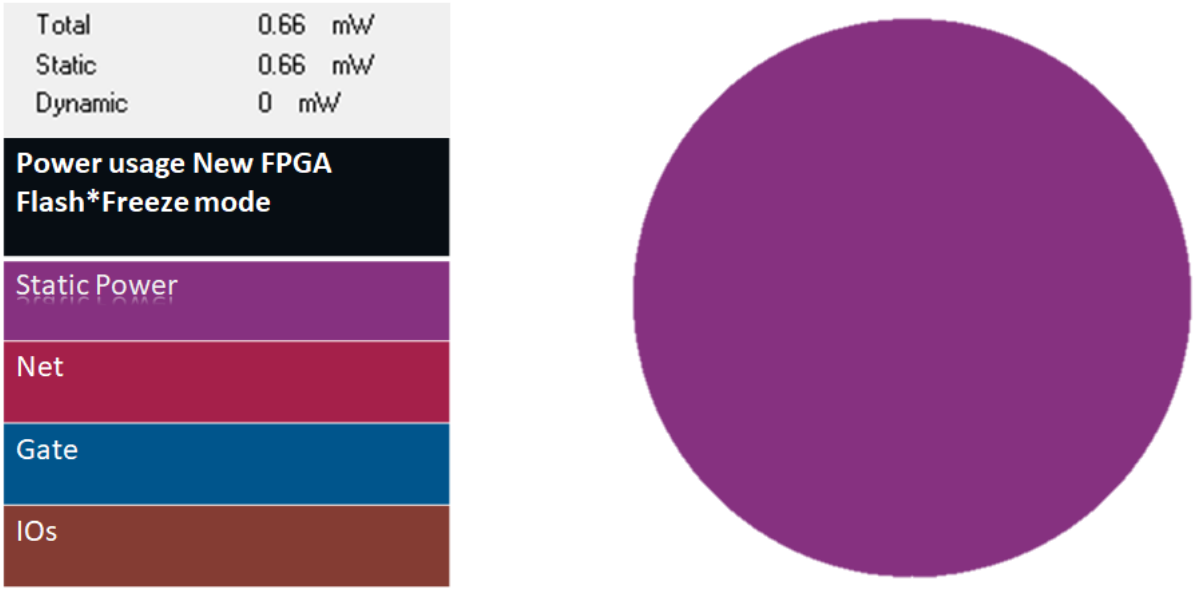


Figure 3.36: Power consumption of the new FPGA with FF enabled

3.5 I2C communication

To transmit the sensor readings to the external world, an I2C slave module was implemented.

3.5.1 Implemented structure

The I2C Slave module implemented is used to interface the FPGA with the outside through the I2C protocol. Through the SDA and SCL channels, the module receives commands and transmits the sensors' readings. It uses a three-state buffer to make SDA as input or output depending on what is needed. It also implements a Register interface in which the sensors' readings are connected to the Protocol Controller through a mux controlled by regAddr and a control register can be written to have the system power on and reset. During this discussion, we will indicate as control register the registers that can be written and read and serve to control the system through I2C. (2.6.5. for more information). A state of synchronization is also used to synchronize SDA and SCL with the 8 MHz clock and to delay them to detect START and STOP conditions. The structure is depicted in Figure 3.37.

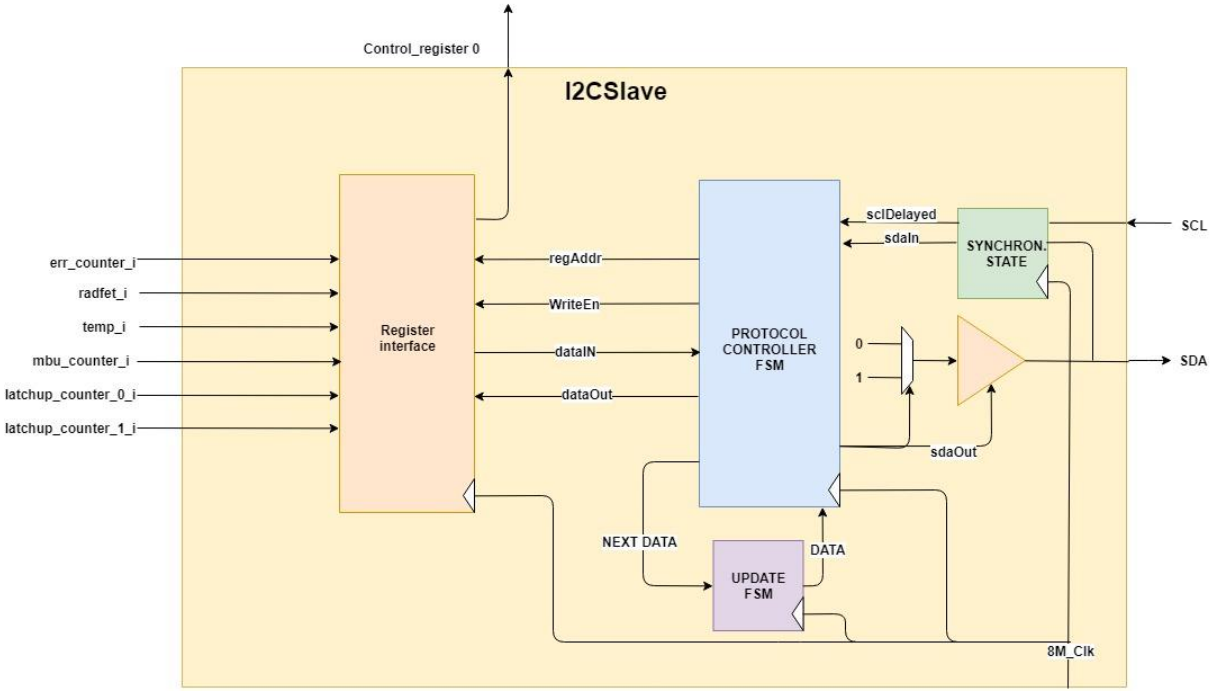


Figure 3.37: I2CSlave module block diagram

In particular, it uses two different FSM: one controlled by several signals (PROTOCOL CONTROLLER FSM) and another

controlled by the clock (UPDATE FSM). This FSM simply updates several variables with the updated value in the PROTOCOL CONTROLLER FSM, every rising edge of the clock. In the Figure below, the UPDATE FSM is depicted.

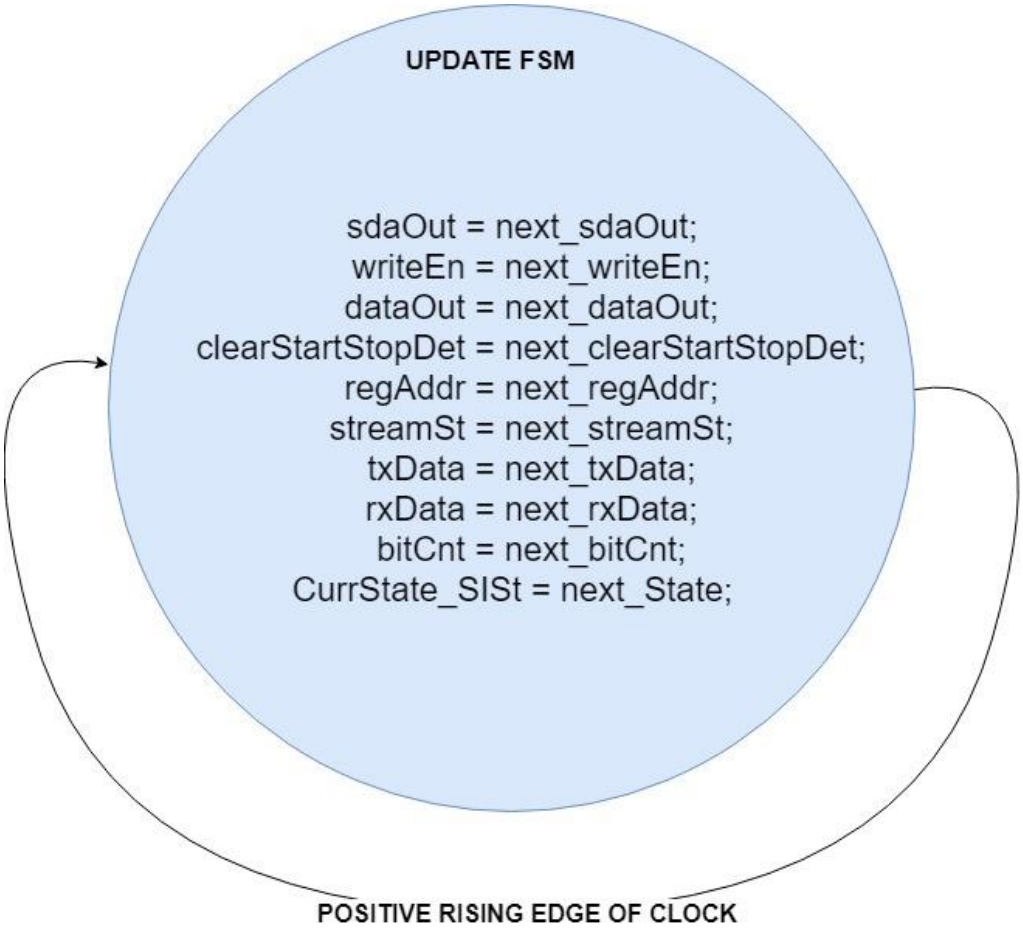


Figure 3.38: UPDATE Finite State Machine

It is important to understand when the PROTOCOL CONTROLLER FSM updates its state. Most of the signals that control this FSM are updated by UPDATE FSM: therefore PROTOCOL CONTROLLER FSM seems that it changes its state only after there is a rising edge of the clock and an UPDATE FSM output has been updated. However, there is also asynchronous signal that is not updated in UPDATE FSM which is SCL. This external signal is made synchronous using different flip flops (the same happens for SDA). In particular, the SCL and SDA input are not delayed by the same time.

Therefore PROTOCOL CONTROLLER FSM modifies its state or checks it, whenever an UPDATE FSM output is changed or each time SCL changes.

In the Figure below, the PROTOCOL CONTROLLER FSM is depicted.

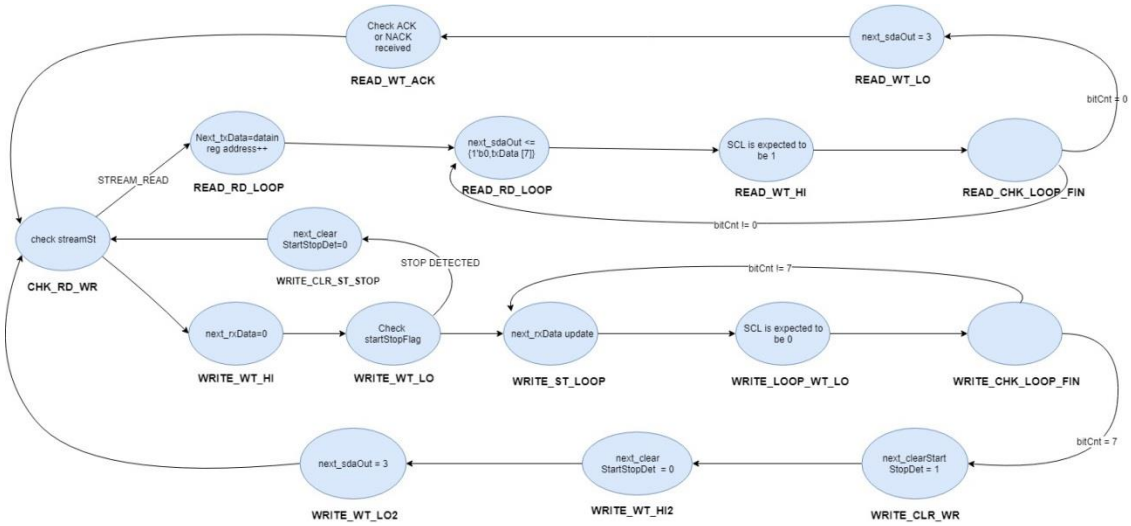


Figure 3.39: PROTOCOL CONTROLLER Finite State Machine

The reset state of the FSM is CHK_RD_WR.

It is important to remember that the state is updated only after the rising edge of the clock. In the PROTOCOL CONTROLLER FSM, only a temporary signal called next_state is updated while CurrState_SISt will be updated with the new value of next_state, only at the next rising edge of the clock. It is the same for all the other variables that are called "next_ [nameofvariable]". This means that if next_state changes, currState_SISt changes at the next rising edge of the clock, and the state of the FSM will change at the second rising edge of the clock following the next_state change. This operating condition implies for the correct operation of the FSM the need to have a system clock much faster than the communication clock.

During CHK_RD_WR is checked if stream_st is equal to STREAM_READ. (It means that a reading has been requested). If this condition is true, next_bitCnt is set equal to 1, next_regAddr is increased by one and the data to be transmitted (next_txData) is set equal to dataIn (This value is the one selected by regAddr through the mux in the register interface module).

In this case, the next state that will be reached will be READ_RD_LOOP. If the above condition is not respected, next_rxData is set equal to 0 and next_state becomes WRITE_WT_HI.

We first study which states follow WRITE_WT_HI.

In WRITE_WT_HI, it expects SCL to be high. When it is equal to one, next_rxData is updated by shifting the bits of rxData to the left of a position and setting the LSB equal to the SDA value. Moreover, next_bitCnt is set equal to one and the next_state becomes WRITE_WT_LO.

When the FSM is in the WRITE_WT_LO state, it expects SCL to be low state. When it is, if a STOP condition was detected, next_clearStartStopDet would be set high to clean the STOP FLAG, next_streamSt would be set equal to STREAM_IDLE and next_state would become WRITE_CLR_ST_STOP. This is possible because the SCL of this FSM is delayed compared to the one that in the i2cslave module is checked to detect the START and STOP. Therefore, STOP has been detected before that this signal goes up.

This case occurs also if there is a variation of SCL and SDA that does not respect I2C protocol (so there is not a start). This condition is made to avoid that nothing happens if SDA and SCL have changed differently from the one defined by I2C.

In WRITE_CLR_ST_STOP, next_clearStartStopDet is set equal to 0 to detect a new START and the FSM returns in the initial state CHK_RD_WR.

If a STOP was not detected, the next state that would be reached will be WRITE_ST_LOOP. In particular, if a START was detected next_rxData would be reset. Instead, if startStopDetState is equal to NULL value (nothing was detected) next_bitCnt is set equal to bitCnt plus one.

When the FSM is in the WRITE_ST_LOOP state, it expects SCL to be high so that data on the line can no longer change. When it is equal to one, next_rxData is updated by shifting to the left of a position the bits of rxData and setting the LSB equal to the input value of SDA. Next_state is set equal to WRITE_LOOP_WT_LO.

In WRITE_LOOP_WT_LO, it expects SCL to be a low state and when it is, the next state becomes WRITE_CHK_LOOP_FIN.

This state is the most important of the structure. First of all is checked, if bitCnt is equal to 7. If it is not, the next_state becomes WRITE_ST_LOOP, and the bit count is increased by one.

If it is, next_sdaOut is set equal to 0 (This is important because in this way SDA is set as output and equal to 0) and the streamSt is checked

If streamSt is equal to STREAM_IDLE, Rx data is checked.

In particular, there are 3 different cases possible:

1) The first 7 bits of rx_data are equal to the slave address, a start was detected, and LSB equal to one: the received sequence is a read command and the next_stream_st is set equal to STREAM_READ.
2) The first 7 bits of rx_data are equal to the slave address, a start was detected and LSB is equal to zero: the received sequence is a write command and the next_streamst is set equal to STREAM_WRITE_ADDR
3) No one of the previous cases: next_sdaOut is set equal to 1 to have a NACK on the SDA line.

If the streamSt is equal to STREAM_WRITE_ADDR, next_streamst is set equal to STREAM_WRITE_DATA and next_regAddr will be set equal to rxData.

If the streamSt is equal to STREAM_WRITE_DATA, next_writeEn is set equal to one and next_dataOut will be set equal to rxData.

For all these cases the next_state will be set equal to WRITE_CLR_WR.

In this state, next_writeEn is set equal to zero and the START flag is cleaned by setting next_clearStartStopDet equal to 1. Furthermore, write_en is checked. If it is equal to one, next_regAddr is increased by one so that the subsequent data received, if there is not a stop, is written in the next address (continuous I2C writing). The next_state becomes WRITE_WT_HI2.

In WRITE_WT_HI2 next_clearStartStopDet is reset to detect a new START or STOP, and expects SCL to be equal to 1. When it is, next_state is set equal to WRITE_WT_LO2 where, after SCL is equal to zero, SDA is set another time as input by setting next_sdaOut equal to 3 (SDA is an inout so it is assigned equal to high impedance to work as input) and next_state becomes CHK_RD_WR.

Now, the states following CHK_RD_WR will be studied when streamSt is equal to STREAM_READ.

As we have already said, in this particular condition, the state reached is READ_RD_LOOP. In this state, it expects SCL to be a low state and when it is, next_sdaOut is set equal to 0 or 1 and next_txData is set equal to a byte made of txData shifted by one position to the left and with the LSB equal to zero. The next_state becomes READ_WT_HI.

In this state, it expects SCL to be high and the state changes to READ_CHK_LOOP_FIN.

In READ_CHK_LOOP_FIN bitCnt is checked. If it is equal to zero, the state becomes READ_WT_LO. If it is not, next_bitCnt is increased by one and the FSM returns in READ_RD_LOOP.

In READ_WT_LO, it expects SCL to be 0. When it is, next_sdaOut is set equal to 3 and the next_state becomes READ_WT_ACK.

In this state, it expects SCL to be high and SDA is controlled. If it is equal to 1 (NACK) next_streamSt is set equal STREAM_IDLE. In the case of NACK or ACK, the FSM returns in the idle state.

3.5.2 I2C Writing

Automatically the FSM reaches WRITE_WT_LO and remains in this state until SCL is not pulled down by the Master. When it is low, if a START is detected, communication can begin. We study which states are crossed by the FSM to write to a control register.

If a start is detected the FSM reaches WRITE_ST_LOOP where next_rxData is updated by shifting to the left of a position all the bits of rxData and setting LSB equal to the value of SDA only when SCL is equal to 1 because SDA will change only when SCL is zero. The next state that will be reached, will be WRITE_LOOP_WT_LO where it expects SCL to be low. The next_state is also changed to WRITE_CHK_LOOP_FIN where bitCnt is controlled and FSM returns to WRITE_ST_LOOP. This cycle is repeated 8 times (considering this step). At the 8^{th} time, the received data is checked because the stream_st is equal to STREAM_IDLE (default value). If the first 7 bits of rx_data are equal to the slave address and the 8^{th} equal to zero, the stream data is changed to STREAM_WRITE_ADDR, and next_sdaOut is set equal to zero (In this way the slave takes possession of the SDA line when it is left by the Master and it pulls down the SDA line to have an ACK). To send the ACK the FSM reaches WRITE_CLR_WR, where the start flag is cleaned and the next_state is changed to WRITE_WT_HI2. In this state, it expects SCL to be high and the next_state becomes WRITE_WT_LO2. Since SCL is high, the ACK was recognized by the master because the slave forced the SDA line to go down thanks to the previous operations.

When SCL is low, it can leave the control of SDA because in I2C communication SDA cannot change during the high front of SCL, or a START or a STOP would be recognized. Therefore, in the WRITE_WT_LO2 state, next_sdaOut is set equal to 3 only when SCL is equal to zero (SDA is an inout signal so it is assigned equal to high impedance to work as input).

CHK_RD_WR is reached again. In this case, the byte received will be the address of the control register where we want to write, as the I2C defines. The stream_st is different from READ_STREAM, and so next_state becomes WRITE_WT_HI. In this state, when SCL is equal to one, next_rxData is updated

by shifting all the bits of rxData to the left of a position and setting the LSB equal to the value of SDA.

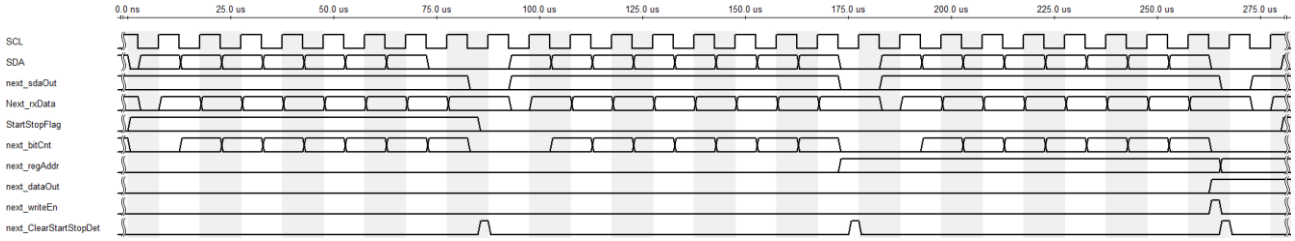
In WRITE_WT_LO, when SCL is equal to zero, the StartStopDetState is checked and this time it will be equal to NULL because it was previously cleared. Furthermore, bitCnt is increased by one. During the previous phase, in WRITE_WT_LO, StartStopDetState was equal to START value and next_rxData was cleaned. Therefore the operations made in WRITE_WT_HI were useless and it was not necessary to increase bitCnt. Now it is necessary to increase bitCnt because we are acquiring the MSB of the byte.

The next state that will be reached, will be WRITE_CHK_LOOP_FIN where bit count is controlled and the FSM returns to WRITE_ST_LOOP. This cycle is repeated 7 times (considering this step). At 7^{th} time, the next_regAddr is set equal to rx_data because the stream_st is equal to STREAM_WRITE_ADDR, next_sdaOut is set equal to zero (In this way the slave takes possession of the SDA line when it is left by the Master and it pulls down the SDA line to have an ACK) and the next_stream_st is changed to STREAM_WRITE_DATA. To send the ACK the FSM reaches WRITE_CLR_WR, where the start flag is cleaned and the next_state is changed to WRITE_WT_HI2. In this state, it expects SCL to be high and the next_state is changed to WRITE_WT_LO2. Since SCL is high, the ACK was recognized by the master because the slave forced the SDA line to go down thanks to the previous operations.

The next byte will be the data to be written to the control register, selected with the previous step. The steps that the FSM goes through are almost the same. Only two steps change from the previous case.

1) The seventh time WRITE_CHK_LOOP_FIN is reached, stream_st is equal to STREAM_WRITE_DATA: next_writeEn is set to high and the dataOut is set equal to rx_data.
2) In WRITE_CLR_WR, besides the other operations already described, next_write_en is reset and next_regAddr is increased by one (this is important because it gives the possibility of having a continuous write in the register interface).

The write on the register interface can be repeated or it can be stopped by a STOP, which once it is detected, closes the communication and resets the stream_st to the default value.



Time Diagram 3.1: Timing diagram of I2C WRITING

3.5.3 I2C Reading

Automatically the FSM reaches WRITE_WT_LO and remains in this state until SCL is not pulled down by the Master. When it is low, if a START is detected, communication can begin. We study which states are crossed by the FSM to read the sensors' readings from the register interface.

If a start is detected the FSM reaches WRITE_ST_LOOP where next_rxData is updated by shifting to the left of a position all the bits of rxData and setting LSB equal to the value of SDA only when SCL is equal to 1 because SDA will change only when SCL is zero. The next state that will be reached, will be WRITE_LOOP_WT_LO where it expects SCL to be high. The state is also changed to WRITE_CHK_LOOP_FIN where bitCnt is controlled and FSM returns to WRITE_ST_LOOP. This cycle is repeated 8 times (considering this step). At the 8^{th} time, the received data is checked because the stream_st is equal to STREAM_IDLE (default value). If the first 7 bits of rx_data are equal to the slave address and the 8^{th} equal to one, the stream data is changed to STREAM_READ, and next_sdaOut is set equal to zero (In this way the slave takes possession of the SDA line when it is left by the Master and pulls down it to have an ACK). To send the ACK the FSM reaches WRITE_CLR_WR, where the start flag is cleaned and the next_state is changed to WRITE_WT_HI2. In this state it expects SCL to be high and the next_state becomes WRITE_WT_LO2. Since SCL is high, the ACK was recognized by the master because the slave forced the SDA line to go down thanks to the previous operations.

It can leave the control of SDA when SCL is low because in I2C communication SDA cannot change during the high front of SCL or a START or a STOP would be recognized. Therefore, in the WRITE_WT_LO2 state, next_sdaOut is set equal to 3 only when SCL is equal to zero (SDA is an inout signal so it is assigned equal to high impedance to work as input).

CHK_RD_WR is reached again but this time the stream_st is STREAM_READ so another condition is satisfied concerning the previous case. In particular, next_txData (the data that we want to transmit) is set equal to the byte selected by regAdd in the register interface. Next_regAddr is increased by one and bitCnt is set to one.

The next state that will be reached, will be READ_RD_LOOP where it expects SCL to be low. When the condition is met, next_txData is shifted from one position on the left and the next_sdaOut LSB is set equal to the tx_data MSB.

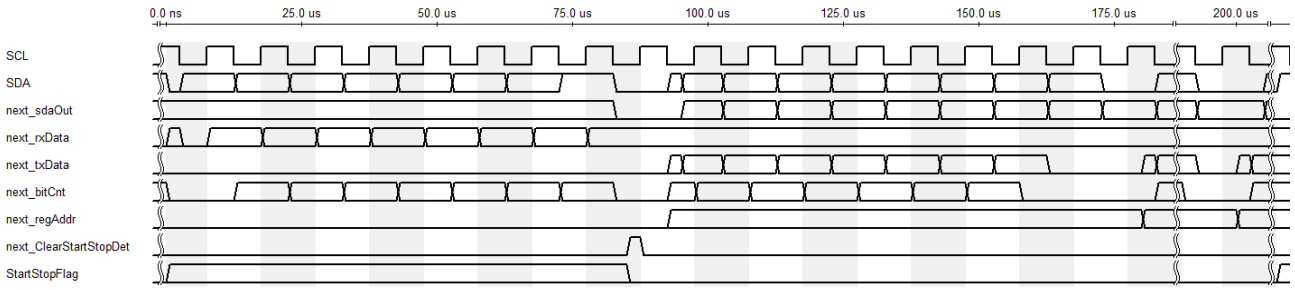
If it is equal to zero, SDA must be forced to a low value because the data value that we want to transmit is zero. Setting this value of next_sdaOut, the SDA line is pulled down by the SLAVE because an ACK bit is sent.

If it is equal to one, SDA must be forced to high value because the data value that we want to transmit is one. Setting this value of next_sdaOut, the SDA line is pulled up by the SLAVE because a NACK bit is sent.

The next state that will be reached, will be READ_WT_HI where it expects SCL to be high and the state is updated to READ_CHK_LOOP_FIN when this condition is met.

In this state, it expects bitCnt to be equal to zero and if this condition is not verified, the FSM returns in READ_RD_LOOP, and bitCnt is increased by 1. This cycle is repeated 8 times (considering also this passage). At the 8th time, the byte has been sent and we must receive an ACK and so, the next state will not be READ_RD_LOOP but READ_WT_LO.

In this state, it expects SCL to be low and next_sdaOut is set equal to 3: in this way, the slave releases the SDA line. The next state that will be reached, will be READ_WT_ACK where it expects SCL to be high. If a NACK is received the next stream_st will be STREAM_IDLE. If an ACK is received, another byte is sent repeating this cycle. In both cases, the FSM returns in CHK_RD_WR.



Time diagram 3.2: Timing diagram of I2C READING

3.5.4 Register Interface structure and the principle of working

This paragraph will be explained how the register interface works and how it is possible to connect to it new signals to transmit them via I2C.

In Figure 3.40, the register interface structure is depicted.

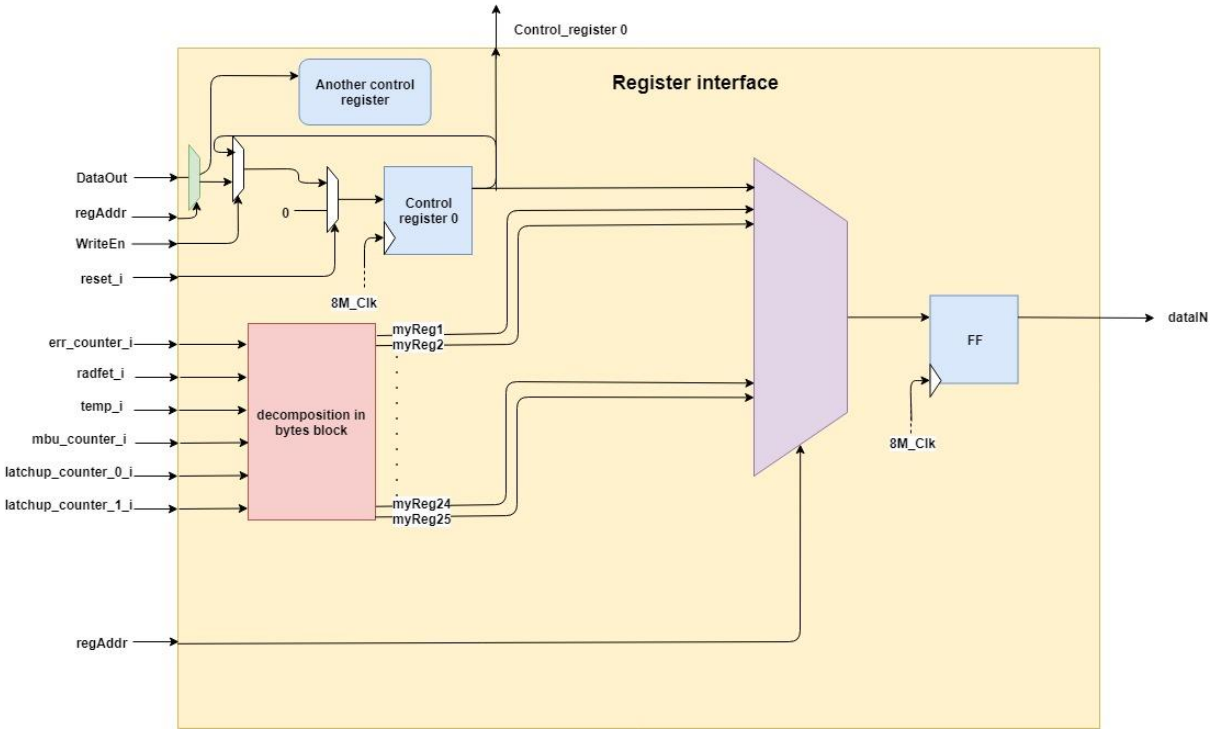


Figure 3.40: Register Interface module block diagram

As it is possible to see, the sensors' readings are the input of a block called decomposition in a byte. It is not a real module. This decomposition is carried out by linking the different Verilog modules together. In particular, the real decomposition takes place in the I2C_Top.v file where the signals of 32 and 24 bits are respectively decomposed in 4 or 3 bytes and divided into different signals.

These new signals obtained are connected to the register interface in the i2cSlaveTop.v file where they are organized to have, for example, that the signal called reg0 is the control register 0 while the signal called reg25 is a constant value (0x18). Once we understand that this decomposition module is made through the interconnection of different Verilog modules that contain within them the register interface and have the only purpose of organizing the signals to be used as input and output for this module we can begin to study how it works.

The i2cSlave does not always see the same dataOut: its value depends on the selected value of regAddr. In particular, regAddr controls two different multiplexers. One for reading and the other for writing: not all signals are connected to this second multiplexer. This is because there are two kinds of signals:

1) The signals that can be only read: The signals that can only be read: this group contains the signals obtained from the reading sensors after the decomposition and the division operation.
2) The signals can be read and write: they are signals that we want to modify to control the system via I2C (Control Register).

The reading Multiplexer (in purple) connects a signal (that can be one from reg0 to reg25) to dataOut and which line is connected to dataOut depends on regAddr.

In particular, every time that there is a rising edge of the clock the value of regAddr is checked and dataOut is set equal to the set reg with that particular regAddr.

The writing Multiplexer (in green) connects dataIn to one line (which line, always depends on the value of regAddr), and if the reset line is low, and writeEn is high, it modifies the value of a control register at the rising edge of the clock.

However, for the designed system, it's possible to select only the control register 0, but only because the other regs do not have to be written and it is not necessary to use more than one byte to control the system. In particular, with this structure, it is possible to add new a control register like the previous one and select via regAddr which one we want to write. To write a new value to the selected register, it is also necessary that write_En is set high by the Protocol Controller. Otherwise, the register control does not change his output value, and in our case, reg0 assumes his previous value.

In case it is selected as a regAddr different from 0x00 (regAddr for control register 0), nothing will be written.

Conclusion

In this thesis work, a space payload capable of monitoring radiation in the space environment was developed. In particular, a first study was conducted on the effects of radiation and on the techniques for mitigating the SEEs. This overview was fundamental to know how to develop the SpaceRadMonMx system in the best possible way.

SpaceRadMonMx, as explained during the thesis, is a system based on an MCU and FPGA. The way of interfacing these two devices were studied and detailed in the state of the art research where the simplest and cheapest way to interface them has been chosen. Then, the SpaceRadMonMx firmware was described in all its features. First, the sensors used to monitor the radiation effects have been described: RadFet and SRAM for Single Event Upset counting and Single Event Latchup monitoring. To control them, FPGA implements several modules written in Verilog and the state machines developed have been described in this work.

The novelty of the SpaceRadMonMx is to use the MCU instead of a discrete ADC to monitor the analog voltages. Indeed in this version of the system, the MCU is used to acquire analog voltage from RadFET via an ADC and transmit data to the FPGA via the SPI protocol. The MCU configuration is reported and a study on the sampling rate measurement was conducted. In particular, different techniques have been used to obtain a better resolution of the measurements. The best method to use is the FFT analysis through a coherent sampling that also allowed to measure the ENOB of the ADC. In the future, it may be possible to increase the ADC resolution using an oversampling technique.

To communicate with the MCU, a module has been developed that emulates an SPI master. The behaviour of the SPI Master was studied using another microcontroller and replicated in the FPGA.

In particular, to make the system at low power, a study was conducted on different types of FPGA and the best was chosen for this purpose. The FlashFreeze technology, equipped on the particular FPGA chosen was described, and different graphs were plotted in the chapter to show the advantages of use it.

To communicate the data, the I2C protocol was implemented through several modules in the FPGA.

The whole system has been successfully tested.

Bibliography

[1] A. Holmes-Siedle, Len Adams. *Handbook of radiation effects*. Oxford University Press, 2nd edition, 2002, ISBN 019850733X.

[2] M. Brugger, R.G. Alia, S. Danzeca, R. Denz, J. Mekki, P. Oser, P. Peronnard, J. P. De Carvalho Saraiva, G. Spiezia, J. Steckert, Y. Thurel, S.Uznanski. *Radiation effects, Calculation Methods and Radiation Test Challenges in Accelerator Mixed Particle and Energy Environments*. RADECS 2014 Short Course, 2014.

[3] Faccio, F. *COTS for the LHC radiation environment: the rules of the game*. 2000.

[4] Kastensmidt, Fernanda Lima. *SEE Mitigation Strategies for Digital Circuit Design Applicable to ASIC and FPGAs*. 2007 IEEE NSREC Short Course, 2007.

[5] Petersen, Edward L. *SINGLE EVENT UPSETS IN SPACE: BASIC CONCEPTS*. Tutorial Short Course IEEE1983 Nuclear and Space Radiation Effects Conference, 1983.

[6] Schwank, Jim. *TOTAL DOSE EFFECTS IN MOS DEVICES*. 2002 IEEE NSREC, 2002.

[7] Toro, Rocendo Bracamontes Del. *eetimes*. 30 July 2008.

[8] Daode Zhang, Yurong Pan, and Xinyu Hu. “Design of High-Speed Parallel Data Interface Based on ARM & FPGA.” *JOURNAL OF COMPUTERRS, VOL. 7* (2012).

[9] J. Jean Rossario Raj, S.M.K. Rahman, Sneh Anand. “8051 microcontroller to FPGA and ADC interface design for high speed.” *Engineering Science and Technology,* (2016).