UiO : Department of Physics
University of Oslo

Exploring the Single Event Effect Sensitivity of the TMS570 MCU for a CubeSat Application

Yassine Elfarri

Master's Thesis, Autumn 2018

Abstract

The thesis aims to explore the single events sensitivity of TMS570 microcontroller for the CubeSat application. This device is automotive with a range of features that can be used to increase the reliability of the devices in a radiation environment. To understand the sensitivity of the devices ones needs to test in the devices as close as possible to the environment, it will operate in. The decapping methods are explored in this thesis. This will be important to be able to perform heavy ions testing. In the rest of the test, we explore the sensitivity of the devices in mixed field environment CHARM.

Acknowledgments

I would like to thank associate professor Ketil Røed for the opportunity to induce me to the embedded world of electronics in a radiation environment. Your help and guidance challenged me to ask the right questions.

I dedicate a special thanks for Olav Bjerke, Pål Bjerke and Ottar Opland for using there personal time and knowledge to provide a 3D Xray at the facility of FFI. I thank my classmates Emil Ulvestad and Eirik Nobuki Kosaka for the support and incredible journey. Last but not least I thank my wonderful girlfriend Alla Johannessen for her unwavering support and encouragement during my work on this thesis.

Nomenclature

ADC Analog-to-Digital Converter

CCS Code Composer Studio

CERN Conseil Européen pour la Recherche Nucléaire

CHARM Cern High energy AcceleRator Mixed field/facility

CMOS Complementary Metal–Oxide–Semiconductor

COTS Commercial off-the-shelf

DAC Digital-to-Analog Converter

DCC Dual clock comparator

DMA Direct Memory Access

DUT Device Under Test

EDAC Error Detection And Correction

ESM Error Signaling Module

FMPLL Frequency Modulated Phase-Locked Loop

GCR Galactic Cosmic Ray

HALCOGEN Hardware Abstraction Layer COde GENerator

HEH High Energy Hadrons

I2C Inter-Integrated Circuit

IC Integrated Circuit

LEO Low Earth Orbit

LET Linear Energy Transfere

LHC Large Hadron Collider

LPOCLKDET Low Power Oscillator Clock Detector

m-NLP Multi-Needle Langmuir Probe

MBU Multiple Bit Upset

MCU Multiple Cell Upset

NIEL Non-Ionizing Energy-Loss

OBC On Board Computer

SAA South Atlantic Anomaly

SBU Single Bit Upset

SCPI Standard Commands for Programmable Instruments

SEE Single Event Effects

SEFI Single Event Functional Interrupt

SEL Single Event Latchup

SEPs Solar Energetic Particles

SET Single Event Transiant

SEU Single Event Upset

SPI Serial Peripheral Interface

SRAM Static Random-Access Memory

UART Universal Asynchronous Receiver-Transmitter

Contents

List of Figures

List of Tables

1 | Introduction

This chapter we will present the motivation for validating the m-NLP system in radiation environment, the main goals that and an overview of every chapter.

1.1 4DSpace

4DSpace is a research group at the University Of Oslo with a cross-faculty corporation that involves the departments of Informatics, Mathematics, and Physics. The goal of the research group is to understand plasma instabilities and turbulence in the ionosphere.

The sun is the primary source of changes in space weather. During the high solar activity, large amounts of particles are ejected into space. When that considerable amount of particles reach the ionosphere in high concentrations, they cause irregularities in form of plasma structures [13]. Those plasma structures lead to deterioration of radio signals due to phenomena called scintillation.

In the long term, studying the effects of space weather on earth ionosphere will help predict the electron density and help to mitigate the impact of scintillations in the northern region. The space weather and its effects on the ionosphere are beyond the scope of this thesis. However, it provides an understanding of the goals and motivations of the 4DSPACE research group.

1.2 Motivation

A nanosatellite is used to study plasma turbulence in the ionosphere. This satellite is equipped with m-NLP system [14, 15] that enables the 4Dspace group to collect and process data. The system is a scientific instrument developed by UIO. The scientific payload uses four Langmuir probes at a fixed bias to measure different plasma parameters. The instrument is made entirely of commercially available electronic components.

Commercial off-the-shelf (COTS) are increasingly used in Nanosatellites.COTS are increasingly attractive for space applications because of the low price tag,

small size, high performance and power efficiency[16]. The risk of using them in a radiation environment need to be understood and mitigated. Screening the devices that will operate in the radiation environment contribute to a better understanding of the sensitivity and causes of failures. This increase the confidence level that this device will perform in a radiation environment and throughout the lifetime of the planned mission.

1.2.1 Goals

The primary objective of this thesis is to understand whether the selected programmable COTS in m-NLP systems can operate in Low Earth Orbit (LEO). The work will contribute by laying the groundwork for the system level testing of the m-NLP at an accelerated radiation facility. Decapping is one of the stages of testing that is essential to test components with heavy ions[17]. Heavy ions testing was not performed in this thesis. However, The goal is to explore the possibility of using the available resources at the University of Oslo. This will enable the future projects to do in-house injection testing with laser and cyclotron. The rest of the thesis is developing firmware for radiation testing, along with data collection and processing. The radiation test is done at CHARM. In the thesis we will be highlighting some mitigation techniques that can be utilized to improve the reliability of the system.

1.3 Outline

The thesis is divided into the follow chapters.

- Chapter 1 Introduction to the thesis
- Chapter 2 Relevant theoretical framework for this thesis
- Chapter 3 The process of exposing COTS die
- Chapter 4 The firmware development of m-NLP system
- Chapter 5 Radiation validation of m-NLP system at CHARM
- Chapter 6 Conclusion

2 | Theoretical Framework

This chapter is an overview of relevant theoretical fundamentals of radiation interaction with electronics. The primary focus is the Complementary Metal–Oxide–Semiconductor (CMOS) technology. The chapter starts with an introduction to radiation environments, the radiation interaction with silicon material, followed by radiation effects on electronics, and mitigation techniques.

2.1 Radiation Environment

Understanding the radiation environment is crucial to find suitable Commercial off-the-shelf (COTS). Engineers have to take into account the radiation environment when selecting components that are not designed to operate in harsh condition. Understand the ionization energy of the particles population and whether it is a concern for the devices used in this environment. Radiation is classified as natural and artificial[3, 18]. The Natural radiation is ionizing particles trapped in belts around planets, Galactic Cosmic Ray (GCR), and solar events. Artificial radiation is a human-made source that is used either in the medical domain, nuclear power plants and high energy experiments like the Large Hadron Collider (LHC).

This section discusses the radiation environment in space with a more emphasize on the LEO, and LHC.

2.1.1 Space Radiation Environment

Space is not an empty vacuum it is populated by ionized particles and electromagnetic radiation from Galactic Cosmic Ray (GCR) and Solar Energetic Particles (SEPs)[3, 18].

GCR are made up of highly energetic particles that originate from sources beyond our solar system. these particles can reach a maximum energy of TeV [3]. They are continues present with some fluctuation. The solar activity has an inverse correlation to the GCR. During the maximum solar activity, GCR is

decreasing due to the hot ionized gas ejected by the sun. This gas contains a magnetic field that deflects the low energy GCR [18].

SEPs are the results of event such as coronal mass ejections and flares[3]. These events produce energetic protons, heavy ions and electrons with energies up to GeV[3]. The flux level of solar particles is directly correlated to the solar activity. This activity has an 11 years cycle that divided into four low activity years and seven highly active years[3]. The high energy particles from GCR and SEPs have high enough energies to penetrate the earth magnetic field[3]. These particles are hazardous to orbiting satellites.

Earth's Radiations Belts

The particles that do not have enough energy to penetrate the magnetic field of earth tend to get deflected or trapped into belts[3]. These belts are called after James Van Allen. He confirmed the existence of radiation belts in the late 50s. The Van Allen belts are made of an inner and outer belts[3]. The inner belts are about 2500 km from the equator[3]. This belt is populated mostly by protons with energies that can reach up to 100s MeV[3]. The magnetic pole is tilted and offset with respect to the rotational axis of the earth. This tilts the inner belts as close as 300 km over South America[3]. This phenomenon is called South Atlantic Anomaly (SAA).

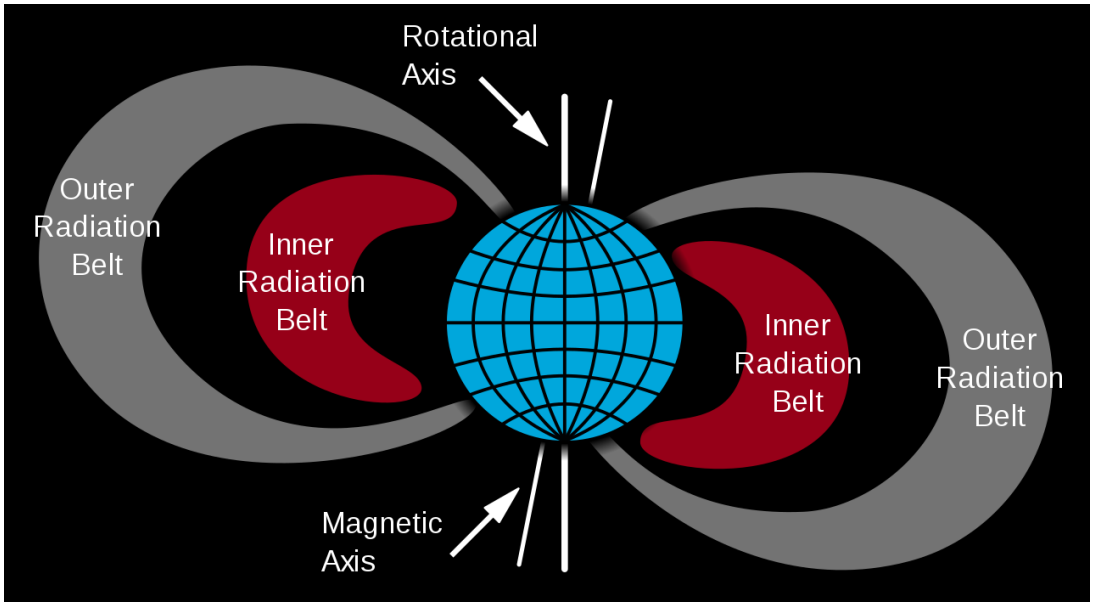


Figure 2.1: An artist image of earth Van Allen radiation belts. Public Domain Image is by NASA [1] .

Since the 70s much effort went into building models of radiation belts. NASA's AP-8 model is the standard for trapped protons. The model takes into account protons with energies greater than 10 MeV[2]. Such a model can be generated using SPENVIS tool Figure 2.2[2].

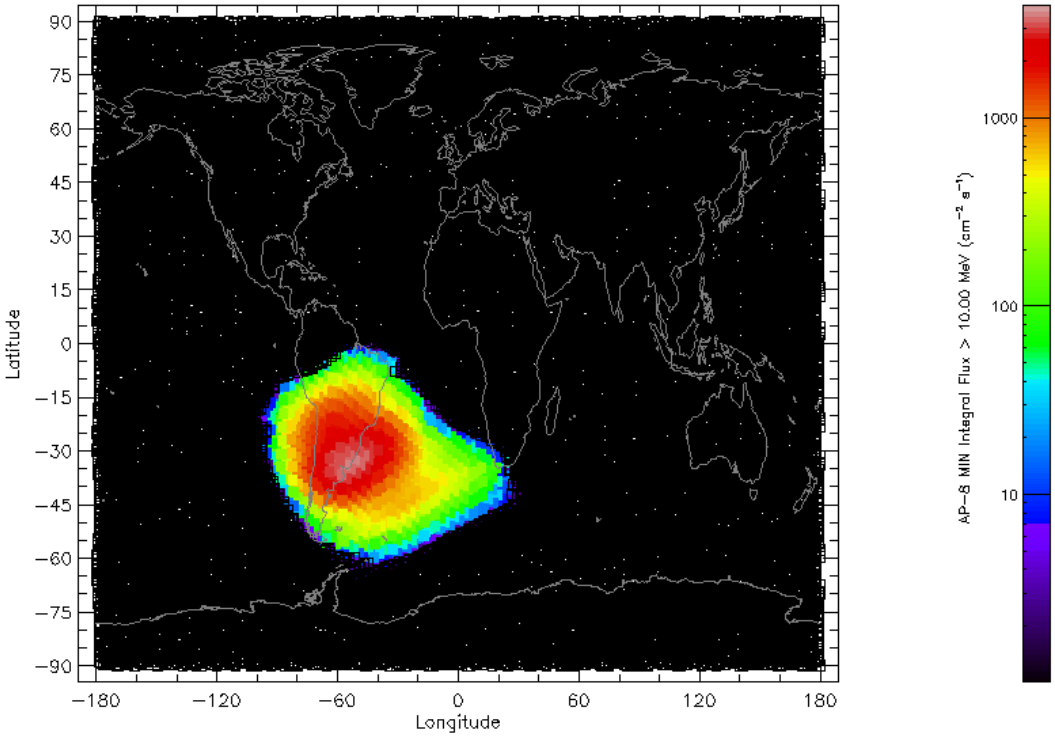


Figure 2.2: World map of SAA proton flux at 600 km, generated using SPENVIS[2]

Atmosphere

The particles from GCR and SEPs that has enough energy enters the atmosphere. They interact with nitrogen and oxygen atoms creating a cascade of secondary particles [3]. The resulting particles from this interaction are protons, electrons, neutrons, heavy ions, muons, and pions[3]. According to barth, the neutrons are the most import result of this interaction. Neutrons are the dominant cause of failures in electronics in lower altitudes lower than 20 km. The neutron flux varies from 1 KeV to 100 MeV. According to Barth[3] neutrons with energies greater than 10 MeV are significant contributors to SEU.

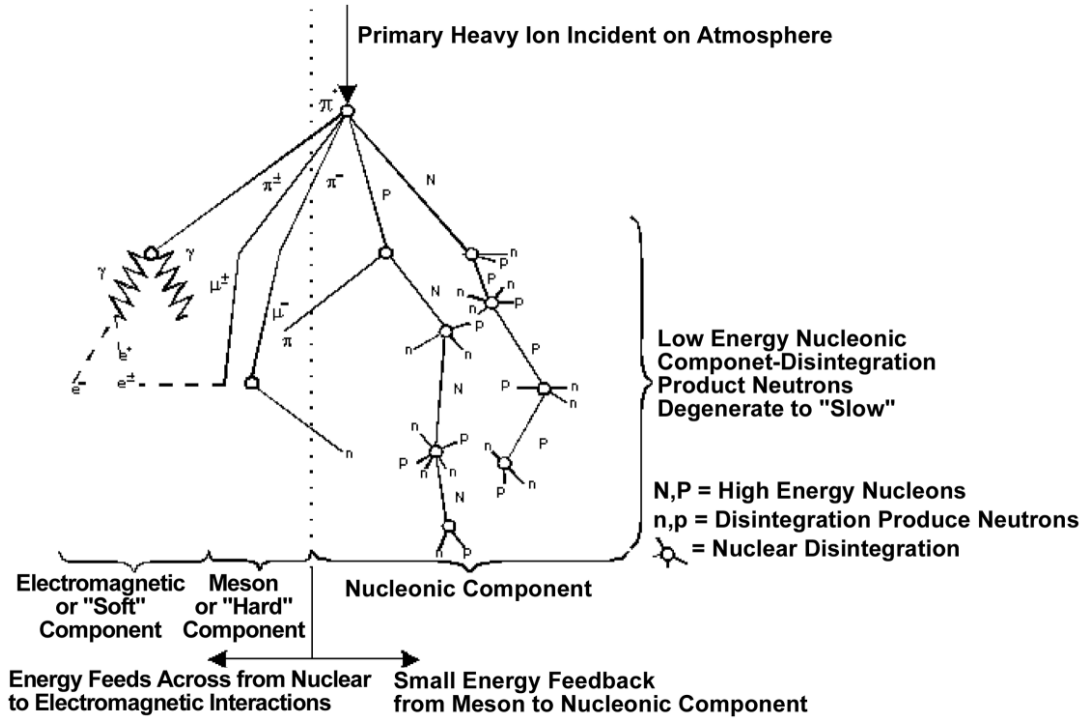


Figure 2.3: GCR and SEPs interaction with the atmosphere cause cascade of secondary particles.©IEEE June 2003[3] . Reprinted, with permission.

2.1.2 CHARM

Radiations environments in Conseil Européen pour la Recherche Nucléaire (CERN) facilities such as Cern High energy AcceleRator Mixed field/facility (CHARM) are populated with particles ranging from thermal energies to up to GeVs [11]. This mixed environment is composed of neutrons, protons, muons, pions, kaons, and leptons[11]. In relation to radiation effects in electronics, these particles are grouped intoHigh Energy Hadrons (HEH), which are hadrons [1] above 20 MeV, and thermal neutrons with energies around 0.025 eV[19].

This facility was built with the aim to provide a well-characterized mixed field radiation environment for testing electronics[11, 19]. The radiation field in the test area is adjustable by changing the shielding and target configuration[11]. The radiation field is generated by a proton beam of 24 GeV impinging on a target of various materials such as e.g. copper and aluminum[11]. The intensity of the radiation can be controlled by changing the intensity of the beam. The flux of HEH in CHARM ranges from a minimum to a maximum by merely varying test location, target, and beam intensity configuration[11].

[1]Hadrons are neutrons, protons, pions and kaons[11]

At Test location G0 is considered for the Device Under Test (DUT) placement at the facility Figure 2.1. The HEH flux is normalized at 100 MeV together with the flux expected in 600 km circular LEO orbit with a 98 inclination [20, 4, 19]. This orbit is the same as the orbit for the previous nanosatellite NORSAT-1 [21] and the most likely for the future missions. The energy spectrum for particles at the last mentioned position is representative of the LEO environment [20, 4, 19]. Figure 2.5 shows differential HEH spectrum obtain using FLUKA simulation tool together with data extracted using CREAM96 data [20, 4, 19]. The Annual levels fluences at LEO does not exceed 10^9 HEH cm^{-2} as shown in Table 2.1[11]. This can be achieved easily in test location G0 in less than a week of exposure time[11].

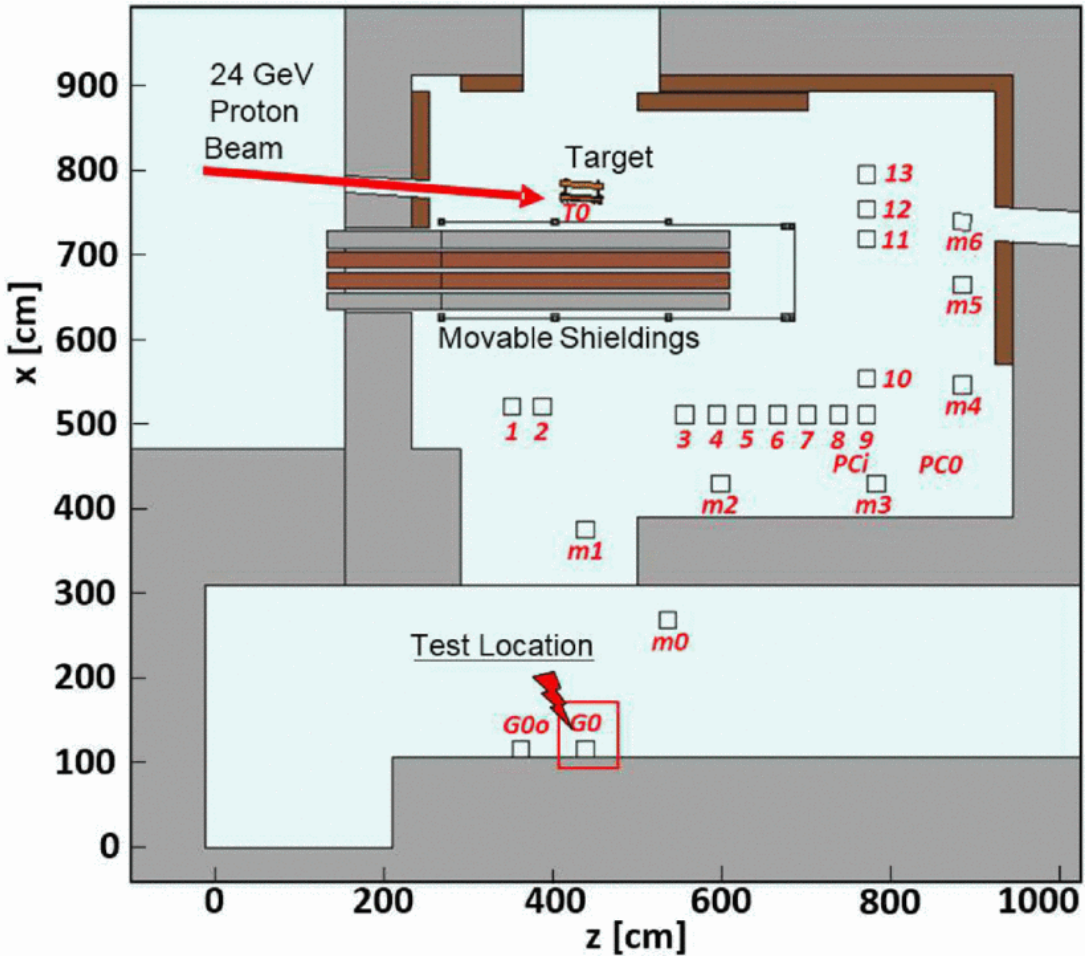


Figure 2.4: CHARM irradiation area and test location of our system ©IEEE Sept 2016[4] . Reprinted, with permission.

Table 2.1: Hadrons($>$ 20 MeV/cm^2/year) for different radiation environments. Data is from J.Mekki[11].

Spectrum	Ground level	Avionics	ISS	LHC Machine	LHC detectors
Flux	1-2e^5	2e^7	10^9	10^6 - 10^{11}	$>10^{11}$

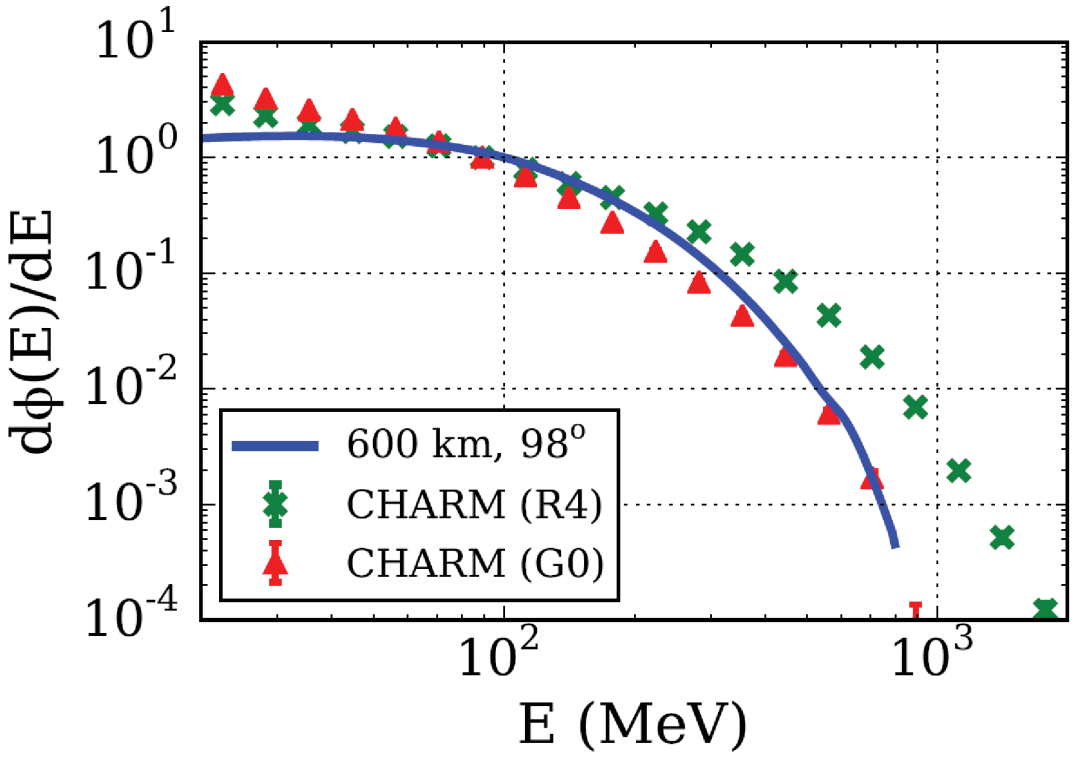


Figure 2.5: CHARM HEH flux normalized at 100 MeV for test G0 , and the expected HEH flux for a 600-km LEO orbit with 98 inclination. ©IEEE April 2017[4] . Reprinted, with permission.

2.2 Radiation Interaction

Radiation is an energy transfer mechanisms[22]. The transfer occurs either as an emission or transmission of energy in the form of electromagnetic waves or particles that passes through a medium[22]. In this thesis, we are interested in radiation sources that originate from an atomic or nuclear process. We use Glenn F.Knoll's[12] categorization which divides radiation into two main categories with

two subgroups for each Table 2.2.

Table 2.2: Glenn F.Knoll's[12] categorization of radiation that originates from an atomic or nuclear process

Charged particle radiation	Fast electrons
	Heavy charged particles
Uncharged radiation	Electromagnetic radiation
	Neutrons

Knoll[12] divides radiation into charged particulate radiation and uncharged radiation. The charged particulate radiation interact with material via Coulomb forces. This category is further divided into two subsections fast electrons and heavy charged particles. The uncharged radiation does not interact using Coulomb force with the material, but rather via other mechanisms. This group contains neutrons and electromagnetic radiation. In this thesis, we are interested in heavy charged particles and neutrons.

The radiation ability to penetrate thickness is a key factor when examine effects of radiations on matter whether it is electronics components or biological. A radiation source that has a high penetration is call hard radiation and low penetration ability is called soft[12]. this terminology is useful when comparing two particle penetrability of material.

Heavy Charged Particles

Heavy charged particles include all charged particles that have a mass of 1 atomic mass unit or higher[12]. this includes protons, alpha particles and fission fragments[12]. These particles are capable of causing Single Event Effects (SEE)[7]. They are found naturally in space (Section 2.1.1) and impurities in device package[6]. These charged particles interaction with materials via direct ionization. The energy transfer is dominated by electronic (Coulomb) interaction[19]. This interaction can results in excitation or ionization of atomic electrons[12]. When the ionization occurs in electronics devices it cause an effect know as SEE as discussed in Section 2.4.

Neutrons

Neutrons generated in various nuclear processes. They found both in LHC and atmosphere[6, 19]. They are further divided into slow and fast neutrons

subcategories[12]. The neutron ability of interaction with material is dependent on it kinetic energy. Neutrons interact with matter through indirect ionization which is know asNon-Ionizing Energy-Loss (NIEL) [6]. This is a nuclear collection with can be elastic or inelatic. The Most important is inelastic collisions [8, 23]. When neutrons collide with Si it does initiate a nuclear reaction, resulting in emissions of secondary nuclear fragments[24]. This fragments are usually a lighter ions protons,and alpha particles with can directly ionize the silicon [24, 23].

Slow neutrons usually are not issues in causing SEE in electronics. There kinect energy is below 1 MeV [6]. This role becomes important when interacting with Boron-10. Boron is used as p-type doping in silicon or dielectric layer (BPSG)[6]. the interaction of a slow neutron with Boron-10 is, and it produces fragments of lithium-7 and alpha particles with are both capable of causing SEE[6].

As Neutrons, the rest of other hadrons such as protons, pions, and muons transfer energy by indirect ionization. Protons can interact both by direct and indirect ionization. For protons with energies above 50 MeV they are similar to neutrons and the dominant interaction is indirect ionization[25, 23].

2.2.1 Units

The unit for measuring radiation energy is electron volt(eV) [12]. It is defined as the kinetic energy gain by an electron that is accelerated through a potential difference of 1 volt. For example, if we take a particle with a net charge of +1 and accelerate it through 1000 volts the energy gain by the particle will be 1KeV. The equivalent value of electron volts in joule(J) is

$$1eV = 1.602 \times 10^{-19} J \tag{2.1}$$

The activity of radioisotope[12] is defined to be the rate of decay and is given by the fundamental law of radioactive decay. It measured using Si unit Bq which is a unit that measures the rate of disintegration of a radioactive object per second. However, this does not tell us about the energy deposited into a material.

Absorbed dose is the measurement of energy deposited by a radiation source per unit mass of the martial[12], the units used are Gray or Rad. These two units are the measurements of energy deposited in the material by the particle. Even if both units common in literature it is important to note that Gray(Gy=$J \cdot Kg^{-1}$) is an SI unit.

$$1Gy = 100rad \tag{2.2}$$

2.2.2 Stopping Power

The total stopping power is defined as the rate of energy loss by particle per unit distance[12, 24, 23]. The units of stopping power is MeV/cm [12]. The

energy transfer to the material is due to electronic(Coulomb) and nuclear(elastic or inelastic) contributions[19]. The stopping power is described using the Beth's formula [12]. This formula is valid for heavy charged particles with energies that is greater than 1 MeV/amu [19]. In expression B equation (2.4) the Z (atomic number of absorb atoms) is only significant value for a none relativistic charged particles. This because the second term in B equation (2.4) will vary slowly with increase of particle velocity/energy[12]. When talking about the absorber material the significant values are the product of N(number of density of absorber atoms)and Z.

For the same absorbent two factors describe the stopping power of particles, velocity and the charge of the incident particle equation (2.5). Particles with a larger z have greater stopping power, thus having a greater ionizing ability[12].

The stopping power is inversely proportional to particle velocity/energy[12]. As the particle propagate throw the absorber, it will low energy/velocity. When enough velocity is lost the charged particles, spend more time close to electrons. This allows a higher energy transfer to electrons.

$$S = -\frac{\delta E}{\delta x} = \frac{4\pi e^4 z^2}{m_0 v^2} NB \tag{2.3}$$

$$B \equiv Z[ln\frac{2m_0 v^2}{I} - ln(1 - \frac{v^2}{c^2}) - \frac{v^2}{c^2}] \tag{2.4}$$

$$S \propto \frac{z^2}{v^2} \tag{2.5}$$

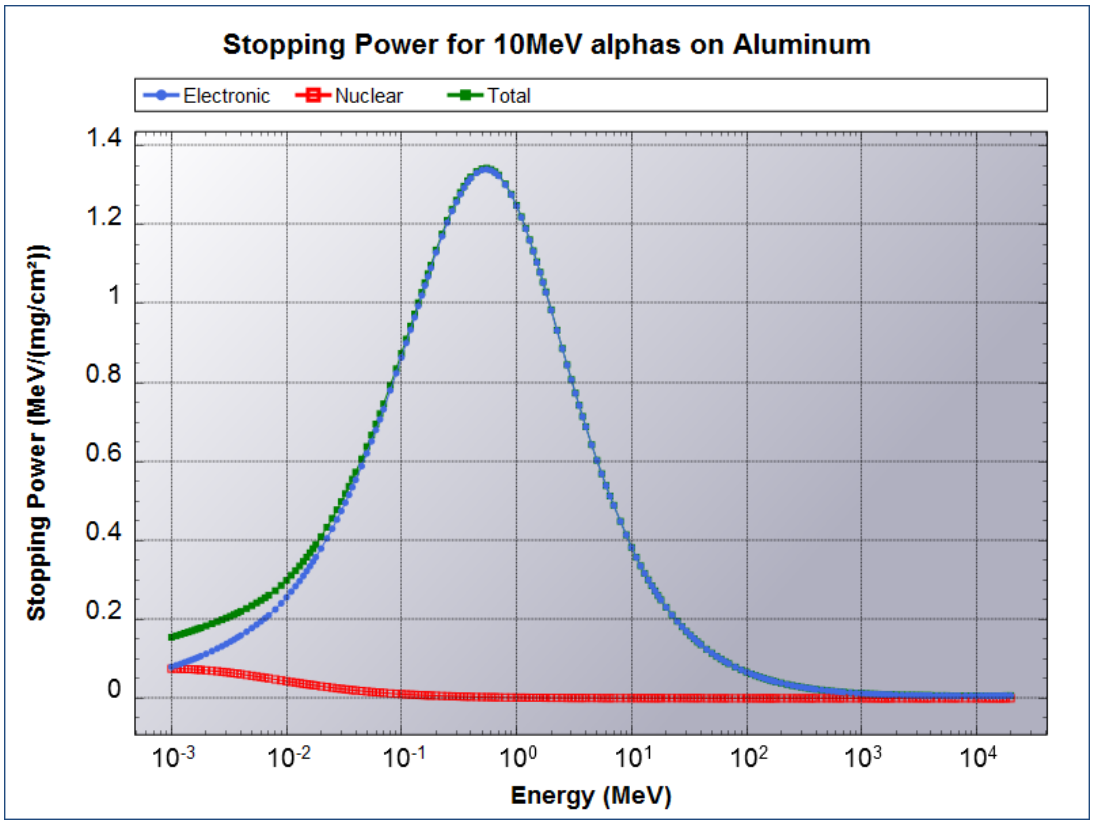


Figure 2.6: Bragg's curves for alpha particles. This plots are generated using Nucleonica[5] web portal for nuclear data.

2.2.3 Linear Energy Transfer

Linear Energy Transfere (LET) is local energy deposited along the particle track[12] and the stopping power is total energy loss [19].In literature LET and stopping power is used interchangeably. For heavy ions, they are nearly equal since predominant energy loss is the electronic contribution, excluding Delta electrons and bremsstrahlung effects.

$$LET = \frac{1}{\rho} \cdot \frac{\delta E}{\delta x} \tag{2.6}$$

2.2.4 CMOS technology

Most used technology today for constructing electronic components is CMOS. This construction technique used n-type and p-type MOSFET transistors. In silicon substrate one electron hole pair is produced for every 3.6 eV of energy lost

by ion.[6]. According to Baumann [6] the radiation interaction with p-n junction is three-phase process as seen in Figure 2.7.

(a) When ionizing radiation passes, throw a p-n junction. Along the path, it will create a set of holes and electrons.

(b) Free electrons are collected rapidly by an electric field. A funnel is created which extended the depletion region deep into the substrate. This increases the collection area of electrons. A more extensive collection area means a higher current spike. The duration of this process lasts only for a few nanoseconds.

(c)After this, a slow phase which lasts in hundreds of nanoseconds until all carrier has been collected, recombined or diffused away from the junction.

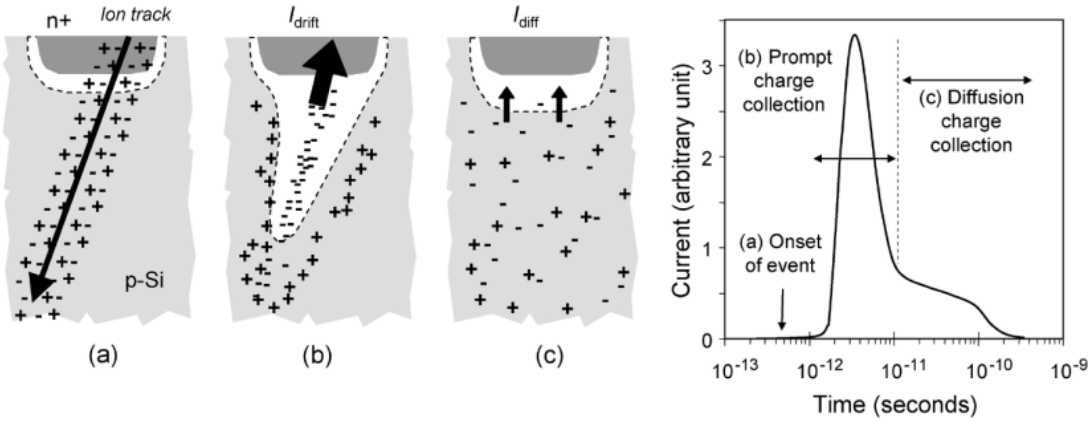


Figure 2.7: Radiation interaction with p-n junction. ©IEEE April 2017 [6] . Reprinted, with permission.

2.3 Cumulative Effects

As the name suggests, these effects are because of successive additions, in our case radiation interaction with the material. It is a long-term exposure of electronic components to radiation that introduces defects to the material which in turn changes the electrical properties of the device this effects can be attenuated using shielding to the devices operating in a radiation environment[7]. The categorized into Total Ionizing Dose and Displacement Damage[26, 7].

Total Ionizing Dose is due to the accumulated exposure to ionizing radiation[7]. The prolonged exposure to the ionizing dose alters the properties of dielectric materials, such as the one found in CMOS based transistors[7]. This material plays a role of electrically isolate the gate from the channel. When a voltage is applied to the gate an electrical field open channel allowing electrons to flow between the drain and source of the transistor. An ionizing energy interaction with dielectric

insulator results in holes trapped into the gate oxide, this causes a change in the voltage requires to change the state of the transistor[26, 7]. This build-up of the positively trapped particles can cause NPN transistor to get stuck in on-state even if the gate voltage is zero[26, 7].

Displacement Damage is due to accumulating of NIEL (non-ionizing energy-loss) interaction. Fluences is the unit used to measure the displacement damage in a device[26, 7]. Defects introduced to the material is due to nuclear interactions as mentioned earlier interacts with the material through elastic and inelastic collisions these interactions displace atoms from it lattice position[26, 7]. In optoelectronics, this displacement reduces the sensitivity of the device to photons. This alteration of the material causes degradation of the current transfer ratio (CTR) of the device[7].

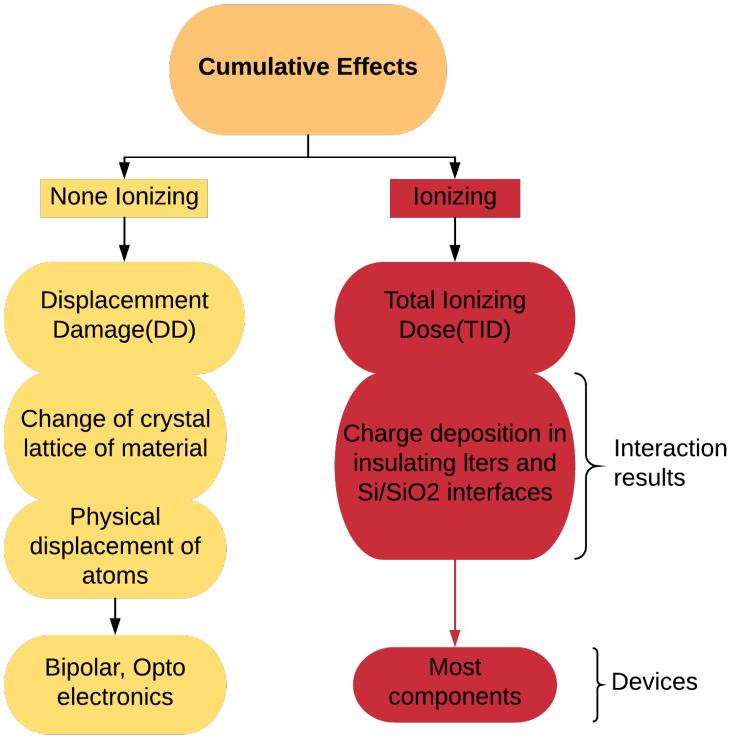


Figure 2.8: An overview cumulative effects [7]

2.4 Single Event Effects

The single event effects (SEEs) are disruption caused by a single particles ionizing a sensitive volume of an electronic device[8, 7]. The sensitive volume is the area that ionizing particles have to strike to cause a hard or soft error[8, 7]. SEEs has a stochastic nature, which is a statistical approach to a single particle to strike a sensitive volume [8]. This probability is expressed in cross-section[8, 7]. These errors are induced by heavy ions,protons, and neutrons [8]. The categorization is divided between hard and soft errors[7, 8]. In one hand soft errors are compromised of failures that are recoverable by a reset or a rewrite, in other hand, hard errors are none recoverable that may result in erroneous operation or permanent damage to the component[7].

Soft errors are the nondestructive errors induced in the electronics[7, 8]. Even if there are not damaging the component, they can compromise the reliability of the device in question[7]. This reliability is in term of data or functionality of the device. Soft errors, in turn, are divided into subcategories[8]. Each subcategory is based on the effects caused by bit flips on a device.

- Single Event Upset (SEU)
- Single Event Functional Interrupt (SEFI)
- Single Event Transient (SET)
- Multiple Bit Upset (SBU)
- Single Bit Upset (MBU)
- Multiple Cell Upset (MCU)

Hard errors are destructive effects caused by particles in electronics[7, 8]. The deposited energy by particle requires to induce hard errors tend to be higher than the energy for SEU[7]. They are often caused by heavy particles[7]. When considering component to operate in a radiation environment, it is crucial to find the threshold of the destructive event occurring in that device[7]. The subcategorization is based on technology used for constructing electronic component[7].

- Single Event Burnout (SEB) occurring in power MOSFET,BJT(IGBT) and power diodes.
- Single Event Gate Rupture (SEGR) occurring in power MOSFETs.
- Single Event Latch up (SEL) occurring in CMOS.

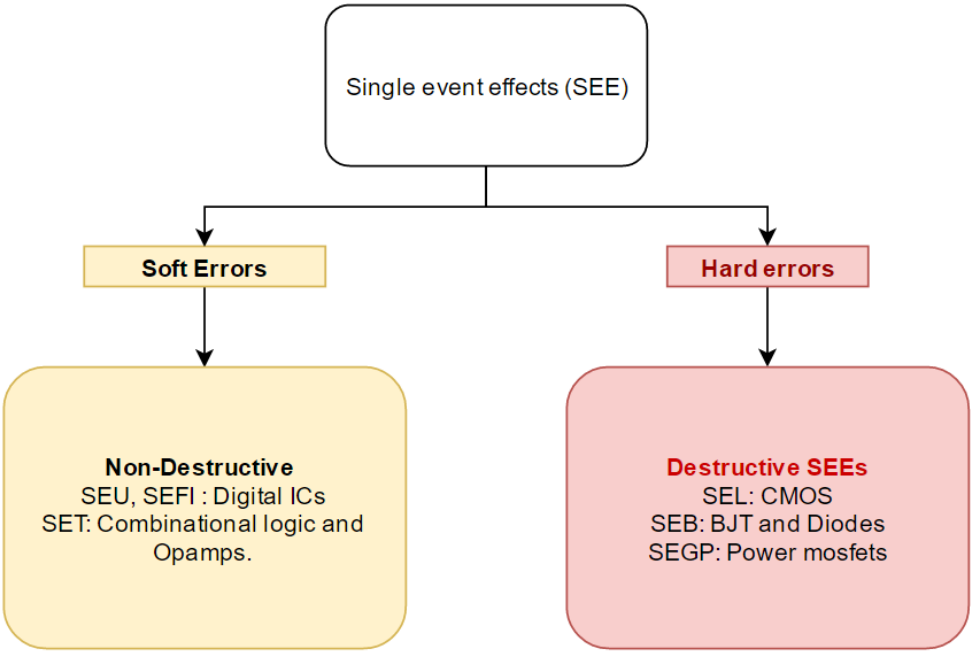


Figure 2.9: An overview over the single event effects categorization[7]

2.5 Soft Errors

It is crucial to understand the categorization by the device to understand Single Event Upset (SEU). A bit-flip in memory may go unnoticed if it happens in a measured value[8]. It different story if the same happens in control data stored in the SRAM. This can result in the system to crash or enter an undefined state[8].

SEU is the alteration of stored data by radiation event[7, 8] . This term is used to cover Single Bit Upset (SBU),Multiple Bit Upset (MBU), and Multiple Cell Upset (MCU).

- SBU a particle causes a single bit flip in the memory cell.
- MBU an event caused two or more bit flips in a memory cell.
- MCU when an even causes upset of two or more memory cells or latch.

SEFI is a special of SEU since it happens when devices such as microcontrollers stop responding or enter an undefined state of operation[8, 7]. This can happen when a bit flip occurs in control registers[8]. The only way to get the system back to normal operation is to rest the device[8, 7, 24]. That is why often it is

advised to use an external watchdog to reset the microcontroller if no response is detected[7].

Single Event Transiant (SET) is a voltage spike generated via particle strike near a sensitive area[8, 7]. This event cause voltage to propagate through an analog device and combinatorial logic[8, 7]. If SET results in a wrong value been latched, then it is considered SEU[8].

As explained in Nicolaidis[8] the ability of radiation to alter the state of a transistor is dependent on the critical charge of the transistor. This is the minimum charge of the devices requires to change its state. The simplest approximation is to consider the total capacity Ci at the node and the supply voltage Vdd. For CMOS, the total capacitance at the node is the sum of capacitance in drain-substrate and drain-substrate capacity[8]. When charge deposited by radiation in a sensitive area is larger than the critical charge a change of the devices state occurs.

$$Qc = Ci \cdot Vdd \tag{2.7}$$

The presence of a high-Z material like tungsten increase the SEU in devices[27]. This increases as well the MCU cross section for devices when exposed to neutrons with energies greater then 100MeV[27].

2.5.1 6T Static Random-Access Memory (SRAM)

According to Nicolaidis [8] when enough energy is deposited by hadrons or heavy particles in the sensitive area of a MOS transistor, it will result in a change of state. In the case of the 6T cell, a change of the state of one cell results in a wrong value to latch into the cell. For this scenario to occur, the energy collected has to be greater than the critical charge of the device. If we consider that Node A = 1 in Figure 2.10 , The 6T cell design has a feedback loop this used to store the value in the cell. When radiation alters the node A from 1 to 0, this feedback loop reinforces a new value.

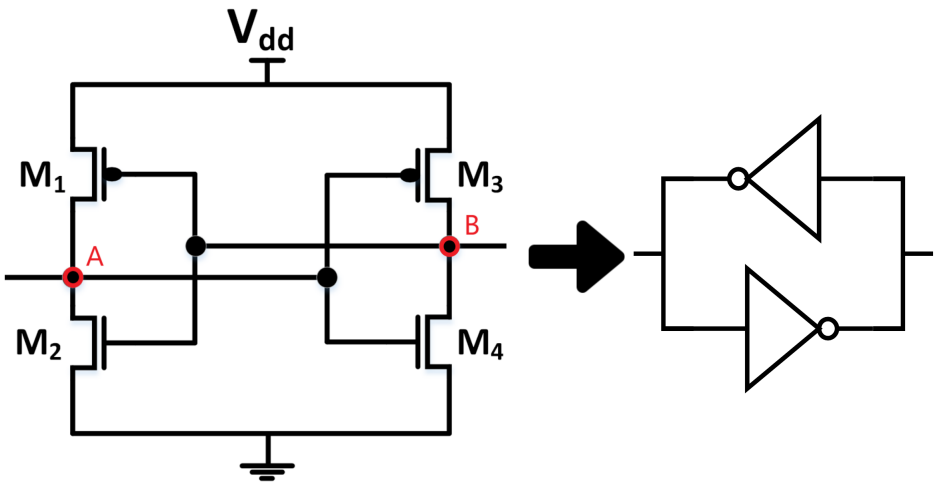


Figure 2.10: Transistor schematic and logic gates for a 6T SRAM cell [8]

2.5.2 Mixed-signal integrated devices

Mixed-signal devices contain both digital and analog circuit. Complex devices like microcontrollers are mostly digital. However, these devices contain analog circuits like ADC. SET is an effect commonly seen in combinatorial logic and analog circuits [8]. According to Nicolaidis[8] for SET to be considered a SEU the particles have to cause a transient able to propagate through the circuit to the memory element. Transient pulse has to have enough amplitude and duration to change the state of the latch. The transient has to arrive at the latching edge of a clock. The higher the operating frequency, the greater the probability of a transient been latch.

2.6 Single Event Latchup (SEL)

SEL is a destructive event that occurs in CMOS technologies[7, 8]. Latch-up initiates by ion striking the sensitive area. If enough charge is free to turn on one of the parasitic BJT structures, it will cause an increase in current drawn by the cell. If going unchecked the current will rise until the device is damaged[8]. This can be interrupted by cutting the power supply to the device.

SEL is not only caused by heavy ions. In the case of the complex devices like microcontrollers, The has been a few cases where SEL is observed in Hadron environment [7]. However, Tungsten near the sensitive areas can introduce a strong

SEL cross section [28]. J.R. Schwank[29] observed for some SRAM are sensitive to SEL. He showed that proton interaction with silicon could not generate nuclear recoils with enough LET to induces SEL. Both papers [28, 29] determines that proton interaction with tungsten plugs in most high-density Integrated Circuit (IC) are the primary cause of SEL in hadron environment.

Paul E.Dodd[30] has shown that latch-up is highly dependent on temperature and power supply dependent. A high temperature or power supply increases the SEL cross section in the device.

2.7 Mitigation

One approach to reducing soft errors in a system is by removing the sources of errors. Baumann [6] showed that semiconductors containing Boron-10 are sensitive to low energy neutrons. By removing Boron-10, the amount of bit flips caused by thermal neutrons are eliminated[6]. Purification of materials can reduce alpha contamination in solder causing the SEU in ICs. Producers results in methods of keeping materials with high alpha particles away by physically separating from the sensitive circuit[6]. All those methods are mitigations can be done by the IC developers. From a user, perspective it is challenging to access information on what producers have done in-process level.

In the case of COTS both concerning hard and soft errors, one has to manage error risk to the system. In a radiation environment, high energy particles that can penetrate shield and package as mention in Section 2.1. Manufacturer of COTS uses different methods to increase the reliability of their products. The approach to mitigate the soft errors are more highly dependent on a device in question. Complex devices such as microcontroller would have a broader mix of different methods to ensure the reliability of the circuit[10]. The mitigation techniques are categories into process, circuit redundancy, and recovery.

Process and Layout

Generally speaking, when talking about a process level the idea is limiting the charge collection in substrate[6, 17] this is done by using process technologies that Silicon on insulator or epitaxial substrates both limit collection areas of charge[6, 17]. There is another counter-measurement that are used to reduce further the rate of SEE. This is out the scope of the thesis. For the interested check "space product assurance" [17].

Mitigation by Circuit Redundancy

Redundancy can be increased by using radiation hardened circuit design[8, 17]. Those libraries are made out of components can resist change by increasing circuitry redundancy[17]. This limits the impact of errors on the system[7]. The redundancy is implemented as triple modular redundancy (TMR) with a concept of a majority voter[7]. This method requires at least two voters to have the same value to work. This can increase the reliability of the circuit as well decrease SET that will latch a wrong value to a latch.

Mitigation by Recovery

This two examples of recovery methods parity check and watchdog timer.

Parity Check

Parity is a bit added to the binary code to increase the reliability of the data transmitted [31]. The received and transmitted are configured to have the same parity type even or odd. The parity bit is used by the receiver to check that all packages are correct. The error can be detected but not correct. The parity can be implemented on hardware or software level[31]. According to Kenneth A.Label[31], other algorithms use the same concept of the parity. This algorithms such as Hamming code can detect double errors and correct single ones. Another algorithm is Cyclic redundancy check (CRC) used to detect an error. The beauty of this method is that it can be used as a check for a large of data packages.

Watchdog Timer

The watchdog timer can be implemented either in hardware, or software, [31]. They can be either integrated internally or externally[31]. A watchdog timer is a recovery technique that can be used to increase the reliability of a device such as microcontrollers. This method reliable periodically single send by the CPU to a watcher dog timer, this signal indicates that the devices I stilling working within the time windows desired [32, 10]. If the signal is not received within the time frame from the microcontroller the watchdog timer takes action to recover the functionality[10]. This action can be a reset signal issued to restart the microcontroller. This device is handy to deal with Single Event Functional Interrupt (SEFI) special when physical access to the devices is not possible[31].

2.8 Cross section

According to Nicolaidis[8] SEE cross section is the probability of that incoming particles will induce an SEE. Cross section is determined by the ratio of the number events observable and,F is particle fluence multiplied by total bits monitored. The unit of cross sections is [cm2/bit] or [cm2/device]. Fluence total number of particles passing a unit of surface area in a given interval of time[particles/cm2]

$$\sigma = \frac{N}{f * bits} \tag{2.8}$$

$$Uncertainty = \frac{\sqrt{N_{events}}}{N_{events}} \tag{2.9}$$

2.8.1 Test SEU memory

According to Nicolaidis [8] memory testing is done by writing a known value to the memory. During the irradiation of the devices, the values are read from the memory to determine the number of single bit flips that occurred. This count of values along with the fluences the devices have been subjected to determines the sensitivity of the SRAM.

2.9 Future Trends

According to [8, 30]SEU effects worsen with the technology scaling where the electronics industry is decreasing device dimensions and increases the number of transistors per chip. A result of this trend is the decrease in the supply voltage, and the node capacitance. For space environment, this means that the charge necessary to induce an SEU has been decreased as well. For SEL the trend of reduction of the power supply reduces the sensitivity of the circuits.

2.10 Conclusion

Understanding the radiation environment and interactions with electronics are essential when using COTS in spaces. It requires that designers of hardware or firmware to identify the components that have mitigation features that can be utilized to reduce the impact of radiation induce failures. This increases system reliability and the success of the mission.

3 | Decapping

The chapter is documenting decapping of a TMS570 microcontroller and AD7768 Analog-to-Digital Converter (ADC). We start by explaining the background information and the relevance to environment radiation verification of COTS. We show the different methods available, and the one choosen base on the resources available at UIO. The chapter provides an insight into the process of decapping and the steps needed.

3.1 Motivation and Goal

It is important to test complex microcontroller COTS to determind LET threshold of destructive events caused by heavy charged particles [17]. Heavy ion ground level testing is performed in vacuum[17],as explained in Section 2.2 Heavy ions are directly ionizing. They lose energy as they propagate through materials including air and devices packaging. Decapping ensures that ionization energy loss occurs in the die. That helps us correctly determine LET threshold required to induce SEL in Device Under Test (DUT). From that, we can obtain a cross section curve used to estimate the SEL rate that the devices will experience in the environment[7]. The failure analysis of COTS increases the confidence level on whether the device suitable for this radiation environment. The goal is to explore and share the method used to expose TMS570 die.

3.2 Background information

Decapping is a process to remove packaging to IC in order to expose the die. In our case this enables us to irradiate the device either using laser or heavy ions. IC package for TMS570 and AD7768 are both QFP packages in different sizes. It is not easy to tell what kind of materials are used by just looking at those packages. Generally speaking, QFPs package are made using process of moulding where it covers wire bonding and die [33]. The die is glued into a copper plate called lead frame which serves as a structural and thermal. Most of

the manufacturers do not disclose the type of packaging used. Texas instruments producer behind the TMS570 microcontroller. They disclose a list of materials used in there component as there initiative to use greener materials[34]. Analog the owner of AD7768 ADC. They do not disclose any information about the materials used in their products. Even with information from Texas instruments, it is nearly impossible to determine what kind of package is in use. However, We can see that copper is used for bonding wire[34]. Package wise the only thing that can be said is it is. It is a molded Epoxy based packaging [33]. We follow methods done by Matthew J. [35]and Huiwei [36] as this two show how the shift from gold to other metals has made it challenging to use wet etching. Gold is inert and does not corrode during wet decapping. Unlike gold, bonding wires made of Copper, silver, and aluminum are easily corroded by wet etching.

Wet etching is dependent on solution temperature, exposure time, and the surface state of the wires (light coat such as Pd on wires to reduce corrosion) [35, 36]. Those are important as acids attack copper bonding wires. The overexposure to acids such as nitric acid can cause wire thinning or pitting, and bond pad corrosion, all which impacts the mechanical and electrical properties of circuit[35]. Both Matthew J. and Huiwei are using the nitric acid, or a nitric, sulfuric acid mix is used [35]. The process starts decapping with pre-cavity by a mechanical drill or lase, followed by an acid etching to expose the die[35, 36].

3.2.1 Environment, Health and Safety (EHS)

It is essential to take precautions when working with lab graded chemicals. This especially the case when handling acids. Those acids have a concentration higher than 90% precaution, and correct handling is essential. We followed UIO guidelines for safety when working in this experiment[37]. A background reading was done to understand the results of reactions to ensure safety. When the decision reached on the method, were discussed with the EHS coordinator in the chemistry department. From those meetings, we discussed which chemical that will be used and execution order. It is to ensure that reaction that will take place is safe.

The following precautions were taken when working with acid.

- Flammable chemicals have to be kept in hazardous storage cabinets.
- All the experiments were executed under a fume hood.
- While the reaction is taking place, the fume hood has to be closed, and airflow set to Max.
- We did not work directly from large bottles of acid. We transported chemical using pipes to smaller containers.

- Reduces risk of spill and limit chemical fumes by closing the flasks, and containers.
- Gloves do not stop acids they give time to remove them.
- Safety glasses to protect the eyes.
- Lab-coats (Gives you time to remove).
- Equipment that will be in contact with acid were made from glass.
- None glass materials can only be used after acid is neutralized with acetone e.g. stainless steel Tweezers.
- Be aware of placements of fire equipment and chemical treatments stations.

3.2.2 Requirements

The requirements was set based on the safety requirements and availability of tools at UIO.

Table 3.1: The requirements of decapping processing

Requirements for decapping
Decapping in room temperature.
Using locally availible resources at UIO.
Develop process that can be followed in future.
Removing package without damaing IC.
Functional testing of IC

3.3 Method

Decapping was started by sending TMS570 to FFI for a 3D Xray. This analysis helps us determine the placement of the dies, size, and bonding wire height. Knowing the area the bonding wire is placed help use determine the depth of cavity and surface area that need to milled in the package. This model from FFI helps to have a 3D model where you can measure distances accurately. After

determining the position. The IC was given to I-LAB with measurement of depth and area of the cavity. When the cavities received back, the IC was optically analyzed. IF we found no damage, one could continue to wet etching in the lab. The wet etching is time-consuming as it is slow and demands optical inspection after each amount of steps. If the sample if pass the visual inspection the IC then is cleaned and then we are done. The same processes from Mechanical removal were done to do all. AD7768 was never X-rayed at FFI. An overview of the steps of the decapping process is seen in Figure 3.1.

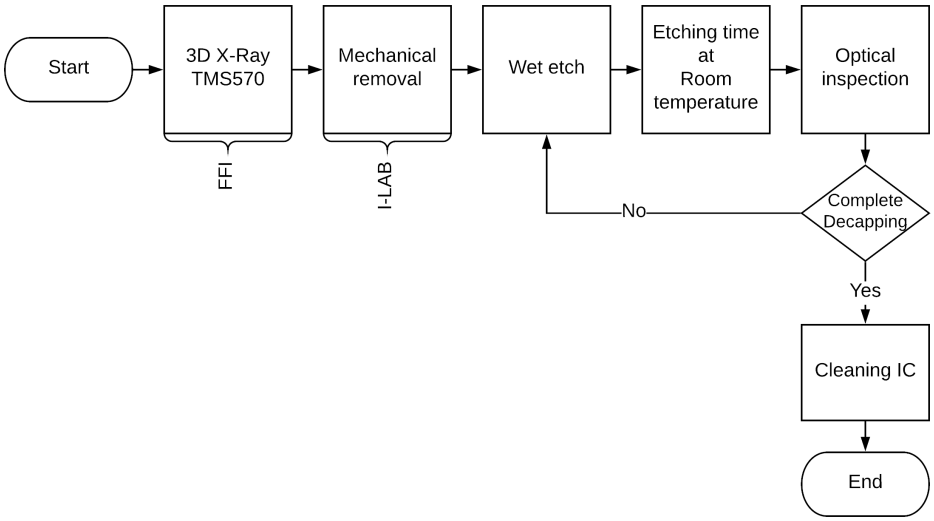


Figure 3.1: The procedure followed for decapping.

3.3.1 3D Xray

At FFI the engineers took an initial scan to find the best optimal way to show the wire bonding in the IC package. After multiple scans, it was clear that only a small area could be scanned to reveal the wire bonding.

The small surface area as seen from Figure 3.2 resulted in the left side is a bit been shorter than the right side, that is because the IC was not center correctly. Taking direct measurements from lead frame is nearly impossible to get a precise reading from it. We can measure the die area and depth we want to remove the package mechanically then do the rest with a wet chemical etching. We made the assumptions that the die is centered in the lead frame and that all die are placed in same precision by Texas Instruments.

The way we can go about it by using the corners of IC to find the center of the package. The center of the package will be used as a reference point to

the CNC machine. This, not the ideal way of doing it but this can give us an approximation of a start point where we can start drilling. One could assume that 3D x-ray scan reveals the letters on the surface on the package which could be used as a reference for the drill points. This latter method has even more limitations since some letter on the package changes. That makes it even harder to have repeatability in making cavities.

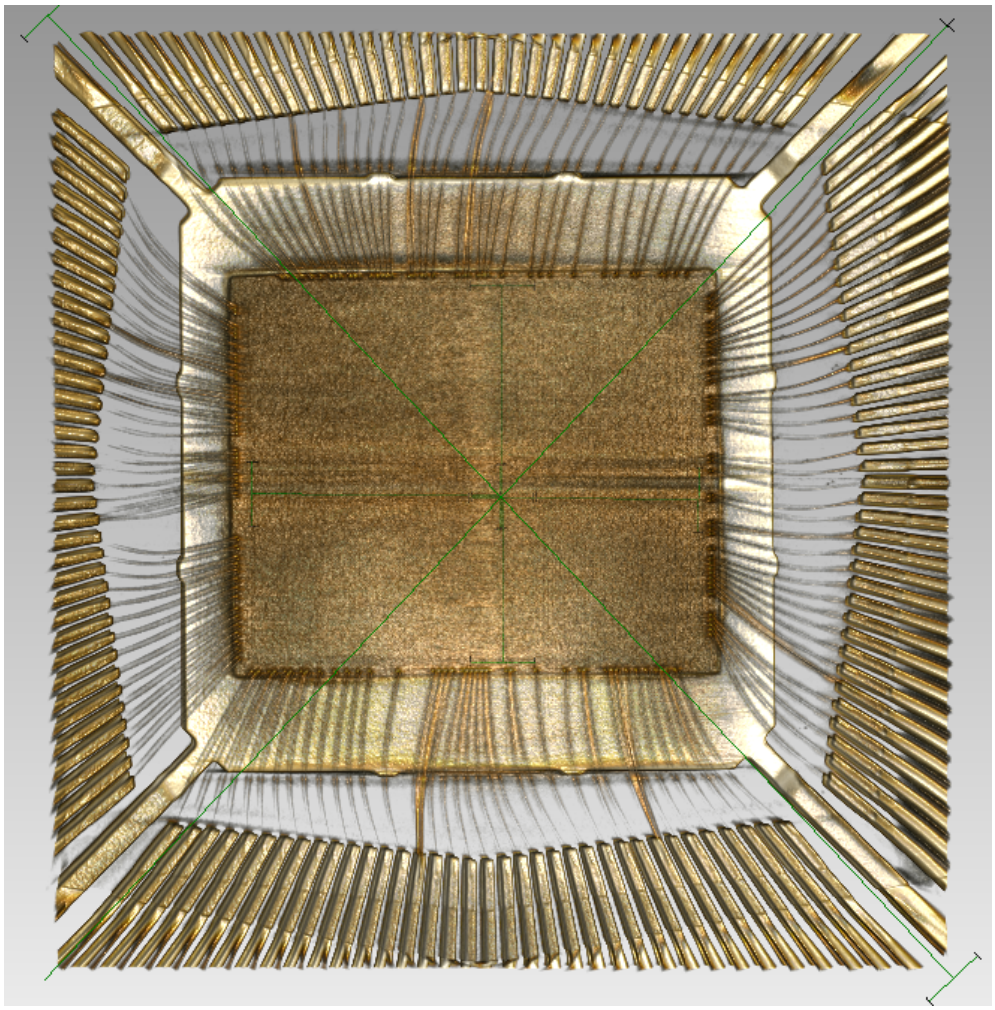


Figure 3.2: 3D X-ray of TMS570.

Measurements for CNC machines

In Figure 3.3 show the maximum distance relative to the center of the die. The CNC machine needs to remove according to this values. From center 2.80 mm

from left and right and 2.28 mm top and bottom. Those are the absolute maximum values, so the cavity won't cause damages the bonding wires. That is assuming that the CNC machine has a high accuracy in the posting. If we stay within the parameter described above, we can remove 0.5 mm in depth from the package without damaging the IC. This as well the absolute maximum

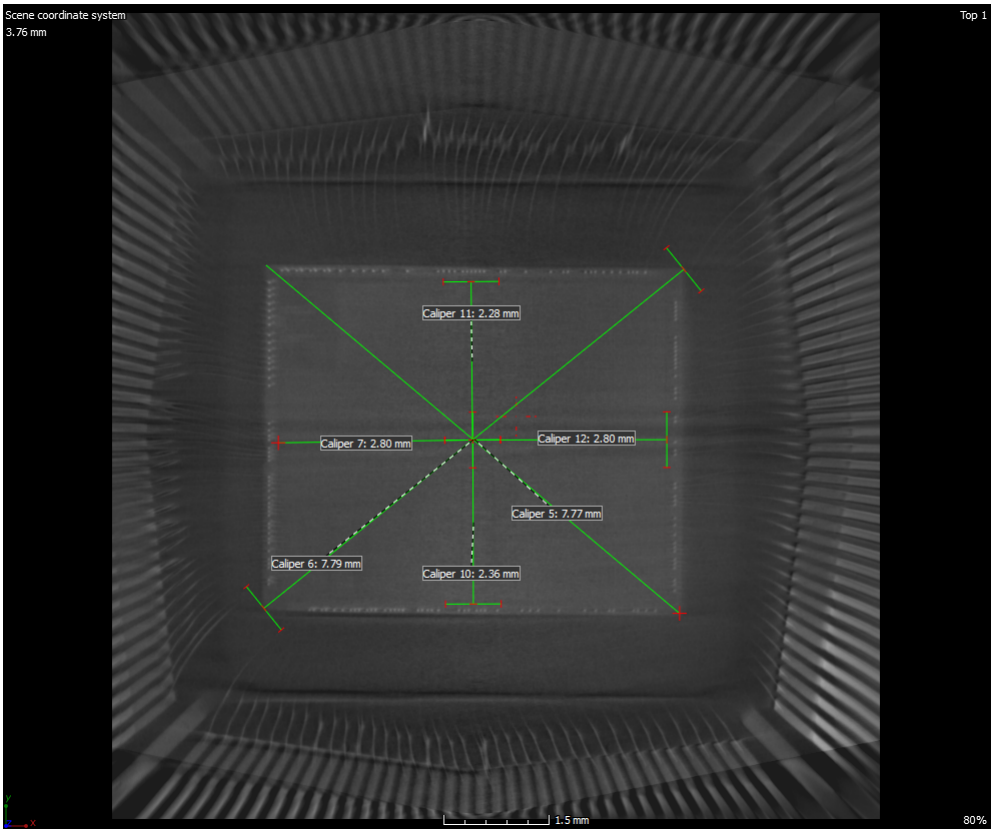


Figure 3.3: Measurements that give to I-LAB for CNC of device.

3.3.2 Mechanical CNC execution

The measurements were given to the mechanical workshop at the physics department. The CNC process needed to have solutions that amount the IC in an amount to hold it in place during the CNC process. The mount role has to hold the IC steady in place. It ensures that measurements are correct and not change during the process. The as well prevent pins from getting stressed during the process. The plate that is screw down is used as a reference point to the area the drill needed to move in. This hopefully will reduce the damage to the bonding wires. The metal mount has groves for pins so that they slid along sides. The

center pins had to endure a bit of friction. Pushing in the IC had to be done with care.

The holding metal block held tightly in the CNC machine. The CNC machine is hand operated in 3 axes with 3 turning wheels that are turned by the operator. The precision is in 1/100 mm according to the operator. The operator centered the drill head on the center of the IC then he started moving with the measurements to creating a "well" in the package.

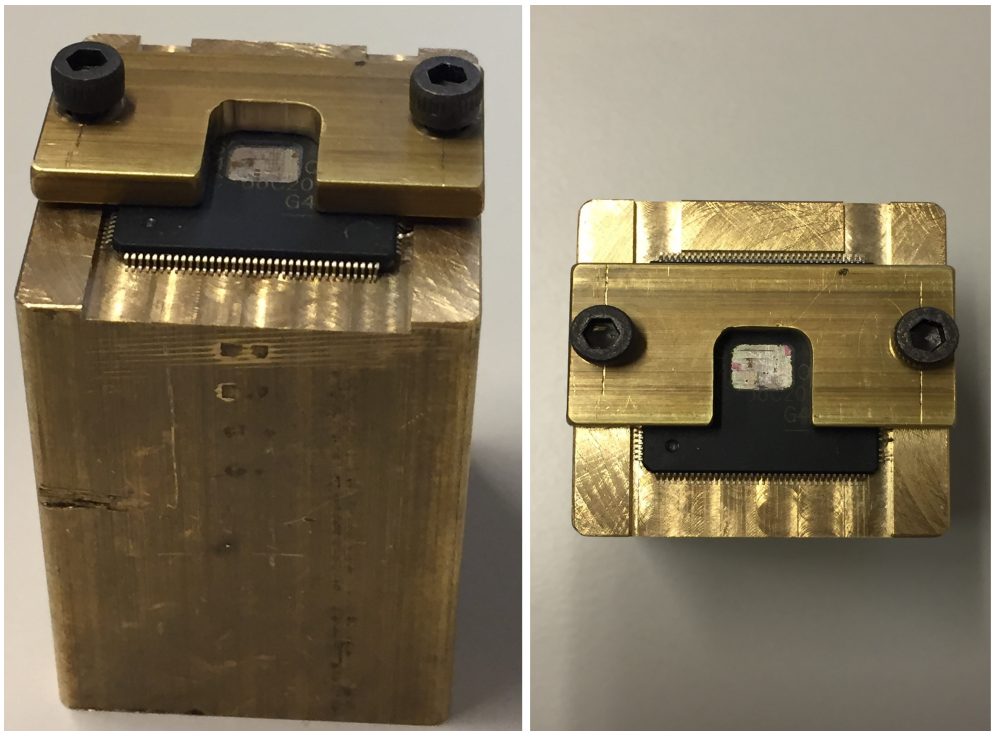


Figure 3.4: Measurements that give to I-LAB for CNC of device.

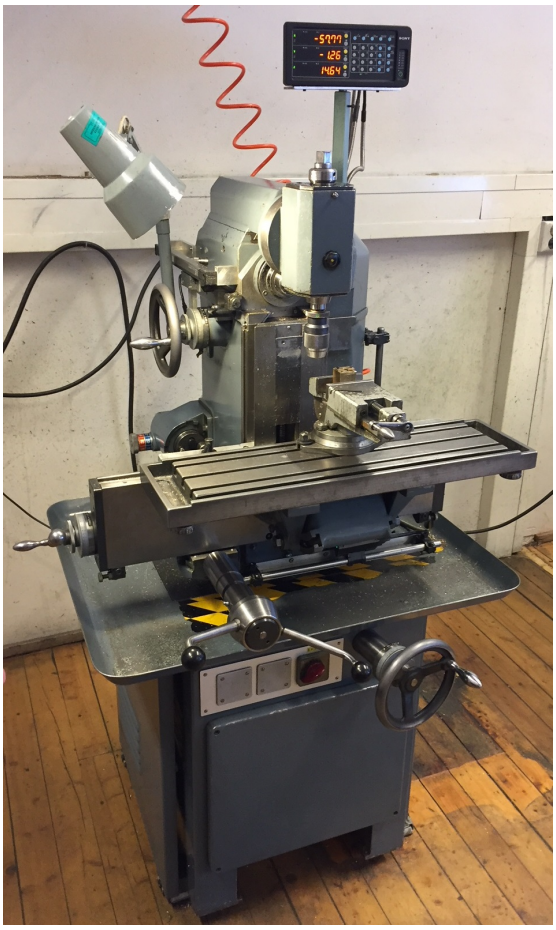


Figure 3.5: Measurements that give to I-LAB for CNC of device.

3.3.3 Wet Etch

The wet etch is done as seen in the Figure 3.6. It is done experimentally by first finding the time need to remove the package. Removing the package without corroding the bonding wires is a must since we intend to mount the IC on PCB. The solutions mix chosen is three drops of nitric acid to one drop of sulfuric acid [35]. The use of sulfuric acid is to dilute nitric acid. Those two acids do not react which each other. We did some tests with just sulfuric acid and noticed nothing happens to package beside a slight discoloration. The challenge was to decapped with three uncontrollable variables as temperature, solution delivery and mixing exposure times. All the experiments were done at room temperature. One of the idea that we tested was to head the devices into a controlled temperature, but that quickly found to be difficult. The solutions delivery method was using a glass pipette with a rubber bulb. When adding pressure on the bulb acids drops

are delivered on the IC surface. The pipettes were used for mixing the solutions three drops of nitric acid and one drop of sulfuric acid. The exposure time started when the last drop of sulfuric acid was added. During the experiment, the timing of acid exposure was adjusted based on the visual inspection. If no visual trace of bonding wires then we used ten mins. if we see bonding wires after the rens, then we reduce the exposure time to 5 min. Then 2.5 min intervals until we remove all the package covering the die.

The Rensing process starts with using acetone which neutralizes the acid. Then it is followed by a none ionized water rens. We could not find any ionized water, so we used purified instead. A careful dab with paper to remove any access water will do. Rense using isopropanol is to remove any dust of particulates on the surface of a die. The last step in the process is dehumidification nitrogen or any other means. We could not find a suitable method available at the moment.

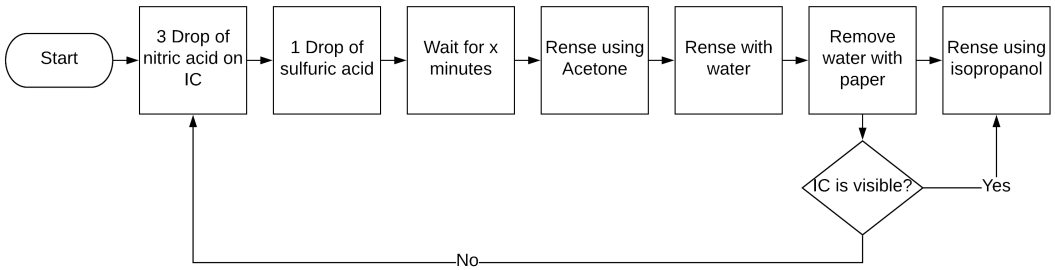


Figure 3.6: Measurements that give to I-LAB for CNC of device.

Table 3.2: equipment's used for wet etch.

Equipment	Purpose
2xBeaker	Pippette holding and rensing with aceton.
2xPipette with Bulb	Used to deliver acid on the IC.
2xPipette Serological	Move acid from larger flask to Erlenmeyer Flask.
2xErlenmeyer Flask	Hold sulfuric and nitric acid.
Petri dish	used to hold IC.
3xLab wash bottles	Used to hold Acetone,water and Isopropanol
Fuming nitric acid	Use to remove packaging.
Sulfuric acid	Used to control the concentration of nitric acid.
Acetone	Neutralise sulfuric and Fuming nitric acid.
Water	Rensing.
Isopropanol	Final rensing to remove any particles.

3.4 Results

The decapping is to a certain degree is possible to achieve using CNC and chemical means. The results show it is possible, but the repeatability is low. The bonding wires made of gold seemed more resistance to nitric acid attacks. The copper one we much more sensitive to contract with nitric acid. The difference packaging properties between AD7768 and TMS570 results in adjustements to the CNC and etching process. The AD7768 package was much thinner than that of TMS570. From the CNC it was clear that the device's package was softer as it was easier to make grooves on the package by merely sliding the drill on the package. The most case of damage to IC wasn't by the acid attack on wire bonding. It was instead that CNC machine did a significant amount of damage. The operator manages to damage either the wire bonding or the die by itself Figure 3.11. The holder of the packages as seen in Figure 3.4 caused much stress on pins after each operation the user had to straighten the pins again.

Both TMS570 and AD7768 IC that have been drilled closed to die experienced

short wet etching and more considerable area exposure of the die. The IC with least drilled depth where exposed long to acid. This did not damage wire bonding, but it took longer to expose the die. The process had to be repeated multiple times. Each time we wash acid some of it ends up on pins which made very corroded and very brittle. An extreme example can be seen in Figure 3.10 all the pins became so brittle that they fall off during a the rens process.

Table 3.3: Damaged ICs, only one TMS570 was not damaged in the process

IC	CNC	ACID	Total
AD7768	5	1	6
TMS570	5	1	7

3.4.1 TMS570 Decapping

Decapping of TMS570 Figure 3.7 is the result obtained from CNC as close as possible to the die. The dies show some scratching on it surface caused by the drill bit. The Second that was noticed under the microscope is that the copper bonding suffers from the thinning effects. The last mentioned causes the structural properties of the wire bonding to change which lead to bending or breaks Figure 3.8. The Thinning will mean as well that the wire bonding changes the electrical properties of the IC. The last IC in Figure 3.9 showed that is more promising then the rest. Here only the dies that have been exposed this keeps bonding wires protected from corrosion. However, When this IC was mounted on PCB to test its functionality, a CLK pin broke off. Those IC were handled with care during the mounting processes. It shows how crucial it is to minimize the exposure to nitric acid.

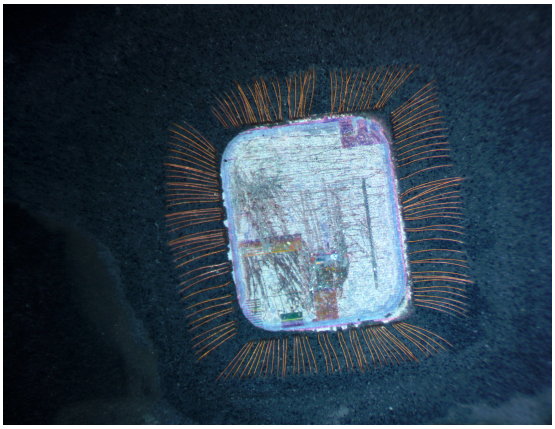


Figure 3.7: TMS570 decapped exposing wire Bonding.

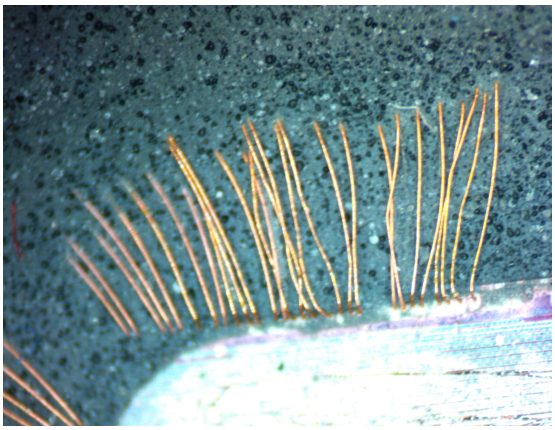


Figure 3.8: TMS570 decapped with exposing of bonding wire close up.

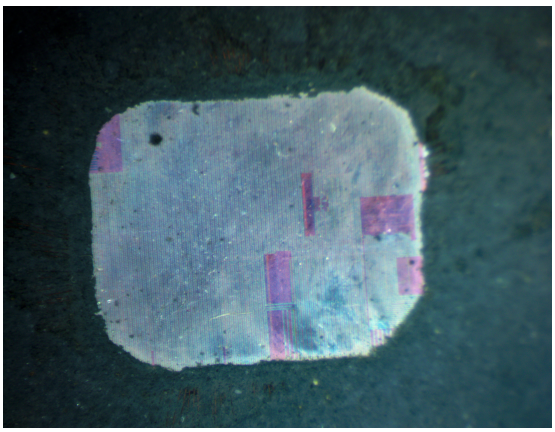


Figure 3.9: TMS570 decapped without exposing bonding wires.

3.4.2 AD7768 decapping

The gold wire bonding withstands nitric acid effects. Not thinning effects observed under the microscope Figure 3.10. The pins where far less resistant to acid attacks as in the previously mentioned figure. The lengthen exposure caused the pins to fall off during the rens process. Most of the damages cause the IC is due the fact caused by the drill. Looking at the Figure 3.11 it shows damaged caused by the drill bit on the die.

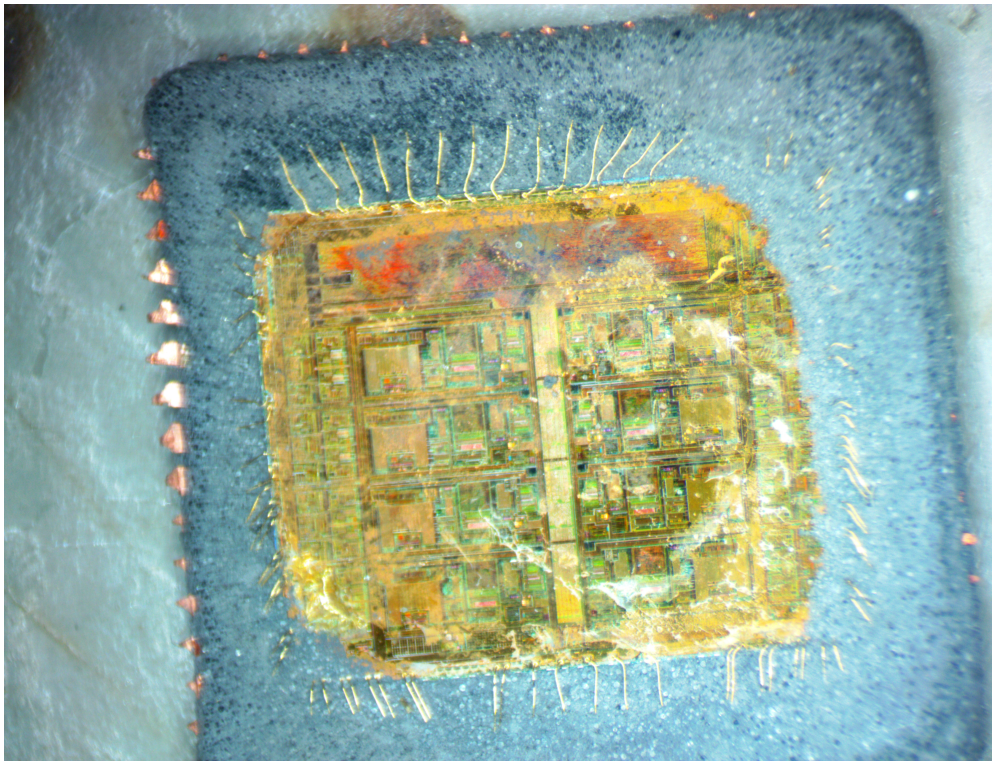


Figure 3.10: AD7768 ADC decapped exposing wire bonding.

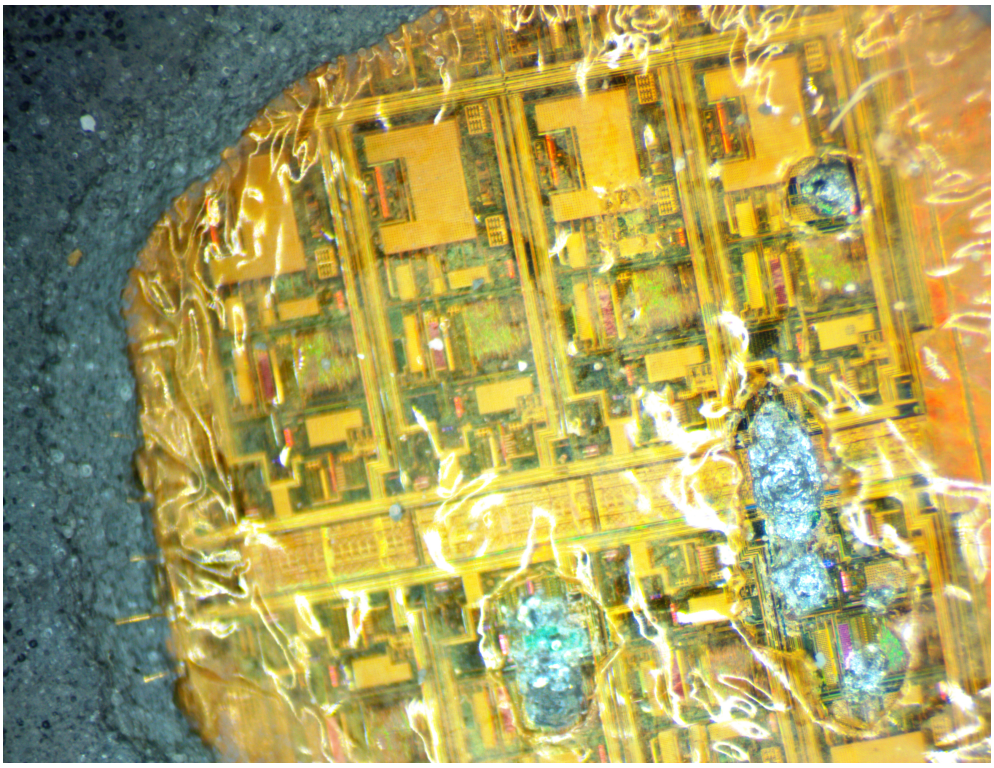


Figure 3.11: AD7768 ADC CNC damage.

3.5 Discussion

We had only one successful decapping. From a visual standpoint, this had good results the pins were intact and slight corrosion from the washing process. The devices were mounted on board to test whether the functionality is still intact of the device. During the soldering one, pins broke off clean. It is clear that acid and mechanical stress did weaken the mechanical properties of this pin.

The results from the decapping process were promising but were restricted by the control ability of variables during the experiment. The tools used were mostly basic to do the process of decapping. The main reason for damages to the die and bonding wires were because of the mechanical removal of plastic Table 3.3. The accuracy proved by CNC has proven to be low in precision and repeatability. Even if the same dimensions were given to the operator, some human errors did occur along the way. The main reasons are the method used to measure the center of the devices. It was clear that putting the drill accurately on the center was not a simple task. This caused offset from the center of the package to damage the bonding wires. For the successful removal, small errors result in the

significant different amount of plastic that encloses the die. This changed the required etching time for each experiment. This cause pins and bonding to be exposed longer to acid.

One of the parameters that were not taken into consideration during the experiment was the surface area of the package. The larger the area, the more consideration had to be taken to determine a better exposure time for the acid and the mixture. The behavior acid on the packaging is hard to predict. The removal tended to be not even. In general acid interaction with material is not isometric. During the experiment, there was always some different amount of material converging the die.

The only way we reduced the humidity is by leaving the IC with bags of moisture absorb of silica gel. This bags normally shipped with electronics to ensure a low moisture levels in packing. This was more as a precaution and a last resource. We have no idea if this method has worked or not. Nitrogen gas drying was not possible to get access to. This would have been a more favorite approach[35].

3.5.1 Future Work

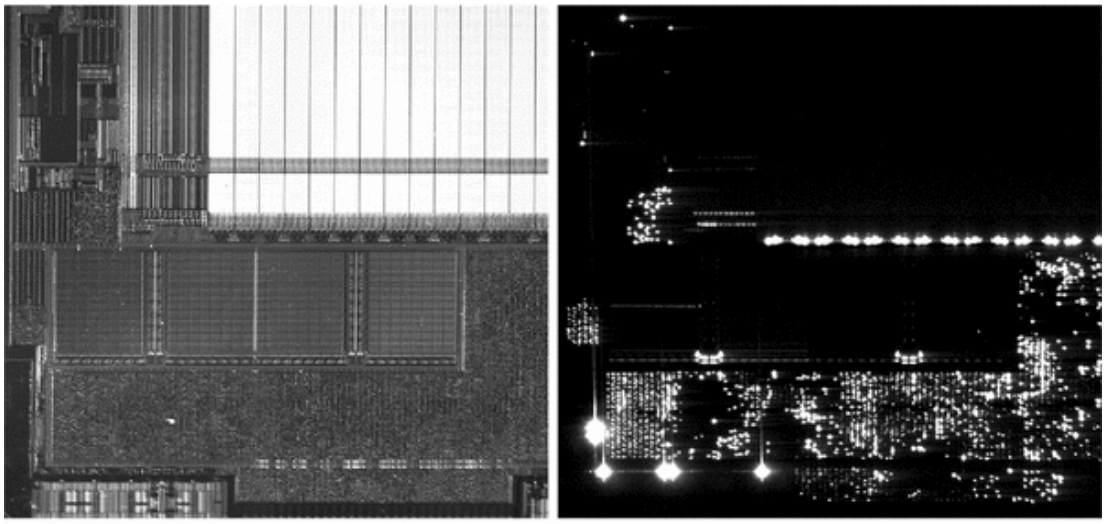
In the article by Mathew advises removing as much package as possible before starting wet etch process [35]. He advice using laser ablation to have a higher precision, and repeatability. This reduces the total time needed for a wet etch to some couple of seconds. The methods of laser removals would provent to be accurate but that boils down to what kind of capability the devices used to create a cavity. An industry laser machines will do much better just at accuracy and reputability. While a hobbits devices would have a low intensity suitable for engraving into matterials. If one can you this capability to engrave a cavirty slowly in a controble manner it may result in event get as expose the die.

3.5.2 Regions Mapping of a Die

Decapping aims to expose the die for physical fault injection. Failure analysis using heavy ions or laser helps to understand the robustness of the COTS. This helps expose the limitations of the recovery mechanisms implemented by the producer and helps develop a mitigation strategy. Having the ability of an in house fault injection will help develop and test new methods of bitflip mitigation before testing in a radiation facility. Laser fault injection of COTS is challenging. The user has to map different areas using to determine their functionality before start with injection testing.

One of the methods that have been widely used by the fault analysis community is photonic emissions analysis [38, 9]. According to Schlosser and Dmitry

[38, 9] when MOSFET transistors changes state a photon is emitted. This effect is called hot carrier luminescence and was discovered in the 1950 [38]. An electric field accelerates charge carriers traveling between the drain and source. In this process, some of the electrons gain energy. The energy is released as a form of a photon with a frequency of 1400 nm. The intensity of the photon is directly correlated to the operational power supply. A decrease in features size lowers the intensity of the emitted photon [38] . To detect this effect one needs a microscope with a Near InfraRed(NIR) imaging sensor. When Writing an alternating pattern to the SRAM will help switch on and off transistors, and the results will be as seen in figureFigure 3.12.



(a) Reflective type image of ATMega SRAM **(b)** Emission image of ATMega SRAM

Figure 3.12: Photonic emissions analysis of ATMega SRAM.©Springer April 2013[9] . Reprinted, with permission.

3.6 Conclusion

The results from the decapping process were promising but were restricted by the control ability of variables during the experiment. The tools used were mostly basic to do the process of decapping. This method requires sacrificing a good amount of devices before getting function device. It is necessary to expose the die to be able to do any form for heavy ions or laser testing. In the future, one should explore the use of a computer numerical control (CNC) laser to do the decapping process.

4 | m-NLP System Firmware

This chapter presents the development of firmware for the m-NLP system. This chapter begins with an introduction to the functionality of the system, the programmable components, and the process of developing firmware for radiation testing.

4.1 Motivation and Goal

The main controller of the m-NLP system is a microcontroller from Texas Instruments. The TMS570LS12x series includes important safety mechanisms that make this series an attractive candidate for COTS in LEO. Initially, this microcontroller family is built with enhanced safety features for transport applications. The safety features are implemented to deal with systematic and random failures. In our case, radiation is the source of random failures. The firmware is developed with the goal that the system will be tested in a radiation ground facility. This will help us determined the sensitivity of the devices. The purpose of the firmware is data collection so we can determine the cross-section of Hercules SRAM and Registers, Ad7768 registers, and LTC3778 registers.

4.2 Background Information

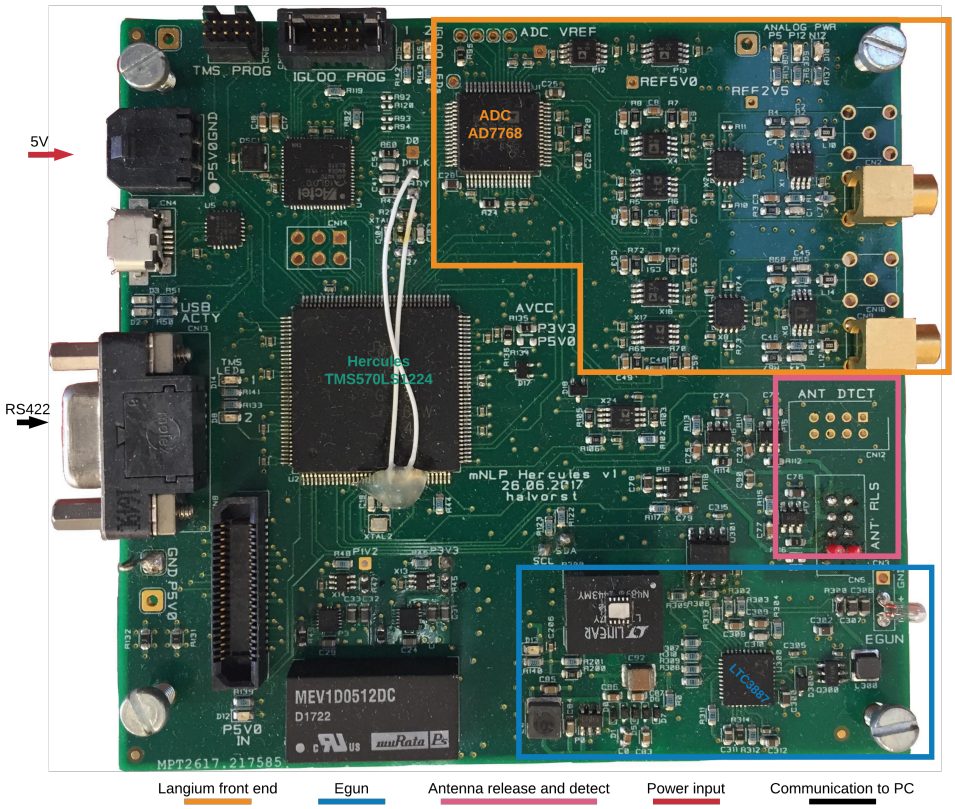


Figure 4.1: The next generation m-NLP system hardware, designed by ELAB.

The m-NLP system is used to collect data on plasma density in the ionosphere using four Langmuir probes. An Egun is used to control satellite platform potential. Tore Andre Bekkeng developed this method, and the E-lab group developed electronics. The board seen in Figure 4.1 is the first generation m-NLP system with microcontroller as the main control unit. This board was the first hardware design iteration. The board had to be verified by functional testing, along with the firmware development for radiation testing. The Board contains main control unit TMS570 microcontroller, Bias Digital-to-Analog Converter (DAC), AD7768 ADC and LTC3887 DC/DC digital power controller. We approach AD7768 and LTC3887 from the functionality point of view. The functionality is based on the technical information got from the E-lab. The theoretical background behind the operation of Langmuir and E-gun are beyond the scope of this thesis. You can read more about miniature Langmuir and E-gun in the Ph.D. thesis of Tore Andre Bekkeng.

Hercules is the main controller tasked primarily with the task of data collection

from external ADC. Also, The controller has housekeeping tasks with external modules for either current monitoring, power lanes management, and device configurationFigure 4.2. The data from the TMS570 will be communicated back to the central On Board Computer (OBC) via RS-422 communication.

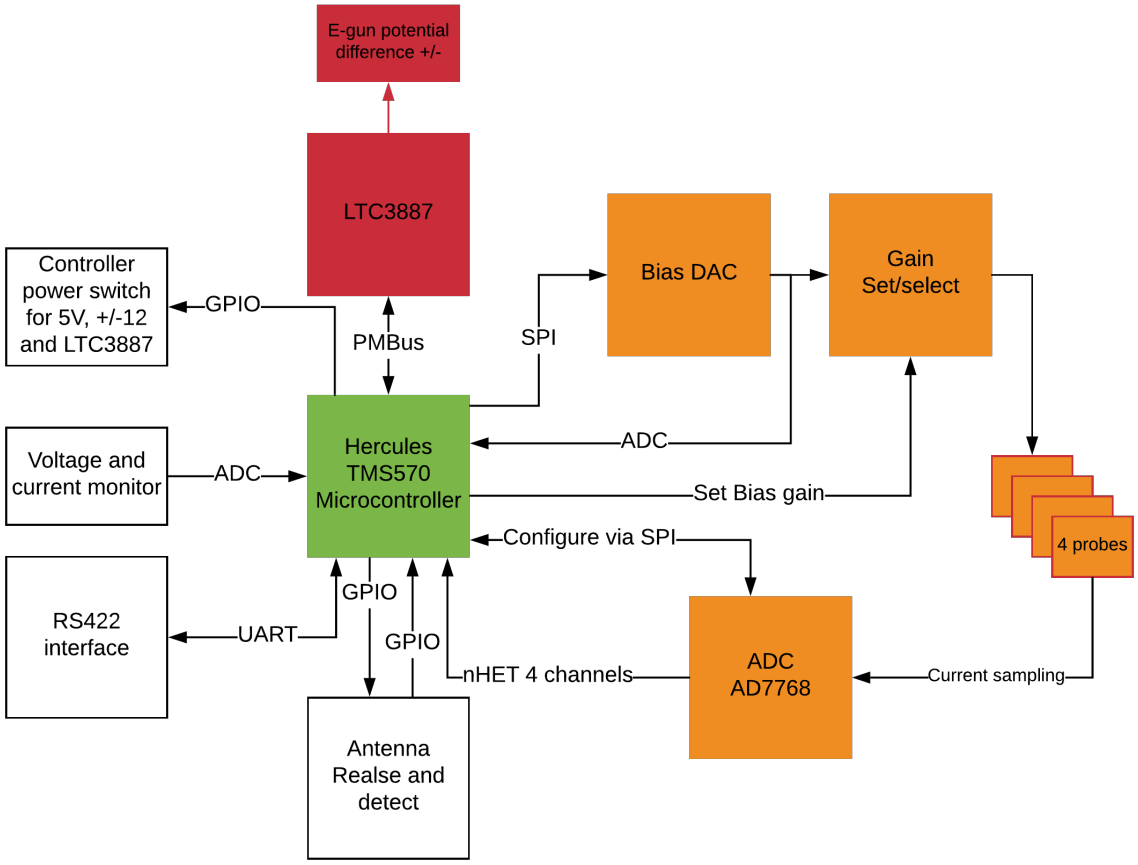


Figure 4.2: Overview of m-NLP communication

4.2.1 ADC-AD7768

AD7768 is an external ADC used to sampling current at a fixed point from four probes Figure 4.3. Hercules programmes this device via SPI. Hercules provides a CLK signal, and the indication start/stop data conversion. In return, AD7768 communicates back via nHET interface. nHET is an interface with own event-driven co-processor which can be programmed to handle custom data transmissions[10]. When ready signal is received from the AD7768, nHET starts

shifting data simultaneously from channel 1 to 4 into an internal buffer[39]. This data awaits to be transfer to SRAM by nHET local Direct Memory Access (DMA) called HTU[10].

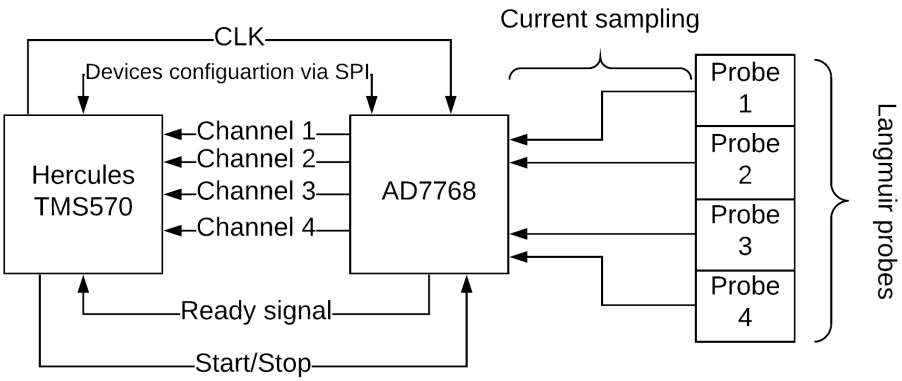


Figure 4.3: AD7768 board configuration and communication paths

4.2.2 DC/DC digital power controller-LTC3887

LTC3887 is a programmable DC/DC digital power controller. This device is command based system is expected to be configured each time at the start of its operation by the microcontroller via Inter-Integrated Circuit (I2C)(PMBus). This devices contains Registers, SRAM and ECC[40]. We did not explore the mitigation available since they were deemed as a secondary task. However, the presence of complicated logical in this device compelled us to include the device as part of the radiation test. The role of the device is the ability to translate a command into voltage difference between the positive and negative terminals of the Egun. The user can change the potential difference from 0 to 5V. The 5V is not the absolute upper limit. It is set to prevent any malfunction to the LTC3887. The devices can provide the user with current, Voltage output between the terminals and Device internal temperature. One can use this temperature sensor to determined the ambient temperature. To do so, one needs to calibrate the device. We will not be doing that as its out the scope of this thesis.

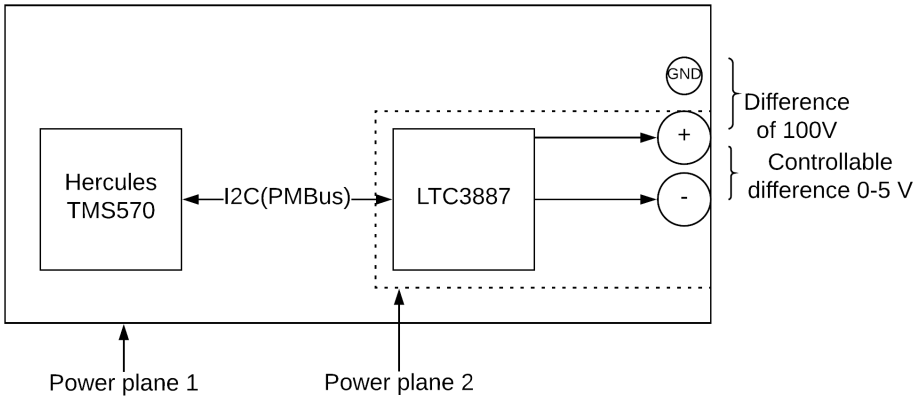


Figure 4.4: LTC3887 board configuration and communication paths

4.2.3 Bias DAC

This module is controlled by Serial Peripheral Interface (SPI) communication. The user can configure each channel individually. This flexibility gives the user the ability to bias probes with different voltage. To verify the bias voltage output Hercules internal ADC is used to read back values on those lines. The interaction with the device is via SPI. This a write-only operations communication[41]. The read-back of configuration is not possible, so the only way to determine that the command executed correctly is by internal ADC read-backFigure 4.5.

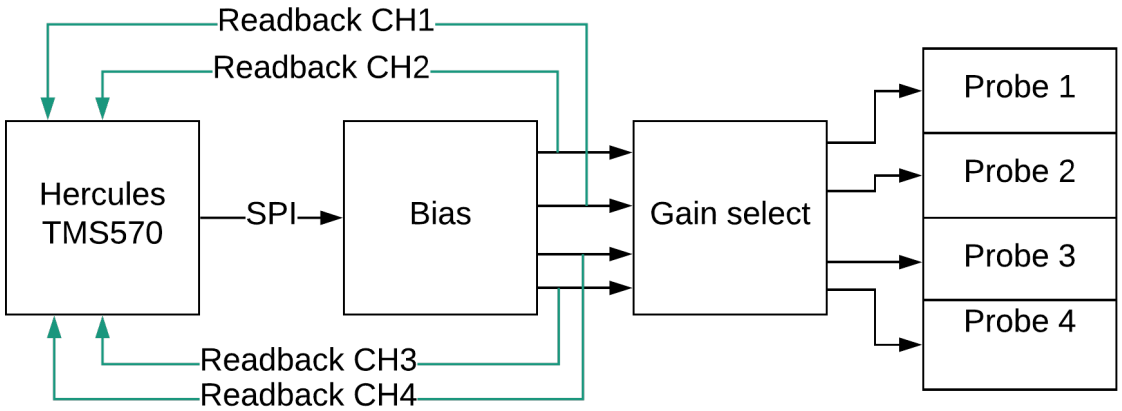


Figure 4.5: Bias DAC board configuration and communication.

4.2.4 Power Rails Monitoring and Control

The TMS570 controller is tasked to monitor current and voltage on difference power lanes as seen in Figure 4.6. The blue dots on the figure represent the switches those are not turn on by default. The user has to use GPIO pins to switch on the current supply. The green lines show the lines that are has a monitor using internal ADC.

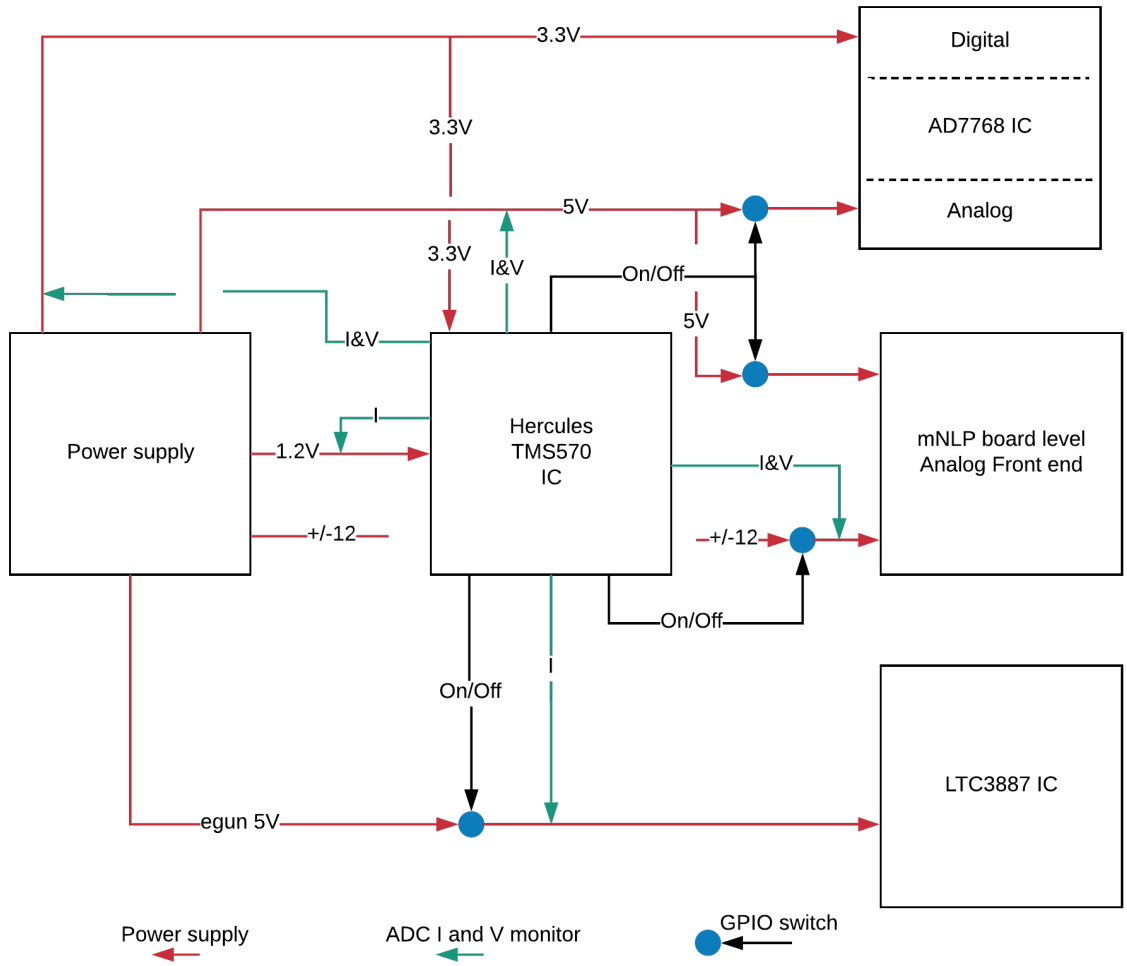


Figure 4.6: Overview of m-NLP system power lines. Current and voltage monitor of different supply lines

4.2.5 Error Signaling Module (ESM)

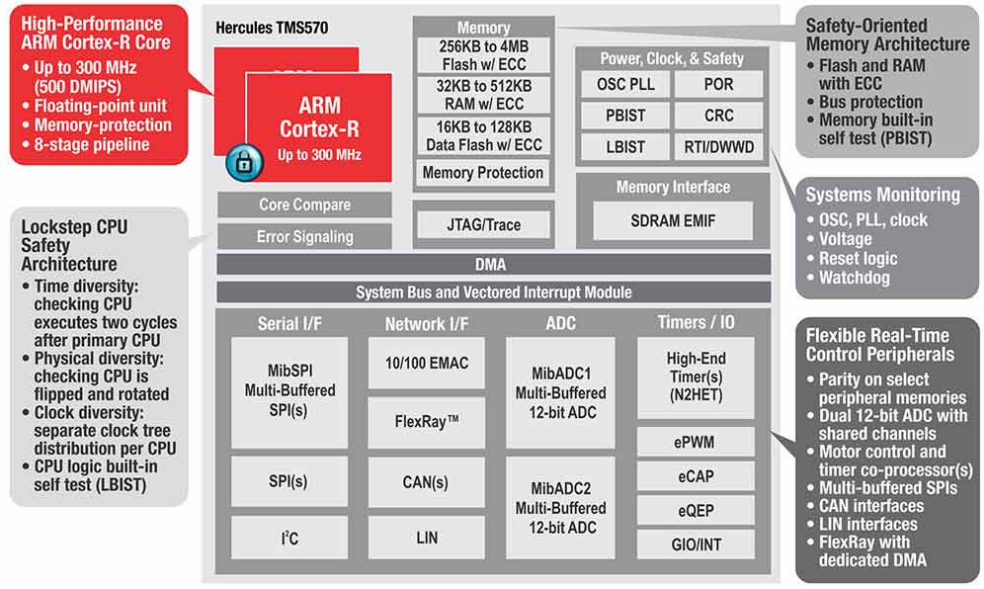


Figure 4.7: Overview of TMS570 safety features [10]

The TMS570 series have safety functionalities embedded into this microcontroller. This functionalities are divided into programmable and none programmable features[32]. A none programmable safety features are hardware implemented that cannot be turn off [32]. The programmable features are the most relevant once to us. They can be utilized to increase the reliability of the TMS570. This microcontroller has 64 programmable functionality. We will not be going through all of them. Only a few are highlighted in this thesis.

This microcontroller is filled with safety features and redundancy. Some features are fault/slips clock detection, Dual LockStep CPU, EDAC for SRAM and parity bits between internal busses. All safety functionality report to the one error handler[32, 10, 42]. It is important to keep in mind that the programmable safety function will not report to ESM if they are not activated. The error signal module is a module that manages error conditions in the microcontroller. The errors are classified into three different levels, where each level/group have properties that described the severity [32, 42]. The user has more flexibility in dealing with errors that have low severity[32]. The ESM monitor 128 error channels with 64 channels for Group1,32 channels group2 and 32 channels group3[10]. Depending on the severity of the report to ESM, an action is taken by issuing a flag, nError, and interrupt.

This ESM is handling errors internally or externally. The external errors are

exported via nError pin to an external/board level watchdog circuit[32]. This monitor issues a cold reset via nPORRST. Such a device is the TPS6538x family for automotive use, and this device includes an array of fault detection and reaction freatures[42]. Some of the most notable are Voltage monitor, current limiter and reset when an error signal is issued. The circuit design used at this moment did not include external monitor capability, so it is critical to see how Hercules circuit mange in a radiation environment on it own. The m-NLP control system focuses fundamentally on handling error locally. We do not want to mitigate against radiation effect. We rather detect them using the ESM module. This module helps us monitor nearly all the TMS570 circuit.

ESM Severity group 1

The group 1 is a low severity group that gives the user the flexibility of configuring the issued errors to an interrupt, error output pin or just nothing. This flexibility means that the user can choose, not to deal with the error, issue a soft reset or a routine that resolves the error. This all depends on the configuration of the system. This group has to be configured by the user. When accessing address from SRAM ECC values are checked to determine whether the value in that address is correct. An ESM flag is raised when an error is detected. The user has had to handle this flag.

ESM Severity group 2

The group 2 error is a none maskable high priority error that signals both the nERROR pin and interrupt. The interrupt generation has a predefined behavior, an example CCMR4 - dual-CPU lock-step error can cause the device to reset. This type of reset is a warm reset, that means that the content of ESM status register will be preserved and can be read to determine the cause of the issue. The way this error function when every an error is detected it will behave accordingly look at the table groups and activities for detailed information.

ESM Severity group 3

The group 3 error is the highest severity ranking, that is means when issued it causes a CPU abort and asserts the error pin output. Such an error is a double error detected in an SRAM address. This results in a system to abort the CPU and issues a signal to an external watchdog circuit. If no watchdog time is existing it is important to implement an internal watchdog which will reset the device.

4.2.6 Requirements

When the Firmware was under development, the test facility was not decided. The firmware had to be as flexible as possible. It has to be suited for a mono-energetic facility and mix field CHARM test facility. CHARM facility has some requirements mainly none accessibility to the device physically for a week. This required more functionality to secure that change or fix could be introduced during the week via a boot-loader. The cabling distance between the PC and m-NLP had a distance of about 40 m in length. Operation in such lengths one needs to ensure that the package transfer contains reliable information to eliminate any distorted data packages. We do not know the rate of bitflips that can occur in the system, and how often double bit error would occur. The Universal Asynchronous Receiver-Transmitter (UART) have to run as fast as possible to ensure that all single bit flips are transferred. This has to happen before a none recoverable double error forces the system to reset itself. SEU are none recoverable once the device is restarted. The following requirements for functionality Table 4.2.

Table 4.1: Task that need to be performed by TMS570

Description	Interface
Communicate with PC	UART
Current and voltage monitoring on power lines	Internal ADC
Antenna release and detect	GPIO
Setting Bias DAC	SPI
Bias DAC readback	Internal ADC
Bias gain set/select	GPIO
Control Switches for 5V, +/-12V and EGUN	GPIO
Configure and control egun -LTC3887	PMBus(I2C*)
Configure AD7768 external ADC	Configure via SPI
Received data from AD7768	nHET

Table 4.2: Features that system need to perform for radiation testing

Description	Purpose
Command line interface	Ability to request any package or configuration
Reprogram via bootloader	Upload new firmware without JTAG
Fast way to transmitt via uart	Retrieve as much data
CRC16 with uart	Increase the relability of the transmitted data
SRAM pattern	save a large array of known pattern
Register	Read all readable registers
Board status	Powerline monitor, DAC bias, and LTC current.
AD7768	Write to register and readback
LTC3887	Write to register and readback

Table 4.3: Safety features that have to be a part of the system

Description	Purpose
Enabling SRAM ECC	Autocorrect single bit that occurs in device
Disabling SRAM ECC	Do not correct single bit that occurs in device
Interface respons timeout	External devices have to respond within a time limit
Enable PBIST	Startup test to check for faulty behavor of SRAM
Enable parity bit	Transfere data from SRAM to CPU with parity

4.3 Method

The system was developed using the Texas instrument tools Code Composer Studio (CCS) and Hardware Abstraction Layer COde GENerator (HALCOGEN). The TMS570 is a sophisticated device with the considerable amount of features to increase the reliability of our system, and the development process included the use of HALCOGEN to generate drivers. Generated drivers do not mean that everything handed down to the user. The user has to determine the behavior of every single section of the microcontroller. The development of the process was started by consulting the technical documentation provided for the product, then execute it in HALCOGEN, write user code and test the behavior Figure 4.8. The development process was divided into modules as see in Table 4.1 this is to increase the testability.

HALCOGEN gives the user the possibility to choose between FreeRTOS OS or bare bone. Our choice was to go with a barebone solution as we could control the number of dynamic variables in SRAM. It is because we do not know how sensitive the SRAM is to radiation and how often bit-flips will occur. A bit flip in a control dynamic data variable will cause a SEFI. The FreeRTOS is deemed excellent at the predictable timing response, but its most substantial disadvantage is the dynamic data stored in SRAM.

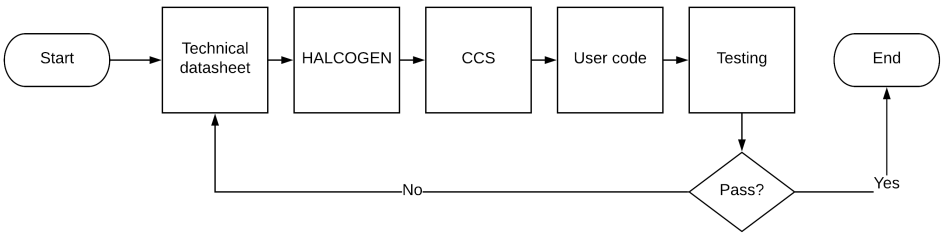


Figure 4.8: Process of firmware developement

4.4 Results

The results of this chapter are rather the firmware and the features that are developed for radiation testing.

4.4.1 Command Line Interface

The development of a command line interface was mainly practical. There are three reasons behind the choosing this instead of a loop based system. The first reason was testing the firmware. Implementing the Command base interface enabled testing commands and arguments to ensure that the firmware performs as expected. This includes testing valid and none valid commands. Negative acknowledgment (NAK) command is issued when the system detects a wrong command. In case of a valid command, the system replays by executing the command and return the package to the PC. The second reason for choosing this method over a based loop system was to eliminate change to the firmware that has been tested. This prevents the user from introducing bugs and increase the reliability of the system. The last reason was the flexibility of the device and the data that can be requested. Instead of hard-coding the packages structure, It is moved to the control software in PC. This allows us to change the packaged requested without changing the firmware.

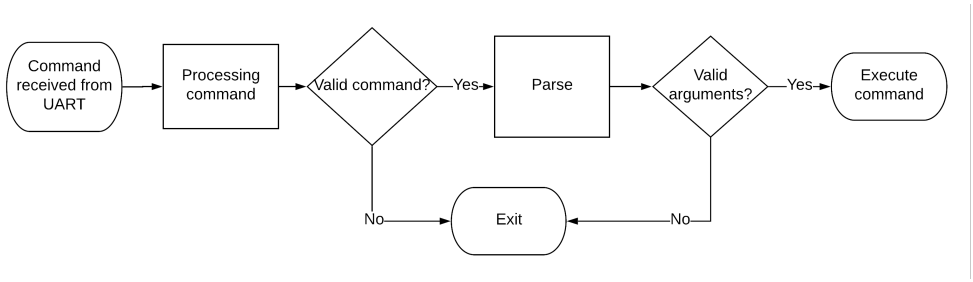


Figure 4.9: Behavior of command line interface

4.4.2 UART Speed and Reliability

UART is the communication protocol used between the PC and m-NLP system. The baud rate is set to be 230.4 kHz which is the maximum possible by TMS570 series. This communication is sent over RS-422. RS-422 is a differential point to point transmission standard. The data have to be transmitted over at least 60 meters. To make sure that the data received is correct we have to use CRC16 and send data as packages. The commands will be issued by the PC then answer is issued by m-NLP. This a standard slave master configuration to control the data flow. Working in a radiation environment and the overlong length it is standard to add parity to verify the correctness of data received. Securing the data transmission is done using CRC16. CRC16 has an advantage over other parity algorithms mainly in the processing needed to encode packages. It is

simple and needs only two bytes to verify a package with a maximum size of 249 bytes. The biggest drawback of CRC16 is its inability to correct a faulty package. The usual method to correct the error is to request the same package again. Due to using the CRC16, It decided to disable UART parity in transmission. The CRC16 decoder checks the validity of the transmission.

4.4.3 SRAM Pattern

0x48 value stores 140000 bytes from start address 0x08001600. The area is about 73% of the total SRAM available on TMS570. This a way to detected bit flips caused by radiation so we can determine the sensitivity of the device. During the radiation test, we will be running rounds with configurations with SRAM. One will be with ECC off where the error is read by directly checking for deviation from a defined the pattern. The second configuration is to run it with ECC on and use the ESM to determine whether the value is correct of fault. The ESM is a flag is issued only when address accessed is has a bitflip. The ESM does not correct errors automatically when detected. The user has to write back the same flag value into ESM, and This starts a hardware routine to write a correct value to the address into the SRAM. So to prevent the probability of double errors occurring a correction method is developed for the memory. This method checks the ESM register after accessing each read address. When a single error is detected the software starts a routine that corrects the data.

4.4.4 Reading Registers

To determine the sensitivity of registers, one needs to read and verify that no bit flips have occurred. The register can be a bit tricky as one can not write a pattern to determine whether they have changed on not. Many registers have reserved bits that are used internally by the digital logic. This can be read but can not be changed. One of the ways of doing this is by recording all available register to the user in a run simulation to the one that will be done during radiation testing. It is to store all possible combinations possible. This has to be done for multiple days to have large enough sample to determine what is normal values for this registers and what is a deviation caused by radiation. More on this topic is explained in Chapter 5.

4.4.5 Communication with LTC3887

Reading and writing to register in this device required implementation of PMBus protocol. This protocol an extension of SMBus protocol with a specific focus on power control. SMBus and PMBus protocol is compatible with I2C timing.

The difference is the order of information sent. The information that devices require to interact with the device are address type, command, length of data read or written. The address type can Global or PAGED. During the development process, we detected that the initial board that was used faulty. These boards are the first generation of, so we had to programme the functionality of the device to ensure that the hardware works as expected. After discovering behavior issues with the board. The same test was performed on another board. We found out the board I was using has issues with Egun circuitry. That one of the primary decision where made to determine that full programmability of the LTC3887 has to be done to verify that hardware is stable. This as well to have full control of the board behavior under normal conditions. After establishing the communication and programming the behavior of the device. We select three register device temperature, current drawn and the potential difference between positive and negative terminals. The temperature sensor is embedded internally in the device. It is a measure of the IC operational temperature.

4.4.6 Communication with AD7768

The configuration of ADC is done using SPI protocol. The communication is to configured the device analog devices technical datasheet called this communication method off frame protocol [39]. For a send bits long values with the most significant as read or write a bit. Then followed by the same value but this time for a read functionality. If the command issued is illegal, this will be detected by the device, and an illegal error will be sent back.

4.4.7 Board Status

This includes a range of different analog values collected one of the important values are current and voltage which is monitored by TMS570 internal ADCFigure 4.6. This values will help users determine whether a specific device causes a latchup. This helps narrow down to the component that is the source of latchup. This category includes monitoring the Bias DAC readback. It is to verify that the values set to DAC does not get altered by radiation. The final value is to monitor the current drawn between the positive and negative terminal of the LTC3887. This is done to determine whether the consequence of a bit flips to the current drawn by the Egun.

4.4.8 Boot-loader

The Bootloader is the first firmware to run after a reboot. This firmware has to be in lowest possible addresses in Flash memory to be considered a boot-

loader. The bootloader was not developed but instead integrated into the code. We had to create a new system link files that have firmware offset of 0x20000 from the start address. This allows the bootloader to have the lowest value to be the first firmware to run at the start of the system. Bootloader enables us to change firmware without the need of needing an external debugger. As well if we are in test areas like CHARM the distance is too large to use the JTAG programmer. Thus the only way to reprogram the device will be via a bootloader. The bootloader runs with ECC on parity and parity bits to increase the reliability of the program. When the device is uploaded to SRAM the bootloader, the system is installed in flash memory.

4.5 Discussion

All of the set requirements were met for the development of the firmware for the m-NLP system. However, this does not mean that improvements are not possible for future iterations. One of the features that deemed secondary is the ability to collect data in software about which part of execution has failed. An example of the timeout that was implemented when communication with AD7768 and LTC3887. When those two devices fail to reply within the assigned time the software continues. The user externally will not receive any information that those devices did not respond. This information helps to understand whether AD7768 and LTC3887 experience some functional interrupts. This is important to understand what parts of the software failed. This one example many other sections of the firmware can report the errors, but missing part is the error handling implementation.

For ECC off no software mitigation techniques were implemented in the firmware. The redundancy by duplication was avoided the idea that we went with that reducing the footprint of dynamic values to an absolute minimum will reduce the chance of bit flip in a control value [8]. When the ECC is on the fact that ECC corrects and detects errors, it is more enough to use this method. The method I used after reading a memory address is ESM check. It is to detect and correct bitflips. This helps to correct data in different address gradually. This reduces the chance of error accumulation in the long run. This method does not cover all the SRAM, but it helps reduce the risk of double errors occurring in often used areas. To be able to cover all the SRAM one needs to implement a scrubbing mechanism of SRAM as a part of housekeeping task[43].

4.5.1 Future Work

The important future works for this device are utilizing the mitigation capability that the TMS570 family provides. The user has to define the how the programmable mitigation functionality can be utilized properly. Here are some suggestions on mitigation functionality that could be useful to implement. All the information in this section is from technical datasheet of the TMS570 family [10, 32, 42].

Clock

Hercules microcontroller is asynchronous systems that require a clock signal to be able to function properly. The microcontroller supply as well the AD7768 with a clock as seen in Figure 4.3. It is important that the user handle errors when detected. The TMS570 family have three safety features related to clock.

FMPLL slip detector module detect slips in phase lock between input and output signals. The response to such an event is programmable and up to the user on how to modify the response to such error. The user can choose between resetting the devices or bypass on slips. Bypassing on slips means a flag is raised to the ESM where the user can determine the behavior in firmware. This enables the user to switch to other clock sources as PLL2 or internal Oscillators.

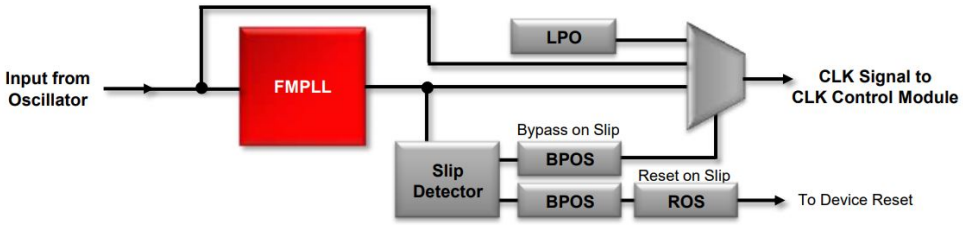


Figure 4.10: FMPLL slip detector module ©Texas Instruments [10] .

DCC are a diagnostic feature that uses two clocks to detect drift between frequencies between two clocks sources. The TMS570 family has two PLLs available for the user. This two PLLs can be used in DCC modules to detect any drift between the systems clock and reference clock. The user can determine the drift tolerance before it is indicated to the ESM module. This particularly important if real-time response required.

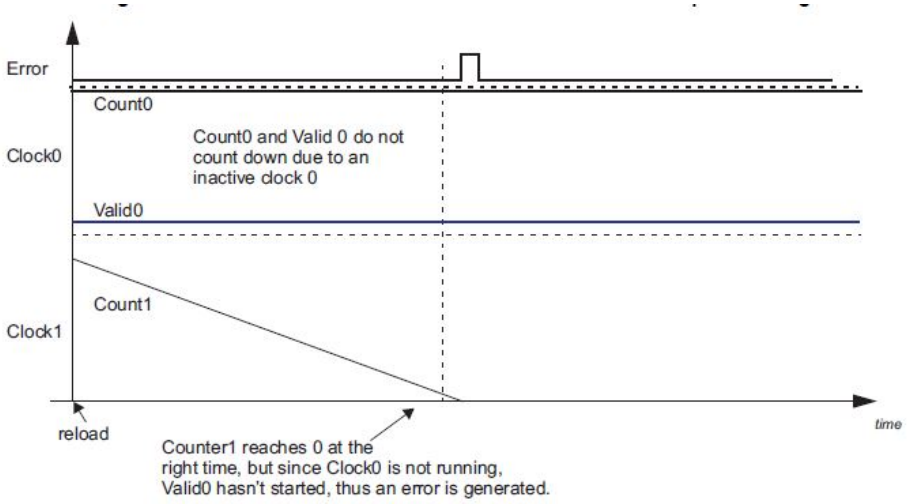


Figure 4.11: Dual clock comparator (DCC) module ©Texas Instruments [10]

This safety feature is used to detect the failure of the primary clock oscillator. The module compares the crystal frequency to two internal oscillators HF LPO and LF LPO. If the crystal exceeds the defined ranges, hardware issues a response by either resetting device or switching to internal clock source LPO. This user determines this.

The LPOCLKDET is a feature that used to detect the failure of the primary clock oscillator. The modules compare the value from crystal frequency to two internal oscillators. Using that two oscillator the user can define the range of the desired frequency that the external crystal should be operating. The range is made up of an upper and lower frequency where the detectors issue a flag to the ESM module. It is essential to note clock frequencies failures that are within the valid range are not detectable with this module.

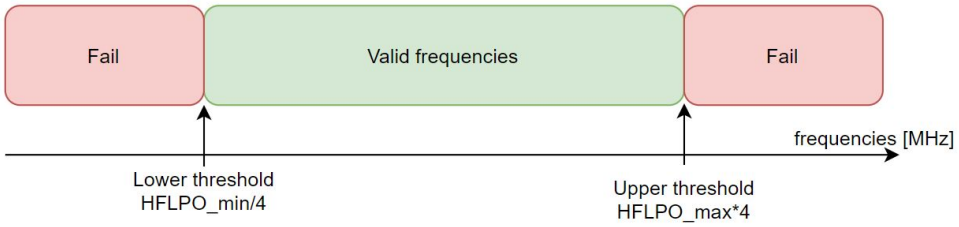


Figure 4.12: Low Power Oscillator Clock Detector (LPOCLKDET) module ©Texas Instruments [10]

Internal watchdog

The Hercules microcontroller family supports an internal watchdog timer. This functionality is useful to reset the devices if a SEFI results in malfunction of the software. The watchdog timer will ensure that the device restarts to resume functionality. The watchdog implemented can be two types of a digital watch (DWD) or digital windowed watchdog (DWWD). The difference lays in the how often the CPU have to check back with the watchdog timer. The DWD is a single threshold which means that only one response is required before the timeout counter expires. After a response, the timeout count is automatically reset. In the DWWD mode, the user has to define an upper and lower timeout threshold. CPU has to between this time window to be counted as a valid. The DWWD allows the user to determine the windows that are acceptable for the CPU to respons.

Figure 13-10. Digital Windowed Watchdog Timing Example

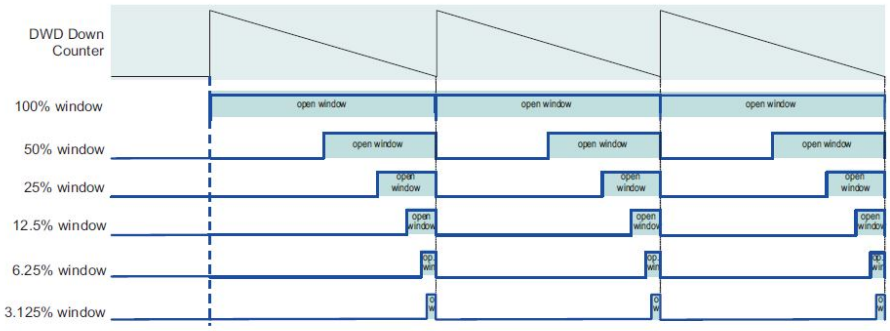


Figure 13-11. Digital Windowed Watchdog Operation Example (25% Window)

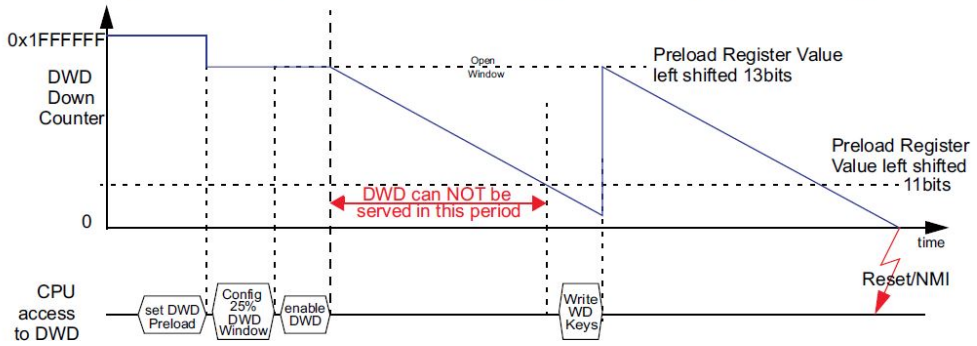


Figure 4.13: Watchdog timer module ©Texas Instruments [10]

4.6 Conclusion

All the primary requirement put for the Circuit were achieved. The firmware is developed to be a command based systems that can send packages with about SRAM pattern, Registers of TMS570, AD7768, and LTC3887. On the top of that, a method of using the Error Signaling Module (ESM) is used as a bitflips monitor. This method allows the detects and corrects of errors to reduces the chance of uncorrectable double errors from occurring.

5 | Radiation Testing of m-NLP System at CHARM Facility

This chapter is about the test that was conducted at CHARM Facility. The chapter lays out the technical requirements that were the need to be able to run such a test. An automatic software to control data from the m-NLP system. The data collected is stored into files for analysis. The software to control power supply to log voltage and currents data into files. Finally, the data process that needs to be done to extract the information that is needed.

5.1 Motivation and Goal

CHARM is the facility that was used for the testing m-NLP system. As mentioned earlier it is a mix field environment with high energetic hadrons. The goal of this chapter is to test m-NLP system in this radiation environment. It is to determine the sensitivity and the rate of failure in such a radiation environment. The devices that will be a part of the test are TMS570, AD7768 ,DAC and LTC3887 devices.

5.2 Background Information

Radiation environment in CHARM is discussed in the theoretical framework chapter. Here we will mostly refer to the technical challenge that rises from testing the system in CHARM facility. The length of the testing is one week. During this period there is no physical access to the m-NLP system. The total transmission over the cables for the power supply and data is 40 meters Figure 5.1 [44]. This data is transferred using duplex RS-422. For this chapter, we need to develop control software for the m-NLP and the power supply for data collection. The m-NLP control software is tasked with two roles one for configuration the system and the other for data collection. The configuration mode is categories into a bootloader and run mode. The bootloader mode enables the user to upload new firmware to the m-NLP system. The run mode is the mode where the user can

change the configurations of the running board like witching powerlines or change the packages that we need to access.

The power supply control software has the role of monitoring and collection data on voltage and current. The monitor capability is to switch offline if the current is higher then a defined limit by the user.

Both control software will be collecting data for a total period of two weeks. This data is collected into a PC. The PC used is a small size client with a 250 GB data storage and the ability to connect to the internet. Teamviewer is used to access PC remotely. It is used to monitor and introduce changes. For the processing files, a cloud base service was used to sync the files from PC at CHARM to a local disk at the University of Oslo.

The facility provides the data relevant to the radiation environment during the testing period. This data includes the time stamp, fluence, and dose at the test position.

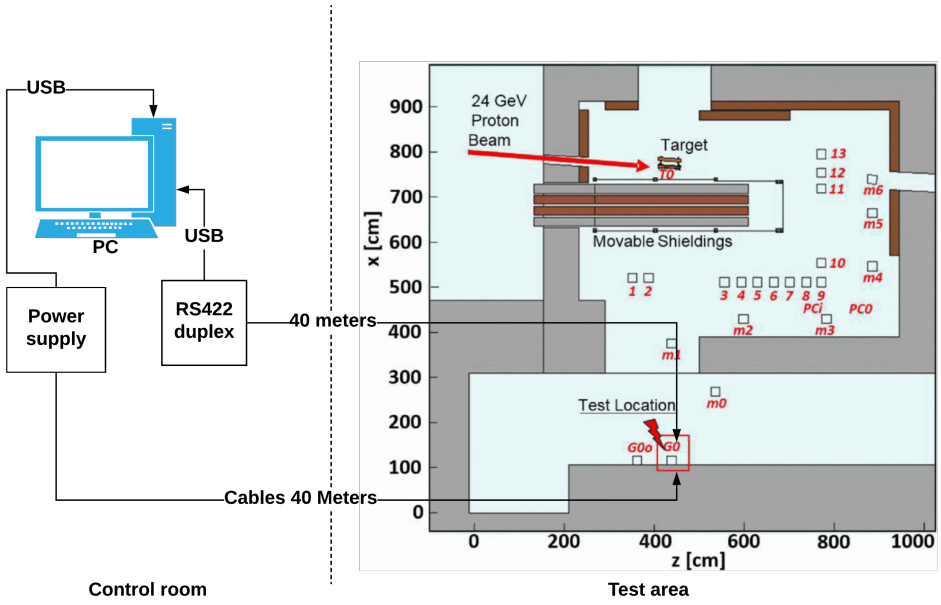


Figure 5.1: Test position at CHARM facility

5.2.1 Requirements

The Software developed for data collection and processing have to be developed in the same language. The software development should be as flexible as reused

by the two systems for power data collection and m-NLP. We set the following requirements for data collection to be Table 5.1.

Table 5.1: General features for power and m-NLP data collection software for PC

Description	Purpose
Command line	Flexibility to reconfigurate the system
Data collection in file system	All data receive has to be store in a set of files
Multiple small files	Store in multiple files with a size of 100 Mb
Fast developement time	features versus developement time
Auto mode	automatic data collection from m-NLP
Stop auto mode	Stop the auto mode in safe manner
Time stamp	Data packages are timestaped by the PC

The requirement for data collection is to determine the sensitivity of the devices listed in Table 5.2. The languages used to collect data will be the same to reduces the development time. The goal of this to determine the radiation sensitivity from the collected data. This is done during the data processing phase. In this phase, we will be looking for bitflips from those we will determine the cross section of each device.

Table 5.2: Goal to determind sensitivity of the follow devices in LEO

Description
TMS570 registers
TMS570 SRAM
DAC
LTC3887 registers
LTC3887 analog
AD7768

5.3 Method

Collection and processing of data are the way of determines the hard and soft errors that occurred in the system during the radiation test. The first one data collection will quickly highlight the most important results. Then data processing will tell us about the meaning of the data we have received.

5.3.1 Data collection

Data collection and control was done by two systems as see in Figure 5.2. To the right we have a programmable power supply and m-NLP system is shown to the right.

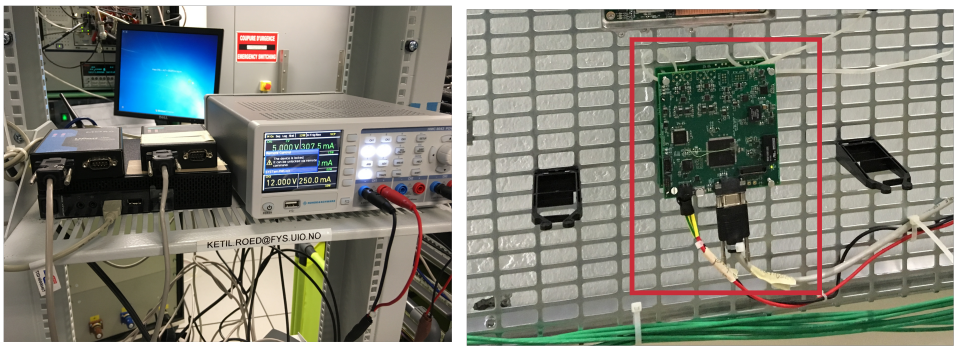


Figure 5.2: To the left is the control room with a rack of PC, Moxa and power supply. To the right is the test area showing G0 position.

Power supply

Control software communicates with power supply via USB using Standard Commands for Programmable Instruments (SCPI)[45, 46]. The power supply has a role to collect data and monitoring of current and voltage to determine the SEL of this device. The current is always monitored during the data collection process. That is to detect a current surge in each channel. When current is greater then the defined threshold the software sends a command to turn off the channel. Protecting the devices from experience a SEL that damages the devices permanently. As well logging the SEL to determine the sensitivity of the devices in this environment. Since we have multiple devices on the same board, we want to extract as much data as possible to determine the SEE for other devices.

The user can configure the upper current threshold from the command line of the software. Its possible as well to turn off individual channels manually. Along with this, a real-time monitoring platform was running to displaying the current per channel. It is to enable the user to monitor the current drawn by the devices manually. The control software was written in python scripting language with an implementation of threading to reduces the latency between detection and response. The response of the power supply was measured to be about 34 ms in average for three channels.

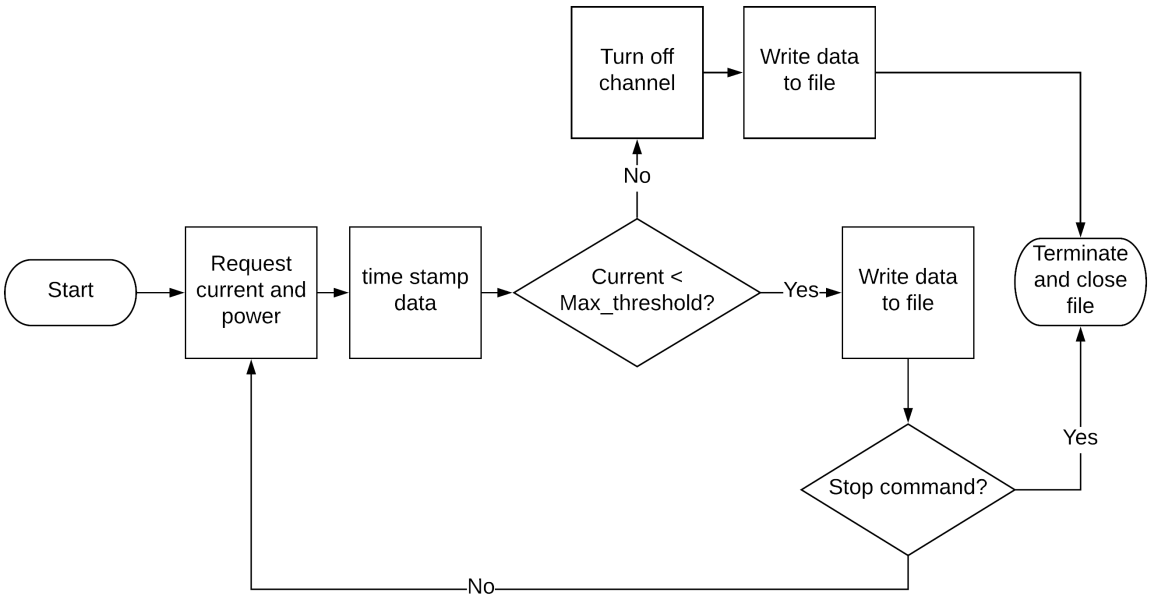


Figure 5.3: Data collection and monitoring using power supply software.

m-NLP system

The data collection by m-NLP control software is a bit different as here the data is packaged based. When the software is in automatic data mode, the pc requested packages in the following order. It is a continues loop that will stop only if the user issues a stop command. One revolution take about 30 seconds getting all the packages. The software does not relay on handshake to request packages but it is done by using a thread that put in sleep mode with an interval of 30 seconds.

- SRAM for TMS570.
- Register data includes TMS570, LTC3887 and AD7768.
- Board status data is analog data.

The SRAM data include information about time, address, values in that address and finally the ESM registers values. In this way, we use the same data package to run with ECC and without ECC. The ESM register is used to monitor all the faults that have been detected by the module. The main reason why this module is read on each SRAM address is to use it as a detection method of bitflip when the ECC is on.

The register information made of time, address and value of that address. The data accessed is all registers available for the user in a none privileged mode in TMS570, LTC3887, and AD7768 devices. The privileged mode is a mode where the user is given access to all register. It is only possible when JTAG is used. The Registers in TMS570 covers nearly all register in the devices, here we are monitoring 3100 32 bits registers. The AD7768 and LTC3887 we are monitoring all the registers.

Board status data is made up of the following data. Current and voltage data helps us narrow down the area where the devices that have experienced SEL. The readback of DAC channels helps determined the sensitivity of the DAC as this is the only way we can detect a bitflip. As for The LTC3887 reading, the analog value of current between the positive and negative terminals will help us understand the consequences a bitflip has on the measured value.

- Current and Voltage from power traces.
- Four channels of DAC.
- LTC3887 measured current drawn by +/- terminals.

5.3.2 Data Processing

The data collected was a considerable amount. With a varying degree of complexity to extract information about the sensitivity of each device. The total amount of data is 47 GB for a two weeks run. This data was rearranged into data frames to simplify data processing. The data frames were divided based on the following data types Figure 5.4.

The power supply and board information data set were the easiest to analyze as they contained analog measurements. The power supply was relatively easy and was ready to be put straight into a data frame for further processing. The Board information data were extracted from m-NLP data then rearranged into data frames. The Challenge was in processing data from SRAM and all other registers.

The software language chosen for this task is python with Jupyter labs as a platform for rapid data analysis. The software is written is not a stand-alone. It is made into modules that can be used an reused in different stages of the data analysis. The behavior of m-NLP data is very dependent on the data of the power supply, devices stop responding and change of firmware. each time one of those events occurs SEU that is not extracted will is lost.

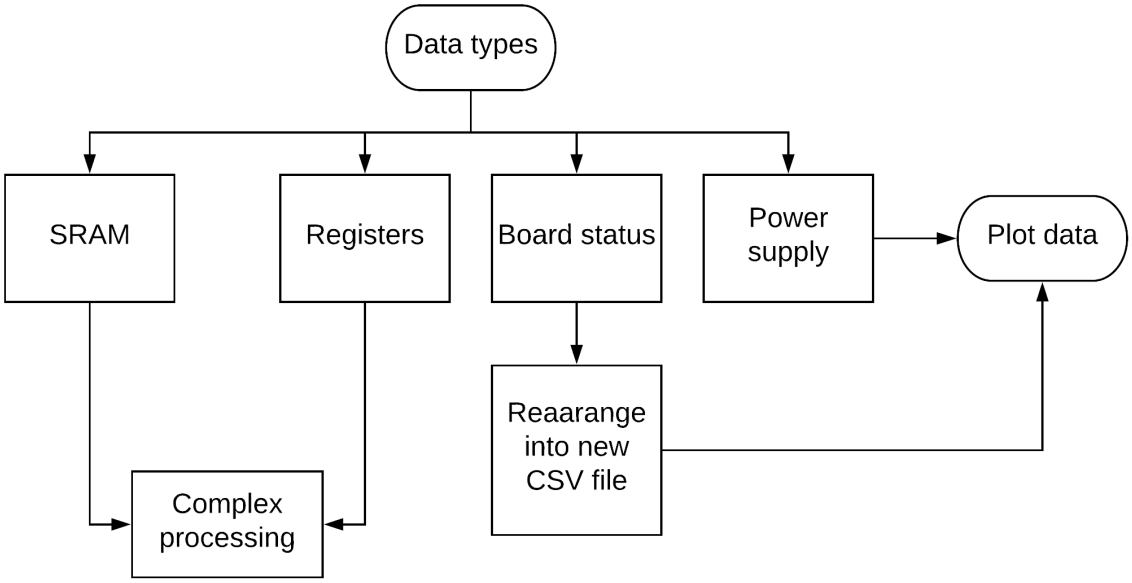


Figure 5.4: Data types that is collected and the process of data analysis

SRAM Data Processing

The amount of data here is overwhelming. The most interesting here is the data that has a deviation from the patter written to the SRAM. We have to extract those deviations to be able to determine the single event upsets that have occurred. The data extract is put into data frames. This gives us the flexibility to rearrange and manipulating data. The descending sort the SRAM data will allow us to extra all the data which different then the pattern into a new file made only of bit-flips detected. The same has to be done to rearrange the data from Ascending this time. This method helped the processing a large amount of data for SRAM in a matter of 10 minutes.

It helps to determine the correctness of the bitflips detected. This for example as [7, 18] said that the should a relation between the increase of the count SEU and the fluence of that the devices have received. Another way to check using bitflip frequency by address. The radiation interaction is random it is expected the chance of recursing of a bitflip in same address should be low. The last one is more of an assumption rather than a factual.

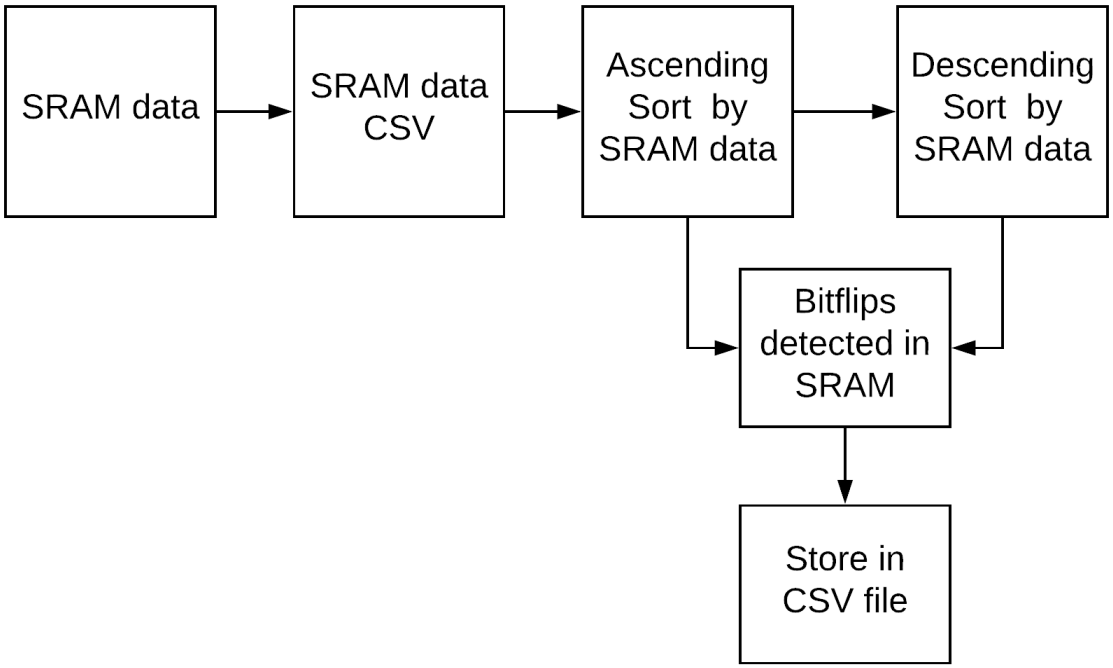


Figure 5.5: Processing method of the SRAM data. The results is file that contains only detected bitflips

Registers Data processing

Determining whether SEU has occurred in registers are not an easy task. In the microcontroller, the size of the registers are 32 bits, and different bits in a single register means different configurations. In the technical datasheet[10], some spots of the register are reserved read-only. These reserved registers are used by the microcontroller logic, as noticed this have a very dynamic behavior. The way we approach this problem is by providing reference data for comparing and detecting deviation. The reference collected data is run in the same way as the ones run at CHARM facility. The two data-frames will be compared to remove all the duplicates. A duplication is determined when address and values stored is recurring multiple times. The duplicates represent the typical values in registers that present in both cases. Same as SRAM here we are interested only in the deviation and values which are not recurring. This method works well for registers that are static or changing seldom. The method is not enough for highly dynamic registers that change values regularly. After the process, we were left with 300 registers. The only way was to manually consult the datasheet to

determine the position and role of that anomalies. If the bitflip in question was apart of the reserved bit, then data will be excluded. This process helped reduce the data to 47 confirmed deviations from the default values. The last step in our verification is to plot bitflips frequency by address.

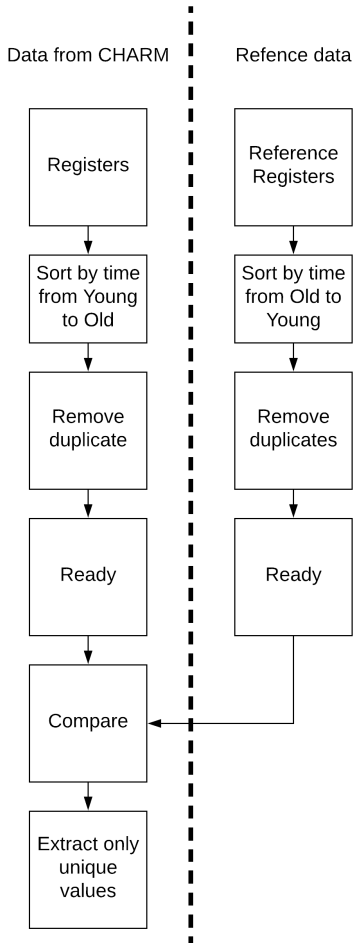


Figure 5.6: Processing method of the Registers data. The results a file that contains only detected few register. Those are compare personally to determine whether it is a bit flip or not

5.4 Results

The results will be categories based on device. We will comment on each category and the significance of the key results. After analysis we found out that all the bit

flips detected on all devices where of SBU. The test was running for two weeks. We achieved all the requirements set. During the test period, we experienced seven functional interrupts. The m-NLP system stopped responding to the commands. Three of them caused the system to reset. To determine the reason of reset we check register that preserves their values even after reset. Those registers are known as shadow registers. The ESM module has such a register. Checking the value of the shadow register, we determined that the cause of the three resets is dual lock CPU.

5.4.1 Current and Voltage measured

The systems were running in two weeks no SEL where detected current spikes to indicate. The Figure 5.7 shows the times when we turn off and on corresponding. The significant data gaps in power data are due to the automatic sync with a cloud solution. This cause some files to be written. The main reason for that a combination of files name management by the python software and the behavior of Dropbox. The event file that was tasked to log events that exceed the defined current threshold was not affected. During the two weeks of testing no SEL where detected. This corresponds with the finding of Jano [43]. It is important to keep in mind that this is true for this LOT number of COTS.

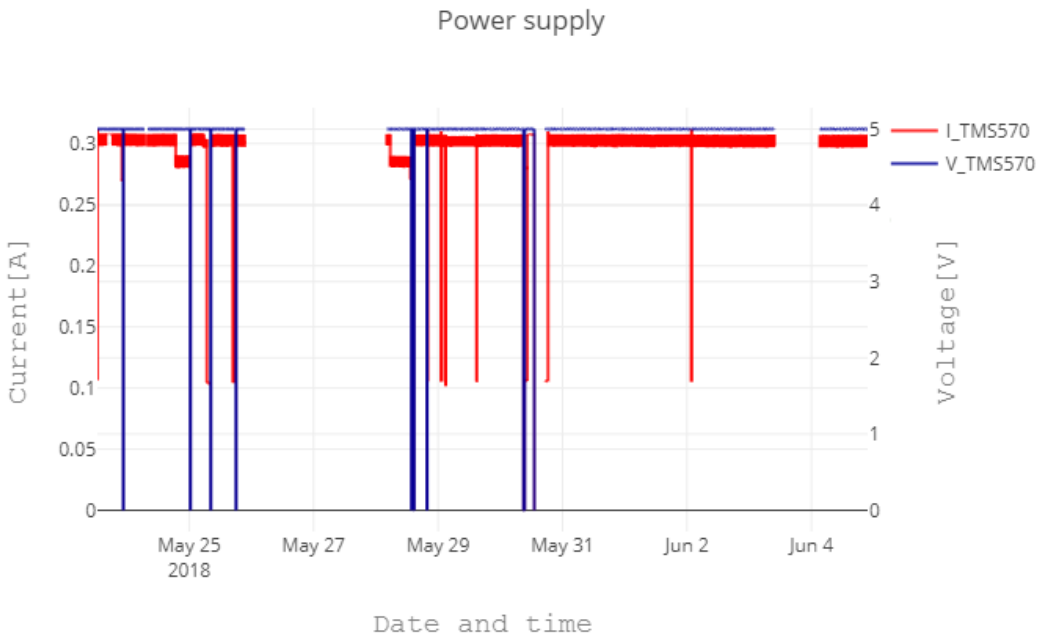


Figure 5.7: Current and voltage measured from power supply fo m-NLP system

5.4.2 Board status

The board status two figures Figure 5.8 and Figure 5.9 show that no effect on the DAC and internal ADC of Hercules. The values got in two weeks test are as expected.

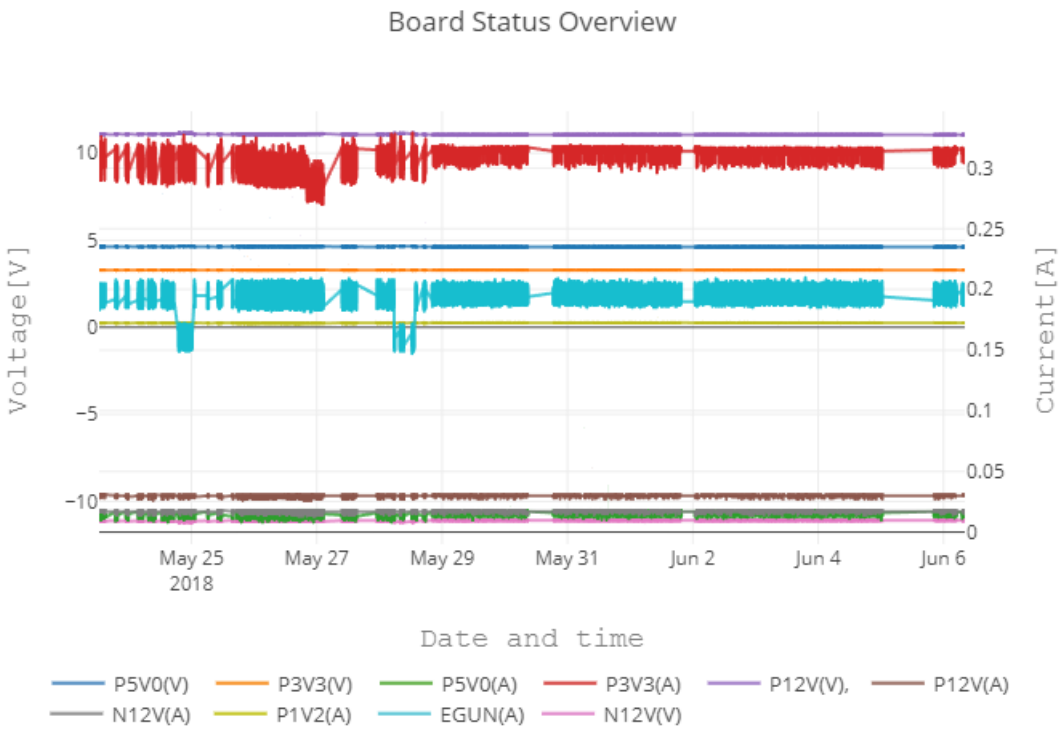


Figure 5.8: Board status of voltage and current for power traces

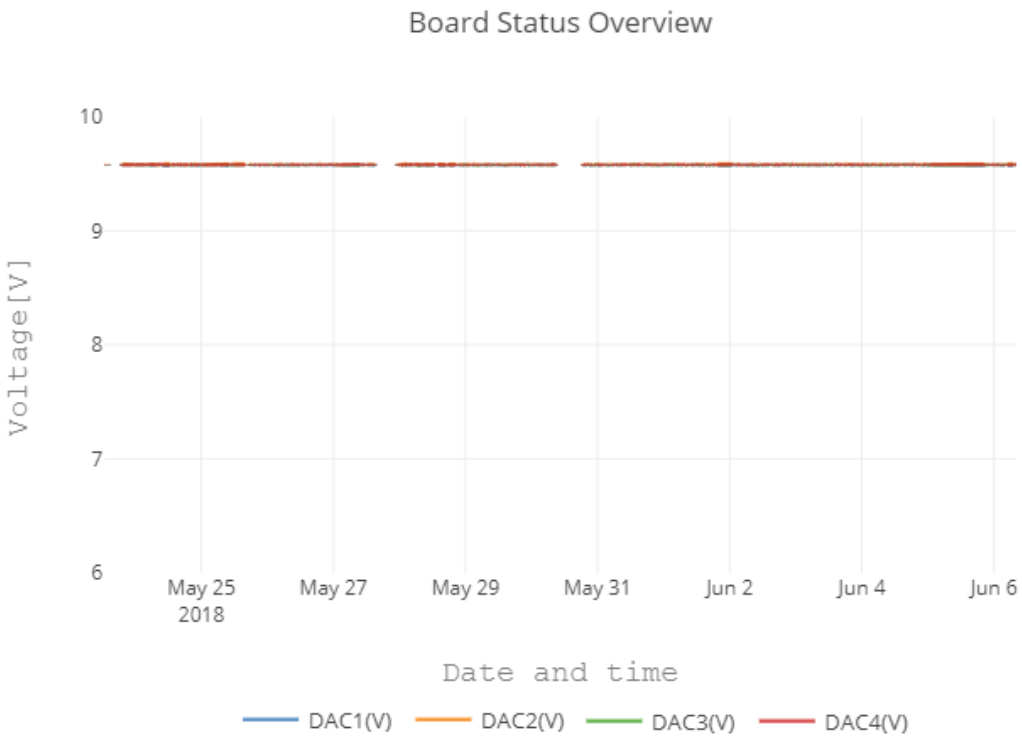


Figure 5.9: Board status DAC voltage read back

5.4.3 SRAM

The SRAM data is made of data collected from ECC on and ECC off modes. One exciting fact when the ECC is on we found out that the ESM module could be used as a built-in radiation detector Figure 5.10. The data collected for SRAM was only from the last week. The second week had less fluence than the first. The reason lays in technical difficulties faced during the testing period. When disabling the ECC in HALCOGEN, it merely means function call to initiate ECC is removed. This works when one firmware is running. When adding a bootloader that initiates ECC the same configurations are passed to running firmware. A work around this is by calling the assembly method to disable ECC.

$$\sigma = \frac{N_{event}}{F \times Bits} = \frac{103}{0.8944 \times 10^9 \times 1.12 \times 10^6} = 1.03 \times 10^{-13} \frac{cm^2}{bit} \pm 9.85\% \quad (5.1)$$

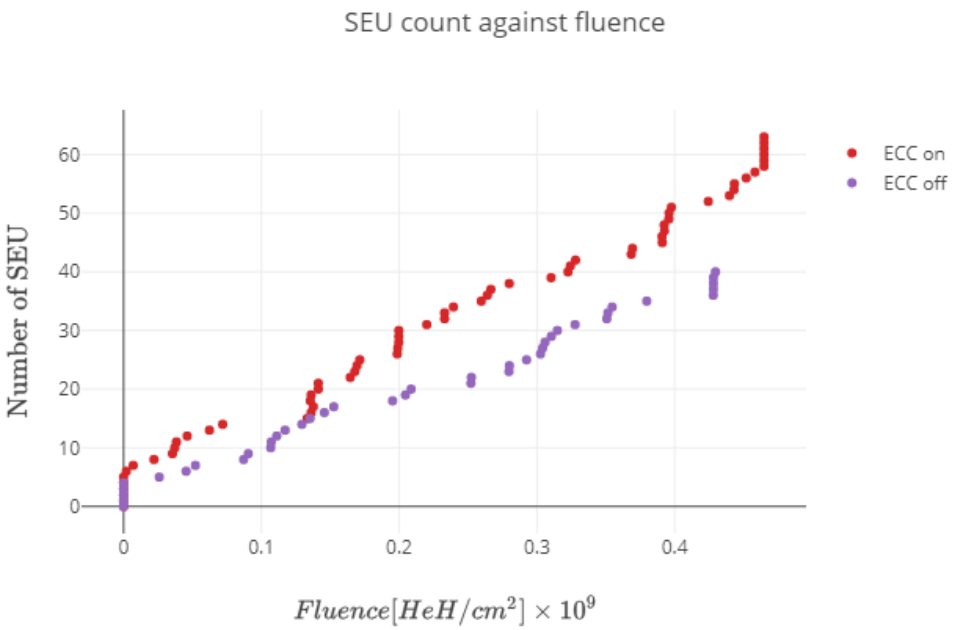


Figure 5.10: SRAM counted single event upset for ECC on and off

ECC ON

The frequency ESM flag raised in Figure 5.11 are all for the SRAM. The module that determines the correctness of the value stored in the register divided 32 bits

values into 2. Even and odd bytes these flags represent that division. We have no comment on why one flag is raised more often than the other.

The Figure 5.12 show the flag detection by address. All addresses are displayed as unique, but only one of them that has three flags raised to ESM that is since we read some SRAM values that are not scrubbed and the ESM flag is not cleared. The flag is cleared ' -on after the first address and last time after the end address of stored pattern. In general, when analyzing data from ESM we see we see that the data is indeed is corrected back to correspondent address. To do so, one has to write back the same flag to the ESM module which triggers a mechanism that writes the correct value back into SRAM address.

$$\sigma = \frac{N_{event}}{F \times Bits} = \frac{63}{0.464 \times 10^9 \times 1.12 \times 10^6} = 1.21 \times 10^{-13} \frac{cm^2}{bit} \pm 12.5\% \quad (5.2)$$

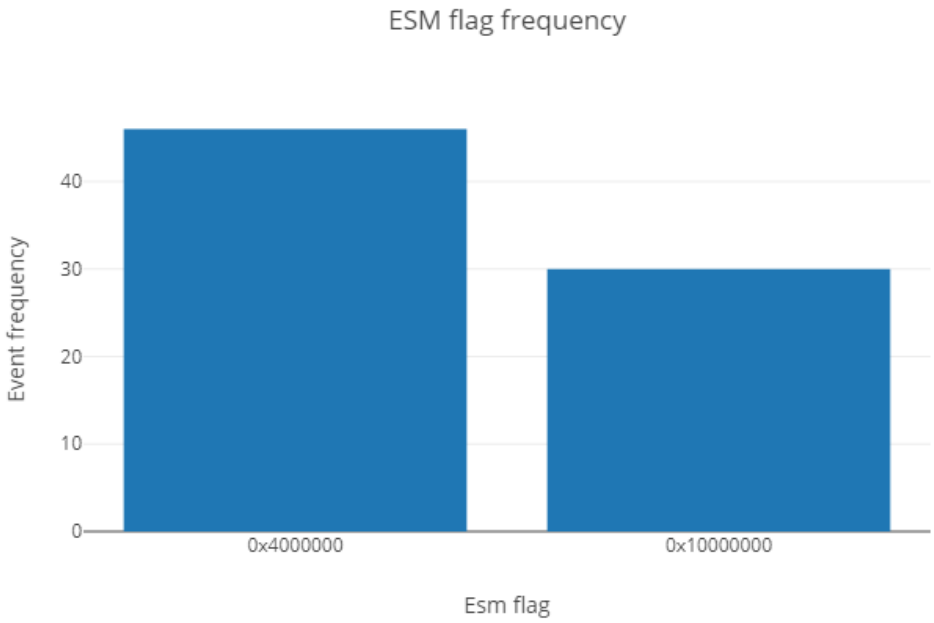


Figure 5.11: TMS570 ECC on Frequency of ESM flag

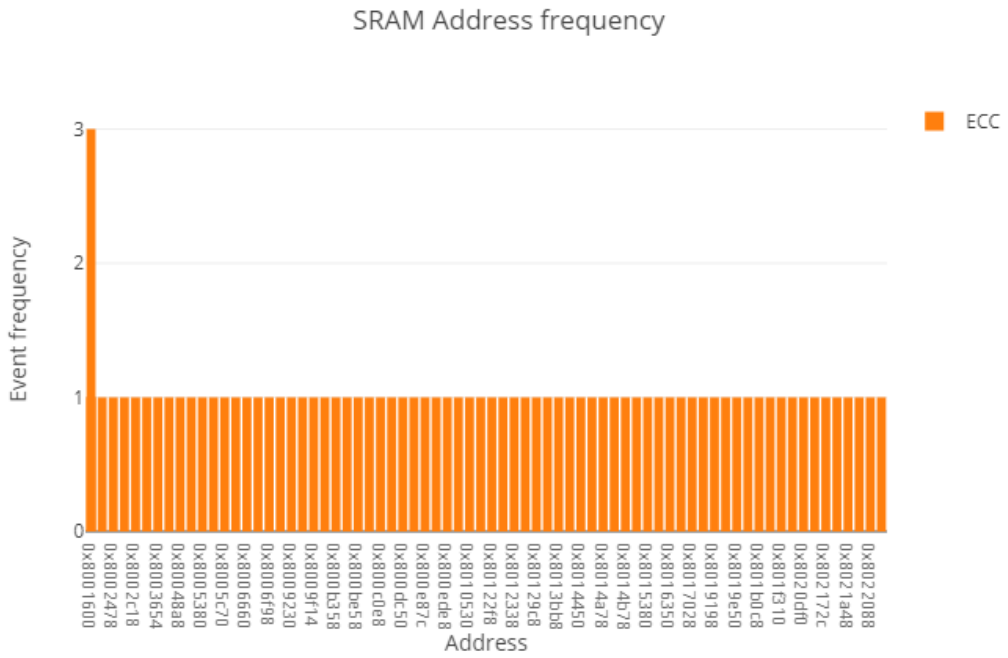


Figure 5.12: SRAM with ECC on frequency of bitflips per address

ECC OFF

The ECC off bit flips appeared by address is unique as expect Figure 5.13. The pattern chosen to write to the SRAM was unfortunate we could have chosen a pattern like checkerboards which will help determine whether SRAM is easier to bitflip to 0 or 1. That depends on the configuration of the pull-up and down transistors in the cell.

$$\sigma = \frac{N_{event}}{F \times Bits} = \frac{40}{0.429 \times 10^9 \times 1.12 \times 10^6} = 8.32 \times 10^{-14} \frac{cm^2}{bit} \pm 15.8\% \quad (5.3)$$

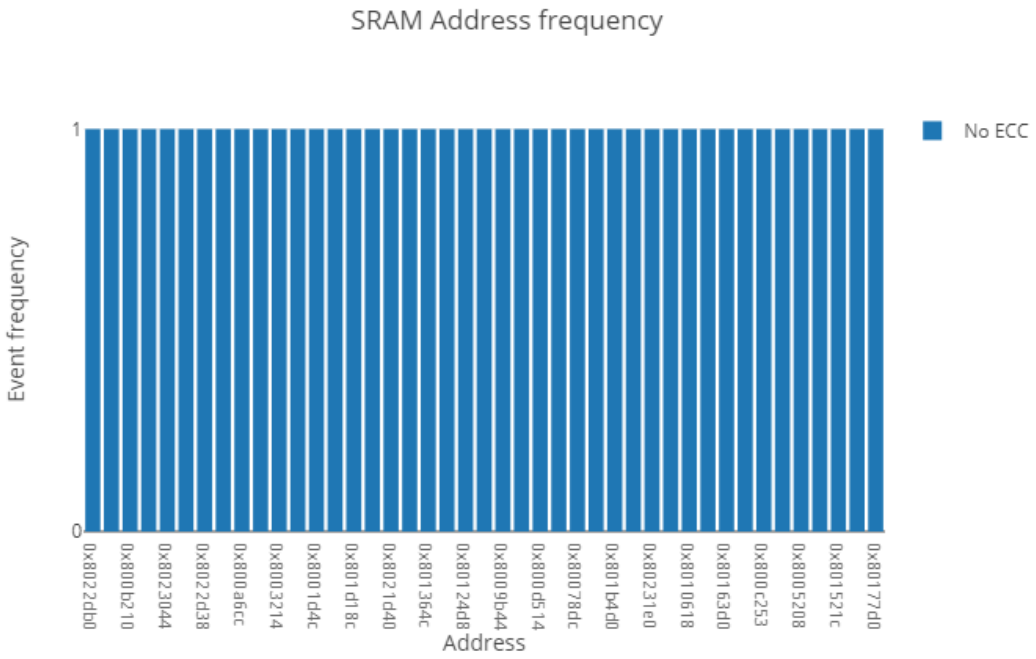


Figure 5.13: SRAM with ECC off frequency of bitflips per address

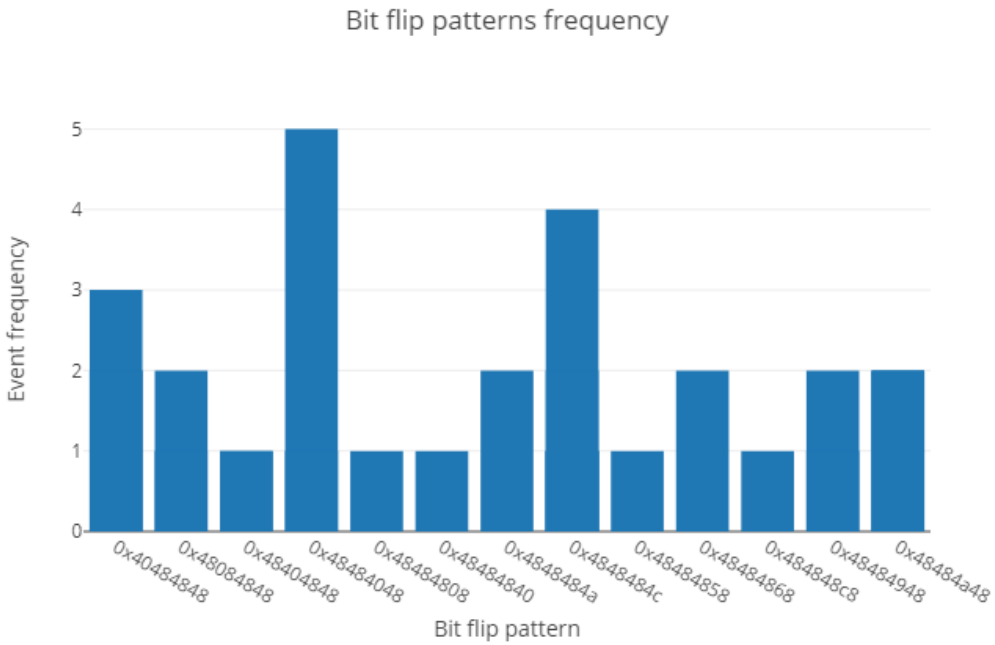


Figure 5.14: SRAM bitflip pattern

5.4.4 TMS570 registers

The SEU upset was reduced to the total shown in Figure 5.15. We have three addresses that appear to have a frequency of two times. A possibility for bitflip to occur in same address is quite low. We choose to disqualify those three values from the cross-section. That reduces the number of SEU to 45.

$$\sigma = \frac{N_{event}}{F \times Bits} = \frac{45}{39.08 \times 10^9 \times 9.92 \times 10^4} = 1.16 \times 10^{-14} \frac{cm^2}{bit} \pm 14.9\% \quad (5.4)$$

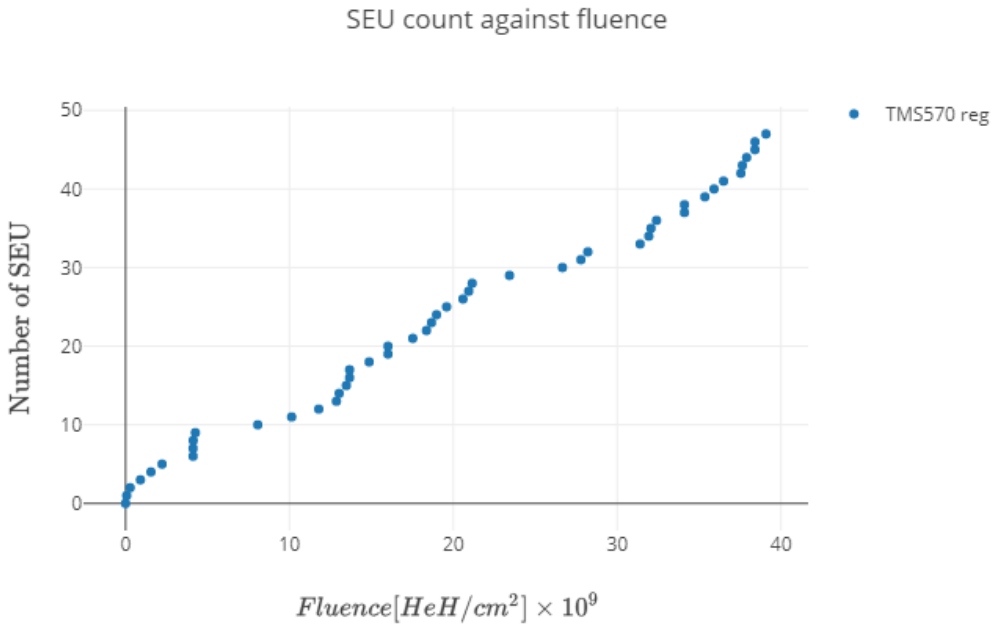


Figure 5.15: TMS570 counted single event upset in registers

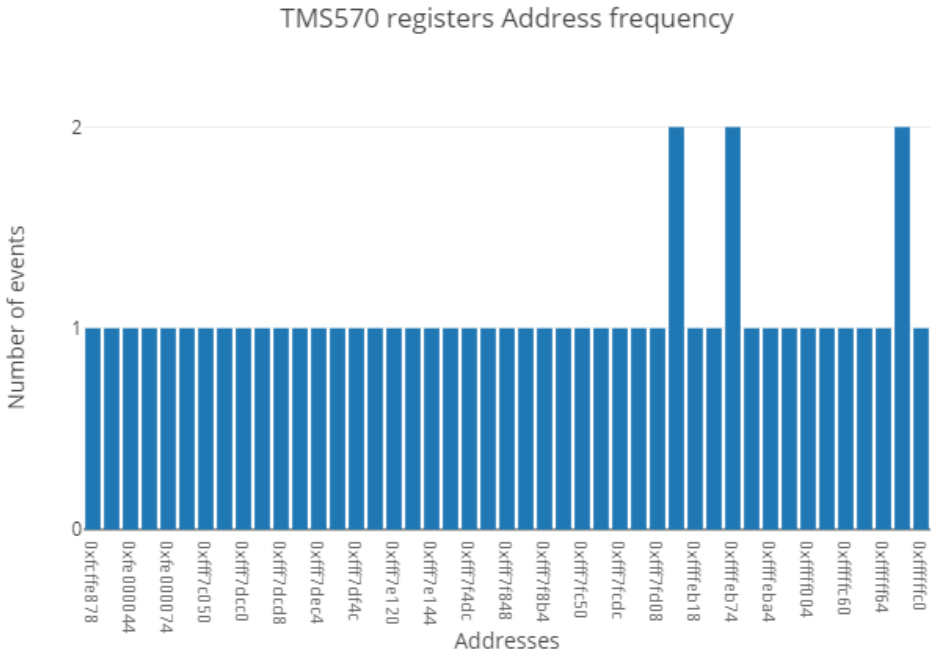


Figure 5.16: TMS570 number of single event upset in registers

5.4.5 LTC3887 registers and board status

This here we had some exciting effect observed in the LTC3887. In the radiation test, the devices values start alternating dramatically. This can be seen in the values in the frequency of events by address Figure 5.17. This is as well the case when looking at Figure 5.17 and Figure 5.18 the analog values read in this state does not make any sense. The system was recovered to it normal operation only after a rest was issued. The Figure 5.18 shows the current measured by LTC3887 is in the range of 60 A. This value is not corrent. It is clear when checking the total current consumption by the Egun circuitry that everything is usual. The explanation for this two events is that they occurred at the same times the registers start alternating dramatically.

$$\sigma = \frac{N_{event}}{F \times Bits} = \frac{64}{41.89 \times 10^9 \times 944} = 1.61 \times 10^{-12} \frac{cm^2}{bit} \pm 12.5\% \tag{5.5}$$

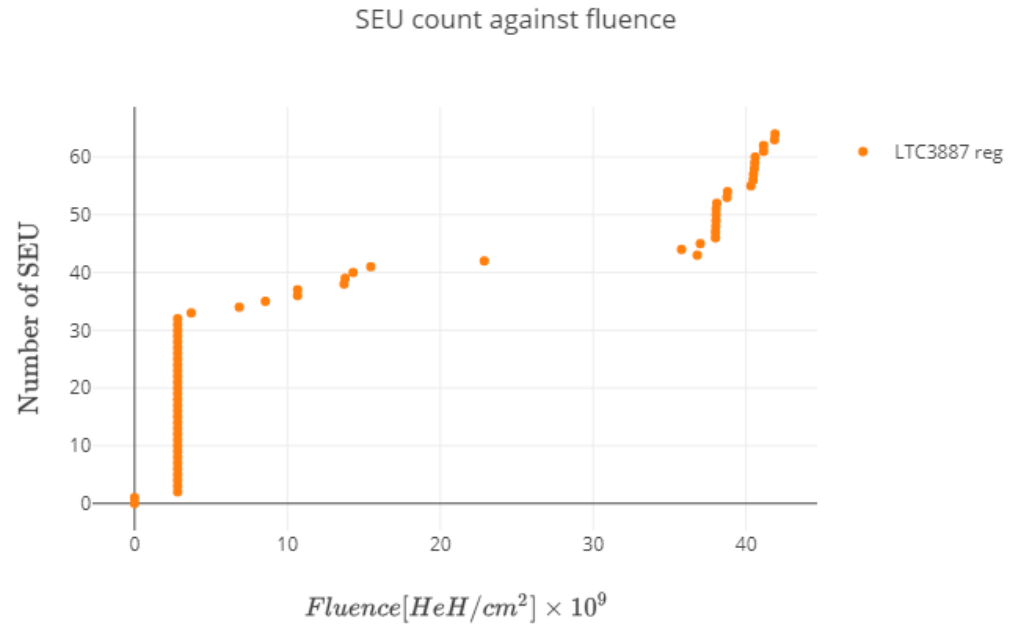


Figure 5.17: LTC3887 counted single event upset in registers

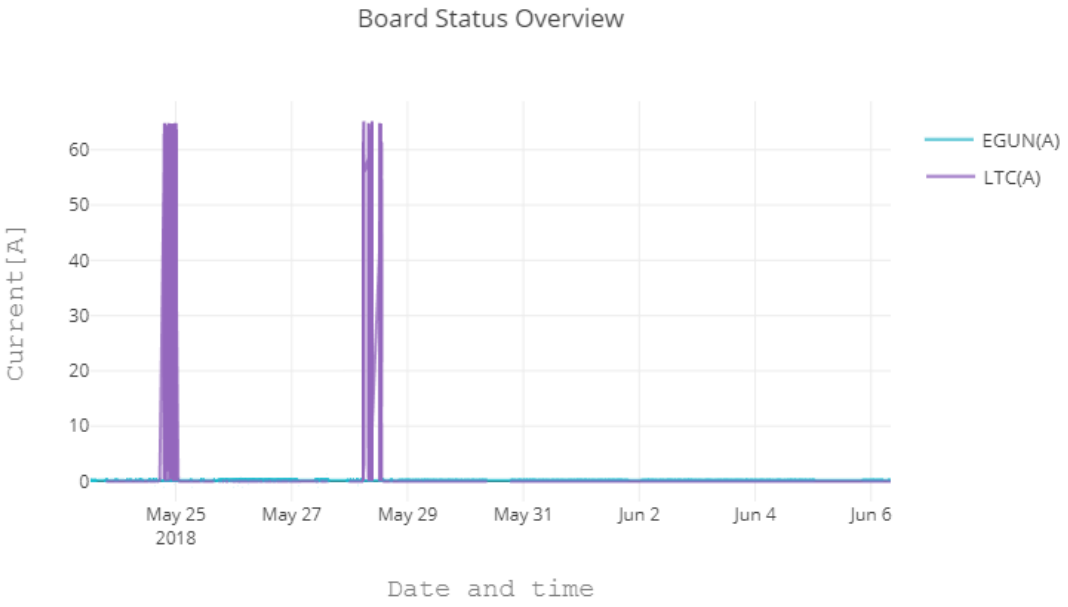


Figure 5.18: Board status LTC3887 current drawn

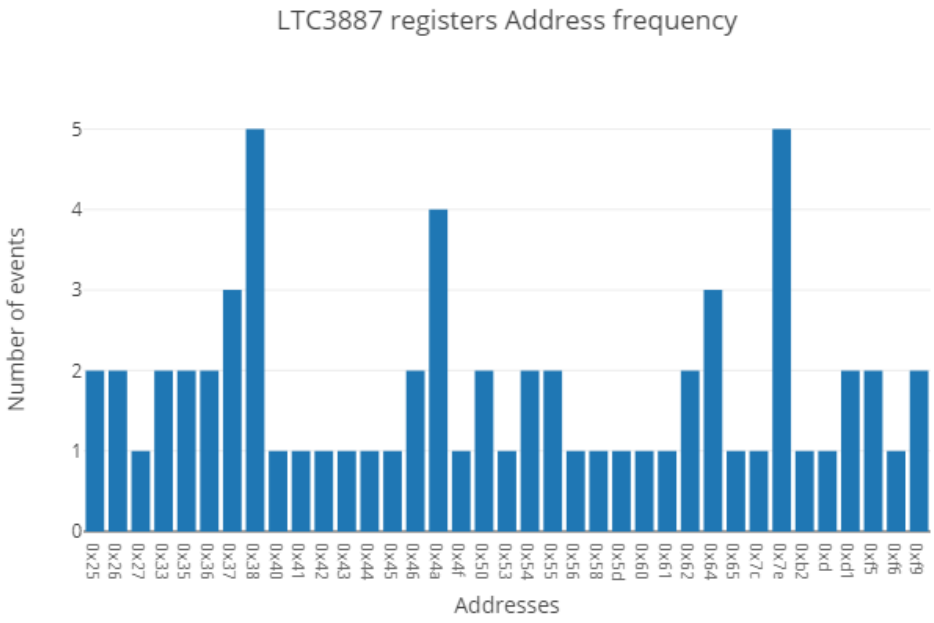


Figure 5.19: LTC3887 number of Single event upsets by address

5.4.6 AD7768 registers

The AD7768 ADC showed very little an over all small sensitivity to radiation. Three bit-flips on the first and the three on the second week. This test does not include analog data output from the ADC.

$$\sigma = \frac{N_{event}}{F \times Bits} = \frac{4}{4.09 \times 10^9 \times 256} = 4.77 \times 10^{-12} \frac{cm^2}{bit} \pm 50.0\% \tag{5.6}$$

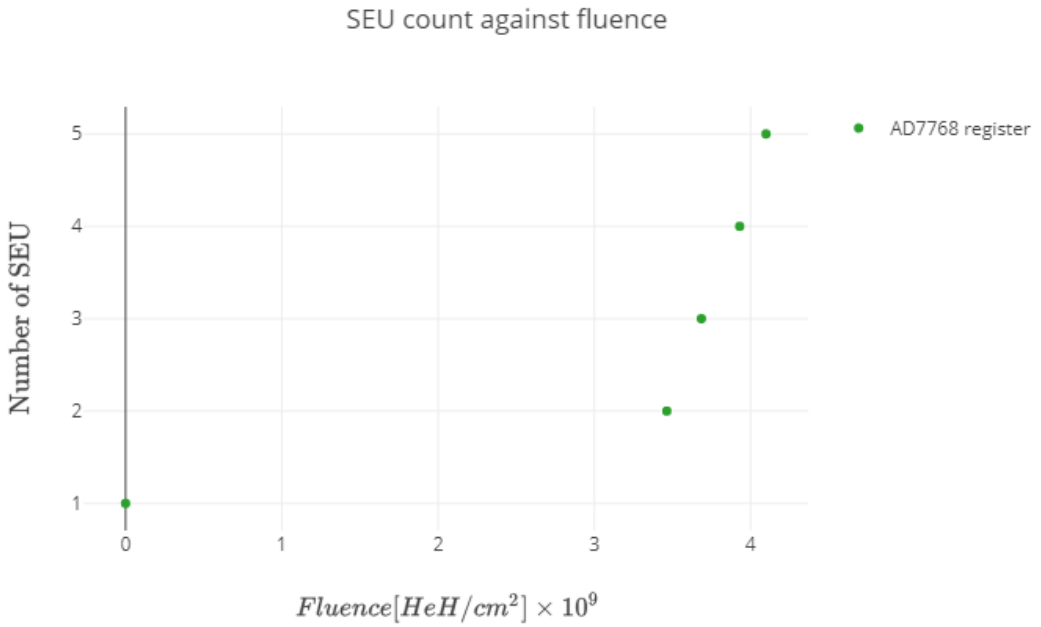


Figure 5.20: AD7768 counted single event upset in registers

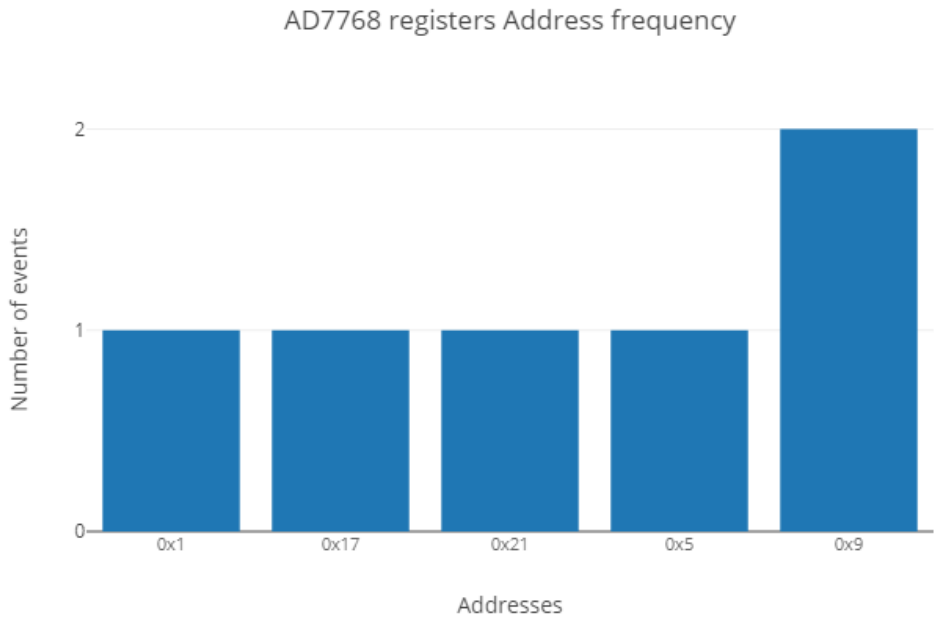


Figure 5.21: AD7768 number of Single event upsets by address

5.5 Discussion

Most of the targets that were set for this chapter were achieved. Along the way, some technical difficulties occurred during testing one related to the SRAM and other cloud base synchronization. The technical difficulties associated with the firmware is related turning the ECC on and off. In the first run we tough that the ECC was off, but after some hours of running, we found that no bit-flips where detected. After some examining we realized that it is a combination how of HALCOGEN and boot-loader behaves, this causes a wrong configuration of the error detection and correction module.

AD7768

We tested the digital part of the AD7768 ADC. The cross-section data we got is for bitflips in the control registers of the device. We did not collect data from the analog part of the device. To be sure about the sensitivity of the analog device one need to test the analog data output[47]. We had a limited amount of SRAM memory to be able to buffer data from ADC and test SRAM. The decision was made to prioritize the SRAM sensitivity over the data received from AD7768.

The choice was made as well since we had no idea on how sensitive the SRAM is to the radiation and how often we would detect bitflips. Testing both SRAM and AD7768 could prove to be hard as we would be able to say for sure where the bitflip has occurred. In future one could use Error Detection And Correction (EDAC) present in the Hercules microcontroller increase the reliability of SRAM. In that way, one can use the analog output of AD7768 with the knowledge that bitflips occurring in the TMS570 RAM are detected and correct.

LTC3887

The LTC3887 has experienced two SEFI it proven to be useful to check the analog values. This showed clearly that the device needed to be reset. Its necessary to add some monitoring function that can detect that the device is not functioning as expected. This can be used by monitoring the values of the analog circuitry.

SRAM

The SRAM pattern chosen was random. This pattern has more zeros than ones. The rule of thumb for testing SRAM is using a checkerboard pattern[8, 18]. This pattern consists of an alternating between 0s and 1s written to the memory. In general the SRAM data it shows that one can use ESM circuit as a detector for radiation. The same method can be used on flight as on board radiation detection unit.

We have no explaination about difference between the counted SEU for ECC on and off. The firmware used was ECC on where we have a small methods that disables the ECC functionality. We could not find an explaination on why such difference has occured.

SEL

No SEL was detected in the mix field environment. Devices that show SEL sensitivity is deemed unacceptable for space mission [48]. From the perspective of indirectly ionizing particles, O'Neill shows that probability of SEL is very low in short time missions [49]. However, since we tested in CHARM, we do not know the sensitivity of the devices to heavy particles is. One of the drawbacks of testing in CHARM is the temperature cycling in LEO cannot be simulated during the testing phase. The SEL is very depend on temperature [50].

Comparison with other Microctrollers

The data comparison with other complex devices from litterature. We could not find spesific values for registers.

Table 5.3: Data from literature for SEU and SEL

Devices	SEU SRAM	SEU Registers	SEL
TMS570	1.03×10^{-13}9.85%	1.16×10^{-12}14.9%	None
LTC3887	-	1.61×10^{-12}12.5%	None
AD7768	-	4.77×10^{-12}50%	None
MSP320[51]	1×10^{-12}	-	None
TMS570*[51]	1×10^{-12}	-	None

5.5.1 Future Work

Heavy ions testing of the circuit is required to determine the sensitivity. Quinn tested the TMS570 Hercules with heavy ions, and the results found that the device is not sensitive to SEL[51]. The LET used was up to 60 MeV$\times\frac{cm^2}{mg}$. Even though the test detected no SEL it s important to keep in mind that this is true for tested batch []. Changed by the produces can happen without notifying the users this may cause the device to become sensitive. Texas Instruments changed TMS570 from 130 nm node to 65nm in 2011[52]. To be able to test the device one need to remove the package to expose the die. The same test has to be done for LTC3887 and AD7768.

AD7768 have to undergo another test where the testing the analog and registers.

5.6 Conclusion

The devices tested at the CHARM facility show an acceptable amount of bitflips. The SRAM of the TMS570 has the larges value of 103 Single Event Upset (SEU) in one week. We have shown in the chapter that this can be corrected using software methods with Error Signaling Module (ESM) to ensure that no accumulations of errors will result in uncorrectable errors. In the case of the registers in all three devices, needs to implement periodic reconfigurations to write over any errors. Regarding Single Event Latchup (SEL) none where detected in hadrons environment. That can not be said about heavy ions. One need to decapped all the three devices to be able to test in a SEL heavy ion sensitivity.

6 | Conclusion

The results from the decapping process were promising but were restricted by the control ability of variables during the experiment. The tools used were mostly basic to do the process of decapping. This method requires sacrificing a good amount of devices before getting function device. It is necessary to expose the die to be able to do any form for heavy ions or laser testing. In the future, one should explore the use of a computer numerical control (CNC) laser to do the decapping process.

All the primary requirement put for the Circuit were achieved. The firmware is developed to be a command based systems that can send packages with about SRAM pattern, Registers of TMS570, AD7768, and LTC3887. On the top of that, a method of using the Error Signaling Module (ESM) is used as a bitflips monitor. This method allows the detects and corrects of errors to reduces the chance of uncorrectable double errors from occurring.

The devices tested at the CHARM facility show an acceptable amount of bitflips. The SRAM of the TMS570 has the larges value of 103 Single Event Upset (SEU) in one week. We have shown in the chapter that this can be corrected using software methods with Error Signaling Module (ESM) to ensure that no accumulations of errors will result in uncorrectable errors. In the case of the registers in all three devices, needs to implement periodic reconfigurations to write over any errors. Regarding Single Event Latchup (SEL) none where detected in hadrons environment. That can not be said about heavy ions. One need to decapped all the three devices to be able to test in a SEL heavy ion sensitivity.

Appendices

A | SPENVIS Spacecraft Trajectory

The trajectory is based on NORSAT-1 LEO orbit[21]. The mission duration is 1 year.

Table A.1: Spacecraft craft trajectory configuration in SPENVIS

Trajectory duration	30 days
Perige Altitude [km]	584
Apogee altitude [km]	607
Inclination [deg]	90.0
R.asc of asc. node [deg w.r.t. gamma50]	108.4140
Argument of perigee [deg]	67.8160
True anomaly[deg]	292.4826

B | m-NLP System Firmware

B.1 Command Line Interface

```
/*
 * Commandline.h
 *
 *  Created on: Sep 21, 2017
 *      Author: Yassine Elfarri
 */

#ifndef INC_COMMANDLINE_H_
#define INC_COMMANDLINE_H_

#include "sys_common.h"

#define True '1'
/** @def True
*   @brief Alias name for 1
*/

#define False '0'
/** @def False
*   @brief Alias name for 0
*/

/*############################################################ ASCII Chars
    ##############################################*/

#define CR '\r'
/** @def CR
*   @brief Alias name for Carriage return
*/
#define LF '\n'
/** @def LF
*   @brief Alias name for Line feed
*/
#define BS '\x8'
/** @def BS
*   @brief Alias name for Backspace
*/



/*############################################################ Help menu "?"
    ##############################################*/
enum cmd_commands
{
    SCI_HELP,/*!< corresponded to Help menu*/
    ENABLE_RAILS,/*!< Enabling rails 0 turning of all off and 7 is opposite*/
    IVMeasur, /*!< Read Current and Voltage values on enabled rails*/
    DAC_SPI_config,
    DAC_Gain_SEL_EN,
    DAC_Status,
    ANT_DTC_RLS_config,
    LTC_ON_OFF,
    LTC_Write,
    LTC_Read,
    LTC_tempature,
    LTC_I,
    LTC_Readback,
    LTC_get_V,
    LTC_set_voltage,
```

```
    AD7768_hetstart,
    AD7768_hetstop,
    AD7768_read_regs,
    AD7768_write_reg,
    AD7768_setMCLK,
    AD7768_setDCLK,
    AD7768_DRate,
    AD7768_setfilter,
    AD7768_start,
    AD7768_reset,
    AD7768_getregs,
    AD7768_print_settings,
    AD7768_print_samples,
    TMS570_reg_addr,
    TMS570_SRAM,
    TMS570_Status,
    TMS570_samples,
    SCI_RESET,/*!< Soft reset !NB reset only squential logic*/
};


#define lastcmd   SCI_RESET

typedef struct{
    unsigned char cmd_no;
    const char *command;
    const char *usage;
    const char *description;
}CLIMenu;

#define ltc_ch0 0



void wait(uint32 time);
void SCINewline(uint32 numOflines);
int SendStringSCI(char data[100], int Newline);
int SendRawSCI(int data, int Newline);
void SciEndCmd();
void SciSendACK(uint8 ack);
void SciInvalidCmd();
void SciInvalidArgs();
void SciTooManyargs();
void start_info();
void get_HWinfo(void);
void executeCmd(int cmdNum, char arg1[7],char arg2[7],char arg3[7]);
void init();
void Parsing();
void putChar(char c);
int toupper(int c);



#endif /* INC_COMMANDLINE_H_ */
```

```
/*
 * Commandline.c
 *
 *  Created on: Sep 21, 2017
 *      Author: Yassine Elfarri
 *
 *  @brief  uart command line file
*   @date 21-Sep-2017
*   @version 01.06.0
*
*   *   This file contains:
*   - Initializing need drivers
*   - uart commandline menu and parsing
*   .
*   This file is for commandline behavor.
 *
 */

#include "system.h"
#include "reg_system.h"
#include "stdio.h"
#include "sci.h"
#include "reg_sci.h"
#include "string.h"
#include "Commandline.h"
#include "SysBoard.h"
#include "adc.h"
#include "Sysboard.h"
```

```
29 #include "i2c.h"
30 #include "spi.h"
31
32
33
34 /** @typedef CmdTable[]
35 *   @brief containt command values, string and description
36 */
37 CLIMenu const CmdTable[] = {
38  {SCI_HELP                , "?"       , "-", "Available  command list."},
39  {ENABLE_RAILS            , "ENA512"  , "1", "<0-7> Enable/disable P5V,DCDC12V,P5V EGUN"},
40  {IVMeasur                , "IV"      , "-", "Voltage and current measurements"},
41  {DAC_SPI_config          , "DACC"    , "3", "<Channel>,<Mode>,<Voltage> value"},
42  {DAC_Gain_SEL_EN         , "DACSE"   , "1", "<0-3>Enable gain and select G10/20"},
43  {DAC_Status              , "DACST"   , "-", "Return value of select and Enable values"},
44  {ANT_DTC_RLS_config      , "ANT"     , "1", "<0-3>Antenna Realse and detect"},
45  {LTC_ON_OFF              , "LTCON"   , "1", "<0-1>Turn on/off ltc"},
46  {LTC_Write               , "LTCW"    , "-", "Write to LTC via I2C"},
47  {LTC_Read                , "LTCR"    , "-", "Read from LTC via I2C"},
48  {LTC_tempature           , "LTCT"    , "-", "Read from LTC internal temperature"},
49  {LTC_I                   , "LTCI"    , "-", "Read from LTC output current"},
50  {LTC_Readback            , "LTCRB"   , "-", "Readback config regs"},
51  {LTC_get_V               , "LTCGV"   , "-", "Get output voltage"},
52  {LTC_set_voltage         , "LTCSV"   , "1", "<1-3k>set output milivolt"},
53  {AD7768_hetstart         , "ADSTA"   , "-", "Start HET"},
54  {AD7768_hetstop          , "ADSTP"   , "-", "Stop HET"},
55  {AD7768_read_regs        , "ADR"     , "1", "<reg_addr> Read from register for AD"},
56  {AD7768_write_reg        , "ADW"     , "2", "<reg_addr><value>write to register for AD"},
57  {AD7768_setMCLK          , "ADMCK"   , "1", "<32-8-4>Set MCLK division for AD"},
58  {AD7768_setDCLK          , "ADDCK"   , "1", "<8-4-2> Set DCLK division for AD"},
59  {AD7768_DRate            , "ADDRA"   , "1", "<32-64-128-256-512-1024> set Drate for AD"},
60  {AD7768_setfilter        , "ADSTF"   , "1", "<0=wide-1=sinc5>Set filter for AD"},
61  {AD7768_start            , "ADST"    , "-", "Start AD7768"},
62  {AD7768_reset            , "ADRST"   , "-", "Reset AD7768"},
63  {AD7768_getregs          , "ADGRS"   , "-", "Read all readable reg from AD"},
64  {AD7768_print_settings   , "ADCON"   , "-", "values for MCLK_DIV, fmod,decRate, ODC, and DCLK "
      },
65  {AD7768_print_samples    , "ADC"     , "-", "<number of samples> print number of samples"},
66  {TMS570_reg_addr         , "REG"     , "-", "print all registers and address"},
67  {TMS570_SRAM             , "SRAM"    , "-", "Request SRAM values"},
68  {TMS570_Status           , "ST"    , "-", "Request SRAM values"},
69  {TMS570_samples          , "SAM"     , "-", "Request SRAM values"},
70  {SCI_RESET               , "RESET"   , "-", "Cold reboot"}
71  };
72
73 uint8 CommandEntered =0; // indicates where to reformating the received command
74 char PromptChar[50]; // stored SCI RX value
75
76
77 /** @fn void Parsing()
78 *   @brief Function for reformating received chars from command line used in main
79 *
80 *   User can add arguments in this function as well.
81 *
82 */
83 void Parsing()
84 {
85     uint8 x,y;
86     char cmdData[20];
87     int cmdvalue;
88     int SWCaseVal=0;
89
90     char args1[7] = {0};
91     char args2[7] = {0};
92     char args3[7] = {0};
93
94
95
96     if(CommandEntered == 1)
97       {
98
99                       //save the first section of command
100                    for(x=0; x<50;x++)
101                    {
102                        if(PromptChar[x] != 0){
103                            cmdData[x]= PromptChar[x];
104                        }
105                        else
106                        {
107                            while(x<50)
108                            {
109                                    if(PromptChar[x] != 0)
110                                    {
```

```
111                                     switch (SWCaseVal)
112                                     {
113                                           case 1:
114                                             if(y < sizeof(args1)-1){
115                                                 args1[y] = PromptChar[x];
116                                              }
117                                              else
118                                              {
119                                                 SciSendACK(2);
120                                                 SciInvalidArgs();
121                                                 goto exit;
122                                              }
123                                              break;
124
125                                              case 2:
126                                              if(y < sizeof(args2-1)){
127                                                   args2[y] = PromptChar[x];}
128                                              else
129                                              {
130                                                   SciSendACK(2);
131                                                   SciInvalidArgs();
132                                                   goto exit;
133                                              }
134                                              break;
135
136                                              case 3:
137                                                   if(y < sizeof(args3-1)){
138                                                   args3[y] = PromptChar[x];}
139                                                   else
140                                                   {
141                                                   SciSendACK(2);
142                                                   SciInvalidArgs();
143                                                   goto exit;
144                                                   }
145                                                   break;
146                                              // more args can be added here
147                                           }
148                                            y++;
149                                       }
150                                       else
151                                       {
152                                          y=0;
153                                          if(SWCaseVal <4){
154                                          SWCaseVal++;}
155                                       }
156
157                                  x++;
158
159                              }
160                          }
161
162
163
164                     }
165
166
167                     for(y=0; y <= lastcmd;y++)
168                     {
169
170                          if(strcmp(CmdTable[y].command,cmdData) == 0)
171                          {
172                            cmdvalue=CmdTable[y].cmd_no;
173                            executeCmd(cmdvalue,args1,args2,args3);
174                            goto exit;
175                          }
176                     }
177
178                     if(strlen(( char*)PromptChar) != 0)
179                      {
180                           SciSendACK(2);
181                           SciInvalidCmd();
182                      }
183
184               exit:
185
186                  // clear strings
187                 for(x=0; x<sizeof(PromptChar);x++ )
188                 {
189                   PromptChar[x] = 0;
190                 }
191
192                  for(x=0; x<=sizeof(cmdData);x++ )
193                  {
```

```
194                     cmdData[x]=0;
195                 }
196                 for(x=0; x<=sizeof(args1); x++)
197                 {
198                     args1[x]=0;
199                     args2[x]=0;
200                     args3[x]=0;
201                 }
202                 y=0;
203                 x=0;
204                 SWCaseVal =0;
205
206                 SCINewline(1);
207                 CommandEntered = 0;
208
209
210
211                  }
212
213
214 }
215
216
217 /** @fn void putChar(char c)
218 *   @brief This function is used by sci interrupt to add single char to array of chars.
219 *   @param[in] c receives a single char
220 *
221 *   This method does not need to be changed by the user. This function does trigger as well
     on carried return and backspace.
222 *
223 */
224
225 void putChar(char c)
226 {
227     static volatile int i;
228
229     if ( i == 49)
230     {
231         while(i > -1)
232         {
233             PromptChar[i--] = 0;
234         }
235
236         sciSend(scilinREG, 2,(unsigned char *)&"\r\n");
237         while ( !sciIsTxReady(scilinREG));
238     }
239
240     else if(c == '\x8')
241     {
242         if(i == 0){PromptChar[i] = 0;}
243         else {
244         PromptChar[i] = 0;
245         PromptChar[i-1]=0;
246         --i;
247         }
248
249         sciSend(scilinREG, 1,(unsigned char *)&"\x8");
250         while ( !sciIsTxReady(scilinREG));
251         sciSend(scilinREG, 1,(unsigned char *)&" ");
252         while ( !sciIsTxReady(scilinREG));
253         sciSend(scilinREG, 1,(unsigned char *)&"\x8");
254         while ( !sciIsTxReady(scilinREG));
255
256
257     }
258
259     else if(c == '\r')
260     {
261         PromptChar[i] = 0;
262         // start register value
263         CommandEntered = 1;
264         sciSend(scilinREG, 1,(unsigned char *)&" ");
265         while ( !sciIsTxReady(scilinREG));
266         sciSend(scilinREG, 1,(unsigned char *)&"\x8");
267         i=0;
268     }
269     else if(c == '\x20')
270     {
271         PromptChar[i++] = 0;
272         sciSend(scilinREG, 1,(unsigned char *)&" ");
273         while ( !sciIsTxReady(scilinREG));
274     }
275
```

```
276
277     else {
278          PromptChar[i] = toupper(c);
279          sciSend(scilinREG,1,(unsigned char *)&PromptChar[i++]);
280          while ( !sciIsTxReady(scilinREG));
281     }
282
283     //wait(30);
284     while ( !sciIsTxReady(scilinREG));
285
286
287 }
288
289 /** @fn void executeCmd(int cmdNum, char arg1[],char arg2[])
290 *   @brief this function executes command modes
291 *   @param[in] type int value passed by parsing() to select meny
292 *   @param[in] first argument of type char passed by parsing()
293 *   @param[in] second argument of type char passed by parsing()
294 *
295 *   The is already implemented to received values from parsing function. Here the user can
     add more menu values
296 *
297 */
298
299 void executeCmd(int cmdNum, char arg1[],char arg2[],char arg3[])
300 {
301   char  tempChar[100]={0};
302   int x;
303
304   for(x=0; x<sizeof(PromptChar);x++ )
305   {
306       PromptChar[x] = 0;
307    }
308
309   switch(cmdNum)
310   {
311       case SCI_HELP:
312
313                               if(arg1[0]!= 0 || arg2[0]!= 0)
314                               {
315                                   SciSendACK(2);
316                                   //SciTooManyargs();
317                                   //goto Error;
318                               }
319
320                               SciSendACK(1);
321                               for(x=0;x<=lastcmd;x++)
322                               {
323                                   snprintf(tempChar,sizeof(tempChar)," %9s   %5s    %s"
324                                                                       ,CmdTable[x].command,CmdTable[x].
     usage,
325                                                                       CmdTable[x].description);
326                                   while ( !sciIsTxReady(scilinREG));
327                                   SendStringSCI(tempChar,True);
328
329                               }
330
331                               break;
332
333       case ENABLE_RAILS:
334                               cmd_enable_rails(atoi(arg1), atoi(arg2));
335                               break;
336
337
338       case IVMeasur:
339                               cmd_IV_measur(atoi(arg1),atoi(arg2));
340                               break;
341
342
343       case DAC_SPI_config:
344                                 cmd_DAC_SPI_config(atoi(arg1), atoi(arg2),atoi(arg3));
345                                 break;
346
347
348       case DAC_Gain_SEL_EN:
349                                cmd_DAC_Gain_Sel_EN(atoi(arg1),atoi(arg2));
350                                break;
351
352
353       case DAC_Status:
354                               cmd_DAC_GAIN_SET_STATUS(atoi(arg1), atoi(arg2));
355                               break;
356       case ANT_DTC_RLS_config:
```

```
357                             cmd_Antenna_RLS_DTCT(atoi(arg1),atoi(arg2));
358                             break;
359         case LTC_ON_OFF:
360                             cmd_ltc3887_turn_on_off_channel(atoi(arg1));
361                             break;
362         case LTC_Write:
363                           if(atoi(arg1) == 0 || atoi(arg2) ==  0)
364                           {
365                                 SciSendACK(2);
366                           }
367                           SendStringSCI("Not implemented",True);
368                           break;
369         case LTC_Read:
370                           if(atoi(arg1) == 0 || atoi(arg2) ==  0)
371                           {
372                                 SciSendACK(2);
373                           }
374                           SendStringSCI("Not implemented",True);
375                           break;
376
377         case LTC_tempature:
378                           cmd_ltc3887_Read_internal_Temperature(atoi(arg1),atoi(arg2));
379                           break;
380         case LTC_I:
381                           cmd_ltc3887_output_current(atoi(arg1),atoi(arg2));
382                           break;
383         case LTC_Readback:
384                             cmd_ltc3887_read_config();
385                             break;
386         case LTC_get_V:
387                             cmd_ltc3887_Read_output_voltage(atoi(arg1),atoi(arg2));
388                             break;
389         case LTC_set_voltage:
390                            cmd_ltc3887_set_output_voltage(atoi(arg1),0);
391                             break;
392
393         case AD7768_hetstart:
394                         start_het();
395                         break;
396         case AD7768_hetstop:
397                         stop_het();
398                         break;
399         case AD7768_read_regs:
400                         cmd_AD7768_GET_REG((uint8)atoi(arg1));
401                         break;
402         case AD7768_write_reg:
403                         cmd_AD7768_SET_REG((uint8)atoi(arg1),(uint8)atoi(arg2));
404                         break;
405         case AD7768_setMCLK:
406                          cmd_AD7768_SET_MCLK_div(atoi(arg1));
407                          break;
408         case AD7768_setDCLK:
409                         cmd_AD7768_SET_DCLK_div(atoi(arg1));
410                         break;
411         case AD7768_DRate:
412                         cmd_AD7768_SET_DRATE(atoi(arg1));
413                         break;
414         case AD7768_setfilter:
415                         cmd_AD7768_SET_FILTER(atoi(arg1));
416                         break;
417         case AD7768_start:
418                         cmd_AD7768_run();
419                         break;
420         case AD7768_reset:
421                         cmd_AD7768_stop();
422                         break;
423         case AD7768_getregs:
424                         cmd_AD7768_GET_AllConfigs();
425                         break;
426         case AD7768_print_settings:
427                         ad7768_print_settings();
428                         break;
429         case AD7768_print_samples:
430                         cmd_AD7768_print_Data();
431                         break;
432         case TMS570_reg_addr:
433                         reg_data_cmd();
434                         break;
435
436         case TMS570_SRAM:
437                         RAM_data_read();
438                         break;
439         case TMS570_Status:
```

```
440                         raw_board_status();
441                         dac_status();
442                         break;
443
444         case TMS570_samples:
445
446                             raw_board_status();
447                             dac_status();
448                             reg_data_cmd();
449                             wait(200);
450                             RAM_data_read();
451                             wait(200);
452                             //cmd_AD7768_print_Data();
453                             //wait(100);
454                             break;
455
456         case SCI_RESET:
457                             if(arg1[0]!= 0 || arg2[0]!= 0)
458                             {
459                                     SciSendACK(2);
460                                     //SciTooManyargs();
461                                     //goto Error;
462                             }
463                             SciSendACK(1);
464                             while ( !sciIsTxReady(scilinREG));
465                             SendStringSCI("Restarting..",True);
466                             wait(3000);
467                             systemREG1->SYSECR = (0x10) << 14;
468
469             break;
470
471             /**       To add new arguments you need to add a new switch case:
472                 *     - First define value in Commandline.h e.g SCI_HELP
473                 *     - Add line to CmdTable[] in Commandline.c
474                 *     - add argument in function Parsing() under switch(SWCaseVal)
475                 *     - Still in parsing() increase  value in if(SWCaseVal < value)
476             */
477
478     }
479
480     //Error:
481      //SciEndCmd();
482 }
483
484
485 /** @fn void init()
486 *   @brief add modules that needs to be initiated to this function
487 *
488 *
489 *   This function is the first function to be run during startup/rest of mcu
490 *
491 */
492
493 void init()
494 {
495         /*Initialize sci*/
496         sciInit();
497         /*Enable SCI interrupt*/
498         sciEnableNotification(scilinREG, SCI_TX_INT);
499         gioInit();
500         hetInit();
501         stop_het();
502         canInit();
503         spiInit();
504
505         /*internal ADC*/
506         adcInit();
507         adcResetFiFo(adcREG1, adcGROUP1);
508         adcStartConversion (adcREG1, adcGROUP1);
509
510         /*I2C ltc3887 digital switcher*/
511         i2cInit();
512         i2cSetBaudrate(i2cREG1, 100);
513         i2cSetMode(i2cREG1, I2C_MASTER);
514
515         /*Turn on the rails*/
516         P5V0_EN_PIN_On;
517         DCDC12V_EN_PIN_On;
518         P5V0_EGUN_EN_PIN_On;
519         wait(1000);
520
521         ad7768_init();
522         /*Enable interrupt*/
```

```
523         _enable_IRQ();
524
525         /*Enable SCI interrupt*/
526          sciEnableNotification(scilinREG, SCI_RX_INT);
527
528          /*Config LTC3887*/
529          ltc3887_init();
530          ltc3887_turn_on_off_channel(0,true);
531          start_het();
532
533          RAMtest_init();
534
535          start_info();
536          esm_monitor();
537
538
539
540 }
541
542
543
544
545 void wait(uint32 time)
546 {
547     while(time--);
548 }
549
550
551
552 /** @fn void SCINewline(int newline)
553 *   @brief simplify the use of uart by packing it in function
554 *   @param[in] data[] is char value that will be send
555 *   @param[in] newline True will result in newline in terminal.
556 *
557 *   @return  returns 1 if function is successful
558 *
559 *   simplifies the use of uart example user needs only  SendStringSCI("This is a string",True
    );
560 *
561 */
562
563 void SCINewline(uint32 numOflines)
564 {
565     while( numOflines--){
566         // wait for Tx buffer to be ready
567         while ( !sciIsTxReady(scilinREG));
568         sciSend(scilinREG, 2,(unsigned char *)&"\r\n");
569     };
570
571 }
572
573 /** @fn int SendStringSCI(char data[, int Newline)
574 *
575 *   @brief simplify the use of uart by packing it in function
576 *   @param[in] data[] is char value that will be sent
577 *   @param[in] newline True will result in newline in terminal.
578 *
579 *   @return  returns 1 if function is successful
580 *
581 *   simplifies the use of uart example user needs only  SendStringSCI("This is a string",True
    );
582 *
583 *   True indicate starting a newline.
584 */
585
586 int SendStringSCI(char data[], int Newline)
587 {
588     uint32 i;
589     uint32 size ;
590     size = strlen(data);
591
592     for(i = 0; i <size; ++i)
593     {
594         while ( !sciIsTxReady(scilinREG));
595         sciSend(scilinREG,1, (unsigned char *) &data[i]);
596
597     }
598
599     if(Newline == True)
600     {
601         while ( !sciIsTxReady(scilinREG));
602         sciSend(scilinREG, 2,(unsigned char *)&"\r\n");
603     }
```

```
604
605     return 1;
606 }
607
608 int SendRawSCI(int data, int Newline)
609 {
610
611     while ( !sciIsTxReady(scilinREG));
612     sciSend(scilinREG,4,(uint8*)&data);
613
614     if(Newline == True)
615     {
616         while ( !sciIsTxReady(scilinREG));
617         sciSend(scilinREG, 2,(unsigned char *)&"\r\n");
618     }
619
620     return 1;
621 }
622
623 void SciEndCmd()
624 {
625     while ( !sciIsTxReady(scilinREG));
626     SendStringSCI("+",True);
627 }
628
629 /** @fn void SciSendACK(uint8 ack)
630 *
631 *   @brief Ack or NAK depending on input value
632 *   @param[in] ack show if command is ack or not:
633 *              - ack is 1 the command is acknoledged
634 *              - ack is 2 the command is not acknoledged
635 *
636 *   this should be added in the good for give feedback to the user
637 */
638 void SciSendACK(uint8 ack)
639 {
640    if(ack == 1)
641    {
642        while ( !sciIsTxReady(scilinREG));
643        SendStringSCI("...<ACK>", True);
644    }
645    else if(ack == 2)
646    {
647        while ( !sciIsTxReady(scilinREG));
648        SendStringSCI("...<NAK>", True);
649    }
650    while ( !sciIsTxReady(scilinREG));
651    SCINewline(1);
652 }
653
654 /** @fn void SciInvalidArgs()
655 *
656 *   @brief If command is not found
657 *
658 *
659 *   Seen in terminal if the command entered is not found
660 */
661 void SciInvalidCmd()
662 {
663   while ( !sciIsTxReady(scilinREG));
664   SendStringSCI("Invalid command",True);
665 }
666
667 /** @fn void SciInvalidArgs()
668 *
669 *   @brief feedback to user  about args overflow
670 *
671 *
672 *   This is show in terminal when the argumenmts surpase 8 bit value
673 */
674
675 void SciInvalidArgs()
676 {
677    while ( !sciIsTxReady(scilinREG));
678    SendStringSCI("Invalid arguments",True);
679 }
680
681 /** @fn void SciTooManyargs()
682 *
683 *   @brief feedback to user
684 *
685 *
686 *   This see when user enters arguments for commands that does not require them
```

```
687 */
688
689
690 void SciTooManyargs()
691 {
692    while ( !sciIsTxReady(scilinREG));
693    SendStringSCI("Too many cmd args",True);
694 }
695 /** @fn void start_info(void)
696 *
697 *   @brief this used to share screen on debugg mode.
698 *
699 *
700 * This function displayed bebugg function of the system
701 */
702 void start_info()
703 {
704     /*Send to user prompt*/
705
706             // header
707             /* build info*/
708             char developer[20] = " Yassine Elfarri";
709             char SWRev[10] = " 2.0.1";
710             SCINewline(1);
711             SendStringSCI(" == UIO Hercules MCU safety ==", True);
712             while ( !sciIsTxReady(scilinREG));
713
714             SendStringSCI(" Dev:",False);
715             SendStringSCI(developer,False);
716             SendStringSCI(" SWRev:",False);
717             SendStringSCI( SWRev,True);
718
719             while ( !sciIsTxReady(scilinREG));
720             get_HWinfo();
721             SendStringSCI("= Type ? for help =", True);
722
723 }
724 /** @fn void get_HWinfo(void)
725 *
726 *   @brief reads and sends values of LOT and WAFER to UART
727 *
728 *
729 *   This function reads and print manufacturing value LOT WAFER to UART
730 */
731
732 void get_HWinfo(void)
733 {
734         uint32 LOT_NUM, WAFER_num;
735         uint8 WAFER_Y_COOR, WAFER_X_COOR;
736
737         /** - LOT number */
738         LOT_NUM = ((systemREG1->DIEIDH) & 0x00FFFFFF );
739
740         /** - Wafer number */
741         WAFER_num = ((systemREG1->DIEIDL) >> 24  ) ;
742
743         /** - Wafer Y coordinates*/
744         WAFER_Y_COOR = (((systemREG1->DIEIDL) & 0x00FFF000) >> 12  );
745
746         /** - Wafer X coordinates*/
747         WAFER_X_COOR = (((systemREG1->DIEIDL) & 0x00000FFF));
748
749         obc_debug("LOT#,%d,WAFER#,%d,WAFER_Y,%d,WAFER_X,%d",LOT_NUM,WAFER_num,WAFER_Y_COOR,
    WAFER_X_COOR);
750
751         //Clock frequency
752         //cast float to an int
753         uint8 value =(uint8) GCLK_FREQ;
754         obc_debug("System Clock  -> %d MHz", value);
755
756         //Serial port
757         //TODO get current value
758         obc_debug("SCI Baud rate -> 230400");
759 }
```

B.2 Drivers and Functionality

```
1 /*
2  * SysBoard.h
```

```
 *
 *  Created on: Sep 21, 2017
 *      Author: Yassine Elfarri
 */

#ifndef INC_SysBoard_H_
#define INC_SysBoard_H_
#include "adc.h"
#include "gio.h"
#include "het.h"
#include "can.h"
#include "stdio.h"
#include "stdlib.h"
#include "Commandline.h"
#include "reg_adc.h"
#include "spi.h"
#include "reg_spi.h"
#include "i2c.h"
#include "sci.h"
#include "reg_sci.h"
#include "math.h"
#include "htu.h"
#include "sys_vim.h"
#include "system.h"
#include "obc_communication.h"


/*================================ SWITCHES ======================================*/

/** @def Led1On
*   @brief set Led 1 to "on"
*
*   Turn on led1(D14) on mNLP board
*/
#define Led1On  gioSetBit(hetPORT1, PIN_HET_30, 0)
/** @def Led1Off
*   @brief Set Led 1 to "off"
*
*   Turn off led1(D14) on mNLP board
*/
#define Led1Off gioSetBit(hetPORT1, PIN_HET_30, 1)
/** @def Led2On
*   @brief Set Led 2 to "on"
*
*   Turn on led2(D8) on mNLP board
*/
#define Led2On  gioSetBit(hetPORT1, PIN_HET_17, 0)
/** @def Led2Off
*   @brief Led 2 off
*
*   Turn off led2(D8) on mNLP board
*/
#define Led2Off gioSetBit(hetPORT1, PIN_HET_17, 1)

/** @def Led1Toggle
*   @brief Led 1 toggle value
*
*   Turn off led1(D14) on mNLP board
*/
#define Led1Toggle gioSetBit(hetPORT1, PIN_HET_30, gioGetBit(hetPORT1, PIN_HET_30) ^ 1)

/** @def Led2Toggle
*   @brief Led 2 toggle value
*
*   Turn off led2(D8) on mNLP board
*/
#define Led2Toggle gioSetBit(hetPORT1, PIN_HET_17, gioGetBit(hetPORT1, PIN_HET_17) ^ 1)


/** @def P5V0_EN_PIN_On
*   @brief Turn On enables current sense x21 to do measurements.
*
*   Turns on transistor P19 for current sense(x21) on mNLP board. When On LED P5 on analog
    PWR will light up.
*   - Current can be read by Hercules's ADC Curr_P5V0(Pin 62) AD1IN
*/
#define P5V0_EN_PIN_On gioSetBit(hetPORT1, PIN_HET_9, 1)

/** @def P5V0_EN_PIN_Off
*   @brief Turn Off disables current sense x21 measurments
*
*   Turns off transistor P19 for current sense(x21) on mNLP board. When Off LED P5 on analog
    PWR will off or dim.
```

```
84 */
85
86 #define P5V0_EN_PIN_Off gioSetBit(hetPORT1, PIN_HET_9, 0)
87
88 /** @def DCDC12V_EN_PIN_Off
89 *   @brief Turn Off disables current sense x22(P12v),X23(N12) and output +/- 12V
90 *
91 *   Turns off transistor P14 for current senses (x22 and x23) on mNLP board.When Off LED P12
     and N12 on analog PWR will off or dim.
92 *   - Turns off +/- 12V
93 *   - Disable current sense
94 */
95
96 #define DCDC12V_EN_PIN_Off gioSetBit(hetPORT1, PIN_HET_22, 0)
97
98 /** @def DCDC12V_EN_PIN_On
99 *   @brief  enables current sense x22(P12v),X23(N12) and output +/- 12V
100 *
101 *   Turns on transistor P14 for current senses (x22 and x23) on mNLP board.When On LED P12
     and N12 on analog PWR will on.
102 *   - !!Obs turning on this function will output a volatage of 25V rating !!!
103 *   - Enables positive and negative 12V output to langmiur
104 *   - Enable hercules ADC current sense CURR_N12V(pin 70 AD1IN[70]) and CURR_P12V (pin 71
     AD1IN[00])
105 */
106
107 #define DCDC12V_EN_PIN_On gioSetBit(hetPORT1, PIN_HET_22, 1)
108
109 /** @def P5V0_EGUN_EN_PIN_Off
110 *   @brief  disables current sense x16 and EGUN functionality
111 *
112 *   Turns off transistor P7 for current senses x16 on mNLP board.When On LED D13 lights on.
113 *   - !!Obs turning on this function will output a high volatage !!!
114 *   - This disables voltage to EGUN
115 *   - NO current measurements can be done with hercules ADC
116 */
117
118 #define P5V0_EGUN_EN_PIN_Off gioSetBit(hetPORT1, PIN_HET_25, 0)
119
120 /** @def P5V0_EGUN_EN_PIN_On
121 *   @brief  enables current sense x16 and turns on EGUN
122 *
123 *   Turns on transistor P7 for current senses x16 on mNLP board.When On LED D13 lights on.
124 *   - !!Obs turning on this function will output a high volatage !!!
125 *   - Enables use of EGUN
126 *   - Enable current sense done by hercules in pin CURR_P5V0_EGUN(pin 65) AD1IN(21)
127 */
128 #define P5V0_EGUN_EN_PIN_On gioSetBit(hetPORT1, PIN_HET_25, 1)
129 /*! This struct holds values of the switches. */
130 typedef struct Power_switches{
131     int P5V0_EN_PIN; /*!< 5V rail to frontend and AD7768*/
132     int DCDC12V_EN_PIN; /*!< DCDC -12 and +12 volt rail*/
133     int P5V0_EGUN_EN_PIN; /*!< 5V rail for Egun*/
134 }Power_Rails;
135 /*============================== Hercule's ADC ======================================*/
136
137 /** @def ADCmVPerBit
138 *   @brief ADC resolution which is mV per bits
139 *
140 */
141 #define ADCmVPerBit    3300.0/4095.0
142 /** @def ADC_Channels_cnt
143 *   @brief total number of channels that is read from
144 *
145 */
146 #define ADC_Channels_cnt       10
147 /*! This enum holds values for the order of the adc channels, the ADC multiplexes between
148  * channels starting from the lowest channel AD1IN[00] number to the highest. */
149 enum adc_group1_channels{
150     P12V_V_ID , /*!< corresponded to channel AD1IN[00]*/
151     P12V_I_ID,  /*!< corresponded to channel AD1IN[01]*/
152     P5V0_V_ID,  /*!< corresponded to channel AD1IN[04]*/
153     P3V3_I_ID,  /*!< corresponded to channel AD1IN[07]*/
154     N12V_I_ID,  /*!< corresponded to channel AD1IN[09]*/
155     P3V3_V_ID,  /*!< corresponded to channel AD1IN[11]*/
156     N12V_V_ID,  /*!< corresponded to channel AD1IN[17]*/
157     P1V2_I_ID,  /*!< corresponded to channel AD1IN[18]*/
158     P5V0_I_ID,  /*!< corresponded to channel AD1IN[19]*/
159     EGUN_I_ID,  /*!< corresponded to channel AD1IN[21]*/
160 };
161
162 /*! This struct holds global values that conversion from from ADC. */
163 typedef struct MeasuredVI{
```

```
164     int P12V_V; /*!< Voltage P12 to channel AD1IN[00]*/
165     int P12V_I; /*!< Current P12 to channel AD1IN[01]*/
166     int P5V0_V; /*!< Voltage P5V0 to channel AD1IN[04]*/
167     int P3V3_I; /*!< Current P3V3 to channel AD1IN[07]*/
168     int N12V_I; /*!< Current N12V to channel AD1IN[09]*/
169     int P3V3_V; /*!< Voltage P3V3 to channel AD1IN[11]*/
170     int N12V_V; /*!< Voltage N12V to channel AD1IN[17]*/
171     int P1V2_I; /*!< Current P1V2 to channel AD1IN[18]*/
172     int P5V0_I; /*!< Current P5V0 to channel AD1IN[19]*/
173     int EGUN_I; /*!< Current EGUN to channel AD1IN[21]*/
174 }VI;
175 extern struct MeasuredVI MON_I_V;
176
177 /** @def VMONP5V0
178 *   @brief  Voltage ratio based on voltage dividers  used to get correct value
179 */
180 #define VMONP5V0            2.0
181 /** @def VMON_P3V3
182 *   @brief  Voltage ratio based on voltage dividers  used to get correct value
183 */
184 #define VMON_P3V3           2.0
185 /** @def VMON_P12V
186 *   @brief  Voltage ratio based on voltage dividers  used to get correct value
187 */
188 #define VMON_P12V           5.02
189 /** @def VMON_N12V
190 *   @brief  Voltage ratio based on voltage dividers  used to get correct value
191 */
192 #define VMON_N12V           5.0
193
194 /** @def VMON_N12V
195 *   @brief  the current measured on shunt resistor by current sens
196 */
197 #define ADC_IMON           1.0/(200.0*0.05)
198 /** @def Magnitude_threshold1
199 *   @brief  magnitude threshold interrupt, is it used full ?
200 */
201 #define Magnitude_threshold1
202
203 /*============================== Antenna release and detection ==============================
        */
204
205 /** @def Antenna_Release_Reg
206 *   @brief  GIO register for antenna release
207 */
208 #define Antenna_Release_Reg     gioPORTA
209
210 /** @def Ant_RLS_1_On
211 *   @brief  Switch RLS 1 on
212 */
213 #define Ant_RLS_1_On    gioSetBit(gioPORTA, 7, 1)
214 /** @def Ant_RLS_1_On
215 *   @brief  Switch RLS 1 off
216 */
217 #define Ant_RLS_1_Off   gioSetBit(gioPORTA, 7, 0)
218 /** @def Ant_RLS_2_On
219 *   @brief  Switch RLS 2 on
220 */
221 #define Ant_RLS_2_On    gioSetBit(gioPORTA, 6, 1)
222 /** @def Ant_RLS_2_Off
223 *   @brief  Switch RLS 2 off
224 */
225 #define Ant_RLS_2_Off   gioSetBit(gioPORTA, 6, 0)
226 /** @def Ant_RLS_3_On
227 *   @brief  Switch RLS 3 On
228 */
229 #define Ant_RLS_3_On    gioSetBit(gioPORTA, 5, 1)
230 /** @def Ant_RLS_3_Off
231 *   @brief  Switch RLS 3 Off
232 */
233 #define Ant_RLS_3_Off   gioSetBit(gioPORTA, 5, 0)
234 /** @def Ant_RLS_4_On
235 *   @brief  Switch RLS 4 On
236 */
237 #define Ant_RLS_4_On    gioSetBit(gioPORTA, 2, 1)
238 /** @def Ant_RLS_4_Off
239 *   @brief  Switch RLS 4 Off
240 */
241 #define Ant_RLS_4_Off   gioSetBit(gioPORTA, 2, 0)
242
243 /** @def Antenna_DTCT_1_2_Reg
244 *   @brief  Gio register for antenna detection 1 and 2
245 */
```

```
246 #define Antenna_DTCT_1_2_Reg canREG2
247 /** @def Antenna_DTCT_3_4_Reg
248 *   @brief  Gio register for antenna detection 3 and 4
249 */
250 #define Antenna_DTCT_3_4_Reg canREG3
251
252 /*! This struct holds values of the switches. */
253 typedef struct antenna_status{
254     int Antenna_1; /*!< status of Antenna number 1*/
255     int Antenna_2; /*!< status of Antenna number 2*/
256     int Antenna_3; /*!< status of Antenna number 3*/
257     int Antenna_4; /*!< status of Antenna number 4*/
258 }Antenna;
259 /*================================== BIAS DAC ==========================================*/
260 /*===== Source datasheet and mNLP schematics  ==========================================*/
261
262 /** @def Gain_Enable_Off
263 *   @brief GIO Gain_Enable_Off off for channel 1 to 4
264 */
265 #define Gain_Enable_Off gioSetBit(hetPORT1, PIN_HET_15, 0) // for channel 1 to 4
266 /** @def Gain_Enable_On
267 *   @brief GIO Gain_Enable_On on for channel 1 to 4
268 */
269 #define Gain_Enable_On gioSetBit(hetPORT1, PIN_HET_15, 1)  // for channel 1 to 4
270 /** @def Gain_SEL_G10
271 *   @brief Set Gain to 10 for channel 1 to 4
272 */
273 #define Gain_SEL_G10 gioSetBit(hetPORT1, PIN_HET_19, 0)    // for channel 1 to 4
274 /** @def Gain_SEL_G20
275 *   @brief Set Gain to 20 for channel 1 to 4
276 */
277 #define Gain_SEL_G20 gioSetBit(hetPORT1, PIN_HET_19, 1)    // for channel 1 to 4
278
279
280 /** @def BIAS_ADC_Channels_cnt
281 *   @brief total number of bias channels that is read from
282 *
283 */
284 #define BIAS_ADC_Channels_cnt       4
285 /*! This enum holds values for the order of the adc channels, the ADC multiplexes between
286  * channels starting from the lowest channel AD1IN[06] number to the highest. */
287 enum BIAS_adc_group2_channels{
288     DAC_BCH3_ID, /*!< corresponded to channel AD1IN[06]*/
289     DAC_BCH2_ID, /*!< corresponded to channel AD1IN[08]*/
290     DAC_BCH1_ID, /*!< corresponded to channel AD1IN[14]*/
291     DAC_BCH4_ID, /*!< corresponded to channel AD1IN[22]*/
292 };
293
294 /*! This struct holds global values that conversion from from bias voltage values.*/
295 typedef struct Bias_Measured_V{
296     int DAC_BIAS_CH1;/*!< Bias voltage read back CH1 to channel AD1IN[14]*/
297     int DAC_BIAS_CH2;/*!< Bias voltage read back CH2 to channel AD1IN[08]*/
298     int DAC_BIAS_CH3;/*!< Bias voltage read back CH3 to channel AD1IN[06]*/
299     int DAC_BIAS_CH4;/*!< Bias voltage read back CH3 to channel [22]*/
300 }BIAS_V;
301 extern struct Bias_Measured_V BIAS_MON_V;
302
303 /*! This struct holds values of Gain_Select and whether its on or off.*/
304 typedef struct Gain_sel_om{
305     int Gain_on_off;/*!< Gain_on = 1 and Gain_off=0*/
306     int Gain_10;/*!<Gain_10  is selected when it s equal to 1*/
307     int Gain_20;/*!< Gain _20 is selected when it s equal to 1*/
308 }gain_sel;
309 /*========== ltc3887  Digital switcher regulator ========================*/
310 /*========== Source LTC3887 datasheet and LTpowerplay    ================*/
311
312 /** @def LTC3887_Page
313 *   @brief  Channel (page) presently selected for any
314 *            paged command.
315 *            - Paged : N
316 *            - Typed : R/W byte
317 *            - Data format : REG
318 *            - Default value : 0x00
319 */
320 #define LTC3887_Page 0x00
321 /** @def LTC3887_OPERATION
322 *   @brief   Operating mode control. On/off, margin high and margin
323 *               low.
324 *                - type R/W byte
325 *                - Paged N
326 *                - Data format is REG
327 *                - NVM Y
328 *                - Default value. 0x40
```

```
329 */
330 #define LTC3887_OPERATION 0x01
331
332
333 /** @def LTC3887_ON_OFF_CONFIG
334 *    @brief     RUN pin and PMBus bus on/off command configuration.
335 *                 - type R/W byte
336 *                 - Paged Y
337 *                 - Data format is REG
338 *                 - NVM Y
339 *                 - Default value. 0x40
340 */
341 #define LTC3887_ON_OFF_CONFIG 0x02
342
343 /** @def LTC3887_CLEAR_FAULTS
344 *    @brief     Clear any fault bits that have been set.
345 *                 - type R/W byte
346 *                 - Paged Y
347 *                 - Data format is REG
348 *                 - NVM Y
349 *                 - Default value. 0x40
350 */
351
352 #define LTC3887_CLEAR_FAULTS         0x03
353
354 /** @def LTC3887_PAGE_PLUS_WRITE
355 *    @brief     Write a command directly to a specified page
356 *                 - type W block
357 *                 - Paged N
358 *                 - Data format is -
359 *                 - NVM -
360 *                 - Default value. -
361 */
362
363
364 #define LTC3887_PAGE_PLUS_WRITE      0x05
365
366 /** @def LTC3887_PAGE_PLUS_READ
367 *    @brief     Read a command directly from a specified page
368 *                 - type R/W byte
369 *                 - Paged Y
370 *                 - Data format is REG
371 *                 - NVM Y
372 *                 - Default value. 0x40
373 */
374
375
376 #define LTC3887_PAGE_PLUS_READ       0x06
377 /** @def LTC3887_WRITE_PROTECT
378 *    @brief     Level of protection provided by the device
379 *                  against accidental changes.
380 *                 - type R/W block
381 *                 - Paged N
382 *                 - Data format is
383 *                 - NVM -
384 *                 - Default value -
385 */
386
387
388 #define LTC3887_WRITE_PROTECT        0x10
389 /** @def LTC3887_STORE_USER_ALL
390 *    @brief     Store user operating memory to EEPROM.
391 *                 - type R/W block
392 *                 - Paged N
393 *                 - Data format -
394 *                 - NVM -
395 *                 - Default value -
396 */
397
398
399 #define LTC3887_STORE_USER_ALL       0x15
400 /** @def LTC3887_RESTORE_USER_ALL
401 *    @brief     Restore user operating memory from EEPROM.
402 *                 - type R/W byte
403 *                 - Paged N
404 *                 - Data format is REG
405 *                 - NVM Y
406 *                 - Default value. 0x40
407 */
408
409 #define LTC3887_RESTORE_USER_ALL     0x16
410 /** @def LTC3887_CAPABILITY
411 *    @brief     Summary of PMBus optional communication
```

```
412 *                - type R/W byte
413 *                 - Paged N
414 *                 - Data format is REG
415 *                 - NVM Y
416 *                 - Default value. 0x40
417 */
418 #define LTC3887_CAPABILITY 0x19
419 /** @def LTC3887_SMBALERT_MASK
420 *   @brief    Mask ALERT activity.
421 *                - type R/W byte
422 *                 - Paged Y
423 *                 - Data format is REG
424 *                 - NVM Y
425 *                 - Default value. 0x40
426 */
427
428 #define LTC3887_SMBALERT_MASK 0x1B
429 /** @def LTC3887_VOUT_MODE
430 *   @brief    Output voltage format and exponent (2^12).
431 *                - type R/W byte
432 *                 - Paged Y
433 *                 - Data format is REG
434 *                 - NVM Y
435 *                 - Default value. 0x40
436 */
437
438 #define LTC3887_VOUT_MODE   0x20
439 /** @def LTC3887_VOUT_COMMAND
440 *   @brief    Nominal output voltage set point.
441 *                - type R/W byte
442 *                 - Paged Y
443 *                 - Data format is REG
444 *                 - NVM Y
445 *                 - Default value. 0x40
446 */
447
448 #define LTC3887_VOUT_COMMAND 0x21
449 /** @def LTC3887_VOUT_MAX
450 *   @brief    Upper limit on the commanded output voltage including VOUT_MARGIN_HI.
451 *                - type R/W byte
452 *                 - Paged Y
453 *                 - Data format is REG
454 *                 - NVM Y
455 *                 - Default value. 0x40
456 */
457
458 #define LTC3887_VOUT_MAX  0x24
459 /** @def LTC3887_VOUT_MARGIN_HIGH
460 *   @brief    Margin high output voltage set point. Must be greater than VOUT_COMMAND.
461 *                - type R/W byte
462 *                 - Paged Y
463 *                 - Data format is REG
464 *                 - NVM Y
465 *                 - Default value. 0x40
466 */
467
468
469 #define LTC3887_VOUT_MARGIN_HIGH 0x25
470 /** @def LTC3887_VOUT_MARGIN_HIGH
471 *   @brief    Margin low output voltage set point. Must be less than VOUT_COMMAND.
472 *                - type R/W byte
473 *                 - Paged Y
474 *                 - Data format is REG
475 *                 - NVM Y
476 *                 - Default value. 0x40
477 */
478
479
480 #define LTC3887_VOUT_MARGIN_LOW 0x26
481 /** @def LTC3887_VOUT_TRANSITION_RATE
482 *   @brief    Rate the output changes when VOUT commanded to a new value.
483 *                - type R/W byte
484 *                 - Paged Y
485 *                 - Data format is REG
486 *                 - NVM Y
487 *                 - Default value. 0x40
488 */
489
490 #define LTC3887_VOUT_TRANSITION_RATE 0X27
491 /** @def LTC3887_FREQUENCY_SWITCH
492 *   @brief    Switching frequency of the controller.
493 *                - type R/W byte
494 *                 - Paged N
```

```
495 *                 - Data format is REG
496 *                 - NVM Y
497 *                 - Default value. 0x40
498 */
499
500
501 #define LTC3887_FREQUENCY_SWITCH 0x33
502
503 /** @def LTC3887_VIN_ON
504 *    @brief    Input voltage at which the unit should start power conversion.
505 *                - type R/W byte
506 *                 - Paged N
507 *                 - Data format is REG
508 *                 - NVM Y
509 *                 - Default value. 0x40
510 */
511
512
513 #define LTC3887_VIN_ON 0x35
514 /** @def LTC3887_VIN_OFF
515 *    @brief    Input voltage at which the unit should stop power conversion.
516 *                - type R/W byte
517 *                 - Paged N
518 *                 - Data format is REG
519 *                 - NVM Y
520 *                 - Default value. 0x40
521 */
522
523
524 #define LTC3887_VIN_OFF 0x36
525
526 /** @def LTC3887_IOUT_CAL_GAIN
527 *    @brief    The ratio of the voltage at the current sense pins to the sensed current. For
      devices using a fixed
528 *                current sense resistor, it is the resistance value in mohm.
529 *                - type R/W byte
530 *                 - Paged Y
531 *                 - Data format is REG
532 *                 - NVM Y
533 *                 - Default value. 0x40
534 */
535
536 #define LTC3887_IOUT_CAL_GAIN 0x38
537 /** @def LTC3887_VOUT_OV_FAULT_LIMIT
538 *    @brief    Output overvoltage fault limit
539 *                - type R/W byte
540 *                 - Paged Y
541 *                 - Data format is REG
542 *                 - NVM Y
543 *                 - Default value. 0x40
544 */
545
546 #define LTC3887_VOUT_OV_FAULT_LIMIT 0x40
547 /** @def LTC3887_VOUT_OV_FAULT_RESPONSE
548 *    @brief    Action to be taken by the device when an output overvoltage fault is detected.
549 *                - type R/W byte
550 *                 - Paged Y
551 *                 - Data format is REG
552 *                 - NVM Y
553 *                 - Default value. 0x40
554 */
555
556 #define LTC3887_VOUT_OV_FAULT_RESPONSE 0x41
557 /** @def LTC3887_VOUT_OV_WARN_LIMIT
558 *    @brief    Output over voltage warning limit.
559 *                - type R/W byte
560 *                 - Paged Y
561 *                 - Data format is REG
562 *                 - NVM Y
563 *                 - Default value. 0x40
564 */
565
566
567 #define LTC3887_VOUT_OV_WARN_LIMIT 0x42
568 /** @def LTC3887_VOUT_UV_WARN_LIMIT
569 *    @brief    Output under voltage warning limit.
570 *                - type R/W byte
571 *                 - Paged Y
572 *                 - Data format is REG
573 *                 - NVM Y
574 *                 - Default value. 0x40
575 */
576
```

```
577 #define LTC3887_VOUT_UV_WARN_LIMIT 0x43
578 /** @def LTC3887_VOUT_UV_FAULT_LIMIT
579 *   @brief     Output undervoltage fault limit.
580 *               - type is R/W byte
581 *                 - Paged is Y
582 *                 - Data format is L16
583 *                 - units is V
584 *                 - NVM is Y
585 *                 - Default value. 0.9 0x0E66
586 */
587 #define LTC3887_VOUT_UV_FAULT_LIMIT          0x44
588 /** @def LTC3887_VOUT_UV_FAULT_RESPONSE
589 *   @brief     Action to be taken by the device when an output undervoltage fault is detected
      .
590 *               - type is R/W byte
591 *                 - Paged is Y
592 *                 - Data format is REG
593 *                 - units is none
594 *                 - NVM is Y
595 *                 - Default value 0xB8
596 */
597 #define LTC3887_VOUT_UV_FAULT_RESPONSE       0x45
598 /** @def LTC3887_IOUT_OC_FAULT_LIMIT
599 *   @brief     Output overcurrent fault limit.
600 *               - type is R/W byte
601 *                 - Paged is Y
602 *                 - Data format is L11
603 *                 - units is A
604 *                 - NVM is Y
605 *                 - Default value 29.75 0xDBB8
606 */
607 #define LTC3887_IOUT_OC_FAULT_LIMIT          0x46
608 /** @def LTC3887_IOUT_OC_FAULT_RESPONSE
609 *   @brief     Action to be taken by the device when an output over current fault is detected
      .
610 *               - type is R/W byte
611 *                 - Paged is Y
612 *                 - Data format is REG
613 *                 - units is none
614 *                 - NVM is Y
615 *                 - Default value 0x00
616 */
617 #define LTC3887_IOUT_OC_FAULT_RESPONSE       0x47
618 /** @def LTC3887_IOUT_OC_WARN_LIMIT
619 *   @brief    Output overcurrent warning limit
620 *                 - type is R/W byte
621 *                 - Paged is Y
622 *                 - Data format is L11
623 *                 - units is A
624 *                 - NVM is Y
625 *                 - Default value 20.0 0xDA80
626 */
627 #define LTC3887_IOUT_OC_WARN_LIMIT           0x4A
628 /** @def LTC3887_OT_FAULT_LIMIT
629 *   @brief    External overtemperature fault limit.
630 *                 - type is R/W byte
631 *                 - Paged is Y
632 *                 - Data format is L11
633 *                 - units is C
634 *                 - NVM is Y
635 *                 - Default value 100.0 0xEB20
636 */
637 #define LTC3887_OT_FAULT_LIMIT               0x4F
638 /** @def LTC3887_OT_FAULT_RESPONSE
639 *   @brief    Action to be taken by the device when an external overtemperature fault is
     detected,
640 *               - type is R/W byte
641 *                 - Paged is Y
642 *                 - Data format is REG
643 *                 - units is NONE
644 *                 - NVM is Y
645 *                 - Default value 0xB8
646 */
647 #define LTC3887_OT_FAULT_RESPONSE            0x50
648 /** @def LTC3887_OT_WARN_LIMIT
649 *   @brief     External overtemperature warning limit
650 *                 - type is R/W byte
651 *                 - Paged is Y
652 *                 - Data format is L11
653 *                 - units is C
654 *                 - NVM is Y
655 *                 - Default value 85.0 0xEAA8
656 */
```

```
657 #define LTC3887_OT_WARN_LIMIT                0x51
658 /** @def LTC3887_UT_FAULT_LIMIT
659 *    @brief      External under temperature fault limit.
660 *                 - type is R/W byte
661 *                   - Paged is Y
662 *                   - Data format is L11
663 *                   - units is C
664 *                   - NVM is Y
665 *                   - Default value 40.0 0xE580
666 */
667 #define LTC3887_UT_FAULT_LIMIT               0x53
668 /** @def LTC3887_UT_FAULT_RESPONSE
669 *    @brief      Action to be taken by the device when an external undertemperature fault is
        detected.
670 *                 - type is R/W byte
671 *                   - Paged is Y
672 *                   - Data format is REG
673 *                   - units is -
674 *                   - NVM is Y
675 *                   - Default value  0xB8
676 */
677 #define LTC3887_UT_FAULT_RESPONSE            0x54
678 /** @def LTC3887_VIN_OV_FAULT_LIMIT
679 *    @brief      Input supply overvoltage fault limit.
680 *                 - type is R/W byte
681 *                   - Paged is N
682 *                   - Data format is L11
683 *                   - units is V
684 *                   - NVM is Y
685 *                   - Default value  15.5 0xD3E0
686 */
687 #define LTC3887_VIN_OV_FAULT_LIMIT           0x55
688 /** @def LTC3887_VIN_OV_FAULT_RESPONSE
689 *    @brief     Action to be taken by the device when an input over voltage fault is detected.
690 *                   - type is R/W byte
691 *                   - Paged is Y
692 *                   - Data format is Reg
693 *                   - units is -
694 *                   - NVM is Y
695 *                   - Default value  0x80
696 */
697 #define LTC3887_VIN_OV_FAULT_RESPONSE        0x56
698 /** @def LTC3887_VIN_UV_WARN_LIMIT
699 *    @brief      Input supply undervoltage warning limit.
700 *                   - type is R/W byte
701 *                   - Paged is N
702 *                   - Data format is L11
703 *                   - units is V
704 *                   - NVM is Y
705 *                   - Default value  6.3 0xCB26
706 */
707 #define LTC3887_VIN_UV_WARN_LIMIT            0x58
708 /** @def LTC3887_IIN_OC_WARN_LIMIT
709 *    @brief      Input supply overcurrent warning limit.
710 *                   - type is R/W byte
711 *                   - Paged is N
712 *                   - Data format is L11
713 *                   - units is A
714 *                   - NVM is Y
715 *                   - Default value   10.0 0xD280
716 */
717 #define LTC3887_IIN_OC_WARN_LIMIT            0x5D
718 /** @def LTC3887_TON_DELAY
719 *    @brief      Time from RUN and/or Operation on to output rail turn-on.
720 *                   - type is R/W byte
721 *                   - Paged is Y
722 *                   - Data format is L11
723 *                   - units is ms
724 *                   - NVM is Y
725 *                   - Default value   0.0 0x8000
726 */
727 #define LTC3887_TON_DELAY                    0x60
728 /** @def LTC3887_TON_RISE
729 *    @brief      Time from when the output starts to rise until the output voltage reaches the
        VOUT
730 *                commanded value.
731 *                   - type is R/W byte
732 *                   - Paged is Y
733 *                   - Data format is L11
734 *                   - units is ms
735 *                   - NVM is Y
736 *                   - Default value    8.0 0xD200
737 */
```

```
738 #define LTC3887_TON_RISE                        0x61
739 /** @def LTC3887_TON_MAX_FAULT_LIMIT
740 *    @brief     Maximum time from the start of TON_RISE for VOUT to cross the
        VOUT_UV_FAULT_LIMIT.
741 *                 - type is R/W byte
742 *                 - Paged is Y
743 *                 - Data format is L11
744 *                 - units is ms
745 *                 - NVM is Y
746 *                 - Default value     10.00 0xD280
747 */
748
749 #define LTC3887_TON_MAX_FAULT_LIMIT             0x62
750 /** @def LTC3887_TON_MAX_FAULT_RESPONSE
751 *    @brief     Action to be taken by the device when a TON_MAX_FAULT event is detected.
752
753 *                 - type is R/W byte
754 *                 - Paged is Y
755 *                 - Data format is REG
756 *                 - units is ms
757 *                 - NVM is Y
758 *                 - Default value     0xB8
759 */
760 #define LTC3887_TON_MAX_FAULT_RESPONSE          0x63
761 /** @def LTC3887_TOFF_DELAY
762 *    @brief      Time from RUN and/or Operation off to the start of TOFF_FALL ramp.
763 *                 - type is R/W byte
764 *                 - Paged is Y
765 *                 - Data format is L11
766 *                 - units is ms
767 *                 - NVM is Y
768 *                 - Default value     0x80000
769 */
770
771 #define LTC3887_TOFF_DELAY                      0x64
772 /** @def LTC3887_TOFF_FALL
773 *    @brief      Time from when the output starts to fall until the output reaches zero volts.
774 *                 - type is R/W byte
775 *                 - Paged is Y
776 *                 - Data format is L11
777 *                 - units is ms
778 *                 - NVM is Y
779 *                 - Default value     8.00 0xD200
780 */
781 #define LTC3887_TOFF_FALL                       0x65
782 /** @def LTC3887_TOFF_MAX_WARN_LIMIT
783 *    @brief      Maximum allowed time, after TOFF_FALL completed, for the unit to decay below
        12.5%
784 *                 - type is R/W byte
785 *                 - Paged is Y
786 *                 - Data format is L11
787 *                 - units is ms
788 *                 - NVM is Y
789 *                 - Default value     150 0xF258
790 */
791 #define LTC3887_TOFF_MAX_WARN_LIMIT             0x66
792 /** @def LTC3887_STATUS_BYTE
793 *    @brief      One byte summary of the units fault condition
794 *                 - type is R/W byte
795 *                 - Paged is Y
796 *                 - Data format is REG
797 *                 - units is -
798 *                 - NVM is -
799 *                 - Default value  -
800 */
801 #define LTC3887_STATUS_BYTE                     0x78
802 /** @def LTC3887_STATUS_WORD
803 *    @brief      Two byte summary of the units fault condition
804 *                 - type is R/W byte
805 *                 - Paged is Y
806 *                 - Data format is REG
807 *                 - units is -
808 *                 - NVM is -
809 *                 - Default value  -
810 */
811
812 #define LTC3887_STATUS_WORD                     0x79
813 /** @def LTC3887_STATUS_VOUT
814 *    @brief       Output voltage fault and warning status.
815 *                 - type is R/W byte
816 *                 - Paged is Y
817 *                 - Data format is REG
818 *                 - units is -
```

```
819 *                  - NVM is -
820 *                  - Default value  -
821 */
822 #define LTC3887_STATUS_VOUT                   0x7A
823 /** @def LTC3887_STATUS_IOUT
824 *    @brief       Output voltage fault and warning status.
825 *                  - type is R/W byte
826 *                  - Paged is Y
827 *                  - Data format is REG
828 *                  - units is -
829 *                  - NVM is -
830 *                  - Default value  -
831 */
832 #define LTC3887_STATUS_IOUT                   0x7B
833 /** @def LTC3887_STATUS_INPUT
834 *    @brief       Input supply fault and warning status
835 *                  - type is R/W byte
836 *                  - Paged is N
837 *                  - Data format is REG
838 *                  - units is -
839 *                  - NVM is -
840 *                  - Default value  -
841 */
842 #define LTC3887_STATUS_INPUT                  0x7C
843 /** @def LTC3887_STATUS_TEMPERATURE
844 *    @brief       Output voltage fault and warning status. For READ_TEMERATURE_1.
845 *                  - type is R/W byte
846 *                  - Paged is Y
847 *                  - Data format is REG
848 *                  - units is -
849 *                  - NVM is -
850 *                  - Default value  -
851 */
852 #define LTC3887_STATUS_TEMPERATURE            0x7D
853 /** @def LTC3887_STATUS_CML
854 *    @brief       Communication and memory fault and warning status.
855 *                  - type is R/W byte
856 *                  - Paged is N
857 *                  - Data format is REG
858 *                  - units is -
859 *                  - NVM is -
860 *                  - Default value  -
861 */
862 #define LTC3887_STATUS_CML                    0x7E
863 /** @def LTC3887_STATUS_MFR_SPECIFIC
864 *    @brief       Manufacturer specific fault and state information.
865 *                  - type is R/W byte
866 *                  - Paged is Y
867 *                  - Data format is REG
868 *                  - units is -
869 *                  - NVM is -
870 *                  - Default value  -
871 */
872 #define LTC3887_STATUS_MFR_SPECIFIC           0x80
873 /** @def LTC3887_READ_VIN
874 *    @brief       Measured input supply voltage.
875 *                  - type is R byte
876 *                  - Paged is N
877 *                  - Data format is L11
878 *                  - units is V
879 *                  - NVM is -
880 *                  - Default value  -
881 */
882 #define LTC3887_READ_VIN                      0x88
883 /** @def LTC3887_READ_VIN
884 *    @brief       Measured input supply voltage.
885 *                  - type is R byte
886 *                  - Paged is N
887 *                  - Data format is L11
888 *                  - units is A
889 *                  - NVM is -
890 *                  - Default value  -
891 */
892 #define LTC3887_READ_IIN                      0x89
893 /** @def LTC3887_READ_VOUT
894 *    @brief       Measured input supply voltage.
895 *                  - type is R byte
896 *                  - Paged is Y
897 *                  - Data format is L16
898 *                  - units is V
899 *                  - NVM is -
900 *                  - Default value  -
901 */
```

```
902 #define LTC3887_READ_VOUT                       0x8B
903 /** @def LTC3887_READ_IOUT
904 *   @brief      Measured output current
905 *               - type is R byte
906 *               - Paged is Y
907 *               - Data format is L16
908 *               - units is V
909 *               - NVM is -
910 *               - Default value  -
911 */
912 #define LTC3887_READ_IOUT                       0x8C
913 /** @def LTC3887_READ_TEMPERATURE_1
914 *   @brief      External temperature sensor. This is the value
915 *               used for all temperature related processing,
916 *               including IOUT_CAL_GAIN.
917 *               - type is R byte
918 *               - Paged is Y
919 *               - Data format is L11
920 *               - units is C
921 *               - NVM is -
922 *               - Default value  -
923 */
924 #define LTC3887_READ_TEMPERATURE_1              0x8D
925 /** @def LTC3887_READ_TEMPERATURE_2
926 *   @brief      Internal die temperature. Does not affect any other registers.
927 *               - type is R byte
928 *               - Paged is N
929 *               - Data format is L11
930 *               - units is C
931 *               - NVM is -
932 *               - Default value  -
933 */
934 #define LTC3887_READ_TEMPERATURE_2              0x8E
935 /** @def LTC3887_READ_DUTY_CYCLE
936 *   @brief      Duty cycle of the top gate control signal.
937 *               - type is R byte
938 *               - Paged is Y
939 *               - Data format is L11
940 *               - units is %
941 *               - NVM is -
942 *               - Default value  -
943 */
944 #define LTC3887_READ_DUTY_CYCLE                 0x94
945 /** @def LTC3887_READ_FREQUENCY
946 *   @brief       Measured PWM switching frequency.
947 *               - type is R byte
948 *               - Paged is Y
949 *               - Data format is L11
950 *               - units is kHz
951 *               - NVM is -
952 *               - Default value  -
953 */
954 #define LTC3887_READ_FREQUENCY                  0x95
955 /** @def LTC3887_READ_POUT
956 *   @brief        Measured output power
957 *               - type is R byte
958 *               - Paged is Y
959 *               - Data format is L11
960 *               - units is W
961 *               - NVM is -
962 *               - Default value  -
963 */
964 #define LTC3887_READ_POUT                       0x96
965 /** @def LTC3887_PMBUS_REVISION
966 *   @brief      PMBus revision supported by this device. Current revision is 1.2.
967 *               - type is R byte
968 *               - Paged is N
969 *               - Data format is REG
970 *               - units is W
971 *               - NVM is -
972 *               - Default value  0x22
973 */
974 #define LTC3887_PMBUS_REVISION                  0x98
975 /** @def LTC3887_MFR_ID
976 *   @brief      The manufacturer ID of the LTC3887 in ASCII.
977 *               - type is R String
978 *               - Paged is N
979 *               - Data format is ASC
980 *               - units is -
981 *               - NVM is -
982 *               - Default value  LTC
983 */
984 #define LTC3887_MFR_ID                          0x99
```

```
985 /** @def LTC3887_MFR_ID
986 *    @brief      The manufacturer ID of the LTC3887 in ASCII.
987 *                - type is R String
988 *                - Paged is N
989 *                - Data format is ASC
990 *                - units is -
991 *                - NVM is -
992 *                - Default value  LTC3887
993 */
994 #define LTC3887_MFR_MODEL                       0x9A
995 /** @def LTC3887_MFR_SERIAL
996 *    @brief      Serial number of this specific unit.
997 *                - type is R Block
998 *                - Paged is N
999 *                - Data format is CF
1000 *                - units is -
1001 *                - NVM is -
1002 *                - Default value  -
1003 */
1004 #define LTC3887_MFR_SERIAL                      0x9E
1005 /** @def LTC3887_MFR_VOUT_MAX
1006 *    @brief      Maximum allowed output voltage including VOUT_OV_FAULT_LIMIT..
1007 *                - type is R word
1008 *                - Paged is Y
1009 *                - Data format is L16
1010 *                - units is V
1011 *                - NVM is -
1012 *                - Default value  5.7 0x5B34
1013 */
1014 #define LTC3887_MFR_VOUT_MAX                    0xA5
1015 /** @def LTC3887_USER_DATA_00
1016 *    @brief      OEM RESERVED. Typically used for part serialization.
1017 *                - type is R/W Word
1018 *                - Paged is N
1019 *                - Data format is Reg
1020 *                - units is -
1021 *                - NVM is Y
1022 *                - Default value  -
1023 */
1024 #define LTC3887_USER_DATA_00                    0xB0
1025 /** @def LTC3887_USER_DATA_01
1026 *    @brief      Manufacturer reserved for LTpowerPlay
1027 *                - type is R/W Word
1028 *                - Paged is y
1029 *                - Data format is Reg
1030 *                - units is -
1031 *                - NVM is Y
1032 *                - Default value  -
1033 */
1034 #define LTC3887_USER_DATA_01                    0xB1
1035 /** @def LTC3887_USER_DATA_02
1036 *    @brief      OEM RESERVED. Typically used for part serialization
1037 *                - type is R/W Word
1038 *                - Paged is y
1039 *                - Data format is Reg
1040 *                - units is -
1041 *                - NVM is Y
1042 *                - Default value  -
1043 */
1044 #define LTC3887_USER_DATA_02                    0xB2
1045 /** @def LTC3887_USER_DATA_03
1046 *    @brief      A NVM word available for the user.
1047 *                - type is R/W Word
1048 *                - Paged is N
1049 *                - Data format is Reg
1050 *                - units is -
1051 *                - NVM is Y
1052 *                - Default value  0x0000
1053 */
1054 #define LTC3887_USER_DATA_03                    0xB3
1055 /** @def LTC3887_USER_DATA_04
1056 *    @brief      A NVM word available for the user.
1057 *                - type is R/W Word
1058 *                - Paged is N
1059 *                - Data format is Reg
1060 *                - units is -
1061 *                - NVM is Y
1062 *                - Default value  0x0000
1063 */
1064 #define LTC3887_USER_DATA_04                    0xB4
1065 /** @def LTC3887_MFR_EE_UNLOCK
1066 *    @brief      Unlock user EEPROM for access by MFR_EE_ ERASE and MFR_EE_DATA commands.
1067 *                - type is R/W Word
```

```
1068 *                - Paged is N
1069 *                - Data format is Reg
1070 *                - units is -
1071 *                - NVM is Y
1072 *                - Default value  -
1073 */
1074 #define LTC3887_MFR_EE_UNLOCK                0xBD
1075 /** @def LTC3887_MFR_EE_ERASE
1076 *   @brief  Initialize user EEPROM for bulk programming by MFR_EE_DATA.
1077 *                - type is R/W Word
1078 *                - Paged is N
1079 *                - Data format is Reg
1080 *                - units is -
1081 *                - NVM is Y
1082 *                - Default value  -
1083 */
1084 #define LTC3887_MFR_EE_ERASE                 0xBE
1085 /** @def LTC3887_MFR_EE_DATA
1086 *   @brief  Data transferred to and from EEPROM using sequential PMBus word reads or writes.
     Supports bulk programming.
1087 *                - type is R/W Word
1088 *                - Paged is N
1089 *                - Data format is Reg
1090 *                - units is -
1091 *                - NVM is -
1092 *                - Default value  -
1093 */
1094 #define LTC3887_MFR_EE_DATA                  0xBF
1095 /** @def LTC3887_MFR_CHAN_CONFIG_LTC3887
1096 *   @brief  Configuration bits that are channel specific.
1097 *                - type is R/W byte
1098 *                - Paged is Y
1099 *                - Data format is Reg
1100 *                - units is -
1101 *                - NVM is Y
1102 *                - Default value  0x1D
1103 */
1104 #define LTC3887_MFR_CHAN_CONFIG_LTC3887      0xD0
1105 /** @def LTC3887_MFR_CONFIG_ALL_LTC3887
1106 *   @brief  Configuration bits that are common to all pages.
1107 *                - type is R/W byte
1108 *                - Paged is N
1109 *                - Data format is Reg
1110 *                - units is -
1111 *                - NVM is Y
1112 *                - Default value  0x21
1113 */
1114
1115 #define LTC3887_MFR_CONFIG_ALL_LTC3887       0xD1
1116 /** @def LTC3887_MFR_GPIO_PROPAGATE_LTC3887
1117 *   @brief  Configuration bits that are common to all pages.
1118 *                - type is R/W byte
1119 *                - Paged is Y
1120 *                - Data format is Reg
1121 *                - units is -
1122 *                - NVM is Y
1123 *                - Default value  0x21
1124 */
1125 #define LTC3887_MFR_GPIO_PROPAGATE_LTC3887   0xD2
1126 /** @def LTC3887_MFR_PWM_MODE_LTC3887
1127 *   @brief  Configuration that determines which faults are propagated to the GPIO pins.
1128 *                - type is R/W byte
1129 *                - Paged is Y
1130 *                - Data format is Reg
1131 *                - units is -
1132 *                - NVM is Y
1133 *                - Default value  0x6993
1134 */
1135 #define LTC3887_MFR_PWM_MODE_LTC3887         0xD4
1136 /** @def LTC3887_MFR_GPIO_RESPONSE
1137 *   @brief  Action to be taken by the device when the GPIO pin is externally asserted low
1138 *                - type is R/W byte
1139 *                - Paged is Y
1140 *                - Data format is Reg
1141 *                - units is -
1142 *                - NVM is Y
1143 *                - Default value 0xC0
1144 */
1145 #define LTC3887_MFR_GPIO_RESPONSE            0xD5
1146 /** @def LTC3887_MFR_OT_FAULT_RESPONSE
1147 *   @brief  Action to be taken by the device when an internal over Temperature fault is
     detected.
1148 *                - type is R/W byte
```

```
1149 *                  - Paged is Y
1150 *                  - Data format is Reg
1151 *                  - units is -
1152 *                  - NVM is Y
1153 *                  - Default value 0xC0
1154 */
1155 #define LTC3887_MFR_OT_FAULT_RESPONSE          0xD6
1156 /** @def LTC3887_MFR_OT_FAULT_RESPONSE
1157 *    @brief  Report the maximum measured value of READ_IOUT since last MFR_CLEAR_PEAKS
1158 *                  - type is R Word
1159 *                  - Paged is Y
1160 *                  - Data format is L11
1161 *                  - units is A
1162 *                  - NVM is -
1163 *                  - Default value -
1164 */
1165 #define LTC3887_MFR_IOUT_PEAK                  0xD7
1166 /** @def LTC3887_MFR_ADC_CONTROL
1167 *    @brief  ADC telemetry parameter selected for repeated fast ADC read back
1168 *                  - type is R/W Byte
1169 *                  - Paged is N
1170 *                  - Data format is Reg
1171 *                  - units is -
1172 *                  - NVM is -
1173 *                  - Default value 0x00
1174 */
1175 #define LTC3887_MFR_ADC_CONTROL                0xD8
1176 /** @def LTC3887_MFR_ADC_TELEMETRY_STATUS
1177 *    @brief   ADC telemetry status indicating which parameter is most recently converted when
     the short round robin ADC loop is enabled
1178 *                  - type is R/W Byte
1179 *                  - Paged is N
1180 *                  - Data format is Reg
1181 *                  - units is -
1182 *                  - NVM is -
1183 *                  - Default value 0x00
1184 */
1185 #define LTC3887_MFR_ADC_TELEMETRY_STATUS       0xDA
1186 /** @def LTC3887_MFR_RETRY_DELAY
1187 *    @brief  Retry interval during FAULT retry mode
1188 *                  - type is R/W Word
1189 *                  - Paged is Y
1190 *                  - Data format is L11
1191 *                  - units is ms
1192 *                  - NVM is Y
1193 *                  - Default value 350 0xFABC
1194 */
1195 #define LTC3887_MFR_RETRY_DELAY                0xDB
1196 /** @def LTC3887_MFR_RETRY_DELAY
1197 *    @brief   Minimum time the RUN pin is held low by the LTC3887.
1198 *                  - type is R/W Word
1199 *                  - Paged is Y
1200 *                  - Data format is L11
1201 *                  - units is ms
1202 *                  - NVM is Y
1203 *                  - Default value 500 0xFBE8
1204 */
1205 #define LTC3887_MFR_RESTART_DELAY              0xDC
1206 /** @def LTC3887_MFR_VOUT_PEAK
1207 *    @brief Maximum measured value of READ_VOUT since last MFR_CLEAR_PEAKS.
1208 *                  - type is R Word
1209 *                  - Paged is Y
1210 *                  - Data format is L16
1211 *                  - units is V
1212 *                  - NVM is -
1213 *                  - Default value -
1214 */
1215 #define LTC3887_MFR_VOUT_PEAK                  0xDD
1216 /** @def LTC3887_MFR_VOUT_PEAK
1217 *    @brief  Maximum measured value of READ_VIN since last MFR_CLEAR_PEAKS.
1218 *                  - type is R Word
1219 *                  - Paged is N
1220 *                  - Data format is L11
1221 *                  - units is V
1222 *                  - NVM is -
1223 *                  - Default value -
1224 */
1225 #define LTC3887_MFR_VIN_PEAK                   0xDE
1226 /** @def LTC3887_MFR_TEMPERATURE_1_PEAK
1227 *    @brief  Maximum measured value of external Temperature (READ_TEMPERATURE_1) since last
     MFR_CLEAR_PEAKS.
1228 *                  - type is R Word
1229 *                  - Paged is Y
```

```
1230 *                - Data format is L11
1231 *                - units is C
1232 *                - NVM is -
1233 *                - Default value -
1234 */
1235 #define LTC3887_MFR_TEMPERATURE_1_PEAK        0xDF
1236 /** @def LTC3887_MFR_CLEAR_PEAKS
1237 *   @brief  Clears all peak values
1238 *                - type is Send Byte
1239 *                - Paged is N
1240 *                - Data format is -
1241 *                - units is -
1242 *                - NVM is -
1243 *                - Default value -
1244 */
1245 #define LTC3887_MFR_CLEAR_PEAKS               0xE3
1246 /** @def LTC3887_MFR_CLEAR_PEAKS
1247 *   @brief  Digital status of the I/O pads.
1248 *                - type is R Word
1249 *                - Paged is N
1250 *                - Data format is REG
1251 *                - units is -
1252 *                - NVM is -
1253 *                - Default value -
1254 */
1255 #define LTC3887_MFR_PADS                      0xE5
1256 /** @def LTC3887_MFR_ADDRESS
1257 *   @brief  Sets the 7-bit I2C address byte.
1258 *                - type is R/W Byte
1259 *                - Paged is N
1260 *                - Data format is REG
1261 *                - units is -
1262 *                - NVM is Y
1263 *                - Default value 0x4F
1264 */
1265 #define LTC3887_MFR_ADDRESS                   0xE6
1266 /** @def LTC3887_MFR_SPECIAL_ID
1267 *   @brief  Manufacturer code representing the LTC3887.
1268 *                - type is R Word
1269 *                - Paged is N
1270 *                - Data format is REG
1271 *                - units is -
1272 *                - NVM is -
1273 *                - Default value 0x470X
1274 */
1275 #define LTC3887_MFR_SPECIAL_ID                0xE7
1276 /** @def LTC3887_MFR_IIN_OFFSET
1277 *   @brief  Coefficient used to add to the input current to account for the IQ of the part.
1278 *                - type is R/W Word
1279 *                - Paged is Y
1280 *                - Data format is L11
1281 *                - units is A
1282 *                - NVM is Y
1283 *                - Default value  0.050 0X9333
1284 */
1285 #define LTC3887_MFR_IIN_OFFSET                0xE9
1286 /** @def LTC3887_MFR_FAULT_LOG_STORE
1287 *   @brief  Command a transfer of the fault log from RAM to EEPROM.
1288 *                - type is Send Byte
1289 *                - Paged is N
1290 *                - Data format is -
1291 *                - units is -
1292 *                - NVM is -
1293 *                - Default value  -
1294 */
1295 #define LTC3887_MFR_FAULT_LOG_STORE           0xEA
1296 /** @def LTC3887_MFR_FAULT_LOG_CLEAR
1297 *   @brief  Initialize the EEPROM block reserved for fault logging.
1298 *                - type is Send Byte
1299 *                - Paged is N
1300 *                - Data format is -
1301 *                - units is -
1302 *                - NVM is -
1303 *                - Default value  -
1304 */
1305 #define LTC3887_MFR_FAULT_LOG_CLEAR           0xEC
1306 /** @def LTC3887_MFR_READ_IIN
1307 *   @brief  Measured input current per channel
1308 *                - type is R Word
1309 *                - Paged is Y
1310 *                - Data format is L11
1311 *                - units is A
1312 *                - NVM is -
```

```
1313 *                - Default value  -
1314 */
1315 #define LTC3887_MFR_READ_IIN                    0xED
1316 /** @def LTC3887_MFR_FAULT_LOG
1317 *   @brief  Fault log data bytes. This sequentially retrieved data is used to assemble a
         complete fault log.
1318 *                - type is R Block
1319 *                - Paged is N
1320 *                - Data format is Reg
1321 *                - units is -
1322 *                - NVM is Y
1323 *                - Default value  -
1324 */
1325 #define LTC3887_MFR_FAULT_LOG                   0xEE
1326 /** @def LTC3887_MFR_COMMON
1327 *   @brief  Manufacturer status bits that are common across multiple LTC chips.
1328 *                - type is R Byte
1329 *                - Paged is N
1330 *                - Data format is Reg
1331 *                - units is -
1332 *                - NVM is Y
1333 *                - Default value  -
1334 */
1335 #define LTC3887_MFR_COMMON                      0xEF
1336 /** @def LTC3887_MFR_COMPARE_USER_ALL
1337 *   @brief  Compares current command contents with NVM.
1338 *                - type is Send Byte
1339 *                - Paged is N
1340 *                - Data format is Reg
1341 *                - units is -
1342 *                - NVM is Y
1343 *                - Default value  -
1344 */
1345 #define LTC3887_MFR_COMPARE_USER_ALL            0xF0
1346 /** @def LTC3887_MFR_TEMPERATURE_2_PEAK
1347 *   @brief  Compares current command contents with NVM.
1348 *                - type is R Word
1349 *                - Paged is N
1350 *                - Data format is L11
1351 *                - units is C
1352 *                - NVM is -
1353 *                - Default value  -
1354 */
1355 #define LTC3887_MFR_TEMPERATURE_2_PEAK          0xF4
1356 /** @def LTC3887_MFR_TEMPERATURE_2_PEAK
1357 *   @brief  Set numerous parameters for the DC/DC controller including phasing.
1358 *                - type is R/W Byte
1359 *                - Paged is N
1360 *                - Data format is Reg
1361 *                - units is -
1362 *                - NVM is Y
1363 *                - Default value  0x10
1364 */
1365 #define LTC3887_MFR_PWM_CONFIG_LTC3887          0xF5
1366 /** @def LTC3887_MFR_IOUT_CAL_GAIN_TC
1367 *   @brief  Temperature coefficient of the current sensing element.
1368 *                - type is R/W Word
1369 *                - Paged is Y
1370 *                - Data format is CF
1371 *                - units is -
1372 *                - NVM is Y
1373 *                - Default value  3900 0x0F3C
1374 */
1375 #define LTC3887_MFR_IOUT_CAL_GAIN_TC            0xF6
1376 /** @def LTC3887_MFR_TEMP_1_GAIN
1377 *   @brief  Sets the slope of the external temperature sensor.
1378 *                - type is R/W Word
1379 *                - Paged is Y
1380 *                - Data format is CF
1381 *                - units is -
1382 *                - NVM is Y
1383 *                - Default value  1.0 0x4000
1384 */
1385 #define LTC3887_MFR_TEMP_1_GAIN                 0xF8
1386 /** @def LTC3887_MFR_TEMP_1_OFFSET
1387 *   @brief  Sets the offset of the external temperature sensor with respect to 273.1 C
1388 *                - type is R/W Word
1389 *                - Paged is Y
1390 *                - Data format is L11
1391 *                - units is C
1392 *                - NVM is Y
1393 *                - Default value  0.0 0x8000
1394 */
```

```
1395 #define LTC3887_MFR_TEMP_1_OFFSET              0xF9
1396 /** @def LTC3887_MFR_RAIL_ADDRESS
1397 *    @brief  Common address for PolyPhase outputs to adjust common parameters.
1398 *                - type is R/W Byte
1399 *                - Paged is Y
1400 *                - Data format is Reg
1401 *                - units is -
1402 *                - NVM is Y
1403 *                - Default value 0x80
1404 */
1405 #define LTC3887_MFR_RAIL_ADDRESS               0xFA
1406 /** @def LTC3887_MFR_RESET
1407 *    @brief  Commanded reset without requiring a power down
1408 *                - type is Send Byte
1409 *                - Paged is N
1410 *                - Data format is Reg
1411 *                - units is -
1412 *                - NVM is -
1413 *                - Default value -
1414 */
1415 #define LTC3887_MFR_RESET                      0xFD
1416
1417
1418 #define One_Byte    1
1419 #define Two_Bytes   2
1420 #define LTC3887_GLOBAL_ADDRESS      (0xb4 >> 1)
1421 #define LTC3887_PAGED_ADDRESS       (0xb6 >> 1)
1422 #define LTC3887_PAGE 0x00
1423 enum {GLOBAL, PAGED, COMPLETE};
1424 typedef struct{
1425     unsigned int ltc_config_Address;
1426     const int ltc_Command;
1427     const int ltc_byte_Length;
1428     const int ltc_Lower_byte;
1429     const int ltc_upper_byte;
1430 }ltc_config;
1431
1432 typedef struct ltc_read
1433 {
1434     uint16 ltc_voltage;
1435     uint16 ltc_temperature;
1436     uint16 ltc_current;
1437 }ltc_read_values;
1438 extern struct ltc_read ltc_m_values;
1439 /*=========================================== AD7768 External ADC
      ==========================================*/
1440  /************************* sources is analog.com ad7768 drivers and ELAB-Halvor + Erelend***
       */
1441
1442 /** @def MCLK_FREQ
1443 *    @brief the clock frequency by ECLK
1444 */
1445 #define MCLK_FREQ system_get_sys_MCLK()
1446
1447 /** @def ad7768_Illegal_Command
1448 *    @brief the AD7768 SPI detects whether it received an illegal command.
1449 *    This illegal command issued when :
1450 *        - Illegal command is writen to read only register
1451 *        - Address register is none existing
1452 *        - Read from a register address that does not exist
1453 *
1454 */
1455 #define ad7768_Illegal_Command 0x0E00
1456 /** @def AD7768_channel_standby
1457 *    @brief This register is to configurate Bit[n] = CH_n
1458 *           - status: not used
1459 *           - defult value 0x0
1460 *           - Type:RW
1461 */
1462 #define AD7768_channel_standby 0x00
1463 /** @def AD7768_Channel_Mode_A
1464 *    @brief This register is to configurate bit[3]filter_type_A and Bit[2-0]DEC_rate_A
1465 *           - status: used to configure filter and dec_rate
1466 *           - defult value 0x0D
1467 *           - Type:RW
1468 */
1469 #define AD7768_Channel_Mode_A           0x01
1470 /** @def AD7768_Channel_Mode_B
1471 *    @brief This register is to configurate bit[3]filter_type_b and Bit[2-0]DEC_rate_B
1472 *           - status: not used
1473 *           - defult value 0x0D
1474 *           - Type:RW
1475 */
```

```
1476 #define AD7768_Channel_Mode_B            0x02
1477 /** @def AD7768_Channel_mode_select
1478 *    @brief This register is to configurate Bit[n] = CH_n_mode
1479 *           - status: not used
1480 *           - defult value 0x00
1481 *           - Type:RW
1482 */
1483 #define AD7768_Channel_mode_select       0x03
1484 /** @def AD7768_POWER_MODE
1485 *    @brief This register is to configurate Bit[7] = SLEEP_MODE, Bit[5-4] = Power_mode, Bit[3]
          = LVDS_enable, Bit[1-0]= MCLK_div
1486 *           - When setting power mode a mclk_div should be choosen accordinling
1487 *           - status: Used to configure MCLK_DIV and POWER_MODE
1488 *           - defult value 0x00
1489 *           - Type:RW
1490 */
1491 #define AD7768_POWER_MODE                0x04
1492 /** @def AD7768_General_configuration
1493 *    @brief This register is to configurate look at (REGISTER MAP DETAILS (SPI CONTROL) page
          72 AD7768/AD7768-4)
1494 *           - status: Not used
1495 *           - defult value 0x08
1496 *           - Type:RW
1497 */
1498 #define AD7768_General_configuration     0x05
1499 /** @def AD7768_General_configuration
1500 *    @brief This register is to configurate look at (REGISTER MAP DETAILS (SPI CONTROL) page
          72 AD7768/AD7768-4)
1501 *           - status: not used
1502 *           - defult value 0x80
1503 *           - Type:RW
1504 */
1505 #define AD7768_Data_control              0x06
1506 /** @def AD7768_Interface_configuration
1507 *    @brief This register is to configurate  CRC_SELECT(Bits[3-2]) and DCLK_DIV(BITs[1-0])
1508 *           - status: used to config DCLK_DIV
1509 *           - defult value 0x0
1510 *           - Type:RW
1511 */
1512 #define AD7768_Interface_configuration   0x07
1513 /** @def AD7768_BIST_control
1514 *    @brief This register is to configurate this has bit[0]=RAM_BIST_START
1515 *           - status: not used
1516 *           - defult value 0x00
1517 *           - Type:RW
1518 */
1519 #define AD7768_BIST_control              0x08
1520 /** @def AD7768_Device_status
1521 *    @brief This register is to configurate this has bit[0]=RAM_BIST_START
1522 *           - status: not used
1523 *           - defult value 0x00
1524 *           - Type:RW
1525 */
1526 #define AD7768_Device_status             0x09
1527 /** @def AD7768_Revision_ID
1528 *    @brief This register for read IC revision number
1529 *           - status: not used
1530 *           - defult value 0x06
1531 *           - Type:R
1532 */
1533 #define AD7768_Revision_ID               0x0A
1534 /** @def AD7768_GPIO_control
1535 *    @brief This register GPIO info
1536 *           - status: not used
1537 *           - defult value 0x00
1538 *           - Type:RW
1539 */
1540 #define AD7768_GPIO_control              0x0E
1541 /** @def AD7768_GPIO_write_data
1542 *    @brief This register Change output for the GPIO
1543 *           - status: not used
1544 *           - defult value 0x00
1545 *           - Type:RW
1546 */
1547 #define AD7768_GPIO_write_data           0x0F
1548 /** @def AD7768_GPIO_read_data
1549 *    @brief register reads the GPIO value
1550 *           - status: not used
1551 *           - defult value 0x00
1552 *           - Type:R
1553 */
1554 #define AD7768_GPIO_read_data            0x10
1555 /** @def AD7768_Precharge_Buffer_1
```

```
1556 *    @brief register : note used
1557 *            - status: not used
1558 *            - defult value 0xFF
1559 *            - Type:RW
1560 */
1561 #define AD7768_Precharge_Buffer_1       0x11
1562 /** @def AD7768_Precharge_Buffer_2
1563 *    @brief register : look at datasheet
1564 *            - status: not used
1565 *            - defult value 0xFF
1566 *            - Type:RW
1567 */
1568 #define AD7768_Precharge_Buffer_2       0x12
1569 /** @def AD7768_Positive_reference_precharge_buffer
1570 *    @brief register : look at datasheet
1571 *            - status: not used
1572 *            - defult value 0x00
1573 *            - Type:RW
1574 */
1575 #define AD7768_Positive_reference_precharge_buffer  0x13
1576 /** @def AD7768_Negative_reference_precharge_buffer
1577 *    @brief register : look at datasheet
1578 *            - status: not used
1579 *            - defult value 0x00
1580 *            - Type:RW
1581 */
1582 #define AD7768_Negative_reference_precharge_buffer  0x14
1583 /** @def AD7768_Channel_0_offset_MSB
1584 *    @brief register : look at datasheet
1585 *            - status: not used
1586 *            - defult value 0x00
1587 *            - Type:RW
1588 */
1589 #define AD7768_Channel_0_offset_MSB    0x1E
1590 /** @def AD7768_Channel_0_offset_MID
1591 *    @brief register : look at datasheet
1592 *            - status: not used
1593 *            - defult value 0x00
1594 *            - Type:RW
1595 */
1596 #define AD7768_Channel_0_offset_MID    0x1F
1597 /** @def AD7768_Channel_0_offset_LSB
1598 *    @brief register : look at datasheet
1599 *            - status: not used
1600 *            - defult value 0x00
1601 *            - Type:RW
1602 */
1603 #define AD7768_Channel_0_offset_LSB    0x20
1604 /** @def AD7768_Channel_1_offset_MSB
1605 *    @brief register : look at datasheet
1606 *            - status: not used
1607 *            - defult value 0x00
1608 *            - Type:RW
1609 */
1610 #define AD7768_Channel_1_offset_MSB    0x21
1611 /** @def AD7768_Channel_1_offset_MID
1612 *    @brief register : look at datasheet
1613 *            - status: not used
1614 *            - defult value 0x00
1615 *            - Type:RW
1616 */
1617 #define AD7768_Channel_1_offset_MID    0x22
1618 /** @def AD7768_Channel_1_offset_LSB
1619 *    @brief register : look at datasheet
1620 *            - status: not used
1621 *            - defult value 0x00
1622 *            - Type:RW
1623 */
1624 #define AD7768_Channel_1_offset_LSB    0x23
1625 /** @def AD7768_Channel_2_offset_MSB
1626 *    @brief register : look at datasheet
1627 *            - status: not used
1628 *            - defult value 0x00
1629 *            - Type:RW
1630 */
1631 #define AD7768_Channel_2_offset_MSB    0x24
1632 /** @def AD7768_Channel_2_offset_MID
1633 *    @brief register : look at datasheet
1634 *            - status: not used
1635 *            - defult value 0x00
1636 *            - Type:RW
1637 */
1638 #define AD7768_Channel_2_offset_MID    0x25
```

```
1639 /** @def AD7768_Channel_2_offset_LSB
1640 *    @brief register : look at datasheet
1641 *           - status: not used
1642 *           - defult value 0x00
1643 *           - Type:RW
1644 */
1645 #define AD7768_Channel_2_offset_LSB     0x26
1646 /** @def AD7768_Channel_3_offset_MSB
1647 *    @brief register : look at datasheet
1648 *           - status: not used
1649 *           - defult value 0x00
1650 *           - Type:RW
1651 */
1652 #define AD7768_Channel_3_offset_MSB     0x27
1653 /** @def AD7768_Channel_3_offset_MID
1654 *    @brief register : look at datasheet
1655 *           - status: not used
1656 *           - defult value 0x00
1657 *           - Type:RW
1658 */
1659 #define AD7768_Channel_3_offset_MID     0x28
1660 /** @def AD7768_Channel_3_offset_LSB
1661 *    @brief register : look at datasheet
1662 *           - status: not used
1663 *           - defult value 0x00
1664 *           - Type:RW
1665 */
1666 #define AD7768_Channel_3_offset_LSB     0x29
1667 /** @def AD7768_Channel_0_gain_MSB
1668 *    @brief register : look at datasheet
1669 *           - status: not used
1670 *           - defult value 0x00
1671 *           - Type:RW
1672 */
1673 #define AD7768_Channel_0_gain_MSB       0x36
1674 /** @def AD7768_Channel_0_gain_MID
1675 *    @brief register : look at datasheet
1676 *           - status: not used
1677 *           - defult value 0x00
1678 *           - Type:RW
1679 */
1680 #define AD7768_Channel_0_gain_MID       0x37
1681 /** @def AD7768_Channel_0_gain_LSB
1682 *    @brief register : look at datasheet
1683 *           - status: not used
1684 *           - defult value 0x00
1685 *           - Type:RW
1686 */
1687 #define AD7768_Channel_0_gain_LSB       0x38
1688 /** @def AD7768_Channel_1_gain_MSB
1689 *    @brief register : look at datasheet
1690 *           - status: not used
1691 *           - defult value 0x00
1692 *           - Type:RW
1693 */
1694 #define AD7768_Channel_1_gain_MSB       0x39
1695 /** @def AD7768_Channel_1_gain_MID
1696 *    @brief register : look at datasheet
1697 *           - status: not used
1698 *           - defult value 0x00
1699 *           - Type:RW
1700 */
1701 #define AD7768_Channel_1_gain_MID       0x3A
1702 /** @def AD7768_Channel_1_gain_LSB
1703 *    @brief register : look at datasheet
1704 *           - status: not used
1705 *           - defult value 0x00
1706 *           - Type:RW
1707 */
1708 #define AD7768_Channel_1_gain_LSB       0x3B
1709 /** @def AD7768_Channel_2_gain_MSB
1710 *    @brief register : look at datasheet
1711 *           - status: not used
1712 *           - defult value 0x00
1713 *           - Type:RW
1714 */
1715 #define AD7768_Channel_2_gain_MSB       0x3C
1716 /** @def AD7768_Channel_2_gain_MID
1717 *    @brief register : look at datasheet
1718 *           - status: not used
1719 *           - defult value 0x00
1720 *           - Type:RW
1721 */
```

```
1722 #define AD7768_Channel_2_gain_MID       0x3D
1723 /** @def AD7768_Channel_2_gain_LSB
1724 *    @brief register : look at datasheet
1725 *           - status: not used
1726 *           - defult value 0x00
1727 *           - Type:RW
1728 */
1729 #define AD7768_Channel_2_gain_LSB       0x3E
1730 /** @def AD7768_Channel_3_gain_MSB
1731 *    @brief register : look at datasheet
1732 *           - status: not used
1733 *           - defult value 0x00
1734 *           - Type:RW
1735 */
1736 #define AD7768_Channel_3_gain_MSB       0x3F
1737 /** @def AD7768_Channel_3_gain_MID
1738 *    @brief register : look at datasheet
1739 *           - status: not used
1740 *           - defult value 0x00
1741 *           - Type:RW
1742 */
1743 #define AD7768_Channel_3_gain_MID       0x40
1744 /** @def AD7768_Channel_3_gain_LSB
1745 *    @brief register : look at datasheet
1746 *           - status: not used
1747 *           - defult value 0x00
1748 *           - Type:RW
1749 */
1750 #define AD7768_Channel_3_gain_LSB       0x41
1751 /** @def AD7768_Channel_0_sync_offset
1752 *    @brief register : look at datasheet
1753 *           - status: not used
1754 *           - defult value 0x00
1755 *           - Type:RW
1756 */
1757 #define AD7768_Channel_0_sync_offset    0x4E
1758 /** @def AD7768_Channel_1_sync_offset
1759 *    @brief register : look at datasheet
1760 *           - status: not used
1761 *           - defult value 0x00
1762 *           - Type:RW
1763 */
1764 #define AD7768_Channel_1_sync_offset    0x4F
1765 /** @def AD7768_Channel_2_sync_offset
1766 *    @brief register : look at datasheet
1767 *           - status: not used
1768 *           - defult value 0x00
1769 *           - Type:RW
1770 */
1771 #define AD7768_Channel_2_sync_offset    0x50
1772 /** @def AD7768_Channel_3_sync_offset
1773 *    @brief register : look at datasheet
1774 *           - status: not used
1775 *           - defult value 0x00
1776 *           - Type:RW
1777 */
1778 #define AD7768_Channel_3_sync_offset    0x51
1779 /** @def AD7768_Diagnostic_Rx
1780 *    @brief register : look at datasheet
1781 *           - status: not used
1782 *           - defult value 0x00
1783 *           - Type:RW
1784 */
1785 #define AD7768_Diagnostic_Rx            0x56
1786 /** @def AD7768_Diagnostic_mux_control
1787 *    @brief register : look at datasheet
1788 *           - status: not used
1789 *           - defult value 0x00
1790 *           - Type:RW
1791 */
1792 #define AD7768_Diagnostic_mux_control   0x57
1793 /** @def AD7768_Modulator_delay_control
1794 *    @brief register : look at datasheet
1795 *           - status: not used
1796 *           - defult value 0x00
1797 *           - Type:RW
1798 */
1799 #define AD7768_Modulator_delay_control 0x58
1800 /** @def AD7768_Chop_control
1801 *    @brief register : look at datasheet
1802 *           - status: not used
1803 *           - defult value 0x00
1804 *           - Type:RW
```

```
1805 */
1806 #define AD7768_Chop_control             0x59
1807
1808 /**
1809  * @brief ad7768_config_address
1810  *
1811  *  This pointer used to read the values AD7768_config_table
1812  */
1813 typedef struct{
1814     unsigned int ad7768_config_Address;/**< stores position at config table. */
1815 }ad7768_config;
1816 /**
1817  * @brief holds values for ad7768 filter settings
1818  *
1819  */
1820 enum{ad7768_filter_wide,
1821     ad7768_filter_SINC};
1822 /**
1823  * @brief struct ad7768_settings_t holds values of the  following configurations to AD7768
1824  *       - MCLK is clock provided by TMS570 ECLK 8MHz
1825  *       - DCLK_DIV control division of the DCLK clock used to clock out conversion data on
     the DOUTx pins this derived from MCLK
1826  *       - decRate is Decimation rate output a data output from each channel. The decimation
     rates allow the user to reduce the measurement bandwidth, reducing the speed but
     increasing the resolution.
1827  *       - filter mode configured
1828  *       - fmod is the internal modular frequency that is used by each of the ADCs in the
     AD7768 this is deerived from MCLK. this dependent on MCLK and MCLK_div. this fMOD, to
     reject tones or harmonics related to the modulator clock.
1829  *       - DCLK this controller the rate of the output data data like DRDY framing output,
     and the data output pins DOUT[0-7]
1830  *       - ODR the output data rate 32 128 and 256 kSPS
1831  */
1832 typedef struct {
1833     uint8_t MCLK;        /**< Master clock. */
1834     uint8_t DCLK_DIV;    /**< data clock division. */
1835     uint8_t MCLK_DIV;   /**< Master clock division. */
1836     uint16_t decRate; /**< Decimation rate. */
1837     uint8 filter;     /**<Filter wide or SINC read more in datasheet for AD7768. */
1838     uint8_t fMOD;     /**<frequency modular. */
1839     uint8_t DCLK;     /**<Data Clock. */
1840     float ODR;      /**<Output data rate in Kilo samples per second */
1841 } ad7768_settings_t;
1842 /*============================== HTU and NHET ==============================*/
1843 /* High End Timer read address = ((instruction number * 4) + 2) */
1844 #define NHET_RAM_ADDRESS                        (38)
1845
1846 /* Number of 32-bit elements. Four data channels and one counter */
1847 #define NHET_DATA_FIELD_ELEMENT_COUNT           (5)
1848
1849 /* HTU_FRAME_TRANSFER_COUNT_MAX must be a multiple of
1850  * NHET_DATA_FIELD_ELEMENT_COUNT but smaller or equal
1851  * to 0xff which is the maximum HTU frame size
1852  */
1853 #define HTU_FRAME_TRANSFER_COUNT_MAX            (255)
1854 uint32_t htu_buffer_a[HTU_FRAME_TRANSFER_COUNT_MAX];
1855 uint32_t htu_buffer_b[HTU_FRAME_TRANSFER_COUNT_MAX];
1856
1857 #define ADC_BUFFER_SAMPLE_COUNT                 (4000)
1858 #define ADC_BUFFER_CHANNEL_COUNT                (2)
1859 #define ADC_BUFFER_SIZE                         (NHET_DATA_FIELD_ELEMENT_COUNT * \
1860                                                  ADC_BUFFER_SAMPLE_COUNT)
1861 struct adc_samples_s {
1862     union {
1863         struct {
1864             uint16_t counter[ADC_BUFFER_SAMPLE_COUNT];
1865             uint16_t data[ADC_BUFFER_CHANNEL_COUNT][ADC_BUFFER_SAMPLE_COUNT];
1866             uint64_t x;
1867             uint16_t index;
1868         };
1869         uint8_t all[ADC_BUFFER_SAMPLE_COUNT];
1870     };
1871 };
1872
1873
1874
1875 /* The 7 least significant bits of the counter channel is not used */
1876 #define SAMPLE_COUNTER_SHIFT                    (7)
1877 /*============================== reg_data  and safety features ==============================
     */
1878
1879 /*============================== Functions ==================================================
     */
```

```
1880 void IVmeasurements();
1881 void EnablePower5V12V(int );
1882 void EnablePower5V12_readback();
1883 int Antenna_RLS_DTCT(int);
1884 void Antenna_get_status();
1885 void DAC_SPI_confi(uint8 ,uint8 ,uint8);
1886 void DAC_Gain_SET_SELECT(uint8);
1887 void DAC_get_gain_status();
1888 void DAC_GAIN_SET_STATUS();
1889 double pow(double,double);
1890 void cmd_enable_rails(int,int);
1891 void cmd_IV_measur(int,int);
1892 void cmd_DAC_SPI_config(int, int,int);
1893 void cmd_DAC_Gain_Sel_EN(int,int);
1894 void cmd_DAC_GAIN_SET_STATUS(int,int);
1895 void cmd_Antenna_RLS_DTCT(int,int);
1896
1897 int ltc_I2C_TxRx_timeout (uint8);
1898 void I2C_config(uint32, uint32, bool);
1899 int PMBus_read (uint8 ,uint8*, uint8 );
1900 int PMBus_write (uint8 , uint8* , uint8 );
1901 int ltc3887_ready (void);
1902 int ltc3887_init (void);
1903 int ltc3887_read_config();
1904 int ltc3887_write_register (uint8 , uint8 ,uint8 , uint16 );
1905 int ltc3887_read_register (uint8, uint8 ,uint8, uint16*);
1906 int ltc3887_L16u_reg (uint16, uint8*);
1907 int ltc3887_L16u_data (uint8 * ,uint16*);
1908 int ltc3887_L5s_11s_reg (uint16 ,uint8 *);
1909 int ltc3887_L5s_11s_data (uint8 *, uint16 *);
1910 int ltc3887_turn_on_off_channel (uint8, bool);
1911 int ltc3887_set_output_voltage (uint8, uint16);
1912 int ltc3887_output_current (uint8, uint16 * );
1913 int ltc3887_Read_internal_Temperature (uint8 , uint16 * );
1914 int ltc3887_Read_output_voltage (uint8, uint16 *);
1915 void cmd_ltc3887_turn_on_off_channel(int);
1916 void cmd_ltc3887_Read_output_voltage(int,int);
1917 void cmd_ltc3887_Read_internal_Temperature(int,int);
1918 void cmd_ltc3887_output_current(int, int);
1919 void cmd_ltc3887_set_output_voltage(int, int);
1920 void cmd_ltc3887_read_config();
1921
1922 int high_speed_transfer_unit_init (void);
1923 void start_het (void);
1924 void stop_het (void);
1925 int ad7768_init (void);
1926 int ad7768_spi_read (uint8, uint8 *);
1927 int ad7768_spi_write (uint8 , uint8);
1928 int ad7768_calc_fMOD (void);
1929 int ad7768_calc_DCLK (void);
1930 int ad7768_calc_ODR (void);
1931 int ad7768_set_DCLK_div (uint8);
1932 int ad7768_set_MCLK_div (uint8);
1933 int ad7768_set_Drate (uint16);
1934 int ad7768_set_Filter (uint8);
1935 int ad7768_reset();
1936 int ad7768_sync();
1937 void ad7768_print_settings();
1938 void cmd_AD7768_GET_REG(uint8 arg);
1939 void cmd_AD7768_SET_REG(uint8 reg,uint8 value);
1940 void cmd_AD7768_GET_AllConfigs(void);
1941 void cmd_AD7768_SET_FILTER(uint8 arg);
1942 void cmd_AD7768_SET_DRATE(uint16 arg);
1943 void cmd_AD7768_SET_MCLK_div(uint8 arg);
1944 void cmd_AD7768_run(void);
1945 void cmd_AD7768_stop(void);
1946 void cmd_AD7768_print_Data(void);
1947 void cmd_AD7768_SET_DCLK_div(uint8 arg);
1948
1949 int reg_data_MibAdc2();
1950 int reg_data_MibAdc1();
1951 int reg_data_Dcan3();
1952 int reg_data_Dcan2();
1953 int reg_data_Dcan1();
1954 int reg_data_ePWN1();
1955 int reg_data_ePWN2();
1956 int reg_data_ePWN3();
1957 int reg_data_ePWN4();
1958 int reg_data_ePWN5();
1959 int reg_data_ePWN6();
1960 int reg_data_ePWN7();
1961 int reg_data_eCAP1();
1962 int reg_data_eCAP2();
```

```
1963 int reg_data_eCAP3();
1964 int reg_data_eCAP4();
1965 int reg_data_eCAP5();
1966 int reg_data_eCAP6();
1967 int reg_data_eQEP1();
1968 int reg_data_eQEP2();
1969 int reg_data_Gio();
1970 int reg_data_GioA();
1971 int reg_data_GioB();
1972 int reg_data_I2C();
1973 int reg_data_NHET1();
1974 int reg_data_NHET2();
1975 int reg_data_HTU1();
1976 int reg_data_HTU2();
1977 int reg_data_IOMM();
1978 int reg_data_MibSpi1();
1979 int reg_data_Spi2();
1980 int reg_data_MibSpi3();
1981 int reg_data_Spi4();
1982 int reg_data_MibSpip5();
1983 int reg_data_Lin2();
1984 int reg_data_Lin1();
1985 int reg_data_CcmR4();
1986 int reg_data_Crc();
1987 int reg_data_Dcc1();
1988 int reg_data_Dcc2();
1989 int reg_data_Dma();
1990 int reg_data_Esm();
1991 int reg_data_flashWrapper();
1992 int reg_data_Pbist();
1993 int reg_data_PMM();
1994 int reg_data_Rti();
1995 int reg_data_Stc();
1996 int reg_data_Sys();
1997 int reg_data_Sys2();
1998 int reg_data_Vim();
1999 int reg_data_VimPar();
2000 int reg_data_Pom();
2001 int reg_data_Emif();
2002 int reg_data_Pcr();
2003 int reg_data_RamWrapper_Even();
2004 int reg_data_RamWrapper_Odd();
2005 int RAM_data_read();
2006 void reg_data_cmd();
2007 void RAMtest_init();
2008 void raw_board_status();
2009 static int add_to_buffer (unsigned int a, unsigned int b, int state,uint8 message_id);
2010 void dac_status();
2011 int esm1_read();
2012 int esm2_read();
2013 void ecc_data_test();
2014
2015 #endif /* INC_SysBoard_H_ */
```

```
1  /*
2   * SysBoard.c
3   *
4   *  Created on: Sep 21, 2017
5   *      Author: Yassine Elfarri
6   */
7
8  #include "SysBoard.h"
9  /**
10  * @brief  declare the struct of adc samples
11  *
12  */
13 volatile struct adc_samples_s adc_samples = {
14
15     .index = 0,
16 };
17
18 /* TODO set false/true using command */
19 bool raw_data = true;
20 bool new_sampling_run = false;
21
22 /**
23  * @brief  initial MeasuredVI gobal variable to be used internal adc to read current and
       voltage values.
24  *
25  */
26 struct MeasuredVI MON_I_V;
27 /**
28  * @brief  initial Bias_Measured_V gobal variable to be used internal adc to readback values
```

```
     of bias applied on the probes. This part of DAC bias
 *
 */
struct Bias_Measured_V BIAS_MON_V;

struct ltc_read ltc_m_values;

/** @fn void EnablePower5V12V(int number)
*   @brief Enables power rails 5V,DCDC 12V, and Egun P5V0.
*   @param[in] number a single variable that turn on power rails values should be between
     0-7.
*
*
*   The reasoning behind this to make a think binary of 3 bits.
*   the first bit is from MSB to LSB where 5V is the MSB, MID = DCDC 12V, and lab egun P5v0
*
*        number = (5V)^2+(DCDC 12V)^1+(egun P5v0)^0
*        where  0 is off and 1 is on
*
*
*
*/
void EnablePower5V12V(int number)
{
   switch(number)
   {
      case 0:
               P5V0_EN_PIN_Off;
               DCDC12V_EN_PIN_Off;
               P5V0_EGUN_EN_PIN_Off;
          break;

      case 1:
               P5V0_EN_PIN_Off;
               DCDC12V_EN_PIN_Off;
               P5V0_EGUN_EN_PIN_On;

            break;
      case 2:
               P5V0_EN_PIN_Off;
               DCDC12V_EN_PIN_On;
               P5V0_EGUN_EN_PIN_Off;

            break;
      case 3:
                P5V0_EN_PIN_Off;
                DCDC12V_EN_PIN_On;
                P5V0_EGUN_EN_PIN_On;

             break;
      case 4:
                P5V0_EN_PIN_On;
                DCDC12V_EN_PIN_Off;
                P5V0_EGUN_EN_PIN_Off;

              break;
      case 5:
               P5V0_EN_PIN_On;
               DCDC12V_EN_PIN_Off;
               P5V0_EGUN_EN_PIN_On;

              break;
      case 6:
              P5V0_EN_PIN_On;
              DCDC12V_EN_PIN_On;
              P5V0_EGUN_EN_PIN_Off;

               break;
      case 7:
               P5V0_EN_PIN_On;
               DCDC12V_EN_PIN_On;
               P5V0_EGUN_EN_PIN_On;

               break;
   }
   EnablePower5V12_readback();
}
/**
 * @brief  initial Status_power_rails gobal get status of which pin is on or off */
 struct Power_switches Status_power_rails;

 /**@fn void EnablePower5V12_readback()
  * @brief Function reads back the setting of the pins
```

```
110   *
111   * This functions reads back the configuration of 3 rails
112   *  - DCDC12V: supplies P12V and N12V
113   *  - P5V0 : Supplied for the AD7768 and others circuits that requires this value
114   *  - P5V0_EGUN: supply for Egun.
115   */
116
117 void EnablePower5V12_readback()
118 {
119     Status_power_rails.P5V0_EN_PIN =  gioGetBit(hetPORT1,PIN_HET_9);
120     Status_power_rails.DCDC12V_EN_PIN = gioGetBit(hetPORT1,PIN_HET_22);
121     Status_power_rails.P5V0_EGUN_EN_PIN =  gioGetBit(hetPORT1,PIN_HET_25);
122 }
123 /** @fn void IVmeasurements()
124 *   @brief  Reads current and voltage values from the TMS570 internal ADC.
125 *   The values are updated only if the method is call.
126 *    Voltage monitor values:
127 *       - Egun 5V0 trace monitor
128 *       - P12V positive 12 volt trace
129 *       - N12V negative 12 volt trace
130 *       - P3V3 3.3V power trace monitor
131 *       - P5V0 5V power trace
132 *
133 *    Current values:
134 *       - P1V2_I current monitor
135 *       - Egun 5V0 trace monitor
136 *       - P12V positive 12 volt trace
137 *       - N12V negative 12 volt trace
138 *       - P3V3 3.3V power trace monitor
139 *       - P5V0 5V power trace
140 *
141 * Todo: from technical document i see that possible to use ADC DMA unit to update values
        continues without taking resources from the CPU.
142 *
143 */
144 void IVmeasurements()
145 {
146    /*number of ADC channels to read from */
147     adcData_t data[ADC_Channels_cnt];
148     /*Empty ADC FIFO to ensure reading the correct channel in right order*/
149     adcResetFiFo(adcREG1, adcGROUP1);
150     /*Conversion of values and put them in FIFO*/
151     adcStartConversion (adcREG1, adcGROUP1);
152     //wait(20000);
153     /*Wait for the conversion to complete*/
154     while (!adcIsConversionComplete(adcREG1, adcGROUP1));
155     /*Start reading data from FIFO*/
156     adcGetData(adcREG1, adcGROUP1,data);
157     /*Odd issues the if function should have do this*/
158     MON_I_V.P1V2_I= ADCmVPerBit * ADC_IMON * data[P1V2_I_ID].value ;
159
160     MON_I_V.EGUN_I = ADCmVPerBit * ADC_IMON * data[EGUN_I_ID].value ;
161
162     MON_I_V.P5V0_I = ADCmVPerBit * ADC_IMON * data[P5V0_I_ID].value ;
163     MON_I_V.P5V0_V = ADCmVPerBit * VMONP5V0 * data[P5V0_V_ID].value ;
164
165      MON_I_V.P3V3_V = ADCmVPerBit *  VMON_P3V3 * data[P3V3_V_ID].value ;
166      MON_I_V.P3V3_I = ADCmVPerBit * ADC_IMON * data[P3V3_I_ID].value ;
167
168      MON_I_V.P12V_V = ADCmVPerBit * VMON_P12V * data[P12V_V_ID].value ;
169      MON_I_V.P12V_I = ADCmVPerBit * ADC_IMON * data[P12V_I_ID].value ;
170
171      MON_I_V.N12V_V = ADCmVPerBit * VMON_N12V * data[N12V_V_ID].value ;
172      MON_I_V.N12V_I = ADCmVPerBit * ADC_IMON * data[N12V_I_ID].value ;
173
174      if(!raw_data){
175          obc_debug("P1V2(mA),%d", MON_I_V.P1V2_I);
176          obc_debug("Egun(mA), %d", MON_I_V.EGUN_I);
177          obc_debug("P5V0(mV)(mA),%d,%d", MON_I_V.P5V0_V, MON_I_V.P5V0_I);
178          obc_debug("P3V3(mV)(mA),%d,%d", MON_I_V.P3V3_V, MON_I_V.P3V3_I);
179          obc_debug("P12V(mV)(mA),%d,%d", MON_I_V.P12V_V, MON_I_V.P12V_I);
180          obc_debug("N12V(mV)(mA),-%d,%d", MON_I_V.N12V_V, MON_I_V.N12V_I);
181          }
182
183
184 }
185
186 /** @fn void Antenna_RLS_DTCT()
187 *   @brief  Reads current and voltage values from the TMS570 internal ADC.
188 *   @param[in] Antenna_RLS_NR annetna released each number corspondes to antena
189 *
190     the function received the antenna that needed to be released wait for the systems until
        it detects a release.
```

```
191      number of antennas to be release are 4. to release each antenna method should be called
         each time.
192
193      Antenna is released when can2-3 input is pulled low. For testing use a large resistor
         connect one side to GND and other to  corrisponding channel pin. when the pin is pulled
         down a realse signal is detected.
194 */
195 int Antenna_RLS_DTCT(int Antenna_RLS_NR)
196 {
197
198     switch(Antenna_RLS_NR)
199     {
200         case 0:
201             Ant_RLS_1_On;
202             while(canIoRxGetBit(Antenna_DTCT_1_2_Reg));
203             Ant_RLS_1_Off;
204             return 1;
205
206         case 1:
207             Ant_RLS_2_On;
208             while(canIoTxGetBit(Antenna_DTCT_1_2_Reg));
209             Ant_RLS_2_Off;
210             return 1;
211
212         case 2:
213             Ant_RLS_3_On;
214             while(canIoRxGetBit(Antenna_DTCT_3_4_Reg));
215             Ant_RLS_3_Off;
216             return 1;
217
218         case 3:
219             Ant_RLS_4_On;
220             while(canIoTxGetBit(Antenna_DTCT_3_4_Reg));
221             Ant_RLS_4_Off;
222             return 1;
223
224         default:
225             return 0;
226
227     }
228
229 }
230
231 /**
232  * @brief  initial antenna_status get status of antenna[1-4]
233  *
234  */
235  struct antenna_status antenna_getstatus;
236
237  /**@fn void Antenna_get_status()
238   * @brief Function reads back the setting of the pins
239   *
240   * This functions get back the configuration  antennas 0 is for released
241   *
242   */
243 void Antenna_get_status()
244 {
245
246     antenna_getstatus.Antenna_1 = canIoRxGetBit(Antenna_DTCT_1_2_Reg) ;
247     antenna_getstatus.Antenna_2 = canIoTxGetBit(Antenna_DTCT_1_2_Reg) ;
248     antenna_getstatus.Antenna_3 = canIoRxGetBit(Antenna_DTCT_3_4_Reg) ;
249     antenna_getstatus.Antenna_4 = canIoTxGetBit(Antenna_DTCT_3_4_Reg) ;
250 }
251
252
253
254
255 /** @fn void DAC_SPI_confi(uint8 dac_channel ,uint8 dac_mode, uint8 bias_voltage)
256 *   @brief Setts a channel to a spesific bias voltage and mode
257 *   @param[in] dac_channel channel
258 *   @param[in] dac_mode
259 *   @param[in] bias_voltage in voltages
260 *
261 *   the param are:
262 *       - channel 0-3 for DAC
263 *       - Mode :
264 *           - 0 = write to channel, do not update.
265 *           - 1 = write to channel. update channel bias voltage
266 *           - 2 = write and update all the channels with the same bias voltage.
267 *           - 3 = power down DAC.
268 *
269 *    The command send over SPI :
270 *    The dac is labeled X15 in mNLP Hercules schematics. The model is a TI DAC104S085. This
```

```
    DAC is configured to receive data via spi.
271 *    The data received will be put into a shift register. The shift register size if 16 bits.
272 *
273 *       | A1| A0| OP1| OP0| D11| D10| D9| D8| D7| D6| D5| D4| D3| D2| D1| D0  |
274 *       A1 is MSB bit 15 and D0 is LSB bit[0].
275 *
276 *       - A[1-0]  this are address bit to channel from channel[0-3]. values 00 for channel 0
277 *       - OP[1-0] Operation bits determ the mode named earlier
278 *       - D[11-2] the those bit determin the bias voltage.
279 *
280 *
281 *
282 */
283
284 void DAC_SPI_confi(uint8 dac_channel ,uint8 dac_mode, uint8 bias_voltage)
285 {
286
287     if(!gioGetBit(hetPORT1, PIN_HET_22))
288         {
289             DCDC12V_EN_PIN_On;
290
291             //SendStringSCI("12V is turned on.",True);
292             //implement wait for rail to reach 12V
293         }
294     spiDAT1_t dataconfig1_t;
295     dataconfig1_t.CS_HOLD = FALSE;
296     dataconfig1_t.WDEL    = True;
297     dataconfig1_t.DFSEL   = SPI_FMT_0;
298     dataconfig1_t.CSNR    = 0xFE;
299     /* Calculate code to write to DAC to do
300          * 10bit DAC = 1024, 5V range * 4 weighting from OPAMPs = 20
301          * change opamp weight if gain is changed
302     */
303     int calc_value = (10 - bias_voltage) * 1024/20;
304     uint16 Buff_tx;
305
306     /*Format the DAC to type DAC can read*/
307     Buff_tx = (dac_channel & 0x3) << 14 |(dac_mode & 0x3) << 12 |(calc_value & 0x3ff) << 2;
308
309
310     spiTransmitData(spiREG3, &dataconfig1_t, 1,(uint16*)&Buff_tx);
311
312 }
313 /** @fn DAC_Gain_SET_SELECT(uint8 Gain_SEL )
314 *   @brief Setts a channel to a spesific bias voltage and mode
315 *   @param[in] Gain_SEL the gain can be selected to be 10 or 20
316 *
317 *    valid params are:
318 *                  - 0 turn off gain and set Gain to 10 as default
319 *                  - 1 turn on gain the values is gain = 10
320 *                  - 2 gain to 10
321 *                  - 3 set gain to 20
322 *
323 *
324 */
325
326 void DAC_Gain_SET_SELECT(uint8 Gain_SEL )
327 {
328     switch(Gain_SEL)
329      {
330         case 0:
331             Gain_Enable_Off;
332             Gain_SEL_G10;
333             break;
334
335         case 1:
336                 Gain_Enable_On;
337                 Gain_SEL_G10;
338                 break;
339         case 2:
340             Gain_SEL_G10;
341              break;
342         case 3:
343             Gain_SEL_G20;
344             break;
345
346         default:
347                SendStringSCI("Not valid value",True);
348
349      }
350
351 }
352
```

```
353 struct Gain_sel_om Gain_status;
354 /** @fn DAC_get_gain_status()
355 *   @brief Get status of which gain is selected
356 *
357 *   This functions get the value of gain from GPIO channels:
358 *   - Gain is enabled to channel 1-4 is value 1
359 *   - GAin_10 is selected to channels is value 0
360 *   - GAin_20 is slected to channels value 1
361 *   This values are from the GPIO PIN_HET_15 and 19
362 */
363 void DAC_get_gain_status()
364 {
365     Gain_status.Gain_on_off =gioGetBit(hetPORT1,PIN_HET_15);
366     if(gioGetBit(hetPORT1,PIN_HET_19) == 0)
367     {
368         Gain_status.Gain_10 =1;
369         Gain_status.Gain_20=0;
370     }
371     else if(gioGetBit(hetPORT1,PIN_HET_19) == 1)
372     {
373         Gain_status.Gain_10 =0;
374         Gain_status.Gain_20=1;
375     }
376
377 }
378
379 /** @fn DAC_GAIN_SET_STATUS()
380 *   @brief Readback value of all channels
381 *
382 *   this function will be handy to verify the functionality of DAC value during remote
      operation
383 *
384 *
385 *
386 */
387 void DAC_GAIN_SET_STATUS()
388 {
389
390
391
392         adcData_t data[BIAS_ADC_Channels_cnt];
393         adcResetFiFo(adcREG1, adcGROUP2);
394         adcStartConversion (adcREG1, adcGROUP2);
395         //wait(20000);
396         while (!adcIsConversionComplete(adcREG1, adcGROUP2));
397         adcGetData(adcREG1, adcGROUP2,data);
398
399         BIAS_MON_V.DAC_BIAS_CH1= ((data[DAC_BCH1_ID].value - 3134)*- 2.9) +500 ;
400
401         BIAS_MON_V.DAC_BIAS_CH2=  ((data[DAC_BCH2_ID].value - 3134) *-2.9) + 500;
402
403         BIAS_MON_V.DAC_BIAS_CH3= (( data[DAC_BCH3_ID].value - 3134)*-2.9) + 500;
404
405         BIAS_MON_V.DAC_BIAS_CH4= ((data[DAC_BCH4_ID].value - 3134) *-2.9) + 500;
406
407         if(!raw_data){
408             obc_debug("Bias CH1(mV): %d",BIAS_MON_V.DAC_BIAS_CH1);
409             obc_debug("Bias CH2(mV): %d",BIAS_MON_V.DAC_BIAS_CH2);
410             obc_debug("Bias CH3(mV): %d",BIAS_MON_V.DAC_BIAS_CH3);
411             obc_debug("Bias CH4(mV): %d",BIAS_MON_V.DAC_BIAS_CH4);
412         }
413 }
414
415
416 /** @fn void cmd_enable_rails( int arg1, int arg2)
417 *   @brief This function  cmd check for validation of value
418 *   @param[in] arg1 should be between [0-7]
419 *   @param[in] arg2 not used just to check for unvalid args
420 *
421 */
422
423 void cmd_enable_rails( int arg1, int arg2)
424 {
425     if(arg1 > 7 || arg2 != 0)
426     {
427         SciSendACK(2);
428         //SciTooManyargs();
429         //goto Error;
430     }
431     EnablePower5V12V(arg1);
432 }
433
434 /** @fn void cmd_IV_measur(int arg1,int arg2)
```

```
435 *   @brief This function  cmd check for validation of value
436 *   @param[in] arg1 not used just to check for unvalid calls
437 *   @param[in] arg2 not used just to check for unvalid args
438 *
439 */
440
441
442 void cmd_IV_measur(int arg1,int arg2)
443 {
444     if(arg1!= 0 || arg2!= 0)
445     {
446         SciSendACK(2);
447     }
448
449     IVmeasurements();
450 }
451
452 /** @fn void cmd_DAC_SPI_config(int arg1, int arg2,int arg3)
453 *   @brief This function  cmd check for validation of value
454 *   @param[in] arg1 channel to be set
455 *   @param[in] arg2  Mode value to be set
456 *   @param[in] arg3  Voltage value to be selected
457 */
458
459 void cmd_DAC_SPI_config(int arg1, int arg2,int arg3)
460 {
461
462     if(arg1> 3 || arg2> 3)
463     {
464             SciSendACK(2);
465             //SciTooManyargs();
466     }
467    DAC_SPI_confi(arg1,arg2,arg3);
468 }
469
470 /** @fn void cmd_DAC_Gain_Sel_EN(int arg1, int arg2)
471 *   @brief This function  cmd check for validation of value
472 *   @param[in] arg1 set gain
473 *   @param[in] arg2  none
474 *
475 */
476 void cmd_DAC_Gain_Sel_EN(int arg1, int arg2)
477 {
478     if(arg1> 3 || arg2> 3)
479     {
480         SciSendACK(2);
481     }
482
483     DAC_Gain_SET_SELECT(arg1);
484 }
485
486 /** @fn void cmd_DAC_GAIN_SET_STATUS(int arg1,int arg2)
487 *   @brief This function  cmd check for validation of value
488 *   @param[in] arg1 channel to be set
489 *   @param[in] arg2  Mode value to be set
490 *
491 */
492 void cmd_DAC_GAIN_SET_STATUS(int arg1, int arg2)
493 {
494     if(arg1!= 0 || arg2!=0)
495     {
496         SciSendACK(2);
497     }
498
499     DAC_GAIN_SET_STATUS();
500 }
501
502 /** @fn void cmd_Antenna_RLS_DTCT(int arg1,int arg2)
503 *   @brief This function  cmd check for validation of value
504 *   @param[in] arg1 channel to be set
505 *   @param[in] arg2  Mode value to be set
506 *
507 */
508
509 void cmd_Antenna_RLS_DTCT(int arg1,int arg2)
510 {
511     if(arg1> 3 || arg2> 3)
512    {
513         SciSendACK(2);
514    }
515    Antenna_RLS_DTCT(arg1);
516 }
517 /************************************ LTC3887 functions
```

```
    ****************************************/
518
519 /**
520 *   @brief table that hold the information from type of command, register address, size and
        values
521 *
522 *   the LTC3887 uses two type of  address values use some register can be access using global
        address while other needs a paged address.
523 *   this table is the best way to hold  use the correct address with the command. Some
        registers requires two bytes while other only one.
524 *   this a good way to this information on what to expect while reading and writing.
525 *
526 *   - Address , command, size to be written or read, LSB then MSB.
527 */
528 ltc_config const ltc_config_table[] = {
529     {GLOBAL, LTC3887_WRITE_PROTECT                  , One_Byte,  0x00, 0x00},
530     {GLOBAL, LTC3887_FREQUENCY_SWITCH               , Two_Bytes, 0xEE, 0x02},
531     {GLOBAL, LTC3887_VIN_ON                         , Two_Bytes, 0xC0, 0xCB},
532     {GLOBAL, LTC3887_VIN_OFF                        , Two_Bytes, 0x40, 0xCB},
533     {GLOBAL, LTC3887_VIN_OV_FAULT_LIMIT             , Two_Bytes, 0x00, 0xD3},
534     {GLOBAL, LTC3887_VIN_UV_WARN_LIMIT              , Two_Bytes, 0x80, 0xCB},
535     {GLOBAL, LTC3887_IIN_OC_WARN_LIMIT              , Two_Bytes, 0x80, 0xCA},
536     {GLOBAL, LTC3887_STATUS_INPUT                   , One_Byte,  0x00, 0x00},
537     {GLOBAL, LTC3887_STATUS_CML                     , One_Byte,  0x00, 0x00},
538     {GLOBAL, LTC3887_USER_DATA_00                   , Two_Bytes, 0x00, 0x00},
539     {GLOBAL, LTC3887_USER_DATA_02                   , Two_Bytes, 0x00, 0x00},
540     {GLOBAL, LTC3887_USER_DATA_04                   , Two_Bytes, 0x00, 0x00},
541     {GLOBAL, LTC3887_MFR_CONFIG_ALL_LTC3887         , One_Byte,  0x61, 0x00},
542     {GLOBAL, LTC3887_MFR_ADDRESS                    , One_Byte,  0x4F, 0x00},
543     {GLOBAL, LTC3887_MFR_PWM_CONFIG_LTC3887         , One_Byte,  0x00, 0x00},
544     {GLOBAL, LTC3887_Page                           , One_Byte,  0x0, 0x00},
545     {PAGED,  LTC3887_OPERATION                      , One_Byte,  0x80, 0x00},
546     {PAGED,  LTC3887_ON_OFF_CONFIG                  , One_Byte,  0x1E, 0x00},
547     {PAGED,  LTC3887_VOUT_COMMAND                   , Two_Bytes, 0x00, 0x10},
548     {PAGED,  LTC3887_VOUT_MAX                       , Two_Bytes, 0x00, 0x58},
549     {PAGED,  LTC3887_VOUT_MARGIN_HIGH               , Two_Bytes, 0xAE, 0x67},
550     {PAGED,  LTC3887_VOUT_MARGIN_LOW                , Two_Bytes, 0x66, 0x0E},
551     {PAGED,  LTC3887_VOUT_TRANSITION_RATE           , Two_Bytes, 0x8F, 0x82},
552     {PAGED,  LTC3887_IOUT_CAL_GAIN                  , Two_Bytes, 0x58, 0xFA},
553     {PAGED,  LTC3887_VOUT_OV_FAULT_LIMIT            , Two_Bytes, 0x00, 0x68},
554     {PAGED,  LTC3887_VOUT_OV_FAULT_RESPONSE         , One_Byte,  0x00, 0x00},
555     {PAGED,  LTC3887_VOUT_OV_WARN_LIMIT             , Two_Bytes, 0xD7, 0x67},
556     {PAGED,  LTC3887_VOUT_UV_WARN_LIMIT             , Two_Bytes, 0xCD, 0x0C},
557     {PAGED,  LTC3887_VOUT_UV_FAULT_LIMIT            , Two_Bytes, 0x33, 0x0B},
558     {PAGED,  LTC3887_VOUT_UV_FAULT_RESPONSE         , One_Byte,  0x00, 0x00},
559     {PAGED,  LTC3887_IOUT_OC_FAULT_LIMIT            , Two_Bytes, 0xB8, 0xA2},
560     {PAGED,  LTC3887_IOUT_OC_FAULT_RESPONSE         , One_Byte,  0x00, 0x00},
561     {PAGED,  LTC3887_IOUT_OC_WARN_LIMIT             , Two_Bytes, 0x8F, 0xA2},
562     {PAGED,  LTC3887_OT_FAULT_LIMIT                 , Two_Bytes, 0x20, 0xEB},
563     {PAGED,  LTC3887_OT_FAULT_RESPONSE              , One_Byte,  0x00, 0x00},
564     {PAGED,  LTC3887_OT_WARN_LIMIT                  , Two_Bytes, 0xA8, 0xEA},
565     {PAGED,  LTC3887_UT_FAULT_LIMIT                 , Two_Bytes, 0xDA, 0xFD},
566     {PAGED,  LTC3887_UT_FAULT_RESPONSE              , One_Byte,  0x00, 0x00},
567     {PAGED,  LTC3887_VIN_OV_FAULT_RESPONSE          , One_Byte,  0x00, 0x00},
568     {PAGED,  LTC3887_TON_DELAY                      , Two_Bytes, 0x00, 0x80},
569     {PAGED,  LTC3887_TON_RISE                       , Two_Bytes, 0x58, 0x02},
570     {PAGED,  LTC3887_TON_MAX_FAULT_LIMIT            , Two_Bytes, 0xE8, 0x03},
571     {PAGED,  LTC3887_TON_MAX_FAULT_RESPONSE         , One_Byte,  0xB8, 0x00},
572     {PAGED,  LTC3887_TOFF_DELAY                     , Two_Bytes, 0x00, 0x80},
573     {PAGED,  LTC3887_TOFF_FALL                      , Two_Bytes, 0x00, 0xD2},
574     {PAGED,  LTC3887_TOFF_MAX_WARN_LIMIT            , Two_Bytes, 0x58, 0xF2},
575     {PAGED,  LTC3887_STATUS_VOUT                    , One_Byte,  0x00, 0x00},
576     {PAGED,  LTC3887_STATUS_IOUT                    , One_Byte,  0x00, 0x00},
577     {PAGED,  LTC3887_STATUS_TEMPERATURE             , One_Byte,  0x00, 0x00},
578     {PAGED,  LTC3887_STATUS_MFR_SPECIFIC            , One_Byte,  0x11, 0x00},
579     {PAGED,  LTC3887_USER_DATA_01                   , Two_Bytes, 0x61, 0xD7},
580     {PAGED,  LTC3887_USER_DATA_03                   , Two_Bytes, 0x00, 0x00},
581     {PAGED,  LTC3887_MFR_CHAN_CONFIG_LTC3887        , One_Byte,  0x1D, 0x00},
582     {PAGED,  LTC3887_MFR_GPIO_PROPAGATE_LTC3887     , Two_Bytes, 0x93, 0x69},
583     {PAGED,  LTC3887_MFR_PWM_MODE_LTC3887           , One_Byte,  0xC1, 0x00},
584     {PAGED,  LTC3887_MFR_GPIO_RESPONSE              , One_Byte,  0xC0, 0x00},
585     {PAGED,  LTC3887_MFR_RETRY_DELAY                , Two_Bytes, 0xBC, 0xFA},
586     {PAGED,  LTC3887_MFR_RESTART_DELAY              , Two_Bytes, 0xE8, 0xFB},
587     {PAGED,  LTC3887_MFR_IIN_OFFSET                 , Two_Bytes, 0x33, 0x93},
588     {PAGED,  LTC3887_MFR_IOUT_CAL_GAIN_TC           , Two_Bytes, 0x00, 0x00},
589     {PAGED,  LTC3887_MFR_TEMP_1_GAIN                , Two_Bytes, 0x00, 0x40},
590     {PAGED,  LTC3887_MFR_TEMP_1_OFFSET              , Two_Bytes, 0x00, 0x80},
591     {PAGED,  LTC3887_MFR_RAIL_ADDRESS               , One_Byte,  0x80, 0x00},
592     {GLOBAL, LTC3887_Page                           , One_Byte,  0x1, 0x00},
593     {PAGED,  LTC3887_OPERATION                      , One_Byte,  0x00, 0x00},
594     {PAGED,  LTC3887_ON_OFF_CONFIG                  , One_Byte,  0x1E, 0x00},
595     {PAGED,  LTC3887_VOUT_COMMAND                   , Two_Bytes, 0x00, 0xA0},
596     {PAGED,  LTC3887_VOUT_MAX                       , Two_Bytes, 0x00, 0xB0},
```

```
    {PAGED,   LTC3887_VOUT_MARGIN_HIGH              , Two_Bytes, 0xCD, 0xA0},
    {PAGED,   LTC3887_VOUT_MARGIN_LOW               , Two_Bytes, 0x33, 0x9F},
    {PAGED,   LTC3887_VOUT_TRANSITION_RATE          , Two_Bytes, 0x8F, 0x82},
    {PAGED,   LTC3887_IOUT_CAL_GAIN                 , Two_Bytes, 0x9A, 0xBB},
    {PAGED,   LTC3887_VOUT_OV_FAULT_LIMIT           , Two_Bytes, 0x9A, 0xA1},
    {PAGED,   LTC3887_VOUT_OV_FAULT_RESPONSE        , One_Byte,  0xB8, 0x00},
    {PAGED,   LTC3887_VOUT_OV_WARN_LIMIT            , Two_Bytes, 0x48, 0xA1},
    {PAGED,   LTC3887_VOUT_UV_WARN_LIMIT            , Two_Bytes, 0xB8, 0x9E},
    {PAGED,   LTC3887_VOUT_UV_FAULT_LIMIT           , Two_Bytes, 0x66, 0x9E},
    {PAGED,   LTC3887_VOUT_UV_FAULT_RESPONSE        , One_Byte,  0xB8, 0x00},
    {PAGED,   LTC3887_IOUT_OC_FAULT_LIMIT           , Two_Bytes, 0xB8, 0xDB},
    {PAGED,   LTC3887_IOUT_OC_FAULT_RESPONSE        , One_Byte,  0x00, 0x00},
    {PAGED,   LTC3887_IOUT_OC_WARN_LIMIT            , Two_Bytes, 0x80, 0xDA},
    {PAGED,   LTC3887_OT_FAULT_LIMIT                , Two_Bytes, 0x20, 0xEB},
    {PAGED,   LTC3887_OT_FAULT_RESPONSE             , One_Byte,  0xB8, 0x00},
    {PAGED,   LTC3887_OT_WARN_LIMIT                 , Two_Bytes, 0xA8, 0xEA},
    {PAGED,   LTC3887_UT_FAULT_LIMIT                , Two_Bytes, 0x80, 0xE5},
    {PAGED,   LTC3887_UT_FAULT_RESPONSE             , One_Byte,  0xB8, 0x00},
    {PAGED,   LTC3887_VIN_OV_FAULT_RESPONSE         , One_Byte,  0x80, 0x00},
    {PAGED,   LTC3887_TON_DELAY                     , Two_Bytes, 0x00, 0x80},
    {PAGED,   LTC3887_TON_RISE                      , Two_Bytes, 0x00, 0xD2},
    {PAGED,   LTC3887_TON_MAX_FAULT_LIMIT           , Two_Bytes, 0x80, 0xD2},
    {PAGED,   LTC3887_TON_MAX_FAULT_RESPONSE        , One_Byte,  0xB8, 0x00},
    {PAGED,   LTC3887_TOFF_DELAY                    , Two_Bytes, 0x00, 0x80},
    {PAGED,   LTC3887_TOFF_FALL                     , Two_Bytes, 0x00, 0xD2},
    {PAGED,   LTC3887_TOFF_MAX_WARN_LIMIT           , Two_Bytes, 0x58, 0xF2},
    {PAGED,   LTC3887_STATUS_VOUT                   , One_Byte,  0x00, 0x00},
    {PAGED,   LTC3887_STATUS_IOUT                   , One_Byte,  0x00, 0x00},
    {PAGED,   LTC3887_STATUS_TEMPERATURE            , One_Byte,  0x00, 0x00},
    {PAGED,   LTC3887_STATUS_MFR_SPECIFIC           , One_Byte,  0x11, 0x00},
    {PAGED,   LTC3887_USER_DATA_01                  , Two_Bytes, 0x64, 0xAE},
    {PAGED,   LTC3887_USER_DATA_03                  , Two_Bytes, 0x00, 0x00},
    {PAGED,   LTC3887_MFR_CHAN_CONFIG_LTC3887       , One_Byte,  0x1D, 0x00},
    {PAGED,   LTC3887_MFR_GPIO_PROPAGATE_LTC3887    , Two_Bytes, 0x93, 0x69},
    {PAGED,   LTC3887_MFR_PWM_MODE_LTC3887          , One_Byte,  0x83, 0x00},
    {PAGED,   LTC3887_MFR_GPIO_RESPONSE             , One_Byte,  0xC0, 0x00},
    {PAGED,   LTC3887_MFR_RETRY_DELAY               , Two_Bytes, 0xBC, 0xFA},
    {PAGED,   LTC3887_MFR_RESTART_DELAY             , Two_Bytes, 0xE8, 0xFB},
    {PAGED,   LTC3887_MFR_IIN_OFFSET                , Two_Bytes, 0x33, 0x93},
    {PAGED,   LTC3887_MFR_IOUT_CAL_GAIN_TC          , Two_Bytes, 0x3C, 0x0F},
    {PAGED,   LTC3887_MFR_TEMP_1_GAIN               , Two_Bytes, 0x00, 0x40},
    {PAGED,   LTC3887_MFR_TEMP_1_OFFSET             , Two_Bytes, 0x00, 0x80},
    {PAGED,   LTC3887_MFR_RAIL_ADDRESS              , One_Byte,  0x80, 0x00},
    {COMPLETE,  0x00, 0x00, 0x00, 0x00}
};



/** @fn int ltc_I2C_TxRx_timeout (uint8 TxRx)
*   @brief Checks if I2C tx or RX buffer is ready to be used
*   @param[in] TxRx choose out Tx= 1 or Rx = 2
*
*   @return  returns 0 if function is successful, and return 1 = failed when ready flag is
     never received
*
*   Checks if the Tx or Rx buffer flag is set. Returns 0 if flag is not set , else turns flag
     it  self.
*   This function will return Return_Fail only if Param[in] not equal to 1 or 2.
*/

int ltc_I2C_TxRx_timeout (uint8 TxRx) {

   #define timeout 0x200000
    int cnt=0;

    switch(TxRx)
    {
        case 1:
            while(cnt < timeout) {
                if (i2cIsTxReady(i2cREG1) != 0U)
                    return 0;
                cnt++;
            }
            break;

        case 2:
            while(cnt < timeout) {
                    if (i2cIsRxReady(i2cREG1) != 0U)
                        return 0;
                    cnt++;
               }
            break;

```

```
678         default:
679                 return 1;
680     }
681
682     return 1;
683 }
684 /** @fn void I2C_config(uint32 length, uint32 dir, bool reset_in)
685 *   @brief Checks if I2C tx or RX buffer is ready to be used
686 *   @param[in] length size of byte read/write
687 *   @param[in] dir  Transmission or receive mode
688 *   @param[in] reset_in reset I2C configuration before each call
689 *
690 *  This function reconfigures I2C settings to meet the need of switching between rx and tx as
      well different sizes of
691 *  bytes that needs to be read and written to the LTC3887 digital power controller.
692 *
693 *
694 */
695 void I2C_config(uint32 length, uint32 dir, bool reset_in)
696 {
697
698         if(reset_in == 1)
699         {
700             /* Module in reset. Status bits cleared */
701             i2cREG1->MDR = I2C_RESET_IN;
702         }
703         /* Length of data to be read/transimi */
704         i2cREG1->CNT = length;
705         /*  read in technical documents page 1821*/
706         i2cREG1->MDR = I2C_FREE_RUN | I2C_STOP_COND |I2C_MASTER | dir | I2C_START_COND |
      I2C_RESET_OUT;
707 }
708
709 /** @fn int PMBus_read (uint8 address, uint8 * reg_and_data, uint8 length)
710 *   @brief Function that use SMBUS protocol to read from LTC3887
711 *   @param[in] address 8bits address
712 *   @param[out] *reg_and_data values of the register
713 *   @param[in] length the length of value read it could be 1 byte or 2 bytes
714 *
715 *   @return  returns 0 if function is successful. returns 1 failed( I2C ready flag never
      received)
716 *
717 *   this function reads from send the following data in order
718 *       - Write address(Global or paged)+ register
719 *       - Read from register (1 byte or 2 bytes)
720 *
721 *   Information about SMBus and PMBus
722 *   - SMBus is a two-wire interface that is often used to interconnect a variety of system
      management chips to one or more host systems. It uses I2C with some extensions as the
      physical layer. There is also a protocol layer, which defines classes of data and how
      that data is structured. Both the physical layer and protocol layer add a level of
      robustness not originally embodied in the I2C specification. The SMBus Slave component
      supports most of the SMBus version 2.0 Slave device specifications with numerous
      configurable options.
723 *
724 *   - PMBus is an extension to the more generic SMBus protocol with specific focus on power
      conversion and power management systems. With some slight modifications to the SMBus
      protocol, the PMBus specifies application layer commands, which are not defined in the
      SMBus. The PMBus component presents all possible PMBus revision 1.2 commands and allows
      you to select which commands are relevant to an application.
725 *
726 *
727 */
728 int PMBus_read (uint8 address, uint8 * reg_and_data, uint8 length) {
729
730     int cnt;
731     /*Delay between each operation*/
732     wait(4000);
733
734     /* slave address use 0-6 bits for a 7 bits com*/
735     i2cREG1->SAR =  (uint32_t)(address);
736
737     I2C_config(1,I2C_TRANSMITTER,1);
738     if (ltc_I2C_TxRx_timeout(1))
739             return 1;
740     i2cREG1->DXR = (uint32)reg_and_data[0];
741
742
743
744     if (ltc_I2C_TxRx_timeout(1))
745         return 1;
746
747
```

```
748     I2C_config((uint32)length,I2C_RECEIVER,0);
749     for (cnt=1; cnt < length+1; cnt++) {
750         if (ltc_I2C_TxRx_timeout(2))
751                 return 1;
752         reg_and_data[cnt] = ((uint8)i2cREG1->DRR);
753 
754     }
755 
756 
757     i2cREG1->STR = (uint32)I2C_SCD_INT;
758 
759     return 0;
760 }
761 
762 /** @fn int PMBus_write (uint8 address, uint8 * reg_and_data, uint8 length)
763 *   @brief Function that use write protocol from LTC3887
764 *   @param[in] address 8bits address
765 *   @param[out] *reg_and_data values of the register
766 *   @param[in] length the length of value read it could be 1 byte or 2 bytes
767 *
768 *   @return  returns 0 if function is successful. returns 1 failed( I2C ready flag never
        received)
769 *
770 *   this function reads from send the following data in order
771 *       - Write address(Global or paged)+ register + data
772 *
773 *   Information about SMBus and PMBus
774 *   - SMBus is a two-wire interface that is often used to interconnect a variety of system
        management chips to one or more host systems. It uses I2C with some extensions as the
        physical layer. There is also a protocol layer, which defines classes of data and how
        that data is structured. Both the physical layer and protocol layer add a level of
        robustness not originally embodied in the I2C specification. The SMBus Slave component
        supports most of the SMBus version 2.0 Slave device specifications with numerous
        configurable options.
775 *
776 *   - PMBus is an extension to the more generic SMBus protocol with specific focus on power
        conversion and power management systems. With some slight modifications to the SMBus
        protocol, the PMBus specifies application layer commands, which are not defined in the
        SMBus. The PMBus component presents all possible PMBus revision 1.2 commands and allows
        you to select which commands are relevant to an application.
777 *
778 *
779 */
780 int PMBus_write (uint8 address, uint8 * reg_and_data, uint8 length) {
781 
782     int cnt;
783 
784     while(i2cIsBusBusy(i2cREG1));
785 
786 
787     i2cREG1->SAR =  (uint32_t)(address);
788 
789 
790     I2C_config((uint32)length,I2C_TRANSMITTER,1);
791 
792     for (cnt=0; cnt < length; cnt++) {
793         if (ltc_I2C_TxRx_timeout(1))
794                 return 1;
795            i2cREG1->DXR = (uint32)reg_and_data[cnt];
796     }
797 
798 
799     if (ltc_I2C_TxRx_timeout(1))
800         return 1;
801 
802 
803     i2cREG1->STR = (uint32)I2C_SCD_INT;
804 
805 
806     return 0;
807 }
808 
809 /** @fn int ltc3887_ready (void)
810 *   @brief Function checks if the LTC3887 ready to received a write or read command
811 *
812 *   @return  returns 0 if function is successful. returns 1 failed( I2C ready flag never
        received)
813 *
814 *   This function checks if the LTC3887 is ready by ready buffer. when the buffer value is 68
        it is an indication that LTC3887 is ready to receive a command.
815 *
816 */
817 int ltc3887_ready (void) {
```

```
818
819     uint8 reg_and_data[] = {LTC3887_MFR_COMMON, 0x0};
820     do{
821         if (PMBus_read(LTC3887_GLOBAL_ADDRESS, reg_and_data, 1))
822             return 1;
823      }
824     while((reg_and_data[1] & 0x68) != 0x68);
825
826     return 0;
827 }
828
829 /** @fn int ltc3887_init (void)
830 *   @brief Function i use to write the correct configuration to the ltc3887
831 *   @return  returns 0 if function is successful. returns 1 failed
832 *
833 *  To be able to use this model as intented the systems needs a to write the right
      configurations. This configuration got from ELAB-erlend
834 *  Since he design and configurate an earlier test circuit to fit the need for the our mNLP
      system.
835 *
836 *  The functions reads from table of commands and data ltc_config_table.
837 *
838 */
839 int ltc3887_init (void) {
840
841
842         int cnt=0;
843         uint8 reg_and_data[3];
844         uint16 address;
845
846
847         if (ltc3887_turn_on_off_channel (0, false))
848             return 1;
849
850
851         while (ltc_config_table[cnt].ltc_config_Address != COMPLETE ){
852
853                 switch (ltc_config_table[cnt].ltc_config_Address) {
854
855                         case GLOBAL:
856                             address = LTC3887_GLOBAL_ADDRESS;
857                             break;
858
859                         case PAGED:
860                             address = LTC3887_PAGED_ADDRESS;
861                             break;
862
863                         default:
864                                     return 1;
865
866                 }
867
868
869                 reg_and_data[0] = ltc_config_table[cnt].ltc_Command;
870                 reg_and_data[1] = ltc_config_table[cnt].ltc_Lower_byte;
871                 reg_and_data[2] = ltc_config_table[cnt].ltc_upper_byte;
872
873                 if (ltc3887_ready())
874                     return 1;
875
876                 if (PMBus_write(address, reg_and_data, (ltc_config_table[cnt].ltc_byte_Length
      + 1)))
877                     return 1;
878
879                 cnt++;
880       }
881       return 0;
882     }
883
884 /** @fn int ltc3887_read_config()
885 *   @brief Readback of all the registers. this used to verify that none are effect either
      under debbuging and testing or the system.
886 *   @return  returns 0 if function is successful. returns 1 failed
887 *
888 *   THe functions compare the values that should be read and from the table. Data will be
      send our regardless whether the system has detected an err or not.
889 *   if a different value is read then written it in dicated by sending Err1 or Err2 = upper
      bite (MSB)
890 */
891
892 int ltc3887_read_config()
893 {
894
```

```
895     int cnt = 0;
896     uint8 reg_and_data[3];
897     uint16 address;
898 
899     while (ltc_config_table[cnt].ltc_config_Address != COMPLETE ) {
900 
901         switch (ltc_config_table[cnt].ltc_config_Address) {
902                 case GLOBAL:
903                     if ((ltc_config_table[cnt].ltc_Command == LTC3887_PAGE) && (
    ltc_config_table[cnt].ltc_Lower_byte == 0x01))
904                         return 0;
905 
906                     address = LTC3887_GLOBAL_ADDRESS;
907                     break;
908 
909                 case PAGED:
910                     address = LTC3887_PAGED_ADDRESS;
911                     break;
912 
913                 default:
914                             return 0;
915         }
916 
917         reg_and_data[0] = ltc_config_table[cnt].ltc_Command;
918 
919         //char temp[60];
920 
921 
922         if (ltc3887_ready())
923             return 1;
924 
925         if (PMBus_read(address, reg_and_data, ltc_config_table[cnt].ltc_byte_Length))
926             return 1;
927 
928         if (ltc_config_table[cnt].ltc_byte_Length == 1){
929 
930             if (reg_and_data[1] != ltc_config_table[cnt].ltc_Lower_byte)
931                 if (raw_data)
932                 {
933                     add_to_buffer (reg_and_data[0],reg_and_data[1], CONTINUE,
    Message_TMS_REG);
934                 }
935                 else{
936                     obc_debug("%x, %x Err1",reg_and_data[0], reg_and_data[1]);
937                 }
938             else
939                 if (raw_data)
940                 {
941                     add_to_buffer (reg_and_data[0],reg_and_data[1], CONTINUE,
    Message_TMS_REG);
942                 }
943                 else{
944                     obc_debug("%x, %x",reg_and_data[0], reg_and_data[1]);
945                 }
946 
947         }
948         if (ltc_config_table[cnt].ltc_byte_Length == 2)
949         {
950             if (reg_and_data[2] != ltc_config_table[cnt].ltc_upper_byte)
951                 if (raw_data)
952                 {
953                     add_to_buffer (reg_and_data[0],((reg_and_data[2]<<8) | reg_and_data[1])
    , CONTINUE,Message_TMS_REG);
954                 }
955                 else{
956                     obc_debug("%x, %x, %x err2",reg_and_data[0], reg_and_data[1],
    reg_and_data[2]);
957                 }
958             else
959                 if (raw_data)
960                 {
961                     add_to_buffer (reg_and_data[0],((reg_and_data[2]<<8) | reg_and_data[1])
    , CONTINUE,Message_TMS_REG);
962                 }
963                 else{
964                     obc_debug("%x, %x, %x",reg_and_data[0], reg_and_data[1], reg_and_data
    [2]);
965                 }
966         }
967         //SendStringSCI("\r\n",False);
968         cnt++;
969      }
970     return 0;
```

```
971 }
972 /** @fn int ltc3887_L16u_reg (uint16 data, uint8 * reg_addr)
973 *   @brief this function convert data received to format linear for voltage related commands
974 *   @param[in] data needed to be converted e.g milliVolts
975 *   @param[out] reg_addr valid register value to to write
976 *   @return  returns 0 if function is successful. (todo, detected overflow??)
977 *
978 *   this function is done according to LTC3887 datasheet consolt the table about data format.
979 *   Y = (data * 1/1000 * (2**-12)) where Y is register value
980 *
981 */
982 int ltc3887_L16u_reg (uint16 data, uint8 * reg_addr) {
983
984     /* Y = (Value * 1/1000 * (2**-12)) */
985     /*voltage in mV*/
986
987     float tmp = (float)((data)/0.244140625) ;
988     reg_addr[0] = (uint8)tmp;
989     reg_addr[1] = (uint8)((uint16)tmp >> 8);
990     return 0;
991 }
992
993 /** @fn int ltc3887_L16u_data (uint8 * reg_addr,uint16* data)
994 *   @brief this function convert register received to format from linear to integer
995 *   @param[in] reg_addr valid register value to to write
996 *   @param[out] data needed to be converted
997 *   @return  returns 0 if function is successful. (todo, detected overflow??)
998 *
999 *   this function is done according to LTC3887 datasheet consolt the table about data format.
1000 *   data = (Y * 1000* (2**-12)) the data could voltage in mv
1001 *
1002 */
1003 int ltc3887_L16u_data (uint8 * reg_addr,uint16* data)
1004 {
1005     /* Value = (Y * 1000* (2**-12)) */
1006     /* voltage in mV */
1007
1008     uint16 y = ((reg_addr[1] << 8) | reg_addr[0])& 0xffff;
1009     double tmp;
1010     tmp = ((double)y*1000)/4096.0;
1011     * data =(uint16)tmp;
1012
1013     return 0;
1014 }
1015 /** @fn int ltc3887_L5s_11s_reg (uint16  data,uint8 * reg_addr)
1016 *   @brief this function convert data to L5s_11s_reg format
1017 *   @param[in] data needed to be converted
1018 *   @param[out] reg_addr valid register value to to write
1019
1020 *   @return  returns 0 if function is successful. (todo, detected overflow??)
1021 *
1022 *   this function is done according to LTC3887 datasheet consolt the table about data format.
1023 *   Value = (Y * (2**-N))
1024 *   Y = 2's complement of b[10:0]
1025 *   N = 2's complement of b[15:11]
1026 *   this function is not implemented, this can be useful to read register that holds values
      in format L5s_11s  and convert
1027 *   them to readable format of data.
1028 *
1029 */
1030
1031 int ltc3887_L5s_11s_reg (uint16  data,uint8 * reg_addr)
1032 {
1033     /*
1034     * Value = (Y * (2**-N))
1035     * Y = 2's complement of b[10:0]
1036     * N = 2's complement of b[15:11]
1037     *
1038     * Assuming Y is positive and N is negative
1039     */
1040  return 0;
1041 }
1042 /** @fn int ltc3887_L5s_11s_data (uint8 * reg_addr, uint16 * data)
1043 *   @brief This function is read register of type L5s_11s data format and converts it to a
      readable data
1044 *   @param[in]  reg_addr valid register value to to write
1045 *   @param[out] data needed to be converted e.g milliVolts
1046 *   @return  returns 0 if function is successful. (todo, detected overflow??)
1047 *
1048 *   this function is done according to LTC3887 datasheet consolt the table about data format.
1049 *   data = (Y * (2**-N))
1050 *   Y = 2's complement of b[10:0]
1051 *   N = 2's complement of b[15:11]
```

```
1052 *
1053 *   Assuming Y is positive and N is negative
1054 *
1055 */
1056 int ltc3887_L5s_11s_data (uint8 * reg_addr, uint16 * data) {
1057
1058     /* data = (Y * (2**-N))
1059      * Y = 2's complement of b[10:0]
1060      * N = 2's complement of b[15:11]
1061      *
1062      * Assuming Y is positive and N is negative
1063      */
1064
1065     double tmp;
1066     uint32_t y = ((reg_addr[1] << 8) | reg_addr[0]) & 0x7ff;
1067     uint8 n = (((reg_addr[1] << 8) | reg_addr[0]) >> 11) & 0x1f;
1068
1069     tmp = (1/pow(2.0, (double)(((~n)+1) & 0xf)))*y*1000;
1070     *data = (uint16)tmp;
1071
1072     return 0;
1073 }
1074
1075 /** @fn int ltc3887_turn_on_off_channel (uint8 channel, bool on_off)
1076 *   @brief This function is read register of type L5s_11s data format and converts it to a
      readable data
1077 *   @param[in] channel channel to turn on off currently we are using only one channel[0]
1078 *   @param[in] on_off 1=on
1079 *
1080 *   @return  returns 0 if function is successful.
1081 *
1082
1083 *   this function  important to remamber to turn off the channel before writting new
      configurations. and that power rail for egun is on!.
1084 *
1085 */
1086
1087
1088 int ltc3887_turn_on_off_channel (uint8 channel, bool on_off) {
1089
1090    uint8 data;
1091
1092     if (on_off) {
1093         data = 0x80;
1094     }
1095     else
1096     {
1097         data = 0x0;
1098     }
1099     ltc3887_write_register (LTC3887_OPERATION, 3,3,data);
1100     return 0;
1101 }
1102
1103 /** @fn int ltc3887_set_output_voltage (uint8 channel, uint16 voltage)
1104 *   @brief This function is a setting voltage for channel[0]
1105 *   @param[in] channel  default value is [0]
1106 *   @param[in] voltage in mV to control egun
1107 *
1108 *   @return  returns 0 if function is successful.
1109 *
1110
1111 *   this function varies the potential applied to the egun. Read about this in Tore andre
      bekkeng PHD source ask ketil roed.
1112 *
1113 */
1114
1115 int ltc3887_set_output_voltage (uint8 channel, uint16 voltage) {
1116
1117
1118     if(ltc3887_write_register( LTC3887_VOUT_COMMAND, 3,1, voltage))
1119                 return 1;
1120
1121     return 0;
1122 }
1123
1124 /** @fn int ltc3887_output_current (uint8 channel, uint16 * output_current)
1125 *   @brief Function to read current value applied to channel[0]
1126 *   @param[in] channel default value is [0]
1127 *   @param[out] output_current current value read from ltc3887 on this channel
1128 *
1129 *   @return  returns 0 if function is successful.
1130 *
1131 *
```

```
1132 *
1133 */
1134
1135 int ltc3887_output_current (uint8 channel, uint16 * output_current) {
1136
1137
1138     if(ltc3887_read_register (LTC3887_READ_IOUT, 2,2, output_current))
1139         return 1;
1140
1141     return 0;
1142 }
1143
1144 /** @fn int ltc3887_Read_internal_Temperature(uint8 channel, uint16 * External_Temperature)
1145 *   @brief Function to read current value applied to channel[0]
1146 *   @param[in] channel default value is [0]
1147 *   @param[out] External_Temperature value read from ltc3887 on this channel
1148 *
1149 *   @return  returns 0 if function is successful.
1150 *
1151 * This is can be used to read the operational temperature of the IC. Could be use to measure
      approximate temperature of the system.
1152 */
1153 int ltc3887_Read_internal_Temperature(uint8 channel, uint16 * External_Temperature)
1154  {
1155
1156      if(ltc3887_read_register (LTC3887_READ_TEMPERATURE_2,2,2, External_Temperature))
1157          return 1;
1158
1159      *External_Temperature /= 1000;
1160      *External_Temperature -= 20;
1161      return 0;
1162
1163  }
1164
1165 /** @fn int ltc3887_Read_output_voltage(uint8 channel, uint16 * voltage)
1166 *   @brief Function to read current value applied to channel[0]
1167 *   @param[in] channel default value is [0]
1168 *   @param[out] voltage value read from ltc3887 on this channel
1169 *
1170 *   @return  returns 0 if function is successful.
1171 *
1172 *  this a verification method to read that value set is valid.
1173 */
1174
1175 int ltc3887_Read_output_voltage(uint8 channel, uint16 * voltage)
1176  {
1177
1178
1179     if(ltc3887_read_register (LTC3887_READ_VOUT, 2,1, voltage))
1180         return 1;
1181      return 0;
1182
1183  }
1184
1185 /** @fn int ltc3887_read_register (uint8 reg_command, uint8 lenght,uint8 dataformat, uint16*
      read_data)
1186 *   @brief Function to read current value applied to channel[0]
1187 *   @param[in] reg_command default value is [0]
1188 *   @param[in] lenght byte length of read value.
1189 *   @param[in] dataformat L16U_data or L5s_11s_data other can be added
1190 *   @param[out] read_data
1191 *
1192 *   @return  returns 0 if function is successful.
1193 *
1194 *  This functions made to meet the needs of PMBus and format the read value to readable
      integer.
1195 */
1196
1197 int ltc3887_read_register (uint8 reg_command, uint8 lenght,uint8 dataformat, uint16*
      read_data) {
1198     #define L16U_data 1
1199     #define L5s_11s_data 2
1200     #define channel 0
1201
1202     uint8 reg_and_data[3];
1203
1204     if (ltc3887_ready())
1205         return 1;
1206     reg_and_data[0] = LTC3887_PAGE;
1207     reg_and_data[1] = channel;
1208
1209     if (PMBus_write(LTC3887_GLOBAL_ADDRESS, reg_and_data, lenght))
1210             return 1;
```

```
1211
1212     if (ltc3887_ready())
1213              return 1;
1214
1215     reg_and_data[0] = reg_command;
1216
1217     if (PMBus_read(LTC3887_GLOBAL_ADDRESS, reg_and_data, lenght))
1218         return 1;
1219
1220     switch(dataformat)
1221     {
1222
1223         case L16U_data:
1224             if (ltc3887_L16u_data(&reg_and_data[1], read_data))
1225                         return 1;
1226             break;
1227         case L5s_11s_data:
1228             if (ltc3887_L5s_11s_data(&reg_and_data[1], read_data))
1229                         return 1;
1230             break;
1231
1232     }
1233
1234     return 0;
1235 }
1236
1237
1238 /** @fn int ltc3887_write_register (uint8 reg_command, uint8 length,uint8 dataformat, uint16
     data)
1239 *    @brief Function to read current value applied to channel[0]
1240 *    @param[in] reg_command default value is [0]
1241 *    @param[in] length byte length of read value.
1242 *    @param[in] dataformat L16U_data or L5s_11s_data other can be added
1243 *    @param[out] data
1244 *
1245 *    @return  returns 0 if function is successful.
1246 *
1247 *   This functions made to meet the needs of PMBus and format the read value to readable
     integer.
1248 */
1249
1250 int ltc3887_write_register (uint8 reg_command, uint8 length,uint8 dataformat, uint16 data)
1251 {
1252
1253     #define L16U_reg 1
1254     #define L5s_11s_reg 2
1255     #define reg_addr 3
1256     #define channel 0
1257
1258     uint8 reg_and_data[3];
1259
1260     if (ltc3887_ready())
1261         return 1;
1262
1263     reg_and_data[0] = LTC3887_PAGE;
1264     reg_and_data[1] = channel;
1265
1266     if (PMBus_write(LTC3887_GLOBAL_ADDRESS, reg_and_data, length-1))
1267         return 1;
1268
1269     if (ltc3887_ready())
1270         return 1;
1271     reg_and_data[0] = reg_command;
1272
1273     switch(dataformat)
1274        {
1275
1276             case L16U_reg:
1277                 if (ltc3887_L16u_reg(data, &reg_and_data[1]))
1278                             return 0;
1279                 break;
1280             case L5s_11s_reg:
1281                 if (ltc3887_L5s_11s_reg(data,&reg_and_data[1]))
1282                             return 0;
1283                 break;
1284             case reg_addr:
1285                   reg_and_data[1] = (uint8)data;
1286                   length = length-1;
1287                 break;
1288
1289        }
1290
1291     if (PMBus_write(LTC3887_PAGED_ADDRESS, reg_and_data, length))
```

```
1292             return 1;
1293
1294
1295     return 0;
1296 }
1297
1298 /** @fn void cmd_ltc3887_turn_on_off_channel(int arg1)
1299 *   @brief This function  cmd check for validation of value
1300 *   @param[in] arg1 channel to be set
1301 *
1302 */
1303
1304 void cmd_ltc3887_turn_on_off_channel(int arg1)
1305 {
1306     if( arg1>1)
1307     {
1308         SciSendACK(2);
1309         //SciTooManyargs();
1310     }
1311     ltc3887_turn_on_off_channel(ltc_ch0,arg1);
1312 }
1313
1314 /** @fn void cmd_ltc3887_Read_output_voltage(int arg1,int arg2)
1315 *   @brief This function  cmd check for validation of value
1316 *   @param[in] arg1 channel to be set
1317 *   @param[in] arg2 channel to be set
1318 */
1319
1320 void cmd_ltc3887_Read_output_voltage(int arg1,int arg2)
1321 {
1322
1323     if(arg1 != 0 || arg2 != 0)
1324     {
1325           SciSendACK(2);
1326     }
1327     ltc3887_Read_output_voltage(0,&(ltc_m_values.ltc_voltage));
1328     if(!raw_data)
1329         obc_debug("LTC Voltage(mV),%d",ltc_m_values.ltc_voltage);
1330 }
1331
1332 /** @fn void cmd_ltc3887_Read_internal_Temperature(int arg1,int arg2)
1333 *   @brief This function  cmd check for validation of value
1334 *   @param[in] arg1 to detect fault args
1335 *   @param[in] arg2 detect fault command
1336 *
1337 *   This args are used just to void wrong arg inputs
1338 */
1339
1340 void cmd_ltc3887_Read_internal_Temperature(int arg1,int arg2)
1341 {
1342
1343     if(arg1 != 0 || arg2 != 0)
1344     {
1345             SciSendACK(2);
1346     }
1347     ltc3887_Read_internal_Temperature(0, &(ltc_m_values.ltc_temperature));
1348     if(!raw_data){
1349         obc_debug("LTC temperature(C),%d",ltc_m_values.ltc_temperature);
1350     }
1351 }
1352 /** @fn void cmd_ltc3887_output_current(int arg1,int arg2)
1353 *   @brief This function  cmd check for validation of value
1354 *   @param[in] arg1 to detect fault args
1355 *   @param[in] arg2 detect fault command
1356 *
1357 *   This args are used just to void wrong arg inputs
1358 */
1359 void cmd_ltc3887_output_current(int arg1, int arg2)
1360 {
1361
1362     if(arg1 != 0 || arg2 != 0)
1363     {
1364         SciSendACK(2);
1365     }
1366     ltc3887_output_current(0,&(ltc_m_values.ltc_current));
1367     if(!raw_data)
1368         obc_debug("LTC current(mA),%d",ltc_m_values.ltc_current);
1369 }
1370
1371 /** @fn void cmd_ltc3887_set_output_voltage(int arg1,int arg2)
1372 *   @brief This function  cmd check for validation of value
1373 *   @param[in] arg1 to detect fault args
1374 *   @param[in] arg2 detect fault command
```

```
1375 *
1376 *   This args are used just to void wrong arg inputs
1377 */
1378
1379
1380 void cmd_ltc3887_set_output_voltage(int arg1, int arg2)
1381 {
1382     if(arg1< 0 || arg1>3000 )
1383     {
1384        SciSendACK(2);
1385     }
1386     ltc3887_set_output_voltage(0,arg1);
1387 }
1388
1389 /** @fn void cmd_ltc3887_read_config(int arg1, int arg2)
1390 *   @brief This function  cmd check for validation of value
1391 *   @param[in] arg1 to detect fault args
1392 *   @param[in] arg2 detect fault command
1393 *
1394 *   This args are used just to void wrong arg inputs
1395 */
1396 void cmd_ltc3887_read_config()
1397 {
1398      ltc3887_read_config();
1399 }
1400
1401
1402 /*================================== AD7768 External ADC ==================================*/
1403
1404 /** @fn int system_get_sys_MCLK()
1405 *   @brief Function to read current value applied to channel[0]
1406 *
1407 *   @return  returns  Eclk value in MHz.
1408 *
1409 * This function gets the value of MCLK to ADC based on source clk used, division and clock
       rate. If the user chooses to
1410 *  change values in halcogen this has to be up to date.
1411 */
1412
1413 int system_get_sys_MCLK()
1414 {
1415     int clk;
1416     int eclk;
1417     //sources set to external osc 16MHz
1418     if(((systemREG1->ECPCNTL)& 0x00080000) == 0)
1419     {
1420         clk = (int)OSC_FREQ ;
1421
1422     }
1423     //source set to VCLK source
1424     else if(((systemREG1->ECPCNTL)& 0x00080000) == 1)
1425     {
1426         clk = (int)GCLK_FREQ;
1427     }
1428
1429     eclk = clk/(((systemREG1->ECPCNTL)&0x0000FFFF)+1);
1430     return eclk;
1431 }
1432
1433 ad7768_settings_t ad7768;
1434
1435 /** @fn int ad7768_init (void)
1436 *   @brief init configuartion to the AD7768
1437 *
1438 *   @return  returns 0 when success, -1 when wrong value of ECLK is detected.
1439 *
1440 */
1441
1442 int ad7768_init (void) {
1443
1444
1445     ad7768.MCLK = MCLK_FREQ;
1446
1447     if(MCLK_FREQ==8){
1448         ad7768.DCLK_DIV = 4;
1449         ad7768_set_DCLK_div(ad7768.DCLK_DIV);
1450
1451         ad7768.MCLK_DIV= 8;
1452         ad7768_set_MCLK_div(ad7768.MCLK_DIV);
1453
1454         ad7768.decRate = 64;
1455         ad7768_set_Drate(ad7768.decRate);
1456
```

```
1457         ad7768.filter = ad7768_filter_wide;
1458         ad7768_set_Filter(ad7768.filter);
1459 
1460     }
1461     else{
1462          return -1;
1463     }
1464 
1465 
1466     ad7768.fMOD = ad7768_calc_fMOD();
1467     ad7768.DCLK = ad7768_calc_DCLK();
1468     ad7768.ODR = ad7768_calc_ODR();
1469 
1470     ad7768_sync();
1471 
1472     high_speed_transfer_unit_init();
1473 
1474     return 0;
1475 }
1476 
1477 
1478 
1479 /** @fn int ad7768_spi_read(uint8 regaddr, uint8 *read_value)
1480 *   @brief this function uses SPI interface to read registers
1481 *   @param[in] regaddr address to the register
1482 *   @param[out] read_value value read from register
1483 *
1484 *   @return  returns 0 when success, -1 for spi timeout and 2 for illegal command respons.
1485 *
1486 *   This function uses the SPI, AD7768 uses an off frame protocol which means two 16 frames
     are sent
1487 *   1st frame for read is address then followed with address and read
1488 *
1489 */
1490 
1491 int ad7768_spi_read(uint8 regaddr, uint8 *read_value)
1492 {
1493     /*Return value from SPI5 function*/
1494     uint16 buffer[2];
1495     /*configure SPI5*/
1496     spiDAT1_t spi5_ctrl_dataconfig_t;
1497     spi5_ctrl_dataconfig_t.CS_HOLD = FALSE;
1498     spi5_ctrl_dataconfig_t.WDEL    = TRUE;
1499     spi5_ctrl_dataconfig_t.DFSEL   = SPI_FMT_0;
1500     spi5_ctrl_dataconfig_t.CSNR    = 0xFE;
1501 
1502     /*Tx buffer[0] bit 15 is set for read*/
1503     buffer[0] = 0x8000 | (regaddr & 0x7F) << 8;
1504     /*Rx buffer[1] empty*/
1505     buffer[1] = 0x0;
1506 
1507     /*AD7768 uses an off frame protocol which means two 16 frames are sent
1508      * 1st frame for read is address then followed with address and read*/
1509     /*Transmitt address to read from*/
1510     if(spiTransmitData(spiREG5, &spi5_ctrl_dataconfig_t, 1, (uint16 *)&buffer[0]) !=0U)
1511         return -1;
1512     if(spiTransmitAndReceiveData(spiREG5, &spi5_ctrl_dataconfig_t, 1, (uint16 *)&buffer[0], (
     uint16 *)&buffer[1]) !=0U)
1513         return -1;
1514 
1515     /*Check if an illegal command*/
1516     if (buffer[1]== ad7768_Illegal_Command) {
1517         return 2;
1518     }
1519 
1520     *read_value = buffer[1] & 0xFF;
1521     /*success*/
1522     return 0;
1523 
1524 }
1525 
1526 /** @fn int ad7768_spi_write(uint8 regaddr, uint8 write_value)
1527 *   @brief this function uses SPI interface to write registers
1528 *   @param[in] regaddr address to the register
1529 *   @param[in] write_value value to be written to the register
1530 *
1531 *   @return  returns 0 when success, -1 for spi timeout and 2 for illegal command respons.
1532 *
1533 *   This function uses the SPI, AD7768 uses an off frame protocol which means two 16 frames
     are sent
1534 *   1st frame for read is address then followed with address and read
1535 *
1536 */
```

```
1537 int ad7768_spi_write(uint8 regaddr, uint8 write_value)
1538 {
1539
1540     /*Buffer[0] for Tx and Buffer[1] for rx*/
1541     uint16 buffer[2];
1542     /*Configurate SPI5*/
1543     spiDAT1_t spi5_ctrl_dataconfig_t;
1544     spi5_ctrl_dataconfig_t.CS_HOLD = FALSE;
1545     spi5_ctrl_dataconfig_t.WDEL    = TRUE;
1546     spi5_ctrl_dataconfig_t.DFSEL   = SPI_FMT_0;
1547     spi5_ctrl_dataconfig_t.CSNR    = 0xFE;
1548
1549     /*Writ command as specifised in datasheet*/
1550     buffer[0] = (regaddr & 0x7F) << 8 | write_value;
1551
1552     if(spiTransmitData(spiREG5, &spi5_ctrl_dataconfig_t, 1, (uint16 *)&buffer[0]) !=0U)
1553             return -1;
1554     if(spiTransmitAndReceiveData(spiREG5, &spi5_ctrl_dataconfig_t, 1, (uint16 *)&buffer[0], (
      uint16 *)&buffer[1]) !=0U)
1555             return -1;
1556     /*Check if an illegal command*/
1557     if (buffer[1]== ad7768_Illegal_Command){
1558             return 2;
1559     }
1560     /*success */
1561     return 0;
1562
1563 }
1564 /** @fn int ad7768_set_DCLK_div(uint8 arg)
1565 *   @brief  this functions sets value for DCLK div
1566 *   @param[in] arg value of DCLK div <2-4-8>
1567 *   @return  returns 0 when success, -1 for readback value does not match written value
1568 *
1569 *
1570 *       DCLK divider. These bits control division of the DCLK clock used to clock out
      conversion data on the DOUTx pins.
1571 *       The DCLK signal is derived from the MCLK applied to the AD7768. The DCLK divide mode
      allows the user to optimize the
1572 *      DCLK output to fit the application. Optimizing the DCLK per application depends on the
       requirements of the user.
1573 *      When the AD7768 are using the highest capacity output on the fewest DOUTx pins, for
      example, running in decimate
1574 *       by 32 using the DOUT0 and DOUT1 pins, the DCLK must equal the MCLK; thus, in this
      case, choosing the no division
1575 *       setting is the only way the user can output all the data within the conversion period
      . There are other cases,
1576 *       however, when the ADC may be running in fast mode with high decimation rates,
1577 *       or in median or eco mode where the DCLK does not need to run at the same speed as
      MCLK. In these cases,
1578 *        the DCLK divide allows the user to reduce the clock speed and makes routing and
      isolating such signals easier
1579 *            - 00 divide by args = 8
1580 *            - 01 divide by args = 4
1581 *            - 10 divide by args = 2
1582 *            - 11 No division
1583 *            - bit shift to the right MCLK_freq >> arguments
1584 *
1585 */
1586 int ad7768_set_DCLK_div(uint8 arg) {
1587
1588
1589     uint8 reg = AD7768_Interface_configuration;
1590     uint8 read_reg;
1591     uint8 write_reg;
1592
1593
1594    switch(arg)
1595    {
1596        case 8:
1597            write_reg = 0x00;
1598            break;
1599        case 4:
1600            write_reg = 0x01;
1601            break;
1602        case 2:
1603            write_reg = 0x10;
1604            break;
1605        default:
1606            return 1;
1607
1608    }
1609     ad7768_spi_write(reg, write_reg);
1610     ad7768_spi_read(reg, &read_reg);
```

```
1611
1612        if(read_reg == write_reg){
1613            ad7768.DCLK_DIV = arg;
1614            ad7768_sync();
1615        }
1616        else{
1617            return -1;
1618        }
1619
1620        return 0;
1621
1622    }
1623
1624    /** @fn int ad7768_set_MCLK_div(uint8_t arg)
1625    *   @brief  this functions sets value for MCLK div
1626    *   @param[in] arg value of MCLK div <2-4-8>
1627    *   @return  returns 0 when success, -1 for readback value does not match written value
1628    *
1629    *
1630    *       bit[5:4] power_mode this sets currents modes used by ADC does not effect MCLK_div
1631    *      - when sett to 00 eco
1632    *      - 10 median
1633    *      - 11 fast
1634    *
1635    *        bit[1:0] MCLK_DIV controls div ration between ECLK and clock used in each adc module
         .
1636    *             - 00 MCLK/32 with a base MCLK of 32.768 MHZ set to eco mode
1637    *             - 10 MCLK/8  set to median mode
1638    *             - 11 MCLK/4 for fast mode
1639    *
1640    */
1641    int ad7768_set_MCLK_div(uint8_t arg) {
1642
1643        uint8 reg = AD7768_POWER_MODE;
1644        uint8 read_reg;
1645        uint8 write_reg;
1646
1647        switch(arg)
1648        {
1649            case 4:
1650                write_reg = 0x33;
1651                break;
1652            case 8:
1653                write_reg = 0x22;
1654                break;
1655            case 32:
1656                write_reg = 0x00;
1657                break;
1658            default:
1659                /*invalid argument*/
1660                return 1;
1661
1662        }
1663
1664
1665        ad7768_spi_write(reg, write_reg);
1666
1667        ad7768_spi_read(reg, &read_reg);
1668
1669        if(read_reg == write_reg){
1670            ad7768.MCLK_DIV = arg;
1671            ad7768_sync();
1672        }
1673        else{
1674            return -1;
1675        }
1676
1677        return 0;
1678
1679    }
1680
1681    /** @fn int ad7768_set_Drate(uint16 arg)
1682    *   @brief  this functions sets decimation rate
1683    *   @param[in] arg decimation rates <32-64-128-256-512-1024> Kilo samples per second
1684    *   @return  returns 0 when success, -1 for readback value does not match written value
1685    *
1686    *   The ADC provides the user with two channels Mode A and B. Using the channel mode select
         register (Register 0x03),
1687    *    the user can assign each channel to either Channel Mode A or Channel Mode B,
1688    *    which maps that mode to the required ADC channels. These modes allow different filter
         types and decimation
1689    *    rates to be selected and mapped to any of the ADC channels. In this we are using just
         one rate for all channels.
```

```
1690 */
1691
1692 int ad7768_set_Drate(uint16 arg) {
1693
1694
1695     uint8 reg = AD7768_Channel_Mode_A;    //Config channel mode A register address
1696
1697
1698     uint8 read_reg;
1699     uint8 write_reg;
1700     //Read in current value of config register to get filter setting.;
1701     ad7768_spi_read(reg, &read_reg);
1702
1703     switch(arg)
1704     {
1705         case 32:
1706             write_reg = (read_reg & 0x08) | 0x00;
1707             break;
1708         case 64:
1709             write_reg = (read_reg & 0x08) | 0x01;
1710             break;
1711         case 128:
1712             write_reg = (read_reg & 0x08) | 0x02;
1713             break;
1714         case 256:
1715             write_reg = (read_reg & 0x08) | 0x03;
1716             break;
1717         case 512:
1718             write_reg = (read_reg & 0x08) | 0x04;
1719             break;
1720         case 1024:
1721             write_reg = (read_reg & 0x08) | 0x05;
1722             break;
1723         default:
1724             return 1;
1725     }
1726
1727
1728     ad7768_spi_write(reg, write_reg);
1729
1730     ad7768_spi_read(reg, &read_reg);
1731
1732     if(read_reg == write_reg){
1733         ad7768.decRate = arg;
1734         ad7768_sync();
1735     }
1736     else{
1737         return -1;
1738     }
1739
1740     return 0;
1741 }
1742 /** @fn int ad7768_set_Filter(uint16 arg)
1743 *   @brief  this functions sets filter type
1744 *   @param[in] arg should be set to either ad7768_filter_SINC=1 or ad7768_filter_wide=0
1745 *   @return  returns 0 when success, -1 for readback value does not match written value
1746 *
1747 *   The ADC provides the user with two channels Mode A and B. Using the channel mode select
     register (Register 0x03),
1748 *    the user can assign each channel to either Channel Mode A or Channel Mode B,
1749 *    which maps that mode to the required ADC channels. These modes allow different filter
     types and decimation
1750 *    rates to be selected and mapped to any of the ADC channels. In this we are using just
     one rate for all channels.
1751 */
1752 int ad7768_set_Filter(uint8 arg) {
1753
1754         uint8 reg = AD7768_Channel_Mode_A;
1755
1756         uint8 read_reg;
1757         uint8 write_reg;
1758
1759         /*Read Drate setting and store it since they share same register*/
1760         ad7768_spi_read(reg, &read_reg);
1761         read_reg = read_reg & 0x07;
1762
1763         write_reg = (arg <<3) | read_reg;
1764
1765         ad7768_spi_write(reg, write_reg);
1766
1767         ad7768_spi_read(reg, &read_reg);
1768
1769         if(read_reg == write_reg){
```

```
1770                ad7768.filter = arg;
1771                ad7768_sync();
1772            }
1773            else{
1774                return -1;
1775            }
1776
1777            return 0;
1778    }
1779    /** @fn void ad7768_print_settings()
1780    *   @brief  Function prints values of MCK_div, DCLK_div, fMOD, decRate, and filter type used.
1781    *
1782    *   function prints values configure the AD7768.
1783    *       - fmod Modulator frequency The internal modulator frequency (fMOD)
1784    *       - decRate Decimation rate data rate from each channel in kSPS
1785    *       - MCLK_div master clock division that ad7768 is operation with
1786    *       - DCLK_div data clock use to sample on all channels
1787    *       - filter in all probs
1788    */
1789    void ad7768_print_settings(){
1790        if(raw_data)
1791        {
1792
1793        }
1794        else{
1795            obc_debug("MCLK_DIV,%d,fMOD(MHz),%d,DecRate,%d,ODR(SPS),%d,DCLK_DIV,%d,DCLK(MHz),%d,
         Filter,%d",
1796                ad7768.MCLK_DIV, ad7768_calc_fMOD(), ad7768.decRate, ad7768_calc_ODR(), ad7768.
         DCLK_DIV, ad7768_calc_DCLK(), ad7768.filter);
1797        }
1798    }
1799    /** @fn int ad7768_calc_DCLK (void)
1800    *   @brief  Function calculates the data clock from MCLK and DCLK_DIV
1801    *
1802    *   the function calculate the data clock rate.
1803    *
1804    */
1805    int ad7768_calc_DCLK (void) {
1806        /*DCLK = MCLK/DCLK_DIV*/
1807        return ad7768.MCLK/ad7768.DCLK_DIV;
1808    }
1809    /** @fn int ad7768_calc_fMOD (void)
1810    *   @brief  Function calculates the function modular frequency
1811    *   @return returns Fmod value
1812    *
1813    *   the function calculate fmod
1814    *
1815    *   that is used by each of the ADCs in the AD7768/AD7768-4 is derived from the externally
         applied MCLK signal.
1816    *   The MCLK division bits allow the user to control the ratio between the MCLK frequency and
          the internal modulator
1817    *   clock frequency. This control allows the user to select the division ratio that is best
         for their configuration.
1818    *   The appropriate clock configuration depends on the power mode, the decimation rate,
1819    *   and the base MCLK frequency available in the system. The relationship between the input
         signal and the modulator
1820    *   frequency is expressed in a normalized manner as a ratio of the input signal (fIN) to the
          modulator frequency (fMOD).
1821    *   This data demonstrates the ADC frequency response relative to out of band tones when
         using the wideband filter.
1822    *   The input frequency (fIN) is swept from dc to 20 MHz. In fast mode, using an 8.192MHz
         fMOD frequency,
1823    *   the x-axis spans ratios of fIN/fMOD from 0 to 2.44 (equivalent to fIN of 0 Hz to 20 MHz).
1824    *   A similar characteristic occurs in median and eco modes.
1825    *
1826    */
1827    int ad7768_calc_fMOD (void) {
1828        /*fMOD = MCLK/MCLK_DIV*/
1829        return ad7768.MCLK/ad7768.MCLK_DIV;
1830    }
1831
1832    /** @fn  int ad7768_calc_ODR(void)
1833    *   @brief  Function calculates the output data rate ODR
1834    *   @return returns ODR value
1835    *
1836    *        This function calculate the output data rate.
1837    *           Thus, for this example, where MCLK = 32.768 MHz, ODR = (32.768 MHz/32)/64 = 16
         kHz
1838    *           Minimizing the DCLK frequency means selecting DCLK =
1839    *           MCLK/8, which results in a 4 MHz DCLK signal. The period of DCLK in this case is
         1/4 MHz = 250 ns.
1840    *           The data conversion on each DOUTx pin is 32 bits long. The conversion data takes
1841    *           32 x250 ns = 8 us to be output. All 32 bits must be output within the ODR period
```

```
     of 1/16 kHz,
1842 *          which is approximately 64 us. In this case, the 8 us required to read out the
     conversion data is well within the
1843 *          64 us  between conversion outputs.
1844 *
1845 */
1846  int ad7768_calc_ODR (void) {
1847     return ((ad7768.MCLK/ad7768.MCLK_DIV)*1000*1000)/ad7768.decRate;}
1848
1849  /** @fn  int ad7768_sync()
1850  *   @brief  function to apply the new configures to AD7768
1851  *   @return 0 for sucess
1852  *
1853  *
1854  *    this function toggles GPIO pin to signal that AD will apply the new configurations.
     The default of this pin is active higth to detect a sync signal the signal should put low
      for a short period then to active hight again.
1855  *    SPI control offers the superset of flexibility and diagnostics to the user. The
     following sections highlight the functionality and
1856  *    diagnostics offered when SPI control is used. After any change to these configuration
     register settings, the user must provide a sync signal to the AD7768/AD7768-4
1857  *   through either the SPI_SYNC command, or by applying the
1858  *   appropriate pulse to the START pin or SYNC_IN pin to ensure
1859  *   that the configuration changes are applied correctly to the ADC and digital filters.
1860  *
1861  *
1862  */
1863 int ad7768_sync() {
1864     /*Toggel signal twice for start conversion*/
1865     /* Can1 TX -> 1 - 0 - 1*/
1866
1867     canREG1->TIOC = ((canREG1->TIOC & 0xFFFFFFFDU) | (0 << 1U));
1868     canREG1->TIOC = ((canREG1->TIOC & 0xFFFFFFFDU) | (1 << 1U));
1869
1870     return 0;
1871
1872 }
1873 /** @fn  int ad7768_reset()
1874 *   @brief  function to apply the new configures to AD7768
1875 *   @return 0 for success
1876 *
1877 *   After a power-on or reset, the AD7768/AD7768-4 default
1878 *   configuration is set to the following low current consumption settings:
1879 *   - Eco power mode with fMOD = MCLK/32.
1880 *   - Interface configuration of DCLK = MCLK/8, header output enabled, and CRC disabled.
1881 *   - Filter configuration of Channel Mode A and Channel Mode B is set to sinc5 and
     decimation = x1024.
1882 *   - Channel mode select is set to 0x00, and all channels are assigned to Channel Mode A.
1883 *   - The analog input precharge buffers are enabled and the reference precharge buffers are
     disabled on all channels.
1884 *   - The offset, gain, and phase calibration are set to the zero position.
1885 *   - Continuous conversion mode is enabled.
1886 *
1887 *
1888 */
1889 int ad7768_reset() {
1890     /*Toggle signal twice for reset reset configs of AD7768*/
1891     /* Can1 RX -> 1 - 0 - 1*/
1892
1893     canREG1->RIOC = ((canREG1->RIOC & 0xFFFFFFFDU) | (0 << 1U));
1894     canREG1->RIOC = ((canREG1->RIOC & 0xFFFFFFFDU) | (1 << 1U));
1895
1896     return 0;
1897
1898 }
1899
1900 /** @fn  void cmd_AD7768_GET_REG(uint8 arg)
1901 *   @brief  function used to readback the values of from single register from CMD.
1902 *   @param[in] arg
1903 *
1904 *   function used by commandline file to check of validity of arguments passed.
1905 *
1906 *
1907 */
1908 void cmd_AD7768_GET_REG(uint8 arg)
1909 {
1910     uint8 spi_rx;
1911     uint32 return_value;
1912
1913     if(arg == 0){
1914         SciSendACK(2);
1915     }
1916     return_value = ad7768_spi_read(arg, &spi_rx);
```

```
1917
1918        obc_debug("value %x rtr %u",spi_rx,return_value);
1919    }
1920
1921
1922    /** @fn  void cmd_AD7768_SET_REG(uint8 reg,uint8 value)
1923    *   @brief  function used to set register value from commandline
1924    *   @param[in] reg
1925    *   @param[in] value
1926    *
1927    *   this function writes value from cmd to register this used for debugging
1928    *
1929    *
1930    */
1931    void cmd_AD7768_SET_REG(uint8 reg,uint8 value)
1932    {
1933
1934        if(reg == 0){
1935            SciSendACK(2);
1936        }
1937        ad7768_spi_write(reg, value);
1938    }
1939    /**
1940    *   @brief  table ad7768_config used to read the values from all register this used for
         debugging
1941    *
1942    *   this function writes value from read all values in register for debugging
1943    *
1944    *
1945    */
1946    ad7768_config const AD7768_config_table[] = {
1947       {AD7768_Channel_Mode_A},
1948       {AD7768_Channel_Mode_B},
1949       {AD7768_Channel_mode_select},
1950       {AD7768_POWER_MODE},
1951       {AD7768_General_configuration},
1952       {AD7768_Data_control},
1953       {AD7768_Interface_configuration},
1954       {AD7768_BIST_control},
1955       {AD7768_Device_status},
1956       {AD7768_Revision_ID},
1957       {AD7768_GPIO_control},
1958       {AD7768_GPIO_write_data},
1959       {AD7768_GPIO_read_data},
1960       {AD7768_Precharge_Buffer_1},
1961       {AD7768_Precharge_Buffer_2},
1962       {AD7768_Positive_reference_precharge_buffer},
1963       {AD7768_Negative_reference_precharge_buffer},
1964       {AD7768_Channel_0_offset_MSB},
1965       {AD7768_Channel_0_offset_MID},
1966       {AD7768_Channel_0_offset_LSB},
1967       {AD7768_Channel_1_offset_MSB},
1968       {AD7768_Channel_1_offset_MID},
1969       {AD7768_Channel_1_offset_LSB},
1970       {AD7768_Channel_2_offset_MSB},
1971       {AD7768_Channel_2_offset_MID},
1972       {AD7768_Channel_2_offset_LSB},
1973       {AD7768_Channel_3_offset_MSB},
1974       {AD7768_Channel_3_offset_MID},
1975       {AD7768_Channel_3_offset_LSB},
1976       {AD7768_Channel_0_gain_MSB},
1977       {AD7768_Channel_0_gain_MID},
1978       {AD7768_Channel_0_gain_LSB},
1979       {AD7768_Channel_1_gain_MSB},
1980       {AD7768_Channel_1_gain_MID},
1981       {AD7768_Channel_1_gain_LSB},
1982       {AD7768_Channel_2_gain_MSB},
1983       {AD7768_Channel_2_gain_MID},
1984       {AD7768_Channel_2_gain_LSB},
1985       {AD7768_Channel_3_gain_MSB},
1986       {AD7768_Channel_3_gain_MID},
1987       {AD7768_Channel_3_gain_LSB},
1988       {AD7768_Channel_0_sync_offset},
1989       {AD7768_Channel_1_sync_offset},
1990       {AD7768_Channel_2_sync_offset},
1991       {AD7768_Channel_3_sync_offset},
1992       {AD7768_Diagnostic_Rx},
1993       {AD7768_Diagnostic_mux_control},
1994       {AD7768_Modulator_delay_control},
1995       {AD7768_Chop_control},
1996       {570}
1997    };
1998
```

```
1999 /** @fn  void cmd_AD7768_GET_AllConfigs(void)
2000 *   @brief  function is used to read all configuration registers
2001 *
2002 *   all redable registers are read when calling this function.
2003 *
2004 *
2005 */
2006 void cmd_AD7768_GET_AllConfigs(void)
2007 {
2008
2009     uint8 spi_rx;
2010     int cnt = 0;
2011     while(AD7768_config_table[cnt].ad7768_config_Address != 570)
2012     {
2013         ad7768_spi_read(AD7768_config_table[cnt].ad7768_config_Address, &spi_rx);
2014         if(raw_data)
2015         {
2016             add_to_buffer (AD7768_config_table[cnt].ad7768_config_Address,spi_rx, CONTINUE,
     Message_TMS_REG);
2017
2018         }else{
2019                 obc_debug("%x %x",AD7768_config_table[cnt].ad7768_config_Address,spi_rx);
2020         }
2021         cnt++;
2022     }
2023
2024
2025
2026 }
2027
2028 /** @fn  void cmd_AD7768_SET_FILTER(uint8 arg)
2029 *   @brief  function is used to set function filter
2030 *
2031 *  function is used to set value for filter from cmd
2032 *
2033 *
2034 *
2035 */
2036 void cmd_AD7768_SET_FILTER(uint8 arg)
2037 {
2038     if(arg == 1 || arg == 0)
2039     {
2040         ad7768_set_Filter(arg);
2041     }
2042 }
2043
2044 /** @fn  void cmd_AD7768_SET_FILTER(uint8 arg)
2045 *   @brief  function is used to set function drate
2046 *
2047 *  function is used to set value for drate
2048 *
2049 *
2050 *
2051 */
2052 void cmd_AD7768_SET_DRATE(uint16 arg)
2053 {
2054     if (arg == 32 || arg == 64 || arg == 128 || arg == 256 || arg == 1024)
2055     {
2056         ad7768_set_Drate(arg);
2057     }
2058
2059     else
2060         obc_debug("Error-argument");
2061
2062 }
2063
2064
2065 /** @fn  void cmd_AD7768_SET_MCLK_div(uint8 arg)
2066 *   @brief  function is used to set MCLK_div from cmd
2067 *
2068 *  function is used to set value for MCLK_div from cmd. used for debugging
2069 *
2070 *
2071 *
2072 */
2073 void cmd_AD7768_SET_DCLK_div(uint8 arg)
2074 {
2075     if(arg==8 || arg==4 || arg ==2) {
2076         ad7768_set_DCLK_div(arg);
2077     }
2078     else
2079         obc_debug("Error-argument");
2080 }
```

```
2081 /** @fn  void cmd_AD7768_SET_MCLK_div(uint8 arg)
2082 *   @brief  function is used to set MCLK_div from cmd
2083 *
2084 *  function is used to set value for MCLK_div from cmd. used for debugging
2085 *
2086 *
2087 *
2088 */
2089 void cmd_AD7768_SET_MCLK_div(uint8 arg)
2090 {
2091     if(arg==32 || arg==8 || arg ==4) {
2092             ad7768_set_MCLK_div(arg);
2093     }
2094     else
2095         obc_debug("Error-argument");
2096 }
2097 /** @fn  void cmd_AD7768_run(void)
2098 *   @brief  function is used to sync system
2099 *
2100 *  function is used to sync from cmd. used for debugging
2101 *
2102 *
2103 *
2104 */
2105 void cmd_AD7768_run(void)
2106 {
2107     ad7768_sync();
2108 }
2109 /** @fn  void cmd_AD7768_stop(void)
2110 *   @brief  function is used to stop
2111 *
2112 *  function send reset signal to ad7768
2113 *
2114 */
2115 void cmd_AD7768_stop(void)
2116 {
2117     ad7768_reset();
2118
2119 }
2120
2121 /** @fn  void cmd_AD7768_manual_init(void)
2122 *   @brief  function can used to reinitialize ad7768
2123 *
2124 *  function can be used after reset is called.
2125 *
2126 */
2127 void cmd_AD7768_manual_init()
2128 {
2129     ad7768_init();
2130 }
2131 /** @fn  void cmd_AD7768_print_Data(void)
2132 *   @brief  function used to print all data received via HTU and NHET,
2133 *
2134 *  function not implemented
2135 *
2136 */
2137 void cmd_AD7768_print_Data(void)
2138 {
2139     add_to_buffer(0, 0, NEW_RUN, MESSAGE_CONFIRM_ADC);
2140     int j,cnt = 0;
2141     while(cnt < ADC_BUFFER_SAMPLE_COUNT-1)
2142     {
2143         if(ADC_BUFFER_CHANNEL_COUNT == 2)
2144             if(raw_data)
2145             {
2146                 add_to_buffer(adc_samples.counter[cnt], adc_samples.data[j][cnt++], CONTINUE,
      MESSAGE_CONFIRM_ADC);
2147             }
2148             else{
2149                 obc_debug("%d,%d",adc_samples.counter[cnt],adc_samples.data[j][cnt++]);
2150             }
2151         else
2152             obc_debug("%d,%d,%d,%d,%d",adc_samples.counter[cnt],adc_samples.data[j][cnt],
     adc_samples.data[j+1][cnt],adc_samples.data[j+2][cnt],adc_samples.data[j+3][cnt++]);
2153
2154     }
2155     add_to_buffer(0, 0, RUN_COMPLETE, MESSAGE_CONFIRM_ADC);
2156
2157
2158 }
2159
2160 /****************************************** HTU + buffer
      ************************************************/
```

```
2161 /*****************this code under is made by erlend and halvor from ELAB this is not mine
         ******************/
2162
2163
2164
2165
2166
2167
2168 int high_speed_transfer_unit_init (void) {
2169
2170     /* Configure DCP0 for transfer of sampled data from N2HET RAM to CPU RAM */
2171
2172     /* Initial Transfer Count Register (HTU ITCOUNT)
2173      * 20-16     IETCOUNT = Initial Element Transfer Count
2174      * 7-0       IFTCOUNT = Initial Frame Transfer Count
2175      */
2176     htuRAM1->DCP[0].ITCOUNT = (NHET_DATA_FIELD_ELEMENT_COUNT << 16) |
2177                               (HTU_FRAME_TRANSFER_COUNT_MAX / NHET_DATA_FIELD_ELEMENT_COUNT)
         ;
2178
2179     /* Initial N2HET Address and Control Register (HTU IHADDRCT)
2180      * 23        DIR - Direction of Transfer
2181      * 22        SIZE - Size of Transferred Data
2182      * 21        ADDMH - Addressing Mode N2HET Address
2183      * 20        ADDFM - Addressing Mode Main Memory Address
2184      * 19-18     TMBA - Transfer Mode for Buffer A
2185      * 17-16     TMBB - Transfer Mode for Buffer B
2186      * 12-2      IHADDR - Initial N2HET Address
2187      */
2188     htuRAM1->DCP[0].IHADDRCT = (htuRAM1 ->DCP[0].IHADDRCT & 0x0) |
2189                                (0x0 << 23) |
2190                                (0x0 << 22) |
2191                                (0x0 << 21) |
2192                                (0x0 << 20) |
2193                                (0x3 << 18) |
2194                                (0x3 << 16) |
2195                                (NHET_RAM_ADDRESS << 2);
2196
2197     /* DCP0 start address of destination buffers */
2198     htuRAM1->DCP[0].IFADDRA = (unsigned int)htu_buffer_a;
2199     htuRAM1->DCP[0].IFADDRB = (unsigned int)htu_buffer_b;
2200
2201     /* Enable DCP0 CPA */
2202     htuREG1->CPENA = 0x00000001;
2203
2204     /* Enable buffer full interrupt for DCP0 CPA and CPB */
2205     htuREG1->BFINTS = 0x00000003;
2206
2207     /* enable HTU */
2208     htuREG1->GC = 0x00010000;
2209
2210     return EXIT_SUCCESS;
2211 }
2212
2213 /* Interrupt assignment 11: HET TU high level
2214  * It currently takes about 4ms to fill htu_buffer_x
2215  *
2216  * NOTE:
2217  * HTU is not supported by HalCoGen, and sys_vim.c must be
2218  * manually updated after a rebuild of sources. For channel 11
2219  * in s_vim_init, replace phantomInterrupt with
2220  * high_speed_transfer_unit_interrupt
2221  */
2222 #pragma CODE_STATE(high_speed_transfer_unit_interrupt, 32)
2223 #pragma INTERRUPT(high_speed_transfer_unit_interrupt, IRQ)
2224
2225 void high_speed_transfer_unit_interrupt (void) {
2226
2227     volatile uint32_t offset_register, i, j;
2228
2229     /* Read of offset register will automatically clear the interrupt flags */
2230     offset_register = htuREG1->INTOFF0;
2231
2232     switch(offset_register & 0x00000300) {
2233
2234         case 0x00000100:
2235
2236             if ((offset_register & 0x0000000F) == 0x0) {
2237
2238                 // TODO run science routine on data in htu_buffer_a
2239
2240
2241                 for (i=0; i < HTU_FRAME_TRANSFER_COUNT_MAX; i +=
```

```
    NHET_DATA_FIELD_ELEMENT_COUNT) {
2242
2243                    adc_samples.counter[adc_samples.index] = (htu_buffer_a[i] >>
    SAMPLE_COUNTER_SHIFT);
2244
2245                    for (j=0; j < ADC_BUFFER_CHANNEL_COUNT; j++)
2246                        adc_samples.data[j][adc_samples.index] = (htu_buffer_a[i+j+1] & 0
    xffff);
2247
2248                    adc_samples.index++;
2249                    if (adc_samples.index > ADC_BUFFER_SAMPLE_COUNT)
2250                        adc_samples.index = 0;
2251                }
2252
2253
2254            } else if ((offset_register & 0x0000000F) == 0x1) {
2255
2256                // TODO run science routine on data in htu_buffer_b
2257
2258
2259                for (i=0; i < HTU_FRAME_TRANSFER_COUNT_MAX; i +=
    NHET_DATA_FIELD_ELEMENT_COUNT) {
2260
2261                    adc_samples.counter[adc_samples.index] = (htu_buffer_b[i] >>
    SAMPLE_COUNTER_SHIFT);
2262
2263                    for (j=0; j < ADC_BUFFER_CHANNEL_COUNT; j++)
2264                        adc_samples.data[j][adc_samples.index] = (htu_buffer_b[i+j+1] & 0
    xffff);
2265
2266                    adc_samples.index++;
2267                    if (adc_samples.index > ADC_BUFFER_SAMPLE_COUNT)
2268                        adc_samples.index = 0;
2269                }
2270
2271
2272            }
2273            break;
2274
2275        default:
2276            break;
2277    }
2278 }
2279
2280
2281
2282
2283
2284 void stop_het (void)
2285 {
2286
2287     hetREG1->GCR = ( 0x00000000U
2288                    | (uint32)((uint32)0U << 24U)
2289                    | (uint32)((uint32)1U << 16U)
2290                    | (0x00000000U));
2291 }
2292
2293 void start_het (void){
2294
2295     new_sampling_run = true;
2296     hetREG1->GCR = ( 0x00000001U
2297                    | (uint32)((uint32)0U << 24U)
2298                    | (uint32)((uint32)1U << 16U)
2299                    | (0x00000000U));
2300 }
2301
2302
2303
2304
2305 ////////   TEST
2306
2307
2308
2309
2310 static int add_to_buffer (unsigned int a, unsigned int b, int state,uint8 message_id) {
2311
2312     uint8_t buffer[MESSAGE_PAYLOAD_LENGHT_MAX];
2313     static int buffer_index = 0;
2314
2315     if (state == NEW_RUN)
2316         buffer_index = 0;
2317
2318     buffer[buffer_index + 0] = (uint8_t)(a >> 24);
```

```
2319     buffer[buffer_index + 1] = (uint8_t)(a >> 16);
2320     buffer[buffer_index + 2] = (uint8_t)(a >> 8);
2321     buffer[buffer_index + 3] = (uint8_t)(a >> 0);
2322     buffer[buffer_index + 4] = (uint8_t)(b >> 24);
2323     buffer[buffer_index + 5] = (uint8_t)(b >> 16);
2324     buffer[buffer_index + 6] = (uint8_t)(b >> 8);
2325     buffer[buffer_index + 7] = (uint8_t)(b >> 0);
2326 
2327     buffer_index += 8;
2328 
2329     /* About to run out of buffer space. Send message */
2330     if ((buffer_index > (MESSAGE_PAYLOAD_LENGHT_MAX - (8 * 2))) || (state == RUN_COMPLETE)) {
2331         obc_raw_bytes (buffer, buffer_index,message_id);
2332         buffer_index = 0;
2333     }
2334 
2335     return EXIT_SUCCESS;
2336 }
2337 
2338 
2339 ////////    TEST SLUTT
2340 
2341 
2342 
2343 
2344 
2345 /******************************** readout system
     *****************************************/
2346 int reg_data_MibAdc2()
2347  {
2348      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7C200;
2349      unsigned int * last_address_pointer =(unsigned int *) 0xFFF7C3FC;
2350      int reg_cnt = 0;
2351      add_to_buffer (0,0, NEW_RUN,Message_TMS_REG);
2352      do
2353      {
2354 
2355          if (raw_data) {
2356              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
2357          } else {
2358              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2359          }
2360 
2361          start_address_pointer++;
2362          reg_cnt++;
2363 
2364      }
2365      while(start_address_pointer <= last_address_pointer);
2366 
2367      return reg_cnt;
2368  }
2369 
2370 int reg_data_MibAdc1()
2371 {
2372      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7C000;
2373      unsigned int * last_address_pointer =(unsigned int *) 0xFFF7C1FC;
2374      int reg_cnt = 0;
2375      do
2376      {
2377 
2378          if (raw_data) {
2379              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
2380          } else {
2381              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2382          }
2383 
2384          start_address_pointer++;
2385          reg_cnt++;
2386 
2387      }
2388      while(start_address_pointer <= last_address_pointer);
2389 
2390      return reg_cnt;
2391  }
2392 
2393 
2394 int reg_data_Dcan3()
2395 {
2396      unsigned int * start_address_pointer =(unsigned int *) 0xfff7E000;
2397      unsigned int * last_address_pointer =(unsigned int *) 0xFFF7E1FC;
2398      int reg_cnt = 0;
```

```
2399      do
2400      {
2401           if (raw_data) {
2402                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2403           } else {
2404                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2405           }
2406
2407           start_address_pointer++;
2408           reg_cnt++;
2409
2410      }
2411      while(start_address_pointer <= last_address_pointer);
2412
2413      return reg_cnt;
2414  }
2415
2416 int reg_data_Dcan2()
2417 {
2418      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7DE00;
2419      unsigned int * last_address_pointer =(unsigned int *) 0xFFF7DFE4;
2420      int reg_cnt = 0;
2421      do
2422      {
2423
2424           if (raw_data) {
2425                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2426           } else {
2427                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2428           }
2429
2430           start_address_pointer++;
2431           reg_cnt++;
2432
2433      }
2434      while(start_address_pointer <= last_address_pointer);
2435
2436      return reg_cnt;
2437  }
2438
2439 int reg_data_Dcan1()
2440 {
2441      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7DC00;
2442      unsigned int * last_address_pointer =(unsigned int *) 0xFFF7DDE4;
2443      int reg_cnt = 0;
2444      do
2445      {
2446
2447           if (raw_data) {
2448                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2449           } else {
2450                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2451           }
2452
2453           start_address_pointer++;
2454           reg_cnt++;
2455
2456      }
2457      while(start_address_pointer <= last_address_pointer);
2458
2459      return reg_cnt;
2460  }
2461
2462 int reg_data_ePWN1()
2463 {
2464      unsigned int * start_address_pointer =(unsigned int *) 0xFCF78C00;
2465      unsigned int * last_address_pointer =(unsigned int *) 0xFCF78CFC;
2466      int reg_cnt = 0;
2467      do
2468      {
2469           if(start_address_pointer == (unsigned int *)0xFCF78c3c || 0xFCF78c38)
2470                     {
2471                          goto skip;
2472                     }
2473           if (raw_data) {
2474                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2475           } else {
2476                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2477           }
```

```
2478          skip:
2479          start_address_pointer++;
2480          reg_cnt++;
2481
2482      }
2483      while(start_address_pointer <= last_address_pointer);
2484
2485      return reg_cnt;
2486  }
2487
2488
2489
2490  int reg_data_ePWN2()
2491  {
2492      unsigned int * start_address_pointer =(unsigned int *) 0xFCF78D00;
2493      unsigned int * last_address_pointer =(unsigned int *) 0xFCF78DFC;
2494      int reg_cnt = 0;
2495      do
2496      {
2497
2498          if(start_address_pointer == (unsigned int *)0xFCF78d3c || 0xFCF78D38)
2499                                  {
2500                                          goto skip;
2501                                  }
2502          if (raw_data) {
2503              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2504          } else {
2505              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2506          }
2507          skip:
2508          start_address_pointer++;
2509          reg_cnt++;
2510
2511      }
2512      while(start_address_pointer <= last_address_pointer);
2513
2514      return reg_cnt;
2515  }
2516
2517  int reg_data_ePWN3()
2518  {
2519      unsigned int * start_address_pointer =(unsigned int *) 0xFCF78E00;
2520      unsigned int * last_address_pointer =(unsigned int *) 0xFCF78EFC;
2521      int reg_cnt = 0;
2522      do
2523      {
2524          if(start_address_pointer == (unsigned int *)0xFCF78E3c|| 0xFCF78E38)
2525                                  {
2526                                          goto skip;
2527                                  }
2528          if (raw_data) {
2529              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2530          } else {
2531              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2532          }
2533          skip:
2534          start_address_pointer++;
2535          reg_cnt++;
2536
2537      }
2538      while(start_address_pointer <= last_address_pointer);
2539
2540      return reg_cnt;
2541  }
2542
2543  int reg_data_ePWN4()
2544  {
2545      unsigned int * start_address_pointer =(unsigned int *) 0xFCF78F00;
2546      unsigned int * last_address_pointer  =(unsigned int *) 0xFCF78FFC;
2547      int reg_cnt = 0;
2548      do
2549      {
2550          if(start_address_pointer == (unsigned int *)0xFCF78F3c|| 0xFCF78F38)
2551                                  {
2552                                          goto skip;
2553                                  }
2554
2555          if (raw_data) {
2556              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2557          } else {
```

```
2558                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2559            }
2560            skip:
2561            start_address_pointer++;
2562            reg_cnt++;
2563
2564        }
2565        while(start_address_pointer <= last_address_pointer);
2566
2567        return reg_cnt;
2568    }
2569
2570    int reg_data_ePWN5()
2571    {
2572        unsigned int * start_address_pointer =(unsigned int *) 0xFCF79000;
2573        unsigned int * last_address_pointer  =(unsigned int *) 0xFCF790FC;
2574        int reg_cnt = 0;
2575        do
2576        {
2577            if(start_address_pointer == (unsigned int *)0xFCF7803c|| 0xFCF78038)
2578                                {
2579                                    goto skip;
2580                                }
2581            if (raw_data) {
2582                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
        Message_TMS_REG);
2583            } else {
2584                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2585            }
2586            skip:
2587            start_address_pointer++;
2588            reg_cnt++;
2589
2590        }
2591        while(start_address_pointer <= last_address_pointer);
2592
2593        return reg_cnt;
2594    }
2595
2596    int reg_data_ePWN6()
2597    {
2598        unsigned int * start_address_pointer =(unsigned int *) 0xFCF79100;
2599        unsigned int * last_address_pointer  =(unsigned int *) 0xFCF791FC;
2600        int reg_cnt = 0;
2601        do
2602        {
2603
2604            if(start_address_pointer == (unsigned int *)0xFCF7813c|| 0xFCF78138)
2605                                {
2606                                    goto skip;
2607                                }
2608            if (raw_data) {
2609                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
        Message_TMS_REG);
2610            } else {
2611                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2612            }
2613            skip:
2614            start_address_pointer++;
2615            reg_cnt++;
2616
2617        }
2618        while(start_address_pointer <= last_address_pointer);
2619
2620        return reg_cnt;
2621    }
2622
2623    int reg_data_ePWN7()
2624    {
2625        unsigned int * start_address_pointer =(unsigned int *) 0xFCF79200;
2626        unsigned int * last_address_pointer  =(unsigned int *) 0xFCF792FC;
2627        int reg_cnt = 0;
2628        do
2629        {
2630            if(start_address_pointer == (unsigned int *)0xFCF7823c|| 0xFCF78238)
2631                                {
2632                                    goto skip;
2633                                }
2634            if (raw_data) {
2635                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
        Message_TMS_REG);
2636            } else {
2637                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
```

```
2638          }
2639          skip:
2640          start_address_pointer++;
2641          reg_cnt++;
2642
2643      }
2644      while(start_address_pointer <= last_address_pointer);
2645
2646      return reg_cnt;
2647  }
2648
2649  int reg_data_eCAP1()
2650  {
2651      unsigned int * start_address_pointer =(unsigned int *) 0xFCF79300;
2652      unsigned int * last_address_pointer  =(unsigned int *) 0xFCF793FC;
2653      int reg_cnt = 0;
2654      do
2655      {
2656          if(start_address_pointer == (unsigned int *)0xFCF79314 || 0xFCF79310 )
2657          {
2658                      goto skip;
2659          }
2660          if (raw_data) {
2661              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2662          } else {
2663              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2664          }
2665
2666          skip:
2667          start_address_pointer++;
2668          reg_cnt++;
2669
2670      }
2671      while(start_address_pointer <= last_address_pointer);
2672
2673      return reg_cnt;
2674  }
2675
2676  int reg_data_eCAP2()
2677  {
2678      unsigned int * start_address_pointer =(unsigned int *) 0xFCF79400;
2679      unsigned int * last_address_pointer  =(unsigned int *) 0xFCF794FC;
2680      int reg_cnt = 0;
2681      do
2682      {
2683          if(start_address_pointer == (unsigned int *)0xFCF79414 || 0xFCF79410 )
2684          {
2685                              goto skip;
2686          }
2687
2688          if (raw_data) {
2689              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2690          } else {
2691              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2692          }
2693
2694          skip:
2695          start_address_pointer++;
2696          reg_cnt++;
2697
2698      }
2699      while(start_address_pointer <= last_address_pointer);
2700
2701      return reg_cnt;
2702  }
2703
2704
2705
2706  int reg_data_eCAP3()
2707  {
2708      unsigned int * start_address_pointer =(unsigned int *) 0xFCF79500;
2709      unsigned int * last_address_pointer  =(unsigned int *) 0xFCF795FC;
2710      int reg_cnt = 0;
2711      do
2712      {
2713          if(start_address_pointer == (unsigned int *)0xFCF79514 || 0xFCF79510 )
2714                  {
2715                              goto skip;
2716                  }
2717
2718          if (raw_data) {
```

```
2719                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
    Message_TMS_REG);
2720            } else {
2721                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2722            }
2723            skip:
2724            start_address_pointer++;
2725            reg_cnt++;
2726
2727        }
2728        while(start_address_pointer <= last_address_pointer);
2729
2730        return reg_cnt;
2731    }
2732
2733    int reg_data_eCAP4()
2734    {
2735        unsigned int * start_address_pointer =(unsigned int *) 0xFCF79600;
2736        unsigned int * last_address_pointer  =(unsigned int *) 0xFCF796FC;
2737        int reg_cnt = 0;
2738        do
2739        {
2740            if(start_address_pointer == (unsigned int *)0xFCF79614 || 0xFCF79610 )
2741                    {
2742                                goto skip;
2743                    }
2744
2745            if (raw_data) {
2746                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
    Message_TMS_REG);
2747            } else {
2748                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2749            }
2750            skip:
2751            start_address_pointer++;
2752            reg_cnt++;
2753
2754        }
2755        while(start_address_pointer <= last_address_pointer);
2756
2757        return reg_cnt;
2758    }
2759
2760    int reg_data_eCAP5()
2761    {
2762        unsigned int * start_address_pointer =(unsigned int *) 0xFCF79700;
2763        unsigned int * last_address_pointer  =(unsigned int *) 0xFCF797FC;
2764        int reg_cnt = 0;
2765        do
2766        {
2767
2768            if(start_address_pointer == (unsigned int *)0xFCF79714 || 0xFCF79710 )
2769                    {
2770                                goto skip;
2771                    }
2772            if (raw_data) {
2773                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
    Message_TMS_REG);
2774            } else {
2775                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2776            }
2777
2778            skip:
2779            start_address_pointer++;
2780            reg_cnt++;
2781
2782        }
2783        while(start_address_pointer <= last_address_pointer);
2784
2785        return reg_cnt;
2786    }
2787
2788    int reg_data_eCAP6()
2789    {
2790        unsigned int * start_address_pointer =(unsigned int *) 0xFCF79800;
2791        unsigned int * last_address_pointer  =(unsigned int *) 0xFCF798FC;
2792        int reg_cnt = 0;
2793        do
2794        {
2795            if(start_address_pointer == (unsigned int *)0xFCF79814 || 0xFCF79810 )
2796             {
2797                                goto skip;
2798             }
```

```
2799
2800          if (raw_data) {
2801              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2802          } else {
2803              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2804          }
2805          skip:
2806          start_address_pointer++;
2807          reg_cnt++;
2808
2809      }
2810      while(start_address_pointer <= last_address_pointer);
2811
2812      return reg_cnt;
2813  }
2814
2815  int reg_data_eQEP1()
2816  {
2817      unsigned int * start_address_pointer =(unsigned int *) 0xFCF79900;
2818      unsigned int * last_address_pointer  =(unsigned int *) 0xFCF799FC;
2819      int reg_cnt = 0;
2820      do
2821      {
2822
2823          if(start_address_pointer == (unsigned int *)0xFCF7997C || 0xFCF79978)
2824          {
2825               goto skip;
2826          }
2827          if (raw_data) {
2828              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2829          } else {
2830              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2831          }
2832
2833          skip:
2834          start_address_pointer++;
2835          reg_cnt++;
2836
2837      }
2838      while(start_address_pointer <= last_address_pointer);
2839
2840      return reg_cnt;
2841  }
2842
2843  int reg_data_eQEP2()
2844  {
2845      unsigned int * start_address_pointer =(unsigned int *) 0xFCF79A00;
2846      unsigned int * last_address_pointer  =(unsigned int *) 0xFCF79AFC;
2847      int reg_cnt = 0;
2848      do
2849      {
2850          if(start_address_pointer == (unsigned int *)0xFCF79a7C || 0xFCF79a78)
2851          {
2852                       goto skip;
2853          }
2854
2855          if (raw_data) {
2856              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2857          } else {
2858              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2859          }
2860
2861          skip:
2862          start_address_pointer++;
2863          reg_cnt++;
2864
2865      }
2866      while(start_address_pointer <= last_address_pointer);
2867
2868      return reg_cnt;
2869  }
2870
2871
2872
2873  int reg_data_Gio()
2874  {
2875      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7BC00;
2876      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7BC33;
2877      int reg_cnt = 0;
2878      do
```

```
2879      {
2880
2881          if (raw_data) {
2882              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2883          } else {
2884              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2885          }
2886
2887          start_address_pointer++;
2888          reg_cnt++;
2889
2890      }
2891      while(start_address_pointer <= last_address_pointer);
2892
2893      return reg_cnt;
2894  }
2895
2896 int reg_data_GioA()
2897 {
2898      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7BC34;
2899      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7BC53;
2900      int reg_cnt = 0;
2901      do
2902      {
2903
2904          if (raw_data) {
2905              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2906          } else {
2907              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2908          }
2909
2910          start_address_pointer++;
2911          reg_cnt++;
2912
2913      }
2914      while(start_address_pointer <= last_address_pointer);
2915
2916      return reg_cnt;
2917  }
2918
2919 int reg_data_GioB()
2920 {
2921      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7BC54;
2922      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7BCFF;
2923      int reg_cnt = 0;
2924      do
2925      {
2926
2927          if (raw_data) {
2928              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2929          } else {
2930              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2931          }
2932
2933          start_address_pointer++;
2934          reg_cnt++;
2935
2936      }
2937      while(start_address_pointer <= last_address_pointer);
2938
2939      return reg_cnt;
2940  }
2941
2942 int reg_data_I2C()
2943 {
2944      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7D400;
2945      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7D4FF;
2946      int reg_cnt = 0;
2947      do
2948      {
2949
2950          if (raw_data) {
2951              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
2952          } else {
2953              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2954          }
2955
2956          start_address_pointer++;
2957          reg_cnt++;
```

```
2958        }
2959        while(start_address_pointer <= last_address_pointer);
2960
2961        return reg_cnt;
2962    }
2963
2964
2965    int reg_data_NHET1()
2966    {
2967        unsigned int * start_address_pointer =(unsigned int *) 0xFFF7B800;
2968        unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7b8FF;
2969        int reg_cnt = 0;
2970        do
2971        {
2972
2973            if (raw_data) {
2974                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
        Message_TMS_REG);
2975            } else {
2976                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
2977            }
2978
2979            start_address_pointer++;
2980            reg_cnt++;
2981
2982        }
2983        while(start_address_pointer <= last_address_pointer);
2984
2985        return reg_cnt;
2986    }
2987
2988    int reg_data_NHET2()
2989    {
2990        unsigned int * start_address_pointer =(unsigned int *) 0xFFF7B900;
2991        unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7b9FF;
2992        int reg_cnt = 0;
2993        do
2994        {
2995
2996            if (raw_data) {
2997                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
        Message_TMS_REG);
2998            } else {
2999                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3000            }
3001
3002            start_address_pointer++;
3003            reg_cnt++;
3004        }
3005        while(start_address_pointer <= last_address_pointer);
3006
3007        return reg_cnt;
3008    }
3009    int reg_data_HTU1()
3010    {
3011           unsigned int * start_address_pointer =(unsigned int *) 0xFFF7A400;
3012           unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7A47b;
3013           int reg_cnt = 0;
3014           do
3015           {
3016
3017                if(start_address_pointer == (unsigned int *)0xFFF7A418 || 0xFFF7A415  )
3018                {
3019                    goto skip;
3020                }
3021                if (raw_data) {
3022                    add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
        Message_TMS_REG);
3023                } else {
3024                    printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3025                }
3026
3027                skip:
3028                start_address_pointer++;
3029                reg_cnt++;
3030
3031           }
3032           while(start_address_pointer <= last_address_pointer);
3033
3034           return reg_cnt;
3035    }
3036    int reg_data_HTU2()
3037    {
```

```
3038      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7A500;
3039      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7A5FF;
3040      int reg_cnt = 0;
3041      do
3042      {
3043          if(start_address_pointer == (unsigned int *)0xFFF7A518 ||  0xFFF7A514)
3044          {
3045              goto skip;
3046          }
3047          if (raw_data) {
3048              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3049          } else {
3050              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3051          }
3052          skip:
3053          start_address_pointer++;
3054          reg_cnt++;
3055
3056      }
3057      while(start_address_pointer <= last_address_pointer);
3058
3059      return reg_cnt;
3060  }
3061
3062  int reg_data_IOMM()
3063  {
3064      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFEA00;
3065      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFEBFF;
3066      int reg_cnt = 0;
3067      do
3068      {
3069
3070
3071          if (raw_data) {
3072              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3073          } else {
3074              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3075          }
3076
3077          start_address_pointer++;
3078          reg_cnt++;
3079
3080      }
3081      while(start_address_pointer <= last_address_pointer);
3082
3083      return reg_cnt;
3084  }
3085
3086
3087  int reg_data_MibSpi1()
3088  {
3089      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7F400;
3090      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7F5FF;
3091      int reg_cnt = 0;
3092      do
3093      {
3094
3095          if (raw_data) {
3096              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3097          } else {
3098              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3099          }
3100
3101          start_address_pointer++;
3102          reg_cnt++;
3103
3104      }
3105      while(start_address_pointer <= last_address_pointer);
3106
3107      return reg_cnt;
3108  }
3109
3110
3111  int reg_data_Spi2()
3112  {
3113      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7F600;
3114      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7F7FF;
3115      int reg_cnt = 0;
3116      do
3117      {
```

```
3118
3119          if (raw_data) {
3120              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3121          } else {
3122              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3123          }
3124
3125          start_address_pointer++;
3126          reg_cnt++;
3127
3128      }
3129      while(start_address_pointer <= last_address_pointer);
3130
3131      return reg_cnt;
3132  }
3133
3134  int reg_data_MibSpi3()
3135  {
3136      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7F800;
3137      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7F9FF;
3138      int reg_cnt = 0;
3139      do
3140      {
3141
3142          if (raw_data) {
3143              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3144          } else {
3145              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3146          }
3147
3148          start_address_pointer++;
3149          reg_cnt++;
3150
3151      }
3152      while(start_address_pointer <= last_address_pointer);
3153
3154      return reg_cnt;
3155  }
3156
3157  int reg_data_Spi4()
3158  {
3159      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7FA00;
3160      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7FBFF;
3161      int reg_cnt = 0;
3162      do
3163      {
3164
3165          if (raw_data) {
3166              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3167          } else {
3168              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3169          }
3170
3171          start_address_pointer++;
3172          reg_cnt++;
3173
3174      }
3175      while(start_address_pointer <= last_address_pointer);
3176
3177      return reg_cnt;
3178  }
3179
3180  int reg_data_MibSpip5()
3181  {
3182      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7FC00;
3183      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7FDFF;
3184      int reg_cnt = 0;
3185      do
3186      {
3187
3188          if (raw_data) {
3189              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3190          } else {
3191              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3192          }
3193
3194          start_address_pointer++;
3195          reg_cnt++;
3196
```

```
3197      }
3198      while(start_address_pointer <= last_address_pointer);
3199
3200      return reg_cnt;
3201  }
3202
3203  int reg_data_Lin2()
3204  {
3205      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7E500;
3206      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7E5FF;
3207      int reg_cnt = 0;
3208      do
3209      {
3210
3211          if (raw_data) {
3212              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3213          } else {
3214              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3215          }
3216
3217          start_address_pointer++;
3218          reg_cnt++;
3219
3220      }
3221      while(start_address_pointer <= last_address_pointer);
3222
3223      return reg_cnt;
3224  }
3225
3226  int reg_data_Lin1()
3227  {
3228      unsigned int * start_address_pointer =(unsigned int *) 0xFFF7E400;
3229      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF7E4FF;
3230      int reg_cnt = 0;
3231      do
3232      {
3233
3234          if (raw_data) {
3235              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3236          } else {
3237              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3238          }
3239
3240          start_address_pointer++;
3241          reg_cnt++;
3242
3243      }
3244      while(start_address_pointer <= last_address_pointer);
3245
3246      return reg_cnt;
3247  }
3248
3249  int reg_data_CcmR4()
3250  {
3251      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFF600;
3252      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFF6FF;
3253      int reg_cnt = 0;
3254      do
3255      {
3256
3257          if (raw_data) {
3258              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3259          } else {
3260              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3261          }
3262
3263          start_address_pointer++;
3264          reg_cnt++;
3265
3266      }
3267      while(start_address_pointer <= last_address_pointer);
3268
3269      return reg_cnt;
3270  }
3271
3272
3273  int reg_data_Crc()
3274  {
3275      unsigned int * start_address_pointer =(unsigned int *) 0xFE000000;
3276      unsigned int * last_address_pointer  =(unsigned int *) 0xFE000143;
```

```
3277      int reg_cnt = 0;
3278      do
3279      {
3280
3281          if (raw_data) {
3282              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3283          } else {
3284              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3285          }
3286
3287          start_address_pointer++;
3288          reg_cnt++;
3289
3290      }
3291      while(start_address_pointer <= last_address_pointer);
3292
3293      return reg_cnt;
3294  }
3295
3296  int reg_data_Dcc1()
3297  {
3298      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFEC00;
3299      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFEC2B;
3300      int reg_cnt = 0;
3301      do
3302      {
3303
3304          if (raw_data) {
3305              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3306          } else {
3307              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3308          }
3309
3310          start_address_pointer++;
3311          reg_cnt++;
3312
3313      }
3314      while(start_address_pointer <= last_address_pointer);
3315
3316      return reg_cnt;
3317  }
3318
3319  int reg_data_Dcc2()
3320  {
3321      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFF400;
3322      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFF42B;
3323      int reg_cnt = 0;
3324      do
3325      {
3326
3327          if (raw_data) {
3328              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3329          } else {
3330              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3331          }
3332
3333          start_address_pointer++;
3334          reg_cnt++;
3335
3336      }
3337      while(start_address_pointer <= last_address_pointer);
3338
3339      return reg_cnt;
3340  }
3341
3342
3343  int reg_data_Dma()
3344  {
3345      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFF000;
3346      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFF1D7;
3347      int reg_cnt = 0;
3348      do
3349      {
3350
3351          if (raw_data) {
3352              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3353          } else {
3354              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3355          }
```

```
3356
3357          start_address_pointer++;
3358          reg_cnt++;
3359
3360      }
3361      while(start_address_pointer <= last_address_pointer);
3362
3363      return reg_cnt;
3364  }
3365
3366  int reg_data_Esm()
3367  {
3368      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFF500;
3369      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFF55B;
3370      int reg_cnt = 0;
3371      do
3372      {
3373
3374          if (raw_data) {
3375              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3376          } else {
3377              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3378          }
3379
3380          start_address_pointer++;
3381          reg_cnt++;
3382          wait(20);
3383
3384      }
3385      while(start_address_pointer <= last_address_pointer);
3386
3387      return reg_cnt;
3388  }
3389
3390
3391
3392
3393  int reg_data_flashWrapper()
3394  {
3395      unsigned int * start_address_pointer =(unsigned int *) 0xFFF87000;
3396      unsigned int * last_address_pointer  =(unsigned int *) 0xFFF870c3;
3397      int reg_cnt = 0;
3398      do
3399      {
3400
3401          if (raw_data) {
3402              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3403          } else {
3404              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3405          }
3406
3407          start_address_pointer++;
3408          reg_cnt++;
3409
3410      }
3411      while(start_address_pointer <= last_address_pointer);
3412
3413      return reg_cnt;
3414  }
3415
3416  int reg_data_Pbist()
3417  {
3418      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFE500;
3419      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFE5CF;
3420      int reg_cnt = 0;
3421      do
3422      {
3423
3424          if (raw_data) {
3425              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3426          } else {
3427              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3428          }
3429
3430          start_address_pointer++;
3431          reg_cnt++;
3432
3433      }
3434      while(start_address_pointer <= last_address_pointer);
3435
```

```
3436      return reg_cnt;
3437  }
3438
3439  int reg_data_PMM()
3440  {
3441      unsigned int * start_address_pointer =(unsigned int *) 0xFFFF0000;
3442      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFF00C3;
3443      int reg_cnt = 0;
3444      do
3445      {
3446
3447          if(start_address_pointer == (unsigned int *)0xFFFF0000 || 0xFFFFe5cc )
3448          {
3449              goto skip;
3450          }
3451
3452          if (raw_data) {
3453              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3454          } else {
3455              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3456          }
3457
3458          skip:
3459          start_address_pointer++;
3460          reg_cnt++;
3461
3462      }
3463      while(start_address_pointer <= last_address_pointer);
3464
3465      return reg_cnt;
3466  }
3467
3468
3469  int reg_data_Rti()
3470  {
3471      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFFC00;
3472      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFFCBF;
3473      int reg_cnt = 0;
3474      do
3475      {
3476
3477          if (raw_data) {
3478              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3479          } else {
3480              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3481          }
3482
3483          start_address_pointer++;
3484          reg_cnt++;
3485
3486      }
3487      while(start_address_pointer <= last_address_pointer);
3488
3489      return reg_cnt;
3490  }
3491
3492  int reg_data_Stc()
3493  {
3494       unsigned int * start_address_pointer =(unsigned int *) 0xFFFFE600;
3495       unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFE63F;
3496       int reg_cnt = 0;
3497       do
3498       {
3499
3500           if (raw_data) {
3501               add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3502           } else {
3503               printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3504           }
3505
3506           start_address_pointer++;
3507           reg_cnt++;
3508
3509       }
3510       while(start_address_pointer <= last_address_pointer);
3511
3512       return reg_cnt;
3513    }
3514
3515  int reg_data_Sys()
```

```
3516 {
3517      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFFF00;
3518      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFFFFB;
3519      do
3520      {
3521
3522          if (raw_data) {
3523              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3524          } else {
3525              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3526          }
3527
3528          start_address_pointer++;
3529
3530      }
3531      while(start_address_pointer <= last_address_pointer);
3532
3533      return 0;
3534  }
3535
3536 int reg_data_Sys2()
3537 {
3538      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFE100;
3539      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFE1FF;
3540      do
3541      {
3542
3543          if (raw_data) {
3544              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3545          } else {
3546              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3547          }
3548
3549          start_address_pointer++;
3550
3551      }
3552      while(start_address_pointer <= last_address_pointer);
3553
3554      return 0;
3555  }
3556
3557 int reg_data_Vim()
3558 {
3559      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFFE00;
3560      unsigned int * last_address_pointer  =(unsigned int *) 0xfffffedc;
3561      int reg_cnt = 0;
3562      do
3563      {
3564
3565          if (raw_data) {
3566              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3567          } else {
3568              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3569          }
3570
3571          start_address_pointer++;
3572          reg_cnt++;
3573
3574      }
3575      while(start_address_pointer <= last_address_pointer);
3576
3577      return reg_cnt;
3578  }
3579
3580 int reg_data_VimPar()
3581 {
3582      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFFDEC;
3583      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFFDFF;
3584      int reg_cnt = 0;
3585      do
3586      {
3587
3588          if (raw_data) {
3589              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3590          } else {
3591              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3592          }
3593
3594          start_address_pointer++;
```

```
3595            reg_cnt++;
3596
3597        }
3598        while(start_address_pointer <= last_address_pointer);
3599
3600        return reg_cnt;
3601    }
3602
3603    int reg_data_Pom()
3604    {
3605        //?
3606        unsigned int * start_address_pointer =(unsigned int *) 0xFFA04000;
3607        unsigned int * last_address_pointer  =(unsigned int *) 0xFFA04FFF;
3608        int reg_cnt = 0;
3609        do
3610        {
3611
3612
3613            if(start_address_pointer == (unsigned int *)0xFFF7A518 ||  0xFFF7A514)
3614            {
3615                        goto skip;
3616            }
3617            if (raw_data) {
3618                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
        Message_TMS_REG);
3619            } else {
3620                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3621            }
3622
3623            skip:
3624            start_address_pointer++;
3625            reg_cnt++;
3626
3627        }
3628        while(start_address_pointer <= last_address_pointer);
3629
3630        return reg_cnt;
3631    }
3632    int reg_data_Emif()
3633    {
3634        unsigned int * start_address_pointer =(unsigned int *) 0xFCFFE800;
3635        unsigned int * last_address_pointer  =(unsigned int *) 0xFCFFE8DC;
3636        int reg_cnt = 0;
3637
3638        do
3639        {
3640
3641            if (raw_data) {
3642                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
        Message_TMS_REG);
3643            } else {
3644                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3645            }
3646
3647            start_address_pointer++;
3648            reg_cnt++;
3649        }
3650        while(start_address_pointer <= last_address_pointer);
3651
3652        return reg_cnt;
3653    }
3654
3655    int reg_data_Pcr()
3656    {
3657        unsigned int * start_address_pointer =(unsigned int *) 0xFFFFE000;
3658        //last is 0xffffe0ac
3659        unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFE0AC;
3660        int reg_cnt = 0;
3661        do
3662        {
3663
3664            if (raw_data) {
3665                add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
        Message_TMS_REG);
3666            } else {
3667                printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3668            }
3669
3670            start_address_pointer++;
3671            reg_cnt++;
3672
3673        }
3674        while(start_address_pointer <= last_address_pointer);
```

```
3675
3676      return reg_cnt;
3677  }
3678
3679  int reg_data_RamWrapper_Even()
3680  {
3681      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFF800;
3682      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFF84B;
3683      int reg_cnt = 0;
3684      do
3685      {
3686
3687
3688          if(start_address_pointer == (unsigned int *)0xFFFFF814  || 0xFFFFF810 )
3689          {
3690             goto skip;
3691           }
3692          if (raw_data) {
3693              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3694          } else {
3695              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3696          }
3697          skip:
3698          start_address_pointer++;
3699          reg_cnt++;
3700
3701      }
3702      while(start_address_pointer <= last_address_pointer);
3703      return reg_cnt;
3704  }
3705
3706  int reg_data_RamWrapper_Odd()
3707  {
3708      unsigned int * start_address_pointer =(unsigned int *) 0xFFFFF900;
3709      unsigned int * last_address_pointer  =(unsigned int *) 0xFFFFF94B;
3710      int reg_cnt = 0;
3711      do
3712      {
3713          if(start_address_pointer == (unsigned int *)0xFFFFF914 || 0xFFFFF910)
3714          {
3715             goto skip;
3716          }
3717
3718          if (raw_data) {
3719              add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
     Message_TMS_REG);
3720          } else {
3721              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3722          }
3723
3724          skip:
3725          start_address_pointer++;
3726          reg_cnt++;
3727
3728      }
3729      while(start_address_pointer <= last_address_pointer);
3730
3731      return reg_cnt;
3732  }
3733
3734
3735
3736
3737  #define  TSIZE1 140000
3738  #pragma RETAIN(Control_array)
3739  #pragma location = 0x08001600
3740  uint8    Control_array[TSIZE1];
3741  uint8    Control_Value =0x48;
3742
3743  void RAMtest_init()
3744  {
3745  memset(&Control_array,Control_Value,sizeof(Control_array));
3746  }
3747
3748  int RAM_data_read()
3749  {
3750      unsigned int * start_address_pointer =(unsigned int *) &Control_array[0];
3751      unsigned int * last_address_pointer  =(unsigned int *) &Control_array[TSIZE1-1];
3752      add_to_buffer (0,0, NEW_RUN,Message_TMS_REG);
3753      int reg_cnt = 0;
3754      //unsigned int * test_address_pointer =(unsigned int *) 0x08000000;
3755      //add_to_buffer (test_address_pointer,*test_address_pointer, CONTINUE,Message_TMS_REG);
```

```
3756      //add_to_buffer ( ESM0_monitor(),ESM1_monitor(), CONTINUE,Message_TMS_REG);
3757      do
3758      {
3759
3760
3761
3762          if (raw_data) {
3763              if(reg_cnt == 0)
3764                  {
3765                      add_to_buffer (start_address_pointer,*start_address_pointer, NEW_RUN,
      Message_TMS_REG);
3766                      add_to_buffer ( reg_cnt,ESM0_monitor(), CONTINUE,Message_TMS_REG);
3767                      reg_cnt++;
3768                  }
3769                  else if(reg_cnt == 28)
3770                  {
3771                      add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3772                      add_to_buffer ( reg_cnt,ESM0_monitor(), RUN_COMPLETE,Message_TMS_REG);
3773                      reg_cnt =0;
3774                  }
3775                  else{
3776                     add_to_buffer (start_address_pointer,*start_address_pointer, CONTINUE,
      Message_TMS_REG);
3777                      add_to_buffer ( reg_cnt,ESM0_monitor(), CONTINUE,Message_TMS_REG);
3778                      reg_cnt++;
3779                  }
3780          } else {
3781              printf("\r\n %x ,%x",start_address_pointer,*start_address_pointer);
3782          }
3783          start_address_pointer++;
3784
3785      }
3786      while(start_address_pointer <= last_address_pointer);
3787      add_to_buffer (0,0, RUN_COMPLETE,Message_TMS_REG);
3788      return reg_cnt;
3789  }
3790
3791  void reg_data_cmd()
3792  {
3793      int reg8_read_cnt = 0;
3794      reg8_read_cnt +=reg_data_MibAdc2();
3795      reg8_read_cnt +=reg_data_MibAdc1();
3796      reg8_read_cnt +=reg_data_Dcan3();
3797      reg8_read_cnt +=reg_data_Dcan2();
3798      reg8_read_cnt +=reg_data_Dcan1();
3799      reg8_read_cnt +=reg_data_ePWN1();
3800      reg8_read_cnt +=reg_data_ePWN2();
3801      reg8_read_cnt +=reg_data_ePWN3();
3802      reg8_read_cnt +=reg_data_ePWN4();
3803      reg8_read_cnt +=reg_data_ePWN5();
3804      reg8_read_cnt +=reg_data_ePWN6();
3805      reg8_read_cnt +=reg_data_ePWN7();
3806      reg8_read_cnt +=reg_data_eCAP1();
3807      reg8_read_cnt +=reg_data_eCAP2();
3808      reg8_read_cnt +=reg_data_eCAP3();
3809      reg8_read_cnt +=reg_data_eCAP4();
3810      reg8_read_cnt +=reg_data_eCAP5();
3811      reg8_read_cnt +=reg_data_eCAP6();
3812      reg8_read_cnt +=reg_data_eQEP1();
3813      reg8_read_cnt +=reg_data_eQEP2();
3814      reg8_read_cnt +=reg_data_Gio();
3815      reg8_read_cnt +=reg_data_GioA();
3816      reg8_read_cnt +=reg_data_GioB();
3817      reg8_read_cnt +=reg_data_I2C();
3818      reg8_read_cnt +=reg_data_NHET1();
3819      reg8_read_cnt +=reg_data_NHET2();
3820      reg8_read_cnt +=reg_data_HTU1();
3821      reg8_read_cnt +=reg_data_HTU2();
3822      reg8_read_cnt +=reg_data_IOMM();
3823      reg8_read_cnt +=reg_data_MibSpi1();
3824      reg8_read_cnt +=reg_data_Spi2();
3825      reg8_read_cnt +=reg_data_MibSpi3();
3826      reg8_read_cnt +=reg_data_Spi4();
3827      reg8_read_cnt +=reg_data_MibSpip5();
3828      reg8_read_cnt +=reg_data_Lin2();
3829      reg8_read_cnt +=reg_data_Lin1();
3830      reg8_read_cnt +=reg_data_CcmR4();
3831      reg8_read_cnt +=reg_data_Crc();
3832      reg8_read_cnt +=reg_data_Dcc1();
3833      reg8_read_cnt +=reg_data_Dcc2();
3834      reg8_read_cnt +=reg_data_Dma();
3835      reg8_read_cnt +=reg_data_Esm();
```

```
3836     reg8_read_cnt +=reg_data_flashWrapper();
3837     reg8_read_cnt +=reg_data_Pbist();
3838     reg8_read_cnt +=reg_data_PMM();
3839     reg8_read_cnt +=reg_data_Rti();
3840     reg8_read_cnt +=reg_data_Stc();
3841     reg8_read_cnt +=reg_data_Sys();
3842     reg8_read_cnt +=reg_data_Sys2();
3843     reg8_read_cnt +=reg_data_Vim();
3844     reg8_read_cnt +=reg_data_VimPar();
3845     reg8_read_cnt +=reg_data_Pom();
3846     reg8_read_cnt +=reg_data_Emif();
3847     reg8_read_cnt +=reg_data_Pcr();
3848     //reg8_read_cnt +=reg_data_RamWrapper_Even();
3849     //reg8_read_cnt += reg_data_RamWrapper_Odd();
3850     cmd_AD7768_GET_AllConfigs();
3851     cmd_ltc3887_read_config();
3852     add_to_buffer (0,0, RUN_COMPLETE,Message_TMS_REG);
3853     //printf("\r\n Number regs %d \r\n", reg8_read_cnt);
3854     /*total number of register that are printed 4580
3855      * those varies in size from 8 to 32 bits*/
3856     //obc_debug("Number regs %d", reg8_read_cnt);
3857
3858 }
3859
3860 void raw_board_status()
3861 {
3862     IVmeasurements();
3863
3864
3865     add_to_buffer (MON_I_V.P5V0_V,MON_I_V.P5V0_I, NEW_RUN, MESSAGE_CONFIRM_BOARD_STATUS);
3866     add_to_buffer (MON_I_V.P3V3_V,MON_I_V.P3V3_I, CONTINUE,MESSAGE_CONFIRM_BOARD_STATUS);
3867     add_to_buffer (MON_I_V.P12V_V,MON_I_V.P12V_I, CONTINUE,MESSAGE_CONFIRM_BOARD_STATUS);
3868     add_to_buffer (MON_I_V.N12V_V,MON_I_V.N12V_I, CONTINUE,MESSAGE_CONFIRM_BOARD_STATUS);
3869     add_to_buffer (MON_I_V.P1V2_I,MON_I_V.EGUN_I, CONTINUE,MESSAGE_CONFIRM_BOARD_STATUS);
3870     add_to_buffer (MON_I_V.P1V2_I,MON_I_V.EGUN_I, CONTINUE,MESSAGE_CONFIRM_BOARD_STATUS);
3871     add_to_buffer (MON_I_V.P1V2_I,MON_I_V.EGUN_I, RUN_COMPLETE,MESSAGE_CONFIRM_BOARD_STATUS);
3872     //cmd_AD7768_print_Data();
3873
3874 }
3875
3876 void dac_status()
3877 {
3878     DAC_GAIN_SET_STATUS();
3879     cmd_ltc3887_Read_output_voltage(0,0);
3880     cmd_ltc3887_Read_internal_Temperature(0,0);
3881     cmd_ltc3887_output_current(0, 0);
3882     add_to_buffer (BIAS_MON_V.DAC_BIAS_CH1,BIAS_MON_V.DAC_BIAS_CH2, NEW_RUN,
          MESSAGE_CONFIRM_DAC);
3883     add_to_buffer (BIAS_MON_V.DAC_BIAS_CH3,BIAS_MON_V.DAC_BIAS_CH4,CONTINUE ,
          MESSAGE_CONFIRM_DAC);
3884     add_to_buffer (ltc_m_values.ltc_current,ltc_m_values.ltc_temperature, RUN_COMPLETE,
          MESSAGE_CONFIRM_DAC);
3885 }
3886
3887 int esm1_read()
3888 {
3889     unsigned int temp=0;
3890     unsigned int * ESM_address_pointer =(unsigned int *) 0xFFFFF518;
3891     wait(2);
3892     if(*ESM_address_pointer != 0)
3893     {
3894         unsigned int temp = *ESM_address_pointer;
3895         *ESM_address_pointer =temp;
3896     }
3897
3898   return temp;
3899 }
3900
3901 int esm2_read()
3902 {
3903
3904     unsigned int temp=0;
3905     unsigned int * ESM_address_pointer =(unsigned int *) 0xfffff51c;
3906     wait(2);
3907     if(*ESM_address_pointer != 0)
3908     {
3909         unsigned int temp = *ESM_address_pointer;
3910         *ESM_address_pointer =temp;
3911     }
3912
3913   return temp;
3914 }
3915
```

```
3916
3917 void ecc_data_test()
3918 {
3919     checkRAMECC();
3920
3921
3922 }
```

B.3 System Main

```
1  /** @file sys_main.c
2  *   @brief Application main file
3  *   @date 05-Oct-2016
4  *   @version 04.06.00
5  *
6  *   This file contains an empty main function,
7  *   which can be used for the application.
8  */
9
10 /*
11 * Copyright (C) 2009-2016 Texas Instruments Incorporated - www.ti.com
12 *
13 *
14 *  Redistribution and use in source and binary forms, with or without
15 *  modification, are permitted provided that the following conditions
16 *  are met:
17 *
18 *    Redistributions of source code must retain the above copyright
19 *    notice, this list of conditions and the following disclaimer.
20 *
21 *    Redistributions in binary form must reproduce the above copyright
22 *    notice, this list of conditions and the following disclaimer in the
23 *    documentation and/or other materials provided with the
24 *    distribution.
25 *
26 *    Neither the name of Texas Instruments Incorporated nor the names of
27 *    its contributors may be used to endorse or promote products derived
28 *    from this software without specific prior written permission.
29 *
30 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
31 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
32 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
33 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
34 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
35 *  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
36 *  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
37 *  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
38 *  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
39 *  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
40 *  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
41 *
42 */
43
44
45 /* USER CODE BEGIN (0) */
46 #include "sys_common.h"
47 #include "system.h"
48 #include "reg_system.h"
49 #include "stdio.h"
50 #include "sci.h"
51 #include "reg_sci.h"
52 #include "string.h"
53 #include "Commandline.h"
54 #include "SysBoard.h"
55 #include "htu.h"
56
57 /* USER CODE END */
58
59 /* Include Files */
60
61 #include "sys_common.h"
62
63 /* USER CODE BEGIN (1) */
64 // Received char
65 char c; //stores revceived char from sci receive
66
67
68
69 /* USER CODE END */
```

```
70
71 /** @fn void main(void)
72 *   @brief Application main function
73 *   @note This function is empty by default.
74 *
75 *   This function is called after startup.
76 *   The user can use this function to implement the application.
77 */
78
79 /* USER CODE BEGIN (2) */
80
81
82 /* USER CODE END */
83
84 int main(void)
85 {
86 /* USER CODE BEGIN (3) */
87     init();
88     //Ant_RLS_1_On;
89     //Ant_RLS_2_On;
90     //Ant_RLS_3_On;
91     //Ant_RLS_4_On;
92
93
94     while(1)
95     {
96         Parsing();
97     }
98 /* USER CODE END */
99 }
100
101
102 /* USER CODE BEGIN (4) */
103
104 /** @fn void sciNotification(sciBASE_t *sci, unsigned flags)
105 *   @brief standard function for SCI RX interrupt
106 *   @param[in] scilinREG is base address for modul
107 *   @param[in] SCI_RX_INT value
108 *
109 *   @return none
110 *
111 *   this module has to be initiated for compiler to initial inturrupt for SCI.
112 *   this functions reformates the received char from terminal and stores them
113 *   in char array called PromptChar[].
114 *
115 */
116 void sciNotification(sciBASE_t *sci, unsigned flags)
117 {
118
119
120    /*Wait for the Tx buffer to empty*/
121    if (SCI_TX_INT == (flags & SCI_TX_INT ))
122    {
123
124        Led1Toggle;
125        /*send new characters*/
126        sciReceive(sci,1,(unsigned char *)&c);
127
128
129    }
130
131    /*Wait for the Rx buffer to empty*/
132     else if (SCI_RX_INT == (flags & SCI_RX_INT ))
133     {
134
135        putChar(c);
136
137     }
138
139 }
140
141 /* Interrupt */
142 void esmGroup1Notification(int bit)
143 {
144     return;
145 }
146
147 void esmGroup2Notification(int bit)
148 {
149     return;
150 }
151
152
```

```
153
154
155 /* USER CODE END */
```

B.4 Firmware offset and Bootloader

The modification of firmware location

```
1  /*****************************************************************************/
2  /* sys_link.cmd                                                              */
3  /*                                                                           */
4  /*
5  * Copyright (C) 2009-2016 Texas Instruments Incorporated - www.ti.com
6  *
7  *
8  *  Redistribution and use in source and binary forms, with or without
9  *  modification, are permitted provided that the following conditions
10 *  are met:
11 *
12 *    Redistributions of source code must retain the above copyright
13 *    notice, this list of conditions and the following disclaimer.
14 *
15 *    Redistributions in binary form must reproduce the above copyright
16 *    notice, this list of conditions and the following disclaimer in the
17 *    documentation and/or other materials provided with the
18 *    distribution.
19 *
20 *    Neither the name of Texas Instruments Incorporated nor the names of
21 *    its contributors may be used to endorse or promote products derived
22 *    from this software without specific prior written permission.
23 *
24 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
25 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
26 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
27 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
28 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
29 *  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
30 *  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
31 *  DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
32 *  THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
33 *  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
34 *  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
35 *
36 */
37
38 /*                                                                           */
39 /*****************************************************************************/
40 /* USER CODE BEGIN (0) */
41 /* USER CODE END */
42
43
44 /*****************************************************************************/
45 /* Linker Settings                                                           */
46
47 --retain="*(.intvecs)"
48
49 /* USER CODE BEGIN (1) */
50 /* USER CODE END */
51
52 /*****************************************************************************/
53 /* Memory Map                                                                */
54
55 MEMORY
56 {
57     VECTORS (X)   : origin=0x00020000 length=0x00000020
58     FLASH0  (RX)  : origin=0x00020020 length=0x0013FFE0
59     STACKS  (RW)  : origin=0x08000000 length=0x00001500
60     RAM     (RW)  : origin=0x08001500 length=0x0002EB00
61     AJSM    (RX)  : origin=0xF0000000 length=0x00000010
62
63 /* USER CODE BEGIN (2) */
64 /* USER CODE END */
65 }
66
67 /* USER CODE BEGIN (3) */
68 /* USER CODE END */
69
70 /*****************************************************************************/
71 /* Section Configuration                                                     */
72
```

```
SECTIONS
{
    .intvecs : {} > VECTORS
    .text    : {} > FLASH0
    .const   : {} > FLASH0
    .cinit   : {} > FLASH0
    .pinit   : {} > FLASH0
    .bss     : {} > RAM
    .data    : {} > RAM
    .sysmem  : {} > RAM
  .ajsm    : {} > AJSM


/* USER CODE BEGIN (4) */
/* USER CODE END */
}

/* USER CODE BEGIN (5) */
/* USER CODE END */


/*-----------------------------------------------------------------------------*/
/* Misc                                                                        */

/* USER CODE BEGIN (6) */
/* USER CODE END */
/*-----------------------------------------------------------------------------*/
```

B.5 CRC16 packing and uart message

This not developed by me but rather custimized and integrated. The source of CRC16 and OBC hex commands are from the electronic lab.

```
/** @file obc_communication.c
*   @brief communication protocol
*   @date 10-mai-2018
*   @Author: Erlend baardsen ELAB
*
*   This file contains an empty main function,
*   which can be used for the application.
*/
#include "system.h"
#include "reg_system.h"
#include "stdio.h"
#include "sci.h"
#include "reg_sci.h"
#include "string.h"
#include "obc_communication.h"

static uint16_t calculate_crc16 (uint8_t *buf, size_t size) {

    uint16_t crc = 0;
    int i;

    while(size--) {

        crc = crc ^ *buf++ << 8;
        for (i=0; i<8; i++) {

            if (crc & 0x8000) {

                crc = (crc << 1) ^ 0x1021;
            } else {

                crc = crc << 1;
            }
        }
    }
    return crc;
}

static int message_send_uart (struct message_s * message) {

    int i;
    uint16_t crc;

```

```
    /* Copy payload_lenght to end of active pay-load
     * in order to simplify CRC calculation
     */
    if (message->payload_length < MESSAGE_PAYLOAD_LENGHT_MAX)
        message->payload[message->payload_length] = message->payload_length;

    crc = calculate_crc16 (&(message->all[1]), (message->payload_length + 2));

    /* Check that message is within the maximum */
    if (message->payload_length > MESSAGE_PAYLOAD_LENGHT_MAX)
        return EXIT_FAILURE;

    sciSendByte(scilinREG, MESSAGE_START);

    sciSendByte(scilinREG, message->id);

    for (i=0; i < message->payload_length; i++)
        sciSendByte(scilinREG, message->payload[i]);

    sciSendByte(scilinREG, message->payload_length);

    sciSendByte(scilinREG, (uint8_t)(crc >> 8));
    sciSendByte(scilinREG, (uint8_t)(crc >> 0));

    sciSendByte(scilinREG, MESSAGE_END);

    return EXIT_SUCCESS;
}

int obc_debug (char * format, ...) {

    struct message_s debug_message;

    va_list args;
    va_start (args, format);
    vsnprintf ((char *)&debug_message.payload, MESSAGE_PAYLOAD_LENGHT_MAX, format, args);
    va_end (args);
    debug_message.id = MESSAGE_INDICATE_DEBUG_TEXT;
    debug_message.payload_length = strlen((const char*)&debug_message.payload);
    message_send_uart (&debug_message);

    return EXIT_SUCCESS;
}

int obc_raw_bytes (uint8_t *buffer, size_t size,uint8 message_id) {

    struct message_s debug_message;
    int i;

    debug_message.id = message_id;
    debug_message.payload_length = size;

    for (i=0; ((i < size) && (i < MESSAGE_PAYLOAD_LENGHT_MAX)); i++)
        debug_message.payload[i] = buffer[i];

    message_send_uart (&debug_message);

    return EXIT_SUCCESS;
}
```

```
#ifndef OBC_COMMUNICATION_H
#define OBC_COMMUNICATION_H

#include <stdint.h>
#include <stdarg.h>
#include <stdlib.h>

int obc_debug (char * format, ...);
int obc_raw_bytes (uint8_t * buffer, size_t size, uint8 message_id);

/* Communication model:
 *                      |                |
 * SERVICE USER         | SERVICE        |   SERVICE USER
 *                      | PROVIDER       |
 *                      |                |
 *  Request             |                |   Indication
 *  ------------------> |                | ------------------>
 *                      |                |
 *  Confirmation        |                |   Response
 *  <------------------ |                | <------------------
 *                      |                |
 *
 * Confirmation is the acknowledgment of the reception
```

```
24  * of a request.
25  *
26  * Response is the acknowledgment of the reception
27  * of an indication.
28  *
29  */
30
31 enum buffer_state_e {NEW_RUN, CONTINUE, RUN_COMPLETE};
32
33 /* Indications for test usage */
34 #define MESSAGE_INDICATE_DEBUG_TEXT              (0x3E)
35 #define Message_TMS_REG                          (0x3F)
36 #define MESSAGE_CONFIRM_BOARD_STATUS             (0x22)
37 #define MESSAGE_CONFIRM_ADC                      (0x39)
38 #define MESSAGE_CONFIRM_DAC                      (0x33)
39
40 #define MESSAGE_START                            (0xaf)
41 #define MESSAGE_END                              (0x5f)
42
43 #define MESSAGE_ALL_LENGHT_MAX                   (0xff)
44 #define MESSAGE_NON_PAYLOAD_LENGHT               (6)
45 #define MESSAGE_PAYLOAD_LENGHT_MAX               (MESSAGE_ALL_LENGHT_MAX -
       MESSAGE_NON_PAYLOAD_LENGHT)
46
47 #define INCOMING_BUFFER_LENGHT                   (MESSAGE_ALL_LENGHT_MAX * 2)
48
49 /* CRC calculation include: id, pay-load and payload_lenght
50  * start and end byte is not included in the calculation
51  *
52  * Polynomial is 0x1021 and initial value is 0
53  * (XMODEM/YMODEM CRC16)
54  *
55  */
56 struct message_s {
57     union {
58         struct {
59             uint8_t start;
60             uint8_t id;
61             uint8_t payload[MESSAGE_PAYLOAD_LENGHT_MAX];
62             uint8_t payload_length;
63             uint8_t crc_msb;
64             uint8_t crc_lsb;
65             uint8_t end;
66         };
67         uint8_t all[MESSAGE_ALL_LENGHT_MAX];
68     };
69 };
70
71 #endif /* OBC_COMMUNICATION_H */
```

C | Radiation Testing

C.1 Power supply

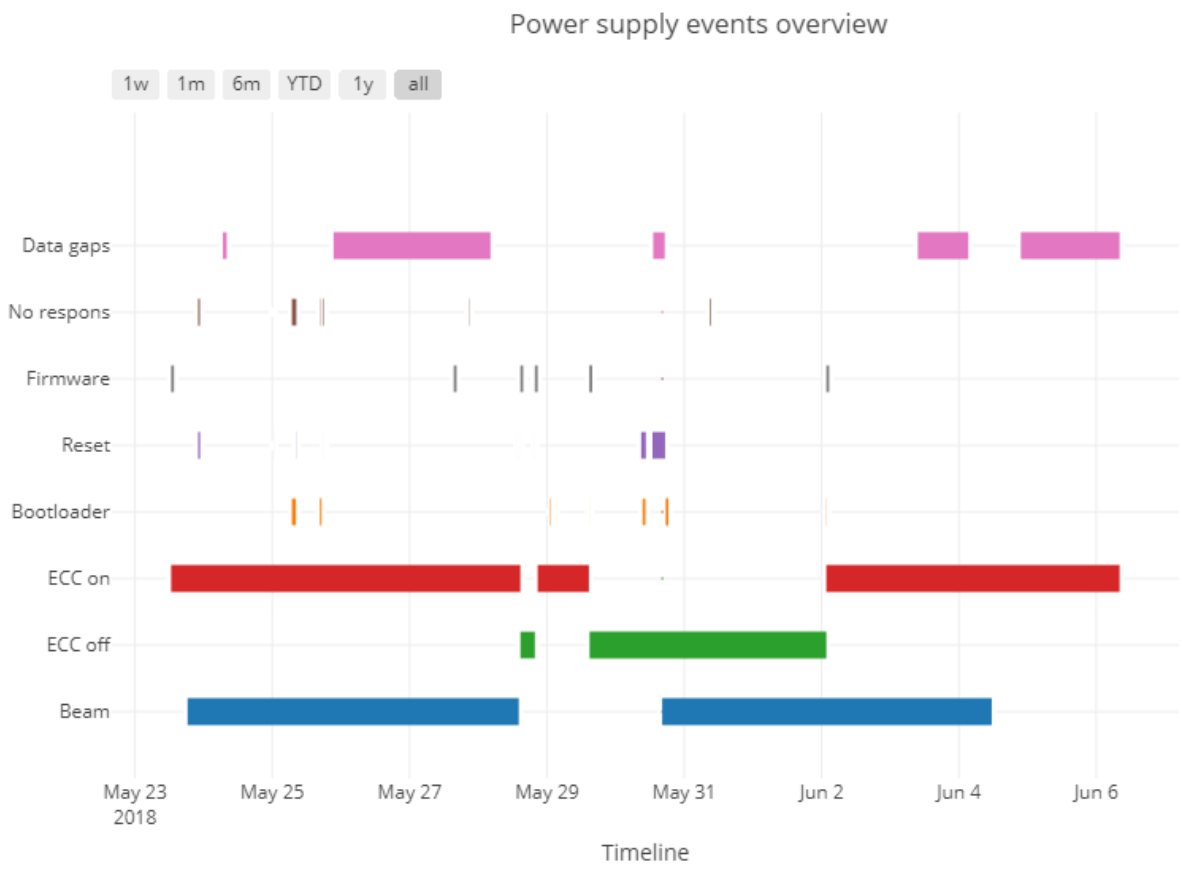


Figure C.1: Event that effect the Power supply data collection

C.2 m-NLP events

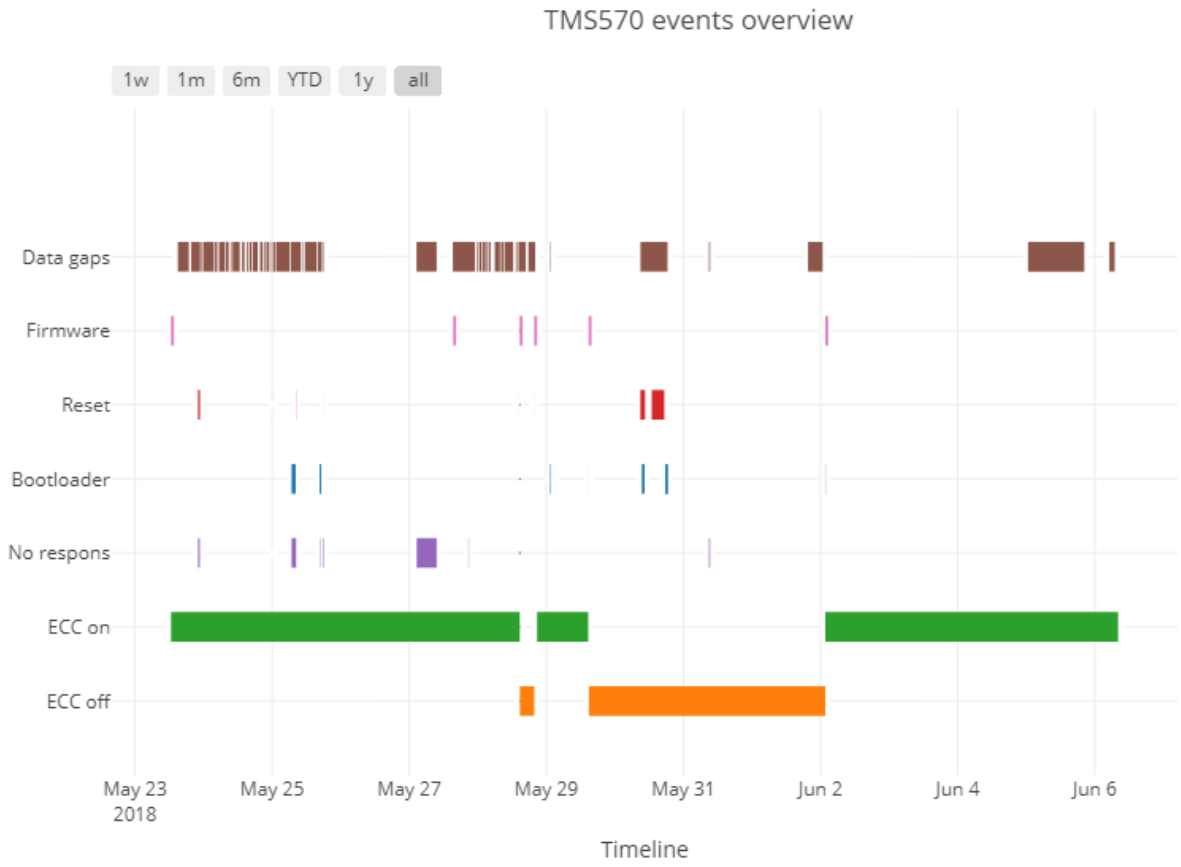


Figure C.2: TMS570 relevent events that affected data collection

Bibliography

[1] Chris-martin, "Van Allen radiation belts," Jul. 2018. [Online]. Available: https://en.wikipedia.org/wiki/Van_Allen_radiation_belt#/media/File:Van_Allen_radiation_belt.svg

[2] "Spenvis: Trapped particle radiation models." [Online]. Available: https://www.spenvis.oma.be/help/background/traprad/traprad.html#APAE

[3] J. Barth, C. Dyer, and E. Stassinopoulos, "Space, atmospheric, and terrestrial radiation environments," *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 466–482, Jun. 2003. [Online]. Available: http://ieeexplore.ieee.org/document/1208571/

[4] R. Secondo, R. Alia, P. Peronnard, M. Brugger, A. Masi, S. Danzeca, A.-S. Merlenghi, L. Dusseau, and J.-R. Vaille, "Analysis of SEL on Commercial SRAM Memories and Mixed-Field Characterization of a Latchup Detection Circuit For LEO Space Applications," *IEEE Transactions on Nuclear Science*, pp. 1–1, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7892958/

[5] Joseph Magill, "Nucleonica web portal for nuclear data," Mar. 2011. [Online]. Available: https://www.nucleonica.com

[6] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, Sep. 2005. [Online]. Available: http://ieeexplore.ieee.org/document/1545891/

[7] F. Faccio, "COTS for the LHC radiation environment: the rules of the game," p. 16. [Online]. Available: http://lhc-electronics-workshop.web.cern.ch/LHC-electronics-workshop/2000/plenary/faccio_plenary.pdf

[8] M. Nicolaidis, Ed., *Soft errors in modern electronic systems*, ser. Frontiers in electronic testing. New York: Springer, 2011, no. 41, oCLC: ocn699778302.

[9] A. Schlösser, D. Nedospasov, J. Krämer, S. Orlic, and J.-P. Seifert, "Simple photonic emission analysis of AES," *Journal of Cryptographic Engineering*, vol. 3, no. 1, pp. 3–15, Apr. 2013. [Online]. Available: http://link.springer.com/10.1007/s13389-013-0053-7

[10] TMS570LS12x/11x, "TMS570 technical reference manual TMS570ls12x," Apr. 2015.

[11] J. Mekki, M. Brugger, R. G. Alia, A. Thornton, N. C. D. S. Mota, and S. Danzeca, "CHARM: A Mixed Field Facility at CERN for Radiation Tests in Ground, Atmospheric, Space and Accelerator Representative Environments," *IEEE Transactions on Nuclear Science*, vol. 63, no. 4, pp. 2106–2114, Aug. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7508970/

[12] G. F. Knoll, *Radiation detection and measurement*, 4th ed. Hoboken, N.J: John Wiley, 2010.

[13] Yaqi Jin, "Characterization of GPS Scintillations in the Polar Ionosphere," *Phd, University of Oslo*, pp. 1–3, May 2016.

[14] T. A. Bekkeng, "Protype Development of a Multi-Needle Langmuir Probe System." [Online]. Available: https://www.duo.uio.no/bitstream/handle/10852/11230/bekkeng.pdf?sequence=1

[15] K. S. Jacobsen, A. Pedersen, J. I. Moen, and T. A. Bekkeng, "A new Langmuir probe concept for rapid sampling of space plasma electron density," *http://iopscience.iop.org*, vol. 21, no. 8, p. 9, Jul. 2010. [Online]. Available: http://iopscience.iop.org/article/10.1088/0957-0233/21/8/085902/meta#citations

[16] K. A. LaBel and M. J. Sampson, "Thoughts on Commercial Off the Shelf (COTS) Electronics for Space," the NASA Electronic Parts and Packaging Program (NEPP) Electronics Technology Workshop. [Online]. Available: https://nepp.nasa.gov/workshops/etw2013/talks/Tue_June11_2013/1030_LaBel_Sampson_Thoughts%20on%20COTS%20Electronics%20for%20Space.pdf

[17] *Space product assurance : Technique for radiation effects mitigation in ASICs and FPGAs handbook*. ECSS Secretariat ESA-ESTEC, Nov. 2016.

[18] M. Bagatin, S. Gerardin, and K. Iniewski, Eds., *Ionizing radiation effects in electronics: from memories to imagers*, ser. Devices, circuits, and systems.

Boca Raton ; London ; New York: CRC Press : Taylor & Francis Group, 2016, oCLC: ocn908375937.

[19] Rubén GARCÍA ALÍA, "Radiation Fields in High Energy Accelerators and their impact on Single Event Effects," Ph.D. dissertation, UNIVERSITE MONTPELLIER 2, Dec. 2014. [Online]. Available: https://cds.cern.ch/record/2012360?ln=en

[20] R. Secondo, R. G. Alia, P. Peronnard, M. Brugger, A. Masi, S. Danzeca, A. Merlenghi, E. Chesta, J. R. Vaille, M. Bernard, and L. Dusseau, "System Level Radiation Characterization of a 1u CubeSat Based on CERN Radiation Monitoring Technology," *IEEE Transactions on Nuclear Science*, pp. 1–1, 2018. [Online]. Available: http://ieeexplore.ieee.org/document/8268561/

[21] "NORSAT 1 - Orbit." [Online]. Available: https://www.heavens-above.com/satinfo.aspx?satid=42826&lat=0&lng=0&loc=Unspecified&alt=0&tz=UCT

[22] J. Fraden, *Handbook of modern sensors: physics, designs, and applications*, 5th ed. Cham Heidelberg New York Dordrecht London: Springer, 2016, oCLC: 930757614.

[23] H. H. K. Tang, "Nuclear physics of cosmic ray interaction with semiconductor materials: Particle-induced soft errors from a physicist's perspective," *IBM Journal of Research and Development*, vol. 40, no. 1, pp. 91–108, Jan. 1996. [Online]. Available: http://ieeexplore.ieee.org/document/5389442/

[24] H. H. Tang and K. P. Rodbell, "Single-Event Upsets in Microelectronics: Fundamental Physics and Issues," *MRS Bulletin*, vol. 28, no. 02, pp. 111–116, Feb. 2003. [Online]. Available: http://www.journals.cambridge.org/abstract_S0883769400017504

[25] N. Seifert, B. Gill, K. Foley, and P. Relangi, "Multi-cell upset probabilities of 45nm high-k + metal gate SRAM devices in terrestrial and space environments," in *2008 IEEE International Reliability Physics Symposium*. IEEE, Apr. 2008, pp. 181–186. [Online]. Available: http://ieeexplore.ieee.org/document/4558882/

[26] C. C. Foster, "Total Ionizing Dose and Displacement-Damage Effects in Microelectronics," *MRS Bulletin*, vol. 28, no. 02, pp. 136–140, Feb. 2003. [Online]. Available: http://www.journals.cambridge.org/abstract_S0883769400017553

[27] M. A. Clemens, B. D. Sierawski, K. M. Warren, M. H. Mendenhall, N. A. Dodds, R. A. Weller, R. A. Reed, P. E. Dodd, M. R. Shaneyfelt, J. R. Schwank, S. A. Wender, and R. C. Baumann, "The Effects of Neutron Energy and High-Z Materials on Single Event Upsets and Multiple Cell Upsets," *IEEE Transactions on Nuclear Science*, vol. 58, no. 6, pp. 2591–2598, Dec. 2011. [Online]. Available: http://ieeexplore.ieee.org/document/6068268/

[28] R. G. Alia, M. Brugger, M. Cecchetto, F. Cerutti, S. Danzeca, M. Delrieux, M. Kastriotou, M. Tali, and S. Uznanski, "RSEE testing in the 24 GeV proton beam at the CHARM facility," *IEEE Transactions on Nuclear Science*, pp. 1–1, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8347062/

[29] J. Schwank, M. Shaneyfelt, J. Baggio, P. Dodd, J. Felix, V. Ferlet-Cavrois, P. Paillet, D. Lambert, F. Sexton, G. Hash, and E. Blackmore, "Effects of particle energy on proton-induced single-event latchup," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2622–2629, Dec. 2005. [Online]. Available: http://ieeexplore.ieee.org/document/1589248/

[30] P. Dodd, M. Shaneyfelt, J. Schwank, and G. Hash, "Neutron-induced latchup in SRAMs at ground level." IEEE, 2003, pp. 51–55. [Online]. Available: http://ieeexplore.ieee.org/document/1197720/

[31] K. LaBel and M. Gates, "Single-event-effect mitigation from a system perspective," *IEEE Transactions on Nuclear Science*, vol. 43, no. 2, pp. 654–660, Apr. 1996. [Online]. Available: http://ieeexplore.ieee.org/document/490908/

[32] TMS570LS12x, "Safety Manual for TMS570ls12x and 11x Hercules™ ARM®-Based Safety Critical Microcontrollers," Dec. 2015.

[33] W. J. Greig, *Integrated circuit packaging, assembly, and interconnections*. New York: Springer, 2006, oCLC: ocm62132603.

[34] "Texas Instruments Quality & reliability: Materials data base," Aug. 2018. [Online]. Available: http://www.ti.com/materialcontent/en/report?pcid=249877&opn=TMS5701227CPGEQQ1

[35] M. J. Lefevre, F. Beauquis, J. Yang, M. Obein, P. Gounet, and S. Barberan, "New method for decapsulation of copper wire devices using LASER and sub-ambient temperature chemical etch." IEEE, Dec. 2011, pp. 769–773. [Online]. Available: http://ieeexplore.ieee.org/document/6184523/

[36] H. Wu, Y. Liang, W. Du, S. He, Y. Wang, and D. Lei, "Study of silver alloy wire decapsulation methods for failure analysis." IEEE, Jul. 2017, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/8060206/

[37] "Health and Safety and the Environment For staff and students." [Online]. Available: https://www.mn.uio.no/kjemi/om/hms/hse-manual-department-of-chemistry-2018-01.pdf

[38] Dmitry Nedospasov, "SECURITY OF THE IC BACKSIDE," Thesis, Universität Berlin, Germany, 2014. [Online]. Available: http://users.sec.t-labs.tu-berlin.de/~nedos/Nedospasov_Thesis.pdf

[39] "Technical datasheet AD7768."

[40] "Technical datasheet LTC3887." [Online]. Available: http://www.analog.com/media/en/technical-documentation/data-sheets/3887fd.pdf

[41] T. instruments, "DAC104s085-xx 10-Bit Micro Power QUAD Digital-to-Analog Converter With Rail-to-Rail Output," May 2016. [Online]. Available: http://www.ti.com/lit/ds/symlink/dac104s085.pdf

[42] TMS570LS1224, "TMS570ls1224 16- and 32-Bit RISC Flash Microcontroller overview," Feb. 2015.

[43] Jáno Gebelein, "FPGA Fault Tolerance in Radiation Environments," Phd thesis, JohannWolfgang Goethe University, Frankfurt Germany, 2016. [Online]. Available: https://indico.gsi.de/event/5339/contribution/2/material/0/0.pdf

[44] G. Tsiligiannis, S. Danzeca, R. Garcia-Alia, A. Infantino, A. Lesea, M. Brugger, A. Masi, S. Gilardoni, and F. Saigne, "Radiation Effects on Deep Submicrometer SRAM-Based FPGAs Under the CERN Mixed-Field Radiation Environment," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1511–1518, Aug. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8292957/

[45] R&S®, "R&S®HMC804x Power Supply SCPI Programmers Manual," Apr. 2016. [Online]. Available: https://www.rohde-schwarz.com/us/manual/r-s-hmc804x-power-supply-scpi-programmers-manual-manuals-gb1_78701-172161.html

[46] PyVISA Authors, "PyVISA Documentation," Aug. 2018. [Online]. Available: https://media.readthedocs.org/pdf/pyvisa/latest/pyvisa.pdf

[47] K. McCarty, J. Coss, D. Nichols, G. Swift, and K. LaBel, "Single event effects testing of the Crystal CS5327 16-bit ADC," in *Workshop Record. 1994 IEEE Radiation Effects Data Workshop*. Tucson, AZ, USA: IEEE, 1994, pp. 86–96. [Online]. Available: http://ieeexplore.ieee.org/document/633040/

[48] D. Hiemstra, S. Yu, and M. Pop, "Single event upset characterization of a personal computer micro-controller system-on-a-chip using proton irradiation," in *2003 IEEE Radiation Effects Data Workshop*. Monterey, CA, USA: IEEE, 2003, pp. 108–112. [Online]. Available: http://ieeexplore.ieee.org/document/1281358/

[49] P. O'Neill, G. Badhwar, and W. Culpepper, "Internuclear cascade-evaporation model for LET spectra of 200 MeV protons used for parts testing," *IEEE Transactions on Nuclear Science*, vol. 45, no. 6, pp. 2467–2474, Dec. 1998. [Online]. Available: http://ieeexplore.ieee.org/document/736487/

[50] P. Dodd, M. Shaneyfelt, J. Schwank, and G. Hash, "Neutron-induced soft errors, latchup, and comparison of SER test methods for SRAM technologies." IEEE, 2002, pp. 333–336. [Online]. Available: http://ieeexplore.ieee.org/document/1175846/

[51] H. Quinn, T. Fairbanks, J. L. Tripp, G. Duran, and B. Lopez, "Single-Event Effects in Low-Cost, Low-Power Microprocessors," in *2014 IEEE Radiation Effects Data Workshop (REDW)*. Paris, France: IEEE, Jul. 2014, pp. 1–9. [Online]. Available: http://ieeexplore.ieee.org/document/7004596/

[52] Karl Greb and Dev Pradhan, "Hercules™ Microcontrollers: Real-time MCUs for safety-critical products white paper," Sep. 2011. [Online]. Available: http://www.ti.com/lit/wp/spry178/spry178.pdf