Trabajo Hyper-Kamiokande

Descripción detallada del trabajo realizado

Borja Martínez Sánchez

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Introduction:TFG Objectives

It is well known that telecommunications have played a pivotal role in the course of society since its emergence, being a discipline that nowadays is necessary in almost everywhere, whether for 2-person telephone communication or for military applications.

The field of telecommunications encompasses several branches of knowledge, one of them being electronic systems, where I specialize.
Therefore, this Final Degree Thesis of the Degree in Telecommunication Technologies and Services Engineering is a clear evidence of the importance of electronic systems in the advancement of human beings to explore and investigate the behaviour of the universe in all of its aspects.

This Final Degree Thesis is part of the Hyper-Kamiokande (HKK) project, which will be described in more detail in the following sections. It is a massive global project that started in 2018 with around 300 researchers from 15 different countries, and over time the number has grown, which means that project coordination is crucial in the development of the whole project.
The responsibilities of the UPV form the nucleus of the experiment's electronics, given the module's design originating from the UPV, particularly the Institute for Molecular Imaging Technologies (i3M). This TFG can be seen as a step forward in the development of the Date Processing Board (DPB) software, making use of the tools available and those provided by the HKK project partners. The objectives of this dissertation are summarised in the following list:

- **Develop DPB software** in order to allow us to read the information collected by the sensors, process commands coming from the Data Adquisition Module (DAQ) and transmit the gathered data as packets to the DAQ.
- **Learn how to work in embedded environments** as it is the AMD Vitis software platform, which allow us to develop C or C++ application code that will run on ARM processors. Therefore, we will be able to debug and run code on our DPB System on Module (SoM).
- **Programme software in Linux** for our DPB as it runs on an embedded OS derived from Linux, Petalinux. Therefore to develop software that will run on this OS, Linux drivers will be used and modified if necessary to achieve the desired functionality. The operation and execution flow of the drivers themselves must be understood in order to be used.
- **Develop *slow control* applications** with the aim of precisely monitoring and managing low-frequency signals or events from the DPB, prioritizing stability and accuracy over real-time responsiveness.
- **Create date structures** by using JavaScript Object Notation (JSON) format so as to parse the gathered information and be able to communicate with the DAQ by following these data structures.
- **Manage alarm systems** asserted by the sensing components in the board so as to be able to act and report in case any of the components are in an undesired working area and may compromise the operation or reliability of the DPB.
- **Test preparation and automation** using the Robot framework for testing the mass production of boards (about 900 will be produced for the detector). The aim is to integrate the previously designed software into the test software and to prepare and enumerate the test cases in the Robot framework in order to be able to verify all necessary test cases automatically.

The neutrino itself

Within the field of physics there are countless subfields that study different aspects of everything around us. The project on which this dissertation is based is based on the speciality of physics called *Particle physics*, which is also known as high-energy physics, because many of these particles can only be seen in large collisions provoked in particle accelerators. This discipline of physics is responsible for demonstrating the existence of particles classified according to certain characteristics as bosons or fermions. Nonetheless, it also encounters the difficulty of having been able to demonstrate particles that are almost non-detectable to this day.

Within these elusive particles lies the neutrino, a subatomic entity generated during a radioactive decay and scattering phenomenon. In this instance, the neutrino arises from beta decay, as proposed in Fermi's theory, wherein a sizeable neutral particle (n) disintegrates into a proton (p^+), an electron (e^-), and a neutrino (v_e).

$$n- > p^+ + e^- + v_e$$

(1.1)

The first person to postulate the existence of the neutrino theoretically was Wolfgang Pauli in 1930, but it remained undetected for 25 years because this hypothetically predicted particle had to be massless, chargeless and without strong interaction.

Finally, in 1956, Clyde Cowan, Frederick Reines, Francis B. "Kiko" Harrison, Herald W. Kruse, and Austin D. McGuire were able to demonstrate the existence of the neutrino experimentally by using a beam of neutrons to pump a tank of pure water. By observing the subsequent emission of the protons, they were able to demonstrate the existence of the neutrino. This test was called the neutrino experiment.

Over the years, different types of radioactive decays have been discovered that can give rise to neutrinos, such as natural and artificial nuclear reactions, supernova events or the spin-down of a neutron star. Furthermore, it has been discovered that there are different leptonic flavours of neutrinos originating from the weak interactions, electron neutrino, muon neutrino and tau neutrino,each flavor is associated with the correspondingly named charged lepton and similar to some other neutral particles, neutrinos oscillate between different flavors in flight as a consequence.

The discipline that studies the phenomena caused by neutrinos from space encounters the difficulty of detecting neutrinos because they interact with almost nothing or only weakly. A configuration for detecting a decent amount of neutrinos will be explained in the next section. This precise configuration is the basis of the project on which this TFG has been developed.

Hyper-Kamiokande Project (HKK)

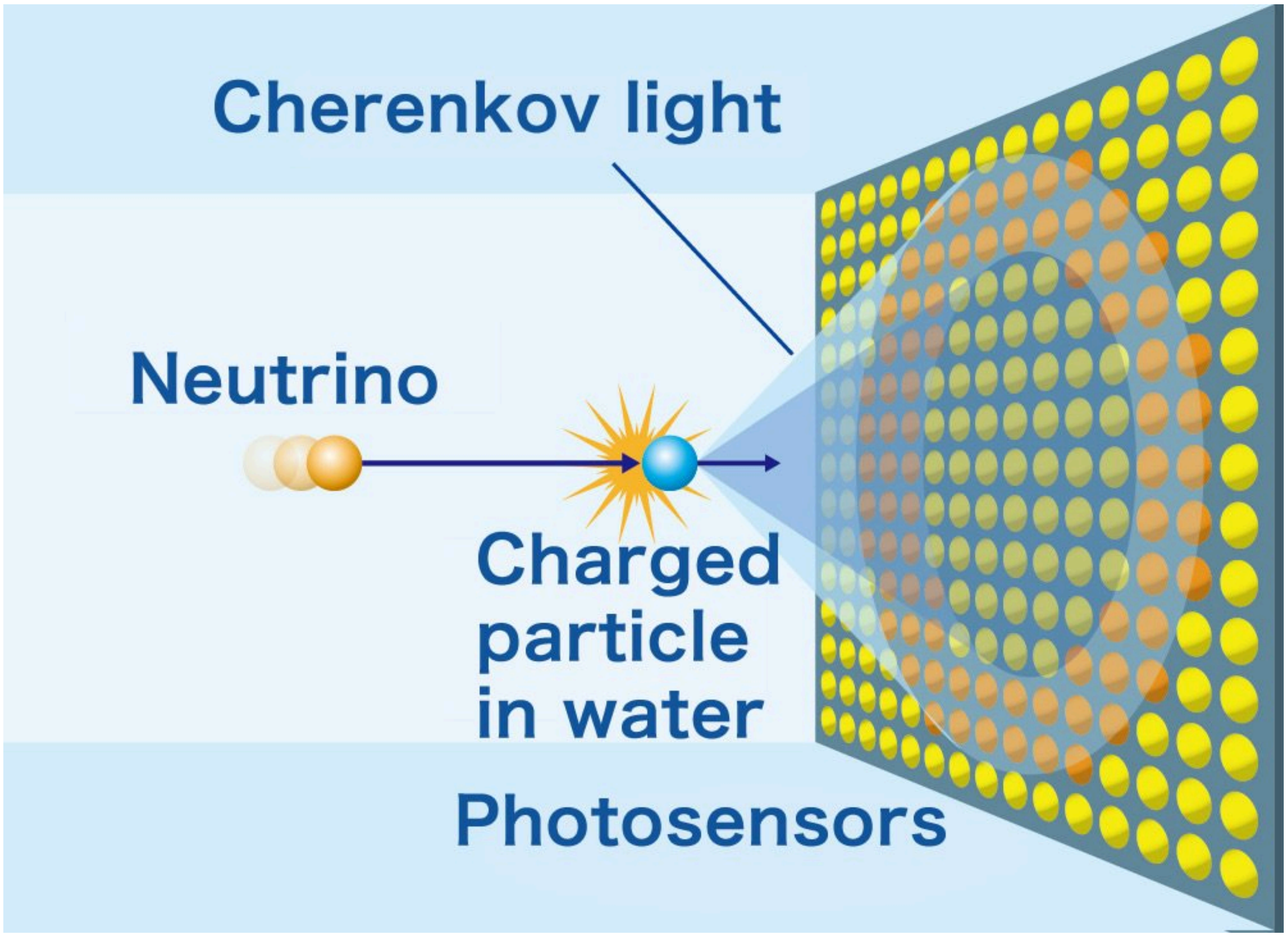
Hyper-Kamiokande Project (HKK) physical basis and its predecessor Super-Kamiokande

The physics apparatus used to study neutrinos is referred to as a neutrino detector, built to be isolated from any other influence like cosmic rays or background radiation .These neutrino detectors are huge structures that work following a neutrino detection technique of the existent ones let it be scintillators (like in the Cowan-Reines neutrino experiment), radiochemical methods, radio detectors or *Cherenkov light* detectors.The experiment that gives name to this chapter is based on the latter: the *Cherenkov light* detection.

This detectors are huge water-filled tanks enriched with deuterium and gadolinium. This medium is ideal for neutrino interaction as the interaction of one of this subatomic particles with the electrons or nuclei of water can produce a charged particle faster than the speed of light in water. This produces a cone of light called *Cherenkov light* and can be defined as the equivalent of light to a sonic boom in acoustic waves.

The water tank is surrounded by photosensible sensors called Phototubes, a cell filled with gas or a vacuum tube sensitive to light. The most used king of phototube is the Photomultipliertube (PMT) due to its high sensitiveness.

This PMT detects the *Cherenkov light* produced by the neutrino interaction. By sensing the pattern of light many information of the neutrino can be inferred, such as direction, energy and sometimes the flavor information of the incident neutrino.

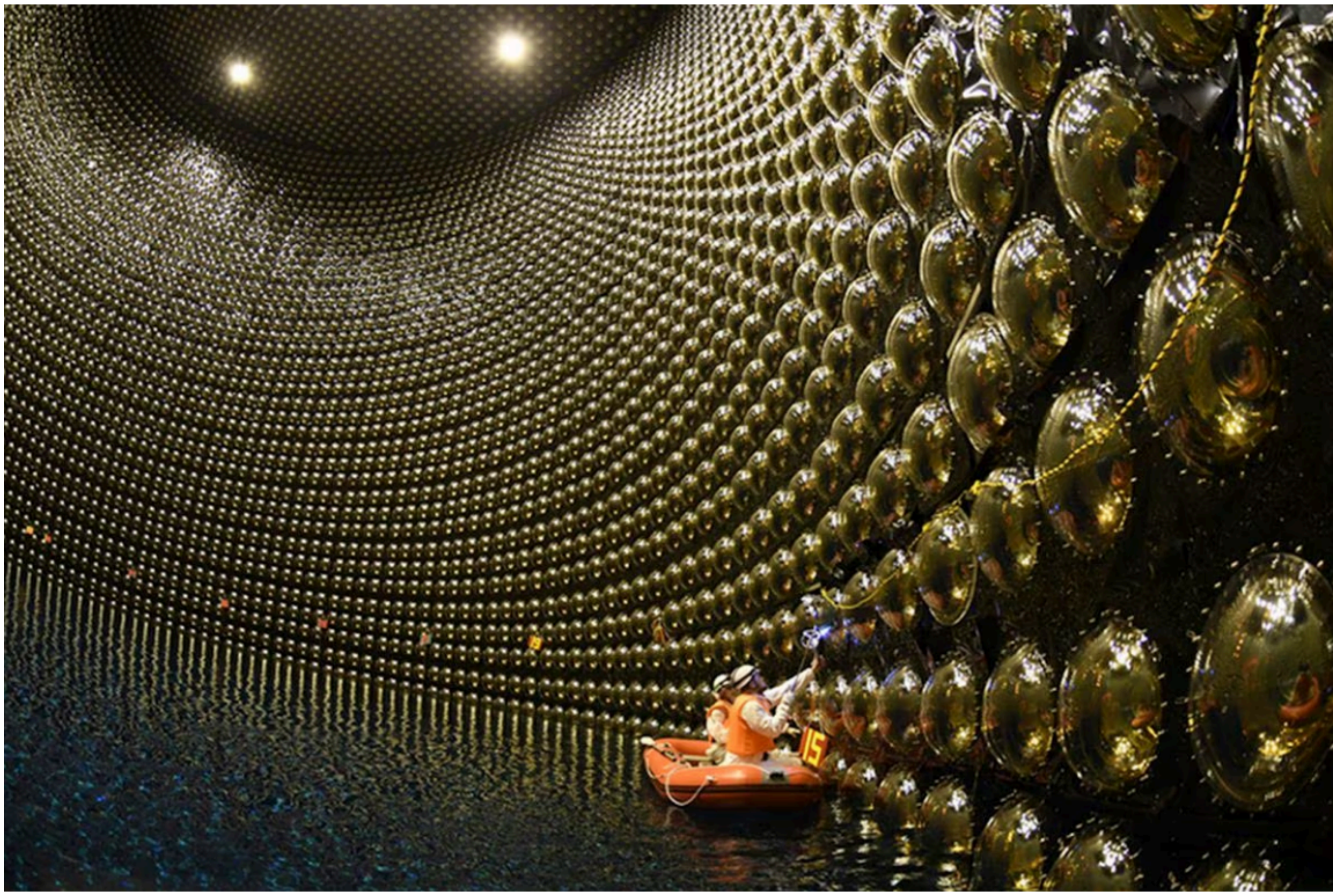


How a PMT detects *Cherenkov light* phenomenon

.

These detections are exceedingly rare due to the low probability of a neutrino interacting with matter. Therefore, the larger the water tank and the greater the number of PMTs, the more interactions can be detected within the same timeframe.

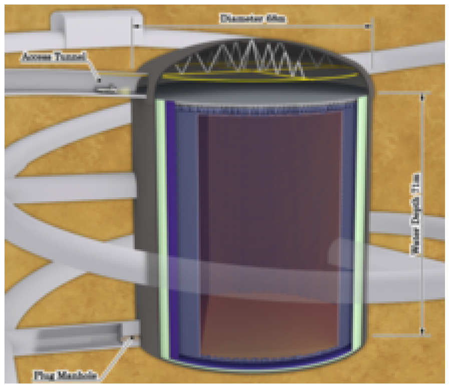
The largest neutrino detector currently in operation is the Super-Kamiokande (SKK). "Kamiokande" is a fusion of several words: KAMIOKA Neutrino Detection Experiment. Situated beneath Mount Ikeno near the city of Hida in the Gifu Prefecture, Japan, Kamioka is the facility that oversees this detector. The detector has undergone up to four revisions for various reasons, such as cascade failures or the replacement of the 6000 PMTs, along with upgrades to electronics in the latest iteration, Super-Kamiokande IV. These versions have not led to an increase in the number of PMTs or their percentage of coverage, but rather to measures to protect the technology used.


Interior of the Super-Kamiokande, predecessor of the Hyper-Kamiokande

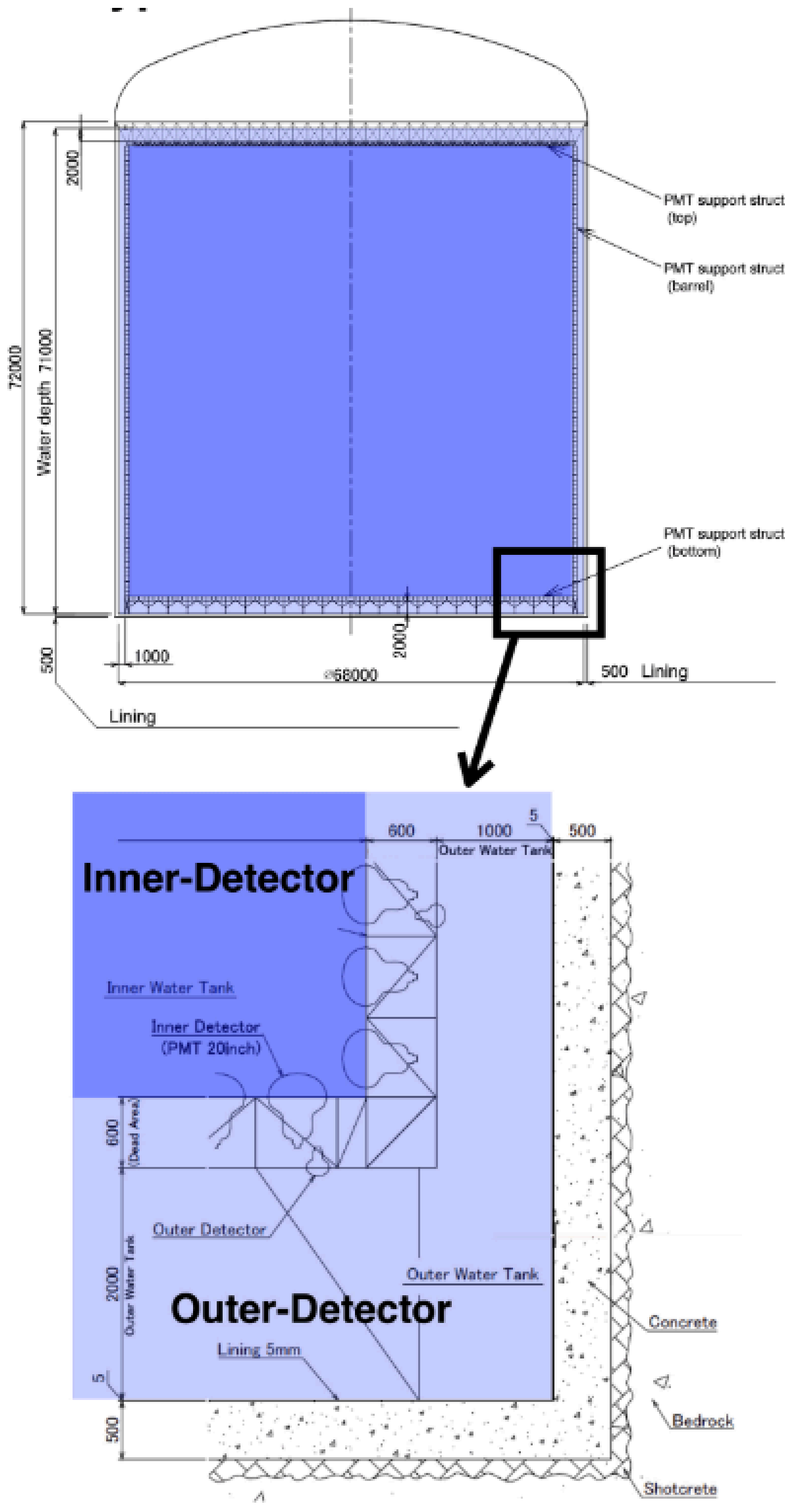
Structure of the HKK

Hyper-Kamiokande is a neutrino detector project still under construction (estimated to start operation in 2027), which takes place in the Kamioka mines in Japan replacing the Super-Kamiokande. Although the project is based in Japan, it involves research institutes from 22 different countries. The aim of the project is to search for anti-neutrinos coming from supernovas, proton decays and detect neutrinos from natural sources such as the Earth, the atmosphere, the Sun and the cosmos, as well as to study neutrino oscillations from the neutrino beam of the artificial accelerator.

Hyper-Kamiokande is planned to be the world's largest neutrino detector, surpassing its predecessor Super-Kamiokande, which is 71 metres high and 68 metres in diameter. The detector, filled with ultrapure water, will have about 40,000 photomultiplier tubes as detectors inside the detector and 10,000 detectors outside the detector. Although HKK is bigger than SKK, by including almost 4 times the number of PMTs of its predecessor, HKK achieves a 40% photocathode coverage, the same as SKK.



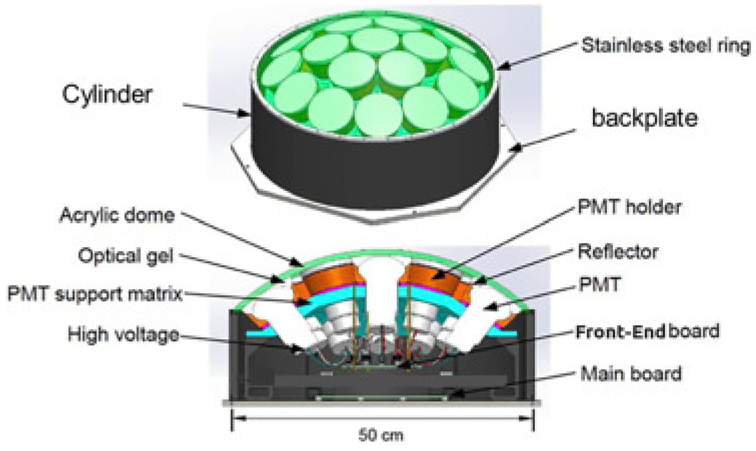
HKK water tank concept sketch



Cross-section of the Hyper-Kamiokande

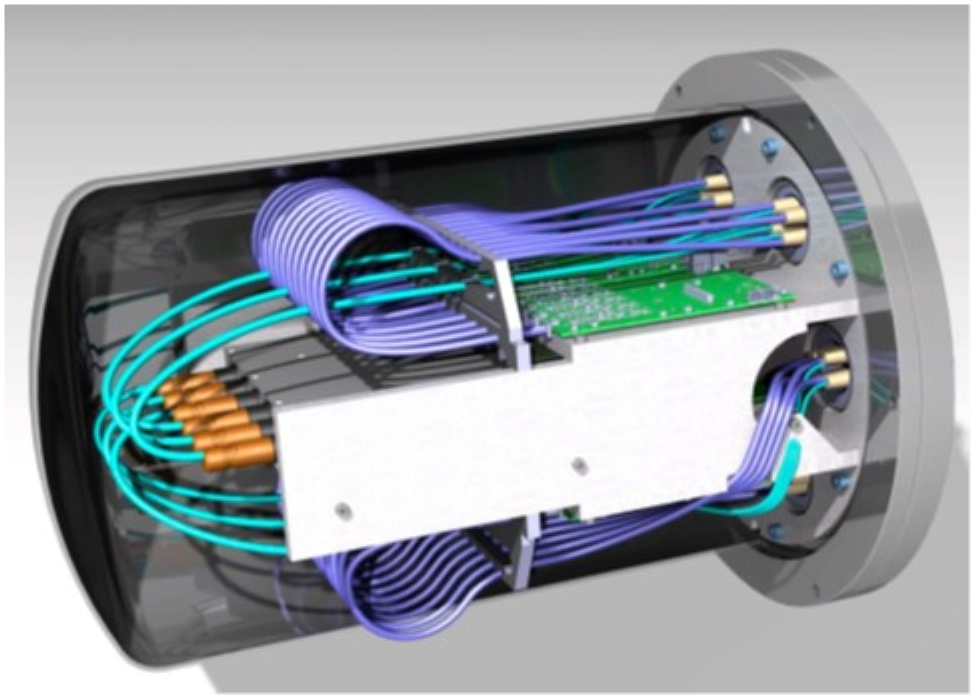
The detector design comprises a cylindrical tank with outer dimensions of 78 meters in height and 68 meters in diameter. It is filled with 260,000 metric tons of ultrapure water to create a water

Cherenkov detector. This tank will be surrounded by highly sensitive photodetectors, which boast a 50% higher efficiency compared to the SKK ones, thus allowing for greater precision in measuring light intensity and detection time. These photomultiplier tubes (PMTs), specifically the Hamatsu R12860 model, will enhance the detection of signatures such as those produced in neutrino interactions. Consequently, this setup will enable researchers to more accurately measure the direction and velocity of neutrinos passing through the detector.



PMT structure

The PMTs, along with the rest of the electronics, will be housed in hermetically sealed vessels submerged in the water inside the detector, following the same structure as the SKK.



Interior of the vessel

As can be seen in the previous figure, the electronics are concentrated inside the vessel, where the information from the PMTs is sampled in the digitizers and then sent to the DPB. The DPB is responsible for communicating the different modules both outside and inside the vessel, it acts as a hub inside the vessel.

Since the electronics are located in a place that is difficult to access, as it would mean emptying the detector of water, high reliability is required in this project, at least 10 years. For this reason, robust systems have been chosen and the electronics used must be monitored.

HKK objectives

The HKK experiment stands at the forefront of contemporary neutrino research, poised to unlock profound insights into the fundamental properties of these elusive particles. With its innovative design and enhanced capabilities, HKK ventures into uncharted territories of particle physics, aiming to shed light on mysteries ranging from neutrino oscillations to the enigmatic nature of dark matter.

In the realm of neutrino oscillation measurements, HKK endeavors to employ both accelerator and atmospheric neutrinos to unravel mysteries such as determining the mass hierarchy, investigating CP violation in the lepton sector, and precisely measuring oscillation parameters. Additionally, it aims to explore phenomena such as sterile neutrinos and potential violations of Lorentz invariance.

In solar neutrino measurements, HKK aims to address discrepancies observed between solar and reactor neutrino measurements, particularly focusing on the θ_{12} sector. It intends to achieve this by studying day-night asymmetry in solar neutrino flux and exploring novel avenues such as monitoring solar fusion reactions and observing higher-energy neutrino flux.

HKK seeks to build upon the nucleon decay research legacy of Kamiokande and SKK (Super-Kamiokande) by significantly enhancing limits on proton decays. It plans to utilize advanced PMT technology to improve performance, especially in detecting gamma rays from neutron capture, which is crucial for reducing neutrino backgrounds in proton decay searches.

In terms of supernova burst neutrinos, HKK aims to detect and analyze a substantial number of neutrinos from Galactic supernovae, allowing for detailed studies of these explosive events. It complements other experiments such as DUNE in its sensitivity to various types of supernova neutrinos.

For supernova relic neutrinos, HKK could contribute significantly by focusing on higher energy regions, complementing the efforts of SKK. Introducing gadolinium enhances sensitivity by distinguishing neutrino interactions from background events.

In dark matter searches, HKK aims to improve upon SKK's capabilities in detecting dark matter through neutrino signals, particularly from neutralino annihilation in regions of high dark matter density like the core of the Sun and the Galactic center. It also seeks to detect low-mass neutralinos, which are challenging to detect in direct-detection experiments.

In the pursuit of understanding the universe at its most fundamental level, the HKK experiment represents a beacon of scientific exploration. Through its multifaceted approach and collaborative efforts, HKK is poised to unravel some of the most profound mysteries of the cosmos, shaping our understanding of particle physics for generations to come.

Project organization

As it has been mentioned previuously, the HKK project involves research institutes from all over the world, which are in charge of different tasks within the project whether they are related to physics, electronics or any other relevant field. The project tasks have been divided in 7 different FD (Far Detector) groups.

This TFG is developed inside of the FD4 group as our group in the I3M at the UPV belongs to this FD group. FD4 group tasks focus on developing electronics front-end inside the vessel for the inner sensors of the neutrino detector so as to be able to gather information from the sensors, transform it in order to allow the processing unit to process the data properly.

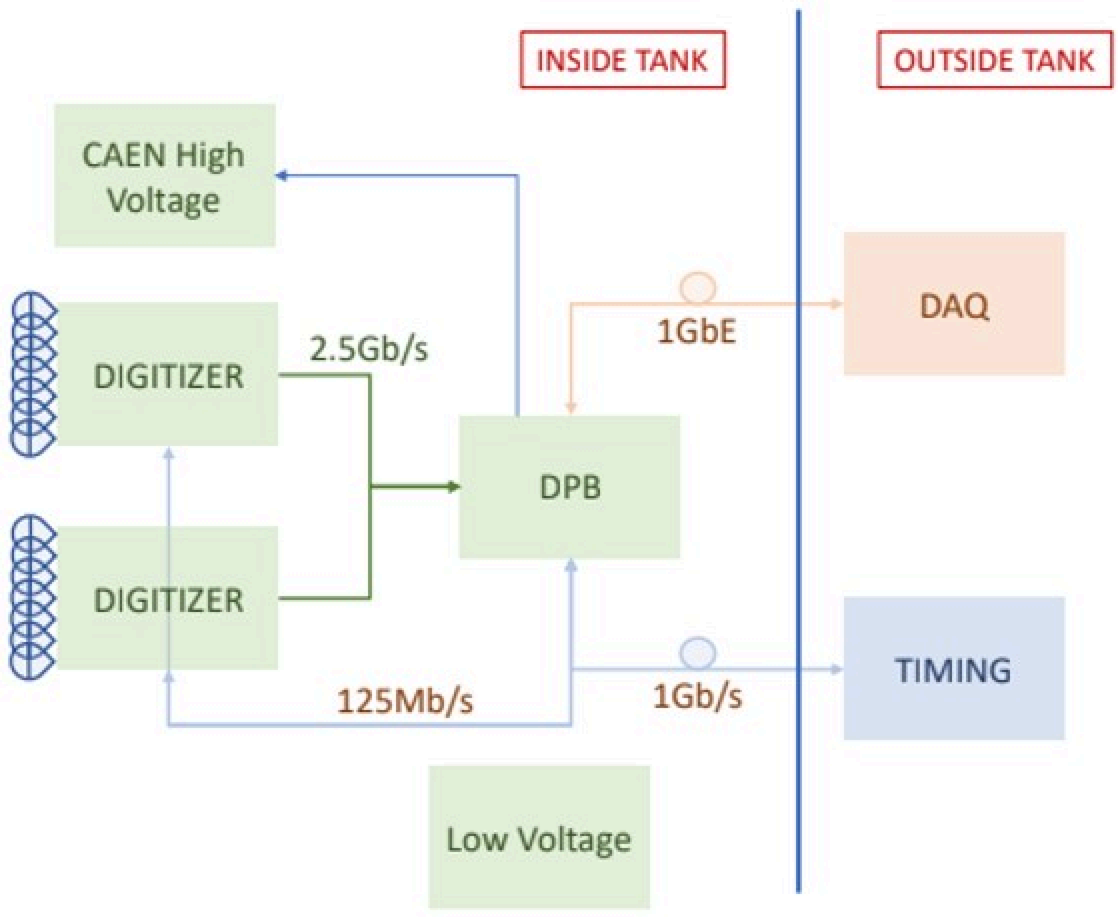


Diagram of communication between the different modules of the vessel

.

As it can be seen in the previous figure, the electronics inside the vessel consits of high and low voltage modules used as power supplies for the rest of the components, digitizers and the DPB, and communicate outside the vessel with the DAQ.

The UPV is in charge of the development of the DPB focusing on collecting and transmitting information to the DAQ and on employing redundancy to maximise reliability.
This is because the DBP, being the hub of the front-end electronics inside the vessel, is responsible for communicating all the modules and as the vessel is submerged in water, the electronics must be reliable enough to last for more than 10 years without needing to be repaired or replaced.

Leveraged technology

Due to the amount of data we intend to work with and the need to customise our board for our application, we have chosen to use a SoM with the AMD Zynq UltraScale+ MPSoC as the processing system, which is combined in the SoM together with logic devices which form a user-programmable logic. The MPSoC includes various controllers such as UART, I2C or eMMC interfaces which provide communication with the peripherals and integrates a monitoring system for the chip itself and its subsystems. In addition, the Zynq UltraScale+ has support for lightweight operating systems, which can be a great benefit if you take advantage of the functionality of the operating system's own drivers.

The SoM will be integrated on a board designed exclusively for our project with the necessary peripherals. With this SoM implementation, we will enjoy great flexibility and customisation in our design without sacrificing the processing power of a high-performance chip such as the one offered by AMD.

System on Module (SoM) and Data Processing Board (DPB) election

In the realm of embedded systems development, System-on-Module (SoM) technology has emerged as a transformative solution, particularly for academic institutions like universities engaged in research and development projects. SoM refers to a compact, integrated circuit board that encapsulates essential components such as processors, memory, and I/O interfaces within a single package.

Understanding System-on-Module (SoM)

System-on-Module (SoM) is a comprehensive computing platform condensed into a small, modular package. These modules typically include a microprocessor or System-on-Chip (SoC), memory components (both RAM and ROM), storage options, power management circuitry, and various peripheral interfaces. SoM modules are standardized in form factors such as COM Express, SMARC, and Qseven, facilitating easy integration into diverse hardware configurations.

Advantages of SoM Technology:

1. **Cost Efficiency:**
    - **Reduced Development Costs:** One of the primary advantages of SoM technology for universities lies in its ability to lower development costs. Instead of investing resources in designing custom PCBs and integrating individual components, universities can procure pre-built SoM modules. While the upfront cost of SoM modules may seem higher compared to standalone chips, the overall development expenditure, including labour and prototyping, is significantly reduced.
    - **Lower Total Cost of Ownership (TCO):** Despite initial investment differences, SoM technology often leads to a lower Total Cost of Ownership (TCO) over the project lifecycle. This is attributed to reduced development time, minimized risk of errors during hardware integration, and streamlined maintenance processes.
2. **Time Efficiency:**
    - **Accelerated Development Cycles:** SoM modules expedite the development process by eliminating the need for designing intricate hardware configurations from scratch. This acceleration is particularly beneficial for universities engaged in time-sensitive research projects or academic initiatives with strict deadlines.
    - **Rapid Prototyping:** SoM technology facilitates rapid prototyping, allowing researchers and students to quickly iterate through design concepts and experiment with various configurations. This agility fosters innovation and enables timely validation of hypotheses.
3. **Risk Mitigation:**
    - **Enhanced Reliability:** SoM modules undergo rigorous testing and validation procedures during manufacturing, ensuring high levels of reliability and performance. By leveraging pre-tested and validated modules, universities mitigate the risk of hardware failures and associated costs, safeguarding project budgets and timelines.
    - **Quality Assurance:** SoM vendors adhere to industry standards and quality control measures, providing universities with assurance regarding the integrity and functionality of the modules. This reliability is crucial for academic endeavors where consistency and reproducibility are paramount.

4. **Resource Optimization:**
    - **Focused Resource Allocation:** By adopting SoM technology, universities can reallocate resources previously dedicated to hardware design towards other aspects of research and development, such as software development, data analysis, and experimentation. This focused resource allocation enhances overall project efficiency and productivity.
    - **Skills Utilization:** SoM technology reduces the dependency on specialized hardware design expertise within university research teams. Instead, academic resources can be channeled towards leveraging domain-specific knowledge and interdisciplinary collaboration, fostering a conducive environment for innovation and knowledge exchange.

Therefore, as part of a university research institute, it has been decided to commission a specialised company, Enclustra, to manufacture the customised SoM. This avoids possible problems with the manufacturing and welding of the SoM, as this will be taken care of by Enclustra.

Preparation of the environment to be used on the board

Starting with the environment to work on the DPB (Data Processing Board) or DPM (Data Processing Module), we will use PetaLinux, a Xilinx software development tool based on a light version of Linux.

The universal availability of the Linux source code and the infinite number of drivers available in Linux gives us greater flexibility and ease of working at the application level on the DPB.

To implement this operating system (OS) on the DPB we have used the Xilinx software, Vivado, and through the JTAG port we have loaded on a 16 GB eMMC as non-volatile memory, both the relevant boot files and the custom image of the PetaLinux project, *image.ub*. As boot files we find *BOOT.BIN* which is the First Stage Boot Loader and *boot.scr*, which is a script on how to boot up. Then we have selected the eMMC as the main boot option by by means of switches from the board itself. In the boot process, the OS is loaded onto the RAM and the RAM is worked on.

Once the OS has been implemented, the connection with the DPB has to be configured. Despite the possibility of maintaining the connection via JTAG, the main source of communication of the DPB is going to be via Ethernet,through SSH protocol, so one of the SFP ports of the DPB has been used to make an Ethernet connection with the equipment by means of an SFP transceiver. For this purpose, the configuration of a 125 MHz PLL for the corresponding Ethernet clock signal was included in the customization of the PetaLinux version.

It should be noted that the main communication with the DPB will be via Ethernet, so the JTAG port after the initial loading of the boot files will only be considered for debugging actions.

Once the connection has been configured, a local DHCP server has been set up to assign a fixed IP to the DPB and facilitate the connection via SSH to the board. For this purpose, the subnet has been declared with a very basic configuration on the server:

```
subnet 20.0.0.0 netmask 255.255.255.0 {
  range 20.0.0.2 20.0.0.30;
  option routers 20.0.0.1;
}
```

The network interface in question has been assigned the address 20.0.0.1 and the subnet has been declared with a small arbitrary range, and the DBP has been assigned the fixed IP 20.0.0.33, an address outside the range, since otherwise the server would return an error. It should be noted that the SFP ports of the DBP are designed to use fibre optic ports, so sometimes the equipment is not able to detect the connection on the Ethernet port using Ethernet cable with RJ-45, so the interface has to be deactivated and then re-activated and assigned the address 20.0.0.1 and the problem is solved, in the case of using a fibre optic port, this problem does not arise.

With the fixed IP address already assigned, it is now possible to access the board via SSH and communicate with it using the following command:

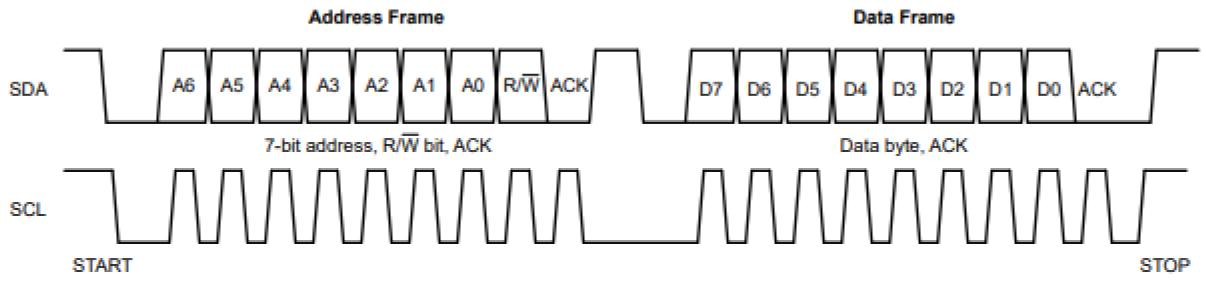
```
ssh root@20.0.0.33
#Here we would enter the relevant password
```

To finish with the establishment of the working environment, we only have to create the application project that is going to be developed on a customised platform of our project, in the Vitis IDE software of Xilinx. The application project has been named DBP2_App.

I^{2}C protocol and how it is implemented on our board

To achieve communication between the different components on the board and the terminal, the I^{2}C protocol is used, a communication protocol based on a Master-Slave system where the communication bus is divided into 2 lines, SCL for the clock and SDA for the data, which are connected to a pull-up resistor each, so the default level is high level.

The operation of this protocol consists of the start of the transmission by the Master which jointly indicates the address of the slave to which it is directed with an address of 7 bits (we have sensors that have an address of 6 bits plus a reserved bit, which can be configurated to differentiate each slave in a physical way), in addition, it is indicated with a bit if the operation to be carried out is reading or writing. The data transmission is guided by the clock line and the data is transmitted in byte size, transmitting from MSB to LSB.



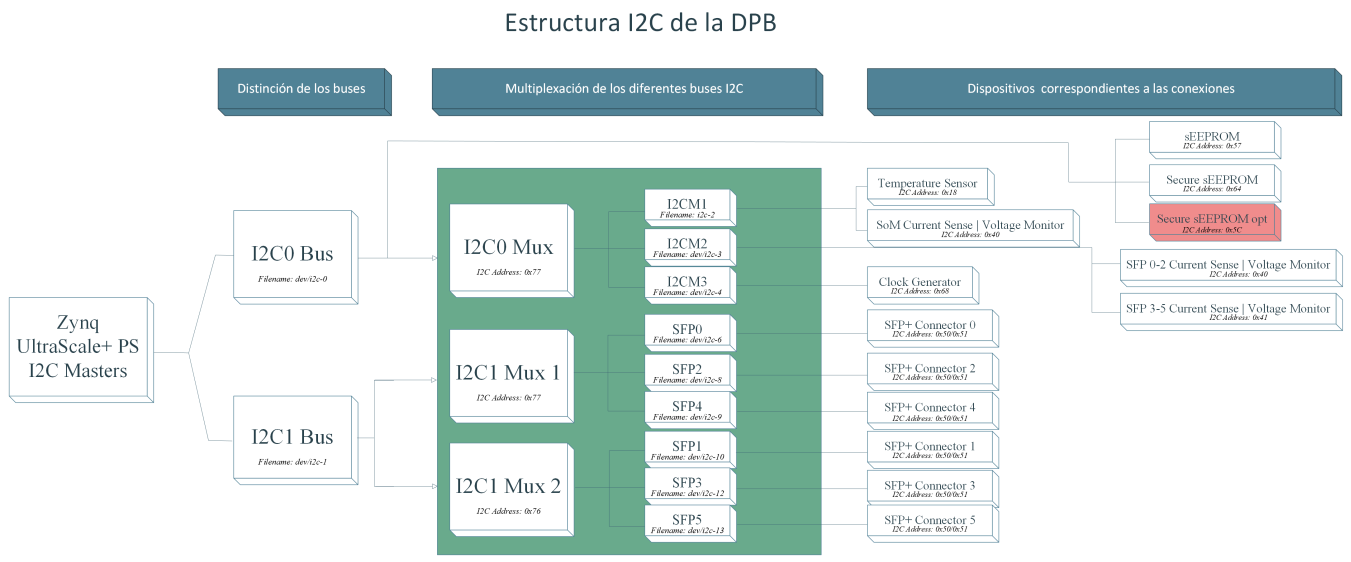
Addressing and data frames I^{2}C

.

For the write operation on the slave, once communication has been established, the register to be written to and the data to be written must be indicated. The master is responsible for receiving the corresponding ACK and NACK during communication and the end of communication sequence.

The read operation follows a similar process to the write operation, indicating the register to be read and the master is in charge of sending the corresponding ACKs and NACKs during the communication and the end of communication sequence.

In our case the communication process will be based on the functions provided by the Linux libraries that allow us to open/close the communication and read/write registers simply by calling defined functions and indicating the necessary arguments. In addition, these functions allow us to operate with vectors in order to read or write consecutive data with a single function.



Structure of the I^{2}C of our DPB

.

In the previous block diagram you can see how the I^{2}C buses of our DPB are structured, the corresponding filename of each of the I^{2}C bus outputs designated by the multiplexers and the slave addresses of each module with which we intend to communicate.

As can be seen, the current sensors, the SFP connectors and the temperature sensor that we intend to use all use the I^{2}C protocol to communicate. However, the temperature sensor and the current sensors use 16-bit registers, while the SFPs use 1-byte sized registers. The I^{2}C protocol carries byte-sized frames, so in the case of 16-bit registers it involves performing 2 consecutive operations (either read or write) on the same register address, whereas for 8-bit registers it will involve a single operation per register address. In our case, the Linux I^{2}C driver will make it much easier to perform this type of consecutive operations.

Detailed information about the available sensors and their capabilities for slow control tasks.

Regarding the sensor units available in our DPB we find, as previously mentioned in the I^{2}C section, a temperature sensor (MCP9844), several current and voltage sensors (INA3221) for the SFP transceivers and the SoM and the SFP transceivers themselves, which provide us with very relevant information about their operating status and that we should keep track of.

Current sensors INA3221

The current sensors installed in the DPB provide us with the possibility of monitoring up to 3 different channels from the same sensor. In addition, it allows us to measure the bus voltage with respect to GND(*Bus Voltage*) or the voltage difference between the IN+ and IN- terminals of each channel(*Shunt Voltage*). In our case, a resistive element with a value of 0.05 $Omega$ has been placed between IN+ and IN-, which is useful for obtaining both the current and the power consumed in each channel.

It should be noted that this sensor allows us to configure alerts and warnings for voltage values obtained in *Shunt Voltage* measurement mode to detect if the voltage difference between terminals of our resistor exceeds or does not reach certain values and to be able to act accordingly. We also have an alert if in *Bus Voltage* measurement mode, which informs us if all the channels being measured have a voltage higher than that marked by the limits or if any of the channels has a voltage lower than the lower limit. We are also provided with the option to obtain the sum of the *Shunt Voltage* of all channels and set a limit to configure an alert. All named alerts and warnings are collected in the *Mask/Enable* register where the sum of the *Shunt Voltage*, warnings and alarms can also be enabled or disabled.

In the following table you can see the most influential registers for our application, a short description of these registers, their default value and the type of register it is, whether it is read-only or read-write.

POINTER ADDRESS (Hex)	**REGISTER NAME**	**DESCRIPTION**	**BINARY (Power-On Reset)**	**HEX (Power-On Reset)**	**TYPE**
0	Configuration	All-register reset, shunt and bus voltage ADC conversion times and operating mode.	01110001 00100111	7127	R/W

POINTER ADDRESS (Hex)	REGISTER NAME	DESCRIPTION	BINARY (Power-On Reset)	HEX (Power-On Reset)	TYPE
1	Channel-1 Shunt Voltage	Averaged shunt voltage value.	00000000 00000000	0000	R
2	Channel-1 Bus Voltage	Averaged bus voltage value.	00000000 00000000	0000	R
3	Channel-2 Shunt Voltage	Averaged shunt voltage value.	00000000 00000000	0000	R
4	Channel-2 Bus Voltage	Averaged bus voltage value.	00000000 00000000	0000	R
5	Channel-3 Shunt Voltage	Averaged shunt voltage value.	00000000 00000000	0000	R
6	Channel-3 Bus Voltage	Averaged bus voltage value.	00000000 00000000	0000	R
7	Channel-1 Critical Alert	Contains limit value to compare each conversion value to determine if the corresponding limit has been exceeded.	01111111 11111000	7FF8	R/W
8	Channel-1 Warning Alert	Contains limit value to compare to averaged measurement to determine if the corresponding limit has been exceeded.	01111111 11111000	7FF8	R/W

POINTER ADDRESS (Hex)	REGISTER NAME	DESCRIPTION	BINARY (Power-On Reset)	HEX (Power-On Reset)	TYPE
9	Channel-2 Critical Alert	Contains limit value to compare each conversion value to determine if the corresponding limit has been exceeded.	01111111 11111000	7FF8	R/W
A	Channel-2 Warning Alert	Contains limit value to compare to averaged measurement to determine if the corresponding limit has been exceeded.	01111111 11111000	7FF8	R/W
B	Channel-3 Critical Alert	Contains limit value to compare each conversion value to determine if the corresponding limit has been exceeded.	01111111 11111000	7FF8	R/W
C	Channel-3 Warning Alert	Contains limit value to compare to averaged measurement to determine if the corresponding limit has been exceeded.	01111111 11111000	7FF8	R/W
D	Shunt-Voltage Sum	Contains the summed value of the each of the selected shunt voltage conversions.	00000000 00000000	0000	R

POINTER ADDRESS (Hex)	REGISTER NAME	DESCRIPTION	BINARY (Power-On Reset)	HEX (Power-On Reset)	TYPE
E	Shunt-Voltage Sum Limit	Contains limit value to compare to the Shunt Voltage Sum register to determine if the corresponding limit has been exceeded.	01111111 11111110	7FFE	R/W
F	Mask/Enable	Alert configuration, alert status indication, summation control and status.	00000000 00000010	0002	R/W
10	Power-Valid Upper Limit	Contains limit value to compare all bus voltage conversions to determine if the Power Valid level has been reached.	00100111 00010000	2710	R/W
11	Power-Valid Lower Limit	Contains limit value to compare all bus voltage conversions to determine if the any voltage rail has dropped below the Power Valid range.	00100011 00101000	2328	R/W
FE	Manufacturer ID	Contains unique manufacturer identification number.	01010100 01001001	5449	R
FF	Die ID	Contains unique die identification number.	00110010 00100000	3220	R

INA3221 Current Sensor Registers

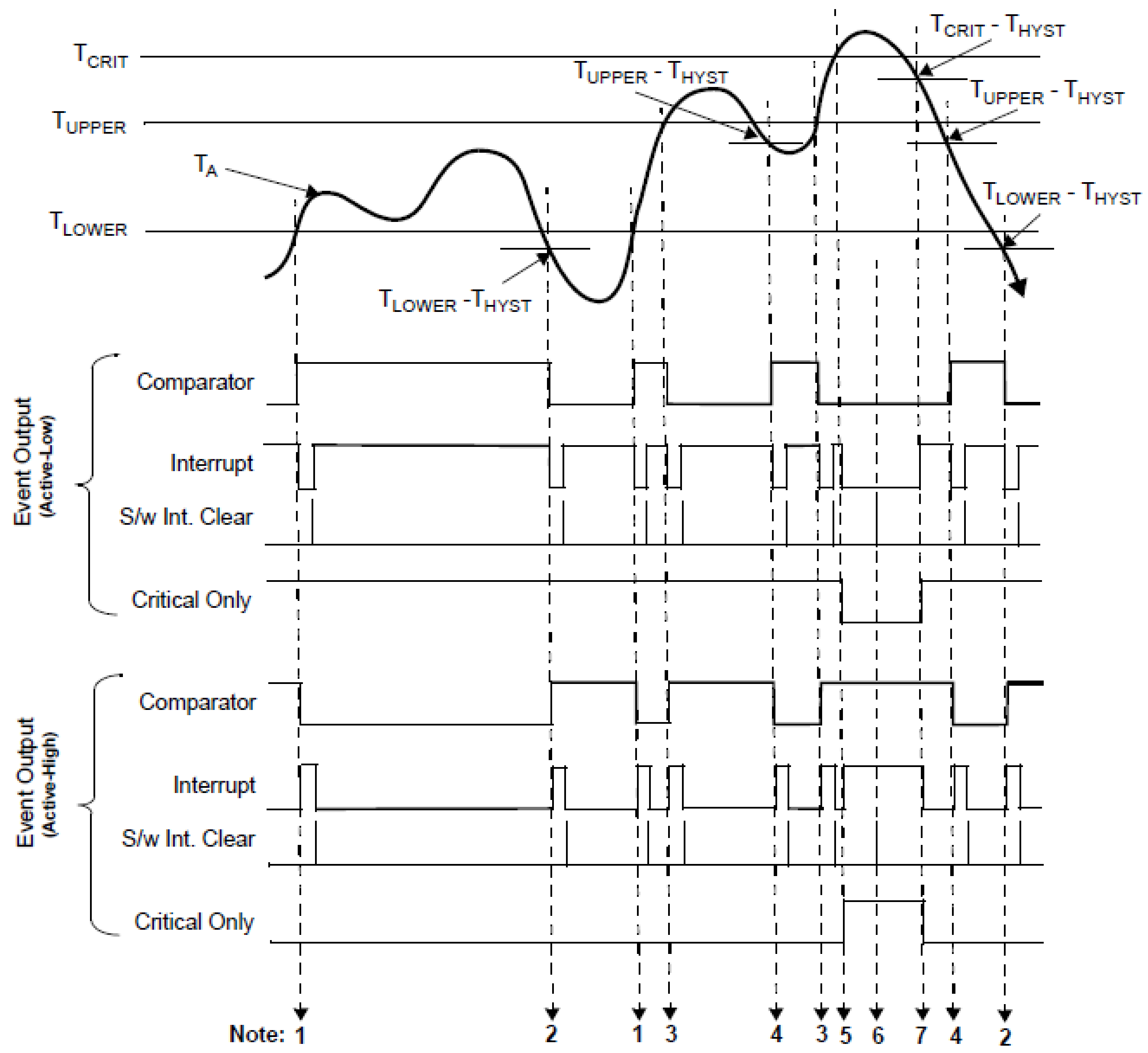
It is worth mentioning that all voltage data is given in 2's complement and uses 13 bits, bit 15 of the register (MSB) determines the sign and bit 14-3 the voltage data.
For *Shunt Voltage* the full scale range is 163.8 mV and the LSB is 40 µV, in the case of *Bus Voltage* the LSB is 8 mV and although the full scale range of the ADC is 32.76 V, the full scale range in the case of *Bus Voltage* is 26 V since it is not recommended to apply more voltage.

Temperature sensor MCP9844

The temperature sensor MCP9844 is a great IC to monitor the temperature of the environment where our DPB works, an essential magnitude to ensure operating conditions within the working range of our electronics.

This temperature sensor provides us with the events tool that facilitates the monitoring of the ambient temperature. The MCP9844 allows us to set temperature limits, only modifiable if enabled in the configuration register, both upper and lower and even critical temperature (only higher than the upper limit). Once the limits have been established, from the configuration register you can enable or disable the events and you can configure the event as an interruption or as a comparison, decide whether the event is active at high or low level and decide whether only the critical temperature limit is taken into account or all the limits are taken into account.

In addition, the sensor has several functionalities such as the option to include a certain hysteresis value to the temperature limits (only applicable in case of temperature drop), the possibility to modify the measurement resolution (lower resolution value will imply a longer conversion time) or the possibility to switch off the sensor if desired.



Operation of the alarms MCP9844 Temperature Sensor

Below is a table of the registers presented by this temperature sensor and its default value.

Register Address (Hexadecimal)	Register Name	Default Register Data (Hexadecimal)	Power-Up Default Register Description
0x00	Capability	0x00EF	Event output deasserts in shutdown I^2C™ time out 25 ms to 35 ms. Accepts VHV at A0 Pin 0.25°C

Register Address (Hexadecimal)	Register Name	Default Register Data (Hexadecimal)	Power-Up Default Register Description
			Resolution . Measures temperature below 0°C ±1°C accuracy over active range Temperature event output
0x01	CONFIG	0x0000	Comparator mode Active-Low output Event and critical output Output disabled Event not asserted Interrupt cleared Event limits unlocked Critical limit unlocked Continuous conversion 0°C Hysteresis
0x02	T_{UPPER}	0x0000	0°C
0x03	T_{LOWER}	0x0000	0°C
0x04	T_{CRIT}	0x0000	0°C
0x05	T_A	0x0000	0°C
0x06	Manufacturer ID	0x0054	—
0x07	Microchip Device ID/ Device Revision	0x0601	—
0x09	Resolution	0x8001	Most Significant bit is set by default 0.25°C Measurement Resolution

MCP9844 Temperature Sensor Registers

In this case the temperature data is encoded in 2's complement and is presented as a 13-bit data, 1 bit determining the sign and 12 determining the temperature data, so the manufacturer provides us with the following equations to obtain the data in degrees Celsius.

If Temperature $\geq 0°$C

$$T_A(ºC) = (UpperByte * 2^4 + LowerByte * 2^{-4})$$

(2.2.1)

If Temperature < 0 °C

$$T_A(ºC) = (UpperByte * 2^4 + LowerByte * 2^{-4}) - 256$$

(2.2.2)

Where *UpperByte* are bits 15-8 of the T_A register and *LowerByte* are bits 7-0 of the same register.

In the case of the temperature limits these are defined by 11 bits, 1 bit determining the sign and 10 bits to encode the absolute temperature data.

SFP Transceiver AFBR-5715ALZ

Los transceptores SFP de fibra óptica tienen la función principal ser los puertos de comunicación de la placa. These transceivers have an EEPROM memory that is divided into two pages, which correspond to the slave addresses I^2C 0x50 and 0x51 in our case.

The SFPs collect information on highly relevant real-time quantities and are located on the second page of the EEPROM (0x51), such as the temperature within the module, the supply voltage supplied to them, the laser bias current and both the transmitted and received optical power.

On the same second page of the SFP EEPROM is the possibility to use alerts and warnings based on a range already determined by the manufacturer to monitor the status of the SFP transceivers.

Although the first page of the EEPROM is mainly based on transceiver identification characters such as part number and revision or vendor name, we can also find relevant information about the status and operation of the transceiver as we can find in this memory space the wavelength of the laser to know in which window it is working and the register that tells us if the status signals

TX_DISABLE, TX_FAULT and RX_LOS have been configured by hardware.

In both pages of the EEPROM we find one or more registers dedicated to a Checksum that will allow us to check the status of the EEPROM itself.

Below are several tables representing the EEPROM registers of the SFP transceivers.

Byte Decimal	Data Notes
0	SFP physical device
1	SFP function defined by serial ID only
2	LC optical connector
6	1000BaseSX
11	Compatible with 8B/10B encoded data
12	1200Mbps nominal bit rate (1.25Gbps)
16	550m of 50/125mm fiber @ 1.25Gbps
17	275m of 62.5/125mm fiber @ 1.25Gbps
20-35	'AVAGO' - Vendor Name ASCII character
37	Vendor OUI
38	Vendor OUI
39	Vendor OUI
40-55	'AFBR-5715ALZ' - Vendor Part Number ASCII characters
56-59	Vendor Revision Number ASCII character
60	Hex Byte of Laser Wavelength
61	Hex Byte of Laser Wavelength
63	Checksum for bytes 0-62
65	Hardware SFP TX_DISABLE, TX_FAULT, & RX_LOS

Byte Decimal	Data Notes
68-83	Vendor Serial Number, ASCII
84-91	Vendor Date Code, ASCII
95	Checksum for bytes 64-94

SFP transceiver EEPROM page 1 registers

Byte Decimal	Notes	Byte Decimal	Notes	Byte Decimal	Notes
0	Temp H Alarm MSB	26	Tx Pwr L Alarm MSB	104	Real Time Rx P_{AV} MSB
1	Temp H Alarm LSB	27	Tx Pwr L Alarm LSB	105	Real Time Rx P_{AV} LSB
2	Temp L Alarm MSB	28	Tx Pwr H Warning MSB	106	
3	Temp L Alarm LSB	29	Tx Pwr H Warning LSB	107	
4	Temp H Warning MSB	30	Tx Pwr L Warning MSB	108	
5	Temp H Warning LSB	31	Tx Pwr L Warning LSB	109	
6	Temp L Warning MSB	32	Rx Pwr H Alarm MSB	110	Status/ Control
7	Temp L Warning LSB	33	Rx Pwr H Alarm LSB	111	
8	VCC H Alarm MSB	34	Rx Pwr L Alarm MSB	112	Flag Bits

Byte Decimal	Notes	Byte Decimal	Notes	Byte Decimal	Notes
9	VCC H Alarm LSB	35	Rx Pwr L Alarm LSB	113	Flag Bit
10	VCC L Alarm MSB	36	Rx Pwr H Warning MSB	114	
11	VCC L Alarm LSB	37	Rx Pwr H Warning LSB	115	
12	VCC H Warning MSB	38	Rx Pwr L Warning MSB	116	Flag Bits
13	VCC H Warning LSB	39	Rx Pwr L Warning LSB	117	Flag Bits
16	Tx Bias H Alarm MSB	95	Checksum for Bytes 0-94	120	
17	Tx Bias H Alarm LSB	96	Real Time Temperature MSB	121	
18	Tx Bias L Alarm MSB	97	Real Time Temperature LSB	122	
19	Tx Bias L Alarm LSB	98	Real Time Vcc MSB	123	
20	Tx Bias H Warning MSB	99	Real Time Vcc LSB	124	
21	Tx Bias H Warning LSB	100	Real Time Tx Bias MSB	125	
22	Tx Bias L Warning MSB	101	Real Time Tx Bias LSB	126	

Byte Decimal	Notes	Byte Decimal	Notes	Byte Decimal	Notes
23	Tx Bias L Warning LSB	102	Real Time Tx Power MSB	127	
24	Tx Pwr H Alarm MSB	103	Real Time Tx Power LSB	128	
25	Tx Pwr H Alarm LSB				

SFP transceiver EEPROM page 2 registers

To interpret the data of the magnitudes on page 2 in bit format, the following clarifications from the manufacturer must be taken into account depending on the magnitude to be interpreted:

- **Temperature (Temp):** Temperature values are encoded as 16-bit integers in two's complement, which allows both positive and negative values to be represented. Each unit in this representation is equivalent to 1/256 of a degree Celsius (ºC).
- **Power Supply Voltage (VCC):** This parameter is represented as a 16-bit unsigned integer, which means that it can only have positive values. Each increment in this value corresponds to 100 microvolts (µV).
- **Laser Bias Current (Tx Bias):** The laser bias current is decoded as a 16-bit unsigned integer, which means that it can only be positive. Each increment in this value represents 2 microamperes (µA).
- **Average Transmitted Optical Power (Tx Pwr):** This parameter is represented as a 16-bit unsigned integer, where each increment corresponds to 0.1 microwatt (µW) of transmitted optical power.
- **Average Optical Power Received (Rx Pwr):** Similar to the previous parameter, the average optical power received is encoded as a 16-bit unsigned integer. Each unit of this value represents 0.1 microwatt (µW) of received optical power.

As can be seen in the register table on the second page of the EEPROM, there is a status register and this describes the following cases.

Bit #	Status/Control Name	Description
7	Tx Disable State	Digital state of SFP Tx Disable Input Pin (1 = Tx_Disable asserted)
6	Soft Tx Disable	Read/write bit for changing digital state of SFP Tx_Disable function
4	Rx Rate Select State	Digital state of SFP Rate Select Input Pin (1 = full bandwidth of 155 Mbit)
2	Tx Fault State	Digital state of the SFP Tx Fault Output Pin (1 = Tx Fault asserted)
1	Rx LOS State	Digital state of the SFP LOS Output Pin (1 = LOS asserted)
0	Data Ready (Bar)	Indicates transceiver is powered and real-time sense data is ready (0 = Ready)

Breakdown of SFP transceiver status bits

As for the registers dedicated to the *flags*, these contain the indicator bits of the previously mentioned alerts and warnings. The following table shows their distribution in the relevant registers.

Byte	Bit #	Flag Bit Name	Description
112	7	Temp High Alarm	Set when transceiver internal temperature exceeds high alarm threshold.
	6	Temp Low Alarm	Set when transceiver internal temperature exceeds low alarm threshold.
	5	VCC High Alarm	Set when transceiver internal supply voltage exceeds high alarm threshold.
	4	VCC Low Alarm	Set when transceiver internal supply voltage exceeds

Byte	Bit #	Flag Bit Name	Description
			low alarm threshold.
	3	Tx Bias High Alarm	Set when transceiver laser bias current exceeds high alarm threshold.
	2	Tx Bias Low Alarm	Set when transceiver laser bias current exceeds low alarm threshold.
	1	Tx Power High Alarm	Set when transmitted average optical power exceeds high alarm threshold.
	0	Tx Power Low Alarm	Set when transmitted average optical power exceeds low alarm threshold.
------	-------	---------------	-------------
113	7	Rx Power High Alarm	Set when received P_Avg optical power exceeds high alarm threshold.
	6	Rx Power Low Alarm	Set when received P_Avg optical power exceeds low alarm threshold.
------	-------	---------------	-------------
116	7	Temp High Warning	Set when transceiver internal temperature exceeds high warning threshold.
	6	Temp Low Warning	Set when transceiver internal temperature exceeds low warning threshold.
	5	VCC High Warning	Set when transceiver internal supply voltage exceeds high warning threshold.
	4	VCC Low Warning	Set when transceiver internal supply voltage exceeds low warning threshold.
	3	Tx Bias High Warning	Set when transceiver laser bias current exceeds high warning threshold.
	2	Tx Bias Low	Set when transceiver laser bias current exceeds low

Byte	Bit #	Flag Bit Name	Description
		Warning	warning threshold.
	1	Tx Power High Warning	Set when transmitted average optical power exceeds high warning threshold.
	0	Tx Power Low Warning	Set when transmitted average optical power exceeds low warning threshold.
------	-------	---------------	-------------
117	7	Rx Power High Warning	Set when received P_Avg optical power exceeds high warning threshold.
	9	Rx Power Low Warning	Set when received P_Avg optical power exceeds low warning threshold.

Breakdown of the *flags* of SFP transceivers

Data gathering from AMS, PS and PL SYSMON and channel differentiation

Due to the sensors together with ADC converters with which Xilinx has equipped our module and its system monitoring hardware block (SYSMON), we can access a large amount of real-time information from the PS and the PL via the Linux driver "xilinx-ams". This Linux driver exports real-time data into files using *sysfs*, a pseudo file system provided by the Linux kernel which exports information into virtual files.

This information collected from the PS and PL is differentiated into different channels which are explained in the following table.

Sysmon Block	Channel	Details	Measurement	File
PS	7	LPD temperature measurement.	Temperature	*in_temp*

Sysmon Block	Channel	Details	Measurement	File
Sysmon				*in_temp* *in_temp* *in_temp*
	8	FPD temperature measurement (REMOTE).	Temperature	*in_temp* *in_temp* *in_temp* *in_temp*
	9	VCC PS LPD voltage measurement (supply1).	Voltage	*in_volta* *in_volta*
	10	VCC PS FPD voltage measurement (supply2).	Voltage	*in_volta* *in_volta*
	11	PS Aux voltage reference (supply3).	Voltage	*in_volta* *in_volta*
	12	DDR I/O VCC voltage measurement.	Voltage	*in_volta* *in_volta*
	13	PS IO Bank 503 voltage measurement (supply5).	Voltage	*in_volta* *in_volta*
	14	PS IO Bank 500 voltage measurement (supply6).	Voltage	*in_volta* *in_volta*
	15	VCCO_PSIO1 voltage measurement.	Voltage	*in_volta* *in_volta*
	16	VCCO_PSIO2 voltage measurement.	Voltage	*in_volta* *in_volta*
	17	VCC_PS_GTR voltage measurement (VPS_MGTRAVCC).	Voltage	*in_volta* *in_volta*
	18	VTT_PS_GTR voltage measurement	Voltage	*in_volta*

Sysmon Block	Channel	Details	Measurement	File
		(VPS_MGTRAVTT).		*in_volta*
	19	VCC_PSADC voltage measurement.	Voltage	*in_volta* *in_volta*
--------------	---------	--	------------	----------
PL Sysmon	20	PL temperature measurement.	Temperature	*in_temp* *in_temp* *in_temp* *in_temp*
	21	PL Internal voltage measurement, VCCINT.	Voltage	*in_volta* *in_volta*
	22	PL Auxiliary voltage measurement, VCCAUX.	Voltage	*in_volta* *in_volta*
	23	ADC Reference P+ voltage measurement.	Voltage	*in_volta* *in_volta*
	24	ADC Reference N- voltage measurement.	Voltage	*in_volta* *in_volta*
	25	PL Block RAM voltage measurement, VCCBRAM.	Voltage	*in_volta* *in_volta*
	26	LPD Internal voltage measurement, VCC_PSINTLP (supply4).	Voltage	*in_volta* *in_volta*
	27	FPD Internal voltage measurement, VCC_PSINTFP (supply5).	Voltage	*in_volta* *in_volta*
	28	PS Auxiliary voltage measurement (supply6).	Voltage	*in_volta* *in_volta*
	29	PL VCCADC voltage measurement (vccams).	Voltage	*in_volta* *in_volta*

The chart starts from channel 7 as the previous channels are from the AMS CTRL SYSMON block and display repeated information from the PL.

The information obtained is displayed in ADC code in the *_raw* file and has to be scaled with the value obtained in the *_scale* file. In the case of temperature, an offset from the *_offset* file must also be applied. Every file is a virtual file generated by the *sysfs* file system.

The expressions used to convert the values read to the corresponding magnitude are shown below.

$$V_{XX}(V) = (in\_voltageXX\_raw * in\_voltageXX\_scale) * \frac{1}{2^{n\_bits}}$$

(3.1)

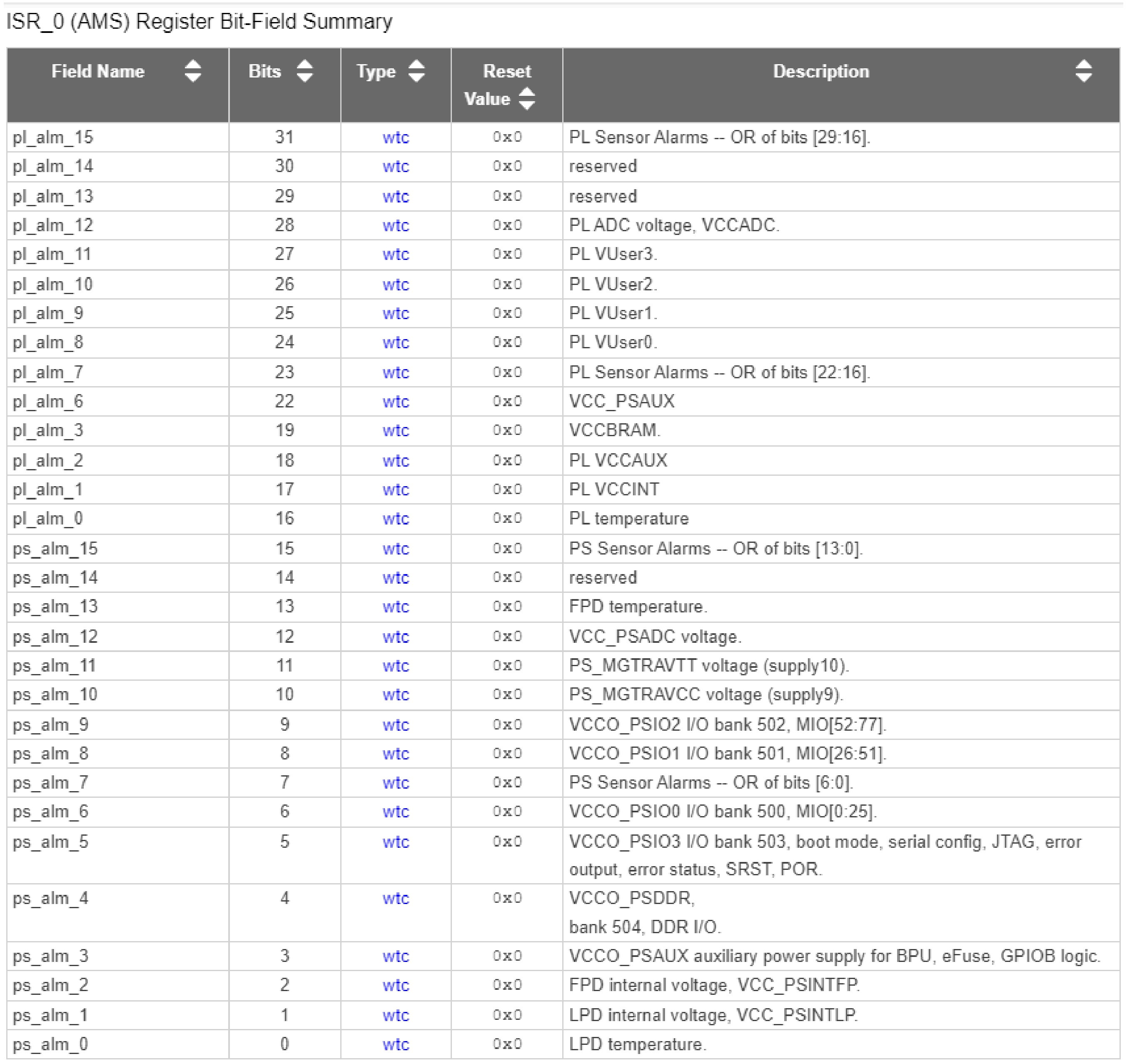
$$T_{XX}(^{\text{º}}C) = (in\_tempXX\_raw + in\_tempXX\_offset) * \frac{in\_tempXX\_scale}{2^{n\_bits}}$$

(3.2)

Where XX defines the selected channel number in voltage or temperature and "n_bits" defines the number of bits of the ADC used, in our case 10 bits. The offset in the case of temperature is added since a negative number is returned.

Xilinx also offers alarms applied to the voltages and temperatures measured on the previously mentioned channels and the Linux driver allows us to configure and read these alarms also using the *IIO_EVENT_MONITOR* tool of Linux itself. *IIO_EVENT_MONITOR* is a generic application from the Linux kernel developed to catch and report different types of events from Industrial Input Output (IIO) devices. In order to detect events or enabling event detection, the *IIO_EVENT_MONITOR* application works with the *sysfs* file system.

In the case of temperature, there are only alarms that are activated if a certain temperature is exceeded, while in the case of voltage, there are alarms for both overvoltage and undervoltage, but without specifying whether the limit exceeded is the lower or upper limit as the alarm is a single bit, so it does not discriminate between falling or rising event (shown as *either*).



ISR_0 (AMS) Register Bit-Field Summary

Field Name	Bits	Type	Reset Value	Description
pl_alm_15	31	wtc	0x0	PL Sensor Alarms -- OR of bits [29:16].
pl_alm_14	30	wtc	0x0	reserved
pl_alm_13	29	wtc	0x0	reserved
pl_alm_12	28	wtc	0x0	PL ADC voltage, VCCADC.
pl_alm_11	27	wtc	0x0	PL VUser3.
pl_alm_10	26	wtc	0x0	PL VUser2.
pl_alm_9	25	wtc	0x0	PL VUser1.
pl_alm_8	24	wtc	0x0	PL VUser0.
pl_alm_7	23	wtc	0x0	PL Sensor Alarms -- OR of bits [22:16].
pl_alm_6	22	wtc	0x0	VCC_PSAUX
pl_alm_3	19	wtc	0x0	VCCBRAM.
pl_alm_2	18	wtc	0x0	PL VCCAUX
pl_alm_1	17	wtc	0x0	PL VCCINT
pl_alm_0	16	wtc	0x0	PL temperature
ps_alm_15	15	wtc	0x0	PS Sensor Alarms -- OR of bits [13:0].
ps_alm_14	14	wtc	0x0	reserved
ps_alm_13	13	wtc	0x0	FPD temperature.
ps_alm_12	12	wtc	0x0	VCC_PSADC voltage.
ps_alm_11	11	wtc	0x0	PS_MGTRAVTT voltage (supply10).
ps_alm_10	10	wtc	0x0	PS_MGTRAVCC voltage (supply9).
ps_alm_9	9	wtc	0x0	VCCO_PSIO2 I/O bank 502, MIO[52:77].
ps_alm_8	8	wtc	0x0	VCCO_PSIO1 I/O bank 501, MIO[26:51].
ps_alm_7	7	wtc	0x0	PS Sensor Alarms -- OR of bits [6:0].
ps_alm_6	6	wtc	0x0	VCCO_PSIO0 I/O bank 500, MIO[0:25].
ps_alm_5	5	wtc	0x0	VCCO_PSIO3 I/O bank 503, boot mode, serial config, JTAG, error output, error status, SRST, POR.
ps_alm_4	4	wtc	0x0	VCCO_PSDDR, bank 504, DDR I/O.
ps_alm_3	3	wtc	0x0	VCCO_PSAUX auxiliary power supply for BPU, eFuse, GPIOB logic.
ps_alm_2	2	wtc	0x0	FPD internal voltage, VCC_PSINTFP.
ps_alm_1	1	wtc	0x0	LPD internal voltage, VCC_PSINTLP.
ps_alm_0	0	wtc	0x0	LPD temperature.

AMS alarms register set

Slow control functions

After an exhaustive study of all the sensing elements, the operation of each one, their corresponding characteristics and alerts and the communication channel with these devices, I can start programming the different functions that will allow us to communicate with these devices and configure them to our needs or obtain the desired information.

Regarding the application to be programmed, it has been decided that it will be an application that will have several threads and sub-processes that will allow us to monitor the status of the DPB periodically, control the different alarms that we have and attend to interruptions. The threads will be defined using POSIX threads execution model implemented as pthreads in C programming language. All the gathered information will then be transmitted to the DAQ in a JSON format.

The DAQ will also send configuration commands and the application will have parse and read the command and act according to these commands.

Device and AMS alarms initialization

In order to start the application, all the I^{2}C devices to be used must be initialized, as well as the *IIO_EVENT_MONITOR* to be able to receive the alarms coming from the AMS.

```c
struct DPB_I2cSensors{

        struct I2cDevice dev_pcb_temp;
        struct I2cDevice dev_sfp0_2_volt;
        struct I2cDevice dev_sfp3_5_volt;
        struct I2cDevice dev_som_volt;
        struct I2cDevice dev_sfp0_A0;
        struct I2cDevice dev_sfp1_A0;
        struct I2cDevice dev_sfp2_A0;
        struct I2cDevice dev_sfp3_A0;
        struct I2cDevice dev_sfp4_A0;
        struct I2cDevice dev_sfp5_A0;
        struct I2cDevice dev_sfp0_A2;
        struct I2cDevice dev_sfp1_A2;
        struct I2cDevice dev_sfp2_A2;
        struct I2cDevice dev_sfp3_A2;
        struct I2cDevice dev_sfp4_A2;
        struct I2cDevice dev_sfp5_A2;
};
```

Firstly, every I^{2}C device has been gathered and globally defined in a struct. Then, the following functions have been developed to initialize each of the I^{2}C devices:

```
int init_tempSensor (struct I2cDevice *dev) {
        int rc = 0;
        uint8_t manID_buf[2] = {0,0};
        uint8_t manID_reg = MCP9844_MANUF_ID_REG;
        uint8_t devID_buf[2] = {0,0};
        uint8_t devID_reg = MCP9844_DEVICE_ID_REG;
        :
        :
        :
```

As it can be seen in the function init_tempSensor(), every I^{2}C device is initialized following a similar strucutre. Firstly, the device is started using the fucntion i2c_start from the I^{2}C Linux driver, then we check if we are initializing the correct device by checking specific registers and comparing them to their expected value and if any step fails, the function returns ans specifies the error.

For SFPs, the initialization process is a bit different since each of the two pages of their EEPROM is initialized as an independent devices. Furthermore, we use the checksum contained in the memory pages to verify the proper status of the memory pages.

We use the following function to verify that the checksum value is correct:

```
int checksum_check(struct I2cDevice *dev,uint8_t ini_reg, uint8_t checksum_val, int size)
        int rc = 0;
        int sum = 0;
        uint8_t byte_buf[size] ;

        rc = i2c_readn_reg(dev,ini_reg,byte_buf,1);  //Read every register from ini_reg to
                        if(rc < 0)
                                return rc;
        for(int n=1;n<size;n++){
        ini_reg ++;
        rc = i2c_readn_reg(dev,ini_reg,&byte_buf[n],1);
                if(rc < 0)
                        return rc;
        }

        for(int i=0;i<size;i++){
                sum += byte_buf[i];  //Sum every register read in order to obtain the che
        }
        uint8_t calc_checksum = (sum & 0xFF); //Only taking the 8 LSB of the checksum as

        if (checksum_val != calc_checksum){ //Check the obtained checksum equals the devi
                printf("Checksum value does not match the expected value \r\n");
                return -EHWPOISON;
        }
        return 0;
}
```

In order to verify the checksums from any SFP, a function was needed that calculates the current checksum value and compares it to the expected value. So as to calculate the checksum, the function needs to be given the I^{2}C device, the address of the first register to be counted in the checksum calculation, the expected checksum value and the amount of registers to be counted in the checksum calculation.

The function sums every register in the given range, and only takes the 8 LSB as the SFP registers size is 1 byte.

```
int init_I2cSensors(struct DPB_I2cSensors *data){

        data->dev_pcb_temp.filename = "/dev/i2c-2";
        data->dev_pcb_temp.addr = 0x18;
    :
    :
    :
```

Finally, in this function we define every filename and slave address for every I^{2}C device and we call every initialization functions mentioned previously and in case any initialization misses, it is reported which device has failed.

```
int stop_I2cSensors(struct DPB_I2cSensors *data){

        i2c_stop(&data->dev_pcb_temp);
    :
    :
    :
```

It has also been developed the stop_I2cSensors function to termiante the I^{2}C devices by using I^{2}C Linux driver function i2c_stop(), even though it is not exepcted to be used as the application should be permantently active.

Regarding the *IIO_EVENT_MONITOR* initialization, it has been executed as a subprocess that will detect AMS alarms as events and it is executed by the following function:

```
int iio_event_monitor_up() {
    pid_t pid = fork(); // Create a child process

    if (pid == 0) {
        // Child process
        // Path of the .elf file and arguments
        char *args[] = {"/run/media/mmcblk0p1/IIO_MONITOR.elf", "-a", "/dev/iio:device0",

        // Execute the .elf file
        if (execvp(args[0], args) == -1) {
            perror("Error executing the .elf file");
            return -1;
        }
    } else if (pid > 0) {
        // Parent process
        // You can perform other tasks here while the child process executes the .elf fil
    } else {
        // Error creating the child process
        perror("Error creating the child process");
        return -1;
    }
    return 0;
}
```

This function executes the *IIO_EVENT_MONITOR* application through its release file. It should be emphasized that this *IIO_EVENT_MONITOR* is slightly customized by us so as to include shared memory configuration to communicate the main application with it.

At first, it was recommended to use the function system() to execute the process as it was a bash script. Nevertheless, it did not resulted as expected so it was decided to use the function execvp(), which also provides us with a more visual and convenient way of passing the necessary arguments to the function.

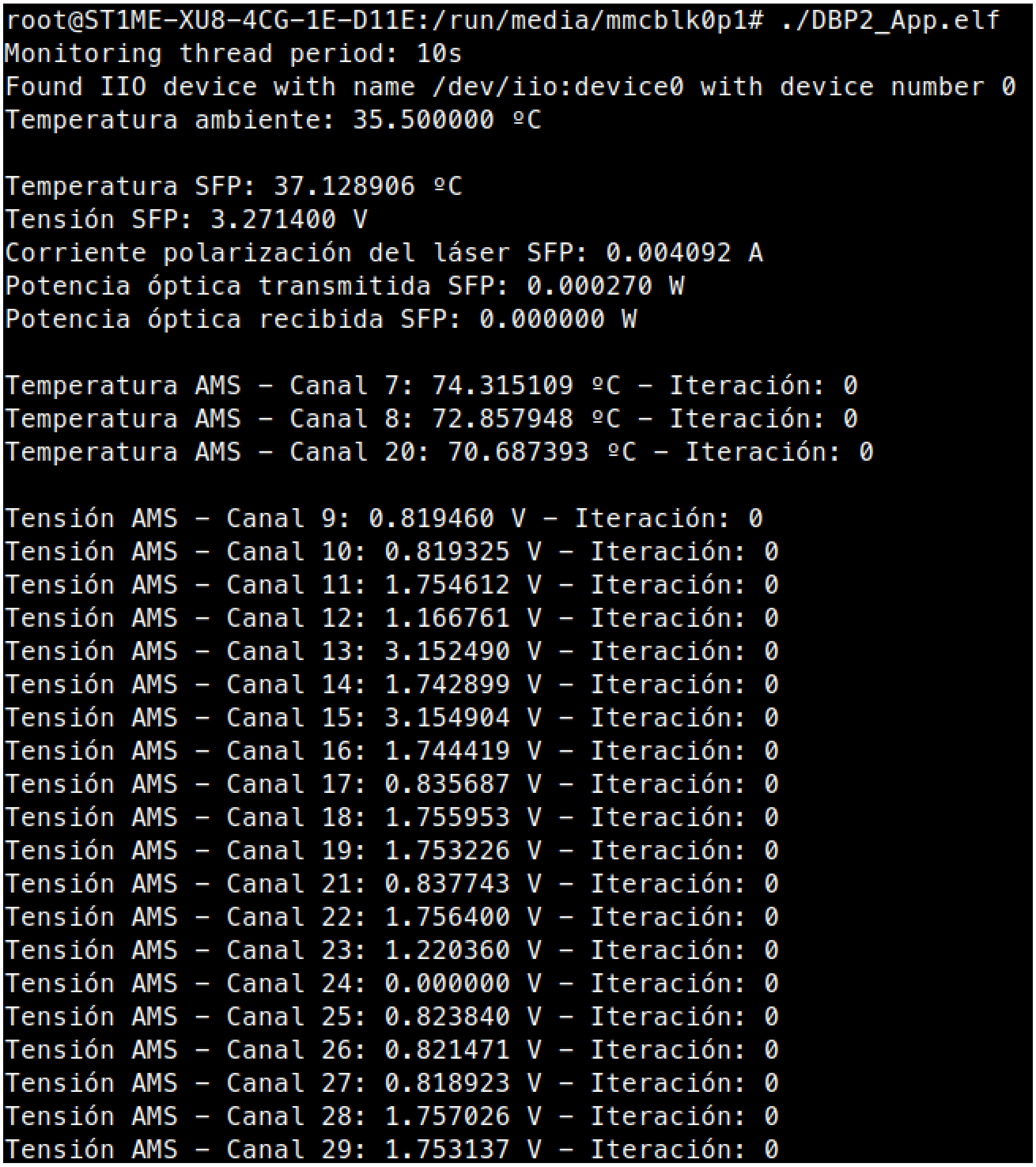
Monitoring thread

The first thread to be developed in tha application has been the monitoring thread, which consists in a periodic function that every iteration reads every magnitude from every device available and after it gathers all the data, it reports it to the DAQ. This will allow us to have real time information on the status of the DPB.

In order to be able to read information from the I^2C sensors by using the I^2C Linux driver, I have used the functions i2c_write() to write the address of the register I want to read in the register pointer, so that by using the function i2c_read() I can read the desired register byte to byte as all the I^2C devices that have been used 2 bytes registers and allow continuous reading apart from the SFPs.

In order to read from the SFPs, I used the function i2c_readn_reg() which is an implicit combination of i2c_write() to write in the register pointer and i2c_read(). As the SFPs do not allow continuous reading and their register size is 1 byte, i2c_readn_reg() has been used two times to read MSB and LSB of the desired data and it has been specified the the appropriate register address for each operation.

Regarding the data provided by the AMS, we obtain them in ADC code by calling a function that will access the *sysfs* files generated by the "xilinx-ams" driver, and the final magnitude is obtained by applying the conversion explained in the chapter "Data gathering from AMS, PS and PL SYSMON and channel differentiation" depending on whether we are dealing with voltage or temperature.



```
root@ST1ME-XU8-4CG-1E-D11E:/run/media/mmcblk0p1# ./DBP2_App.elf
Monitoring thread period: 10s
Found IIO device with name /dev/iio:device0 with device number 0
Temperatura ambiente: 35.500000 ºC

Temperatura SFP: 37.128906 ºC
Tensión SFP: 3.271400 V
Corriente polarización del láser SFP: 0.004092 A
Potencia óptica transmitida SFP: 0.000270 W
Potencia óptica recibida SFP: 0.000000 W

Temperatura AMS - Canal 7: 74.315109 ºC - Iteración: 0
Temperatura AMS - Canal 8: 72.857948 ºC - Iteración: 0
Temperatura AMS - Canal 20: 70.687393 ºC - Iteración: 0

Tensión AMS - Canal 9: 0.819460 V - Iteración: 0
Tensión AMS - Canal 10: 0.819325 V - Iteración: 0
Tensión AMS - Canal 11: 1.754612 V - Iteración: 0
Tensión AMS - Canal 12: 1.166761 V - Iteración: 0
Tensión AMS - Canal 13: 3.152490 V - Iteración: 0
Tensión AMS - Canal 14: 1.742899 V - Iteración: 0
Tensión AMS - Canal 15: 3.154904 V - Iteración: 0
Tensión AMS - Canal 16: 1.744419 V - Iteración: 0
Tensión AMS - Canal 17: 0.835687 V - Iteración: 0
Tensión AMS - Canal 18: 1.755953 V - Iteración: 0
Tensión AMS - Canal 19: 1.753226 V - Iteración: 0
Tensión AMS - Canal 21: 0.837743 V - Iteración: 0
Tensión AMS - Canal 22: 1.756400 V - Iteración: 0
Tensión AMS - Canal 23: 1.220360 V - Iteración: 0
Tensión AMS - Canal 24: 0.000000 V - Iteración: 0
Tensión AMS - Canal 25: 0.823840 V - Iteración: 0
Tensión AMS - Canal 26: 0.821471 V - Iteración: 0
Tensión AMS - Canal 27: 0.818923 V - Iteración: 0
Tensión AMS - Canal 28: 1.757026 V - Iteración: 0
Tensión AMS - Canal 29: 1.753137 V - Iteración: 0
```

Results obtained by the monitoring thread

As it can be seen in the previous figure, the monitoring thread starts indicating its period and gathers information from every sensor available (only a part of the data obtained is shown in the image). Every periodic iteration of the thread will collect data and send it to the DAQ.

Alarms threads

After developing the monitoring periodic thread, the same periodic structure has been followed for the development of the alarm threads. However, the period will be much shorter than in the monitoring thread as detecting any alarm is much more critical.

It has been decided to divide the alarm thread in two different threads, one for the I^2C devices and the other for the AMS alarms. This decision has been made since the I^2C devices detect alarm information by reading from a log while the AMS detects it through the *IIO_EVENT_MONITOR* subprocess. Furthermore, the conversion time of the I^2C devices restricted the alarm triggering of the AMS too much. Therefore, the period of each thread has been set depending on the most restrivtive conversion time.

Regarding the I^2C devices alarms thread, it has been necessary to use a POSIX sempahore, semaphore, to synchronize the I^2C bus usage and avoid race condition between any other thread.

In regards to the operation of the thread, it calls functions that read the flags registers of the I^2C devices and check if there is an active flag, clean it if necessary and depending on which flag is activated, the event is communicated to the DAQ.

In order to develop the AMS alarms thread, we firstly need the Linux kernel tool *IIO_EVENT_MONITOR*, a generic application to detect events coming from IIO devices, which working along the "xilinx-ams" driver allows us to catch AMS alarms. For this purpose, the *IIO_EVENT_MONITOR* has been modified in order to be able to transmit the detected event information to our main application.

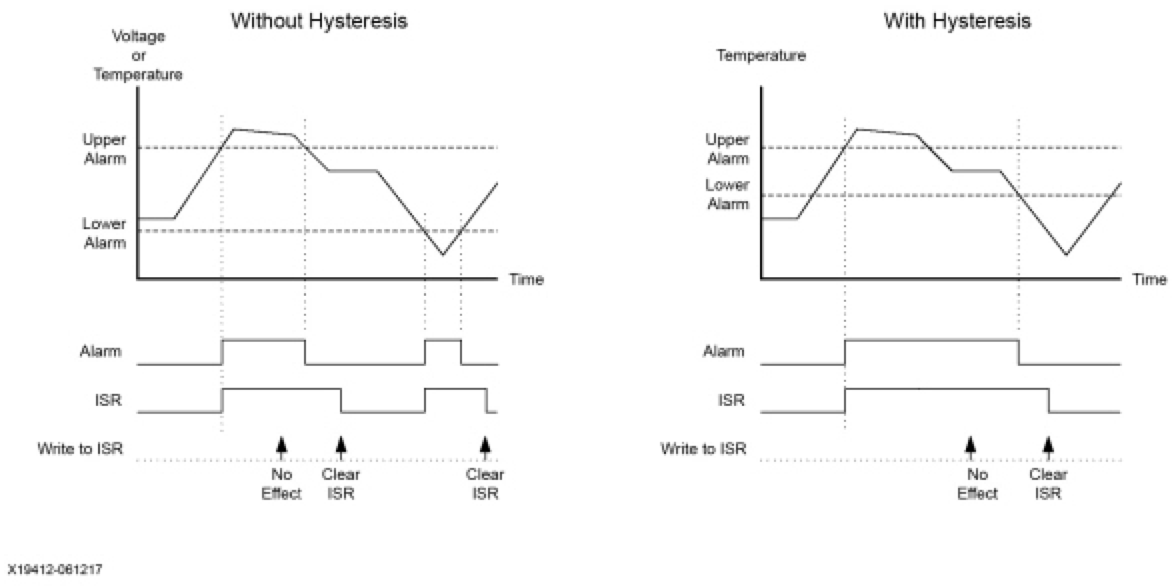
First of all, a segment of shared memory has been established between the main application and the *IIO_EVENT_MONITOR* sub-process to transmit the necessary event information and set up semaphores to synchronise sub-process and thread. One semaphore is used to force the AMS alarm thread to start *IIO_EVENT_MONITOR*, and the other semaphore is used to indicate to the alarm thread that an event has been detected and the thread will handle it accordingly.

As the *IIO_EVENT_MONITOR* does not provide us the value of the magnitude that has triggered the alarm and does not differenciate rising and falling voltage events, the thread has been configured to take care of this.

It should be noted that in the process of debugging this thread, several bugs have been detected in the "xilinx-ams" driver that we have solved to ensure the correct functioning of our application.

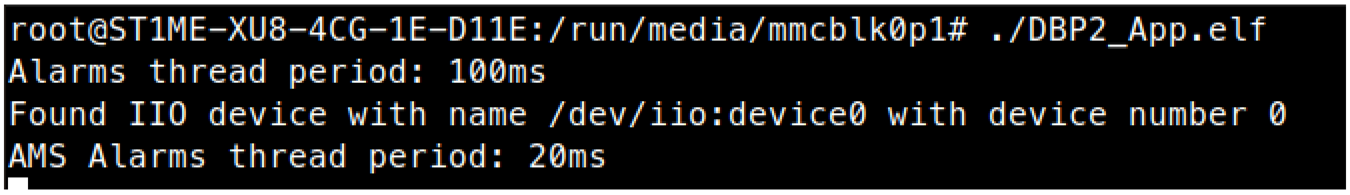
The first encountered bug is due to the driver masking an alarm when it is triggered so that it is not detected again until it has been previously reset to normal values to avoid the same alarm going off constantly. However, the unmasking process proposed an impossible situation and did not allow an alarm to go off more than once despite a previous reset as it should work. By modifying several logical operations, the problem was solved without affecting its performance.

The other problem we found was a lack of functionality of the driver, since the AMS by default enables hysteresis for the temperature alarm and the lower limit places it at 0 degrees. The problem is that the driver does not allow you to modify the lower limit or disable hysteresis, so it only allowed you to enable temperature alarms once. We made the decision to disable hysteresis from the driver since it does not suit our application and with this we find the desired operation of the AMS alarms



Difference between AMS temperature alarm with hysteresis on and hysteresis off

Both alarms threads will be started at the beginning of the application execution, reporting their periods and confirming that *IIO_EVENT_MONITOR* has executed correctly.



```
root@ST1ME-XU8-4CG-1E-D11E:/run/media/mmcblk0p1# ./DBP2_App.elf
Alarms thread period: 100ms
Found IIO device with name /dev/iio:device0 with device number 0
AMS Alarms thread period: 20ms
```

Alarms threads start

Parsing data