Master RADMEP

**Master *Radiation and its Effects on MicroElectronics and Photonics Technologies* (RADMEP)**

# SINGLE EVENT EFFECTS INSTRUMENTATION FOR SYSTEM-ON-MODULE TESTING

Master Thesis Report


Presented by

Mauricio Ernesto Rodriguez Alas


and defended at

University Jean Monnet

9 September 2024

| | |
|---|---|
| Academic Supervisor: | Prof. Frédéric Saigné, University of Montpellier |
| Host Supervisor: | Prof. Luigi Dilillo, University of Montpellier |
| Jury Committee: | Prof. Sylvain Girard, University Jean Monnet<br>Prof. Arto Javanainen, University of Jyväskylä<br>Prof. Paul Leroux, KU Leuven<br>Prof. Frédéric Saigné, University of Montpellier |

Erasmus+ UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE JYVÄSKYLÄN YLIOPISTO UNIVERSITY OF JYVÄSKYLÄ KU LEUVEN UNIVERSITÉ DE MONTPELLIER

# Abstract

Radiation testing and qualification of complex systems is a challenging and demanding process due to the interactions and dependencies between systems. This thesis presents the development of a low-cost, compact, robust, and highly synchronized testing instrument designed to standardize Single Events Effects testing for modern System-on-Chip devices. By increasing logging capabilities and timing synchronization, we can get better control over the systems under test. The instrumentation performance is demonstrated experimentally during a neutron irradiation campaign, showcasing its reliability and ability to improve complex system testing.

# Table of Contents

# List of Abbreviations

| Abbreviation | Definition |
|---|---|
| iES | Institute of Electronics and Systems |
| WP | Work Package |
| SEE | Single Event Effect |
| SEU | Single Event Upset |
| SEL | Single Event Latch-up |
| DD | Displacement Damage |
| TID | Total Ionizing Dose |
| FPGA | Field Programmable Gate Array |
| SoC | System on Chip |
| MPSoC | Multiple Processor System on Chip |
| SoM | System on Module |
| SRAM | Static Random Access Memory |
| COTS | Commercial Off The Shelf |
| HDL | Hardware Description Language |
| SUT | System Under Test |
| UM | University of Montpellier |
| RADIAC | *Radiation et Composants* |
| RADNEXT | Radiation facility Network for the Exploration of effects for industry and research |
| TI | Texas Instrument |

# 1 Introduction

The Institute of Electronics and Systems, under the authority of the University of Montpellier, conducts research under five thematic axes: Energy, Instrumentation, Photonics and Waves, Materials, and Reliability and Systems Under Constrained Environments [1]. Under this last axis, the RADIAC team, conducts research on the topics of reliability of electronics in radiation environments, developing expertise in monitoring, modeling, understanding and testing of components, instruments, and systems [2].

As part of a larger effort to create a network of expertise, facilities, and services related to radiation effects of electronics components and systems, the University of Montpellier and RADIAC group takes part as an academic partner in the EU-funded RADNEXT project (RADiation facility Network for the Exploration of effects for industry and research, Grant Agreement ID: 101008126) [3]. The RADNEXT project is conducted as a joint-research activity at a European level and guided by four work packages (WP) [3], given:

- WP5: Radiation monitors, dosimeters, and beam characterization.
- WP6: Standardization of system-level radiation qualification methodology.
- WP7: Cumulative radiation effects electronics.
- WP8: Complementary modeling tools.

This work is conducted based on the objectives and framework of WP6: Standardization of system-level radiation qualification methodology, which seeks to test and qualify integrated systems under radiation, and to determine which setup and stimuli are most optimal to evaluate the response of complex systems such as memories, FPGAs, and SoC [3]. Specifically, within the WP, the testing of systems is focused on adding high observation capabilities [4], identifying good practices for system-level testing, and creating pass/fail criteria [3]. The proposed instrumentation tries to simplify the radiation testing instrumentation needed in terms of hardware and software, building on past experiments conducted by the RADIAC group.

This document is structured as follows: Section 1 presents the objectives of the work, a brief state of the art and related work, Section 2 discusses the methodology used, the Design and development of the platform, Section 3 provides the results and data obtained, Section 4 discusses the results obtained, and Section 5 presents future outlook work.

## 1.1 Objective of this work

- To improve the understanding of radiation effects on SoC by means of improved instrumentation.
- To identify the key observability metrics in SoC under radiation.
- To observe previously defined metrics by proposing improved instrumentation hardware and software.
- To improve controllability parameters for SoC under radiation.
- To propose a way to synchronize and correlate collected data with observed radiation effects.

## 1.2 Contributions

System-level testing of complex systems presents multiple challenges and opportunities. The increased interest in using COTS components for space applications requires new testing approaches to evaluate these systems in harsh environments, as these components are usually not designed for radiation effects but provide cost reduction and design flexibility for New Space actors [5]. System-level testing presents itself as a more efficient testing method, that incorporates all the complexities of the system into the test.

Some advantages of system-level testing are the possibility to observe the fault propagation in the system, compare mitigation techniques, and get information about complex failure modes [6]. Some disadvantages are the inherent complexity of the system, additional resources, caches, and memories, which make some systems highly susceptible to single-event induced failures [7]. Thus, identifying failure modes of the devices becomes challenging, more so if the system is running multiple processing cores.

This work proposes a low-cost platform for testing complex devices such as FPGAs, MPSoC, and/or memories that adds observability and controllability of target devices by using a reliable FPGA platform, allowing for further customization of the test procedure, including but not limited to implementing worst-case conditions of the target device i.e. temperature, voltage, and application following the SEE Testing Guidelines of [8] and previous observations of high observability and system-level testing demonstrated by the research group in [9].

## 1.3 Scope and Limitations

- The scope of this work is limited to SEE of devices and will not study the effects of TID.
- The electronics systems under study are limited to the commercial components previously found in literature, namely AMD Zynq-7000, Artix-7, and Microchip Polarfire MPSoC.
- The experimental validation of the platform is limited to SEE under a neutron beam at the ISIS ChipIR facility.

## 1.4 State of the Art and Background

### 1.4.1 Neutron Radiation

Atmospheric neutrons are generated from the interaction of cosmic rays with the atmosphere. The flux of particles is dependent on altitude and latitude [9], and affects most aircraft at flight altitudes around 10 to 12 km [10]. Neutrons are uncharged particles and can penetrate the material deeply without being magnetically affected. Neutrons will interact with the nucleus in the device by recoils, generating ionizing nuclear fragments [11]. Not only are neutrons a concern for avionics, but they increasingly become a reliability concern for highly integrated and downscaled devices.

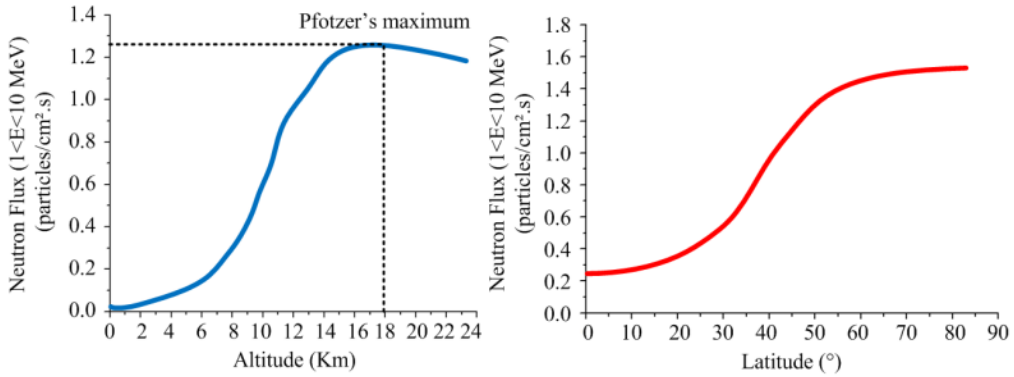



*Figure 1. Neutron Flux by altitude (left) and latitude (right). Adapted from [12].*

Neutrons are increasingly capable of producing Soft Errors with each new technology node. As devices shrink their size, the upset probability increases with the reduction of the charge necessary to induce an upset [13]. These Soft Errors can negatively affect functionality and reliability of critical systems, such as avionics, memories or FPGAs.

### 1.4.2 Radiation Effects on FPGAs

FPGAs are reconfigurable integrated circuits widely used for their flexibility and performance. However, their complexity makes them particularly vulnerable to radiation-induced errors. Single-Event Effects are caused by the interaction of particles with materials in electronic devices [14]. FPGAs can be affected in their integrated circuits including flip-flops or memory cells. Usually, error detection and correction are employed to mitigate these effects. Some, but not all, relevant radiation effects on FPGAs are described next.

**Single Event Upsets (SEU)**

Upsets can happen in memory elements. SRAM-based FPGAs are particularly susceptible to this effect. This can corrupt the data or change the logic states inside the device, leading to undesired behavior or system failures. Techniques such as Triple Modular Redundancy is also employed to detect and overcome SEU-induced soft errors [15], where three instances of the same design are utilized with a majority voter to ensure no corrupted data is passed to the system.

**Single Event Latch-up (SEL)**
SEL is a condition where a high-energy particle induces a parasitic PNPN thyristor structure in the device, causing a short circuit between power and ground. This can result in a significant increase in current, potentially leading to permanent damage.
If it is not stopped it will lead to thermal failure. Furthermore, SEL probability is known to increase at higher temperatures [16].

**Configuration Memory Corruption**
FPGAs rely heavily on configuration memory to define their operation. Radiation can cause corruption in this memory, leading to incorrect circuit operation. Detection and correction of these errors can be mitigated by configuration memory scrubbing, where the memory is being read constantly, correct, and report any corrected errors. Correction of the memory is carried out by partial or full reconfiguration of the memory [15].

### 1.4.3 System-Level Testing

With the increasing interest in testing COTS systems for atmospheric and space applications, component-level testing has become a time and resource intensive process, therefore, a simplified approach is needed, where only the most critical components and tested [6]. With increasing complexity in devices, such as memories, FPGAs, SoC, and multicore MPSoC, it becomes increasingly hard or nearly impossible to partition these systems. We want to test the whole functionality of the system and monitor it during the irradiation test. The main advantage of system-level testing is we acquire radiation data for the whole system, incorporating the margins given by the system into the performance [6].

Th complexity of SoC devices naturally increases the resources needed to properly test the devices. Only in the reconfigurable part of the FPGA, the Configurable Logic Blocks plus the non-reconfigurable cores will present different cross sections and sensitivities. Secondly, the design itself can be a source of amplification or masking of errors [17]. Finally, with the number of resources in a complex device, there needs to be a selection of data streams that need to be monitored, and later correlate any effects seen on these devices to a SEU or SEFI. To reduce complexity and recurring engineering costs, a trend of using FPGA-based test setups has emerged, especially to test other FPGAs or memories [17].

### 1.4.4 Related Work

For SoC testing of SEE, evaluation or development boards are utilized [18]. One of the problems is the reduction in controllability of the device, since in the case of a SEL or SEFI, there is no independent control of the SUT power. The gain in time by using development boards comes at the expense of board complexity, since a lot of hardware overhead is needed. Additionally, it has been shown that the extra components increase the risk of system failure [6]. Some solutions have been proposed for the standardization of radiation testing.

Besides the development boards traditionally used for testing, some long-term radiation testing platforms exist, such as the CRaTeBo modular platform used for SoC, FPGA, and memory testing. The board presented uses radiation-tolerant components and custom

ASICs designed at CERN, which makes it expensive to source and manufacture [19]. Despite fulfilling all the objectives for the testing application, companies are still interested in testing a more diverse set of COTS SoC, with reduced testing cost and time to test.

Another example of long-term measuring platform is the one presented in [20], developed by CERN to monitor radiation exposure of various equipment and characterize radiation induced effects. It has the capability to measure SEE, TID and DD by using commercial memories, RadFets and PIN diodes.

### 1.4.5 Improved Observability and Controllability

An approach for improved observability has been demonstrated in [21] in which not only the internal SoC status in monitored, but also the SUT total current is measured and checked for determining different types of SEE. Additional peripherals are also added, to be able to correlate the data collected and have a better understanding of the effects on the system: real time clock, temperature, and communication are added to the metrics collected.

This correlation of data using enhanced metrics has been demonstrated by previous experiments in the group, presented in the work by the RADIAC group in [22], where with the additional temporal synchronization, different SEE events were classified by using current monitoring. The key addition in this approach is the addition of precise timestamps for every measurement, allowing faster response to different events. In addition to precise timing, statistical methods are also employed for the data analysis, where each type of SEE can be calculated separately, by using the rate at which SEU or SEFIs happen, it's possible to distinguish between them, this has been proven in [17]. The addition of precision timestamping allows for the correlation of data, from different sensors, and to gain a better understanding of how radiation affects the device.

# 2 Methodology

## 2.1 Methodological Approach

First, a comparison of existing platforms, existing monitoring techniques and instruments was conducted, identifying the appropriate parameters that the system needs to monitor, and comparing to existing solutions in terms of complexity, cost, ease of use, parameters monitored.

Second, once key metrics are identified, an initial design phase is carried out, listing the Use Cases of the Platform, as well as its overall requirements. Finally, after requirements are gathered, an initial system architecture is proposed and then iterated until the system fulfills all requirements. We then move on to component selection, based on previous experiments or literature. Since we don't need the components to be tolerant to radiation, the costs and complexity of selection is diminished.

A hardware solution was developed, based on a custom PCB, utilizing an FPGA platform that can be reconfigured for future iterations. The proposed instrument was validated during a neutron irradiation campaign in the ISIS ChipIR neutron facility.

## 2.2 Use Cases

The main use of the platform is to facilitate the testing, monitoring, and reporting of a SoC under test. To simplify the hardware needed to conduct radiation campaigns, while keeping the cost of development low. As presented in 1.4.4 the main method of testing FPGAs and SoCs is to use evaluation kits, but these come with a lot of hardware overhead that can make it even harder to diagnose SEE during irradiation and may not be suited to the common objective of testing the SoC/FPGA without much error masking from the additional hardware. The proposed solution is made open source to open the possibility of collaboration and for any researcher to quickly develop a base system or to use widely available commercial modules.

## 2.3 Design and Development

### 2.3.1 Requirements

The whole testing system is composed of two boards:

- Monitoring Board (this work): this thesis describes the work done with this board and design, along with the Carrier board it is part of a bigger system that interfaces and synchronizes with the SUT and communicates back with the host computer.
- Carrier Board: this board provides the interfaces, power, sensors and communication to the Monitoring Board. This board holds the SoC under test in the irradiation room. This board was designed by Mattos A. *et al.* and it is described in more detail in [23].

Together, both boards comprise the full instrument, and are required for the proposed goals. For clarity, a diagram and architecture of the whole system are described in 2.3.2 System Architecture.

The requirements of the Monitoring Board are shown in Table 1:

| ID | Requirement | Note/Rationale |
|---|---|---|
| 1 | The system should have a minimum response time of 100 millisecond. | This frequency is selected to have a reasonable response time in case of SEL, since the thermal effects are in the millisecond range [24]. |
| 2 | The system should operate at a minimum wired distance of 100 meters. | The maximum cable distance between the control room and the irradiation room. |
| 3 | The system should be able to control peripheral devices. | Such as sensors, GPIO, and power. |
| 4 | The system should be able to measure temperature from the device. | To have data regarding temperature dependence of different effects. |
| 5 | The system should be able to measure current and voltage from the SUT. | To monitor current consumption and detect SEL and anomalies. |
| 6 | The system should be able to timestamp the incoming data from the SUT. | To be able to precisely correlate different events in time. |
| 7 | The system should be able to output data to a serial terminal in the control room | To be able to monitor the experiment in real time |
| 8 | The system shall be able to read a watchdog to monitor the SUT for hangups, log the error and restart the system. | In case we have hangups in the SUT, the Monitoring board can try to reboot the system for recovery. |
| 9 | The system shall be able to detect latch-ups and cut the power when a current threshold is surpassed. | To be able to detect and log current events, and be able to cut power to the SUT to avoid thermal damage. |
| 10 | The system shall be able to interface to a host computer with minimal setup. | To facilitate experiment setup and reduce configuration errors. |
| 11 | The system should have multiple power options. | To accommodate for different control room setups in different facilities. |

*Table 1. Monitoring Board Requirements.*

Additionally, some key design drivers were specified, to guide the selection of components, interfaces, and features. The summary of the key design drivers is listed in Table 2:

| ID | Design Driver | Note |
|---|---|---|
| 1 | It would be useful to be able to program and reprogram the SUT board over cabled distances of up to 100 meters. This would be done using the JTAG programming interface. | During radiation testing, if the device fails, it's useful to reprogram the board, to try to recover device functionality, and to change the device design for different tests or comparisons. |
| 2 | Most radiation facilities provide Ethernet connections by default. | This connector is the common denominator between facilities, and could be used in almost all cases. |
| 3 | The Carrier Board components will be indirectly irradiated and might be subject to SEE and TID. | We want to minimize components, complex components that could prevent observability of the device. |
| 4 | Both boards should use COTS components for availability, cost, and scalability. | This would minimize sourcing issues; parts are easily replaceable, and we can keep the overall system cost low. |
| 5 | Both boards should be easily manufacturable. | All components should be hand solderable, including passives and Surface Mount Devices. |

*Table 2. Design Drivers for the whole system.*

### 2.3.2 System Architecture

Figure 2 shows the initial system architecture for the monitoring board. So far only the communication protocols internal to the PCB have been defined, and the justification of ICs based on previous requirements will be detailed in section
2.3.3 Selection of Components.

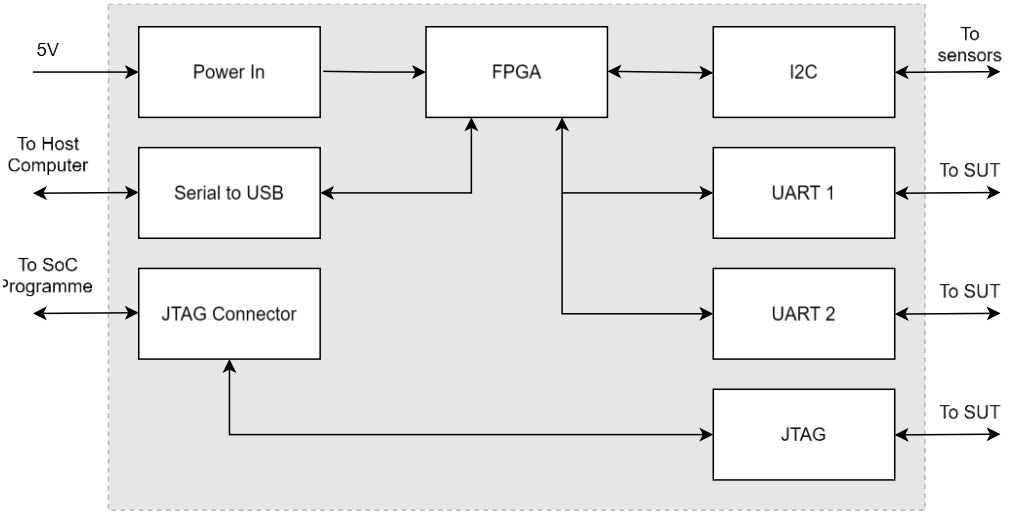



*Figure 2. Simplified System Architecture for Monitoring Board.*

From the architectural diagram, going from left to right, we can see a simple power interface, this is selected to be 5V, since a common option found from power supplies or wall adapters, a Serial to USB component simplifies the connection to the host computer and allows for interaction with the rest of the system, a JTAG connection where the different SoC programmers are connected to program/reprogram the SUT. We can see that the FPGA sits at the heart of almost all synchronization tasks, making this component crucial to the overall setup. On the right side, we see the selected protocols that the FPGA will be communicating: I2C for the temperature, current, GPIO, and Watchdog devices; two UART ports that will be used to communicate with the SUT, and finally the outgoing JTAG connection to the device.

### 2.3.3 Selection of Components

With an initial list of requirements, design drivers and overall architecture to guide the selection, the following components are selected for the Monitoring Board:

| ID | Component | Qty | Note/Rationale |
|---|---|---|---|
| 1 | TEM0001-01A-ABC-2 SMF2000 | 1 | Flash-based FPGA to synchronize and control the experiment |
| 2 | FT4232H-56Q MINI MDL | 1 | UART to USB FTDI Chip with 4 channels |
| 3 | SN65LVDT41PW | 1 | LVDS Transceiver for JTAG programming |
| 4 | SN65HVD30MDREP | 2 | RS485 3.3V Transceiver |
| 5 | SN65HVD232D | 2 | CAN 3.3V Transceiver |
| 6 | P82B96DR | 1 | I2C Long Distance Buffer |
| 14 | HARTING 09455511123 | 1 | RJ45 3-port Connector |
| 17 | DCJ200-10-A-K1-K | 1 | DC Power Jack |
| 18 | TE Connectivity 5103310-1 | 1 | JTAG connector |

*Table 3. Selected Components for Monitoring Board.*

| ID | Component | Qty | Note/Rationale |
|---|---|---|---|
| 3 | SN65LVDT14PW | 1 | LVDS Transceiver for JTAG programming |
| 4 | SN65HVD30MDREP | 2 | RS485 3.3V Transceiver |
| 5 | SN65HVD232D | 2 | CAN 3.3V Transceiver |
| 6 | P82B96DR | 1 | I2C Long Distance Buffer |
| - | TPS35AA38AGADDFRQ1 | 1 | External Watchdog |
| - | TMP100NA3K | 1 | I2C Temperature Sensor |
| - | LP38693MP | 1 | Low Dropout Regulator for Setup |
| - | MIC29302AWD | 1 | SoM Power Supply |
| - | INA219BIDR | 1 | I2C SoM Power Monitor |
| - | TCA9535PWR | 1 | I2C GPIO Extender |

*Table 4. Mirror Components for Carrier Board.*

Table 3 and Table 4 shows the main components selected for the Monitoring and Carrier board. To illustrate this interface dependency a full system diagram is shown in Figure 3, based on the full system design by Mattos A. *et al.* [23]. For a full breakdown of the components and their costs, refer to Appendix A – Full Bill of Materials for .

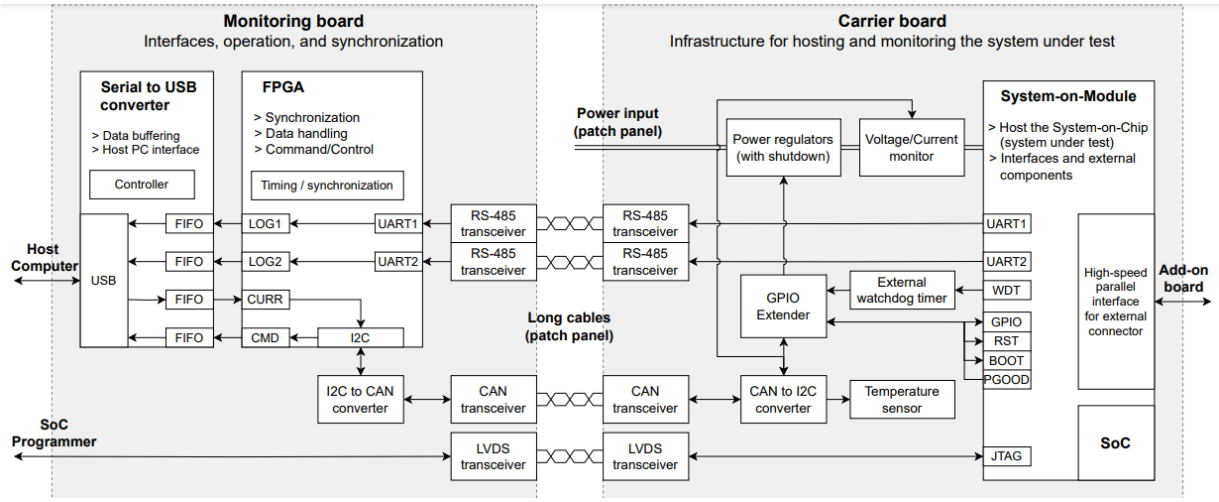



*Figure 3. Full setup diagram, with Monitoring (left) and Carrier board (Right). Adapted from [23]*

The component selection from Table 3 and Table 4 was based on the previous requirements, previously utilized components, and past experiments:

- The TEM0001-01A-ABC-2 (SMF2000) FPGA was chosen due to the previous experiments conducted in [22] and [25]. The component shows high tolerance to radiation effects, due to being a flash-based FPGA, some generations of these FPGAs have shown no radiation induced upsets in the flash cells, in contrast to SRAM based devices [26]. Additionally, there was current availability of the part in the laboratory stock, as it had already been ordered for previous experiments.
- FTDI Chip FT4232H-56Q was selected to simplify the connection between the host computer and the Monitoring Board, as it provides four different serial ports with different configurations, only requiring one single USB port and minimal drivers [27]. This allows the host computer to monitor four serial ports

at the same time, and as proven in [28] [29] it allows for data transfer rates of up to 12 Mbit/s.

- UART has been utilized in similar experiments before and has proven to be simple to use as a communication protocol between the host computer and a device under test while requiring minimal hardware to setup [30].
- The UART over RS-485 protocol has been proven in multiple experiments to be convenient and easy to use, especially for long-distance communication links for FPGA applications [31].
- I2C over CAN, a reference designed was tested by Texas Instruments in which I2C is converted into differential lines, that can be used for long distance communication in noisy environments [32]. In this setup up to 300-meter cables can be utilized by varying the setup time of the I2C protocol.
- JTAG over LVDS, a reference TI transceiver is used, due to its low power consumption, potential high data rate, and long-distance capability. It is an efficient protocol, with simple termination and low noise generation [33]. Since the topology of using a JTAG programmer is a point-to-point scheme, requires minimal hardware. This would allow to fulfill the programming over long distances requirement, while also adding additional debug capabilities to the system.

For reference, the Carrier board is shown in Figure 4 is an essential part of the monitoring system, since it hosts the SUT under the beam, contains the power regulators for the SUT, interfaces with the Monitoring board to provide the status of the device, hosts the I2C peripherals (GPIO, temperature, current, watchdog), and hosts the mirrored transceivers to convert the signals for long cables communications, including the LVDS, CAN and RS-485 transceivers.

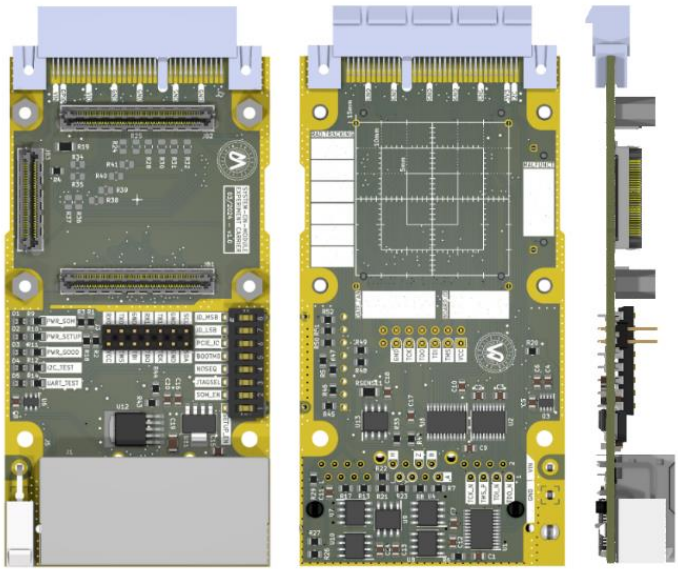


*Figure 4. Carrier Board. Adapted from [23].*

The Carrier board in conjunction with the Monitoring board, comprise the full testing setup. The full details of the board and further results from the setup validation are published in the Journal of Instrumentation in [23]. The work covered in this thesis

covers the design of the Monitoring board, as part of a bigger effort to make radiation testing more accessible and reduce test development time.

### 2.3.4 Prototyping and Development of PCB

After the component selection and architecture were finalized, an initial PCB design was commissioned, using KiCAD [34], this is a free and open-source EDA tool for PCB design and manufacture. It has been partly supported by CERN as an Open-Source initiative [35] to make it more efficient and be able to support designs with up to 32 layers.

Based on the previously described system architecture and the validated requirements, a series of PCBs were designed, including the Monitoring, Carrier board, and additional boards to test the Carrier board full functionality. This work covers the design of the Monitoring board, and its full schematic diagram can be found in Appendix B.

The boards were sent for manufacturing using JLCPCB, which is an assembly service for PCBs, the boards were designed and ordered using a 4-layer configuration, using a Signal-Power-Ground-Signal stack up, as it is commonly used to have signals close to a reference plane [36]. The details for the PCB fill, trace width and pad dimensions can be found in Table 5. The rendered model for the front and back planes of the PCB can be seen in Figure 5. Rendered front and back PCB assembly.

| **Specification** | **Value** |
|---|---|
| Layers | 4 |
| Layer Stack up | Signal-Power-Ground-Signal |
| PCB dimensions | 115 x 75 mm |
| Base Material | FR4 Standard TG 135-140 |
| PCB Thickness | 1.6 mm |
| Surface Finish | HASL (with lead) |
| Outer/Inner Copper Weight | 1 oz / 0.5 oz |
| Vias | Plugged, min size 0.3 mm |

*Table 5. PCB Construction Details.*

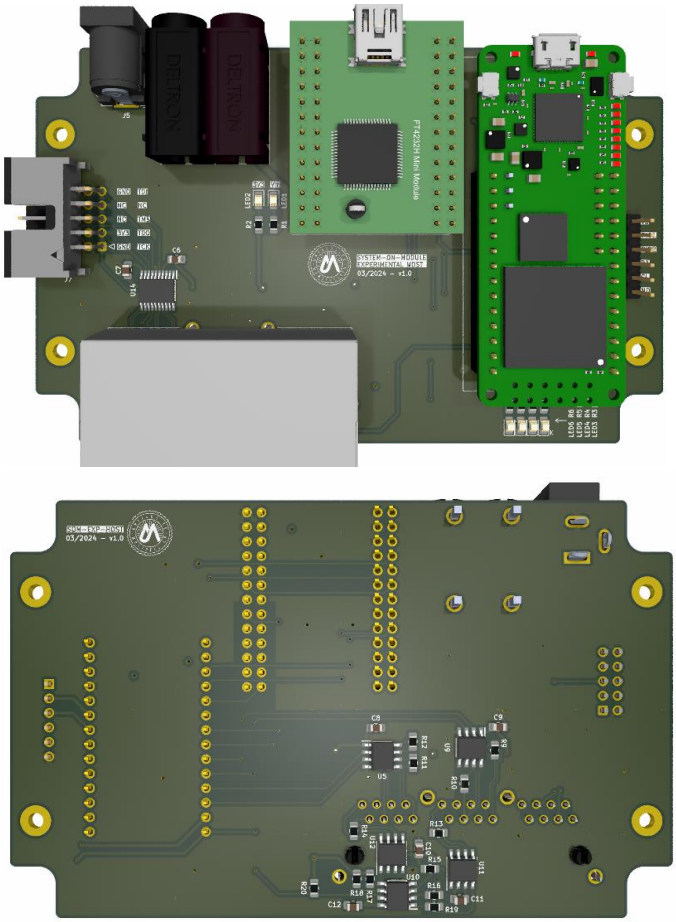


*Figure 5. Rendered front and back PCB assembly.*

The PCBs and its components were assembled using the stencil provided by the PCB manufacturer, a hot air gun and low-temperature solder, an initial continuity test was carried out, next the JTAG interface was tested with a simple circuit, where only the JTAG over LVDS was tested, after the initial test and some debugging with the programmer, a successful connection to a test FPGA was conducted. The JTAG Programmer used to program the device is based on a similar FTDI device that is used to simplify the connection between the host computer and the Monitoring board.

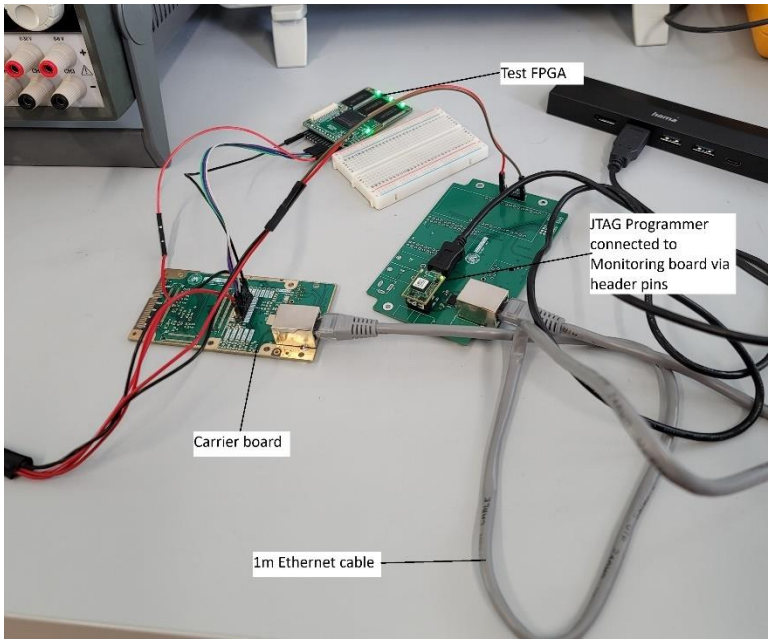



*Figure 6. Test setup for JTAG over LVDS.*

Figure 6 shows the test setup used to test the first interface of the boards, where only the JTAG over LVDS transceivers are connected and powered up, the Monitoring board is connected to a TEM0009-02 programmer, used for programming and connecting to Microsemi FPGAs [37]. The programming test is conducted using the supported Libero tools and FlashPro-5. A successful test is achieved, and signal integrity is tested.

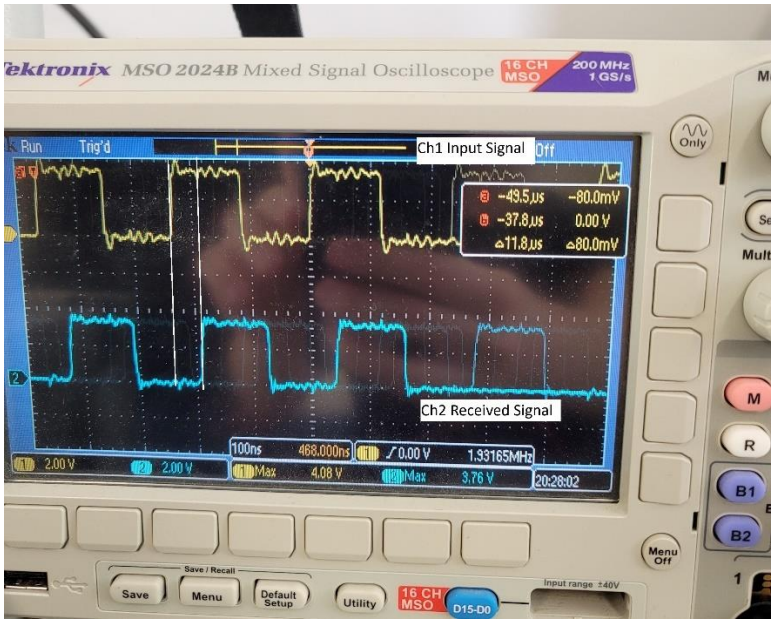



*Figure 7. Board to board signal test.*

Figure 7 shows the board-to-board signal test performed: using a function generator a 3.3V 2 MHz signal is introduced in Channel 1 in the Monitoring board and the output signal is measured in Channel 2, we can see that some of the high frequency components

are attenuated. Although not ideal, the signal quality at this point is sufficient for our reprogramming test.
After a basic signal test was conducted with a 1-meter ethernet cable using the JTAG programmer, multiple distances of cables were tested, and different programmers tested, since Microsemi and Xilinx FPGAs use different programmers and software tools. The summary of results is shown on Table 6. Notice that the programmer frequency for the FlashPro-5 is dropped immediately to 1 MHz if the cable length is more than 1 m.

| Programmer/FPGA | Programmer Frequency (MHz) | Cable Length (m) | Programming Status |
|---|---|---|---|
| FlashPro-5/Microchip | 10 | 1 | PASS |
| | 1 | 10 | PASS |
| | 1 | 30 | PASS |
| | 1 | 50 | FAIL |
| | 1 | 100 | FAIL |
| Vivado/Xilinx | 10 | 1 | PASS |
| | 4 | 10 | PASS |
| | 2 | 30 | PASS |
| | 1 | 50 | PASS |
| | 1 | 100 | FAIL |

*Table 6. Results for JTAG Programming Test.*

One issue encountered with the JTAG programmer, specifically with the FlashPro/Microchip programmer was the tool was not able to program the FPGA test board if the cable length was greater than 30 meters or if the programmer's frequency was anywhere above 1 MHz, while the Vivado/Xilinx tools and FPGA was capable of being programmed up to 50 meters, but using 1 MHz for the programmer's clock. Figure 8 below shows the setup used to test the programming over JTAG, this setup was also used to test the I2C communication. At the beginning it was tested using an Arduino and the code found in Appendix D – Code used to test I2C Communication, this allowed to test if the I2C devices were responding properly, namely the temperature sensor, the current sensor and the GPIO IC. This test was conducted successfully using up to 50 m.

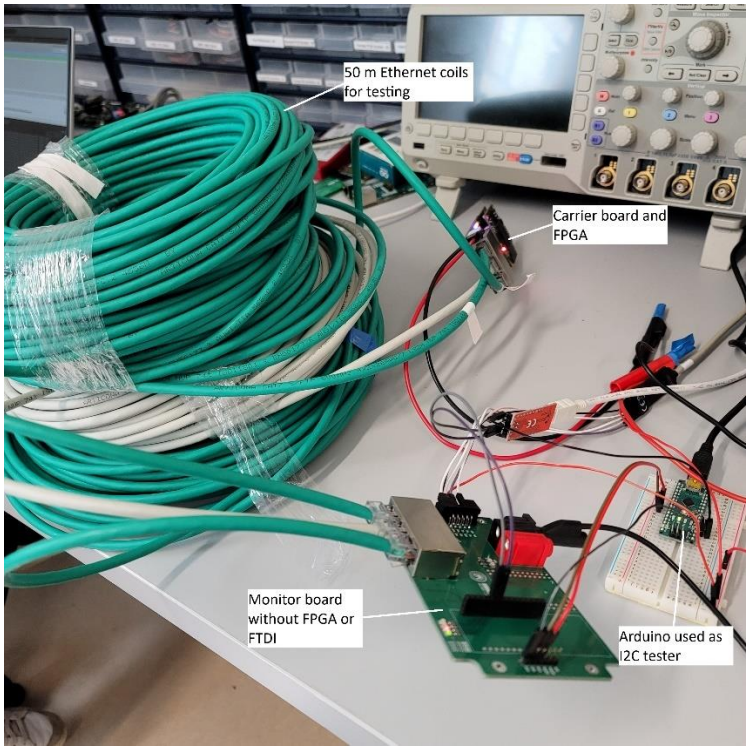



*Figure 8. I2C test setup.*

The Arduino in this setup was used as an I2C connectivity test, since the required program for the SMF2000 FPGA was in development. This is the FPGA in the Monitoring board that is used to synchronize the received data. After these initial steps, the I2C, JTAG, and UART interfaces were validated, the development of the SMF2000 FPGA design was conducted.

### 2.3.5 Prototyping and Development of HDL blocks

The following section describes the design and development of the HDL code used inside the SMF2000, first the block diagram of the overall system is presented, and immediately the relevant details of each component are described. The implementation is purely HDL, and it was synthesized and implemented using Libero SoC 2023.2, the simulations were performed and visualized in Vivado Xilinx 2018.2 to have a better visualization and debugging experience.

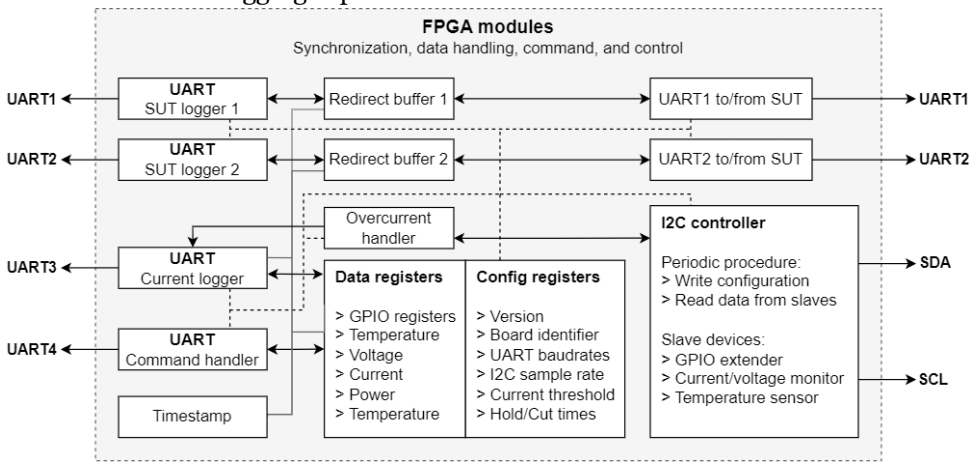



*Figure 9. Block diagram of HDL design for SMF2000. Adapted from [23].*

Figure 9 describes the block diagram of the logic used inside the SMF2000 FPGA. Following the diagram, on the left side, we can see four UART channels that are connected physically to the FTDI chip, Channel 1 and 2 are used to receive logging data from the SUT, and are connected to a Redirect Buffer that acts as a FIFO and adds the precise timestamp to the data received. On the right side, the redirected Ch1 and Ch2 are physically connected to the RS-485 transceivers described previously.

Channel 3 is used exclusively to monitor the device current, the current data is registered in the Register block (Data registers) while the data is constantly being updated by the I2C controller, additionally this channel has an Overcurrent Handler that detects when the current data is over certain thresholds and power-cycles the device for a specified amount of time.

Channel 4 is used to send commands to the Data and Configuration Register instance. It can read-and-write any of the registers, and it is used to get the data from the remaining I2C peripherals (GPIO, Temperature).
The I2C Controller coordinates the I2C peripherals read/write operations, and writes the obtained data from Current Monitor, GPIO and Temperature to the Data Registers.

The design is broken down into simpler building blocks, this is to simplify the design and testing. The full design is composed of the following blocks:

- Top: highest entity, encapsulates all other components.
- UART: implements serial data communication protocol. Used to communicate to the FTDI chip and to the SUT device.
- Current Report: the Overcurrent Handler constantly monitors the electrical current value, and reports with a bit flag if an overcurrent event occurs. Reports overcurrent events to the UART interface.

- Overcurrent Handler: manages the GPIO and on/off times to be able to power-cycle the device in case an overcurrent event occurs.
- Command: receives commands from UART Channel 4 to be able to read/write to the Data and Configuration Registers.
- FIFO: a simple First In-First Out memory that holds incoming data until it can be written.
- Hex to UTF8: converts Hexadecimal values used inside the HDL blocks to human-readable UTF-8 characters.
- UTF8 to Hex: converts UTF-8 characters coming from the UART command channel 3 to hexadecimal values used inside the HDL blocks.
- I2C: protocol implementation, data, and state machine implementation of the protocol.
- I2C Controller: handles all different I2C device read/write operations in a single block, including GPIO, INA Current Monitor, and GPIO. Also handles initial configuration for the peripherals.
- I2C GPIO: implements register-level read/write operations required for the configuration and functioning of the peripheral.
- I2C INA: implements register-level read/write operations required for the configuration and functioning of the peripheral. Handles the calibration operations.
- I2C TMP100: implements register-level read/write operations required for the configuration and functioning of the peripheral.
- Redirect Buffer: receives UART data from the SUT, adds the data to a FIFO and sends the data to the host UART with a timestamp.
- Registers: stores Data and Configuration registers. Implements 10 Configuration registers and 7 Data registers for the I2C peripherals.
- Timestamp: implements millisecond timestamp that is used by the rest of the system.

The following section describes each in detail and how they relate to each other. For brevity, the full state machine diagrams, HDL code, and in-depth explanations are shown in Appendix E – VHDL Code and State Machines.

**UART - UART.vhd**

The UART serial interface was utilized in other SEE experiments in [22] [38] where it was used successfully and simplifies the communication between devices and the host computer. It implements a standard UART controller based on the protocol description in [39], although the IP supports flow control, it is not implemented in this design, to save physical pins in the PCB. The UART block is central in communication between the host computer and the SUT, and it is instantiated multiple times throughout the design, it's utilized by the Redirect Buffer, Command Interface, and Current Report.

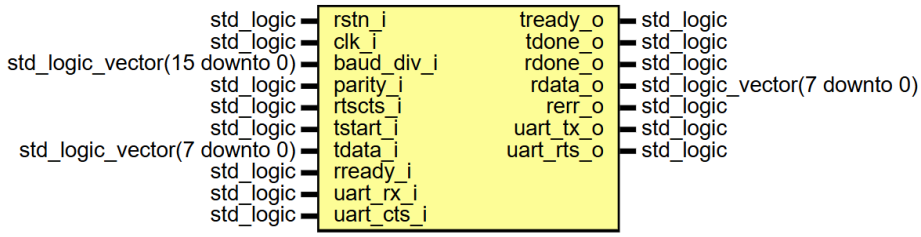



*Figure 10. UART block design.*

**Redirect Buffer – redirect_buffer.vhd**

This block makes use of the UART and timestamp implementation to buffer the incoming data from the SUT, add a precise timestamp using a FIFO memory, and forwards it to the host computer serial by using a second UART instantiation. Additionally, the redirect buffer will detect if data is being lost, and will skip sending timestamps, to avoid the loss of data from the SUT.

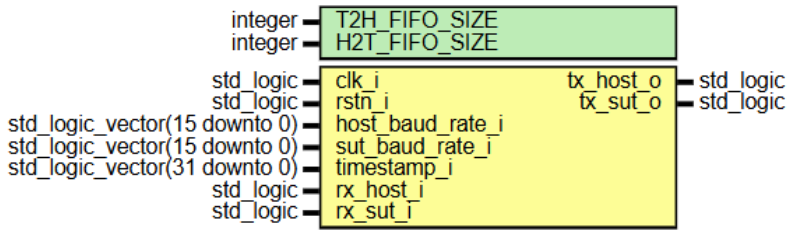



*Figure 11. Redirect Buffer block diagram.*

**Current Report – current_report.vhd**

The current report uses one UART instance to report when an overcurrent event occurs, it uses the *overcurrent_o* output logic to signal if the current threshold has been reached, and to timestamp the event. Additionally, it uses one process to increase the sample rate temporarily until the internal FIFO is full of current measurements. The most important inputs are the *curr_th_i*, the current threshold and *curr_rdata_i*, the received current data, using both these inputs to make the internal comparison for an overcurrent event.

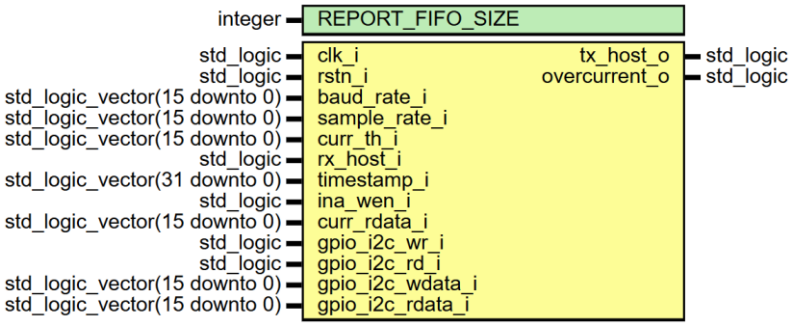



*Figure 12. Overcurrent Report block diagram.*

**Overcurrent Handler – overcurrent_handler.vhd**

The Overcurrent Handler takes the *overcurrent_i* input that is connected to the Current Report *overcurrent_o* output. It uses an internal state machine to turn the SUT on or off in case of an overcurrent event. It uses an internal counter to compare with the input values *on_time_i* and *off_time_i,* which controls the time the SUT is switched on and off in case of an overcurrent event.

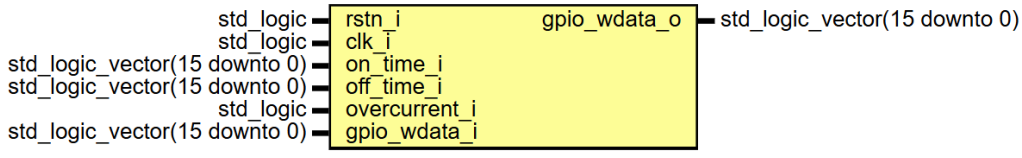



*Figure 13. Overcurrent Handler block diagram.*

### Command Interface – cmd.vhd

This interface makes use of the *tx_o* and *rx_i* pins to communicate with the host computer through Channel 4 of the FTDI, internally it makes use of a UART instance, and uses the *uart_rdone_w* signal to control the state machine and determine when the command has been received to process. The input/output to the block is determined by the received command that is used to read/write from the Registers block. It also instantiates *hex_utf8.vhd* and *utf8_hex.vhd* to convert the outgoing reads and the incoming commands, respectively.

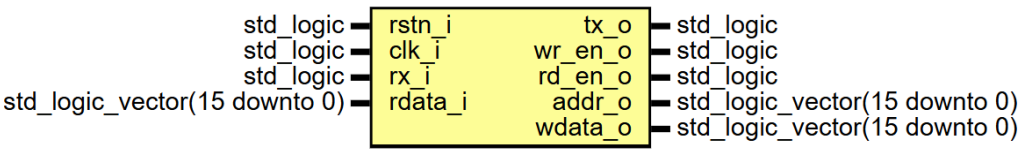



*Figure 14. Command Interface block diagram.*

### Registers – registers.vhd

The registers are comprised of 10 configuration registers that hold informational values such as the program version, board name, I2C and UART baud rates, and the configuration for the current monitoring, including the current sample rate, current thresholds, and on/off times when an overcurrent event is detected.

There are 7 data registers, the GPIO Tri State, GPIO Read/Write values, the current value of voltage, power, current read from the INA, and the temperature value from the TMP100 IC. This allows the system to be fully configurable, this is accomplished by using the Command Interface via the serial terminal or a Python script. With this easy-to-use interface, we can change any required parameters at runtime.

The Registers are connected to the Command Interface to read/write the corresponding registers, and to the I2C controller, which updates the corresponding register value. All the read/write operations are controlled with five ports: *wr_en_i, rd_en_i, addr_i, wdata_i, rdata_o* – write enable, read enable, address, write data, read data.

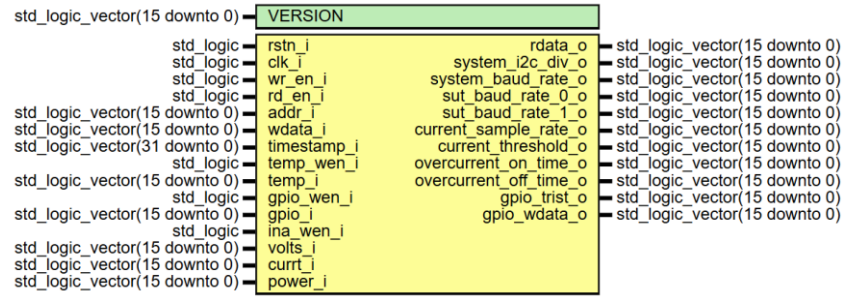



*Figure 15. Block diagram (top) and Data and Configuration Registers (bottom).*

### FIFO – fifo.vhd

Implements a simple memory that can hold data whenever is needed. It has two generic parameters *FIFO_SIZE, DATA_WIDTH* to enable the FIFO to be flexible to the implementation, this way we can reuse the FIFO all over the design.
The main control ports are the *write_i, read_i, data_i,* and *data_o,* which control the write enable, read enable, input data, and read data, respectively. Additionally, the *empty_o* and *valid_o* ports allow for data flow control to only read or request data when the memory is not empty, and there is valid data on the memory.

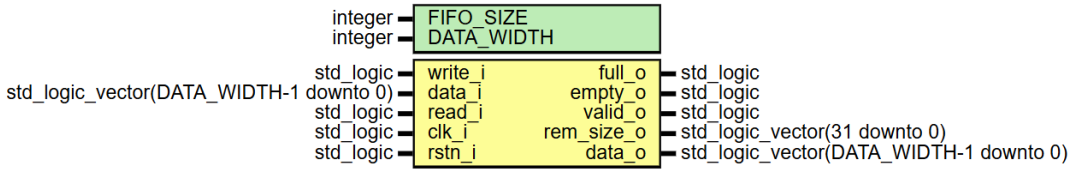



*Figure 16. FIFO block diagram.*

### Hex to UTF8 – hex_utf8.vhd

Converts Hexadecimal values used inside the HDL blocks to human-readable UTF-8 characters. This is an asynchronous block and will keep converting the input without any control signals. Mainly used in conjunction with the Command Interface.

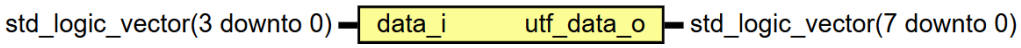



*Figure 17. Hexadecimal to UTF-8 block diagram.*

### UTF8 to Hex – utf8_hex.vhd

Converts UTF-8 characters coming from the UART command channel 3 to hexadecimal values used inside the HDL blocks. This is an asynchronous block and will keep converting the input without any control signals. Mainly used in conjunction with the Command Interface.

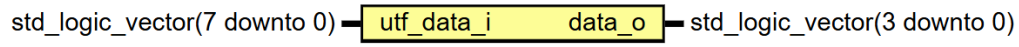



*Figure 18. UTF-8 to Hexadecimal block diagram.*

### I2C – i2c.vhd

This block diagram makes the connection between the peripherals and the I2C controller. It implements the Standard-mode with a Standard Bit Rate of 100 kbps and the two wire communication with the SDA and SCL lines, Serial Data and Serial Clock following the protocol description from [40]. The bit rate parameter is fully configurable, to account for any propagation delay in the long-wired connection, making it possible to reduce the frequency of the I2C communication, and guaranteeing the peripheral communication works.
This IP is instantiated by the I2C controller to select which peripheral is being addressed.

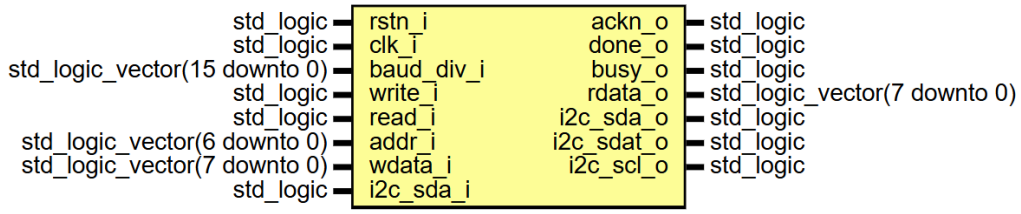



*Figure 19. I2C block diagram.*

**I2C GPIO – i2c_gpio.vhd**

The GPIO block implements the state machine for the required configuration and read/write operations to communicate with the TCA9535 Bus Expander
implements register-level read/write operations required for the configuration and functioning of the peripheral, following the required programming steps listed in the datasheet [41].

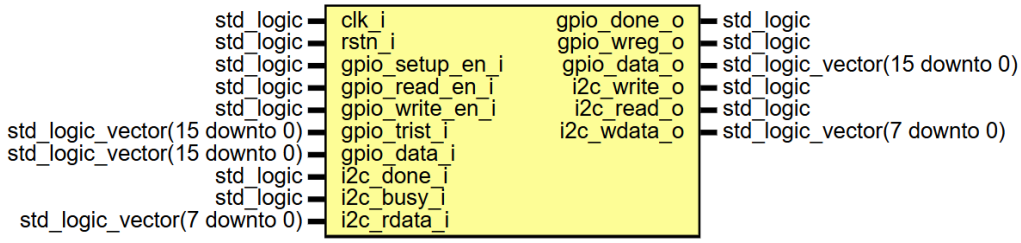



*Figure 20. I2C GPIO Block Diagram.*

The device functionality can be summarized in three steps, all implemented in the VHDL block:

a. Configuration: both configuration registers, two 8-bits registers require their values being set on startup of the device, the values in these registers determine if a corresponding pin port is set as an input or output.
b. Write: the I2C write function sends the START condition with the device address and the last bit set to 0 to signal a Write operation. Figure 21 shows the required bus transaction, sourced from [41].

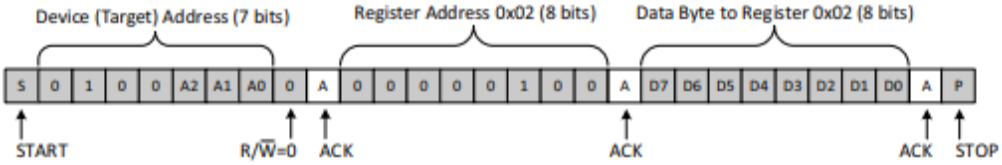



*Figure 21. Example I2C Write procedure for GPIO Bus Expander.*

c. Read: reading from a register requires to send the address of the device with a write instruction, followed by the internal register address to read; after acknowledging the transaction, the data is transmitted until all data exchange is completed. Figure 22 shows the example operation for a Read, sourced from the datasheet of the device [41].

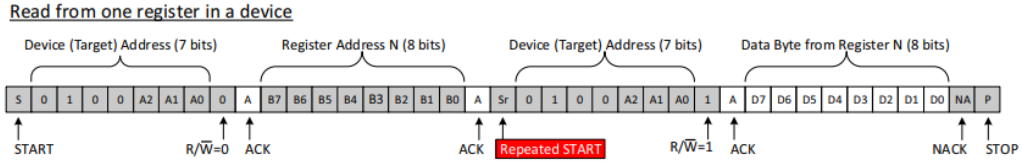



*Figure 22. Example I2C Read procedure for GPIO Bus Expander.*

The internal state machine in the I2C GPIO block implements all these three procedures and repeats them in the I2C Controller by using the *gpio_setup_en_i, gpio_write_en_i,* and *gpio_read_en_i* input ports, which are the setup enable, write enable and read enable for the GPIO peripheral. For the full illustration of the state machine of this block, refer to Appendix E – VHDL Code and State Machines.

**I2C INA – i2c_ina.vhd**

This block implements the state machine that carries the read/write enable operations required for the configuration and functioning of the peripheral. The required programming steps for the device are implemented following the requirements listed in the datasheet [42].

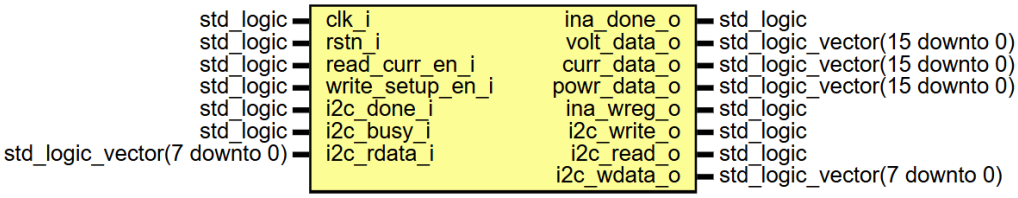



*Figure 23. I2C INA Current Monitor Block Diagram.*

Similar to other peripherals, the device operations are broken down in the following:

a. Configuration: this operation is required on device startup to overwrite the default values on the device. It is required to set the desired 16-bit Calibration Register at address 0x00 as shown in Figure 24, sourced from the device datasheet [42].

**Figure 19. Configuration Register**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RST | — | BRNG | PG1 | PG0 | BADC 4 | BADC 3 | BADC 2 | BADC 1 | SADC 4 | SADC 3 | SADC 2 | SADC 1 | MODE 3 | MODE 2 | MODE 1 |
| R/W-0 | R/W-0 | R/W-1 | R/W-1 | R/W-1 | R/W-0 | R/W-0 | R/W-1 | R/W-1 | R/W-0 | R/W-0 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

*Figure 24. INA Configuration Register. Adapted from [42]*

From MSB to LSB the Configuration Register gives values for: 15 - Reset bit or System-Level Reset, 13 – Bus Voltage Range, 12-11 – PGA gain and range, 10-7 Bus ADC Resolution, 6-3 Shunt ADC resolution, and bits 2-0 give the Operating Mode of the device.

b. Calibration: the Calibration register is necessary to be able to use the Current and Power registers. The value for the Calibration register needs to be calculated using Equation 1, where Current LSB = 1 mA/bit, and $R_{SHUNT}$ = 0.02Ω.

$$Cal = trunc\left(\frac{0.04096}{Current_{LSB} \times R_{SHUNT}}\right)$$

*Equation 1. Calibration Register value.*

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FS15 | FS14 | FS13 | FS12 | FS11 | FS10 | FS9 | FS8 | FS7 | FS6 | FS5 | FS4 | FS3 | FS2 | FS1 | FS0 |
| R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

(1) FS0 is a *void* bit and will always be 0. It is not possible to write a 1 to FS0. CALIBRATION is the value stored in FS15:FS1.

*Figure 25. Calibration Register. Adapted from [42]*

c. Read: When reading from the INA219, the last value stored in the register pointer by a write operation determines which register is read during a read operation. To change the register pointer for a read operation, a new value must be written to the register pointer. If repeated reads from the same register are desired, it is not necessary to continually send the register pointer bytes; the INA219 retains the register pointer value until it is changed by the next write

operation. For our use case, the Bus Voltage, Power and Current register need to be read, at internal addresses 0x02, 0x03, and 0x04

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BD12 | BD11 | BD10 | BD9 | BD8 | BD7 | BD6 | BD5 | BD4 | BD3 | BD2 | BD1 | BD0 | — | CNVR | OVF |

*Figure 26. Bus Voltage Register. Adapted from [42]*

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| PD15 | PD14 | PD13 | PD12 | PD11 | PD10 | PD9 | PD8 | PD7 | PD6 | PD5 | PD4 | PD3 | PD2 | PD1 | PD0 |
| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

*Figure 27. Power Register. Adapted from [42]*

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CSIGN | CD14 | CD13 | CD12 | CD11 | CD10 | CD9 | CD8 | CD7 | CD6 | CD5 | CD4 | CD3 | CD2 | CD1 | CD0 |
| R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 | R-0 |

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

*Figure 28. Current Register. Adapted from [42]*

I2C TMP100 – i2c_tmp100.vhd

The TMP100 block implements the necessary FSM and selects the appropriate register-level read/write operations required for the configuration and functioning of the peripheral according to the device datasheet [43].

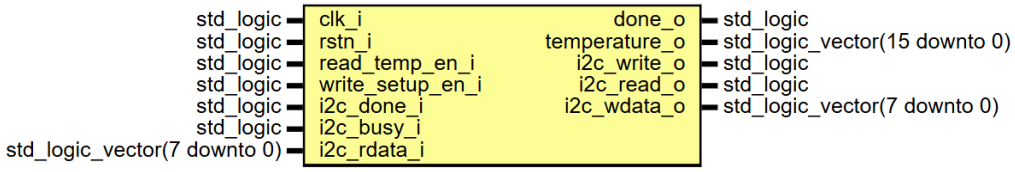



*Figure 29. I2C TMP100 Block Diagram.*

Similar to other I2C devices on the system, the procedure needed are as follows:

a. Configuration: the Configuration Register shown in Figure 30 is a read-and-write register. The correct value must be written on this register for the device to function properly. From MSB to LSB, the configuration bits are as follows: D7 – One Shot mode, single temperature conversion, D6-D5 converter resolution between 9 to 12 bits, D4-D3 Fault Queue, triggered when the $T_{LOW}$ or $T_{HIGH}$ register values are exceeded, D2 polarity of the Alert output, D1 Comparator or Interrupt mode, D0 Shutdown for power saving.

**Table 8. Configuration Register Format**

| BYTE | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| 1 | OS/ALERT | R1 | R0 | F1 | F0 | POL | TM | SD |

*Figure 30. TMP100 Configuration Register.*

b. Read: the temperature conversion is stored in 12-bit representation, divided into two-byte registers, for the read operation byte 1 is read first, followed by byte 2. To access the Temperature Register, the address 0x00 must be written to the Pointer Register with a write operation. After this write operation, the contents of the Temperature registers are written on the bus.

**Table 6. Byte 1 of the Temperature Register**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| T11 | T10 | T9 | T8 | T7 | T6 | T5 | T4 |

**Table 7. Byte 2 of the Temperature Register**

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| T3 | T2 | T1 | T0 | 0 | 0 | 0 | 0 |

*Figure 31. TMP100 Temperature Register.*

### I2C Controller – i2c_controller.vhd

The I2C controller instantiates the I2C block, the I2C GPIO, I2C INA and I2C TMP100 blocks, and acts as the central point for communicating with the peripherals in the Carrier board. The internal state machine continuously writes the initial values to the configuration registers of all three devices: GPIO Bus Expander, INA Current Monitor, and TMP100 temperature sensor. After initial configurations are completed, it gets the read data from the devices, this process is repeated in an FSM loop that also controls the write enable signal to the peripheral instances. A detailed FSM is shown on Appendix E – VHDL Code and State Machines - Figure 102.

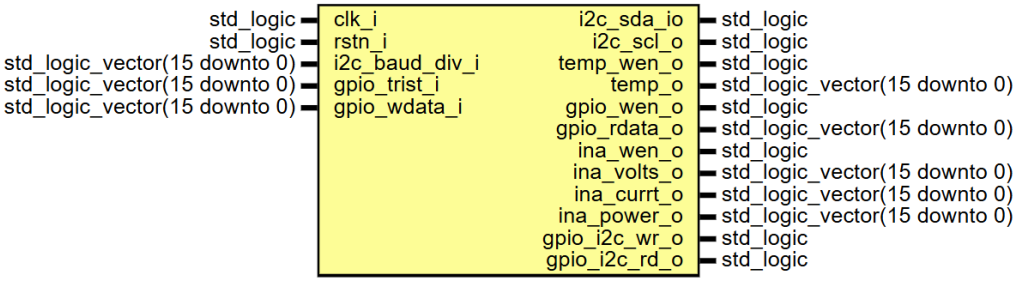



*Figure 32. I2C Controller Block Diagram.*

### Timestamp – timestamp.vhd

The timestamp block is responsible for keeping the internal time in the device and implements a millisecond timestamp that is used by the rest of the system. It uses a simple tick counter, that increases the millisecond count every 49999 ticks at 50 MHz

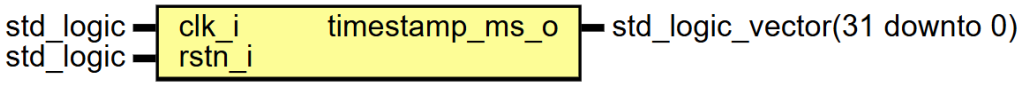



*Figure 33. Timestamp Block Diagram.*

## 2.4 Experimental Design and Procedure

After the PCB was designed, soldered and tested, the next step was to simulate the VHDL blocks described in the previous section. Simulations using Vivado 2018.2, and Libero 2024 were carried out, and the simulation signals were compared to the predefined behavior.

After all simulations were completed and verified, a hardware verification of each block or group of related blocks was carried out. A bitstream for the FPGA was generated and the design was tested on hardware using the SMF2000 FPGA and the Monitoring and Carrier board.

After a successful hardware validation was conducted in the laboratory with a full test setup, a neutron irradiation test was conducted in the ISIS Muon Sourced in the ChipIR beam during July 2024, this further validated the usefulness of the platform. The experimental procedure for testing the SoC boards follows the standard testing under radiation suggested in other experiments [44].

# 3 Results

## 3.1 Overview of Results

The following section presents the results obtained from the Monitoring board, including the visualization of the simulation using Vivado and the obtained logs. The simulations performed serve as unitary and integration test of the system, since the simulations were first carried out block by block, and then integrated into an encompassing system-level simulation. This simulation approach simplified the design, simulation and hardware validation of the overall design. A brief discussion of the results is presented, along with any issues encountered during board testing.

## 3.2 Detailed Presentation of Data

**UART**

A successful UART simulation was conducted and verified at the signal simulation level. Both receive and transmit operations are completed successfully. This block is then integrated into the different blocks that require UART, specifically: Redirect Buffer, Current Report, Command Interface.

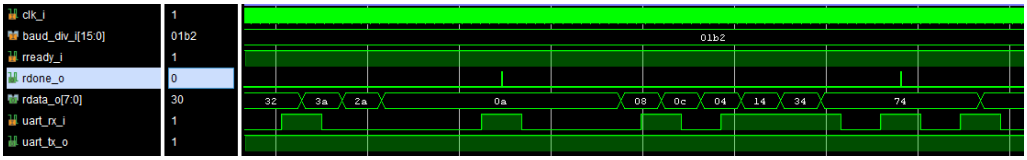



*Figure 34. UART Receive simulation.*

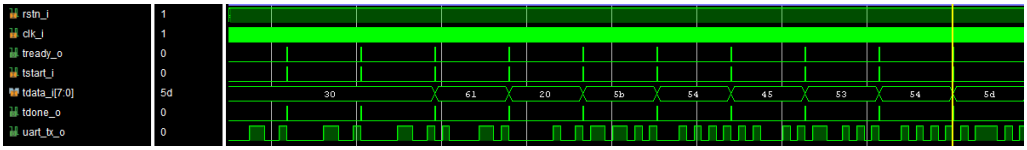



*Figure 35. UART Transmit simulation.*

For the verification part of this block, a simple loopback test [45], where the TX and RX pins on the Carrier board are connected to each other, and one or multiple characters are sent using a Serial Terminal, the received message is the same message, this proves that the serial communication works. Figure 36 shows the output of a simple loopback test using the Visual Studio code Serial Monitor @115200 bps, 8 data bits, no parity. This prototype was carried out in hardware, using the actual FPGA.

```
---- Opened the serial port COM5 ----
---- Sent utf8 encoded message: "testing\n" ----
testing
```

*Figure 36. Simple loopback test with UART.*

### Redirect Buffer

The Redirect Buffer was implemented successfully, being able to receive data from the SUT with the added timestamp. Figure 37 shows 1. the transmitted UART from the redirect buffer, 2. the received UART from the SUT, 3. the transmitted data with the additional timestamped data.

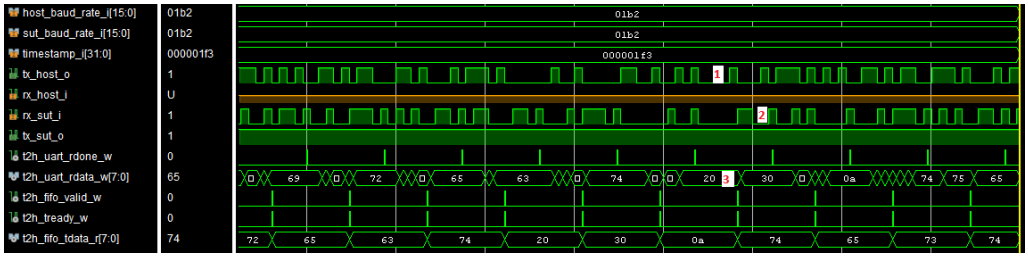



*Figure 37. Redirect Buffer simulation.*

For the functional verification, a simple loopback was implemented in the Carrier board FPGA design, and it was verified that the sent string was returned with a timestamp added. Figure 38 shows the output of the loopback test, where the message sent was "cafe" in hexadecimal, and the return message was the sent message plus the timestamp in milliseconds given in hexadecimal value.

```
---- Opened the serial port COM5 ----
00000020 cafe
00000022 cafe
00000024 cafe
00000027 cafe
0000002a cafe
```

*Figure 38. Redirect Buffer functional verification.*

### Current Report

The Current Report was simulated by 1. setting the *curr_th_i* – current threshold input port to a fixed value, 2. comparing the input current data from the registers *curr_rdata_i* with the threshold value, 3. after the current value is higher than the threshold, an overcurrent is reported using the *overcurrent_o* output port, this last output will be connected to the Overcurrent Handler to turn the device on/off. 4. Finally, all the events get logged in an internal FIFO memory that gets converted to UTF-8 and sent to the UART for reporting. Figure 39 shows the labelled steps described above.

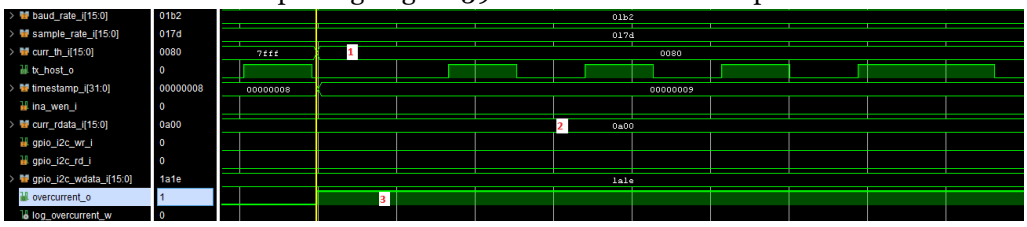



*Figure 39. Current Report simulation.*

### Overcurrent Handler

The Handler takes 1. Waits for overcurrent flag *overcurrent_i* to power-cycle the device, 2. The device is turned on, then off for the duration set on the *on_time_i* and *off_time_i,* which are configured from the Registers, this means that the on-off time for power cycle can be configured before or during the experiment, 3. The *gpio_wdata_o* is used to send the correct value to the GPIO register, to try to cycle the device.

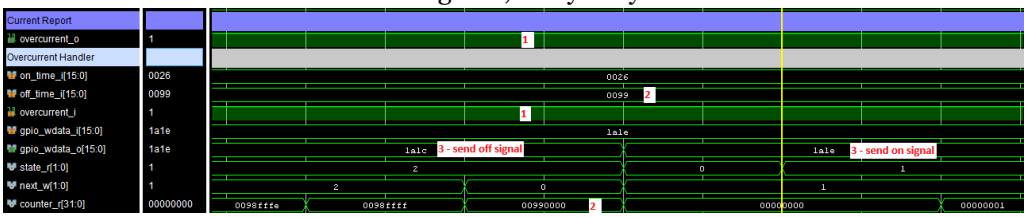



*Figure 40. Overcurrent Handler simulation.*

### Command Interface

This interface is key to the setup operation, as it reads and writes the corresponding registers to power cycle the device using GPIO, read temperature values, and read voltage, current and power from the INA.

The user can interact directly with the command interface by using Channel 4 of the UART and a Serial Monitor with UTF-8 encoding. During the experiment, Python scripts with the proper encoding are used to automatically send the commands to read-and-write to the registers.

Write commands are 40-bits long, with 8-bits to store the *"w"* UTF-8 character, plus 16-bits for the address, and 16-bits for the value to be written. An example command to write *"cafe"* to register address *"0001"* would be *"wcafe0001"*. Figure 41 shows the Commands simulation, by writing data *"cafe"* to address *"0001"*. We also see the correct toggling of the *wr_en_o* port, this is the write enable to the Registers.

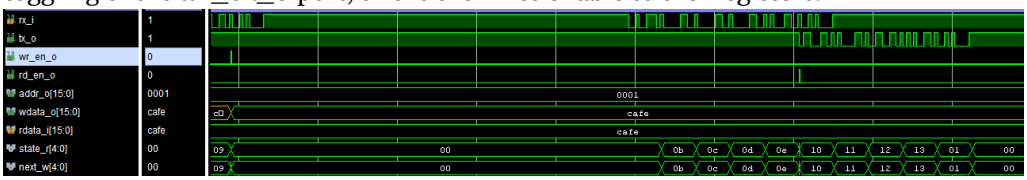



*Figure 41. Command Write simulation.*

Read commands are 24-bits in total: 8-bits to store the *"r"* (read) UTF-8 from the serial, plus 16 bits to decode the address to be read e.g. *"FFFF"* would be an example address, and the full command would be *"rFFFF"*, the equivalent of saying read Registers address "FFFF".

Figure 42 shows the *rd_en_o* signal going high, the read enable signal, the address to be read is *"0001",* and the *read_data_i* is "cafe".

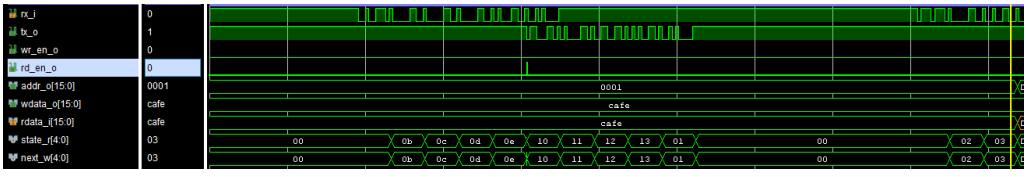



*Figure 42. Command Read simulation.*

The validation on hardware of the Command Interface was done using a Serial Monitor and writing and reading from a series of registers. Figure 43 shows the serial output of reading the board name at address "*0001*" is shown, then a Write operation to register address "*0007*" followed by an immediate read, this is the current threshold register.

```
----- Opened the serial port COM5 -----
r0001
CAFE
w0ABC0007
r0007
0ABC
```

*Figure 43. Validation of Command Interface.*

**Registers**

The Write operation to the Registers is enabled by the *wr_en_i* port, which is connected to the Command interface *wr_en_o* port, in this way the commands can directly control the register writes. Figure 44 shows the Command Interface on the top, signaling the write enable out signal. On the Figure, we can see the *wr_en_i* write enable input port from the Registers being toggled, and the change in the *wdata_i*, the incoming data.

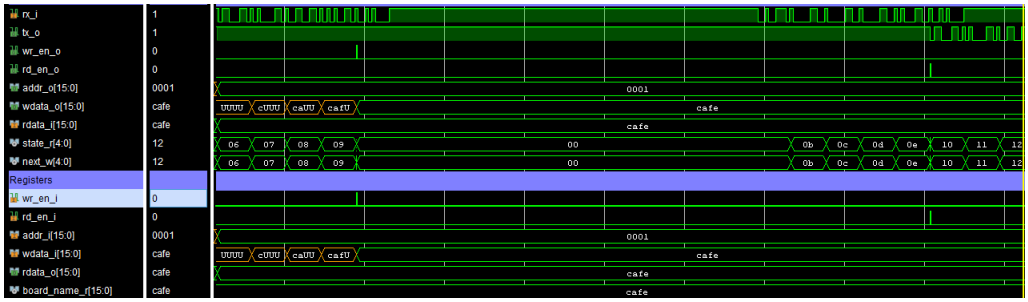



*Figure 44. Register Write simulation.*

The read operation is pretty similar to write, but now the controlling signal is the *rd_en_i* – the read enable port. This is toggled by the Command Controller, and we immediately see the data being outputted in the *rdata_o* port.

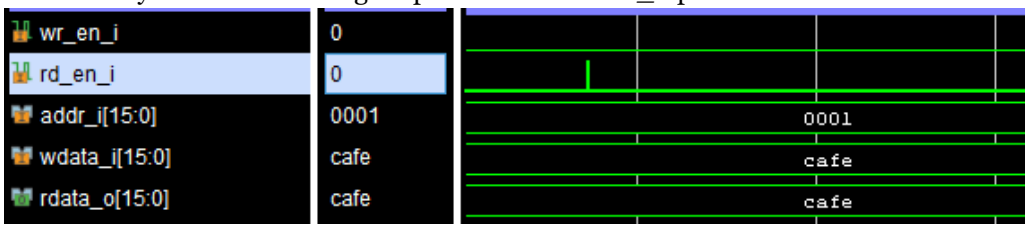



*Figure 45. Register Read simulation.*

The hardware validation for both Registers read-and-write is covered by the test performed for the command interface shown in Figure 43. The Command Interface is tested by writing and reading from the registers, so we can verify both blocks work with a single functional verification.

### FIFO

For the FIFO memory, we can break down the essential operations into two, reading and writing, since the controlling ports for this block are the *write_i* and *read_i*, which enable the writing and reading of the data, respectively. Figure 46 shows the writing operation in 5 annotated steps: 1. Write is enabled via the controlling port, 2. The input data is put on the *data_i* port, 3. The input data at the port is written for the duration of the write enable signal, 4. FIFO *empty_o* status is changed after the first data write, signaling that the memory is no longer empty, 5. As data is inputted in the FIFO, the size of the memory keeps increasing, as well as the next address where the next data is inserted.

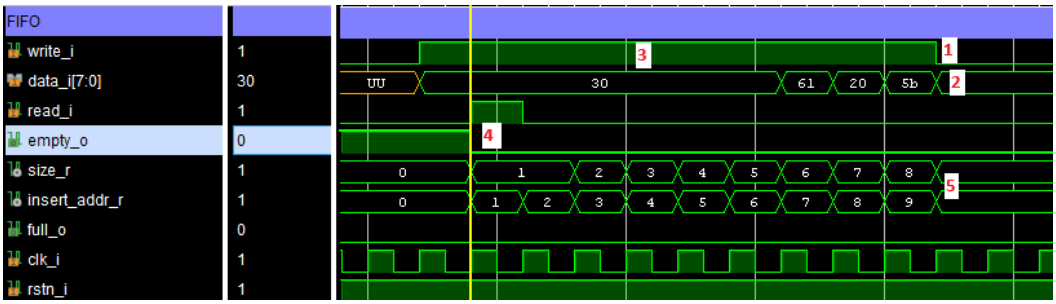



*Figure 46. Writing to FIFO simulation.*

The reading portion of the FIFO is carried out in 5 steps: 1. Port *read_i* is set to 1, enabling the read operation if the FIFO is not empty, this means if *empty_o = 0*, 2. The FIFO internal register takes the value in the corresponding address of *first_address_r,* which is the next value to be read, 3. The output port *data_o* takes the corresponding value. 4. The address pointer *first_address_r* is updated with the new address value *next_first_addr_w*, 5. The *size_r* register gets updated from *new_size_v* with the new size of the FIFO. Figure 47 shows the annotated steps and its corresponding simulated outputs.

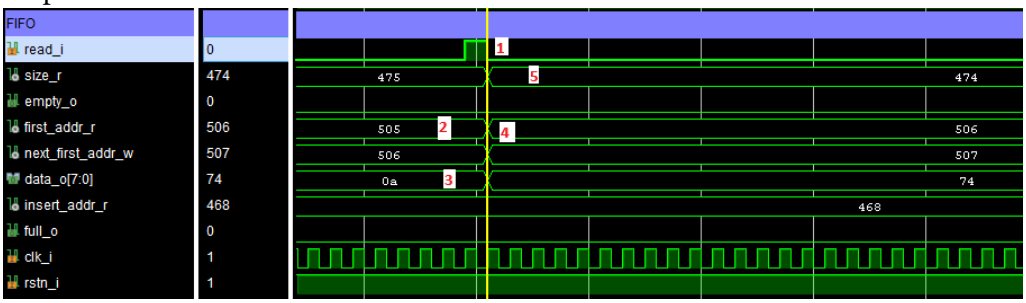



*Figure 47. Reading from FIFO simulation.*

### Hex to UTF-8

Hex to UTF-8 is successfully implemented and simulated, in Figure 48 we can see the input hex codes and its corresponding conversion immediately. The conversions were verified and implemented by using a UTF-8 encoding with its corresponding Unicode code point from the table in [46].

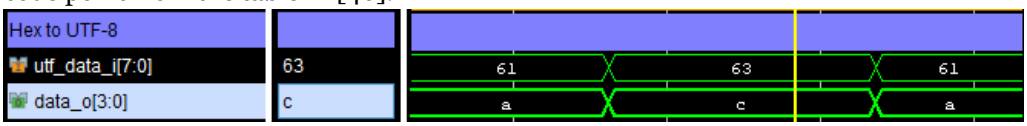



*Figure 48. Hex to UTF-8 simulation.*

### UTF-8 to Hex

UTF-8 to Hexadecimal is successfully implemented and simulated, in Figure 49 we can see the input UTF-8 and its corresponding conversion to hexadecimal immediately in the output port. The conversions were verified and implemented by using a UTF-8 encoding with its corresponding Unicode code point from the table in [46].

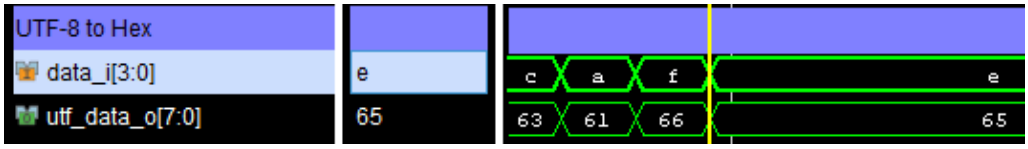



*Figure 49. UTF-8 to Hex simulation.*

### I2C GPIO

For the GPIO we can see from Figure 50 that the enables for setup, read, and write are being correctly simulated, lets see in detail what is being written on each state.

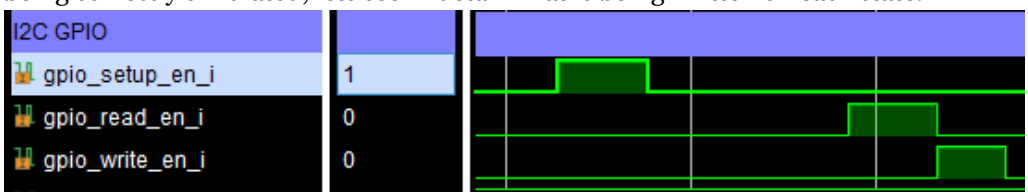



*Figure 50. GPIO FSM enables simulation.*

For the Configure operation, we see that we are sending data to the I2C port *i2c_wdata_o*, which corresponds to the following:
*06* – the address of configuration register 1.
*FF* – the value of the configuration value for register 1.
*07* – the address of the configuration register 2.
*FF* – the value of the configuration value for register 2
With this we are setting all GPIO ports to inputs. This is shown in Figure 51.

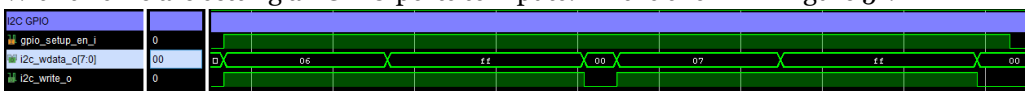



*Figure 51. GPIO Configure simulation.*

For the write operation, we see that we are sending data to the I2C port *i2c_wdata_o*, which corresponds to the following:
*02* – the address of configuration register 1.
*1E* – the LSB value of the output register.
*1A* – the MSB value of the for the output register
With this, we are setting all GPIO outputs to 1A1E. This is shown in Figure 52.

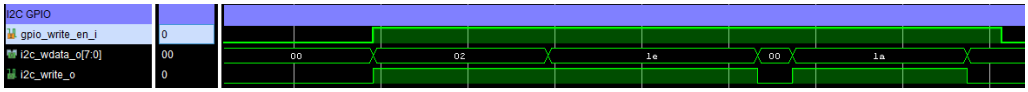



*Figure 52. GPIO Write simulation.*

For the read operation, we see that we are receiving data to the GPIO port *gpio_data_i,* which corresponds to the following:
*00* – write the address of the output register.
*1A* – the MSB value of the output register.
*1E* – the LSB value of the for the output register
With this, we are reading both registers from the GPIO outputs . This is shown in Figure 53.

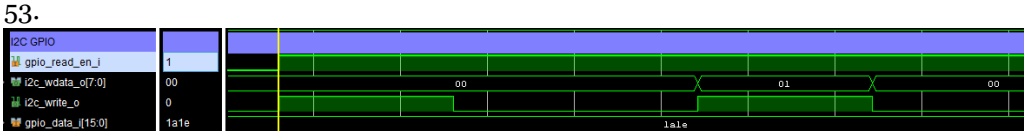



*Figure 53. GPIO Read simulation.*

Finally, the Register map implemented for the GPIO is shown in Table 7. Register map for GPIO. Where the P00 to P17 is the bit number for the register is indicated, its functional use and if the bit is configured as an input or output.

| **Bit** | **P00** | **P01** | **P02** | **P03** | **P04** | **P05** | **P06** | **P07** |
|---|---|---|---|---|---|---|---|---|
| **Function** | SOM_JTAG_SEL | SOM_PWR_EN | SOM_NOSEQ | SOM_PGOOD | SOM_BOOTMODE | SOM_Nrst | SOM_GPIO_0 | SOM_GPIO_1 |
| **I/O** | OUT | OUT | OUT | IN | OUT | OUT | IN/OUT | IN/OUT |
| **Bit** | **P10** | **P11** | **P12** | **P13** | **P14** | **P15** | **P16** | **P17** |
| **Function** | SOM_GPIO_2 | SETUP_PWR_EN | SETUP_WDT_WDO | SETUP_ID_LSB | SETUP_ID_MSB | SETUP_GPIO_TEST | PCIE_GPIO | Not used |
| **I/O** | IN/OUT | OUT | IN | IN | IN | IN/OUT | IN/OUT | NC |

*Table 7. Register map for GPIO.*

**I2C INA**
For the INA we can see from Figure 54 that the enables for Configuration, Calibration, Read, and Write are being correctly simulated, let's see in detail what is being written on each state.
For the Configuration operation, we see that we have enabled the controlling signal *write_setup_en_i*. We send data to the I2C port *i2c_wdata_o,* which corresponds to the following:
*00* – the address of configuration register 1.
*39* – the MSB value of the configuration register.
*9F* – the LSB value of the configuration register.
With this, we are setting the Configuration register to the value "*399F*". This is shown in Figure 54

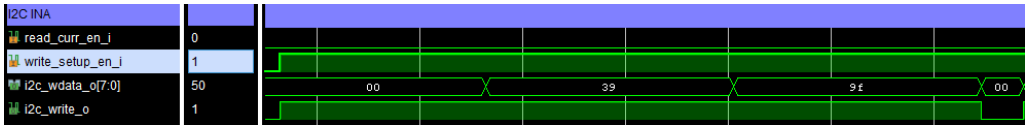



*Figure 54. INA Configuration simulation.*

The Calibration is performed after the Configuration, using the same the controlling signal *write_setup_en_i*. We send data to the I2C port *i2c_wdata_o,* which corresponds to the following:
*05* – the address of the Calibration register.
*50* – the MSB value of the Calibration register.
*00* – the LSB value of the Calibration register.
With this, we are setting the Calibration register to the value "*5000*". This is shown in Figure 55.

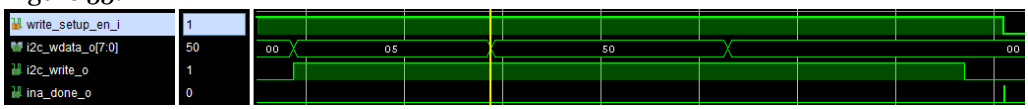



*Figure 55. INA Calibration simulation.*

The Read operation makes use of the controlling signal *read_curr_en_i*. We send data to the I2C port *i2c_wdata_o,* which corresponds to the following:
*02* – the address of the Bus Voltage register.
*00* – the *i2c_read_o* is toggled, to read the incoming data. Voltage data is stored in the internal register and sent to the *volt_data_o* port.
*04* – the address of the Current register.
*00* – the *i2c_read_o* is toggled, to read the incoming current data. Current data is stored in the internal register and sent to the *curr_data_o* port.
*03* – the address of the Power register.
*00* – the *i2c_read_o* is toggled, to read the incoming Power data. Power data is stored in the internal register and sent to the *powr_data_o* port.
We have read the three data registers from the INA peripheral. This simulation is shown in Figure 56.

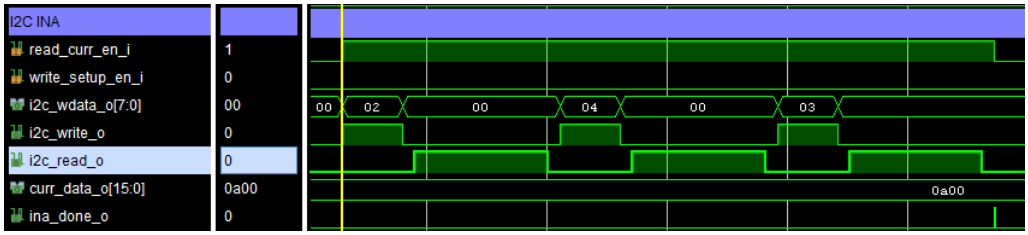



*Figure 56. INA Read simulation.*

**I2C TMP100**

For the TMP100 we can see from that, *write_setup_en_i* and *read_temp_en_i* are being correctly toggled these are the Configuration, and read Read enable signals, let's see in detail what is being written on each state.
For the Configuration operation we see that we have enabled the controlling signal *write_setup_en_i*. We send data to the I2C port *i2c_wdata_o* which corresponds to the following:
*00* – the address of configuration register 1.
*39* – the MSB value of the configuration register.
*9F* – the LSB value of the configuration register.
With this we are setting the Configuration register to the value "*399F*". This is shown in

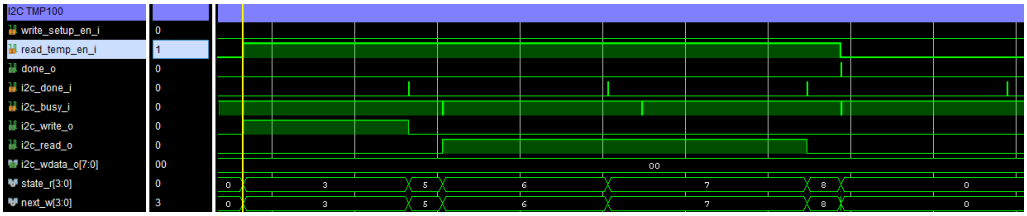



*Figure 57. TMP100 simulation.*

For the Configuration operation, we see that we have enabled the controlling signal *write_setup_en_i*. We send data to the I2C port *i2c_wdata_o,* which corresponds to the following:
*01* – the address of the configuration register.
*60* – the 8-bit values of the configuration register.
With this, we are setting the Configuration register to the value "*60*". This is shown in Figure 58.

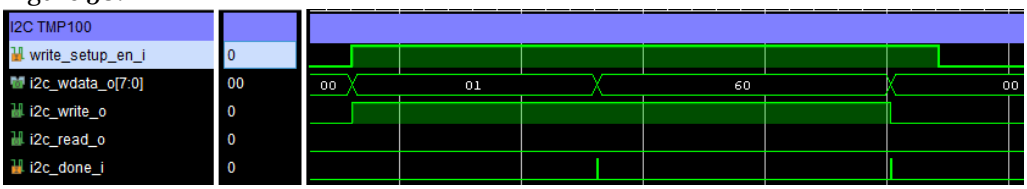



*Figure 58. TMP100 Configuration simulation.*

For the Read operation, we see that we have enabled the controlling signal *read_temp_en_i*. We send data to the I2C port *i2c_wdata_o,* which corresponds to the following:
*00* – the address of the Temperature register. After this value has been written, we disable the *i2c_write_o* signal, *i2c_done_i* signals that the I2C communication is done, and we enable *i2c_read_o* to read the value of the Temperature register.
With this, we are reading the Temperature register value at the internal address "*00*". This is shown in Figure 59.

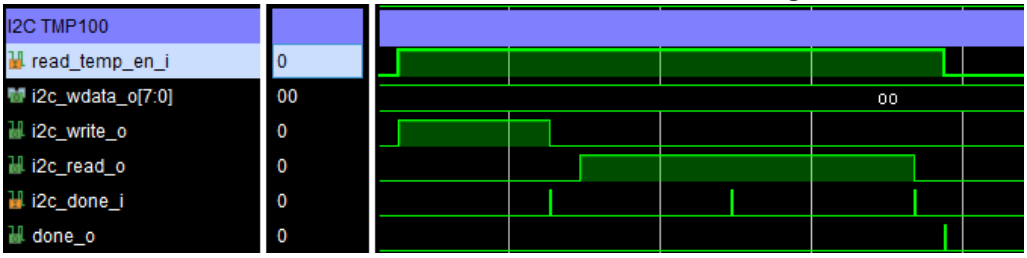



*Figure 59. TMP100 Read simulation.*

### I2C Controller

For the I2C Controller, we can verify that the enable cycles of all three peripherals are being toggled, and that the state machine is functioning properly. We can further verify it with the I2C test, in which we are able to communicate correctly to all the peripherals. The toggling of the enabling signals is shown in Figure 60.

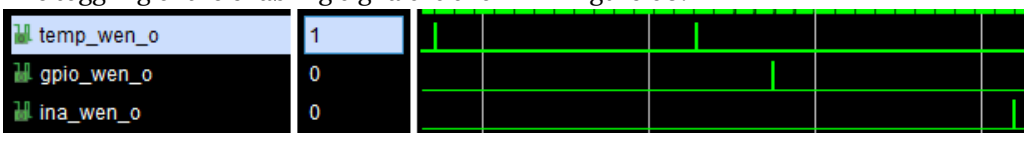



*Figure 60. I2C Controller simulation.*

### I2C

The I2C controller and the I2C implementation work together to have a proper addressing, reading, and writing of the I2C peripherals in the Carrier board. At the highest level of the hierarchy, this can be verified by the correct toggling of the SDA and SCL signals, following the I2C protocol [40]. Figure 61 shows the different states for the I2C using the *state_r* register, we can see the state going from a -> b -> 0 -> 1 -> 2 -> 3 -> f, these states signify the change from STOP -> WAIT -> START -> ADDR -> AACK ->ACKN, respectively. Additionally, we can verify the simulation with the *i2c_sda_o* and *i2c_scl_o* ports, which will eventually be routed to the actual pins on the device.

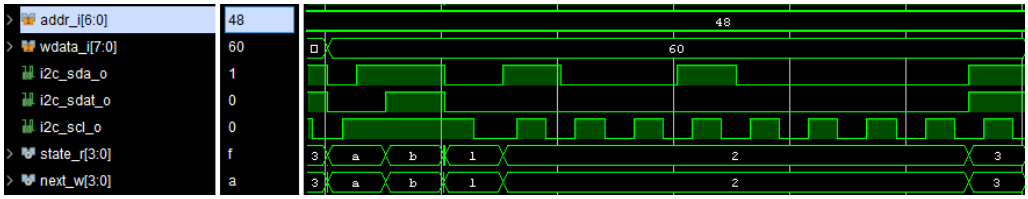



*Figure 61. I2C simulation.*

### Timestamp

The timestamp block is successfully simulated using the tick counter, Figure 62 shows the simulation results, it can be seen that after the tick counter reaches the predefined value of *oxC34F* the millisecond register, *millisecond_r,* is increased, this continues updating the millisecond value and outputs the value on the *timestamp_ms_o* output port. We can see that the update of the port, value, and register happens around the 1 000 000 ns mark, which is equivalent to 1 millisecond.

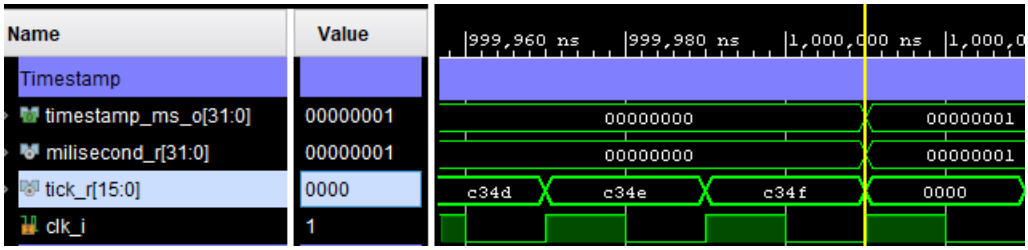



*Figure 62. Timestamp simulation result.*

### System Test

After development of the HDL blocks, and the functional verification of individual blocks, the SMF2000 FPGA was flashed with the complete design and tested on the full setup in the laboratory. The full test setup is shown in Figure 63.

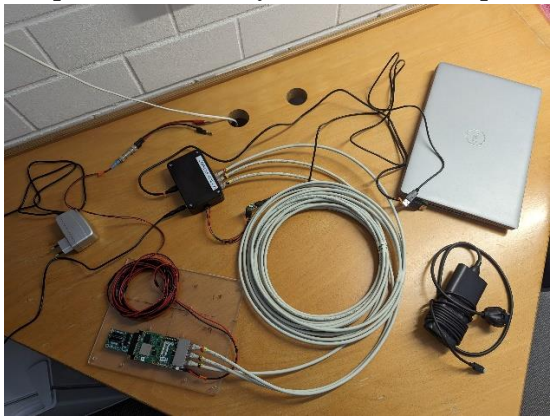


*Figure 63. System test setup.*

The system was left running over a 24-hour period to check for stability and identify any potential setup errors. Additionally, the hardware setup was integrated with the scripting software to monitor, log and control the experiment from a computer terminal, this is based on the series of Python scripts and Make files made by Mattos A. *et al.* in [23] which make possible to interact with the devices, to send and interact using commands to the device, monitor the devices, and continuously log the current.

The scripts generate .txt files as logs, and a Make file is used to simplify the creation of the necessary folders, create a Python virtual environment with the necessary packages, and centralize the initialization of the experiment from a single file. An example current monitor terminal is shown in Figure 64, where we can see where the received data is being logged to, which FTDI we are monitoring, and at which serial settings and the Current information marked with the *[CURR]* tag.

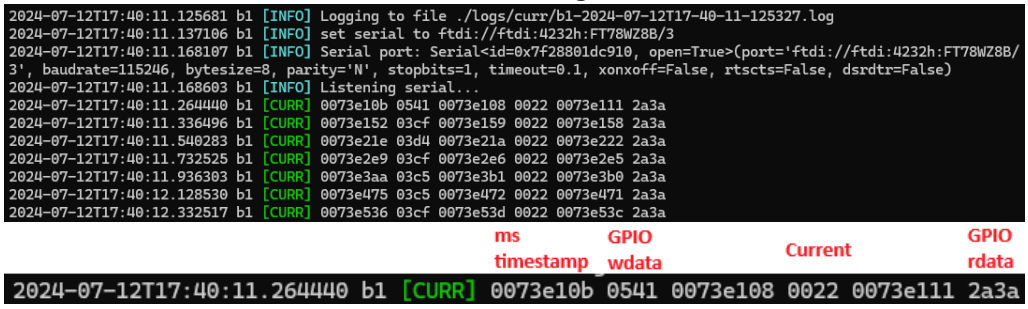



*Figure 64. Current terminal output (top) and annotated example log (bottom).*

A SEL protection test was carried out, with modeled faults that intent to cover the different current levels of the SUT, these states are: device OFF, device ON, device BOOT, Nominal, Nominal (with SEFI), Watchdog timeout, and SEL. *Mattos A. et al.* published the details of this test in [23], Figure 65 was prepared for the mentioned paper and it's presented as the expected response of the SUT under test.

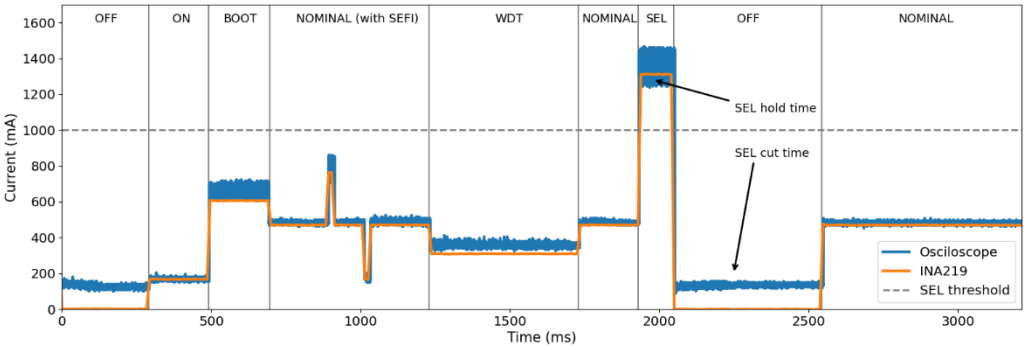



*Figure 65. Emulated current behavior of a SoM during various states. Adapted from [23].*

The test setup was validated in the lab. Next all the necessary hardware was prepared for a heavy-ion irradiation campaign in June 2024 using uranium $U^{28+}$ in the GSI Helmholtz Centre for Heavy Ion Research, unfortunately there was a problem with the beam equipment and the test could not be carried out under irradiation, nonetheless the test setup was validated with the facilities' patch panel and interfaces, the communication interfaces including UART and I2C were tested, and the current monitoring was carried out. Figure 66 shows the test setup used with one of the Carrier boards connected.

For the JTAG validation, the board could only be reprogrammed up to 50 meters using the Vivado tools. Using the Libero programmer only 30 meters of JTAG programming distance could be achieved, this is consistent with the tests carried out in the laboratory. The main hypothesis is that the timing requirements for each of the programming tools are different and that more testing or a better solution is needed when the user must interchange between the different FPGA vendors using the proposed instrumentation.

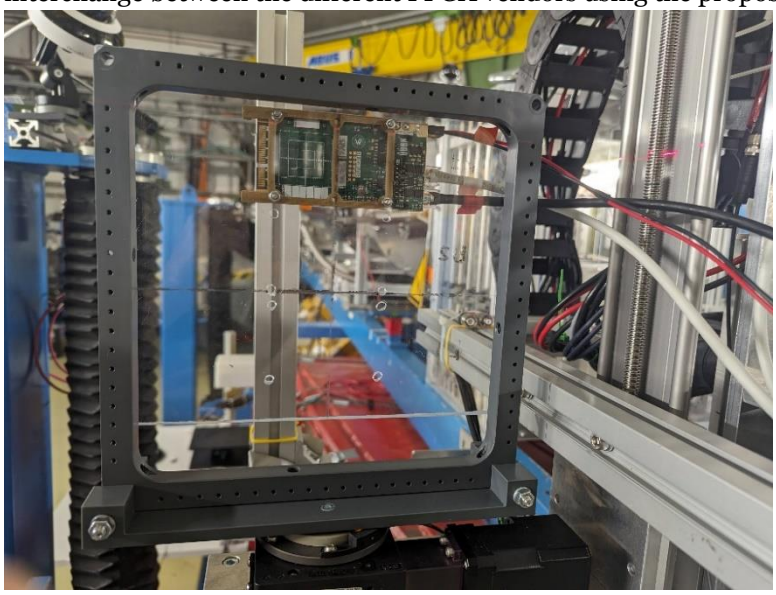


*Figure 66. GSI Test setup.*

A second experiment was conducted in July 2024 at the ISIS Neutron and Muon Source, in the ChipIR beam under an atmospheric neutron beam, for this experiment, the full characterization of the Carrier board was completed, all the details regarding this test will be published in future papers that will contain the radiation response of a Polarfire MPSoC and a custom hardened processor developed in the group [30]. The test setup used for this experiment is shown in Figure 67 and Figure 68.

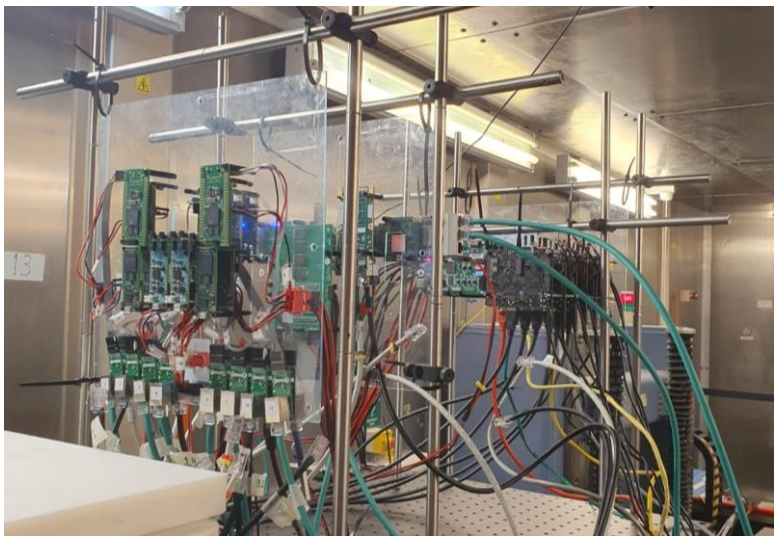


*Figure 67. ChipIR Test setup.*

*Figure 68. ChipIR Test setup, sideview.*

In addition to the SoM irradiation, the Carrier board transceivers were also irradiated directly, to measure the response of the system and prove a worst-case scenario. Using a UART loopback test, no errors were detected on the UART communications, which proves the robustness of the RS-485 transceivers. The I2C over CAN did not present any errors either. During this irradiation test, the SoM was replaced by a fixed resistor to simulate power consumption. Some register errors on the GPIO input port were reported. The full details of the irradiation test are reported in [23].

Finally, an analysis of a classified SEFI during the experiment is shown in Figure 69. It shows the irradiation of a Microchip Polarfire SoC. During this evaluation, we can observe a failure of the device at 327 seconds, followed by a failure to recover, and an attempt to restart the device by the watchdog timer flag. The device failed to recover using the watchdog, until a beam glitch occurred (the marked read area on the figure), this indicates that the device might have some SEE detection that locks up the initialization of the device when errors are detected. At timestamp 2600 seconds, we can see that the device is finally able to recover.

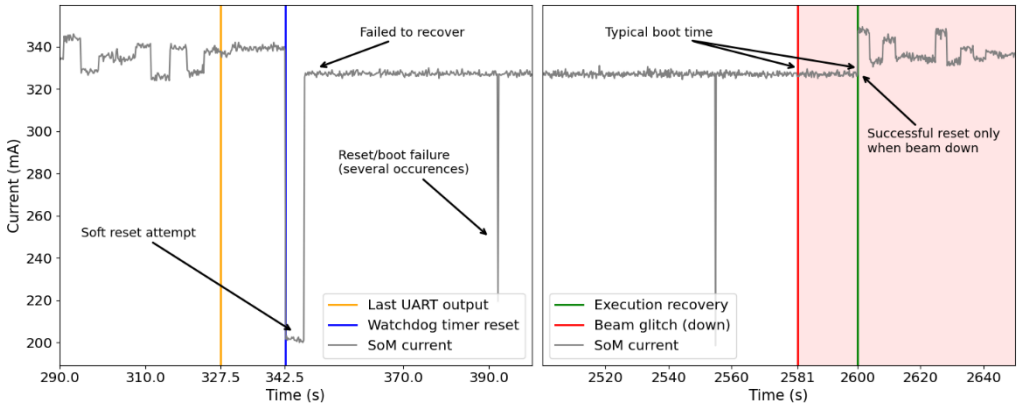



*Figure 69. Investigation of SEFI events, including a watchdog timer reset using the current and fault events in superposition with enhanced synchronization. Adapted from [23].*

We can also see from the timing diagram that the timing response of the system with the setup tested in the experiment, with around 50 meters of cabling distance, is less than 1 second, close to around 100 milliseconds. With this metric we show that the initial requirement of having a fast response time is fulfilled.

# 4 Conclusion

This work presents the full design of an improved instrumentation system for SoM and FPGA radiation testing with enhanced observability. After initial validations in a laboratory setting, we have demonstrated the full setup in a real experiment environment, overcoming the limitations of current methodologies found in the literature, by providing a common platform for multiple vendors of SoC.

The results obtained during demonstrated the usefulness of the setup, with increased synchronization and observability, it was proven to be reliable, and tolerant to errors, most importantly that we can reduce the test operator interventions and errors by automating the device recovery process after a critical error. During the experiment, we tested a Polarfire SoC with great success, providing better understanding of the effects on the device at a system level. Additionally, we were able to correlate observed parameters such as current and communication link, to device effects. The platform will serve for multiple experiments that will support the group's efforts in the future.

Currently, some limitations with the platform exist, such as the reprogramming capabilities using JTAG, and more development time is needed to come up with a better solution that will allow full or partial reconfiguration of a running device. This would allow for more complex testing scenarios that would enhance the testing capabilities for SoM devices.

Overall, the proposed requirements were fulfilled. The required response time, cabled wire distance, peripheral devices sensing and control, including temperature, GPIO, and current were achieved. These measurements can be timestamped by the controlling hardware, allowing for the correlation of effects. By using standard communication links such as serial UART, we can monitor the system's response, and to automatically respond to events without the need for user interaction.

The platform developed is adaptable in its control hardware, due to the reprogrammable nature of the FPGA, and it can support additional devices and peripherals in the future. The project is intended to be open source, this would encourage other users to implement their own testing with the system, would foster collaboration and innovation, by allowing researchers to come up with their own solutions and contributing back to the project. The cost of the platform is relatively low compared to existing solutions, and has the potential to reduce platform selection, test development, and radiation testing.

# 5 Future Work

This work opens the doors for even more granular but simplified radiation analysis, by automating setup, experiment execution, and proposing a unified environment for logs, data, and analysis.

In the future, the platform can accommodate custom devices using the open-source SoM specification, connecting additional devices to the Carrier board using the PCIe connector, which allows the use of secondary devices such as SRAMs. These memories can be utilized by the SUT for additional functionality, but also to observe SEU errors and correlate the fluence, with the high precision synchronization. These has been proven in other monitoring systems such as the radiation monitor presented in [20]. Latch-up free memories can be utilized, selected from COTS test data, these memories typically work in a range of voltages that can be varied, this voltage variation can be utilized as a variable parameter for different cross sections using the same memory.

Additional work is needed to overcome the programming limitations via the JTAG interface during the experiment. Other interfaces could be explored, such as using optical transceivers instead of wired differential transceivers, this would guarantee signal integrity even under the worst radiation conditions and long links. Another option would be to explore independent or vendor tools for custom programming jobs, specifically for the Microchip devices, which present the major limitations when programming via JTAG. By exploring reduced clock frequency programming, a slower clock can be utilized to program the device in the worst-case timing scenario, while still achieving the required timing closure.

Further validation under different radiation sources is required. The setup will be tested under high-energy protons at the Proton Irradiation Facility at the Center for Neutron and Muon Sciences in September 2024 and validating the test setup under different beam conditions.

Improvements such as additional temperature and voltage control for worst-case testing scenarios would be an asset for future iterations of the platform. The improvement of the device temperature monitoring to a more appropriate peripheral is needed. More testing is needed in scaling the instrumentation for larger systems or more complex designs, such as heterogeneous systems.

Finally, more research and proposals are needed regarding the guidelines for testing complex devices, FPGAs, memories, and heterogeneous systems, such as the standardization of observability metrics. An open observability framework would encourage collaborative best practices and metrics that will eventually lead to better testing guidelines and outcomes.

# References


[1] "Organigram of the University of Montpellier," 6 June 2024. [Online]. Available: https://www.ies.umontpellier.fr/wp-content/uploads/2024/06/organigramme_ies_20240606.pdf. [Accessed August 2024].

[2] University of Montpellier, "RADIAC - Radiation and Componets," 2024. [Online]. Available: https://www.ies.umontpellier.fr/la-recherche-et-linnovation/les-equipes-de-recherche/radiac/. [Accessed August 2024].

[3] R. G. A. e. al., ""Heavy Ion Energy Deposition and SEE Intercomparison Within the RADNEXT Irradiation Facility Network,"," *IEEE Transactions on Nuclear Science,* vol. 70, no. doi: 10.1109/TNS.2023.3260309, pp. 1596-1605, 2023.

[4] D.-J. L. a. C. .. -N. J. L. Chin-Lung Chuang, "A snapshot method to provide full visibility for functional debugging using FPGA," in *13th Asian Test Symposium*, Kenting, Taiwan, 2004.

[5] H. Bokil, "COTS Semiconductor Components for the New Space Industry," in *2020 4th IEEE Electron Devices Technology & Manufacturing Conference (EDTM)*, Penang, Malaysia, 2020.

[6] F. S. P.-X. W. Tomasz Rajkowski, "Radiation Qualification by Means of the System-Level Testing: Opportunities and Limitations," *MDPI Electronics,* vol. 378, no. 11, 2022.

[7] E. P. J. S. J. G. a. M. W. W. Stirk, "Comparison of Neutron Radiation Testing," *IEEE Transactions on Nuclear Science,* vol. 70, no. 4, pp. 505-514, 2023.

[8] NASA, "Guideline for Single-Event Effect (SEE)," JPL, Pasadena, 2018.

[9] J. Leray, "Effects of atmospheric neutrons on devices, at sea level and in avionics embedded systems," *Microelectronics Reliability,* vol. 47, no. 9-11, pp. 1827-1835, 2007.

[10] S. A. a. G. G. C. D. Frost, "A new dedicated neutron facility for accelerated SEE testing at the ISIS facility," in *2009 IEEE International Reliability Physics Symposium*, Montreal, 2009 .

[11] P. Goldhagen, "Cosmic-Ray Neutrons on the Ground and in the Atmosphere," *MRS Bulletin,* vol. 28, no. 2, pp. 131-135, 2003.

[12] D. G. Toro, "Temporal Filtering with Soft Error Detection and Correction Technique for Radiation Hardening Based on a C-element and BICS," Université de Bretagne Occidentale, Bretagne, 2014.

[13] N. K. J. K. a. K. S. H. Kobayashi, "Alpha particle and neutron-induced soft error rates and scaling trends in SRAM," pp. 206-211, 2009.

[14] M. B. S. G. A. C. a. C. D. F. C. Cazzaniga, "First Tests of a New Facility for Device-Level, Board-Level and System-Level Neutron Irradiation of Microelectronics," *IEEE Transactions on Emerging Topics in Computing,* vol. 9, no. 1, pp. 104-108, 2021.

[15] Actel, "Effects of Neutrons on Programmable Logic - White Paper," Actel, California, 2002.

[16] G. B. a. J.-M. Palau, "Single particle-induced latchup," *Nuclear Science, IEEE Transactions,* vol. 43, no. 2, p. 522–532, 1996.

[17] H. Quinn, "Challenges in Testing Complex Systems," *IEEE TRANSACTIONS ON NUCLEAR SCIENCE,* vol. 61, no. 2, pp. 766-786, 2014.

[18] V. V. e. al, "Configuration Memory Scrubbing of the Xilinx Zynq-7000 FPGA using a Mixed 2-D Coding Technique," in *2019 19th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Montpellier, 2019.

[19] P. G. R. F. M. B. M. L. D. F. S. J. B. S. D. A. M. A. Scialdone, "CRaTeBo: a high-speed, radiation-tolerant and versatile testing platform for FPGA radiation qualification for high-energy particle accelerator applications," *Journal of Instrumentation,* vol. 19, 2024.

[20] S. Danzeca, "a. The new version of the Radiation Monitor system for the electronics at the CERN: electronic components radiation hardness assurance and sensors qualification," Université de Montpellier, Montpellier, 2015.

[21] I. S. e. al, "Enhancement of System Observability During System-Level Radiation Testing through Total Current Consumption Monitoring," *IEEE Transactions on Nuclear Science,* vol. 71, no. 8, pp. 1948-1955, 2024.

[22] A. Mattos, D. Santos, L. Luza, V. Gupta and L. Dilillo, "Investigation of Single-Event Effects for Space Applications: Instrumentation for In-Depth System Monitoring," *Electronics 2024,* vol. 13, no. 10, p. 1822, 2024.

[23] M. R. A. D. A. S. L. D. André M. P. Mattos, "Open-source, low-cost, and robust instrumentation for single-event effect qualification of system-on-chip," *Journal of Instrumentation,* 2024.

[24] D. R. B. e. al, "Single-Event Latchup in a 7-nm Bulk FinFET Technology," *IEEE Transactions on Nuclear Science,* vol. 68, no. 5, pp. 830-834, , 2021.

[25] R. S. e. al., "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,* vol. 64, no. 8, pp. 2107-2114, 2017.

[26] Microsemi, "Single Event Effects A Comparison of Configuration Upsets and Data Upsets," Microsemi, Aliso Viejo, 2015.

[27] FTDI Chip, "FT4232H-56Q," Future Technology Devices International Limited, 2024. [Online]. Available: https://ftdichip.com/products/ft4232h-56q/. [Accessed August 2024].

[28] F. H. R. e. al., "Development of FPGA-Based Ingest System with Multi-Output Interface for Receiving Low Resolution Remote Sensing Satellite Data," in *2023 IEEE 7th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, Purwokerto, 2023.

[29] K. Ramesh, "High-Precision and Low-Latency FPGA-Based Weather Station," in *2022 International Conference on Computer, Power and Communications (ICCPC)*, Chennai, 2022.

[30] A. M. P. M. D. R. M. a. L. D. D. A. Santos, "Characterization of a Fault-Tolerant RISC-V System-on-Chip for Space Environments," in *2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Juan-Les-Pins, 2023.

[31] J. K. G. -Y. K. B. S. a. H. Y. G. Yoon, "Multiple RS-485 interface management FPGA design for Power micro-metering," in *2019 10th International Conference on Power Electronics and ECCE Asia (ICPE 2019 - ECCE Asia)*, Busan, 2019.

[32] Texas Instruments, "Reference Design for I2C Range Extension: I2C with CAN," Texas Instruments - TI Designs, Dallas, 2019.

[33] Texas Instruments, "LVDS Owner's Manual," Texas Instruments, Dallas, 2008.

[34] KiCad, "About KiCad," KiCad, February 2024. [Online]. Available: https://www.kicad.org/about/kicad/. [Accessed August 2024].

[35] A. D. Rosso, "KiCad software gets the CERN treatment," CERN, February 2015. [Online]. Available: https://home.cern/news/news/computing/kicad-software-gets-cern-treatment. [Accessed August 2024].

[36] Z. Peterson, "Designing a 4 Layer Stackup With 50 Ohm Impedance PCB Traces," Altium, July 2021. [Online]. Available: https://resources.altium.com/p/designing-4-layer-pcb-stackup-50-ohm-impedance. [Accessed August 2024].

[37] Trenz Electronics, "TEM0009-02 FPGA USB-Programmer JTAG," Trenz Electronics, 2024. [Online]. Available: https://shop.trenz-electronic.de/en/TEM0009-02-FPGA-USB-programmer-JTAG-for-development-with-Microchip-FPGAs. [Accessed August 2024].

[38] D. A. S. L. M. L. V. G. T. B. a. L. D. A. M. P. Mattos, "Investigation on Radiation-Induced Latch-Ups in COTS SRAM Memories On-Board PROBA-V," *IEEE Transactions on Nuclear Science,* pp. 1-1, 2024.

[39] Texas Instruments, "KeyStone Architecture - Universal Asynchronous Receiver Transmitter," Texas Instruments, Dallas, 2010.

[40] W. Joseph, "Application Note - A Basic Guide to I2C," Texas Instruments, Dallas, 2022.

[41] Texas Instruments, "TCA9535 Low-Voltage 16-Bit I2C and SMBus Low-Power I/O Expander," May 2022. [Online]. Available: https://www.ti.com/lit/ds/symlink/tca9535.pdf?ts=1723716654478&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTCA9535%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Dasc-null-null-gpn_en-cpc-pf-google-wwe%2526utm_content%253D. [Accessed August 2024].

[42] Texas Instruments, "INA219 Zerø-Drift, Bidirectional Current/Power Monitor With I," December 20215. [Online]. Available: https://www.ti.com/lit/ds/symlink/ina219.pdf?ts=1723730677958&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FINA219%253Fqgpn%253Dina219%2526bm-verify%253DAAQAAAAJ_____0HXJ00BQ7Ulz8JPQ8rSr6jvBh99jveIbxCsJSQKpcZr0Ckno0GmaAsKAEGetAeDgAnROnOd_tPMvojXA. [Accessed August 2024].

[43] Texas Instruments, "TMP10x Temperature Sensor With I," November 2015. [Online]. Available: https://www.ti.com/lit/ds/symlink/tmp100.pdf?ts=1723746154424&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTMP100%252Fpart-details%252FTMP100NA%252F3K%253Fbm-

verify%253DAAQAAAAJ_____wr4hCsFJ3JlzJnI8KhkXTSyI1Y1bpEcQ-GmpULbPJHaI4pxri3clQhmHcWnGu5C. [Accessed August 2024].

[44] D. A. S. e. al., "Neutron Irradiation Testing and Analysis of a Fault-Tolerant RISC-V System-on-Chip," in *2022 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Austin, 2022.

[45] National Instruments, "How to Perform a Serial Loopback Test," National Instruments, Jul 2023. [Online]. Available: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000YFtECAW&l=en-US. [Accessed August 2024].

[46] UTF-8 Chartable, "UTF-8 encoding table and Unicode characters page with code points U+0000 to U+00FF," UTF-8 Chartable, February 2024. [Online]. Available: https://www.utf8-chartable.de/. [Accessed August 2024].

[47] NASA, "Field Programmable Gate Array (FPGA) Single Event Effect," JPL, Pasadena, 2012.

[48] P. S. J. H. A. a. E. A. B. E. L. Petersen, "Calculation of Cosmic-Ray Induced Soft Upsets and Scaling in VLSI Devices," *IEEE Transactions on Nuclear Science,* vol. 29, no. 6, pp. 2055-2063, 1982.

[49] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability,* vol. 5, no. 3, pp. 305-316, 2005.

[50] E. Normand, "Single-event effects in avionics," *IEEE Transactions on Nuclear Science,* vol. 43, no. 2, pp. 461-474, 1996.

[51] P. G. R. F. M. B. M. L. D. F. S. J. B. S. D. A. M. ] A. Scialdone, "CRaTeBo: a high-speed, radiation-tolerant and versatile testing platform for FPGA radiation qualification for high-energy particle accelerator applications," *Journal of Instrumentation,* vol. 19, 2024.

# Appendix

## Appendix A – Full Bill of Materials for System

Full BOM for Monitoring Board

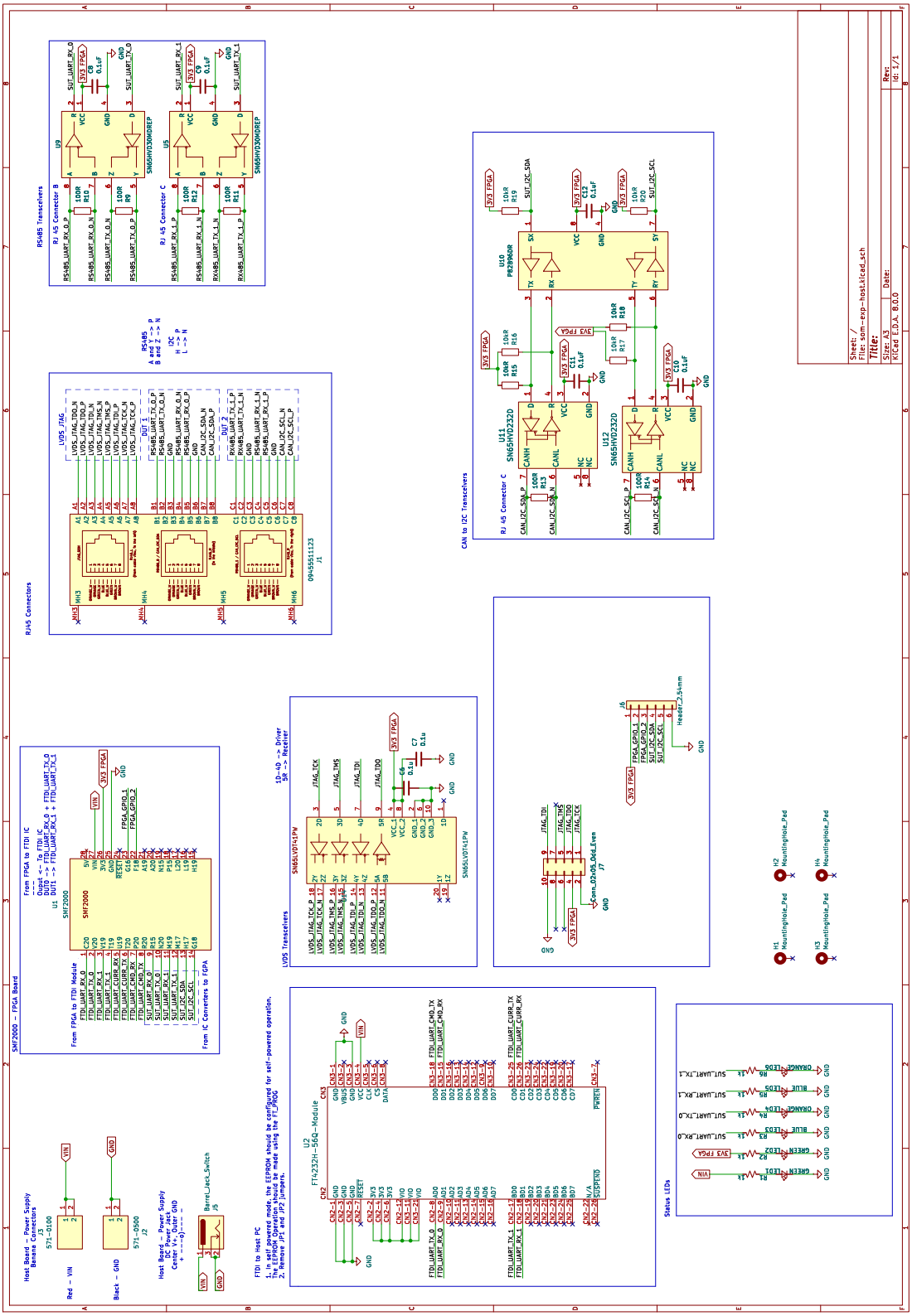
| ID | Component | Qty | Note/Rationale | Price € | Source |
|---|---|---|---|---|---|
| 1 | TEM0001-01A-ABC-2 SMF2000 | 1 | Flash Based FPGA to synchronize and control experiment | 41.00 | Link |
| 2 | FT4232H-56Q MINI MDL | 1 | UART to USB FTDI Chip with 4 channels | 28.64 | Link |
| 3 | SN65LVDT41PW | 1 | LVDS Transceiver for JTAG programming | 7.06 | Link |
| 4 | SN65HVD30MDREP | 2 | RS485 3.3V Transceiver | 3.36 | Link |
| 5 | SN65HVD232D | 2 | CAN 3.3V Transceiver | 1.97 | Link |
| 6 | P82B96DR | 1 | I2C Long Distance Buffer | 2.79 | Link |
| 7 | Bivar SM0805PGC | 2 | Green LED SMD | 0.32 | Link |
| 8 | Bivar SM0805RC | 2 | Red LED SMD | 0.24 | Link |
| 9 | Bivar SM0805UOC | 2 | Orange LED SMD | | Link |
| 10 | YAGEO AC0805FR-071KL | 6 | 1kΩ SMD Resistors | 0.02 | Link |
| 11 | RC0805FR-07100RL | 6 | 100 Ω Resistors | 0.01 | Link |
| 12 | CRCW0805100KFKEA | 6 | 10 KΩ Resistors | 0.01 | Link |
| 13 | Vishay VJ0805Y104KXAMR | 7 | 0.1 uF Capacitors | 0.08 | Link |
| 14 | HARTING 09455511123 | 1 | RJ45 3-port Connector | 4.74 | Link |
| 15 | Deltron 571-0500 | 1 | Banana Connector – Red | 1.83 | Link |
| 16 | Deltron 571-0100 | 1 | Banana Connector - Black | 1.44 | Link |
| 17 | DCJ200-10-A-K1-K | 1 | DC Power Jack | 0.70 | Link |
| 18 | TE Connectivity 5103310-1 | 1 | JTAG connector | 1.43 | Link |
| 19 | Header 2.54mm | 1 | 1 row 6 pin header | NA | NA |

Full BOM for Carrier Board

| ID | Component | Qty | Note/Rationale | Price € | Source |
|---|---|---|---|---|---|
| 3 | SN65LVDT14PW | 1 | LVDS Transceiver for JTAG programming | 6.14 | Link |
| 4 | SN65HVD30MDREP | 2 | RS485 3.3V Transceiver | 3.36 | Link |
| 5 | SN65HVD232D | 2 | CAN 3.3V Transceiver | 1.97 | Link |
| 6 | P82B96DR | 1 | I2C Long Distance Buffer | 2.79 | Link |
| | TPS35AA38AGADDFRQ1 | | External Watchdog | 2.51 | Link |
| | TMP100NA3K | 1 | I2C Temperature Sensor | 1.35 | Link |

| | | | | | |
|---|---|---|---|---|---|
| | LP38693MP | | Low Dropout Regulator for Setup | 1.37 | Link |
| | MIC29302AWD | 1 | SoM Power Supply | 2.33 | Link |
| | INA219BIDR | 1 | I2C SoM Power Monitor | 1.79 | Link |
| | TCA9535PWR | 1 | I2C GPIO Extender | 1.12 | Link |

# Appendix B – Monitoring Board Schematic



# Appendix C – Monitoring Board PCB Design by layers

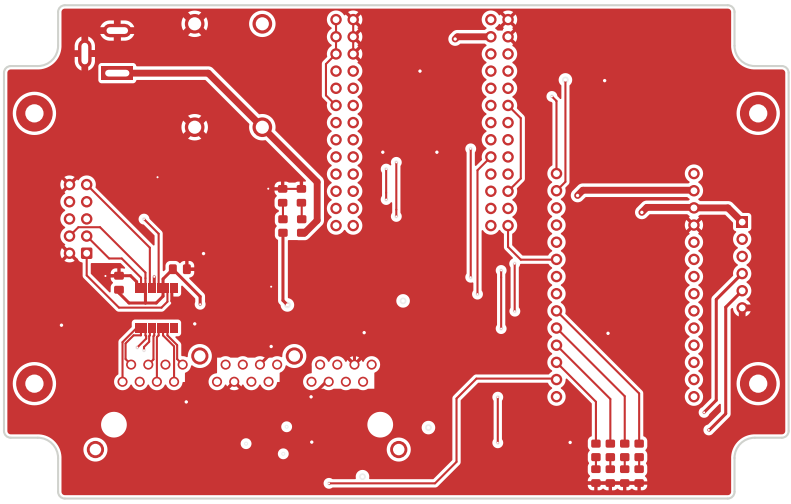


*Figure 70. Layer 1 – Front signal plane*

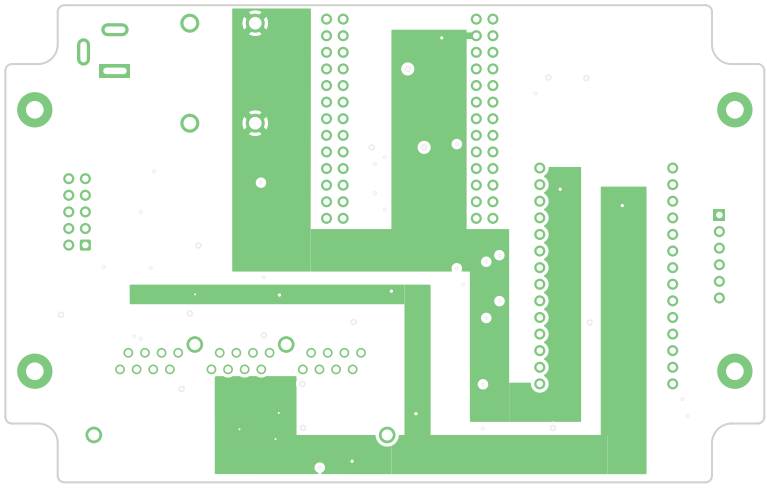


*Figure 71. Layer 2 - Power plane.*

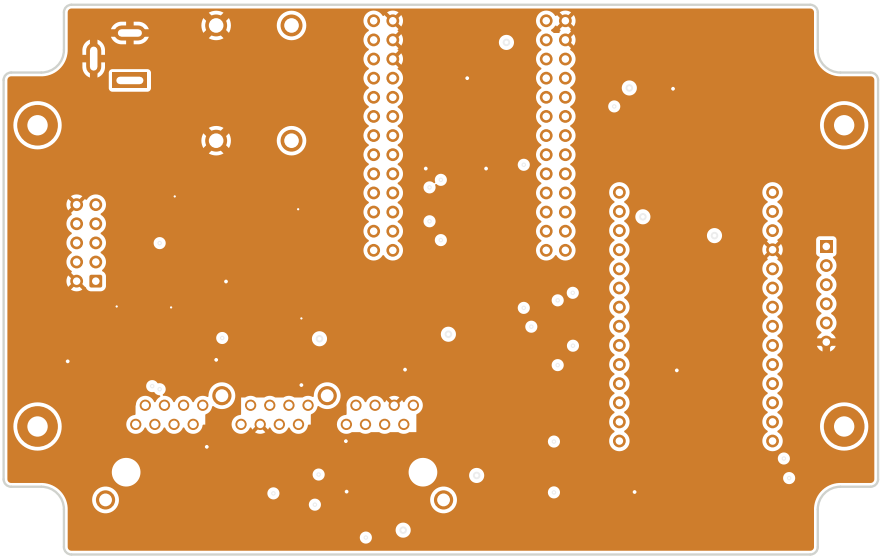


*Figure 72. Layer 3 - Ground plane.*

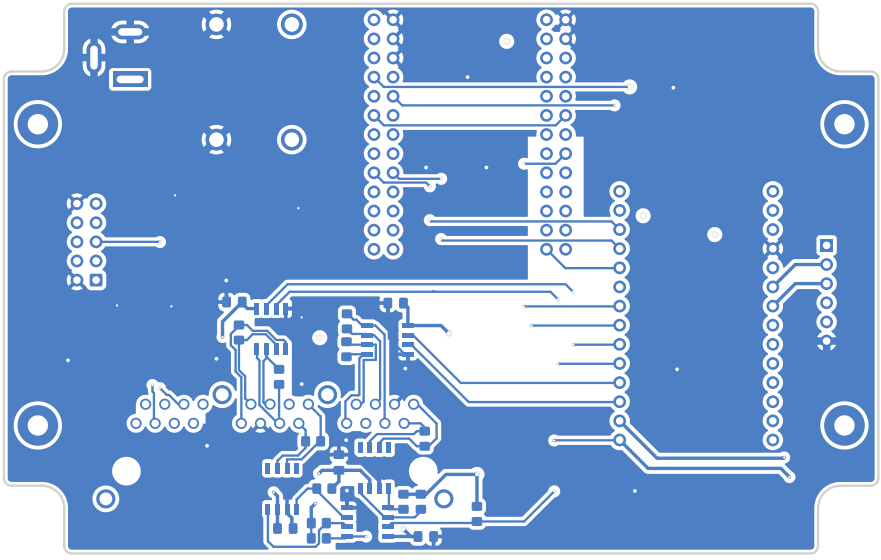


*Figure 73. Layer 4 - Back signal plane.*

## Appendix D – Code used to test I2C Communication

```
#include <Wire.h>

void setup() {
  Wire.begin();

  Serial.begin(9600);
  while (!Serial); // Leonardo: wait for serial monitor
  Serial.println("\nI2C Scanner");
}

void loop() {
  int nDevices = 0;

  Serial.println("Scanning...");

  for (byte address = 1; address < 127; ++address) {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    byte error = Wire.endTransmission();

    if (error == 0) {
      Serial.print("I2C device found at address 0x");
      if (address < 16) {
        Serial.print("0");
      }
      Serial.print(address, HEX);
      Serial.println("  !");

      ++nDevices;
    } else if (error == 4) {
      Serial.print("Unknown error at address 0x");
      if (address < 16) {
        Serial.print("0");
      }
      Serial.println(address, HEX);
    }
  }
  if (nDevices == 0) {
    Serial.println("No I2C devices found\n");
  } else {
    Serial.println("done\n");
  }
  delay(5000); // Wait 5 seconds for next scan
}
```

# Appendix E – VHDL Code and State Machines

The following Appendix gives more details on the implementation of each individual VHDL block, including it's block diagram, ports description, signals, procedures, state machines, and instantiations.

## UART

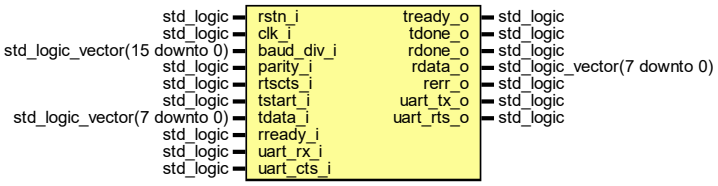



*Figure 74. UART block diagram.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| rstn_i | in | std_logic | |
| clk_i | in | std_logic | |
| baud_div_i | in | std_logic_vector(15 downto 0) | |
| parity_i | in | std_logic | |
| rtscts_i | in | std_logic | |
| tready_o | out | std_logic | |
| tstart_i | in | std_logic | |
| tdata_i | in | std_logic_vector(7 downto 0) | |
| tdone_o | out | std_logic | |
| rready_i | in | std_logic | |
| rdone_o | out | std_logic | |
| rdata_o | out | std_logic_vector(7 downto 0) | |
| rerr_o | out | std_logic | |
| uart_rx_i | in | std_logic | |
| uart_tx_o | out | std_logic | |
| uart_cts_i | in | std_logic | |
| uart_rts_o | out | std_logic | |

*Table 8. UART port description.*

Signals

| Name | Type | Description |
|---|---|---|
| baud_div_max_w | std_logic_vector(15 downto 0) | |
| baud_div_mid_w | std_logic_vector(15 downto 0) | |
| tx_curr_r | std_logic_vector(2 downto 0) | |
| tx_next_w | std_logic_vector(2 downto 0) | |
| tcounter_r | std_logic_vector(2 downto 0) | |
| tbaud_r | std_logic_vector(15 downto 0) | |
| tmax_w | std_logic | |
| rx_curr_r | std_logic_vector(2 downto 0) | |
| rx_next_w | std_logic_vector(2 downto 0) | |
| ctl_rbaud_clr_w | std_logic | |
| ctl_rbaud_cnt_w | std_logic | |
| ctl_rbit_clr_w | std_logic | |
| ctl_rbit_cnt_w | std_logic | |
| ctl_reg_rdata_w | std_logic | |
| ctl_reg_rparity_w | std_logic | |
| rdata_r | std_logic_vector(8 downto 0) | |
| rbaud_r | std_logic_vector(15 downto 0) | |
| rbaud_max_w | std_logic | |
| rbaud_mid_w | std_logic | |
| rcounter_r | std_logic_vector(2 downto 0) | |
| rbit_max_w | std_logic | |

*Table 9. UART signals.*

Constants

| Name | Type | Value | Description |
|---|---|---|---|
| TX_IDLE | std_logic_vector(2 downto 0) | "000" | |
| TX_WAIT_CTS | std_logic_vector(2 downto 0) | "001" | |
| TX_START | std_logic_vector(2 downto 0) | "010" | |
| TX_DATA | std_logic_vector(2 downto 0) | "011" | |

| Name | Type | Value | Description |
|---|---|---|---|
| TX_PARITY | std_logic_vector(2 downto 0) | "100" | |
| TX_STOP | std_logic_vector(2 downto 0) | "101" | |
| RX_IDLE | std_logic_vector(2 downto 0) | "000" | |
| RX_START | std_logic_vector(2 downto 0) | "001" | |
| RX_DATA | std_logic_vector(2 downto 0) | "010" | |
| RX_PARITY | std_logic_vector(2 downto 0) | "011" | |
| RX_STOP | std_logic_vector(2 downto 0) | "100" | |

*Table 10. UART constants.*

Processes

- p_TX_FSM: ( clk_i, rstn_i )
- p_TX_NEXT: ( all )
- p_TX_COUNTERS: ( clk_i, rstn_i )
- p_RX_FSM: ( clk_i, rstn_i )
- p_RX_NEXT: ( all )
- p_RX_COUNTERS: ( clk_i, rstn_i )
- p_RX_DATA: ( clk_i, rstn_i )

State machines

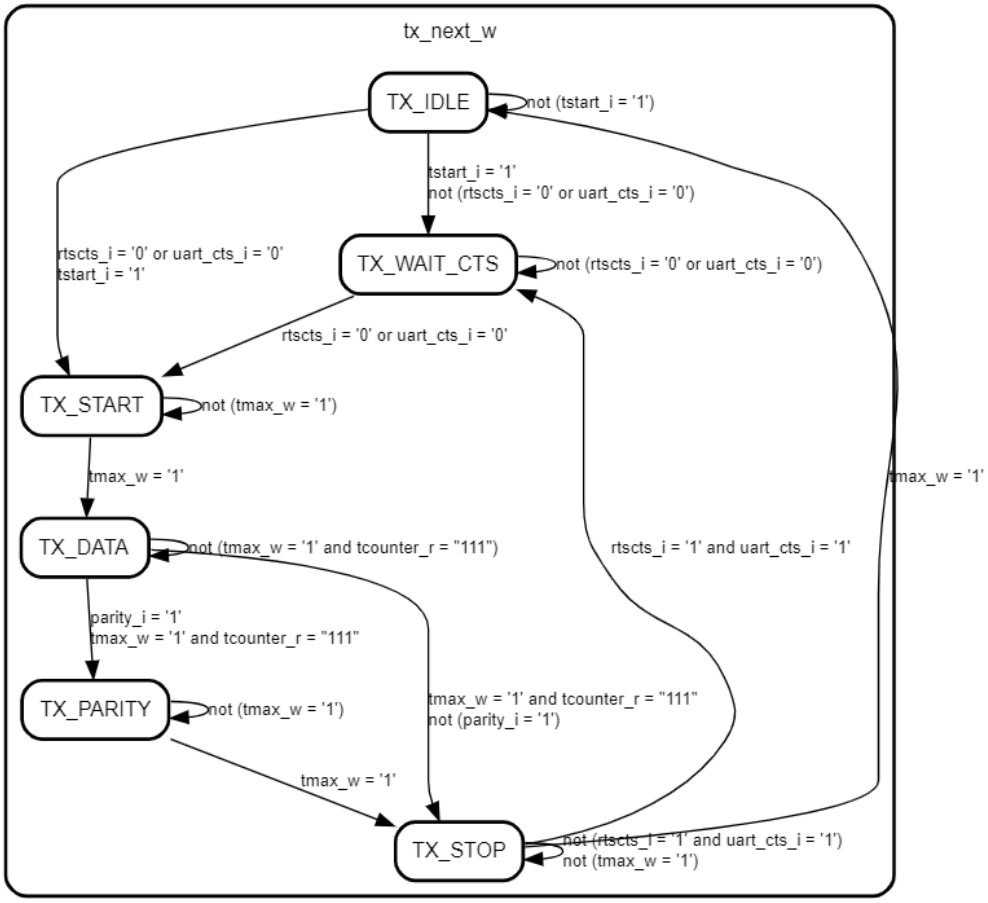



*Figure 75. UART Transmit FSM.*

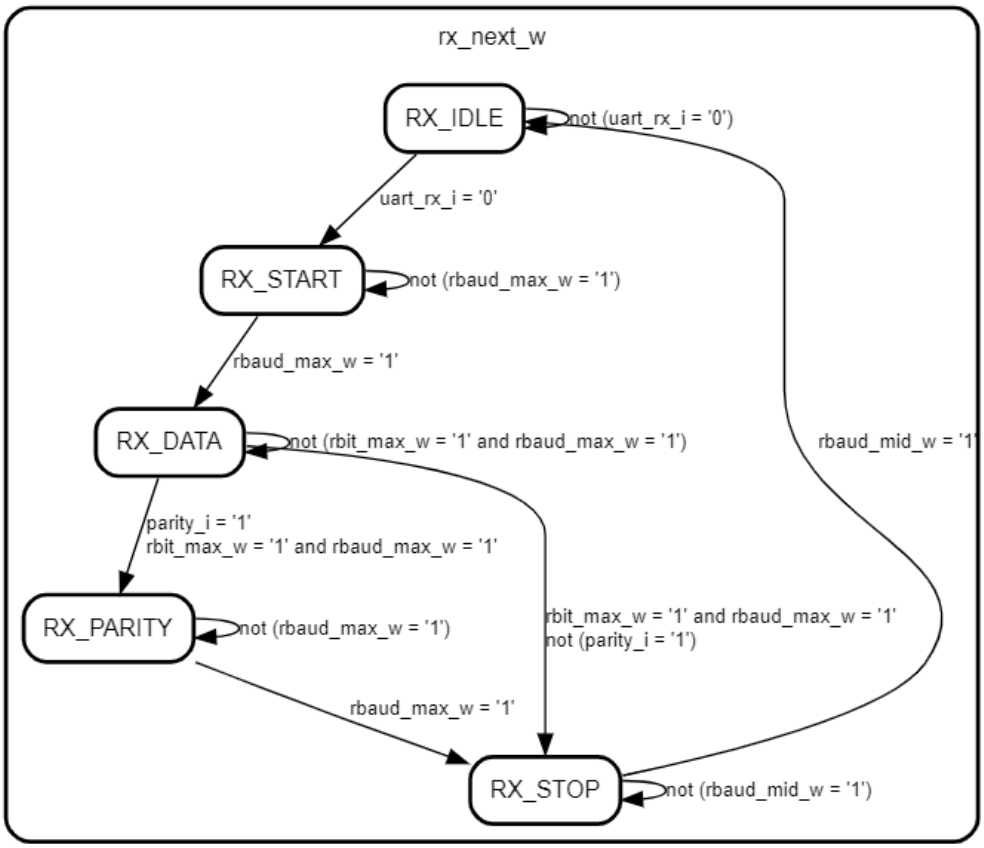
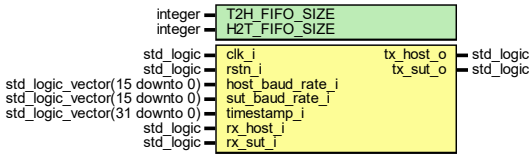



*Figure 76. UART Receive FSM.*

### Redirect buffer




Generics

| Generic name | Type | Value | Description |
|---|---|---|---|
| T2H_FIFO_SIZE | integer | 64 | |
| H2T_FIFO_SIZE | integer | 64 | |

*Table 11. Redirect Buffer generics.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| clk_i | in | std_logic | |
| rstn_i | in | std_logic | |
| host_baud_rate_i | in | std_logic_vector(15 downto 0) | |
| sut_baud_rate_i | in | std_logic_vector(15 downto 0) | |
| timestamp_i | in | std_logic_vector(31 downto 0) | |
| tx_host_o | out | std_logic | |
| rx_host_i | in | std_logic | |
| rx_sut_i | in | std_logic | |
| tx_sut_o | out | std_logic | |

*Table 12. Redirect Buffer ports description.*

Signals

| Name | Type | Description |
|---|---|---|
| t2h_uart_rdone_w | std_logic | |
| t2h_uart_rdata_w | std_logic_vector(7 downto 0) | |
| t2h_fifo_valid_w | std_logic | |
| t2h_tready_w | std_logic | |
| t2h_fifo_tdata_r | std_logic_vector(7 downto 0) | |
| h2t_uart_rdone_w | std_logic | |
| h2t_uart_rdata_w | std_logic_vector(7 downto 0) | |
| h2t_fifo_valid_w | std_logic | |

| Name | Type | Description |
|---|---|---|
| h2t_tready_w | std_logic | |
| h2t_fifo_tdata_r | std_logic_vector(7 downto 0) | |

*Table 13. Redirect Buffer signals.*

Instantiations

- uart_transceivers_u: work.uart
- uart_host_u: work.uart

### Current Report

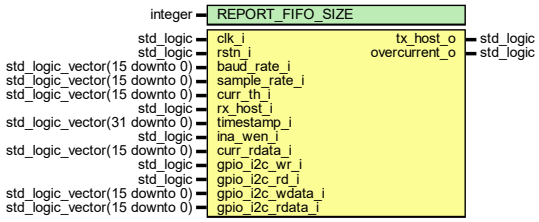



*Figure 77. Current Report block diagram.*

Generics

| Generic name | Type | Value | Description |
|---|---|---|---|
| REPORT_FIFO_SIZE | integer | 64 | |

*Table 14. Current Report generics.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| clk_i | in | std_logic | |
| rstn_i | in | std_logic | |
| baud_rate_i | in | std_logic_vector(15 downto 0) | |
| sample_rate_i | in | std_logic_vector(15 downto 0) | |
| curr_th_i | in | std_logic_vector(15 downto 0) | |
| rx_host_i | in | std_logic | |
| tx_host_o | out | std_logic | |
| timestamp_i | in | std_logic_vector(31 downto 0) | |
| ina_wen_i | in | std_logic | |
| curr_rdata_i | in | std_logic_vector(15 downto 0) | |
| gpio_i2c_wr_i | in | std_logic | |
| gpio_i2c_rd_i | in | std_logic | |

| Port name | Direction | Type | Description |
|---|---|---|---|
| gpio_i2c_wdata_i | in | std_logic_vector(15 downto 0) | |
| gpio_i2c_rdata_i | in | std_logic_vector(15 downto 0) | |
| overcurrent_o | out | std_logic | |

*Table 15. Current Report ports description.*

Signals

| Name | Type | Description |
|---|---|---|
| current_timestamp_w | std_logic_vector(31 downto 0) | |
| current_w | std_logic_vector(15 downto 0) | |
| log_overcurrent_w | std_logic | |
| gpio_wtimestamp_w | std_logic_vector(31 downto 0) | |
| gpio_wdata_w | std_logic_vector(15 downto 0) | |
| gpio_rtimestamp_w | std_logic_vector(31 downto 0) | |
| gpio_rdata_w | std_logic_vector(15 downto 0) | |
| log_gpio_w | std_logic | |
| sample_counter_r | std_logic_vector(31 downto 0) | |
| sample_max_w | std_logic | |
| fifo_write_w | std_logic | |
| fifo_wdata_w | std_logic_vector(REPORT_SIZE-1 downto 0) | |
| fifo_full_w | std_logic | |
| fifo_empty_w | std_logic | |
| fifo_valid_w | std_logic | |
| fifo_read_w | std_logic | |
| fifo_rdata_w | std_logic_vector(REPORT_SIZE-1 downto 0) | |

*Table 16. Current Report signals.*

Constants

| Name | Type | Value | Description |
|---|---|---|---|
| REPORT_SIZE | integer | 144 | |

*Table 17. Current Report constants.*

Processes

- sample_rate_p: ( rstn_i, clk_i )

Instantiations

- fifo_u: work.fifo

### Overcurrent handler

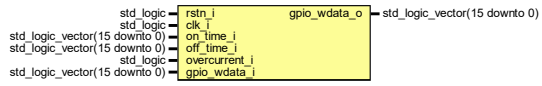



*Figure 78. Overcurrent Handler block diagram.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| rstn_i | in | std_logic | |
| clk_i | in | std_logic | |
| on_time_i | in | std_logic_vector(15 downto 0) | |
| off_time_i | in | std_logic_vector(15 downto 0) | |
| overcurrent_i | in | std_logic | |
| gpio_wdata_i | in | std_logic_vector(15 downto 0) | |
| gpio_wdata_o | out | std_logic_vector(15 downto 0) | |

*Table 18. Overcurrent Handler ports description.*

Signals

| Name | Type | Description |
|---|---|---|
| state_r | std_logic_vector(1 downto 0) | |
| next_w | std_logic_vector(1 downto 0) | |
| counter_r | std_logic_vector(31 downto 0) | |
| on_time_done_w | std_logic | |
| off_time_done_w | std_logic | |
| sut_force_off_w | std_logic | |

*Table 19. Overcurrent Handler signals.*

Constants

| Name | Type | Value | Description |
|---|---|---|---|
| IDLE | std_logic_vector(1 downto 0) | "00" | |
| SUT_ON | std_logic_vector(1 downto 0) | "01" | |

| | | | |
|---|---|---|---|
| SUT_OFF | std_logic_vector(1 downto 0) | "10" | |
| MASK_GPIO_WDATA_SOM_OFF | std_logic_vector(15 downto 0) | x"FFDD" | |

*Table 20. Overcurrent Handler constants.*

Processes

- state_p: ( rstn_i, clk_i )
- next_p: ( all )
- counter_p: ( rstn_i, clk_i )

State machines

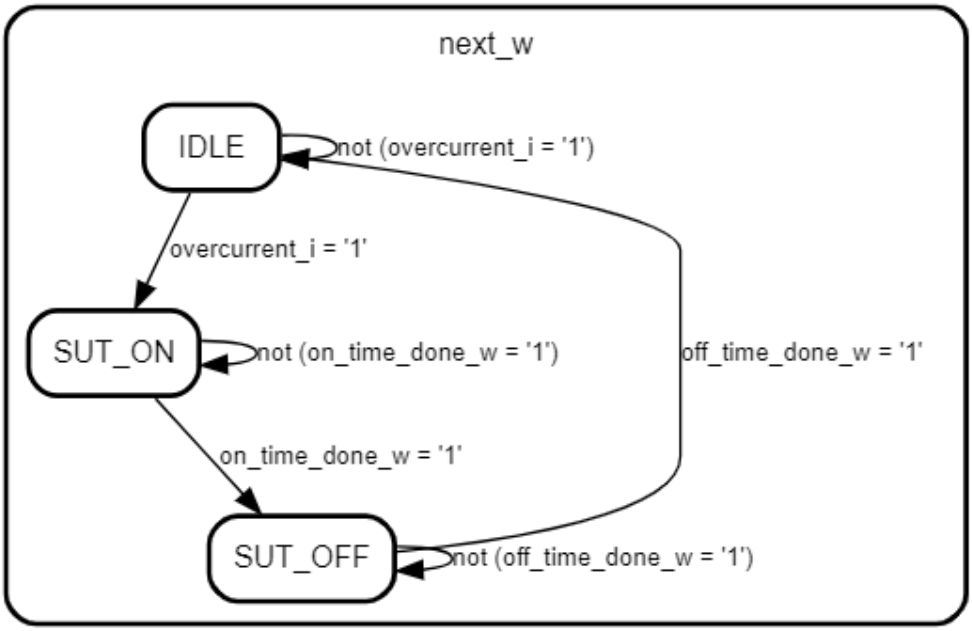



*Figure 79. Overcurrent Handler FSM.*

## Command Interface

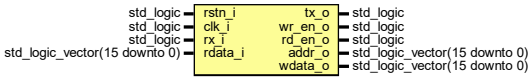



*Figure 80. Command Interface block diagram.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| rstn_i | in | std_logic | |
| clk_i | in | std_logic | |
| rx_i | in | std_logic | |
| tx_o | out | std_logic | |
| wr_en_o | out | std_logic | |
| rd_en_o | out | std_logic | |
| addr_o | out | std_logic_vector(15 downto 0) | |
| wdata_o | out | std_logic_vector(15 downto 0) | |

| Port name | Direction | Type | Description |
|---|---|---|---|
| rdata_i | in | std_logic_vector(15 downto 0) | |

*Table 21. Command Interface ports description.*

Signals

| Name | Type | Description |
|---|---|---|
| state_r | std_logic_vector(4 downto 0) | |
| next_w | std_logic_vector(4 downto 0) | |
| uart_rdone_w | std_logic | |
| uart_rdata_w | std_logic_vector(7 downto 0) | |
| uart_tstart_w | std_logic | |
| uart_tdata_w | std_logic_vector(7 downto 0) | |
| uart_tdone_w | std_logic | |
| en_waddr_w | std_logic_vector(3 downto 0) | |
| en_wdata_w | std_logic_vector(3 downto 0) | |
| en_wreg_w | std_logic | |
| en_raddr_w | std_logic_vector(3 downto 0) | |
| en_rdata_w | std_logic_vector(3 downto 0) | |
| en_rreg_w | std_logic | |
| addr_r | std_logic_vector(15 downto 0) | |
| data_r | std_logic_vector(15 downto 0) | |
| converted_rdata_w | std_logic_vector (3 downto 0) | |
| selected_wdata_w | std_logic_vector(3 downto 0) | |
| utf8_tdata_w | std_logic_vector(7 downto 0) | |
| write_lf_w | std_logic | |
| reg_data_w | std_logic_vector(15 downto 0) | |

*Table 22. Command Interface signals.*

Constants

| Name | Type | Value | Description |
|---|---|---|---|
| IDLE | std_logic_vector (4 downto 0) | "00000" | |
| WRITE_LF | std_logic_vector (4 downto 0) | "00001" | |
| WADDR3 | std_logic_vector (4 downto 0) | "00010" | |

| | | | |
|---|---|---|---|
| WADDR2 | std_logic_vector (4 downto 0) | "00011" | |
| WADDR1 | std_logic_vector (4 downto 0) | "00100" | |
| WADDR0 | std_logic_vector (4 downto 0) | "00101" | |
| WDATA3 | std_logic_vector (4 downto 0) | "00110" | |
| WDATA2 | std_logic_vector (4 downto 0) | "00111" | |
| WDATA1 | std_logic_vector (4 downto 0) | "01000" | |
| WDATA0 | std_logic_vector (4 downto 0) | "01001" | |
| REG_WRITE | std_logic_vector (4 downto 0) | "01010" | |
| RADDR3 | std_logic_vector (4 downto 0) | "01011" | |
| RADDR2 | std_logic_vector (4 downto 0) | "01100" | |
| RADDR1 | std_logic_vector (4 downto 0) | "01101" | |
| RADDR0 | std_logic_vector (4 downto 0) | "01110" | |
| REG_READ | std_logic_vector (4 downto 0) | "01111" | |
| RDATA3 | std_logic_vector (4 downto 0) | "10000" | |
| RDATA2 | std_logic_vector (4 downto 0) | "10001" | |
| RDATA1 | std_logic_vector (4 downto 0) | "10010" | |
| RDATA0 | std_logic_vector (4 downto 0) | "10011" | |

*Table 23. Command Interface constants.*

Processes

- current_state_p: ( clk_i, rstn_i )
- next_state_p: ( all )
- get_address_p: ( clk_i )
- get_data_p: ( clk_i )

Instantiations

- utf8_hex_u: work.utf8_hex
- hex_utf8_u: work.hex_utf8
- uart_u_1: work.uart

State machines

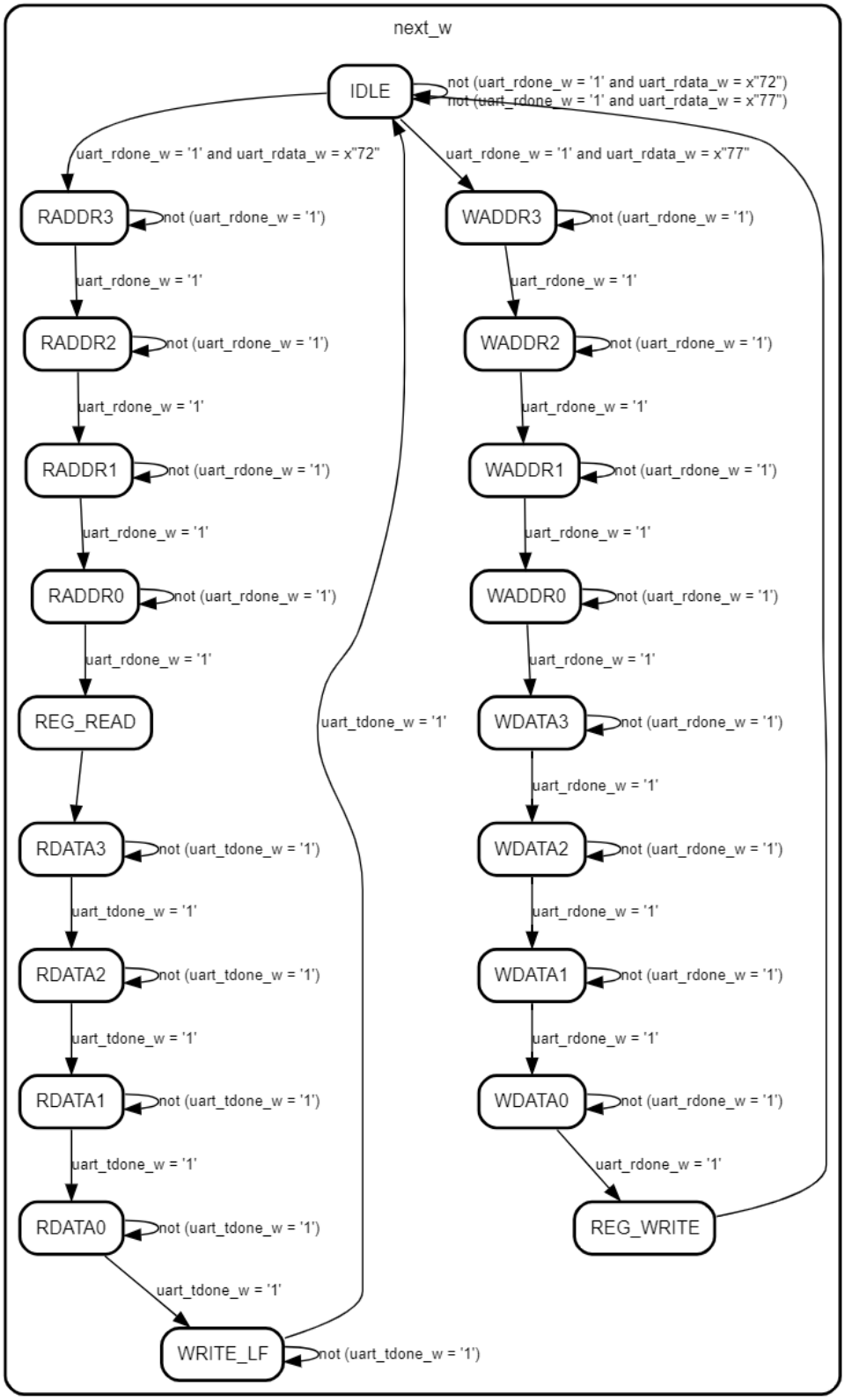



*Figure 81. Command Interface FSM.*

## Registers

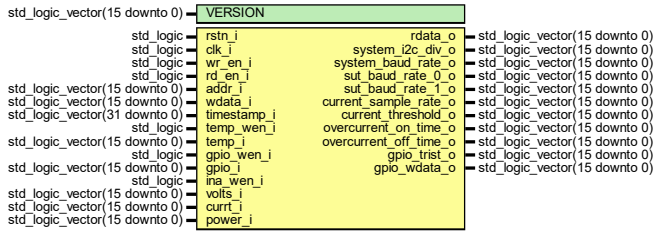



*Figure 82. Registers block diagram.*

Generics

| Generic name | Type | Value | Description |
|---|---|---|---|
| VERSION | std_logic_vector(15 downto 0) | | |

*Figure 83. Registers generics.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| rstn_i | in | std_logic | |
| clk_i | in | std_logic | |
| wr_en_i | in | std_logic | |
| rd_en_i | in | std_logic | |
| addr_i | in | std_logic_vector(15 downto 0) | |
| wdata_i | in | std_logic_vector(15 downto 0) | |
| rdata_o | out | std_logic_vector(15 downto 0) | |
| system_i2c_div_o | out | std_logic_vector(15 downto 0) | |
| system_baud_rate_o | out | std_logic_vector(15 downto 0) | |
| sut_baud_rate_0_o | out | std_logic_vector(15 downto 0) | |
| sut_baud_rate_1_o | out | std_logic_vector(15 downto 0) | |
| current_sample_rate_o | out | std_logic_vector(15 downto 0) | |
| current_threshold_o | out | std_logic_vector(15 downto 0) | |

| Port name | Direction | Type | Description |
|---|---|---|---|
| overcurrent_on_time_o | out | std_logic_vector(15 downto 0) | |
| overcurrent_off_time_o | out | std_logic_vector(15 downto 0) | |
| timestamp_i | in | std_logic_vector(31 downto 0) | |
| gpio_trist_o | out | std_logic_vector(15 downto 0) | |
| gpio_wdata_o | out | std_logic_vector(15 downto 0) | |
| temp_wen_i | in | std_logic | |
| temp_i | in | std_logic_vector(15 downto 0) | |
| gpio_wen_i | in | std_logic | |
| gpio_i | in | std_logic_vector(15 downto 0) | |
| ina_wen_i | in | std_logic | |
| volts_i | in | std_logic_vector(15 downto 0) | |
| currt_i | in | std_logic_vector(15 downto 0) | |
| power_i | in | std_logic_vector(15 downto 0) | |

*Figure 84. Registers ports description.*

Signals

| Name | Type | Description |
|---|---|---|
| version_w | std_logic_vector(15 downto 0) | |
| board_name_r | std_logic_vector(15 downto 0) | |
| system_i2c_div_r | std_logic_vector(15 downto 0) | |
| system_baud_rate_r | std_logic_vector(15 downto 0) | |
| sut_baud_rate_0_r | std_logic_vector(15 downto 0) | |
| sut_baud_rate_1_r | std_logic_vector(15 downto 0) | |
| current_sample_rate_r | std_logic_vector(15 downto 0) | |

| Name | Type | Description |
|---|---|---|
| current_threshold_r | std_logic_vector(15 downto 0) | |
| overcurrent_on_time_r | std_logic_vector(15 downto 0) | |
| overcurrent_off_time_r | std_logic_vector(15 downto 0) | |
| gpio_tri_st_r | std_logic_vector(15 downto 0) | |
| gpio_read_r | std_logic_vector(15 downto 0) | |
| gpio_write_r | std_logic_vector(15 downto 0) | |
| temperature_r | std_logic_vector(15 downto 0) | |
| current_r | std_logic_vector(15 downto 0) | |
| voltage_r | std_logic_vector(15 downto 0) | |
| power_r | std_logic_vector(15 downto 0) | |

*Figure 85. Registers signals.*

Constants

| Name | Type | Value | Description |
|---|---|---|---|
| ADDR_VERSION | std_logic_vector (15 downto 0) | x"0000" | |
| ADDR_BOARD_NAME | std_logic_vector (15 downto 0) | x"0001" | |
| ADDR_SYSTEM_I2C_DIV | std_logic_vector (15 downto 0) | x"0002" | |
| ADDR_SYSTEM_UART_BAUD_RATE | std_logic_vector (15 downto 0) | x"0003" | |
| ADDR_SUT_UART_BAUD_RATE0 | std_logic_vector (15 downto 0) | x"0004" | |
| ADDR_SUT_UART_BAUD_RATE1 | std_logic_vector (15 downto 0) | x"0005" | |
| ADDR_CURRENT_SAMPLERATE | std_logic_vector (15 downto 0) | x"0006" | |
| ADDR_CURRENT_THRESHOLD | std_logic_vector (15 downto 0) | x"0007" | |
| ADDR_OVERCURRENT_ON_TIME | std_logic_vector (15 downto 0) | x"0008" | |
| ADDR_OVERCURRENT_OFF_TIME | std_logic_vector (15 downto 0) | x"0009" | |

| Name | Type | Value | Description |
|---|---|---|---|
| ADDR_TIMESTAMP_H | std_logic_vector (15 downto 0) | x"0100" | |
| ADDR_TIMESTAMP_L | std_logic_vector (15 downto 0) | x"0101" | |
| ADDR_GPIO_TRI_ST | std_logic_vector (15 downto 0) | x"0200" | |
| ADDR_GPIO_READ | std_logic_vector (15 downto 0) | x"0201" | |
| ADDR_GPIO_WRITE | std_logic_vector (15 downto 0) | x"0202" | |
| ADDR_VOLTAGE | std_logic_vector (15 downto 0) | x"0203" | |
| ADDR_CURRENT | std_logic_vector (15 downto 0) | x"0204" | |
| ADDR_POWER | std_logic_vector (15 downto 0) | x"0205" | |
| ADDR_TEMPERATURE | std_logic_vector (15 downto 0) | x"0206" | |
| DEFAULT_BOARD_NAME | std_logic_vector (15 downto 0) | x"CAFE" | |
| DEFAULT_SYSTEM_I2C_DIV | std_logic_vector (15 downto 0) | x"01F4" | |
| DEFAULT_SYSTEM_BAUDRATE | std_logic_vector (15 downto 0) | x"01b2" | |
| DEFAULT_SUT_UART_BAUDRATE | std_logic_vector (15 downto 0) | x"01b2" | |
| DEFAULT_CURRENT_SAMPLERATE | std_logic_vector (15 downto 0) | x"017d" | |
| DEFAULT_CURRENT_THRESHOLD | std_logic_vector (15 downto 0) | x"7FFF" | |
| DEFAULT_OVERCURRENT_ON_TIME | std_logic_vector (15 downto 0) | x"0026" | |
| DEFAULT_OVERCURRENT_OFF_TIME | std_logic_vector (15 downto 0) | x"0099" | |

| Name | Type | Value | Description |
|---|---|---|---|
| DEFAULT_GPIO_TRISTATE | std_logic_vector (15 downto 0) | x"FFFF" | |
| DEFAULT_GPIO_WRITE | std_logic_vector (15 downto 0) | | |

*Figure 86. Registers constants.*

Processes

- register_defaults_p: ( clk_i, rstn_i )
- register_gpio_data_p: ( clk_i, rstn_i )
- register_temperature_data_p: ( clk_i, rstn_i )
- register_ina_data_p: ( clk_i, rstn_i )

## FIFO

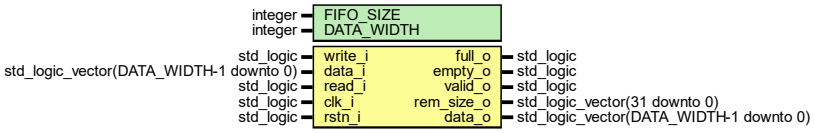



*Figure 87. FIFO block diagram.*

Generics

| Generic name | Type | Value | Description |
|---|---|---|---|
| FIFO_SIZE | integer | | |
| DATA_WIDTH | integer | | |

*Table 24. FIFO generics.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| write_i | in | std_logic | |
| data_i | in | std_logic_vector(DATA_WIDTH-1 downto 0) | |
| read_i | in | std_logic | |
| clk_i | in | std_logic | |
| rstn_i | in | std_logic | |
| full_o | out | std_logic | |
| empty_o | out | std_logic | |
| valid_o | out | std_logic | |
| rem_size_o | out | std_logic_vector(31 downto 0) | |
| data_o | out | std_logic_vector(DATA_WIDTH-1 downto 0) | |

*Table 25. FIFO ports descriptions.*

Signals

| Name | Type | Description |
|---|---|---|
| fifo_r | fifo_t(FIFO_SIZE-1 downto 0) | |
| first_addr_r | integer range 0 to FIFO_SIZE-1 | |
| insert_addr_r | integer range 0 to FIFO_SIZE-1 | |
| size_r | integer range 0 to FIFO_SIZE | |
| next_first_addr_w | integer range 0 to FIFO_SIZE-1 | |
| next_insert_addr_w | integer range 0 to FIFO_SIZE-1 | |

| Name | Type | Description |
|---|---|---|
| full_w | std_logic | |
| empty_w | std_logic | |

*Table 26. FIFO signals.*

Types

| Name | Type | Description |
|---|---|---|
| fifo_t | array(natural range <>) of std_logic_vector(DATA_WIDTH-1 downto 0) | |

*Table 27. FIFO Types.*

Processes

- p_MAIN: ( clk_i, rstn_i )

### Hex to UTF-8

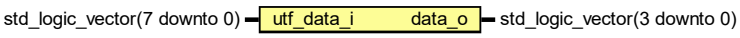



*Figure 88. Hex to UTF-8 block diagram.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| utf_data_i | in | std_logic_vector(7 downto 0) | |
| data_o | out | std_logic_vector(3 downto 0) | |

*Table 28. Hex to UTF-8 ports descriptions.*

### UTF-8 to Hex

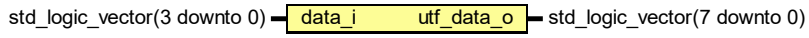



*Figure 89. UTF-8 to Hex block diagram.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| data_i | in | std_logic_vector(3 downto 0) | |
| utf_data_o | out | std_logic_vector(7 downto 0) | |

*Table 29. UTF-8 to Hex ports description.*

### I2C

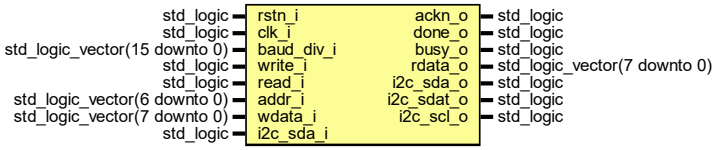



*Figure 90. I2C block diagram.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| rstn_i | in | std_logic | |
| clk_i | in | std_logic | |
| baud_div_i | in | std_logic_vector(15 downto 0) | |
| write_i | in | std_logic | |
| read_i | in | std_logic | |
| ackn_o | out | std_logic | |
| done_o | out | std_logic | |
| busy_o | out | std_logic | |
| addr_i | in | std_logic_vector(6 downto 0) | |
| wdata_i | in | std_logic_vector(7 downto 0) | |
| rdata_o | out | std_logic_vector(7 downto 0) | |
| i2c_sda_i | in | std_logic | |
| i2c_sda_o | out | std_logic | |
| i2c_sdat_o | out | std_logic | |
| i2c_scl_o | out | std_logic | |

*Table 30. I2C ports descriptions.*

Signals

| Name | Type | Description |
|---|---|---|
| state_r | std_logic_vector(3 downto 0) | |
| next_w | std_logic_vector(3 downto 0) | |
| sda_in_w | std_logic | |
| baud_count_w | std_logic | |
| bit_count_w | std_logic | |
| get_ack_w | std_logic | |

| Name | Type | Description |
|---|---|---|
| get_data_w | std_logic | |
| baud_counter_r | std_logic_vector(15 downto 0) | |
| baud_counter_max_w | std_logic | |
| baud_counter_mid_w | std_logic | |
| baud_counter_first_half_w | std_logic | |
| baud_counter_second_half_w | std_logic | |
| baud_counter_mid_quarter_w | std_logic | |
| scl_stop_w | std_logic | |
| bit_counter_r | std_logic_vector(2 downto 0) | |
| bit_counter_zero_w | std_logic | |
| addr_data_w | std_logic_vector(7 downto 0) | |
| ackn_r | std_logic | |
| rdata_r | std_logic_vector(7 downto 0) | |
| conf_baud_count_max_w | std_logic_vector(15 downto 0) | |
| conf_mid_baud_div_w | std_logic_vector(15 downto 0) | |
| conf_quarter_baud_div_w | std_logic_vector(15 downto 0) | |
| conf_threequarter_baud_div_w | std_logic_vector(15 downto 0) | |

*Table 31. I2C signals.*

Constants

| Name | Type | Value | Description |
|---|---|---|---|
| IDLE | std_logic_vector(3 downto 0) | x"0" | |
| I2C_START | std_logic_vector(3 downto 0) | x"1" | |
| I2C_ADDR | std_logic_vector(3 downto 0) | x"2" | |
| I2C_AACK | std_logic_vector(3 downto 0) | x"3" | |
| I2C_WDATA | std_logic_vector(3 downto 0) | x"4" | |
| WDONE | std_logic_vector(3 downto 0) | x"5" | |
| I2C_WACK | std_logic_vector(3 downto 0) | x"6" | |
| I2C_RDATA | std_logic_vector(3 downto 0) | x"7" | |
| RDONE | std_logic_vector(3 downto 0) | x"8" | |

| Name | Type | Value | Description |
|---|---|---|---|
| I2C_RACK | std_logic_vector(3 downto 0) | x"9" | |
| I2C_STOP | std_logic_vector(3 downto 0) | x"A" | |
| I2C_WAIT | std_logic_vector(3 downto 0) | x"B" | |
| I2C_ACKN | std_logic_vector(3 downto 0) | x"F" | |

*Table 32. I2C constants.*

Processes

- current_p: ( clk_i, rstn_i )
- next_p: ( state_r, write_i, read_i, baud_counter_max_w, bit_counter_zero_w, ackn_r )
- baud_counter_p: ( clk_i, rstn_i )
- baud_p: ( clk_i )
- rdata_p: ( clk_i )

State machines

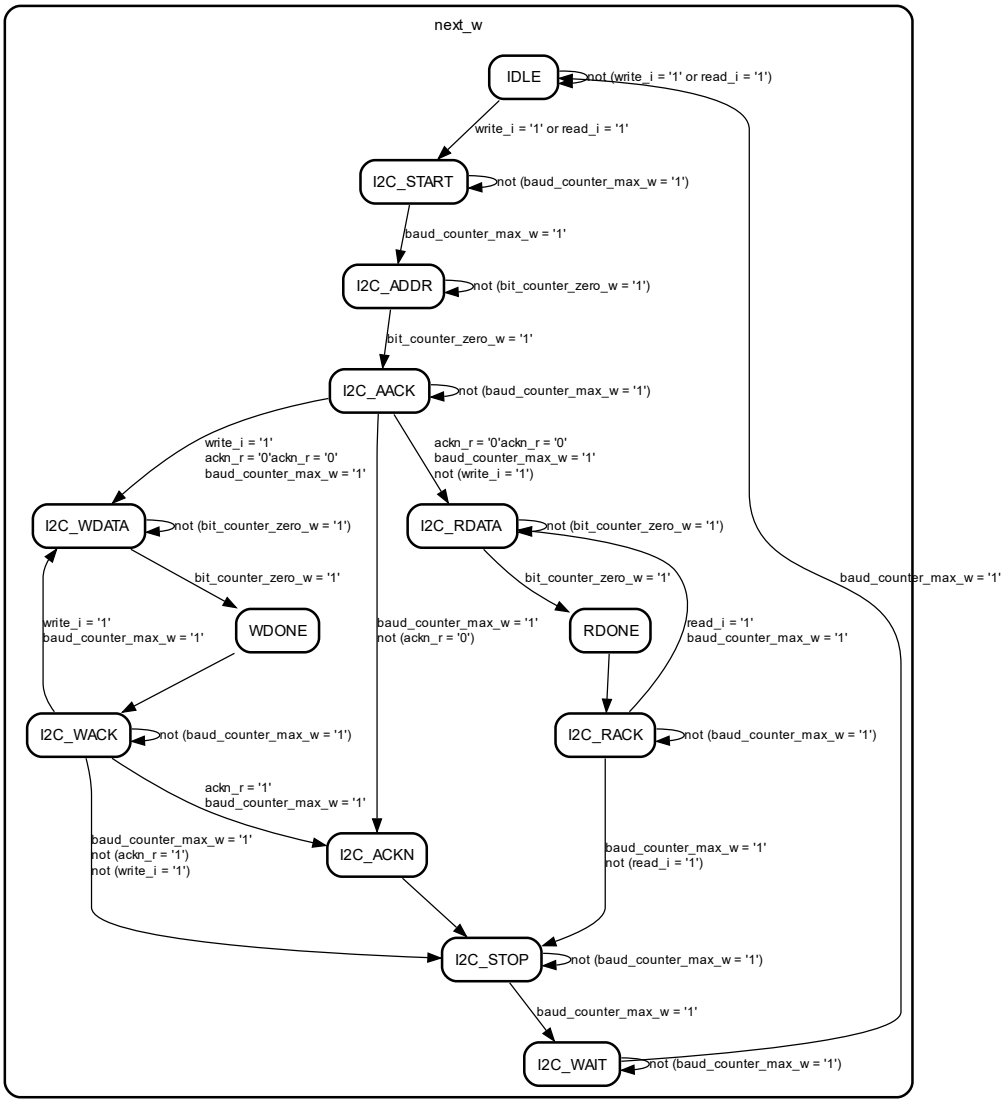



*Figure 91. I2C FSM.*

### I2C GPIO

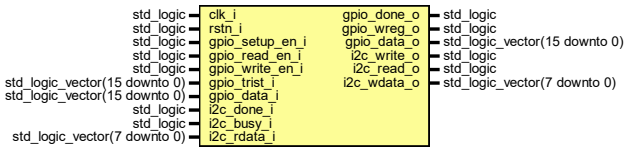



*Figure 92. I2C GPIO block diagram.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| clk_i | in | std_logic | |
| rstn_i | in | std_logic | |
| gpio_setup_en_i | in | std_logic | |
| gpio_read_en_i | in | std_logic | |
| gpio_write_en_i | in | std_logic | |
| gpio_trist_i | in | std_logic_vector(15 downto 0) | |
| gpio_data_i | in | std_logic_vector(15 downto 0) | |
| gpio_done_o | out | std_logic | |
| gpio_wreg_o | out | std_logic | |
| gpio_data_o | out | std_logic_vector(15 downto 0) | |
| i2c_done_i | in | std_logic | |
| i2c_busy_i | in | std_logic | |
| i2c_write_o | out | std_logic | |
| i2c_read_o | out | std_logic | |
| i2c_rdata_i | in | std_logic_vector(7 downto 0) | |
| i2c_wdata_o | out | std_logic_vector(7 downto 0) | |

*Table 33. I2C GPIO ports descriptions.*

Signals

| Name | Type | Description |
|---|---|---|
| state_r | std_logic_vector(4 downto 0) | |
| next_w | std_logic_vector(4 downto 0) | |
| gpio_read_r | std_logic_vector(15 downto 0) | |

*Table 34. I2c GPIO signals.*

Constants

| Name | Type | Value | Description |
|---|---|---|---|
| IDLE | std_logic_vector(4 downto 0) | "00000" | |
| CFG_WRITE_PTR1 | std_logic_vector(4 downto 0) | "00001" | |
| CFG_WRITE_BYTE1 | std_logic_vector(4 downto 0) | "00010" | |
| CFG_WAIT1 | std_logic_vector(4 downto 0) | "00011" | |
| CFG_WRITE_PTR2 | std_logic_vector(4 downto 0) | "00100" | |
| CFG_WRITE_BYTE2 | std_logic_vector(4 downto 0) | "00101" | |
| GPIO_WRITE_PTR1 | std_logic_vector(4 downto 0) | "00110" | |
| GPIO_WRITE_BYTE1 | std_logic_vector(4 downto 0) | "00111" | |
| GPIO_WRITE_WAIT | std_logic_vector(4 downto 0) | "01000" | |
| GPIO_WRITE_BYTE2 | std_logic_vector(4 downto 0) | "01010" | |
| GPIO_READ_PTR1 | std_logic_vector(4 downto 0) | "01011" | |
| GPIO_READ_BYTE1 | std_logic_vector(4 downto 0) | "01100" | |
| GPIO_READ_WAIT | std_logic_vector(4 downto 0) | "01101" | |
| GPIO_READ_PTR2 | std_logic_vector(4 downto 0) | "01110" | |
| GPIO_READ_BYTE2 | std_logic_vector(4 downto 0) | "01111" | |
| GPIO_REG_WRITE | std_logic_vector(4 downto 0) | "10000" | |
| BUSY | std_logic_vector(4 downto 0) | "10001" | |

| Name | Type | Value | Description |
|---|---|---|---|
| DONE | std_logic_vector(4 downto 0) | "10010" | |

*Table 35. I2C GPIO constants.*

Processes

- current_state_p: ( clk_i, rstn_i )
- next_state_p: ( all )
- register_gpio_data_p: ( clk_i, rstn_i )

State machines

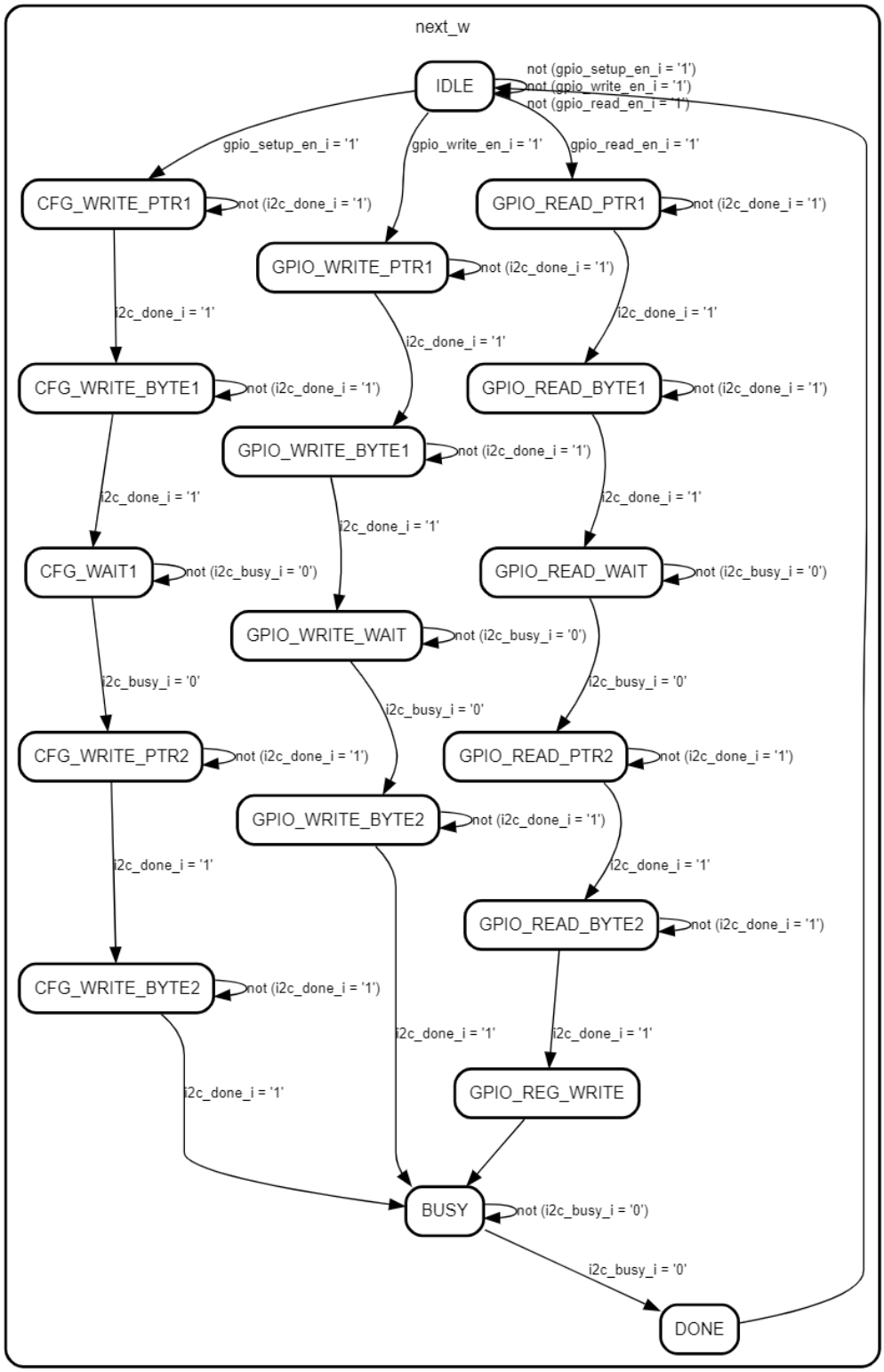



*Figure 93. I2C GPIO FSM.*

### I2C INA Current Monitor

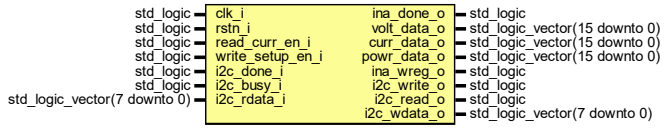



*Figure 94. I2C INA block diagram.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| clk_i | in | std_logic | |
| rstn_i | in | std_logic | |
| read_curr_en_i | in | std_logic | |
| write_setup_en_i | in | std_logic | |
| ina_done_o | out | std_logic | |
| volt_data_o | out | std_logic_vector(15 downto 0) | |
| curr_data_o | out | std_logic_vector(15 downto 0) | |
| powr_data_o | out | std_logic_vector(15 downto 0) | |
| ina_wreg_o | out | std_logic | |
| i2c_done_i | in | std_logic | |
| i2c_busy_i | in | std_logic | |
| i2c_write_o | out | std_logic | |
| i2c_read_o | out | std_logic | |
| i2c_rdata_i | in | std_logic_vector(7 downto 0) | |
| i2c_wdata_o | out | std_logic_vector(7 downto 0) | |

*Table 36. I2C INA ports descriptions.*

Signals

| Name | Type | Description |
|---|---|---|
| state_r | std_logic_vector(4 downto 0) | |
| next_w | std_logic_vector(4 downto 0) | |
| voltage_r | std_logic_vector(15 downto 0) | |
| current_r | std_logic_vector(15 downto 0) | |
| power_r | std_logic_vector(15 downto 0) | |

*Table 37. I2C INA signals.*

Constants

| Name | Type | Value | Description |
|---|---|---|---|
| IDLE | std_logic_vector(4 downto 0) | "00000" | |
| CFG_WRITE_PTR | std_logic_vector(4 downto 0) | "00001" | |
| CFG_WRITE_MSB | std_logic_vector(4 downto 0) | "00010" | |
| CFG_WRITE_WAIT1 | std_logic_vector(4 downto 0) | "00011" | |
| CFG_WRITE_LSB | std_logic_vector(4 downto 0) | "00100" | |
| CFG_WRITE_WAIT2 | std_logic_vector(4 downto 0) | "00101" | |
| CALIBR_WRITE_PTR | std_logic_vector(4 downto 0) | "00110" | |
| CALIBR_WRITE_MSB | std_logic_vector(4 downto 0) | "00111" | |
| CALIBR_WRITE_WAIT1 | std_logic_vector(4 downto 0) | "01000" | |
| CALIBR_WRITE_LSB | std_logic_vector(4 downto 0) | "01001" | |
| VOLT_WRITE_PTR | std_logic_vector(4 downto 0) | "01010" | |
| VOLT_READ_MSB | std_logic_vector(4 downto 0) | "01011" | |
| VOLT_WAIT1 | std_logic_vector(4 downto 0) | "01100" | |
| VOLT_READ_LSB | std_logic_vector(4 downto 0) | "01101" | |
| VOLT_WAIT2 | std_logic_vector(4 downto 0) | "01110" | |
| CURR_WRITE_PTR | std_logic_vector(4 downto 0) | "01111" | |
| CURR_READ_MSB | std_logic_vector(4 downto 0) | "10000" | |

| Name | Type | Value | Description |
|---|---|---|---|
| CURR_WAIT1 | std_logic_vector(4 downto 0) | "10001" | |
| CURR_READ_LSB | std_logic_vector(4 downto 0) | "10010" | |
| CURR_WAIT2 | std_logic_vector(4 downto 0) | "10011" | |
| POWR_WRITE_PTR | std_logic_vector(4 downto 0) | "10100" | |
| POWR_READ_MSB | std_logic_vector(4 downto 0) | "10101" | |
| POWR_WAIT1 | std_logic_vector(4 downto 0) | "10110" | |
| POWR_READ_LSB | std_logic_vector(4 downto 0) | "10111" | |
| BUSY | std_logic_vector(4 downto 0) | "11000" | |
| DONE | std_logic_vector(4 downto 0) | "11001" | |
| INA_REG_WRITE | std_logic_vector(4 downto 0) | "11010" | |

*Table 38. I2C INA constants.*

Processes

- current_state_p: ( clk_i, rstn_i )
- next_state_p: ( all )
- register_data_p: ( clk_i, rstn_i )

Configuration Register Value
39F9
Calibration Register Value
*5000*

State machines

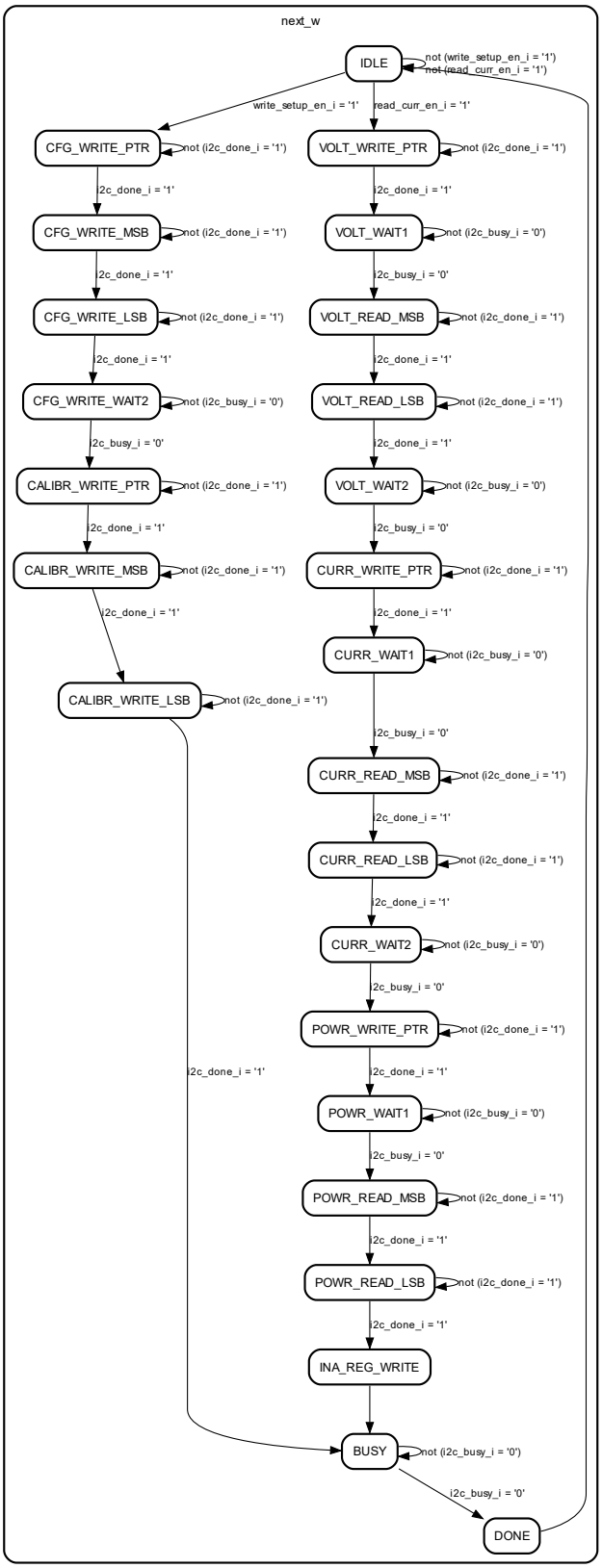



*Table 39. I2C INA FSM.*

### I2C TMP100

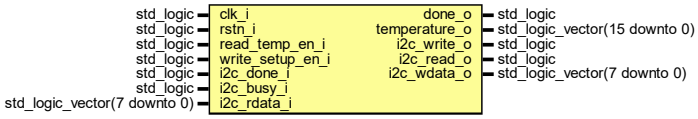



*Figure 95. I2C TMP100 block diagram.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| clk_i | in | std_logic | |
| rstn_i | in | std_logic | |
| read_temp_en_i | in | std_logic | |
| write_setup_en_i | in | std_logic | |
| done_o | out | std_logic | |
| temperature_o | out | std_logic_vector(15 downto 0) | |
| i2c_done_i | in | std_logic | |
| i2c_busy_i | in | std_logic | |
| i2c_write_o | out | std_logic | |
| i2c_read_o | out | std_logic | |
| i2c_rdata_i | in | std_logic_vector(7 downto 0) | |
| i2c_wdata_o | out | std_logic_vector(7 downto 0) | |

*Figure 96. I2C TMP100 ports descriptions.*

Signals

| Name | Type | Description |
|---|---|---|
| state_r | std_logic_vector(3 downto 0) | |
| next_w | std_logic_vector(3 downto 0) | |
| temperature_r | std_logic_vector(15 downto 0) | |

*Figure 97. I2C TMP100 signals.*

Constants

| Name | Type | Value | Description |
|---|---|---|---|
| IDLE | std_logic_vector(3 downto 0) | x"0" | |
| CFG_WRITE_PTR | std_logic_vector(3 downto 0) | x"1" | |
| CFG_WRITE | std_logic_vector(3 downto 0) | x"2" | |

| Name | Type | Value | Description |
|---|---|---|---|
| TMP_WRITE_PTR | std_logic_vector(3 downto 0) | x"3" | |
| TMP_WAIT | std_logic_vector(3 downto 0) | x"5" | |
| TMP_READ_BYTE1 | std_logic_vector(3 downto 0) | x"6" | |
| TMP_READ_BYTE2 | std_logic_vector(3 downto 0) | x"7" | |
| BUSY | std_logic_vector(3 downto 0) | x"8" | |
| DONE | std_logic_vector(3 downto 0) | x"9" | |

*Figure 98. I2C TMP100 constants.*

Processes

- current_p: ( clk_i, rstn_i )
- next_p: ( all )
- register_temp_data_p: ( clk_i, rstn_i )

Registers

- Configuration Register

State machines

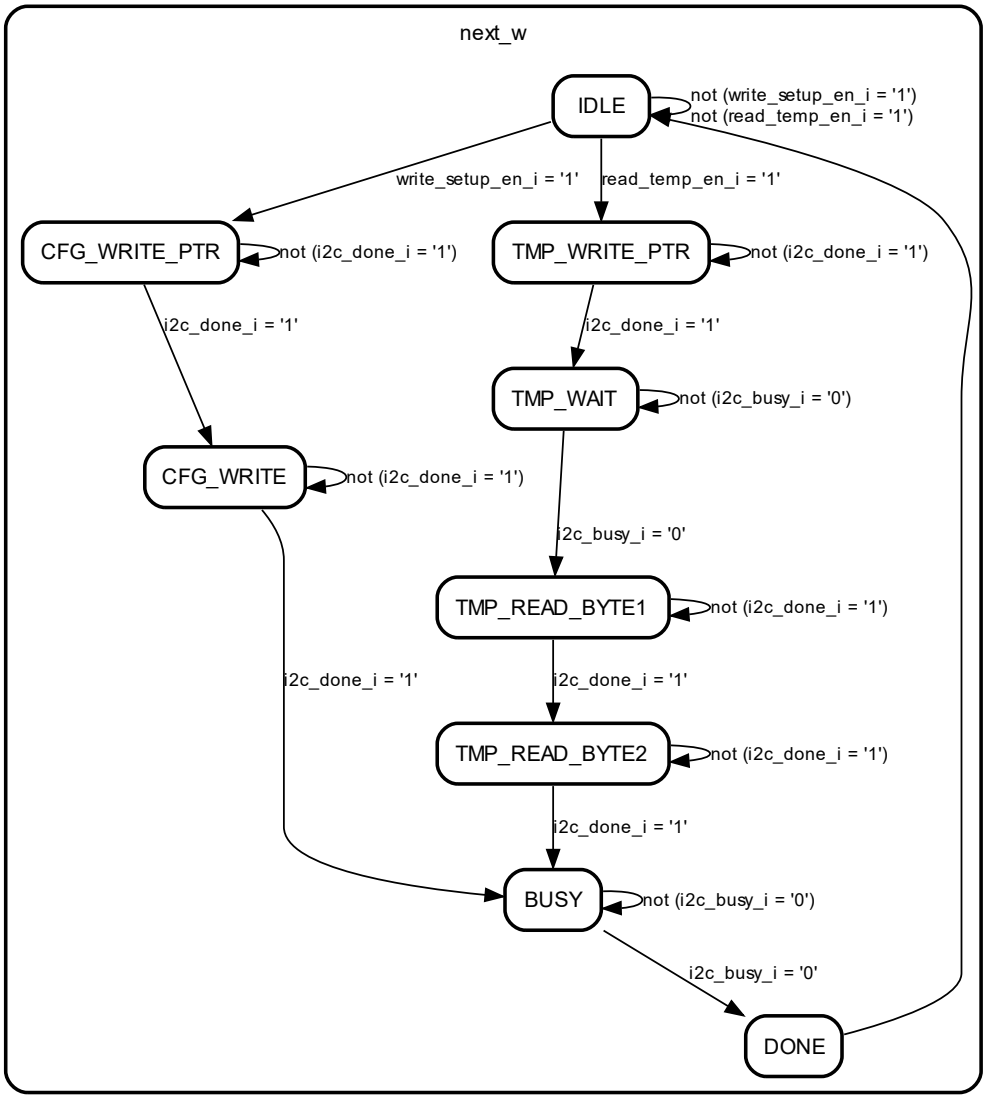



*Figure 99. I2C TMP100 FSM.*

## I2C Controller

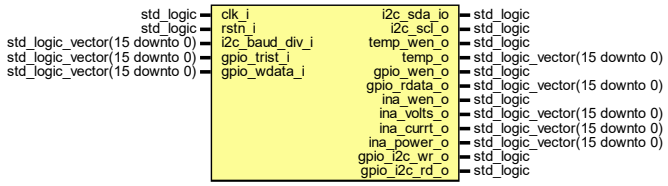



*Figure 100. I2C Controller block diagram.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| clk_i | in | std_logic | |
| rstn_i | in | std_logic | |
| i2c_baud_div_i | in | std_logic_vector(15 downto 0) | |
| i2c_sda_io | inout | std_logic | |
| i2c_scl_o | out | std_logic | |
| temp_wen_o | out | std_logic | |
| temp_o | out | std_logic_vector(15 downto 0) | |
| gpio_wen_o | out | std_logic | |
| gpio_rdata_o | out | std_logic_vector(15 downto 0) | |
| gpio_trist_i | in | std_logic_vector(15 downto 0) | |
| gpio_wdata_i | in | std_logic_vector(15 downto 0) | |
| ina_wen_o | out | std_logic | |
| ina_volts_o | out | std_logic_vector(15 downto 0) | |
| ina_currt_o | out | std_logic_vector(15 downto 0) | |
| ina_power_o | out | std_logic_vector(15 downto 0) | |
| gpio_i2c_wr_o | out | std_logic | |
| gpio_i2c_rd_o | out | std_logic | |

*Figure 101. I2C Controller ports descriptions.*

Signals

| Name | Type | Description |
|---|---|---|
| state_r | std_logic_vector(3 downto 0) | |
| next_w | std_logic_vector(3 downto 0) | |
| temp_done_w | std_logic | |
| temp_en_w | std_logic | |
| temp_setup_w | std_logic | |
| i2c_tmp100_write_w | std_logic | |
| i2c_tmp100_read_w | std_logic | |
| i2c_tmp100_done_w | std_logic | |
| i2c_tmp100_busy_w | std_logic | |

| Name | Type | Description |
|---|---|---|
| i2c_tmp100_wdata_w | std_logic_vector(7 downto 0) | |
| i2c_tmp100_rdata_w | std_logic_vector(7 downto 0) | |
| gpio_done_w | std_logic | |
| gpio_wen_w | std_logic | |
| gpio_ren_w | std_logic | |
| gpio_setup_w | std_logic | |
| i2c_gpio_write_w | std_logic | |
| i2c_gpio_read_w | std_logic | |
| i2c_gpio_done_w | std_logic | |
| i2c_gpio_busy_w | std_logic | |
| i2c_gpio_wdata_w | std_logic_vector(7 downto 0) | |
| i2c_gpio_rdata_w | std_logic_vector(7 downto 0) | |
| ina_done_w | std_logic | |
| ina_en_w | std_logic | |
| ina_setup_w | std_logic | |
| i2c_ina_done_w | std_logic | |
| i2c_ina_busy_w | std_logic | |
| i2c_ina_write_w | std_logic | |
| i2c_ina_read_w | std_logic | |
| i2c_ina_rdata_w | std_logic_vector(7 downto 0) | |
| i2c_ina_wdata_w | std_logic_vector(7 downto 0) | |
| i2c_sda_o_w | std_logic | |
| i2c_sdat_w | std_logic | |
| i2c_write_w | std_logic | |
| i2c_read_w | std_logic | |
| i2c_done_w | std_logic | |
| i2c_busy_w | std_logic | |
| i2c_ackn_w | std_logic | |
| i2c_addr_w | std_logic_vector(6 downto 0) | |

| Name | Type | Description |
|---|---|---|
| i2c_wdata_w | std_logic_vector(7 downto 0) | |
| i2c_rdata_w | std_logic_vector(7 downto 0) | |

*Table 40. I2C Controller signals.*

Constants

| Name | Type | Value | Description |
|---|---|---|---|
| IDLE | std_logic_vector(3 downto 0) | x"0" | |
| CFG_TEMP | std_logic_vector(3 downto 0) | x"1" | |
| READ_TEMP | std_logic_vector(3 downto 0) | x"2" | |
| CFG_GPIO | std_logic_vector(3 downto 0) | x"3" | |
| WRITE_GPIO | std_logic_vector(3 downto 0) | x"4" | |
| READ_GPIO | std_logic_vector(3 downto 0) | x"5" | |
| CFG_INA | std_logic_vector(3 downto 0) | x"6" | |
| READ_INA | std_logic_vector(3 downto 0) | x"7" | |

*Table 41. I2C Controller constants.*

Processes

- current_p: ( clk_i, rstn_i )
- next_p: ( all )

Instantiations

- i2c_u: work.i2c
- i2c_tmp100_u: work.i2c_tmp100
- i2c_gpio_u: work.i2c_gpio
- i2c_ina_u: work.i2c_ina

State machines

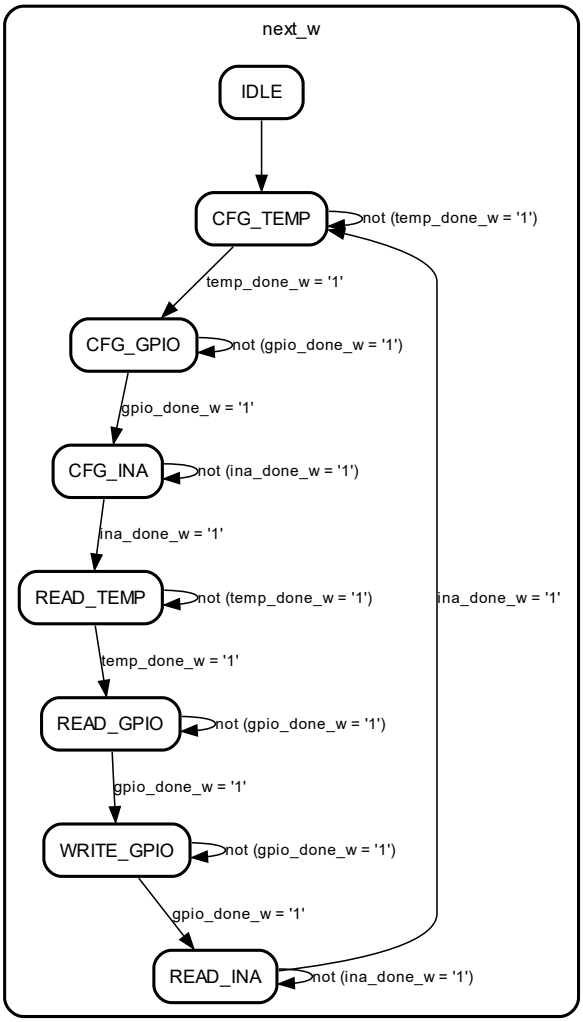



*Figure 102. I2C Controller FSM.*

### Timestamp

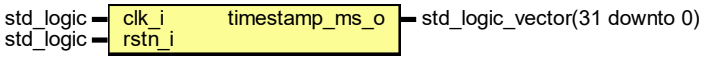



*Figure 103. Timestamp block diagram.*

Ports

| Port name | Direction | Type | Description |
|---|---|---|---|
| clk_i | in | std_logic | |
| rstn_i | in | std_logic | |
| timestamp_ms_o | out | std_logic_vector(31 downto 0) | |

*Figure 104. Timestamp ports descriptions.*

Signals

| Name | Type | Description |
|---|---|---|
| tick_r | std_logic_vector(15 downto 0) | |
| milisecond_r | std_logic_vector(31 downto 0) | |

*Figure 105. Timestamp signals.*

Constants

| Name | Type | Value | Description |
|---|---|---|---|
| CLK_FREQUENCY | integer | 50000000 | |
| TICK_COUNTER | std_logic_vector(15 downto 0) | x"C34F" | |

*Figure 106. Timestamp constants.*

Processes

- count_up_p: ( clk_i, rstn_i )