



BERGISCHE UNIVERSITÄT WUPPERTAL
FAKULTÄT FÜR ELEKTROTECHNIK,
INFORMATIONSTECHNIK UND MEDIENTECHNIK

A Highly Scalable Control Interface for the ATLAS Detector Control System at the High-Luminosity LHC

Dissertation

vorgelegt von Dominic Ecker

Erstgutachter: Prof. Dr.-Ing. Dietmar Tutsch

Zweitgutachter: Prof. Dr. Christian Zeitnitz

Abstract

The upcoming upgrade of the Large Hadron Collider (LHC) as part of the High-Luminosity LHC (HL-LHC) project will extend the accelerator’s potential and operational lifetime. The upgrade aims to significantly increase the number of collisions, thereby enhancing the rate of rare interactions and thus the discovery potential. To cope with this increased interaction rate and the associated increase in radiation within the detector cavern, extensive upgrades to the detectors and their infrastructure are required. These include new detector systems with state-of-the-art instrumentation and technologies capable of handling both the increased radiation and the higher event rate.

The ATLAS Detector Control System (DCS) is responsible for the safe and coherent operation of the detector. To meet the requirements of the High-Luminosity Large Hadron Collider (HL-LHC) environment, a scalable monitoring and control solution was developed to address associated challenges, including the increased radiation environment. The EMCI-EMP DCS interface is a complementary system based on two dedicated hardware platforms. The Embedded Monitoring and Control Interface (EMCI) serves as a radiation-tolerant interface for control and monitoring data of the detector Front-End electronics. The Embedded Monitoring Processor (EMP) is a Multi-Processing System-on-Chip (MPSoC)-based platform that operates as the high-performance Back-End, connecting up to 12 EMCIs via high-speed optical links to the supervisory layer of the DCS. The accompanying software and firmware ecosystem provides a modular and standardized framework that streamlines the deployment of EMCI-EMP systems across diverse applications.

This thesis presents the EMCI-EMP DCS interface and its firmware/software ecosystem. It further introduces a scalable Slow-Control Direct Memory Access (SC-DMA) framework for CPU-efficient slow-control data exchange, achieving an aggregate throughput exceeding 650 Mbit/s and demonstrating stable operation with up to 12 concurrently operating EMCIs.

Contents

1. Introduction	1
1.1. Problem Statement and Motivation	2
1.2. Thesis Outline	3
2. The ATLAS Detector at the Large Hadron Collider	5
2.1. The Large Hadron Collider	5
2.2. The ATLAS Detector	7
2.2.1. Magnet System	8
2.2.2. Inner Detector	9
2.2.3. Calorimeter	11
2.2.4. Muon Spectrometer	13
2.3. The High-Luminosity LHC Project	14
3. Radiation Environment and Electronics at the HL-LHC	17
3.1. Radiation Effects on Electronic Components	17
3.1.1. Radiation Types and Impact	18
3.1.2. Dose Effects	18
3.1.3. Single-Event Effects	19
3.1.4. Radiation Mitigation Strategies in HL-LHC Electronics	20
3.2. ATLAS Radiation Environment	22
3.2.1. ATLAS Background Radiation	22
3.2.2. Projection to the ATLAS HL Radiation Conditions	23
3.2.3. Conclusion	26
4. The ATLAS Detector Control System	27
4.1. Requirements	27
4.2. System Design	28
4.3. Standard Middleware OPC UA	30
4.4. Embedded Local Monitoring Board	31
4.4.1. Hardware	32
4.4.2. Firmware	33
4.4.3. Radiation Qualification	34
4.5. DCS Requirements for HL-LHC	35
4.5.1. ELMB2	36
4.5.2. EMCI-EMP	36

5. The EMCI-EMP DCS Interface	38
5.1. EMCI-EMP Interface Overview	38
5.1.1. System Architecture	38
5.1.2. Functional Scope of the EMCI-EMP Interface	40
5.1.3. EMCI-EMP Target Applications	41
5.2. EMCI: Design and Technical Specification	42
5.2.1. Hardware Architecture	43
5.2.2. Radiation Tolerance and Operational Requirements	45
5.2.3. The Low Power GigaBit Transceiver	45
5.3. EMP: Design and Technical Specification	52
5.3.1. TE0807 MPSoC Module	52
5.3.2. EMP Baseboard	55
5.4. EMP Chassis	60
6. The EMCI-EMP Ecosystem	64
6.1. Architectural Overview	64
6.2. Fundamental Concepts and Technologies	66
6.2.1. The Advanced eXtensible Interface	67
6.2.2. The Userspace I/O System	74
6.2.3. The Quick OPC UA Server Generation Framework	76
6.3. The EMP FPGA Firmware Reference Project	77
6.3.1. The emp-lpgbt IP-core	79
6.3.2. The FW-Debughub IP-core	82
6.3.3. The Generic EMP-MGT IP-core	83
6.3.4. Clock-Domain Architecture	84
6.4. epos - The EMP Operating System	86
6.5. Software	87
6.5.1. The LpGbtSw Library	88
6.5.2. empTools	89
6.5.3. Additional Software Utilities	91
6.5.4. OPC UA Servers	93
6.6. Contributions	95
7. Benchmark and Characterization Study	96
7.1. Test Environment	96
7.1.1. lpGBT Configuration	97
7.1.2. The eLink Interface IP-core	99
7.1.3. Software	100
7.2. Testing Procedure	100
7.3. Benchmark Results	102
7.3.1. Programmed Delay	103
7.3.2. Zero Inter-Transaction Delay	104

7.3.3. Stability Test	107
7.4. Conclusion	107
8. The Slow-Control DMA Subsystem	110
8.1. Design Requirements	111
8.2. Architectural and Functional Overview	112
8.2.1. SC-DMA Customization GUI	114
8.2.2. Configuration Interface	115
8.2.3. Watchdog	117
8.2.4. AXI4-Read Engine	118
8.2.5. Downlink Frame Assembler	120
8.2.6. Uplink Data Collector	124
8.2.7. AXI4-Write Engine	127
8.3. Testbench	129
8.4. Scalability Analysis	132
8.4.1. Frame Assembler: Resource and Timing Scaling	132
8.4.2. Frame Collector: Resource and Timing Scaling	133
8.4.3. System-Level Resource Validation	134
8.5. Kernel Driver	135
8.5.1. Character Device Interface	136
8.5.2. IOCTL Interface	136
8.5.3. Interrupt Handling and Synchronization	137
8.6. Conclusion	137
9. Characterization and Benchmarking	139
9.1. Test Hardware and SC-DMA System Integration	139
9.2. Test Application	141
9.3. Benchmark Results	143
9.3.1. Programmed Delay	143
9.3.2. Zero Inter-Transaction Delay	150
9.3.3. Long-Term Stability Test	152
9.4. Conclusion	153
10. Summary and Conclusions	155
Bibliography	159
List of Figures	177
List of Tables	186
Acknowledgements	187

Appendices	189
A. Thermal study	190
A.1. Introduction	190
A.2. Requirements for continuous FireFly Operation	190
A.3. Test Setup and Test Environment	191
A.3.1. Production Rack	191
A.3.2. EMP Chassis	192
A.4. Methods of Temperature Measurement	194
A.5. Module Utilization	195
A.5.1. PL Utilization	195
A.5.2. PS Utilization	196
A.5.3. Power Consumption	196
A.6. Results of the Thermal Tests	196
A.6.1. Conclusions	201
A.6.2. Re-validation with Pre-Production EMP and Crate	202
B. quasar Design Diagram	203
B.1. EMP Monitoring & Control OPC UA Server	203
C. SC-DMA IP Core - Finite-State Machines	204
C.1. Watchdog FSM	204
C.2. AXI4-Read Engine FSM	207
C.3. Downlink Frame Assembler	209
C.4. AXI4-Write Engine FSM	212
C.5. Uplink Frame Collector	215
D. SC-DMA Linux Kernel Module	220
D.1. Character Device Interface	220
D.1.1. <code>open()</code>	220
D.1.2. <code>mmap()</code>	221
D.1.3. <code>release()</code>	221
D.2. <code>ioctl</code> Interface	222
D.3. Interrupt Handler	223

1. Introduction

The European Organization for Nuclear Research (Organisation européenne pour la recherche nucléaire), commonly known as CERN, was founded in 1954 to establish a world-class research facility dedicated to fundamental particle physics. Conceived as a collaborative initiative among European nations, CERN's primary mission is to advance understanding of the universe by investigating its most basic building blocks and the forces that govern their interactions.

To this end, CERN operates the world's largest and most sophisticated physics laboratory. It is home to a network of state-of-the-art particle accelerators, including the Large Hadron Collider (LHC), the world's highest-energy particle accelerator. The LHC is a circular accelerator complex with two counter-rotating beam pipes that accelerate bunches of protons to nearly the speed of light before bringing them into collisions at center-of-mass energies of up to 13.8 TeV. These collisions are located at four dedicated locations around the ring, the so-called Interaction Points (IPs), corresponding to the positions of the four particle detectors: ALICE, ATLAS, CMS, and LhCB. These detectors facilitate the recording and reconstruction of particles produced in collisions, allowing physicists to investigate the fundamental forces of nature and the structure of matter.

The experiments have played a pivotal role in modern physics, most notably through the discovery of the Higgs boson. Beyond that, CERN's research and developments have been applied across many areas of society, including cancer research, medical imaging, and the development of the World Wide Web. Today, over 17500 people from more than 110 nationalities and institutes in over 70 countries are involved in the design, construction, and operation of the research infrastructure, the experiments, and the analysis of the resulting physics data.

To maintain CERN's position as a world-class research facility over the coming decade and to extend the lifespan of the LHC, a large-scale upgrade of both the accelerator complex and the experiments is underway. The HL-LHC project aims to enhance the LHC's performance by increasing the integrated luminosity, i.e., the number of collisions produced per bunch crossing, thereby enabling the experiments to observe rare processes with higher statistical precision. However, the resulting increase in particle collisions also leads to higher occupancies and radiation levels around the IPs, creating a harsh environment for the detectors and the on-detector readout and control electronics. To cope with these challenges, the experiments must upgrade their tracking systems, Front-End (FE) electronics, and detector control and data acquisition infrastructure to ensure reliable operation under the more demanding HL-LHC conditions.

1.1. Problem Statement and Motivation

The Detector Control System (DCS) and the Data Acquisition and Trigger System (TDAQ) constitute the operational backbone of the ATLAS detector, a 45 m in length and 25 m-diameter apparatus with a total mass of about 7000 t. The DCS ensures the safe and coherent operation of the detector and provides a homogeneous interface to all detector subsystems and their infrastructure. It enables uninterrupted monitoring of the detector, control of its operational state, and state transitions at any time. Operational fault conditions and errors are detected and reported through the DCS, and automated or manual actions are taken to rectify them. Notably, the DCS monitors and controls more than 10^6 parameters across the detector and its infrastructure.

In view of the HL-LHC upgrade, the ATLAS DCS must maintain system integrity under significantly increased radiation levels. The integration of entirely new detector systems, such as the Inner Tracker (ITk), further reinforces these demands by requiring a scalable, high I/O-density DCS interface. Together, these factors impose stringent requirements on the DCS hardware in terms of radiation tolerance, scalability, and channel density. To this end, a novel generic DCS interface has been developed based on two custom platforms:

- The Embedded Monitoring and Control Interface, a radiation-tolerant FE interface which interfaces directly with the on-detector electronics.
- The EMP, a System-on-Module (SoM)-based compact processing platform that serves as the off-detector readout device for multiple EMCIs and connects to the distributed DCS Back-End (BE).

Together, the EMCI and the eEMP implement a flexible, subsystem-agnostic DCS interface that can be adapted to various on-detector electronics while providing a well-defined interface.

While the development of the EMCI and EMP addresses hardware-related requirements, the novel system introduces a broad set of embedded system challenges that require coherent, structured solutions. These include, among others, a robust runtime environment for the EMP Processing System (PS) and dedicated software/firmware blocks for generic system tasks. These efforts resulted in the development of the EMCI-EMP ecosystem, a comprehensive framework designed to address typical needs and requirements, enabling rapid, standardized adoption of the novel DCS interface across various on-detector FE electronics.

A central prerequisite for the ecosystem is a scalable, CPU-efficient data path for accessing FE monitoring and control functionality. Following an analysis of a conventional register-based access mechanism, fundamental limitations in throughput and

scalability were identified. Therefore, to ensure the system's scalability and fully exploit the capabilities of the EMCI-EMP platform, a custom, high-throughput, CPU-efficient solution for Direct Memory Access (DMA) based FE communication is presented.

The motivation of this work is therefore twofold. Firstly, there is a need for a radiation-tolerant, scalable, and generic DCS FE interface capable of meeting the harsh requirements of the HL-LHC environment. Secondly, the development of a cohesive firmware and software ecosystem around the EMCI-EMP DCS interface is essential. This ecosystem provides standard software and firmware building blocks and a scalable, high-throughput data path for heterogeneous FE devices, thereby ensuring a robust, maintainable, and standardized system deployment within the ATLAS DCS.

1.2. Thesis Outline

In the first part of the thesis, the LHC and the ATLAS detector are introduced in detail, including their various subsystems. Then, an overview of the HL-LHC project is provided, detailing its motivation, key challenges, and the planned operational schedule. The challenges posed by radiation to electronic components are discussed next, focusing on the most relevant effects and underlying physical mechanisms. Building on this, the theoretical mechanisms by which radiation is produced and distributed during high-energy collisions are discussed, and the correlation between instantaneous and integrated luminosity and the exposed radiation is derived from radiation-field measurements from previous LHC runs. Using this correlation and simulation results, the radiation fields expected during the HL-LHC era are used to infer constraints on on-detector electronics.

Before addressing the impact of the HL-LHC upgrade on the ATLAS DCS, the system itself is first introduced. An overview of its scope, architecture, and the currently used FE hardware, the Embedded Local Monitoring Board (ELMB) is provided. Then, the radiation tolerance of this FE hardware is re-evaluated against the expected HL-LHC radiation levels, highlighting the need for a future-proof solution and motivating the EMCI-EMPDCS interface.

On this basis, the new EMCI and EMP hardware platforms and the overall system architecture are discussed, followed by an introduction of the accompanying firmware and software ecosystem. This common framework solution provides a broad range of standard software/firmware building blocks developed to satisfy common requirements for all foreseen use cases. Following this, the initial benchmark study,

which uses a register-map-based mechanism to access FE-related control and monitoring data, is presented, highlighting scalability and throughput limitations and outlining the motivation for a dedicated solution beyond commercial Intellectual Property Cores (IP-cores). This leads to the presentation of the Slow-Control Direct Memory Access (SC-DMA) subsystem for the EMCI-EMP architecture, a fully custom DMA solution tailored for the specific requirements of the novel DCS interface. The discussion of this subsystem is followed by a dedicated characterization and benchmark study that assesses its scalability and performance under representative conditions. The thesis concludes with a summary of the work and a discussion of the relevance of the EMCI-EMP as the next-generation DCS platform for the HL-LHC era.

2. The ATLAS Detector at the Large Hadron Collider

The scientific field of particle physics seeks to understand the fundamental building blocks of matter and the forces that govern their interactions. To investigate at the smallest scales, particle accelerators accelerate subatomic particles to high energies and collide them. These high-energy collisions recreate conditions similar to those immediately after the Big Bang, allowing researchers to study rare processes and discover previously unknown particles. The LHC accelerates beams of hadrons, particles composed of two or more quarks, bound together by the strong nuclear force, to nearly the speed of light and collides them. Through these collisions, the LHC aims to explore the fundamental forces of the universe by observing the particles and interactions they produce. This includes studying the Higgs boson, testing the known Standard Model of Particle Physics, and searching for signs of physics beyond the Standard Model, such as dark matter, supersymmetric particles, and extra dimensions [1].

This chapter provides an overview of the LHC and the ATLAS detector, one of the four experiments at the LHC. The ATLAS detector is a general-purpose detector designed to observe a wide range of particles using a complex system of subdetectors to reconstruct and analyze the particles produced in collisions.

2.1. The Large Hadron Collider

The LHC [2] is the world's largest and most powerful circular particle accelerator and collider, with a circumference of 27 km and located 100 m underground. Constructed between 2000 and 2008 within the existing tunnel system of the Large Electron-Positron (LEP) collider, the LHC went into operation in 2009. Its two-ring design for counter-rotating particle beams enables proton-proton collisions with a center-of-mass energy of 13.8 TeV and lead-lead collisions with a center-of-mass energy of 1150 TeV.

The existing tunnel, constructed already between 1983 and 1988 to house the LEP, has an internal regular diameter of only 3.7 m. This restricted cross-sectional area prevented the installation of two separate accelerator rings. Consequently, the LHC had to integrate both counter-rotating beam pipes within a single magnetic structure occupying the same ring [4]. This magnetic structure consists of a dense array of superconducting magnets arranged in a lattice that ensures beam alignment and

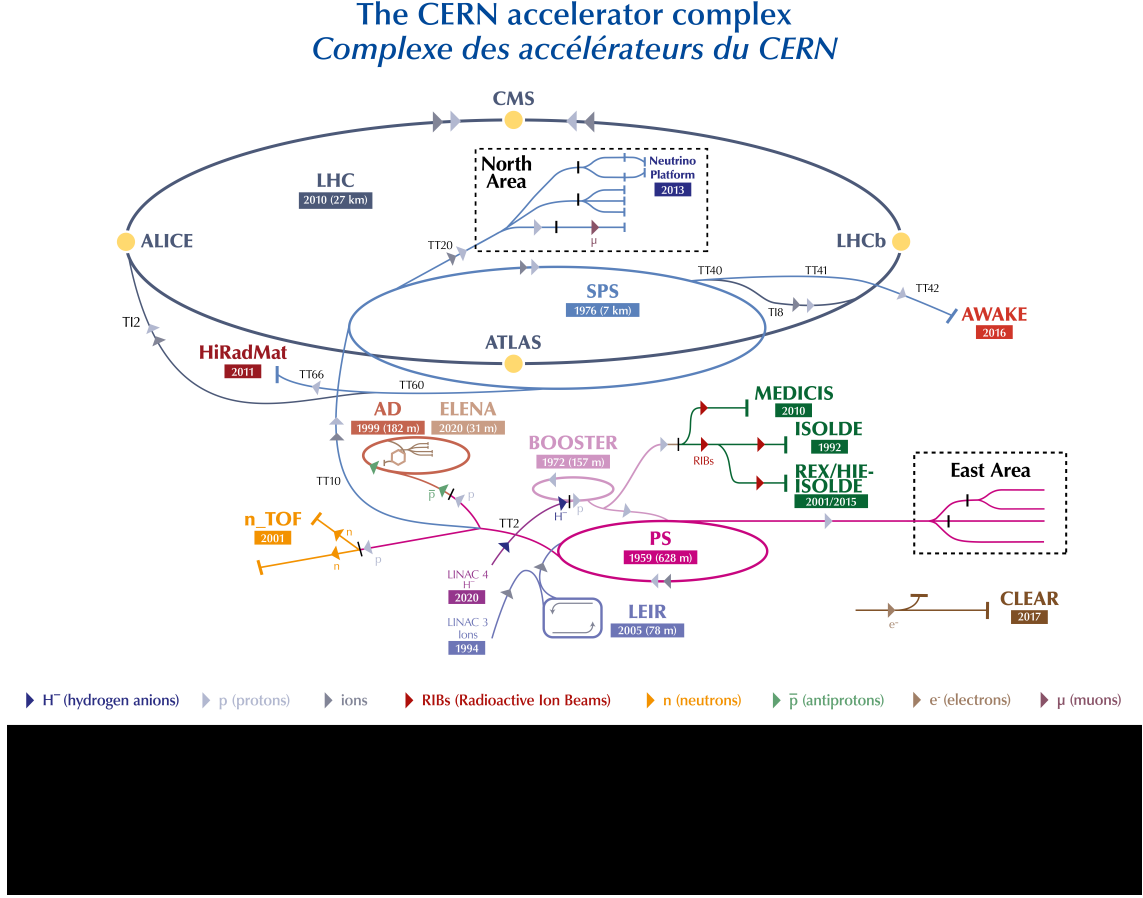


Figure 2.1: Overview of CERN’s accelerator complex and the different experiments [3].

stability throughout the accelerator’s 27 km circumference. A total of 1232 superconducting dipole magnets, each 15 m in length, are employed to bend the trajectories of the proton beams along their circular paths. These are supplemented by higher-order multipole magnets, including quadrupoles, sextupoles, octupoles, and decapoles, which are responsible for focusing and correcting the beam trajectories in both transverse planes.

Before particles are injected into the LHC, they pass through a multi-stage pre-acceleration chain, illustrated in Figure 2.1, using CERN’s network of accelerator facilities. The process is initiated with a hydrogen gas source, where protons are extracted and initially accelerated by the Linear accelerator 4 (Linac4) to 160 MeV. They are then sent to the Proton Synchrotron Booster (PSB), which increases their energy to 2 GeV, followed by the Proton Synchrotron, which augments their energy to 26 GeV. In the ensuing stage, the Super Proton Synchrotron (SPS) ac-

celerates the protons to 450 GeV, after which they are then alternately injected in bunches of up to 1.8×10^{11} protons into the counter-rotating beam pipes of the LHC.

Inside the LHC, these proton bunches are then accelerated to their final energy of 6.9 TeV using superconducting radio-frequency (RF) cavities [5]. These are metallic resonant structures that generate an oscillating electromagnetic field at 400 MHz, which transfers energy to the particles as they pass through. The RF system is powered by high-power, intensity-modulated klystrons, enabling precise control over the accelerating field. Synchronization between the RF field and the circulating bunches ensures efficient acceleration while maintaining longitudinal bunch compression and stability.

The two counter-propagating beam pipes intersect at four locations around the accelerator ring. At these IPs, the beams are brought into collision at the centers of the four particle detectors: ATLAS [6], CMS [7], ALICE [8], and LHCb [9]. The following presents a comprehensive description of the ATLAS detector and its various subsystems.

2.2. The ATLAS Detector

The ATLAS detector is one of two general-purpose detectors at the LHC and is the largest-volume particle detector ever built. While both CMS and ATLAS pursue similar scientific objectives, such as the exploration of new particles and the study of fundamental interactions, they rely on different technologies and magnet systems to achieve these goals.

The ATLAS detector features a toroidal geometry, measuring 46 m in length and 25 m in diameter, with a forward-backward-symmetric design about the IP. The detector is capable of recording data across a wide range of physics, from the precision measurement of the Higgs boson to searches for signatures of extra dimensions and particles that could comprise dark matter. To this end, the detector incorporates various subsystems, as illustrated in Fig. 2.2, which are arranged in concentric layers that surround the beam pipe. In the following subsections, each subsystem, including the magnet system, the Inner Detector (ID), the calorimeters, and the muon spectrometer, is detailed, with focus on their functionality and roles within the detector architecture.

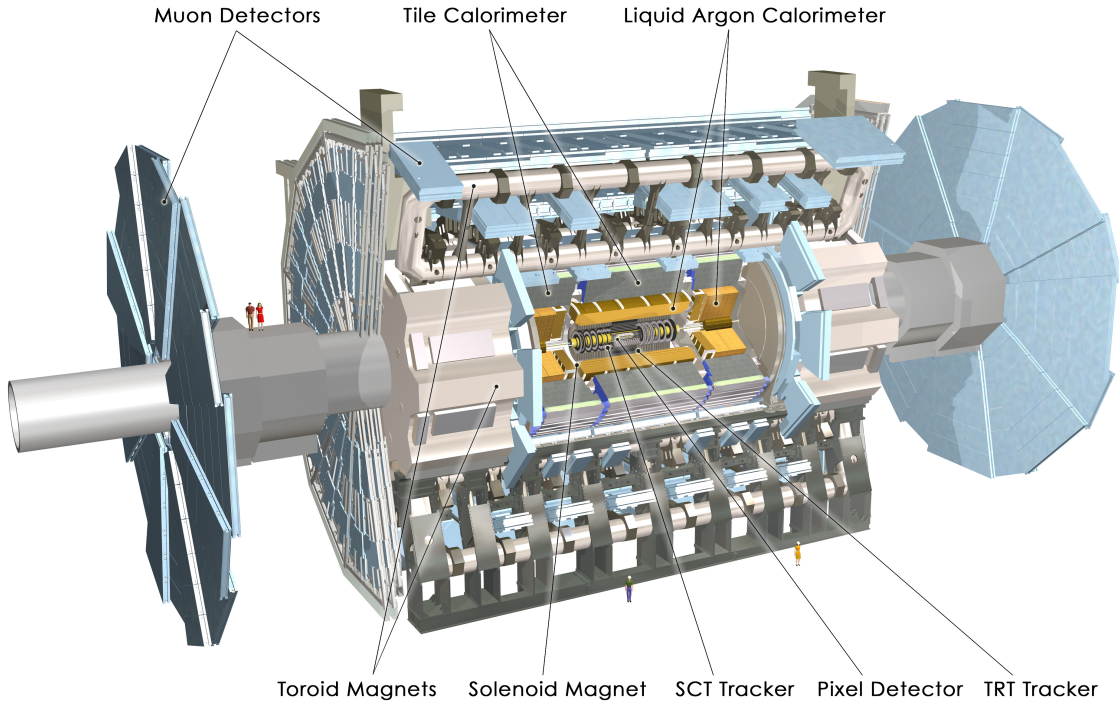


Figure 2.2: Overview of the ATLAS detector and its toroidally arranged sub-detectors and magnet systems [10].

2.2.1. Magnet System

The ATLAS detector incorporates two distinct superconducting magnet systems: the Central Solenoid Magnet and the Toroid Magnet [11]. These systems enable the measurement of a particle's momentum and charge by bending its trajectory as it traverses the detector. To generate the required magnetic fields, both magnet systems are cooled to approximately 4.5 K, ensuring that the superconducting materials operate with negligible electrical resistance. This allows for the stable generation of powerful magnetic fields, sufficient to bend the trajectories of particles produced at high energies. The resulting magnetic-field configuration, including the field lines of both magnet systems, is illustrated in Figure 2.3.

Central Solenoid Magnet The central solenoid magnet [13] surrounds the ID at the core of the experiment and generates a 2 T axial magnetic field, which is aligned parallel to the beam pipe. The field is generated by more than 9 km of niobium-titanium superconductor wires embedded into strengthened, pure aluminum strips. The magnet's compact design, with a total thickness of 4.5 cm, minimizes material

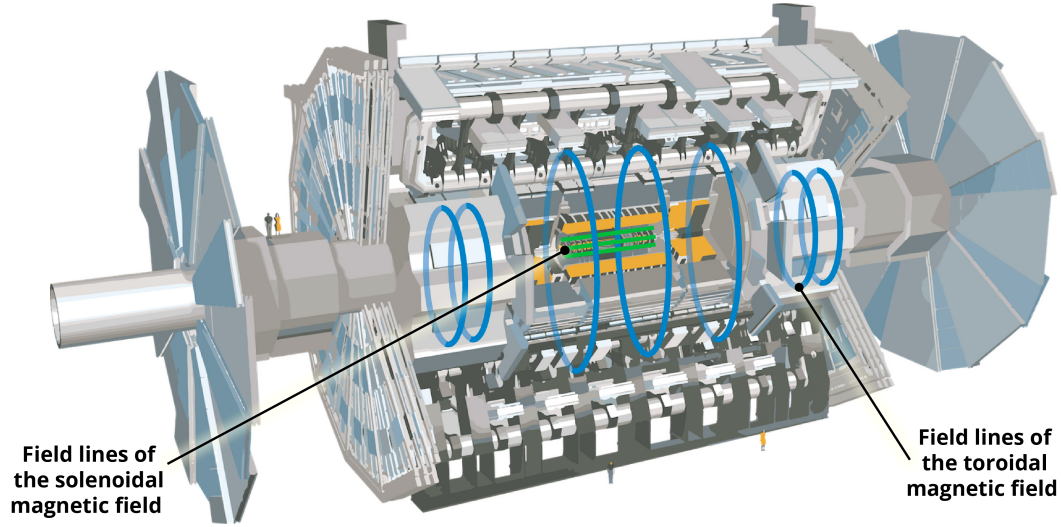


Figure 2.3: *Illustration of the magnetic fields and their respective field lines within the ATLAS detector [12].*

interference with the particles being studied, thereby reducing unwanted interactions and enhancing data quality.

Toroid Magnet The ATLAS toroid magnet system comprises an eight-coil barrel toroid that surrounds the central region of the experiment, and two end-cap toroids on either side of the detector to extend the magnetic coverage to particles leaving the detector. With a total length of 25.3 m, an outer diameter of 20.1 m, and a mass of approximately 830 t, the barrel toroid is the largest toroidal magnet ever constructed. The superconducting system generates a toroidal magnetic field of up to 3.5 T, with field lines looping azimuthally around the beam axis. It ensures the deflection of muons (heavy electrons) over a large detector volume, thereby enabling precise momentum and charge measurements at high energies.

2.2.2. Inner Detector

The ID [14] constitutes the innermost layer of the ATLAS detector. As illustrated in Fig. 2.4, it consists of three complementary sensor systems: the Pixel Detector, the Semiconductor Tracker (SCT), and the Transition Radiation Tracker (TRT). The high sensitivity of these systems enables precise measurements of the electrical

charge, direction, momentum, and trajectories of electrically charged particles produced in proton-proton collisions. Although it remains a compact detector system, measuring just 6.2 m in length with a circumference of only 2.1 m.

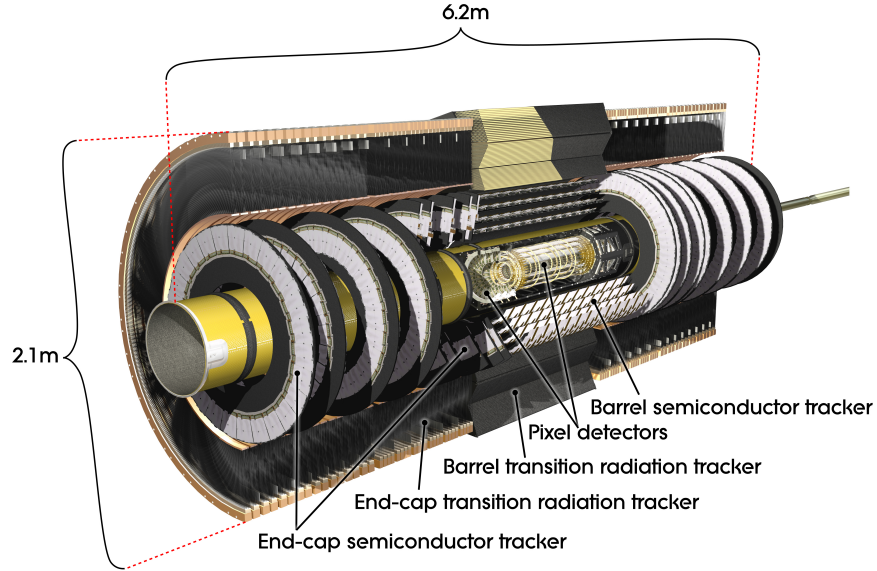


Figure 2.4: Overview of the Inner Detector, showing its three constituting subsystems: the Pixel, SCT, and TRT detector [15].

Pixel Detector The Pixel detector [16], located just 3.3 cm from the LHC beam axis, functions as the first tracking layer for any charged particle that is produced during collisions. The detector comprises four barrel layers and three end-cap disks on each side, integrating a total of 2,024 modules with approximately 92 million $n^+ - in - n$ sensors, commonly referred to as pixels [17]. The standard pixel size is measured at $50\,\mu\text{m} \times 400\,\mu\text{m}$, while the pixels of the Insertable B-Layer (IBL), the innermost Pixel layer, measure at $50\,\mu\text{m} \times 250\,\mu\text{m}$, thus providing a higher granularity. The passage of a charged particle through the silicon sensor results in the deposition of energy, which is detected with a position resolution of nearly $10\,\mu\text{m}$.

Semiconductor Tracker The SCT [18] surrounds the Pixel detector and comprises four cylindrical barrel layers and 18 planar end-cap disks. It is constructed from 4,088 double-sided silicon micro-strip modules, incorporating more than 6 million readout strips. In total, the SCT provides a silicon surface coverage of approximately

60 m². The detector's geometry and high channel density ensure that each charged particle traverses at least four silicon layers, enabling trajectory reconstruction with a spatial resolution of up to 25 μm .

Transition Radiation Tracker The TRT [19] constitutes the third and outermost sensor system of the ID. It consists of 300,000 thin-walled drift tubes, referred to as "straws", which are arranged in a cylindrical geometry around the beam axis. Each straw measures 4 mm in diameter and contains a gas mixture with a 30 μm gold-plated tungsten wire positioned at its center. The collision of particles with the gas results in ionization, thereby creating an electrically detectable signal. The TRT operates in conjunction with a dense radiator material interleaved between the straws. This material induces transition radiation when high-energy electrons traverse boundaries between materials with different dielectric constants. The radiation is absorbed in the gas, producing additional ionization and enabling the distinction between electrons and heavier particles [14].

2.2.3. Calorimeter

In particle-physics detectors, calorimeters [20] are designed to measure particle energy by requiring them to deposit their full energy. This can be interpreted as the particle coming to a halt. Upon entering the calorimeter, a high-energy particle interacts with the detector material, producing a cascade of secondary particles. This process is referred to as a shower. The energy of the original particle is then reconstructed from the collected signal generated by these interactions. Calorimeters are generally classified into two primary categories: electromagnetic calorimeters, which are optimized for detecting electrons and photons, and hadronic calorimeters, which are designed to measure the energy of hadrons, particles that contain quarks, such as protons and neutrons. The radial and longitudinal arrangement of the different calorimeter systems within the ATLAS detector, surrounding the ID, is shown in Figure 2.5.

Liquid Argon Calorimeter The Liquid Argon (LAr) Calorimeter [22] surrounds the ID and is designed to measure the energy of electrons, photons, and hadrons. It incorporates alternating layers of dense absorber materials, such as lead, copper, or tungsten, which are interleaved with liquid argon. When traversed by particle showers, the argon becomes ionized, and the resulting ionization current is collected and measured to reconstruct the energy of the impacting particle. To maintain the

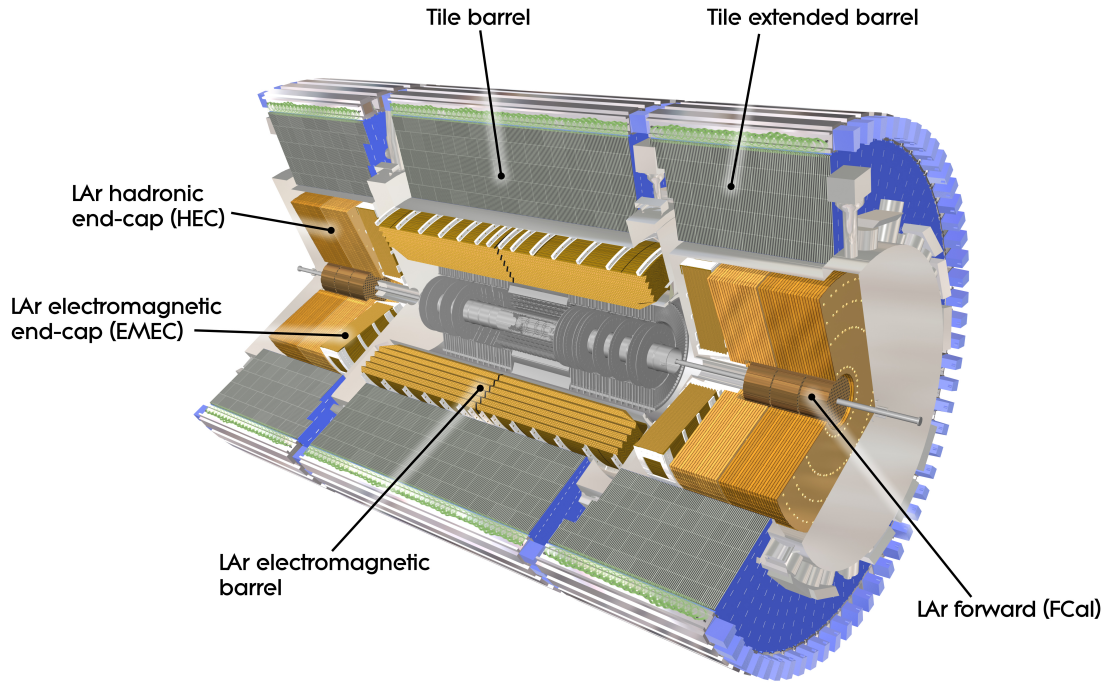


Figure 2.5: Overview of the location of the ATLAS calorimeter and its subsystems [21].

argon in its liquid state, the system is cooled to approximately -184°C . Vacuum-insulated feedthroughs ensure reliable signal transmission to the external readout electronics.

Furthermore, the calorimeter’s central part features an accordion-like configuration, designed to optimize the efficacy of electron and photon identification. This specialized geometry folds absorber plates and readout electrodes in a wave-like pattern, thereby closing gaps between modules and ensuring a hermetic seal. In addition to its electromagnetic barrel and end-cap layer, the LAr hadronic end-cap serves as the hadronic calorimeter in the forward region, as the Tile Calorimeter does not provide coverage there.

Tile Hadronic Calorimeter The Tile Calorimeter [23], which surrounds the LAr Calorimeter, is designed to measure the energy of hadronic particles that do not deposit their full energy in the LAr Calorimeter. These particles include protons and neutrons, in particular. The calorimeter comprises alternating layers of steel absorber plates and plastic scintillator tiles. When hadrons traverse steel, they generate particle showers. These subsequent secondary particles excite the scintillator

material, thereby producing photons. Wavelength-shifting fibers collect this light and transmit it to photomultiplier tubes, where it is converted into an electrical signal.

2.2.4. Muon Spectrometer

The ATLAS muon spectrometers [24] constitute the outermost layer of the detector, intending to determine the position and to measure the momentum of muons. A muon is a minimum-ionizing particle that is frequently characterized as a heavy electron due to its mass, which is approximately 200 times greater than that of an electron. Due to their substantial mass and minimal interaction with dense detector materials, muons traverse the ID and calorimeter system with minimal energy loss. The muon spectrometer facilitates this characteristic by tracking muons over a large volume and reconstructing their trajectories in the magnetic field to infer their momentum. To achieve both high spatial resolution and fast event selection, the muon system uses a combination of precise tracking detectors and fast-response trigger chambers, each optimized for specific muon-detection requirements. An overview of the coverage and layout of the ATLAS muon spectrometers is shown in Figure 2.6.

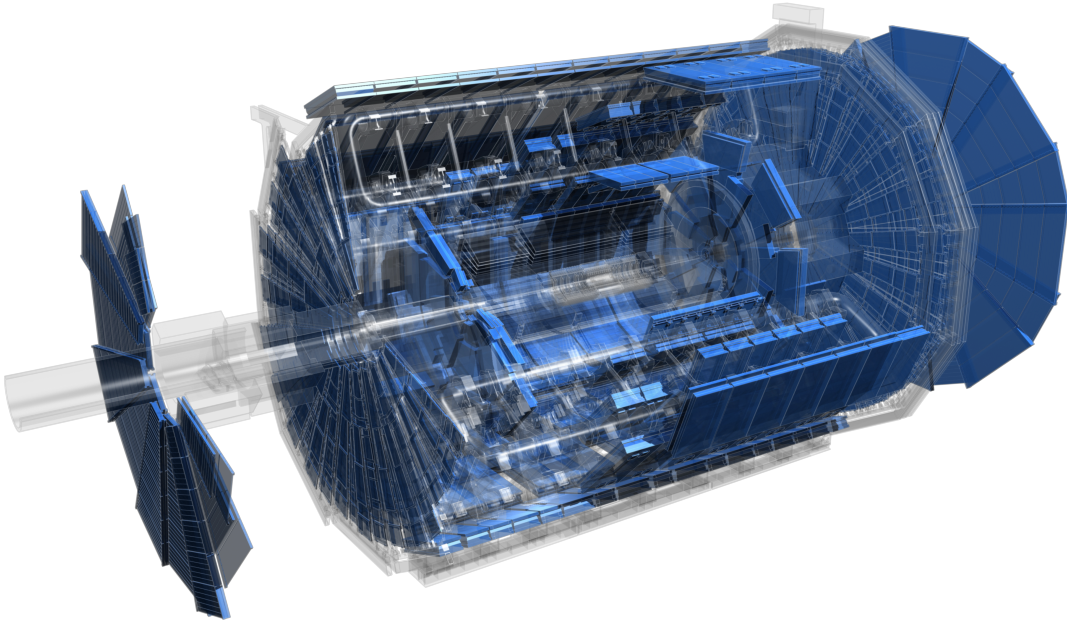


Figure 2.6: Computer-generated image detailing the areas of the ATLAS detector covered by the muon spectrometers [25].

Precision Detectors The precision detectors of the muon system, known as Monitored Drift Tubes (MDTs), have been designed to determine a muon’s position with a precision of 100 μm . The MDTs consist of aluminum tubes with a diameter of 3 cm that are filled with a gas mixture. As muons traverse the tubes, they ionize the gas, thereby releasing electrons that subsequently drift towards a central wire, where they induce a detectable signal. The total number of tubes exceeds 380,000, arranged in multiple layers to reconstruct each muon’s trajectory.

Fast-Response Detectors To select potentially interesting events in real-time, ATLAS employs fast-response muon detectors, including Resistive Plate Chambers (RPCs) in the barrel region and Thin Gap Chambers (TGCs) in the end-caps. These detectors respond within 2.5 μs by detecting ionization in gas volumes between electrodes or wire planes. Recent technological advancements, including Micromegas (MMGs) and Small-Strip Thin Gap Chambers (STGCs), have been installed during the most recent detector upgrade. These chambers are utilized in regions with high radiation levels near the beam pipe, offering both rapid response and high resolution.

2.3. The High-Luminosity LHC Project

The LHC remains the world’s most powerful particle accelerator at present, and this situation is expected to remain unaltered for at least two decades. Maximizing its scientific potential is therefore a top priority and is explicitly identified as a key objective in the European Strategy for Particle Physics. Consequently, efforts are underway to extend the LHC’s operational lifetime and enhance its capabilities through a substantial upgrade known as the High-Luminosity Large Hadron Collider (HL-LHC) project [26]. The present schedule of the remaining operations until the commencement of the Long Shutdown 3 (LS3), during which the upgrades will be installed, is presented in Fig. 2.7.

The primary objective of the HL-LHC project is to significantly increase the number of collisions delivered to the experiments, thereby enabling the collection of larger datasets. This is accomplished by increasing the instantaneous luminosity (L_{int}), a measure of the collision rate, which is expressed in units of inverse barns per second ($1/(\text{b s})$). The HL-LHC aims to achieve an instantaneous luminosity 5 to 7.5 times higher than that of the original LHC design. Reaching this ambitious goal required approximately a decade of intensive research and development, encompassing the design, testing, and integration of a wide range of advanced technologies. The HL-LHC machine configuration will therefore rely on several cutting-edge innovations, including:

2. The ATLAS Detector at the Large Hadron Collider

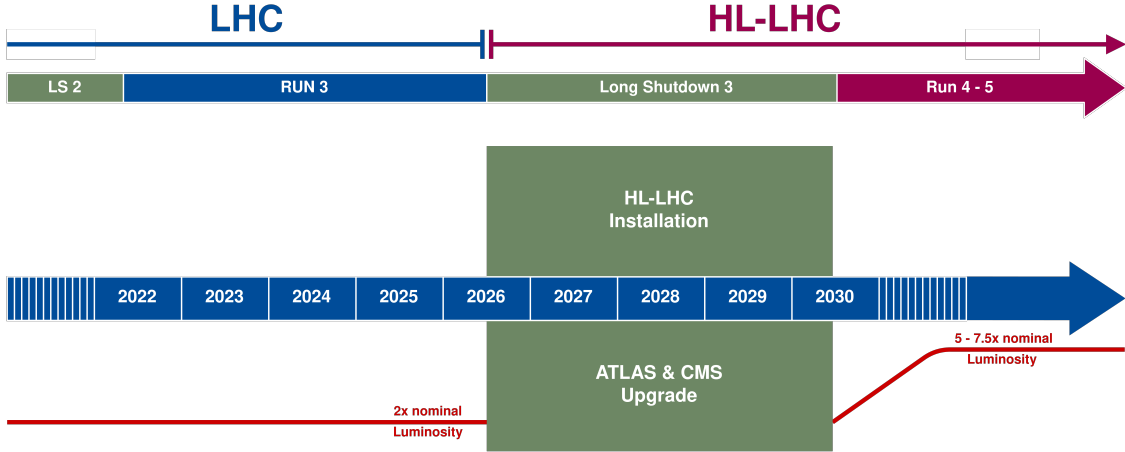


Figure 2.7: Timeline detailing the installation of the HL-LHC and ATLAS/CMS upgrades during the LS3 and the final start of operation (adapted from [27]).

- High-field quadrupoles to achieve the required beam focusing
- Compact superconducting RF cavities with ultra-precise phase control
- High-power superconducting links (to transmit energy with nearly zero loss)

The HL-LHC upgrade necessitates adapting the ATLAS detector to address novel challenges and fulfill emerging requirements, thereby capitalizing on the increased luminosity. Higher collision rates will result in more rare events and thus more collectible data, as well as a denser event environment. This necessitates not only collecting more data but also operating the detectors with greater precision to identify and differentiate overlapping particle interactions (pile-up). Furthermore, given that the radiation exposure within the detector scales approximately linearly with the luminosity, the requirements on electronic components in the experiment cavern are more demanding. Consequently, this requires redesigning or developing new sub-detectors and FE electronics to withstand the anticipated increase in radiation levels during HL-LHC operation.

To address these challenges, the ATLAS Phase-II Upgrade, scheduled for installation during LS3, involves a complete replacement of the ID and associated infrastructure with a novel, all-silicon tracker, known as the Inner Tracker (ITk) [28]. The system will feature extended angular coverage towards the forward regions, higher granularity, and improved radiation tolerance to cope with increased occupancies and long-term radiation damage. In addition, the ITk is designed for a significantly higher data-acceptance rate, enabling it to handle the increase in the read-out event rate from approximately 100 kHz to 1 MHz. In parallel, substantial upgrades

are planned for the calorimeter and muon systems, including the implementation of new FE, trigger, and readout electronics [29].

A critical aspect of upgrading the ATLAS detector for HL-LHC operation is to ensure the long-term reliability of the electronics and associated infrastructure under extreme conditions. To define the requirements for developing such electronics and to evaluate the current systems employed, it is essential to have a basic understanding of the expected radiation environment and its impact on the detector electronics. The subsequent chapter thus provides a comprehensive overview of the various types of radiation, their effects on electronics, and mitigation strategies commonly employed during the development process. Furthermore, the chapter will discuss the fundamental process by which radiation is generated within the detector and how it scales with luminosity, enabling projections and discussions of the conditions expected during HL-LHC operations.

3. Radiation Environment and Electronics at the HL-LHC

The upgrade of the LHC to the HL-LHC poses substantial challenges for both the accelerator infrastructure and the associated experiments. It demands the development of advanced technologies, including high-field superconducting magnets, ultraprecise RF cavities, and increasingly complex detector systems capable of processing data at rates ten times higher than those achieved today. A critical aspect common to all systems is the reliable operation of electronic components in a significantly harsher radiation environment.

Given the rising collision rate at the HL-LHC, a substantial increase in the radiation background within the ATLAS detector cavern is to be expected. The elevated radiation environment directly impacts the performance, reliability, and expected lifetime of both on-detector electronics and supporting infrastructure. As these components must remain operational throughout the HL-LHC's lifetime, they must meet stringent radiation-tolerance specifications. However, establishing these requirements necessitates a fundamental understanding of the impact effects themselves and the expected radiation environment.

This chapter introduces the primary radiation types that impact electronic components, together with the standard metrics used to quantify radiation tolerance and the commonly applied mitigation strategies to enhance robustness. A subsequent discussion explores the origins of radiation within the experimental cavern, highlighting the correlation between luminosity and radiation fluence. This correlation, in conjunction with appropriate simulations, facilitates the determination of requirements for radiation-tolerant electronics and infrastructure within ATLAS.

3.1. Radiation Effects on Electronic Components

The interaction of radiation with semiconductor devices depends on two factors: the type of particle and the mechanism of energy transfer. Furthermore, some effects manifest gradually due to cumulative radiation exposure, while others occur suddenly as a result of a single particle strike. A comprehensive understanding of these processes requires distinguishing between ionizing and non-ionizing radiation,

as they produce distinct forms of damage in electronic materials. This section introduces the primary categories of radiation and their physical effects on semiconductors, and then discusses the primary impact of radiation-induced effects. The section concludes with an outline of the mitigation strategies commonly used in the design of electronics for radiation-prone environments.

3.1.1. Radiation Types and Impact

The classification of radiation can be broadly defined by its ability to ionize matter, which in turn determines its interaction with semiconductor materials. This results in classifying radiation into two categories: ionizing and non-ionizing, each of which leads to different degradation effects in electronic components.

- **Ionizing radiation** is based on particles with sufficient energy to dislodge electrons from atoms, thereby generating electron-hole pairs in semiconductor materials. Common ionizing particles include alpha (α) and beta (β) particles, neutrons, protons, and high-energy photons such as gamma rays and X-rays. While these particles vary in mass, charge, and penetration depth, they share the ability to disrupt electronic structures directly and induce charge buildup in insulating or dielectric layers.
- **Non-ionizing radiation** comprises particles and electromagnetic waves that do not possess enough energy to ionize atoms. However, certain types, particularly low-energy neutrons and protons, can still transfer significant kinetic energy to atoms in a crystalline lattice, displacing them from their regular positions. This displacement leads to the formation of crystal defects that degrade the electrical properties of semiconductor devices over time.

3.1.2. Dose Effects

As mentioned, in addition to its direct interaction with matter, radiation can also cause cumulative changes over time that gradually degrade semiconductor performance. These effects, termed "Dose Effects", emerge from persistent radiation exposure and are typically classified into two categories: Total Ionizing Dose (TID), a surface effect, and Displacement Damage (DD), a volume effect.

- **Total Ionizing Dose:** Continuous exposure of electronic components to ionizing radiation induces parasitic changes in semiconductor materials. In MOSFET and bipolar devices, these changes primarily result from the generation, transport, and trapping of charge carriers, particularly holes, in insulating layers such as gate oxides. Additionally, sufficiently high doses can increase

leakage currents and cause a loss of isolation in CMOS circuits. The TID quantifies the accumulated energy deposited per unit of mass of material, serving as a metric for determining the threshold beyond which a device's performance degrades or fails. The International System of Units (SI) defines the unit of absorbed doses as gray (Gy), where $1 \text{ Gy} = 1 \text{ J/kg}$. However, in most specifications, and also throughout this thesis, the older unit rad is used. One rad corresponds to the absorption of 100 erg ($1 \text{ erg} = 100 \text{ nJ}$) per gram of matter, which is equivalent to 0.01 J/kg or 0.01 Gy [30].

- **Displacement Damage:** These damages are defined as cumulative, volumetric degradation caused by non-ionizing radiation, primarily from neutron and proton doses. These particles transfer momentum to the lattice atoms, displacing them from their positions and creating defects in the crystal structure. Such defects are characterized by their electrical activity, which introduces traps that reduce carrier lifetimes and impair charge transport properties. In BJTs, this results in increased base current and reduced current gain. In MOSFETs, significant damage can degrade carrier mobility, reducing drive strength and slowing switching speed. This damage is often described using the Non-Ionizing Energy Loss (NIEL) scaling hypothesis, which states that the damage caused by energetic particles is roughly proportional to the displacement damage cross section (D). While NIEL is typically expressed in units of $\text{keV} \times \text{cm}^2/\text{g}$, the displacement damage cross section D is often given in $\text{MeV} \cdot \text{mb}$. For silicon, the two are related by $100 \text{ MeV} \cdot \text{mb} \approx 2.144 \text{ keV} \cdot \text{cm}^2/\text{g}$ [30].

However, in the ATLAS context, displacement damage is commonly expressed as the equivalent fluence of 1 MeV neutrons in silicon. To account for the dependence of the displacement-damage cross section on particle type and energy, the contribution of each particle is weighted by a hardness factor, defined as the ratio of the corresponding displacement-damage cross section in silicon, $W_Q(E)$, to that of 1 MeV neutrons, $W_{Q,n}(1 \text{ MeV})$. The resulting fluence therefore corresponds to the fluence of 1 MeV neutrons producing the same average NIEL damage in silicon. While the NIEL of 1 MeV neutrons is defined as $95 \text{ MeV} \cdot \text{mb}$, this reference value must be determined indirectly from broader neutron spectra, introducing a systematic uncertainty in the absolute normalization of the neutron-equivalent scale [31].

3.1.3. Single-Event Effects

In contrast to cumulative damage, Single-Event Effects (SEEs) are defined as a random disruption in an electronic device caused by a single ionizing particle or

photon impacting the semiconductor. This impact can lead to transient malfunctions or even permanent device failure. Therefore, SEEs are generally categorized as either nondestructive or destructive mechanisms. Nondestructive (soft) SEEs result in transient errors, such as incorrect output values or bit flips, without causing permanent damage to the component. Conversely, destructive (hard) SEEs cause irreversible damage, ultimately leading to the component's failure [30]. In the ATLAS context, the SEE-related tolerance requirements are typically expressed through the flux of hadrons with energies above 20 MeV, $h_{E>20\text{ MeV}}$.

Below, the differences between the two categories of SEEs are summarized, along with their specific impact on electronic devices:

- **Single-Event Upset (SEU):** A SEU occurs when a high-energy particle strikes a sensitive region in a semiconductor device, depositing enough charge to flip the state of a memory cell or logic circuit. This transient change does not cause permanent damage but can corrupt data or alter program flow. The probability of such events is characterized by the SEU cross section (δ_{SEU}), which depends on particle fluence and energy. Advanced CMOS scaling increases SEU sensitivity due to reduced capacitance and charge collection volumes, resulting in an increased probability of Multi-Bit Upsets (MBUs) from a single strike [30].
- **Destructive SEEs Mechanisms:** Single Event Latch-up (SEL), Single Event Burnout (SEB), and Single Event Gate Rupture (SEGR) are classified as destructive SEEs, as they can cause permanent damage or functional failure in semiconductor devices. SEL occurs when a single charged particle induces a parasitic low-impedance path between power and ground in CMOS or Bipolar CMOS (BiCMOS) bulk technologies. This leads to a sustained high-current state that may persist even after the initial event and can result in thermal destruction. SEB and SEGR, in contrast, are typically triggered by heavy-ion or high-energy particle strikes in power devices such as MOSFETs or BJTs. An SEB resembles an SEL as it involves a critical increase in current. It occurs specifically in the drain region of a power device, where impact ionization and subsequent avalanche breakdown cause irreparable damage. SEGR occurs when a particle strike induces a localized electric field strong enough to rupture the gate oxide of a MOSFETs, causing a permanent short between the gate and the channel [30].

3.1.4. Radiation Mitigation Strategies in HL-LHC Electronics

Given the diverse ways in which radiation affects electronics, mitigation strategies must be applied at multiple levels, from semiconductor technology choices to cir-

cuit design, firmware protection, and system architecture. The subsequent section provides a brief overview of the most widely adopted strategies for HL-LHC electronics development to ensure reliable operation throughout the experiment's duration.

- **Advanced CMOS processes:** Submicro CMOS processes are selected for radiation-hard, on-detector Application-Specific Integrated Circuits (ASICs), as their thin gate-oxides offer high intrinsic tolerance to TID effects. The reduced oxide thickness limits charge trapping, a primary cause of threshold shifts and increased leakage current. An example of such a design is the RD53 pixel readout ASIC [32], which is based on a 65 nm CMOS process and has demonstrated robust operation up to 600 Mrad.
- **Redundant logic:** The so-called Triple Modular Redundancy (TMR) is widely adopted for critical logic blocks. Functional units are being replicated, and the majority vote is used to mask SEUs effectively. The redundant instances are physically separated to reduce the risk of simultaneous MBUs caused by a single particle strike. Nevertheless, TMR can be compromised if more than one logic copy is affected simultaneously by a particle strike [33].
- **Firmware protection:** Memory cells and configuration registers often include built-in Error Correction Codes (ECCs) to detect and correct SEUs and MBUs. Additionally, periodic firmware-level scrubbing routines can be implemented to read back and rewrite memory contents, thereby preventing the accumulation of bit flips over time. Cyclic Redundancy Checkss (CRCs) are also commonly used in communication links and memory interfaces to detect corrupted data [34].
- **Shielding and placement:** The detector incorporated several radiation shields, such as the barrel toroids and forward shielding disks, which attenuate radiation backgrounds. Nevertheless, due to geometric constraints and the fact that shielding the volume near the IP would interfere with particles, the innermost detector electronics cannot be shielded. To mitigate this, sensitive electronics, such as Field Programmable Gate Arrays (FPGAs), power supplies, and readout systems, are relocated to less-exposed areas, such as the underground service cavern. These are then connected to their corresponding FE modules via optical or shielded copper links [35].
- **Component screening and selection:** In cases where the use of application-specific ASICs is not feasible, Commercial Off-The-Shelf (COTS) components must be utilized. However, this requires stringent pre-selection to ensure consistent radiation resistance across all production lots. The screening process entails the irradiation of samples under TID, NIEL, and SEE conditions that surpass the operational requirements by safety factors, as defined in the "Safety

Factor and Testing Procedure Recommendations of the Radiation Effects Task Force" [31].

3.2. ATLAS Radiation Environment

After providing a comprehensive overview of the fundamental radiation mechanisms, their impact on electronic components, and the available mitigation strategies, the subsequent section discusses the specific radiation conditions within the ATLAS experiment. The origin and spatial distribution of the radiation field in the detector cavern are examined, with emphasis on its dependence on the delivered number of collisions at the IP. This correlation is evident in LHC Run 2 measurements, which show a strong relationship between accumulated radiation and integrated luminosity. The discussion then extends to HL-LHC conditions, based on detailed simulations, to quantify the expected increase in TID and NIEL levels and to define the corresponding challenges for reliable detector electronics during HL-LHC operation.

3.2.1. ATLAS Background Radiation

The predominant source of the radiation field within the ATLAS cavern originates from secondary particles produced in inelastic proton-proton collisions at the IP. Typically, each inelastic collision produces approximately 120 secondary particles, including charged hadrons, neutrons, photons, and electrons. These particles constitute the primary radiation background and are predominantly absorbed by the detector. However, a fraction of these particles emit at small polar angles and interact with surrounding accelerator components, generating tertiary particle showers that may scatter back into the cavern, thereby further contributing to the local radiation field. The contributions from beam-halo or beam-gas processes are comparatively minor.

Consequently, the number of emitted radiation particles can be considered directly proportional to the rate of inelastic interactions at the IP, which is determined by two factors: the luminosity delivered by the machine and the relevant interaction cross section. The instantaneous luminosity L_{inst} [$1/(\text{cm}^2 \text{s})$] is a quantitative measure of the particle flux available for interactions at the IP. The inelastic cross section σ_{inel} (expressed in inverse barns, with $1 \text{ b} = 1 \times 10^{-24} \text{ cm}^2$), functions as a metric quantifying the probability of occurrence of inelastic processes. Their product then results in the expected interaction rate:

$$R = \sigma_{\text{inel}} \cdot L_{\text{ints}} \quad (3.1)$$

The total number of inelastic interactions over a specific period is then accordingly given by the product of the time-integrated luminosity L_{int} and the inelastic cross section:

$$N_x = \sigma_{\text{inel}} \times \int L dt = \sigma_{\text{inel}} \times L_{\text{int}} \quad (3.2)$$

Consequently, accumulated radiation quantities such as TID and NIEL track the integrated luminosity.

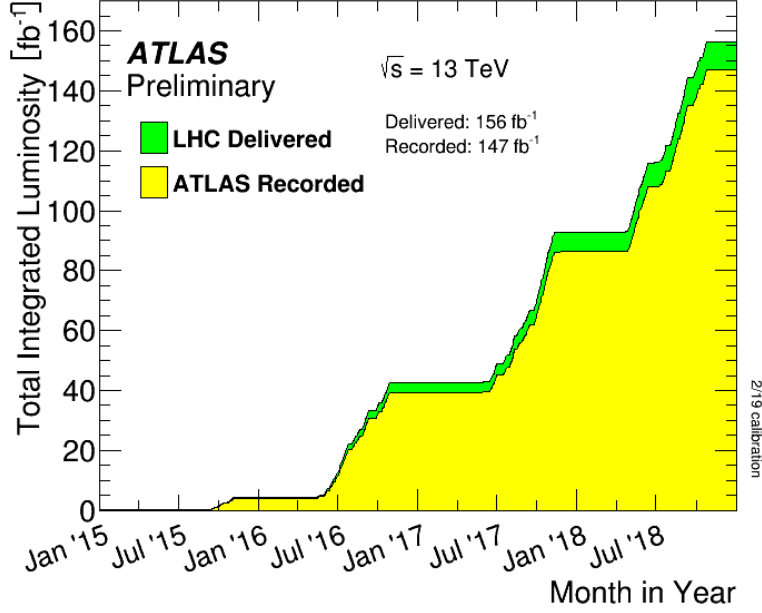


Figure 3.1: Integrated luminosity for p - p physics at $\sqrt{s} = 13$ TeV during LHC Run 2 [36].

The data from LHC Run 2 verify this relationship. As illustrated in Figure 3.1, the delivered and recorded luminosity for the years 2015-2018 is displayed. Figure 3.2 presents the measured TID and 1 MeV neutron fluence in the Inner Detector during the same period. The TID was monitored using Radiation-Sensitive p-MOSFETs (RadFETs), while the 1 MeV neutron-equivalent fluence was measured using BPW34 diode sensors. In both plots, the colored bands indicate sensor spread at specific (r, z) positions, while the central line corresponds to position-average measures. This average is consistently scaled proportionally to the integrated luminosity, thereby validating the predicted proportionality.

3.2.2. Projection to the ATLAS HL Radiation Conditions

The HL-LHC baseline foresees up to $L_{\text{int}} \approx 4000$ 1/fb delivered to ATLAS over its operational lifetime. This increase in integrated luminosity implies a roughly

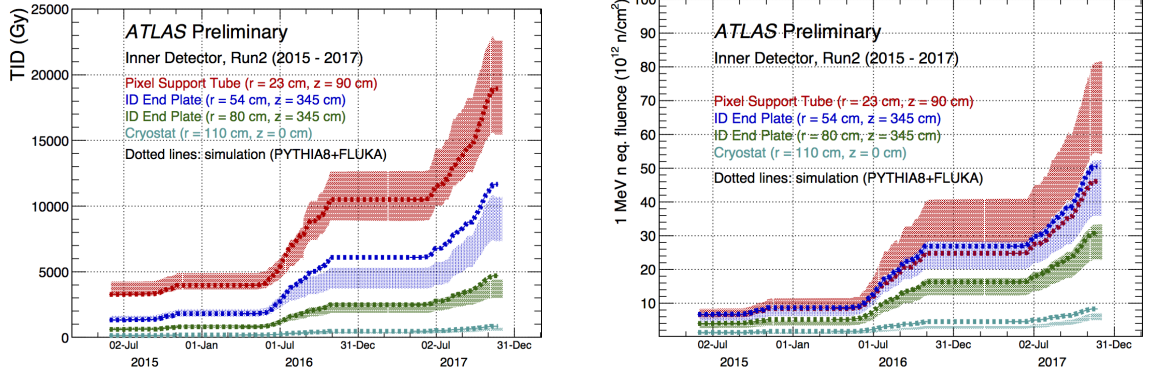


Figure 3.2: TID measurements from RadFETs (left) and 1 MeV neutron fluence measurements from BPW34 diode sensors (right), in the ID during LHC Run 2, shown together with the corresponding pre-run simulation results [37].

order-of-magnitude increase in cumulative radiation exposure relative to the LHC, and necessitates a detailed study of TID and NIEL in both detector and service areas. To assess this, the Radiation Simulation Working Group conducted simulations using the GCALOR [38] package interfaced with the GEometry ANd Tracking (GEANT3) framework [39]. While GEANT3 handles particle tracking, electromagnetic interactions, and detector geometry, GCALOR models hadronic interactions, combining the High-Energy Transport Code (HETC) [40] for charged hadrons up to 10 GeV, FLUKA-based hadronic interaction models [41] at higher energies, and the Monte Carlo Ionization Chamber Analysis Package (MICAP) neutron transport code for neutrons below 20 MeV [42]. Figure 3.3 presents the simulated radiation maps in cylindrical coordinates, with the longitudinal coordinate z (along the beam axis) on the horizontal axis and the radial distance r from the beam line on the vertical axis.

The following characteristics can be derived from the simulations:

- **Spatial concentration:** Peak TID and NIEL are concentrated at small radii and in proximity to the interaction region (ITk, forward regions). The fluence and dose decrease steeply with increasing r and $|z|$.
- **Absorber shadowing:** Massive detector elements, such as the LAr and Tile calorimeters, provide strong local attenuation, reducing the dose and fluence behind these absorbers by several orders of magnitude. This results in the formation of a distinct "shadow" region, where the outer detector regions experience a substantial reduction in exposure.

The simulations indicate that the TID in the silicon tracker region with the highest exposure is between 10^5 and 10^7 Gy, with the absolute peak confined to the innermost pixel regions. The corresponding 1 MeV neutron fluence approaches $10^{15} - 10^{16}$

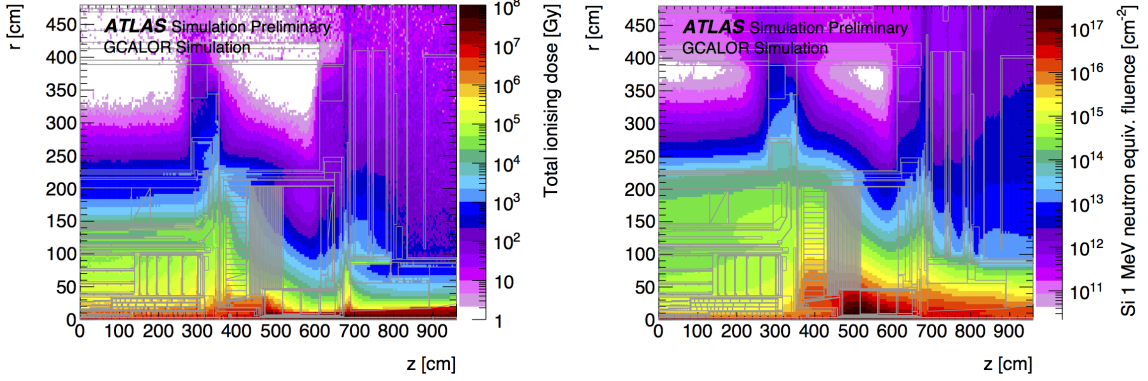


Figure 3.3: Simulation results for the TID in Gy and the total fluence of hadrons with $E > 20$ MeV per cm^2 , for an integrated luminosity of 4000 fb^{-1} , in the tracking and calorimeter regions of the Phase-II ATLAS detector. The geometry model used is symmetric in azimuth, and with respect to $z = 0$ [37].

$n_{\text{eq}}/\text{cm}^2$ in the same volumes. Outside the calorimeter volumes, the TID falls below 50 Gy to 100 Gy and NIEL drops to $10^{10} - 10^{12} n_{\text{eq}}/\text{cm}^2$ range.

Apart from TID and NIEL, SEEs also represent a critical reliability concern for electronics installed within the ATLAS detector. The nature of SEEs is distinct from that of cumulative effects, which can be directly estimated from simulation results. SEEs are inherently stochastic and strongly depend on device technology and operating conditions. Their occurrence is therefore directly linked to the radiation environment in which the devices operate. Within ATLAS, this environment is characterized by a mixed spectrum of particles, ranging from thermal (0.0025 eV) to several TeV. For SEEs, the most critical contribution comes from high-energy hadrons, typically above 20 MeV, which are the leading cause of upsets in CMOS technologies.

Estimating SEE rates is a two-step process. First, the local particle flux and spectrum at a given location must be obtained from simulations. Then, the device sensitivity must be determined experimentally through an irradiation test, in which the SEE cross-section, σ_{SEE} , is measured as a function of particle energy. The expected SEE rate is then calculated as follows:

$$R_{\text{SEE}} = \int_{E_{\text{th}}}^{\infty} \sigma_{\text{SEE}}(E) \Phi(E) dE \quad (3.3)$$

where $\Phi(E)$ is the simulated flux spectrum and E_{th} is the minimum energy threshold at which the device becomes sensitive to SEEs [43].

For HL-LHCs electronics, this implies that the simulation defines the boundary conditions of the particle environment, while device-level irradiation provides the re-

quired sensitivity information. The resulting SEE rate estimates are used to determine the operational requirements for on-detector systems. Consequently, implementing SEE mitigation strategies, including triple modular redundancy, configuration memory scrubbing, error-correcting codes, and the selection of radiation-hardened technologies, is essential for the design of front-end electronics [43].

3.2.3. Conclusion

From the simulations above, the following requirements for the detector electronics and service infrastructure can be defined:

- Electronics placed at a small radius (ITk and forward regions) must tolerate TID of up to 1×10^7 Gy and NIEL up to 10^{16} n_{eq}/cm². Device selection, development, and qualification for these zones must follow the most stringent criteria.
- Calorimeter structures strongly attenuate the radiation field; nonetheless, intermediate regions can still see doses of up to 1×10^4 Gy and require radiation-tolerant or remotely located electronics.
- Cavern walls and service caverns fall into much lower categories (TID < 100 Gy, NIEL < 10^{11} n_{eq}/cm²), permitting the use of less-stringent radiation-tolerant component classes and easing maintenance access.

These spatially varying requirements motivate the two primary strategies employed in ATLAS upgrades. First, sensitive, high-complexity electronics are relocated to low-radiation areas wherever feasible. Second, when on-detector electronics are unavoidable, radiation-tolerant electronics are implemented and tailored to the simulated local exposure.

The following chapter introduces the ATLAS DCS. It provides a comprehensive overview covering the system requirements to ensure reliable and coherent detector operation, the fundamental system architecture, the common middleware standard, and the FE-interfacing hardware in use. Subsequently, the requirements imposed on the DCS by the HL-LHC era are outlined, together with the strategy adopted to ensure the system remains capable of supporting safe detector operation.

4. The ATLAS Detector Control System

The ATLAS Detector Control System (DCS) [44], in conjunction with the Data Acquisition and Trigger System (TDAQ) [45], serves as the operational backbone of the detector. While the TDAQ system is responsible for the readout, communication, and temporary storage of data from physics events, the DCS enables control and monitoring of detector hardware and infrastructure. Its primary function is to ensure safe, reliable, and coherent detector operation at all times.

This chapter provides a comprehensive overview of the ATLAS DCS, primarily based on Ref. [44]. It first summarizes the functional requirements and describes the overall system architecture. Next, the common middleware standard used within the DCS is introduced, which provides the standardized interface between the Back-End and Front-End. The chapter then discusses the FE hardware that enables the system's operation. Finally, the challenges posed by the HL-LHC upgrade are examined, together with the corresponding strategies and solutions developed to ensure that the DCS remains robust and can provide safe detector operation during the HL-LHC era.

4.1. Requirements

The ATLAS detector is organized in a hierarchical, tree-like structure of detectors, subdetectors, subsystems, and technical infrastructure, each with a defined degree of operational independence. The DCS must reflect this structure in both its design and implementation. It must support the transition of the detector to any predefined operational state while ensuring continuous monitoring and archiving of operational parameters. Consequently, it facilitates the identification and communication of anomalous conditions and supports both manual and automated processes for their rectification.

Given the experiment's multi-decade timescale, spanning more than 30 years from research and development through construction, commissioning, and full-scale operation, the system had to evolve seamlessly from small-scale prototypes to full-scale deployment in an integrated detector environment. Moreover, the heterogeneity of technologies employed throughout the experiment necessitated a standardized, homogeneous interface to minimize maintenance and integration effort and facilitate the development of a consistent, uniform operator interface.

To fulfill these requirements, the DCS must offer high scalability and flexibility while remaining platform-independent. It must be capable of controlling and monitoring more than 10^6 parameters while interacting with external systems, including the LHC, CERN services, the ATLAS magnets, and the Detector Safety System (DSS). Furthermore, the system must be fault-tolerant, support remote diagnostics, and use hardware that meets the required radiation-tolerance standards, as the detector is inaccessible during operation due to ionizing radiation.

4.2. System Design

As mentioned before, the design of the ATLAS DCS must address the experiment's large scale and multi-decade lifetime while maintaining flexibility and scalability. However, given the fact that these challenges are common to all LHC experiments and the accelerator complex, the Joint Controls Project (JCOP) was initiated in 1998. The initiative's objective was to validate a set of commercial technologies and incorporate them into a reusable software framework to promote standardization across CERN. The result of this initiative was the JCOP framework, which provides a catalog of modular, experiment-agnostic components and integration tools that reduce development effort and support long-term maintenance of control systems. As part of this technology selection process, a detailed evaluation of industrial Supervisory Control and Data Acquisition (SCADA) systems was conducted, resulting in the selection of Process Visualization and Steering System (PVSS), developed by ETM and later acquired by Siemens and renamed as SIMATIC WinCC Open Architecture (WinCC OA) [46], as the supervisory backbone of the DCS. WinCC OA supports large-scale, distributed deployments and interfaces with devices through standardized middleware. Its adoption ensured that the ATLAS supervisory layer would rely on a stable, industry-supported platform, while the JCOP Framework provided experiment-specific extensions and integration tools. Adopting the JCOP framework, the ATLAS DCS also implements the standard SCADA paradigm of separating device-facing logic from supervisory control.

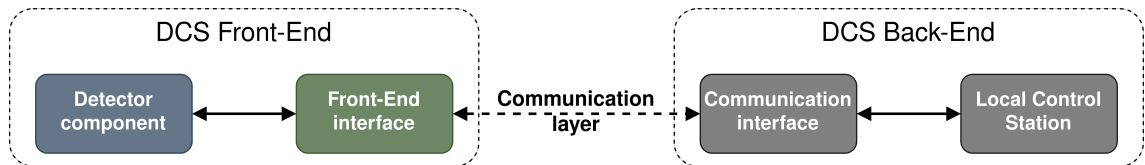


Figure 4.1: Definition of Front-End and Back-End components within a DCS system. Adapted from Ref. [47].

4. The ATLAS Detector Control System

The ATLAS DCS is architecturally organized into FE and BE components as shown in Figure 4.1. The FE comprises detector-facing hardware and interface electronics, including custom FE modules, power services, and General Purpose Input/Output (GPIO) boards, such as the Embedded Local Monitoring Board (ELMB). These FE devices connect to the BE via Ethernet, or in legacy installations, via Controller Area Network (CAN).

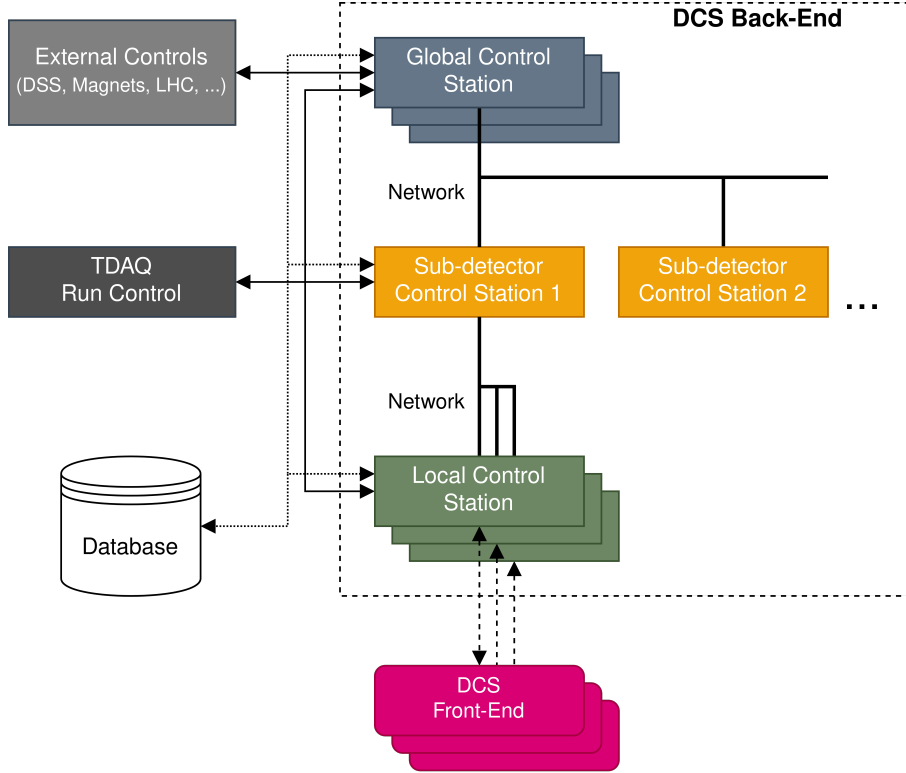


Figure 4.2: Back-End architecture of the ATLAS DCS. Adapted from Ref. [47]).

The DCS BE is based on interconnected control stations that deploy a distributed SCADA system using WinCC OA. Communication between the detector-facing interfaces and the Local Control Stations (LCSs) is provided by a middleware layer based on the industry-standard Open Platform Communications Unified Architecture (OPC UA) [48]. As illustrated in Figure 4.2, the BE is tiered into three logical layers: LCSs that handle process-level control of subsystems; Sub-Detector Control Stations (SCSs) that coordinate and enable stand-alone operation of individual subdetectors; and the Global Control Stations (GCSs), which host central services, interfaces to external systems, and the operator Human-Machine Interfaces (HMIs) in the ATLAS control room. The complete BE consists of approximately 150 commercial rack servers interconnected via a dedicated Local Area Network (LAN). The hierarchical detector structure is implemented as a distributed Finite-State Machine

(FSM) in software, facilitating the propagation of control states between the operator's HMI and individual devices.

4.3. Standard Middleware OPC UA

In the context of distributed control systems, middleware functions as the intermediary between applications and components. The role of middleware is to manage a diverse range of data models while ensuring portability and optimal performance within a hierarchical mesh of heterogeneous components. For the ATLAS DCS, OPC UA was selected as the standard for all system communications. Accordingly, each DCS FE interface exposes its control and monitoring functions through a OPC UA endpoint.

OPC UA, as a platform-independent, service-oriented architecture designed for industrial automation and machine-to-machine communication across devices, operating systems, and programming environments. Key characteristics of the OPC UA architecture are:

- **Cross-platform and language-agnostic:** OPC UA is operating system-independent and portable across a wide range of platforms, supporting development in multiple programming environments. The reference stack is implemented in standard ANSI C, while Software Development Kits (SDKs) are available in C++, Java, .NET, Python, and other languages.
- **Flexible communication models:** Supports both client-server and publish-subscribe communications paradigms, operating over multiple transport protocols including TCP/IP, UDP, WebSockets, and MQTT.
- **Security framework:** Incorporates built-in mechanisms for authentication, authorization, encryption, and data integrity, ensuring secure data exchange in critical environments.
- **Advanced Data Modeling :** The OPC UA information model enables the creation of semantic-rich, hierarchical representations of devices, processes, and systems. Beyond simple variables and parameters, it supports complex data types, relationships, methods, and event-driven interactions. Its graph-based architecture allows flexible structuring, including simple variables, trees, and cyclic graphs, supporting the organization of data.

Within the DCS, numerous OPC UA servers are deployed to interface with a wide range of devices, including power supplies, VME crates, and the ELMB. To simplify and standardize the development of these servers and to minimize maintenance efforts, the quick OPC UA server generation framework (quasar) (see Section 6.2.3)

was introduced. This framework provides a structured and automated approach for generating OPC UA servers, ensuring consistency across implementations and accelerating the deployment of new device interfaces [49].

4.4. Embedded Local Monitoring Board

The hardware requirements for the ATLAS DCS have been primarily driven by the already demanding conditions within the experiment’s cavern during the LHC era, where equipment must function reliably in the presence of a strong magnetic field and withstand radiation levels integrated over the experiment’s lifetime. Nevertheless, general requirements such as the use of low-cost commercial components, low power consumption, and high I/O density apply to any equipment used, regardless of the environmental condition.

The complexity of the detector, with its variety of customized designs and FE electronics, both within and outside the detector, as well as the requirement to monitor and control these components, presented a significant challenge. Moreover, the need to measure environmental parameters, such as temperature and humidity, within the cavern further necessitated a suitable DCS FE device. Given the absence of a commercially available solution that met the specific requirements, the DCS shifted its focus towards developing a low-cost, purpose-built I/O concentrator. Given the CERN Fieldbus Group’s prior adoption of the CAN fieldbus as a common standard for FE communication, the new device was designed from the outset to operate on CAN. This ensured immediate compatibility with the experiment’s fieldbus topology and leveraged CAN’s robust electromagnetic compatibility. The resulting device, the ELMB, provides a compact, robust, and standardized CAN-based I/O interface for detector and cavern monitoring [44].

The Embedded Local Monitoring Board (ELMB) is a custom-designed I/O concentrator developed as a standardized solution for interfacing with custom FE electronics in the ATLAS P1 installations. At its core, the ELMB features an 8-bit ATmega128 microcontroller from ATMEL, which operates at 4 MHz from a local crystal oscillator. Despite its assembly from commercial off-the-shelf (COTS) components, particular emphasis was placed on selecting electromagnetic and radiation-tolerant components to meet the stringent requirements of the ATLAS environment.

The following provides a comprehensive overview of the ELMB, including its hardware and firmware, as well as a summary of its radiation qualification. This facilitates an evaluation of the current status of the DCS, taking into account the challenges associated with the upcoming LHC update. The design and firmware

implementation of the ELMB are described in detail in Ref. [44], which serves as the primary reference for this section.

4.4.1. Hardware

The ELMB is architecturally divided into three electrically isolated sections (see Fig. 4.3), each operating within its own supply domain. Every domain is equipped with a dedicated voltage regulator that features current limiting and thermal protection to mitigate fault conditions, including over-current events that may arise from SEL. The inter-domain signaling is enabled by galvanically isolated optocouplers, which maintain electrical isolation while providing reliable communication between domains.

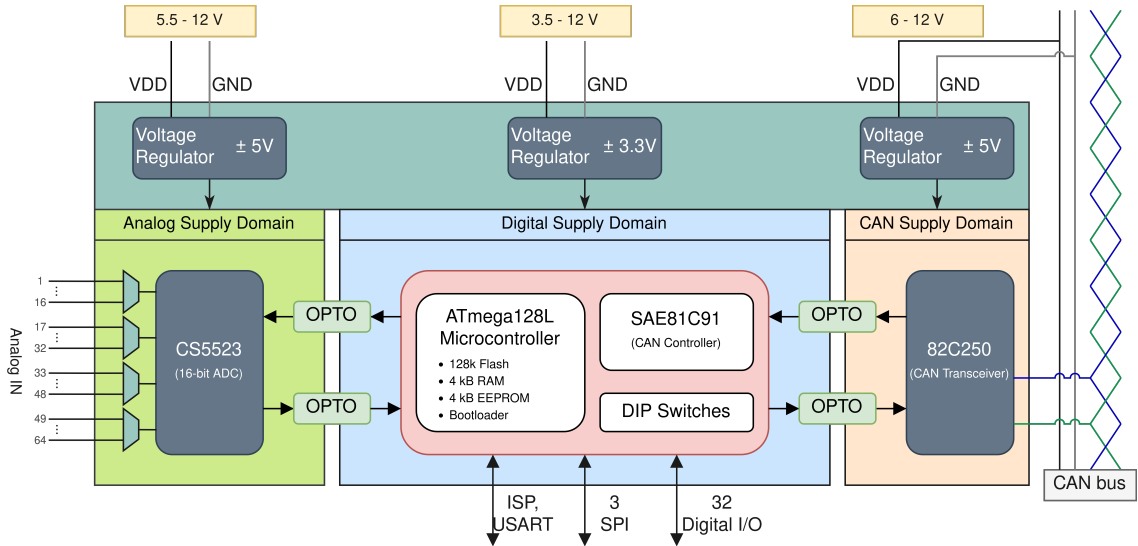


Figure 4.3: Block diagram of the Embedded Local Monitoring Board, showing the three electrically isolated and separately powered functional sections. Adapted from Ref. [50].

The ELMB is built around an ATmega128 microcontroller that manages data acquisition, I/O, and communications. The microcontroller interfaces with the on-board CAN controller via Serial Peripheral Interface (SPI) and provides In-System Programming (ISP) and Universal Synchronous/Asynchronous Receiver Transmitter (USART) interfaces for serial programming and communication.

The analog measurement is performed by a 16-bit delta-sigma Analog-to-Digital Converter (ADC) with four differential input channels, capable of unipolar or bipolar measurements. These four ADC inputs are time-multiplexed with 16 analog

multiplexers, yielding up to 64 individually addressable analog channels. The ADC is connected to the ATmega128 via SPI, allowing the microcontroller to schedule conversions and configure conversion parameters as needed. In addition, the microcontroller exposes 32 digital I/O lines, increasing the ELMB's versatility for a wide range of monitoring and control tasks.

The ELMB's compact dimension of 50 mm x 67 mm is well-suited for space-constrained installations. To facilitate deployment in stand-alone configurations, a general-purpose motherboard was developed, providing standardized connectors for digital I/O, analog inputs, CAN bus, and power.

4.4.2. Firmware

The ELMB firmware comprises two distinct applications written in C and programmed into the microcontroller's flash memory: the ELMB bootloader and an ELMBio firmware application.

The bootloader enables remote in-situ firmware upgrades over the CAN bus. Upon system startup, a valid firmware update can be written directly to the microcontroller's flash memory using the CANopen protocol. After the update is complete, the ATmega128 executes the application code from flash.

The ELMBio firmware application is the standard firmware used on the ELMB and provides full access to the device's analog and digital I/O interfaces. It is a general-purpose CANopen application that enables ELMB functionality without requiring custom firmware development. The firmware supports all 64 analog inputs via the onboard multiplexed ADC, the 16 digital inputs, the 16 digital outputs, and an external Digital-to-Analog Converter (DAC). The firmware is compliant with the CANopen DS-401 device profile for I/O modules [51], enabling configuration and data exchange through standard CANopen services. Configuration parameters, such as the ADC channel conversion rate, can be configured during in-system testing or production using CANopen Service Data Objects (SDOs). The configuration values are stored in the device's EEPROM, ensuring persistence across reboots. Access to analog and digital I/O data is facilitated by Process Data Objects (PDOs), a lightweight message type within the CANopen protocol that incurs minimal protocol overhead. These PDOs facilitate efficient synchronous or asynchronous data transmission, whereby analog data is transmitted upon the occurrence of a predefined threshold and digital inputs are initiated upon state transitions.

To ensure reliable operation in radiation-prone environments, the firmware implements a set of software-based protection mechanisms against SEEs. These include a supervisory watchdog timer, periodic register refreshing, and CRCs on memory

contents to detect and mitigate potential data corruption. Additionally, CANopen provides built-in error-control protocols that facilitate the monitoring and management of networks comprising multiple CANopen nodes. A central Network Management (NMT) master is responsible for supervising the network by controlling the states of individual nodes based on the following CANopen protocols:

- The heartbeat protocol verifies node availability by having nodes send heartbeat messages at regular intervals.
- The Node/life-guarding protocol uses a Remote Transmission Request (RTR) sent by the NMT master, to which each node replies with a Node Guard message containing its state and a toggle bit.

4.4.3. Radiation Qualification

The ELMB was designed to operate in the high radiation environment of the LHC. As it is built from radiation-sensitive COTS components, including CMOS SRAM, EEPROM Flash memory, bipolar devices, and optocouplers, these components were selected and qualified to meet the required radiation tolerance. To estimate the accumulated radiation levels expected over a decade of uninterrupted operation at the ELMB's intended installation locations, dedicated simulations were performed for the three standard radiation metrics, TID, NIEL, and SEE. Furthermore, to account for uncertainties in the accuracy of these simulations, as well as to consider potential Low-Dose-Rate (LDR) effects and lot-to-lot variations in COTS-based designs, the final required radiation tolerance levels incorporate additional safety factors, defined in the "ATLAS Policy on Radiation Tolerant Electronics" [52]. In the ELMB qualification case, the incorporation of the applied safety factors in conjunction with the simulated fluences/doses yielded the requisite radiation-tolerance levels, including all safety margins, as follows:

- TID: 14 Gy
- NIEL: 9×10^{10} neutrons/cm²
- SEE: 1×10^{10} hadrons/cm²

A comprehensive irradiation program was conducted between 1998 and 2003. This program involved an initial component screening in 1998, pre-series tests from 2001 to 2003, and final production-series qualification in 2003. The production series of the ELMB achieved the following qualified tolerances:

- TID: 140 Gy
- NIEL: 6×10^{11} neutrons/cm²

- SEE: 5×10^{10} hadrons/cm²

The outcomes of these irradiation campaigns demonstrated that the ELMB production units significantly exceed the requirements for TID, NIEL, and SEE, providing comfortable operating margins despite simulation uncertainties and batch variability. Consequently, the safe operation of the ELMB within the detector and DCS is ensured throughout the LHC's operational lifespan.

4.5. DCS Requirements for HL-LHC

The HL-LHC upgrade, with its significant increase in particle collision rates, integrated luminosity, and radiation levels, poses unprecedented challenges for the ATLAS DCS. Nevertheless, the DCS must continue to ensure the coherent, safe, and reliable operation of all detector subsystems and infrastructure, even under these more demanding conditions. Consequently, a comprehensive reevaluation of the DCS, particularly its FE-interfacing hardware and associated infrastructure, is required to ensure the long-term integrity and stability of detector operation.

The ELMB demonstrated to be a reliable and robust DCS FE solution for LHC Run 1-3 under the given conditions. However, it was not designed to withstand the radiation level expected during HL-LHC operation. Critical components, such as ADCs and optocouplers, are prone to radiation-induced failures, including SEUs and cumulative degradation. Consequently, a majority of ELMBs would be expected to encounter irreversible bit flips or permanent loss of functionality before the end of Run 4. Beyond this issue, the ELMB architecture also exhibits intrinsic limitations that no longer meet the requirements of new Phase-II detector systems. The device's constrained I/O density limits the number of parameters that can be monitored and controlled per board, whereas modern detector systems require coverage of thousands of parameters. Consequently, covering all channels of Phase-II detector systems would necessitate deploying a large number of ELMBs. This would increase infrastructure complexity, including cabling and power-distribution requirements, making the architecture difficult to scale efficiently.

The ELMB's inadequate radiation tolerance alone renders it unsuitable for operation at the HL-LHC. However, it is essential to note that not all ATLAS subsystems will undergo a complete replacement. In several cases, parts of the existing infrastructure will be retained, including the CAN-based DCS. This creates a dilemma: while retained infrastructure continues to require a CAN-compatible radiation-hardened FE device, newly developed detector systems demand a more scalable and performant DCS architecture with higher radiation tolerance, I/O density, and communication

throughput. Addressing these divergent requirements has motivated the adoption of a dual strategy for the Phase-II DCS, resulting in the development of two complementary solutions:

- **ELMB2**, an increasingly radiation-tolerant drop-in replacement for the legacy ELMB, designed for subsystems where the existing DCS infrastructure is retained.
- **EMCI-EMP**, a modular, high-bandwidth, radiation-hard interface developed to meet the demands of new detector systems.

4.5.1. ELMB2

The ELMB2 is the direct successor of the original ELMB, which was designed as a pin-compatible, drop-in replacement for retained DCS infrastructure. The device maintains the compact form factor, the CANopen interface, and the identical analog/digital channel layout. This compatibility ensures that any ELMB2 can replace an ELMB in existing crates, enabling cabling without requiring system-level modifications. To ensure long-term operability under HL-LHC conditions, the radiation-sensitive components of the ELMB, including the voltage regulators and optocouplers, were replaced with more radiation-tolerant equivalents. A dedicated irradiation campaign demonstrated that the ELMB2 remains fully functional up to a TID of 32 krad and a neutron fluence of 1.0×10^{13} neutrons/cm². The tests show a significant improvement in NIEL and TID tolerance compared to the ELMB. Consequently, the ELMB2 provides a robust and sustainable solution for subsystems that retain their CAN-based DCS infrastructure at the foreseen locations [53] [54].

4.5.2. EMCI-EMP

In contrast to the ELMB2, which primarily addresses the requirements of subsystems retaining their legacy infrastructure, the EMCI-EMP system has been developed to overcome the architectural limitations of the ELMB and to provide a scalable solution for newly designed detector systems. The system comprises two complementary platforms: the Embedded Monitoring and Control Interface (EMCI) and the Embedded Monitoring Processor (EMP).

The EMCI serves as the radiation-hard on-detector FE, providing modular, flexible interfaces for control and monitoring. Conversely, the EMP, operating in a radiation-free environment, serves as the system's backbone, connecting up to 12 EMCIs in a star topology while interfacing with the distributed control system. The EMCI and

EMP together address the performance and scalability requirements of the HL-LHC upgrade.

The subsequent chapter will present the overall system architecture in detail, along with a comprehensive overview of both the EMCI and the EMP, including their hardware designs and component selections.

5. The EMCI-EMP DCS Interface

The Embedded Monitoring and Control Interface (EMCI) and the Embedded Monitoring Processor (EMP) constitute the next-generation system developed to extend and complement the ATLAS DCS during the HL-LHC era. The system is based on a modular approach that addresses the strict radiation tolerances for on-detector hardware and overcomes the architectural scalability and performance limitations of the legacy DCS FE solution.

This chapter first provides a comprehensive introduction to the EMCI-EMP DCS interface. It outlines the system's architecture, operational spectrum, and identified target applications within the ATLAS detector. Thereafter, the hardware design and components of both the EMCI and EMP are presented in detail.

5.1. EMCI–EMP Interface Overview

The EMCI–EMP system is a next-generation DCS interface designed to provide reliable, high-performance monitoring and control of ATLAS on-detector components. This section provides a thorough overview of the system, focusing on its architecture, operational capabilities, and intended applications within the ATLAS detector.

5.1.1. System Architecture

During HL-LHC operations, the monitoring and control of on-detector components within ATLAS will typically be based on two dedicated paths:

- A primary path dedicated exclusively to DCS functions, based on independent FE interfaces, communication, and power lines.
- A secondary path that uses the communication infrastructure of the on-detector electronics and shares resources with other types of data, such as readout data.

The path used to implement monitoring control of the respective on-detector FE depends largely on whether the target requires a high degree of reliability and availability or if its data is subject to less critical use. For critical monitoring and control applications, it is imperative to implement the primary path using a dedicated, independent DCS FE-interface. Critical use cases include, for example, temperature monitoring, where temperature is used for functions such as software interlocks,

powering, cooling, and gas distribution controls. This path must be highly reliable and available at all times, even when the detector and integrated electronics are not fully powered. In contrast, the secondary path relies on the communication infrastructure of other subsystems, such as TDAQ. This results in DCS data sharing resources with other data types, such as trigger or readout data. Therefore, its implementation should be limited to less important DCS data or when the primary path cannot be implemented due to space constraints or service availability.

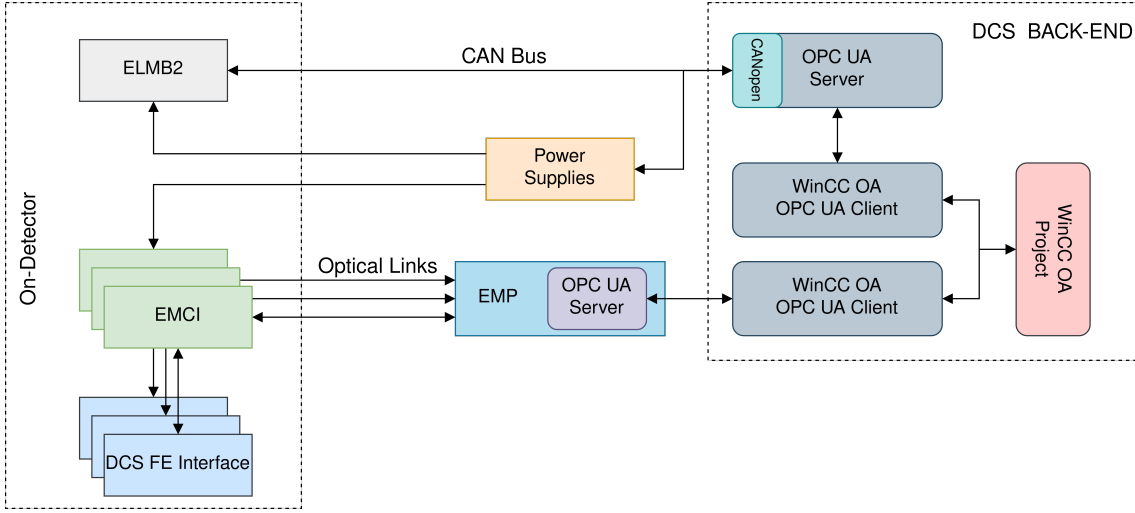


Figure 5.1: Schematic view of the primary DCS data path, for control and monitoring data which require a high degree of reliability and availability. Implementation of this path is either based on the ELMB2 or on the EMCI-EMP system. Adapted from Ref. [47].

As shown in Figure 5.1, the primary path for the DCS utilizes either the ELMB2 or the EMCI-EMP system to connect to the dedicated DCS FE-interfaces of on-detector hardware. As a drop-in replacement for legacy systems, the ELMB2 preserves the existing ELMB infrastructure and provides a straightforward upgrade path for established setups. By contrast, the EMCI-EMP system is a next-generation solution explicitly designed for the HL-LHC era. It is built around two complementary platforms: the EMCI and the EMP.

The EMCI serves as the interface that connects to the FE-interface of the on-detector hardware. Designed with radiation-tolerant components, it ensures reliable operation in high-radiation environments, enabling flexible placement within the detector to meet system requirements. The EMCI interfaces with various detector FE technologies via electrical Links (eLinks) and connects to the EMP through high-speed optical links. The EMP, in turn, serves as the system's backbone and is built from

COTS components and therefore must be installed in low-radiation environments (e.g., counting rooms). A single EMP can communicate with up to 12 EMCIs, enabling a scalable, flexible system design. It also supports communication rates of up to 320 Mbit/s with FE electronics, which is several orders of magnitude higher than the current ELMB standard of 500 kbit/s. A diagram of the entire EMCI-EMP system architecture is shown in Figure 5.2.

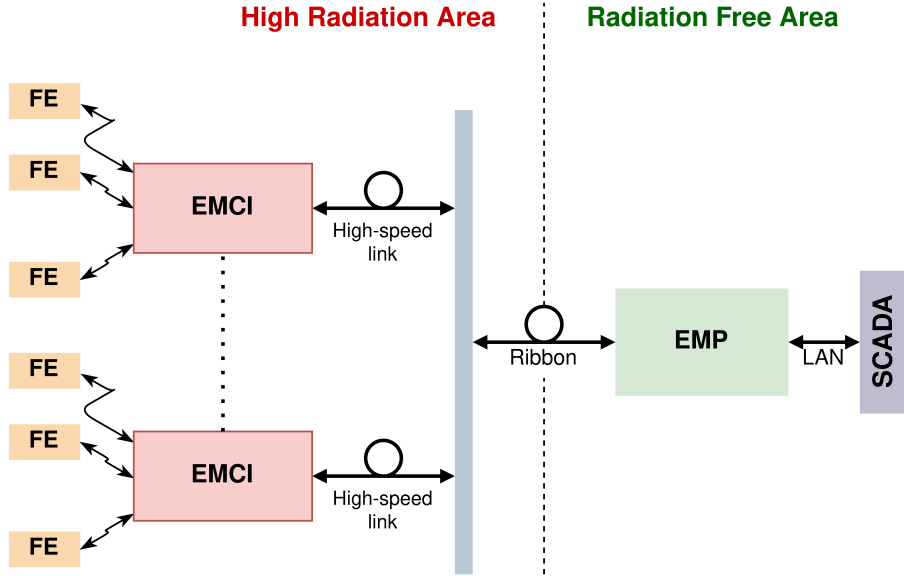


Figure 5.2: EMCI-EMP system overview. The EMP forms the backbone of the novel DCS interface, connecting up to 12 EMCIs in a star topology via high-speed optical links and interfacing with the distributed SCADA system via LAN.

Definition

The term **uplink** is used to describe the data link from FE Electronics to the EMP. In the opposite direction, the link is called **downlink**.

5.1.2. Functional Scope of the EMCI-EMP Interface

The EMCI-EMP interface supports a wide range of potential operational tasks, reflecting the diverse requirements of FE electronics and the detector infrastructure. Its design enables flexible integration with various FE technologies while ensuring reliable monitoring and control. Its functional scope encompasses:

- **EMP hardware control:** Includes configuration and status readback of on-board peripheral components, clock generation, and sensor acquisition to support reliable operation and fault detection.
- **EMCI configuration and control:** Provides register-level access to the EMCI, for configuration, monitoring, and operational management of internal functions. In addition, the EMCI's slow-control interface exposes dedicated I/O pins for user-defined tasks.
- **FE electronics integration:** Supports user-defined protocols via the eLink interfaces, ensuring connectivity with a broad range of on-detector FE modules. Payload framing, DC balancing, and optional error correction are handled in firmware.
- **Extended interfacing options:** The EMP supports both single-ended and differential signaling through the SoM's I/O banks. Additional interfaces, including CAN, Universal Asynchronous Receiver-Transmitter (UART), SPI, and I2C, can be implemented in Programmable Logic (PL) using dedicated IP-cores. Processing system-based interfaces, such as Multiplexed I/Os (MIOs), further enhance integration with external systems and debugging tools.

The diverse functionality of the EMCI-EMP system underscores its high versatility and configurability, enabling compact and application-specific implementations within the DCS. Mandatory functions, such as configuration and monitoring of EMP and EMCI hardware, are implemented alongside custom FE applications, while respecting the system's processing capabilities and available logic resources. This approach is designed to ensure an optimal balance of functionality, performance, and resource utilization.

5.1.3. EMCI-EMP Target Applications

The design and development of the EMCI-EMP interface were guided by several potential target applications identified during the early planning phase of the DCS upgrade. These use cases played a crucial role in designing the hardware architecture, ensuring that the final system meets the diverse operational requirements of the target applications. While the majority of applications will employ the complete EMCI-EMP system, which includes both the EMP and EMCI, there are applications, such as the ITk Interlock system, in which only the EMP is used. The following summarizes the target applications that will utilize the EMCI-EMP system during the Phase-II upgrade:

- **ITk Strips:** The EMCI-EMP will control and monitor AMAC ASICs, transmitting and receiving signals via the EMCI [55].

- **ITk Pixel interface to MOPS:** The EMP will communicate with custom FPGA-based Mopshub FE devices via EMCI-served eLinks, providing CAN communication to the MOPS FE monitoring ASIC [56].
- **TileCal SIN sensor readout:** The EMP will configure and control EMCIs to process digitally frequency-modulated signals, which convey positional data from TileCal system probes [57].
- **TGC Charge Monitoring System:** The EMP will host EMCIs to interface with custom FPGA-based modules connected via the EMCI for monitoring applications [47].
- **ITk Interlock:** In this case, the EMP will directly interface with custom FPGA-based interlock modules within counting rooms via differential signaling, without using EMCIs [47].

In addition to the application-specific control and monitoring functions listed above, all deployments may require auxiliary slow-control I/O capabilities. Depending on the installation, these I/O channels may be either located locally at the EMP chassis (e.g, for nearby temperature probes) or in proximity to the EMCI.

In summary, these application examples demonstrate the flexibility of the EMCI-EMP system. By supporting a wide range of FE technologies through a standard, modular interface, the system enables high standardization within the DCS, facilitating rapid integration into the BE.

5.2. EMCI: Design and Technical Specification

The harsh radiation environment expected during HL-LHC operation no longer permits the use of microprocessors or FPGAs in direct proximity of the IP. Consequently, the intelligence in the FE electronics is traded off against radiation tolerance and high-bandwidth devices, which primarily provide robust data transport and basic monitoring functionality. This approach shifts the bulk of control logic and data processing to the BE.

Accordingly, the EMCI primarily serves as a bidirectional interface between the on-detector FE and the high-performance EMP. The device's design is based on radiation-tolerant components from the CERN Radiation Hard Optical Link project, ensuring reliable operation in the harsh radiation environment of the ATLAS cavern.

This section introduces the EMCI by outlining its hardware architecture, the key building blocks that enable its functionality, and the radiation tolerance of its main

components. The presented information regarding the EMCI is described in detail in its Electronics Specification (Ref. [58]), which serves as the primary reference.

5.2.1. Hardware Architecture

The EMCI is based, as illustrated in Figure 5.3, on the low-power GigaBit Transceiver (lpGBT) [59], a radiation-tolerant Serializer/Deserializer (SerDes) ASIC that aggregates FE signals into a single bidirectional data stream. This data stream is interfaced with the Versatile Transceiver+ (VTRx+) [60], a high-speed optical transceiver that establishes optical connectivity between the EMCI and the EMP.

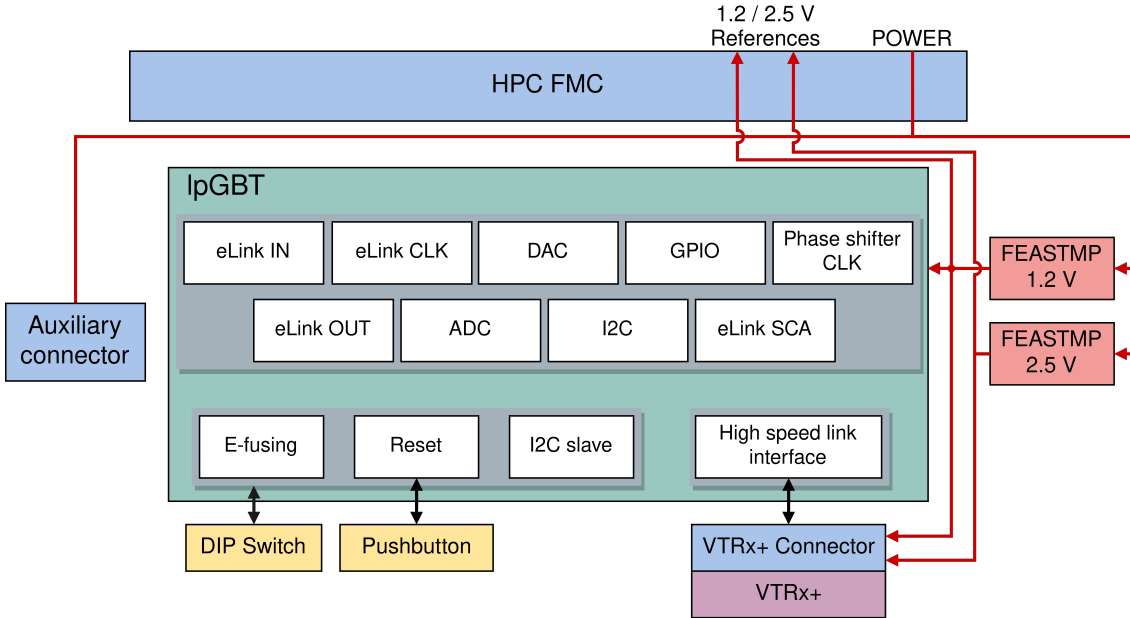


Figure 5.3: Block diagram of the EMCI. At its core, the EMCI is based on the lpGBT, a radiation-tolerant ASIC, and is powered by two FEASTMP DC-DC converters. Connectivity to the EMP is provided via a VTRx+ optical transceiver, while the eLinks for interfacing with on-detector FEs are accessible through the High-Pin-Count (HPC) FPGA Mezzanine Card (FMC) connector. Adapted from Ref. [58].

Connectivity with the FE electronics is established via the lpGBT's eLinks. These eLinks, along with other interfaces, are routed to the HPC FMC connector on the EMCI. The pinout of this connector partially conforms to the VITA 57.1 FMC standard, thereby ensuring full compliance with all ground and power assignments. While some non-general-purpose pins are repurposed, the design ensures compatibility with standardized carrier boards, thereby removing the risk of electrical damage.

To access the eLinks via the FMC connector, a customized motherboard is required to separate and route the signals according to the specific needs of the FE electronics.

The EMCI can be powered either via the FMC connector or, for debugging, via the auxiliary connector. It accepts an input voltage in the range of 5 V to 12 V, which is regulated by two on-board FEASTMP DC/DC converters [61], to provide the required 1.2 V and 2.5 V supply rails. The FEASTMP is a DC/DC converter module developed at CERN, based on the FEAST2 ASIC, and is specifically designed for operation in high-radiation and strong magnetic-field environments. Figure 5.4 shows a computer-generated image of the EMCI, showing the key components such as the FEASTMP DC/DC converters, the lpGBT, and the HPC FMC connector.

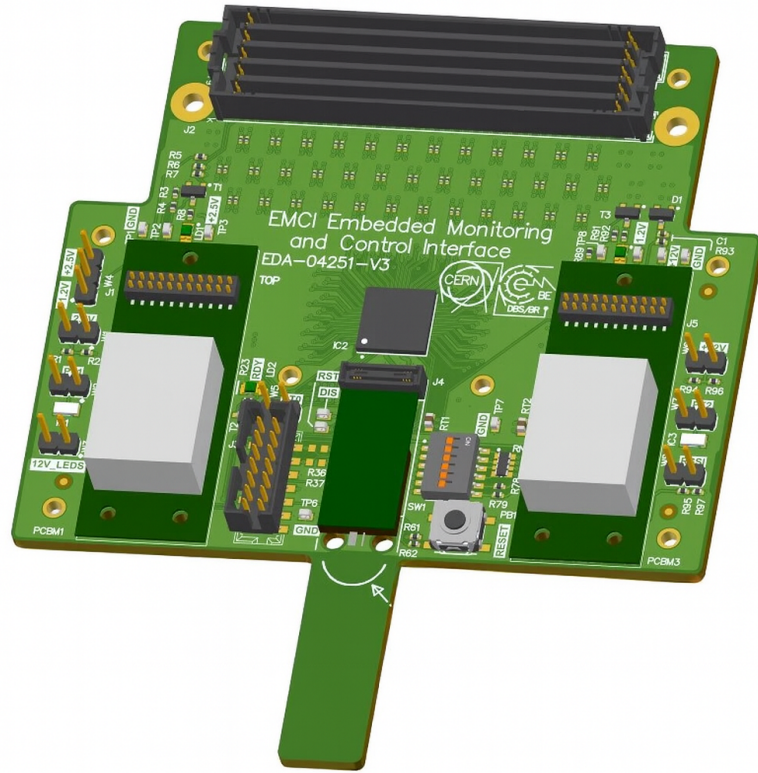


Figure 5.4: Computer generated image of the EMCI Printed Circuit Board (PCB) [62].

5.2.2. Radiation Tolerance and Operational Requirements

The EMCI has been designed for operation in the radiation environment of the HL-LHC. Consequently, its primary active components, namely lpGBT, VTRx+, and FEASTMP, must possess the necessary radiation-tolerance qualification. As such, the development and testing of these components have been conducted to ensure compliance with the requisite radiation requirements. A comprehensive report of the results from the corresponding qualification campaigns is available in the respective datasheets for the components [59–61]. Table 5.1 summarizes the radiation levels for which these components are qualified.

	Total Ionizing Dose	Total Neutron Fluence	Total Hadron Fluence
lpGBT	200 Mrad	-	-
VTRx+	100 Mrad	$1 \times 10^{15}/\text{cm}^2$	$1 \times 10^{15}/\text{cm}^2$
FEAST2	100 Mrad	$5 \times 10^{14}/\text{cm}^2$	$8.5 \times 10^{14}/\text{cm}^2$

Table 5.1.: Overview of the radiation qualification of EMCI components.

In addition to these radiation-tolerant components, the EMCI also integrates active COTS components, such as a Zener diode and three Light-Emitting Diodes (LEDs) controlled by N-channel MOSFETs. While the Zener diode showed no evidence of radiation-induced degradation, the LEDs and MOSFETs are not required for operation, and their power circuitry is designed to be disconnected by removing a dedicated jumper, thereby avoiding any risk of failure and not impacting system functionality.

5.2.3. The Low Power GigaBit Transceiver

This section delineates the lpGBT ASIC around which the EMCI is built. A more comprehensive description of it can be found in the lpGBT documentation (Ref. [59]). The following offers a concise overview of the most relevant functionalities for the EMCI-EMP DCS interface.

The lpGBT is a radiation-tolerant ASIC developed at CERN for use in the harsh HL-LHC radiation environment. It serves mainly as a SerDes, converting between a bidirectional high-speed serial link and multiple parallel eLinks.

The bidirectional high-speed can be used to establish a connection to the counting rooms, offering a fixed downlink rate of 2.56 Gbit/s and a configurable uplink rate of either 5.12 Gbit/s or 10.24 Gbit/s. Both directions employ interleaved Reed-Solomon Forward Error Correction (FEC) codes [63] to detect and correct transmission errors, and scrambling to limit DC imbalance. Transmission and reception of these

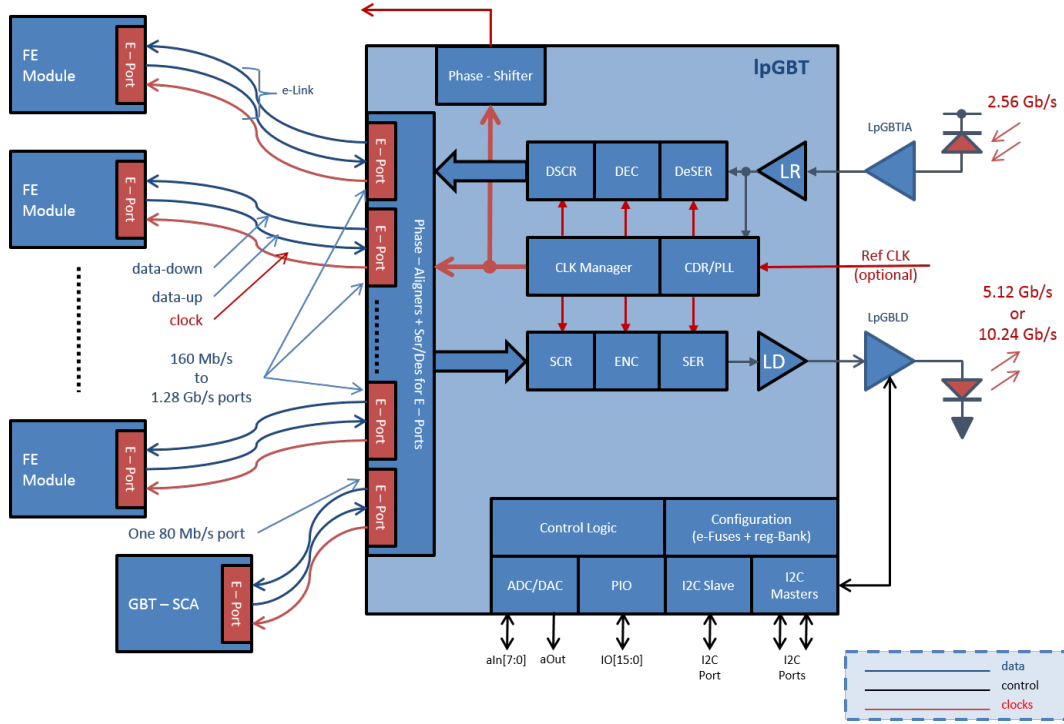


Figure 5.5: Architecture of the lpGBT ASIC [59].

high-speed serial data streams are implemented using dedicated circuitry, including a high-speed line driver for transmission and an equalizer to compensate for electrical signal degradation on reception. Notably, the lpGBT implements dedicated Clock and Data Recovery (CDR) and Phase-Locked Loop (PLL) circuits that extract the embedded clock from the incoming serial data stream and generate an internal 40 MHz clock. This enables the device to operate without a dedicated reference clock, provided that a valid downlink stream is available. Otherwise, when operated in a transmit-only configuration, an external clock source is required. The high-speed interface is designed to connect directly to Versatile Link+ (VL+)-compliant [64] optoelectronic modules, such as the VTRx+.

Towards the FEs, the lpGBT provides multiple parallel eLinks. Each eLink is driven by a dedicated ePort and consists of three differential pairs: one for data transmission, one for data reception, and one for reference-clock distribution. These links feature programmable signal amplitude to support requirements on transmission distance, bit rates, and power consumption. In total, the lpGBT supports 28 input eLinks and 16 output eLinks, organized into groups of four known as eGroups. The number of active input eGroups depends on the selected uplink data rate and FEC scheme, as summarized in Table 5.2.

Uplink Line Rate	5.12 Gbit/s		10.24 Gbit/s	
FEC schema	FEC5	FEC12	FEC5	FEC12
Active eGroups	7/7	6/7	7/7	6/7

Table 5.2.: Overview of active eGroups depending on the configured uplink line rate and the FEC scheme. The FEC5 and FEC12 denote different Reed-Solomon FEC schemes, where FEC5 can correct up to 5 (or 10 consecutive) wrong bits and FEC12 can correct up to 12 (or 24 consecutive) wrong bits at the respective line rates [59].

Moreover, the data rate of each eLink can be configured. Increasing the bandwidth of an eLink results in a decrease in the number of active eLinks in the respective group, causing a trade-off between maximum per-link bandwidth and available channel count (see Table 5.3).

Direction	Downlink			Uplink					
Line Rate [Gbit/s]	2.56			5.12			10.24		
Active eLinks per group	4	2	1	4	2	1	4	2	1
eLink Bandwidth [Mbit/s]	80	160	320	160	320	640	320	640	1280

Table 5.3.: Overview of the achievable eLink bandwidth depending on eGroup configuration and the uplink and downlink line rates [59].

In addition to its primary function as a data SerDes, the lpGBT implements additional capabilities for the control and monitoring of detector hardware, such as:

- Three generic I2C masters
- A 16-bit programmable I/O port
- A 10-bit ADC with eight multiplexed inputs
- On-Chip temperature monitoring
- Up to 8 multiplexed current sources to drive external temperature sensors
- A 8-bit DAC

lpGBT Configuration and Control

The lpGBT is designed to operate in several distinct modes and supports a wide range of application-specific settings, including configurable line rates, eGroup con-

figurations, clock speeds, driver strengths, and more. Consequently, configuration is imperative to ensure the device's intended functionality.

To manage these configurations, the lpGBT implements a total of 498 internal 8 bit registers, each assigned to a unique 16 bit address. Of these, the first 336 registers are writable and initialized to zero upon power-up, while the remaining registers are read-only and primarily used for device monitoring and diagnostics. Register access is coordinated by a simple bus arbiter, with access distributed among four distinct functional blocks: the e-Fuses block, the Read-Only Memory (ROM) block, the I2C slave, and the Serial Control (SC) block. The e-Fuses and ROM blocks can only write to registers, whereas the I2C slave and SC interfaces can also read from the read-only registers to retrieve internal status information.

During the power-up sequence, a dedicated Power-Up State Machine (PUSM) initiates the configuration process and, depending on the selected boot mode, selects either the e-Fuses or the ROM as the source for register initialization. The e-Fuses are organized into eight groups, with each group containing 32 fuses. Each group is responsible for programming one 8 bit register. This configuration enables the initialization of up to 256 registers during power-up. Conversely, the ROM offers a rudimentary, predefined configuration that functions as a backup mechanism. Following the completion of the initialization process, the e-Fuses and the ROM are deactivated, and control of the register bus is transferred to the I2C slave and the SC interfaces. In contrast to the fixed-priority arbitration during start-up, access between the I2C and SC blocks is dynamically shared, with the arbiter assigning priority to the I2C slave when parallel access requests occur. The registers that have not been initialized via the EEPROM or the ROM must be configured afterward via the I2C or SPI interface.

To support multi-lpGBT systems and avoid conflicts during configuration, both the I2C and SC interfaces utilize the same, fully configurable 7 bit address. The four least significant bits of the address are set using the external pins **ADR3** through **ADR0**, while the three most significant bits are defined by the **ChipAddressBar[2:0]** field in register **0x36**. The configurable addressing schema is imperative in systems with several lpGBTs on the same bus.

I2C Configuration. The I2C slave port of the lpGBT facilitates both read and write operations on the device's configuration register, regardless of the ASIC's operational mode. The slave interface is designed to support addressing of either a single register or a block of consecutive registers. To execute an access, the master transmits a valid 7 bit slave address, the corresponding R/W bit, and the register address. For write operations, the corresponding data bytes are subsequently transmitted by the master, with the register address being automatically incremented

after each 8-bit word. For read operations, the master issues a repeated **START** condition, along with the slave address and the read bit. The lpGBT then provides the requested data. After each frame is transmitted, the address is automatically incremented, and the next data frame is sent until the master issues a **STOP** condition, thereby enabling multi-register reads within a single sequence.

SC Configuration. In contrast to the I2C approach, configuration of the lpGBT via the SC channel requires several prerequisites to ensure a reliable connection over the Internal Control (IC)/External Control (EC) channel. Most importantly, the internal PLL must be locked, which can be accomplished by loading a minimal configuration from the built-in ROM or from the programmed e-fuses. This preliminary configuration provided the fundamental CDR/PLL, equalizer, line driver, and IC/EC channel settings, thereby enabling the ASIC to establish communication with the backend and to lock to a reference clock. The remaining configuration must then be set through either the IC channel (transceiver mode) or the EC channel (transmitter mode).

Frame Format

The lpGBT employs distinct frame formats for the downlink and uplink, which share a similar overall structure but differ in detail due to their respective data rates (2.56, 5.12, or 10.24 Gbit/s) and FEC schemes (FEC5 or FEC12). In total, five frame types are defined: one fixed downlink frame format and four possible uplink frame formats, depending on the chosen link configuration.

Downlink Frame The downlink uses a fixed 64 bit frame, transmitted synchronously with the LHC bunch-crossing period (25 ns), yielding a constant data rate of 2.56 Gbit/s. The frame is divided into several fields, as illustrated in Fig. 5.6, and summarized as follows:

- **Header:** Split and interleaved to enhance transmission robustness, the header comprises three sub-fields:
 - **H Field:** A 4 bit sequence ($H[3:0] = 4b'1001'$) marking the frame start. The header ensures DC balance, detection redundancy, and reliable frame alignment.
 - **IC Field:** 2 bit for the IC channel, used to configure and control the lpGBT, operating at 80 Mbit/s.

- **EC Field:** 2 bit for the EC channel, operating also at 80 Mbit/s. The signal is made available on the differential output pair EDOUTECP and EDOUTECN.
- **D Field:** The 32 bit user-data payload is divided into 4 Byte, each corresponding to an eGroup. The eLink data rate and the distribution of the respective 8 bit among the eLinks are directly related and depend on the eGroup configuration.
- **FEC-Field:** A 24 bit FEC code used for error detection and correction.

To ensure the reliable operation of the CDR circuit, which requires frequent signal transitions, the IC, EC, and D fields are scrambled before being inserted into the frame. Transmission is Most Significant Bit (MSB)-first, with H(3) as the leading bit and FEC(0) as the trailing bit.

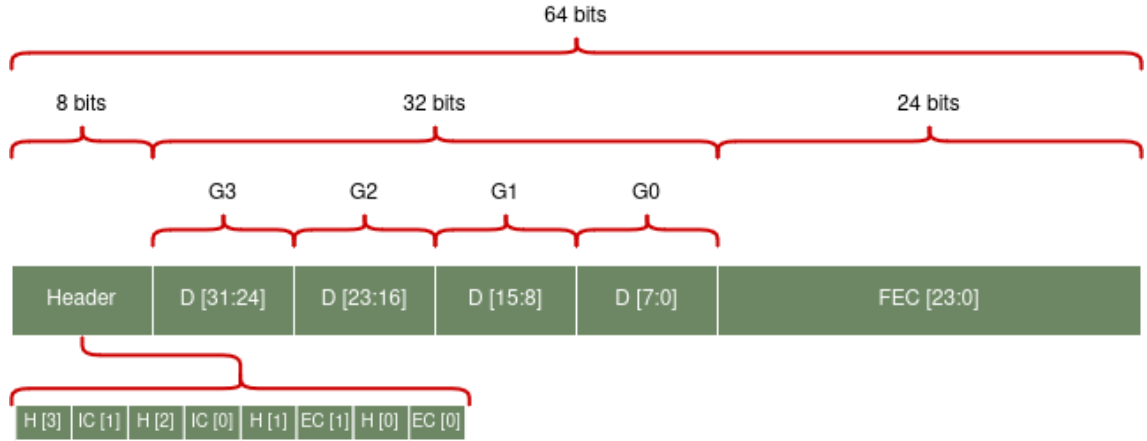


Figure 5.6: Overview of the lpGBT downlink frame structure before interleaving. Adapted from Ref. [59].

Uplink Frame Unlike the fixed downlink format, the uplink frame structure depends on two parameters: the optical uplink data rate and the selected FEC scheme. Although there are four defined frame types, they all share the same general layout. Differences arise from the total frame length, length of the configured FEC scheme (FEC5 or FEC12), and the mutually dependent length of the user-data payload as summarized in Table 5.4.

Since the EMP uses a fixed uplink configuration that operates at 10.24 Gbit/s with FEC5 encoding, the lpGBT must be configured accordingly. Consequently, only the

Data rate	Frame length	FEC	D Field	FEC Field	LM Field
5.12 Gbit/s	128 bit	FEC5	112 bit	10 bit	0
5.12 Gbit/s	128 bit	FEC12	96 bit	24 bit	2
10.24 Gbit/s	256 bit	FEC5	224 bit	20 bit	6
10.24 Gbit/s	256 bit	FEC12	192 bit	48 bit	10

Table 5.4.: Uplink frame structure as a function of data rate and FEC scheme [59].

relevant frame type for EMCI-EMP operations will be discussed. The uplink frame is illustrated in Fig. 5.7 and is based on the following components:

- **Header:** Split and interleaved for reliability, it contains:
 - **H Field:** A 2 bit sequence ($H[1:0] = 2b'10'$) that delimits the start of the frame.
 - **IC Field:** 2 bit for the uplink IC channel, operating at 80 Mbit/s.
 - **EC Field:** 2 bit the uplink EC channel, operating at 80 Mbit/s. The signal is received on the differential input pair EDINECP and EDINECN.
- **LM Field:** 10 bit Latency Measurement (LM) field, used to estimate the round-trip latency of the optical link.
- **D Field:** A 224 bit user-data payload, divided into seven 32 bit chunks, each mapped to a specific lpGBT eGroup.
- **FEC Field:** A 24 bit FEC code used for error detection and correction.

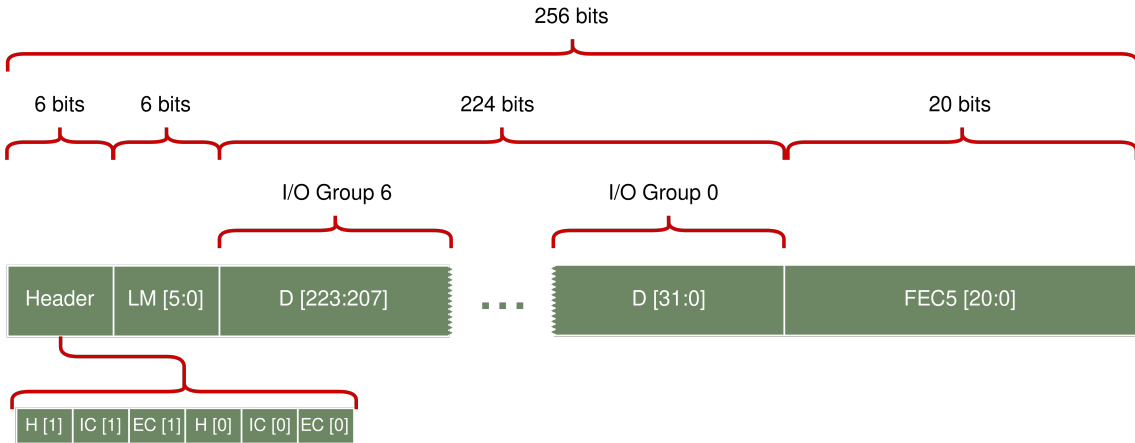


Figure 5.7: Overview of the lpGBT uplink frame structure for the configuration using FEC5 and an uplink data rate of 10.24 Gbit/s.

5.3. EMP: Design and Technical Specification

The EMP constitutes BE of the novel DCS interface. It serves as a central processing and data aggregation platform, interfacing with multiple EMCIs in a star topology. In total, the EMP supports up to twelve EMCIs, each connected via high-speed optical links. Communication with the experiment's SCADA system is implemented over a LAN through a redundant Ethernet interface.

The EMP is designed for deployment in environments without elevated radiation levels and is based entirely on COTS components. The baseboard is built around a TE0807 SoM from Trenz Electronics, which features an AMD Zynq UltraScale+ Multiprocessing System-on-Chip (MPSoC). The design of this baseboard is compatible with most interfaces and I/Os of the SoM, enabling connectivity to a wide range of analog and digital peripherals. The implementation of high-speed optical communication with the EMCIs uses two Samtec FireFly modules: a 12-channel transmitter and a 12-channel receiver. These modules are routed directly to the MPSoC's GTH Multi-Gigabit Transceivers (MGTHs), ensuring full compliance with the VL+ optical link specification for high-energy physics environments.

In operation, the EMP supervises and configures all connected EMCIs, aggregates and processes FE monitoring data, and implements control and readout logic for the FE electronics. The subsequent subsections offer a comprehensive overview of the EMP's hardware architecture and communication capabilities.

5.3.1. TE0807 MPSoC Module

The following section provides a brief overview of the TE0807 SoM module as described in the Technical Reference Manual (Ref. [65]). The TE0807 series from Trenz Electronic, as depicted in Fig. 5.8, is a compact platform measuring 52 x 76 mm. It integrates an AMD Zynq UltraScale+ MPSoC with all essential support components and circuits. Each module offers up to 8 GB of 64 bit DDR4 SDRAM and up to 512 MB of dual-parallel SPI flash memory. The MPSoC itself is housed in a 900-pin BGA package, with access to high-speed transceivers, PL, and processing system interfaces. Most signals of the MPSoC are routed through four 160-pin Board-to-Board (B2B) SoM connectors, which expose MIO pins, FPGA I/O banks, clock signals, high-speed serial transceiver lanes, and auxiliary signals such as JTAG, UART, and power management interfaces. This modular design of the TE0807 series facilitates straightforward integration with custom carrier boards and supports the implementation of application-specific interfaces.

The TE0807 implements a complete on-board power management system. All required MPSoC power rails are generated locally using 14 switching regulators and 13

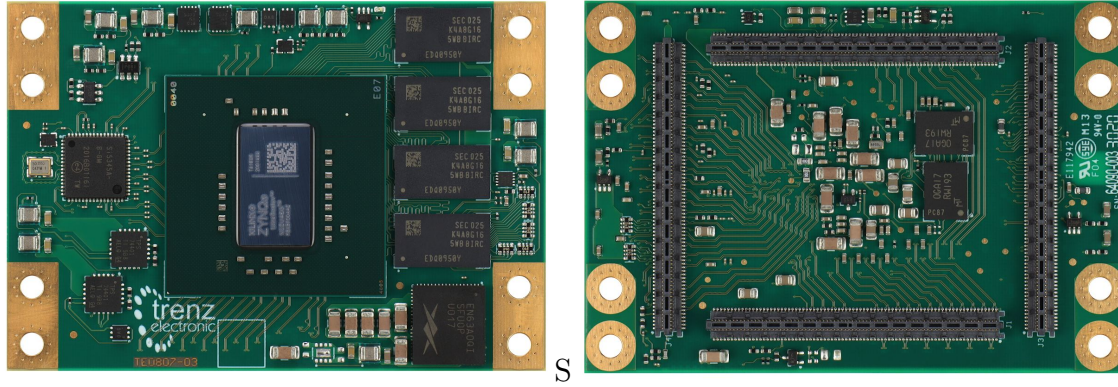


Figure 5.8: Front and back side of the TE0807 MPSoC module [66].

Low-Dropout Regulatorss (LDOs). The four major power domains of the MPSoC - the Low Power Domain (LPD), Full Power Domain (FPD), PL, and battery - are supplied and controlled independently. As a result, the module requires only a single external 3.3 V input, simplifying carrier board design by eliminating the need for additional sequencing or domain-specific regulation.

Local high-speed clock generation is enabled by an onboard Si5345 jitter-attenuating clock multiplier that generates programmable low-jitter clocks. These clocks are routed directly to the GTH high-speed transceiver banks, enabling deterministic, reliable clock distribution. The Si5345 is supplied with reference inputs from an on-board 50 MHz crystal and a 25 MHz Low Voltage CMOS (LVCMOS) oscillator, while additional clock inputs and outputs are available via the B2B connector.

In summary, the TE0807 module series integrates all critical infrastructure required by the MPSoC. When mounted on a custom baseboard, the carrier only needs to provide the external 3.3 V supply and to route the I/O, MIO, clock, and power signals to the dedicated physical layer. All remaining functions, including DDR4 SDRAM initialization, power sequencing, system clocking, and flash boot, are managed on the module.

The EMP is compatible with any module from the TE0807 family. However, in its standard configuration, the TE0807-04-4BE81-AK module is employed. This module integrates an AMD Zynq UltraScale+ ZU4EG-1E MPSoC, 4 GB of DDR4 SDRAM, two 64 MB dual-parallel SPI flash devices for boot and configuration, and a 2 kbit serial EEPROM containing a unique EUI-48-compliant MAC address. Via the module's B2B connectors, a total of 204 PL I/Os, 65 PS MIOs, 4 PS-GTR, and 16 GTH high-speed transceivers, as well as standard interfaces (I2C, UART, and JTAG) are accessible.

AMD Zynq UltraScale+ ZU4EG MPSoC

The AMD Zynq UltraScale+ ZU4EG MPSoC is a heterogeneous device that combines a multi-core PS with a PL fabric, providing a compact and flexible platform for embedded control and monitoring applications. The following gives a brief overview of the key architectural components, following the Zynq UltraScale+ MPSoC Data Sheet (Ref. [67]).

The device's PS is comprised of three primary subsystems:

- A quad-core, 64 bit ARM Cortex-A53 Application Processing Unit (APU) supporting execution of an embedded operating system such as Linux.
- A dual-core ARM Cortex-R5F Real-Time Processing Unit (RPU), optimized for deterministic control and low-latency tasks.
- A Platform Management Unit (PMU), implemented as a discrete microcontroller, is responsible for power management, thermal monitoring, and fault handling. The PMU functions independently of the APU and RPU, providing support for secure boot, system initialization, and runtime supervision.

In addition to its primary subsystems, the PS integrates a wide range of peripheral controllers, including UART, I2C, SPI, CAN, USB 2.0, and Gigabit Ethernet. These components can be configured for external interfacing through the MIO subsystem. Furthermore, the PS includes four PS-GTR transceivers, which support standard interfaces such as PCIe, SATA, USB 3.0, and DisplayPort.

The PL fabric complements the PS, offering configurable logic cells, distributed Random-Access Memory (RAM), block RAM, Digital Signal Processing (DSP) slices, PLLs, and 16 GTH MGT. These GTH transceivers are tightly integrated into the PL and operate independently of the standard I/O banks. They support high-speed serial communication for custom protocols and data links within the user-defined logic, supporting line rates from 500 Mbit/s to 16.375 Gbit/s. In essence, the PL facilitates application-specific hardware implementations, including protocol bridges, custom processing pipelines, and hardware accelerators.

In addition to the fixed-function peripheral controllers present in the PS, the PL can host "soft" peripherals implemented using AMD IP-cores. These IP-cores provide predefined hardware blocks that implement the peripheral controller's HDL design within the PL fabric. They can be connected to external I/O pins, and their interface to the PS is via the Advanced eXtensible Interface (AXI) protocol (see Sec. 6.2.1).

Furthermore, the UltraScale+ architecture integrates two System Monitors (SYS-MON), one in the PS and one in the PL. Each of these devices is equipped with an

ADC and on-chip sensors to monitor temperatures, supply voltages, and external voltages. These measurements can be accessed via the Advanced Peripheral Bus (APB), enabling system diagnostics and fault handling.

5.3.2. EMP Baseboard

The EMP baseboard is a custom carrier for the TE0807 SoM. It is designed to expose the SoM's digital and analog communication interfaces and GPIOs, thus extending the Zynq's PS and PL connectivity to dedicated physical layers and connectors. This enables the EMP to be used in diverse embedded monitoring and control applications within the DCS. Figure 5.9 presents a computer-generated 3D model of the baseboard, showing the placement of key connectors and components.

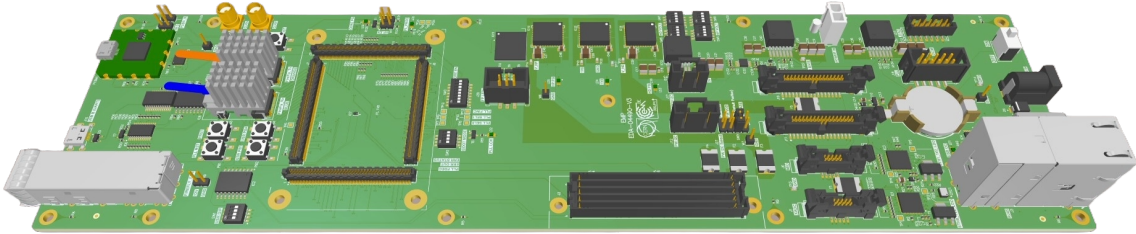


Figure 5.9: Computer generated 3D image of the EMP baseboard [68].

A high-level block diagram of the EMP baseboard is provided in Figure 5.10, illustrating how the Zynq UltraScale+ MPSoC module interfaces with the baseboard's memory devices, optical transceiver modules, mezzanine connectors, and slow-control peripherals. The following provides a comprehensive overview of the carrier's functional domains and peripherals. Please refer to the Electronics Specifications of the EMP (Ref. [69]) for more information.

PS Peripheral Interfaces

The EMP baseboard provides the physical infrastructure necessary to access the peripheral controllers integrated into the Zynq's PS, including Physical Transceivers (PHYs), supporting circuitry, and connectors. These PS peripherals are exposed through the MIO subsystem, which maps controller signals to dedicated MPSoC I/Os, which are then routed to the carrier via the SoM's B2B connector. The instantiation of these controllers and the MIO pin assignments are defined in the hardware design and propagated to the board-specific device tree, enabling

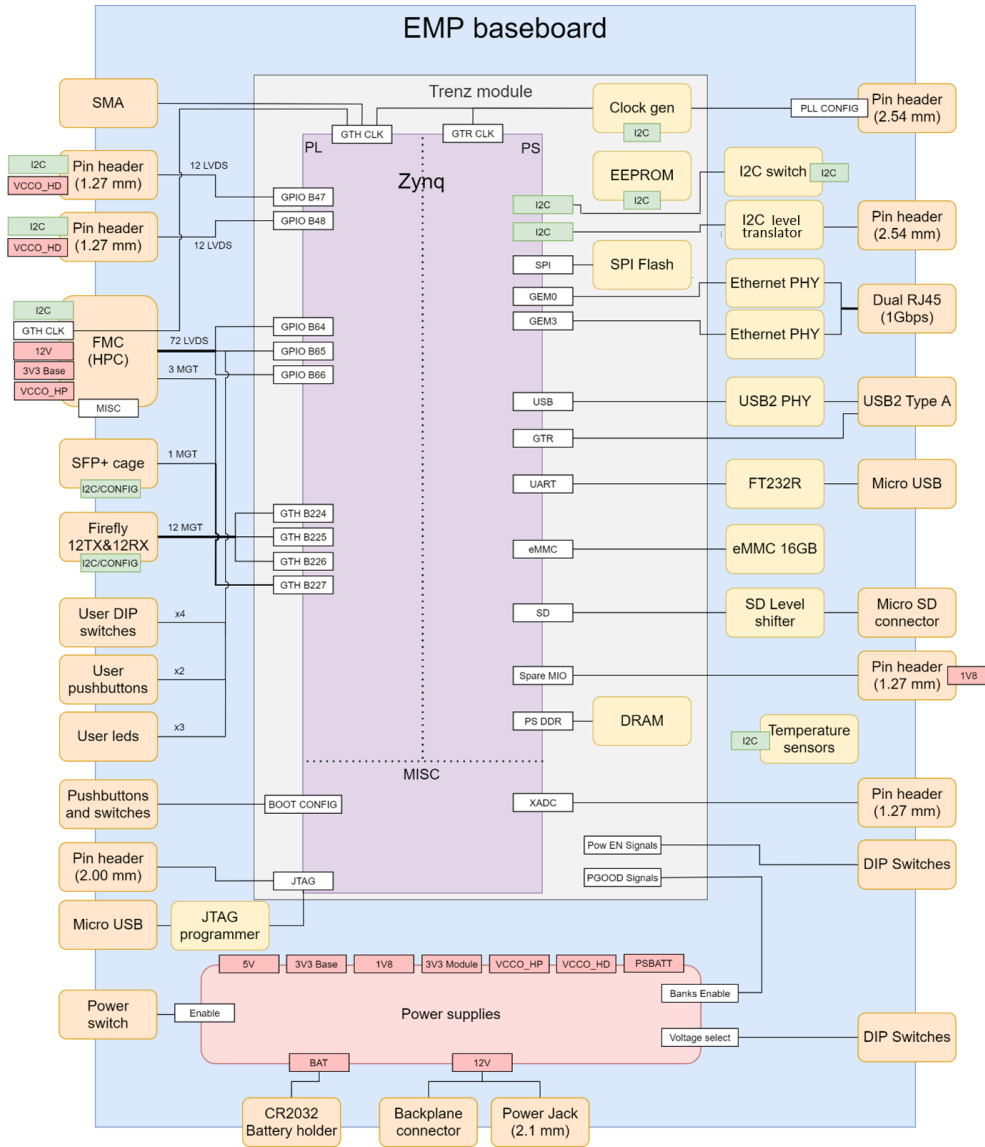


Figure 5.10: Block diagram of the EMP baseboard, illustrating the integration of the Zynq on the TE0807 MPSoC module via the B2B connectors and the external interfaces provided by the EMP baseboard [69].

straightforward integration of PS interfaces with baseboard components. The following overview summarizes the PS-based interfaces implemented on the EMP baseboard:

- Two redundant 1 Gbit/s Ethernet interfaces are provided via a stacked (dual) RJ45 connector. Each interface is implemented by an Ethernet-to-RGMII PHY connected to one of the Zynq PS's Gigabit Ethernet MAC (GEM)

controllers. The RGMII PHY signals and the shared Management Data Input/Output (MDIO) bus are routed through MIO pins.

- A USB 2.0 (Type-A) connector is available on the baseboard. The USB 2.0 functionality is facilitated by a dedicated PHY chip connected to the PS via MIOs.
- The PS supplies two I2C master controllers, both of which are routed to the baseboard through MIOs. The first I2C master controls an octal bidirectional Inter-Integrated Circuit (I2C) switch, which in turn connects to various baseboard and SoM peripherals. These peripherals include the EEPROM, the Si5345 clock multiplier, the Enhanced Small Form-factor Pluggable (SFP+) module, both FireFly modules, and three temperature sensors situated in thermally sensitive regions of the board. The second I2C master is routed through a level-shifter to a 4-pin locking connector and is reserved for monitoring the EMP power supply.
- Furthermore, three spare MIO lines (not assigned to a dedicated PS peripheral) are routed to a 10-pin header. Depending on the application's specific requirements, these lines can be redistributed to alternative PS peripherals.

PL Configuration Interfaces

The EMP baseboard supports a range of programming and debugging workflows, including network-based boot and remote configuration. In addition, the EMP is equipped with two physical JTAG interfaces, both of which access the same JTAG Test Access Port (TAP) routed to the Zynq device:

- An onboard USB-JTAG module (PCBM1) is exposed via a Micro-USB connector for convenient on-bench programming.
- A 14-pin header (J23) is provided for use with a dedicated programming cable or production programmer.

Xilinx devices use the JTAG port as the primary interface for several essential activities. During the development and implementation stages, Vivado's Hardware Manager uses the JTAG to identify the target device, download bitstreams, and attach on-chip debug probes, such as the Integrated Logic Analyzer (ILA), to capture internal signals at system speed. Additionally, JTAG enables indirect programming of configuration flashes via the FPGA JTAG path.

Optical High-Speed Interfaces

The EMP can support up to 13 high-speed optical links, leveraging the Zynq MPSoC MGTs as the physical-layer SerDes. Twelve of these MGTs are routed to two Samtec high-speed connector systems on the EMP baseboard, which are intended to host a pair of 12-channel Samtec FireFly transceiver modules [70]. Each FireFly module is equipped with a 12-fibre pigtail terminated with a MTP/MPO-12 short-boot male connector. The thirteenth MGT is routed to a standard SFP+ cage on the baseboard. Although this SFP+ link is compatible with the VL+ specification, it is not intended for EMCI connectivity; instead, it is reserved for optional debug, monitoring, or control functions.

All three optical modules provide a dedicated Two-Wire Interface (TWI) for control and monitoring purposes. In practice, this TWI implements an I2C-compatible management bus for accessing configuration registers, status information, and diagnostic data. SFP+ devices typically implement the Digital Diagnostic Monitoring Interface (DDMI) via a standardized EEPROM map defined in the SFF-8431/SFF-8472 specifications. In contrast, the FireFly management interface adheres to the register model outlined in the SFF-8626 specifications while incorporating vendor-specific extensions to enhance control and diagnostic functionality.

GPIOs

The FPGA fabric of the MPSoC exposes two distinct classes of programmable I/O banks used on the EMP baseboard: High-Performance (HP) I/O banks, optimized for high-speed interfaces, and High-Density (HD) I/O banks, intended for dense, lower-speed connectivity. HP banks are designed to support LVDS and other high-speed differential standards, operating at low I/O voltages within the range of 1.0 V to 1.8 V. Conversely, HD banks offer a broad single-ended voltage range spanning 1.2 V to 3.3 V, well-suited for general-purpose signaling and legacy peripherals.

The ZU4EG MPSoC chipset is equipped with 48 HD and 156 HP GPIOs, which are exposed via the B2B connector on the EMP baseboard. The HD GPIOs of banks 45 and 45 are routed to two generic 34-pin headers, while 144 of the 156 HP GPIOs are routed to the EMP's HPC FMC connector. This FMC connector adheres to the VITA 57.1 pinout standard and power supply conventions to ensure compatibility with commercial FMC modules. Consequently, unpopulated pins on the EMP side FMC are left unused to prevent accidental shorts or unintended signals. The remaining 12 HP GPIOs are retained on the baseboard for dedicated control functions, such as resetting the I2C switch or controlling status LEDs.

The HD and HP banks are supplied from independent supply rails and can be configured independently. To this end, the EMP baseboard is equipped with two dedicated DC-DC converters, each controlled by a pair of DIP switches. These switches enable the user to adjust the switching frequency and output voltage according to predefined DC-DC output settings.

Memories

In addition to the on-module DDR4 SDRAM, QSPI flash, and EEPROM integrated on the TE0807 SoM, the EMP baseboard provides additional non-volatile storage via a 16 GB eMMC device connected to the Zynq PS via the dedicated SD/eMMC interface. Furthermore, the baseboard integrates a microSD card socket that interfaces with the Zynq PS through a bidirectional level shifter, thereby ensuring proper voltage translation and enabling the use of standard 3.3 V microSD cards.

Both storage interfaces support boot functionality and can host the operating system's root filesystem. The eMMC provides a robust, solder-down storage solution well-suited for production use. Nevertheless, programming the device is only possible from the PS itself, as it lacks an external programming interface. Conversely, the microSD socket facilitates media exchange, making it well-suited for rapid development, testing, or recovery.

Zynq Boot Modes

The Zynq's BootROM can boot the system from various sources, including QSPI, SD, eMMC, USB 2.0, PS JTAG, and external NAND flash. The EMP is designed to support several of these options, with the relevant configuration selected via a dedicated 4 bit DIP switch that directly connects to the Zynq's boot mode pins. This enables selecting the desired boot source before the system is powered up. A complete list of supported options and their associated DIP switch settings is provided in Table 5.5.

Power Distribution

The EMP baseboard accepts an external supply via either a 4.2 mm pitch Molex connector, or a 2.1 mm barrel jack. The input voltage range supported on both connectors is 6 V to 36 V, with a nominal and recommended input level of 12 V. While the 2.1 mm barrel jack is primarily intended for laboratory or stand-alone setups, the

Boot Mode	Mode Pins [3:0]	Pin Location
PS JTAG	0000	JTAG
SD1	0101	MIO[51:43]
Quad-SPI (32b)	0001	MIO[12:0]
eMMC	0110	MIO[22:13]

Table 5.5.: Settings of the DIP switches on the EMP baseboard for configuring the supported boot modes [69].

Molex connector is designated as the standard input in production setups, where the EMP is housed within a crate and powered by a local supply.

The primary 12 V input is stepped down by three DC–DC converters to generate the principal supply domains required by the SoM and peripheral circuitry:

- 3.3 V supply dedicated to baseboard components
- 3.3 V supply dedicated exclusively to the SoM
- 5 V supply for additional baseboard peripherals

The 5 V rail is further stepped down to 1.8 V by a downstream converter to feed auxiliary baseboard circuits. The 1.8 V domain is also used as input to two LMZ30604 DC–DC regulator modules, whose outputs supply the HD and HP I/O banks of the PL. Each LMZ30604 output voltage is user-selectable via two independent pairs of 4 bit DIP switches, allowing for dynamic configuration of the HD and HP I/O voltages.

5.4. EMP Chassis

The EMP chassis is a standard 19-inch, 1U rack-mount unit designed to house up to three EMP baseboards along with a commercial redundant power supply. Each EMP is powered via the supply’s 12 V rails, which are connected to the baseboard through the 4.2 mm pitch Molex connector. The chassis is designed for installation in a standard 19-inch production rack equipped with horizontal backdoor ventilation and Uninterruptible Power Supply (UPS) backed power supply.

The crate design was primarily driven by the EMP’s significant heat dissipation, particularly from the SoM and FireFly modules. During an initial series of tests with the first EMP prototype, the behavior of the EMP and especially the FireFly modules was studied in the absence of a forced airflow. Therefore, the Xilinx Integrated Bit Error Ratio Tester (IBERT) IP-core was used to apply a representative load to the PL and to keep all GTH transceivers continuously active, while

an artificial CPU load was generated on the PS using the Linux command-line tool **stress**. During the test, all on-board temperature sensors were read out and logged at regular intervals, enabling later analysis of the data.

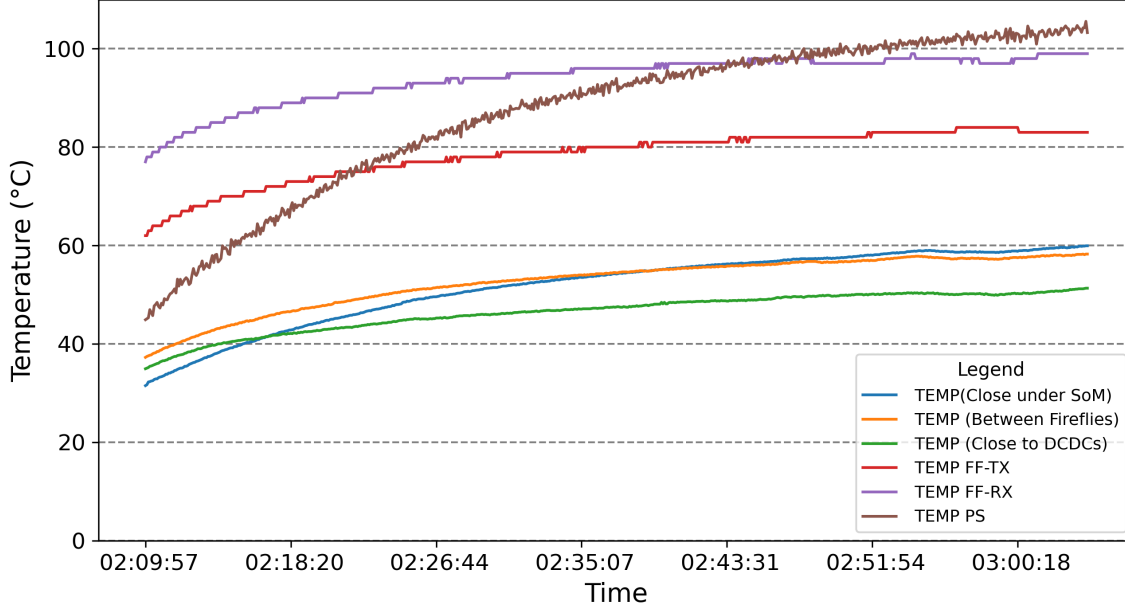


Figure 5.11: Evolution of the measured temperatures on the EMP baseboard without forced airflow. The plot shows the measured values for the I2C temperature sensors (close below the SoM, between the FireFly modules, and near the DC/DC converters) as well as the internal temperatures of the FireFly modules and the Zynq PS.

The analysis of the recorded data is presented in Figure 5.11. It shows that the measured PS temperature increases monotonically throughout the one-hour test run and reached approximately 105 °C at the end of the test, i.e., already beyond the recommended maximum junction temperature of 100 °C. Moreover, the corresponding curve has not yet reached saturation, implying that prolonged operation under these conditions would further increase the junction temperature and approach the over-temperature shutdown threshold. At the same time, the FireFly receiver module reaches about 100 °C while the transmitter stabilizes around 82 °C, i.e. both devices are operated more than 10 °C above the maximum temperature of 70 °C, and more than 30 °C above the recommended temperature for continuous operation, causing accelerated aging of the both modules.

In addition, the infrared image of the FireFly and SoM, as presented in Figure 5.12, shows maximum surface temperatures of only about 50 °C for the FireFly heatsink and 90 °C for the SoM heatsink. This corresponds to an excess temperature difference of 20 °C between the on-silicon temperature sensor and the external heatsink,

indicating substantial thermal resistance and thus inefficient heat dissipation. These observations demonstrate that reliable long-term operation of the EMP requires a controlled, forced airflow and motivated a dedicated thermal study under realistic conditions in an operational environment.

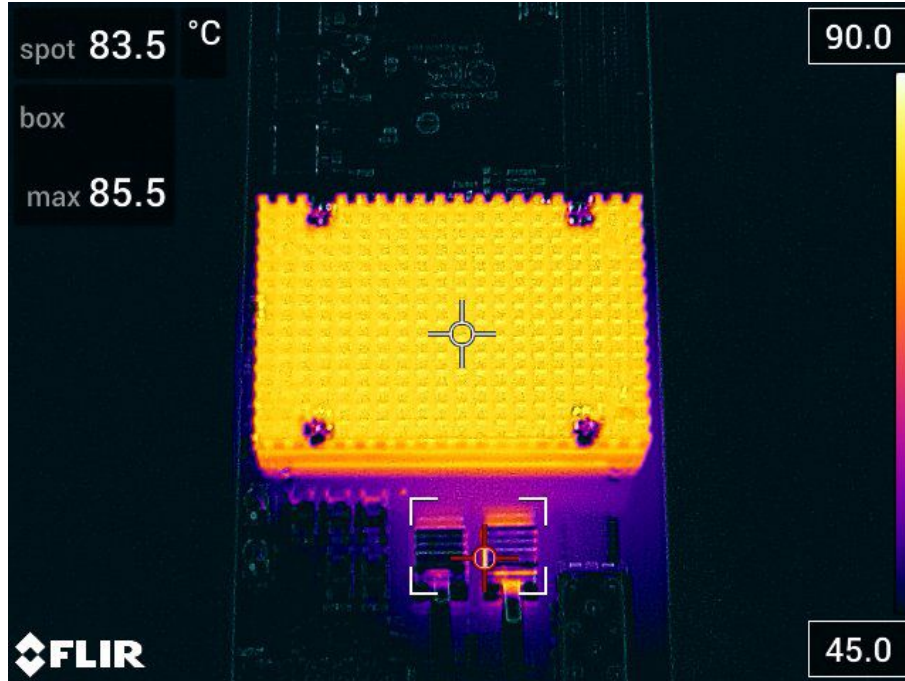


Figure 5.12: Infrared image of the FireFly and SoM heatsink during the no-airflow test.

The conducted study, including methodology and detailed results, is presented in Appendix A. In summary, to validate the thermal behavior of the EMP chassis under operational conditions, a fully populated prototype was deployed in an ATLAS counting room and subject to a typical operational load. Based on the measured temperature profiles, several mechanical modifications were implemented to optimize the internal airflow and to reduce thermal isolation of critical components. The updated design was subsequently reevaluated under identical conditions and was confirmed to maintain all components within the specified operational margins, thereby validating the crate layout in an operational environment. Figure 5.13 shows a schematic view of the final EMP chassis layout.

Optionally, the chassis can be equipped with three front-panel-mounted fans, each aligned with one EMP baseboard. While not mandatory in standard rack installations with functional backdoor ventilation, these fans ensure sufficient airflow to maintain critical components within the specified limits, even in the event of back-

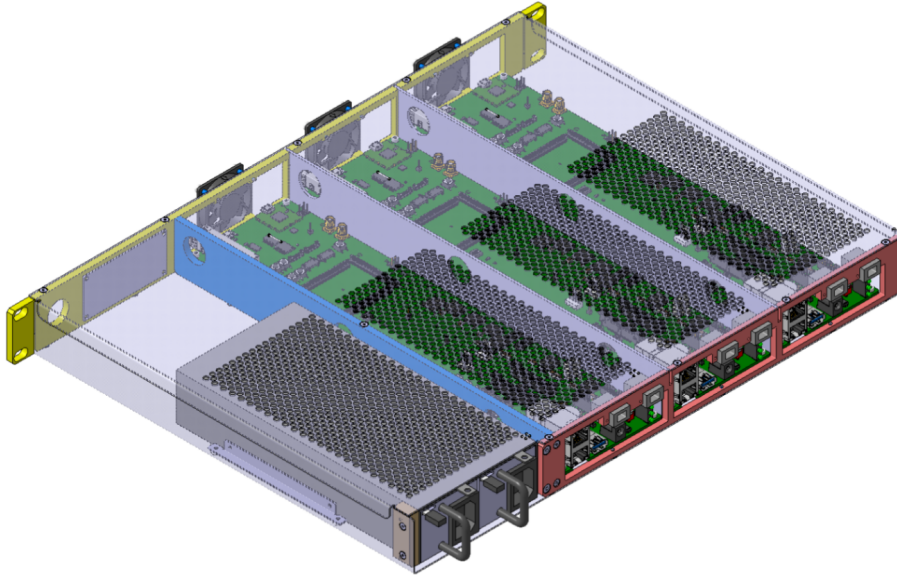


Figure 5.13: Computer-generated image of the 19-inch rack-mount EMP chassis, showing three slots reserved for EMP boards and a fourth slot housing a redundant server power-supply [71].

door ventilation failure. Their use is only imperative when the crate is operated in standalone laboratory setups without rack-based cooling.

6. The EMCI-EMP Ecosystem

The EMCI-EMP system is a versatile infrastructure for implementing a wide range of embedded control, monitoring, and data-processing applications within the DCS. Leveraging the heterogeneous Zynq MPSoC architecture, it combines the hardware flexibility of FPGA-based PL with the software capabilities of the PS. This enables the EMP to accommodate a wide range of FE devices, supporting various communication standards and processing requirements.

Given the variety of targeted subsystems within the DCS (see Sec. 5.1.3), the diversity of application requirements, and the complexity of EMP hardware, a consistent development methodology and a standard baseline for software and firmware components are essential. To address this, a modular reference framework, the EMCI-EMP ecosystem [72], has been developed, delineating standard interfaces, communication flows, and reusable building blocks. The objective is to promote a certain degree of standardization across various target applications. As such, it aims to accelerate system integration, enhance maintainability, and ensure compatibility with existing hardware requirements. The components comprising the framework are designed as reusable building blocks that can be easily adapted to application-specific requirements.

The subsequent section introduces the architecture of the reference framework. It provides an overview of the most common software/firmware stacks for various target applications using the EMCI-EMP system. The individual components are subsequently described, along with an overview of their functionality, the needs they address, and their integration into the overall system.

6.1. Architectural Overview

The configuration of each EMP instance is determined mainly by its operational context. However, in practice, each deployed system integrates standard applications alongside FE-specific applications. To this end, the reference framework offers a modular architecture that incorporates standard tools, reusable firmware IP-cores, and common hardware support.

The design philosophy of the EMCI-EMP ecosystem ensures that differences in functional requirements between multiple FE systems can be addressed without re-engineering the entire platform. Therefore, it promotes reusability of core components while supporting the rapid adaptation of the EMP to new FE hardware

or operational constraints. This not only accelerates system deployment but also ensures that different subsystems within the DCS share a consistent, well-defined interface to the distributed control infrastructure.

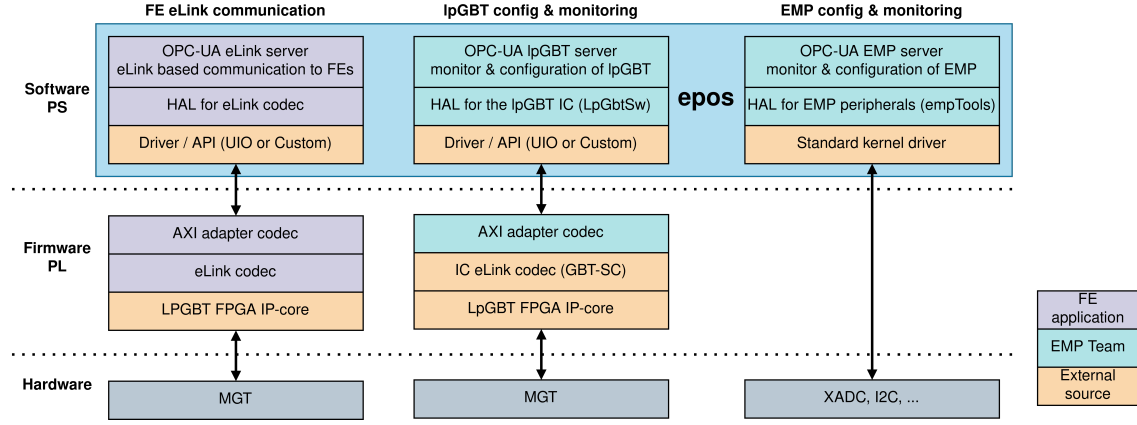


Figure 6.1: Modular architecture of the EMCI-EMP interface, showing building blocks for specific target applications. Orange denotes components from external sources; turquoise indicates modules provided as part of the reference software; lila represents custom building blocks to be developed for the respective FE hardware. Adapted from Ref. [73].

Figure 6.1 shows the modular architecture of the EMCI-EMP ecosystem, illustrating typical system applications and their constituent building blocks. The diagram organizes each target application as a vertical stack of multiple software, firmware, and hardware layers, while also delineating the respective development responsibilities. While the implementation of FE-specific eLink applications remains the full responsibility of the subsystem developers, the control and monitoring stacks for the lpGBT and EMP are delivered as standardized components. Providing these two stacks in a unified manner minimizes redundant development effort across deployed EMCI-EMP instances, thereby ensuring consistency in system control, monitoring, and diagnostics. An overview of the available components is summarized as follows:

- **Firmware:** The EMP FPGA firmware reference design implements the processing pipeline for lpGBT monitoring and control. This pipeline encompasses the encoding and decoding of lpGBT frames, the high-speed MGT interface to the optical transceiver modules, and custom AXI adapters that facilitate communication with the PS. Given the similarity in functionality of FE-specific eLink communication, such as the lpGBT channel encoding/decoding, the reference design is organized into modular IP-cores, enabling subsystem developers to integrate their custom logic through well-defined adoption points.

- **Embedded Software and Utilities:** General requirements common to all applications, such as access to shared hardware resources (e.g., Ethernet connectivity) and standardized software components (e.g. Operating System (OS), drivers, tooling), are addressed through the EMP Operating System (epos). It provides a comprehensive runtime environment, encompassing a bootable system image, a Linux kernel, device tree implementations for the EMP hardware, and a set of device drivers and utilities. As such, epos ensures a consistent operational environment across all stacks, facilitates deployment, and offers a reliable foundation for various applications. Furthermore, dedicated software libraries and tools complement the PS stack, providing support for application development and hardware integration.
- **DCS Interface:** The DCS standard middleware layer, OPC UA, is adopted as the uniform communication interface, ensuring seamless integration of each system with the distributed control system. OPC UA exposes configuration, monitoring, and control interfaces in a standardized manner, enforcing a standard interaction model between local applications and remote supervisory systems. To further accelerate development and promote standardization, the CERN-developed quasar framework (see Sec. 6.2.3) is used to implement OPC UA servers. Within the EMCI-EMP ecosystem, dedicated quasar-based servers are provided for the control, monitoring, and diagnostics of EMCI and EMP hardware.

These layers provide a consistent, scalable platform for development and integration. The following section introduces the fundamental concepts, technologies, and frameworks that underpin the architecture. Subsequently, a more thorough discussion of firmware and software components will be presented.

6.2. Fundamental Concepts and Technologies

Modern System-on-Chip (SoC)-based embedded systems depend on standardized interfaces, kernel subsystems, and software frameworks to ensure efficient communication between hardware, firmware, and higher-level applications. This section introduces the fundamental technologies that underpin the EMCI-EMP ecosystem, as well as the development tools and frameworks employed throughout its implementation. These include the Advanced eXtensible Interface (AXI), the Linux Userspace I/O (UIO) subsystem, and quasar, CERN's framework for building OPC UA servers.

6.2.1. The Advanced eXtensible Interface

The Advanced eXtensible Interface is part of ARM's Advanced Microcontroller Bus Architecture (AMBA) specification and has become the industry standard for high-performance on-chip communication. The following will discuss the AXI protocol standard in detail based on the "AMBA AXI Protocol Specification" (Ref. [74]), which serves as the primary reference.

AXI, introduced with AMBA 3.0 and extended in AMBA 4.0, enables highly parallel, pipelined communication between multiple masters and slaves in complex SoC designs. Its development was driven by the need to overcome the throughput and concurrency limitations of earlier AMBA protocols, such as the Advanced High-performance Bus (AHB) and the APB, by providing an interface that supports independent address, data, and control phases, multiple outstanding transactions, and out-of-order responses.

At its core, the AXI standard establishes a uniform signaling and handshake mechanism while maintaining flexibility in configuring the underlying interconnect topology. This separation enables designers to reuse AXI-compliant IP-cores across different SoC architectures without requiring modification. In a standard system configuration, processors, DMA engines, or accelerators function as AXI masters, while memory controllers, peripherals, and other devices serve as slaves. The communication between them uses dedicated unidirectional channels, coordinated via a deterministic, straightforward handshake.

AXI Protocol Family and Use Cases

According to the AMBA 4.0 specification, three primary AXI protocol variants have been defined. Each of these variants has been optimized for a specific class of communication:

- **AXI4:** A high-performance memory-mapped interface for bulk data transfers between bus masters (e.g., processors, DMA engines) and slaves (memories, high-speed peripherals). AXI4 supports burst transactions and can sustain multiple outstanding transactions concurrently, maximizing throughput in complex SoC designs.
- **AXI4-Lite:** A lightweight, simplified memory-mapped interface used for low-throughput tasks such as reading and writing of control and status registers. It shares the same structure as AXI4 but omits advanced features, resulting in a small logic footprint and ease of use.

- **AXI4-Stream:** A streaming (non-memory-mapped) interface for high-speed unidirectional data flows. Unlike AXI4/AXI4-Lite, the AXI4-Stream protocol has no address or response channels and relies on a simple handshake over a single data channel for flow control. This eliminates address overhead and allows an unlimited burst length for data streams, thereby maximizing throughput for feed-forward data paths.

All three AXI variants utilize the exact fundamental handshake mechanism for data transfers. This mechanism is triggered when the source asserts **VALID**, indicating that it has data or control information available. Concurrently, the destination drives **READY**, signaling its readiness to receive the information. Data or control information is transferred only when both **VALID** and **READY** are asserted within a single clock cycle. At this point, the signal is considered acknowledged. In AXI terminology, each clock cycle in which a **VALID/READY** handshake completes on a data channel is called a *Data Beat*, representing the transfer of one complete data word on the parallel bus.

AXI4 and AXI4-Lite

The architecture of both the AXI4 and AXI4-Lite interfaces implements a five-channel communication structure, as illustrated in Figure 6.2. These independent unidirectional channels facilitate concurrent read and write operations.

- **Write Address (AW):** Carries the write command information from master to slave, including the target address (**AWADDR**) and control signals such as burst length (**AWLEN**) and size (**AWSIZE**). The master asserts **AWVALID** to indicate a valid address, and the slave responds with **AWREADY** when it is ready to accept the address.
- **Write Data (W):** Transfers the actual write data from master to slave (**WDATA**), along with byte-enable strobes (**WSTRB**) and a flag marking the final beat of a burst (**WLAST**). Further, each data beat is acknowledged by its own handshake, where the master asserts **WVALID** when valid data is available on the bus and the slave raises **WREADY** when it is ready to receive the data beat.
- **Write Response (B):** Manages the write response from the slave back to the master after a write transaction completes. This channel carries the status code **BRESP**, indicating, for example, slave-sided transaction errors. The response channel also implements its own handshake signal where the slave asserts **BVALID** when the response is ready, and the master responds with **BREADY** to acknowledge it.

- **Read Address (AR):** Carries read request information from master to slave, including the address to read from (ARADDR) and control signals similar to the AW channel (burst, length, size, etc.). The signals ARVALID/ARREADY are used for the address handshake on read requests.
- **Read Data (R):** Returns requested data from slave to master. This channel conveys the data (RDATA) along with a response code (RRESP), and if the read is a burst, the slave asserts RLAST with the final data beat. Each read data beat is transferred when both handshake signals RVALID and RREADY are high.

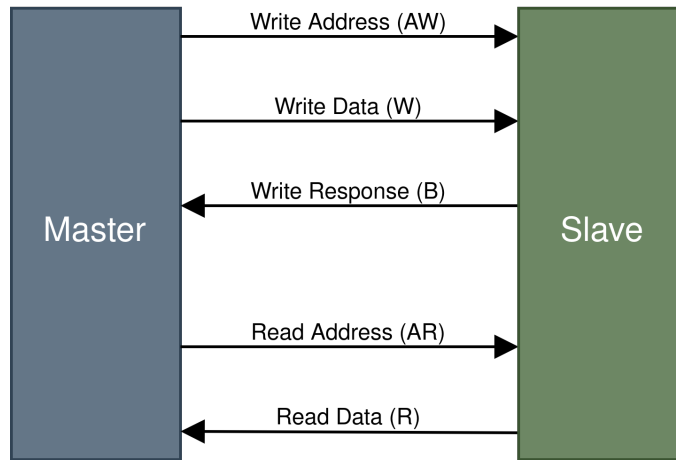


Figure 6.2: The five independent channels of an AXI4 memory-mapped interface (applicable to both AXI4 and AXI4-Lite). Each channel is unidirectional and has its own pair of VALID/READY handshake signals. The separation of address (AW, AR), data (W, R), and response (B) channels enables write and read transactions to progress concurrently and be pipelined, thereby improving throughput compared to single-channel buses. Adapted from Ref. [74].

Every AXI4 and AXI4-Lite read or write transactions consist of an address phase (via the AR or AW channel) followed by one or more data beats on the corresponding data channel (via the R or W channel) during the data phase. Additionally, during a write transaction, the slave responds on the B channel to validate and complete the transaction. As such, each of these unidirectional channels defines its own pair of handshake signals, thereby decoupling the producer and consumer. This decoupling enables each side to throttle the transfer without loss or deadlock. It is essential to note that once a VALID is asserted, the source is responsible for maintaining the data in a stable state until the destination asserts READY, thereby completing the transfer. Similarly, the destination can assert READY at any time, even before VALID is high, or withdraw it to throttle the transactions. A waveform diagram of both an AXI4 write and AXI4 read transaction is shown below. Figure 6.3 illustrates the

signal timing of a typical AXI4 write transaction, whereas Figure 6.4 corresponds to an AXI4 read transaction, with both figures showing all relevant active channels during each phase.

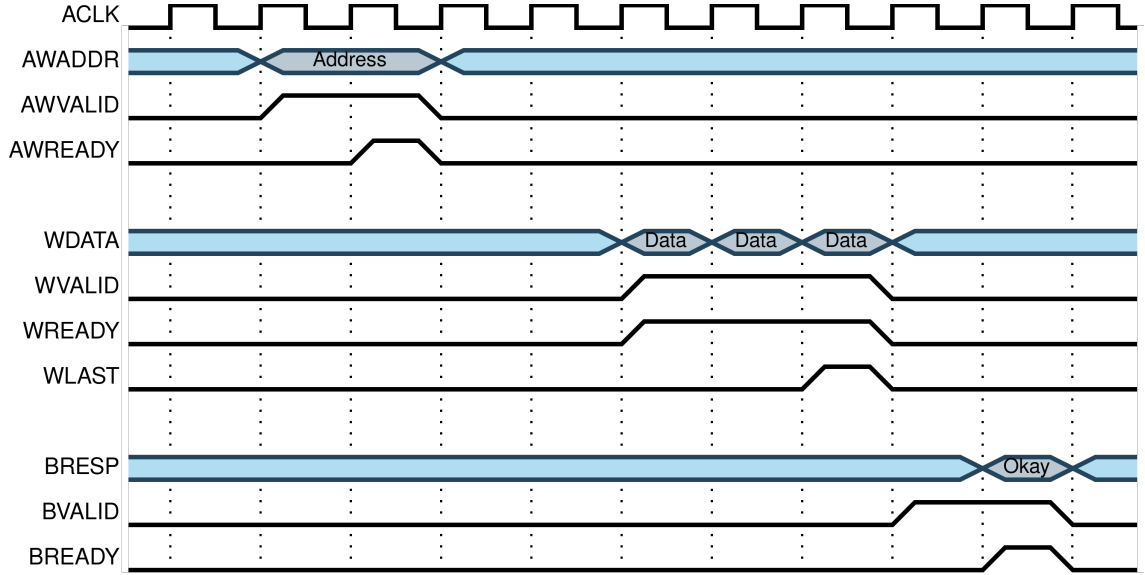


Figure 6.3: AXI4 write transaction timing. During the address phase, the master drives *AWADDR*/*AWVALID* and, optionally, *WDATA*/*WVALID*. The slave acknowledges the address phase by asserting *AWREADY*. During the data phase, the slave asserts *WREADY* when it is ready to receive data, and the master transmits data over *WDATA* while keeping *WVALID* asserted. On the last dataword, the master additionally asserts *WLAST* to mark the last data beat. After the data phase, the master asserts *BVALID*, which the slave acknowledges by asserting *BREADY* and transmitting the status code via *BRESP*, thereby ending the transaction. Adapted from Ref. [74].

As outlined, AXI4 and AXI4-Lite share a common architectural foundation; however, the AXI4 protocol incorporates several additional features, including burst support, transaction identifiers, and optional sideband signals, thereby enhancing configuration and flexibility.

AXI4 is designed for high-bandwidth, memory-mapped communication. The system maintains the five-channel architecture while introducing enhanced channel configurability to optimize data transfer across various applications. A pivotal aspect of AXI4 is its ability to support burst transfers, variable data widths, and a variety of auxiliary signals that facilitate data ordering, protection, and Quality of Service (QoS) as summarized below:

- **Burst Transactions:** AXI4 enables a single read or write request to transfer up to 256 data beats in one burst. These transactions can take different forms,

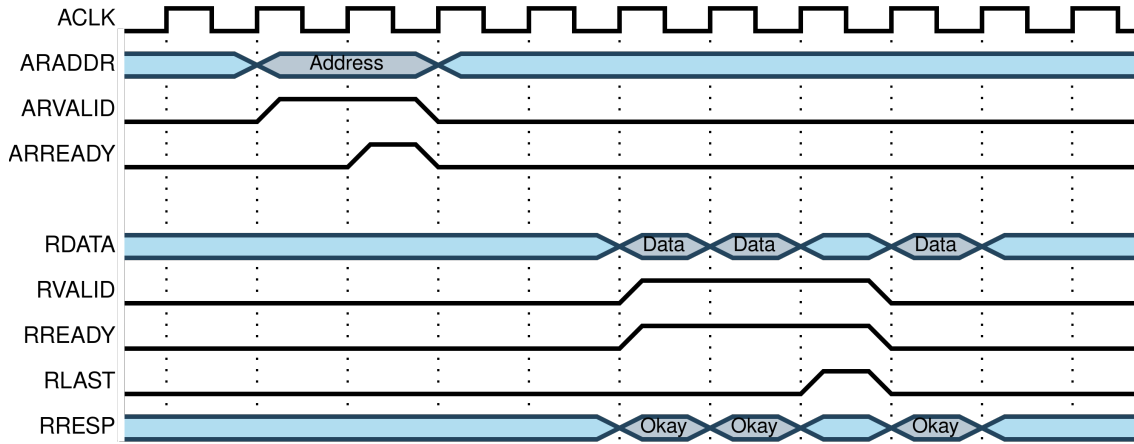


Figure 6.4: AXI4 read transaction timing. During the address phase, the master drives *ARADDR*/*ARVALID*. The slave acknowledges the address phase by asserting *ARREADY*. During the data phase, the master asserts *RREADY* when it is ready to receive data, and the slave transmits data over *RDATA* while keeping *RVALID* asserted. On the last dataword, the slave additionally asserts *RLAST* to mark the last data beat. During the entire transaction, the status of each data beat is transmitted in parallel to the data via the *RESP* signal. Adapted from Ref. [74].

such as incrementing the address, repeating the address, or wrapping around within a predefined address boundary. The master must indicate the burst type, the per-beat size, and the burst length on the address channel. After the address handshake, the master sends a sequence of data beats on the W channel, asserting the *WLAST* signal with the final beat to mark the end of the burst. The slave, after accepting all data, responds on the B channel to complete the transaction. In the read case, the slave returns data beats on the R channel after the completion of the address phase handshake and asserts *RLAST* on the final beat of the burst.

- **Transaction IDs:** AXI4 supports an identifier tag on transactions, allowing responses to be matched to requests and enabling out-of-order completion in interconnects. A master can issue multiple read/write requests with different identifiers; the interconnect and slave can reorder responses as long as transactions with the same identifier are completed in order.
- **Optional Signal Extensions:** The AXI4 specification defines several optional sideband signals to support advanced features: cache hints, protection levels, quality of service, region identification, and user-defined signals. These signals enable the master to convey additional metadata about transactions, such as whether a read is cacheable or bufferable, or to which subordinate region an address maps in a complex system.

Conversely, AXI4-Lite is designed for tasks that do not require the full complexity or bandwidth of AXI, such as reading and writing control and status registers in peripheral devices. To this end, AXI4-Lite defines a more simplified subset of the AXI4 protocol. As such, the protocol preserves the fundamental handshake and the five-channel architecture; yet it supports only single-beat transfers and reduces the set of auxiliary signals. Therefore, AXI4-Lite masters and slaves do not implement the burst length, burst type, or other burst-related signals. Additionally, **WLAST/RLAST** are not required, since the data phase of each transaction is executed within a single clock cycle. Moreover, the system generally utilizes a single identifier or, in some cases, omits these signals entirely, thereby ensuring that all operations are inherently sequential. Therefore, each master must wait for each transaction to complete before issuing the next one.

At the system level, AXI4 and AXI4-Lite are fully interoperable. An AXI4 master is capable of communicating with an AXI4-Lite slave through an interconnect that adapts the unused burst and identifier signals. This compatibility facilitates a unified communication framework across high-speed and control paths within the same SoC.

AXI4-Stream Interface

AXI4-Stream is a specialized, non-memory-mapped variant optimized for continuous data streams. It establishes a connection between a single source and a single destination in a unidirectional manner, thereby eliminating the address and response phases. Instead, it employs a single data channel with **VALID/READY** flow control, optionally accompanied by qualifier signals such as:

- **TLAST**: Indicates the end of a frame or packet.
- **TSTRB, TKEEP**: Byte qualifiers for partial transfers.
- **TID, TDEST, TUSER**: Provides identifiers, routing information, or defined side-band signals.

This lightweight structure minimizes latency and allows for unlimited burst lengths, rendering AXI4-Stream well-suited for real-time or high-bandwidth pipelines, including DMA transfers to FPGA logic, sensor data capture, and image processing chains. The **VALID/READY** handshake ensures reliable flow control by transmitting data only when the consumer can accept it, thereby preventing overflow or underrun without additional signaling.

AXI Integration in SoC Designs

In systems comprising a single master and a single slave, the two can be directly connected to achieve minimal latency and maximum bandwidth. However, modern SoC designs often incorporate multiple masters and slaves that must function concurrently. In such environments, an AXI interconnect, frequently implemented as a crossbar or hierarchical fabric, provides the necessary connectivity between all communicating agents.

While AXI delineates the interface contract between IP-cores, it does not stipulate the physical interconnection of these interfaces within the SoC. Each AXI master and slave can be connected to a central interconnect fabric that manages routing, arbitration, and synchronization. This modular design facilitates IP-core reuse and simplifies SoC integration, provided that all components are AXI-compliant and can communicate seamlessly through the fabric.

To ensure correct and efficient operation, each AXI must interconnect and perform several key functions.

- **Address decoding and routing:** The interconnect is responsible for decoding transaction addresses to ascertain the target slave peripheral or memory. Each slave is assigned a region of the memory/peripheral address map. The interconnect compares the address against these regions and forwards the transaction to the interface of the corresponding slave.
- **Arbitration:** The arbitration logic is responsible for determining which transfer is to be given priority when two or more masters attempt to access the same resource concurrently. Interconnects generally include an arbiter for each slave port, which monitors all incoming requests and grants access to one at a time.
- **Transaction tracking:** An AXI interconnect is required to manage multiple transactions concurrently. Therefore, it is necessary to monitor transaction identifiers and ensure that responses are directed to the appropriate master and linked to the correct transaction. By supporting multiple outstanding transactions, the AXI fabric maximizes the utilization of parallel resources and ensures low latency, as faster slaves are not blocked behind slower ones.
- **Pipelined Data Paths:** To facilitate high operating frequencies over extended interconnect wires, AXI fabrics often implement pipelining through the use of register slices or FIFO buffers within the channel. These pipeline stages disrupt combinational paths, thereby enabling the interconnect to meet the required timing constraints. The AXI protocol is designed to tolerate such

pipeline delays by implementing its handshake mechanism. A pipelined interconnect facilitates concurrency by allowing multiple transfers to occur across different pipeline stages.

In essence, the AXI interconnect functions as a critical element of a SoC, facilitating deterministic and efficient communication among multiple heterogeneous IP-cores. Abstracting address mapping, arbitration, and flow control from individual components enables scalable, reusable system architectures that can be adapted to varying performance and integration requirements.

Summary

The AXI protocol family provides a scalable and modular communication backbone for modern SoC. AXI facilitates efficient data exchange across heterogeneous IP-cores through its unified VALID/READY handshake, flexible channel architecture, and protocol variants tailored for memory-mapped and streaming communication. When combined with configurable interconnect topologies, such as crossbars, AXI ensures both high performance and design reusability, forming the foundation of many contemporary FPGA-based systems.

6.2.2. The Userspace I/O System

For many simple, memory-mapped hardware peripherals, e.g., custom IP-cores implemented in the PL of an SoC, writing a full Linux device driver would introduce a huge amount of complexity. However, if a peripheral can be controlled through a set of registers and, optionally a interrupt line, it is often sufficient to provide user-space access to its mapped memory and a mechanism to receive and handle its interrupts. In particular, if the device's control logic does not inherently benefit from the resources available in the kernel space.

To support exactly this class of peripherals, the Linux Userspace I/O (UIO) subsystem [75] was introduced. UIO is a minimal kernel framework that exposes a device's memory-mapped regions and interrupt events to user space, enabling the driver logic to be implemented in a user-space process. This approach significantly reduces development complexity and mitigates the risk associated with writing and maintaining custom kernel modules. However, UIO is not a universal driver interface. Devices such as USB controllers or network adapters, which already have well-established kernel subsystems, are not considered suitable for UIO. Instead, UIO is intended for devices that can be fully controlled through a memory-mapped interface, typically generate interrupts, and do not fit into any of the established kernel driver frameworks.

In practice, each UIO device registered within the UIO subsystem is accessed via a character device file at `/dev/uioX` and a corresponding set of sysfs attribute files under `/sys/class/uio/uioX/`. The sysfs interface [76] provides device metadata, including map descriptors, physical base addresses, and memory sizes of each region, which can be used to configure access to these regions. The character device, on the other hand, facilitates direct hardware interaction by accessing the device's address space. Applications can use `mmap()` to map the device's memory into their own virtual address space. While hardware interactions can be performed by writing the mapped memory, interrupts are handled by reading and writing directly from the character device.

The UIO subsystem therefore implements a mechanism that allows a device's control logic to be entirely in user space without requiring a custom kernel driver. To further simplify this pattern, higher-level helper libraries are commonly deployed, such as `libuio`, which streamlines access to UIO devices.

libuio

The `libuio` library [77] is a lightweight C helper library for the Linux UIO subsystem. It offers a structured Application Programming Interface (API) that facilitates the discovery and opening of UIO devices, the mapping of their memory regions into user space, the management of interrupts, and the implementation of low-level read/write operations. By encapsulating these recurring routines, `libuio` significantly reduces boilerplate code and improves reliability when developing user-space drivers or hardware access libraries. The library itself is typically distributed as a shared library and can be integrated seamlessly into C or C++ projects by including its header in an extern "C" block.

Its core functionalities can be summarized as follows:

- **UIO Device Discovery:** `Libuio` enables the scanning of UIO sysfs entries to identify devices by name or index. This approach eliminates the use of hard-coded device numbers, thereby enabling applications to dynamically identify the appropriate device, even when device numbering changes due to changes in the device tree or runtime configuration.
- **Memory Mapping:** For each UIO device, `libuio` provides the capability to map one or more memory regions into the user space. It retrieves the physical address and size from sysfs, calculates the correct offset, and uses `mmap` to expose the region to the application. This enables direct access to hardware registers via user-space pointers, eliminating the need for manual offset management.

- **Interrupt Handling:** The library streamlines interrupt handling by making the UIO file descriptor accessible to user code. Applications can block on read calls and wait for an interrupt or a timeout. After handling the event, writing to the descriptor re-arms the interrupt. This encapsulates the fundamental UIO behavior of reading and re-enabling Interrupt Requests (IRQs), thereby enabling user-space drivers to manage interrupts in a portable manner.
- **Multiple Maps Support:** UIO devices may expose multiple memory regions (for instance, separate configuration and data spaces). Libuio offers an API that facilitates the mapping of individual regions by index, thereby enabling straightforward access to all functional areas of a complex device.

6.2.3. The Quick OPC UA Server Generation Framework

The quasar framework [49, 78] is a CERN-developed, generic, extensible toolkit for generating OPC UA servers that simplifies and standardizes the integration of heterogeneous hardware into large-scale distributed control systems. Figure 6.5 presents an overview of the framework’s architecture, emphasizing and putting in context the different constituting layers.

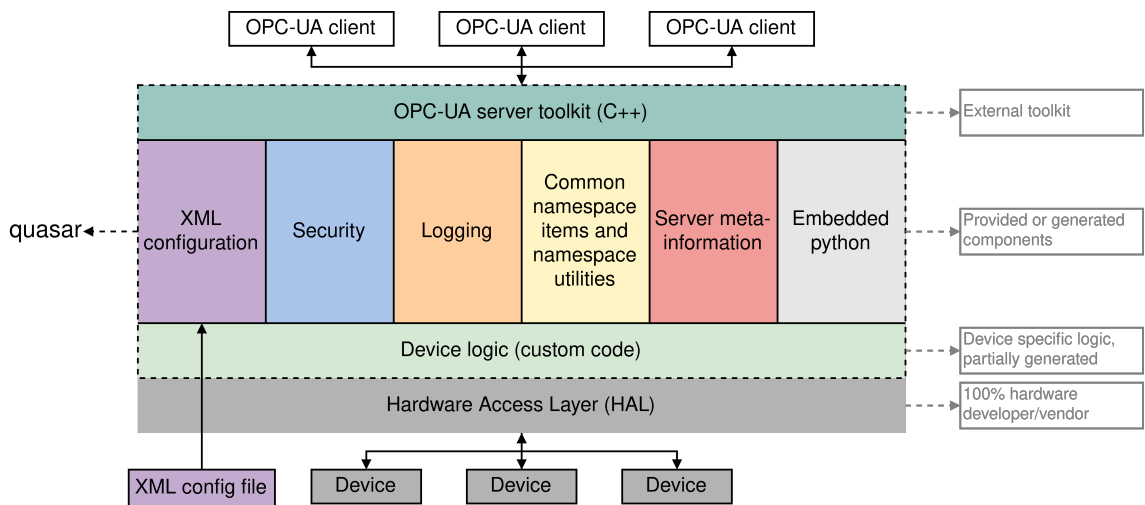


Figure 6.5: Overview of the quasar framework, illustrating the layered architecture including the HAL, Device logic, generated server components, and the external OPC UA server toolkit. Adapted from Ref. [79].

The lowest layer of the architecture is the Hardware Access Layer (HAL), which provides a uniform API for managing hardware access and must be provided by the hardware developer or vendor. The device logic layer, situated above, implements

device-specific business logic and serves as an interface between the HAL and the higher-level layers of the quasar framework. These higher-level layers are composed of multiple modules, each addressing a specific functional aspect, with the uppermost module implementing the address space, which exposes the server's information model to clients and is implemented using a UA SDK.

The structure of the address space is defined via the quasar design file, which provides an Extensible Markup Language (XML)-based object-oriented description of the device or system, its specific relations, and its attributes. In addition, a separate XML-based configuration file is required that defines the concrete instances and parameters of these objects. A dedicated schema-to-C++ mapping generator then uses these files to instantiate and initialize the objects and link the corresponding address space. Moreover, additional modules that operate purely in the address space provide services such as calculated variables, component-based logging, or server metadata management.

The object-oriented device model is a key aspect of the quasar framework. Based on the description, quasar automatically generates the OPC UA address space and the corresponding device-logic code stubs. This eliminates the need for developers to understand low-level OPC UA internals and significantly reduces the overall implementation effort. In summary, the framework promotes a clear separation between device-specific implementation and standardized server interfaces, thereby improving maintainability, accelerating development, and ensuring consistent integration within the DCS.

6.3. The EMP FPGA Firmware Reference Project

The EMP FPGA firmware reference project [80, 81] provides a modular and extensible foundation for implementing FE control and monitoring logic on the EMP platform. The design, as illustrated in Figure 6.6, establishes a standardized baseline architecture that combines reusable infrastructure and IP-cores with well-defined integration points for custom application logic. Its primary objectives include enabling communication between EMP, the lpGBT, and the FE hardware, as well as ensuring compatibility with the software and control layers built upon it.

To achieve these goals, the firmware integrates custom IP-cores with standard Xilinx building blocks to implement communication paths and data pipelines. Particular emphasis is placed on modularity, allowing subsystem developers to extend or modify designated logic regions without interfering with the core infrastructure.

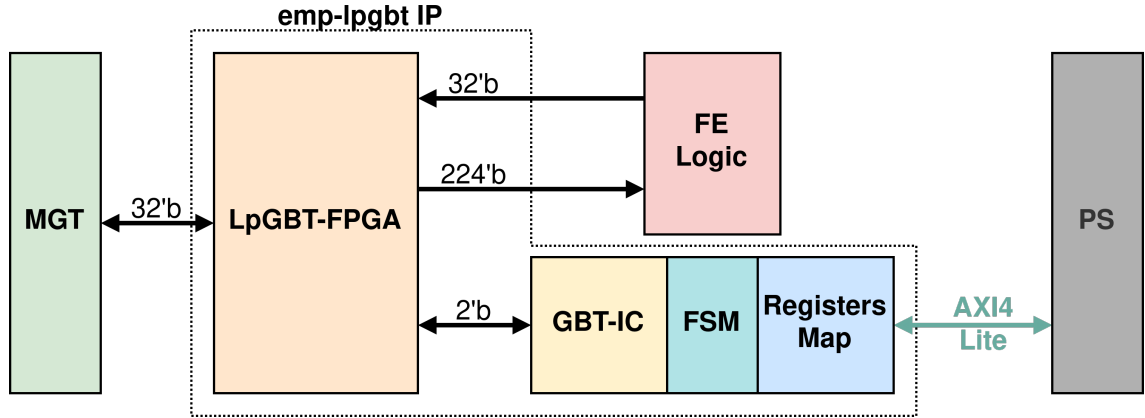


Figure 6.6: Overview of the lpGBT frame codec fabric: The lpGBT IC channel is exposed via an AXI4-Lite interface to the PS. A dedicated FSM manages register transactions with the GBT-IC IP-core [82], which handles lpGBT fields and acts as a SerDes between the LPGBT-FPGA IP-core [83] and FSM. The LPGBT-FPGA implements frame coding and connects directly to the MGT, while FE logic interfaces directly through the emp-lpgbt IP-core.

One of the central use cases addressed by the reference project is the configuration, control, and monitoring of the lpGBT. The operation of the ASIC requires configuring its internal registers, particularly to lock its CDR/PLL circuits to the downlink data stream, thereby recovering a stable system clock. In operational deployments, these registers are written over the IC field of the lpGBT frame, requiring the EMP to implement the full uplink/downlink pipeline, including frame encoding/decoding and IC field handling. To support this functionality, the firmware project includes an end-to-end IC channel communication pipeline. This covers the AXI4-Lite interface on the PS side, through the emp-lpgbt IP-core, to the optical transceivers via the MGT IP-core. In parallel, the eLink data path incorporated adoption points for FE-specific logic, enabling integration of custom data into the lpGBT frame structure. This ensures reusability, system-level consistency, and clear boundaries between infrastructure and application logic, thereby promoting a well-defined separation between the two.

The subsequent subsections provide a detailed overview of the individual IP-cores that comprise the EMP FPGA reference design. Each core is discussed in terms of its functionality, interface specification, and role within the overall architecture, highlighting how the control path for lpGBT monitoring and control is implemented and how the general lpGBT frame encoding/decoding is performed.

6.3.1. The emp-lpgbt IP-core

The emp-lpGBT IP core is a custom firmware design that comprises multiple sub-cores. It implements a complete bidirectional pipeline that enables the configuration and monitoring of the lpGBT via the IC channel and provides interfaces for FE-specific eLink data integration. An overview of the core's architecture is presented in Figure 6.7.

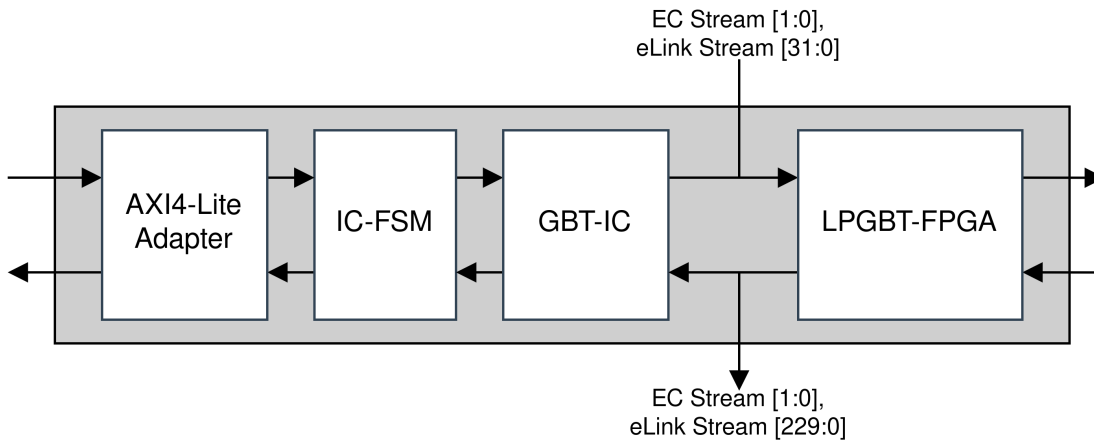


Figure 6.7: High-level overview of the architecture of the emp-lpgbt IP-core.

The core integrates a custom AXI4-Lite adapter IP-core, providing software with direct access to a memory-mapped register space containing control, address, and status registers. This interface enables the software to issue read and write requests targeting specific lpGBT configuration registers. The execution of these transactions is coordinated by a dedicated IC-FSM, which translates the software commands into the corresponding signal sequences and synchronizes them via a dual-clock FIFO for interfacing with the GBT-IC IP-core. The GBT-IC module then performs serialization and IC channel coding in accordance with the lpGBT protocol. The resulting bitstream is multiplexed with the EC and user eLink data streams to form the composite input, which is transferred through a clock Clock Domain Crossing (CDC) module from the 40 MHz logic domain to the 320 MHz MGT domain. The data stream is then processed by the LPGBT-FPGA IP-core, where the final frame encoding is carried out. The lpGBT frame is subsequently interfaced with the MGT IP-core.

Conversely, the incoming serial data stream from the lpGBT is received from the MGT IP-core and first decoded by the LPGBT-FPGA IP-core before being transferred back into the logic domain through the reverse CDC module. The frame is then demultiplexed into its IC, EC, and eLink data components. The IC stream

is passed through the GBT-IC module, where the payload is extracted, deserialized, and decoded. After synchronization via the dual-clock FIFO, the IC-FSM writes the resulting data to the memory-mapped registers, making it accessible to the software.

Although the AXI4-Lite adapter, IC-FSM, and GBT-IC module are dedicated to handling the IC field, the EC and eLink data fields are propagated in parallel through the LPGBT-FPGA's frame encoding and decoding. The according signals are made available as interfaces, enabling straightforward integration of subsystem-specific logic while preserving compatibility with the standardized lpGBT frame pipeline. In the following, the individual components of the core, the GBT-IC IP-core, and the LPGBT-FPGA IP-core will be discussed in detail, with particular emphasis on their specific functions within the lpGBT frame pipeline.

The GBT-IC IP-core

The GBT-IC IP-core was initially developed in conjunction with the SCA-EC IP-core as components of the generic GBT-SC module for FPGAs [82, 84]. The purpose of this development was to generate the serial stream for both IC and EC types of GigaBit Transceiver (GBTx) slow-control communication [85]. The SCA-EC IP-core is designed to implement the High-Level Data Link Control (HDLC) protocol, which serializes and deserializes EC control data for up to 41 Giga-Bit Transceiver - Slow Control Adapter (GBT-SCA) ASICs. The GBT-IC, on the other hand, implements an HDLC subset, enabling the control and monitoring of the IC stream of a single GBTx link.

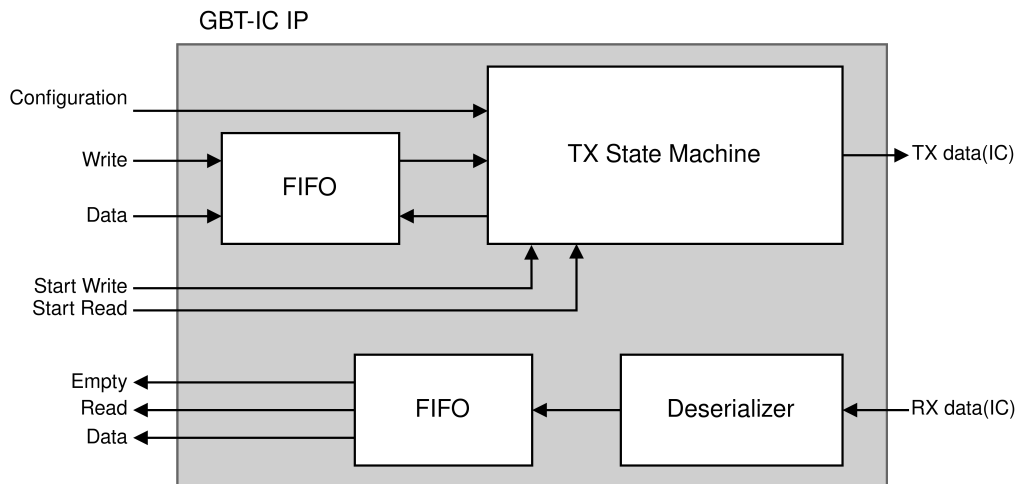


Figure 6.8: Architecture of the GBT-IC IP-core. Adapted from Ref. [84].

Despite its original development for the GBTx system, the IC protocols of the lpGBT and the GBTx are fully compatible. It is therefore recommended that when the IC channel of an lpGBT is to be used, the GBT-IC IP-core is instantiated alongside the LPGBT-FPGA IP-core.

The LPGBT-FPGA IP-core

The LPGBT-FPGA IP-core [83, 86] provides the implementation that serves as the direct counterpart to the lpGBT ASIC. Its design supports all frame coding schemes defined for both the downlink and uplink path, and it is therefore configurable with respect to the lpGBT's uplink bandwidth and FEC coding (see Sec. 5.2.3). To accommodate these options, the LPGBT-FPGA IP-core provides a modular structure in which the relevant building blocks must be instantiated according to the intended uplink configuration. The architecture of the core, as depicted in Fig. 6.9, can thus be characterized as asymmetric, given its integration of different bandwidths and FEC codings for uplink and downlink. The EMP FPGA reference design implements the core to operate the uplink at 10.24 Gbit/s with the FEC5 coding, reflecting the standard setup for the EMCI-EMP system.

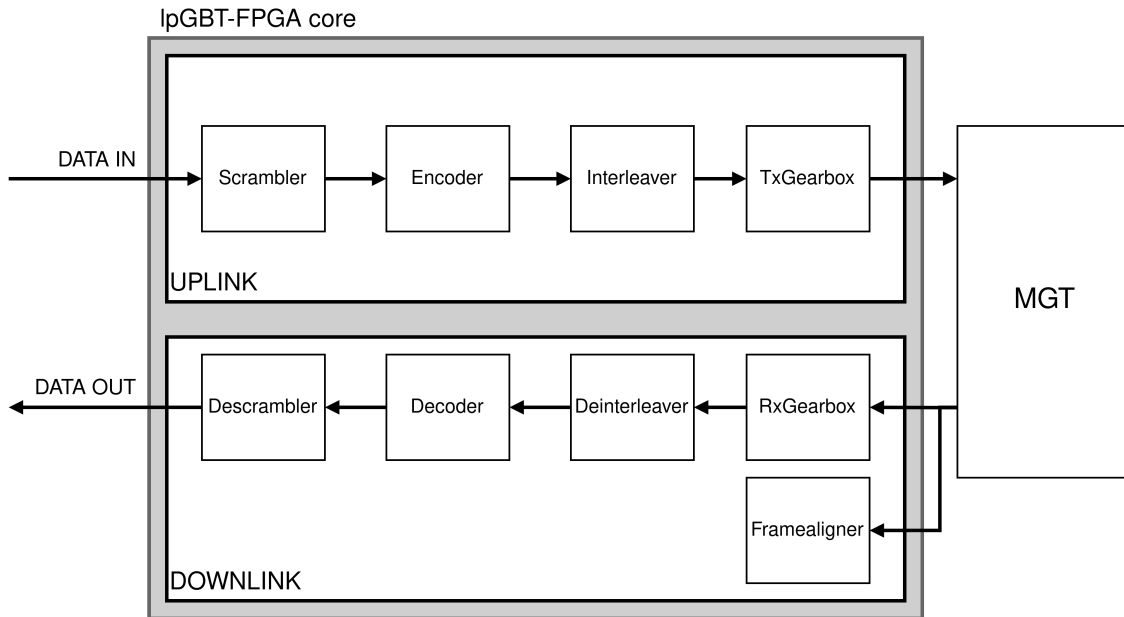


Figure 6.9: Overview of the asymmetric architecture of the LPGBT-FPGA IP-core. Adapted from Ref. [87].

The downlink datapath, which encodes data according to the lpGBT specification, is composed of four blocks:

- Scrambler: Randomizes the outgoing 36 bit data word, using a scrambling function of order 36 based on a recursive equation.
- Encoder: Computes the FEC code for the outgoing frame using a Reed-Solomon scheme as defined by the lpGBT protocol.
- Interleaver: Reorders the encoded frame bits to spread potential burst errors across multiple FEC blocks. This improves the overall error correction capability of the downlink frame.
- TX-Gearbox: Implements a register-based CDC to transfer data from the internal serial clock domain to the MGT clock domain. It also handles oversampling by duplicating bits when the word rate is lower than the transceiver clock rate.

The uplink datapath, on the other hand, decodes data according to the lpGBT specification and is based on five different blocks:

- Frame Aligner: Detects the position of the frame header and requests a phase shift on the input clock to align the received data.
- RX-Gearbox: Handles CDC between the MGT and the internal logic. It collects oversampled bits from the MGT and reorders them to reconstruct the original data word.
- Deinterleaver: Reverses the interleaving applied by the lpGBT to restore the original order of the FEC codewords.
- Decoder: Applies Reed-Solomon decoding (FEC5 or FEC12, depending on the configuration) to detect and correct transmission errors.
- Descrambler: Recovers the original user data by inverting the scrambling function used by the uplink.

6.3.2. The FW-Debughub IP-core

The FW-Debughub IP-core is a custom firmware design for the EMP FPGA reference project. It serves as a dedicated debug and control bridge between the PL and the PS. It is intended primarily to enable software access to key firmware status and control features, enabling monitoring and diagnostics of EMP firmware.

Communication with the PS is implemented via a custom AXI4-Lite slave adapter, similar to the adapter implemented for the GBT-IC IP-core. This AXI4-Lite slave adapter features a dedicated register map tailored to debug and control functionality. The core itself supports monitoring of the lpGBT link by acquiring the uplink frame

alignment, the MGT status, and the LPGBT-FPGA fabric status. Additionally, it provides control over several reset signals of system cores within the hardware design, allowing for targeted or complete resets to be performed directly from the PS.

6.3.3. The Generic EMP-MGT IP-core

The EMP-MGT IP-core is a scalable, configurable, multi-transceiver core designed for the EMP FPGA hardware design. It can instantiate up to thirteen UltraScale+ GTH transceivers preconfigured using the Xilinx Transceiver Wizard and aligned with lpGBT interface requirements. As shown in Figure 6.10, the core facilitates parameterized instantiation through a Graphical User Interface (GUI), thereby ensuring the design's dynamic adaptability to system requirements while optimizing resource utilization.

The core can be instantiated employing two distinct modes. The first mode is designated for the optical FireFly modules, in which the core instantiates only the MGT channels routed to FireFly modules. Conversely, the second mode is defined for the SFP+ module, in which the core instantiates only the single MGT channel routed to the SFP+ module. This separation reflects the differences between the connected optics modules. However, the separation is primarily due to the clock constraints inherent in the UltraScale+ GTH transceiver architecture, in which a reference clock attached to a specific quad can be shared with at most two other quads. Given that the FireFly modules are connected to a total of 12 MGTs, which occupy three complete quads, a single reference clock is utilized and shared among the two remaining quads. However, the SFP+ module connected via channel 15 of quad four and therefore requires its own dedicated clock.

Furthermore, the FireFly MGTs allows individual channels to be disabled, thereby excluding them from instantiation, and to select each quad's master reset. The SFP+ mode is single-channel and therefore does not provide the same per-channel configurability and uses a fixed master reset. Moreover, the GUI allows each channel's Transmit (TX)/Receive (RX) polarity to be configured separately.

From a system integration perspective, the EMP-MGT IP-core is intended to interface directly with the emp-lpgbt IP-core's framing logic, thereby serving as the physical transceiver interface between the frame pipeline and the optical modules on the baseboard.

Component Name: emp_generic_mgt_0

Module Selector

Module Selector: FireFly (dropdown menu with FireFly and SFP+ options)

QUAD 0

Channels Q0: ☒ Channel 0, ☒ Channel 1, ☒ Channel 2, ☒ Channel 3

Master RESET Q0: Reset Quad 0: 0 (dropdown)

QUAD 1

Channels Q1: ☒ Channel 4, ☒ Channel 5, ☒ Channel 6, ☒ Channel 7

Master RESET Q1: Reset Quad 1: 4 (dropdown)

QUAD 2

Channels Q2: ☒ Channel 8, ☒ Channel 9, ☒ Channel 10, ☒ Channel 11

Master RESET Q2: Reset Quad 2: 8 (dropdown)

QUAD 3

Channels Q3: ☒ Channel 12

FireFly Settings

FireFly RX Polarity: 0xE07 (dropdown)

FireFly TX Polarity: 0xFF (dropdown)

SFP+ Settings

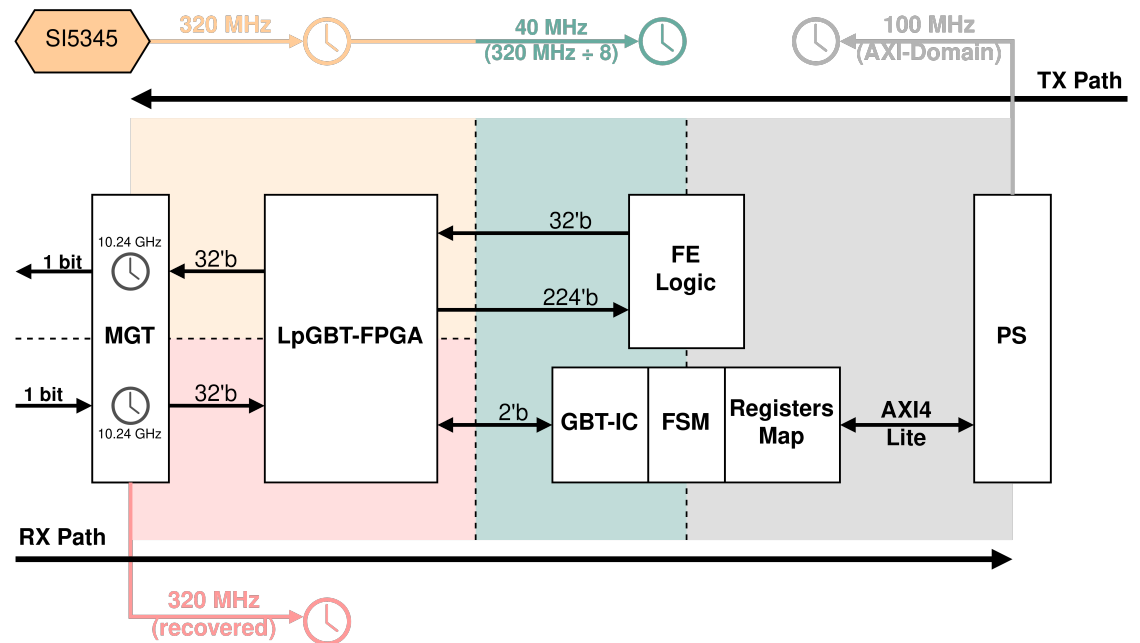
SFP+ RX Polarity: "1" (dropdown)

SFP+ TX Polarity: "0" (dropdown)

Figure 6.10: Overview of the customization GUI of the EMP-MGT IP-core. The GUI enables the parametrization.

6.3.4. Clock-Domain Architecture

The EMP FPGA reference firmware operates across several distinct clock domains, as required by the integrated IP-cores and communication protocols. The high-speed serial communication through the MGTs and the encoding/decoding fabric of the LPGBT-FPGA IP-cores rely on 320 MHz domains that are either reference-synchronous (downlink) or link-synchronous (uplink). In contrast, the IC channel handling within the GBT-IC IP-core and the IC-FSM are operating in a 40 MHz clock domain,



tightly coupled to the data being handled. Therefore, control paths and the IC-FSM implement dedicated dual-clock FIFOs or synchronizer stages to hand off and accept data and control signals between the 100 MHz domain and the 40 MHz domain.

6.4. epos - The EMP Operating System

The epos framework [72, 88] establishes a coherent and extensible software environment for the EMP, unifying the boot process and the operating system while providing standard utilities for operation. The framework is composed of three main components:

- **epos-bsp:** A customized Xilinx PetaLinux Board Support Package (BSP) [89] designed explicitly for the EMP hardware platform. The BSP integrates the complete board reference design, including the hardware description, device tree, and configuration files required to build the EMP boot image and associated Linux kernel modules. Based on this BSP, a PetaLinux project can be created to enable the tailored, pre-configured building and packaging of essential system software components, such as the First Stage Bootloader (FSBL), Arm Trusted Firmware, U-Boot, PMU Firmware, and the Linux kernel. This approach streamlines the generation of boot images and kernel modules, ensuring reproducibility and consistent hardware support across all system deployments.
- **epos-rootfs:** A build environment for the EMP's root filesystem (rootfs) based on CERN's standard AlmaLinux 9 distribution. The environment supports the integration of additional packages and libraries at build time, including dependencies for typical EMP applications such as quasar-based OPC UA servers. The resulting rootfs is production-ready and supports multiple deployment options, via SD card, on-board eMMC, or Network File System (NFS).
- **etools:** A suite of utilities for managing FPGA firmware and device-tree overlays. From a Vivado hardware design, the tools automatically generate the FPGA bitstream and the corresponding device-tree overlay, which can be dynamically applied from the PS side. This facilitates runtime reconfiguration of PL-based modules while ensuring uninterrupted operation of the base operating system.

Collectively, these components offer a comprehensive and standardized software stack for the EMP. The framework ensures reproducibility and consistency in system

deployment and ensures long-term maintainability by aligning with CERN’s established software standards. Additionally, both emp-rootfs and emp-bsp are automatically built and deployed via GitLab Continuous Integration (CI)/Continuous Delivery (CD) jobs. This provides users with a streamlined and user-friendly system configuration process.

6.5. Software

The EMP’s function within the DCS is to enable access to the connected hardware, i.e., the EMCI and FE interfaces, as well as the EMP peripherals. This access must be exposed through the standard DCS middleware layer, OPC UA, thereby ensuring seamless integration into the SCADA system. By adopting OPC UA, an additional abstraction layer is introduced, allowing hardware-related parameters such as temperatures and voltages to be represented directly within the OPC UA address space. This approach avoids exposing low-level register contents or raw bitstreams, instead ensuring that the information provided is human-readable and system-agnostic.

The EMP is a software-centric platform in which the PS governs both the PL and the connected peripherals. As such, hardware access is also managed by the PS. While the PS can directly control EMP peripherals, it is not suited for tasks such as line coding or operating high-speed serial transceivers (MGTs), which are required for communication with the lpGBT and, henceforth, the connected FE-interfaces. These functions are delegated to the PL, which is optimized for such functionality. The interface between PS and PL is established through the AXI protocol, which exposes memory-mapped resources of the PL to software running on the PS.

To support this functionality within the DCS, the EMP employs a three-layered software stack, comprising the driver layer, the HAL, and the middleware layer. Hardware access to on-board EMP peripherals is provided at the driver layer through standard Linux kernel drivers. However, access to devices implemented in the PL must either rely on custom drivers or leverage the Linux UIO subsystem (see Sec. 6.2.2). The development and maintenance of such custom drivers can be costly; thus, the UIO subsystem provides a generic mechanism for mapping hardware register spaces to user space. It thereby offers a standardized software interface to PL devices via the AXI4-Lite protocol. The libuio library further simplifies the hardware access via the UIO interface by providing user-space abstractions for interacting. On top of this driver layer, the HAL is implemented through device-specific libraries, such as LpGbtSw and empTools, which encapsulate the higher-level logic of the lpGBT and EMP peripherals. These HALs rely on kernel driver APIs, such as libuio,

to perform the actual register access, but expose this functionality through structured, device-aware software interfaces. The implementation of the middleware layer is facilitated by the quasar framework (see Sec. 6.2.3), which provides a standardized way to build OPC UA servers and leverages the HAL implementation to expose hierarchical device access to the DCS SCADA system.

The following subsections offer a comprehensive overview of the software stack's constituent components. These include the high-level HAL libraries LpGbtSw and empTools, as well as additional utilities, such as AirHDL and PuzzledLizardWizard, which facilitate the implementation and integration of the stack. The chapter concludes with the presentation of the two quasar-based OPC UA servers that provide control and monitoring capabilities for the EMCI and EMP hardware.

6.5.1. The LpGbtSw Library

The LpGbtSw library [72, 90] is a versatile C++-based software framework that provides comprehensive support for the lpGBT ASIC. It serves as an abstraction layer that encapsulates the lpGBT's business logic. This logic encompasses detailed knowledge of how to access peripheral interfaces, such as ADC channels, GPIOs, and I2C masters. Additionally, the library provides a convenient mechanism for configuring and reading the lpGBT registers.

The library provides a polymorphic API for AXI4-Lite register-based interactions with the lpGBT, enabling developers to create portable, backend-agnostic software. In the context of EMP-based systems, the LpGbtSw offers a specialized backend implementation that facilitates communication with the GBT-IC IP-core through the AXI4-Lite interface, leveraging the libuio library. Furthermore, a pure software lpGBT simulator is available, which emulates the chip's behavior based on its register map in a request-reply fashion. This simulator is utilized for the development and testing of systems in scenarios where access to an EMP or an lpGBT-based FE is restricted or non-existent.

A collection of standalone tools, referred to as demonstrators, complements the library and enables console-based interaction with the lpGBT, as shown in Figure 6.12. These tools allow users to execute a series of functions, including reading ADC channels, manipulating GPIOs, accessing individual registers, and configuring the lpGBT via its IC channel. Notably, these operations can be performed without the necessity of custom application code, thereby facilitating the integration of these tools into existing systems.

The LpGbtSw is primarily designed to function as a HAL for an OPC UA server, thereby facilitating efficient and flexible communication with the lpGBT across a range of hardware configurations and operating conditions. The library's modular

```
$ sudo ./lpGbtAdc -a lpGBT-uio://emp_lpGBT_9 -f 1 -i 1

[RegisterClerkFactory.cpp:27, INF] getClerk for lpGBT-uio://emp_lpGBT_9
[LpGbtUioBackend.cpp:21, INF] Initializing uio device 'emp_lpGBT_9'
[LpGbtUioBackend.cpp:24, INF] The magic number for this uio device is: 656d7049
[LpGbtAdc.cpp:94, INF] For channel 0 got ADC value: 851
[LpGbtAdc.cpp:94, INF] For channel 1 got ADC value: 1023
[LpGbtAdc.cpp:94, INF] For channel 2 got ADC value: 1023
[LpGbtAdc.cpp:94, INF] For channel 3 got ADC value: 1023
[LpGbtAdc.cpp:94, INF] For channel 4 got ADC value: 1023
[LpGbtAdc.cpp:94, INF] For channel 5 got ADC value: 1023
[LpGbtAdc.cpp:94, INF] For channel 6 got ADC value: 1023
[LpGbtAdc.cpp:94, INF] For channel 7 got ADC value: 1023
```

Figure 6.12: Output of the *lpGbtAdc* demonstrator from the *LpGbtSw* library, showing the raw ADC values read from the lpGBT over the IC channel using the UIO backend.

architecture enables the integration of new backends, ensuring seamless extensibility to other lpGBT-based systems.

6.5.2. empTools

The empTools [91] is a versatile software library that provides board-level peripheral support and access to the FW-Debughub in the FPGA reference design. It implements the per-device logic required to control and monitor EMP baseboard peripherals, including the FireFly/SFP+ modules, the Si5345 clock generator, and the temperature sensors.

Peripheral I2C devices are connected to one of the Zynq's PS I2C masters via a bidirectional switch. Each peripheral is represented in the Linux OS through a dedicated `sysfs/dev` node, instantiated from the corresponding device-tree entries during system boot. The switch itself is managed by a standard platform driver, enabling transparent access to each multiplexed PS interface without requiring explicit switch handling in user space. Device access itself is structured through a two-layer HAL stack. The first layer provides a generic, low-level PS abstraction that encapsulates file descriptor management, `ioctl` calls, and buffer handling behind a uniform register-oriented API. This ensures that all device-specific implementations can rely on a consistent and reusable communication mechanism. The second layer builds upon this base to implement device-specific logic, exposing typed register accessor, bit-field abstractions, and convenience functions tailored to the semantics of each peripheral.

The functionality of these low-level access layers is provided by a set of demonstrator tools that enable direct interaction with EMP peripherals from the command line (see Figure 6.13). These tools serve as both configuration and debugging aids for development when deploying the related EMP OPC UA monitoring server is not

```

$ sudo ./fireflyI2C -m 1 -i

[main_firefly.cpp:113, INF] Common measurement, diagnostic & control parameters:
[main_firefly.cpp:120, INF] Status:                01000000
[main_firefly.cpp:136, INF] Status Summaries:       00100000
[main_firefly.cpp:145, INF] Temperature Alarms:       00000000
[main_firefly.cpp:154, INF] Vcc3.3 Alarms:            00000000
[main_firefly.cpp:156, INF] Internal Temp Monitor:    39°C
[main_firefly.cpp:157, INF] Vcc Monitor:             3.226V
[main_firefly.cpp:158, INF] Channel Disable:         00000000 0000    [0 = enabled, 1 = disabled]

```

Figure 6.13: Command-line output of the *empTools* demonstrator for the FireFly modules, showing typical measurements, diagnostic values, and control parameters of a single FireFly module, read from its internal registers via the TWI interface.

convenient. Furthermore, the library implements a set of system services that leverage the peripheral HAL to automate critical monitoring and configuration tasks at runtime. This encompasses the automated programming of the Si5345 clock generator during the system’s initialization and the continuous temperature monitoring of the FireFly modules. Notably, the FireFly service incorporates an interlock mechanism that deactivates the internal laser drivers and output channels if the module’s critical temperature threshold is exceeded for an extended period. This reduces power consumption, thereby maintaining the temperature within the operational range and preventing thermal damage.

In addition to the PS-based peripherals, the library also implements access to the AXI4-Lite interface of the FW-DebugHub via the Linux UIO subsystem. This interface facilitates control of firmware resets and monitoring of lpGBT link status, including frame alignment, lpGBT, and MGT fabric status, as well as FEC error counters. The related software design employs a two-layered approach similar to the I2C peripheral’s implementation. The first layer implements memory access to specified registers by integrating *libuio*, while the second layer implements the semantic functions for controlling resets or retrieving status information. Dedicated demonstrator tools are also provided, enabling users to query critical firmware system parameters directly from the command line.

Additionally, the *empTools* library provides support for the Xilinx SYSMON, which is integrated within every Zynq UltraScale+ MPSoC. Each device implements one SYSMON block in both the PS and the PL, thereby enabling monitoring of on-chip and off-chip voltages and junction temperatures. The SYSMON is exposed in Linux via the IIO subsystem, which provides two nodes for each value: the raw ADC count and the scaling factor. To facilitate seamless integration, the library offers a lightweight wrapper class that implements functions to retrieve and convert the raw value into physical units, such as volts and degrees Celsius. The wrapper is designed to handle both scaling and unit conversion internally, thereby offering a convenient interface for monitoring PS and PL supplies, reference voltages, and


```
$ sudo ./sysMonitor

[xsysmon.cpp:10, INF] Temperatures
[xsysmon.cpp:11, INF] LPD temperature: 44.841 °C
[xsysmon.cpp:12, INF] FPD temperature: 46.395 °C
[xsysmon.cpp:13, INF] PL temperature : 46.675 °C

[xsysmon.cpp:15, INF] AMS CTRL
[xsysmon.cpp:16, INF] System PLLs voltage      : 1.21 V
[xsysmon.cpp:17, INF] Battery voltage          : 1.767 V
[xsysmon.cpp:18, INF] PL Internal voltage       : 0.857 V
[xsysmon.cpp:19, INF] Block RAM voltage         : 0.858 V
[xsysmon.cpp:20, INF] PL Aux voltage            : 1.803 V
[xsysmon.cpp:21, INF] Voltage for six DDR I/O PLLs : 1.811 V
[xsysmon.cpp:22, INF] VCC_PSINTFP_DDR voltage   : 0.836 V
```

Figure 6.14: Output of the SYSMON demonstrator tool, reporting on-chip temperatures and supply voltages obtained from the Industrial I/O (IIO) subsystem.

thermal sensors. An example console readout produced by this wrapper is shown in Figure 6.14. The integration of system monitoring information complements the empTools, delivering comprehensive monitoring and control of the entire EMP hardware.

Similar to the LpGbtSw, the primary function of the empTools is to serve as the HAL layer for an OPC UA server. Consequently, it facilitates remote control and monitoring of EMP hardware in a structured, DCS-conforming manner, while allowing for seamless integration into the SCADA system.

6.5.3. Additional Software Utilities

The implementation of a UIO interface for a custom PL device requires exact specification of the interface, particularly the memory space. On the PL side, the AXI4-Lite interface and its associated register map must be defined, including the structure, organization, and mapping into the system address space. On the software side, the low-level access code must correctly reflect the memory-mapped implementation to ensure reliable access to signals. During development, when modifications and updates occur frequently, the process can quickly become time-consuming and error-prone, especially in complex systems.

To address these challenges, dedicated utilities are available that streamline the generation of memory space implementations and their corresponding software interfaces, thereby accelerating the overall development workflow. In particular, the open-source AirHdl tool, in conjunction with the CERN-developed PuzzledLizard-Wizard, establishes a complementary toolchain that bridges the gap between hardware register definition and low-level access code. Together, these tools facilitate high-level, type-safe software development.

AirHDL

AirHdl [92] is a web-based register generator for AXI4-based designs in VHDL and SystemVerilog. It provides an intuitive graphical interface that enables developers to formally specify registers, bit-fields, and their semantics in a consistent and easily maintainable manner. Once defined, the register map can be automatically translated into synthesizable HDL, either as a VHDL component or a SystemVerilog module, with an AXI4-Lite interface on the host-facing side. In addition to its HDL components, AirHdl facilitates software development and project documentation by generating a variety of complementary files, including:

- C/C++ header files
- SystemVerilog testbenches
- VHDL testbenches
- HTML documentation
- Markdown documentation
- IP-XACT XML descriptions
- JSON representations

By automating these development steps, AirHdl significantly reduces manual effort, ensures consistency between hardware and software implementations, and provides a reliable, standardized interface for memory-mapped peripherals.

PuzzledLizardWizard

PuzzledLizardWizard [72, 93] is a code generation tool that builds on JSON register mapping definitions exported from AirHdl. Based on the provided JSON export, the tool automates the generation of low-level C and C++ HALs with integrated libuio support for AXI4 Lite peripherals. The generated HALs offer type-safe register access and bitfield abstractions while extending the common command-request/response pattern with interrupt-safe handlers.

As part of the EMP workflow, the PuzzledLizardWizard enables rapid generation of maintainable HAL skeletons for user-defined IP-cores. By eliminating repetitive boilerplate code, the potential for integration errors is reduced, and developers can focus on developing the higher-level HAL.

6.5.4. OPC UA Servers

The lpGBT OPC UA server and the EMP Monitoring & Control OPC UA server have been developed using CERN’s quasar framework. These servers are designed to abstract hardware access provided by their respective HALs for the EMCI and EMP hardware, mapping functionality to an OPC UA address space. This approach facilitates access to lpGBT and EMP monitoring and control through standardized network services that are easily integrated into the distributed SCADA backend. The servers complement their respective application domains, thereby forming two fundamental pillars of the EMCI-EMP ecosystem. As such, they enable the coherent and safe operation of the novel DCS interface.

lpGBT OPC UA Server

The lpGBT OPC UA server [72, 94] exposes the control and status interfaces of the lpGBT ASIC via the OPC UA protocol. It functions as an interface between the lpGBT on the EMCI and the SCADA-based DCS BE. This enables remote clients to monitor and control the lpGBT across the network, with a focus on streamlined access and seamless integration into the experiment’s control infrastructure [94]. The quasar-generated OPC UA information model of the lpGBT OPC UA server is shown in Figure 6.15.

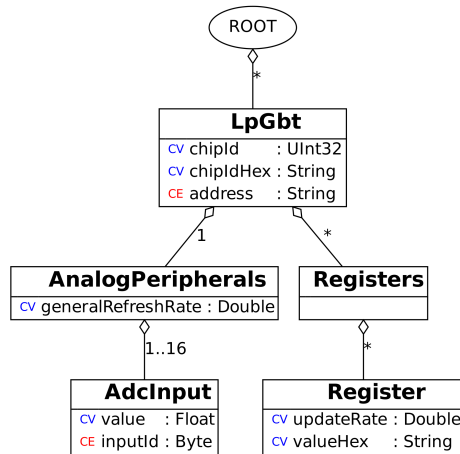


Figure 6.15: Quasar-generated OPC UA information model for an lpGBT device.

As illustrated in Figure 6.16, the quasar-built server internally wraps the LpGbtSw C++ library. The server exposes these capabilities via the address space, enabling remote clients to operate the lpGBT safely and coherently.

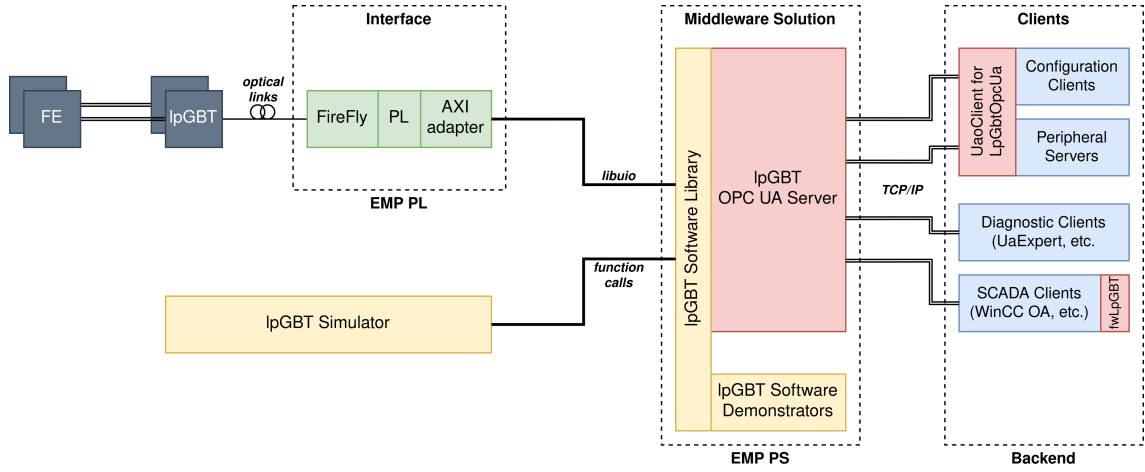


Figure 6.16: Architecture of the lpGBT OPC UA server ecosystem, including its interface with the hardware, and the connectivity to various clients via OPC UA. The server communicates with the lpGBT through the LpGbtSw library to access the hardware registers. External client applications using OPC UA over TCP/IP, including general-purpose diagnostic tools (e.g., UA Expert) and the SCADA system, which integrates lpGBT control into the DCS. Adapted from Ref. [72].

EMP Monitoring & Control OPC UA Server

The EMP Monitoring & Control OPC UA server [95] is a software component that facilitates remote, network-based access to the EMP hardware. Its primary function is to monitor system parameters, such as temperatures, voltages, and power states. Furthermore, it enables the monitoring of various peripheral devices on the EMP, such as the SI5345 clock multiplier and the Samtec FireFly transceiver modules, and provides access to the debug and reset functionalities of the EMP firmware. The complete information model of the EMP Monitoring & Control OPC UA Server is provided in Appendix B.1.

As presented in Figure 6.17, at its core, the server uses the empTools C++ HAL to interface with the EMP boards' subsystems. It communicates with on-board peripheral devices, accesses the FW-Debughub in EMP firmware, and reads data from the Xilinx SYSMON in PS and PL. All these data and controls are represented as nodes in the server's address space, providing a comprehensive view of the EMP's status and ensuring the safe and coherent operation of the EMP hardware within the DCS.

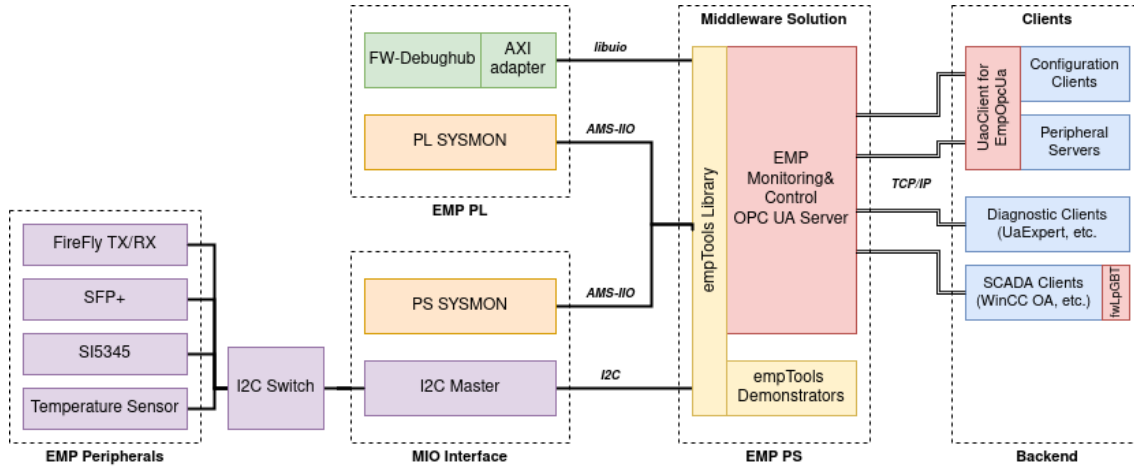


Figure 6.17: Architecture of the EMP Monitoring & Control OPC UA server. The server interfaces with the PL and PS SYSMON, the FW-Debughub, and the PS periphery, using the empTools library API. Like the lpGBT server, the EMP OPC UA server communicates with remote clients over TCP/IP, allowing SCADA applications and diagnostic tools to access the EMP’s status and perform control operations.

6.6. Contributions

Over the past three years, contributions to the EMCI-EMP ecosystem have primarily focused on the domains of EMP firmware, embedded software, and system integration. In the context of the EMP FPGA Firmware Reference Project, the scope of work entailed migrating the codebase from Vivado 2021.2 to 2023.2, designing and implementing the FW-Debughub IP-core, developing AXI4-Lite slave adapters, and comprehensively redesigning the firmware clock architecture. Issues related to clock domain crossing (CDC) were addressed in the emp-lpgbt IP core. Furthermore, the generic EMP-MGT IP core was modified to incorporate parameterization and a GUI, thereby facilitating GTH MGT channel selection in accordance with system requirements.

In the domain of software engineering, the empTools C++ library was designed and implemented to provide structured access to EMP monitoring and control interfaces. The EMP Monitoring & Control OPC UA server was developed using the quasar framework to expose EMP diagnostics and control functionality via a standardized network interface. Additional contributions included migrating and customizing the epos-bsp, with updates to the device tree and kernel integration. Further contributions to epos have been made through the migration to the most recent Vivado toolchain for epos-BSP and etools, with the primary objective of ensuring compatibility with the EMP FPGA Reference Firmware Project.

7. Benchmark and Characterization Study

As part of a technical evaluation campaign, a system benchmarking study was conducted. The objective of this study was to characterize the scalability, long-term stability, and performance of the EMCI-EMP system. In particular, the benchmarks targeted the scalability of the system when increasing the number of connected EMCIs from one up to the maximum of twelve, the stability of sustained operation over extended periods, and the achievable data throughput between the EMCIs and the EMP. In addition, the study considered how the attainable throughput depends on the EMP CPU load, thereby quantifying the trade-off between processing resources and data rate.

Due to the unavailability of FE hardware for the campaign's test setup, a dedicated data path configuration was implemented using the lpGBT's built-in loopback feature. This enables the retransmission of received downlink data on the uplink, thereby returning it to the EMP and establishing a closed communication path. As such, the system's full bandwidth can be exercised under controlled conditions, enabling a systematic evaluation of performance independent of external hardware constraints.

The ensuing sections first introduce the test environment in detail, focusing on the firmware and software components. Subsequently, the measurement methodology is outlined, followed by the presentation and analysis of the benchmark results. The final sections of the chapter discuss observed performance boundaries, identify potential bottlenecks, and outline directions for possible improvements and alternative implementation strategies.

7.1. Test Environment

The hardware setup utilized in this study, as shown in Figure 7.1, consists of a single EMP connected to 12 EMCIs. Regarding the software and firmware environment, the existing EMCI-EMP ecosystem served as the basis for implementing test-specific customizations. The PS of the EMP runs on an AlmaLinux 9 OS, based on the epos framework, and provides essential software support, including the UIO subsystem and key ecosystem libraries such as libuio, LpGbtSw, and empTools. In the context of firmware, the EMP FPGA reference project served as the fundamental basis. It was extended with a dedicated IP-core to enable eLink data communication with the

EMCI. The block was integrated into the reference project by interfacing with the emp-lpgbt IP-core's eLink data interfaces. The subsequent discussion will provide a comprehensive overview of the study-specific configurations, the developed IP-core, and the test-specific software application.



Figure 7.1: Benchmark study test setup comprising one EMP connected to 12 EMCIs.

7.1.1. IpGBT Configuration

The IpGBTs were configured to operate their uplink at 10.24 Gbit/s with FEC5 encoding, matching the settings of the EMP FPGA firmware reference project on which the test setup is based. Consequently, each uplink frame comprises a 224 bit wide eLink data field, which is further subdivided into seven 32 bit uplink groups. In contrast, the downlink frame consists of 64 bit, with 32 bit allocated for eLink data.

As already stated, the IpGBT implements multiple so-called "built-in test features" that can be grouped into two main categories: internal chip self-test and data-path validation. In this study, the latter is of particular interest. The data-path validation features compose a broad range of link testing capabilities, including loopback modes, test-pattern generators and checkers, dedicated test outputs, and eye-diagram monitors. These features facilitate system evaluation, production testing, and in-system debugging by providing standardized verification procedures.

7.1.2. The eLink Interface IP-core

Since the EMP FPGA reference firmware project does not provide a generic mechanism for accessing eLinks from the PS, as this is usually implemented in an application-specific manner for each specific FE, a dedicated IP-core was developed to provide this functionality. To this end, the eLink Interface IP-core was designed to facilitate PS access to the eLink data paths of the emp-lpgbt IP-core, thereby enabling the embedding of data into downlink frames and the extraction of received data from uplink frames. Figure 7.3 illustrates the architecture of the eLink Interface

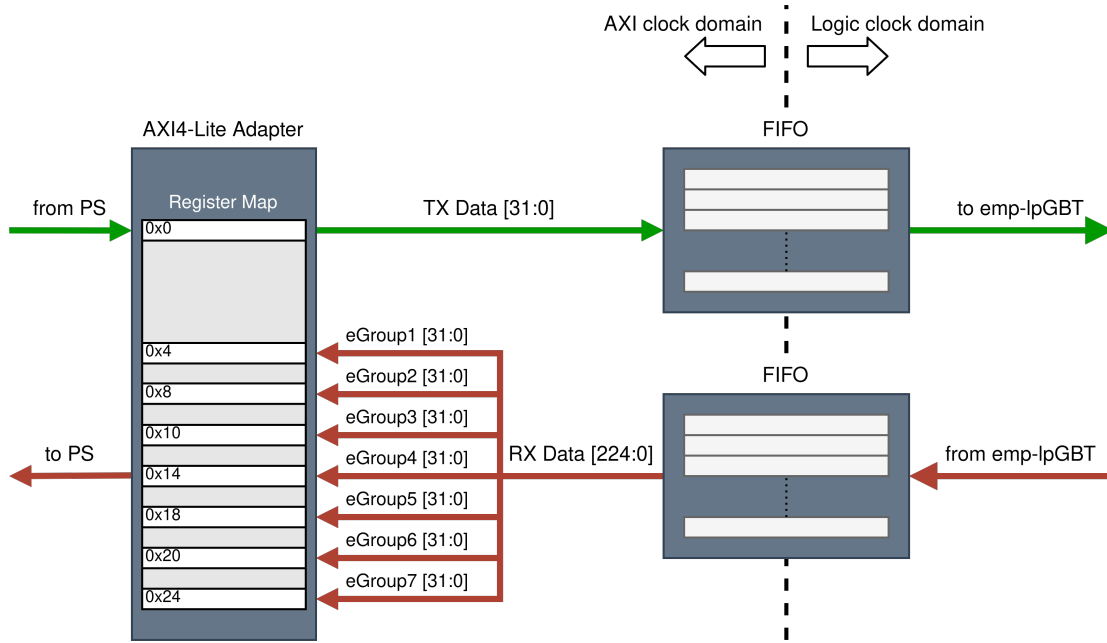


Figure 7.3: Functional diagram of the eLink Interface IP core and the data path of eLink data within EMP.

IP-core. As shown, the core is equipped with a dedicated AXI4-Lite adapter that enables register-based access from the PS. The adapter maintains a single 32 bit data register for downlink data and, in turn, seven 32 bit registers for uplink data, effectively partitioning the full 224 bit uplink frame into its seven eGroups. Since the AXI adapter operates in the 100 MHz clock domain, whereas the eLink data interfaces of the emp-lpgbt IP-core operate in the 40 MHz domain, the design incorporated two dual-clock FIFOs to synchronize the two domains. In addition, the FIFOs serve as short-term buffers, which are not explicitly required by the benchmark implementation, but still illustrate a general-purpose integration strategy for more complex system configurations. To complement the data path architecture and avoid reliance on a purely polling-based scheme, a dedicated interrupt logic has been

integrated into the core's design. Upon receiving and storing new uplink data in the corresponding register, the core asserts an IRQ to the PS, enabling event-driven processing of uplink data.

7.1.3. Software

On the software side, access to the eLink Interface IP-core is implemented using the UIO subsystem with libuio support. The AXI4-Lite register accessor was generated using the PuzzledLizardWizard tool, which produces a lightweight C++ HAL for the core's AXI4-Lite address space. This HAL enables direct interaction with the AXI4-Lite adapter of the eLink Interface IP-core and maintains a flexible and portable software structure.

Building on this HAL, a dedicated test application was developed to assess the EMCI-EMP system via the eLink Interface IP-core. As illustrated in the flowchart present in Figure 7.4, the application writes a 32 bit data word to the downlink data register within the AXI4-Lite address space. The eLink Interface IP-core reads it from the register, synchronizes it to the 40 MHz emp-lpgbt clock domain, and forwards the data word via the eLink user data path to the emp-lpgbt IP-core. There, the data is embedded into the lpGBT downlink frame, decoded, and finally transmitted via the MGT IP-core to the lpGBT. The lpGBT, configured in loopback mode, replicates the 32 bit data word upon reception into each of the seven available uplink data groups and transmits the resulting frame back to the EMP. On the FPGA side, the emp-lpgbt IP-core receives the returned data frame, extracts the user data, and forwards the 224 bit eLink data word to the eLink Interface IP-core. Here, the data is synchronized into the AXI timing domain, and each eGroup word is subsequently written to its corresponding uplink data register, after which the core asserts the IRQ to signal the availability of new data. In response, the test application acknowledges the IRQ and reads back the uplink registers to verify data integrity by comparing the received data words with the initially transmitted data. Errors are logged and counted to monitor reliability. Next, the application enters a programmed inter-transaction delay after which the sequence is repeated.

7.2. Testing Procedure

The benchmark study was conducted across a range of system configurations, from one to twelve parallel active EMP-EMCI links. For each configuration, the EMP and the EMCI were configured using the tools available in the software ecosystem, including programming the SI5345 clock multiplier, with empTools, and configuring the lpGBT into loopback mode via the LpGbtSw register API. Each EMCI-EMP

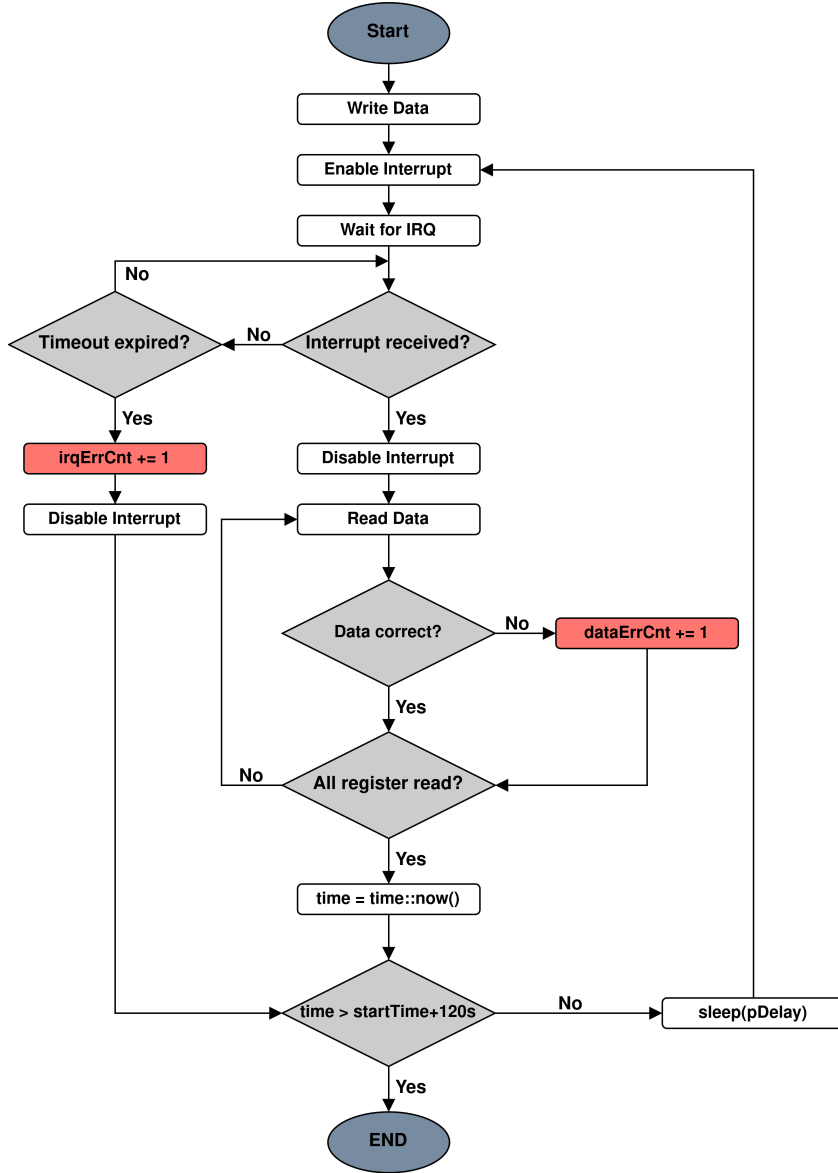


Figure 7.4: Flowchart representing the implementation of the benchmark application. For each iteration, the application writes a 32bit data word to the corresponding register of the eLink Interface IP-core and then waits for the associated IRQ. Once the interrupt is received, it is disabled, the received data is read from the register, and their integrity is verified. Finally, the application checks the time elapsed since the process started: if the fixed limit of 120s has not yet been reached, it sleeps for the configured inter-transaction delay before starting the next iteration; otherwise, it terminates and returns.

link in a given test series was operated through an independent instance of the benchmark application on the Zynq's PS. These processes were launched with high

scheduling priority and pinned to specific CPU cores, enabling a systematic study of the impact of additional links on CPU utilization.

For each system configuration, the test procedure followed the discussed data transmission routine, in which the programmed inter-transaction delay was progressively reduced between test runs from an initial value of 1 s down to 0 s in logarithmic steps. The systematic reduction of the inter-transaction ensured a finer granularity as it approached zero and allowed the system behavior to be probed across different timing regimes. Each delay setting was operated for 120 s, during which the benchmark application continuously issued transactions and recorded relevant metrics directly to a CSV file. These metrics comprise CPU utilization, the number of transmitted data frames, the observed error rate, and the number of missed interrupts, all logged for subsequent analysis. In addition, a separate process for each EMP-EMCI link was launched to poll all lpGBT ADC channels at 1 Hz using the LpGbtSw. The polling frequency of 1 Hz was chosen to reflect a typical update rate for slow control values. This additional workload was introduced to demonstrate that the lpGBT IC channel communication and the lpGBT eLink data transfer can operate simultaneously on PS and PL without interfering with each other.

The following section presents the results of the benchmark study. As outlined above, the number of active EMP-EMCI links and the number of CPU cores allocated to the benchmark application were varied during the system evaluation. In configurations with up to four links, each benchmark process was pinned to a separate CPU core. Once all four cores were occupied, additional processes were evenly mapped across the existing cores: in the configuration with eight active links, each core ran two benchmark processes; in the configuration with twelve active links, three benchmark processes shared each core.

7.3. Benchmark Results

The results of the performed tests are presented below. The analysis was divided into three groups. Initially, the data with a programmed inter-transaction delay greater than zero are examined. Subsequently, the data with a zero inter-transaction delay is analyzed, thereby presenting the maximum achievable data throughput. Finally, the result of a 12-hour stability test is provided, in which the system's stability under maximum load with 12 parallel EMCI links was assessed.

7.3.1. Programmed Delay

The presented data provide a quantitative overview of the system's performance and scalability across multiple configurations of the test setup. Figure 7.5 shows the correlation between the average throughput per EMCI and the corresponding measured CPU load. The results show an almost linear correlation for configurations in which the number of active EMCIs matches the number of available CPU cores. This linear trend indicates efficient parallel execution of independent transactions across CPU threads, with minimal contention on the bus interface. Once the number of EMCIs exceeds the number of CPU cores, the per-EMCI frame rate begins to decline, and the total CPU load increases disproportionately due to scheduling overheads and resource contention in the Linux kernel. In comparison, Figure 7.6 shows

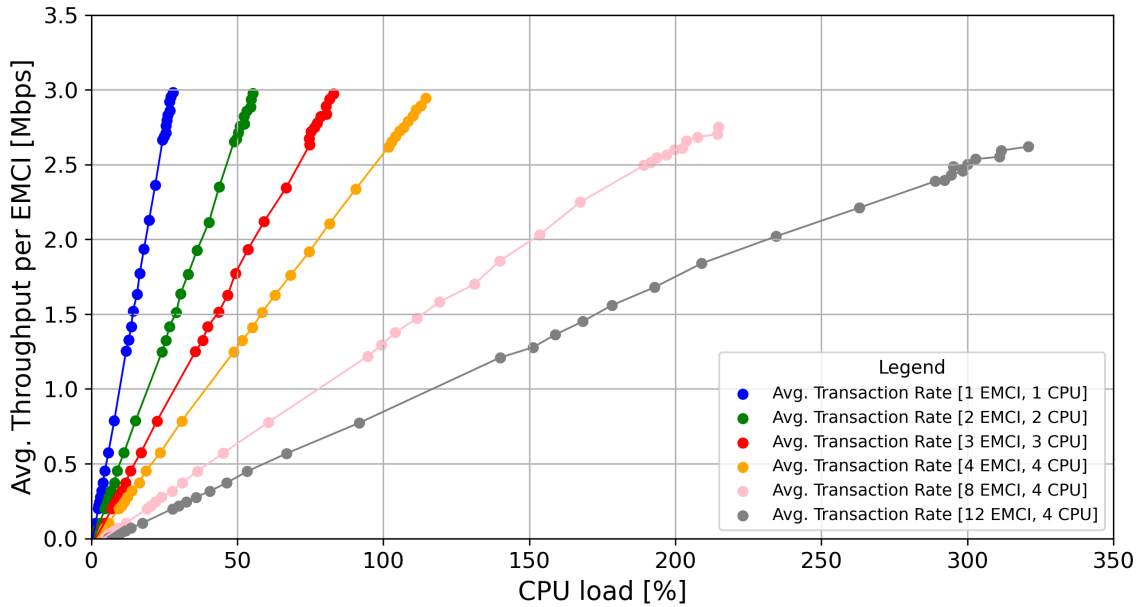


Figure 7.5: Average throughput per EMCI [Mbps] as a function of the CPU load [%] for different configurations of EMCIs and utilized CPU cores. The CPU load is given as the sum over all four cores, such that 100 % corresponds to one fully utilized core and 400 % to all four cores fully utilized.

the aggregate throughput of all links as a function of CPU load, demonstrating that the total frame rate scales nearly linearly up to full CPU utilization. The data indicate that neither contention among benchmark processes sharing a CPU core nor link-specific effects introduces measurable throughput variations; each additional link contributes approximately the same incremental frame rate, independent of the number of processes scheduled on a given core. Figure 7.7 depicts the dependence of the average throughput on the programmed inter-transaction delay, displayed

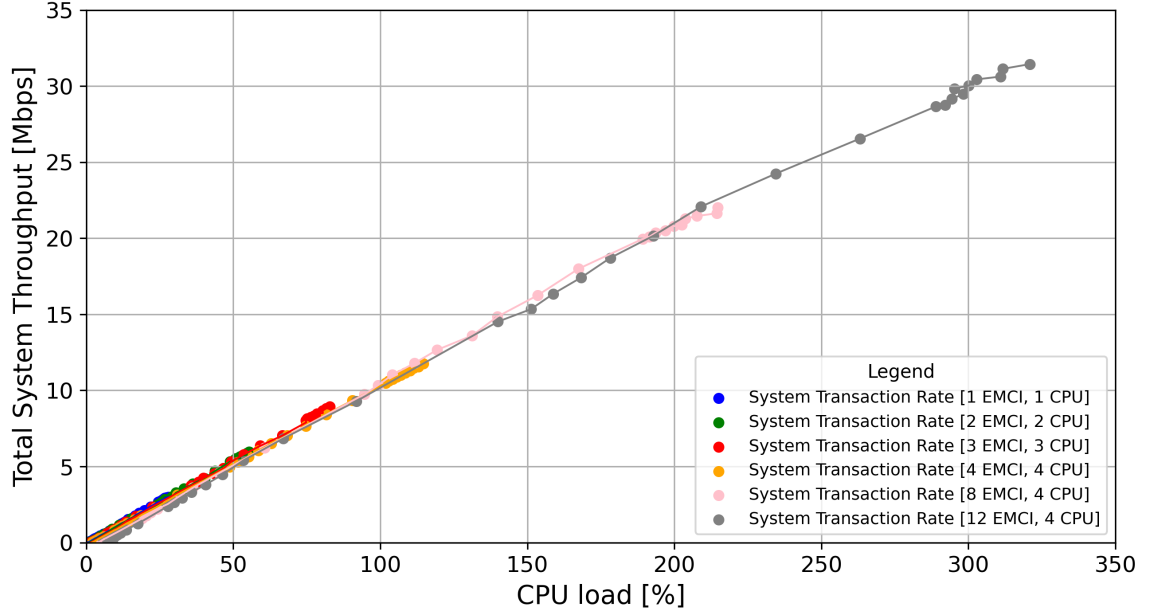


Figure 7.6: Total system throughput [Mbps] as a function of the CPU load [%]. Note the near-linear, almost identical scaling across all configurations.

on a logarithmic scale. At large delays, the throughput decreases inversely with the delay duration. However, for sleep durations below approximately 10 μ s, the frame rate improvement becomes marginal. This arises from the intrinsic timing granularity of the Linux scheduler and the overhead of frequent system calls, which prevent accurate timing and continuous transaction dispatch at microsecond-scale intervals. The corresponding CPU utilization as a function of the inter-transaction delay, shown in Figure 7.8, confirms that system load is directly correlated with the rate of AXI4-Lite transactions. CPU usage increases sharply for shorter sleep durations, reaching saturation at low delay values, where interrupt latency and context-switch overhead dominate. Conversely, at longer delays exceeding 1 ms, the CPU load converges toward idle levels as the transaction frequency decreases. Overall, the results demonstrate that the AXI4-Lite communication path scales efficiently with the number of CPU cores but remains fundamentally limited by software-driven transaction overhead and kernel-timing resolution.

7.3.2. Zero Inter-Transaction Delay

The following bar charts summarize the measured average and total frame rates for different system configurations without any programmed inter-transaction delay. In this test condition, all active CPU cores are fully loaded, allowing a direct assessment

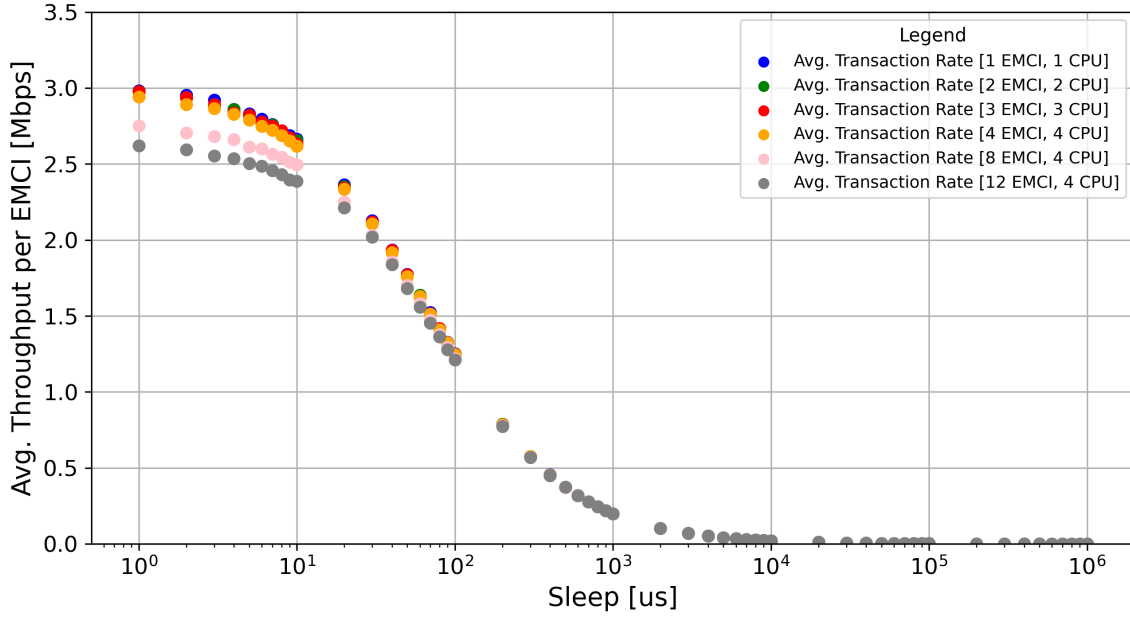


Figure 7.7: Achieved average throughput rate per EMCI [Mbps] as a function of the programmed delay [us].

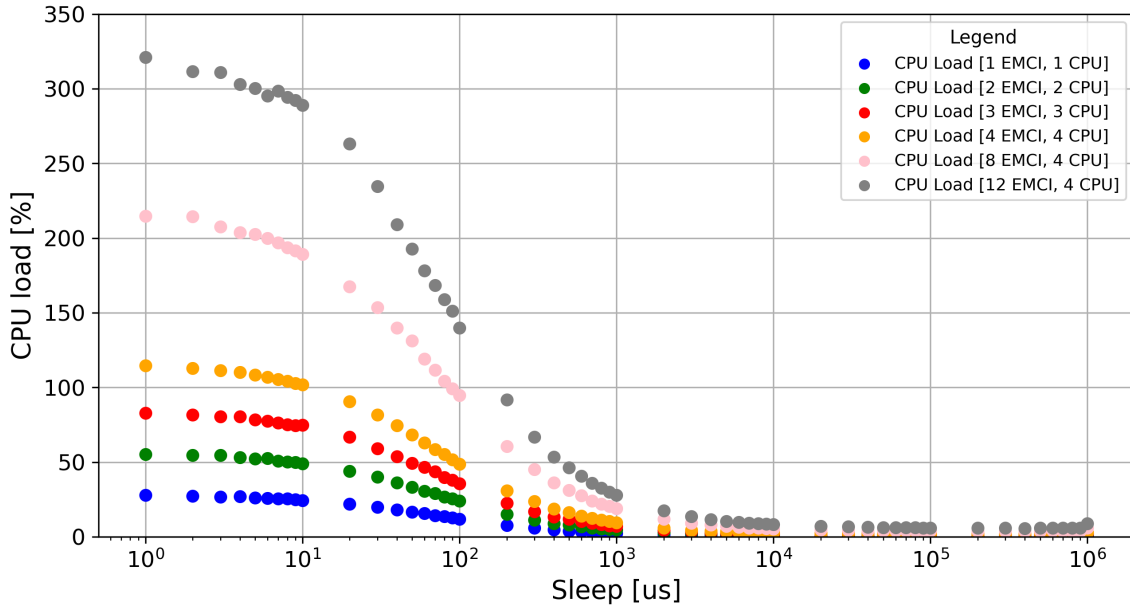


Figure 7.8: Average CPU load [%] as a function of the programmed inter-transaction delay [us] between requests.

of the achievable throughput and scalability of the implemented AXI4-Lite eLink accessor.

As shown in Figure 7.9, configurations with one, two, and three active EMCI connections exhibit comparable average throughput. However, configurations with four EMCI exhibit a substantial deviation from this trend. This effect is likely attributable to the cumulative scheduling and synchronization overhead that arises when all CPU cores are fully utilized. In contrast, the configuration with 8 and 12 active EMCI exhibits a significant decrease in per-link throughput, as expected. The

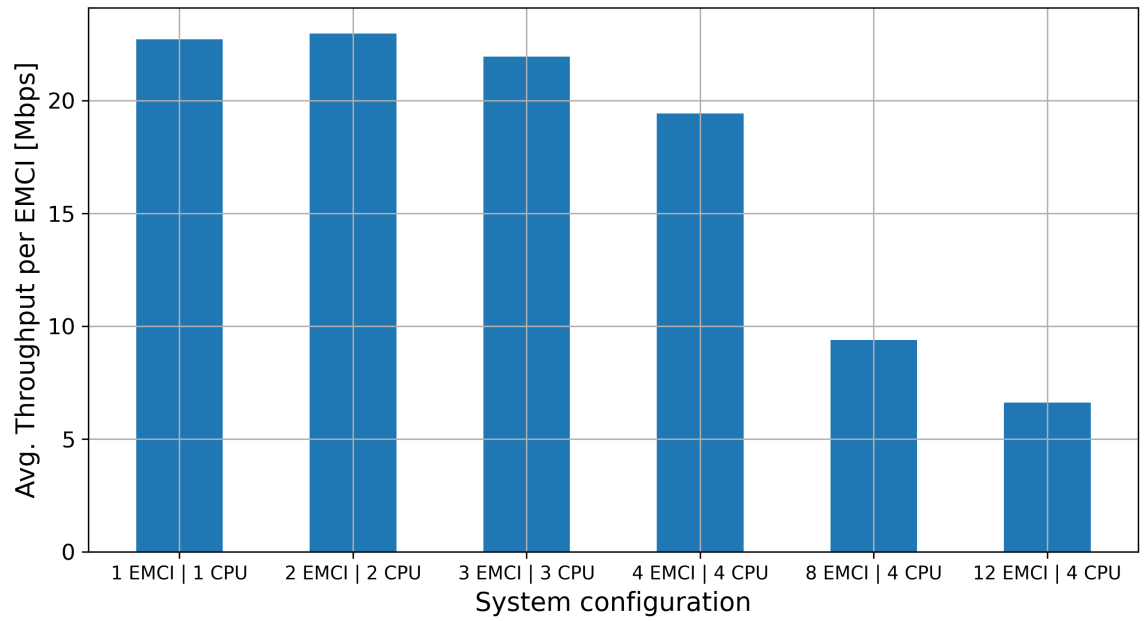


Figure 7.9: The bar chart shows the average throughput [Mbps] per EMCI for different system configurations with no delay between requests.

results presented in Figure 7.10 show that configurations employing the same number of CPU cores reach comparable maximum total frame rates, regardless of the number of active EMCI links. In particular, all configurations using four CPU cores converge to the same throughput ceiling. Therefore, increasing the number of links from four to eight or twelve does not increase the total frame rate; it simply distributes it across more channels. This behavior is consistent with the decrease in average per-link frame rate discussed earlier and correlates with limitations imposed by CPU resources, not by the AXI4-Lite bus or the lpGBT data path. Overall, these measurements show the intrinsic scalability boundaries of the AXI4-Lite-based eLink accessor under maximum load conditions.

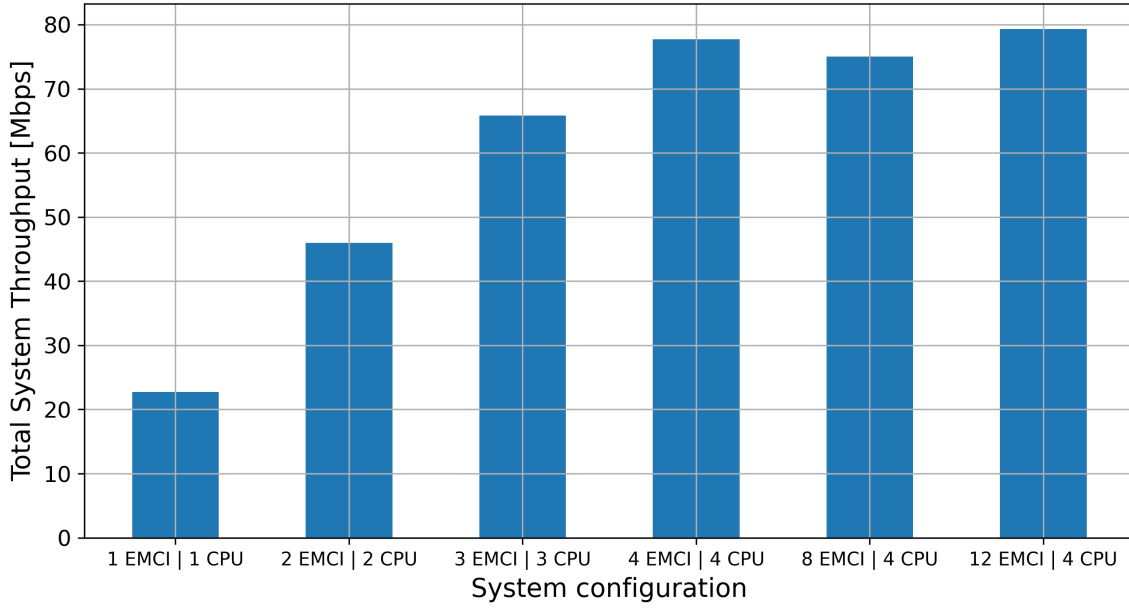


Figure 7.10: The bar chart shows the total achieved throughput [Mbps] for different system configurations with no delay between requests.

7.3.3. Stability Test

In addition, the system configuration with twelve EMCIs operating in parallel and without programmed inter-transaction delay was operated continuously for 12 hours. Throughout the test, no transmission errors, timeouts, missed interrupts, or synchronization issues were detected, thereby validating the system's stability during continuous operation. As illustrated in Figure 7.11, the accumulated number of successfully transmitted and received data frames during the test is presented and demonstrates a linear progression, indicating that the scheduled transactions were processed without interruption. In total, approx. 3246 Gbit of data were received error-free during the 12 hours.

7.4. Conclusion

The conducted benchmark study demonstrates that the EMP platform can reliably operate with the intended maximum of 12 EMCIs. Across all evaluated configurations, no transmission errors were detected, and all interrupt events were successfully acknowledged within their expected timing windows, even under sustained high-load

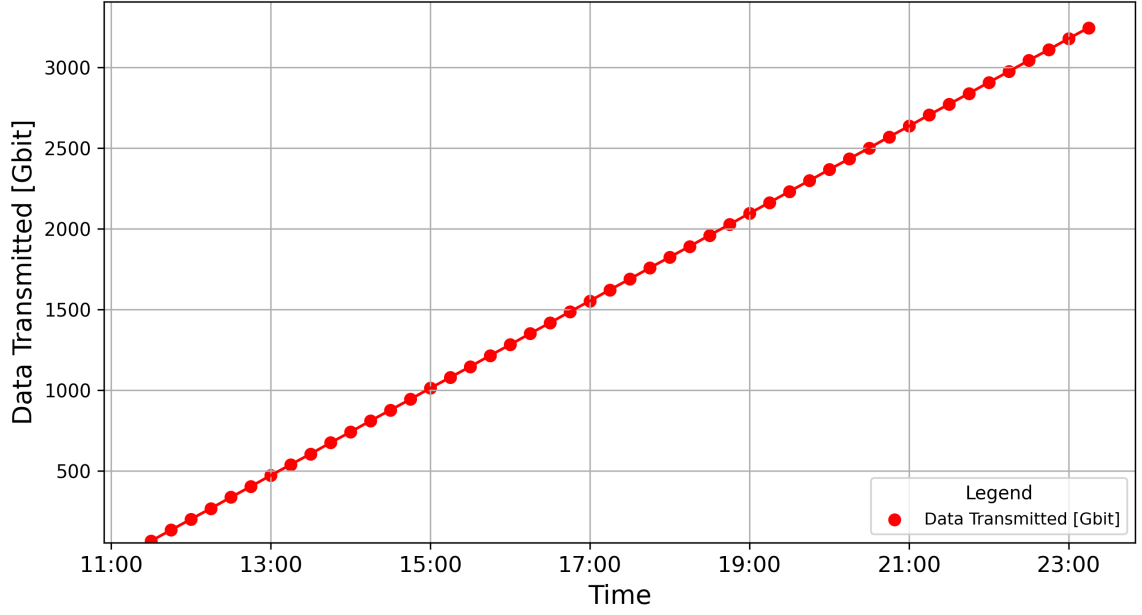


Figure 7.11: Data received without errors and missed IRQ over a period of 12 hours.

conditions. The study further verifies that the IC channel communication, implemented as a dedicated software-firmware path, can operate concurrently with the eLink transmission layers, without mutual interference. This confirms that the EMP architecture supports stable parallel operations of multiple different communication channels. It is important to note that the implemented test architecture intentionally prioritizes simplicity, transparency, and deterministic behavior over raw performance.

However, despite the robustness and deterministic behavior of the evaluated setup, several fundamental limitations emerge from the chosen AXI4-Lite communication approach. The protocol, while ideal for configuration and low-throughput control paths, inherently supports only single-beat transactions without burst capability. Each 32 bit access thus requires a complete handshake sequence, resulting in high protocol overhead and limited bandwidth. Combined with the copy-based access mechanism of the Linux UIO subsystem and the libuio library, which transfers data between kernel and user space for each operation, the processing load scales directly with transaction frequency. As a result, the achievable throughput is primarily constrained by CPU resources rather than by the AXI4-Lite interface.

These constraints make it challenging to handle more complex data structures, such as FE protocols. Thus, supporting different FE types would require each subsystem to implement its own custom protocol engines within the PL. This would con-

sequently result in heterogeneous EMP configurations, increased firmware development effort, and greater complexity in maintenance. Consequently, to ensure long-term maintainability and scalability of the system, a more flexible and efficient bus between the PS and PL is desirable, which provides higher throughput, structured frame handling, and reduced CPU utilization. In this context, AXI4-based transactions, with their burst-capable data paths and the ability to eliminate copy overhead between user and kernel space via DMA, offer a promising alternative.

Xilinx DMA IP Solutions

To overcome the intrinsic performance limitations of the AXI4-Lite-based eLink accessor, several Xilinx DMA IP-cores were evaluated as potential alternatives. The most relevant options, the AXI DMA [96] and AXI Central DMA [97] IP-cores, are both designed to provide high-throughput data movement between memory-mapped AXI4 interfaces and AXI4-Stream endpoints. Both cores offer mature, feature-rich implementations that sustain high transfer rates in SoC designs.

However, their architectural paradigm is optimized for generic memory-to-memory or memory-to-stream communications. Integrating these IP-cores into the EMP architecture would require additional translation logic to convert between the AXI4-Stream interface and the lpGBT-specific frame format. Furthermore, the Xilinx DMA solutions are accompanied by extensive software stacks that rely on the DMA-Engine framework for their configuration and control. While this abstraction simplifies integration for most applications, it also introduces significant overhead and limits flexibility. The driver hides many low-level transaction configuration options, making it difficult to implement specific, tailored control logic. In addition, both cores have a comparatively large resource footprint in terms of Lookup Tables (LUTs), Flip-Flop (FF), and Block RAM (BRAM), which conflicts with the available resources on the EMP, especially given the need for up to twelve independent data paths. Consequently, despite their technical capabilities, the Xilinx DMA IP-cores were deemed unsuitable for the targeted application, which prioritizes simplicity, resource efficiency, and low driver overhead.

These factors contributed to the decision to implement a customized DMA solution, the Slow-Control Direct Memory Access IP-core, explicitly designed to meet the requirements of the EMCI-EMP architecture. In contrast to generic Xilinx implementations, the SC-DMA IP-core provides a tightly integrated, low-overhead data path between PS and PL, with structured frame handling aligned with the lpGBT protocol and a minimal driver interface. The subsequent chapter will present the design, architecture, and validation of the SC-DMA IP-core in detail.

8. The Slow-Control DMA Subsystem

The benchmark study presented in the previous chapter highlights a central limitation of the EMCI-EMP communication architecture. Since the selected AXI4-Lite-based approach imposes strict bounds on throughput and transaction concurrency at the PS-PL interface, it fundamentally constrains the achievable system performance. Consequently, access to the eLink data path would require substantial custom, FE-specific logic. This, in turn, leads to fragmented firmware deployments, in which each subsystem integrating the EMCI-EMP system must implement its own dedicated PS-PL interface and control stack, reducing standardization across deployments and significantly increasing long-term maintenance effort.

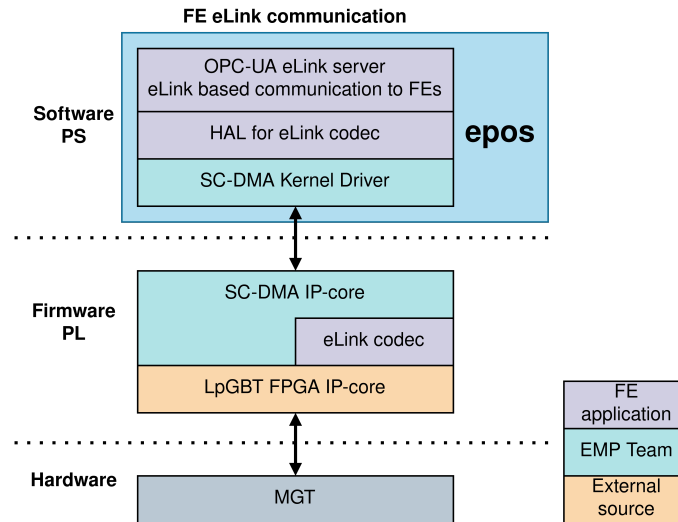


Figure 8.1: Software, firmware and hardware layers of the SC-DMA based FE eLink communication stack.

To overcome these limitations and address the absence of suitable COTS Data-Movement IP-cores, a Slow-Control Direct Memory Access (SC-DMA) subsystem has been developed. It implements a generic, high-performance lpGBT-compatible communication framework that enables the efficient transfer of slow-control FE data between the PS and the PL. The subsystem comprises a custom-developed DMA IP-core that implements the high-throughput data mover and a complementary custom Linux kernel module that provides a unified software interface for configuration, control, and data transfer. Figure 8.1 situates these components within the FE eLink

communication application stack of the EMCI-EMP software and firmware architecture and shows how they integrate with the surrounding layers.

This chapter first outlines the functional requirements that were defined and adhered to throughout the development process. Next, the architecture and internal functional units of the IP-core are discussed, followed by a description of the simulation and validation testbench used for verification. Finally, the functionality and software interfaces of the developed kernel module are discussed.

8.1. Design Requirements

The SC-DMA subsystem was designed based on a set of functional and non-functional requirements, tailored to the EMCI-EMP system. The most important design guidelines and target features relevant to integration, robustness, and operational reliability are outlined below:

- **Integration and Compatibility:** To ensure interoperability, the core must integrate seamlessly into the existing EMP FPGA firmware reference framework. Therefore, it must implement interfaces that enable seamless integration with the eLink data interfaces of the emp-lpgbt IP-core, and implement the system memory interface according to the FPGA's specifications.
- **eLink Frame Format:** The design must incorporate a flexible parameterization concerning the various eLink configurations of the connected EMCI, and consequently, the associated lpGBT frame format. The objective of this parameterization is to enable the IP-core to deserialize downlink eLink frames into lpGBT-compliant frames and to recover and serialize data from incoming lpGBT frames into eLink frames.
- **Resource Efficiency:** FPGA resource utilization should be minimized to achieve a balance between functionality, performance, and area efficiency. The scalability of the IP-core must be evaluated for various parallel EMP-EMCI link configurations to ensure the EMP FPGA reference firmware, including 12 SC-DMA instances, still fits the available resources of the EMP PL. Consequently, the use of lightweight logic constructs is recommended wherever feasible to minimize resource consumption.
- **Error Detection and Reporting:** The core should include mechanisms that can detect and report data transmission errors. This includes timeout events that indicate stalled communication or protocol violations in received data streams. Detected errors should be reported through a dedicated status register.

- **Fault Recovery and Watchdog Supervision:** To enhance the system's robustness, a functional unit capable of recovering from unexpected fault states, such as stalled FSMs, should be implemented. If inactivity or protocol violations are detected, an internal recovery mechanism should reinitialize the affected logic blocks, allowing the system to resume regular operation.
- **Software Overhead:** The IP-core should be designed to minimize the software overhead for control, configuration, and monitoring, limiting it to a small, well-defined set of instructions. Similarly, the corresponding Linux kernel module should be lightweight, providing only the essential abstractions necessary for reliable and efficient operation.

By adhering to these guidelines, seamless integration of the SC-DMA subsystem into the EMCI-EMP architecture can be ensured. In addition, they ensure that the core delivers robust, resource-efficient data transfer, preserves configurability to accommodate diverse lpGBT configurations, and keeps the software control layer lightweight and maintainable.

8.2. Architectural and Functional Overview

The SC-DMA IP-core is a custom firmware design developed to extend the EMCI-EMP ecosystem by providing a scalable, high-throughput data path for eLink access via system memory. The core is equipped with two 128 bit-wide, AMBA-compliant AXI4 interfaces: one for reading from system memory and the other for writing to it. Additionally, the core implements interfaces for integration with the emp-lpgbt IP-core, thereby supporting the lpGBT frame format.

The architectural design of the SC-DMA IP-core comprises six functional units, each with a clearly defined role within the high-speed slow-control data path. Figure 8.2 shows the architectural design and outlines the overall structure, communication interfaces, and the interconnections between the individual units.

The SC-DMA IP-core separates the uplink and downlink data paths into distinct chains, with each chain implementing its own dedicated AXI4 and emp-lpgbt facing interface. The downlink path consists of the AXI4-Read Engine, which retrieves eLink data frames from system memory, and the Downlink Frame Assembler, which formats eLink data into lpGBT-compliant downlink frames. Conversely, the uplink path consists of the Uplink Frame Collector, which monitors incoming lpGBT uplink frames to recover eLink data frames, and the AXI4-Write Engine, which commits the recovered eLink data frames to the designated memory region.

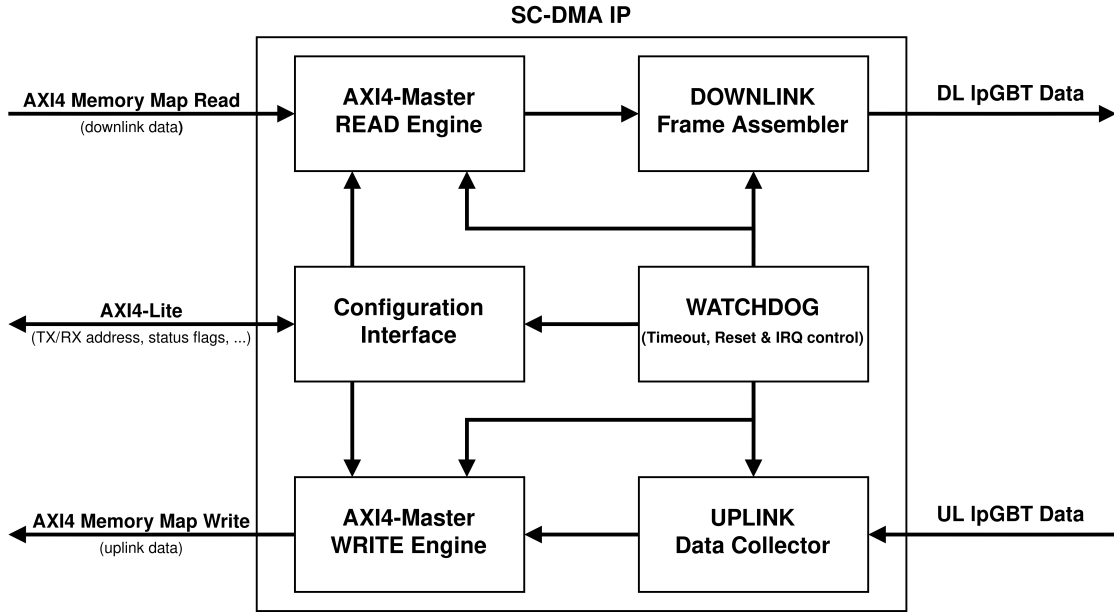


Figure 8.2: An architectural overview of the SC-DMA IP core and its constituting functional entities, including their basic interconnections.

In addition, two dedicated units, the Watchdog and the Configuration Interface, implement the IP-core's control and supervisory functionality. The Configuration Interface implements an 32 bit-wide AXI4-Lite-based register map interface that exposes a comprehensive set of registers, enabling the control and monitoring of the SC-DMA core. The Watchdog, on the other hand, serves as the supervisory unit of the SC-DMA core. It continuously monitors ongoing transactions to verify completion within predefined timing constraints. In case of a system timeout, a stalled FSM, or a protocol violation, the Watchdog initiates automatic recovery to restore proper operation. Furthermore, it governs the generation and handling of IRQs, ensuring reliable interaction with the PS.

The subsequent subsections will provide a comprehensive overview of the operational behavior and functionality of each architectural unit. However, it is first necessary to examine the configuration and parametrization that enable the IP-core's adherence to the lpGBT frame format, thereby enabling the interpretation and alignment of lpGBT frames. Given that the lpGBT protocol supports flexible eLink mapping and variable per-link data width, the core must reflect the exact configurations of the connected hardware, thereby ensuring that the data fields extracted from or inserted into lpGBT frames correspond to the expected eLink and eGroup assignments.

8.2.1. SC-DMA Customization GUI

The SC-DMA IP-core features a GUI that allows customization of each instance via pre-synthesis generic parameters, as illustrated in Figure 8.3. The GUI enables configuring the AXI4-Lite base address of the Configuration Interface, the AXI transaction identifier, the eLink frame delimiter, and parameterization of each uplink and downlink eGroup. During synthesis, the settings are propagated as VHDL generics, ensuring that each instantiated core operates according to the required hardware parameters.

The screenshot displays the SC-DMA Customization GUI with the following configuration details:

- Component Name:** scDMA_0
- Config Interface Addr:** 0xB0000000
- AXI ID:** "000"
- eGroup Configuration:**
 - Downlink eGroups:**
 - eGroup 1: 1 eLink channel (selected)
 - eGroup 2: Disabled
 - eGroup 3: 1 eLink channel
 - eGroup 4: 2 eLink channel
 - eGroup 5: 4 eLink channel
 - Uplink eGroup:**
 - eGroup 1: 4 eLink channel
 - eGroup 2: 4 eLink channel
 - eGroup 3: 4 eLink channel
 - eGroup 4: 4 eLink channel
 - eGroup 5: 4 eLink channel
 - eGroup 6: 4 eLink channel
 - eGroup 7: 4 eLink channel
- eLink Frame Delimiter:**
 - Start Byte:** "11010000"
 - End Byte:** "00001011"

Figure 8.3: Overview of the customization GUI of the SC-DMA IP-core. The GUI enables the parametrization to meet operational requirements for integration with lpGBT hardware and within EMP firmware.

The available configuration fields are summarized as follows:

- **BaseAddr:** Defines the memory-mapped address of the AXI4-Lite Configuration Interface. The address must be unique for each instantiated SC-DMA core and must be aligned with the Vivado address space configuration. This ensures correct address decoding and non-overlapping register access within the system memory map. The alignment ensures that the PS can reliably access each instance's control and status registers via AXI interconnects without contention or address ambiguity.
- **AXI ID:** Specifies the identification tag the core uses for AXI4 transactions: AWID, BID, ARID, and RID. This tag guarantees the association of read and write operations initiated by each SC-DMA instance in shared or multi-master AXI environments. Assigning distinct tags to multiple DMA instances connected to the same interconnect or memory bus ensures that responses can be correctly matched to their originating requests. Thus, transaction integrity is preserved, and interconnect-level conflicts during concurrent data transfers are prevented.
- **eGroup Configuration:** Defines the eGroup configuration of the connected EMCI hardware for both uplink and downlink directions. Each eGroup can be configured independently to contain one, two, or four active eLinks, or be disabled entirely. The configuration delineates the position and field width of each active eLink within the lpGBT frame (see Table 5.3). An increase in the number of eLinks per eGroup results in a reduction of the bandwidth available per eLink, whereas a decrease leads to an increase in the bandwidth per eLink. The configuration ensures the correct interpretation and alignment of eLink data within the lpGBT uplink and downlink frames.
- **eLink Frame Delimiter:** Defines the `Start Byte` and `End Byte` delimiters, which specify the unique byte patterns used to identify the beginning and end of an eLink frame in the uplink data streams.

8.2.2. Configuration Interface

The Configuration Interface serves as the control and status interface between the PS and the SC-DMA IP-core. It provides a structured set of memory-mapped registers that enable software to manage DMA address configuration, trigger transactions, read back status and error information, and acknowledge IRQs. The register map interfaces with the PS using the AXI4-Lite protocol and was designed using AirHDL. An overview of the implemented registers, including their offsets, types, and access, is shown in Figure 8.4. The available registers are summarized as follows:

Offset	Name	Description	Type	Access
0x0	START	Control register to initiate a DMA transaction.	Register	WRITE_ONLY
0x4	BUSY	Status register indicating an ongoing DMA operation.	Register	READ_ONLY
0x8	SRC_ADDR	Control register defining the physical base address of the source memory region.	Register	WRITE_ONLY
0xC	DST_ADDR	Control register defining the physical base address of the destination memory region.	Register	WRITE_ONLY
0x10	STATUS	Status register providing summarized feedback on the most recent transaction.	Register	READ_ONLY
0x14	ACK	Control register used by the PS to acknowledge completion after processing an interrupt.	Register	WRITE_ONLY
0x18	RESET	Control register for resetting internal FSMs and buffers of the SC-DMA core.	Register	WRITE_ONLY

Figure 8.4: Overview of the AXI4-Lite register map. This map describes the implementation of the configuration and monitoring interface of the SC-DMA IP-core. It shows each register's offset, name, and access properties.

- **START:** Configured to implement a single usable bit, which, when set to a logic high, initiates a DMA transaction. After the transfer is initiated, the internal state machines automatically execute the entire transaction. The bit is automatically cleared after one clock cycle.
- **BUSY:** Reading back the register indicates the current operational state of the DMA engine. To ensure the engine is idle, the software must poll this register before initiating a new transfer.
- **SRC_ADDR:** Configuration register that stores the 32 bit source memory address from which data is read during a DMA transfer. The register must be configured before asserting the **START** signal.
- **DST_ADDR:** Configuration register that stores the 32 bit destination memory address from which data is read during a DMA transfer. The register must be configured before asserting the **START** signal.
- **STATUS:** Provides transaction-related status information and error flags, including the last status of both the AXI4-Read and Write Engine, the AXI slave response indicating transmission status (via the **RRESP/BRESP** channel), a flag indicating observed timeouts, a dedicated flag indicating if a recovery was executed, and the IRQ acknowledge delay. The register is automatically updated during runtime, and the software must poll it after the IRQ is processed.
- **ACK:** Interrupt acknowledgment register. The register is configured to implement a single usable bit. When this bit is set to a logic high, it clears the pending interrupt flag and re-arms the interrupt controller for subsequent

events. The register must be written during IRQ handling and before initiating another AXI transaction. The register is automatically cleared after one clock cycle.

- **RESET:** The register enables a soft-reset mechanism for the AXI engine. Writing a logic high resets the internal control logic, the various FSMs, and registers to their default state. The register is automatically cleared after one clock cycle.

In summary, the AXI4-Lite Configuration Interface provides a streamlined, robust command interface between the PS and the SC-DMA core. By abstracting the complex AXI transactions into a minimal register protocol, software interactions are minimized, and the design goals of scalability and fault transparency are considered.

8.2.3. Watchdog

The Watchdog serves as the supervisory control unit of the SC-DMA IP-core. It ensures operational integrity by monitoring the state and progress of the AXI4-Read and AXI4-Write Engines to detect stalled behavior or transmission errors. In addition, the Watchdog is responsible for controlling IRQs and their acknowledgment.

Before a transaction begins, the Watchdog remains in the IDLE state. During this, all internal counters are cleared, and no interrupt is asserted. When a transaction is triggered, the Watchdog marks the DMA engine as busy and moves into the supervisory phase. There, the states of the read and write engines are continuously monitored, along with their activity durations. If both engines complete within their specified time limits, the transaction is marked as successful; if either engine fails to finish within its specified time window, a timeout condition is raised. In both cases, whether due to regular completion or timeout, the watchdog captures a summary of the transaction and writes it to the status register. Thereafter, the Watchdog transitions to the notification state, signaling completion via an IRQ. The kernel driver then has to acknowledge the IRQ and read back the status register. Only then can the Watchdog advance execution and either enter a recovery sequence if a timeout is detected or return to the IDLE state if the transaction was completed successfully. This "handshake" ensures that the software is explicitly informed of the last transaction, regardless of whether an error was detected.

If the transaction summary indicates a timeout, the Watchdog enters the recovery phase after the handshake completes. It then asserts a dedicated reset signal towards the SC-DMA logic to reset the FSMs, buffers, and FIFOs. The reset is

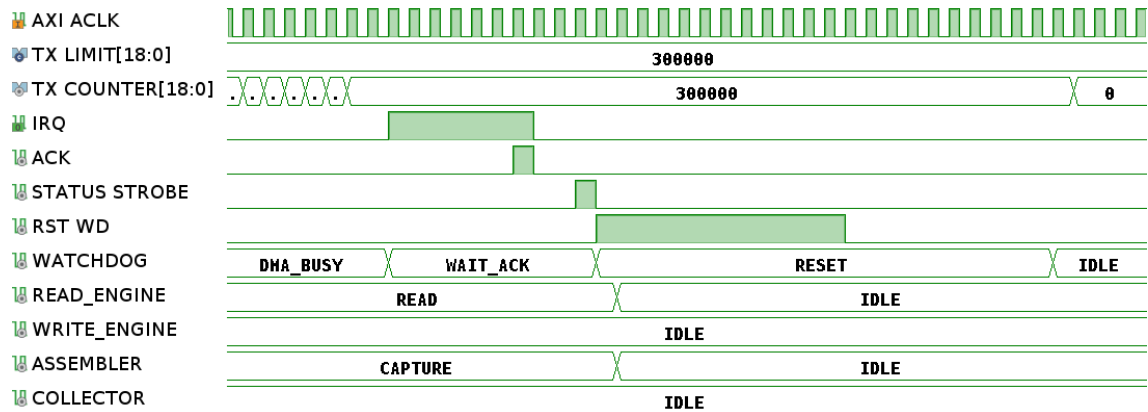


Figure 8.5: Simulation of an AXI-Read Engine timeout and the automated recovery sequence initiated through the Watchdog. It is shown that the transmit timeout counter increases until it reaches the configured limit `TX LIMIT`, upon which the Watchdog leaves the `DMA_BUSY` state and enters the notification state `WAIT_ACK`, asserting the `IRQ`. When the software acknowledges the `IRQ` (`ACK`) and the `STATUS STROBE` signals that the status register got read, the watchdog transitions to the `RESET` state, asserts the `RST_WD`, and forces all DMA FSMs to return to `IDLE`, restoring the SC-DMA subsystem to a well-defined state.

held for a predefined number of clock cycles, and upon release, the watchdog remains in the recovery state for the same number of clock cycles to allow the FIFOs to be reinitialized. Thereafter, the watchdog returns to the `IDLE` state, checks both AXI4 engines to clear the busy flags, and becomes available for new transactions.

Overall, the watchdog FSM provides a structured supervision scheme that combines monitoring, time-failure detection, software notifications, and autonomous recovery. This underscores the robustness requirement of the SC-DMA subsystem, particularly in fault conditions that otherwise would leave the data path in an undefined or stalled state. The corresponding VHDL implementation is provided in Appendix C.1.

8.2.4. AXI4-Read Engine

The AXI4-Read Engine implements the data-acquisition path of the SC-DMA IP-core and manages the transfer of eLink data from system memory into the PL for downlink transmission. Before the start of a transaction, the kernel driver must write a descriptor containing the source memory address to the `SRC_ADDR` register. Then, upon a triggered transaction, the AXI4-Read Engine automatically requests

a fixed burst of 16 data words from the designated source address, with each data beat corresponding to a single 128-bit eLink frame. Consequently, the entire burst transfers 16 eLink frames, i.e., the maximum number available per lpGBT. Due to strict memory alignment in both firmware and software, and to the AXI interface incrementing addresses during a burst transaction, the frames can be identified through their consecutive memory locations. That means eLinks are to be indexed by their memory offset from the base address, eliminating the need for additional headers or control information within the data stream.

Functional Description

The control logic of the AXI4-Read Engine is implemented as a four-state FSM synchronized with the 100 MHz AXI clock domain, as detailed in Appendix C.2. During the inter-transaction phase, the engine remains in the IDLE state and waits for the transaction trigger from the AXI4-Lite Configuration interface. As shown in Figure 8.6, upon reception, the AW-channel transaction parameters are configured, including the source address and burst length, and the ARVALID flag is set to initiate the address phase. Should the memory controller or AXI interconnect already have asserted ARREADY, the FSM will directly raise the RREADY flag to signal it is ready to receive data and will immediately transit to the read phase; otherwise, the FSM enters an intermediate handshake phase in which ARVALID remains set until the request is accepted, and only then RREADY is asserted.

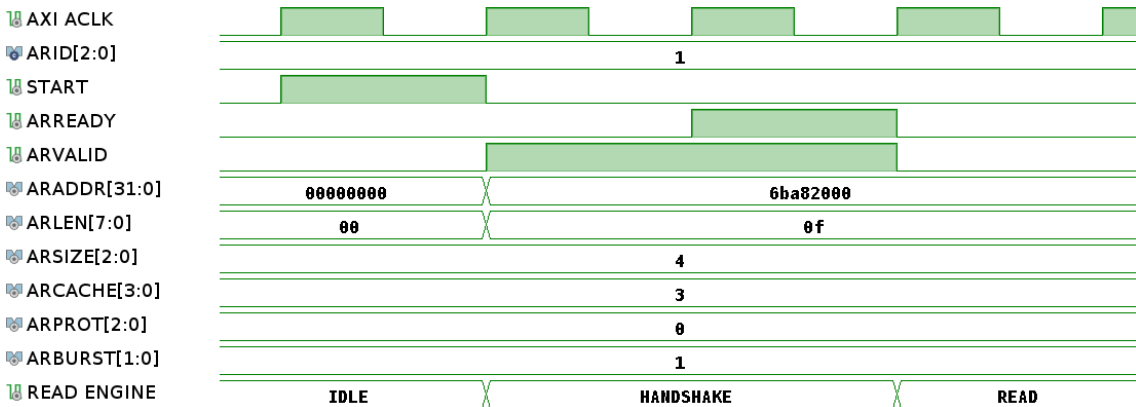


Figure 8.6: Address phase of the AXI4-Read Engine. Upon detection of the *START* bit, the engine leaves the *IDLE* state, configures the AXI AR-channel, and asserts *ARVALID*. The handshake is complete when *ARREADY* is asserted from the slave, after which the Read Engine transitions to the *READ* state.

During the read phase, as shown in Figure 8.7, the engine maintains the *RREADY* signal asserted while concurrently monitoring the *RVALID* channel to capture incom-

ing data beats. In addition, the RRESP channel is observed throughout the read phase, and upon a transaction error, the corresponding read-response field of the status register is set accordingly. It should be noted that the read engine supervises the entire transaction, but the received valid data beats are interfaced directly with the Downlink Frame Assembler for subsequent frame formatting. Upon reception of the final data beat, the FSM de-asserts the RREADY signal, enables the Uplink Frame Collector, and moves to the final FSM stage. The FSM stalls until the Watchdog enters the notification stage, thereby signaling the transaction's completion. The AXI4-Read Engine then returns to the IDLE state, where all process parameters are cleared, and the engine remains re-armed for subsequent transactions.

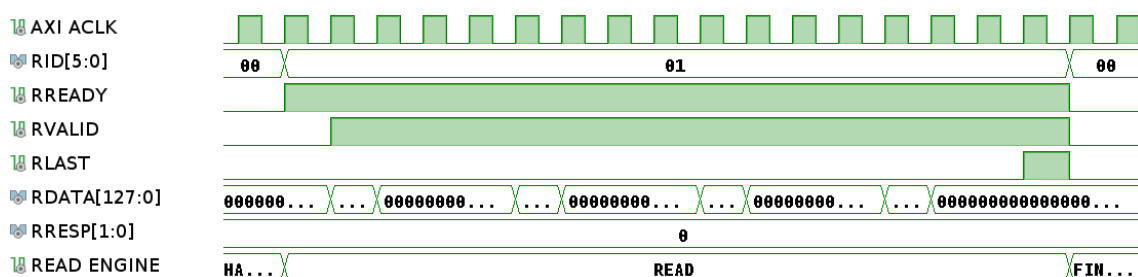


Figure 8.7: Read-data phase of the AXI4-Read Engine following a successful address phase. With RREADY asserted, the engine marks itself ready for the reception of data. The engine accepts 128 bit data beats on RDATA whenever RVALID is high, while RID and RRESP confirm the transaction identifier and the read response, respectively. The Read Engine remains in the READ state until the final beat of the burst and only then moves to FINISHED.

Overall, the AXI4-Read engine implements a well-structured FSM that ensures compliance with the AXI4 specification and enables a deterministic, high-throughput data flow between the PS and the PL. In addition, the combination of fixed-burst transactions, strict address alignment, and autonomous control minimizes PS intervention and improves efficiency for repetitive downlink data transfers.

8.2.5. Downlink Frame Assembler

The Downlink Frame Assembler implements the lpGBT frame generation entity of the SC-DMA IP-core. Its primary task is to format the eLink frames received from system memory into lpGBT-compliant downlink frames. To this end, the Assembler is implemented as an AXI clock-domain-synchronous FSM that manages the capture and buffering of data from the RDATA channel, the deserialization of eLink frames,

and the parallel assembling of lpGBT downlink frames. Since both the deserialization and assembling of lpGBT frames are implemented sequentially, an additional dual-clock FIFO was implemented. This FIFO serves as a buffer that temporarily stores all assembled downlink frames until packaging is complete. At the same time, it synchronizes the buffered frames to the 40 MHz clock domain to interface them to the emp-lpgbt IP-core. Figure 8.8 illustrates a simplified abstraction of the Downlink Frame Assembler, depicting the deserialization and assembly process. The corresponding VHDL source code can be found in Appendix C.3.

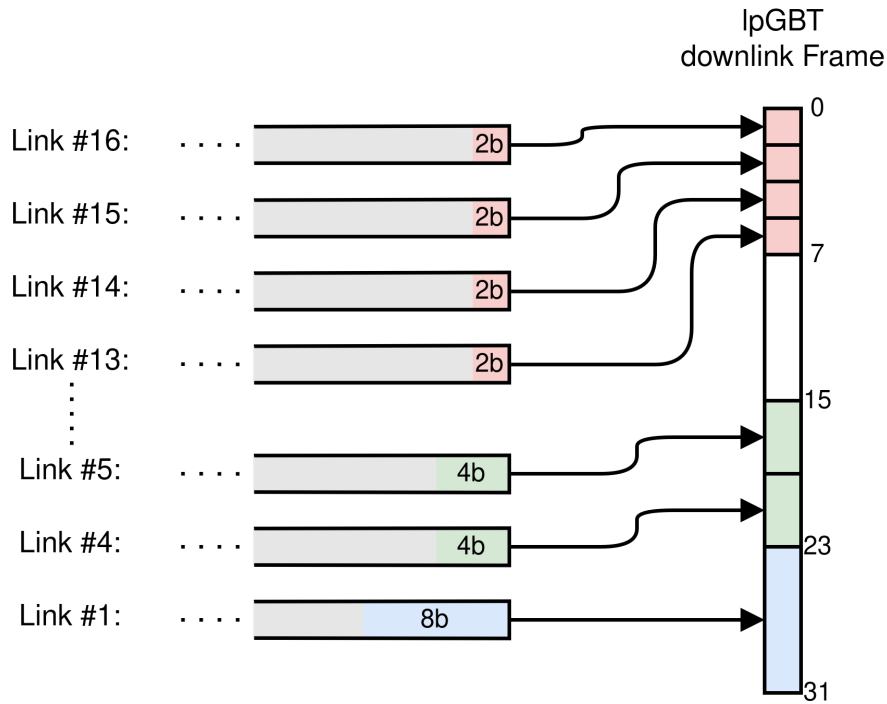


Figure 8.8: Abstract representation of the lpGBT frame assembly process performed by the Downlink Frame Assembler. The process extracts data slices from each buffered eLink according to the specific eGroup configuration and the related eLink field width, and copies them to their target positions within the lpGBT downlink frame. The bit-field widths shown are chosen for illustration only and are selected arbitrarily.

Functional Description

The Assembler remains in the IDLE state until the AXI4-Read Engine asserts the RREADY signal, thereby signaling readiness to receive data from the memory controller. As illustrated in Figure 8.9, the FSM moves into the CAPTURE phase when detecting RREADY, and it starts monitoring the RVALID signal for valid data beats on

the RDATA channel. Upon reception, each valid 128 bit word is stored sequentially into a dedicated array structure, ensuring the indexed buffering of the entire 16-beat data burst. Upon completion of the read phase, the FSM autonomously transitions to the SORT phase for subsequent deserialization and packaging.

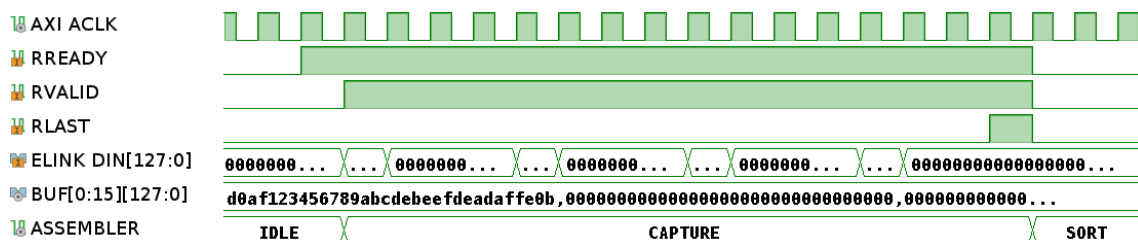


Figure 8.9: After the Assembler leaves the IDLE state and enters the CAPTURE phase, the valid data beats on RDATA are stored into successive entries of the BUF array until the end of the burst is detected through RLAST.

During the SORT phase, the Assembler performs a controlled deserialization of the buffered eLink data to assemble lpGBT-compliant downlink frames. Based on the eGroup configuration determined during synthesis from the VHDL generics obtained from the customization GUI, a configuration table was derived that specifies, for each active eLink, the corresponding group assignment, eLink index, and field width within the downlink frame. Owing to strict memory alignment, the use of address-incremental AXI bursts, and sequential buffering during the CAPTURE phase, the Assembler can iterate over this table and process only dedicated array indices, implicitly discarding any data that may have been written to non-active memory locations. For each table entry, the Assembler can therefore compute the target position within the lpGBT frame and copy the corresponding data slice from the buffered eLink frame to that destination. In addition, a dedicated per-frame pointer tracks the already consumed data and is shifted accordingly to the field width after each copy operation.

Once the configuration table has been fully traversed, the FSM transitions to the WRITE phase, where the assembled lpGBT frame is written to the FIFO. The FSM then returns to the SORT stage to generate the next frame. This cycle repeats until all buffered eLink frames have been fully deserialized and packed into downlink frames. Only after the final frame has been processed, the FSM asserts the SEND DATA flags and returns to the IDLE state. Figure 8.10 illustrates the simulation of such a SORT/WRITE cycle, showing assembly of an lpGBT frame and its subsequent transfer to the FIFO.

Since the FIFO is implemented as a dual-clock FIFO that synchronizes the buffered lpGBT frames into the emp-lpgbt timing domain, the readout must be controlled

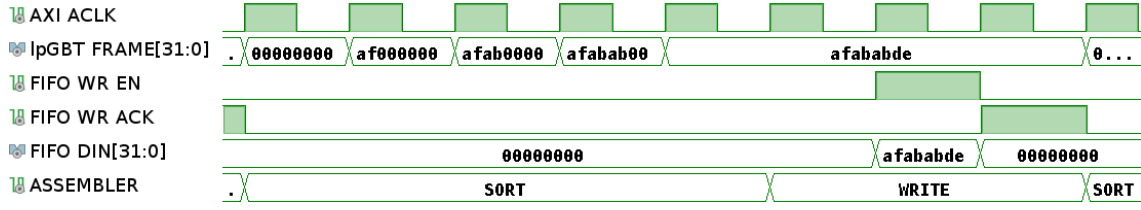


Figure 8.10: Single cycle of the *SORT/WRITE* of the Downlink Frame Assembler. After collecting eLink frames, the Assembler enters the *SORT* state, where it deserializes the buffered data into an lpGBT-compliant downlink frame *lpGBT FRAMES*. Once the frame is assembled, the FSM transitions to *WRITE* and asserts *FIFO WR ENABLE*, causing the 32 bit lpGBT frame on *FIFO DIN* to be consumed by the FIFO.

by a process operating in this specific domain. To this end, the *SEND* (*AXI ACLK*) flag asserted by the Assembler is first transferred from the AXI clock domain to the 40 MHz emp-lpGBT clock domain via a dedicated synchronizer, yielding the *SEND* (40 MHz) signal. As shown in Figure 8.11, once the *SEND DATA SYNC* flag is detected, a dedicated FIFO readout process asserts the *READ ENABLE* signal and initiates a structured readout of the FIFO content until it is completely drained. The resulting lpGBT frames are streamed directly to the external data ports of the SC-DMA IP-core, which are intended to be connected to the emp-lpGBT IP-core. As soon as the read interface indicates that the FIFO is empty, the read process de-asserts the read-enable signal. The Assembler FSM then clears the *SEND* flags, thereby leaving both the FIFO readout process and the Assembler FSM re-armed for subsequent transactions.

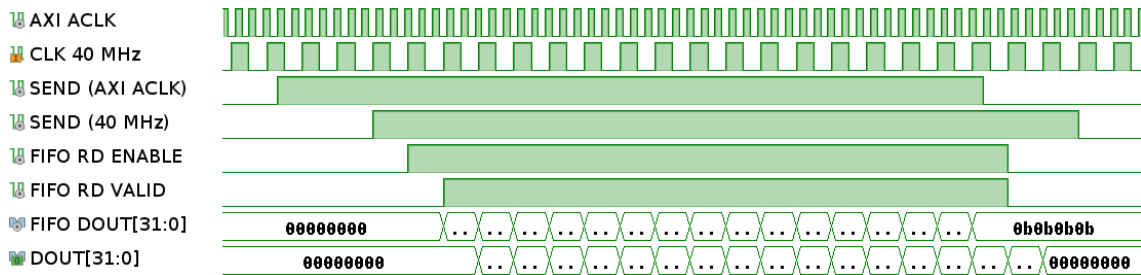


Figure 8.11: Simulation of the send process for downlink lpGBT frames. The *SEND* (40 MHz) flag synchronized from the *SEND* (*AXI ACLK*) flag set through the Assembler FSM enables the FIFO read controller, which asserts *FIFO RD ENABLE* and streams the 32 bit data frames on *FIFO DOUT* as long as *FIFO RD VALID* is high. The data are forwarded directly to the *DOUT* bus for interfacing with the emp-lpGBT IP-core.

In summary, the Downlink Frame Assembler serves as an intermediary between the eLink data frames and lpGBT-compliant downlink frames. The implemented

FSM provides a deterministic mapping between the two frame formats, enabling streamlined data alignment and full protocol compliance.

8.2.6. Uplink Data Collector

The Uplink Data Collector is the counterpart to the Downlink Frame Assembler. In contrast to the Downlink Frame Assembler, which deserializes eLink frames into lpGBT-compliant frames, the Uplink Data Collector extracts valid eLink patterns from the continuous lpGBT uplink stream and recovers the eLink frames through structure serialization. To this end, the Collector evaluates every active eLink channel in parallel within a dedicated logic block. This parallel topology is determined during synthesis using a configuration table similar to that used by the Downlink Frame Assembler. The table is derived from the VHDL generics specifying the uplink eGroup configurations.

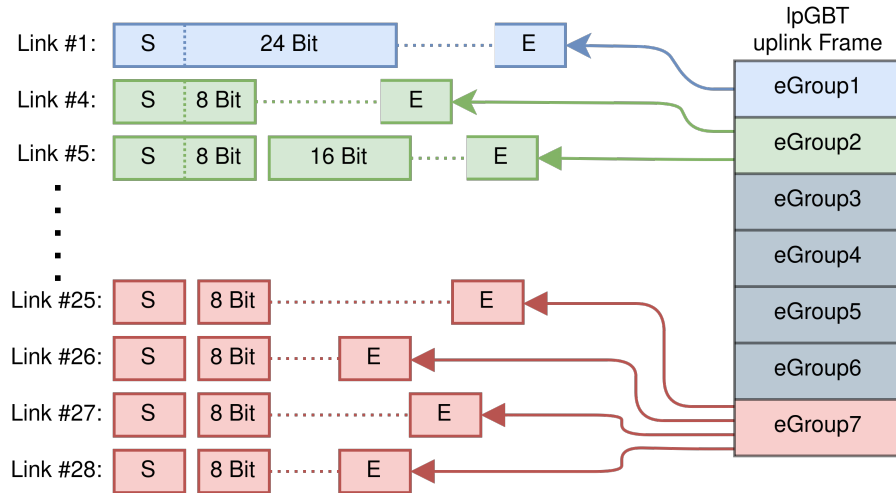


Figure 8.12: Abstract representation of the frame collection process implemented by the Uplink Frame Collector. For each lpGBT uplink eGroup, incoming eLink channels are continuously scanned to identify valid frame start (S) and end (E) delimiters. Upon detection of the start delimiter, the data patterns (8 bit, 16 bit and 32 bit) are buffered into a 3D array until the end delimiter is received. The frames are stored in the array such that their locations can later be used to determine the eGroup association and eLink index.

The recovered eLink patterns are buffered in a 3D array that assigns a dedicated column to each eLink index, with the depth sized to match the width of the corresponding eLink data field, enabling sequential, structured buffering of the extracted patterns. This enables the deterministic reconstruction of each eLink frame after it

has been fully collected. The recovered frames are finally written to a dual-clock FIFO that synchronizes between the 40 MHz and 100 MHz timing domains to interface with the AXI4-Write Engine. Figure 8.12 illustrates a simplified abstraction of the Uplink Frame Collector, depicting the extraction from the lpGBT uplink frame and the reconstruction of the eLink frames. The full VHDL listing of the implemented FSM is given in Appendix C.5.

Functional Description

Once the collector has been activated, the FSM transitions from **IDLE** to the **COLLECTION** stage, and each dedicated logic block begins monitoring its eLink channel within the lpGBT uplink frames, searching for valid eLink data. As soon as the start-of-frame delimiter is detected on a channel, the respective status is set to active, and data patterns from subsequent frames are written directly into the 3D array. Each logic block remains in this capture mode, appending data to the corresponding array column until the end-of-frame delimiter is detected. The channel's status is then set to finished, and the respective logic block remains stalled till all remaining active channels have terminated. Figure 8.13 summarizes this parallel capturing phase based on simulation results. To protect the capture phase against overflow condi-

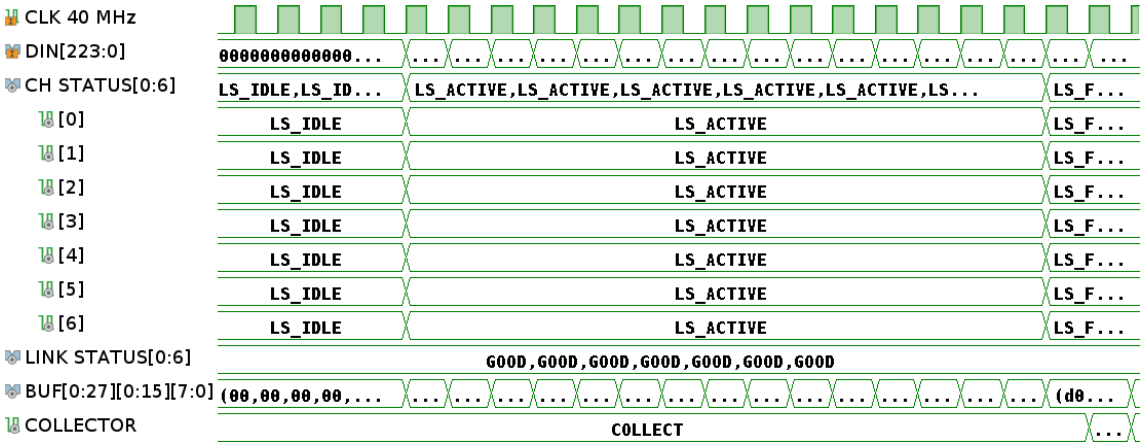
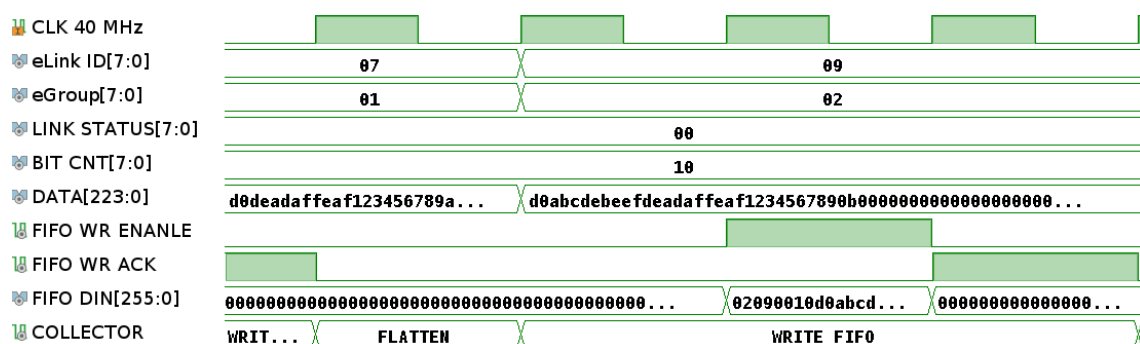


Figure 8.13: Simulation of the Uplink Collector during the **COLLECT** phase. Incoming 224 bit lpGBT frames on **DIN** are monitored per eLink channel, with each per-channel state transitioning from **LS_IDLE** to **LS_ACTIVE** once valid frame patterns have been detected. Subsequent data patterns are directly written to the **BUF** array while the status of each link is recorded in the **LINK STATE**. The Collector remains in the **COLLECT** phase until all active logic blocks **LS_FINISHED**.

tions or stalled channels, the FSM implements dedicated protection mechanisms. Suppose the buffer depth of a lane exceeds its predefined maximum of 128 bit, the



Once all logic blocks have either completed successfully or been fault-terminated, the FSM proceeds to the frame reconstruction stage. During this, the Collector iterates over the configuration table and sequentially reconstructs each eLink frame from the respective patterns buffered in the 3D array into a 224 bit vector. In addition, for each reconstructed eLink frame, a 32 bit metadata header is generated that contains the eGroup and eLink identifiers, the link status, and the data bit count. The header is concatenated with the flattened eLink frame to form a 256 bit frame, which is subsequently written to the dual-clock FIFO. The total frame length of 256 bit was selected because the eLink frame already occupies 128 bit, and concatenating any header would yield an unfavorable overall frame length. Since a width of 256 bit corresponds to two AXI data beats on the implemented 128 bit wide bus, this width was deemed the most reasonable to transmit the eLink frame including a metadata header.

8.2.7. AXI4-Write Engine

The AXI4-Write Engine serves as the counterpart to the AXI4-Read Engine, managing the transfer of received eLink frames into system memory, implementing the same 128 bit-wide, AMBA-compliant AXI4 interface. The embedded logic is managed by a dedicated FSM, synchronized to the 100 MHz AXI clock domain, that controls the entire transaction sequence, encompassing the address, data, and response phases; Appendix C.4 contains the corresponding VHDL code.

Functional Description

Upon completion of the Uplink Frame Collector, a **DONE** flag signals that the reconstructed frames have been written to the FIFO and can now be committed to system memory. Once the flag is detected, the write engine calculates the required burst length based on the FIFO occupancy.

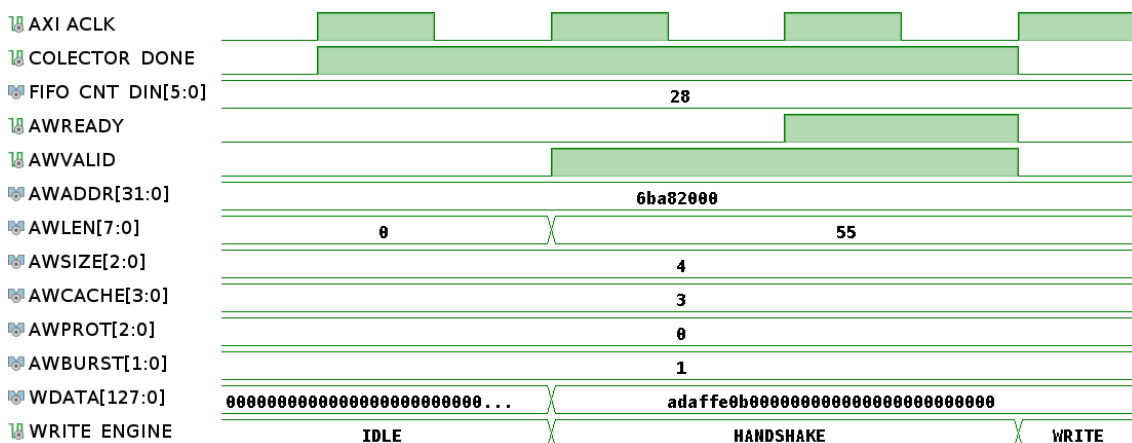


Figure 8.15: Address phase of the AXI4-Write Engine. When the uplink Collector signals completion (**COLLECTOR DONE**), the write engine configures the AXI AW-channel, asserts **AWVALID**, leaves **IDLE**, and enters the **HANDSHAKE** state. The transaction is accepted once **AWREADY** is asserted. At this point, the engine is ready to stream the data beats and transitions to the **WRITE** stage.

As shown in Figure 8.15, the FSM directly prepares the AXI AW-channel, using the memory address stored in the **DST_ADDR** register and the calculated burst length, and asserts the **AWVALID** signal. In parallel, the first 256 bit word is already read from the FIFO, and the AXI4 W-channel is prepared accordingly. Consistent with the AXI4-Read Engine, the FSM will directly move to the write phase in case the memory controller or interconnect already has asserted **AWREADY** and the handshake completes immediately; otherwise, the FSM will enter an intermediate handshake phase,

in which **AWVALID** remains asserted until the request is accepted. The prepared W-channel maintains the **WREADY** signal and the **WDATA** channel stable until the address phase is finished, and the first write handshake is completed.

Once the address phase is completed, the FSM moves to the data phase, where it continuously drives the AXI4 W-channel and streams the FIFO content into system memory. To enable a continuous data stream, the engine alternates between transmitting the lower and upper 128 bit of each 256 bit FIFO data word while, in parallel, fetching the next word from the FIFO. Figure 8.16 illustrates the interleaved process that ensures data is present on the AXI W-channel without idle cycles, thereby maximizing bus utilization.

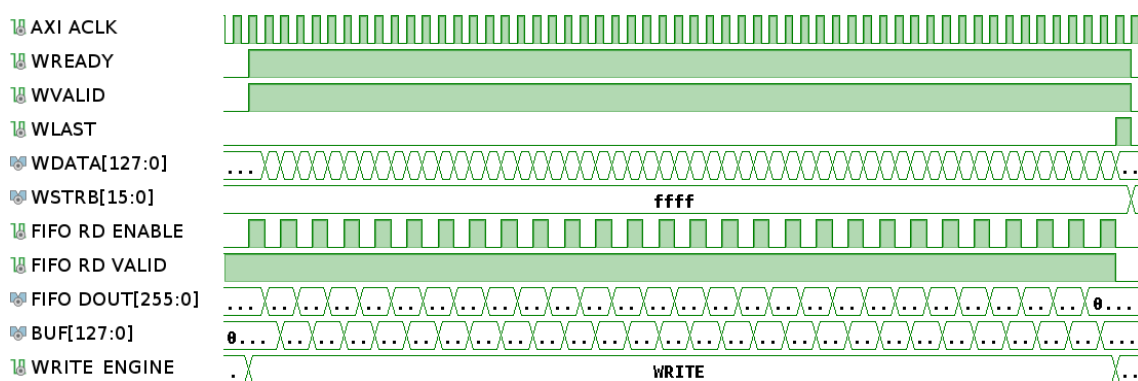


Figure 8.16: Write phase of the AXI4-Write Engine. During the **WRITE** state, the engine asserts **WVALID** and drives 128 bit words on **WDATA**, sourced from the 256 bit FIFO output **FIFO DOUT**. The process interleaves the FIFO so that the next data word is already read as soon as the previous data word is transmitted, ensuring continuous streaming of data beats on the AXI bus as long as **WREADY** is asserted by the slave. The final data beat is marked through **WLAST**, upon which the FSM transitions into the response phase.

As shown in Figure 8.17, on the final beat of the burst, the FSM asserts **WLAST** to mark the end of the transfer and then transitions into the response phase. There, the handshake of the last transmitted data beat is checked, and only then the **BREADY** signal is asserted. The FSM awaits now the response from the memory controller, and as soon as **BVALID** is detected and **BRESP** indicates successful completion, the FSM writes the appropriate write response field of the status register and then transitions into the final state, where it remains stalled until the Watchdog enters the notification stage, signaling transaction completion. Only then the AXI4-Write engine returns to the **IDLE** state, re-armed for subsequent transactions.

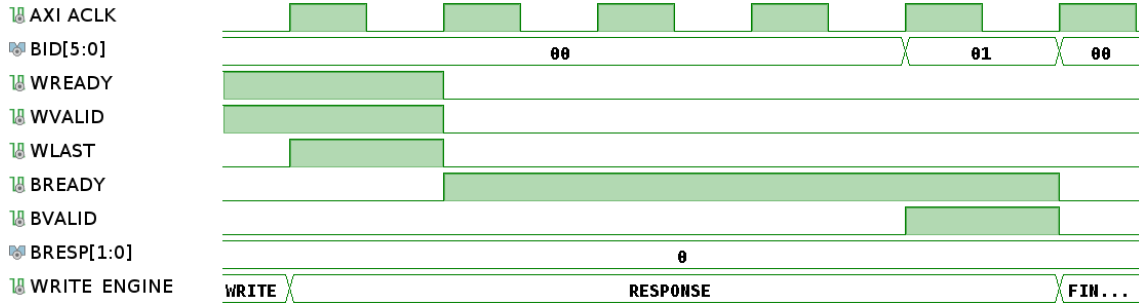


Figure 8.17: During the response phase, the engine checks for the final handshake on the last data beat of the write phase and only after asserts *BREADY*. Next, the engine waits for a valid response *BVALID* and *BRESP* confirming the successful completion of the burst write to system memory, and then transitions into the *Finished* state.

8.3. Testbench

To verify the functionality, timing behavior, and multi-channel concurrency of the SC-DMA IP-core, a simulation testbench was developed. This testbench is based on a block design that implements multiple SC-DMA instances, which are connected via AXI interconnect to a single memory subsystem. The objective was to replicate the EMP's architectural characteristics to create a simulation and debugging environment that closely resembles the real hardware deployment.

The implemented block design, as shown in Figure 8.18, instantiates six independent SC-DMA IP-cores that are connected to a shared memory model via AXI Crossbar IP-cores [98]. The EMP itself offers up to four AXI-FPD HP slaves for interfacing with the system memory. In the target firmware, up to 12 SC-DMA instances are foreseen, which corresponds to 24 AXI masters that must be distributed across these HP slaves. For the testbench, six SC-DMA IP-cores were chosen to emulate half of the maximum configuration while already reproducing the intended per-interconnect-fan-out and arbitration behavior. To this end, four dedicated AXI Crossbar IP-cores are used, evenly distributing the 24 masters so that each AXI HP slave serves a total of six SC-DMA master interfaces. Since the testbench is intended to resemble the actual hardware configuration, the six SC-DMA IP-core are connected via two AXI Crossbars to independent AXI Block RAM Controller IP-cores [99], each of which is connected to an independent port of the implemented memory model. The memory model itself is implemented using a pre-programmed Block Memory Generator IP-core [100] that provides data frames to the AXI4-Read engine during simulation and accepts write-back traffic from the AXI4-Write engine. Additionally, two AXI Verification IP-cores [101] are integrated as protocol monitors, one between each AXI Crossbar and Block RAM Controller. These Verification

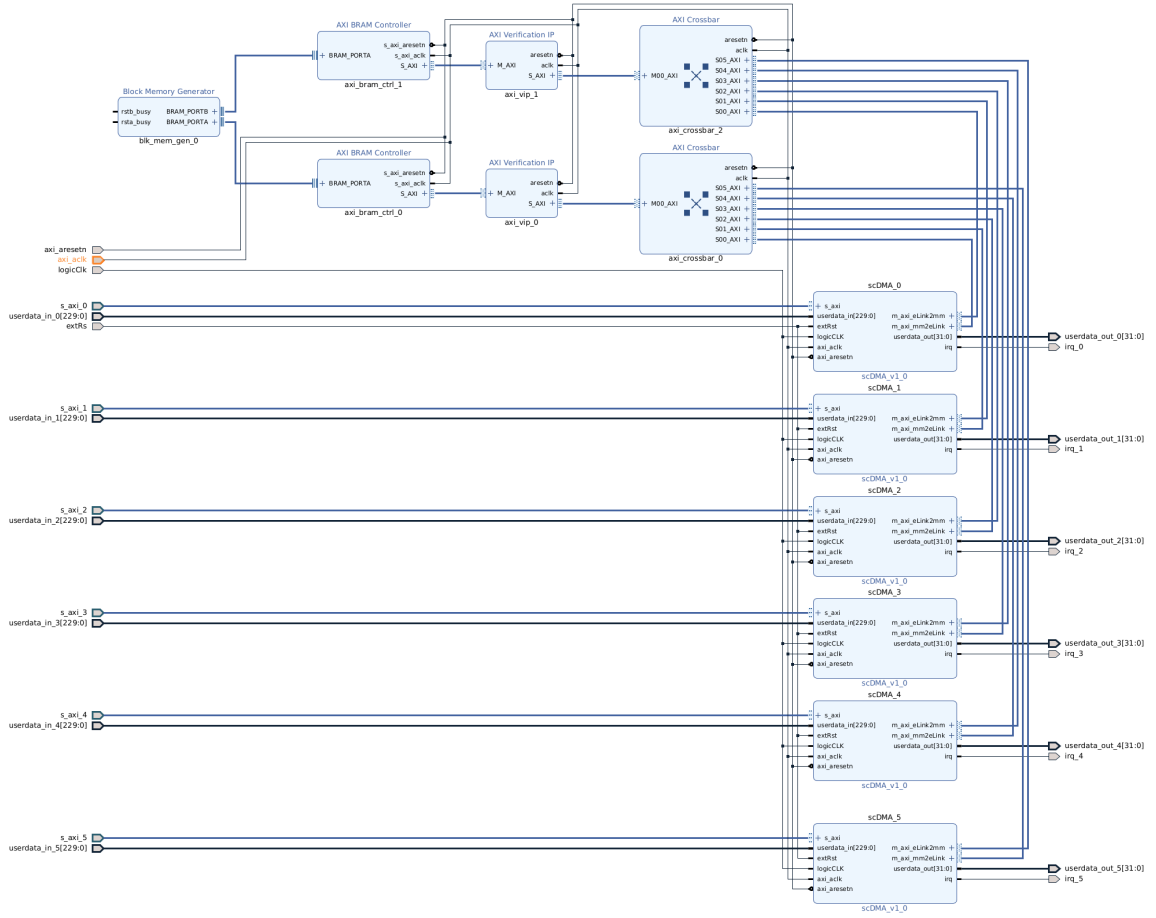


Figure 8.18: Block-level architecture of the SC-DMA validation testbench. Six instances of the SC-DMA IP-core are instantiated in parallel, each connected via its AXI4 master interface to the shared memory subsystem composed of a Block Memory Generator and two AXI BRAM Controllers. Separate AXI Crossbars fan out the read and write channels, enabling concurrent access to the same BRAM-based memory by all SC-DMA instances. The integration of additional AXI Verification IP-cores into the shared AXI bus enables transaction-level debugging and diagnostics of the AXI protocol during simulation.

IP-cores are configured to pass-through AXI transactions, thereby analyzing the transactions to provide transaction-level protocol debugging and detailed diagnostics during simulation.

The simulation is implemented using a dedicated VHDL testbench that integrates a stimulus process for each SC-DMA IP-core to manage AXI4-Lite register transactions for configuring each instance and to trigger transactions via the Configuration Interface. Before transactions are initiated, these processes configure their respect-

ive SC-DMA instance by writing the source and destination address descriptors into the core's corresponding registers. Then, the **START** bit triggers the engine, and each SC-DMA instance starts requesting data from the block memory, assembling downlink frames, and streaming the lpGBT frames onto the dedicated downlink interface. To emulate the commonly used hardware test setup that relies on the lpGBT loopback mode (see Sec. 7.1.1), the testbench mimics this mechanism. Therefore, each downlink frame is concatenated seven times into a 224 bit vector and returned to the respective eLink data interface of the respective SC-DMA instance. The received lpGBT frames are subsequently formatted back into eLink frames and written back via the AXI4 write interface to the corresponding memory region of the block memory.

Upon completion of a DMA transfer, the SC-DMA cores assert the IRQ lines, which are monitored by the respective stimulus processes. The stimulus process immediately acknowledges this by writing to the dedicated IRQ-clear register and subsequently reading the status register. This re-arms the IP-core, thereby enabling the simulation to process multiple consecutive transactions. Consequently, the model facilitates verification of back-to-back operations, a critical component for evaluating the SC-DMA architecture under continuous, sustained loads. In summary, the

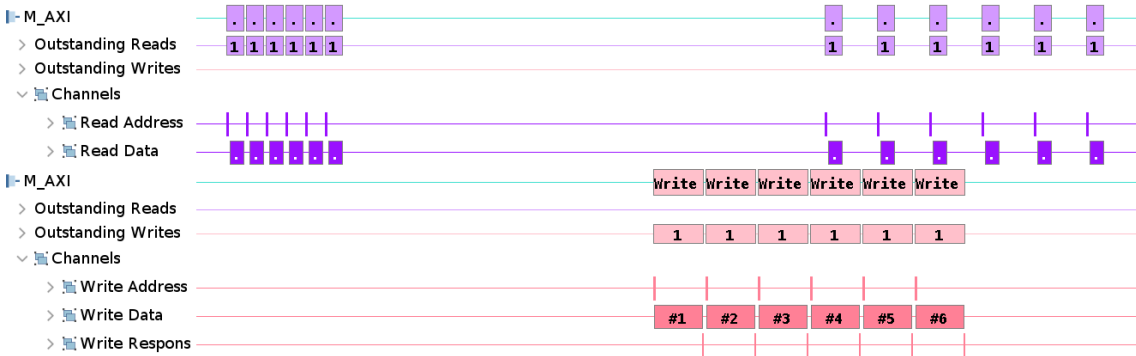


Figure 8.19: Transaction-level simulation of the multi-channel validation testbench. The Figure shows the IRQ signals for each SC-DMA IP-core, along with the AXI protocol view obtained from the AXI Verification IP-cores, which summarizes the activity traced between the AXI crossbar master interfaces and the AXI BRAM Controllers. The simulation is used to verify the AXI protocol compliance of the SC-DMA IP-core as well as the arbitration, ordering, and latency behavior of the AXI Crossbars.

testbench provides a controlled and realistic environment for assessing both the standalone behavior of the SC-DMA IP-core and the concurrent operation of multiple instances under conditions that closely resemble the EMP hardware architecture and the commonly used test setup. As illustrated in Figure 8.19, the testbench enables the systematic verification of AXI protocol compliance, arbitration behavior,

latency, concurrency, and interrupt handling across all instantiated DMA channels. Consequently, the testbench provides a robust platform for functional verification and debugging of the SC-DMA IP-core.

8.4. Scalability Analysis

The evaluation of the SC-DMA IP-core resource footprint was conducted to assess its feasibility for integration into the EMP FPGA firmware. Since the baseline firmware already implements 12 IC channel pipelines, MGT instances, and lpGBT frame encoding/decoding fabrics, resources are already constrained. In addition, extremely high occupancy is, in any case, unfavorable, as it increases routing complexity and restricts possible FE-specific additions.

Overall, the scaling of the SC-DMA IP-core's resource footprint is mainly driven by structures that are subject to specific configurations, primarily through the number of active eLink channels. This applies to both the Downlink Frame Assembler and the Uplink Frame Collector implementations, which provide per-channel logic, storage structures, and other components that scale with the number of active eLinks. Conversely, the AXI4-Read Engine, the AXI4-Write Engine, the AXI4-Lite Configuration Interface, and the Watchdog implement entirely static structures that are mostly independent of the IP-core's configuration. In the following discussion, therefore, only the Assembler and the Collector are reviewed in detail to determine their specific resource scaling and to consider their timing behavior. Subsequently, the SC-DMA IP-core is integrated into the EMP FPGA reference firmware, and the resulting post-implementation resource utilization is reviewed to confirm that the overall footprint remains compatible with the available PL resources.

8.4.1. Frame Assembler: Resource and Timing Scaling

The Downlink Frame Assembler is designed to deserialize eLink frames and assemble lpGBT downlink frames. This logic is implemented as a single block that is reused for all eLinks, whereas the combinational logic remains effectively independent of the number of active eLinks. The only structure that scales accordingly to the active eLinks is the pointer array that marks the position of already consumed bits within each eLink buffer. This results in a nearly linear increase in FF usage and negligible LUT overhead.

However, the timing behavior of the implemented FSM is adversely affected by an increasing number of eLinks. As summarized in Table 8.1, the frame assembly latency grows approximately linearly with the number of active eGrops for a fixed

number of eLinks per eGroup. This follows from the fact that the Assembler implements only a single datapath, so exactly one eLink data slice is copied into the lpGBT frame per clock cycle. Therefore, the number of clock cycles scales with the total number of eGroups to be processed. At the same time, increasing the number of eLinks per eGroup reduces the number of bits available per eLink within each lpGBT frame. Consequently, the number of lpGBT frames that must be generated to transmit a complete eLink frame is doubled when 2 instead of 1 eLink is active within an eGroup, and again if 4 instead of 2 eLinks are active within an eGroup. The same factors also apply to the number of write cycles required to push the generated lpGBT frames into the FIFO, further contributing to the overall latency.

active eLinks / eGroups	Latency [ns]			
	1	2	3	4
1	800	960	1120	1280
2	1920	2560	3200	3840
4	5120	7680	10 240	12 800

Table 8.1.: *Assembler latency for different combinations of active eGroups and eLinks.*

Overall, the Downlink Frame Assembler is designed as a trade-off between resource efficiency and assembly latency. Its single datapath architecture is highly resource-efficient, but the assembly latency increases with the number of active eLinks and eGroups. However, since the downlink path is not time-critical, the resulting latency is acceptable and has been deliberately traded against the reduced resource usage.

8.4.2. Frame Collector: Resource and Timing Scaling

The Uplink Frame Collector is designed to process every received lpGBT uplink frame on a cycle-by-cycle basis and therefore implements parallel datapaths to handle each active eLink separately. Each of these datapaths is implemented as a discrete logic block that handles the per-lane control paths, including collection, buffering, state tracking, and pointer management. Since these logic blocks are replicated for every active eLink and implemented as dedicated hardware, they primarily drive the Collector's resource footprint. As a result, with more active eLinks, LUT usage increases approximately linearly, while collection latency remains stable due to the parallel architecture.

The post-synthesis utilization results listed in Table 8.2, show how the LUT and FF usage scale according to the number of active eGroups and eLinks. For a fixed

eGroup configuration (1, 2, or 4 eLinks per eGroup), both LUT and FF utilization increase almost linearly with the number of active eGroups. This results from the fact that each additional eGroup adds a fixed number of additional eLinks, and the logic blocks are replicated accordingly. Considering the scaling of LUT and FF per active eLink, the numbers converge approximately to $\mathcal{O}(10^2)$ LUTs and $\approx 150 - 170$ FFs per active link, especially for configurations with high eLink counts. The consumption of LUTs can be, in general, attributed to each instantiated control path and the flattening network, which concatenates the buffered eLink patterns into the 224 bitvector. The consumption of FFs, on the other hand, is mainly dominated by the per-channel frame buffer, which, although described as a multi-dimensional array, is implemented as registers rather than block RAM.

active eGroups	1 eLink/eGroup		2 eLink/eGroup		4 eLink/eGroup	
	LUT	FF	LUT	FF	LUT	FF
1	278	421	388	563	728	1248
2	392	559	546	839	970	1414
3	511	696	904	1112	1492	1971
4	626	833	864	1392	1729	2521
5	747	969	1107	1665	2247	3072
6	928	1104	1366	1968	2599	3630
7	977	1239	1548	1280	3039	4173

Table 8.2.: Post-synthesis utilization table showing the LUT and FF scaling of the Uplink Frame Collector for different eGroup configurations.

In summary, the Uplink Frame Collector features a parallel architecture in which each eLink channel is processed by an independent control path. The resulting LUT and FF utilization scales approximately linearly with the number of active channels, with only a small additional overhead from global timeout and completion logic. In contrast to the Downlink Frame Assembler, the trade-off between low latency and resource efficiency was chosen in favor of low latency, as the Uplink Frame Collector implements a time-critical path.

8.4.3. System-Level Resource Validation

To validate these scaling considerations at the system level, an upper-bound configuration was implemented in the EMP FPGA reference design. For this purpose, 12 instances of the SC-DMA IP-core were integrated, each configured with 16 enabled uplink and 16 enabled downlink eLink channels. This reflects the settings expected for the maximum link configuration in DCS applications. The correspond-

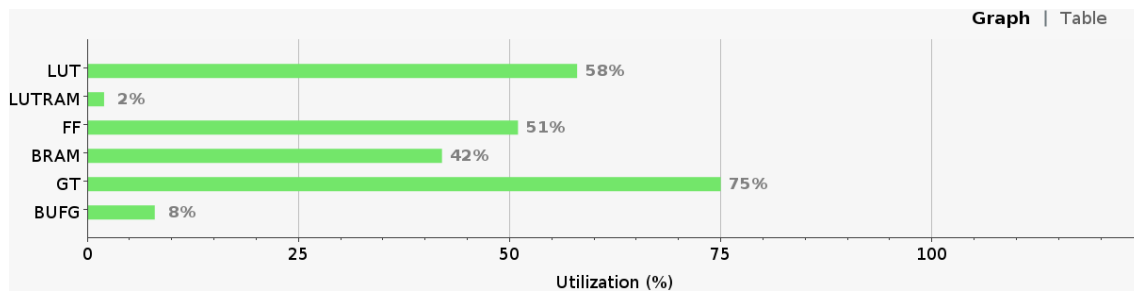


Figure 8.20: Post-implementation resource utilization for the EMP FPGA reference firmware design with 12 instantiated SC-DMA IP-cores.

ing post-implementation utilization, as shown in Figure 8.20, confirms sufficient resource headroom for additional subsystems and future extensions. Moreover, the intended operational scenarios do not foresee more than 10 parallel EMCI links per EMP, thereby providing additional margin compared to the demonstrated configuration. In summary, both the block-level scaling trends and the integrated reference-design results demonstrate that the SC-DMA IP-core scales predictably with the number of active eLinks and remains well within the available FPGA resource budget.

8.5. Kernel Driver

To complement the SC-DMA subsystem, a dedicated Linux kernel driver was developed. The primary objective is to provide a standardized interface to the SC-DMA IP-core, enabling control, configuration, and monitoring through a deterministic interaction model. The driver abstracts the DMA engine's complexity into a compact set of operations, facilitating integration into higher-level software components, such as user-space libraries or applications. To this end, the driver implements a strict transaction model that avoids internal buffering, queues, or command pipelines and instead exposes a thin abstraction layer.

The kernel module is implemented as a platform driver and is bound to the hardware instances via their device-tree identifier (`compatible = "elinkdriver"`). During probing, the driver maps the AXI4-Lite control register interface into kernel space, installs an interrupt handler for DMA completion, and registers a character device that serves as the user space access point.

The following subsections describe the main features of the kernel module, including its character device, `ioctl` interface, interrupt handling, and synchronization

mechanism. The corresponding implementation of the features discussed is provided in Appendix D.

8.5.1. Character Device Interface

Each SC-DMA instance is exposed as a standard Linux character device, providing the primary interface for interacting with the DMA engine. The associated file operations `open()`, `mmap()`, and `release()` define the interactions and ensure that memory allocations, register accesses, and synchronization mechanisms are consistently handled. The following summarizes the specific implementation of the three functions:

- `open()`: When a character device is opened, the driver first allocates a private device node structure that encapsulates per-file state information. Then a DMA-coherent 16 kB buffer is allocated, and its physical base address is written to the `SRC_ADDR` register. Next, the `DST_ADDR` register is programmed with a fixed offset from the physical base address. This divides the contiguous buffer into distinct transmit and receive regions at known offsets, enabling a deterministic, non-overlapping memory layout from the DMA's perspective.
- `mmap()`: The function provides user-space access to the allocated memory regions. When a user-space process calls `mmap()` on the character device, the driver retrieves the corresponding device node structure and calculates the requested mapping size from the Virtual Memory Area (VMA) descriptor. To prevent inconsistent mappings, the requested size is checked against the allocated buffer's size; if they do not match, the call is rejected. Otherwise, the coherent DMA buffer is mapped into the user process's virtual address space. This zero-copy mapping enables user-space applications to write and read eLink frames directly from system memory without additional copying.
- `release()`: When the device is closed, the `elink_release()` tears down all per-open resources, frees the DMA-coherent buffer, and deallocates the device node structure, to ensure that each DMA session cleanly releases its memory and state, preventing resource leaks and guaranteeing that every subsequent session starts from a well-defined context.

8.5.2. IOCTL Interface

The `ioctl()` interface provides the primary control planes between user space and the DMA engine. It exposes a single command that allows applications to trigger DMA transfers and retrieve status information upon completion.

When the `ELINKDRV_DMA_START` is issued, the driver clears the IRQ Handler flag and then reads the `BUSY` register to ensure that the DMA engine is idle and no transaction is currently in progress. If the DMA engine is busy, the command fails and returns `-EBUSY`. Otherwise, the driver triggers the DMA engine by writing to the `START` register, then blocks the calling process until either the interrupt handler signals completion or a 5 s timeout expires. Once the IRQ is processed, the driver reads the `STATUS` register, copies the status to user space, and returns zero to the user-space application, indicating transaction completion. If instead the timeout expires, the command immediately terminates and returns `-ETIMEDOUT`.

8.5.3. Interrupt Handling and Synchronization

The interrupt handler and the `ioctl`-based trigger path are coupled via a completion scheme that uses a shared flag and a dedicated wait queue. Upon DMA initiation, the driver clears the internal IRQ Handler flag, issues the transaction trigger via the `START` register, and then puts the process to sleep on a wait queue. As long as the IRQ Handler remains cleared, the process is blocked for up to a configured timeout, and the transaction is considered in flight.

Once the DMA engine completes the transaction, it asserts the IRQ, and the kernel dispatches the corresponding interrupt service routine. This routine acknowledges the interrupt by writing to the `ACK` register, updates the IRQ Handler flag, and consequently wakes the process sleeping on the device's wait queue.

8.6. Conclusion

In summary, the SC-DMA IP-core meets the central design objectives discussed. It integrates seamlessly into the EMP FPGA reference project, supports the lpGBT eLink frame format for both uplink and downlink, implements a robust error detection, reporting, and fault-recovery mechanism, and significantly reduces software overhead through an efficient driver interface. However, while the functional behavior has been verified and the architectural requirements are met, the system's actual runtime and operational capabilities remain to be quantified.

To evaluate the performance of the SC-DMA system and assess its value to the EMCI-EMP ecosystem, a dedicated benchmark study was conducted. The following chapter presents this study, which aims to determine the system's actual throughput, CPU load, and scalability as the number of concurrent EMCI links increases. Using a methodology similar to that in the benchmark study of the AXI4-Lite-based eLink

Interface IP-core in Chapter 7, the results enable a direct comparison of both eLink interface solutions.

9. Characterization and Benchmarking

The characterization and benchmarking of the SC-DMA subsystem, in conjunction with its integration into the EMCI-EMP ecosystem, is intended to assess the overall capabilities of the novel solution under realistic operational conditions. The primary objective of this study is to quantify the stability of sustained operation, the achievable data throughput, and the resulting CPU load when accessing eLinks through the SC-DMA engine. A central aspect of this study is whether the newly developed DMA solution can reproduce the reliable behavior observed in earlier tests and maintain stable operation across configurations ranging from a single EMCI up to twelve links operating in parallel.

Due to the unavailability of FE hardware during the evaluation phase, the subsequent benchmark campaign leveraged the lpGBT's internal loopback mechanism, introduced in Chapter 7, to replicate the test environment utilized in the initial study. This approach establishes a closed communication path and recreates the characteristics to enable comparison with earlier results.

The chapter provides a comprehensive overview of the specific aspects that distinguish the SC-DMA test environments from the previously studied environments. Subsequently, it provides a comprehensive overview of the firmware and software components involved, details the test methodology, and presents and analyzes the benchmark study results.

9.1. Test Hardware and SC-DMA System Integration

The benchmarking campaign utilizes a test setup analogous to that presented in Chapter 7, comprising a single EMP connected to up to twelve EMCIs. However, it incorporates two adjustments: the integration of the SC-DMA subsystem and the use of extended optical connections. All twelve uplink and downlink channels are routed through 100 m optical fibers, providing link conditions that more accurately reflect those expected during operation.

The deployed firmware is derived from the EMP FPGA reference project. For each EMP-EMCI pipeline, the previously used eLink-Interface IP-core was replaced with

an instance of the SC-DMA IP-core. The corresponding uplink and downlink eGroup configuration of each DMA channel was defined as follows:

- Downlink: Each downlink eGroup was configured to use a single eLink, corresponding to the full 8 bit width of the respective group.
- Uplink: Each enabled uplink eGroup must be configured to operate with four eLinks, such that each uses 8 bit of the 32 bit-wide uplink group.

This configuration was selected so that, in lpGBT loopback mode, the EMCI loops back the 32 bit downlink frame, comprising four groups, each containing a single 8 bit part of an eLink frame, to all seven uplink eGroups. Consequently, each uplink group comprises a 32 bit word consisting of four unique 8 bit frame patterns, thereby creating a closed data path. This guarantees that the lpGBT uplink frame contains detectable and verifiable data that are recognized as an eLink frame by the Uplink Frame Collector and can be easily checked against the transmitted data. Conversely, the number of active uplink eGroups remains configurable because the DMA engine bypasses data from these eGroups, resulting in a reduced firmware resource footprint, shorter AXI write bursts, and reduced software data validation and generation. During the benchmarking process, two configurations were evaluated. The first configuration comprised four active eGroups, corresponding to 16 active uplinks, the maximum expected in DCS applications, where each downlink is associated with a corresponding uplink channel. The second configuration comprised seven active eGroups, which corresponds to the system's maximum achievable uplink bandwidth.

The integration of the SC-DMA instances with the uplink and downlink eLink data paths is implemented via the eLink Stream interfaces of the emp-lpgbt IP-core (see Figure 6.7). Concurrently, the integration with the Zynq UltraScale+ MPSoC memory controller is implemented via four Crossconnect IP-cores, each connecting 6 of the 24 AXI masters (12 AXI4-Read and 12 AXI4-Write interfaces) to four AXI FPD HP interfaces. Notably, the integration does not modify the baseline design's architectural structure. Consequently, it guarantees that any behavioral inconsistencies detected can be attributed to the integration of the SC-DMA subsystem.

Regarding the software, the test environment operates within the established EMCI-EMP ecosystem. AlmaLinux 9 serves as the runtime platform, and higher-level control and monitoring tasks continue to rely on standard ecosystem libraries, such as LpGbtSw and empTools. In contrast to the previous benchmark campaign, which used the generic UIO subsystem to implement eLink access, the study presented uses the custom-developed DMA kernel module. The combination of these components forms a controlled and reproducible evaluation environment that remains directly

comparable to the initial study while incorporating the new SC-DMA infrastructure.

9.2. Test Application

The developed benchmark application is a C program that can be instantiated in multiple concurrent processes to exercise the SC-DMA subsystem under configurable load conditions. The primary objective is to generate pseudo-random traffic for each DMA channel, write the generated data to the mapped memory, trigger the DMA engine, and verify end-to-end data integrity upon receiving the data back. Furthermore, the C program records detailed performance metrics, including throughput, CPU load, and scheduler behavior, for subsequent analysis of the tests. Figure 9.1 provides a general overview of the interactions among the user-space application, the Linux kernel driver, the SC-DMA IP-core, system memory, and the interrupt controller.

Upon initiation, the program parses the command-line arguments to determine the number of active DMA channels and an optional inter-transaction delay. For each active EMCI-EMP link, the program then forks a child process that drives exactly one SC-DMA instance. The parent process synchronizes the concurrent initiation of all child processes by implementing a simple pipe-based synchronization barrier. After this, the parent process sleeps while waiting for the child processes to complete.

Each child process is initially bound to a specific CPU core, thereby ensuring a round-robin distribution of processes across all available cores. This prevents process migration between cores, improves cache locality, and enables a clear separation of the executed processes, allowing a precise analysis of their behavior. Furthermore, each worker is scheduled using a real-time round-robin policy, which constrains the scheduler's decisions, leading to more predictable time slots and reduced jitter, especially when multiple processes share a core or compete with other system tasks. Subsequently, each process opens its corresponding SC-DMA character device, memory-maps a 16 kB coherent buffer exported by the kernel driver, and enters the main execution loop. Within that loop, each child process generates four independent 128 bit pseudo-random downlink frames. These frames contain a fixed start and end byte, enabling the SC-DMA IP-core to detect frame boundaries and a 14 bit payload. Note that the transmitted payload pattern will repeat roughly every 300s due to the periodicity of the chosen random number generator. The generated frames are then written into the TX region of the mapped buffer and, in parallel, buffered in a local array for subsequent verification. Thereafter, the process invokes the SC-DMA IP-core and awaits its completion. Upon transaction completion, the

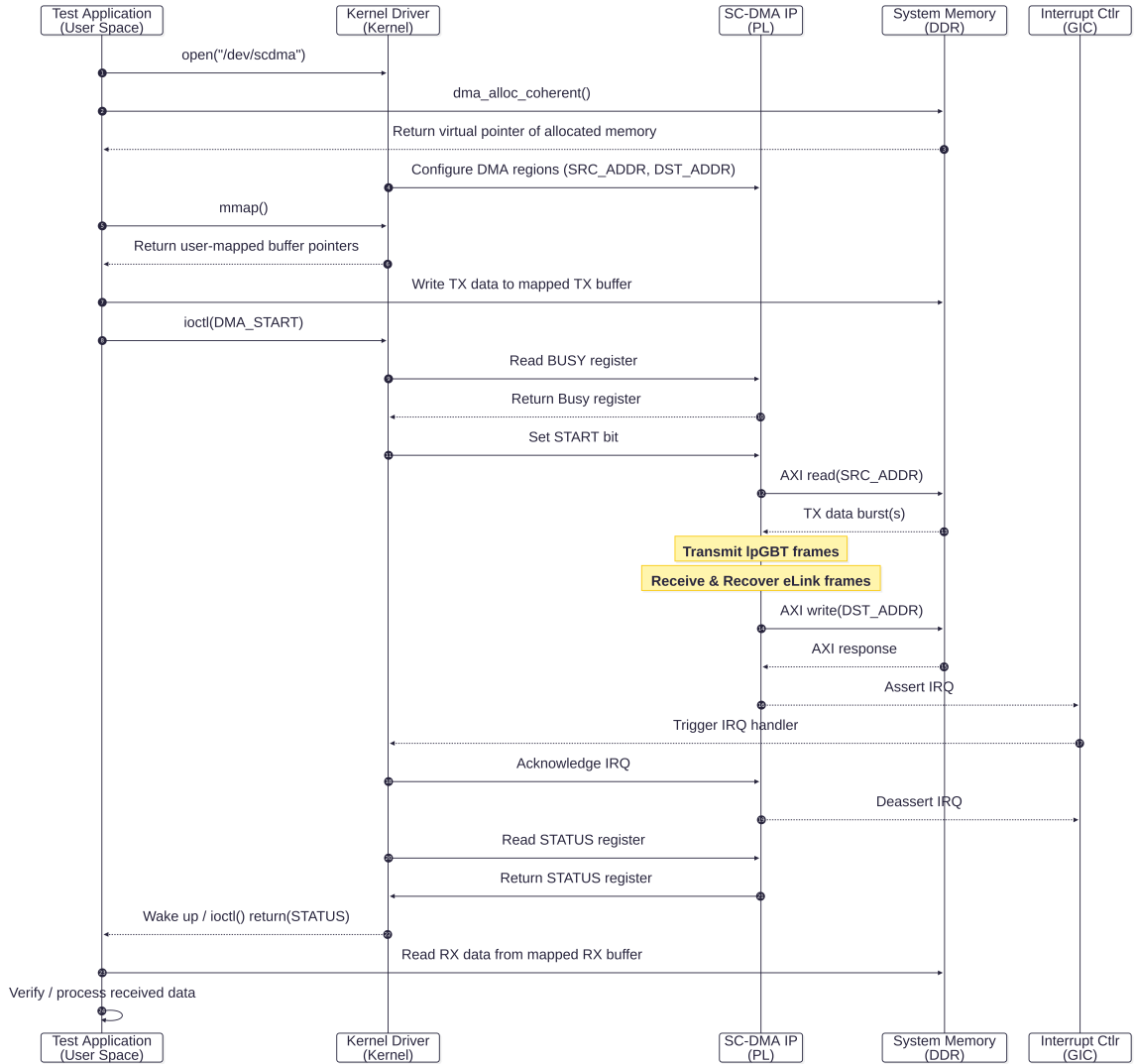


Figure 9.1: Sequence diagram showing the interaction between the user-space test application, the kernel driver, the SC-DMA IP-core in the PL, the system memory, and the interrupt controller during a single DMA transaction.

status descriptor returned by the DMA engine and the received uplink frames are validated, and the corresponding counters are incremented. The program then transitions to the inter-transaction delay phase, after which the main loop is executed repeatedly until the programmed test duration times out.

9.3. Benchmark Results

The benchmark campaign was executed using the application described in the preceding section. To automate the measurement campaign, a dedicated Python script was implemented that systematically traverses all system configurations under study. The script processes the configurations (1, 2, 3, 4, 8, and 12 active EMCI-EMP links) and, for each configuration, repeatedly launches the benchmark applications with progressively shorter inter-transaction delays from 1s to 0s in logarithmic steps. Each of these steps was executed for 10 minutes, during which the relevant performance metrics were systematically documented to a CSV file at 1-minute intervals.

The results of the benchmarking campaign are presented in the following sections. As in the preceding study, the analysis is divided into three sections. Initially, measurements obtained with a programmed inter-transaction delay are examined to provide insight into how the DMA scales under increasing load and increased uplink count. Subsequently, the datasets recorded without any inter-transaction delay are analyzed, which indicate the system's maximum sustainable data rate. Finally, the result of a 167-hour stability test is provided, in which the system's stability was assessed under maximum load with 12 parallel EMP-EMCI links.

9.3.1. Programmed Delay

The benchmark campaign was conducted for two distinct uplink configurations. In the initial configuration, each DMA core was set to operate with 16 active uplink eLinks, corresponding to four active eGroups with each four eLinks enabled per group (see Table 5.3). This reflects the maximum configuration foreseen for DCS operation. In the second configuration, the number of active uplink eLinks was increased to 28 active uplink eLinks, corresponding to seven active eGroups with four eLinks enabled per group, to study how DMA performance and CPU load scale with the amount of data transported per transaction and to exercise the full bandwidth of the system. The following discussion first presents results from configurations with 16-uplink eLinks and then compares them with measurements obtained with 28-uplink eLinks.

Configuration with 16-uplinks per EMCI

The primary set of results examines the correlation between CPU utilization and the programmed inter-transaction delay. As shown in Figure 9.2, the average CPU load per core is shown as a function of the inter-transaction delay across all evaluated

system configurations. It is shown that the CPU load increases monotonically as the transaction delay decreases. When the delay is reduced below approximately $100\ \mu\text{s}$, the CPU load increases, reflecting the increased time spent in user-space processing and kernel-level DMA handling.

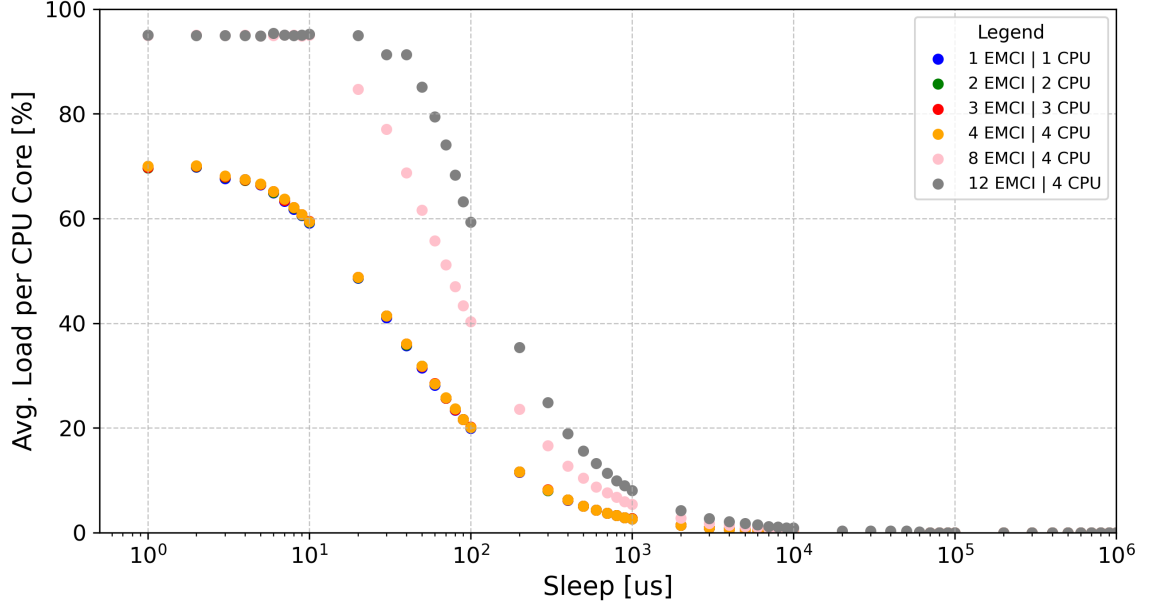


Figure 9.2: Average CPU load per core as a function of the inter-transaction delay of different system configurations. The markers denoting configurations with 1, 2, 3, and 4 active EMCIs largely overlap, leaving only the yellow markers visible.

In the configuration where the number of active EMCIs is balanced to the number of used CPU cores (1, 2, 3, or 4 active EMCIs), the corresponding markers in Figure 9.2 lie almost exactly on top of one another, so predominantly the 4 EMCI/4 core markers are visible. This overlap shows that, for a given delay, the average per-core utilization across these four configurations differs by at most 1%. Such behavior is expected since each EMP-EMCI link is handled by a dedicated, but identical process pinned to its own core. At the shortest delay of $1\ \mu\text{s}$, the per-core load in these balanced configurations peaks at around 70%.

For the configuration with 8 and 12 active EMCIs, the average load per core increases more steeply, as the delay is reduced. In these cases, multiple DMA processes share each CPU core, so the total amount of work per core increases roughly in proportion to the number of active EMCIs. Once the inter-transaction delay drops below a few tens of microseconds, this combined processing nearly saturates all CPU cores ($\approx 95\%$ to 100%).

The corresponding throughput is summarized in Figure 9.3, which shows the average per-EMCI throughput as a function of total CPU utilization. A high degree of similarity is exhibited in balanced link/core configurations, for which each of these four configurations achieves a maximum per-link throughput of approximately 60 Mbit/s at a total per-core utilization of approximately 70 %. This verifies that each EMCI-EMP can sustain a stable, reproducible throughput when granted exclusive access to a CPU core. Furthermore, it shows that the SC-DMA subsystem does not introduce any systematic performance skew between multiple parallel links.

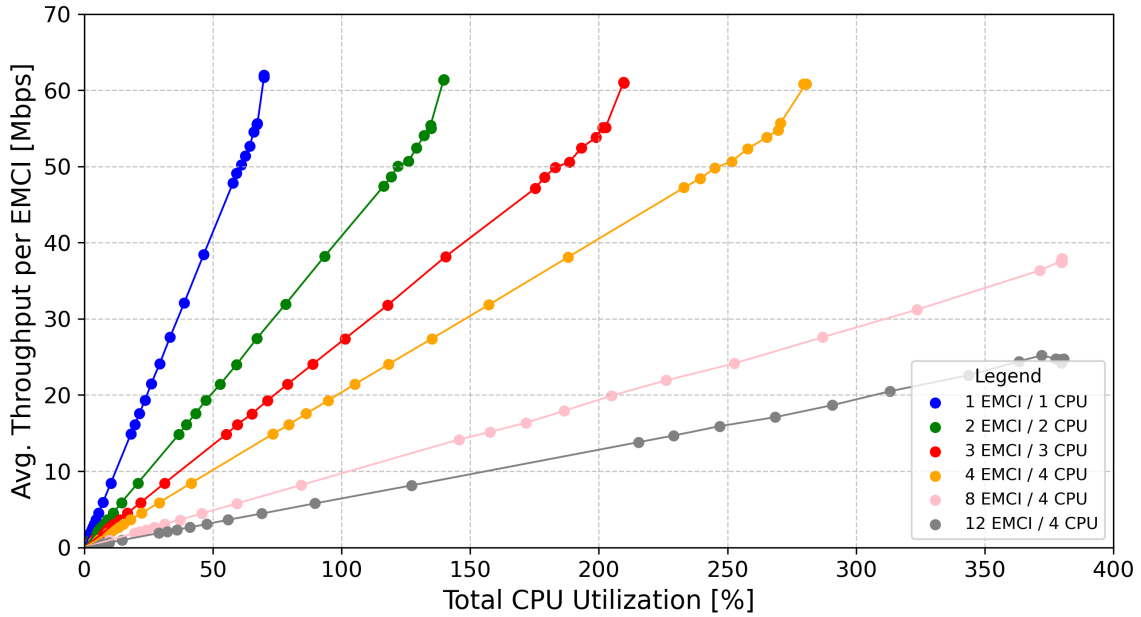


Figure 9.3: Average throughput per EMCI as a function of total CPU utilization.

In contrast to exclusive core-link assignment, configurations with 8 and 12 active EMCIs exhibited a significant decrease in per-link throughput. In these cases, the number of active links exceeds the number of available CPU cores, so each core must service either 2 (8 links) or 3 (12 links) DMA channels in parallel. Consequently, the available CPU time per link decreases, leading to lower per-link throughput.

The scaling of the total system throughput is illustrated in Figure 9.4, which shows the sum of all per-link throughputs as a function of total CPU utilization. Across all tested configurations, the data points lie close to a single straight line, indicating an almost perfectly linear relation between CPU load and aggregate throughput. At balanced link/core ratios, each EMCI-EMP link effectively operates as an independent copy, resulting in both linear scaling of total throughput and CPU utilization.

For higher channel counts, the total throughput continues to follow the linear trend as the CPU usage approaches 400 %, at which point all four cores are fully occupied. At this juncture, all links must share the fixed CPU budget and the total throughput saturates accordingly. Overall, the nearly perfect linear behavior across a wide range of configurations demonstrates excellent scaling of the whole platform and the firmware software stack.

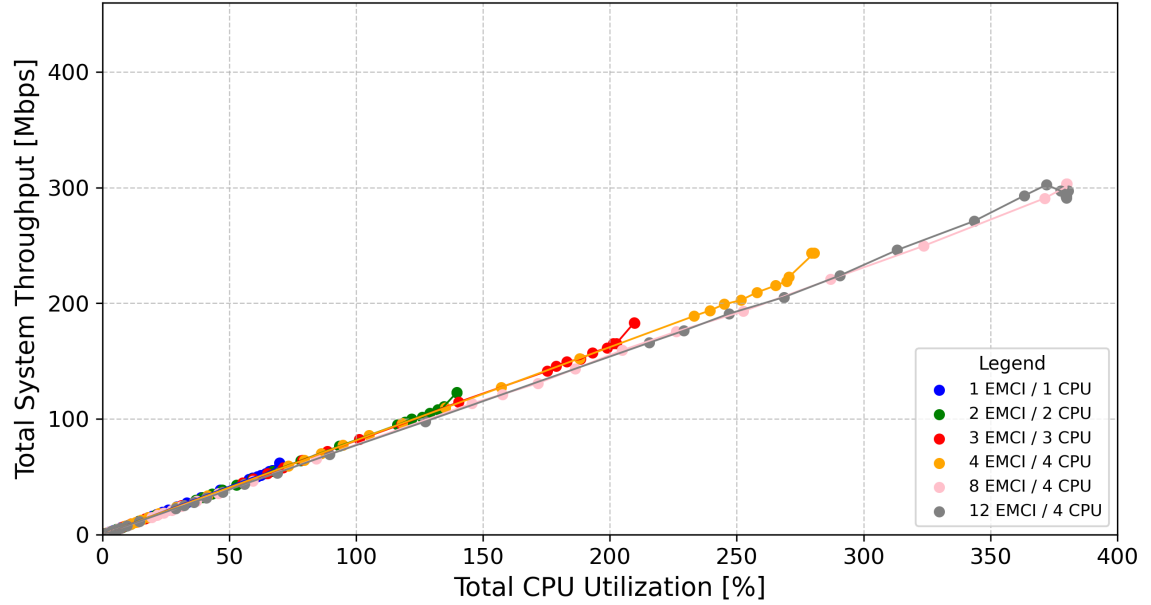


Figure 9.4: Total system throughput as a function of total CPU utilization.

The slight deviations from the global linear trend at the highest throughputs at balanced link/core configurations, visible in both Figures 9.3 and 9.4, occur for the smallest inter-transaction delays. Here, the requested inter-transaction delay approaches the time granularity of the OS scheduler and its timers, and the OS cannot accurately measure the delay for each process, thereby introducing a minor timing inaccuracy for each DMA channel. While in the 1 EMCI/1 core configuration, this effect is barely visible, it increases with the number of channels operating in parallel, reaching a pronounced level in the 4 EMCI/4 core configuration. However, in configurations with 8 and 12 parallel channels, the effect is largely suppressed because the CPU cores approach saturation at inter-transaction delays of around 10 μ s. Consequently, the effective timing between the processes is dominated by CPU saturation, context-switch overhead, and wake-up latencies rather than by the nominal sleep values. The observed deviations from ideal linear scaling should therefore be attributed to the timing granularity, rather than to any intrinsic misbehavior of the SC-DMA subsystem.

Impact of increased uplink count

To investigate how performance scales with the amount of data transported per uplink transaction, the benchmark campaign was repeated with 28 active upstream eLinks per EMCI. As illustrated in Figure 9.5, the average throughput per EMCI is shown as a function of total CPU utilization for the 28-uplink configurations. The data demonstrate a high degree of similarity to the 16-uplink data set, as shown in Figure 9.2: the throughput increases approximately linearly with CPU utilization until the corresponding cores reach approximately 70 %. However, the 28-uplink configurations achieve considerably higher per-link throughput, of roughly 90 Mbit/s.

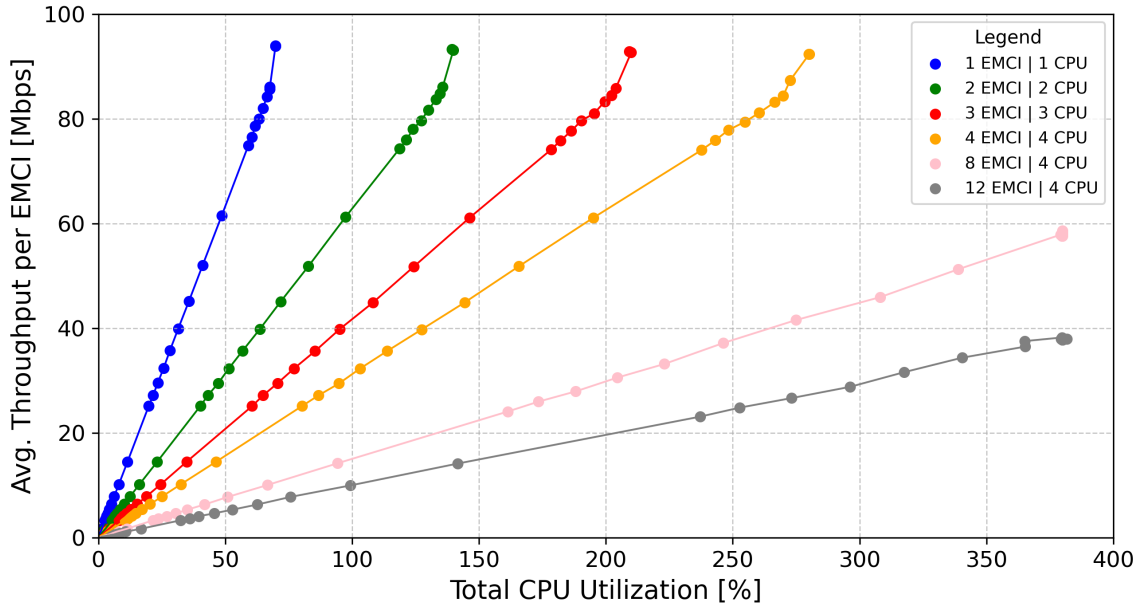


Figure 9.5: Average throughput per EMCI as a function of total CPU utilization in a configuration in the 28-uplink configuration.

This similarity in CPU behavior between the two configurations is further highlighted in Figure 9.6, which plots the total CPU utilization as a function of the programmed delay for both 16 and 28-uplinks. For all evaluated system configurations, the curves for both uplink configurations differ only by 1 % to 5 % at a given delay and thus follow almost identical trajectories. This suggests that, to first order, the CPU load is dominated by the rate of DMA transactions and interrupt events rather than by the exact payload size of each transfer. Consequently, an increase in the number of active uplinks results in a largely unaltered relation between delay and CPU utilization.

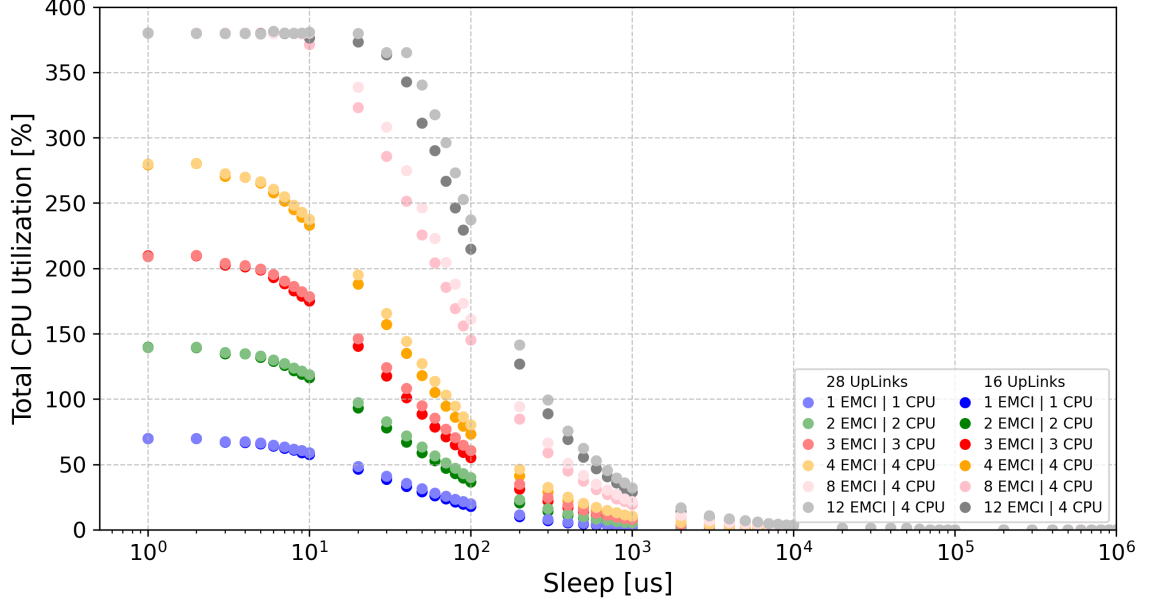


Figure 9.6: Total CPU utilization as a function of the programmed inter-transaction delay comparing the 16- and 28-uplink configuration.

A closer look at the average transaction rate per EMCI in Figure 9.7 reveals, however, that the 28-uplink configuration achieves a lower rate than the 16-uplink configuration at the same total CPU utilization. This indicates that each DMA transaction in the 28-uplink case consumes more CPU time, resulting in fewer transactions per second being issued for a given CPU budget.

As illustrated in Figure 9.8, the CPU-load skew between user and kernel space quantifies the difference in average CPU utilization attributed to user and system time and is defined as:

$$\Delta u_{\text{CPU}} = \frac{t_{\text{usr}} - t_{\text{ker}}}{t_{\text{usr}} + t_{\text{ker}}}, \quad (9.1)$$

where t_{usr} t_{ker} denote the user- and kernel-space CPU times, respectively, accumulated for each sampling point. The resulting positive skew Δu_{CPU} indicates that user-space processing dominates. For small inter-transaction delays, the skew is systematically higher in the 28-uplink configuration, indicating a larger fraction of time spent in user space than in the 16-uplink case. This is consistent with the increased amount of eLink data that must be generated and validated per transaction. In addition, Figure 9.5 shows that the effective throughput per EMCI is higher for 28-uplinks at any given point. The combination of these observations indicates that in this configuration, the system performs fewer DMA transactions per second, but each transaction transports more data. Hence, an increase in throughput at

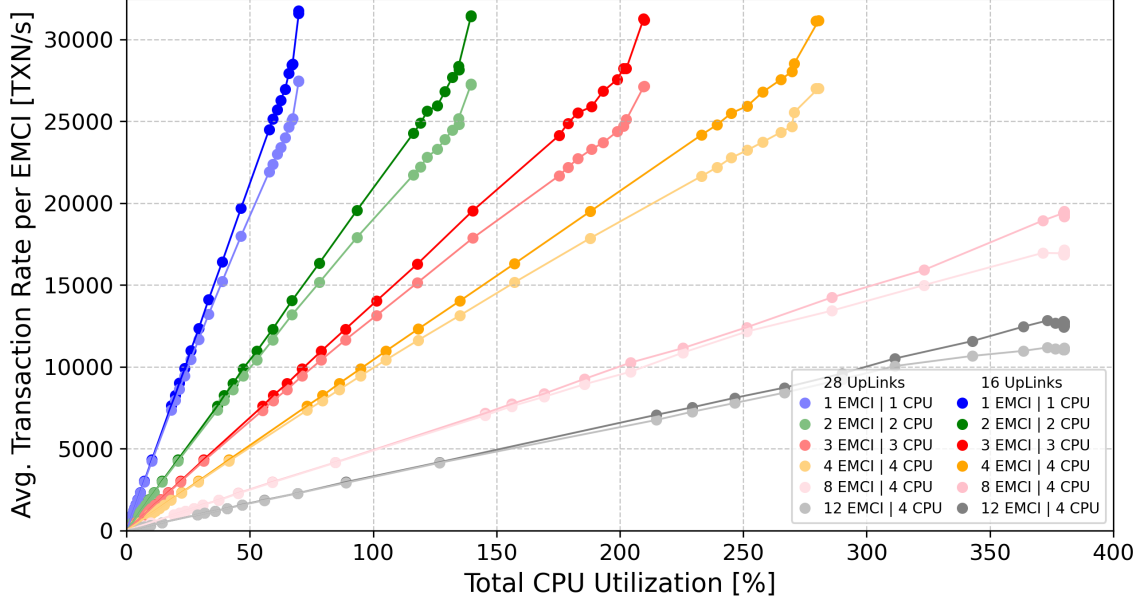


Figure 9.7: Average transaction rate per EMCI as a function of total CPU utilization for the 16- and 28-uplink configurations.

a constant CPU load is achieved by increasing the useful payload per transaction rather than increasing the transaction rate.

A dedicated test of the 28-uplink and 12 active EMCIs configuration and a programmed 1 μ s inter-transaction delay, but without data validation, yields an aggregated throughput of approximately 540 Mbit/s. For comparison, the corresponding configuration with data validation enabled achieves about 460 Mbit/s, i.e., a throughput increase of roughly 17%. The CPU load nevertheless remains dominated by user space, indicating that the dominant contribution originates from the test application rather than from the DMA engine. This suggests that the SC-DMA subsystem itself is not the limiting factor.

In summary, the comparison between the 16- and 28-uplink configurations demonstrates that the SC-DMA subsystem exhibits near-ideal scaling with the amount of data transported per transaction. The CPU cost per DMA transaction is mainly independent of the uplink configuration; instead, the predominant factor is the benchmark application and its data-generation and validation mechanisms. However, increasing the number of active uplinks directly corresponds to a proportional increase in effective throughput, while preserving the linear relation between total system throughput and CPU utilization, as observed for the 16-uplink configuration.

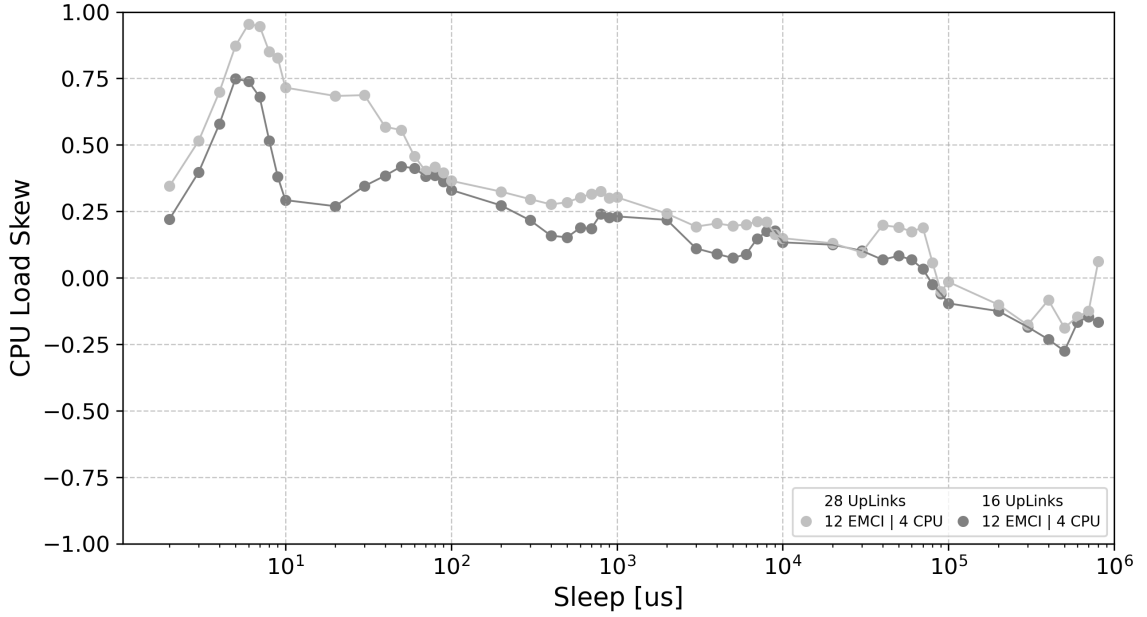


Figure 9.8: CPU load skew between user and kernel space as a function of the programmed inter-transaction delay for the configuration with 12 active EMCI-EMP links, comparing the 16- and 28-uplink configuration.

9.3.2. Zero Inter-Transaction Delay

The ensuing analysis will consider the special case in which the benchmark application was operated without any inter-transaction delay. In this configuration, the application omits the sleep statement entirely, enabling each worker process to issue DMA transactions back-to-back, constrained solely by the CPU and OS. This mode does not represent typical applications; however, it does provide an upper bound on the maximum sustainable throughput and highlights the system's behavior in a fully CPU-bound regime.

As illustrated in Figure 9.9, the maximum average throughput per EMCI across all tested system configurations is summarized, showing the results for 16 and 28 active uplinks. All configurations achieve significantly higher per-link data rates than the programmed-delay regime discussed in the previous subsection, since no CPU time is spent waiting between transactions. For link/core balanced configurations with 1, 2, 3, and 4 active EMCI-EMP links, the 16-uplink configurations achieve per-link throughputs of approximately 85 Mbit/s, and the 28-uplink configuration reaches per-link throughputs of up to 120 Mbit/s.

The corresponding total system throughput is illustrated in Figure 9.10. In the 28-uplink configuration, the aggregated throughput increases nearly linearly from

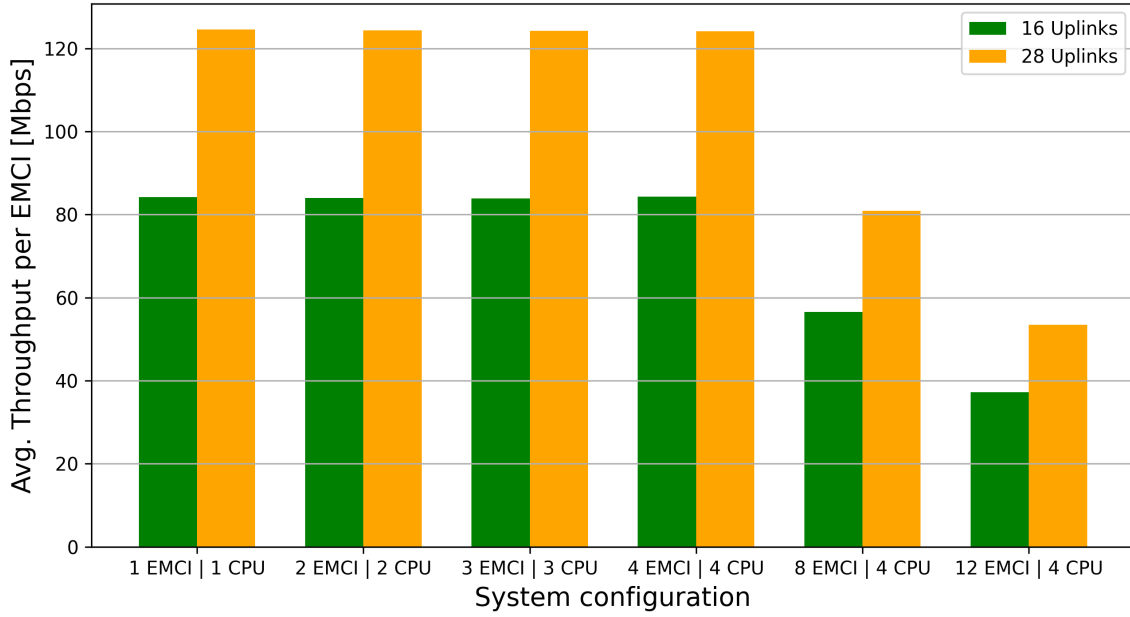


Figure 9.9: Maximum average throughput per EMCI for all evaluated system configurations, comparing the 16- and 28-uplink configurations in the zero-delay regime.

approximately 125 Mbit/s in the 1 EMCI/1 CPU setup to almost 500 Mbit/s in the 4 EMCI/4 CPU configuration. When 8 or 12 EMCIs are operated in parallel, the total throughput saturates at approximately 650 Mbit/s. The same qualitative behavior is observed for the 16-uplink configuration, but with lower absolute values.

In summary, the zero-delay measurements show that removing the inter-transaction delay increases both the per-link and the aggregated throughput. In the balanced configurations, the per-link throughput increases to about 85 Mbit/s for the 16 uplink and up to 120 Mbit/s for the 28 uplink configuration. When 8 or 12 EMCIs are operated in parallel, the total throughput saturates at around 650 Mbit/s or 450 Mbit/s respectively, with a corresponding reduction in per-link rate as multiple channels share the same CPU cores. This behavior indicates that the limiting factor is the available CPU resource, rather than any intrinsic bandwidth limitation of the SC-DMA subsystem. A dedicated test with data validation disabled demonstrates that, for the 28-uplink configuration, the system can achieve an aggregate throughput exceeding 930 Mbit/s, corresponding to a performance increase of more than 40 %.

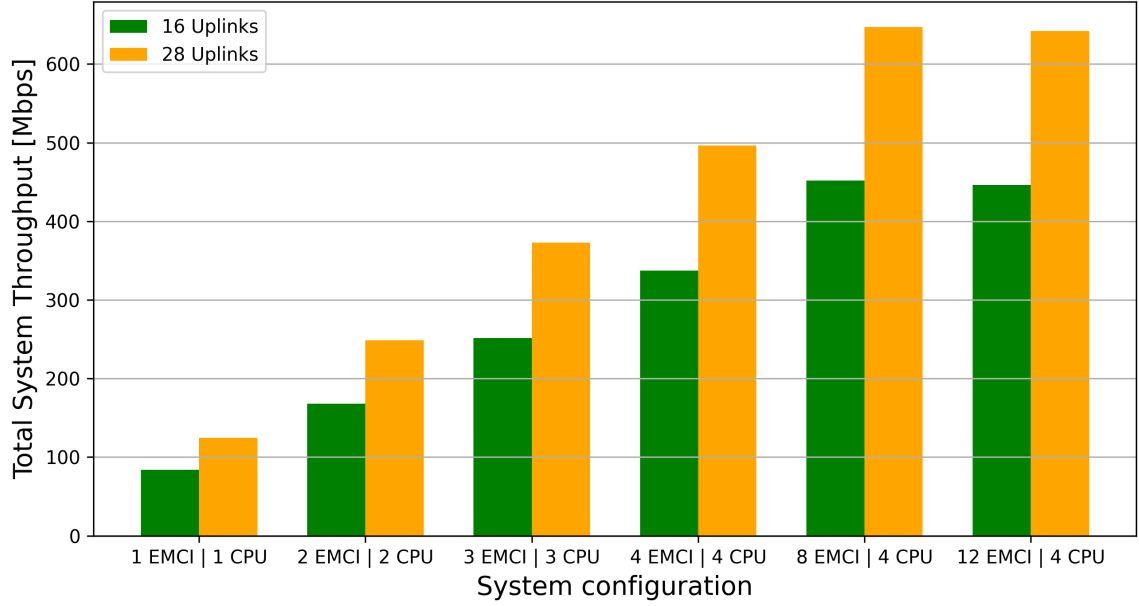


Figure 9.10: Maximum total system throughput for all evaluated system configurations in the zero-delay regime, comparing the 16-and 28-uplink configurations.

9.3.3. Long-Term Stability Test

To complement the benchmark study, a dedicated long-term stability test was conducted. The objective of this test was to verify that the SC-DMA subsystem can sustain a high data rate for extended periods without performance degradation or instability. To this end, the 16-uplink configuration with 12 active EMCI-EMP links was operated continuously for more than 167 h. The benchmark application was configured without inter-transaction delays, such that each worker process issued DMA transactions back-to-back. Figure 9.11 summarizes the evolution of the transmitted data over time. It shows the cumulative amount of received data, which increases linearly over the measurement period and reaches several hundred Tbit.

The linear increase in transmitted data indicates that the DMA engine, kernel driver, and user-space benchmark application maintain stable operation over extended periods. During the test, no performance degradation, anomalous behavior, or transmission error was observed, indicating that the SC-DMA subsystem is suitable for continuous high-throughput operation.

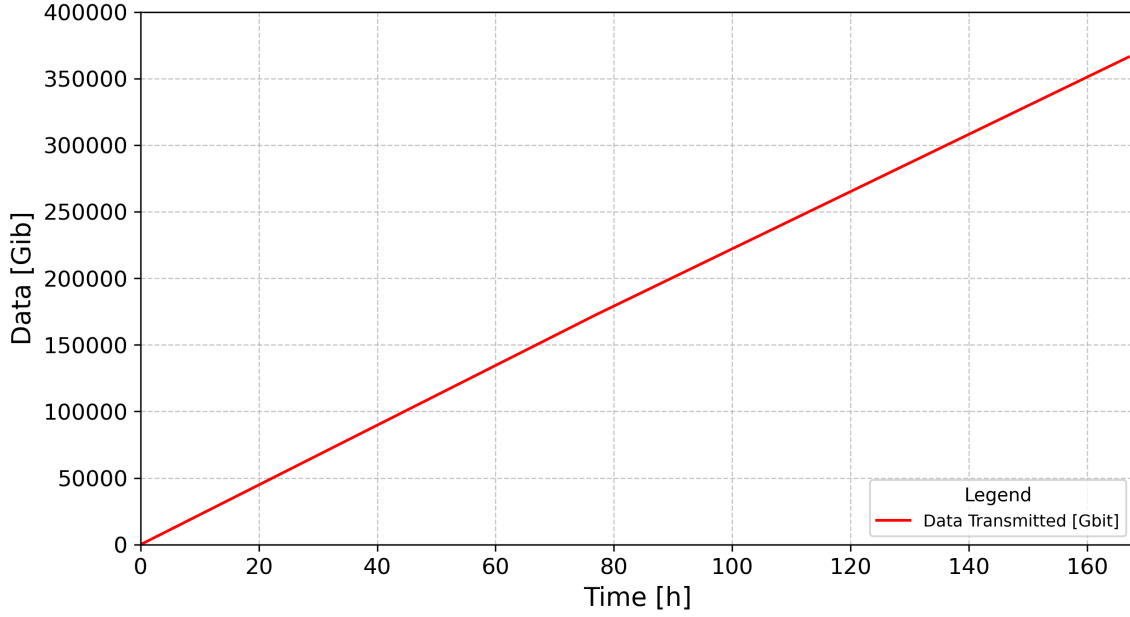


Figure 9.11: Long-term stability test of the SC-DMA subsystem: cumulative data transmitted as a function of time over a period of more than 167 h.

9.4. Conclusion

The benchmark study presented in this chapter demonstrates that the SC-DMA subsystem provides a scalable and CPU-efficient data path across a wide range of EMP-EMCI configurations. For programmed inter-transaction delays, the system exhibits predictable, CPU-bound behavior: CPU utilization increases monotonically with decreasing delay, and total throughput scales linearly with the number of active links and the available CPU budget. For cases where the number of links corresponds to the number of CPU cores, each link achieves a stable per-link throughput. Consequently, the SC-DMA implementation does not induce link-to-link performance variations.

A comparison of configurations with 16 and 28 active uplinks confirms that the dominant cost per DMA transaction is primarily independent of the payload size. The increase in uplinks results in more valid data transported per transaction, while the overall number of transactions decreases slightly. Thus, a higher effective throughput is achieved at a similar CPU load. An analysis of the zero-delay regime revealed that both the 16- and 28-uplink configurations become fully CPU-bound, and the maximum achievable throughput is limited by available processing resources rather than the DMA engine.

The long-term stability test demonstrated that the SC-DMA subsystem can sustain a high throughput over more than 167 h without degradation or errors. Overall, all results confirm that the proposed SC-DMA architecture meets the targeted performance, robustness, and scalability requirements, demonstrating its potential as a valuable addition to the EMCI-EMP ecosystem.

10. Summary and Conclusions

The development of the EMCI-EMP system represents a substantial improvement for the ATLAS Detector Control System (DCS) addressing the requirements of the High-Luminosity Large Hadron Collider (HL-LHC) era. The newly developed system provides a flexible and radiation-tolerant architecture for a wide range of detector control and monitoring tasks. Its generic design enables adoption across diverse Front-End (FE) devices and offers a standardized interface to the distributed DCS Back-End (BE). In the HL-LHC environment, on-detector FE electronics are deliberately kept simple to meet radiation and reliability constraints, thereby relocating control and monitoring responsibilities to the BE. This, in turn, demands higher-bandwidth communication links and more processing-intensive services, which the EMCI-EMP system is designed to provide.

The Embedded Monitoring and Control Interface (EMCI) and the Embedded Monitoring Processor (EMP) form a complementary hardware system. The EMCI, implemented as a radiation-tolerant FE interface, is based on CERN-developed components such as the low-power GigaBit Transceiver (lpGBT) ASIC, which aggregates all FE signals into a single bidirectional channel and interfaces with the Versatile Transceiver+ (VTRx+), an optical transceiver to transmit data through a high-speed optical link. It supports up to 16 downlink and 28 uplink devices via generic electrical Links (eLinks), permitting flexible protocol definitions and connection topologies tailored to the specific FE design. The EMP, in turn, is a compact platform that serves as the backbone of the DCS control and monitoring system. It is designed around a Trenz Electronics System-on-Module (SoM) hosting a highly flexible AMD Xilinx Zynq Multiprocessing System-on-Chip (MPSoC). It can connect to up to 12 EMCIs in star topology through optical links and interfaces with the Supervisory Control and Data Acquisition (SCADA) system over a Local Area Network (LAN). Together, these platforms provide the hardware foundation for a scalable and forward-compatible DCS infrastructure.

Complementing the hardware, a comprehensive firmware and software ecosystem has been developed. The EMP FPGA reference design establishes a standardized baseline architecture that combines reusable infrastructure and Intellectual Property Cores (IP-cores). It implements the BE counterpart to the lpGBT and provides a complete control and monitoring path for the ASIC. The firmware is complemented by a software suite that includes the EMP Operating System (epos), dedicated hardware abstraction layers, and a set of quasar-based OPC UA servers. Together, these components provide a runtime environment and enable control and monitoring of EMP and EMCI hardware, as well as seamless integration with the DCS BE. In

summary, the ecosystem enables developers to access a wide range of production-ready components, ensuring consistency and ease of deployment for integration with their FE electronics.

A central aspect of the thesis is the design and implementation of the Slow-Control Direct Memory Access (SC-DMA) subsystem, which provides a high-throughput, CPU-efficient data path between the EMP system memory and the EMCI's eLink channels. In contrast to commercial Direct Memory Access (DMA) engines, which are typically optimized for long, contiguous data streams, the SC-DMA subsystem is tailored to large numbers of small, frame-based control and monitoring transactions. The protocol translation between linear eLink frames in system memory and the complex, interleaved frame structure of the lpGBT is facilitated through serializing and deserializing eLink frames in hardware. Consequently, the SC-DMA effectively accelerates the protocol framing and unpacking that would otherwise have to be implemented in software or additional logic. In addition, the IP-core implements a configuration register interface that enables deterministic software control of transactions, complemented by the Watchdog unit, which ensures safe and coherent operation, reports errors, and recovers the engine if required. This hardware-accelerated protocol framing constitutes a novel application of DMA principles, especially in the DCS domain. Compared with the register-based AXI4-Lite approach, the SC-DMA IP-core offers a frame-oriented eLink interface that supports variable per-link payload sizes, as well as comprehensive transaction-level status reporting and watchdog-assisted recovery, enabling robust error detection and handling.

On the software side, the SC-DMA IP-core is complemented by a custom Linux kernel module that encapsulates the low-level control and access to each instantiated core behind a dedicated character device. The driver handles kernel-side memory mapping, manages source and destination address descriptors, and processes Interrupt Requests (IRQs). Through the character device interface, user applications can access a SC-DMA instance, map the associated memory region into user space, and initiate DMA transactions without interacting directly with the underlying hardware.

During an extensive benchmark campaign with up to 12 concurrent DMA channels communicating with 12 EMCIs, the SC-DMA subsystem was evaluated for stability, performance, and scalability. The results demonstrate that the developed solution sustains high transaction and throughput rates over extended periods without transmission errors, while significantly reducing CPU load compared to register-map-based interaction between the Processing System (PS) and Programmable Logic (PL) using the Linux Userspace I/O (UIO) subsystem. The benchmarking further shows that the architecture scales almost perfectly linearly with the number of parallel DMA channels and that the interrupt handling and watchdog mechanism provide

robust operation even under high-load and marginal-timing conditions. At the same time, the study reveals practical limitations of the current validation model: per-frame verification in user space accounts for a non-negligible fraction of the overall CPU load and saturates available processing resources before the underlying SC-DMA subsystem reaches its intrinsic throughput limitations.

Following the successful completion of an ATLAS-internal review process, the EMCI-EMP project will move into production. A total of 54 EMP and 165 EMCI will be manufactured, assembled, and tested before being handed over to the ATLAS subsystems. During this phase, the role of the EMP team will gradually shift from system development towards support. The team will continue to maintain and develop the EMCI-EMP ecosystem, while the individual subsystems will integrate the system with their respective FE electronics. Nevertheless, several directions for further development of the EMCI-EMP ecosystem have already been identified.

On the firmware side, both the FW-Debughub and the SC-DMA IP-core offer potential for refinement. Regarding the FW-Debughub, extended firmware diagnostics could be implemented, and the SC-DMA IP-core could be extended to include subsystem-specific features. On the software side, future work will focus on the OPC UA server, extension of the SC-DMA driver interface, and the integration of all components into automated build and test flows.

Overall, the work presented in this thesis establishes the EMCI-EMP DCS interface as a solid and extensible basis for a broad range of detector control and monitoring tasks during the HL-LHC era. The combination of a radiation-tolerant, lpGBT-based FE and a versatile, MPSoC-based BE, together with a coherent firmware and software stack, provides a unified platform that can be adapted to the specific needs of diverse on-detector FE electronics. Within this ecosystem, the SC-DMA subsystem addresses the performance limitations between the PS and the PL by establishing a dedicated, CPU-efficient data path that enables direct eLink access through system memory. Additionally, the SC-DMA subsystem serves as a hardware accelerator for the lpGBT protocol framing, presenting complex frame-based traffic as a simple memory abstraction to software. The validation campaign demonstrated the efficacy of the SC-DMA subsystem by showing that it can process high transaction rates over an extended period and scale efficiently to a large number of concurrent DMA channels. Consequently, it can be concluded that the SC-DMA subsystem is well-suited to function as a key building block within the EMCI-EMP ecosystem.

In the longer term, the EMCI-EMP system and the SC-DMA subsystem provide a solid foundation for a reusable DCS platform beyond the specific use cases addressed in this work. Their modular design and comprehensive ecosystem make it an attractive candidate for future adoption by additional ATLAS subsystems and,

potentially, by other experiments with similar requirements for radiation tolerance, scalability, and maintainability.

Bibliography

- [1] Oliver Brüning et al. ‘70 years at the high-energy frontier with the CERN accelerator complex’. In: *Nature Rev. Phys.* 6.10 (2024), pp. 628–637. DOI: 10.1038/s42254-024-00758-5. URL: <https://cds.cern.ch/record/2917069>.
- [2] Lyndon R Evans and Philip Bryant. ‘LHC Machine’. In: *JINST* 3 (2008). This report is an abridged version of the LHC Design Report (CERN-2004-003), S08001. DOI: 10.1088/1748-0221/3/08/S08001. URL: <https://cds.cern.ch/record/1129806>.
- [3] Ewa Lopienska. ‘The CERN accelerator complex, layout in 2022’. In: (2022). General Photo. URL: <https://cds.cern.ch/record/2800984>.
- [4] Stephan Russenschuck. ‘Electromagnetic design of superconducting accelerator magnets’. In: (2004). DOI: 10.5170/CERN-2004-008.71. URL: <https://cds.cern.ch/record/808348>.
- [5] Frank Gerigk. ‘Cavity types’. In: (2011). Comments: presented at the CERN Accelerator School CAS 2010: RF for accelerators, Ebeltoft, 8-17 June 2010; CERN-2011-007, pp. 277-298. DOI: 10.5170/CERN-2011-007.277. arXiv: 1111.4897. URL: <https://cds.cern.ch/record/1401088>.
- [6] G Aad et al. ‘The ATLAS Experiment at the CERN Large Hadron Collider’. In: *JINST* 3 (2008). Also published by CERN Geneva in 2010, S08003. DOI: 10.1088/1748-0221/3/08/S08003. URL: <https://cds.cern.ch/record/1129811>.
- [7] S Chatrchyan et al. ‘The CMS experiment at the CERN LHC’. In: *JINST* 3 (2008). Also published by CERN Geneva in 2010, S08004. DOI: 10.1088/1748-0221/3/08/S08004. URL: <https://cds.cern.ch/record/1129810>.
- [8] K Aamodt et al. ‘The ALICE experiment at the CERN LHC’. In: *JINST* 3 (2008). Also published by CERN Geneva in 2010, S08002. DOI: 10.1088/1748-0221/3/08/S08002. URL: <https://cds.cern.ch/record/1129812>.
- [9] A Augusto Alves et al. ‘The LHCb Detector at the LHC’. In: *JINST* 3 (2008). Also published by CERN Geneva in 2010, S08005. DOI: 10.1088/1748-0221/3/08/S08005. URL: <https://cds.cern.ch/record/1129809>.
- [10] Joao Pequenaio. ‘Computer generated image of the whole ATLAS detector’. 2008. URL: <https://cds.cern.ch/record/1095924>.
- [11] H H J ten Kate. ‘The ATLAS Superconducting Magnet System’. In: *IEEE Trans. Appl. Supercond.* 16 (2006), pp. 499–503. DOI: 10.1109/TASC.2006.871348. URL: <https://cds.cern.ch/record/1071117>.

-
- [12] H H J ten Kate. ‘Superconducting magnet system for the ATLAS detector at CERN’. 1999. DOI: 10.1109/77.783428. URL: <https://cds.cern.ch/record/409763>.
 - [13] A Yamamoto et al. ‘Design and development of the ATLAS central solenoid magnet’. In: *IEEE Trans. Appl. Supercond.* 9.2, pt.1 (1999), pp. 852–855. DOI: 10.1109/77.783430. URL: <https://cds.cern.ch/record/404719>.
 - [14] *ATLAS inner detector*. Technical design report. ATLAS. Geneva: CERN, 1997. URL: <https://cds.cern.ch/record/331063>.
 - [15] Joao Pequeno. ‘Computer generated image of the ATLAS inner detector’. 2008. URL: <https://cds.cern.ch/record/1095926>.
 - [16] G Aad et al. ‘ATLAS pixel detector electronics and sensors’. In: *JINST* 3 (2008), P07007. DOI: 10.1088/1748-0221/3/07/P07007. URL: <https://cds.cern.ch/record/1119279>.
 - [17] F G Hügging et al. ‘Prototype performance and design of the ATLAS pixel sensors’. In: *Nucl. Instrum. Methods Phys. Res., A* 465.1 (2001), pp. 77–82. DOI: 10.1016/S0168-9002(01)00350-3. URL: <https://cds.cern.ch/record/516880>.
 - [18] J N Jackson. ‘The ATLAS semiconductor tracker (SCT)’. In: *Nucl. Instrum. Methods Phys. Res., A* 541 (2005), pp. 89–95. DOI: 10.1016/j.nima.2005.01.043. URL: <https://cds.cern.ch/record/908854>.
 - [19] A Vogel. *ATLAS Transition Radiation Tracker (TRT): Straw Tube Gaseous Detectors at High Rates*. Tech. rep. Geneva: CERN, 2013. URL: <https://cds.cern.ch/record/1537991>.
 - [20] Christian Wolfgang Fabjan and F Gianotti. ‘Calorimetry for Particle Physics’. In: *Rev. Mod. Phys.* 75 (2003), pp. 1243–1286. DOI: 10.1103/RevModPhys.75.1243. URL: <https://cds.cern.ch/record/692252>.
 - [21] Joao Pequeno. ‘Computer Generated image of the ATLAS calorimeter’. 2008. URL: <https://cds.cern.ch/record/1095927>.
 - [22] ATLAS Collaboration. *ATLAS Liquid-Argon Calorimeter: Technical Design Report*. Tech. rep. CERN-LHCC-96-041, ATLAS-TDR-2. CERN, 1996. URL: <https://cds.cern.ch/record/331061>.
 - [23] ATLAS Collaboration. *ATLAS Tile Calorimeter: Technical Design Report*. Tech. rep. CERN-LHCC-96-042, ATLAS-TDR-3. CERN, 1996. URL: <https://cds.cern.ch/record/331062>.
 - [24] L Pontecorvo. ‘The ATLAS Muon Spectrometer’. In: (2004). revised version number 1 submitted on 2003-07-27 16:31:16. DOI: 10.1140/epjcd/s2004-04-013-y. URL: <https://cds.cern.ch/record/676896>.

- [25] Daniel Dominguez, Kevin Patrice Moles and Sascha Mehlhase. ‘ATLAS detector schematics’. In: (2021). URL: <https://cds.cern.ch/record/2777214>.
- [26] G. Apollinari et al., eds. *High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report V. 0.1*. CERN Yellow Reports: Monographs. Geneva: CERN, 2017. DOI: 10.23731/CYRM-2017-004. URL: <https://cds.cern.ch/record/2284929>.
- [27] CERN. *Project Schedule*. 2025. URL: <https://project-hl-lhc-industry.web.cern.ch/content/project-schedule> (visited on 12/12/2025).
- [28] ATLAS Collaboration. ‘The ATLAS ITk Detector System for the Phase-II LHC Upgrade’. In: *Nuclear Instruments and Methods in Physics Research Section A* 1045 (2023), p. 167597. DOI: 10.1016/j.nima.2022.167597. URL: <https://cds.cern.ch/record/2816188>.
- [29] Tony Affolder. ‘ATLAS Phase II Upgrade’. In: (2022). URL: <https://cds.cern.ch/record/2799535>.
- [30] R. Baumann and Kirby Kruckmeyer. *Radiation handbook for electronics, available on TI.com (<https://www.ti.com/applications/industrial/aerospace-defense/technical-documents.html>)*. Texas Instruments, Jan. 2019.
- [31] G. Borghello et al. *Safety Factor and Testing Procedure Recommendations of the Radiation Effects Task Force*. Tech. rep. CERN, 2020. URL: <https://edms.cern.ch/ui/file/2400345/1/ATL-COM-UPGRADE-2020-016.pdf>.
- [32] V. Re et al. ‘Ionizing Radiation Effects on the Noise of 65 nm CMOS Transistors for Pixel Sensor Readout at Extreme Total Dose Levels’. In: *IEEE Trans. Nucl. Sci.* 65.1 (2018), pp. 550–557. DOI: 10.1109/TNS.2017.2777741. URL: <https://cds.cern.ch/record/2310978>.
- [33] Marc Stettler et al. ‘Radiation effects and mitigation strategies for modern FPGAs’. In: (2004). DOI: 10.5170/CERN-2004-010.112. URL: <https://cds.cern.ch/record/814041>.
- [34] Vasileios Vlagkoulis et al. ‘Configuration Memory Scrubbing of SRAM-Based FPGAs Using a Mixed 2-D Coding Technique’. In: *IEEE Trans. Nucl. Sci.* 69.4 (2022), pp. 871–882. DOI: 10.1109/TNS.2022.3151977. URL: <https://cds.cern.ch/record/2807909>.
- [35] S Baranov et al. *Estimation of Radiation Background, Impact on Detectors, Activation and Shielding Optimization in ATLAS*. Tech. rep. Geneva: CERN, 2005. URL: <https://cds.cern.ch/record/814823>.
- [36] *Public ATLAS Online Luminosity Plots for Run-2 of the LHC*. URL: <https://twiki.cern.ch/twiki/bin/view/AtlasPublic/LuminosityPublicResultsRun2>.

-
- [37] *Radiation Simulation Public Results*. URL: <https://twiki.cern.ch/twiki/bin/view/AtlasPublic/RadiationSimulationPublicResults>.
- [38] C. Zeitnitz and T.A. Gabriel. ‘The GEANT-CALOR interface and benchmark calculations of ZEUS test calorimeters’. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 349.1 (1994), pp. 106–111. ISSN: 0168-9002. DOI: [https://doi.org/10.1016/0168-9002\(94\)90613-0](https://doi.org/10.1016/0168-9002(94)90613-0). URL: <https://www.sciencedirect.com/science/article/pii/0168900294906130>.
- [39] René Brun et al. *GEANT*. CERN Program Library. Long Writeup W5013. Geneva: CERN, 1993. URL: <https://cds.cern.ch/record/1073159>.
- [40] T. A. Gabriel, B. L. Bishop and R. G. Alsmiller. *Calculation of Neutron and Proton-Induced Nuclear Reactions by Monte Carlo Methods*. Tech. rep. ORNL-4564. Oak Ridge, Tennessee: Oak Ridge National Laboratory, 1970. URL: <https://www.osti.gov/>.
- [41] A Ferrari et al. *FLUKA*. CERN Yellow Reports: Monographs. Geneva: CERN, 2005. DOI: 10.5170/CERN-2005-010. URL: <https://cds.cern.ch/record/898301>.
- [42] J. O. Johnson and T. A. Gabriel. *A user’s guide to MICAP: A Monte Carlo Ionization Chamber Analysis Package*. Technical Report ORNL/TM-10340. Oak Ridge National Lab., TN (USA), Jan. 1988. URL: <https://www.osti.gov/biblio/5567506>.
- [43] Rubén García Alía et al. ‘Single event effects in high-energy accelerators’. In: *Semicond. Sci. Technol.* 32.3 (2017), p. 034003. DOI: 10.1088/1361-6641/aa5695. URL: <https://cds.cern.ch/record/2320261>.
- [44] A Barriuso Poy et al. ‘The detector control system of the ATLAS experiment’. In: *JINST* 3 (2008), P05006. DOI: 10.1088/1748-0221/3/05/P05006. URL: <https://cds.cern.ch/record/1107601>.
- [45] D A Scannicchio. ‘ATLAS trigger and data acquisition’. In: *Nucl. Instrum. Methods Phys. Res., A* 617.1-3 (2010), pp. 306–309. DOI: 10.1016/j.nima.2009.06.114. URL: <https://cds.cern.ch/record/1290345>.
- [46] ETM professional control GmbH. *SIMATIC WinCC Open Architecture*. URL: <https://www.winccoa.com/product-information.html>.
- [47] et al. Beretta M. Moschovakos P. *DCS: Requirements Document for HL-LHC*. Tech. rep. CERN, 2020. URL: <https://edms.cern.ch/document/2276493>.

- [48] *OPC Unified Architecture: Interoperability for Industrie 4.0 and the Internet of Things*. Brochure. Version V17. Update: Cloud Initiative. OPC Foundation, 2024. URL: <https://opcfoundation.org/wp-content/uploads/2023/05/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN.pdf> (visited on 12/12/2025).
- [49] Piotr P Nikiel et al. ‘Generic OPC UA Server Framework’. In: *J. Phys.: Conf. Ser.* 664.8 (2015), p. 082039. DOI: 10.1088/1742-6596/664/8/082039. URL: <https://cds.cern.ch/record/2134637>.
- [50] Björn I Hallgren et al. ‘The Embedded Local Monitor Board (ELMB) in the LHC Front-end I/O Control System’. In: (2001). DOI: 10.5170/CERN-2001-005.325. URL: <https://cds.cern.ch/record/530675>.
- [51] Vector CANtech, Inc. *Getting Started with CANopen*. Application Note AN-AON-1-1102. Stuttgart, BW: Vector CANtech, Inc., May 2008.
- [52] M Dentan. ‘Overview of the ATLAS policy on radiation tolerant electronics’. In: (2000). DOI: 10.5170/CERN-2000-010.270. URL: <https://cds.cern.ch/record/478882>.
- [53] V. Ryjov, B. D. Serrano and B. A. Madsen. *Neutron Radiation Test-2021 Report Embedded Local Monitor Board 2 (ELMB2)*. Tech. rep. CERN, 2021. URL: https://edms.cern.ch/ui/file/2594624/1/ELMB2_NIEL_Radiation_Test_Report_2021_v1.0_docx_cpdx.pdf.
- [54] V. Ryjov et al. *CC60 Radiation Test-2021 Report Embedded Local Monitor Board 2 (ELMB2)*. Tech. rep. CERN, 2021. URL: https://edms.cern.ch/ui/file/2613346/1/ELMB2_CC60_Radiation_Test_Report_2021_v1.1_docx_cpdx.pdf.
- [55] W.J. Ashmanskas et al. ‘The AMACStar ASIC for the HL-LHC ATLAS ITk Strip detector: design, verification, testing, and quality assurance’. In: *Journal of Instrumentation* 20.08 (Aug. 2025), P08003. DOI: 10.1088/1748-0221/20/08/P08003. URL: <https://doi.org/10.1088/1748-0221/20/08/P08003>.
- [56] A. Qamesh et al. ‘System integration of ATLAS ITK Pixel DCS ASICs’. In: *Journal of Instrumentation* 18.05 (May 2023), p. C05003. DOI: 10.1088/1748-0221/18/05/C05003. URL: <https://doi.org/10.1088/1748-0221/18/05/C05003>.
- [57] Antonio Cervello Duato. *Upgrade of the ATLAS Tile Calorimeter for the High Luminosity-LHC*. Tech. rep. Geneva: CERN, 2025. DOI: 10.1051/epjconf/202533801001. URL: <https://cds.cern.ch/record/2939786>.
- [58] EMCI-EMP team. *Embedded Monitoring and Control Interface*. Tech. rep. CERN, 2025. URL: https://edms.cern.ch/ui/file/2332346/2/EMCI_electronic_specifications_v1.3_docx_cpdx.pdf.

-
- [59] Paulo Moreira et al. *lpGBT documentation*. 2024. URL: <https://cds.cern.ch/record/2809058>.
- [60] Jan Troska et al. ‘The VTRx+, an optical link module for data transmission at HL-LHC’. In: *PoS TWEPP-17* (2017), p. 048. DOI: 10.22323/1.313.0048. URL: <https://cds.cern.ch/record/2312396>.
- [61] CERN. *FEASTMP*. Aug. 2016. URL: https://power-distribution.web.cern.ch/assets/datasheets/FEAST2Mod_Datasheet_gb2016.pdf.
- [62] EMCI-EMP team. *EMCI - DC Option 1 (Default) Picture*. URL: https://edms.cern.ch/ui/file/3252507/1/EDA-04251-V3-0_3D-top.jpg.
- [63] Irving S. Reed and Gustave Solomon. ‘Polynomial Codes over Certain Finite Fields’. In: *Journal of the Society for Industrial and Applied Mathematics* 8.2 (1960), pp. 300–304. DOI: 10.1137/0108018. URL: <https://doi.org/10.1137/0108018> (visited on 07/12/2025).
- [64] L. Olanterä J. Troska C. Soos. *The Versatile Link+ Application Note*. Tech. rep. CERN, 2025. URL: <https://edms.cern.ch/ui/file/2149674/1/VTRxPlusApplicationNote.pdf>.
- [65] Trenz Electronic. *TE0807 TRM*. June 2021. URL: https://www.trenz-electronic.de/trenzdownloads/Trenz_Electronic/Modules_and_Module_Carriers/5.2x7.6/TE0807/REV03/Documents/TRM-TE0807-03-v.27-06_10_2021.pdf.
- [66] Trenz Electronic. *MPSoC Module with AMD Zynq™ UltraScale+™ ZU4EG-1E*. URL: <https://www.trenz-electronic.de/en/MPSoC-Module-with-AMD-Zynq-UltraScale-ZU4EG-1E-4-GByte-DDR4-5.2-x-7.6-cm/TE0807-04-4BE81-A>.
- [67] Xilinx. *Zynq UltraScale+ MPSoC Data Sheet: Overview*. URL: <https://docs.amd.com/v/u/en-US/ds891-zynq-ultrascale-plus-overview>.
- [68] EMCI-EMP team. *Mechanical drawings*. URL: https://edms.cern.ch/ui/file/3264458/1/EDA-04490-V3-0_3D-top_jpg_cpdpdf.pdf.
- [69] EMCI-EMP team. *Embedded Monitoring Processor*. Tech. rep. CERN, 2025. URL: https://edms.cern.ch/ui/file/2332346/2/EMCI_electronic_specifications_v1.3_docx_cpdpdf.pdf.
- [70] Samtec, Inc. *CERN Versatile Link+ FireFly™ (CERN-B) Data Sheet*. Samtec confidential datasheet; available under NDA within the Versatile Link+ project. If you have an internal document ID (e.g., CERN EDMS), add it as `number = {...}`. Samtec, Inc. Sept. 2019.
- [71] EMCI-EMP team. *J3095535 – EMP CHASSIS V2*. URL: https://edms.cern.ch/ui/file/3189960/1/J3095535_-_EMP_CHASSIS_V2.pdf.

- [72] Paris Moschovakos et al. ‘The Embedded Monitoring Processor for High Luminosity LHC’. In: *JACoW ICALEPCS 2023* (2023), pp. 1470–1474. DOI: 10.18429/JACoW-ICALEPCS2023-THPDP063. URL: <https://cds.cern.ch/record/2896178>.
- [73] D. Ecker et al. ‘EMCI-EMP: developments and experience with the novel detector control solution’. In: *Journal of Instrumentation* 20.01 (Jan. 2025), p. C01020. DOI: 10.1088/1748-0221/20/01/C01020. URL: <https://doi.org/10.1088/1748-0221/20/01/C01020>.
- [74] Arm Limited. *AMBA AXI Protocol Specification*. ARM IHI 0022. Version Issue L. Non-confidential. Arm Limited. 27th Aug. 2025. URL: <https://developer.arm.com/documentation/ih0022/latest/> (visited on 12/12/2025).
- [75] The kernel development community. *The Userspace I/O HOWTO*. URL: <https://www.kernel.org/doc/html/v4.14/driver-api/uio-howto.html>.
- [76] Patrick Mochel and Mike Murphy. *sysfs – The filesystem for exporting kernel objects*. Linux kernel documentation (revised 16 Aug 2011). 2011. URL: <https://docs.kernel.org/filesystems/sysfs.html> (visited on 12/12/2025).
- [77] Benedikt Spranger. *libuio - UserspaceIO helper library*. URL: <https://github.com/missinglinkelectronics/libuio>.
- [78] Piotr Nikiel, Paris Moschovakos and Stefan Schlenker. ‘quasar : The Full-Stack Solution for Creation of OPC-UA Middleware’. In: (2020), MOPHA100. DOI: 10.18429/JACoW-ICALEPCS2019-MOPHA100. URL: <https://cds.cern.ch/record/2778537>.
- [79] S. Schlenker, P. Nikiel, P. Moschovakos et al. *SoCs and Control System Integration*. June 2019. URL: https://indico.cern.ch/event/799275/contributions/3444104/attachments/1862433/3070213/SoCs_and_Control_System_Integration.pdf.
- [80] EMCI-EMP team. *EMP-Firmware Gitlab Repository*. URL: <https://gitlab.cern.ch/emci-emp/emp-firmware>.
- [81] D Blasco Serrano et al. ‘Description and status of the EMCI-EMP interface’. In: *JINST* 17.06 (2022), p. C06012. DOI: 10.1088/1748-0221/17/06/C06012. URL: <https://cds.cern.ch/record/2816683>.
- [82] GBT Team. *GBT-SC Gitlab repository*. URL: <https://gitlab.cern.ch/gbtsc-fpga-support/gbt-sc>.
- [83] GBT Team. *LPGBT-FPGA Gitlab Repository*. URL: <https://gitlab.cern.ch/gbt-fpga/lpgbt-fpga>.
- [84] Julian Maxime Mendez et al. *New slow-control FPGA IP for GBT based system and status update of the GBT-FPGA project*. 2018. DOI: 10.22323/1.313.0083. URL: <https://cds.cern.ch/record/2312287>.

-
- [85] P Moreira et al. *GBTX manual*. CERN, 2021. URL: <https://cds.cern.ch/record/2809057>.
 - [86] Julian Maxime Mendez et al. ‘New slow-control FPGA IP for GBT based system and status update of the GBT-FPGA project’. In: *PoS TWEPP-17* (2018), p. 083. DOI: 10.22323/1.313.0083. URL: <https://cds.cern.ch/record/2312287>.
 - [87] GBT Team. *LPGBT-FPGA Documentation*. URL: <https://lpgbt-fpga.web.cern.ch/doc/html/>.
 - [88] EMCI-EMP team. *epos Gitlab Repository*. URL: <https://gitlab.cern.ch/emci-emp/epos>.
 - [89] Advanced Micro Devices, Inc. *PetaLinux Tools Documentation: Reference Guide*. UG1144. Version 2023.2 (English). Advanced Micro Devices, Inc. 18th Oct. 2023. URL: <https://docs.amd.com/r/2023.2-English/ug1144-petalinux-tools-reference-guide/Overview> (visited on 12/12/2025).
 - [90] P. Moschovakos. *LpGbtSoftware Gitlab Repository*. URL: <https://gitlab.cern.ch/atlas-dcs-emp/LpGbtSw>.
 - [91] D. Ecker. *empTools Gitlab Repository*. URL: <https://gitlab.cern.ch/emci-emp/emp-tools>.
 - [92] noasic GmbH. *Register Management Done Right*. URL: <https://airhdl.com/#/>.
 - [93] M. Kristensen P. Nikiel P. Moschovakos. *PuzzledLizardWizard Gitlab Repository*. URL: <https://github.com/missinglinkelectronics/libuio/>.
 - [94] P. Moschovakos. *OpcUaLpGbtServer Gitlab Repository*. URL: <https://gitlab.cern.ch/atlas-dcs-opcua-servers/OpcUaLpGbtServer>.
 - [95] D. Ecker. *OpcUaEmpMonitor Gitlab Repository*. URL: <https://gitlab.cern.ch/atlas-dcs-opcua-servers/opcuaempmon>.
 - [96] Advanced Micro Devices, Inc. *AXI DMA LogiCORE IP Product Guide*. Version 7.1. 24th June 2025. URL: https://docs.amd.com/r/en-US/pg021_axi_dma (visited on 12/12/2025).
 - [97] Advanced Micro Devices, Inc. *AXI Central Direct Memory Access (CDMA) LogiCORE IP Product Guide*. Version 4.1. 11th July 2025. URL: <https://docs.amd.com/r/en-US/pg034-axi-cdma> (visited on 12/12/2025).
 - [98] Xilinx, Inc. *AXI Interconnect v2.1 LogiCORE IP Product Guide*. PG059. Version 2.1. Xilinx, Inc. 17th May 2022. URL: <https://docs.amd.com/r/en-US/pg059-axi-interconnect> (visited on 12/12/2025).
 - [99] Xilinx, Inc. *AXI Block RAM (BRAM) Controller v4.1 Product Guide*. PG078. Version 4.1. Xilinx, Inc. 22nd May 2019. URL: <https://docs.amd.com/v/u/en-US/pg078-axi-bram-ctrl> (visited on 12/12/2025).

- [100] Xilinx, Inc. *Block Memory Generator v8.4 Product Guide*. PG058. Version 8.4. Xilinx, Inc. 6th Aug. 2021. URL: <https://docs.amd.com/v/u/en-US/pg058-blk-mem-gen> (visited on 12/12/2025).
- [101] Advanced Micro Devices, Inc. *AXI Verification IP LogiCORE IP Product Guide*. PG267. Version 1.1. Advanced Micro Devices, Inc. 7th June 2024. URL: <https://docs.amd.com/r/en-US/pg267-axi-vip> (visited on 12/12/2025).
- [102] Inter-Tech Elektronik Handels GmbH. *ASPOWER R1A-KH0400: Specifications*. Article No. 99997245. Inter-Tech Elektronik Handels GmbH. 7th Sept. 2021. URL: https://www.inter-tech.de/files/images/psu/99997245/ASPOWER_R1A-KH0400-e.pdf (visited on 12/12/2025).

List of Acronyms

ADC Analog-to-Digital Converter.

AHB Advanced High-performance Bus.

AMBA Advanced Microcontroller Bus Architecture.

APB Advanced Peripheral Bus.

API Application Programming Interface.

APU Application Processing Unit.

ASIC Application-Specific Integrated Circuit.

AXI Advanced eXtensible Interface.

B2B Board-to-Board.

BE Back-End.

BER Bit Error Rate.

BGA Ball Grid Array.

BiCMOS Bipolar CMOS.

BJT Bipolar Junction Transistor.

BRAM Block RAM.

BSP Board Support Package.

CAN Controller Area Network.

CD Continuous Delivery.

CDC Clock Domain Crossing.

CDR Clock and Data Recovery.

CI Continuous Integration.

CIC Common Infrastructure Controls.

CMOS Complementary Metal–Oxide–Semiconductor.

COTS Commercial Off-The-Shelf.

CRC	Cyclic Redundancy Checks.
CSV	Comma-Separated Values.
DAC	Digital-to-Analog Converter.
DCS	Detector Control System.
DD	Displacement Damage.
DDMI	Digital Diagnostic Monitoring Interface.
DDR4 SDRAM	Double Data Rate Synchronous Dynamic Random Access Memory.
DIP	Dual In-line Package.
DMA	Direct Memory Access.
DSP	Digital Signal Processing.
DSS	Detector Safety System.
EC	External Control.
ECC	Error Correction Code.
EEPROM	Electrically Erasable Programmable Read-Only Memory.
eLink	electrical Link.
ELMB	Embedded Local Monitoring Board.
EMCI	Embedded Monitoring and Control Interface.
eMMC	embedded MultiMedia Card.
EMP	Embedded Monitoring Processor.
FE	Front-End.
FEC	Forward Error Correction.
FF	Flip-Flop.
FIFO	First In, First Out.
FMC	FPGA Mezzanine Card.
FPD	Full Power Domain.
FPGA	Field Programmable Gate Array.
FSBL	First Stage Bootloader.

FSM Finite-State Machine.

GBT-SCA Giga-Bit Transceiver - Slow Control Adapter.

GBTx GigaBit Transceiver.

GCS Global Control Station.

GEANT3 GEometry ANd Tracking.

GPIO General Purpose Input/Output.

GUI Graphical User Interface.

HAL Hardware Access Layer.

HD High-Density.

HDL Hardware Description Language.

HDLC High-Level Data Link Control.

HETC High-Energy Transport Code.

HL-LHC High-Luminosity Large Hadron Collider.

HLT High Level Trigger.

HMI Human-Machine Interfaces.

HP High-Performance.

HPC High-Pin-Count.

HTML Hypertext Markup Language.

I2C Inter-Integrated Circuit.

IBERT Integrated Bit Error Ratio Tester.

IBL Insertable B-Layer.

IC Internal Control.

ID Inner Detector.

IIO Industrial I/O.

ILA Integrated Logic Analyzer.

IP Interaction Point.

IP-core Intellectual Property Core.

IRQ Interrupt Request.

ISP In-System Programming.

ITk Inner Tracker.

JCOP Joint Controls Project.

JSON JavaScript Object Notation.

JTAG Joint Test Action Group.

LAN Local Area Network.

LAr Liquid Argon.

LCS Local Control Station.

LDO Low-Dropout Regulators.

LDR Low-Dose-Rate.

LED Light-Emitting Diode.

LEP Large Electron-Positron.

LHC Large Hadron Collider.

Linac4 Linear accelerator 4.

LM Latency Measurement.

LPD Low Power Domain.

lpGBT low-power GigaBit Transceiver.

LS3 Long Shutdown 3.

LUT Lookup Table.

LVCMOS Low Voltage CMOS.

MBU Multi-Bit Upset.

MDIO Management Data Input/Output.

MDT Monitored Drift Tube.

MGT Multi-Gigabit Transceivers.

MICAP Monte Carlo Ionization Chamber Analysis Package.

MIO Multiplexed I/O.

MMG Micromega.

MOSFET Metal–Oxide–Semiconductor Field-Effect Transistor.

MPO Multifiber Termination Push-on/Pull-off.

MPSoC Multiprocessing System-on-Chip.

MSB Most Significant Bit.

MTP Multi-fiber Termination Push-on.

NFS Network File System.

NIEL Non-Ionizing Energy Loss.

NMT Network Management.

OPC UA Open Platform Communications Unified Architecture.

OS Operating System.

PCB Printed Circuit Board.

PCIe Peripheral Component Interconnect express.

PDO Process Data Objects.

PHY Physical Transceiver.

PL Programmable Logic.

PLL Phase-Locked Loop.

PMU Platform Management Unit.

PRBS Pseudo-Random Binary Sequence.

PS Processing System.

PSB Proton Synchrotron Booster.

PUSM Power-Up State Machine.

PVSS Process Visualization and Steering System.

QoS Quality of Service.

QSPI Quad Serial Peripheral Interface.

quasar quick OPC UA server generation framework.

RadFET Radiation-Sensitive p-MOSFET.

RAM Random-Access Memory.

RF radio-frequency.

ROM Read-Only Memory.

rootfs root filesystem.

RPC Resistive Plate Chamber.

RPU Real-Time Processing Unit.

RTR Remote Transmission Request.

RX Receive.

SATA Serial ATA.

SC Serial Control.

SC-DMA Slow-Control Direct Memory Access.

SCADA Supervisory Control and Data Acquisition.

SCS Sub-Detector Control Station.

SCT Semiconductor Tracker.

SD Secure Digital.

SDK Software Development Kit.

SDO Service Data Objects.

SEB Single Event Burnout.

SEE Single-Event Effect.

SEGR Single Event Gate Rupture.

SEL Single Event Latch-up.

SerDes Serializer/Deserializer.

SEU Single-Event Upset.

SFP+ Enhanced Small Form-factor Pluggable.

SI International System of Units.

SoC System-on-Chip.

SoM System-on-Module.

SPI Serial Peripheral Interface.

SPS Super Proton Synchrotron.

SRAM Static Random-Access Memory.

STGC Small-Strip Thin Gap Chamber.

SYSMON System Monitors.

TAP Test Access Port.

TCP/IP Transmission Control Protocol/Internet Protocol.

TDAQ Data Acquisition and Trigger System.

TGC Thin Gap Chamber.

TID Total Ionizing Dose.

TMR Triple Modular Redundancy.

TRT Transition Radiation Tracker.

TWI Two-Wire Interface.

TX Transmit.

UA Unified Architecture.

UART Universal Asynchronous Receiver-Transmitter.

UIO Userspace I/O.

UPS Uninterruptible Power Supply.

USART Universal Synchronous/Asynchronous Receiver Transmitter.

USB Universal Serial Bus.

VCSEL Vertical-Cavity Surface-Emitting Laser.

VL+ Versatile Link+.

VMA Virtual Memory Area.

VME Versa Module Eurocard.

VTRx+ Versatile Transceiver+.

WinCC OA WinCC Open Architecture.

XML Extensible Markup Language.

XVC Xilinx Virtual Cable.

List of Figures

2.1.	Overview of CERN's accelerator complex and the different experiments [3].	6
2.2.	Overview of the ATLAS detector and its toroidally arranged sub-detectors and magnet systems [10].	8
2.3.	Illustration of the magnetic fields and their respective field lines within the ATLAS detector [12].	9
2.4.	Overview of the Inner Detector, showing its three constituting sub-systems: the Pixel, SCT, and TRT detector [15].	10
2.5.	Overview of the location of the ATLAS calorimeter and its sub-systems [21].	12
2.6.	Computer-generated image detailing the areas of the ATLAS detector covered by the muon spectrometers [25].	13
2.7.	Timeline detailing the installation of the HL-LHC and ATLAS/CMS upgrades during the LS3 and the final start of operation (adapted from [27]).	15
3.1.	Integrated luminosity for p-p physics at $\sqrt{s}=13$ TeV during LHC Run 2 [36].	23
3.2.	TID measurements from RadFETs (left) and 1 MeV neutron fluence measurements from BPW34 diode sensors (right), in the ID during LHC Run 2, shown together with the corresponding pre-run simulation results [37].	24
3.3.	Simulation results for the TID in Gy and the total fluence of hadrons with $E > 20$ MeV per cm^2 , for an integrated luminosity of 4000 fb^{-1} , in the tracking and calorimeter regions of the Phase-II ATLAS detector. The geometry model used is symmetric in azimuth, and with respect to $z = 0$ [37].	25
4.1.	Definition of Front-End and Back-End components within a DCS system. Adapted from Ref. [47].	28
4.2.	Back-End architecture of the ATLAS DCS. Adapted from Ref. [47]).	29
4.3.	Block diagram of the Embedded Local Monitoring Board, showing the three electrically isolated and separately powered functional sections. Adapted from Ref. [50].	32

5.1.	Schematic view of the primary DCS data path, for control and monitoring data which require a high degree of reliability and availability. Implementation of this path is either based on the ELMB2 or on the EMCI-EMP system. Adapted from Ref. [47].	39
5.2.	EMCI-EMP system overview. The EMP forms the backbone of the novel DCS interface, connecting up to 12 EMCIs in a star topology via high-speed optical links and interfacing with the distributed SCADA system via LAN.	40
5.3.	Block diagram of the EMCI. At its core, the EMCI is based on the lpGBT, a radiation-tolerant ASIC, and is powered by two FEASTMP DC-DC converters. Connectivity to the EMP is provided via a VTRx+ optical transceiver, while the eLinks for interfacing with on-detector FEs are accessible through the HPC FMC connector. Adapted from Ref. [58].	43
5.4.	Computer generated image of the EMCI PCB [62].	44
5.5.	Architecture of the lpGBT ASIC [59].	46
5.6.	Overview of the lpGBT downlink frame structure before interleaving. Adapted from Ref. [59].	50
5.7.	Overview of the lpGBT uplink frame structure for the configuration using FEC5 and an uplink data rate of 10.24 Gbit/s.	51
5.8.	Front and back side of the TE0807 MPSoC module [66].	53
5.9.	Computer generated 3D image of the EMP baseboard [68].	55
5.10.	Block diagram of the EMP baseboard, illustrating the integration of the Zynq on the TE0807 MPSoC module via the B2B connectors and the external interfaces provided by the EMP baseboard [69]. . .	56
5.11.	Evolution of the measured temperatures on the EMP baseboard without forced airflow. The plot shows the measured values for the I2C temperature sensors (close below the SoM, between the FireFly modules, and near the DC/DC converters) as well as the internal temperatures of the FireFly modules and the Zynq PS.	61
5.12.	Infrared image of the FireFly and SoM heatsink during the no-airflow test.	62
5.13.	Computer-generated image of the 19-inch rack-mount EMP chassis, showing three slots reserved for EMP boards and a fourth slot housing a redundant server power-supply [71].	63
6.1.	Modular architecture of the EMCI-EMP interface, showing building blocks for specific target applications. Orange denotes components from external sources; turquoise indicates modules provided as part of the reference software; lila represents custom building blocks to be developed for the respective FE hardware. Adapted from Ref. [73].	65

6.2.	The five independent channels of an AXI4 memory-mapped interface (applicable to both AXI4 and AXI4-Lite). Each channel is unidirectional and has its own pair of VALID/READY handshake signals. The separation of address (AW, AR), data (W, R), and response (B) channels enables write and read transactions to progress concurrently and be pipelined, thereby improving throughput compared to single-channel buses. Adapted from Ref. [74].	69
6.3.	AXI4 write transaction timing. During the address phase, the master drives AWADDR/AWVALID and, optionally, WDATA/WVALID . The slave acknowledges the address phase by asserting AWREADY . During the data phase, the slave asserts WREADY when it is ready to receive data, and the master transmits data over WDATA while keeping WVALID asserted. On the last dataword, the master additionally asserts WLAST to mark the last data beat. After the data phase, the master asserts BVALID , which the slave acknowledges by asserting BREADY and transmitting the status code via BRESP , thereby ending the transaction. Adapted from Ref. [74].	70
6.4.	AXI4 read transaction timing. During the address phase, the master drives ARADDR/ARVALID . The slave acknowledges the address phase by asserting ARREADY . During the data phase, the master asserts RREADY when it is ready to receive data, and the slave transmits data over RDATA while keeping RVALID asserted. On the last dataword, the slave additionally asserts RLAST to mark the last data beat. During the entire transaction, the status of each data beat is transmitted in parallel to the data via the RESP signal. Adapted from Ref. [74]. . .	71
6.5.	Overview of the quasar framework, illustrating the layered architecture including the HAL, Device logic, generated server components, and the external OPC UA server toolkit. Adapted from Ref. [79]. . .	76
6.6.	Overview of the lpGBT frame codec fabric: The lpGBT IC channel is exposed via an AXI4-Lite interface to the PS. A dedicated FSM manages register transactions with the GBT-IC IP-core [82], which handles lpGBT fields and acts as a SerDes between the LPGBT-FPGA IP-core [83] and FSM. The LPGBT-FPGA implements frame coding and connects directly to the MGT, while FE logic interfaces directly through the emp-lpGBT IP-core.	78
6.7.	High-level overview of the architecture of the emp-lpgbt IP-core. . .	79
6.8.	Architecture of the GBT-IC IP-core. Adapted from Ref. [84]. . . .	80
6.9.	Overview of the asymmetric architecture of the LPGBT-FPGA IP-core. Adapted from Ref. [87].	81
6.10.	Overview of the customization GUI of the EMP-MGT IP-core. The GUI enables the parametrization.	84
6.11.	Clock-domain overview of the EMP FPGA firmware reference design.	85

6.12.	Output of the <code>lpGbtAdc</code> demonstrator from the <code>LpGbtSw</code> library, showing the raw ADC values read from the lpGBT over the IC channel using the UIO backend.	89
6.13.	Command-line output of the <code>empTools</code> demonstrator for the FireFly modules, showing typical measurements, diagnostic values, and control parameters of a single FireFly module, read from its internal registers via the TWI interface.	90
6.14.	Output of the SYSMON demonstrator tool, reporting on-chip temperatures and supply voltages obtained from the IIO subsystem. . .	91
6.15.	Quasar-generated OPC UA information model for an lpGBT device.	93
6.16.	Architecture of the lpGBT OPC UA server ecosystem, including its interface with the hardware, and the connectivity to various clients via OPC UA. The server communicates with the lpGBT through the <code>LpGbtSw</code> library to access the hardware registers. External client applications using OPC UA over TCP/IP, including general-purpose diagnostic tools (e.g., UA Expert) and the SCADA system, which integrates lpGBT control into the DCS. Adapted from Ref. [72]. . .	94
6.17.	Architecture of the EMP Monitoring & Control OPC UA server. The server interfaces with the PL and PS SYSMON, the FW-Debughub, and the PS periphery, using the <code>empTools</code> library API. Like the lpGBT server, the EMP OPC UA server communicates with remote clients over TCP/IP, allowing SCADA applications and diagnostic tools to access the EMP's status and perform control operations. . .	95
7.1.	Benchmark study test setup comprising one EMP connected to 12 EMCIs.	97
7.2.	Data path on the lpGBT for DownLink Group Data Source loopback. Adapted from Ref. [58].	98
7.3.	Functional diagram of the eLink Interface IP core and the data path of eLink data within EMP.	99
7.4.	Flowchart representing the implementation of the benchmark application. For each iteration, the application writes a 32 bit data word to the corresponding register of the eLink Interface IP-core and then waits for the associated IRQ. Once the interrupt is received, it is disabled, the received data is read from the register, and their integrity is verified. Finally, the application checks the time elapsed since the process started: if the fixed limit of 120 s has not yet been reached, it sleeps for the configured inter-transaction delay before starting the next iteration; otherwise, it terminates and returns.	101

7.5.	Average throughput per EMCI [Mbps] as a function of the CPU load [%] for different configurations of EMCIs and utilized CPU cores. The CPU load is given as the sum over all four cores, such that 100 % corresponds to one fully utilized core and 400 % to all four cores fully utilized.	103
7.6.	Total system throughput [Mbps] as a function of the CPU load [%]. Note the near-linear, almost identical scaling across all configurations.	104
7.7.	Achieved average throughput rate per EMCI [Mbps] as a function of the programmed delay [μ s].	105
7.8.	Average CPU load [%] as a function of the programmed inter-transaction delay [μ s] between requests.	105
7.9.	The bar chart shows the average throughput [Mbps] per EMCI for different system configurations with no delay between requests. . . .	106
7.10.	The bar chart shows the total achieved throughput [Mbps] for different system configurations with no delay between requests.	107
7.11.	Data received without errors and missed IRQ over a period of 12 hours.	108
8.1.	Software, firmware and hardware layers of the SC-DMA based FE eLink communication stack.	110
8.2.	An architectural overview of the SC-DMA IP core and its constituting functional entities, including their basic interconnections.	113
8.3.	Overview of the customization GUI of the SC-DMA IP-core. The GUI enables the parametrization to meet operational requirements for integration with lpGBT hardware and within EMP firmware. . . .	114
8.4.	Overview of the AXI4-Lite register map. This map describes the implementation of the configuration and monitoring interface of the SC-DMA IP-core. It shows each register's offset, name, and access properties.	116
8.5.	Simulation of an AXI-Read Engine timeout and the automated recovery sequence initiated through the Watchdog. It is shown that the transmit timeout counter increases until it reaches the configured limit <code>TX_LIMIT</code> , upon which the Watchdog leaves the <code>DMA_BUSY</code> state and enters the notification state <code>WAIT_ACK</code> , asserting the <code>IRQ</code> . When the software acknowledges the <code>IRQ</code> (<code>ACK</code>) and the <code>STATUS_STROBE</code> signals that the status register got read, the watchdog transitions to the <code>RESET</code> state, asserts the <code>RST_WD</code> , and forces all DMA FSMs to return to <code>IDLE</code> , restoring the SC-DMA subsystem to a well-defined state.	118

-
- 8.6. Address phase of the AXI4-Read Engine. Upon detection of the **START** bit, the engine leaves the **IDLE** state, configures the AXI **AR**-channel, and asserts **ARVALID**. The handshake is complete when **ARREADY** is asserted from the slave, after which the Read Engine transitions to the **READ** state. 119

 - 8.7. Read-data phase of the AXI4-Read Engine following a successful address phase. With **RREADY** asserted, the engine marks itself ready for the reception of data. The engine accepts 128 bit data beats on **RDATA** whenever **RVALID** is high, while **RID** and **RRESP** confirm the transaction identifier and the read response, respectively. The Read Engine remains in the **READ** state until the final beat of the burst and only then moves to **FINISHED**. 120

 - 8.8. Abstract representation of the lpGBT frame assembly process performed by the Downlink Frame Assembler. The process extracts data slices from each buffered eLink according to the specific eGroup configuration and the related eLink field width, and copies them to their target positions within the lpGBT downlink frame. The bit-field widths shown are chosen for illustration only and are selected arbitrarily. 121

 - 8.9. After the Assembler leaves the **IDLE** state and enters the **CAPTURE** phase, the valid data beats on **RDATA** are stored into successive entries of the **BUF** array until the end of the burst is detected through **RLAST**. 122

 - 8.10. Single cycle of the **SORT/WRITE** of the Downlink Frame Assembler. After collecting eLink frames, the Assembler enters the **SORT** state, where it deserializes the buffered data into an lpGBT-compliant downlink frame **lpGBT FRAMES**. Once the frame is assembled, the FSM transitions to **WRITE** and asserts **FIFO WR ENABLE**, causing the 32 bit lpGBT frame on **FIFO DIN** to be consumed by the FIFO. . . . 123

 - 8.11. Simulation of the send process for downlink lpGBT frames. The **SEND** (40 MHz) flag synchronized from the **SEND** (AXI **ACLK**) flag set through the Assembler FSM enables the FIFO read controller, which asserts **FIFO RD ENABLE** and streams the 32 bit data frames on **FIFO DOUT** as long as **FIFO RD VALID** is high. The data are forwarded directly to the **DOUT** bus for interfacing with the emp-lpgbt IP-core. 123

- 8.12. Abstract representation of the frame collection process implemented by the Uplink Frame Collector. For each lpGBT uplink eGroup, incoming eLink channels are continuously scanned to identify valid frame start (S) and end (E) delimiters. Upon detection of the start delimiter, the data patterns (8 bit, 16 bit and 32 bit) are buffered into a 3D array until the end delimiter is received. The frames are stored in the array such that their locations can later be used to determine the eGroup association and eLink index. 124
- 8.13. Simulation of the Uplink Collector during the COLLECT phase. Incoming 224 bit lpGBT frames on DIN are monitored per eLink channel, with each per-channel state transitioning from LS_IDLE to LS_ACTIVE once valid frame patterns have been detected. Subsequent data patterns are directly written to the BUF array while the status of each link is recorded in the LINK STATE. The Collector remains in the COLLECT phase until all active logic blocks LS_FINISHED. 125
- 8.14. Uplink Collector FLATTEN/WRITE FIFO sequence. After all active frames have been collected, the Collector first flattens the eLink frames into a 224 bit vector and concatenates the 32 bit metadata vector. Then the resulting 256 bit word is written to the FIFO, after which the remaining frames are processed in the same manner. Once all received frames are pushed to the FIFO, the Collector transitions into the FINISHED state. 126
- 8.15. Address phase of the AXI4-Write Engine. When the uplink Collector signals completion (COLLECTOR DONE), the write engine configures the AXI AW-channel, asserts AWVALID, leaves IDLE, and enters the HANDSHAKE state. The transaction is accepted once AWREADY is asserted. At this point, the engine is ready to stream the data beats and transitions to the WRITE stage. 127
- 8.16. Write phase of the AXI4-Write Engine. During the WRITE state, the engine asserts WVALID and drives 128 bit words on WDATA, sourced from the 256 bit FIFO output FIFO DOUT. The process interleaves the FIFO so that the next data word is already read as soon as the previous data word is transmitted, ensuring continuous streaming of data beats on the AXI bus as long as WREADY is asserted by the slave. The final data beat is marked through WLAST, upon which the FSM transitions into the response phase. 128
- 8.17. During the response phase, the engine checks for the final handshake on the last data beat of the write phase and only after asserts BREADY. Next, the engine waits for a valid response BVALID and BRESP confirming the successful completion of the burst write to system memory, and then transitions into the Finished state. . . 129

8.18.	Block-level architecture of the SC-DMA validation testbench. Six instances of the SC-DMA IP-core are instantiated in parallel, each connected via its AXI4 master interface to the shared memory subsystem composed of a Block Memory Generator and two AXI BRAM Controllers. Separate AXI Crossbars fan out the read and write channels, enabling concurrent access to the same BRAM-based memory by all SC-DMA instances. The integration of additional AXI Verification IP-cores into the shared AXI bus enables transaction-level debugging and diagnostics of the AXI protocol during simulation. . .	130
8.19.	Transaction-level simulation of the multi-channel validation testbench. The Figure shows the IRQ signals for each SC-DMA IP-core, along with the AXI protocol view obtained from the AXI Verification IP-cores, which summarizes the activity traced between the AXI crossbar master interfaces and the AXI BRAM Controllers. The simulation is used to verify the AXI protocol compliance of the SC-DMA IP-core as well as the arbitration, ordering, and latency behavior of the AXI Crossbars.	131
8.20.	Post-implementation resource utilization for the EMP FPGA reference firmware design with 12 instantiated SC-DMA IP-cores.	135
9.1.	Sequence diagram showing the interaction between the user-space test application, the kernel driver, the SC-DMA IP-core in the PL, the system memory, and the interrupt controller during a single DMA transaction.	142
9.2.	Average CPU load per core as a function of the inter-transaction delay of different system configurations. The markers denoting configurations with 1, 2, 3, and 4 active EMCIs largely overlap, leaving only the yellow markers visible.	144
9.3.	Average throughput per EMCI as a function of total CPU utilization.	145
9.4.	Total system throughput as a function of total CPU utilization. . . .	146
9.5.	Average throughput per EMCI as a function of total CPU utilization in a configuration in the 28-uplink configuration.	147
9.6.	Total CPU utilization as a function of the programmed inter-transaction delay comparing the 16- and 28-uplink configuration.	148
9.7.	Average transaction rate per EMCI as a function of total CPU utilization for the 16- and 28-uplink configurations.	149
9.8.	CPU load skew between user and kernel space as a function of the programmed inter-transaction delay for the configuration with 12 active EMCI-EMP links, comparing the 16- and 28-uplink configuration.	150

9.9.	Maximum average throughput per EMCI for all evaluated system configurations, comparing the 16- and 28-uplink configurations in the zero-delay regime.	151
9.10.	Maximum total system throughput for all evaluated system configurations in the zero-delay regime, comparing the 16-and 28-uplink configurations.	152
9.11.	Long-term stability test of the SC-DMA subsystem: cumulative data transmitted as a function of time over a period of more than 167 h. .	153
A.1.	Dimensions for Standard Pin-fin Heatsink.	190
A.2.	DCS rack in SDX1 Level 2 with deployed EMP test crate (below the three large empty white cover panels)	191
A.3.	EMP crate layout.	192
A.4.	Fully assembled prototype EMP crate v1.	193
A.5.	FireFly modules with heatsink fan.	193
A.6.	Temperature evolution without an equipped FireFly fan.	197
A.7.	Temperature evolution with additional FireFly fan during a period without stable beams.	198
A.8.	Temperature evolution with equipped FireFly fans during a period with regular stable beams (computing farm active).	199
A.9.	Temperature evolution with equipped FireFly fan during the simulation of an intermittent failure of the rack door ventilation by opening the rack door. The figure shows two periods of increased temperature: the first, an initial test of approximately 1 hour, and the second, a test of approximately 20 hours. The figure also shows that the temperature returns to normal within less than 2 minutes after the rack door is closed.	200
A.10.	EMP crate v2 fully equipped. The chassis fans (top) serve as a replacement for the FireFly heatsink fans.	202
B.1.	OPC UA information model for the EMP monitoring server.	203

List of Tables

5.1.	Overview of the radiation qualification of EMCI components.	45
5.2.	Overview of active eGroups depending on the configured uplink line rate and the FEC scheme. The FEC5 and FEC12 denote different Reed-Solomon FEC schemes, where FEC5 can correct up to 5 (or 10 consecutive) wrong bits and FEC12 can correct up to 12 (or 24 consecutive) wrong bits at the respective line rates [59].	47
5.3.	Overview of the achievable eLink bandwidth depending on eGroup configuration and the uplink and downlink line rates [59].	47
5.4.	Uplink frame structure as a function of data rate and FEC scheme [59].	51
5.5.	Settings of the DIP switches on the EMP baseboard for configuring the supported boot modes [69].	60
8.1.	Assembler latency for different combinations of active eGroups and eLinks.	133
8.2.	Post-synthesis utilization table showing the LUT and FF scaling of the Uplink Frame Collector for different eGroup configurations. . . .	134

Acknowledgements

This thesis and the work presented herein would not have been possible without the expertise, guidance, and encouragement of many people, whom I would like to acknowledge here.

First, I would like to thank Prof. Dr.-Ing. Dietmar Tutsch and Prof. Dr. Christian Zeitnitz for their trust and for allowing me to pursue my doctoral studies at CERN within the Gentner Programme.

I am also grateful to my team, with whom I have worked over the past three years. Paris, Niko, Slava, and Pablo—thank you for the many valuable discussions and for your support.

My sincere thanks go to Vladimir. I greatly appreciate the countless hours we spent in the lab troubleshooting hardware issues, your support when deadlines were looming, our many conversations, and the numerous coffee capsules I was allowed to “borrow.”

I am especially grateful to Stefan for making this work possible and for supervising the project. Thank you for being so patient, for reviewing so many texts, for taking the time to discuss and reflect, and above all for your optimism.

Finally, I would like to thank my family and my wife for their continuous support and encouragement.

This work has been sponsored by the Wolfgang Gentner Programme of the German Federal Ministry of Education and Research (grant no. 13E18CHA)

Appendices

A. Thermal study

A.1. Introduction

This study builds on previous laboratory tests, which confirmed the need for the EMP components, particularly through the FireFly transmitter heatsink, to maintain the operating temperature below 50°C , as recommended by the manufacturer. Therefore, it was essential to evaluate the thermal conditions of all devices within a fully equipped EMP crate, operated under various use cases and rack ventilation configurations. These studies will help us determine the correct number of spare critical components required for the experiment's lifetime.

A.2. Requirements for continuous FireFly Operation

The EMP FireFly modules are equipped with a “Standard Pin-Fin” heatsink, as shown in Figure A.1, designed to provide low thermal resistance from the base to the fins, with omnidirectional airflow. To prevent premature wear, it is recommended to keep the “Case Temperature”, the internal, monitorable temperature of the heatsink, at 50°C or lower. Generally, this ‘case’ must be maintained below 70°C . This

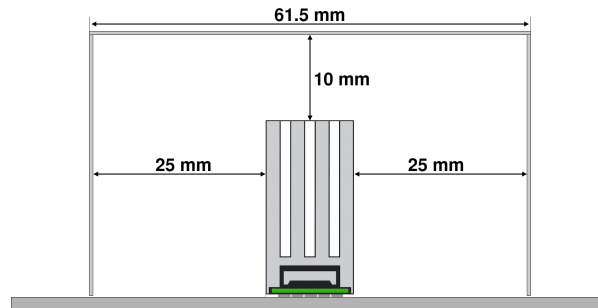


Figure A.1: *Dimensions for Standard Pin-fin Heatsink.*

is especially important for the transmitter module based on the Vertical-Cavity Surface-Emitting Laser (VCSEL) array, as its performance and overall lifespan are significantly influenced by its operating temperature. The temperature values for the 12-channel Rx and TX modules can be read from the module's memory map via the TWI interface [70].

A.3. Test Setup and Test Environment

A.3.1. Production Rack

The EMP crate was used in a production rack in the SDX1 rack room (see Figure A.2). This rack is equipped with a backdoor cooler for horizontal ventilation with air forced from the front to the back of the rack. Note that the rack was adequately covered with dummy front panels in all empty slots, ensuring that air propagates only through the rack equipment. The production rack was in full operation, with multiple computers, yielding a total continuous power consumption of around 1.2 kW, representing a realistic environment for a rack hosting DCS equipment powered by the UPS distribution. This rack configuration (power, airflow, and ventilation) is standard for DCS equipment racks in USA15 and US15, which are the underground service caverns for the ATLAS experiment, where EMP units are planned to be deployed across all use cases.



Figure A.2: DCS rack in SDX1 Level 2 with deployed EMP test crate (below the three large empty white cover panels)

Further, it should be noted that the SDX1 rack room has an environment temperature comparable to that of USA15/US15, except that it rises by a few degrees during physics operation with stable LHC beams due to elevated High Level Trigger (HLT) farm heat dissipation.

A.3.2. EMP Chassis

The EMP crate is a standard 1U-size crate, which can house up to three EMP baseboards. The crate is powered by using a commercial redundant power supply with two mains inputs, which can be inserted in a reserved space in the crate. Figure A.3 shows a schematic layout of the crate with the arrangement of the three EMP's and the Power Supply.

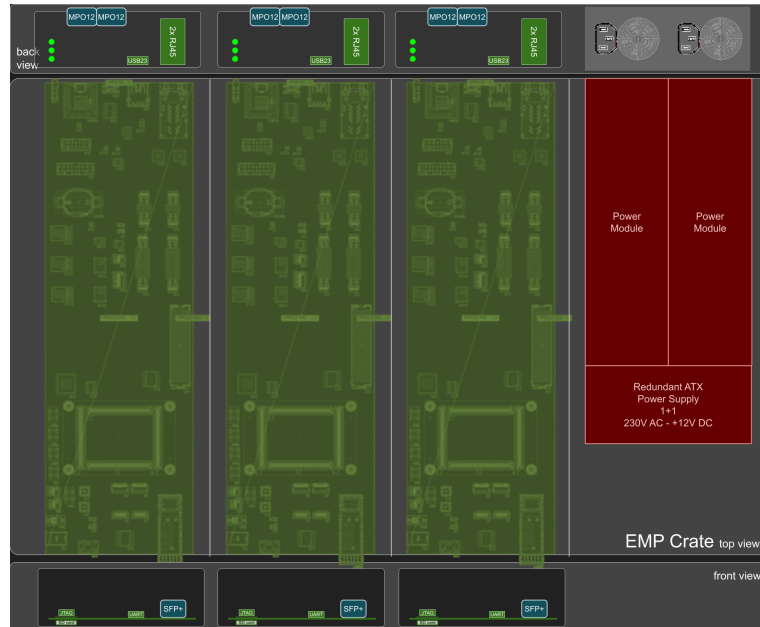


Figure A.3: *EMP crate layout.*

For the thermal study, one crate was fully populated with three EMP baseboards, each equipped with a SoM and a pair of FireFly modules (TX/RX modules) (see Figure A.4). Further, the setup includes a 400 W redundant 1 HU server power supply [102].

The orientation of the EMP baseboards is aligned to the forced airflow, as the air drawn in by the horizontal ventilation passes the board in a longitudinal direction from the front to the back of the crate. The inlet air first passes through the FireFly modules and then through the SoM, between its heatsink crate and chassis

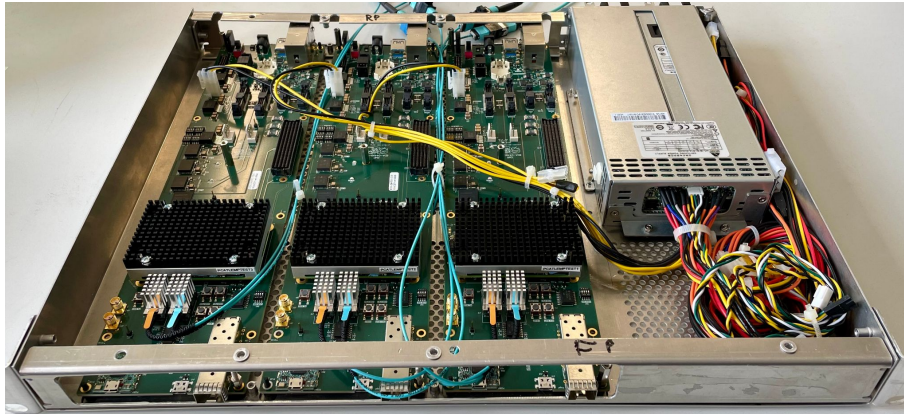


Figure A.4: Fully assembled prototype EMP crate v1.

top plate, and on both sides of the SoM. Note that compared to the initial crate layout, the baseboards were rotated 180° to achieve precisely this effect (incoming air first passing the FireFly modules). The original rotation resulted in the FireFly modules being completely isolated from the airflow, as the SoM covered them. As a result, the airflow around the FireFly modules stagnated, and the heat could not be dissipated.

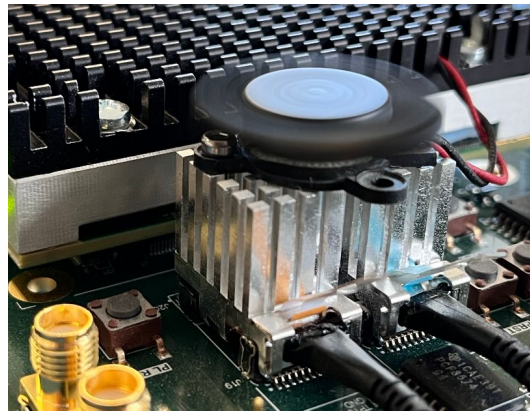


Figure A.5: FireFly modules with heatsink fan.

After initial tests with the setup described above, an additional change was introduced to ensure that the Firefly ventilation would also be sufficient for lab or system test setups where no production-level air cooling is available. As shown in Figure A.5, a small fan was installed on each pair of FireFly modules. This fan is powered by a 12 V source directly from the EMP baseboard and generates a vertically directed airflow from the top of the Firefly heatsink towards the base-

board. This setup has already been tested in a laboratory setup before being used in a production-like environment. The laboratory test has shown that the fan can provide the necessary cooling capacity to keep both FireFly modules below the 50 °C threshold at an ambient temperature of approximately 23 °C without additional vertical airflow. Note that the extra fan is not considered a critical element of the airflow for the production environment, as the backdoor coolers provide proper forced airflow. Thus, in production, if any of these fans fail, the Firefly would still operate under appropriate conditions (see Sec. A.6).

Further improvements/changes introduced to improve ventilation conditions:

- Added vertical separation between power supply and EMP baseboards
- Added optional separation between EMP baseboards
- Optimization of the perforation scheme on the upper and lower panels
- Added fixation holes on front panels for optional active horizontal ventilation

These changes were considered in the design of crate version 2; see Figure 5.13.

A.4. Methods of Temperature Measurement

To record the temperatures of each of the three EMP baseboards during the thermal study, a Python-based logger was used. This logger was ensured to be active throughout the study, and that all sensors were read every 30 s. The data were then saved to a CSV file for analysis. The following sensors have been included:

- On-board 12 bit I2C temperature sensor between FireFly modules (MAX1827)
- On-board 12 bit I2C temperature sensor between DC/DC converter (MAX1827)
- On-board 12 bit I2C temperature sensor below SoM (MAX1827)
- On-chip temperature sensor on FireFly transmitter module
- On-chip temperature sensor on FireFly receiver module
- Xilinx SYSMON FPD temperature sensor
- Xilinx SYSMON LPD temperature sensor
- Xilinx SYSMON PL temperature sensor

To correlate the measured temperatures with the ambient conditions of the rack area, the “Ambient” temperature and the “Rack” temperature, data from the Common Infrastructure Controls (CIC) DCS system were used for the analysis.

The “Ambient” temperature is measured by a sensor attached to the wall of the rack room. Therefore, although the absolute measured value cannot be directly correlated with the data measured in the crate, the relative deviations can be used to determine the activity of the HLT computing farm and its effect on the EMP baseboards.

The “Rack” temperature refers to a sensor mounted in front of the rack’s heat exchanger fins and measures the air temperature before it passes through the backdoor heat exchanger and is released into the room. The measured temperature can be used indirectly, as a function of the “Ambient” temperature, to determine the dissipated heat, i.e., the energy dissipated by the airflow from the components in the rack. It should be noted that, in addition to the EMP crate, other devices are housed in the rack, and thermal contact between the sensor and the heat exchanger fins cannot be ruled out.

A.5. Module Utilization

During the test, each SoM was subjected to a specific load on the PS and PL to generate power consumption comparable to that of a later production load. The aim was also to ensure that the laser drivers of the FireFly modules remain continuously busy and that the status of the links can be tested remotely at any time. The utilization of the PS and PL is briefly described below.

A.5.1. PL Utilization

The IBERT IP-core for GTH transceivers on UltraScale+ architectures was implemented on the PL. This core is an evaluation and demonstration platform for the transceivers and offers configurations for different line rates, reference clock rates, and logic widths. In addition, it contains data pattern generators and checkers that allow the transmission of Pseudo-Random Binary Sequence (PRBS) via each channel.

The IBERT IP-core has been implemented along with the tools required to access its debug core via the Xilinx Virtual Cable (XVC) protocol. This enables access to the debug core via a TCP/IP-based protocol that behaves like a JTAG adapter, allowing remote access to the SoM and thus enabling checks of the Bit Error Rate (BER) and the connection status. The power consumption of an EMP with the IBERT

IP-core was evaluated using a standard laboratory power supply, with a measured value of 18 W. For comparison, the same setup was employed for the standard EMP firmware, with the EMP connected to 12 EMCIs. This resulted in approximately 17.6 W of consumption, indicating that the IBERT IP-core can be considered a comparable load to future production applications.

A.5.2. PS Utilization

The 4-core Cortex-A53 processor of the MPSoC was stressed throughout the entire test. For this purpose, the Linux command-line tool “stress” was used, which can load each core heavily. In addition to the temperatures, the CPU load was permanently logged and remained at 100 % on each core.

A.5.3. Power Consumption

The estimated power consumption of a fully equipped crate is approximately 74 W. Of this, 18 W are attributed to the operating load of the power supply, as measured by running the power supply without a load, while the remaining 56 W are distributed among the three EMPs.

A.6. Results of the Thermal Tests

The results of the thermal tests are discussed in the following section. The tests ran for one month and comprised over 120,000 data sets containing the measured values from all the sensors listed in the section A.4. In addition, data from rack and room monitoring via the CIC DCS system have been used to investigate temperature changes on the boards as a function of ambient temperature.

The test can be divided into two different series, with and without the additional fan on the FireFly heatsink. All test series are discussed below, using data recorded on one board as an example. The remaining data for the other two boards are available here.

Test without additional FireFly fan

The first scenario was a test without the additional fan on the heatsink of the FireFly modules. The test was conducted during a period with no stable beams, resulting in relatively constant ambient and rack temperatures because the computer farm was

not loaded. The test ran for about 20 hours and yielded transmitter temperatures of 36 °C to 37 °C and receiver temperatures of about 42 °C. The SoM temperature was also below 50 °C throughout the test. The data from this test are shown in Figure A.6.

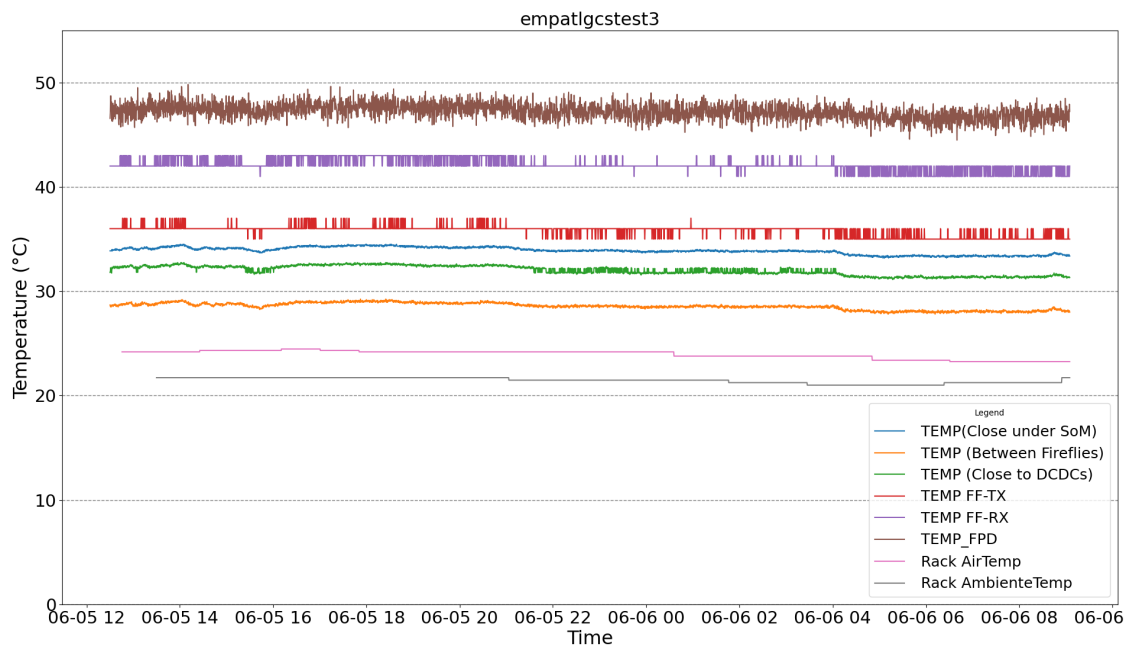


Figure A.6: *Temperature evolution without an equipped FireFly fan.*

Test with additional Firefly fan

The second series of tests was carried out with the additional fan on the heatsink of the Firefly module. The tests ran for approximately one month, recording periods when the computing farm was inactive, as in the previous test, and periods when the farm was active during stable beam operation.

As already mentioned, the room monitoring data was also used for a more detailed analysis. This allows the measured temperatures on the EMP baseboards to be analyzed as a function of the room and rack ambient temperature. While the HLT computing farm is loaded, the temperature in the room rises by approximately 2 °C to 3 °C, while the temperature in the rack increases by approximately 4 °C to 5 °C. If, on the other hand, the farm is not active, an almost stable room temperature of approximately 20 °C and a stable rack temperature of approximately 23 °C can be observed.

The following diagrams show how the temperatures at the EMP baseboards behave in both cases. It should be noted that in all cases the critical threshold value of the Firefly module was not exceeded. Therefore, it can be concluded that ventilation provides the required cooling capacity under the given circumstances, with a margin well above 10 °C for the TX FireFly temperature.

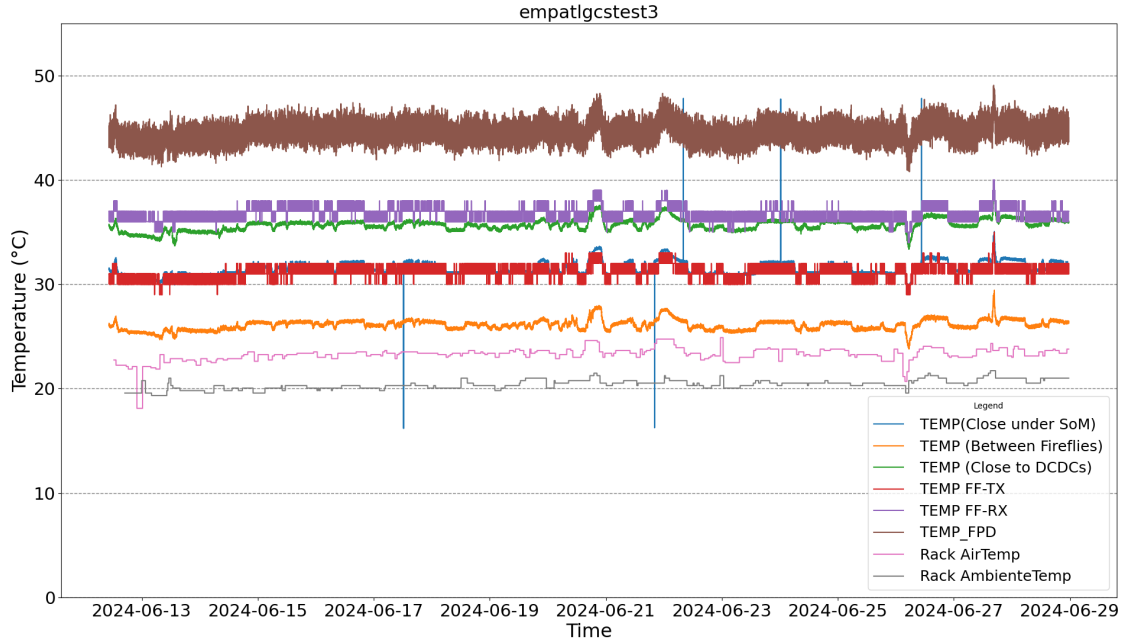


Figure A.7: Temperature evolution with additional FireFly fan during a period without stable beams.

Figure A.7 shows the behavior when the computing farm is not active, and as can be seen, the room and rack temperatures are only subject to minimal fluctuations. This is also reflected by the data from the EMP baseboard. Overall, the temperature appears constant, as reflected in the data. The temperature of the FireFly modules is also clearly below the threshold values: 32 °C for the TX module and 37 °C for the RX module. Figures A.6 and A.7 show a direct comparison of the setup with and without additional FireFly heatsink fans, respectively. The setup with the additional ventilation is approximately 5 °C to 6 °C lower per module than without the fan.

In contrast, Figure A.8 shows the recorded data while the computing farm is regularly active during LHC fills. Here, you can clearly see the correlation between the temperatures measured on the EMP and the room-monitoring temperatures. During these periods, the room monitoring temperatures rise from 20 °C to 23 °C and the rack monitoring temperatures rise from 22.5 °C to 27.5 °C (note that the rack heat

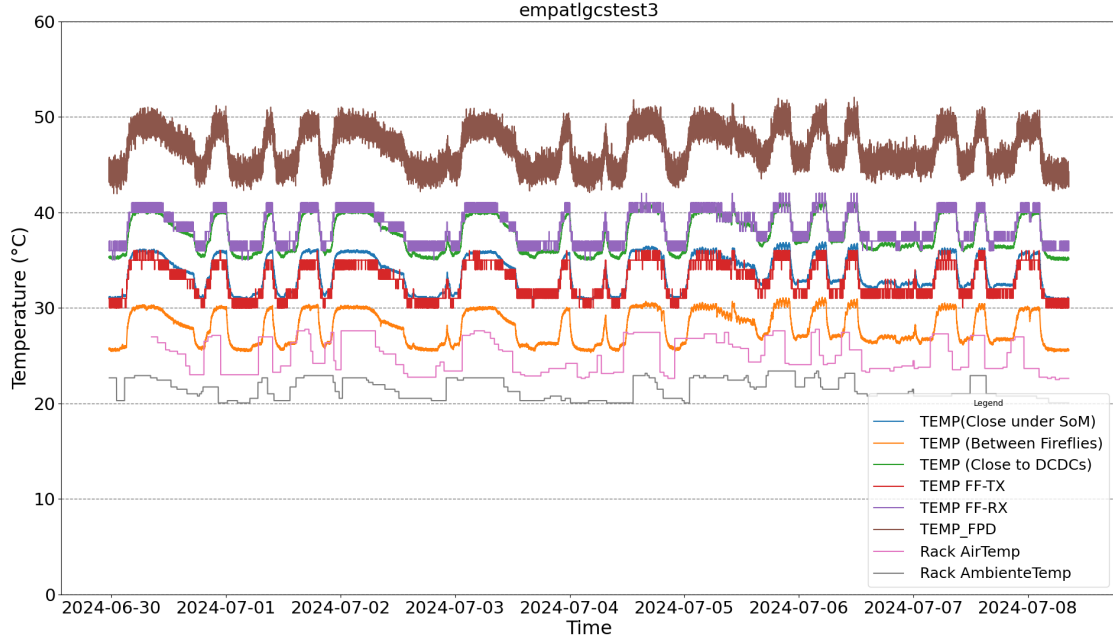


Figure A.8: Temperature evolution with equipped FireFly fans during a period with regular stable beams (computing farm active).

dissipation is largely dominated by the computers as mentioned above, not the EMP crate). The temperatures measured on the EMP follow the changes with a maximum temperature difference of 6 °C to 7 °C on the SoM. However, all temperatures are well within the required range for continuous operation.

Open rack door

Furthermore, to simulate intermittent backdoor ventilation failure, the rack door was opened for approximately 20 hours. The data recorded during this period are shown in Figure A.9. It should be noted that the critical threshold temperature for the transmitter module is not exceeded and remains below 50 °C, owing to the additional fan on the FireFly heatsink. At just above 52 °C, the temperature of the receiver module is slightly higher than the specified threshold value. Still, no increased aging effect is expected for this module, except that a failure of the backdoor cooler in production can usually be rectified within hours. It should be further noted that this scenario of a backdoor failure is an exceptional case that occurred in the past, much less than once per 5 years, averaged over all racks equipped with backdoor coolers, and usually only one of the three fans within the door failed while the others continued operating.

Last but not least, this scenario also indicates that the EMP crate can operate well under degraded cooling conditions in USA15. In such situations the USA15/US15 room temperature will increase by an absolute maximum of 10 °C (usually around 10 °C) but the backdoor fans will remain active even if the rack cooling water stops circulating and thus forced airflow will still be present in the rack which can limit the EMP temperature increase to less than the observed maximum of approximately 20 °C for the SoM and approximately 15 °C for the Firefly TX/RX modules.

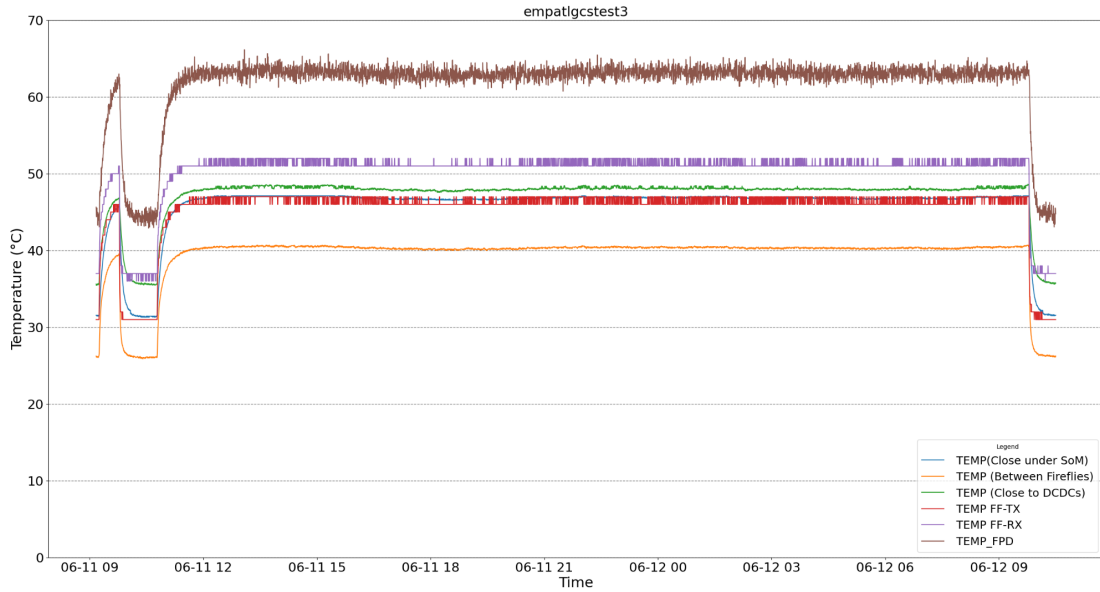


Figure A.9: Temperature evolution with equipped FireFly fan during the simulation of an intermittent failure of the rack door ventilation by opening the rack door. The figure shows two periods of increased temperature: the first, an initial test of approximately 1 hour, and the second, a test of approximately 20 hours. The figure also shows that the temperature returns to normal within less than 2 minutes after the rack door is closed.

In addition, for situations in which all cooling mechanisms would fail or be exceptionally unavailable, a software-based monitoring service within the EMP PS was developed as a systemd service ([91]) to query the temperatures of the FireFly modules at regular intervals. If a module's temperature consistently exceeds a defined threshold of 70 °C over 4 minutes, the service initiates a shutdown sequence for the affected FireFly modules. This process involves disabling all optical channels of both modules, deactivating the RX module output signal, and setting the RX output amplitude to zero. This brings their temperatures back within acceptable limits. While in this state, the FireFly modules are inoperable and cannot be used for communication. Reactivation of the modules requires user intervention.

Note that the current implementation of the protection mechanism is intended for integration setups and laboratory testing. It will later be incorporated into the production software — the EMP Monitoring & Control OPC UA Server — so users can activate or deactivate it. Nonetheless, the protection remains essential to prevent permanent damage and the consequent incapacitation of the EMP function.

A.6.1. Conclusions

The thermal study conducted for the EMP in a production-like environment shows that the system operates well within the required temperature ranges, with no expected negative impact on lifetime. The different test scenarios provide key insights into the thermal behavior of the setup under various environmental and operational conditions.

- Without equipped FireFly fan: the baseline test, conducted without additional FireFly ventilation and thus only passively cooled EMPs, showed that the EMP crate's FireFly modules maintain stable temperatures during the recorded periods. The measured transmitter module temperatures are approximately 36 °C to 37 °C, and the receiver module is approximately 42 °C. These values are well below the recommended maximum operation temperature of 50 °C, confirming the effectiveness of the existing cooling solution under stationary conditions. Note that for a TX-12 FireFly module heatsink temperature of 40 °C, the expected cumulative failure rate is below 0.01 % for the expected lifetime of 15 years for an EMP device [EMP specification].
- With equipped FireFly fan: Test with the equipped additional FireFly heatsink fan resulted in a further reduction of module temperatures by approximately 5 °C to 6 °C. Even under demanding conditions, when the computing farm was active during stable beam phases, the FireFly modules were operated well below the required limit of 50 °C.
- Simulated cooling failures: A stress test simulating a rack ventilation failure by opening the rack door demonstrated that even under adverse conditions, the EMP crate continued to operate without exceeding the recommended temperatures. The transmitter module remained below 50 °C, while the receiver module reached around 52 °C, which is considered not to impact the module's lifetime. This indicates that the EMP crate can also be operated safely in the event of temporary cooling failures, consistent with recommendations for scenarios with reduced cooling.

In summary, all tests, with and without the FireFly heatsink fan, as well as the simulated rack ventilation failure, showed that the EMP can be operated safely

within the recommended temperature ranges. This validates the horizontal rack ventilation as sufficient for the EMP use case, ensuring that the system can also withstand infrequent cooling system failures without compromising the lifetime of the optical transmitters.

A.6.2. Re-validation with Pre-Production EMP and Crate

Following the pre-production of EMP (version 3) and the EMP crate (version 2), the thermal study was repeated in the same rack in SDX1. Note that the crate version 2 hosts optional chassis fans, one per EMP (see also Fig. A.10), which were used instead of the previously used FireFly heatsink fan.

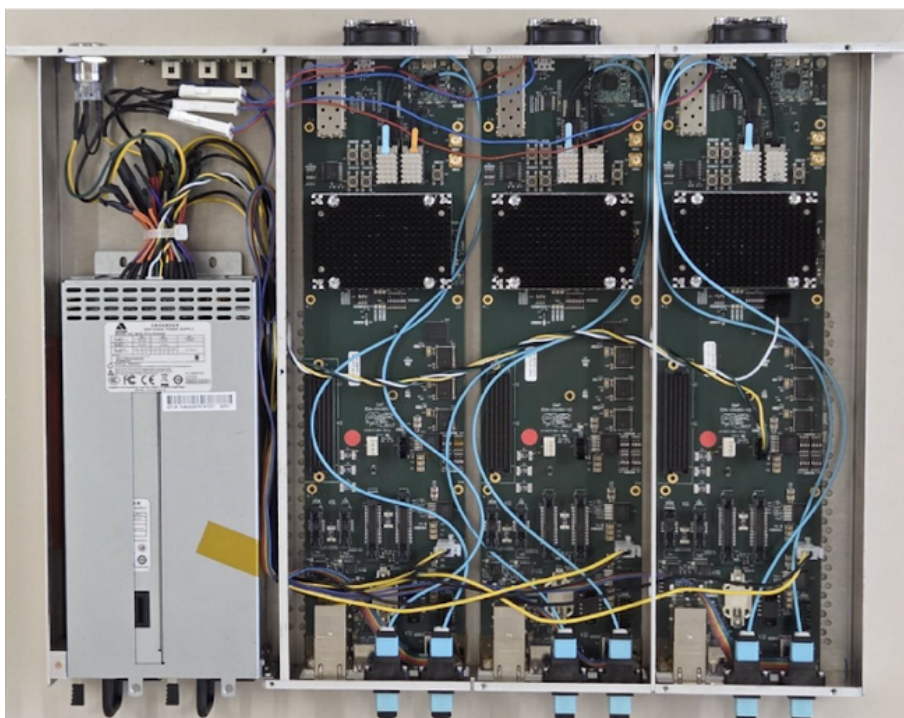


Figure A.10: EMP crate v2 fully equipped. The chassis fans (top) serve as a replacement for the FireFly heatsink fans.

These tests confirmed the same thermal performance of the crate, with temperature variation of less than 2 °C relative to each of the scenarios discussed in the previous sections.

B. quasar Design Diagram

B.1. EMP Monitoring & Control OPC UA Server

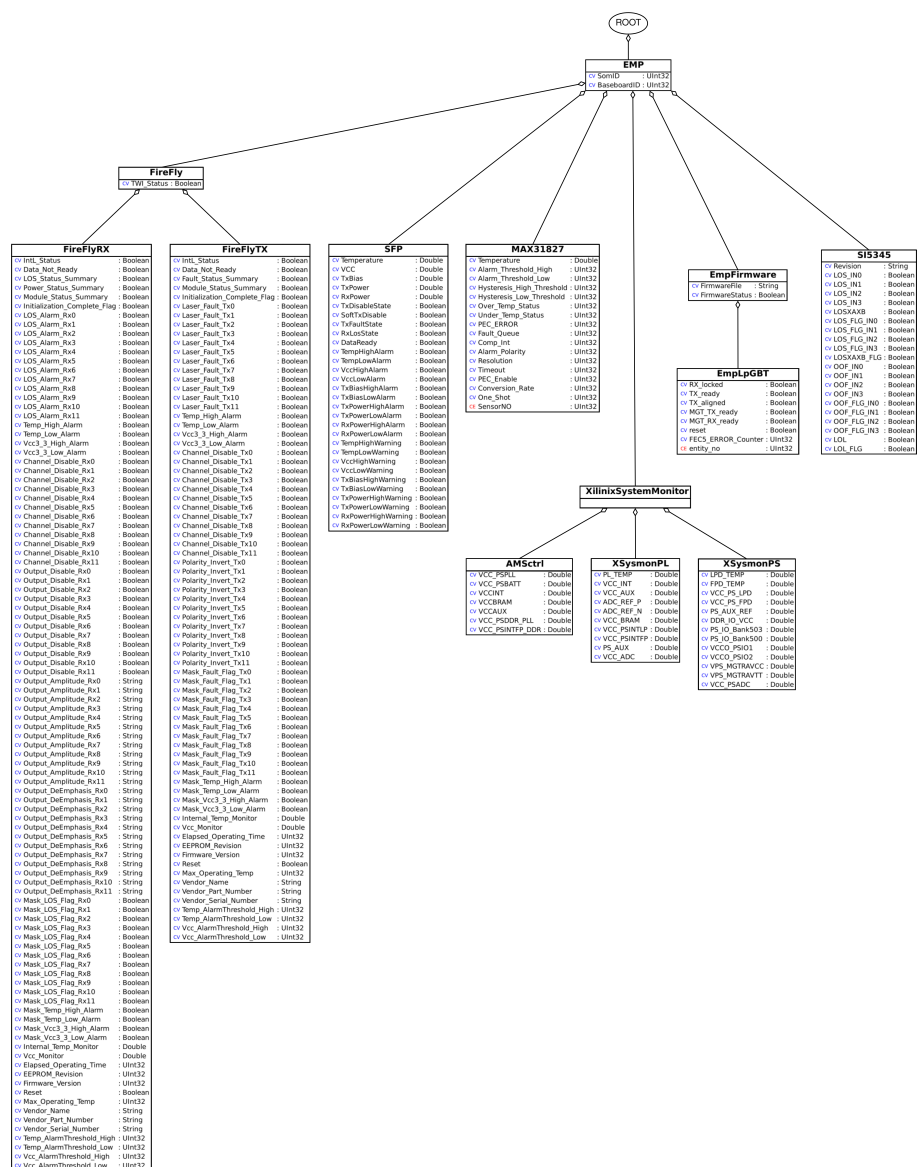


Figure B.1: *OPC UA information model for the EMP monitoring server.*

C. SC-DMA IP Core - Finite-State Machines

C.1. Watchdog FSM

```
RD_STS <= std_logic_vector(to_unsigned(RD_STS'pos(READ_ENGINE), 3));
WR_STS <= std_logic_vector(to_unsigned(WR_STS'pos(WRITE_ENGINE), 3));

RST_GLOBAL <= RST or not AXI_ARESETN or RST_WD;
RST40 <= RST_GLOBAL_SYNC or RST_WD_SYNC;

AXI_WATCHDOG_FSM: process(AXI_ACLK, RST)
begin
    if rising_edge(AXI_ACLK) then
        if RST_GLOBAL = '1' then
            TX_TO_CNT <= (others => '0');
            RX_TO_CNT <= (others => '0');
            IRQ_DLY_CNT <= (others => '0');
            RST_ACT_CNT <= (others => '0');
            RST_WD <= '0';
            IRQ <= '0';
            WATCHDOG <= IDLE;
        else
            case WATCHDOG is
                -----
                -- WATCHDOG: IDLE --
                -----
            when IDLE =>
                TX_TO_CNT <= (others => '0');
                RX_TO_CNT <= (others => '0');
                IRQ_DLY_CNT <= (others => '0');
                RST_ACT_CNT <= (others => '0');
                IRQ <= '0';

                if READ_ENGINE = IDLE and WRITE_ENGINE = IDLE then
                    BUSY <= '0';
                end if;

                if START = '1' and BUSY = '0' then
                    WATCHDOG <= DMA_BUSY;
```

```
BUSY <= '1';
end if;

-----
-- WATCHDOG: DMA BUSY --
-----

when DMA_BUSY =>
  if READ_ENGINE /= FINISHED then
    if TX_TO_CNT < TX_TIMEOUT_LIMIT then
      TX_TO_CNT <= TX_TO_CNT + 1;
    else
      TIMEOUT(0) <= '1';
    end if;
  end if;

  if WRITE_ENGINE /= IDLE then
    if RX_TO_CNT < RX_TIMEOUT_LIMIT then
      RX_TO_CNT <= RX_TO_CNT + 1;
    else
      TIMEOUT(1) <= '1';
    end if;
  end if;

  if (READ_ENGINE = FINISHED and WRITE_ENGINE = FINISHED) or TIMEOUT
    /= "00" then
    READ_STATUS <= RD_STS;
    WRITE_STATUS <= WR_STS;
    WATCHDOG <= WAIT_ACK;
  end if;

  -----
  -- WATCHDOG: WAIT ACK --
  -----

when WAIT_ACK =>
  IRQ <= '1';
  IRQ_DLY_CNT <= IRQ_DLY_CNT + 1;

  if ACK = '1' then
    IRQ <= '0';
    IRQ_DELAY <= std_logic_vector(IRQ_DLY_CNT);

    if TIMEOUT /= "00" then
      RECOVER <= '1';
    end if;
  end if;
```

```
if STATUS_STROBE = '1' then
    RECOVER <= '0';
    READ_STATUS <= (others => '0');
    WRITE_STATUS <= (others => '0');
    TIMEOUT <= (others => '0');
    IRQ_DELAY <= (others => '0');

    if RECOVER = '1' then
        RST_WD <= '1';
        WATCHDOG <= RESET;
    else
        WATCHDOG <= IDLE;
    end if;
end if;

-----
-- WATCHDOG: RESET --
-----

when RESET =>
    RST_ACT_CNT <= RST_ACT_CNT + 1;

    if RST_ACT_CNT >= RESET_CYCLES then
        WATCHDOG <= IDLE;
    elsif RST_ACT_CNT >= RESET_CYCLES/2 then
        RST_WD <= '0';
    end if;
end case;
end if;
end if;
end process;
```

C.2. AXI4-Read Engine FSM

```
AXI4_READ_FSM: process(AXI_ACLK, RST_GLOBAL)
begin
  if rising_edge(AXI_ACLK) then
    if RST_GLOBAL = '1' or RST_WD = '1' then
      READ_ENGINE <= IDLE;
      ARVALID <= '0';
      RREADY <= '0';
      READ_RESP <= "00";
    else
      case READ_ENGINE is
        -----
        -- AXI4-READ ENGINE: IDLE --
        -----
        when IDLE =>
          if STATUS_STROBE = '1' then
            READ_RESP <= "00";
          end if;

          if START = '1' and BUSY = '0' then
            ARADDR <= SRC_ADDR;
            ARLEN <= std_logic_vector(to_unsigned(DL_WORDCNT-1, 8));
            ARVALID <= '1';

            if arready = '1' then
              RREADY <= '1';
              READ_ENGINE <= READ;
            else
              READ_ENGINE <= HANDSHAKE;
            end if;
          end if;
        -----
        -- AXI4-READ ENGINE: HANDSHAKE --
        -----
        when HANDSHAKE =>
          if ARREADY = '1' then
            ARVALID <= '0';
            RREADY <= '1';
            READ_ENGINE <= READ;
          end if;
        -----
        -- AXI4-READ ENGINE: READ --
        -----
```

```
when READ =>
  COLLECTOR_ENABLE <= '1';
  ARVALID <= '0';

  if RVALID = '1' and RRESP /= "00" then
    READ_RESP <= RRESP;
  end if;

  if RLAST = '1' then
    COLLECTOR_ENABLE <= '0';
    RREADY <= '0';
    READ_ENGINE <= FINISHED;
  end if;

  -----
  -- AXI4-READ ENGINE: FINISHED --
  -----

when FINISHED =>
  if WATCHDOG = WAIT_ACK then
    READ_ENGINE <= IDLE;
  end if;
end case;
end if;
end if;
end process;
```

C.3. Downlink Frame Assembler

```
frameAssembler: process(AXI_ACLK, RST100)
    variable link : lane_info_t;
    variable linkMsb : integer range 0 to 7;
    variable linkgrp : integer range 0 to 3;
    variable linkWidth : LaneWidth;
    variable linkIndex : integer range 0 to 15;
    variable linkFrameMsb : integer range 0 to 31;
begin
    if rising_edge(AXI_ACLK) then
        if RST100 = '1' then
            ASSEMBLER <= IDLE;
            SEND_EN <= '0';
            IDX <= 0;
            CH_NUMBER <= 0;
            POINTER <= (others => 127);
        else
            case ASSEMBLER is
                -----
                -- FRAME ASSEMBLER: IDLE --
                -----
                when IDLE =>
                    SEND_EN <= '0';
                    IDX <= 0;
                    CH_NUMBER <= 0;
                    POINTER <= (others => 127);

                    if FIFO_EMPTY /= '1' then
                        SEND_EN <= '1';
                    end if;

                    if RREADY = '1' then
                        ASSEMBLER <= CAPTURE;
                    end if;
                -----
                -- FRAME ASSEMBLER: CAPTURE --
                -----
                when CAPTURE =>
                    if RVALID = '1' and RID(2 downto 0) = ID then
                        BUF(CH_NUMBER) <= ELINK_DIN;
                        CH_NUMBER <= CH_NUMBER + 1;
                        ASSEMBLER <= SORT when RLAST = '1' else CAPTURE;
                    end if;
```

```

-----
-- FRAME ASSEMBLER: SORT --
-----

when SORT =>
  if IDX < ACTIVE_LANE_INFO'length then
    linkGrp := ACTIVE_LANE_INFO(IDX).grp;
    linkIndex := ACTIVE_LANE_INFO(IDX).IDX;
    linkMsb := ACTIVE_LANE_INFO(IDX).msb;
    linkFrameMsb := (3 - linkGrp) * 8 + linkMsb;
    linkWidth := ACTIVE_LANE_INFO(IDX).bw;

    if POINTER(IDX) /= 0 then
      FIFO_WR_PENDING <= '1';

      case linkWidth is
        when W8 =>
          lpGBT_FRAME(linkFrameMsb downto linkFrameMsb-7) <=
            BUF(linkIndex)(POINTER(IDX) downto POINTER(IDX)-7);
          POINTER(IDX) <= POINTER(IDX) - 8 when POINTER(IDX) > 7 else 0;
        when W4 =>
          lpGBT_FRAME(linkFrameMsb downto linkFrameMsb-3) <=
            BUF(linkIndex)(POINTER(IDX) downto POINTER(IDX)-3);
          POINTER(IDX) <= POINTER(IDX) - 4 when POINTER(IDX) > 3 else 0;
        when W2 =>
          lpGBT_FRAME(linkFrameMsb downto linkFrameMsb-1) <=
            BUF(linkIndex)(POINTER(IDX) downto POINTER(IDX)-1);
          POINTER(IDX) <= POINTER(IDX) - 2 when POINTER(IDX) > 1 else 0;
        when others =>
          null;
      end case;
    end if;
    IDX <= IDX + 1;
  else
    ASSEMBLER <= WRITE;
  end if;

-----
-- FRAME ASSEMBLER: WRITE --
-----

when WRITE =>
  if FIFO_WR_PENDING = '1' then
    FIFO_WR_EN <= '1';
    FIFO_DIN <= lpGBT_FRAME;
    FIFO_WR_PENDING <= '0';
  else

```



```
FIFO_WR_EN <= '0';
FIFO_DIN <= (others => '0');
end if;

if FIFO_WR_ACK = '1' then
    IDX <= 0;
    lpGBT_FRAME <= (others => '0');
    ASSEMBLER <= IDLE when all_zero(POINTER) = true else SORT;
end if;
end case;
end if;
end if;
end process;

fifoDrain: process(CLK40, RST40)
begin
    if rising_edge(CLK40) then
        if RST40 = '1' then
            FIFO_RD_EN <= '0';
        elsif SEND_EN_SYNC = '1' then
            FIFO_RD_EN <= '1' when FIFO_EMPTY /= '1' else '0';
        end if;
        USERDATA_OUT <= FIFO_DOUT when FIFO_RD_VALID = '1' else (others =>
            '0');
        end if;
    end process;
```

C.4. AXI4-Write Engine FSM

```

AXI4_WRITE_FSM: process(AXI_ACLK, RST_GLOBAL)
    variable fifoCnt : integer range 0 to 64 := 0;
begin
    if rising_edge(AXI_ACLK) then
        if RST_GLOBAL = '1' or RST_WD = '1' then
            fifoCnt := 0;
            AWVALID <= '0';
            WVALID <= '0';
            WLAST <= '0';
            BREADY <= '0';
            WRITE_RESP <= "00";
            WRITE_ENGINE <= IDLE;
        else
            case WRITE_ENGINE is
                -----
                -- AXI4-WRITE ENGINE: IDLE --
                -----

            when IDLE =>
                if STATUS_STROBE = '1' then
                    WRITE_RESP <= "00";
                end if;

                AWADDR <= DST_ADDR;
                fifoCnt := to_integer(unsigned(FIFO_CNT_DOUT)) * 2;

                if COLECTOR_DONE = '1' and FIFO_RD_VALID = '1' then
                    AWLEN <= std_logic_vector(to_unsigned(fifoCnt - 1, 8));
                    AWVALID <= '1';
                   WSTRB <= (others => '1');
                    BREADY <= '1';
                    HALF_DONE <= '1';
                    WDATA <= FIFO_DOUT(127 downto 0);
                    BUF <= FIFO_DOUT(255 downto 128);

                    if AWREADY = '1' then
                        FIFO_RD_EN <= '1';
                        WVALID <= '1';
                        WRITE_ENGINE <= WRITE;
                    else
                        WRITE_ENGINE <= HANDSHAKE;
                    end if;
                end if;
            end if;
        end if;
    end if;
end if;

```

```

-----
-- AXI4-WRITE ENGINE: HANDSHAKE --
-----

when HANDSHAKE =>
  if AWREADY = '1' then
    FIFO_RD_EN <= '1';
    WVALID <= '1';
    AWVALID <= '0';
    WRITE_ENGINE <= WRITE;
  end if;

-----
-- AXI4-WRITE ENGINE: WRITE --
-----

when WRITE =>
  WSTRB <= (others => '1');
  AWVALID <= '0';
  WVALID <= '1';
  FIFO_RD_EN <= '0';

  if WVALID = '1' and WREADY = '1' and HALF_DONE = '1' then
    fifoCnt := fifoCnt - 1;
    WDATA <= BUF;
    HALF_DONE <= '0';

    if fifoCnt = 1 then
      WLAST <= '1';
      WRITE_ENGINE <= RESPONSE;
    end if;

  elsif WVALID = '1' and WREADY = '1' and HALF_DONE = '0' then
    fifoCnt := fifoCnt - 1;
    HALF_DONE <= '1';
    FIFO_RD_EN <= '1';

    if FIFO_RD_VALID = '1' then
      WDATA <= FIFO_DOUT(127 downto 0);
      BUF <= FIFO_DOUT(255 downto 128);
    end if;
  end if;

-----
-- AXI4-WRITE ENGINE: RESPONSE --
-----

when RESPONSE =>
  if WVALID = '1' and WREADY = '1' then

```

```
        fifoCnt := fifoCnt - 1;
        WLAST <= '0';
        WVALID <= '0';
       WSTRB <= (others => '0');
        BREADY <= '1';
    end if;

    if BVALID = '1' and BRESP = "00" then
        BREADY <= '0';
        WRITE_RESP <= BRESP;
        WRITE_ENGINE <= FINISHED;
    end if;

    -----
    -- AXI4-WRITE ENGINE: FINISHED --
    -----

    when FINISHED =>
        if WATCHDOG = WAIT_ACK then
            fifoCnt := 0;
            WRITE_ENGINE <= IDLE;
        end if;
    end case;
end if;
end if;
end process;
```

C.5. Uplink Frame Collector

```
timeoutCTRL: process(CLK40, RST)
begin
  if rising_edge(CLK40) then
    if RST = '1' then
      GLOBAL_TO_CNT <= (others => '0');
      TO_REACHED <= '0';
    elsif COLLECTOR = COLLECT then
      if GLOBAL_TO_CNT >= TIMEOUT_LIMIT then
        TO_REACHED <= '1';
      else
        GLOBAL_TO_CNT <= GLOBAL_TO_CNT + 1;
      end if;
    else
      GLOBAL_TO_CNT <= (others => '0');
      TO_REACHED <= '0';
    end if;
  end if;
end process;

frameCollector: process(CLK40, RST)
  variable link : lane_info_t;
  variable idx : integer range 0 to 27 := 0;
  variable linkBw : LaneWidth := W0;
  variable linkIdx : integer range 0 to 27 := 0;
  variable grpWord : std_logic_vector(31 downto 0) := (others => '0');
  variable chunk16 : std_logic_vector(15 downto 0) := (others => '0');
  variable chunk8 : std_logic_vector(7 downto 0) := (others => '0');
begin
  if rising_edge(CLK40) then
    if RST = '1' then
      LANE_STS <= (others => LS_IDLE);
      POINTER <= (others => 0);
      FIFO_WR_CNT <= 0;
      FIFO_WR_EN <= '0';
      FIFO_DIN <= (others => '0');
      COLLECTOR <= IDLE;
    else
      case COLLECTOR is
        -----
        -- FRAME COLLECTOR: IDLE --
        -----
        when IDLE =>
```

```
DONE <= '0';
FIFO_WR_CNT <= 0;
POINTER <= (others => 0);
LANE_STS <= (others => LS_IDLE);
LANE_RES <= (others => GOOD);

if ENABLE = '1' then
    COLLECTOR <= COLLECT;
end if;

-----
-- FRAME COLLECTOR: COLLECT --
-----

when COLLECT =>
    if TO_REACHED = '1' then
        for idx in ACTIVE_LANE_INFO'range loop
            if LANE_STS(idx) /= LS_FINISHED and LANE_STS(idx) /=
                LS_DEACTIVATED then
                if LANE_RES(idx) /= OVERFLOW then
                    LANE_STS(idx) <= LS_ERROR;
                    LANE_RES(idx) <= TIMEOUT;
                end if;
            end if;
        end loop;
        COLLECTOR <= FLATTEN;
    elsif all_collected(LANE_STS) then
        COLLECTOR <= FLATTEN;
    else
        for idx in ACTIVE_LANE_INFO'range loop
            link := ACTIVE_LANE_INFO(idx);
            linkIdx := link.idx;
            linkBw := link.bw;
            grpWord := frame_in(GRP_RANGES(link.grp) downto
                GRP_RANGES(link.grp)-31);

            case linkBw is
            when W32 =>
                if grpWord(31 downto 24) = START_BYTE and LANE_STS(idx) =
                    LS_IDLE then
                    BUF_32_BIT(linkIdx)(0) <= grpWord;
                    POINTER(linkIdx) <= 1;
                    LANE_STS(idx) <= LS_ACTIVE;
                elsif LANE_STS(idx) = LS_ACTIVE then
                    if POINTER(link.idx) >= DEPTH_BUF32 then
                        LANE_STS(idx) <= LS_ERROR;
```

```
    LANE_RES(idx) <= OVERFLOW;
else
    BUF_32_BIT(linkIdx)(POINTER(linkIdx)) <= grpWord;
    POINTER(linkIdx) <= POINTER(linkIdx) + 1;

    if has_end_byte(grpWord, END_BYTE) then
        LANE_STS(idx) <= LS_FINISHED;
    end if;
end if;
end if;

when W16 =>
    chunk16 := grpWord(link.msb downto link.msb-15);
    if chunk16(15 downto 8) = START_BYTE and LANE_STS(idx) =
        LS_IDLE then
        BUF_16_BIT(linkIdx)(0) <= chunk16;
        POINTER(linkIdx) <= 1;
        LANE_STS(idx) <= LS_ACTIVE;
    elsif LANE_STS(idx) = LS_ACTIVE then
        if POINTER(link.idx) >= DEPTH_BUF16 then
            LANE_STS(idx) <= LS_ERROR;
            LANE_RES(idx) <= OVERFLOW;
        else
            BUF_16_BIT(linkIdx)(POINTER(linkIdx)) <= chunk16;
            POINTER(linkIdx) <= POINTER(linkIdx) + 1;

            if has_end_byte(chunk16, END_BYTE) then
                LANE_STS(idx) <= LS_FINISHED;
            end if;
        end if;
    end if;

when W8 =>
    chunk8 := grpWord(link.msb downto link.msb-7);
    if chunk8 = START_BYTE and LANE_STS(idx) = LS_IDLE then
        BUF_8_BIT(linkIdx)(0) <= chunk8;
        POINTER(linkIdx) <= 1;
        LANE_STS(idx) <= LS_ACTIVE;
    elsif LANE_STS(idx) = LS_ACTIVE then
        if POINTER(link.idx) >= DEPTH_BUF8 then
            LANE_STS(idx) <= LS_ERROR;
            LANE_RES(idx) <= OVERFLOW;
        else
            BUF_8_BIT(linkIdx)(POINTER(linkIdx)) <= chunk8;
```

```
    POINTER(linkIdx) <= POINTER(linkIdx) + 1;

    if has_end_byte(chunk8, END_BYTE) then
        LANE_STS(idx) <= LS_FINISHED;
    end if;
end if;
end if;

when others =>
    null;
end case;
end loop;
end if;

-----
-- FRAME COLLECTOR: FLATTEN --
-----

when FLATTEN =>
    FIFO_WR_EN <= '0';
    FIFO_DIN <= (others => '0');

if FIFO_WR_CNT < LANE_STS'length then
    if LANE_STS(FIFO_WR_CNT) /= LS_DEACTIVATED then
        link := ACTIVE_LANE_INFO(FIFO_WR_CNT);
        STATUS <= encode_status(LANE_RES(FIFO_WR_CNT));
        EGROUPE <= std_logic_vector(to_unsigned(link.grp,8));
        BIT_CNT <= std_logic_vector(to_unsigned(POINTER(link.idx),8));
        ELINK <= std_logic_vector(to_unsigned(link.idx,8));

        case link.bw is
            when W32 =>
                DATA <= flatten_buffer32(BUF_32_BIT(link.idx))(223 downto 0);
                BUF_32_BIT(link.idx) <= (others => (others => '0'));
            when W16 =>
                DATA <= flatten_buffer16(BUF_16_BIT(link.idx))(223 downto 0);
                BUF_16_BIT(link.idx) <= (others => (others => '0'));
            when W8 =>
                DATA <= flatten_buffer8(BUF_8_BIT(link.idx))(223 downto 0);
                BUF_8_BIT(link.idx) <= (others => (others => '0'));
            when others => null;
        end case;

        FIFO_WR_PENDING <= '1';
        COLLECTOR <= WRITE_FIFO;
    end if;
```



```
FIFO_WR_CNT <= FIFO_WR_CNT + 1;
else
  COLLECTOR <= FINISHED;
end if;

-----
-- FRAME COLLECTOR: WRITE FIFO --
-----

when WRITE_FIFO =>
  if FIFO_WR_PENDING = '1' then
    FIFO_WR_EN <= '1';
    FIFO_DIN <= EGROUP & ELINK & STATUS & BIT_CNT & DATA;
    FIFO_WR_PENDING <= '0';
  else
    FIFO_WR_EN <= '0';
    FIFO_DIN <= (others => '0');
  end if;

  if FIFO_WR_ACK = '1' then
    COLLECTOR <= FLATTEN;
  end if;

  -----
  -- FRAME COLLECTOR: FINISHED --
  -----

when FINISHED =>
  if to_integer(unsigned(FIFO_CNT_DIN)) = FIFO_WR_CNT then
    DONE <= '1';
    COLLECTOR <= IDLE;
  end if;
end case;
end if;
end if;
end process;
```

D. SC-DMA Linux Kernel Module

D.1. Character Device Interface

D.1.1. open()

```
1 static int elink_open(struct inode *inode, struct file *file) {
2     struct dmaDevice *dma;
3     struct file_private *priv;
4     uint32_t addr_low;
5
6     // Find dmaDevice associated with this cdev
7     dma = container_of(inode->i_cdev, struct dmaDevice, cdev);
8     if (!dma)
9         return -ENODEV;
10
11     // Allocate per-open structure
12     priv = kzalloc(sizeof(*priv), GFP_KERNEL);
13     if (!priv)
14         return -ENOMEM;
15
16     priv->dev = dma;
17
18     // Allocate the coherent DMA memory per open
19     priv->buf.size = DMA_BUF_SIZE;
20     priv->buf.vir = dma_alloc_coherent(dma->dev, priv->buf.size, &
21     priv->buf.phy, GFP_KERNEL);
22     if (!priv->buf.vir) {
23         kfree(priv);
24         return -ENOMEM;
25     }
26
27     addr_low = (uint32_t)(priv->buf.phy & 0xFFFFFFFF);
28     dmaWrite(dma, DST_ADDR_REG, addr_low + RX_MM_OFFSET);
29     dmaWrite(dma, SRC_ADDR_REG, addr_low);
30     wmb();
31     file->private_data = priv;
32     return 0;
33 }
```

D.1.2. mmap()

```
1 static int elink_mmap(struct file *file, struct vm_area_struct *vma
    ) {
2     struct file_private *priv = file->private_data;
3     size_t size = vma->vm_end - vma->vm_start;
4
5     if (size != priv->buf.size)
6         return -EINVAL;
7
8     return dma_mmap_coherent(priv->dev->dev, vma, priv->buf.vir,
    priv->buf.phy, priv->buf.size);
9 }
```

D.1.3. release()

```
1 static int elink_release(struct inode *inode, struct file *file) {
2     struct file_private *priv = file->private_data;
3     if (priv) {
4         if (priv->buf.vir)
5             dma_free_coherent(priv->dev->dev, priv->buf.size, priv
    ->buf.vir, priv->buf.phy);
6         kfree(priv);
7         file->private_data = NULL;
8     }
9     return 0;
10 }
```

D.2. ioctl Interface

```

1 static long elink_ioctl(struct file *file, unsigned int cmd,
2     unsigned long arg) {
3     struct file_private *priv = file->private_data;
4     struct dmaDevice *dmaDev;
5     long wait_ret;
6     u32 status;
7
8     if (!priv)
9         return -ENODEV;
10
11     dmaDev = priv->dev;
12
13     if (!dmaDev)
14         return -ENODEV;
15
16     switch (cmd) {
17     case ELINKDRV_DMA_START:
18         WRITE_ONCE(dmaDev->irq_handled_flag, 0);
19
20         if (dmaRead(dmaDev, BUSY_REG) == 0x00) {
21             /* Trigger DMA */
22             dmaWrite(dmaDev, START_REG, START_VAL);
23
24             /* Wait for completion */
25             wait_ret = wait_event_interruptible_timeout(dmaDev->wq,
26                 READ_ONCE(dmaDev->irq_handled_flag), msecs_to_jiffies(5000));
27             status = dmaRead(dmaDev, STATUS_REG);
28
29             /* Copy status back to userspace */
30             if (copy_to_user((u32 __user *)arg, &status, sizeof(
31                 status)))
32                 return -EFAULT;
33
34             if (wait_ret == 0) {
35                 return -ETIMEDOUT;
36             } else if (wait_ret < 0) {
37                 return wait_ret;
38             }
39         } else {
40             return -EBUSY;
41         }
42
43         return 0;
44     default:
45         return -ENOTTY;
46     }
47 }

```

D.3. Interrupt Handler

```
1 static irqreturn_t elinkdriver_irq(int irq, void *dev_id) {
2     struct dmaDevice *dma = dev_id;
3     dmaWrite(dma, ACK_REG, ACK_VAL);
4     WRITE_ONCE(dma->irq_handled_flag, 1);
5     wake_up_interruptible(&dma->wq);
6     return IRQ_HANDLED;
7 }
```