Update on SoCs in ATLAS Detector Control
The Embedded Monitoring Processor as Example

Dominic Ecker

ATLAS Central DCS – Bergische Universität Wuppertal

EMP-Team: D. Ecker, P. Moschovakos, J.B. Olesen, V. Ryjov, S. Schlenker

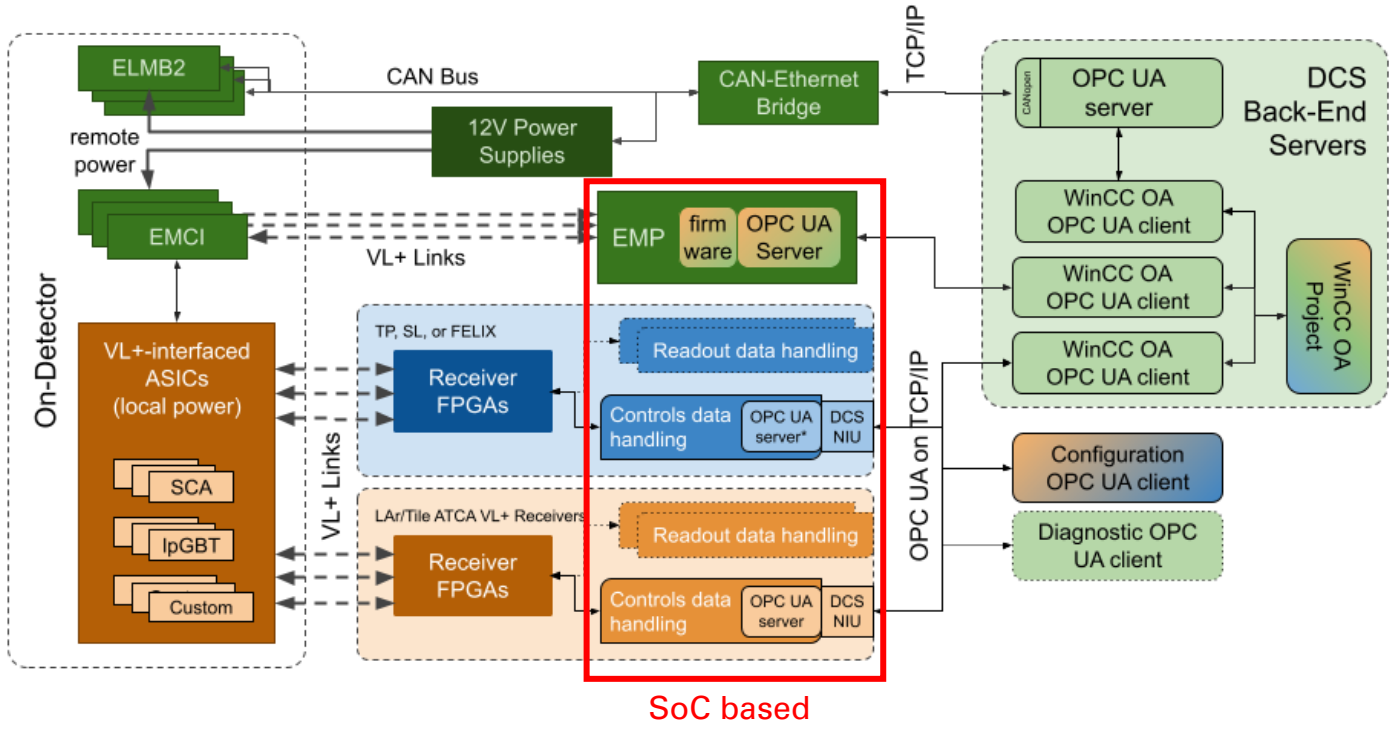
Outline

- Introduction
  - Introduction & requirements
  - Phase-II interface for on-detector FEs
  - DCS standard OPC UA
- The quasar framework
  - quasar – a short introduction
  - Workflow
  - Design approach
  - quasar ecosystem – SoC related news
- The Embedded Monitoring Processor (EMP)
  - EMP-EMCI environment
  - The Embedded Monitoring Processor – EMP
  - Current hardware status of the EMP
  - EMP software & firmware architecture

INTRODUCTION

Phase-II DCS interfaces for on-detector Front-Ends

- Two different paths with respect to communication and powering
  - Primary path for DCS functions with independent communication and power lines.
  - Secondary path uses the communication infrastructure of on-detector electronics interfacing with the off-detector Back-End via Versatile Links (optical)



Introduction & Requirements

- SoCs already widely adopted in ATLAS for Detector Controls and their Applications
- They will play a major role in custom ATCA electronics during ATLAS Phase-II upgrade
- Requirements for DCS BE (Back-End) interface of VL (Versatile Link) receiver with SoC devices:
  - SoC devices should be used for the implementation of the **OPC UA-based DCS BE** interface
  - This OPC UA-based interface should be implemented using the commonly supported **quasar framework**
  - On-detector components which are part of the DCS communication path (e.g., lpGBT) shall be monitored
- Key benefits of SoC within ATLAS DCS
  - Flexibility towards hardware through programmable logic
  - Programmable logic increases scalability of the entire communication chain

DCS protocol standard OPC UA

- What is OPC UA?
  - Cross-platform, open source IEC62541 standard for data exchange
  - TCP based transport mechanism with message-based security model
  - Based on client server communication
  - Ability to exchange data between different industrial automation systems
  - Expandable, open and standardized information model allowing the modeling of complex data structure
  - Commercial SDKs as well as open-source stacks are available in several programming languages

➔ Common standard for ATLAS DCS interfaces

THE QUASAR FRAMEWORK

quasar – a short introduction

The quick OPC UA server generation framework

- CERN made software framework for generating model-driven OPC-UA server
- Hides the OPC UA complexity and transfers the focus to the device the server is developed for
- Reduces coding by around 90%
- Generate C++ code based on user defined XML device description aka the “design”.
- Offers a simple per device API which
  - wraps the OPC UA API
  - provides code stubs for the user-specific code
- Multiple toolsets to simplify the most common tasks required for controls
  - e.g., building OPC UA client applications (C++, WinCC OA)

➔Documentation available here: Read the docs

➔YouTube tutorial available here: Quasar OPC UA Tutorials

➔ 1st and 2nd SoC Workshop quasar references

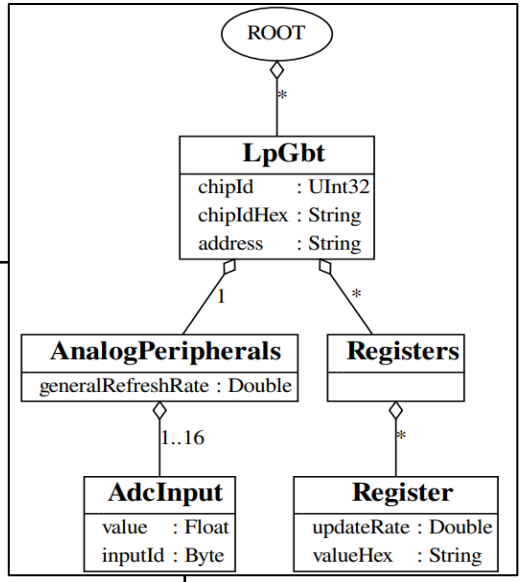
Workflow

- Get quasar:
  $ git clone https://github.com/quasar-team/quasar.git -b v1.6.1 --recursive
- Resolve quasar dependencies
  - specific instructions available for CC 7, CS 8, RHEL 7&8, Ubuntu, AlmaLinux 9 and MS Windows
  - setup help available for other Linux distributions and embedded (PetaLinux, Yocto)
- Create a project:
  $ ./quasar.py create_project ../myOpcUaServer
- Create target specific xml-design
- Build the server:
  $ ./quasar.py build
- OPTIONAL: Enable open-source implementation of the OPC UA API in your project directory:
  $./quasar.py enable_module open62541-compat v1.4.3
- Create target device logic coding and config file population
- Build the server:
  $ ./quasar.py build
- Run OPC UA server:
  $ ./build/bin/OpcUaServer ./build/bin/config.xml

Design approach

- XML based, object-oriented design approach
- Objects can be individually designed and linked to other objects within a hierarchy
- Each object can contain constant values, variables and methods
- You can decide how a value is initialized, the access level (read-write) and the data type
- The design generates xml configuration schema for server config files

```
<d:design projectShortName="OpcUaSampleServerNameChangeItInDesign" >
<d:root>
        <d:hasobjects instantiateUsing="configuration" class="LpGbt"></d:hasobjects>
</d:root>

<d:class name="LpGbt">

        <d:devicelogic></d:devicelogic>

        <d:configentry dataType="UaString" name="address"
        storedInDeviceObject="true"></d:configentry>

        <d:cachevariable initializeWith="valueAndStatus"
                dataType="OpcUa_UInt32" name="chipId" nullPolicy="nullAllowed"
                addressSpaceWrite="forbidden" initialStatus="OpcUa_BadWaitingForInitialData">
        </d:cachevariable>

        <d:hasobjects instantiateUsing="configuration" class="AnalogPeripherals"></d:hasobjects>
        <d:hasobjects instantiateUsing="configuration" class="Registers"></d:hasobjects>

</d:class>
</d:design>
```



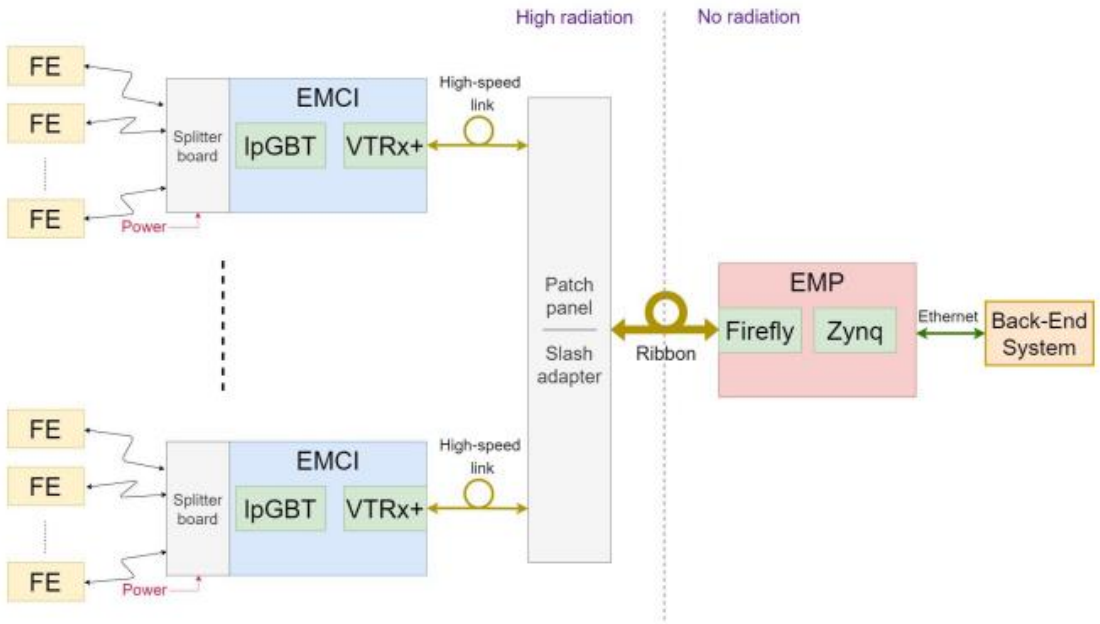
quasar ecosystem – SoC related news

- Current Release: quasar 1.6.1
- 50 quasar-based server projects in production or development:
  - 3 on embedded production systems (Zynq, RPi), more than 11 in development for PHASE-II (mostly Zynq US+ based)
- Latest Features:
  - Quasar Sanitizers – A new feature that assists developers in identifying and fixing code issues
  - Improved Quasar ThreadPool – Affecting all async operations of Quasar (transaction ordering)
  - Enhanced functionality of calculated variables
- Backend options
  - Unified Automation C++ SDK
    - Quasar version support UA SDK 1.5.x, 1.6.x and 1.7.x
    - Support for versions lower than 1.6.x will be discontinued in the future.
  - C++ Open62541-compat is supported as **open-source** solution to Quasar developers
  - What about python?
- Fully python-based quasar solution is available in a limited fashion through MilkyWay
  - Based on FreeOpcUa stack, using quasar designs
  - Currently used in production by ATLAS TDAQ
  - Anybody interested to join effort?

THE EMBEDDED MONITORING PROCESSOR (EMP)

EMP-EMCI environment

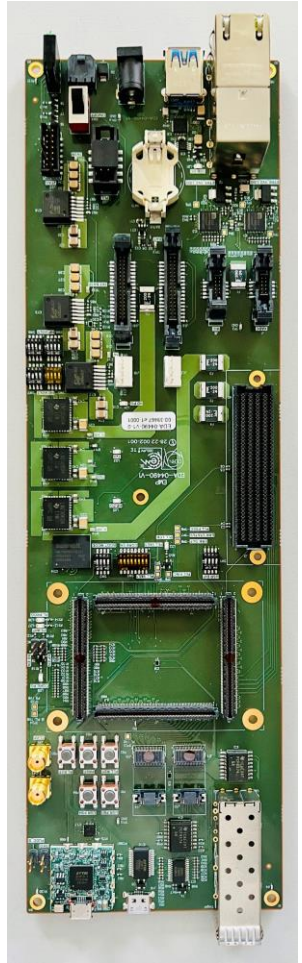
- Requirements for radiation hardening in DCS lead to the development of EMP and EMCI (Embedded Monitoring and Control Interface)
  ➔ Two CERN made boards.
- The EMCI – EMP system is a generic solution to interface multiple Front-Ends (FEs)
- The EMCI (based on CERNs lpGBT) sits close to the FEs and transmits all data between FEs and the BE
  - providing ADC/DAC, slow control and eLink interfaces
- The EMP, the BE board and can interface up to 12 EMCIs via optical links. It is accessible directly from the Network.



The Embedded Monitoring Processor – EMP

Hardware Design v1 of the EMP is composed of a 2 parts:

- The „brain“ of the EMP – a Trenz MPSoC-Module (SoM) featuring:
  - AMD Zynq™ UltraScale+™
- The EMP baseboard featuring:
  - Samtec Firefly CERN-B-T12 (12 TX) & Samtec Firefly CERN-B-R12 (12 RX)
  - SFP+ interface
  - FMC High Pin Count connector (72 LVDS, 3 MGT pairs, I2C interface)
  - 2 pin Headers (12 LVDS)
  - PS interfaces via dual ethernet, USB3 and UART
  - XADC (accessible though a 10-pin header)



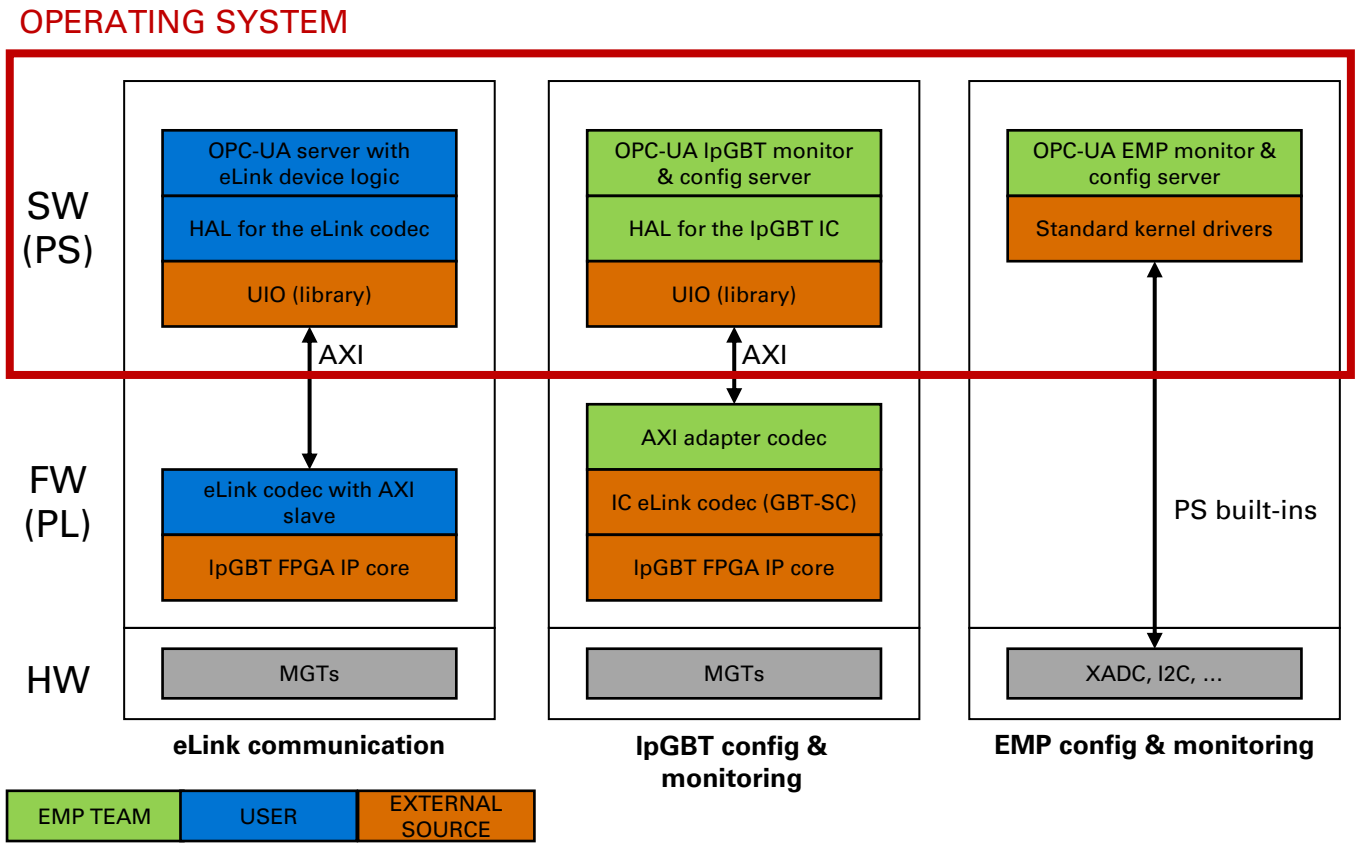
The „brain“ of the EMP – TE0807 MPSoC

- Key Feature of the TE0807 MPSoC:
  - XCZU4EG provides a Quad ARM Cortex-A53
  - PL exposes 204 User I/Os
  - 16 GTH transceivers
  - SI5345 – 10 output clock chip (used for GTH transceivers)
  - 4 Gbyte DDR4
  - 128 MByte SPI Boot Flash
  - Schematics are available and easy to reuse by Designers
- Related Trenz baseboard (TEBF0808)
  - Design reference for EMP baseboard (based on Trenz schematics)
  - Used for developing process by Users and EMP Team
  - Availability still problematic

Current hardware status of the EMP project

- EMP baseboard prototype v1 available since Q4/2022
- EMP-Hardware verification ongoing:
  - Firefly (IBERT test successful) – EMCI communication
  - SFP+ communication to EMCI verified
  - I2C configuration & monitoring for peripheral devices verified
  - Dual ethernet support tested
  - JTAG & UART tested
  - SD & NFS boot tested
- BSP available containing hardware support for EMP baseboard
- v2 EMP baseboard expected for Q1/2024

EMP software & firmware architecture



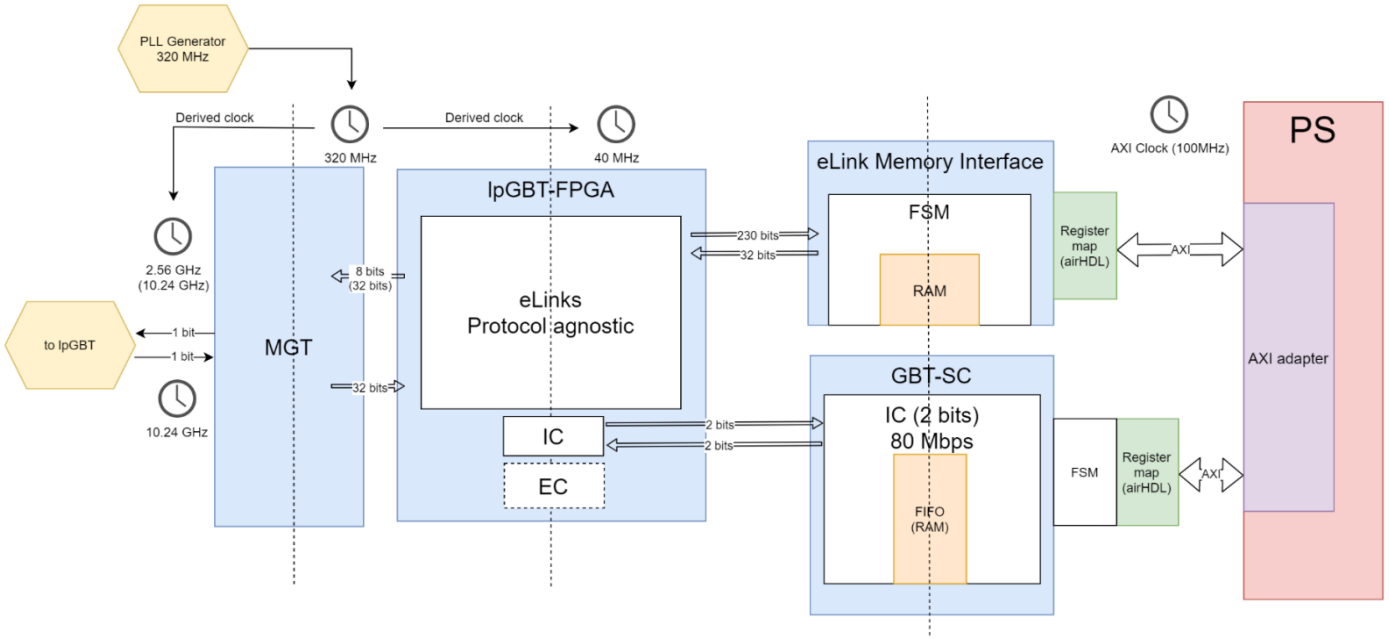
- SW & FW architecture must cover three different functionality modes:
  - elink communication
    - Specific to the target device
  - EMCI configuration & monitoring
  - EMP configuration & monitoring
- Each usage mode is composed of common or custom SW, FW & HW blocks

Common need for an Operating System, basic software support for EMP peripherals & common FW building blocks

EMP firmware

Custom IP cores in EMP firmware:

- lpGBT-FPGA IP (provided by GBT Team)
  - Down- and Uplink data path to interface the lpGBT through its high-speed link
- GBT-SC IP (provided by GBT Team)
  - Decodes the IC and EC channel frames of the lpGBT
- eLink Memory Interface (custom IP)
  - Custom IP core which serves as an example how to make use of the eLinks (uplink/downlink)



Common IP cores in EMP firmware:

- AXI Interconnect IP
  - Register based slave communication to Zynq PS
  - AXI4 register maps generated with airhdl
- IBERT IP
  - Used for testing Firefly & SFP+ transceivers

EPOS – The EMP Operating System

- Commonalities of EMP HW lead to a need for a boot image, a Linux kernel, and a file system with basic hardware peripherals and software tooling support!
- EPOS is the framework to build the necessary infrastructure for the EMP

EPOS BSP

- PetaLinux BSP containing the basic periphery support
  - Device tree for common HW
  - Configuration to build & package the files needed for booting the EMP

EMP ROOTFS

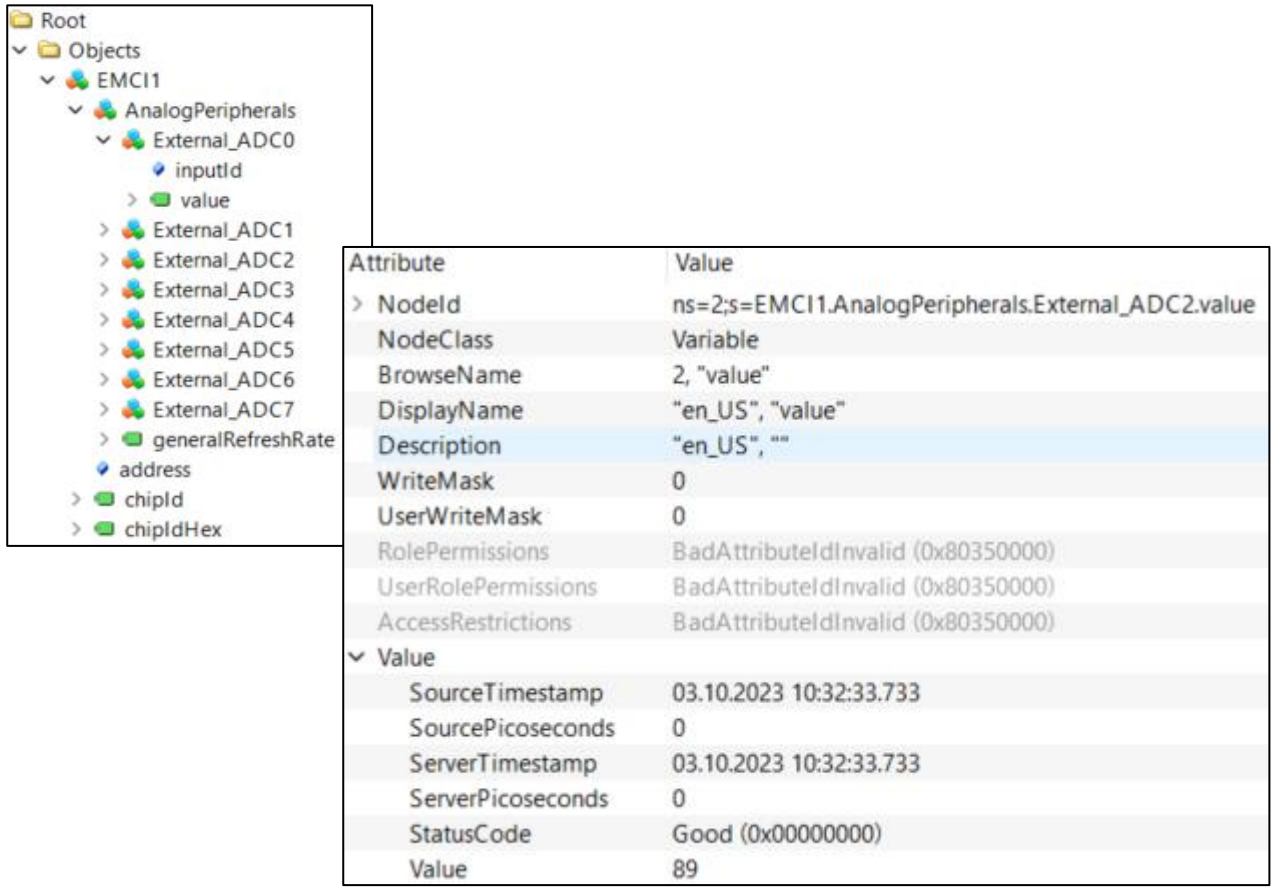
- Set of tools to create the file system deployed in the EMP
  - Standard OS distribution supported by CERN IT (currently CS 8, Alma 9 in progress)
  - Resolves dependencies required for EMP usage and Quasar server development

➔ **Can serve as template for other SoC based projects**

EMP software

- Lpgbt Software (development in progress – functionality will further evolve)
  - Abstraction layer implementing the lpGBT business logic
  - Communicates with PL via UIO
  - Contains the knowledge on how to access the lpGBT interfaces (ADC, GPIO & I2C)
  - Supports lpGBT v0 and v1
  - Contains a set of standalone demonstrator tools to readout the ADC, raw registers and lpGBT configuration
  - ADC readout and lpGBT configuration already demonstrated with EMCI-EMP
- EMP Software
  - Direct communication and control to peripheral devices on EMP baseboard
  - Based on standard kernel drivers
  - Used to implement monitoring and control of EMP hardware
- eLink Software
  - Requires custom HAL for eLink codec
  - Communication with PL via UIO
  - Example application shows basic usage by reading data of a specified eLink via register maps from PS

lpGBT in OPC UA address space



- Example of the lpGBT software stack within a quasar-built OPC UA server

THANK YOU!

EXTRA

emp-tools and etools

- emp-tools:
  - Set of SW tools to control and monitor peripheral on-board devices via I2C like:
    - Programming the S5345 PLL chip (necessary to drive 320 MHz clock for optical links)
    - Readout Baseboard Temperatures
    - Configuration and monitoring of optical drivers (SFP+, Firefly)

- etools:
  - Set of SW tools to
    - create device tree overlays from hardware designs (.xsa)
    - load firmware from Zynq PS via device tree overlay

Global picture of the OPC UA lpGBT server