A SOFTWARE SUITE FOR THE RADIATION TOLERANT GIGA-BIT TRANSCEIVER - SLOW CONTROL ADAPTER

P. Moschovakos*, P.P. Nikiel, S. Schlenker, CERN, Geneva, Switzerland
H. Boterenbrood, Nikhef, Amsterdam, Netherlands
A. Koulouris, National Technical University of Athens, Athens, Greece

Abstract

The future upgrades of the LHC (Large Hadron Collider) will increase its luminosity. To fulfil the needs of the detector electronic upgrades and in particular to cope with the extreme radiation environment, the GBT-SCA (Giga-Bit Transceiver - Slow Control Adapter) Application-specific integrated circuit (ASIC) was developed for the control and monitoring of on-detector electronics. To benefit maximally from the ASIC, a flexible and hardware interface agnostic software suite was developed.

A hardware abstraction layer - the *SCA software package* - exploits the abilities of the chip, maximizes its potential performance for back-end implementations, provides control over ASIC configuration, and enables concurrent operations wherever possible. An OPC UA server was developed on top of the *SCA software library* to integrate seamlessly with distributed control systems used for detector control and Trigger/DAQ (Data AcQuisition) configuration, both of which communicate with the GBT-SCA via network-attached optical link receivers based on FPGAs.

This paper describes the architecture, design and implementation aspects of the *SCA software suite* components and their application in the ATLAS experiment.

SCA SOFTWARE SUITE CONTEXT

Introduction

The GBT-SCA, or SCA for short, is a radiation-tolerant ASIC and part of a chip-set of the GBT project, in which a bi-directional 4.8 Gbps optical link architecture has been developed using a SEU robust protocol [1], providing simultaneous transfer of readout data, timing and trigger signals as well as slow control and monitoring data, by multiplexing multiple logical electrical data links of 80, 160 or 320 Mbps, called E-links [2] onto a single optical link using the rad-tolerant GBTX ASIC on the front-end side. The SCA's purpose is to interface to control and monitoring signals of front-end electronics on the detectors, using two redundant 80Mbps E-links to connect to a GBTX.

The SCA employs the HDLC (High-level Data Link Control) protocol on its E-links in a synchronous request-reply communication pattern. On top of the HDLC data link layer, a custom protocol has been implemented to address the different hardware devices (or channels) present on the SCA chip.

The SCA has 16 independent I^{2}C (Inter-Integrated Circuit) serial bus masters, an SPI serial bus master, a JTAG serial bus, 32 GPIO ports (General Purpose I/O), an ADC with 31 analogue inputs, an embedded temperature sensor and 4 independent DAC. The SCA request and reply message layout is shown in Fig. 1

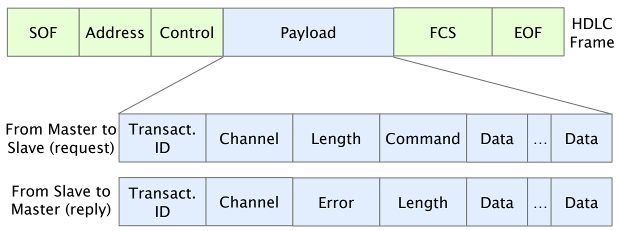

Figure 1: The SCA request and reply message layout.

The channels operate independently from each other in order to allow concurrent transactions[1] and perform concurrent transfers from their end-devices [3].

Functionality and Requirements

The software package is required to provide a high level of abstraction and means for interfacing to all communication channels of an SCA profiting from the hardware parallelism in-between independent channels. In order to ensure reliability, the software needs to do the necessary bookkeeping for the synchronous communication and transaction tracking.

Moreover, the software is required to achieve high performance and low latency, including features like grouping of requests to perform lengthy operations, such as field-programmable gate array (FPGA) programming, requiring transfer of large amounts of data over JTAG. Since thousands of SCAs will be used in the detector systems, scalability is an important design aspect. At the same time, monitoring and control tasks require a high availability of close to 100%, implying the need of a high level of robustness.

Finally, the software needs to adapt to different communication scenarios of the SCA. A simulated chip needs to be supported as well as prototype board communication interfaces for development and testing purposes. For the final production system, the *SCA software* is interfaced with the optical link receiver system - FELIX [4, 5] via a dedicated communication link called netIO [6].

Integration Overview

Figure 2 shows an overview of the SCA integration chain, illustrating the interplay of the *SCA software suite* components.

* paris.moschovakos@cern.ch

[1] Data are gettiing serialized in the physical line, while HDLC sequence number is used to keep the traffic in order when the data are ready to be transmitted

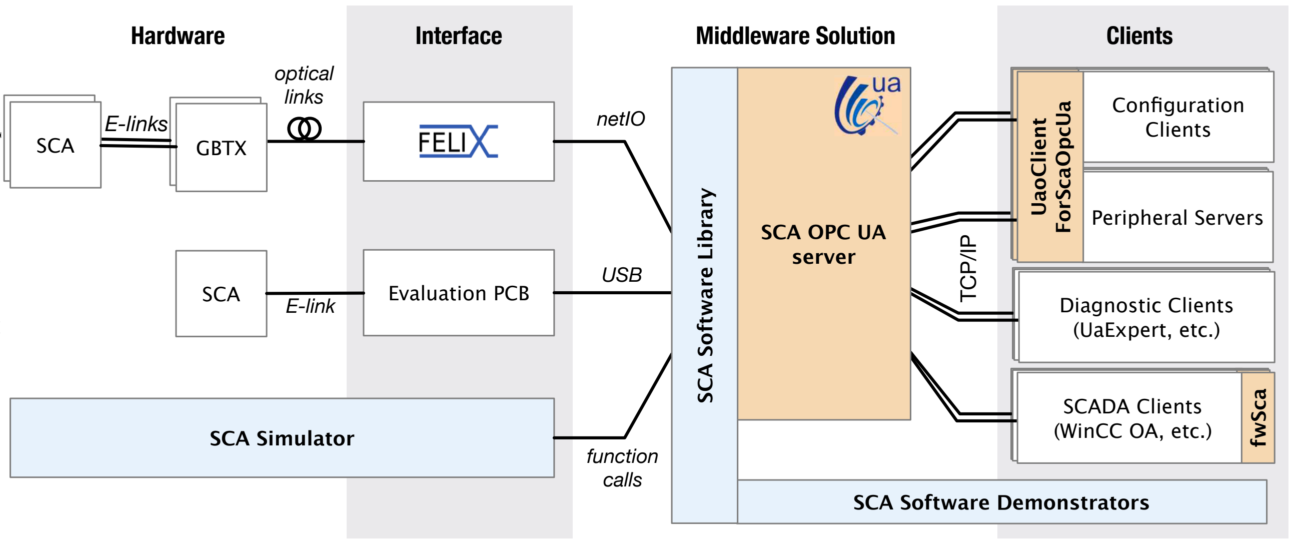

Figure 2: Global picture of the software suite. The *SCA Software package*, in light blue, comprises the *SCA Software API* to communicate with the SCA via different back-ends, the *SCA Simulator* to emulate SCA traffic for testing and development, and the *Demonstrator* tools which are used for standalone operations. The *SCA OPC UA* server and its ecosystem, in orange, is the middleware of choice to exchange data with the front-ends. *UaoClientForScaOpcUa* is a library that clients use to communicate with the SCA server. Finally, the *fwSca* module automatizes the integration of the server data into SCADA systems.

THE SCA SOFTWARE PACKAGE

In the *SCA Software package* [7] core there is a library that is structured in modules that implement the required functionality in various layers. The library was designed to be flexible and easily adaptable to the diverse systems intended to use it by its polymorphic HDLC back-end. A block diagram of the software architecture of this library is shown in Fig. 3. Moreover, the *SCA Software package* contains the *Demonstrators* which are tools that directly use the library and are used for testing and for low level diagnostics. Finally, as part of the package, an *SCA Simulator* was developed that is able to generate SCA traffic, simulating realistic SCA behaviour, in order to allow for development and testing without real hardware.

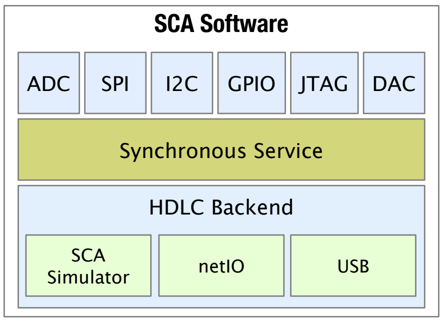

Figure 3: *SCA Software Library* stack.

HDLC Back-end

The HDLC back-end is a software abstraction of the back-end to be used. The existing back-end implementations were created for netIO (for FELIX-based systems), ScaSimulator and for the SCA evaluation board via USB. The HDLC back-end is independent of the actual SCA data provider as a polymorphic interface unifying sending requests and subscribing to replies. It facilitates the handling of the payload and organizes a common addressing scheme among back-ends.

Synchronous Service

The synchronous service is responsible for transaction tracking, time-out handling and most importantly synchronization of multiple threads accessing the same SCA. Further, it allows for full concurrency among SCA channels.

SCA Communication Interfaces Library

The SCA communication interfaces library is a high level abstraction library to control the user interface ports and the configuration of the ASIC. A user can perform complex operations e.g. SPI/I^2C configuration of an ASIC or programming a Xilinx FPGA via JTAG etc with simple API calls.

Accompanying Demonstrators

The *SCA Software package* also includes standalone low level tools, called *Demonstrators*. The *Demonstrators* which use the SCA API to perform standalone operations, like I^2C write/read or ADC monitoring, are used as debugging and diagnostic tools or as an example of the SCA API usage for example SPI configuration of a specific ASIC.

THE SCA OPC UA ECOSYSTEM

The Detector Control Systems (DCS) has chosen OPC UA [8] as its standard middleware for the following reasons:

- Focus on communicating with industrial equipment and systems for data collection and control
- Open specification and various implementations available (free or commercial)
- Cross-platform
- Service-oriented architecture
- Inherent complexity
- Robust security
- Integral information model, which is the foundation of the infrastructure necessary for information integration where vendors and organizations can model their complex data into an OPC UA namespace.

SCA OPC UA Server

The *SCA OPC UA* server [9] was implemented using the *quasar* [10–12] framework, a software framework for the efficient creation of OPC UA servers offering a very efficient development path. The *SCA OPC UA* server takes advantage of the *quasar* built-in features such as calculated variables, different types of variables and methods and advanced threading.

The server architecture divides the SCA channels into device classes, corresponding to the respective hardware function such as the I^2C or ADC interfaces. In addition, a *Global Statistician* module was developed to collect and measure general statistics across the setup and to expose the collected metrics to the clients. Finally, an *SCA Supervisor* software module oversees the state of the system and provides supervisory functionality such as automatic recovery of communication loss with SCAs, SCA ID validation and other administrative tasks.

The structure of all *quasar* classes used in the server are described by the *quasar* design diagram shown in Fig. 5.

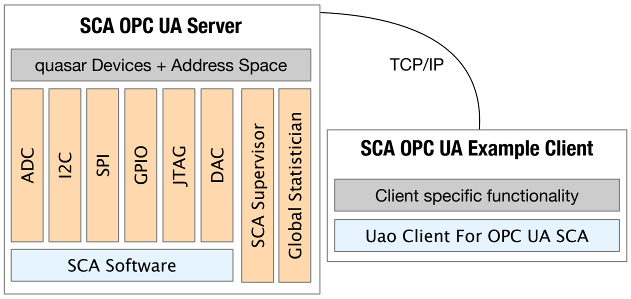
SCA OPC UA Server
quasar Devices + Address Space
ADC
I2C
SPI
GPIO
JTAG
DAC
SCA Supervisor
Global Statistician
SCA Software
TCP/IP
SCA OPC UA Example Client
Client specific functionality
Uao Client For OPC UA SCA

Figure 4: *SCA OPC UA* stack.

SCA OPC UA Clients

Applications from different domains will need to have access to the same SCA system and use it for distinct purposes. For that reason, the general *SCA OPC UA* server is used as the hub from where the data pass and get synchronized, while any specialized application was chosen to be decentralized (as shown in Fig. 2). This solution makes the maintenance reasonable, has the advantage of dividing responsibilities among different communities, allows for staging e.g. making higher level wrapper applications and allows for interoperability between diverse clients.

To support the concept, a *quasar* generated [13] C++ library, namely *UaoClientForOpcUaSca* [14] is provided for building ad-hoc OPC UA clients. This library supplies the interface to the *SCA OPC UA* server and is created based on information sourced from the design of the server. Applications in ATLAS, that use the aforementioned library, are Trigger/DAQ OPC UA clients used for configuration, or peripheral servers which perform sub-detector specific higher level operations. The simplified architecture of the OPC UA SCA server and an example *SCA OPC UA* client that uses this library is depicted in Fig. 4.

The server allows for the usage of general purpose test clients for diagnostic purposes such as the UaExpert [15].

Finally, the most common way that an OPC UA server is used by the DCS is through SCADA WinCC OA [16] based systems. Those systems employ OPC UA clients which connect to the servers to retrieve the data and visualize the information in a user interface, usually deployed in a counting room where the system is monitored by a shifter.

In the case of WinCC OA, the OPC UA connectivity is done via a module, namely *fwSca*, which is based on code generated by *quasar* tooling and is supplied by the *SCA software suite*. This module allows for fast integration as it creates all the necessary configuration in the WinCC OA side based on a priori information of the *SCA OPC UA* information schema.

PERFORMANCE

The server has been designed to serve setups of various sizes and types. The biggest challenge is the NSW (New Small Wheel) upgrade project of ATLAS. In this system, 6976 SCAs are employed, distributed over different types of front-end electronic boards. The traffic of the SCAs are handled by 30 FELIX hosts with 18 optical fibre connections each, and a similar number of *SCA OPC UA* servers.

In an early integration setup, the *SCA OPC UA* server was tested against a full sector slice of the NSW micromegas sub-system with 8 detector layers fully equipped with their front-end electronics. The slice serves 160 SCAs, handled by a single server which was used in various realistic scenarios. The SCA are separated in three categories/types of electronics with different functionality and interfaces as described in the Table 1.

The setup used one FELIX host equipped with an Intel(R) Xeon(R) CPU E5-1650 v4 @3.60 GHz. The *SCA*

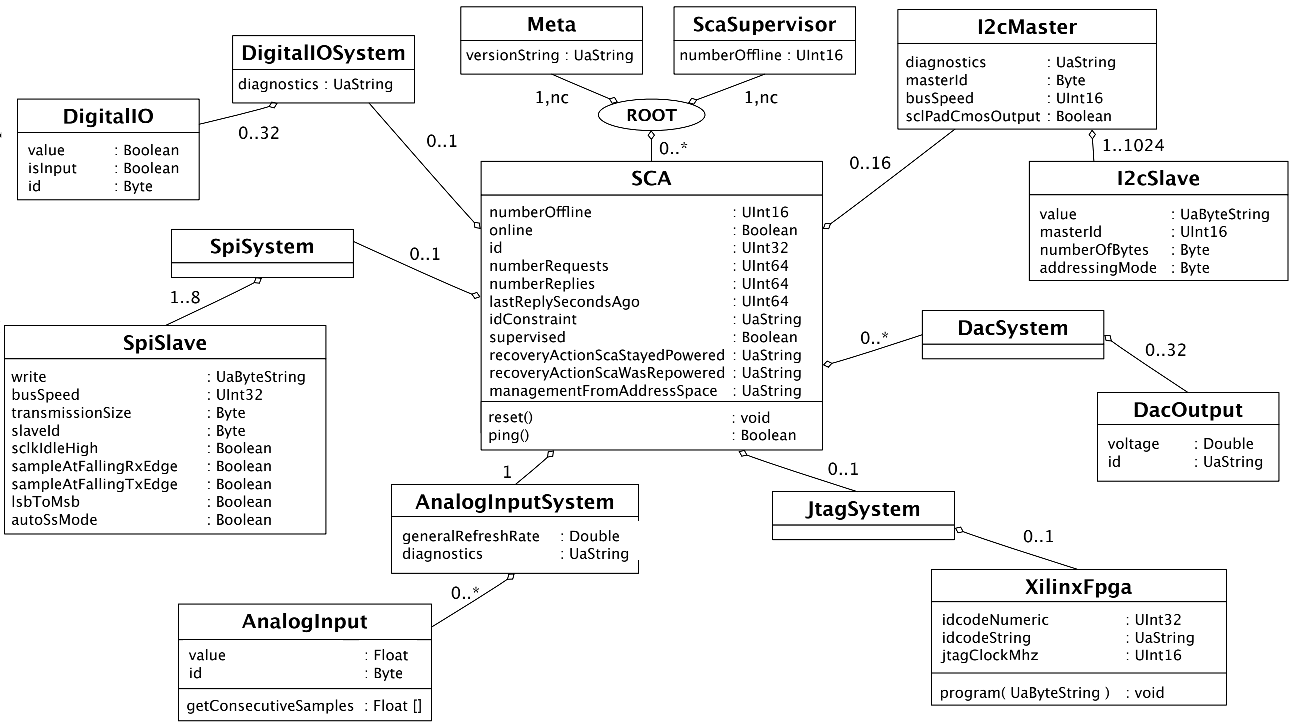

Figure 5: *quasar* design diagram of *SCA OPC UA* server.

OPC UA server runs in the FELIX host machine along with FELIX software. In a first constant-throughput scenario, a WinCC OA SCADA application was monitoring the analogue inputs from a separate host while three OPC UA clients were connected and used for diagnostics. In the second burst traffic scenario, the server was used by 128 additional configuration clients.

Table 1: SCA channel usage in the ATLAS NSW micromegas full sector slice. The setup was used to evaluate the performance of the server.

Board Name	MMFE8	ADDC	L1DDC
Functionality	readout	trigger aggregator	data aggregator
SCA Numbers	128	16	16
ADC Inputs	15	10	9
Calculated variables	15	10	9
I^{2}C Master	2	6	2
I^{2}C Slave	44+60	6	2
SPI Slave	8	-	-
GPIO	19	18	-

Constant-throughput Monitoring Traffic

Even when no configuration activities are performed, the server is used constantly to provide monitoring data from the detector electronics. Those data which mostly come from analogue inputs correspond to the minimum possible activity of the server. In the aforementioned setup, the global request rate was measured to be ~7800 requests/s for 2192 ADC inputs (each analogue input read consists of two SCA requests). That resulted in an actual refresh rate of ~2 Hz per analogue input. The CPU usage of the server reached ~25% in average and the used share of available physical memory was 340 MB, a metric that is stable and not dependent on the usage.

On-demand SCA Traffic - Front-end Configuration

The most challenging in terms of open sessions and process complexity is the configuration of the front-end boards. Emulating the cold start of the NSW micromegas detector, a full sector configuration was attempted in addition to the constant-throughput monitoring traffic as described above. During the process, up to 58 concurrent sessions were established from various OPC UA clients. The configuration clients were programming the front-end electronics using a combination of interleaving operations in-between GPIO, I^{2}C, and SPI totalling around 2700 requests for each SCA.

The global request rate reached ~35000 requests/s. The instantaneous CPU usage peaked at 218%. The total time to initialize all the front-ends was measured to be 10 s.

CONCLUSION

A comprehensive solution for the multi-purpose radiation tolerant GBT-SCA ASIC was presented. The architecture allows concurrent usage by multiple applications and emphasizes reliability, availability, and scalability resulting in a solution suitable for large experimental physics control systems. The software stack provides a high-level abstraction, taking advantage of all ASIC functions, making communication and design aspects of the hardware largely transparent. The usage of the *quasar* framework in the design and implementation of the *SCA OPC UA* ecosystem proved to be an ideal choice due to its efficient development process which enabled the users within the detector controls and data acquisition teams to swiftly integrate the GBT-SCA into their applications. Usability and good performance was demonstrated with a large setup from the New Small Wheel detector upgrade project within the ATLAS experiment at CERN.

REFERENCES

[1] K Wyllie *et al.*, "A Gigabit Transceiver for Data Transmission in Future High Energy Physics Experiments", in *Phys. Procedia*, vol. 37, pp. 1561–1568, 2012. http://cds.cern.ch/record/2103391

[2] S Bonacini, K Kloukinas, and P Moreira. "E-link: A Radiation-Hard Low-Power Electrical Link for Chip-to-Chip Communication", in *Proc. Topical Workshop on Electronics for Particle Physics*, Paris, France, Sep. 2009, pp. 422-425. doi:10.5170/CERN-2009-006.422

[3] A. Caratelli *et al.*, "The GBT-SCA, a radiation tolerant ASIC for detector control and monitoring applications in HEP experiments", *J. Instrum.*, vol. 10, no. 3, p. C03034, 2015. doi:10.1088/1748-0221/10/03/C03034

[4] J. Narevicius, J.T. Anderson, A. Borga, *et al.*, "FELIX: The New Approach for Interfacing to Front-end Electronics for the ATLAS Experiment", CERN, Geneva, Switzerland, Tech. Rep. ATL-DAQ-PROC-2016-009. doi:10.1109/RTC.2016.7543142.

[5] W. Wu, *et al.*, "FELIX: the New Detector Interface for the ATLAS Experiment", *IEEE Trans. Nucl. Sci.*, vol. 66, no. 7, pp. 986–992, 2019. doi:10.1109/TNS.2019.2913617

[6] J. Schumacher, "Utilizing HPC Network Technologies in High Energy Physics Experiments", in *Proc. IEEE 25th Annual Symposium on High-Performance Interconnectsv(HOTI'17*, Santa Clara, CA, USA, Aug. 2017, pp. 57–64. doi:10.1109/HOTI.2017.25

[7] A. Koulouris, P. Moschovakos, and P.P. Nikiel, "Sca-Software gitlab repository", https://gitlab.cern.ch/atlas-dcs-common-software/ScaSoftware

[8] W.Mahnke, S.H. Leitner, and M.Damm. "OPC unified architecture", Berlin: Springer, 2009.

[9] H. Boterenbrood, A. Koulouris, P. Moschovakos, and P.P. Nikiel, "OpcUaSca gitlab repository", https://gitlab.cern.ch/atlas-dcs-opcua-servers/ScaOpcUa

[10] S. Schlenker, B. Farnham, P. P. Nikiel, C.-V. Soare, D. Abalo Miron, and V. Filimonov, "quasar - A Generic Framework for Rapid Development of OPC UA Servers", in *Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15)*, Melbourne, Australia, Oct. 2015, pp. 602–605. doi:10.18429/JACoW-ICALEPCS2015-WEB3O02

[11] P.P. Nikiel, B. Farnham, V. Filimonov, and S. Schlenker, "Generic OPC UA Server Framework", *J. Phys.: Conf. Ser.*, vol. 664, p. 082039 (8 pages), 2015. doi:10.1088/1742-6596/664/8/082039

[12] P. P. Nikiel, P. Moschovakos, and S. Schlenker, "quasar : The Full-Stack Solution for Creation of OPC-UA Middleware", presented at the 17th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'19), New York, NY, USA, Oct. 2019, paper MOPHA100.

[13] P.P. Nikiel and K. Korcyl, "Object Mapping in the OPC-UA Protocol for Statically and Dynamically Typed Programming Languages", *Computing and Informatics*, vol. 37, pp. 946–968, 2018. doi:10.4149/cai2018_4_946

[14] P. Moschovakos and P.P. Nikiel, "UaoClientForOpcUaSca gitlab repository", https://gitlab.cern.ch/atlas-dcs-opcua-servers/UaoClientForOpcUaSca

[15] UaExpert, Unified Automation, https://www.unified-automation.com/products/development-tools/uaexpert.html

[16] "Modern, efficient and flexible simatic wincc open architecture v3.15 ", ETM professional control GmbH, a Siemens Company, Tech. Rep. 6ZB5370-1EG02-0BA0-V2, Jan. 2017. https://siemens.com/wincc-open-architecture