SoCs and Control System Integration

Stefan Schlenker, CERN

Input from and thanks to: Piotr Nikiel, Paris Moschovakos, Ben Farnham, Michael Ludwig, Viatcheslav Filimonov, Revital Kopelianski
Motivation

Integration of devices into detector control system (DCS) back-end

► **Basic principle:** each Front-End (FE) hardware component should be monitored/control by DCS

► **Usually four main scenarios** for mainstream hardware components to be interfaced:

1. On-detector FE electronics interfaced via optical links and FPGA receivers
2. Entirely independent (signal+power) monitoring/control via e.g. ELMB2/ELMB++ (on/off-detector) or other custom equipment
3. Off-detector power supply systems
4. Off-detector electronics crates ( =ATCA in ATLAS Phase II upgrades)

► **DCS back-end integration** shall be done via
   - Ethernet+TCP/IP as standard FE interface
   - **Standard middleware** for interfacing with SCADA system / back-end applications

   ATLAS has chosen **OPC UA** (since Phase-0 upgrades)
On-Detector Paths

Legend:

- **Hardware**
- **Software**
- **Firmware**

**System-on-Chip Workshop**, 12th-14th of June, CERN
Off-Detector Paths

Example ATCA

ATCA shelf

ATCA board

Shelf Manager

SNMP agent

IPMI via I²C

IPMC

IPMC

Custom ASICs, FPGAs, etc.

IPbus core

SoC: FPGA +processor

OPC UA Server

DCS Back-End

WinCC OA

OPC UA server

WinCC OA OPC UA client

OPC UA server

WinCC OA OPC UA client

OPC UA server

WinCC OA OPC UA client

Configuration OPC UA client

Diagnostic OPC UA client

Legend:

Hardware Software Firmware

SoC

Legend:

OPC UA server

SNMP

WinCC OA

Project

(opional)

Hardware Software Firmware

System-on-Chip Workshop, 12th-14th of June, CERN
Why SoC?

► To interface hardware as done for many Ethernet-interfaced devices nowadays: with embedded Linux

► SoC in general is very flexible since it allows to run software and programmable logic/firmware in the same device ⇒ very useful also for non-DCS users

► Optical Receivers for on-detector data: SoC simplifies early extraction of controls data within combined streams and easily provides network interface with TCP/IP

► To allow embedding of standard middleware closer to hardware
  o Avoids ecosystem of ad-hoc/custom protocols over Ethernet which are otherwise used
  o Hardware responsible/developer can concentrate on hw interface tool/library usually anyway needed, integrate with middleware – framework/tooling may be provided by central group
  o Can provide an abstraction layer if desired, examples:
    • calculating+calibrating a temperature value from several FPGA registers can be done in embedded software
    • FPGA configuration may be done conveniently via client call transferring e.g. a binary bit file
  o DCS back-end experts may avoid dealing with hardware-specific protocols or low-level data handling, can concentrate e.g. on WinCC OA integration (tooling provided by central DCS/JCOP)
OPC Unified Architecture

Industrial machine-to-machine communication protocol for interoperability

- Originally developed by OPC Foundation for IoT applications (keyword Industry 4.0)
- OO Information modeling capabilities
- Enhanced security, performance and scalability
- Supports buffering, session mgmt, pub-sub, per-connection heartbeats/timeouts, discovery
- Multi-platform implementation, lightweight embedding possible
- Commercial SDKs available with stack from OPC foundation or open source stack implementations (C, C++, Java, JS, Python) for servers and clients

- Excellent experience in ATLAS since 2012
- Fully supported by JCOP
- Still requires expertise and effort in programming with OPC UA ...

- Provide development environment and generate OPC UA related code?
quasar – Quick opcUA Server generAtion fRamework

A tool for rapid C++ server development and more

► CERN-made (currently ATLAS DCS, BE-ICS, alumni) framework for model-driven creation of OPC-UA software components
  o generates servers, clients (Uao, UaoForPython), SCADA integration layer (fwQuasar), etc...
► Made with effort efficiency in mind (design, development, testing, deployment)
► quasar-based software used in LHC experiments (JCOP) as well as beyond CERN
► quasar can build 100% free and open source OPC-UA servers and clients
► Validated on different platforms, operating systems, software deployment strategies etc...
► Choice of OPC-UA stack used: UA SDK (paid license) or open62541 (free & open-source)
► Dependencies (all open source):
  C++ compiler (gcc ...),
  boost (regex, thread, system, program-options),
  jre, cmake, xsd, python (+lxml, +enum34, +six),
  xerces-c, libxml2, openssl
  OPC-UA server toolkit (C++) – Unified Automation
  OPC UA client
  OPC UA client
  OPC UA client
  OPC-UA server configuration
  Security (X509 certificate handling)
  Logging
  Common namespace items and namespace utilities
  Server meta-information
  Embedded python
  Device logic
  Device access layer
  XML config file
  Hardware
  Hardware
  Remote process

 Commercial/OS toolkit
 Provided or generated components
 Device specific logic, partially generated
 100% application developer/vendor

Stefan Schlenker, CERN
### Modus Operandi

#### Developer benefits:

- **Design file** can be created using provided XSD schema
- Roughly 50-90% of code can be generated
- User sections of **Device Logic** stubs are well separated, merging tool simplifies re-generation after design changes or quasar upgrades
- **CMake** based build system with pre-built toolchains for several platforms
- Can generate rpms or perform INSTALL ((Yocto+PetaLinux)
- **Configuration file** can be created using generated XSD schema

---

Get + understand model of target device/system

Fill/edit **Design File**

Generate UA address space + configuration module

(Re-)generate **Device Logic** stubs and variable handling

Implement/merge user sections of **Device Logic**

Choose platform, build server + test binaries

Fill **Configuration File**

Test, evaluate …

Device model is OK

Device Logic is OK

Generate SCADA types, instances, UA addressing

END
quasar Server Example Design

Generated class diagram of an advanced quasar server for GBT-SCA interfacing

Authors: SCA-SW team
Components & Tools

Tools

- **Design visualization**: UML generator
- **Platform** toolchains: Linux x86_64, i686, ARM (Raspbian, YOCTO, PetaLinux, CentOS), Windows 32/64
- **Easy RPM generator**
- **Generated program to test full address space**
- **Documentation**: in-design doc to generate auto-documentation in config schema and address space
- **Software management**: consistency checker helps using versioning system

Archiving
- SQL and NoSQL archivers

XML configuration
- **Generated schema** simple creation
- **Validation tool** verify design constraints
- **Generated loader** for object instantiation and runtime access to configuration

Logging
- Provides API and exchangeable back-end, component based

Client Generation
- **Generate client code** from quasar-based server project, enables server chains, no OPC-UA specific code for users!

WinCC OA Integration
- Generates WinCCOA data types, instances and addresses
- **python support**
  - Embed python scripts inside server device logic
  - Embed server project in an existing python application (no C++ coding)

CalculatedVariables
- attaching synthetic variables to existing hardware-collected data (without writing any code)

Server meta-information
- # Items, memory usage, thread pool size, run time ...

More to come…
## Some quasar-based projects

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LArPurity</td>
<td>ATLAS Liquid Argon calorimeter purity analyzer</td>
<td>Production since 2015</td>
<td></td>
</tr>
<tr>
<td>IpBus</td>
<td>Generic IpBus</td>
<td>Production since 2018</td>
<td>will become deprecated</td>
</tr>
<tr>
<td>ATLAS Wiener</td>
<td>Wiener VME crates interfaced with CAN</td>
<td>Production since 2015</td>
<td></td>
</tr>
<tr>
<td>TileHvMicro</td>
<td>HV Micro, ATLAS Tile calorimeter</td>
<td>Production since 2015</td>
<td></td>
</tr>
<tr>
<td>CAEN</td>
<td>CAEN power supplies, JCOP</td>
<td>Production since 2018</td>
<td></td>
</tr>
<tr>
<td>ISEG</td>
<td>ISEG power supplies, JCOP</td>
<td>Production since 2017</td>
<td></td>
</tr>
<tr>
<td>JCOP Wiener</td>
<td>SNMP and CAN Wiener devices</td>
<td>Development</td>
<td></td>
</tr>
<tr>
<td>FtkSbc</td>
<td>SBC monitoring, ATLAS FTK</td>
<td>Production since 2017</td>
<td></td>
</tr>
<tr>
<td>SCA</td>
<td>GBT-SCA for ATLAS NSW, LAr and BIS</td>
<td>In test</td>
<td>+ Uao</td>
</tr>
<tr>
<td>HvSys</td>
<td>HVSys monitoring and controls over RS232, ATLAS TRT</td>
<td>Production since 2017</td>
<td></td>
</tr>
<tr>
<td>Generic SNMP</td>
<td>Generic SNMP</td>
<td>Development</td>
<td></td>
</tr>
<tr>
<td>LAr LTDB</td>
<td>LTDB+LATOME monitoring and controls</td>
<td>Development</td>
<td>+ Uao, + peripheral</td>
</tr>
<tr>
<td>gFEX</td>
<td>ATLAS L1 TDAQ gFEX</td>
<td>Development</td>
<td>+ embedded (Zynq)</td>
</tr>
<tr>
<td>eFEX</td>
<td>ATLAS L1 TDAQ eFEX</td>
<td>Development</td>
<td>+ used from Python</td>
</tr>
<tr>
<td>ATCA Shelf Manager</td>
<td>nVent Schroff (aka Pigeon Point) PPS MIB</td>
<td>In test</td>
<td></td>
</tr>
<tr>
<td>ELMB++</td>
<td>ELMB++ Receiver</td>
<td>Development</td>
<td>+ embedded (Zynq)</td>
</tr>
</tbody>
</table>
quasar on SoC: Yocto/PetaLinux, native or x-compiler builds

quasar-based OPC-UA servers natively built for Yocto / PetaLinux (based on Yocto)

► Validated for platforms/devices:
  o Zynq 7020 dev board (PetaLinux): server for monitoring/control using SoC’s programmable logic (XADC, GPIO, ...)
  o ZU19 (Yocto): server under development for publishing ATLAS gFex status data
  o Raspberry Pi (Yocto): made independent RPi Linux distribution with OPC-UA server
  o qemu (Yocto)

► Documentation for build process is available

Natively built on SoC (ATLAS MuCTPi project):

► CentOS 7 natively boots on the SoC (see yesterdays presentation from Panagiotis Papageorgiou)

► Successfully built a quasar-based server on ZCU102
  o Equipped the quasar server with hardware monitoring (2-hour effort) and added monitoring of I2C sensor on SoC
  o Reached publishing frequency of 1.8 kHz, imposed by I2C readout rate, with marginal CPU usage

Cross-compiling quasar projects

► Tested in multiple environments
  o RCE (Reconfigurable Computing Element, Zynq 7k) from SLAC with ArchLinux software distribution
  o Raspberry Pi: Raspbian cross-compiler on a desktop and RPi’s sys-root
  o even cross-compiled for Windows using Linux desktop ;)

► Needs the compiler and the sysroot (rootsfs…)

Empty quasar server mem footprint

<table>
<thead>
<tr>
<th>Platform</th>
<th>RSS (MB)</th>
<th>Exec size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86_64 (CC7, laptop)</td>
<td>7.5M</td>
<td>2.6M</td>
</tr>
<tr>
<td>armv7l (PetaLinux, Zynq)</td>
<td>12M</td>
<td>1.4M</td>
</tr>
</tbody>
</table>
Pointers to quasar

- Project page: https://github.com/quasar-team/quasar, distributed under LGPL
- ecosystem (optional modules), WinCC OA integration, C++ OPC-UA client generation facility
- Mailing list: quasar-developers@cern.ch
- Documentation, video tutorials, quasar papers etc.: https://github.com/quasar-team/quasar/wiki
- Regular developer meetings, happy to receive contributions or feedback!
- Support: community effort, JCOP support for DCS systems

 ESV Extensive tutorial for quasar on SoC in the following contribution (by Piotr Nikiel)!
SoC / OPC UA Network Integration

- Although OPC-UA traffic can be secured, SoC devices should still be secured to mitigate risks and avoid necessity of frequent PC-like software maintenance.

- Need for isolating SoC even within experiment network (EN).

- Possible solution being explored: use LanDB sets to restrict SoC access to trusted back-end machines.
Conclusions

- Various applications / use cases for SoCs in the context of control systems
- Embedding of OPC UA on SoCs allows convenient and flexible integration
- quasar framework available for SoC platforms which generates software based on object-oriented device models
- Significantly reduces software development effort on SoC (server) and back-end (client)
- Support and experience with quasar for Yocto/PetaLinux and native ARM CentOS, cross-compilers
- Network integration of SoC for controls is a challenge not to underestimate
Backup
Why SoC on ATCA Blades for slow control?

- Parameters to be monitored on each blade are numerous: potentially more than hundred, not all of them accessible by the IPMC, e.g. optical transceiver power, temps

- IPMC path via shelf manager
  - Entirely proprietary software on shelf manager (also embedded Linux-like OS), vendor-locked
  - Bottleneck of IPMC communication based on IPMI over I2C, limits monitoring scalability (ok for up to few tens of parameters continuously monitored in an entire shelf at 1Hz)
  - Load on shelf manager and IPMB which is not primarily done for external monitoring/control but for self-consistent management of the shelf itself

- SoC with Ethernet interface provides independent link to DCS

- SoC is often anyway foreseen for run control / configuration / DAQ monitoring tasks

- SoC allows flexibility on the board to get data from I2C buses or high-speed chip-to-chip interfaces (e.g. via programmable logic on FPGA and SoC) which are anyway present for data or configuration handling
quasar system-wide example

based on simplified existing project

hardware 1
HAL1
OPC-UA server type “1”

high-throughput link with calibration data

UaO client for “1”
DAQ’s group calibration+configuration software in C++

WinCC OA

hardware 2
HAL2
OPC-UA server type “2”

hardware connectivity
OPC-UA

UaO client for “1”

UaO client for “2”

OPC-UA server type “3”

“peripheral server” in cascaded server concept