RCE Based Readout and Evolution

Matthias Wittgen, 13.06.2019
Reconfigurable Cluster Element (RCE) is a SLAC R&D project for a generic DAQ/trigger system

- ATLAS: CSC, AFP, Pixel IBL/ITk test setups
- Other: LCLS experiments, LSST, HPS, proto DUNE

RCE SoC Workshop 2009 at CERN: https://indico.cern.ch/event/57836/

- ATCA packing as well as spin-off standalone high-speed I/O (HSIO) board
- SoC boards with custom Rear-Transition Modules (RTM) specialized for user applications
- First generation (Gen 1, 2007) based on Virtex4/PPC405 SoC
- Second generation (only some prototype boards, 2011) based on Virtex5/PPC440
- Current third generation (Gen 3, 2013) based on Zynq-7000 SoC
  - Cluster-On-Board (COB) - ATCA board
  - HSIO2 - standalone readout board
Evolution of RCE Operating Systems and Middleware

- **Gen 1** runs **RTEMS**, a open source real-time operating system
  - RTEMS Board Support Package for Virtex4/5 developed at SLAC
    - Based on Xilinx support libraries
  - Some ATLAS **TDAQ SW** packages were ported to PPC/RTEMS for board control configuration and monitoring
    - CORBA (omni), IPC, OH, IS
    - **CMT** used for RTEMS cross-compilation
- **Gen 3** (ARM32) runs **RTEMS** or **Linux**
  - RTEMS **without SMP support**: only one out of two ARM cores utilized
  - RTEMS RCE development abandoned to focus on Linux
    - Support for some legacy applications
- Evolution of middleware
  - **TDAQ/IPC** to Remote Call Framework (**RCF**) to **lightweight** custom solutions
RCE Gen 1 Usage in ATLAS Pixel Detector

- RCE Gen 1 used for Insertable-B-Layer (IBL) Stave Testing
  - Readout of FEI-4 pixel modules
  - Full pixel distributed pixel calibration system implemented with ATLAS TDAQ SW like IPC, OH, etc. on RTEMS
  - Application Example: Threshold S-Curve fitting on PPC405 (no FPU)
Gen 3 COB with Custom RTM

- Zynq-7000
- Up to 4 Data Processing Modules (DPMs) with two RCE each
- Data Transmission Module (DTM) for network connectivity
ATLAS Cathode Strip Chambers (CSC) Readout

Front View: COB

Generic COBs

ATCA shelf-manager integrated with DCS

Rear View: RTM

Application specific RTM
Connections (G-Link) to detector front-end modules in the back: G-Link S-Link to Readout System
Cathode Strip Chamber (CSC) Readout

- **DPMs** running RTEMS
  - Software Processing between FW blocks (data link from detector and S-Link)
  - Feature Extraction and Data Formatting
  - Highly optimized C++ code running only on one CPU core
- **DTM** running ArchLinux
  - Software Monitoring Tasks
- **No direct integration** into run control
  - Host PC translating state transitions (IPC to SUN/RPC)
  - BOOST property trees/JSON used as configuration DB storage and serialization format
CSC Run Control Integration

- **CSC integration** (control, configuration and monitoring)
**HSIO2 Board**

- I/O Carrier Board designed around a **DTM mezzanine** card
- Standalone non-ATCA version with RTM option
- Artix (access to RTM) configured and controlled by Zynq-7000 (DTM)
  - Packet based custom protocol (**PGP** - “Pretty Good Protocol”) used between Artix and DTM
ATLAS Forward Proton (AFP) Detector Readout

- **Gen 3 HSIO2** reading out **FEI-4** (front-end) pixel modules
- HSIO2 is running **ArchLinux**
- First iteration used a **TDAQ port** (**PMG, IPC**, etc.) to run HSIO2 directly in the ATLAS partition
- Replaced by Remote Call Framework (**RCF**)
  - Link to RCF
  - Concern about network isolation for RCE
HSIO2 Readout for RD53 Pixel Chip

Front-End-Board (FEB)
Two connectivity choices

- PGPv3 protocol
- RSSI protocol (reliable UDP)

Server with 10G Ethernet Card

FW/SW library provides various connectivity choices
HSIO2 can be quickly adapted for multiple applications
Next generation “HSIO” will use ZCU102 (Xilinx evaluation board)

AXI DMA engine not quite working yet for ARM64

FPGA Mezzanine Card (FMC) already available and being tested with a KCU105

FMC with 4 mini display ports replacing the FEB
Successfully tested
1.28 Gb/s readout of RD53
Firmware and Hardware Abstraction Library

- Firmware blocks for RCE
  - [https://github.com/slaclab/rce-gen3-fw-lib](https://github.com/slaclab/rce-gen3-fw-lib)
  - Example COB targets:
    - [https://github.com/slaclab/rce-gen3-fw-examples](https://github.com/slaclab/rce-gen3-fw-examples)
  - FW building blocks for a wide range of Xilinx FPGAs/SoCs (RCE like, PCIe cards)
- **Ruckus**: Vivado Build System based on Makefiles
  - [https://github.com/slaclab/ruckus](https://github.com/slaclab/ruckus)

- Hardware Abstraction SW library
  - Rogue
    - [https://github.com/slaclab/rogue](https://github.com/slaclab/rogue)
    - **Python bindings** for rapid DAQ prototyping and GUIs
    - Can be configured for pure C++ use
  - Abstraction Example
    - RD53 Front-End-Board Readout can use 10G Ethernet with the RSSI (reliable UDP) protocol or PG Pv3
Thoughts on OS Evolution

- RCE project started with RTEMS
  - Outdated network stack based on BSD (no IPV6 headers)
  - Although POSIX compliant hard to port most modern applications
  - Without RTOS certain features need to move into FW
- ArchLinux
  - Rolling releases
  - No scheduled releases make distribution of common middleware and applications a challenge
    - Used to distribute a ArchLinux snapshot
- Requirements for an embedded OS
  - Scheduled releases
  - Cross-install capability
  - Well maintained binary distribution for ARM32/64
- Settled on CentOS7 as it is the CERN default Linux
- Fedora a viable alternative of newer packages are needed
  - Alternatively LCG support for ARM32/64
- Watch out for ARM32 support vanishing
Thoughts on Middleware

• Tried different *middleware and OS ports* to control/monitor SoCs
  • ATLAS **TDAQ IPC** (AFP, pixel): **IDL based**
    • Would have exposed embedded system directly to the ATLAS partition
  • Now **IPC proxy** is available to isolate embedded systems
  • Switched to **RCF** (Remote Call Framework), **no IDL, but programmatic serialization**
    • Sun/RPC with CSC used JSON/RPC but only one XDR (data format) functions signature: text in and out
  • **Push of monitoring data** (e.g. UDP/JSON) preferable over poll method mainly implemented in pcph1ct10
  • Ideally want to **separate transport layer from data format**
Lightweight Middleware Approach

• **ZeroMQ** as transport layer
  • Implements a **wide-range** of **network patterns**
  • Sufficient for monitoring and control

• Other choices available
  • **NetIO** (used for FELIX)
  • Socket library similar to ZeroMQ

• Reliable SLAC Streaming Interface (RSSI)
  • **UDP based**, available in FW/SW
  • Aimed at **high performance point-to-point** data streaming

• JSON-like C++ memory structure that preserves data-typing
  • Can serialize itself into various data formats
  • **JSON**, **MSGPACK**
  • `schema-less`
  • Also used to be stored directly in configuration database as BLOB

• Lots of choices and combinations of transport layers and serialization formats, but
  • **Only one memory representation**
  • Suitable to describe histogram on embedded systems
    • **Conversion to ROOT outside of embedded systems**
C++ Recursive Variant as a Generic Data Structure

Memory efficient recursive C++ variant class, which translates directly into msgpack
C++17 required

Scalar types stored directly in variant

- unique pointer to map<string, value_t>
- unique pointer to vector<value_t>
- unique pointer to string

Unique pointer to vector<scalar_type> for homogenous arrays
Variant Application Example

Text Files

C++ variant

```cpp
variant v;

v["Vc"] = std::vector<float> {
    -9.111073e-07,
    -3.726413e-07,
    -1.472652e-06,
    9.132476e-08,
    2.268095e-06,
    3.840781e-07,
    2.412659e-06,
    -2.396183e-06,
    1.162529e-06,
    4.364388e-07,
    -1.451580e-06
};

v["Vd"] = std::vector<float> {
    1.043219e-01,
    -8.074539e-03,
    -4.467762e-02,
    -1.351129e-03,
    -1.141260e-02,
    7.554598e-03,
    -9.032832e-03,
    5.823742e-03,
    -5.865677e-03,
    4.069965e-03,
    3.157052e-03
};
```

MSGPACK (DB BLOB)

Flat memory structures that get reinterpreted by classes implementing functionality

Example: Map to Eigen vector (zero-copy)
RCE Software Environment Components

- **Supported Hardware**
  - COB: DTM, DPM, HSIO2 and ZCU102
  - Zynq-7000, Ultrascale+ (ARM32 and ARM64)

- **Software Development Kit** provides various open-source tools and scripts as RPM
  - *Crosstool-ng* to generate cross compilers
  - *ARM* and *AARCH64* cross compilers
  - *Xilinx kernel* with SLAC modifications
    - *10G ethernet support* for COB
  - *AXI stream DMA* drivers (ARM to PL interface)

- **ATCA tools**
  - Interacts with ATCA shelf manager through IPMI
    - *Reset* RCEs, *shelf monitoring* and *power management*
  - Python scripts to *cross-install CentOS7* and to create a bootable SD card for all supported hardware platform

- RPM SPECS, scripts, etc: [https://gitlab.cern.ch/rce/rcesdk](https://gitlab.cern.ch/rce/rcesdk)
Conclusions

- OS and Software choices are the result of a 10 year evolution of the RCE platform
  - Avoid building your own Linux distribution
    - Exception from the rule: IBLROD/Linux/PPC
  - Avoid rolling release Linux distributions
- Keep the interface to the SoC simple
  - No need for ROOT on ARM or 50 TDAQ packages
- Push monitoring data instead of polling
- Exchange Data formats and interfaces matter, not specific software implementations
ZCU102 Application Example
Cross Compilation Workflow

Note: the cross-compiler generated with crosstool-ng uses compiler headers and libraries of devtoolset-7 cross-installed to the ARM ROOTFS
Simple RPC Echo Example

CentOS7/x86

Encode

MSGPACK BLOB

ZMQ/RPC

CentOS7/x86

C++ Variant

ZCU102/CentOS7

MSGPACK BLOB

Decode

C++ Variant

ZMQ/RPC

MSGPACK BLOB

Modify

C++ Variant

ZMQ/Resp

MSGPACK BLOB

Encode

C++ Variant

MSGPACK BLOB

Decode

C++ Variant
Bootstrapping a ZCU102

Install the RCE SDK repository definition for yum

```bash
```

Now the RCE SDK can be installed

```bash
sudo yum install -y rce_sdk
```

Run the global RCE SDK setup script

```bash
source /opt/rce/setup.sh
```

The RCE SDK sets up dandified yum (DNF) and installs a statically linked version of QEMU to emulate an ARM CPU.

To run the script to create a cross installed CentOS7 root filesystem for ZCU102

```bash
sudo mkdir /opt/rce/rootfs/
mkrootfs --root=/opt/rce/rootfs/centos7_64 --arch=aarch64
```

List available SD cards:

```bash
mkrcesd -l
```

SD card device candidates found:

<table>
<thead>
<tr>
<th>Device</th>
<th>Size [GB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdg</td>
<td>31.914983424</td>
</tr>
</tbody>
</table>

Copy a root file system from the host to an SD card:

```bash
mkrcesd -v --root=/opt/rce/rootfs/centos7_64 ZCU102 sdg
```
Run Echo Server Example

Simple ZeroMQ Echo Server
https://gitlab.cern.ch/wittgen/zmqrpc

Clone
git clone --recurse-submodules https://:@gitlab.cern.ch:8443/wittgen/zmqrpc.git
cd zmqrpc/
mkdir build.x86_64 build.arm64

Setup devtoolset-7 and RCE SDK
source /opt/rh/devtoolset-7/enable
source /opt/rce/setup.sh

Point tools to CentOS7 ROOTFS to use
export CENTOS7_ARM64_ROOT=/opt/rce/rootfs/centos7_64

Build application
(cd build.x86_64 && cmake3 .. && make -j4 )
(cd build.arm64 && cmake3 .. -DCMAKE_TOOLCHAIN_FILE=../toolchain/rce-arm64 && make -j4 )

Copy to SD card on ZCU102 and run
scp `build.arm64/server` zcu102

Run server on ZCU102
ssh zcu102 server

Run client
build.x86_64/client -c tcp://zcu102:5555 -n 10 -s 1000