The CMS Inner Tracker DAQ system for the High Luminosity upgrade of LHC: from single-chip testing, to large-scale assembly qualification

Outline
- The pixel detector for the LHC upgrade
- Hardware for module testing
- DAQ software
- Summary

• DAQ architecture
  • Calibrations
  • Using external devices
  • GUI

• Monitoring
• Multithreading
• Toward a distributed system

Mauro Dinardo
Università degli Studi di Milano Bicocca and INFN, Italy
on behalf of the CMS-DAQ working group

11/05/2023 Norfolk
High Luminosity upgrade of LHC (~2029)
Increase instantaneous luminosity to improve discovery potential of ATLAS and CMS experiments

**Aim:** preserve physics object performance despite average number of interactions per crossing up to 200 → experiments need significant upgrade

Design constraints for the InnerTracker (IT)
- Total n. pixels ~2 billion
- **Trigger rate ~750 kHz** → high speed throughput
- Hit rate ~3 GHz/cm² (innermost layer) and trigger latency 12.8 μs → buffer depth
- Unprecedented radiation level (~2x10^{16} n_{eq}/cm², 1 Grad, for innermost layer)

**InnerTracker ReadOut Chip (ROC)**
- ASIC in 65 nm CMOS technology
- Module flavours: 1×2 ROCs and 2×2 ROCs
- **336x432 = 145 152 pixels**, zero suppression, 50x50 μm²
- **Time-over-Threshold** hit charge measurement with 4 bits ADC
- Global threshold with **per pixel 5 bits threshold trimming**, dual-slope gain
- **Data compression**, a.k.a. binary-tree encoding, to save 30% bandwidth
**InnerTracker ReadOut Chip (ROC)**

- Clock, trigger, commands to ROC @160 Mbps
- Multiple lane readout up to 3.84 Gbps (1.28 Gbps x 3 lanes)
- Data merging capabilities:

**InnerTracker optical readout**

- Data from modules are sent via twisted pairs, or flex cable, to Lowpower GigaBitTransceiver (LpGBT) chip
- LpGBT sends (receives) data to (from) a custom System-on-Chip board, a.k.a. Data, Trigger and Control - DTC via optical fiber at 10 Gbps (2.5 Gbps)
Ph2-ACF is a C++ software designed to handle both OuterTracker (OT) and InnerTracker (IT) hardware.

**Middleware API:** wraps firmware calls and handshakes into C++ functions, i.e. object-oriented libraries describing system components, readout chips, portcard/hybrids, micro DTCs), and their properties (values, status)

- **Description classes:** containing device IDs, configuration, local register caching, etc
- **Interface classes:** functions for sending commands and receiving data

**Scans and utilities**
- read/write configuration files
- perform calibrations
- monitor the hardware
- **DetectorContainer**, i.e. a generic type data structure mapping the relations between the different hardware components

In particular Ph2-ACF contains a set of general classes

- **SystemController**: main class holding references to all required objects
- **FileParser**: class to read XML configuration file ➔ instantiate all required objects
- **Tool**: class implementing functionalities common to all calibrations
- **RegManager**: class managing the IPbus calls
Software architecture: **Phase2 - Acquisition and Control Framework**

**Interface classes** make calls to `BoardFWInterface` which has the SW to FW command-translation.

Write from and read to the chip

- `ChipInterface`
- `LpGBTInterface`
- `OT_CBC_Interface`
- `OT_MPA_Interface`
- `IT_RD53A_Interface`
- `IT_RD53B_Interface`

Write from and read to the micro DTC board

- `BoardFWInterface`
- `OT_FWInterface`
- `IT_FWInterface`
- `OT_micro DTC`
- `IT_micro DTC`

**Description classes** allow for an abstract representation of the hardware hierarchy.

DetectorContainer: tree structure, based on `std::vector`, representing the detector hierarchy. A DetectorContainer object is instantiated and filled at configuration time, and the type of each layer are the description classes. Its reference, `fDetectorContainer`, is owned by the SystemController.

Legend

- Inheritance
- Software calls
- IPbus transactions

Legend

- Base class
- OT concrete class
- IT concrete class
- Physical hardware
Every calibration is a C++ class configured through XML file
- All calibrations inherit from the Tool class
- Some are basic calibrations, e.g. PixelAlive, SCurve, Gain
- Some are extensions and inherit from basic one, e.g. ThresholdEqualization

"Writing a new calibration is easy, just need to fill the proper member functions"

Physics data taking, e.g. at testbeams, is a special "calibration" that has no prior knowledge of the number of accumulated events
- Continuous polling of memory checking for new events until asynchronous STOP signal is issued
- Live plots during data taking
Concurrency is naturally solved by the TCP/IP client/server protocol

Need to coherently combine different data streams: i.e. HardwareInterface (e.g. from front-end chip calibrations) and ExtDeviceMonitorController in one unique file ➔ not trivial because have different “life span”:

- Calibrations ➔ from Start to Stop
- Monitoring external devices ➔ from “System-On” to “System-Off”
- Monitoring front-end electronics ➔ from “Configure” to “Destroy” … see next slide
Graphical User Interface and Monitoring

**Graphical User Interface - GUI**
- Based on Py-QT5 (Ph2-ACF decoupled from GUI)
- Interacting with Ph2-ACF via named pipes
  - Soon move to Ph2-ACF python bindings to C++ API, GUI will communicate with Ph2-ACF via Google protobuf

**Monitoring front-end electronics**
- Voltages, Currents, and Temperatures defined in XML file
- Data stored in ROOT files vs time
- Asynchronous with respect to other tasks
Example (noise measurement) of running-time gain thanks to data decoding parallelization:

- **decoding** is second most time-consuming operation, we gained ~10% on a 6 core CPU
- **IPBus transaction** is first most time-consuming operation, which can only be improved once we move to the SoC

- **Calibration thread**: runs a calibration while listening to commands like stop, pause, etc…
- **Decoding thread**: bit-unpacking and interpretation are intrinsically parallelizable tasks because data are logically and physically separated by triggers
- **Monitoring thread**: monitor frontend hardware asynchronously with respect to other tasks
Thinking bigger … toward a distributed system

Mauro Dinardo, Università degli Studi di Milano Bicocca and INFN
A glimpse of the final system

**DTC board for OT**
- 72 optical inputs
- Sends stubs to Track Finder on up to 48x 25Gbps links
- Sends event fragments to central DAQ over 4x 25Gbps links
- In total **216 DTCs** and **200 million channels**

**DTC board for IT**
- 72 optical inputs
- Sends event clusters to central DAQ over 16x 25Gbps optical links
- In total **36 DTCs**
- Up to 380 ROCs, ~**55 million channels** on average, per DTC

**OT module**
LpGBT VTRx+

**IT module**
LpGBT VTRx+

**DTC board for OT**
- 72 optical inputs
- Sends stubs to Track Finder on up to 48x 25Gbps links
- Sends event fragments to central DAQ over 4x 25Gbps links
- In total **216 DTCs** and **200 million channels**

**DTC board for IT**
- 72 optical inputs
- Sends event clusters to central DAQ over 16x 25Gbps optical links
- In total **36 DTCs**
- Up to 380 ROCs, ~**55 million channels** on average, per DTC

**Advanced-TCA-compliant custom board**
- Up to **2 Ultrascale+ FPGAs**
- CPU co-processor (**System-on-Chip**): x86-based Com-Express or Zynq-based mezzanine
To meet the higher requirements needed by the CMS experiment to cope with the LHC Luminosity Upgrade, the tracker underwent to a major R&D program.

The DAQ for the InnerTracker module testing (Ph2-ACF) was presented in terms of the overall architecture and capabilities:

- The Ph2-ACF software and firmware are still under development, nonetheless, they are already in a mature stage to support users to characterise the modules both on a test bench and with test beams.

Ph2-ACF is being designed with the aim of handling a distributed system with a great number of modules, similar to the final system.

**Summary**

**Dependencies**

- CERN ROOT: [https://root.cern/](https://root.cern/)
- IPbus: [https://github.com/ipbus/ipbus-software](https://github.com/ipbus/ipbus-software)
- boost, protobuf, pybind11
Summary

Ph2-ACF - SystemController

- GUI
- XML configuration
- Description classes
- IPbus libraries
- Core classes
- Interface classes
- Tool class
- Monitorings
- Calibrations
- TCP/IT server
- External devices

Legend:
- Yellow: Ph2-ACF - SystemController
- Red: Core classes
- Blue: Adapted to specific system
- Pink: Auxiliary
**InnerTracker module testing: the hardware (optical readout)**

- **“DIO5” FMC board:**
  - ext. trigger, clk., busy

- **Optical readout FMC board:**
  - up to 8 SFP connectors

- **Demonstrator board:**
  - Low-power GigaBit Transceiver chip

- **Optical fiber connection:**
  - 1 SFP = 2 fibers

- **Module: RD53A chip x 4**
  - Display Port connection
  - Flex cable connection

- **Nano crate with micro DTC board**

- **IPbus**
  - Control/Monitor DAQ

- **micro DTC:**
  - Xilinx Kintex 7 FPGA
  - two FMC connectors
  - DDR3 RAM 4 Gbit

- **FPGA firmware tasks:**
  - encode/decode chip commands
  - send bunch of triggers and injections
  - store data in memory buffer
  - handle external clock and trigger handshake
  - abstract communication-details to support multiple front-end hardware

- **Single Chip Card: RD53A chip**

- **Nano crate with micro DTC board**

- **Optical connection:**
  - 1 SFP = 2 fibers

- **15 Mauro Dinardo, Università degli Studi di Milano Bicocca and INFN**

- **InnerTracker module testing: the hardware (optical readout) with FPGA firmware tasks**

- **Module prototype**
  - IT port-card

- **Simple additions to describe the pin mapping of the wafer probing card in the uDTC project**

- **Preparations for CROC testing**
  - Modifications required to adapt interfaces to test systems

- **CERN FMC**
  - Kansas FMC
  - Commercial multi-SFP FMC
  - single chip readout
  - single chip/multi-module readout
  - optical readout

- **Optical connection**

- **Nano crate with micro DTC board**

- **SFP = 2 fibers**

- **Optical connection**

- **Nano crate with micro DTC board**

- **SFP = 2 fibers**

- **Optical connection**

- **Nano crate with micro DTC board**

- **SFP = 2 fibers**

- **Optical connection**

- **Nano crate with micro DTC board**

- **SFP = 2 fibers**

- **Optical connection**

- **Nano crate with micro DTC board**

- **SFP = 2 fibers**

- **Optical connection**

- **Nano crate with micro DTC board**

- **SFP = 2 fibers**

- **Optical connection**
InnerTracker module testing: the hardware (electrical readout)

**micro DTC:**
- Xilinx Kintex 7 FPGA
- two FMC connectors
- DDR3 RAM 4 Gbit

**Electrical readout FMC board:**
- up to 4 single chip or up to 3 quad-modules (module with 4 chips)
- access to Hit-Or signals

**"DIO5" FMC board:**
- ext. trigger, clk., busy

**“DIO5” FMC board:**
- ext. trigger, clk., busy

**Display Port connection**
- Single Chip Card: RD53A chip

**Nano crate with micro DTC board**

**IPbus**
- Control/Monitor DAQ
Ultimately a calibration sequence needs to:

• Iterate on one or more chip registers
• Inject charge and send triggers ➔ handled at firmware level
• Record the response (hit/no hit, pulse height)
• Run the sequence on a subset of channels and iterate over all subsets

Class Tool provides a series of member functions common to all calibrations:

• Tool::measureData(...) ➔ sends trigger + measure occupancy/charge
• Tool::scanDac(...) ➔ sends triggers + measure occupancy/charge + scan register
• ...

Calibrations need to store information (efficiency, register values, histograms) with same hierarchy of the detector, such a structure can be easily copied from fDetectorContainer into a DetectorDataContainer:

```
ContainerFactory::copyAndInitStructure<float>(*fDetectorContainer, *myContainer);
```

DetectorDataContainer: each layer of the tree can be of any type, and the data stored can be normalised and averaged
Ph2-ACF is being written to be used for module characterisation both on test bench and with test beams, but also aiming to be ported and applied to the final detector.

**Calibration procedures are divided into two parts**

- **Calibration loops** (run on “remote computer”)
  - front-end register scans and/or adjustments
  - data analysis with quantitative outcome (fully ROOT independent)

- **Plotting** (run on “local computer”)
  - plotting and data visualisation (ROOT dependent)

**Sequence of operations**

1. Users interact with the **MiddlewareInterface** (on the “local computer”) which sends state machine commands through the network to the **MiddlewareController** (on the “remote computer”)

2. The **MiddlewareController** decodes the commands and launch the corresponding calibration, which just needs to implement the base state machine commands: Start, Stop, Pause, Resume, etc...

3. On the “remote computer” the **DQMStreamer** serialise the data and sends them through the network

4. On the “local computer”, the **DQMInterface** collects the data and produces the plots
Integrate a new system into Ph2-ACF

The integration of a new system in Ph2-ACF, from the back-end board to the front-end chip, requires well-defined steps:

- Development of the concrete implementation of the FWInterface, ChipInterface, and Chip classes, satisfying the interface of the relative abstract classes.

- If needed, development of detector-specific calibration routines satisfying the interface of the Tool class, and taking advantage of the Container data type.

- Use of the XML configuration file describing the relationship of the different hardware components.