The ATLAS MUCTPI and System-on-Chip

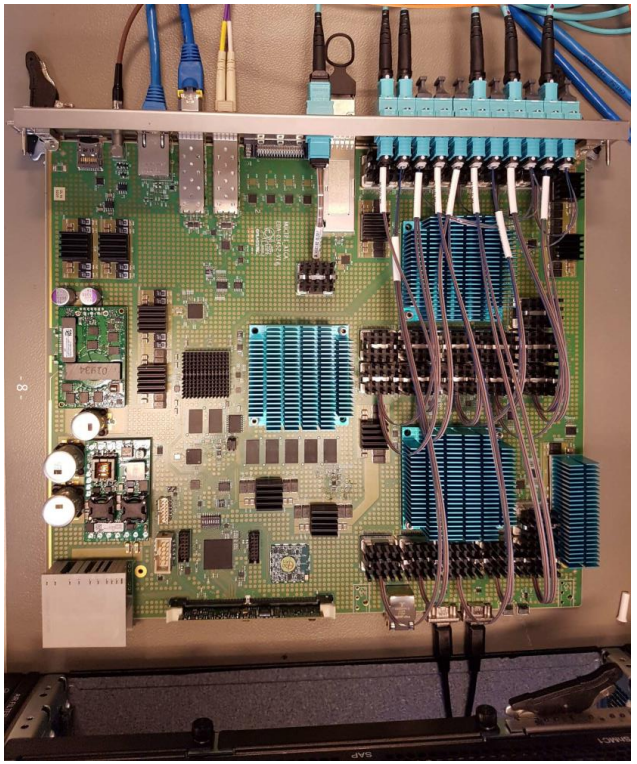
- Introduction
  - Operating System
    - Low-Level Software
    - Hardware Control
  - Run Control
- Summary and Outlook

ATLAS - MUCTPI

→ **Muon-to-Central-Trigger-Processor Interface**

= single ATCA blade,

upgrade project for Run 3, replaces 18 VME modules



- **Functionality of the MUCTPI:**
  - Data concentration of 208 muon trigger sector inputs
  - Overlap removal, i.e. avoid double counting of single muons
  - Provide muon trigger objects to topological trigger and muon threshold multiplicity to Central Trigger Processor (CTP)
  - Trigger processing implemented in high-end FPGAs with many high-speed links
- **Use of the SoC:**

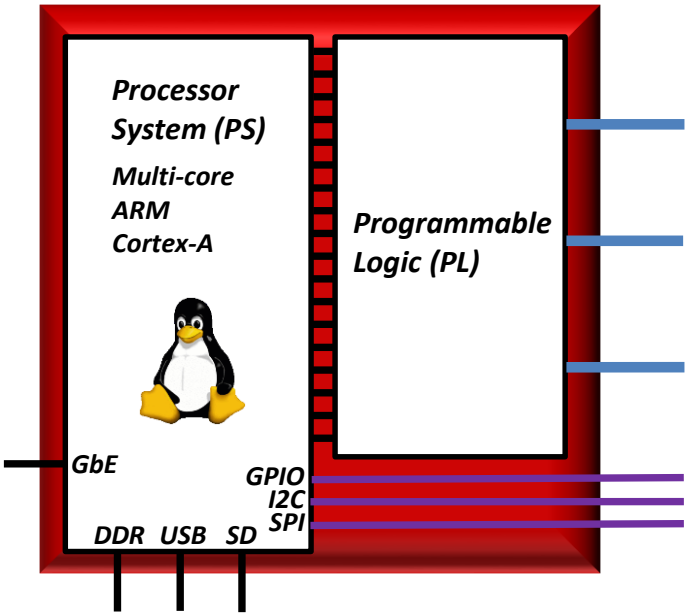
  Configuration, control and monitoring of the blade:
  - Hardware control of clock, power, and optical modules and configuration of FPGAs
  - Run control of processing FPGAs: read/write status/control registers and memories/LUTs, read counter registers
- **Several versions of prototypes:**
  - V1, V2 (available): Xilinx Zynq 7Z030,
    dual ARM Cortex A9 (armv7, 32-bit), 666 MHz, 1 GByte DDR3
  - V3 (June 2019): Xilinx Zynq Ultrascale+ MPSoC ZU3eg,
    quad ARM Cortex A53 (armv8/aarch64, 64-bit), 1.2 GHz, 4 GByte DDR4

System-on-Chip

System-on-Chip:

Processor system + programmable logic ⇒ *"CPU and FPGA"*

- **Processor system (PS) = like CPU:**
  - **Currently all are multi-core ARM processors**
  - **Has memory and peripherals, e.g. GbEthernet, I2C, SPI, GPIO, etc.**
  - **Runs software: "bare-metal" application or operating system, e.g. Linux**
- **Programmable logic (PL) = like FPGA:**
  - **Has logic cells, memory blocks, I/O links, and MGTs**
  - **Implements real-time data logic, interfaces to the other processing FPGAs, can implement more peripherals, e.g. 10GbEthernet, etc.**



→ Concentrate on software part …

Operating System (1)

→ Use Linux, we have not seen any need for a real-time operating system

- **Xilinx PetaLinux:**
  - **Simple to start with, but gets more complicated for adding user software**
- **Yocto + Xilinx meta layer:**
  - **Builds toolchain, i.e. cross-compiler and system libraries**
  - **Builds bootloader, i.e. U-Boot**
  - **Builds Linux kernel:**
    - Based on kernel sources from Xilinx: linux-xlnx
  - **Builds a root file system (rootfs):**
    - Can be populated with potentially a lot of software, you choose what you want, e.g. ssh, NFS, python, iptables, etc.
  - **Allows user software to be added in a very flexible way:**
    - Add your own recipe, re-use autoconf, Makefile or CMake etc.

Howto: https://twiki.cern.ch/twiki/bin/view/SystemOnChip/GettingStartedWithYocto

Note: Xilinx/Vivado and Xilinx/SDK are used to prepare other boot files:

- **Bitstream file:** for programmable logic
- **First-Stage Boot Loader (FSBL):** software to initialize hardware and to load bitstream file and bootloader
- **Device tree files:** used by the Linux kernel

Operating System (2)

Full operating system, provides:

- ssh for logon
- bash, python, tcl, expect, etc. for scripting
- NFS mount of other file systems, NTP for precise date and time
- Linux standard tools for network: ip, ifconfig, DHCP client, …
- iptables or firewalld for network security
- Linux standard tools for I2C, SPI, GPIO, e.g. *i2cdetect*, *spidev_test*, */sys/class/gpio*, …
- More can be added easily …

⇒ **Everything we expect from an operating system, just like a PC (x86_64)**

Alternatively: use CentOS/ARM for root file system:

- Run kernel produced by Yocto (or PetaLinux), to get all Xilinx drivers
- Cross-install a CentOS/aarch64 rootfs
- Add user software by cross-compiling with CentOS/aarch64 rootfs, or natively on ARM architecture

→ See presentations and demos on Thursday morning by P. Papageorgiou and M. Wittgen

Low-Level Software (1)

→ User application software to access all hardware features of MUCTPI:

- **Hardware configuration and monitoring:**
  - **Based on the Linux system interfaces (Linux devices) for I2C, SPI, and GPIO.**
  - **Libraries to describe the MUCTPI specific components and their registers.**
  - **Allows to configure and monitor hundreds of parameters for voltage, current, temperature, optical power values, clock settings, etc.**
- **FPGA configuration:**
  - **Xilinx Virtual Cable (XVC) = Xilinx software to run JTAG commands.**
  - **Slave Serial Configuration for FPGAs:**
    software program which writes a bitstream file to a firmware block which configures FPGA, actually used to configure all processing FPGAs of the MUCTPI in < 5 sec.
  - **xdevcfg for Zynq/PL: possible to re-configure PL part of SoC after booting Linux if required.**

Low-Level Software (2)

→ User application software to access all hardware features of MUCTPI:

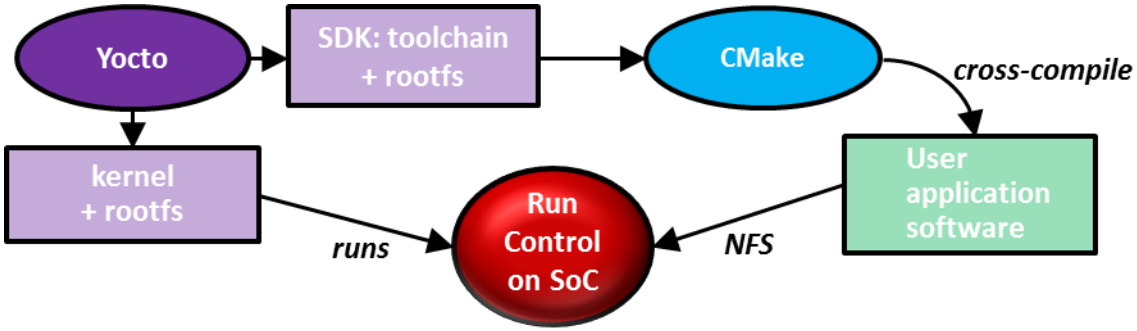
- **Access to FPGA resources:**
  - **Memory-mapped access to all FPGA resources using */dev/mem* (possibly UIO in the future), i.e. to AXI Chip-2-Chip to processor FPGAs or local AXI to Zynq/PL**
  - **Tools *peek* and *poke* for interactive access**
  - **Developed a kernel module and a library for contiguous memory support and DMA**
- **"HardwareCompiler":**
  - **Describe all registers with their bit fields, memories and FIFOs implemented in AXI space in an XML file**
  - **Compile from XML file and produce:**
    - VHDL code for firmware development
    - C++ code for software development

  ⇒ Allows co-development of firmware and software

Low-Level Software (3)

Build low-level software:

- Either use with Yocto as other packages with their own recipe ⇒ meta-l1ct layer
- Or use the Yocto SDK* = toolchain (cross-compiler) + rootfs, with other build environment, e.g. CMake



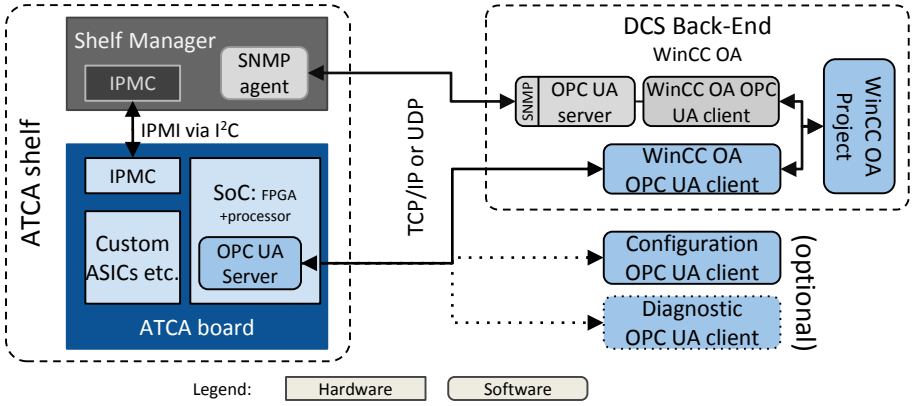
- Alternatively: use the Yocto toolchain (cross-compiler), and CentOS rootfs for building

- *SDK = Software Development Kit

Hardware Control Software

→ Detector Control System (DCS)

- **MUCTPI/IPMC → Shelf manager → DCS:**
  - Used for limited number of critical values, mandatory control of ATCA shelf
  - The shelf manager takes automatic actions, e.g. regulating fan speed
  - Path to DCS is based on SNMP running on shelf manager
  - DCS is used to display values, sends alarms (without actions), allows one to power on/off the blade and control the fan speed manually if required



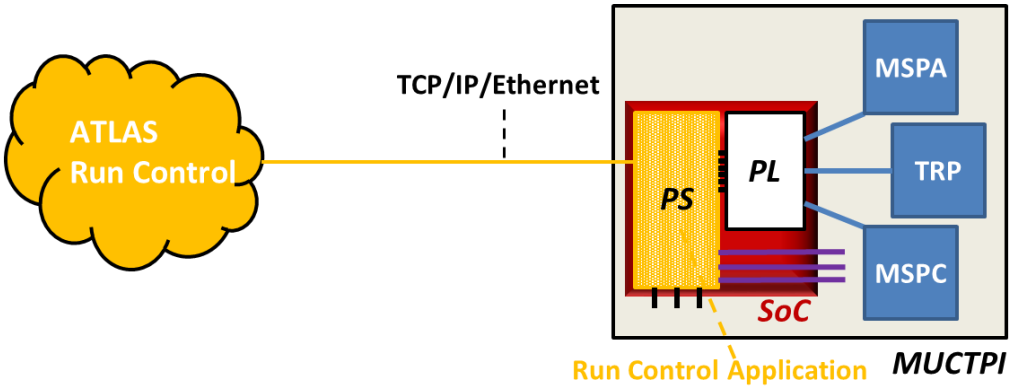
- **MUCTPI/SoC → DCS:**
  - Much larger number of complementary values: not necessarily critical, more run oriented, e.g. optical input power from Muon Sector Logics
  - Based on low-level software, e.g. I2C, SPI, etc.
  - Send data to Detector Control System (DCS) for display, alarms, logs, etc.
  - Have tested successful running of an OPC-UA server on SoC

→ See presentations and tutorials on Friday morning by S. Schlenker and P. Nikiel

Run Control Software (1)

→ **Run control application using MUCTPI low-level software:**

- **Run a run control application on SoC:**
  - **Cross-compile TDAQ* packages for SoC**
  - **Develop run control application for SoC using the low-level software**
  - **This scheme works exactly as before with the SBC in the VME crate**

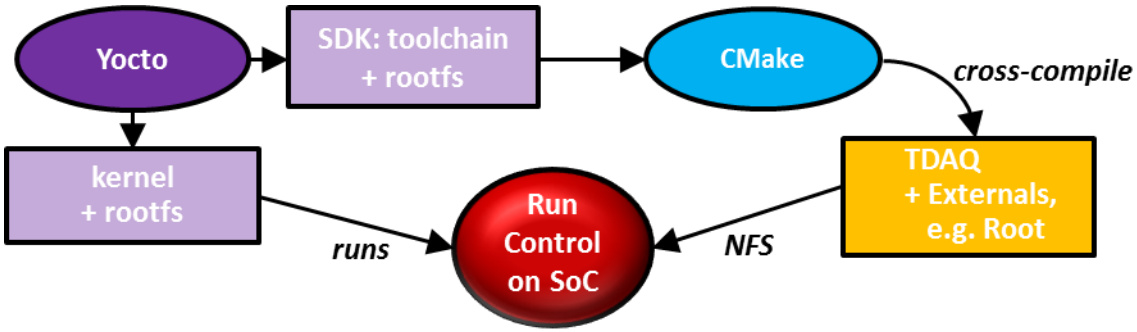


*TDAQ software = all software for Run Control Services

Run Control Software (2)

Build run control software:

- Compiling TDAQ with Yocto is difficult because directory structures are very different
- Use the Yocto SDK = toolchain + rootfs, with run control build environment, i.e. CMake

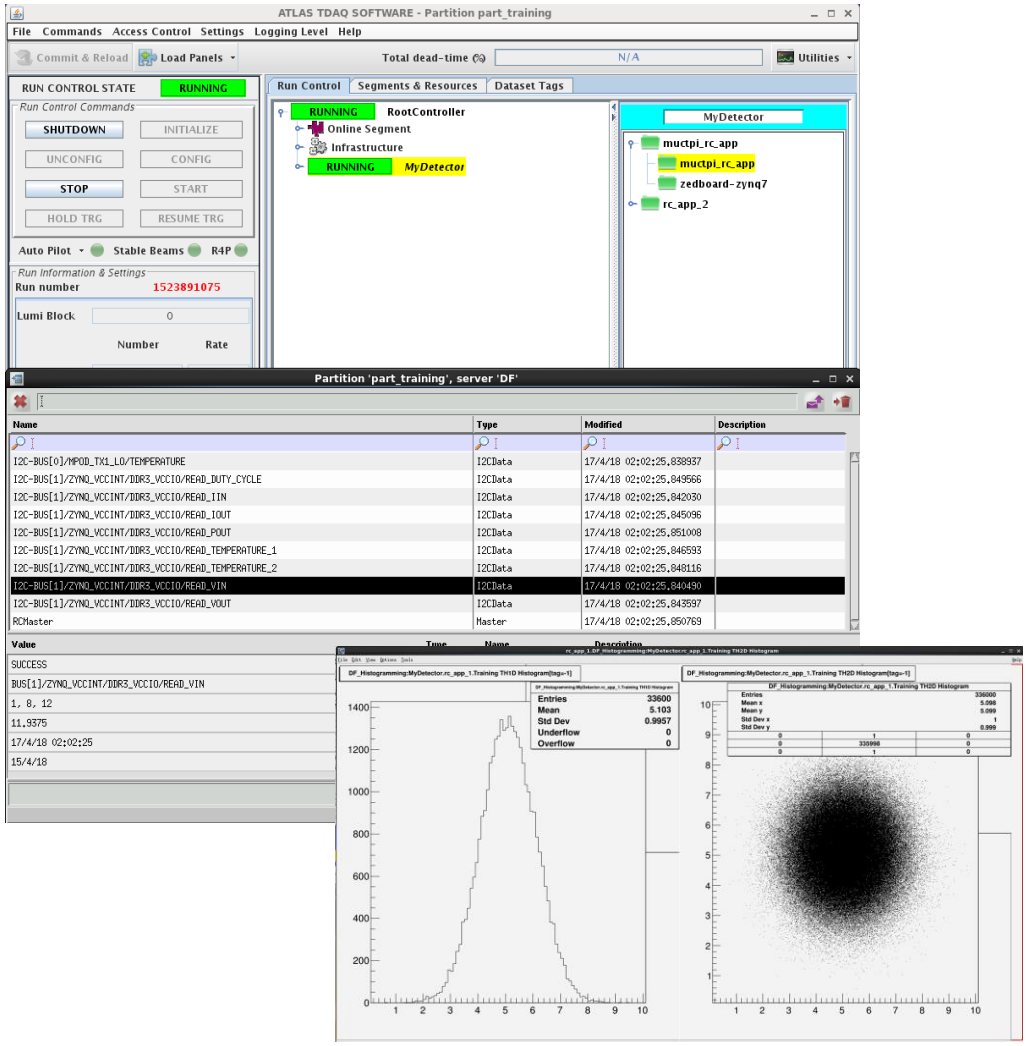


Experience building run control software:

- Almost no changes to source code required, exception *rcc_timestamp* for reading CPU time
- Select what is being built, only packages required for a run control application
- Do not build JAVA software for SoC, but still have to specify JAR files, so point to existing JAR files in ATLAS and SFT cvmfs repositories
- Some patches to CMake files required, mainly for code generators, which run natively
- Same remarks apply to compiling ROOT

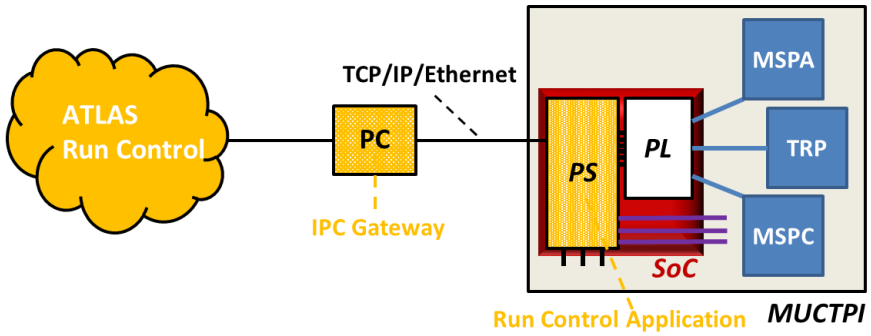
Run Control Software (3)

- **Works!**
- **Run control application:**
  - Responds to state change commands
  - Reads configuration
  - Writes monitoring data
- **Have cross-compiled ROOT:**
  - Provides histograms and graphs (***shown are dummy examples, the real firmware is being worked on***)



Run Control Software (4)

- **ATLAS is on a technical control network (ATCN):**
  - **Our operating system is not a CERN certified one**
  - **MUCTPI/SoC has to be isolated on a private network behind a host PC, which has an interface in ATCN**
- **IPC Gateway:**
  - **IPC = Inter Process Communication, in ATLAS it is based on CORBA**
  - **IPC Gateway program (thanks to S. Kolos):**
    - Runs on the isolating host PC
    - Forwards IPC/CORBA requests from ATCN to the isolated network and vice versa
    - Does not require modifications to existing software or configuration databases
    - **Was tested successfully in a lab setup!**



Summary

Working solution for MUCTPI in Run 3:

- **Build kernel using Yocto/meta-xilinx**
- **Build rootfs using Yocto or CentOS/aarch64 (preferred)**
- **Build low-level software by cross-compiling against the rootfs**
- **Run an OPC-UA server for hardware control based on the low-level software**
- **Run a run control application based on the low-level software,**
  together with an IPC gateway program on host PC to fulfil network security requirements

Outlook

Open questions:

Support, in particular, long-term support for hardware and software?

⇒ When looking at phase 2 (Run 4) we might have potentially o(100-1000) systems in ATLAS using SoCs!

- **Common hardware?**
  - **Could be helpful to make recommendations for common hardware**
  - **Providing a common SoC mezzanine might be useful, in particular, to overcome hardware obsolescence**
- **Common software?**
  - **Currently there is a variety of operating systems:**
    - Who will provide maintenance and long-term support?
    - Will not be maintained by experiments' system administrators
    - Will not be allowed to connect on a technical control network, e.g. ATLAS or CMS networks; will have to be isolated behind a networking switch and/or a host PC.

⇒ Provide a common operating system:

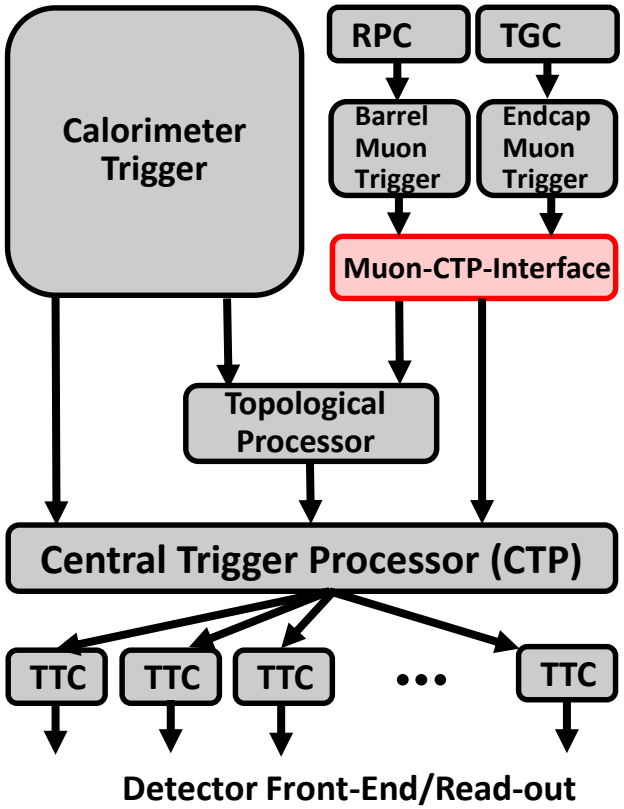
- **CentOS/aarch64: see presentation and demos by P. Papageorgiou and M. Wittgen on Thursday morning**

Additional Material

ATLAS – MUCTPI (1)

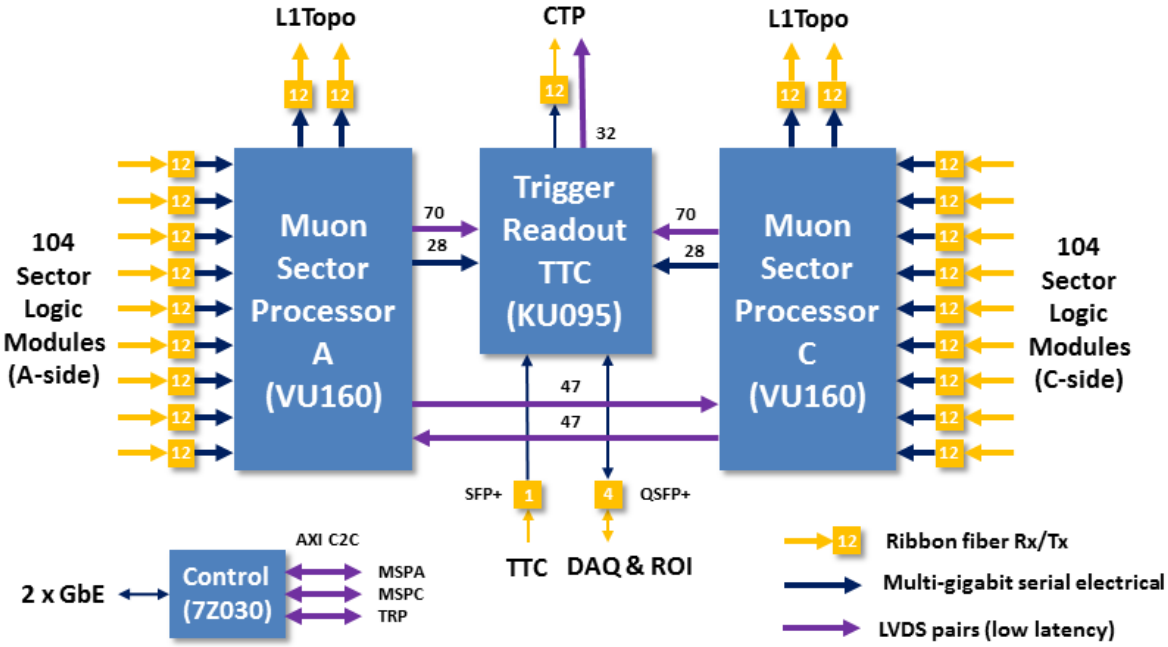
→ Muon-to-Central-Trigger-Processor Interface

- **Functionality of the MUCTPI:**
  - Data concentration of 208 muon trigger sector inputs
  - Overlap removal, i.e. avoid double counting of single muons that are detected by more than one chamber due to geometrical overlap of the chambers and the trajectory of the muon in the magnetic field
  - Provide muon trigger objects to topological trigger and muon candidate multiplicities to Central Trigger Processor
  - On a Level-1 Accept, provide all muon candidates in a time window to DAQ for readout
  - Provide muon trigger rates and per-bunch histograms per sector for monitoring

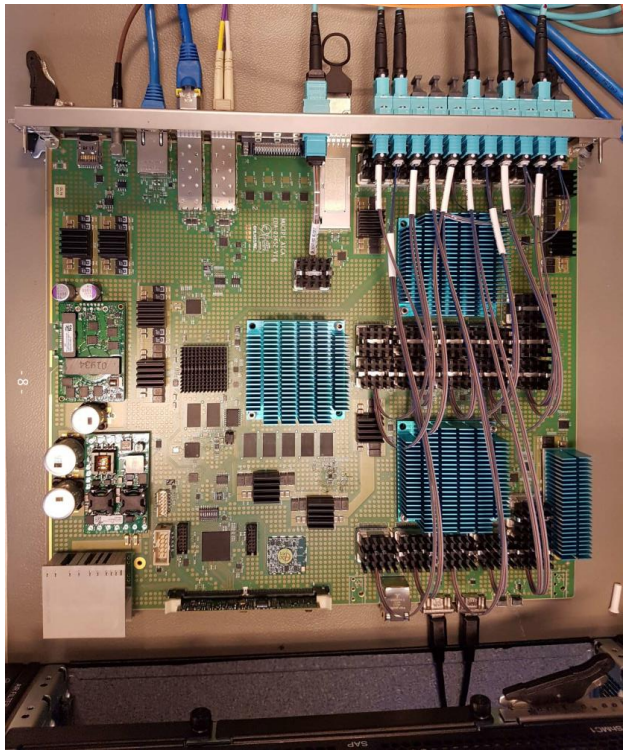


ATLAS – MUCTPI (2)

- **Implementation:**
  - **Single ATCA blade, instead of a VME crate with 18 modules as before**
  - **High-speed optical links on input and output, almost 300 links at 6.4 GB/s**
  - **Two high-end FPGAs (Muon Sector Processor, MSP, Xilinx Ultrascale+ Virtex) for overlap handling , counting, and providing muon candidates to Topological Processor**
  - **One FPGA (Trigger&Readout Processor, TRP, Xilinx Kintex Ultrascale) for total count of muon candidates and readout of trigger information**
  - **One System-on-Chip (Control Processor, Xilinx Zynq) for control, configuration, and monitoring**



ATLAS – MUCTPI (3)



- **Use of the SoC:**

  Configuration, control and monitoring of the board:

  - **Hardware control of clock, power, and optical modules and configuration of FPGAs**
  - **Run control of processing FPGAs: read/write status/control registers and memories/LUTs, read counter registers**

- **Several versions of prototypes:**

  - **V1, V2 (available): Xilinx Zynq 7Z030, dual ARM Cortex A9 (armv7, 32-bit), 666 MHz, 1 GByte DDR3**
  - **V3 (JUN-2019): Xilinx Zynq Ultrascale+ MPSoC ZU3eg, quad ARM Cortex A53 (armv8/aarch64, 64-bit), 1.2 GHz, 4 GByte DDR4**

Run Control Software (alternative)

→ **Run control application using MUCTPI low-level software:**

Alternative approach

- **Using RemoteBus:**
  - **Similar to Remote Procedure Call (RPC)**
  - **C++, simple stack for arguments**
  - **Multiple clients, mutli-threaded server**
  - **Queuing of requests to mitigate overhead**