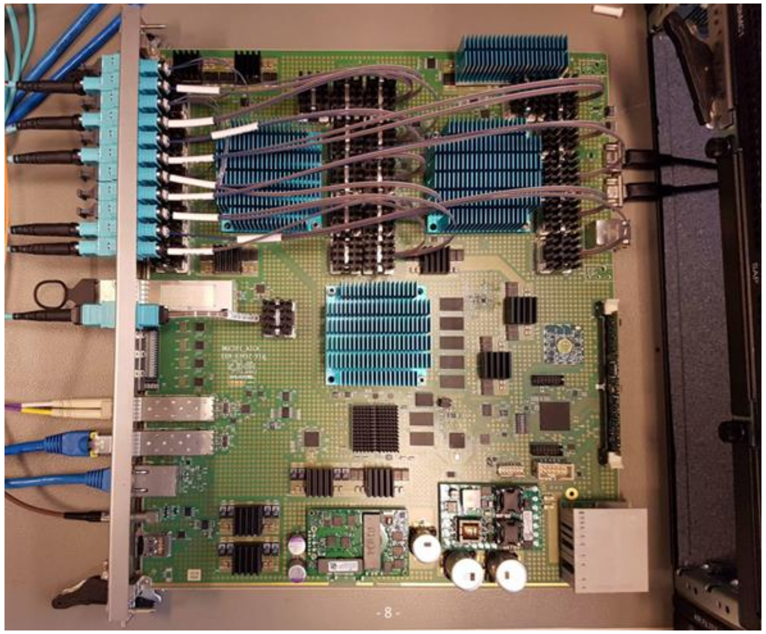
Software Framework for the System-on-Chip of the ATLAS MUCTPI

On behalf of ATLAS Level-1 Central Trigger (L1CT) Team.

ATLAS - MUCTPI

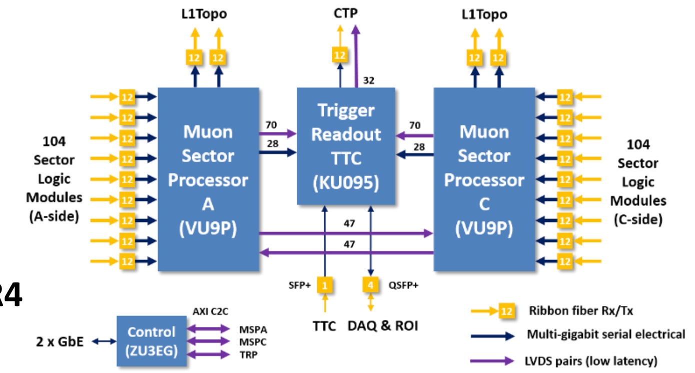
→ **Muon-to-Central-Trigger-Processor Interface** - upgrade project for Run 3

= single ATCA blade, 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
- **Control of the MUCTPI:**

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

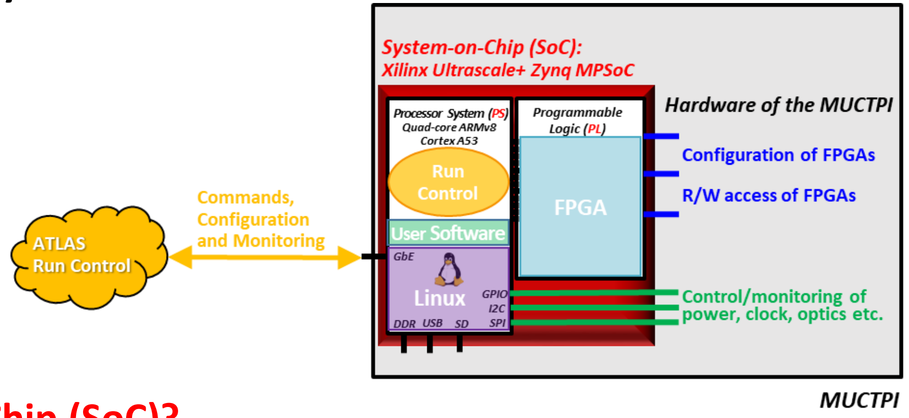

L1Topo
CTP
L1Topo
12
32
70
28
104 Sector Logic Modules (A-side)
Muon Sector Processor A (VU9P)
Trigger Readout TTC (KU095)
Muon Sector Processor C (VU9P)
104 Sector Logic Modules (C-side)
47
47
SFP+
1
4
QSFP+
TTC
DAQ & ROI
AXI C2C
2 x GbE
Control (ZU3EG)
MSPA
MSPC
TRP
Ribbon fiber Rx/Tx
Multi-gigabit serial electrical
LVDS pairs (low latency)

System-on-Chip

Why use a System-on-Chip (SoC)?

Previously, we had a Single-Board Computer (SBC) in the VME crate

SBC was running Scientific Linux CERN (SLC), the user application software was fully integrated into the ATLAS TDAQ system



What is a System-on-Chip (SoC)?

⇒ System-on-Chip (SoC) = FPGA + CPU *(note, that this is a slightly more restrictive definition than usual)*

- **Programmable logic (PL) = FPGA:**
  - **Has logic cells, memory blocks, I/O links, and MGTs**
  - **Implements interfaces to the workload FPGAs, can implement real-time logic or additional peripherals**
- **Processor system (PS) = CPU:**
  - **Based on multi-core ARM processors**
  - **Has peripherals: GbE for communication with run control; I2C, SPI, GPIO, etc. to control the hardware**
  - **Runs software: Linux + user application software**

SoC Firmware&Software

What do we need to configure the SoC?

- Firmware for the SoC/PL
- Software for the SoC/PS:
  - System-level software: boot loader + operating system (kernel + rootfs)
  - User-level software: application software, specific to module, and implementation in workload FPGAs

How do we prepare the firmware/software for the SoC?

- Firmware: use Xilinx Vivado
  - Bit-stream file for SoC/PL configuration
  - Hardware description file (.xsa) to be used for software development
- Software: use PetaLinux *(first concentrate on system-level s/w)*
  - First-Stage Boot Loader (FSBL): initialise PS and peripherals
  - U-Boot (secondary program loader): load operating system
  - Operating system: Linux kernel + device tree blob + root file system
  - Some other files (for power management and ARM secure boot), which we have never modified

$\Rightarrow$ **Result running Linux on SoC**

Note: Previously we used Yocto, because we also built the user software in it, now we are building the user software differently (see later), and PetaLinux provides everything needed.

System Software (1): CentOS Linux

The operating system is key to the operation of the SoC and its integration into the ATCN (ATLAS Technical Control Network) and the ATLAS TDAQ software!

Use CentOS:

- Widely accepted, available for armv7 and aarch64, support available from CERN-IT for aarch64
- Change in CentOS strategy in DEC-2020: CentOS 7 support ends JUN-2024, CentOS 8 support ends DEC-2021, and is replaced by CS8 (CentOS Stream 8), which is supported until AUG-2024
- Linux Future Committee was set up at CERN to follow up. Necessity of support for ARM is recognised.
- Will hear more about it on Friday, A. Iribarren, "ARM64 Linux Support and Future CERN Linux"
- In the meantime, for ATLAS for Run 3, we will continue to use CentOS C7, at least to start with

How to prepare CentOS/arm root file system:

- Cross install on a host PC, e.g. *dnf … --forcearch=aarch64 … groupinstall 'Minimal Install'*
- Full recipe available on the SoC twiki page
- Mount root file system using NFS
- Using U-Boot uenv.txt file loaded from host PC (TFTP), one can choose which kernel and rootfs to boot

Result:

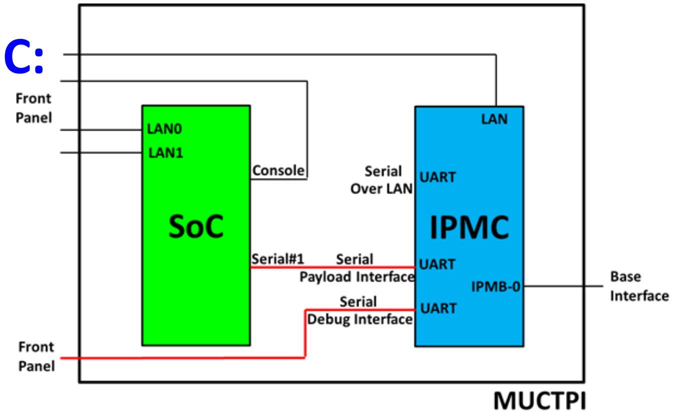
- Looks like the Single–Board Computers (VME), and the desktop systems used in ATLAS/L1CT
- Can be managed as any other SBC, e.g. ssh, users, network, etc.
- Can install additional software packages as previously, e.g. python, expect, etc.

System Software (2): ATCA Identity

- **MAC address:**
  - Required for any (Ethernet) network end point
  - MAC address is specific to the blade (e.g. store in I2C EEPROM):
    ⇒ replacing an ATCA blade implies changing the DHCP declaration
- **Better: Use PICMG HPM.3 Client ID = Shelf Address + Slot Number:**
  - Client ID is specific to the geographical position (e.g. hemisphere or η & φ region)
    ⇒ replacing an ATCA blade is transparent

SoC has serial Payload Interface (PI) with CERN IPMC:

- SoC → IPMC: "Send Message" command to IPMC containing a "Get Shelf Address Info" to ShMgr ⇒ Shelf Address
- SoC →IPMC: "Get Address Info" commands to IPMC and to ShMgr using "SendMessage" command ⇒ Slot Number



Implementation:

- FSBL: using XUartPs_xxxx() functions - proof of principle
- U-Boot: using UCLASS_SERIAL with putc() and getc() functions, call in misc_init_r() function or interactively from U-Boot command shell
- Linux/C++: used for payload sensor data *(see later)*

⇒ Software for FSBL and U-Boot is available in the SoC gitlab group

Note: Not critical for MUCTPI, because single board in experiment.

User application software

What about user application software?

Previously we used Yocto to build the user software

Our software is part of the ATLAS TDAQ software, and we want to run control and monitoring applications.

- ⇒ Use ATLAS TDAQ build tool = CMake, also used by L1CT previously
- ⇒ Cross compilation works with CMake:
  Use CentOS/arm root file system as sysroot, use TDAQ CMake functions, and add a toolchain file for the cross compiler
- ⇒ Build cross compiler from source at gcc.gnu.org (x86_64 → armv7 or aarch64)

Re-use work flow from previous generation:

- Describe all registers of the MUCTPI with the bit fields, memories and FIFOs in XML file

MUCTPI Register Map

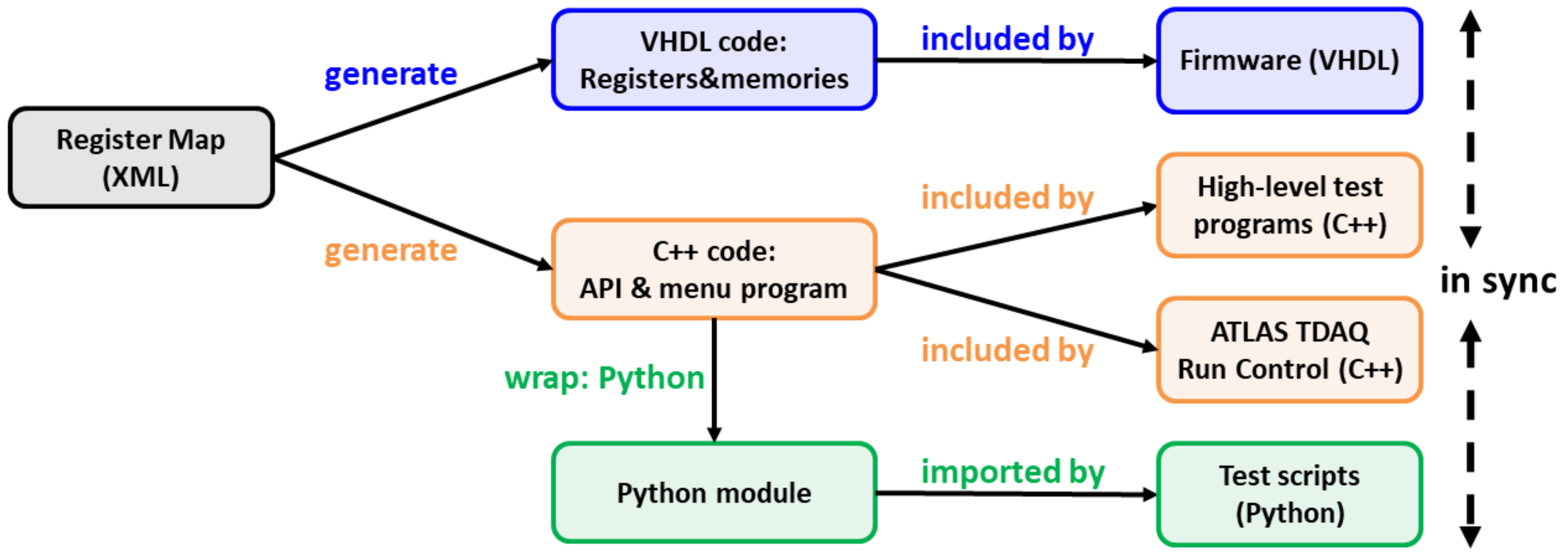
Description of the MUCTPI’s workload FPGAs’ firmware:

- Registers and memories, which have addresses, width, mask, access type etc.
- Can be arranged in blocks, or blocks of blocks (hierarchy)
- Register data are described by bit string, which have bit fields and symbolic values
- Blocks, registers, and fields can exist in “multiples”, i.e. arrays

Example:

```
<module name=“MuonSectorProcessor” size=“0x10000000” type=“D32” … >
  <block name=“MLT”>
    <register name=“SnapPlayControl” addr=“0x000bf404”>
      <field name=“Mode”              mask=“0x00000003”>
        <value name=“OFF”             data=“0x00000000”/>
        <value name=“MUL”             data=“0x00000001”/>
        <value name=“PG”              data=“0x00000002”/>
        <value name=“SPY”             data=“0x00000003”/>
      </field>
      <field name=“SpyEnable”         mask=“0x00000008” form=“BOOLEAN”/>
      <field name=“BcidOffset”        mask=“0xfff00000” form=“NUMBER”>
    </register>
    …
  </block>
  …
</module>
```

MUCTPI Flow of Firmware&Software Development



1) XML → VHDL and C++: hwcompiler (developed by L1CT)
2) C++ → Python wrapper: Simplified Wrapper and Interface Generator (SWIG), available e.g. from WLCG

For C++ and Python, calling of the hwcompiler and SWIG is fully integrated into the ATLAS TDAQ CMake environment:
⇒ automatic generation and wrapping, CMake rebuilds the full software

Mapping of Workload FPGAs

Access to workload FPGAs:

- **Device tree overlay: use Linux Userspace I/O (UIO) for all mappings, i.e. "compatible = "generic-uio""**
- **Registers and memories are mapped (AXI) into processor system of SoC using /dev/uioN *(N=number, typically 0..16)***
- **Use Linux sysfs /sys/class/uio/uioN to identify the mappings**
- **C++ code accesses /dev/uioN using read/write functions**
- **We developed a kernel module for DMA: provides contiguous memory allocation and setting up and starting of the DMA engine**

MUCTPI Run Control

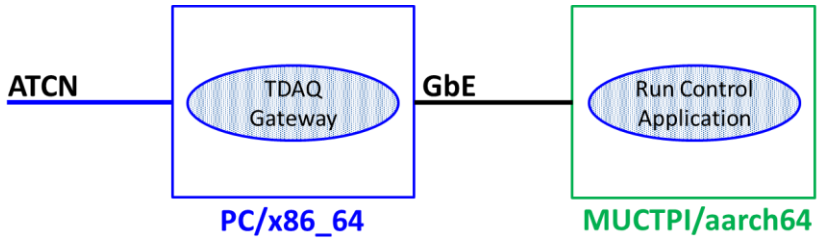
The ATLAS TDAQ software is cross compiled in the same way:

⇒ Build a run control application to be run on the MUCTPI

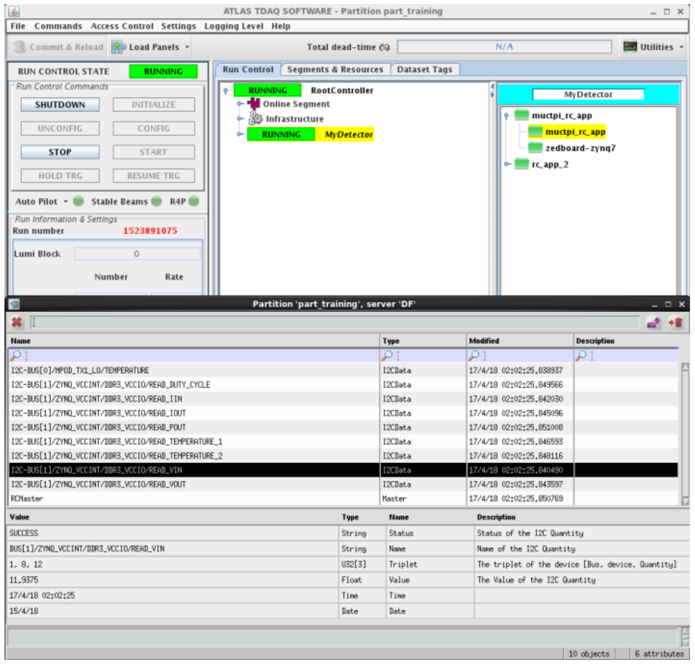
Worldwide LHC Computing Grid (WLCG) provides builds for aarch64:

⇒ Required by ATLAS TDAQ software

We are currently developing MUCTPI run control applications for configuration, control, and monitoring of the MUCTP



Note: the TDAQ gateway running on the host PC is necessary until SoC/CentOS will be allowed on the ATCN.



Run Information & Settings
Run number 1523891075
Lumi Block 0
Number Rate

Name	Type	Modified	Description
I2C-BUS[0]/MPOD_TX1_LO/TEMPERATURE	I2CData	17/4/18 02:02:25.838937	
I2C-BUS[1]/ZYNQ_VCCINT/DDR3_VCCIO/READ_DUTY_CYCLE	I2CData	17/4/18 02:02:25.849566	
I2C-BUS[1]/ZYNQ_VCCINT/DDR3_VCCIO/READ_IIN	I2CData	17/4/18 02:02:25.842030	
I2C-BUS[1]/ZYNQ_VCCINT/DDR3_VCCIO/READ_IOUT	I2CData	17/4/18 02:02:25.845096	
I2C-BUS[1]/ZYNQ_VCCINT/DDR3_VCCIO/READ_POUT	I2CData	17/4/18 02:02:25.851008	
I2C-BUS[1]/ZYNQ_VCCINT/DDR3_VCCIO/READ_TEMPERATURE_1	I2CData	17/4/18 02:02:25.846593	
I2C-BUS[1]/ZYNQ_VCCINT/DDR3_VCCIO/READ_TEMPERATURE_2	I2CData	17/4/18 02:02:25.848116	
I2C-BUS[1]/ZYNQ_VCCINT/DDR3_VCCIO/READ_VIN	I2CData	17/4/18 02:02:25.840490	
I2C-BUS[1]/ZYNQ_VCCINT/DDR3_VCCIO/READ_VOUT	I2CData	17/4/18 02:02:25.843597	
RCMaster	Master	17/4/18 02:02:25.850769	

Value	Type	Name	Description
SUCCESS	String	Status	Status of the I2C Quantity
BUS[1]/ZYNQ_VCCINT/DDR3_VCCIO/READ_VIN	String	Name	Name of the I2C Quantity
1, 8, 12	U32[3]	Triplet	The triplet of the device [Bus, device, Quantity]
11.9375	Float	Value	The Value of the I2C Quantity
17/4/18 02:02:25	Time	Time	
15/4/18	Date	Date	

ATLAS TDAQ GUI

MUCTPI Hardware Control

One example of hardware control using the IPMC Payload Interface:

Stand-alone application running on SoC:

- **Reads some temperature & voltage values from the MUCTPI hardware (using I2C)**
- **Updates them as payload-controlled sensors in the IPMC (using the Payload Interface, provided by CERN IPMC)**

⇒ The sensors can be used by the ShelfManager to define thresholds and to adapt the shelf fan speed, or to shut off the MUCTPI payload if required.

Software available at https://gitlab.cern.ch/atlas-l1ct/IpmcSerialInterface

Continuous Integration (1)

Use gitlab/CI pipelines for building the software:

- **Automate building of software**
- **Detect errors early**
- **Run regularly (nightly) or on demand (release)**
- **Adapt more easily to new or changing requirements and software**
- **Provide “documentation” in form of scripts**

Use docker images (x86_64):

- **Run independently from host configuration**

Tutorial on Thursday by M. Wyzlinski:

- ***Does not assume any previous knowledge of gitlab/CI***
- ***Does not assume any ATLAS, TDAQ, or MUCTPI-specific software***

Continuous Integration (2)

Flow for the development:

0. Vivado (not yet CI):
   - Produce FPGA bit stream file, and hardware description file
1. PetaLinux:
   - Add U-Boot I2C and IPMC communication, the device tree overlay, and the DMA kernel module, etc.
   - Produce U-Boot, device tree blob, and Linux kernel
2. CentOS/arm + cross compiler:

   2.1 Build minimal root file system with additional packages

   2.2 Build cross compiler:
   - Produce docker image that can be easily used for cross compilation
3. TDAQ Software:

   3.1 Add build tools required by TDAQ

   3.2 Add software dependencies needed for TDAQ

   3.3 Build TDAQ software:
   - Following a recipe established by ATLAS TDAQ for building a subset of the TDAQ software for aarch64
4. MUCTPI Software:
   - Build MUCTPI-specific (partially generated) low-level software and run control applications

Note: While for aarch64 (64-bit), software from WLCG is being used during step 3.1 and 3.2, for armv7 (32-bit) that software needs to be built, e.g. TBB, Boost.

Continuous Integration (3)

Deployment of the results:

Install artefacts on several host PCs for our prototype systems:

- CentOS/arm root file system + TDAQ and L1CT software:
  - Used for booting of the MUCTPI, mounting of the root file system, and mounting of the user application software
- Cross compiler:
  - Used for local compilation during development: very useful for firmware development with Python script, or for software development of the MUCTPI software and run control applications
- We have a single script for setup on the host PC (development) or on the MUCTPI (deployment)

⇒ Developer/user does not notice the “cross” environment

Summary

- We have successfully built a new MUCTPI using the ATCA standard and a System-on-Chip (SoC)
- When running CentOS on the SoC and using cross-compilation, the software work flow is exactly as before when using VME and a Single-Board Computer
- The software work flow is described and run using gitlab/CI pipelines:
  - Provide software for deployment (MUCTPI) and for development (host PC)
  - Provide software regularly (nigthly), and on demand (release)
- Currently installing the MUCTPI into the experiment

Open Questions

- *Future Linux? Support for arm? Support for docker images with cross compiler?*
- *Future of TDAQ/arm or WLCG/arm?*
- *Future of file systems? AFS, EOS, CVMFS?*
  We use EOS for distribution of CI artefacts … but had some difficulties …
- *We have several prototype MUCTPIs: how to provide host configuration, e.g. users, network, systemd, etc. – how to provide system administration?*
- *How to share software for common developments, e.g. hwcompiler or IPMC Payload Interface?*
- *Common hardware? Too late for MUCTPI (being installed), but for future L1CT developments?*