SoC Infrastructure for the ATLAS Phase-II Level-0 Central Trigger

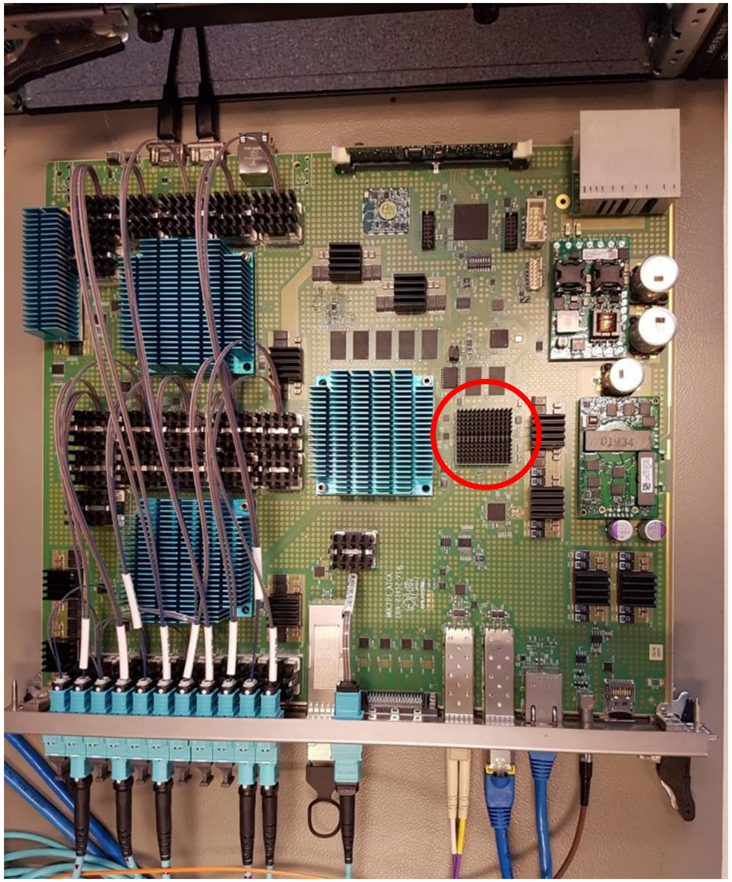
- **Introduction**
- **Build Automation**
- **Booting & Host Configuration**
- **Summary & Outlook**

on behalf of ATLAS L0CT team

ATLAS Phase-I MUCTPI

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

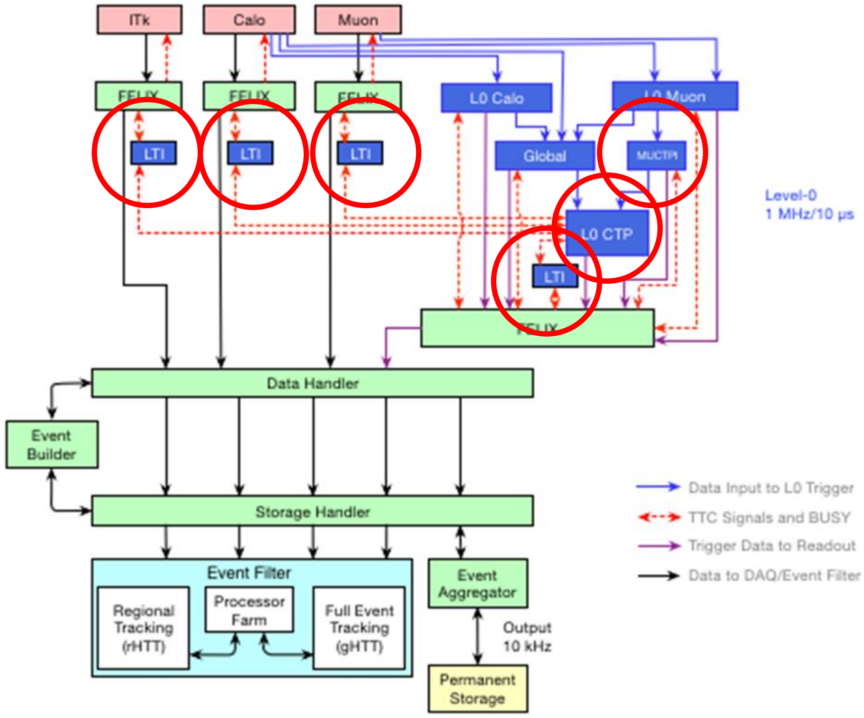
- **ATCA blade with SoC**
  = Programming Logic (PL) + Processor System (PS):
  - Earlier prototypes: Xilinx Zynq 7000 SoC
  - Later prototypes and production modules using Xilinx Zynq Ultrascale+ MPSoC
- **Trigger processing implemented in FPGAs controlled and monitored by SoC**
- **MUCTPI installed in ATLAS since August 2021: smooth commissioning and operation**
- **Several MUCTPIs in lab as spares and for potential software developments**



ATLAS Phase-II L0CT

- **MUCTPI: two MUCTPIs of existing type with new firmware (current plan)**
- **L0 Central Trigger Processor (L0CTP) = new module**
- **Local Trigger Interface (LTI), replacing current timing modules = new modules, around 48 in experiment**

⇒ All modules will be ATCA blades and use SoC, most likely Xilinx Kria SoM

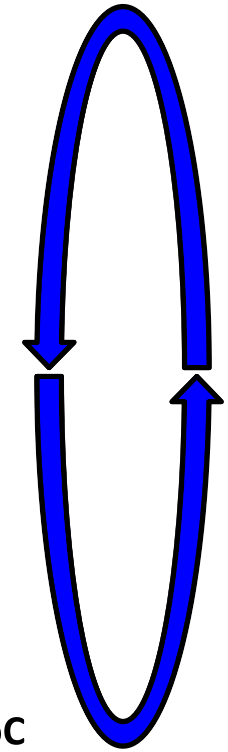


In addition, several evaluation boards, prototypes, and production modules in lab will be used concurrently

⇒ Requires support for several different SoC-based modules (types and instances of type), each with their specific firmware and software

Development Workflow

- **Firmware:**
  - Design (VHDL) using Xilinx Vivado
  - Produce SoC PL bit file, and hardware description file (.xsa file)
  - Produce processing FPGA bit files
- **Boot files:**
  - First-stage boot loader (FSBL)
  - Other files: PMU firmware and ARM-trusted firmware (ATF)
  - Secondary Program Loader = U-Boot
  - Linux kernel, device tree, and root file system = CentOS → Alma
- **User application software:**
  - Low-level: access to module-specific features, based on firmware
  - High-level: integrate into ATLAS TDAQ using run control processes on SoC

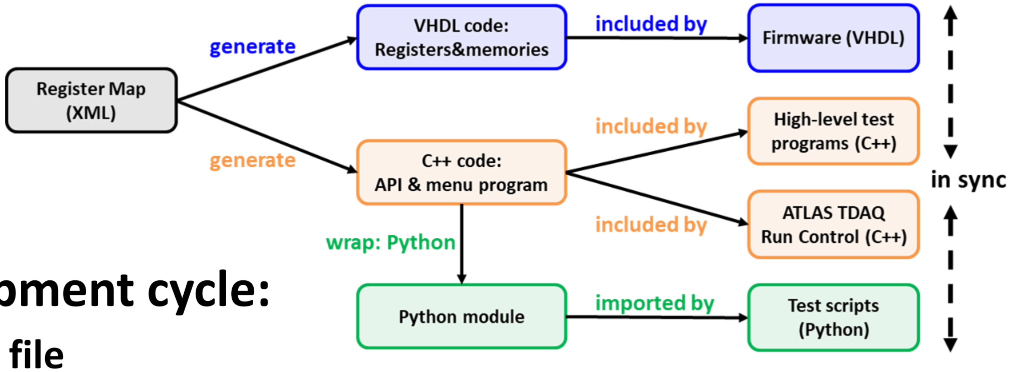


⇒ **During development, iterate over new firmware, new boot files, new software**

Build Automation (1)

First aspect: consistency of firmware and software

- **Firmware and software are built from common XML file:**
  - Previously presented: hardware compiler, improved with automatic generation of address decoder
  - Translates register definitions to VHDL code and C++ (run control) and Python (interactive testing)



⇒ **Provides a fast and local development cycle:**
  - Firmware development: load new bit file
  - Recompile software locally to test new firmware, i.e. cross-compilation against a release ⇒ new software on NFS available on SoC
  - Test new firmware interactively using Python
  - Skip recompilation of boot files and reboot (if only processing FPGAs need new firmware)

Build Automation (2)

Second aspect: actual compilation of firmware and software

- **Automate building of bit files, boot files, and software**
  ⇒ Use Continuous Integration (CI)
- **Use GitLab CI and describe compilation in YAML files:**
  Once new firmware (and XML file) pushed into GitLab repository
  ⇒ Rebuild FPGA bit files
  ⇒ Rebuild software
- **YAML files make use of GitLab variables to select what to build**
  ⇒ Rebuild bit files, boot files, and software for a given type of module (= platform)
- **Nightly builds provide a common basis for developers to continue their work**
- **At regular moments, a build on request is used as a stable release for operation**

Build Automation: Firmware

- Xilinx Vivado is used for building the firmware
- We rebuild the bit files for the processing FPGAs; the SoC PL provides Chip2Chip communication with processing FPAGs and stays very stable
- GitLab runners with a shell executor run on dedicated firmware PCs
- A TCL script is used to generate the necessary IPs, to compile the firmware, and to generate the bit files
- The firmware is structured using git submodules:
  - Common parts used by several processing FPGAs
  - Parts specific for the different prototypes of the MUCTPI
- Nightly builds produce all bit files for testing
- Firmware for the future LTI and CTP will be structured in a similar way; we may use docker images for the building

Build Automation: Boot Files

Xilinx PetaLinux is used for the generation of the necessary boot files: FSBL, PMUfw, ATF, U-Boot, Linux kernel, and device tree

- **GitLab runners with docker executors use image prepared by CERN ATS**
  → *Big thanks to Adrian Byszuk and SY-EPC-CCE!*
- **Modifications to Xilinx ZynqMP template:**
  - U-Boot, get shelf address and slot number (see later)
  - Kernel configurations, e.g. selection of device drivers
  - Modify the device tree, e.g. external devices, UIO, etc.
- **Since there are several different L0CT modules, a common template was developed using Yocto layers**
  → *Tutorial on SoC PetaLinux Template, THU 5-OCT 10.00 by Giulio Muscarello*
- **Normally use the PetaLinux kernel; successful tests with Alma 9 kernel were done in collaboration with ATLAS TDAQ SysAdmins**
  → *ATLAS Network and Booting, FRI 6-OCT 16.30 by Quentin Duponnois*
- **Do not use the PetaLinux root file system, but CentOS 7, currently moving to Alma 9**
- **Compile *axi* device driver for AXI access and DMA**

Build Automation: Software

User application software: based on ATLAS TDAQ software + L0CT specific software

- **Build docker images with root file system and cross-compiler**
- **Build dependencies (32-bit)* or copy from LCG (64-bit)**
  **** 32-bit support to be phased out by Phase-2***
- **Use ATLAS TDAQ build script (CMake) for cross compiling TDAQ software**
- **Could use ATLAS TDAQ build for ARM (64-bit) but need the cross-compilers, (32-bit)* for earlier prototypes of MUCTPI and both cross-compilers for local development**
- **GitLab CI uses GitLab variables to select what needs to be built: e.g. docker image, TDAQ, or L0CT software, i.e. do not have to rebuilt everything all the time**
  → Tutorial on GitLab CI Parallel Builds, THU 5-OCT 11.20 by Kareen Arutjunjan
- **Nightly build for having common development basis, and on request for releases**

Deployment

- **Deploy results (= CI artifacts) to all host PCs*:**

  ** In phase-1: SoC modules are locally connected to a host PC due to network isolation in ATCN, in phase-2: SoC modules will be directly connected, software installed on TDAQ servers*

  - Install processing FPGA bit files
  - Install SoC boot files, and kernel and root file system
  - Install cross-compiler for local compilation
  - Install TDAQ and L0CT software
  - Push approach, being replaced by running Puppet on host PCs (pull approach)
- **Provide a single script that allows users to set up environment**
  - Select module, boot files, root files system, and software release:
    - On host PC: for booting and for local compilation
    - On SoC: for running L0CT software
- **Use GitLab CD to run several tests on all hosts:**
  - Could potentially (re-)boot SoC: disruptive, only manually!
  - Run local cross-compilation
  - Login (ssh) to SoC
  - Execute script with simple tests, e.g. read some I2C values, etc.

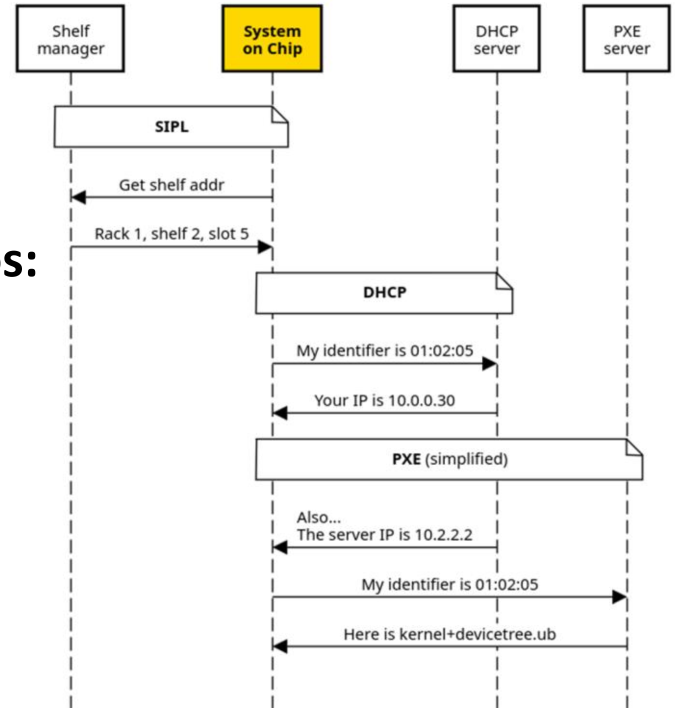
→ Tutorial on Test Automation, FRI 6-OCT 10.00 by. Michal Husejko

Booting

Crucial aspect: identity of a module (type and instance)

Boot files and software alone are not sufficient;
need also to know which given module of a given module type

- **Identity can be defined by module or by location in the installation:**
  - **Module: e.g. MAC, read from EEPROM**
  - **Location: e.g. shelf address and slot number, read from IPMC (required in ATCA)**
  - **Could use a mix of both**
- **Use identity to obtain boot files, several scenarios:**
  - **Static IP address and host name**
  - **Read from U-Boot script file**
  - **Use DHCP with or without PXE (ATLAS)**
  - **Can be used with UEFI and GRUB**
  - *→ ATLAS Network and Booting, FRI 6-OCT 16.35 by Quentin Duponnois*



Host Database

Crucial aspect: single "source of truth" for all modules identities

- **Keep a base of all modules:**
  - Module identifier: MAC, USB (console, JTAG)
  - Module type (=platform): e.g muctpi-v4, kria, lti-zcu102, etc.
  - Module and host name: e.g. MUCTPI-v4-01, LTI-5, etc.
  - Host PC the module is connected to *(in phase-1: network isolation in ATCN)*

1	Module Name	Host Name	Host PC	Platform	JTAG address	MAC (primary)	IP address
6	MUCTPI v3 #2	apl-tdq-muctpi-02	pc-tdq-muctpi-01	muctpi-v3	210308A11DF7	08:00:30:00:3a:24	10.145.53.24
7	MUCTPI v4 #1	mpsoc-muctpi-v4-1	pcphl1ct08	muctpi-v4	210308AFAC74	08:00:30:00:3a:28	10.145.4.1
8	MUCTPI v4 #2	mpsoc-muctpi-v4-2	pcphl1ct11	muctpi-v4	210308AFAC7F	08:00:30:00:3a:2c	10.145.4.2
9	MUCTPI v4 #3	mpsoc-muctpi-v4-3	pcphl1ct07	muctpi-v4	210308AFAC71	08:00:30:00:3a:30	10.145.4.3
10	Kria #1	kria-1	pcphl1ct24	kria	210408B08F23A	00:0a:35:0d:63:11	10.145.75.1
11	Kria #2	kria-2	pcphl1ct24	kria	210408B08FA5A	00:0a:35:0a:7e:5a	10.145.75.2
12	LTI #1	lti-1	pcphl1ct14	lti-zcu102	210308AE69F6	80:d3:36:00:50:00	10.145.76.1
13	LTI #2	lti-2	pcphl1ct14	lti-zcu102	210308A750ED	80:d3:36:00:50:03	10.145.76.2

- **To start with, use a CSV file:**
  - Update information in git repository
  - Deploy on all host PCs:
    - Use for configuration of DHCP and DNS servers
    - Udev rules: create /dev files for console and JTAG with information from host database
  - Use for deployment of CI build results
  - Use with single script to boot any system; knows which SoC is connected
- **May possibly move to some other format later (LDAP?)**

Host Configuration

Crucial aspect: identity of the running module – host name

- **Configure software and services, use systemd and Puppet**
- **Systemd:**
  - **→ Run daemons and tasks:**
  - – **Linux and network services**
  - – **Module initialization: bit files of processing FPGAs**
- **Puppet:**
  - **→ Provide configuration:**
  - – **Udev rules: create /dev files for I2C, GPIO, SPI, etc.**
  - – **Users: add/remove users**
  - – **For L0CT services: hardware monitoring (using IPMC), pmg server, run control processes**

Summary

- **Scalable system:**
  - **From single module to multiple modules of multiple types**
- **Automated system for development:**
  - **Use firmware-software co-development**
  - **Use GitLab CI for firmware, boot files, and user application software**
  - **Provide local compilation for fast development cycle**
- **Configurable system for booting and host configuration:**
  - **Identity from module (MAC) and/or from location (Client ID, IPMC)**
  - **List of boot files provided from single source of information using DHCP and PXE**
  - **Provide host configuration using systemd and Puppet**

Outlook

To be done:

- **Continue testing Alma 9 kernel + root file system**
- **Further investigate booting with UEFI and GRUB**
- **Test integration of module into ATLAS TDAQ test bed**
- **Improve automated testing: more tests and a dashboard of results**
- **Implement user authentication using LDAP**
- **Alternative implementation for host database**

- **Maintain Phase-1 system and get ready for Phase-2 system**

References

- **ATLAS L1CT Hardware Compiler:**
  https://gitlab.cern.ch/atlas-l1ct/hwcompiler
- **CERN ATS Docker Files for HDL EDA Software:**
  https://gitlab.cern.ch/cce/docker_build
- **SoC PetaLinux Template:**
  https://gitlab.cern.ch/soc/petalinux-template
- **SoC IPMC Communication (SIPL):**
  https://gitlab.cern.ch/soc/u-boot-sipl
- **SoC CentOS Root File System:**
  https://gitlab.cern.ch/soc/centos-rootfs