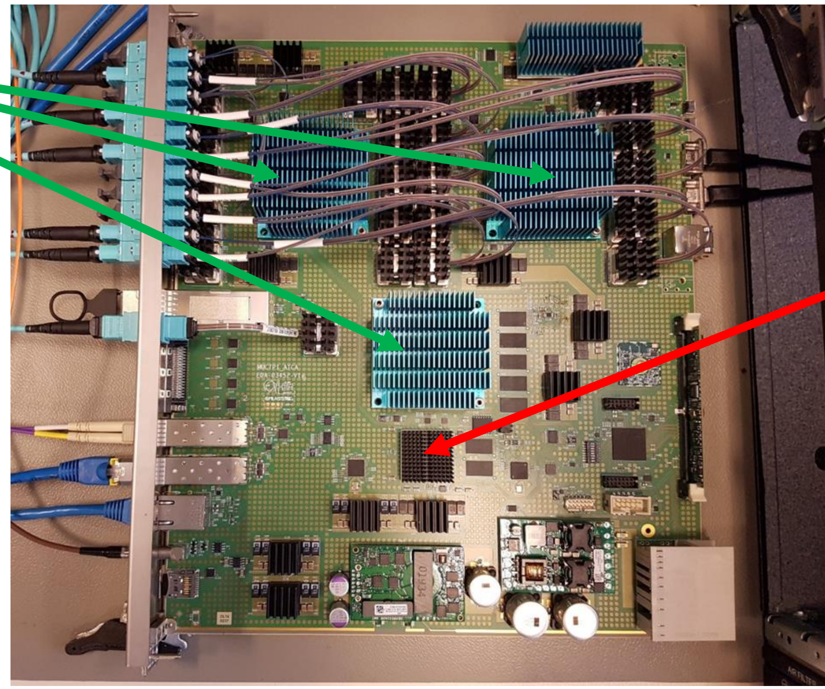
# #9 – Continuous Integration for the Software and the Firmware of the new ATLAS Muon-Central-Trigger-Processor-Interface (MUCTPI) – 1/2

**FPGAs**

**⇒ Firmware**



**System-on-Chip**

**⇒ Software**
**= Operating System +**
**User Application Software**

**The MUCTPI is an electronics module with three FPGAs and one System-on-Chip. The FPGAs provide real-time processing of trigger data, while the System-on-Chip runs software to control the hardware of the module and the operation of the FPGAs.**

# #9 – Continuous Integration for the Software and the Firmware of the new ATLAS Muon-Central-Trigger-Processor-Interface (MUCTPI) – 2/2

**Several developers are working concurrently on the development of firmware and software for the various prototype and production modules of the MUCTPI.**

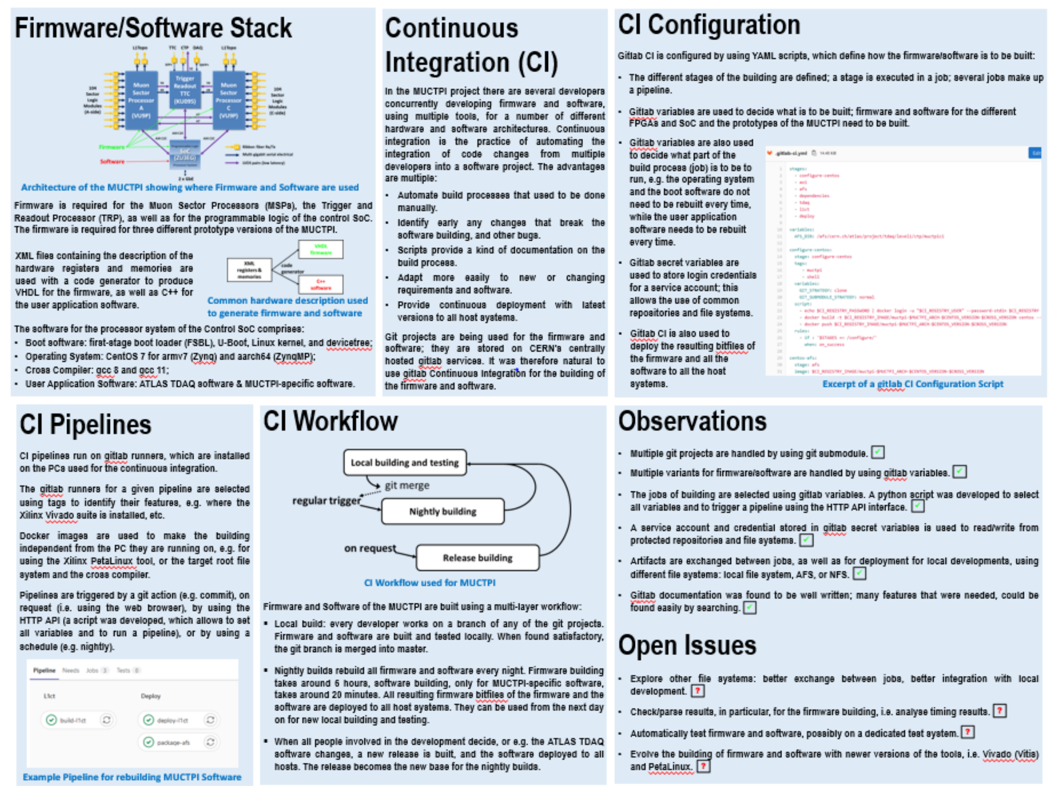
**We use continuous integration for the firmware and the software in order to improve greatly the development process.**

The poster shows

- Why we are using continuous integration;
- How we configure it and run pipelines;
- How it is integrated with our workflow;
- Some observations we made and open issues we encountered.

Please come visit me during the poster session.

Thank you.



## Firmware/Software Stack



Architecture of the MUCTPI showing where Firmware and Software are used

Firmware is required for the Muon Sector Processors (MSPs), the Trigger and Readout Processor (TRP), as well as for the programmable logic of the control SoC. The firmware is required for three different prototype versions of the MUCTPI.

XML files containing the description of the hardware registers and memories are used with a code generator to produce VHDL for the firmware, as well as C++ for the user application software.



Common hardware description used to generate firmware and software

The software for the processor system of the Control SoC comprises:

- Boot software: first-stage boot loader (FSBL), U-Boot, Linux kernel, and devicetree;
- Operating System: CentOS 7 for armv7 (Zynq) and aarch64 (ZynqMP);
- Cross Compiler: gcc 8 and gcc 11;
- User Application Software: ATLAS TDAQ software & MUCTPI-specific software.

## Continuous Integration (CI)

In the MUCTPI project there are several developers concurrently developing firmware and software, using multiple tools, for a number of different hardware and software architectures. Continuous integration is the practice of automating the integration of code changes from multiple developers into a software project. The advantages are multiple:

- Automate build processes that used to be done manually.
- Identify early any changes that break the software building, and other bugs.
- Scripts provide a kind of documentation on the build process.
- Adapt more easily to new or changing requirements and software.
- Provide continuous deployment with latest versions to all host systems.

Git projects are being used for the firmware and software; they are stored on CERN's centrally hosted gitlab services. It was therefore natural to use gitlab Continuous Integration for the building of the firmware and software.

## CI Configuration

Gitlab CI is configured by using YAML scripts, which define how the firmware/software is to be built:

- The different stages of the building are defined; a stage is executed in a job; several jobs make up a pipeline.
- Gitlab variables are used to decide what is to be built; firmware and software for the different FPGAs and SoC and the prototypes of the MUCTPI need to be built.
- Gitlab variables are also used to decide what part of the build process (job) is to be to run, e.g. the operating system and the boot software do not need to be rebuilt every time, while the user application software needs to be rebuilt every time.
- Gitlab secret variables are used to store login credentials for a service account; this allows the use of common repositories and file systems.
- Gitlab CI is also used to deploy the resulting bitfiles of the firmware and all the software to all the host systems.

Excerpt of a gitlab CI Configuration Script

## CI Pipelines

CI pipelines run on gitlab runners, which are installed on the PCs used for the continuous integration.

The gitlab runners for a given pipeline are selected using tags to identify their features, e.g. where the Xilinx Vivado suite is installed, etc.

Docker images are used to make the building independent from the PC they are running on, e.g. for using the Xilinx PetaLinux tool, or the target root file system and the cross compiler.

Pipelines are triggered by a git action (e.g. commit), on request (i.e. using the web browser), by using the HTTP API (a script was developed, which allows to set all variables and to run a pipeline), or by using a schedule (e.g. nightly).



Example Pipeline for rebuilding MUCTPI Software

## CI Workflow



CI Workflow used for MUCTPI

Firmware and Software of the MUCTPI are built using a multi-layer workflow:

- Local build: every developer works on a branch of any of the git projects. Firmware and software are built and tested locally. When found satisfactory, the git branch is merged into master.
- Nightly builds rebuild all firmware and software every night. Firmware building takes around 6 hours, software building, only for MUCTPI-specific software, takes around 20 minutes. All resulting firmware bitfiles of the firmware and the software are deployed to all host systems. They can be used from the next day on for new local building and testing.
- When all people involved in the development decide, or e.g. the ATLAS TDAQ software changes, a new release is built, and the software deployed to all hosts. The release becomes the new base for the nightly builds.

## Observations

- Multiple git projects are handled by using git submodule. ✓
- Multiple variants for firmware/software are handled by using gitlab variables. ✓
- The jobs of building are selected using gitlab variables. A python script was developed to select all variables and to trigger a pipeline using the HTTP API interface. ✓
- A service account and credential stored in gitlab secret variables is used to read/write from protected repositories and file systems. ✓
- Artifacts are exchanged between jobs, as well as for deployment for local developments, using different file systems: local file system, AFS, or NFS. ✓
- Gitlab documentation was found to be well written; many features that were needed, could be found easily by searching. ✓

## Open Issues

- Explore other file systems: better exchange between jobs, better integration with local development. ?
- Check/parse results, in particular, for the firmware building, i.e. analyse timing results. ?
- Automatically test firmware and software, possibly on a dedicated test system. ?
- Evolve the building of firmware and software with newer versions of the tools, i.e. Vivado (Vitis) and PetaLinux. ?