Continuous Integration for the Software and the Firmware of the new ATLAS Muon-Central-Trigger-Processor-Interface (MUCTPI)

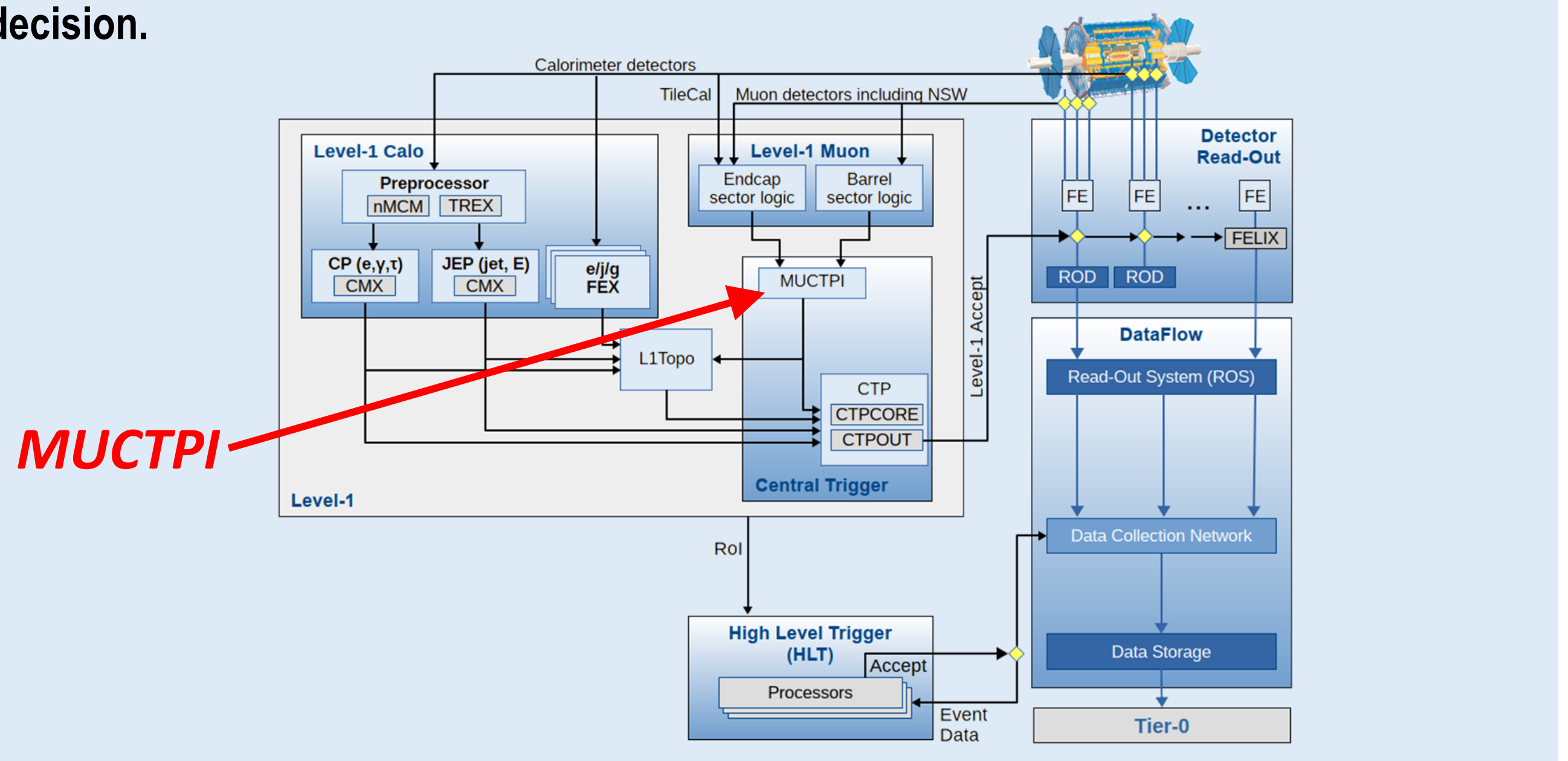
Y. Afik, P. Czodrowski, S. Haas, A. Koulouris, A. Kulinska, A. Marzin, T. Pauly, O. Penc, S. Perrella, V. Ryjov, R. Spiwoks, L. Sanfilippo, R. Simoniello, P. Vichoudis, T. Wengler, M. Wyzlinski
CERN, Switzerland

ATLAS Experiment @ LHC

The ATLAS experiment is a general-purpose experiment at the Large Hadron Collider (LHC) at CERN. It observes proton-proton collisions at an energy of almost 14 TeV and a bunch crossing (BC) rate of 40 MHz. Up to 52 pile-up collisions are expected for Run 3, which started in 2022. This results in more than $2{*}10^9$ interactions per second and requires the use of a trigger system in order to select those events which are interesting to physics and which can be recorded to permanent storage at a reasonable rate. The ATLAS trigger system consists of a Level-1 trigger based on custom electronics, which reduces the event rate to a maximum of 100 kHz, and a high-level trigger system based on commercial computers, which reduces the event rate to around 1 kHz.

Level-1 Trigger System

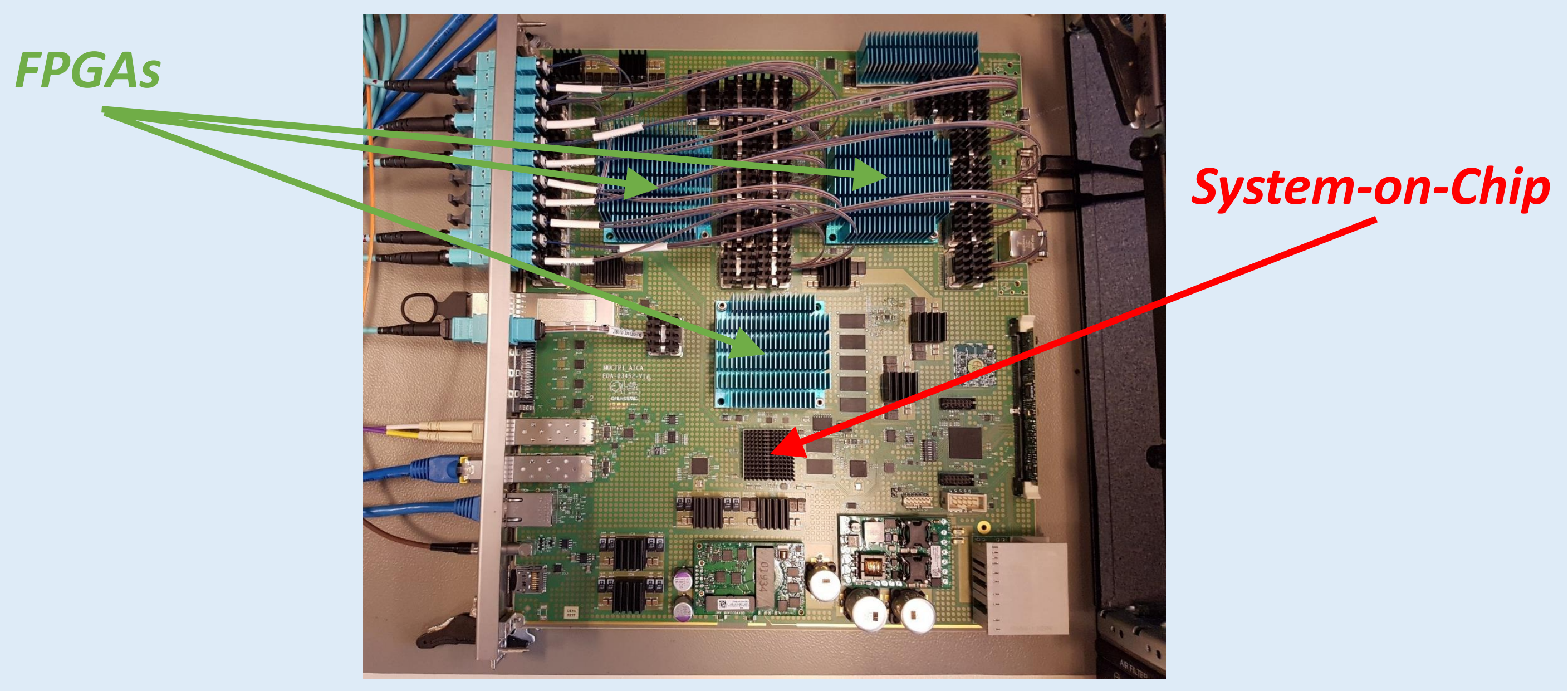
The first-level trigger uses reduced-granularity information from the calorimeters and dedicated muon trigger detectors. The trigger information is based on multiplicities and topologies of trigger candidate objects. The muon trigger is based on several dedicated muon trigger detectors in the barrel and endcap regions. The Muon-to-Central-Trigger-Processor Interface (MUCTPI) combines the muon candidate counts from both regions, taking into account double counting of single muons that are detected by more than one chamber due to geometrical overlap of the muon chambers and the trajectory of the muon in the magnetic field. The MUCTPI sends the muon candidate information to the Topological Processor and the muon counts to the Central Trigger Processor (CTP), which combines the trigger information from the calorimeter trigger, the MUCTPI, and the Topological Processor in order to make the final Level-1 decision.



The ATLAS Trigger and Data Acquisition System

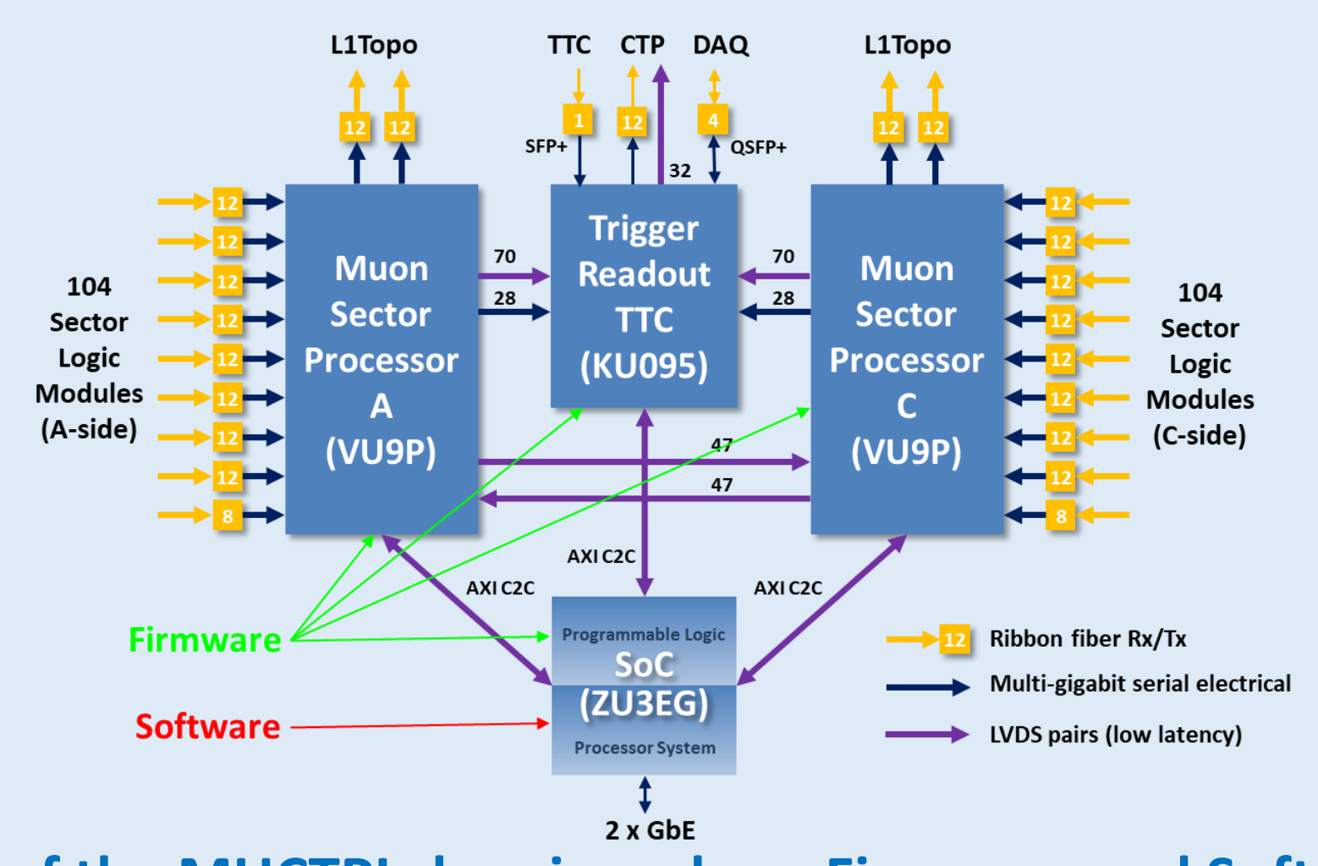
Upgrade of the MUCTPI

The MUCTPI upgrade is part of the overall upgrade of ATLAS on the road to High-Luminosity LHC and is in line with the development of the New Small Wheel of the muon trigger system installed before Run 3. The new MUCTPI uses optical links to replace bulky electrical cables. Those links allow the muon trigger detectors to send more muon candidates with more precise information. The new MUCTPI provides improved overlap handling and full-precision information to the Topological Processor. The new MUCTPI is built as a single ATCA blade. It receives 208 optical links and uses two FPGAs for the overlap handling, counting of muon candidates, and sending candidates to the Topological Processor. A third FPGA provides the total count of muon candidates to the CTP, and readout data to the data acquisition system. A System-on-Chip (SoC) provides hardware control of the new MUCTPI and integrates it into the ATLAS run control system.



The new MUCTPI Module

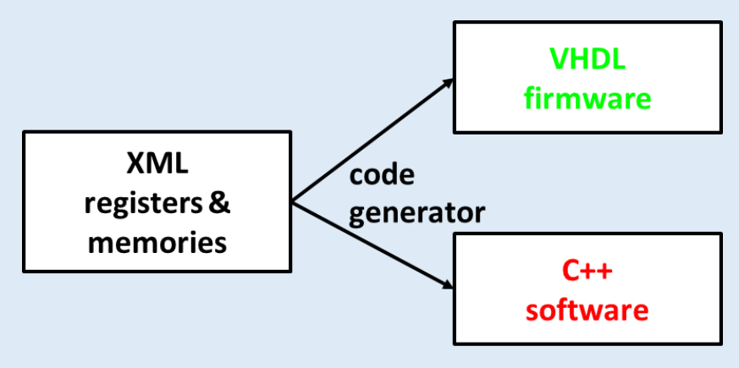
Firmware/Software Stack


L1Topo
TTC
CTP
DAQ
L1Topo
SFP+
QSFP+
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)
70
28
32
70
28
47
47
AXI C2C
AXI C2C
AXI C2C
Firmware
Software
Programmable Logic
SoC (ZU3EG)
Processor System
2 x GbE
Ribbon fiber Rx/Tx
Multi-gigabit serial electrical
LVDS pairs (low latency)

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:

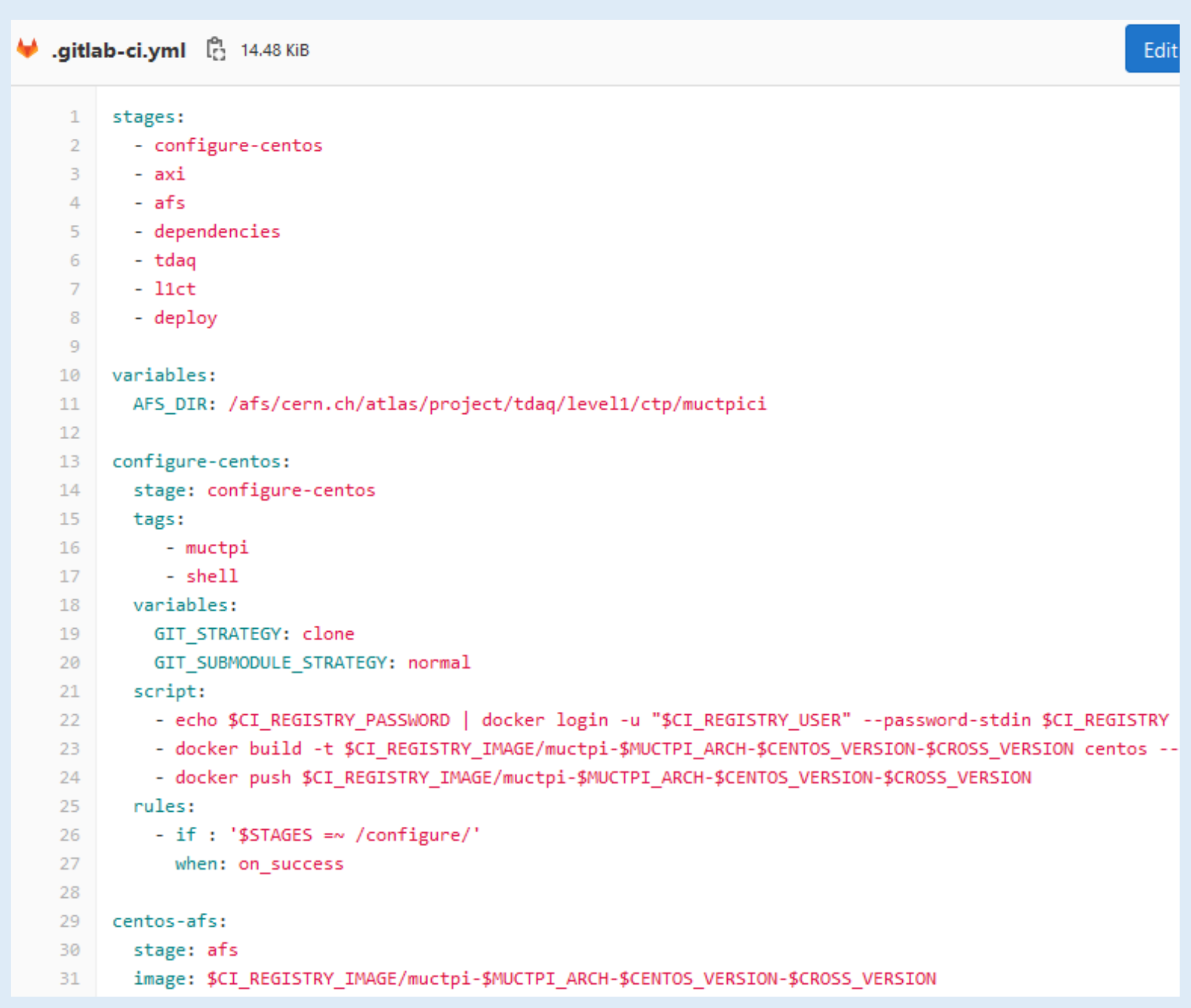
- Automate build processes that used to be done manually.
- Identify early any changes that break the software building, and other bugs.
- Scripts provide a kind of documentation on the build process.
- Adapt more easily to new or changing requirements and software.
- Provide continuous deployment with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



.gitlab-ci.yml 14.48 KiB Edit

```
stages:
  - configure-centos
  - axi
  - afs
  - dependencies
  - tdaq
  - l1ct
  - deploy

variables:
  AFS_DIR: /afs/cern.ch/atlas/project/tdaq/level1/ctp/muctpici

configure-centos:
  stage: configure-centos
  tags:
     - muctpi
     - shell
  variables:
    GIT_STRATEGY: clone
    GIT_SUBMODULE_STRATEGY: normal
  script:
    - echo $CI_REGISTRY_PASSWORD | docker login -u "$CI_REGISTRY_USER" --password-stdin $CI_REGISTRY
    - docker build -t $CI_REGISTRY_IMAGE/muctpi-$MUCTPI_ARCH-$CENTOS_VERSION-$CROSS_VERSION centos --
    - docker push $CI_REGISTRY_IMAGE/muctpi-$MUCTPI_ARCH-$CENTOS_VERSION-$CROSS_VERSION
  rules:
    - if : '$STAGES =~ /configure/'
      when: on_success

centos-afs:
  stage: afs
  image: $CI_REGISTRY_IMAGE/muctpi-$MUCTPI_ARCH-$CENTOS_VERSION-$CROSS_VERSION
```

Excerpt of a gitlab CI Configuration Script

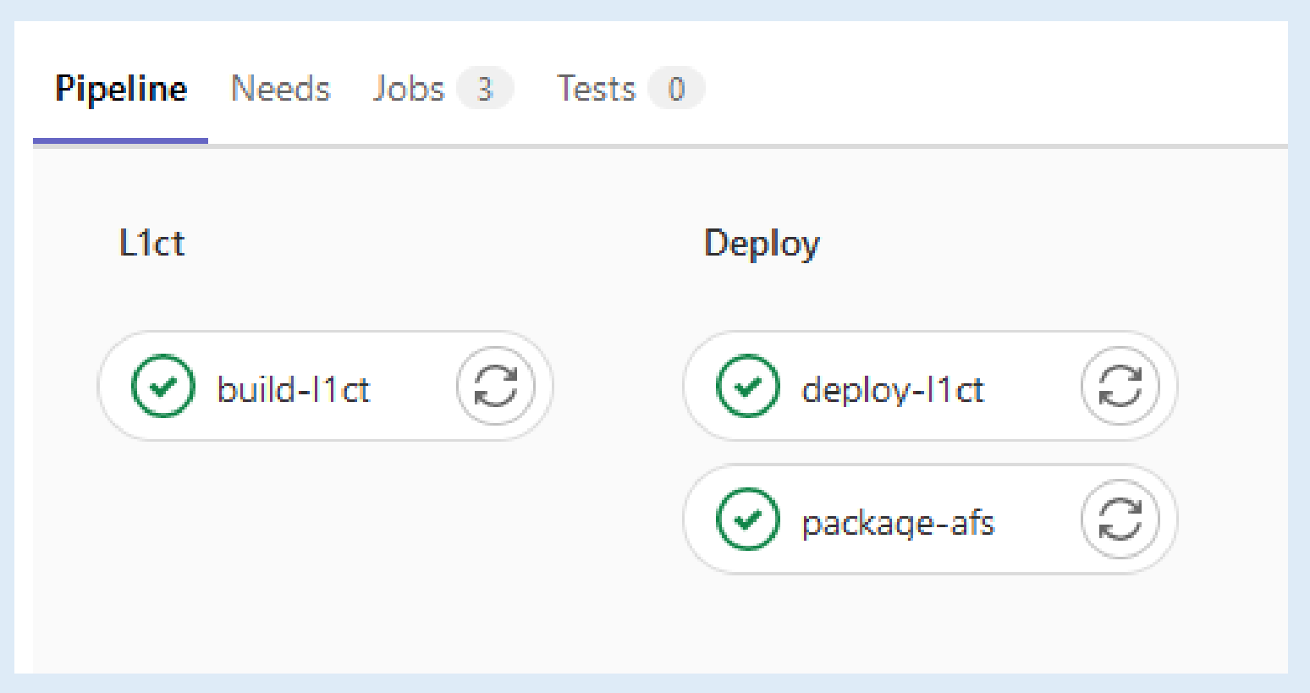
CI Pipelines

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.

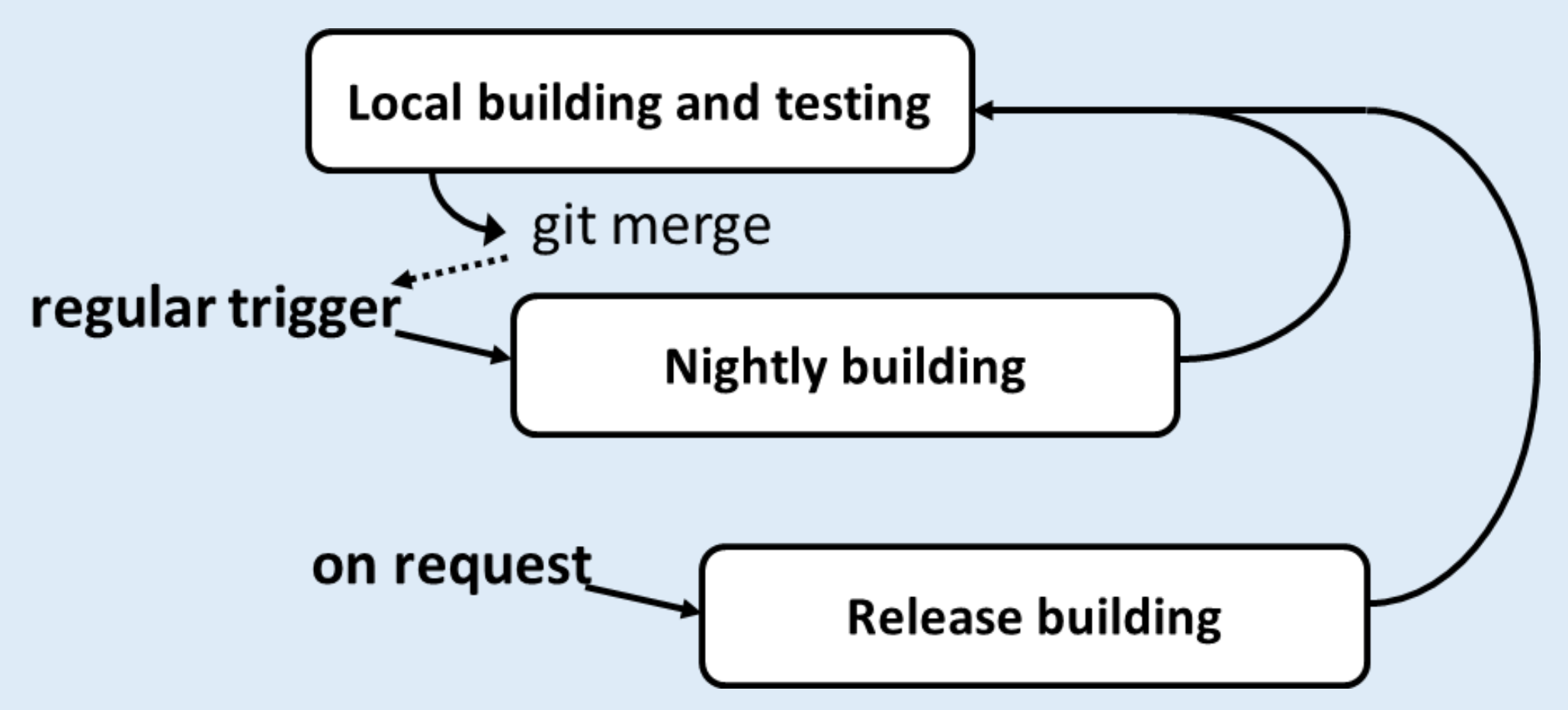
Docker images are used to make the building independent from the PC they are running on, e.g. for using the Xilinx PetaLinux tool, or the target root file system and the cross compiler.

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. ?

Summary

Continuous integration greatly improves collaborative and combined development of firmware and software. We are planning to extend the use of continuous integration in the current and in future projects.