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

R. Spiwoks, Y. Afik, P. Czodrowski, S. Haas, A. Koulouris, A. Kulinska, A. Marzin, T. Pauly, O. Penc, S. Perrella, V. Ryjov, L. Sanfilippo, R. Simoniello, P. Vichoudis, T. Wengler, M.Wyzlinski
on behalf of the ATLAS Collaboration

***Abstract*— The new Muon-Central-Trigger-Processor Interface (MUCTPI) is part of the Phase-I upgrade of the ATLAS Level-1 trigger system for Run 3 of the Large Hadron Collider at CERN. The new MUCTPI has three high-end FPGAs and one System-on-Chip (SoC). The FPGAs receive and process muon candidate information arriving on 208 high-speed optical serial links. Processed trigger information and summary data are sent to other parts of the trigger and the data acquisition. The SoC controls, configures and monitors the hardware and the operation of the MUCTPI. The FPGA part of the SoC provides communication with the processing FPGAs, while the processor system runs software for communication with the run control system of the ATLAS experiment. All software necessary to run the MUCTPI, including operating system and run control software is being built using continuous integration. CentOS Linux, cross-compilation and the existing framework for building of the ATLAS trigger and data acquisition (TDAQ) software are being used in order to deploy the TDAQ software directly on the SoC. After the successful use of continuous integration of the software, also the firmware is being built using that scheme. This paper describes the advantages of the use of continuous integration, our experience, as well as the difficulties that needed to be overcome.**

***Index Terms*— Continuous Integration, System-on-Chip.**

I. Introduction

ATLAS [1] is a general-purpose experiment at the Large Hadron Collider (LHC) at CERN, which observes proton-proton collisions at a center-of-mass energy of almost 14 TeV at a bunch crossing rate of 40 MHz. Up to 52 pile-up collisions are expected for Run 3, which started in 2022. This results in more than 10^9 interactions per second and requires the use of a trigger system in order to select the events most interesting for physics studies within the constraints on the maximum event rate that can be recorded.

The ATLAS trigger and data acquisition (TDAQ) system [2], see Figure 1, consists of a first-level trigger based on custom electronics and firmware, and a high-level trigger based on custom-off-the-shelf hardware and processing software. The first level trigger uses information from the calorimeters and the muon trigger detectors, Resistive Plate Chambers (RPC) in the barrel and Thin-Gap Chambers (TGC), as well as Micromegas (MM) and small-strip Thin-Gap Chambers (sTGC) from the New Small Wheels in the endcap. The Muon-Central-Trigger-Processor Interface (MUCTPI) receives muon candidate information from all muon sectors, and calculates multiplicities, i.e. counts of muon candidates, avoiding double counting of single muons that are detected by more than one muon sector due to the geometrical overlap of the chambers and the trajectory of the muons in the magnetic field; this is called overlap handling. The MUCTPI further calculates muon trigger object information and sends it to the Topological Trigger Processor, which combines it with trigger objects from the calorimeters. The muon multiplicity information is sent to the Central Trigger Processor (CTP), which combines it with the trigger information from the calorimeters and the topological trigger to form the final first-level decision.

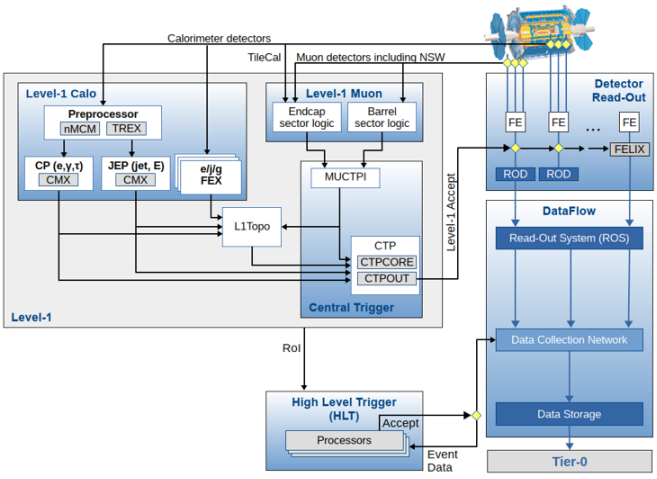

Fig. 1. Schematic of the ATLAS TDAQ system.

II. The New MUCTPI

Compared to the MUCTPI used in Run 1 and Run 2 [4], the new MUCTPI [5] receives more trigger candidates with more detailed data using optical links instead of electrical ones. It improves the overlap handling by taking into account overlap between octants, which was previously impossible. It sends full precision information to the topological trigger processor using optical links, while previously it was sending coarse information over electrical cables. The new MUCTPI is built as

Manuscript submitted August 4, 2022, revised November 20, 2022.
All authors are with CERN, Switzerland (e-mail: Ralf.Spiwoks@cern.ch).

a single ATCA blade, see Figure 2. It uses two Muon Sector Processors (MSPs), one Trigger and Readout Processor (TRP), and a System-on-Chip (SoC), see Figure 3. The MSPs are implemented using Xilinx Virtex Ultrascale+ FPGAs [6]. They receive trigger muon information from the 208 muon sectors, implement the overlap handling, and send trigger objects to the topological trigger system.

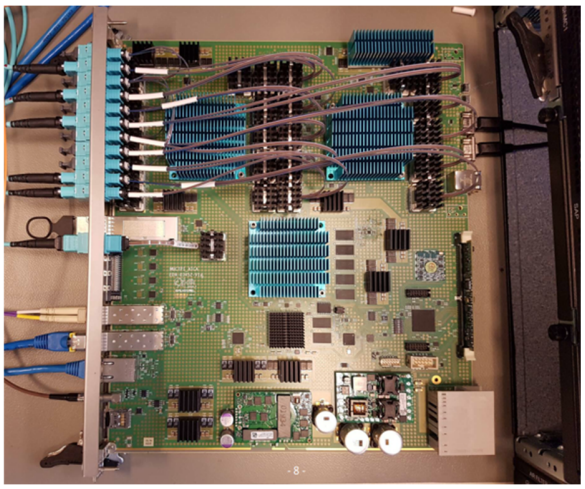

Fig. 2. Photograph of the new MUCTPI.

The TRP is implemented using a Xilinx Kintex Ultrascale FPGA [6]. It combines the muon trigger information, and sends muon trigger multiplicities to the CTP and trigger data to the DAQ. The SoC is a Xilinx Zynq Ultrascale+ MPSoC [6]. It is used for control, configuration, and monitoring of the hardware and of the operation of the MUCTPI.

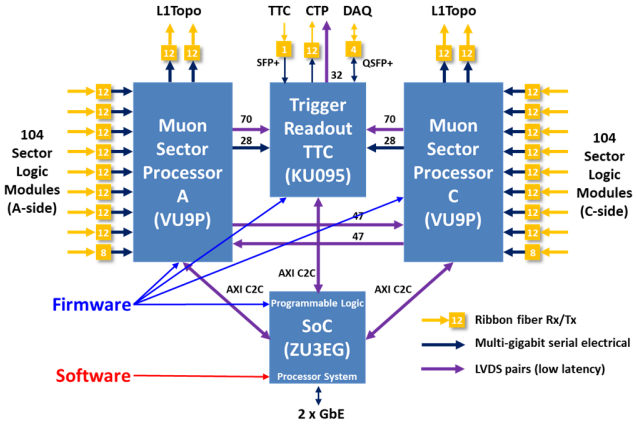

Fig. 3. Architecture of the new MUCTPI showing where Firmware (blue) and Software (red) are used.

The SoC consists of two different elements: programmable logic (PL) and a processor system (PS). The PL is like an FPGA and is being used, among other things, to provide read and write access to registers and memories implemented in the other processing FPGAs. The PS runs an operating system, and configures and controls the hardware of the MUCTPI using standard interfaces; it further has two Gigabit Ethernet connections, which are used for the communication with the ATLAS run control system [7].

Three versions of prototypes of the MUCTPI were successively built with different hardware components: while the latest version, V3, contains the components described above, the previous versions, V1 and V2, are using a Xilinx Zynq 7000 SoC [6], and version V1 is using Xilinx Virtex Ultrascale FPGA [6]. All versions are fully functional and are fully supported for use in the experiment, as well as in the test labs for continuous development.

III. The Firmware and Software Stack

In order to operate the MUCTPI, a number of firmware and software components need to be built and run on the corresponding hardware: the firmware for the MSPs, the TRP, and the SoC PL, as well as the software for booting the SoC PS, the root file system, the cross compiler and the user application software.

A. Firmware

The firmware for the MUCTPI MSPs and TRP FPGAs is primarily based on VHDL design files and some Xilinx Intellectual Property (IP) cores. Block diagram design entry is used for the SoC PL part as well as for the interface to the AXI bus infrastructure in the processing FPGAs. The registers and memories of the FPGAs are described in XML files. Using a custom code generator the XML files are used to generate some of the VHDL code. The same files are also used by the user application software to generate the access functions, see Figure 4. This guarantees consistency between the firmware implementation and the low-level software used to access the hardware.

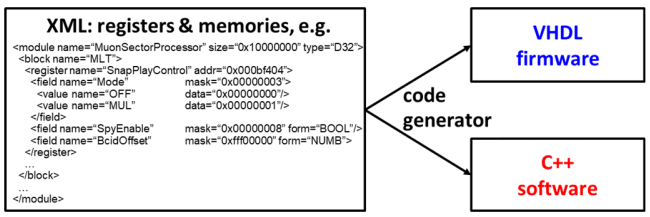

Fig. 4. Common XML files used for Firmware (blue) and Software (red) Development.

The firmware is implemented using the Xilinx Vivado design suite [6]; a Tcl script is used within Xilinx Vivado to generate the necessary IPs, build the firmware, and generate the programming files.

The firmware source code is stored in several Git repositories [8]. The project is organized in eight top-level repositories for the two types of processing FPGAs and the SoC, considering the three MUCTPI prototype versions. In addition, code common to a given type of FPGA or SoC as well as code common to all designs is stored in separate Git repositories. In order to avoid code duplication, the top-level repositories make use of Git submodules to include the relevant code.

B. Boot Software

The Xilinx Vivado design suite produces a hardware description file for the SoC. That file is imported by the Xilinx PetaLinux tools [6] for building the first-stage boot loader (FSBL), which initializes the SoC hardware, the U-Boot system loader, which loads the operating system, as well as the Linux kernel, a custom kernel module for DMA, and the device tree. It could also be used to build a root file system, but CentOS is being used instead as described below.

User-specific files in the PetaLinux project-specific meta-user layer are used to modify the U-Boot, so that it can communicate with the ATCA IPMC module [9] of the MUCTPI in order to get information about the ATCA shelf it is running in. In addition, the boot sequence is modified in order to download an environment file from the host PC. This enables us to have full flexibility on the kernel, device tree and root file system to be booted.

C. Root File System

The root file system used on the SoC is CentOS Linux [10]. This choice is driven by the fact that the same root file system is being used in the majority of sub-detectors of the ATLAS experiment, including other modules of the Level-1 central trigger system. The team developing software for the MUCTPI and the other modules of the Level-1 central trigger system thus works on a common basis.

Although CentOS is being used with different CPU architectures, i.e. x86_64 and arm, the configuration issues are the same and this makes it easier to support it. We further think that the security certification, which is necessary for being on the strictly controlled technical network of the experiment, will be possible in the future only if the same operating system for the whole experiment will be used.

Both variants of CentOS required to run on the two types of SoC for the different prototypes of the MUCTPI, i.e. armv7 (for the Zynq 7000) and aarch64 (for the Zynq Ultrascale+ MPSoC), are built using cross installation with the DNF package manager [11]. The root files systems are mounted on the SoC using the Network File System (NFS) from a host PC.

We are using CentOS 7 to be fully in line with the ATLAS policy, and we will follow the strategy chosen by ATLAS when CentOS 7 will come to its end of life, which is planned for 2024.

D. Cross Compiler

In order to build all the user application software, cross compilation is used with the gcc compiler [12] for armv7 and for aarch64. While the armv7 version is built from source code, the aarch64 version is used directly from the Worldwide LHC Computing Grid (WLCG) [13].

E. User Application Software

The MSP and TRP firmware provides register and memory descriptions provided in XML files. With the help of a custom code generator those files are compiled to C++ code, which provides access to the registers and memories. The code makes use of the device file mappings provided in the Xilinx Linux kernel, and which correspond to the AXI Chip2Chip connections in the firmware [6]. This enables us to access to the MSP and TRP firmware features directly from the SoC.

In addition, the user application software uses the run control software provided by ATLAS TDAQ [7], which is cross compiled in order to run on the SoC. Run control applications running on the SoC allow full integration of the MUCTPI into the ATLAS run control system.

IV. Continuous Integration

Continuous integration (CI) is the practice of automating the integration of code changes from multiple developers into a common software project, see [14]. It allows early identification of changes, which possibly break the building of the software, and other bugs. The scripts used for CI provide documentation of the build process and allow one to adapt more easily to new or changing requirements and changing software. At the same time, CI also provides a basis for continuous deployment (CD).

The firmware and software of the MUCTPI are stored in Git projects on the central GitLab [15] services provided at CERN. For that reason it was natural to use GitLab CI for continuous integration.

A GitLab CI [16] project uses a YAML file, which describes the build process as a pipeline that is made from individual jobs which are executed in ordered stages. GitLab runners execute pipelines on PCs reserved for CI. The GitLab runners are selected by using tags, which indicate their features, e.g. running Xilinx Vivado or the cross compiler. A pipeline can be triggered by a Git action, e.g. Git commit, interactively from the web browser, by other pipelines, by using the HTTP API, or by using a schedule, e.g. every night at a given time. For each job, conditions when to execute it can be configured freely using GitLab provided environment variables, which gives great flexibility when designing the build process. Common considerations are the aforementioned trigger conditions of the overall pipeline, the source or target branch or particular file changes.

The first Git projects of the MUCTPI using CI were those of the user application software, followed by the root file system and the cross compiler, and then the boot software. In the end, the CI methodology was also extended to the building of the MUCTPI firmware. However, in this article the different uses will be presented in the following sections in their logical order from firmware to software, adding deployment as a final stage and a CI script.

Our goal for using CI and CD for the firmware and the software of the MUCTPI was to integrate code changes into the master branch, to provide build checking and automatic deployment to all the host systems, which the developers are working on locally. This is possible because the team of developers is rather small with less than ten people. In-depth verification of the produced firmware and software was not intended, at least not to start with when detailed test programs do not yet exist and the focus is on development. The testing is rather performed by the developers themselves, although ideas and plans for such an extension of the CI/CD exist, see sections V and VI.

A. CI for Firmware

For the CI of the firmware, YAML files and GitLab variables were added to each Git project for building a given type of an FPGA and the version of the MUCTPI prototype, so that the firmware can be built by the developer working on it, by triggering the pipeline from the GitLab GUI. In addition, a new Git project was created with a YAML file and several GitLab variables that enable calling each combination of FPGA and MUCTPI prototype version in a child pipeline. Both firmware projects, for the MSP and the TRP, in all three variants of the prototypes are rebuilt every night, so that up-to-date firmware versions are always available for testing by all developers in the next morning.

B. CI for Boot Software

The building of the boot software depends on the use of Xilinx PetaLinux, which is free for use, but which needs registration of the user. This was implemented by downloading the Xilinx PetaLinux software and installing it in a docker image, which is made available only to people of the Level-1 central trigger project, who are registered with Xilinx. The docker image is then used by a GitLab runner for executing a job. The job by defintion downloads the Git project files, copies user-specific files for the U-Boot modifications and the DMA kernel module into the meta-user directory of the PetaLinux project, and makes several petalinux calls for configuring and building the project. The resulting files for booting the SoC, i.e. the FSBl, the U-Boot, the Linux kernel, the DMA kernel module, and the device tree, are stored in a common file system.

C. CI for Root File System

The CentOS operating system is built using a docker image: starting from a CentOS7/x86_64 docker image, the CentOS7 armv7 and aarch64 variants were cross installed using DNF [11]. The installation starts with a minimal installation, and additional packages are installed that are needed later for runtime, e.g. networking tools and the python scripting language. The selection of those packages is provided in a file and can be modified to include further packages.

In addition, the DMA kernel module produced by PetaLinux is copied into the root file system, as well as a number of systemd modules [18], which are to be called at boot time and which will set up the MUCTPI SoC in the network and with the services required to work in the experiment.

D. CI for Cross Compiler

The docker images with the CentOS operating systems are further used to install the gcc cross compiler for armv7 by building it from scratch. The same was done for other pre-dependencies of the ATLAS TDAQ and MUCTPI-specific software, e.g. Boost [19] and TBB [20]. For aarch64 the cross compiler and the other dependencies are taken from WLCG using a distributed file system. The docker images with the CentOS root file system and the cross compiler are the main workhorses for the building of the user application software.

E. CI User Application Software

The user application software consists of a part of the ATLAS TDAQ software and MUCTPI-specific software. The building of the ATLAS TDAQ software itself is based on CMake and fully supports cross compilation. A script is provided in the ATLAS TDAQ software to build any number of ATLAS TDAQ projects, which are included as Git submodules.

The building of the MUCTPI-specific software follows the ATLAS TDAQ scheme and also uses CMake. All necessary Git projects are included as Git submodules and are built using the same script provided by the ATLAS TDAQ software.

The YAML file for building the software determines the particular architecture as well as which parts of the software build to execute by evaluating dedicated CI variables set at job creation.

F. CD of Firmware and Software

The deployment of the nightly builds and the release builds was constantly extended to provide bitfiles and software directly on a number of hosts for the different prototypes of the MUCTPI. While this works in the lab, it does not work for the experiment which is protected behind a network gateway. Another CI job was added to provide all software in a tarball for deployment at the experiment. The pipeline for the building of the MUCTPI-specific software, the deployment on the host systems, as well as the providing of a tarball for the experiment is shown in Figure 5.

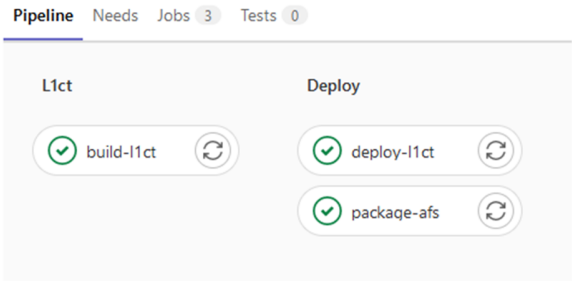

Fig. 5. Pipeline for the building and deployment of the user application software for the MUCTPI.

In addition to a regular building and deployment of the bitfiles and software, we want to be able to develop and debug software using local development. This mode of operation requires the additional installation of the cross compiler on the local host, while the root file is already installed there, because the host provides it to the MUCTPI using NFS. The NFS exportation is then extended to the local user directory, which is mounted on the SoC. This allows one to develop software on the PC and to test it on the MUCTPI, and if found working well, to check it into Git; the nightly or release build will then pick it up, build it and deploy it on all host systems.

G. CI/CD Script

A python script was written to manage the building of all the software of the MUCTPI using the HTTP API of GitLab. By the help of this script one can select which of the two different architectures, armv7 and aarch64, and which version of the software needs to be built. It is also possible to select which part

of the build is to be made, so that some parts do not need to be rebuild constantly, e.g. the operating system or the boot software. Intermediate results of the pipeline are stored on a common file system. The script was further extended to cover the deployment of the software in different forms to different host systems. In summary, by this script any part of the software or all of it can be rebuilt on request.

V. Workflow with Continuous Operation

Having described the elements of the CI for the MUCTPI, we now show how they are being used in the workflow for the MUCTPI, which has three steps, see Figure 6:

1) Local build: This step does not include any CI. The developers work on a branch of the Git project. Firmware and software are built and tested locally. Firmware and software are consistent because they are using the same XML files. Tests are carried out by the developer on the local system. When found satisfactory, the local Git branch is pushed and merged into master.
2) Nightly build: This step makes use of the CI/CD. The firmware and software are rebuilt every night. The master branches of all projects are taken to pick up all the latest changes of all developers. Since firmware and software are using the same XML files, consistency at this level is guaranteed. If any changes made during the day break the building of the firmware or the software, then the developers go back the next day and check what went wrong. If the building works, all resulting bitfiles of the firmware and all software are deployed to all host systems, and can be used from the next day on for new local building and testing.
3) Release build: This step makes use of the CI/CD. When all people involved in the development decide, 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 deployment in the experiment.

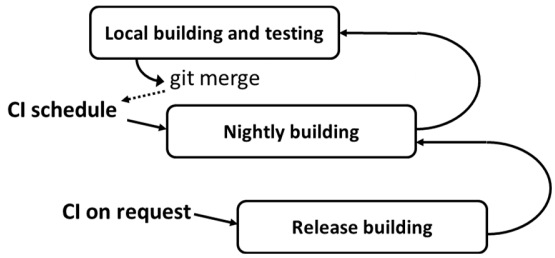

Fig. 6. Workflow with Continuous Integration for the MUCTPI.

VI. Observations and Issues

One of the difficulties of running CI is using it for multiple projects. This is where Git submodules play an essential role. One of the first things we had to do was to create new super projects for the firmware and for the software, which include all the other modules as submodules. Right now we have not yet combined the firmware and the software super projects, since we are using nightly builds to pick up the changes made to the common XML files the firmware and software are based on.

Another difficulty was to provide for different variants of the hardware of the different prototype version or the different CPU architectures of the SoC. This is where GitLab variables are essential: they allow one to parametrize a common YAML file or other build scripts for different variants.

Another GitLab CI feature we found very useful is that a pipeline does not always have to run completely, when writing the YAML file with rules for each job, one can decide which part of the pipeline is run by selecting the jobs using GitLab variables. This reduces run time, and leaves out the parts that only very rarely change, e.g. the boot software and the root file system and cross compiler.

One of the biggest difficulties we encountered, was to pass some of the artifacts, like the root file system, between the different stages of the pipelines and for the local development due to the size and structure of the artifact. While we wanted to use EOS [21] in the beginning, this turned out to be very unpractical, in particular, for the large root file system with its many small files. Copying the root file system was extremely slow and took several hours. Also using a squashfs file system [22] did not work because we explicitly wanted to write to the root file system several times between different jobs, e.g. to add the kernel module for DMA or to add a number of systemd modules. We reverted to using AFS [23], which is still widely used although it is not supposed to be supported for much longer and new disk quota are not provided or existing quota extended. Since we are using the file system for sharing within the Level-1 central trigger team, within the same institute, NFS could be another good solution, in particular for the local development. We are also discussing other solutions with our colleagues from the ATLAS TDAQ software.

The difficulty that some file systems can only be accessed using the right credentials, e.g. AFS, was solved by using a service account and providing the password in a masked GitLab variable.

When building the software fails due to an error, those errors can usually be easily found by looking at the GitLab logs. However, for the building of firmware this is not necessarily so easy. Scripts parsing the results of the firmware build will need to be developed in order to check if and how well the building worked. Checking if all timing constraints were met can be achieved using simple parsing, and we are planning to include those checks in the near future.

The fact that we are deploying on a number of host systems can also be solved by using GitLab variables. In addition, a script to set up on the local host PC is being used to set up the right values for booting the SoC, and for running the local development.

GitLab documentation proved to be generally well written, in particular, in comparison to other software packages. It was relatively easy to find solutions to the features we needed. There also is a lively discussion forum [24].

We installed GitLab runners for running the CI pipelines on a number of dedicated PCs under our control. The nightly build for the firmware takes about 6 hours on two PCs, and the software (only MUCTPI-specific software) about 20 minutes on a third PC. This is sufficient for the time being, but with the

extension of CI increasing, more PCs might need to be added. Of course, using our own PCs for the CI, requires some system administration work, and so a centrally organized alternative could be something to be investigated.

VII. Results and Outlook

Using CI greatly improved collaborative development of the firmware and the software of the MUCTPI during its development, as well as during the exploitation phase we are now moving into. CI provides us with the latest files for firmware and software on all host systems; it helps to find errors early. It provides documentation on the build method itself. We have constantly increased the use of CI from the user application software to the firmware, as well as to the deployment.

We will need to understand how CI will work with migrating to newer versions of the proprietary tools, i.e. migrating from Xilinx Vivado to Vitis [6], or to a newer version of Xilinx PetaLinux, in particular, for the project-specific files that we are adding the U-Boot building, and the hooks that PetaLinux provides for customization.

We have not yet used CI for automatic testing. This could also further enhance the quality of the firmware and software, if adequate tests and a dedicated test system can be found to be used by the CI.

VIII. Acknowledgments

We would like to thank all our colleagues from the SoC interest group for the many enriching discussions on all firmware and software related issues around the use of SoC [25]. During the second CERN SoC Workshop organized by the SoC interest group several presentations showed the use of CI and CD, in particular [27] and [28].

References

[1] ATLAS Collaboration, “The ATLAS Experiment at the CERN Large Hadron Collider”, J. Instr., vol. 3, Aug. 2008, Art. no. S08003.
[2] ATLAS Collaboration, “Technical Design Report for the Phase-I Upgrade of the ATLAS TDAQ System”, CERN/LHCC/2013-018, 2013. [Online]. Available: https://cds.cern.ch/record/1602235. Accessed on: July 7, 2022.
[3] ATLAS Collaboration, “Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System”, CERN/LHCC/2017-020, 2017. [Online]. Available: https:cds.cern.ch/record/2285584. Accessed on: March 3, 2021.
[4] M. V. Silva Oliveira, S. Artz, B. Bauss, H. Boterenbrood, V. Buescher, A. S. Cerqueira et al., "The ATLAS Level-1 Muon to Central Trigger Processor Interface for Run 2 of the LHC", J. Instr., vol. 10, Feb. 2015, Art. no. C02027.
[5] S. Perrella, Y. Afik, A. Armbruster, P Czodrowski, N. Ellis, S. Haas et al., “Integration and Commissioning of the ATLAS Muon-to-Central-Trigger-Processor Interface for Run 3”, J. Instr., vol. 17, Apr. 2022, Art. no. C04006.
[6] Xilinx Inc. [Online]. Available: http://www.xilinx.com. Accessed on: July 7, 2022.
[7] ATLAS Collaboration, “The ATLAS Data Acquisition and High Level Trigger System”, J. Instr., vol. 11, Jun. 2016, Art. no. P06008.
[8] Version control system GIT. [Online]. Available: https://git-scm.com.
[9] J. Mendez, V. Bobillier, S. Haas, M. Joos, S. Mico, F. Vasey, “CERN-IPMC Solution for AdvancedTCA Blades”, in TWEPP 2017, Santa Cruz, CA, United States, 2018. [Online]. Available: https://cds.cern.ch/record/2312397. Accessed on: November 20, 2022.
[10] The CentOS Project. [Online]. Available: http://www.centos.org. Accessed on: June 29, 2022.
[11] Software package manager DNF. [Online]. Available: http://fedoraproject.org/wiki/DNF. Accessed on: June 29, 2022.
[12] The GNU compiler collection GCC. [Online]. Available: https://gcc.gnu.org. Accessed on: June 29, 2022.
[13] The Worldwide LHC Computing Grid. [Online]. Available: https://wlcg.web.cern.ch. Accessed on: June 29, 2022.
[14] Atlassian Corporation Plc. [Online]. Available: https://www.atlassian.com/continuous-delivery/continuous-integration. Accessed on: June 29, 2022.
[15] GitLab Inc. [Online]. Available: https://about.gitlab.com. Accessed on: June 29, 2022.
[16] GitLab Continuous Integration/Continuous Delivery. [Online]. Available: https://docs.gitlab.com/ee/ci. Accessed on: June 29, 2022.
[17] Docker Inc. [Online]. Available: https://docker.com. Accessed on: June 29, 2022.
[18] Systemd System and Service Manager. [Online]. Available: https://systemd.io. Accessed on: June 29, 2022.
[19] Boost C++ LIbraries [Online]. Available: https://www.boost.org. Accessed on: October 5, 2022.
[20] Intel oneAPI Threading Building Blocks [Online]. Available: https://github.com/oneapi-src/oneTBB. Accessed on : October 5, 2022.
[21] EOS Open Storage [Online]. Available: https://eos-web.web.cern.ch.
[22] SquashFS File System. Available: https://docs.kernel.org/filesystems/squashfs.html. Accessed on : October 6, 2022.
[23] OpenAFS distributed file system. [Online]. Available: https://www.openafs.org. Accessed on: June 29, 2022.
[24] GitLab discussion forum. [Online]. Available: https://forum.gitlab.com. Accessed on: June 29, 2022.
[25] System-on-Chip interest group at CERN. E-mail: system-on-chip@cern.ch. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/SystemOnChip. Accessed on: June 29, 2022.
[26] 2nd CERN SoC Workshop [Online]. Available: https://indico.cern.ch/event/996093. Accessed at : October 6, 2022.
[27] M. Husejko, “Setting up basic GitLab CI and CD Environment for Zynq-based Designs”, presented at the *2nd CERN SoC Workshop,* Geneva, Switzerland, Jun. 7-11, 2021.
[28] R. Spiwoks, “Software Framework for the System-on-Chip of the ATLAS MUCTPI”, presented at the *2nd CERN SoC Workshop,* Geneva, Switzerland, Jun. 7-11, 2021.