PROCEEDINGS OF SCIENCE

An automated pipeline for continuous integration of FPGA firmware and software for the LHCb Run3 upgrade

Paolo Durante*[a], Luis Granado Cardoso[a], Joao Vitor Viana Barbosa[a], Federico Alessio[a] and Guillaume Vouters[b]
[a] *CERN (European Organization for Nuclear Research)*
[b] *LAPP (Laboratoire d'Annecy-le-Vieux de Physique des Particules)*
E-mail: paolo.durante@cern.ch

The readout system for the upcoming Run3 upgrade of the LHCb experiment at CERN is based around a common readout board called PCIe40. By reconfiguring the onboard FPGA with dedicated firmware, this common board can be used to serve very different roles within the upgraded LHCb experiment. A continuous integration pipeline was implemented in order to automatically cross-validate the tight interaction between the different FPGA firmwares and the associated DAQ and control software, all being actively developed in parallel. We present challenges and solutions for applying this kind of practices, traditionally limited mainly to the field of software engineering, also to hardware-in-the-loop validation of FPGA firmware and SCADA-based control systems.

Topical Workshop on Electronics for Particle Physics (TWEPP2018)
17-21 September 2018
Antwerp, Belgium

*Speaker.

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0).

PoS(TWEPP2018)069

1. Introduction

The LHCb Online group provides engineering support and development to the different LHCb subdetectors, mainly in relation to readout board firmware, low-level software for frontend configuration and data acquisition, and control system components. Given the relatively small size of the core developer group (half a dozen people), the very hetereogeneous nature of the different subsystems to be supported, and the constant evolution of the needs of subdetector groups under very different circumstances (ranging from frontend validation, to test beams, to detector assembly and final commissioning), a clear need emerged for a way to reduce the time and effort required to detect, reproduce and correct integration issues during the development cycle. The following sections describe, in order, the different phases of this cycle.

2. Development

In addition to the core team, other members of the LHCb collaboration from several external institutes also actively participate in the development. The benefits of automated integration testing are apparent in such a geographically distributed structure, since issues like failing builds of failing tests can be flagged and reported automatically to the participants regardless of their location or timezone. Source control is organized using git, and git repositories are managed through the GitLab infrastructure provided by the IT department at CERN [1].

3. Code review

The readout FPGA firmware is organized into logically separate git submodules, allowing maintainers of the corresponding pieces of functionality to version their code independently. A dedicated top-level repository tracks the state of the firmware submodules, and provides a unified location from which different FPGA firmwares can be built. When developers are ready to integrate their changes in the mainline repository, helper scripts are provided that will automatically update the desired submodules and submit a merge request for review. Readout- and control-software organization follows a similar layout, spanning across multiple repositories, simplified by the absence of submodules and submodule-tracking scripts (as they can be built either independently or incrementally, unlike on the FPGA).

4. Build

Submission of a firmware merge request automatically triggers a dedicated GitLab continuous integration pipeline. This pipeline executes Questa RTL simulations according to several predefined firmware configurations and, if successful, executes the Quartus FPGA synthesis flow for each of the different applications where the common readout boards are used throughout LHCb. The flow automatically targets all required hardware platforms, including legacy (but still used and maintained) revisions of our readout boards (figures 1, 2), in addition to the final PCIe40 design to be used in Run3 (figure 3)[2]. For each board type, FPGA synthesis needs to be repeated for both the data plane and the control plane of each of the six LHCb subdetectors. Given the complexity of modern high-capacity FPGAs, producing all required permutations requires of the order of

PoS(TWEPP2018)069

100 hours of computation. Distributing synthesis jobs across a small computing cluster (~5 nodes throughout 2017 and 2018, to be upgraded at the end of 2018) reduced this turnaround to a single day, which was still unacceptable. As a further optimization, our pipeline keeps the result of each synthesis job in an internal cache and tracks changes in all firmware components across successive invocations of the same job configurations. This automated dependency tracking is used to selectively trigger only pipeline jobs whose dependencies have changed. For example: changes to a specific subdetector implementation, or to board-specific logic, will result in repeating only the jobs associated to that specific subdetector, or that specific board, respectively. In our experience, these localized changes tend to be the most common as the project matures, in these cases our optimization can shorten a day-long build down to about ten hours, suitable for overnight execution and next-day evaluation. A similar pipeline was implemented to automate the build of the readout and control software as well, however in this case the requirements are much less demanding, both in terms of runtime (minutes in place of hours) and of computational resources (virtualized Docker containers in place of dedicated hardware with fast CPUs and tens of gigabytes of main memory).

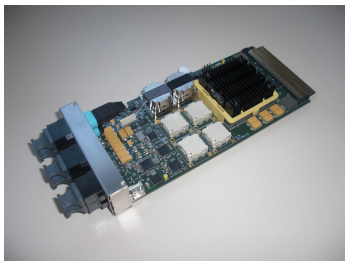

Figure 1: Legacy platform

- Stratix V FPGA
- 24 optical FE links
- 10G Ethernet readout

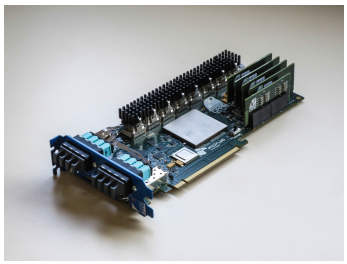

Figure 2: Development board

- Arria X FPGA
- 48 optical FE links
- PCI Express readout

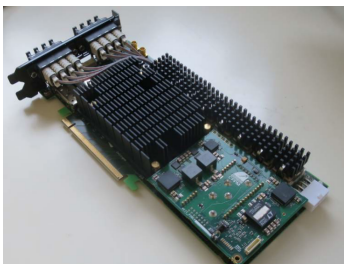

Figure 3: Final version

- Arria X FPGA
- 48 optical FE links
- PCI Express readout

5. Packaging and reporting

The build process can either fail (in which case developers are automatically alerted of the issue), or succeed and produce a series of outputs. For each operating system and architecture to be supported, the software pipeline will generate:

- Documentation
- Software executables and reusable software libraries
- Loadable kernel modules to interface the FPGA over PCI Express, or to emulate the FPGA
- JCOP components to be installed in the control system

Likewise, the firmware pipeline produces:

PoS(TWEPP2018)069

- Documentation
- SRAM Object Files to reconfigure the readout board FPGA at runtime
- Programmable Object Files to persist the FPGA firmware on the readout board flash memory
- Firmware quality reports

Firmware reports are generated by analyzing the internal Quartus result database and are stored in JSON format to be easily imported by dedicated scripts for plotting and analysis. The reports include metrics such as:

- FPGA resource utilization (ALMs, BRAMs, DSPs, HSSIs)
- System resource utilization (CPU time, wall time, memory)
- Critical path and Total Negative Slack for all clocks and all timing corners

Finally, both software and firmware are automatically packaged in RPM format and published to dedicated RPM repositories for distribution, as is customary on all Linux installations deployed at CERN. The repositories provide two distinct release channels. The *unstable* channel reserved for new features and fixes undergoing testing, and the *stable* channel meant for validated releases ready for use by the rest of the LHCb collaboration.

6. Deployment

While using RPMs as a distribution mechanism simplifies update management and cross-package compatibility checking, it is not sufficient to fully describe (and reproduce, in case of issues) a running system that might be deployed in a distant institute or production site. To improve traceability, dedicated tools were implemented to extract additional information from a running system, including:

- JCOP components installed in the control system and their versions
- Readout hardware being used and state of the corresponding kernel drivers
- Readout firmware being used, including checksum, timestamp, toolchain version, git hashes and versions of all the constituent submodules

The pieces of information in the last item of the list are automatically embedded in the FPGA images during synthesis, and then retrieved in the field from a programmed FPGA over USB or PCI Express.

7. Testing

The LHCb control system is based on the Siemens WinCC Open Architecture [3]. WinCC OA is used for configuring, monitoring and controlling the state of a system as small as a tabletop setup in a lab or a testbeam, or as big as the entirety of LHCb. Successfull operation of either

PoS(TWEPP2018)069

kind of setup requires all hardware, firmware and software components to correctly integrate with each other. In order to validate this kind of integration we have instrumented WinCC OA with dedicated tools and scripts, implementing an automated test harness. A test harness consists of a linear sequence of steps, each step representing operations that an operator or an automatic run control would execute on the system (for example: programming a readout board, synchronizing all its clocks sources, configuring its optical links, configuring some frontends, starting a run, reading out data, and so on...). The process generates a report in TAP format. The "Test Anything Protocol" (TAP) is a formal specification used for communication between unit tests and a test harness.

8. Acceptance

After testing (both manual and scripted), maintainers can decide to accept currently unstable versions (thus promoting them to become the new stable releases) for the firmware and software modules that have been modified since the last release cycle, or to iterate more in case of issues. Depending on complexity, new code can either be fast-tracked for immediate release, or require up to days or weeks of iteration. In either case, the release process has also been automated to require, for the maintainer, only a single operation through git or the GitLab interface for each repository to release from.

9. Conclusions

At the time of this writing, continuous integration has been part of our development flow for over one year. Initially introduced to assist in the software transition from the SLC6 to the CC7 operating system, and from the initial 10GbE-based prototype to the final PCIe-based FPGA readout, its scope has progressively increased to encompass automated firmware simulation and compilation for the six LHCb subdetectors (each requiring multiple configurations) across three hardware platforms, and automated packaging and deployment of JCOP components and WinCC OA projects, while still retaining support for legacy hardware and software.

Broken builds are quickly identified and can be immediately acted upon, while nightly builds ensure that firmware images for all desired configurations are always up to date, immediately available for deployment, and easily traceable for comparative testing.

As the LHCb upgrade advances towards Run3, these tools and practices will prove invaluable in ensuring a smooth ramp up and commissioning of the readout system for the entire experiment.

References

[1] A. G. Alvarez, B. Aparicio Cotarelo, A. Lossent, T. Andersen, A. Trzcinska, D. Asbury, N. Hlimyr and H. Meinhard, "Extending software repository hosting to code review and testing," J. Phys. Conf. Ser. **664** (2015) no.6, 062018. doi:10.1088/1742-6596/664/6/062018

[2] J. P. Cachemiche, P. Y. Duval, F. Hachon, R. Le Gac and F. Réthoré, "The PCIe-based readout system for the LHCb experiment," JINST **11** (2016) no.02, P02013. doi:10.1088/1748-0221/11/02/P02013

[3] L. G. Cardoso, C. Gaspar, J. V. V. Barbosa and F. Alessio, "Controlling DAQ electronics using a SCADA framework," doi:10.1109/RTC.2016.7543123

PoS(TWEPP2018)069