GitLab-CI for FPGA development at LHCb
The LHCb experiment
The LHCb experiment
Proprietary hardware

We receive data from the detectors (proprietary rad-hard protocols over optical fiber), process it and emit it in a “COTS-friendly” format. We have to produce different firmware (sometimes more than one) for each device and for each sub-detector.

AMC40 (Legacy)  PCIe40 (Production)

GBT  Front-end rad-hard optics  GBT

10GbE  Back-end high-speed links  PCIe
Our use case

- FPGA firmware
  - Multiple repositories (submodules)
  - Languages: VHDL, Verilog, TCL
  - Toolchain: Quartus (proprietary)
  - CI: shell runners

- Low-level software
  - Multiple repositories (independent)
  - Languages: C, C++, Python
  - Toolchain: GCC, Python...
  - CI: shared runners

- SCADA middleware
  - One repository
  - Languages: CTRL (Siemens proprietary)
  - Toolchain: PVSS (proprietary)
  - CI: dind? [wip]

- Each has its own CI pipelines, but in the end we also have to test the integration of the different pieces working together ("CI²")
**FPGA development 101**

- Circuit is defined in a Hardware Description Language
- Simulator + testbench + input vectors reproduce behavior of device and allow some form of debugging
- Compiler toolchain maps specification into device-specific gate array configuration and interconnection
  - Lots of TCL scripts
  - 48 GB RAM!! (vendor recommendation)
  - As many GHz as you can afford
  - Still very time consuming
- Workarounds
  - Distribute different compilations to different machines
  - Reuse artifacts aggressively
FW synthesis combinatorics

“MiniDAQ”

Control Plane

Data Plane

TEST

SciFi/FF
SciFi/FV

MUON

VELO

RICH/LO
RICH/HO

CALO

UT/x3
UT/x4
UT/x5

“MiniDAQ”

Control Plane

Data Plane

TEST

SciFi/FF
SciFi/FV

MUON

VELO

RICH/LO
RICH/HO

CALO

UT/x3
UT/x4
UT/x5

Debug images

Timing Plane

AMC40

PCle40v1

PCle40v2.x

03/10/2018

GITLAB@CERN DAY

7
Firmware pipeline

- Executed nightly and when users send changes upstream
- 33 compilations (for now)
- Most compilations take several hours (somewhere between 4 to 10 depending on complexity)
- Custom logic to avoid a rebuild whenever possible (EDA tools have very poor dependency tracking)
Low-level software and kernel drivers for our custom hardware
Main platform is CERN Centos7
We still build for SLC6 and i386 for legacy platform
SCADA “pipeline”

Program FPGA ➔ Subscribe ➔ Reload ➔ Read information

Check PRBS ← Check RxCounters ← Check RxReady ← Configure

Start Run ➔ Check Triggers ➔ Stop Run ➔ TAP report
GitLab issues

   StuckCiJobsWorker wrongly detects, cancels 'stuck' builds when per-job timeout is more than an hour
   - Some of our jobs can easily require several hours
   - We have less machines than we have jobs
   - For now, we set up cronjobs running in background that use the GitLab API to find jobs that have been killed for inactivity and resuscitate them

   Relative submodule link to a nested project fails to resolve
   - Our firmware flow relies on submodules
   - A user should be able to click on a submodule and open the corresponding repository at the correct commit state
   - It’s already fixed on gitlab.com but not yet in the instance deployed at CERN
Final remarks

- Overall, our experience with GitLab has been very positive!

- I’m slowly converting everyone in our group to use GitLab CI
  - HDL developers are not a typical audience for this kind of tools
  - SCADA developers depend on proprietary GUI tooling which is hard to integrate

- Work in progress
  - Package SCADA projects into self-contained Docker containers
  - Validate simulation output before attempting a firmware build
  - Automate hardware-in-the-loop integration tests

- Ideally
  - Deploy/rollback from GitLab (what would this look like?)
  - Publish metrics and monitoring hardware performance (Grafana/Prometheus?)
"I SPEND A LOT OF TIME ON THIS TASK. I SHOULD WRITE A PROGRAM AUTOMATING IT!"

**THEORY:**

```
WORK

WRITING CODE

WORK ON ORIGINAL TASK

AUTOMATION TAKES OVER

FREE TIME

TIME
```

**REALITY:**

```
WORK

WRITING CODE

DEBUGGING

ON GOING DEVELOPMENT

RETHINKING

NO TIME FOR ORIGINAL TASK ANYMORE

TIME
```
Firmware pipeline - stages

Jobs in “Prepare” stage

- create_mr
  - Users run a script to submit changes upstream
  - Pipeline picks up the new branch and creates a merge request automatically

- cache_master
  - Previous compilations are preserved and cached
  - Custom TCL script during FPGA synthesis checks project against git history for modifications

- clean_mr
  - Ensure the merged branch is deleted
  - Update cache for master branch with last compilation

Jobs in “Tag” stage

- create_tag
  - Manual job
  - When maintainer wants to create a new release, a tag is created according to our naming convention

- clean_tag
  - Remove project cache for tag once firmware has been released
  - Every cache amounts to tens of gigabytes