Gitlab CI for FPGA/SoC Projects

Carmen Marcos, Christos Gentsos*
(IT-CA-GES)

6 October 2023

Outline

- What is CI, why do we care?
- Project purpose and objectives
- Gitlab CI basics
- Gitlab Runners, our cluster
- Defining CI jobs
- Lazy pulling
- Existing images – possible improvements
- Current state and future plans
- Questions?

What is CI, why do we care?

What is CI in FPGA/SoC Projects?

- Continuous Integration: first used in software development
- Involves *regularly* and *automatically* integrating code changes from multiple contributors into a shared codebase
- The term has come to be mostly used to refer to automatic testing and building at the repository level → a *pillar* when it comes to applying CI principles

Benefits:

- Automated and consistent system for various types of tests (unit tests, system-level tests, linting, code coverage, ...) and builds (bitstream, drivers, utilities)
- Whole sets of jobs easy to launch → used more often → less breakage, easier bug-hunting on regressions
- Facilitates common development and code sharing (could be IP core libraries)

Project purpose and Objectives

Why this project?

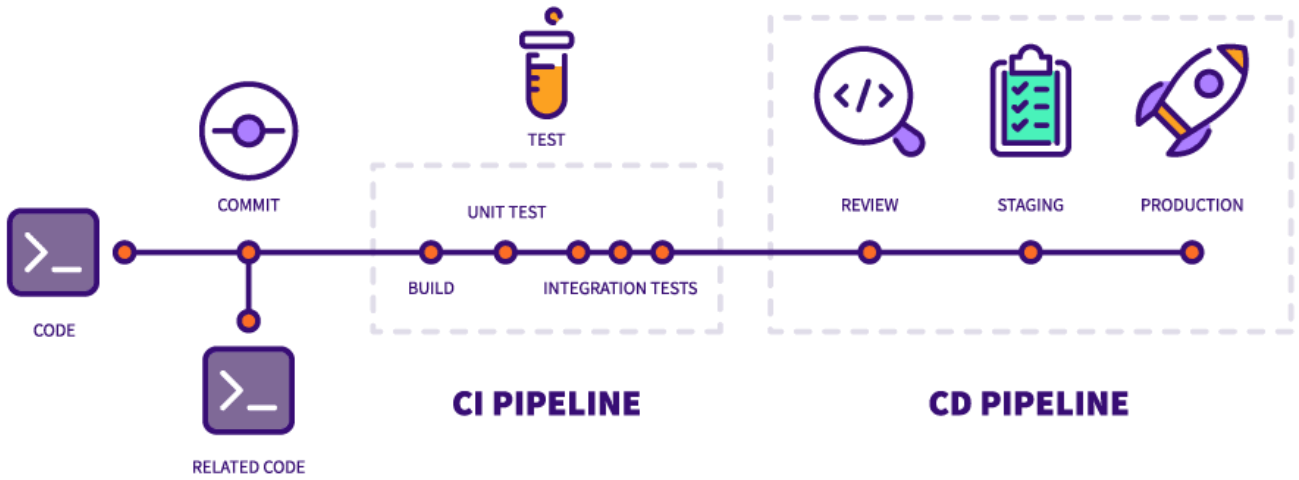
- Enabling CI practices can greatly benefit FPGA/SoC design
- Some teams have already done considerable work to get there, but lack of centralized infrastructure is a pain:
  - Hard to efficiently set up and (most importantly) maintain build clusters
  - Electronics engineers are not IT → even harder
  - Huge toolchains (O(100GB) for toolchains) don't make it any easier, either
  - Effort multiplied across all the different teams, very inefficient at scale

Our objectives:

- Scalable, maintained VM-based cluster to run the CI jobs, adapted to the resource requirements of EDA tools (IT-PW)
- Easy to use Docker images for EDA tools, like simulators and toolchains (IT-CA)
- Document the above clearly to support the 100+ projected users

Gitlab CI basics

- Code changed often → sometimes changes reach the main branch many times/day
- Each change can introduce a bug, especially when touching fundamental building blocks in complex systems
- Ideally one should run a “full” test suite and build after each change to the main branch (or on a schedule – nightly, for example)

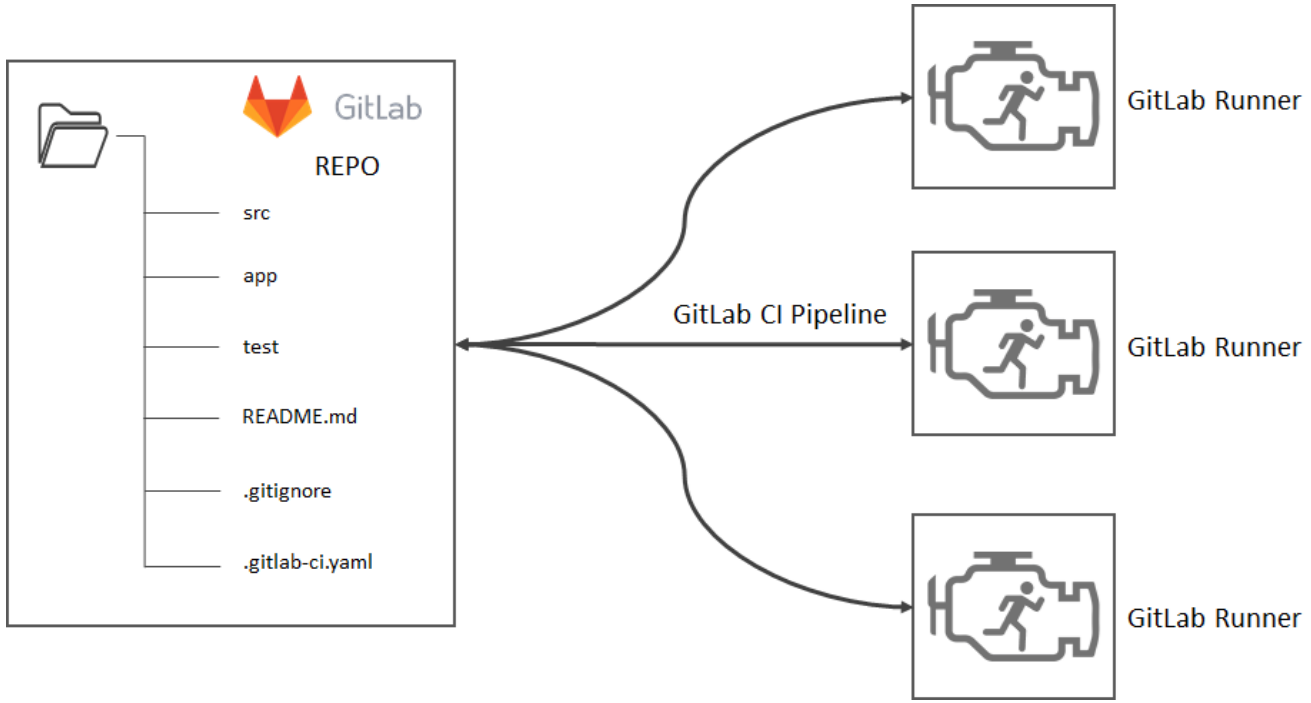


Gitlab CI gives tools to do exactly that

- Pipeline can be started after each push or merge request
- A pipeline can bundle many jobs together: tests, builds, doc generation
- One can also test or even deploy on real HW (out of scope)

Gitlab Runners, our cluster

- A runner is a computer (or a k8s cluster) that will execute CI jobs from a pipeline
- It can be either private, registered to a project/group; or *instance* **(shared),** *visible to all projects*
- A runner is assigned one or more tags, depending on its purpose / capabilities (e.g. can be fpga-small or fpga-bigmem in our cluster)



- The cluster that we will deploy will mainly consist of (20-30?) instances with 29GB RAM, 16VCPUs, 160GB SSD → fine for most tasks
- We can get quota for a few machines of double that size (60GB RAM)
- We can request quota for more machines if we see that we need it
  - easy to scale up or down

Defining CI jobs

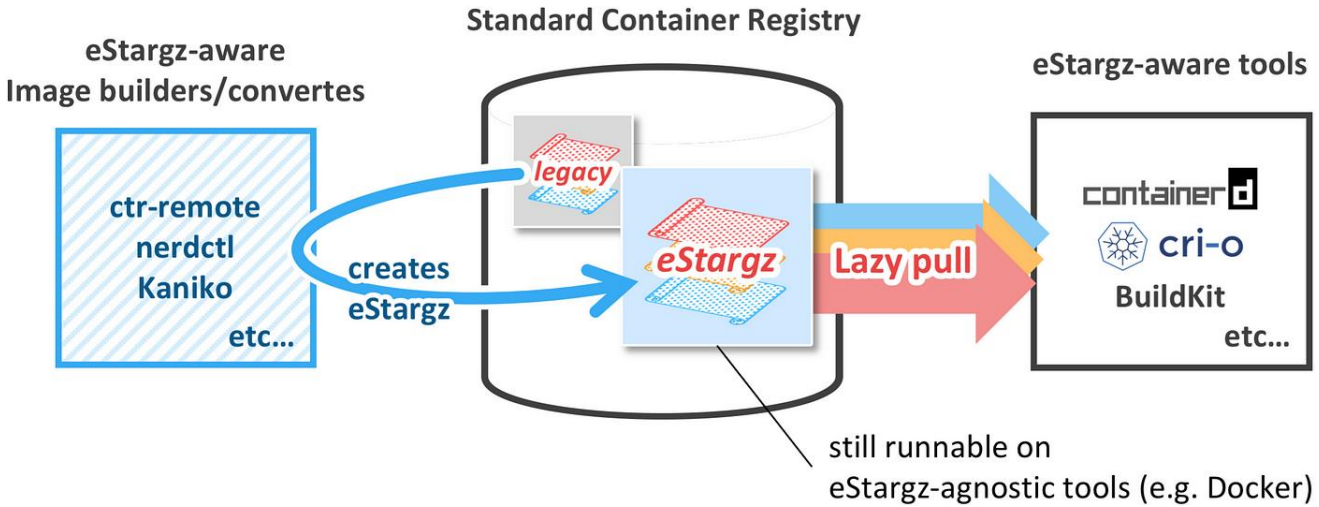
```
stages:
  - test

test-job:
  stage: test
  tags:
        - fpga-mid
  image: registry.cern.ch/ci4fpga/vivado:2022.2
  script:
        - vivado -mode batch -source build.tcl
  artifacts:
        paths:
        - "*.rpt"
        - "*.bit"
```

- In a special file (.gitlab-ci.yml) we can declare the stages and jobs that make up a pipeline
- For each job, we assign a runner tag and a docker image name (e.g. fpga-bigmem and registry.cern.ch/vivado:2023.1), the commands to be ran and any resulting files to be kept
- The right runner type will be instantiated based on the tag, downloading the docker image and running the specified commands
- If all goes well, the resulting files can be found on the gitlab web page

Lazy Pulling

- To pull the image can be one of the most time-consuming steps in the container lifecycle…
  - …but in many cases only a small part of the pulled data is ever read
  - Imagine a case of an FPGA toolchain installation, full of 10s of GBs of device data for all families – but we only need one family in a run!

- Lazy pulling allows a container to be started with just the necessary data pulled
  - 100GB image as an example: 15 secs vs 15 minutes (with a 1Gbps connection)
  - eStargz format, allows chunks of data to be fetched on-demand



Existing Images

Toolchains

- **Xilinx ISE**
  - 14.7
- **Xilinx Vivado**
  - 2018.1
  - 2018.3
  - 2019.1
  - 2019.2
  - 2021.1
  - 2021.2
  - 2022.1
  - 2022.2
- **Microsemi Libero**
  - 11.9
- **Intel Quartus Prime**
  - 20.1

Simulators

- **Aldec Riviera-Pro**
  - 2021.04
  - 2022.10
  - 2023.04
- **Mentor ModelSim**
  - 10.7d
- **Mentor Questasim**
  - 2022.1
- **Ghdl**

Other

- **OpenOCD (Ubuntu)**
- **Doxygen**
- **Typhoon HIL**
- **PetaLinux**
  - 2018.1
  - 2019.2
  - 2021.1
  - 2021.2
  - 2022.1
  - 2022.2

More to be added...

Possible improvements

- Adoption of lazy pulling is only one of those
- We can adapt the images to run the tools as the user that launched the job
  - This can enable us to use Europractice licenses with CI (not possible today)
- We can consider implementing license checks that run on container startup to avoid CI hogging licenses when humans need them
- Richer images (we don’t care about making the image larger because of lazy pulling)
  - Pre-compiled libraries for multiple releases of each vendor toolchain
  - Possibility to have tools like embedded compilers etc pre-installed
- Consistent images open up the possibility of the community sharing and collaborating on common, “base” build “recipes” that might be used across multiple projects and teams

Current state and future plans

Still a work in-progress

- Most of the Dockerfile adaptation needed to utilize lazy pulling is complete
- The staging cluster has been used to implement small projects, very fast (around 3 minutes to bring up a new VM, transfer the necessary parts of the otherwise huge Vivado, and generate a bitstream)
- Beta-testing by ATS and EP users at the end of 2023 will be available as proof of concept

Expected Outcome:

- Assist common development and code sharing
- Eliminate technical burden of creating and maintaining the necessary infrastructure
- Provide and maintain the simulation and synthesis tools and their licensing configuration
- Make all this available to both accelerators and the experiments

Current state and future plans

Please feel free to join the ***ci4fpga-beta*** e-group to get informed when the beta is available!

Many thanks!

Any questions?