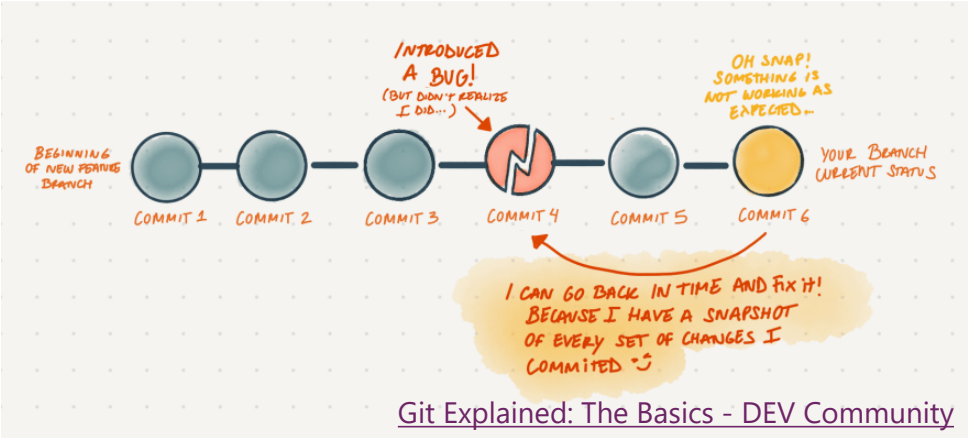
GitLab and CI for Vivado Projects

Chaowaroj (Max) Wanotayaroj
Makoto Tomoto
Yasuyuki Okumura

KEK Inter-University Research Institute Corporation High Energy Accelerator Research Organization
素粒子原子核研究所 Institute of Particle and Nuclear Studies
東京大学 素粒子物理国際研究センター International Center for Elementary Particle Physics The University of Tokyo

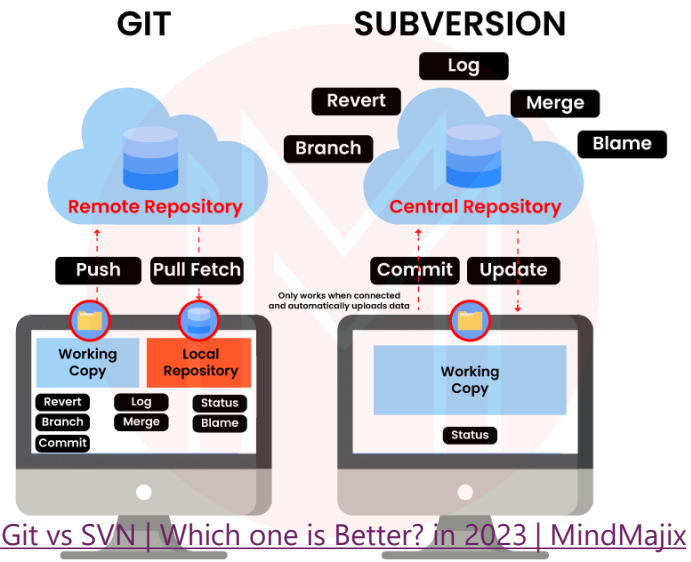
Introduction – Git

- For code development, we want to keep track of changes in each version



Git Explained: The Basics - DEV Community

- In the past decade, Git has emerged as the standard for version control system
  - Replacing the Subversion (SVN)
- Git is a distributed version control system
  - Full code+history on your machine
  - Faster for most operations

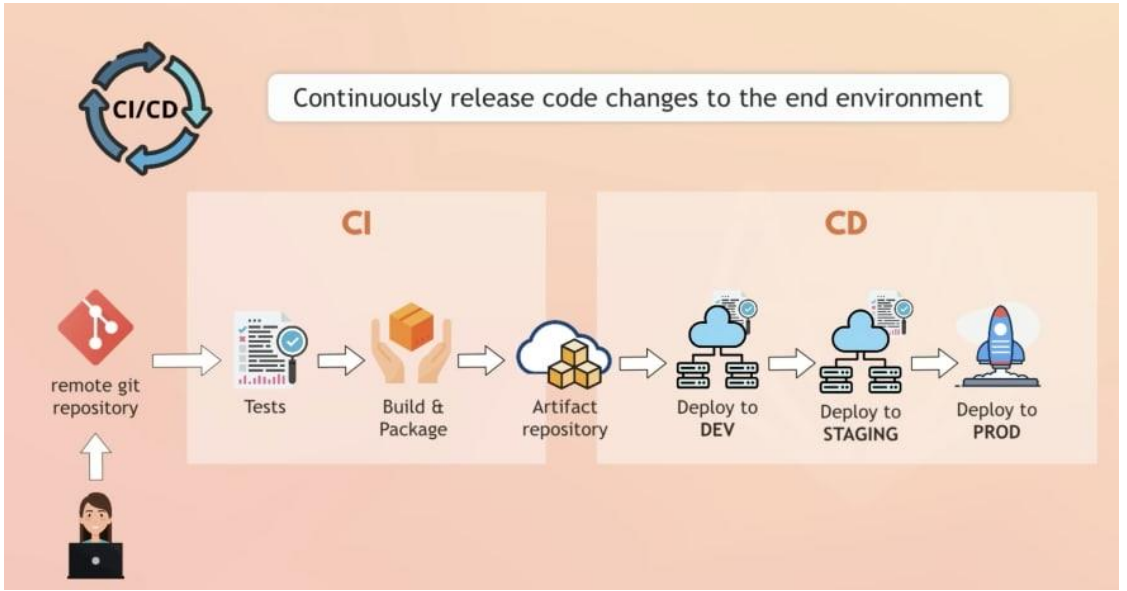


▷ Git vs SVN | Which one is Better? in 2023 | MindMajix

Introduction - Git vs. GitHub

- A service such as GitHub or GitLab let you host a Git repository
- And offer other features on top of it
  - Bug tracking
  - Automate building (compiling, etc.)
- GitLab CI/CD: continuous integration and continuous delivery
  - CI: Test&Build
  - CD: Deploy

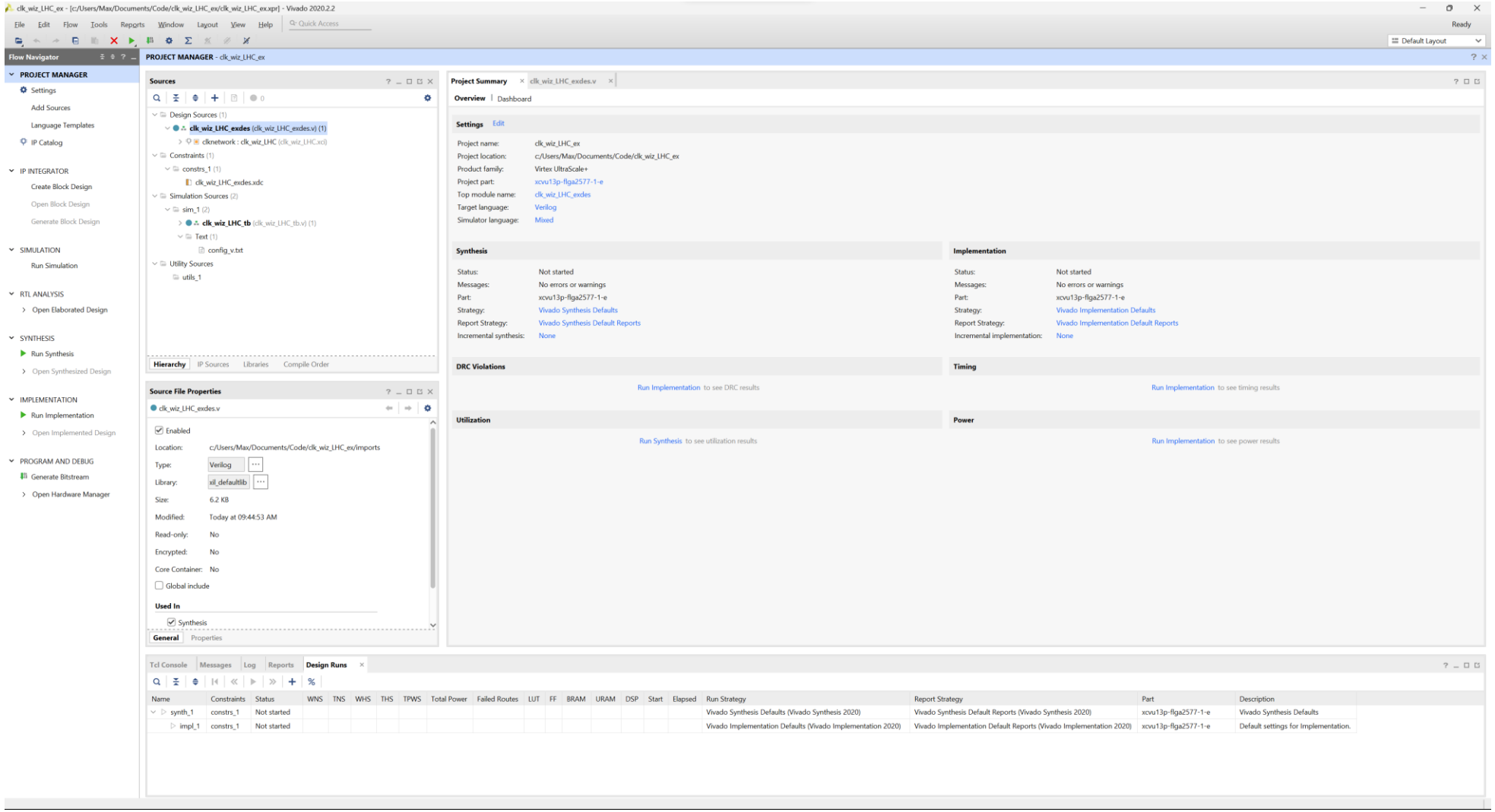
GitLab



Vivado IDE

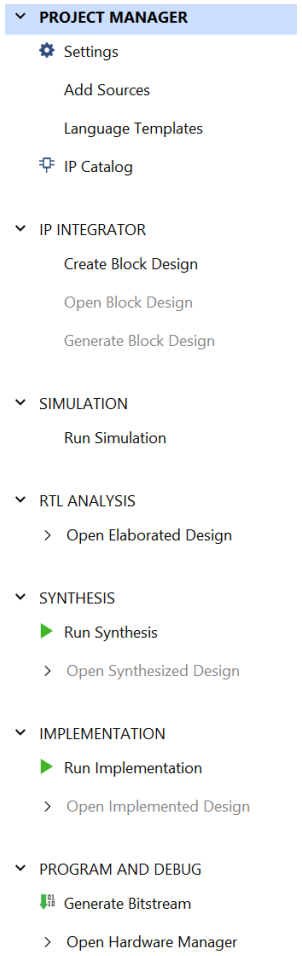
Vivado IDE

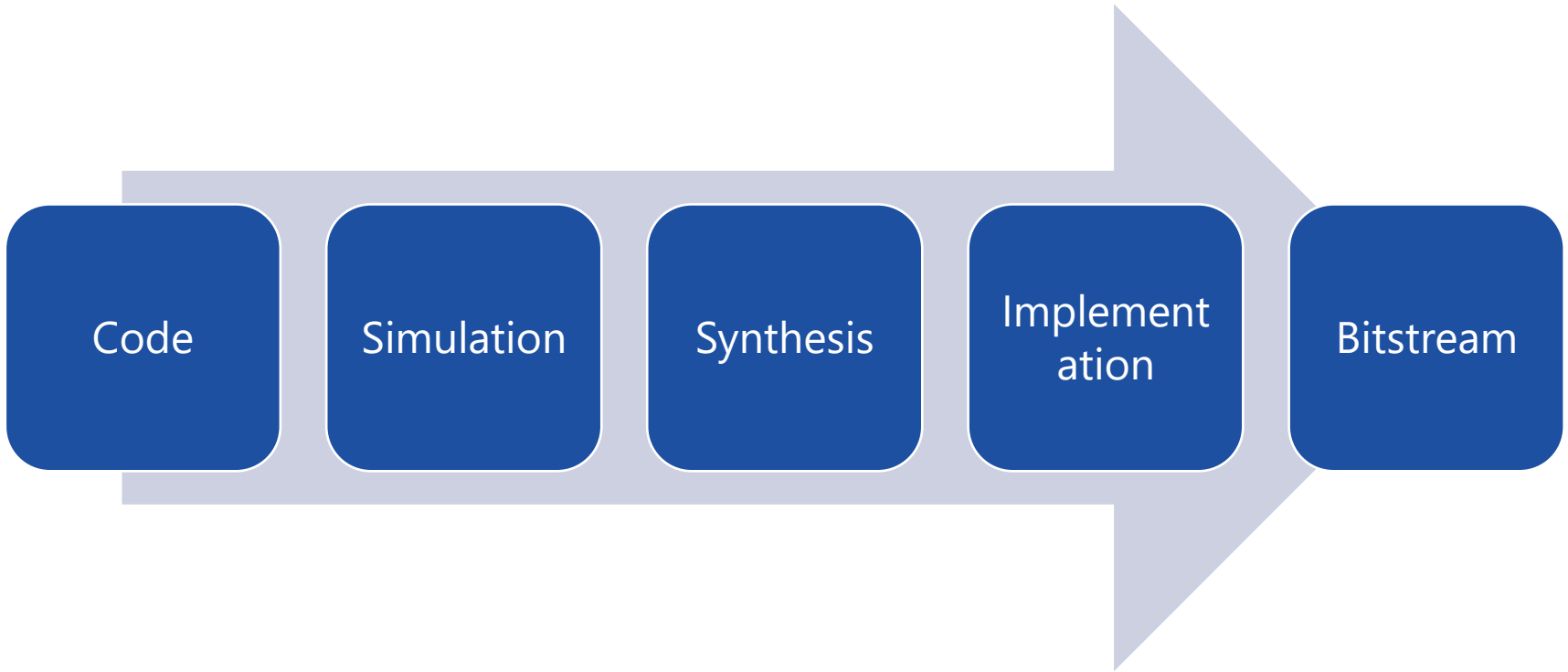
- Vivado is an IDE (Integrated Design Environment) by Xilinx/AMD for FPGA firmware design



Design Flow

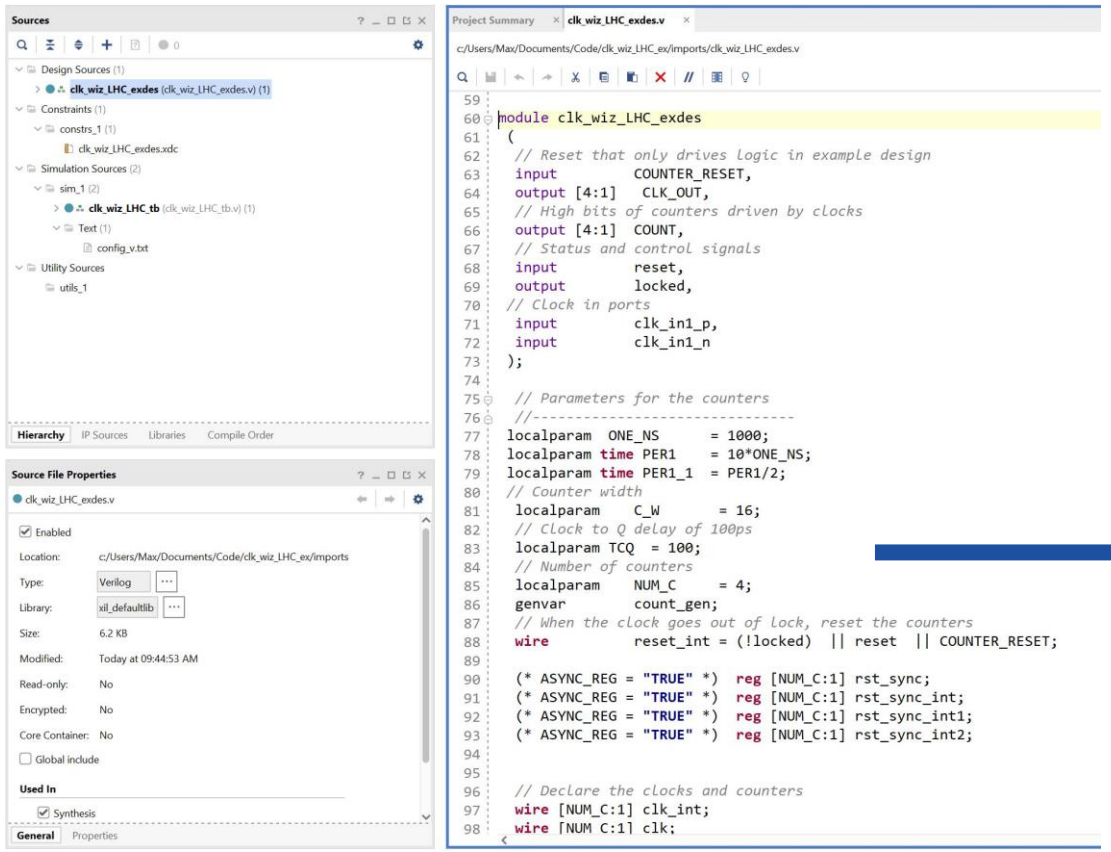
- Typical design flow for a FPGA firmware

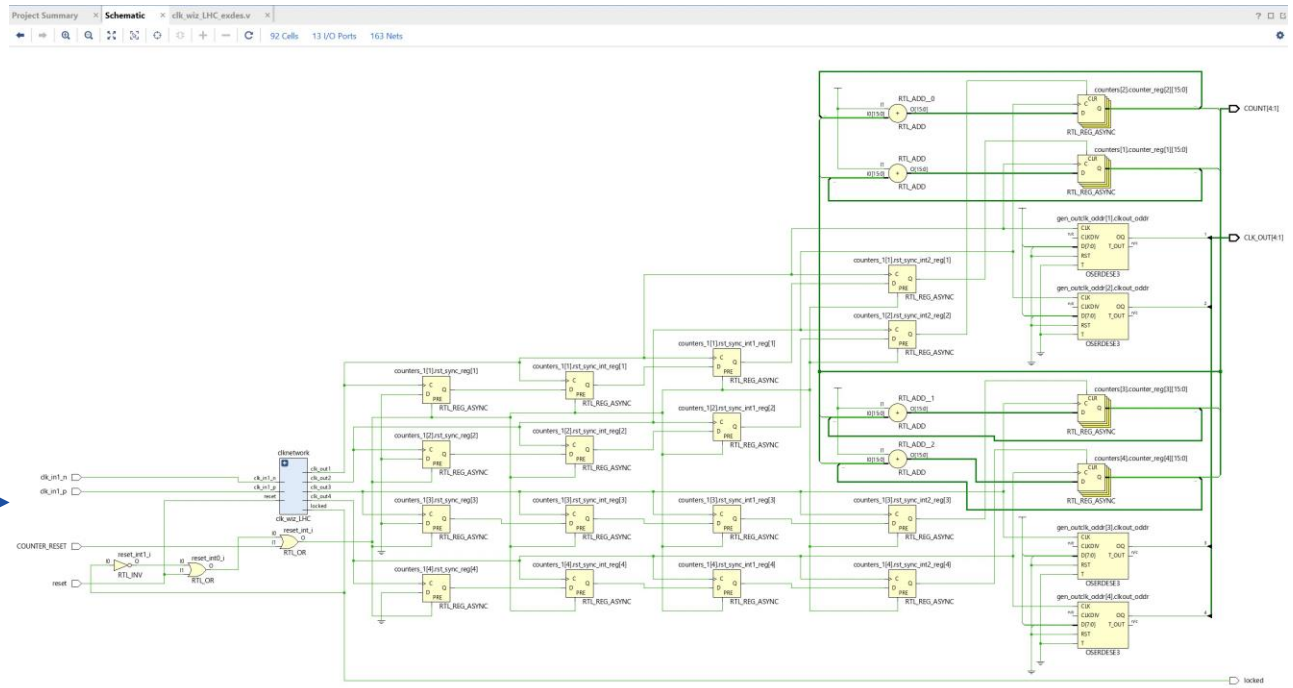




Code

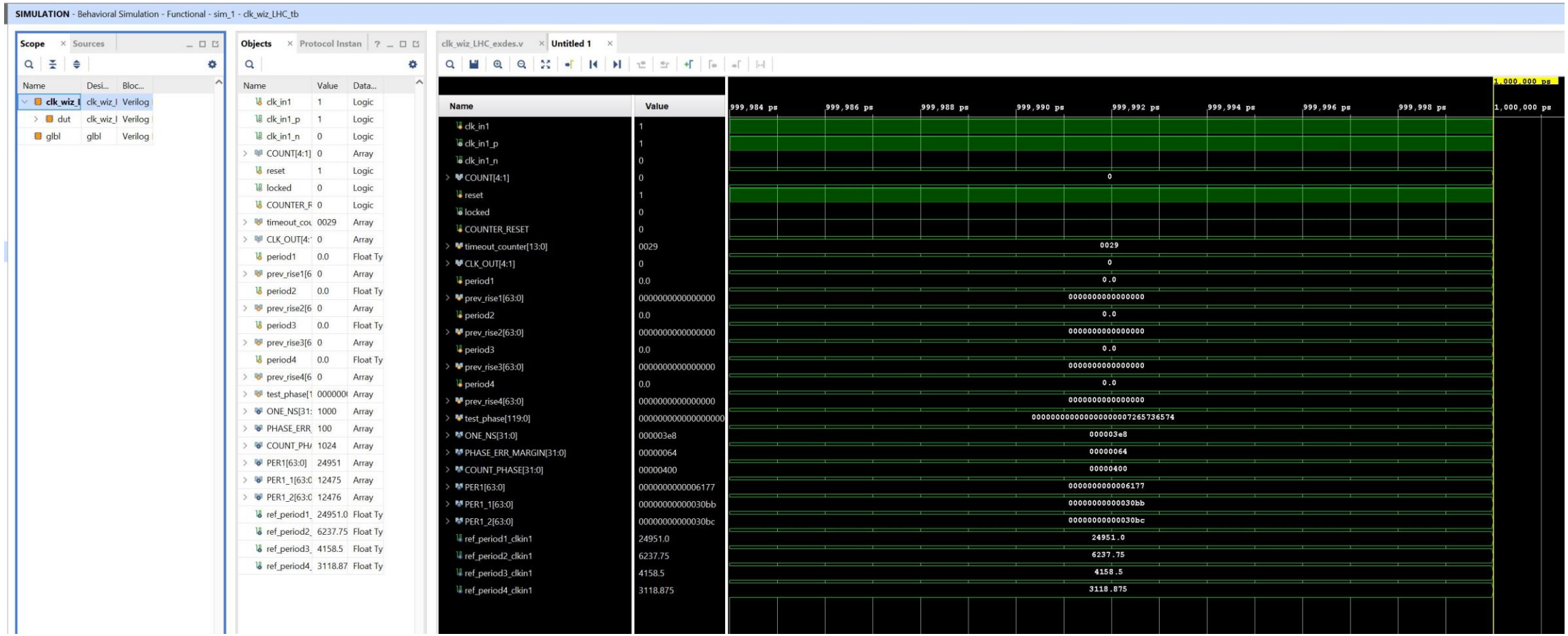
- Register Transfer Level (RTL) text or block designs
- Constrains
- Schematic: Graphic representation





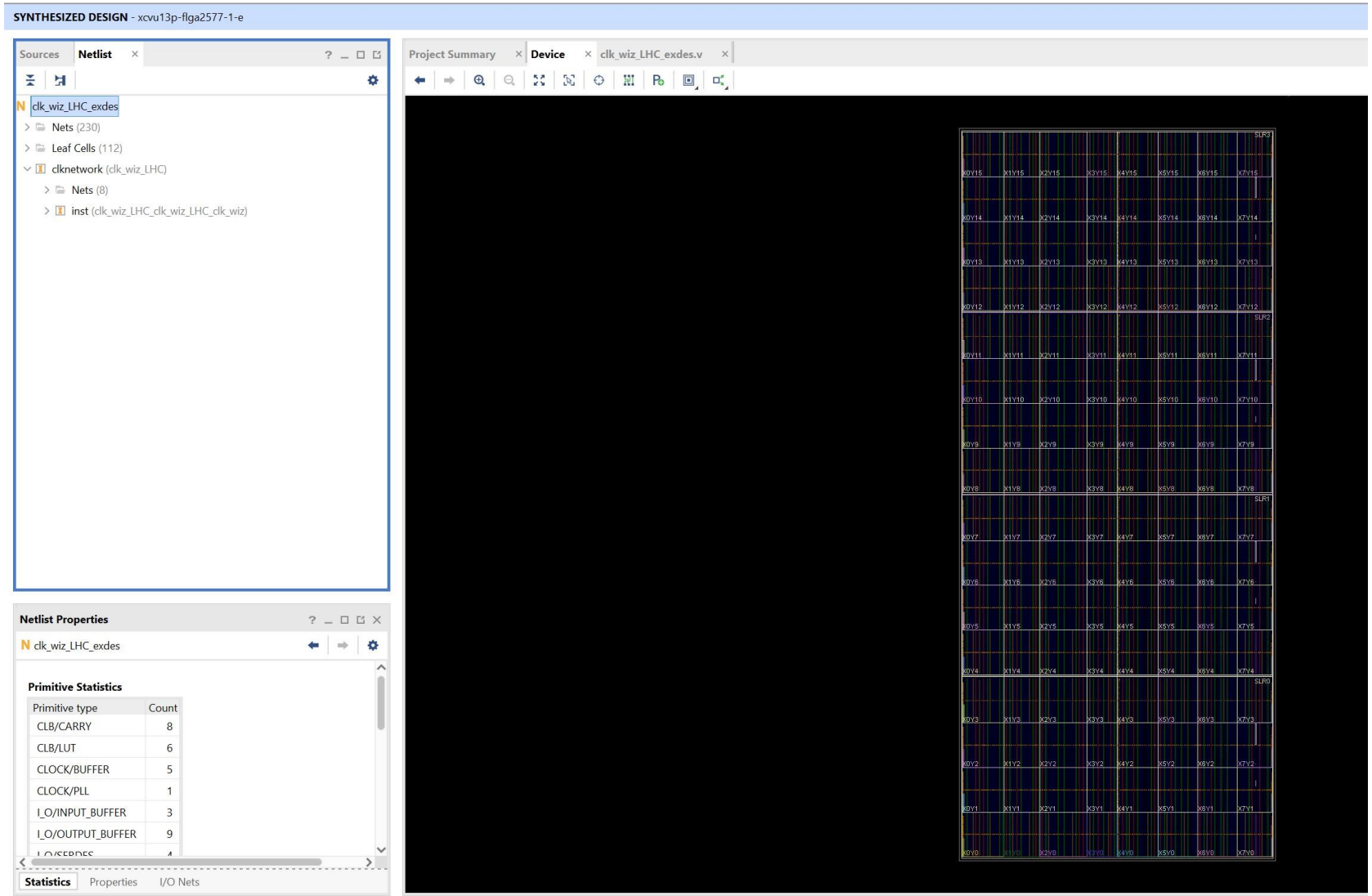
Simulation

- Check behavior of the design



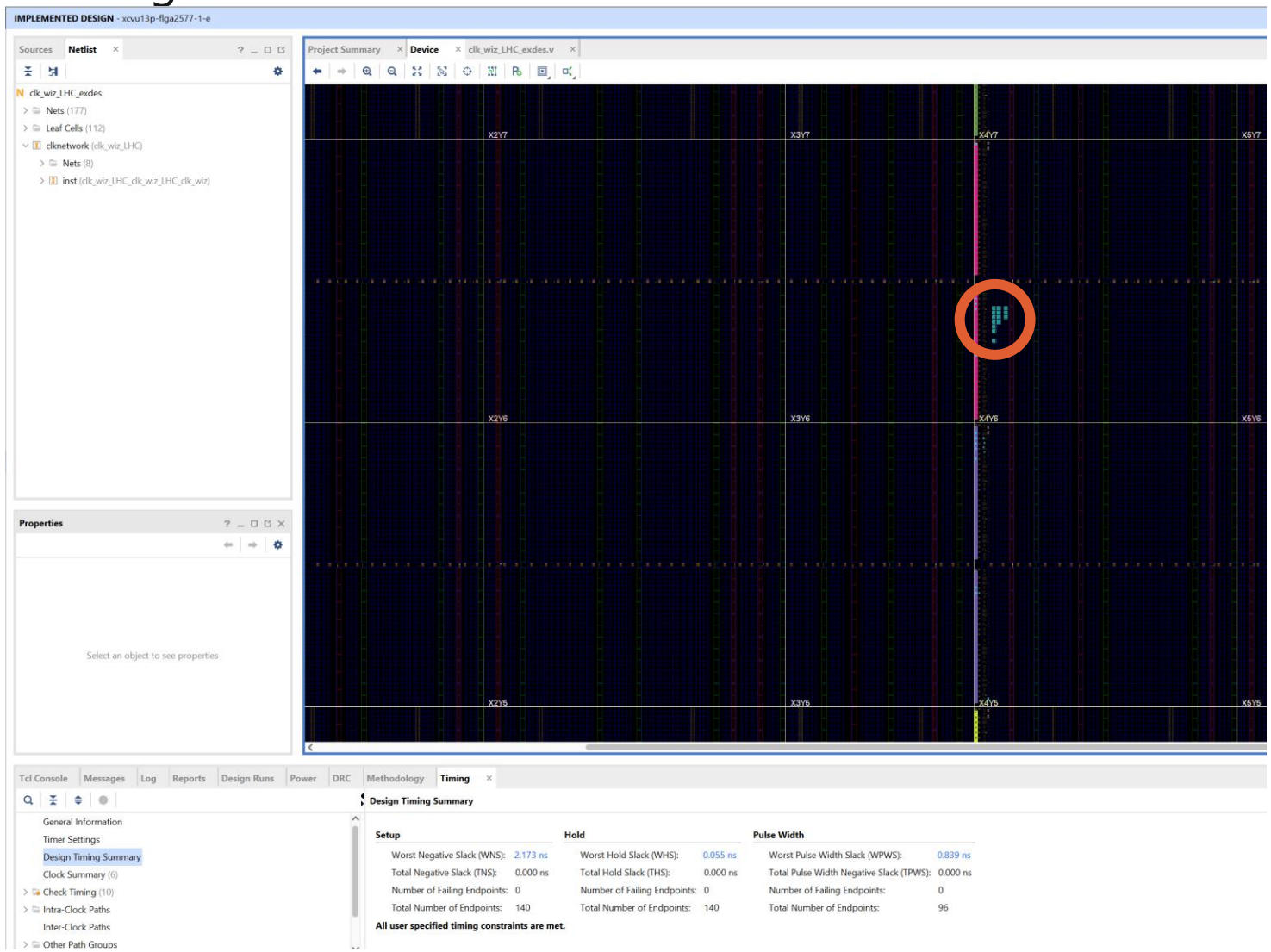
Synthesis

- "Compile" the design into list of logic gates and other FPGA primitives



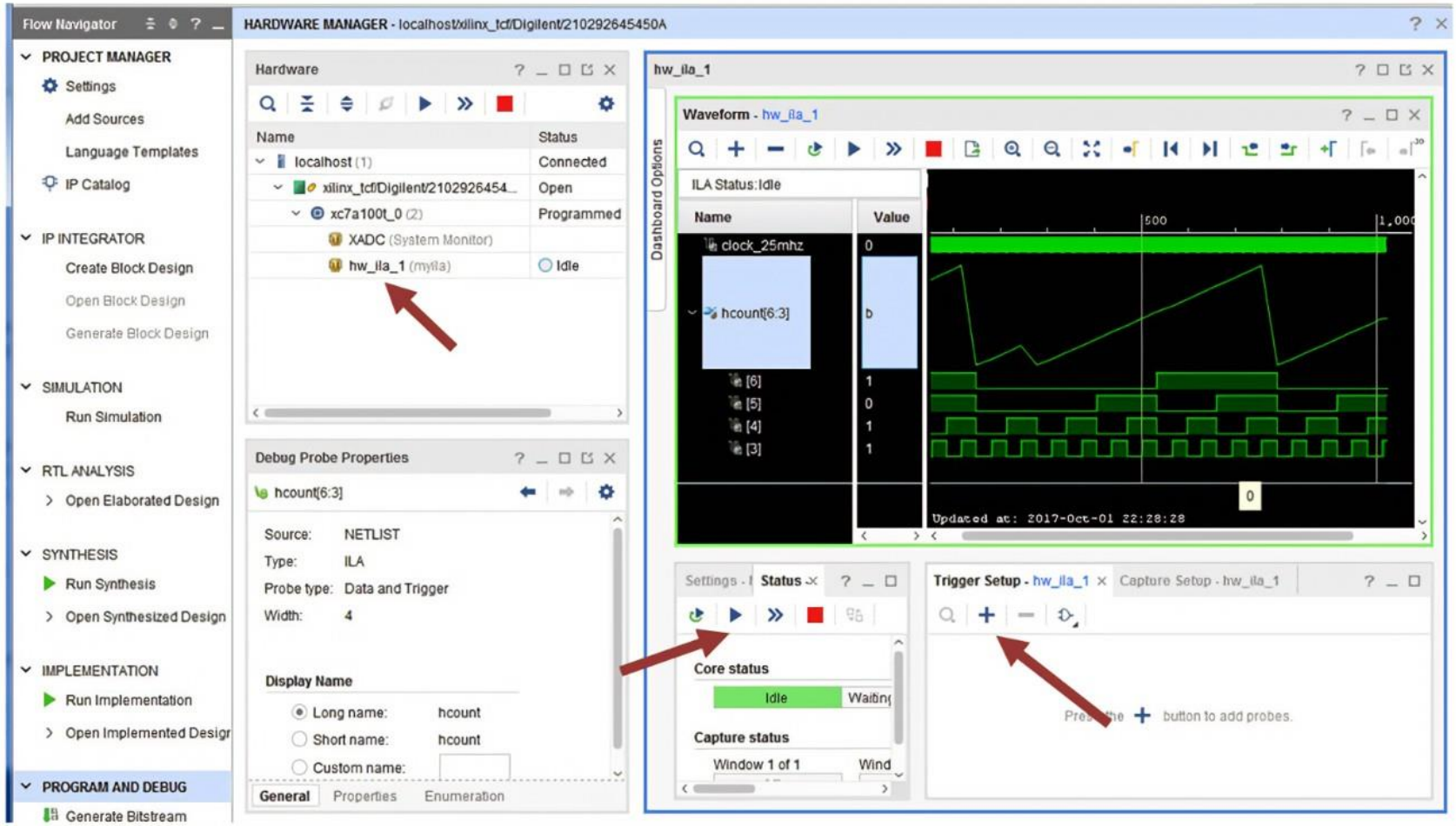
Implementation

- Map the synthesized design to actual hardware



Bitstream

- Bitstream to be programed on hardware
- Check and debug



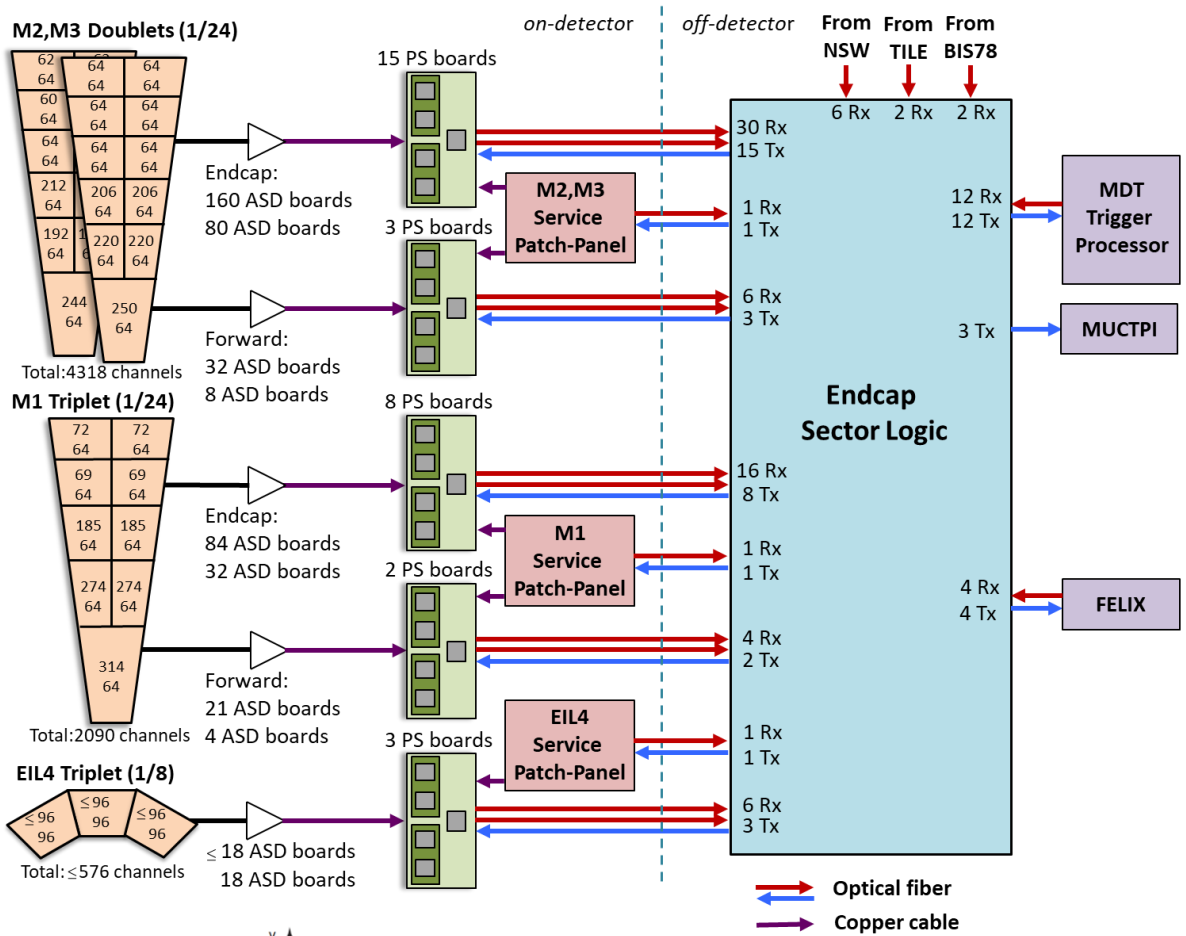
Integrated Logic Analyzer ILA (mit.edu)

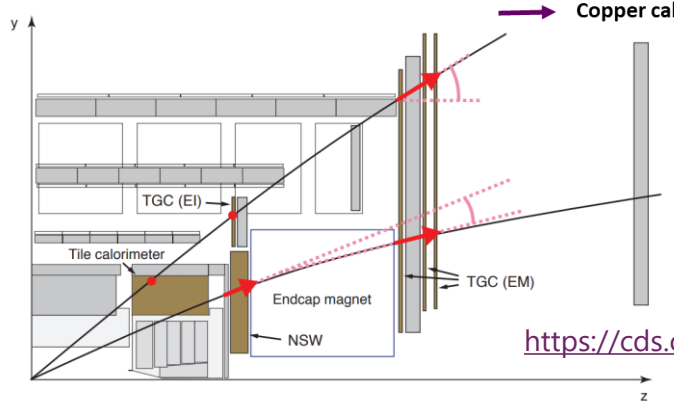
The Sector Logic (SL) Board

SL Board

- Develops for TGC Phase-II upgrade
- On the off-detector side of ATLAS
- Gathering and processing data from many parts of the detector
  - One Endcap Sector Logic board covers 1/24 of the Big Wheel TGC (one forward trigger sector and two endcap trigger sectors)
  - Also receive information from NSW and Tile calorimeter
  - Provide L0 MDT Trigger Processor with relevant data and receive back reconstructed tracks
- Sending all hit information to the SL board allow more sophisticate algorithm

https://cds.cern.ch/record/2285580

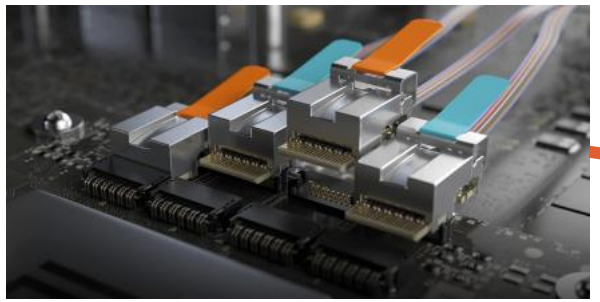




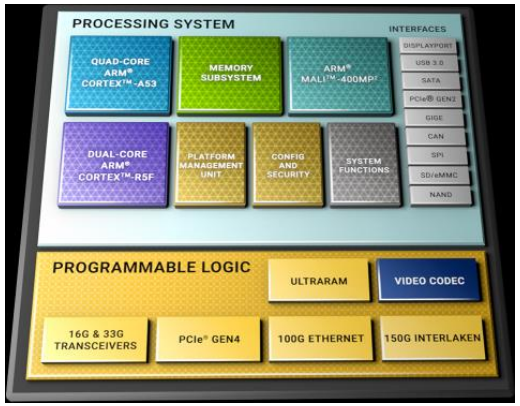
https://cds.cern.ch/record/2285584

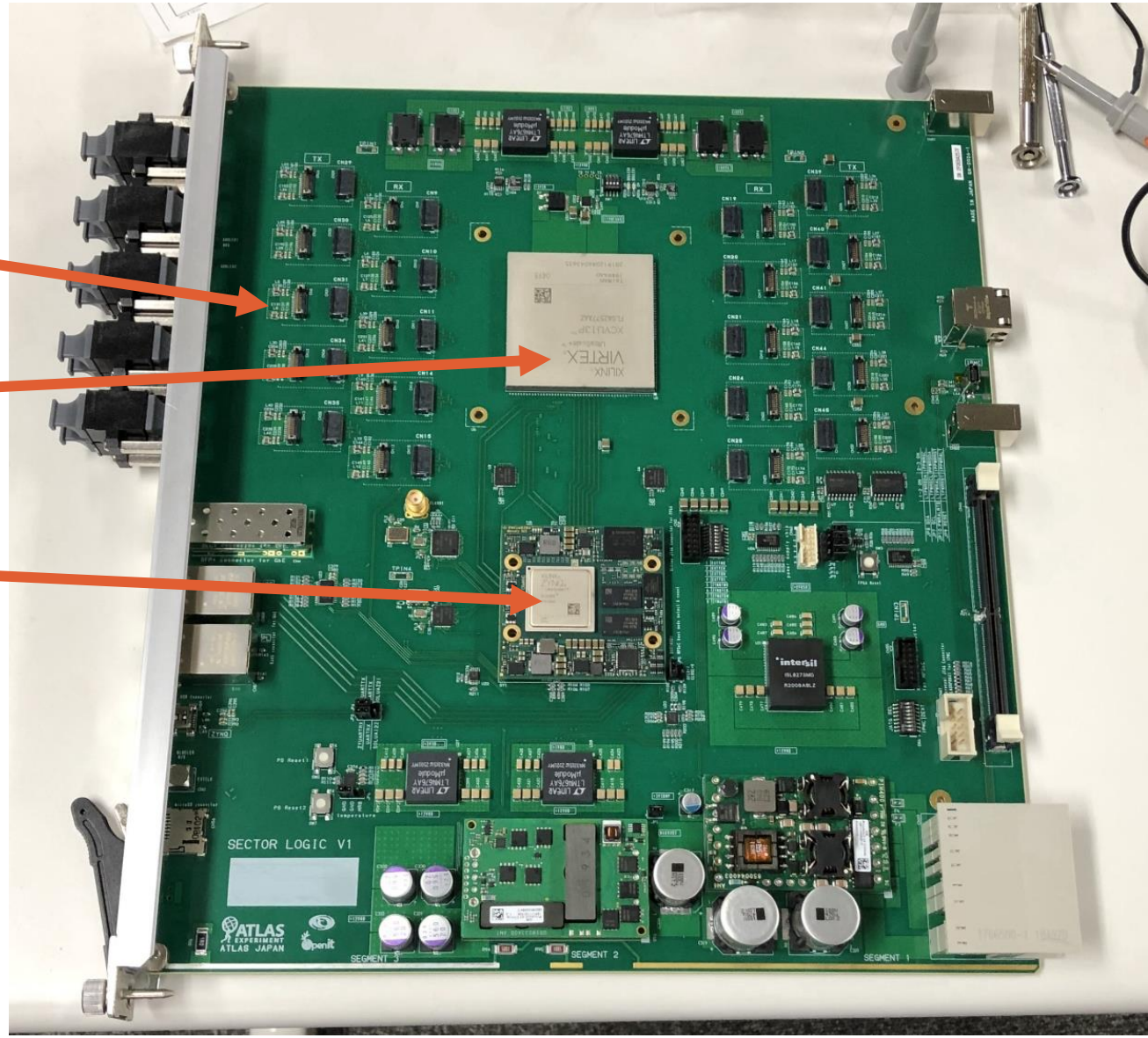
Hardware (for this talk)

- Firefly connectors (elec->optical transceivers)



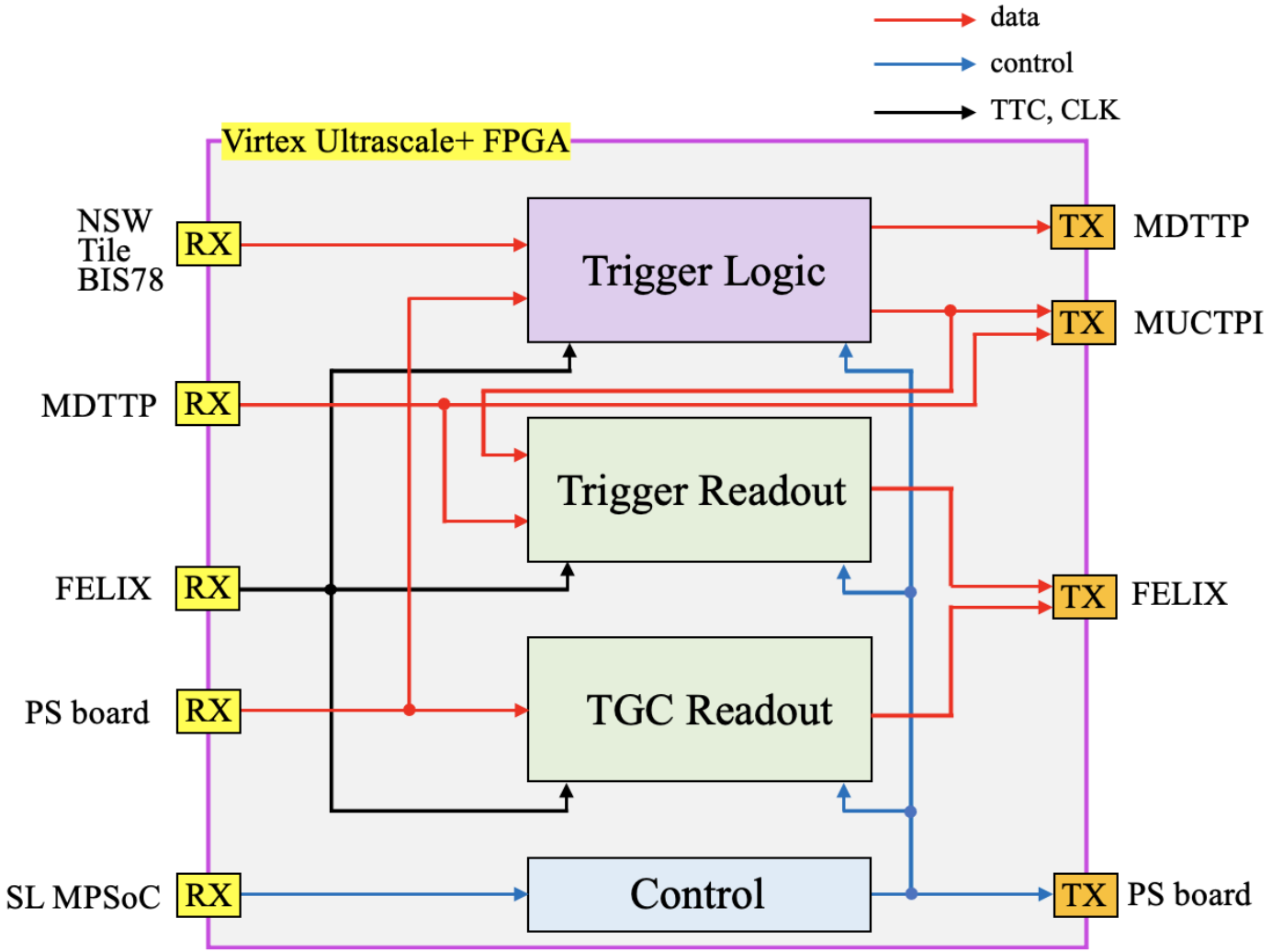
- Virtex Ultrascale+
  - High-performance FPGA
- Zynq Ultrascale+ MPSoC
  - Multiprocessor System on a Chip





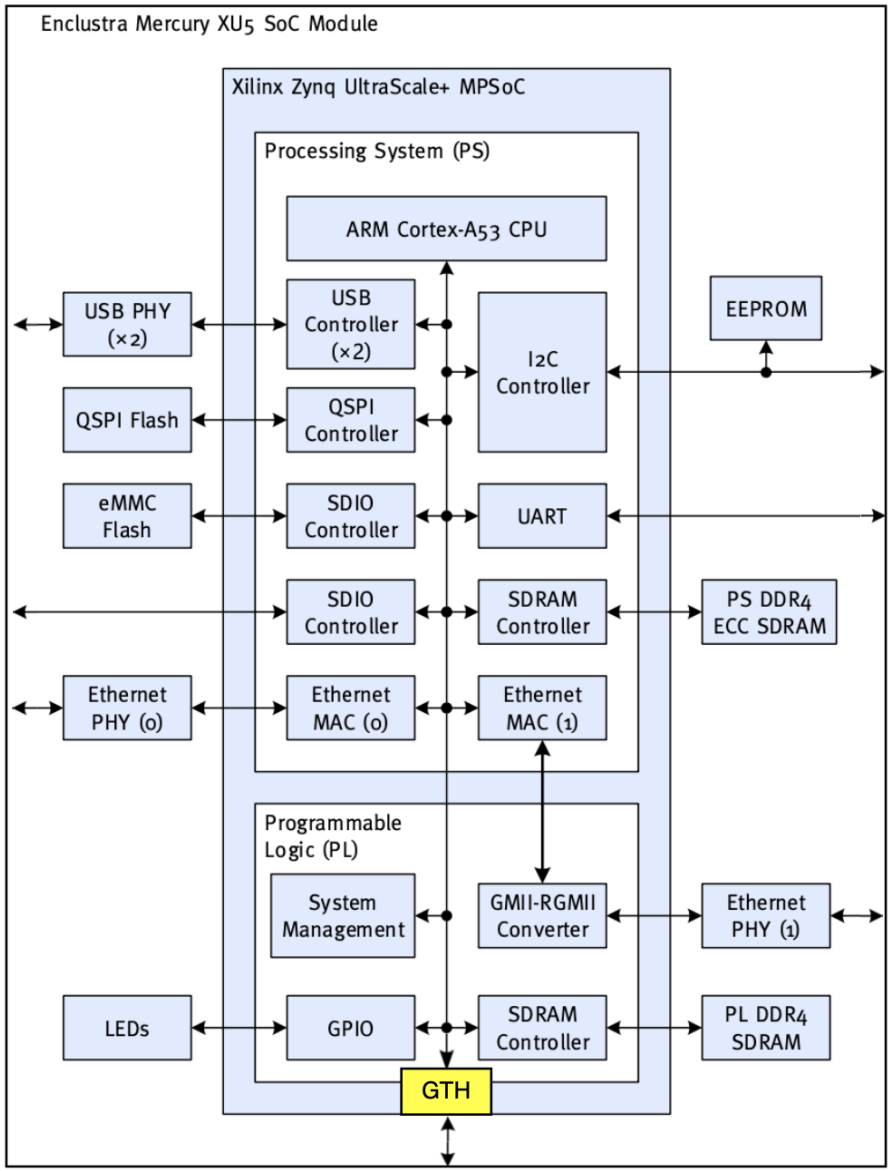
Software Components

- Firmware for the big FPGA (Virtex Ultrascale+)



Software Components

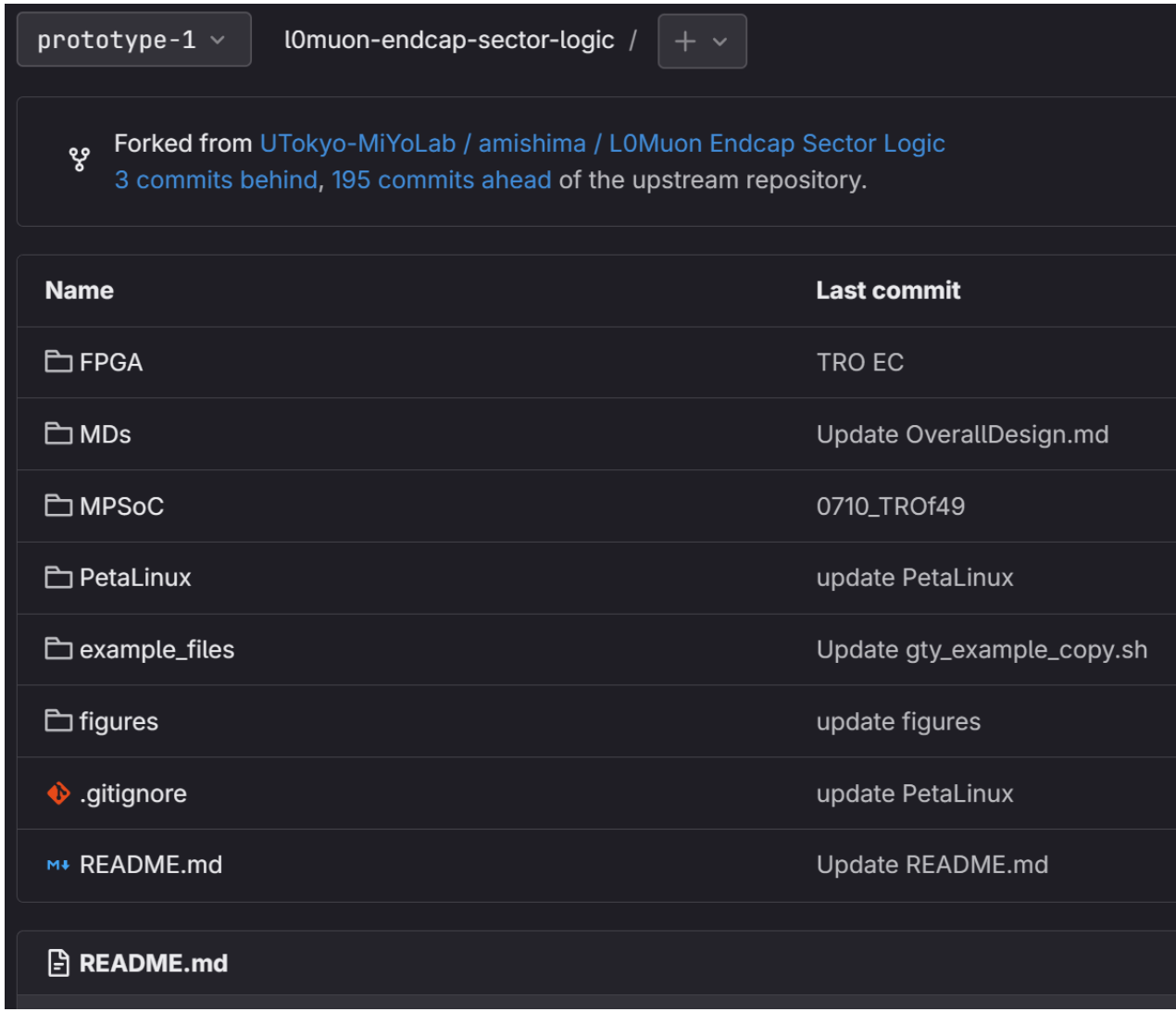
- Firmware for the MPSoC (Zynq)
- Embedded Linux (PetaLinux)
  - Run on ARM CPU on MPSoC



SL Firmware on GitLab

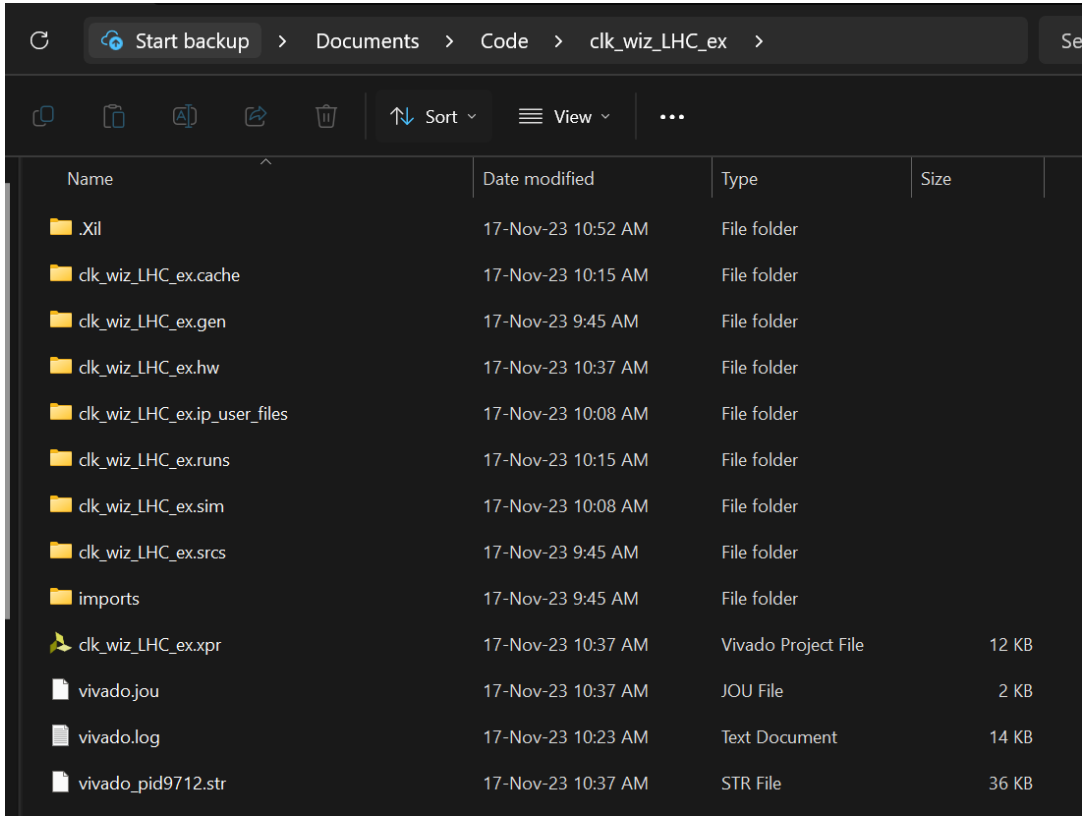
SL Firmware

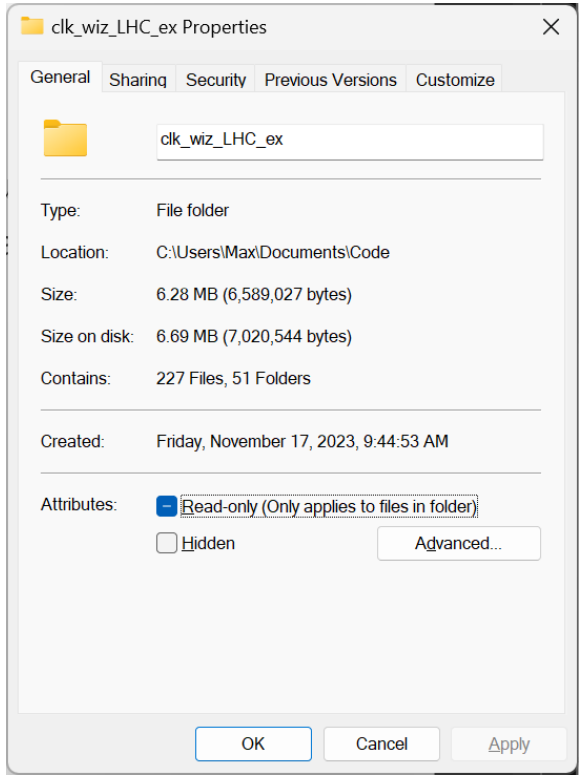
- Originally, one project with a directory per component
- Components are:
  - Firmware for Virtex Ultrascale+ (FPGA)
  - Firmware for the MPSoC
  - Embedded Linux (PetaLinux) for the MPSoC
  - Documentations in MDs directory



Vivado Project on Git

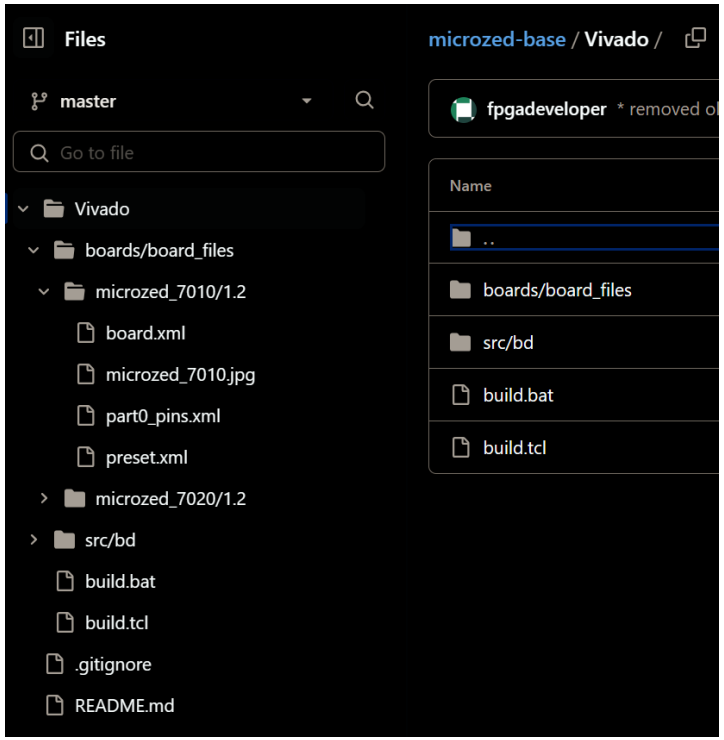
- Putting Vivado Project on Git is not simple
  - Many temporary files are scattered in many directories
  - Many proprietary IP Core only provided as binary
    - Git was designed to track changes in plain text files

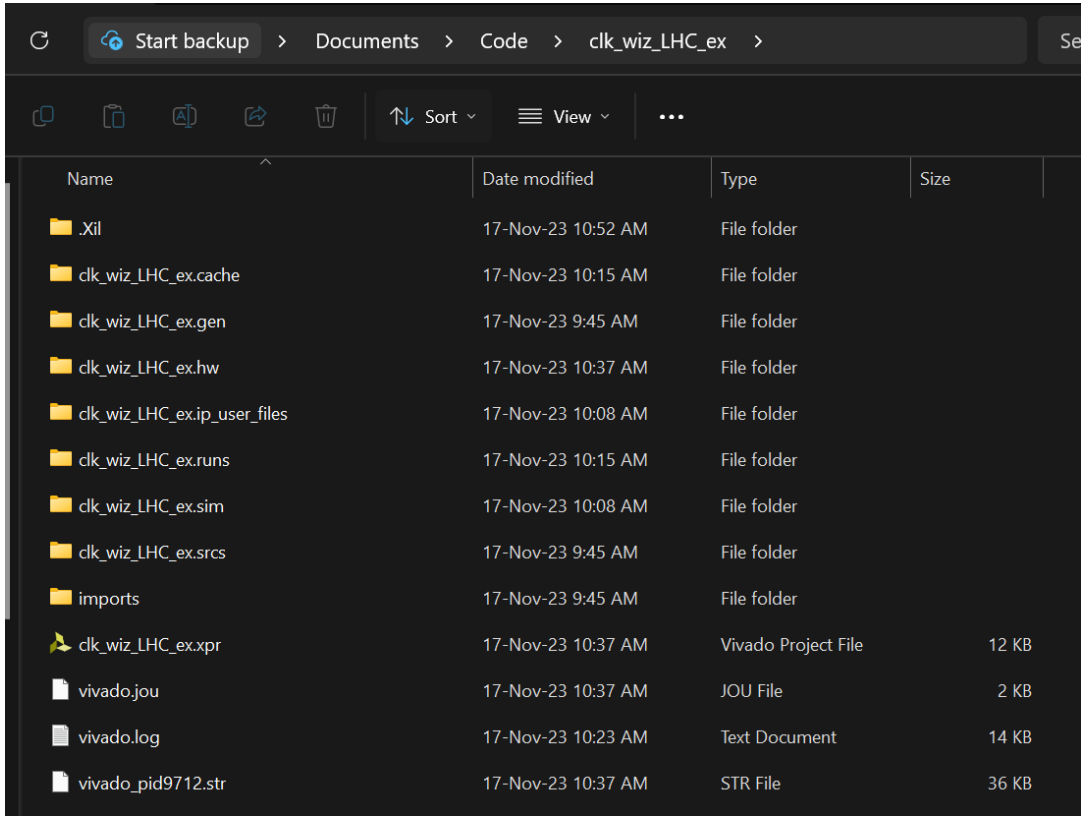




Git-friendly vs. User-friendly

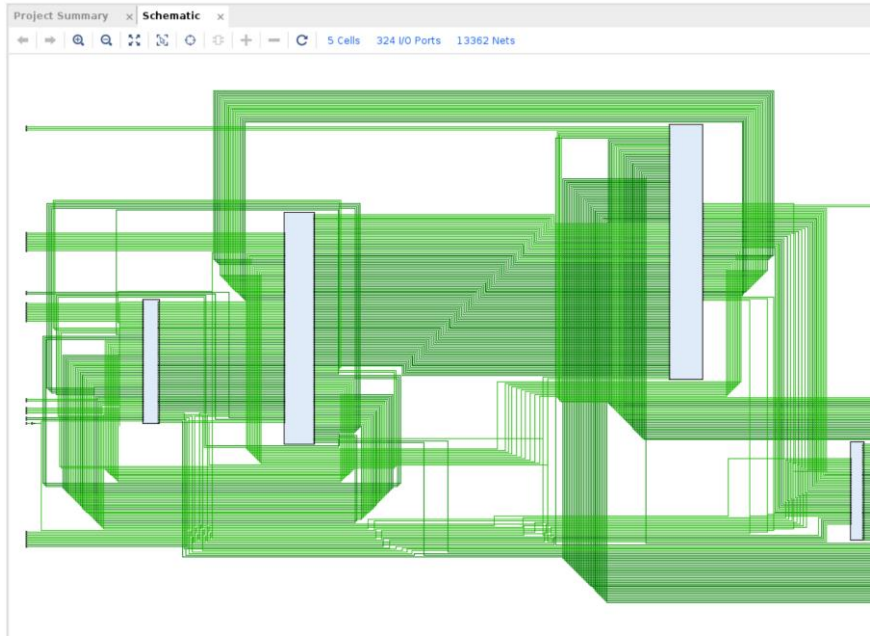
- Git-friendly: Use project-building script
  - ➕ Minimum number of tracked files
  - ➕ Clean commit history
  - ➖ User need to pick what to push
- User-friendly: Track everything
  - ➕ Minimum interventions from users
  - ➖ Unnecessary changes in commits
  - ➖ Very large repository

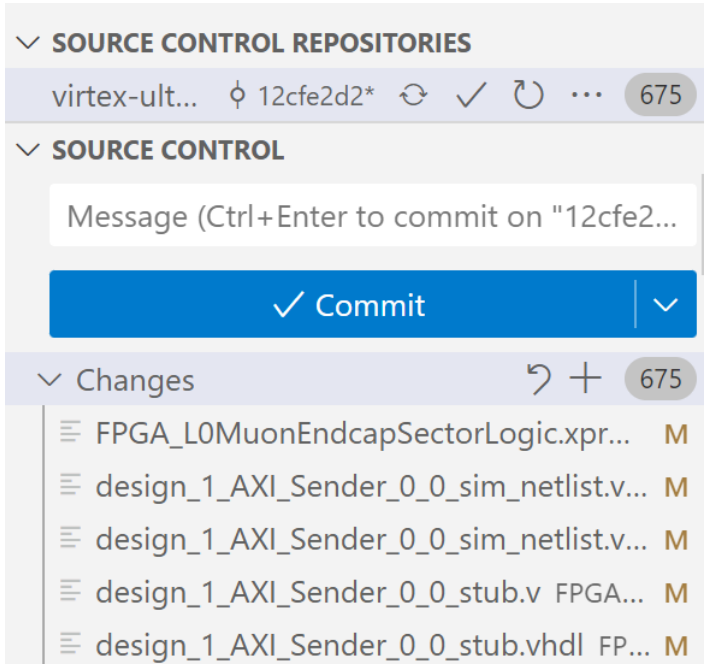




Git-friendly vs. User-friendly

- Git-friendly: Many tools such as Hog (HDL on Git) can help
  - Opinion: Still complicate
- User-friendly: Use .gitignore to minimize tracking unneeded files
  - Still messy
  - For SL firmware:
    - Open schematic produces 675 “changes”
    - 5000+ for generating bitstream

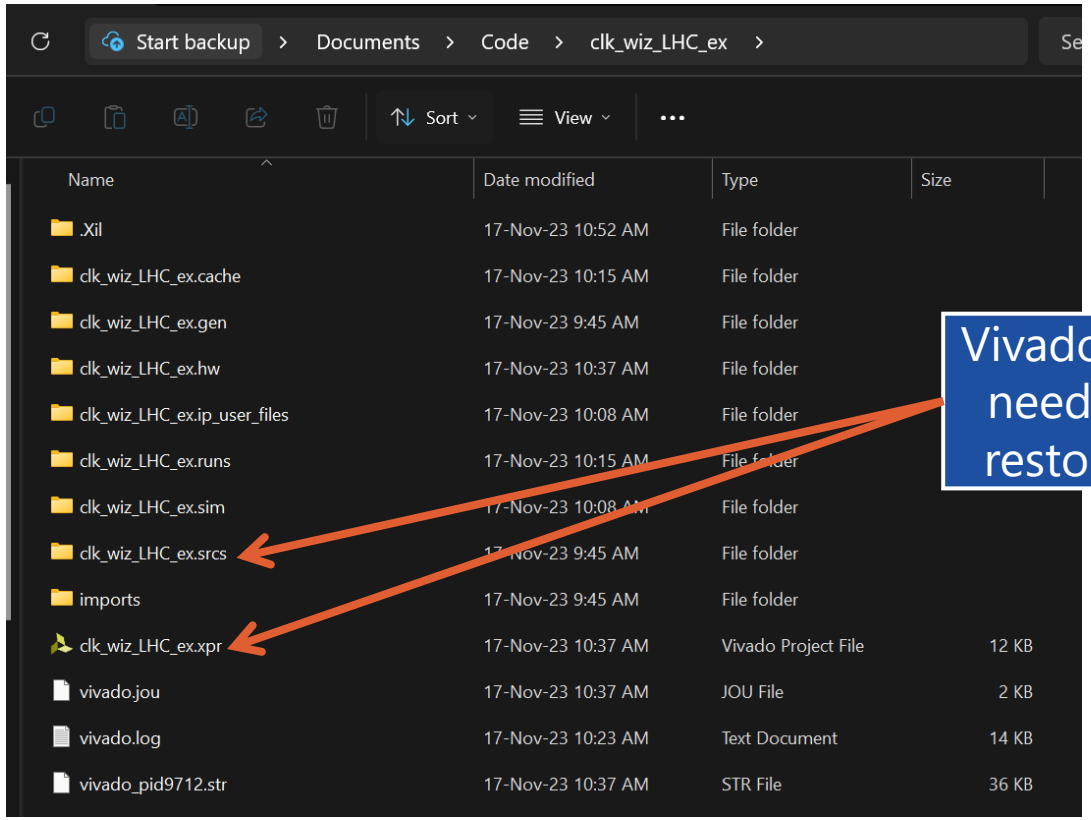




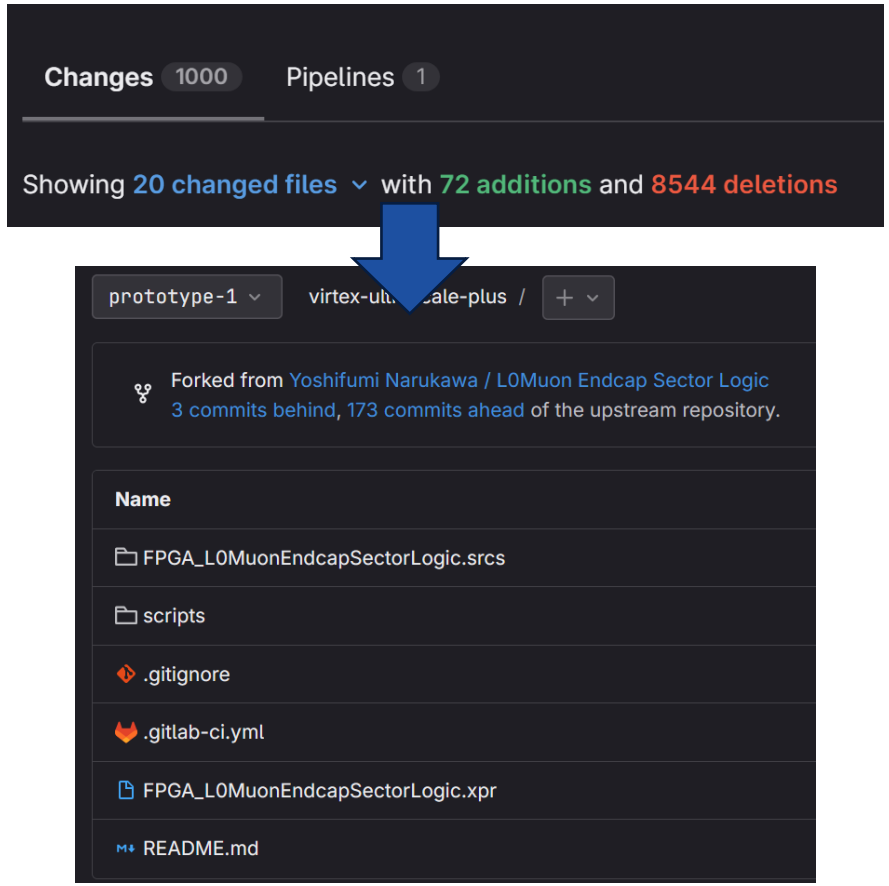
Good News for v2020.2

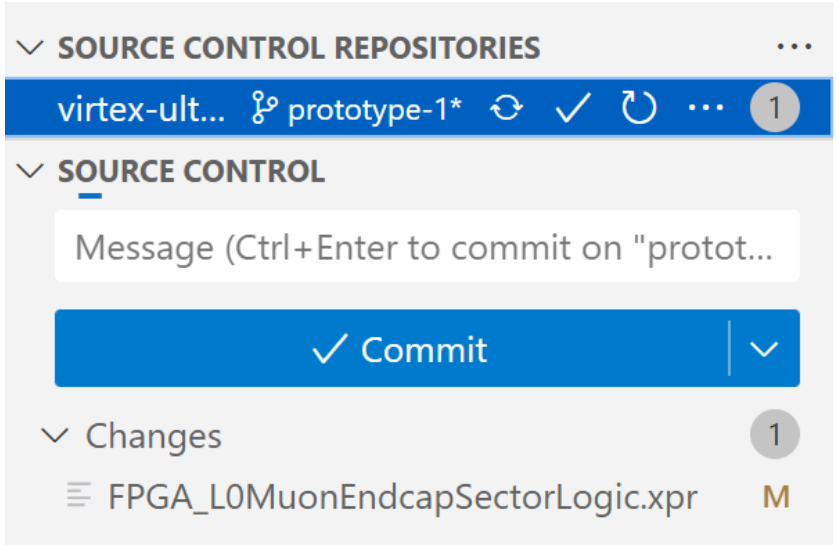
Revision Control Philosophy from Vivado 2020.2 Onwards [Link]

- In 2020.2, significant improvements are made to the Vivado project directory structure to improve the ability to interact with the revision control systems. [...] The project can be re-created by restoring the project.srcs directory and project.xpr file.



New Directory Structure



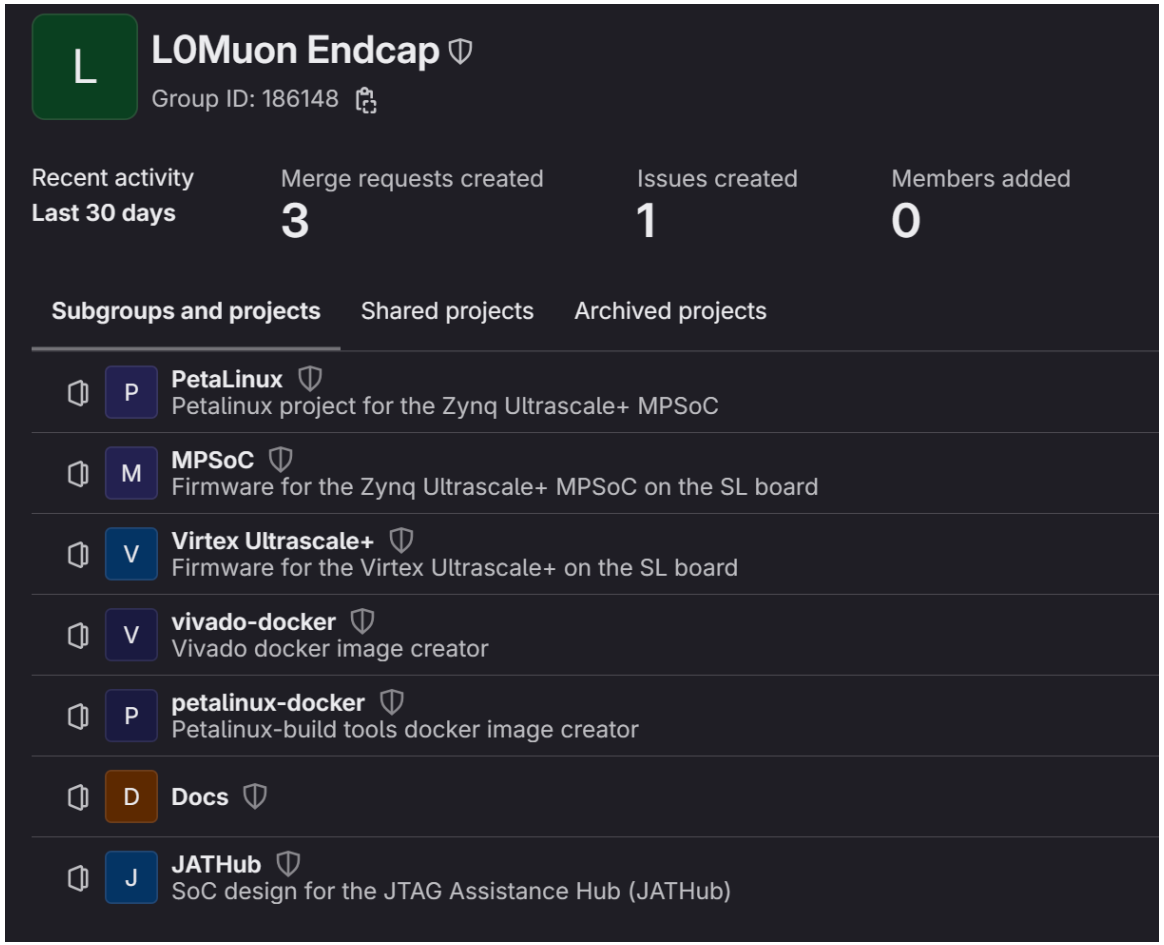


No more thousands of changes after simply opening the schematic

- Easy to put on GitLab
- User work in Vivado normally. Commit all tracked files and directories
- Not as clean as Hog Package, but hopefully much simpler for users

New Project Structure for SL

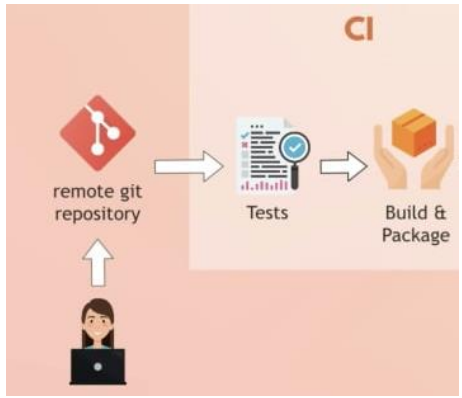
- Separate each component to be its own project in GitLab
  - All under the same “group”

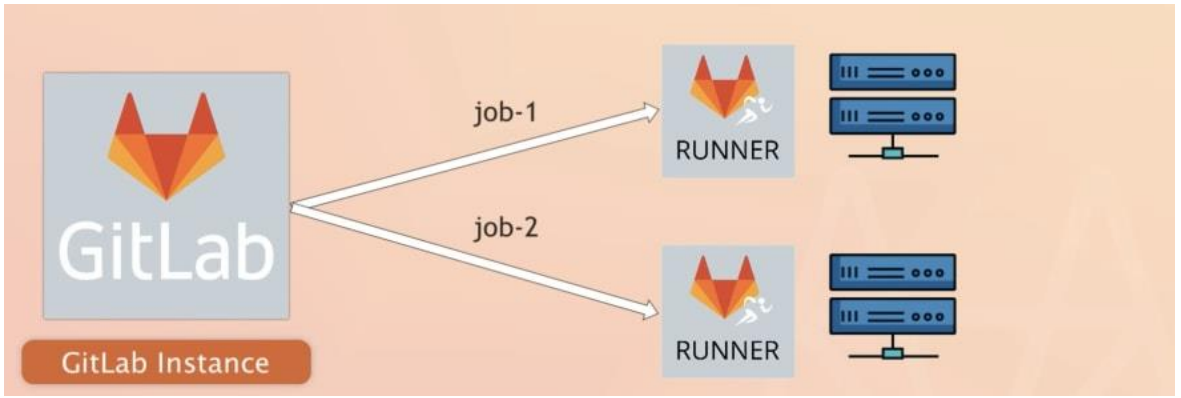


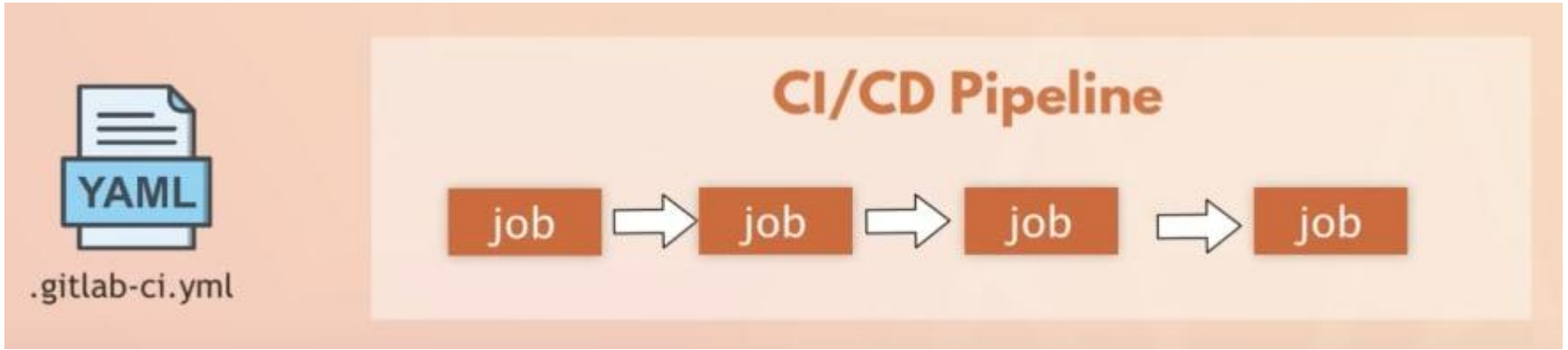
GitLab-CI for SL Firmware

GitLab CI/CD

- Automate check (simulation) and build (synthesize, implement, generate bitstream)
- Triggered when new commits are pushed and run on machines with gitlab-runner installed
- A YAML file (.gitlab-ci.yml) tell GitLab what to do (defines jobs)
  - A lot easier with each firmware on its own project

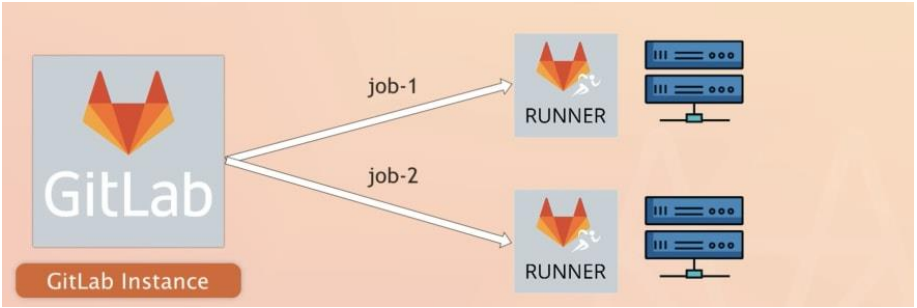


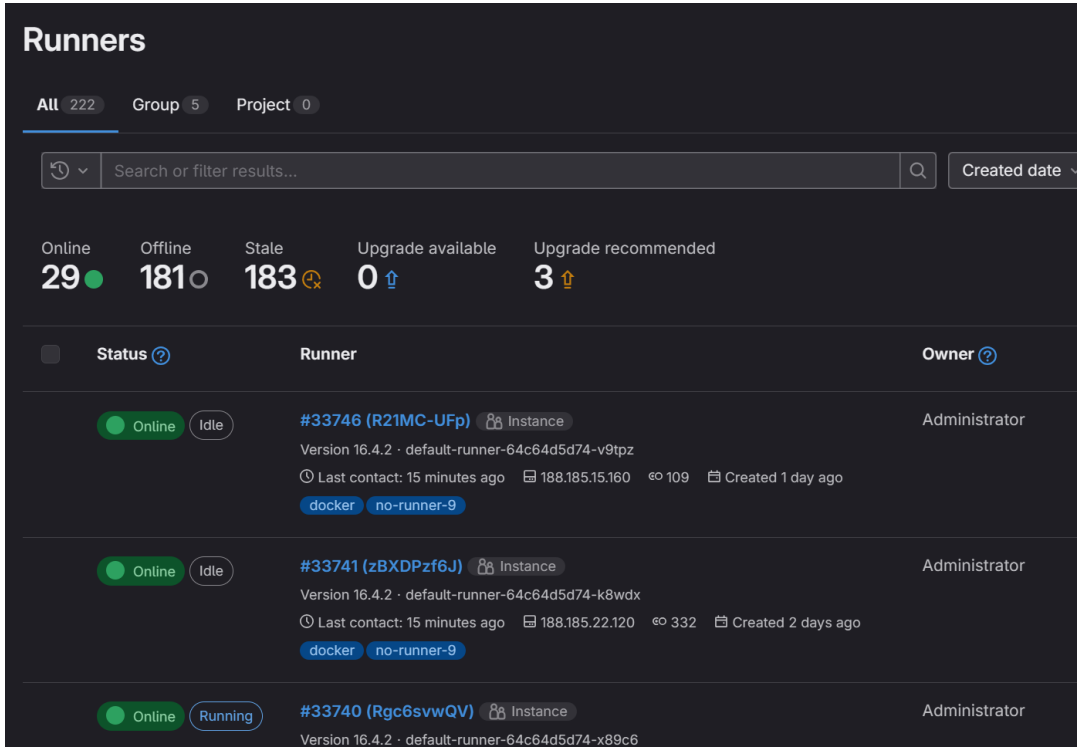




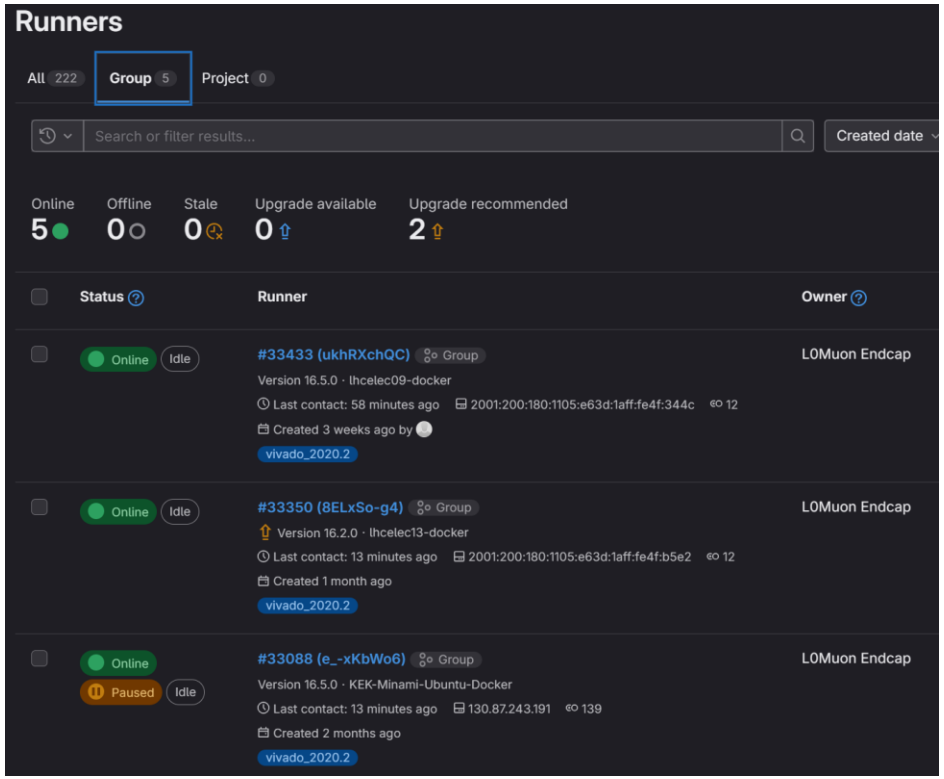
GitLab Runner

- Software that run the jobs
- Compatible with KEK and UTokyo firewall





Shared Runners from CERN

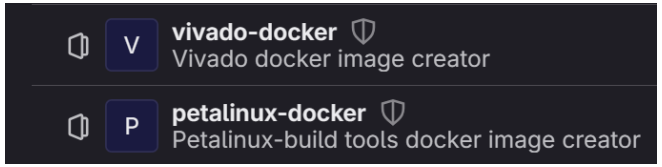


Private Runners at KEK and UTokyo

GitLab CI/CD for Beginners [FREE Course] - DEV Community

Executors and Docker

- Many of “Executors” for GitLab Runner. We use:
  - Shell: Like running in a terminal
  - Docker: Run in a Docker container
- Prefer Docker executor for future compatibility
  - Experience: Upgrade Ubuntu 20.04 LTS to 22.04 LTS break petalinux build
  - Scripts to build them are also on GitLab



	vivado-docker image	petalinux-docker image
Software	• Ubuntu 18.04 • Vivado 2020.2+dependencies	• Ubuntu 18.04 • Petalinux 2020.2+dependencies
Size	~50 GB	~11 GB
License	Need	No need
Target Runner	Private machines (KEK/UTokyo)	CERN Share runners

.gitlab-ci.yml

- Example from FPGA firmware (simplified)

```yaml
image:
  name: gitlab-registry.cern.ch/l0muon-endcap/vivado-docker:2020.2

stages: # List of stages for jobs, and their order of execution
  - check_syntax
  - sim
  - synth
  - impl
  - gen_bitstream

check_syntax-job:
…
…
gen_bitstream-job:
  stage: gen_bitstream
  script:
    - echo "Running gen_bitstream"
    - vivado -mode batch -source scripts/gen_bitstream.tcl -tclargs ${PROJECT_NAME}.xpr
  artifacts:
    paths:
      - ${PROJECT_NAME}.runs/impl_1/
    exclude:
      - ${PROJECT_NAME}.runs/impl_1/*.dcp
```

.gitlab-ci.yml

- "image" sector specify the docker image to use

```yaml
image:
  name: gitlab-registry.cern.ch/l0muon-endcap/vivado-docker:2020.2
```

```
Running with gitlab-runner 16.2.0 (782e15da)
2 on utokyo docker 8ELxSo-g, system_id: s_45657d31d342
3Resolving secrets00:00
5Preparing the "docker" executor59:41
6Using Docker executor with image gitlab-registry.cern.ch/l0muon-endcap/vivado-docker:2020.2 ...
7Using helper image: registry.gitlab.com/gitlab-org/gitlab-runner/gitlab-runner-helper:x86_64-782e15da
8Pulling docker image registry.gitlab.com/gitlab-org/gitlab-runner/gitlab-runner-helper:x86_64-782e15da ...
9Using docker image sha256:e25833d65f8929de622e5f7587e9fd7975524c0cefc573f1b605049b18b2c97a for
registry.gitlab.com/gitlab-org/gitlab-runner/gitlab-runner-helper:x86_64-782e15da with digest registry.gitlab.com/gitlab-
org/gitlab-runner/gitlab-runner-
helper@sha256:a5fba688be690f75d7c0e5c6b668a6a8f50b041e83981f275d55b0ef1f0ec727 ...
10Authenticating with credentials from job payload (GitLab Registry)
11Pulling docker image gitlab-registry.cern.ch/l0muon-endcap/vivado-docker:2020.2 ...
12Using docker image sha256:513177a800eb544d4075e1db893011ac33769b7650b35c9c421adab76583521c for gitlab-
registry.cern.ch/l0muon-endcap/vivado-docker:2020.2 with digest gitlab-registry.cern.ch/l0muon-endcap/vivado-
docker@sha256:80aab9092561035fcfcd4b1b5bcd9a2aa2081f82087c3cb4bda312d16cba577c ...
14Preparing environment00:19
15Running on runner-8elxso-g-project-165910-concurrent-0 via lhcelec13...
```

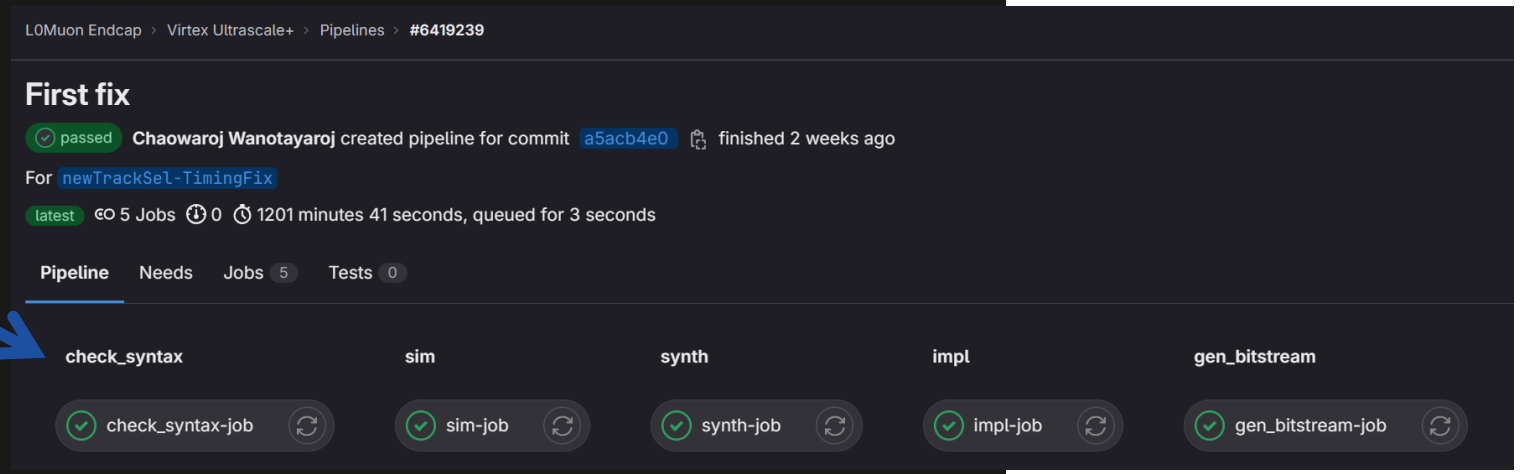
.gitlab-ci.yml

- "stage" section list the jobs in order

```yaml
image:
  name: gitlab-registry.cern.ch/l0muon-endcap/vivado-docker:2020.2

stages: # List of stages for jobs, and their order of execution
  - check_syntax
  - sim
  - synth
  - impl
  - gen_bitstream

check_syntax-job:
…
…
gen_bitstream-job:
  stage: gen_bitstream
  script:
    - echo "Running gen_bitstream"
    - vivado -mode batch -source scripts/gen_bitstream.tcl -tclargs ${PROJECT_NAME}.xpr
  artifacts:
    paths:
      - ${PROJECT_NAME}.runs/impl_1/
    exclude:
      - ${PROJECT_NAME}.runs/impl_1/*.dcp
```



.gitlab-ci.yml

- The “script” section list commands to run

```yaml
check_syntax-job:
  stage: check_syntax
  script:
    - echo "Checking syntax"
    - vivado -mode batch -source scripts/check_syntax.tcl -tclargs ${PROJECT_NAME}.xpr
```

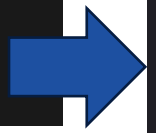
```
14 Getting source from Git repository 00:14
15 Fetching changes with git depth set to 20...
16 Reinitialized existing Git repository in /builds/l0muon-endcap/virtex-ultrascale-plus/.git/
17 Checking out a5acb4e0 as detached HEAD (ref is newTrackSel-TimingFix)...
...
27 Executing "step_script" stage of the job script 06:52
28 Using docker image sha256:513177a800eb544d4075e1db893011ac33769b7650b35c9c421adab76583521c for gitlab-registry.cern.ch/l0muon-endcap/vivado-docker:2020.2 with digest gitlab-registry.cern.ch/l0muon-endcap/vivado-docker@sha256:80aab9092561035fcfcd4b1b5bcd9a2aa2081f82087c3cb4bda312d16cba577c ...
29 $ source /tools/Xilinx/Vivado/2020.2/settings64.sh
30 $ source $LICENSE
31 $ echo "Checking syntax"
32 Checking syntax
33 $ vivado -mode batch -source scripts/check_syntax.tcl -tclargs ${PROJECT_NAME}.xpr
```

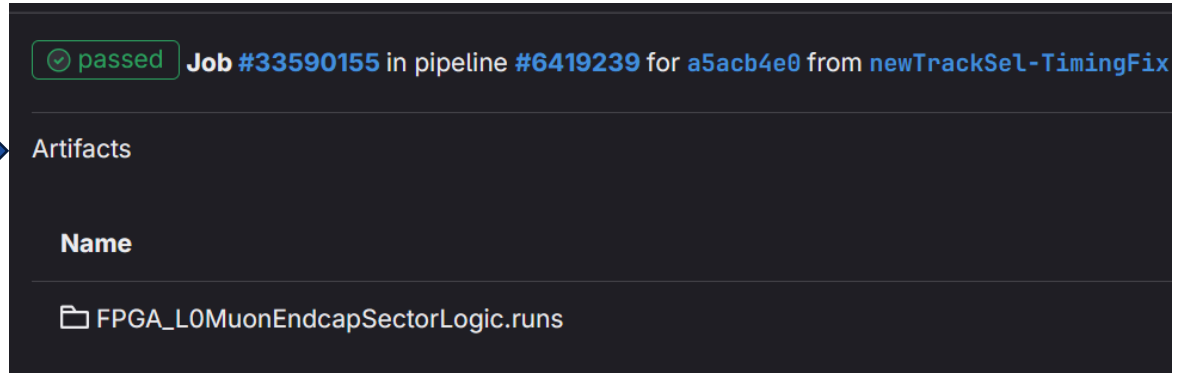
.gitlab-ci.yml

- The “artifacts” section tells what to keep (or not)

```yaml
artifacts:
    paths:
      - ${PROJECT_NAME}.runs/impl_1/
    exclude:
      - ${PROJECT_NAME}.runs/impl_1/*.dcp
```

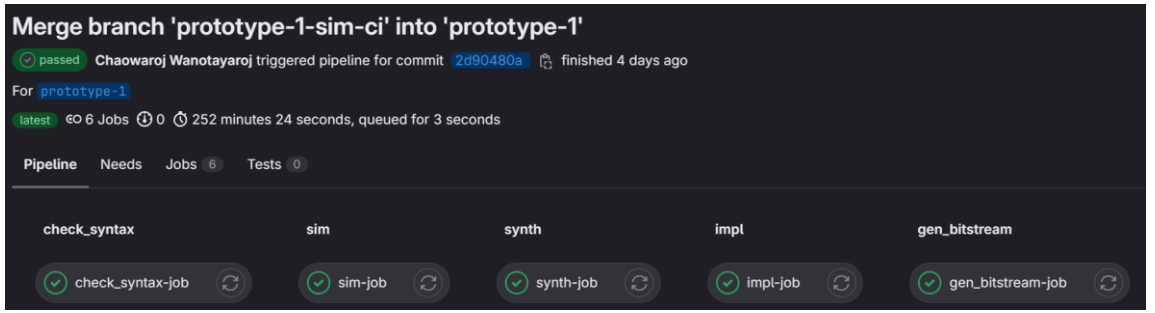
```
Uploading artifacts for successful job 00:16
Uploading artifacts...
FPGA_L0MuonEndcapSectorLogic.runs/impl_1/: found 65 matching artifact files and directories
FPGA_L0MuonEndcapSectorLogic.runs/impl_1/*.dcp: excluded 4 files
Uploading artifacts as "archive" to coordinator... 201 Created id=33590155 responseStatus=201 Created token=64_y5VsM
Cleaning up project directory and file based variables 00:01
Job succeeded
```





Current CI Pipeline Status

- Pipeline
  - ☑ Virtex Ultrascale+
  - ☑ MPSoC
  - ☑ Petalinux
- Vivado & Petalinux docker image are ready to be used
  - gitlab-registry.cern.ch/l0muon-endcap/vivado-docker
    - Need license for your device
  - gitlab-registry.cern.ch/l0muon-endcap/petalinux-docker
- Associate documentations: static website auto-created by GitLab-CI (see backup)



Summary

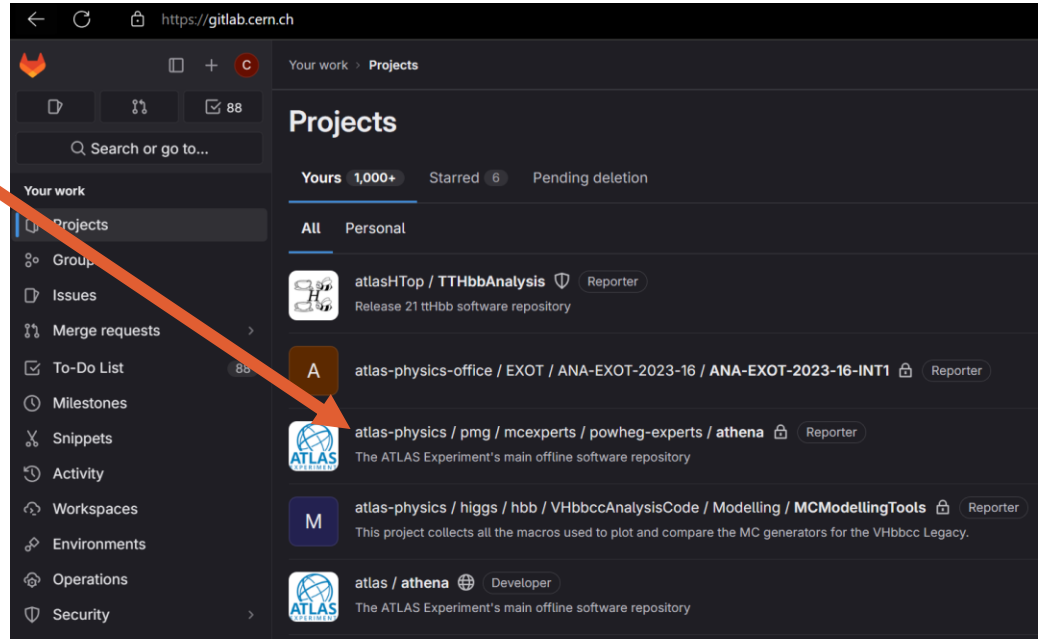
- Git is an essential tool for collaborative software development
- GitLab-CI can automate building and testing
- FPGA design workflow using Vivado is not very git-friendly, but in v2020.2 a significant change has been made to address the issue
- We restructure the GitLab projects to achieve:
  - Large reduction of the number of tracking files
  - A very simple Vivado workflow for users
- Build pipelines are tested and working well
- Compile the build environment (OS, libraries, tools, etc.) into docker images for future compatibility

Backup

Introduction - Git vs. GitHub

- A service such as GitHub or GitLab let you host a Git repository
- And offer other features on top of it
  - Bug tracking
  - Automate building (compiling, etc.)
- CERN host their own GitLab instance
  - Can't do that with GitHub
- GitLab projects can be organized in a group
  - E.g. <Group1>/<Subgroup1>/<Project Name>

GitLab



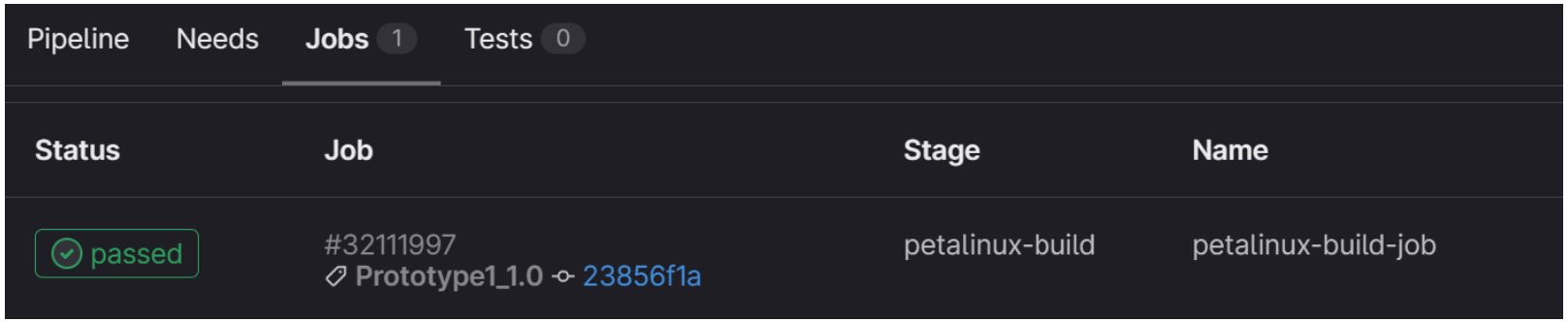
GitLab CI/CD

- From GitLab page on CI/CD:
  - CI/CD falls under DevOps (the joining of development and operations teams) and combines the practices of continuous integration and continuous delivery. CI/CD automates much or all of the manual human intervention traditionally needed to get new code from a commit into production, encompassing the build, test (including integration tests, unit tests, and regression tests), and deploy phases, as well as infrastructure provisioning. With a CI/CD pipeline, development teams can make changes to code that are then automatically tested and pushed out for delivery and deployment.
- CERN IT offers many DevOps solutions such as Jenkins which is more advance than GitLab CI/CD
- We only needs an automate build and test systems
  - GitLab CI/CD can do that and nicely integrated into GitLab web interface
- The actual build and test jobs are done on machine with gitlab-runner installed
  - CERN project a shared pool for all project on CERN GitLab
  - gitlab-runner software can be installed and register as private runner for a project or a group
- GitLab looks for .gitlab-ci.yml file in the repository for CI directive

PetaLinux Builder Tools

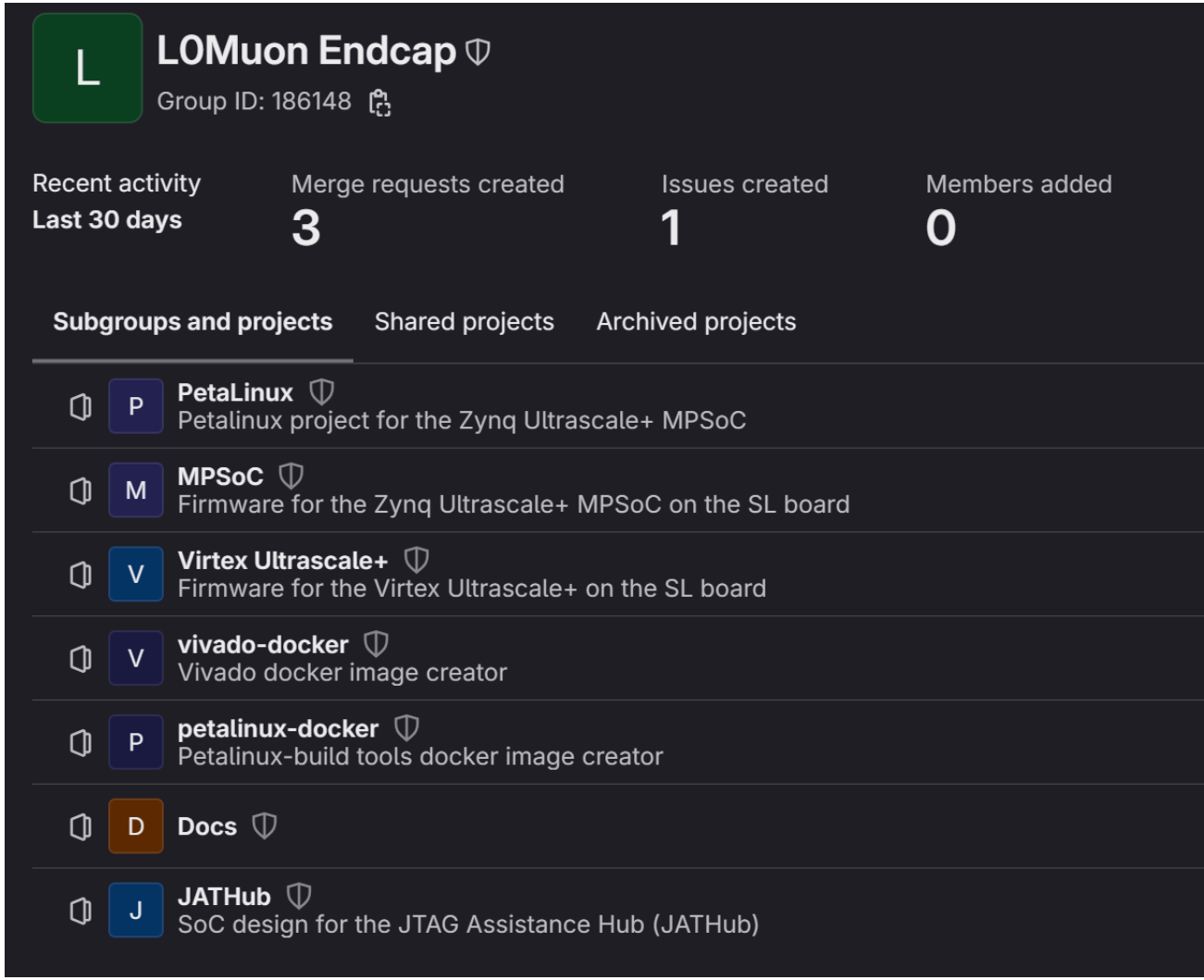
- As a bonus for having a docker image for building PetaLinux, it can run on Shared Runners pool from CERN
  - Compatible with the new Kubernetes clusters

```
[0KRunning with gitlab-runner 15.10.1 (dcfb4b66)[0;m
[0K  on runners-k8s-default-runners-575cd88bc4-rk99r PVQ-GR3u, system ID: r_jWkAVo4T5Rss[0;m
[0K  feature flags: FF_KUBERNETES_HONOR_ENTRYPOINT:true, FF_USE_ADVANCED_POD_SPEC_CONFIGURATION:true[0;m
section_start:1693440766:resolve_secrets
[0K[0K[36;1mResolving secrets[0;m[0;m
section_end:1693440766:resolve_secrets
[0Ksection_start:1693440766:prepare_executor
[0K[0K[36;1mPreparing the "kubernetes" executor[0;m[0;m
[0KUsing Kubernetes namespace: gitlab[0;m
[0KUsing Kubernetes executor with image gitlab-registry.cern.ch/l0muon-endcap/petalinux-builder:2020.2 ...[0;m
[0KUsing attach strategy to execute scripts...[0;m
```



New Project Structure

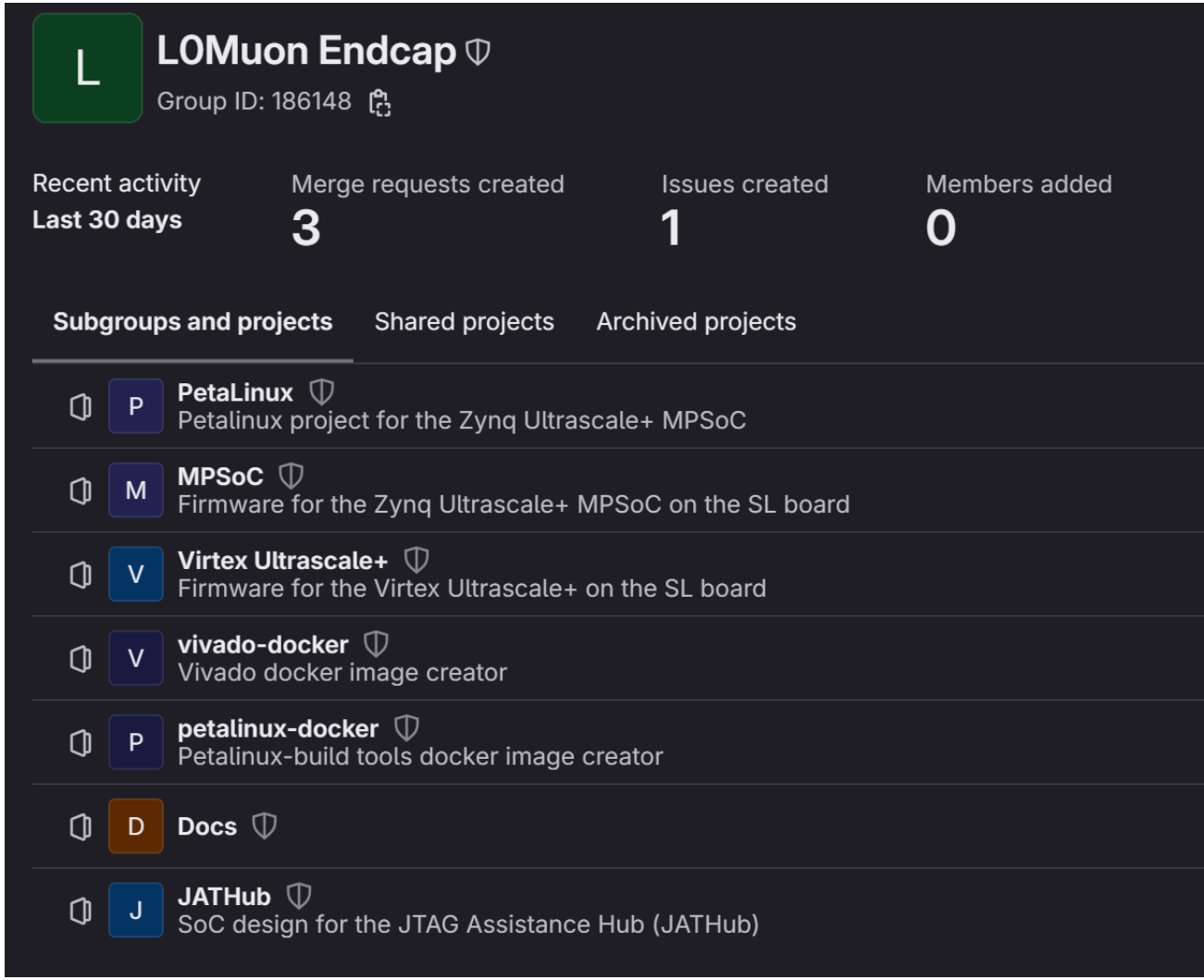
- Separate each firmware to be its own project in GitLab
  - All under the same “group”
- Allow individual CI config (.gitlab-ci.yml)
- Git Tagging to synchronize compatible releases
- Any project that is sitting under personal namespace can be forked over to group namespace without losing history
  - Easy migration of any project anywhere on CERN GitLab instance



New Project Structure

Current Projects:

- Docs
- Virtex Ultrascale+
  - Large FPGA on the SL board
- MPSoC
  - Zynq FPGA on the mezzanine board
- PetaLinux
  - MPSoC's Linux distro image builder
- vivado-docker
  - Docker image builder for Vivado
- petalinux-docker
  - Docker image builder for petalinux
- JATHub



GitLab-CI

- Setup automatic pipeline with GitLab-CI
  - Automate check & build (synthesize, implement, generate bitstream)
- GitLab-CI will trigger a pipeline which will be run using machines at KEK and UTokyo
  - To our surprise, neither KEK nor UTokyo firewall is an issue for GitLab Runner
- GitLab Runner can be installed as many type of “Executors”
  - We use Shell Executor and Docker Executor
    - Shell Executor: simple, like running manually through terminal
    - Docker Executor: clean, static environment
- CERN has a pool or shared runners we can use
  - Used to be Docker executor, recently begin migrating to Kubernetes
  - Limitations:
    - Recommend not going over 10GB for the image
    - Vivado licenses we have are fixed per hardware and cannot be easily apply to the shared runners

Running Pipelines

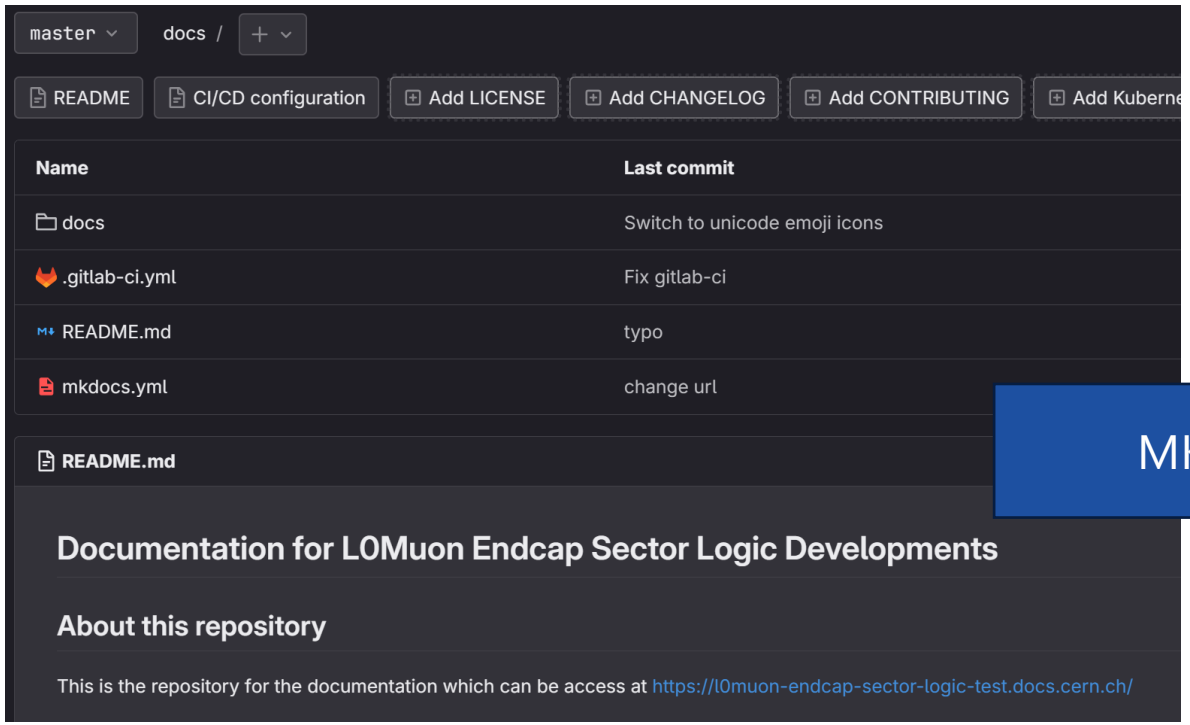
- Original idea:
  - Use shell executor. Keep it simple
  - Self-host runner (KEK and UTokyo)
- Given the timeline of Phase-II/Run4/5, it is important to ensure future compatibility
  - Already run into a problem upgrading from Ubuntu 20.04 LTS to 22.04 LTS. Petalinux 2020.2 can no longer run!?
- Switch to docker container
  - Clean and static environment for pipeline
  - Ensure future compatibility
  - Side benefits
    - Can run on CERN shared runner when license is not an issue (Petalinux). Tested on the new Kubernetes cluster
    - Provide a simple way to run Vivado on MacOS (only need Docker)

Running Pipelines

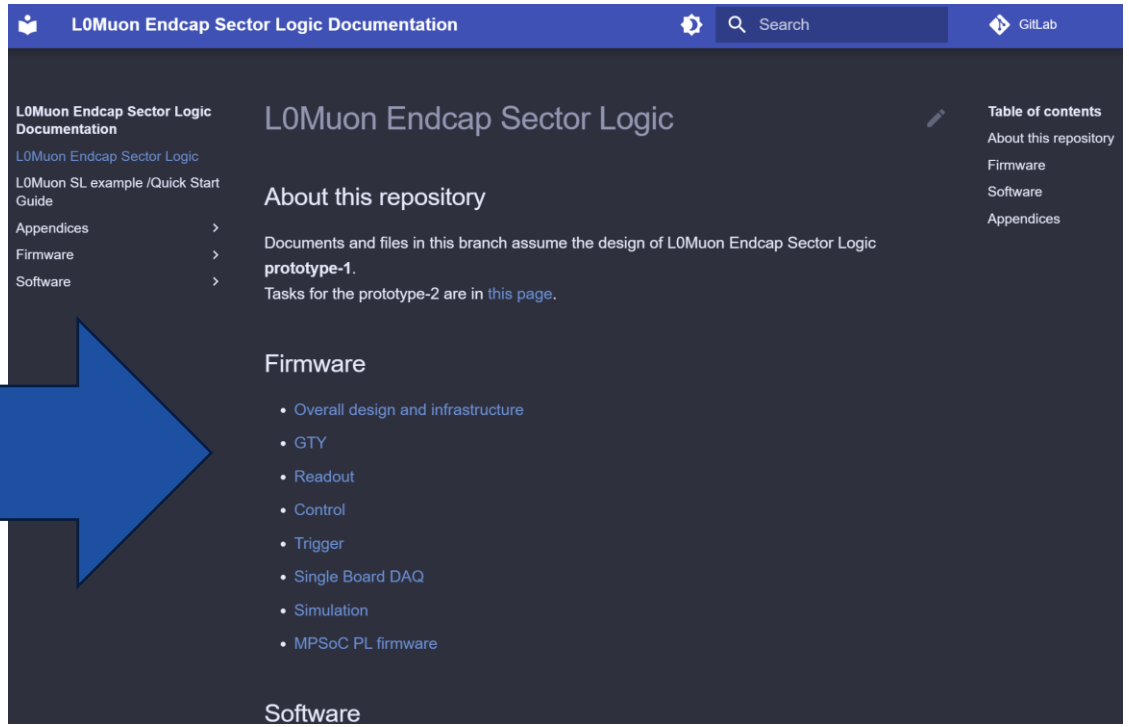
- Now both Petalinux and firmware (Virtex&Zynq) generations run on docker containers
- Petalinux: ~11GB image. No license needed. Can run on CERN share runners
- Vivado: ~50GB image. Need license. Most likely need dedicate machines
  - Due to the size, may want to change pull policy (always -> if-not-present)
  - Tested at both KEK and Utokyo
  - Runners available to all projects in l0muon group
- Dockerfiles and scripts to build the images are also on GitLab
  - But the image building itself was not intend to be used with CI
    - Can be done in theory with Kaniko, but downloading the installers are a pain and the images are not expected to be rebuilt many times

Documentations

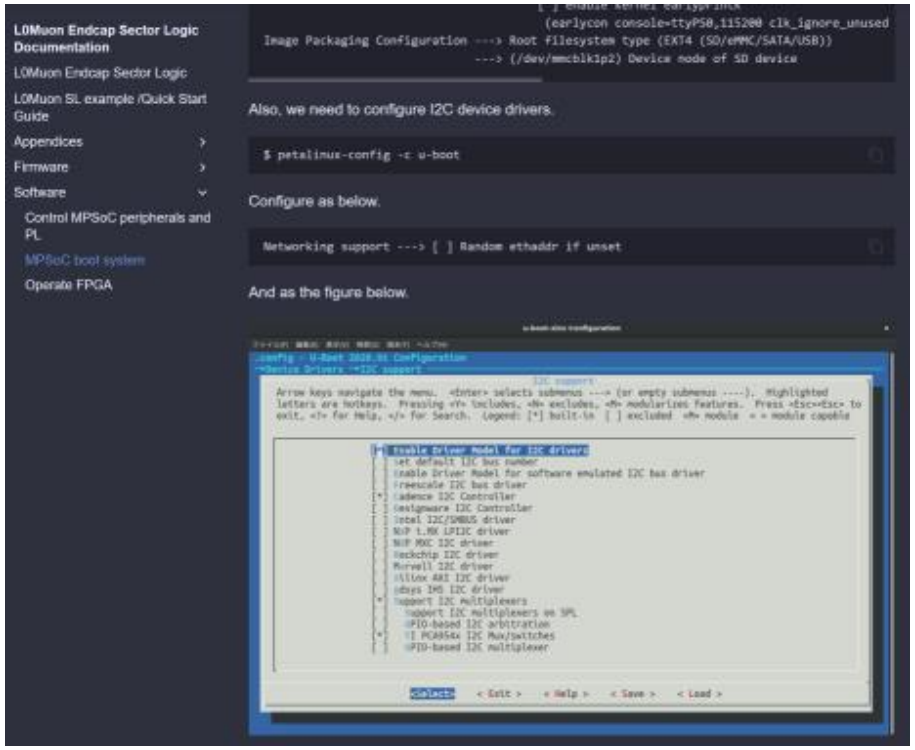
- Move all Markdown files (*.md) to its own project
  - See details: Create a MkDocs website (like this one) - ABP Computing @ CERN
- Static documentation webpages auto-generated using MKDocs
  - https://l0muon-endcap-sector-logic-test.docs.cern.ch/

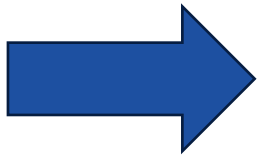


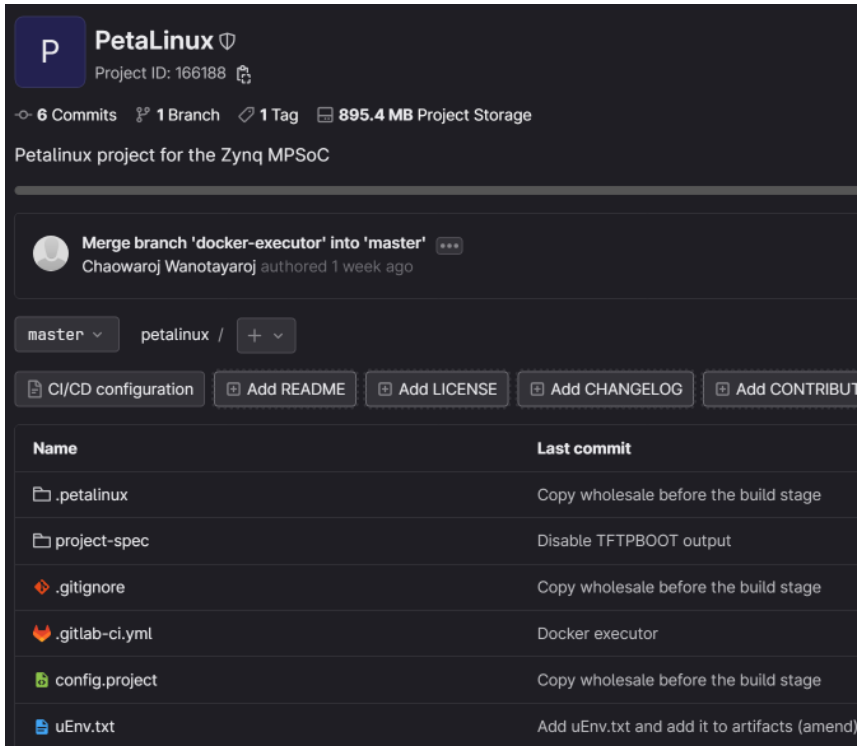
MKDocs



PetaLinux on GitLab







- Before: A list of instruction on how to build an image with a few files scattered around the big project and GUI that need manual input
- After: A project on GitLab one can clone and build
  - Can be automated with CI