**Guideline How to work**

**with Gitlab CI and withOUT Gitlab CI**

1. **With Gitlab CI (Run test app by GitLab runner)**

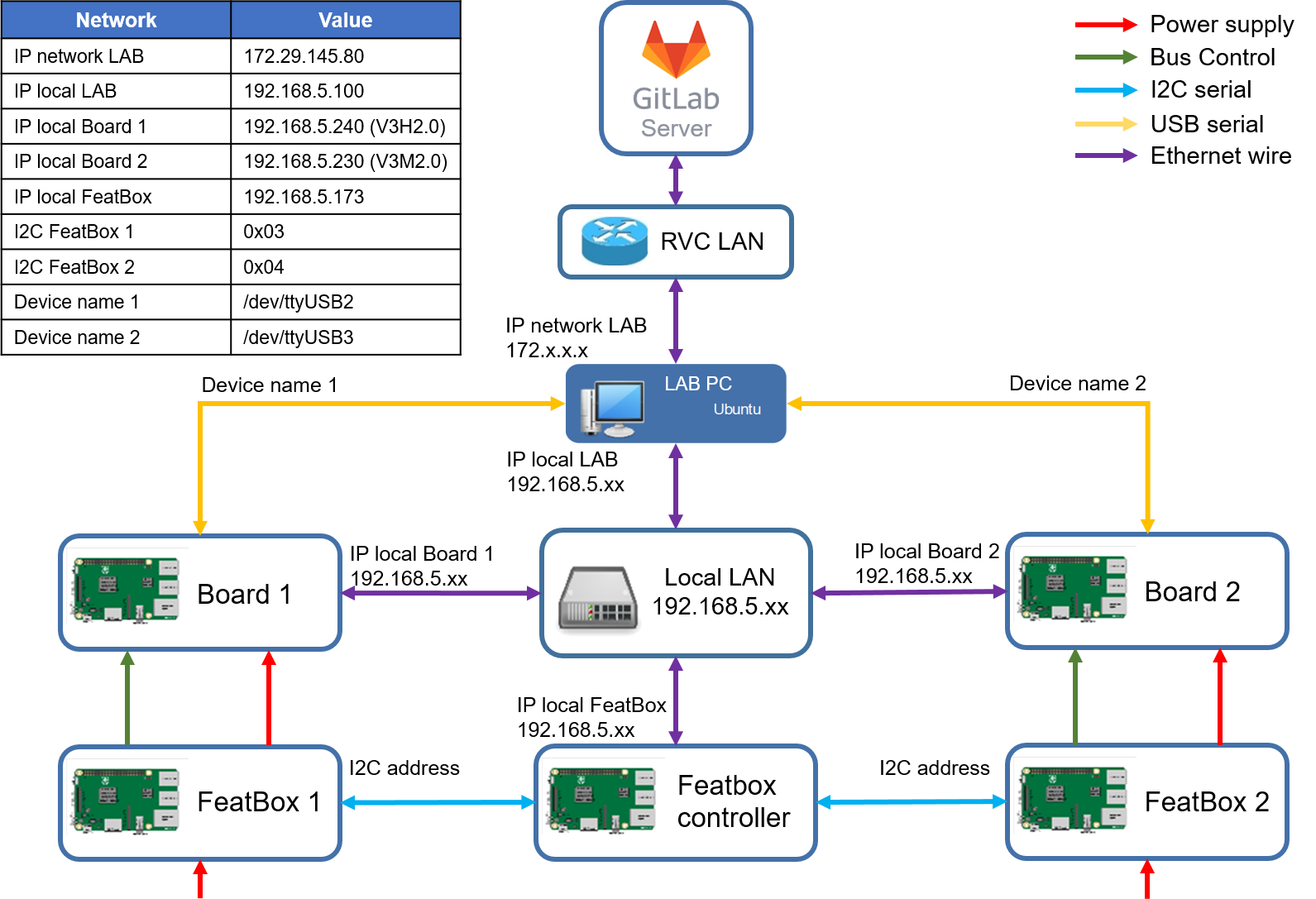
****

Figure 1.1. Connection diagram of gitlab CI

1. How to Git Lab CI work in ours project

We have 2 stage: build and test. After running stage build, binary file will be stored in Terminal PC (refer figure 2.1).

Stage test will be copy this file to LAB PC. Board will be mount to LAB PC where binary stored. Stage test will run binary file (test app) on board and return result in CI. Stage test can control to on/off/boot board through FeatBox

1. How to add new software component (new test app)

You can see the structure as below:

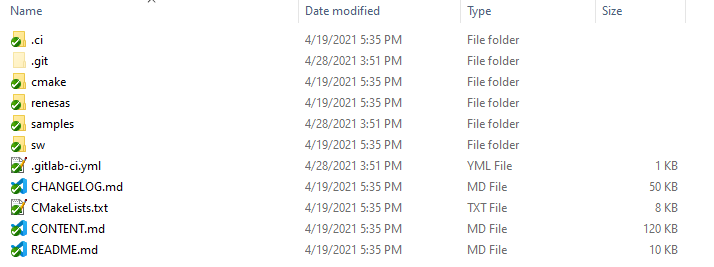


Figure 1.2.

* To add new software, we need to pay attention to 3 components in this folder
  + .ci : this folder contain ci file system
  + .gitlab-ci.yml : this file is very important, you will edit it for execute stage and include file in .ci folder
  + renesas/application/platform : This folder contain application. We will add our sample app in application folder
* First we go to the renesas/application/platform:

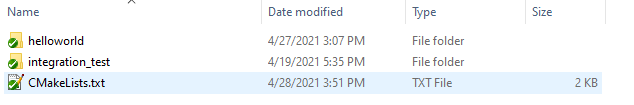


Figure 1.3.

* These are application for testing, now we will create our sample application. In this case is parkassist app.
* You will open renesas/application/CmakeLists.txt: Use rcar\_add\_subdirectory() function to add module the build list:

rcar\_add\_subdirectory(platform/helloworld)

rcar\_add\_subdirectory(platform/integration\_test)

rcar\_add\_subdirectory(platform/parkassist) # add this line to add subdirectory parkassist

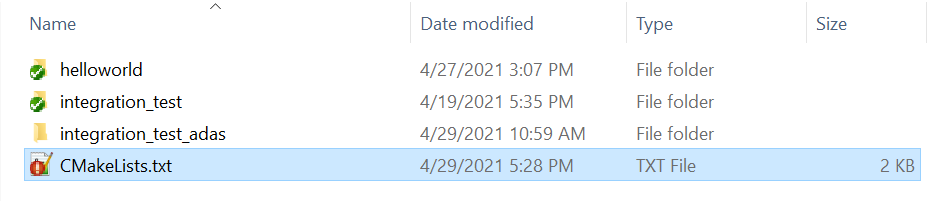


Figure 1.4.

* Then we write CMakeLists.txt file, follow How to write Cmake file for test application in R-Car Environment\_20201214.pdf. (page 198)

set(app\_name "helloworld")

set(app\_version "0.2.0")

cmake\_minimum\_required(VERSION 3.10.2)

project(${app\_name})

if(NOT RCAR\_SOC)

set(RCAR\_SOC

V3H1

)

endif()

find\_package(rcar-xos REQUIRED)

if(RCAR\_TARGET\_OS STREQUAL "linux" )

#Linux configuration

elseif(RCAR\_TARGET\_OS STREQUAL "qnx")

# QNX config will be placed here.

set(source

${CMAKE\_CURRENT\_SOURCE\_DIR}/application/src/target/qnx/main.c

)

set(header

)

set(include\_dir

# add any additional 3rd party include directories your applicationrequires

# Note: Include directories from xOS Software components are automatically added when you list them in link\_libs

)

set(link\_lib

# This sample does not need any other sw compoents

# Examples could be: osal, osal\_wrapper

)

# [optional] for 3rd Party libraries

set(link\_dir

# Additional Link directories for 3rd party libraries

)

#---------------------------------------------------------------------------------------------------------------------

# [optional] Any specifics that an application may require, like OS specifics, SoC specifics

#----------------------------------------------------------------------------------------------------------------------

# Linker dependencies for baremetal/freertos environments

if (CMAKE\_LINKER MATCHES "armlink")

set(linker\_script "${CMAKE\_CURRENT\_SOURCE\_DIR}/application/lscript.scat")

endif()

#----------------------------------------------------------------------------------------------------------------------

# Test settings, e.g. which and how to run the tests for this application

#----------------------------------------------------------------------------------------------------------------------

set(test\_args

# Set this variable of your test requires some command line arguments to be given.

)

set(test\_fail\_regex

# Set this variable if this test has other fail log than the default

# Default fail regex is: [^RESULT]<FAIL>)|ERROR|error:|Segmentation fault|result NG

)

set(test\_timeout

# Default timeout is 10m, change it if your test take more time (unit is second)

300

)

else()

# Other OS

endif()

#======================================================================================================================

# DO NOT CHANGE ANYTHING BELOW THIS LINE, except you know what to do ;)

#======================================================================================================================

rcar\_configure\_application()

* Focus on:
* **app\_name**: set your app name in here, not same with another app
* **source**: list all file source in here
* **header**: list all file header in here
* **include\_dir**: add your directory include path here
* **link\_lib**: add your lib use in this app here
* After that write rcar\_env\_support.cmake file, follow Write module.cmake file in R-Car Environment\_20201214.pdf (page 195).

set(SUPPORTED\_SOC V3H2)

set(SUPPORTED\_OS QNX)

set(SUPPORTED\_XIL any)

1. How to add one more item in stage build

Now we go to /rcar-env/.ci/yml, this folder includes all yml file system for CI. From here, we will edited file build.yml

|  |
| --- |
| Build QNX - V3H2 - IFS:  before\_script:  - cd $SDK\_DIR  - source $SDK\_DIR/qnxsdp-env.sh  - cd -  - git checkout $CI\_COMMIT\_SHA  - cd renesas/os/bsp/qnx/src/  - make clean  - make TARGET=v3h  - mkdir -p $TARGET\_BIN\_DIR1  - cp images/\*.bin $TARGET\_BIN\_DIR1  - echo "Please get V3U QNX binaries at $TARGET\_BIN\_DIR1"  - cd ../  extends: .build\_template  tags:  - v3x\_adas\_st\_builder |

|  |
| --- |
| Build QNX - V3H2 - sample\_app:  before\_script:  - cd $SDK\_DIR  - source $SDK\_DIR/qnxsdp-env.sh  - cd -  - git checkout $CI\_COMMIT\_SHA  - cd renesas/os/bsp/qnx/src/  - make clean -C src/app/camera\_ctrl  - make -C src/app/camera\_ctrl  - mkdir -p $TARGET\_BIN\_DIR1  - cp /data2/yocto/native\_build/gitlab-runner/builds/QrZPSCms/1/Gen3\_QNX/v3u\_qnx71\_bsp/renesas/os/bsp/qnx/src/src/app/camera\_ctrl/aarch64/o-le/camera\_ctrl $TARGET\_BIN\_DIR1  - echo "Please get V3U QNX binaries at $TARGET\_BIN\_DIR1"  - cd ../  extends: .build\_template  tags:  - v3x\_adas\_st\_builder |

|  |
| --- |
| Build QNX - V3U – integration\_app:  before\_script:  - cd $ SDK\_QNX\_DIR  - source $ SDK\_QNX\_DIR/qnxsdp-env.sh  - cd -  - git checkout $CI\_COMMIT\_SHA  - mkdir -p build  - cd build  - cmake -G "Unix Makefiles" -DCMAKE\_TOOLCHAIN\_FILE="../cmake/toolchain\_qnx\_7\_1\_0.cmake" -DCMAKE\_PREFIX\_PATH="../cmake"  -DRCAR\_SOC="${RCAR\_SOC}" -DCMAKE\_BUILD\_TYPE="${BUILD\_TYPE}" -DCMAKE\_C\_COMPILER\_FORCED=TRUE  -DCMAKE\_CXX\_COMPILER\_FORCED=TRUE -DRCAR\_COMPILER\_FLAGS\_CHECK=OFF ..  - cmake --build . --target frontview\_integration\_test\_v3h2  - mkdir -p $TARGET\_BIN\_DIR\_QNX  - cp bin/\* $TARGET\_BIN\_DIR\_QNX  - echo "Please get V3U QNX binaries at $TARGET\_BIN\_DIR\_QNX"  - cd ../  extends: .build\_template  tags:  - v3x\_adas\_st\_builder |

* As all you know, we only build 3 thing
* IFS (Image File System): it only use to build IFS to load when boot board to run, it not change anything
* sample\_app\_BSP: sample app of BSP (e.g: camera\_ctrl, discom\_test, ...), it not change anything
* integration\_app: this include sample app (helloworld\_sample\_app, impsample, imrlxsample) and own application (parkassist, frontview, drive\_record).
* So I give instruction as below to you:
* **tag**: -this is tag name of runner, it depend on your config runner, changed it if you create new runner for this stage.
* If you want to build all application in one job build, use “cmake –build .”
* If you want to add more app, add this line “cmake --build . --target <target\_name>”

1. How to add one more item in stage test

* In this folder, open notepad++ or Visual Studio code to create new yml file name test\_parkassist.yml

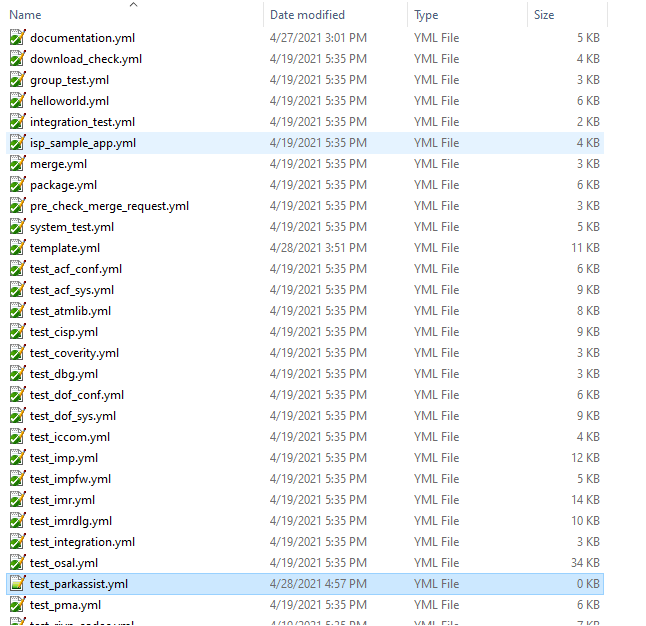


Figure 1.5.

* Write content test\_parkassist.yml:

|  |
| --- |
| * Integration Test - frontview - QNX - V3H2: * extends: .test\_template * variables: * TEST\_NAME: frontview\_integration\_test\_v3h2 * tags: * - v3x\_adas\_st\_runner * only: * refs: * - merge\_requests * - web * - tags * changes: # script will be executed if any of the files and subdirectories inside the below directory has update. * - renesas/applications/platform/integration\_test\_adas/\*\*/\*.\* * - .ci/yml/\*.yml * needs: [Build QNX - V3H2 – integration\_app] |

* variables TEST\_NAME: this is name of file binary you want execute it on board
* only/refs: this job only run when has merge\_requests, web (manual)
* needs: job test will download artifacts of job “Build QNX – V3H2 – integration\_app” to execute test.
* If you want to add more test, just copy job test above, change job name and variable TEST\_NAME, and directory path. Example:

|  |
| --- |
| Integration Test - helloworld - QNX - V3H2:  extends: .test\_template  variables:  TEST\_NAME: helloworld\_sample\_app\_v3h2  tags:  - v3x\_adas\_st\_runner  only:  refs:  - merge\_requests  - web  - tags  changes: # script will be executed if any of the files and subdirectories inside the below directory has update.  - samples/helloworld\_sample\_app/\*\*/\*.\*  - .ci/yml/\*.yml  needs: [Build QNX - V3H2 – integration\_app] |

* Now we going to add: include - local: .ci/yml/test\_parkassist.yml into .gitlab-ci.yml file
* Now for your new branch appear to remote repository, you have to push your branch for update content you just add.

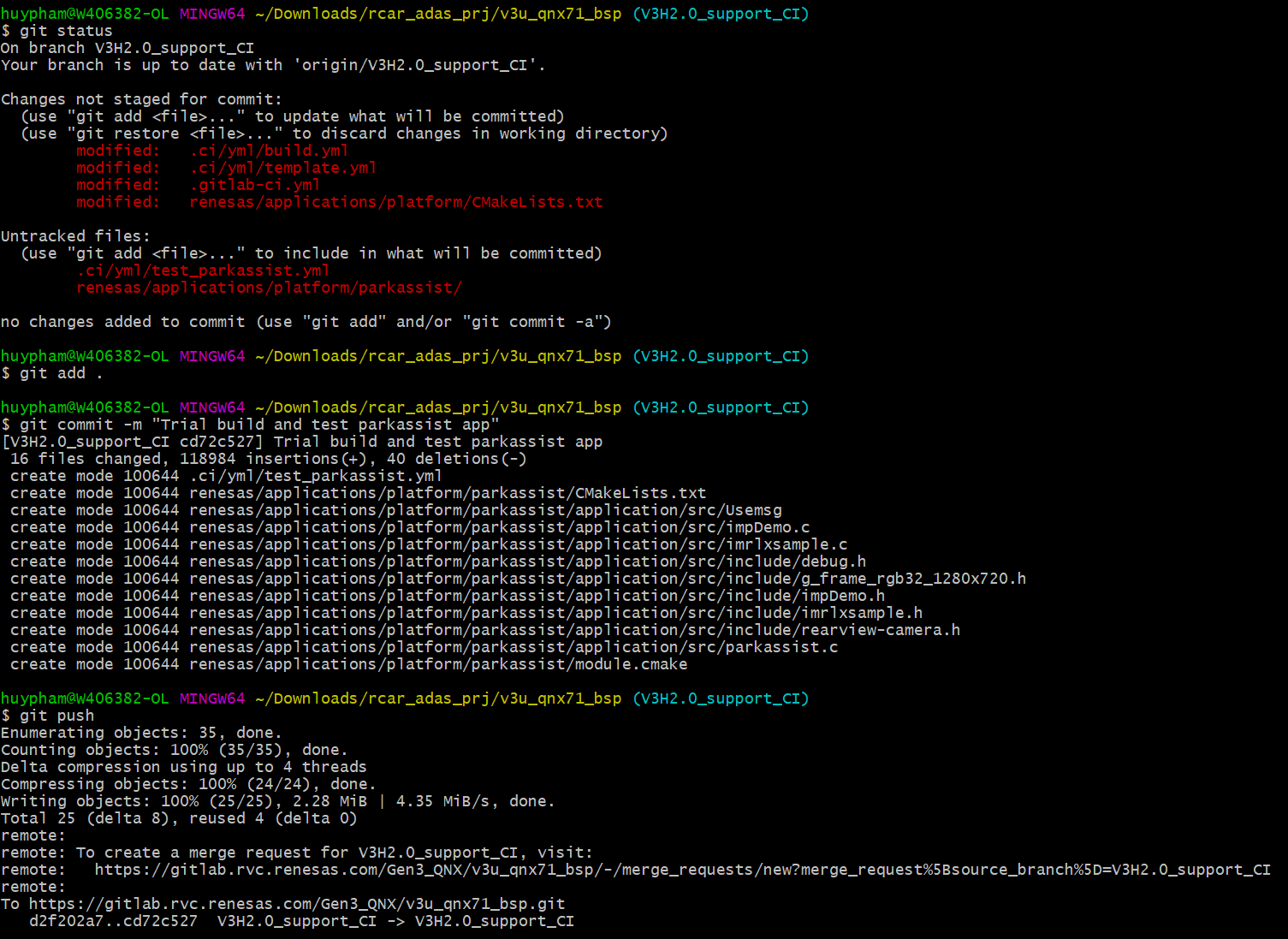


Figure 1.6.

1. How to run CI after pushing on GitLab

* After pushing successfully, we get a comment as below:

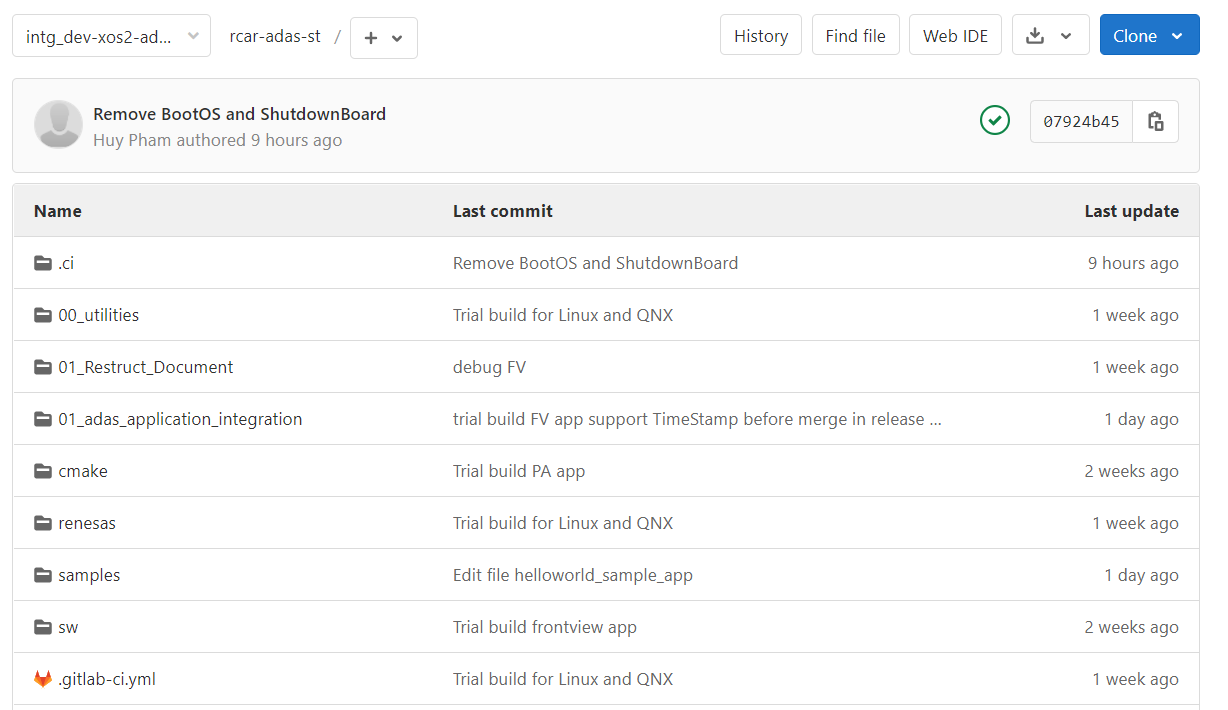


Figure 1.7.

* Click -> CI/CD -> Pipeline -> Run Pipeline :

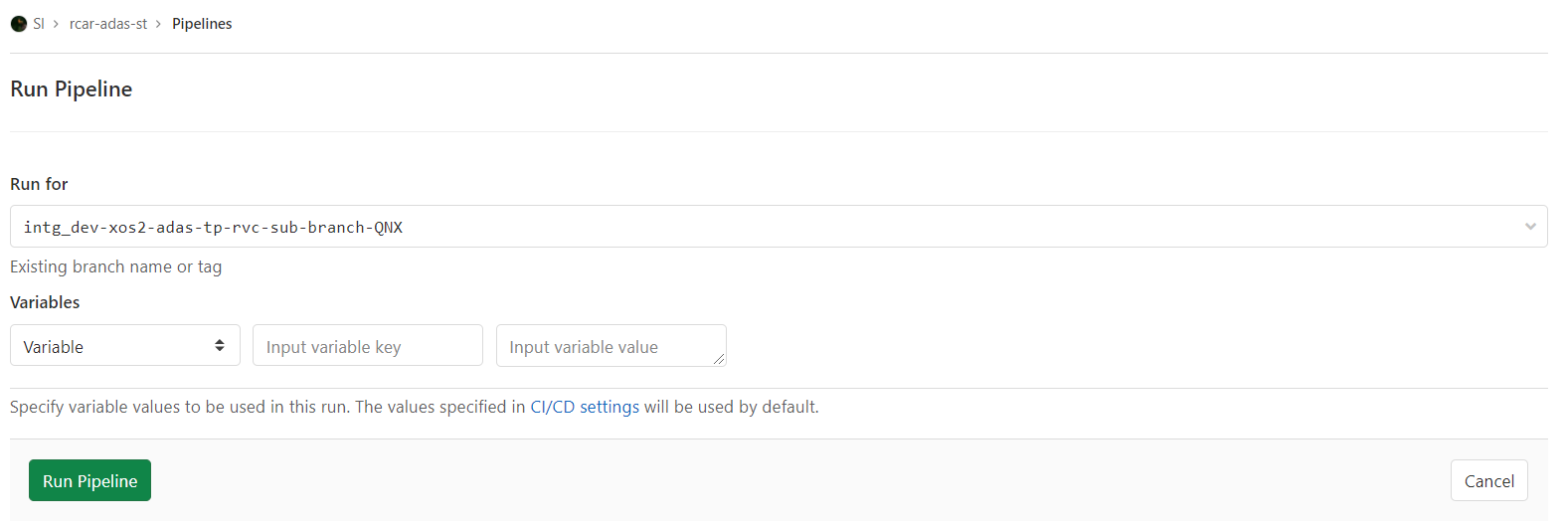


Figure 1.8.

* Search for our branch and click on Run Pipeline
* The GUI flow chart:

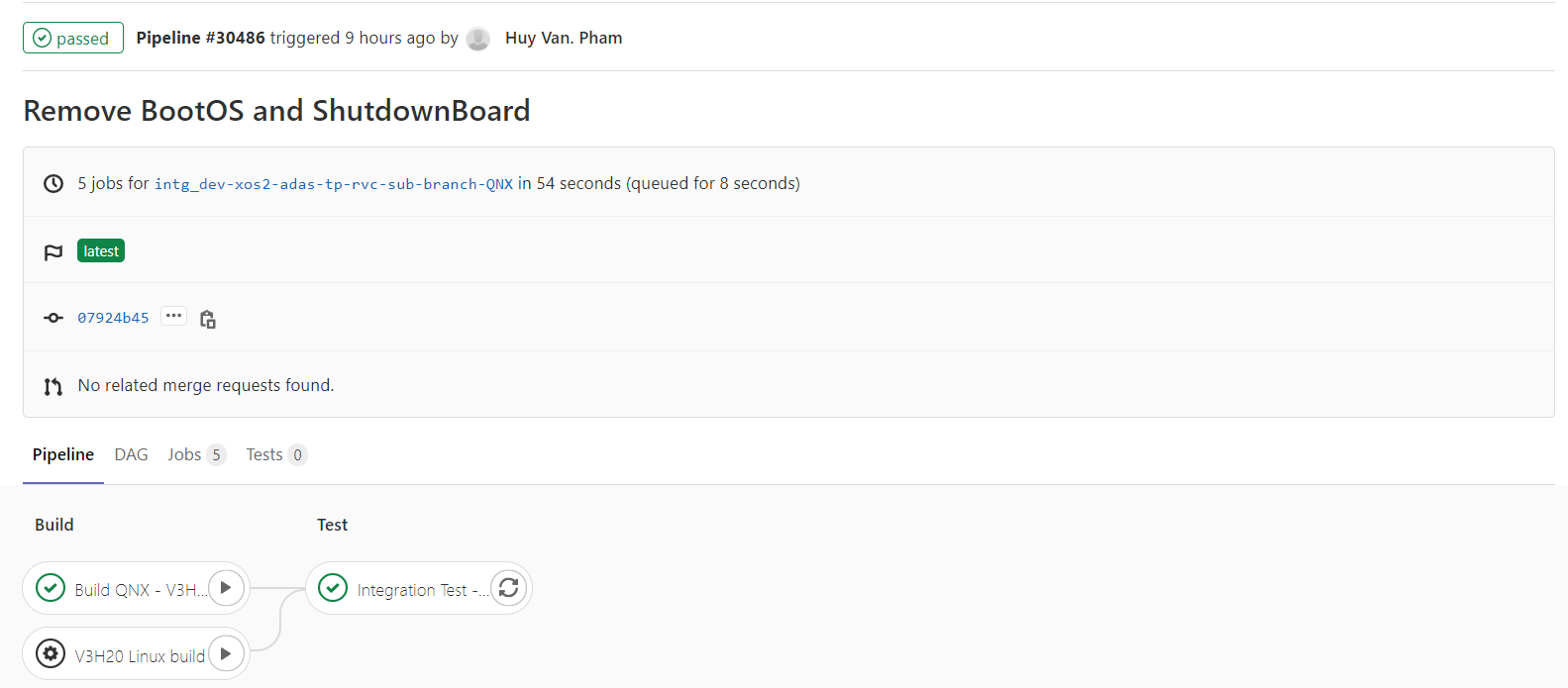


Figure 1.9.

1. Q&A internal review meeting

Question (Chung-san): Could you change setting CI minimalist diff with rcar-env for support merged ?

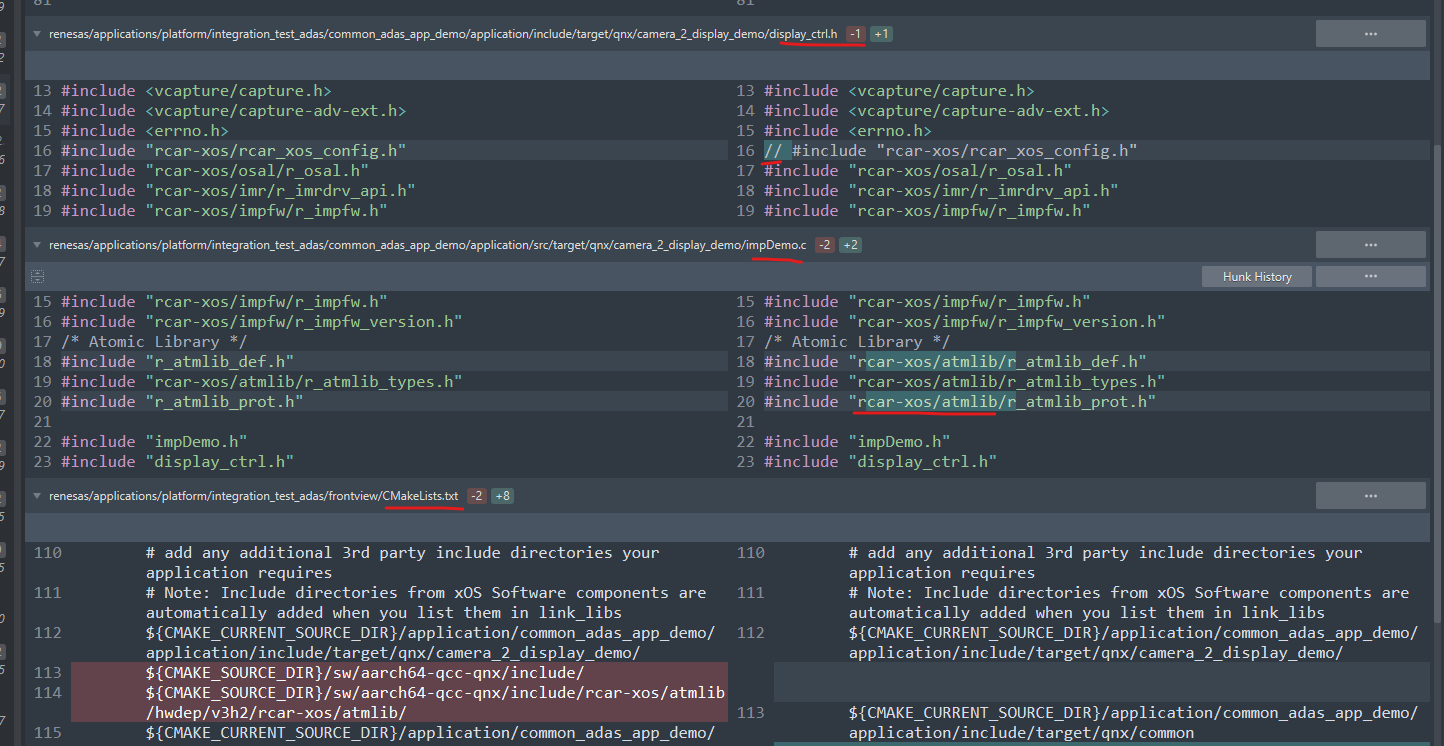


Figure 1.10.

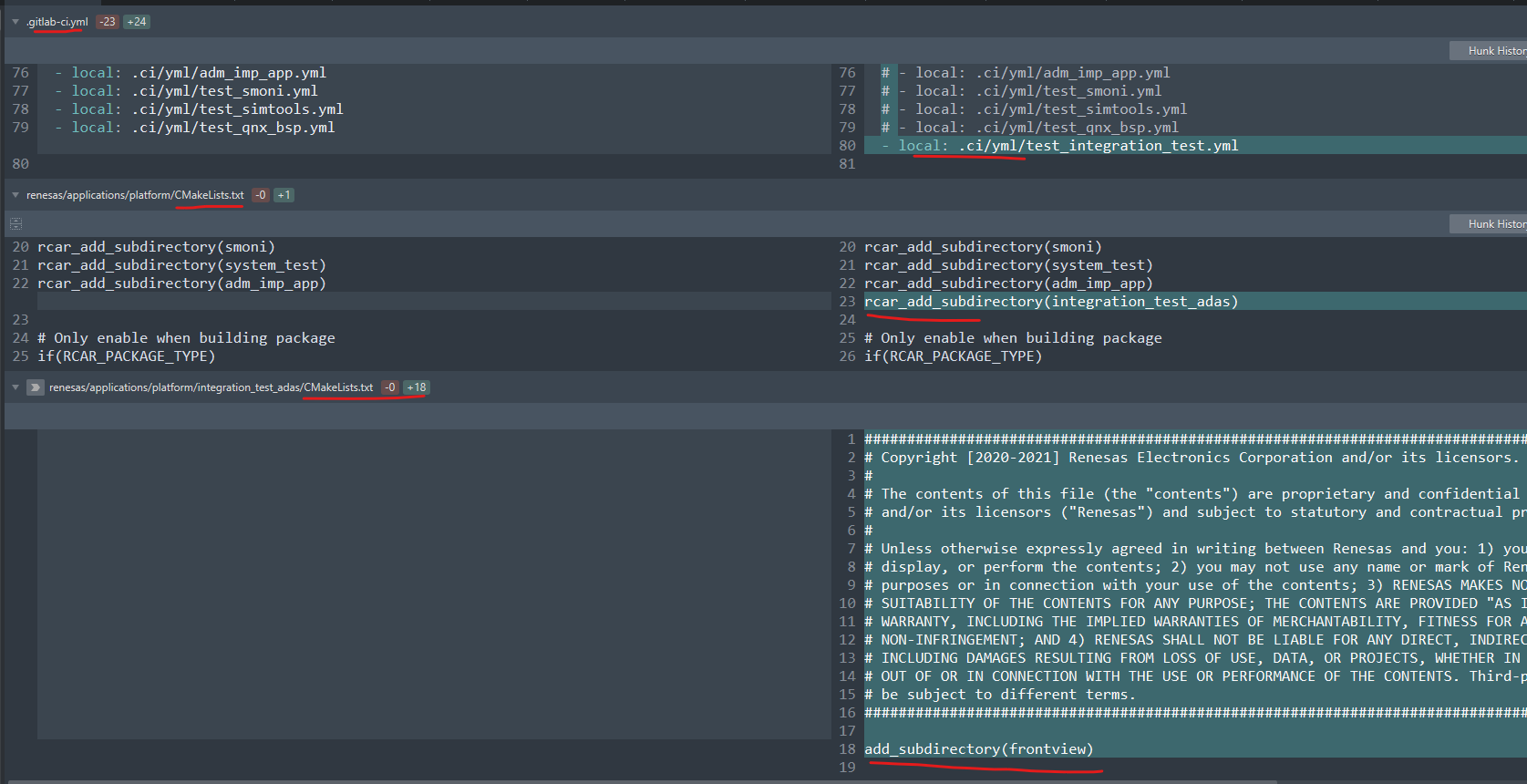


Figure 1.11.

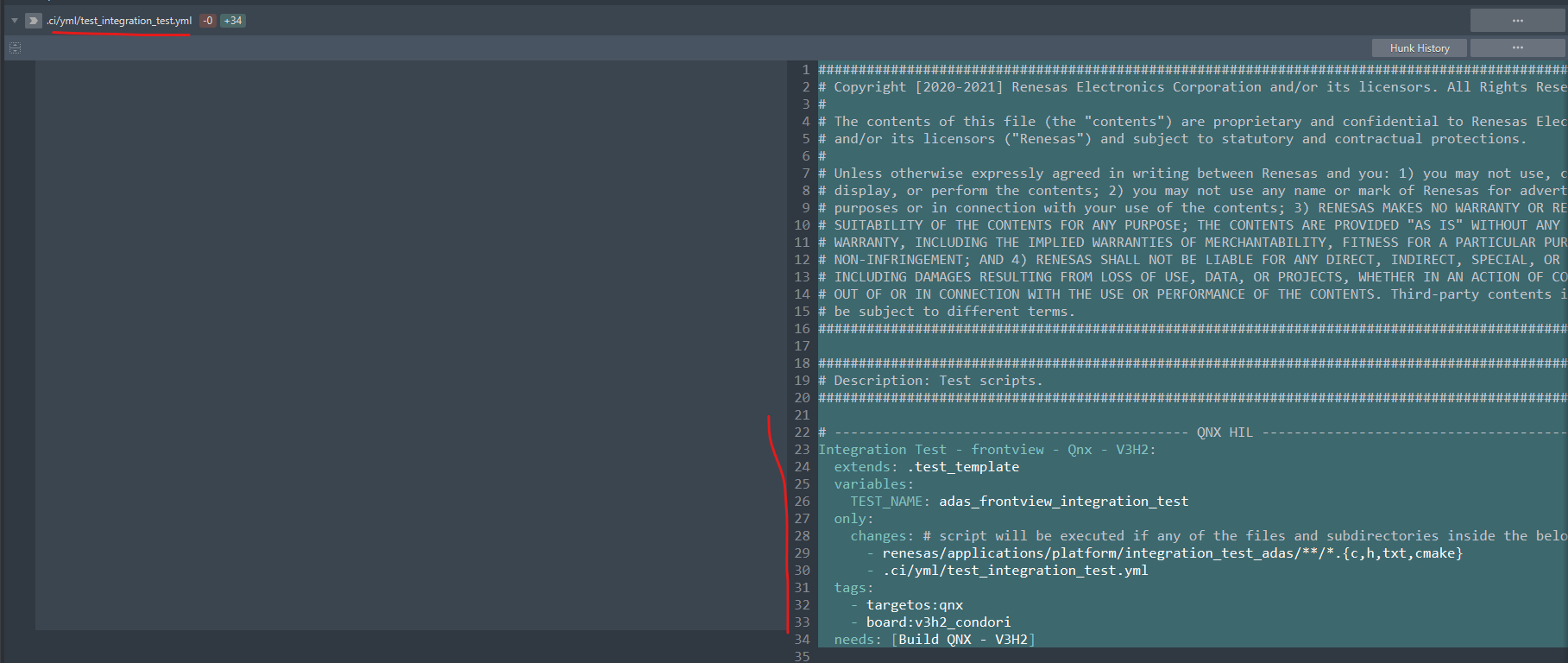


Figure 1.12.

Answer: I will changed all diff in s/c QNX before push it on release branch intg\_dev-xos2-adas-tp-rvc but you still need (because we have agreed with Linux Team that only for the official release feature from both Linux and QNX)

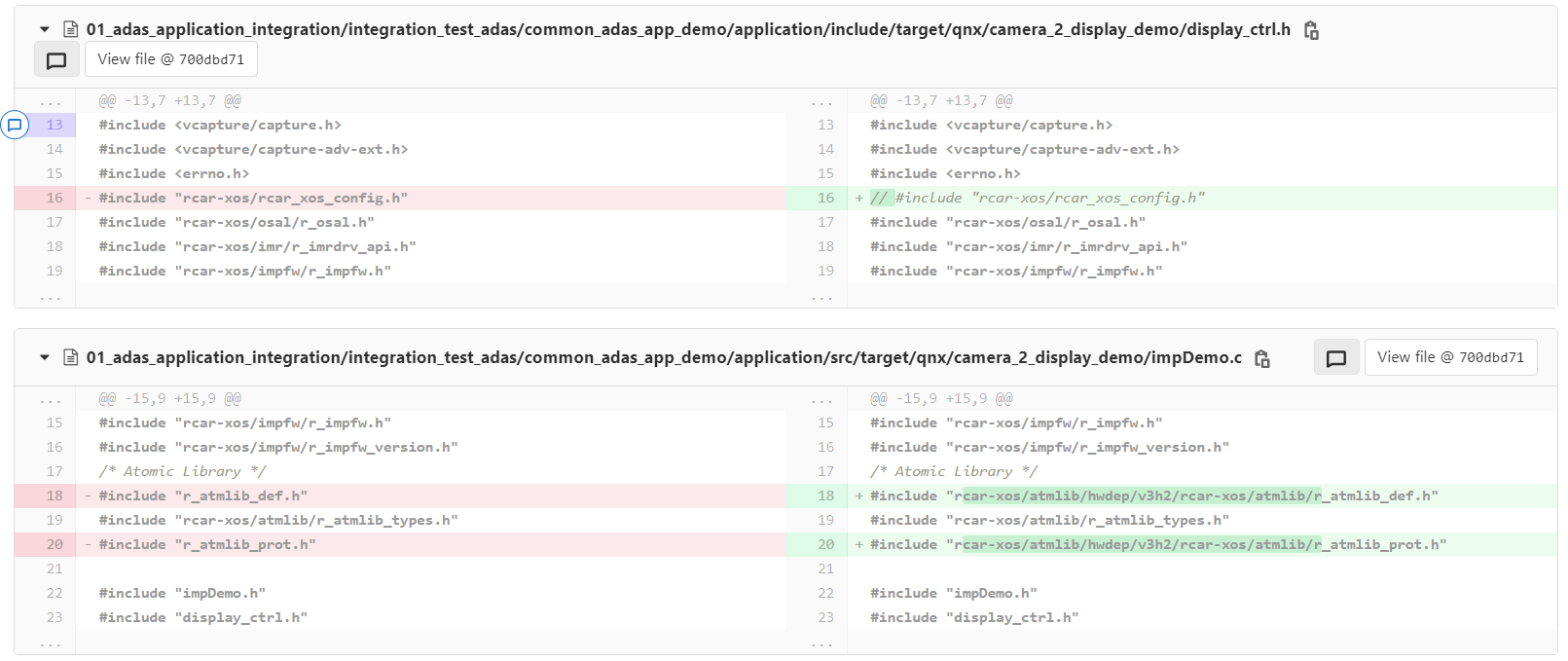
  


Figure 1.13.

* Add **rcar\_add\_subdirectory(integration\_test\_adas/<folder\_name>)** to file CMakeLists.txt at this location: renesas/application/platform (I had prepare at this location: rcar-adas-st/01\_adas\_application\_integration/CMakeLists.txt)
* create file test\_integration\_test.yml in location .ci/yml/ and include this file into .gitlab-ci.yml in rcar-env.

Question (Danh-san): How to auto create pipeline and remove stage test in CI?

Answer: Go to file template.yml and add this command in .build\_template



Figure 1.14.

And remove stage test out of CI, you comment in file .gitlab-ci.yml like this:

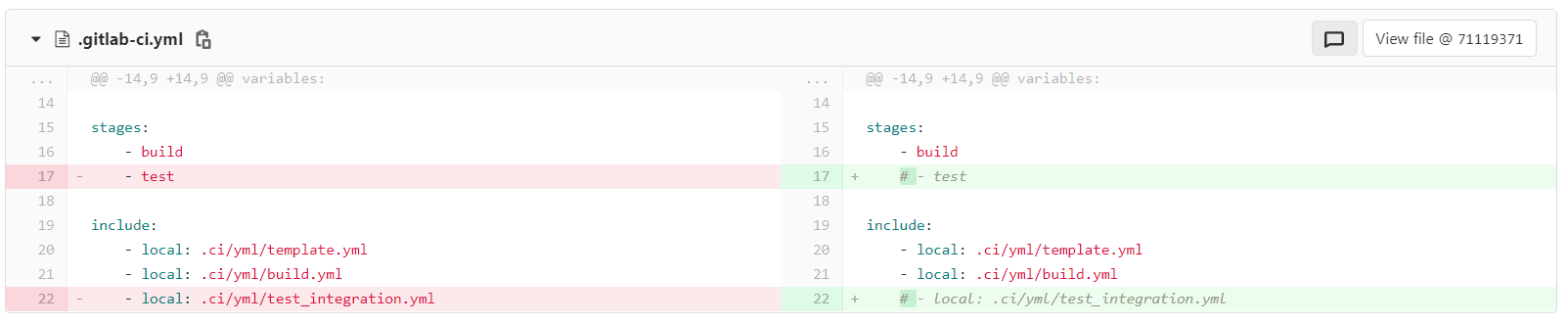


Figure 1.15.

Question (Thinh-san): At a time, could build concurrent 2 job for V3H2 and V3M2?

**Answer: GitLab** **Runner** **runs** **only** **one** **job** **at** **a** **time** by default.

1. **Without Gitlab CI (Run test app by manual)**

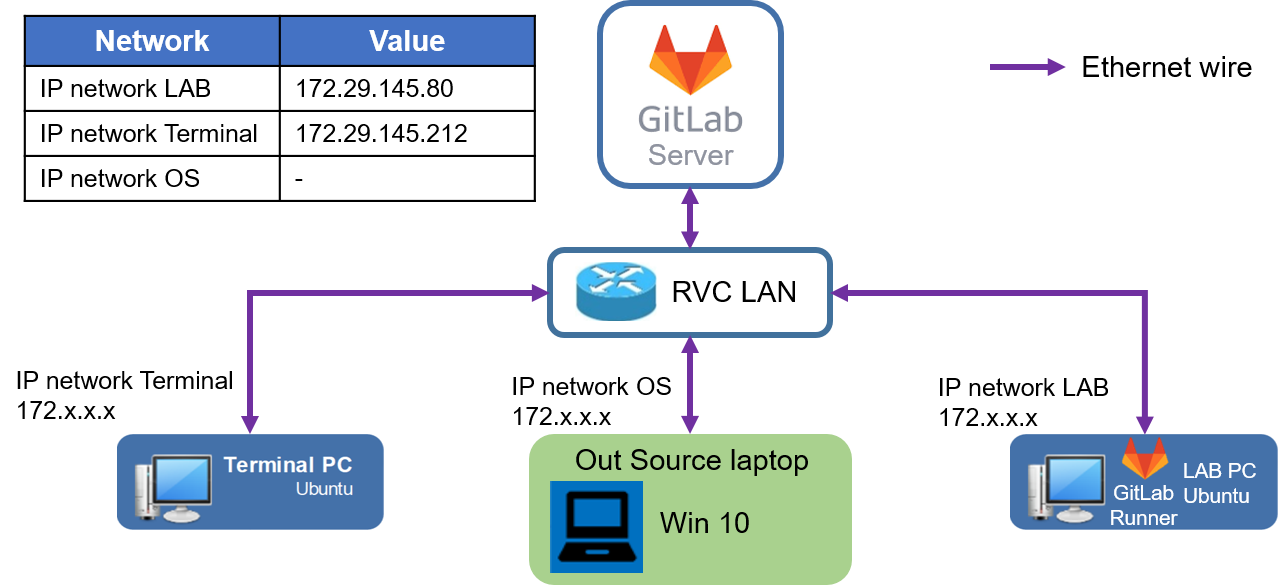


Figure 2.1.

1. Run test app by manual

* After Gitlab CI run stage build to build s/c, binary file will be stored in Terminal PC at path:

|  |
| --- |
| /shsv/RCarSW/Multimedia\_data/SystemVerificationMM\_App/Suppliers/RSS1\_OSS/01\_Prj\_ADAS/01\_QNX\_binarie/v3u\_${CI\_COMMIT\_BRANCH}\_job${CI\_JOB\_ID} |

* You must get it from Terminal PC and copy it to LAB PC by manual:
* Copy to server LAB PC: Terminal PC -> Local -> LAB PC
* At local terminal (use Moba Xterm):

|  |
| --- |
| scp <user\_name>@<IP network Terminal PC>:<path>/<file\_name> <local\_dir to store binary file>  scp <local\_dir to store binary file>/<file\_name> <user\_name>@<IP network LAB PC>:<path\_mounted\_to\_board> |

* Run test app on board:
* Method 1: Use terminal of Board (It is not recommended to use this way, which will affect Gitlab CI users)
* SSH to LAB PC

|  |
| --- |
| ssh <user\_name>@<IP network LAB PC> |

* Access to serial of board:

|  |
| --- |
| minicom –D /dev/<device\_name> - b 115200  Ex: minicom –D /dev/ttyUSB3 –b 115200 |

* Go to folder stored binary file and run it

|  |
| --- |
| Ex: /ci\_test/parkassist\_v3m2 |

* Exit access:

|  |
| --- |
| Press “Ctrl + A” => Press “Q” => Select “Leave” |

* Method 2: ssh to board from LAB PC, Terminal PC
* SSH to LAB PC or Terminal PC

|  |
| --- |
| ssh <user\_name>@<IP network LAB/Terminal PC> |

* SSH to board

|  |
| --- |
| ssh root@<board IP> -o “UserUnknownHostsFile /dev/null” |

* Go to folder stored binary file and run it

|  |
| --- |
| Ex: /ci\_test/parkassist\_v3m2 |

* Exit access:

|  |
| --- |
| exit |

1. To use FeatBox to control Board

* SSH to LAB PC

|  |
| --- |
| ssh <user\_name>@<IP network LAB PC> |

* SSH to FeatBox Controller

|  |
| --- |
| ssh pi@<IP network FeatBox> |

* Go to folder stored script auto\_ctrl.py

|  |
| --- |
| cd ~/01\_featbox\_sofware |

* Run command to control board:

|  |
| --- |
| ./auto\_ctrl.py <I2C address featbox> <command> |

* These are the commands used regularly:

Init board in the first time using featbox to control board

|  |
| --- |
| ./auto\_ctrl.py 0x03 init condor\_v3h  ./auto\_ctrl.py 0x04 init eagle\_v3m |

Provide power for board

|  |
| --- |
| ./auto\_ctrl.py 0x03 on |

Stop providing power for Board

|  |
| --- |
| ./auto\_ctrl.py 0x03 off |

Turn on the power switch for automatic boot board

|  |
| --- |
| ./auto\_ctrl.py 0x03 uboot  ./auto\_ctrl.py 0x03 boot |

Turn off the power switch for board

|  |
| --- |
| ./auto\_ctrl.py 0x03 unboot |

Press and release the Reset button on board

|  |
| --- |
| ./auto\_ctrl.py 0x03 reset |

To get more information about command control board

|  |
| --- |
| ./auto\_ctrl.py |