Integrated Development Environment e² studio

How to automate CUnit tests in e² studio and Jenkins

Introduction

Describes how to provide automatic continuous integration of Renesas e² studio projects utilizing Git, Jenkins and CUnit tests. This document references the following series of how to Application Notes. The user is advised to have these Applications Notes available and open to refer to the various sections.

- Application Note: How to use EGit in e² studio
  Integrated Development Environment e² studio: How to use EGit in e² studio (renesas.com)
- Application Note: How to use Jenkins to automate build and report
  How to use Jenkins to automate build and report (renesas.com)
- Application Note: How to use CUnit in e² studio
  Integrated Development Environment e² studio: How to use CUnit in e² studio (renesas.com)

Contents

1. Overview

1.1 Background

Git is a version control system for tracking changes in source files. EGit is an Eclipse feature, which can be installed into Renesas e² studio, providing a user interface onto the most common Git workflows. The underlying mechanism that Git uses to allow collaboration between users' source code is the repository. The "How to use EGit in e² studio", Application Note (Integrated Development Environment e² studio: How to use EGit in e² studio (renesas.com)) describes installation of EGit, into Renesas e² studio, and its use.

Jenkins is an automation server and can be used to facilitate the building of developers' source code changes (continuous integration) when committed to the version control system, in our example, Git. The user interface to Jenkins is accessed from a standard web browser. It is highly configurable and supports many standard software tools through its use of plugins, for example generating reports from unit tests (automated test). Jenkins is a service that runs on the client or a server and is separate from Renesas e² studio. The "How to use Jenkins to automate build and report" Application Note (How to use Jenkins to automate build and report (renesas.com)) describes client/server installation and how to setup automatic continuous integration build jobs.

CUnit is a framework for writing and running unit tests in the C programming language. CUnit is built as a static library which is linked with the user's application and testing code. The "How to use CUnit in e² studio", Application Note (Integrated Development Environment e² studio: How to use CUnit in e² studio (renesas.com)) describes the process of writing tests to test the user's application code, using Renesas e² studio, in a form suitable for a Jenkins job to run automatically.

1.2 Purpose

In this document we consolidate the information presented in the EGit, Jenkins and CUnit usage Application Notes, using a worked example.

We'll create an empty git repository and using the SampleCUnit project example, described in Integrated Development Environment e² studio: How to use CUnit in e² studio (renesas.com), we'll show the steps necessary to push it to this repository. We'll then setup a client-side Jenkins server, create and configure a single job which builds and runs the CUnit tests. The figure below depicts the process that is available once these steps have been completed.

1		Author application and CUnit test	In Renesas e² studio create the application and CUnit test(s) or make changes to an existing test.
2		Push change	Using EGit within Renesas e² studio, push the changes into the remote Git repository.
3		Trigger	The Jenkins server is setup with a build and test job which periodically polls the remote Git repository to see if any changes have been pushed. Once changes are detected the build job is started.
4		Build	The build job pulls the latest changes from the remote Git repository into its workspace on the server and builds the SampleCUnit project.
5		Test	The SampleCUnit binary is then executed which runs the CUnit tests and generates a report.
6		Result report	In the Jenkins UI, the developer can inspect the test results.

1.3 Operating Environment

Renesas have confirmed the operation procedure explained in this document in the following environment.

OS	Windows 10 x64
IDE	e² studio 2022-01
Jenkins	2.332.1
Git	2.35.1
CUnit	2.1.2

In this document uses the following target device and toolchain with an example project.

Target device	RX64M
Target toolchain	GCC for Renesas RX 8.3.0.202104-GNURX-ELF

2. Create CUnit Sample Project

This chapter describes the steps of the target project creation method.

2.1 Creating a project

Follow the steps described in Integrated Development Environment e² studio: How to use CUnit in e² studio (renesas.com), create CUnit library project “CUnit” and the test target project “SampleCUnit”.

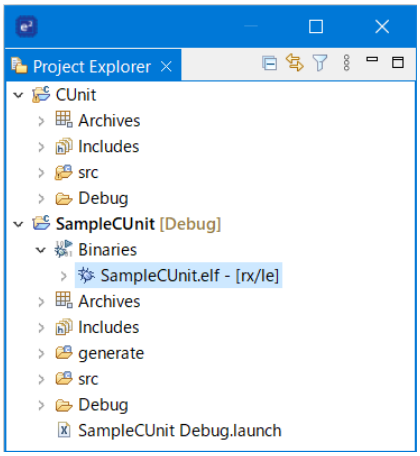


Figure 1

2.2 Modifying a project for automation test

To build a project in Jenkins, you have to make the project self-contained. Therefore, you need to link CUnit library as a static library. Also, in order to run unit tests for a project in Jenkins, you need to change the test results to be output to a file instead of being output to the console screen.

1) Copy the "Headers" folder of the CUnit project to the SampleCUnit project with following the steps below.
   a. In the [Project Explorer] view, select (CUnit > src >) “Headers”, hold down the Ctrl key and drag and drop it into “SampleCUnit”.
   b. The [File and Folder Operation] dialog is appeared. Check the [Copy files and folders] radio button and click the [OK] button.
2) Copy the “libCunit.a” file of the CUnit project to the SampleCUnit project with following the steps below.
   a. In the [Project Explorer] view, select “SampleCUnit”, and select context menu [New] > [Folder].
   b. The [New Folder] dialog is appeared. Input “lib” to the [Folder name:] edit box and click the [Finish] button.
   c. In the [Project Explorer] view, select (CUnit > Archives >) “libCUnit.a”, hold down the Ctrl key and drag and drop it into (SampleCUnit >) “lib”.
   d. The [File Operation] dialog is appeared. Check the [Copy files] radio button and click the [OK] button.

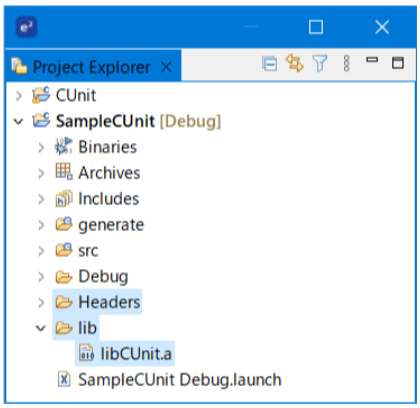


Figure 2

3) In the [Project Explorer] view, select “SampleCUnit” and select menu [Project] > [C/C++ Project Settings].

4) The [Property: SampleCUnit] dialog is appeared. Select tree node “Compiler > Includes” on the [Tool Settings] tab. Replace “${workspace_loc:/CUnit/Headers}” to “${workspace_loc/${ProjName}/Headers}” in the [Include file directories] list box.

5) Select tree node “Linker > Source” and replace “${workspace_loc:/CUnit/Debug/libCUnit.a}” to “${workspace_loc:/${ProjName}/lib/libCUnit.a}” in the [Additional input files] list box. Next, click [Apply and Close] button.

6) To set CUnit automatic mode, be sure to modify "SampleCUnit.c" file as follows.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Basic.h"

int main(void);
extern AddTests();

int main(void)
{
  // Define the run mode for the basic interface
  // Verbose mode - maximum output of run details
  CU_BasicRunMode mode = CU_BRM_VERBOSE;

  // Define error action
  // Runs should be continued when an error condition occurs (if possible)

  CU_ErrorAction error_action = CUEA_IGNORE;
  // Initialize the framework test registry
  if (CU_initialize_registry()) {
    printf("Initialization of Test Registry failed.\n");
  }
  else {
    // Call add test function
    AddTests();

    // Set the basic run mode, which controls the output during test runs
    CU_basic_set_mode(mode);

    // Set the error action
    CU_set_error_action(error_action);

    // Run all tests in all registered suites
    CU_automated_run_tests();
    printf("Tests completed.\n");

    // Clean up and release memory used by the framework
```

```
      CU_cleanup_registry();
    }
    return 0;
}
```

7) In the [Project Explorer] view, select “SampleCUnit” and select context menu [Build Project]. Build will be started. And make sure there are no errors.

8) In the [Project Explorer] view, select “SampleCUnit > Binary > SampleCUnit.elf” and select context menu [Show in Local Terminal] > [Terminal].

9) The [Terminal] view is appeared. Input “rx-elf-run SampleCUnit.elf” and push Enter key.
You set CUnit automatic mode, the test result is not displayed on the [Terminal] view.
“Debug/CUnitAutomated-Results.xml” file will be saved as the test result file.

3. Create Git Repository

This chapter describes the steps of the Remote/Local repository creation method.

3.1 Creating a remote repository

1) Launch Renesas e² studio and switch to the [Git] perspective and in the [Git Repositories] view, click the [Create a new Git Repositories and add it to this view] button on the toolbar.

2) The [Create a Git Repositories] dialog is appeared. Input the remote repository folder in the [Repository directory:] editbox, e.g. “\\192.168.0.43\Shared\git\remote\cunit”.

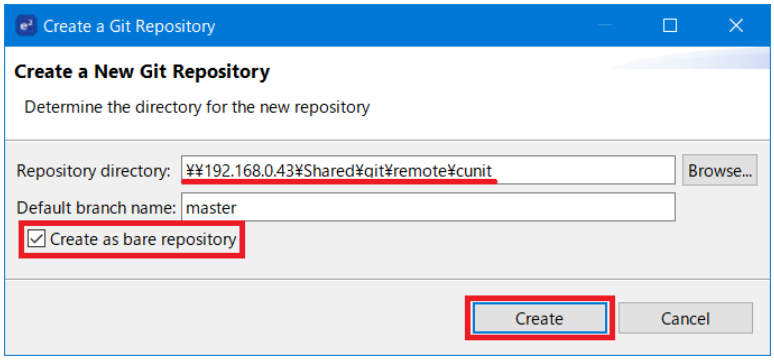


Figure 3

3) Check the [Create as a bare repository] checkbox and click the [Create] button.
If you select the [Create as a bare repository] checkbox, the new repository will not have a working directory. You can only add content by pushing changes from another repository.

Refer to: https://wiki.eclipse.org/EGit/User_Guide/3._Tasks#Bare_Repositories

This repository view can now be removed (not deleted) from the Git Repositories view because we will never work directly in this repository; it will always be via clones of this repository.

4) In the [Git Repositories] view, select “cunit” and press the [Delete] key. The [Delete Repository] message is appeared, click the [Remove from View] button.

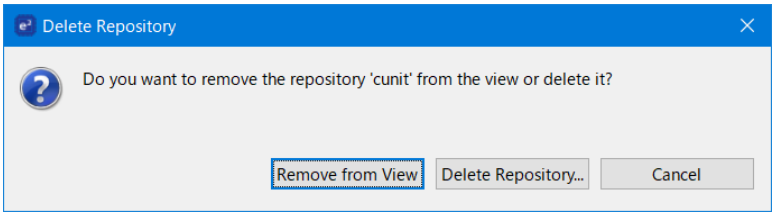


Figure 4

3.2 Creating a local repository

1) Switch to the [Git] perspective. In the [Git Repositories] view, click [Clone a Git Repository and add the clone to this view] button. The [Clone Git Repository - Source Git Repository] dialog is appeared. Input the remote repository folder to [Repository path:] edit box and select “file” in [Protocol] combo box. And click [Next >] button.

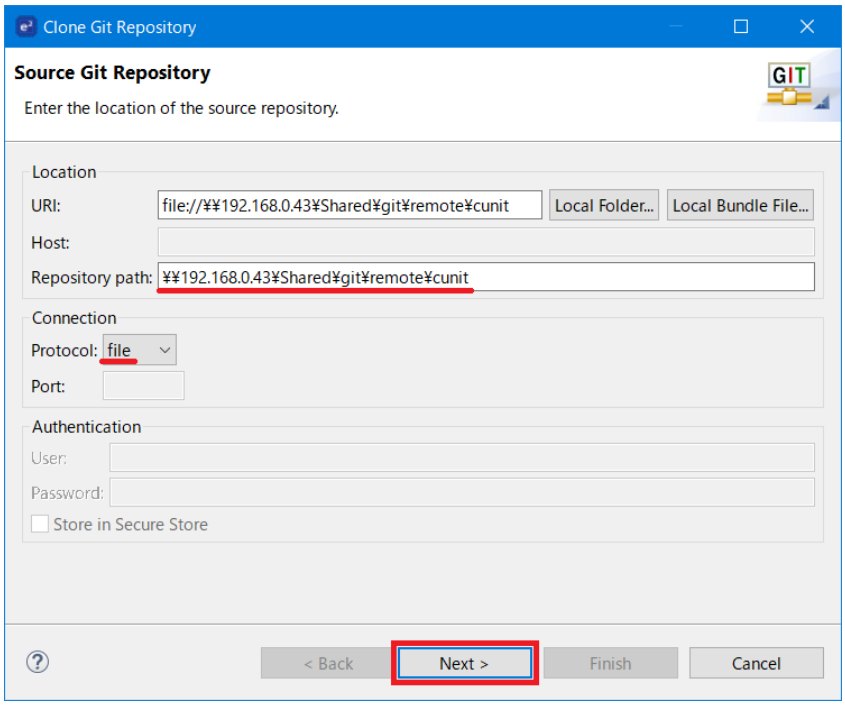


Figure 5

2) In the [Clone Git Repository - Branch Selection] dialog, click [Next >] button.

3) The [Clone Git Repository – Local Destination] dialog is appeared. In the [Directory:] field enter the folder on the local repository, e.g. “C:\git\local\cunit”, and click [Finish] button.

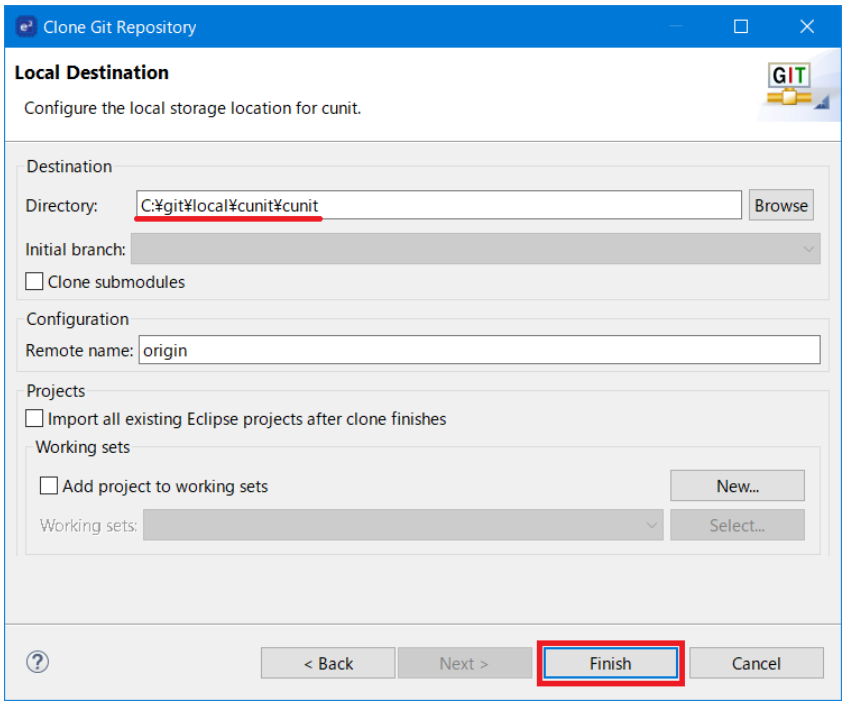


Figure 6

3.3 Starting version control of project

1) Switch to the [C/C++] perspective. In the [Project Explorer] view, select “SampleCUnit” project, and click context menu [Team] > [Share Project...].

2) The [Share Project] dialog is appeared. Select the local repository in the [Repository] combo box and click [Finish] button.

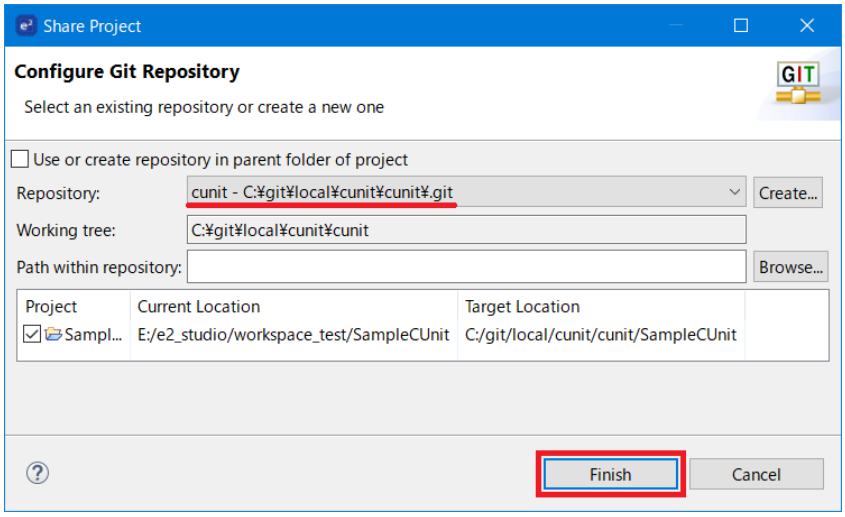


Figure 7

3) Switch to the [Git] perspective. In the [Git Repositories] view, select “cunit”.

4) In the [Git Staging] view, the new files in the project appeared in the [Unstaging Changes] list box.

5) Click [Add all files including not selected ones to the index] button on the [Unstaging Changes] list box. The items in the [Unstaging Changed] list box is moved to the [Staged Changed] list box. Then enter your message in the [Commit Message] edit box (e.g. Initial) and click the [Commit and Push...] button.

6) The [Push Results] dialog is appeared. Check the message and click the [Close] button.

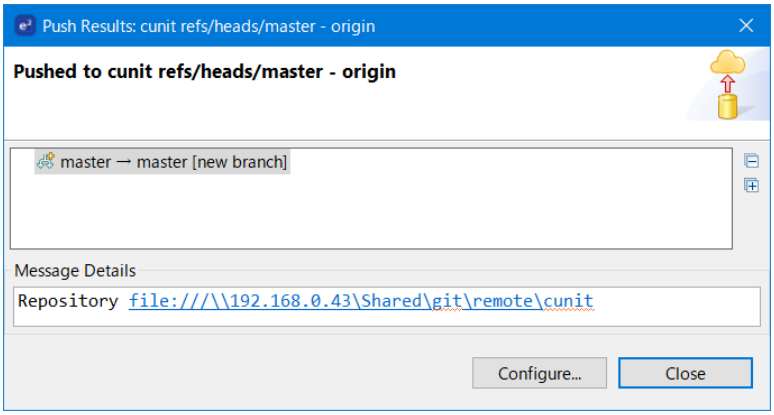


Figure 8

The SampleCUnit project has been committed to the local repository and pushed to the remote repository.

4. Setup Jenkins client

This chapter describes how to install Git plug-in and xUnit plug-in in Jenkins.

4.1 Installing Git plugin

For Jenkin to recognize Git, you need to install the Git plug-in in Jenkin. Refer to the application note (How to use Jenkins to automate build and report (renesas.com)) to install the Git tool and Git plug-in.

- Git tool
- Gi plug-in

4.2 Installing xUnit plugin

The xUnit plug-in allows you to share the test results of running the test tool in Jenkins. CUnit creates a test report in automated mode, which xUnit helps to share the results with other members within Jenkins.

1) On the [Jenkins] page, click the [Jenkins Manager]. Next, click [Plugin Manager].
2) The [Plugin Manager] page is appeared. Click the [Available] tab and enter “xUnit” in the filter. The page content is filtered by “xUnit”.

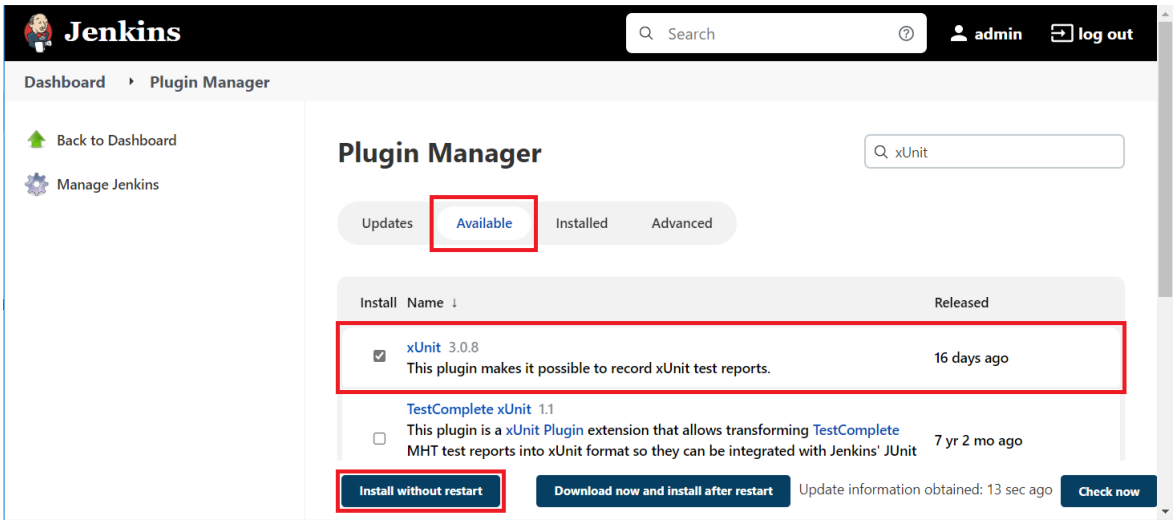


Figure 9

3) Check the [xUnit] and click the [Install without restart] button. The installation will begin.
4) When the installation is complete, check the [Restart Jenkins when installation is completed and no jobs are running]. Jenkins will restart automatically.
5) Log in to Jenkins again.

Now xUnit is available in Jenkins.

4.3 Setting up environment variables

You must set the environment variables to ensure that the commands for the test job that you create later are correctly searched for and executed.

Follow these steps to set the environment variable Path:

1) On the [Jenkins] page, click the [Manage Jenkins], and then click the [System Settings].

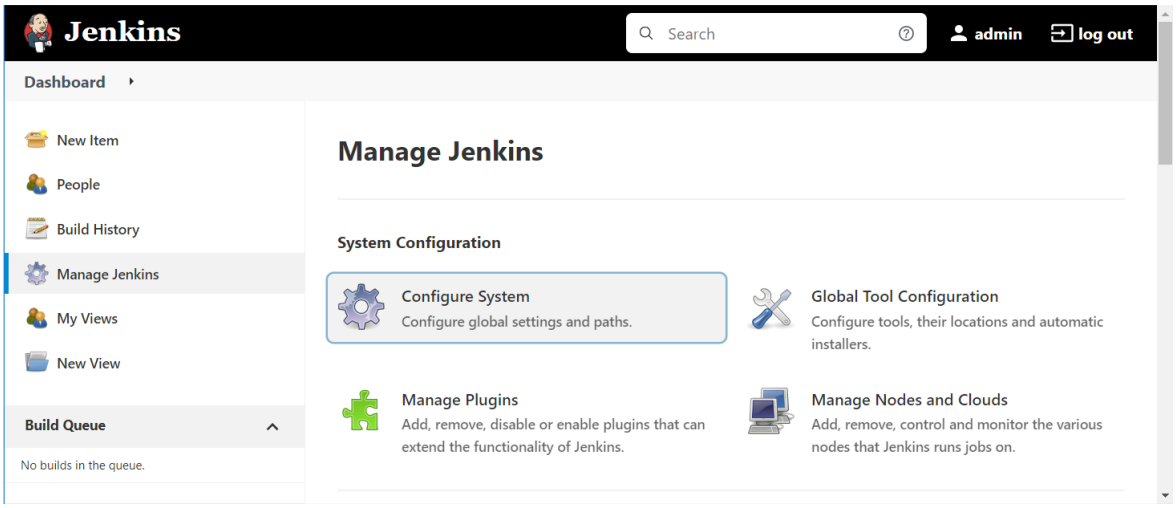


Figure 10

2) The configuration page is appeared. Scroll to the [Global properties] and check the [Environment variables] check box. Specify the [Name] and [Value] as follows:
[Name]: Path
[Value]: < GCC for Renesas RX C/C++ Toolchain installation directory >\rx-elf\rx-elf\bin;$Path

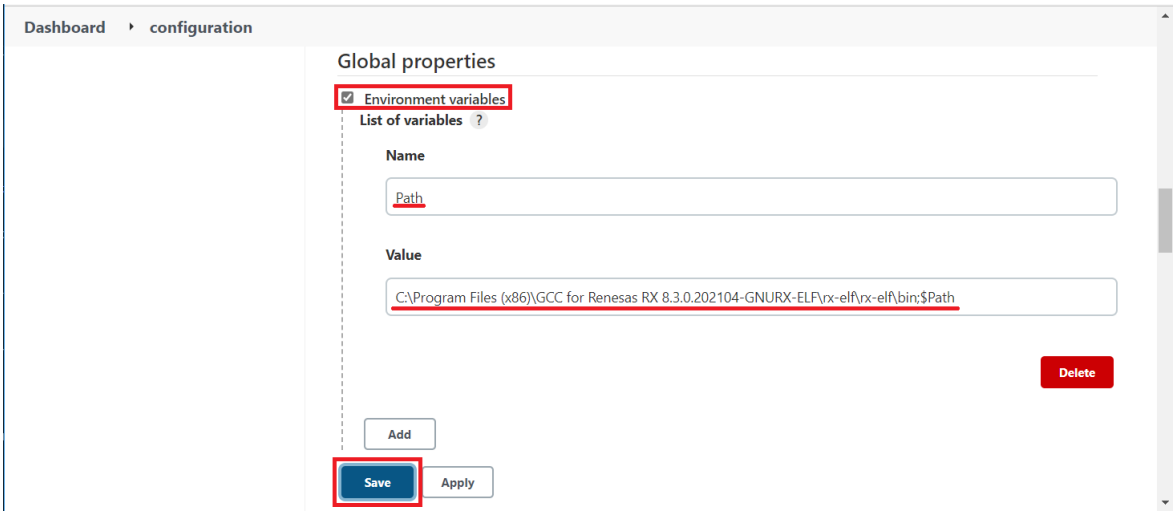


Figure 11

3) Click the [Save] button. Once saved, the [Dashboard] page is appeared.

5. Create Build and Test Jenkins Job

This chapter describes how to create jobs in Jenkins.

5.1 Creating Build job

1) On the [Jenkins] page, click the [New Item].

2) In the [Enter an item name] edit box, enter “Build_SampleCUnit” and click the [Freestyle project]. Then click the [OK] button.

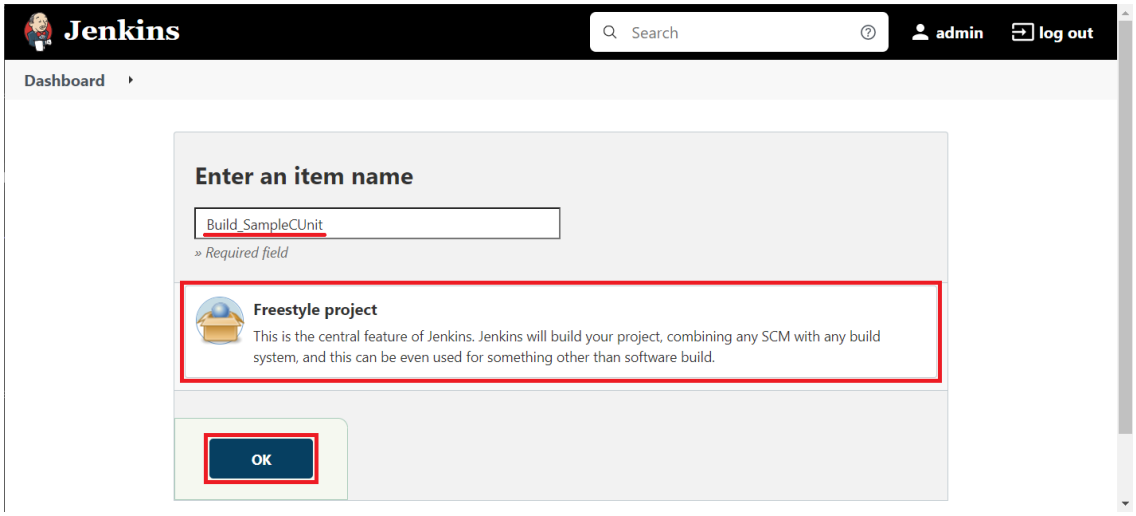


Figure 12

3) A screen for setting advanced job settings is displayed.

4) Click the [Source Code Control] tab. The [Source Code Management] is appeared, so check the [Git] and specify the remote repository in the [Repository URL] edit box. If necessary, add your credentials.

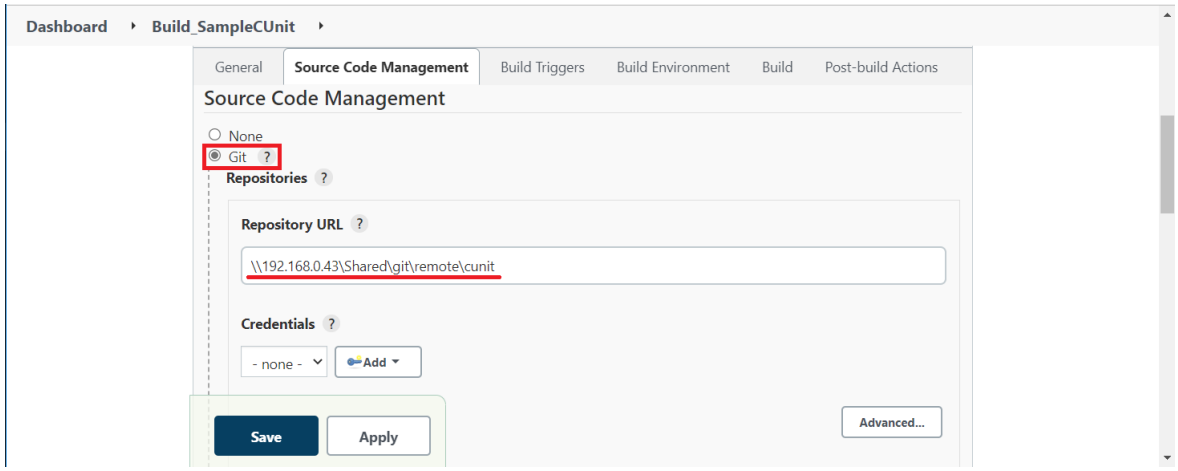


Figure 13

Note: In the build job, the following error may occur.

```
ERROR: Checkout of Git remote 'XXXXXXXX' aborted because it references a local
directory, which may be insecure. You can allow local checkouts anyway by setting
the system property 'hudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT' to true.
Finished: FAILURE
```

Due to Jenkins security enhancements, access to repositories located at the local URL or path of the Git plugin is not allowed. (Git | Jenkins plugin)

To grant access, add "-Dhudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT=true" to "jenkins.xml" in the Jenkins installation folder.

```
<arguments>-Xrs -Xmx256m -
Dhudson.lifecycle=hudson.lifecycle.WindowsServiceLifecycle -
Dhudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT=true -jar ...
```

5) Click the [Build Triggers] tab. Set when the build runs. To build every time the project is updated in the Git repository, check the [Poll SCM]. This will periodically check for updates to your Git repository and run the build if there are any updates.

6) The interval at which to check for Git repository updates is specified in cron-like syntax. Here, enter “H/15 * * * *”. This means checking the Git repository every 15 minutes. For more information, click the [?] button.

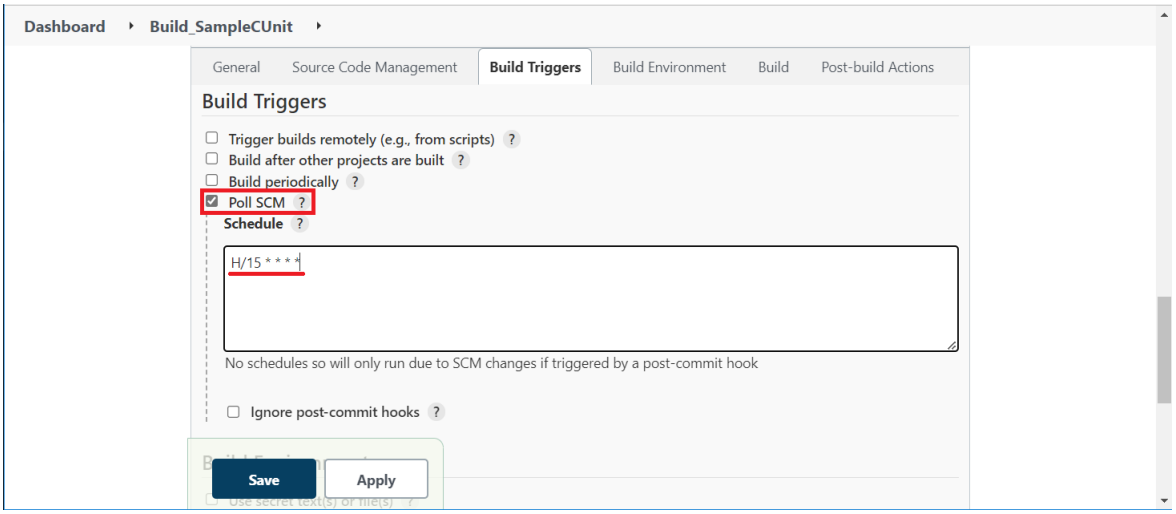


Figure 14

7) Click the [Build] tab. In the [Add build step] combo box, select “Execute Windows batch command”.

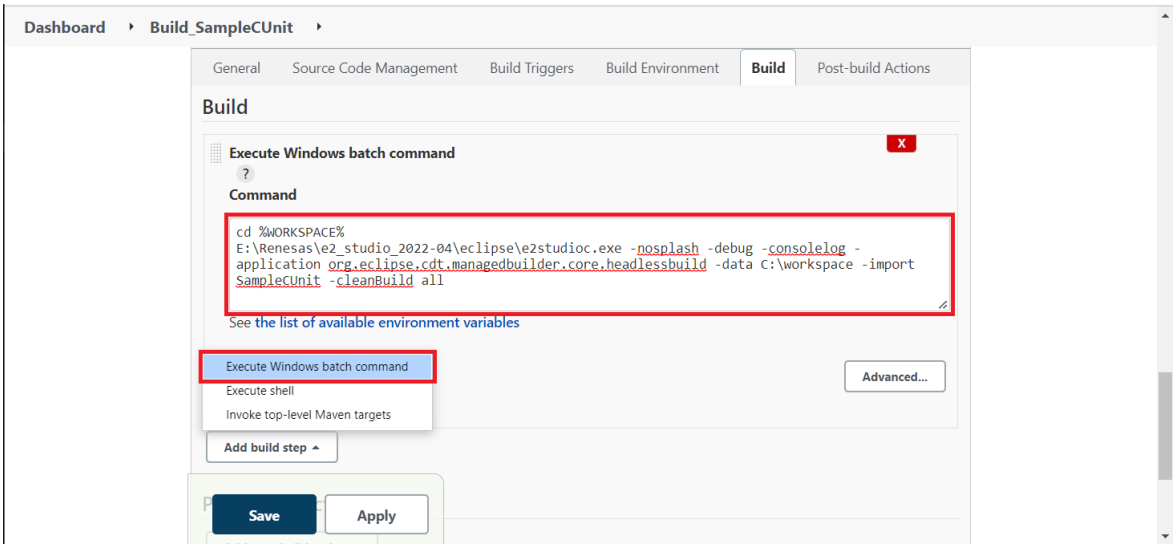


Figure 15

8) In the [Command] edit box, enter the following command for running a headless build of e² studio:

A headless build is to build a project on the command line without using the e² studio UI. For more information on command-line options, see the help for the e² studio headless build feature.

```
cd %WORKSPACE%
<e² studio install folder>\eclipse\e2studioc.exe -nosplash -debug -
consolelog -application org.eclipse.cdt.managedbuilder.core.headlessbuild
-data <workspace folder> -import <project name> -cleanBuild all
```

- <e² studio install folder>: Specify the e² studio installation directory.
- e2studioc.exe: e² studio 2021-10 and earlier versions, specify eclipsec.exe.
- <workspace folder>: Headless builds do not use the e² studio UI, however require a workspace specification as usual. Specify a workspace folder that does not contain the project you want to use.
- <project_name>: Specify the name of the project registered in the Git repository. Here, you specify "SampleCUnit".

9) Click the [Save] button. Once saved, the Project Build_SampleCUnit page is appeared.

5.2 Creating Test job

1) On the Jenkins page, click the [New Item].
2) In the [Enter an item name] edit box, type “UnitTest_SampleCUnit” and click [Freestyle project]. Then click the [OK] button.

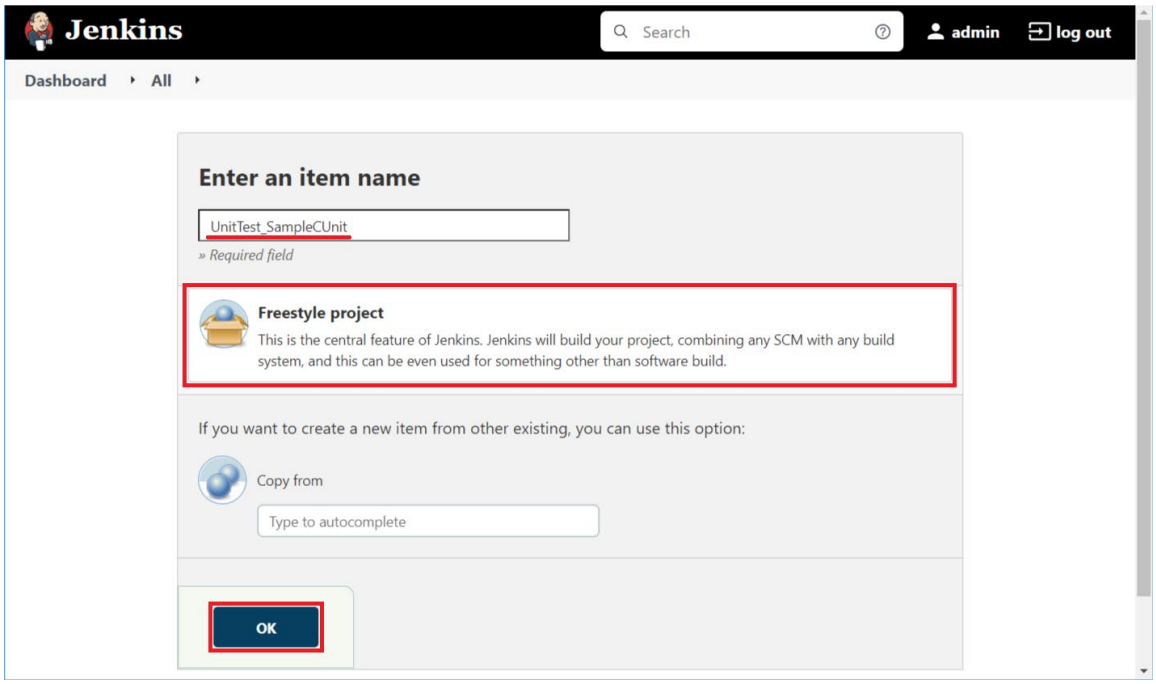


Figure 16

3) The screen for setting advanced job settings is displayed.
4) Click the [Build Triggers] tab. When [Build Trigger] is displayed, check the [Build after other project are built] and enter "Build_SampleCUnit" in the [Projects to watch] edit box.

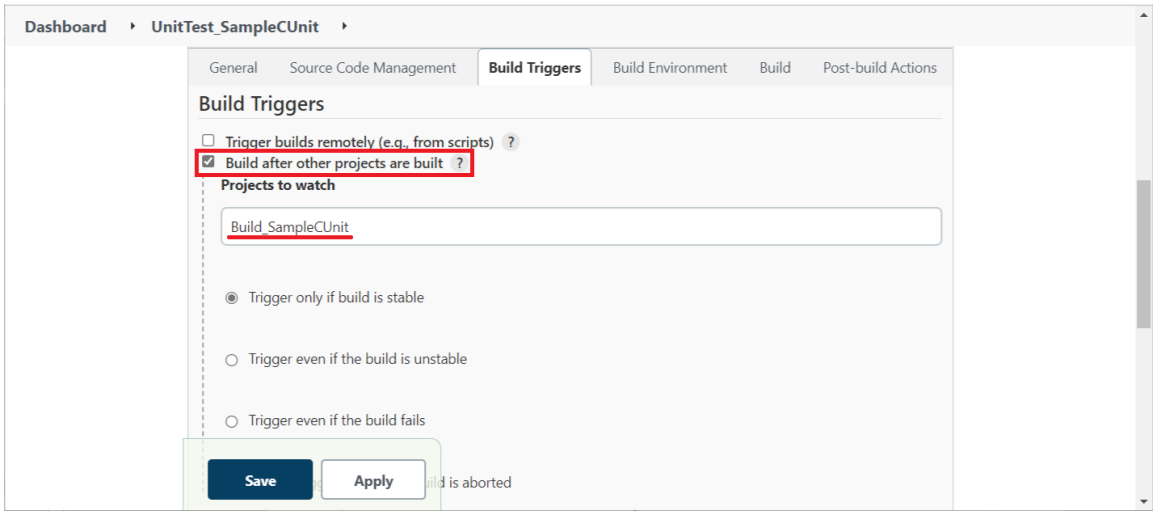


Figure 17

5) Click the [Build] tab. [Build] will be displayed, so in the [Add build step] combo box, select "Execute Windows batch command".

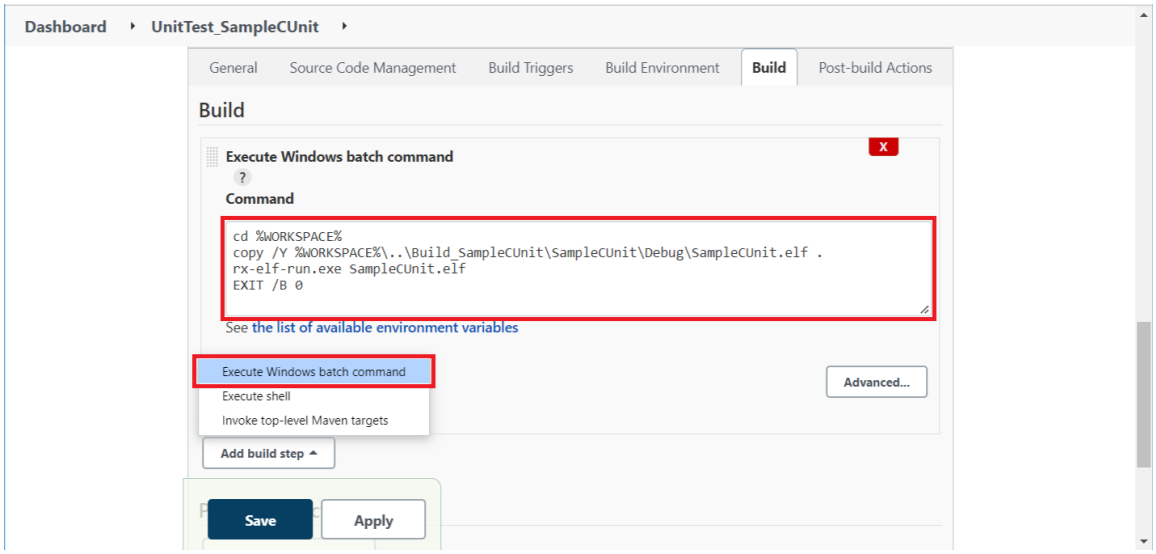


Figure 18

6) In the [Command] edit box, enter the following command, and then click the [Save] button.

```
cd %WORKSPACE%
copy
/Y %WORKSPACE%\..\Build_SampleCUnit\SampleCUnit\Debug\SampleCUnit.elf .
rx-elf-run.exe SampleCUnit.elf
EXIT /B 0
```

7) Click the [Post-build Actions] tab. The [Post-build Actions] page is displayed, so select "Publish xUnit test result report" in the [Add post-build action] combo box.

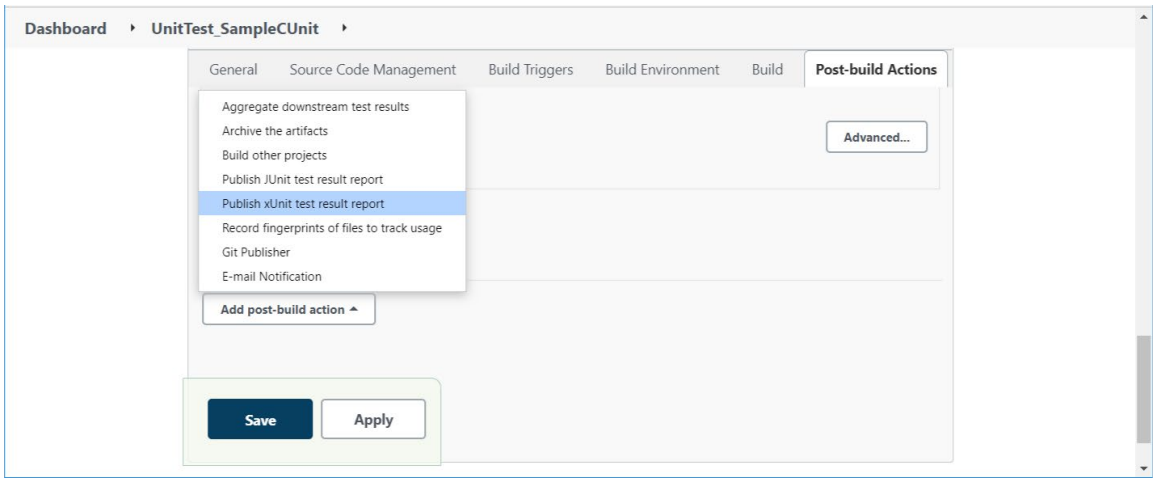


Figure 19

8) Then, in the [Add] combo box under [Report Type], select “CUnit-2.1 (default)”.

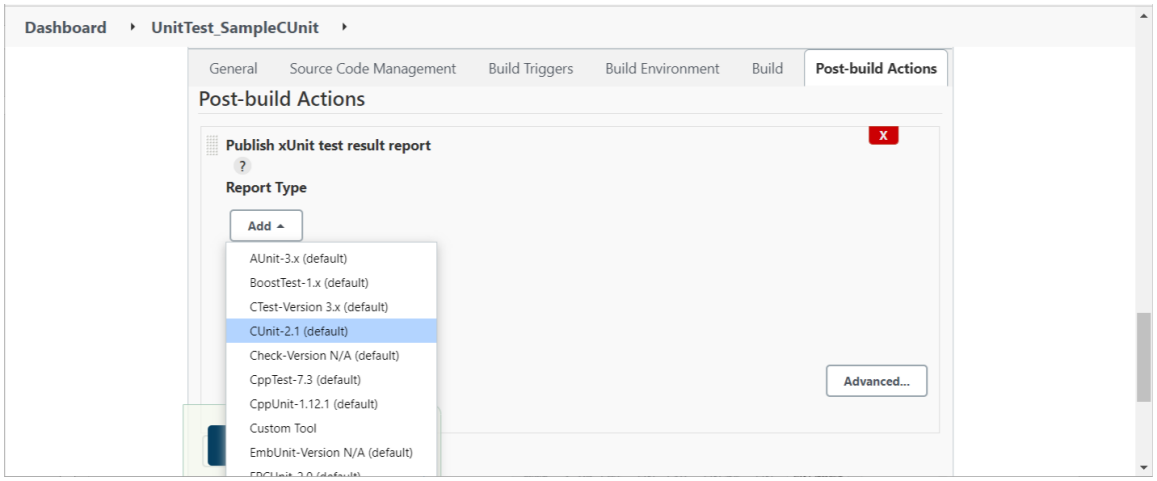


Figure 20

9) In the [Includes Pattern] edit box, type “*.xml”.

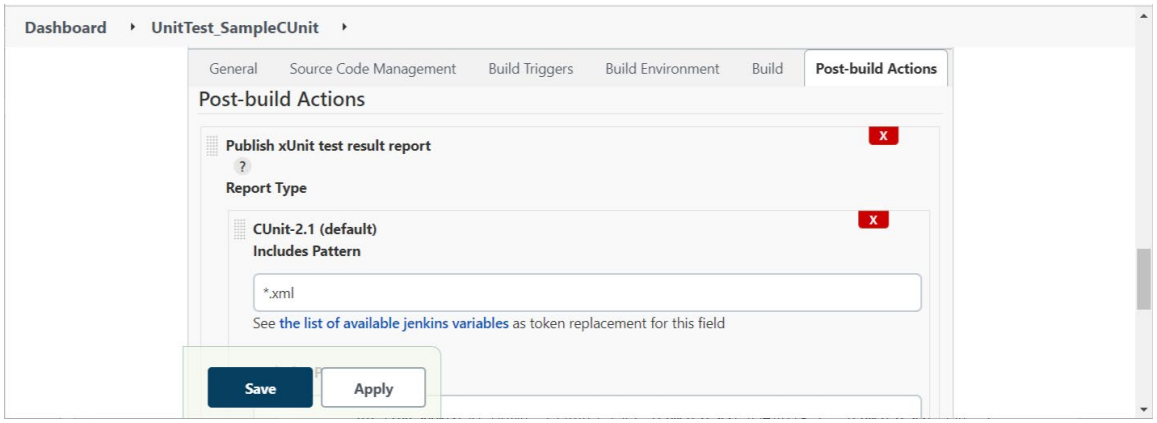


Figure 21

10) In the [Add] combo box for the [Thresholds], select “Failed Tests”. The [Failed Tests] group box appears. Then, in the [Add] combo box, select “Skipped Tests”. The [Skipped Tests] group box appears. Click the [Advanced...] button. Under Choose your threshold mode, check the [Use a percent of tests].

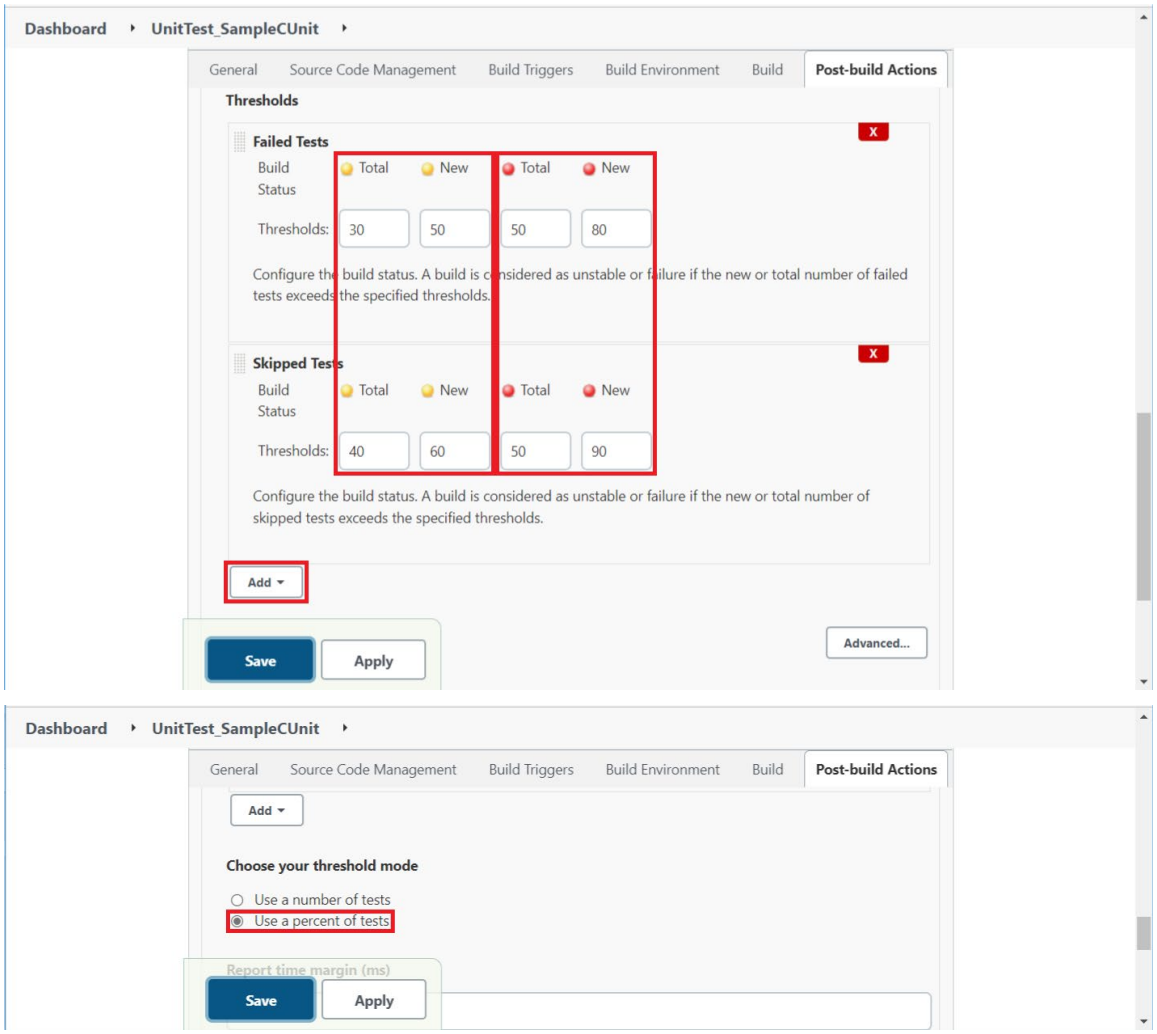


Figure 22

11) Enter the threshold shown in the figure. The meaning of the specified number is as follows.

- When the job is determined to be "unstable ( )"
  - More than 30% of "Total" test cases in "Failed Tests" fail
  - More than 50% of the "New" test cases in "Failed Tests" fail
  - Skip more than 40% of "Total" test cases in "Skipped Tests"
  - Skip more than 60% of "New" test cases in "Skipped Tests"
- When the job is determined to be "failed ( )"
  - More than 50% of "Total" test cases in "Failed Tests" fail
  - More than 80% of the "New" test cases in "Failed Tests" fail
  - Skip more than 50% of "Total" test cases in "Skipped Tests"
  - Skip more than 90% of "New" test cases in "Skipped Tests"

12) Click the [Save] button. When the save is complete, the [Project Test_SampleCUnit] page appears.

6. Test

This chapter describes how to add new tests and review test results.

6.1 Add a new test

Modify the SampleCUnit project by adding new tests and committing them to the Git repository. When Jenkins SCM polling confirms the changes, Build_SampleCUnit job runs. When Build_SampleCUnit job completes Test_SampleCUnit the job runs.

1) Start e² studio and select the [C/C++] perspective.
2) In the [Project Explorer] view, double-click “(SampleCUnit > src >) testsource.c” to display it in the editor.
3) Add the new test code shown in yellow below and save it.
   - testsource.c

```
…

static void test_Add_02(void) {
  CU_ASSERT_EQUAL(10, add(1,9));
}

static void test_Add_03(void) {
  CU_ASSERT_EQUAL(5, add(2,3));
}

// This is a test case used to test subtract() function in source.c
static void test_Subtract(void) {
  // 0 is expected value, and subtract(1,1) is actual return value.
  // If expected value is not same, assertion occurs.
  CU_ASSERT_EQUAL(0, subtract(1,1));
}

// This is a test suite
static CU_TestInfo tests_Add[] = {
    // Register yesy case to test suite
    {"test_Add_01", test_Add_01},
    {"test_Add_02", test_Add_02},
    {"test_Add_03", test_Add_03},
    CU_TEST_INFO_NULL,
};
…
```

4) Build the project to make sure there are no errors.
5) Select the [Git] perspective and in the [Git Staging] view, move the modified file testsource.c from the [Unstaged Changes] list box to the [Staged Changes] list box. Add a message (e.g. “Update test case”) to the [Commit Message] edit box.
6) Click the [Commit and Push...] button.

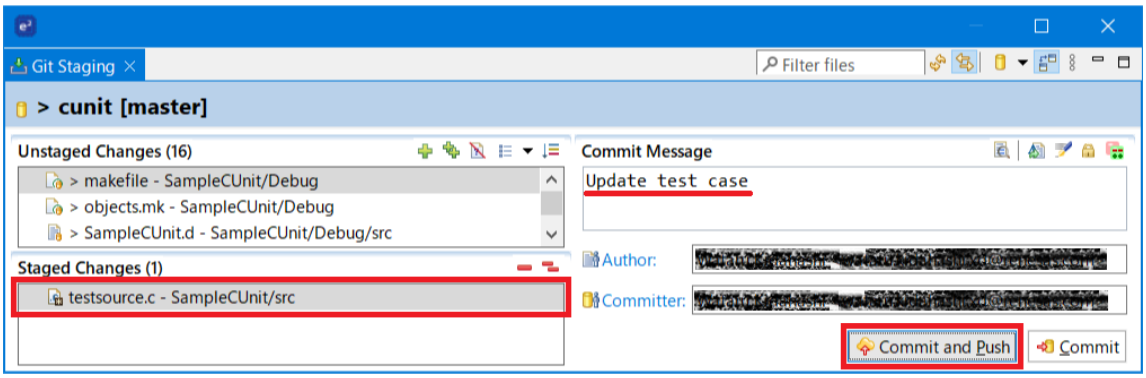


Figure 23

When the Git polling period ends, Jenkins realizes that there are pending changes and runs the Build_SampleCUnit job. Continue to run the Test_SampleCUnit job. The test results are displayed in the transition graph of the test results, as shown below.

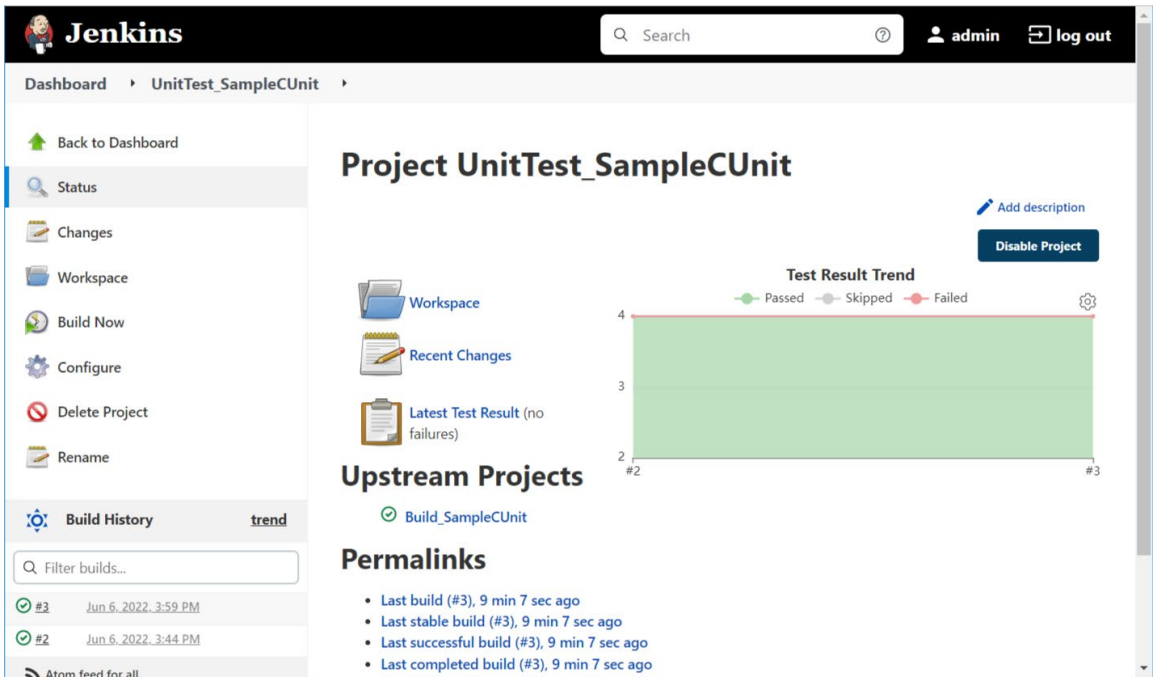


Figure 24

6.2 Verify test result

1) On the [Jenkins] page, click the “#number” for the Latest Successful Build for Test_SampleCUnit job.

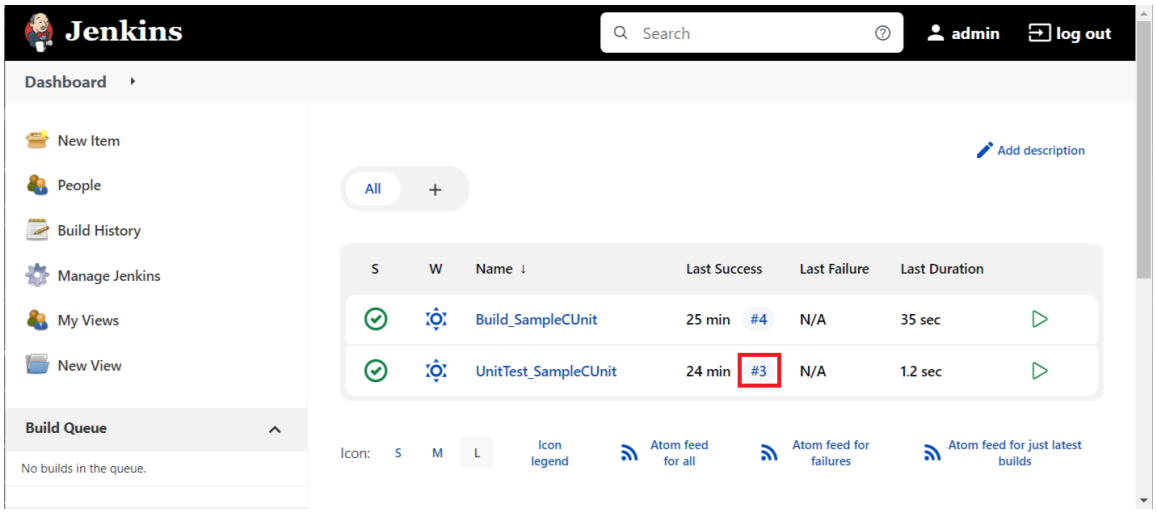


Figure 25

2) The [Build # Number] page appears, so click “Test Result”.

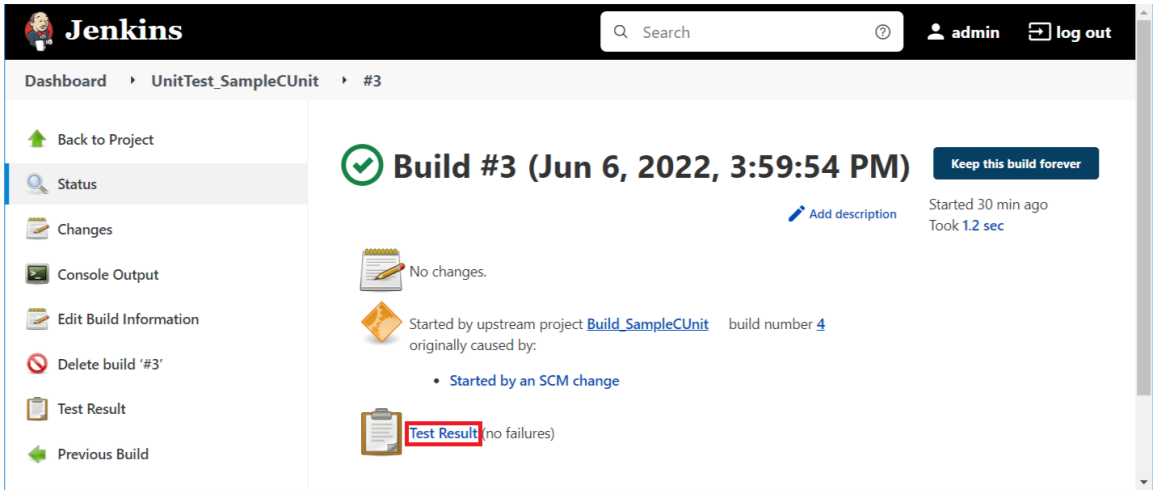


Figure 26

3) The [Test Result] page appears. Click “(root)” for the package.

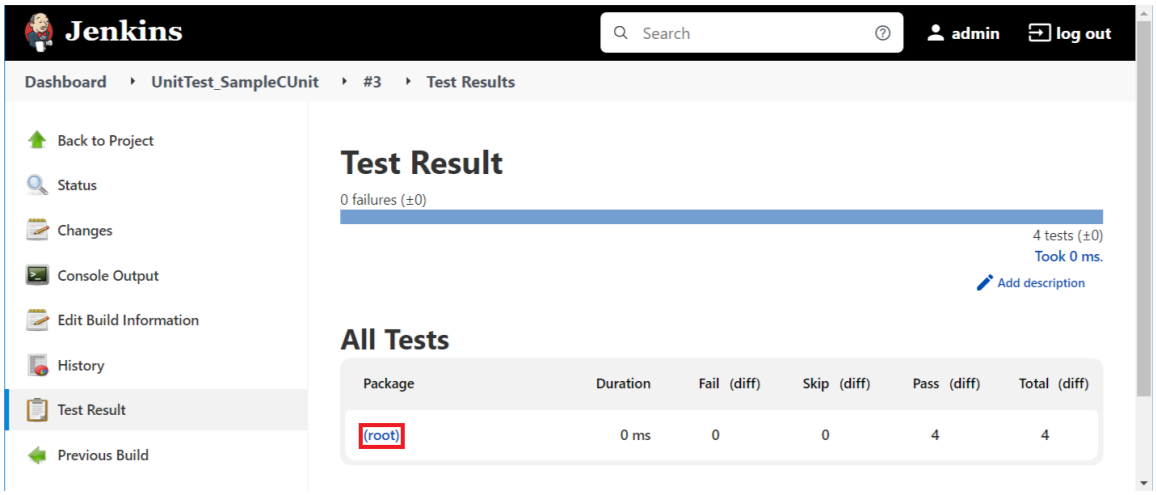


Figure 27

4) The [Test Result: (root)] page appears. The results of the tests defined in the testsource .c file are displayed.

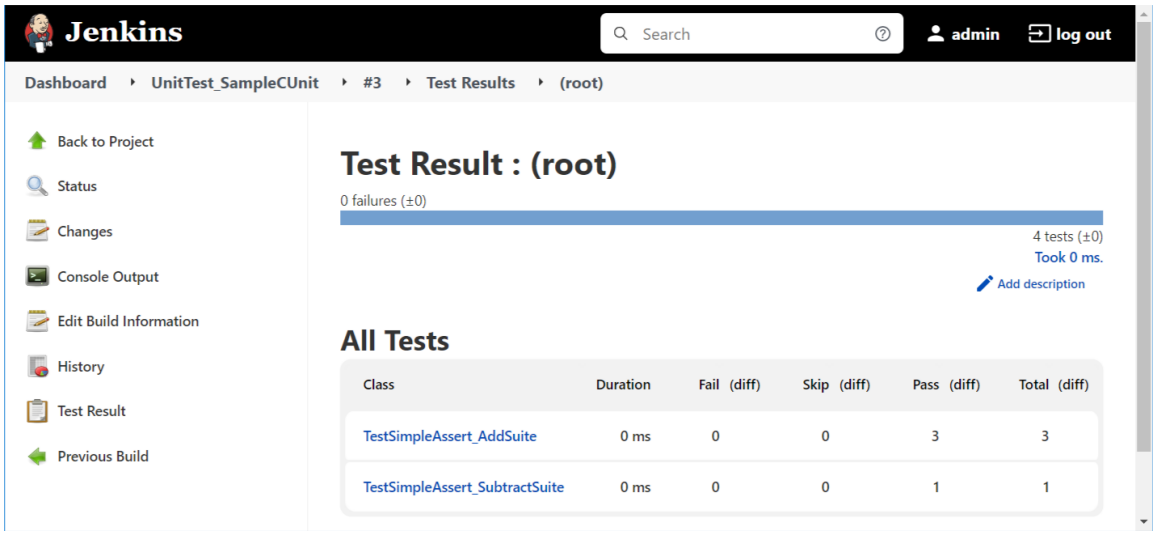
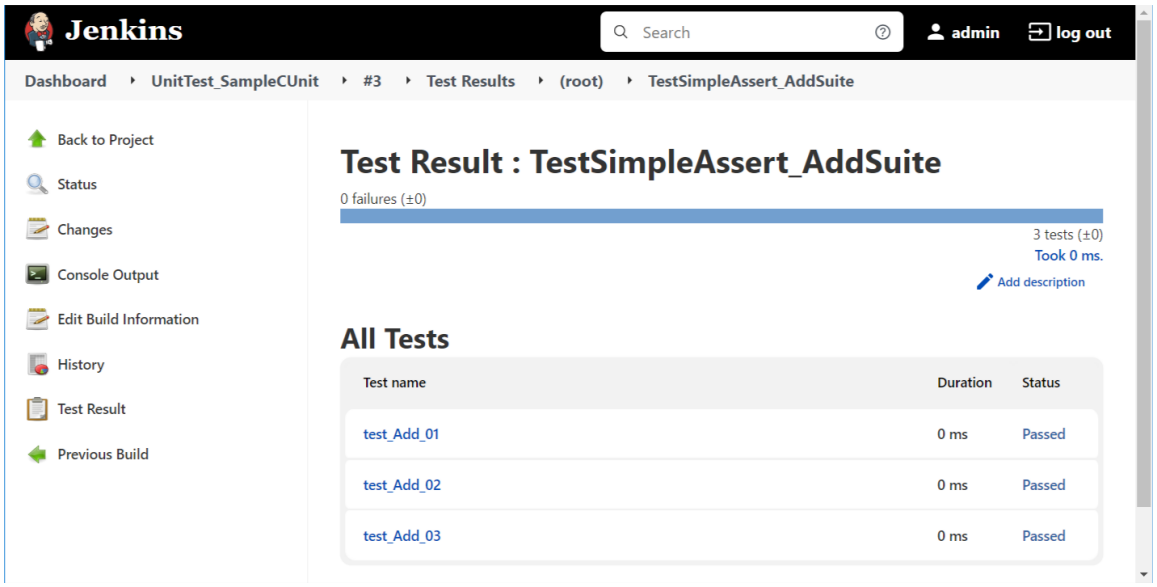


Figure 28

5) Click on each class to see the detailed results of the test case.



7. Website and Support

- e² studio

  https://www.renesas.com/software-tool/e-studio
- Jenkins

  https://jenkins.io/
- Git

  https://git-scm.com
- CUnit

  http://cunit.sourceforge.net/

Revision History

Rev.	Date	Description	
		Page	Summary
1.01	Jun.15.22	All	Reviewed everything based on "How to automate CUnit tests in R20AN0526EE0100e2 Studio and Jenkins".

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.