統合開発環境 e² studio

Jenkins を使用した CUnit 単体テストの自動実行方法

はじめに

このドキュメントでは、Jenkins と Git を使用して、継続的インテグレーションシステムを構築し、プロジェクトのビルド、および単体テスト（CUnit）の実行を自動化する方法を説明します。下記のアプリケーションノートを参照しながら、読み進めていただくことをお勧めいたします。

- アプリケーションノート：「e² studio から EGit を利用する方法」
  統合開発環境 e² studio：e² studio から EGit を利用する方法 (renesas.com)
- アプリケーションノート：「Jenkins を使用して e² studio のプロジェクトをビルドする方法」
  Jenkins を使用して e² studio のプロジェクトをビルドする方法 (renesas.com)
- アプリケーションノート：「e² studio での CUnit の使用方法」
  統合開発環境 e² studio：e² studio での CUnit の使用方法 (renesas.com)

目次

1. 概要 3
1.1 背景 3
1.2 目的 3
1.3 動作確認環境 4

2. プロジェクト作成 5
2.1 プロジェクトを作成する 5
2.2 自動テスト用にプロジェクトを修正する 5

3. Git リポジトリ作成 8
3.1 リモートリポジトリを作成する 8
3.2 ローカルリポジトリを作成する 8
3.3 プロジェクトのバージョン管理を開始する 10

4. Jekins セットアップ 11
4.1 Git プラグインをインストールする 11
4.2 xUnit プラグインをインストールする 11
4.3 環境変数を設定する 12

5. Jenkins ジョブ 13
5.1 ビルドジョブを作成する 13
5.2 テストジョブを作成する 15

6. 動作確認 19
6.1 新しいテストを追加する 19
6.2 テスト結果を確認する 20

7.　参照情報 23
7.1　Web サイト 23

改訂記録 24

1. 概要

1.1 背景

e² studio は、Eclipse をベースとした統合開発環境です。様々なオープンソースソフトウェアのプラグインを組み込んで、機能を追加／拡張することができます。

- Jenkins

  Jenkins は、継続的インテグレーション（CI）、および継続的デリバリー（CD）を実現するためのツールです。「ソフトウェアのリリーススピードの向上」、「開発プロセスの自動化」、「開発コストの削減」といった利用目的に適しています。CI/CD ツールはたくさんありますが、その中でも Jenkins は、汎用性が高いことからユーザーから大きな支持を得ています。ビルドやテストに限らずどんなスクリプトでも実行できるので、ビルドプロセスに合わせてビルドからリリースまで柔軟に自動化を行うことができます。

  アプリケーションノート「Jenkins を使用して e² studio のプロジェクトをビルドする方法 (renesas.com)」では、Jenkins のインストールと、ソースファイルのコミットに連動したビルド実行のセットアップ方法について説明しています。

- Git

  Git は、プログラムのソースコードなどの変更履歴を記録、追跡するための分散型バージョン管理システムです。EGit は、Eclipse 上でバージョン管理システム Git を使用するためのプラグインです。

  アプリケーションノート「統合開発環境 e² studio：e² studio から EGit を利用する方法 (renesas.com)」では、e² studio への EGit のインストールとプロジェクトのバージョン管理方法について説明しています。

- CUnit

  CUnit は、テスト構造の構築にシンプルなフレームワークを使用し、一般的なデータ型をテストするための豊富なアサーションセットを提供します。また、テストの実行と結果のレポート作成のために、いくつかのインターフェースが提供されています。これらには、コード制御のテストおよびレポート用の自動化されたインターフェース、およびユーザーがテストを実行して結果を動的に表示できるインタラクティブなインターフェースが含まれます。

  アプリケーションノート「統合開発環境 e² studio：e² studio での CUnit の使用方法 (renesas.com)」では、e² studio を使用してユーザーのアプリケーションコードをテストするためのソース記述方法とその手順について説明しています。

1.2 目的

このドキュメントでは、クライアントマシン上でローカルに実行される Git リポジトリと Jenkins サーバーを作成し、CUnit 単体テストフレームワークを使用して完全なエンドツーエンドの継続的インテグレーションシステムを構築するために必要な手順を詳しく説明します。

アプリケーションノート「統合開発環境 e² studio：e² studio での CUnit の使用方法 (renesas.com)」で説明されている SampleCUnit プロジェクトを例として使用し、下表に記載されている 3～5 の作業を自動化します。

1		Author application and CUnit test	e² studio で、テスト対象プロジェクトを作成して、テストを追加します。
2		Push change	e² studio で EGit を使用して、プロジェクトのバージョン管理を行います。
3		Trigger	Jenkins のビルドジョブで、プロジェクトの Git リポジトリをポーリングしてプッシュをチェックします。
4		Build	Jenkin は、プロジェクトの Git リポジトリのプッシュを検出するとビルドジョブを実行します。
5		Test	Jenkins は、ビルドジョブが正常終了すると、テストジョブを実行します。
6		Result report	Jenkins UI で、テストジョブの結果を確認することができます。

1.3　動作確認環境

このドキュメントで説明する操作手順については、弊社にて以下の環境で確認を実施しています。ただし、オープンソースのソフトウェアとの連携になりますので、弊社が動作を保証するものではありません。あらかじめご了解の程お願い申し上げます。

［OS］

- OS　　Windows10（日本語版）

［ツール］

- e² studio　　2022-01
  - EGit　　6.1.0.202203080745-r
- Jenkins　　2.332.1
  - xUnit　　3.0.6
  - Git plugin　　4.11.0
- Git　　2.35.1
- CUnit　　2.1.2

［プロジェクト］

例として以下のデバイス、およびツールチェーンを指定したプロジェクトを使用します。

- デバイス　　RX610
- ツールチェーン　　GCC for Renesas RX C/C++ Toolchain v 8.3.0.202104-GNURX-ELF

このドキュメントでは、e² studio（EGit）、Jenkins（Git）、Git、CUnit のインストール方法について説明しません。それらのツールについては、事前に準備が整っているものとして説明を進めます。

2. プロジェクト作成

この章では、対象となるプロジェクトの作成手順について説明します。

2.1 プロジェクトを作成する

アプリケーションノート「統合開発環境 e² studio：e² studio での CUnit の使用方法 (renesas.com)」の手順に従って、CUnit ライブラリプロジェクト「CUnit」とテスト対象プロジェクト「SampleCUnit」を作成してください。

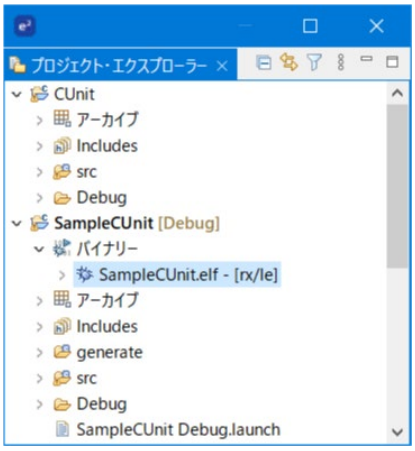

図 1　プロジェクト構成

2.2 自動テスト用にプロジェクトを修正する

Jenkins でプロジェクトをビルドするには、プロジェクトを自己完結型にしなければなりません。そのため、CUnit を静的ライブラリとしてリンクする必要があります。また、Jenkins でプロジェクトの単体テストを実行するには、テスト結果をコンソール画面に出力するのではなくファイルに出力するよう変更する必要があります。

1) CUnit プロジェクトの「Headers」フォルダーを SampleCUnit プロジェクトへ下記手順でコピーします。
   a. ［プロジェクト・エクスプローラー］ビューで「（CUnit > src >）Headers」を選択して、「Ctrl」キーを押しながら、「SampleCUnit」へドラッグ&ドロップします。
   b. ［ファイルおよびフォルダーの操作］ダイアログが表示されます。［ファイルおよびフォルダーをコピー(C)］チェックボックスをチェックして、［OK］ボタンをクリックします。
2) CUnit プロジェクトの「libCUnit.a」ファイルを SampleCUnit プロジェクトへ以下の手順でコピーします。
   a. ［プロジェクト・エクスプローラー］ビューで「SampleCUnit」を選択してコンテキスト・メニュー［新規(N)］>［フォルダー］を選択します。
   b. ［新規フォルダー］ダイアログが表示されます。［フォルダ名(N):］編集ボックスに「lib」を入力して、［終了(F)］ボタンをクリックします。
   c. ［プロジェクト・エクスプローラー］ビューで「（CUnit > アーカイブ >）libCUnit.a」を選択して、「Ctrl」キーを押しながら、「（SampleCUnit >）lib」へドラッグ&ドロップします。
   d. ［ファイル操作］ダイアログが表示されます。［ファイルをコピー(C)］チェックボックスをチェックして、［OK］ボタンをクリックします。

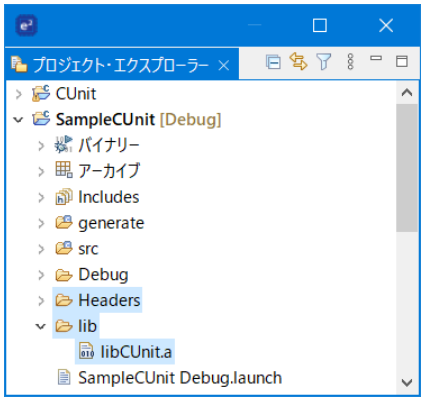

図 2　「Headers」フォルダーと「libCUnit.a」ライブラリ

3) ［プロジェクト・エクスプローラー］ビューで SampleCUnit プロジェクトを選択して、メニュー［プロジェクト(P)］>［C/C++ Project Settings］を選択します。

4) ［プロパティ: SampleCUnit］ダイアログが表示されます。［ツール設定］タブのツリーで「Compiler > Includes」を選択します。［Include file directories］リストボックスの「${workspace_loc:/CUnit/Headers}」を「${workspace_loc/${ProjName}/Headers}」に置き換えます。

5) ［ツール設定］タブのツリーで「Linker > Source」を選択します。［Additional input files］リストボックスの「${workspace_loc:/CUnit/Debug/libCUnit.a}」を「${workspace_loc:/${ProjName}/lib/libCUnit.a}」に置き換えます。［適用して閉じる］ボタンをクリックします。

6) CUnit 自動モードを設定するために、「SampleCUnit.c」ファイルを下記内容に書き換えます。

```
#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) ［プロジェクト・エクスプローラー］ビューで SampleCUnit プロジェクトを選択して、コンテキスト・メニュー［プロジェクトのビルド(B)］を選択します。ビルドが実行されますので、エラーが発生していないことを確認します。

8) ［プロジェクト・エクスプローラー］ビューで「SampleCUnit > バイナリー > SmapleCUnit.elf」を選択して、コンテキスト・メニュー［Show in Local Terminal］>［Terminal］を選択します。

9) ［ターミナル］ビューが表示されます。カーソルが表示されていますので、「rx-elf-run SampleCUnit.elf」を入力して、「Enter」キーを押します。

   CUnit を自動モードに変更したため、テスト結果は、［ターミナル］ビューに表示されません。結果を保存したファイルとして、「Debug/CUnitAutomated-Results.xml」ファイルが生成されます。

3. Git リポジトリ作成

この章では、リモートリポジトリ、およびローカルリポジトリの作成手順について説明します。

3.1 リモートリポジトリを作成する

1) 「Git」パースペクティブを選択します。［Git Repositories］ビューで、ツールバーの［Create a new Git Repositories and add it to this view］ボタンをクリックします。

2) ［Create a Git Repositories］ダイアログが表示されます。［Repository directory:］編集ボックスにリモートリポジトリのフォルダーを入力します。例として、「¥¥192.168.0.43¥Shared¥git¥remote¥cunit」を入力します。

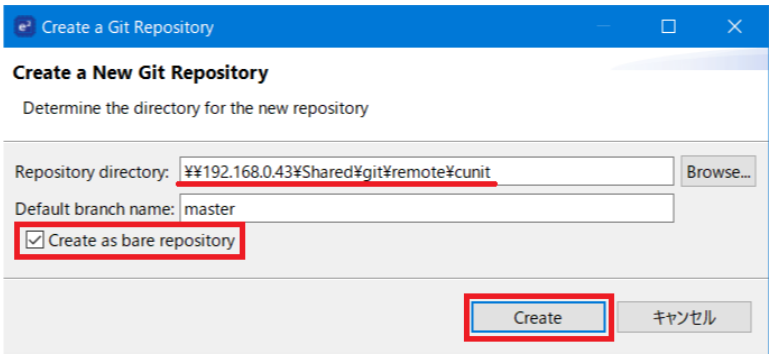

図 3　［Create a new Git Repository］ダイアログ

3) ［Create as bare repository］チェックボックスをチェックして、［Create］ボタンをクリックします。

［Create as bare repository］をチェックすると、作成されたリモートリポジトリには、作業ディレクトリがありません。ローカルリポジトリから変更をプッシュすることによってのみコンテンツを追加できます。

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

リモートリポジトリのコンテンツは直接操作されることがないため（常にローカルリポジトリを経由してコンテンツは操作されます）、［Git Repositories］ビューから取り除く（削除ではない）ことができます。

4) ［Git Repositories］ビューで、「cunit」リポジトリを選択し、［Delete］キーを押します。［Delete Repository］メッセージが表示されますので、［Remove from View］ボタンをクリックします。

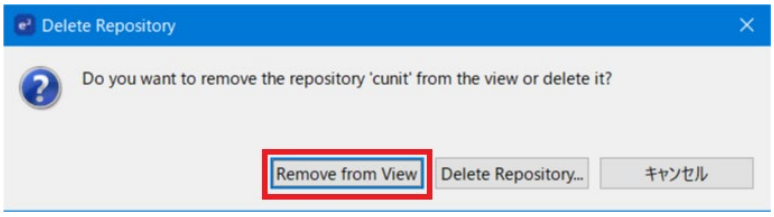

図 4　［Delete Repository］メッセージ

3.2 ローカルリポジトリを作成する

1) 「Git」パースペクティブを選択します。［Git Repositories］ビューで、ツールバーの［Clone a Git Repository and add the clone to this view］ボタンをクリックします。［Clone Git Repository - Source Git Repository］ダイアログが表示されます。［Repository path:］編集ボックスにリモートリポジトリを入力し、［Protocol:］コンボボックスで「file」を選択して、［次へ(N) >］ボタンをクリックします。

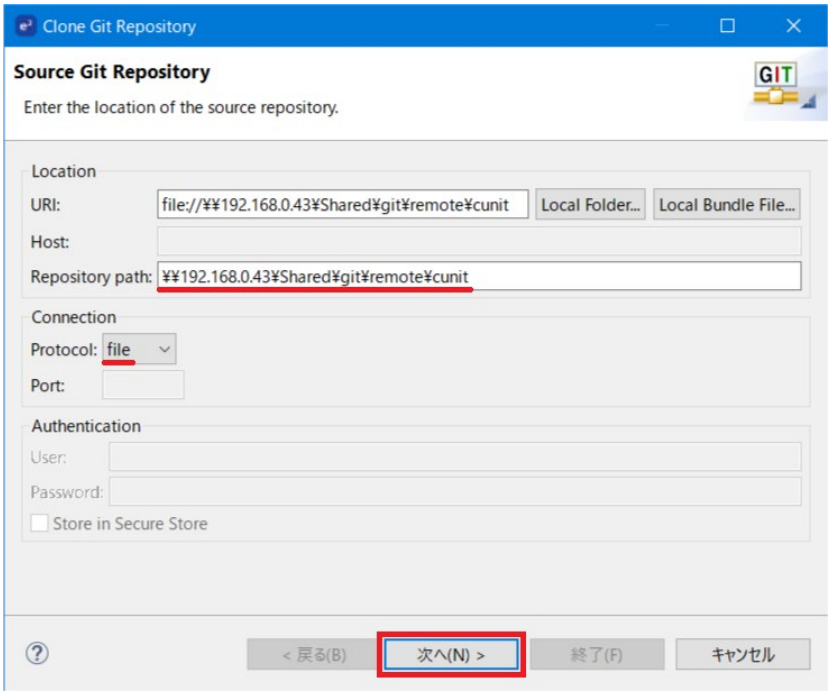

図 5　［Clone Git Repository - Source Git Repository］ダイアログ

2) ［Clone Git Repository - Branch Selection］ダイアログが表示されますので、［次へ(N) >］ボタンをクリックします。

3) ［Clone Git Repository - Local Destination］ダイアログが表示されます。［Directory:］編集ボックスに、ローカルリポジトリとしてローカルホストのフォルダーを指定します。例として「C:¥git¥local¥cunit」を入力して、［終了(F)］ボタンをクリックします。

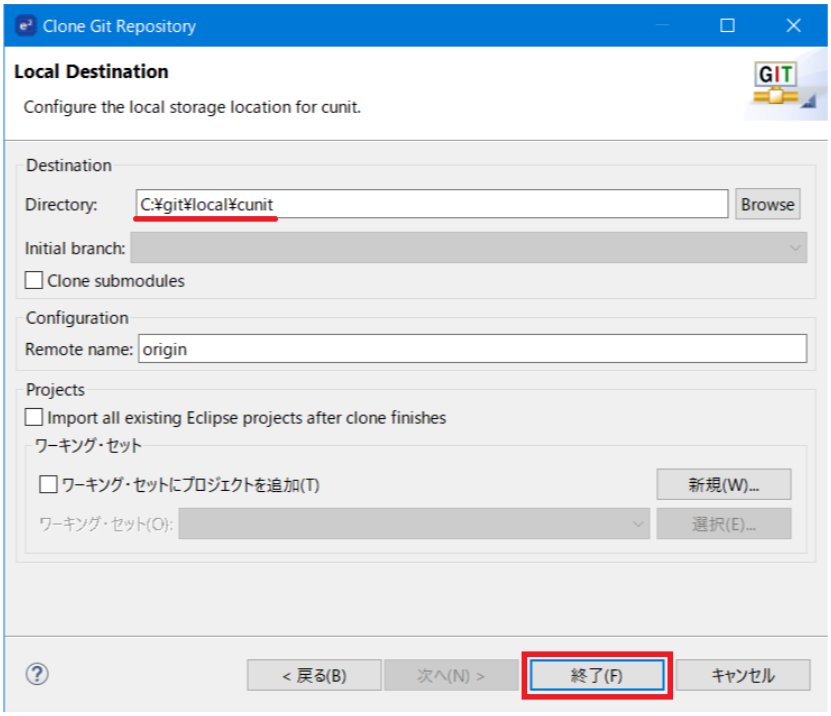

図 6　［Clone Git Repository - Local Destination］ダイアログ

3.3 プロジェクトのバージョン管理を開始する

1) 「C/C++」パースペクティブを選択します。［プロジェクト・エクスプローラー］ビューで「SampleCUnit」プロジェクトを選択して、コンテキスト・メニュー［Team］>［プロジェクトの共用(S)…］を選択します。

2) ［Share Project］ダイアログが表示されます。［Repository:］コンボボックスでローカルリポジトリを選択し、［終了(F)］ボタンをクリックします。

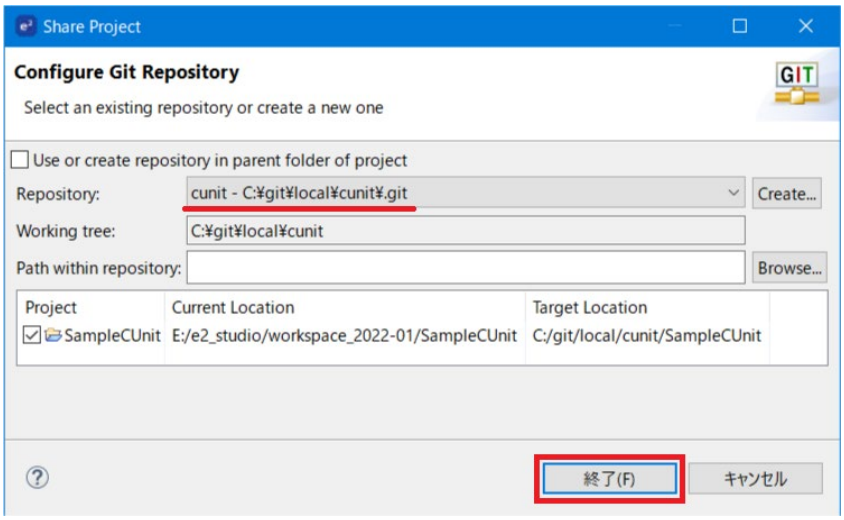

図 7　［Share Project - Configure Git Repository］ダイアログ

3) 「Git」パースペクティブを選択します。［Git Repositories］ビューで「cunit」を選択します。

4) ［Git Staging］ビューで、プロジェクト内の新しいファイルが［Unstaging Changes］リストボックスに表示されます。

5) ［Unstaging Changed］の［Add all files including not selected ones to the index］ボタンをクリックします。［Unstaging Changed］リストボックスのアイテムが、［Staged Changed］リストボックスに移動します。次に、［Commit Message］編集ボックスにメッセージを入力して（例：Initial）、［Commit and Push…］ボタンをクリックします。

6) ［Push Results］ダイアログが表示されます。メッセージを確認して、［閉じる(C)］ボタンをクリックします。

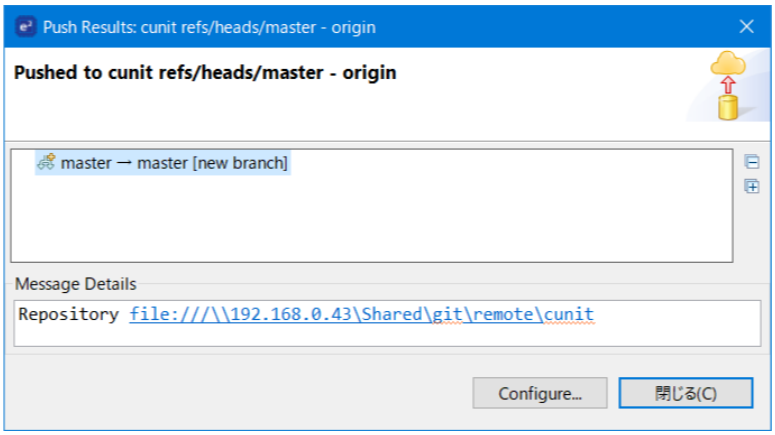

図 8　［Push Results］ダイアログ

SampleCUnit プロジェクトが、ローカルリポジトリにコミットされ、リモートリポジトリにプッシュされました。

4. Jekins セットアップ

この章では、Jenkins に Git プラグイン、および xUnit プラグインをインストールする方法について説明します。

4.1 Git プラグインをインストールする

Jenkin に Git を認識させるためには、Jenkin に Git プラグインをインストールする必要があります。アプリケーションノート「Jenkins を使用して e² studio のプロジェクトをビルドする方法 (renesas.com)」を参考に、Git ツール、および Git プラグインをインストールしてください。

- Git ツール
- Git プラグイン

4.2 xUnit プラグインをインストールする

xUnit プラグインを使用すると、Jenkins でテストツールを実行したテスト結果を共有できます。CUnit は自動モードでテストレポートを作成し、xUnit はそれを Jenkins 内で他メンバと結果を共有するのに役立ちます。

xUnit プラグインをインストールするには、次の手順に従ってください。

1) ［Jenkins］ページで［Jenkins の管理］をクリックし、次に［プラグインの管理］をクリックします。

2) ［Plugin Manager］ページが表示されます。［利用可能］タブをクリックして、フィルターに「xUnit」を入力します。ページ内容が、「xUnit」でフィルタリングされます。

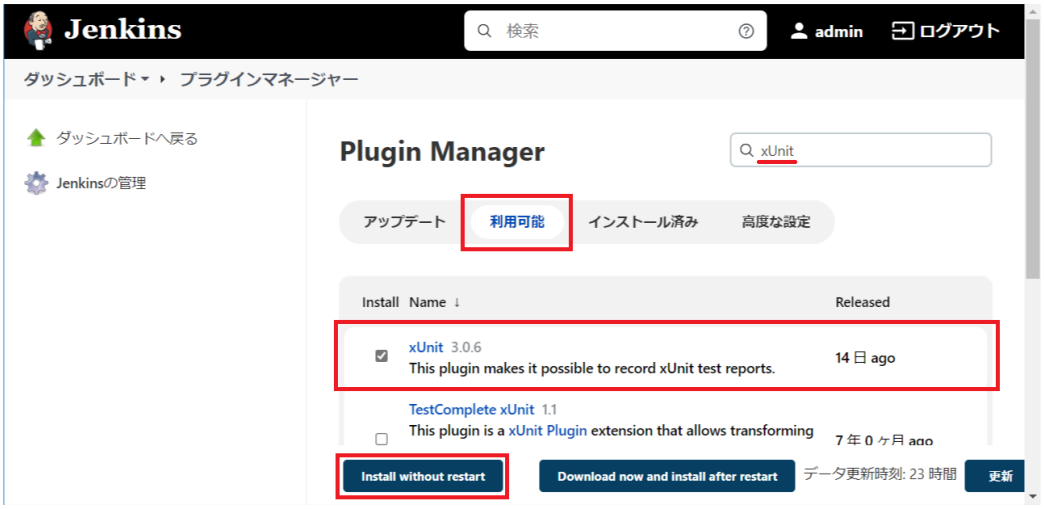

図 9　［プラグインマネージャー］ページ

3) ［xUnit］をチェックして、［Install without restart］ボタンをクリックします。インストールが開始されます。

4) インストールが完了したら、［インストール完了後、ジョブがなければ Jenkins を再起動する］をチェックします。Jenkins が自動的に再起動します。

5) Jenkins にログインします。

これで Jenkins から xUnit が使用可能になりました。

4.3　環境変数を設定する

後ほど作成するテストジョブのコマンドが正しく検索されて実行されるようにするために環境変数を設定する必要があります。

次の手順に従って、環境変数 Path を設定してください。

1)　［Jenkins］ページで［Jenkins の管理］をクリックし、次に［システムの設定を］をクリックします。

図 10　［Jenkins の管理］ページ

2)　［設定］ページが表示されます。スクロールして［グローバル プロパティ］を表示し、［環境変数］をチェックします。以下の通りに、［キー］と［値］を指定します。
［キー］Path
［値］< GCC for Renesas RX C/C++ Toolchain のインストール・ディレクトリ>¥rx-elf¥rx-elf¥bin;$Path

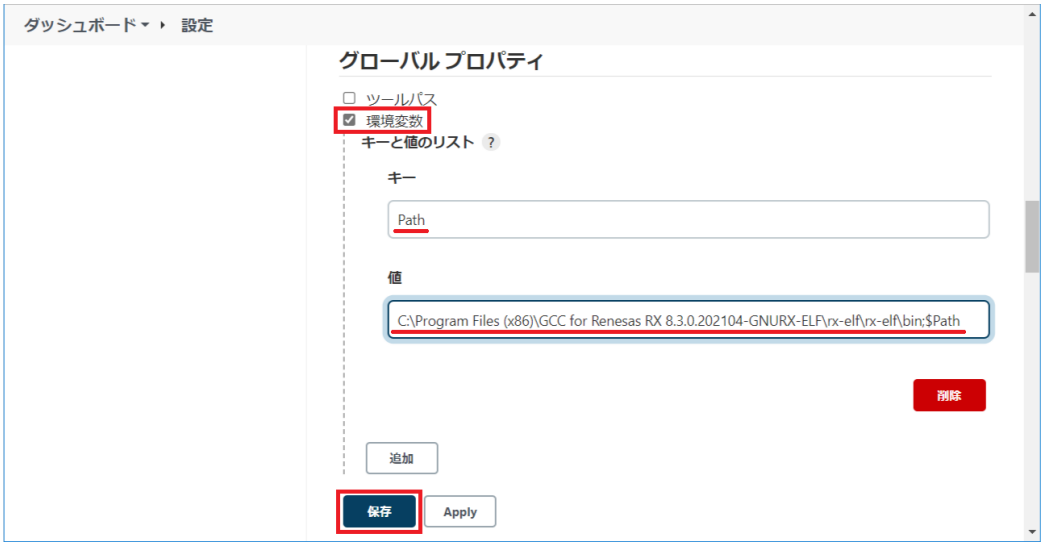

図 11　［グローバル プロパティ］

3)　［保存］ボタンをクリックします。保存されると、［ダッシュボード］ページが表示されます。

5. Jenkins ジョブ

この章では、Jenkins のジョブの作成方法について説明します。

5.1 ビルドジョブを作成する

1) ［Jenkins］ページで、［新規ジョブ作成］をクリックします。

2) ［ジョブ名入力］編集ボックスに「Build_SampleCUnit」を入力して、［フリースタイル・プロジェクトのビルド］をクリックします。次に［OK］ボタンをクリックします。

図 12　新規ジョブ作成（ビルドジョブ）

3) ジョブの詳細設定を行うための画面が表示されます。

4) ［ソースコード管理］タブをクリックします。［ソースコード管理］が表示されますので、［Git］をチェックし、［リポジトリ URL］編集ボックスにリモートリポジトリを指定します。必要に応じて、認証情報を追加してください。

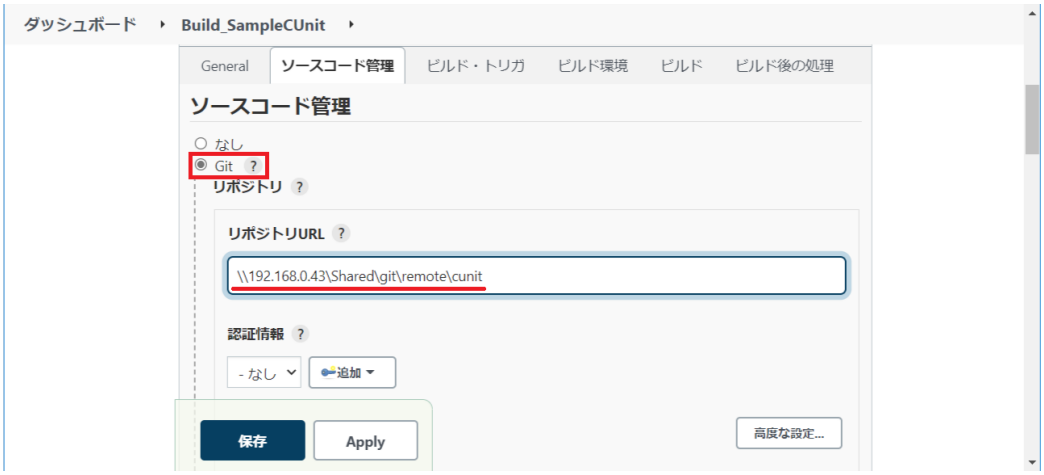

図 13　［ソースコード管理］ページ

※ ビルドジョブにおいて、下記エラーが発生する場合があります。

```
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
```

Jenkins のセキュリティ強化対策により、Git プラグインのローカル URL またはパスにあるリポジトリへのアクセスが許可されていません。（Git | Jenkins plugin）

アクセスを許可するには、Jenkins インストール・フォルダにある「jenkins.xml」に“-Dhudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT=true”を追加してください。

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

5) ［ビルド・トリガ］タブをクリックします。ビルドの実行タイミングを設定します。プロジェクトが Git リポジトリで更新されるたびにビルドするには、［SCM をポーリング］をチェックします。これにより、Git リポジトリの更新が定期的にチェックされ、更新がある場合はビルドが実行されます。

6) Git リポジトリの更新を確認する間隔は、cron 風の構文で指定します。ここでは、「H/15 * * * *」を入力します。これは、15 分ごとに Git リポジトリをチェックすることを意味します。詳細については、［スケジュール］編集ボックス右横の［?］ボタンをクリックしてください。

図 14　［ビルド・トリガ］ページ

7) ［ビルド］タブをクリックします。［ビルド手順の追加］コンボボックスで、「Windows バッチコマンドの実行」を選択します。

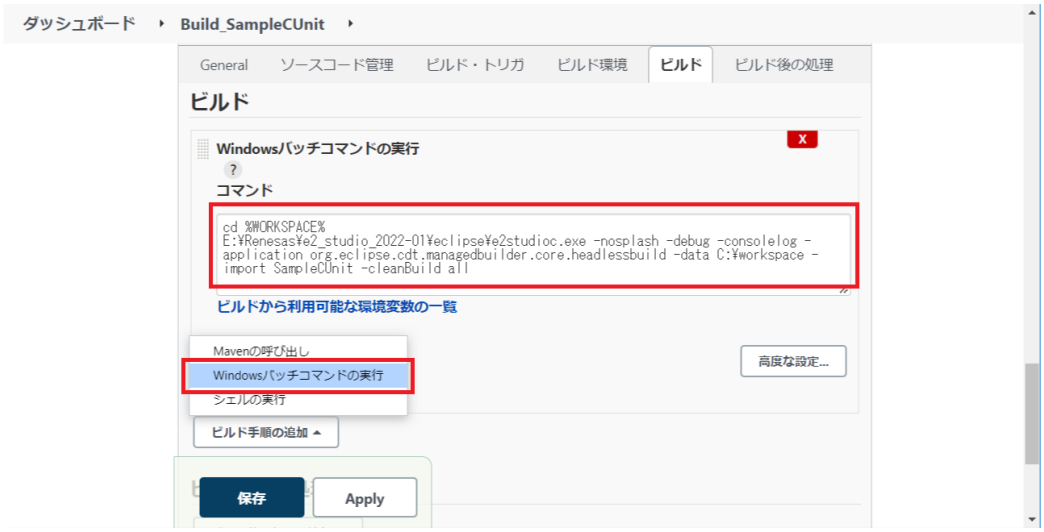

図 15　［ビルド］ページ

8) ［コマンド］編集ボックスに、e² studio のヘッドレス・ビルド実行のための下記コマンドを入力します。

ヘッドレス・ビルドは、e² studio UI を使用せずにコマンド・ラインでプロジェクトをビルドすることです。コマンド・ライン・オプションの詳細は、e² studio ヘッドレス・ビルド機能のヘルプをご参照ください。

```
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>：e² studio のインストール・ディレクトリを指定してください。
- e2studioc.exe：e² studio 2021-10 およびそれ以前のバージョンでは、eclipsec.exe を指定してください。
- <workspace folder>：ヘッドレス・ビルドでは、e² studio UI は使用しませんが、通常と同じようにワークスペースの指定が必要になります。使用するプロジェクトが含まれていないワークスペース・フォルダを指定してください。
- <project_name>：Git リポジトリに登録されているプロジェクト名を指定してください。ここでは、「SampleCUnit」を指定します。

9) ［保存］ボタンをクリックします。保存されると、「プロジェクト Build_SampleCUnit」ページが表示されます。

5.2 テストジョブを作成する

1) ［Jenkins］ページで、［新規ジョブ作成］をクリックします。

2) ［ジョブ名入力］編集ボックスに「UnitTest_SampleCUnit」を入力して、［フリースタイル・プロジェクトのビルド］をクリックします。次に［OK］ボタンをクリックします。

図 16　新規ジョブ作成（テストジョブ）

3) ジョブの詳細設定を行う画面が表示されます。

4) ［ビルド・トリガ］タブをクリックします。［ビルド・トリガ］が表示されますので、［他プロジェクトの後にビルド］をチェックして、［対象プロジェクト］編集ボックスに「Build_SampleCUnit」を入力します。

図 17　［ビルド・トリガ］ページ

5) ［ビルド］タブをクリックします。［ビルド］が表示されますので、［ビルド手順の追加］コンボボックスで、「Windows バッチコマンドの実行」を選択します。

図 18　［ビルド］ページ

6) ［コマンド］編集ボックスに、下記コマンドを入力して、［保存ボタン］をクリックします。

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

7) ［ビルド後の処理］タブをクリックします。［ビルド後の処理］画面が表示されますので、［ビルド後の処理の追加］コンボボックスで「Publish xUnit test result report」を選択します。

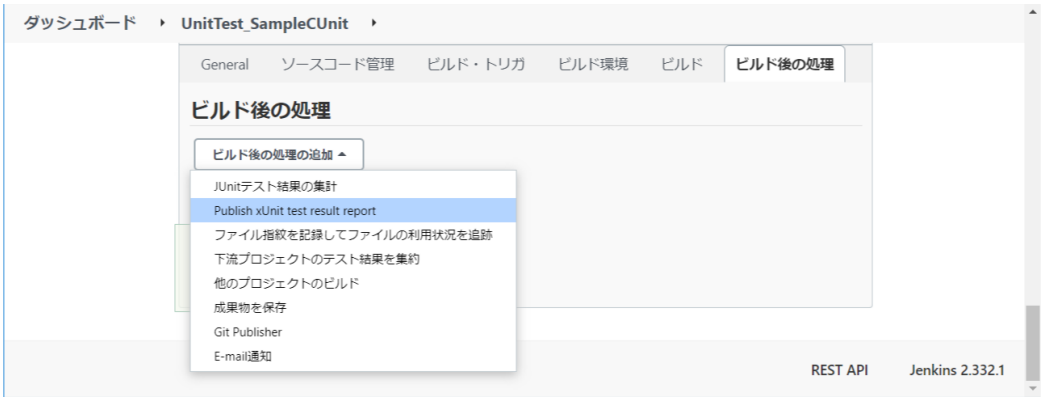

図 19　ビルド後の処理の追加

8)　次に、［Report Type］の［追加］コンボボックスで「CUnit-2.1 (default)」を選択します。

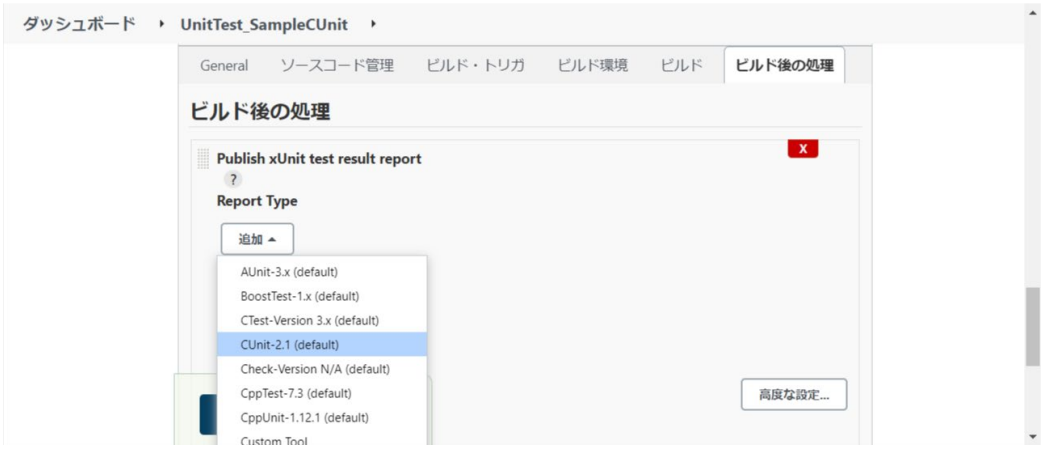

図 20　Report Type

9)　［Includes Pattern］編集ボックスに「*.xml」を入力します。

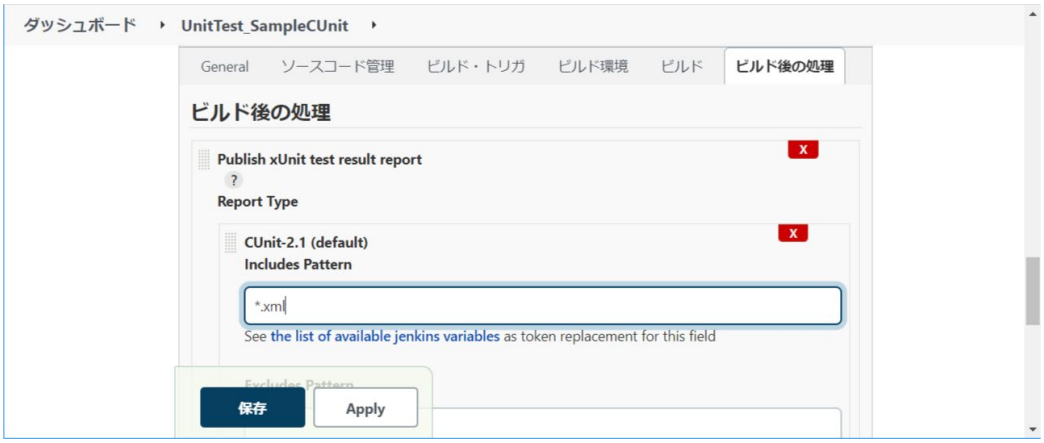

図 21［ビルド後の処理］ページ（Pattern 編集）

10)［Thresholds］の［追加］コンボボックスで「Failed Tests」を選択します。［Failed Tests］グループボックスが表示されます。次に、［追加］コンボボックスで「Skipped Tests」を選択します。［Skipped Tests］グループボックスが表示されます。［高度な設定］ボタンをクリックします。［Choose your threshold mode］で［Use a percent of tests］をチェックします。

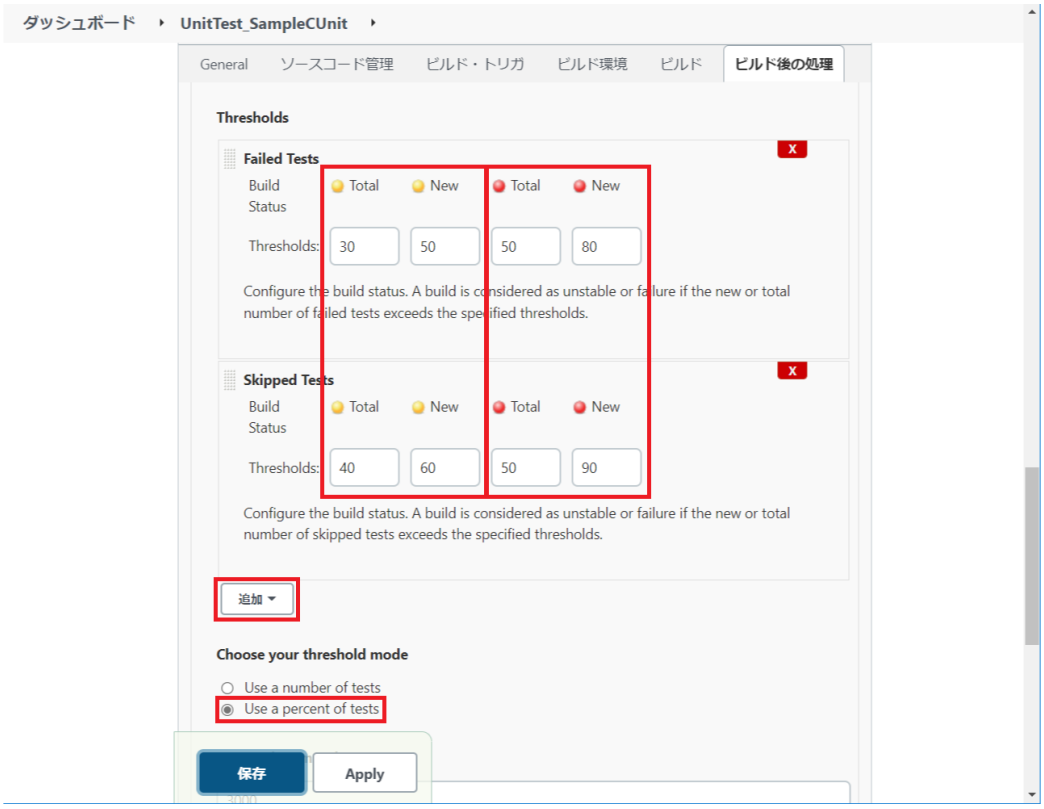

図 22　［ビルド後の処理］ページ（Thresholds 設定）

11) 図に示す閾値を入力します。指定した数値の意味は以下の通りです。

- ジョブが「unstable（  ）」と判断される場合
  - 「Failed Tests」の「Total」テストケースの 30%以上が失敗
  - 「Failed Tests」の「New」テストケースの 50%以上が失敗
  - 「Skipped Tests」の「Total」テストケースの 40%以上をスキップ
  - 「Skipped Tests」の「New」テストケースの 60%以上をスキップ
- ジョブが「failed（  ）」と判断される場合
  - 「Failed Tests」の「Total」テストケースの 50%以上が失敗
  - 「Failed Tests」の「New」テストケースの 80%以上が失敗
  - 「Skipped Tests」の「Total」テストケースの 50%以上をスキップ
  - 「Skipped Tests」の「New」テストケースの 90%以上をスキップ

12) ［保存］ボタンをクリックします。保存が行われると、「プロジェクト Test_SampleCUnit」ページが表示されます。

6. 動作確認

この章では、新しいテストを追加して、テスト結果を確認する方法について説明します。

6.1 新しいテストを追加する

新しいテストを追加して Git リポジトリにコミットすることにより、SampleCUnit プロジェクトを変更します。 Jenkins SCM ポーリングによって変更が確認されると、Build_SampleCUnit ジョブが実行されます。Build_SampleCUnit ジョブが完了すると Test_SampleCUnit ジョブが実行されます。

1) e² studio を起動して、「C/C++」パースペクティブを選択します。

2) ［プロジェクト・エクスプローラー］ビューで「（SampleCUnit > src >）testsource.c」をダブルクリックして、エディタに表示します。

3) 下記の黄色で示された新しいテストコードを追加して、保存します。

- 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) プロジェクトをビルドして、エラーがないことを確認します。

5) 「Git」パースペクティブを選択して、［Git Staging］ビューで、変更されたファイル「testsource.c」を［Unstaged Changes］リストボックスから［Staged Changes］リストボックスに移動します。［Commit Message］編集ボックスにメッセージ（例：Update test case）を追加します。

6) ［Commit and Push…］ボタンをクリックします。

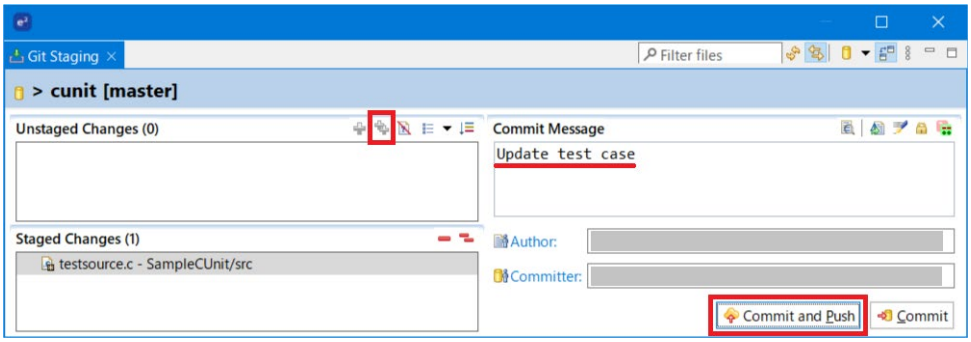

図 23　［Git Staging］ビュー

Git のポーリング期間が終了すると、Jenkins は保留中の変更があることを認識し、Build_SampleCUnit ジョブを実行します。続けて、Test_SampleCUnit ジョブを実行します。テスト結果は、以下に示すように、テスト結果の推移グラフに表示されます。

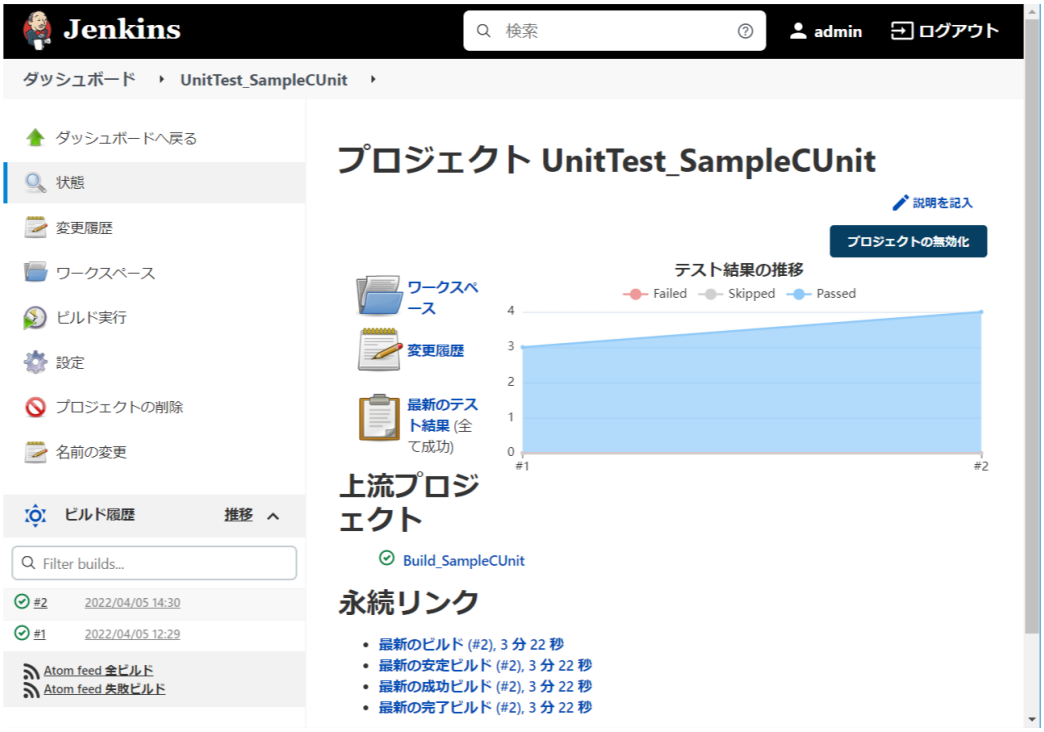

図 24　テスト結果の推移

6.2　テスト結果を確認する

1)　［Jenkins］ページで、Test_SampleCUnit ジョブの「最新の成功ビルド」の#番号をクリックします。

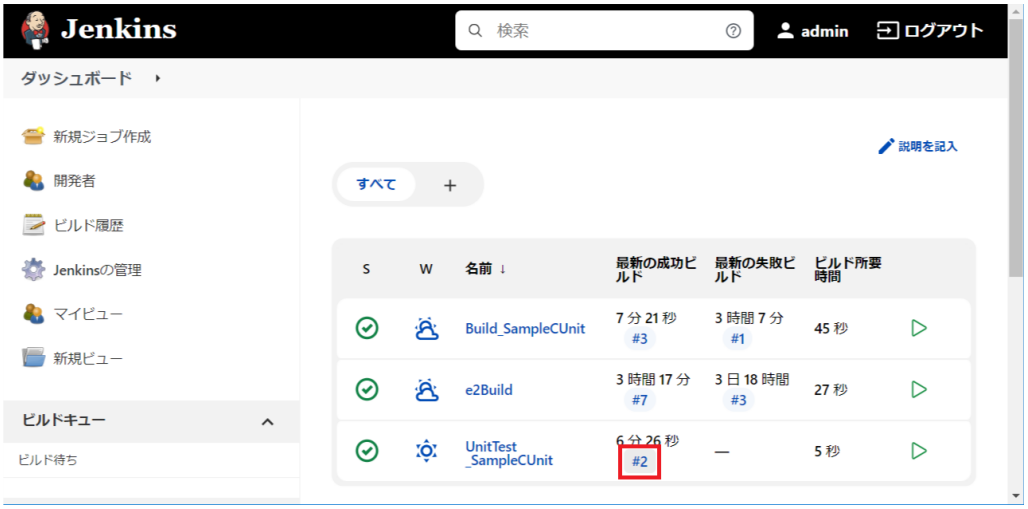

図 25　［Test_SampleCUnit ジョブ］ページ

2)　［ビルド#番号］ページが表示されますので、［テスト結果］をクリックします。

図 26　［ビルド#番号］ページ

3)　［テスト結果］ページが表示されます。パッケージの「(root)」をクリックします。

図 27　［テスト結果］ページ

4)　［テスト結果：(root)］ページが表示されます。testsource.c ファイルで定義したテストの結果が表示されています。

図 28　［テスト結果 : (root)］ページ

5)　各クラスをクリックすると、テストケースの詳細な結果が表示されます。

図 29　テストケースの詳細な結果

7. 参照情報

7.1 Web サイト

- e² studio
  https://www.renesas.com/software-tool/e-studio
- Jenkins
  https://jenkins.io/
- Git
  https://git-scm.com
- CUnit
  http://cunit.sourceforge.net/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.01	Jun.15.22	すべて	「R20AN0526EE0100 How to automate CUnit tests in e² studio and Jenkins」をベースに全面見直し

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。