

統合開発環境 e² studio

e² studio での CUnit の使用方法 (CC-RX)

はじめに

CUnit は C で単体テストを作成、管理、実行するためのシステムです。

このドキュメントでは、e² studio 環境で CUnit を使用した単体テストを実施する方法について説明します。

目次

1. 概要	2
1.1 目的	2
1.2 動作確認環境	2
2. セットアップ	3
2.1 CUnit をインストールする	3
2.2 CUnit ライブラリを作成する	3
3. 使用方法	7
3.1 テスト対象プロジェクトを作成する	7
3.2 CUnit を使用した単体テストを実行する	12
4. 参考情報	14
4.1 Web サイト	14
4.2 他のデバイス、コンパイラ、またはデバッグ環境を使用する場合	14
改訂記録	15

1. 概要

1.1 目的

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。

CUnit は、C で単体テストを記述、管理、および実行するためのシステムです。ユーザーのテストコードにリンクするための静的ライブラリとして構築します。テスト構造を構築するための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。さらに、テスト実行結果のレポート作成のためのインターフェースも提供しています。

このドキュメントでは、e² studio 上で CUnit を使用した単体テストの実施方法について説明します。

1.2 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 2022-04
- 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 CC-RX V3.04

2. セットアップ

この章では、CUnit をインストールしてセットアップする手順について説明します。

[注意事項]

- CUnit-2.1-2 をダウンロードして使用してください。最新版である CUnit-2.1-3 にはいくつかの問題がありビルドエラーが発生する場合があります。
- コンパイラ（および Windows システム）は、「curse」モジュールをサポートしません。

2.1 CUnit をインストールする

CUnit を「<https://sourceforge.net/projects/cunit/files/CUnit/>」からダウンロードします。圧縮ファイルを解凍して、任意のフォルダに CUnit パッケージを展開します。

2.2 CUnit ライブラリを作成する

CUnit は、ユーザーのコードにリンクする静的ライブラリとして構築する必要があります。CUnit のライブラリプロジェクトを作成して、ライブラリを生成します。

- 1) メニュー [ファイル(F)] > [新規(N)] > [Renesas C/C++ Project] > [Renesas RX] を選択します。
- 2) [New C/C++ Project - Templates for New C/C++ Project] ダイアログが表示されます。右側リストボックスで「Renesas CC-RX C/C++ Library Project」を選択して、[次へ(N)>] ボタンをクリックします。

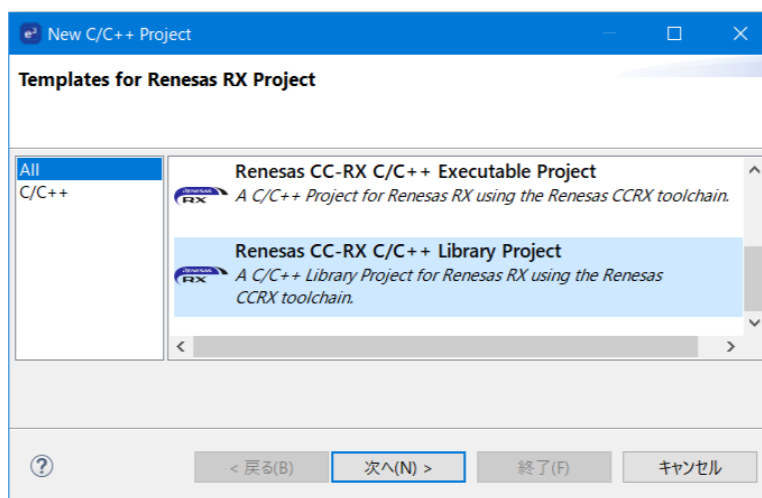


図 1

- 3) [New Renesas CC-RX Library Project - New Renesas CC-RX Library Project] ダイアログが表示されます。[プロジェクト名(P):] 編集ボックスに「CUnit」を入力して、[次へ(N)>] ボタンをクリックします。
- 4) [New Renesas CC-RX Library Project - Select toolchain, device & debug settings] ダイアログが表示されます。下記情報を設定します。残りの箇所は、デフォルトのままとします。設定を終了したら [終了(F)] ボタンをクリックします。
 - [ツールチェーン:] コンボボックス : Renesas CCRX
 - [ツールチェーン・バージョン:] コンボボックス : 任意（このドキュメントでは「v3.04.00」を選択）
 - [ターゲット・デバイス] 編集ボックス : 任意（このドキュメントでは「R5F56107VxFP」を選択 [RX600 > RX610 > RX610 - 144pin > R5F56107VxFP] ）

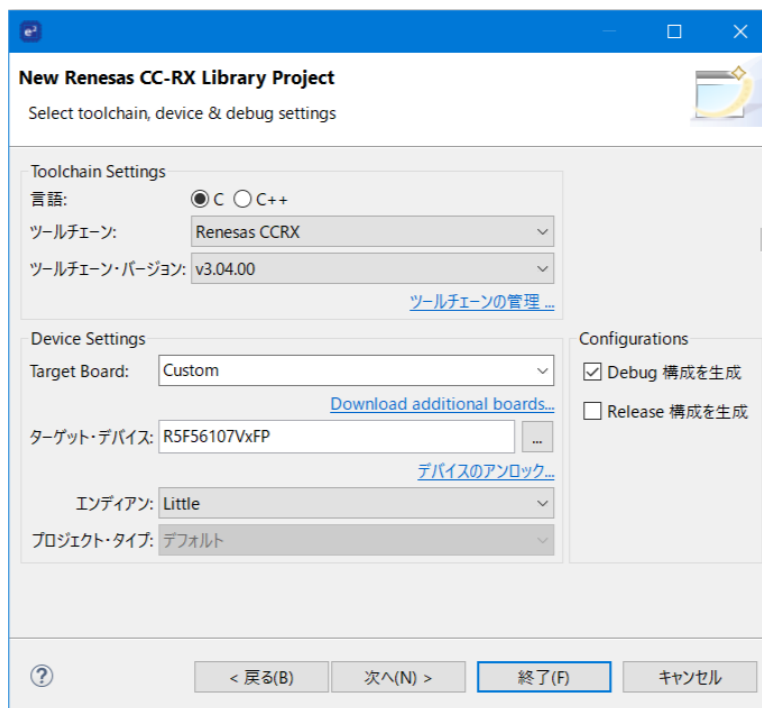


図 2

- 5) Library プロジェクトが作成されて、[プロジェクト・エクスプローラー] ビューに、CUnit プロジェクトが表示されます。デフォルトのソースファイルは必要ないため、「src」フォルダ内のファイル (sample1.c、sample2.c、sample3.S) を削除します。
- 6) CUnit パッケージをインストールしたフォルダの「CUnit」フォルダにある、「¥Headers」、 「¥Sources¥Basic」および「¥Sources¥Freamwork」を CUnit プロジェクトの「src」フォルダへコピーします。

プロジェクトは、次のようになります。

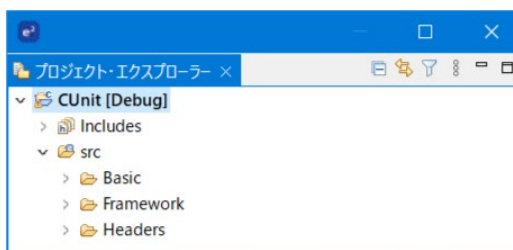


図 3

- 7) [プロジェクト・エクスプローラー] ビューで CUnit プロジェクトを選択して、コンテキスト・メニュー [C/C++ Project Settings Ctrl+Alt+P] を選択します。
- 8) [プロパティ: CUnit] 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 > ソース」を選択します。[インクルード・ファイルを検索するフォルダ (-include)] リストボックスに「\${workspace_loc}/\${ProjName}/src/Headers」を追加します。

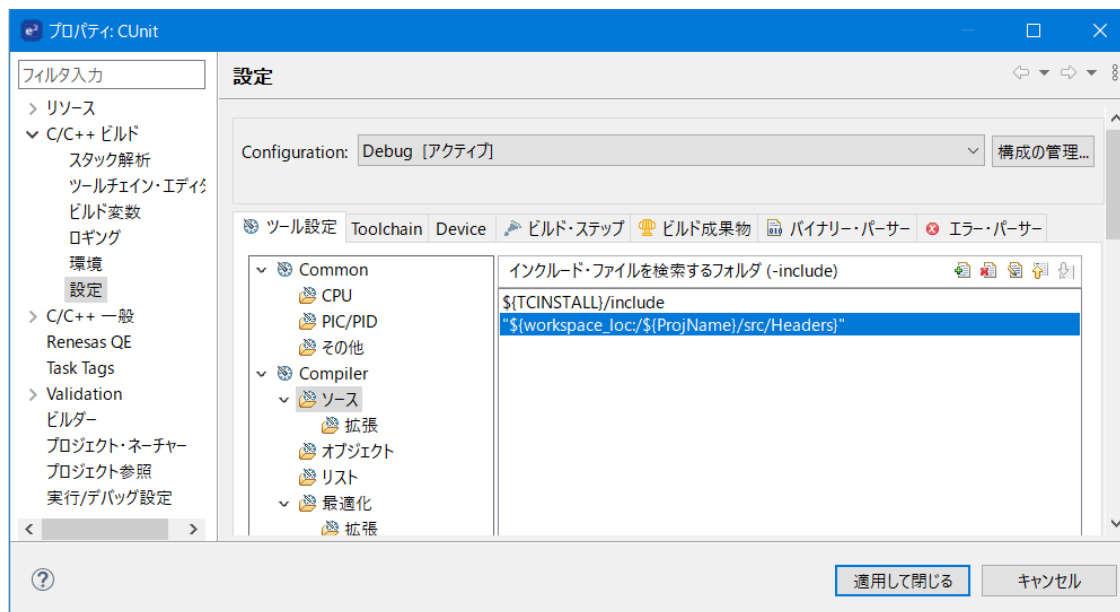


図 4

- 9) [ツール設定] タブのツリーで「Compiler > ソース > 拡張」を選択します。[C ソース (-lang)] コンボボックスで「C99 言語」を選択します。[適用して閉じる] ボタンをクリックします。

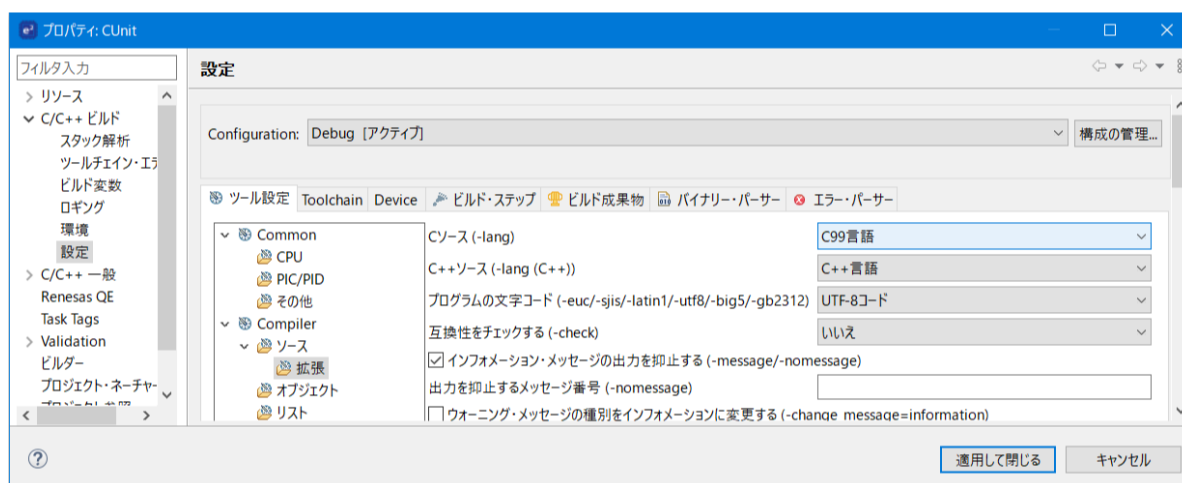


図 5

- 10) 「Headers」フォルダに下記内容のファイルを作成して追加します。

(CC-RX は、time.h をサポートしていませんのでダミーのファイルを作成する必要があります)

- time.h

```
#ifndef TIME_H_
#define TIME_H_

typedef int clock_t;
#define CLOCKS_PER_SEC 1000
#define clock() (0)

#endif
/* TIME_H_ */
```

- 11) [プロジェクト・エクスプローラー] ビューで CUnit 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、「Debug」フォルダに「CUnit.lib」ファイルが表示されます。

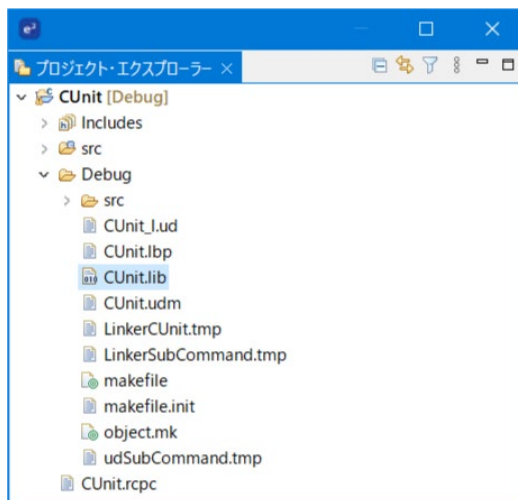


図 6

CUnit ライブラリ・ファイル「CUnit.lib」は、CUnit テストフレームワークを提供します。同一コンパイラのプロジェクトであれば、どの C/C++ プロジェクトでも使用することができます。

3. 使用方法

この章では、CUnit を使用したテストを実行するプロジェクトの作成方法、およびテストの実行方法について説明します。

3.1 テスト対象プロジェクトを作成する

テスト対象プロジェクトは、Renesas CC-RX C/C++ Executable Project として作成します。

- 1) メニュー [ファイル(F)] > [新規(N)] > [Renesas C/C++ Project] > [Renesas RX] を選択します。
- 2) [New C/C++ Project - Templates for Renesas RX Project] ダイアログが表示されます。右側リストボックスで「Renesas CC-RX C/C++ Executable Project」を選択します。[次へ(N)>] ボタンをクリックします。

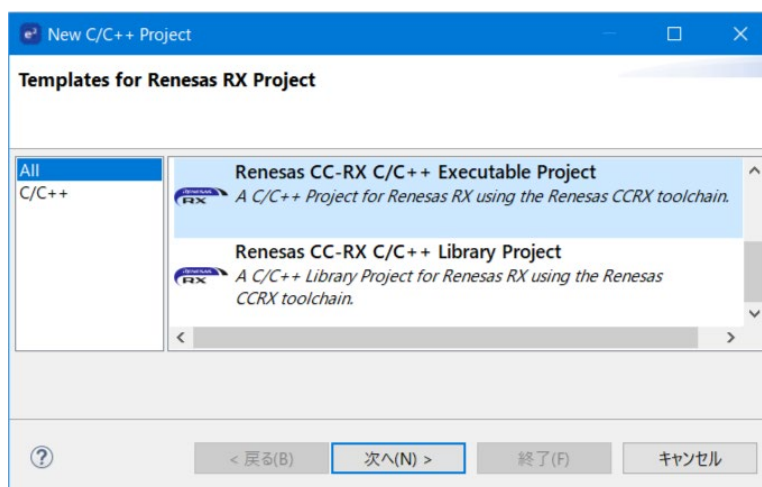


図 7

- 3) [New Renesas CC-RX Executable Project - New Renesas CC-RX Executable Project] ダイアログが表示されます。[プロジェクト名(P):] 編集ボックスに「SampleCUnit」を入力して、[次へ(N) >] ボタンをクリックします。
- 4) [New Renesas CC-RX Executable Project - Select toolchain, device & debug settings] ダイアログが表示されます。下記情報を設定します。残りの箇所は、デフォルトのままとします。設定を終了したら [次へ(N) >] ボタンをクリックします。
 - [ツールチェーン:] コンボボックス : Renesas CCRX
 - [ツールチェーン・バージョン:] コンボボックス : 任意 (このドキュメントでは「v3.04.00」を選択)
 - [ターゲット・デバイス] 編集ボックス : 任意 (このドキュメントでは「R5F56107VxFP」を選択 [RX600 > RX610 > RX610 - 144pin > R5F56107VxFP])
 - [Hardware Debug 構成を生成] チェックボックスのチェックをはずす。
 - [Debug 構成を生成] チェックボックスをチェックし、「RX Simulator」を選択する。

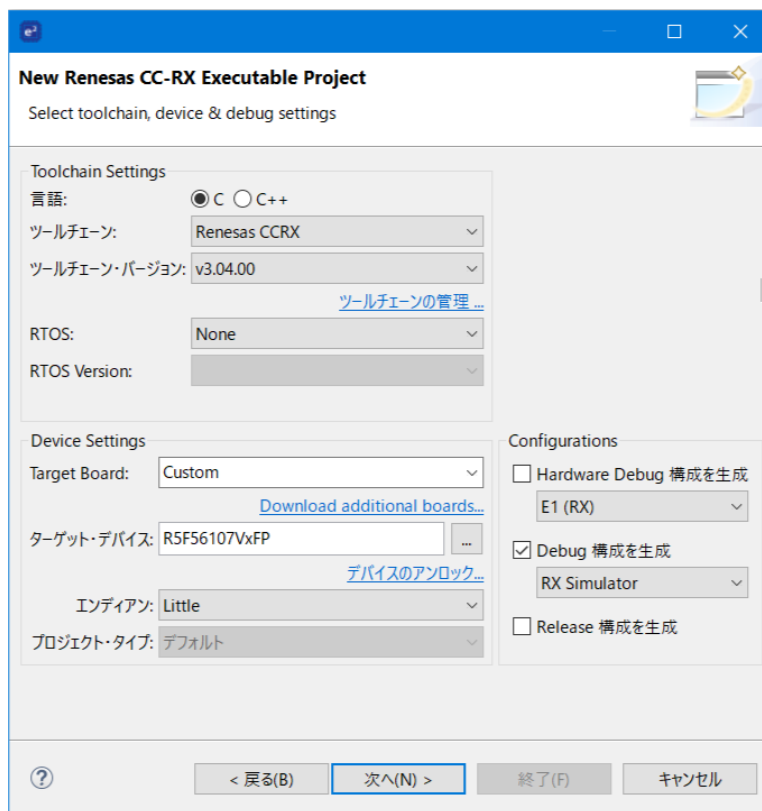


図 8

- 5) [New Renesas CC-RX Executable Project - コーディング・アシストツールの選択] 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。
- 6) [New Renesas CC-RX Executable Project - Settings The Contents of Files to be Generated] ダイアログが表示されます。[Renesas デバッグ仮想コンソールを使用する] チェックボックスをチェックして、[終了(F)] ボタンをクリックします。

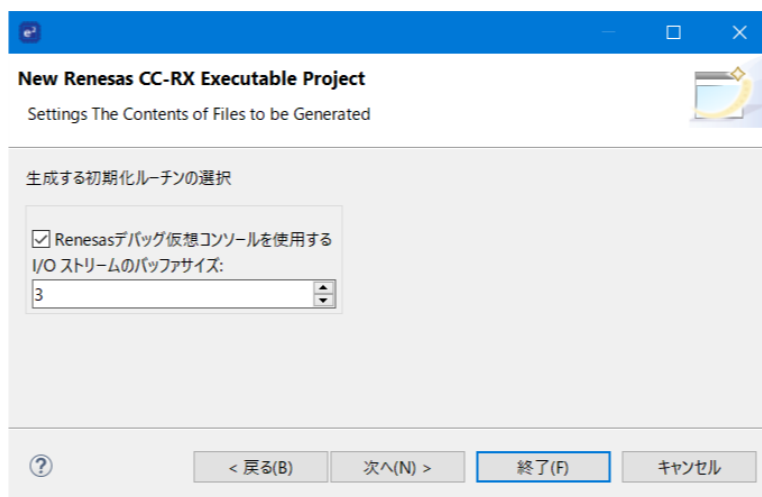


図 9

Executable プロジェクトが作成され、[プロジェクト・エクスプローラー] ビューに SampleCUnit プロジェクトが表示されます。

- 7) 「src」フォルダに下記内容のファイルを作成します
 - source.h

```
#ifndef SOURCE_H_
#define SOURCE_H_
```



```
int add(int a, int b);
int subtract(int a, int b);

#endif
/* SOURCE_H_ */
```

- source.c

```
#include "source.h"

int add(int a, int b) {
    return a + b;
}

int subtract(int a, int b) {
    return a - b;
}
```

- testsource.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "CUnit.h"
#include "source.h"

// This is a test case used to test add() function in source.c
static void test_Add_01(void) {
    // Equal Assertion is used in this test case.
    // 1 is expected value, and add(1,0) is actual return value.
    // If expected value is not same, assertion occurs.
    // We can refer the Reference document for the other useful
    assertion.
    CU_ASSERT_EQUAL(1, add(1,0));
}

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

// 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 test case to test suite
    {"test_Add_01", test_Add_01},
    {"test_Add_02", test_Add_02},
    CU_TEST_INFO_NULL,
};
```

```
static CU_TestInfo tests_Subtract[] = {
    {"test_Subtract", test_Subtract},
    CU_TEST_INFO_NULL,
};

// Declare the test suite in SuiteInfo
static CU_SuiteInfo suites[] = {
    {"TestSimpleAssert_AddSuite", NULL, NULL, tests_Add},
    {"TestSimpleAssert_SubtractSuite", NULL, NULL, tests_Subtract},
    CU_SUITE_INFO_NULL,
};

void AddTests(void) {
    // Retrieve a pointer to the current test registry
    assert(NULL != CU_get_registry());

    // Flag for whether a test run is in progress
    assert(!CU_is_test_running());

    // Register the suites in a single CU_SuiteInfo array
    if (CU_register_suites(suites) != CUE_SUCCESS) {
        // Get the error message
        printf("Suite registration failed - %s\n", CU_get_error_msg());
        exit(EXIT_FAILURE);
    }
}
```

8) 既存ファイル「SampleCUnit.c」の内容を、テストを実行するための下記コードに置き換えます。

- SampleCUnit.c

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

void exit(long);
void abort(void);
int main(void);
extern void 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
printf("Tests completed with return value %d.\n",
CU_basic_run_tests());

// Clean up and release memory used by the framework
CU_cleanup_registry();
}
return 0;
}

void abort(void) {}
void exit(long exitcode) {}

```

9) 既存ファイル「sbrk.h」の内容を、テストを実行するための下記コードに置き換えます。

- sbrk.h

```

/* size of area managed by sbrk */
#define HEAPSIZE 0x800

```

- 10) [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 [C/C++ Project Settings Ctrl+Alt+P] を選択します。
- 11) [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 > ソース」を選択します。[インクルード・ファイルを検索するフォルダ (-include)] リストボックスに「\${workspace_loc:/CUnit/src/Headers}」を追加します。

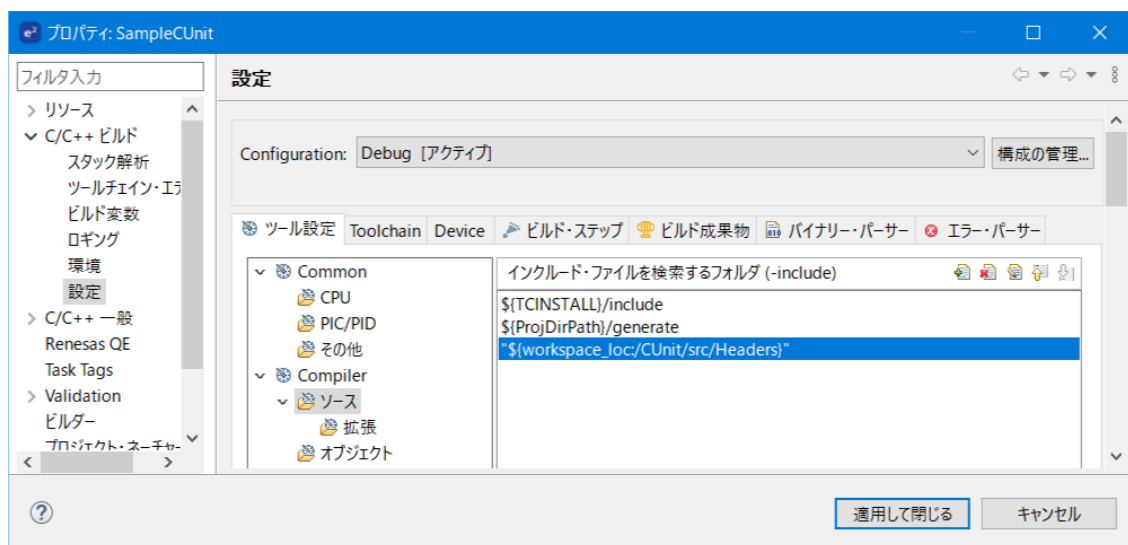


図 10

- 12) [ツール設定] タブのツリーで「Linker > 入力」を選択します。[リンクするリロケータブル・ファイル、ライブラリ・ファイルおよびバイナリ・ファイル (-input/-library/-binary)] リストボックスに「\${workspace_loc:/CUnit/Debug/CUnit.lib}」を追加します。

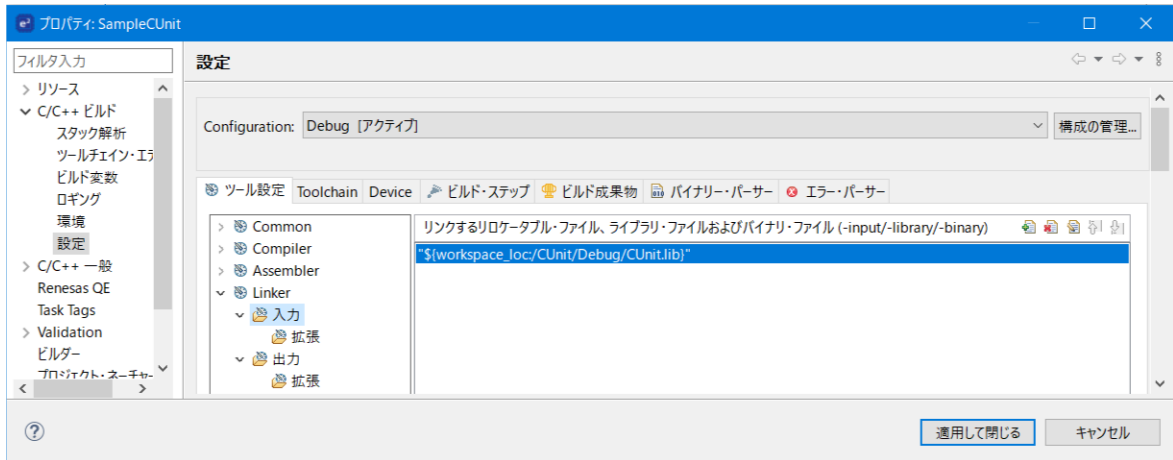


図 11

- 13) [ツール設定] タブのツリーで「Library GeneratorLinker > 構成」を選択します。[C 言語標準ライブラリ関数の構成 (-lang)] コンボボックスで「C89 規格および C99 規格準拠」を選択します。次に、[ctype.h (C89/C99) : 文字操作用ライブラリ (-head=type)] チェックボックスをチェックします。そして、[適用して閉じる] ボタンをクリックします。

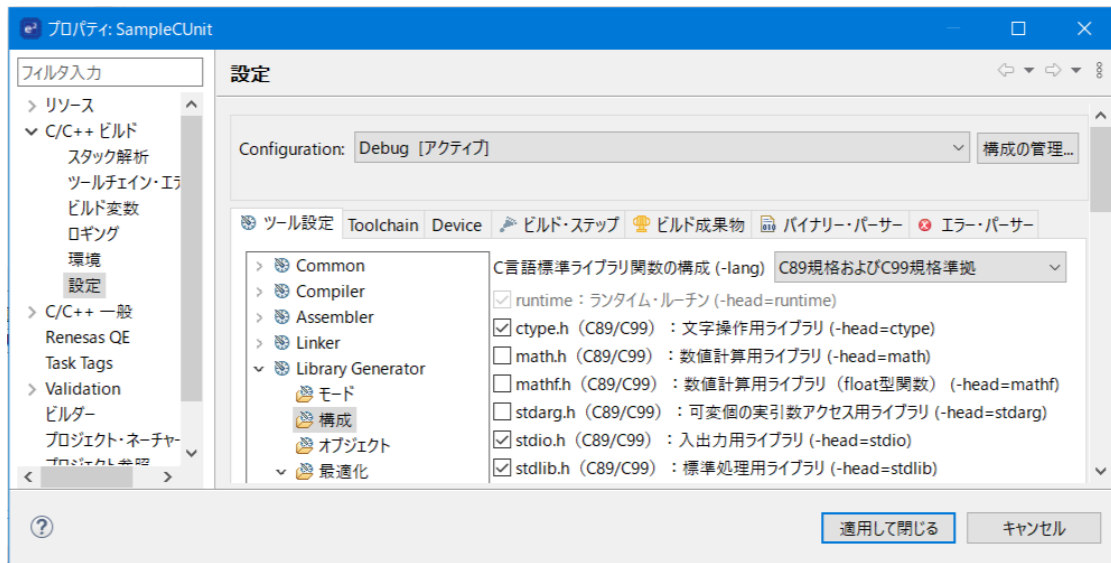


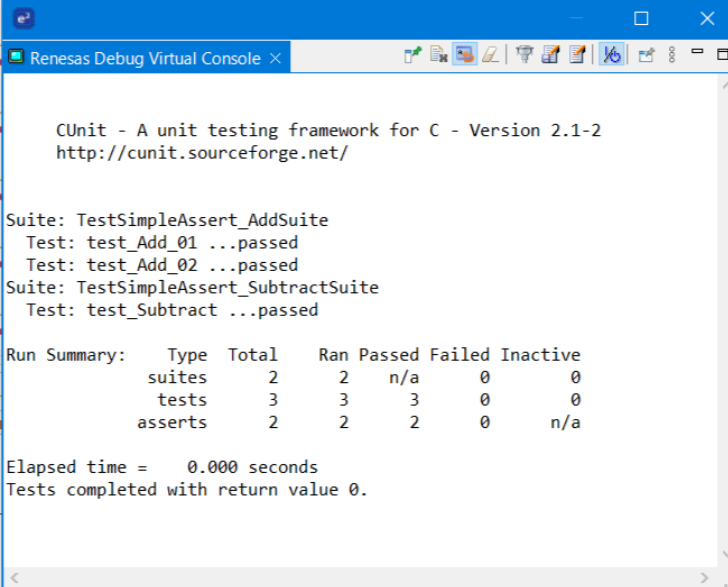
図 12

- 14) [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、「バイナリー」フォルダに「SampleCUnit.abs」ファイルと「SampleCUnit.x」ファイルが表示されます。

3.2 CUnit を使用した単体テストを実行する

- 1) メニュー [実行(R)] > [デバッグの構成(B)] を選択します。
- 2) [デバッグ構成] ダイアログが表示されます。「Renesas Simulator Debugging > SampleCUnit Debug」を選択して、[デバッグ(D)] ボタンをクリックします。
- 3) シミュレータが起動します。メニュー [Renesas View] > [デバッグ] > [Renesas Debug Virtual Console] を選択します。

- 4) [Renesas Debug Virtual Console] ビューが表示されます。メニュー [実行(R)] > [再開(M)] を何度か選択すると、「void Except_BRK(void){ wait(); }」でプログラムが停止します。
- 5) 以下のように、[Renesas Debug Virtual Console] ビューにテスト結果が表示されます。



```
CUnit - A unit testing framework for C - Version 2.1-2
http://cunit.sourceforge.net/

Suite: TestSimpleAssert_AddSuite
Test: test_Add_01 ...passed
Test: test_Add_02 ...passed
Suite: TestSimpleAssert_SubtractSuite
Test: test_Subtract ...passed

Run Summary:
  Type      Total   Ran Passed Failed Inactive
  suites    2       2   n/a    0      0
  tests     3       3    3     0      0
  asserts   2       2    2     0     n/a

Elapsed time = 0.000 seconds
Tests completed with return value 0.
```

図 13 [ターミナル] ビューのテスト結果表示

4. 参考情報

4.1 Web サイト

- e² studio
<https://www.renesas.com/software-tool/e-studio>
- CUnit
<http://cunit.sourceforge.net/>

4.2 他のデバイス、コンパイラ、またはデバッグ環境を使用する場合

このドキュメントは、CC-RX 用シミュレーション環境と printf を組み合わせた環境を前提とした説明になっていますが、Arm 系コア向けのデバッガでは semi-hosting 機能等によりコンソール出力が可能です。また、エミュレータがコンソール出力機能を持たず printf での出力が行えなくても「Dynamic printf」を利用すればコンソールへの表示は可能です。

「Dynamic printf」の使用方法は、下記ページのビデオでご確認いただけます。

[e² studio Tips - ソースコードを変更せずに printf デバッグする方法 \(Dynamic Printf を利用する\) | Renesas](#)

[例]

以下のような自作の printf を作成してそこに「Dynamic printf」を指定すると、本ドキュメントと同様の結果を得ることができます。

- xprintf.h

```
#ifndef XPRINTF_H_
#define XPRINTF_H_

#define printf xPrintf
void xPrintf(const char* format, ...);

#endif
```

- xprintf.c

```
void xPrintf(const char* format, ...);

void xPrintf(const char* format, ...)
{
    static char szBuf[512];
    va_list ap;
    va_start(ap, format);

    vsprintf(szBuf, format, ap);

    va_end(ap); /* here place Dynamic Printf as "%s",szBuf */
}
```

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.02	Jul.12.22	2 ページ 14 ページ	動作環境を e2 studio 2022-04 に更新 「Dynamic printf」の説明を追加
			「

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しないでください。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えば、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、ノイズ輻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まれます。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、これに限られません。）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た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。