統合開発環境 e2 studio

e2 studio での CUnit の使用方法

R20AN0322JJ0101
Rev.1.01
2014.10.01

要旨

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。それはユーザのテスト・コードにリンクされる(静的あるいは動的な)ライブラリとしてビルドされます。

このドキュメントでは、ルネサスの e² studio 環境で CUnit を使用する方法について説明します。

目次

1. はじめに

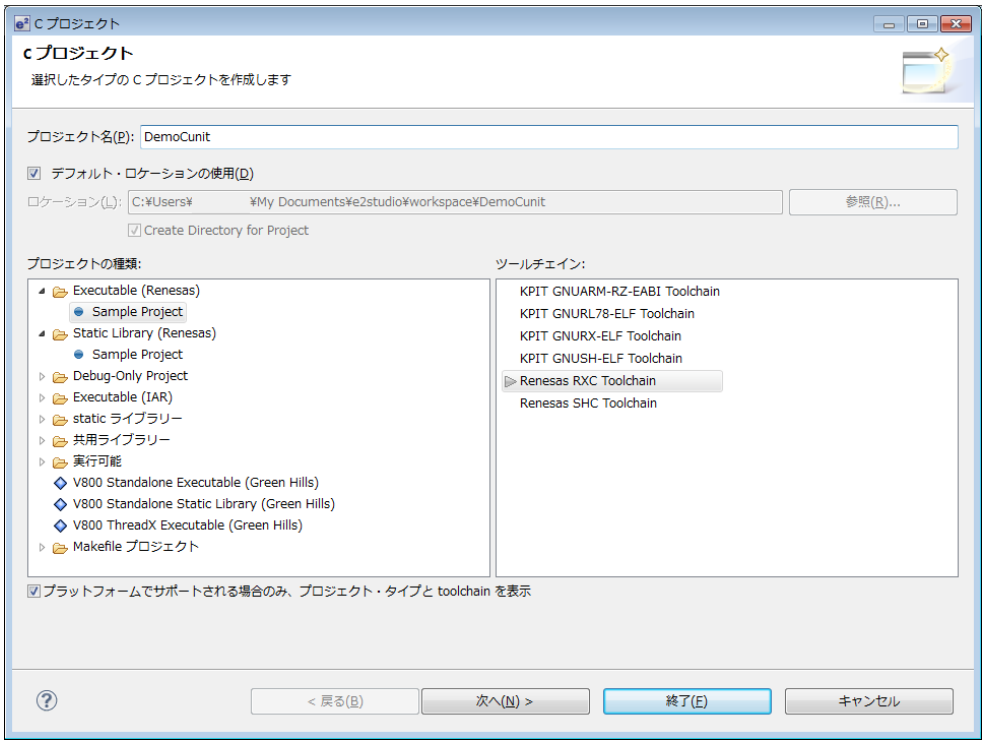
CUnit は C で単体テストを作成、管理、実行するためのシステムです。テスト対象のコードにリンク可能な(static あるいは dynamic)ライブラリとして提供されます。

CUnit はテスト構造のビルドに簡単な枠組みを使用します。一般的なデータ型のテストには豊富なアサーションが利用できます。テストの実行と結果の出力用に複数のインタフェースも用意されています。これらは、ユーザが動的にテストを実行し、結果を見ることができる対話型インターフェースと、ユーザ入力なしにコード制御でテストとレポート出力を行うための自動化インタフェースが含まれます。

このドキュメントでは、ルネサスの e2 studio 環境で CUnit を使用する方法について説明します。CUnit についてより詳細を知りたい場合は、http://cunit.sourceforge.net/doc/index.html を参照してください。

2. ルネサス e^2 studio 環境での CUit の使用方法

1. http://sourceforge.net/projects/cunit/ から CUnit をダウンロードします。CUnit パッケージを解凍します。
2. e^2 studio を起動し、以下のようにルネサス・ツールチェーンで C プロジェクトを生成します:
   a. ‘ファイル’ > ‘新規’ > ‘C Project’ を選択
   b. ‘C プロジェクト’ ダイアログで、プロジェクト名を指定。‘Executable (Renesas)’ カテゴリ > ‘Sample Project’ > ‘Renesas RXC Toolchain’を選択し、‘終了’ボタンを押します。



3. プロジェクトに新しいソースファイルを追加します。例えば:

source.c (このファイルは、テスト対象の関数を含みます。)

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

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

source.h (このファイルは、テスト対象の関数のプロトタイプ宣言を含みます。)

```
#ifndef SOURCE_H_
#define SOURCE_H_

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

#endif /* SOURCE_H_ */
```

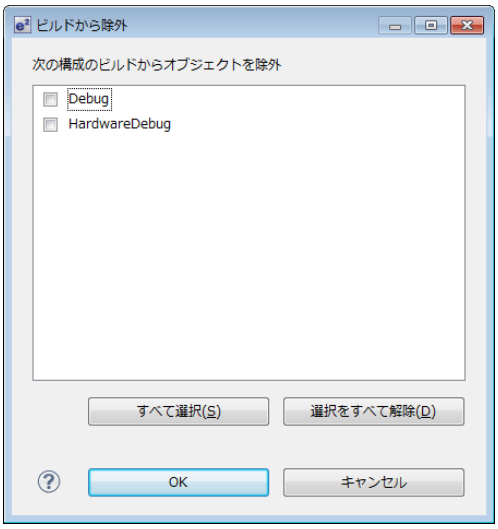
4. CUnit のファイルをプロジェクトにコピーします(テストに使用する CUnit アサーションが含まれています)。

- ‘Basic.c’ (CUnit-2.1-2¥CUnit¥Sources¥Basic)

5. CUnit パッケージからフォルダーをプロジェクトにコピーします (CUnit ヘッダーとフレームワークを含みます)

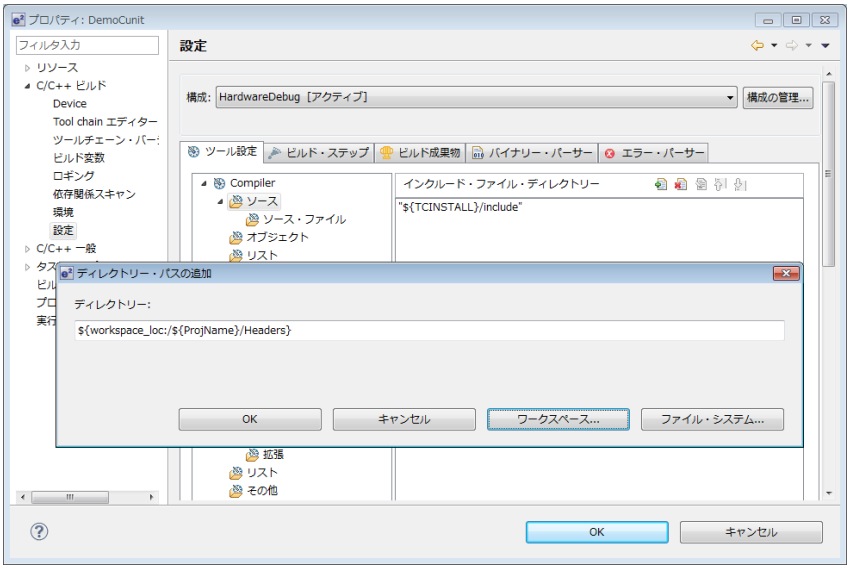
- ‘Header’ フォルダー (CUnit-2.1-2¥CUnit)
- ‘Framework’ フォルダー (CUnit-2.1-2¥CUnit¥Sources)

‘Header’ 、‘Framework’フォルダーはビルドから除外されているので、ソースフォルダーとして認識されていません。‘Header’ と ‘Framework’のフォルダを右クリックし、‘ビルドから除外…’を選択 > ‘選択をすべて解除’ ボタンを選択 > ‘OK’ ボタンを押すことでこれらのフォルダーをビルドの対象とすることができます。



6. ヘッダー・ファイルを使用するため‘Header’ フォルダーを ‘インクルード・ファイル・ディレクトリ’ に追加します。

インクルードディレクトリを以下のように追加します: ‘プロパティ:DemoCunit’ ダイアログで、‘C/C++ ビルド’ > ‘設定’を選択 > ‘ツール設定’ タブを選択 > ‘Compiler’ > ‘ソース’ > ツールバーの‘追加…’ ボタンを押す > ‘ディレクトリ・パスの追加’ ダイアログで ディレクトリ・パスを入力> ‘OK’ ボタンを押します。


プロパティ: DemoCunit
フィルタ入力
リソース
C/C++ ビルド
Device
Tool chain エディター
ツールチェーン・バー:
ビルド変数
ロギング
依存関係スキャン
環境
設定
C/C++ 一般
設定
構成: HardwareDebug [アクティブ]
構成の管理...
ツール設定
ビルド・ステップ
ビルド成果物
バイナリー・パーサー
エラー・パーサー
Compiler
ソース
ソース・ファイル
オブジェクト
リスト
インクルード・ファイル・ディレクトリー
"${TCINSTALL}/include"
ディレクトリー・パスの追加
ディレクトリー:
${workspace_loc:/${ProjName}/Headers}
OK
キャンセル
ワークスペース...
ファイル・システム...
拡張
リスト
その他
OK
キャンセル

7. ファイルをプロジェクトにコピーします(Renesas デバッグ仮想コンソールに結果を表示するのに、この手順を行います)。これらの低水準インターフェース・ルーチンについては、コンパイラのマニュアル等を参照してください。

- lowlvl.src, lowsrc.c, lowsrc.h

8. reset_program.c の//Use SIM I/O を含む行のコメントを削除してください。(C ライブラリ関数の初期化のため)

```
#ifdef __cplusplus              // Use SIM I/O
extern "C" {
#endif
extern void _INIT_IOLIB(void);
extern void _CLOSEALL(void);
#ifdef __cplusplus
}
#endif

    _INIT_IOLIB();              // Use SIM I/O

    _CLOSEALL();                // Use SIM I/O
```

9. プロジェクトに‘testsource.c’ を追加します(このファイルは‘source.c’の関数をテストするための関数を含んでいます)。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "CUnit.h"
#include "source.h"
// source.c の add()関数をテストするのに使用されるテストケース
static void test_Add_01(void)
{
  //Equal アサーションがこのテストこのテストケースで使用される。
  //1 は期待値で add(1,0) は実際の返り値。
  //もし期待値が同じでなければアサートが発生する。
  //他の有用なアサーションはリファレンス· ドキュメントに記載されている。
  CU_ASSERT_EQUAL(1,add(1,0));
}
static void test_Add_02(void)
{
  CU_ASSERT_EQUAL(10,add(1,9));
}
// source.c の subtract ()関数をテストするのに使用されるテストケース
static void test_Subtract(void)
{
  //0 は期待値で subtract (1,1) は実際の返り値。
  //もし期待値が同じでなければアサートが発生する。
  CU_ASSERT_EQUAL(0,subtract(1,1));
}
//これはテスト・スイート
static CU_TestInfo tests_Add[] = {
  //Register test case to test suite
  { "testAdd_01", test_Add_01 },
  { "testAdd_02", test_Add_02 },
  CU_TEST_INFO_NULL,
};
static CU_TestInfo tests_Subtract[] = {
```

```
  { "testSubtract", test_Subtract },
  CU_TEST_INFO_NULL,
};
// SuiteInfo でテストスイートを宣言
static CU_SuiteInfo suites[] = {
 { "TestSimpleAssert_AddSuite",  NULL,  NULL,   tests_Add },
 { "TestSimpleAssert_SubtractSuite", NULL,  NULL,    tests_Subtract },
  CU_SUITE_INFO_NULL,
};
void AddTests(void)
{
  //現在のテスト・リポジトリのポインタを取得する。
  assert(NULL != CU_get_registry());
  //テスト実行が進行しているかどうかのフラグ
  assert(!CU_is_test_running());
  //単一 CU_SuiteInfo アレイにスイートを登録する。
  if (CU_register_suites(suites) != CUE_SUCCESS) {
    //エラーメッセージの取得
    printf("suite registration failed - %s¥n",CU_get_error_msg());
    exit(EXIT_FAILURE);
  }
}
```

10. ‘DemoCUnit.c’をオープンします (テスト関数を呼び、テストケースを実行します。)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Basic.h"
void exit(long);
void abort(void);
int main(void);
extern AddTests();
int main(void)
{
  //ベーシック・インターフェイスのための実行モードを定義する。
  //Verbose モード – 実行の詳細を最も多く出力する。
    CU_BasicRunMode mode = CU_BRM_VERBOSE;
    //エラー・アクションの定義
    //エラーが発生した場合も、可能であれば実行を続ける
    CU_ErrorAction error_action = CUEA_IGNORE;
    //フレームワーク・テスト・レジストリの初期化
    if (CU_initialize_registry()) {
      printf("¥nInitialization of Test Registry failed.");
    }
    else {
      // add テスト関数の呼び出し
      AddTests();
      //テスト実行の間出力を制御するベーシック実行モードの設定
      CU_basic_set_mode(mode);
      //エラー・アクションの設定
       CU_set_error_action(error_action);
       //すべての登録されたスイートのすべてのテストの実行
      printf("¥nTests completed with return value %d.¥n",
                CU_basic_run_tests());
      //クリーン・アップとフレームワークで使用されたメモリのため関数を呼び出す
      CU_cleanup_registry();
    }
```

```
    return 0;
}
void abort(void){}
void exit(long exitcode){}
```

11.ヒープサイズを変更するため、‘sbrk.h’をオープンします。（malloc との alloc 関数についての動的なメモリ割り当てのメモリサイズを増やします）

- `Old: #define HEAPSIZE 0x400`
- `New: #define HEAPSIZE 0x800`

12. ‘Standard Library’の‘ctype.h’ を追加します。

ctype.h’を追加するには: ‘プロパティ:DemoCunit’ ダイアログで、‘C/C++ ビルド’ > ‘設定’を選択 > ‘ツール設定’ タブを選択 > ‘Standard Library’ > ‘内容’ > ‘ctype.h (C89/C99): 文字操作用ライブラリー’ を選択 > ‘OK’ ボタンを押します。

13. ‘Framework¥TestRun.c’ と ‘Framework¥Util.c’ をオープンします。 関数 ‘_snprintf’ を ‘printf’へ変更します。 (このステップは C89 の場合のみ必要です)

例:

- 旧: `snprintf(buf, 33, "%d", number);`
- 新: `printf("%d", number);`

14. プロジェクトをビルド可能にするためにダミーのファイルを追加します。

‘Headers’フォルダーに‘time.h’ を追加します。

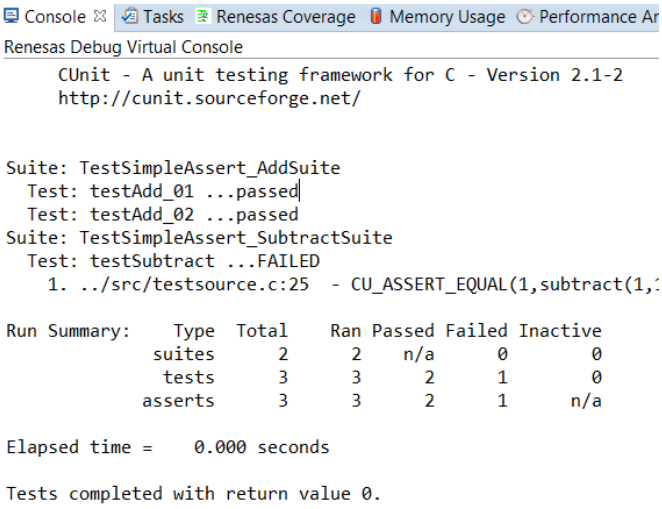
例:

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

15.プロジェクトをビルドします。

16.デバッガに接続し、プログラムを実行します。

17.デバッグ仮想デバッグコンソール’ パネルをオープンし、デバッグ・テストの結果を確認します。



3. 参考情報

3.1 CUnit Web サイト

http://cunit.sourceforge.net/doc/index.html

ホームページとサポート窓口<website and support,ws>

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/contact/

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. 未使用端子の処理
   【注意】未使用端子は、本文の「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。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】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. リザーブアドレス（予約領域）のアクセス禁止
   【注意】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、将来の機能拡張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。
4. クロックについて
   【注意】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。
5. 製品間の相違について
   【注意】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、内部ROM、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。お客様の機器・システムの設計において、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には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 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に関し、当社は、何らの責任を負うものではありません。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. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製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。
5. 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）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、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（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障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。
7. 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、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、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。
8. 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。お客様が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。
9. 本資料に記載されている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。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 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。

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その総株主の議決権の過半数を直接または間接に保有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。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/