Integrated Development Environment e² studio

How to use CUnit in e² studio (CC-RX)

Introduction

CUnit is a system for writing, administering, and running unit tests in C. It is built as a library (static or dynamic) which is linked with the user's testing code.

CUnit uses a simple framework for building test structures and provides a rich set of assertions for testing common data types. In addition, several different interfaces are provided for running tests and reporting results.

This document describes how to use CUnit to automate unit testing using Renesas GCC Executable projects created in e² studio.

Contents

1. Overview

1.1 Purpose

CUnit is a system for writing, administering, and running unit tests in C. It is provided as a (static or dynamic) library linkable with test target code.

CUnit uses a simple framework for building test structures, and provides a rich set of assertions for testing common data types. In addition, several different interfaces are provided for running tests and reporting results. These include automated interfaces for code-controlled testing and reporting without user inputs, as well as interactive interfaces allowing the user to run tests and view results dynamically.

This document describes how to use CUnit in Renesas e^2 studio environment. If you would like to know more about CUnit, please refer to http://cunit.sourceforge.net/doc/index.html.

1.2 Operation Environment

Target device	RX610
IDE	e² studio 2022-04
Toolchains	CC-RX V3.04
CUnit version	2.1.2

2. Getting started with CUnit

This section shows how to setup CUnit to e² studio.

[Important notes]

- Download and use CUnit-2.1-2. CUnit--2.1-3 has some problems which causes build errors. Besides, CUnit-2.1-2 package lacks header file “ExampleTests.h”. Don’t build examples.
- The compiler (and Windows system) does not support “curse” module. Don’t build “curse”.

2.1 Building CUnit library

CUnit can be built to be a static library to be linked to user’s code. This section shows how to build the static library.

1) Download CUnit-2.1.2 from https://sourceforge.net/projects/cunit/files/CUnit/2.1-2/. Extract compressed file to get CUnit package.
2) Launch e² studio. In "C/C++" perspective, click [File] > [New] > [Renesas C/C++ Project] > [Renesas RX].
3) In the [New C/C++ Project - Templates for Renesas RX Project] dialog, choose "Renesas CC-RX C/C++ Library Project" and click [Next >] button.

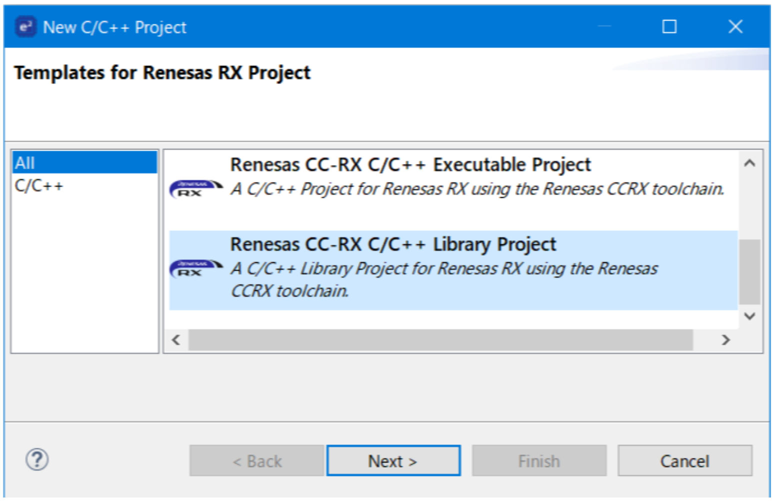

Figure 1

4) In [Project name:] enter the name "CUnit" and click [Next >] button.
5) In the [New Renesas CC-RX Library Project - Select toolchain, device & debug settings] page, enter the following information (other values can remain at default):
   - Toolchain: "Renesas CCRX"
   - Toolchain Version: e.g. "v3.04.00”
   - Target Device: e.g. "R5F56107VxFP"

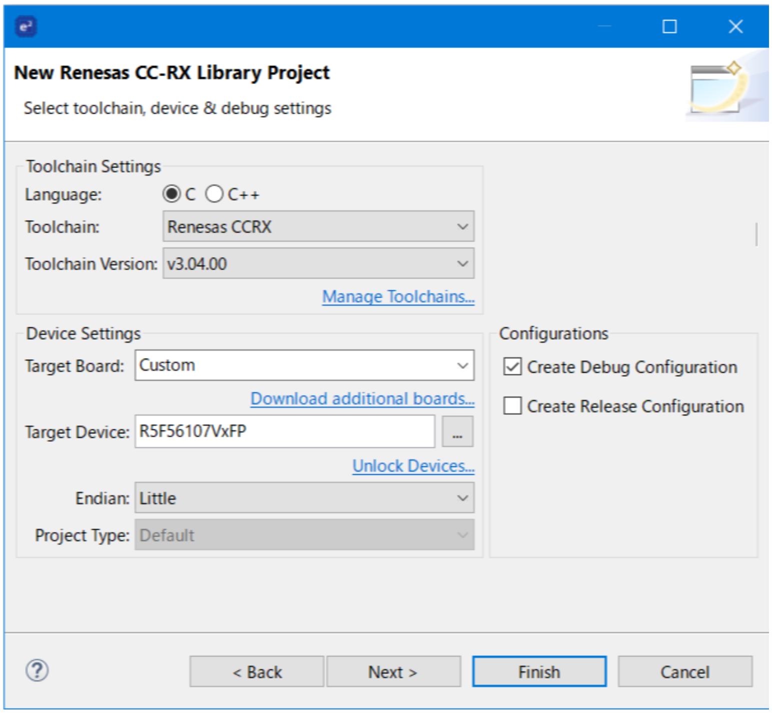

Figure 2

6) Click [Finish] button.

7) In the [Project Explorer] view, expand the CUnit project and delete files (sample1.c, sample2.c and sample3.s) in the folder "src".

8) From the CUnit directory, downloaded and extracted previously, copy “Headers” into the “src” folder in CUnit library project. Then, copy “Basic” and “Framework” of “Sources” subdirectories in CUnit into the "src" folder in CUnit library project. This can be accomplished, in Windows, using either the clipboard or by drag and drop from a File Explorer into e² studio.

9) The project should resemble the figure below:

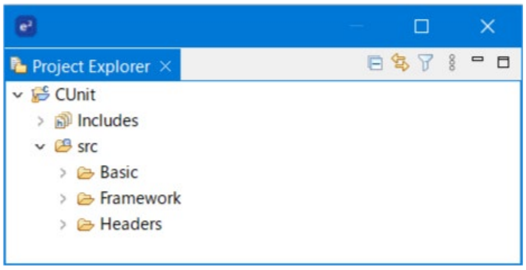

Figure 3

10) Open project properties, select [C/C++ Build] > [Settings], [Compiler] > [Source], then in [Include file directories (-include)] click [Add...] button and add include file directory "${workspace_loc:/${ProjName}/src/Headers}".

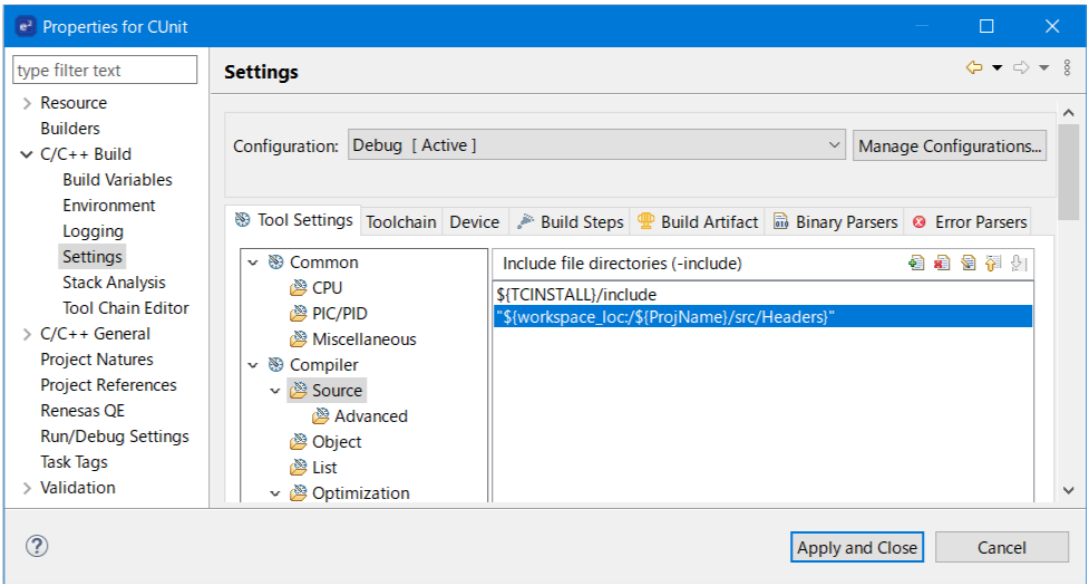

Figure 4

11) Select [Compiler] > [Source] > [Advanced], then select "C99" in [C source file (-lang)]. Next click [Apply and Close] button.

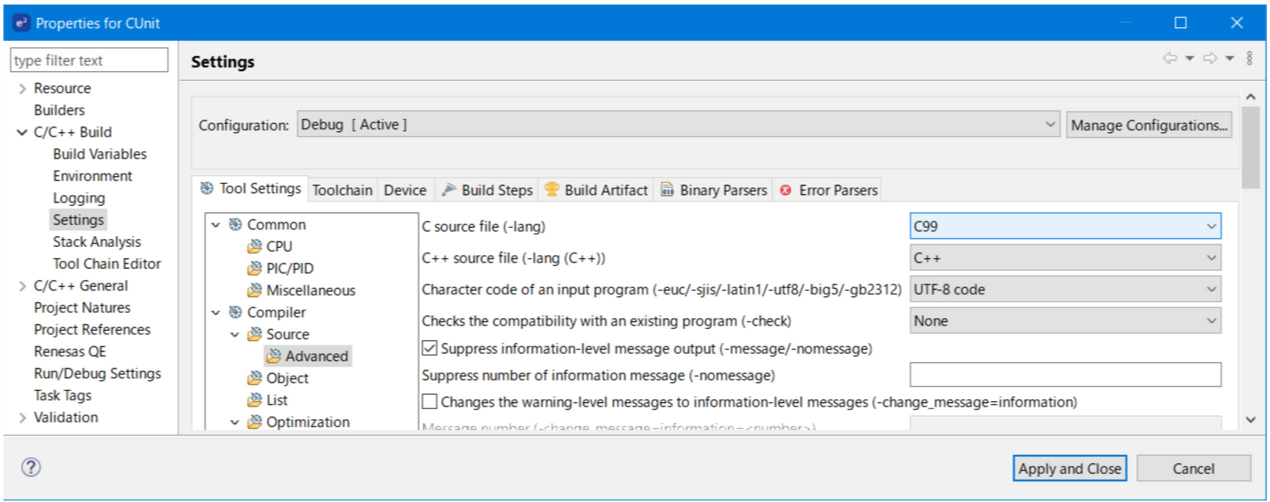

Figure 5

12) Create the following new files in "src\Headers" folder since CC-RX does not support "time.h":

- time.h

```
#ifndef TIME_H_
#define TIME_H_

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

#endif
/* TIME_H_ */
```

13) Build the project. The file "CUnit.lib" will appear inside the "Debug" folder, as shown in the figure below.

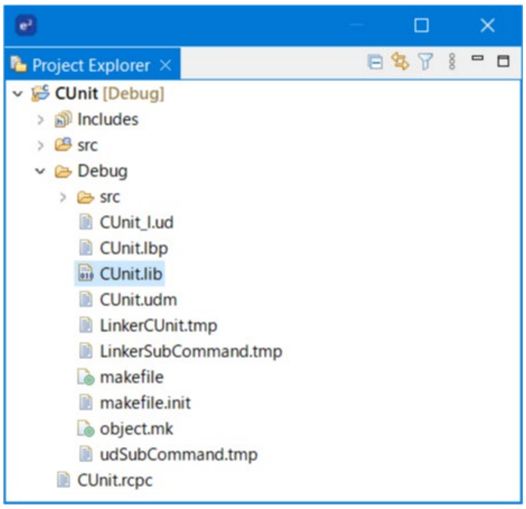

Figure 6

The CUnit library file, “CUnit.lib”, can now be used in any C/C++ project to provide a CUnit test framework.

2.2 Performing unit testing using CUnit

1) In "C/C++" perspective, click [File] > [New] > [Renesas C/C++ Project] > [Renesas RX].

2) In the [New C/C++ Project - Templates for Renesas RX Project] dialog, choose "Renesas CC-RX C/C++ Executable Project" and click [Next >] button.

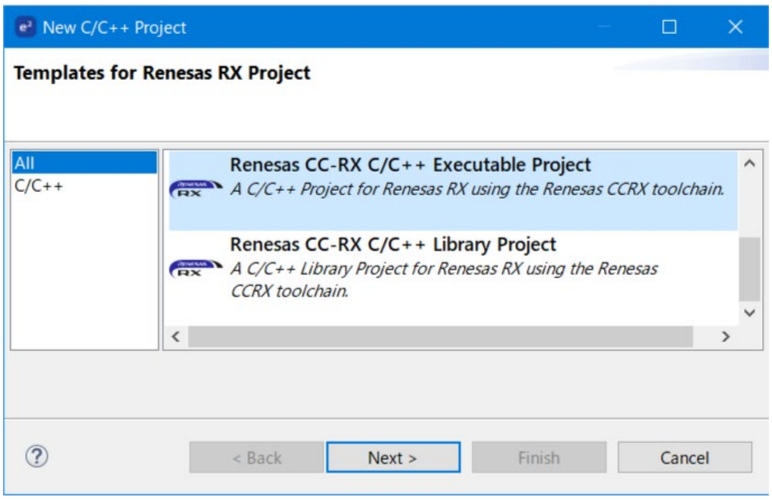

Figure 7

3) In [Project name:] enter the name "SampleCUnit" and click [Next >] button.

4) In the [New Renesas CC-RX Executable Project - Select toolchain, device & debug settings] page, enter the following information (other values can remain at default):

- Toolchain: "Renesas CCRX"
- Toolchain Version: e.g. "v3.04.00"
- Target Device: e.g. "R5F56107VxFP"
- Uncheck [Create Hardware Debug Configuration]
- Check [Create Debug Configuration] for “RX Simulator”.

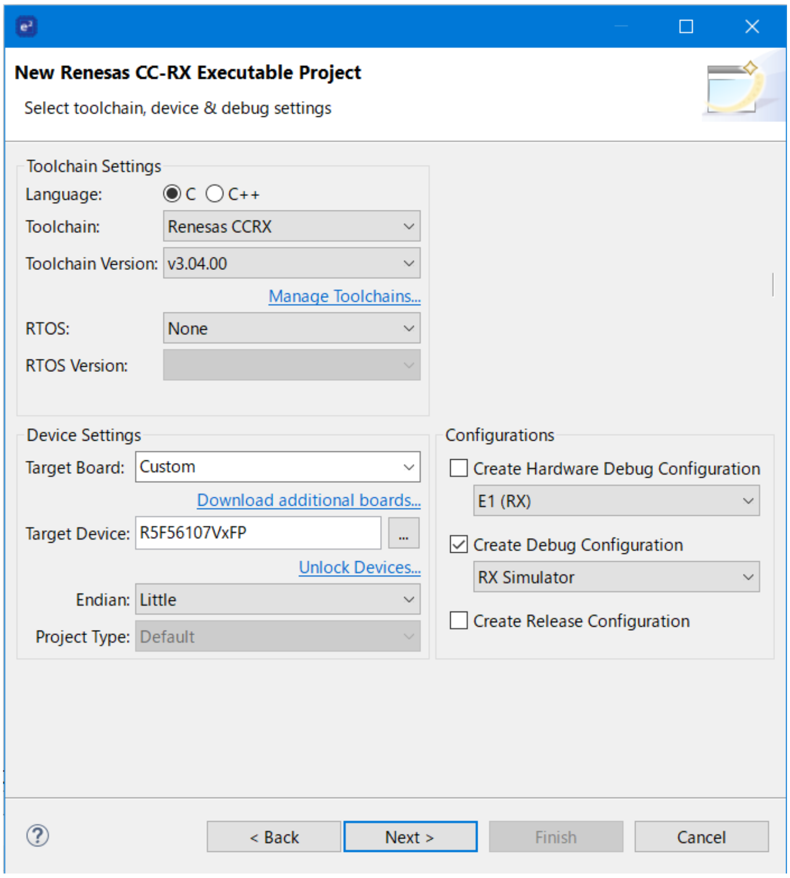

Figure 8

5) Keep clicking [Next >] button until the [New Renesas CC-RX Executable Project - Settings The Contents of Files to be Generated] page is reached. Check [Use Renesas Debug Virtual Console]. Click [Finish] button.

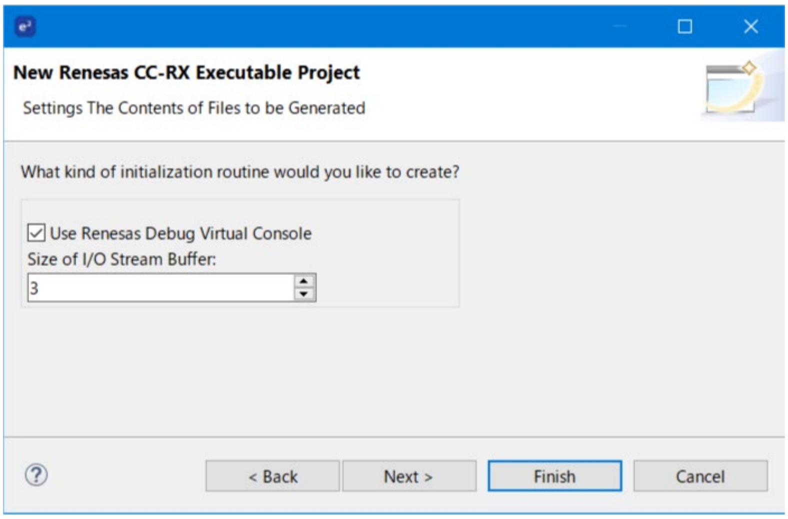

Figure 9

6) Create the following new files to be tested in “src” folder:

- 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);
  }
}
```

7) Replace the contents of the existing source file, “SampleCUnit.c”, and add code to run the test.

- 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) {}
```

8) Replace the contents of the existing header file “sbrk.h” in “generate” folder.

- sbrk.h

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

9) Open project properties, select [C/C++ Build] > [Settings], [Compiler] > [Source], then in [Include file directories (-include)] click [Add...] button and add the include file directory from the CUnit project, "${workspace_loc:/CUnit/src/Headers}".

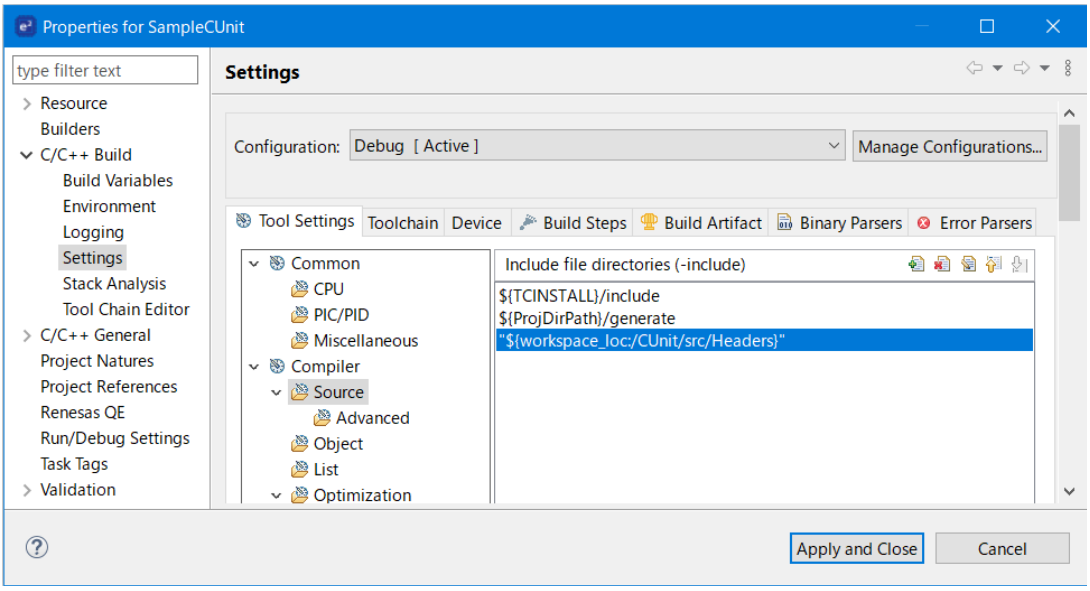

Figure 10

10) In [Linker] > [Input], [Relocatable files, object files and library files (-input/-library/-binary)], add the CUnit library "${workspace_loc:/CUnit/Debug/CUnit.lib}".

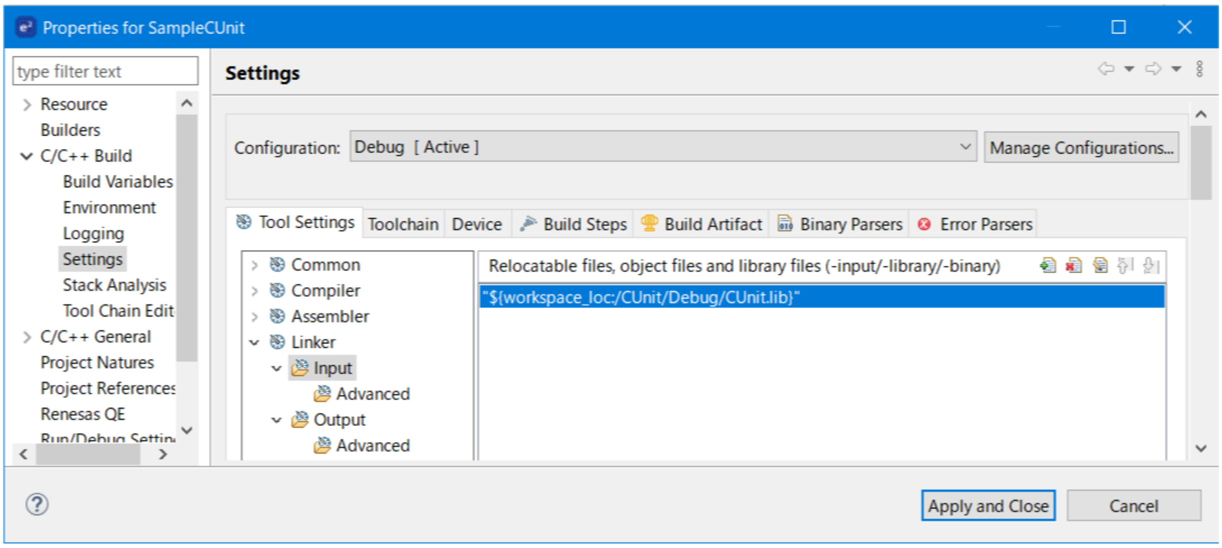

Figure 11 Add CUnit library to linker

11) In [Library Generator] > [Standard Library], choose “C99” in [Library configuration (-lang)] and check [ctype.h (C89/C99): Character classification routines (-head=ctype)] checkbox. Then, click [Apply and Close] button.

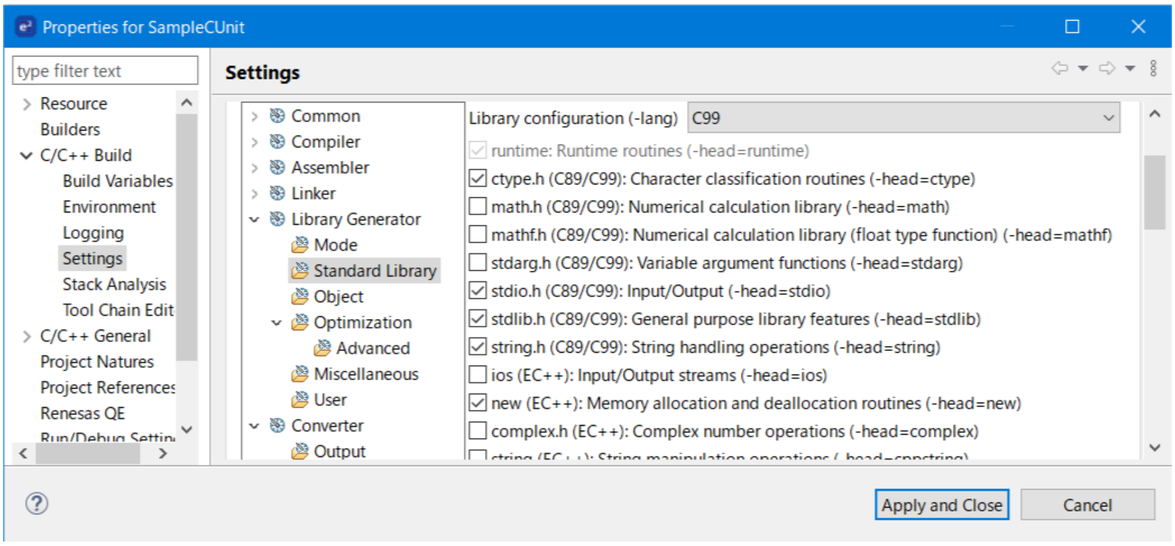

Figure 12 Check Optimization setting

12) Build the project.

13) To run the test harness use Renesas Simulator Debugging. To do this, select menu [Run] > [Debug Configurations...].

14) In the [Debug Configurations] dialog, choose [Renesas Simulator Debugging] > [SampleCUnit Debug]. Then, click [Debug] button.

15) Simulator is executed. Select menu [Renesas Views] > [Debug] > [Renesas Debug Virtual Console].

16) [Renesas Debug Virtual Console] view is displayed. Select menu [Run] > [Resume] several times. You can see that your program stops at the “void Excep_BRK(void){ wait(); }”.

17) At that time, the test result is displayed in the [Renesas Debug Virtual Console] view, as shown in the figure below:

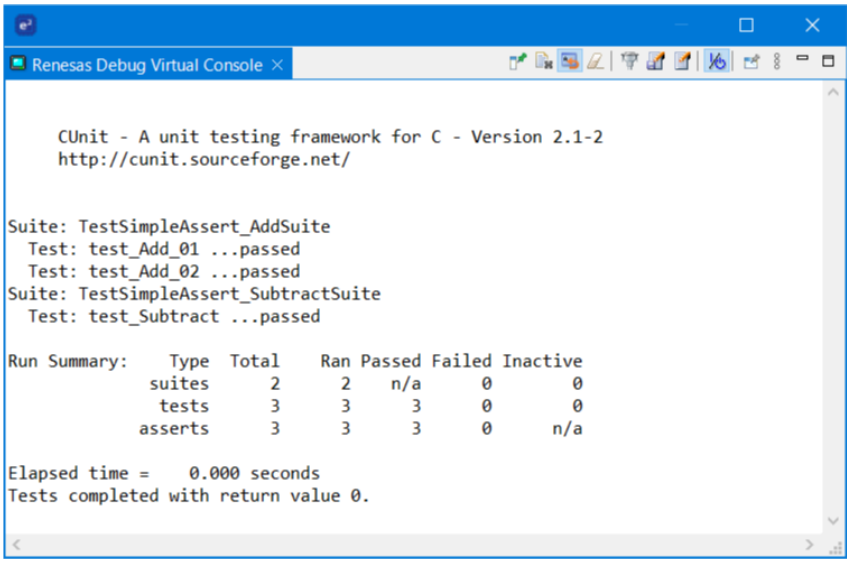
Renesas Debug Virtual Console
CUnit - A unit testing framework for C - Version 2.1-2
http://cunit.sourceforge.net/
Suite: TestSimpleAssert_AddSuite
Test: test_Add_01 ...passed
Test: test_Add_02 ...passed
Suite: TestSimpleAssert_SubtractSuite
Test: test_Subtract ...passed
Run Summary: Type Total Ran Passed Failed Inactive
suites 2 2 n/a 0 0
tests 3 3 3 0 0
asserts 3 3 3 0 n/a
Elapsed time = 0.000 seconds
Tests completed with return value 0.

Figure 13

3. Reference information

3.1 Website and Support

- e² studio

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

- CUnit

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

3.2 When using other devices or compiler or debugger

This document assumes an environment that combines CC-RX simulation environment and printf, but in the debugger for Arm cores, console output is possible by semi-hosting function etc. In addition, even if the emulator does not have a console output function and output with printf cannot be performed, it is possible to display on the console by using "Dynamic printf"

You can see how to use "Dynamic printf" in the video on the following page.

e² studio Tips - How to Use Printf Debugging Without Changing the Source Code (Using Dynamic Printf) | Renesas

[Example]

If you create your own printf as shown below and specify "Dynamic printf" there, you can get the same result as in this document.

- 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 */
}
```

Revision History

Rev.	Date	Description	
		Page	Summary
1.02	Jul.12.22	Page 2 Page 14	Update the operation environment of e^2 studio. Add the explanation of “Dynamic printf”.

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.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

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/.

Trademarks