Introduction

Welcome to the world of development environment CS+.
This tutorial introduces you to the integrated development environment CS+ and its usage. By carrying out all the steps described in this tutorial, from creating a program to debugging of the microcontroller, you can easily experience the operation of CS+.
In this tutorial, you will use the E1 (on-chip debugging emulator) and RH850/C1x evaluation board (from Sunny Giken Inc.) to actually experience microcontroller system development using CS+.

Features of CS+

CS+ is an integrated development environment that provides an environment for developing microcontrollers from code generation, build, and debugging all in one tool.

Easy GUI customization

You can customize the screen as you like using such features as "docking", "floating", or "automatic hiding" to manipulate various panels of CS+ at will. CS+ now provides a feature to save the development environment in addition to the conventional feature to save the project environment. All of these features help you develop a microcontroller system more smoothly.

Easy preparation of development environment

Since the development environment for system development is integrated, it is easy to install the required tools. Because it is equipped with an automatic update function, you can update the software to its latest version (including documents) with a single click.

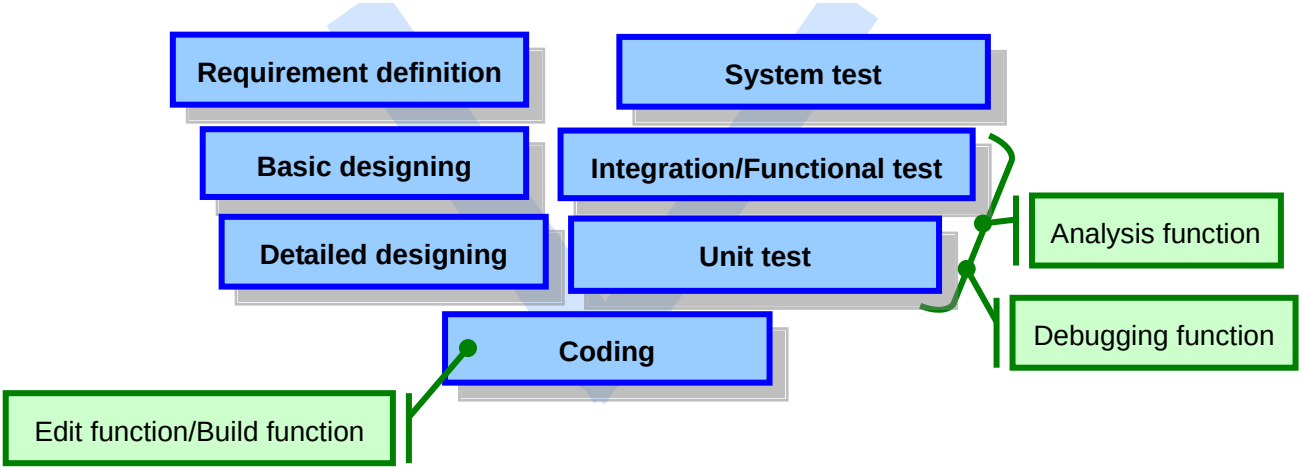
Reading this tutorial and the following document enables you to learn Overeview of programming for the RH850 multi-core.

Overview of programming for the RH850 multi-core(R20UT3069EJ)

Flow of Microcontroller System Development

This section describes a flow of system development using CS+.

General flow of system development (V-shaped model)



Following are the functions of CS+ corresponding to the flow of system development.

<Function>	<Description>
Edit function/ Build function	The edit function is for editing a program. After completing creation of a program, the built function is used to build the program.
Debugging function	This function enables you to debug the object code after downloading it to the target microcontroller.
Analysis function	This function helps you improve the execution performance and control the quality of a program by checking the analysis result.

Overview of Sample Program

This section describes an overview of the sample program and target board (RH850/C1x EVALUATION BOARD).

1. Overview of sample program

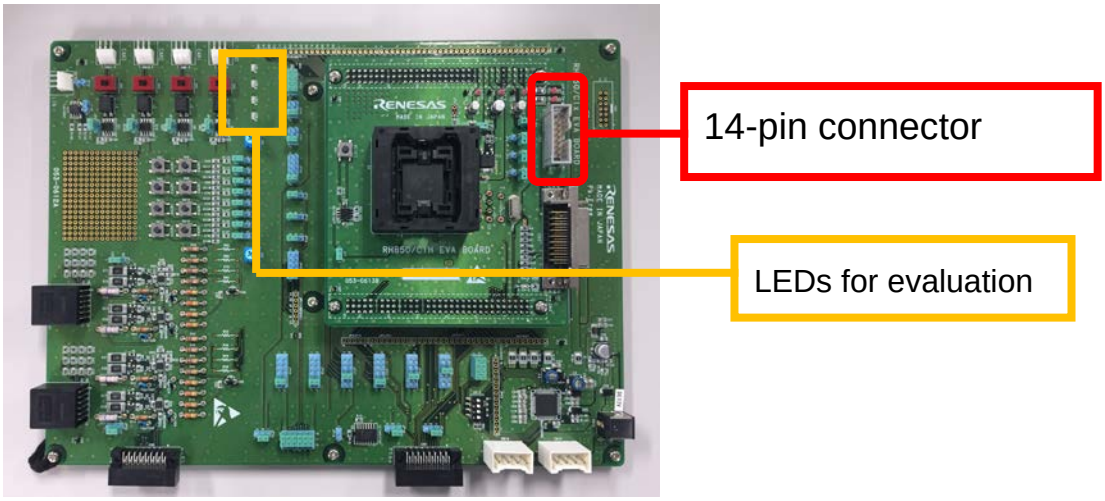
The program used here controls (turns on/off) a different LED for each core (CPU1 and CPU2) of RH850/C1H.
For a detailed description of the program, refer to appendix, Description of Sample Programs.

CPU1 core: Controls LED1 and makes LED1 turn on and off.
CPU2 core: Controls LED2 and makes LED2 turn on and off.

2. Overview of target board (RH850/C1x EVALUATION BOARD)

The following is an overview of RH850/C1x EVALUATION BOARD which is used as the target board.

RH850/C1x EVALUATION BOARD



LED8 to LED15: Lights when P4_n (n = 8-11) of port group 4 is high.
CN10: Used at on-chip debugging or data writing

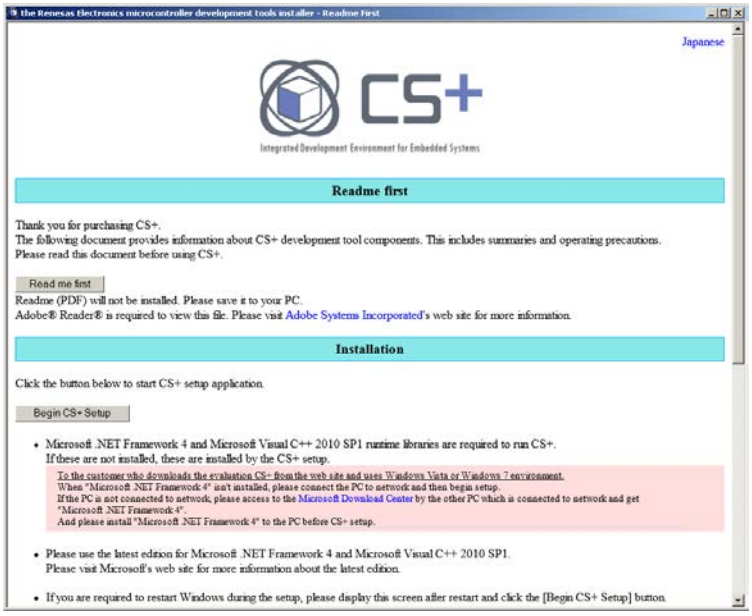
Installing

This section describes a procedure to install CS+.

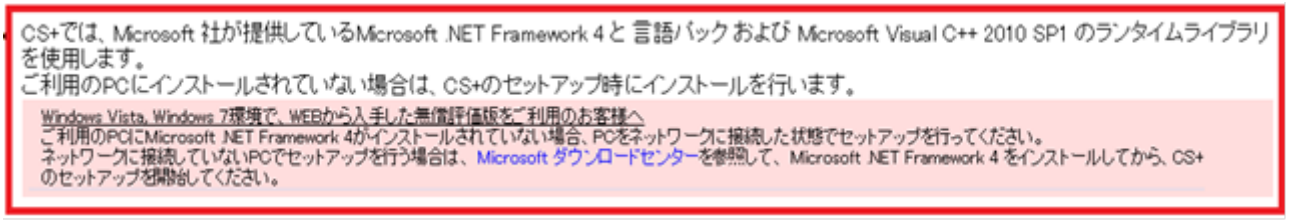
1. Installing Microsoft software products prior to installation

You must install .NET Framework and Visual C++ Runtime Library before installing CS+. If these software products have not been installed in the PC used, they will be installed at the time of the setup of CS+.

Insert CS+ product DVD into the drive of the PC.
The following screen appears automatically.



Install the required software products.

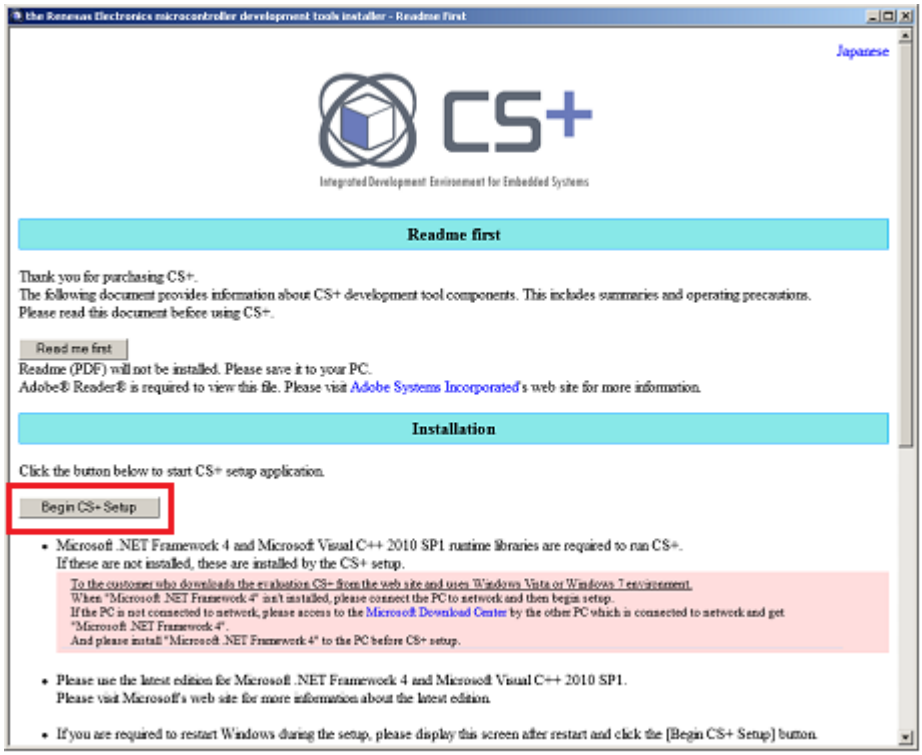


Installing

2. Running the integrated installer

CS+ products are installed by running the integrated installer.

Click [Begin CS+ Setup] and start the setup of CS+.



Make settings following the instructions provided by the installation wizard. As a final step, click the [Finish] button and complete installation.
* Restart the PC after installation.

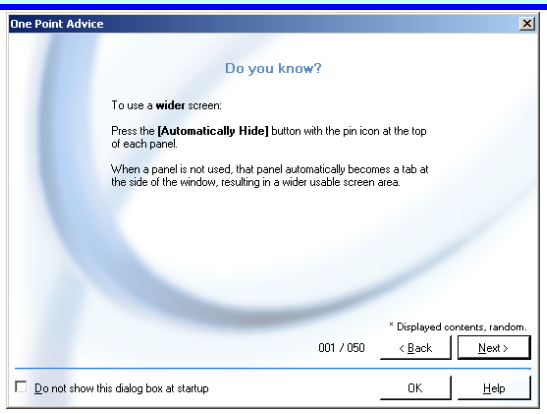
Starting CS+

This section describes the procedures from starting CS+ to creating a project.

1. Starting CS+

Start CS+ by selecting [Start] > [All Programs] > [Renesas Electronics CS+] > [CS+].

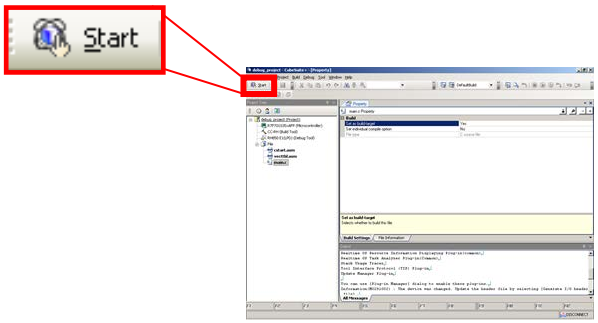
The One Point Advice dialog box opens when CS+ is started. Click the [Next] button if you want to read the content. Clicking the [OK] button displays the start screen of CS+.



Tip

About the Start panel

When you start to use CS+ to create a new project, click the "Start panel" button (see the figure below). The Start panel opens where you can easily create a new project or open the project you used recently or your favorite project. (The Start panel is displayed when you start CS+ for the first time after installation. If you have created a project, the latest project opens when you start CS+.)



Starting CS+

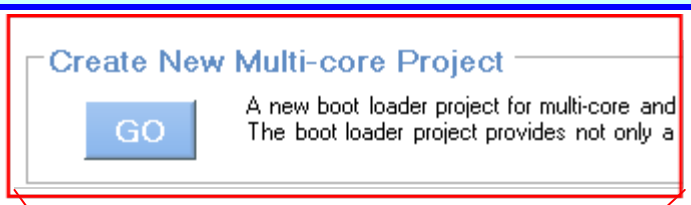
2. Loading the sample project

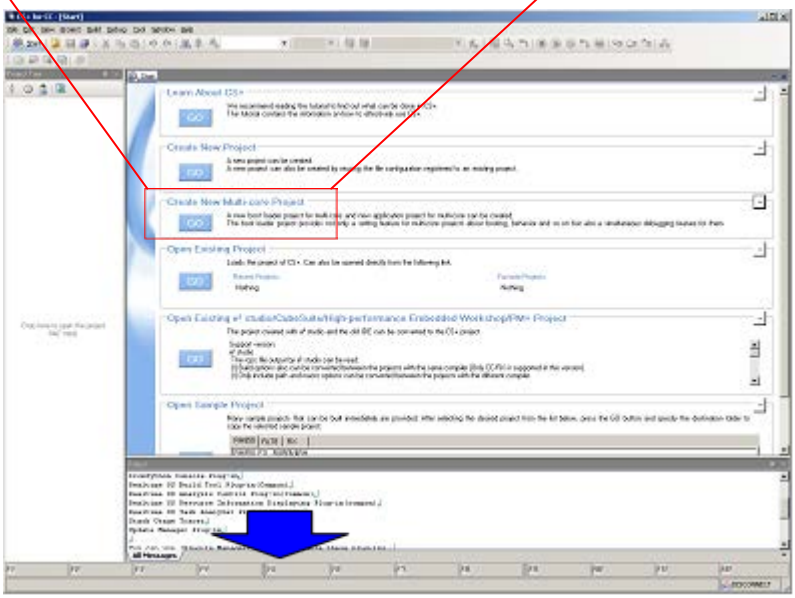
In this step, load the sample project.

This document describes the step using a project that has been created according to the construction method for a project using CS+.
For details, refer to Tutorial for RH850 Multi-core Environment (Build).

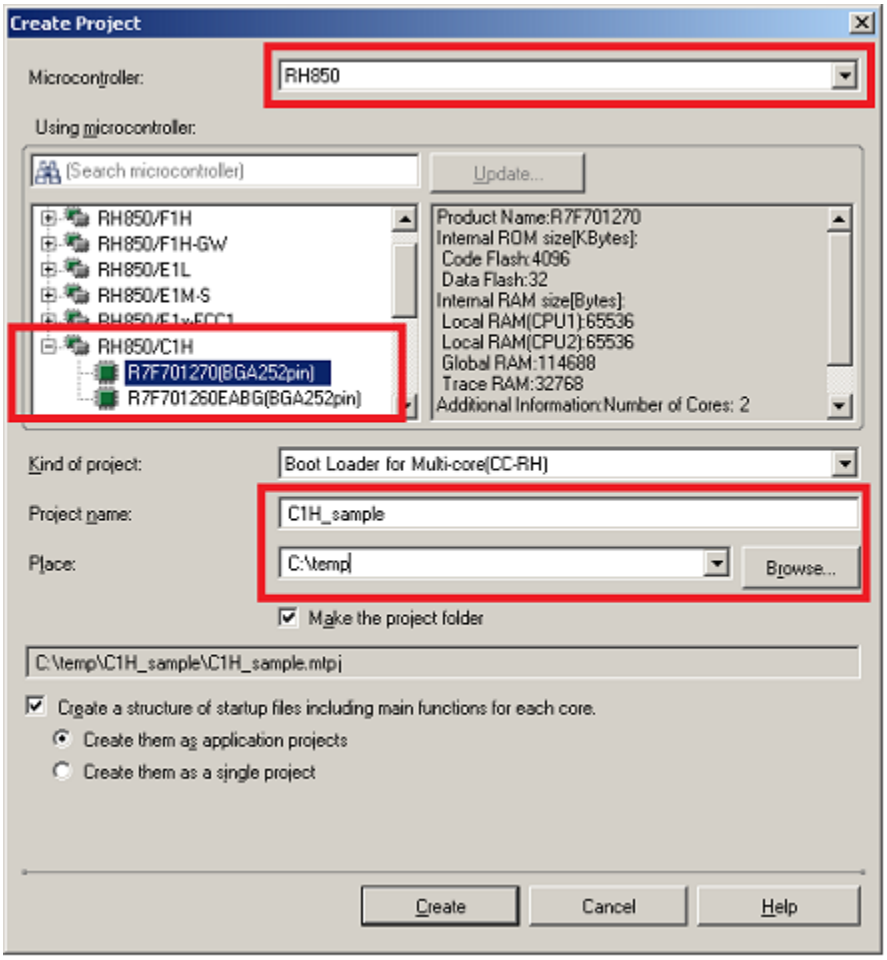
Tutorial for CS+ Ver3.01.00 RH850 Multi-core (Build2)(R20UT3400EJ)

In the "Create New Multi-core Project" field, click the [GO] button.





Select [RH850] for [Microcontroller] and [RH850/C1H] or [R7F701270] for [Using microcontroller] and set [Project name] and [Place]. In this document, "C1H_sample" is set for [Project name].

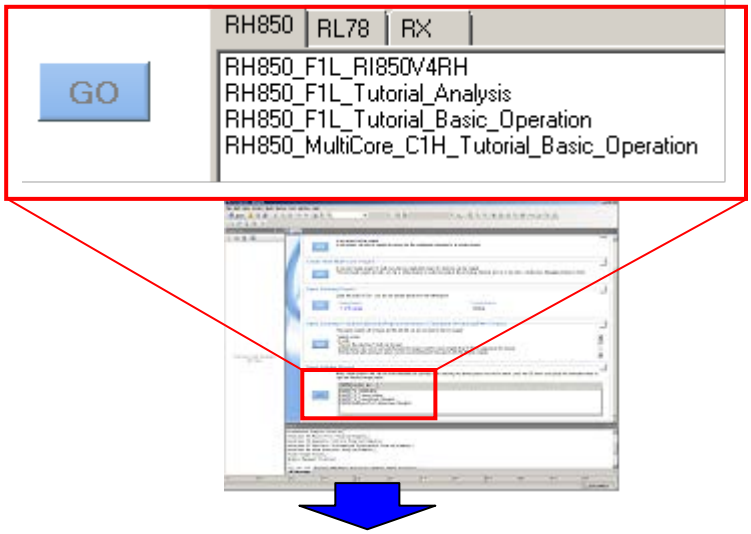


Tip

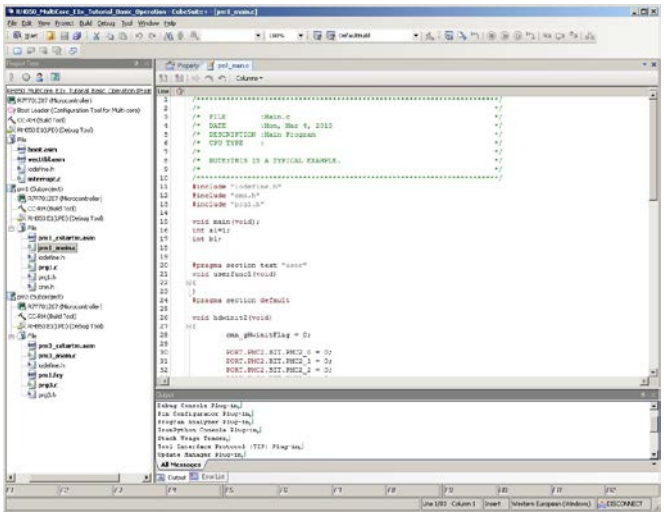
About sample projects

CS+ provides sample projects.
Sample projects are in a state after the "Editing the Program" operations in this document have been performed.
When using a sample project provided by CS+, load the sample project as shown below.

In the "Open Sample Project" field, from the [RH850] tab, select "RH850_Multicore_C1H_Tutorial_Basic_Operation", then click the [GO] button.



Follow the on-screen instructions. A sample project opens as shown in the figure below.

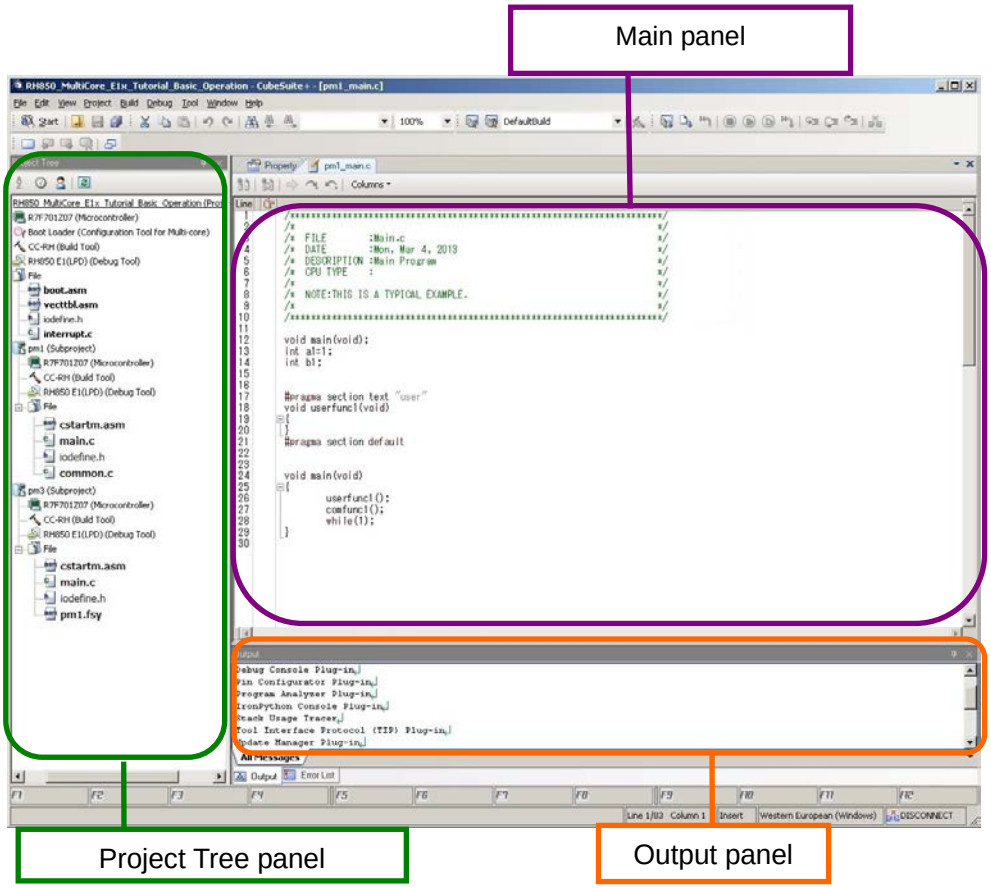


Manipulating Windows

In CS+, you can customize the windows at will. This section describes the configuration of windows and window customization functions such as "automatic hiding", "floating", and "docking".

1. Configuration of windows

The following figure shows the configuration of windows of CS+.



Project Tree panel: Displays the functions of CS+ corresponding to the flow of system development.
Main panel: Displays the panel (Editor panel, etc.) corresponding to the function selected in the Project Tree.
Output panel: Displays the output results.

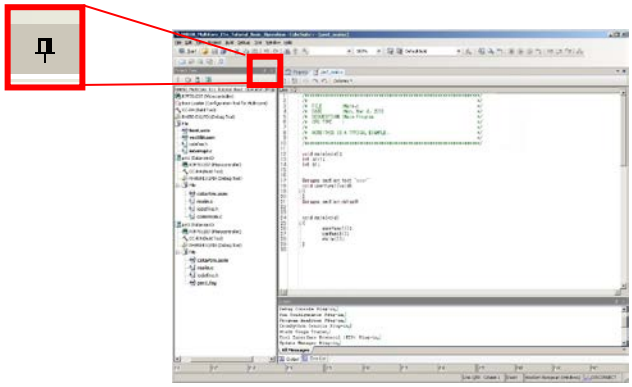
Manipulating Windows

2. Automatic hiding

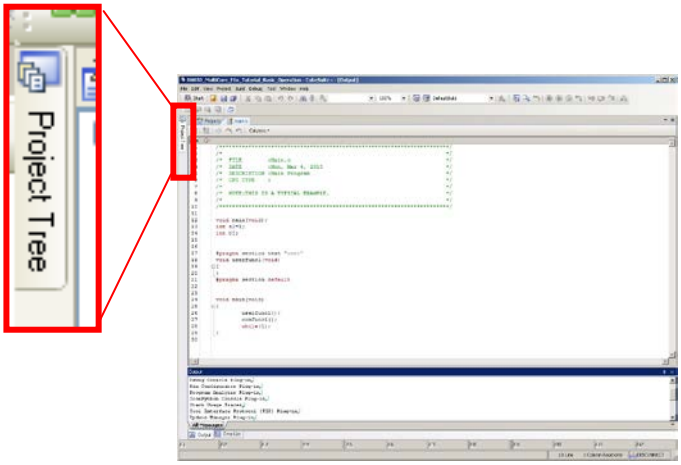
By clicking the Pin icon on the title bar of each panel, you can easily change the setting that determines whether or not to hide the panel automatically. By hiding the panels not necessary for operation, you can use the screen more effectively.

(a) Hiding the panel automatically (example: Project Tree)

Click the Pin icon in the Project Tree.

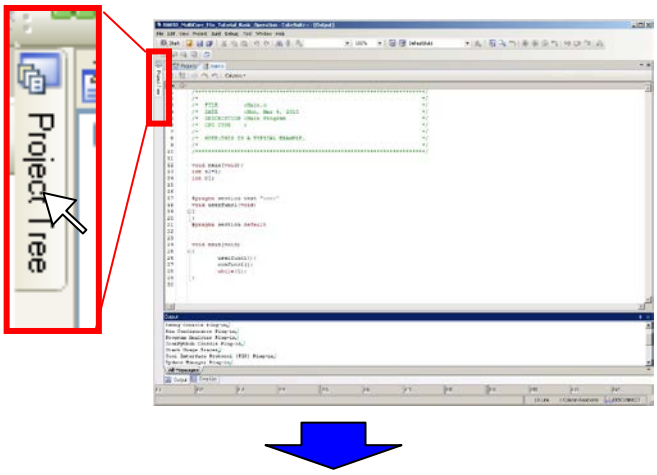


The Project Tree automatically disappears and the tab representing it appears.

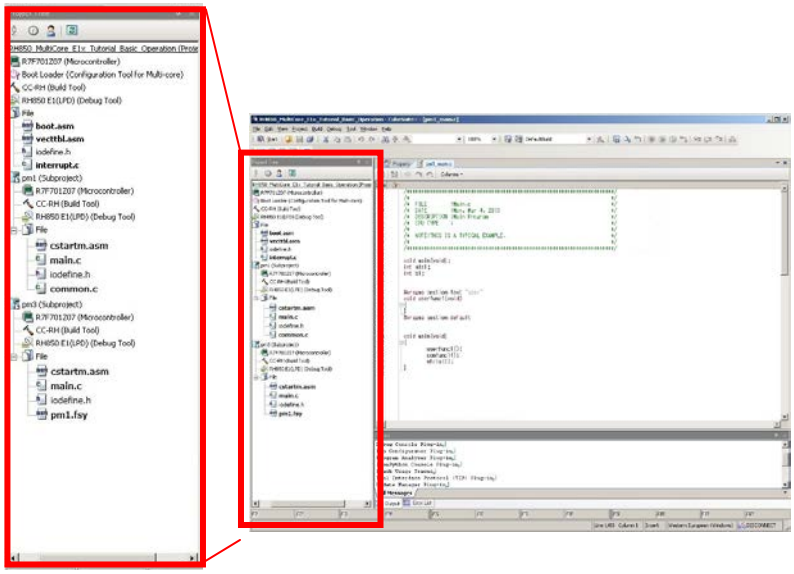


(b) Displaying the hidden panel (example: Project Tree)

Place the pointer on the [Project Tree] tab.



The Project Tree slides out.



Tip

Locations to hide the panels

You can hide the panel in three locations: one to the left of the window, one to the right of the window and one below the window. You can also hide multiple panels in the same location.

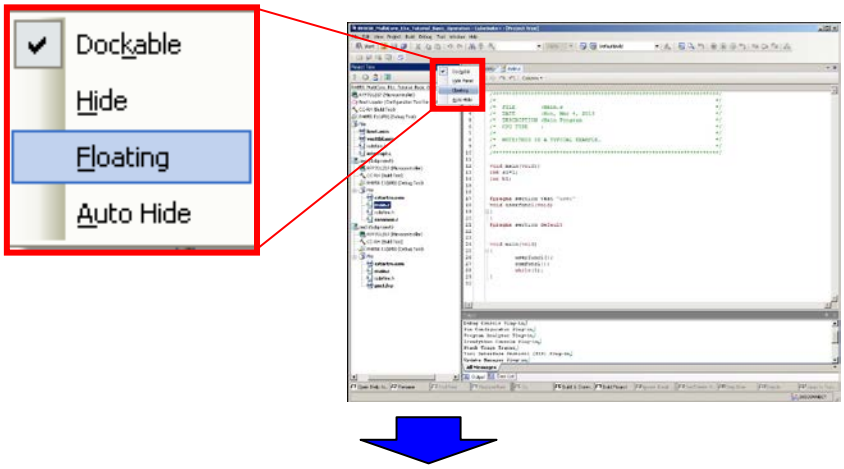
Manipulating Windows

3. Floating

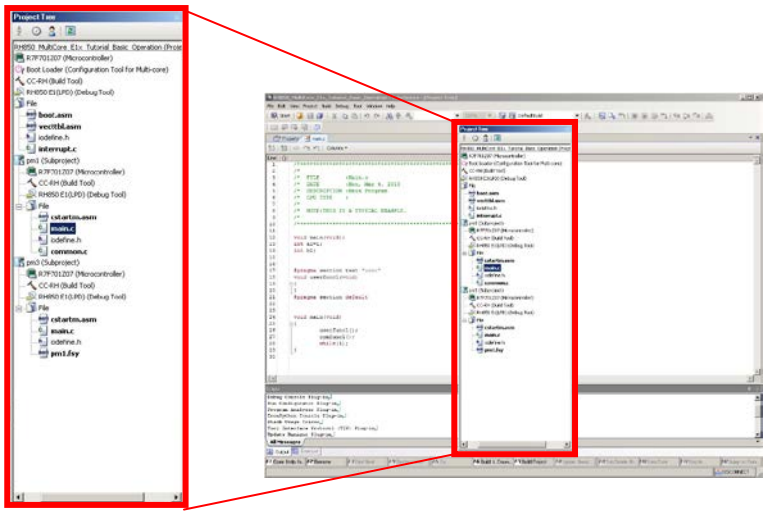
Right-clicking on the title bar and selecting [Floating] from the menu allows you to move the panel at will.

(a) Making the panel float (example: Project Tree)

Right-click on the title bar and select [Floating].



The panel enters the floating state.



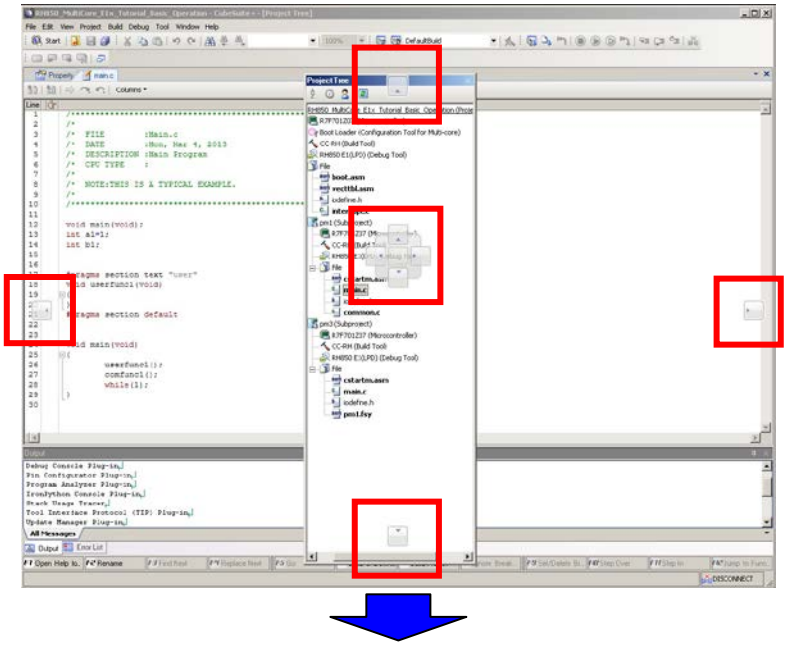
Manipulating Windows

4. Docking

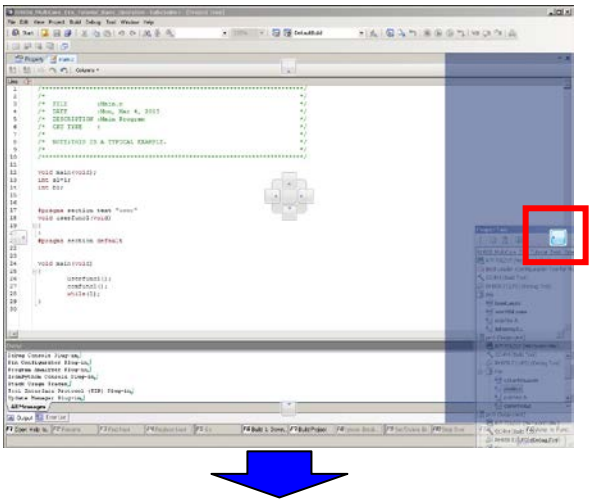
You can attach the floating panel to any of the four sides of another panel such as the main panel. You can easily change the position of the panel by dragging and dropping it to a desired location using a navigation icon.

(a) Moving the panel (example:Project Tree)

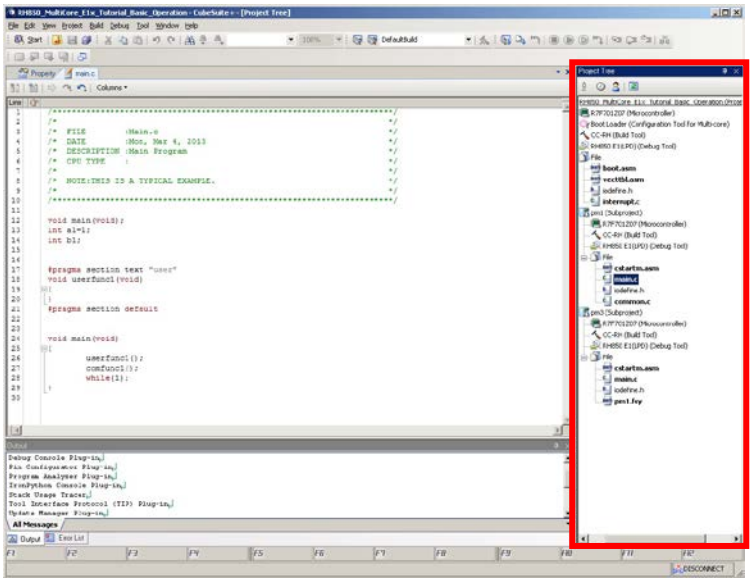
When you drag the floating panel, the navigation icon appears.



Place the pointer on the navigation icon located in the desired destination location and the destination area is highlighted in blue.



Drop the panel there and the Project Tree moves to the desired location (the figure below shows the example of attaching it to the right of the main panel).

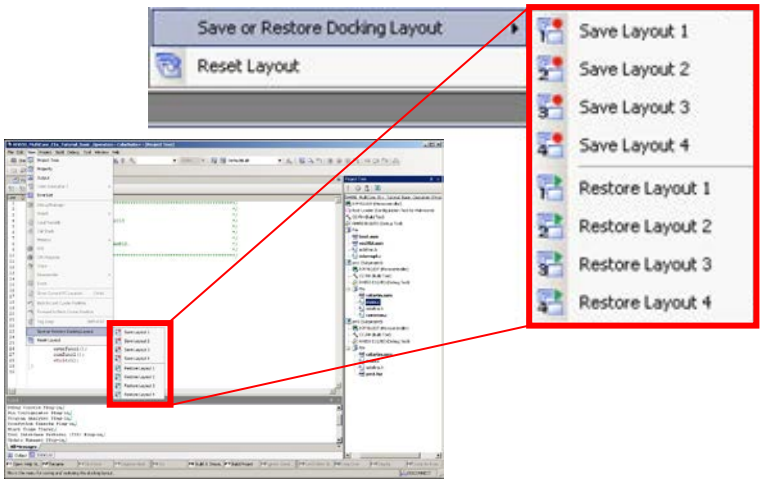


Tip

Saving and restoring layouts

You can save up to four panel layout (panel location information) states for before and after connecting to the debug tool. To do so, from the menu bar, select [View] -> [Save or Restore Docking Layout].

* Becomes a debugging-specific layout only when the debug tool is connected.



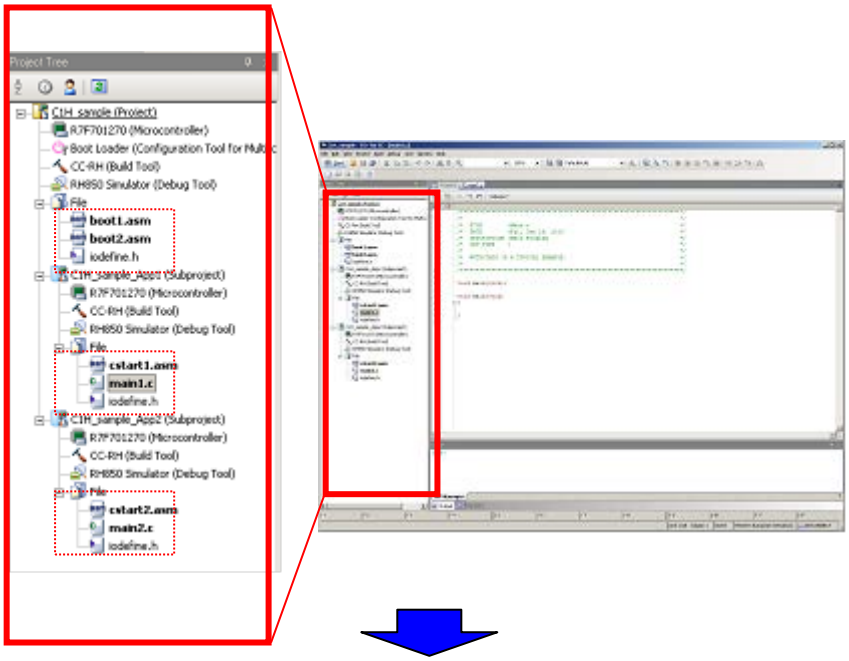
Editing the Program

In this section, you edit the user program.
First, the basic editing method is explained, and then you can edit the program through a simple copy and paste procedure. Edit the program following the steps listed below.

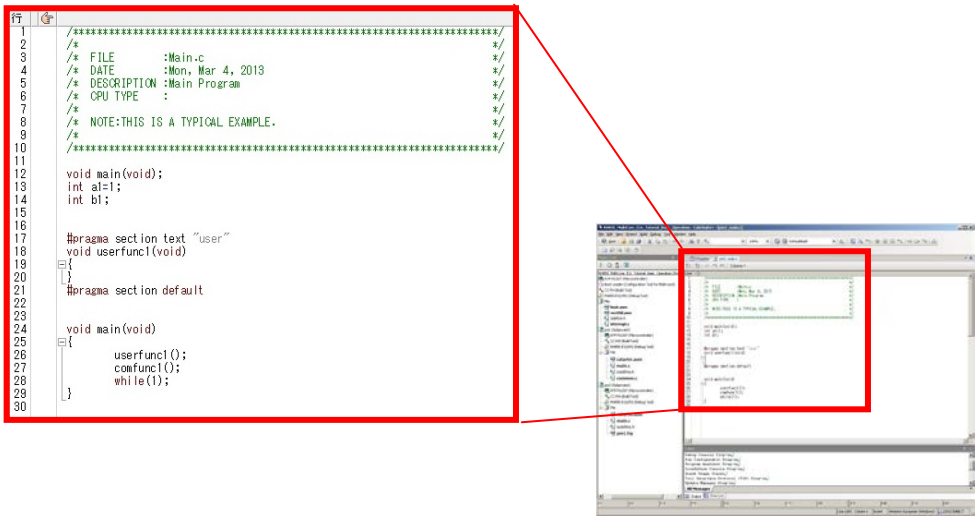
1. How to open a source code file

The following step describes how to open a source code file.

Find the source code file you want to edit in the Project Tree and double-click it.



This displays the source code in the main panel.



Editing the Program

2. Editing

Edit the program following the steps listed below.

Copy the following code and paste it in the main() function of main1.c.

```
void main(void)
{
                    hwinit();
                    g_flag = 1;
                    set();
                    g_counter = 0;

                    while(1){
                                        g_counter++;
                                        if(g_counter == 0x1FFFFF){
                                                            g_counter = 0;
                                                            outputLED1();
                                        }
                    }
}
```

To add include statements to beginning of main1.c, copy and paste the following code.

```
#include "iodefine.h"
#include "common.h"
```

Copy the following code and paste it in the main() function of main2.c.

```
void main(void)
{
                    hwinit();
                    wait();
                    g_counter_pe2 = 0;

                    while(1){
                                        g_counter_pe2++;
                                        if(g_counter_pe2 == 0x1FFFFF){
                                                            g_counter_pe2 = 0;
                                                            outputLED2();
                                        }
                    }
}
```

To add include statements to beginning of main2.c, copy and paste the following code.

```
#include "iodefine.h"
#include "common.h"
```

Delete the semicolon at the beginning of the following comment lines in boot1.asm to make them valid.

```
.L.entry_PE1:
                jarl            _hdwinit_PE1, lp        ; initialize hardware
```

```
.L.entry_PE2:
                jarl            _hdwinit_PE2, lp        ; initialize hardware
$ifdef USE_TABLE_REFERENCE_METHOD
                jarl            _set_table_reference_method, lp ; set table reference method
$endif
                jr32            __cstart_pm2
```

```
;-----------------------------------------------------------------------
;               hdwinit_PE1
; Specify RAM addresses suitable to your system if needed.
;-----------------------------------------------------------------------
                .section        ".text", text
                .align          2
_hdwinit_PE1:
                mov             lp, r14                 ; save return address

                ; clear Global RAM
                mov             GLOBAL_RAM_ADDR, r6
                mov             GLOBAL_RAM_END, r7
                jarl            _zeroclr4, lp

                ; clear Local RAM PE1
                mov             LOCAL_RAM_PE1_ADDR, r6
                mov             LOCAL_RAM_PE1_END, r7
                jarl            _zeroclr4, lp

                mov             r14, lp
                jmp             [lp]
```

```
;-----------------------------------------------------------------------
;               zeroclr4
;-----------------------------------------------------------------------
                .align          2
_zeroclr4:
                br              .L.zeroclr4.2
.L.zeroclr4.1:
                st.w            r0, [r6]
                add             4, r6
.L.zeroclr4.2:
                cmp             r6, r7
                bh              .L.zeroclr4.1
                jmp             [lp]
```

Copy the following code and paste it immediately before hdwinit_PE1 in boot1.asm.

```
;-----------------------------------------------------------------------
;                       RAM address
;-----------------------------------------------------------------------
                        GLOBAL_RAM_ADDR   .set          0xfeef0000
                        GLOBAL_RAM_END    .set          0xfef0bfff

                        LOCAL_RAM_PE1_ADDR              .set            0xfebf0000
                        LOCAL_RAM_PE1_END               .set            0xfebfffff

                        LOCAL_RAM_PE2_ADDR              .set            0xfe9f0000
                        LOCAL_RAM_PE2_END               .set            0xfe9fffff
```

Copy the following code and paste it immediately before zeroclr4 in boot1.asm.

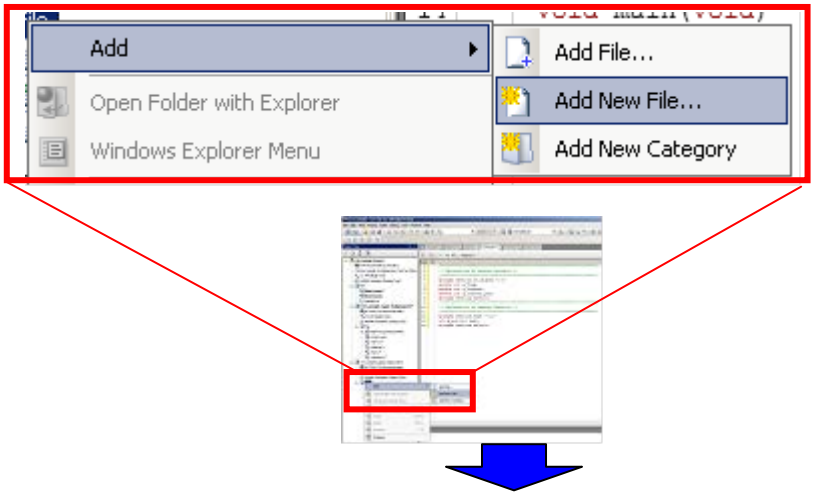
```
;-----------------------------------------------------------------------
;                       hdwinit_PE2
; Specify RAM addresses suitable to your system if needed.
;-----------------------------------------------------------------------
                        .section        ".text", text
                        .align          2
_hdwinit_PE2:
                        mov             lp, r14             ; save return address

                        ; clear Local RAM PE2
                        mov             LOCAL_RAM_PE2_ADDR, r6
                        mov             LOCAL_RAM_PE2_END, r7
                        jarl            _zeroclr4, lp

                        mov             r14, lp
                        jmp             [lp]
```

Editing the Program

3. Adding files

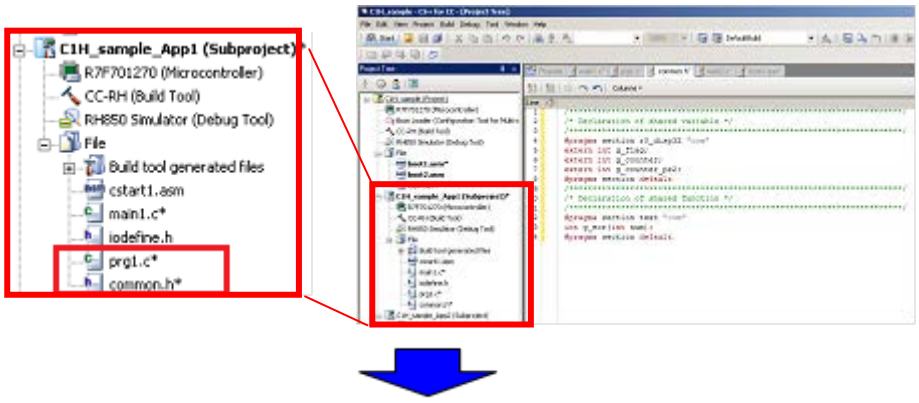
Add programs following the steps listed below.

Select a file of C1H_sample_App1 (subproject) in the Project Tree, right-click it to display the pop-up menu, and then select "Add New File..." from the pop-up menu.



Add the following two files. Set the C1H_sample¥C1H_sample_App1 folder as the place where the files are to be created.

C1H_sample_App1
- prg1.c
- common.h



Copy the following code and paste to prg1.c.

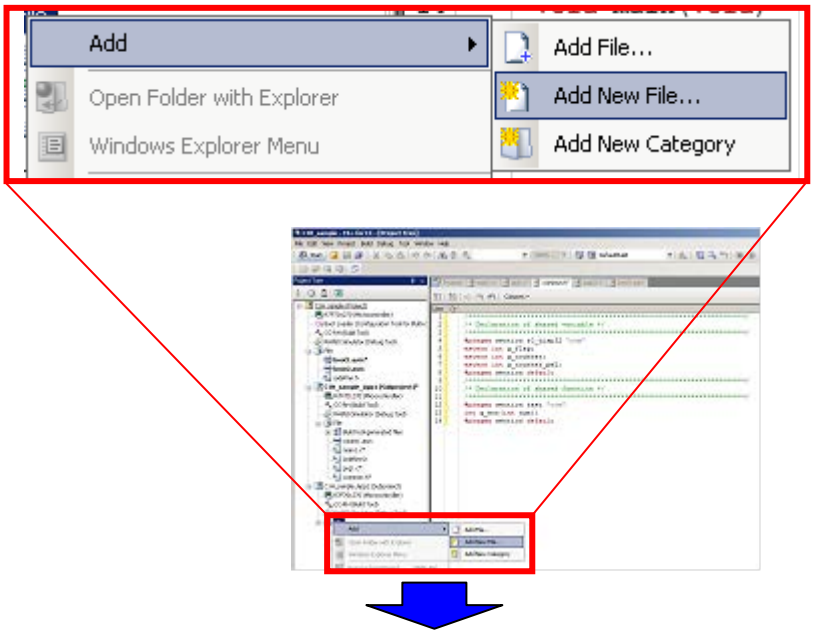
```
#include "iodefine.h"
#include "common.h"

#define LED1 PORT.P4.BIT.P4_11
#define LED2 PORT.P4.BIT.P4_10
#define LED3 PORT.P4.BIT.P4_9
#define LED4 PORT.P4.BIT.P4_8

/***********************************************************************/
/* Definition of unshared variable for PE1                             */
/***********************************************************************/
int led1 = 0;
int led2 = 0;
int led3;
int led4;

/***********************************************************************/
/* Definition of shared variable                                       */
/***********************************************************************/
#pragma section r0_disp32 "com"
int g_flag = 0;
int g_counter;
int g_counter_pe2;
#pragma section default

/***********************************************************************/
/* Definition of unshared function for PE1                             */
/***********************************************************************/
void hwinit()
{
                        PBG.FSGD3DPROT5.UINT32 = 0x000e02d5;
                        PORT.PM4.UINT16 = 0;

                        led3 = 0;
                        led4 = 0;

                        LED1 = led1;
                        LED2 = led2;
                        LED3 = led3;
                        LED4 = led4;
}

void set()
{
                        MEV.G0MEV0 = 0;
```

```
}

void outputLED1()
{
                        led1 = g_eor1(led1);
                        LED1 = led1;
}

void outputLED2()
{
                        led2 = g_eor1(led2);
                        LED2 = led2;
}

void outputLED3()
{
                        led3 = g_eor1(led3);
                        LED3 = led3;
}

void outputLED4()
{
                        led4 = g_eor1(led4);
                        LED4 = led4;
}

/************************************************************************/
/* Definition of shared function                                        */
/************************************************************************/
#pragma section text "com"
int g_eor1(int num)
{
                        return num^1;
}
#pragma section default
```

Copy the following code and paste to common.h.

```
/***********************************************************************/
/* Declaration of shared variable                                      */
/***********************************************************************/
#pragma section r0_disp32 "com"
extern int g_flag;
extern int g_counter;
extern int g_counter_pe2;
#pragma section default

/***********************************************************************/
/* Declaration of shared function                                      */
/***********************************************************************/
#pragma section text "com"
int g_eor(int num);
#pragma section default
```

Select a file of C1H_sample_App2 (subproject) in the Project Tree, right-click it to display the pop-up menu, and then select "Add New File..." from the pop-up menu.



Add the following two files. Set the C1H_sample¥C1H_sample_App2 folder as the place where the files are to be created.

C1H_sample_App2
 - prg2.c

Copy the following code and paste to prg2.c.

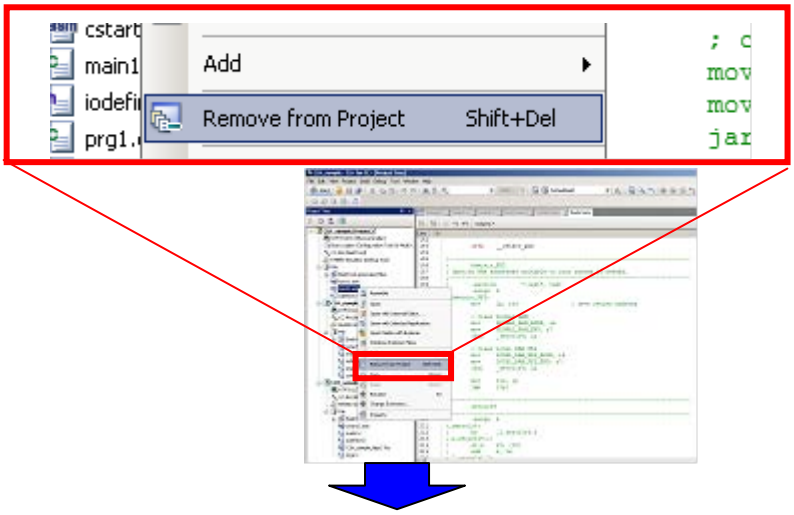
```
#include "iodefine.h"
#include "common.h"

#define LED1 PORT.P4.BIT.P4_11
#define LED2 PORT.P4.BIT.P4_10
#define LED3 PORT.P4.BIT.P4_9
#define LED4 PORT.P4.BIT.P4_8

/***********************************************************************/
/* Definition of unshared variable for PE2                             */
/***********************************************************************/
int led1;
int led2;
int led3 = 0;
int led4 = 0;

/***********************************************************************/
/* Definition of unshared function for PE2                             */
/***********************************************************************/
void hwinit()
{
                        led1 = 0;
                        led2 = 0;
                        MEV.G0MEV0 = 1;
}

void wait()
{
                        while(MEV.G0MEV0);
}

void outputLED1()
{
                        led1 = g_eor1(led1);
                        LED1 = led1;
}

void outputLED2()
{
                        led2 = g_eor1(led2);
                        LED2 = led2;
}

void outputLED3()
{
                        led3 = g_eor1(led3);
                        LED3 = led3;
}

void outputLED4()
{
                        led4 = g_eor1(led4);
                        LED4 = led4;
}
```

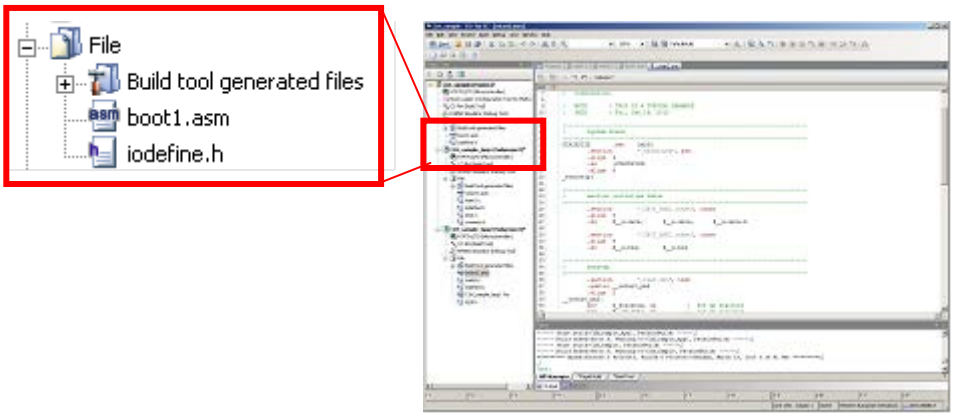
Editing the Program

4. Deleting files

Delete programs following the steps listed below.

Select boot2.asm of C1H_sample (project) in the Project Tree, right-click it to display the pop-up menu, then select "Remove from Project" from the pop-up menu.



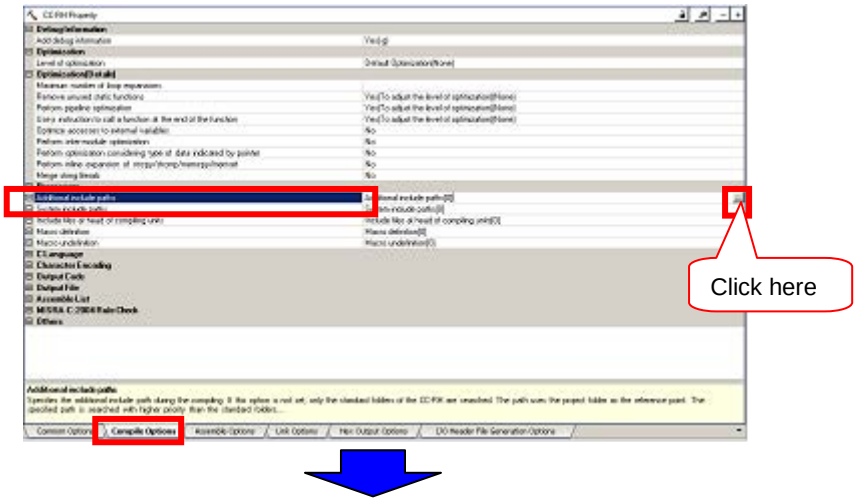


Editing the Program

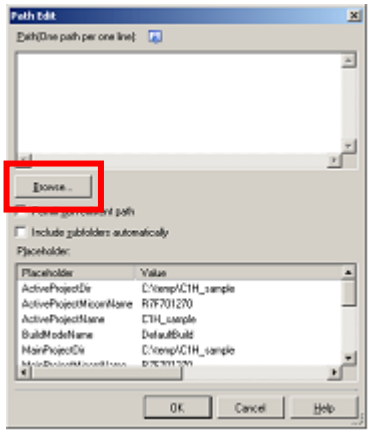
5. Changing the property of the compiler (CC-RH)

Change the property of CC-RH following the steps listed below.

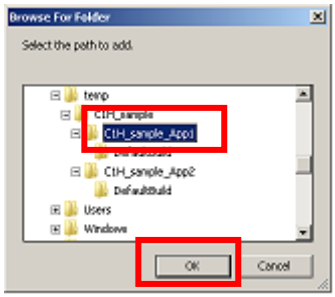
Display the property of CC-RH of C1H_sample_App2 (subproject) in the Project Tree, then click the Add button to add an additional include path as a compile option.



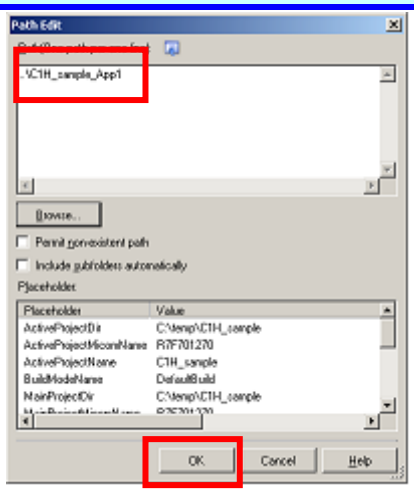
This opens the Path Edit dialog box. Click the Browse button.



This opens the Browse For Folder dialog box. Select the following folder shown below.
C1H_sample¥C1H_sample_App1



After confirming that the folder has been added, click the OK button.

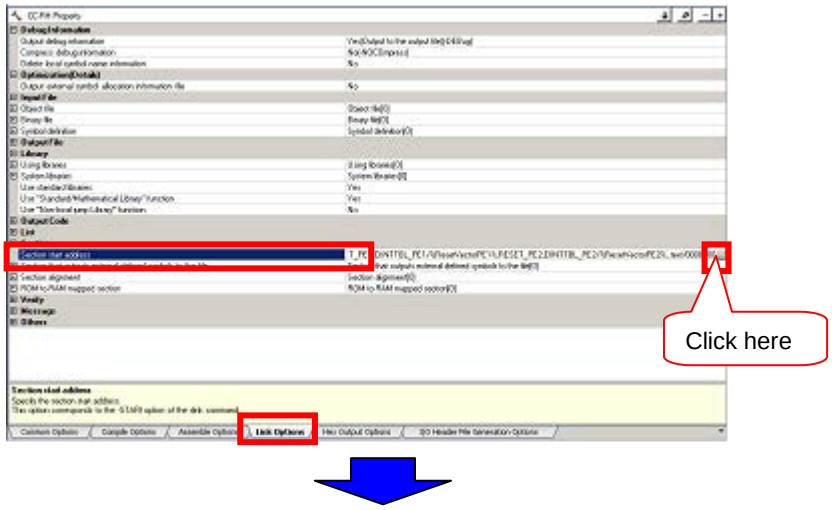


Editing the Program

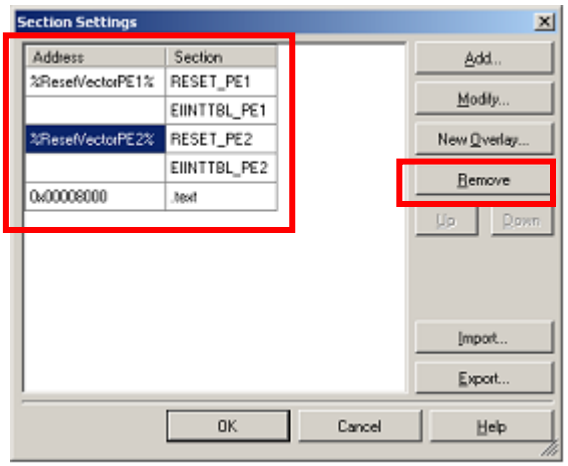
6. Deleting the section

Change the section start address following the steps listed below.

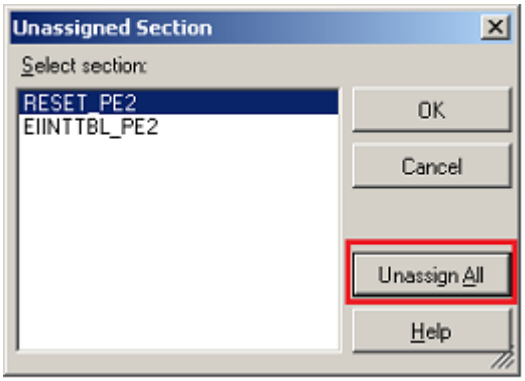
Display the property of CC-RH of C1H_sample (project) in the Project Tree, then click the Edit button at the section start address of the section group as a link option.



This opens the [Section Settings] dialog box. Select "%ResetVectorPE2%" and click the [Remove] button.



This opens the [Unassigned Section] dialog box. Click the [Unassign All] button.

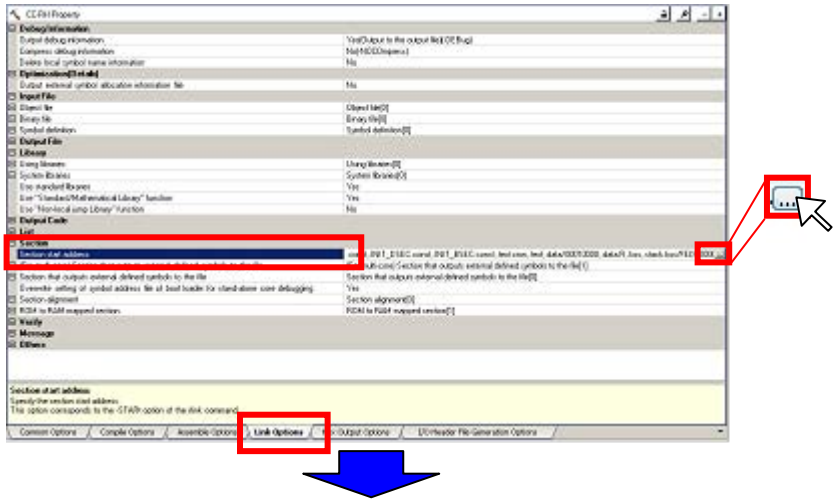


Editing the Program

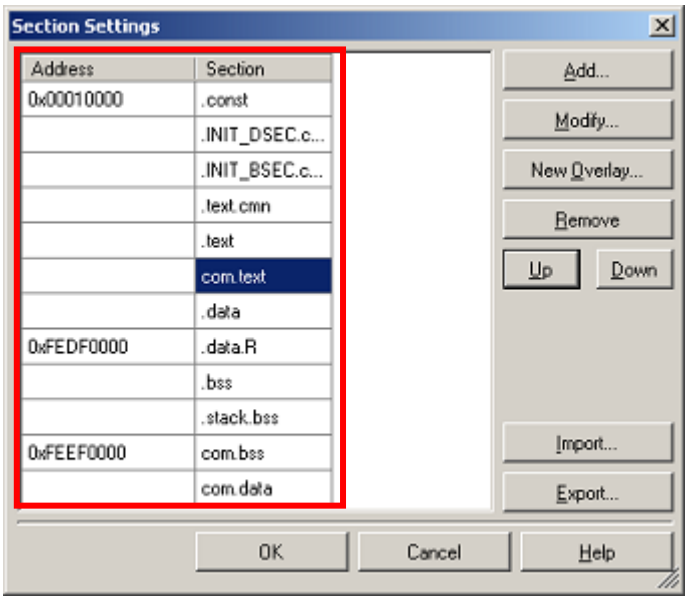
7. Editing the section start address

Follow the steps listed below to change the section start address.

Open the property of CC-RH for C1H_sample_App1 (subproject) in the Project Tree, then click the [Edit] button at the section start address of the section group on the [Link Options] tabbed page.



This opens the [Section Settings] dialog box. Click the [Add] button to add three sections "com.text", "com.bss", and "com.data". Then make and edit settings as shown below.

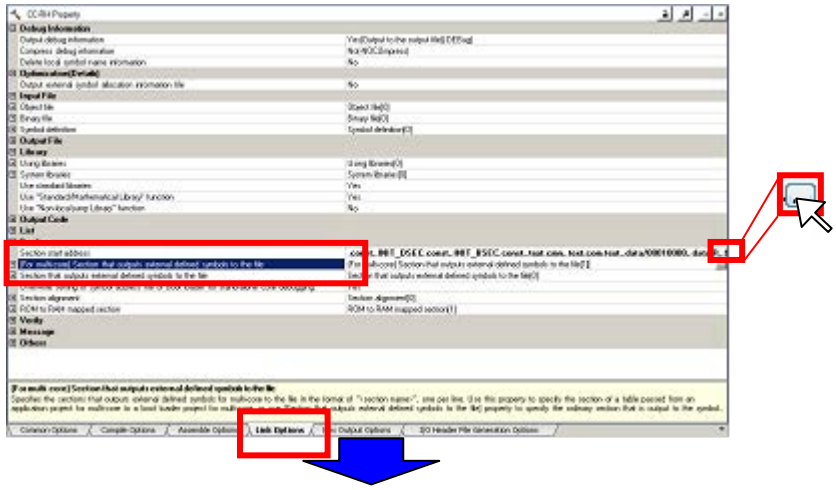


Editing the Program

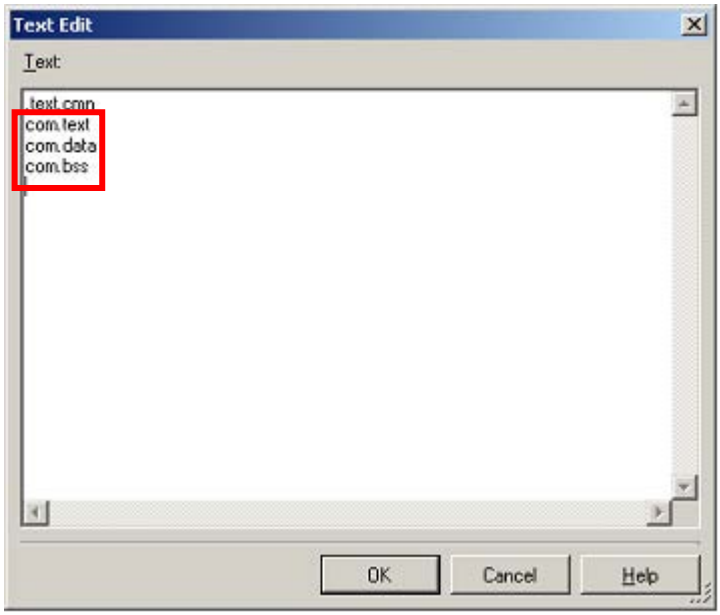
8. Editing the option for outputting external defined symbols for a multi-core to a file

Follow the steps listed below to change the option for outputting external defined symbols for a multi-core.

Open the property of CC-RH for C1H_sample_App1 (subproject) in the Project Tree, then click the [Edit] button at "Section that outputs external defined symbols to the file" of the section group on the [Link Options] tabbed page.



This opens the [Section Settings] dialog box. Click the [Add] button to add three sections "com.text", "com.bss", and "com.data". Then make and edit settings as shown below.

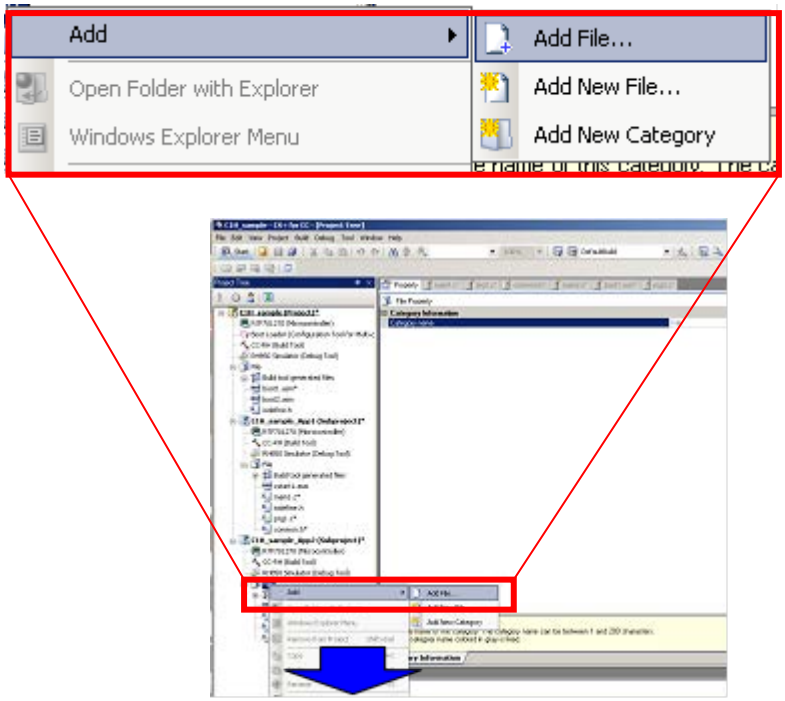


Editing the Program

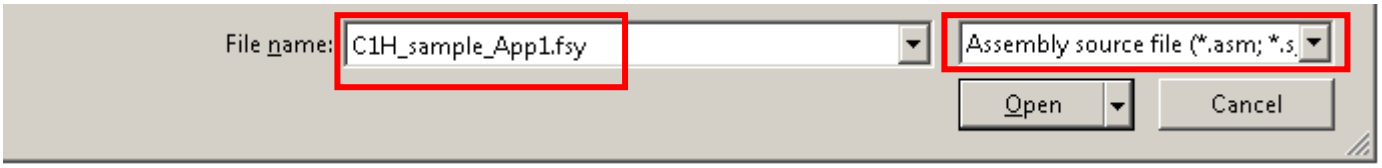
9. Adding files

Follow the steps listed below to add the C1H_sample_app1.fsy file to C1H_sample_app2.

Select a file of C1H_sample_App2 (subproject) in the Project Tree, right-click it to display the pop-up menu, and then select "Add New File..." from the pop-up menu.



Select the assembly source file, open C1H_sample_App1 and DefaultBuild, and then select "C1H_sample_App1.fsy" to add the file.

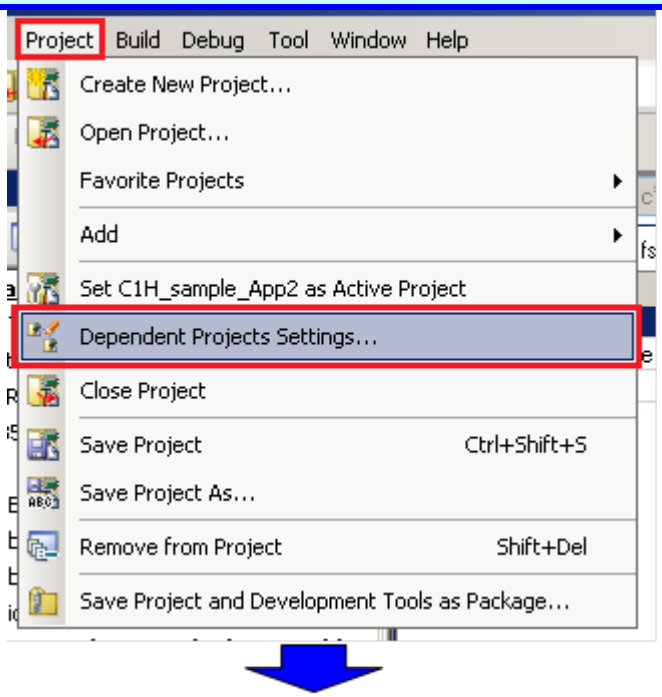


Editing the Program

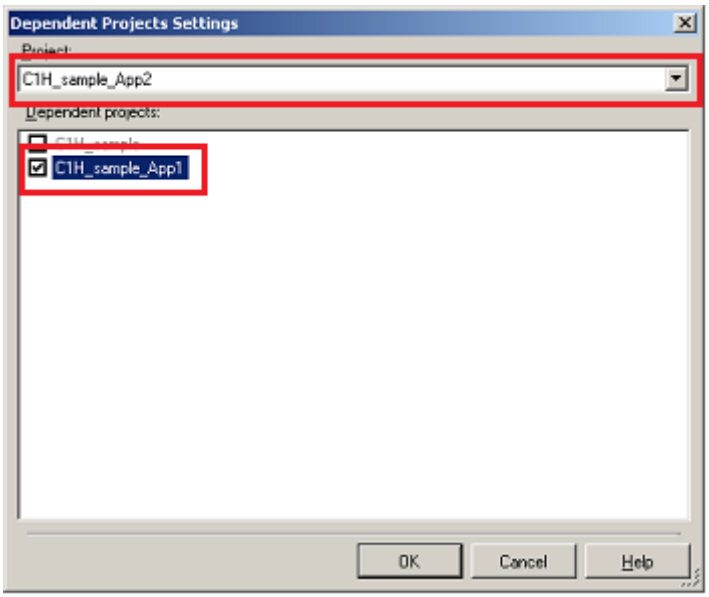
10. Setting dependent projects

Follow the steps listed below to set dependent projects.

Select [Dependent Projects Settings...] from the [Project] menu.



Select "C1H_sample_App2" for [Project] and then select "C1H_sample_App1".



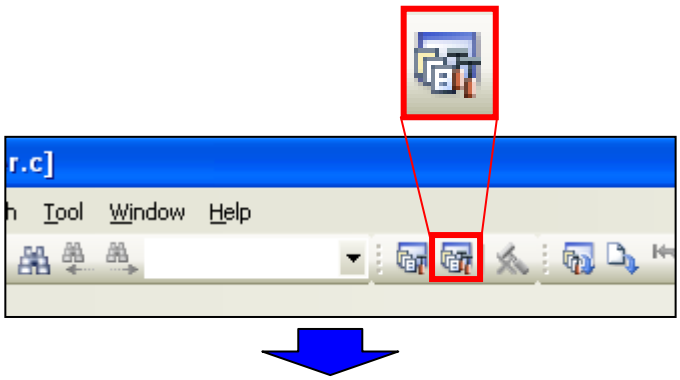
Rebuilding a Program

Rebuild the program of the loaded sample project.

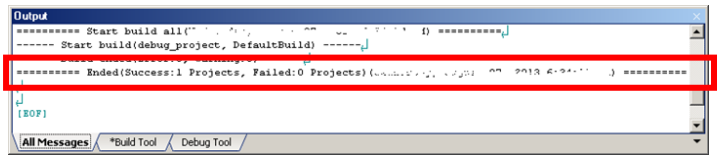
1. Building a project

In this step, rebuild the program of the loaded sample project.

Click the [Rebuild Project.] button.



Check to see if the rebuild process has been completed correctly. If the build process has been completed correctly, a load module file is created.



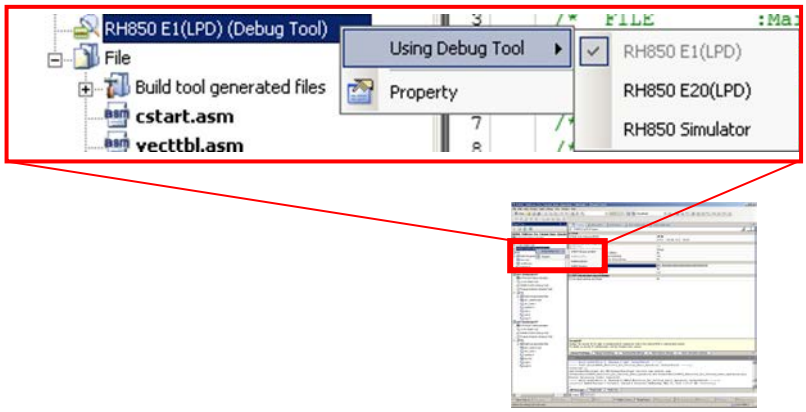
Connecting a Debugger and Downloading

In this section, you debug the program using the E1. First, make preparations for debugging.

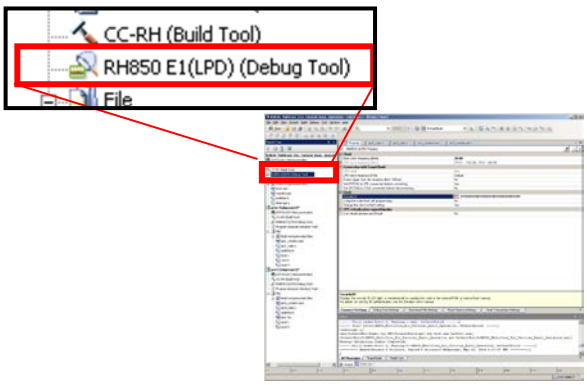
1. Selecting a debug tool

In this step, select [RH850 E1(LPD) (Debug Tool)] as a debug tool to be used.

Right-click on the Debug Tool in the Project Tree and select [Using Debug Tool] -> [RH850 E1(LPD)].



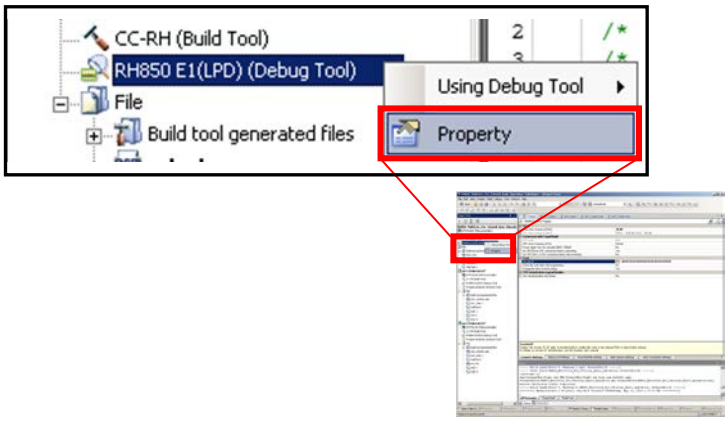
[RH850 E1(LPD) (Debug Tool)] is selected as a debug tool.



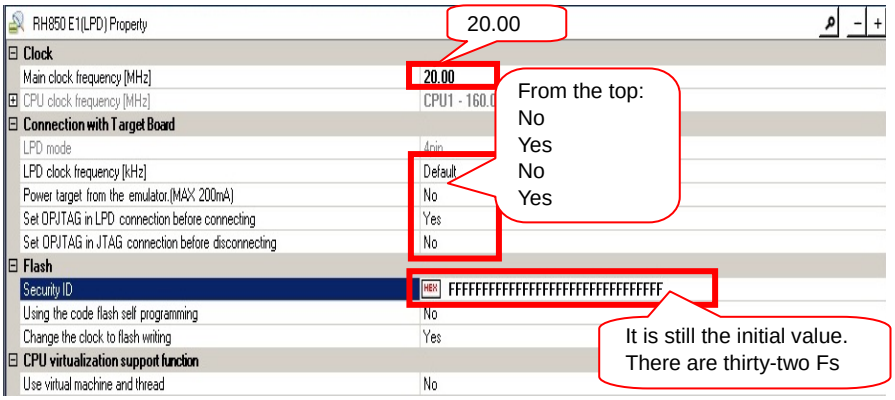
Connecting a Debugger and Downloading

2. Setting E1 for connection to the target board

Right-click the debug tool in the Project Tree to select [Property].



Make settings as shown below in the [Connection with Target Board] tab.

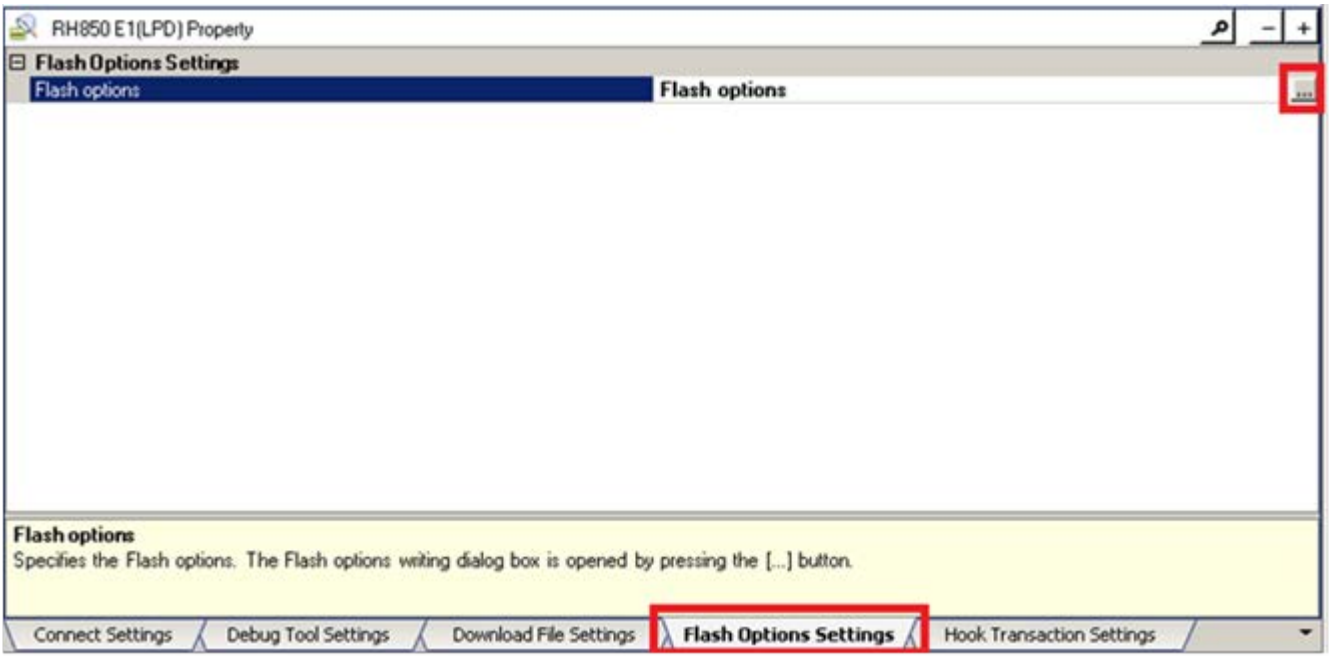


Tip

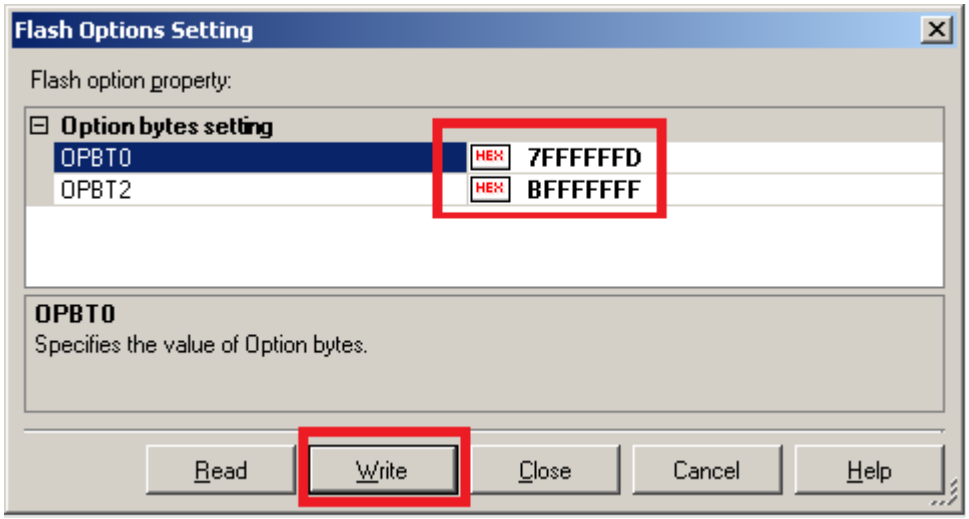
About the security ID

The 128-bit ID code can be written to the microcontroller so that the flash memory contents are not read by an unauthorized user. If the code that is input by the user when the debugger is started does not match the ID code written to the microcontroller, flash memory cannot be accessed. Settings should be made by a flash programmer. When a blank product (all flash memory contents are erased) is used, only F should be input for the security ID.

Select the [Flash Options Settings] tab and click the [Edit] button.



Make the following settings in [Option bytes setting] and click the [Write] button.

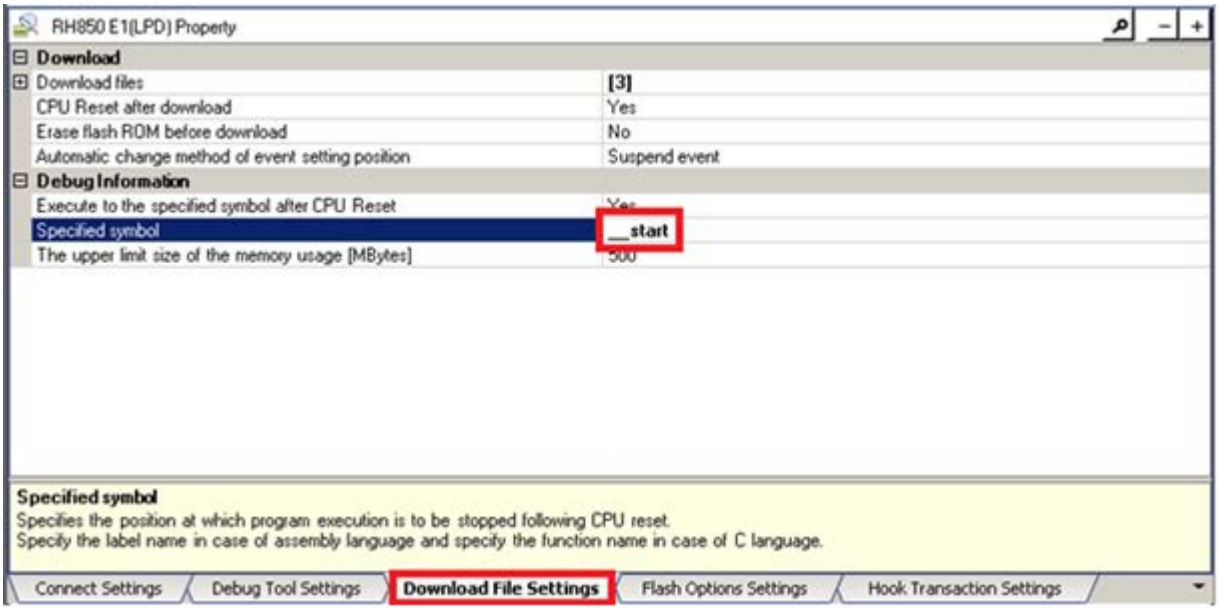


Tip

About option bytes

In flash memory, there is an extended area (option bytes) for holding data specified by the user for various purposes. In the RH850/C1H microcontroller, not only are settings for the debugging interface made, but settings for WDT-related features and the operating mode and startup area of the microcontroller are to be made.
When the program of this tutorial is used, set the OPBT0 register to H'7FFFFFFD and the OPBT2 register to H'BFFFFFFF.

Select the [Download File Settings] tab and set the Spesified symbol to “__start”.

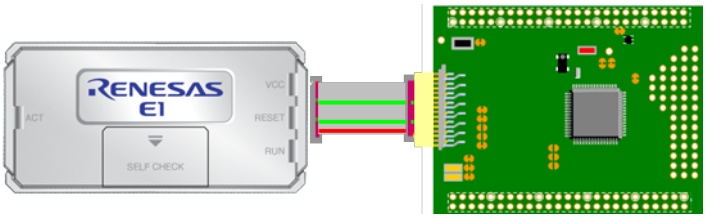


Connecting a Debugger and Downloading

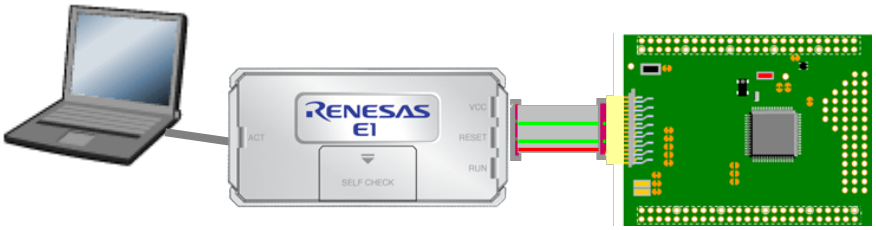
3. Connecting E1

In this step, on-chip debugging is performed using E1.

Connect E1 to the RH850/C1H board (RH850/C1X EVALUATION BOARD). Align pin 1 of the connector.



Connect E1 to the PC. ("Found New Hardware Wizard" appears when E1 is connected for the first time. Select "Install software automatically" and install the USB driver following the instructions.)



Turn on the power of RH850/C1x EVALUATION BOARD.

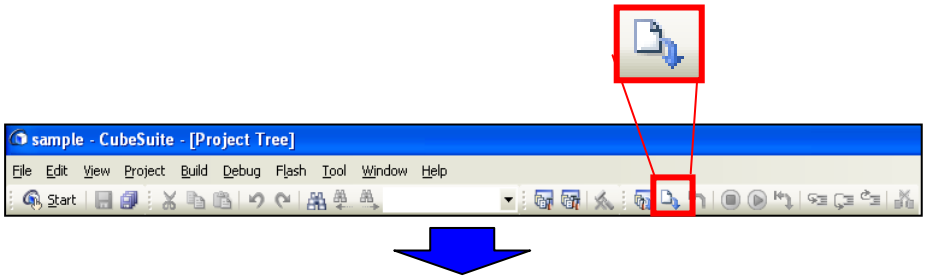
Connecting a Debugger and Downloading

4. Downloading a load module file to E1

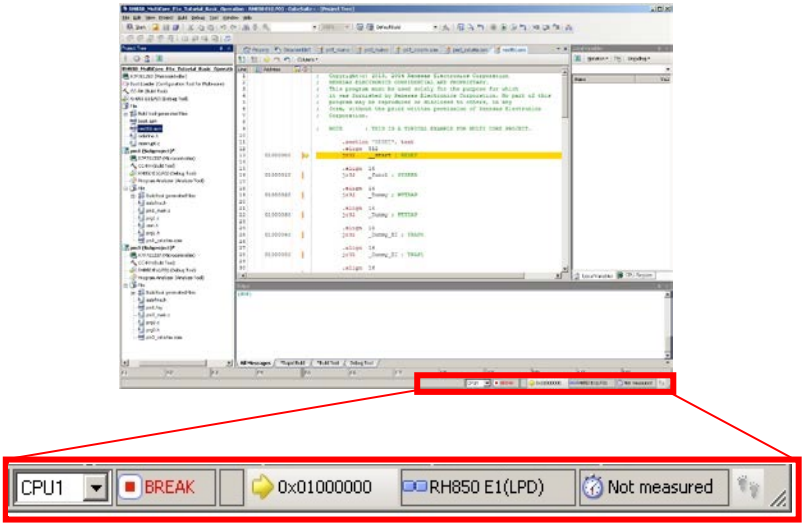
In this step, download the load module file generated by the build process to the target microcontroller.

When download is complete, the program can be executed.

Click the Download button in the menu.



The status bar of the main window changes as shown below.



Switching Cores

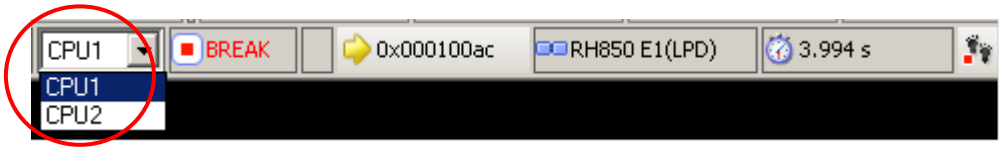
Now that download of the load module file to the target is complete, let's switch the core to be debugged.

1. Switching the core

There are two methods for switching the core to be debugged.

a. Switch the core from the status bar

The core can be switched using the drop-down list on the status bar of the main window.



b. Switch the core from the Debug Manager panel

Selecting [View] menu -> [Debug Manager] opens the Debug Manager panel. The core can be switched in the Debug Manager panel.

Here, the state of CPU1 being selected should not be changed.

Tip

Core to be debugged

Running or stopping of the program cannot be performed by only one core running or stopping the program.
Running or stopping of the program must be performed by both cores operating in a coordinated manner.
When CPU1 is set as the core to be debugged, referencing or changing memory by CS+ is effective only for CPU1. To perform operations for the other core, the core to be debugged has to be switched.

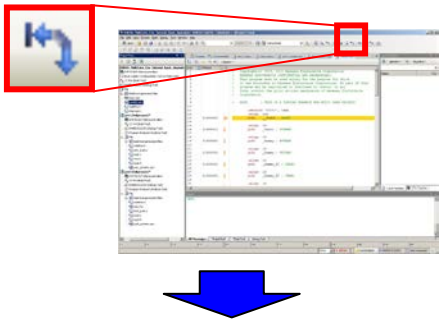
Running and Stopping the Program

Now that download of the load module file to the target is complete, the program can be executed. First, run and stop the program.

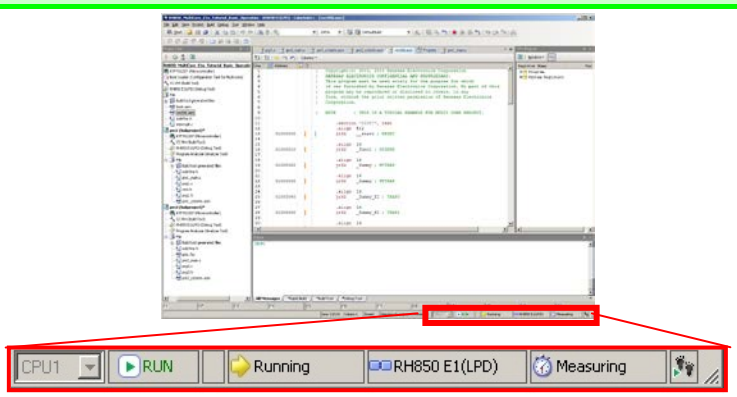
1. Running the program

Run the program after resetting the CPU.

Click the [Restart] button in the menu.



This runs the program and displays [RUN] in the status bar.



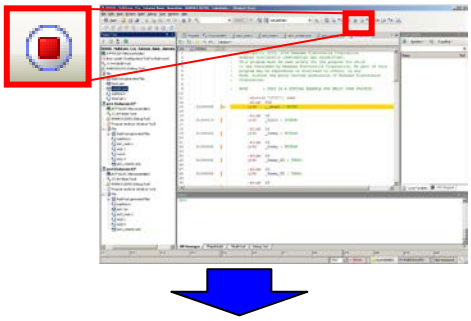
LED1 and LED2 on RH850/C1x EVALUATION BOARD light alternately.

Running and Stopping the Program

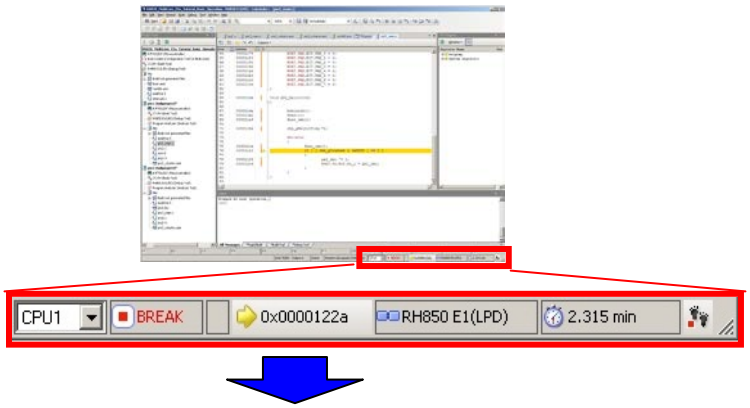
2. Stopping the program

In this step, stop the program.

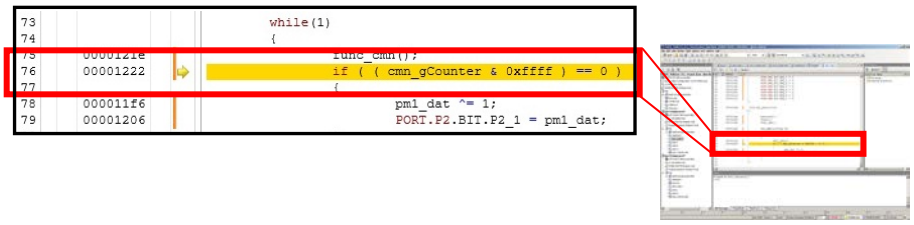
Click the Stop button.



This stops the program and displays [BREAK] in the status bar.



The source code line where the program stopped (current position of the program counter (PC)) is highlighted in yellow.

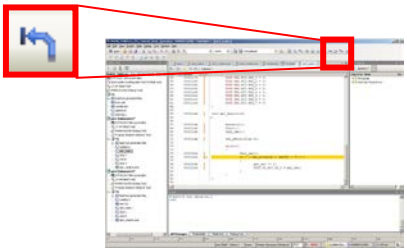


Running and Stopping the Program

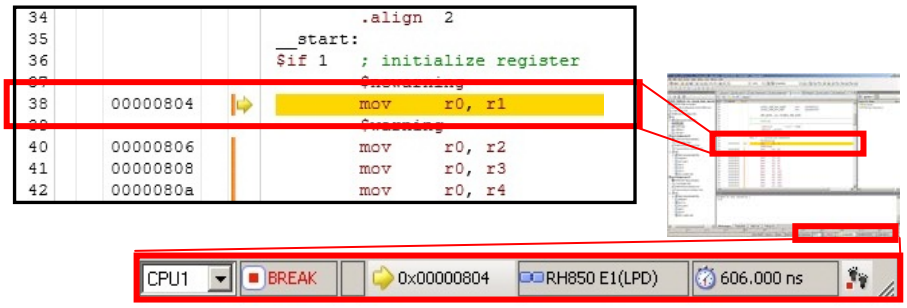
3. Resetting the program

In step 1, you have reset and run the program using a single button, but you can also carry out the reset operation independently.

Click the Reset button.



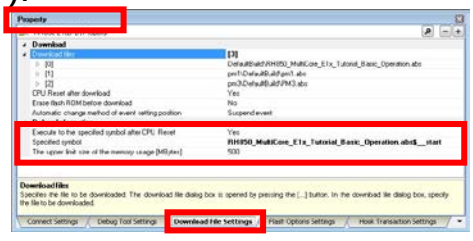
The program is reset and the program counter (PC) goes back to the start of the program.



Tip

About the program counter

The program counter (PC) is a control register that holds information on the next program address. When the RH850/C1H microcontroller generates a reset signal, 00000000H is set in the PC for user mode and 01000000H is set in the PC for user boot mode. In the program of this tutorial, since the program is set to run up to the __start function after a reset, the program enters the break state after executing the __start function. Such kind of behavior can be changed in the [Download File Settings] tab of the Property panel of RH850 E1(LPD).

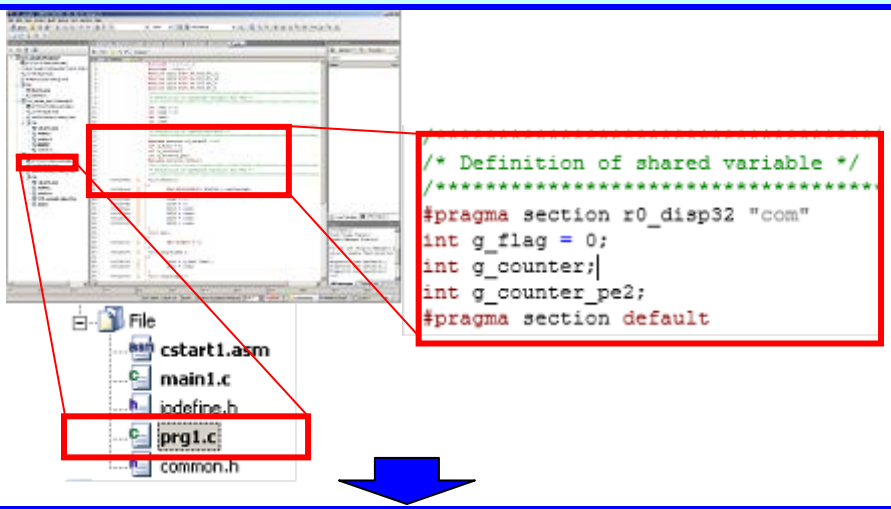


Referring to a Variable

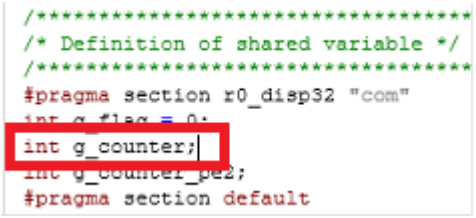
Watch function

By registering a variable as a watch-expression, it is possible to view the value of that variable. Here, register the two shared variables (variables which are located in a shared area that can be referenced from both the CPU1 core and CPU2 core) "g_counter" and "g_counter_pe2" as watch-expressions and confirm that the values of the variables are incremented. "g_counter" and "g_counter_pe2" are variables that are incremented in the program by the CPU1 core and CPU2 core, respectively.

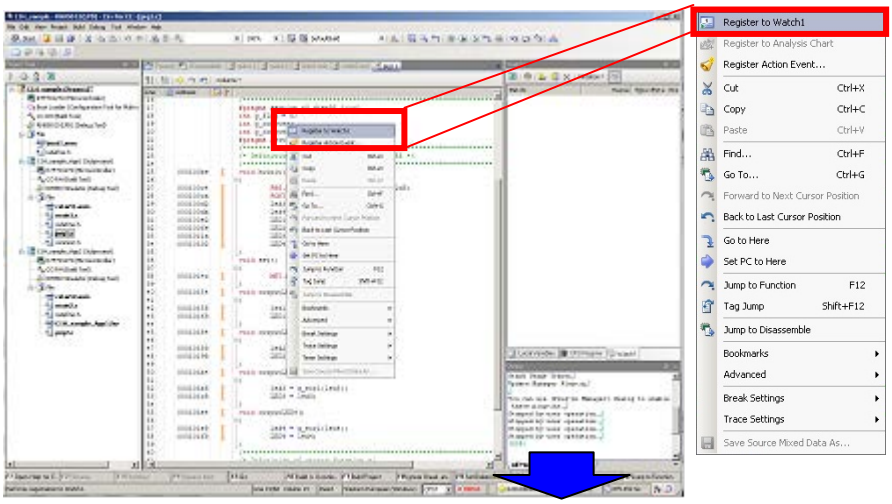
In the Project Tree, double click "prg1.c" to display the source code file.



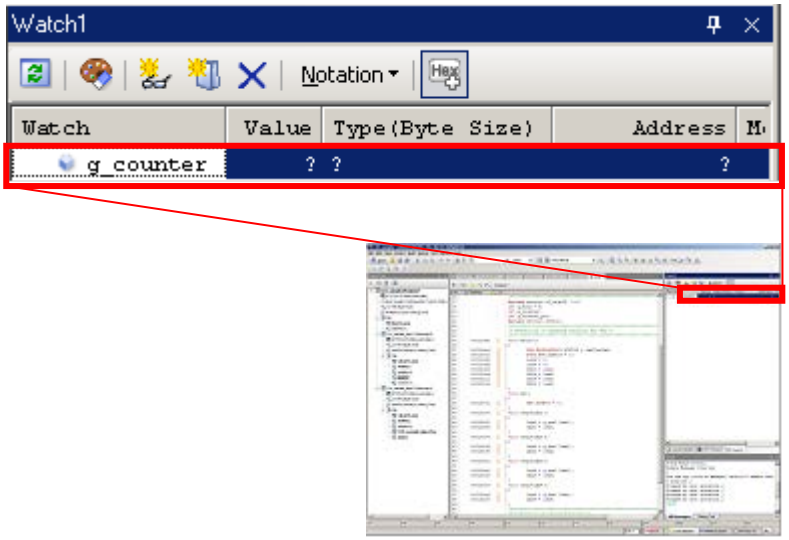
Select variable "g_counter" in the source code.



While "g_counter" is selected, right-click the mouse and select [Register to Watch1] from the menu.

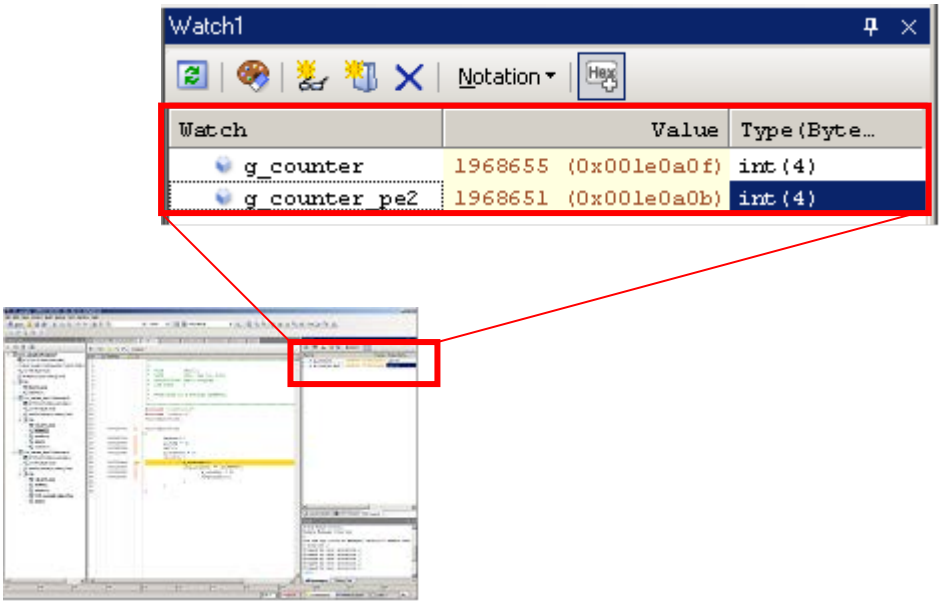


This opens the Watch panel where you can confirm that the variable is now registered. The current value of "g_counter" is ?.



In a similar manner, register "g_counter_pe2" in the Watch panel from main2.c.

Click the [Restart] button in the menu. After several seconds have passed, click the [Stop] button.


Watch1
Notation
Hex
Watch
Value
Type(Byte...
g_counter
1968655 (0x001e0a0f)
int(4)
g_counter_pe2
1968651 (0x001e0a0b)
int(4)

Setting a Breakpoint

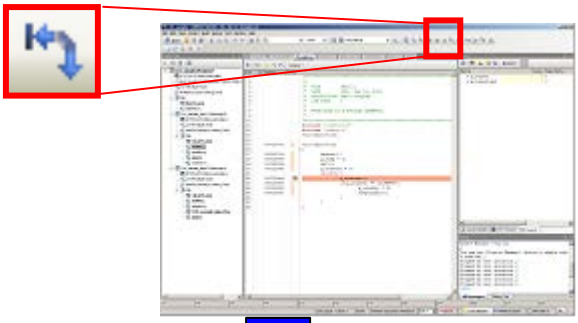
Setting a breakpoint

If you want to stop the program at a specific position intentionally in the source code, you can break the program before executing the instruction at the specified address ("before execution" break) by setting a breakpoint.

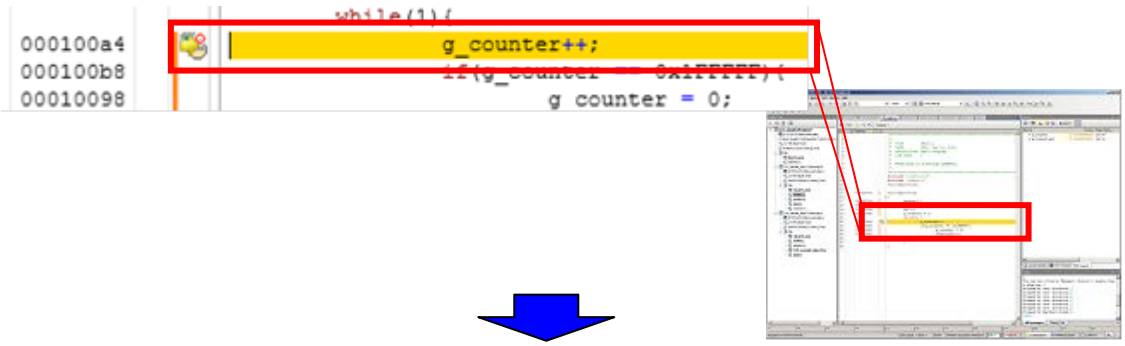
Let's see how the variable (g_counter) that was registered as a watch-expression changes by running the program and causing it to break.

Click the empty column to the left of the desired line in the source code as shown below. A hardware break is set and the line is highlighted in red.

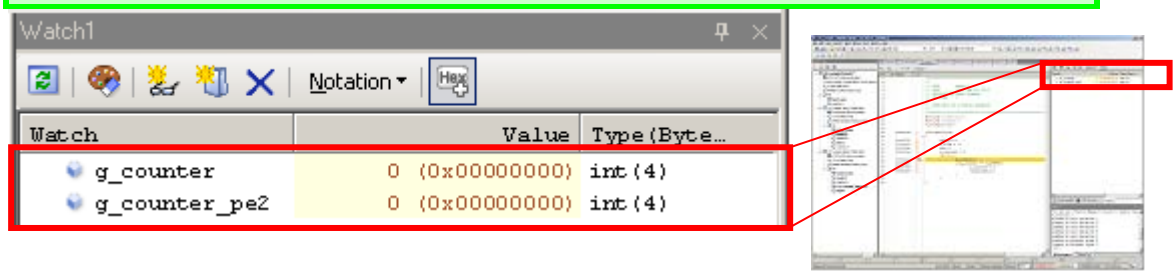
Click the [Restart] button in the menu.



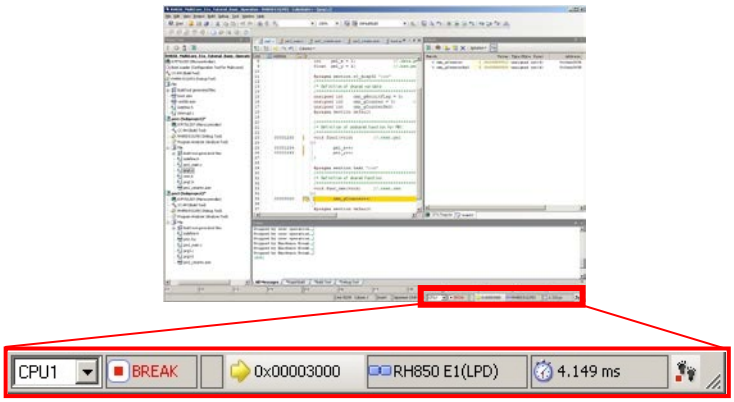
The program breaks at the line where the break has been set and the breakpoint line is highlighted in yellow.



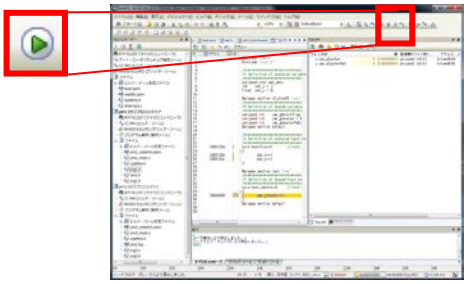
See the Watch panel and confirm that the value of [g_counter] is counted up to 0x0.



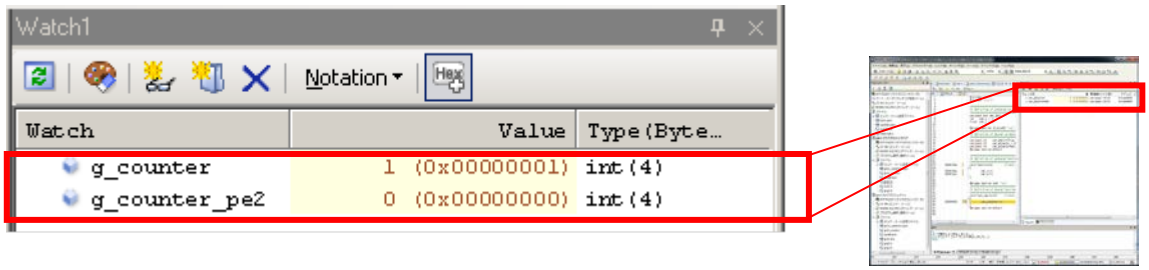
The status bar shows [BREAK] and the PC value at the break.



Click the [Go] button in the menu.



The program breaks again at the line where the break has been set. See the Watch panel and confirm that the value of [g_counter] is counted up to 0x1.

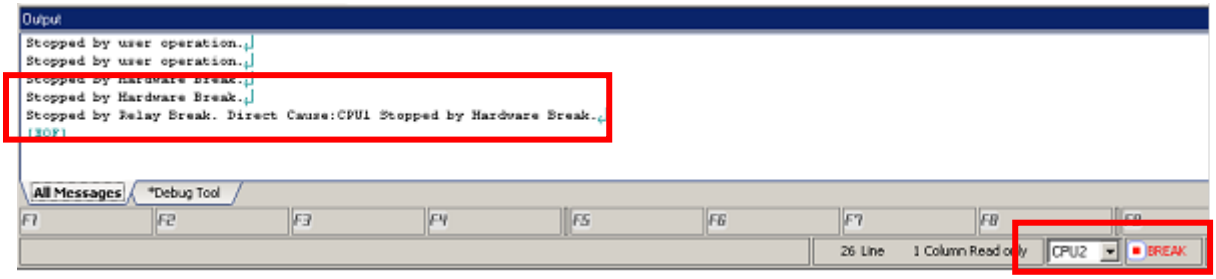


Tip

Break in a multi-core device

Normally, the position where a break occurred in the program is displayed when a break occurs. In a multi-core device, if a break was caused by a break source of another core (core not being debugged), a break occurs in the target core (core being debugged) at an address where no break condition has been set. The break source can be checked in the Output panel.

In the example below, the Output panel shows that a break (relay break) in another core is the break source.



Acquiring the Execution History

Acquiring the execution history

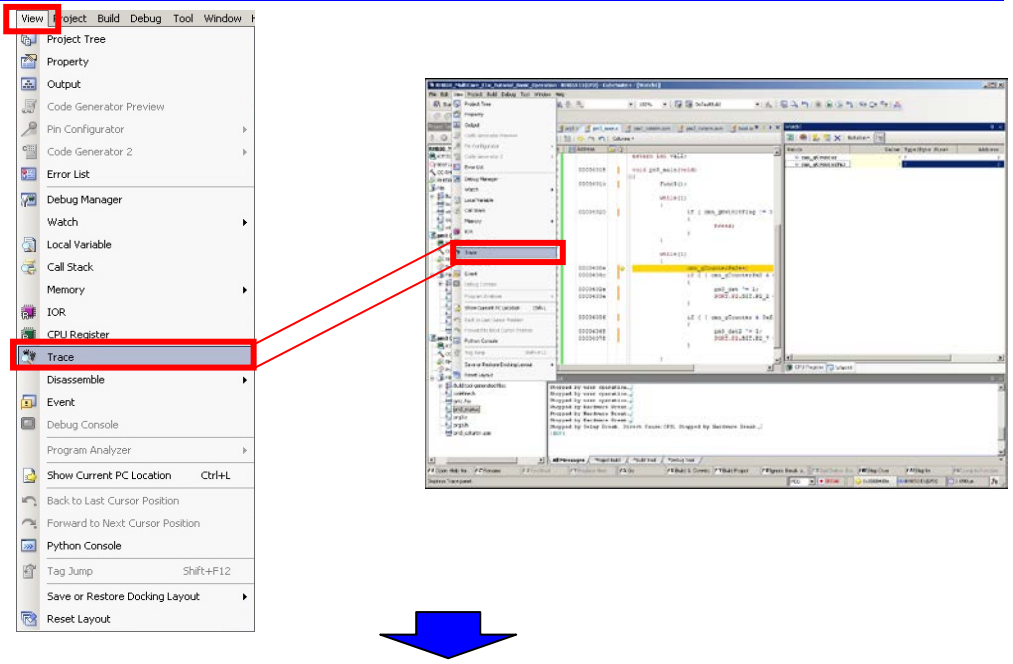
In general, the execution history of the program is called trace information. If a program gets out of control, it is extremely difficult to investigate the cause only from the memory contents or stack information after the runaway occurred. However, using the trace function and analyzing the acquired trace information enables the process until the runaway occurred to be directly investigated.

Collecting the Execution History

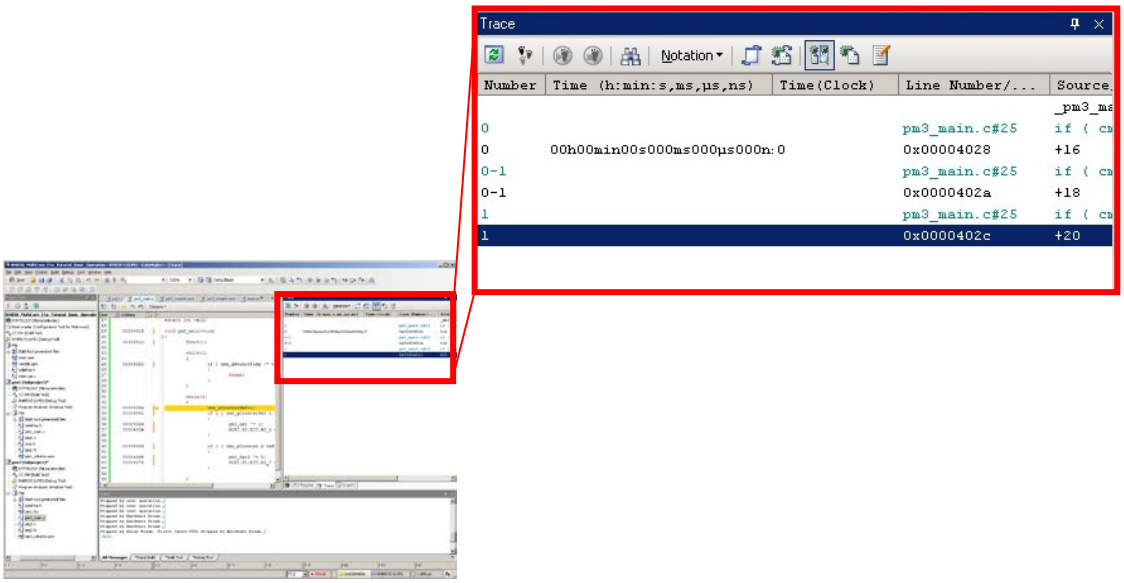
Setting trace operation

When the trace function starts recording, the execution process of the program currently running is recorded in trace memory (when program execution stops, the trace function also stops automatically).
Settings related to tracing need to be made in advance to use the trace function.

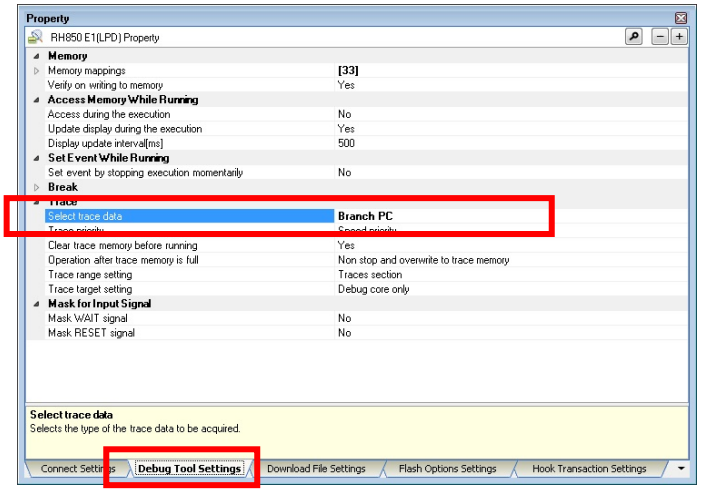
Select [Trace] from the [View] menu.



This opens the Trace panel.

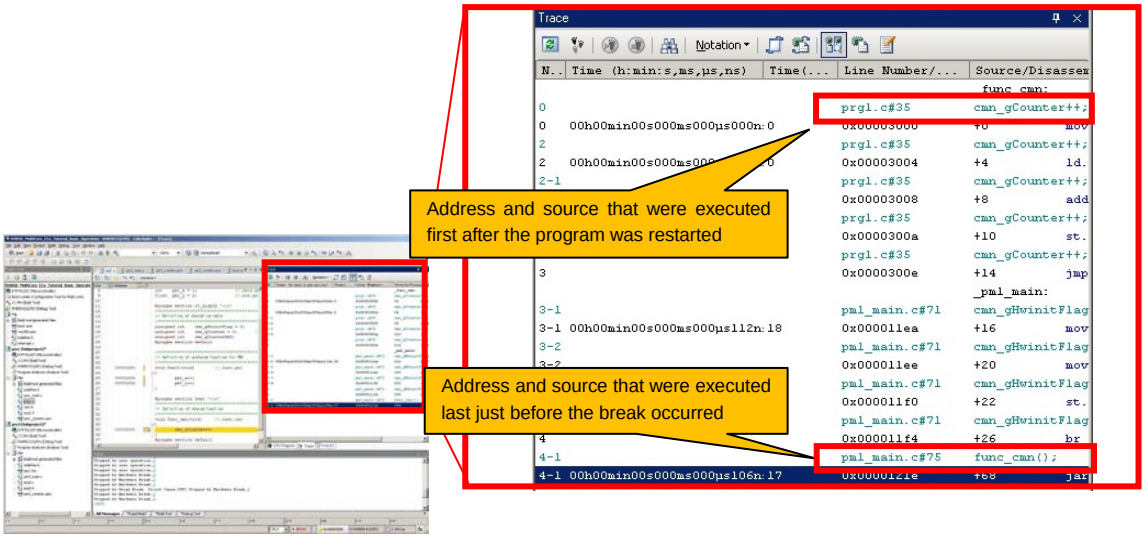


Settings for tracing can be made in the [Trace] category on the [Debug Tool Settings] tab of the Property panel.
Select the [Debug Tool Settings] tab and make the settings as shown below.



Click the [Go] button in the menu.

A break occurred and the execution history is displayed in the Trace window.

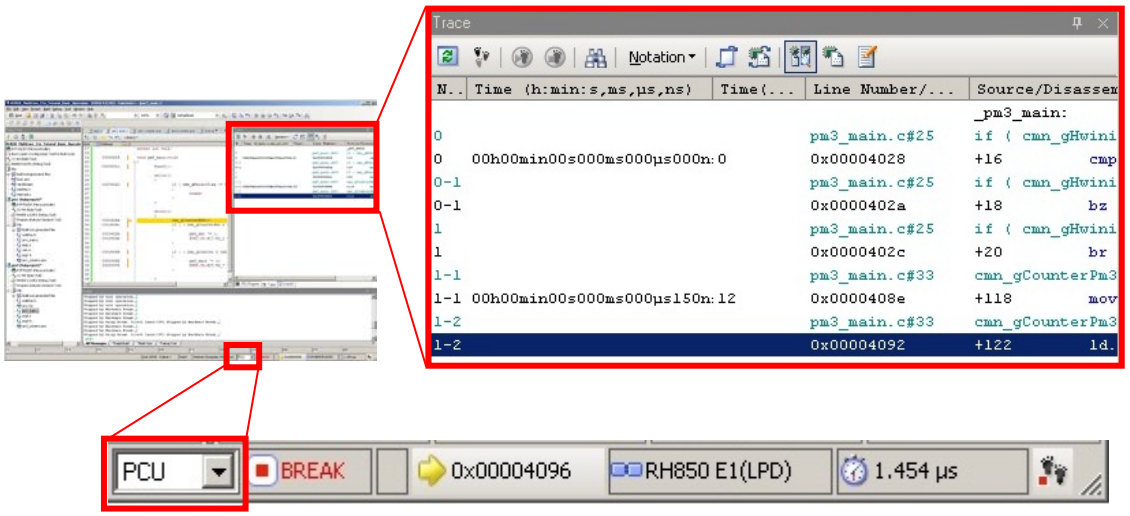


Tip

Tracing in a multi-core device

When trace data is acquired with the core to be debugged set to CPU1, only information for the CPU1 side can be observed. In order to acquire trace data for the CPU2 side, execute the program again after switching the core to be debugged to CPU2.

After trace data has been acquired with the core to be debugged set to CPU1, even though the core to be debugged is switched to CPU2, trace data for the CPU2 side cannot be observed.



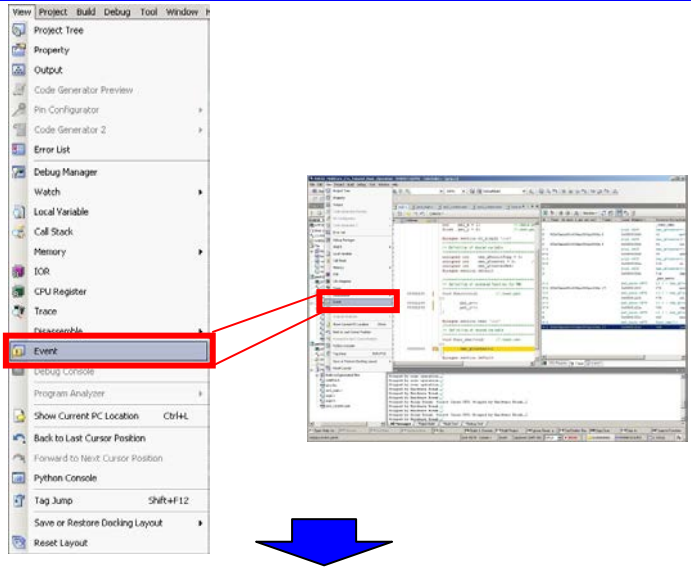
Canceling a Break

Canceling a Break

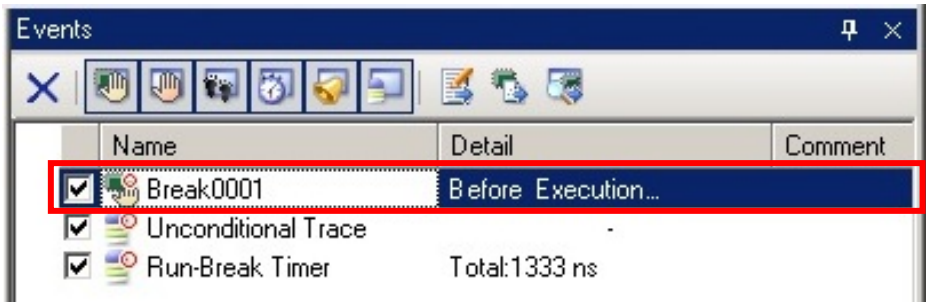
In this section, you cancel the break which was set in the previous section. The previously set break is set as a hardware break. A hardware break is registered as an event. Deleting the event will cancel the hardware break.

An event is an operation of the microcontroller such as fetch, read, and write. The event can be used as an action trigger to enable debugging functions such as setting a breakpoint or tracing. The hardware break that was set in the previous section was an event causing the program to break when a particular address is fetched (before it is executed).

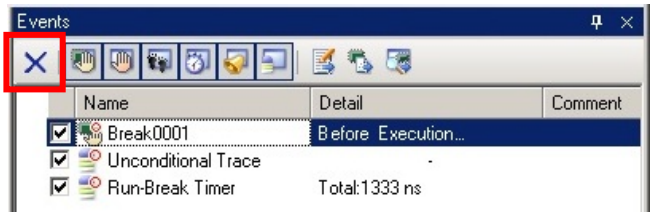
Select [Event] from the [View] menu.



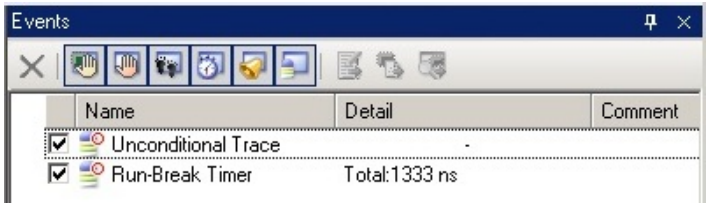
This opens the Event panel. Select [Break0001].



Left-click the Delete button. This cancels the access break.



Confirm that [Break0001] has been deleted from the Event panel.

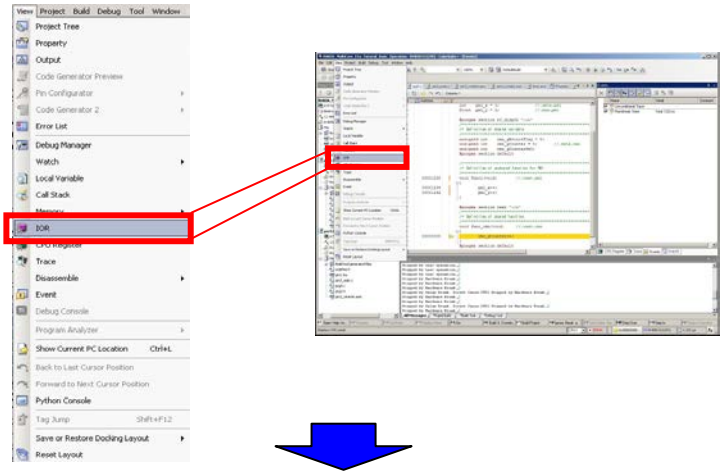


Displaying Special Function Registers (IORs)

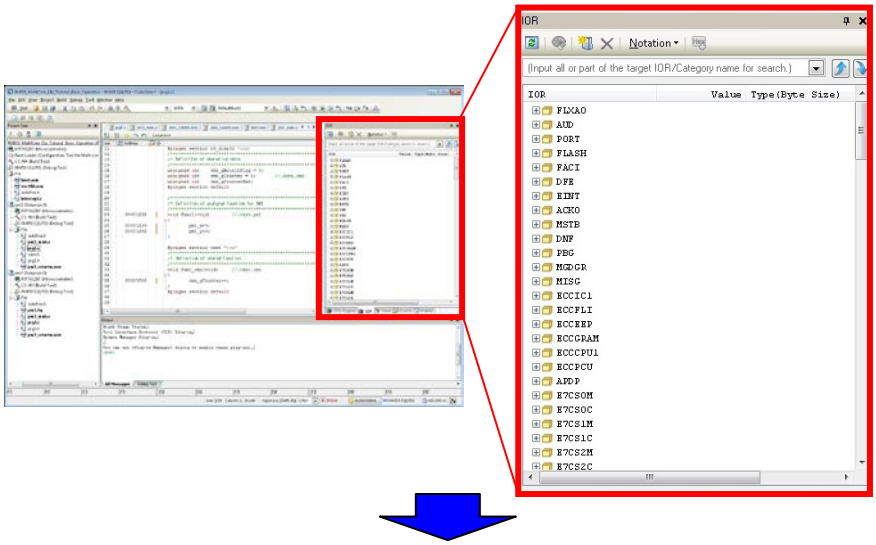
Displaying IORs

In this section, you observe the values of registers that implement peripheral functions built in the microcontroller. First, let's make the panel float so that it is easier to view.

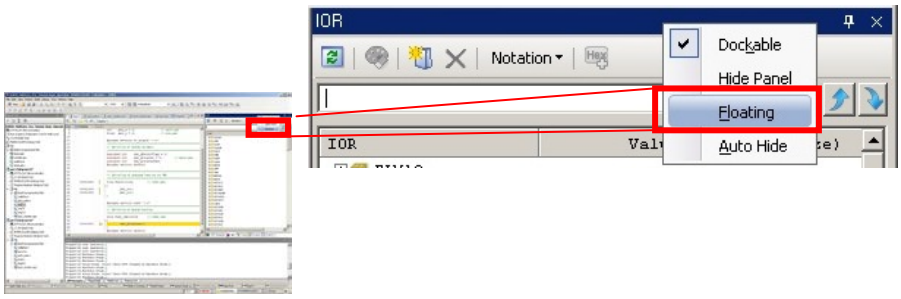
Select [IOR] from the [View] menu.



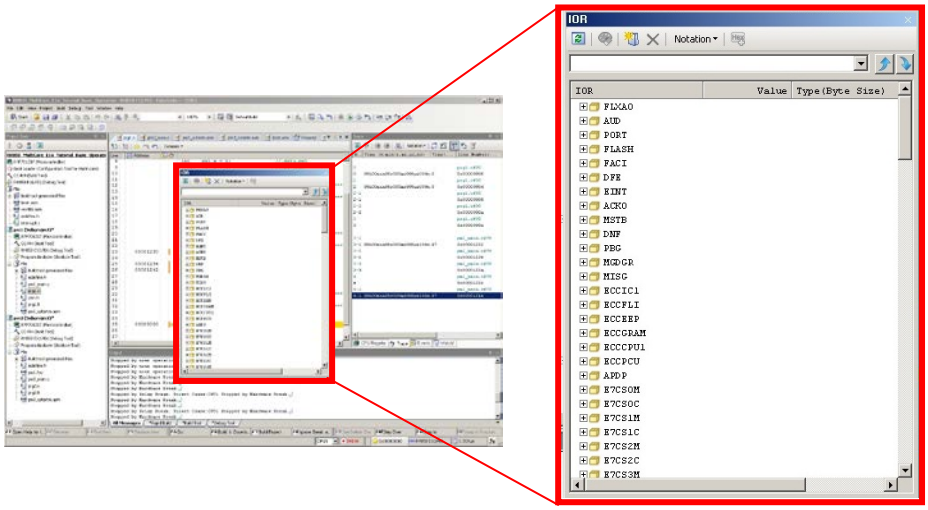
This opens the IOR panel.



Right-click on the title bar of the IOR panel and check [Floating].



The IOR panel enters the floating state and it becomes easier to view.



Tip

About display of IOR Bits

The IOR panel does not support displaying of IOR bits. Therefore, in order to check the bits of an IOR, that IOR has to be registered in a Watch panel so it can be referenced.

Select [Add New Watch] from the context menu in the desired Watch panel and input a watch-expression. To specify register bits, enter as shown below.

AAA0.BBB.CCC

<Module name>.<Register name>.<Bit name>

[Example]

Watch-expression for registering the P2_1 bit in the P2 register of a (general I/O) port in a Watch panel:

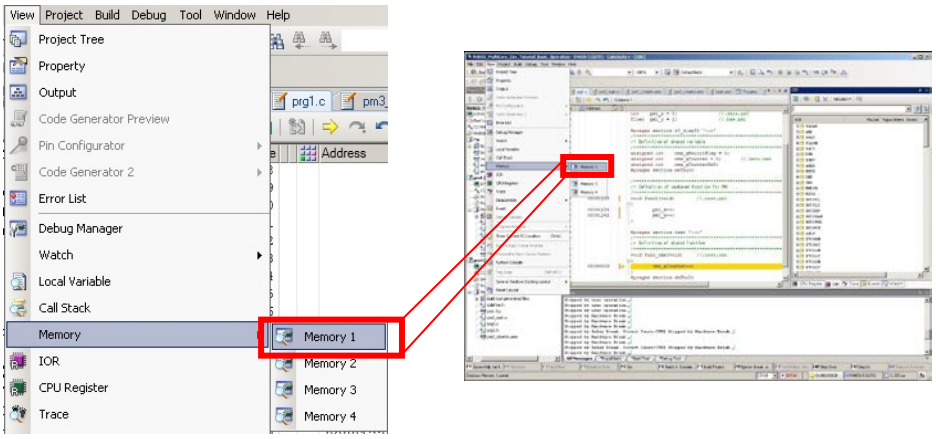
PORT.P2.P2_1

Displaying Memory

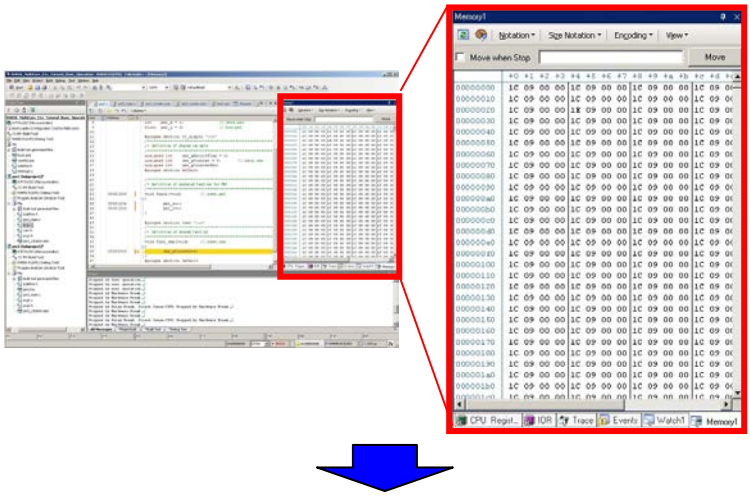
Displaying the Memory panel

The Memory panel displays the state of memory. In this example, two of the four Memory panels are displayed. If [Memory 1] and [Memory 2] are displayed at the same time, they are tabbed by default, making it impossible to view both panels at same time. Let's dock these two panels so that they can be viewed together.

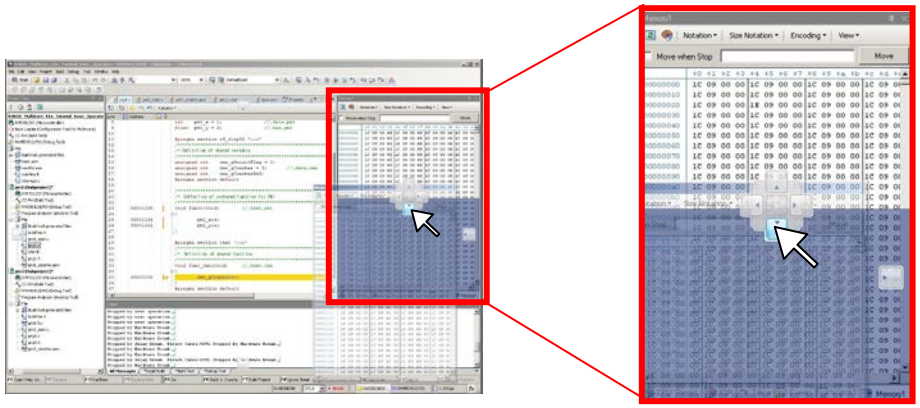
Select [Memory 1] from the [View] menu.



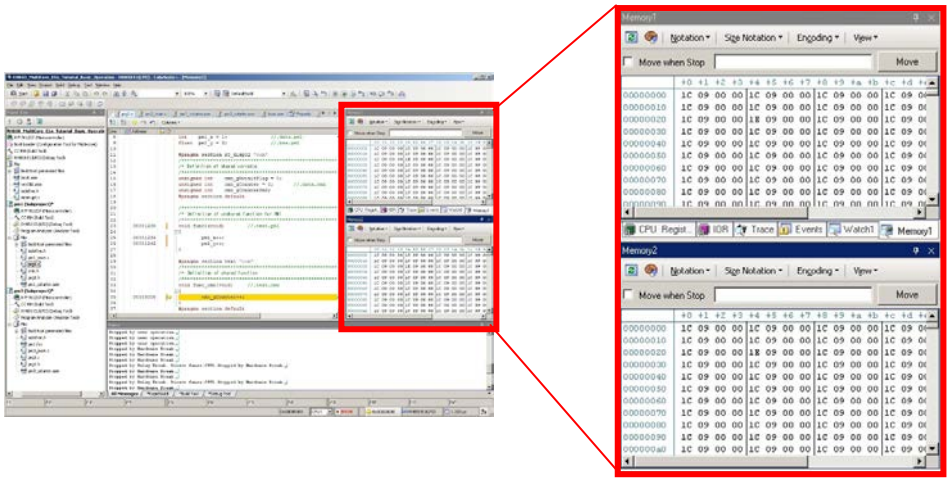
The Memory 1 panel is displayed.
(Similarly, display the Memory 2 panel.)



Make the Memory 2 panel enter the floating state and dock it to the Output panel.



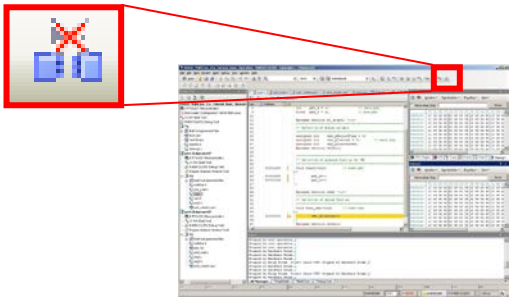
It becomes possible to view the Memory 1 and Memory 2 panels at the same time.



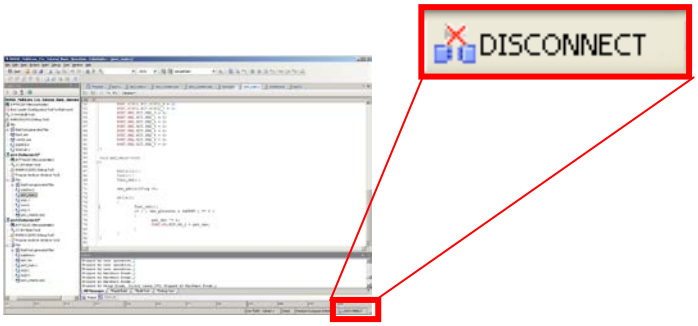
Disconnecting a Debug Tool

When finishing debugging, disconnect the debug tool.

Click the "Disconnect from Debug Tool" button.



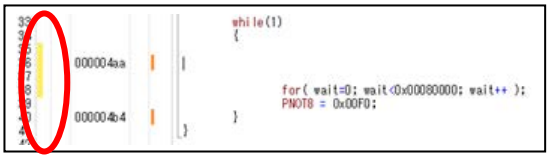
The status bar in the main window shows [DISCONNECT] as shown below and debugging finishes.



Tip

Downloading a program

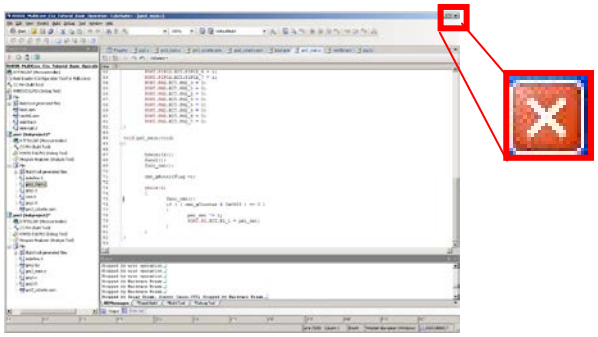
If a program is changed after being downloaded to the target, you have to perform the build process again and download the program. If a program is changed after downloading, the right side of the line number turns yellow (or green) and breakpoints cannot be set.



Termination Procedure

This section describes the termination procedure.

Click the Close button in the main window.

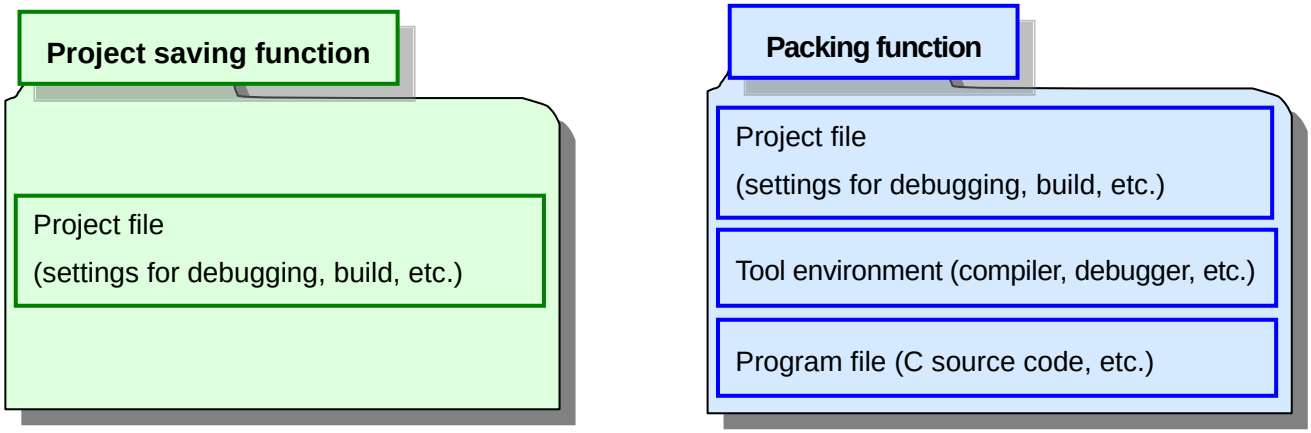


This closes the main window and exits CS+. If the environment is not saved, a window that prompts you to select whether or not to save it will appear, so follow the instructions.

Tip

Saving the development environment (project saving function and packing function)

CS+ provides two functions (project saving function and packing function) for saving the development environment. Each of these functions is used to save the contents shown in the figure below. Either saving function can be chosen depending on the development phase.



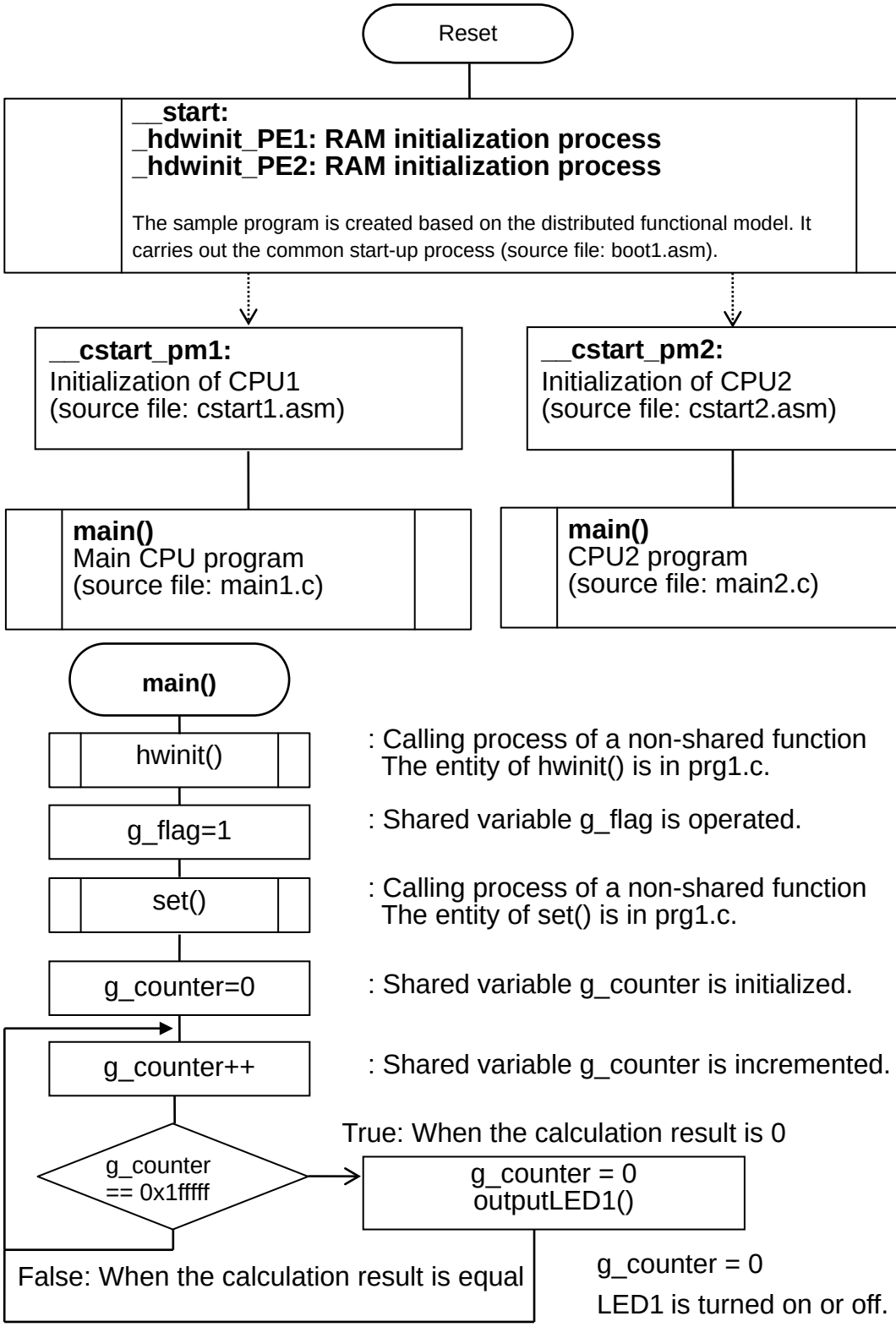
About Flash Programming

When using the E1 emulator to write a .hex file to the microcontroller, use Renesas Flash Programmer (RFP).

- Start Renesas Flash Programmer (RFP) by selecting [Start] -> [All Programs] -> [Renesas Electronics Utilities] -> [Programming tools].
- Refer to the user's manual for the usage method.

Description of Sample Programs

The flow diagrams of sample programs are shown below.



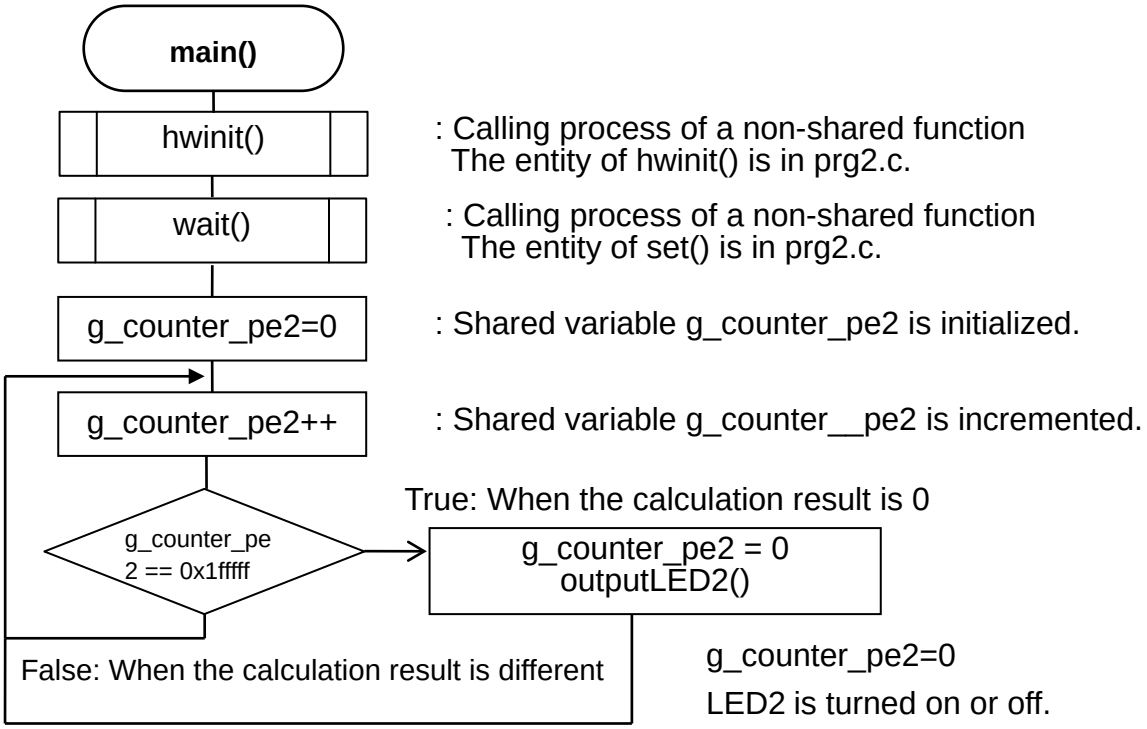
hwinit() : Hardware is initialized.

wait() : Wait until the set() function is called.

set() : Wake up the CPU kept waiting by the wait() function.

outputLED1() : LED1 is turned on or off.

outputLED2() : LED2 is turned on or off.



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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 5.0