Introduction

This application note describes how to replace the source codes created by the CA78K0R C compiler for the integrated development environment CS+ with the source codes supported by the CC-RL C compiler for the integrated development environment CS+.

The applicable C compiler versions are as follows.

- CA78K0R V1.20 and later
- CC-RL V1.01.00

Target Device

RL78 Family

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.
Contents

1. Methods for Migrating Projects from CA78K0R to CC-RL ...................................................... 3

2. Manual Migration Method .................................................................................................................... 4
   2.1 Generating Source Codes Automatically .................................................................................. 4
   2.2 Adding Source Codes Other Than Automatically Generated Source Codes ....................... 7
      2.2.1 Adding Source Codes ...................................................................................................... 7
      2.2.2 Adding User Initialization Function ............................................................................... 8
   2.3 Correcting Added Parts ............................................................................................................... 11
      2.3.1 Accessing Special Function Registers (SFR) ................................................................. 11
      2.3.2 Enabling Interrupt Functions ......................................................................................... 13
      2.3.3 Enabling CPU Control Instructions .............................................................................. 13
      2.3.4 Replacing Absolute Address Specification (__directmap) .............................................. 14
      2.3.5 Replacing Variable saddr Area Allocation (sreg, __sreg) ............................................... 15
      2.3.6 near/far Attributes ......................................................................................................... 16

3. Migration Method Using Porting Support Function ................................................................. 17
   3.1 Creating Project by Using Existing Project ............................................................................. 17
   3.2 Adding Include File .................................................................................................................. 18
   3.3 Changing Startup File ............................................................................................................. 19
   3.4 Deleting Special Function Registers (SFR) Access Description .......................................... 20

4. ROM Allocation Method ................................................................................................................. 21

5. Sample Code .................................................................................................................................. 23

6. Reference Documents .................................................................................................................... 23
1. Methods for Migrating Projects from CA78K0R to CC-RL

Two methods are available to replace the source codes created by the CA78K0R C compiler for the integrated development environment CS+ with the source codes supported by the CC-RL C compiler for the integrated development environment CS+.

In the first method, create a new project with the integrated development environment CS+, then manually port the source codes created by the CA78K0R C compiler for the integrated development environment CS+, and finally create a project supported by the CC-RL C compiler for the integrated development environment CS+. In the second method, use the porting support function in the integrated development environment CS+ to change the source codes created by the CA78K0R C compiler for the integrated development environment CS+ to a new project supported by the CC-RL C compiler for the integrated development environment CS+.

Section 2 explains the manual migration method. Section 3 explains the migration method using the porting support function.
2. Manual Migration Method

2.1 Generating Source Codes Automatically

Source codes are automatically generated using the code generator tool in the CC-RL C compiler for the integrated development environment CS+. Set the code generator tool by referring to the existing source codes that were created by the CA78K0R C compiler for the integrated development environment CS+.

(1) Under [Project Tree], click [Clock Generator] in [Code Generator (Design Tool)]. (Figure 2.1 A).

(2) Perform “Pin assignment” and click the [Fix settings] button. (Figure 2.1 B)

Note: To set other functions, it is necessary to set the pin assignment. When the pin assignment setting is decided once, it is not possible to change it later.
(3) Refer to the existing source codes that were created by the CA78K0R C compiler for the integrated development environment CS+ and set each function.

Figure 2.2 Code Generator Setting Window (2)
(4) On completion of all the function settings, click the [Generate Code] button at the top of the window (Figure 2.3 C) to generate codes (automatic source code generation).

![Figure 2.3 Code Generator Setting Window (3)](image-url)
2.2 Adding Source Codes Other Than Automatically Generated Source Codes

2.2.1 Adding Source Codes

Add the source codes required for the source codes generated automatically with the source generator tool.

First, check the difference between the source code created with CA78K0R and the automatically generated source code. To check the difference, use the software that can compare multiple text files.

Next, add the difference to the automatically generated source code. Include the source code between “/* Start user code for include. Do not edit comment generated here */” and “/* End user code. Do not edit comment generated here */”.

If the source codes are added to the position other than the above, automatically generating source code again by pressing the [Generate Code] button in the automatic generator tool will clear the source codes added to the position other than the above. To prevent the codes from being cleared, change the setting as shown below in the code generator tool.

As indicated in the red box in Figure 2.4, change [Generate file] in [Generate File Mode] from [Merge file] to [Do nothing if file exists].

![Figure 2.4 Code Generator Property](image-url)
2.2.2 Adding User Initialization Function

The function for user initialization process (R_×××_Create_UserInit (××× is the function name)) created by the code generator tool in CA78K0R is not automatically generated by the code generator tool in CC-RL. The process of calling the user initialization function within the function initialization function is not also automatically generated.

r_cg_dmac.c for CA78K0R

```c
void R_DMA0_Create(void)
{

```

```c
r_cg_dmac.c for CC-RL

```c
void R_DMA0_Create( void )
{

```

```c
Figure 2.5 Difference between CA78K0R and CC-RL Code Generator Tools

The process of calling the user initialization function “R_DMA0_Create_UserInit()” is not automatically generated.
An example of using the function for DMA0 user initialization (R_DMA0_Create_UserInit) is shown to describe how to add the user initialization function.

1. Copy “R_DMA0_Create_UserInit” included in r_cg_dmac_user.c that is the source code for CA78K0R to r_cg_dmac_user.c that is the source code for CC-RL.

```c
#include "r_cg_userdefine.h"

void R_DMA0_Create_UserInit(void)
{
    /*Start user code. Do not edit. Comment generated here.*/
    DENO = 1U; /* Enable DMA0 operation*/
    DRAO = (uint16_t)&AdResult; /* Set destination RAM address*/
    DBCO = ADC_USED_CH_NUM*ADC_EXEC_TIMES;
    DENO = 0U; /* Disable DMA0 operation*/

    /*End of function R_DMA0_Create_UserInit*/
}
```

Figure 2.6 Adding User Initialization Function
(2) Globally declare the added function.

```c
void R_DMA0 Create(void);
void R_DMA0 Start(void);
void R_DMA0 Stop(void);
/*Start-user-code-for-function-Do-not-edit-comment-generated-here*/
void R_DMA0 Create UserInit(void);
/*End-user-code-Do-not-edit-comment-generated-here*/
#endif
```

Figure 2.7 Description Example in CC-RL Header File “r_cg_dma.h”

(3) Add the process to be added to call the user initialization function in the R_MAIN_UserInit() function in the `r_main.c` file.

```c
/*Function-Name: R_MAIN_UserInit
 *Description: This function adds user code before implementing main function,
 *Arguments: None
 *Return Value: None
 *Start-user-code Do-not-edit-comment-generated-here*/
void R_MAIN_UserInit(void)
{
  /*Start-user-code-for-function-Do-not-edit-comment-generated-here*/
   R_DMA0 Create UserInit();....../*Initialized-destination-address-for-DMA*/
...E1();
.../*End-user-code-Do-not-edit-comment-generated-here*/
}
/*Start-user-code-for-adding-Do-not-edit-comment-generated-here*/
```

Figure 2.8 Adding Process of Calling User Initialization Function for CC-RL

This completes the process of adding the user initialization function.
2.3 Correcting Added Parts

A warning message or an error may occur if the source code added in section 2 is left unchanged. In this case, the description should be corrected according to the CC-RL specifications.

The main differences in the description specifications between CA78K0R and CC-RL are described below.

2.3.1 Accessing Special Function Registers (SFR)

(1) Change the method of accessing the special function registers (SFR).

CA78K0R: #pragma sfr
CC-RL: #include "iodefine.h"

Because CC-RL does not support #pragma sfr, include the define header file for the sfr access “iodefine.h” that is automatically generated by the code generator tool.
(2) Correct the port register description.

When CA78K0R is used, “bit number” is added at the end of a register name. When CC-RL is used, “bit_no bit number” is added at the end of a register name.

r_main.c for CA78K0R

```c
88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116
.....+/+AD-conversion-stop+/*
.....-R_ADC_Stop();
.....+/+Check-result-of-AD-conversion-data+/*
.....if (result == 0x00)
.....{...
.....+/+LED1-turn-on+/*
.....P8.bit_no2 = 0;
.....while (1U)
.....{ ...
.....+/+Do-Nothing+/*
.....}
.....+/+Next-AD-test+/*
.....}
.....+/+testVoltageIndex;
.....}...
.....+/+AD-test-NG+/*
.....{...
.....+/+LED-blinks+/*
.....-R_Main_Blink_Led();
.....}
.....+/+End-user-code.-Do-not-edit-comment-generated-here+/*
```  

r_main.c for CC-RL

```c
83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
.....+/+Gets-the-check-result-of-AD-conversion-data+/*
.....result = R_Main_Check_AD_Data(testVoltageIndex);
.....+/+AD-conversion-stop+/*
.....-R_ADC_Stop();
.....+/+Check-result-of-AD-conversion-data+/*
.....if (result == 0x00)
.....{...
.....+/+LED1-turn-on+/*
.....P8.bit_no2 = 0U;
.....while (1U)
.....{ ...
.....+/+Do-Nothing+/*
.....}
.....}...
.....+/+Next-AD-test+/*
```  

Figure 2.9 Port Register Descriptions
2.3.2 Enabling Interrupt Functions

Replace the #pragma directive with a function.

1. In case of di
   #pragma di \rightarrow \_DI();
   (When r_cg_macrodriver.h is used, DI(); can also be used.)

2. In case of ei
   #pragma ei \rightarrow \_EI();
   (When r_cg_macrodriver.h is used, EI(); can also be used.)

2.3.3 Enabling CPU Control Instructions

Replace the #pragma directive with a function.

1. In case of halt
   #pragma halt \rightarrow \_halt();
   (When r_cg_macrodriver.h is used, HALT(); can also be used.)

2. In case of stop
   #pragma stop \rightarrow \_stop();
   (When r_cg_macrodriver.h is used, STOP(); can also be used.)

3. In case of brk
   #pragma brk \rightarrow \_brk();
   (When r_cg_macrodriver.h is used, BRK(); can also be used.)

4. In case of nop
   #pragma nop \rightarrow \_nop();
   (When r_cg_macrodriver.h is used, NOP(); can also be used.)
2.3.4 Replacing Absolute Address Specification (__directmap)

To specify absolute addresses with CA78K0R, “__directmap” is used. To specify absolute addresses with CC-RL, “#pragma address” is used.

Change (1) “__directmap type specification variable name = start address;” to (2) “#pragma address variable name = start address” and (3) “type specification variable name;”.

Example)

1) __directmap uint8_t p130_high = {0xFE900};
2) #pragma address p130_high = 0xFE900U
3) uint8_t __near p130_high;

r_main.c for CA78K0R

```c
//************ Global variables and functions ************

/* Start user code for global - Do not edit comment generated here */

__directmap uint8_t __near p130_high = {0xFE900};
__directmap uint8_t __near p130_low = {0xFE901};
__directmap uint8_t __near adc_snooze = {0xFE902};

uint8_t __near buffer_count;...................../* buffer-counter */
uint16_t __near result_buffer[MAX BUFFER];...../* AD-converter-result-buffer */

/* End user code - Do not edit comment generated here */
```

r_main.c for CC-RL

```c
/* Start user code for pragma - Do not edit comment generated here */

#pragma address p130_high = {0xFE900U}
#pragma address p130_low = {0xFE901U}
#pragma address adc_snooze = {0xFE902U}
#pragma address get_addr = {0xFEA00U}

/* End user code - Do not edit comment generated here */

/* Start user code for global - Do not edit comment generated here */

uint8_t __near p130_high;
uint8_t __near p130_low;
uint8_t __near adc_snooze;
uint16_t __near get_addr[MAX BUFFER];

uint8_t __near buffer_count;...................../* buffer-counter */
uint16_t __near result_buffer[MAX BUFFER];...../* AD-converter-result-buffer */

/* End user code - Do not edit comment generated here */
```

Figure 2.10 Description of __directmap
2.3.5 Replacing Variable saddr Area Allocation (sreg, __sreg)

To allocate a variable to the saddr area with CA78K0R, “sreg (or __sreg)” is used. To allocate a variable to the saddr area with CC-RL, “__saddr” is used.

Change (1) “sreg type specification variable name; (or __sreg type specification variable name;)” to (2) “__saddr type specification variable name;”.

Example)

(1) sreg uint32_t g_PulseWidth[8];

(2) __saddr uint32_t g_pulse_width[8];

r_main.c for CA78K0R

```c
52 53 54 55 56 57 58 59
(1) sreg uint32_t g_PulseWidth[8]; /* Store pulse width */
uint8_t g_Times; /* Measurement times counter */
/* End user code...Do not edit comment generated here */
```

r_main.c for CC-RL

```c
43 44 45 46 47 48 49 50 51 52 53 54 55
(2) __saddr uint32_t g_pulse_width[8]; /* Store pulse width */
uint8_t g_Times; /* Measurement times counter */
/* End user code...Do not edit comment generated here */
void R_DMA_Init(void);
```

Figure 2.11 Example of Changing sreg Description
2.3.6 near/far Attributes

As for the memory model, the small, medium, and large models are available in CA78K0R, but only the small and medium models are available in CC-RL.

When the small model or the medium model in CA78K0R is used, the same memory model of the small model or the medium model in CC-RL is used.

When the large model in CA78K0R is used, the medium model in CC-RL is used.

In addition, if neither the near area nor the far area is specified for a function or a variable, the function or variable is allocated in the near area. Therefore, the __far type qualifier is used to allocate a function or variable in the far area.

In case of the source code that references the area extending over 64 K such as the source code that references the CRC calculation result data, the pointer attribute should be changed to the far attribute.

Example)

1) Change “uint16_t *oc_calc_hs_crc;” to “__far uint16_t *oc_calc_hs_crc;.”

2) Change “oc_calc_hs_crc = (uint16_t *)HIGHSPEED_CALC_ADDR;” to “oc_calc_hs_crc = (__far uint16_t *)HIGHSPEED_CALC_ADDR;.”

r_main.c for CA78K0R

```c
uint16_t oc_calc_hs_crc; /* Program result (General-Purpose) */
uint16_t count;

/* Hi-speed CRC */
/* Get Hi-speed CRC calculated result that OC output. */
oc_calc_hs_crc = (uint16_t *)HIGHSPEED_CALC_ADDR;
result_hs_crc = R_HighSpeedCRCProc(); /* Process of Hi-speed CRC */
/* The results are compared and it outputs it to LED. */
if (result_hs_crc == oc_calc_hs_crc) /* Hi-speed CRC */
    P6.2 = 0;
```

r_main.c for CC-RL

```c
... uint16_t result_hs_crc; /* Program result (General-Purpose) */
... __far uint16_t *oc_calc_hs_crc; /* Pointer of OC result (Hi-Speed) */
...

/* Hi-speed CRC */
/* Get Hi-speed CRC calculated result that OC output. */
oc_calc_hs_crc = ( __far uint16_t *)HIGHSPEED_CALC_ADDR;
result_hs_crc = R_HighSpeedCRCProc(); /* Process of Hi-speed CRC */
/* The results are compared and it outputs it to LED. */
if (result_hs_crc == oc_calc_hs_crc) /* Hi-speed CRC */
    P6.2 = 0;
...}
```

Figure 2.12 Example of Changing Pointer Attribute
3. Migration Method Using Porting Support Function

This section explains how to migrate the existing project of the CA78K0R C compiler for the integrated development environment CS+ to the source codes of a new project of the CC-RL C compiler for the integrated development environment CS+.

3.1 Creating Project by Using Existing Project

(1) Click the [Start] button at the top of the window to display the start menu.

(2) Click the [GO] button in the [Create New Project] item in the start menu.

(3) Select the microcontroller to be used.

(4) Select [Application (CC-RL)] for [Kind of project].

(5) Select the [Pass the file composition of an existing project to the new project] checkbox and enter the project file name to be passed in [Project to be passed:]. Then, select the [Copy composition files in the diverted project folder to a new project folder] checkbox.

(6) Click the [Create] button to create a project.

![Figure 3.1 Start Menu Window of CS+](image-url)
3.2 Adding Include File

Click [CC-RL (Build Tool)] in [Project Tree] and open the [Include files at head of compiling units] item in the [Compile Options] tab. Then add “iodefine.h”.

![Figure 3.2 Compile Options](image-url)
3.3 Changing Startup File

When the main function and the hdwinit function are registered in the existing project, use the following procedure to exclude the files that are automatically generated during project creation (main.c and hdwinit.asm) from the target of build.

(1) Right-click [main.c] in [Project Tree] to display the menu.
(2) Select [Property] from the menu.
(3) Change the [Set as build-target] item from [Yes] to [No] in the property of the file. (The same procedure is used to change the setting of hdwinit.asm.)
3.4 Deleting Special Function Registers (SFR) Access Description

Delete the description of ‘#pragma sfr’ in “r_cg_macrodriver.h”.

Because the porting support function does not support the replacement of the absolute address specification (__directmap), the absolute address specification should be replaced manually according to 2.3.4.
4. ROM Allocation Method

To allocate sections with CA78K0R, a link directive file is used. To allocate sections with CC-RL, the link option section is used for setting. In addition, the -start option can also be used to allocate sections.

Click [CC-RL (Build Tool)] in [Project Tree].

![Figure 4.1 Link Option Setting Example in CC-RL (CS+) (1)](image)
(1) Click the [Link Option] tab.

(2) Open the [Section] item.

(3) Change [Layout sections automatically] from [Yes] to [No].

(4) Click the […] button in the [Section start address] item.

(5) Specify the section settings in the section settings window that appears.

(6) Click the [OK] button to finish the section settings.

Important
- The allocation of the ROM area/RAM area in each section cannot be changed.
- The allocation areas for the SFR area, the interrupt vector area (section .vect), and the CALLT function table area (section .callt0) have already been determined and thus are not specified.
- The sections for the saddr area (sections .sdataR and .sbss) should be allocated within the range of the .saddr area.

Figure 4.2 Link Option Setting Example in CC-RL (CS+)

These are set by default
5. **Sample Code**

The sample code is available on the Renesas Electronics website.

6. **Reference Documents**

RL78Family User’s Manual: Software (R01US0015E)

RL78 Compiler CC-RL User’s Manual (R20UT3123E)

Integrated Development Environment for the RL78 Family - Migrating from the CA78K0R to the CC-RL

  (Project Manipulation) (R20UT3415E)
  (Coding) (R20UT3416E)
  (Linkage Editor Options) (R20UT3417E)
  (Compiler Options and Assembler Options) (R20UT3418E)

CS+ Code Generator Tool Integrated Development Environment User’s Manual: RL78 API Reference

  (R20UT3102E)

CS+ V3.01.00 Integrated Development Environment User’s Manual: Message (R20UT3286E)

CS+ V3.01.00 Integrated Development Environment User’s Manual: Project Operation (R20UT3287E)

(The latest information can be downloaded from the Renesas Electronics website.)

**Website and Support**

Renesas Electronics website

http://www.renesas.com/

Inquiries

http://www.renesas.com/contact
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Feb. 26, 2016</td>
<td>First edition issued</td>
</tr>
</tbody>
</table>

すべての商標および登録商標は、それぞれの所有者に帰属します。
General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU 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. 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on
   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.
   - The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the 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.