

[Notes]

R20TS0772EJ0100

Rev.1.00

Dec. 01, 2021

RX Family

Flash Module Using Firmware Integration Technology,

RX Driver Package

Outline

When using the product in the title, note the following points.

1. Note on the callback after "R_FLASH_Erase" (flash type 3 and 4, non-blocking mode)
2. Note on the response time after "R_FLASH_Erase", "R_FLASH_BlankCheck", and "R_FLASH_Write" (flash type 1, non-blocking mode)
3. Note on "R_FLASH_Write" (flash type 1, non-blocking mode)

1. Note on the Callback After "R_FLASH_Erase" (Flash Type 3 and 4, Non-Blocking Mode)

1.1 Applicable Products

- (1) RX Family Flash Module Using Firmware Integration Technology

(Hereafter referred to as the Flash Module)

The applicable revisions and documents are as follows.

Table 1.1 Flash Module applicable products

Revision	Document number
Rev.2.10	R01AN2184EU0210
Rev.3.00	R01AN2184EU0300
Rev.3.10	R01AN2184EU0310
Rev.3.20	R01AN2184EU0320
Rev.3.30	R01AN2184EU0330
Rev.3.40	R01AN2184EU0340
Rev.3.41	R01AN2184EU0341
Rev.3.42	R01AN2184EU0342
Rev.3.50	R01AN2184EU0350
Rev.4.00	R01AN2184EJ0400
Rev.4.10	R01AN2184EJ0410
Rev.4.20	R01AN2184EJ0420
Rev.4.30	R01AN2184EJ0430
Rev.4.40	R01AN2184EJ0440
Rev.4.50	R01AN2184EJ0450
Rev.4.60	R01AN2184EJ0460
Rev.4.70	R01AN2184EJ0470
Rev.4.80	R01AN2184EJ0480

(2) RX Driver Package

The Flash Module in (1) is also included in the RX Driver Packages.

The product names, revisions, documents of the RX Driver Packages, and the revisions of the Flash Module are as follows.

Table 1.2 Products that include the Flash Module

RX Driver Package product name	RX Driver Package revision	Document number	Flash Module revision
RX Family RX Driver Package Ver.1.12	Rev.1.12	R01AN3651EJ0112	Rev.2.10
RX Family RX Driver Package Ver.1.13	Rev.1.13	R01AN3859EJ0113	Rev.3.20
RX Family RX Driver Package Ver.1.14	Rev.1.14	R01AN4191EJ0114	Rev.3.30
RX Family RX Driver Package Ver.1.15	Rev.1.15	R01AN4372EJ0115	Rev.3.30
RX Family RX Driver Package Ver.1.16	Rev.1.16	R01AN4471EJ0116	Rev.3.40
RX Family RX Driver Package Ver.1.17	Rev.1.17	R01AN4572EJ0117	Rev.3.41
RX Family RX Driver Package Ver.1.18	Rev.1.18	R01AN4659EJ0118	Rev.3.42
RX Family RX Driver Package Ver.1.19	Rev.1.19	R01AN4677EJ0119	Rev.3.50
RX Family RX Driver Package Ver.1.20	Rev.1.20	R01AN4794EJ0120	Rev.4.00
RX Family RX Driver Package Ver.1.21	Rev.1.21	R01AN4843EJ0121	Rev.4.10
RX Family RX Driver Package Ver.1.22	Rev.1.22	R01AN4873EJ0122	Rev.4.20
RX Family RX Driver Package Ver.1.23	Rev.1.23	R01AN4976EJ0123	Rev.4.40
RX Family RX Driver Package Ver.1.24	Rev.1.24	R01AN5267EJ0124	Rev.4.40
RX Family RX Driver Package Ver.1.25	Rev.1.25	R01AN5371EJ0125	Rev.4.50
RX Family RX Driver Package Ver.1.26	Rev.1.26	R01AN5401EJ0126	Rev.4.50
RX Family RX Driver Package Ver.1.27	Rev.1.27	R01AN5600EJ0127	Rev.4.60
RX Family RX Driver Package Ver.1.29	Rev.1.29	R01AN5826EJ0129	Rev.4.60
RX Family RX Driver Package Ver.1.30	Rev.1.30	R01AN5882EJ0130	Rev.4.60
RX Family RX Driver Package Ver.1.31	Rev.1.31	R01AN5975EJ0131	Rev.4.70
RX Family RX Driver Package Ver.1.32	Rev.1.32	R01AN6013EJ0132	Rev.4.80

(3) FIT Modules used in combination with Flash Module and their application notes

The problem may occur when certain FIT Modules are used with the Flash Module in (1).

Examples

- RX Family Flash Memory Data Management Module Using Firmware Integration Technology (R20AN0507EJ)
<https://www.renesas.com/jp/en/document/apn/rx-family-flash-memory-data-management-module-using-firmware-integration-technology>
- RX Family Firmware Update Module Using Firmware Integration Technology (R01AN5824EJ)
<https://www.renesas.com/jp/en/document/apn/rx-family-firmware-update-module-using-firmware-integration-technology-application-notes>
- RX Family TSIP (Trusted Secure IP) Module Firmware Integration Technology (R20AN0548EJ)
<https://www.renesas.com/jp/en/document/apn/rx-family-tsip-trusted-secure-ip-module-firmware-integration-technology-binary-version>

1.2 Applicable Devices

RX64M, RX651, RX65N, RX66N, RX66T, and RX671 groups

RX71M, RX72M, RX72N, and RX72T groups

1.3 Details

If a flash memory error occurs after "R_FLASH_Erase" is executed, the callback function sends the error message to the user application. The size of the message may become the number of the blocks specified as the parameter for "R_FLASH_Erase".

1.4 Conditions

The problem occurs when the following conditions are met.

Condition 1: The flash module is set to the non-blocking mode.

Condition 2: Refer to the code snippets in (1). Interval A takes longer than the erasure time.

Condition 3: A flash memory error has occurred.

The following shows an example of Condition 2, in which an interrupt has occurred during the interval.

(1) The main routine process and the problem

The code snippets show "r_flash_fcu.c", in which the final erase command is written (FLASH_FACI_CMD_FINAL) and the current operation variable is judged (g_current_parameters.current_operation). If the interval between the two events takes longer than the erasure time, and if a flash memory error occurs meanwhile, an FIFERR interrupt is requested before the judgement.

The current operation variable is changed during the FIFERR interrupt. The judgement of the variable does not match after the interrupt. Therefore, "break" does not occur, but the erasure process continues.

```
flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
    (Omitted)
    Write the final erase command
    *g_pfcu_cmd_area = (uint8_t) FLASH_FACI_CMD_BLOCK_ERASE;
    *g_pfcu_cmd_area = (uint8_t) FLASH_FACI_CMD_FINAL;
    Interval A
    /* Return if in BGO mode. Processing will finish in FRDYI interrupt */
    if ((g_current_parameters.current_operation == FLASH_CUR_CF_BGO_ERASE)
        || (g_current_parameters.current_operation ==
            FLASH_CUR_DF_BGO_ERASE))
    {
        break;
        Judge the current operation variable
        It does not break
    }
    (Omitted)
}
```

The codes are expanded by the assembler as follows.

The final erase command is written in (2).

If an interrupt request occurs during (2) when the one-cycle instruction is executed, and if the processing of the interrupt takes longer than the erasure time, and if a flash memory error occurs meanwhile, an FIFERR interrupt is requested before the current operation variable is judged.

```
(*g_pfcu_cmd_area = (uint8_t) FLASH_FACI_CMD_FINAL;)
(1) mov.l    [r8], r14
(2) mov.b    #208, [r14]
(if ((g_current_parameters.current_operation == FLASH_CUR_CF_BGO_ERASE))
(3) mov.l    16[r7], r2
```

1.5 Workaround

Refer to the following "r_flash_fcu.c" and change the functions written in red.

Before modification

```
flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
    (Omitted)
    *g_pfcu_cmd_area = (uint8_t) FLASH_FACI_CMD_BLOCK_ERASE;
    *g_pfcu_cmd_area = (uint8_t) FLASH_FACI_CMD_FINAL;

    /* Return if in BGO mode. Processing will finish in FRDYI interrupt
    */
    if ((g_current_parameters.current_operation ==
FLASH_CUR_CF_BGO_ERASE)
        || (g_current_parameters.current_operation ==
FLASH_CUR_DF_BGO_ERASE))
    {
        break;
    }
    (Omitted)
}
```

After modification

```
flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
    (Omitted)
    *g_pfcu_cmd_area = (uint8_t) FLASH_FACI_CMD_BLOCK_ERASE;
    *g_pfcu_cmd_area = (uint8_t) FLASH_FACI_CMD_FINAL;

    /* Return if in BGO mode. Processing will finish in FRDYI interrupt
    */
    if ((g_current_parameters.bgo_enabled_cf == true)
        || (g_current_parameters.bgo_enabled_df == true))
    {
        break;
    }
    (Omitted)
}
```

1.6 Schedule for Fixing the Problem

The problem will be fixed in the next version.

2. Note on the Response Time After "R_FLASH_Erase", "R_FLASH_BlankCheck", and "R_FLASH_Write" (Flash Type 1, Non-Blocking Mode)

2.1 Applicable Products

(1) RX Family Flash Module Using Firmware Integration Technology

The applicable revision and document are as follows.

Table 2.1 Flash Module applicable product

Revision	Document number
Rev.4.80	R01AN2184EJ0480

(2) RX Driver Package

The Flash Module in (1) is also included in the RX Driver Package.

The product name, revision, document of the RX Driver Package, and the revision of the flash module are as follows.

Table 2.2 Product that includes the Flash Module

RX Driver Package product name	RX Driver Package revision	Document number	Flash Module revision
RX Family RX Driver Package Ver.1.32	Rev.1.32	R01AN6013EJ0132	Rev.4.80

(3) FIT Modules used in combination with Flash Module and their application notes

The problem may occur when certain FIT Modules are used with the Flash Module in (1).

Examples

- RX Family Flash Memory Data Management Module Using Firmware Integration Technology (R20AN0507EJ)
<https://www.renesas.com/jp/en/document/apn/rx-family-flash-memory-data-management-module-using-firmware-integration-technology>
- RX Family Firmware Update Module Using Firmware Integration Technology (R01AN5824EJ)
<https://www.renesas.com/jp/en/document/apn/rx-family-firmware-update-module-using-firmware-integration-technology-application-notes>
- RX Family TSIP (Trusted Secure IP) Module Firmware Integration Technology (R20AN0548EJ)
<https://www.renesas.com/jp/en/document/apn/rx-family-tsip-trusted-secure-ip-module-firmware-integration-technology-binary-version>

2.2 Applicable Devices

RX110, RX111, RX113, RX130, RX13T, and RX140 groups

RX230, RX231, RX23E-A, RX23T, RX23W, RX24T, and RX24U groups

2.3 Details

It may take long to respond to the user application after "R_FLASH_Erase", "R_FLASH_BlankCheck", and "R_FLASH_Write" are executed.

2.4 Conditions

The problem occurs when Condition 1 and any of the three conditions are met.

Condition 1: The flash module is set to the non-blocking mode.

Condition 2: Interval A or B* takes longer than the erasure time.

Condition 3: Interval A or B* takes longer than the blank check.

Condition 4: Interval A or B* takes longer than the programming time.

*Refer to the code snippets in (1).

The following shows an example of Condition 2 to 4, in which an interrupt has occurred during the interval.

(1) The main routine process and the problem

Condition 2

The code snippets show "r_flash_nofcu.c", in which "flash_df_erase" or "flash_cf_erase" is executed and the current operation variable is judged (g_current_parameters.current_operation). If the interval between the two events takes longer than the erasure time, an FRDYI interrupt is requested before the judgement.

During the FRDYI interrupt, the current operation variable is changed and FRDY is set to 0. The judgement of the variable does not match after the interrupt. Therefore, "return" does not occur, but "flash_wait_frdy" is executed. However, since FRDY is 0, a timeout occurs in "flash_wait_frdy", causing the delay in response.

```

flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
  (Omitted)
  if (FLASH.FENTRYR.WORD == 0x0080)
  {
    #ifndef FLASH_NO_DATA_FLASH
      flash_df_erase(block_address, num_blocks);
    #endif
  }
  (Omitted)
  else if (FLASH.FENTRYR.WORD == 0x0001)
  {
    flash_cf_erase(block_address, num_blocks);
  }
  (Omitted)
  /* Return if in BGO mode. Processing will finish in FRDYI interrupt */
  if ((g_current_parameters.current_operation == FLASH_CUR_CF_BGO_ERASE)
      || (g_current_parameters.current_operation == FLASH_CUR_DF_BGO_ERASE))
  {
    return err;
  }
  (Omitted)
  err = flash_wait_frdy();
  if (err != FLASH_SUCCESS)
  {
    return err;
  }
  (Omitted)
}

```

The diagram illustrates the execution flow of the `flash_erase` function. It shows two main paths: one for `flash_df_erase` and another for `flash_cf_erase`. The code is divided into sections by horizontal dashed lines. Two intervals, **Interval A** and **Interval B**, are marked with green boxes and red dashed arrows. **Interval A** spans from the start of the function to the first judgement point. **Interval B** spans from the start of the second judgement point to the end of the function. The judgement points are highlighted with red boxes and labeled "Judge the current operation variable". The first judgement point is followed by a red box containing `return err;`, which is labeled "It does not return". The second judgement point is followed by a red box containing `err = flash_wait_frdy();`, which is labeled "Timeout". The final `if (err != FLASH_SUCCESS)` block is also highlighted with a red box.

Condition 3

The code snippets show "r_flash_nofcu.c", in which "flash_df_blankcheck" or "flash_cf_blankcheck" is executed and the current operation variable is judged. If the interval between the two events takes longer than the blank check, an FRDYI interrupt is requested before the judgement.

During the FRDYI interrupt, the current operation variable is changed and FRDY is set to 0. The judgement of the variable does not match after the interrupt. Therefore, "return" does not occur, but "flash_wait_frdy" is executed. However, since FRDY is 0, a timeout occurs in "flash_wait_frdy", causing the delay in response.

```
flash_err_t flash_blankcheck(const uint32_t start_address, const uint32_t
num_bytes, flash_res_t *result)
{
(Omitted)
    if (FLASH.FENTRYR.WORD == 0x0080)
    {
#ifdef FLASH_NO_DATA_FLASH
        flash_df_blankcheck(start_address, (start_address + num_bytes) - 1);
#endif
    }
    if (FLASH_CFG_CODE_FLASH_ENABLE == 1)
        else if (FLASH.FENTRYR.WORD == 0x0001)
        {
            flash_cf_blankcheck(start_address, (start_address + num_bytes) - 1);
        }
    #endif
    else
    {
        /* should never get here */
        return FLASH_ERR_FAILURE;
    }

    /* Return if in BGO mode. Processing will finish in FRDYI interrupt */
    if ((g_current_parameters.current_operation ==
FLASH_CUR_CF_BGO_BLANKCHECK)
        || (g_current_parameters.current_operation ==
FLASH_CUR_DF_BGO_BLANKCHECK))
    {
        return err;
    }

    /* In blocking mode, wait for FRDY or timeout. Return if error. */
    err = flash_wait_frdy();
    if (FLASH_SUCCESS == err)
    {
        *result = FLASH_RES_BLANK;
    }
    else if (FLASH_ERR_FAILURE == err)
    {
        *result = FLASH_RES_NOT_BLANK;
        err = FLASH_SUCCESS;
    }
    else
    {
        /* timeout occurs */
    }
(Omitted)
```

Interval A (green box) points to the interval between the end of the first blank check and the start of the second blank check.

Interval B (green box) points to the interval between the end of the second blank check and the start of the third blank check.

Judge the current operation variable (yellow box) points to the condition check for the current operation variable.

It does not return (yellow box) points to the `return err;` statement.

Timeout (yellow box) points to the `err = flash_wait_frdy();` statement.

Condition 4

The code snippets show "r_flash_nofcu.c", in which "flash_df_write" or "flash_cf_write" is executed and the current operation variable is judged. If the interval between the two events takes longer than the programming time, an FRDYI interrupt is requested before the judgement.

During the FRDYI interrupt, the current operation variable is changed and FRDY is set to 0. The judgement of the variable does not match after the interrupt. Therefore, "break" does not occur, but "flash_wait_frdy" is executed. However, since FRDY is 0, a timeout occurs in "flash_wait_frdy", causing the delay in response.

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
(Omitted)
    /* Conversion to the P/E address from the read address */
    if (FLASH.FENTRYR.WORD == 0x0080)
    {
#ifdef FLASH_NO_DATA_FLASH
        flash_df_write(g_current_parameters.src_addr,
g_current_parameters.dest_addr);
#endif
    }
    if (FLASH_CFG_CODE_FLASH_ENABLE == 1)
        else if (FLASH.FENTRYR.WORD == 0x0001)
        {
            flash_cf_write(g_current_parameters.src_addr,
g_current_parameters.dest_addr);
        }
    #endif

    g_current_parameters.total_count--;

    if (g_current_parameters.current_operation == FLASH_CUR_STOP) // err
occurred; addr known good; protection err
    {
        err = FLASH_ERR_ACCESSW;
        break;
    }

    /* Return if in BGO mode. Processing will finish in FRDYI interrupt */
    if ((g_current_parameters.current_operation == FLASH_CUR_CF_BGO_WRITE)
|| (g_current_parameters.current_operation ==
FLASH_CUR_DF_BGO_WRITE))
    {
        break;
    }

    /* In blocking mode, wait for FRDY or timeout. Return if error. */
    err = flash_wait_frdy();
    if (err != FLASH_SUCCESS)
    {
        break;
    }
}
(Omitted)
```

2.5 Workaround

Refer to the following "r_flash_nofcu.c" and change the functions written in red.

Before modification

```
flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
(Omitted)
    /* Return if in BGO mode. Processing will finish in FRDYI interrupt */
    if ((g_current_parameters.current_operation == FLASH_CUR_CF_BGO_ERASE)
        || (g_current_parameters.current_operation == FLASH_CUR_DF_BGO_ERASE))
    {
        return err;
    }
(Omitted)
}
```

```
flash_err_t flash_blankcheck(const uint32_t start_address, const uint32_t
num_bytes, flash_res_t *result)
{
(Omitted)
    /* Return if in BGO mode. Processing will finish in FRDYI interrupt */
    if ((g_current_parameters.current_operation ==
FLASH_CUR_CF_BGO_BLANKCHECK)
        || (g_current_parameters.current_operation ==
FLASH_CUR_DF_BGO_BLANKCHECK))
    {
        return err;
    }
(Omitted)
}
```

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
(Omitted)
    /* Return if in BGO mode. Processing will finish in FRDYI interrupt */
    if ((g_current_parameters.current_operation == FLASH_CUR_CF_BGO_WRITE)
        || (g_current_parameters.current_operation ==
FLASH_CUR_DF_BGO_WRITE))
    {
        break;
    }
(Omitted)
}
```

After modification

```
flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
(Omitted)
    /* Return if in BGO mode. Processing will finish in FRDYI interrupt */
    if ((g_current_parameters.bgo_enabled_cf == true)
        || (g_current_parameters.bgo_enabled_df == true))
    {
        return err;
    }
(Omitted)
```

```
flash_err_t flash_blankcheck(const uint32_t start_address, const uint32_t
num_bytes, flash_res_t *result)
{
(Omitted)
    /* Return if in BGO mode. Processing will finish in FRDYI interrupt */
    if ((g_current_parameters.bgo_enabled_cf == true)
        || (g_current_parameters.bgo_enabled_df == true))
    {
        return err;
    }
(Omitted)
```

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
(Omitted)
    /* Return if in BGO mode. Processing will finish in FRDYI interrupt */
    if ((g_current_parameters.bgo_enabled_cf == true)
        || (g_current_parameters.bgo_enabled_df == true))
    {
        break;
    }
(Omitted)
```

2.6 Schedule for Fixing the Problem

The problem will be fixed in the next version.

3. Note on "R_FLASH_Write" (Flash Type 1, Non-Blocking Mode)

3.1 Applicable Products

(1) RX Family Flash Module Using Firmware Integration Technology

The applicable revision and document are as follows.

Table 3.1 Flash Module applicable product

Revision	Document number
Rev.4.80	R01AN2184EJ0480

(2) RX Driver Package

The Flash Module in (1) is also included in the RX Driver Package.

The product name, revision, document of the RX Driver Package, and the revision of the Flash Module are as follows.

Table 3.2 Product that includes the Flash Module

RX Driver Package product name	RX Driver Package revision	Document number	Flash Module revision
RX Family RX Driver Package Ver.1.32	Rev.1.32	R01AN6013EJ0132	Rev.4.80

(3) FIT Modules used in combination with Flash Module and their application notes

The problem may occur when certain FIT Modules are used with the Flash Module in (1).

Examples

- RX Family Flash Memory Data Management Module Using Firmware Integration Technology (R20AN0507EJ)
<https://www.renesas.com/jp/en/document/apn/rx-family-flash-memory-data-management-module-using-firmware-integration-technology>
- RX Family Firmware Update Module Using Firmware Integration Technology (R01AN5824EJ)
<https://www.renesas.com/jp/en/document/apn/rx-family-firmware-update-module-using-firmware-integration-technology-application-notes>
- RX Family TSIP (Trusted Secure IP) Module Firmware Integration Technology (R20AN0548EJ)
<https://www.renesas.com/jp/en/document/apn/rx-family-tsip-trusted-secure-ip-module-firmware-integration-technology-binary-version>

3.2 Applicable Devices

RX110, RX111, RX113, RX130, RX13T, and RX140 groups

RX230, RX231, RX23E-A, RX23T, RX23W, RX24T, and RX24U groups

3.3 Details

"R_FLASH_Write" may write excessive data (the minimum program size x 1).

3.4 Conditions

The problem occurs when the following conditions are met.

Condition 1: The flash module is set to the non-blocking mode.

Condition 2: Refer to the code snippets in (1). Interval A or B takes longer than the programming time.

The following shows an example of Condition 2, in which an interrupt has occurred during the interval.

(1) The main routine process and the problem

The code snippets show "r_flash_nofcu.c", in which "flash_df_write" or "flash_cf_write" is executed and the total count variable is decremented (g_current_parameters.total_count). If the interval between the two events takes longer than the programming time, an FRDYI interrupt is requested before the decrement.

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
(Omitted)
while(g_current_parameters.total_count > 0)
{
/* Conversion to the P/E address from the read address */
if (FLASH.FENTRYR.WORD == 0x0080)
{
#ifdef FLASH_NO_DATA_FLASH Write to the FCR register at the end of the function
flash_df_write(g_current_parameters.src_addr,
g_current_parameters.dest_addr);
#endif
}
#if (FLASH_CFG_CODE_FLASH_ENABLE == 1)
else if (FLASH.FENTRYR.WORD == 0x0001)
{
Write to the FCR register at the end of the function
flash_cf_write(g_current_parameters.src_addr,
g_current_parameters.dest_addr);
}
#endif
Interval A
Interval B
g_current_parameters.total_count--; Decrement the total count variable
if (g_current_parameters.current_operation == FLASH_CUR_STOP) // err
occurred; addr known good; protection err
{
err = FLASH_ERR_ACCESSW;
break;
}
}
(Omitted)
```

(2) An FRDYI interrupt and the problem

In "flash_write" in "r_flash_nofcu.c", if an FRDYI interrupt is requested while the total count variable is not decremented, the if statement in the following FRDYI interrupt process is fulfilled. As the result, Process C or D is executed, causing excessive data (the minimum program size x 1) to be written.

```

R_BSP_ATTRIB_STATIC_INTERRUPT void Excep_FCU_FIFERR(void)
{
  (Omitted)
  else if (FLASH_CUR_CF_BGO_WRITE ==
g_current_parameters.current_operation)
  {
    err = flash_wait_frdy();
    if (FLASH_SUCCESS == err)
    {
      if (g_current_parameters.total_count > 0)
      {
        g_current_parameters.src_addr += FLASH_CF_MIN_PGM_SIZE;
        g_current_parameters.dest_addr += FLASH_CF_MIN_PGM_SIZE;
        g_current_parameters.wait_cnt = WAIT_MAX_ROM_WRITE;
        g_current_parameters.total_count--;

        flash_cf_write(g_current_parameters.src_addr,
g_current_parameters.dest_addr);

        return;
      }
    }
  }
  (Omitted)
  else if (FLASH_CUR_DF_BGO_WRITE ==
g_current_parameters.current_operation)
  {
    err = flash_wait_frdy();
    if (FLASH_SUCCESS == err)
    {
      if (g_current_parameters.total_count > 0)
      {
        g_current_parameters.src_addr += FLASH_DF_MIN_PGM_SIZE;
        g_current_parameters.dest_addr += FLASH_DF_MIN_PGM_SIZE;
        g_current_parameters.wait_cnt = WAIT_MAX_DF_WRITE;
        g_current_parameters.total_count--;

        flash_df_write(g_current_parameters.src_addr,
g_current_parameters.dest_addr);

        return;
      }
    }
  }
  (Omitted)
}

```

Applied if Condition 2 is met in Interval B and the total count variable is not decremented

Process C

Applied if Condition 2 is met in Interval A and the total count variable is not decremented

Process D

3.5 Workaround

Refer to the following "r_flash_nofcu.c" and change the functions written in red.

Before modification

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
(Omitted)
while(g_current_parameters.total_count > 0)
{
/* Conversion to the P/E address from the read address */
if (FLASH.FENTRYR.WORD == 0x0080)
{
#ifdef FLASH_NO_DATA_FLASH
flash_df_write(g_current_parameters.src_addr,
g_current_parameters.dest_addr);
#endif
}
#if (FLASH_CFG_CODE_FLASH_ENABLE == 1)
else if (FLASH.FENTRYR.WORD == 0x0001)
{
flash_cf_write(g_current_parameters.src_addr,
g_current_parameters.dest_addr);
}
#endif

g_current_parameters.total_count--;

if (g_current_parameters.current_operation == FLASH_CUR_STOP) // err
occurred; addr known good; protection err
{
err = FLASH_ERR_ACCESSW;
break;
}
}
(Omitted)
```

After modification

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
(Omitted)
    while(g_current_parameters.total_count > 0)
    {
        g_current_parameters.total_count--;

        /* Conversion to the P/E address from the read address */
        if (FLASH.FENTRYR.WORD == 0x0080)
        {
#ifdef FLASH_NO_DATA_FLASH
            flash_df_write(g_current_parameters.src_addr,
g_current_parameters.dest_addr);
#endif
        }
        #if (FLASH_CFG_CODE_FLASH_ENABLE == 1)
            else if (FLASH.FENTRYR.WORD == 0x0001)
            {
                flash_cf_write(g_current_parameters.src_addr,
g_current_parameters.dest_addr);
            }
        #endif

        if (g_current_parameters.current_operation == FLASH_CUR_STOP) // err
occurred; addr known good; protection err
        {
            err = FLASH_ERR_ACCESSW;
            break;
        }
    }
(Omitted)
}
```

3.6 Schedule for Fixing the Problem

The problem will be fixed in the next version.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Dec.01.21	-	First edition issued

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.

The past news contents have been based on information at the time of publication. Now changed or invalid information may be included.

The URLs in the Tool News also may be subject to change or become invalid without prior notice.

Corporate Headquarters

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

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/