[Notes]

RX600 & RX200 Series Simple Flash API for RX

R20TS0805EJ0100
Rev.1.00
Feb. 01, 2022

Outline

When using the RX600 & RX200 series simple flash API for RX, note the following points.

1. Notes on the erasure when R_FlashErase or R_FlashEraseRange is executed for the data flash in the non-blocking mode (BGO)
2. Notes on the error response when R_FlashWrite is executed with a correct argument in the non-blocking mode (BGO)
3. Notes on the execution of the API function after R_FlashWrite has failed in the non-blocking mode (BGO)
4. Notes on the case when an FCU command is made in the API function, a timeout occurs, and FLASH_FAILURE returns
5. Notes on the demo program (flash_api_demo.c)

1. Notes on the Erasure When R_FlashErase or R_FlashEraseRange Is Executed for the Data Flash in the Non-Blocking Mode (BGO)

1.1 Applicable Products

(1) RX600 & RX200 series simple flash API for RX (flash API)

The applicable revisions and documents are as follows.

Table 1.1 Flash API applicable products

Revision	Document number
Rev.2.10	R01AN0544EU0210
Rev.2.20	R01AN0544EU0220
Rev.2.30	R01AN0544EU0230
Rev.2.40	R01AN0544EU0240
Rev.2.50	R01AN0544EU0250

(2) The application note related to the problem

The problem may occur when any of the flash API in (1) is used with other products.

The application note in the link below is related to the problem.

- RX600 & RX200 Series Virtual EEPROM for RX (R01AN0724EU0170)

  https://www.renesas.com/jp/en/search?keywords=R01AN0724

1.2 Applicable Devices

RX630, RX631, RX63N, RX63T groups

RX210, RX21A, RX220 groups

1.3 Details

When the flash API function R_FlashErase or R_FlashEraseRange is executed for the data flash in the non-blocking mode (BGO), the function may erase only the first block specified, and the remaining blocks may be left.

1.4 Conditions

The problem occurs when the following conditions are met.

Condition 1: The flash API is set to the non-blocking mode (BGO).

Condition 2: The data flash is set to be erased.

Condition 3: Refer to the code snippets in 1.5, (1). The Process A in the main routine takes longer than the erasure time.

1.5 Why the Problem Occurs

The following shows an example of Condition 3 with R_FlashErase, in which an interrupt has occurred during the Process A.

(1) The main routine process and the problem

Refer to the code snippets. In R_FlashErase, the erase command (flash_erase_command) is executed, the Process A follows it, and the erase size variable (g_bgo_bytes) is set. If the Process A takes longer than the erasure time, an FRDY interrupt is requested in the main routine before the variable is set.

```
uint8_t R_FlashErase (uint32_t block)
{
(Omitted)
    /* Erase real data flash blocks until the 'fake' block is erased .*/
    while (0 < bytes_to_erase){
                                                          Erase command
        /* Send FCU command to erase block */
        result = flash_erase_command((FCU_BYTE_PTR)p_addr);
        /* Advance pointer to next block */
        p_addr += DF_ERASE_BLOCK_SIZE;

        /* Subtract off bytes erased */                          Process A
        bytes_to_erase -= DF_ERASE_BLOCK_SIZE;

#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)
        /* Set global variables so that erase can continue in ISR. */
        g_bgo_flash_addr = p_addr;
        g_bgo_bytes = bytes_to_erase;
                                          Set the erase size variable
        /* Return, check result and continue erasure later in ISR */
        return FLASH_SUCCESS;
#endif
(Omitted)
```

(2) The process in the FRDY interrupt

The following shows flash_ready_isr, the function in the FRDY interrupt routine.

If an FRDY interrupt is requested and flash_ready_isr is executed while the erase size variable has not been set in the main routine, the subsequent if statement is not satisfied. As the result, the Process B is not executed, and the erase commands for the subsequent blocks are not made.

```
void flash_ready_isr (void)
{
(Omitted)
    /* Check state and see if anything else needs to be done */
    if( g_flash_state == FLASH_ERASING )
    {
        /* Erase is done */
#if defined(DF_GROUPED_BLOCKS)
        /* If we are erasing data flash then we need to see if all requested
           blocks are erased. */
        if( FLD_PE_MODE == g_current_mode )
        {
            /* Check to see if there are more bytes to erase. */
            if( 0 < g_bgo_bytes ) {

                /* Send FCU command to erase block */
                ret = flash_erase_command((FCU_BYTE_PTR)g_bgo_flash_addr);

                /* Advance pointer to next block */
                g_bgo_flash_addr += DF_ERASE_BLOCK_SIZE;

                /* Subtract off bytes erased */
                g_bgo_bytes -= DF_ERASE_BLOCK_SIZE;

                /* Only continue if last command was successful */
                if( ret == FLASH_SUCCESS )
                {
                    /* Exit ISR */
                    return;
                }
            }
        }
#endif

        /* Leave Program/Erase Mode */
        exit_pe_mode(g_bgo_flash_addr);

        /* Release flash state */
        flash_release_state();

        /* Flash operation finished callback function */
        FlashEraseDone();
    }
(Omitted)
```

If statement with erase size variable

Process B

1.6 Workaround

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

Before modification (R_FlashErase)

```
uint8_t R_FlashErase (uint32_t block)
{
    /* Declare address pointer */
    uint32_t p_addr;
    /* Declare erase operation result container variable */
    uint8_t  result = FLASH_SUCCESS;

    /* Make sure valid block was input. */
    if (false == flash_valid_block_check(block))
    {
        return FLASH_ERROR_ADDRESS;
    }

    /* Do we want to erase a Data Flash block or ROM block? */
    if( block >= BLOCK_DB0 )
    {
        /* Set current FCU mode to data flash PE */
        g_current_mode = FLD_PE_MODE;
    }
    else
    {
#if defined(FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING)
        /* Set current FCU mode to ROM PE */
        g_current_mode = ROM_PE_MODE;
#else
        /* ROM operations are not enabled! Enable them in
r_flash_api_rx_config.h */
        return FLASH_FAILURE;
#endif
    }

(Omitted)

    /* Erase real data flash blocks until the 'fake' block is erased .*/
    while (0 < bytes_to_erase){

        /* Send FCU command to erase block */
        result = flash_erase_command((FCU_BYTE_PTR)p_addr);

        /* Advance pointer to next block */
        p_addr += DF_ERASE_BLOCK_SIZE;

        /* Subtract off bytes erased */
        bytes_to_erase -= DF_ERASE_BLOCK_SIZE;

#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)
        /* Set global variables so that erase can continue in ISR. */
        g_bgo_flash_addr = p_addr;
        g_bgo_bytes = bytes_to_erase;

        /* Return, check result and continue erasure later in ISR */
        return FLASH_SUCCESS;
#endif
```

```
(Omitted)

#if defined(FLASH_API_RX_CFG_ROM_BGO)
    if( g_current_mode == ROM_PE_MODE )
    {
        /* Set global variable in case an error occurs and it needs to be
         * cleared in the flash ready interrupt later. */
        g_bgo_flash_addr = p_addr;

        /* Return, check result later in ISR */
        return FLASH_SUCCESS;
    }
#endif

#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)
    if( g_current_mode == FLD_PE_MODE )
    {
        /* Return, check result later in ISR */
        return FLASH_SUCCESS;
    }
#endif
(Omitted)
```

After modification (R_FlashErase)

```
uint8_t R_FlashErase (uint32_t block)
{
    /* Declare address pointer */
    uint32_t p_addr;
    /* Declare erase operation result container variable */
    uint8_t  result = FLASH_SUCCESS;

    uint8_t current_mode = READ_MODE;

    /* Make sure valid block was input. */
    if (false == flash_valid_block_check(block))
    {
        return FLASH_ERROR_ADDRESS;
    }

    /* Do we want to erase a Data Flash block or ROM block? */
    if( block >= BLOCK_DB0 )
    {
        /* Set current FCU mode to data flash PE */
        g_current_mode = FLD_PE_MODE;
        current_mode = FLD_PE_MODE;
    }
    else
    {
#if defined(FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING)
        /* Set current FCU mode to ROM PE */
        g_current_mode = ROM_PE_MODE;
        current_mode = ROM_PE_MODE;
#else
        /* ROM operations are not enabled! Enable them in
r_flash_api_rx_config.h */
```

```
        return FLASH_FAILURE;
#endif
    }

(Omitted)

#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)
    /* Set global variables so that erase can continue in ISR. */
    g_bgo_flash_addr = p_addr + DF_ERASE_BLOCK_SIZE;
    g_bgo_bytes      = bytes_to_erase - DF_ERASE_BLOCK_SIZE;
#endif

    /* Erase real data flash blocks until the 'fake' block is erased .*/
    while (0 < bytes_to_erase){

        /* Send FCU command to erase block */
        result = flash_erase_command((FCU_BYTE_PTR)p_addr);

        /* Advance pointer to next block */
        p_addr += DF_ERASE_BLOCK_SIZE;

        /* Subtract off bytes erased */
        bytes_to_erase -= DF_ERASE_BLOCK_SIZE;

#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)

        /* Return, check result and continue erasure later in ISR */
        return FLASH_SUCCESS;
#endif

(Omitted)

#if defined(FLASH_API_RX_CFG_ROM_BGO)
    if( current_mode == ROM_PE_MODE )
    {
        /* Set global variable in case an error occurs and it needs to be
         * cleared in the flash ready interrupt later. */
        g_bgo_flash_addr = p_addr;

        /* Return, check result later in ISR */
        return FLASH_SUCCESS;
    }
#endif

#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)
    if( current_mode == FLD_PE_MODE )
    {
        /* Return, check result later in ISR */
        return FLASH_SUCCESS;
    }
#endif

(Omitted)
```

Before modification (R_FlashEraseRange)

```
uint8_t R_FlashEraseRange (uint32_t start_addr, uint32_t bytes)
{
(Omitted)
    /* Erase real data flash blocks until the 'fake' block is erased .*/
    while(0 < bytes) {

        /* Send FCU command to erase block */
        result = flash_erase_command((FCU_BYTE_PTR)start_addr);

        /* Advance pointer to next block */
        start_addr += DF_ERASE_BLOCK_SIZE;

        /* Subtract off bytes erased */
        bytes -= DF_ERASE_BLOCK_SIZE;

#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)
        /* Set global variables so that erase can continue in ISR. */
        g_bgo_flash_addr = start_addr;
        g_bgo_bytes = bytes;

        /* Return, check result and continue erasure later in ISR */
        return FLASH_SUCCESS;
#endif
(Omitted)
```

After modification (R_FlashEraseRange)

```
uint8_t R_FlashEraseRange (uint32_t start_addr, uint32_t bytes)
{
(Omitted)
#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)
    /* Set global variables so that erase can continue in ISR. */
    g_bgo_flash_addr = start_addr + DF_ERASE_BLOCK_SIZE;
    g_bgo_bytes      = bytes - DF_ERASE_BLOCK_SIZE;
#endif

    /* Erase real data flash blocks until the 'fake' block is erased .*/
    while(0 < bytes) {

        /* Send FCU command to erase block */
        result = flash_erase_command((FCU_BYTE_PTR)start_addr);

        /* Advance pointer to next block */
        start_addr += DF_ERASE_BLOCK_SIZE;

        /* Subtract off bytes erased */
        bytes -= DF_ERASE_BLOCK_SIZE;

#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)

        /* Return, check result and continue erasure later in ISR */
        return FLASH_SUCCESS;
#endif
(Omitted)
```

1.7 Schedule for Fixing the Problem

The problem will be fixed in the next version.

2. Notes on the Error Response When R_FlashWrite Is Executed with a Correct Argument in the Non-Blocking Mode (BGO)

2.1 Applicable Products

(1) RX600 & RX200 series simple flash API for RX (flash API)

The applicable revisions and documents are as follows.

Table 2.1 Flash API applicable products

Revision	Document number
Rev.2.30	R01AN0544EU0230
Rev.2.40	R01AN0544EU0240
Rev.2.50	R01AN0544EU0250

(2) The application note related to the problem

The problem may occur when any of the flash API in (1) is used with other products.

The application note in the link below is related to the problem.

- RX600 & RX200 Series Virtual EEPROM for RX (R01AN0724EU0170)
  https://www.renesas.com/jp/en/search?keywords=R01AN0724

2.2 Applicable Devices

RX610 group

RX621, RX62N, RX62T, RX62G groups

RX630, RX631, RX63N, RX63T groups

RX210, RX21A, RX220 groups

2.3 Details

When the flash API function R_FlashWrite is executed in the non-blocking mode (BGO), data may not be written successfully even though the write data and write address are correct. In this case, the function returns FLASH_FAILURE, and the callback function FlashError is executed.

2.4 Conditions

The problem occurs when the following conditions are met.

Condition 1: The flash API is set to the non-blocking mode (BGO).

Condition 2: Refer to the code snippets in 2.5, (1). The Process A in the main routine takes longer than the programming time.

2.5 Why the Problem Occurs

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

(1) The main routine process and the problem

Refer to the code snippets. In R_FlashWrite, data_flash_write or rom_write is called and the program commands are executed, the Process A follows them, and the conditional branches with a mode variable (g_current_mode) are made. If the Process A takes longer than the programming time, an FRDY interrupt is requested before the conditional branch.

```
uint8_t R_FlashWrite (uint32_t flash_addr,
                      uint32_t buffer_addr,
                      uint16_t bytes)
{
(Omitted)
#ifdef FLASH_API_RX_CFG_FLASH_TO_FLASH
       if( g_flash_to_flash_op == 1 )
       {
           if( g_current_mode == FLD_PE_MODE )          Program commands
           {
               result = data_flash_write( flash_addr,
                                          (uint32_t)&g_temp_array[0],
                                          num_byte_to_write);
           }
           else
           {
               result = rom_write( flash_addr, (uint32_t)&g_temp_array[0],
                                   num_byte_to_write);
           }
       }
       else
       {
           if( g_current_mode == FLD_PE_MODE )
           {
               result = data_flash_write( flash_addr, buffer_addr,
                                          num_byte_to_write);
           }
           else
           {
               result = rom_write( flash_addr, buffer_addr,
                                   num_byte_to_write);
           }
       }
#else
```

```
        if( g_current_mode == FLD_PE_MODE )
        {
            result = data_flash_write(flash_addr, buffer_addr,
                                      num_byte_to_write);
        }
        else
        {
            result = rom_write(flash_addr, buffer_addr, num_byte_to_write);
        }
#endif


        /* Check the container variable result for errors */
        if( result != FLASH_SUCCESS )
        {
            /* Data flash write error detected, break from flash write
               while loop prematurely */
            break;
        }
#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)
        if( g_current_mode == FLD_PE_MODE )
        {
            /* Return FLASH_SUCCESS, rest of programming will be done
               in interrupt */
            return FLASH_SUCCESS;
        }
#endif
#if defined(FLASH_API_RX_CFG_ROM_BGO)
        if( g_current_mode == ROM_PE_MODE )
        {
            /* Return FLASH_SUCCESS, rest of programming will be done
               in interrupt */
            return FLASH_SUCCESS;
        }
#endif
(Omitted)
```

Process A

Conditional branch with mode variable

Conditional branch with mode variable

(2) The process in the FRDY interrupt

The following shows flash_ready_isr, the function in the FRDY interrupt routine.

If flash_ready_isr is executed due to an FRDY interrupt request while the conditional branches with a mode variable have not been completed in the main routine, flash_release_state is executed at the end of the programming and the mode variable (g_current_mode) is changed to READ_MODE.

Back in the main routine, the conditional branches with a mode variable (g_current_mode) are not satisfied. As the result, unintended processing is performed and the problem in 2.3 occurs.

```
void flash_ready_isr (void)
{
(Omitted)
   else if( g_flash_state == FLASH_WRITING )
   {
(Omitted)
          /* Check the result for errors */
          if( ret != FLASH_SUCCESS )
          {
              /* Error detected during programming, stop and return */
              /* Leave Program/Erase Mode and clear any error flags */
              exit_pe_mode(g_bgo_flash_addr);

              /* Release flash state */
              flash_release_state();

              /* Operation failure, use callback function to alert user */
              FlashError();

              /* Exit ISR */
              return;
          }
(Omitted)
```

Change the mode variable

2.6 Workaround

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

Before modification

```
uint8_t R_FlashWrite (uint32_t flash_addr,
                      uint32_t buffer_addr,
                      uint16_t bytes)
{
    /* Declare result container and number of bytes to write variables */
    uint8_t  result = FLASH_SUCCESS;
    uint32_t num_byte_to_write;
#ifdef FLASH_API_RX_CFG_FLASH_TO_FLASH
    /* Local variable when using FLASH_API_RX_CFG_FLASH_TO_FLASH */
    uint16_t i;
#endif

(Omitted)

    /* Do we want to program a DF area or ROM area? */
    if( flash_addr < g_flash_BlockAddresses[ROM_NUM_BLOCKS-1] )
    {
        /* Set current FCU mode to data flash PE */
        g_current_mode = FLD_PE_MODE;
    }
    else
    {
        /* Set FCU to ROM PE mode */
        g_current_mode = ROM_PE_MODE;
    }

(Omitted)

#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)
       if( g_current_mode == FLD_PE_MODE )
       {
           /* Return FLASH_SUCCESS, rest of programming will be done
              in interrupt */
           return FLASH_SUCCESS;
       }
#endif
#if defined(FLASH_API_RX_CFG_ROM_BGO)
       if( g_current_mode == ROM_PE_MODE )
       {
           /* Return FLASH_SUCCESS, rest of programming will be done
              in interrupt */
           return FLASH_SUCCESS;
       }
#endif

(Omitted)
```

After modification

```
uint8_t R_FlashWrite (uint32_t flash_addr,
                      uint32_t buffer_addr,
                      uint16_t bytes)
{
    /* Declare result container and number of bytes to write variables */
    uint8_t  result = FLASH_SUCCESS;
    uint32_t num_byte_to_write;
    uint8_t current_mode = READ_MODE;
#ifdef FLASH_API_RX_CFG_FLASH_TO_FLASH
    /* Local variable when using FLASH_API_RX_CFG_FLASH_TO_FLASH */
    uint16_t i;
#endif

(Omitted)

    /* Do we want to program a DF area or ROM area? */
    if( flash_addr < g_flash_BlockAddresses[ROM_NUM_BLOCKS-1] )
    {
        /* Set current FCU mode to data flash PE */
        g_current_mode = FLD_PE_MODE;
        current_mode = FLD_PE_MODE;
    }
    else
    {
        /* Set FCU to ROM PE mode */
        g_current_mode = ROM_PE_MODE;
        current_mode = ROM_PE_MODE;
    }

(Omitted)

#if defined(FLASH_API_RX_CFG_DATA_FLASH_BGO)
       if( current_mode == FLD_PE_MODE )
       {
           /* Return FLASH_SUCCESS, rest of programming will be done
              in interrupt */
           return FLASH_SUCCESS;
       }
#endif
#if defined(FLASH_API_RX_CFG_ROM_BGO)
       if( current_mode == ROM_PE_MODE )
       {
           /* Return FLASH_SUCCESS, rest of programming will be done
              in interrupt */
           return FLASH_SUCCESS;
       }
#endif

(Omitted)
```

2.7 Schedule for Fixing the Problem

The problem will be fixed in the next version.

3. Notes on the Execution of the API Function After R_FlashWrite Has Failed in the Non-Blocking Mode (BGO)

3.1 Applicable Products

(1) RX600 & RX200 series simple flash API for RX (flash API)

The applicable revisions and documents are as follows.

Table 3.1 Flash API applicable products

Revision	Document number
Rev.2.30	R01AN0544EU0230
Rev.2.40	R01AN0544EU0240
Rev.2.50	R01AN0544EU0250

(2) The application note related to the problem

The problem may occur when any of the flash API in (1) is used with other products.

The application note in the link below is related to the problem.

- RX63N-256K Renesas Starter Kit Sample Code for e2 studio (R01AN2507EG0100)
  https://www.renesas.com/jp/en/search?keywords=R01AN2507

3.2 Applicable Devices

RX610 group

RX621, RX62N, RX62T, RX62G groups

RX630, RX631, RX63N, RX63T groups

RX210, RX21A, RX220 groups

3.3 Details

If the flash API function R_FlashWrite is executed in the non-blocking mode (BGO), programming for a certain address fails, and an API function is executed after the failure, FLASH_FAILURE may return.

3.4 Conditions

The problem occurs when the following conditions are met.

Condition 1: The flash API is set to the non-blocking mode (BGO).

Condition 2: Programming for a certain address in a ROM area has failed.

Condition 3: A program error has occurred (a program for an area protected by lock bits, or a program for an already-programmed area).

The following shows an example of Condition 2 with a RX63N group 2MB ROM capacity product.

In a RX63N group product, there are four ROM areas (the boundary is 512KB). Table 3.2 shows the areas and addresses. The problem occurs when programming for any of the addresses has failed.

The program address is calculated as follows.

Program address = Start address of the area - program unit (e.g., RX63N group = 128B)

Table 3.2 RX63N group 2MB ROM capacity product

Area	Address range (read address)	Program address (read address)
Area 3	FFE0 0000h - FFE7 FFFFh	0xFFE7FF80
Area 2	FFE8 0000h - FFEF FFFFh	0xFFEFFF80
Area 1	FFF0 0000h - FFF7 FFFFh	0xFFF7FF80
Area 0	FFF8 0000h - FFFF FFFFh	0xFFFFFF80

3.5 Why the Problem Occurs

When an error occurs, the flash API makes an FCU command during the FRDY interrupt process in exit_pe_mode to clear the error. Normally, the command is made to a correct area; however, if the conditions above are met, the command is made to a wrong area (e.g., if failed to write to area 3, the command is made to area 2). As the result, the process is interrupted and the error continues. If the subsequent API function is executed in the state, the problem in 3.3 occurs.

3.6 Workaround

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

Before modification

```
static void exit_pe_mode (uint32_t flash_addr)
{
    /* Declare wait timer count variable */
    volatile int32_t wait_cnt;

    /* Declare address pointer */
    FCU_BYTE_PTR p_addr;

    /* Cast flash address so that it can be used as pointer and will be
       accessed correctly. */
    p_addr = (FCU_BYTE_PTR)flash_addr;

    /* Set wait timer count duration */
    wait_cnt = WAIT_MAX_ERASE;

    /* Iterate while loop whilst FCU operation is in progress */
    while(FLASH.FSTATR0.BIT.FRDY == 0)
    {
        /* Decrement wait timer count variable */
        wait_cnt--;

        /* Check if wait timer count value has reached zero */
        if(wait_cnt == 0)
        {
            /* Timeout duration has elapsed, assuming operation failure and
               resetting the FCU */
            flash_reset();

            /* Break from the while loop prematurely */
            break;
        }
    }

    /* Check FSTATR0 and execute a status register clear command if needed */
    if(     (FLASH.FSTATR0.BIT.ILGLERR == 1)
        ||  (FLASH.FSTATR0.BIT.ERSERR  == 1)
        ||  (FLASH.FSTATR0.BIT.PRGERR  == 1))
    {

        /* Clear ILGLERR */
        if(FLASH.FSTATR0.BIT.ILGLERR == 1)
        {
            /* FASTAT must be set to 0x10 before the status clear command
               can be successfully issued  */
            if(FLASH.FASTAT.BYTE != 0x10)
            {
                /* Set the FASTAT register to 0x10 so that a status clear
                   command can be issued */
                FLASH.FASTAT.BYTE = 0x10;
            }
        }
```

```
        /* Send status clear command to FCU */
        *p_addr = 0x50;
    }
(Omitted)
```

After modification

```
static void exit_pe_mode(uint32_t flash_addr)
{
    /* Declare wait timer count variable */
    volatile int32_t wait_cnt;

    /* Declare address pointer */
    FCU_BYTE_PTR p_addr;

    /* Set wait timer count duration */
    wait_cnt = WAIT_MAX_ERASE;

    /* Iterate while loop whilst FCU operation is in progress */
    while (0 == FLASH.FSTATR0.BIT.FRDY)
    {
        /* Decrement wait timer count variable */
        wait_cnt--;

        /* Check if wait timer count value has reached zero */
        if (0 == wait_cnt)
        {
            /* Timeout duration has elapsed, assuming operation failure and
               resetting the FCU */
            flash_reset();

            /* Break from the while loop prematurely */
            break;
        }
    }

    /* Check FSTATR0 and execute a status register clear command if needed */
    if (   (1 == FLASH.FSTATR0.BIT.ILGLERR)
        || (1 == FLASH.FSTATR0.BIT.ERSERR)
        || (1 == FLASH.FSTATR0.BIT.PRGERR))
    {

        /* Set pointer to command area  */
        if (0x0001 == FLASH.FENTRYR.WORD)
        {
            /* Area 0 */
            p_addr = (FCU_BYTE_PTR) (g_flash_BlockAddresses[0]);
        }
#if defined(ROM_AREA_1)
        else if (0x0002 == FLASH.FENTRYR.WORD)
        {
            /* Area 1 */
            p_addr = (FCU_BYTE_PTR) (ROM_AREA_0 - ROM_PROGRAM_SIZE);
        }
#endif
#if defined(ROM_AREA_2)
```

```
        else if (0x0004 == FLASH.FENTRYR.WORD)
        {
            /* Area 2 */
            p_addr = (FCU_BYTE_PTR) (ROM_AREA_1 - ROM_PROGRAM_SIZE);
        }
#endif
#if defined(ROM_AREA_3)
        else if (0x0008 == FLASH.FENTRYR.WORD)
        {
            /* Area 3 */
            p_addr = (FCU_BYTE_PTR) (ROM_AREA_2 - ROM_PROGRAM_SIZE);
        }
#endif
        else if (0x0080 == FLASH.FENTRYR.WORD)
        {
            /* Data flash area */
            p_addr = (FCU_BYTE_PTR) (DF_ADDRESS);
        }
        else
        {
            /* Data flash area */
            p_addr = (FCU_BYTE_PTR) (DF_ADDRESS);

            /* Enter data flash P/E mode */
            /* Set FENTRYD bit(Bit 7) and FKEY (B8-15 = 0xAA) */
            FLASH.FENTRYR.WORD = 0xAA80;
        }

            /* Clear ILGLERR */
        if (1 == FLASH.FSTATR0.BIT.ILGLERR)
        {
            /* FASTAT must be set to 0x10 before the status clear command
             can be successfully issued  */
            if (0x10 != FLASH.FASTAT.BYTE)
            {
                /* Set the FASTAT register to 0x10 so that a status clear
                 command can be issued */
                FLASH.FASTAT.BYTE = 0x10;
            }
        }

        /* Send status clear command to FCU */
        *p_addr = 0x50;
    }
(Omitted)
```

3.7 Schedule for Fixing the Problem

The problem will be fixed in the next version.

4. Notes on the Case When an FCU Command is Made in the API Function, a Timeout Occurs, and FLASH_FAILURE Returns

4.1 Applicable Products

(1) RX600 & RX200 series simple flash API for RX (flash API)

The applicable revisions and documents are as follows.

Table 4.1 Flash API applicable products

Revision	Document number
Rev.2.00	R01AN0544EU0200
Rev.2.10	R01AN0544EU0210
Rev.2.20	R01AN0544EU0220
Rev.2.30	R01AN0544EU0230
Rev.2.40	R01AN0544EU0240
Rev.2.50	R01AN0544EU0250

(2) The application note related to the problem

The problem may occur when any of the flash API in (1) is used with other products.

The application note in the link below is related to the problem.

- RX63N Group, RX631 Group Flash Bootloader with the USB Peripheral CDC (R01AN1710JJ0100)

  https://www.renesas.com/jp/en/search?keywords=R01AN1710

4.2 Applicable Devices

RX610 group

RX621, RX62N, RX62T, RX62G groups

RX630, RX631, RX63N, RX63T groups

RX210, RX21A, RX220 groups

4.3 Details

If an FCU command is interrupted in any of the flash API functions below, a timeout occurs, and FLASH_FAILURE returns, the subsequent API function may not be executed properly.

Functions: R_FlashErase, R_FlashEraseRange, R_FlashWrite, R_FlashProgramLockBit

4.4 Conditions

The problem occurs when the following conditions are met.

Condition 1: Any of the following functions: R_FlashErase, R_FlashEraseRange, R_FlashWrite, R_FlashProgramLockBit

Condition 2: The flash API is set to the blocking mode.

Condition 3: After an FCU command is made, FSTATR0.FRDY bit has not become ”1”, and a timeout and FLASH_FAILURE have occurred.

4.5 Why the Problem Occurs

When the conditions above are met, flash_reset is executed. The function initializes the PCKAR register, but it does not clear the global variable (g_fcu_pclk_command).

If the global variable (g_fcu_pclk_command) is not cleared, the PCKAR register works with the initial value; therefore, the problem in 4.3 occurs.

4.6 Workaround

Refer to the following and add the function written in red in flash_reset in r_flash_api_rx.c.

Before modification

```
static void flash_reset (void)
{
(Omitted)

}
```

After modification

```
static void flash_reset (void)
{
(Omitted)
    g_fcu_pclk_command = 0;
}
```

4.7 Schedule for Fixing the Problem

The problem will be fixed in the next version.

5. Notes on the Demo Program (flash_api_demo.c)

5.1 Applicable Products

RX600 & RX200 series simple flash API for RX (flash API)

The applicable revisions and documents are as follows.

Table 5.1 Flash API applicable products

Revision	Document number
Rev.2.30	R01AN0544EU0230
Rev.2.40	R01AN0544EU0240

5.2 Applicable Devices

RX610 group

RX621, RX62N, RX62T, RX62G groups

RX630, RX631, RX63N, RX63T groups

RX210, RX21A, RX220 groups

5.3 Details

If a program is executed in r_flash_api_rx_config.h with the ROM programming and the ROM programming non-blocking mode (BGO) enabled, the program runs out of control when the main function is executed.

5.4 Conditions

The problem occurs when both of the following definitions in r_flash_api_rx_config.h are valid.

Definition 1: FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING (ROM program)

Definition2: FLASH_API_RX_CFG_ROM_BGO (ROM program non-blocking mode <BGO>)

5.5 Workaround

Refer to the following demo program (flash_api_demo.c) and change the functions written in red.

Before modification

```
(Omitted)
#ifdef FLASH_API_RX_CFG_ROM_BGO

/* We will also need some RAM space to hold the vector table */
static uint32_t ram_vector_table[256];

/* If using ROM BGO then this sample code needs to be in RAM */
#pragma section FRAM
#endif

(Omitted)
/**********************************************************************
**
* Function Name: flash_api_demo_rom_tests
* Description  : Tests out the Flash API on the ROM
* Arguments    : none
* Return Value : none
**********************************************************************
*/
(Omitted)
```

After modification

```
(Omitted)
#ifdef FLASH_API_RX_CFG_ROM_BGO

/* We will also need some RAM space to hold the vector table */
static uint32_t ram_vector_table[256];

#endif
(Omitted)
/* If using ROM BGO then this sample code needs to be in RAM */
#ifdef FLASH_API_RX_CFG_ROM_BGO
#pragma section FRAM
#endif
/**********************************************************************
**
* Function Name: flash_api_demo_rom_tests
* Description  : Tests out the Flash API on the ROM
* Arguments    : none
* Return Value : none
**********************************************************************
*/
(Omitted)
```

5.6 Schedule for Fixing the Problem

The problem will be fixed in the next version.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Feb.01.22	-	First edition issued

Corporate Headquarters

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

Trademarks

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/

TS Colophon 4.3