【注意事項】

RX600&RX200 シリーズ RX 用シンプルフラッシュ API

R20TS0805JJ0100
Rev.1.00
2022.02.01 号

概要

RX600 & RX200 シリーズ RX 用シンプルフラッシュ API の使用上の注意事項を連絡します。

1. ノンブロッキングモード(BGO)でデータフラッシュに対して R_FlashErase 関数または R_FlashEraseRange 関数を実行した際の消去に関する注意事項
2. ノンブロッキングモード(BGO)で R_FlashWrite 関数を正常な引数で実行した際のエラー応答に関する注意事項
3. ノンブロッキングモード(BGO)で R_FlashWrite 関数を実行し、エラーとなる場合の後続の API 関数実行に関する注意事項
4. API 関数で FCU コマンド発行後、処理が完了せずにタイムアウトで “FLASH_FAILURE” が返ってきた場合の注意事項
5. デモプログラム (flash_api_demo.c) に関する注意事項

1. ノンブロッキングモード(BGO)でデータフラッシュに対して R_FlashErase 関数または R_FlashEraseRange 関数を実行した際の消去に関する注意事項

1.1 該当製品

(1) RX600 & RX200 シリーズ RX 用シンプルフラッシュ API

（以下、フラッシュ API）

該当するリビジョンおよび資料番号は、以下のとおりです。

表 1.1 フラッシュ API 該当製品一覧

フラッシュ API のリビジョン	資料番号
Rev.2.10	R01AN0544JU0210
Rev.2.20	R01AN0544JU0220
Rev.2.30	R01AN0544JU0230
Rev.2.40	R01AN0544JU0240
Rev.2.50	R01AN0544EU0250

(2) フラッシュ API と組み合わせて使用されるアプリケーションノートについて

(1)のフラッシュ API と組み合わせて使用されることにより、問題となる現象が発生する可能性があります。

一例となりますが以下のアプリケーションノートは該当します。

- RX600 & RX200 シリーズ RX 用仮想 EEPROM (R01AN0724JU0170)
  https://www.renesas.com/jp/ja/search?keywords=R01AN0724

1.2 該当デバイス

RX630、RX631、RX63N、RX63T グループ

RX210、RX21A、RX220 グループ

1.3 内容

フラッシュ API の R_FlashErase 関数または R_FlashEraseRange 関数をデータフラッシュに対してノンブロッキングモード(BGO)で実行した場合、指定された最初のブロックのみ消去され、以降のブロックが消去されない可能性があります。

1.4 発生条件

以下の条件が全て成立することにより問題となる現象が発生します。

条件 1： フラッシュ API をノンブロッキングモード(BGO)の設定で使用
条件 2： 消去対象の領域をデータフラッシュに指定
条件 3： メインルーチン内の特定区間(下図(1)の区間 A)の処理時間がイレーズ時間より長い

1.5 発生理由

以下、R_FlashErase 関数を例に発生条件 3 において区間 A で割り込みが発生した場合を説明します。

(1) メインルーチン内の処理と今回の問題

以下、R_FlashErase 関数の flash_erase_command 関数内でイレーズコマンドが実行されてから、イレーズサイズ変数(g_bgo_bytes)を設定するまでの区間 A の処理時間がイレーズ時間よりも長い場合、メインルーチン内でイレーズサイズ変数が設定される前に FRDY 割り込み要求が発生します。

```
uint8_t R_FlashErase (uint32_t block)
{
(中略)
    /* Erase real data flash blocks until the 'fake' block is erased .*/
    while (0 < bytes_to_erase){                                  イレーズコマンドを実行

        /* 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;                                  区間 A

#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;                  イレーズサイズ変数の設定

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

(2) FRDY 割り込み内の処理

以下は FRDY 割り込みルーチンである flash_ready_isr 関数の処理です。

メインルーチン内でイレーズサイズ変数が設定されないまま、FRDY 割り込み要求が発生して flash_ready_isr 関数が実行されると、下図の if 文の条件が不成立となって区間 B が実行されず、以降のブロックに対してイレーズコマンドが発行されなくなります。

```
void flash_ready_isr (void)
{
（中略）
    /* 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();
    }
（中略）
```

イレーズサイズ変数を使った条件文

区間 B

1.6 回避策

r_flash_fcu.c 内の以下関数の赤字部分を変更してください。

修正前 (R_FlashErase 関数)

```
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
    }

（中略）

    /* 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

(中略)

#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
(中略)
```

修正後 (R_FlashErase 関数)

```
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
    }

(中略)

#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

(中略)

#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

(中略)
```

修正前 (R_FlashEraseRange 関数)

```
uint8_t R_FlashEraseRange (uint32_t start_addr, uint32_t bytes)
{
(中略)
    /* 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
(中略)
```

修正後 (R_FlashEraseRange 関数)

```
uint8_t R_FlashEraseRange (uint32_t start_addr, uint32_t bytes)
{
(中略)
#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
(中略)
```

1.7　恒久対策

次期バージョンで改修予定です。

2. ノンブロッキングモード(BGO)で R_FlashWrite 関数を正常な引数で実行した際のエラー応答に関する注意事項

2.1 該当製品

(1) RX600 & RX200 シリーズ RX 用シンプルフラッシュ API

（以下、フラッシュ API）

該当するリビジョンおよび資料番号は、以下のとおりです。

表 2.1 フラッシュ API 該当製品一覧

フラッシュ API のリビジョン	資料番号
Rev.2.30	R01AN0544JU0230
Rev.2.40	R01AN0544JU0240
Rev.2.50	R01AN0544EU0250

(2) フラッシュ API と組み合わせて使用されるアプリケーションノートについて

(1)のフラッシュ API と組み合わせて使用されることにより、問題となる現象が発生する可能性があります。

一例となりますが以下のアプリケーションノートは該当します。

- RX600 & RX200 シリーズ RX 用仮想 EEPROM (R01AN0724JU0170)
  https://www.renesas.com/jp/ja/search?keywords=R01AN0724

2.2 該当デバイス

RX610 グループ

RX621、RX62N、RX62T、RX62G グループ

RX630、RX631、RX63N、RX63T グループ

RX210、RX21A、RX220 グループ

2.3 内容

フラッシュ API の R_FlashWrite 関数をノンブロッキングモード(BGO)で実行した場合、書き込みデータと書き込みアドレスが正常であっても、データが正しく書き込めない可能性があります。このとき、関数から”FLASH_FAILURE”が返り、コールバック関数である FlashError 関数が実行されます。

2.4 発生条件

以下の条件が全て成立することにより問題となる現象が発生します。

条件 1： フラッシュ API をノンブロッキングモード(BGO)の設定で使用
条件 2： メインルーチン内の特定区間(下図(1)の区間 A)の処理時間がプログラム時間より長い

2.5 発生理由

以下では発生条件 2 において区間 A で割り込みが発生した場合を説明します。

(1) メインルーチン内の処理と今回の問題

以下、R_FlashWrite 関数で呼び出される data_flash_write 関数または rom_write 関数によってプログラムコマンドが実行されてから、モード変数(g_current_mode)を使った条件分岐までの区間 A の処理時間がプログラム時間よりも長い場合、条件分岐によって関数からリターンされる前に FRDY 割り込み要求が発生します。

```
uint8_t R_FlashWrite (uint32_t flash_addr,
                      uint32_t buffer_addr,
                      uint16_t bytes)
{
（中略）
#ifdef FLASH_API_RX_CFG_FLASH_TO_FLASH
       if( g_flash_to_flash_op == 1 )
       {
           if( g_current_mode == FLD_PE_MODE )
           {
               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);
           }
```

プログラムコマンドを実行

```
        }
#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
(中略)
```

区間 A

モード変数を使った条件分岐

モード変数を使った条件分岐

(2) FRDY 割り込み内の処理

以下は FRDY 割り込みルーチンである flash_ready_isr 関数の処理です。

メインルーチン内でモード変数を使った条件分岐が実行されないまま、FRDY 割り込み要求により flash_ready_isr 関数が実行されると、プログラム処理の最後に flash_release_state 関数が実行され、モード変数(g_current_mode)の値が READ_MODE に書き換えられます。

これにより、メインルーチンに戻った後、モード変数 (g_current_mode)を使った条件分岐が非成立となり、意図しない処理が実行されて 2.3 内容で示す問題が発生します。

```
void flash_ready_isr (void)
{
（中略）
    else if( g_flash_state == FLASH_WRITING )
    {
（中略）
            /* 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;
            }
（中略）
```

モード変数の書き換え

2.6 回避策

r_flash_api_rx.c 内の以下それぞれの関数の赤字部分を変更してください。

修正前

```
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

(中略)

    /* 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;
    }

(中略)

#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

(中略)
```

修正後

```
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

(中略)

    /* 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;
    }

(中略)

#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

(中略)
```

2.7　恒久対策

次期バージョンで改修予定です。

3. ノンブロッキングモード(BGO)で R_FlashWrite 関数を実行し、エラーとなる場合の後続の API 関数実行に関する注意事項

3.1 該当製品

(1) RX600 & RX200 シリーズ RX 用シンプルフラッシュ API

（以下、フラッシュ API）

該当するリビジョンおよび資料番号は、以下のとおりです。

表 3.1 フラッシュ API 該当製品一覧

フラッシュ API のリビジョン	資料番号
Rev.2.30	R01AN0544JU0230
Rev.2.40	R01AN0544JU0240
Rev.2.50	R01AN0544EU0250

(2) フラッシュ API と組み合わせて使用されるアプリケーションノートについて

(1)のフラッシュ API と組み合わせて使用されることにより、問題となる現象が発生する可能性があります。

一例となりますが以下のアプリケーションノートは該当します。

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

3.2 該当デバイス

RX610 グループ

RX621、RX62N、RX62T、RX62G グループ

RX630、RX631、RX63N、RX63T グループ

RX210、RX21A、RX220 グループ

3.3 内容

フラッシュ API の R_FlashWrite 関数をノンブロッキングモード(BGO)で実行し、かつ特定のアドレスへのプログラムに失敗した場合、そのあとに API 関数を実行すると” FLASH_FAILURE”が返る問題が発生します。

3.4 発生条件

以下の条件が全て成立することにより問題となる現象が発生します。

条件 1： フラッシュ API をノンブロッキングモード(BGO)の設定で使用
条件 2： 各 ROM 領域の特定アドレスへのプログラム
条件 3： プログラムエラーの発生(ロックビットによってプロテクトされた領域へのプログラム、およびプログラム済み領域へのプログラム)

条件 2 について RX63N グループの ROM 容量が 2 MB の製品を例に説明します。

RX63N グループには 512 KB を境に ROM 領域が 4 つ(領域 0～領域 3)存在します。表 3.2 に示す各領域の特定アドレスへのプログラムに失敗することで本問題が発生します。

特定プログラムアドレスの計算式は下記の通りです。

特定プログラムアドレス = 各領域の先頭アドレス – プログラム単位(例：RX63N グループは 128 B)

表 3.2 RX63N グループ ROM 容量 2 MB 製品の例

領域	アドレス範囲 (読み出し用アドレス)	特定プログラムアドレス (読み出し用アドレス)
領域 3	FFE0 0000h - FFE7 FFFFh	0xFFE7FF80
領域 2	FFE8 0000h - FFEF FFFFh	0xFFEFFF80
領域 1	FFF0 0000h - FFF7 FFFFh	0xFFF7FF80
領域 0	FFF8 0000h - FFFF FFFFh	0xFFFFFF80

3.5 発生理由

フラッシュ API は、エラーが発生すると FRDY 割り込み処理内の exit_pe_mode 関数でエラーを解除するために FCU コマンドを発行します。通常は対象領域に対して正しく FCU コマンドを発行しますが、上記の発生条件が成立すると、対象領域とは異なる領域(領域 3 への書き込みに失敗した場合は領域 2)に対して FCU コマンドを発行してしまい、エラーが解除できずに処理が終了します。その後、エラー状態のまま次の API 関数を実行することで 3.3 内容で示す問題が発生します。

3.6 回避策

”r_flash_api_rx.c”内の以下の関数の赤字部分を変更してください。

修正前

```
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;
    }
（中略）
```

修正後

```
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;
    }
(中略)
```

3.7　恒久対策

次期バージョンで改修予定です。

4. API 関数で FCU コマンド発行後、処理が完了せずにタイムアウトで “FLASH_FAILURE” が返ってきた場合の注意事項

4.1 該当製品

(1) RX600 & RX200 シリーズ RX 用シンプルフラッシュ API

（以下、フラッシュ API）

該当するリビジョンおよび資料番号は、以下のとおりです。

表 4.1 フラッシュ API 該当製品一覧

フラッシュ API のリビジョン	資料番号
Rev.2.00	R01AN0544JU0200
Rev.2.10	R01AN0544JU0210
Rev.2.20	R01AN0544JU0220
Rev.2.30	R01AN0544JU0230
Rev.2.40	R01AN0544JU0240
Rev.2.50	R01AN0544EU0250

(2) フラッシュ API と組み合わせて使用されるアプリケーションノートについて

(1)のフラッシュ API と組み合わせて使用されることにより、問題となる現象が発生する可能性があります。

一例となりますが以下のアプリケーションノートは該当します。

- RX63N グループ、RX631 グループ USB ペリフェラル CDC によるフラッシュブートローダ (R01AN1710JJ0100)
  https://www.renesas.com/jp/ja/search?keywords=R01AN1710

4.2 該当デバイス

RX610 グループ

RX621、RX62N、RX62T、RX62G グループ

RX630、RX631、RX63N、RX63T グループ

RX210、RX21A、RX220 グループ

4.3 内容

フラッシュ API の R_FlashErase 関数、R_FlashEraseRange 関数、R_FlashWrite 関数、R_FlashProgramLockBit 関数において、FCU コマンド発行後に処理が完了せず、タイムアウトで”FLASH_FAILURE”が返ってきた場合、以降 API 関数が正しく実行できない可能性があります。

4.4 発生条件

以下の条件が全て成立することにより問題となる現象が発生します。

条件 1： 対象関数は R_FlashErase 関数、R_FlashEraseRange 関数、R_FlashWrite 関数、R_FlashProgramLockBit 関数

条件 2： フラッシュ API をブロッキングモードの設定で使用
条件 3： FCU コマンド発行後、FSTATR0.FRDY ビットが”1”にならずにタイムアウトで”FLASH_FAILURE”が発生

4.5 発生理由

上記の発生条件が成立すると flash_reset 関数が実行されます。flash_reset 関数が実行されると PCKAR レジスタが初期化されますが、グローバル変数(g_fcu_pclk_command)はクリアされません。

グローバル変数(g_fcu_pclk_command)がクリアされないと、PCKAR レジスタは再設定されず、以後初期値のまま動作するため、4.3 内容に示す問題が発生します。

4.6 回避策

r_flash_api_rx.c 内の flash_reset 関数の赤字部分を変更してください。

修正前

```
static void flash_reset (void)
{
(中略)

}
```

修正後

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

4.7 恒久対策

次期バージョンで改修予定です。

5. デモプログラム (flash_api_demo.c)に関する注意事項

5.1 該当製品

RX600 & RX200 シリーズ RX 用シンプルフラッシュ API

（以下、フラッシュ API）

該当するリビジョンおよび資料番号は、以下のとおりです。

表 5.1 フラッシュ API 該当製品一覧

フラッシュ API のリビジョン	資料番号
Rev.2.30	R01AN0544JU0230
Rev.2.40	R01AN0544JU0240

5.2 該当デバイス

RX610 グループ

RX621、RX62N、RX62T、RX62G グループ

RX630、RX631、RX63N、RX63T グループ

RX210、RX21A、RX220 グループ

5.3 内容

r_flash_api_rx_config.h で ROM プログラミングと ROM プログラミングのノンブロッキングモード(BGO)を有効にしてプログラムを実行すると、main 関数を実行したタイミングでプログラムが暴走します。

5.4 発生条件

r_flash_api_rx_config.h の以下の定義が 2 つとも有効な場合に問題となる現象が発生します。

定義 1： FLASH_API_RX_CFG_ENABLE_ROM_PROGRAMMING (ROM プログラミング)
定義 2： FLASH_API_RX_CFG_ROM_BGO (ROM プログラミングのノンブロッキングモード(BGO))

5.5 回避策

デモプログラム(flash_api_demo.c)の以下の赤字部分を変更してください。

修正前

```
(中略)
#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

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

修正後

```
(中略)
#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
(中略)
/* 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
***********************************************************************
*/
(中略)
```

5.6 恒久対策

次期バージョンで改修予定です。

以上

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Feb.01.22	-	新規発行

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/