【注意事項】

R20TS0772JJ0100
Rev.1.00
2021.12.01 号

RX ファミリ フラッシュ モジュール Firmware Integration Technology, RX Driver Package

概要

タイトルに記載している製品の使用上の注意事項を連絡します。

1. フラッシュタイプ 3、4 でノンブロッキングモード時に”R_FLASH_Erase”関数を実行した際のコールバック関数に関する注意事項
2. フラッシュタイプ 1 でノンブロッキングモード時に”R_FLASH_Erase”関数、”R_FLASH_BlankCheck”関数、”R_FLASH_Write”関数を実行した際の応答に関する注意事項
3. フラッシュタイプ 1 でノンブロッキングモード時に”R_FLASH_Write”関数を実行した際の書き込みに関する注意事項

1. フラッシュタイプ 3、4 でノンブロッキングモード時に”R_FLASH_Erase()”関数を実行した際のコールバック関数に関する注意事項

1.1 該当製品

(1) RX ファミリ フラッシュ モジュール Firmware Integration Technology

（以下、フラッシュモジュール）

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 フラッシュモジュール該当製品一覧

フラッシュモジュールのリビジョン	資料番号
Rev.2.10	R01AN2184JU0210
Rev.3.00	R01AN2184JU0300
Rev.3.10	R01AN2184JU0310
Rev.3.20	R01AN2184JU0320
Rev.3.30	R01AN2184JU0330
Rev.3.40	R01AN2184JU0340
Rev.3.41	R01AN2184JU0341
Rev.3.42	R01AN2184JU0342
Rev.3.50	R01AN2184JU0350
Rev.4.00	R01AN2184JJ0400
Rev.4.10	R01AN2184JJ0410
Rev.4.20	R01AN2184JJ0420
Rev.4.30	R01AN2184JJ0430
Rev.4.40	R01AN2184JJ0440
Rev.4.50	R01AN2184JJ0450
Rev.4.60	R01AN2184JJ0460
Rev.4.70	R01AN2184JJ0470
Rev.4.80	R01AN2184JJ0480

(2) RX Driver Package

(1)のフラッシュモジュールは RX Driver Package にも同梱されています。
該当する RX Driver Package の製品名、リビジョン、資料番号および同梱しているフラッシュモジュールのリビジョンは、以下のとおりです。

表 1.2 フラッシュモジュール同梱製品一覧

RX Driver Package の製品名	RX Driver Package のリビジョン	資料番号	同梱しているフラッシュモジュールのリビジョン
RX ファミリ RX Driver Package Ver.1.12	Rev.1.12	R01AN3651JJ0112	Rev.2.10
RX ファミリ RX Driver Package Ver.1.13	Rev.1.13	R01AN3859JJ0113	Rev.3.20
RX ファミリ RX Driver Package Ver.1.14	Rev.1.14	R01AN4191JJ0114	Rev.3.30
RX ファミリ RX Driver Package Ver.1.15	Rev.1.15	R01AN4372JJ0115	Rev.3.30
RX ファミリ RX Driver Package Ver.1.16	Rev.1.16	R01AN4471JJ0116	Rev.3.40
RX ファミリ RX Driver Package Ver.1.17	Rev.1.17	R01AN4572JJ0117	Rev.3.41
RX ファミリ RX Driver Package Ver.1.18	Rev.1.18	R01AN4659JJ0118	Rev.3.42
RX ファミリ RX Driver Package Ver.1.19	Rev.1.19	R01AN4677JJ0119	Rev.3.50
RX ファミリ RX Driver Package Ver.1.20	Rev.1.20	R01AN4794JJ0120	Rev.4.00
RX ファミリ RX Driver Package Ver.1.21	Rev.1.21	R01AN4843JJ0121	Rev.4.10
RX ファミリ RX Driver Package Ver.1.22	Rev.1.22	R01AN4873JJ0122	Rev.4.20
RX ファミリ RX Driver Package Ver.1.23	Rev.1.23	R01AN4976JJ0123	Rev.4.40
RX ファミリ RX Driver Package Ver.1.24	Rev.1.24	R01AN5267JJ0124	Rev.4.40
RX ファミリ RX Driver Package Ver.1.25	Rev.1.25	R01AN5371JJ0125	Rev.4.50
RX ファミリ RX Driver Package Ver.1.26	Rev.1.26	R01AN5401JJ0126	Rev.4.50
RX ファミリ RX Driver Package Ver.1.27	Rev.1.27	R01AN5600JJ0127	Rev.4.60
RX ファミリ RX Driver Package Ver.1.29	Rev.1.29	R01AN5826JJ0129	Rev.4.60
RX ファミリ RX Driver Package Ver.1.30	Rev.1.30	R01AN5882JJ0130	Rev.4.60
RX ファミリ RX Driver Package Ver.1.31	Rev.1.31	R01AN5975JJ0131	Rev.4.70
RX ファミリ RX Driver Package Ver.1.32	Rev.1.32	R01AN6013JJ0132	Rev.4.80

(3) フラッシュモジュールと組み合わせて使用される FIT モジュールやアプリケーションノートについて

(1)のフラッシュモジュール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ような FIT モジュールは該当します。

- ＲＸファミリ フラッシュメモリデータ管理モジュール Firmware Integration Technology(R20AN0507JJ)
  https://www.renesas.com/jp/ja/document/apn/rx-family-flash-memory-data-management-module-using-firmware-integration-technology
- RX ファミリ ファームウェアアップデートモジュール Firmware Integration Technology(R01AN5824JJ)
  https://www.renesas.com/jp/ja/document/apn/rx-family-firmware-update-module-using-firmware-integration-technology-application-notes
- RX ファミリ TSIP(Trusted Secure IP)モジュール Firmware Integration Technology(R20AN0548JJ)
  https://www.renesas.com/jp/ja/document/apn/rx-family-tsip-trusted-secure-ip-module-firmware-integration-technology-binary-version

1.2 該当デバイス

RX64M、RX651、RX65N、RX66N、RX66T、および RX671 グループ

RX71M、RX72M、RX72N、および RX72T グループ

1.3 内容

フラッシュモジュールの”R_FLASH_Erase”関数を実行しフラッシュメモリに関するエラーが発生した際、ユーザアプリケーションに対してコールバック関数によるエラー通知が”R_FLASH_Erase”関数の引数として指定したブロック数分行われる可能性があります。

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：フラッシュモジュールをノンブロッキングモードの設定で使用
条件 2：メインルーチン内の特定区間(下図(1)の区間 A)の処理時間がイレーズ時間より長い
条件 3：フラッシュメモリに関するエラーが発生

以下では条件 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_flash_fcu.c”内でイレーズコマンドの最終コマンド書き込み(FLASH_FACI_CMD_FINAL)が実行されてから、カレントオペレーション変数(g_current_parameters.current_operation)を判定するまでの区間の処理時間がイレーズ時間よりも長く、かつフラッシュメモリに関するエラーが発生した場合、メインルーチン内でカレントオペレーション変数が判定される前に FIFERR 割り込み要求が発生します。
エラー発生時には FIFERR 割り込み内でカレントオペレーション変数を変更するため FIFERR 割り込みから戻った後、カレントオペレーション変数の判定が不一致となりブレークしないためイレーズ処理が継続されます。

```
flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
(中略)                                  イレーズコマンドの最終コマンド書き込みを実行
        *g_pfcu_cmd_area = (uint8_t) FLASH_FACI_CMD_BLOCK_ERASE;
        *g_pfcu_cmd_area = (uint8_t) FLASH_FACI_CMD_FINAL;
区間 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))
                                                  カレントオペレーション変数の判定
        {
            break;
                                  ここでブレークされません
        }
(中略)
```

上記のコードは以下のようにアセンブラ展開されます。
イレーズコマンドの最終コマンド書き込みは以下の②において実行されます。
以下の②の 1 サイクル命令が実行されるタイミングで割り込み要求が発生し、かつその割り込み処理時間がイレーズ時間よりも長く、かつフラッシュメモリに関するエラーが発生した場合、メインルーチン内でカレントオペレーション変数が判定される前に FIFERR 割り込み要求が発生します。

```
(*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 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

```
flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
（中略）
       *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;
       }
（中略）
```

修正後

```
flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
（中略）
       *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;
       }
（中略）
```

1.6 恒久対策

次期バージョンで改修予定です。

2. フラッシュタイプ 1 でノンブロッキングモード時に”R_FLASH_Erase”関数、”R_FLASH_BlankCheck”関数、”R_FLASH_Write”関数を実行した際の応答に関する注意事項

2.1 該当製品

(1) RX ファミリ フラッシュ モジュール Firmware Integration Technology

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 フラッシュモジュール該当製品一覧

フラッシュモジュールのリビジョン	資料番号
Rev.4.80	R01AN2184JJ0480

(2) RX Driver Package

(1)のフラッシュモジュールは RX Driver Package にも同梱されています。
該当する RX Driver Package の製品名、リビジョン、資料番号および同梱しているフラッシュモジュールのリビジョンは、以下のとおりです。

表 2.2 フラッシュモジュール同梱製品一覧

RX Driver Package の製品名	RX Driver Package のリビジョン	資料番号	同梱しているフラッシュモジュールのリビジョン
RX ファミリ RX Driver Package Ver.1.32	Rev.1.32	R01AN6013JJ0132	Rev.4.80

(3) フラッシュモジュールと組み合わせて使用される FIT モジュールやアプリケーションノートについて

(1)のフラッシュモジュール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ような FIT モジュールは該当します。

- R X ファミリ フラッシュメモリデータ管理モジュール Firmware Integration Technology(R20AN0507JJ)
  https://www.renesas.com/jp/ja/document/apn/rx-family-flash-memory-data-management-module-using-firmware-integration-technology
- RX ファミリ ファームウェアアップデートモジュール Firmware Integration Technology(R01AN5824JJ)
  https://www.renesas.com/jp/ja/document/apn/rx-family-firmware-update-module-using-firmware-integration-technology-application-notes
- RX ファミリ TSIP(Trusted Secure IP)モジュール Firmware Integration Technology(R20AN0548JJ)
  https://www.renesas.com/jp/ja/document/apn/rx-family-tsip-trusted-secure-ip-module-firmware-integration-technology-binary-version

2.2 該当デバイス

RX110、RX111、RX113、RX130、RX13T、および RX140 グループ

RX230、RX231、RX23E-A、RX23T、RX23W、RX24T および RX24U グループ

2.3 内容

フラッシュモジュールの”R_FLASH_Erase”関数、”R_FLASH_BlankCheck”関数、”R_FLASH_Write”関数を実行した際、ユーザアプリケーションへの応答に時間がかかる可能性があります。

2.4 発生条件

以下の条件 1 と条件 2～4 のいずれかが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：フラッシュモジュールをノンブロッキングモードの設定で使用
条件 2：メインルーチン内の特定区間(下図(1)の区間 A、B)の処理時間がイレーズ時間より長い
条件 3：メインルーチン内の特定区間(下図(1)の区間 A、B)の処理時間がブランクチェック時間より長い
条件 4：メインルーチン内の特定区間(下図(1)の区間 A、B)の処理時間がプログラム時間より長い

以下では条件 2～4 において区間 A、B 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、条件 2 において”r_flash_nofcu.c”内で”flash_df_erase”関数または”flash_cf_erase”関数が実行されてから、カレントオペレーション変数(g_current_parameters.current_operation)を判定するまでの区間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カレントオペレーション変数が判定される前に FRDYI 割り込み要求が発生します。
FRDYI 割り込み内でカレントオペレーション変数を変更し FRDY を 0 にするため、FRDYI 割り込みから戻った後、カレントオペレーション変数の判定が不一致となりリターンせず”flash_wait_frdy”関数が実行されます。このとき、FRDY が 0 のため”flash_wait_frdy”関数内でタイムアウト処理が実行され、応答に時間がかかります。

```
flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
(中略)
    if (FLASH.FENTRYR.WORD == 0x0080)
    {
#ifndef FLASH_NO_DATA_FLASH
        flash_df_erase(block_address, num_blocks);
#endif
    }
(中略)
    else if (FLASH.FENTRYR.WORD == 0x0001)
    {
        flash_cf_erase(block_address, num_blocks);
    }
(中略)
    /* 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;
    }
(中略)
    err = flash_wait_frdy();
    if (err != FLASH_SUCCESS)
    {
        return err;
    }
(中略)
```

区間 A
区間 B
カレントオペレーション変数の判定
ここでリターンされません
ここでタイムアウトが発生します

以下、条件 3 において”r_flash_nofcu.c”内で”flash_df_blankcheck”関数または”flash_cf_blankcheck”関数が実行されてから、カレントオペレーション変数を判定するまでの区間の処理時間がブランクチェック時間よりも長い場合、メインルーチン内でカレントオペレーション変数が判定される前に FRDYI 割り込み要求が発生します。
FRDYI 割り込み内でカレントオペレーション変数を変更し FRDY を 0 にするため、FRDYI 割り込みから戻った後、カレントオペレーション変数の判定が不一致となりリターンせず”flash_wait_frdy”関数が実行されます。このとき、FRDY が 0 のため”flash_wait_frdy”関数内でタイムアウト処理が実行され、応答に時間がかかります。

```
flash_err_t flash_blankcheck(const uint32_t start_address, const uint32_t
num_bytes, flash_res_t *result)
{
(中略)
    if (FLASH.FENTRYR.WORD == 0x0080)
    {
#ifndef 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)                          区間 A
    {
        flash_cf_blankcheck(start_address, (start_address + num_bytes) - 1);
    }
#endif
    else                                                            区間 B
    {
        /* 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))                  カレントオペレーション変数の判定
    {
        return err;                 ここでリターンされません
    }

    /* In blocking mode, wait for FRDY or timeout. Return if error. */
    err = flash_wait_frdy();        ここでタイムアウトが発生します
    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 */
    }
(中略)
```

以下、条件 4 において”r_flash_nofcu.c”内で”flash_df_write”関数または”flash_cf_write”関数が実行されてから、カレントオペレーション変数を判定するまでの区間の処理時間がプログラム時間よりも長い場合、メインルーチン内でカレントオペレーション変数が判定される前に FRDYI 割り込み要求が発生します。
FRDYI 割り込み内でカレントオペレーション変数を変更し FRDY を 0 にするため、FRDYI 割り込みから戻った後、カレントオペレーション変数の判定が不一致となりブレークせず”flash_wait_frdy”関数が実行されます。このとき、FRDY が 0 のため”flash_wait_frdy”関数内でタイムアウト処理が実行され、応答に時間がかかります。

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
(中略)
       /* Conversion to the P/E address from the read address */
       if (FLASH.FENTRYR.WORD == 0x0080)
       {
#ifndef 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)
       {                                                          区間 A
           flash_cf_write(g_current_parameters.src_addr,
g_current_parameters.dest_addr);
       }
#endif

       g_current_parameters.total_count--;                        区間 B

       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))
       {                                          カレントオペレーション変数の判定
           break;          ここでブレークされません
       }

       /* In blocking mode, wait for FRDY or timeout. Return if error. */
       err = flash_wait_frdy();                   ここでタイムアウトが発生します
       if (err != FLASH_SUCCESS)
       {
           break;
       }
(中略)
```

2.5 回避策

”r_flash_nofcu.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

```
flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
(中略)
    /* 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;
    }
(中略)
```

```
flash_err_t flash_blankcheck(const uint32_t start_address, const uint32_t
num_bytes, flash_res_t *result)
{
(中略)
    /* 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;
    }
(中略)
```

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
(中略)
        /* 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;
        }
(中略)
```

修正後

```
flash_err_t flash_erase(uint32_t block_address, uint32_t num_blocks)
{
(中略)
    /* 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;
    }
(中略)
```

```
flash_err_t flash_blankcheck(const uint32_t start_address, const uint32_t
num_bytes, flash_res_t *result)
{
(中略)
    /* 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;
    }
(中略)
```

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
(中略)
        /* 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;
        }
(中略)
```

2.6 恒久対策

次期バージョンで改修予定です。

3. フラッシュタイプ 1 のノンブロッキングモード時に”R_FLASH_Write”関数を実行した際の書き込みに関する注意事項

3.1 該当製品

(1) RX ファミリ フラッシュ モジュール Firmware Integration Technology

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 フラッシュモジュール該当製品一覧

フラッシュモジュールのリビジョン	資料番号
Rev.4.80	R01AN2184JJ0480

(2) RX Driver Package

(1)のフラッシュモジュールは RX Driver Package にも同梱されています。
該当する RX Driver Package の製品名、リビジョン、資料番号および同梱しているフラッシュモジュールのリビジョンは、以下のとおりです。

表 3.2 フラッシュモジュール同梱製品一覧

RX Driver Package の製品名	RX Driver Package のリビジョン	資料番号	同梱しているフラッシュモジュールのリビジョン
RX ファミリ RX Driver Package Ver.1.32	Rev.1.32	R01AN6013JJ0132	Rev.4.80

(3) フラッシュモジュールと組み合わせて使用される FIT モジュールやアプリケーションノートについて

(1)のフラッシュモジュール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ような FIT モジュールは該当します。

- ＲＸファミリ フラッシュメモリデータ管理モジュール Firmware Integration Technology(R20AN0507JJ)
  https://www.renesas.com/jp/ja/document/apn/rx-family-flash-memory-data-management-module-using-firmware-integration-technology
- RX ファミリ ファームウェアアップデートモジュール Firmware Integration Technology(R01AN5824JJ)
  https://www.renesas.com/jp/ja/document/apn/rx-family-firmware-update-module-using-firmware-integration-technology-application-notes
- RX ファミリ TSIP(Trusted Secure IP)モジュール Firmware Integration Technology(R20AN0548JJ)
  https://www.renesas.com/jp/ja/document/apn/rx-family-tsip-trusted-secure-ip-module-firmware-integration-technology-binary-version

3.2 該当デバイス

RX110、RX111、RX113、RX130、RX13T、および RX140 グループ

RX230、RX231、RX23E-A、RX23T、RX23W、RX24T および RX24U グループ

3.3 内容

フラッシュモジュールの”R_FLASH_Write”関数を実行した際、最小プログラムサイズ 1 回分多く書き込まれる可能性があります。

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

以下では条件 2 において区間 A、B 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_flash_nofcu.c”内で”flash_df_write”関数または”flash_cf_write”関数が実行されてから、トータルカウント変数(g_current_parameters.total_count)をデクリメントするまでの区間の処理時間がプログラム時間よりも長い場合、メインルーチン内でトータルカウント変数がデクリメントされる前に FRDYI 割り込み要求が発生します。

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
（中略）
    while(g_current_parameters.total_count > 0)
    {
        /* Conversion to the P/E address from the read address */
        if (FLASH.FENTRYR.WORD == 0x0080)
        {
#ifndef 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;
        }
（中略）
```

関数の最後で FCR レジスタに書き込みを実行

関数の最後で FCR レジスタに書き込みを実行

区間 A

区間 B

トータルカウント変数をデクリメント

(2) FRDYI 割り込み内の処理と今回の問題

”r_flash_nofcu.c”内の"flash_write”関数においてトータルカウント変数がデクリメントされないまま、FRDYI 割り込み要求が発生することにより、以下の FRDYI 割り込み処理内の if 文の条件が成立するため、下図の処理 C、または D のプログラムが実行され、最小プログラムサイズ 1 回分多く書き込まれます。

```
R_BSP_ATTRIB_STATIC_INTERRUPT void Excep_FCU_FIFERR(void)
{
（中略）
    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;
            }
（中略）
    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;
            }
（中略）
```

区間 B で条件 2 が成立し、トータルカウント変数がデクリメントされない場合

処理 C

区間 A で条件 2 が成立し、トータルカウント変数がデクリメントされない場合

処理 D

3.5 回避策

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

修正前

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
(中略)
    while(g_current_parameters.total_count > 0)
    {
        /* Conversion to the P/E address from the read address */
        if (FLASH.FENTRYR.WORD == 0x0080)
        {
#ifndef 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;
        }
(中略)
```

修正後

```
flash_err_t flash_write(const uint32_t src_address, const uint32_t
dest_address, uint32_t num)
{
(中略)
    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)
        {
#ifndef 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;
        }
(中略)
```

3.6 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	Dec.01.21	-	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/

TS Colophon 4.2