Renesas Synergy™
内蔵フラッシュメモリ内のアプリケーション更新の事例

R30AN0304JJ0104
Rev. 1.04
2018.08.28

要旨

本アプリケーションノートでは、Renesas Synergy™ マイコンに搭載されている Memory Mirror Function (MMF)と Startup Area Select の機能を活用した内蔵フラッシュ内のアプリケーション更新の事例を説明します。この事例では、内蔵フラッシュ内にアプリケーションを保存する領域を２つ設定し、現在起動するアプリケーションが保存されている領域とは異なるもう一方の領域に新しいアプリケーションを書き込みます。次回起動時には、その新しい方を動作させることでアプリケーションの更新を行います。

動作確認デバイス

SK-S7G2, PK-S5D9, DK-S3A7

使用機能

以下の SSP モジュール、および、MCU 機能を使用しています。

種別	モジュール名／機能名
MCU	Memory Mirror Function (MMF)
	Startup Area Select
	Battery Backup Function
Framework	sf_el_ux_comms
	sf_message
HAL Driver	r_flash_hp(SK-S7G2, PK-S5D9) r_flash_lp(DK-S3A7)
	r_gpt

8 月公開のテクニカルアップデートに対する対応について

2018 年 8 月に発行された S5 シリーズのテクニカルアップデート対し、本アプリケーションノートでは以下の対策を実施しました。

本アプリケーションノートを S5 シリーズに適応される場合は、下記の制限事項に留意してください。S7,S3 シリーズには、この制限事項はありません。

内容の詳細はテクニカルアップデートをご参照ください。

Document No. TN-SY*-A033A/E Rev. 1.00

[制限事項] r_flash_hp ドライバ

RAM 上に展開・実行される関数の内、option-setting memory を書き換える処理を下記に示す RAM アドレス(SRAMHS)に配置しないでください。

0x1FFF0000 - 0x1FFF07FF

0x1FFF2000 – 0x1FFE23FF

[対策]

PK-S5D9 向けのサンプルソフト（アプリケーションプログラムとフラッシュ書き換えプログラム）において、option-setting memory を書き換える処理が上記アドレスに配置されることがないように、リンカスクリプトを設定しています。

本サンプルでは SRAMHS の領域全体(1FFE 0000h – 1FFF FFFFh)を使用しない様にリンカスクリプトを変更していますが、option-setting memory を書き換える処理において、問題となるアドレスへアクセスされることがないようにお客様側でご対応頂ければ SRAMHS はこれまで通り使用可能です。この場合、synergy/ssp/src/driver/r_flash_hp/hw/target/hw_flash_hp.c にある HW_FLASH_HP_configurationSet 関数内の処理について考慮してください。同関数については PLACE_IN_RAM_SECTION が設定されているため、.code_in_ram セクションが問題のアドレスに重ならいないようにリンカスクリプトを設定することで、同関数の実行アドレスについて対策することが可能です。また、同関数内のスタックを含むデータアクセスに関するアドレスについても考慮が必要になりますので、ご注意ください。尚、同関数は r_flash_hp の以下の API にてコールされます。

- R_FLASH_HP_AccessWindowSet
- R_FLASH_HP_AccessWindowClear
- R_FLASH_HP_StartUpAreaSelect

アプリケーションプログラムには、アプリケーション上でプログラムのスワップを可能とする機能が動作確認用として実装されています。このスワップ時には option-setting memory を書き換えるため、アプリケーションプログラムにも本対策が適応されています。このスワップ機能の処理を削除することで、アプリケーションプログラムにおいてもすべての SRAMHS 領域が使用可能となります。

本アプリケーションノートのサンプルソフトで SRAMHS をご使用になりたい場合、テクニカルアップデートの内容をよくご理解の上、制限事項に該当しないようにお客様側で対策をお願いいたします。

目次

1. はじめに

1.1 概要

近年、組み込み製品においては、不具合修正や機能追加といった理由で、その出荷時に書き込まれているアプリケーションを書き換える機能が必要とされることが多くなっています。Renesas Synergy™ マイコンは、この機能を実現するのに有用な機能として、Memory Mirror Function (MMF)と Startup Area Select の２つを備えています。Startup Area Select 機能を利用することで、起動時に読み出されるコードフラッシュのブロックを２つのブロックのどちらかに切り替えることができます。また、MMF を利用することで、物理的に異なるアドレスの内蔵フラッシュメモリに対して、同一のアドレスでアクセスすることができます。この２つの機能を組み合わせることで、アプリケーションの実行アドレスは一意でありながら、それを保存する内蔵フラッシュメモリの領域を２つ用意することができ、また、どのアプリケーションで起動するかを選択することができます。

1.2 評価環境

本アプリケーションノートでは、以下の環境で動作確認をしています。

SSP (Synergy Software Package)	v1.3.3
e² studio	v5.4.0.023
IAR EWARM for Synergy	V7.7.1.3.13746
RFP (Renesas Flash Programmer)	v3.03.00
Tera Term	v4.9.5
Board	SK-S7G2 PK-S5D9 DK-S3A7

1.3 本アプリケーションノートの動作確認

本アプリケーションノートには、SK-S7G2, PK-S5D9, DK-S3A7 ボードで動作確認するための環境（work_demo,zip）が付属しています。それぞれ、e2studio(GNU Compiler)環境でビルドされています。本アプリケーションノートに関する動作を確認する場合は、「10 内蔵フラッシュ書き換えの動作確認」を参照してください。

1.4 留意事項

本アプリケーションノートでは BTFLG を操作します。デバッグ時の予期せぬ事態で Reserved 領域を書き換えることにより、ソフトウェアやデバッガが起動しない事態が発生する事があります。特に FSPR ビットに一度 0 を書きこんでしまった際には以降の BTFLG 変更を受け付けなくなります(Access Window Setting Control Register(AWSC)内の FSPR ビットは再度 1 に戻すことはできません。)。十分に留意して開発を行ってください。

動作に問題が発生した場合、「10.1 出荷イメージデータの書き込み」を試してください。

S3A7 の場合、フラッシュドライバ(r_flash_lp)で AWSC 内の BTFLG を書き換えると、同時に AWSC の Reserved bit に 0 を書き込みます。AWSC 内の Reserved bit は 0 にすると 1 に戻すことが出来ない仕様となっている為、BTFLG 書き換え後に Renesas Flash Programmer(RFP)から BTFLG を初期化しようとしてもベリファイに失敗します。S3A7 において、BTFLG の初期化をしたい場合は、RFP の設定でベリファイを無効にして書き込みを実施してください。尚、この問題は Issue ID 11593 として管理されており、SSP v1.5.0 で修正される予定です。それ以降は上記の対策は不要になる予定ですので、本問題の修正につきましては、SSP のリリースノートでご確認ください。

1.5 用語

本アプリケーションノートでは以下のように用語を用います。

BTFLG	Startup Area Select 機能を実現するための MCU 内の機能。 0000 0000h～0000 1FFFh をブロック 0、 0000 2000h～0000 3FFFh をブロック 1 とすると、BTFLG=1（初期値）の場合は、MCU から 0000 0000h～0000 1FFFh へのアクセスはブロック 0 へのアクセスとなります。一方、BTFLG=0 の場合、MCU から 0000 0000h～0000 1FFFh へのアクセスはブロック 1 へのアクセスとなります。
アプリケーション保存領域	フラッシュローダーを除く領域を２分割したアプリケーションを保存するための内蔵フラッシュの領域。
アプリケーション実行領域	アプリケーション実行時に MMF を通してアクセスする領域。起動直後に BTFLG の設定に対応する MMF が設定された後は、アプリケーションはこの領域で実行されます。
更新モード	アプリケーション実行中にホスト PC からコマンドを受け、更新処理専用のプログラムで再起動するときのモード。
フラッシュローダー	更新モードで起動時に動作する更新処理専用のプログラム。ホスト PC と通信を行い、更新データの受信、コマンドに対する応答の送信を行い、受信した更新データを内蔵フラッシュに書き込みます。出荷時に書き込まれ、以後は書き換えられません。
更新モードフラグ	アプリケーション実行中に、ホスト PC から更新モードによる再起動のコマンドを受信した場合に VBATT バックアップレジスタ(VBTBKRn)内に設定されます。フラッシュローダー起動直後にクリアされます。

1.6 参考文献

[1] Renesas Synergy™ Software Package v1.3.0 User's Manual (R01US0315EU0100 Rev. 01.00, 3 Sep 2017)

[2] S7G2 User's Manual: Microcontrollers (R01UM0001EU0120 Rev.1.20, Aug 26, 2016)

[3] Renesas Synergy™ Starter Kit SK-S7G2 User's Manual (R12UM0004EU0100 Rev. 1.00, Oct 8, 2015)

[4] S5D9 User's Manual: Microcontrollers (R01UM0004EU0100 Rev. 1.00, Nov 3, 2016)

[5] Renesas Synergy™ Promotion Kit PK-S5D9 User's Manual (R12UM0009EU0101 Rev. 1.01, Apr 11, 2017)

[6] S3A7 User's Manual: Microcontrollers (R01UM0002EU0130 Rev.1.30, Feb 7, 2017)

[7] Renesas Synergy™ Development Kit DK-S3A7 User's Manual (R12UM0003EU0100 Rev. 1.00 Oct 5, 2015)

[8] Renesas Flash Programmer V3.03 フラッシュ書き込みソフトウェア ユーザーズマニュアル

[9] Renesas Synergy™ Platform System Specifications for Standard Boot Firmware (R01AN3280EU0110 Rev.1.10, Oct 13, 2017)

2. 全体概要

2.1 内蔵フラッシュ書き換え機能概要

本アプリケーションノートで実装する内蔵フラッシュ書き換えの機能概要を以下に示します。

- 内蔵フラッシュ内に２つのアプリケーション保存領域を持ち、２つのアプリケーションを同時に保存可能です。
- BTFLG を切り替えることで、保存されている２つのアプリケーションのどちらで起動するかを選択することが可能です。
- デバイスはホスト PC と USB CDC ACM 通信を行うことで、コマンド受信やその応答の送信を行います。
- ホスト PC から更新要求コマンドを受信すると、デバイスは更新モードで再起動し、フラッシュローダーを実行します。
- フラッシュローダーは、ホスト PC から送信される更新データ（モトローラ S レコード形式）を受信します。
- フラッシュローダーは、受信したモトローラ S レコードのチェックサムを検証します。
  フラッシュローダーは、現在の BTFLG の設定を確認し、その設定に対応するアプリケーション保存領域とは別のアプリケーション保存領域に対して、受信したデータを書き込みます。
- フラッシュローダーは、内蔵フラッシュへの書き込み後ベリファイを行い、エラーの場合はホスト PC にエラーを通知します。
- フラッシュローダーは、受信した全データの書き込みが正常に完了した場合、BTFLG を切り替えます。
- フラッシュローダー動作時に通信断絶や電源断の発生した場合でも、次回起動時には直前に動作していたアプリケーションで起動します。

2.2 ハードウェア構成

本書に付属のサンプルアプリケーション実行に必要なハードウェア構成を表 1に示します。

表 1 ハードウェア構成

デバイス	用途
SK-S7G2 ボード, PK-S5D9 ボード, DK-S3A7 ボード	―
USB ケーブル（A）オス - （micro-B）オス	電源供給、デバッガ接続
USB ケーブル（A）オス - （micro-B）オス	ボードーホスト PC 間データ通信用
ホスト PC（Windows 7, Windows 8, Windows10）	フラッシュ書き換えアプリケーション実行

2.3 メモリマッピング

本アプリケーションでは、表 2、表 3、表 4 のようにメモリ領域を設定します。

表 2 メモリマッピング(S7G2)

メモリ領域		用途
開始アドレス	終了アドレス	
0000 0000h	0000 1FFFh	ブートブロック 0
0000 2000h	0000 3FFFh	ブートブロック 1
0000 4000h	0001 FFFFh	フラッシュローダー領域
0002 0000h	001F FFFFh	アプリケーション保存領域 0
0020 0000h	0021 FFFFh	(未使用)
0022 0000h	003F FFFFh	アプリケーション保存領域 1
0200 0000h	0201 FFFFh	(未使用)
0202 0000h	021F FFFFh	アプリケーション実行領域
0220 0000h	023F FFFFh	(未使用)

※: グレー部分は MMF 領域

表 3 メモリマッピング(S5D9)

メモリ領域		用途
開始アドレス	終了アドレス	
0000 0000h	0000 1FFFh	ブートブロック 0
0000 2000h	0000 3FFFh	ブートブロック 1
0000 4000h	0001 FFFFh	フラッシュローダー領域
0002 0000h	000F FFFFh	アプリケーション保存領域 0
0010 0000h	0011 FFFFh	(未使用)
0012 0000h	001F FFFFh	アプリケーション保存領域 1
0200 0000h	0201 FFFFh	(未使用)
0202 0000h	020F FFFFh	アプリケーション実行領域
0210 0000h	021F FFFFh	(未使用)

※: グレー部分は MMF 領域

表 4 メモリマッピング(S3A7)

メモリ領域		用途
開始アドレス	終了アドレス	
0000 0000h	0000 1FFFh	ブートブロック 0
0000 2000h	0000 3FFFh	ブートブロック 1
0000 4000h	0001 FFFFh	フラッシュローダー領域
0002 0000h	0007 FFFFh	アプリケーション保存領域 0
0008 0000h	0009 FFFFh	(未使用)
000A 0000h	000F FFFFh	アプリケーション保存領域 1
0200 0000h	0201 FFFFh	(未使用)
0202 0000h	0207 FFFFh	アプリケーション実行領域
0208 0000h	020F FFFFh	(未使用)

※: グレー部分は MMF 領域

3. 動作概要

3.1 システム動作概要

図 1に本アプリケーションノートでの内蔵フラッシュ書き換え動作の概要を示します。また、同図内の詳細について、表 5に示します。

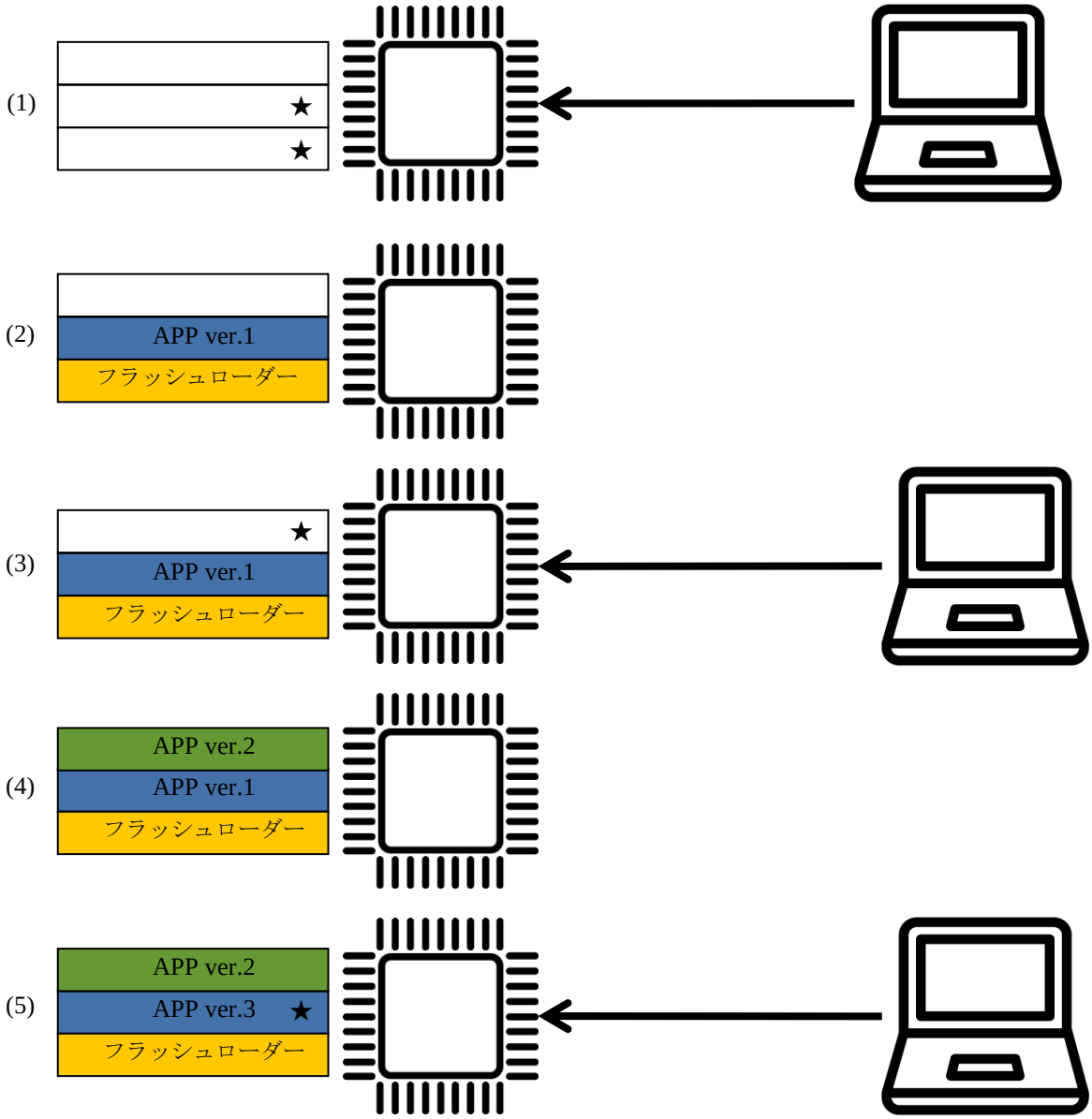


図 1 内蔵フラッシュ書き換え

表 5　内蔵フラッシュ書き換え動作

(1)	初期出荷プログラムの書き込み	初期バージョンである APP ver.1 と更新用プログラムをホスト PC から内蔵フラッシュ（★で示す）へ書き込みます。
(2)	初期出荷プログラムの書き込み完了	APP ver.1 と更新用プログラムが書き込まれ、アプリケーションとして APP ver.1 が起動します。
(3)	APP ver.2 の書き込み開始	APP ver.2 を APP ver.1 が保存されている方とは別の領域（★で示す）に書き込みます。
(4)	APP ver.2 の書き込み完了	APP ver.1 と APP ver.2 が書き込まれた状態になります。起動するアプリケーションを APP ver.2 に切り替えます。
(5)	APP ver.3 の書き込み開始／完了	APP ver.3 を APP ver.1 が保存されている領域（★で示す）に書き込みます。結果、APP ver2 と APP ver.3 が書き込まれた状態になります。起動するアプリケーションを APP ver.3 に切り替えます。

3.2　Startup Area Select と MMF を利用したアプリケーション更新の動作概要

本アプリケーションノートでは、内蔵フラッシュ内に設定した 2 つのアプリケーション保存領域に対して、現在起動するアプリケーションの保存領域には影響を与えずに、もう一方のアプリケーション保存領域に対して、更新データを書き込みます。この更新が正常に完了した後は、その更新された方のアプリケーション保存領域のアプリケーションで起動します。

これを実現するため、Renesas Synergy™ マイコンに搭載されている Startup Area Select と MMF を活用しています。ここでは、本アプリケーションノートにおける、それらの機能の利用方法について説明します。

図 2 、図 3、図 4は本アプリケーションノートの動作概要を示しており、図 2の(a)~(e)は S7G2、図 3(a)~(e)は S5D9、 図 4(a)~(e)は S3A7 における動作概要です。

図 2(a)は、ブートブロック 0 とアプリケーション保存領域 0 がアクティブな状況において、それとは異なるアプリケーションをブートブロック 1 とアプリケーション保存領域 1 に書き込むことを示しています。図 2(b)は、それが完了した状態を示しています。ここで、BTFLG を 1 から 0 に変更すると、図 2(c)に示すように、ブートブロック 0 とブートブロック 1 のアドレスが入れ替わります。この状態で再起動すると、ブートブロック 1 から起動します。しかし、アプリケーション 1 はアプリケーション保存領域 1 に保存されており、図 2(a)のアプリケーション 0 の場合とブートブロックとアプリケーション保存領域のアドレス配置が異なります。

図 2(c)において MMF を有効にし、アプリケーション保存領域 0 とアプリケーション保存領域 1 のアドレス差分 0020 0000h を MMSFR に設定すると、図 2(d)になります。ここでは、0000 0000h～001F FFFFh は通常のアドレスで内蔵フラッシュにアクセスし、0020 0000h 以降は図中のアドレスにおいて **2** で示したように MMF を通してアクセスします。図 2(d)において MMSFR を 0000 0000h に設定すると、図 2(e)になります。図 2(d)で★で示したブートブロック 1、および、アプリケーション 1 と、図 2(e)で★で示したブートブロック 0、および、アプリケーション 0 は、アドレス配置が同じになります。

よって、アプリケーション 0 とアプリケーション 1 は、図 2(e)と図 2(d)で示す Startup Area Select と MMF の設定により、同じアドレス配置で動作させることができます。

S5D9, S3A7 においても動作概要は上記と同様ですが、コードフラッシュの容量が異なる為アプリケーション 0 保存領域、アプリケーション 1 保存領域、MMSFR の設定値が異なります。図 2の(a) ~(e)は、図 3、図 4 の(a)~(e)のそれぞれに対応します。

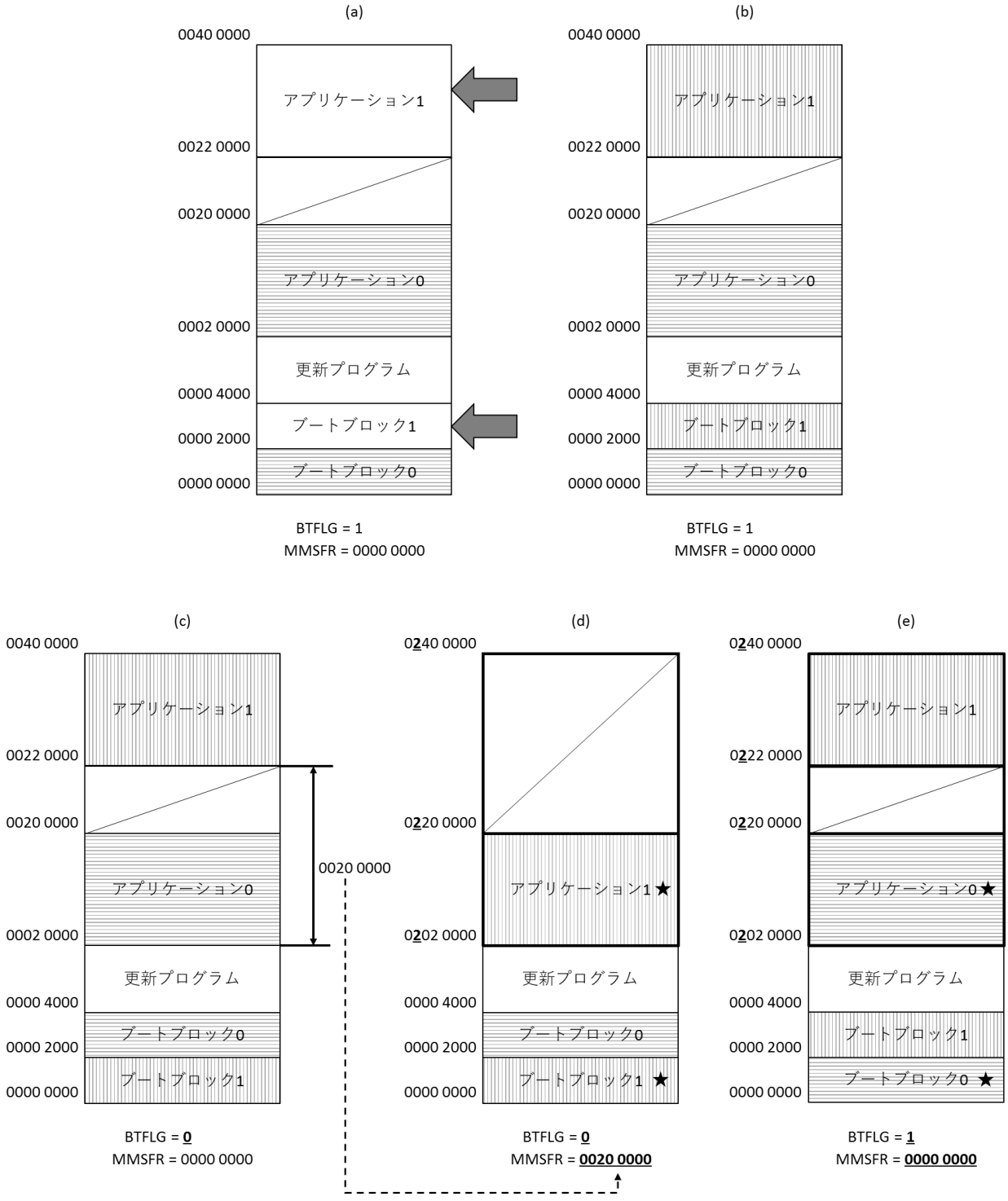


図 2 Startup Area Select と MMF を利用したアプリケーションの共存(S7G2 の例)

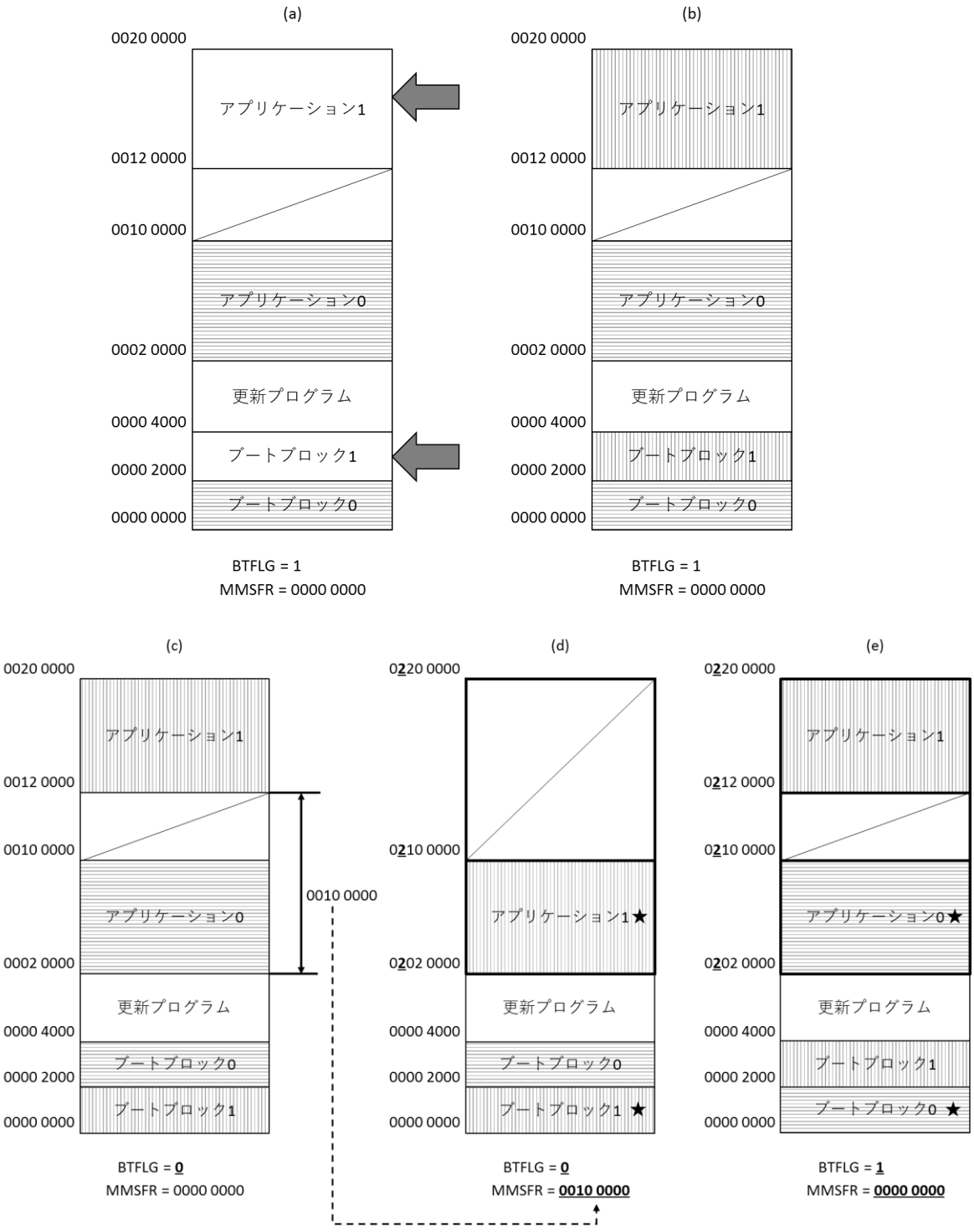
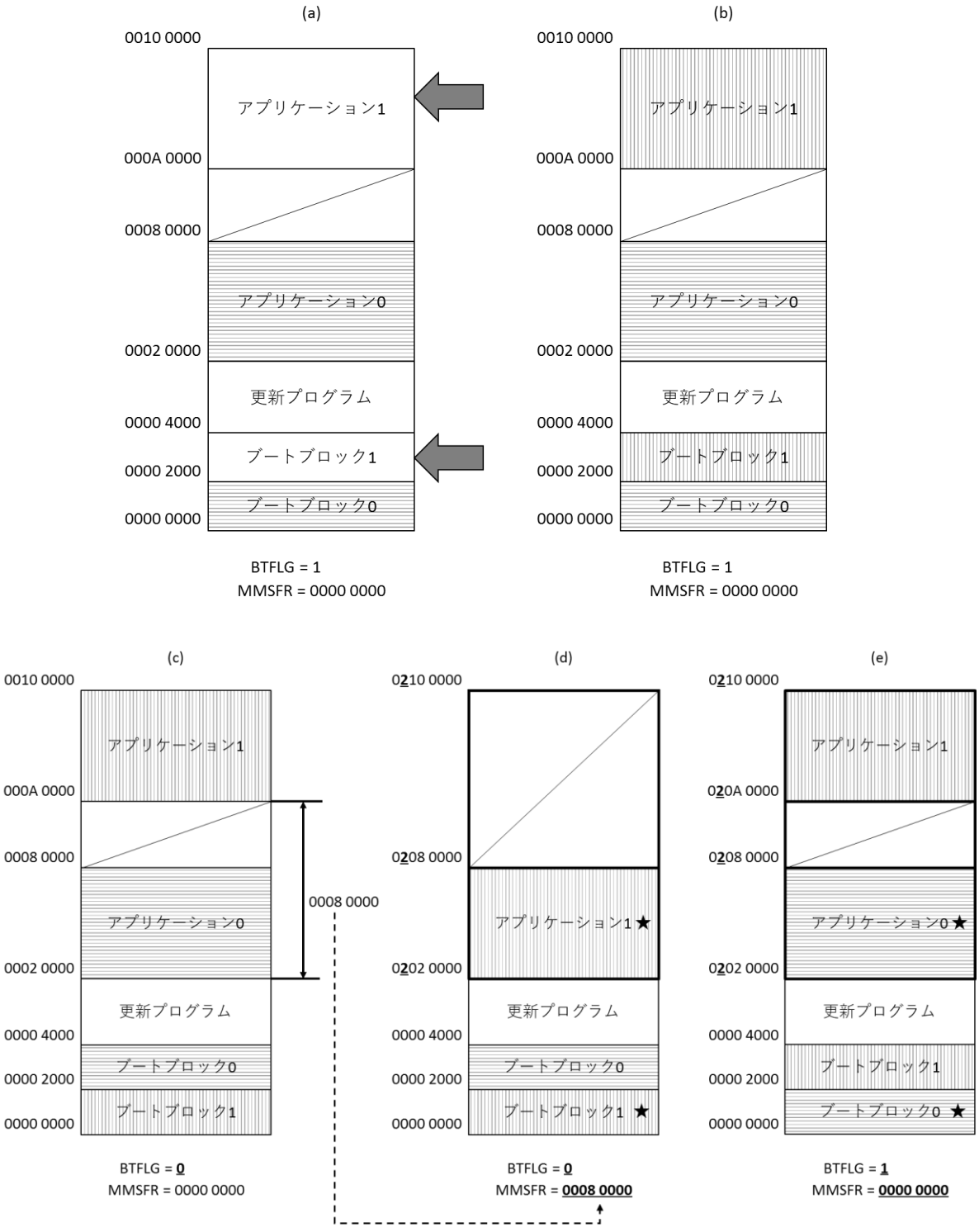


図 3 Startup Area Select と MMF を利用したアプリケーションの共存(S5D9 の例)



3 ^ 図 4 Startup Area Select と MMF を利用したアプリケーションの共存(S3A7 の例)

起動処理のシーケンス

S7 において、図 2で示したアプリケーションの共存方法と、表 2に示すアドレスにフラッシュローダーを配置した場合について、その起動時のシーケンスを図 5に示します。また、各シーケンスの動作概要を表 6に示します。図 5中の(1)～(11)は表 6のそれぞれに対応します。

S5D9 と S3A7 における起動処理のシーケンスは S7 の場合と同一ですが、表 6中の(4)の内容が異なります。

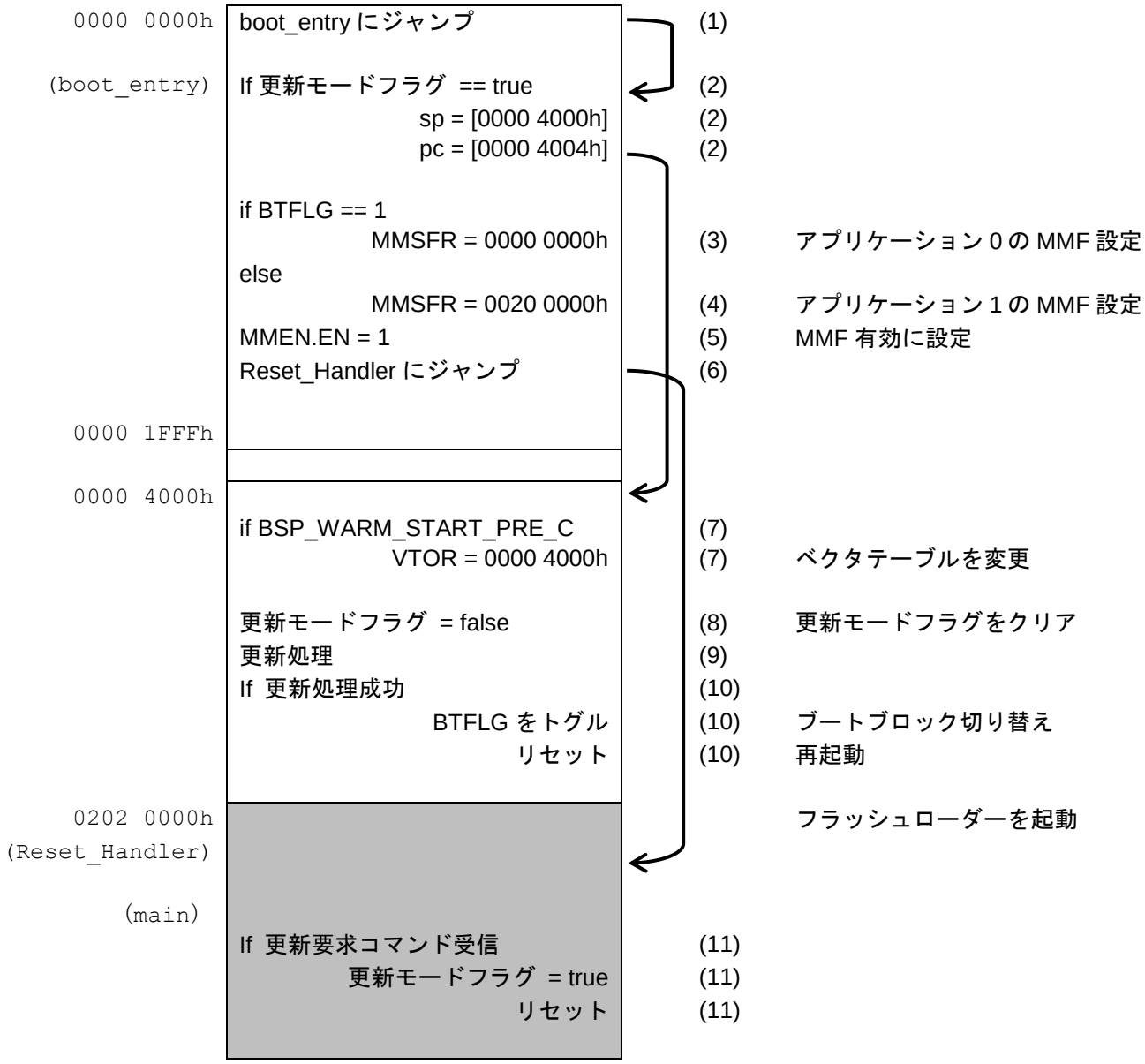


図 5 起動シーケンス(S7G2 の例)

表 6　起動シーケンスにおける動作概要

リセットベクタ	(1)	0000 0004h に保存されている boot_entry のアドレスにジャンプします。
boot_entry	(2)	更新モードフラグが true の場合、sp レジスタと pc レジスタをそれぞれ 0000 4000h に保存されている値と 0000 4004h に保存されている値に変更します。これにより、0000 4000h 以降に保存されているフラッシュローダーを起動します。
	(3)	BTFLG = 1 の場合、MMF 領域の 0200 0000h から 0000 0000h にアクセスできるように設定します。
	(4)	BTFLG = 0 の場合、MMF 領域の 0200 0000h から 0020 0000h にアクセスできるように設定します。
	(5)	MMF を有効にします。
	(6)	MMF 領域にある Reset_Handler にジャンプします。
フラッシュローダー	(7)	R_BSP_WarmStart がコールされた時の event が BSP_WARM_START_PRE_C の場合、VTOR を 0000 0400h に変更し、ベクタテーブルを変更します。以降、フラッシュローダーが動作します。
	(8)	更新モードフラグをクリアします。
	(9)	ホスト PC から更新データを受信し、更新処理を行います。
	(10)	更新処理に成功した場合、BTFLG をトグルしてブートブロックを切り替え、リセットします。
アプリケーション	(11)	ホスト PC から更新要求コマンドを受信した場合、更新モードフラグを true に設定し、リセットします。

3.4 ホスト PC 間通信プロトコル

本アプリケーションノートでは、図 6～図 8に示すプロトコルでホスト PC と通信を行い、内蔵フラッシュの更新処理を行います。図中の各コマンドと応答については表 7に示します。

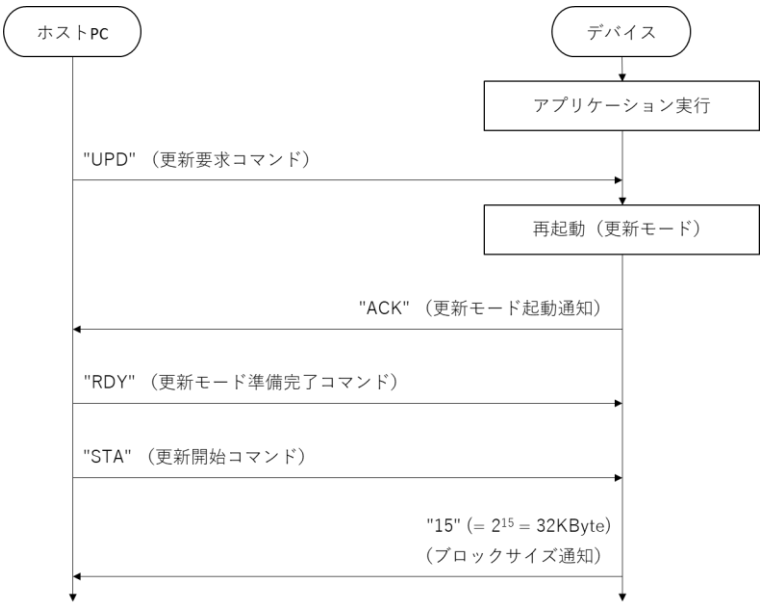


図 6 更新処理フロー（更新モード開始）

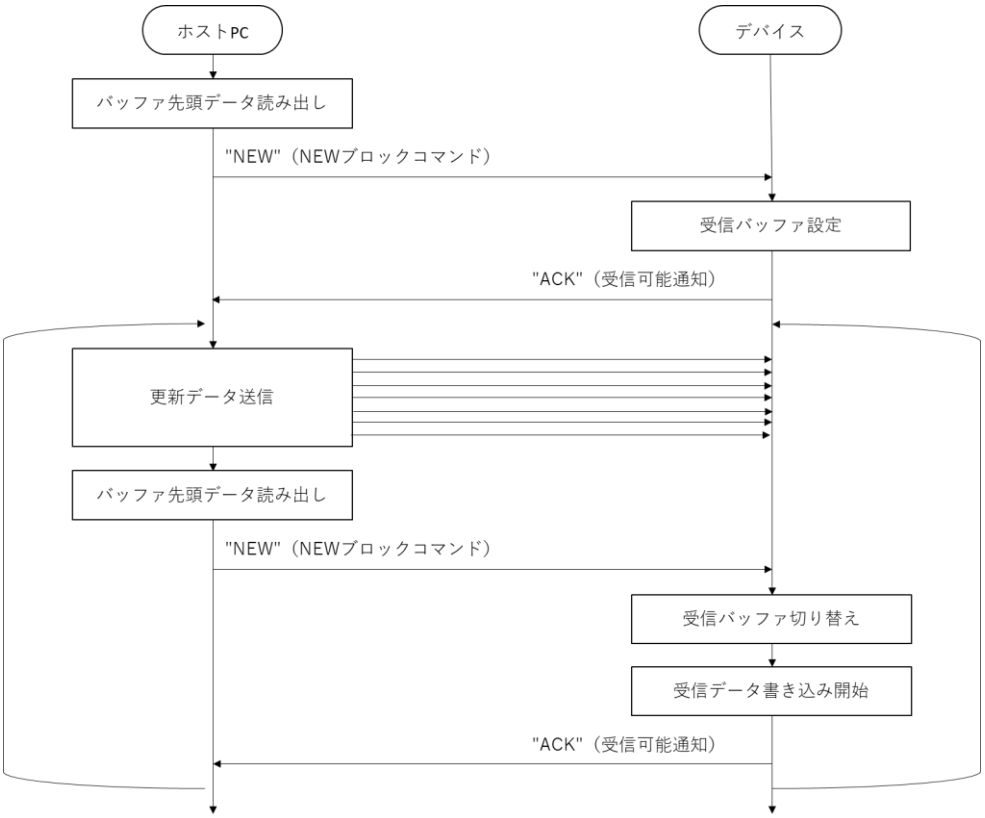


図 7 更新処理フロー（更新データ転送と書き込み）

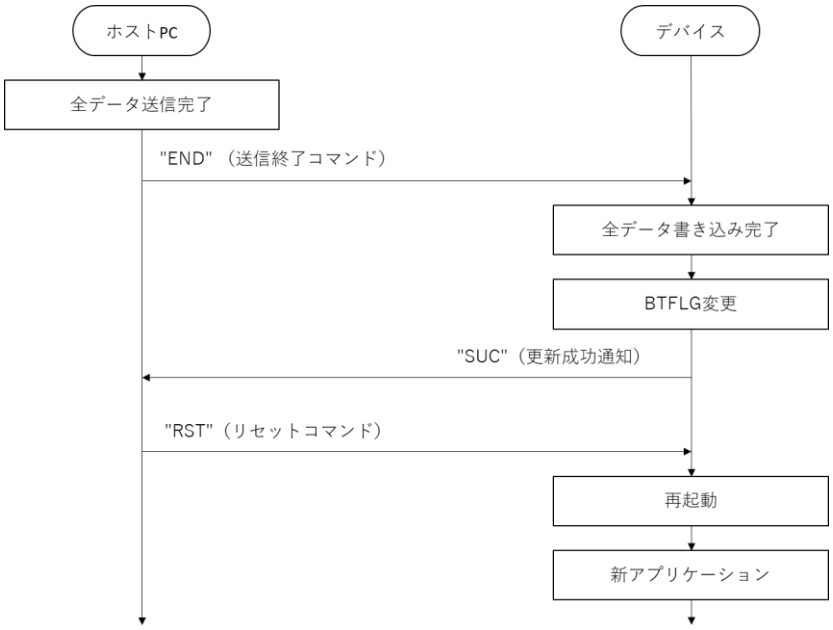


図 8　更新処理フロー（書き込み完了と再起動）

表 7　コマンドと応答

方向	コマンド／通知		機能
ホスト PC → デバイス	“UPD”	更新要求コマンド	デバイスを更新モードで再起動
	“RDY”	更新モード準備完了コマンド	更新モードの準備が完了したことを通知
	“STA”	更新開始コマンド	更新処理の開始
	“NEW”	New ブロックコマンド	ブロックの先頭データの送信開始通知
	“END”	送信終了コマンド	全データ送信完了通知
	“SWP”	ブート変更コマンド	BTFLG をトグルして、ブートブロックを切り替える
	“RST”	リセットコマンド	デバイスを再起動させる
ホスト PC ← デバイス	“n”	ブロックサイズ通知	更新処理プログラムが受信処理に使用する受信バッファのサイズが 2^n バイトであることを 2 桁の数字で通知する（例 : ”15” の場合、2^{15}=32768 バイト）
	“ACK”	更新モード起動通知	ホスト PC からの”UPD”コマンドに対する応答
		ACK 通知	ホスト PC からの”NEW”コマンドに対する応答
	“SUC”	更新成功通知	更新処理成功
	“ERR”	エラー発生通知	更新処理中エラー発生
	“FAI”	更新失敗通知	更新処理失敗

図 6は、ホスト PC から更新要求コマンドを受信したデバイスが更新モードで再起動することを示しています。再起動後、ホスト PC に対して更新モード起動通知を送信し、ホスト PC から更新モード準備完了コマンドの受信を待ちます。この受信において、1 秒以内に同コマンドが受信できなかった場合は、再度、更新モード起動通知の送信と、更新モード準備完了コマンドの受信待ちを繰り返し行います。更新モード準備完了コマンドを受信すると、ホスト PC からの更新開始コマンド受信を待ち、その受信後、デバイスが用意する受信バッファのブロックサイズを通知します。本アプリケーションノートでは、マイコンが持つコードフラッ

シュの最大ブロックサイズを通知しています。図 6 は S7G2 の例となっており、S7G2 のコードフラッシュの最大ブロックサイズである 32KB(= 2^15)を示す”15”を通知しています。S5D9 においても、最大ブロックサイズが 32KB なので”15”を通知します。S3A7 の場合、コードフラッシュの最大ブロックサイズは 2KB(= 2^11)なので、”11”を通知します。

図 7は、ホスト PC の更新データ送信と、それを受信したデバイスの書き込みに関するフローを示しています。ホスト PC は、デバイスの用意する受信バッファの先頭データを更新データから読み出すと、New ブロックコマンドを送信します。デバイスは New ブロックコマンドを受信すると、受信バッファを用意し、ACK を通知します。デバイスからの ACK を受信したホスト PC は、モトローラ S レコード形式のデータを 1 行ずつデバイスに送信します。ホスト PC は、次のブロックに対する更新データを読み出すと、New ブロックコマンドを送信します。デバイスは、この New ブロックコマンドを受信すると、それまで受信していたバッファに対する書き込み処理を開始し、それとは別のもう一つのバッファを受信バッファに設定し、ACK を通知します。

図 8は、ホスト PC が全データを送信完了した後の処理を示しています。ホスト PC は全データの送信完了を送信終了コマンドでデバイスに通知します。これを受信したデバイスは、全データの書き込み完了後、BTFLG を 1→0、あるいは、0→1 に変更し、ホスト PC に更新成功を通知します。ホスト PC からリセットコマンドを受信すると、デバイスは変更された BTFLG の指すブートブロックで起動します。これにより、直前の更新処理で書き込まれた新しいアプリケーションが起動します。

3.5 更新モードフラグの書き換え

本アプリケーションノートでは、更新モードフラグを VBATT バックアップレジスタ(VBTBKRn)に格納しています。バックアップレジスタはマイコンが持つバッテリバックアップ機能の一つです。電力低下が生じた場合に、リアルタイムクロックやサブクロック発信器へバッテリによる部分給電を行います。バックアップレジスタはマイコンへの電源供給が遮断されるまで保持され、各種リセットによって初期化されません。そのため、リセット後も保持したいデータの一時的な退避に使用します。バックアップレジスタは 512 個の 1 バイトレジスタから構成されており、このレジスタへ任意のデータを格納することができます。

本アプリケーションノートでは PC から更新モードによる再起動コマンドを受信した際に、バックアップレジスタに更新モードフラグを書き込んだ後にリセットします。リセット後にブートプログラム上でバックアップレジスタ内の更新モードフラグを確認し、同フラグがセットされていた場合にフラッシュローダーを起動します。フラッシュローダー起動直後にバックアップレジスタに書き込まれた更新モードフラグをクリアします。

バックアップレジスタへの書き込みは「6.3VBATTバックアップレジスタ(VBTBKRn)へのアクセス方法」を参照してください。S3A7 のみバックアップレジスタへのアクセス手順が S7G2,S5D9 と異なります。

本アプリケーションノートでは、更新モードフラグを VBATT バックアップレジスタの 1 バイトのデータで簡易的に管理しています。しかし、パワーオンリセットで起動時は同レジスタの値は不定ですので、この時、同レジスタの値が更新モード用の値と一致し、意図せず更新モードで起動する可能性が考えられます。この問題を避けるため、更新モードフラグのサイズや値を調整することを検討してください。更には、より確実な方法として、更新モードフラグの管理を同レジスタでの管理ではなく、データフラッシュで管理する方法も検討してください。

4. アプリケーションへの機能追加

本アプリケーションノートでは、アプリケーションに対して、ホスト PC から更新要求コマンドを受信するためのスレッド(以下 Hidden Thread と称します)を追加します。本アプリケーションノートに添付の Blinky、Hello World、Weather Panel の各アプリケーションにおいては、元となったアプリケーションに対して、この Hidden Thread を追加しています。この機能を実装するため、同スレッドには Communications Framework on sf_el_ux_comms と Flash Driver on r_flash_hp(S3A7 の場合 Flash Driver on r_flash_lp)を追加します。（図 9）

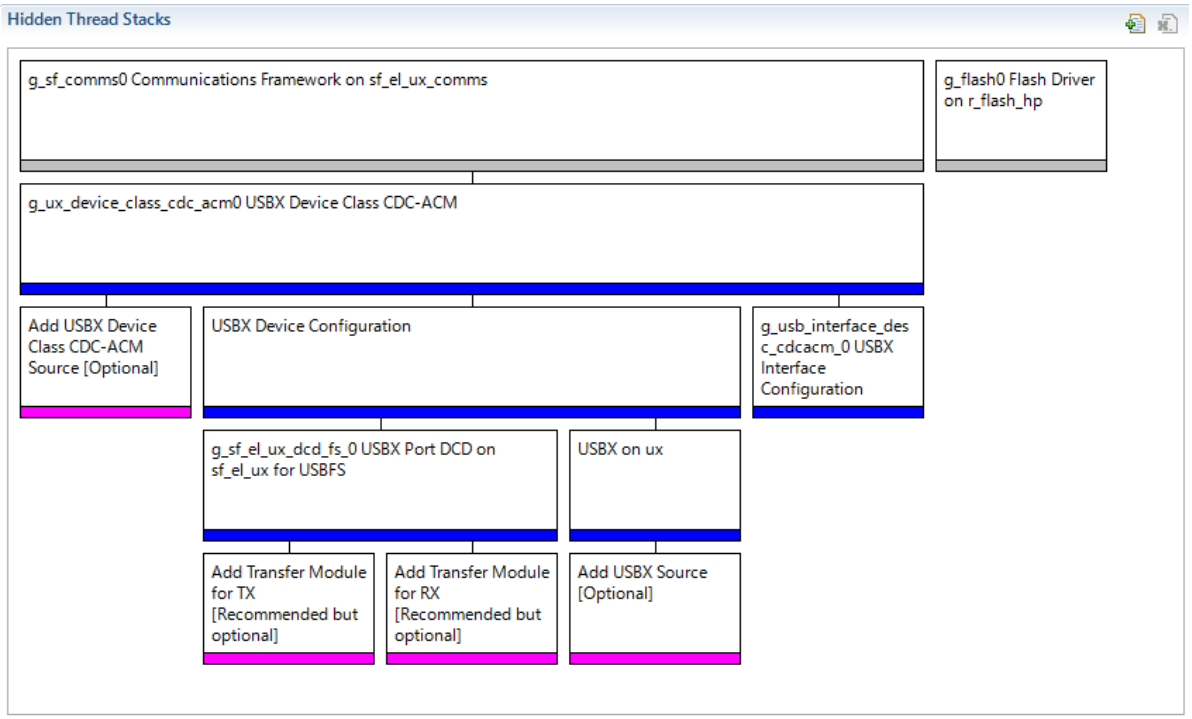


図 9 更新コマンド受信用スレッドのコンフィギュレーション(S7G2, S5D9)

4.1 sf_el_ux_comms の追加

Hidden Thread において、PC と USB CDC ACM で通信を行うため、Communications Framework on sf_el_ux_comms のインスタンス g_sf_comms0 を追加します。Hidden Thread を選択し、「Framework」→「Connectivity」→「Communications Framework on sf_el_ux_comms」を選択することで追加します。USBX Port DCD on sf_el_ux for USBFS にある Transfer Driver on r_dtc は、TX、RX の両方とも削除します。

4.2 Flash Driver の追加

本アプリケーションノートでは Hidden Thread に Flash Driver を追加しています。これは実行中のアプリケーションが”SWAP”コマンドを受信し、実行するアプリケーションを切り替える際、BTFLG への書き換えが必要になるためです。

このため、Flash Driver のインスタンス g_flash0 を追加します。

4.2.1 Flash Driver on r_flash_hp の追加(S7G2, S5D9)

Hidden Thread を選択し、「Driver」→「Storage」→「Flash Driver on r_flash_hp」を選択することで追加します。

図 10に示すように、g_flash0 のプロパティにおいて、「Code Flash Programming Enable」を「Disabled」に設定しています。、これは、S7G2 では Hidden Thread ではコードフラッシュに書き込みを行わないため、この初期値のままとしています。S5D9 では BTFLG がコードフラッシュ領域に配置されており、BTFLG を書き換える際にコードフラッシュへアクセスします。そのため、S5D9 においては「Enable」に設定する必要があります。尚、Data Flash へは書き込みを行わないため、「Data Flash Background Operation」を「Disabled」に設定しています。

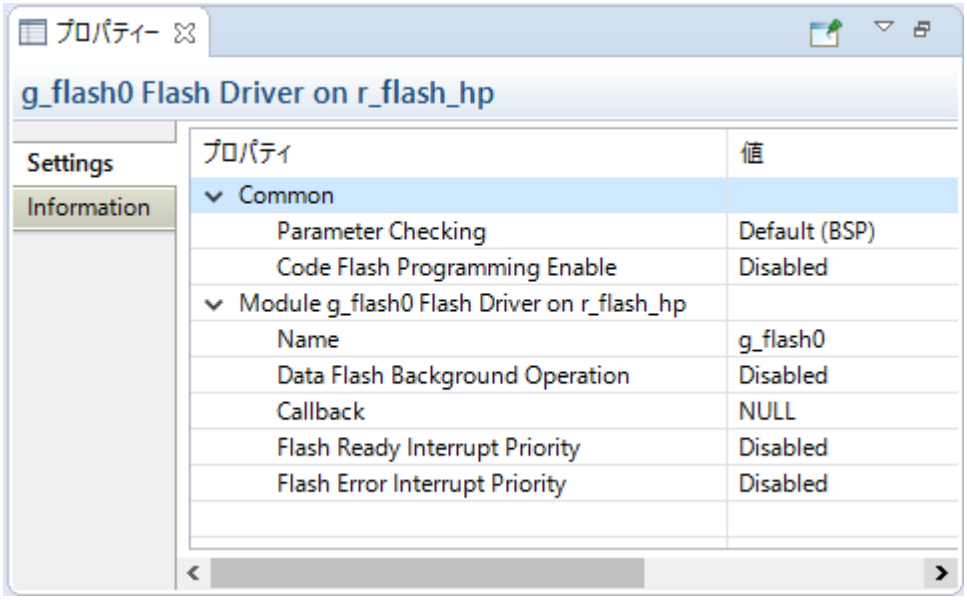


図 10 g_flash0 のコンフィギュレーション(S7G2, S6D9)

4.2.2 Flash Driver on r_flash_lp の追加(S3A7)

Hidden Thread を選択し、「Driver」→「Storage」→「Flash Driver on r_flash_lp」を選択することで追加します。g_flash0 のプロパティにおいて、「Code Flash Programming Enable」を「Enable」にします。これは S5D9 と同様に BTFLG がコードフラッシュに配置されており、アプリケーション上から BTFLG を切り替える際に「Enable」にしておく必要があるためです。「Data Flash Background Operation」は「Disabled」に設定しています。

4.3 Hidden Thread の実装

Hidden Thread の動作フローの概要を図 11に示します。

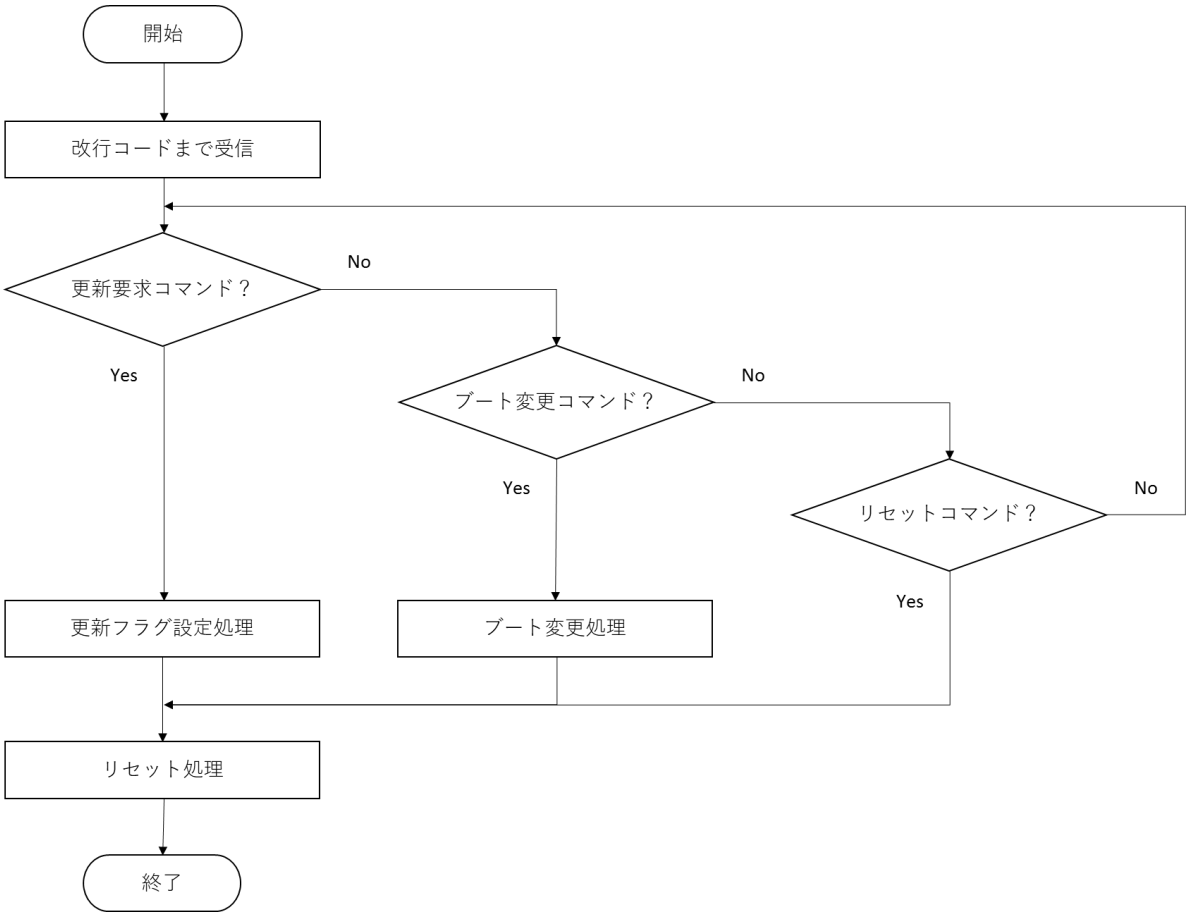


図 11 Hidden Thread のフロー図

Hidden Thread は、更新要求コマンドを受信後、VBATT バックアップレジスタ内に更新モードフラグを設定した後、ソフトウェアリセットを行います。この再起動により、フラッシュローダーが起動します（図 5、および、表 6の(2)参照）。また、ブート変更コマンドを受信後、BTFLG をトグルした後にソフトウェアリセットを行います。この再起動後は直前に起動していたアプリケーションではないもう一方のアプリケーションで起動します。リセットコマンドを受信した場合はソフトウェアリセットだけを行います。

4.4 ブート処理の変更

4.4.1 boot_entry

図 5に示すように、起動直後に実行する処理は、通常の Synergy アプリケーションで用いられる Reset_Handler ではなく、boot_entry を実行します。この boot_entry において、更新モードフラグが設定されている場合はフラッシュローダーの起動を行い、それ以外の場合は BTFLG の値に応じて MMF の設定を行った後、Reset_Handler にジャンプします。

このため、ベクタテーブル__Vectors を設定している以下のファイルの内容を変更する必要があります。

- S7G2 … synergy¥ssp¥src¥bsp¥cmsis¥Device¥RENESAS¥S7G2¥Source¥ startup_S7G2.c
- S5D9 … synergy¥ssp¥src¥bsp¥cmsis¥Device¥RENESAS¥S7G2¥Source¥ startup_S5D9.c
- S3A7 … synergy¥ssp¥src¥bsp¥cmsis¥Device¥RENESAS¥S3A7¥Source¥ startup_S3A7.c

変更は以下の手順で行います。

1. e² studio のプロジェクト・エクスプローラにおいて上記の startup_{デバイス名}.c ファイルで右クリックし、「ビルドから除外…」でビルド対象から外します。
2. 同ファイルを src フォルダの直下にコピーします。
3. 上でコピーした src/startup_{デバイス名}.c において__Vectors 内の Reset_Handler を boot_entry に変更します。

boot_entry 関数は hidden_thread_entry.c で定義しています。

4.5 リンカスクリプト

表 2に示したようなメモリマッピングを行うために必要なメモリリージョンを設定します。GNU コンパイラの場合、script フォルダ内にあるリンカスクリプトにおいて、表 8、表 9、表 10に示すようにメモリリージョンを設定します。

表 8 アプリケーションのメモリリージョン(SK-S7G2)

リージョン名	開始アドレス	終了アドレス	サイズ	メモリデバイス
FLASH_BLOCK0	0000 0000h	0000 1FFFh	0000 2000h	Code Flash
FLASH_BLOCK1	0000 2000h	0000 3FFFh	0000 2000h	
FLASH_APP_PHY	0002 0000h	001F FFFFh	001E 0000h	
FLASH_APP_MMF	0202 0000h	021F FFFFh	001E 0000h	
RAM	1FFE 0000h	2007 FFFFh	000A 0000h	SRAM
DATA_FLASH	4010 0000h	4010 FFFFh	0001 0000h	Data Flash
QSPI_FLASH	6000 0000h	63FF FFFFh	0400 0000h	QSPI
SDRAM	9000 0000h	91FF FFFFh	0200 0000h	SDRAM*1

*1 : SK-S7G2 は非搭載

表 9 アプリケーションのメモリリージョン(PK-S5D9)

リージョン名	開始アドレス	終了アドレス	サイズ	メモリデバイス
FLASH_BLOCK0	0000 0000h	0000 1FFFh	0000 2000h	Code Flash
FLASH_BLOCK1	0000 2000h	0000 3FFFh	0000 2000h	
FLASH_APP_PHY	0002 0000h	000F FFFFh	000E 0000h	
FLASH_APP_MMF	0202 0000h	020F FFFFh	000E 0000h	
RAM	1FFE 0000h	2007 FFFFh	000A 0000h	SRAM
DATA_FLASH	4010 0000h	4010 FFFFh	0001 0000h	Data Flash
QSPI_FLASH	6000 0000h	63FF FFFFh	0400 0000h	QSPI
SDRAM	9000 0000h	91FF FFFFh	0200 0000h	SDRAM*1

*1 : PK-S5D9 は非搭載

表 10 アプリケーションのメモリリージョン(DK-S3A7)

リージョン名	開始アドレス	終了アドレス	サイズ	メモリデバイス
FLASH_BLOCK0	0000 0000h	0000 1FFFh	0000 2000h	Code Flash
FLASH_BLOCK1	0000 2000h	0000 3FFFh	0000 2000h	
FLASH_APP_PHY	0002 0000h	0007 FFFFh	0006 0000h	
FLASH_APP_MMF	0202 0000h	0207 FFFFh	0006 0000h	
RAM	2000 0000h	2003 FFFFh	0003 0000h	SRAM
DATA_FLASH	4010 0000h	4010 3FFFh	0000 4000h	Data Flash
QSPI_FLASH	6000 0000h	63FF FFFFh	0400 0000h	QSPI

リンカスクリプトで対応が必要な箇所を以下に示します。e2studio 環境におけるリンカスクリプトの記述をコード 1に示します。

- エントリを boot_entry に設定するため、ENTRY(boot_entry)を設定します。
- コードフラッシュに配置するコードを FLASH_BLOCK0 と FLASH_APP_PHY に分けて配置するため、.boot セクションと.text セクションに分けます。
- .boot セクションは FLASH_BLOCK0 に配置します。
- .text セクションは FLASH_APP_MMF に配置し、その実体を FLASH_APP_PHY に配置します。
- .data セクションは RAM に配置し、その初期値データの実体は FLASH_APP_PHY に配置します。

```
MEMORY
{
  FLASH_BLOCK0(rx)   : ORIGIN = 0x00000000, LENGTH = 0x00002000
  FLASH_BLOCK1(rx)   : ORIGIN = 0x00002000, LENGTH = 0x00002000
  FLASH_APP_PHY(rx)  : ORIGIN = 0x00020000, LENGTH = 0x001E0000
  FLASH_APP_MMF(rx)  : ORIGIN = 0x02020000, LENGTH = 0x001E0000
  RAM (rwx)          : ORIGIN = 0x1FFE0000, LENGTH = 0x000A0000
  DATA_FLASH (rx)    : ORIGIN = 0x40100000, LENGTH = 0x00010000
  QSPI_FLASH (rx)    : ORIGIN = 0x60000000, LENGTH = 0x04000000
  SDRAM (rwx)        : ORIGIN = 0x90000000, LENGTH = 0x02000000
}

ENTRY(boot_entry)

SECTIONS
{
.boot :
{
        (略)
} > FLASH_BLOCK0 = 0xFF

.text :
{
        (略)
} > FLASH_APP_MMF AT> FLASH_APP_PHY = 0xFF

.data :
{
    (略)
} > RAM AT> FLASH_APP_PHY
```

コード 1　e2studio，アプリケーションのリンカスクリプト（script¥S7G2.ld 抜粋）

IAR EWARM for Synergy 環境におけるリンカスクリプトの記述をコード 2 に示します。

- 表 8に示すメモリリージョンを定義するために、define symbol 文で各メモリリージョンの開始アドレスと終了アドレスを定義しています。
- FLASH_BLOCK0 の開始アドレスと終了アドレスを region_BOOT_start , region_BOOT_end で定義します。
- FLASH_APP_MMF の開始アドレスと終了アドレスを region_MMF_start , region_MMF_end で定義します。
- FLASH_APP_PHY の開始アドレスと終了アドレスを region_FLASH_start, region_FLASH_end で定義します。
- 定義した開始アドレスと終了アドレスのシンボルを用いて、define region 文で各メモリリージョンのアドレス範囲を定義します。
- FLASH_BLOCK0 の開始～終了までのアドレス範囲を BOOT_Region と定義し、定義したシンボルからメモリリージョンのアドレス範囲を指定します。
- FLASH_APP_MMF の開始～終了までのアドレス範囲を FLASH_APP_MMF と定義し、定義したシンボルからメモリリージョンのアドレス範囲を指定します。
- FLASH_APP_PHY の開始～終了までのアドレス範囲を FLASH_APP_PHY と定義し、定義したシンボルからメモリリージョンのアドレス範囲を指定します。
- コードフラッシュに配置するコードを FLASH_BLOCK0 と FLASH_APP_PHY に分けて配置するため、BOOT_Region リージョンを定義します。
- .text や.rodata などの各セクションは、表 8で示したメモリリージョンに手動で配置します。手動で配置するセクションを initialize manually で指定します。
- Place in 文と Place at start of 文を用いて、定義したメモリリージョンへの配置を行います。
- boot_ernty を BOOT_Region に配置します。
- .text セクションは FLASH_APP_MMF に配置し、その実体を FLASH_APP_PHY に配置します。
- .data セクションは RAM に配置し、その初期値データの実体は FLASH_APP_PHY に配置します。

```
define symbol region_VECT_start             = 0x00000000;
define symbol region_VECT_end               = 0x000003FF;
define symbol region_ROMREG_start           = 0x00000400;
define symbol region_ROMREG_end             = 0x000004FF;
define symbol region_BOOT_start             = 0x00000500;
define symbol region_BOOT_end               = 0x00001FFF;
define symbol region_FLASH_start            = 0x00020000;
define symbol region_FLASH_end              = 0x001FFFFF;
define symbol region_MMF_start              = 0x02020000;
define symbol region_MMF_end                = 0x021FFFFF;
define symbol region_RAM_start              = 0x1FFE0000;
define symbol region_RAM_end                = 0x2007FFFF;
define symbol region_DF_start               = 0x40100000;
define symbol region_DF_end                 = 0x4010FFFF;
define symbol region_SDRAM_start            = 0x90000000;
define symbol region_SDRAM_end              = 0x91FFFFFF;
define symbol region_QSPI_start             = 0x60000000;
define symbol region_QSPI_end               = 0x63FFFFFF;

define region VECT_region    = mem:[from region_VECT_start to region_VECT_end];
define region ROMREG_region  = mem:[from region_ROMREG_start to region_ROMREG_end];
define region BOOT_region    = mem:[from region_BOOT_start to region_BOOT_end];
                             (略)
define region QSPI_region    = mem:[from region_QSPI_start to region_QSPI_end];

initialize manually          { readwrite };
initialize manually          { section .qspi_non_retentive };
                             (略)
initialize manually          { section .code_in_ram };
initialize manually          { section .text* };
initialize manually          { section .rodata* };
                             (略)
```

コード 2　EWARM for Synergy，アプリケーションのリンカスクリプト(script¥S7G2.icf 抜粋)

```
place at start of VECT_region     { ro section .vectors };
place in VECT_region              { ro section .vector.* };
place in ROMREG_region            { ro section .rom_registers };
place in BOOT_region              {  section .boot_entry,
                                     block VECT_INFO,
                                     block LOCK_LOOKUP)   };

place at start of FLASH_APP_MMF   {  section .text*,
                                     section .rodata*,
                                     section .version,
                                     block USB_DEV_DESC_BLK,
                                  };
place at start of FLASH_APP_PHY   {  section .text*_init,
                                     section .rodata*_init,
                                     section .version_init,
                                     block USB_DEV_DESC_BLK_INIT,
                                     block QSPI_NON_RETENTIVE_INIT_BLOCK,
                                     lock RAM_INIT_CODE,
                                     ro,
                                     section .data_init,
                                  };
place at start of FLASH_region    { block VECT_INFO };
place in FLASH_region             { block LOCK_LOOKUP,
                                  ro,
                                  ro section .rodata,
                                  block QSPI_NON_RETENTIVE_INIT_BLOCK,
                                  block RAM_INIT_CODE,
                                  block USB_DEV_DESC_BLK };
place in RAM_region               { 略 };
place in DF_region                { 略 };
place in SDRAM_region             { 略 };
place in QSPI_region              { 略 };
place in QSPI_region              { 略 };
place in RAM_region               { 略 };
(略)
```

コード 3　EWARM for Synergy, プリケーションのリンカスクリプト(script¥S7G2.icf 抜粋, コード 2 の続き)

4.6 初期値データの RAM コピー元設定(EWARM for Synergy)

アプリケーション実行後の初期化処理において、EWARM for Synergy 環境では、ROM に格納された初期値データを RAM へ展開する際に MMF アドレス領域からコピーするように明示的に設定する必要があります。MMF が有効である場合、MMF の仮想アドレスである 0x0200 0000 をオフセットとして実アドレスに加えます。

SystemInit 関数内において、RAM の初期値データを ROM から RAM へコピーする処理を行いますが、コード 1 に示すように記述することで、MMF 領域の ROM データを RAM へコピーします。

```
#elif defined(__ICCARM__)
        #define ADDR_MMF       0x02000000UL
        uint32_t                    addr_offset = 0;
        if ( __get_PC() >= ADDR_MMF )
        {
            addr_offset = ADDR_MMF;
        }
        bsp_section_copy((uint8_t *)((uint32_t)__section_begin(".data_init") | addr_offset ),
                                (uint8_t *)__section_begin(".data"),
                                (uint32_t)__section_size(".data"));
        /* Copy functions to be executed from RAM. */
        #pragma section=".code_in_ram"
        #pragma section=".code_in_ram_init"
        bsp_section_copy((uint8_t *)((uint32_t)__section_begin(".code_in_ram_init") | addr_offset),
                                (uint8_t *)__section_begin(".code_in_ram"),
                                (uint32_t)__section_size(".code_in_ram"));
        /* Copy main thread TLS to RAM. */
        #pragma section="__DLIB_PERTHREAD_init"
        #pragma section="__DLIB_PERTHREAD"
        bsp_section_copy((uint8_t *)((uint32_t)__section_begin("__DLIB_PERTHREAD_init") | addr_offset),
                                (uint8_t *)__section_begin("__DLIB_PERTHREAD"),
                                (uint32_t)__section_size("__DLIB_PERTHREAD_init"));
#endif
```

コード 4　ROM から RAM へのデータコピー処理(src/system_S7G2.c 抜粋)

4.7 留意事項

本アプリケーションノートでは、アプリケーション実行時に更新要求コマンドを受信した場合、アプリケーションの実行状態に関わらずソフトウェアリセットを行っています。これは、アプリケーション本体には手を加えない簡易な方式としているためです。

5. フラッシュローダー

5.1 フラッシュローダーの概要

フラッシュローダーは、更新モードフラグが設定された状態で起動した時に実行されます。図 7、図 8に示すようにホスト PC と通信を行うことで更新データを受信し、受信したデータを直前まで起動していたアプリケーションのアプリケーション保存領域とは別のアプリケーション保存領域に書き込みます。

フラッシュローダーは更新データ受信用のバッファを２つ持ち、受信した更新データをどちらかのバッファに格納します。このバッファ（受信バッファ）に対するデータ受信が完了した後（図 7の”NEW”受信後）は、受信バッファをもう一方のバッファに切り替え、受信したデータの入ったバッファを内蔵フラッシュに書き込む処理を開始します。その後、ホスト PC に受信バッファの切り替え処理が完了したことを通知（図 7の”ACK”通知）し、更新データの受信を続けます。

更新データの受信処理と更新データの書き込み処理は、それぞれ異なるスレッドで処理しているため、更新データ受信と、受信済み更新データの書き込みは並行して行なわれます。この動作についてのシーケンスを図 12に示します。

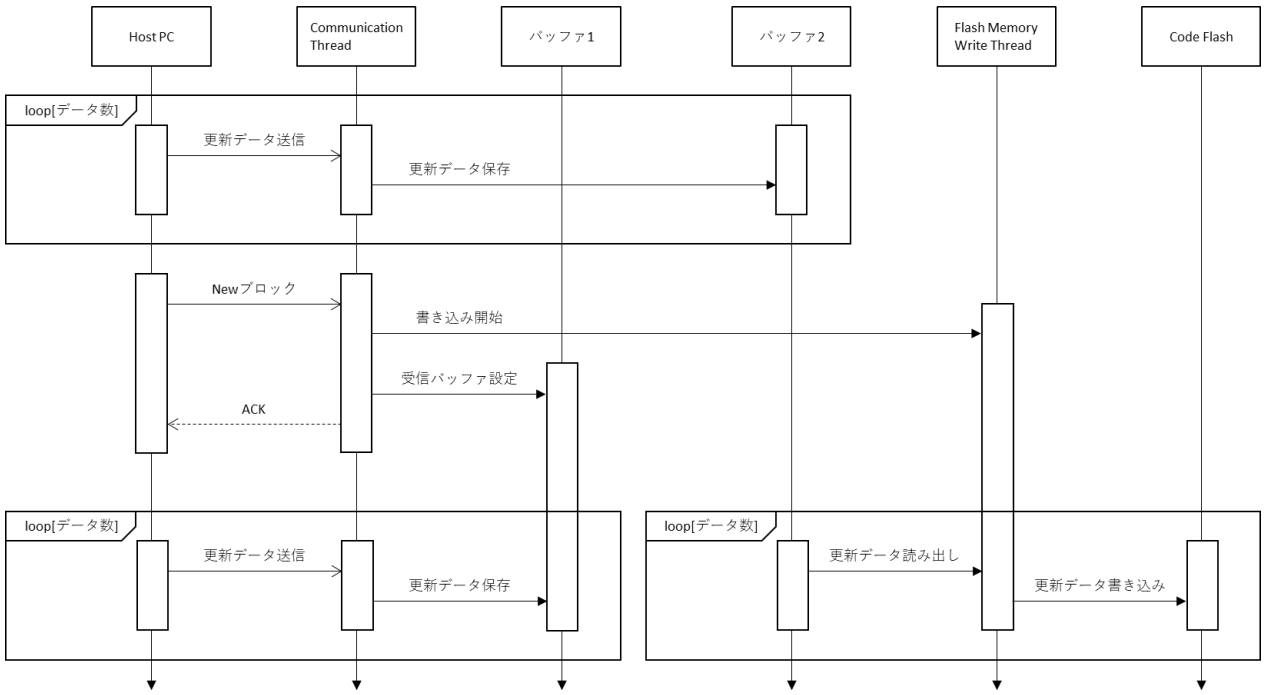


図 12 フラッシュローダーのシーケンス（更新データ受信から更新データ書き込み）

5.2　フラッシュローダーの構成

フラッシュローダーは図 13に示す構成になっています。また、図内の各モジュールの機能概要を表 11に示します。

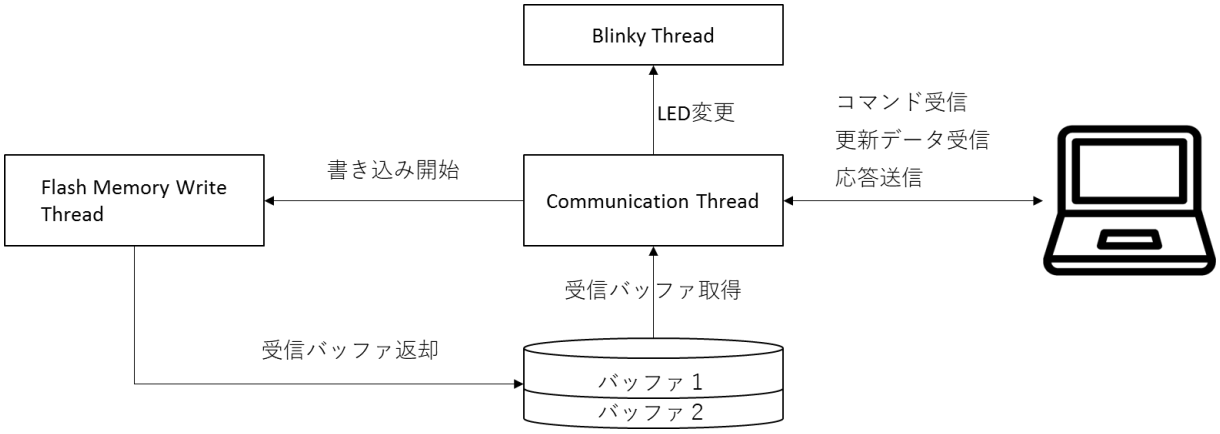


図 13 フラッシュローダーの構成

表 11　フラッシュローダーのモジュールとその機能概要

モジュール	機能		
Blinky Thread	フラッシュローダーの動作状態を LED で表示します。		
	SK-S7G2 PK-S5D9	（　　橙）点滅	更新開始コマンド待ち状態
		（　赤　）点滅	更新データ受信状態
		（緑　　）点滅	更新処理完了状態
		（　赤、橙）点滅（交互）	更新失敗
		（緑、赤、橙）点滅（同時）	通信エラー発生
	DK-S3A7	（　　緑　）点滅	更新開始コマンド待ち状態
		（　　赤）点滅	更新データ受信状態
		（　　緑）点滅	更新処理完了状態
		（　　緑/赤）点滅(交互)	更新失敗
		（　赤　緑/赤）点滅(同時)	通信エラー発生
Communication Thread		ホスト PC と通信を行い、各コマンドや更新データの受信、および、応答の送信を行います。受信バッファに対するデータ受信が完了すると、受信バッファをもう一方のバッファに切り替えた後、Flash Memory Write Thread に対して、受信が完了したバッファの書き込み要求を行います。	
Flash Memory Write Thread		更新データの受信が完了したバッファのデータを内蔵フラッシュに書き込む処理を行います。書き込み完了後、そのバッファを返却します。	
バッファ 1 バッファ 2		ホスト PC から受信した更新データを保存するバッファであり、同サイズの２つのバッファで構成されます。バッファサイズはコードフラッシュのブロックサイズの最大値と同じサイズであり、S7G2, S5D9 の場合は 32KB、S3A7 の場合は 2KB とします。	

5.3 フラッシュローダーの動作概要

5.3.1 Blinky Thread

フラッシュローダーの動作状態を LED の点滅で示します。点滅する LED のパターンは、blinky_set_led 関数で指定します。

5.3.2 Communication Thread

Communication Thread のフローの概要を図 14に示します。一部の処理は省略しています。

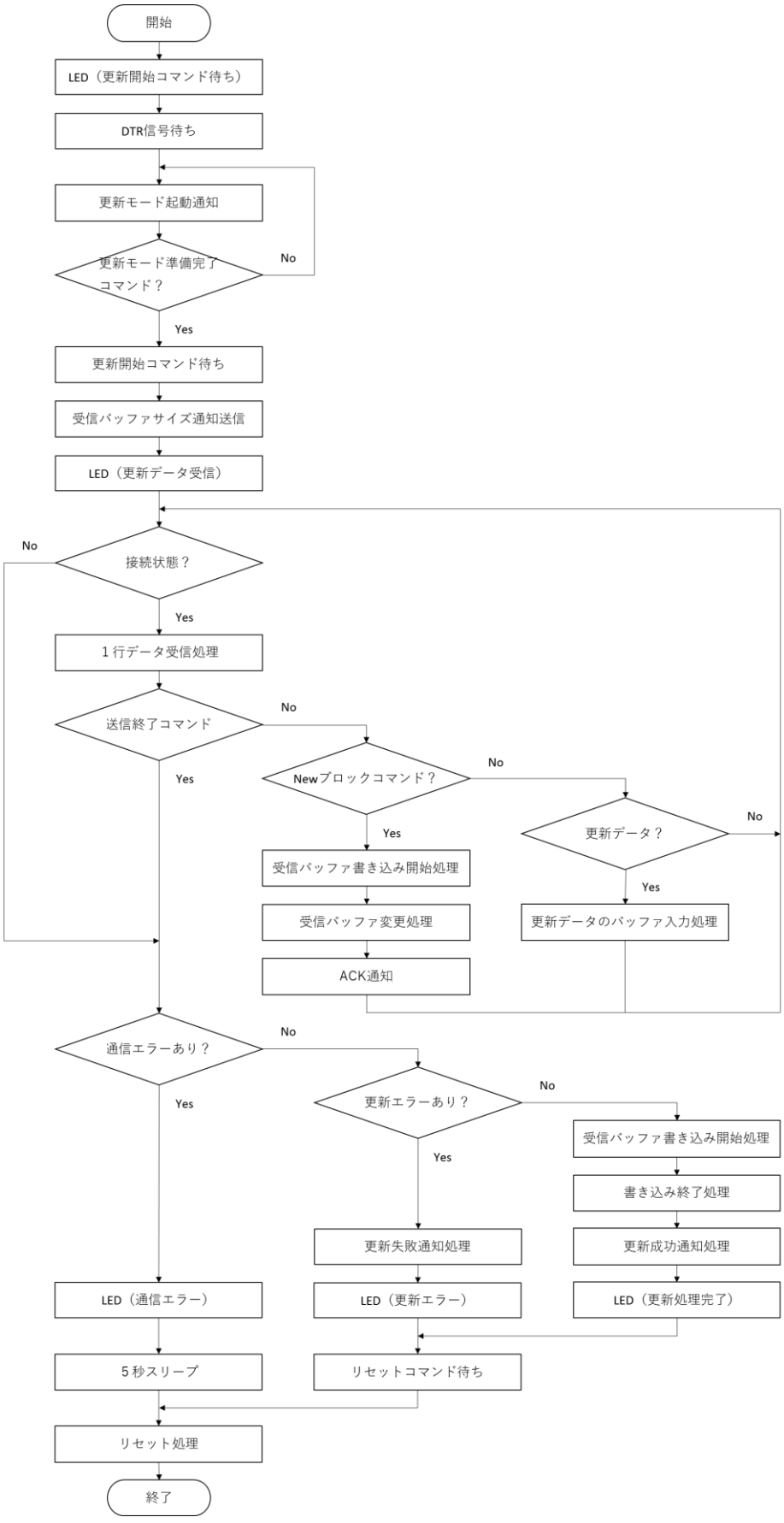


図 14 Communication Thread のフロー

更新モード準備完了コマンドの受信処理は、一定時間内に受信がない場合にタイムアウトを行います。このタイムアウトを行うためのタイマ処理として、r_gpt ドライバを利用しています。同ドライバで 1 秒のタイマを設定した g_timer_comm を用意し、受信処理開始前に同タイマをスタートさせます。同タイマのコールバック関数として callback_timer_comm を登録し、タイマが失効した場合は tx_thread_wait_abort 関数で本スレッドの待ち状態を解除します。

5.3.3 Flash Memory Write Thread

Flash Memory Write Thread は、書き込み要求イベントを受信し、通知されたイベント内の書き込み情報に従って、内蔵フラッシュへの書き込みを行います。同スレッドのフローの概要を図 15に示します。Communication Thread から通知された書き込み要求イベントを受信すると、そのイベント内の書き込み開始アドレス、サイズ、書き込みデータを使って、内蔵フラッシュの書き込み処理を開始します。書き込み処理完了後、コールバック関数をコールして、書き込みが完了したバッファを返却します。

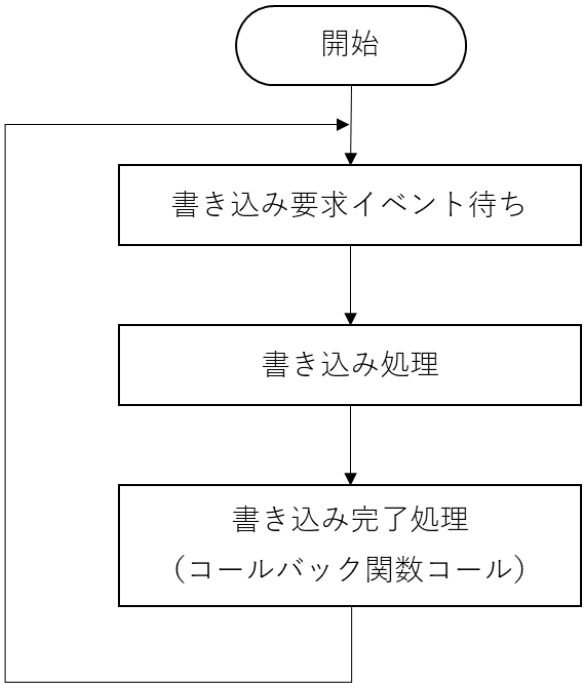


図 15 Flash Memory Write Thread のフロー

書き込み処理のフローを図 16に示します。同図において、ブランクチェック処理前に割り込み禁止に設定し、書き込み処理完了後に割り込み禁止を解除しています。これは、内蔵フラッシュの消去処理中、および、書き込み処理中においては、内蔵フラッシュの読み出しができない状態になるため、この時に別のスレッドに動作が切り替わると問題が発生するからです。

内蔵フラッシュ書き換え処理は RAM 上で実行されます。この RAM 実行に関しては、r_flash_hp ドライバと r_flash_lp ドライバの PLACE_IN_RAM_SECTION、および、system_S7G2.c 内の SystemInit 関数内の「/* Copy initialized RAM data from ROM to RAM. */」コメントの処理を参照してください。

Flash Memory Write Thread のコンフィグレーションを図 17に示します。コードフラッシュへのアクセスが発生するため、r_flash_hp ドライバ(DK-S3A7 使用の場合は r_flash_lp ドライバ)のインスタンス g_flash0 の「Code Flash Programming Enable」を Enable に設定しています。尚、Data Flash へは書き込みを行わないため、「Data Flash Background Operation」を「Disabled」に設定しています。

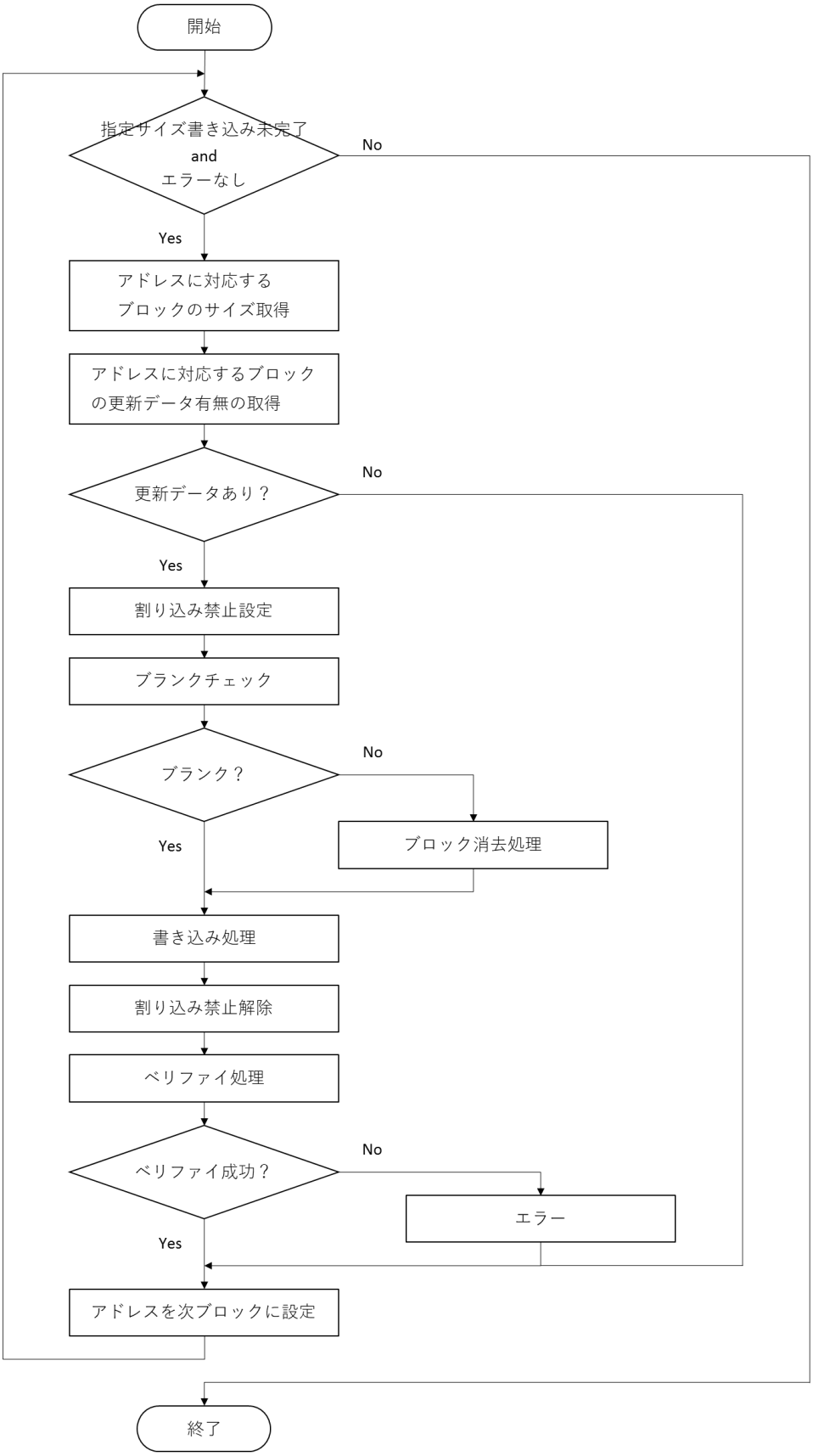


図 16 書き込み処理フロー(flwrite_exec 関数)

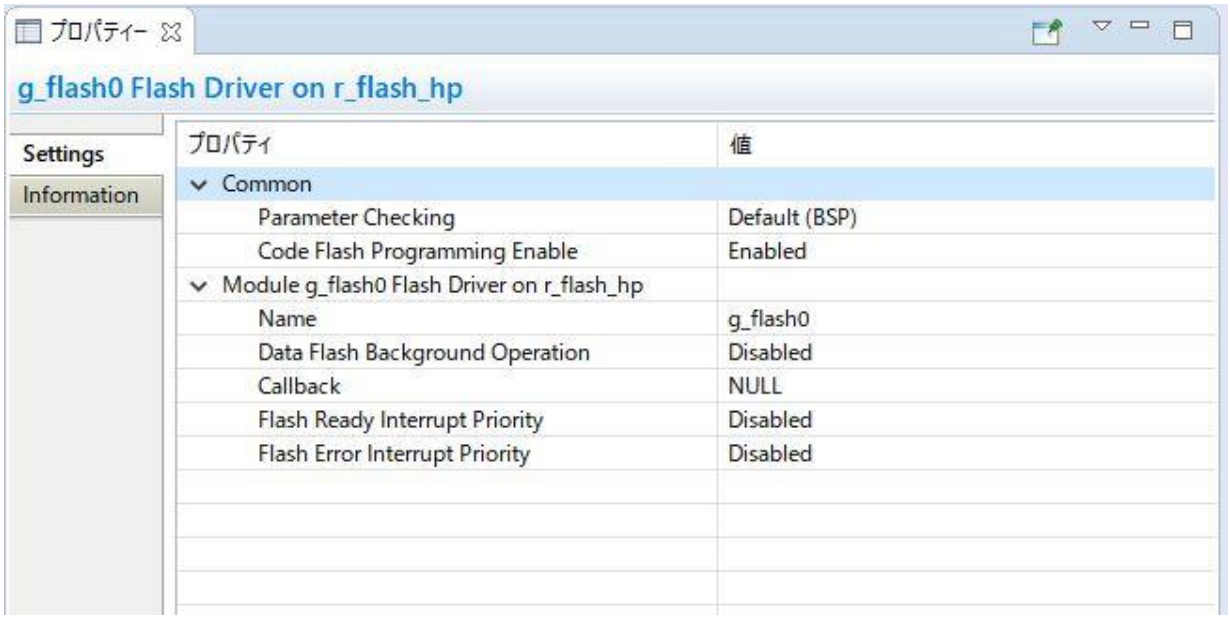


図 17 Flash Memory Write Thread, g_flash0 のコンフィグレーション

5.3.4 flblock モジュール

内蔵フラッシュのブロック管理に関する機能を提供します。関数一覧を表 12に示します。

表 12 flblock モジュールの関数一覧

関数名	機能
flblock_initialize	初期化処理。ブロックに対する更新データの有無を記録するテーブルを初期化します。
flblock_get_info	指定したアドレスに対応するブロック番号とそのサイズを取得します。
flblock_set_dirty	指定されたアドレスの属するブロックに更新データ有りを記録します。
flblock_get_dirty	指定されたアドレスの属するブロックについて、更新データの有無の情報を取得します。

5.3.5 flupdate モジュール

受信した更新データのバッファ管理に関する機能を提供します。関数一覧を表 13に示します。

flupdate_put_data 関数では、ブートブロック 0 に対するデータはブートブロック 1 へ書き込むようにアドレスを調整しています。また、BTFLG が 1 の場合、アプリケーション保存領域 0 に対するデータはアプリケーション保存領域 1 への書き込みとなるように、書き込み先のアドレスを調整しています。MCU グループ毎に設定が異なりますので、ADDR_SHIFT_BOOT、および、ADDR_SHIFT_MMF の各定義を参照してください。

表 13 flupdate モジュールの関数一覧

関数名	機能
flupdate_initialize	初期化処理。受信バッファの初期化などを行います。
flupdate_write_buffer	受信が完了したバッファの書き込み要求を行います。
flupdate_new_buffer	更新データ受信用のバッファを取得します。
flupdate_release_buffer	更新データ受信用のバッファを返却します。
flupdate_put_data	受信した 1 行のモトローラ S レコードデータを受信バッファに書き込みます。
flupdate_finalize	更新データの全書き込みの完了を待ち、その後、BTFLG を変更します。

5.3.6 リンカスクリプト

フラッシュローダーは、そのリセットベクタを 0000 4000h に設定したイメージとなるので、メモリリージョン FLASH の ORIGIN を 0000 4000h に設定します。フラッシュローダーにおける、e2studio 環境でのリンカスクリプトの記述をコード 4、IAR EW for Synergy 環境でのリンカスクリプトをコード 5 に示します。

```
MEMORY
{
  FLASH (rx)          : ORIGIN = 0x00004000, LENGTH = 0x0400000
  RAM (rwx)           : ORIGIN = 0x1FFE0000, LENGTH = 0x00A0000
  DATA_FLASH (rx)     : ORIGIN = 0x40100000, LENGTH = 0x0010000
  QSPI_FLASH (rx)     : ORIGIN = 0x60000000, LENGTH = 0x4000000
  SDRAM (rwx)         : ORIGIN = 0x90000000, LENGTH = 0x2000000
}
```

コード 5 フラッシュローダーのリンカスクリプト(script¥S7G2.ld 抜粋)

```
define symbol region_VECT_start      = 0x00000000;
define symbol region_VECT_end        = 0x000003FF;
define symbol region_ROMREG_start    = 0x00000400;
define symbol region_ROMREG_end      = 0x000004FF;
define symbol region_FLASH_start     = 0x00004000;
define symbol region_FLASH_end       = 0x003FFFFF;
define symbol region_RAM_start       = 0x1FFE0000;
define symbol region_RAM_end         = 0x2007FFFF;
define symbol region_DF_start        = 0x40100000;
define symbol region_DF_end          = 0x4010FFFF;
define symbol region_SDRAM_start     = 0x90000000;
define symbol region_SDRAM_end       = 0x91FFFFFF;
define symbol region_QSPI_start      = 0x60000000;
define symbol region_QSPI_end        = 0x63FFFFFF;
```

コード 6　フラッシュローダーのリンカスクリプト(script¥S7G2.icf 抜粋)

6.　各 MCU グループにおける差分

6.1　　フラッシュメモリの仕様

S7G2、S5D9、S3A7 ではフラッシュメモリの仕様が異なり、ソフトウェアで留意すべき点があります。本アプリケーションノートで考慮すべき差分を以下に示します。

6.1.1　　Flash Driver

S7G2 と S5D9、S3A7 ではフラッシュメモリへのデータ書き込み時に使用するドライバが異なります。使用するデバイスに応じて Synergy Configurator で表 14に示す Flash Driver を選択するようにしてください。

表 14 MCU グループごとの使用ドライバ

MCU グループ	ドライバ(Synergy Configurator 上の表記)
S7G2, S5D9	Flash Driver on r_flash_hp
S3A7	Flash Driver on r_flash_lp

6.1.2 メモリ構成、最大ブロックサイズ

図 18は S7G2、S5D9、S3A7 のメモリ構成を示しています。各デバイスでは最大ブロックサイズが異なります。

S7G2 と S5D9 のコードフラッシュメモリは 8K バイトと 32K バイトのブロックで分割されており、S3A7 では 2K バイトのブロックで分割されています。使用する MCU グループによって、サンプルソフトにおける受信バッファサイズや書き込むメモリ領域サイズの設定が異なりますので留意してください。

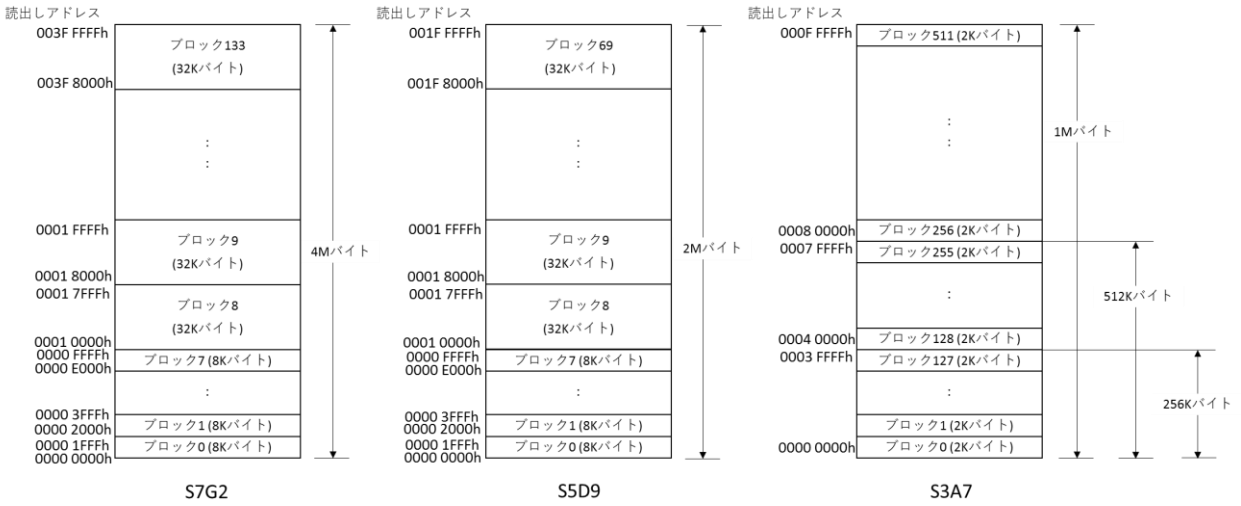


図 18 製品グループごとのフラッシュメモリ構成

6.2 BTFLG のアドレス

S7G2, S5D9, S3A7 では、BTFLG の配置されているメモリの番地が異なります。各 MCU グループの BTFLG が配置されているレジスタアドレスを表 15に示します。S7G2 ではデータフラッシュの領域に BTFLG をセットするレジスタが配置されていますが、S5D9 と S3A7 はコードフラッシュに配置されています。

表 15 MCU グループごとの BTFLG 配置アドレス

MCU グループ	レジスタ名	アドレス	フラッシュメモリの区分
S7G2	AWS	0x4012 0064h	データフラッシュ
S5D9	AWS	0x0100 A164h	コードフラッシュ
S3A7	AWSC	0x0101 0008h	コードフラッシュ

S5D9 と S3A7 において、startupAreaSelect()関数による BTFLG 書換はコードフラッシュへアクセスするため、事前に書き込み前に割り込み禁止設定にする必要があります。書き込み中に割り込み等でコードフラッシュへのアクセスが発生した場合、読みだすことが出来ずにプログラムが暴走するのを防ぐためです。

フラッシュローダーのソースコード上では、flupdate_finalize()関数の実行前に割り込み禁止の設定を行っています。アプリケーションのソースコード上では、”SWAP”コマンドを受け付けた後に呼び出される SwapBootFlag()関数の実行前に割り込み禁止の設定を行っています。

6.3 VBATT バックアップレジスタ(VBTBKRn)へのアクセス方法

更新モードフラグを格納する VBATT バックアップレジスタ VBTBKRn はレジスタライトプロテクションの保護領域となっています。通常はレジスタライトプロテクション機能が有効になっており、解除することで書き込みが可能です。解除するにはプロテクトレジスタ(PRCR)の PRC1 ビットを 1 にします。

VBTBKRn へのアクセスは MCU グループによって手順が異なります。ご使用の MCU に応じて表 16に示す手順でアクセスしてください。

表 16 MCU グループごとのバックアップレジスタへのアクセス手順

MCU グループ	アクセス手順
S7G2, S5D9	特別な手順無く R_SYSTEM->VBTBKR[n](n = 0 ~ 511)へ 8 ビット単位で読出し/書き込みが可能です。
S3A7	1. パワーオンリセット後、VBTCR1.BPWSWSTP ビットを 1 にします。 2. VBTSR.VBTRVLD ビットが 1 になるのを待ちます。 3. R_SYSTEM->VBTBKR[n](n = 0 ~ 511)に対して 8 ビット単位で読出し/書き込みが可能になります。 4. VBTCR1.BPWSSTP ビットを 0 にクリアします。

6.4 MCU 依存のソースコードについて

サンプルプログラムのソースコードには、これまでに示した各 MCU グループごとのレジスタアドレスやメモリ構成の差分などを反映させた依存コードがあります。これらは #if 文でマクロ定義や処理の分岐を行っています。

お使いのデバイスにあわせて、ソースコード内の#if defined(BSP_MCU_GROUP_{デバイス名})の定義文を参照してください。

7. 動作サンプルの作成手順(e2studio)

本アプリケーションノートに付属の動作サンプルの作成手順について説明します。ここでは、表 17において「入力」で示すファイルから「出力」で示すファイルを生成します。

表 17 動作サンプル関連ファイル

入力／出力	ファイル名	用途
入力	Rewrite_App_Blinky_{ボード名}/	Blinky アプリケーション 1, 2, 3 の作成用プロジェクト
	Rewrite_App_HelloWorld_{ボード名}/	Hello World アプリケーションの作成用プロジェクト (S7G2,S5D9 のみ)
	Rewrite_App_WeatherPanel_{ボード名}/	Weather Panel アプリケーションの作成用プロジェクト(S7G2 のみ)
	Rewrite_App_FlashLoader.zip	フラッシュローダー作成用プロジェクト
出力	Rewrite_App_Blinky_{ボード名}_1.srec	アプリケーション Blinky1（LED1 点滅）
	Rewrite_App_Blinky_{ボード名}_2.srec	アプリケーション Blinky2（LED2 点滅）
	Rewrite_App_Blinky_{ボード名}_3.srec	アプリケーション Blinky3（LED3 点滅）
	Rewrite_App_HelloWorld_{ボード名}.srec	アプリケーション Hello World
	Rewrite_App_WeatherPanel_{ボード名}.srec	アプリケーション Weather Panel
	Rewrite_FlashLoader__{ボード名}.srec	フラッシュローダー
	Error_Checksum.srec	チェックサムエラー動作確認用データ

ここでは、実験用のアプリケーションとして、「Blinky アプリケーション」、「HelloWorld アプリケーション」、「WeatherPanel アプリケーション」をビルドし、動作確認用フォルダ（以下、work_demo）にコピーします。ホスト PC が Widows 10 の場合、ボードが”Synergy USB Boot”と認識されてドライバが正常に動作しない場合があります。この時は、各プロジェクトにおいて、Communications Framework on sf_el_ux_comms にある USBX Device Configuration において、Product ID を 0x0000 に設定してください。

7.1 Blinky アプリケーション

以下の手順で点滅する LED が異なる 3 つの Blinky アプリケーションを作成します。

1. e² studio において、「ファイル」→「インポート」を選択します。
2. 「選択」ダイアログにおいて、「既存プロジェクトをワークスペースへ」を選択し、「次へ」ボタンを押します。
3. 「プロジェクトのインポート」ダイアログにおいて、「アーカイブ・ファイルの選択」に Rewrite_App_Blinky_{ボード名}を選択し、「終了」ボタンを押します。
4. インポートされたプロジェクトにおいて、src/blinky_thread_entry.c を開き、LED_BLINK を 1 に設定してプロジェクトをビルドします。

5. Debug フォルダに生成された Rewrite_App_Blinky_{ボード名}_SSP1.3.3.srec のファイル名を、Rewrite_App_Blinky_{ボード名}_1.srec に変更します。
6. 上記の 4, 5 に示した要領で、LED_BLINK を 2 に設定し、Rewrite_App_Blinky_{ボード名}_ 2.srec を生成します。
7. 上記の 4, 5 に示した要領で、LED_BLINK を 3 に設定し、Rewrite_App_Blinky_{ボード名}_ 3.srec を生成します。
8. 得られた 3 つの srec ファイルを work_demo フォルダにコピーします。

7.2 HelloWorld アプリケーション

7.1に示した要領で Rewrite_App_HelloWorld_{ボード名}をインポート、ビルドし、得られた Rewrite_App_HelloWorld_{ボード名}.srec を work_demo フォルダにコピーします。HelloWorld アプリケーションは SK-S7G2, PK-S5D9 で動作します。

7.3 WeatherPanel アプリケーション

7.1に示した要領で Rewrite_App_WeatherPanel_{ボード名}をインポート、ビルドし、得られた Rewrite_App_WeatherPanel_{ボード名}.srec を work_demo フォルダにコピーします。スプラッシュスクリーンの表示時間を短くするため、SplashScreenEventHandler 関数内の gx_system_timer_start 関数コールにおいて、第３引数の値を小さい値に変更しています。WetherPanel アプリケーションは SK-S7G2 のみにおいて動作します。

7.4 フラッシュローダー

7.1に示した要領で Rewrite_FlashLoader_{ボード名}をインポート、ビルドし、得られた Rewrite_FlashLoader_{ボード名}.srec を work_demo フォルダにコピーします。また、フォルダ内の Initial_BTFLG_{デバイス名}.srec も同フォルダにコピーします。

7.5 デバッグ

9で述べるホスト PC の更新処理制御アプリケーションを用いて、7.1～7.3で作成した各アプリケーションと7.4で作成したフラッシュローダーを組み合わせた動作のデバッグを行う場合、図 19で示すように、フラッシュローダーのプログラム（Rewrite_FlashLoader_{ボード名}.elf）を最初にロードした後、アプリケーションのプログラムをダウンロードするように「イメージとシンボルをロード」で設定するとデバッグが可能です。この時、BTFLG=1 である必要があるので、BTFLG=0 の場合は Initial_BTFLG_{デバイス名}.srec の値を RFP で書き込むなどして、BTFLG=1 に設定した後にデバッグを行ってください。

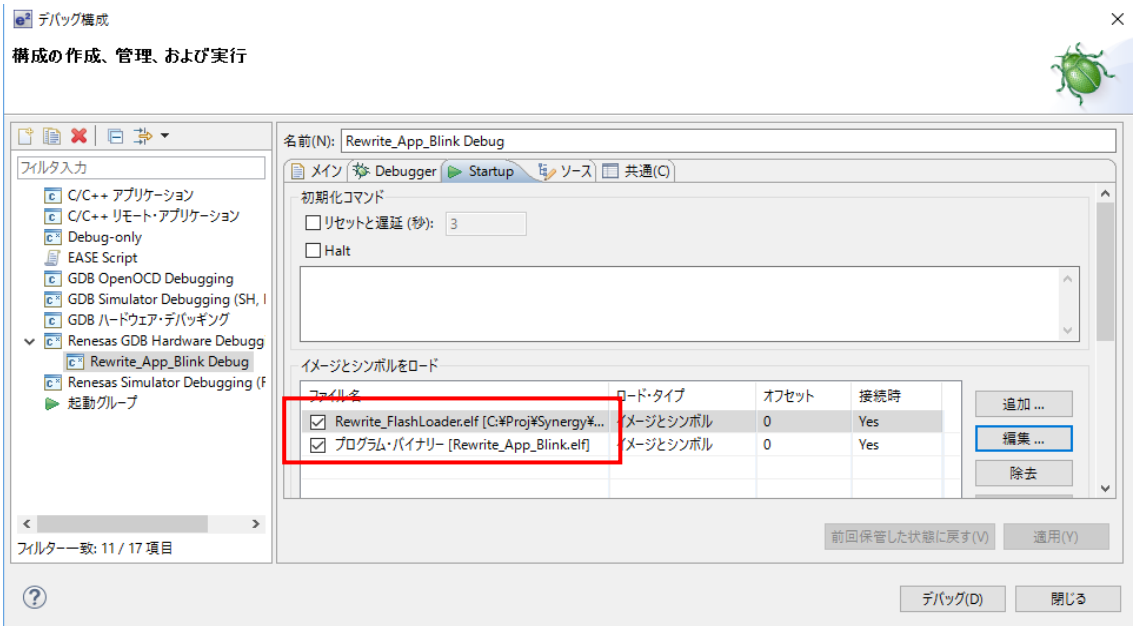


図 19 デバッグ時のイメージのロード

7.6 チェックサムエラー検証ファイル

任意のアプリケーションの更新データファイルにおいて、意図的にチェックサムエラーとなるデータに改変したファイルを用意します。本アプリケーションノートでは、Rewrite_App_Blinky_{ボード名}_1 .srec の 2500 行目のチェックサムを改変し、Error_Checksum.srec を作成しています。

8. 動作サンプルの作成手順(IAR EWARM for Synery)

IAR EWARM for Synergy における実験用アプリケーションの動作確認手順を示します。「Blinky アプリケーション」、「HelloWorld アプリケーション」、「WeatherPanel アプリケーション」をビルドし、動作確認用フォルダ（以下、work_demo）にコピーします。ホスト PC が Widows 10 の場合、ボードが”Synergy USB Boot”と認識されてドライバが正常に動作しない場合があります。この時は、各プロジェクトにおいて SSC を起動し、Communications Framework on sf_el_ux_comms にある USBX Device Configuration にて、Product ID を 0x0000 に設定してください。

8.1 Blinky アプリケーション

以下の手順で点滅する LED が異なる 3 つの Blinky アプリケーションを作成します。

1. EWARM for Synergy において、「ファイル」→「開く」→「ワークスペース」を選択します。
2. ファイル選択おいて、でワークスペースファイル 「Rewrite_{ボード名}.eww」 選択し、「開く」ボタン押します。
3. インポートされたワークスペースにおいて、Rewrite_App_Blinky_{ボード名}プロジェクトがアクティブになっていることを確認します。アクティブでない場合プロジェクト名で「右クリック」→「アクティブに設定」をクリックします。
4. ワークスペースウィンドウにおいて、Rewrite_App_Blinky_{ボード名}プロジェクト名を「右クリック」→「オプション」を選択。表示されたオプションダイアログにおいて、「出力コンバータ」カテゴリをクリック→「追加出力ファイルを生成」にチェックし、ビルド時に srec ファイルが生成されるようにします。この際、図 20に示すように出力フォーマットが Motorola になっていることを確認します。
5. メニューバー「Renesas Synergy」→「コンフィグレータ」をクリックして Synergy Standalone Configrator(SSC)起動します。起動後に「Generate Project Content」をクリックし SSP のソースコードを生成します。生成後、SSC を終了します。
6. ワークスペースウィンドウにおいて、synergy/ssp/bsp/cmsis/Device/RENESAS/{デバイス名}/Source/に配置されている startup_{デバイス名}.c、 system_{デバイス名}.ファイルをそれぞれ「右クリック」→「オプション」を選択。開いたオプションダイアログにおいて、「ビルドから除外」にチェックします。
7. インポートされたプロジェクトにおいて、src/blinky_thread_entry.c を開き、LED_BLINK を 1 に設定してプロジェクトをビルドします。
8. Debug/Exe フォルダに生成された Rewrite_App_Blinky_SK_S7G2_SSP1.3.3.srec のファイル名を Rewrite_App_Blinky_{ボード名}_ 1.srec に変更します。
9. 上記の 7, 8 に示した要領で、LED_BLINK を 2 に設定し、Rewrite_App_Blinky_{ボード名}_2.srec を生成します。
10. 上記の 7, 8 に示した要領で、LED_BLINK を 3 に設定し、Rewrite_App_Blinky_{ボード名}_ 3.srec を生成します。
11. 得られた 3 つの srec ファイルを work_demo フォルダにコピーします。

8.2 HelloWorld アプリケーション

8.1に示した手順に加え、ビルド前に以下の手順を実行します。液晶への描画を行っている為、専用のライブラリコード生成が必要です。ライブラリコードの生成には GUIX Studio がインストールされている必要があります。GUIX Studio はギャラリーからダウンロード可能です。

1. ワークスペースウィンドウの src/フォルダを「右クリック」→「追加(A)」→「ファイルの追加(F)」を選択します。

2. プロジェクト名/guix_studio/guiapp.gxp を選択し「開く」クリックします。該当ファイルが表示されない場合、「全てのファイル」を選択し全拡張子を見えるようにしてください。
3. メニューバー「ツール」→「ビューアの設定(V)…」をクリックします。
4. 「ビューアの設定」ダイアログにおいて、「新規作成(N)」をクリックします。
5. 「ビューア拡張子の編集」ダイアログにおいて、ファイル拡張子(F) : に「.gxp」を入力し「ok」をクリックします。アクション(A)は「エクスプローラの関連付けを使用」を選択します。
6. 上記の 2 で追加した guiapp.gxp をダブルクリックして GUIX Studio を起動します。
7. GUIX Studio 起動後、メニューバーの「Configure」→「Project/Displays」をクリックします
8. 「Configure Project」ダイアログにおいて、「ToolChain」のプルダウンで「IAR」を選択し Save をクリックします。
9. メニューバーの「Project」→「Generate All Output Files」をクリックし、「Select Export Resources」ダイアログが表示されたらそのまま Generate をクリックしコードを生成します。
10. 「All Output Files have been updated」が表示されたら GUIX Studio を終了します。

上記と 7.1.1 に示した手順を行った後に Rewrite_App_HelloWorld_{ボード名}をビルドし、得られた Rewrite_App_HelloWorld_{ボード名}.srec を work_demo フォルダにコピーします。HelloWorld アプリケーションは SK-S7G2, PK-S5D9 で動作します。

8.3 WeatherPanel アプリケーション

8.1、8.2に示した要領で Rewrite_App_WeatherPanel_{ボード名}をビルドし、得られた Rewrite_App_WeatherPanel_{ボード名}.srec を work_demo フォルダにコピーします。スプラッシュスクリーンの表示時間を短くするため、SplashScreenEventHandler 関数内の gx_system_timer_start 関数コールにおいて、第 3 引数の値を小さい値に変更しています。WetherPanel アプリケーションは SK-S7G2 のみにおいて動作します。

8.4 フラッシュローダー

8.1に示した要領で Rewrite_FlashLoader_{ボード名}をビルドし、得られた Rewrite_FlashLoader_{ボード名}_SSP1.3.3.srec を work_demo フォルダにコピーします。また、同 zip ファイル内の Initial_BTFLG_{デバイス名}.srec も同フォルダにコピーします。

尚、フラッシュローダーは自動生成されたコードを改変なくそのまま使用しているため、8.1の手順 6 で示したビルドから除外する作業は不要です。

8.5 デバッグ

9で述べるホスト PC の更新処理制御アプリケーションを用いて、8.1～8.3で作成した各アプリケーションと8.4で作成したフラッシュローダーを組み合わせた動作のデバッグを行う場合、以下に示す手順で準備を行うとデバッグが可能です。

1. Blinky, HelloWorld, WeatherPanel のいずれかのアプリケーションプロジェクトにおいて、プロジェクト名で「右クリック」→オプションを選択します。
2. オプションダイアログにおいて「リンカ」カテゴリをクリック。「ライブラリ」タブにおいて、「デフォルトのプログラムエントリをオーバーライドする」にチェックし、エントリシンボルに「boot_entry」を入力。
3. オプションダイアログにおいて「デバッガ」カテゴリをクリック。「設定」タブにおいて、「指定位置まで実行」のチェックを外し、main 関数より以前の処理のデバッグを行えるようにします。

4. 同じく「デバッガ」カテゴリにおいて、「イメージ」タブをクリック。図 20に示すように、「追加イメージのダウンロード」のいずれかにチェックを入れ、パスにフラッシュローダーのプログラム（Rewrite_FlashLoader_{ボード名}.out）を指定します。オフセットに「0」を入力します。
5. 設定をしたアプリケーションプロジェクトをアクティブにし、ビルド、ダウンロード、実行をします。

デバッグの際、BTFLG=1 である必要があるので、BTFLG=0 の場合は Initial_BTFLG_{デバイス名}.srec の値を RFP で書き込むなどして、BTFLG=1 に設定した後にデバッグを行ってください。

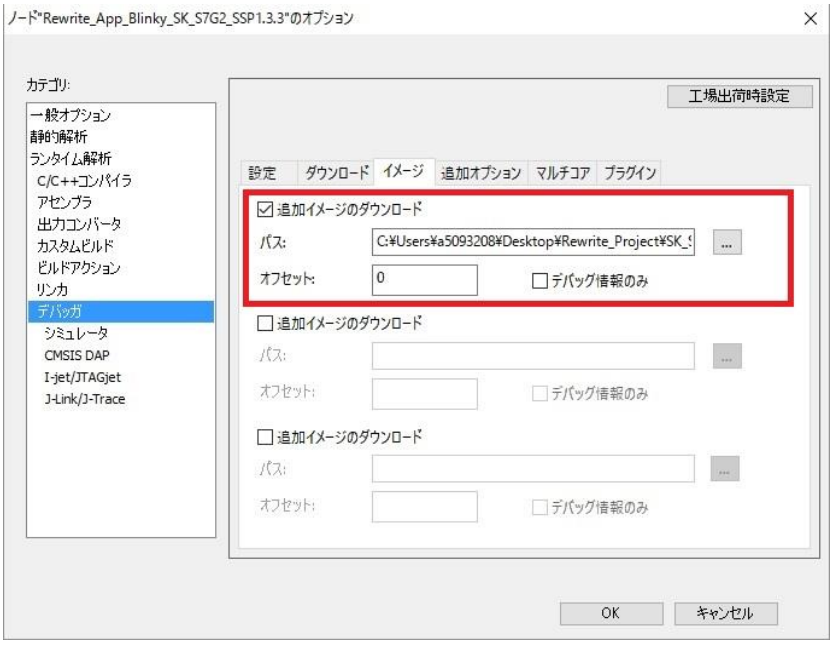


図 20 デバッグ時のイメージのロード(EWARM for Synergy)

9. ホスト PC の更新処理制御アプリケーション

本アプリケーションノートでは、デバイスと更新データや応答の送受信を行うアプリケーションを、Tera Term のマクロ言語 Tera Term Language(TTL)でその動作を記述します(FlashProgrammer.ttl)。この動作フローの概要を図 21に示します。

このマクロでは、新しいアプリケーションのデータを srec ファイルから 1 行ずつデータを読出しデバイスに送信します。また、デバイスから通知されるバッファサイズに従い、NEW ブロックコマンドでそのバッファの区切りをデバイス側に通知します。この時、1 行のデータの中にバッファを跨ぐデータが存在する場合、このデータをそのまま送信し、その後で NEW ブロックコマンドを送信すると、その送信したデータの内、バッファ境界の後のデータはバッファに正常に格納されないため、フラッシュメモリに書き込まれません。このため、そのようなデータの場合は、バッファ境界の前のデータ、NEW ブロックコマンド、バッファ境界の後のデータに送信を分けて処理を行います。

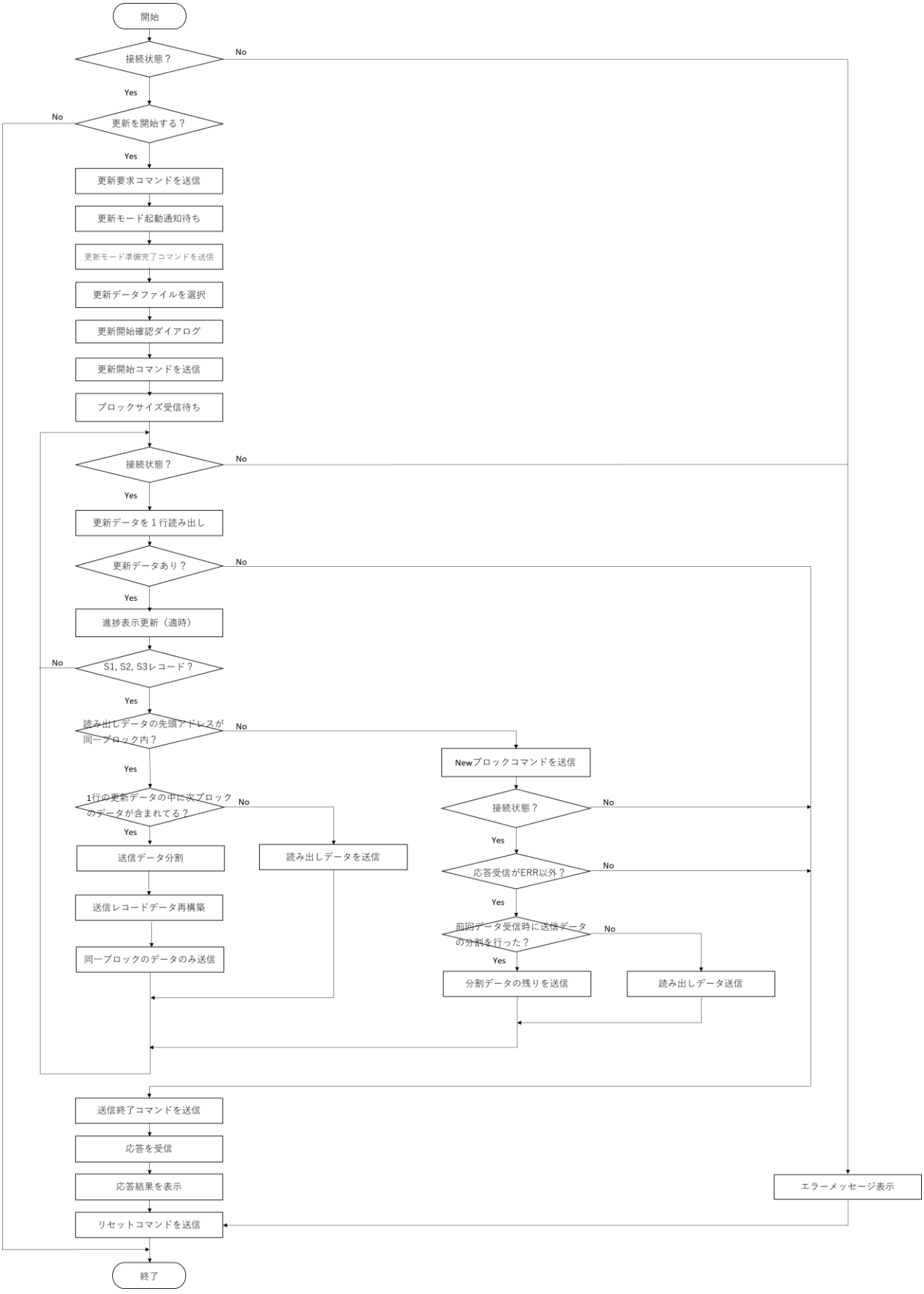


図 21 ホスト PC 側更新アプリケーションのフロー

10. 内蔵フラッシュ書き換えの動作確認

10.1 出荷イメージデータの書き込み

7、または、8で生成したアプリケーション、フラッシュローダー、および、BTFLG 初期化用データを、RFP を用いて以下に示す手順でデバイスに書き込みます。

1. ボードのジャンパ、スイッチを切替え USBboot モードで起動するように変更します
   SK-S7G2 ボード：J1 を 2-3 ショートに設定します。
   PK-S5D9 ボード：J1 を 2-3 ショートに設定します。
   DK-S3A7 ボード：S5 の BOOT スイッチを ON にします。
2. ボードとホスト PC を USB ケーブルで接続します。
   SK-S7G2 ボード：J5 とホスト PC を USB ケーブルで接続します。
   PK-S5D9 ボード：J5 とホスト PC を USB ケーブルで接続します。
   DK-S3A7 ボード：J2 とホスト PC を USB ケーブルで接続します。
3. ボードの電源端子にケーブルを接続し、ボードへ電源を投入します。
   SK-S7G2 ボード：J19 にケーブルを接続します。
   PK-S5D9 ボード：J19 にケーブルを接続します。
   DK-S3A7 ボード：J15 にケーブルを接続します。
4. ボードを一度リセットします。
   SK-S7G2 ボード：J2 ショート後、リリース
   PK-S5D9 ボード：J2 ショート後、リリース
   DK-S3A7 ボード：S7 押下後、リリース
5. ホスト PC で RFP を起動します。
6. 「ファイル」-「新しいプロジェクトの作成」を選択し、以下を設定します。
   「マイクロコントローラ」=Synergy
   「プロジェクト名」=任意
   「ツール詳細」=適切な COM
7. 「接続」ボタンを押します。
   以下のエラー・メッセージが表示された場合は、ボードをリセットした後に、再度、「接続」ボタンを押します。
   エラー(E3000105): デバイスから応答がありません。
8. 「プログラムファイル」に、同一フォルダ内にある Initial_BTFLG.srec 、Rewrite_App_Blinky1.srec、Rewrite_FlashLoader.srec を設定します。（「プログラムファイル」に複数ファイルを選択する方法については、RFP のユーザーズマニュアルを参照してください。）
9. ボードをリセットした後、[スタート]ボタンを押します。
   ここでベリファイに失敗した場合、書き込み開始時に BTFLG=0 となっていたためにベリファイに失敗している可能性があります。この場合は、もう一度、書き込みを行うことで正常に書き込みとベリファイが完了します。
10. 書き込み完了後、RFP を終了し、ボードのジャンパ、スイッチを戻しシングルチップモードで起動するように変更します。
11. ボードを再起動し、工場出荷イメージに設定したアプリケーションが動作することを確認します。
    SK-S7G2 ボード：J1 を 1-2 ショートに設定します。
    PK-S5D9 ボード：J1 を 1-2 ショートに設定します。
    DK-S3A7 ボード：S5 の BOOT スイッチを OFF にします。

10.2 動作確認（正常動作）

10.1で用意した出荷イメージデータ書き込み済みデバイスに対して、ホスト PC から更新データを送信し、内蔵フラッシュのアプリケーションを更新する動作確認を行います。ここでは、表 18に示すシナリオでアプリケーションの更新を行います。

表 18　内蔵フラッシュ書き換え動作確認シナリオ(S7G2 の例)

	アプリケーション保存領域 0	アプリケーション保存領域 1
出荷時	Blinky1 *	（未使用）
Blinky2 に更新	Blinky1	Blinky2 *
Blinky3 に更新	Blinky3 *	Blinky2
HelloWorld に更新	Blinky3	HelloWorld *
WeatherPanel に更新	WeatherPanel *	HelloWorld

*：起動する側のアプリケーション

上記の内蔵フラッシュ書き換えを行う手順を以下に示します。

1. ボードとホスト PC を USB ケーブルで接続します。
   SK-S7G2：J5 とホスト PC を USB ケーブルで接続します。
   PK-S5D9：J5 とホスト PC を USB ケーブルで接続します。
   DK-S3A7：J2 とホスト PC を USB ケーブルで接続します。
2. ケーブルを接続し、ボードに電源を投入します。Windows 10 のホスト PC においてドライバが正常に動作しない場合、7、または、8に記載されている内容に従って、USBX Device Configuration を変更したプロジェクトでビルドし直してください。
   SK-S7G2 ボード：J5 とホスト PC を USB ケーブルで接続します。
   PK-S5D9 ボード：J5 とホスト PC を USB ケーブルで接続します。
   DK-S3A7 ボード：J2 とホスト PC を USB ケーブルで接続します。
3. ホスト PC で Tera Term を起動し、「設定」－「端末」－「改行コード」において以下に設定します。
   「受信」＝AUTO
   「送信」＝CR
4. 同 TeraTerm において、「コントロール」－「マクロ」において、FlashProgrammer.ttl を選択します。
5. 「更新を開始しますか？」に対して、「はい」を選択します。
   この時、SK-S7G2, PKS5D9 では LED3（橙）が点滅します。DK-S3A7 では LED3(緑)が点滅します。
6. ファイル選択ダイアログが表示されるので、Blinky2 のファイルを選択し、「開く」ボタンを押します。
7. 「更新を開始します。」が表示されるので、「OK」ボタンを押します。
   更新が開始され、デバイスでは LED2（赤）が点滅し、ホスト PC では進捗を示すダイアログが表示されます。
8. 更新が成功すると、デバイスでは LED1（緑）が点滅します。また、ホスト PC では「更新成功　リセットして再起動します。」が表示されるので、「OK」ボタンを押します。
   Blinky2 アプリケーションでデバイスが再起動します。
9. 「もう一度、更新を行いますか？」に対して「はい」を選択し、手順 6 に戻ります。
   手順 6 から手順 9 までを、更新データファイルを Blinky3、HelloWorld、WeatherPanel の順に変えながら動作を確認します。
10. 「もう一度、更新を行いますか？」に対して「いいえ」を選択すると更新処理を終了します。

10.3　動作確認（通信切断）

更新処理中に通信ケーブルが切断された場合、更新処理前に動作していたアプリケーションで問題なく再起動する動作を確認します。

1. 動作確認（正常動作）で述べた要領でアプリケーションの更新を行います。
2. 手順 7 の実行時、更新処理中の状態で J5 に接続されている USB ケーブルを外します。
3. ホスト PC のフラッシュローダーでエラーを示すダイアログが表示されます。
4. デバイスでは、LED1, LED2, LED3 が同時に点滅してエラーの発生を示します。
5. デバイスは約 5 秒後に再起動し、更新処理前に動作していたアプリケーションが動作します。

10.4 動作確認（デバイス電源断）

更新処理中に電源ケーブルが切断された場合、次の起動時には、更新処理前に動作していたアプリケーションで問題なく動作することを確認します。

1. 動作確認（正常動作）で述べた要領でアプリケーションの更新を行います。
2. 手順 7 の実行時、更新処理中の状態で J19 に接続されている USB ケーブルを外します。
3. ホスト PC のフラッシュローダーでエラーを示すダイアログが表示されます。
4. 外した USB ケーブルを接続してデバイスを起動すると、更新処理前に動作していたアプリケーションが動作します。

10.5 動作確認（異常データ受信）

更新データの受信時に何らかの原因で異常データを受信した場合、次の起動時には、更新処理前に動作していたアプリケーションで問題なく動作することを確認します。

1. 動作確認（正常動作）で述べた要領でアプリケーションの更新を行います。
2. 手順 6 の更新データファイル選択時、チェックサムエラーのデータを持つ更新データファイルを選択し、更新処理を続けます。
3. ホスト PC のフラッシュローダーでエラーを示すダイアログが表示されます。
4. デバイスでは、LED2（赤）と LED3（橙）が交互に点滅し、更新失敗を示します。
5. ホスト PC の再起動確認ダイアログで「はい」を選択するとデバイスが再起動し、更新処理前に動作していたアプリケーションが動作します。

10.6 動作確認（起動アプリケーション切り替え）

両方のアプリケーション保存領域にアプリケーションが書き込まれている状態で、起動するアプリケーションを他方のアプリケーションに切り替える動作を確認します。

1. 内蔵フラッシュ書き換え処理を 1 回以上行い、両方のアプリケーション保存領域にアプリケーションが書き込まれている状態にし、デバイスを起動します。
2. ホスト PC において TeraTerm を起動し、マクロファイルとして SwapBoot.ttl を選択します。
3. 「スタートアップ領域を切り替えますか？」と表示されたタイアログにおいて、「はい」を選択すると、起動していたアプリケーションとは別のアプリケーションで再起動します。

ホームページとサポート窓口

11. ルネサス エレクトロニクスホームページ
    http://japan.renesas.com/

12. お問合せ先
    http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2017-07-10	-	初版
1.01	2017-10-11	4	「1.2　評価環境」 SSP v1.3.0 への変更。 SSP のバージョンが異なる場合の注意事項を追加。
		5	参考文献 SSP v1.3.0 のユーザーズマニュアルに変更。 RFP のユーザーズマニュアルを追加。
		8	図 1 中の文字表示不備を修正。
		14	図 6 中の誤字を修正。
		23	図 12 に「DTR 信号待ち」を追加。
		26	Initial_Image.mot を削除。
		-	「6.2　出荷イメージデータの作成」を削除。（Rev. 1.00 の 27 ページ）
		29	「8.1　出荷イメージデータの書き込み」 「6.2　出荷イメージデータの作成」削除に伴う文言の変更、および、手順 7 のプログラムファイルの選択を複数プログラムファイル選択方法に変更。
		26 29	「6.1 アプリケーションの生成」、「8.2 動作確認（正常動作）」に、ホスト PC が Widows 10 の場合の注意事項を追加。
1.02	2018-05-22	1	動作確認デバイスに PK-S5D9, DK-S3A7 を追加。 仕様機能 の HAL Driver に r_flash_lp を追加。
		4	「1.2 評価環境」に IAR EWARM for Synergy を追加。 「1.3 本アプリケーションノートの動作確認」に PK-S5D9, DK-S3A7 を追加。 「1.4 留意事項」に FSPR 書換えに関する注意事項と、S3A7 における RFP 使用時の留意事項を追加。
		5	「1.5 用語」 更新モードフラグの格納先の記述を「データフラッシュ」から「VBATT バックアップレジスタ(VBTBKRn)」に変更 「1.6 参考文献」 SSP 1.3.3 のユーザーズマニュアルに変更。 S5D9 グループのユーザーズマニュアルを追加。 S3A7 グループのユーザーズマニュアルを追加。 PK-S5D9 ボードのユーザーズマニュアルを追加。 DK-S3A7 ボードのユーザーズマニュアルを追加。 Synergy boot firmware の仕様(R01AN3280EU0110)を追加 [4] SK-S7G2 Schematic v3.0 (January 06, 2016)を削除
		6	「2.2 ハードウェア構成」 表 2 に PK-S5D9, DK-S3A7 の情報を追加
		7	「2.3 メモリマッピング」S5D9 と S3A7 のメモリマッピングを表 3, 表 4 として追加
		9 10 11 12	「3.Startup Area Slect と MMF を利用したアプリケーション更新の動作概要」 S5D9 と S3A7 の事例を追加し、それぞれのメモリマップ図 3, 図 4 を追加。
		13	「3.3 起動処理のシーケンス」S5D9 と S3A7 における起動処理のシーケンスの差分を追記。
		16 17	「ホスト PC 間通信プロトコル」S5D9, S3A7 における差分を追記。
		17	「3.5 更新モードフラグの書き換え」の章を追加

Rev.	発行日	改訂内容	
		ページ	ポイント
1.02	2018-05-22	18	「4.アプリケーションへの機能追加」S7G2, S5D9, S3A7 における設定の差を追記 「4.2 Flash Loader の追加」章の名前を変更。Flash Driver 追加理由を追加
		18 19	「4.1.1 Flash Driver on r_flash_hp の追加(S7G2, S5D9)」の章を追加
		19	「4.1.2 Flash Driver on r_flash_lp の追加(S3A7)」の章を追加
		20 21	「4.4.1 ブート処理の変更」S5D9, S3A7 での変更ファイルのロケーションを追加
		21	「4.5 リンカスクリプト」 S5D9, S3A7 のメモリリージョン 表 9, 表 10 を追加
		22 23 24 25	「4.5 リンカスクリプト」 IAR EWARM for Synergy 環境におけるリンカスクリプトコード 2, コード 3, 並びにリンカスクリプトの解説を追加。
		26	「4.6 初期値データの RAM コピー元設定(EWARM for Synergy」の章を追加。
		28	「5.2 フラッシュローダーの構成」表 11 に PK-S5D9 ボードと DK-S3A7 ボードの動作状態を追加。
		31 33	「5.3.3 Flash Memory Write Thread」コンフィグレータの設定項目の解説と、図 17 を追加
		34 35	「5.3.4 リンカスクリプト」 IAR WEARM for Synergy 環境におけるリンカスクリプト コード 6 を追加。
		35 36 37	対応 MCU グループの拡張に伴い、「6 各 MCU グループおける差分」の章を新たに追加 表 14, 表 15, 表 16, 図 18 を追加。
		38	「7. 動作サンプルの作成手順(e2studio)」に章の名前を変更
		38 39 40	対応 MCU グループの拡張に伴い、各章の手順説明を S7G2, S5D9, S3A7 の各環境に対応した記述に変更
		41 42 43	「8. 動作サンプルの作成手順(IAR EWARM for Synergy)」の章を新たに追加 図 20 を追加。
		44 45	「9 ホスト PC 更新処理制御アプリケーション」 Tera Term マクロの仕様変更に伴い、マクロの動作説明を変更。フローチャート(図 21)を新しい動作仕様のものに差し替え
		46	「10.1 出荷イメージデータの書込み」 対応 MCU グループの拡張に伴い、SK-S7G2 ボード、PK-S5D9 ボード、DK-S3A7 ボードの書く環境に対応した記述に変更
		46 47	「10.02 動作確認(正常動作)」 対応 MCU グループの拡張に伴い、SK-S7G2 ボード、PK-S5D9 ボード、DK-S3A7 ボードの書く環境に対応した記述に変更

Rev.	発行日	改訂内容	
		ページ	ポイント
1.0.3	2018-08-08	2	2018 年 8 月発行のテクニカルアップデートに対する暫定対策と制限事項を追記。
1.0.4	2018-08-28	2	2018 年 8 月発行のテクニカルアップデートに対する本アプリケーションノートでの暫定対策について、補足を追記。
		5	「1.4 留意事項」 S3A7 の Issue ID 11593 による留意事項について補足を追記。
		18	「3.5 更新モードフラグの書き換え」 更新モードフラグの実装について補足を追記。
		47	「10.1 出荷イメージデータの書き込み」 リセットをすべてのボードで行うように変更。
		全体	参照番号の誤記を修正。

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。