RSK+RZT1

NOR フラッシュブートローダ

R01AN2470JG0100
Rev.1.00
2016.09.16

はじめに

本アプリケーションノートは、RSK+RZT1 に実装されている NOR フラッシュメモリの設定とプログラムを e² studio 環境で行う方法について説明します。

RSK+RZT1 の詳細については、RSK+RZT1 ユーザーズマニュアル (R20UT3551JJ0101) を参照してください。

RZ/T1 の詳細については、RZ/T1 グループユーザーズマニュアル ハードウェア編 (R01UH0483JJ0100) を参照してください。

対象製品

RZ/T1

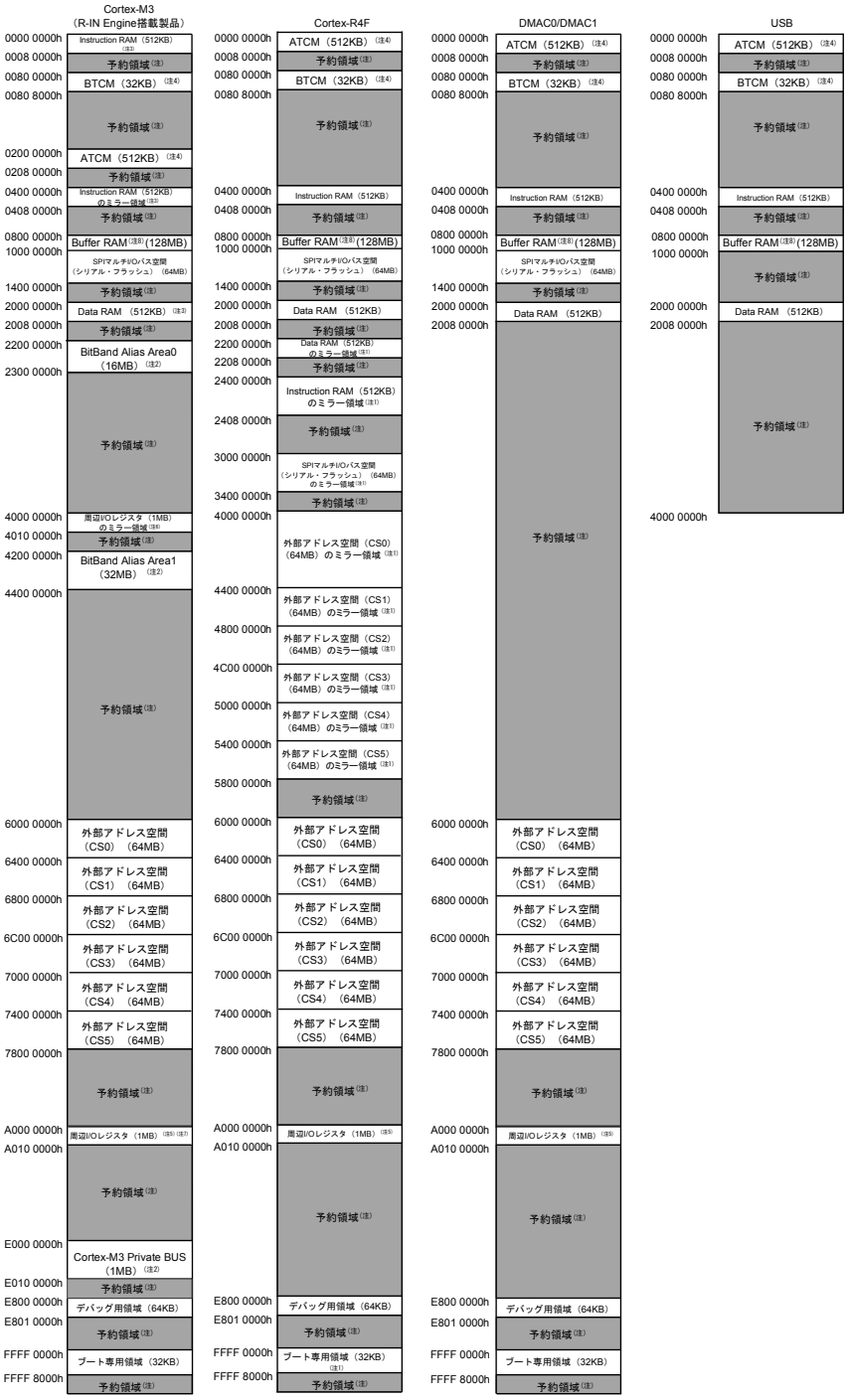
目　次

1. 概要

NOR フラッシュメモリは、書き換え可能な不揮発性メモリとして、今日の電子製品に多く用いられています。1 バイト単位でのランダムアクセスが可能であり、コンピュータがメインメモリから命令を読み出すように、NOR フラッシュメモリから機械語命令の読み出しや書き込みを行うことが可能です。RSK+RZT1 には NOR フラッシュメモリ 2 つ（U3、U4）が実装されており、RZ/T1 を外部 NOR フラッシュメモリからブートする機能の確認・評価や、外部 NOR フラッシュメモリアクセス評価が可能です。NOR フラッシュメモリからのブートは、CS0 空間に接続された U3 からのみ可能です。

1.1 アドレス空間

RZ/T1 グループのアドレス空間の詳細を**図 1.1** に示します。



注1. Cache対象（それ以外の領域は、MPUでキャッシュを有効にしないでください。）
注2. Cortex-M3からのみアクセス可能な空間
注3. Cortex-M3 SRAM空間（Extra-RAM)
注4. Cortex-R4F密結合メモリ空間
注5. 周辺空間
注6. A00E 0000h-A00F FFFFhは、予約領域
注7. A00E 0000hからA00F FFFFhの空間にはアクセスできません（400E 0000h-400FFFFFhの空間だけアクセスできます）。
注8. 詳細は、「28.3.1.3 Buffer Allocator」を参照してください。
注. 予約領域は、アクセスしないでください。

図 1.1 RZ/T1 アドレスマップ

1.2 ブートモード

RZ/T1 は 3 種類のブートモードに対応しています。

- SPI ブートモード : SPI マルチ I/O バス空間に接続されたシリアルフラッシュメモリからブート
- 16 ビットバスブートモード : CS0 空間に接続された NOR フラッシュメモリ (バス幅 16 ビット) からブート
- 32 ビットバスブートモード : CS0 空間に接続された NOR フラッシュメモリ (バス幅 32 ビット) からブート

16 ビットバスブートで NOR フラッシュメモリから RSK+RZT1 を起動する際に必要な設定は以下のとおりです。

表 1.1 ブートモード端子 SW4 の構成

SW4					
SW4.1*	SW4.2*	SW4.3*	SW4.4	SW4.5	SW4.6
ON	OFF	ON	ON	ON	OFF

注. SW4.1 は MD0 に、SW4.2 は MD1 に、SW4.3 は MD3 端子に接続されます。

ブート機能は、RZ/T1 のシステムリセットに続いて実行されるプログラムです。ローダプログラムのロードや、必要に応じてマイコン内に割り当てられたプログラムメモリ (TCM) にユーザアプリケーションプログラムをコピーし、ユーザアプリケーションプログラムに制御を引き継ぎます。このブート処理を図 1.2 に示します。

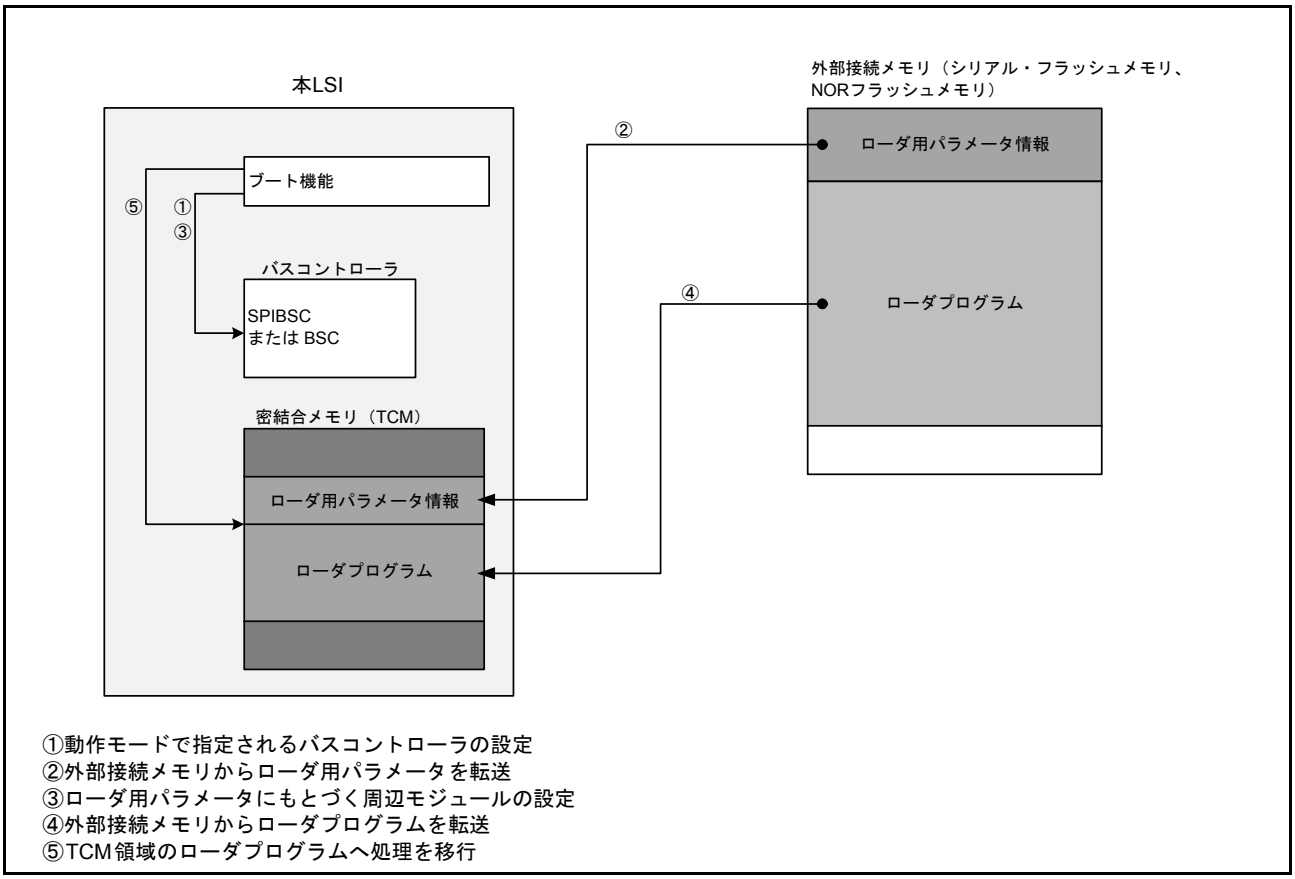
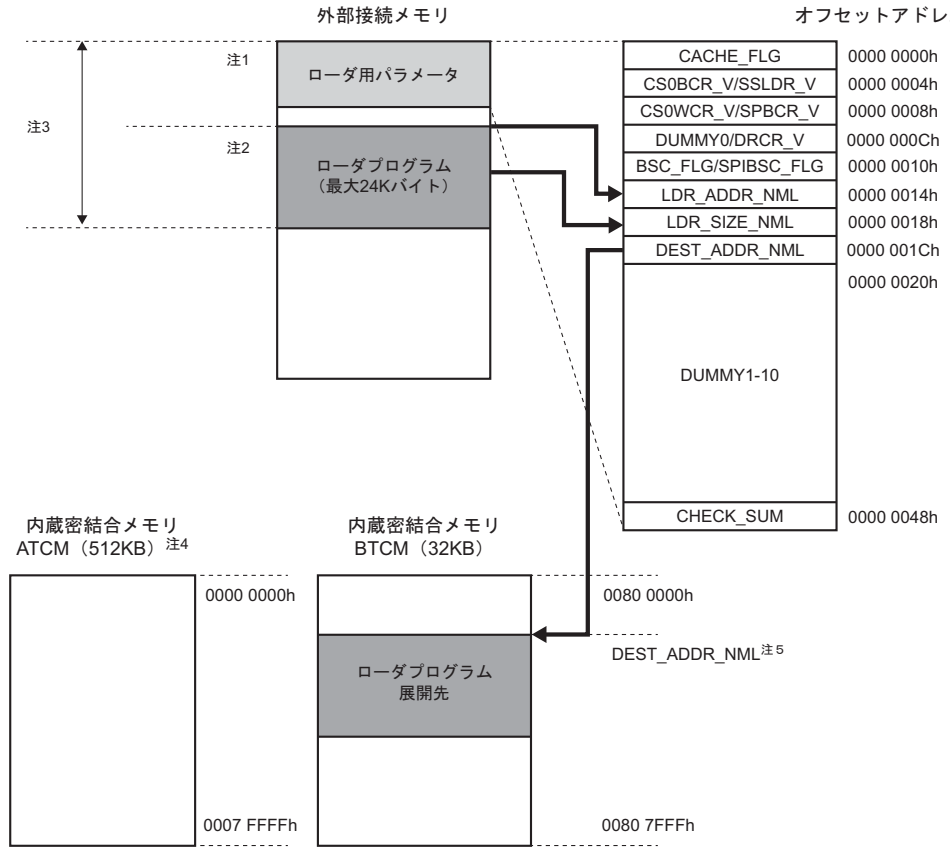


図 1.2 ブート処理の動作概要

1.3 NOR ブートにおけるローダ用パラメータ

ローダ用パラメータには、ローダプログラム情報、ブート処理を高速化するためのキャッシュ設定、バスステートコントローラ設定 (SPIBSC または BSC) などの、ユーザシステムの構成情報が含まれます。ローダ用パラメータは事前に NOR フラッシュメモリに格納しておく必要があります。RZ/T1 内の TCM と NOR フラッシュメモリ間の関係を図 1.3 に示します。TCM 領域として、ユーザアプリケーションプログラムを配置する ATCM と、ローダプログラムを配置する BTCM の 2 つが存在します。



注1. ローダ用パラメータは必ず外部アドレス空間（CS0）（4000 0000h～）に格納してください。
注2. ローダプログラムの格納アドレス（LDR_ADDR_NML）は任意です。ただしプログラムサイズ（LDR_SIZE_NML）は512バイトの倍数かつ最大24Kバイトとしてください。
注3. 16ビット／32ビットバスブートモード時はバスステートコントローラ（BSC）のアドレス出力端子（A1～A20）を使用するため4000 0000h～401F FFFFhの範囲のみを使用してください。
注4. ATCM領域はユーザアプリケーションプログラムの格納を想定しています。
注5. ローダプログラム展開先は0080 2000h～0080 7FFFhの範囲におさまるよう指定してください。

図 1.3 ローダ用パラメータ、ローダプログラムのメモリ配置

ローダ用パラメータは CS0 空間の先頭アドレス 0x40000000 から格納されます。ラベル LDR_ADDR_NML はローダプログラムの NOR フラッシュメモリ内での格納アドレスを指定し、DEST_ADDR_NML はローダプログラムをコピーして実行する際の BTCM 内での格納アドレスを指定します。LDR_ADDR_NML の設定値は 0x4000004C、DEST_ADDR_NML の設定値は 0x00802000 です。

2. NOR フラッシュブートローダ

RZ/T1 は、CS0 空間に接続された NOR フラッシュメモリからブート可能です。バス幅は 16 ビットまたは 32 ビットが選択可能です。RSK+RZT1 の NOR フラッシュメモリは 16 ビットバスで接続しています。

プロジェクト System_Boot_Loader_NOR にはローダプログラム用のソースコードが含まれます。パワーオンリセットの解除後、RZ/T1 のブート機能はハイベクタ 0xFFFF0000 からプログラムを開始します。ローダプログラム内でロウベクタ 0x00000000 に設定するため loader_init.asm ファイル内のアセンブラ関数 “set_low_vec” を呼び出して切り替えを行います。

3. NOR フラッシュメモリのプログラミング

以下の 2 通りの方法を用いて、RSK+RZT1 に実装された NOR フラッシュメモリをプログラミングします。インストーラ実行後、ソフトウェアパッケージをユーザ PC にコピーして使用してください。

- サンプルコード
  - e² studio 環境でのサンプルプログラム System_Boot_Loader_NOR を用いたプログラミング及びデバッグ
  - NOR フラッシュメモリへプログラミングするローダプログラムおよびユーザアプリケーションプログラムも同梱

- バッチファイル
  - ローダプログラムおよびユーザアプリケーションプログラムを NOR フラッシュメモリへプログラミング
  - 統合開発環境とは独立したファイル

同梱のユーザアプリケーションプログラム System_Input_Capture は、アドレス 0x00000000 から始まる ATCM 内に展開され実行されます。このユーザアプリケーションプログラムのバイナリは、RSK+RZT1 の製造時にバッチファイルを用いて NOR フラッシュメモリへ書き込みしています。

4. ユーザアプリケーションプログラムの転送

ローダプログラムは、NOR フラッシュメモリ内のユーザアプリケーションプログラムを開始アドレス 0x40020000 から検査します。検査対象は、ユーザアプリケーションプログラムの開始アドレス、終了アドレス、実行アドレス、ローダプログラムの string 変数です。この 4 項目は、ユーザアプリケーションプログラムの e² studio プロジェクトの start.asm ファイル内にあります。このファイルの一部を以下に示します。

```
    .text
    .code 32

    .global start
    .func   start
start:
    LDR pc, =reset_handler
    LDR pc, =undefined_handler
    LDR pc, =svc_handler
    LDR pc, =prefetch_handler
    LDR pc, =abort_handler
    LDR pc, =reserved_handler
    LDR pc, =irq_handler
    LDR pc, =fiq_handler
code_start:
    .word    start
code_end:
    .word    end
code_execute:
    .word    execute
    .string ".BootLoad_ValidProgramTest."
    .align 4
    .end
```

“.string” は、ローダプログラムがユーザアプリケーションプログラムを検証するためのシグネチャ変数 (signature marker) です。

“start” は、ユーザアプリケーションプログラムのベクタテーブルをロードするアドレスを指定します。ラベル “code_start” および “code_end” には、ユーザアプリケーションプログラム全体の開始アドレス、終了アドレスおよびそのベクタアドレスを指定する変数が含まれています。開始アドレスと終了アドレスは、以下のラベルを用いてユーザアプリケーションプログラムのリンカファイル (.ld) 内で指定されます。

```
ENTRY(start)
PROVIDE(end = .);
```

ラベル “code_execute” には、実行開始アドレスを示す変数 “execute” が含まれます。

“execute” ラベルはリンカファイルにも以下のように指定されます。

```
execute = .;
```

位置を示す定数や .string 変数が、上記のように start.asm ファイルのベクタテーブルの直後に存在しない場合、構成は無効とみなされます。このとき、RSK+RZT1 上の LED0 が長いパルス（2 秒）で 1 回点滅したあと、短いパルス（0.5 秒）で 1 回点滅し、エラーを知らせます。構成が有効の場合、ローダプログラムはユーザアプリケーションプログラムの開始アドレス (NOR フラッシュメモリまたは TCM のいずれか) を確認します。

リンカファイルは、RSK+RZT1 のサンプルプロジェクトごとの “complier_specific” ディレクトリに含まれています。

転送が完了すると、ユーザアプリケーションプログラムは実行アドレス 0x00000000 から実行します。

GNU リンカファイルの詳細は e² studio のヘルプ > 検索からキーワード “GNU linker” を入力して参照してください。

5. NOR フラッシュメモリへのユーザアプリケーションプログラムのロード

RSK+RZT1 で用いるユーザアプリケーションプログラムのビルド構成の初期値は ATCM に直接ロードする構成となっています。NOR フラッシュメモリへロードするためには、追加設定が必要となります。

NOR フラッシュメモリをプログラミングする前に、ユーザは、Segger デバッガを用いて初期化コマンドを発行し、NOR フラッシュメモリへのアクセスを初期化および有効化する必要があります。また、プロジェクトルートフォルダ内の該当する .jlink ファイルを手動で変更する必要があります。以下の手順で NOR フラッシュメモリのプログラミングを有効にしてください。

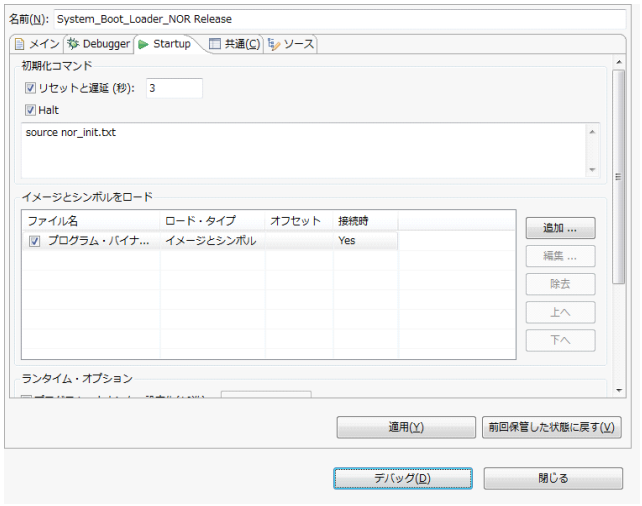
a. プロジェクトルートフォルダ内に nor_init.txt という名前でテキストファイルを作成する。
b. nor_init.txt ファイルに以下のコマンドを追加する。

```
set {unsigned long} 0xA00B0B00 = 0x0000A502
set {unsigned long} 0xA00B0308 = 0x00007E7E
set {unsigned long} 0xA00B0B00 = 0x0000A500
set {unsigned char} 0xA00002FF = 0x00
set $DUMMY = *(0xA00002FF)
printf "DUMMY_READ = 0x%X", $DUMMY
set {unsigned char} 0xA00002FF = 0x40
set $DUMMY = *(0xA00002FF)
printf "DUMMY_READ = 0x%X", $DUMMY
set {unsigned char} 0xA0000200 = 0x22
set {unsigned char} 0xA0000201 = 0x22
set {unsigned char} 0xA0000202 = 0x22
set {unsigned char} 0xA0000203 = 0x22
set {unsigned char} 0xA0000204 = 0x22
set {unsigned char} 0xA0000205 = 0x22
set {unsigned char} 0xA0000206 = 0x22
set {unsigned char} 0xA0000207 = 0x22
set {unsigned char} 0xA0000270 = 0x22
set {unsigned char} 0xA0000271 = 0x22
set {unsigned char} 0xA0000272 = 0x22
set {unsigned char} 0xA0000273 = 0x22
set {unsigned char} 0xA0000274 = 0x22
set {unsigned char} 0xA0000275 = 0x22
set {unsigned char} 0xA0000276 = 0x22
set {unsigned char} 0xA0000277 = 0x22
set {unsigned char} 0xA0000280 = 0x22
set {unsigned char} 0xA0000281 = 0x22
set {unsigned char} 0xA0000282 = 0x22
set {unsigned char} 0xA0000283 = 0x22
set {unsigned char} 0xA0000284 = 0x22
set {unsigned char} 0xA0000285 = 0x22
set {unsigned char} 0xA0000286 = 0x22
set {unsigned char} 0xA0000287 = 0x22
set {unsigned char} 0xA0000288 = 0x22
set {unsigned char} 0xA0000289 = 0x22
set {unsigned char} 0xA000028A = 0x22
set {unsigned char} 0xA000028B = 0x22
set {unsigned char} 0xA000028C = 0x22
set {unsigned char} 0xA000028D = 0x22
set {unsigned char} 0xA000028E = 0x22
set {unsigned char} 0xA000028F = 0x22
set {unsigned char} 0xA0000210 = 0x22
```

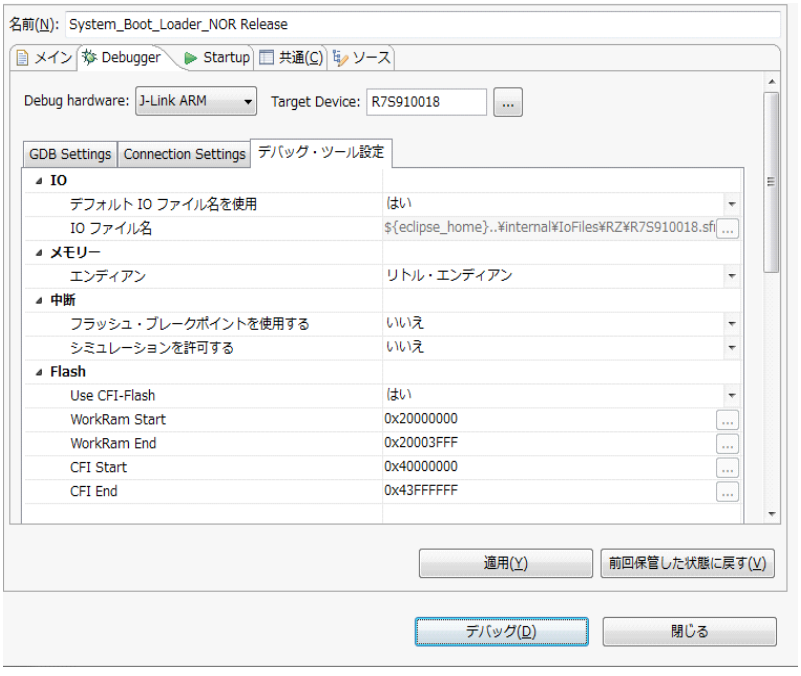
```
set {unsigned char} 0xA0000215 = 0x22
set {unsigned char} 0xA0000216 = 0x22
set {unsigned char} 0xA0000217 = 0x22
set {unsigned char} 0xA00002D6 = 0x23
set {unsigned char} 0xA00002D7 = 0x23
set {unsigned char} 0xA000029A = 0x23
set {unsigned char} 0xA000029B = 0x23
set {unsigned char} 0xA000024F = 0x23
set {unsigned char} 0xA0000211 = 0x22
set {unsigned char} 0xA0000212 = 0x22
set {unsigned char} 0xA0000269 = 0x23
set {unsigned char} 0xA000021E = 0x22
set {unsigned char} 0xA0000083 = 0x58
set {unsigned char} 0xA000008D = 0x02
set {unsigned char} 0xA0000080 = 0xFF
set {unsigned char} 0xA000008E = 0xFF
set {unsigned char} 0xA0000090 = 0xFF
set {unsigned char} 0xA0000091 = 0xFF
set {unsigned char} 0xA0000082 = 0xE7
set {unsigned char} 0xA000009A = 0xC0
set {unsigned char} 0xA0000093 = 0x0C
set {unsigned char} 0xA0000089 = 0x80
set {unsigned char} 0xA00002FF = 0x80
set $DUMMY = *(0xA00002FF)
printf "DUMMY_READ = 0x%X", $DUMMY
set {unsigned long} 0xA0002004 = 0x36DB0C00
set {unsigned long} 0xA0002028 = 0x00000500
set $DUMMY = *(0xA0002004)
printf "DUMMY_READ = 0x%X", $DUMMY
set $DUMMY = *(0xA0002028)
printf "DUMMY_READ = 0x%X", $DUMMY
set {unsigned long} 0xA0002008 = 0x36DB1C00
set {unsigned long} 0xA000202C = 0x00000500
set $DUMMY = *(0xA0002008)
printf "DUMMY_READ = 0x%X", $DUMMY
set $DUMMY = *(0xA000202C)
printf "DUMMY_READ = 0x%X", $DUMMY
```

c.「デバッグの構成＞ Renesas GDB Hardware Debugging ＞プロジェクト名 (HardwareDebug または Release)」の順に選択し、「Startup」タブをクリックする。

d. 以下の画面に示すように “source nor_init.txt” を追加する。



e.「適用」をクリックする。
f. メインタブ「Debugger」をクリックし、サブタブ「デバッグ・ツール設定」を選択する。



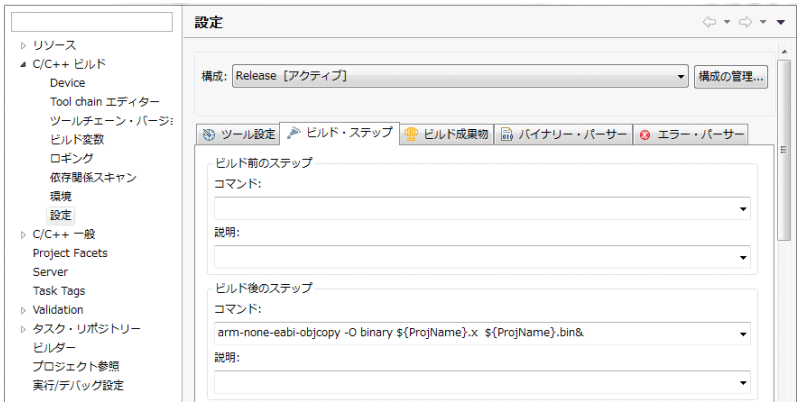
g.「Flash」の設定が上記に示すとおりであることを確認する。
h.「閉じる」をクリックする。
i. ユーザプロジェクトフォルダのルートへと移動し、該当する .jlink ファイル（例 System_Boot_Loader_NOR Release.jlink）を開く。

j. 以下の太字部分の設定となっていることを確認する。

[BREAKPOINTS]
ForceImpTypeAny = 0
ShowInfoWin = 1
EnableFlashBP = 2
BPDuringExecution = 0
[CFI]
CFISize = 0x4000000
CFIAddr = 0x40000000
[CPU]
OverrideMemMap = 0
AllowSimulation = 1
ScriptFile=""
[FLASH]
CacheExcludeSize = 0x00
CacheExcludeAddr = 0x00
MinNumBytesFlashDL = 0
SkipProgOnCRCMatch = 0
VerifyDownload = 1
AllowCaching = 1
EnableFlashDL = 2
Override = 0
Device="R7S910018_R4F"
[GENERAL]
WorkRAMSize = 0x4000
WorkRAMAddr = 0x20000000
RAMUsageLimit = 0x20003FFF
[SWO]
SWOLogFile=""
[MEM]
RdOverrideOrMask = 0x00
RdOverrideAndMask = 0xFFFFFFFF
RdOverrideAddr = 0xFFFFFFFF
WrOverrideOrMask = 0x00
WrOverrideAndMask = 0xFFFFFFFF
WrOverrideAddr = 0xFFFFFFFF

設定を保存し、ファイルを閉じてください。

プロジェクトがバイナリファイルを生成するように構成されていることを確認します。以下の順に操作してください。
「プロジェクトのプロパティ＞ C/C++ ビルド＞設定＞構成：Release ＞「ビルド・ステップ」タブを選択し、「ビルド後のステップ　コマンド：」を指定してください。



何も指定されていない場合、以下のようにコマンドを追加してください。

arm-none-eabi-objcopy -O binary ${ProjName}.x ${ProjName}.bin&

「適用」をクリックします。

「OK」をクリックします。

プロジェクトをビルドします。

以下の手順は、ユーザアプリケーションプログラムを NOR フラッシュメモリにプログラムする方法の一つです。この操作は、アドレス 0x40020000 に書きこまれている既存のアプリケーションプログラムを上書きします。

「デバッグの構成」を開きます。

目的の Release 構成（以下では、Tutorial Release を例に説明します）をクリックします。

「Startup」タブをクリックします。

「コマンドを実行」フィールドに命令を追加します。

以下の命令を先頭に追加します。

restore Release/Tutorial.bin binary 0x40020000

別のユーザアプリケーションプログラムを使用する場合は、"Tutorial" を該当のプログラム名に置き換えてください。

RSK+RZT1 の電源を入れます。

「適用」をクリックし、変更を保存します。

「デバッグ」をクリックします。

Segger J-Link のプログレスバーが短時間表示され、ユーザアプリケーションプログラムが NOR フラッシュメモリにプログラムされたことを示します。

e^2 studio の接続を解除します。

リセットスイッチ (RESET) を押下します。

ローダプログラムによってプログラムされたユーザアプリケーションプログラムが ATCM へロードされ、実行されます。

対象製品の接続やユーザアプリケーションプログラムのダウンロードに関する情報はチュートリアルマニュアル（R20UT3243JJ）を参照してください。

6. ユーザアプリケーションプログラムのデバッグ構成について

デバッグ構成内の各コマンドについて、Tutorial サンプルを用いて以下に説明します。

-exec-interrupt

CPU が現在実行中の動作を停止するための停止信号を送信します。

-data-write-memory-bytes --thread 1 0 "FF" 524288

524288 バイトのデータ (0xFF) をアドレス 0x0 から書き込みます。

-data-write-memory-bytes --thread 1 8396800 "FF" 22784

22784 バイトのデータ (0xFF) をアドレス 0x802000（8396800）から書き込みます。

load applications/RAM_DEBUG_JUMP.x

RAM_DEBUG_JUMP.x は 0x802000 にロードされた小容量のプログラムです。“jump-to” 命令を行い、プログラムカウンタ (PC) を、Tutorial などのサンプルを実行開始する ATCM の先頭アドレスにセットします。

load Release/Tutorial.x

Tutorial サンプルプログラムをロードします。

print /x $pc

e^2 studio のコンソールウィンドウ内に PC の現在のアドレスを示します。

set $pc = 0xFFFF0000

プログラムカウンタをハイベクタ領域 (リセットベクタアドレス) にセットします。

stepi 6

ステップ命令を 6 回実行します。これにより、Cortex R4F の初期命令の一部が実行可能となります。

-exec-interrupt

CPU が現在実行中の動作を停止するための停止信号を送信します。

set $pc = 0x802000

プログラムカウンタのポインタを、アドレス 0x802000 のプログラム RAM_DEBUG_JUMP.x にセットします。

stepi 6

ステップ命令を 6 回実行します。これにより、プログラムカウンタが ATCM(アドレス 0x0) の開始点にセットされます。

-exec-interrupt

CPU が現在実行中の動作を停止するための停止信号を送信します。

ブート処理に関する詳細は、RZ/T1 グループ　ユーザーズマニュアル　ハードウェア編（R01UH0483JJ）の『動作モード』の章を参照してください。

ホームページとサポート窓口

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

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

改訂記録	NOR フラッシュブートローダ アプリケーションノート

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2016.09.16	—	初版発行

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。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。
注2. 本資料において使用されている「当社製品」とは、注１において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/