RZ/A2Mグループ

初期設定例

要旨

本アプリケーションノートでは、RZ/A2Mのブートモード 3（シリアルフラッシュブート 3.3V 品）を使用して、初期設定を行う際に必要な設定項目の例について説明します。

対象デバイス

RZ/A2M

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

目次

1. 仕様

リセット解除後に、SPI マルチ I/O バス空間のシリアルフラッシュメモリに配置されたプログラムにて、SPI マルチ I/O バスコントローラ、クロックパルス発振器、割り込みコントローラ、汎用入出力ポート、メモリ管理ユニット、1 次キャッシュ（L1 キャッシュ）、および 2 次キャッシュ（L2 キャッシュ）の初期設定を行います。

本アプリケーションノートでは、SPI マルチ I/O バスコントローラを SPIBSC、クロックパルス発振器を CPG、割り込みコントローラを INTC、OS タイマを OSTM、FIFO 内蔵シリアルコミュニケーションインタフェースを SCIFA、汎用入出力ポートを GPIO、低消費電力モードを STB、メモリ管理ユニットを MMU とします。

表1.1に使用する周辺機能と用途を、図1.1にサンプルコード実行時の動作環境を示します。

表1.1 使用する周辺機能と用途

周辺機能	用途
SPI マルチ I/O バスコントローラ（SPIBSC）	外部アドレス空間リードモードに設定し、CPU が 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直接リードするための信号を生成
クロックパルス発振器（CPG）	RZ/A2Mの動作周波数の生成
割り込みコントローラ（INTC）	OSTM チャネル 0 および OSTM チャネル 2、SCIFA チャネル 4 の割り込み制御に使用
OS タイマ（OSTM）	OSTM チャネル 0 およびチャネル 2 を使用 • OSTM チャネル 0 LED 点灯および消灯の周期を生成 • OSTM チャネル 2 OS Abstraction Layer による時間管理に使用
FIFO 内蔵シリアルコミュニケーションインタフェース（SCIFA）	SCIFA チャネル 4 を用いて、ホスト PC との通信用として使用
汎用入出力ポート（GPIO）	SCIFA チャネル 4 の兼用端子切り替えに使用、LED の点灯および消灯のための端子制御に使用
低消費電力モード（STB）	RZ/A2Mの周辺 IO のモジュールスタンバイを解除するために使用、保持用内蔵 RAM をライト許可するために使用
メモリ管理ユニット（MMU）、L1 キャッシュ、L2 キャッシュ	RZ/A2Mの外部アドレス空間において、L1 キャッシュの有効領域の指定やメモリタイプの指定などの変換テーブルを生成。L1 キャッシュおよび L2 キャッシュを有効に設定

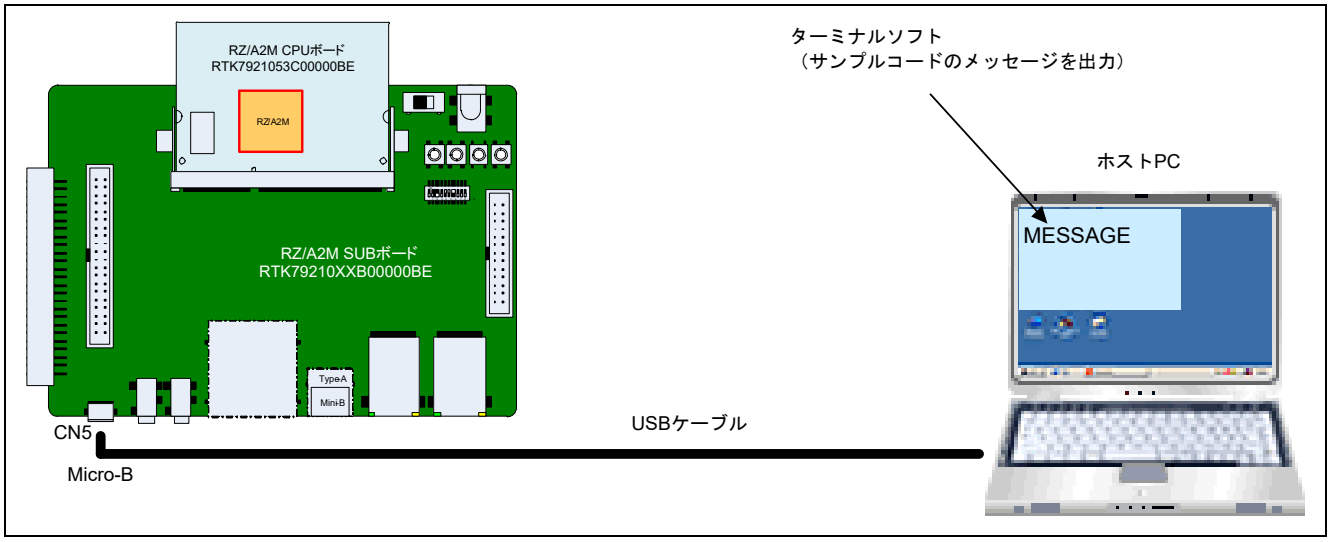

RZ/A2M CPUボード
RTK7921053C00000BE
RZ/A2M
RZ/A2M SUBボード
RTK79210XXB00000BE
TypeA
Mini-B
CN5
Micro-B
USBケーブル
ターミナルソフト
（サンプルコードのメッセージを出力）
ホストPC
MESSAGE

図1.1　動作環境

2. 動作確認条件

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

表2.1 動作確認条件（1/2）

項目	内容
使用 MCU	RZ/A2M
動作周波数（注）	CPU クロック（Iφ）：528MHz 画像処理クロック（Gφ）：264MHz 内部バスクロック（Bφ）：132MHz 周辺クロック 1（P1φ）：66MHz 周辺クロック 0（P0φ）：33MHz QSPI0_SPCLK：66MHz CKIO：132MHz
動作電圧	電源電圧（I/O）：3.3V 電源電圧（1.8/3.3V 切替 I/O（PVcc_SPI））：3.3V 電源電圧（内部）：1.2V
統合開発環境	e2 studio V7.6.0
C コンパイラ	GNU Arm Embedded Toolchain 6-2017-q2-update
	コンパイラオプション（ディレクトリパスの追加は除く） Release コンフィグレーション： -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Os -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -Wstack-usage=100 -fabi-version=0 Hardware Debug コンフィグレーション： -mcpu=cortex-a9 -march=armv7-a -marm -mlittle-endian -mfloat-abi=hard -mfpu=neon -mno-unaligned-access -Og -ffunction-sections -fdata-sections -Wunused -Wuninitialized -Wall -Wextra -Wmissing-declarations -Wconversion -Wpointer-arith -Wpadded -Wshadow -Wlogical-op -Waggregate-return -Wfloat-equal -Wnull-dereference -Wmaybe-uninitialized -g3 -Wstack-usage=100 -fabi-version=0

【注】 クロックモード 1（EXTAL 端子からの 24MHz のクロック入力）で使用時の動作周波数です。

表2.2　動作確認条件（2/2）

項目	内容
動作モード	ブートモード 3（シリアルフラッシュブート 3.3V 品）
ターミナルソフトの通信設定	• 通信速度：115200bps • データ長：8 ビット • パリティ：なし • ストップビット長：1 ビット • フロー制御：なし
使用ボード	RZ/A2M CPUボード　RTK7921053C00000BE RZ/A2M SUBボード　RTK79210XXB00000BE
使用デバイス （ボード上で使用する機能）	• シリアルフラッシュメモリ（SPI マルチ I/O バス空間に接続） メーカ名：Macronix 社、型名：MX25L51245GXD • RL78/G1C（USB 通信とシリアル通信を変換し、ホスト PC との通信に使用） • LED1

3.　関連アプリケーションノート

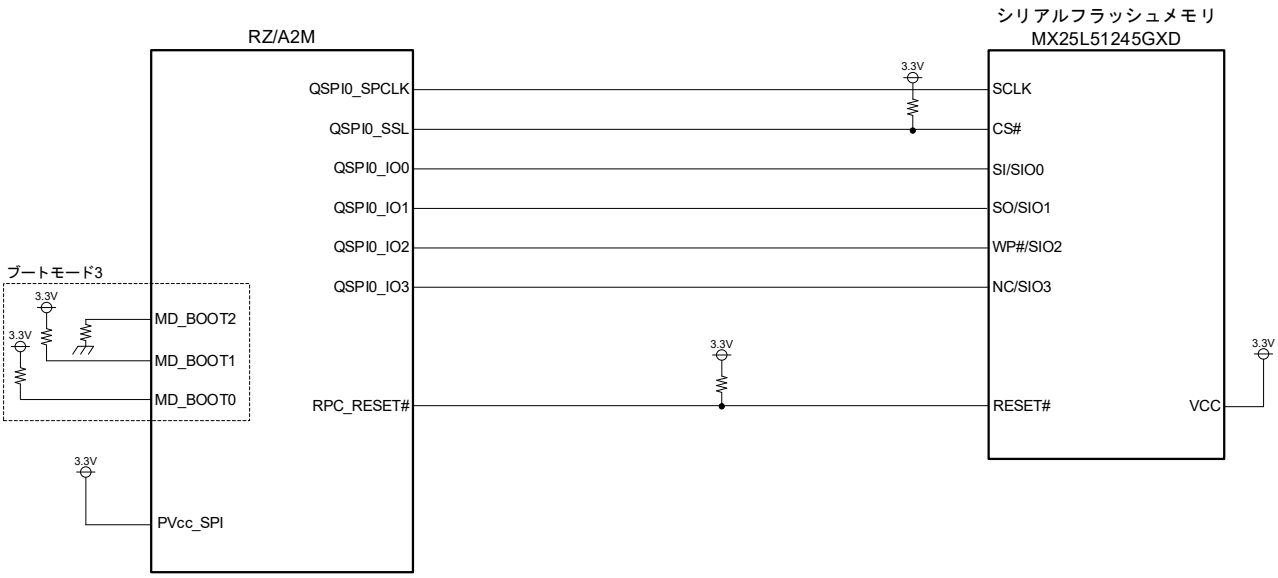
本アプリケーションノートに関連するアプリケーションノートを以下に示します。併せて参照してください。

- RZ/A2Mグループ シリアルフラッシュメモリからのブート例（R01AN4333JJ）

4. ハードウェア説明

4.1 ハードウェア構成例

本アプリケーションノートで紹介する初期設定例は、ブートモード 3 を使用して、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に格納されたプログラムにより処理を行います。図4.1にブートモード3でシリアルフラッシュメモリからブートする場合の接続例を示します。



【注】 #は負論理（またはアクティブロー）を示す記号です。

図4.1 ブートモード 3 でシリアルフラッシュメモリからブートする場合の接続例

4.2 使用端子一覧

表4.1に使用端子と機能を示します。

表4.1 使用端子と機能

端子名	入出力	内容
MD_BOOT2	入力	ブートモードの選択（ブートモード 3 に設定） MD_BOOT2："L"、MD_BOOT1："H"、MD_BOOT0："H"
MD_BOOT1	入力	
MD_BOOT0	入力	
QSPI0_SPCLK	出力	シリアルフラッシュメモリのクロック
QSPI0_SSL	出力	シリアルフラッシュメモリのスレーブセレクト
QSPI0_IO0	入出力	シリアルフラッシュメモリのデータ 0
QSPI0_IO1	入出力	シリアルフラッシュメモリのデータ 1
QSPI0_IO2	入出力	シリアルフラッシュメモリのデータ 2
QSPI0_IO3	入出力	シリアルフラッシュメモリのデータ 3
RPC_RESET#	出力	シリアルフラッシュメモリのリセット
P6_0	出力	LED の点灯および消灯
RxD4(P9_1)	入力	シリアル受信データ信号
TxD4(P9_0)	出力	シリアル送信データ信号

【注】 #は負論理（またはアクティブロー）を示す記号です。

5. ソフトウェア説明

5.1 動作概要

リセット解除後に、RZ/A2M の内蔵 ROM（H'FFFF 0000 番地）に格納されたブート起動用内蔵 ROM プログラムが実行されます。ブート起動用内蔵 ROM プログラムでは、ブートモード 3 でシリアルフラッシュメモリにアクセスするための処理を行った後、SPI マルチ I/O バス空間の先頭アドレスである H'2000 0000 番地に分岐します。

本初期設定プログラムは、H'2000 0000 番地に格納されたローダプログラムと、H'2001 0000 番地に格納されたアプリケーションプログラムから構成されています。

H'2000 0000 番地に格納されたローダプログラムにより、シリアルフラッシュメモリにアクセスするための最適な設定を行い、H'2001 0000 番地に格納されたアプリケーションプログラムのスタートアップ処理に分岐します。

ここでは、アプリケーションプログラムのスタートアップ処理で実施している初期設定例について説明します。スタートアップ処理では、スタックポインタ、MMU、FPU の設定を行い、セクションの初期化を行い、resetprg 関数に分岐します。

resetprg 関数では、RTC と USB 未使用チャネルの初期化処理を実行後、L1 キャッシュおよび L2 キャッシュの有効化と INTC の初期化を行い、割り込み処理高速化のために VBAR に大容量内蔵 RAM のアドレスを設定し、IRQ 割り込みおよび FIQ 割り込みを許可にし、main 関数をコールしています。

main 関数では、CPG、OSTM チャネル 0、SCIFA チャネル 4、GPIO の初期設定処理を行います。これら初期化処理により、シリアルインタフェースに接続されたホスト PC 上のターミナルに文字列（起動メッセージ）を出力し、OSTM のチャネル 0 をインターバルタイマモードに設定してタイマを起動します。500ms の周期で OSTM チャネル 0 の割り込みを発生させ、CPU ボード上の LED を割り込み処理により 500ms ごとに点灯および消灯を繰り返す処理を行います。

ローダプログラムの処理については、「RZ/A2M グループ シリアルフラッシュメモリからのブート例」のアプリケーションノートをご参照ください。

図5.1にメイン処理が実行されるまでの初期設定のシーケンス図を示します。

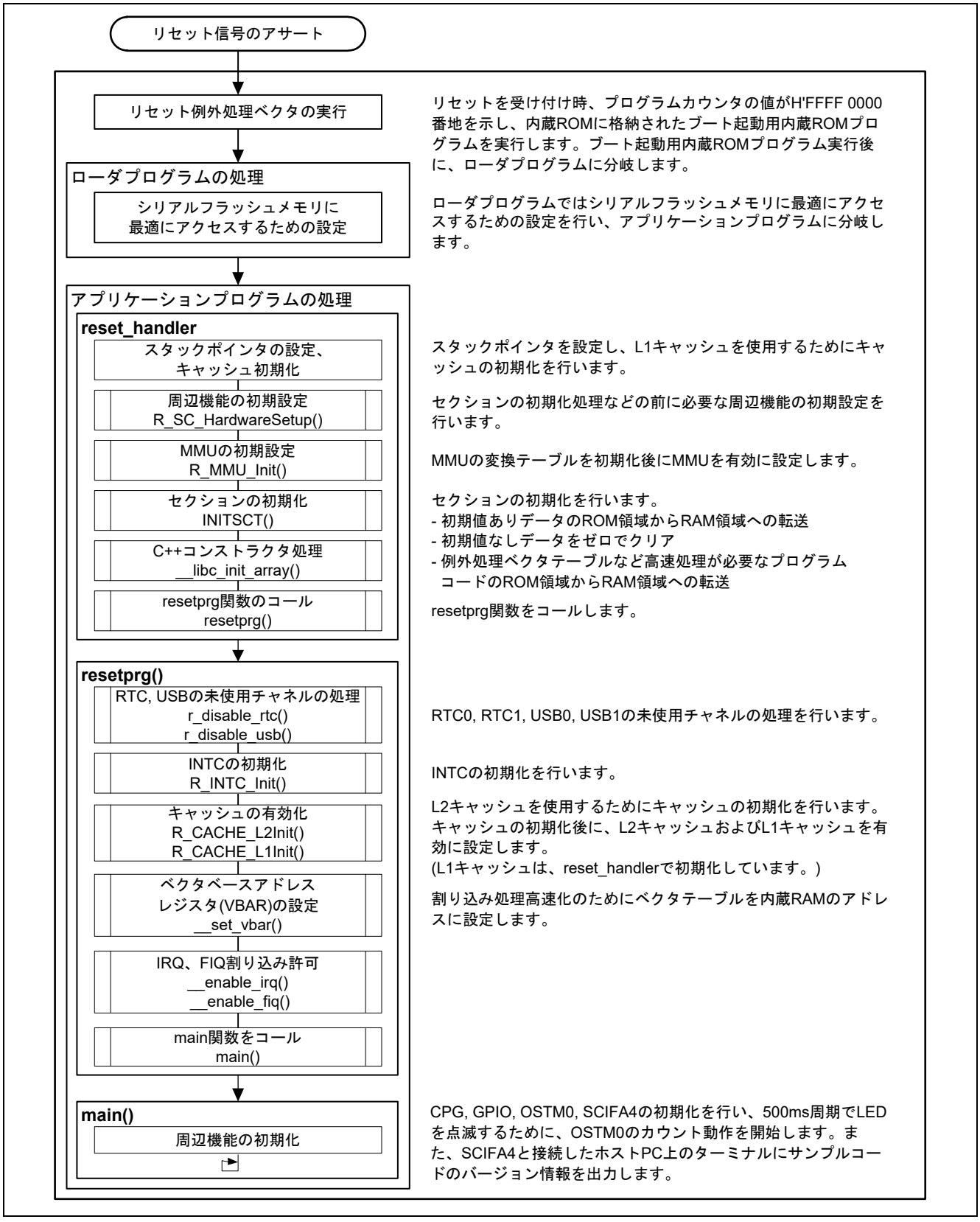


図5.1　メイン処理が実行されるまでの初期設定のシーケンス図

5.2 サンプルコード実行時の周辺機能の設定およびメモリ配置

5.2.1 周辺機能の設定

表5.1にサンプルコード実行時の周辺機能の設定内容を示します。

表5.1 周辺機能の設定内容

モジュール	設定内容
CPG	CPU クロック：PLL 回路の周波数に対して×1/2 倍に設定 内部バスクロック：PLL 回路の周波数に対して×1/8 倍に設定 周辺クロック 1（P1φ）：PLL 回路の周波数に対して×1/16 倍に設定 クロックモード 1（分周器 1：×1/2 倍、PLL 回路：×88 倍）で、入力クロックが 24MHz の場合に以下の周波数となるように設定 • CPU クロック（Iφ）：528MHz • 画像処理クロック（Gφ）：264MHz • 内部バスクロック（Bφ）：132MHz • 周辺クロック 1（P1φ）：66MHz • 周辺クロック 0（P0φ）：33MHz • QSPI0_SPCLK：66MHz（Bφ選択時） • CKIO クロック：132MHz（Bφ選択時）
STB	保持用内蔵 RAM へのライト許可および周辺機能へのクロック供給 STBCR3, STBCR4 で OSTM0, OSTM2, SCIFA4 にクロックを供給
GPIO	PORT6、PORT9 の兼用端子機能を設定 • P6_0：LED の点灯および消灯 • P9_1：RxD4、P9_0：TxD4
OSTM	チャネル 0、チャネル 2 をインターバルタイマモードに設定 • チャネル 0 P1φ=66MHz の時に 500ms ごとに割り込み要求を発生するようにタイマカウントを設定。LED 点灯および消灯処理で使用 • チャネル 2 P1φ=66MHz の時に 1ms ごとに割り込み要求を発生するようにタイマカウントを設定。OS Abstraction Layer による時間管理に使用。
INTC	INTC の初期設定および OSTM チャネル 0 割り込み（割り込み ID が 88）ハンドラの登録、OSTM チャネル 2 割り込み（割り込み ID が 90）ハンドラの登録、SCIFA チャネル 4 割り込み（割り込み ID が 322, 323）ハンドラの登録と実行
SCIFA	チャネル 4 を調歩同期式モードに設定 • データ長：8 ビット • ストップビット長：1 ビット • パリティ：なし • データ転送方向：LSB ファースト転送 P1φ=66MHz の時に、クロックソースを分周なし、ボーレートジェネレータは倍速モード、ビットレートの 8 倍の基本クロックで動作するように設定。ビットレートが 115200bps となるように、ビットレート値に 71 を設定 （誤差：-0.53%）

5.2.2　メモリマップ

図5.2にRZ/A2Mグループのアドレス空間とRZ/A2M CPUボードのメモリマップを示します。

サンプルコードでは、ROM 領域を使用するコードおよびデータを SPI マルチ I/O バス空間に接続したシリアルフラッシュメモリに配置し、RAM 領域を使用するコードおよびデータを大容量内蔵 RAM に配置するようにしています。

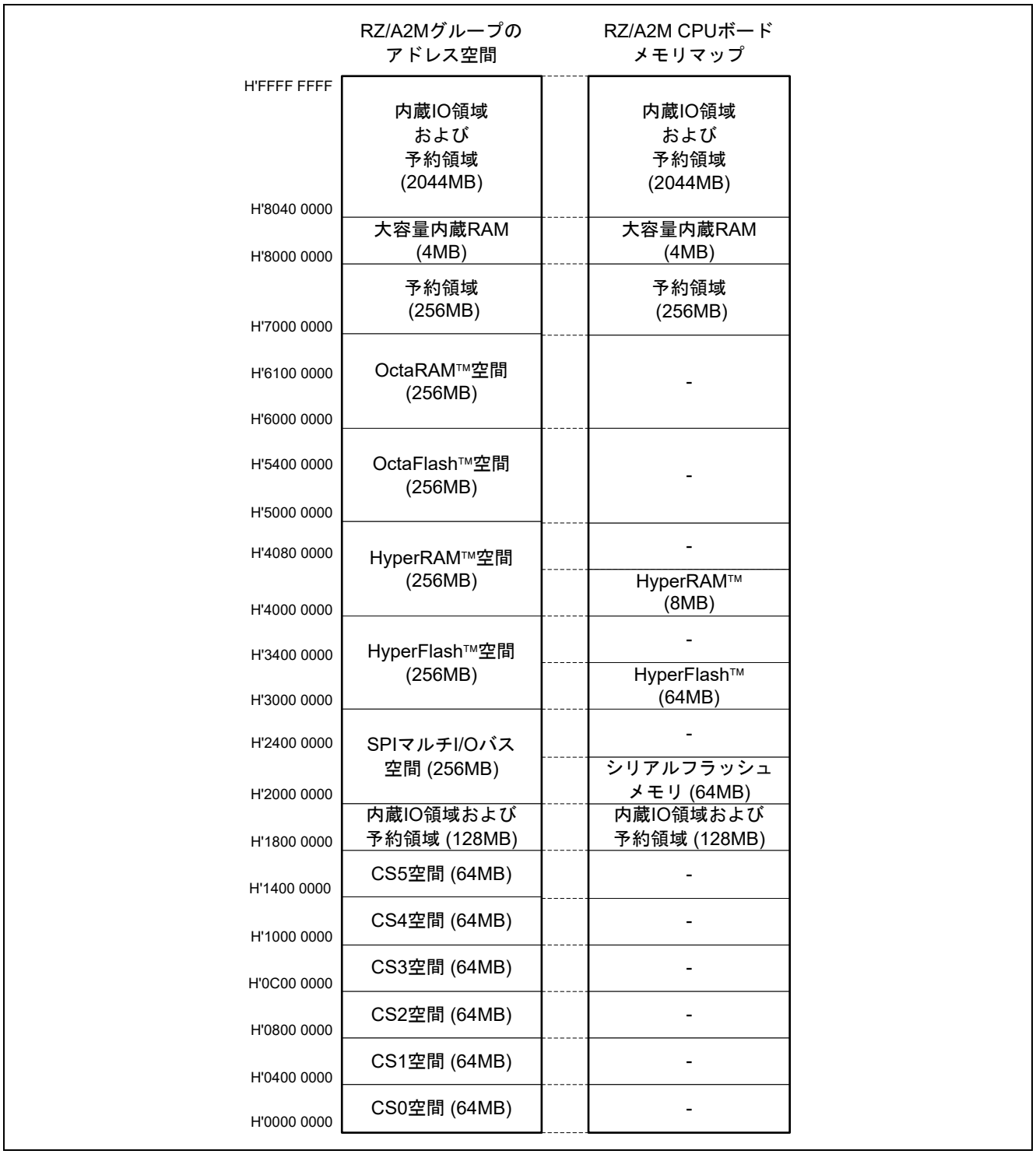


図5.2　メモリマップ

5.2.3 MMU の設定

RZ/A2M CPUボードで使用するハードウェア資源のメモリマップにあわせて、H'0000 0000 番地から 1MB 単位で 4GB の領域を、第 1 レベル記述子の"セクション"の記述子タイプを使用して MMU で管理するように設定しています（MMU コンフィグレーションテーブル（MMU_SC_TABLE[]）は、r_mmu_drv_sc_cfg.h ファイルで定義されています）。システムにあわせてカスタマイズする場合は、最少単位を 1MB としてください。第 1 レベル記述子の変換テーブルは、変換テーブルベース制御レジスタ（TTBCR）、変換テーブルベースレジスタ 0 または 1（TTBR0 または TTBR1）により設定します。すべての変換テーブルの情報を設定完了後に、MMU を許可（システム制御レジスタ（SCTLR）の M ビットを設定）します。MMU についての詳細は、「ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C」を参照してください。

- TTBCR
  変換テーブルのベースアドレスに使用するベースレジスタ（TTBR0 または TTBR1）と、TTBR0 によってアクセスされる変換テーブルのサイズを設定します。サンプルコードでは、N[2:0]ビットに b'000 を設定し、常に TTBR0 を使用し変換テーブルサイズを 16KB（4096 エントリ）としています。
  変換テーブルは、第 1 レベルテーブルを使用し、図5.3に示すように第 1 レベル記述子の記述子タイプがセクションのフォーマットを使用します。セクションのフォーマットは、変換テーブルの各エントリが 1MB のメモリブロックで構成され、4GB（4096 エントリ×1MB）のアドレス空間に対して、変換テーブルにより仮想アドレスから物理アドレスへの変換を行います。
- TTBR0
  変換テーブルを配置するベースアドレス、変換テーブルを配置する領域のキャッシュ可能属性などを設定します。サンプルコードでは、大容量内蔵 RAM に配置した変換テーブルのセクション領域の先頭アドレス"__mmu_page_table_base"の値（linker_script.ld で定義）をベースアドレスに設定しています。

図5.3、表5.2、表5.3および表5.4に第 1 レベル記述子の内容とサンプルコードの設定内容を、表5.5および表5.6にサンプルコードの MMU の設定を示します。

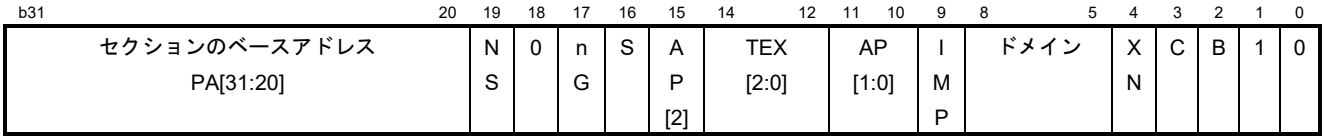


図5.3　第 1 レベル記述子の記述子タイプがセクションのフォーマット

表5.2　第 1 レベル記述子のフィールド

フィールド	内容
TEX[2:0], C, B	メモリ領域属性ビットです。 サンプルコードで使用する内容は、表5.3を参照してください。
XN	実行不可ビットです。XN 属性が"1"の場合、クライアントに設定されたドメイン内のメモリ領域から命令をフェッチしないようにします。 サンプルコードでは、メモリタイプがノーマルメモリの領域を実行可能属性（XN を"0"）に、ストロングリオーダメモリの領域を実行不可属性（XN を"1"）しています。
ドメイン	クライアントとマネージャの 2 種類のドメインアクセスがサポートされており、ドメインアクセスの設定はドメインアクセス制御レジスタ（DACR）で行います。 サンプルコードでは、DACR の D15 フィールドを"b'01（クライアントアクセス）"に設定しており、すべての空間に対して本ドメインフィールドで"b'1111（D15 フィールド）"を指定しています。
IMP	RZ/A2Mでは何も実装されていません。設定した値は無視されます。
AP[2], AP[1:0]	本フィールドとドメインフィールドによって、メモリアクセスの制御を行います。 サンプルコードでは、予約領域がアクセス不可（AP[2:0]を b'000）、予約領域以外のすべての空間を完全アクセス許可（AP[2:0]を b'011）にしています。
S	メモリ領域の共有可能属性を設定するビットです。 サンプルコードでは、メモリタイプがノーマルメモリのすべての空間を共有不可（S を"0"）にしています。
nG	非グローバルビットです。 サンプルコードでは、すべての空間をグローバル（nG を"0"）にしています。
NS（注）	非セキュアビットです。 サンプルコードでは、大容量内蔵 RAM および外部アドレス空間を非セキュア（NS を"1"）に、内蔵周辺モジュール空間をセキュア（NS を"0"）にしています。
PA[31:20]	仮想アドレスから変換後の物理アドレスの b'31～b'20 を指定します。

【注】 RZ/A2Mでは、CPU のセキュリティ状態をセキュア状態（セキュア構成レジスタ（SCR）の NS ビットが"0"）に設定してください。CPU をセキュア状態にすることで、NS ビットの設定が非セキュアおよびセキュアのどちらの領域でも、MMU によりアドレス変換が行われる領域を CPU からアクセスすることができます。
CPU、DMAC、CoreSight を除く内蔵周辺モジュールがバスマスタとなり、L2 キャッシュを経由して外部メモリ空間にアクセスする場合は、サンプルコードでは AXI バス関連レジスタ MSTACCCTL0～MSTACCCTL4 の各内蔵周辺モジュールの設定が"1"（ノンセキュアアクセス）であることを前提に、NS ビットを"1"（非セキュア）に設定して CPU から大容量内蔵 RAM および外部アドレス空間へのアクセスを行います。MSTACCCTL0～MSTACCCTL4 の各内蔵周辺モジュールの設定が"1"（ノンセキュアアクセス）となるようにしてください（初期値は"1"です）。

表5.3　サンプルコードで使用するメモリ領域属性ビットの設定

TEX[2:0]	C	B	メモリタイプ	L1 キャッシュ	L2 キャッシュ
b'000	0	0	ストロングリオーダメモリ	ー	ー
b'000	0	1	デバイス（共有可能）	ー	ー
b'001	1	1	ノーマルメモリ	有効	有効
b'100	0	1	ノーマルメモリ	有効	無効
b'100	0	0	ノーマルメモリ	無効	無効

表5.4　サンプルコードの MMU 設定名とその設定内容

MMU 設定名	メモリタイプ	キャッシュ	NS	AP[2:0](Access Permission)	XN
Unused	ストロングリオーダメモリ	ー	1	Read/Write(b'011)	1
Strongly-ordered(secure, Never-execute)	ストロングリオーダメモリ	ー	0	Read/Write(b'011)	1
Strongly-ordered(non-secure, Never-execute)	ストロングリオーダメモリ	ー	1	Read/Write(b'011)	1
Strongly-ordered(non-secure, Executable)	ストロングリオーダメモリ	ー	1	Read/Write(b'011)	0
Shareable Device(secure)	デバイス	ー	0	Read/Write(b'011)	1
Shareable Device (non-secure)	デバイス	ー	1	Read/Write(b'011)	1
Normal(non-secure, L1 cacheable)	ノーマルメモリ	L1 キャッシュのみ有効	1	Read/Write(b'011)	0
Normal(non-secure, L2 cacheable)	ノーマルメモリ	L2 キャッシュのみ有効	1	Read/Write(b'011)	0
Normal(non-secure, L1/L2 cacheable)	ノーマルメモリ	L1/L2 キャッシュ有効	1	Read/Write(b'011)	0
Normal(non-secure, non-cacheable)	ノーマルメモリ	L1/L2 キャッシュ無効	1	Read/Write(b'011)	0
Reserved	ストロングリオーダメモリ	ー	1	Access Inhibit(b'000)	1

表5.5 MMU の設定(1/2)

仮想アドレス	物理アドレス*2	サイズ	領域	MMU 設定名*1
H'0000 0000 H'03FF FFFF	H'0000 0000 H'03FF FFFF	64MB	CS0 空間 （未使用）	Unused
H'0400 0000 H'07FF FFFF	H'0400 0000 H'07FF FFFF	64MB	CS1 空間 （未使用）	Unused
H'0800 0000 H'0BFF FFFF	H'0800 0000 H'0BFF FFFF	64MB	CS2 空間 （未使用）	Unused
H'0C00 0000 H'0FFF FFFF	H'0C00 0000 H'0FFF FFFF	64MB	CS3 空間 キャッシュ有効領域	Normal(non-secure, L1/L2 cacheable)
H'1000 0000 H'13FF FFFF	H'1000 0000 H'13FF FFFF	64MB	CS4 空間 （未使用）	Unused
H'1400 0000 H'17FF FFFF	H'1400 0000 H'17FF FFFF	64MB	CS5 空間 （未使用）	Unused
H'1800 0000 H'1EFF FFFF	H'1800 0000 H'1EFF FFFF	112MB	予約領域	Reserved
H'1F00 0000 H'1FFF FFFF	H'1F00 0000 H'1FFF FFFF	16MB	内蔵 I/O 領域	Strongly-ordered(secure, Never-execute)
H'2000 0000 H'2FFF FFFF	H'2000 0000 H'2FFF FFFF	256MB	SPI マルチ I/O バス キャッシュ有効領域	Normal(non-secure, L1/L2 cacheable)
H'3000 0000 H'3FFF FFFF	H'3000 0000 H'3FFF FFFF	256MB	HyperFlash キャッシュ有効領域	Normal(non-secure, L1/L2 cacheable)
H'4000 0000 H'4FFF FFFF	H'4000 0000 H'4FFF FFFF	256MB	HyperRAM キャッシュ有効領域	Normal(non-secure, L1/L2 cacheable)
H'5000 0000 H'5FFF FFFF	H'5000 0000 H'5FFF FFFF	256MB	OctaFlash キャッシュ有効領域	Normal(non-secure, L1/L2 cacheable)
H'6000 0000 H'6FFF FFFF	H'6000 0000 H'6FFF FFFF	256MB	OctaRAM キャッシュ有効領域	Normal(non-secure, L1/L2 cacheable)
H'7000 0000 H'7FFF FFFF	H'2000 0000 H'2FFF FFFF	256MB	SPI マルチ I/O バス キャッシュ無効領域	Strongly-ordered(non-secure, Executable)
H'8000 0000 H'803F FFFF	H'8000 0000 H'803F FFFF	4MB	大容量内蔵 RAM キャッシュ有効領域	Normal(non-secure, L1 cacheable)

表5.6 MMU の設定(2/2)

仮想アドレス	物理アドレス*2	サイズ	領域	MMU 設定名*1
H'8040 0000 H'81FF FFFF	H'8040 0000 H'81FF FFFF	28MB	予約領域	Unused
H'8200 0000 H'823F FFFF	H'8000 0000 H'803F FFFF	4MB	大容量内蔵 RAM キャッシュ無効領域	Normal(non-secure, non-cacheable)
H'8240 0000 H'87FF FFFF	H'8240 0000 H'87FF FFFF	92MB	予約領域	Unused
H'8800 0000 H'8BFF FFFF	H'0000 0000 H'03FF FFFF	64MB	CS0 空間（未使用）	Unused
H'8C00 0000 H'8FFF FFFF	H'0400 0000 H'07FF FFFF	64MB	CS1 空間（未使用）	Unused
H'9000 0000 H'93FF FFFF	H'0800 0000 H'0BFF FFFF	64MB	CS2 空間（未使用）	Unused
H'9400 0000 H'97FF FFFF	H'0C00 0000 H'0FFF FFFF	64MB	CS3 空間キャッシュ無効領域	Normal(non-secure, non-cacheable)
H'9800 0000 H'9BFF FFFF	H'1000 0000 H'13FF FFFF	64MB	CS4 空間（未使用）	Unused
H'9C00 0000 H'9FFF FFFF	H'1400 0000 H'17FF FFFF	64MB	CS5 空間（未使用）	Unused
H'A000 0000 H'AFFF FFFF	H'3000 0000 H'3FFF FFFF	256MB	HyperFlash キャッシュ無効領域	Strongly-ordered(non-secure, Executable)
H'B000 0000 H'BFFF FFFF	H'4000 0000 H'4FFF FFFF	256MB	HyperRAM キャッシュ無効領域	Normal(non-secure, non-cacheable)
H'C000 0000 H'CFFF FFFF	H'5000 0000 H'5FFF FFFF	256MB	OctaFlash キャッシュ無効領域	Strongly-ordered(non-secure, Executable)
H'D000 0000 H'DFFF FFFF	H'6000 0000 H'6FFF FFFF	256MB	OctaRAM キャッシュ無効領域	Normal(non-secure, non-cacheable)
H'E000 0000 H'E7FF FFFF	H'E000 0000 H'E7FF FFFF	128MB	予約領域	Reserved
H'E800 0000 H'FFFF FFFF	H'E800 0000 H'FFFF FFFF	384MB	内蔵 I/O 領域	Strongly-ordered(secure, Never-execute)

【注】 1. 「MMU 設定名」に対応する MMU の設定内容は、表5.3および表5.4を参照してください。
2. 仮想アドレスと物理アドレスが異なる領域は赤字にしています。

5.2.4　サンプルコードで使用する仮想アドレス空間

サンプルコードでは、MMU を使用して、図5.4および図5.5に示すように CPU がアクセス可能な仮想アドレス空間を準備しています。

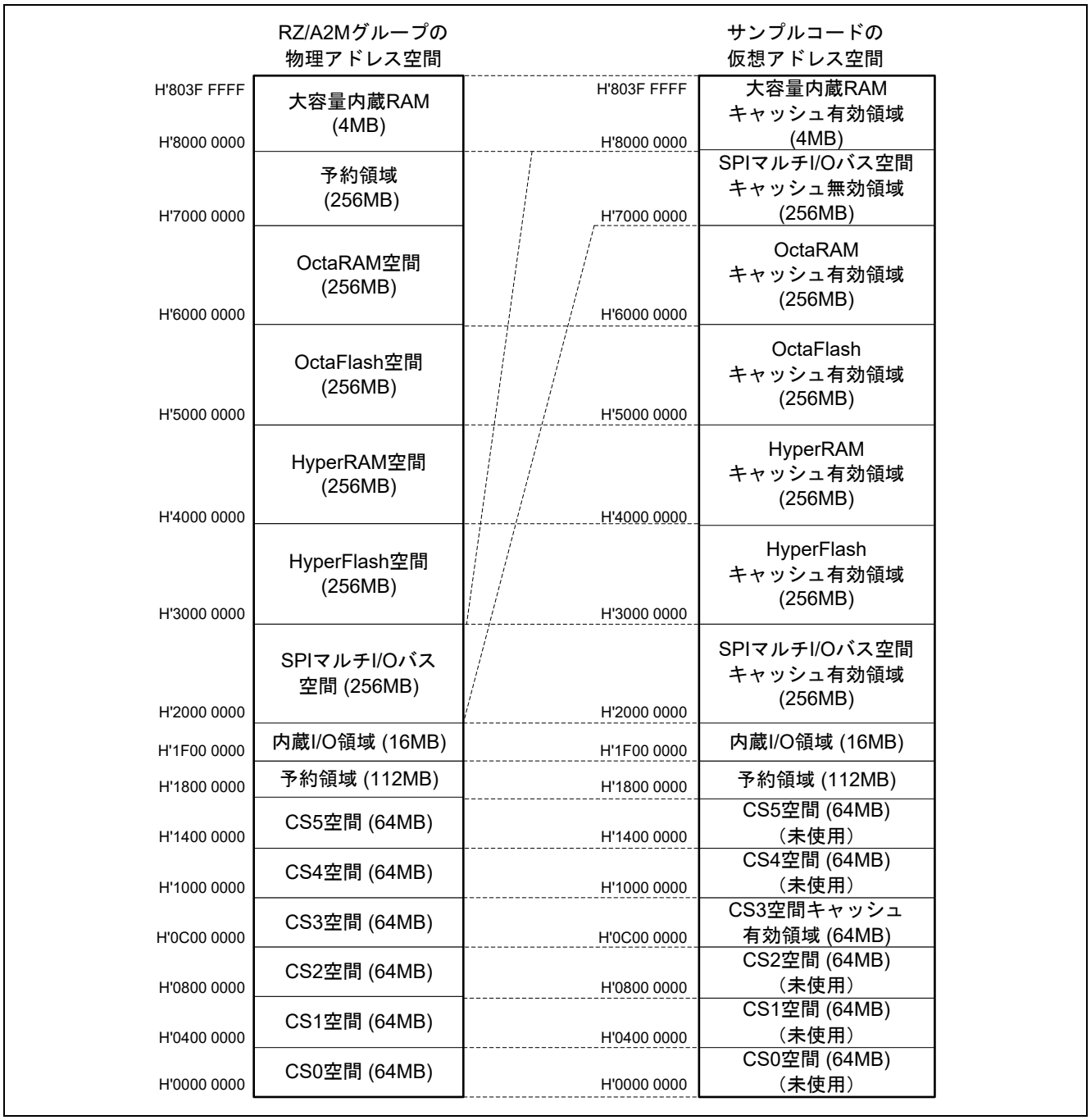
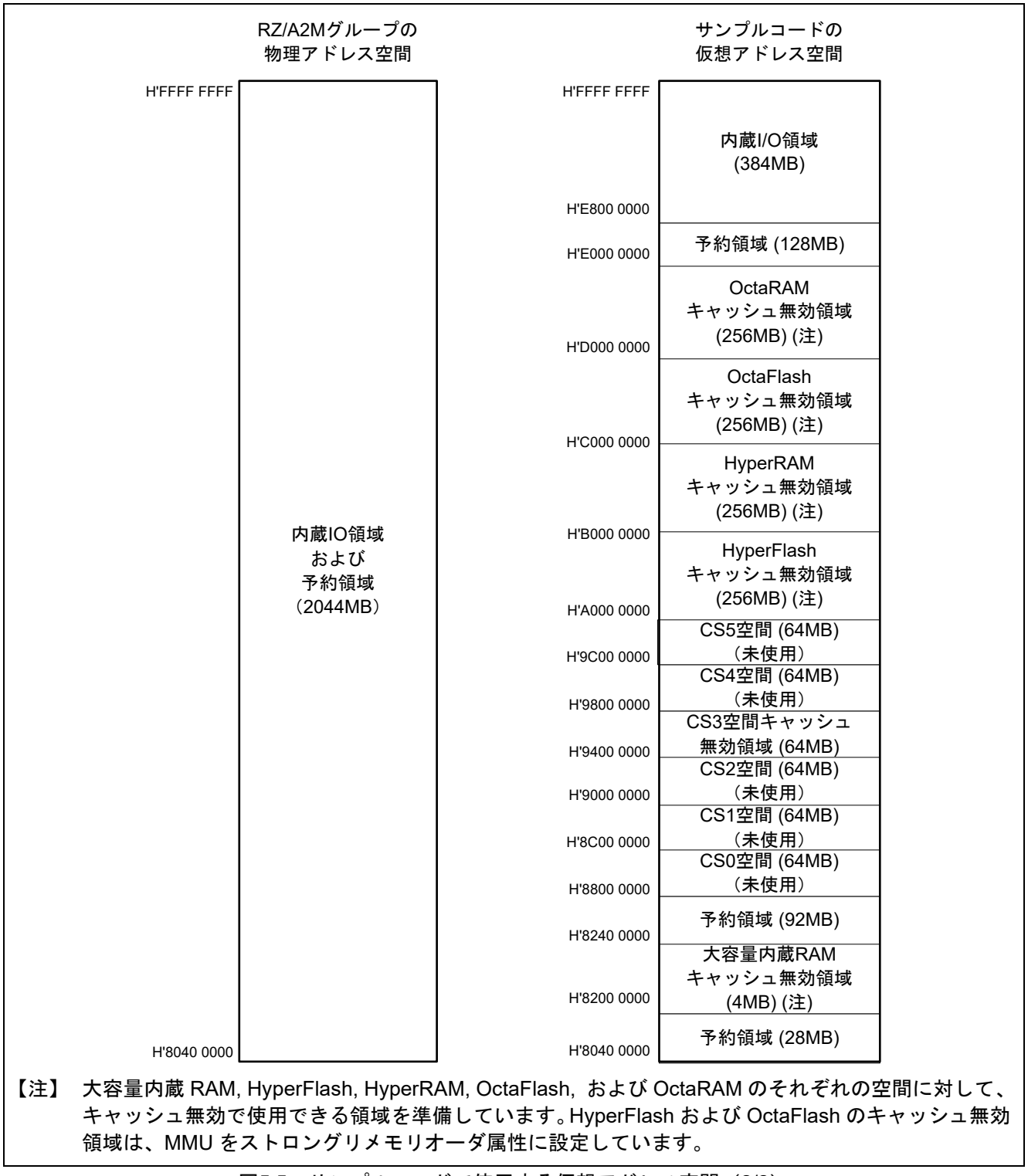


図5.4　サンプルコードで使用する仮想アドレス空間（1/2）



【注】 大容量内蔵 RAM, HyperFlash, HyperRAM, OctaFlash, および OctaRAM のそれぞれの空間に対して、キャッシュ無効で使用できる領域を準備しています。HyperFlash および OctaFlash のキャッシュ無効領域は、MMU をストロングリメモリオーダ属性に設定しています。

図5.5 サンプルコードで使用する仮想アドレス空間（2/2）

5.2.5 サンプルコードのセクション配置

サンプルコードでは、割り込み処理の高速化のため、例外処理ベクタテーブルと IRQ 割り込みハンドラを大容量内蔵 RAM 上に配置して、これらの処理を大容量内蔵 RAM 上で実行するようにしています。例外処理ベクタテーブルおよび IRQ 割り込みハンドラのプログラムデータをシリアルフラッシュメモリ領域から大容量内蔵 RAM 領域への転送処理、初期値なしデータセクションのゼロクリア処理、および初期値ありデータセクションの初期化処理は、INITSCT 関数を使用して行っています。INITSCT 関数では、section.c で定義されているセクション初期化用のテーブルデータを参照して、各セクションの初期化処理を行います。プログラムデータの配置は、リンカスクリプト（linker_script.ld）に記述しています。

表5.7にサンプルコードで使用するセクション名とオブジェクト名一覧を示します。図5.6にサンプルコードの初期状態のセクション配置（ロードビュー）と、INITSCT 関数実行後のセクション配置（実行ビュー）を示します。

表5.7　サンプルコードで使用するセクション名とオブジェクト名一覧

出力セクション名	入力セクション名 入力オブジェクト名	内容	ロード領域	実行領域
LOAD_MODULE1	VECTOR_TABLE	例外処理ベクタテーブル	FLASH	FLASH
LOAD_MODULE2	*/r_cpg/*.o (.text .rodata .data)	CPG の設定処理	FLASH	LRAM
	*/rza_io_regrw.o (.text .rodata .data)	IO レジスタアクセス処理		
	/hwsetup.o (.text .rodata .data)	HardwareSetup の設定処理		
LOAD_MODULE3	*/r_cpg/*.o (.bss)	CPG の設定処理の初期値なしデータ領域	–	LRAM
	*/rza_io_regrw.o (.bss)	IO レジスタアクセス処理用の初期値なしデータ領域		
LOAD_MODULE4	RESET_HANDLER	リセット処理	FLASH	FLASH
	INIT_SECTION */sections.o	セクション初期化処理		
.data	VECTOR_MIRROR_TABLE	例外処理ベクタテーブル	FLASH	LRAM
	*/r_intc_*.o (.text .rodata .data)	INTC ドライバ処理用コード領域		
	IRQ_FIQ_HANDLER	IRQ/FIQ ハンドラ処理		
.bss	なし	なし	–	LRAM
.uncached_RAM	*/r_cache_*.o (.bss)	L1 および L2 キャッシュ設定処理用初期値なしデータ領域（注 2）	–	LRAM
	UNCACHED_BSS	初期値なしデータ領域（非キャッシュ設定）		
.uncached_RAM2	*/r_cache_*.o (.text .rodata .data)	L1 および L2 キャッシュ設定処理（注 2）	FLASH	LRAM
	UNCACHED_DATA	初期値ありデータ領域（非キャッシュ設定）		
.mmu_page_table	なし	MMU 変換テーブル領域	–	LRAM
.stack	なし	システムモードのスタック領域	–	LRAM
		IRQ モードのスタック領域		
		FIQ モードのスタック領域		
		SVC モードのスタック領域		
		アボート(ABT)モードのスタック領域		
.text2	* (.text .text.*)	デフォルトのコード領域	FLASH	FLASH
	* (.rodata .rodata.*)	デフォルトの定数データ領域		
.data2	* (.data .data.*)	デフォルトの初期値ありデータ領域	FLASH	LRAM
.bss2	* (.bss .bss.*) * (COMMON)	デフォルトの初期値なしデータ領域	–	LRAM
.heap	なし	ヒープ領域	–	LRAM

【注】 1. 表中のロード領域および実行領域において、FLASH はシリアルフラッシュメモリの領域を、LRAM は大容量内蔵 RAM の領域を表します。
2. このセクションは、キャッシュ無効領域に配置する必要があります。

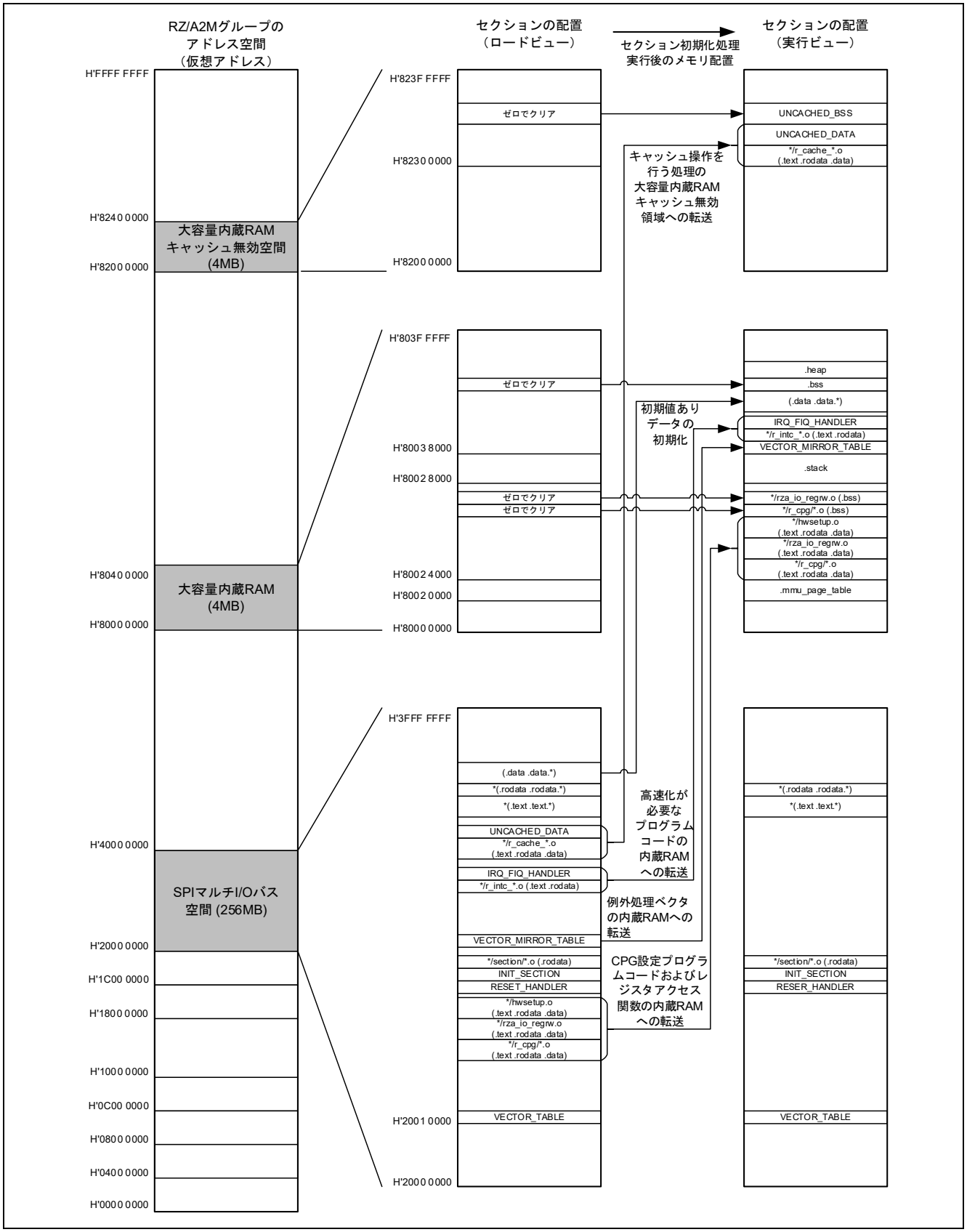
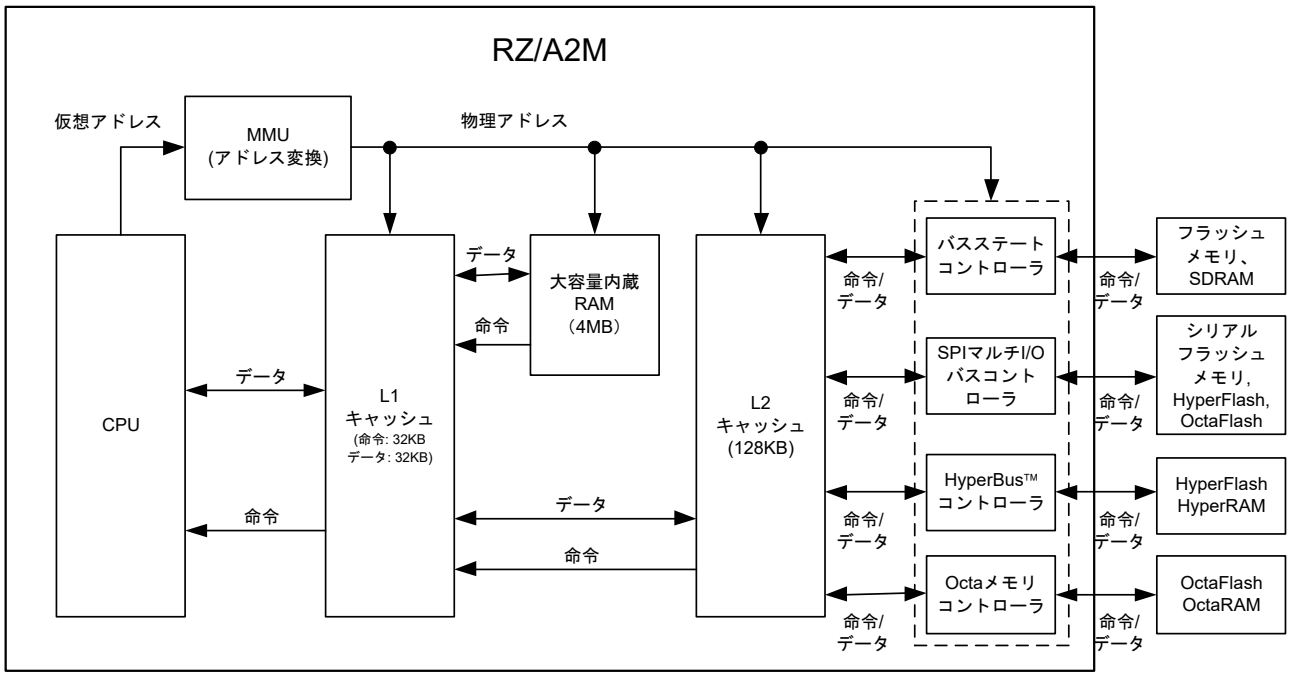

RZ/A2Mグループの
アドレス空間
（仮想アドレス）
セクションの配置
（ロードビュー）
セクション初期化処理
実行後のメモリ配置
セクションの配置
（実行ビュー）
H'FFFF FFFF
H'8240 0000
大容量内蔵RAM
キャッシュ無効空間
(4MB)
H'8200 0000
H'8040 0000
大容量内蔵RAM
(4MB)
H'8000 0000
H'4000 0000
SPIマルチI/Oバス
空間 (256MB)
H'2000 0000
H'1C00 0000
H'1800 0000
H'1000 0000
H'0C00 0000
H'0800 0000
H'0400 0000
H'0000 0000
H'823F FFFF
H'8230 0000
H'8200 0000
H'803F FFFF
H'8003 8000
H'8002 8000
H'8002 4000
H'8002 0000
H'8000 0000
H'3FFF FFFF
H'2001 0000
H'2000 0000
ゼロでクリア
ゼロでクリア
ゼロでクリア
ゼロでクリア
(.data .data.*)
(.rodata .rodata.)
(.text .text.)
UNCACHED_DATA
*/r_cache_*.o
(.text .rodata .data)
IRQ_FIQ_HANDLER
*/r_intc_*.o (.text .rodata)
VECTOR_MIRROR_TABLE
/section/.o (.rodata)
INIT_SECTION
RESET_HANDLER
*/hwsetup.o
(.text .rodata .data)
*/rza_io_regrw.o
(.text .rodata .data)
/r_cpg/.o
(.text .rodata .data)
VECTOR_TABLE
キャッシュ操作を
行う処理の
大容量内蔵RAM
キャッシュ無効
領域への転送
初期値あり
データの
初期化
高速化が
必要な
プログラム
コードの
内蔵RAM
への転送
例外処理ベクタ
の内蔵RAMへの
転送
CPG設定プログラ
ムコードおよびレ
ジスタアクセス
関数の内蔵RAM
への転送
UNCACHED_BSS
UNCACHED_DATA
*/r_cache_*.o
(.text .rodata .data)
.heap
.bss
(.data .data.*)
IRQ_FIQ_HANDLER
*/r_intc_*.o (.text .rodata)
VECTOR_MIRROR_TABLE
.stack
*/rza_io_regrw.o (.bss)
/r_cpg/.o (.bss)
*/hwsetup.o
(.text .rodata .data)
*/rza_io_regrw.o
(.text .rodata .data)
/r_cpg/.o
(.text .rodata .data)
.mmu_page_table
(.rodata .rodata.)
(.text .text.)
/section/.o (.rodata)
INIT_SECTION
RESER_HANDLER
VECTOR_TABLE

図5.6　セクション配置

5.2.6 L1 キャッシュと L2 キャッシュの設定

RZ/A2Mは、L1 キャッシュと L2 キャッシュの 2 種類のキャッシュメモリを内蔵しています（L1 キャッシュは命令 32K バイト/データ 32K バイト、L2 キャッシュは 128K バイト）。

図5.7にメモリ階層におけるL1キャッシュとL2キャッシュのブロック図を示します。



【注】 L1 の命令キャッシュには、仮想的にインデックスが付けられ、物理的にタグが付けられます。また、L1 のデータキャッシュには、物理的にインデックスが付けられ、物理的にタグが付けられます。
上記の図は、RZ/A2Mのキャッシュメモリのメモリ階層のイメージを示しており、キャッシュメモリの動作についての詳細は、「ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C」を参照してください。

図5.7 メモリ階層における L1 キャッシュと L2 キャッシュのブロック図

CPU から外部メモリ空間および内蔵空間へのアクセスを行うときに、MMU の変換テーブルが参照され、仮想アドレスを物理アドレスに変換してアクセスします。MMU の変換テーブルには、図5.3に第1レベル記述子の記述子タイプがセクションのフォーマットで示すように、アドレス変換後の物理アドレスのベースアドレスとともに、そのベースアドレスを先頭とする領域（記述子タイプがセクションの場合は、1MB の領域）のキャッシュ動作をメモリ領域属性に指定します。MMU の変換テーブルによって、外部メモリ空間および内蔵空間のキャッシュ動作を設定します。

キャッシュの有効、無効の制御は、以下のレジスタにより行います。

- CP15 System Control Register（SCTLR）：I ビット（b12）、C ビット（b2）
  I ビット　"0"：命令キャッシュを無効、"1"：命令キャッシュを有効
  C ビット　"0"：データキャッシュを無効、"1"：データキャッシュを有効
  SCTLR には MMU の有効、無効を制御する M ビット（b0）も配置されています（"1"が MMU 有効）。
- Control Register（reg1_control）：L2 Cache enable ビット（b0）
  L2 Cache enable ビット　"0"：L2 キャッシュを無効、"1"：L2 キャッシュを有効
  reg1_control は、CoreLink レベル 2 キャッシュコントローラ（L2C-310）のレジスタです。

【注】 1. L1 命令キャッシュおよび L1 データキャッシュを有効にする前に、MMU、L1 命令キャッシュおよび L1 データキャッシュを無効にした状態で、L1 命令キャッシュ、L1 データキャッシュ、TLB をインバリデートする必要があります。また、同様に、L2 キャッシュも有効にする前に、L2 キャッシュをインバリデートする必要があります。
2. ダイレクトメモリアクセスコントローラ（DMAC）は、L1 キャッシュを経由したアクセスは行いません。このため、L1 キャッシュを有効に設定した領域に、CPU と DMAC で共有してアクセスする場合は、L1 キャッシュのキャッシュ操作を行って、ソフトウェアでコヒーレンシを保証するようにしてください。また、DMAC は、L2 キャッシュを経由して外部メモリへのアクセスを行います。

図5.8および図5.9に L1 キャッシュと L2 キャッシュの初期設定フローを示します。

図5.8　L1 キャッシュと L2 キャッシュの初期設定フロー（1/2）

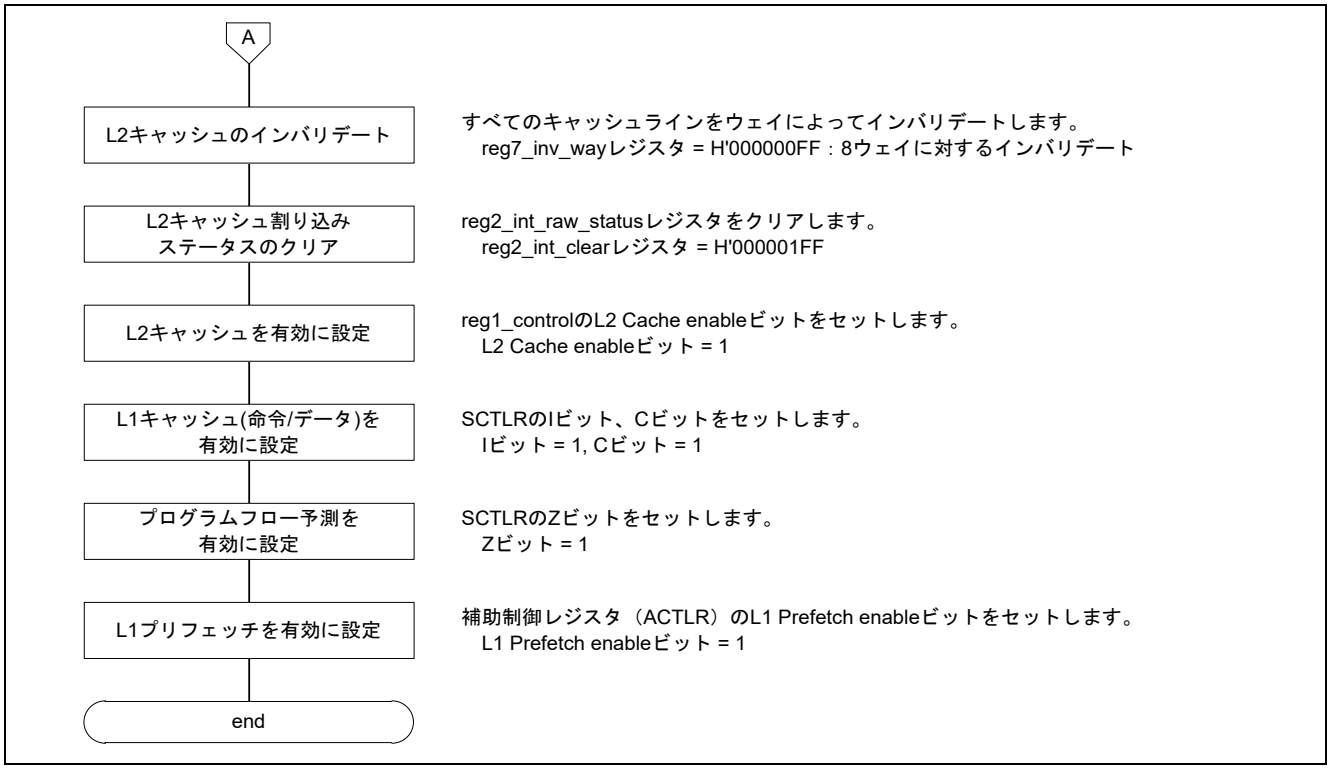


図5.9　L1 キャッシュと L2 キャッシュの初期設定フロー（2/2）

5.2.7 例外処理ベクタテーブル

RZ/A2Mには 7 種類の例外処理（リセット、未定義命令、ソフトウェア割り込み、プリフェッチアボート、データアボート、IRQ、FIQ）があります。サンプルコードでは、アプリケーションプログラムの処理により、ローベクタに切り替え後、VBAR に"__vector_mirror_table_base"が示すアドレスを設定することにより、例外処理ベクタテーブルは"__vector_mirror_table_base"が示すアドレスから 32 バイトの領域に配置されます。各例外には、各例外処理への分岐命令を記述します。図5.10に例外処理ベクタテーブルの記述例として、サンプルコードの例外処理ベクタテーブル内容を示します。

```
vector_table2:
  LDR pc, =reset_handler       ; +0x0000_0000 : Reset exception
  LDR pc, =undefined_handler   ; +0x0000_0004 : Undefined instructions exception
  LDR pc, =svc_handler         ; +0x0000_0008 : Software interrupts exceptions
  LDR pc, =prefetch_handler    ; +0x0000_000c : Prefetch abort exception
  LDR pc, =abort_handler       ; +0x0000_0010 : Data abort exception
  LDR pc, =reserved_handler    ; +0x0000_0014 : Reserved
  LDR pc, =irq_handler         ; +0x0000_0018 : IRQ exception
  LDR pc, =fiq_handler         ; +0x0000_001c : FIQ exception
```

図5.10 例外処理ベクタテーブルの記述例

5.2.8 RTC および USB 未使用チャネルの処理

RZ/A2Mでは、RTC および USB の使用しないチャネルについて、消費電力を低減するために、resetprg 関数の先頭で RTC および USB の各チャネルの未使用処理を実施しています。表5.8に示すRTCおよびUSBの使用/未使用のマクロ定義の値により、未使用時の処理を実行するかどうかを指定します。

表5.8 RTC および USB の使用/未使用のマクロ定義

定数名	設定値	内容
STARTUP_CFG_DISABLE_RTC0	0	RTC チャネル 0 を使用する。
	1 (デフォルト)	RTC チャネル 0 を使用しない。
STARTUP_CFG_DISABLE_RTC1	0	RTC チャネル 1 を使用する。
	1 (デフォルト)	RTC チャネル 1 を使用しない。
STARTUP_CFG_DISABLE_USB0	0	USB チャネル 0 を使用する。
	1 (デフォルト)	USB チャネル 0 を使用しない。
STARTUP_CFG_DISABLE_USB1	0	USB チャネル 1 を使用する。
	1 (デフォルト)	USB チャネル 1 を使用しない。

図5.11にRTC未使用チャネルの処理フローを示します。

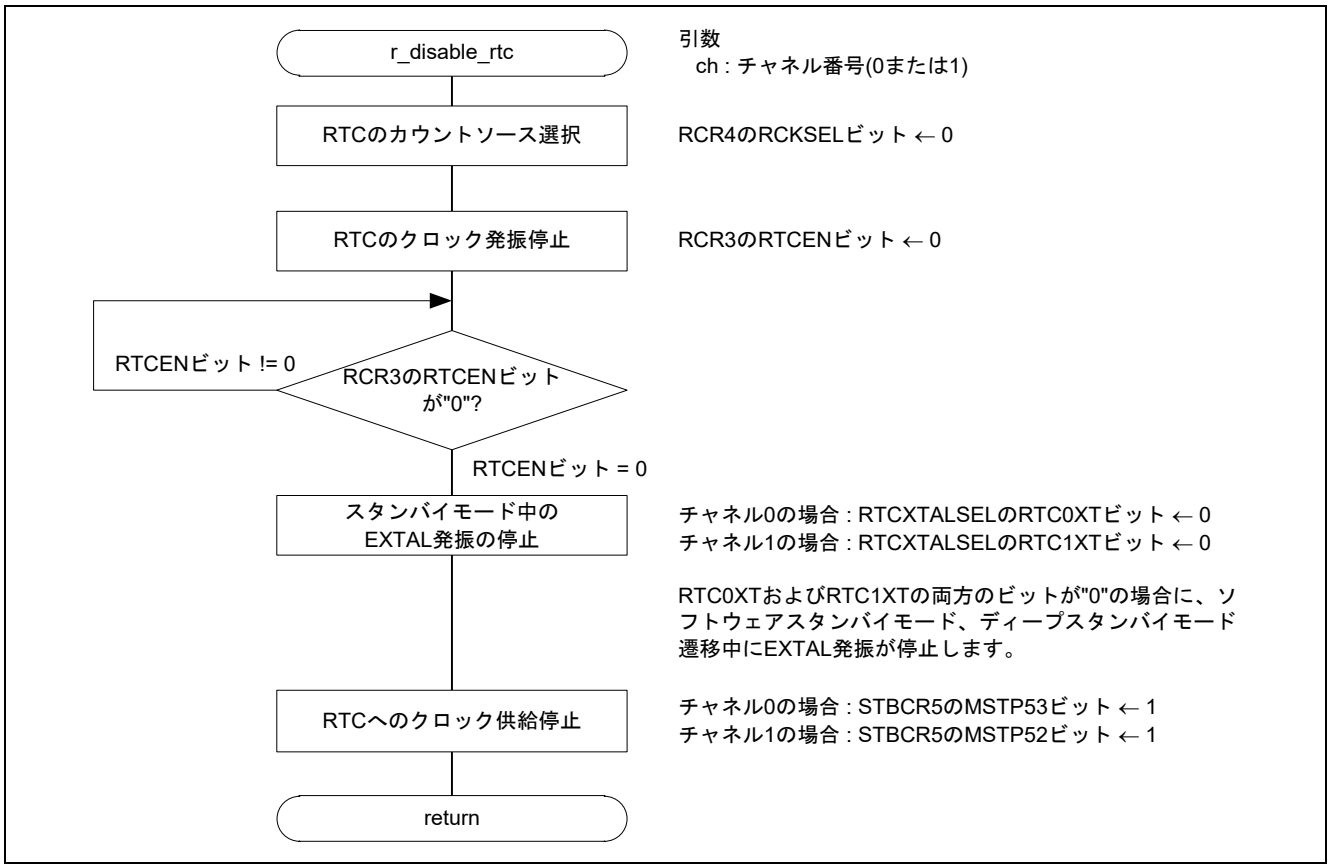


図5.11 RTC 未使用チャネルの処理フロー

図5.12にUSB未使用チャネルの処理フローを示します。

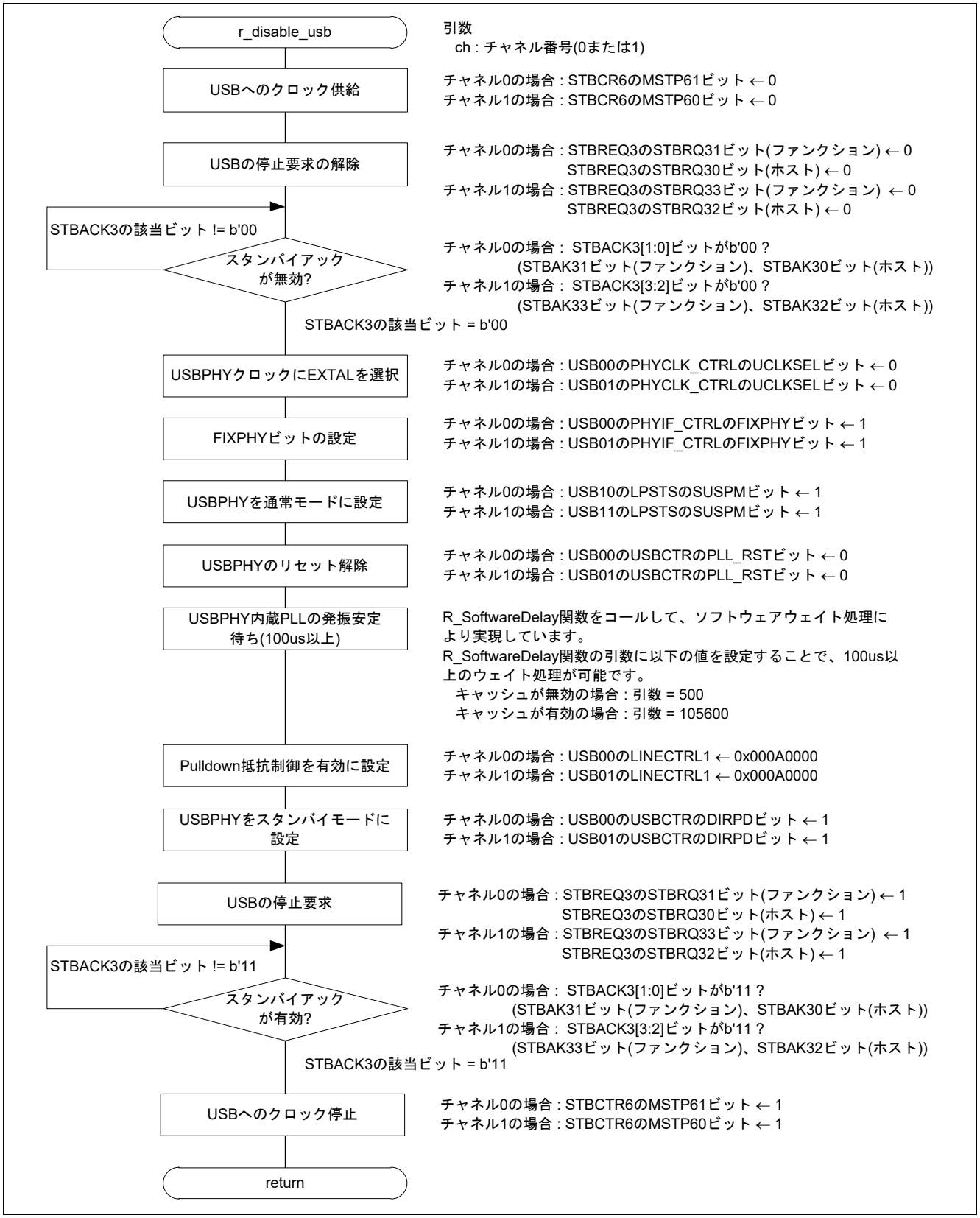


図5.12 USB 未使用チャネルの処理フロー

5.3 使用割り込み一覧

表5.9にサンプルコードで使用する割り込みを示します。

表5.9 サンプルコードで使用する割り込み

割り込み要因（要因 ID）	優先度	処理概要
OSTM0（88）	3	500ms ごとに割り込みを発生
OSTM2（90）	30	1ms ごとに割り込みを発生
RXI4（322）	30	SCIFA の TXI4 割り込みを発生
TXI4（323）	30	SCIFA の RXI4 割り込みを発生

5.4 固定幅整数一覧

表5.10にサンプルコードで使用する固定幅整数を示します。

表5.10 サンプルコードで使用する固定幅整数

シンボル	内容
char_t	8 ビット文字
bool_t	論理型。値は true（1）, false（0）
int_t	高速な整数、符号あり、本サンプルコードでは 32 ビット整数。
int8_t	8 ビット整数、符号あり（標準ライブラリ stdint.h にて定義）
int16_t	16 ビット整数、符号あり（標準ライブラリ stdint.h にて定義）
int32_t	32 ビット整数、符号あり（標準ライブラリ stdint.h にて定義）
int64_t	64 ビット整数、符号あり（標準ライブラリ stdint.h にて定義）
uint8_t	8 ビット整数、符号なし（標準ライブラリ stdint.h にて定義）
uint16_t	16 ビット整数、符号なし（標準ライブラリ stdint.h にて定義）
uint32_t	32 ビット整数、符号なし（標準ライブラリ stdint.h にて定義）
uint64_t	64 ビット整数、符号なし（標準ライブラリ stdint.h にて定義）
float32_t	32 ビット浮動小数
float64_t	64 ビット浮動小数
float128_t	128 ビット浮動小数

5.5 定数一覧

表5.11にサンプルコードで使用する定数を、表5.12、表5.13および表5.14に GPIO ドライバで使用する定数を示します。

表5.11 サンプルコードで使用する定数

定数名	設定値	内容
MAIN_PRV_LED_ON	(1)	LED の点灯
MAIN_PRV_LED_OFF	(0)	LED の消灯

表5.12 GPIO ドライバで使用する定数（端子機能）

定数名	GPIO レジスタ設定値				内容
	PMR	PDR	PODR	PSEL	
GPIO_FUNC_HIZ	0	b'00	-	-	端子機能を不使用（Hi-Z）に設定
GPIO_FUNC_IN	0	b'10	-	-	端子機能を汎用入力ポートに設定
GPIO_FUNC_OUT_HIGH	0	b'11	1	-	端子機能を汎用出力ポート(H レベル)に設定
GPIO_FUNC_OUT_LOW	0	b'11	0	-	端子機能を汎用出力ポート(L レベル)に設定
GPIO_FUNC_PERIPHERAL0	1	b'00	-	0	端子機能を周辺機能 0 に設定
GPIO_FUNC_PERIPHERAL1	1	b'00	-	1	端子機能を周辺機能 1 に設定
GPIO_FUNC_PERIPHERAL2	1	b'00	-	2	端子機能を周辺機能 2 に設定
GPIO_FUNC_PERIPHERAL3	1	b'00	-	3	端子機能を周辺機能 3 に設定
GPIO_FUNC_PERIPHERAL4	1	b'00	-	4	端子機能を周辺機能 4 に設定
GPIO_FUNC_PERIPHERAL5	1	b'00	-	5	端子機能を周辺機能 5 に設定
GPIO_FUNC_PERIPHERAL6	1	b'00	-	6	端子機能を周辺機能 6 に設定
GPIO_FUNC_PERIPHERAL7	1	b'00	-	7	端子機能を周辺機能 7 に設定

表5.13 GPIO ドライバで使用する定数（端子割り込み）

定数名	PmnPFS.ISEL 設定値	内容
GPIO_TINT_DISABLE	0	端子割り込み機能を無効にする
GPIO_TINT_ENABLE	1	端子割り込み機能を有効にする
GPIO_TINT_RESERVED	-	端子割り込み機能が使用できない端子を使用する場合のマクロ定義

表5.14 GPIO ドライバで使用する定数（出力端子駆動能力）

定数名	DSCR 設定値	内容
GPIO_CURRENT_8mA	b'11	端子の駆動能力を 8mA に設定
GPIO_CURRENT_4mA	b'01	端子の駆動能力を 4mA に設定
GPIO_CURRENT_RESERVED	-	駆動能力設定できない端子に指定するリザーブ値

5.6 関数一覧

サンプルコードは、周辺機能を使用するためのインタフェース関数（API 関数）、ユーザシステムの用途に合わせてユーザで準備が必要なユーザ定義関数（API 関数からコールされる関数）、サンプルコードを動作させるために必要なサンプル関数から構成されています。

周辺 IO レジスタをビット単位でアクセスする場合は、サンプルコードでは API 関数の IO レジスタライト関数または IO レジスタリード関数を使用しています。

表5.15にサンプル関数を、表5.16および表5.17に API 関数を、表5.18にユーザ定義関数を示します。

表5.15 サンプル関数

関数名	概要
reset_handler	リセットハンドラ処理（アセンブラ関数）
resetprg	周辺機能の初期設定（INTC、L1 および L2 キャッシュの初期設定）
INITSCT	セクションの初期化（アセンブラ関数）
R_SC_HardwareSetup	周辺機能の初期設定
main	メイン処理
irq_handler	IRQ ハンドラ処理（アセンブラ関数）
INTC_Handler_Interrupt	INTC 割り込みハンドラ処理
fiq_handler	FIQ ハンドラ処理（アセンブラ関数）
NMI_Handler_Interrupt	NMI 割り込みハンドラ処理
direct_open	周辺 IO ドライバ関数のオープン
direct_close	周辺 IO ドライバ関数のクローズ
direct_read	周辺 IO ドライバ関数のリード
direct_write	周辺 IO ドライバ関数のライト
direct_control	周辺 IO ドライバ関数のコントロール
direct_get_version	周辺 IO ドライバ関数のバージョン情報取得
Sample_LED_Blink	OSTM チャネル 0 割り込み処理
r_disable_rtc	RTC の未使用チャネルの設定処理
r_disable_usb	USB の未使用チャネルの設定処理

表5.16 API 関数（1/2）

関数名	概要
R_MMU_Init	MMU の初期設定
R_MMU_WriteTbl	MMU 変換テーブルの設定
R_MMU_ReadTbl	MMU 変換テーブルのリード
R_MMU_VAtoPA	仮想アドレスから物理アドレスへの変換
R_CACHE_L1Init	L1 キャッシュの初期設定
R_CACHE_L1InstEnable	L1 命令キャッシュの有効設定
R_CACHE_L1InstDisable	L1 命令キャッシュの無効設定
R_CACHE_L1InstInvalidAll	すべての L1 命令キャッシュのインバリデート
R_CACHE_L1DataEnable	L1 データキャッシュの有効設定
R_CACHE_L1DataDisable	L1 データキャッシュの無効設定
R_CACHE_L1DataInvalidAll	すべての L1 データキャッシュのインバリデート
R_CACHE_L1DataCleanAll	すべての L1 データキャッシュのクリーニング
R_CACHE_L1DataCleanInvalidAll	すべての L1 データキャッシュのクリーニングおよびインバリデート
R_CACHE_L1DataInvalidLine	L1 データキャッシュのライン（32 バイト）単位のインバリデート
R_CACHE_L1DataCleanLine	L1 データキャッシュのライン（32 バイト）単位のクリーニング
R_CACHE_L1DataCleanInvalidLine	L1 データキャッシュのライン（32 バイト）単位のクリーニングおよびインバリデート
R_CACHE_L1BtacEnable	分岐予測器の有効設定
R_CACHE_L1BtacDisable	分岐予測器の無効設定
R_CACHE_L1BtacInvalidate	分岐予測器のインバリデート
R_CACHE_L1PrefetchEnable	L1 プリフェッチの有効設定
R_CACHE_L1PrefetchDisable	L1 プリフェッチの無効設定
R_CACHE_L2Init	L2 キャッシュの初期設定
R_CACHE_L2CacheEnable	L2 キャッシュの有効設定
R_CACHE_L2CacheDisable	L2 キャッシュの無効設定
R_CACHE_L2InvalidAll	すべての L2 キャッシュのインバリデート
R_CACHE_L2CleanAll	すべての L2 キャッシュのクリーニング
R_CACHE_L2CleanInvalidAll	すべての L2 キャッシュのクリーニングおよびインバリデート
R_INTC_Init	INTC 割り込みコントローラの初期設定
R_INTC_Enable	INTC の割り込みの許可
R_INTC_Disable	INTC の割り込みの禁止
R_INTC_SetPriority	割り込み優先レベルの設定
R_INTC_SetMaskLevel	割り込みマスクレベルの設定
R_INTC_GetMaskLevel	割り込みマスクレベルの取得
R_INTC_RegistIntFunc	割り込みハンドラの登録
R_STB_StartModule	モジュールスタンバイの解除
R_STB_StopModule	モジュールスタンバイへの遷移

表5.17 API 関数（2/2）

関数名	概要
R_CPG_InitialiseHwIf	CPG の初期化処理
R_GPIO_HWInitialise	GPIO ドライバの初期化処理
R_GPIO_InitByPinList	ピンリストによる GPIO 設定
R_GPIO_InitByTable	GPIO コンフィグレーションテーブルによる GPIO 設定
R_GPIO_PinWrite	端子の出力値の設定
R_GPIO_PinRead	端子の入力値の取得
RZA_IO_RegWrite_8	IO レジスタライト関数（8 ビットアクセス可能な IO レジスタ用）
RZA_IO_RegWrite_16	IO レジスタライト関数（16 ビットアクセス可能な IO レジスタ用）
RZA_IO_RegWrite_32	IO レジスタライト関数（32 ビットアクセス可能な IO レジスタ用）
RZA_IO_RegRead_8	IO レジスタリード関数（8 ビットアクセス可能な IO レジスタ用）
RZA_IO_RegRead_16	IO レジスタリード関数（16 ビットアクセス可能な IO レジスタ用）
RZA_IO_RegRead_32	IO レジスタリード関数（32 ビットアクセス可能な IO レジスタ用）

表5.18 ユーザ定義関数

関数名	概要
Userdef_INTC_Pre_Interrupt	INTC 割り込みハンドラの実行前処理
Userdef_INTC_Post_Interrupt	INTC 割り込みハンドラの実行後処理
Userdef_INTC_UndefId	INTC 割り込みの非サポート割り込み ID 受け付け時の処理
Userdef_INTC_UnregisteredID	INTC 割り込みハンドラが未登録の ID 受け付け時の処理
Userdef_PreHardwareSetup	ハードウェア初期化の実行前処理
Userdef_PostHardwareSetup	ハードウェア初期化の実行後処理

5.7 関数仕様

サンプルコードの関数仕様を示します。

reset_handler	
概　要	リセットハンドラ処理
宣　言	reset_handler　FUNCTION {}
説　明	リセット解除後に実行されるアセンブラ関数です。スタックポインタやリセット解除後に最低限必要な周辺機能の初期設定、MMU の初期設定を行い、resetprg をコールします。
引　数	なし
リターン値	なし

resetprg	
概　要	周辺機能の初期設定（INTC、L1 および L2 キャッシュの初期設定）
宣　言	void resetprg(void)
説　明	ハードウェアの初期設定を行い、main 関数をコールします。
引　数	なし
リターン値	なし

INITSCT		
概　要	セクションの初期化	
宣　言	INITSCT　FUNCTION {}	
説　明	引数の DATA セクション初期化テーブルおよび、BSS セクション初期化テーブルに従い、DATA セクションおよび BSS セクションの初期化を行います。	
引　数	r0	DATA セクションの初期化テーブルへのポインタ
	r1	BSS セクションの初期化テーブルへのポインタ
リターン値	なし	

R_SC_HardwareSetup	
概　要	周辺機能の初期設定
宣　言	void R_SC_HardwareSetup(void)
説　明	周辺機能の初期設定を行います。 本サンプルコードでは、Userdef_PreHardwareSetup 関数と Userdef_PostHardwareSetup 関数をコールすることにより､ディープスタンバイモード解除時に端子状態の保持を解除し、保持用内蔵 RAM の各ページへのライトを許可する処理を行います。
引　数	なし
リターン値	なし
注意事項	本関数をコールした時点では、DATA および BSS のセクション領域の初期化を行っていません。

main	
概　要	メイン処理
宣　言	int_t main(void)
説　明	RZ/A2M CPUボードとシリアルインタフェースで接続されたホスト PC 上のターミナルにサンプルコードの情報を表示し、ボード上で LED と接続された GPIO の初期設定を行います。 OSTM チャネル 0 の初期設定を行い、500ms ごとに OSTM チャネル 0 割り込みが発生するようにタイマカウントを設定し起動します。
引　数	なし
リターン値	0

irq_handler	
概　要	IRQ ハンドラ処理
宣　言	irq_handler
説　明	IRQ 割り込み発生時に実行されるアセンブラ関数です。LR_irq, SPSR_irq および汎用レジスタのスタックへの退避と INTC 割り込み発生要因の ID を取得した後、C 言語で記述した INTC_Handler_Interrupt 関数をコールし、発生要因の ID に対応した INTC 割り込みハンドラの処理を行います。 INTC 割り込みハンドラ処理を行った後、汎用レジスタをスタックから復帰し、RFE 命令を実行して LR_irq, SPSR_irq をスタックから復帰し、IRQ 割り込み処理から割り込み発生前の処理にリターンします。 割り込み ID が 1022 および 1023 の場合は、割り込みハンドラ処理は行わず、スタックの退避および復帰の処理のみを行い、割り込み発生前の処理にリターンします。
引　数	なし
リターン値	なし

INTC_Handler_Interrupt	
概　要	INTC 割り込みハンドラ処理
宣　言	void INTC_Handler_Interrupt(uint32_t icciar)
説　明	Irq_handler からコールされる INTC 割り込みハンドラ処理です。 icciar で指定された割り込み ID に対応したハンドラ処理関数 （g_intc_func_table[int_id]に登録された関数）を実行します。 g_intc_func_table[int_id]に割り込みハンドラ関数が登録されていない場合は、Userdef_INTC_UnregisteredID 関数を実行します。割り込みハンドラ処理関数の前後で、Userdef_INTC_Pre_Interrupt 関数および Userdef_INTC_Post_Interrupt 関数を実行します。割り込み ID が 512 以上の場合は、Userdef_INTC_Undefld 関数を実行します。
引　数	uint32_t icciar　　：割り込み ID（0～511）
リターン値	なし

fiq_handler	
概　要	FIQ ハンドラ処理
宣　言	fiq_handler
説　明	FIQ 割り込み発生時に実行されるアセンブラ関数です。LR_fiq, SPSR_fiq および汎用レジスタのスタックへの退避後、C 言語で記述した NMI_Handler_Interrupt 関数をコールし、NMI ハンドラ処理を行います。 NMI ハンドラ処理を行った後、汎用レジスタをスタックから復帰し、RFE 命令を実行して LR_fiq, SPSR_fiq をスタックから復帰し、FIQ 割り込み処理から割り込み発生前の処理にリターンします。
引　数	なし
リターン値	なし

NMI_Handler_Interrupt	
概　要	NMI 割り込みハンドラ処理
宣　言	void NMI_Handler_Interrupt(void)
説　明	fiq_handler からコールされる NMI 割り込みハンドラ処理です。 割り込み ID が 512 の場合のハンドラ処理関数（g_intc_func_table[512]に登録された関数）を実行します。 g_intc_func_table[512]に割り込みハンドラ関数が登録されていない場合は、Userdef_INTC_UnregisteredID 関数を実行します。
引　数	なし
リターン値	なし

direct_open	
概　要	周辺 IO ドライバ関数のオープン
宣　言	int_t direct_open (char_t *p_driver_name, int_t param)
説　明	引数*p_driver_name で指定されたコンフィグレーション名に対応する周辺 IO ドライバの open 関数をコールします（つまり、gs_mount_table に登録されているコンフィグレーション名に対応した周辺 IO ドライバ関数の Open 関数をコールします）。 本関数のリターン値であるハンドル番号を使用して、周辺 IO ドライバ関数のクローズ、リード、ライト、およびコントロールの処理を行います。 サンプルコードでは、本関数を使用して CPG、OSTM0、SCIFA4、および GPIO のドライバ関数をオープンし、初期設定を行っています。
引　数	char_t *p_driver_name　：コンフィグレーション名 int_t param　：オープン属性
リターン値	0 以上　：ハンドル番号 -1　：エラー

direct_close	
概　要	周辺 IO ドライバ関数のクローズ
宣　言	int_t direct_close (int_t handle)
説　明	引数 handle のハンドル番号に対応する周辺 IO ドライバ関数の close 関数をコールします。 サンプルコードでは、OSTM0 および SCIFA4 のドライバ関数をクローズしています。
引　数	int_t handle　：ハンドル番号
リターン値	0 以上　：正常終了 -1　：エラー

direct_read	
概　要	周辺 IO ドライバ関数のリード
宣　言	int_t direct_read (int handle, uint8_t *buff_ptr, uint32_t count)
説　明	引数 handle のハンドル番号に対応する周辺 IO ドライバ関数の read 関数をコールします。read 関数のコール時に、引数*buff_ptr で指定したリードデータの格納バッファのポインタと、引数 count で指定したリードするデータのバイト数 count を渡します。 サンプルコードでは、OSTM0 および SCIFA4 の read 関数をコールしています。
引　数	int handle　：ハンドル番号 uint8_t *buff_ptr　：リードデータ格納バッファのポインタ uint32_t count　：リードするデータのバイト数
リターン値	0 以上　：リードしたデータのバイト数 -1　：エラー

direct_write		
概　要	周辺 IO ドライバ関数のライト	
宣　言	int_t direct_write (int handle, uint8_t *buff_ptr, uint32_t count)	
説　明	引数 handle のハンドル番号に対応する周辺 IO ドライバ関数の write 関数をコールします。write 関数のコール時に、引数*buff_ptr で指定したライトデータの格納バッファのポインタと、引数 count で指定したライトするデータのバイト数 count を渡します。 サンプルコードでは、SCIFA4 の write 関数をコールしています。	
引　数	int handle	：ハンドル番号
	uint8_t *buff_ptr	：ライトデータ格納バッファのポインタ
	uint32_t count	：ライトするデータのバイト数
リターン値	0 以上	：ライトしたデータのバイト数
	-1	：エラー

direct_control		
概　要	周辺 IO ドライバ関数のコントロール	
宣　言	int_t direct_control (int handle, uint32_t ctlCode, void *pCtlStruct)	
説　明	引数 handle のハンドル番号に対応する周辺 IO ドライバ関数の control 関数をコールします。control 関数のコール時に、引数 ctlCode で指定したコントロールコードと、引数*pCtlStruct で指定したコントロールコードに関連する構造体へのポインタを渡します。 サンプルコードでは、OSTM0 タイマのカウント開始、LED に接続された GPIO の初期設定をしています。	
引　数	int handle	：ハンドル番号
	uint32_t ctlCode	：コントロールコード
	void *pCtlStruct	：コントロールコードに関連する構造体のポインタ
リターン値	0 以上	：正常終了
	-1	：エラー

direct_get_version		
概　要	周辺 IO ドライバ関数のバージョン情報取得	
宣　言	int_t direct_get_version(char *p_driver_name, st_ver_info_t *info)	
説　明	引数*p_driver_name で指定されたコンフィグレーション名に対応する周辺 IO ドライバのバージョン情報を返します。	
引　数	char *p_driver_name	：コンフィグレーション名
	st_ver_info_t *info	：バージョン情報を格納するポインタ
リターン値	0 以上	：正常終了
	-1	：エラー

Sample_LED_Blink		
概　要	OSTM チャネル 0 割り込み処理	
宣　言	void Sample_LED_Blink (uint32_t int_sense)	
説　明	OSTM0 の割り込みが受け付けられたときに実行します。 サンプルコードでは、RZ/A2M CPUボード上の LED を 500ms ごとに点滅する処理を行っています。	
引　数	uint32_t int_sense	: 割り込みの検出方法 INTC_LEVEL_SENSITIVE : レベルセンス INTC_EDGE_TRIGGER : エッジセンス
リターン値	なし	

r_disable_rtc		
概　要	RTC の未使用チャネルの設定処理	
宣　言	static void r_disable_rtc (uint32_t ch)	
説　明	「図5.11　RTC未使用チャネルの処理フロー」に記載されている RTC 未使用チャネルの設定処理を行います。	
引　数	uint32_t ch	: RTC のチャネル番号（0 または 1）
リターン値	なし	

r_disable_usb		
概　要	USB 未使用チャネルの設定処理	
宣　言	static void r_disable_usb (uint32_t ch)	
説　明	「図5.12　USB未使用チャネルの処理フロー」に記載されている USB 未使用チャネルの設定処理を行います。	
引　数	uint32_t ch	: USB のチャネル番号（0 または 1）
リターン値	なし	

R_MMU_Init	
概　要	MMU の変換テーブルの初期設定
宣　言	void R_MMU_Init(void)
説　明	MMU の変換テーブルの初期設定を行います。 変換テーブルは、MMU_SC_TABLE のコンフィグレーションテーブルにより、第 1 レベル記述子のセクションタイプで設定します。
引　数	なし
リターン値	なし

R_MMU_WriteTbl	
概　要	MMU 変換テーブルの設定
宣　言	e_mmu_err_t R_MMU_WriteTbl (uint32_t vaddress, uint32_t paddress, uint32_t size, uint32_t entry)
説　明	引数で指定された仮想アドレス空間の第 1 レベル MMU 変換テーブルエントリに対し、物理メモリアドレスを割り当て、領域属性を設定します。 表5.5および表5.6に変換テーブルに設定する第 1 レベル記述子の内容を示します。
引　数	uint32_t vaddress　：設定する仮想アドレスの先頭（b31-b20 のみ有効） uint32_t paddress　：割当先の物理アドレスの先頭（b31-b20 のみ有効） uint32_t size　：設定するエントリ数の範囲（1~4096） uint32_t entry　：設定する属性（b19-b0 のみ有効）
リターン値	MMU_SUCCESS　：正常終了 MMU_ERR_OVERFLOW　：サイズオーバフロー
注意事項	R_MMU_Init 関数をコール後に本関数を呼び出してください。

R_MMU_ReadTbl	
概　要	MMU 変換テーブルのリード
宣　言	uint32_t R_MMU_ReadTbl (uint32_t address)
説　明	引数で指定された仮想アドレスに対応するエントリデータ（第 1 レベル MMU 変換子フォーマット）をリードします。
引　数	uint32_t address　:仮想アドレス（b31-b20 のみ有効）
リターン値	仮想アドレスに対応するエントリデータ
注意事項	R_MMU_Init 関数をコール後に本関数を呼び出してください。

R_MMU_VAtoPA	
概　要	仮想アドレスから物理アドレスへの変換
宣　言	e_mmu_err_t R_MMU_VAtoPA (uint32_t vaddress, uint32_t * paddress)
説　明	引数で指定された仮想アドレスを物理アドレスに変換します。
引　数	uint32_t vaddress　　：仮想アドレス uint32_t * paddress　　：物理アドレスの格納先
リターン値	MMU_SUCCESS　　：正常終了 MMU_ERR_TRANSLATION　　：変換エラー
注意事項	R_MMU_Init 関数をコール後に本関数を呼び出してください。

R_CACHE_L1Init	
概　要	L1 キャッシュの初期設定
宣　言	void R_CACHE_L1Init(void)
説　明	L1 キャッシュ、プログラムフロー予測、および L1 プリフェッチを有効に設定します。
引　数	なし
リターン値	なし

R_CACHE_L1InstEnable	
概　要	L1 命令キャッシュの有効設定
宣　言	void R_CACHE_L1InstEnable(void)
説　明	L1 命令キャッシュを有効に設定します。
引　数	なし
リターン値	なし

R_CACHE_L1InstDisable	
概　要	L1 命令キャッシュの無効設定
宣　言	void R_CACHE_L1InstDisable(void)
説　明	L1 命令キャッシュを無効に設定します。
引　数	なし
リターン値	なし

R_CACHE_L1InstInvalidAll	
概　要	すべての L1 命令キャッシュのインバリデート
宣　言	void R_CACHE_L1InstInvalidAll(void)
説　明	すべての L1 命令キャッシュのインバリデートを行います。
引　数	なし
リターン値	なし

R_CACHE_L1DataEnable	
概　要	L1 データキャッシュの有効設定
宣　言	void R_CACHE_L1DataEnable(void)
説　明	L1 データキャッシュを有効に設定します。
引　数	なし
リターン値	なし

R_CACHE_L1DataDisable	
概　要	L1 データキャッシュの無効設定
宣　言	void R_CACHE_L1DataDisable(void)
説　明	L1 データキャッシュを無効に設定します。
引　数	なし
リターン値	なし

R_CACHE_L1DataInvalidAll	
概　要	すべての L1 データキャッシュのインバリデート
宣　言	void R_CACHE_L1DataInvalidAll(void)
説　明	すべての L1 データキャッシュのインバリデートを行います。
引　数	なし
リターン値	なし

R_CACHE_L1DataCleanAll	
概　要	すべての L1 データキャッシュのクリーニング
宣　言	void R_CACHE_L1DataCleanAll(void)
説　明	すべての L1 データキャッシュのクリーニングを行います。
引　数	なし
リターン値	なし

R_CACHE_L1DataCleanInvalidAll	
概　要	すべての L1 データキャッシュのクリーニングおよびインバリデート
宣　言	void R_CACHE_L1DataCleanInvalidAll(void)
説　明	すべての L1 データキャッシュのクリーニングおよびインバリデートを行います。
引　数	なし
リターン値	なし

R_CACHE_L1DataInvalidLine		
概　要	L1 データキャッシュのライン（32 バイト）単位のインバリデート	
宣　言	e_err_code_t R_CACHE_L1DataInvalidLine(void* line_addr, uint32_t size)	
説　明	引数 line_addr で指定されたアドレスから引数 size で指定された領域について、L1 データキャッシュをキャッシュライン単位でインバリデートします。	
引　数	void* line_addr	：インバリデートする先頭アドレス
	uint32_t size	：インバリデートするバイト数
リターン値	DRV_SUCCESS	：成功
	DRV_ERROR	：失敗

R_CACHE_L1DataCleanLine		
概　要	L1 データキャッシュのライン（32 バイト）単位のクリーニング	
宣　言	e_err_code_t R_CACHE_L1DataCleanLine(void* line_addr, uint32_t size)	
説　明	引数 line_addr で指定されたアドレスから引数 size で指定された領域について、L1 データキャッシュをキャッシュライン単位でクリーニングします。	
引　数	void* line_addr	：クリーニングする先頭アドレス
	uint32_t size	：クリーニングするバイト数
リターン値	DRV_SUCCESS	：成功
	DRV_ERROR	：失敗

R_CACHE_L1DataCleanInvalidLine		
概　要	L1 データキャッシュのライン（32 バイト）単位のクリーニングおよびインバリデート	
宣　言	e_err_code_t R_CACHE_L1DataCleanInvalidLine(void* line_addr, uint32_t size)	
説　明	引数 line_addr で指定されたアドレスから引数 size で指定された領域について、L1 データキャッシュをキャッシュライン単位でクリーニングおよびインバリデートします。	
引　数	void* line_addr	：クリーニングおよびインバリデートする先頭アドレス
	uint32_t size	：クリーニングおよびインバリデートするバイト数
リターン値	DRV_SUCCESS	：成功
	DRV_ERROR	：失敗

R_CACHE_L1BtacEnable	
概　要	分岐アドレスターゲットキャッシュの有効設定
宣　言	void R_CACHE_L1BtacEnable(void)
説　明	分岐アドレスターゲットキャッシュを有効に設定します。
引　数	なし
リターン値	なし

R_CACHE_L1BtacDisable	
概　要	分岐アドレスターゲットキャッシュの無効設定
宣　言	void R_CACHE_L1BtacDisable(void)
説　明	分岐アドレスターゲットキャッシュを無効に設定します。
引　数	なし
リターン値	なし

R_CACHE_L1BtacInvalidate	
概　要	分岐アドレスターゲットキャッシュのインバリデート
宣　言	void R_CACHE_L1BtacInvalidate(void)
説　明	分岐アドレスターゲットキャッシュをインバリデートします。
引　数	なし
リターン値	なし

R_CACHE_L1PrefetchEnable

概　要	L1 プリフェッチの有効設定
宣　言	void R_CACHE_L1PrefetchEnable(void)
説　明	プリフェッチを有効に設定します。
引　数	なし
リターン値	なし

R_CACHE_L1PrefetchDisable

概　要	L1 プリフェッチの無効設定
宣　言	void R_CACHE_L1PrefetchDisable(void)
説　明	プリフェッチを有効に設定します。
引　数	なし
リターン値	なし

R_CACHE_L2Init

概　要	L2 キャッシュの初期設定
宣　言	void R_CACHE_L2Init(void)
説　明	以下の手順にて、L2 キャッシュの初期設定を行います。 1. L2 キャッシュの無効化 2. L2 キャッシュのすべてのキャッシュラインをインバリデート 3. L2 キャッシュコントローラの割り込みステータスのクリア 4. L2 キャッシュの有効化
引　数	なし
リターン値	なし

R_CACHE_L2CacheEnable

概　要	L2 キャッシュの有効設定
宣　言	void R_CACHE_L2CacheEnable(void)
説　明	L2 キャッシュを有効に設定します。
引　数	なし
リターン値	なし

R_CACHE_L2_CacheDisable

概　要	L2 キャッシュの無効設定
宣　言	void R_CACHE_L2CacheDisable(void)
説　明	L2 キャッシュを無効に設定します。
引　数	なし
リターン値	なし

R_CACHE_L2InvalidAll

概　要	すべての L2 キャッシュのインバリデート
宣　言	void R_CACHE_L2InvalidAll(void)
説　明	ウェイにより、すべての L2 キャッシュのインバリデートを行います。
引　数	なし
リターン値	なし

R_CACHE_L2CleanAll

概　要	すべての L2 キャッシュのクリーニング
宣　言	void R_CACHE_L2CleanAll(void)
説　明	ウェイにより、すべての L2 キャッシュのクリーニングを行います。
引　数	なし
リターン値	なし

R_CACHE_L2CleanInvalidAll

概　要	すべての L2 キャッシュのクリーニングおよびインバリデート
宣　言	void R_CACHE_L2CleanInvalidAll(void)
説　明	ウェイにより、すべての L2 キャッシュのクリーニングおよびインバリデートを行います。
引　数	なし
リターン値	なし

R_INTC_Init	
概　要	INTC 割り込みコントローラの初期設定
宣　言	e_r_drv_intc_err_t R_INTC_Init(void)
説　明	GIC 割り込みコントローラの初期設定を行います。 割り込み優先レベルが 0～30 の割り込みを受け付けるようにしています。
引　数	なし
リターン値	INTC_SUCCESS　：正常終了

R_INTC_Enable	
概　要	INTC の割り込みの許可
宣　言	e_r_drv_intc_err_t R_INTC_Enable(e_r_drv_intc_intid_t int_id)
説　明	引数 int_id で指定した割り込み ID の割り込みを許可します。
引　数	e_r_drv_intc_intid_t int_id　：割り込み ID（0～511）
リターン値	INTC_SUCCESS　：正常終了 INTC_ERR_INVALID_ID　：割り込み ID が不正

R_INTC_Disable	
概　要	INTC の ID の割り込みの禁止
宣　言	e_r_drv_intc_err_t R_INTC_Disable(e_r_drv_intc_intid_t int_id)
説　明	引数 int_id で指定した割り込み ID の割り込みを禁止します。
引　数	e_r_drv_intc_intid_t int_id　：割り込み ID（0～511）
リターン値	INTC_SUCCESS　：正常終了 INTC_ERR_INVALID_ID　：割り込み ID が不正

R_INTC_SetPriority	
概　要	割り込み優先レベルの設定
宣　言	e_r_drv_intc_err_t R_INTC_SetPriority(e_r_drv_intc_intid_t int_id, intc_priority_t priority)
説　明	引数 int_id で指定された ID の割り込み優先レベルを引数 priority で指定した値に設定します。
引　数	e_r_drv_intc_intid_t int_id　：割り込み ID（0～511） intc_priority_t priority　：優先レベル（0～31（0：最高レベル、31：最低レベル））
リターン値	INTC_SUCCESS　：正常終了 INTC_ERR_INVALID_ID　：割り込み ID が不正 INTC_ERR_INVALID_PRIORITY：優先レベルが不正
注意事項	割り込み優先レベルに 31、INTC_PRIORITY_LOWEST または INTC_PRIORITY_31 を設定した場合、該当の割り込みは発生しません。

R_INTC_SetMaskLevel	
概　要	割り込みマスクレベルの設定
宣　言	e_r_drv_intc_err_t R_INTC_SetMaskLevel(e_r_drv_intc_priority_t mask_level)
説　明	引数 mask_level で指定された割り込みマスクレベルを割り込み優先度マスクレジスタに設定します。
引　数	e_r_drv_intc_priority_t mask_level　：マスクレベル（0～31（0：最高レベル, 31：最低レベル））
リターン値	INTC_SUCCESS　：正常終了 INTC_ERR_INVALID_PRIORITY：優先レベルが不正
注意事項	指定されたマスクレベルよりも優先レベルの低い割り込み（優先レベルが同じかそれ以上の値）をマスクします。

R_INTC_GetMaskLevel	
概　要	割り込みマスクレベルの取得
宣　言	e_r_drv_intc_err_t R_INTC_GetMaskLevel(e_r_drv_intc_priority_t *mask_level)
説　明	割り込み優先度マスクレジスタの値を取得し、引数*mask_level に格納します。
引　数	e_r_drv_intc_priority_t *mask_level　：マスクレベルを格納するポインタ （0～31（0：最高レベル, 31：最低レベル））
リターン値	INTC_SUCCESS　：正常終了

R_INTC_RegistIntFunc	
概　要	割り込みハンドラの登録関数
宣　言	e_r_drv_intc_err_t R_INTC_RegistIntFunc(e_r_drv_intc_intid_t int_id, void (* func)(uint32_t int_sense))
説　明	引数 int_id の割り込み ID に対する割り込み関数 func を割り込み関数テーブル（g_intc_func_table[int_id]）に登録します。 int_id に 512 が指定された場合は、NMI 発生時に使用する割り込み関数を g_intc_func_table[int_id]に登録します。 割り込み ID が不正な場合は、割り込みハンドラの登録は行いません。
引　数	e_r_drv_intc_intid_t int_id　：割り込み ID（0～512） void (* func) (uint32_t int_sense)　：割り込み関数のポインタ（引数：uint32_t int_sense）
リターン値	INTC_SUCCESS　：正常終了 INTC_ERR_INVALID　：異常終了

R_STB_StartModule	
概　要	モジュールスタンバイの解除
宣　言	int_t R_STB_StartModule(e_stb_module_t module)
説　明	引数 module で指定されたモジュールのモジュールスタンバイ状態を解除します。 本関数をコールすることにより、指定したモジュールに対して、クロックを供給します。
引　数	e_stb_module_t module　：モジュール番号
リターン値	DRV_SUCCESS　：正常動作 DRV_ERROR　：異常動作

R_STB_StopModule	
概　要	モジュールスタンバイへの遷移
宣　言	int_t R_STB_StopModule(e_stb_module_t module)
説　明	引数 module で指定されたモジュールをモジュールスタンバイ状態に遷移します。 本関数をコールすることにより、指定したモジュールに対して、クロックの供給を停止します。
引　数	e_stb_module_t module　：モジュール番号
リターン値	DRV_SUCCESS　：正常動作 DRV_ERROR　：異常動作

R_CPG_InitialiseHwIf	
概　要	CPG の初期化処理
宣　言	int_t R_CPG_InitialiseHwIf(void)
説　明	本関数は､r_cpg_drv_sc_cfg.h の CPG コンフィグレーションデータを使用して､CPG レジスタ（FRQCR、CKIOSEL、SCLKSEL）の設定処理を行います。 本サンプルコードでは、direct_open 関数を経由して本関数がコールされ、「表2.1 動作確認条件（1/2）」に記載されている動作周波数になります。
引　数	なし
リターン値	DRV_SUCCESS　　：正常終了 DRV_ERROR　：r_cpg_drv_sc_cfg.h の CPG コンフィグレーションデータが不正

R_GPIO_HWInitialise	
概　要	GPIO ドライバの初期化処理
宣　言	int_t R_GPIO_HWInitialise(void)
説　明	GPIO ドライバの初期化処理を行います。また、GPIO ドライバの初期化後、r_gpio_drv_sc_cfg.h の GPIO_SC_TABLE_INIT[]のデータを参照して、端子設定処理を実行します。 本サンプルコードでは、direct_open 関数により GPIO ドライバ関数をオープン時にコールされていますが、GPIO_SC_TABLE_INIT[]のデータが定義されていないため、本関数による端子の設定処理は行われません。 本関数は、main 関数にある GPIO ドライバのオープン処理内でコールされます。
引　数	なし
リターン値	DRV_SUCCESS　　：正常終了
注意事項	GPIO ドライバは、本関数により端子機能を割り当てられた端子を「使用中」の状態として管理します。GPIO ドライバでは、「使用中」の状態として管理されている端子は、他の端子機能に再設定することができません。端子機能を再設定する場合、事前に R_GPIO_ClearByPinList 関数により端子を「未使用」状態にしてください。

R_GPIO_InitByPinList	
概　要	ピンリストによる GPIO 設定
宣　言	e_r_drv_gpio_err_t R_GPIO_PinInitByPinList(const r_gpio_port_pin_t * p_pin_list, uint32_t count)
説　明	ピンリストによる GPIO の設定を行います。 GPIO_SC_TABLE_INIT[]または GPIO_SC_TABLE_MANUAL[]にある設定データを参照して、引数 p_pin_list で指定した端子設定処理を行います。 サンプルコードでは、direct_control 関数によりコールされ、RZ/A2M CPUボード上の LED を点灯および消灯させることができるように P6_0 の設定を行っています。
引　数	const r_gpio_port_pin_t *　：pin リストのポインタ p_pin_list uint32_t count　　：設定する pin 数
リターン値	GPIO_SUCCESS　　：正常終了 GPIO_SUCCESS 以外　：異常終了
注意事項	GPIO ドライバは、本関数により端子機能を割り当てられた端子を「使用中」の状態として管理します。GPIO ドライバでは、「使用中」の状態として管理されている端子は、他の端子機能に再設定することができません。端子機能を再設定する場合、事前に R_GPIO_ClearByPinList 関数により端子を「未使用」状態にしてください。

R_GPIO_InitByTable	
概　要	GPIO コンフィグレーションテーブルによる GPIO 設定
宣　言	e_r_drv_gpio_err_t R_GPIO_PinInitByTable(const st_r_drv_gpio_sc_config_t * p_table, uint32_t count)
説　明	引数 p_table により GPIO のコンフィグレーションテーブルを指定して、端子設定処理を行います。 サンプルコードでは、SCIFA ドライバのオープン時にコールされ、P9_0 および P9_1 端子を TxD4 および RxD4 機能に設定しています。
引　数	const st_r_drv_gpio_sc_config_t * p_table : GPIO コンフィグレーションテーブルのポインタ uint32_t count : 設定する端子数
リターン値	GPIO_SUCCESS : 正常終了 GPIO_SUCCESS 以外 : 異常終了
注意事項	GPIO ドライバは、本関数により端子機能を割り当てられた端子を「使用中」の状態として管理します。GPIO ドライバでは、「使用中」の状態として管理されている端子は、他の端子機能に再設定することができません。端子機能を再設定する場合、事前に R_GPIO_ClearByPinList 関数により端子を「未使用」状態にしてください。

R_GPIO_PinWrite	
概　要	端子の出力値の設定
宣　言	e_r_drv_gpio_err_t R_GPIO_PinWrite(e_r_drv_gpio_pin_t pin, e_r_drv_gpio_level_t level)
説　明	引数 pin で指定された端子が汎用出力機能に設定されている場合に 引数 pin で指定された端子に引数 level で指定された値を設定します。
引　数	e_r_drv_gpio_pin_t pin : ピン番号 e_r_drv_gpio_level_t level : 出力レベル GPIO_LEVEL_LOW : 1 GPIO_LEVEL_HIGH : 2
リターン値	GPIO_SUCCESS : 正常終了 GPIO_SUCCESS 以外 : 異常終了

R_GPIO_PinRead	
概　要	端子の入力値の取得
宣　言	e_r_drv_gpio_err_t R_GPIO_PinRead(e_r_drv_gpio_pin_t pin, e_r_drv_gpio_level_t *p_level)
説　明	引数 pin で指定された端子が汎用入力機能に設定されている場合に 引数 pin で指定された端子の値を取得し、引数*p_level に取得します。
引　数	e_r_drv_gpio_pin_t pin　：ピン番号 e_r_drv_gpio_level_t *p_level　：取得レベル格納先 GPIO_LEVEL_LOW : 1 GPIO_LEVEL_HIGH : 2
リターン値	GPIO_SUCCESS　：正常終了 GPIO_SUCCESS 以外　：異常終了

RZA_IO_RegWrite_8	
概　要	IO レジスタライト関数（8 ビットアクセス可能な IO レジスタ用）
宣　言	void RZA_IO_RegWrite_8(volatile uint8_t * ioreg, uint8_t write_value, uint8_t shift, uint32_t mask)
説　明	8 ビットアクセス可能な周辺 IO レジスタへのライト処理を行います。
引　数	volatile uint8_t *ioreg　：ライトする IO レジスタ iodefines 以下のディレクトリで定義された IO レジスタを指定してください。 uint8_t write_value　：IO レジスタへのライト値 uint8_t shift　：IO レジスタの対象ビットへの左シフト値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をシフト値として定義しています。定義された IO レジスタへのシフト値を指定してください。 uint32_t mask　：IO レジスタの対象ビットのマスク値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をマスク値として定義しています。定義された IO レジスタのマスク値を指定してください。 「IOREG_NONMASK_ACCESS」を指定することで、マスク処理（リードモディファイライト処理）を行わずに直接 IO レジスタにライトします。
リターン値	なし
注意事項	本関数を RAM 領域に転送して使用する場合､本関数の配置されたセクションが ROM 領域から RAM 領域へ転送が完了する前に、本関数をコールしないでください。

RZA_IO_RegWrite_16

概　要	IO レジスタライト関数（16 ビットアクセス可能な IO レジスタ用）
宣　言	void RZA_IO_RegWrite_16(volatile uint16_t * ioreg, uint16_t write_value, uint16_t shift, uint32_t mask)
説　明	16 ビットアクセス可能な周辺 IO レジスタへのライト処理を行います。
引　数	volatile uint16_t *ioreg : ライトする IO レジスタ iodefines 以下のディレクトリで定義された IO レジスタを指定してください。
	uint16_t write_value : IO レジスタへのライト値
	uint16_t shift : IO レジスタの対象ビットへの左シフト値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をシフト値として定義しています。定義された IO レジスタへのシフト値を指定してください。
	uint32_t mask : IO レジスタの対象ビットのマスク値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をマスク値として定義しています。定義された IO レジスタのマスク値を指定してください。 「IOREG_NONMASK_ACCESS」を指定することで、マスク処理（リードモディファイライト処理）を行わずに直接 IO レジスタにライトします。
リターン値	なし
注意事項	本関数を RAM 領域に転送して使用する場合､本関数の配置されたセクションが ROM 領域から RAM 領域へ転送が完了する前に、本関数をコールしないでください。

RZA_IO_RegWrite_32	
概　要	IO レジスタライト関数（32 ビットアクセス可能な IO レジスタ用）
宣　言	void RZA_IO_RegWrite_32(volatile uint32_t * ioreg, uint32_t write_value, uint32_t shift, uint32_t mask)
説　明	32 ビットアクセス可能な周辺 IO レジスタへのライト処理を行います。
引　数	volatile uint32_t *ioreg　: ライトする IO レジスタ iodefines 以下のディレクトリで定義された IO レジスタを指定してください。
	uint32_t write_value　: IO レジスタへのライト値
	uint32_t shift　: IO レジスタの対象ビットへの左シフト値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をシフト値として定義しています。定義された IO レジスタへのシフト値を指定してください。
	uint32_t mask　: IO レジスタの対象ビットのマスク値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をマスク値として定義しています。定義された IO レジスタのマスク値を指定してください。 「IOREG_NONMASK_ACCESS」を指定することで、マスク処理（リードモディファイライト処理）を行わずに直接 IO レジスタにライトします。
リターン値	なし
注意事項	本関数を RAM 領域に転送して使用する場合､本関数の配置されたセクションが ROM 領域から RAM 領域へ転送が完了する前に、本関数をコールしないでください。

RZA_IO_RegRead_8		
概　要	IO レジスタリード関数（8 ビットアクセス可能な IO レジスタ用）	
宣　言	uint8_t RZA_IO_RegRead_8(volatile uint8_t * ioreg, uint8_t shift, uint32_t mask)	
説　明	8 ビットアクセス可能な周辺 IO レジスタへのリード処理を行います。	
引　数	volatile uint8_t *ioreg	: リードする IO レジスタ iodefines 以下のディレクトリで定義された IO レジスタを指定してください。
	uint8_t shift	: IO レジスタの対象ビットへの左シフト値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をシフト値として定義しています。定義された IO レジスタへのシフト値を指定してください。
	uint32_t mask	: IO レジスタの対象ビットのマスク値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をマスク値として定義しています。定義された IO レジスタのマスク値を指定してください。 「IOREG_NONMASK_ACCESS」を指定することで、マスク処理を行わずに直接 IO レジスタをリードします。
リターン値	指定した IO レジスタのリード値	
注意事項	本関数を RAM 領域に転送して使用する場合､本関数の配置されたセクションが ROM 領域から RAM 領域へ転送が完了する前に、本関数をコールしないでください。	

RZA_IO_RegRead_16		
概　要	IO レジスタリード関数（16 ビットアクセス可能な IO レジスタ用）	
宣　言	uint16_t RZA_IO_RegRead_16(volatile uint16_t * ioreg, uint16_t shift, uint32_t mask)	
説　明	16 ビットアクセス可能な周辺 IO レジスタへのリード処理を行います。	
引　数	volatile uint16_t *ioreg	: リードする IO レジスタ iodefines 以下のディレクトリで定義された IO レジスタを指定してください。
	uint16_t shift	: IO レジスタの対象ビットへの左シフト値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をシフト値として定義しています。定義された IO レジスタへのシフト値を指定してください。
	uint32_t mask	: IO レジスタの対象ビットのマスク値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をマスク値として定義しています。定義された IO レジスタのマスク値を指定してください。 「IOREG_NONMASK_ACCESS」を指定することで、マスク処理を行わずに直接 IO レジスタをリードします。
リターン値	指定した IO レジスタのリード値	
注意事項	本関数を RAM 領域に転送して使用する場合､本関数の配置されたセクションが ROM 領域から RAM 領域へ転送が完了する前に、本関数をコールしないでください。	

RZA_IO_RegRead_32	
概　要	IO レジスタリード関数（32 ビットアクセス可能な IO レジスタ用）
宣　言	uint32_t RZA_IO_RegRead_32(volatile uint32_t * ioreg, uint32_t shift, uint32_t mask)
説　明	32 ビットアクセス可能な周辺 IO レジスタへのリード処理を行います。
引　数	volatile uint32_t *ioreg　：リードする IO レジスタ iodefines 以下のディレクトリで定義された IO レジスタを指定してください。
	uint32_t shift　：IO レジスタの対象ビットへの左シフト値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をシフト値として定義しています。定義された IO レジスタへのシフト値を指定してください。
	uint32_t mask　：IO レジスタの対象ビットのマスク値 iobitmasks 以下のディレクトリで、IO レジスタ内のビット位置にアクセスするための値をマスク値として定義しています。定義された IO レジスタのマスク値を指定してください。 「IOREG_NONMASK_ACCESS」を指定することで、マスク処理を行わずに直接 IO レジスタをリードします。
リターン値	指定した IO レジスタのリード値
注意事項	本関数を RAM 領域に転送して使用する場合､本関数の配置されたセクションが ROM 領域から RAM 領域へ転送が完了する前に、本関数をコールしないでください。

Userdef_INTC_Pre_Interrupt	
概　要	INTC 割り込みハンドラ処理の実行前の処理
宣　言	e_r_drv_intc_err_t Userdef_INTC_Pre_Interrupt(e_r_drv_intc_intid_t int_id)
説　明	INTC_Handler_Interrupt からコールされるユーザ定義関数です。INTC 割り込みハンドラ処理を実行する前に必要な処理を実装します。サンプルコードでは、処理をせずにリターンします。
引　数	e_r_drv_intc_intid_t int_id　：割り込み ID（0～511）
リターン値	INTC_SUCCESS　：正常終了

Userdef_INTC_Post_Interrupt	
概　要	INTC 割り込みハンドラ処理の実行後の処理
宣　言	e_r_drv_intc_err_t Userdef_INTC_Post_Interrupt(e_r_drv_intc_intid_t int_id)
説　明	INTC_Handler_Interrupt からコールされるユーザ定義関数です。INTC 割り込みハンドラ処理が実行した後に必要な処理を実装します。サンプルコードでは、処理をせずにリターンします。
引　数	e_r_drv_intc_intid_t int_id　：割り込み ID（0～511）
リターン値	INTC_SUCCESS　：正常終了

Userdef_INTC_UndefId	
概　要	INTC 割り込みの非サポート割り込み ID 受け付け時の処理
宣　言	e_r_drv_intc_err_t Userdef_INTC_UndefId(e_r_drv_intc_intid_t int_id)
説　明	INTC_Handler_Interrupt からコールされるユーザ定義関数です。割り込み ID が 512 以上の割り込み ID の割り込みが受け付けられた場合に実行されます。サンプルコードでは、無限ループの処理を実装しています。
引　数	e_r_drv_intc_intid_t int_idint_id　：割り込み ID（512～1021）
リターン値	INTC_SUCCESS　：正常終了

Userdef_INTC_UnregisteredID	
概　要	INTC 割り込みハンドラが未登録の ID 受け付け時の処理
宣　言	e_r_drv_intc_err_t Userdef_INTC_UnregisteredID(e_r_drv_intc_intid_t int_id)
説　明	INTC_Handler_Interrupt からコールされるユーザ定義関数です。割り込みハンドラが未登録の ID 受け付け時の処理が実装されています。サンプルは、CPU への IRQ 割り込み禁止後、無限ループの処理を実装しています。
引　数	e_r_drv_intc_intid_t int_id　：割り込み ID（0～511）
リターン値	INTC_SUCCESS　：正常終了

Userdef_PreHardwareSetup	
概　要	ハードウェア初期化の実行前処理
宣　言	void Userdef_PreHardwareSetup (void)
説　明	ハードウェア初期化の実行前に行う処理を記述するためのユーザ定義関数です。 R_SC_HardwareSetup 関数の先頭でコールされます。 アプリケーションプログラムでは、IOKEEP ビットのクリアを行い、ディープスタンバイから復帰した際に保持されている端子の状態を解除します。
引　数	なし
リターン値	なし

Userdef_PostHardwareSetup	
概　要	ハードウェア初期化の実行後処理
宣　言	void Userdef_PreHardwareSetup (void)
説　明	ハードウェア初期化の実行後に行う処理を記述するためのユーザ定義関数です。 R_SC_HardwareSetup 関数の終端でコールされます。 アプリケーションプログラムでは、保持用内蔵 RAM 各ページのライトイネーブルをライト有効に設定します。
引　数	なし
リターン値	なし

5.8 フローチャート

5.8.1 リセットハンドラ処理

図5.13にリセットハンドラ処理のフローチャートを示します。

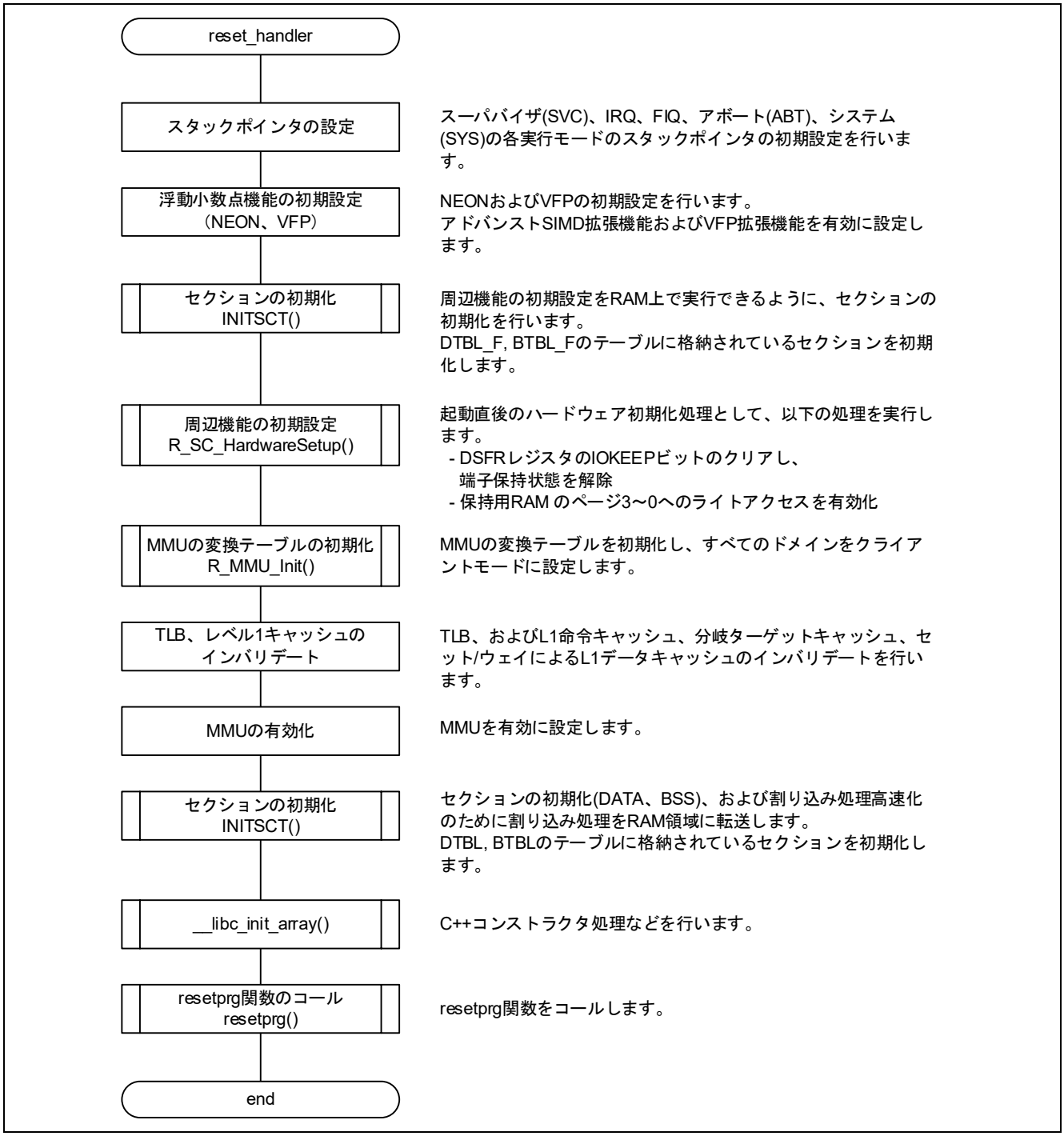
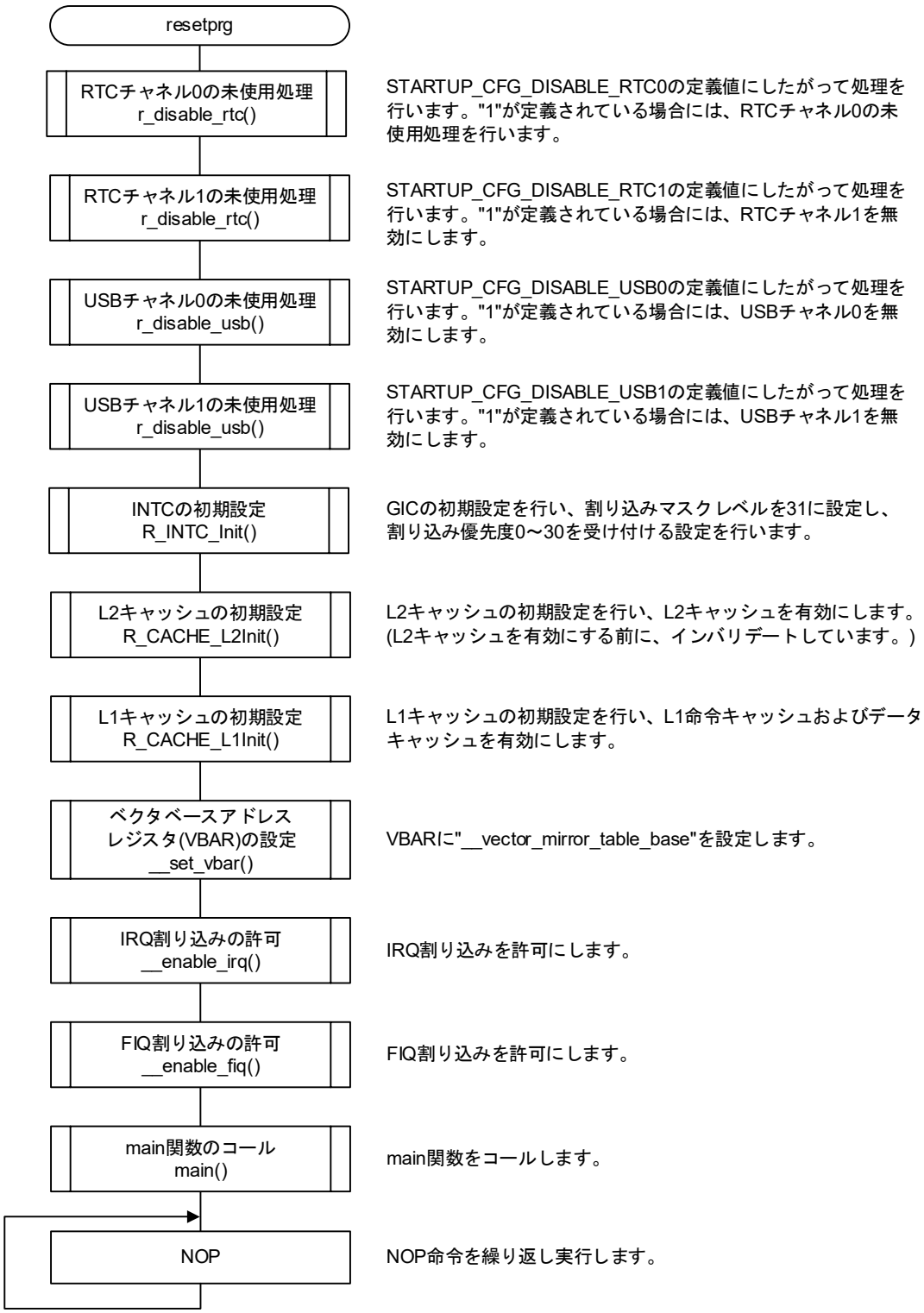


図5.13 リセットハンドラ処理

5.8.2 resetprg 関数

図5.14にresetprg関数のフローチャートを示します。



【注】 L1 命令キャッシュおよび L1 データキャッシュを有効にする前に、MMU、L1 命令キャッシュおよび L1 データキャッシュを無効にした状態で、L1 命令キャッシュ、L1 データキャッシュ、TLB をインバリデートする必要があります。また、同様に、L2 キャッシュも有効にする前に、L2 キャッシュをインバリデートする必要があります。

図5.14 resetprg 関数

5.8.3 メイン処理

図5.15にメイン処理のフローチャートを示します。

図5.15 メイン処理（1/2）

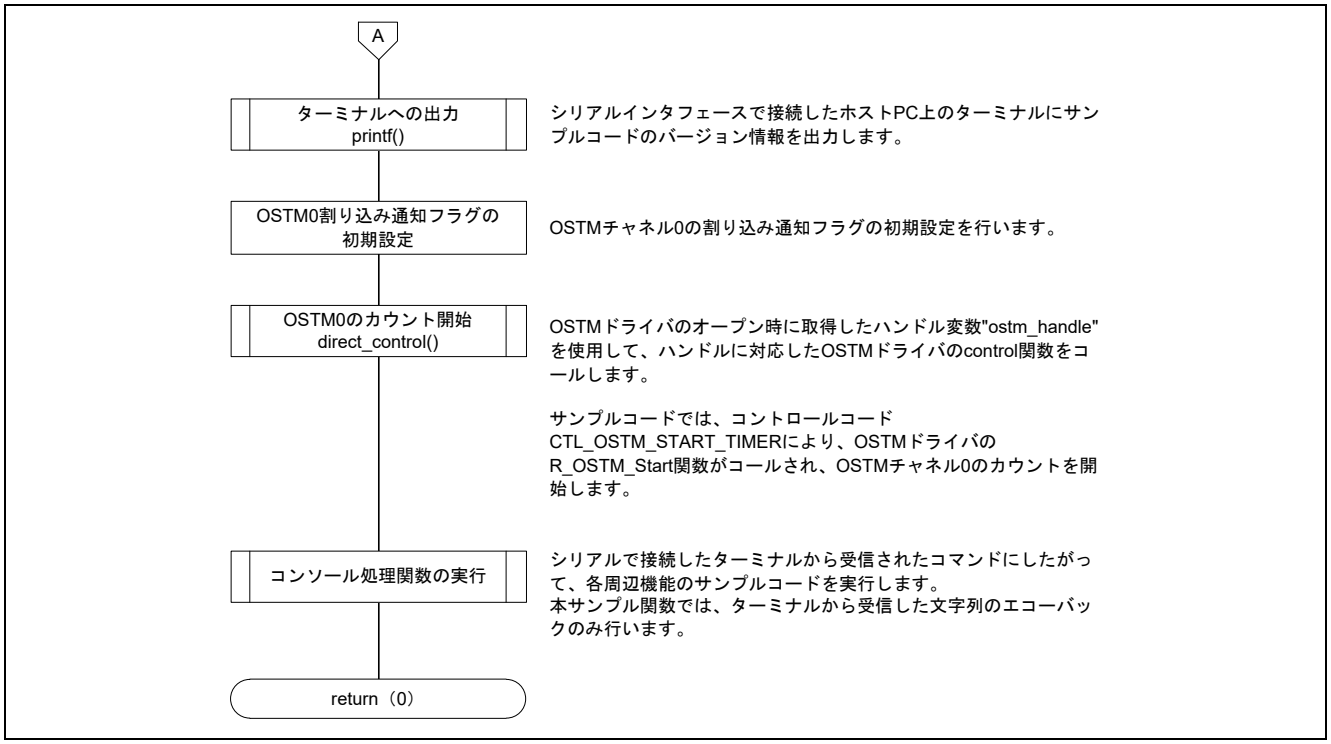


図5.16　メイン処理（2/2）

5.8.4 L2 キャッシュの初期設定関数

図5.17にL2キャッシュの初期設定関数のフローチャートを示します。

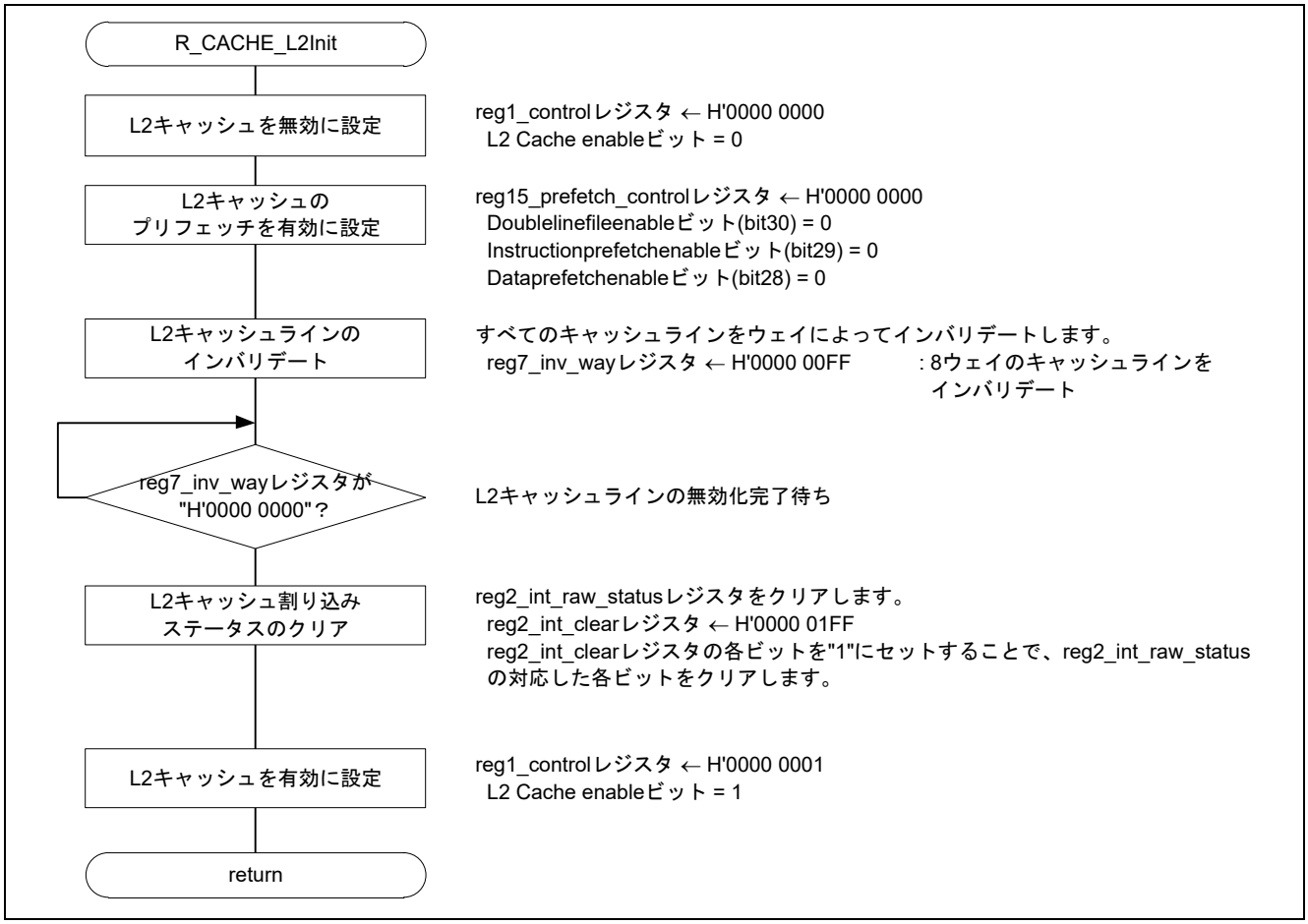


図5.17 L2 キャッシュの初期設定関数

5.8.5 MMU の初期設定関数

図5.18にMMUの初期設定関数のフローチャートを示します。

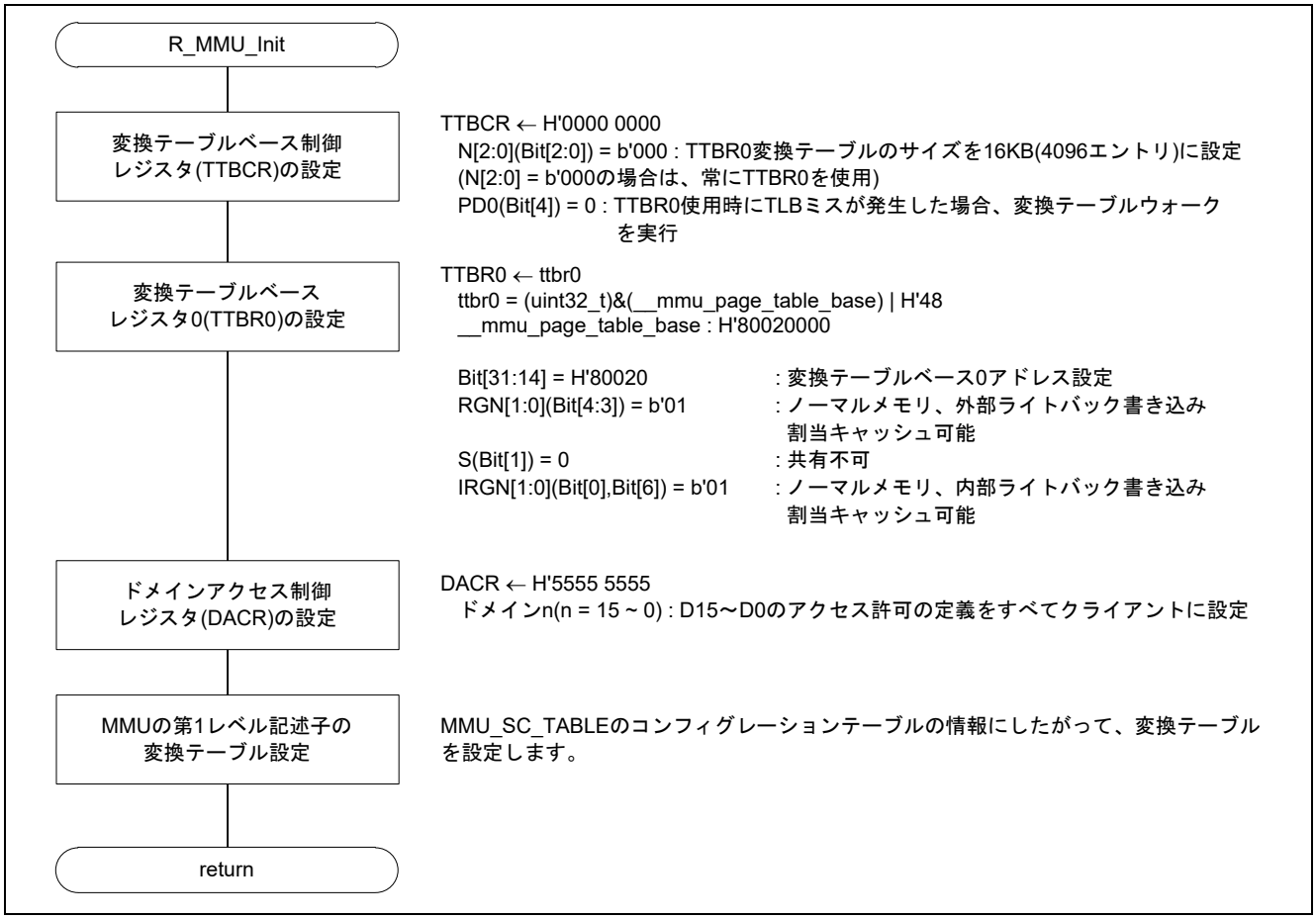


図5.18 MMU の初期設定関数

5.8.6 MMU の変換テーブルの設定関数

図5.19にMMUの変換テーブルの設定関数のフローチャートを示します。

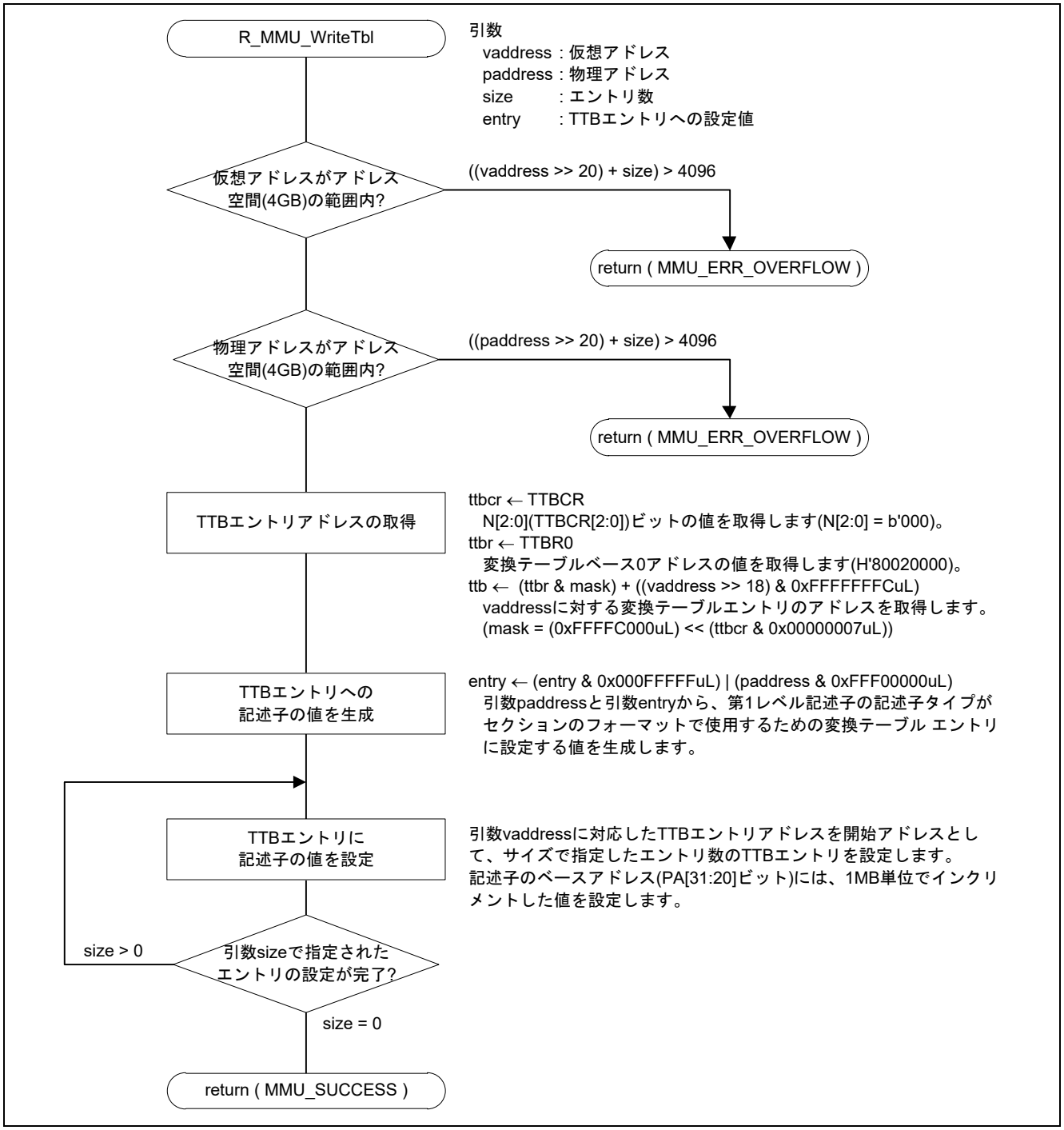


図5.19　MMU の変換テーブルの設定関数

5.8.7 INTC の初期設定関数

図5.20にINTCの初期設定関数のフローチャートを示します。

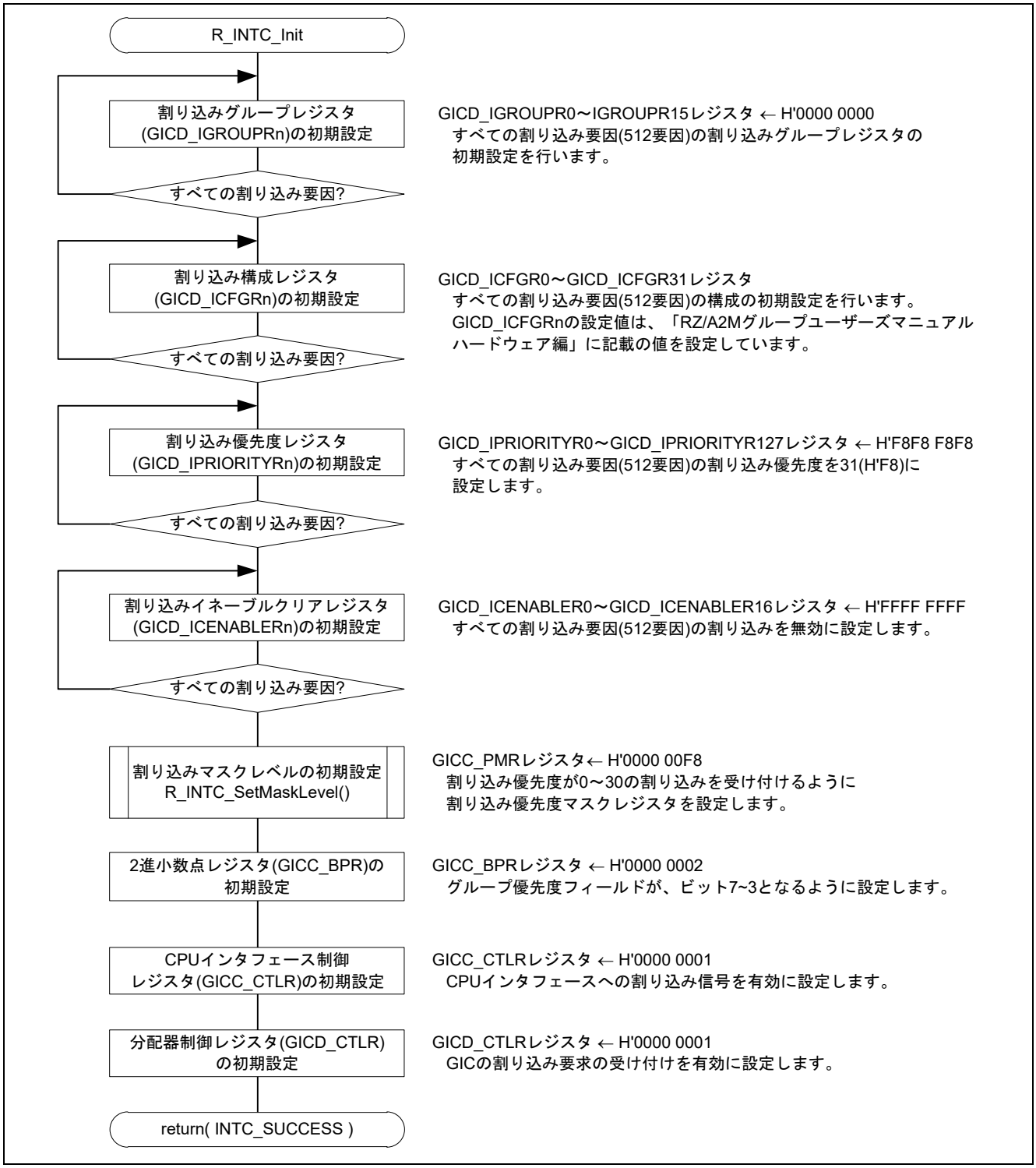


図5.20 INTC の初期設定関数

5.8.8 INTC 割り込みの許可関数

図5.21にINTC割り込みの許可関数のフローチャートを示します。

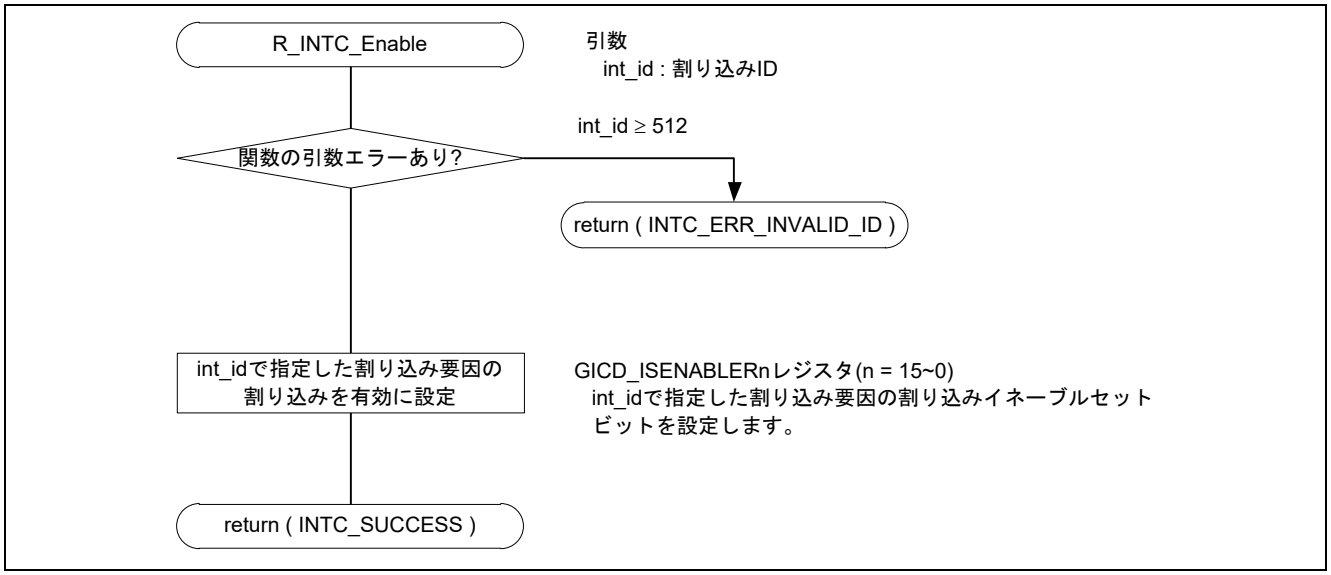


図5.21 INTC 割り込みの許可関数

5.8.9 INTC 割り込みの禁止関数

図5.22にINTC割り込みの禁止関数のフローチャートを示します。

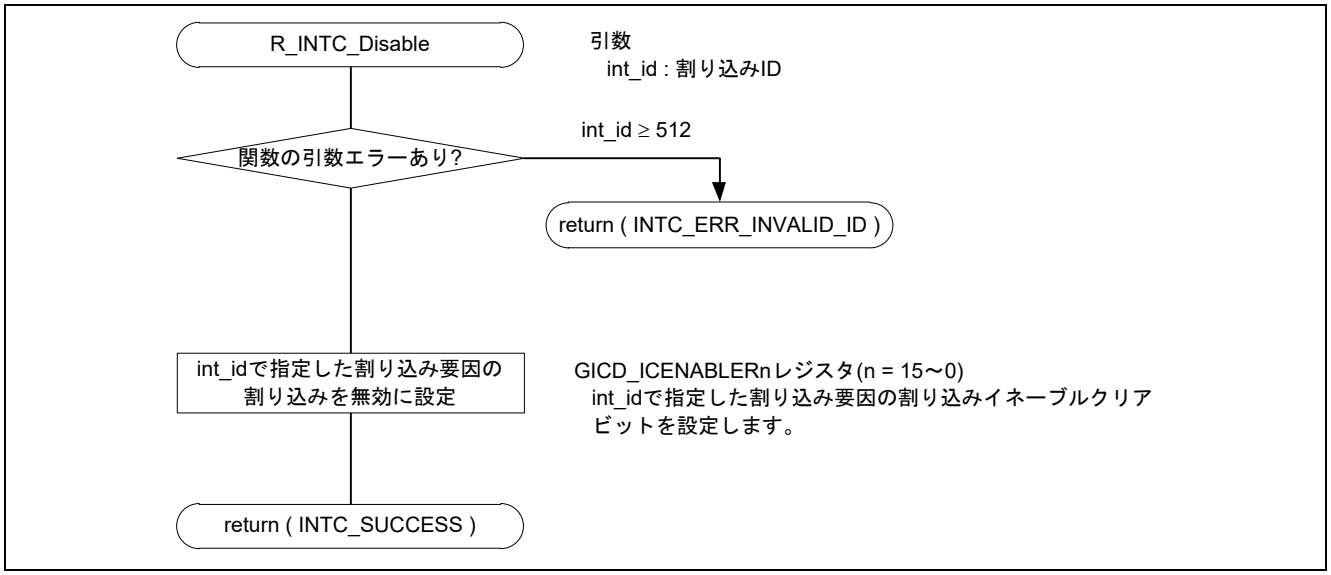


図5.22 INTC 割り込みの禁止関数

5.8.10 INTC 割り込み優先レベルの設定関数

図5.23にINTC割り込み優先レベルの設定関数のフローチャートを示します。

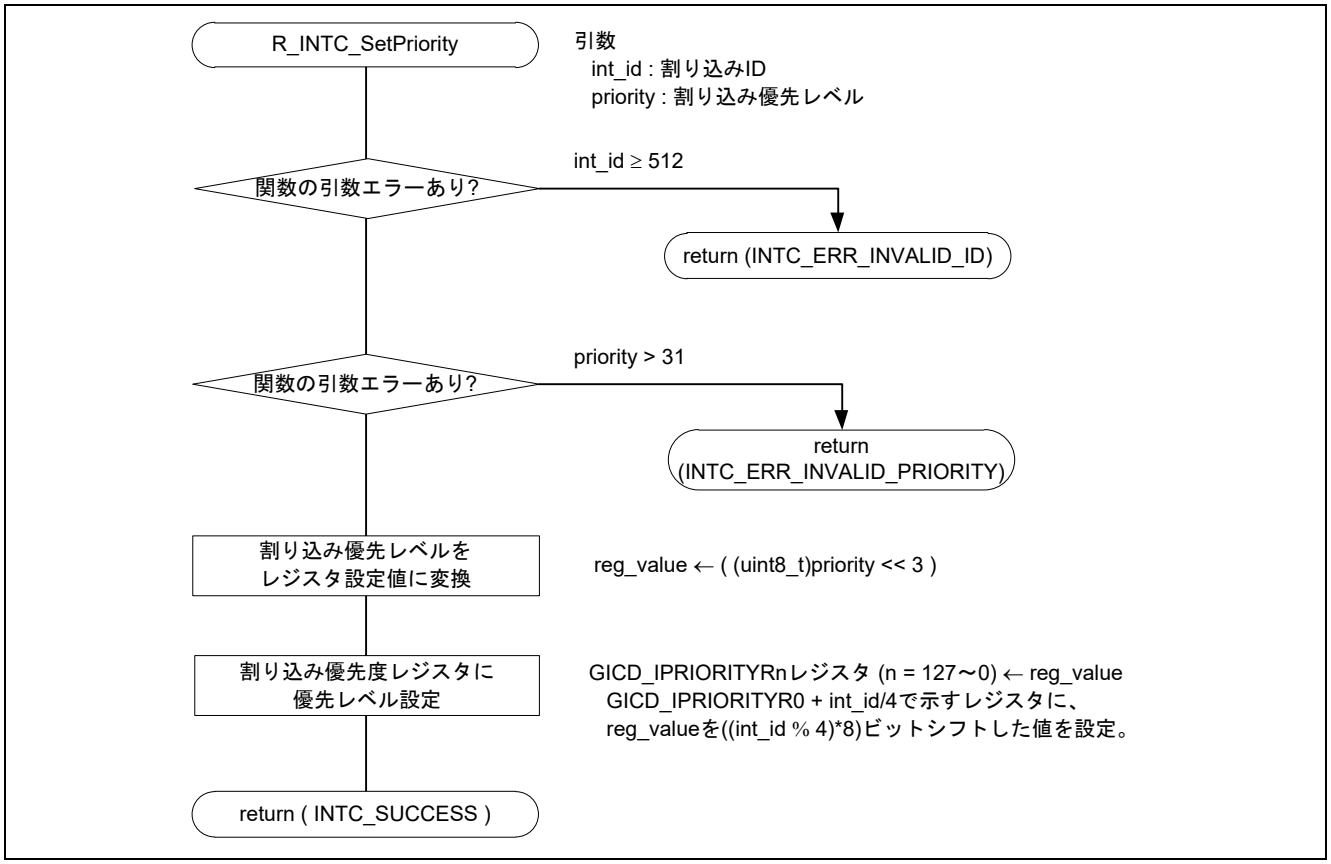


図5.23 INTC 割り込み優先レベルの設定関数

5.8.11 INTC 割り込みマスクレベルの設定関数

図5.24にINTC割り込みマスクレベルの設定関数のフローチャートを示します。

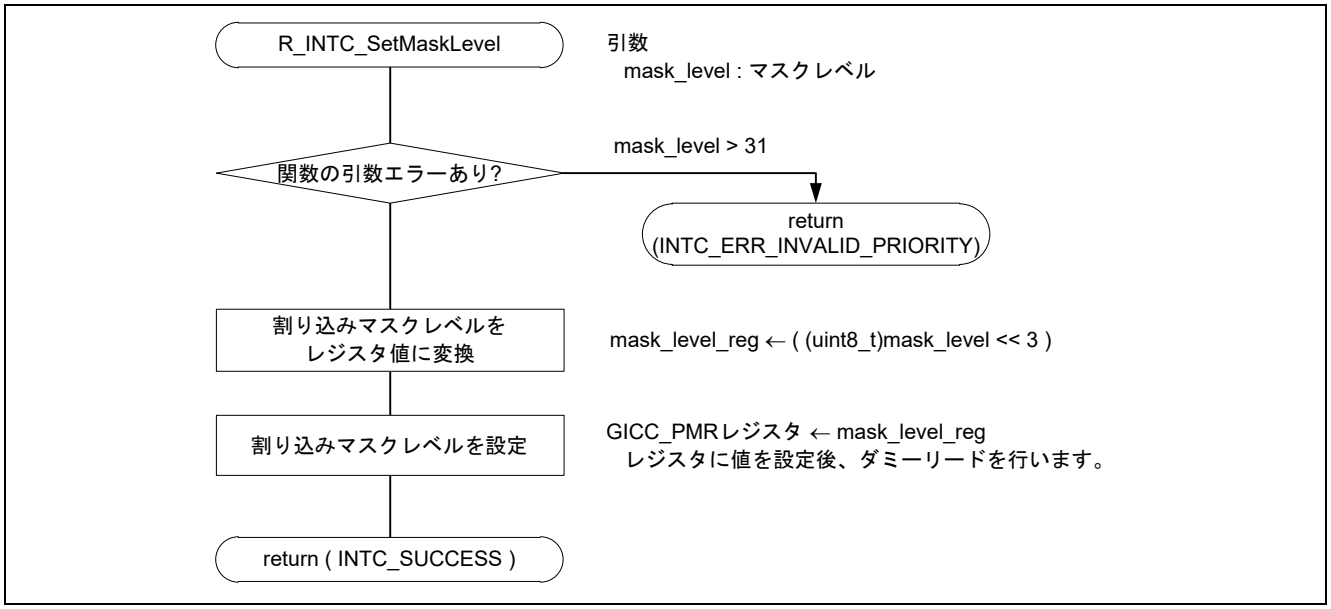


図5.24　INTC 割り込みマスクレベルの設定関数

5.8.12 INTC 割り込みマスクレベルの取得関数

図5.25にINTC割り込みマスクレベルの取得関数のフローチャートを示します。

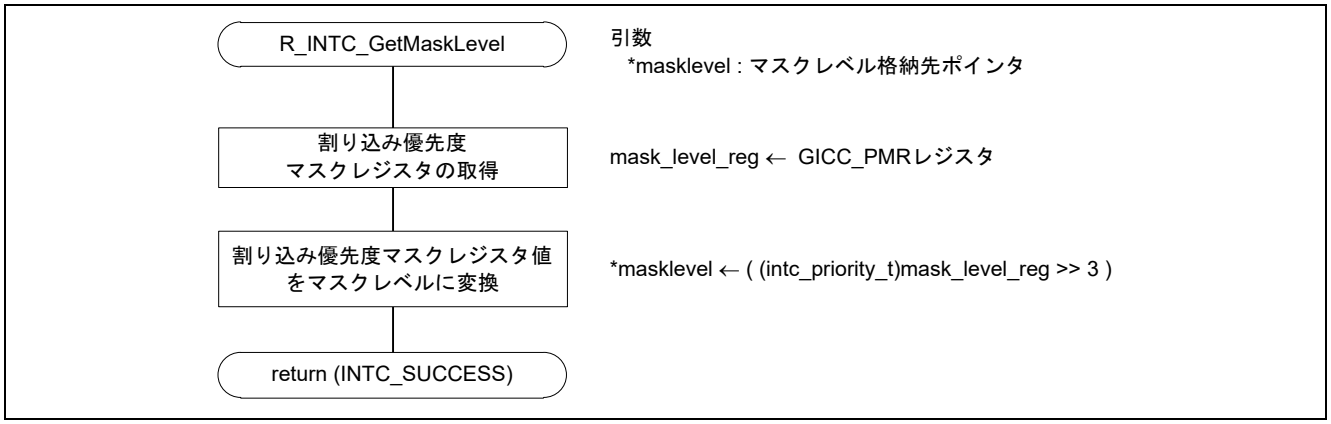


図5.25　INTC 割り込みマスクレベルの取得関数

5.8.13 INTC 割り込みハンドラの登録関数

図5.26にINTC割り込みハンドラの登録関数のフローチャートを示します。

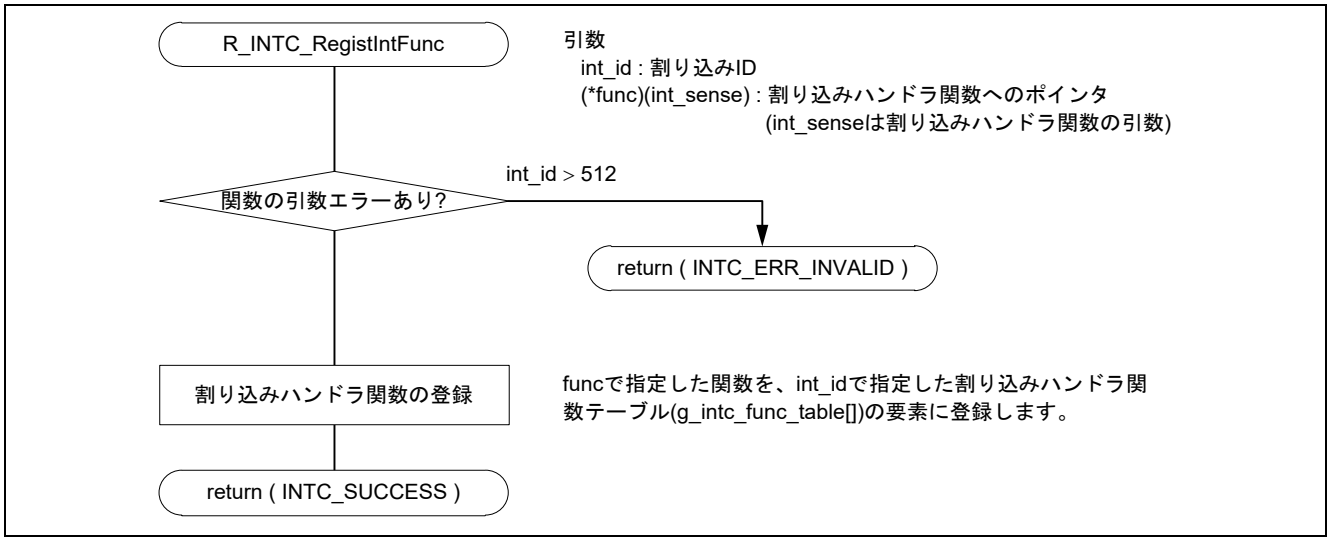


図5.26 INTC 割り込みハンドラの登録関数

5.8.14 IRQ ハンドラ処理

図5.27にIRQハンドラ処理のフローチャートを示します。

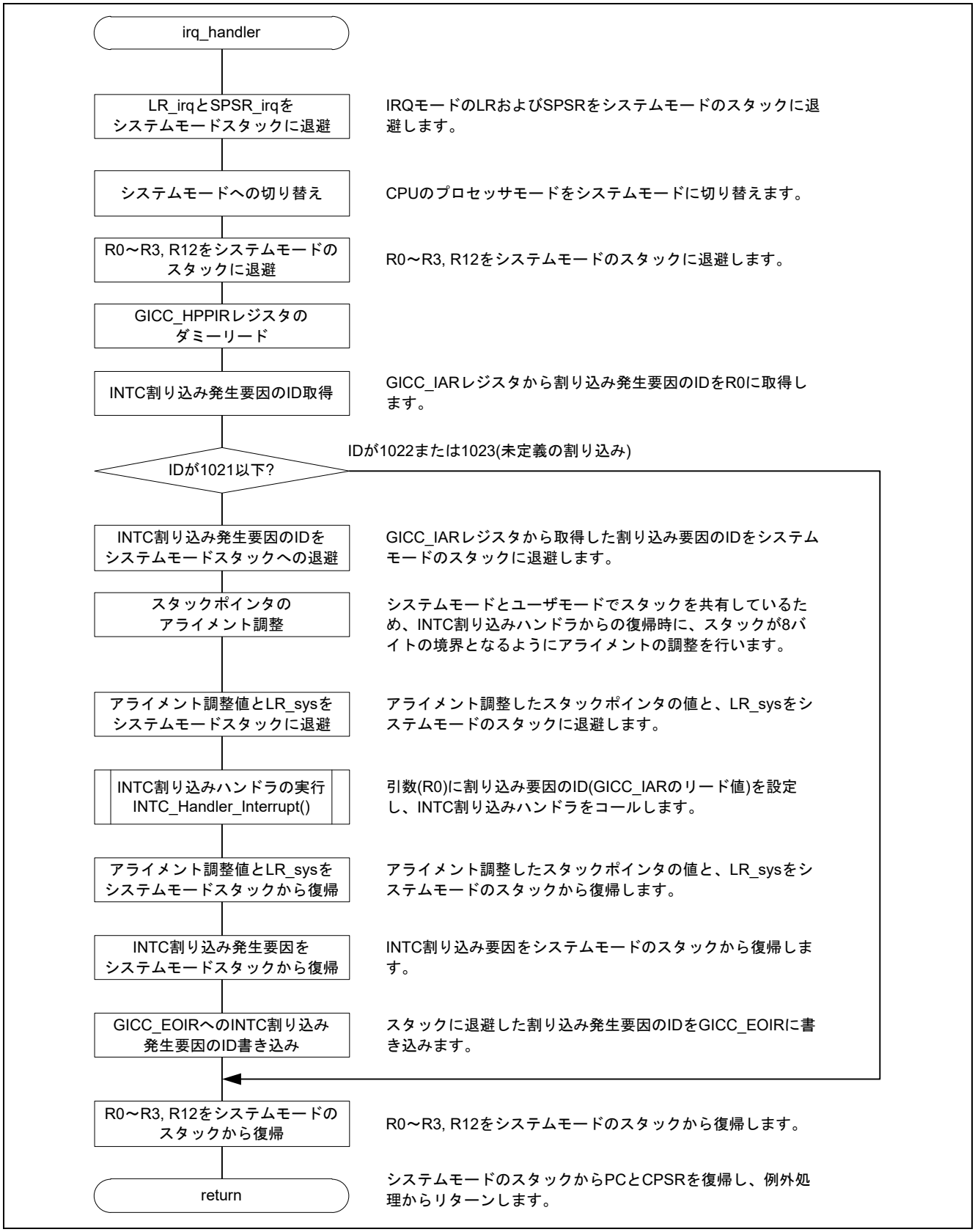


図5.27 IRQ ハンドラ処理

5.8.15 INTC 割り込みハンドラ処理

図5.28にINTC割り込みハンドラ処理のフローチャートを示します。

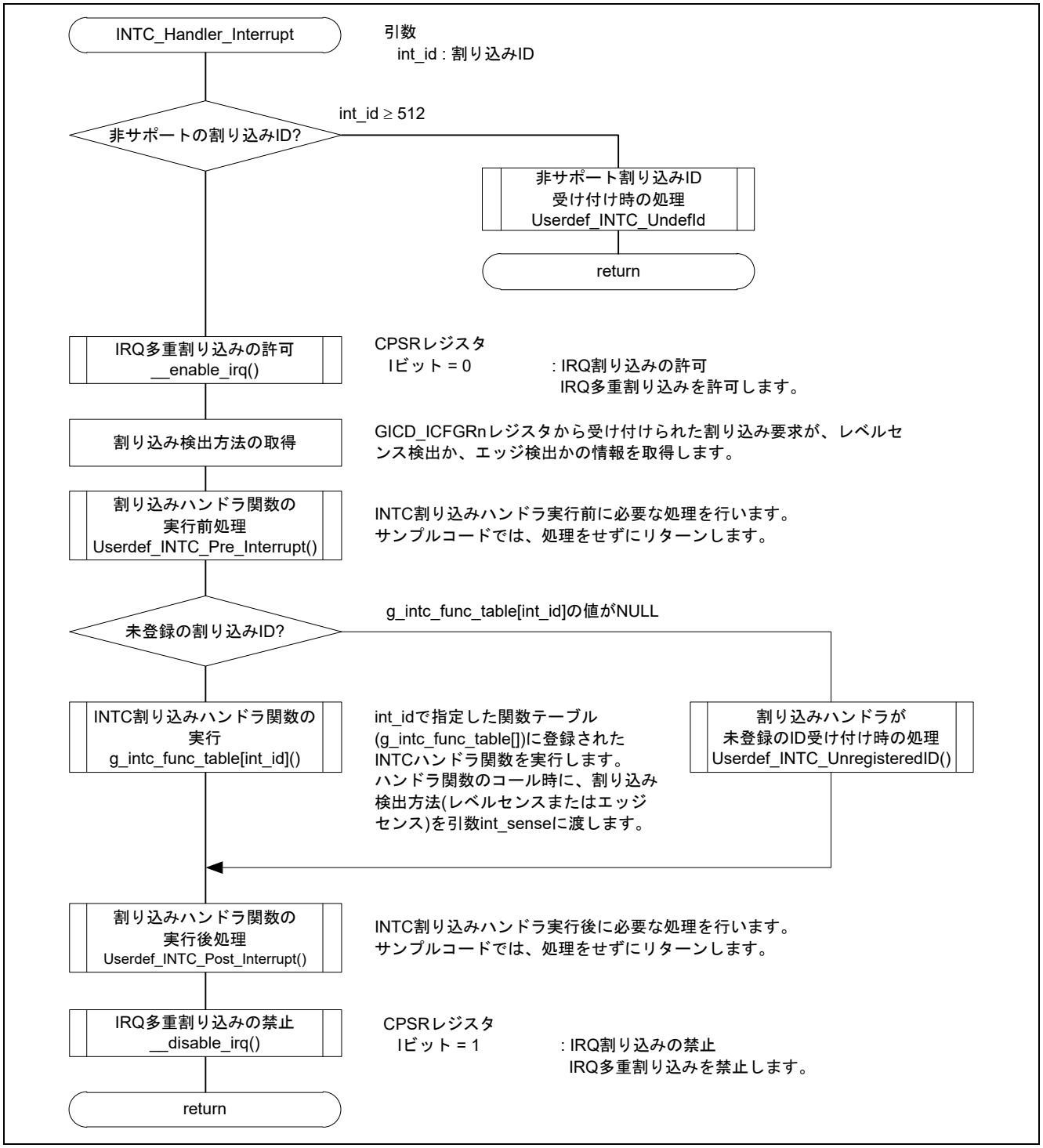


図5.28 INTC 割り込みハンドラ処理

6. サンプルコード

サンプルコードは、ルネサス エレクトロニクスホームページから入手してください。

7. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

RZ/A2Mグループ ユーザーズマニュアル ハードウェア編
（最新版をルネサス エレクトロニクスホームページから入手してください。）

RTK7921053C00000BE（RZ/A2M CPUボード）ユーザーズマニュアル
（最新版をルネサス エレクトロニクスホームページから入手してください。）

RTK79210XXB00000BE（RZ/A2M SUBボード）ユーザーズマニュアル
（最新版をルネサス エレクトロニクスホームページから入手してください。）

Arm Architecture Reference Manual ARMv7-A and ARMv7-R edition Issue C
（最新版を Arm ホームページから入手してください。）

Arm Cortex™-A9 Technical Reference Manual Revision: r4p1
（最新版を Arm ホームページから入手してください。）

Arm Generic Interrupt Controller Architecture Specification - Architecture version2.0
（最新版を Arm ホームページから入手してください。）

Arm CoreLink™ Level 2 Cache Controller L2C-310 Technical Reference Manual Revision: r3p3
（最新版を Arm ホームページから入手してください。）

テクニカルアップデート／テクニカルニュース
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：統合開発

統合開発環境 e2 studio のユーザーズマニュアルは、ルネサス エレクトロニクスホームページから入手してください。
（最新版をルネサス エレクトロニクスホームページから入手してください。）

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
Rev.1.00	Oct.04.18	ー	初版発行
Rev.1.10	Dec.28.18	P4	表1.1　使用する周辺機能と用途 OSTM2 を追加
		P12	表5.1　周辺機能の設定内容 OSTM2 を追加
		P18	表5.6　MMUの設定(2/2) 「CS3 空間キャッシュ無効領域」、「HyperRAM キャッシュ無効領域」、「OctaRAM キャッシュ無効領域」の設定を変更
		P22	表5.7　サンプルコードで使用するセクション名とオブジェクト名一覧 • IO レジスタアクセス用のセクションを移動 • UNCACHED_DATA セクションを追加
		P23	図5.6　セクション配置 • IO レジスタアクセス用のセクションを移動 • UNCACHED_DATA セクションを追加
		P31	表5.9　サンプルコードで使用する割り込み OSTM2 の割り込みを追加
		P32	表5.12　GPIOドライバで使用する定数（端子機能）を追加
		P32	表5.13　GPIOドライバで使用する定数（端子割り込み）を追加
		P32	表5.14　GPIOドライバで使用する定数（出力端子駆動能力）を追加
		P34	表5.16　API関数（1/2） R_MMU_ReadTbl 関数、R_MMU_VAtoPA 関数を追加
		P43～44	5.7　関数仕様 関数仕様に R_MMU_ReadTbl 関数、R_MMU_VAtoPA 関数を新規に追加
		P53～54	5.7　関数仕様 R_GPIO_HWInitialise 関数、R_GPIO_InitByPinList 関数、R_GPIO_InitByTable 関数に注意事項を追加
		P63	図5.13　リセットハンドラ処理 保持用 RAM（ページ 3～0）へのライトアクセス許可処理を追加
Rev.1.20	Apr.15.19	ー	CKIO が出力されるようにサンプルコードの内容を修正
		P25	5.2.6　L1キャッシュとL2キャッシュの設定 キャッシュを有効にする場合の注釈を追加
		P63	図5.13　リセットハンドラ処理 • NEON および VFP の初期設定の順序を変更 • DSFR レジスタの IOKEEP ビットのクリア処理を追加
		P64	図5.14　resetprg関数 キャッシュを有効にする場合の注釈を追加
		P65	図5.15　メイン処理（1/2） CPG ドライバのオープン時の説明を変更

Rev.	発行日	改訂内容	
		ページ	ポイント
Rev.1.30	May.17.19	P6	表2.1　動作確認条件（1/2） コンパイラオプション"-mthumb-interwork"を削除
Rev.1.40	Nov.20.19	P22 P23	R_SC_HardwareSetup 関数の処理を行う入力セクションを追加したことに伴い、以下の表および図を変更 • 表5.7　サンプルコードで使用するセクション名とオブジェクト名一覧 • 図5.6　セクション配置
		P35 P62	Userdef_PreHardwareSetup 関数および Userdef_PostHardwareSetup 関数を追加したことに伴い、関数一覧および関数仕様に追加 • 表5.18　ユーザ定義関数 • 関数仕様
		P63	図5.13　リセットハンドラ処理 「浮動小数点機能の初期設定」と「セクション初期化」の処理の順序の記載に誤記があり修正
		P65	図5.15　メイン処理（1/2） 関数名の誤記を修正（initialise_monitor_handles 関数から R_OS_AbstractionLayerInit 関数に修正）

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. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   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. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「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。
5. クロックについて
   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。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   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、将来の拡張機能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。
8. 製品間の相違について
   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、フラッシュメモリ、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、ノイズ幅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生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、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またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 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、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング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）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、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（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合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、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他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。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. 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。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
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。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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直接的、間接的に支配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。

(Rev.4.0-1　2017.11)

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/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。