RZ/A1Hグループ
リアルタイムクロック使用例（暫定版）

R01AN2189JJ0081
Rev.0.81
2014.09.05

要旨

本アプリケーションノートでは、RZ/A1Hのリアルタイムクロック（以下、RTC とします）の使用例について説明します。

対象デバイス

RZ/A1H

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

RTC の時刻（年、月、日、曜日、時、分、秒）を設定し、RTC による時刻カウント動作を開始します。時刻カウント動作を開始後、RTC から時刻を読み出し、その時刻をホスト PC 上のターミナルに表示します。また、低消費電力モード（ディープスタンバイモード）に遷移し、RTC のアラーム割り込みによってディープスタンバイモードから復帰します。これらの RTC を使用した基本的な動作をターミナルからコマンドを入力することにより実行することができます。なお、RTC はディープスタンバイ中も時刻カウント動作を継続するため、ディープスタンバイモードから復帰後に時刻を再設定する必要がありません。

本アプリケーションノートでは、FIFO 内蔵シリアルコミュニケーションインタフェースを SCIF、低消費電力モードを STB とします。

表 1.1に使用する周辺機能と用途を、図 1.1に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
RTC	時計・カレンダ機能を用いて時刻の設定および時刻の表示に使用、ディープスタンバイモードの解除要因としてアラーム割り込みを使用
SCIF	SCIF チャネル 2 とホスト PC との通信用
STB	RTC へのクロック供給に使用、ディープスタンバイモードへの遷移および解除の制御に使用

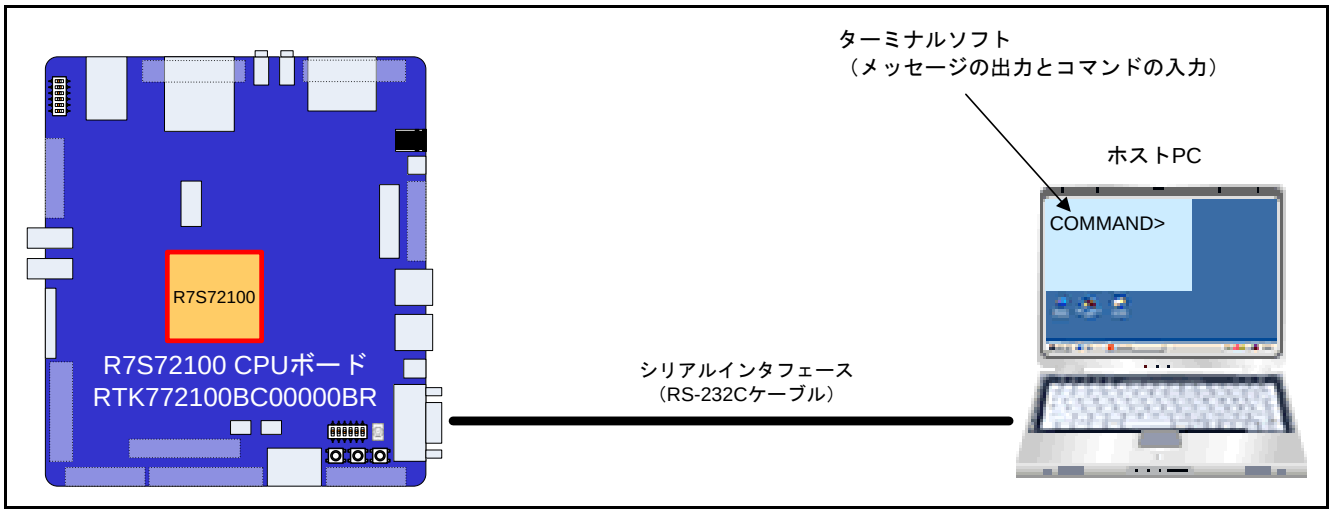


図1.1　動作環境

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

項目	内容
使用マイコン	RZ/A1H
動作周波数（注）	CPU クロック（Iφ）：400MHz 画像処理クロック（Gφ）：266.67MHz 内部バスクロック（Bφ）：133.33MHz 周辺クロック 1（P1φ）：66.67MHz 周辺クロック 0（P0φ）：33.33MHz
動作電圧	電源電圧（I/O）：3.3V 電源電圧（内部）：1.18V
統合開発環境	ARM®統合開発環境 ARM Development Studio 5（DS-5™）Version 5.16
C コンパイラ	ARM C/C++ Compiler/Linker/Assembler Ver.5.03 [Build 102]
	コンパイラオプション（ディレクトリパスの追加は除く） -O3 -Ospace --cpu=Cortex-A9 --littleend --arm --apcs=/interwork --no_unaligned_access --fpu=vfpv3_fp16 -g --asm
動作モード	ブートモード 0 （CS0 空間 16 ビットブート）
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	GENMAI ボード ・R7S72100 CPU ボード　RTK772100BC00000BR
使用デバイス	・NOR フラッシュメモリ（CS0、CS1 空間に接続） メーカ名：Spansion Inc.、型名：S29GL512S10TFI01 ・シリアルインタフェース（Dsub-9 コネクタ） ・LED1

【注】 クロックモード 0（EXTAL 端子からの 13.33MHz のクロック入力）で使用時の動作周波数です。

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/A1Hグループ レジスタ定義ヘッダ・ファイル iodefine.h（R01AN1860JJ）
- RZ/A1Hグループ　初期設定例（R01AN1864JJ）

4. 周辺機能説明

RTC について補足します。基本的な内容は「RZ/A1Hグループ ユーザーズマニュアル ハードウェア編」に記載しています。

RTC は、秒、分、時、曜日、日、月、年を BCD コード化して時刻カウントするレジスタを搭載しています（以降、秒、分、時、曜日、日、月、年カウンタを総称して、時刻カウンタとします）。時刻カウンタを設定し、時刻カウント動作を開始すると、以降は時刻カウンタを、現在の時刻（秒、分、時、曜日、日、月、年）の情報として使用することができます。なお、RTC は、低消費電力モード（ディープスタンバイモード）中でも内蔵水晶発振回路が停止しないため、ディープスタンバイモードへ遷移後も時刻カウント動作は継続して行われます。

また、RTC はアラーム機能をサポートしており、秒、分、時、曜日、日、月、年のいずれか、あるいは組み合わせでアラーム割り込みを発生させることができます。アラーム機能によるアラーム割り込みは、ディープスタンバイモードの解除要因として使用することができ、アラーム時刻を設定後、ディープスタンバイモードに遷移すると、設定したアラーム時刻にディープスタンバイモードから復帰することができます。なお、アラーム割り込みをディープスタンバイモードの解除要因として使用する場合、割り込みコントローラの実行優先度レジスタの設定値に関係なく、ディープスタンバイ解除要因として動作しますので、アラーム割り込みイネーブルフラグにより、アラーム割り込みを許可する必要はありません。

5. ハードウェア説明

5.1 使用端子一覧

表 5.1に使用端子と機能を示します。

表5.1 使用端子と機能

端子名	入出力	内容
A25~A1	出力	NOR フラッシュメモリへのアドレス信号出力
D15~D0	入出力	NOR フラッシュメモリのデータ信号入出力
CS0#	出力	CS0 空間に接続された NOR フラッシュメモリへのデバイス選択信号出力
RD#	出力	NOR フラッシュメモリへのリード制御信号出力
WE0#	出力	NOR フラッシュメモリへのライトイネーブル制御信号出力
MD_BOOT1	入力	ブートモードの選択
MD_BOOT0	入力	MD_BOOT1："L"、MD_BOOT0："L"（ブートモード 0 に設定）
P4_10	出力	LED の点灯および消灯
RxD2	入力	シリアル受信データ信号
TxD2	出力	シリアル送信データ信号
RTC_X1	入力	32.768kHz の RTC 用水晶発振子の接続に使用
RTC_X2	出力	

【注】 #は負論理（またはアクティブロー）を示す記号です。

6. ソフトウェア説明

6.1 動作概要

本サンプルコードは、RTC を使用して 6 種類のサンプル動作を行うことができます。サンプル動作は、ターミナルからコマンドを入力して実行します。表 6.1にサンプル動作の一覧を示します。

サンプルコードを動作させるための基本的なコマンド入力手順を以下に示します。

(1) 図 1.1に示す動作環境にて、GENMAI ボードの電源を ON します。
(2) ターミナルより、"1"+"Enter"を入力し、RTC の初期設定を行います。
(3) ターミナルより、"2"+"Enter"を入力し、ターミナルに表示されるメッセージに従って入力された時刻にて、RTC の時刻カウンタを設定します。
(4) ターミナルより、"4"+"Enter"を入力し、RTC の時刻カウント動作を開始します。
(5) ターミナルより、"3"+"Enter"を入力し、現在の時刻をターミナルに表示します。
(6) ターミナルより、"6"+"Enter"を入力し、ターミナルに表示されるメッセージに従って入力された時刻を、RTC のアラーム時刻に設定します。アラーム時刻を設定後、ディープスタンバイモードに遷移します。
(7) ディープスタンバイ中の時刻カウント動作により、時刻カウンタの値が(6)で設定したアラーム時刻になると、ディープスタンバイモードから復帰します。

表6.1　サンプル動作の一覧

サンプル動作	サンプル動作の内容	コマンド
RTC の初期設定	RTC の初期設定を行います。RTC の時刻カウント動作は停止します。	1
RTC の時刻設定	現在時刻を時刻カウンタに設定します。	2
RTC の時刻表示	時刻カウンタより、現在時刻を表示します。	3
RTC の時刻カウント動作開始	RTC の時刻カウント動作を開始します。	4
RTC の時刻カウント動作停止	RTC の時刻カウント動作を停止します。	5
RTC のアラーム時刻設定およびディープスタンバイモード遷移	RTC のアラーム時刻を設定し、ディープスタンバイモードに遷移します。設定したアラーム時刻になると、ディープスタンバイモードから復帰します。	6

【注】 コマンド 2～6 を実行する前に、コマンド 1 を実行してください。

6.1.1　RTC の初期設定（コマンド 1）

RTC にクロックを供給するために STB を設定し、RTC の初期設定を行います。時刻カウント動作を停止した後に、RTC の初期設定を行います。

本コマンド実行後、RTC の時刻カウント動作は停止状態となります。なお、本コマンドは他のコマンドを実行する前に実行してください。

図 6.1に RTC の初期設定時のターミナルへの表示を示します。

```
RTC SAMPLE> 1

RTC initialize complete.
```

図6.1　初期設定のターミナル表示

6.1.2　RTC の時刻設定（コマンド 2）

RTC の時刻カウンタ（秒、分、時、曜日、日、月、年）を設定します。時刻は 10 進数で入力します。曜日、月、日、年、時、分、秒の順で入力します。なお、"-1"を入力した場合は、RTC の時刻カウンタへの設定は行いません。GENMAI ボードの電源を ON にした後、本コマンドによる時刻設定を一度も行わずに、時刻カウント動作を開始した場合は、「RZ/A1H グループ ユーザーズマニュアル ハードウェア編」に記載されている初期値から、時刻カウント動作を開始します。

表6.2　時刻の入力対応表

時刻	入力可能な値（10 進数）
曜日	0～6 0：日曜日、1：月曜日、2：火曜日、3：水曜日、 4：木曜日、5：金曜日、6：土曜日、 -1：変更なし
月	1～12 -1：変更なし
日	1～31 -1：変更なし
年	0～9999 -1：変更なし
時	0～23 -1：変更なし
分	0～59 -1：変更なし
秒	0～59 -1：変更なし

図 6.2に「2014 年 4 月 1 日火曜日 10 時 15 分」の時刻を設定する例を示します（「秒」の設定をしない場合）。

```
RTC SAMPLE> 2

Enter time (decimal).
Enter "-1" to the item where you do not want to change the time.

Day of week (Sun=0/Mon=1/Tue=2/Wed=3/Thu=4/Fri=5/Sat=6) : 2
Month (1 - 12) : 4
Day (1 - 31) : 1
Year (0 - 9999) : 2014
Hour (0 - 23) : 10
Minute (0 - 59) : 15
Second (0 - 59) : -1

RTC counter setting complete.
```

図6.2　時刻の設定例

6.1.3　RTC の時刻表示（コマンド 3）

RTC の時刻カウンタから BCD コードで時刻情報を取得し、ターミナルに表示します。時刻は、「曜日 月 日, 年 at 時:分:秒」の形式で表示します。

図 6.3に「2014 年 4 月 1 日火曜日 10 時 15 分 45 秒」を表示する例を示します。

```
RTC SAMPLE> 3

------------------------------
Tue. Apr.  1, 2014 at 10:15:45
------------------------------
```

図6.3　時刻の表示例

6.1.4　RTC の時刻カウント開始（コマンド 4）

RTC の時刻カウント動作を開始します。また、時刻カウント動作開始時の時刻を表示します。

図 6.4に「2014 年 4 月 1 日火曜日 10 時 15 分 10 秒」に時刻カウント動作を開始する例を示します。

```
RTC SAMPLE> 4

------------------------------
Tue. Apr.  1, 2014 at 10:15:10
------------------------------

RTC count start.
```

図6.4　時刻カウント開始例

6.1.5　RTC の時刻カウント停止（コマンド 5）

RTC の時刻カウント動作を停止します。また、時刻カウント動作停止時の時刻を表示します。

図 6.5に「2014 年 4 月 1 日火曜日 10 時 15 分 45 秒」に時刻カウント動作を停止する例を示します。

```
RTC SAMPLE> 5

RTC count stop.

------------------------------
Tue. Apr.  1, 2014 at 10:15:45
------------------------------
```

図6.5　時刻カウント停止例

6.1.6　RTC のアラーム時刻設定およびディープスタンバイモード遷移（コマンド 6）

本コマンドを実行すると、下記の処理を実行します。

(1) RTC のアラーム時刻に設定する時刻をターミナルから入力します。
(2) (1)で入力した時刻を、アラーム時刻に設定します。
(3) RTC の時刻カウンタから現在時刻を取得し、ディープスタンバイモード遷移時刻としてターミナルに表示します。
(4) ディープスタンバイモードからの解除要因をアラーム割り込みに設定し、ディープスタンバイモードに遷移します。
(5) ディープスタンバイ中の時刻カウント動作により、時刻カウンタの値が(2)で設定したアラーム時刻になると、ディープスタンバイモードから復帰します。
(6) ディープスタンバイモードの解除要因を確認し、解除要因が RTC のアラーム割り込みであった場合、時刻カウンタから時刻を取得し、ディープスタンバイモード解除時刻としてターミナルに表示します。

図 6.6にディープスタンバイモード解除時刻を「10 時 20 分」に設定し、「2014 年 4 月 1 日火曜日 10 時 15 分 45 秒」に、ディープスタンバイモードに遷移した場合の例を示します。

```
RTC SAMPLE> 6

Transit to deep standby mode.
Enter time to cancel deep standby mode (decimal).
Enter "-1" to the item where you set current time.

Hour (0 - 23) : 10
Minute (0 - 59) : 20

Transit to deep standby mode at..
------------------------------
Tue. Apr.  1, 2014 at 10:15:45
------------------------------

Deep standby mode will be canceled at 10:20:00.


RZ/A1H CPU Board Sample Program. Ver.X.XX
Copyright (C) 2014 Renesas Electronics Corporation. All rights reserved.

Deep standby mode canceled at..
------------------------------
Tue. Apr.  1, 2014 at 10:20:00
------------------------------

select sample program.

SAMPLE>
```

図6.6　アラーム設定およびディープスタンバイモード遷移例

ディープスタンバイモード解除時刻は、周辺機能の初期設定関数（$Sub$$main）からコールされるサンプル関数 STB_CancelDeepStandby で保存しています。また、保存した時刻を main 関数からコールされるサンプル関数 RTC_DispTimeCanDeepStb で、ターミナルに表示しています。

6.2　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

6.2.1　周辺機能の設定

表 6.3にサンプルコード実行時の設定内容を示します。

表6.3　周辺機能の設定内容

モジュール	設定内容
RTC	動作クロック：RTC_X1 からの 32.768kHz を選択 周期割り込みの発生：禁止 桁上げ割り込みの発生：禁止 アラーム割り込みの発生：禁止
SCIF	チャネル 2 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 P1φ=66.67MHz の時に、クロックソースを分周なし、ビットレート値に 17 を設定し、ビットレートが 115200bps となるように設定 誤差は 0.46%
STB	RTC へのクロック供給 ディープスタンバイモードへの遷移 RTC のアラーム割り込みで、ディープスタンバイモード解除 ディープスタンバイモード解除後、外部メモリから起動

6.2.2　サンプルコードのセクション配置

表 6.4および表 6.5にサンプルコードで使用するセクションを示し、図 6.7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、スキャッタローディング機能を使用後のセクション配置（実行ビュー）を示します。

セクションおよびスキャッタローディング機能の詳細については、ARM より提供される「ARM コンパイラツールチェーン リンカの使用」の「イメージの構造と生成」を参照してください。

表6.4　使用するセクション（1/2）

領域の名前	内容	タイプ	ロード領域	実行領域
VECTOR_TABLE	例外処理ベクタテーブル	Code	FLASH	FLASH
RESET_HANDLER	リセットハンドラ処理のプログラムコード領域 この領域は以下のセクションから構成されています ・INITCA9CACHE（L1 キャッシュ設定） ・INIT_TTB（MMU 設定） ・RESET_HANDLER（リセットハンドラ）	Code	FLASH	FLASH
CODE_BASIC_SETUP	動作周波数とフラッシュメモリ最適化のためのプログラムコード領域	Code	FLASH	FLASH
InRoot	この領域は C 標準ライブラリなどのルート領域に配置するセクションから構成されています	Code および RO Data	FLASH	FLASH
CODE_FPU_INIT	NEON および VFP 初期設定のプログラムコード領域 この領域は以下のセクションから構成されています ・CODE_FPU_INIT ・FPU_INIT	Code	FLASH	FLASH
CODE_RESET	ハードウェア初期設定のプログラムコード領域 この領域は以下のセクションから構成されています ・CODE_RESET（スタートアップ処理） ・INIT_VBAR（ベクタベース設定）	Code	FLASH	FLASH
CODE_IO_REGRW	IO レジスタのリード/ライト関数のプログラムコード領域	Code	FLASH	FLASH
CODE	デフォルトのプログラムコード領域 C ソースでセクション名を定義しない Code タイプのセクションは、すべてこの領域に配置されます	Code	FLASH	FLASH
CONST	デフォルトの定数データ領域 C ソースでセクション名を定義しない RO Data タイプのセクションは、すべてこの領域に配置されます	RO Data	FLASH	FLASH

表6.5　使用するセクション（2/2）

領域の名前	内容	タイプ	ロード領域	実行領域
VECTOR_MIRROR_TABLE	例外処理ベクタテーブル （大容量内蔵 RAM に転送して実行するためのセクション）	Code	FLASH	LRAM
CODE_HANDLER_JMPTBL	IRQ 割り込みハンドラのユーザ定義関数のプログラムコード領域	Code	FLASH	LRAM
CODE_HANDLER	IRQ 割り込みハンドラのプログラムコード領域 この領域は以下のセクションから構成されています ・CODE_HANDLER ・IRQ_FIQ_HANDLER	Code	FLASH	LRAM
DATA_HANDLER_JMPTBL	IRQ 割り込みハンドラのユーザ定義関数の登録テーブルデータ領域	RW Data	FLASH	LRAM
DATA_RESET	ハードウェア初期設定の初期値ありデータ領域	RW Data	FLASH	LRAM
BSS_RESET	ハードウェア初期設定の初期値なしデータ領域	ZI Data	―	LRAM
ARM_LIB_STACK	アプリケーションスタック領域	ZI Data	―	LRAM
IRQ_STACK	IRQ モードのスタック領域	ZI Data	―	LRAM
FIQ_STACK	FIQ モードのスタック領域	ZI Data	―	LRAM
SVC_STACK	スーパバイザ（SVC）モードのスタック領域	ZI Data	―	LRAM
ABT_STACK	アボート（ABT）モードのスタック領域	ZI Data	―	LRAM
TTB	MMU 変換テーブル領域	ZI Data	―	LRAM
ARM_LIB_HEAP	アプリケーションヒープ領域	ZI Data	―	LRAM
DATA	デフォルトの初期値ありデータ領域 C ソースでセクション名を定義しない RW Data タイプのセクションは、すべてこの領域に配置されます	RW Data	FLASH	LRAM
BSS	デフォルトの初期値なしデータ領域 C ソースでセクション名を定義しない ZI Data タイプのセクションは、すべてこの領域に配置されます	ZI Data	―	LRAM

【注】 1. 表中のロード領域および実行領域において、FLASH は NOR 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. セクションの名前は基本的に領域と同じ名前にしていますが、RESET_HANDLER、InRoot、CODE_FPU_INIT、CODE_RESET、CODE、CONST、CODE_HANDLER、DATA、BSS の各領域は複数のセクションから構成されています。領域とセクションについては、ARM コンパイラツールチェーンのマニュアルを参照してください。

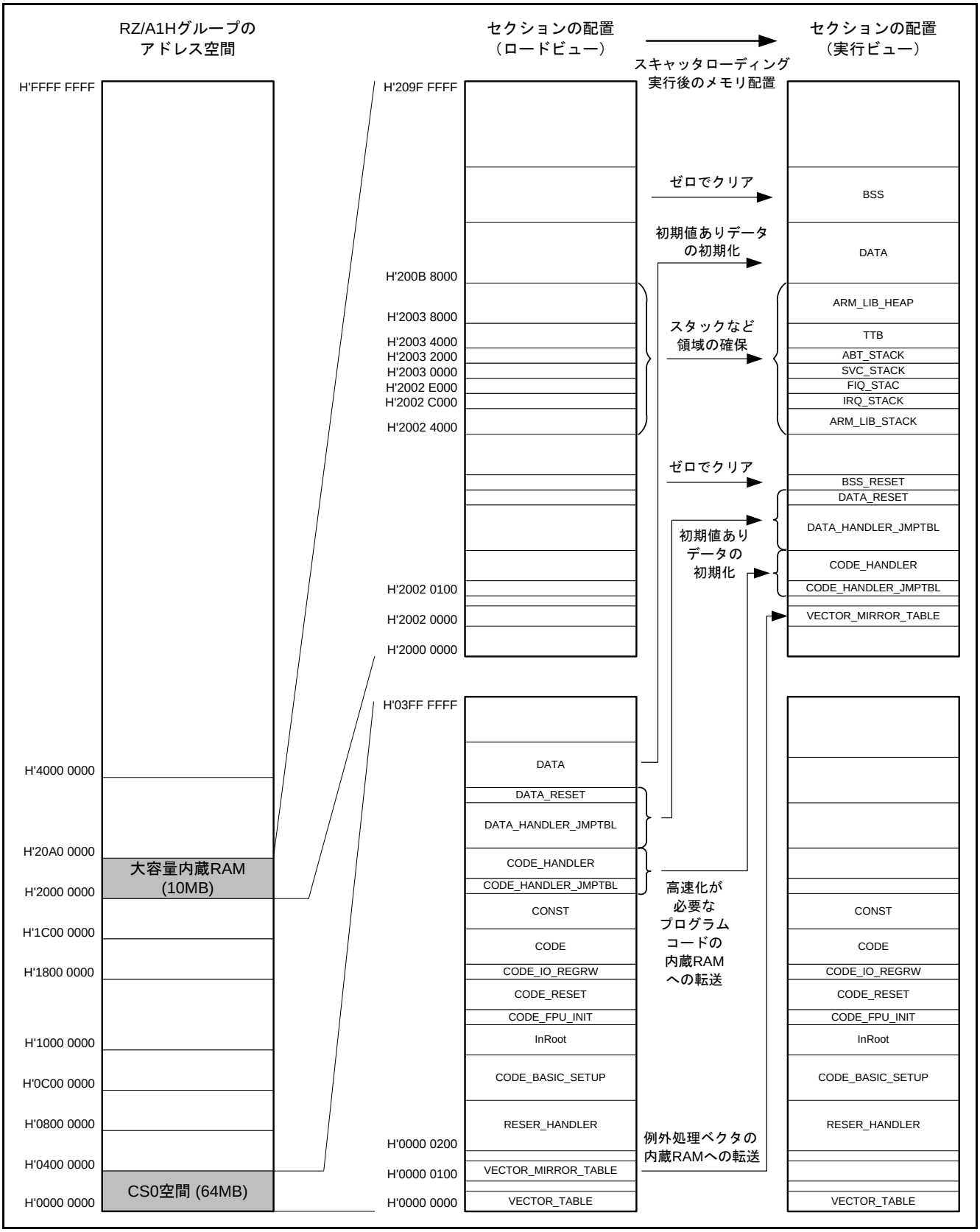

RZ/A1Hグループの
アドレス空間
セクションの配置
（ロードビュー）
スキャッタローディング
実行後のメモリ配置
セクションの配置
（実行ビュー）
H'FFFF FFFF
H'4000 0000
H'20A0 0000
大容量内蔵RAM
(10MB)
H'2000 0000
H'1C00 0000
H'1800 0000
H'1000 0000
H'0C00 0000
H'0800 0000
H'0400 0000
CS0空間 (64MB)
H'0000 0000
H'209F FFFF
H'200B 8000
H'2003 8000
H'2003 4000
H'2003 2000
H'2003 0000
H'2002 E000
H'2002 C000
H'2002 4000
H'2002 0100
H'2002 0000
H'2000 0000
H'03FF FFFF
DATA
DATA_RESET
DATA_HANDLER_JMPTBL
CODE_HANDLER
CODE_HANDLER_JMPTBL
CONST
CODE
CODE_IO_REGRW
CODE_RESET
CODE_FPU_INIT
InRoot
CODE_BASIC_SETUP
RESER_HANDLER
H'0000 0200
H'0000 0100
VECTOR_MIRROR_TABLE
H'0000 0000
VECTOR_TABLE
ゼロでクリア
初期値ありデータ
の初期化
スタックなど
領域の確保
ゼロでクリア
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
BSS
DATA
ARM_LIB_HEAP
TTB
ABT_STACK
SVC_STACK
FIQ_STAC
IRQ_STACK
ARM_LIB_STACK
BSS_RESET
DATA_RESET
DATA_HANDLER_JMPTBL
CODE_HANDLER
CODE_HANDLER_JMPTBL
VECTOR_MIRROR_TABLE
CONST
CODE
CODE_IO_REGRW
CODE_RESET
CODE_FPU_INIT
InRoot
CODE_BASIC_SETUP
RESER_HANDLER
VECTOR_TABLE

図6.7　セクション配置

6.3　固定幅整数一覧

表 6.6にサンプルコードで使用する固定幅整数を示します。

表6.6　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（標準ライブラリにて定義）
int16_t	16 ビット整数、符号あり（標準ライブラリにて定義）
int32_t	32 ビット整数、符号あり（標準ライブラリにて定義）
int64_t	64 ビット整数、符号あり（標準ライブラリにて定義）
uint8_t	8 ビット整数、符号なし（標準ライブラリにて定義）
uint16_t	16 ビット整数、符号なし（標準ライブラリにて定義）
uint32_t	32 ビット整数、符号なし（標準ライブラリにて定義）
uint64_t	64 ビット整数、符号なし（標準ライブラリにて定義）
float32_t	32 ビット浮動小数 （"__ARM_NEON__"を指定時、標準ライブラリにて定義）
float64_t	64 ビット浮動小数（標準ライブラリにて定義） （"__ARM_NEON__"を指定時、標準ライブラリにて定義）
float128_t	128 ビット浮動小数

6.4　定数一覧

表 6.7にサンプルコードで使用する定数を示します。

表6.7　サンプルコードで使用する定数

定数名	設定値	内容
RTC_ENABLE	(1)	時刻カウンタ（秒、分、時、曜日、日、月、年）のそれぞれの設定または読み出しを有効
RTC_DISABLE	(0)	時刻カウンタ（秒、分、時、曜日、日、月、年）のそれぞれの設定または読み出しを無効
 RTC_WK_SUNDAY RTC_WK_MONDAY RTC_WK_TUESDAY RTC_WK_WEDNESDAY RTC_WK_THURSDAY RTC_WK_FRIDAY RTC_WK_SATURDAY RTC_WK_TOTAL	 (0) (1) (2) (3) (4) (5) (6) (7)	曜日 日曜日 月曜日 火曜日 水曜日 木曜日 金曜日 土曜日 曜日の総数
STB_GENERATE_ALARM_INT	(1)	ディープスタンバイモードの解除が RTC アラーム割り込みにより行われた状態の定義
STB_NO_GENERATE_ALARM_INT	(0)	ディープスタンバイモードの解除が RTC アラーム割り込みにより行われていない状態の定義 （パワーオンリセットからの起動もこの定義を使用します）

6.5　構造体一覧

図 6.8にサンプルコードで使用する構造体を示します。

```
/* ==== Type declaration for time setting ==== */
/* ---- Type declaration for registers with 8-bit width ---- */
typedef struct rtc_8
{
   uint8_t value;    /* Setting values for time counter or alarm register   */
   uint8_t enable;   /* Specification for setting or acquisition            */
                     /* (RTC_DISABLE : Disable setting or acquisition,      */
                     /*   RTC_ENABLE  : Enable setting or acquisition)      */
} rtc_8_t;

/* ---- Type declaration for registers with 16-bit width
        (For year counter and year alarm registers) ---- */
typedef struct rtc_16
{
   uint16_t value;   /* Setting values for year counter or year alarm register */
   uint8_t  enable;  /* Specification for setting or acquisition             */
                     /* (RTC_DISABLE : Disable setting or acquisition,       */
                     /*   RTC_ENABLE  : Enable setting or acquisition)       */
} rtc_16_t;

/* ==== Structure declaration for time setting ==== */
typedef struct rtc_time
{                    /*             value       enable                      */
   rtc_8_t  second;  /* Second      (0 to 59)   (RTC_DISABLE or RTC_ENABLE) */
   rtc_8_t  minute;  /* Minute      (0 to 59)   (RTC_DISABLE or RTC_ENABLE) */
   rtc_8_t  hour;    /* Hour        (0 to 23)   (RTC_DISABLE or RTC_ENABLE) */
   rtc_8_t  week;    /* Day of week (0 to 6)    (RTC_DISABLE or RTC_ENABLE) */
   rtc_8_t  day;     /* Day         (1 to 31)   (RTC_DISABLE or RTC_ENABLE) */
   rtc_8_t  month;   /* Month       (1 to 12)   (RTC_DISABLE or RTC_ENABLE) */
   rtc_16_t year;    /* Year        (0 to 9999) (RTC_DISABLE or RTC_ENABLE) */
} rtc_time_t;

/* ==== Structure declaration for alarm-specified information (ENB bit) ==== */
typedef struct rtc_alarm_enb
{
   uint8_t second;   /* Second      (0 or 1) */
   uint8_t minute;   /* Minute      (0 or 1) */
   uint8_t hour;     /* Hour        (0 or 1) */
   uint8_t week;     /* Day of week (0 or 1) */
   uint8_t day;      /* Day         (0 or 1) */
   uint8_t month;    /* Month       (0 or 1) */
   uint8_t year;     /* Year        (0 or 1) */
} rtc_alarm_enb_t;
```

図6.8　サンプルコードで使用する構造体

6.6　関数一覧

サンプルコードは、RTC を制御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。

表 6.8にサンプル関数を、表 6.9にAPI 関数を、表 6.10にユーザ定義関数を示します

表6.8　サンプル関数

関数名	概要
$Sub$$main	周辺機能の初期設定（$Super$$main をコールし main に分岐）
STB_CancelDeepStandby	ディープスタンバイモードの解除要因に応じた処理
RTC_GetTimeCanDeepStb	ディープスタンバイモード解除時刻取得処理
main	メイン処理
RTC_DispTimeCanDeepStb	ディープスタンバイモード解除時刻表示処理
Sample_Main	サンプルコードのメイン処理
Sample_RTC_Main	RTC サンプルコードのメイン処理
Sample_RTC_Init	RTC の初期設定
Sample_RTC_SetTime	RTC の時刻設定
Sample_RTC_GetTime	RTC の時刻表示
Sample_RTC_Start	RTC の時刻カウント動作開始
Sample_RTC_Stop	RTC の時刻カウント動作停止
Sample_RTC_DeepStandby	RTC のアラーム時刻設定およびディープスタンバイモード遷移

表6.9　API 関数

関数名	概要
R_RTC_Init	RTC の初期設定
R_RTC_Open	RTC の時刻カウント動作開始
R_RTC_Close	RTC の時刻カウント動作停止
R_RTC_SetCnt	RTC の時刻カウンタの値設定
R_RTC_GetCnt	RTC の時刻カウンタの値取得
R_RTC_SetAlarm	RTC のアラームレジスタ値設定
R_RTC_GetAlarm	RTC のアラームレジスタ値取得

表6.10　ユーザ定義関数

関数名	概要
Userdef_RTC_Init	RTC の初期設定

6.7　関数仕様

サンプルコードの関数仕様を示します。

$Sub$$main	
概　要	周辺機能の初期設定
宣　言	void $Sub$$main(void)
説　明	周辺機能の初期設定を行い、ライブラリ関数$Super$$main をコールすることで、main 関数に分岐します。 サンプルコードでは、サンプル関数 STB_CancelDeepStandby をコールすることでディープスタンバイモード解除要因に応じた処理を行い、STB、PORT、INTC、L1 キャッシュの初期設定を行い、ベクタベースアドレスを内蔵 RAM 領域に設定しています。また、IRQ および FIQ の割り込みを、ライブラリ関数__enable_irq, __enable_fiq をコールして許可しています。
引　数	なし
リターン値	なし

STB_CancelDeepStandby	
概　要	ディープスタンバイモードの解除要因に応じた処理
宣　言	void STB_CancelDeepStandby(void)
説　明	ディープスタンバイ解除要因フラグレジスタ（DSFR）の解除要因の確認フラグを参照して、ディープスタンバイモードから復帰後に本関数がコールされたかどうかを判定します。ディープスタンバイモードから復帰後にコールされた場合は、ディープスタンバイモードの解除要因に応じた処理を行い、端子の状態保持の解除を行います。 サンプルコードでは、ディープスタンバイモードの解除要因が RTC のアラーム割り込みの場合は、ディープスタンバイ解除要因フラグレジスタ（DSFR）の RTCARF ビットを"0"クリアし、RTC の各カウンタの値を、ディープスタンバイモード解除時の時刻として大容量内蔵 RAM に確保した領域に保存しています。保存した時刻は、サンプル関数 RTC_GetTimeCanDeepStb により取得できます。ディープスタンバイモードから復帰後にコールされたことを通知するためのフラグをセットします。 また、ディープスタンバイ解除要因フラグレジスタ（DSFR）の IOKEEP ビットが"1"の場合、IOKEEP ビットを"0"に設定することで端子の状態保持を解除します。
引　数	なし
リターン値	なし

RTC_GetTimeCanDeepStb

概　要	ディープスタンバイモード解除時刻取得処理	
宣　言	uint32_t RTC_GetTimeCanDeepStb(rtc_time_t * time)	
説　明	ディープスタンバイモードを解除した時刻を引数 time で指定された領域に格納します。	
引　数	rtc_time_t * time	：ディープスタンバイモードを解除した時刻
	time->second.value	：秒（0～59）
	time->minute.value	：分（0～59）
	time->hour.value	：時（0～23）
	time->week.value	：曜日（0～6）
		0：日曜日
		1：月曜日
		2：火曜日
		3：水曜日
		4：木曜日
		5：金曜日
		6：土曜日
	time->day.value	：日（1～31）
	time->month.value	：月（1～12）
	time->year.value	：年（0～9999）
リターン値	STB_GENERATE_ALARM_INT：	ディープスタンバイモードの解除が RTC アラーム割り込みにより行われた状態
	STB_NO_GENERATE_ALARM_INT：	ディープスタンバイモードの解除が RTC アラーム割り込みにより行われていない状態、またはパワーオンリセットから起動した状態
注意事項	ディープスタンバイモードを解除した時刻はサンプル関数 STB_CancelDeepStandby により保存されます。そのため、本関数は関数 STB_CancelDeepStandby の実行後にコールする必要があります。 本関数が STB_NO_GENERATE_ALARM_INT をリターンした場合は、引数 time にディープスタンバイモードを解除した時刻は格納されません。	

main	
概　要	メイン処理
宣　言	int_t main(void)
説　明	GENMAI 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ます。ディープスタンバイモードが解除後にコールされた場合は、ディープスタンバイモードを解除した時刻を表示します。 また、ボード上で LED と接続された PORT 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

RTC_DispTimeCanDeepStb	
概　要	ディープスタンバイモード解除時刻表示処理
宣　言	void RTC_DispTimeCanDeepStb(void)
説　明	ディープスタンバイモードを解除した時刻を取得するサンプル関数です。 サンプルコードでは、サンプル関数 RTC_GetTimeCanDeepStb を使用し RTC の時刻カウンタの値を取得しています。ディープスタンバイモードを解除した時刻が保存されていた場合は、BCD コード化された時刻カウンタの値を整数値に変換し、ターミナルへ表示します。
引　数	なし
リターン値	なし
注意事項	サンプル関数 STB_CancelDeepStandby により、ディープスタンバイモードを解除した時刻として RTC の時刻カウンタの値（BCD）が保存されます。

Sample_Main	
概　要	サンプルコードのメイン処理
宣　言	void Sample_Main(void)
説　明	GENMAI 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 上のターミナルからの文字入力を待ちます。ターミナルから"RTC"＋"Enter"キーを入力することで、RTC のサンプルコードを起動します。
引　数	なし
リターン値	なし

Sample_RTC_Main	
概　要	RTC サンプルコードのメイン処理
宣　言	int32_t Sample_RTC_Main(int32_t argc, char_t ** argv)
説　明	GENMAI 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 上のターミナルからの文字入力を待ちます。入力された文字にしたがって各サンプルを起動します。 "1"＋"Enter"キーの入力時、RTC の初期設定を実行します。 "2"＋"Enter"キーの入力時、RTC の時刻設定を実行します。 "3"＋"Enter"キーの入力時、RTC の時刻表示を実行します。 "4"＋"Enter"キーの入力時、RTC の時刻カウント開始を実行します。 "5"＋"Enter"キーの入力時、RTC の時刻カウント停止を実行します。 "6"＋"Enter"キーの入力時、RTC のアラーム時刻設定およびディープスタンバイモード遷移を実行します。
引　数	int32_t argc　：ターミナルから入力されたコマンド引数の数 char_t **argv　：ターミナルから入力されたコマンドへのポインタ
リターン値	COMMAND_EXIT：RTC サンプルコード処理の終了

Sample_RTC_Init	
概　要	RTC の初期設定
宣　言	int32_t Sample_RTC_Init(int32_t argc, char_t ** argv)
説　明	本関数はRTCの初期設定を行うサンプル関数です。サンプル関数 Sample_RTC_Main の文字入力を待つ処理にて"1"＋"Enter"キー入力時にコールされます。 サンプルコードでは、API 関数 R_RTC_Init を使用して RTC の初期設定を行っています。
引　数	int32_t argc　：ターミナルから入力されたコマンド引数の数 本関数内では使用していません。 char_t **argv　：ターミナルから入力されたコマンドへのポインタ 本関数内では使用していません。
リターン値	COMMAND_SUCCESS：RTC サンプルコード処理の成功

Sample_RTC_SetTime	
概　要	RTC の時刻設定
宣　言	int32_t Sample_RTC_SetTime(int32_t argc, char_t ** argv)
説　明	時刻を RTC の時刻カウンタに設定するサンプル関数です。サンプル関数 Sample_RTC_Main の文字入力を待つ処理にて"2"＋"Enter"キー入力時にコールされます。 サンプルコードでは、ターミナルから入力された時刻（10 進数）を、API 関数 R_RTC_SetCnt を使用して RTC の時刻カウンタに設定しています。時刻は、曜日、月、日、年、時、分、秒の順に、それぞれ入力します（入力有効範囲は、表 6.2を参照してください）。範囲外の数値が入力された場合は、再度入力を待ちます。"-1"が入力された場合は、その項目は RTC の時刻カウンタに設定しません。
引　数	int32_t argc　：ターミナルから入力されたコマンド引数の数 本関数内では使用していません。 char_t **argv　：ターミナルから入力されたコマンドへのポインタ 本関数内では使用していません。
リターン値	COMMAND_SUCCESS：RTC サンプルコード処理の成功
注意事項	本サンプル関数は、サンプル関数 Sample_RTC_Init により初期設定されていることを前提に動作します。

Sample_RTC_GetTime	
概　要	RTC の時刻表示
宣　言	int32_t Sample_RTC_GetTime(int32_t argc, char_t ** argv)
説　明	時刻を RTC の時刻カウンタから取得するサンプル関数です。サンプル関数 Sample_RTC_Main の文字入力を待つ処理にて"3"＋"Enter"キー入力時にコールされます。 サンプルコードでは、API 関数 R_RTC_GetCnt を使用して RTC の時刻カウンタから取得した時刻をターミナルに表示しています。
引　数	int32_t argc　：ターミナルから入力されたコマンド引数の数 本関数内では使用していません。 char_t **argv　：ターミナルから入力されたコマンドへのポインタ 本関数内では使用していません。
リターン値	COMMAND_SUCCESS：RTC サンプルコード処理の成功
注意事項	本サンプル関数は、サンプル関数 Sample_RTC_Init により初期設定されていることを前提に動作します。

Sample_RTC_Start	
概　要	RTC の時刻カウント動作開始
宣　言	int32_t Sample_RTC_Start(int32_t argc, char_t ** argv)
説　明	RTC の時刻カウント動作を開始するサンプル関数です。サンプル関数 Sample_RTC_Main の文字入力を待つ処理にて"4"＋"Enter"キー入力時にコールされます。 サンプルコードでは、RTC の時刻カウンタから取得した時刻をターミナルに表示し、API 関数 R_RTC_Open を使用して RTC の時刻カウント動作を開始しています。
引　数	int32_t argc　：ターミナルから入力されたコマンド引数の数 本関数内では使用していません。 char_t **argv　：ターミナルから入力されたコマンドへのポインタ 本関数内では使用していません。
リターン値	COMMAND_SUCCESS：RTC サンプルコード処理の成功
注意事項	本サンプル関数は、サンプル関数 Sample_RTC_Init により初期設定されていることを前提に動作します。

Sample_RTC_Stop	
概　要	RTC の時刻カウント動作停止
宣　言	int32_t Sample_RTC_Stop(int32_t argc, char_t ** argv)
説　明	RTC の時刻カウント動作を停止するサンプル関数です。サンプル関数 Sample_RTC_Main の文字入力を待つ処理にて"5"＋"Enter"キー入力時にコールされます。 サンプルコードでは、API 関数 R_RTC_Close を使用して RTC の時刻カウント動作を停止し、RTC の時刻カウンタから取得した時刻をターミナルに表示しています。
引　数	int32_t argc　：ターミナルから入力されたコマンド引数の数 本関数内では使用していません。 char_t **argv　：ターミナルから入力されたコマンドへのポインタ 本関数内では使用していません。
リターン値	COMMAND_SUCCESS：RTC サンプルコード処理の成功
注意事項	本サンプル関数は、サンプル関数 Sample_RTC_Init により初期設定されていることを前提に動作します。

Sample_RTC_DeepStandby		
概　要	RTC のアラーム時刻設定およびディープスタンバイモード遷移	
宣　言	int32_t Sample_RTC_DeepStandby(int32_t argc, char_t ** argv)	
説　明	ディープスタンバイモードを解除する要因として、アラーム割り込みを選択し、ディープスタンバイモードへ遷移するサンプル処理です。 サンプル関数 Sample_RTC_Main の文字入力を待つ処理にて"6"＋"Enter"キー入力時にコールされます。 サンプルコードでは、ターミナルから入力された時刻（時、分）にアラーム割り込みが発生するように、時アラームレジスタ（RHRAR）、分アラームレジスタ（RMINAR）に入力された時刻、秒アラームレジスタ（RSECAR）に"0"を設定し、それぞれのレジスタの ENB ビットをセットします。RTC の時刻カウンタから時刻を読み出し、ディープスタンバイモードへ遷移する直前の時刻を、ターミナルへ表示します。RTC アラーム割り込みによりディープスタンバイモードを解除するように STB を設定し、ディープスタンバイモードへ遷移します。ディープスタンバイモード解除後は、外部メモリ（CS0 空間に接続された NOR フラッシュメモリ）から起動するように STB を設定しています。 ターミナルから入力された時刻（時、分）がともに"-1"の場合、または RTC の時刻カウンタから時刻を読み出すときにエラーが発生した場合は、本関数はエラーを返します。	
引　数	int32_t argc	：ターミナルから入力されたコマンド引数の数 本関数内では使用していません。
	char_t **argv	：ターミナルから入力されたコマンドへのポインタ 本関数内では使用していません。
リターン値	COMMAND_SUCCESS	：RTC サンプルコード処理の成功
	COMMAND_ERROR	：RTC サンプルコード処理の失敗
注意事項	本サンプル関数は、サンプル関数 Sample_RTC_Init により初期設定を行い、RTC が時刻カウント動作中であることを前提に動作します。	

R_RTC_Init	
概　要	RTC の初期設定
宣　言	void R_RTC_Init(void)
説　明	RTC の初期設定を行います。 ユーザ定義関数 Userdef_RTC_Init をコールし、Userdef_RTC_Init で RTC の初期設定を行います。
引　数	なし
リターン値	なし

R_RTC_Open	
概　要	RTC の時刻カウント動作開始
宣　言	void R_RTC_Open(void)
説　明	RTC の時刻カウント動作を開始に設定します。 RTC は時刻カウンタの現在時刻を元に時刻カウントを開始します。
引　数	なし
リターン値	なし
注意事項	本 API 関数をコールする前に、API 関数 R_RTC_SetCnt にて有効な時刻を RTC に設定してください。

R_RTC_Close	
概　要	RTC の時刻カウント動作停止
宣　言	void R_RTC_Close(void)
説　明	RTC の時刻カウント動作を停止に設定します。
引　数	なし
リターン値	なし

R_RTC_SetCnt

概　要	RTC の時刻カウンタの値設定	
宣　言	int32_t R_RTC_SetCnt(rtc_time_t * time)	
説　明	引数 time で指定された時刻を RTC の時刻カウンタに設定します。 コントロールレジスタ 2（RCR2）の RESET ビットをセットし、引数 time->second.enable に RTC_ENABLE が設定されていた場合は、引数 time->second.value の時刻を BCD コード化し、RTC の秒カウンタ（RSECCNT）に書き込みます。time->second.enable に RTC_DISABLE が設定されていた場合は、書き込みを行いません。その他の時刻のメンバについても同様に RTC の各カウンタに書き込みます。 RTC が時刻カウント動作中の場合は時刻カウンタへの書き込みが禁止されているため、本関数の始めに時刻カウント動作を停止し、関数の終わりに時刻カウント動作を開始します。	
引　数	rtc_time_t * time	：時刻
		time->second.value　：秒（0～59）
		time->minute.value　：分（0～59）
		time->hour.value　：時（0～23）
		time->week.value　：曜日（0～6） 0：日曜日 1：月曜日 2：火曜日 3：水曜日 4：木曜日 5：金曜日 6：土曜日
		time->day.value　：日（1～31）
		time->month.value　：月（1～12）
		time->year.value　：年（0～9999）
		設定対象の指定（RTC_ENABLE：設定する、RTC_DISABLE：設定しない）
		time->second.enable　：秒カウンタの設定
		time->minute.enable　：分カウンタの設定
		time->hour.enable　：時カウンタの設定
		time->week.enable　：曜日カウンタの設定
		time->day.enable　：日カウンタの設定
		time->month.enable　：月カウンタの設定
		time->year.enable　：年カウンタの設定
リターン値	DEVDRV_SUCCESS	：RTC の時刻カウンタの値設定成功
	DEVDRV_ERROR	：RTC の時刻カウンタの値設定失敗
注意事項	本 API 関数をコールする前に API 関数 R_RTC_Close をコールし､時刻カウント動作を停止してください。	

R_RTC_GetCnt

概　要	RTC の時刻カウンタの値取得	
宣　言	int32_t R_RTC_GetCnt(rtc_time_t * time)	
説　明	RTC の時刻カウンタから時刻を取得し、引数 time で指定された領域に格納します。 引数 time->second.enable に RTC_ENABLE が設定されていた場合は､RTC の秒カウンタ（RSECCNT）から読み出した、BCD コード化された時刻を整数値に変換し、引数 time->second.value に格納します｡time->second.enable に RTC_DISABLE が設定されていた場合は、読み出しを行いません。その他の時刻のメンバについても同様に RTC の各カウンタから読み出します。 RTC の時刻カウンタからの読み出し処理では、コントロールレジスタ 1（RCR1）の桁上げ（CF）フラグを"0"にクリアし時刻カウンタからカウント値を読み出し後に桁上げフラグを確認します｡時刻カウンタからの読み出し時に桁上げフラグがセットされていた場合はその読み出しは無効と判断し､再度時刻カウンタからの読み出し処理を行います。読み出し処理を 2 回実行しても桁上げフラグがセットされていた場合、本関数は DEVDRV_ERROR を返します。	
引　数	rtc_time_t * time	：取得時刻の格納領域
	time->second.value	：秒（0～59）
	time->minute.value	：分（0～59）
	time->hour.value	：時（0～23）
	time->week.value	：曜日（0～6） 0：日曜日 1：月曜日 2：火曜日 3：水曜日 4：木曜日 5：金曜日 6：土曜日
	time->day.value	：日（1～31）
	time->month.value	：月（1～12）
	time->year.value	：年（0～9999）
	取得対象の指定（RTC_ENABLE：取得する、RTC_DISABLE：取得しない）	
	time->second.enable	：秒カウンタの取得
	time->minute.enable	：分カウンタの取得
	time->hour.enable	：時カウンタの取得
	time->week.enable	：曜日カウンタの取得
	time->day.enable	：日カウンタの取得
	time->month.enable	：月カウンタの取得
	time->year.enable	：年カウンタの取得
リターン値	DEVDRV_SUCCESS	：RTC の時刻カウンタの値取得成功
	DEVDRV_ERROR	：RTC の時刻カウンタの値取得失敗

R_RTC_SetAlarm

概　要	RTC のアラームレジスタ値設定	
宣　言	int32_t R_RTC_SetAlarm(rtc_time_t * time, rtc_alarm_enb_t * alarm_enb)	
説　明	引数 time で指定されたアラーム時刻を RTC のアラームレジスタに設定します。また、アラームを有効とする時刻（秒、分、時、曜日、日、月、年）は、引数 alarm_enb のメンバにより指定します。 アラーム割り込みが許可された状態で本関数がコールされた場合は、アラーム割り込みを禁止します。引数 time->second.enable に RTC_ENABLE が設定されていた場合は、引数 time->second.value のアラーム時刻を BCD コード化し、RTC の秒アラームレジスタ（RSECAR）に書き込みます。time->second.enable に RTC_DISABLE が設定されていた場合は、書き込みを行いません。その他のアラーム時刻のメンバについても同様に RTC の各アラームレジスタに書き込みます。 引数 alarm_enb->second の値を、RTC の秒アラームレジスタ（RSECAR）の ENB ビットに書き込みます。その他のアラーム時刻の有効設定情報のメンバについても同様に RTC の各アラームレジスタの ENB ビットに書き込みます。 アラームフラグ（コントロールレジスタ 1（RCR1）の AF ビット）を"0"にクリアします。アラーム割り込みが許可された状態で本関数がコールされた場合は、アラーム割り込みを許可します。 ENB ビットに"1"が設定されたアラームレジスタの時刻と、時刻カウンタが一致したときに、アラームフラグが"1"にセットされます。アラームフラグにより、現在時刻がアラーム時刻となったことを知ることができます。	
引　数	rtc_time_t * time	：アラーム時刻
	time->second.value	：秒（0～59）
	time->minute.value	：分（0～59）
	time->hour.value	：時（0～23）
	time->week.value	：曜日（0～6） 0：日曜日、1：月曜日 2：火曜日、3：水曜日 4：木曜日、5：金曜日 6：土曜日
	time->day.value	：日（1～31）
	time->month.value	：月（1～12）
	time->year.value	：年（0～9999）
	設定対象の指定（RTC_ENABLE：設定する、RTC_DISABLE：設定しない）	
	time->second.enable	：秒アラームの設定
	time->minute.enable	：分アラームの設定
	time->hour.enable	：時アラームの設定
	time->week.enable	：曜日アラームの設定
	time->day.enable	：日アラームの設定
	time->month.enable	：月アラームの設定
	time->year.enable	：年アラームの設定
	rtc_alarm_enb_t * alarm_enb	：アラーム時刻の有効設定情報 （0：アラーム時刻無効、1：アラーム時刻有効）
	alarm_enb->second	：秒（0 または 1）
	alarm_enb->minute	：分（0 または 1）
	alarm_enb->hour	：時（0 または 1）
	alarm_enb->week	：曜日（0 または 1）
	alarm_enb->day	：日（0 または 1）
	alarm_enb->month	：月（0 または 1）
	alarm_enb->year	：年（0 または 1）
リターン値	DEVDRV_SUCCESS	：RTC のアラームレジスタ値設定成功
	DEVDRV_ERROR	：RTC のアラームレジスタ値設定失敗

R_RTC_GetAlarm

概　要	RTC のアラームレジスタ値取得	
宣　言	int32_t R_RTC_GetAlarm(rtc_time_t * time, rtc_alarm_enb_t * alarm_enb)	
説　明	RTC のアラームレジスタからアラーム時刻を取得し、引数 time で指定された領域に格納します。 引数 time->second.enable に RTC_ENABLE が設定されていた場合は、RTC の秒アラームレジスタ（RSECAR）から読み出した、BCD コード化されたアラーム時刻を整数値に変換し、引数 time->second.value に格納します。time->second.enable に RTC_DISABLE が設定されていた場合は、読み出しを行いません。その他のアラーム時刻のメンバについても同様に RTC の各アラームレジスタから読み出します。 RTC の秒アラームレジスタ（RSECAR）の ENB ビットからアラーム時刻の有効設定情報を読み出し、引数 alarm_enb->second に格納します。その他のアラーム時刻の有効設定情報のメンバについても同様に RTC の各アラームレジスタの ENB ビットから読み出します。	
引　数	rtc_time_t * time	：取得したアラーム時刻の格納領域
		time->second.value　：秒（0～59）
		time->minute.value　：分（0～59）
		time->hour.value　：時（0～23）
		time->week.value　：曜日（0～6） 0：日曜日 1：月曜日 2：火曜日 3：水曜日 4：木曜日 5：金曜日 6：土曜日
		time->day.value　：日（1～31）
		time->month.value　：月（1～12）
		time->year.value　：年（0～9999）
		取得対象の指定（RTC_ENABLE：取得する、RTC_DISABLE：取得しない）
		time->second.enable　：秒アラームの取得
		time->minute.enable　：分アラームの取得
		time->hour.enable　：時アラームの取得
		time->week.enable　：曜日アラームの取得
		time->day.enable　：日アラームの取得
		time->month.enable　：月アラームの取得
		time->year.enable　：年アラームの取得
	rtc_alarm_enb_t * alarm_enb	：取得したアラーム時刻の有効設定情報 （0：アラーム時刻無効、1：アラーム時刻有効）
		alarm_enb->second　：秒（0 または 1）
		alarm_enb->minute　：分（0 または 1）
		alarm_enb->hour　：時（0 または 1）
		alarm_enb->week　：曜日（0 または 1）
		alarm_enb->day　：日（0 または 1）
		alarm_enb->month　：月（0 または 1）
		alarm_enb->year　：年（0 または 1）
リターン値	DEVDRV_SUCCESS	：RTC のアラームレジスタ値取得成功
	DEVDRV_ERROR	：RTC のアラームレジスタ値取得失敗

Userdef_RTC_Init	
概　要	RTC の初期設定
宣　言	void Userdef_RTC_Init(void)
説　明	ユーザ定義関数です。RTC の初期設定を行ってください。 サンプルコードでは、RTC のモジュールスタンバイ解除後、時刻カウント動作を停止し、桁上げ割り込み、アラーム割り込み、周期割り込みを禁止にしています。動作クロックとして RTC_X1 からの 32.768kHz を選択し、内蔵水晶発振器を動作するように RTC を設定しています。また、RTC のコントロールレジスタ 2（RCR2）の RESET ビットをセットしています。
引　数	なし
リターン値	なし

6.8　フローチャート

6.8.1　周辺機能の初期設定（$Sub$$main 関数）

図 6.9～図 6.10に周辺機能の初期設定（$Sub$$main 関数）のフローチャートを示します。

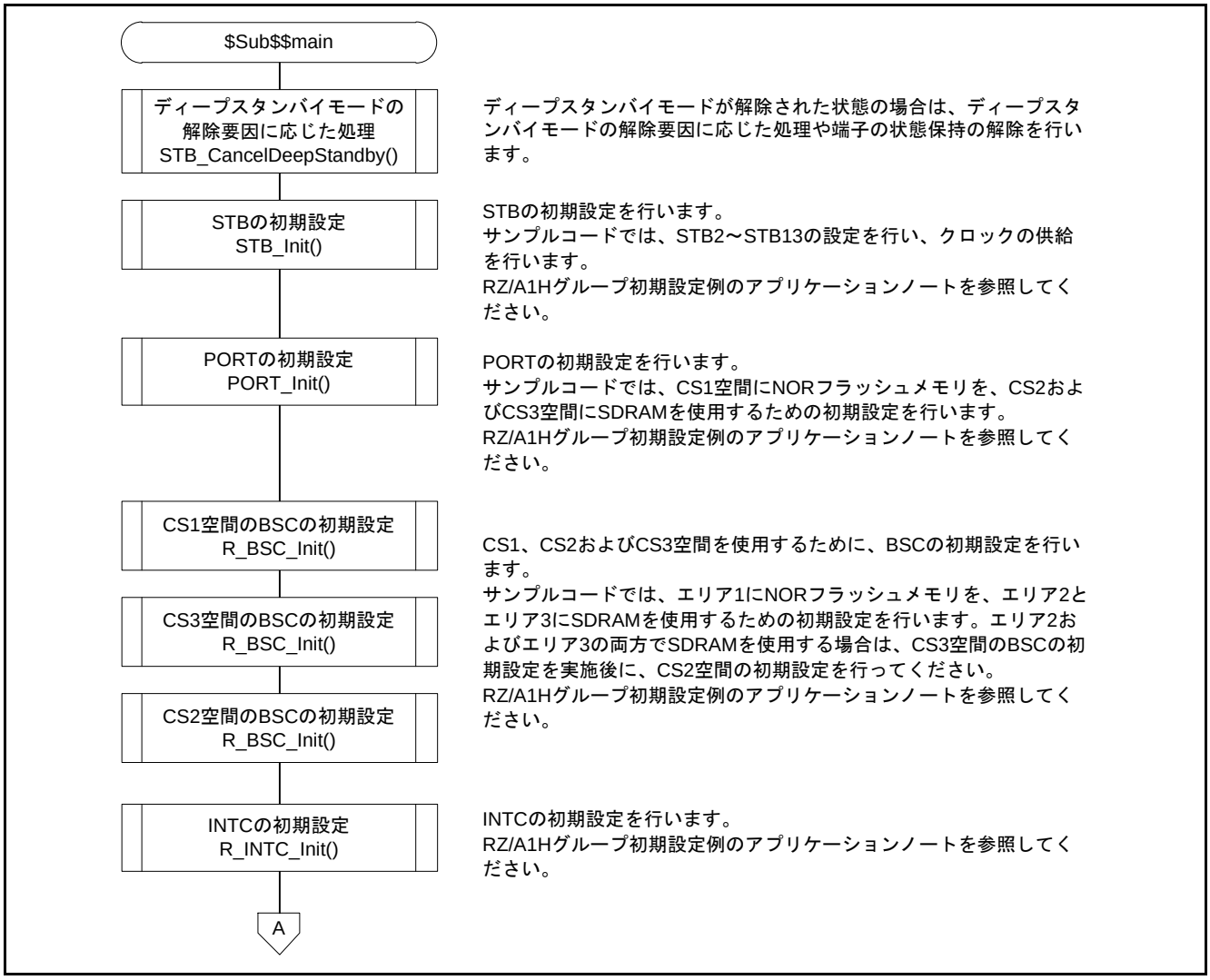


図6.9　周辺機能の初期設定（$Sub$$main 関数）（1/2）

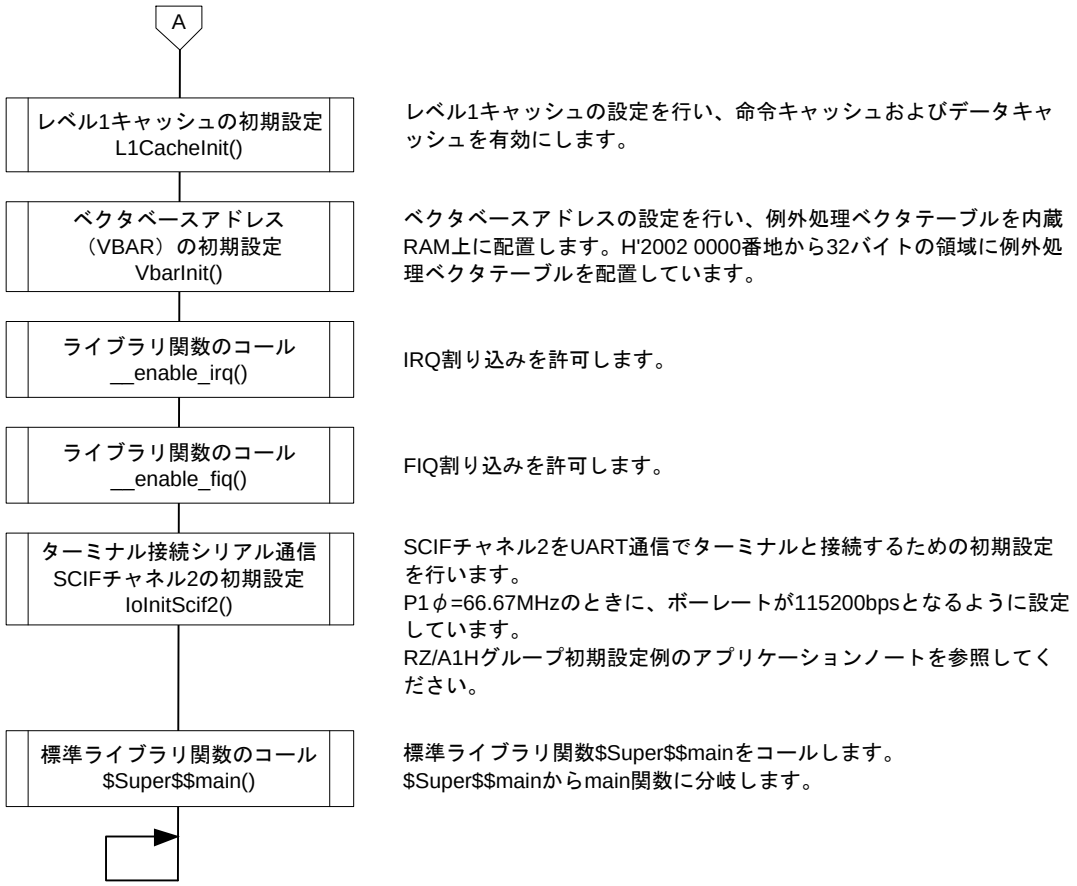


図6.10　周辺機能の初期設定（$Sub$$main 関数）（2/2）

6.8.2　ディープスタンバイモードの解除要因に応じた処理

図 6.11にディープスタンバイモードの解除要因に応じた処理のフローチャートを示します。

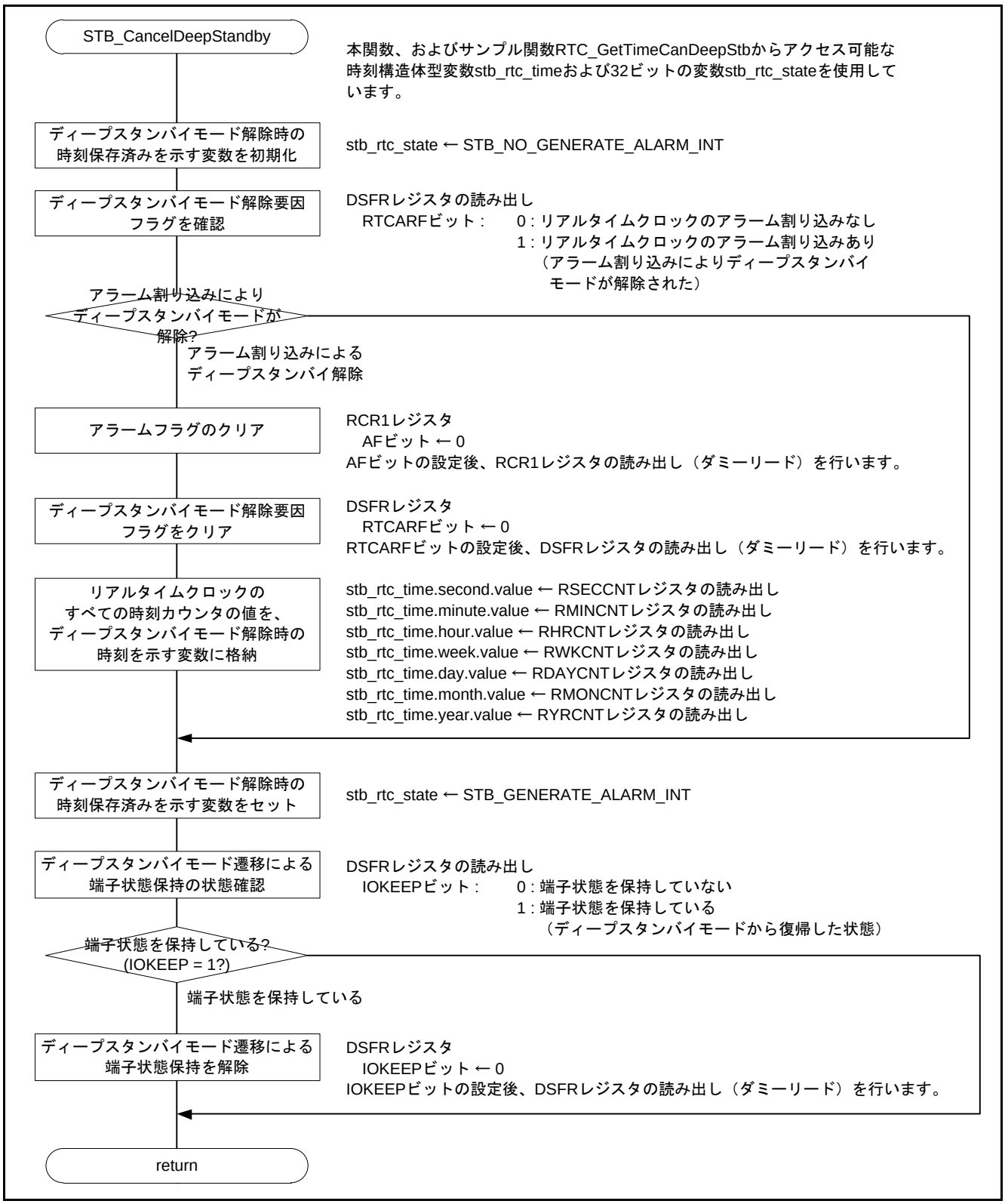


図6.11　ディープスタンバイモードの解除要因に応じた処理

6.8.3　ディープスタンバイモード解除時刻取得処理

図 6.12にディープスタンバイモード解除時刻取得処理のフローチャートを示します。

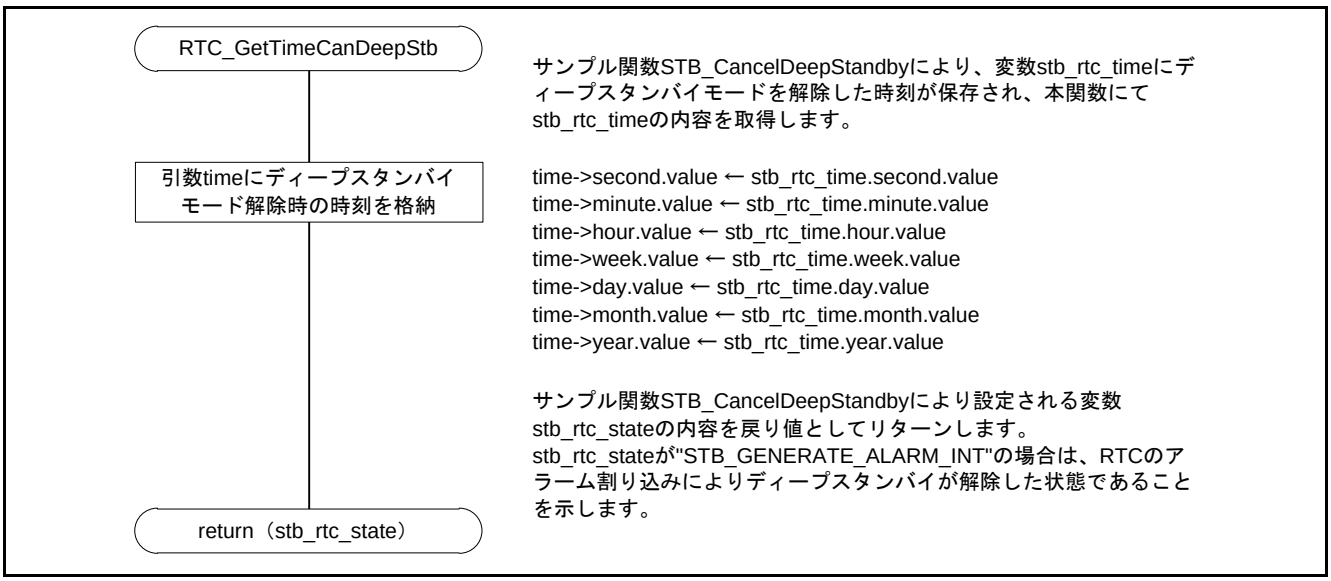


図6.12　ディープスタンバイモード解除時刻取得処理

6.8.4　メイン処理

図 6.13にメイン処理のフローチャートを示します。

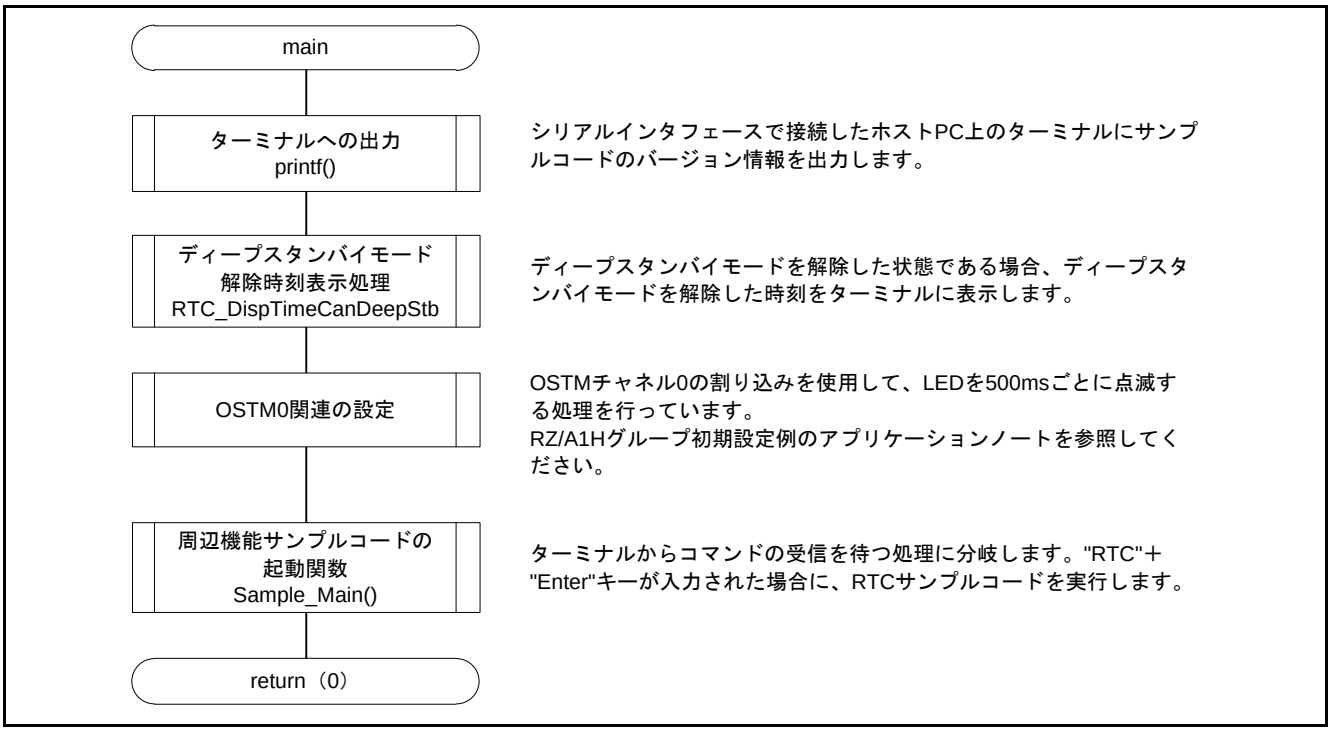


図6.13　メイン処理

6.8.5　ディープスタンバイモード解除時刻表示処理

図 6.14にディープスタンバイモード解除時刻表示処理のフローチャートを示します。

ディープスタンバイモードを解除した時刻を、サンプル関数 RTC_GetTimeCanDeepStb により取得し、BCD コード化された時刻カウンタの値を整数値に変換しターミナルに表示します。

ディープスタンバイモードを解除した時刻として RTC の時刻カウンタの値（BCD）は、サンプル関数 STB_CancelDeepStandby により格納します。

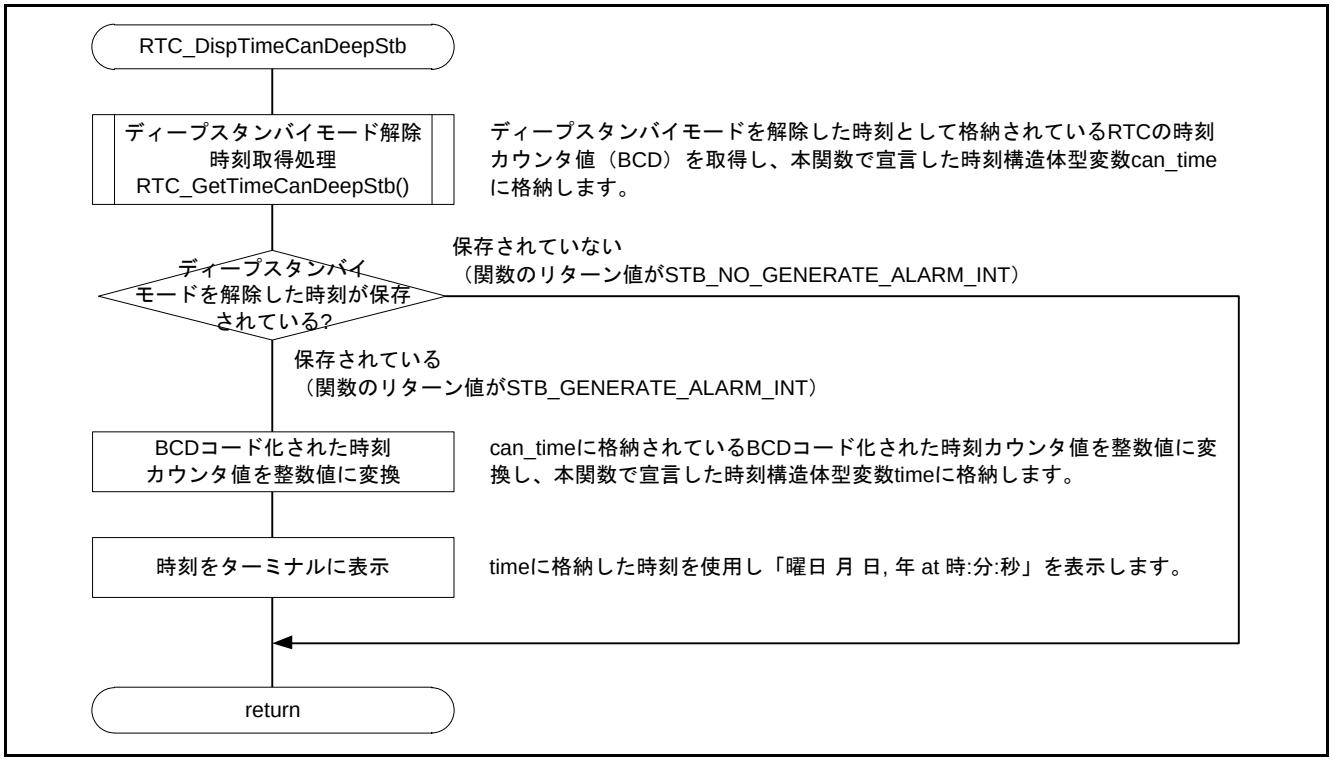


図6.14　ディープスタンバイモード解除時刻表示処理

6.8.6　サンプルコードのメイン処理

図 6.15にサンプルコードのメイン処理のフローチャートを示します。この関数では、ホスト PC 上のターミナルからの文字入力を待ちます。

"RTC"＋"Enter"キーが入力されたときに、RTC サンプルコードを実行します。

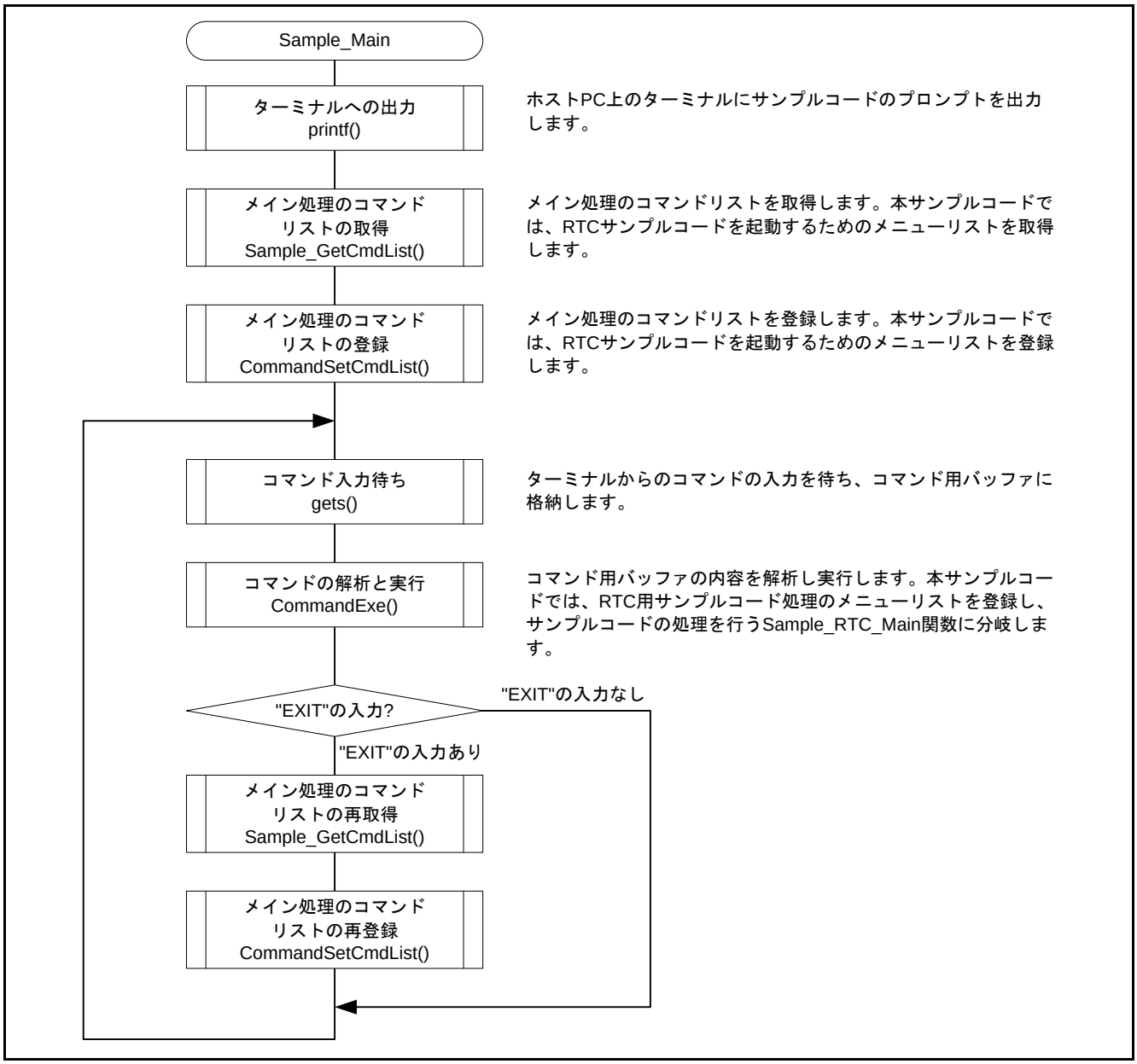


図6.15　サンプルコードのメイン処理

6.8.7　RTC サンプルコードのメイン処理

図 6.16にRTC サンプルコードのメイン処理のフローチャートを示します。この関数では、ホスト PC 上のターミナルからの文字入力を待ち、入力されたコマンドにしたがって、RTC サンプルコードの処理に分岐します。

"1"＋"Enter"キーが入力されたときに、RTC の初期設定を実行します。

"2"＋"Enter"キーが入力されたときに、RTC の時刻設定を実行します。

"3"＋"Enter"キーが入力されたときに、RTC の時刻表示を実行します。

"4"＋"Enter"キーが入力されたときに、RTC の時刻カウント開始を実行します。

"5"＋"Enter"キーが入力されたときに、RTC の時刻カウント停止を実行します。

"6"＋"Enter"キーが入力されたときに、RTC のアラーム時刻設定およびディープスタンバイモード遷移を実行します。

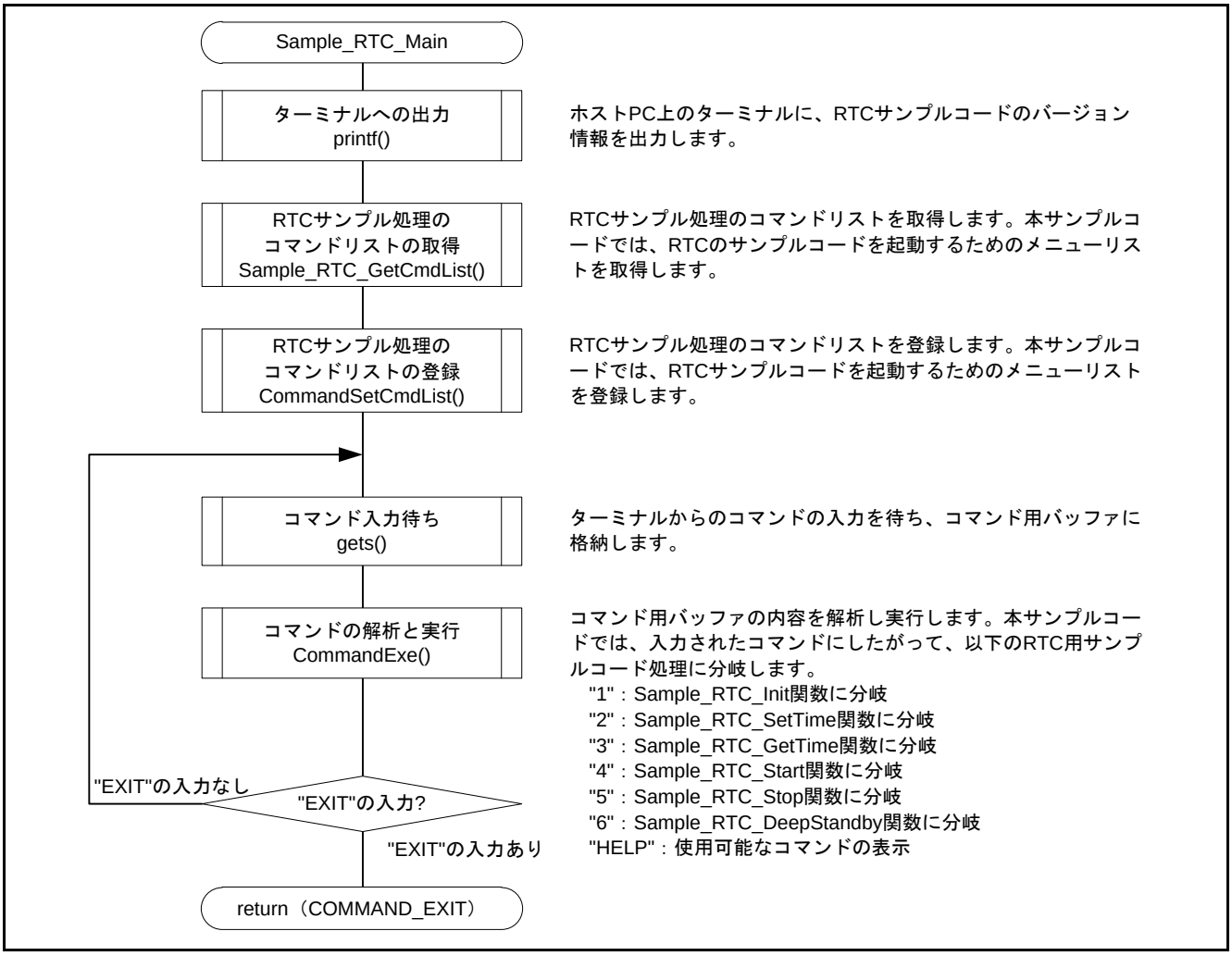
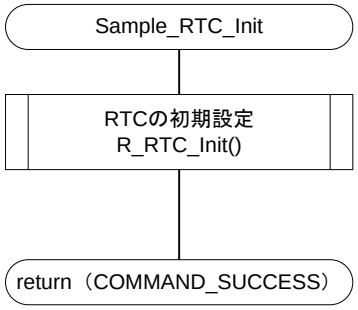


図6.16　RTC サンプルコードのメイン処理

6.8.8　RTC の初期設定

図 6.17にRTC の初期設定のフローチャートを示します。

サンプル関数 Sample_RTC_Main の RTC コマンド待ち処理にて、コマンド 1 を入力したときに動作する関数です。



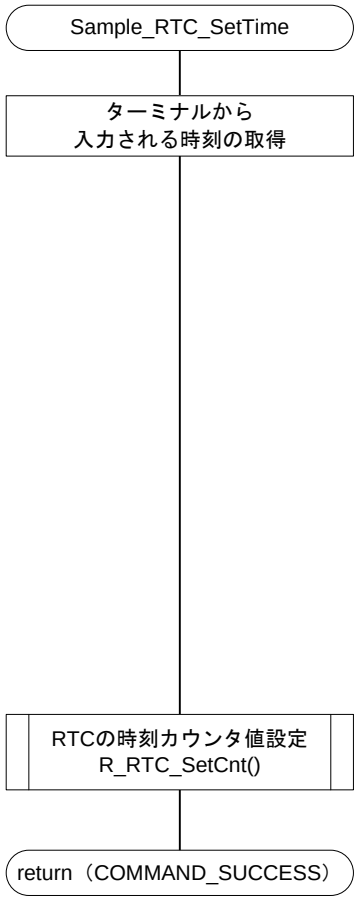
RTCのモジュールスタンバイ解除後、時刻カウント動作を停止し、桁上げ割り込み、アラーム割り込み、周期割り込みを禁止にしています。動作クロックとしてRTC_X1からの32.768kHzを選択し、内蔵水晶発振器を動作するようにRTCを設定しています。また、RTCのコントロールレジスタ2（RCR2）のRESETビットをセットしています。

図6.17　RTC の初期設定

6.8.9　RTC の時刻設定

図 6.18にRTC の時刻設定のフローチャートを示します。

サンプル関数 Sample_RTC_Main の RTC コマンド待ち処理にて、コマンド 2 を入力したときに動作する関数です。



ターミナルからの時刻の入力を待ち、本関数で宣言した時刻構造体型変数timeに入力された値を格納します。時刻にはそれぞれ上限および下限を設け、範囲外の値が入力された場合は再度入力を待ちます。範囲内の値が入力された場合は、time.xxxx.valueに格納し、time.xxxx.enableにRTC_ENABLEを格納します。"-1"が入力された場合は、time.xxxx.enableにRTC_DISABLEを格納します。

曜日	time.week.value	0～6
	time.week.enable	RTC_DISABLEまたはRTC_ENABLE
月	time.month.value	1～12
	time.month.enable	RTC_DISABLEまたはRTC_ENABLE
日	time.day.value	1～31
	time.day.enable	RTC_DISABLEまたはRTC_ENABLE
年	time.year.value	0～9999
	time.year.enable	RTC_DISABLEまたはRTC_ENABLE
時	time.hour.value	0～23
	time.hour.enable	RTC_DISABLEまたはRTC_ENABLE
分	time.minute.value	0～59
	time.minute.enable	RTC_DISABLEまたはRTC_ENABLE
秒	time.second.value	0～59
	time.second.enable	RTC_DISABLEまたはRTC_ENABLE

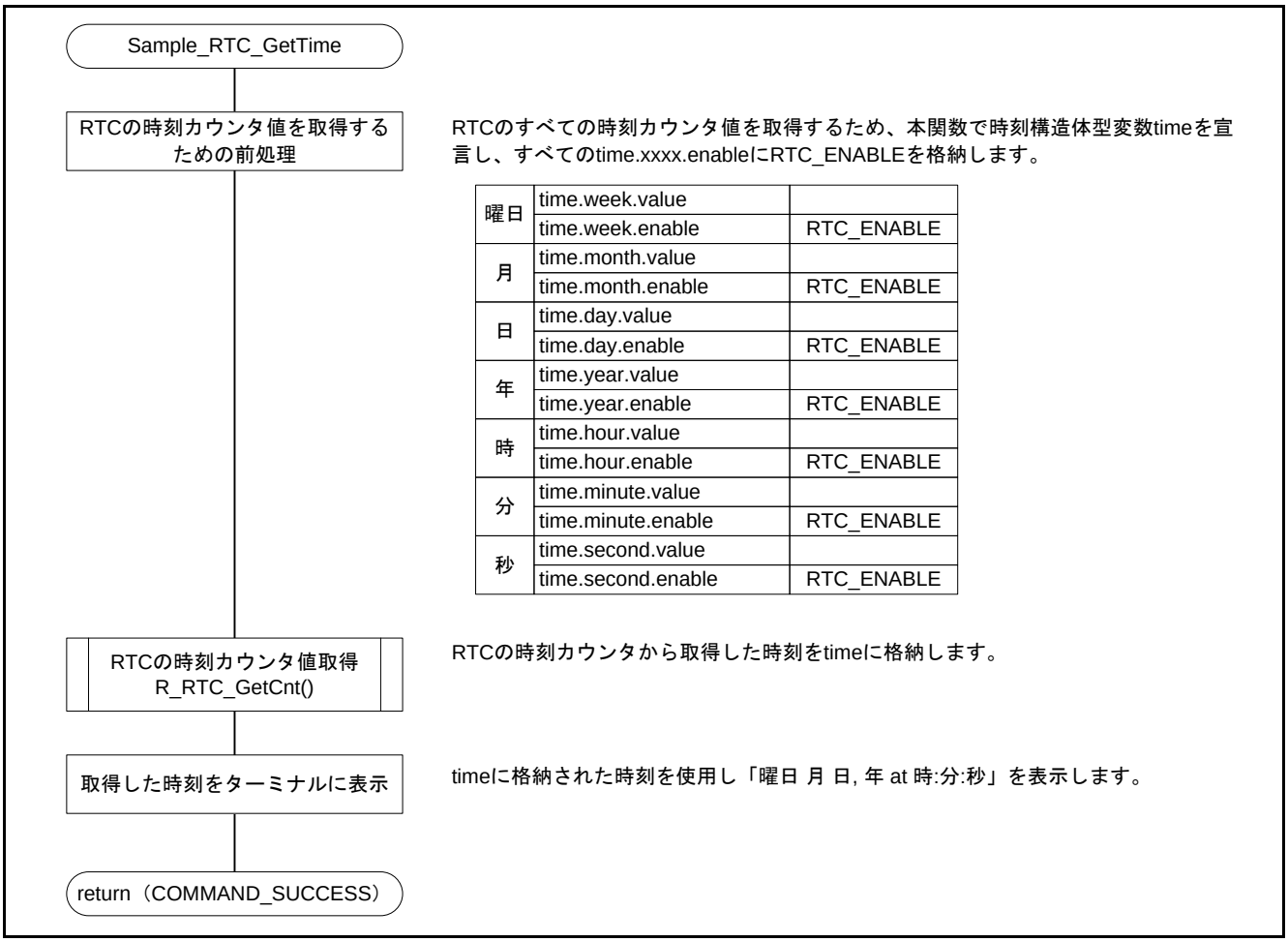
timeに格納した時刻をRTCの時刻カウンタに設定します。ただしtime.xxxx.enableにRTC_ENABLEが設定されている時刻のみ設定を行います。

図6.18　RTC の時刻設定

6.8.10　RTC の時刻表示

図 6.19にRTC の時刻表示のフローチャートを示します。

サンプル関数 Sample_RTC_Main の RTC コマンド待ち処理にて、コマンド 3 を入力したときに動作する関数です。



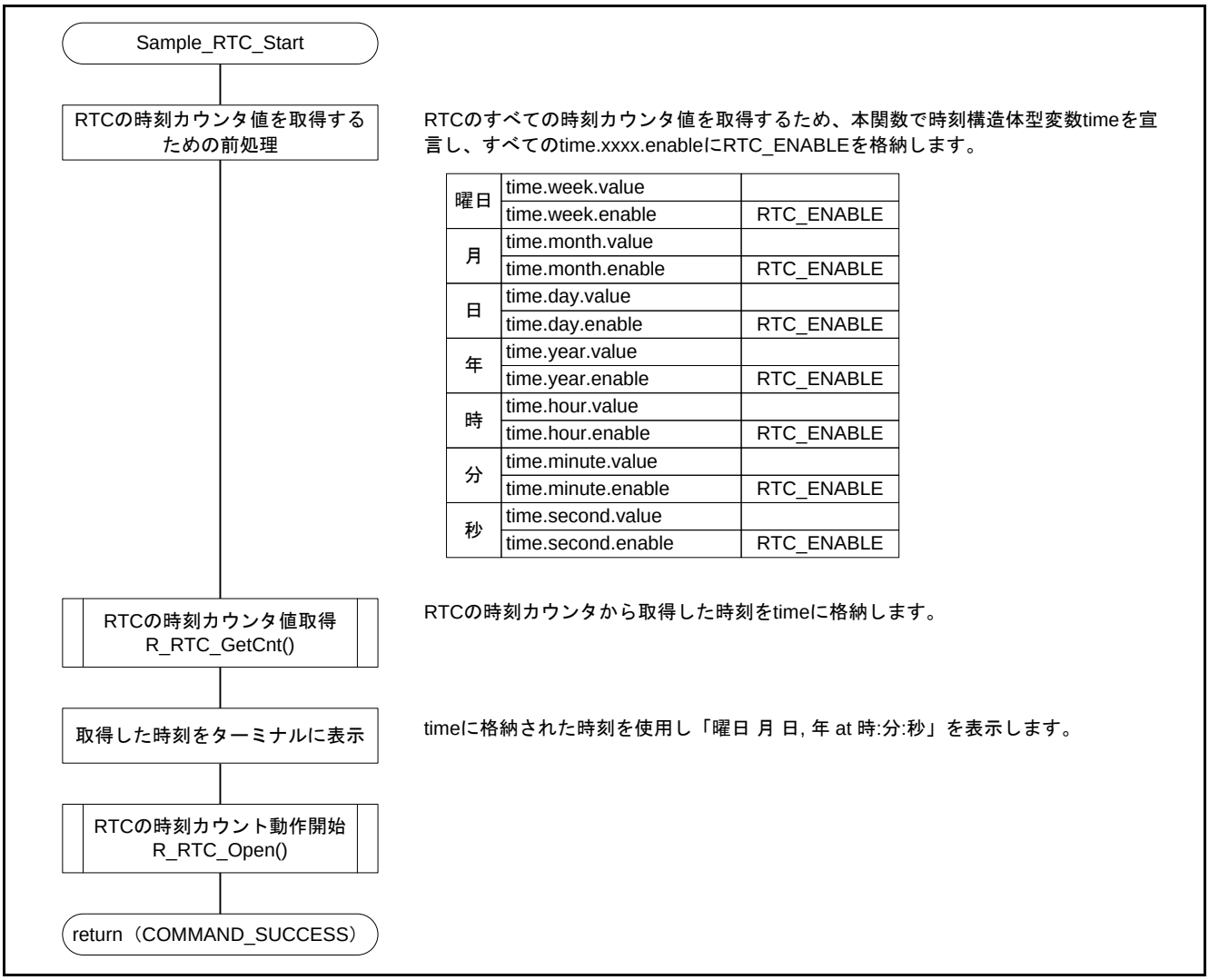
曜日	time.week.value	
	time.week.enable	RTC_ENABLE
月	time.month.value	
	time.month.enable	RTC_ENABLE
日	time.day.value	
	time.day.enable	RTC_ENABLE
年	time.year.value	
	time.year.enable	RTC_ENABLE
時	time.hour.value	
	time.hour.enable	RTC_ENABLE
分	time.minute.value	
	time.minute.enable	RTC_ENABLE
秒	time.second.value	
	time.second.enable	RTC_ENABLE

図6.19　RTC の時刻表示

6.8.11　RTC の時刻カウント動作開始

図 6.20にRTC の時刻カウント動作開始のフローチャートを示します。

サンプル関数 Sample_RTC_Main の RTC コマンド待ち処理にて、コマンド 4 を入力したときに動作する関数です。



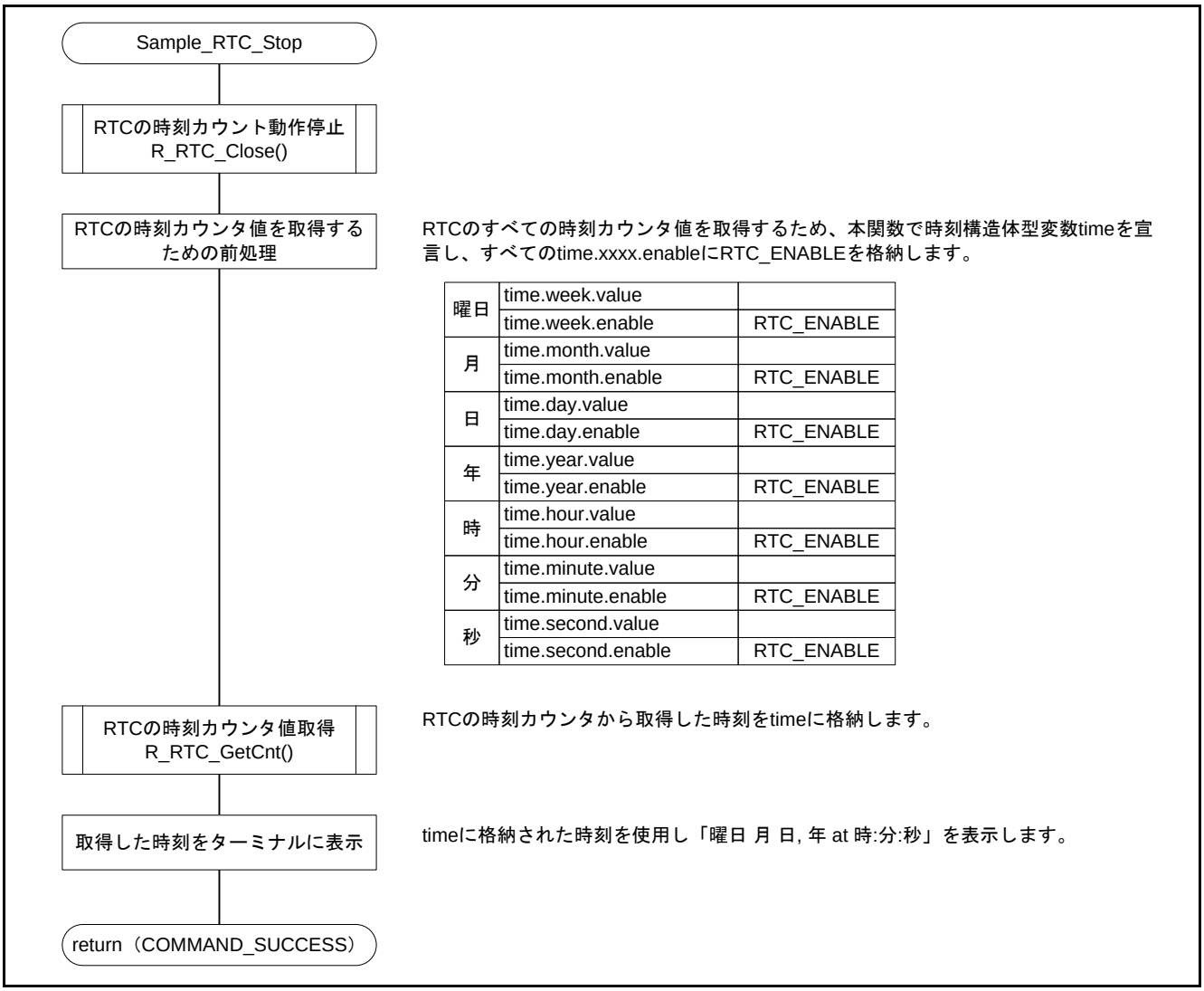
曜日	time.week.value	
	time.week.enable	RTC_ENABLE
月	time.month.value	
	time.month.enable	RTC_ENABLE
日	time.day.value	
	time.day.enable	RTC_ENABLE
年	time.year.value	
	time.year.enable	RTC_ENABLE
時	time.hour.value	
	time.hour.enable	RTC_ENABLE
分	time.minute.value	
	time.minute.enable	RTC_ENABLE
秒	time.second.value	
	time.second.enable	RTC_ENABLE

図6.20　RTC の時刻カウント動作開始

6.8.12　RTC の時刻カウント動作停止

図 6.21にRTC の時刻カウント動作停止のフローチャートを示します。

サンプル関数 Sample_RTC_Main の RTC コマンド待ち処理にて、コマンド 5 を入力したときに動作する関数です。



曜日	time.week.value	
	time.week.enable	RTC_ENABLE
月	time.month.value	
	time.month.enable	RTC_ENABLE
日	time.day.value	
	time.day.enable	RTC_ENABLE
年	time.year.value	
	time.year.enable	RTC_ENABLE
時	time.hour.value	
	time.hour.enable	RTC_ENABLE
分	time.minute.value	
	time.minute.enable	RTC_ENABLE
秒	time.second.value	
	time.second.enable	RTC_ENABLE

図6.21　RTC の時刻カウント動作停止

6.8.13　RTC のアラーム時刻設定およびディープスタンバイモード遷移

図 6.22～図 6.24にRTC のアラーム時刻設定およびディープスタンバイモード遷移のフローチャートを示します。

サンプル関数 Sample_RTC_Main の RTC コマンド待ち処理にて、コマンド 6 を入力したときに動作する関数です。

ターミナルから入力された時刻にアラーム割り込みが発生するように RTC を設定します。アラーム割り込みによりディープスタンバイモードを解除するように STB を設定し、ディープスタンバイモードへ遷移します。

本関数の処理は、RTC の時刻カウンタが動作していることを前提としています。

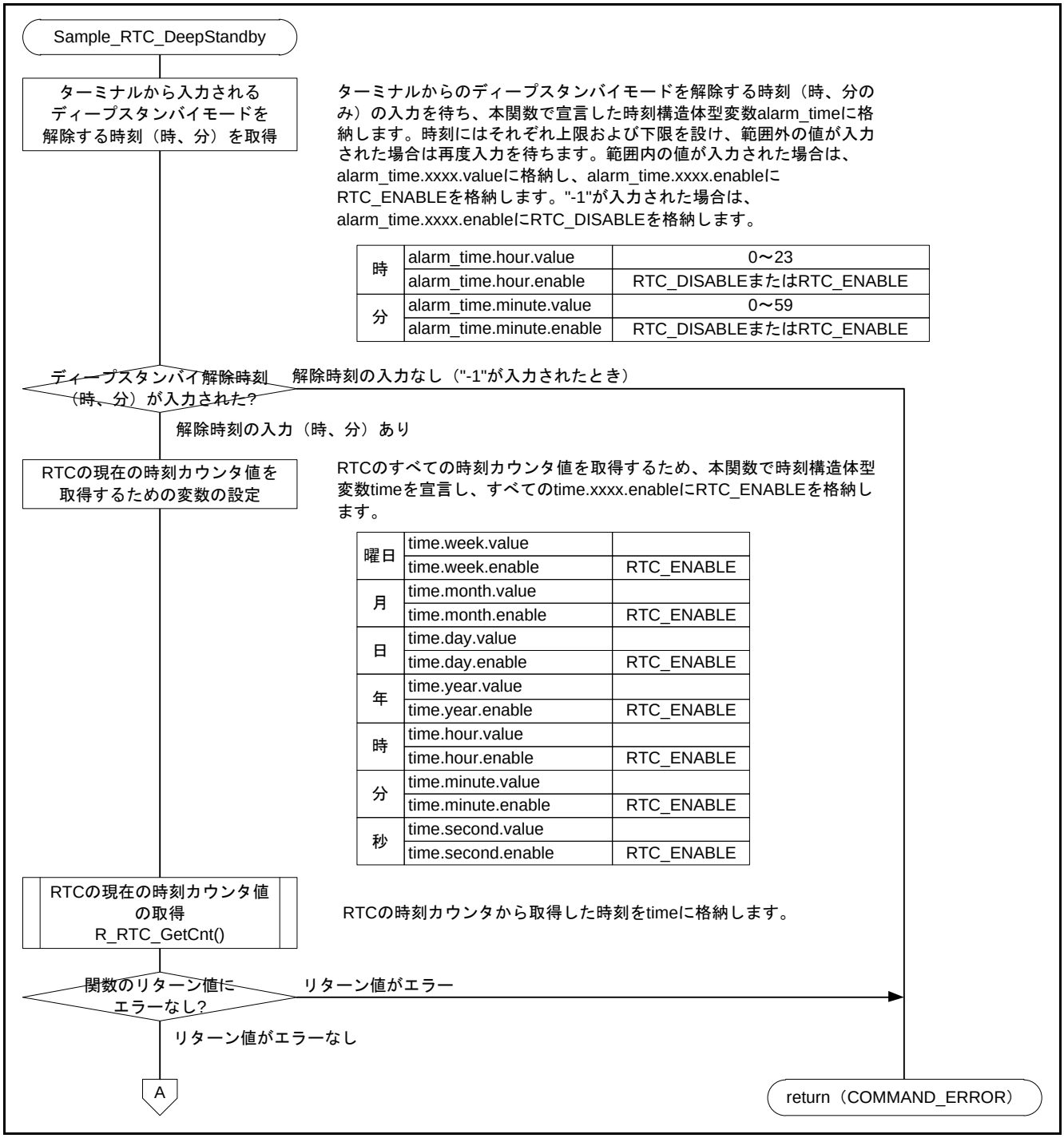
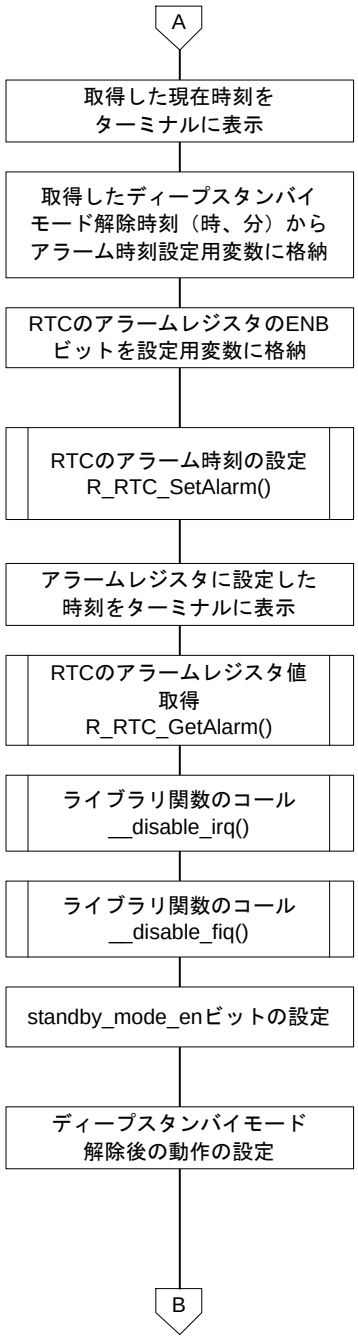


図6.22　RTC のアラーム時刻設定およびディープスタンバイモード遷移（1/3）



timeに格納された時刻を使用し、ディープスタンバイモードへ遷移する時刻として「曜日 月 日, 年 at 時:分:秒」を表示します。

取得したディープスタンバイモード解除時刻（時、分）を、アラーム時刻を設定するための変数に格納します。本サンプルコードでは、取得した時刻をtime.hour.value、time.minute.valueに格納し、time.second.valueには"0"を格納します。

アラーム時刻の有効設定情報の構造体alarm_enbのメンバを設定します。本サンプルコードでは、alarm_enb.hour、alarm_enb.minute、alarm_enb.secondに"1"を格納し、時、分、秒のアラーム時刻を有効にするようにしています。

timeのアラーム設定時刻、alarm_enbのアラーム時刻の有効設定情報を、RTCのアラームレジスタに設定します。

timeに格納された時刻を使用し、ディープスタンバイモードを解除する時刻として「時:分:秒」を表示します。

RTCのアラームレジスタのダミーリードを行います。

IRQ割り込みを禁止します。

FIQ割り込みを禁止します。

PL310のPower Control Register ← H'0000 0001（standby_mode_enビット= 1）
PL310のPower Control Registerの設定後、PL310のPower Control Registerの読み出し（ダミーリード）を行います。

DSCTRレジスタ ← H'00
EBUSKEEPEビット = 0　：ディープスタンバイモードから復帰後、外部メモリ制御端子の状態を保持しない
RAMBOOTビット = 0　：ディープスタンバイモードから復帰後の起動方法はブートモードに従う
DSCTRレジスタの設定後、DSCTRレジスタの読み出し（ダミーリード）を行います。

図6.23　RTC のアラーム時刻設定およびディープスタンバイモード遷移（2/3）

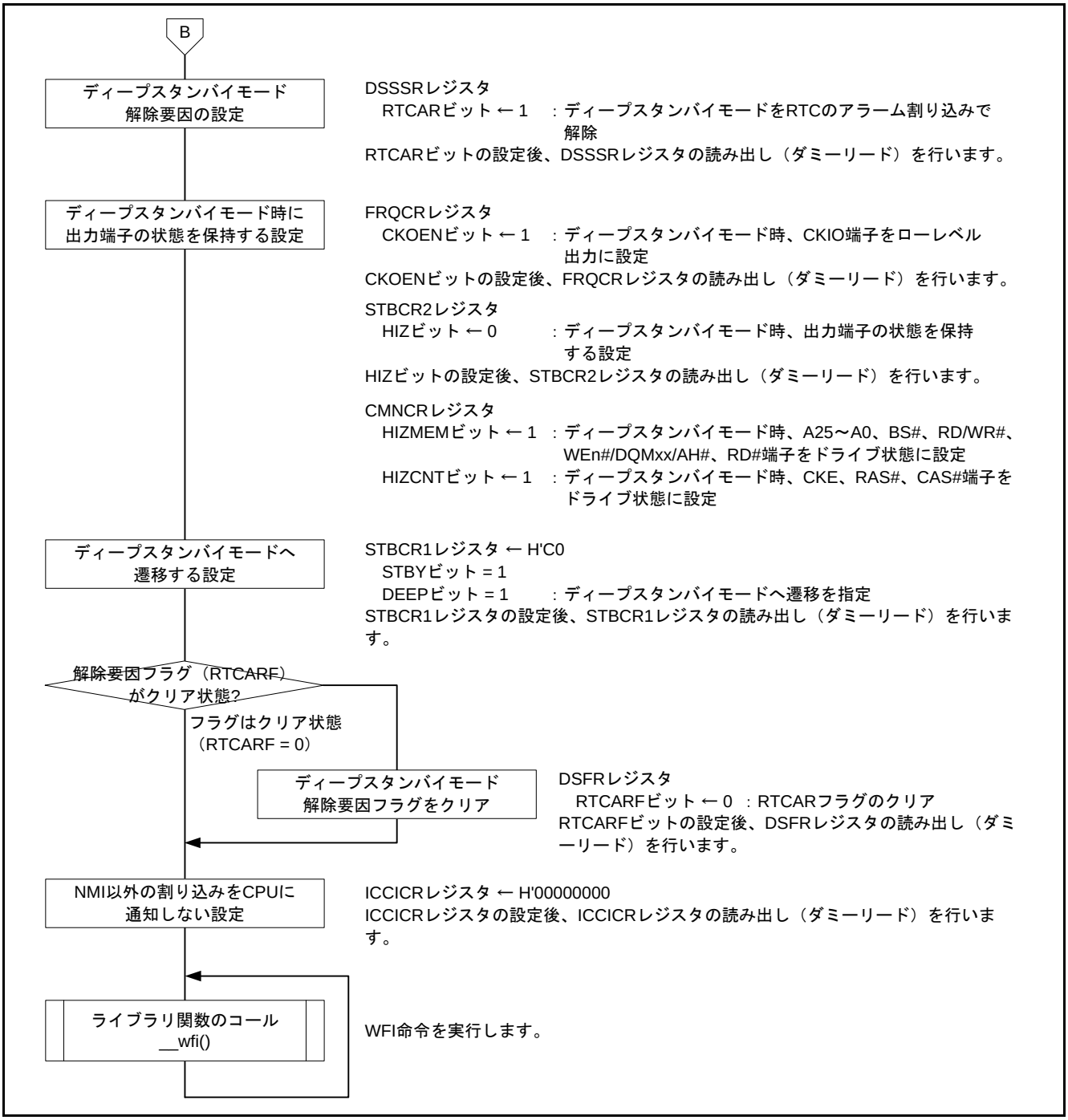


図6.24　RTC のアラーム時刻設定およびディープスタンバイモード遷移（3/3）

6.8.14　RTC の初期設定関数

図 6.25にRTC の初期設定関数のフローチャートを示します。

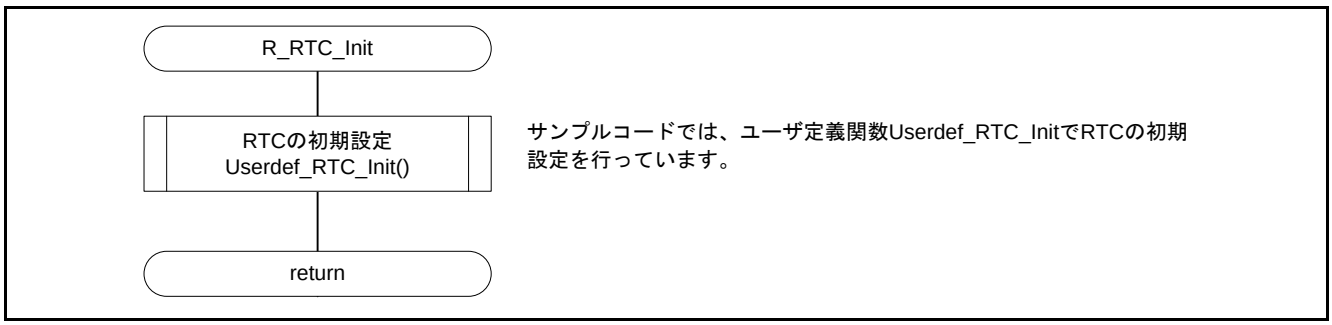


図6.25　RTC の初期設定関数

6.8.15　RTC の時刻カウント動作開始関数

図 6.26にRTC の時刻カウント動作開始関数のフローチャートを示します。

図6.26　RTC の時刻カウント動作開始関数

6.8.16　RTC の時刻カウント動作停止関数

図 6.27にRTC の時刻カウント動作停止関数のフローチャートを示します。

図6.27　RTC の時刻カウント動作停止関数

6.8.17　RTC の時刻カウンタの値設定関数

図 6.28および図 6.29にRTC の時刻カウンタの値設定関数のフローチャートを示します。

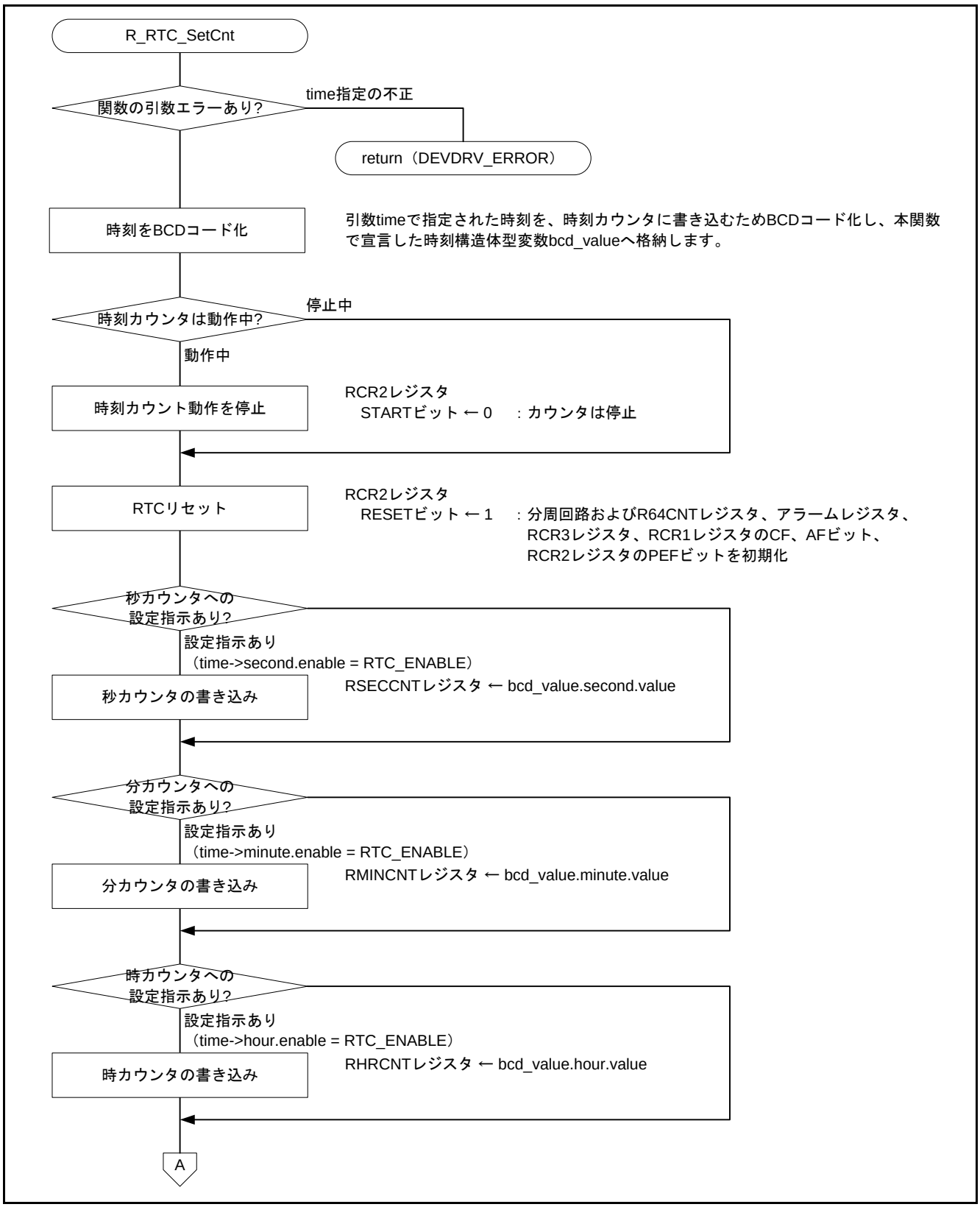


図6.28　RTC の時刻カウンタの値設定関数（1/2）

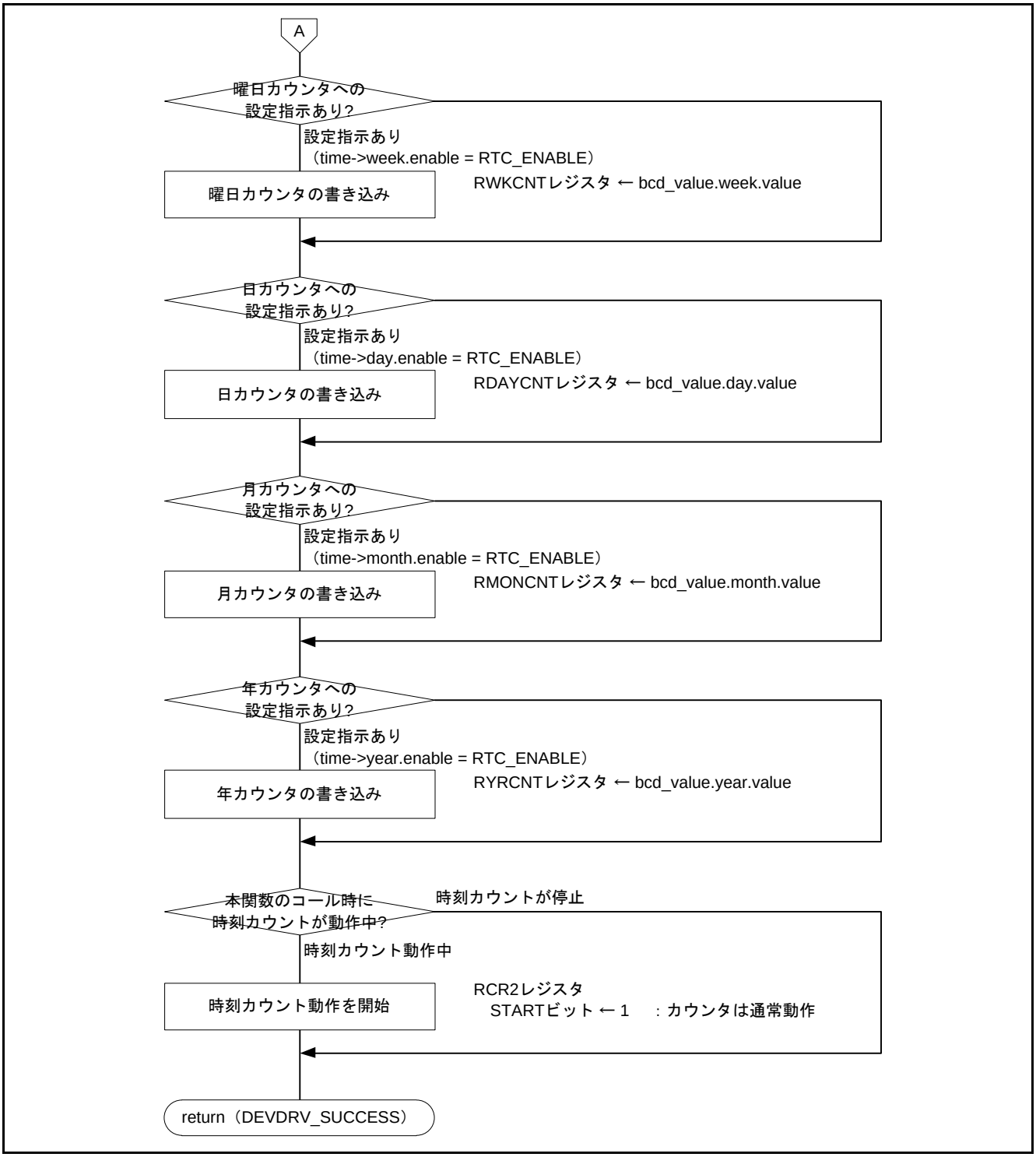


図6.29　RTC の時刻カウンタの値設定関数（2/2）

6.8.18　RTC の時刻カウンタの値取得関数

図 6.30および図 6.31にRTC の時刻カウンタの値取得関数のフローチャートを示します。

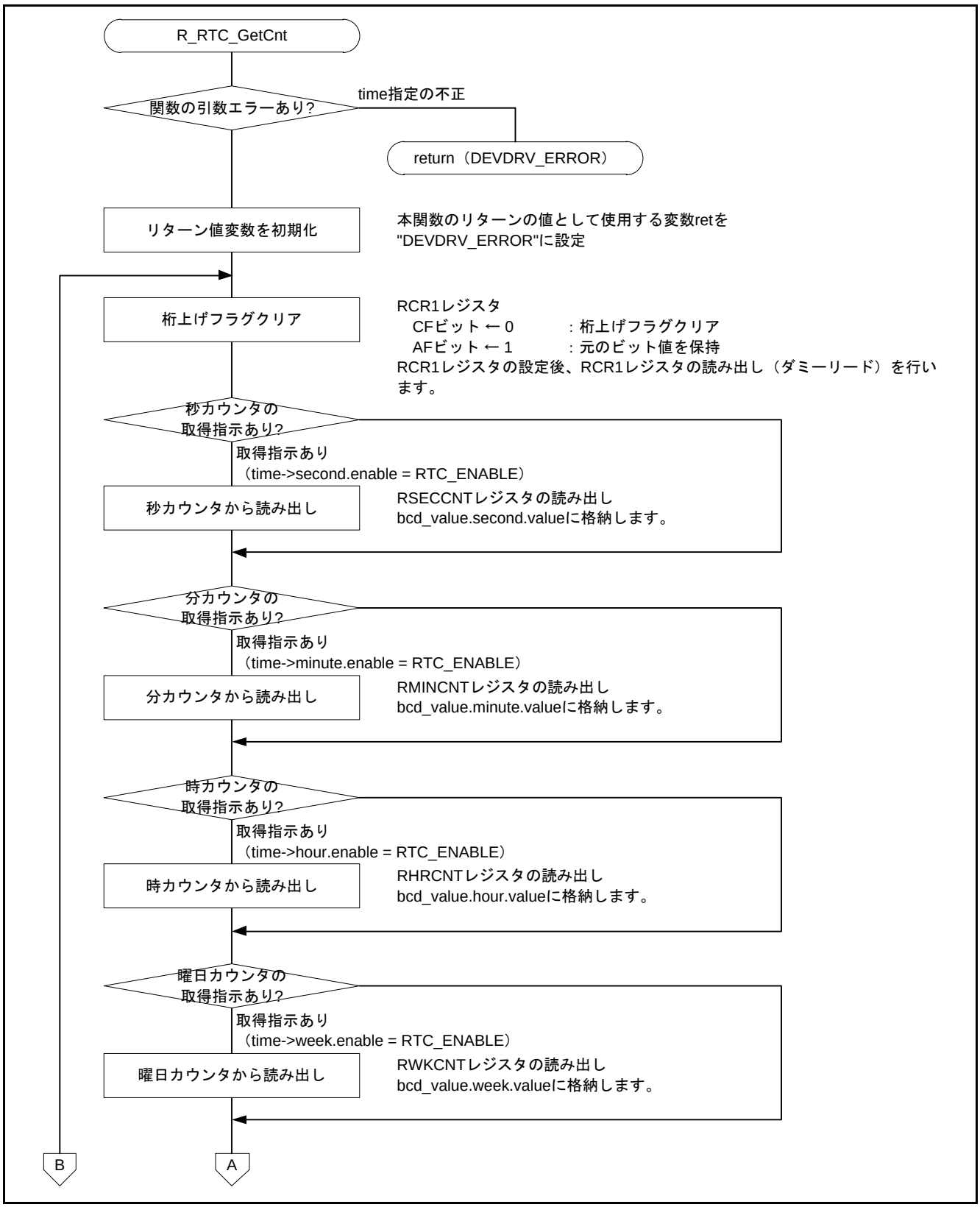


図6.30　RTC の時刻カウンタの値取得関数（1/2）

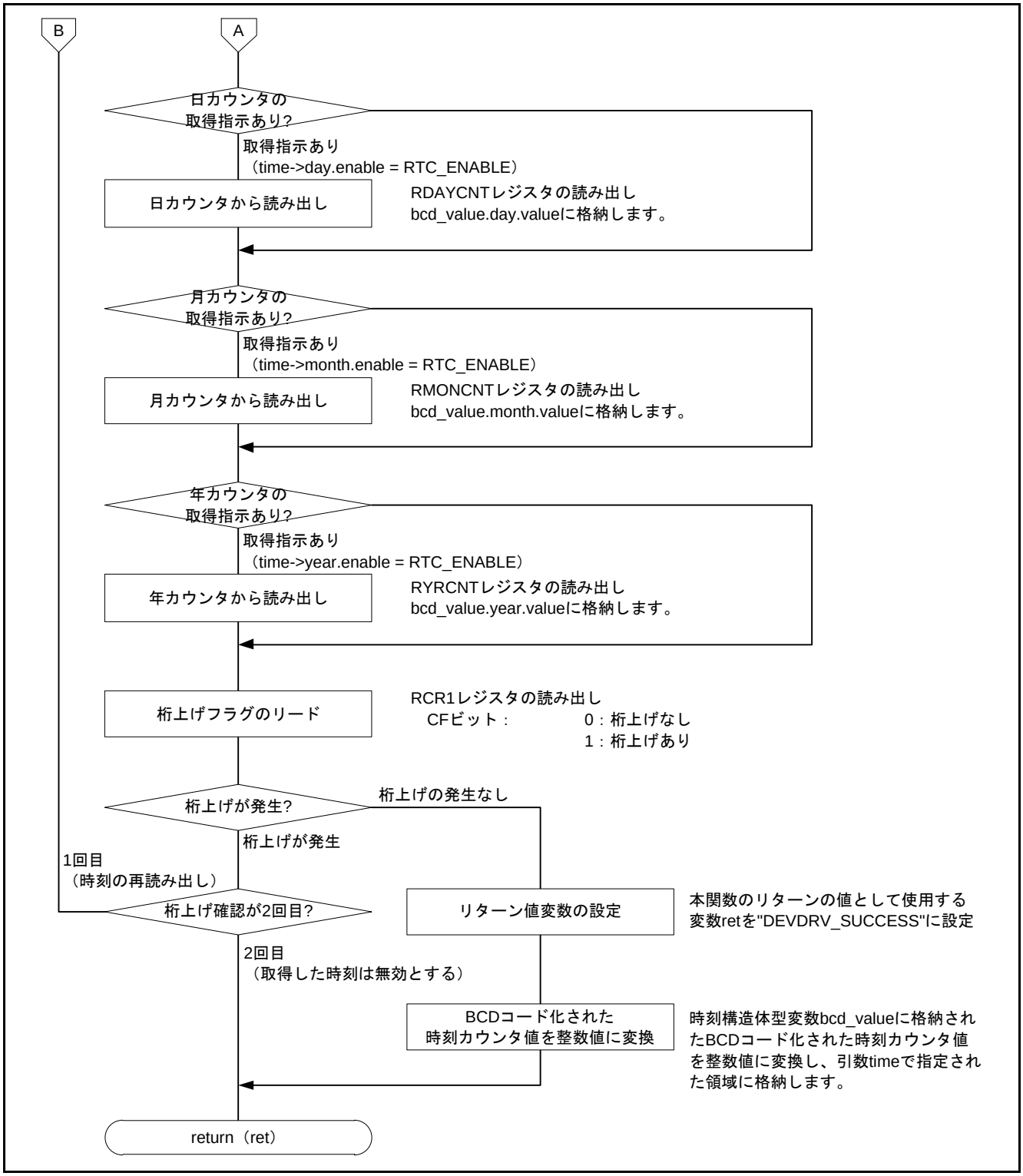


図6.31　RTC の時刻カウンタの値取得関数（2/2）

6.8.19　RTC のアラームレジスタ値設定関数

図 6.32～図 6.34にRTC のアラームレジスタ値設定関数のフローチャートを示します。

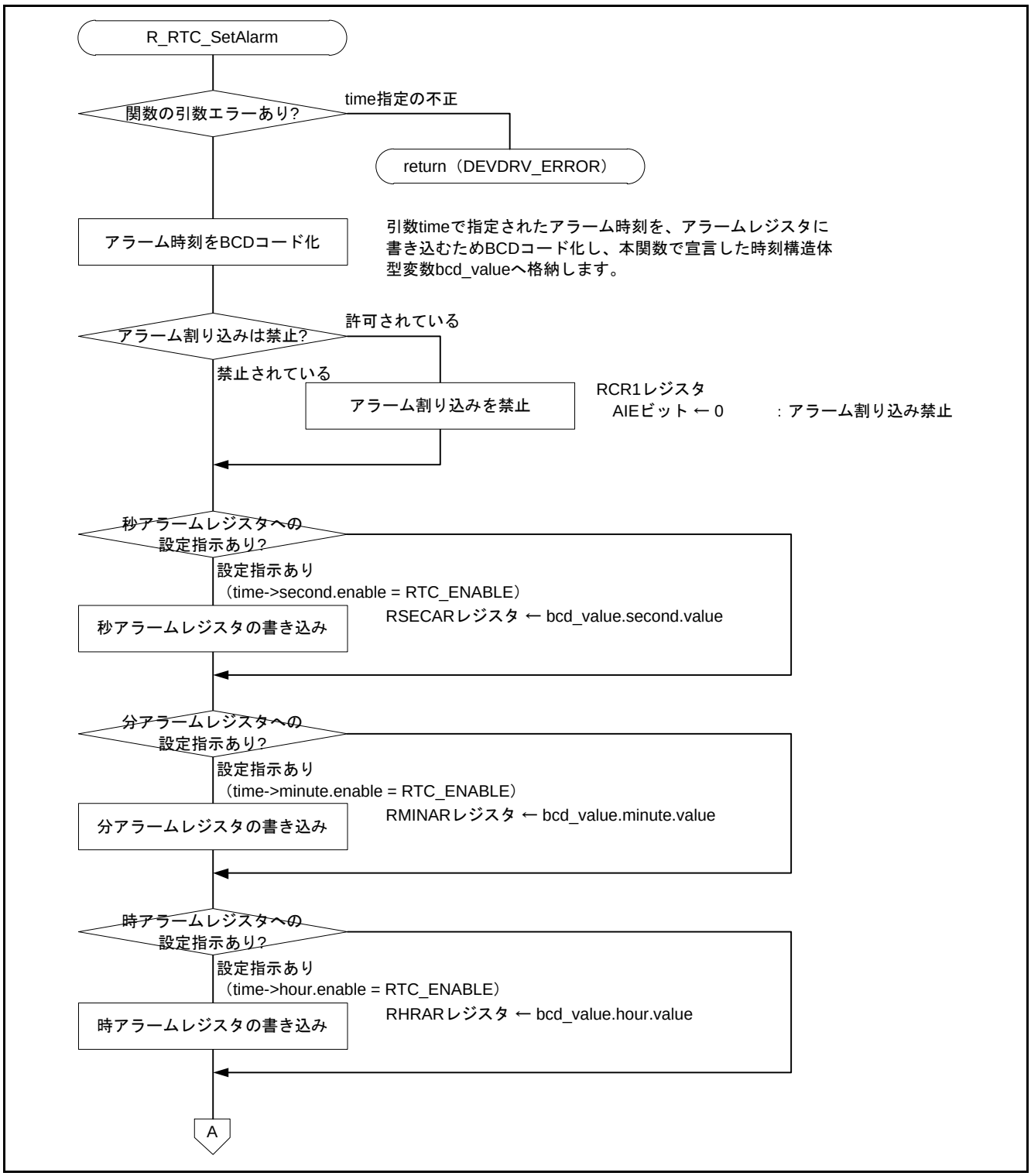


図6.32　RTC のアラームレジスタ値設定関数（1/3）

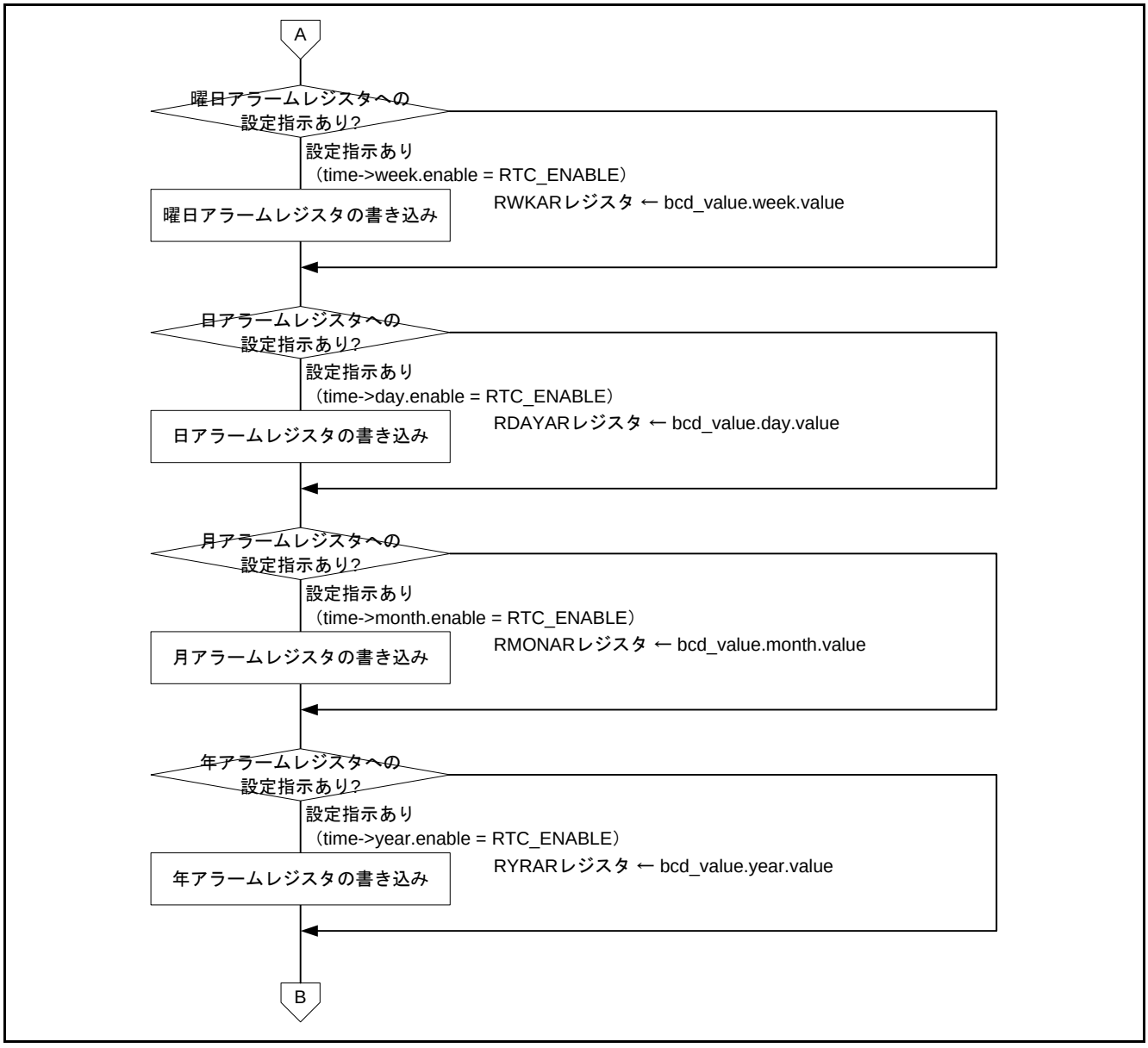


図6.33　RTC のアラームレジスタ値設定関数（2/3）

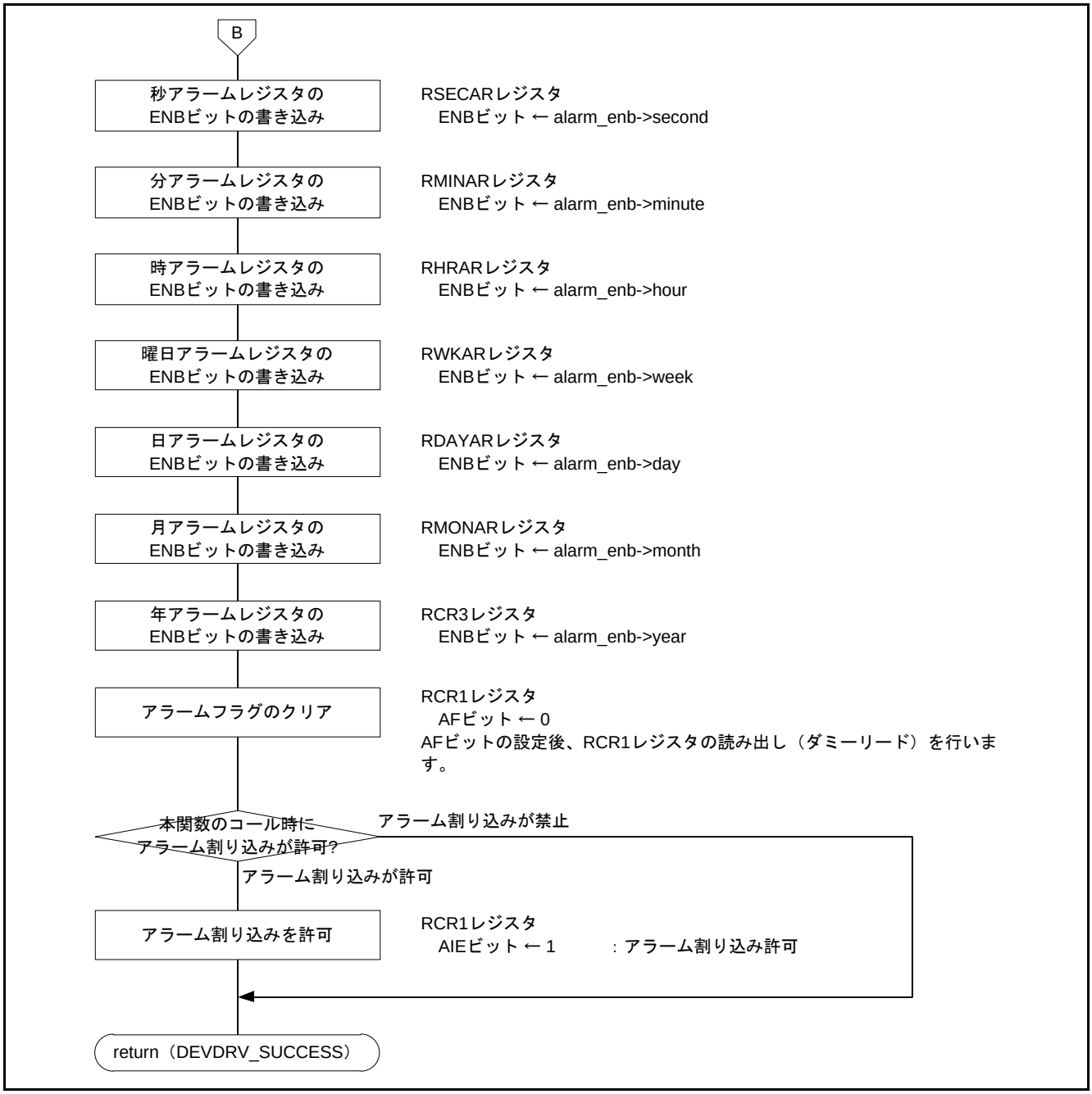


図6.34　RTC のアラームレジスタ値設定関数（3/3）

6.8.20　RTC のアラームレジスタ値取得関数

図 6.35～図 6.37にRTC のアラームレジスタ値取得関数のフローチャートを示します。

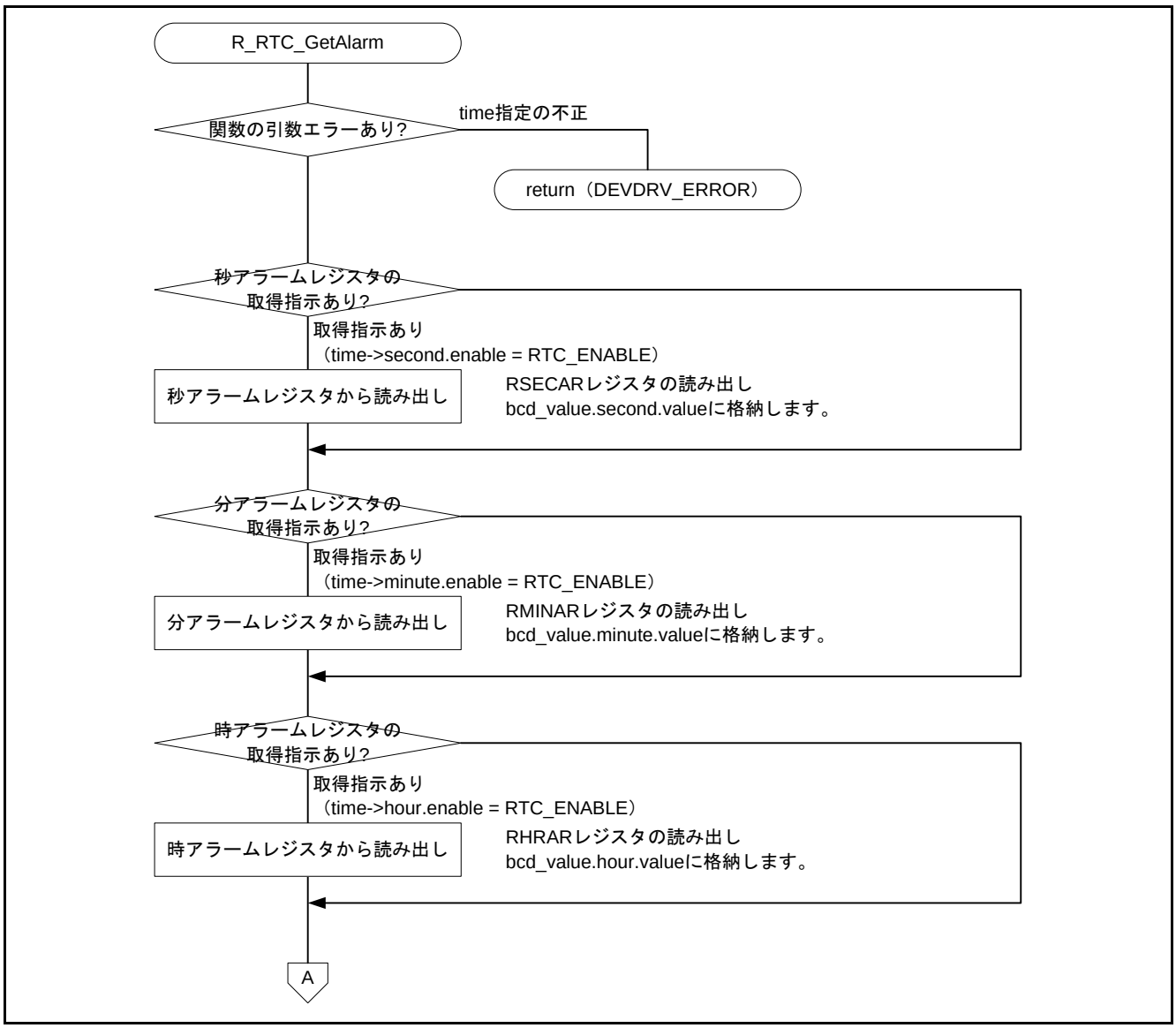


図6.35　RTC のアラームレジスタ値取得関数（1/3）

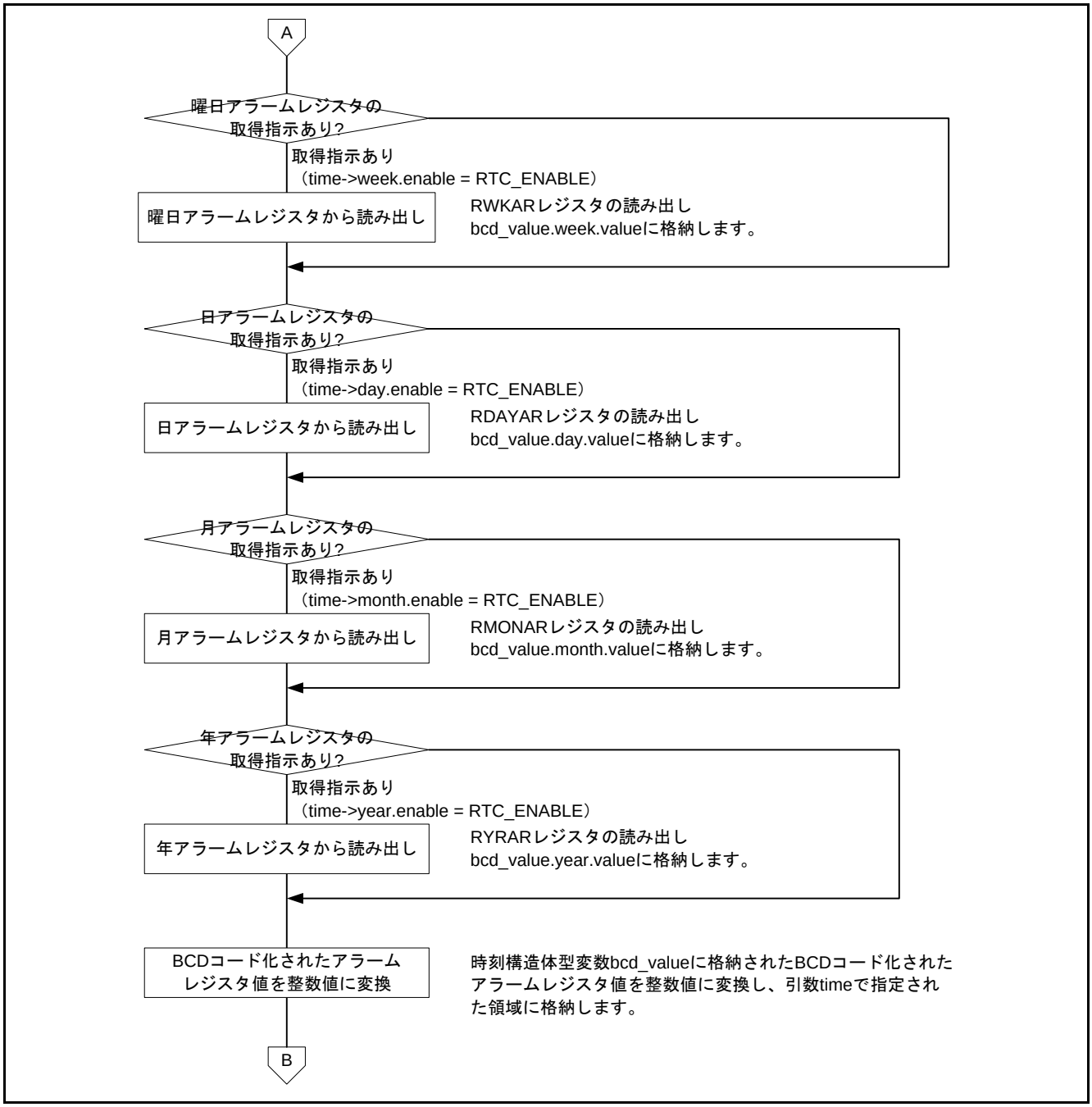


図6.36　RTC のアラームレジスタ値取得関数（2/3）

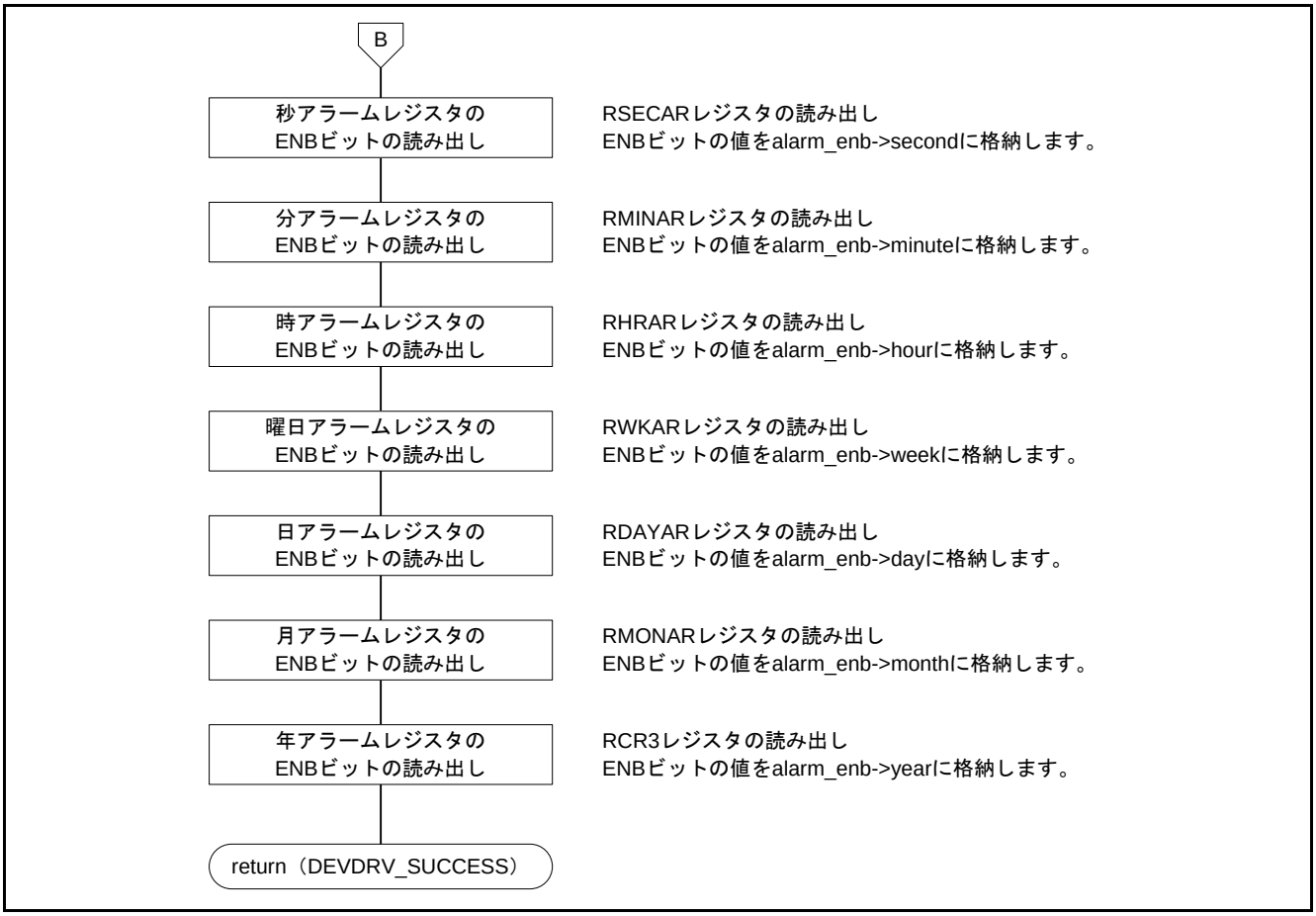


図6.37　RTC のアラームレジスタ値取得関数（3/3）

6.8.21　RTC の初期設定関数

図 6.38にRTC の初期設定関数のフローチャートを示します。

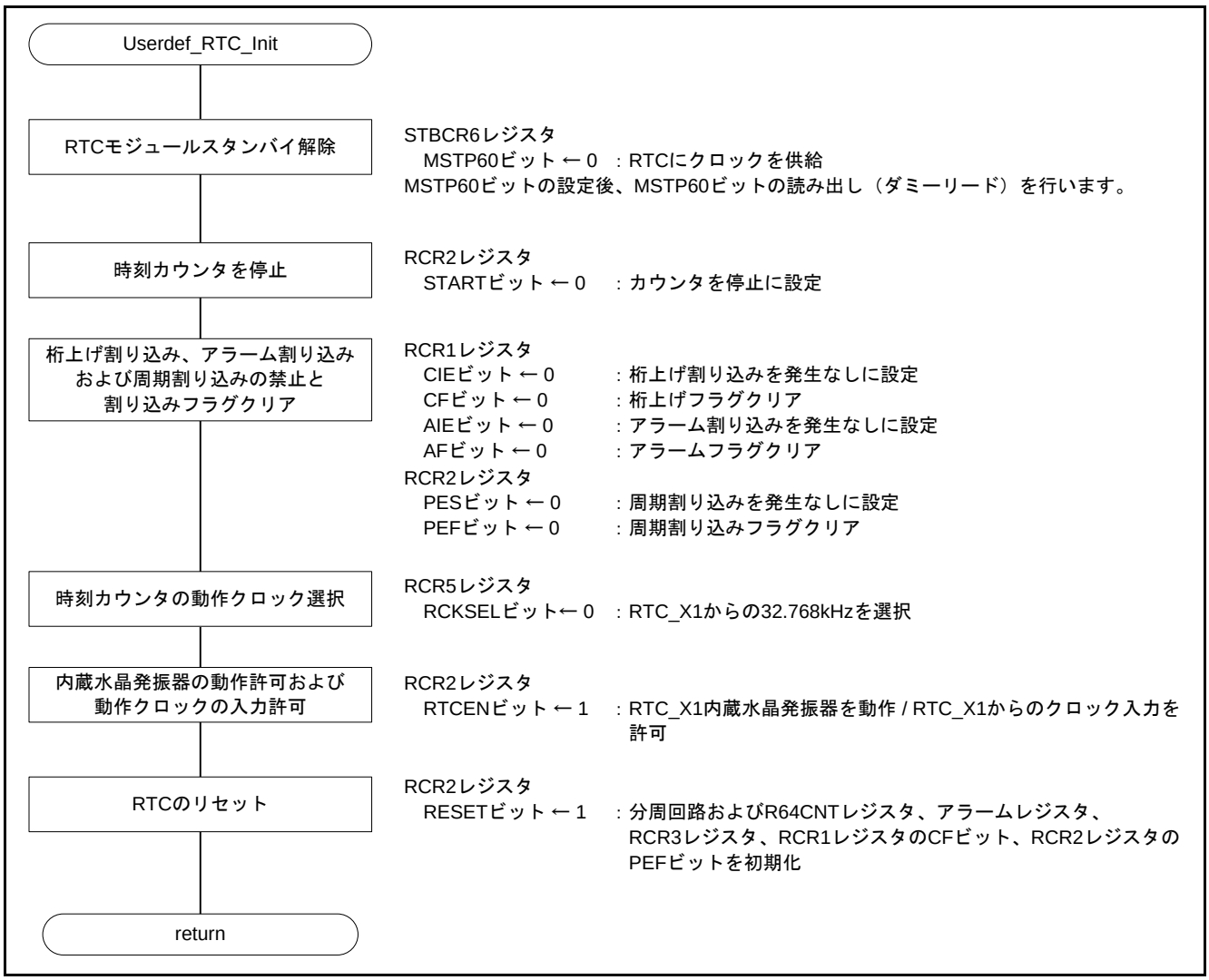


図6.38　RTC の初期設定関数

6.9　サンプルコードの起動

サンプルコードは、GENMAI ボードとシリアルインタフェースで接続したホスト PC 上のターミナルからコマンド入力を行うことで、動作します。

GENMAI ボードに電源を投入後、図 6.39の①に示すメッセージを出力します。RTC のサンプルコードを起動させる場合は、"SAMPLE>"のプロンプトの後に、"RTC"＋"Enter"キーを入力します。図 6.39の②に示すメッセージを出力します。"RTC SAMPLE>"のプロンプトの後に"1"～"6"キー＋"Enter"キーを入力することで、RTC サンプルコードが起動します。

"HELP"＋"Enter"キーを入力することで、図 6.39の③に示すようなサンプルコードの情報を表示します。"EXIT"＋"Enter"キーを入力することで、RTC サンプルコードの動作を終了します。

図 6.39の Ver.X.XX はサンプルコードのメイン処理のバージョンを、Ver.Y.YY は RTC サンプルコードのバージョンを示します。

表示メッセージ

```
RZ/A1H CPU Board Sample Program. Ver.X.XX                                  ①
Copyright (C) 2014 Renesas Electronics Corporation. All rights reserved.

select sample program.

SAMPLE>
```

```
RZ/A1H Realtime Clock(RTC) Sample Program. Ver.Y.YY                        ②
Copyright (C) 2014 Renesas Electronics Corporation. All rights reserved.

select sample program.

RTC SAMPLE>
```

```
RTC SAMPLE> help                                                           ③

  1    : Initialize RTC
  2    : Set time
  3    : Get time
  4    : Start RTC
  5    : Stop RTC
  6    : Transition to Deep Standby Mode
       :  -> Canceled by RTC alarm interrupt
  EXIT : Exit RTC sample

RTC SAMPLE>
```

図6.39　RTC サンプルコード起動時のターミナル表示

7. 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。

8. 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/A1H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。）

R7S72100 RTK772100BC00000BR（GENMAI）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 Generic Interrupt Controller Architecture Specification Architecture version 1.0
（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：開発環境

ARM ソフトウェア開発ツール（ARM Compiler toolchain、ARM DS-5 等）に関しては、ARM ホームページから入手してください。
（最新版を ARM ホームページから入手してください。）

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

Rev.	発行日	改訂内容	
		ページ	ポイント
Rev.0.81	2014.09.05	ー	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　〒100-0004　千代田区大手町2-6-2（日本ビル）

■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/