アプリケーションノート

SAM

目次

1. はじめに

本書は、セキュアアクセスメモリー(SAM)を用いるサンプルソフト SAM について記載しています。製品を開発する際、動作確認用または、プログラム開発の参考としてご利用願います。

2. 用語

用語／略語	定義
CG	Clock Control and Operation Mode
SAM	Secure Access Memory
UART	Universal Asynchronous Receiver Transmitter

3. 関連するドキュメント

ドキュメント	備考
データシート	利用する MCU のデータシートを参照してください
リファレンスマニュアル	利用する各 IP のリファレンスマニュアルを参照してください
アプリケーションノート MCU 利用説明書	利用する MCU 利用説明書を参照してください

4. 対象サンプルプログラム

サンプルプログラム	概要
SAM	セキュアアクセスメモリー機能のサンプルプログラム

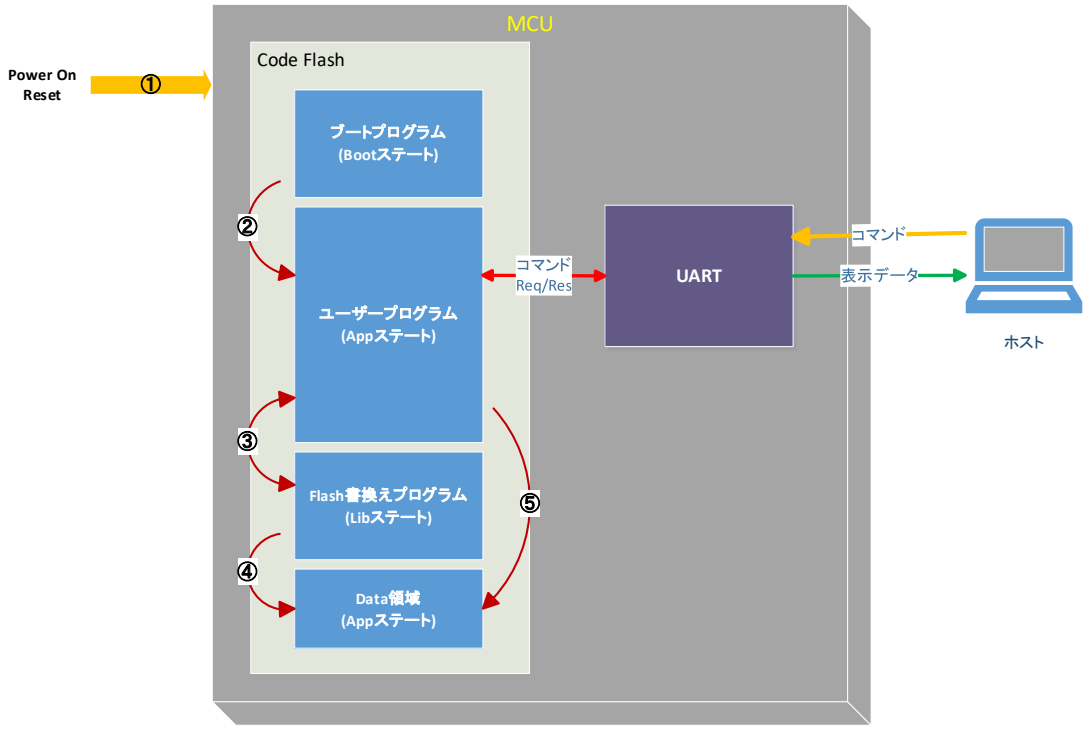
5. サンプルプログラム：SAM

SAM 機能を用いて各ステートの領域設定を行う例です。領域設定後はターミナルソフトから入力されたコマンドに従い、Code Flash の書き換え/消去/読み出しを行います。

5.1. 動作・操作概要

1. Power On Reset 後に Boot ステートに遷移し、Boot プログラムを実行します。この際、Push-Switch(USW_1)を押下しながら Power On Reset を行うと SAM 機能が有効になります。Boot プログラムでは Lib ステートと App ステートの設定を行った後に App ステートに遷移します。

2. App ステートのプログラムは、UART を用いてターミナルソフトと通信を行います。ターミナルソフトから入力された Command に従い、下記の処理を行います。
   - write コマンドを受信： Lib ステートに遷移し、コマンドで指定された文字列を Code Flash に書き込みます。書き込み完了後はユーザープログラムに戻り、App ステートに遷移します。
   - erase コマンドを受信： Lib ステートに遷移した後、Code Flash の消去を行います。消去完了後はユーザープログラムに戻り、App ステートに遷移します。
   - read コマンドを受信： Lib ステートに遷移した後、Code Flash に保存した文字列を読み出してターミナルソフトに表示します。表示完了後は App ステートに遷移します。



各コマンドの詳細は下記のとおりです。

コマンド	フォーマット	概要
write	write 文字列	コマンドで指定した文字列を Code Flash 書き込みます
read	read	直前に書き込んだ文字を Code Flash より読み出します
erase	erase	Code Flash を消去します

5.2. 使用する機能

使用する機能は下記のとおりです。

IP	目的
PORT (Push-Switch)	SAM 有効/無効設定用
SAM	メモリーアクセスにおけるセキュリティー機能
UART	ターミナルソフト通信用

5.3. 使用する割り込み

MCU 利用説明書“サンプル毎の割り込み一覧”を参照願います。

5.4. コンフィグレーション

コンフィグレーション設定

コンフィグレーション	ソフト定義名	設定値 (デフォルト)	説明
Write DATA MAX.	CFG_DATA_LENGTH	10	書き込み可能文字数 (単位: 文字)

5.5. ターミナルソフト出力例

5.5.1. 正常時

```
Secure Enable (*)
command > write abc

write data > abc

command > read

read data > abc
command > erase

erase
command >
```

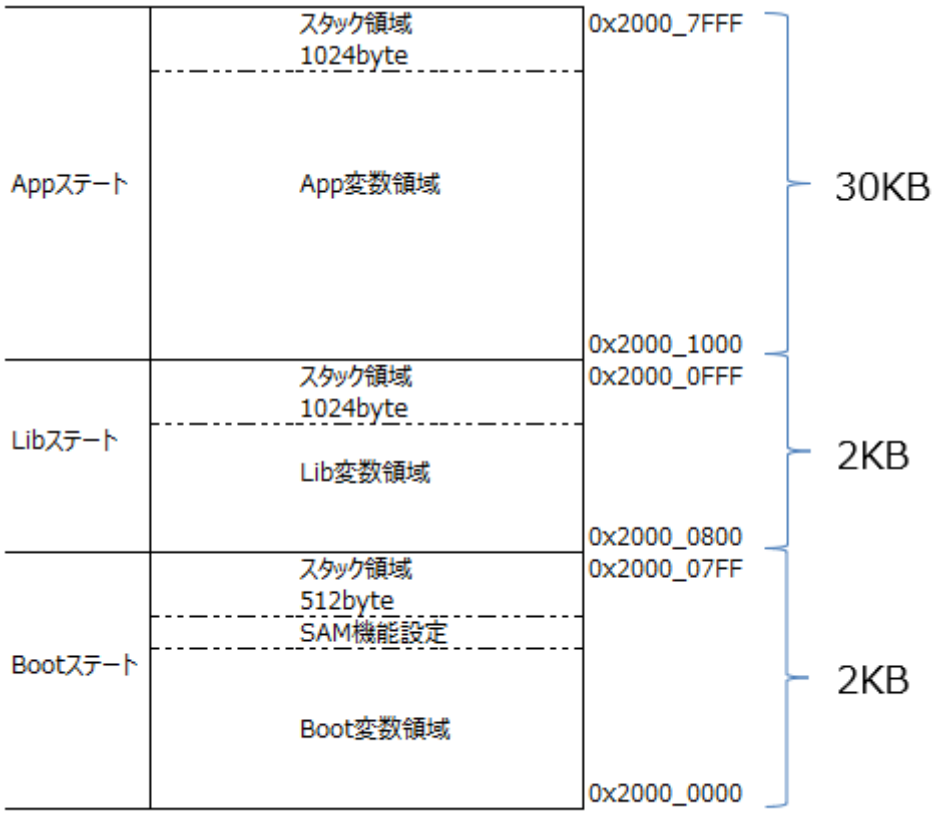
＊ SAM 設定が無効(USW1 を押下せずに Power On Reset 実施)の場合、”Secure Disable”が表示されます

5.5.2. エラー発生時

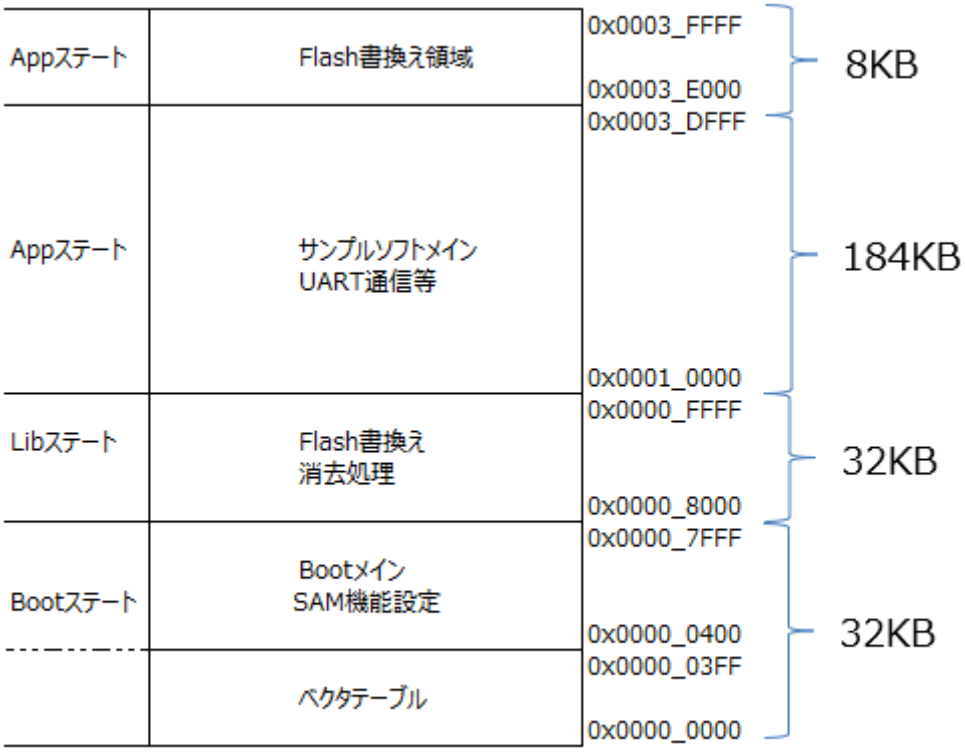
```
command > A

 Command Error !!
command > write 0123456789A

  Parameter Error !!
command > erase

erase
command > read

FFFB error!!
command >
```

6. メモリーマップ

各ステートの Code Flash および RAM 領域のメモリーマップは下記のとおりです。



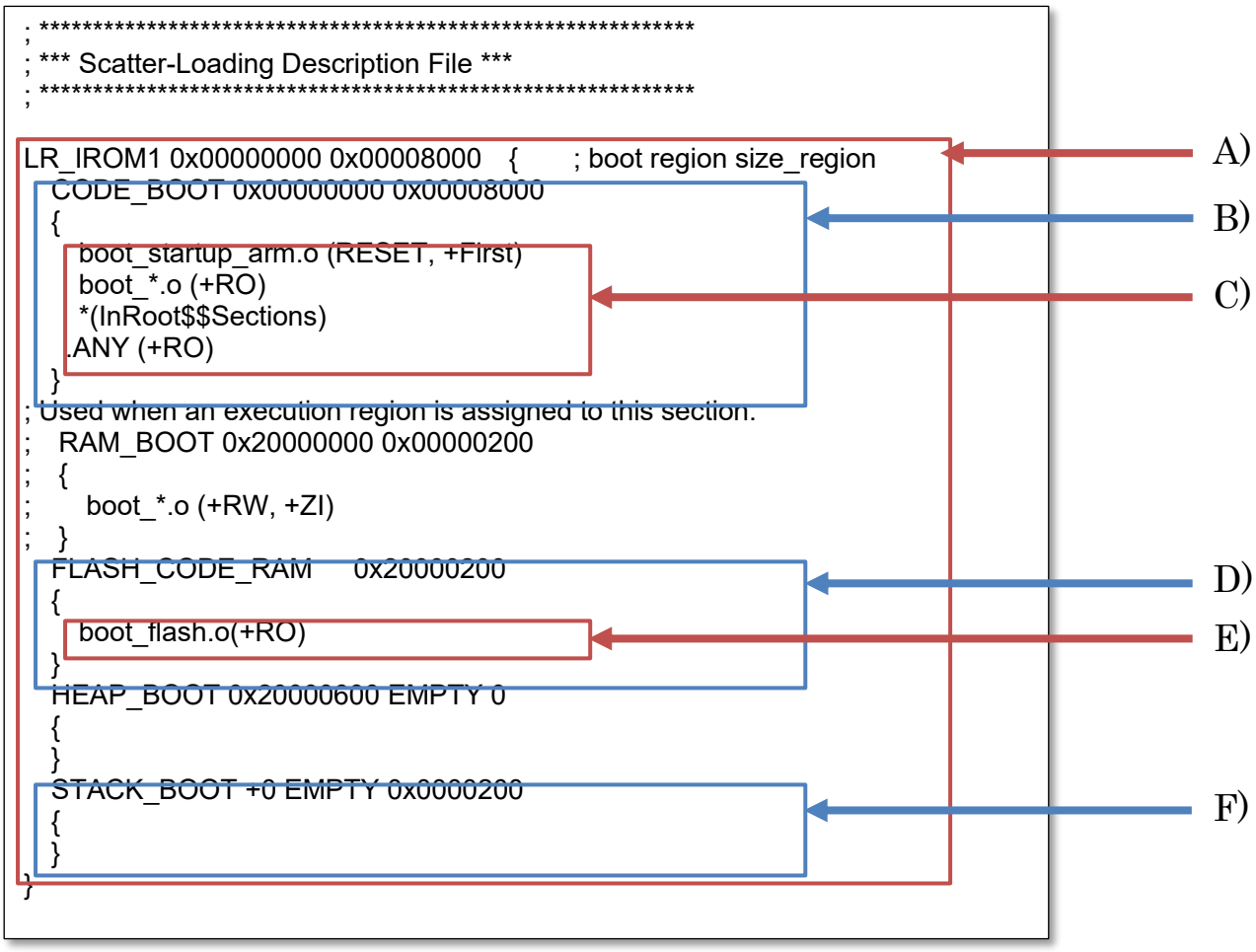
RAM



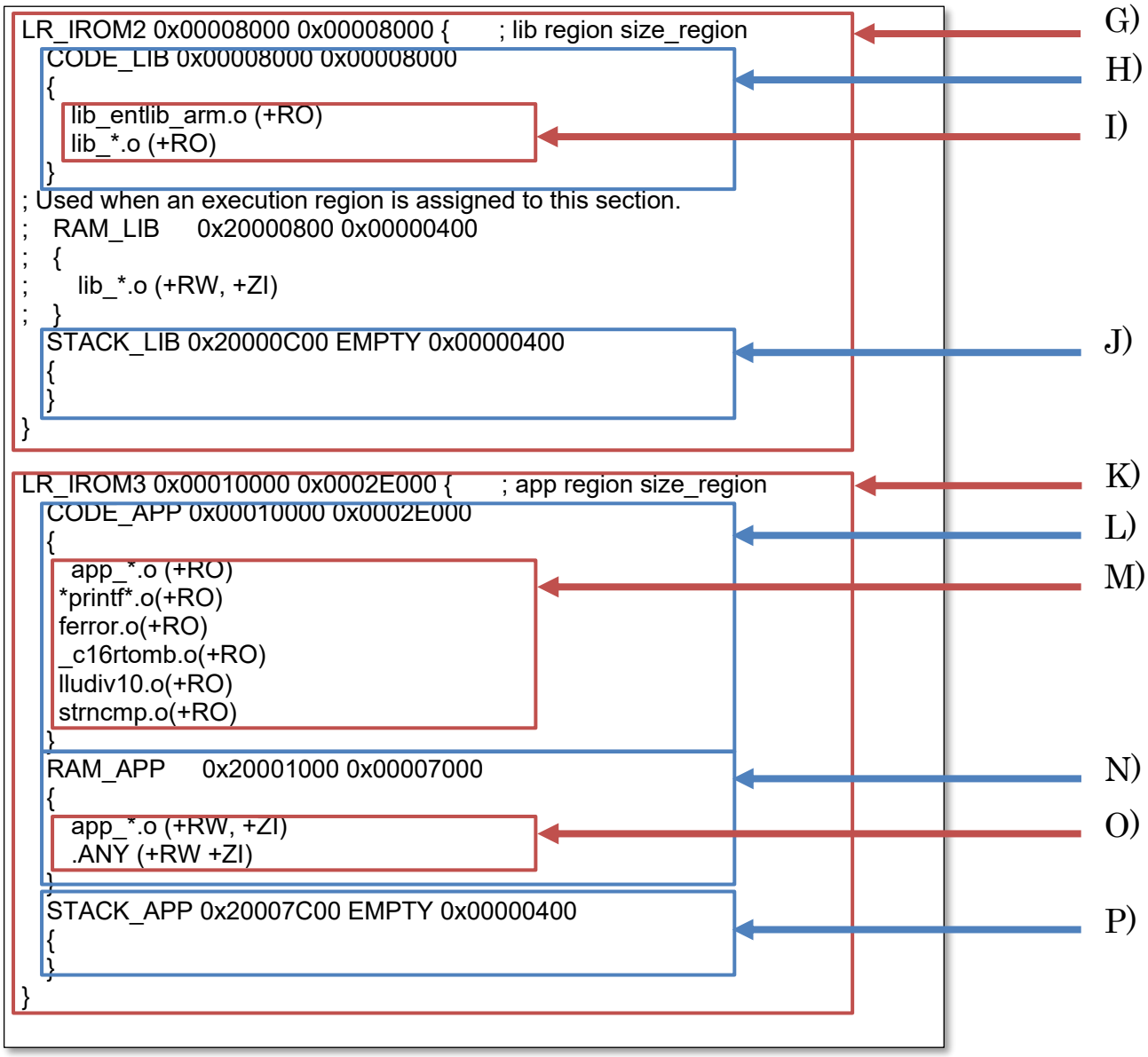
Code Flash

7. リンカスクリプト

7.1. Keil MDK スキャッターファイル(.sct)



A) Boot ステート ロード領域： アドレス 0x0000_0000 - 0x0000_7FFF
B) Boot Code 実行領域： アドレス 0x0000_0000 - 0x0000_7FFF
C) 入力セクション。Boot ステートのプログラムを配置
D) Flash Code 実行領域： アドレス 0x2000_0000 - 0x2000_01FF
E) 入力セクション。SAM 機能設定(SAM 機能 有効・無効)プログラムを配置
F) スタック 実行領域： アドレス 0x2000_0600 - 0x2000_07FF



G) Lib ステート ロード領域： アドレス 0x0000_8000 - 0x0000_FFFF
H) Lib Code 実行領域： アドレス 0x0000_8000 - 0x0000_FFFF
I) 入力セクション。Lib ステートのプログラムを配置。Lib ステートの遷移アドレスを 0x8000 とするため Lib ステートエントリー処理(lib_entlib_arm.o)を先頭に配置
J) スタック 実行領域： アドレス 0x2000_0C00 - 0x2000_0FFF
K) App ステート ロード領域： アドレス 0x0001_0000 - 0x0001_DFFF
L) App Code 実行領域： アドレス 0x0001_0000 - 0x0001_DFFF
M) 入力セクション。App ステートのプログラムを配置。App ステートで使用する標準関数(printf, strncmp など)を配置(詳細は 8.2 節参照)
N) App RAM 実行領域： アドレス 0x2000_1000 - 0x0000_6FFF
O) 入力セクション。App ステートの変数を配置
P) スタック 実行領域： アドレス 0x2000_7C00 - 0x2000_7FFF

7.2. IAR EWARM リンカ設定ファイル(.icf)

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$¥config¥ide¥IcfEditor¥cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x00000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_LR_IROM1_start__     = 0x00000000;
define symbol __ICFEDIT_region_LR_IROM1_end__       = 0x00007FFF;
define symbol __ICFEDIT_region_LR_IROM2_start__     = 0x00008000;
define symbol __ICFEDIT_region_LR_IROM2_end__       = 0x0000FFFF;
define symbol __ICFEDIT_region_LR_IROM3_start__     = 0x00010000;
define symbol __ICFEDIT_region_LR_IROM3_end__       = 0x0003DFFF;
define symbol __ICFEDIT_region_LR_IRAM1_start__     = 0x20000000;
define symbol __ICFEDIT_region_LR_IRAM1_end__       = 0x200007FF;
define symbol __ICFEDIT_region_LR_IRAM2_start__     = 0x20000800;
define symbol __ICFEDIT_region_LR_IRAM2_end__       = 0x20000FFF;
define symbol __ICFEDIT_region_LR_IRAM3_start__     = 0x20001000;
define symbol __ICFEDIT_region_LR_IRAM3_end__       = 0x20007FFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_stack_boot__     = 0x200;
define symbol __ICFEDIT_size_heap_boot__      = 0x200;
define symbol __ICFEDIT_size_stack_lib__      = 0x400;
define symbol __ICFEDIT_size_heap_lib__       = 0x000;
define symbol __ICFEDIT_size_stack_app__      = 0x400;
define symbol __ICFEDIT_size_heap_app__       = 0x000;
/**** End of ICF editor section. ###ICF###*/
```

A)
B)
C)

A) Code 配置アドレスのマクロ定義
Boot ステート： アドレス 0x0000_0000 - 0x0000_7FFF
Lib ステート： アドレス 0x0000_8000 - 0x0000_FFFF
App ステート： アドレス 0x0001_0000 - 0x0003_DFFF

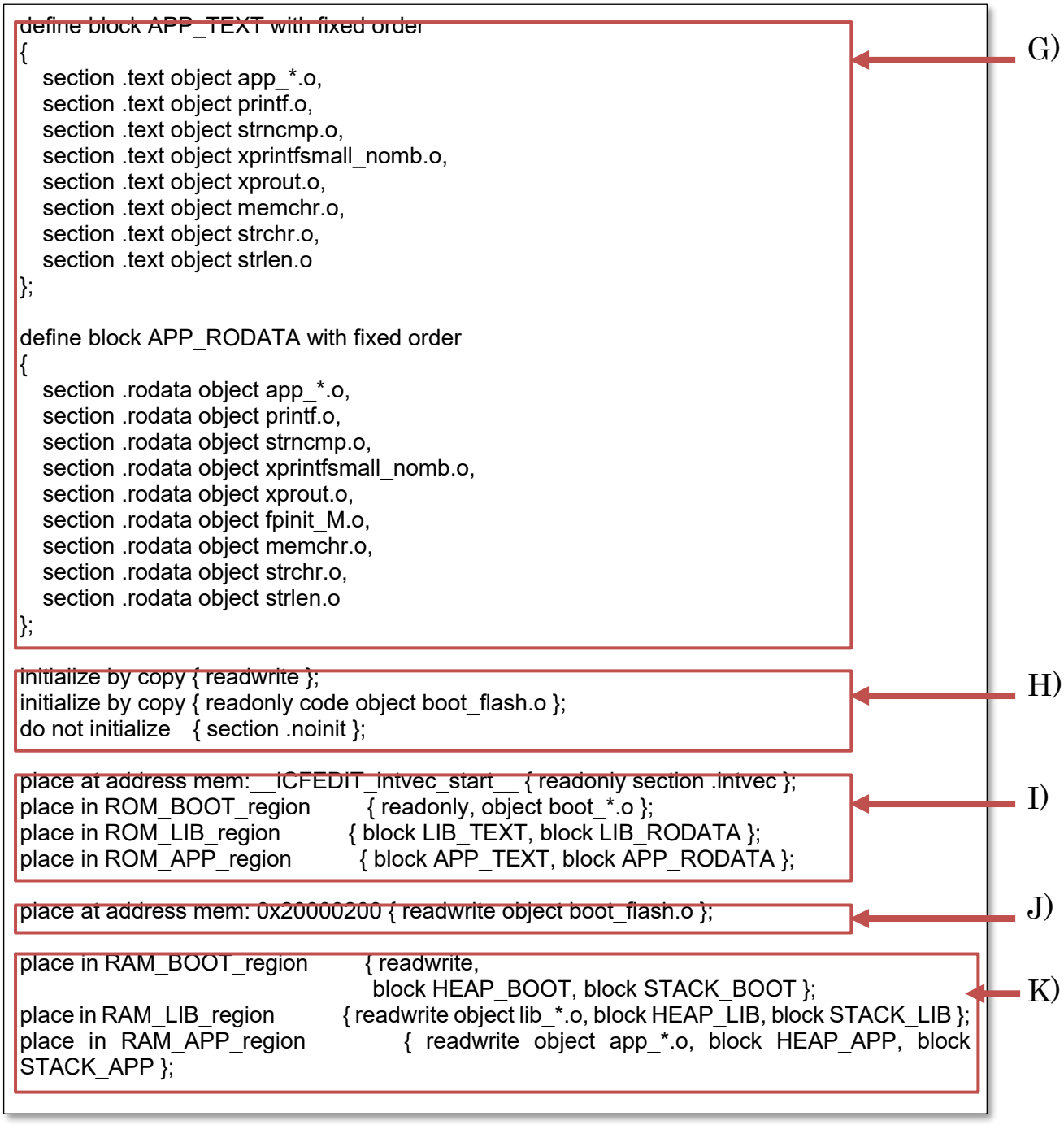
B) RAM 配置アドレスのマクロ定義
Boot ステート： アドレス 0x2000_0000 - 0x2000_07FF
Lib ステート： アドレス 0x2000_0800 - 0x2000_0FFF
App ステート： アドレス 0x2000_1000 - 0x2000_7FFF

C) スタック・ヒープサイズのマクロ定義
Boot ステート： スタック 512bytes, ヒープ 512bytes
Lib ステート： スタック 1024bytes
App ステート： スタック 1024bytes

```
define memory mem with size = 4G;
define region ROM_BOOT_region     = mem:[from __ICFEDIT_region_LR_IROM1_start__
to __ICFEDIT_region_LR_IROM1_end__];
define region ROM_LIB_region      = mem:[from __ICFEDIT_region_LR_IROM2_start__     to
__ICFEDIT_region_LR_IROM2_end__];
define region ROM_APP_region      = mem:[from __ICFEDIT_region_LR_IROM3_start__     to
__ICFEDIT_region_LR_IROM3_end__];
define region RAM_BOOT_region     = mem:[from __ICFEDIT_region_LR_IRAM1_start__
to __ICFEDIT_region_LR_IRAM1_end__];
define region RAM_LIB_region      = mem:[from __ICFEDIT_region_LR_IRAM2_start__     to
__ICFEDIT_region_LR_IRAM2_end__];
define region RAM_APP_region      = mem:[from __ICFEDIT_region_LR_IRAM3_start__     to
__ICFEDIT_region_LR_IRAM3_end__];

define block STACK_BOOT      with alignment = 8, size = __ICFEDIT_size_stack_boot__
{ };
define block HEAP_BOOT       with alignment = 8, size = __ICFEDIT_size_heap_boot__
{ };
define block STACK_LIB       with alignment = 8, size = __ICFEDIT_size_stack_lib__      { };
define block HEAP_LIB        with alignment = 8, size = __ICFEDIT_size_heap_lib__       { };
define block STACK_APP       with alignment = 8, size = __ICFEDIT_size_stack_app__      { };
define block HEAP_APP        with alignment = 8, size = __ICFEDIT_size_heap_app__
{ };

define block LIB_TEXT with fixed order
{
   section .text object lib_entlib_iar.o,
   section .text object lib_main.o
};

define block LIB_RODATA with fixed order
{
   section .rodata object lib_entlib_iar.o,
   section .rodata object lib_main.o
};
```

D)
E)
F)

D) 各ステートの実行領域を指定
E) 各ステートのスタック・ヒープを 8bytes アライメントでメモリブロックに割り当て
F) Lib ステートのプログラムと Read-Only 属性をメモリブロックに割り当て。Lib ステートの遷移アドレスを 0x8000 とするため Lib ステートエントリー処理(lib_entlib_iar.o)を先頭に配置

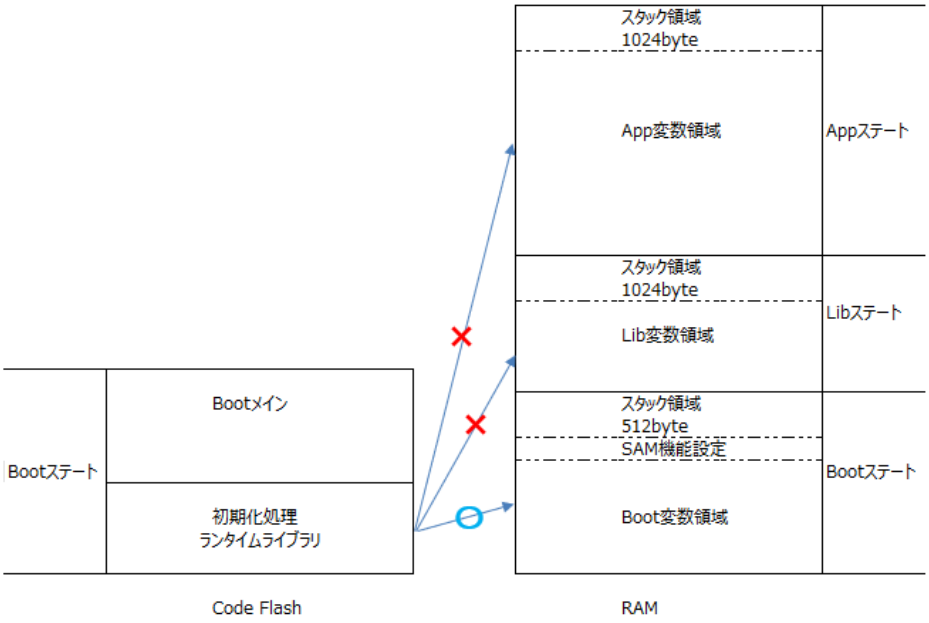


G) App ステートのプログラムと Read-Only 属性をメモリブロックに割り当て。App ステートで使用する標準関数(printf, strncmp など)を手動で配置(詳細は 8.2 節参照)
H) Boot ステートの SAM 機能設定(SAM 機能 有効・無効)プログラムを配置。ROM から RAM へ配置する。
I) 各ステートの ROM を配置する
J) Boot ステートの SAM 機能設定処理を 0x2000_0200 に配置する
K) 各ステートの RAM を配置する

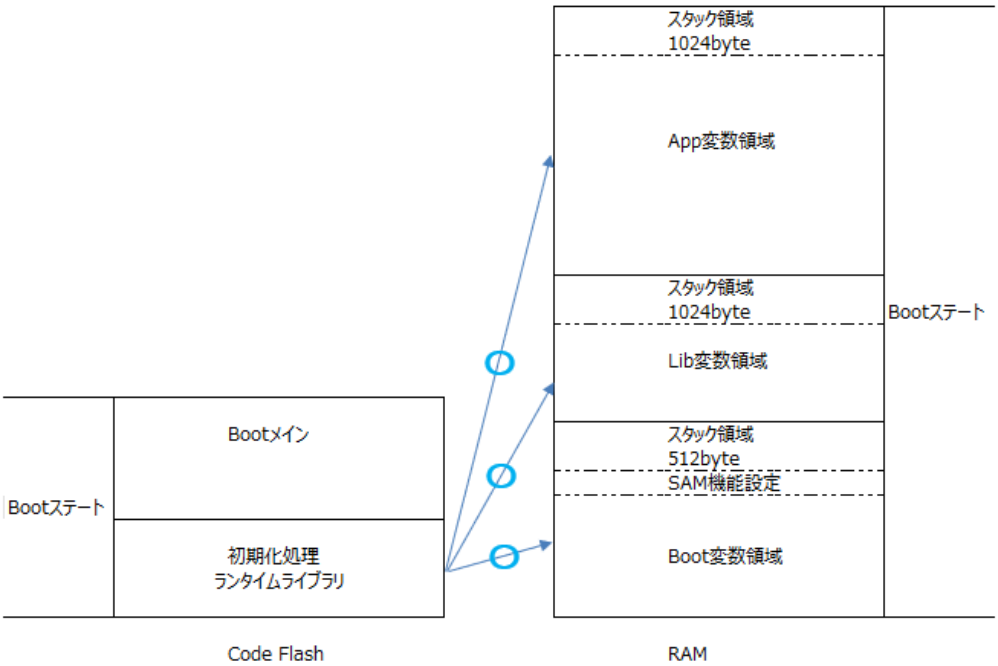
8. 実装上の注意点

8.1. 初期化処理

ランタイムライブラリーを使用して変数などの初期化処理を行うには、Reset 解除後の SAM 設定処理において RAM 領域すべてを Boot ステート領域(SBP)に設定する必要があります。初期化処理の前に各ステートの RAM 領域を設定した場合、初期化処理時に許可されていない領域にアクセスすることになるため SAM リセットを発生します。各ステートの RAM 領域を設定する場合は、初期化処理後の Boot ステートから App ステートに遷移する際に実施してください。



初めから所望の RAM 領域設定を行うとアクセス違反でリセットが発生する



初期化処理後、Boot ステートから App ステートへ遷移するときに所望の RAM 領域を設定する

8.2. 標準関数の手動配置(リンカスクリプト)

printf()や strncmp()といった標準関数はリンカスクリプトで特に指定しない場合、Boot ステート領域に配置されます。そのため、App ステートのプログラムで呼び出す場合は手動で配置する必要があります。手動で配置する方法は、7 章を参照してください。
また標準関数から呼び出される関数オブジェクト(例えば printf.o の場合: ferror.o, _c16rtomb.o, lludiv10.o)も配置が必要です。この関数オブジェクトは map ファイルに記載されています。

8.3. SAM の有効・無効設定

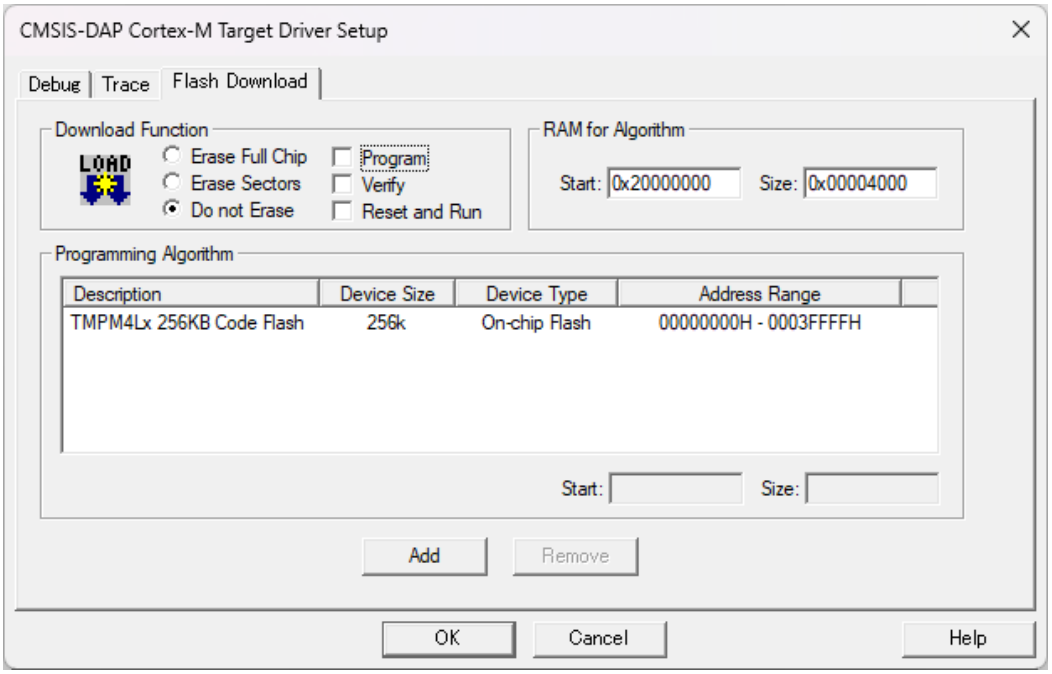
SAM 機能の有効・無効は、フラッシュメモリーの自動 SAMNRFP プログラムコマンドにて解除不能フラッシュプロテクト設定やロック設定など他の設定と同時に行います。
この際、ロック設定は必ず無効になるよう注意して設定する必要があります。
ロック設定が有効になると自動 SAMNRFP 解除コマンドが無効になり、SAM と解除不能フラッシュプロテクトを無効にすることができなくなります。すなわちフラッシュメモリー領域の書き込み・消去が永久に不可能となります。

9. デバッグ時の注意点

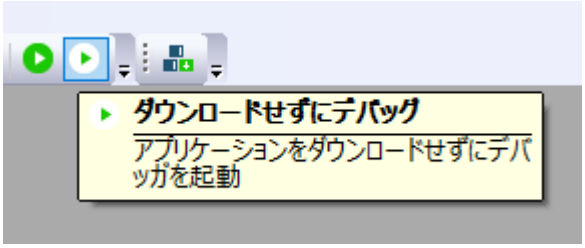
9.1. SAM 機能有効時にデバッガと接続できない

1. 原因
   リセット解除直後は Boot ステート、RAM は App ステート領域の設定になっています。開発ツール(デバッガ)は RAM に Flash Loader を書き込もうとしますが、Boot ステートでの App ステート領域 RAM へのアクセスとなるため、アクセス違反で SAM リセットが発生します。

2. 回避方法
   - Keil MDK の場合：
     “フラッシュ” → “フラッシュツールの設定” → “Setting” ボタン → “Flash Download” タブを選択します。
     “Download Function”項目より、(1)「Do not Erase」を選択、(2)「Program」のチェックをはずす、(3)「Verify」のチェックをはずす を行ってください。



   - IAR EWARM の場合： “ダウンロードせずにデバッグ”で接続する



SAM 機能有効時は、開発ツールが書き込む Flash Loader によってプログラムを書き換えることができません。プログラムを書き換えるためには、SAM 機能を無効化する必要があります。現状では SAM 機能を無効化するする方法は下記のとおりです。

1. “ダウンロードせずにデバッグ”でデバッガ接続する(IAR EWARM)
2. SAM 機能無効化設定処理の ROM→RAM 転送処理を実行
3. SAM 機能無効化設定処理にプログラムカウンターを移動して実行
   ※SAM 機能を有効にする前に、無効化処理が確実に動作できることが前提です

以上の方法で解決しない場合は、下記を実施してください。

1. 開発ツールのメモリーウィンドウで Boot ステートの RAM 領域に SAM 機能無効化処理のマシンコードを打ち込む
2. プログラムカウンターを移動して実行

9.2. SAM 機能有効時に突然リセットが発生する

- 現象
  App ステートのデバッグ後に Lib ステートへ遷移しブレークポイントで停止させた際、リセットが発生する。

- 原因
  App ステートのデバッグでメモリーウィンドウやウォッチウィンドウで App ステート領域の RAM を表示させていたことが原因です。Lib ステートへ遷移したとき、デバッガが App ステート領域の RAM をリードしたため、アクセス違反で SAM リセットが発生しました。

- 回避方法
  メモリーウィンドウやウォッチウィンドウを非表示にしてください。

9.3. SAM 機能有効時にリセットが繰り返し発生する

- 現象
  開発ツール(デバッガ)の停止機能でプログラムが停止できない。

- 原因
  プログラムの配置ミスでアクセス違反が発生していることが原因です。
  アクセス違反により SAM リセットが発生し、プログラムが再び実行開始を繰り返し、停止できなくなっている状態です。

- 回避方法
  スタートアップの Reset_Handler の先頭にブレークポイントを設定してください。
  アクセス違反でリセットが発生した際、この位置でブレークさせることでプログラムが再び実行することを防止できます。

9.4. SAM 機能が有効・無効にならない

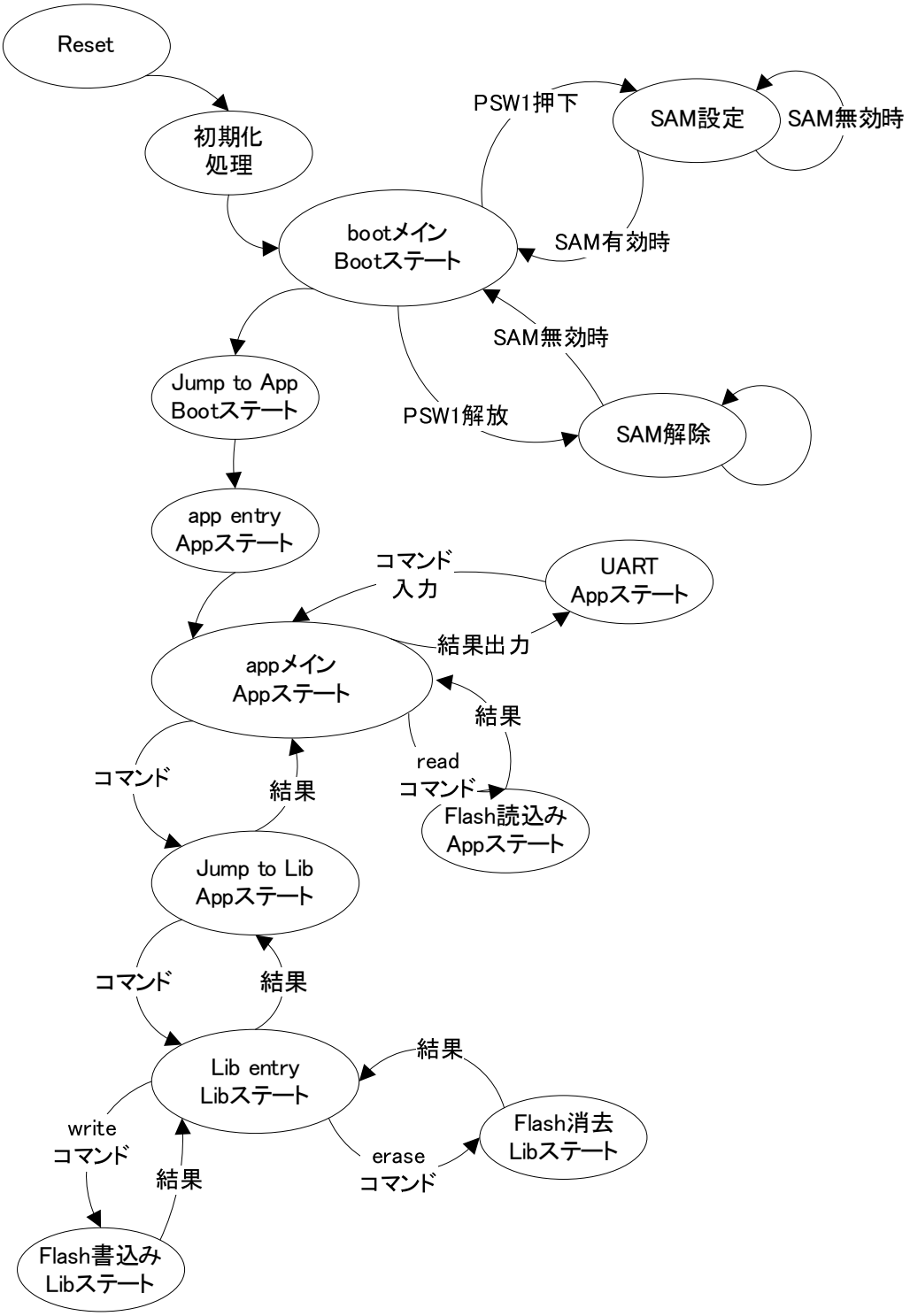
- 現象
  IAR EWARM でデバッグ時、SAM 機能の有効(または無効)設定処理を実行後にデバッガのリセット機能でリセットする。その後デバッガでプログラムを実行した際に SAM 機能が有効(無効)にならない。

  ※Keil MDK ではこの現象は発生しません

- 原因
  ハードウェアがリセットされていないことが原因です。例えば CMSIS DAP を使用している場合、IAR EWARM のオプション内、”CMSIS DAP”にあるリセット設定が”システム(デフォルト)”になっているとこの現象が発生します。

- 回避方法
  使用ボードの電源を入れなおす、リセットボタンを押すなどしてハードウェアをリセットしてください。

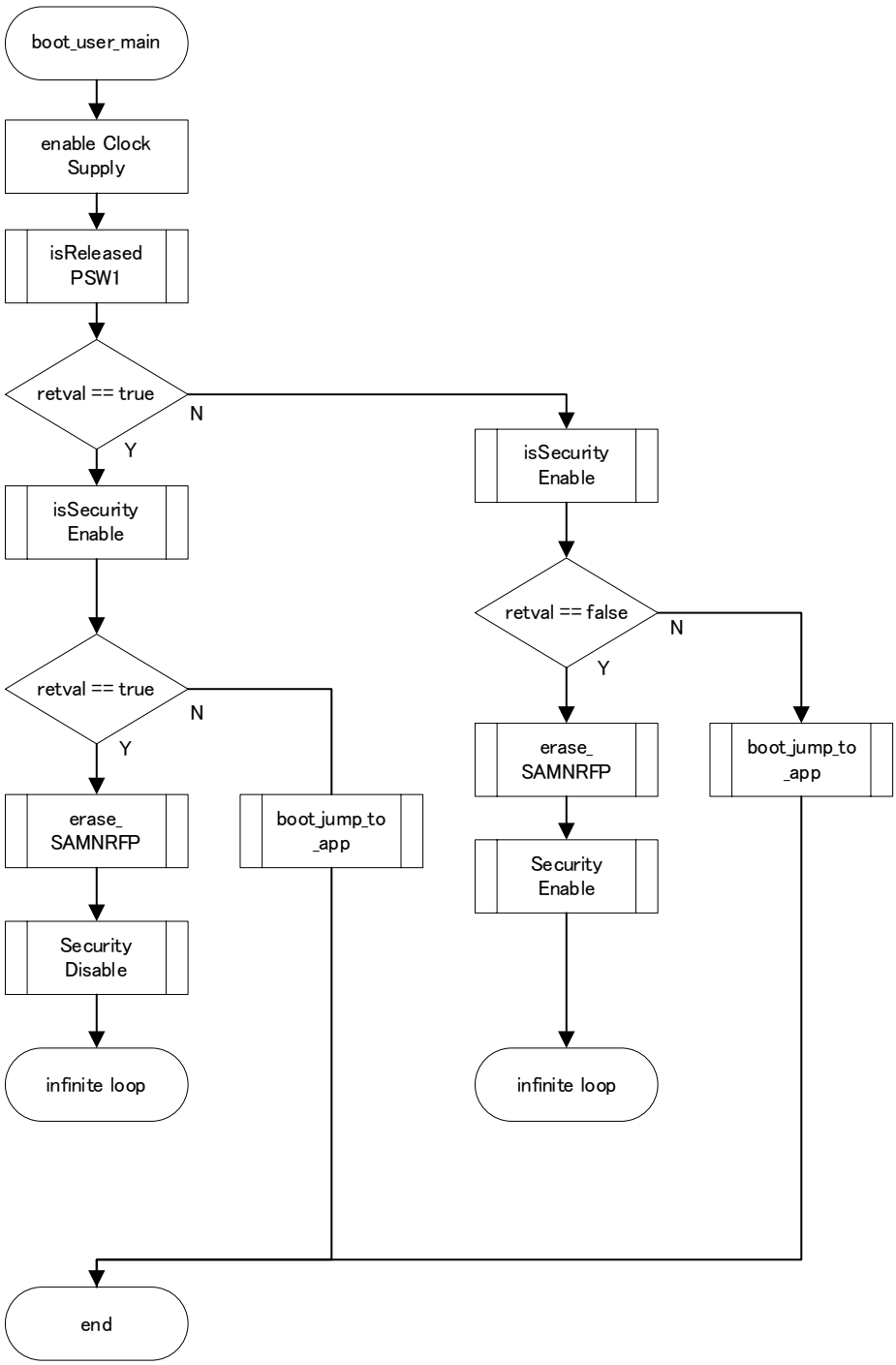
10. 状態遷移図


Reset
初期化
処理
PSW1押下
SAM設定
SAM無効時
bootメイン
Bootステート
SAM有効時
SAM無効時
Jump to App
Bootステート
PSW1解放
SAM解除
app entry
Appステート
コマンド
入力
UART
Appステート
結果出力
appメイン
Appステート
結果
read
コマンド
コマンド
結果
Flash読込み
Appステート
Jump to Lib
Appステート
コマンド
結果
結果
Lib entry
Libステート
write
コマンド
結果
erase
コマンド
Flash消去
Libステート
Flash書込み
Libステート

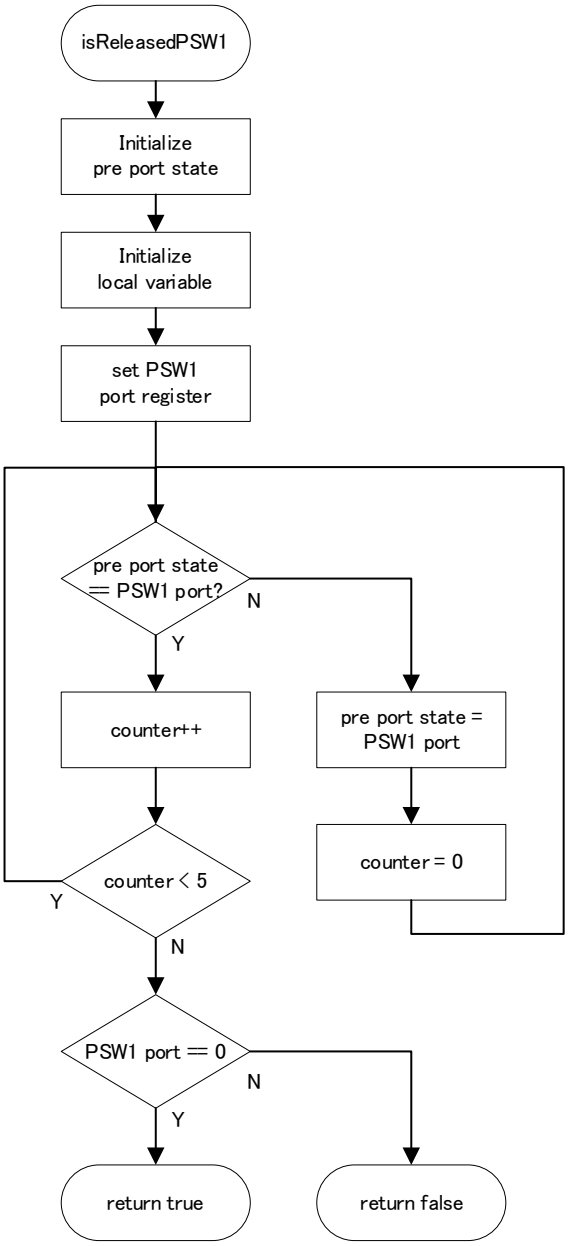
11. フローチャート

11.1. Boot ステート

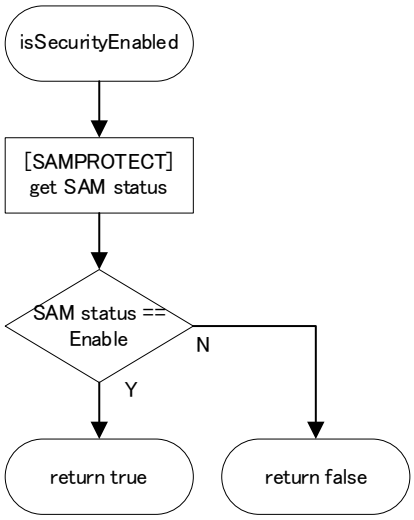
11.1.1. main (boot_user_main)



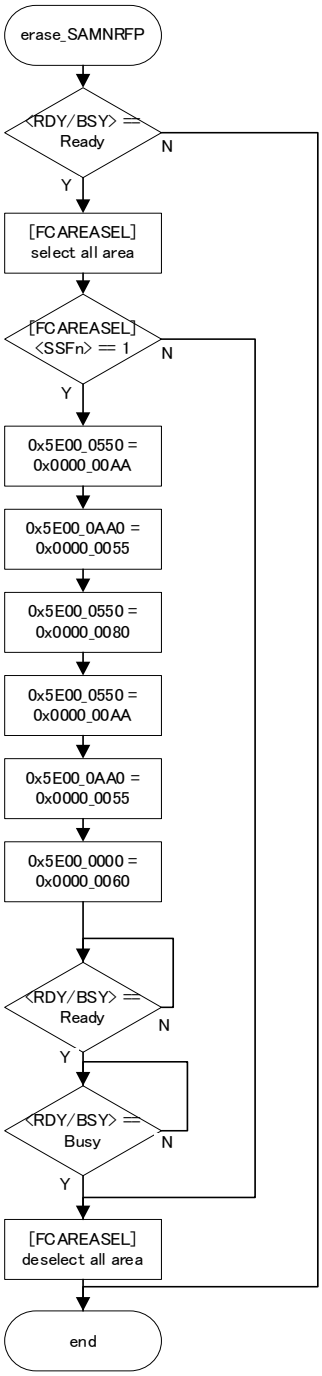
11.1.2. PSW1 押下/解放判定



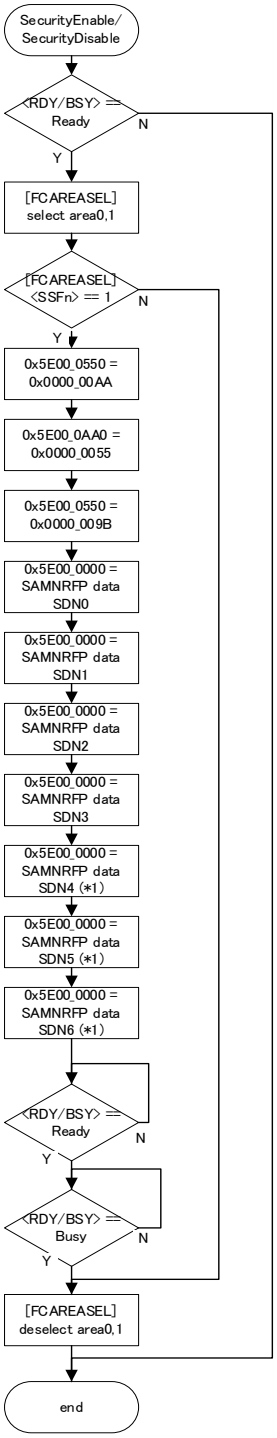
11.1.3. SAM のステータス取得



11.1.4. SAM・CBPEND 設定消去(自動 SAMNRFP 解除)

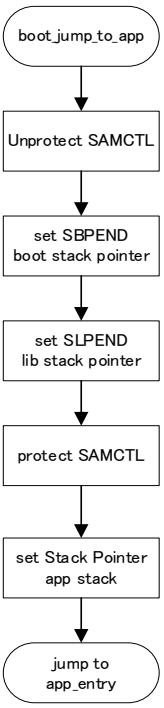


11.1.5. SAM 有効/無効設定



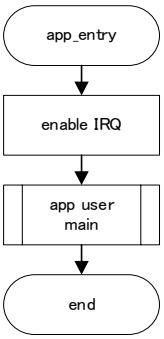
*1:
SecurityEnable() NRFPSET=1
SecurityDisable() NRFPSET=0

11.1.6. Boot ステートから App ステートへのエントリー

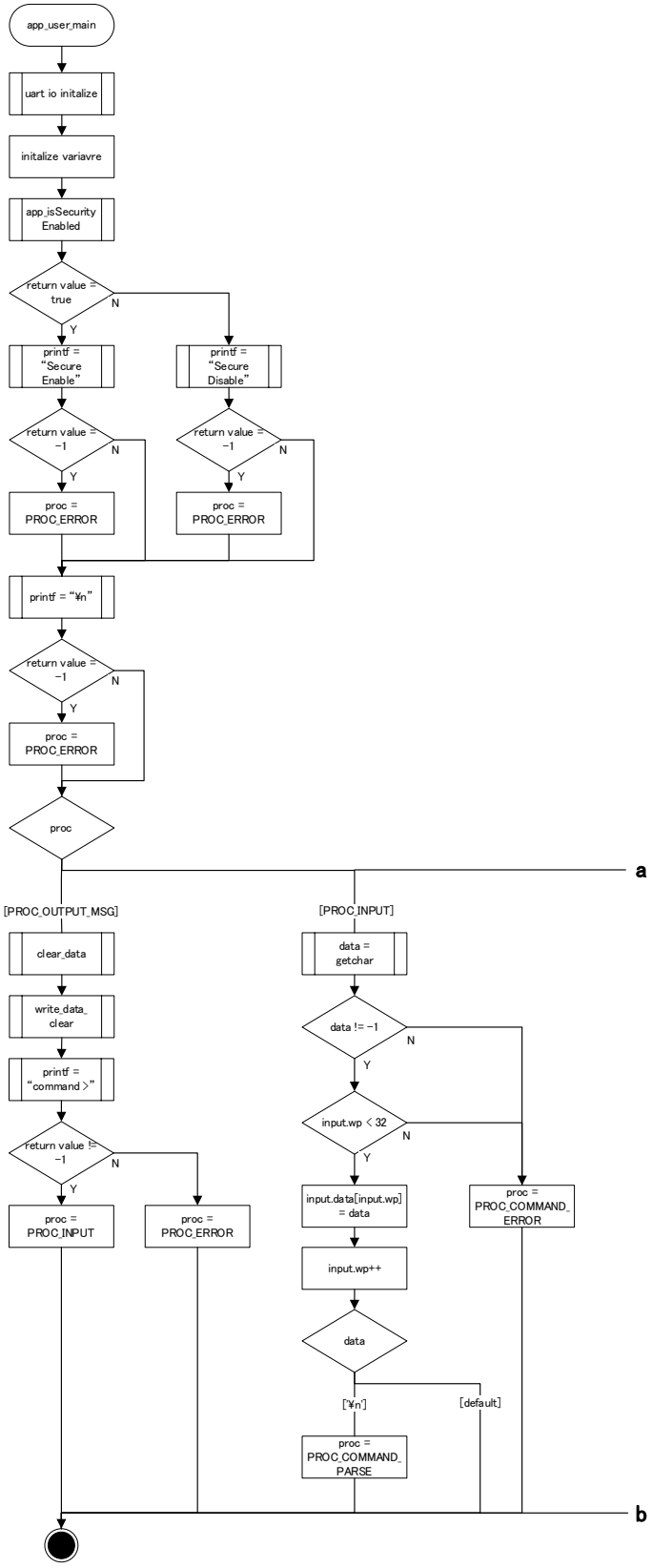


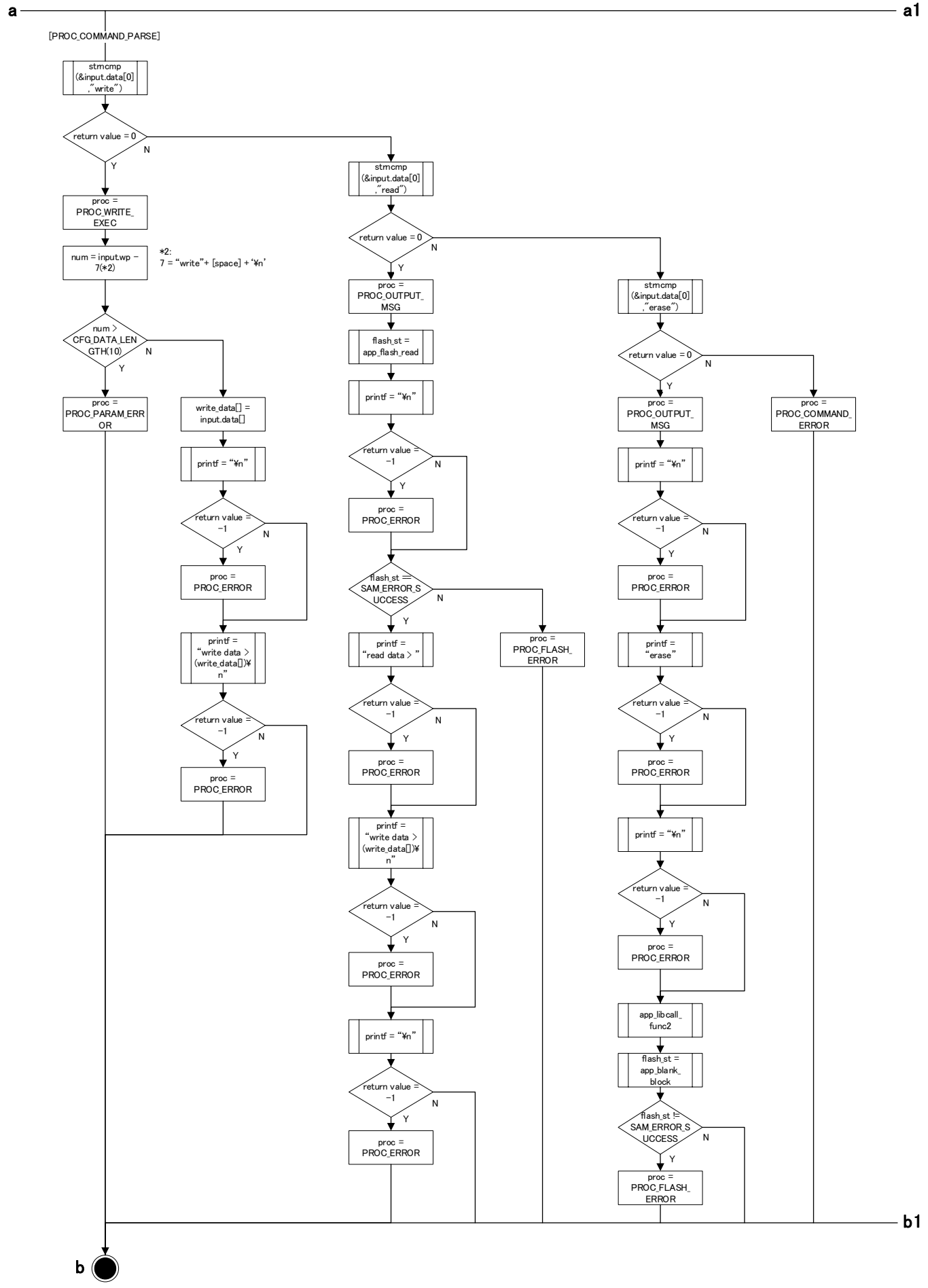
11.2. App ステート

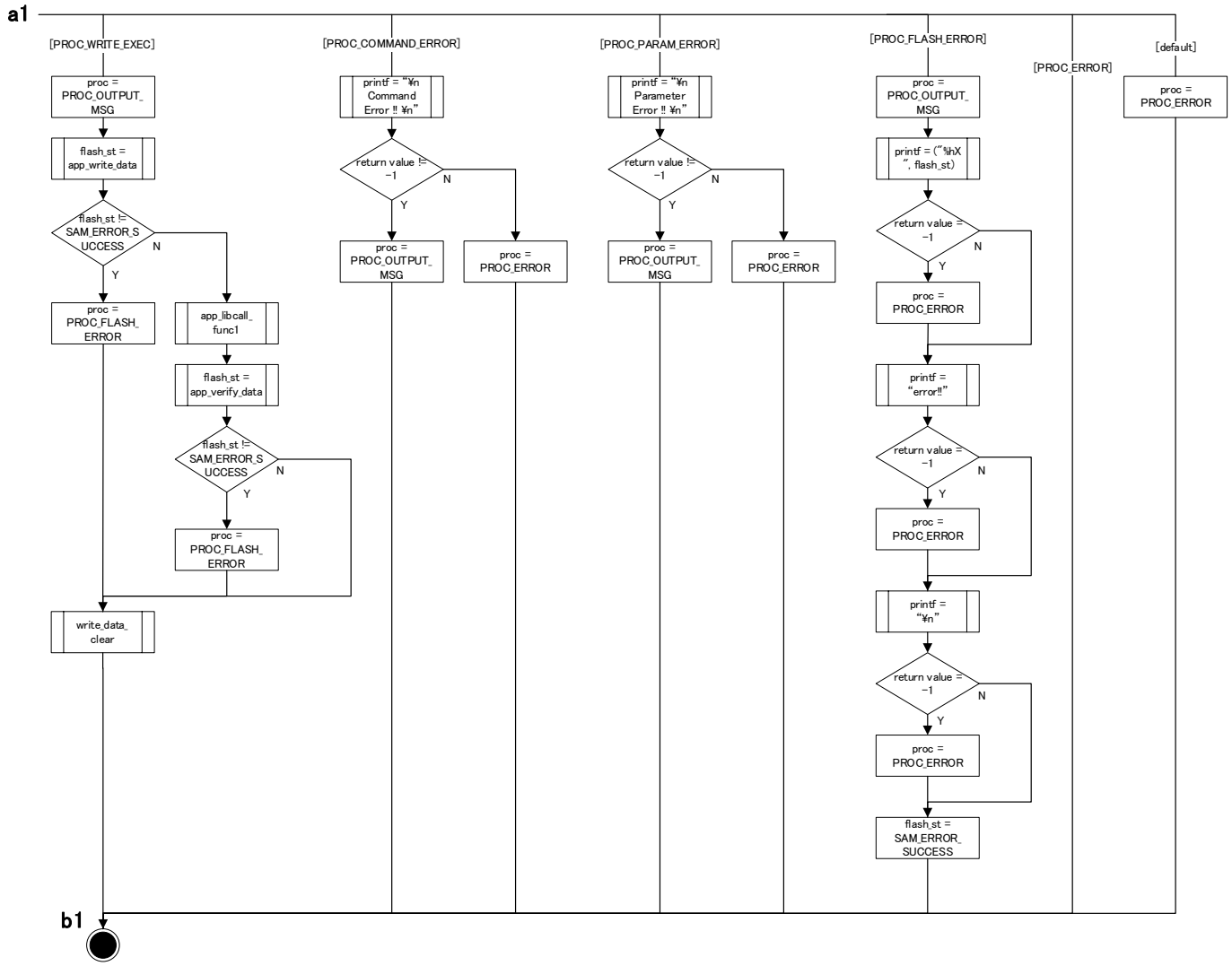
11.2.1. App ステートエントリー関数



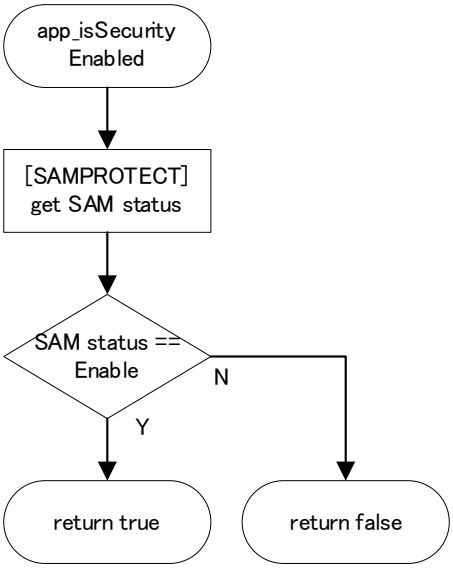
11.2.2. main (app_user_main)



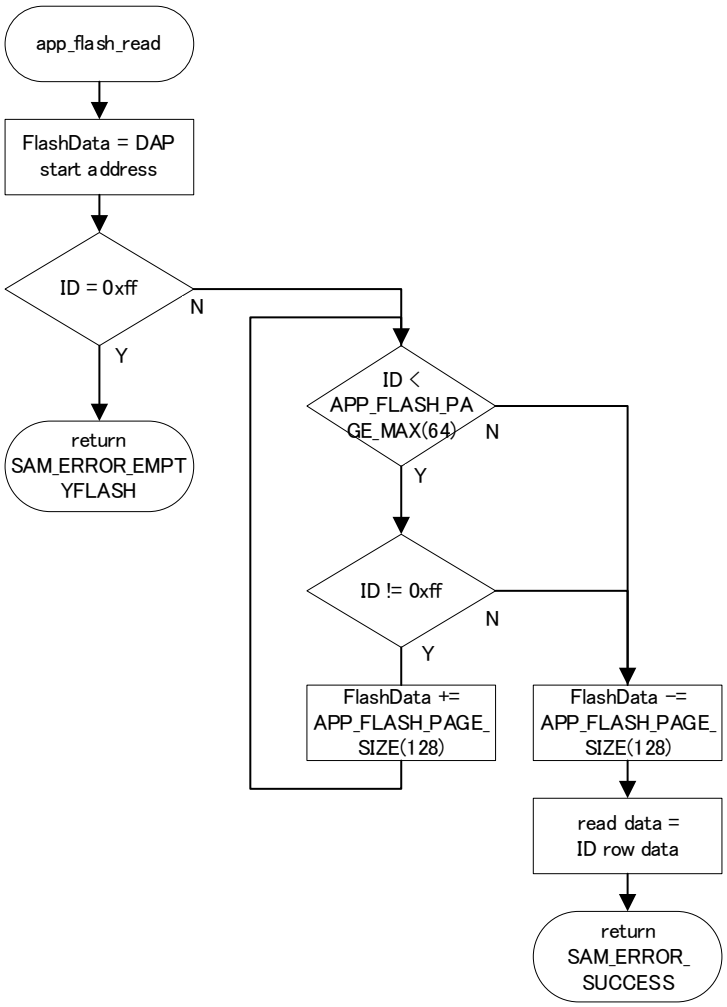

a
a1
[PROC_COMMAND_PARSE]
stmcmp (&input.data[0],"write")
return value = 0
N
Y
proc = PROC_WRITE_EXEC
num = input.wp − 7(*2)
*2: 7 = "write"+ [space] + '¥n'
num > CFG_DATA_LENGTH(10)
N
Y
proc = PROC_PARAM_ERROR
write_data[] = input.data[]
printf = "¥n"
return value = −1
N
Y
proc = PROC_ERROR
printf = "write data > (write_data[])¥n"
return value = −1
N
Y
proc = PROC_ERROR
stmcmp (&input.data[0],"read")
return value = 0
N
Y
proc = PROC_OUTPUT_MSG
flash_st = app_flash_read
printf = "¥n"
return value = −1
N
Y
proc = PROC_ERROR
flash_st == SAM_ERROR_SUCCESS
N
Y
proc = PROC_FLASH_ERROR
printf = "read data > "
return value = −1
N
Y
proc = PROC_ERROR
printf = "write data > (write_data[])¥n"
return value = −1
N
Y
proc = PROC_ERROR
printf = "¥n"
return value = −1
N
Y
proc = PROC_ERROR
stmcmp (&input.data[0],"erase")
return value = 0
N
Y
proc = PROC_OUTPUT_MSG
proc = PROC_COMMAND_ERROR
printf = "¥n"
return value = −1
N
Y
proc = PROC_ERROR
printf = "erase"
return value = −1
N
Y
proc = PROC_ERROR
printf = "¥n"
return value = −1
N
Y
proc = PROC_ERROR
app_libcall_func2
flash_st = app_blank_block
flash_st != SAM_ERROR_SUCCESS
N
Y
proc = PROC_FLASH_ERROR
b1
b


a1
[PROC_WRITE_EXEC]
proc = PROC_OUTPUT_MSG
flash_st = app_write_data
flash_st != SAM_ERROR_SUCCESS
N
Y
proc = PROC_FLASH_ERROR
app_libcall_func1
flash_st = app_verify_data
flash_st != SAM_ERROR_SUCCESS
N
Y
proc = PROC_FLASH_ERROR
write_data_clear
[PROC_COMMAND_ERROR]
printf = "¥n Command Error !! ¥n"
return value != -1
N
Y
proc = PROC_OUTPUT_MSG
proc = PROC_ERROR
[PROC_PARAM_ERROR]
printf = "¥n Parameter Error !! ¥n"
return value != -1
N
Y
proc = PROC_OUTPUT_MSG
proc = PROC_ERROR
[PROC_FLASH_ERROR]
proc = PROC_OUTPUT_MSG
printf = ("%hX", flash_st)
return value = -1
N
Y
proc = PROC_ERROR
printf = "error!!"
return value = -1
N
Y
proc = PROC_ERROR
printf = "¥n"
return value = -1
N
Y
proc = PROC_ERROR
flash_st = SAM_ERROR_SUCCESS
[PROC_ERROR]
[default]
proc = PROC_ERROR
b1

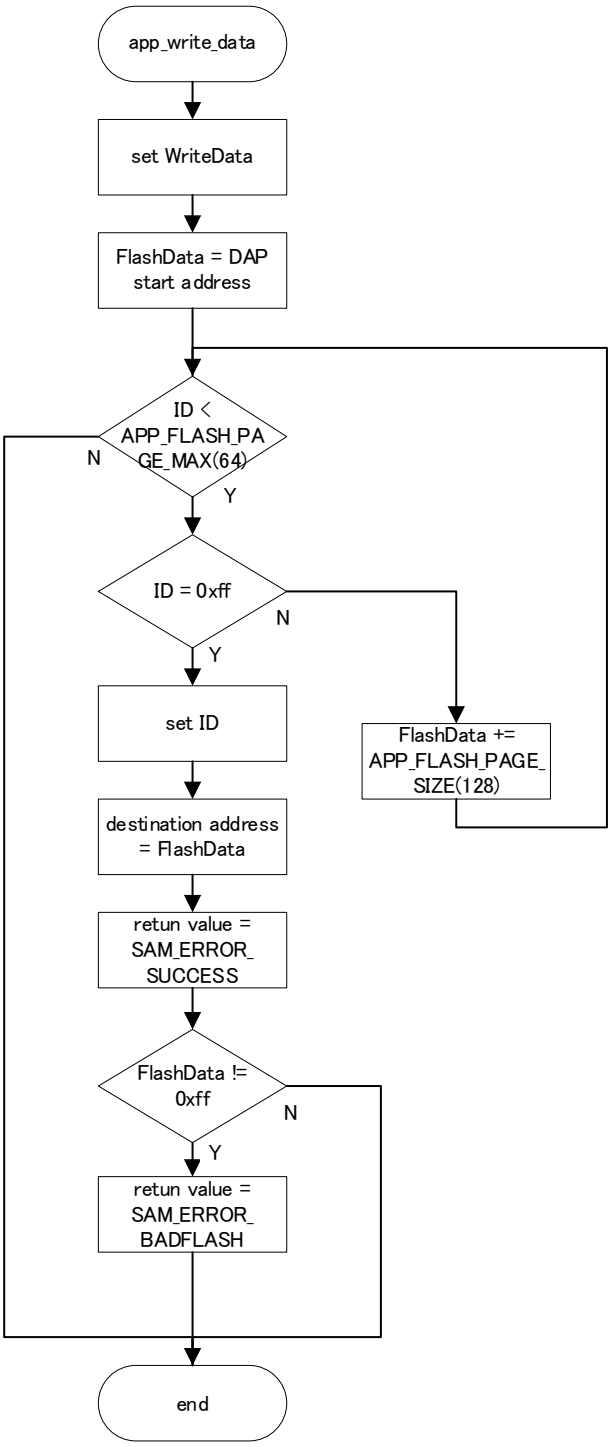
11.2.3. SAM のステータス取得



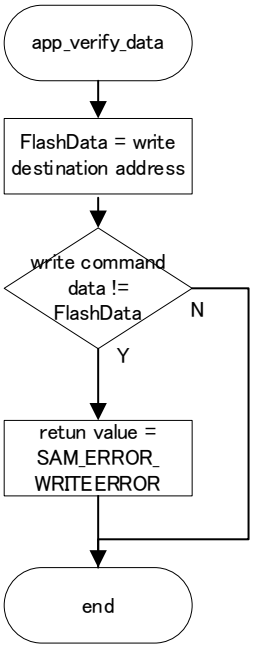
11.2.4. Flash データの読み出し



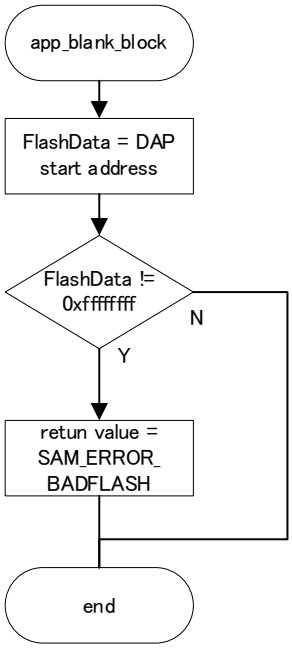
11.2.5. Flash データ書き込み



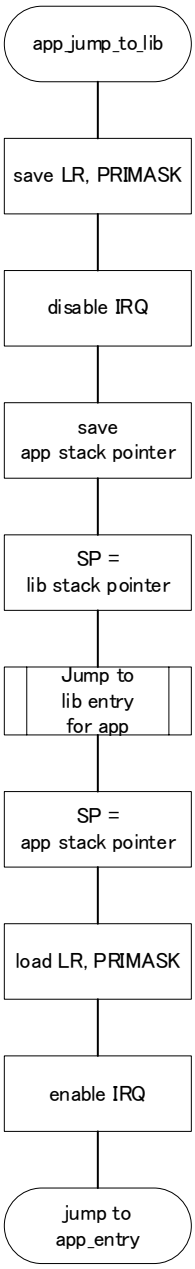
11.2.6. Flash 書き込みデータの検証



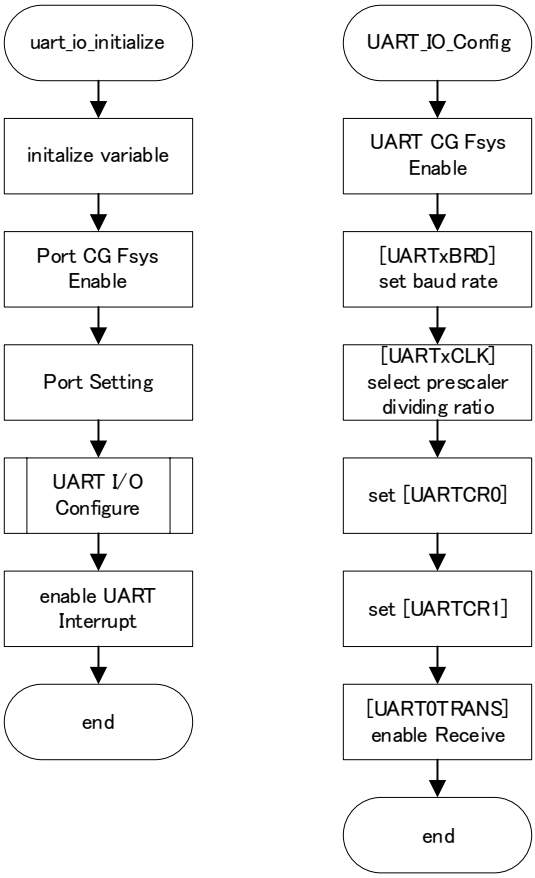
11.2.7. DAP ブランクチェック



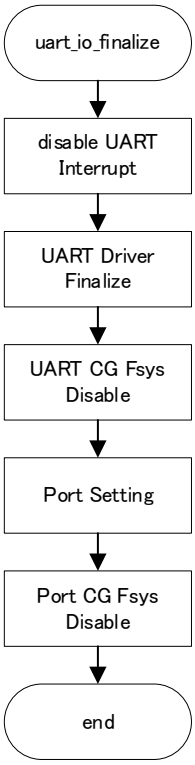
11.2.8. App ステートから Lib ステートへのエントリー



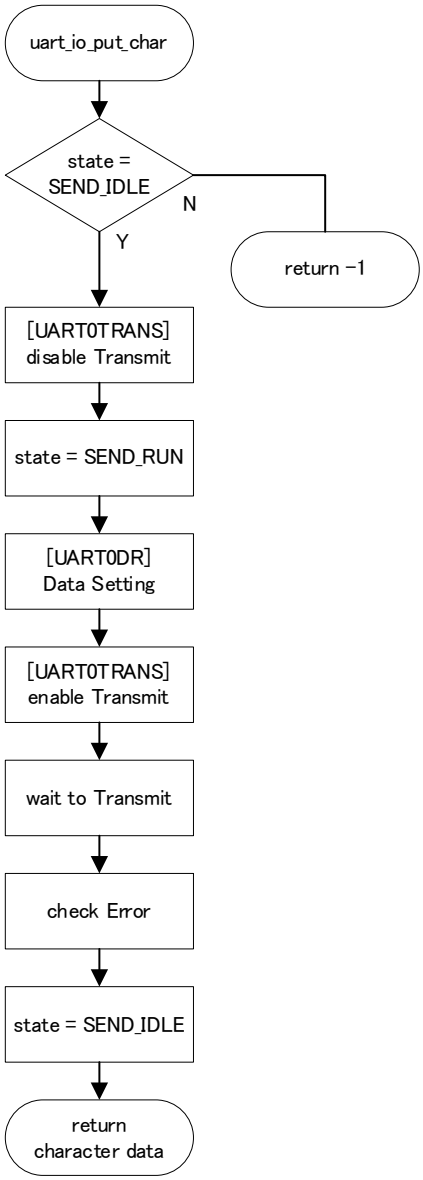
11.2.9. UART 初期化



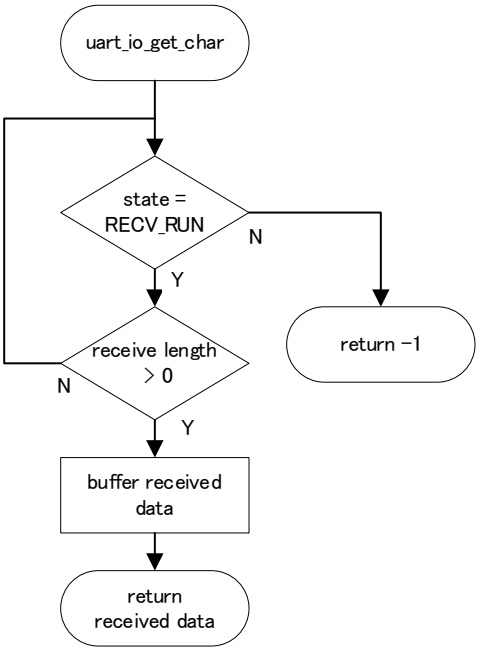
11.2.10. UART 終了



11.2.11. UART による文字データ送信

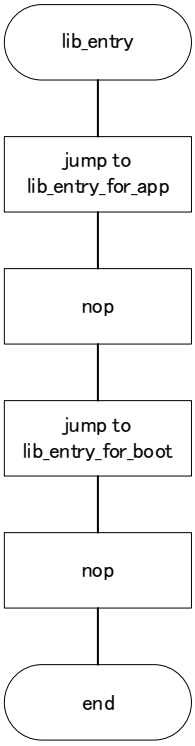


11.2.12. UART による文字データ受信

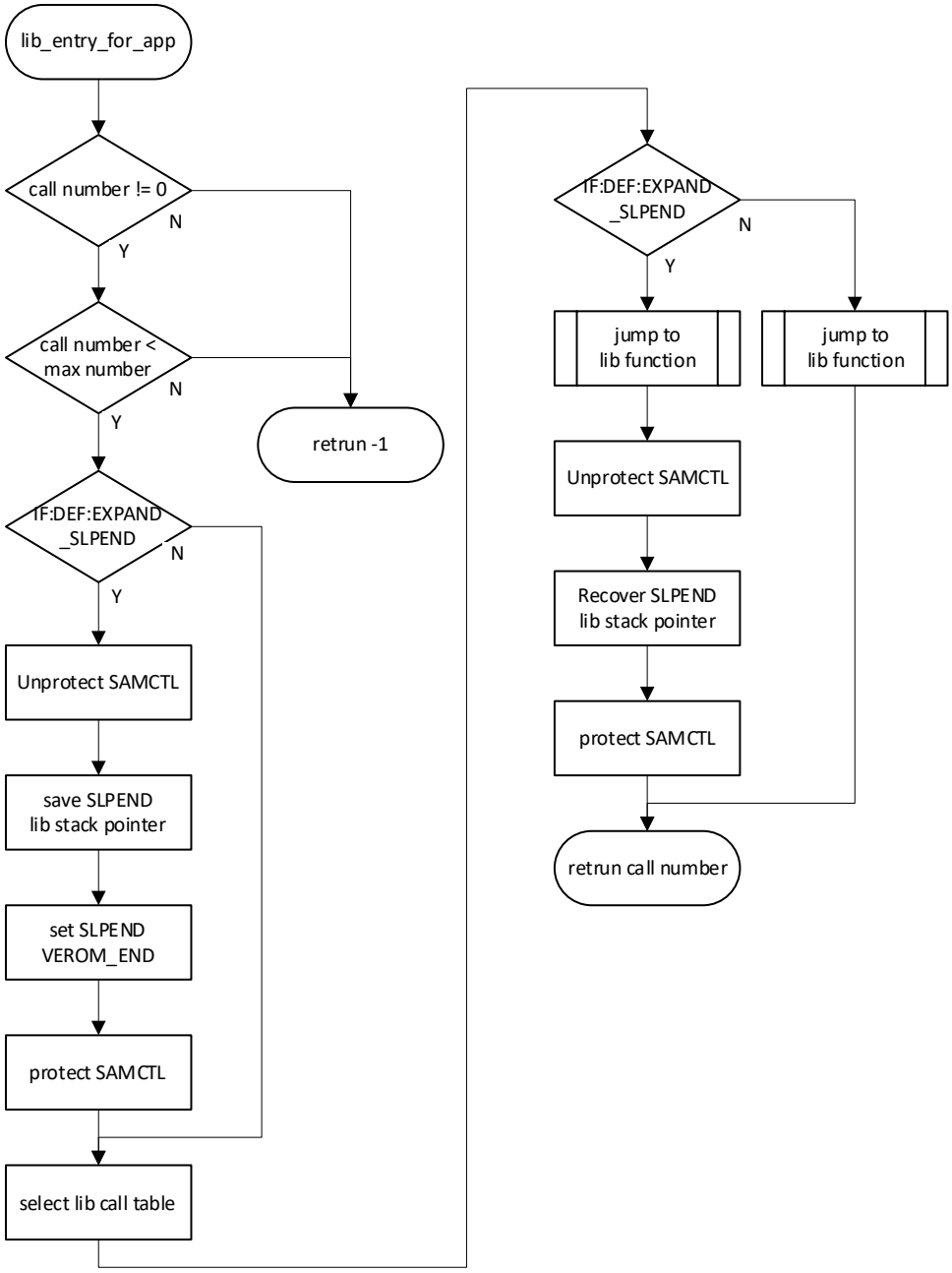


11.3. Lib ステート

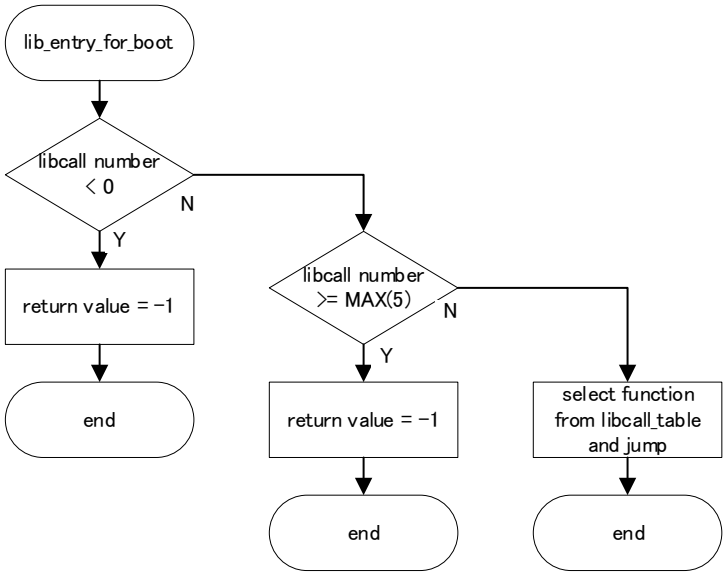
11.3.1. Lib ステートエントリー関数



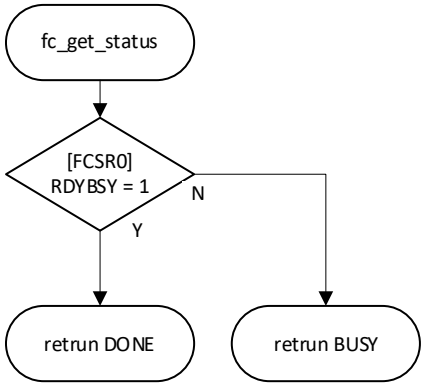
11.3.2. App ステートからの Lib ステートエントリー関数



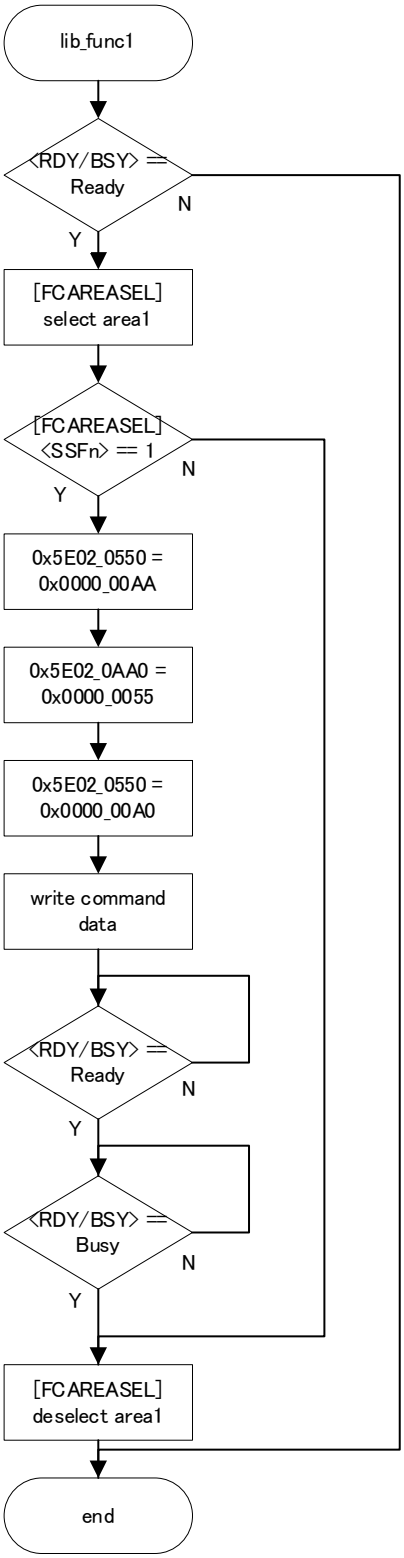
11.3.3. Boot ステートからの Lib 関数呼び出し



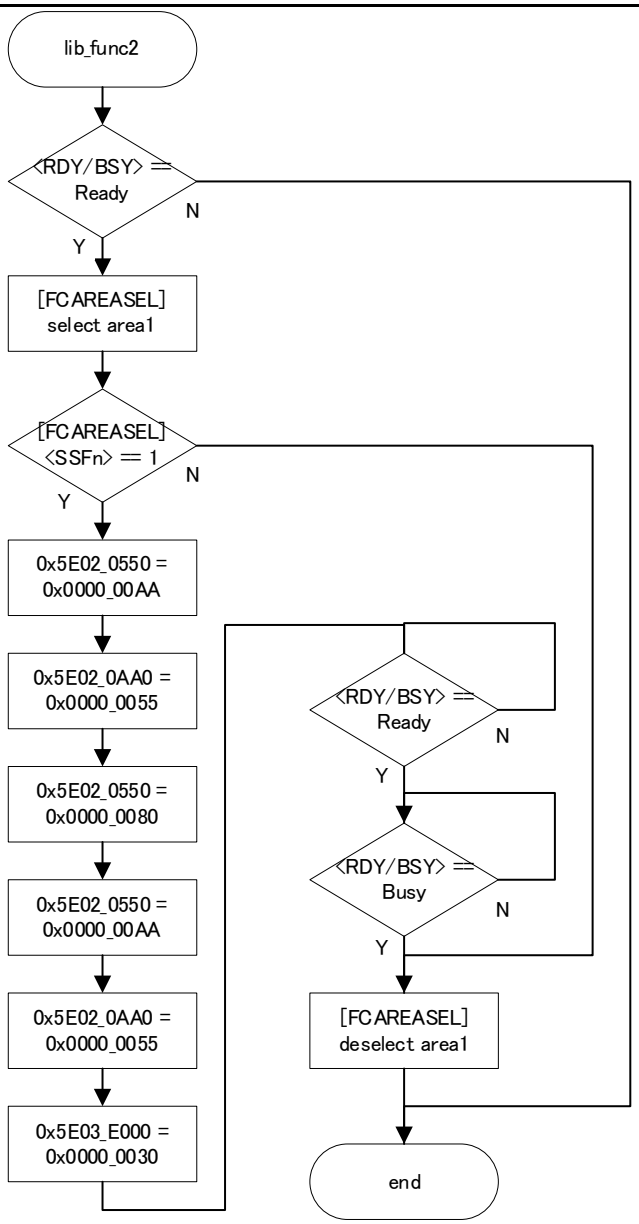
11.3.4. 自動実行時の Ready/Busy 状態取得



11.3.5. Flash ページ書き込み

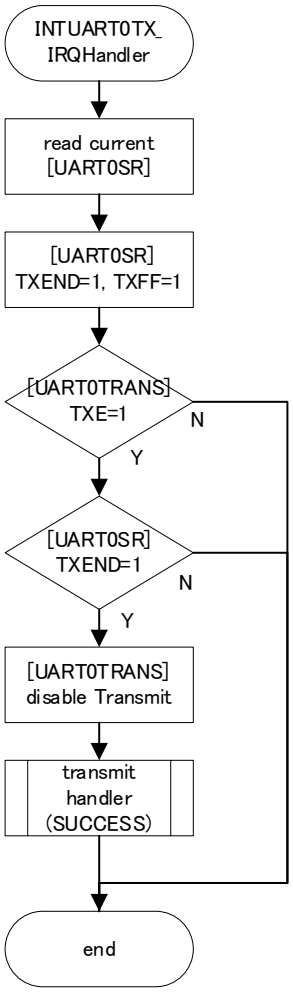


11.3.6. Flash ブロック消去

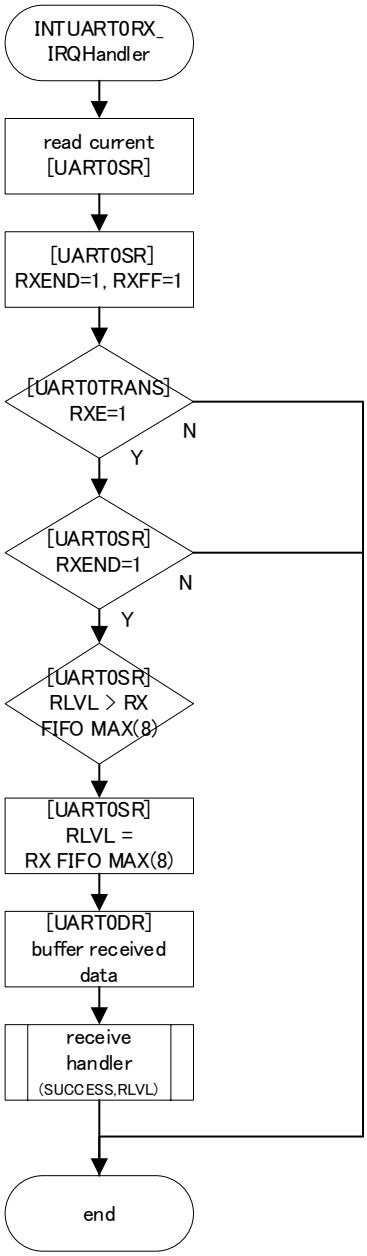

lib_func2
<RDY/BSY> == Ready
N
Y
[FCAREASEL] select area1
[FCAREASEL] <SSFn> == 1
N
Y
0x5E02_0550 = 0x0000_00AA
0x5E02_0AA0 = 0x0000_0055
0x5E02_0550 = 0x0000_0080
0x5E02_0550 = 0x0000_00AA
0x5E02_0AA0 = 0x0000_0055
0x5E03_E000 = 0x0000_0030
<RDY/BSY> == Ready
N
Y
<RDY/BSY> == Busy
N
Y
[FCAREASEL] deselect area1
end

11.4. 割り込み

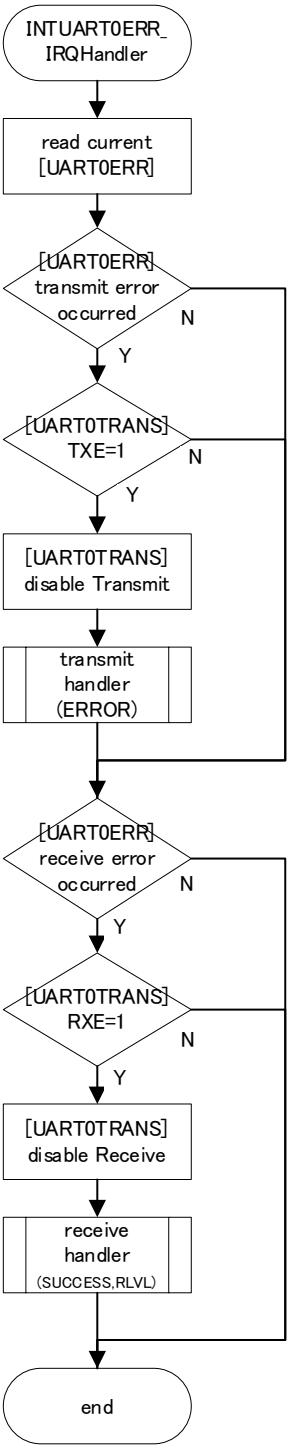
11.4.1. UART 送信 IRQ ハンドラー



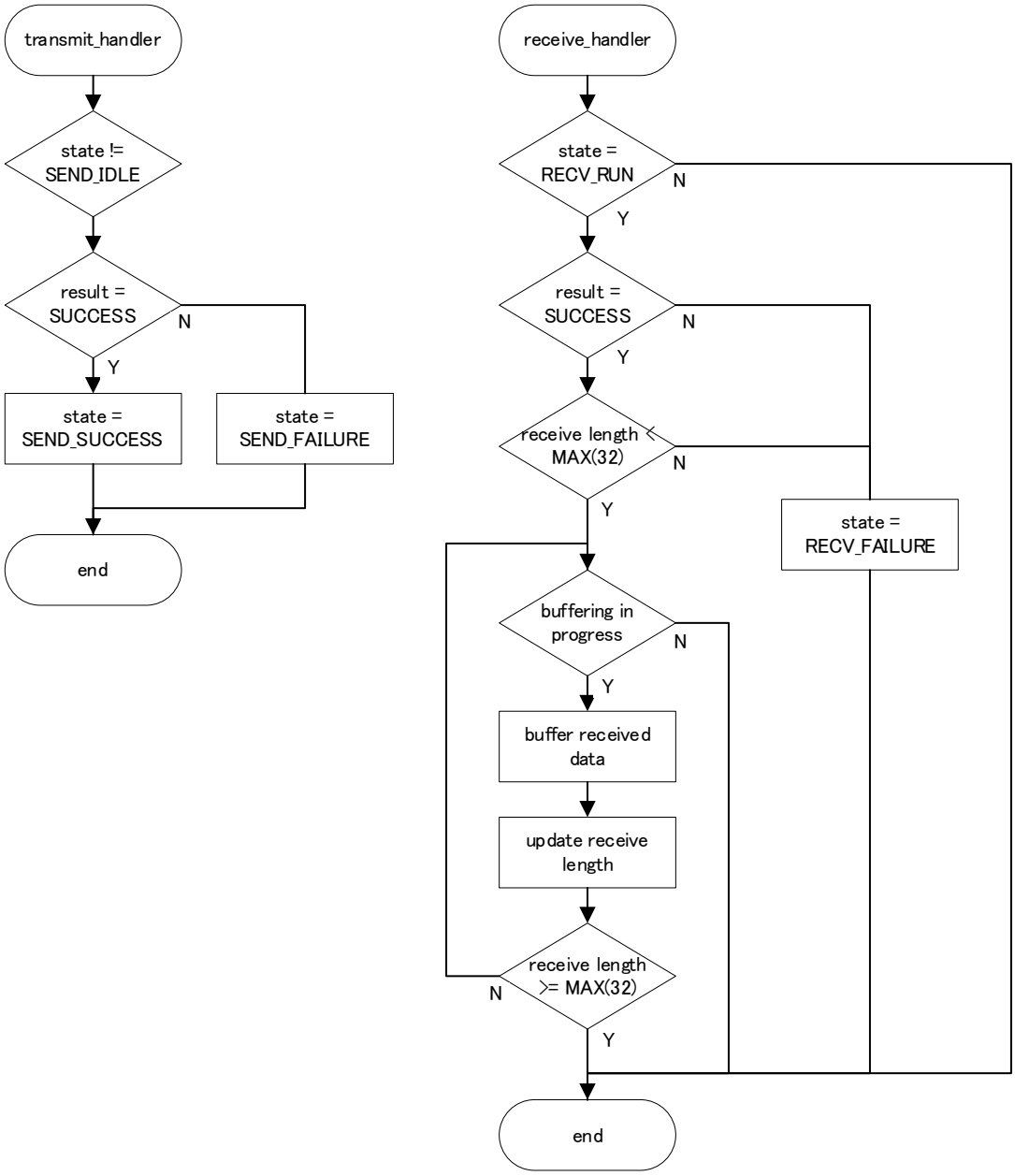
11.4.2. UART 受信 IRQ ハンドラー



11.4.3. UART エラーIRQ ハンドラー



11.4.4. UART 用送信/受信ハンドラー



12. 改訂履歴

Revision	日付	変更項目
1.0	2026-07-30	初版

製品取り扱い上のお願い

株式会社東芝およびその子会社ならびに関係会社を以下「当社」といいます。
本資料に掲載されているハードウエア、ソフトウエアおよびシステムを以下「本製品」といいます。

- 本製品に関する情報等、本資料の掲載内容は、技術の進歩などにより予告なしに変更されることがあります。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。また、文書による当社の事前の承諾を得て本資料を転載複製する場合でも、記載内容に一切変更を加えたり、削除したりしないでください。
- 当社は品質、信頼性の向上に努めていますが、半導体・ストレージ製品は一般に誤作動または故障する場合があります。本製品をご使用頂く場合は、本製品の誤作動や故障により生命・身体・財産が侵害されることのないように、お客様の責任において、お客様のハードウエア・ソフトウエア・システムに必要な安全設計を行うことをお願いします。なお、設計および使用に際しては、本製品に関する最新の情報（本資料、仕様書、データシート、アプリケーションノート、半導体信頼性ハンドブックなど）および本製品が使用される機器の取扱説明書、操作説明書などをご確認の上、これに従ってください。また、上記資料などに記載の製品データ、図、表などに示す技術的な内容、プログラム、アルゴリズムその他応用回路例などの情報を使用する場合は、お客様の製品単独およびシステム全体で十分に評価し、お客様の責任において適用可否を判断してください。
- 本製品は、特別に高い品質・信頼性が要求され、またはその故障や誤作動が生命・身体に危害を及ぼす恐れ、膨大な財産損害を引き起こす恐れ、もしくは社会に深刻な影響を及ぼす恐れのある機器（以下“特定用途”という）に使用されることは意図されていませんし、保証もされていません。特定用途には原子力関連機器、航空・宇宙機器、医療機器（生命直結機器）、車載・輸送機器、防衛関連機器などが含まれますが、本資料に個別に記載する用途は除きます。特定用途に使用された場合には、当社は一切の責任を負いません。なお、詳細は当社営業窓口まで、または当社 Web サイトのお問い合わせフォームからお問い合わせください。
- 本製品を、国内外の法令、規則及び命令により、製造、使用、販売を禁止されている製品に使用することはできません。
- 本資料に掲載してある技術情報は、製品の代表的動作・応用を説明するためのもので、その使用に際して当社及び第三者の知的財産権その他の権利に対する保証または実施権の許諾を行うものではありません。
- 別途、書面による契約またはお客様と当社が合意した仕様書がない限り、当社は、本製品および技術情報に関して、明示的にも黙示的にも一切の保証（機能動作の保証、商品性の保証、特定目的への合致の保証、情報の正確性の保証、第三者の権利の非侵害保証を含むがこれに限らない。）をしておりません。
- 本製品、または本資料に掲載されている技術情報を、大量破壊兵器の開発等の目的、軍事利用の目的、あるいはその他軍事用途の目的で使用しないでください。また、輸出に際しては、「外国為替及び外国貿易法」、「米国輸出管理規則」等、適用ある輸出関連法令を遵守し、それらの定めるところにより必要な手続を行ってください。
- 本製品の RoHS 適合性など、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い合わせください。本製品のご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用ある環境関連法令を十分調査の上、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、当社は一切の責任を負いかねます。