Application Note

SAM

Table of Contents

1. Preface

This application note describes sample software SAM that uses the Secure Access Memory (SAM).
This document helps the user check operation of a product under development and develop its program.

2. Technical Term

Term/Abbreviation	Definition
CG	Clock Control and Operation Mode
SAM	Secure Access Memory
UART	Universal Asynchronous Receiver Transmitter

3. Reference Document

Document	Notes
Data sheet	Refer to the data sheet of MCU to be used.
Reference manual	Refer to the reference manual of each IP to be used.
Application note MCU User Guide	Refer to the MCU user guide to be used.

4. Target Sample Program

Sample Program	Outline
SAM	Sample program of the Secure Access Memory Function

5. Sample Program: SAM

This is the sample software that uses the Secure Access Memory (SAM) function to set memory areas for each state. After setting the memory areas, the software rewrites, erases, or reads Code Flash according to commands entered from a terminal application.

5.1. Operation Overview

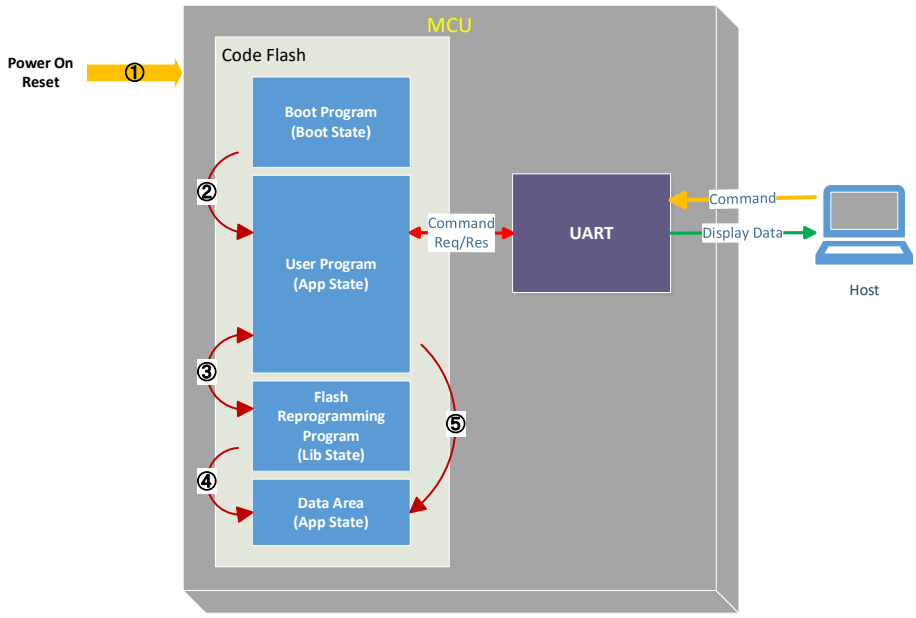
1. After a Power-On Reset, the device transitions to the Boot state and executes the Boot program. If the Power-On Reset is performed while pressing the push switch (USW_1), the Secure Access Memory (SAM) function is enabled. The Boot program sets the Lib state and App state, and then transitions to the App state.

2. The program in the App state communicates with a terminal application via UART. The following operations are performed according to commands entered from the terminal application.

- When the write command is received:
  The device transitions to the Lib state and writes the string specified by the command to Code Flash. After the write operation is completed, control returns to the user program and the device transitions back to the App state.

- When the erase command is received:
  After transitioning to the Lib state, the device erases Code Flash. Upon completion of the erase operation, control returns to the user program and the device transitions back to the App state.

- When the read command is received:
  After transitioning to the Lib state, the device reads the string stored in Code Flash and displays it on the terminal application. After the display operation is completed, the device transitions back to the App state.



Details of each command are as follows.

Command	Format	Description
write	write <string>	Writes the specified string to Code Flash.
read	read	Reads the previously written string from Code Flash.
erase	erase	Erases Code Flash.

5.2. Function to Use

The functions to use are as follows:

IP	Purpose
PORT (Push-Switch)	Enables/disables the SAM function
SAM	Security function for memory access
UART	Communication with a terminal application

5.3. Interrupt to Use

For details, please refer to the “Interrupt List by Sample” in the MCU User’s Manual.

5.4. Configuration

Configuration setting.

Configuration	Soft Definition Name	Current Value (Defaults)	Description
Write DATA MAX.	CFG_DATA_LENGTH	10	Writable Character Count (Unit: character)

5.5. Example of Terminal Emulator Output

5.5.1. Normal Operation

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

write data > abc

command > read

read data > abc
command > erase

erase
command >
```

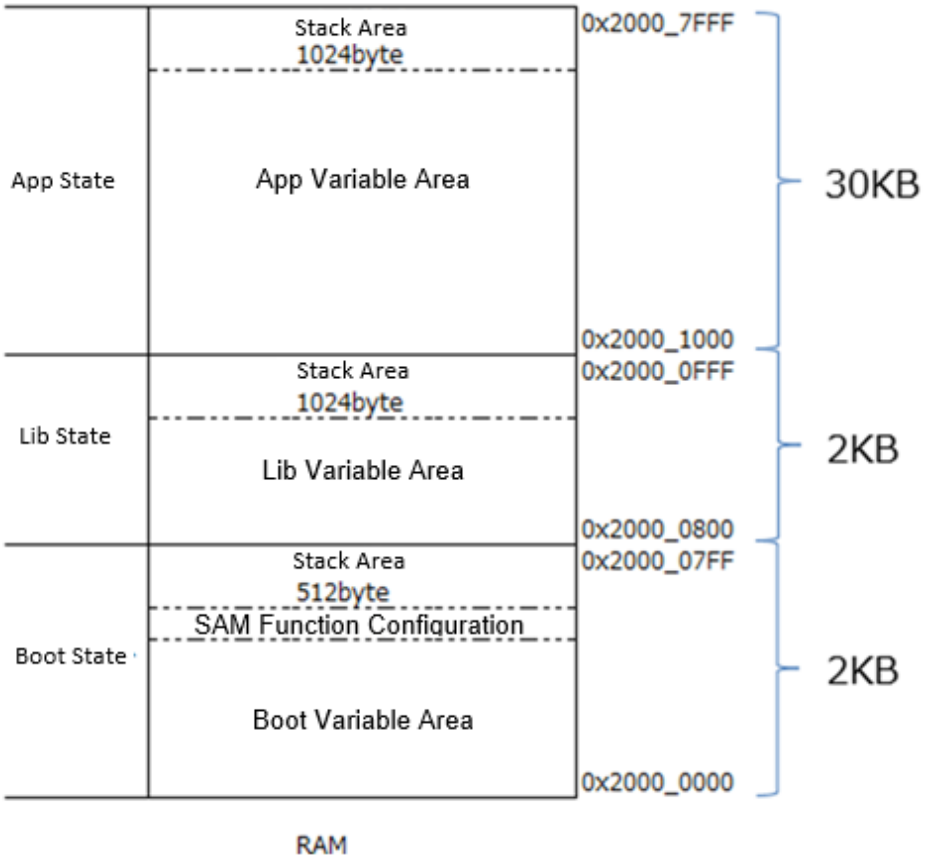
*When the SAM function is disabled (Power-On Reset without pressing USW_1), "Secure Disable" is displayed.

5.5.2. Error Condition

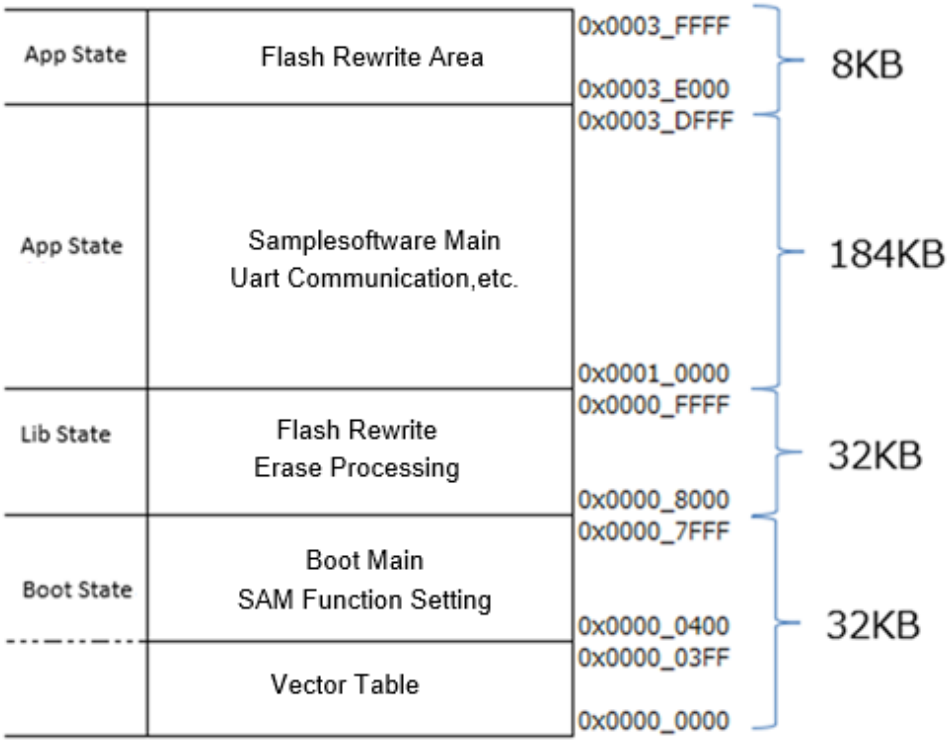
```
command > A

 Command Error !!
command > write 0123456789A

  Parameter Error !!
command > erase

erase
command > read

FFFB error!!
command >
```

6. Memory Map

The Code Flash and RAM memory maps for each state are shown below.



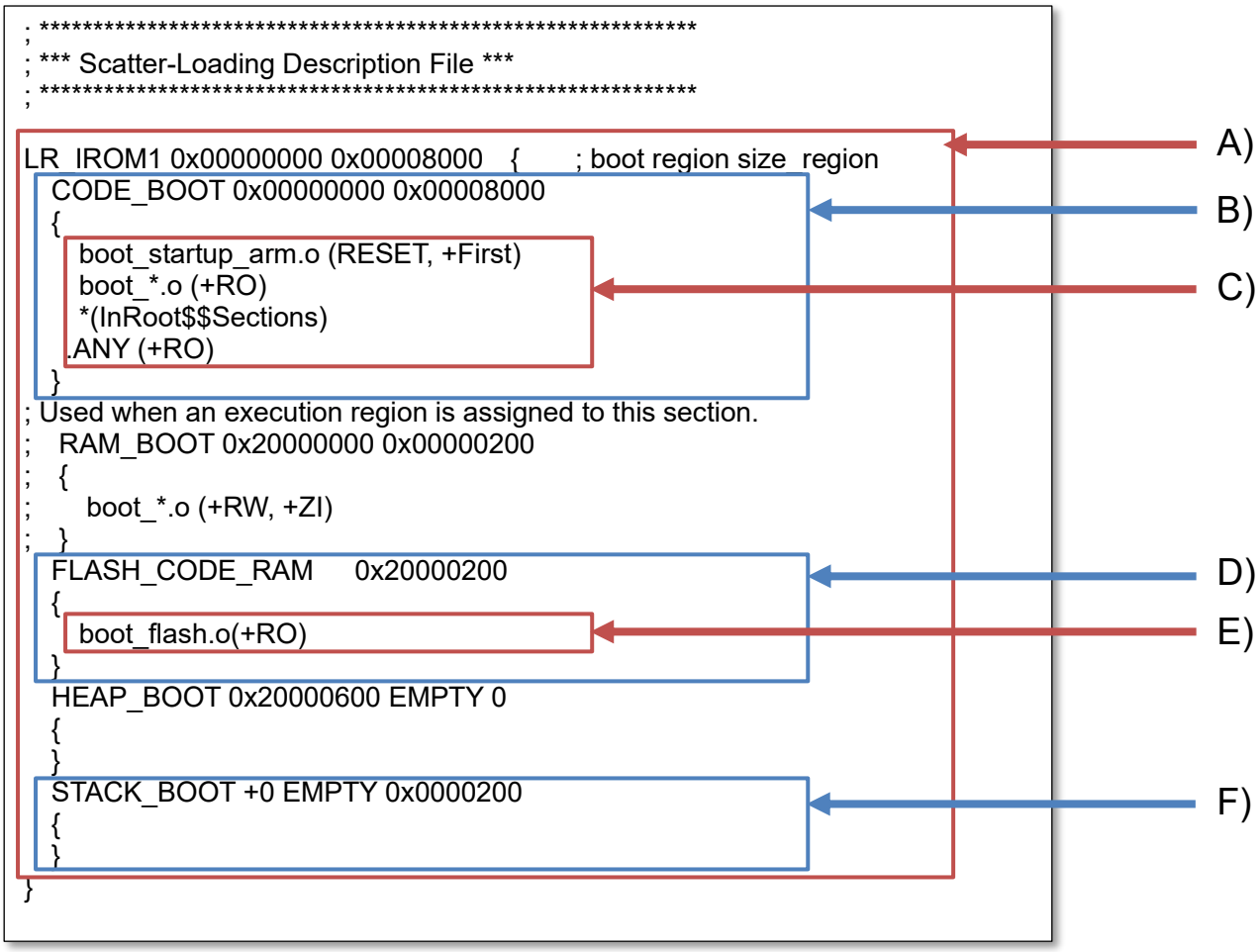
RAM



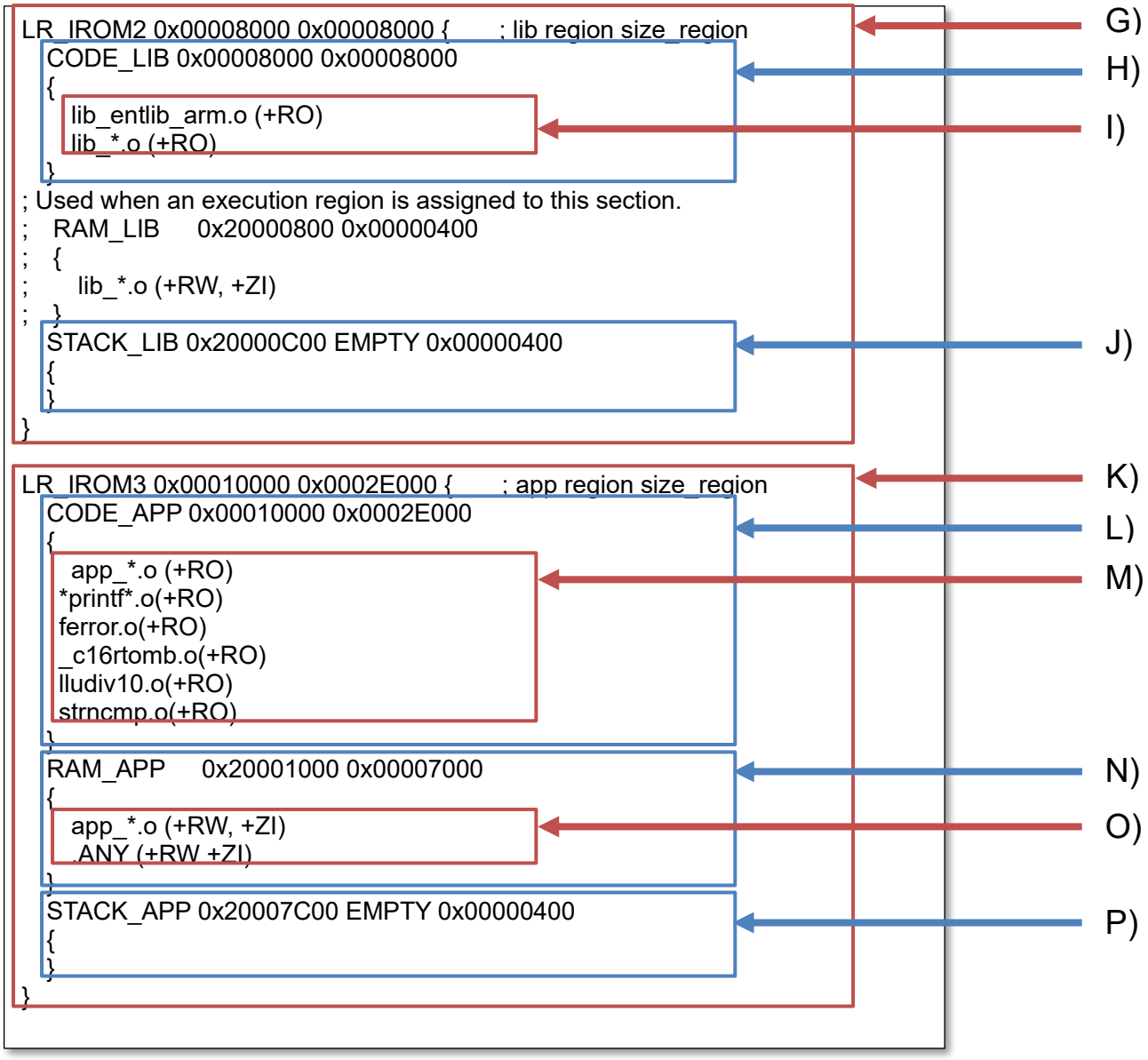
Code Flash

7. Linker Script

7.1. Keil MDK Scatter File(.sct)



A) Boot State Load Area: Address 0x0000_0000 - 0x0000_7FFF
B) Boot Code Execution Area: Address 0x0000_0000 - 0x0000_7FFF
C) Input Section: Contains the Boot state program.
D) Flash Code Execution Area: Address 0x2000_0000 - 0x2000_01FF
E) Input Section: Contains the SAM function setting program (SAM enabled/disabled).
F) Stack Execution Area: Address 0x2000_0600 - 0x2000_07FF



A) Lib State Load Area: Address 0x0000_8000 - 0x0000_FFFF
B) Lib Code Execution Area: Address 0x0000_8000 - 0x0000_FFFF
C) Input Section: Contains the Lib state program.
The Lib state entry process (lib_entlib_arm.o) is placed at the beginning of this section so that the Lib state entry address is 0x8000.
D) Stack Execution Area: Address 0x2000_0C00 - 0x2000_0FFF
E) App State Load Area: Address 0x0001_0000 - 0x0001_DFFF
F) App Code Execution Area: Address 0x0001_0000 - 0x0001_DFFF
G) Input Section: Contains the App state program and standard library functions used by the App state (such as printf and strncmp).
For details, refer to Section 8.2.
H) App RAM Execution Area: Address 0x2000_1000 - 0x2000_6FFF
I) Input Section: Contains variables used by the App state.
J) Stack Execution Area: Address 0x2000_7C00 - 0x2000_7FFF

7.2. IAR EWARM Linker Configuration File(.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 Address Macro Definitions

Boot State: Address 0x0000_0000 - 0x0000_7FFF
Lib State: Address 0x0000_8000 - 0x0000_FFFF
App State: Address 0x0001_0000 - 0x0003_DFFF

B) RAM Address Macro Definitions

Boot State: Address 0x2000_0000 - 0x2000_07FF
Lib State: Address 0x2000_0800 - 0x2000_0FFF
App State: Address 0x2000_1000 - 0x2000_7FFF

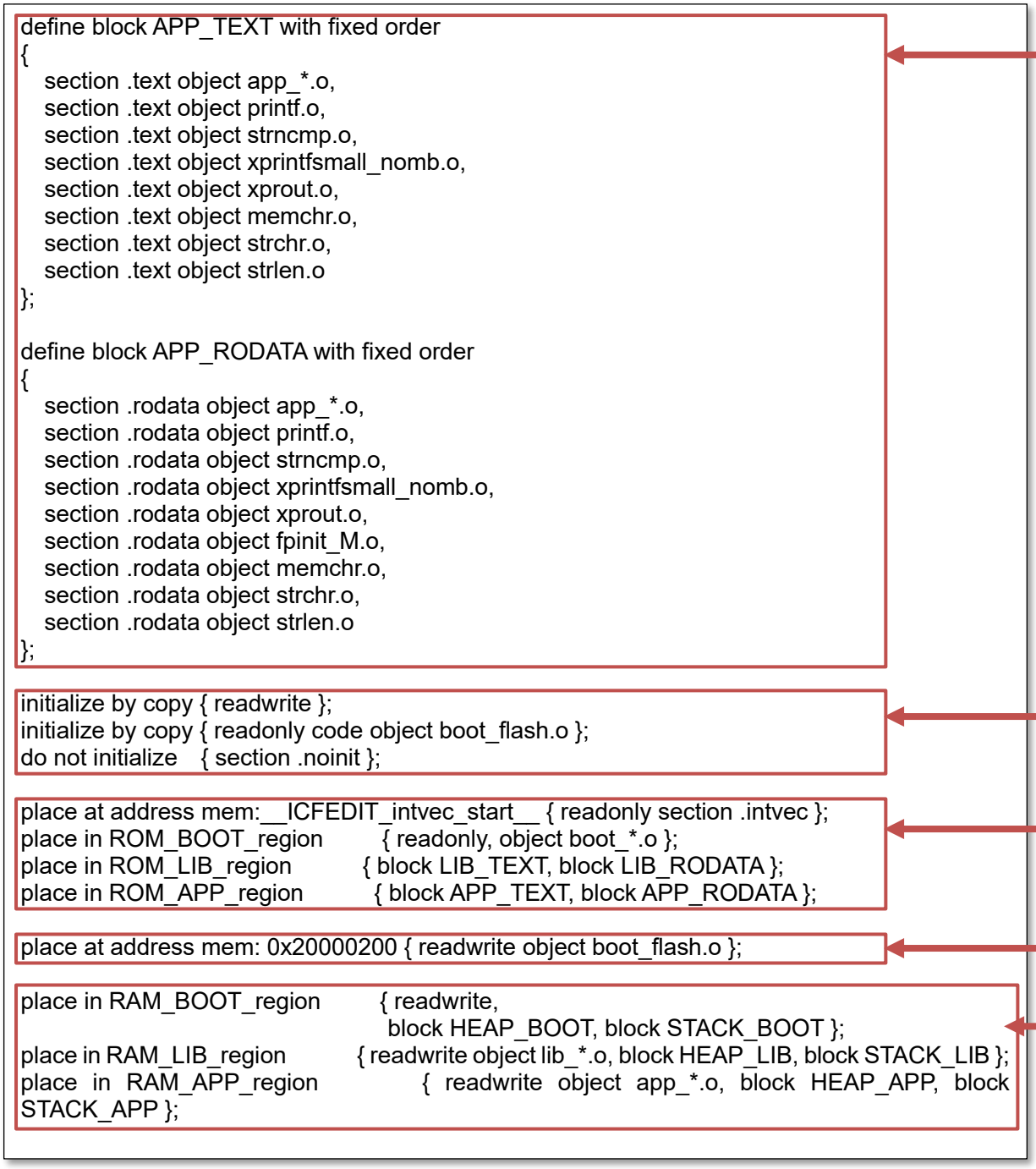
C) Stack and Heap Size Macro Definitions

Boot State: Stack = 512 bytes, Heap = 512 bytes
Lib State: Stack = 1024 bytes
App State: Stack = 1024 bytes

```
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) Defines the execution areas for each state.
E) Assigns the stack and heap of each state to memory blocks with 8-byte alignment.
F) Assigns the Lib state program and read-only sections to memory blocks.
The Lib state entry process (lib_entlib_iar.o) is placed at the beginning so that the Lib state entry address is set to 0x8000.

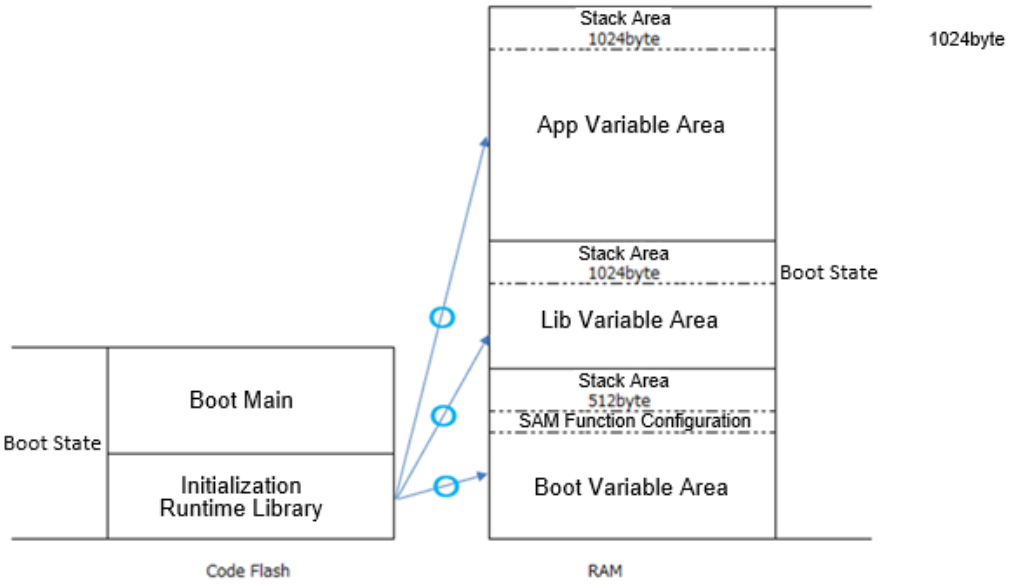


G) Assigns the App state program and read-only sections to memory blocks.
Standard library functions used by the App state (such as printf and strncmp) are manually assigned.
For details, refer to Section 8.2.
H) Assigns the Boot state SAM function setting program (SAM enable/disable) to RAM.
The program is copied from ROM to RAM.
I) Assigns the ROM areas for each state.
J) Assigns the Boot state SAM function setting routine to address 0x2000_0200.
K) Assigns the RAM areas for each state.

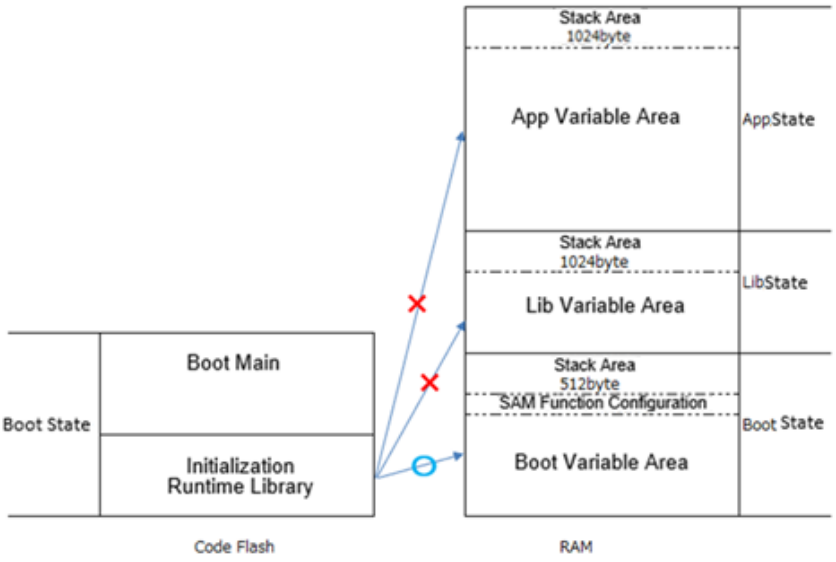
8. Implementation Notes

8.1. Initialization

To initialize variables and other data using the runtime library, all RAM areas must be assigned to the Boot State area(SBP) during the SAM setting process after reset release.
If the RAM areas for each state are set before initialization, unauthorized areas may be accessed during the initialization process, resulting in a SAM reset.
Therefore, the RAM areas for each state should be set when transitioning from the Boot state to the App state, after the initialization process has completed.



Setting the desired RAM areas from the beginning causes an access violation and triggers a reset.



After initialization, set the desired RAM areas during the transition from the Boot state to the App state.

8.2. Manual Assignment of Standard Library Function (Linker Script)

Standard library functions such as printf() and strncmp() are placed in the Boot state area unless explicitly specified in the linker script.

Therefore, when these functions are called from the App state program, they must be assigned manually. For the manual assignment method, refer to Chapter 7.

Function objects called by the standard library functions must also be assigned manually (For example, printf.o requires related function objects such as ferror.o, _c16rtomb.o, and lludiv10.o).

These function objects are listed in the map file.

8.3. SAM Enable/Disable Setting

SAM enable/disable settings are configured together with other settings, such as the irreversible flash protection setting and lock setting, by using the automatic SAMNRFP program command of the flash memory. When configuring these settings, ensure that the lock setting remains disabled.

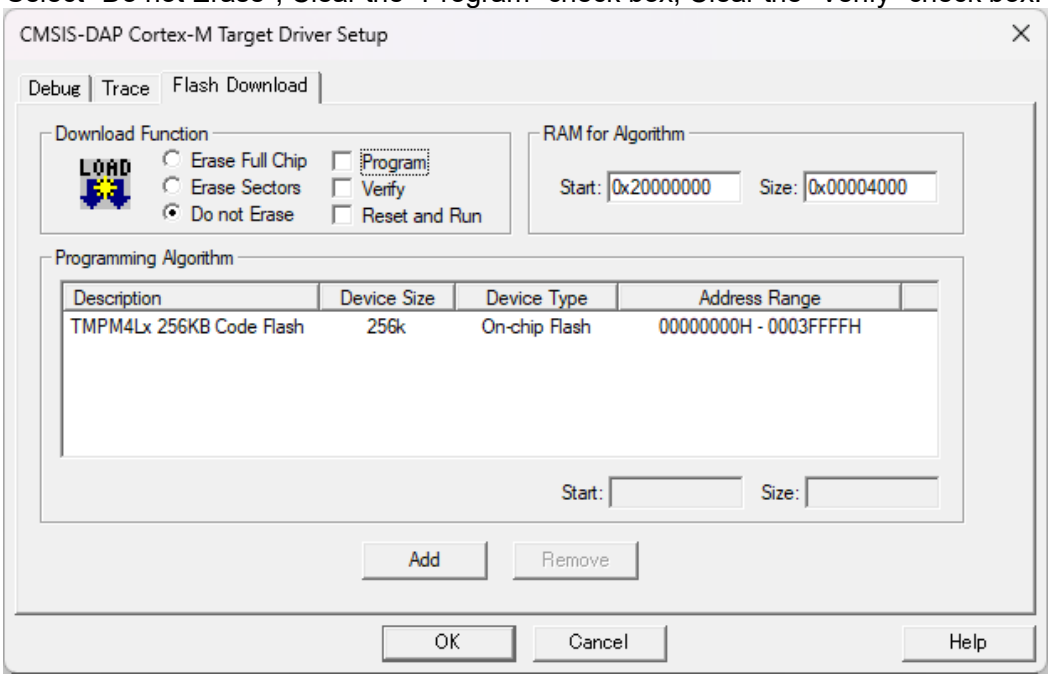
If the lock setting is enabled, the automatic SAMNRFP release command becomes unavailable, and neither SAM nor irreversible flash protection can be disabled.
As a result, writing to and erasing the flash memory area becomes permanently impossible.

9. Debugging Notes

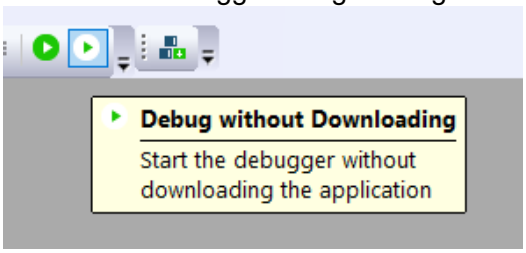
9.1. Unable to Connect to the Debugger When the SAM Function is Enabled

1. Cause
   Immediately after reset release, the device is in the Boot state, while RAM is set to the App state areas. The development tool (debugger) attempts to write a Flash Loader to RAM. Since this requires access to the App state RAM areas from the Boot state, an access violation occurs and a SAM reset is generated.

2. Workaround
   - For Keil MDK:
     Select "Flash" -> "Configure Flash Tools..." -> "Settings" -> "Flash Download".
     Under "Download Function", configure the following settings:
     Select "Do not Erase", Clear the "Program" check box, Clear the "Verify" check box.



   - For IAR EWARM: Connect to the debugger using “Debug Without Downloading”



When the SAM function is enabled, the program cannot be rewritten using the Flash Loader downloaded by the development tool. To rewrite the program, the SAM function must first be disabled. Currently, the SAM function can be disabled as follows:

1. Connect the debugger using "Debug Without Downloading" (IAR EWARM).
2. Execute the ROM-to-RAM transfer process for the SAM disable-setting process.
3. Move the program counter to the SAM disable-setting process and execute it.

Note: Before enabling the SAM function, confirm that the SAM disable-setting process operates correctly.
If the above procedure does not resolve the issue, perform the following:

Enter the machine code of the SAM disable-setting process into the Boot state RAM area using the memory window of the development tool.
Move the program counter to the process and execute it.

9.2. SAM Function Enabled Causes Unexpected Reset

- Symptom
  After debugging in the App state, a reset occurs when the device transitions to the Lib state and stops at a breakpoint.

- Cause
  During App state debugging, RAM areas belonging to the App state were displayed in the Memory window or Watch window.
  When the device transitioned to the Lib state, the debugger read the App state RAM areas, causing an access violation and resulting in a SAM reset.

- Workaround
  Hide the Memory window and Watch window.

9.3. Repeated Resets Occur When the SAM Function Is Enabled

- Symptom
  The program cannot be stopped using the stop function of the development tool (debugger).

- Cause
  An access violation occurs due to incorrect program placement.
  The access violation triggers a SAM reset, causing the program to restart repeatedly and preventing it from being stopped.

- Workaround
  Set a breakpoint at the beginning of Reset_Handler in the startup code.
  When a reset occurs due to an access violation, execution stops at this breakpoint, preventing the program from restarting repeatedly.

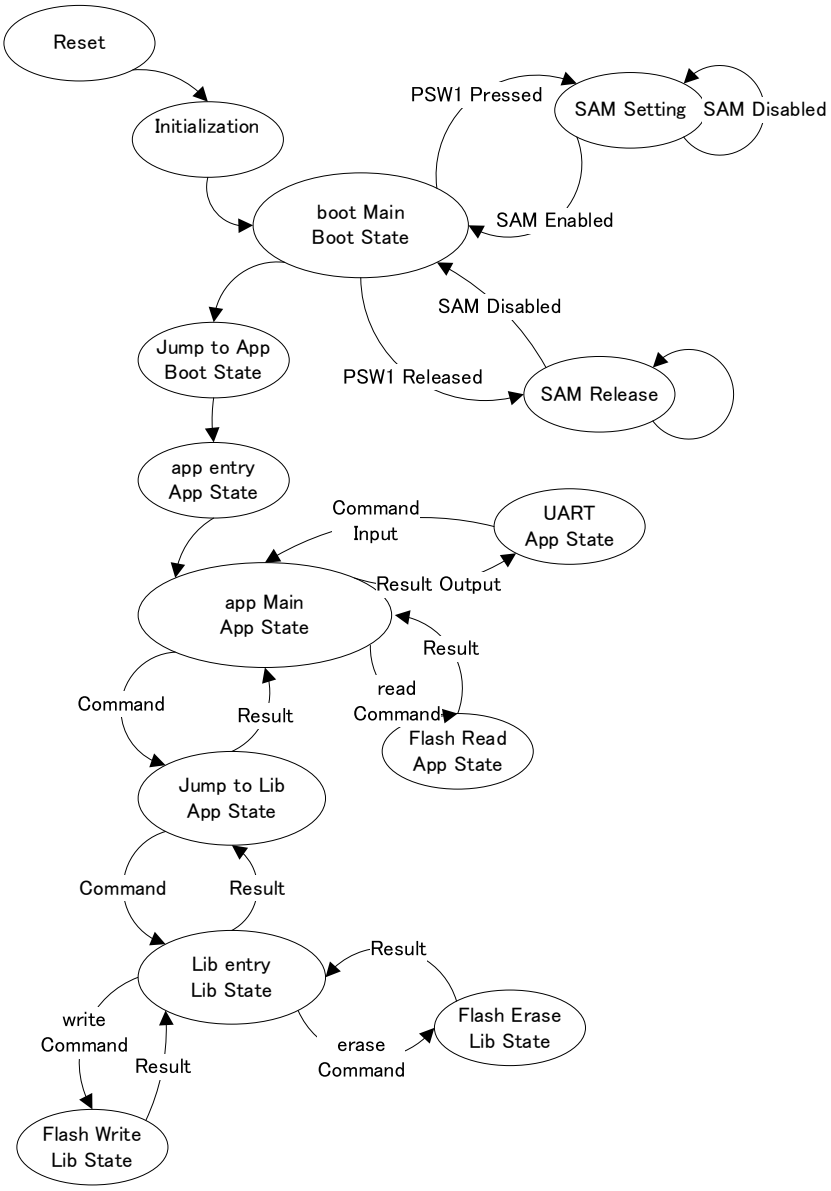
9.4. SAM Function Enable/Disable Setting Does Not Work

- Symptom
  When debugging with IAR EWARM, the debugger reset function is used after executing the SAM function enable(or disable) setting process.
  However, when the program is subsequently executed, the SAM function does not become enabled (or disabled).
  Note: This issue does not occur with Keil MDK.

- Cause
  The hardware is not being reset.For example, when using CMSIS-DAP, this issue can occur if the reset setting under "CMSIS DAP" in the IAR EWARM options is set to "System (Default)".

- Workaround
  Reset the hardware by turning the power off and on again, or pressing the reset button.

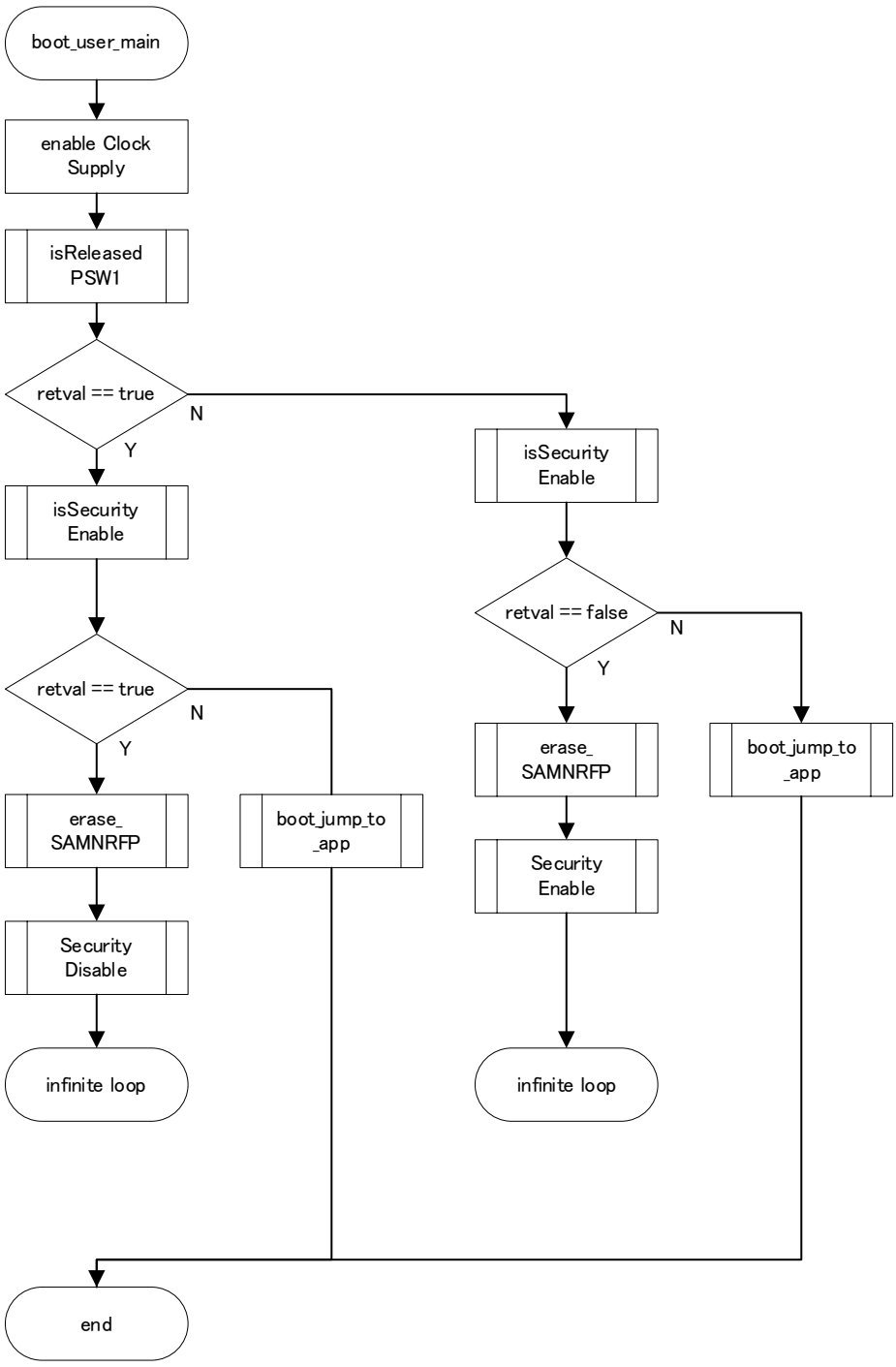
10. State Transition Diagram



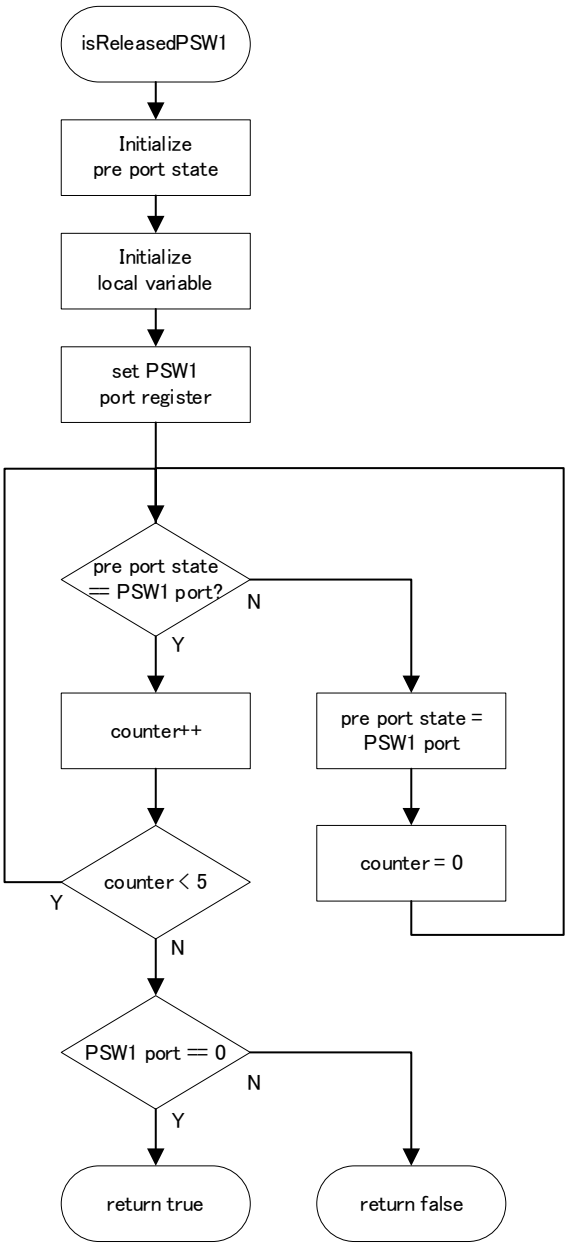
11. Flowchart

11.1. Boot State

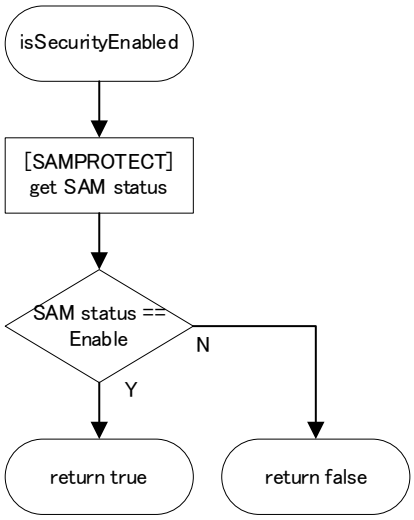
11.1.1. main (boot_user_main)



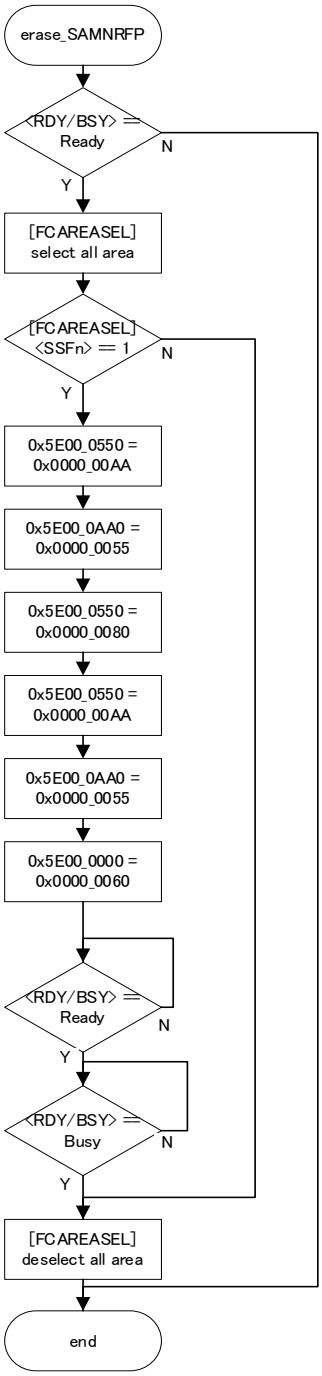
11.1.2. PSW1 Press/Release Detection



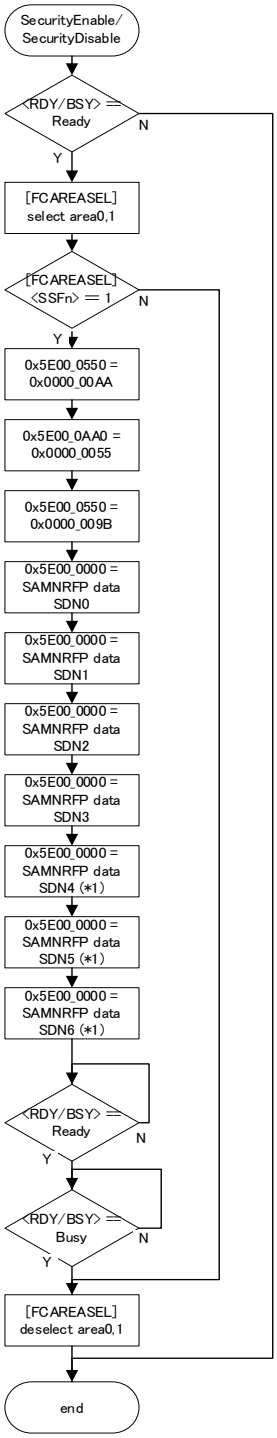
11.1.3. Get SAM Status



11.1.4. SAM and CBPEND Setting Clear (Automatic SAMNRFP Release)

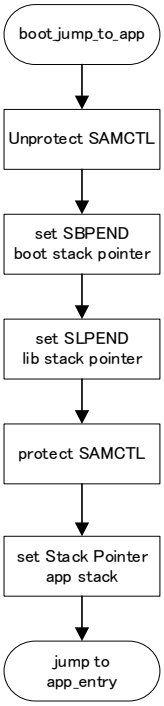


11.1.5. SAM Enable/Disable Setting



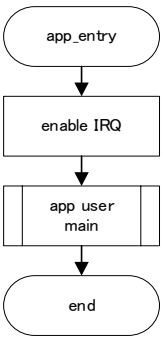
*1:
SecurityEnable() NRFPSET=1
SecurityDisable() NRFPSET=0

11.1.6. Entry form the Boot State to the App State

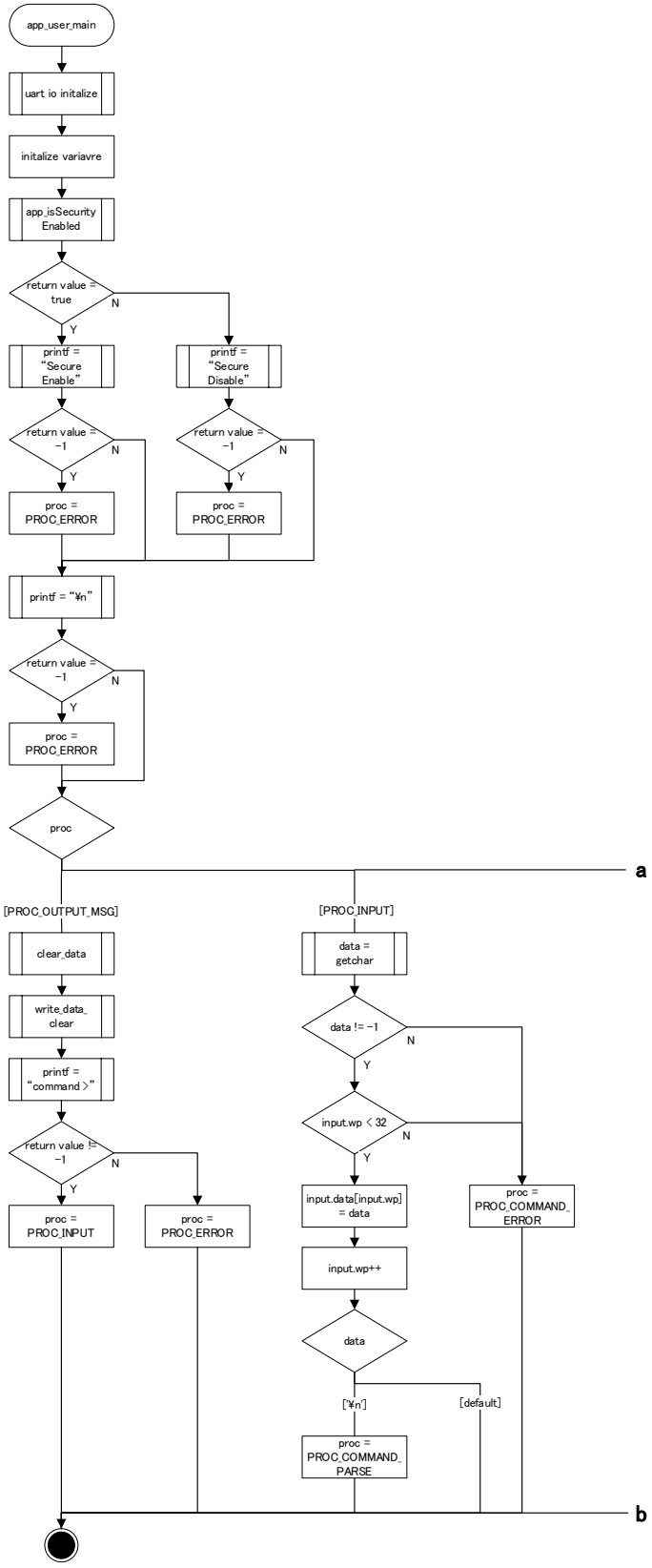


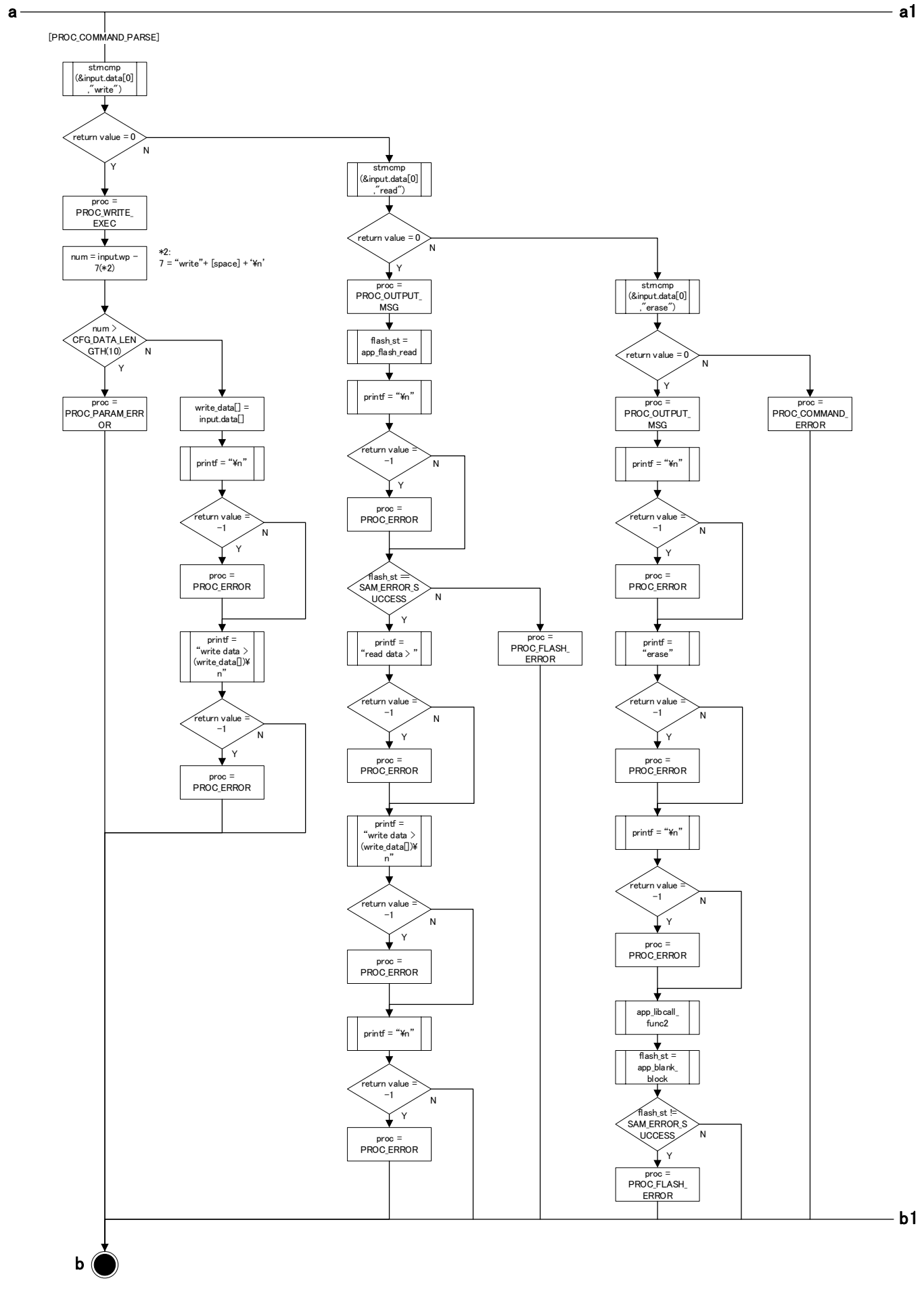
11.2. App State

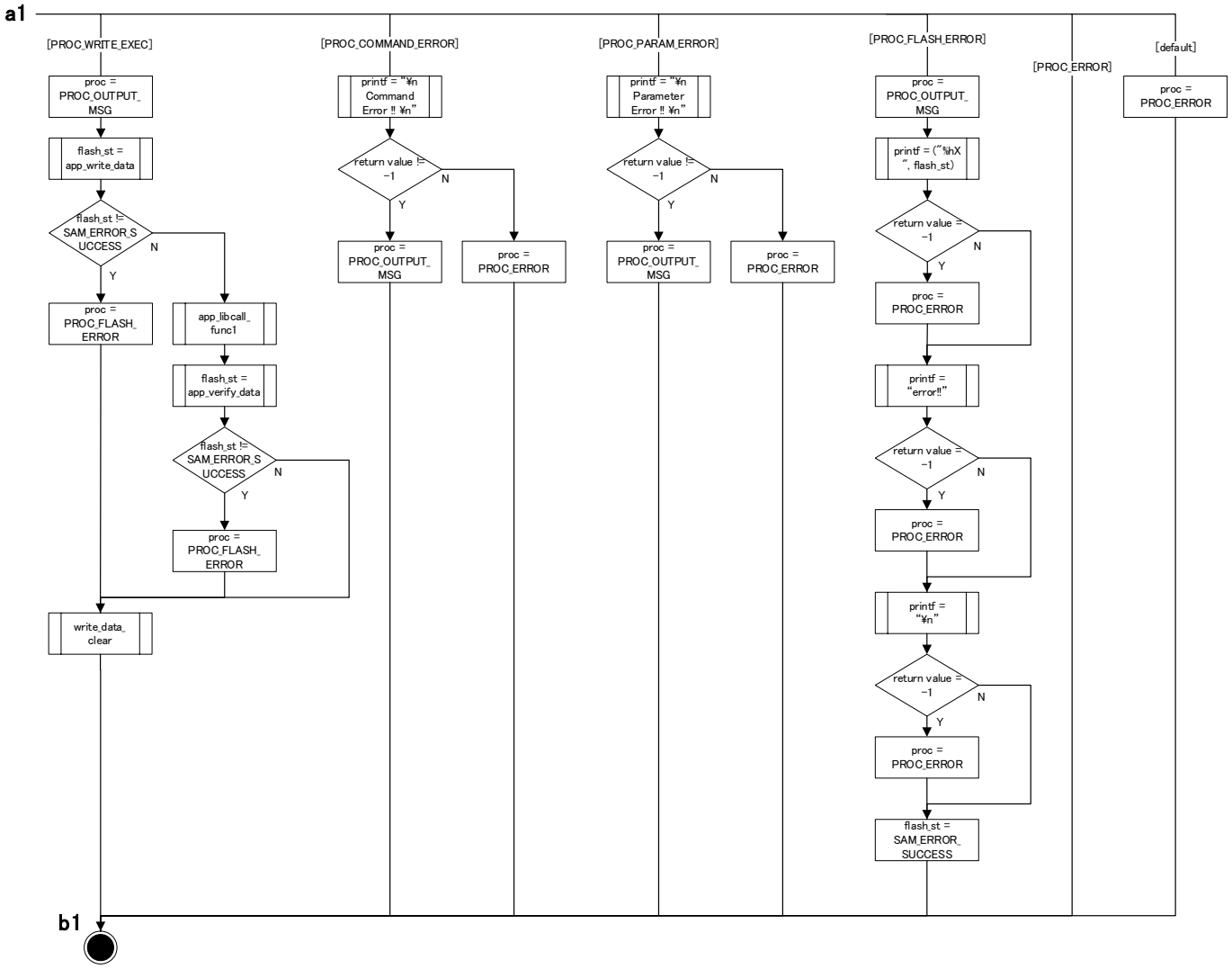
11.2.1. App State Entry Function



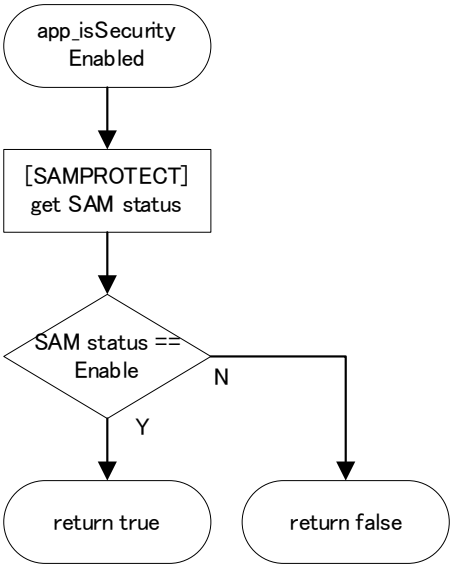
11.2.2. main (app_user_main)



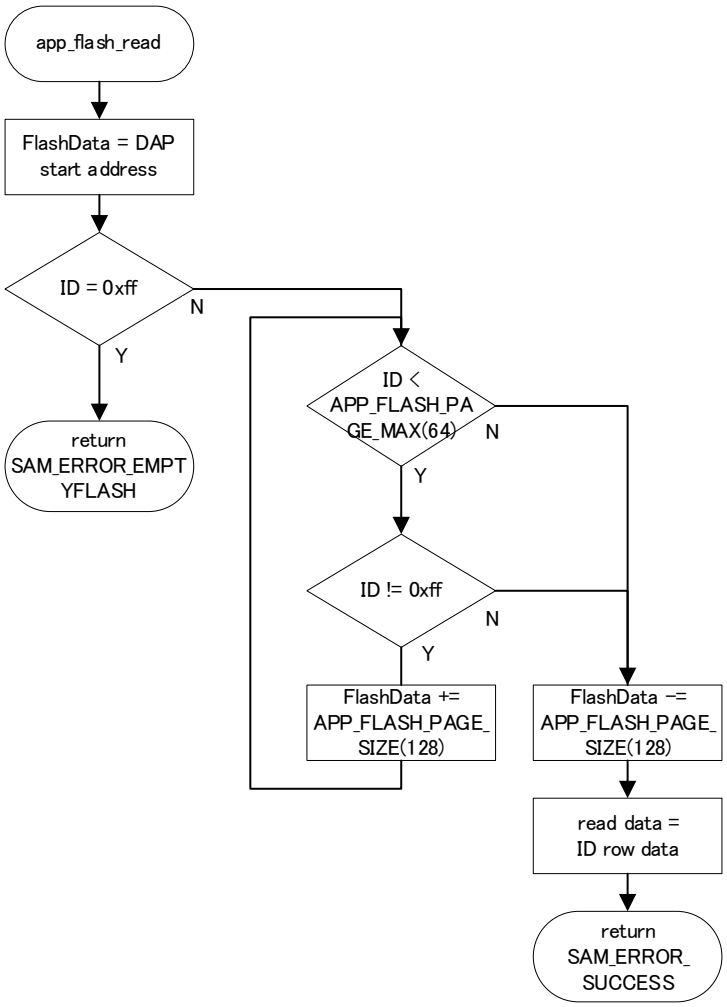

a
a1
[PROC_COMMAND_PARSE]
stmcmp (&input.data[0] ,"write")
return value = 0
N
Y
proc = PROC_WRITE_EXEC
num = input.wp − 7(*2)
*2:
7 = "write"+ [space] + '¥n'
num > CFG_DATA_LENGTH(10)
N
Y
proc = PROC_PARAM_ERROR
write_data[] = input.data[]
printf = "¥n"
return value = −1
N
Y
proc = PROC_ERROR
printf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

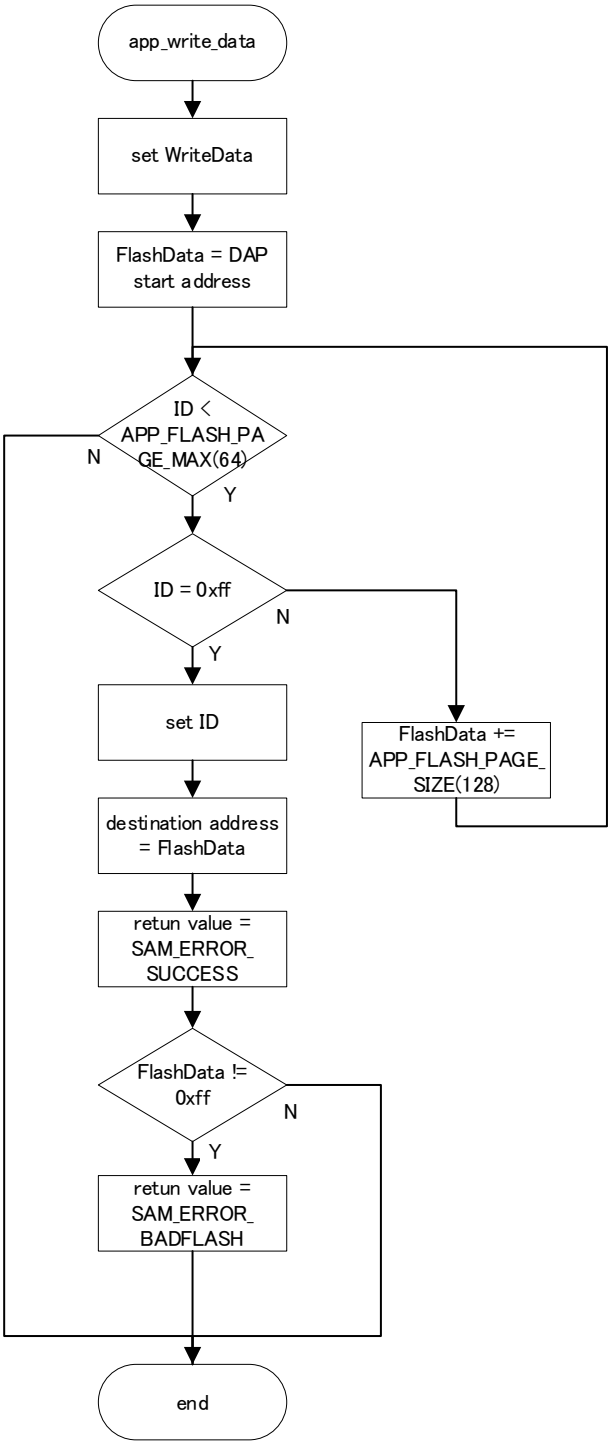
11.2.3. Get SAM Status



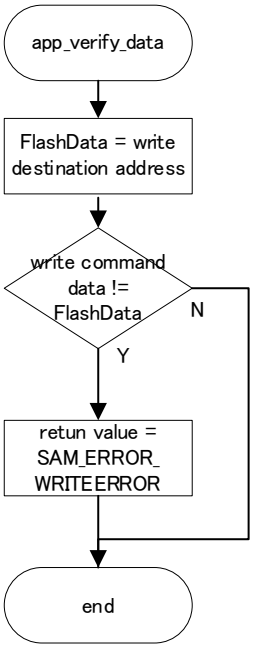
11.2.4. Flash Data Read



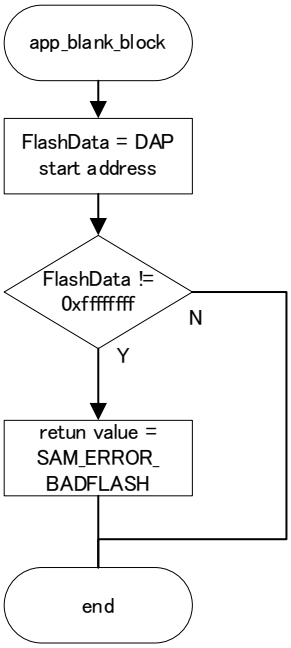
11.2.5. Flash Data Write



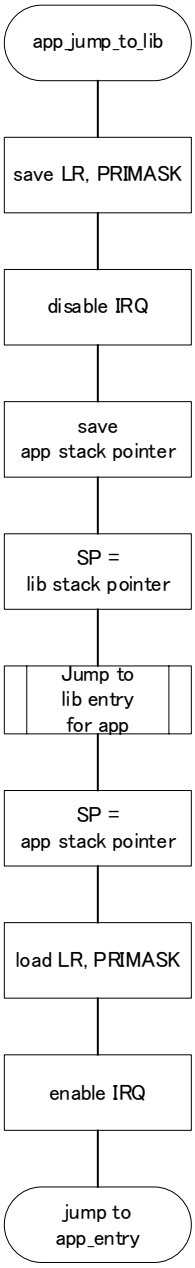
11.2.6. Flash Write Data Verfication



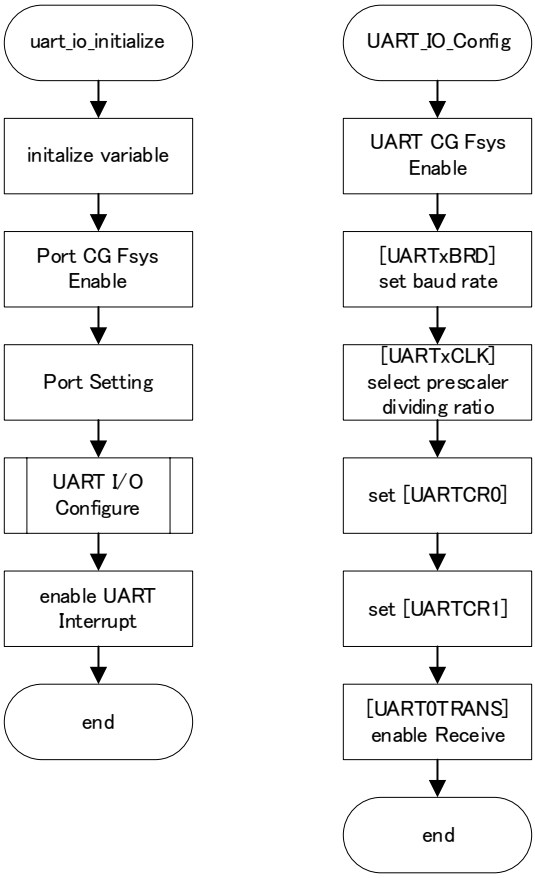
11.2.7. DAP Blank Check



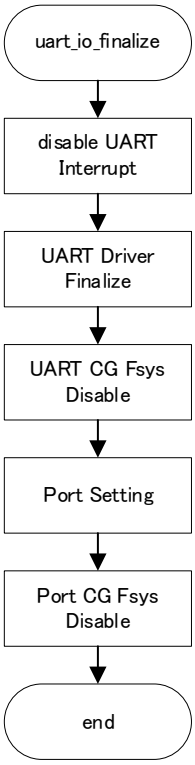
11.2.8. Entry from the App State to the Lib State



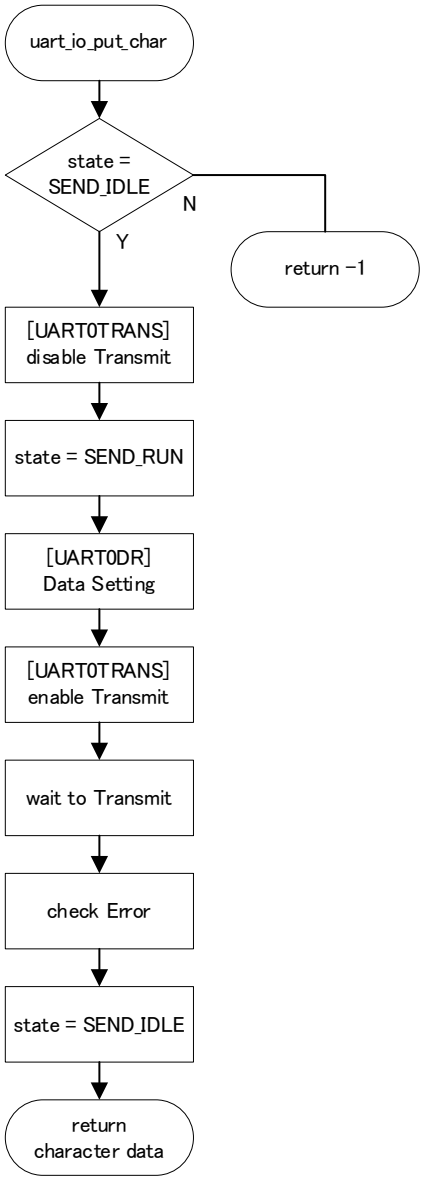
11.2.9. UART Initialization



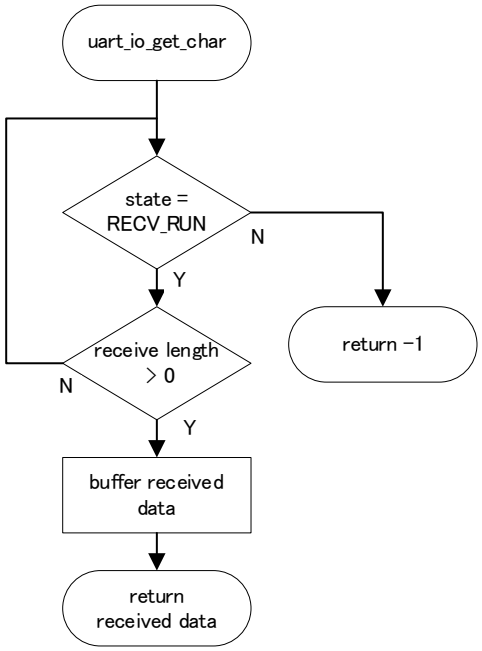
11.2.10. UART Finalization



11.2.11. Transmission of Character Data via UART

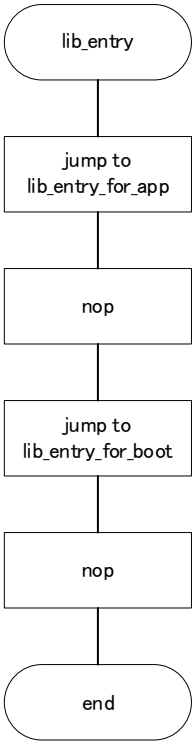


11.2.12. Reception of Character Data via UART

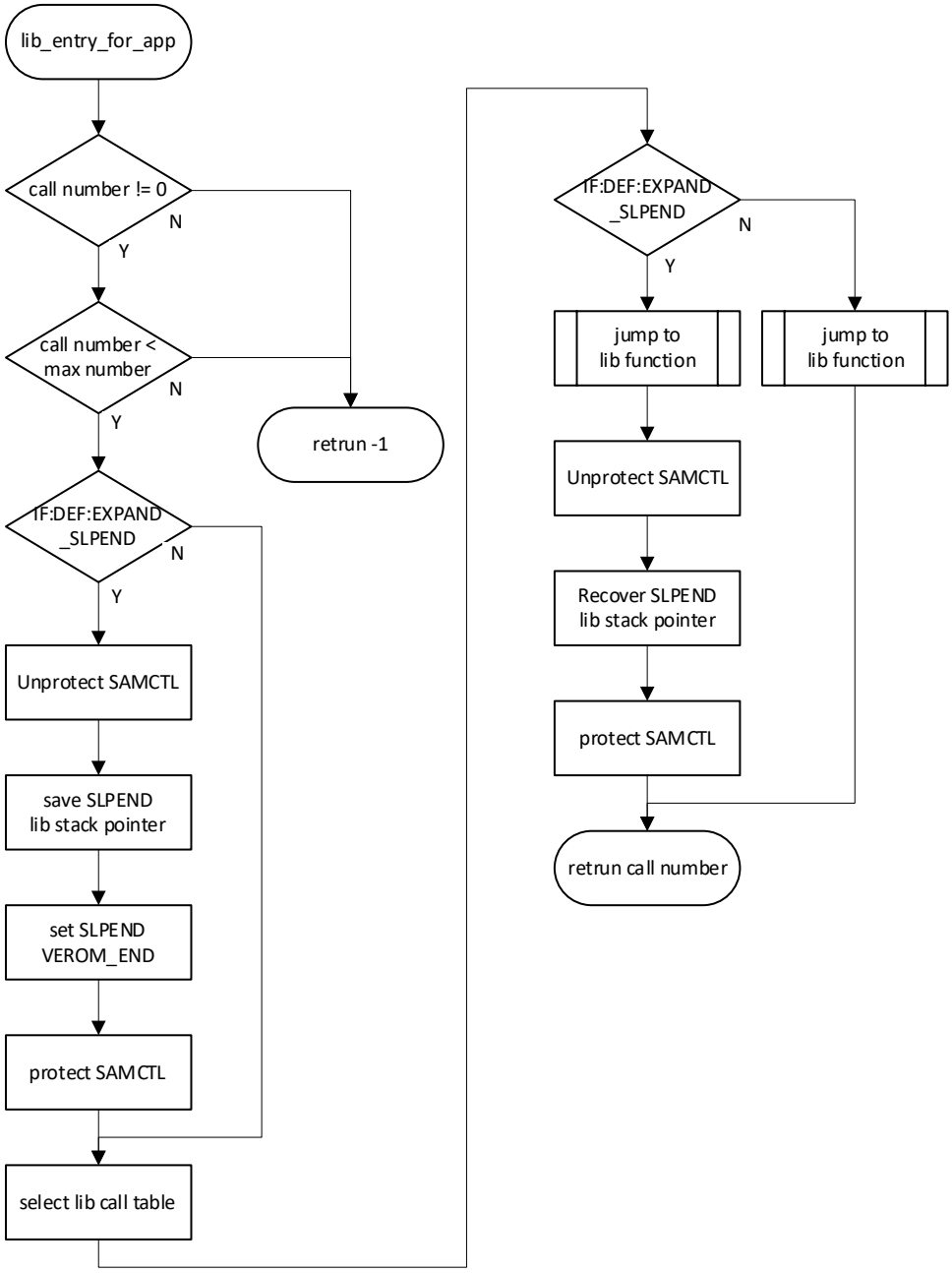


11.3. Lib State

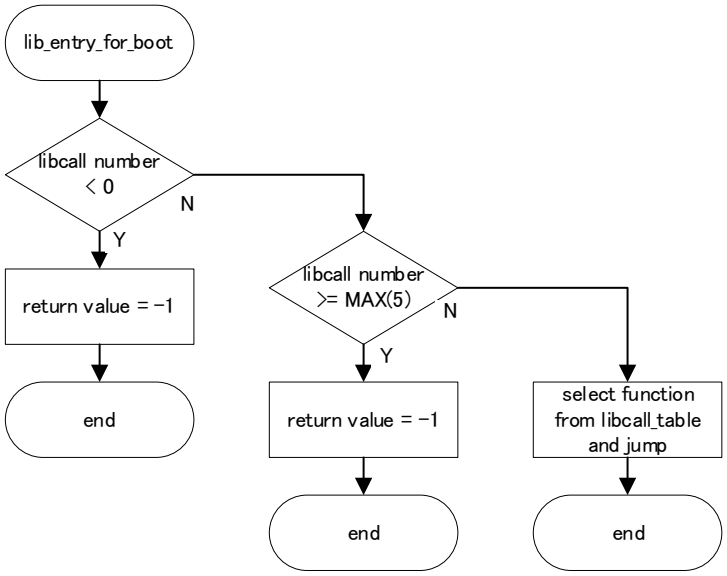
11.3.1. Lib State Entry Function



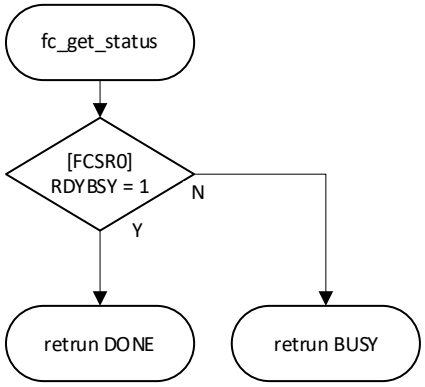
11.3.2. Lib State Entry Function from the App State



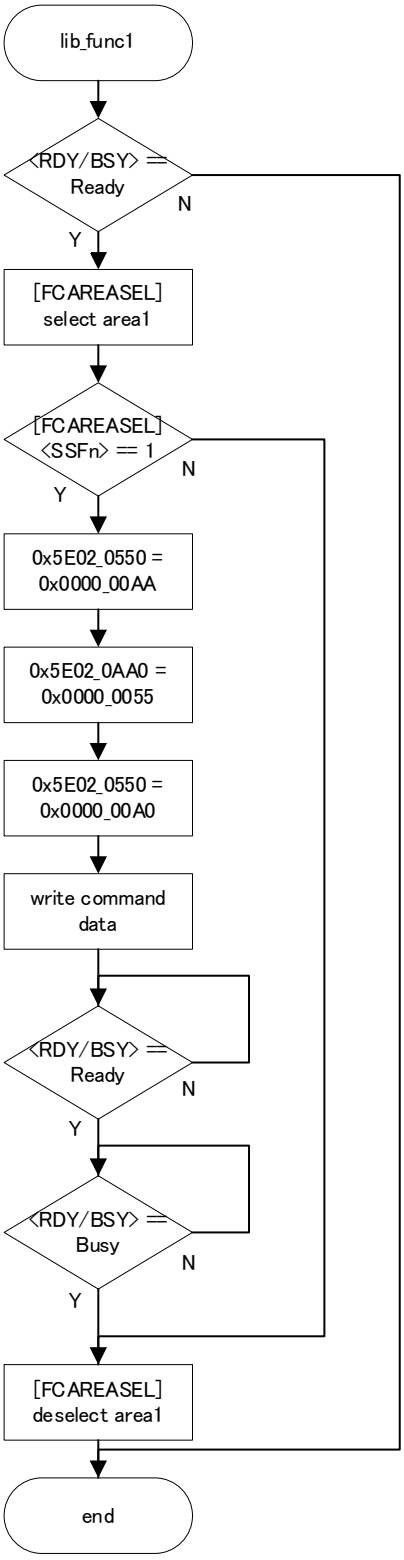
11.3.3. Lib Function Calls from the Boot State



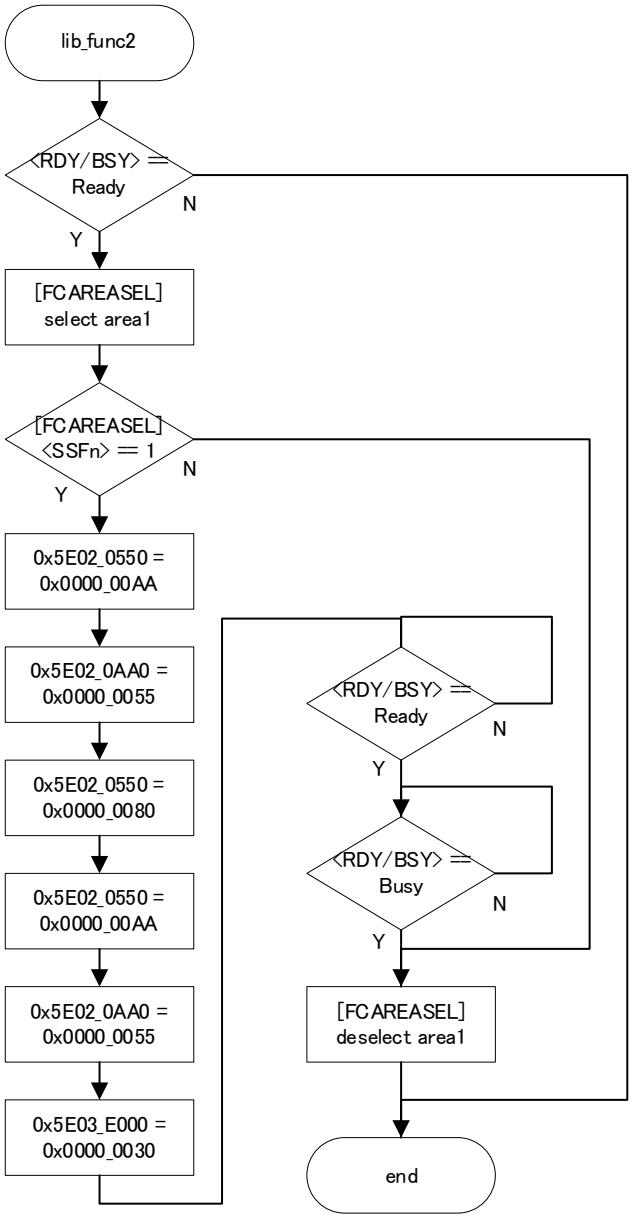
11.3.4. Ready/Busy Status Check During Automatic Operation



11.3.5. Flash Page Write

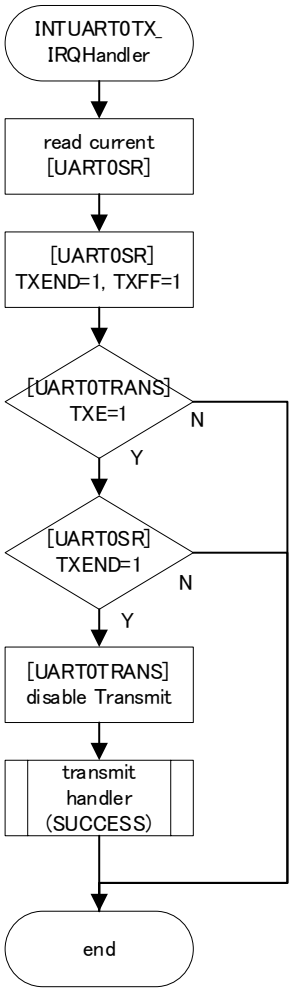


11.3.6. Flash Block Erase

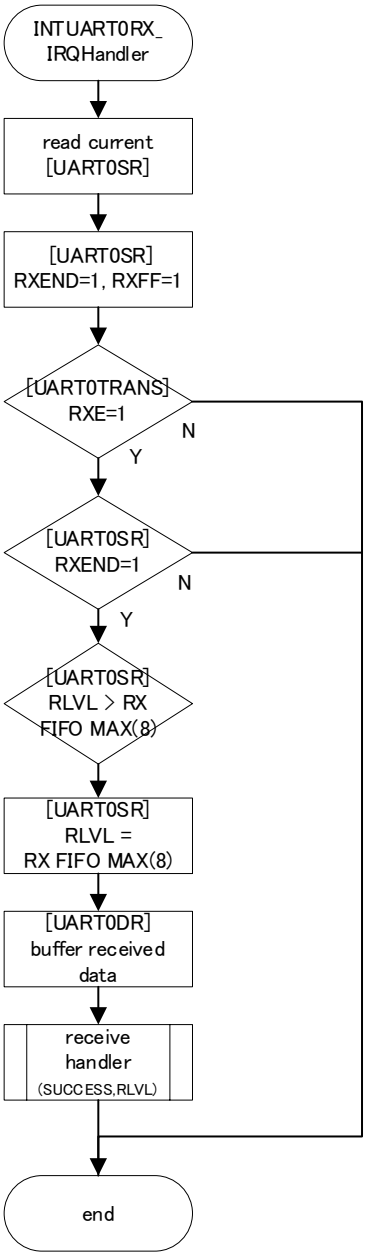


11.4. Interrupt

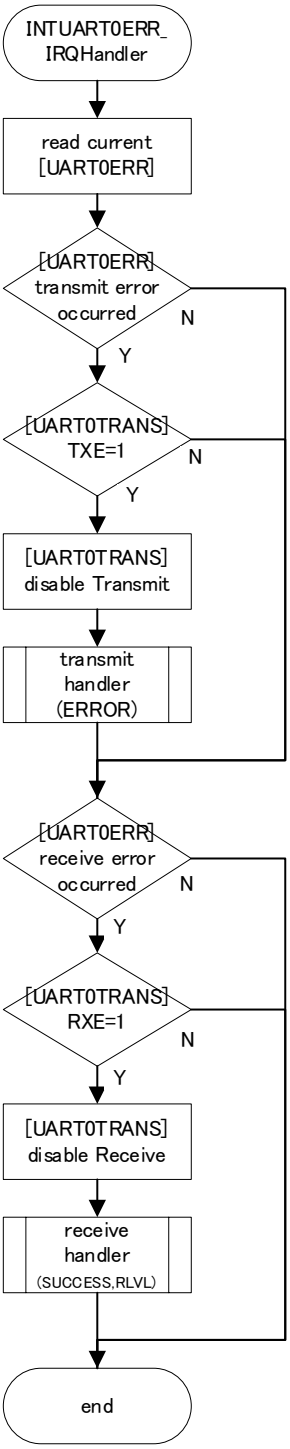
11.4.1. UART Transmit IRQ Handler



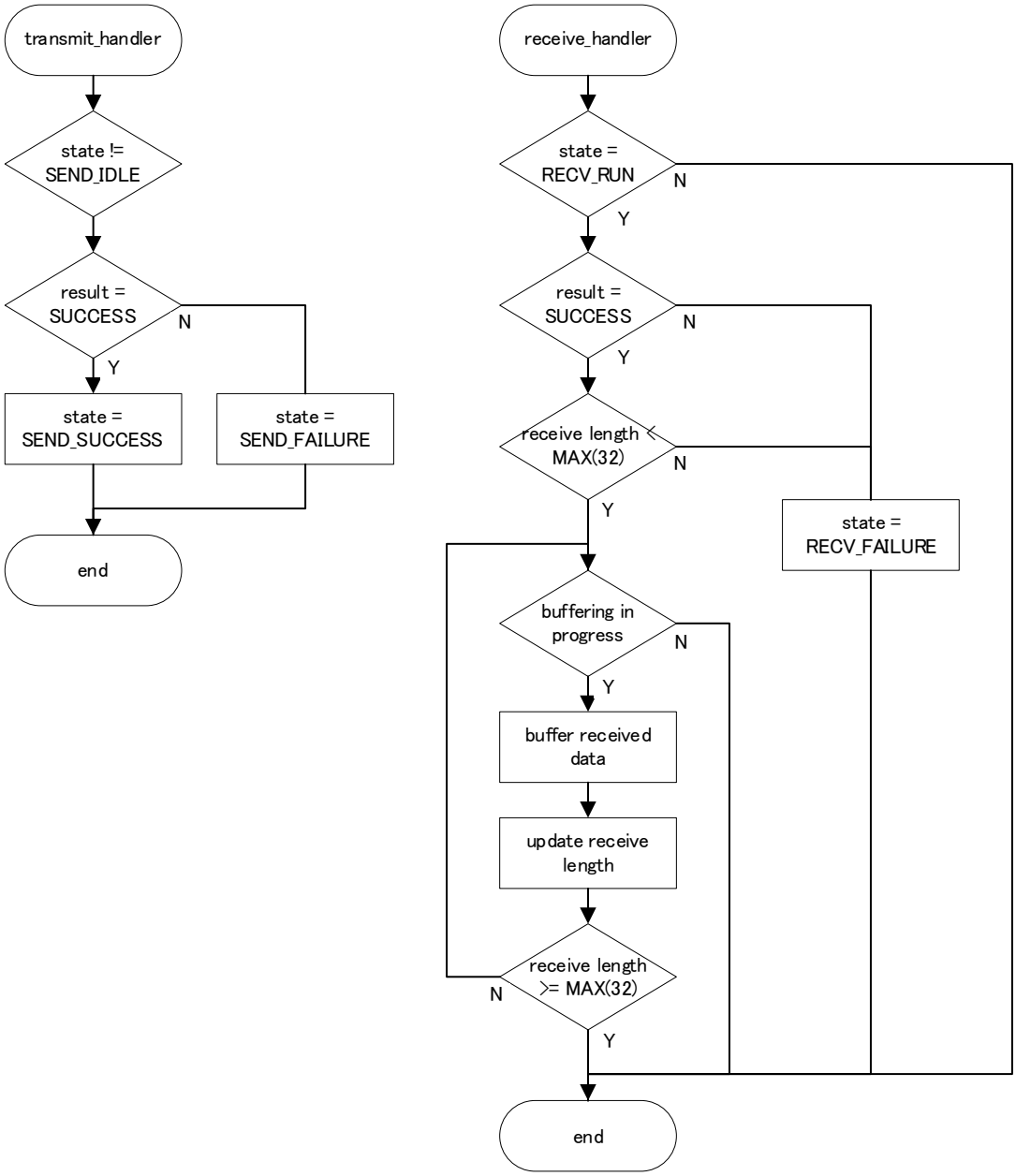
11.4.2. UART Receive IRQ Handler



11.4.3. UART Error IRQ Handler



11.4.4. UART Transmit/Receive Handler



12. Revision History

Revision	Date	Description
1.0	2026-07-30	First release

RESTRICTIONS ON PRODUCT USE

Toshiba Corporation and its subsidiaries and affiliates are collectively referred to as “TOSHIBA”.
Hardware, software and systems described in this document are collectively referred to as “Product”.

- TOSHIBA reserves the right to make changes to the information in this document and related Product without notice.
- This document and any information herein may not be reproduced without prior written permission from TOSHIBA. Even with TOSHIBA's written permission, reproduction is permissible only if reproduction is without alteration/omission.
- Though TOSHIBA works continually to improve Product's quality and reliability, Product can malfunction or fail. Customers are responsible for complying with safety standards and for providing adequate designs and safeguards for their hardware, software and systems which minimize risk and avoid situations in which a malfunction or failure of Product could cause loss of human life, bodily injury or damage to property, including data loss or corruption. Before customers use the Product, create designs including the Product, or incorporate the Product into their own applications, customers must also refer to and comply with (a) the latest versions of all relevant TOSHIBA information, including without limitation, this document, the specifications, the data sheets and application notes for Product and the precautions and conditions set forth in the "TOSHIBA Semiconductor Reliability Handbook" and (b) the instructions for the application with which the Product will be used with or for. Customers are solely responsible for all aspects of their own product design or applications, including but not limited to (a) determining the appropriateness of the use of this Product in such design or applications; (b) evaluating and determining the applicability of any information contained in this document, or in charts, diagrams, programs, algorithms, sample application circuits, or any other referenced documents; and (c) validating all operating parameters for such designs and applications. **TOSHIBA ASSUMES NO LIABILITY FOR CUSTOMERS' PRODUCT DESIGN OR APPLICATIONS.**
- **PRODUCT IS NEITHER INTENDED NOR WARRANTED FOR USE IN EQUIPMENTS OR SYSTEMS THAT REQUIRE EXTRAORDINARILY HIGH LEVELS OF QUALITY AND/OR RELIABILITY, AND/OR A MALFUNCTION OR FAILURE OF WHICH MAY CAUSE LOSS OF HUMAN LIFE, BODILY INJURY, SERIOUS PROPERTY DAMAGE AND/OR SERIOUS PUBLIC IMPACT ("UNINTENDED USE").** Except for specific applications as expressly stated in this document, Unintended Use includes, without limitation, equipment used in nuclear facilities, equipment used in the aerospace industry, Class 3 medical devices, equipment used for automobiles, and military vehicles and munitions. **IF YOU USE PRODUCT FOR UNINTENDED USE, TOSHIBA ASSUMES NO LIABILITY FOR PRODUCT.** For details, please contact your TOSHIBA sales representative or contact us via our website.
- Product shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable laws or regulations.
- The information contained herein is presented only as guidance for Product use. No responsibility is assumed by TOSHIBA for any infringement of patents or any other intellectual property rights of third parties that may result from the use of Product. No license to any intellectual property right is granted by this document, whether express or implied, by estoppel or otherwise.
- **ABSENT A WRITTEN SIGNED AGREEMENT, EXCEPT AS PROVIDED IN THE RELEVANT TERMS AND CONDITIONS OF SALE FOR PRODUCT, AND TO THE MAXIMUM EXTENT ALLOWABLE BY LAW, TOSHIBA (1) ASSUMES NO LIABILITY WHATSOEVER, INCLUDING WITHOUT LIMITATION, INDIRECT, CONSEQUENTIAL, SPECIAL, OR INCIDENTAL DAMAGES OR LOSS, INCLUDING WITHOUT LIMITATION, LOSS OF PROFITS, LOSS OF OPPORTUNITIES, BUSINESS INTERRUPTION AND LOSS OF DATA, AND (2) DISCLAIMS ANY AND ALL EXPRESS OR IMPLIED WARRANTIES AND CONDITIONS RELATED TO SALE, USE OF PRODUCT, OR INFORMATION, INCLUDING WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, ACCURACY OF INFORMATION, OR NONINFRINGEMENT.**
- Do not use or otherwise make available Product or related software or technology for any military purposes, including without limitation, for the design, development, use, stockpiling or manufacturing of nuclear, chemical, or biological weapons or missile technology products (mass destruction weapons). Product and related software and technology may be controlled under the applicable export laws and regulations including, without limitation, the Japanese Foreign Exchange and Foreign Trade Law and the U.S. Export Administration Regulations. Export and re-export of Product or related software or technology are strictly prohibited except in compliance with all applicable export laws and regulations.
- Please contact your TOSHIBA sales representative for details as to environmental matters such as the RoHS compatibility of Product. Please use Product in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. **TOSHIBA ASSUMES NO LIABILITY FOR DAMAGES OR LOSSES OCCURRING AS A RESULT OF NONCOMPLIANCE WITH APPLICABLE LAWS AND REGULATIONS.**