To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

SH7216 Group

Improving Data Flash Programming Efficiency

Introduction

This application note uses sample data flash driver software to present usage methods, setting procedures, and usage examples related to data management.

Note that although the sample tasks and applications presented in this application note have been verified to work as intended, they should be checked in the actual operating environment before being put into actual use.

Target Device

SH7216

Contents

1. Specifications of Sample Data Flash Driver Software

The sample data flash driver software (referred to below as the “data flash driver” or simply “driver”) is a sample device driver program for storing and managing data in the on-chip data flash of the SH7216.

Data can be updated, read, etc., simply by calling user API functions of the data flash driver (referred to below as “data flash driver functions” or “driver functions”).

1.1 Functions

1. Data can be updated or read simply by calling a data flash driver function. With the data flash driver, it is easy to construct a data management system that uses the data flash.
2. The data flash driver assigns a logical data number to each unit of data. It is a logical block type device driver capable of updating and reading data.
3. The data flash driver can manage in the data flash multiple data units of different sizes ranging from 0 to 256 bytes.
4. Instead of erasing and then programming the data flash each time the data is updated, the data flash driver programs available writable areas within the target block, skipping the non-writable areas as necessary. By using the data flash driver it is possible to perform a higher number of data updates than the maximum number of reprogramming (erase) cycles supported by the device.
5. When no more writable areas remain within a block, the data flash driver copies data to another block within the block group to create a writable area. This process is handled automatically by the data flash driver as needed when a data flash driver function is called.
6. If a system shutdown occurs while data is being updated using the data flash driver, the data flash driver restores the data to its state before the update the next time driver initialization takes place.
7. If a system shutdown occurs while data is being erased using the data flash driver, the data flash driver can detect that the erase operation was interrupted the next time driver initialization takes place.

1.2 Usage Notes

1.2.1 Control Exclusive of Other Programs

The user API functions of the data flash driver manipulate control registers and issue commands for the dedicated sequencer (FCU) of the flash memory and data flash. These processes are designed to be performed by the data flash driver only. It is therefore necessary to ensure that other programs do not perform any of the following operations while the data flash driver is operating.

Processes Requiring Exclusive Control by the Data Flash Driver

1. Write access to the FCU control registers
2. Issuing of commands to the FCU
3. Direct access to the data flash address area

1.2.2 Wait/Timeout Detection Timer Module

The driver uses the MCU's on-chip multifunction timer pulse unit (MTU2) for wait and timeout detection processing, so the same channel of this module may not be used by other programs. Make sure that no changes are made to registers related to the operation of the module or to related bits in common registers.

1.3 File Structure

The file structure of the data flash driver is shown below.

Table 1.1 File Structure of Data Flash Driver

File Name	Description
DF_user.h	The user API function header file. This file must be included in order to use the driver. It specifies the clock setting values, data counts, and API function interrupt mask levels that are required when using the data flash driver. Make settings in accordance with the system specifications. This file also defines the prototypes, return values, and arguments of the user API functions.
DF_user.c	This file specifies the data size of each data number for the data flash driver and the area (block group) where data is stored. Make settings in accordance with the system specifications.
DF_common.h	The data flash information setting file. This file must be included in order to use the driver. It contains setting value definitions used by the data flash driver.
DF_drv.h	The data flash driver internal statue setting definition file. This file must be included in order to use the driver.
DF_drv.c	The C source file for the user API functions. The data flash driver comprises functions such as the following: • Initialization function • Format function • Valid data read function • Data update function • Block erase function
DF_control.h	The definition file for FCU control register setting values, FCU control commands, and wait/timeout detection timer module settings. This file must be included in order to use the driver.
DF_control.c	The FCU control function C source file.
DF_typedef.h	The type declaration file for the data flash driver.

1.4 Internal Block Structure of Data Flash

As shown in figure 1.1, the data flash is structured internally as groups of blocks, and the data flash driver manages the states of the blocks within each group.

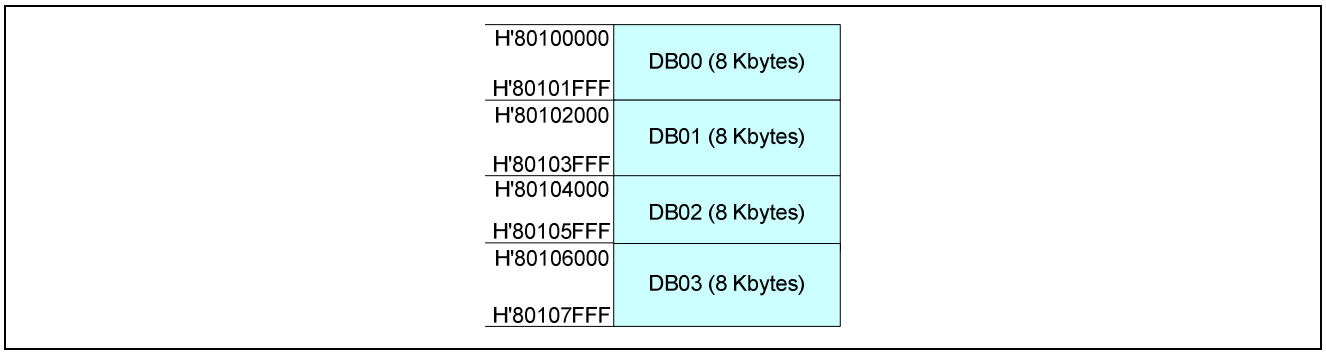

Figure 1.1 Data Flash Block Structure

1.5 Internal Data Structure of Block

The data flash driver constructs a data structure as shown in figure 1.2 for managing data. Each block is divided into a block management information area, data management information areas, and data areas. There is a one-to-one correspondence between each data management information area and a data area.

Block Internal Offset	Contents	Size	
0x000 0x3FF	Data area 6	1,024 bytes	Data area
0x400 0x7FF	Data area 5	1,024 bytes	
0x800 0xBFF	Data area 4	1,024 bytes	
0xC00 0xFFF	Data area 3	1,024 bytes	
0x1000 0x13FF	Data area 2	1,024 bytes	
0x1400 0x17FF	Data area 1	1,024 bytes	
0x1800 0x1BFF	Data area 0	1,024 bytes	
0x1C00 0x1C1F	Data management information area 6	32 bytes	Data management information area
0x1C20 0x1C3F	Data management information area 5	32 bytes	
0x1C40 0x1C5F	Data management information area 4	32 bytes	
0x1C60 0x1C7F	Data management information area 3	32 bytes	
0x1C80 0x1C9F	Data management information area 2	32 bytes	
0x1CA0 0x1CBF	Data management information area 1	32 bytes	
0x1CC0 0x1CDF	Data management information area 0	32 bytes	
0x1CE0 0x1CFF	Block management information area	32 bytes	Block management information area

Figure 1.2 Internal Data Structure of Data Flash Block

1.6 Block Management Information

As shown in figure 1.3, each block management information area is divided into a block ID, a data reclaim end flag, a block erase start flag, and a block erase end flag.

At driver initialization, the block management information is used to confirm the state of the block.

Block Internal Offset	Contents	Size
0x1CE0 0x1CE7	Block erase end flag	8 bytes
0x1CE8 0x1CEF	Block erase start flag	8 bytes
0x1CF0 0x1CF7	Data reclaim end flag	8 bytes
0x1CF8 0x1CFF	Block ID	8 bytes

Figure 1.3 Internal Data Structure of Block Management Information Area

1.6.1 Block ID

This information shows that the block is in a valid state (a state in which data can be written to it).

The block ID is the first information to be programmed in a blank block (a block that is entirely blank). Therefore, when the driver discovers a block in which the block ID area is blank and other areas are not black, it determines that the entirety of the data flash has not been formatted by the driver and sends a format request to the user application as a return value.

In addition, when the driver discovers a blank block, it determines from the state of the block erase start flag of the paired block within the same block group that the block ID is unprogrammed due to an abnormal system shutdown while the block was in the erase processing state or that the data flash has not been formatted. (See table 1.2 for a listing of block state determinations.)

1.6.2 Data Reclaim End Flag

This information shows whether or not all valid data in the block (valid data set in DF_user.c for storage in the same block group) has been stored.

1.6.3 Block Erase Start Flag/Block Erase End Flag

This information shows the erase processing state of the paired block within the same block group. It is used, together with the block ID (1.6.1) and data reclaim end flag (1.6.2), to determine the block state as shown in table 1.2.

1.6.4 Block Erase Outline Flowchart

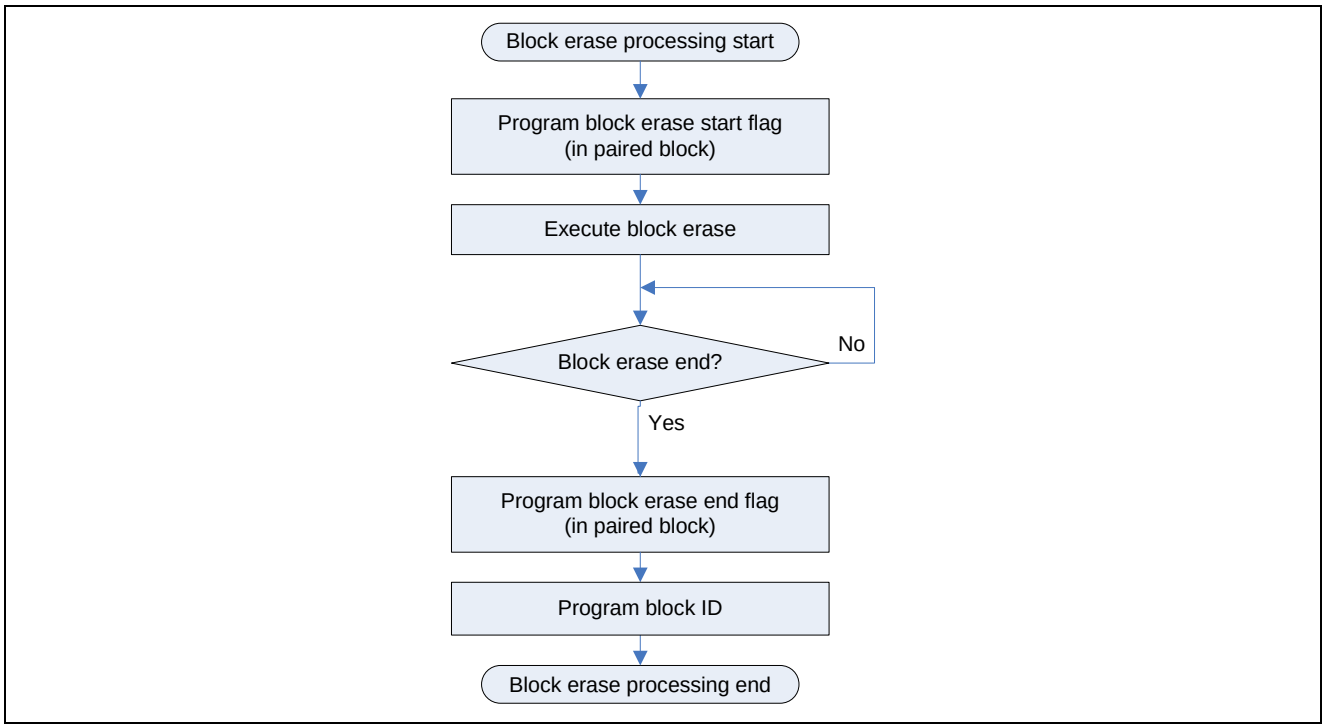

Figure 1.4 Block Erase Processing Outline Flowchart

1.6.5 Block State Determination

At initialization, the driver performs a blank check on all blocks and the management information areas of each block. It then determines the state of each block according to table 1.2.

Table 1.2 Block States

Block	Blank	Non-blank						
Block ID	—	Blank	Non-blank					
Data reclaim end flag	—	—	Blank		Non-blank			
Erase start flag for paired block*1	—	—	Non-blank	Blank	Blank		Non-blank	
Erase end flag for paired block*1	—	—			Non-blank	Blank	Blank	Non-blank
Block state	Block ID unprogrammed	Unformatted		Block awaiting data update	Unformatted	Block for data update		
State of paired block	*2	*3		Block for data update	*3	Block awaiting erase	Block with erase in progress	*2

Notes: 1. Erase start and erase end flags for each block are stored in the paired block.
2. The block state is determined from the block management information in the paired block.
3. The block state is determined from the block management information in the paired block, but a format request for the entirety of the data flash is sent as return value if an unformatted block is discovered in the data flash.

1.7 Data Management Information

This information shows the data storage state. There is a one-to-one correspondence between each of the areas where this information is stored and a data area. Valid data management information indicates that valid data is stored in the corresponding data storage area. (For information on the structure, see 1.5, Internal Data Structure of Block, and 1.8, Searching for Valid Data.)

The data management information comprises a data update start flag and data update end flag. Each data area must have valid data management information corresponding to it.

1.7.1 Data Update Start Flag

This is the first information that is programmed when the data management information is updated. When the area of this flag is in the non-blank state, the driver determines that data management information exists in this position. In addition, the data update state is determined according to the combination of this flag and the data update end flag, which is described below.

This information includes the data number and data checksum.

(1) Data Number

This is a logical number assigned by the driver and used for data management.

These numbers are allocated in order, starting from 0, according to the order of the members of the array (DF_DATA_SIZE_GROUP) in DF_user.c that specifies the data size and storage destination block group of each data unit. The data number is managed as data with a size of 1 byte.

(2) Data Checksum

This checksum value is generated by the driver when data is updated. Each byte of the update data is added together and the lowest byte of the resulting value becomes the checksum value, which is used when searching for valid data during driver initialization and for error detection when checking data already written within the block when reclaiming data.

This checksum value is not used for data checking when reading or reclaiming data by calling the valid data read function.

1.7.2 Data Update End Flag

This information is programmed when a data update ends. When the area of this flag is in the non-blank state, the driver determines that programming of data to the corresponding data storage address has completed successfully. The data management information is updated to an address value that becomes progressively smaller with each update. When a new update of the data management information completes, the old data management information becomes logically invalid.

1.7.3 Data Update Outline Flowchart

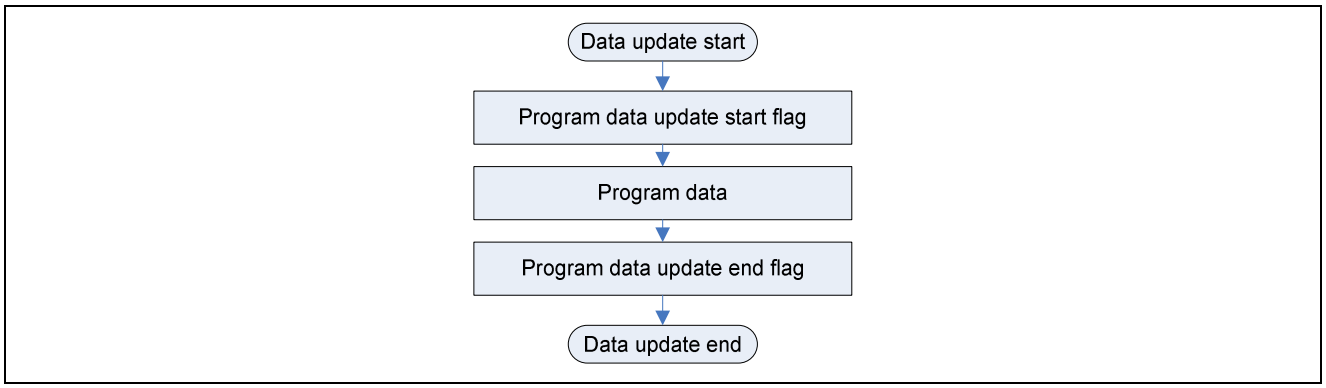

Figure 1.5 Data Update Processing Outline Flowchart

1.7.4 Determining the Data Update State

The driver determines the data update state according to the states of the flags in the data management information, as shown in table 1.3.

Table 1.3 Determining the Data Update State

Data update start flag	Blank		Non-blank	
Data update end flag	Blank	Non-blank	Blank	Non-blank
Data update state	Unused	Invalid	Data update in progress	Data update completed
Description of state	The information area and the corresponding data storage area have not been used, so they are available for use in the next or a later update.	The data management information area is in an illegal state, so the data in the information area and in the corresponding data storage area is invalid.	Either the data management information or the corresponding data storage area is being programmed, so the data in the information area and in the corresponding data storage area is invalid.	A data update completed successfully, and the data in the corresponding data storage area is valid.

1.8 Searching for Valid Data

During initialization, the driver searches for valid data management information for each data number. The driver determines as valid the data management information in the data update completed state (according to table 1.3, Determining the Data Update State) that is stored at the smallest address value. The driver maintains in internal RAM the data storage address corresponding to the valid data management information for each data number. When performing a data read, the driver reads data from the data flash based on the data storage addresses stored in RAM.

1.9 Illustration of Block States and Stored Data

This section illustrates block states and stored data, using the sample driver settings shown in figure 1.6 (in DF_user.h and DF_user.c) and taking data flash block group A (block 0 and block 1) as an example.

```
/* Set data count */
#define DF_DATA_ID_NUM                  (3)             // Data count = 3

/* Set data size and data storage area (block group) for each data number */
const unsigned short DF_DATA_SIZE_GROUP[ DF_DATA_ID_NUM ][2] =
{
/* Data number  0 */            1,              GROUP_A,        // Data size: 1 byte, stored in block group A
/* Data number  1 */            129,            GROUP_A,        // Data size: 129 bytes, stored in block group A
/* Data number  2 */            256,            GROUP_A,        // Data size: 256 bytes, stored in block group A
};
```

Figure 1.6 Sample Driver Settings (in DF_user.h and DF_user.c)

1.9.1 After Formatting

Figure 1.7 illustrates the block states and stored data for block group A immediately after formatting, using the sample driver settings shown in figure 1.6 above. The numbers of the flags in the data management information areas match the numbers of the corresponding data areas.

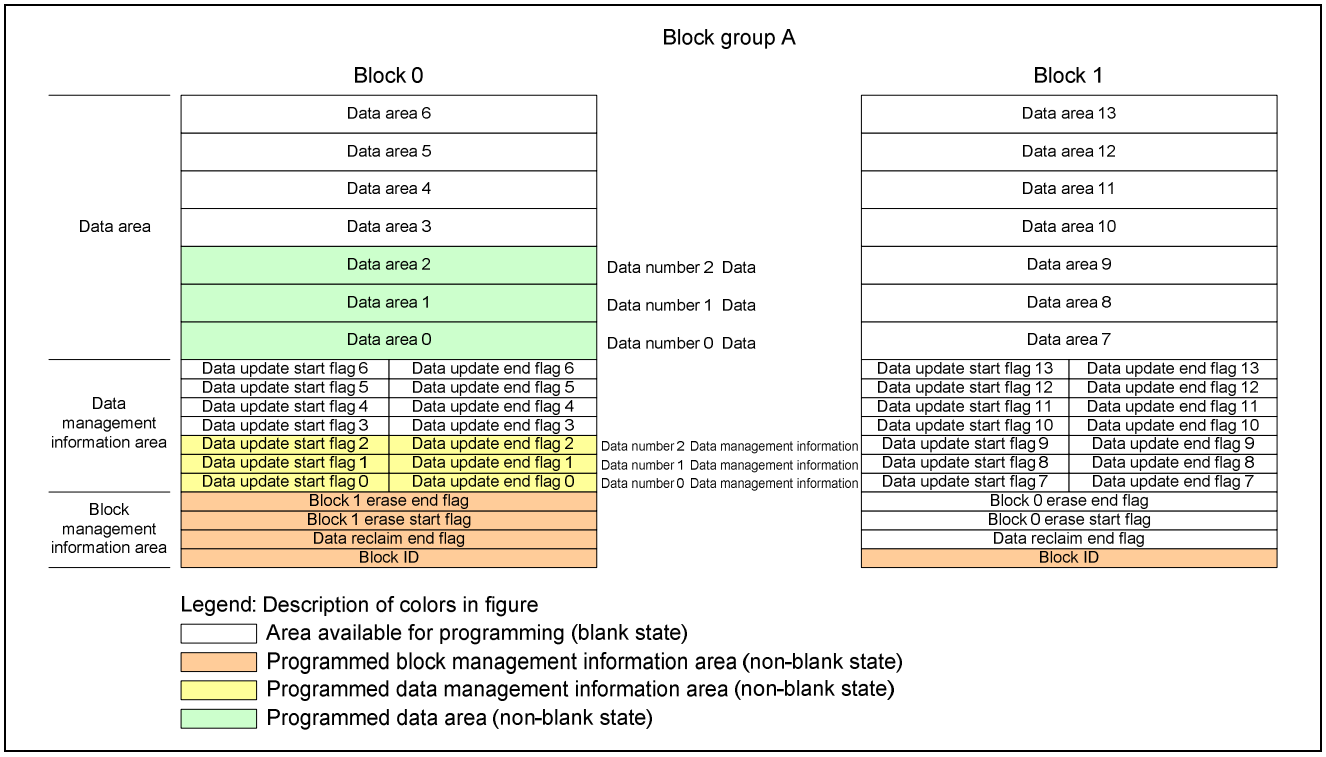

Figure 1.7 Illustration of Block States and Stored Data Immediately after Formatting of Data Flash

When the data flash formatting function is run, the driver programs the value stored in the data buffer area in RAM as the data corresponding to all the data numbers. Therefore, after the data flash formatting function is run the valid data for each data number is an undefined value (the value stored in the data buffer area in RAM immediately before formatting) before the data is updated by the data update processing function.

1.9.2 Data Update – 1 (Normal Operation)

Figures 1.8 to 1.11 illustrate the block states and stored data when from the state shown in figure 1.7, above, the data update function is used to update the data of data number 0.

1. The driver programs the data update start flag of data number 0 in a programmable management information area (figure 1.8 B) in block 0.

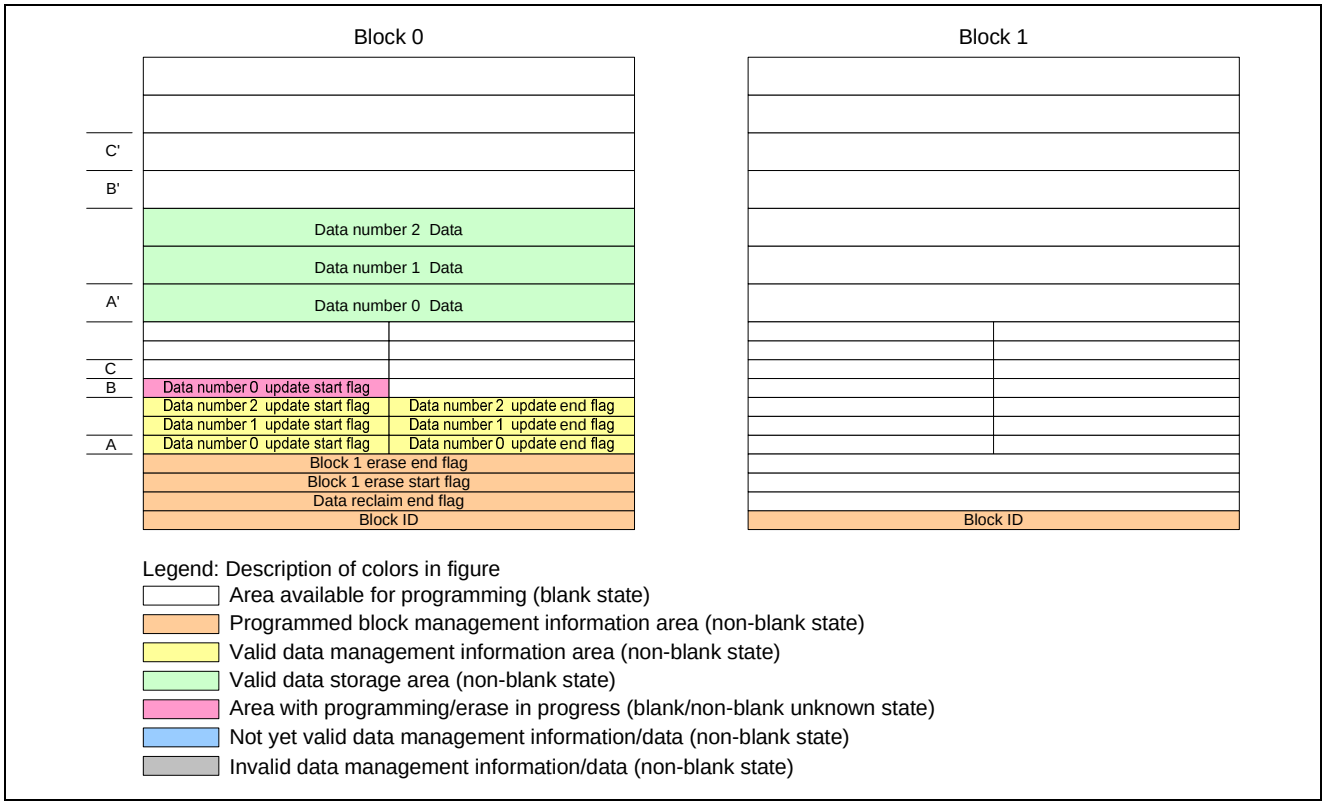

Figure 1.8 Programming of Data Update Start Flag

When a system shutdown occurs in the state shown in figure 1.8, the next time driver initialization takes place the driver considers A to be valid data management information and A' to be valid data for data number 0, and either (a) or (b) below occurs, depending on the blank check result for data update start flag area B.

(a) If data update start flag area B is still in the blank state, data management information area B and data area B' are used when the next data update occurs.

(b) If data update start flag area B is determined to in the non-blank state, data management information area B and data area B' are considered to be invalid areas, and data management information area C and data area C' are used when the next data update occurs.

2. After programming of the data update start flag completes, the data is programmed in the corresponding data area (figure 1.9 B').

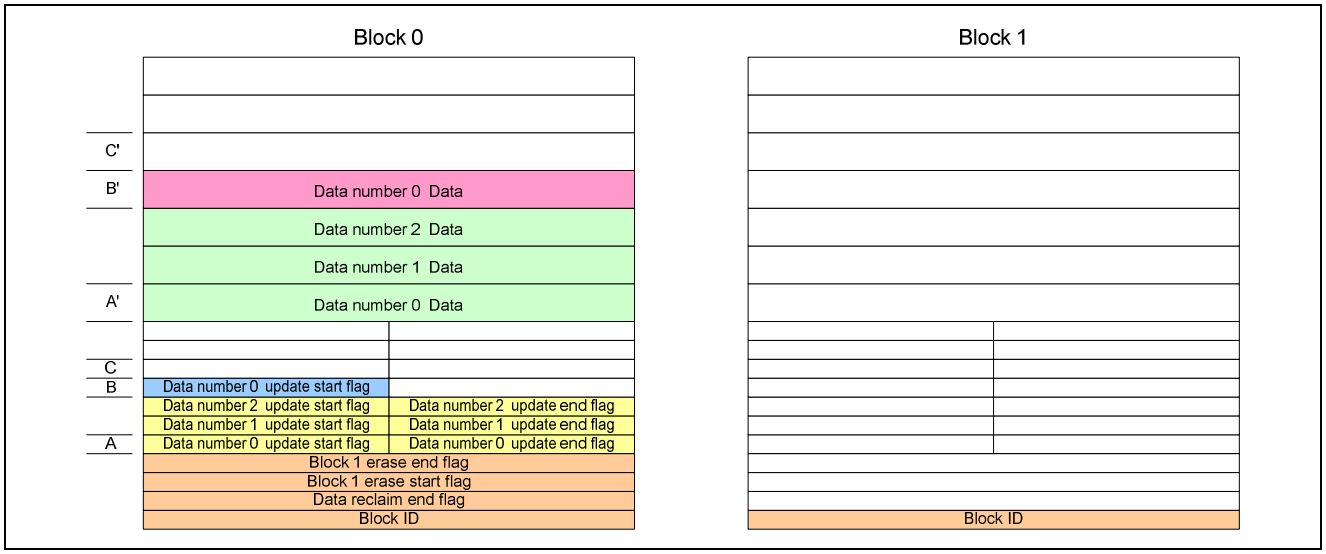

Figure 1.9 Programming of Data

When a system shutdown occurs in the state shown in figure 1.9, the next time driver initialization takes place the driver considers A to be valid data management information and A' to be valid data for data number 0, and data management information area B and data area B' to be invalid areas, so data management information area C and data area C' are used when the next data update occurs.

3. After programming of the data completes, the data update end flag is programmed in the corresponding data management information area (figure 1.10 B).

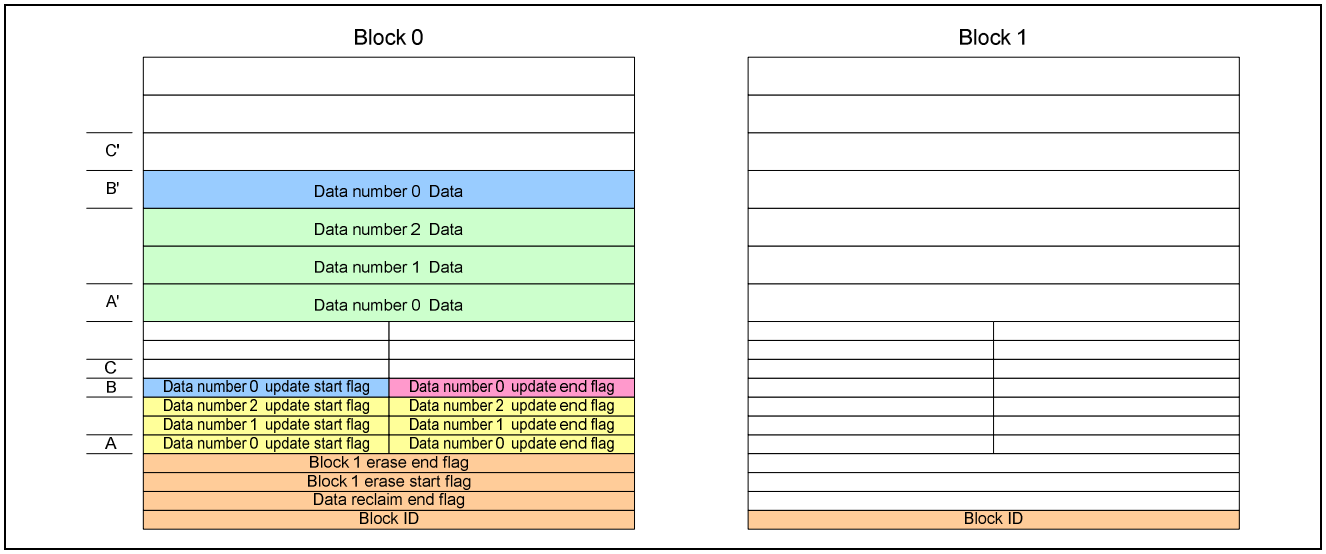

Figure 1.10 Programming of Data Update End Flag

When a system shutdown occurs in the state shown in figure 1.8, the next time driver initialization takes place either (a) or (b) below occurs, depending on the blank check result for data management information area B.

(a) If at least one of the flag areas in data management information area B is still in the blank state, the driver considers A to be valid data management information and A' to be valid data for data number 0, and data management information area B and data area B' to be invalid areas, so data management information area C and data area C' are used when the next data update occurs.

(b) If data management information area B is determined to in the non-blank state, the driver considers B to be valid data management information and B' to be valid data for data number 0, and data management information area A and data area A' to be invalid areas, so data management information area C and data area C' are used when the next data update occurs. Figure 1.11 illustrates the stored data when the data update processing completes.

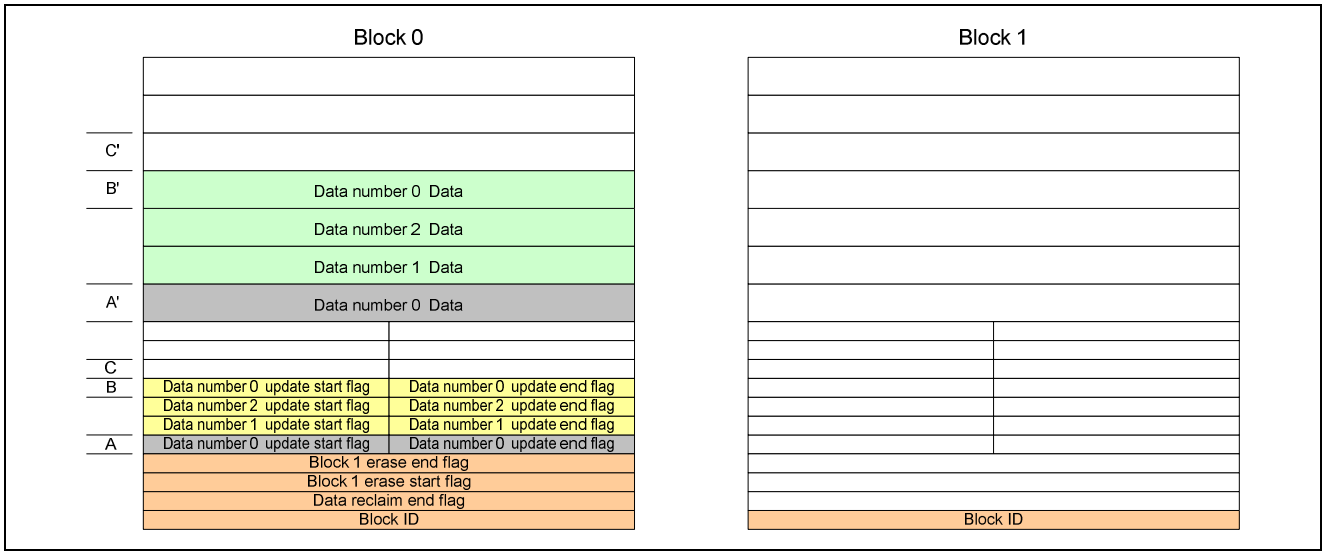

Figure 1.11 Illustration of Stored Data after Data Update Completes

1.9.3 Data Update – 2 (No Areas Available for Programming within Block)

As data updates are performed repeatedly, the block (block 0) used for data updates eventually reaches a state in which no areas are available for programming (figure 1.12).

Figure 1.12 Illustration of Block States and Stored Data with No Areas Available for Programming in Block 0

Figures 1.13 to 1.16 illustrate the block states and stored data when, from the state shown in figure 1.12, the data of data number 1 is updated.

1. The driver programs the data update start flag of data number 1 in a programmable management information area (figure 1.13 D) in block 1.

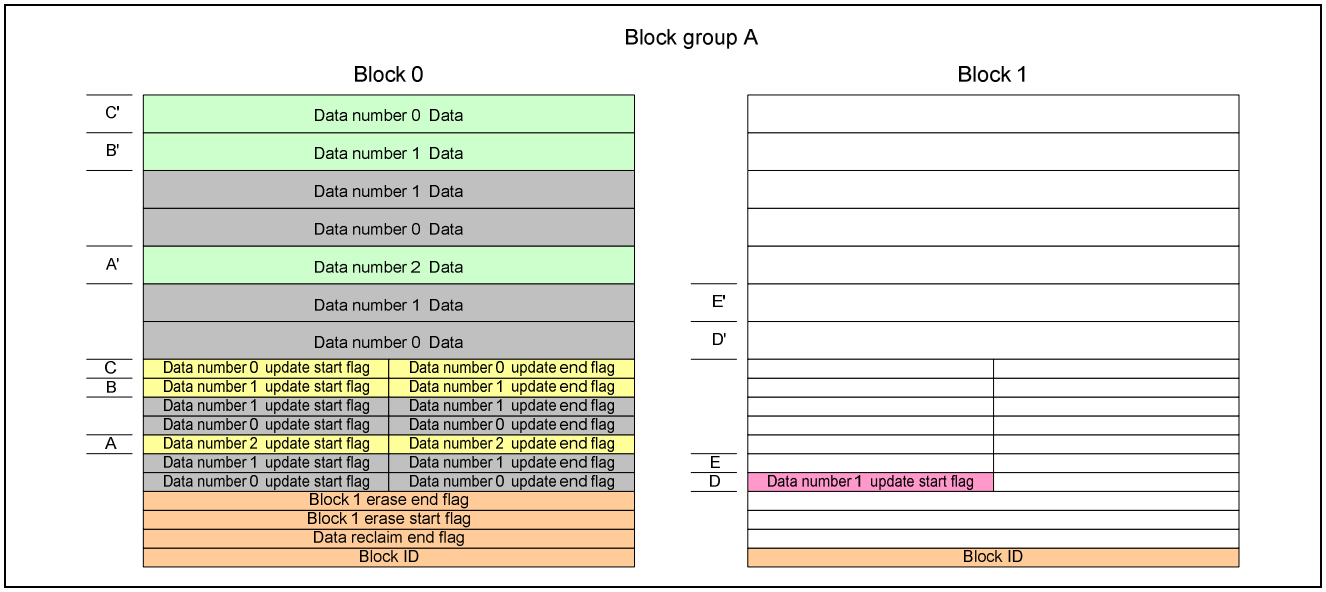

Figure 1.13 Programming of Data Update Start Flag in State with No Areas Available for Programming in Block 0

When a system shutdown occurs in the state shown in figure 1.13, the next time driver initialization takes place the driver considers B in block 0 to be valid data management information and B' to be valid data for data number 1, and either (a) or (b) below occurs, depending on the blank check result for data update start flag area D in block 1.

(a) If data update start flag area D is still in the blank state, data management information area D and data area D' are used when the next data update occurs.

(b) If data update start flag area D is determined to in the non-blank state, data management information area D and data area D' are considered to be invalid areas, and data management information area E and data area E' are used when the next data update occurs.

2. After programming of the data update start flag completes, the data is programmed in the corresponding data area (figure 1.14 D').

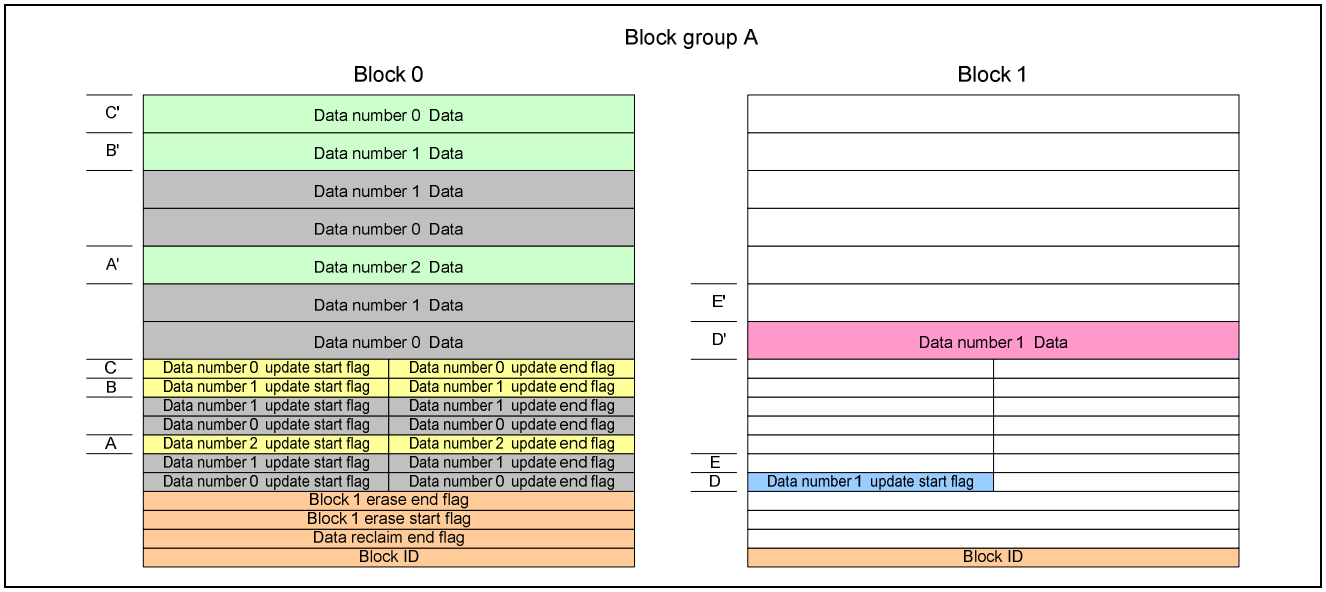

Figure 1.14 Programming of Data in State with No Areas Available for Programming in Block 0

When a system shutdown occurs in the state shown in figure 1.14, the next time driver initialization takes place the driver considers B to be valid data management information and B' to be valid data for data number 1, and data management information area D and data area D' to be invalid areas, so data management information area E and data area E' are used when the next data update occurs.

3. After programming of the data completes, the data update end flag is programmed in the corresponding data management information area (figure 1.15 D).

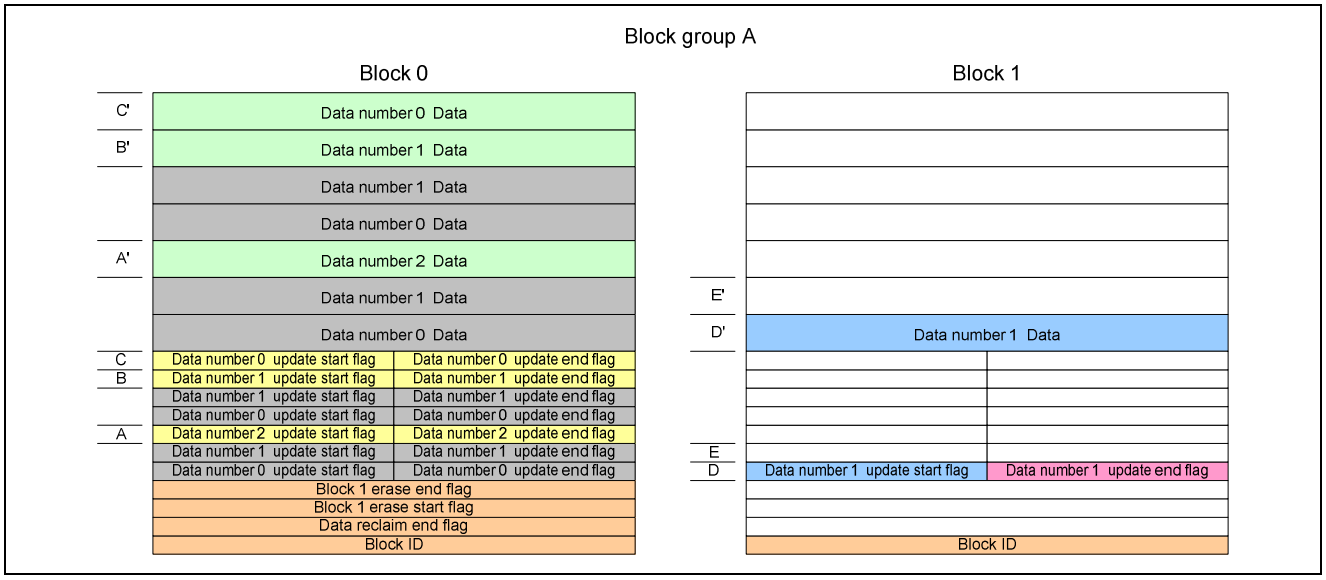

Figure 1.15 Programming of Data Update End Flag in State with No Areas Available for Programming in Block 0

When a system shutdown occurs in the state shown in figure 1.15, the next time driver initialization takes place either (a) or (b) below occurs, depending on the blank check result for data management information area D.

(a) If at least one of the flag areas in data management information area D is still in the blank state, the driver considers B to be valid data management information and B' to be valid data for data number 1, and data management information area D and data area D' to be invalid areas, so data management information area E and data area E' are used when the next data update occurs.

(b) If data management information area D is determined to in the non-blank state, the driver considers D to be valid data management information and D' to be valid data for data number 1, and data management information area B and data area B' to be invalid areas. Figure 1.16 illustrates the stored data when the data update processing completes.

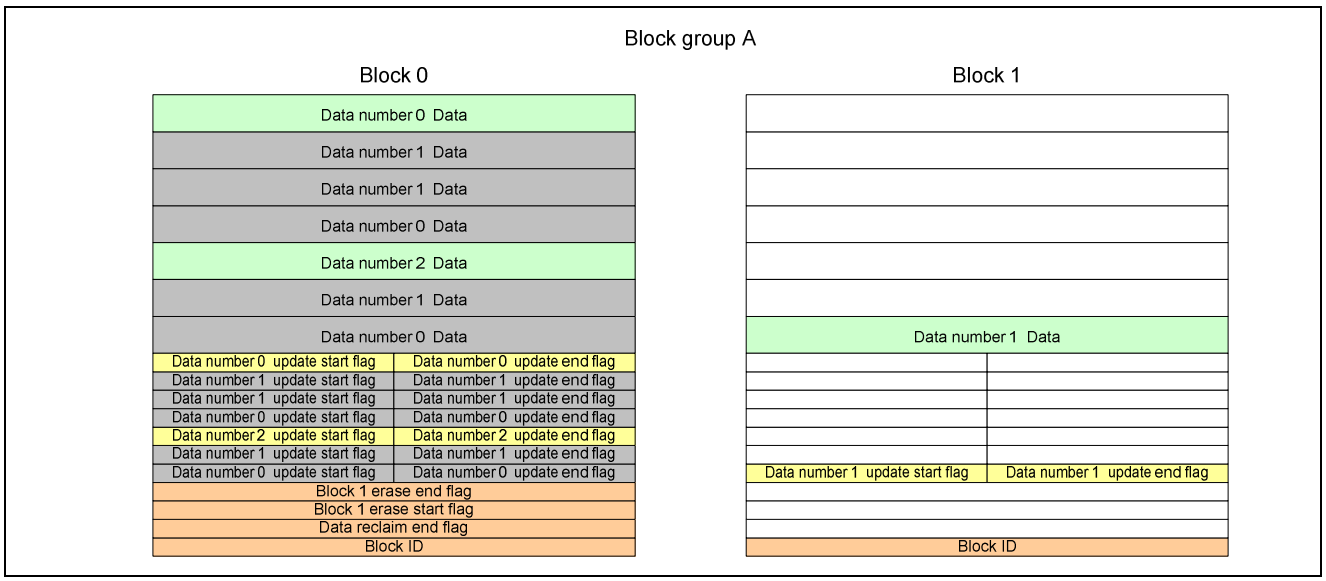

Figure 1.16 Illustration of Stored Data after Data Update Completes in State with No Areas Available for Programming in Block 0

1.9.4 Data Reclaim – 1 (Normal Operation)

After the processing described above (data update when no areas are available for programming in block 0), the driver copies the valid data from block 0 to block 1. This is called data reclaim processing. Data reclaim processing is performed automatically by the data update function. It uses programming procedures equivalent to those of data update processing, but the checksum value in the data update start flag is not recalculated and data is copied between areas.

Figures 1.17 to 1.19 illustrate the block states and stored data in data reclaim processing.

1. From the state shown in figure 1.17, the valid data management information (C) for data number 0 in block 0 is copied to E, and the valid data (C') is copied to E', in block 1.

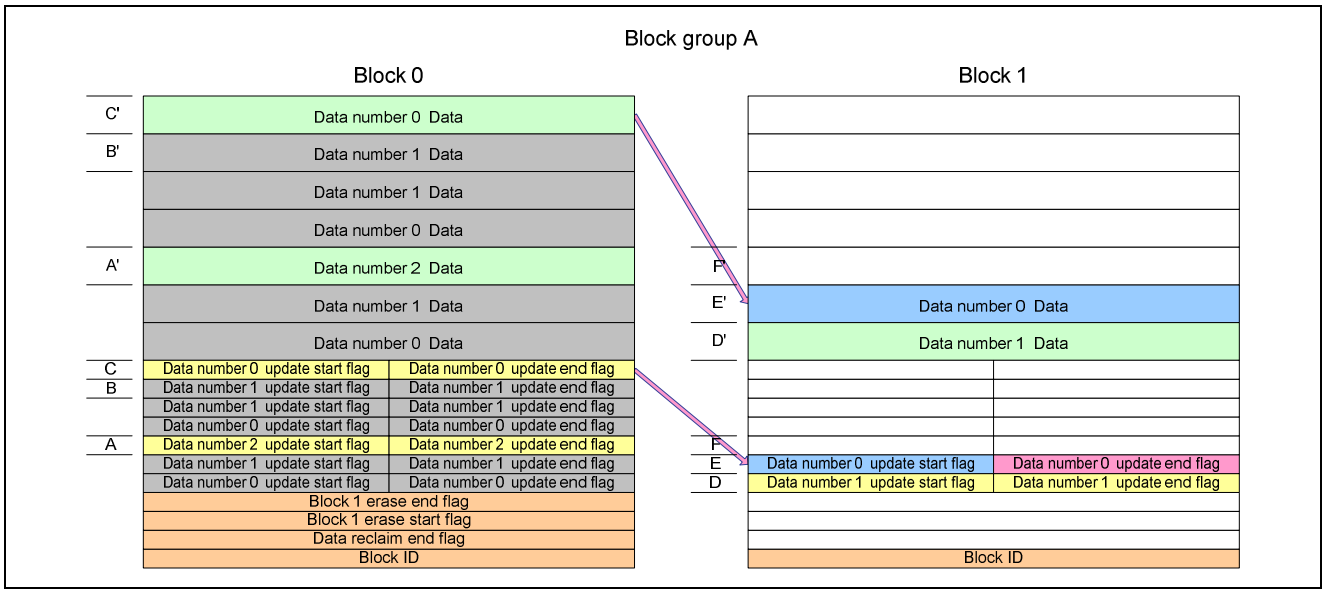

Figure 1.17 Illustration of Data Storage in Data Reclaim Processing – 1

When a system shutdown occurs in the state shown in figure 1.17, either (a) or (b) below takes place the next time driver initialization occurs, depending on the blank check result for data management information area E.

(a) If at least one of the flag areas in data management information area E is still in the blank state, the driver considers C to be valid data management information and C' to be valid data for data number 0, and data management information area E and data area E' to be invalid areas, so data management information area F and data area F' are used when the next data update occurs. After data update processing of F and F', data reclaim processing is performed again.

(b) If data management information area E is determined to in the non-blank state, the driver considers E to be valid data management information and E' to be valid data for data number 0, and data management information area C and data area C' to be invalid areas. After data update processing of F and F', data reclaim processing is performed again.

2. Next, in the same manner, the valid data management information (A) for data number 2 in block 0 is copied to F, and the valid data (A') is copied to F', in block 1.

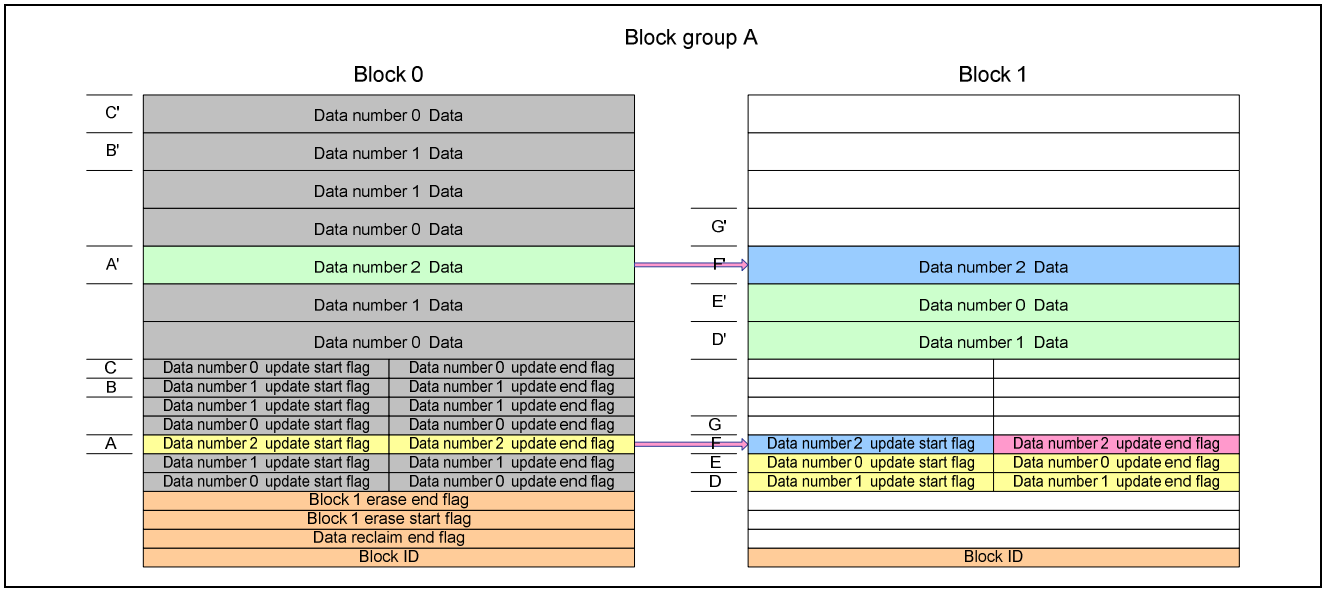

Figure 1.18 Illustration of Data Storage in Data Reclaim Processing – 2

When a system shutdown occurs in the state shown in figure 1.18, either (a) or (b) below takes place the next time driver initialization occurs, depending on the blank check result for data management information area F.

(a) If at least one of the flag areas in data management information area F is still in the blank state, the driver considers A to be valid data management information and A' to be valid data for data number 2, and data management information area F and data area F' to be invalid areas, so data management information area G and data area G' are used when the next data update occurs. After data update processing of G and G', data reclaim processing is performed again.

(b) If data management information area F is determined to in the non-blank state, the driver considers F to be valid data management information and F' to be valid data for data number 2, and data management information area A and data area A' to be invalid areas. At this point all the valid data in block group A is concentrated in block 1, so the driver programs the data reclaim end flag in block 1 and registers block 0 as an erase target block (as in figure 1.19).

3. When data reclaim processing completes (all block valid data in group A concentrated in block 1), the driver programs the data reclaim end flag in block 1 and registers block 0 as an erase target block.

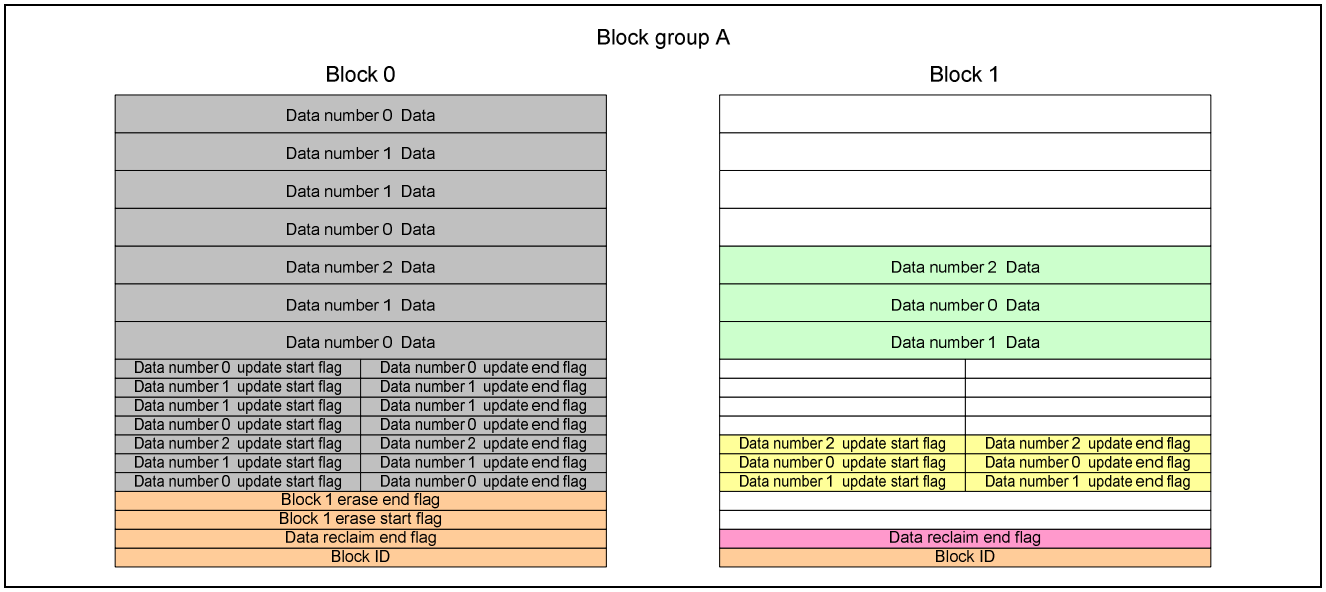

Figure 1.19 Programming of Data Reclaim End Flag

When a system shutdown occurs in the state shown in figure 1.19, if the data reclaim end flag area is blank the next time driver initialization takes place, the driver programs the data reclaim end flag and registers block 0 as an erase target block.

1.9.5 Block Erase – 1 (Normal Operation)

Figures 1.20 to 1.22 illustrate the block states when using the block erase function to erase block 0 from the state shown in figure 1.19.

1. The driver programs the erase start flag in block 1.

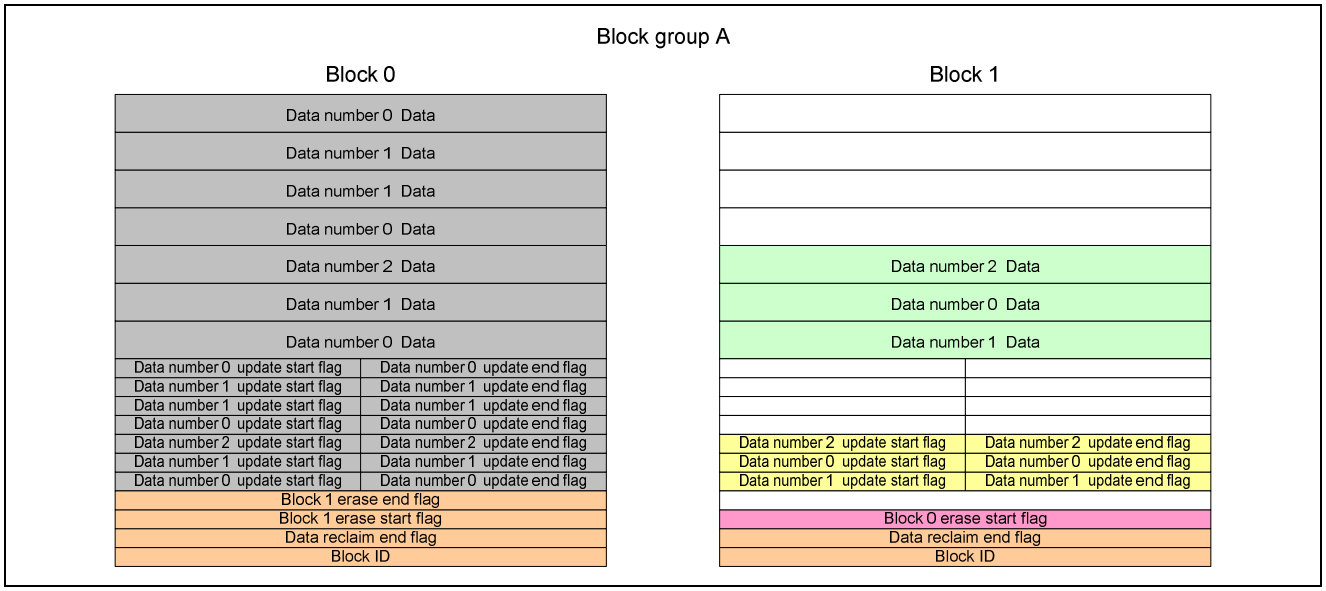

Figure 1.20 Programming of Erase Start Flag

When a system shutdown occurs in the state shown in figure 1.20, the driver registers block 0 as an erase target block the next time driver initialization takes place. Block erase processing is performed again by calling the block erase function. At this time, programming the erase start flag is omitted if the erase start flag area in block 1 is non-blank.

2. Next, block 0 is erased.

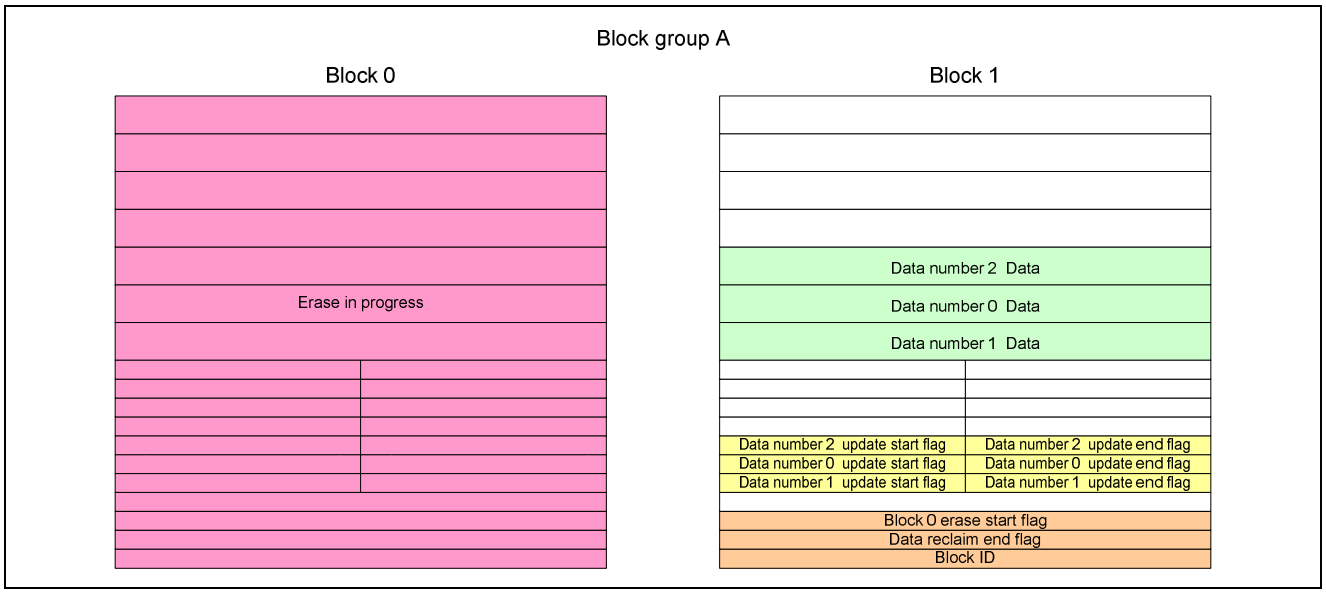

Figure 1.21 Block Erase

When a system shutdown occurs in the state shown in figure 1.21, either (a) or (b) below takes place the next time driver initialization occurs, depending on the blank check result for block 0.

(a) If block 0 is in the blank state, erase processing is determined to have completed, so the driver programs the erase end flag in block 1 (as in figure 1.22), programs the block ID of block 0 (as in figure 1.23), and ends block erase processing of block 0.

(b) If block 0 is in the non-blank state, erase processing is determined to have been interrupted, so the driver registers block 0 as an erase target block. Block erase processing is performed again by calling the block erase function. At this time, programming the erase start flag in block 1 is omitted.

3. After the erase of block 0 completes, the driver programs the erase end flag in block 1.

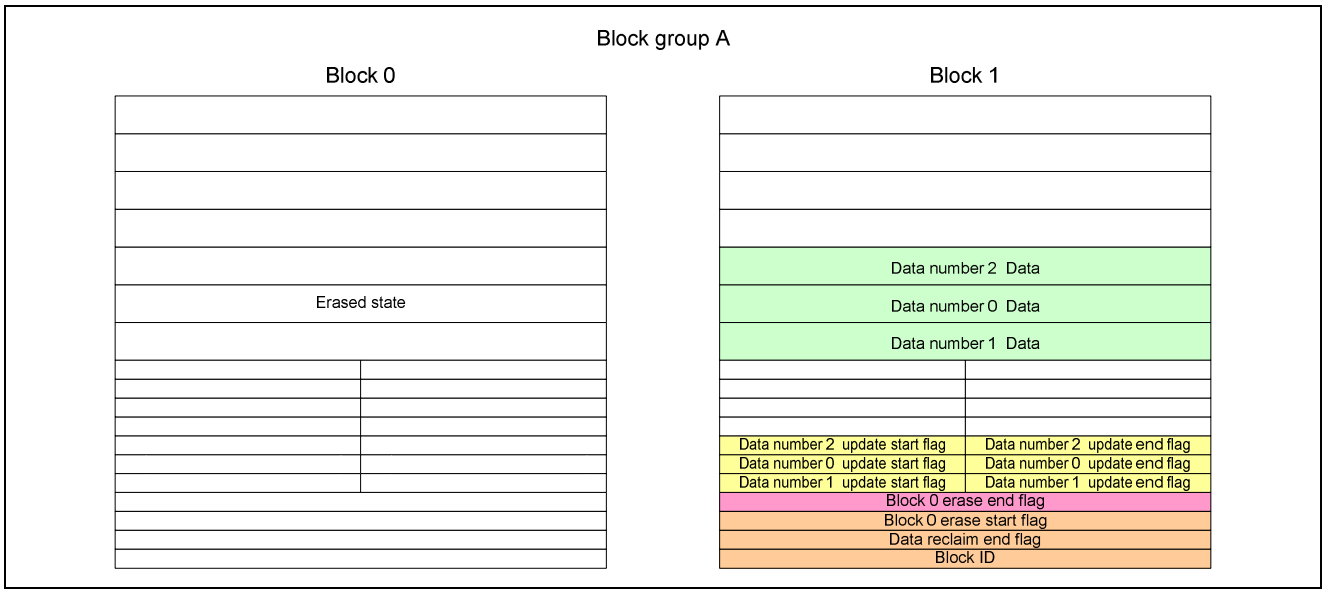

Figure 1.22 Programming of Erase End Flag

When a system shutdown occurs in the state shown in figure 1.22, either (a) or (b) below takes place the next time driver initialization occurs, depending on the blank check result for the erase end flag in block 0.

(a) If the erase end flag in block 1 is in the blank state, programming of the erase end flag is determined to have been interrupted, so the driver programs the erase end flag in block 1. Next, it programs the block ID of block 0 (as in figure 1.23) and ends block erase processing of block 0.

(b) If the erase end flag in block 1 is in the non-blank state, programming of the block ID of block 0 is determined to have been interrupted following programming of the erase end flag, so the driver programs the block ID of block 0 (as in figure 1.23) and ends block erase processing of block 0.

4. Next, the driver programs the block ID of block 0.

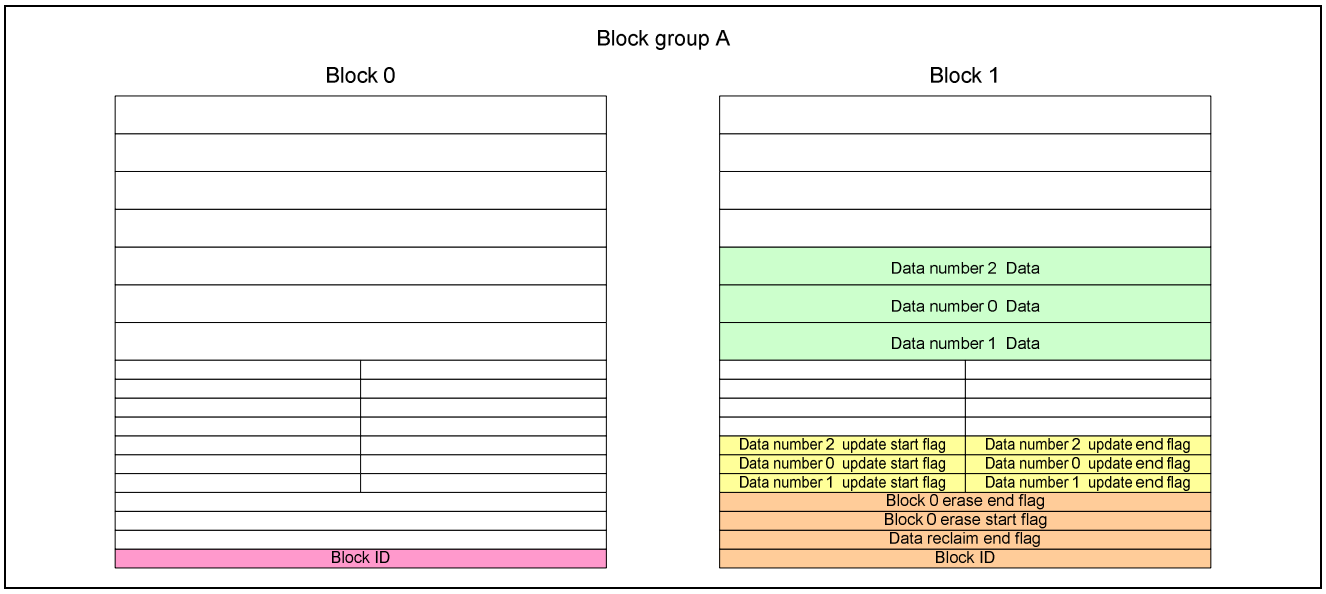

Figure 1.23 Programming of Block ID

When a system shutdown occurs in the state shown in figure 1.23, either (a) or (b) below takes place the next time driver initialization occurs, depending on the blank check result for block 0.

(a) If block 0 is in the blank state, programming of the block ID is determined to have been interrupted, so the driver programs the block ID of block 0 and ends block erase processing of block 0.

(b) If block 0 is in the non-blank state (the block ID area of block 0 is in the non-blank state), the driver ends block erase processing of block 0.

1.9.6 Data Reclaim – 2 (State with Insufficient Copy Area Available)

As described above, when a system shutdown or similar condition occurs during data update function processing or data reclaim processing, the (non-blank) data management information area that was already programmed before processing completion, and the corresponding data area, are considered to be invalid areas, and the areas to be programmed in the next data update are shifted, from a larger to a smaller address.

Figure 1.24 illustrates the block states and stored data following the next time driver initialization occurs when a data update of data number 1 is performed in a state where there is no data area available for programming in block 0.

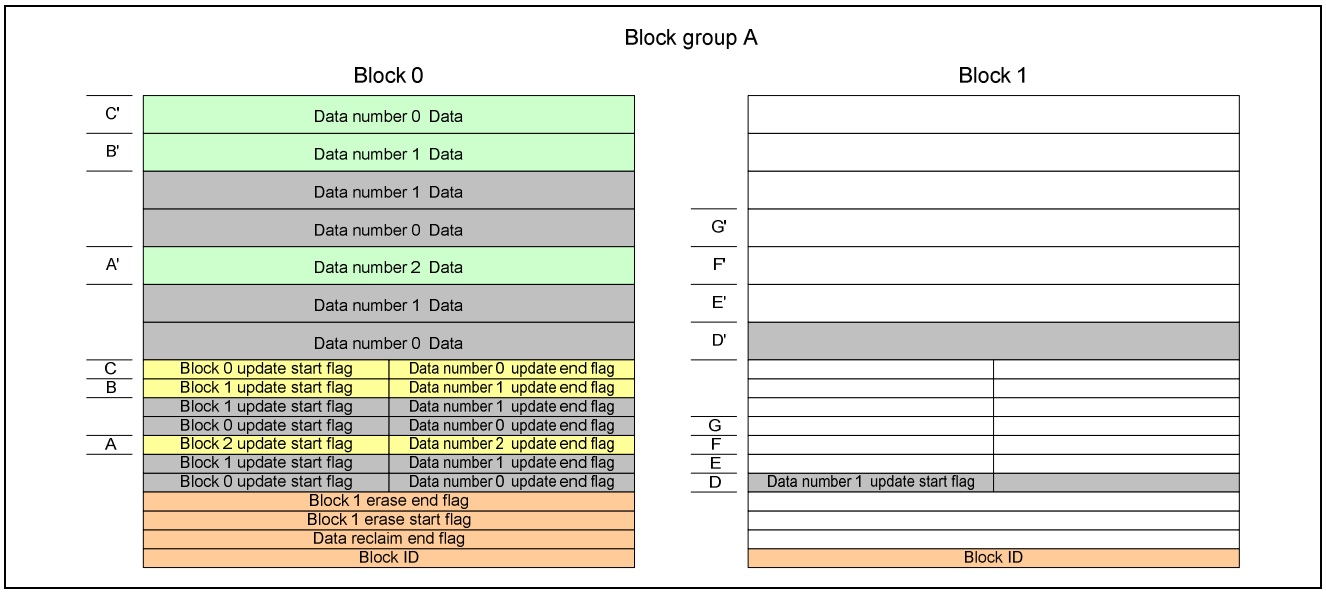

Figure 1.24 Illustration of Data Storage when Data Update Interrupted – 1 (State with Sufficient Copy Area Available)

When a data update of data number 1 is performed again in the state shown in figure 1.24, the update data for data number 1 is programmed in E', after which the driver automatically executes data reclaim processing in which the data number 0 reclaim data is programmed in F' and the data number 2 reclaim data is programmed in G'. (The data management information is also programmed in the corresponding areas (E, F, and G)).

Figure 1.25 illustrates the block states and stored data the next time driver initialization occurs when processing interrupted conditions (a) to (d), like those described above, occur together from the state shown in figure 1.24.

(a) Completion of data update for data number 1 (figure 1.25 E, E')
(b) System shutdown during data reclaim processing for data number 0 (during data programming) following (a) (figure 1.25 F, F')
(c) System shutdown during data update processing for data number 0 (during update start flag programming) (figure 1.25 G)
(d) System shutdown during data update processing for data number 0 (during data programming) (figure 1.25 H')

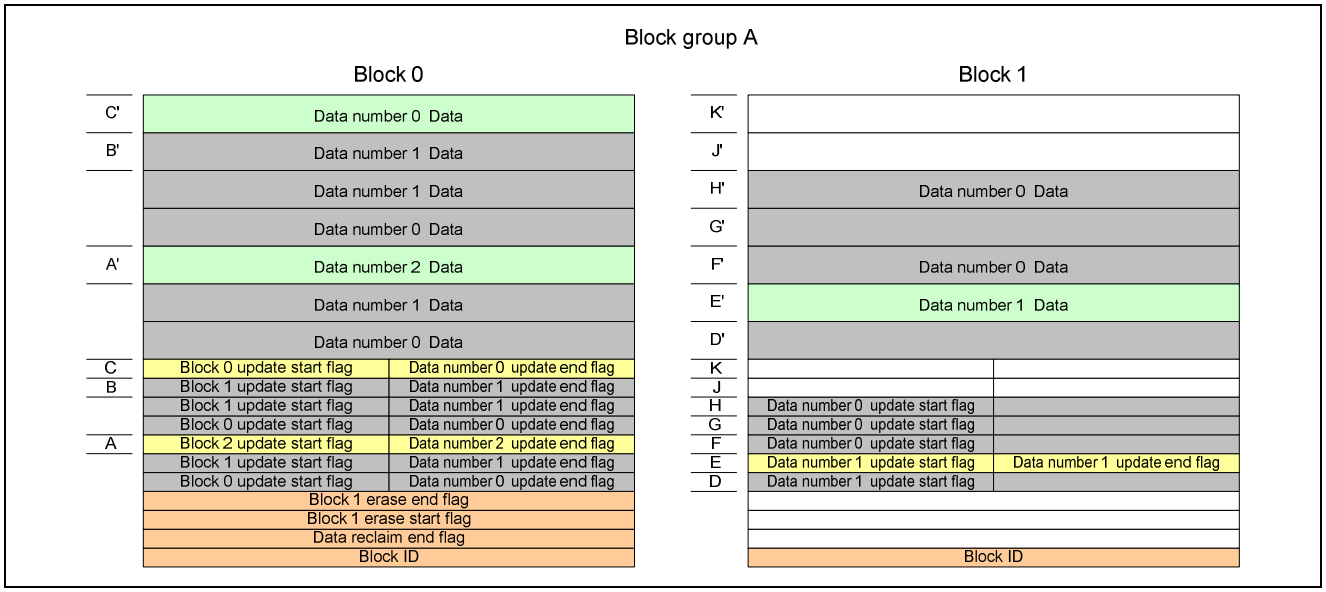

Figure 1.25 Illustration of Data Storage when Data Update Interrupted – 2 (State with Insufficient Copy Area Available)

In the state shown in figure 1.25, the valid data remaining in block 0 comprises two data numbers, 0 and 2, and the areas available for programming in block 1 are also two in number, J (J') and K (K'). Thus, the number of data numbers requiring reclaim processing is equal to the number of areas available for programming. Since data reclaim processing is performed subsequently to data update processing, if a data update request for data number 1 were issued in this state, the only area available for programming in block 1 after updating data for data number 1 by programming J (J') would be K (K'). It would therefore not be possible to copy all the valid data remaining in block 0 (data numbers 0 and 2) to block 1. The result would be that no areas available for programming would be left in block group A, even though valid data remained in both blocks 0 and 1, and it would not be possible to perform further data updates while retaining the valid data in the block group.

Therefore, when the driver determines the possibility that it may be unable to concentrate all the valid data in block group A within block 1 (a state in which the number of data numbers requiring reclaim processing is equal to the number of areas available for programming), it treats all the valid data already updated or copied in block 1 (figure 1.25 E') as invalid data, treats the data in block 0 from before the update or copy (figure 1.25 A', B', C') as valid data, and registers block 1 as an erase target block.

Note: In the above example, the new data for data number 1 in block 1, which was once considered valid, is treated as invalid, and the old data in block 0, which was earlier considered invalid, is treated as valid data.

Subsequently, a block erase of block 1 returns the block group to the state shown in figure 1.12 in 1.9.3, Data Update – 2 (No Areas Available for Programming within Block), and further data update and data reclaim processing is possible once again.

Note: In this case, programming of the erase start flag and erase end flag in block 0 is omitted from the block erase processing function when block 1 is erased.

1.9.7 Data Update – 3 (Data Update During Block Erase Processing)

When the data update processing function is called by an interrupt while a block erase is in progress, basically, the data update processing is given priority and the block erase processing enters a suspended state.

Note: When the data update processing function is called during programming of the erase start flag, erase end flag or block ID, the driver sends a return value (DF_ST_WARNING_DRIVER_BUSY) indicating a busy state, and erase processing continues. In this case, shift the timing and call the data update processing function once again.

Figure 1.26 illustrates the block states and stored data when the data update processing function is called by an interrupt during block erase processing as shown in figure 1.21 in 1.9.5, Block Erase – 1 (Normal Operation).

1. The driver puts erase processing of block 0 into a suspended state.

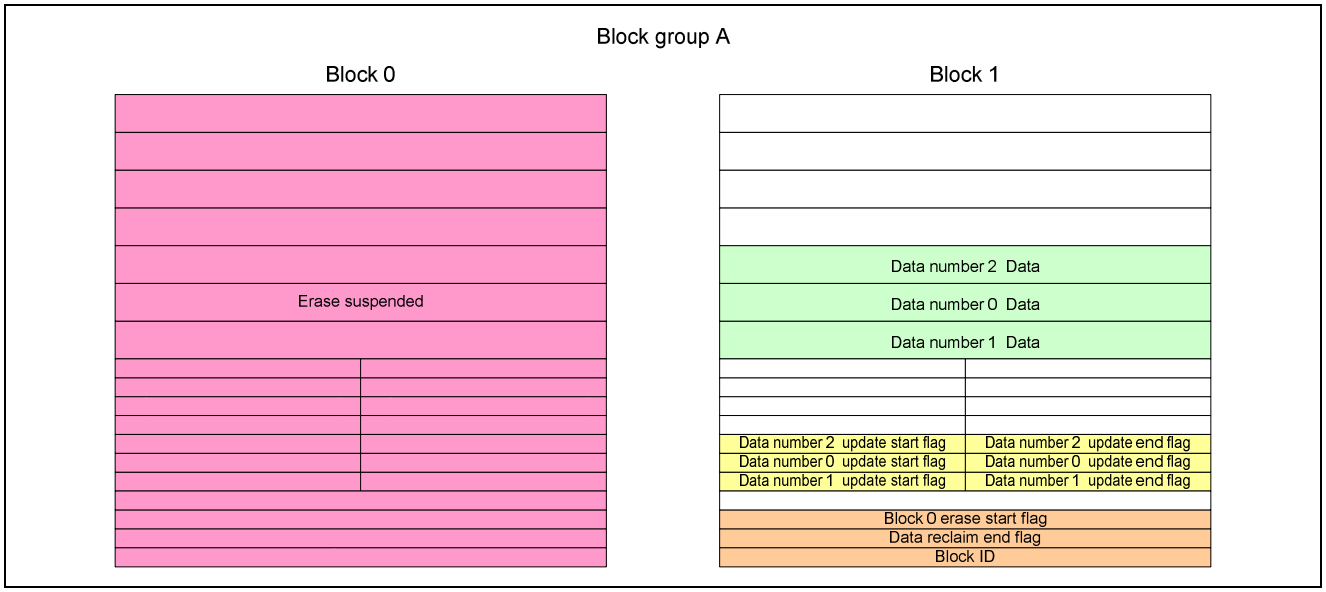

Figure 1.26 Illustration of Data Update During Block Erase Processing – 1 (Erase Suspended)

When a system shutdown occurs in the state shown in figure 1.26, data update processing for data number 2 halts, and the driver registers block 0 as an erase target block the next time driver initialization takes place. Block erase processing is performed again by calling the block erase function. At this time, programming the erase start flag in block 1 is omitted.

2. Next, the data update start flag, data, and data update end flag for data number 2 are programmed, in that order, in areas available for programming in block 1 (figure 1.27 B, B'). (This is equivalent to a normal data update.)

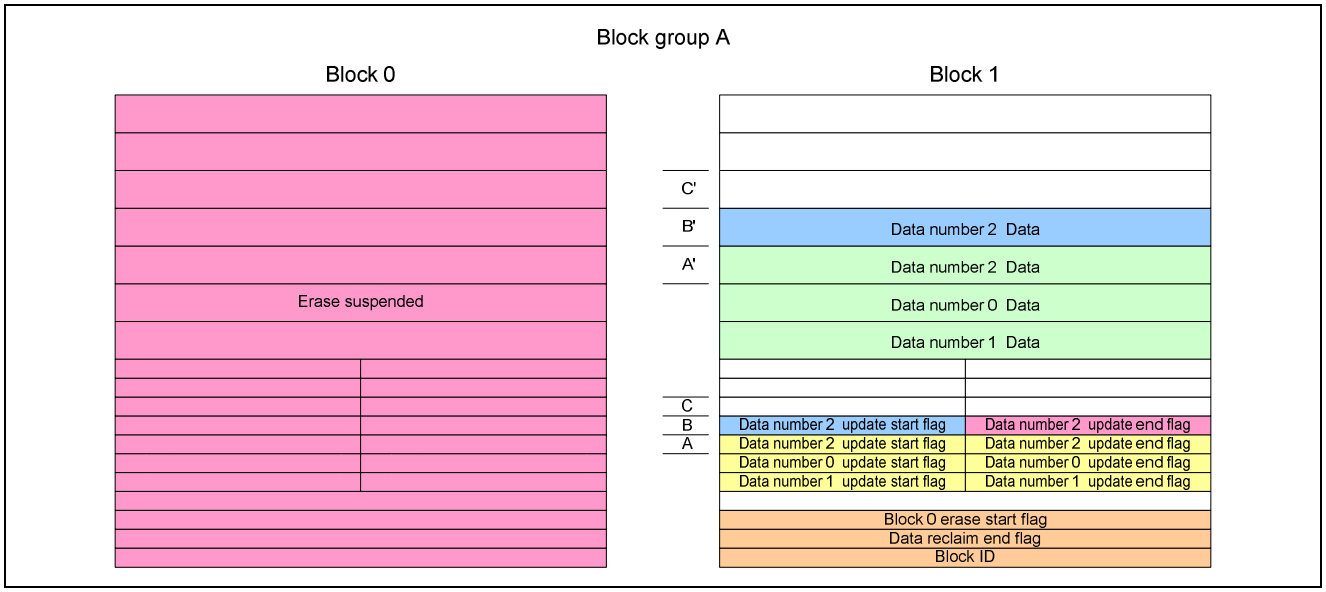

Figure 1.27 Illustration of Data Update During Block Erase Processing – 2 (Programming Update Data)

The determinations and processing the next time driver initialization occurs after a system shutdown takes place during the programming operations shown in figure 1.27 are equivalent to the determinations and processing described in 1.9.2, Data Update – 1 (Normal Operation).
Note that when block 0 is registered as an erase target block and the block erase function is called to perform block erase processing again, programming the erase start flag in block 1 is omitted.

3. After the data update completes, the driver performs resume processing, canceling the erase suspended state and automatically restarting erase processing for block 0.

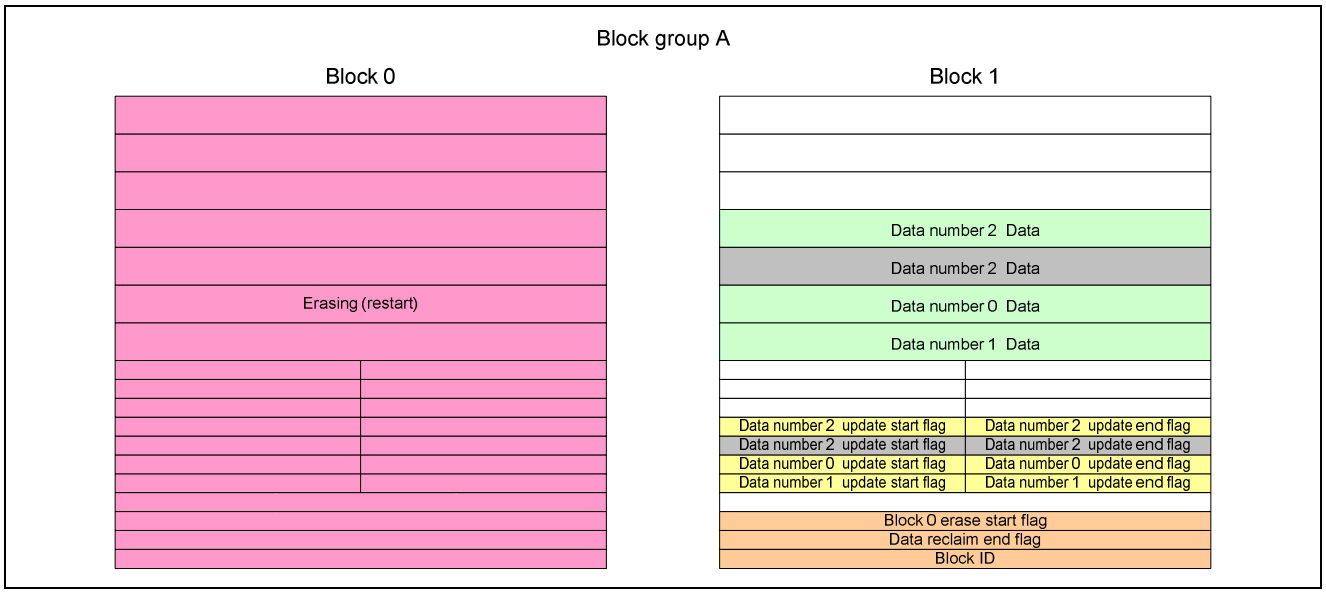

Figure 1.28 Illustration of Data Update During Block Erase Processing – 3 (Erase Restart)

After the state shown in figure 1.28, processing proceeds as described in 1.9.5, Block Erase – 1 (Normal Operation).

1.9.8 Data Update – 4 (No Areas Available for Programming in Block Group)

When the data update function is called in a state where there are no areas available for programming in the block group, the driver sends a return value (DF_ST_WARNING_NO_BLANK_AREA) indicating that there are no blank areas remaining,

The main causes that can result in a state in which no areas are available for programming are as follows.

1. When in the state shown in figure 1.19 in 1.9.4, Data Reclaim – 1 (Normal Operation), data updates continue without the block erase processing function being called to perform a block erase (figure 1.29)

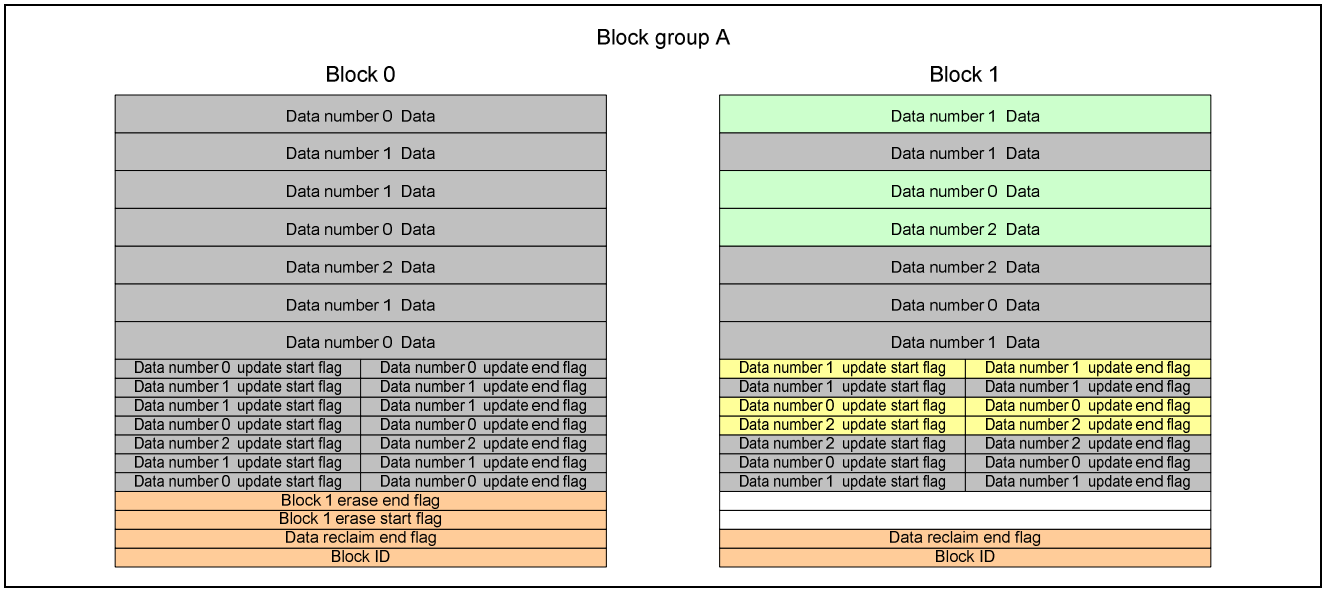

Figure 1.29 Illustration of State with No Areas Available for Programming Due to Failure to Erase Invalid Data (Block)

2. When the state shown in figure 1.25 in 1.9.6, Data Reclaim – 2 (State with Insufficient Copy Area Available), is determined to exist
3. In the state between state (2) and the block erase end following it

4. When data update processing overlaps as described in 1.9.7, Data Update – 3 (Data Update During Block Erase Processing), and there are no areas available for programming in block 1 until block 0 erase end (figure 1.30)

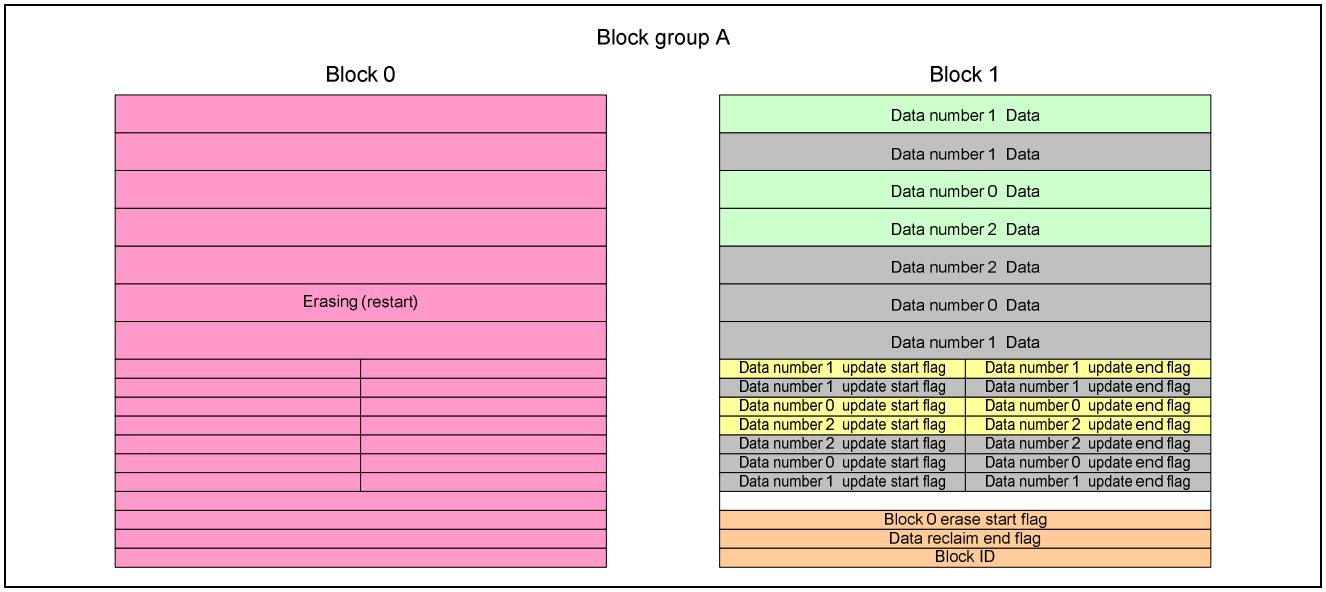

Figure 1.30 Illustration of State with No Areas Available for Programming During Erase Processing

5. When the erase of one block has not completed and there are no areas available for programming in the paired block due to an error or system shutdown other than those described above

1.10 User API Functions

Table 1.4 lists the user API functions of the sample data flash driver software, and table 1.5 lists the type definitions used in the function descriptions.

Table 1.4 List of User API Functions

Function	Label	Brief Description
Driver initialization processing function	DF_Drv_Init	Initializes the data flash driver. In order to use the data flash driver, this function must be called during the system initialization sequence.
Data flash formatting function	DF_Format	Builds the data structure in the data flash to enable control by the driver.
Data update processing function	DF_Write_Data	Updates the data of the data number specified by an argument. If necessary, data reclaim processing is performed as well.
Valid data read processing function	DF_Read_Data	Reads the valid data stored in data flash for the data number specified by an argument, and stores the data at a pointer location specified by an argument.
Block erase processing function	DF_Erase_block	Erases the block registered as an erase target block.
Error handler function	DF_Error_Management	Performs processing according to the error state.
Data size acquisition function	DF_Data_Size	Sends back as a return value the data size of the data number specified by an argument.
Blank area count acquisition function	DF_Blank_Number	Sends back as a return value the number of blank areas within the block being used for data updates in the block group specified by an argument.
Data flash read protect processing function	DF_Read_Protect	Applies read protection to the data flash.
Data flash read protect cancel processing function	DF_Read_Protect_Cancel	Cancels data flash read protection.
Data flash programming/erase protect processing function	DF_WriteErase_Protect	Applies programming/erase protection to the data flash.
Data flash programming/erase protect cancel processing function	DF_WriteErase_Protect_Cancel	Cancels data flash programming/erase protection.

Table 1.5 List of Type Definitions Used in Function Description

Type Definition	Type
_u1	unsigned char
_s2	signed short

1.10.1 Driver Initialization Processing Function

Table 1.6 provides a description of the data flash driver initialization processing function, and figure 1.31 shows an outline flowchart of its operation.

Table 1.6 Data Flash Driver Initialization Processing Function Description

Function			
Data flash driver initialization processing function **_s2 DF_Drv_Init(void)**			
Arguments			
—			
Return values			
_s2	DF_ST_OK	0 : Initialization normal end	
	DF_ST_OK_REQUEST_ERASE	-1 : Initialization normal end	(Request call of block erase processing function)
	DF_ST_ERR_REQUEST_FORMAT	-7 : Unformatted or data missing	(Request call of formatting function)
	DF_ST_ERR_DEVICE	-8 : Device error	(Read-only)
	DF_ST_ERR_INIT	-12 : Device initialization error	(Data flash access prohibited)
	DF_ST_ERR_PARAMETER_INVALID	-14 : Parameter error	(Setting in DF_user.c outside valid range)
Description			
Initializes the data flash driver. In order to use the data flash driver, this function must be called during the system initialization sequence.			
Usage notes			

- In order to use the data flash driver, this function must be called during the system initialization sequence.
- Initialization processing ends when control returns from the function. Use the function's return value to determine whether or not initialization completed successfully.
- A return value of "DF_ST_OK_REQUEST_ERASE" indicates that the driver encountered an "erase all blocks" request during the processing of this function. When this occurs, call the block erase processing function to erase the specified block. If the data update processing function is called without erasing the block first, a data update is performed if there are sufficient areas available for programming in the block, but if there are not sufficient areas available for programming, no data update is performed and a return value of "DF_ST_WARNING_NO_BLANK_AREA" is sent back.
- A return value of "DF_ST_ERR_REQUEST_FORMAT" indicates that the driver has encountered a state where it cannot access data. When this occurs, call the data flash formatting function to rebuild the data area. This will cause all the data in the block group to return to the initial data (undefined value) state.
- A return value of "DF_ST_ERR_DEVICE" indicates that the driver encountered a program error during the processing of this function, resulting in a state in which only reading of data using the valid data read processing function is possible. In this state it is not possible to update data using the data update processing function or to erase data using the block erase processing function.
- A return value of "DF_ST_ERR_INIT" indicates that the driver encountered an FCU firmware verify error or an error during blank checking of the data flash that prevented acquisition of information needed to access the data area, resulting in a state in which it is not possible to read valid data in the data flash, to update data, or to perform a block erase. When this occurs, first call the data flash error handler function to deal with the error, then call the data flash driver initialization processing function once again. (Depending on the state of the device and the error circumstances, it may not be possible to recover normal operation in some cases.)
- A return value of "DF_ST_ERR_PARAMETER_INVALID" indicates a state in which initialization is not possible because a setting value in DF_user.h or DF_user.c is outside the allowable range. When this occurs, ensure that the setting values in DF_user.h and DF_user.c are within the valid range, and recompile the source code.
- The driver functions that can be called by user applications using interrupts while the driver initialization processing function is running are listed below. Calling functions other than those listed could cause malfunctions in the subsequent operation of the driver.
  - — Data flash read protect cancel function
  - — Data flash programming/erase protect cancel function
  - — Data size acquisition function

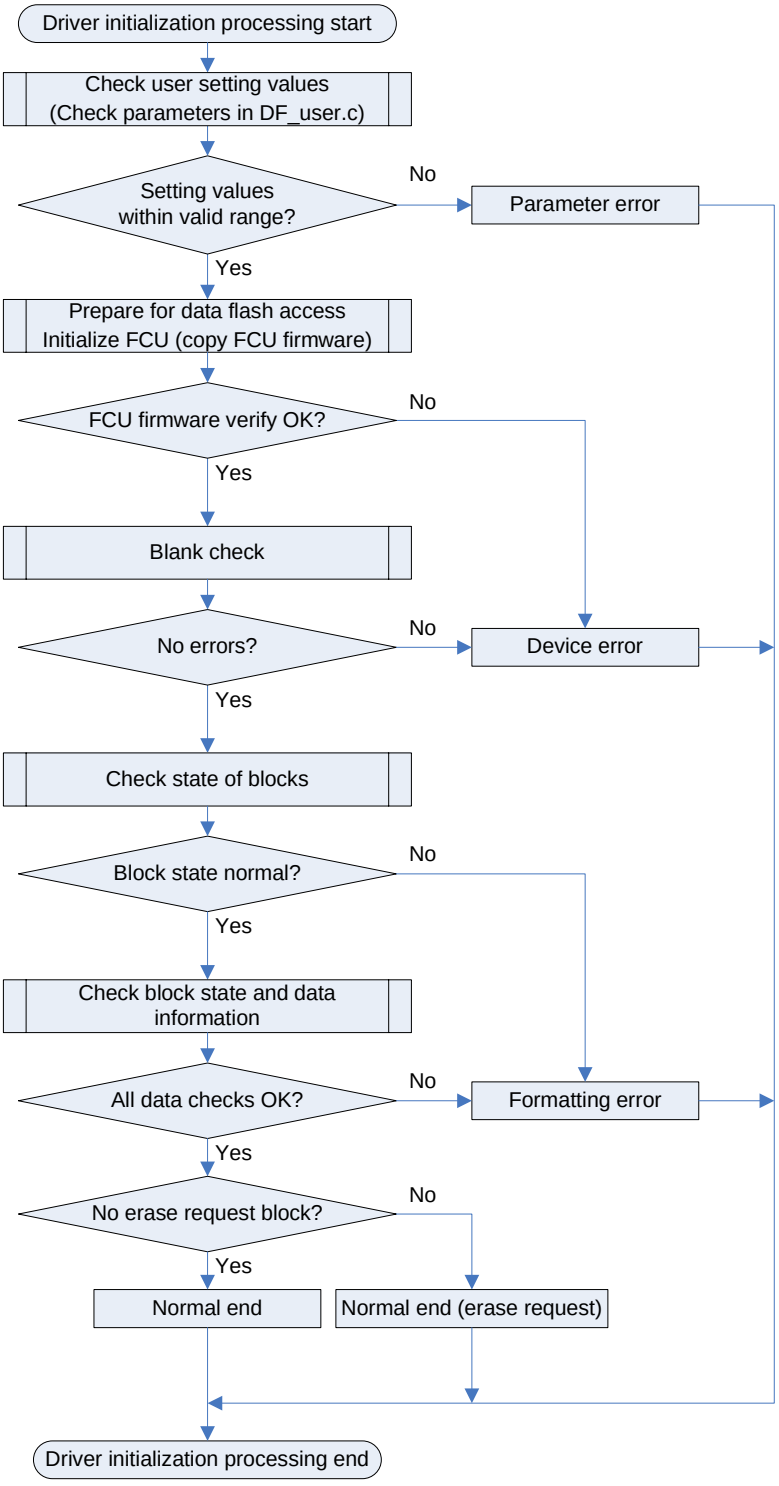

Figure 1.31 Data Flash Driver Initialization Processing Function Outline Flowchart

1.10.2 Formatting Function

Table 1.7 provides a description of the data flash formatting function, and figure 1.32 shows an outline flowchart of its operation.

Table 1.7 Data Flash Formatting Function Description

Function				
Data flash formatting function **_s2 DF_Format(_u1 code)**				
Arguments				
_u1	Format_start	170	: Formatting enable code	
Return values				
_s2	DF_ST_OK	0	: Formatting normal end	
	DF_ST_WARNING_WRITE_ERASE_PROTECT	-5	: Programming prohibited state	(Request protect cancel)
	DF_ST_WARNING_INVALID_ARGUMENT	-6	: Enable code error	(Illegal formatting enable code)
	DF_ST_ERR_FORMAT	-13	: Formatting failure	(Data flash access prohibited)
Description				
Builds the data structure in the data flash to enable control by the driver.				
Usage notes				

- Call this function after the data flash has been initialized by the data flash driver initialization processing function. However, do not call this function if the data flash driver initialization processing function generates a return value of “DF_ST_ERR_DEVICE,” “DF_ST_ERR_INIT,” or “DF_ST_ERR_PARAMETER_INVALID.”
- Formatting ends when control returns from the function. Use the function’s return value to determine whether or not formatting completed successfully.
- After the function is run, a new data area that contains undefined values will have been built, regardless of the previous state of the data flash. Therefore, reading data (IDs) after the function is run but before a data update will produce undefined values.
- A return value of “DF_ST_WARNING_WRITE_ERASE_PROTECT” indicates that programming/erase protection has been applied to the data flash and that formatting cannot be performed. When this occurs, call the data flash programming/erase protect cancel function to cancel programming/erase protection.
- A return value of “DF_ST_WARNING_INVALID_ARGUMENT” indicates that formatting cannot be performed because the argument (formatting enable code: 170) does not match.
- A return value of “DF_ST_ERR_FORMAT” indicates that formatting (building of the data structure) failed, resulting in a state in which the driver cannot read valid data, update data, or perform block erase processing.
- The driver functions that can be called by user applications using interrupts while the data flash formatting function is running are listed below. Calling functions other than those listed could cause malfunctions in the subsequent operation of the driver.
  - — Data flash read protect cancel function
  - — Data flash programming/erase protect cancel function
  - — Data size acquisition function

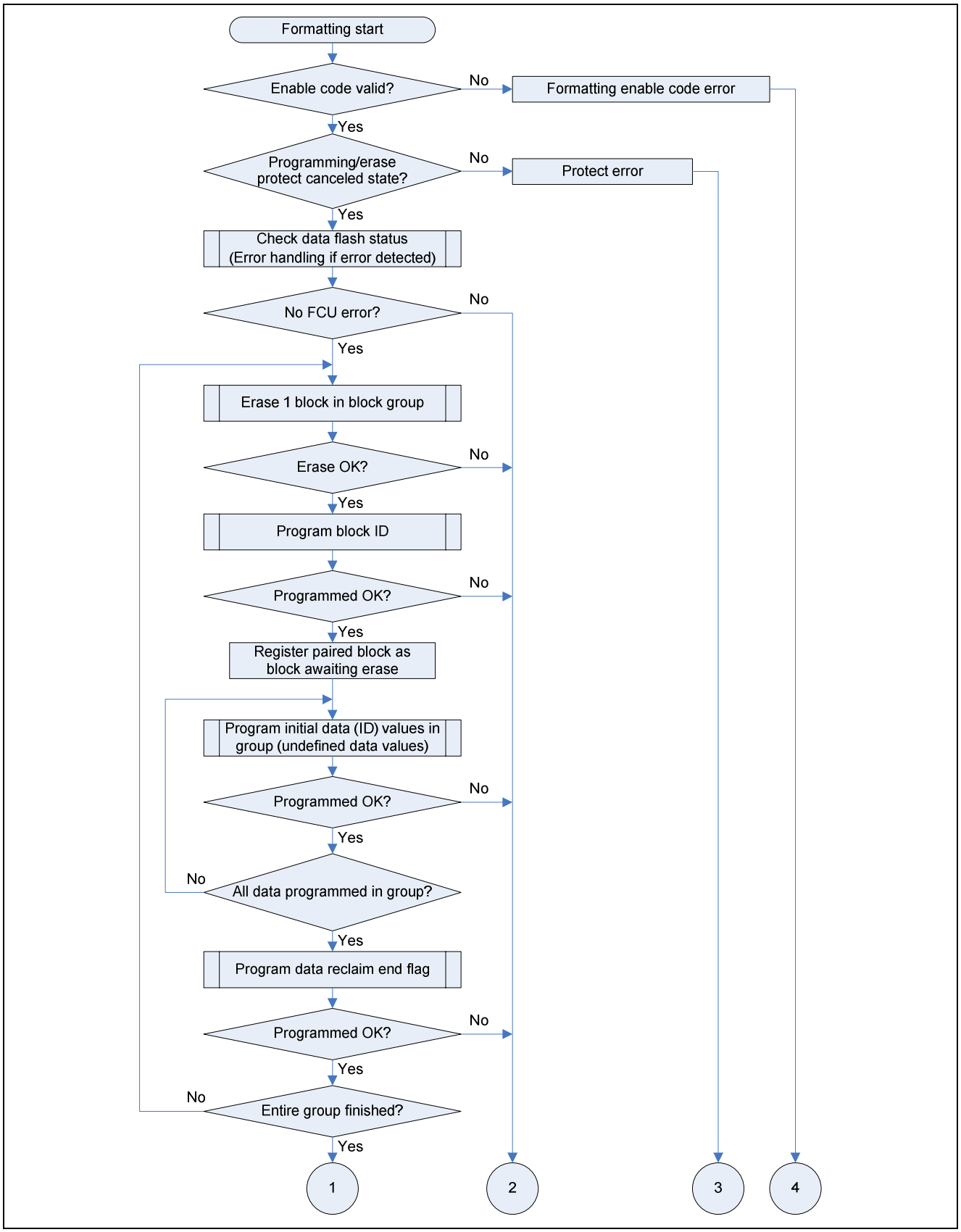

Figure 1.32 Data Flash Formatting Function Outline Flowchart 1

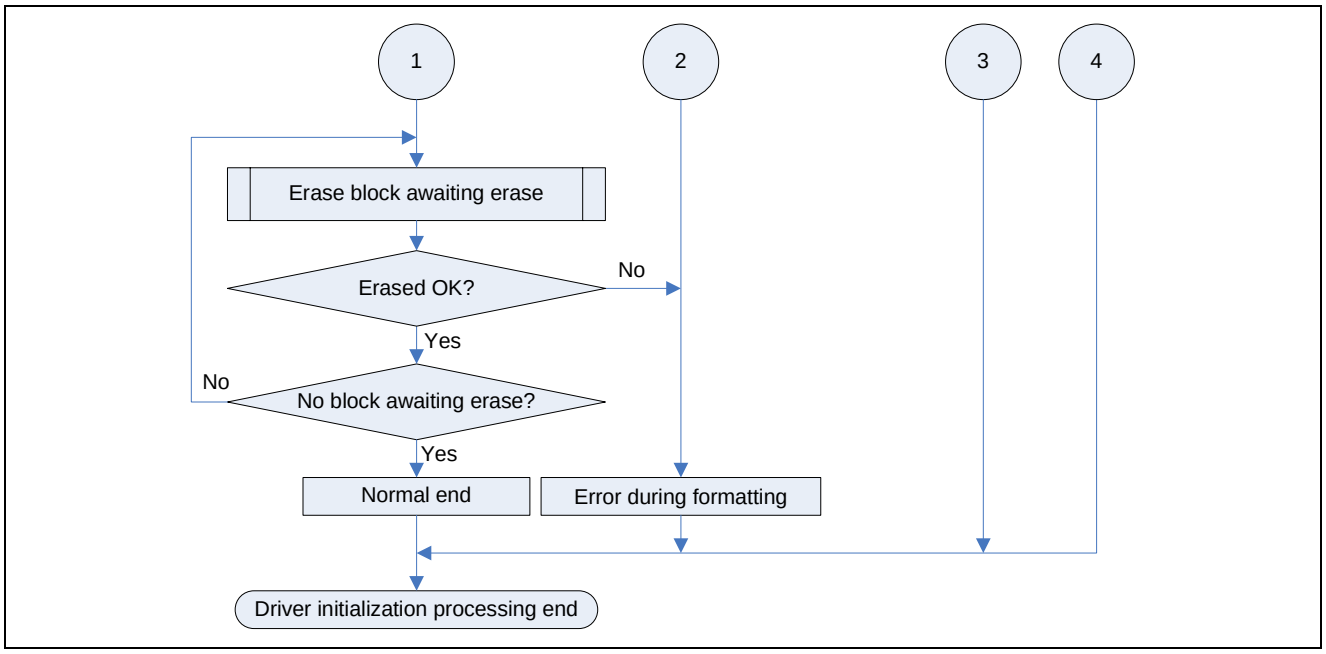

Figure 1.33 Data Flash Formatting Function Outline Flowchart 2

1.10.3 Data Update Processing Function

Table 1.8 provides a description of the data update processing function, and figures 1.34 and 1.35 show an outline flowchart of its operation.

Table 1.8 Data Update Processing Function Description

Function			
Data update processing function **_s2 DF_Write_Data(_u1 data_id, _u1 *user_data_address)**			
Arguments			
_u1	data_id		: Data number to be updated
_u1	*user_data_address		: Pointer to update data storage destination
Return values			
_s2	DF_ST_OK	0	: Data update normal end
	DF_ST_OK_REQUEST_ERASE	-1	: Data update normal end (Request call of block erase processing function)
	DF_ST_WARNING_DRIVER_BUSY	-2	: Driver busy (Wait request)
	DF_ST_WARNING_NO_BLANK_AREA	-3	: No area available for programming (Wait for block erase end) (Request call of block erase processing function)
	DF_ST_WARNING_READ_PROTECT	-4	: Read prohibited state (Request protect cancel)
	DF_ST_WARNING_WRITE_ERASE_PROTECT	-5	: Programming prohibited state (Request protect cancel)
	DF_ST_WARNING_INVALID_ARGUMENT	-6	: Data number error (Data number outside defined range)
	DF_ST_ERR_DEVICE	-8	: Error detected (Read-only)
	DF_ST_ERR_DEVICE_TIMEOUT	-9	: Error detected (Read-only)
	DF_ST_ERR_FCU	-10	: Error detected (Read-only)
	DF_ST_ERR_LOST_ID	-11	: Error detected (Data missing)
Description			
Updates the data of the data number specified by an argument. If necessary, data reclaim processing is performed as well.			

Usage notes

- Call this function after driver initialization by the data flash driver initialization processing function has completed successfully.
- Data update processing ends when control returns from the function. Use the function's return value to determine whether or not the data update completed successfully.
- A return value of "DF_ST_OK_REQUEST_ERASE" indicates that data update processing completed successfully and the driver encountered an "erase all blocks" request during the processing of this function. When this occurs, call the block erase processing function to erase the specified block. If the data update processing function is called without erasing the block first, a data update is performed if there are sufficient areas available for programming in the block, but if there are not sufficient areas available for programming, no data update is performed and a return value of "DF_ST_WARNING_NO_BLANK_AREA" is sent back.
- A return value of "DF_ST_WARNING_DRIVER_BUSY" indicates one of the following states and that a data update cannot be performed.
  - The data update processing function has been called multiple times simultaneously.
  - The valid data read processing function is running.
  - The data flash driver initialization processing function or data flash formatting function is running.
  - The driver was already in an error state when the data update processing function was called.
- A return value of "DF_ST_WARNING_NO_BLANK_AREA" indicates that a data update cannot be performed because there are not sufficient areas available for programming in the block group. When this occurs, perform the data update after block erase processing completed.
- A return value of "DF_ST_WARNING_READ_PROTECT" indicates that reclaim processing cannot be performed because read protection was applied to the data flash after data update processing completed successfully. When this occurs, call the data flash read protect cancel function to cancel read protection. (Reclaim processing will be performed during the next or a subsequent data update.)
- A return value of "DF_ST_WARNING_WRITE_ERASE_PROTECT" indicates that programming/erase protection has been applied to the data flash. Call the data flash programming/erase protect cancel function to cancel programming/erase protection.
- A return value of "DF_ST_WARNING_INVALID_ARGUMENT" indicates that the data number specified in the argument is outside the defined range. The valid range of data numbers is 0 to (DF_DATA_ID_NUM – 1), according to the data count (DF_DATA_ID_NUM) defined in DF_user.h.
- A return value of "DF_ST_ERR_DEVICE," "DF_ST_ERR_DEVICE_TIMEOUT," or "DF_ST_ERR_FCU" indicates that an error was detected during the processing of this function, preventing the data update from completing successfully. When this occurs, call the data flash error handler function to deal with the error. (Depending on the state of the device and the error circumstances, it may not be possible to recover normal operation in some cases.)
- A return value of "DF_ST_ERR_LOST_ID" indicates that after data update processing completed successfully, a data missing condition occurred when the corresponding reclaim data could not be found during data reclaim processing.
- The driver functions that can be called by user applications using interrupts while the data update processing function is running are listed below. Calling functions other than those listed could cause malfunctions in the subsequent operation of the driver.
  - Valid data read processing function
  - Data flash read protect cancel function
  - Data flash programming/erase protect cancel function
  - Data size acquisition function
  - Blank area count acquisition function (Note that the value returned may not be correct, depending on the data update processing state.)

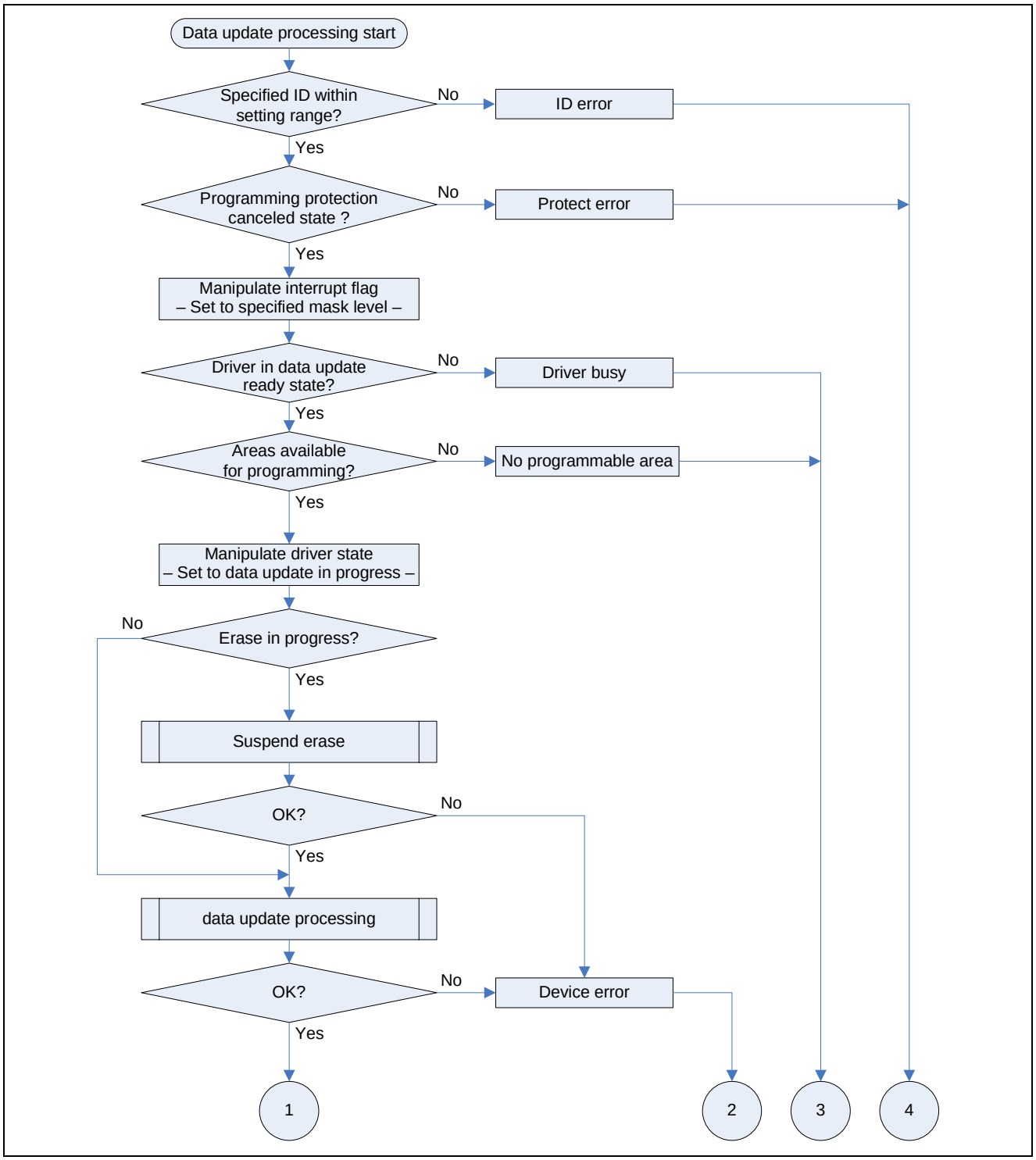

Figure 1.34 Data Update Processing Function Outline Flowchart 1

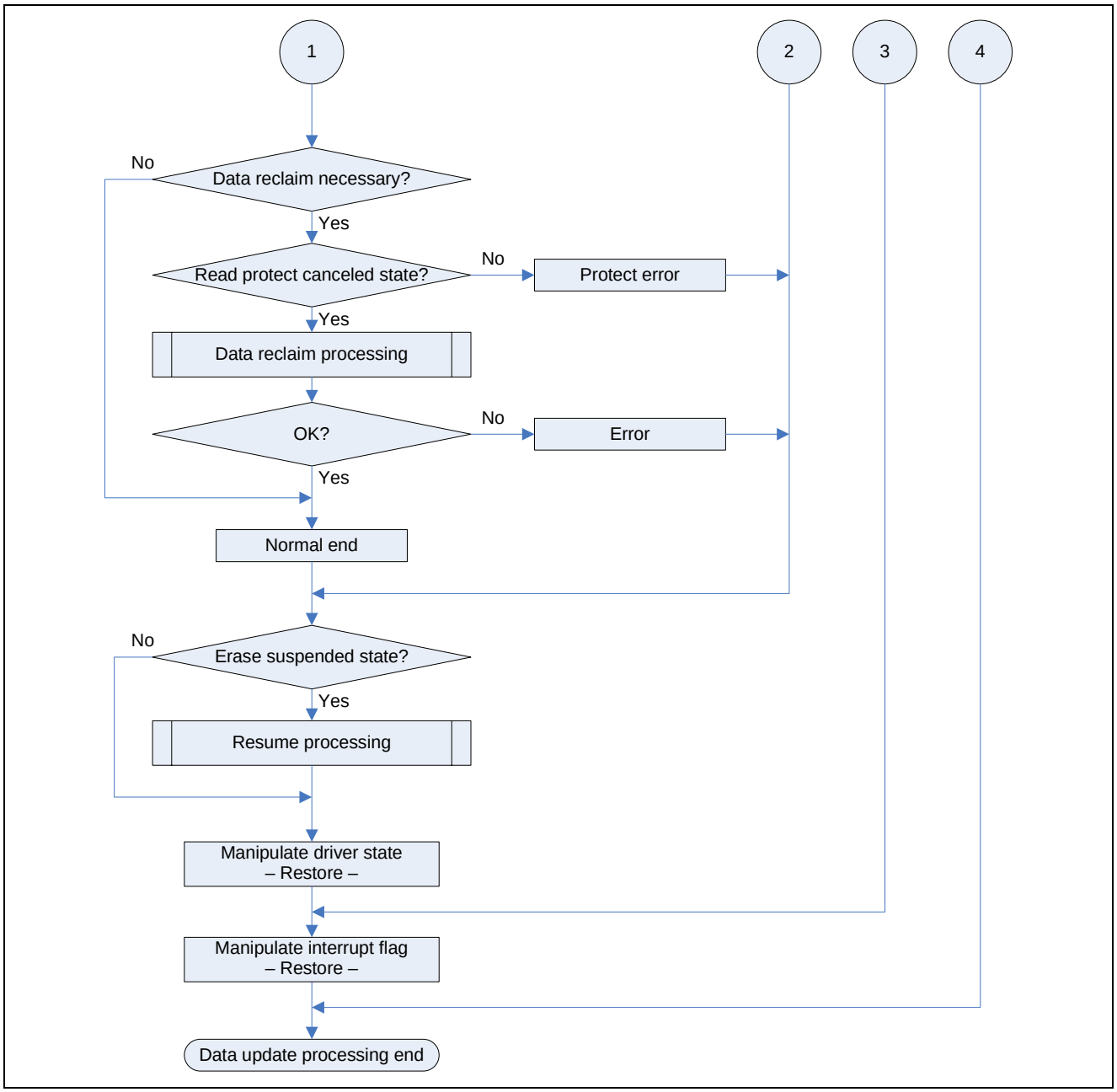

Figure 1.35 Data Update Processing Function Outline Flowchart 2

1.10.4 Valid Data Read Processing Function

Table 1.9 provides a description of the valid data read processing function, and figure 1.36 shows an outline flowchart of its operation.

Table 1.9 Valid Data Read Processing Function Description

Function				
Valid data read processing function **_s2 DF_Read_Data(_u1 data_id, _u1 *user_storage_address)**				
Arguments				
_u1	data_id		: Data number to be read	
_u1	*user_storage_address		: Pointer to read data storage destination	
Return values				
_s2	DF_ST_OK	0	: Read normal end	
	DF_ST_OK_REQUEST_ERASE	-1	: Read normal end (Request call of block erase processing function)	
	DF_ST_WARNING_DRIVER_BUSY	-2	: Driver busy	(Wait request)
	DF_ST_WARNING_READ_PROTECT	-4	: Read prohibited state	(Request protect cancel)
	DF_ST_WARNING_INVALID_ARGUMENT	-6	: Data number error	(Data number outside defined range)
	DF_ST_ERR_DEVICE	-8	: Error detected	
	DF_ST_ERR_DEVICE_TIMEOUT	-9	: Error detected	
	DF_ST_ERR_FCU	-10	: Error detected	
	DF_ST_ERR_LOST_ID	-11	: Error detected	(Data missing)
Description				
Reads the valid data stored in data flash for the data number specified by an argument, and stores the data at a pointer location specified by an argument.				
Usage notes				

- Call this function after driver initialization by the data flash driver initialization processing function has completed successfully.
- Read processing ends when control returns from the function. Use the function's return value to determine whether or not reading completed successfully.
- If the data of the data number to be read is being updated (data update processing has not completed), the valid data previous to the update is read.
- A return value of "DF_ST_OK_REQUEST_ERASE" indicates that data read processing completed successfully and the driver encountered an "erase all blocks" request during the processing of this function.
- A return value of "DF_ST_WARNING_DRIVER_BUSY" indicates one of the following states and that a data update cannot be performed.
  - The valid data read processing function has been called multiple times simultaneously.
  - The data update processing function is running in the block erase suspended state.
  - The data flash driver initialization processing function or data flash formatting function is running.
  - A driver initialization error or data flash formatting error has occurred.
- A return value of "DF_ST_WARNING_READ_PROTECT" indicates that read protection has been applied to the data flash. Call the data flash read protect cancel function to cancel read protection.
- A return value of "DF_ST_WARNING_INVALID_ARGUMENT" indicates that the data number specified in the argument is outside the defined range. The valid range of data numbers is 0 to (DF_DATA_ID_NUM – 1), according to the data count (DF_DATA_ID_NUM) defined in DF_user.h.
- A return value of "DF_ST_ERR_DEVICE," "DF_ST_ERR_DEVICE_TIMEOUT," or "DF_ST_ERR_FCU" indicates that an error was detected during erase suspend or programming suspend processing, but that valid data read processing is being executed.
- A return value of "DF_ST_ERR_LOST_ID" indicates a data missing condition when valid data for the data number specified in the argument cannot be found in the data flash.
- The driver functions that can be called by user applications using interrupts while the valid data read processing function is running are listed below. Calling functions other than those listed could cause malfunctions in the subsequent operation of the driver.
  - Data flash read protect cancel function
  - Data flash programming/erase protect function (Note that when this function is launched by an interrupt while another driver function is running, the already running driver function may impose limitations.)
  - Data flash programming/erase protect cancel function
  - User data size acquisition function
  - Blank area count acquisition function (Note that the value returned may not be correct if a programming suspended state occurs during a data update.)

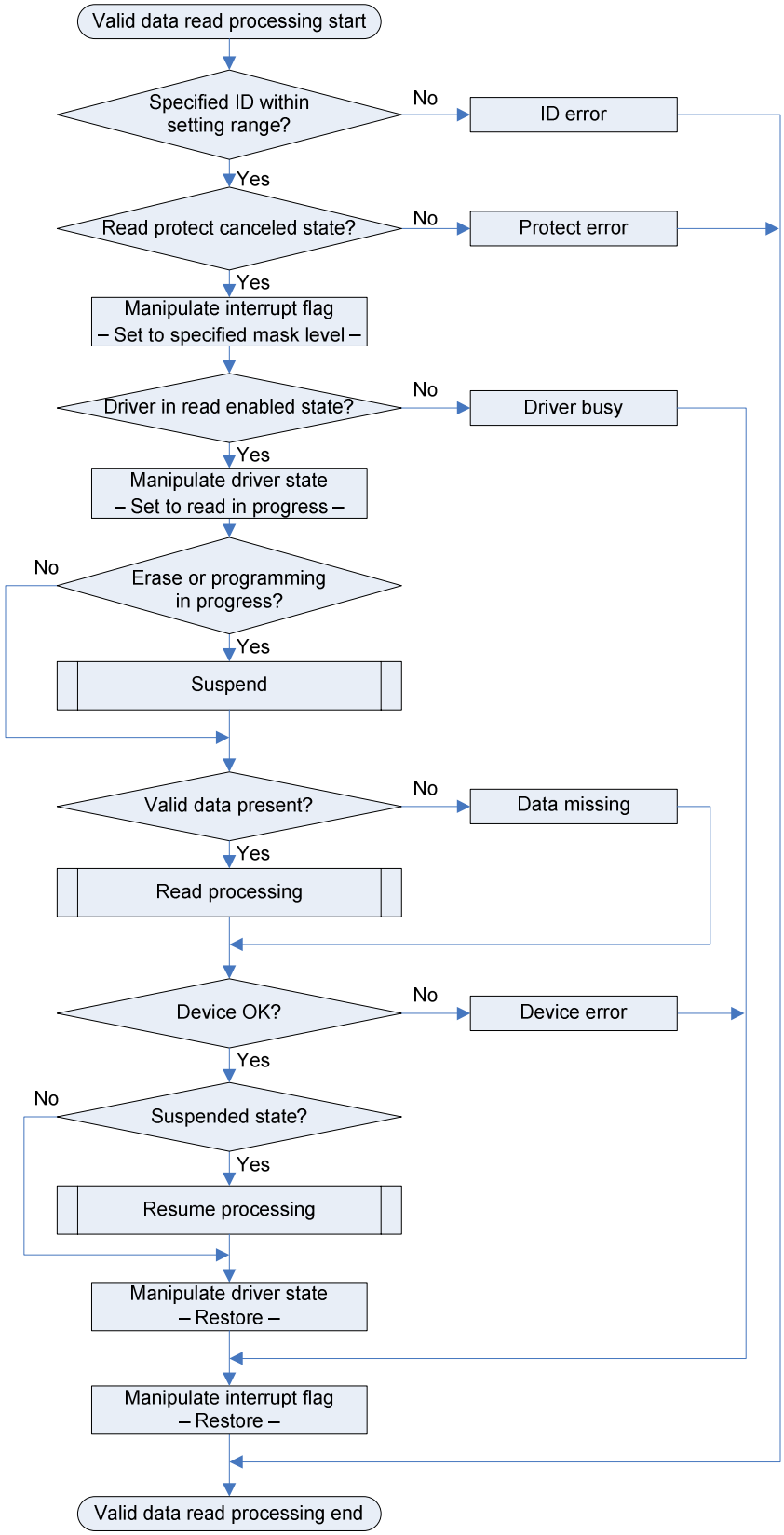

Figure 1.36 Valid Data Read Processing Function Outline Flowchart

1.10.5 Block Erase Processing Function

Table 1.10 provides a description of the block erase processing function, and figures 1.37 and 1.38 show an outline flowchart of its operation.

Table 1.10 Block Erase Processing Function Description

Function			
Block erase processing function **_s2 DF_Erase_block(void)**			
Arguments			
—			
Return values			
_s2	DF_ST_OK_ERASE_BLOCK_NONE	1 : No erase target block	
	DF_ST_OK	0 : Erase normal end	
	DF_ST_OK_REQUEST_ERASE	-1 : Erase normal end (Re-request call of block erase processing function)	
	DF_ST_WARNING_DRIVER_BUSY	-2 : Driver busy	(Wait request)
	DF_ST_WARNING_WRITE_ERASE_PROTECT	-5 : Erase prohibited state	(Request protect cancel)
	DF_ST_ERR_DEVICE	-8 : Error detected	(Data flash access prohibited)
	DF_ST_ERR_DEVICE_TIMEOUT	-9 : Error detected (Setting values in DF_user.c outside valid range)	
Description			
Erases the block registered as an erase target block.			
Usage notes			

- Call this function after driver initialization by the data flash driver initialization processing function has completed successfully.
- Erase processing of one block only of the blocks registered as erase target blocks ends when control returns from the function. Use the function's return value to determine whether or not the erase completed successfully.
- A return value of "DF_ST_OK_ERASE_BLOCK_NONE" indicates that no blocks are registered as erase target blocks.
- A return value of "DF_ST_OK_REQUEST_ERASE" indicates that an "erase all blocks" remains after processing of the function. When this occurs, call the function again to perform erase processing a second time. If the data update processing function is called without erasing the block first, a data update is performed if there are sufficient areas available for programming in the block, but if there are not sufficient areas available for programming, no data update is performed and a return value of "DF_ST_WARNING_NO_BLANK_AREA" is sent back.
- A return value of "DF_ST_WARNING_DRIVER_BUSY" indicates one of the following states and that block erase processing cannot be performed.
  - The block erase processing function has been called multiple times simultaneously.
  - The valid data read processing function is running.
  - The data update processing function is running.
  - The data flash driver initialization processing function or data flash formatting function is running.
  - The driver was already in an error state when the block erase processing function was called.
- A return value of "DF_ST_WARNING_WRITE_ERASE_PROTECT" indicates that programming/erase protection has been applied to the data flash. Call the data flash programming/erase protect cancel function to cancel programming/erase protection.
- A return value of "DF_ST_ERR_REQUEST_FORMAT" indicates that the driver has encountered a state where it cannot access data. When this occurs, call the data flash formatting function to rebuild the data area. This will cause all the data in the block group to return to the initial data (undefined value) state.
- A return value of "DF_ST_ERR_DEVICE" or "DF_ST_ERR_DEVICE_TIMEOUT" indicates that an error was detected during erase processing or the programming of block management information related to erase processing, preventing block erase processing from completing successfully. When this occurs, call the data flash error handler function to deal with the error. (Depending on the state of the device and the error circumstances, it may not be possible to recover normal operation in some cases.)
- The driver functions that can be called by user applications using interrupts while the block erase processing function is running are listed below. Calling functions other than those listed could cause malfunctions in the subsequent operation of the driver.
  - Valid data read processing function
  - Data update processing function
  - Data flash read protect function
  - Data flash read protect cancel function
  - Data flash programming/erase protect cancel function
  - Data size acquisition function
  - Blank area count acquisition function (Note that the value returned may not be correct, depending on the block erase processing state.)

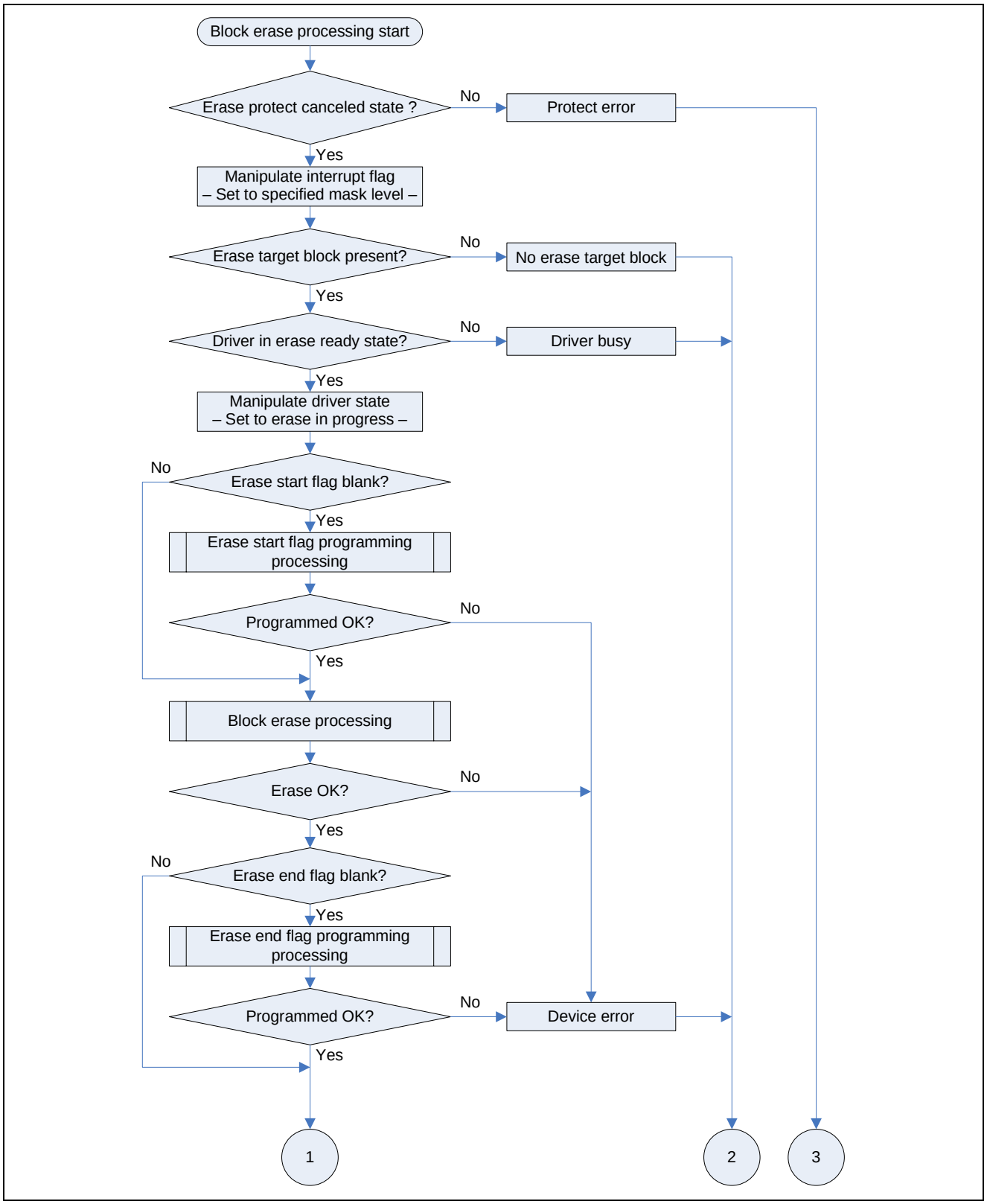

Figure 1.37 Block Erase Processing Function Outline Flowchart 1

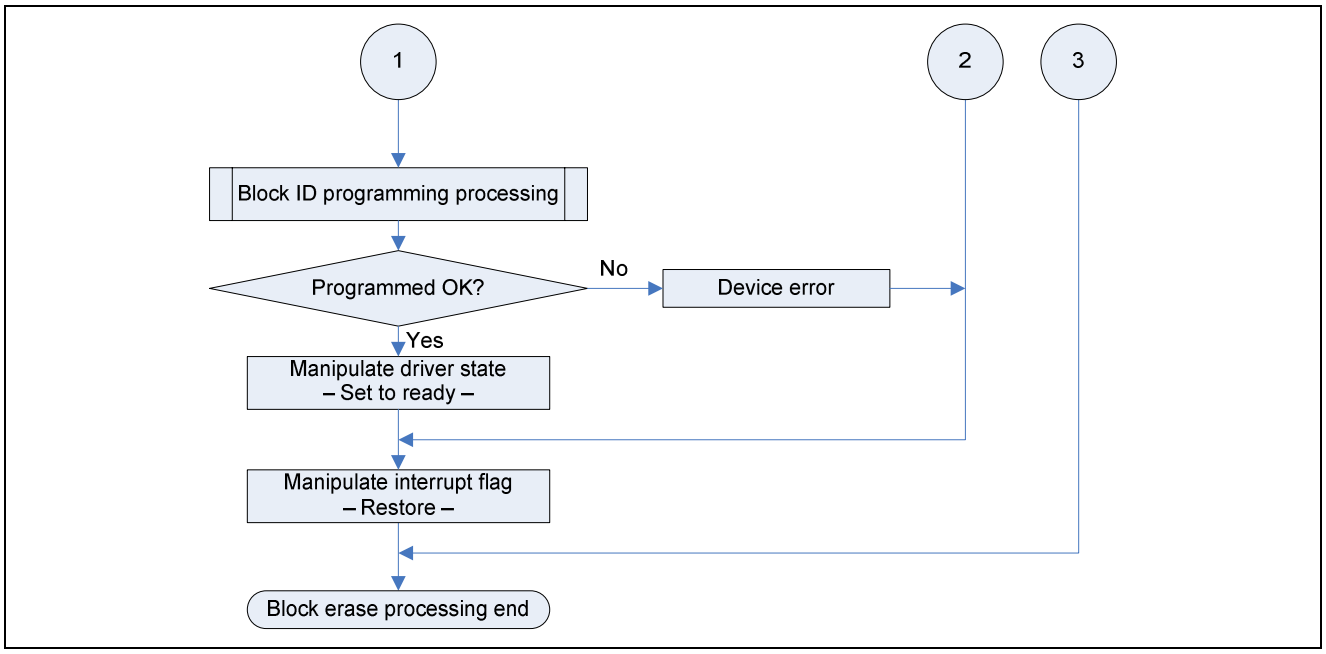

Figure 1.38 Block erase processing function Outline Flowchart 2

1.10.6 Error Handler Function

Table 1.11 provides a description of the error handler function, and figure 1.39 shows an outline flowchart of its operation.

Table 1.11 Error Handler Function Description

Function		
Error handler function **_s2 DF_Error_Management (void)**		
Arguments		
—		
Return values		
_s2	DF_ST_OK	0 : Error handling normal end
	DF_ST_OK_REQUEST_ERASE	-1 : Error handling normal end (Request call of block erase processing function)
	DF_ST_ERR_DEVICE	-8 : Device error
	DF_ST_ERR_DEVICE_TIMEOUT	-9 : Device error
	DF_ST_ERR_FCU	-10 : Device error
Description		
Performs processing according to the error state of the device (data flash) or data flash. In addition, calling this function when in the programming/erase suspended state resets the FCU and cancels the suspended state.		
Usage notes		
• If driver initialization processing does not complete, call the driver initialization processing function. • Error handling ends when control returns from the function. Use the function's return value to determine whether or not error handling completed successfully. (Depending on the state of the device and the error circumstances, it may not be possible to recover normal operation in some cases.) • A return value of "DF_ST_OK_REQUEST_ERASE" indicates that the driver indicates that error handling completed successfully and the driver encountered an "erase all blocks" request during the processing of this function. When this occurs, call the block erase processing function to erase the specified block. • A return value of "DF_ST_ERR_DEVICE," "DF_ST_ERR_DEVICE_TIMEOUT," or "DF_ST_ERR_FCU" indicates that normal operation could not be recovered after error handling.		

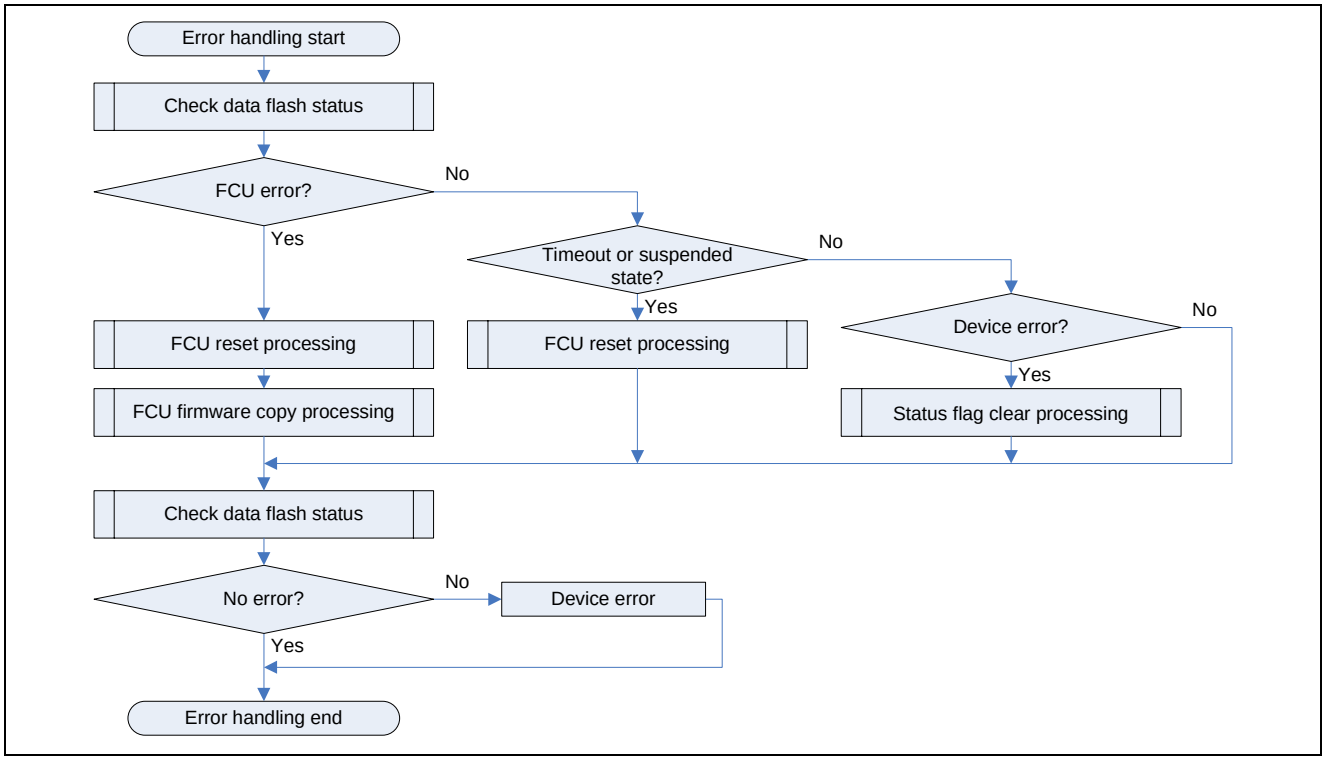

Figure 1.39 Error Handler Function Outline Flowchart

1.10.7 Data Size Acquisition Function

Table 1.12 provides a description of the data size acquisition function, and figure 1.40 shows an outline flowchart of its operation.

Table 1.12 Data Size Acquisition Function Description

Function			
Data size acquisition function **_s2 DF_Data_Size(_u1 data_id)**			
Arguments			
_u1	data_id	: Data number whose size is to be acquired	
Return values			
_s2	0 to 256: Data size		
	DF_ST_WARNING_INVALID_ARGUMENT	-6: Data number error	(Data number outside defined range)
Description			
Sends back as a return value the data size of the data number specified by an argument.			
Usage notes			
• A return value of “DF_ST_WARNING_INVALID_ARGUMENT” indicates that the data number specified in the argument is outside the defined range. The valid range of data numbers is 0 to (DF_DATA_ID_NUM – 1), according to the data count (DF_DATA_ID_NUM) defined in DF_user.h.			

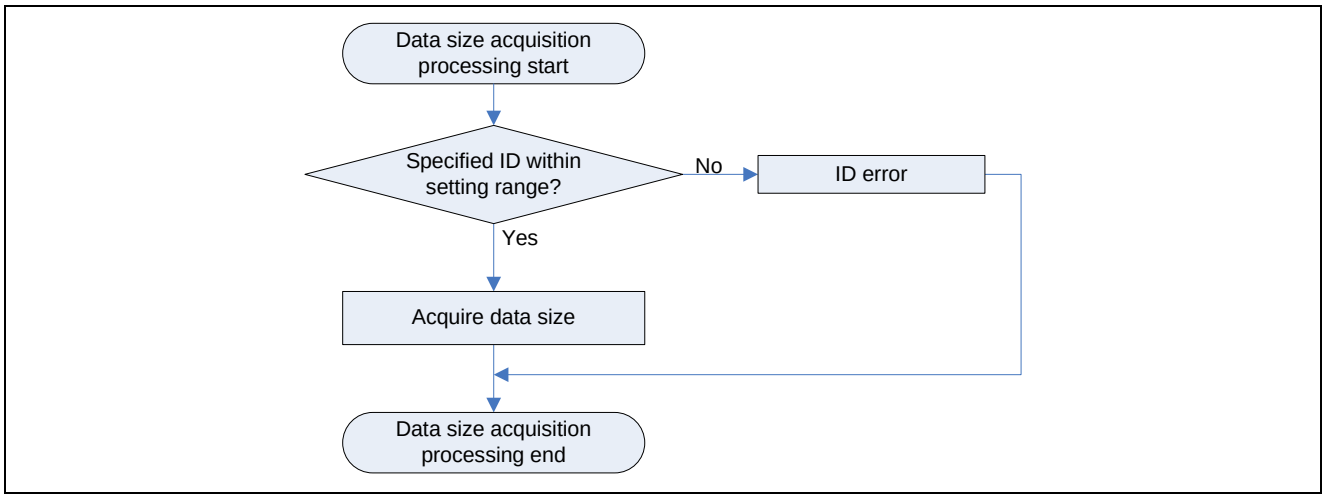

Figure 1.40 Data Size Acquisition Function Outline Flowchart

1.10.8 Blank Area Count Acquisition Function

Table 1.13 provides a description of the blank area count acquisition function, and figure 1.41 shows an outline flowchart of its operation.

Table 1.13 Blank Area Count Acquisition Function Description

Function		
Blank area count acquisition function **_s2 DF_Blank_Number(_u1 grp)**		
Arguments		
_u1 grp		: Block group number whose blank area count is to be acquired
Return values		
_s2	0 or above	: Number of areas available for programming
DF_ST_WARNING_INVALID_ARGUMENT	-6	: Block group number error (Block group number outside defined range)
Description		
Sends back as a return value the number of blank areas within the block being used for data updates in the block group specified by an argument.		
Usage notes		
• Call this function after driver initialization by the data flash driver initialization processing function has completed successfully. • Blank area count acquisition processing ends when control returns from the function. Use the function's return value to determine whether or not blank area count acquisition completed successfully. • Note that the value returned may not be correct if the blank area count acquisition function is called by an interrupt while another driver function is running, or if another driver function is called by an interrupt while the blank area count acquisition function is running. (When a driver function that increases or decreases the blank area count is running, the value returned may be that preceding the increase or decrease.) • A return value of “DF_ST_WARNING_INVALID_ARGUMENT” indicates that the block group number specified in the argument is outside the defined range. The valid range of block group numbers is 0 to (DF_GROUP_NUM – 1), according to the block group count (DF_GROUP_NUM) defined in DF_user.h. • The driver functions that can be called by user applications using interrupts while the blank area count acquisition function is running are listed below. Calling functions other than those listed could cause malfunctions in the subsequent operation of the driver. — Valid data read processing function — Data update processing function — Block erase processing function — Data flash read protect/protect cancel function — Data flash programming/erase protect/protect cancel function —Data size acquisition function		

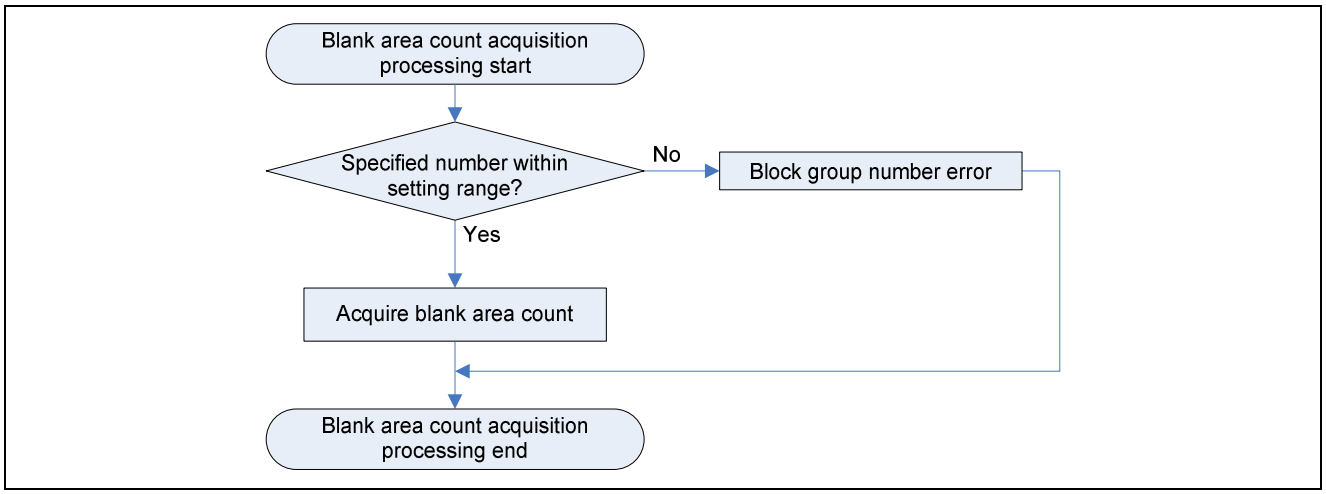

Figure 1.41 Blank Area Count Acquisition Function Outline Flowchart

1.10.9 Data Flash Read Protect Processing Function

Table 1.14 provides a description of the data flash read protect processing function, and figure 1.42 shows an outline flowchart of its operation.

Table 1.14 Data Flash Read Protect Processing Function Description

Function
Data flash read protect processing function **void DF_Read_Protect(void)**
Arguments
—
Return values
—
Description
Applies read protection to the data flash.
Usage notes
• Calling either of the driver functions listed below in the state after this function is run (read protect) could result in the called function failing to complete successfully. — Valid data read processing function — Data update processing function (when it accompanies data reclaim processing) If either of the above driver functions sends back a return value of “DF_ST_WARNING_READ_PROTECT,” call the data flash read protect cancel function to cancel read protection. In addition, if the data flash read protect processing function is called by an interrupt while any of the following driver functions is running, the driver function that was running may fail to complete successfully. — Data flash driver initialization processing function — Valid data read processing function —Data update processing function (when it accompanies data reclaim processing)

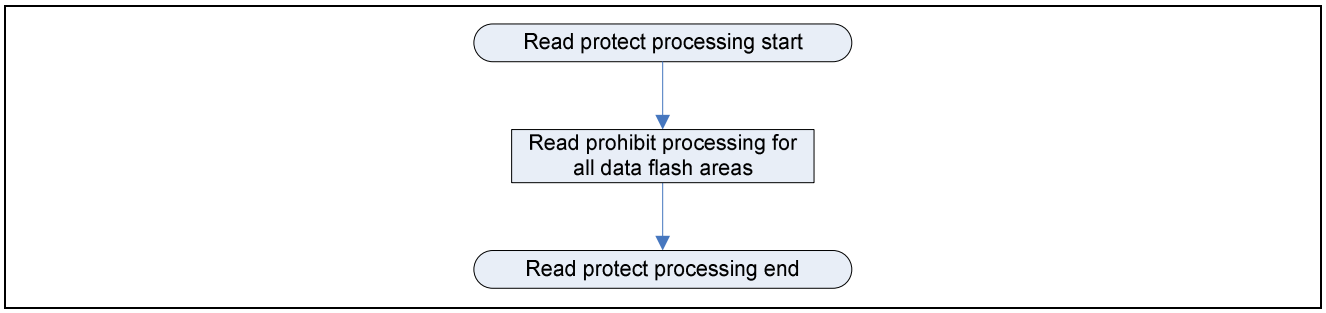

Figure 1.42 Data Flash Read Protect Processing Function Outline Flowchart

1.10.10 Data Flash Read Protect Cancel Processing Function

Table 1.15 provides a description of the data flash read protect cancel processing function, and figure 1.43 shows an outline flowchart of its operation.

Table 1.15 Data Flash Read Protect Cancel Processing Function Description

Function
Data flash read protect cancel processing function **void DF_Read_Protect_Cancel(void)**
Arguments
—
Return values
—
Description
Cancels data flash read protection.
Usage notes
—

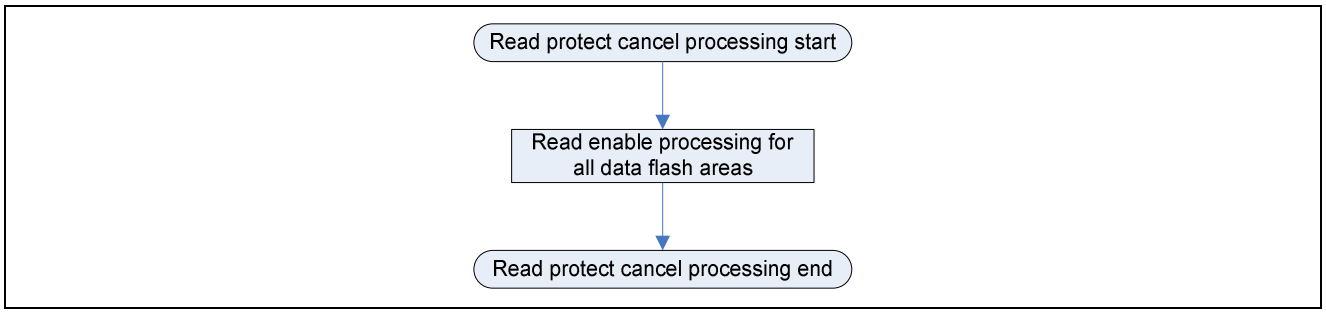

Figure 1.43 Data Flash Read Protect Cancel Processing Function Outline Flowchart

1.10.11 Data Flash Programming/Erase Protect Processing Function

Table 1.16 provides a description of the data flash programming/erase protect processing function, and figure 1.44 shows an outline flowchart of its operation.

Table 1.16 Data Flash Programming/Erase Protect Processing Function Description

Function
Data flash programming/erase protect processing function **void DF_WriteErase_Protect(void)**
Arguments
—
Return values
—
Description
Applies programming/erase protection to the data flash.
Usage notes
• Calling any of the driver functions listed below in the state after this function is run (programming/erase protect) could result in the called function failing to complete successfully. — Data flash formatting function — Data update processing function — Block erase processing function If either of the above driver functions sends back a return value of “DF_ST_WARNING_WRITE_ERASE_PROTECT,” call the data flash programming/erase protect cancel function to cancel programming/erase protection. In addition, if the data flash programming/erase protect processing function is called by an interrupt while any of the following driver functions is running, the driver function that was running may fail to complete successfully. — Data flash driver initialization processing function — Data flash formatting function — Data update processing function — Block erase processing function

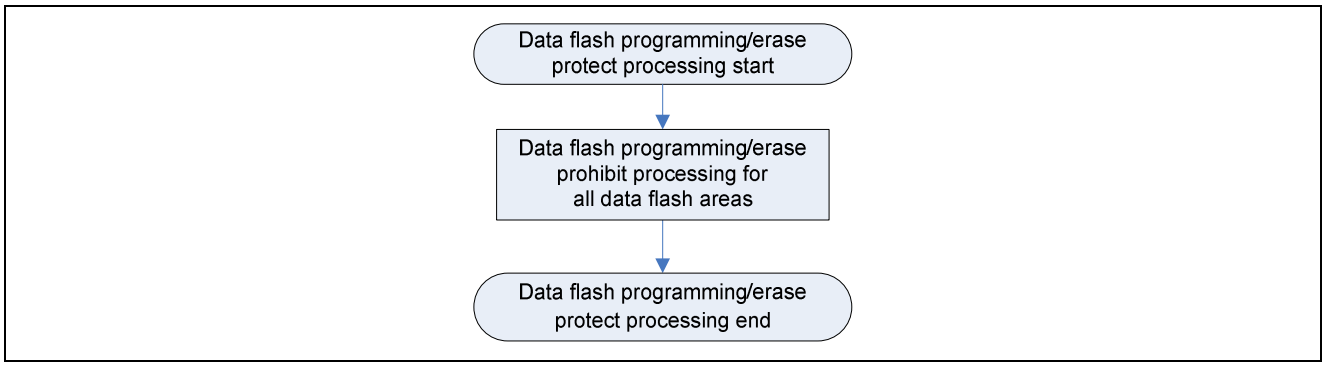

Figure 1.44 Data Flash Programming/Erase Protect Processing Function Outline Flowchart

1.10.12 Data Flash Write/Erase Protect Cancel Processing Function

Table 1.17 provides a description of the data flash programming/erase protect cancel processing function, and figure 1.45 shows an outline flowchart of its operation.

Table 1.17 Data Flash Programming/Erase Protect Cancel Processing Function Description

Function
Data flash programming/erase protect cancel processing function **void DF_WriteErase_Protect_Cancel(void)**
Arguments
—
Return values
—
Description
Cancels data flash programming/erase protection.
Usage notes
—

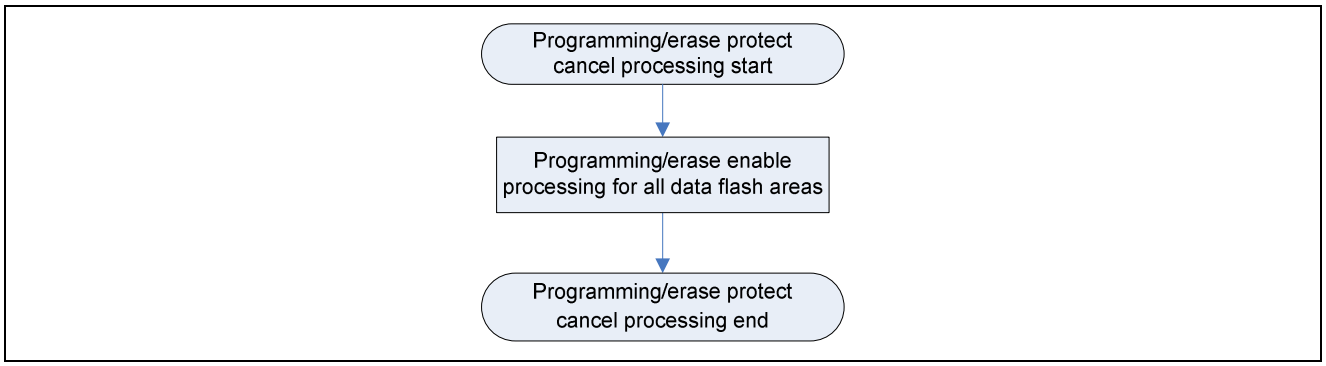

Figure 1.45 Data Flash Programming/Erase Protect Cancel Processing Function Outline Flowchart

1.11 Driver States and Callable Driver Functions

The driver stores specific values (variables) in the RAM to indicate changes in state accompanying the execution of driver functions.

1.11.1 Driver States

Table 1.18 lists the RAM values indicating driver states. The values of the variables in the table are defined in DF_drv.h.

Table 1.18 Correspondence Between Driver States and RAM Variables

Driver State				RAM Value Indicating Driver State		
				drv_status	drv_write_status	drv_erase_status
Before driver initialization				Undefine		
Standby	Normal state			DF_DRV_ST_READY	DF_WRITE_ST_READY	DF_ERASE_ST_READY
	Warning state	Unformatted		DF_DRV_ST_WARNING_REQ_FORMAT	DF_WRITE_ST_BUSY	DF_ERASE_ST_BUSY
	Error state	Error during initialization	Read-only	DF_DRV_ST_READY	DF_WRITE_ST_ERROR	DF_ERASE_ST_ERROR
			Access prohibited	DF_DRV_ST_ERROR_INIT		
		Error during formatting		DF_DRV_ST_ERROR_FORMAT		
		Programming error		DF_DRV_ST_READY		
		Erase error			DF_WRITE_ST_READY	
Busy	Driver initialization processing function running			DF_DRV_ST_INITIALIZATION	DF_WRITE_ST_BUSY	DF_ERASE_ST_BUSY
	Data flash formatting function running			DF_DRV_ST_FORMAT		
	Valid data read processing function running			DF_DRV_ST_BUSY		
	Data update processing function running			—		
	Block erase processing function running				DF_WRITE_ST_READY	
Suspended	Programming suspended			—	DF_WRITE_ST_SUSPEND	DF_ERASE_ST_BUSY
	Erase suspended				DF_WRITE_ST_READY	DF_ERASE_ST_SUSPEND

1.11.2 Driver Functions Executable in Each Driver State

Table 1.19 lists the driver functions that can be called (can be run) in each driver state.

Table 1.19 Correspondence Between Driver States and Executable Driver Functions

Driver State				Driver Function											
				DF_Drv_Init()	DF_Error_Management()	DF_Format()	DF_Read_Data()	DF_Write_Data()	DF_Erase_block()	DF_Blank_Number()	DF_Data_Size()	DF_Read_Protect_Cancel()	DF_WriteErase_Protect_Cancel()	DF_Read_Protect()	DF_WriteErase_Protect()
Before driver initialization				○	×*1									○*2	
Standby	Normal state			Δ*3	○	Δ*4	○	○	○*5	○				○*6	
	Warning state	Unformatted			○*7	○	Δ*8	Δ*9	×*10						
	Error state	Error during initialization	Read-only		○*11	×*16	○	×*12		○					
			Access prohibited			×*1									
		Error during formatting			○*7	Δ*3	×*10								
		Programming error			○*11	×*15	○	×*12		○					
		Erase error						○*13	×*12						
Busy	Driver initialization processing function running			×*15	×*1	×*15	×*1				○			×*14	
	Data flash formatting function running				×*16		×*10							○*6	×*14
	Valid data read processing function running						×*12							×*14	○*6
	Data update processing function running						Δ*18	×*12						×*14	
	Block erase processing function running						○*19	○*20	×*12	Δ*17				○*6	×*14
Suspended	Programming suspended				Δ*19		×*12							×*14	
	Erase suspended														

Notes: 1. Do not call these functions in this state because driver initialization has not completed.
2. Protection is canceled automatically within the driver initialization processing function.
3. Calling this function restarts the driver. Perform the necessary processing according to the return value.
4. Calling this function re-executes data flash formatting. All data becomes undefined.
5. If there is no erase target block, the driver sends back a return value of "DF_ST_OK_ERASE_BLOCK_NONE."
6. Subsequently, until protection is canceled, a protect error is returned when a driver function related to the protect state is run.
7. If after running the error handler function the return value is "DF_ST_OK" or "DF_ST_OK_REQUEST_ERASE," call the data flash formatting function to format the block.

8. Basically this function should not be called because the data flash is an in unformatted state, but it can be called if some data is thought to be missing and, after running the error handler function as an emergency measure, “DF_ST_OK”“DF_ST_OK_REQUEST_ERASE” is the return value. Note that a return value of “DF_ST_ERR_LOST_ID” will be sent back if valid data read processing is performed for a missing data number.
9. Basically this function should not be called because the data flash is an in unformatted state, but it can be called after running the error handler function as an emergency measure, as in note 7. If, together with reclaim processing, the reclaim data is missing, a return value of “DF_ST_ERR_LOST_ID” will be sent back.
10. Do not call these functions because the data flash is in an unformatted state.
11. Depending on the device state, it may not be possible to recover normal operation.
12. The driver sends “DF_ST_WARNING_DRIVER_BUSY” as a return value.
13. If there are no areas available for programming, the driver sends “DF_ST_WARNING_NO_BLANK_AREA” as a return value.
14. The currently running function may not be able to complete correctly. Do not call this function.
15. Do not call this function because it runs without checking the driver state.
16. Normal error detection during processing may not be possible. Do not call this function.
17. Depending on the driver’s running state, the return value may not be correct. (When a driver function that increases or decreases the blank area count is running, the value returned may be that preceding the increase or decrease.)
18. Valid data read processing is given priority, and if the driver is performing programming processing, it is put into the programming suspended state. Data update processing then restarts automatically after read processing ends. However, in the case of a data update in the erase suspended state, read processing is not performed and the driver sense a return value of “DF_ST_WARNING_DRIVER_BUSY.” If the data number to be read is being updated, the pre-update data is read.
19. Valid data read processing is given priority, and if the driver is performing erase processing or programming management information, those operations enter the suspended state. Block erase processing restarts automatically after read processing ends.
20. Data update processing is given priority, and if the driver is performing erase processing or programming management information, that operation enters the suspended state. Block erase processing restarts automatically after data update processing ends.

1.11.3 Execution Priority of Driver Functions

As described above, among the driver functions that can be called, the valid data read processing function, data update processing function, and block erase processing function confirm, when called, that the state is such that they can perform processing. If the state does not allow the function to run, the driver sends back a return value of "DF_ST_WARNING_DRIVER_BUSY."

The basic execution priority of driver functions within the driver is as follows, in order of descending priority.

1. Valid data read processing function
2. Data update processing function
3. Block erase processing function

When the RAM value indicating the driver state is one of those listed in table 1.20, the state is determined to be such that the driver can be executed.

Table 1.20 Correspondence Between Driver States and Executable Driver Functions

Driver Function	Driver State (RAM Value)*1 in Which Driver Can Be Executed		
	drv_status	drv_write_status	drv_erase_status
DF_Read_Data()	DF_DRV_ST_READY	State neither DF_WRITE_ST_BUSY nor DF_ERASE_ST_SUSPEND*2	
DF_Write_Data()	—	DF_WRITE_ST_READY	—
DF_Erase_block()	—	—	DF_ERASE_ST_READY

Notes: 1. See table 1.18, Correspondence Between Driver States and RAM Variables, for the transitions between RAM values indicating the driver state.
2. A state other than programming processing accompanying an erase suspended state.

Note: When a driver function other than the above is called, it is run without first checking the driver state.

1.11.4 Return Values of Driver Functions and Necessary Processing

Table 1.21 lists driver function return values and the processing that must be performed when each value is returned.

Table 1.21 Return Values of Driver Functions and Necessary Processing

Return Value	Description	Necessary Processing
DF_ST_OK_ERASE_BLOCK_NONE	No erase target block.	—
DF_ST_OK	Normal end.	—
DF_ST_OK_REQUEST_ERASE	Erase target block present.	Call the block erase processing function to erase the target block.
DF_ST_WARNING_DRIVER_BUSY	Driver busy state.	After processing of the running driver function ends, run it once again. Return from the interrupt processing that produced this return value.
DF_ST_WARNING_NO_BLANK_AREA	No areas available for programming remain in programming target block group.	Call the block erase processing function to erase the target block. If erase processing is in progress, end erase processing.
DF_ST_WARNING_READ_PROTECT	Data flash read protect state.	Call the data flash read protect cancel processing function to cancel read protection.
DF_ST_WARNING_WRITE_ERASE_PROTECT	Data flash programming/erase protect state.	Call the data flash programming/erase protect cancel processing function cancel programming/erase protection.
DF_ST_WARNING_INVALID_ARGUMENT	Argument error.	Check the argument value.
DF_ST_ERR_REQUEST_FORMAT	Data missing. Unformatted state.	Call the data flash formatting function to build a data structure to enable control by the driver. This causes the data for all data numbers to become undefined values.
DF_ST_ERR_LOST_ID	Data missing.	
DF_ST_ERR_DEVICE	Device error detected.	Call the error handler function. However, if the error was discovered during data update processing or valid data read processing while the block erase processing function was running, do not call the error handler function and perform the necessary processing indicated by the return value of the block erase processing function instead. In addition, if the error was discovered during valid data read processing while the data update processing function was running, do not call the error handler function and perform the necessary processing indicated by the return value of the data update processing function instead. If driver functions frequently send this return value, or if the error handler function sends this return value, the device may have reached the limit of its service life.
DF_ST_ERR_DEVICE_TIMEOUT	Device timeout error detected.	
DF_ST_ERR_FCU	FCU error detected.	
DF_ST_ERR_INIT	Driver initialization error.	After error handling ends, perform initialization processing once again. If the error handler function sends this return value, the device may have reached the limit of its service life.
DF_ST_ERR_FORMAT	Error during formatting of data flash.	After error handling ends, perform formatting once again. If the error handler function's return value is "DF_ST_ERR_INIT," or if this return value is sent back when formatting is retried, the device may have reached the limit of its service life.
DF_ST_ERR_PARAMETER_INVALID	Parameter error.	Check the setting values in DF_user.h and DF_user.c, and recompile the source code.

1.12 ROM and RAM Capacity

The usable ROM and RAM capacity of the sample data flash driver software is indicated below.

Note: The data presented here is for reference. Actual specifications may differ depending on the data count and the conditions used.

1.12.1 Development Tool Used

High-performance Embedded Workshop, Ver. 4.06.00

1.12.2 Compiler Used

Renesas Technology SHC/C++ Compiler, Ver. 9.03.00

1.12.3 ROM and RAM Capacity Used

Table 1.22 shows the ROM and RAM capacity when the sample data flash driver software is compiled using the conditions described above and a data count (number of data numbers) of eight. The figure for RAM capacity used does not include the stack size.

Table 1.22 ROM and RAM Capacity Used

ROM capacity used	7 KB	
RAM capacity used	1 KB	390 + 4 × number of data numbers

2. Making Settings When Using Sample Data Flash Driver Software

The necessary setting items for using the driver and the setting procedure are described below.

2.1 Setting Items

Table 2.1 lists the necessary setting items for using the driver.

Table 2.1 Driver Setting Items

Item	Description
Data count	The total quantity of data (data numbers) that can be stored in the data flash.
Operating frequency	The internal operating frequencies of the MCU used (SH7216). The operating frequency is specified in MHz units. • Data flash (FLD) operating clock • On-chip peripheral module clock
Interrupt disabled levels of driver functions*	The interrupt mask levels when each individual driver functions are running. • Interrupt mask level in driver initialization processing function • Interrupt mask level in data flash formatting function • Interrupt mask level in valid data read processing function • Interrupt mask level in data update processing function • Interrupt mask level in block erase processing function • Interrupt mask level in error handler function • Interrupt mask level in blank area count acquisition function

Note: * When FCU commands are issued during driver function processing as part of data flash programming, erasing, or blank checking, interrupts are prohibited (interrupt mask level 7) temporarily.

2.2 Setting the Data Count

Specify the quantity of data to be stored in the data flash by using the following definition in DF_user.h.

2.3 Setting the Data Size and Storage Area

Specify the site of individual data numbers and the areas where they will be stored by using the following definitions in DF_user.h. Storage area (block group) numbers are defined in DF_user.h. For the block structure, see 1.4, Internal Block Structure of Data Flash.

```
const unsigned short DF_DATA_SIZE_GROUP[ DF_DATA_ID_NUM ][2] =
{
/*                      Data size               Storage area (block group) */
/*                      0 to 256 bytes          GROUP_A, B                 */
/* Data number 0 */     0,                      GROUP_A,
/* Data number 1 */     1,                      GROUP_B,
/* Data number 2 */     8,                      GROUP_B,
/* Data number 3 */     9,                      GROUP_A,
/* Data number 4 */     128,                    GROUP_A,
/* Data number 5 */     256,                    GROUP_A,
};
```

2.4 Setting the Interrupt Prohibited Levels for Driver Functions

The interrupt mask levels during the execution of the corresponding driver functions can be specified by using the definitions in DF_user.h listed below.

Comment out any definitions corresponding to driver functions for which it is not necessary to set or change the interrupt mask level.

```
//                                  Interrupt mask level  Corresponding driver function
#define INT_LVL_df_driver_init      (7)                   // (DF_Drv_Init)
#define INT_LVL_df_format           (7)                   // (DF_Format)
//#define INT_LVL_df_data_read      (7)                   // (DF_Read_Data)
//#define INT_LVL_df_data_write     (6)                   // (DF_Write_Data)
//#define INT_LVL_df_data_erase     (5)                   // (DF_Erase_block)
#define INT_LVL_df_error_management (4)                 // (DF_Error_Management)
//#define INT_LVL_df_blank_number   (3)                   // (DF_Blank_Number)
```

When a driver function is running, the interrupt mask level is set (changed) to the interrupt mask level set as shown above, and the interrupt mask level reverts to its setting immediately previous to the setting (change) when control returns from the function.

3. Notes on Using Data Flash (FLD)

Some items must be borne in mind when using the data flash (FLD). See 28.8, Usage Notes, in the *SH7216 Group Hardware Manual* for details.

4. Reference Documents

- Software Manual
  SH-2A, SH2A-FPU Software Manual Rev.3.00
  (The latest version can be downloaded from the Renesas Technology Web site.)
- Hardware Manual
  SH7216 Group Hardware Manual Rev.1.01
  (The latest version can be downloaded from the Renesas Technology Web site.)

Website and Support

Renesas Technology Website
http://www.renesas.com/
Inquiries
http://www.renesas.com/inquiry
csc@renesas.com

Revision Record

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Mar.10.10	—	First edition issued

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.