NITRO-SDK

Accessing Backup Devices in AGB Game Paks for Nintendo DS™

Version 1.0.2

NTR-06-0439-001-B
Released: Review

Table of Contents

Tables

Figures

Revision History

Version	Revision Date	Description
1.0.2	6/6/2006	1 Revised descriptions. 4 Made corrections to the glossary notation.
1.0.1	4/7/2006	Changes made in NITRO-SDK.
1.0.0	12/27/2005	Initial version.

1 Introduction

The NITRO-SDK provides a set of APIs for accessing the backup device in the Game Boy Advance (AGB) Game Paks. This document describes how to use these APIs to access the backup device.

Please access the backup memory in the AGB Game Paks using the Nintendo-provided NITRO-SDK. DO NOT use your own programs for direct reading and writing.

Currently, APIs are available for the following backup memory devices:

- 256Kbit SRAM
- 512Kbit Flash
- 1Mbit Flash

4K and 64Kbit EEPROMs cannot be accessed from DS applications.

2 How to Access the Devices

The table shows how each kind of device is accessed. DO NOT use any method other than those described below.

Table 2-1 Device Access

Device	Method for Reading	Minimum Unit for Reading	Method for Writing	Minimum Unit for Writing
SRAM	`CTRDG_ReadAgbSram` `CTRDG_ReadAgbSramAsync`	1 byte	`CTRDG_WriteAgbSram` `CTRDG_WriteAgbSramAsync`	1 byte
FLASH	`CTRDG_ReadAgbFlash` `CTRDG_ReadAgbFlashAsync`	1 byte	`CTRDG_WriteAgbFlashSector` `CTRDG_WriteAgbFlashSectorAsync`	4 Kbytes

To access a backup device in an AGB Game Pak, you must first determine the type and the storage capacity of that device. Until this is determined, do not use any of the backup memory-related functions (including `CTRDG_IdentifyAgbBackup`).

To determine the type and the storage capacity of the backup memory device, first use the `CTRDG_GetAgbMakerCode` function to get the maker code. After determining which company made the device, use the `CTRDG_GetAgbGameCode` function to evaluate the game code to determine the type and the storage capacity of the backup memory.

Note: The current version of IS-NITRO-DEBUGGER cannot properly access the AGB backup device. In order to verify operations, you will need to use an actual DS console. A version of the IS-NITRO-DEBUGGER with this problem fixed is planned for release in the future.

3 Warnings for AGB Backup Access Functions

In addition to the content of this manual, the DS Programming Guidelines also contain important information, precautions, and rules regarding AGB backup memory. Be sure to read that document and follow its provisions when using the AGB backup access functions.

4 The AGB Backup Access Functions

This chapter describes the functions used to access AGB backup devices.

4.1 Functions common to all AGB backup devices

Whether the AGB is equipped with SRAM or Flash as the backup device, the device is assigned to the Game Pak RAM region (starting from 0x0A000000) in the memory map.

The AGB backup device access functions described in this manual have the following features:

- The wait cycle is adjusted internally by the functions; the developer does not need to worry about this factor.
- Each function is available in synchronous and asynchronous (*Async) versions. The asynchronous version is realized by executing the synchronous version in a thread created internally by the `CTRDG_Init` function.

Important: DO NOT use any of the AGB backup device access functions until the application has determined that an appropriate device is mounted in the Game Pak. The `CTRDG_IdentifyAgbBackup` function is not an exception.

4.1.1 Function Reference (common to all devices)

```
u16 CTRDG_IdentifyAgbBackup (CTRDGBackupType type)
```

Table 4-1 CTRDG_IdentifyAgbBackup

Arguments	CTRDGBackupType Type	Type of Backup Device that is Mounted in NITRO-CTRDG	
Return value	u16 result	Normal termination Identification error (when the appropriate device is not in the library)	⇒ 0 ⇒ Non-zero

This specifies the type of backup memory device that is mounted in NITRO-CTRDG.

If the backup device is a Flash device, the ID is read to determine which Flash device is mounted in the Game Pak. After the capacity of the Flash and the sector size is obtained, the access speed is configured and the appropriate functions for accessing the Flash are set. The obtained Flash data can be referenced using the global variable `flashType *flash`. For more information about `flashType`, please see the header file `ctrdg_flash.h`.

This function must be called once before any data can be written to or read from the backup device.

If the backup memory device that is mounted in NITRO-CTRDG is a 256Kbit SRAM chip, set the argument to `CTRDG_BACKUP_TYPE_SRAM`. If the device is a 512Mbit SRAM, set the argument to `CTRDG_BACKUP_TYPE_FLASH_512K`. If the device is a 1Mbit Flash, set the argument to `CTRDG_BACKUP_TYPE_FLASH_1M`.

If the device cannot be identified, the function will generate an error and the AGB backup access functions will be unusable.

Note: Specifying `CTRDG_BACKUP_TYPE_FLASH_512K` or `CTRDG_BACKUP_TYPE_FLASH_1M` for the argument initiates a process of writing to the device. If the argument does not match the type of backup device actually mounted in NITRO-CTRDG, the backup data in that device might be destroyed.

```
void CTRDG_SetTaskThreadPriority(u32 priority)
```

Table 4-2 CTRDG_SetTaskThreadPriority

Arguments	u32 priority	Priority of the Task Thread
Return value	None	

This changes the priority of the task thread that executes asynchronous functions.

4.2 256Kbit SRAM

SRAM is assigned to the Game Pak RAM region (starting from 0x0A000000) in the memory map.

The functions described in this manual for accessing SRAM have the following special features:

- These functions are used for both reading and writing. The minimum unit of access for both reading and writing is 1 byte.
- Even when the `CTRDG_WriteAgbSram` function is used to write to the backup memory device, there is no guarantee that the data was written correctly. To verify data was written correctly, execute `CTRDG_VerifyAgbSram` afterwards.

Important: All SRAM access functions internally lock the Game Pak bus for a fixed period.

4.2.1 Function Reference (for SRAM devices)

```
void CTRDG_ReadAgbSram (u32 src, void* dst, u32 size)
void CTRDG_ReadAgbSramAsync (u32 src, void* dst, u32 size, CTRDG_TASK_FUNC callback)
```

Table 4-3 CTRDG_ReadAgbSram and CTRDG_ReadAgbSramAsync

Arguments	u32 *src	Address (in memory map) of the Read Source in SRAM
	Void *dst	Address (in memory map) of the Work region storing the data that was read
	u32 size	Read size, in units of bytes
	`CTRDG_TASK_FUNC` callback	The callback function called at the end of the Read process (Only for asynchronous function)
Return value	None	

Takes the length of data specified by the size argument starting at the specified source SRAM address, and reads it into the Work region, starting from the `dst` address.

```
void CTRDG_WriteAgbSram (u32 dst, const void* src, u32 size)
void CTRDG_WriteAgbSramAsync (u32 dst, const void* src, u32 size, CTRDG_TASK_FUNC
callback)
```

Table 4-4 CTRDG_WriteAgbSram and CTRDG_WriteAgbSramAsync

Arguments	u32 *src	The Address of the Work Region for the Write Source
	void *dst	Address (in memory map) of the write destination in SRAM
	u32 size	Write size, in bytes
	CTRDG_TASK_FUNC callback	The callback function called at the end of the Write process (Only for asynchronous function)
Return value	None	

Takes data from the Work region the length of the size argument and writes it to SRAM, starting from the address in `dst`.

```
u32 CTRDG_VerifyAgbSram (u32 tgt, const void* src, u32 size)
void CTRDG_VerifyAgbSramAsync (u32 tgt, const void* src, u32 size, CTRDG_TASK_FUNC
callback)
```

Table 4-5 CTRDG_VerifyAgbSram and CTRDG_VerifyAgbSramAsync

Arguments	u32 *tgt	Pointer to the SRAM Address (in the Memory Map) of the Verify Target (the Data of the Write Destination)	
	void *src	The pointer to the address of the Work region for the verify source (the original data).	
	u32 size	The verify size, in bytes	
	CTRDG_TASK_FUNC callback	The callback function called at the end of the Verify process (Only for asynchronous function)	
Return value	u32 errorAdr (Only for synchronous function)	Normal termination Verification error	⇒ 0 ⇒ Error address on device side

This function verifies the data from the Work region src address with the data at the SRAM tgt address, for the number of bytes specified in the size argument.

In the synchronous version of the function, 0 is returned if the verification process ends normally. If there is a verification error, the function returns the address where the error occurred.

In the asynchronous version of the function, you can determine whether the Verify process was successful by referencing the `result` member of the `CTRDGTaskInfo` structure, which is an argument of the callback function that returns after this routine is called.

```
u32 CTRDG_WriteAndVerifyAgbSram (u32 dst, const void* src, u32 size)
void CTRDG_WriteAndVerifyAgbSramAsync (u32 dst, const void* src, u32 size,
CTRDG_TASK_FUNC callback)
```

Table 4-6 CTRDG_WriteAndVerifyAgbSram and CTRDG_WriteAndVerifyAgbSramAsync

Arguments	**u32 *dst**	**Pointer to the Write Destination SRAM Address (in Memory Map)**	
	void *src	Address of the write source Work region	
	u32 size	Write size, in bytes	
	`CTRDG_TASK_FUNC` callback	The callback function called at the end of the WriteAndVerify process (Only for asynchronous function)	
Return value	u32 errorAdr (Only for synchronous function)	Normal termination Verification error	⇒ 0 ⇒ Error address on device side

This function performs an internal verification process with `CTRDG_VerifyAgbSram` after writing the data with `CTRDG_WriteAgbSram`. If there is an error, the function retries up to `CTRDG_AGB_SRAM_RETRY_MAX` times. The maximum number of retries is defined in `ctrdg_sram.h`.

In the synchronous version of the function, 0 is returned if the verification process ends normally. If there is a verify error, the function returns the address where the error occurred.

In the asynchronous version of the function, determine whether the `WriteAndVerify` process was successful by referencing the `result` member of the `CTRDGTaskInfo` structure, which is an argument of the callback function that returns after this routine is called.

4.3 512Kbit, 1Mbit Flash

Flash is assigned to the Game Pak RAM region (0x0A000000 to 0xA00FFFF) in the memory map.
A 1-Mbit Flash comprises of two banks of 512 Kbits.

To accommodate the varying types and specifications of Flash used in Game Paks, the 512 Kbits are logically divided into 16 sectors of 32 Kbits (4 Kbytes) each and the Flash device is accessed in units of these sectors.

The functions for accessing Flash have the following special features:

- The access functions described in this manual are used for both reading and writing. The minimum unit of access is 1 byte for reading and one sector (4 KB) for writing.
- If the device is a 1M Flash device, the functions handle bank switching internally; the developer does not need to worry about this factor.
- Even if the `CTRDG_WriteAgbFlashSector` function ends normally, there is no guarantee that the data was written correctly. To verify whether the data was written correctly, execute `CTRDG_VerifyAgbFlash`.

Important: Precautions regarding the use of the Flash access functions:

- Processes required prior to using access functions (common to 512K and 1M Flash)

  DO NOT use any of the Flash functions until the application has determined that a Flash device is mounted in the Game Pak and the capacity of that device (either 512 Kbits or 1 Mbit) has been determined. This rule applies even to the `CTRDG_IdentifyAgbBackup` function.

- The operations of the access functions (common to 512K and 1M Flash)

  Presently, different makes of Flash can be used with a given game title. For this reason, the access functions are in a format that is common to all the makes of Flash. However, since different devices have different specifications, the internal operations of a given access function will differ depending on the make of Flash that exists in the Game Pak, and the function's execution times can differ substantially. Keep this in mind when developing your program and make sure that it operates properly regardless of the Flash manufacturer.

- Reading after an erase operation was interrupted (common to 512K and 1M Flash)

  For some makes of Flash, if the power is cut while data is being erased in a sector, the data in that sector might change every time it is read in the future and subsequent reads will be unstable. After the sector falls into this unstable read state, using VerifyFlash to verify the data after executing ReadFlash can generate an inequality error, with the data appearing different even when read from the same address a second time.

  Erasing the sector can restore it from this unstable read state. Execute a function that includes a process to erase the unstable sector (e.g., `CTRDG_EraseAgbFlash` or `CTRDG_WriteAgbFlashSector`), then try initializing and restoring the data.

- Interrupts (common to 512K and 1M Flash)

  Be aware that the Flash access functions (including the `CTRDG_IdentifyAgbBackup` function) internally disable all interrupts and lock the Game Pak bus for a set amount of time. Because of this behavior, when calling a Flash access function, do not use DMAs which start automatically for timing-specific operations like Direct Sound, V & H blank synchronization, display synchronization, and Game Pak requests.

- Timeout process using ticks (common to 512K and 1M Flash)

  The access functions listed below use tick counts for determining timeouts when accessing Flash. Be sure to call the `OS_InitTick` function before calling any of them.

  Pertinent functions:

  ```
  CTRDG_IdentifyAgbBackup,
  CTRDG_EraseAgbFlashChip,       CTRDG_EraseAgbFlashChipAsync
  CTRDG_EraseAgbFlashSector,     CTRDG_EraseAgbFlashSectorAsync
  CTRDG_WriteAgbFlashSector,     CTRDG_WriteAgbFlashSectorAsync
  ```

`CTRDG_WriteAndVerifyAgbFlash,  CTRDG_WriteAndVerifyAgbFlashAsync`

- Memory Bank Verification Errors (1M Flash only)

  1M Flash shares two 512Kbit banks in the Game Pak RAM region (0x0A000000 to 0xA00FFFF) in the memory map. Be aware that for the functions below, the returned address when there is a verification error does not contain bank information.

  The functions that return a verify error address are:

  ```
  CTRDG_VerifyAgbFlash
  CTRDG_WriteAndVerifyAgbFlash
  CTRDG_VerifyAgbFlashAsync
  CTRDG_WriteAndVerifyAgbFlashAsync
  ```

Important: There is generally a limit to the number of times that data can be overwritten on a Flash device, so you need to be aware of the ways you save data. For example, do not incorporate routines for frequent saving at places such as the screen where parameters are entered and do not write frequently to memory during communications. And, of course, do not incorporate an auto-save function in the game that constantly overwrites the game data. If you do not adhere to these rules, the lifespan of the Flash device will be shortened considerably.

Tip: Stretch out the interval between data overwrites, and use a number of sectors rather than always writing to the same sector. This will reduce the number of overwrites to any one sector.

Comment: Flash devices used with the AGB have warranties that guarantee a maximum of 10,000 cycles of erasing and overwriting. If data is saved to the Flash device at a rate of 30 times a day, that translates to the short lifespan of around one year.

4.3.1 Function Reference (for Flash devices)

```
void CTRDG_ReadAgbFlash (u16 sec_num, u32 offset, u8* dst, u32 size)
void CTRDG_ReadAgbFlashAsync (u16 sec_num, u32 offset, u8* dst, u32 size,
CTRDG_TASK_FUNC callback)
```

Table 4-7 CTRDG_ReadAgbFlash and CTRDG_ReadAgbFlashAsync

Arguments	u16 sec_num	Target Sector Number
	u32 offset	Offset in sector, in bytes
	u8 *dst	The address of the Work region storing the data that was read (Address in memory map)
	u32 size	Read size, in bytes
	`CTRDG_TASK_FUNC` callback	The callback function called at the end of the Read process (Only for asynchronous function)
Return value	None	

Starting from the `offset` address inside the target sector number, this function takes *size* bytes of data from the Flash device, and reads it into the Work region starting from the `dst` address.

This function will operate normally even if the specified read size straddles a sector boundary.

```
u16 CTRDG_EraseAgbFlashChip (void)
void CTRDG_EraseAgbFlashChipAsync (CTRDG_TASK_FUNC callback)
```

Table 4-8 CTRDG_EraseAgbFlashChip and CTRDG_EraseAgbFlashChipAsync

Arguments	**CTRDG_TASK_FUNC callback**	**The callback function called at the end of the EraseChip process (Only for asynchronous function)**	
Return value	u16 result (see note 1) (Only for synchronous function)	Normal termination Chip erase timeout error Internal device error	⇒ 0 ⇒ 0xc003 ⇒ 0xa003 (only for 1M Flash)

This function completely erases the entire Flash chip.

In the asynchronous version of the function, you can determine whether the Erase process was successful by referencing the `result` member of the `CTRDGTaskInfo` structure, which is an argument of the callback function that returns after this routine is called.

```
u16 CTRDG_EraseAgbFlashSector (u16 sec_num)
void CTRDG_EraseAgbFlashSectorAsync (u16 sec_num, CTRDG_TASK_FUNC callback)
```

Table 4-9 EraseAgbFlashSector and CTRDG_EraseAgbFlashSectorAsync

Arguments	**u16 sec_num**	**Target sector Number**	
	`CTRDG_TASK_FUNC` callback	The callback function called at the end of the EraseSector process (Only for asynchronous function)	
Return value	u16 result (see Note 1) (Only for synchronous function)	Normal termination Parameter error (secNo>0x0f) Sector erase timeout error Internal device error	⇒ 0 ⇒ 0x80ff ⇒ 0xc002 ⇒ 0xa002 (only for 1M Flash)

This function erases one sector's worth of the target sector number's data.

Because this routine is called inside `CTRDG_WriteAgbFlashSector`, it does not normally need to be called before writing data.

In the synchronous version of the function, a parameter error is returned if the target sector number is out of range.

In the asynchronous version of the function, you can determine whether the Erase process was successful by referencing the `result` member of the `CTRDGTaskInfo` structure, which is an argument of the callback function that returns after this routine was called.

```
u16 CTRDG_WriteAgbFlashSector (u16 sec_num,u8 *src)
void CTRDG_WriteAgbFlashSectorAsync (u16 sec_num,u8 *src, CTRDG_TASK_FUNC callback)
```

Table 4-10 CTRDG_WriteAgbFlashSector and CTRDG_WriteAgbFlashSectorAsync

Arguments	**u16 sec_num**	**Target sector Number**	
	u8 *src	Address of write source (Address in memory map)	
	`CTRDG_TASK_FUNC` callback	The callback function called at the end of the Write process (Only for asynchronous function)	
Return value	u16 result (see note 1) (Only for synchronous function)	Normal termination Parameter error (secNo>0x0f) Sector erase verification error Sector erase timeout error Internal device error when erasing Program timeout error Internal device error during program	⇒ 0 ⇒ 0x80ff ⇒ 0x8004 (only for Sanyo's Flash) ⇒ 0xc002 ⇒ 0xa002 (only for 1M Flash) ⇒ 0xc001 ⇒ 0xa001 (only for 1M Flash)

This function takes one sector's worth of data (4Kbytes) from the source address and writes it to the target sector number.

This routine internally calls the `CTRDG_EraseAgbFlashSector` function to erase the sector before writing the data. A parameter error is returned if the target sector number is out of range.

You can determine the number of remaining bytes while this routine is executing by referencing the global variable `flash_remainder`.

In the asynchronous version of the function, you can determine whether the Write process was successful by referencing the `result` member of the `CTRDGTaskInfo` structure, which is an argument of the callback function that returns after this routine is called.

```
u32 CTRDG_VerifyAgbFlash (u16 sec_num,u8 *src,u32 size)
void CTRDG_VerifyAgbFlashAsync (u16 sec_num,u8 *src,u32 size, CTRDG_TASK_FUNC
callback)
```

Table 4-11 CTRDG_VerifyAgbFlash and CTRDG_VerifyAgbFlashAsync

Arguments	**u16 sec_num**	**Target sector No.**	
	u8 *src	Address (in memory map) of verify source	
	u32 size	Verify size (bytes)	
	`CTRDG_TASK_FUNC` callback	The callback function called at the end of the Verify process (Only for asynchronous function)	
Return value	u16 errorAdr (Only for synchronous function)	Normal termination Verification error	⇒ 0 ⇒ Error address on device side

This function verifies data of *`size`* bytes in length from the source address with data in the target sector number.

The function will operate normally even if the specified verify size straddles a sector boundary.

In this function, 0 is returned if the verification process ends normally. If there is a verify error, the function returns the address where the error occurred. Note that the routine does not include a parameter check.

In the asynchronous version of the function, you can determine whether the Verify process was successful by referencing the `result` member of the `CTRDGTaskInfo` structure, which is an argument of the callback function that returns after this routine is called.

```
u32 CTRDG_WriteAndVerifyAgbFlash (u16 sec_num, u8 *src,u32 verifysize)
void CTRDG_WriteAndVerifyAgbFlashAsync (u16 sec_num, u8 *src,u32 verifysize,
CTRDG_TASK_FUNC callback)
```

Table 4-12 CTRDG_WriteAndVerifyAgbFlash and CTRDG_WriteAndVerifyAgbFlashAsync

Arguments	u16 sec_num	Target sector No.	
	u8 *src	Address (in memory map) of read source	
	u32 verifysize	Verify size (bytes)	
	`CTRDG_TASK_FUNC` callback	The callback function called at the end of the WriteAndVerify process (Only for asynchronous function)	
Return value	u16 result (see note 1) (Only for synchronous function)	Normal termination Parameter error (secNo>0x0f) Sector erase verification error Sector erase timeout error Internal device error when erasing Program timeout error Internal device error during program Verification error	⇒ 0 ⇒ 0x80ff ⇒ 0x8004 (only for Sanyo's Flash) ⇒ 0xc002 ⇒ 0xa002 (only for 1M Flash) ⇒ 0xc001 ⇒ 0xa001 (only for 1M Flash) ⇒ Error address on device side

This function internally calls `CTRDG_VerifyAgbFlash` to verify `verifysize` bytes of data after writing data with `CTRDG_WriteAgbFlashSector`. In other words, even though one sector's worth of data is written, the specified verify size can be smaller than one sector and can straddle sectors.

If an error occurs, the function will retry up to `CTRDG_AGB_FLASH_RETRY_MAX` times (the maximum number of retries defined in `ctrdg_flash.h`).

If there is a write error, the 32-bit return value will include one of the above 16-bit error codes. If there is a verify error, the 32-bit return value will be the address of the error in the device. Be aware of this difference when checking error codes.

In the asynchronous version of the function, you can determine whether the `WriteAndVerify` process was successful by referencing the `result` member of the `CTRDGTaskInfo` structure, which is an argument of the callback function that returns after this routine is called.

Note: When an error occurs, error codes are returned in accordance to the structure shown below.

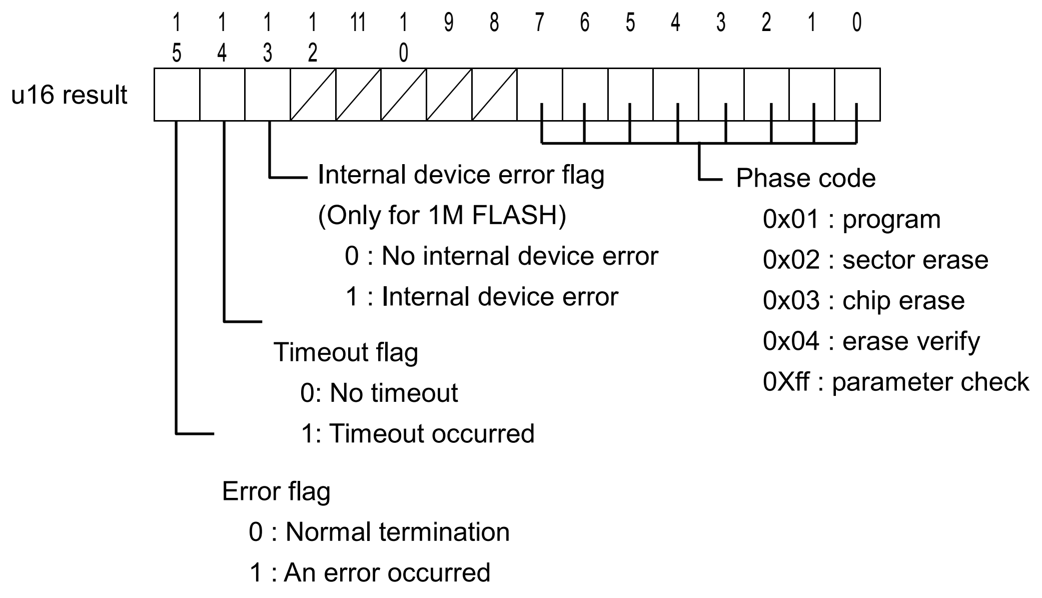


Figure 4-1 Error Code Details

5 Flow Charts Depicting Access to Each Device

This chapter presents flow charts that show the processes involved for accessing each backup memory device.

5.1 Flow Common to All Devices

The access functions are divided into groups based on the type of backup device. If functions inappropriate to a certain type of backup memory are used with that memory (for example, trying to use a Flash function to access SRAM), then this will become a source of bugs. This is why none of the access functions (including `CTRDG_IdentifyAgbBackup`) should be used until after the type of backup memory device and its storage capacity are determined.

To determine the type of backup memory device and its capacity, consult the following flow chart.

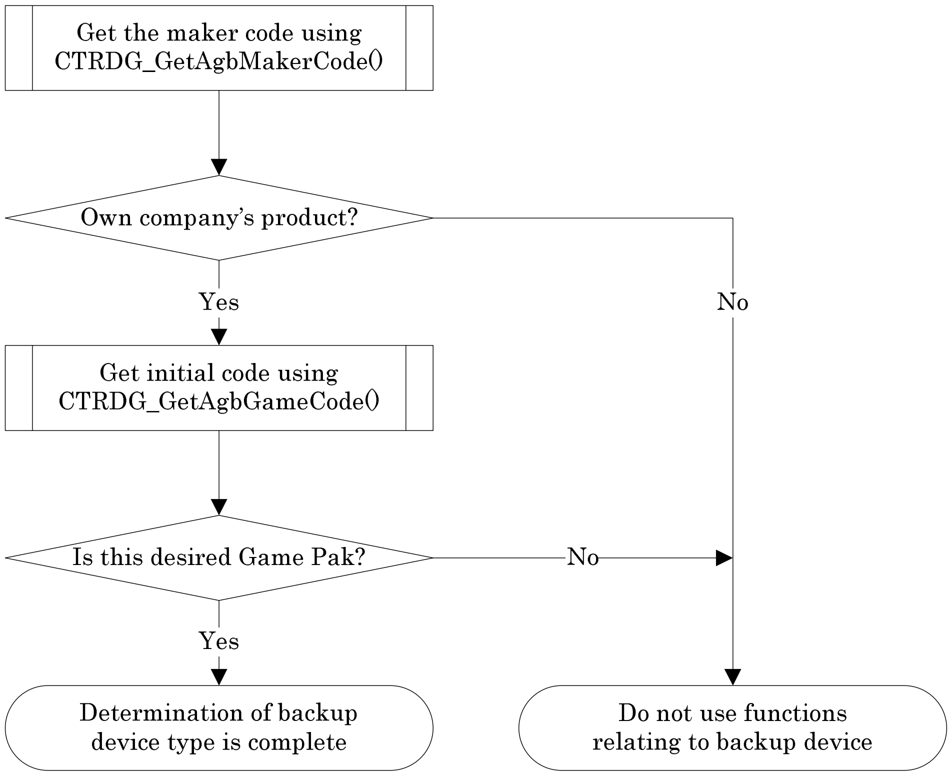

Get the maker code using CTRDG_GetAgbMakerCode()
Own company's product?
Yes
No
Get initial code using CTRDG_GetAgbGameCode()
Is this desired Game Pak?
No
Yes
Determination of backup device type is complete
Do not use functions relating to backup device

Figure 5-1 Common Flow

5.2 256Kbit SRAM

Before using SRAM access functions, first determine that the backup memory in the AGB Game Pak is an SRAM device. The process determining this is shown in the first flow chart.

After determining that the device type is SRAM, call the `CTRDG_IdentifyAgbBackup` function as shown Figure 5-2. After these processes are complete, the system is configured for proper reading and writing.

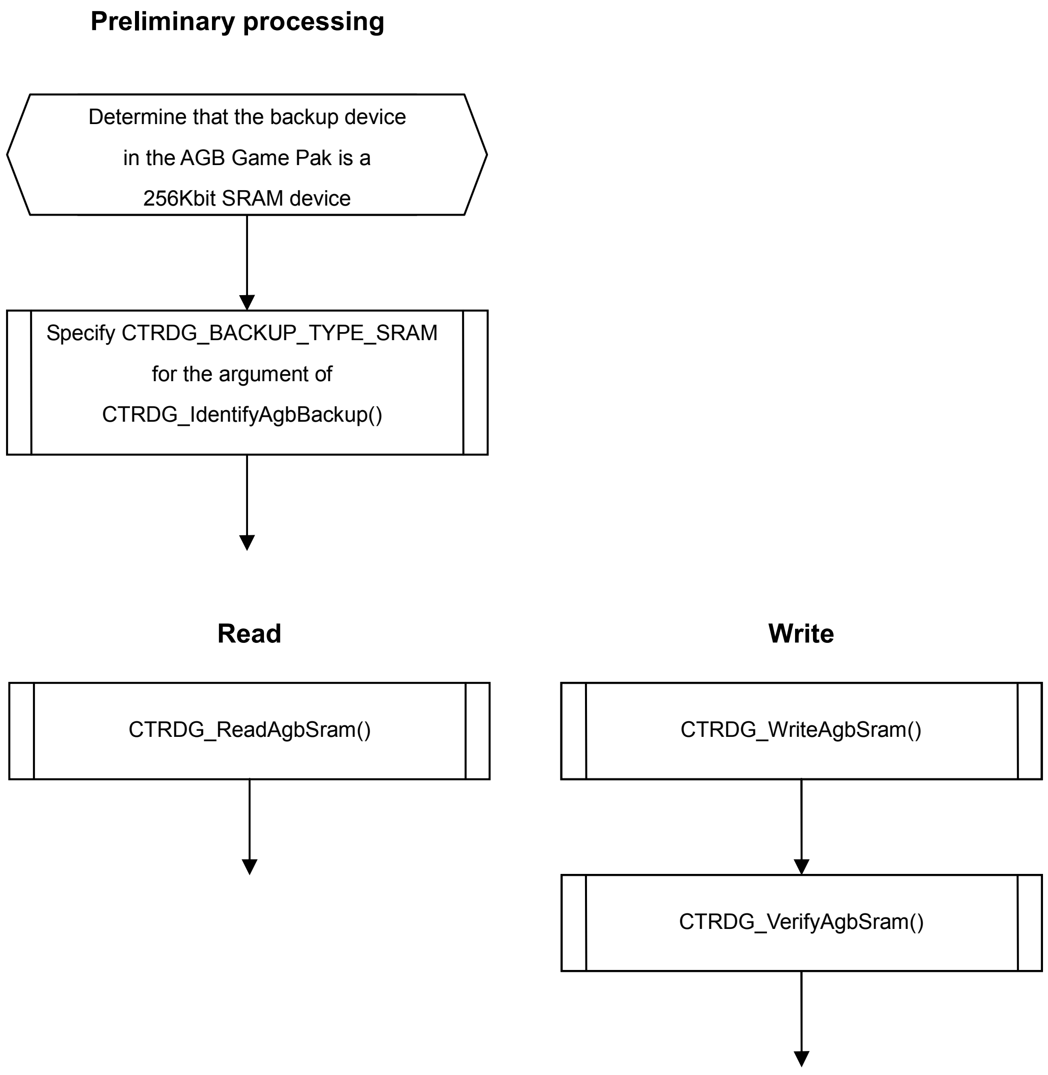
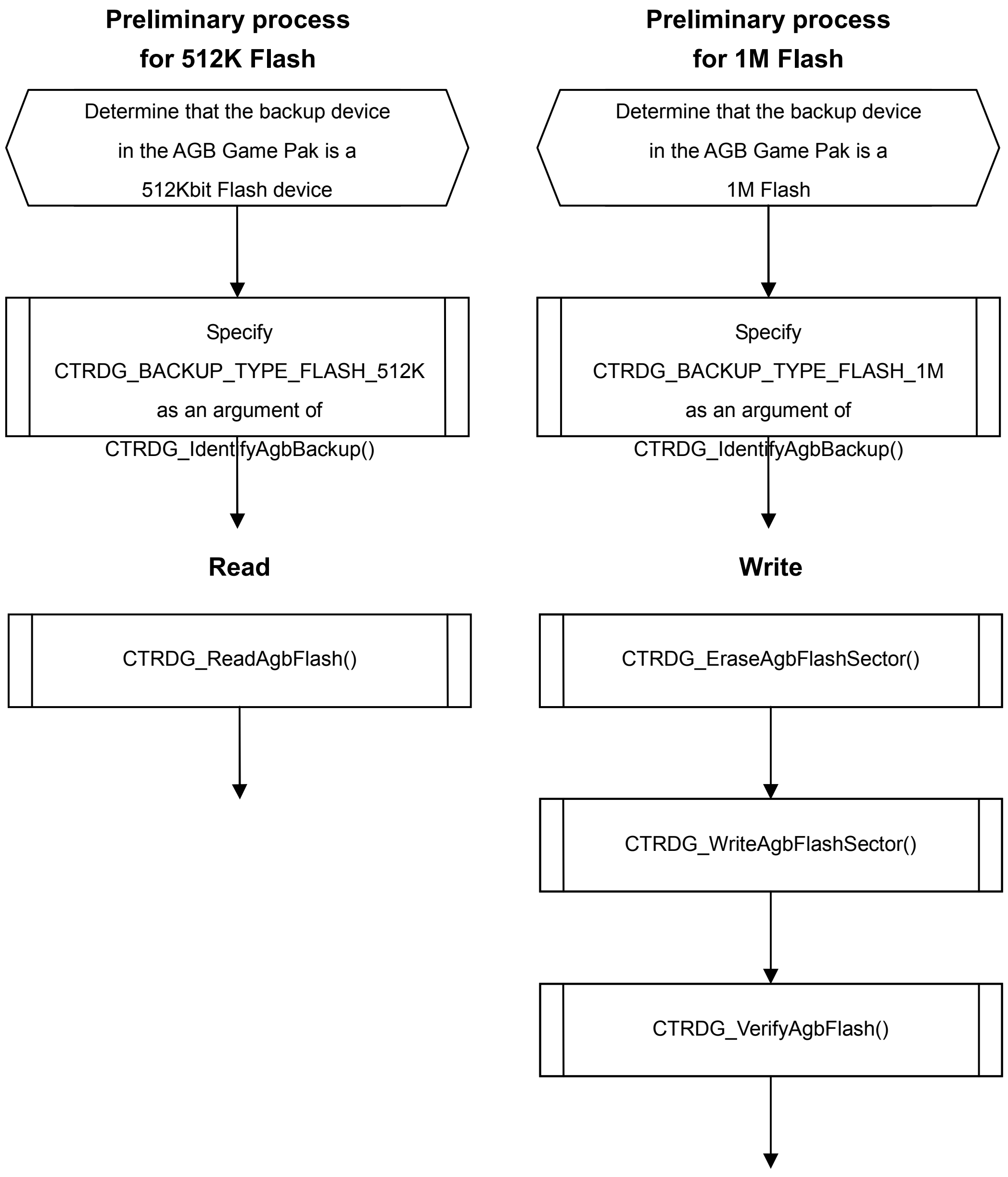

Preliminary processing
Determine that the backup device in the AGB Game Pak is a 256Kbit SRAM device
Specify CTRDG_BACKUP_TYPE_SRAM for the argument of CTRDG_IdentifyAgbBackup()
Read
Write
CTRDG_ReadAgbSram()
CTRDG_WriteAgbSram()
CTRDG_VerifyAgbSram()

Figure 5-2 256Kbit SRAM Process

5.3 512Kbit, 1Mbit FLASH

Before using any of these Flash access functions, first ascertain whether the backup memory in the AGB Game Pak is a Flash device and determine its storage capacity (this is similar to the process shown in the first flow chart).

After determining it is a Flash device, call the `CTRDG_IdentifyAgbBackup` function as shown in Figure 5-3. Once complete, the system is configured for proper reading and writing.



Preliminary process for 512K Flash
Determine that the backup device in the AGB Game Pak is a 512Kbit Flash device
Specify CTRDG_BACKUP_TYPE_FLASH_512K as an argument of CTRDG_IdentifyAgbBackup()
Read
CTRDG_ReadAgbFlash()
Preliminary process for 1M Flash
Determine that the backup device in the AGB Game Pak is a 1M Flash
Specify CTRDG_BACKUP_TYPE_FLASH_1M as an argument of CTRDG_IdentifyAgbBackup()
Write
CTRDG_EraseAgbFlashSector()
CTRDG_WriteAgbFlashSector()
CTRDG_VerifyAgbFlash()

Figure 5-3 512Kbit, 1Mbit FLASH Process