AGB Game Pak Backup Library Manual

Version 11

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

Manual Revision History

Revision Date	Version	Description of Revisions
2/22/2001	11	-Corrected the sentence structures in the opening paragraphs of the library function explanations for all devices. -Added a section for all devices explaining that there is no guarantee that the data has been written correctly even when the functions end normally, so please use a verify function to check the data. -For the 512K Flash, added a section in [Cautions when Using Flash Access Functions] warning of the danger of performing a direct write to the flash region from the user program because of the specifications of the Atmel flash. -For the 512K Flash, added "DMAs that start in sync with V blanks and H blanks" to the actions that are prohibited when the library functions are executing. -For the 4K EEPROM, added "DMAs that start in sync with V blanks and H blanks" to the actions that are prohibited when the library functions are executing. -For the 512K Flash, corrected the section about the timing of calls to SetFlashTimerIntr so that it now says: " ...before the library functions EraseFlashChip, EraseFlashSector or ProgramFlashSector are called..." instead of saying "...before any routine other than IdentityFlash is called...." -For the 4K EEPROM, corrected the section about the timing of calls to SetEepromTimerIntr so that it now says: "...before the write function Program EepromDword is called..." instead of saying "...before the various access routines are called...." -For the 512K Flash, corrected the section about prohibiting interrupts in [Cautions when Using Flash Access Functions] so that it now says: "... set the corresponding interrupt request enable register (this is not the IE register) to 'Disabled'..." instead of saying "set the corresponding IE register to 'Disabled'..." -For the 4K EEPROM, corrected the section about prohibiting interrupts in [Cautions when Using EEPROM Access Functions] so that it now says: "... set the corresponding interrupt request enable register (this is not the IE register) to 'Disabled'..." instead of saying "set the corresponding IE register to 'Disabled'..." -Corrected the operation flow in accordance with the above-described corrections. -Added a section relating to 8Mbit DACS. (There are no plans at the present time to market this device.)
1/30/2001	10	-For the 4K EEPROM , added a section explaining that the functions use DMA when accessing the device, so it is necessary to specify addresses with a 16-bit boundary for the u16 *src and u16 *dst arguments in each of the access functions.
1/26/2001	09	-Fixed the problem with 4K EEPROM for the arguments epAdr, ReadEepromDword and ProgramEepromDword becoming u32. -Added a description to the beginning about specifying EEPROM addresses when using each access function with the 4K EEPROM.
1/17/2001	08	-Description of return parameters for library functions with all devices.
1/16/2001	07	-For 256K SRAM, page 6 was added, “How to Avoid Loss of SRAM Data Due to CPU Lockup with Hot Plug-in of a Game Pak ".

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

Revision Date	Version	Description of Revisions
1/10/2001	06	-Added a flowchart for the function SramFast -Explained that when using 512K Flash made by Atmel, for a fixed period of time, all interrupts are prohibited when calling ProgramFlashSector and EraseFlashSector. Also direct sound cannot be used. For details see page 11, "Cautions when Using Flash Access Functions". In addition, the flowchart for access was updated. -For the 512K Flash, page 12 was added. "Cautions on Life of Flash Rewrite ". -Explained that with 4K EEPROM, using direct sound is prohibited when calling all access functions. For details see page 16, "Cautions when Using EEPROM Access Functions". In addition, the flowchart for access was updated. -For 4K EEPROM, added page 16, "Cautions on Life of EEPROM Rewrite".
12/13/2000	05	-Along with change to Library FLASH function and SRAM function, revised corresponding description. -Along with separating library version, made version number independent (Started using 05) -Along with separating library version, revised document. Added "Chapter 2 Updated Version of Library". Separated Revision History into Manual and Library
12/08/2000	ver.1.1.0	-Along with change to EEPROM function, fixed corresponding description. -Changed name of this manual from "Game Boy Advance Game Pak Backup Library" to "AGB Game Pak Backup Library Manual".
11/30/2000	ver.1.0.1	-Along with change to library filename, fixed corresponding description. -Changed start of EEPROM description. (Added "…Serial Connection…") -Added additional about valid range(0x00-0x3f) for the argument, epAdr, for EEPROM access function.
10/04/2000	ver.1.0.0	-Discontinued handling of 1M DACS. Added, "There are no plans at the present time to market this device." -Corrected the error in the explanation of the "size" argument of WriteSram for 256k SRAM.
09/05/2000 10/01/2000	ver.0.1.1	-Along with change to library, also revised the explanation regarding access to SRAM. -Added a Revision History page.
	ver.0.1.0Beta	-Release of first edition.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

Library Revision History

Revision Date	Revised Area	Version	Description of Revisions
2/22/2001	512K FLASH	ver.1.2.2	- Extended the wait time inside the IdentifyFlash function in consideration of use of the prefetch buffer.
1/10/2001	512K FLASH	ver.1.2.1	-Fixed write problem with FLASH made by Atmel. As a result, for a fixed period of time all interrupts are disabled when calling ProgramFlashSector and EraseFlashSector. For details see page 11, "Cautions when Using Flash Access Functions". -Fixed the bug with Sanyo's FLASH. If you were calling ProgramFlashSector and EraseFlashSector and an interrupt requiring more than 10ms of processing or a DMA occurred, even if the write succeeded the function may return a timeout error. -Disabled the use of direct sound when calling ProgramFlashSector and EraseFlashSector. (This is not a change of a library function, but a caution for the programmers.)
	4K EEPROM	ver.1.1.1	-Fixed bug with ReadEepromDword. Even when data was read normally, a return value other than 0 was returned. –Fixed bug of when you were calling ProgramEepromDword an interrupt or a DMA requiring more than 10ms of processing occurred even if the write succeeded the function may return a timeout error. -When calling all EEPROM access functions, use of direct sound was prohibited. (This is not a change of a library function, but a caution for the user). For details, see page 16, "Cautions when Using EEPROM Access Functions".
12/13/2000	Common		-Separated library version into separate devices
	256K SRAM (AgbSram.h)	ver.1.1.0	-Separation of library version only
	256 SRAM (AgbSramFast.h)	ver.1.0.0	-Set up high speed access function group. Defined in header AgbSramFast.h.
	512K FLASH	ver.1.2.0	-Added change to support of Macronics' FLASH -Changed so can directly call access functions with function pointer
	4K EEPROM	ver.1.1.0	-Separation of library version only
12/08/2000	Common	ver.1.1.0	
	512K FLASH		-Changed write routine to improve rewrite life -Support for Matsushita and Macronics devices
	4K EEPROM		-Changed so all interrupts are prohibited during DMA transfer while executing access functions.
11/30/2000	Common	ver.1.0.1	-Added "Agb" to beginning of filenames. -Added description of functions to function declaration area of each header file.
	256K SRAM		-Changed size of automatic variable used by each access function (increase)
	512K FLASH		-Changed size of automatic variable used by each access function (increase)
10/04/2000	Common	ver.1.0.0	-Added display of Nintendo of America Inc. copyright to header file.
10/01/2000	Common	ver.0.1.1	
	256K SRAM		-Changed so write uses library function WriteSRAM is used when writing.
09/05/2000	Common	ver.0.1.0 Beta	-Release of first edition.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

Table of Contents

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

1 Organization

This library is designed for use when accessing the backup device mounted in the Game Boy Advance (AGB) Game Pak.

As of now, the following three backup devices can be used in the AGB Game Pak. The library supports these devices.

256kbit SRAM
512kbit FLASH ROM
4kbit EEPROM
~~1Mbit DACS~~ (NOTE: There are no plans at the present time to market this device.)

The library is composed of the following five files:

AgbSram.h, AgbSramFast.h.....	Header file for 256kbit SRAM
AgbFlash.h.....	Header file for 512kbit FLASH ROM
AgbEeprom.h.....	Header file for 4kbit EEPROM
~~AgbDacs.h.....~~	~~Header file for 1Mbit DACS~~
libagbbackup.a.....	The library file containing the library function object files for use with the above devices.
AGBBackupLibraryManual.doc.....	This file, which explains the library.

2 Updated Version of Library

The newest versions of the libraries for each device are as follows:

256Kbit SRAM	AgbSram.h	ver.1.1.0
.....	AgbSramFast.h	ver.1.0.0
512kbit FLASHROM	AgbFlash.h	ver.1.2.2
4kbit EEPROM	AgbEeprom.h	ver.1.1.1
~~1Mbit DACS.....~~	~~AgbDacs.h~~	~~ver.1.1.1~~

Additionally, each of the header files above have a description like the one below at the beginning for each library version.

```
/*****************************************************************************/
/*          AgbSram.h                                                        */
/*          256kbit SRAM Library Header  ver.1.1.0                           */
/*          last modified 2000.11.30                                         */
/*          Copyright (C) 2000 NINTENDO Co., Ltd.                            */
/*****************************************************************************/
```


D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

3 How to Access Each Device

The table below shows how to access each device. Access with any procedure other than the ones shown below is prohibited.

Device	How to Read	Minimum Read Unit	How to Write	Minimum Write Unit
SRAM	"ReadSram" function "ReadSramFast" function	1 byte	"WriteSram" function "WriteSramFast" function	1 byte
FLASH	"ReadFlash" function	1 byte	"ProgramFlashSector" function	4 kbyte
EEPROM	"ReadEepromDword" function	8 byte	"ProgramEepromDword" function	8 byte
~~DACS~~	~~Direct read or "ReadDacs" (Either one is OK)~~	~~2 byte~~	~~"ProgramDacsSector" function "ProgramDacs_NE" function~~	~~Depends on the device and the function being used~~

4 Explanation of Library Functions

Explanations of the library functions for each device are given below.

4.1 256 Kbit SRAM

SRAM is allocated to the Game Pak SRAM region (0x0e000000~) in the AGB-CPU memory map.

The following features apply when the library functions are used to access SRAM:

* Library functions are used to access SRAM for both reading and writing. The minimum unit of access for both reading and writing is 1 byte.
* The wait cycle is adjusted inside each access function, so the developer does not need to worry about this.
* There is no guarantee that data has been written correctly to SRAM even when you use the WriteSram(or Fast) function. Thus, in order to be certain that data has been written correctly to SRAM please use VerifySram(or Fast) after writing the data.

****Explanation of differences in access functions described in AgbSram.h and AgbSramFast.h ****

Every time the functions ReadSram and VerifySram of AgbSram.h are called, in order to access SRAM, the required functions are transferred to the main unit's WRAM automatic variable region. Here they are branched and executed. Therefore, in cases where numerous bytes of data are read consecutively, the efficiency is not good.

In order to solve this problem, a group of functions mentioned in AgbSramFast.h are set up. By calling SetSramFastFunc, these functions transfer the necessary functions in advance to the main unit's WRAM static variable region. When executing the functions, it uses the pointer to the functions

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

transferred to the WRAM region. As a result the size of the static variable region used by the library becomes larger (approx. 300 bytes), but the access speed is high.

Depending on the objective, use either AgbSram.h or AgbSramFast.h.

**** How to Avoid Loss of SRAM Data Due to CPU Lockup with a Hot Plug-in of a Game Pak (Important)****

It was discovered while playing a game that if the Game Pak was removed or inserted with the power ON, there was a problem with the backup SRAM data being destroyed.

This hot plug-in is prohibited in the Instruction Booklet for the general consumer. However, if the backup data is lost, a large volume of complaints from consumers can be expected. In order to assure that this problem does not occur, the software needs to be revised. Therefore, the measures described below **(Support Level 1)** must be taken.

[Submitting Master ROM]

When submitting the Software Specification Sheet, add an item stating “Program to Avoid Loss of SRAM-Support Level 1” in the remarks area so that we can verify.

[Summary]

Use a Game Pak interrupt. Program it so that it enters an infinite loop when an interrupt occurs. Also program it to ensure that a Game Pak interrupt is always permitted with a program that runs with a Game Pak.

[Support Level 1 (Required)]

As a sample to support this, "simple.zip" is on the download area of noa-engineering, (http://www.noa-engineering.com). The following example describes the changes in an existing sample source.

1) Enable a Game Pak interrupt as much as possible during the main loop (including sub-routines).

(However, for programs that download using "Single Game Pak support, do not enable a Game Pak interrupt. There is a possibility that if the ROM registration of a Game Pak is checked with no Game Pak inserted, noise may be generated due to the open connector and a Game Pak interrupt would occur. **If a Game Pak is inserted, enable a Game Pak interrupt so that it will not be removed while accessing the Game Pak and abnormal data will not be read.**

When starting a Game Pak program from a downloaded program, the IE flag is cleared once. So, permit a Game Pak interrupt again, using the Game Pak program. It is possible to include it in the initialization routine when starting a Game Pak.)

Example: simple / main.c / AgbMain() / Line 96

```
*(vu16 *)REG_IME = 1;                          // Set IME
*(vu16 *)REG_IE = V_BLANK_INTR_FLAG            // Permit V-Blank Interrupt
                  | CASSETTE_INTR_FLAG;       // Permit Game Pak Interrupt
```

2) Enter an infinite loop when a Game Pak interrupt occurs.

Example: simple / crt0.s / intr_main()/ Line 95
reintr / crt0.s / intr_main()/ Line 102

```
        ands   r0, r1, #CASSETTE_INTR_FLAG   @ Game Pak Interrupt
loop:   bne    loop
jump_intr:
```


D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

3) Prioritize the Game Pak interrupt check.

(With multiple interrupts, there is no need to raise the priority of Game Pak interrupts)

Example: simple / crt0.s / intr_main() / Line 54

```
and    r1, r2, r2, lsr #16            @ r1: IE & IF
ands   r0, r1, #CASSETTE_INTR_FLAG    @Game Pak interrupt
loop:  bne loop
mov    r2, #0
ands   r0, r1, #V_BLANK_INTR_FLAG     @V-Blank interrupt
bne    jump_intr
add    r2, r2, #4


ands   r0, r1, #KEY_INTR_FLAG         @Key interrupt
jump_intr:
```

[Support Level 2 (Recommended)]

1) Carry out **[Support Level 1]**.

2) Execute the routine with the RAM that the Game Pak interrupts are prohibited on.

(For example, routines that change the IME to OFF temporarily).

3) Minimize interrupt processing.

Example: multi_sio / crt0.s / intr_main() / Line 60
main.c / AgbMain() / Line 119
VBlankIntr() / Line 150

For the V-blank interrupt routine, you should only update the sound DMA or set the interrupt flags. An update of VRAM or OAM should be done during V-blank in the main loop.

[Support Level 3 (Recommended)]

As a sample to support this, "reintr.zip" has been uploaded to the download site of noa-engineering (http://www.noa-engineering.com). The following example describes only the changes according to the existing sample source.

1) Carry out **[Support Level 2]**.

2) Select either of the methods described below:

2a) By using multiple interrupts, you can avoid lockups on the Game Pak during the processing of interrupts. Permit multiple interrupts with Game Pak interrupts.

Example: reintr / crt0.s / intr_main() / Line 105

```
ldr  r1, =CASSETTE_INTR_FLAG|TIMER0_INTR_FLAG   @Set IE <- Select multiple interrupts
and  r1, r1, r12
strh r1, [r3]
```

2b) Execute all interrupt processing on RAM.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

Library Functions

Functions described in AgbSram.h

`void` **`ReadSRAM(u8 *src, u8 *dst, u32 size)`**

<Arguments>	u8 *src	: Source address in SRAM
		(The address in the AGB memory map)
	u8 *dst	: Destination address in Work RAM
		(The address in the AGB memory map)
	u32 size	: The size of the data in bytes
<Return Value>	None	

This function reads "size" bytes of data from the argument-specified SRAM address into the work RAM starting from the "dst" address.

`void` **`WriteSRAM(u8 *src, u8 *dst, u32 size)`**

<Arguments>	u8 *src	: Source address in Work RAM
	u8 *dst	: Destination address in SRAM
		(The address in the AGB memory map)
	u32 size	: The size of the data in bytes
<Return Value>	None	

This function writes "size" bytes of data from the argument-specified work RAM address to SRAM starting from the "dst" address.

`u32` **`VerifySRAM(u8 *src, u8 *tgt, u32 size)`**

<Arguments>	u8 *src	: Pointer to the verify origin work RAM (the original data)
	u8 *tgt	: Pointer to the verify target SRAM address
		(The written data)
	u32 size	: The size of the area to verify in bytes
<Return Value>	u32 errorAdr	: Normal end → 0
		Verify Error → Error address on device side

This function verifies "size" bytes of original data from the address "src" and written data from the SRAM address "tgt." It returns 0 if verify terminates normally. If there was a verification error, it returns the address where the error occurred.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

Functions described in AgbSramFast.h

`void` **`SetSramFastFunc()`**

<Arguments> None
<Return Value> None

The ReadSram and VerifySram routines need to operate on WRAM, so they are transferred to the WRAM static region. After this, these functions are called through the pointers (*ReadSram)() and (*VerifySram)(). SetSramFastFunc() is called prior to accessing SRAM (including read).

`void` **`(*ReadSramFast)`**`(u8 *src, u8 *dst, u32 size)`

<Arguments> u8*src :Source address in SRAM
(The address in the AGB memory map)
u8*dst : Destination address in Work RAM
(The address in the AGB memory map)
u32 size : The size of the data in bytes
<Return Value> None

This function reads "size" bytes of data from the argument-specified SRAM address into the work RAM starting from the "dst" address.

`void` **`WriteSramFast`**`(u8 *src, u8 *dst, u32 size)`

<Arguments> u8*src : Source address in Work RAM
u8*dst : Destination address in SRAM
(The address in the AGB memory map)
u32 size : The size of the data in bytes
<Return Value> None

This function writes "size" bytes of data from the argument-specified WRAM address to SRAM starting from the "dst" address.

NOTE: The contents of this function are the same as WriteSram, but if you are using the function group from AgbSramFast.h, use this function.

`u32` **`(VerifySramFast)`**`(u8 *src, u8 *tgt, u32 size)`

<Arguments> u8 *src : Pointer to the verify origin work RAM (the original data)
u8 *tgt : Pointer to the verify target SRAM address
(The written data)
u32 size : The size of the area to verify in bytes
<Return Value> u32 errorAdr : Normal end → 0
Verify Error→ Error address on device side

This function verifies "size" bytes of original data from the address "src" and written data from the SRAM address "tgt." It returns 0 if verify terminates normally. If there was a verification error, it returns the address where the error occurred.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

4.2 512 Kbit Flash ROM

Flash memory is allocated to Game Pak SRAM region (0x0e000000~) in the AGB-CPU memory map.

In order to handle the differences in the specifications of the various kinds of flash memory devices that can be used, the flash memory area is divided into 16 logical sectors of 32 Kbits (4 Kbytes) each, and the device is accessed in units of these sectors.

The following features apply when the library functions are used to access Flash ROM:

* Library functions are used to access Flash ROM for both reading and writing. The minimum unit of access is 1 byte for reading and 1 sector (4 Kbytes) for writing.
* The library functions EraseFlashChip, EraseFlashSector and ProgramFlashSector use any one of the timers 0~3 for time out processing.
* The wait cycle is adjusted inside each access function, so the developer does not need to worry about this.
* There is no guarantee that data has been written correctly to Flash ROM even when ProgramFlashSector ends normally. Thus, in order to be certain that data has been correctly written please use VerifyFlashSector after writing the data.

**** Cautions when Using Flash Access Functions (Important) ****

Currently flash memory from multiple manufacturers is being used. Therefore, a common format of library functions is being used with all flash memory. However, the specifications for each device vary so the operation within each library function is different depending on the type of flash. Also there is a great variance in the execution time.

When programming, be aware of this difference in devices and make sure that the operation is always normal, with all types of flash.

When using Atmel's flash, the probability of noise problems with direct sound is high because interrupts are often disabled by the library. Therefore, when calling library functions, do not use direct sound.

Also, do not use DMA synchronized to V blank, H blank during calls to ProgramFlashSector or EraseFlashSector because this can cause a write failure with Atmel's flash, as described below.

In addition, note that according to the Atmel flash specifications, the device cannot be accessed for a set period (max 20ms) after data has been written to the flash with some method other than the official write command. For this reason, do not have the user program write directly to the Flash area. Doing so may cause subsequent library functions to operate abnormally.

<Atmel's Flash>

With the EraseFlashSector and ProgramFlashSector functions when using Atmel's flash, according to the device's specifications, 4 Kbyte erases and writes are completed in 128 bytes. This device has no erase, so with EraseFlashSector it writes 4 Kbyte portions of FFh in the function. The operation itself is the same as ProgramFlashSector.

Additionally, the time required for each 128 byte erase and write is set at a maximum of 20ms. The program execution time for the 128 byte write in this function is an actual value of approximately 680us (measured with program 1st access 3 cycles / 2nd access 1 cycle, prefetch off). Thus, if you calculate the total execution time its equivalent to: 4K/128 * (20ms+680us)= approx. 660ms.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

Furthermore, according to this device's specifications, the write pulse interval when writing each byte of the 128 bytes must be 150us or less. If it exceeds 150us, the device will stop the write.

Therefore, when calling ProgramFlashSector and EraseFlashSector (period calculated to be approx. 660ms), if an interrupt or a DMA requiring processing of more than 150us occurs in the middle of writing this data, the write will fail. As shown in the diagram below, to counter this, all interrupts within library functions are prohibited for a period of 680µs.

Write Function Processing (Start→)	128byte write(1)	Wait for end of write	128byte write(2)	Wait for end of write	. . . .	128byte write(32)	Wait for end of write	End
Required Time	680us	Max20ms	680us	Max20ms		680us	Max20ms	
Interrupt Status	Prohibited	Allowed	Prohibited	Allowed		Prohibited	Allowed	

To disable interrupts, the IME register is set to 0. Therefore, interrupts enabled before calling this library's functions may be delayed a maximum of 680µs. If the delay causes abnormal operation, be sure and set the corresponding interrupt request enable flag in each control register (this is not the IE register) to "Disabled" before calling the library functions.

<Sanyo's Flash>

Operation of the library functions when using Sanyo's flash is as follows. In the library functions no interrupts are prohibited.

Write Function Processing (Start→)	erase&erase Wait for End	Wait for end of write&write in 1byte units for 4 kbytes	End
Required Time	Max20ms	Wait time for end of 1byte write, Max20us(Not including program operation time)	
Interrupt Status	Allowed		

****Cautions on Life of Flash Rewrite (Important)****

Usually, flash memory is limited by the number of rewrites per sector so you need to be careful how frequently save data is written.

For example, do not do things like frequent saves with the parameter input screen, or frequent writes to the memory during communication.

As is obvious, you cannot use flash memory with games that do auto-saves which frequently carry out rewrites.

If you do not follow these guidelines, you may significantly shorten the life of the product.

<Reference Techniques>

Lengthen the data rewrite interval. Do not write to the same sector. Instead use multiple sectors and try to decrease the number of rewrites for the same sector.

<Remarks>

For the flash memory used with AGB, the manufacturer guarantees a minimum of 10,000 rewrites per 1 sector. This is equal to a life of about 1 year with 30 saves per day.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

Library Functions

`u16` **`IdentifyFlash()`**
<Arguments> None
<Return Value> u16 result : Normal End → 0
Identify Error (When an applicable device is not in the library) →1

This function reads the flash memory ID, determines which kind of flash memory device is installed in the Game Pak, gets the flash memory capacity and sector size, sets the access speed and the access functions that can be used with the flash memory device. The obtained flash data can be referenced using the global variable `flashType*flash`. (For details about flashType please read the header file AGBFlash.h)

This function is called once prior to accessing the flash memory device (including prior to reading the device).

If the device cannot be identified, an error is returned and the subsequent access functions cannot be used.

`u16` **`SetFlashTimerIntr`**`(u8 timerNo, void (**IntrFunc)(void))`
<Arguments> u8 timerNo : The timer No. used by the time-out routine
void (**IntrFunc)(void) : Pointer to the corresponding timer interrup address in the interrupt vector table
<Return Value> u16 result :Normal End → 0
Parameter error (timer No>3) → 1

This function selects the timer interrupt used to determine time-outs when accessing the flash memory.

This function needs to be called at least one time before EraseFlashChip, EraseFlashSector or ProgramFlashSector is called. Once the timer interrupt used with the routine has been set, there is no need to call this routine again, unless the timer interrupt is used with other processes or another interrupt vector table is used.

* As per the library's specifications, when this routine is called a vector of specific timer interrupt routine is forcibly set as the vector of the library function. For this reason, the interrupt table must be in the RAM region at the time that flash is accessed.

`void` **`ReadFlash`**`(u16 secNo,u32 offset,u8 *dst,u32 size)`

<Arguments>	u16 secNo	: Target sector No.
	u32 offset	: Offset in bytes in sector
	u8 *dst	: WRAM address to store read data
	u32 size	: Read size in bytes
<Return Value>	None	

This function reads "size" bytes of data from the address offset by "offset" bytes in the specified flash memory sector No. and loads it to the work RAM starting from the "dst" address.

The function operates normally, even when the specified read size straddles a sector boundary.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

```
u32 VerifyFlashSector(u16 secNo,u8 *src)
<Arguments>  u16 secNo : Target sector No
                  u8 *src : The address where the verification originates  Address in AGB
                            memory map)
<Return Value> u16 result : Normal End → 0
                            Verify Error → Error address on device side
```

Verifies 1 sector (4 kbytes) of data from the src address and the target sector No. from the flash.

If this function has completed the verification normally, 0 is returned. If a verification error has occurred, it returns the address where the error occurred.

This routine does not perform a parameter check.

```
u16 (*EraseFlashChip)()      * Must call SetFlashTimerIntr prior to this function
<Arguments>  None
<Return Value>  u16 result (*1) : Normal End → 0
                                  Chip erase timeout error → 0xc003
```

Erases the entire chip, completely.

```
u16 (*EraseFlashSector)(u16 secNo)  * Must call SetFlashTimerIntr prior to this function
<Arguments>  u16 secNo  : Target sector No.
<Return Value>  u16 result (*1)  : Normal End → 0
                                   Parameter error (secNo>0x0f) → 0x80ff
                                   Sector erase timeout error → 0xc002
```

Erases the target sector No. sector.

This routine is called in the write function ProgramflashSector, so usually there is no need to call it prior to the write. A parameter error is returned when the target sector No. is outside the range.

*With Atmel's flash, all interrupts are often prohibited in the function. For details, see "Cautions when Using Flash Access Functions".

When calling this function, please halt direct sound and DMAs that start in synch with V & H blanks.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

`u16 (*ProgramFlashSector)(u16 secNo,u8 *src)` * Must call SetFlashTimerIntr prior to this function

<Arguments> u16 secN : Target sector No.
u8 *src : Source address
(Address in AGB memory map)
<Return Value> u16 result (*1) : Normal End → 0
Parameter error (secNo>0x0f) → 0x80ff
Sector erase verify error → 0x8004
(ONLY WITH SANYO'S FLASH)
Sector erase timeout error → 0xc002
Program timeout error → 0xc001

Writes 1 sector (4 kbytes) of data from the src address to the target sector No.

In this function, call EraseFlashSector, mentioned previously, to erase the sector. Then write data to the sector.

A parameter error is returned when the target sector No. is outside the range.

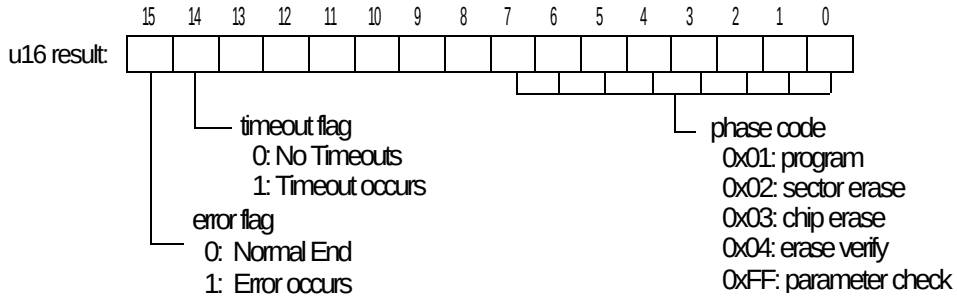
By referencing the global variable, flash_remainder, while executing this routine, you can figure out the remaining number of bytes.

*With Atmel's flash, all interrupts are prohibited in the function. For details, see "Cautions when Using Flash Access Functions"

When calling this function, please halt direct sound and DMAs that start in synch with V & H blanks.

*** 1 Error Code Details**

When an error occurs, the error codes as structured below are returned.



D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

4.3 4 Kbit EEPROM

EEPROM uses the CS, A23, D0 of the Game Pak bus and is a serial connection. It is expressed in the upper half of each wait state region of the Game Pak ROM (0x09000000, 0x0b000000, 0x0d000000~) in the AGB-CPU memory map.

Accordingly, the maximum ROM size is 128 Mbit when EEPROM is used as the backup device.

The following features apply when the library functions are used to access EEPROM:

* The library functions access EEPROM using the ROM wait state 2 region (0x0c000000 ~).
* Library functions are used to access EEPROM for both reading and writing, and the unit of access is double word (8 bytes).
* Because EEPROM is connected in the way described above, the EEPROM target address for these library functions is specified using EEPROM addresses (4kbit range = 0x0000 ~ 0x003F) rather than the usual CPU memory map addresses.
* Because the library functions use DMA3 internally, you must keep DMA3 free while using these functions.
* Because DMA is used for access, addresses with a 16-bit boundary must be specified for the transfer target buffer and transfer origin address arguments of these library functions.
* All interrupts are forcedly prohibited through the control of the IME register while DMA is being used during execution of the library functions ReadEepromDword, VerifyEepromDword and ProgramEepromDword.
* The library function ProgramEepromDword uses any one of the timers 0~3 for time out processing.
* The wait cycle is adjusted inside each access function, so the developer does not need to worry about this.
* There is no guarantee that data has been written correctly even when ProgramEepromDword ends normally. Thus, in order to be certain that data has been correctly written please use VerifyEepromDword after writing the data.

****Cautions when Using EEPROM Access Functions (Important)****

The access functions (ProgramEepromDword, ReadEepromDword, and VerifyEepromDword) for this device, use DMA3 to access. However, when the DMA is executing, this DMA can be interrupted by one with higher priority and the access will fail. Therefore, when DMA is executing, interrupts are prohibited in the library functions. However, this will not prevent DMA 1 and 2 occurring with direct sound. Therefore, do not use direct sound when calling this function. Also, do not use DMAs that start in synch with V & H blanks.

To disable interrupts, the IME register is set to 0. Therefore, interrupts allowed before calling this libraries' functions may occur with a maximum delay of approximately 40us when the function is being executed. For interrupts that may cause abnormal operation because of the delay, be sure and set the corresponding interrupt request enable register (this is not the IE register) to "Prohibited" before calling the library functions.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

**** Cautions on the Life of EEPROM Rewrite (Important)****

Usually, EEPROM is limited by the number of rewrites so you need to be careful how frequently data is saved. For example, do not include routines that save frequently with the parameter input screen, or frequently write to memory during communication. Obviously, you cannot use this with games that do auto-saves which frequently do rewrites. If you do not follow these guidelines, you may significantly shorten the life of the product.

<Remarks>

The AGB uses 4Kbit EEPROM products that are guaranteed by their manufacturers to have a minimum of 100,000 write cycles per address. This is equal to a life of around 300 saves per day, and a life of 1 year.

Library Functions

```
u16 SetEepromTimerIntr(u8 timerNo, void (**IntrFunc)(void))
```

<Arguments> u8 timerNo : The timer No. used by the time-out routine
void (**IntrFunc)(void) : Pointer to the corresponding timer interrupt address in the timer interrupt table
<Return Value> u16 result : Normal End → 0
Parameter error (timerNo>3) → 1

This function sets the timer interrupt used to determine time-outs when accessing the EEPROM.

This function needs to be called at least one time before the write function ProgramEepromDword is called. Once the timer interrupt used with the routine has been set, there is no need to call this routine again, unless the timer interrupt is used with other processes or another interrupt vector table is used.

NOTE: As per the library's specifications, when this routine is called a vector of specific timer interrupt routine is forcibly set as the vector of the library function. For this reason, the interrupt table must be in the RAM region at the time that the EEPROM is accessed.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

```
u16 ReadEepromDword(u16 epAdr, u16 *dst)
```

<Arguments> u16 epAdr : Target EEPROM address (0x00~0x3f)
u16 *dst : Transfer destination address of the read data
(An address with a 16 bit boundary in the AGB memory map)
<Return Value> u16 result (*2) : Normal End → 0
Parameter error (epAdr>0x3f) → 0x80ff

This function reads 8 bytes of data from the specified EEPROM address and loads it in after address "dst."

A parameter error is returned when the target EEPROM address is outside the range.

This function uses DMA for access, so data transfers are performed in units of 16 bits. Accordingly, the transfer destination address u16 *dst must specify an address with a 16-bit boundary, as shown below:

```
    :

  u16 buff[4];
  ReadEepromDword(0,buff);

    :
```

NOTE: All interrupts are prohibited for a certain period of time (about 40 μs) while this function is being called. For details, see “Cautions when Using EEPROM Access Functions.”

Also, please halt direct sound and DMAs that start in synch with V & H blanks when calling this function.

```
u16 ProgramEepromDword(u16 epAdr, u16 *src)
```

Must call SetFlashTimerIntr prior to this function

<Arguments> u16 epAdr : The EEPROM address to be written to. (0x00~0x3f)
u16 *src : The origin address of the data (An address with a 16 bit boundary in the AGB memory map)
<Return Value> u16 result (*2) : Normal End → 0
Parameter error (epAdr>0x3f) → 0x80ff
Program timeout error → 0xc001

This function takes 8 bytes of data from the address "src" and writes it to the target address in EEPROM.

A parameter error is returned when the target EEPROM address is outside the range.

This function uses DMA for access, so data transfers are performed in units of 16 bits. Accordingly, the transfer origin address u16 *src must specify an address with a 16-bit boundary, just like for ReadEepromDword.

*When calling this function all interrupts are prohibited (approx. 40μs). For details see “Cautions when Using EEPROM Access Functions”.

Also, do not use direct sound and DMAs that start in sync with V & H blanks when calling this function.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

```
u16 VerifyEepromDword(u16 epAdr, u16 *src)
```

<Arguments> u16 epAdr : Target EEPROM address (0x00~0x3f)
u16 *src : Address where verification originates (An address with a 16 bit boundary in the AGB memory map)

<Return Value> u16 result (*2) : Normal End → 0
Parameter error (epAdr>0x3f) → 0x80ff
Verify error → 0x8000

This function verifies 8 bytes of data at the specified EEPROM address with the data from address "src."

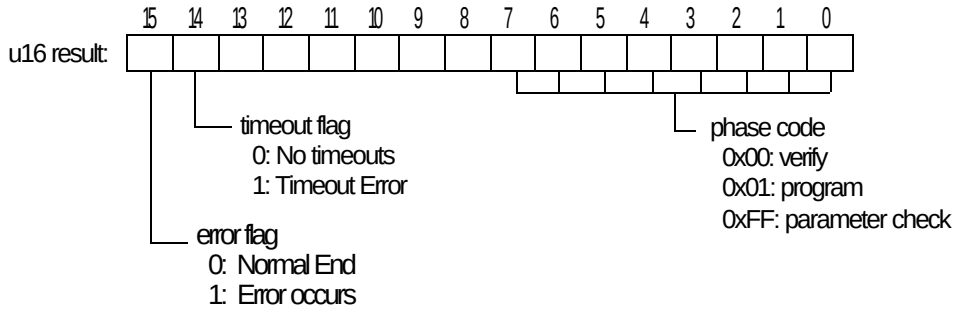
The function uses DMA for access, so data transfers are performed in units of 16 bits. Accordingly, the verify origin address u16 *src must specify an address with a 16-bit boundary, just like for ReadEepromDword.

*When calling this function all interrupts are prohibited (approx. 40µs). For details see "Cautions when Using EEPROM Access Functions".

Also, do not use direct sound and DMAs that start in sync with V & H blanks when calling this function.

***2 Details on Error Code**

When an error occurs, the error codes as structured below are returned.

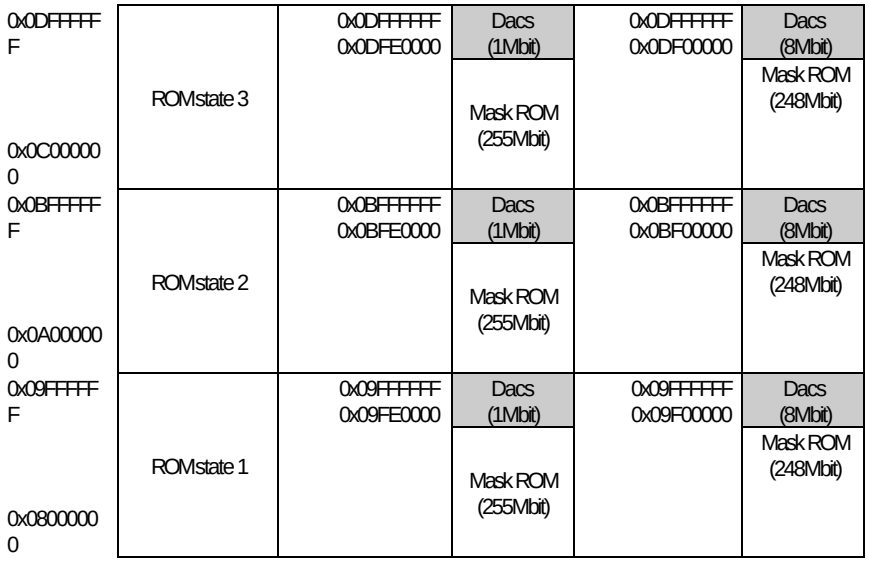


4.4 1Mbit, 8Mbit DACS

****** There are no plans at the present time to market these devices ******

1Mbit and 8Mbit DACS are allocated to the game pak ROM wait state regions in the CPU memory map in the manner shown below. The library functions use the ROM state 3 area to access these devices. Accordingly, when DACS is being used the maximum usable ROM size is (256Mbit DACS size).

When 1Mbit DACS is used, 0x80 must be specified in the ROM registration area for the [Device type (0x080000B4)]. When 8Mbit DACS is used, 0x00 must be specified.



The library functions manage the various DACS sectors in the format shown below.
A 16Kbyte region (0x0DFE0000 ~ 0x0DFE3FFF in the1Mbit DACS and 0x0DFFC000 ~ 0x0DFFFFFF in the 8Mbit DACS) is used for security and for ROM patch functions. Since the areas cannot be used as a backup region, the library functions do not allow them to be specified.

1Mbit DACS

Address	Sector No. (Size)
E000	Sector 0x0D (8Kbyte)
C000	Sector 0x0C (8Kbyte)
A000	Sector 0x0B (8Kbyte)
8000	Sector 0x0A (8Kbyte)
6000	Sector 0x09 (8Kbyte)
4000	Sector 0x08 (8Kbyte)
2000	Sector 0x07 (8Kbyte)
0x0DFF0000	Sector 0x06 (8Kbyte)
E000	Sector 0x05 (8Kbyte)
C000	Sector 0x04 (8Kbyte)
A000	Sector 0x03 (8Kbyte)
8000	Sector 0x02 (8Kbyte)
6000	Sector 0x01 (8Kbyte)
4000	Sector 0x00 (8Kbyte)
0x0DFE0000	Disabled region

8Mbit DACS

Address	Sector No. (Size)
0x0DFFC000	Disabled region
A000	Sector 0x14 (8Kbyte)
8000	Sector 0x13 (8Kbyte)
6000	Sector 0x12 (8Kbyte)
4000	Sector 0x11 (8Kbyte)
2000	Sector 0x10 (8Kbyte)
0x0DFF0000	Sector 0x0F (32Kbyte)
E0000	Sector 0x0E (32Kbyte)
D0000	Sector 0x0D (32Kbyte)
C0000	Sector 0x0C (32Kbyte)
B0000	Sector 0x0B (32Kbyte)
A0000	Sector 0x0A (32Kbyte)
90000	Sector 0x09 (32Kbyte)
80000	Sector 0x08 (32Kbyte)
70000	Sector 0x07 (32Kbyte)
60000	Sector 0x06 (32Kbyte)
50000	Sector 0x05 (32Kbyte)
40000	Sector 0x04 (32Kbyte)
30000	Sector 0x03 (32Kbyte)
20000	Sector 0x02 (32Kbyte)
10000	Sector 0x01 (32Kbyte)
0x0DF00000	Sector 0x00 (32Kbyte)

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

The following features apply when the library functions are used to access DACS

- Library functions are used to access DACS for writing, but for reading it is possible to read data directly from the addresses allocated to DACS. However, it is also possible to read with the ReadDacs function, which can specify sector addresses.
- The minimum unit of access is 2 bytes when reading, and either 1 sector (the size of which depends on the device and the sector number) when writing with ProgramDacsSector or 2 bytes when writing with ProgramDacs_NE. Take care when using ProgramDacs_NE. Note that this function does not perform an erase process, so if data is written to an area in which data has already been written, the value that is written is the AND of the existing data and the newly written data. Thus, for example, if 0x3333 is written to an area where 0x5555 is already written, the result will be 0x1111.
- Due to the characteristics of the library functions, the transfer origin and transfer destination addresses should be specified with a 16-bit boundary. If an odd-numbered address with an 8-bit boundary is specified the functions will not operate properly.
- The library functions EraseDacsChip, EraseDacsSector, ProgramDacs_NE and ProgramDacsSector use any one of the timers 0~3 for time out processing.
- The wait cycle is adjusted inside each access function, so the developer does not need to worry about this.
- The DACS device has a ROM patch function, and out of consideration for times when this is used, the library functions EraseDacsChip, EraseDacsSector, ProgramDacs_NE, and ProgramDacsSector mask the IE register during a part of the time while they are executing. This is done to forcedly prohibit all interrupts, with the exception of the timer interrupts that are used inside each function for time out processing.
- There is no guarantee that data has been written correctly to DACS even when ProgramDacsSector or ProgramDacs_NE ends normally. Thus, in order to be certain that data has been correctly written, please use VerifyDacsSector or VerifyDacs after writing the data.

**** Cautions About Using the DACS Access Functions (Important) ****

As mentioned above, the library functions that access the DACS device (EraseDacsChip, EraseDacsSector, ProgramDacs_NE and ProgramDacsSector) forcedly prohibit all interrupts except timer interrupts during a portion of the time that they are executing. This is done by setting all bits in the IE register to 0 except for the bit that pertains to time out processing. Because of this, an interrupt that was enabled before one of these library functions was called could be delayed by several dozen microseconds to several seconds, in which interrupts are prohibited during execution of the library function. Before calling these library functions be sure to set the interrupt request enable flag in each control register (this is not the IE register) to "Disabled" for any interrupt that might cause faulty operation if delayed.

In addition, during the execution of these functions do not use direct sound and DMAs that start in sync with V blanks and H blanks.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

**** Cautions about the write cycle lifetime of DACS (Important) ****

DACS has a limited number of write cycles, so you need to be careful about the ways you save data to these devices.

For example, do not include routines that frequently save data at places like the parameter input screen, and do not write data frequently during communications.

It is obvious that DACS cannot be used with games that have an auto-save function and frequently rewrite data.

If these cautions are not heeded, the lifetime of the product could be significantly shortened, so please be careful.

< Remarks >

The AGB uses DACS products that are guaranteed by their manufacturers to have a minimum of 100,000 write cycles per sector. That works out to a lifetime of 1 year if data is saved at a rate of around 300 times per day.

Library Functions

u16 **IdentifyDacs**()

<Arguments> None

<Return value> u16 result : Normal end → 0

Identity error (Not a device that is in the library) → 1

This function identifies which DACS device is installed in the game pak, gets the flash capacity and sector size, and sets the access speed. The obtained data can be referenced using the global variable dacsType *dacs. (For details about dacsType read the header file dacs.h)

Be sure to call this function once prior to accessing the DACS (including prior to reading the device).
If the device cannot be identified, an error is returned and the subsequent access functions cannot be used.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

u16 **SetDacsTimerIntr(u8 timerNo, void (**IntrFunc)(void))**

<Arguments> u8 timerNo : The timer No. used by the time-out routine.
void (**IntrFunc)(void) : Pointer to the corresponding time interrupt address in the timer interrupt table.

<Return value> u16 result : Normal end → 0
Parameter error (timerNo > 3) → 1

This selects the timer interrupt that will be used to determine time-outs when the DACS device is being accessed, **and transfers the function for determining time-outs and the function for polling the DACS status to the static buffer secured in WRAM.**

This function needs to be called at least one time before the library functions EraseDacsChip, EraseDacsSector, ProgramDacs_NE or ProgramDacsSector are called. Once the timer interrupt used with this routine has been set, there is no need to call this function again, unless the timer interrupt is used with other processes or another interrupt vector table is used. Be careful when you use the library's WRAM area which is overlaid, there is a chance that the function transferred to WRAM will be overwritten.

As per the library's specifications, when this routine is called, a vector of specific timer interrupt routine is forcibly set as the vector of the library function. For this reason, the interrupt table must be in the RAM region at the time that the DACS device is accessed.

u32 **ExchangeSectorToPhysAdr**(u16 secNo)

<Arguments> u16 secNo : Target sector No.
<Return value> u32 physAdr : Normal end → Starting address of the target sector No. in the AGB-CPU memory map
Parameter error → 0

Converts the DACS sector No. into an address in the AGB-CPU memory map.
A parameter error occurs and 0 is returned if the sector No. is unusual.

u32 **ExchangePhysAdrToSector**(u32 physAdr)

<Arguments> u32 physAdr : DACS address in the AGB-CPU memory map
<Return value> u32 secNo_offset : Normal end → (sector No. << 24) | (Offset in sector)
Parameter error → 0

Converts the DACS address in the AGB-CPU memory map into a DACS sector No. + offset inside sector.

In the return value, bit 31~24 is the sector No. and bit 23~0 is the offset inside the sector.

A parameter error occurs and 0 is returned if an address that is outside the range allocated for DACs is specified.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

void **ReadDacs**(u16 secNo, u32 byteOffset,u16 *dst, u32 byteSize)

<Arguments> u16 secNo : Target sector No.
u32 byteOffset: : Offset inside the sector in units of bytes (Specified as an even number of bytes)
u16 *src : Transfer destination address of the read data (An address with a 16 bit boundary in the AGB memory map)
u32 byteSize : Read byte size (Specified as an even number of bytes)

<Return value> None

This function reads "byteSize" bytes of data starting at the address "byteOffset" bytes away from the start of the target sector No., and loads the data into the region starting from the "dst" address.

The function operates normally even when the specified read size straddles a sector boundary.

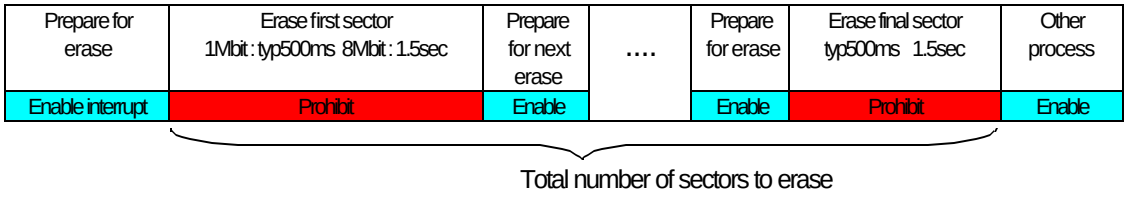
Specify an even number of bytes for byteOffset and byteSize, and specify an address with a 16 bit boundary for *dst.

u16 **EraseDacsChip**()

<Arguments> None
<Return value> u16 result (*3) : Normal end → 0
Lock reset error → 0x8004 (0xc004 when time-out)
Lock set error → 0x8003 (0xc003 when time-out)
Sector erase error → 0x8002 (0xc002 when time-out)

Erases the entire chip.
The error code "sector erase error" is returned when the device does not have a chip erase command so all sectors will be erased by the software.

* When calling this function, please halt direct sound and DMAs that start in synch with V & H blanks.
* All interrupts with the exception of the timer used for time-out processing are prohibited for part of the time inside this function. Be aware that any interrupt that was enabled prior to the call of this function will occur immediately after the end of this prohibition period (which lasts several hundred milliseconds to several seconds).



D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

u16 **EraseDacsSector**(u16 secNo)

<Arguments> u16 secNo : Target sector No.
<Return value> u16 result (*3) : The same codes that are returned by EraseDacsChip, plus:
Parameter error → 0x80ff (1Mbit: secNo > 0x0D, 8Mbit: secNo > 0x14)

Erases the sector specified by the target sector No.

This routine is called from inside the write function ProgramDacsSector, so when you use this function you do not need to call this routine before writing.

A parameter error occurs when the target sector No. is outside the range of the relevant device.

* When calling this function, please halt direct sound and DMAs that start in synch with V & H blanks.
* All interrupts with the exception of the timer used for time-out processing are prohibited for part of the time inside this function. Be aware that any interrupt that was enabled prior to the call of this function will occur immediately after the end of this prohibition period (which lasts several hundred milliseconds to several seconds).

Prepare for erase	Target sector erase process 1Mbit : typ500ms 8Mbit : typ1.5sec)	Other process
Enable interrupt	Prohibit	Enable

u16 **ProgramDacs_NE**(u16 secNo, u32 byteOffset, u16 *src, u32 byteSize)

<Arguments> u16 secNo : Target sector No.
u32 byteOffset : Offset inside the sector in units of bytes (Specified as an even number of bytes)
u16 *src : Write origin address (An address with a 16 bit boundary in the AGB memory map)
u32 byteSize : Write byte size (Specified as an even number of bytes)
<Return value> u16 result (*3) : Normal end → 0
Parameter error → 0x80ff (1Mbit: secNo > 0x0D, 8Mbit: secNo > 0x14, specified offset, size exceeds device's region)
Lock reset error → 0x8004 (0xc004 when time-out)
Lock set error → 0x8003 (0xc003 when time-out)
Sector erase error → 0x8002 (0xc002 when time-out
Program error → 0x8001 (0xc001 when time-out)

This function takes "byteSize" bytes of data from the address "src" and writes it to the address "byteOffset" bytes away from the start of the target sector No.
Note that the EraseDacsSector routine is not called from inside this function, so if data is already written to the destination area, then the value that is written to this address is the AND of the existing data and the new data. Thus, for example, if 0x3333 is written to an address where 0x5555 is already written, the result will be 0x1111.

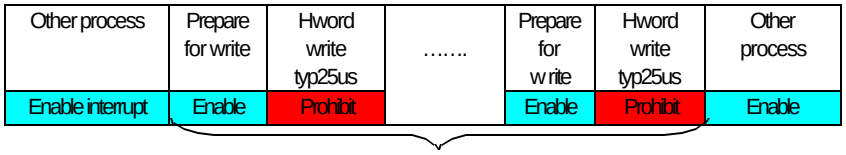
The function operates normally even when the specified write size straddles a sector boundary. However, a parameter error occurs when the specified size exceeds the region allocated to the device.

D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

A parameter error occurs when the target sector No. is outside the range of the relevant device.

Specify an even number of bytes for byteOffset and byteSize, and specify an address with a 16 bit boundary for *src.

* When calling this function, please halt direct sound and DMAs that start in synch with V & H blanks.
* All interrupts with the exception of the timer used for time-out processing are prohibited for part of the time inside this function. Be aware that any interrupt that was enabled prior to the call of this function will occur immediately after the end of this prohibition period (which lasts several dozen microseconds).



u16 **ProgramDacsSector**(u16 secNo,u16 *src)

<Arguments> u16 secNo : Target sector No.
u16 *src : Write origin address (An address with a 16 bit boundary in the AGB memory map)
<Return value> u16 result(*3) : Parameter error → 0x80ff (1Mbit: secNo > 0x0D, 8Mbit: secNo > 0x14)
All other codes same as ProgramDacs_NE

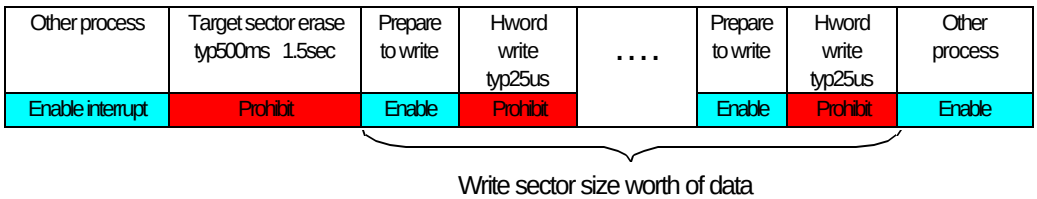
Takes the specified sector size of data from the "src" address and writes it to the target sector No.

All sectors in the 1Mbit DACS are 8kbyte, but in the 8Mbit DACS the sectors 0x00 ~ 0x0E are 32kbyte and the sectors 0x0F ~ 0x14 are 8kbyte. So those sizes determine the amount of data written.

This function makes an internal call to EraseDacsSector, so the sector is erased before data is written.

A parameter error occurs when the target sector No. is outside the range of the relevant device. Specify an address with a 16 bit boundary for *src.

* When calling this function, please halt direct sound and DMAs that start in synch with V & H blanks.
* All interrupts with the exception of the timer used for time-out processing are prohibited for part of the time inside this function. Be aware that any interrupt that was enabled prior to the call of this function will occur immediately after the end of this prohibition period (which lasts several dozen microseconds to several seconds).



D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

u32 **VerifyDacs**(u16 secNo, u16 byteOffset, u16 *src, u32 byteSize)

<Arguments> u16 secNo : target sector No.
u32 byteOffset: : Offset inside the sector in units of bytes (Specified as an even number of bytes)
u16 *src : Verify origin address (An address with a 16 bit boundary in the AGB memory map)
u32 byteSize : Verify byte size (Specified as an even number of bytes)

<Return value> u16 result (*3) : Normal end → 0
Verify error → Error address on device side

This function verifies "byteSize" worth of data at the address "src" and at the address "byteOffset" bytes away from the start of the target sector No.

The function operates normally even when the specified verify size straddles a sector boundary. Specify an even number of bytes for byteOffset and byteSize, and specify an address with a 16 bit boundary for *src.

This routine does not perform a parameter check.

u32 **VerifyDacsSector**(u16 secNo, u16 *src)

<Arguments> u16 secNo : Target sector No.
u16 *src : Verify origin address

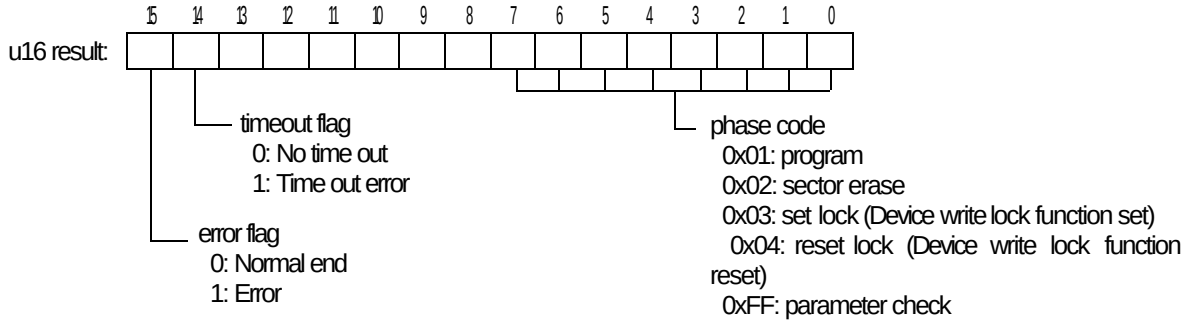
<Return value> u16 result (*3) : Normal end → 0
Verify error → Error address on device side

This function verifies the specified sector's worth of data at the address "src" and in the target sector No.
All sectors in the 1Mbit DACS are 8kbyte, but in the 8Mbit DACS the sectors 0x00 ~ 0x0E are 32kbyte and the sectors 0x0F ~ 0x14 are 8kbyte, so those sizes determine the amount of data verified.

Specify an address with a 16 bit boundary for *src.

This routine does not perform a parameter check.

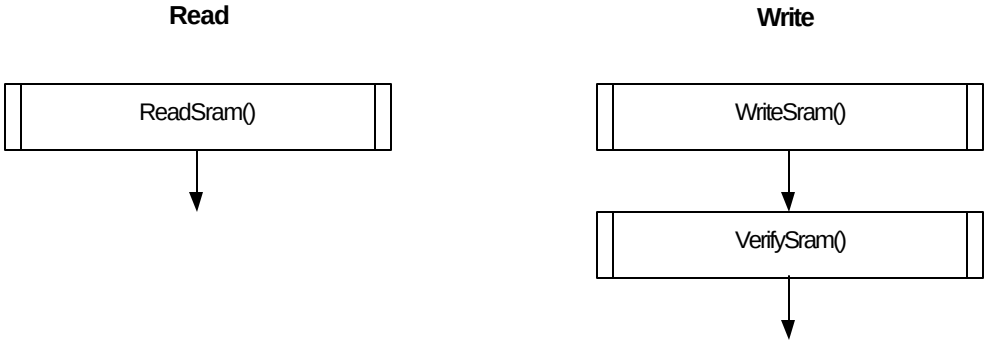
*3 Detailed explanation of the error codes



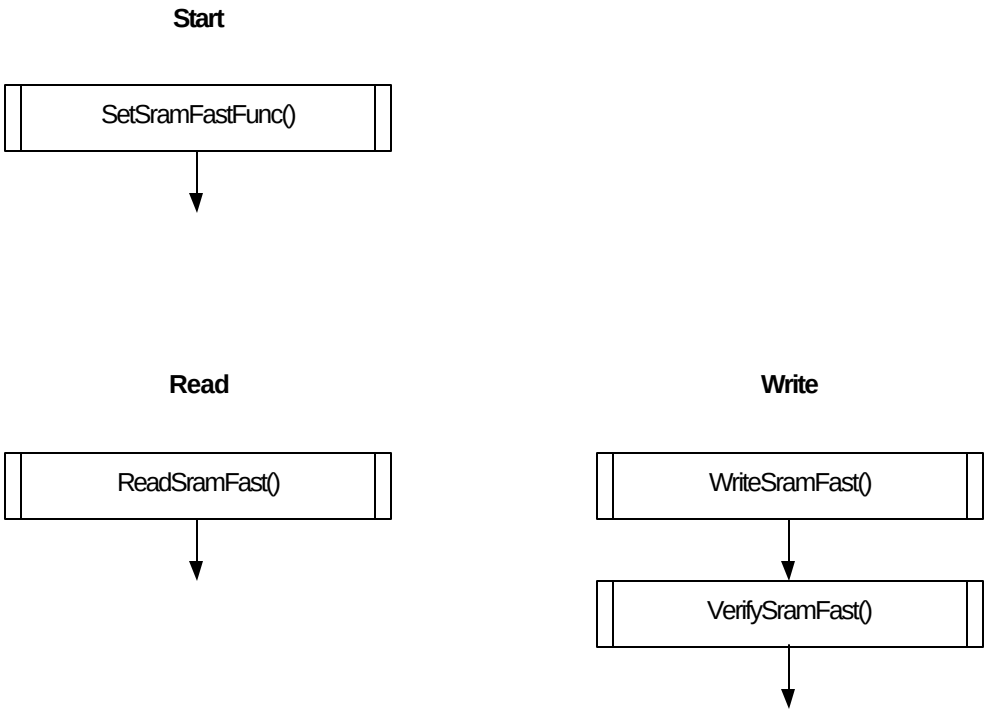
5 Flowchart of Access to Each Device

The following flowcharts broadly show what happens when each device is accessed.

5.1 256kbit SRAM



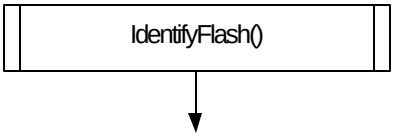
5.2 256kbit SRAM SramFast function



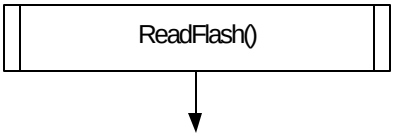
D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

5.3 512kbit FlashROM

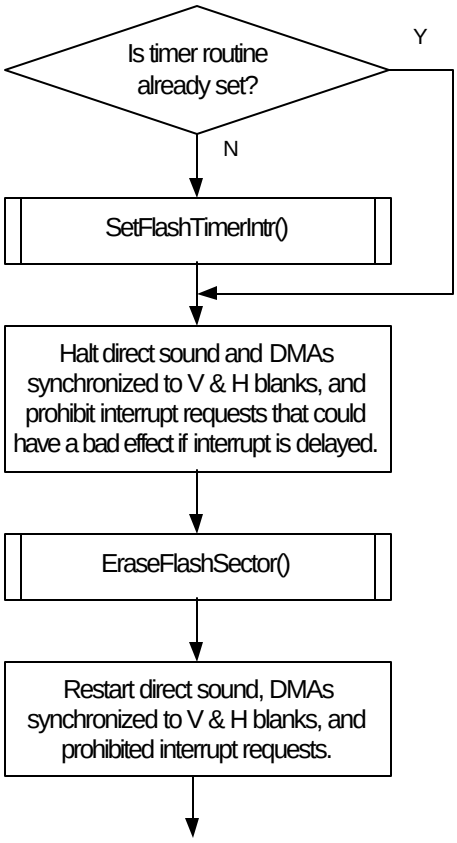
Start



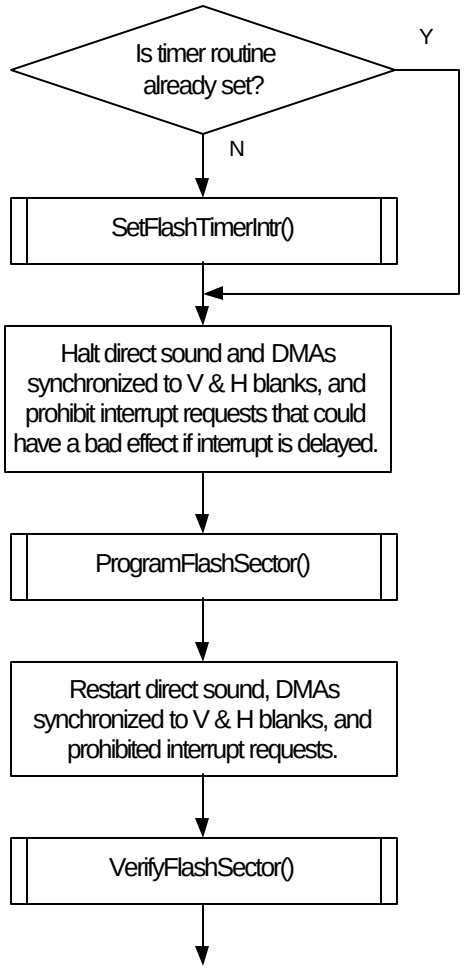
Read



Erase



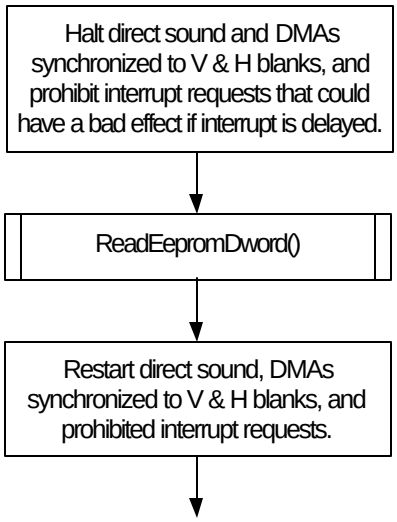
Write



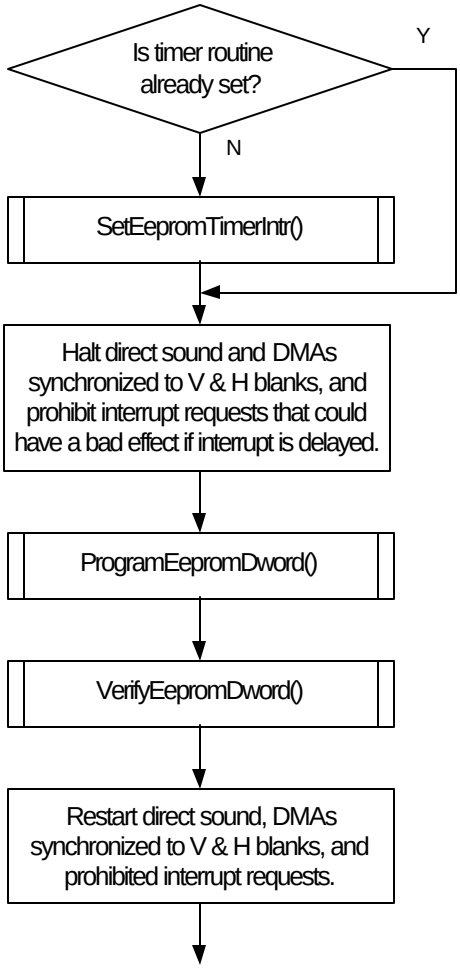
D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

5.4 4Kbit EEPOM

Read



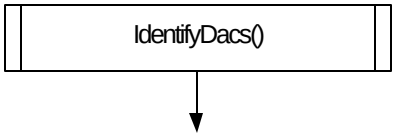
Write



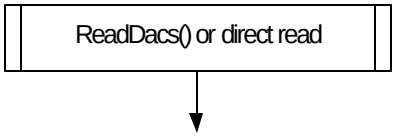
D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

5.5 1M,8Mbit DACS

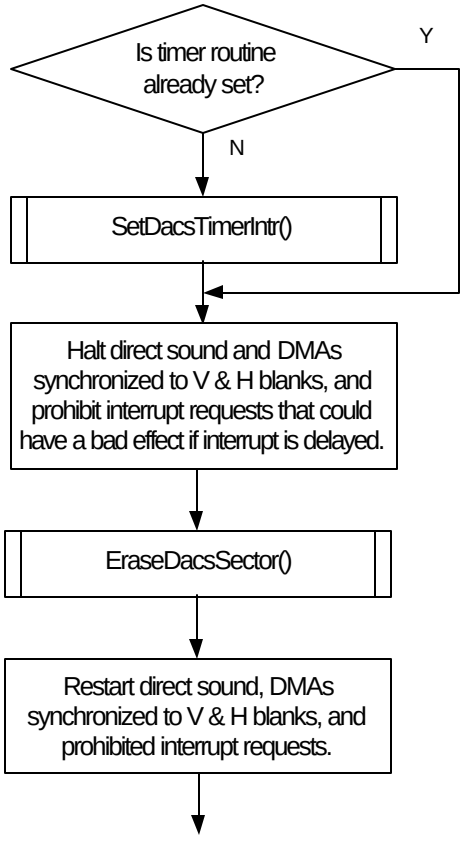
Start



Read

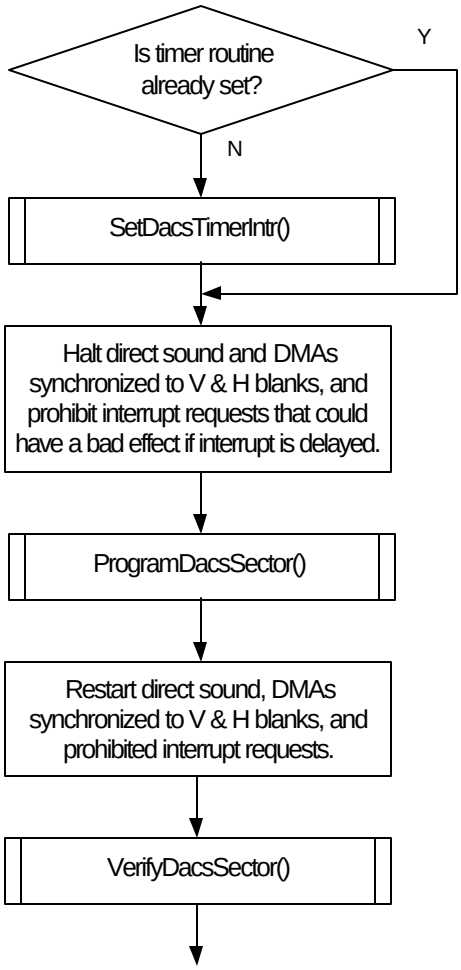


Erase

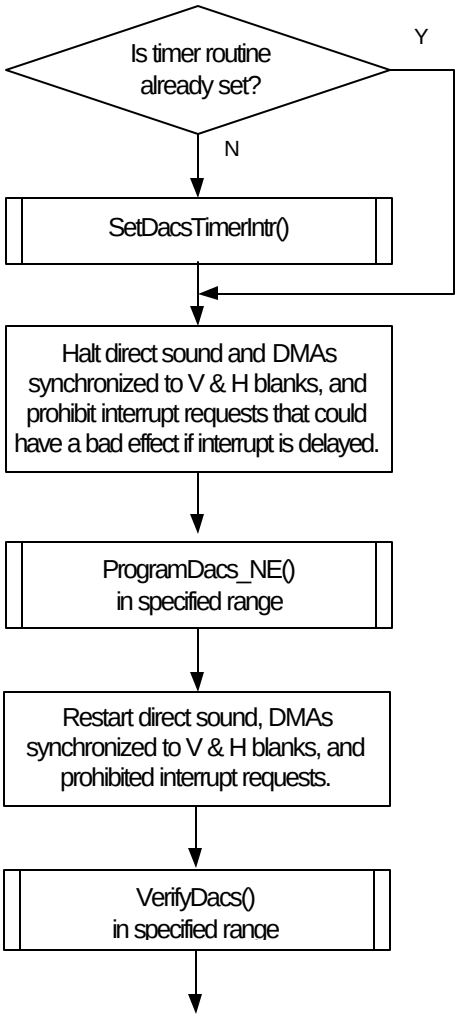


D.C.N. AGB-06-0011-001C1
Released: 3/23/2001

Write (Sector rewrite)



Write (Overwrite without erasing)



D.C.N. AGB-06-0011-001C1
Released: 3/23/2001