DS 用 AGB カートリッジ
バックアップアクセス解説

Ver 1.0.3

任天堂株式会社発行

目次

1 はじめに 4

2 各デバイスへのアクセス方法 5

3 AGBバックアップアクセス関数を使用する際の注意点 5

4 AGBバックアップアクセス関数の解説 6

4.1 AGBバックアップ全般 6

4.1.1 関数リファレンス（AGBバックアップ全般） 6

4.2 256Kbit SRAM 7

4.2.1 関数リファレンス（SRAMデバイス） 8

4.3 512Kbit, 1Mbit FLASH 10

4.3.1 関数リファレンス（FLASHデバイス） 12

5 各デバイスへのアクセス時のフローチャート 17

5.1 全デバイス共通 17

5.2 256Kbit SRAM 18

5.3 512Kbit, 1Mbit FLASH 19

改訂履歴

版	改訂日	改 訂 内 容	担当者
1.0.3	2006-09-26	2 記述修正	奥畑
1.0.2	2006-06-06	1 記述修正 4 用語表記訂正 (CTRDG_TASK_FUNC)	奥畑
1.0.1	2006-04-07	NitroSDK 収録による変更	奥畑
1.0.0	2005-12-27	初版発行	白井

1 はじめに

NITRO-SDK では、ゲームボーイアドバンス(以下 AGB とする)のカートリッジ上に搭載されたバックアップデバイスへのアクセスを行うための一連の API が用意されています。
このドキュメントでは AGB カートリッジバックアップへの基本的なアクセス方法について解説します。

バックアップメモリへのアクセスは必ず弊社提供の NITRO-SDK を介して行って下さい。ユーザが独自に記述したプログラムによる直接リード/ライトは禁止です。

バックアップデバイスは現在、以下の 3 点が使用可能となっており、これらのデバイスに対応した API が用意されています。

- **256KbitSRAM**
- **512KbitFLASH**
- **1MbitFLASH**

（4K,64KbitEEPROMはDSアプリケーションからアクセスすることはできません。）

2 各デバイスへのアクセス方法

各デバイスへのアクセス方法は以下の表のようになっており、記載されている方法以外でのアクセスは禁止します。

デバイス	リード方法	最小リード単位	ライト方法	最小ライト単位
SRAM	CTRDG_ReadAgbSram CTRDG_ReadAgbSramAsync	1byte	CTRDG_WriteAgbSram CTRDG_WriteAgbSramAsync	1byte
FLASH	CTRDG_ReadAgbFlash CTRDG_ReadAgbFlashAsync	1byte	CTRDG_WriteAgbFlashSector CTRDG_WriteAgbFlashSectorAsync	4kbyte

各デバイスにアクセスするには、AGB カートリッジ側のバックアップデバイスの種別と容量を前もって特定しておく必要があります。特定が完了するまでは、バックアップメモリに関するいかなる関数も使用してはいけません。

(CTRDG_IdentifyAgbBackup 関数も例外ではありません。)

バックアップメモリの種別と容量を特定するには CTRDG_GetAgbMakerCode 関数でメーカーコードを取得し、自社製品であることを確認した後 CTRDG_GetAgbGameCode 関数でイニシャルコードを調べバックアップメモリの種別と容量を特定してください。

※ ただし IS-NITRO-DEBUGGER 1.66 以前のバージョンでは AGB バックアップデバイスに正常にアクセスすることはできませんので注意してください。

3 AGB バックアップアクセス関数を使用する際の注意点

本マニュアル以外に、DS プログラミングガイドラインにもバックアップメモリに関する重要な情報や注意点、規定などが記されています。必ず DS プログラミングガイドラインも参照し、記載されている規定を守った上で使用するようにしてください。

4 AGB バックアップアクセス関数の解説

以下に各デバイスのアクセス関数の解説を行います。

4.1 AGB バックアップ全般

AGB バックアップデバイスは SRAM、FLASH 共に、メモリマップ上のカートリッジ RAM 領域(0x0A000000～)に割り当てられています。

本アクセス関数による AGB バックアップデバイスへのアクセスは以下のような特徴を持ちます。

- ウェイトサイクルは各アクセス関数内で調整されるため、開発者が意識する必要はありません。
- アクセス関数には同期関数と非同期関数(*Async)があり、非同期関数は CTRDG_Init 関数内で作成されたスレッドで同期関数を実行することによって非同期を実現しています。

AGB バックアップアクセス関数使用時の注意点(重要)

(1) アクセス使用前にアプリケーションで必要な処理

アプリケーション側で AGB カートリッジ側に適切なデバイスが搭載されていることを特定するまでは、いかなる AGB バックアップアクセス関数も使用してはいけません。

(CTRDG_IdentifyAgbBackup 関数も例外ではありません。)

4.1.1 関数リファレンス(AGB バックアップ全般)

u16 **CTRDG_IdentifyAgbBackup**(CTRDGBackupType type)

＜引数＞	CTRDGBackupType type	NITRO-CTRDG に搭載されているバックアップデバイスの種類	
＜戻り値＞	u16 result	正常終了	⇒ 0
		識別エラー(ライブラリ内に該当デバイスが無い場合)	⇒ 0 以外

NITRO-CTRDG に搭載されているバックアップデバイスを指定します。

バックアップデバイスが FLASH の場合は FLASH の ID を読み出し、どの FLASH がカートリッジに搭載されているのかを識別して、FLASH の容量やセクタサイズの取得、アクセススピードの設定、さらに対応する FLASH 用の各アクセス関数のセットを行います。取得した FLASH のデータはグローバル変数 flashType *flash で参照することができます。(flashType の詳細はヘッダファイル ctrdg_flash.h を参照してください。)

本関数はバックアップデバイスにデータの書き込み及び読み込みを行う前に 1 回コールする必要があります。

256KbitSRAM 使用時には引数に CTRDG_BACKUP_TYPE_SRAM を、512MbitFLASH 使用時には

CTRDG_BACKUP_TYPE_FLASH_512K、1MbitFLASH 使用時は CTRDG_BACKUP_TYPE_FLASH_1M を与えてください。

デバイスを識別できなかった場合はエラーを返し、AGB バックアップアクセス関数は使用不可となります。

※ また引数に CTRDG_BACKUP_TYPE_FLASH_512K または、CTRDG_BACKUP_TYPE_FLASH_1M を与えた場合、デバイスへの書き込み動作が発生しますので、与える引数と異なる種類のデバイスが NITRO-CTRDG に搭載されているとバックアップデータが破壊されることがありますので注意してください。

void **CTRDG_SetTaskThreadPriority**(u32 priority)

＜引数＞	u32 priority	タスクスレッドの優先度
＜戻り値＞	なし	

非同期関数を実行するタスクスレッドの優先度を変更します。

4.2 256Kbit SRAM

SRAM は、メモリマップ上のカートリッジ RAM 領域（0x0A000000～）に割り当てられています。

本アクセス関数による SRAM へのアクセスは以下のような特徴を持ちます。

- アクセスはリード、ライト共に本ドキュメントで示すアクセス関数を用います。アクセスの最小単位はリード、ライト共に 1byte です。
- SRAM に CTRDG_WriteAgbSram 関数を用いてライトを行ったしても、データが正常に書き込まれている保証はありません。データが正常に書きこまれたかどうかを高い信頼度で確認するには書き込み後に CTRDG_VerifyAgbSram を行って下さい。

SRAM アクセス関数使用時の注意点（重要）

（1） SRAM の注意点

すべてのアクセス関数内では一定期間カートリッジバスがロックされますので注意してください。

4.2.1 関数リファレンス（SRAM デバイス）

void **CTRDG_ReadAgbSram** (u32 src, void* dst, u32 size)

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

＜引数＞	u32 *src	リード元の SRAM のアドレス（メモリマップ上のアドレス）
	void *dst	リードしたデータを格納するワーク領域のアドレス （メモリマップ上のアドレス）
	u32 size	バイト単位でのリードサイズ
	CTRDG_TASK_FUNC callback	Read 処理終了時に呼び出されるコールバック関数 (非同期関数の場合のみ)
＜戻り値＞	なし	

引数で指定された SRAM アドレスから size バイトのデータをワーク領域の dst アドレス以降に読み出します。

void **CTRDG_WriteAgbSram** (u32 dst, const void* src, u32 size)

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

＜引数＞	u32 *src	ライト元のワーク領域のアドレス
	void *dst	ライト先の SRAM のアドレス（メモリマップ上のアドレス）
	u32 size	バイト単位でのライトサイズ
	CTRDG_TASK_FUNC callback	Write 処理終了時に呼び出されるコールバック関数 (非同期関数の場合のみ)
＜戻り値＞	なし	

引数で指定されたワーク領域アドレスから、size バイトのデータを SRAM の dst アドレス以降に書き込みます。

u32 **CTRDG_VerifyAgbSram** (u32 tgt, const void* src, u32 size)

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

＜引数＞	u32 *tgt	ベリファイ先 SRAM アドレスへのポインタ （書き込み先のデータ、メモリマップ上のアドレス）	
	void *src	ベリファイ元ワーク領域アドレスへのポインタ（オリジナルのデータ）	
	u32 size	バイト単位でのベリファイサイズ	
	CTRDG_TASK_FUNC callback	Verify 処理終了時に呼び出されるコールバック関数 (非同期関数の場合のみ)	
＜戻り値＞	u32 errorAdr (同期関数の場合のみ)	正常終了 ベリファイエラー	⇒ 0 ⇒ デバイス側エラーアドレス

ワーク領域の src アドレスからのデータと SRAM の tgt アドレスのデータを size バイト分ベリファイします。

本関数は同期版では、ベリファイが正常に終了したならば 0 を返し、ベリファイエラーがあったならばエラーの発生したアドレスを返します。

非同期関数では、本ルーチンの呼び出し後に返ってくるコールバック関数の引数である構造体 CTRDGTaskInfo のメンバ result を参照することで、 Verify 処理に成功したのかを知ることができます。

u32 **CTRDG_WriteAndVerifyAgbSram** (u32 dst, const void* src, u32 size)

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

＜引数＞	u32 *dst	ライト先 SRAM アドレスへのポインタ（メモリマップ上のアドレス）	
	void *src	ライト元ワーク領域アドレス	
	u32 size	バイト単位でのライトサイズ	
	CTRDG_TASK_FUNC callback	WriteAndVerify 処理終了時に呼び出されるコールバック関数 (非同期関数の場合のみ)	
＜戻り値＞	u32 errorAdr (同期関数の場合のみ)	正常終了 ベリファイエラー	⇒ 0 ⇒ デバイス側エラーアドレス

本関数は、内部で CTRDG_WriteAgbSram で書き込みを行った後 CTRDG_VerifyAgbSram でベリファイを行い、エラーの場合は最大で CTRDG_AGB_SRAM_RETRY_MAX（ctrdg_sram.h にて定義）回リトライを行います。

本関数は同期関数では、ベリファイが正常に終了したならば 0 を返し、ベリファイエラーがあったならばエラーの発生したアドレスを返します。

非同期関数では、本ルーチンの呼び出し後に返ってくるコールバック関数の引数である構造体 CTRDGTaskInfo のメンバ result を参照することで、 WriteAndVerify 処理に成功したのかを知ることができます。

4.3 512Kbit, 1Mbit FLASH

FLASH は、メモリマップ上のカートリッジ RAM 領域（0x0A000000～0xA00FFFF）に割り当てられています。（1Mbit FLASH については、512Kbit×2 バンク構成になっています。

本デバイスへのアクセスは、使用される複数種類の FLASH の仕様の違いを吸収するため、512Kbit を論理的に 32Kbit(4Kbyte)単位の 16 個のセクタに分割し、このセクタ単位で行います。

アクセス関数による FLASH へのアクセスは以下のような特徴を持ちます。

- アクセスはリード、ライト共に本ドキュメントで示すアクセス関数を用います。アクセスの最小単位は、リード 1byte、ライトはセクタ(4Kbyte)単位となります。
- 1M FLASH の場合は、各アクセス関数内でバンク切り換えを行っていますが、開発者が意識する必要はありません。
- CTRDG_WriteAgbFlashSector が正常終了したとしても、データが正常に書き込まれている保証はありません。データが正常に書きこまれたかどうかを高い信頼度で確認するには書き込み後に CTRDG_VerifyAgbFlash を行って下さい。

FLASH アクセス関数使用時の注意点（重要）

(1) 512Kbit, 1Mbit FLASH 共通の注意点 1（アクセス関数使用前にアプリケーションで必要な処理）

アプリケーション側で AGB カートリッジ側に FLASH が搭載されていることと、容量（512Kbit または 1Mbit）を特定するまでは、いかなる FLASH 関数も使用してはいけません。

（CTRDG_IdentifyAgbBackup 関数も例外ではありません。）

(2) 512Kbit, 1Mbit FLASH 共通の注意点 2（アクセス関数動作）

現在、1タイトルのゲームに対して複数メーカーの FLASH が使用されることになっています。このためアクセス関数の形式は全ての FLASH で共通のものとしていますが、デバイスの仕様が異なることからアクセス関数内部での動作は各 FLASH 毎に異なり、その実行時間にもかなりの違いがあります。

よってプログラムの際にはこのデバイスの差異を十分に考慮して、両者で正常に動作するよう注意して下さい。

(3) 512Kbit, 1Mbit FLASH 共通の注意点 3（イレース中断後のリード動作）

FLASH のメーカーによっては、イレース中に電源 OFF されるとそのセクタ内において、リード毎にデータが変化してしまう状態（以降リード不安定状態）に陥ることがあります。リード不安定状態は、該当セクタを一度イレースすると回復します。例えば、ReadFlash の後に VerifyFlash を用いてデータベリファイを行う場合、（同じアドレスのデータであっても）2 回の読出しデータが一致せずエラーとなることがあります。このような場合は、該当セクタに対してイレースを含む処理 （CTRDG_EraseAgbFlash 又は CTRDG_WriteAgbFlashSector 等）を行い、データの初期化や修復を試みて下さい。

(4) 512Kbit, 1Mbit FLASH 共通の注意点 4（割り込み）

本アクセス関数内では、すべての FLASH アクセス関数内において一定期間すべての割り込みが禁止されることや、カートリッジバスがロックされることがありますので注意してください。(CTRDG_IdentifyAgbBackup 関数も含みます。)

特に、FLASH アクセス関数コール時はダイレクトサウンドおよび V･H ブランク同期、表示同期、カートリッジリクエスト

等の特定のタイミングで自動起動する DMA は使用しないでください。

(5) 512Kbit, 1Mbit FLASH 共通の注意点 5(チックを使用したタイムアウト処理)

本アクセス関数では、FLASH デバイスへのアクセス時のタイムアウト計測用にチックを使用しています。よって、下記の関数を呼ぶ前に OS_InitTick 関数を呼んでおく必要があります。

【該当する関数】

CTRDG_IdentifyAgbBackup,

CTRDG_EraseAgbFlashChip, CTRDG_EraseAgbFlashChipAsync

CTRDG_EraseAgbFlashSector, CTRDG_EraseAgbFlashSectorAsync

CTRDG_WriteAgbFlashSector, CTRDG_WriteAgbFlashSectorAsync

CTRDG_WriteAndVerifyAgbFlash, CTRDG_WriteAndVerifyAgbFlashAsync

(6) 1Mbit FLASH の注意点

1Mbit FLASH は、メモリマップ上のカートリッジ RAM 領域(0x0A000000～0xA00FFFF)を 512Kbit×2 バンクで共有しているため、下記関数で返されるベリファイエラーアドレスにバンク情報は含まれないことに留意してください。

(ベリファイエラーアドレスを返す関数:CTRDG_VerifyAgbFlash, CTRDG_WriteAndVerifyAgbFlash CTRDG_VerifyAgbFlashAsync, CTRDG_WriteAndVerifyAgbFlashAsync)

※※ Flash の書き換え寿命についての注意事項(重要) ※※

一般的に、フラッシュメモリは、セクタ毎に書き換え回数に制限がありますので、保存の仕方に注意が必要です

例えば、パラメータ入力画面などで頻繁にセーブするルーチンを組んだり、通信途中で頻繁にメモリに書きこむようような使い方などはしてはいけません。

当然ながら、頻繁に書き換えを行うオートセーブを行うようなゲームでは使用してはいけません。

これらを守らない場合、予期せず商品寿命を極端に縮めてしまうことがありますので、十分にご注意ください。

＜参考テクニック＞

データの書き換え間隔を引き延ばしたり、同一セクタに書きこむのではなく、複数のセクタを順番に使用して1セクタあたりの書き換え回数を減らすようにしてください。

＜備考＞

AGB で使用するフラッシュメモリは、イレースと書き込みについては 1 セクタ辺り最低 1 万回のメーカー保証のものを使用しております。これは毎日 30 回程度のセーブで 1 年間程度で寿命となる回数ですので、十分ご注意ください。

4.3.1 関数リファレンス（FLASH デバイス）

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)

＜引数＞	u16 sec_num	対象セクタ№
	u32 offset	セクタ内のバイト単位のオフセット
	u8 *dst	リードしたデータを格納するワーク領域のアドレス （メモリマップ上のアドレス）
	u32 size	バイト単位でのリードサイズ
	CTRDG_TASK_FUNC callback	Read 処理終了時に呼び出されるコールバック関数 (非同期関数の場合のみ)
＜戻り値＞	なし	

FLASH の対象セクタ№内における offset バイト先のアドレスから、size バイト分のデータをワーク領域の dst アドレス以降に読み出します。

セクタ境界をまたいでリードサイズを指定した場合も正常に動作します。

u16 **CTRDG_EraseAgbFlashChip**(void)

void **CTRDG_EraseAgbFlashChipAsync**(CTRDG_TASK_FUNC callback)

＜引数＞	CTRDG_TASK_FUNC callback	EraseChip 処理終了時に呼び出されるコールバック関数 (非同期関数の場合のみ)	
＜戻り値＞	u16 result (※1) (同期関数の場合のみ)	正常終了 チップイレースタイムアウトエラー デバイス内部エラー	⇒ 0 ⇒ 0xc003 ⇒ 0xa003（1M FLASH のみ）

チップ全体を完全にイレースします。

非同期関数では、本ルーチンの呼び出し後に返ってくるコールバック関数の引数である構造体 CTRDGTaskInfo のメンバ result を参照することで、Erase 処理に成功したのかを知ることができます。

u16 **CTRDG_EraseAgbFlashSector**(u16 sec_num)

void **CTRDG_EraseAgbFlashSectorAsync**(u16 sec_num, CTRDG_TASK_FUNC callback)

＜引数＞	u16 sec_num	対象セクタ№	
	CTRDG_TASK_FUNC callback	EraseSector 処理終了時に呼び出されるコールバック関数 (非同期関数の場合のみ)	
＜戻り値＞	u16 result (※1) (同期関数の場合のみ)	正常終了 パラメータエラー(secNo>0x0f) セクタイレースタイムアウトエラー デバイス内部エラー	⇒ 0 ⇒ 0x80ff ⇒ 0xc002 ⇒ 0xa002（1M FLASH のみ）

対象セクタ№のデータを 1 セクタ分イレースします。

このルーチンは書込みルーチン CTRDG_WriteAgbFlashSector の中でコールされるため、通常は書込み前にこのルーチンをコールする必要はありません。また、同期関数では対象セクタ№が範囲外の時はパラメータエラーを返します。

非同期関数では、本ルーチンの呼び出し後に返ってくるコールバック関数の引数である構造体 CTRDGTaskInfo のメンバ result を参照することで、Erase 処理に成功したのかを知ることができます。

u16 **CTRDG_WriteAgbFlashSector**(u16 sec_num,u8 *src)

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

＜引数＞	u16 sec_num	対象セクタ№	
	u8 *src	書込み元アドレス(メモリマップ上のアドレス)	
	CTRDG_TASK_FUNC callback	Write 処理終了時に呼び出されるコールバック関数 (非同期関数の場合のみ)	
＜戻り値＞	u16 result (※1) (同期関数の場合のみ)	正常終了 パラメータエラー(secNo>0x0f) セクタイレースベリファイエラー セクタイレースタイムアウトエラー イレース時デバイス内部エラー プログラムタイムアウトエラー プログラム時デバイス内部エラー	⇒ 0 ⇒ 0x80ff ⇒ 0x8004（三洋製 FLASH のみ） ⇒ 0xc002 ⇒ 0xa002（1M FLASH のみ） ⇒ 0xc001 ⇒ 0xa001（1M FLASH のみ）

src アドレスから1セクタ分(4kbyte)のデータを対象セクタ№に書き込みます。

本ルーチン内で上記の CTRDG_EraseAgbFlashSector をコールし、セクタを消去してから書込みを行います。

また、対象セクタ№が範囲外の時はパラメータエラーを返します。

本ルーチンの実行中、グローバル変数 flash_remainder を参照することで、残バイト数を知ることができます。

非同期関数では、本ルーチンの呼び出し後に返ってくるコールバック関数の引数である構造体 CTRDGTaskInfo のメンバ result を参照することで、 Write 処理に成功したのかを知ることができます。

u32 **CTRDG_VerifyAgbFlash** (u16 sec_num,u8 *src,u32 size)

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

＜引数＞	u16 sec_num	対象セクタ№	
	u8 *src	ベリファイ元アドレス(メモリマップ上のアドレス)	
	u32 size	ベリファイサイズ(byte)	
	CTRDG_TASK_FUNC callback	Verify 処理終了時に呼び出されるコールバック関数 (非同期関数の場合のみ)	
＜戻り値＞	u16 errorAdr (同期関数の場合のみ)	正常終了 ベリファイエラー	⇒ 0 ⇒ デバイス側エラーアドレス

src アドレスからのデータと FLASH の対象セクタ№のデータを size バイト分ベリファイします。

セクタ境界をまたいでベリファイサイズを指定した場合も正常に動作します。

本関数は、ベリファイが正常に終了したならば 0 を返し、ベリファイエラーがあったならばエラーの発生したアドレスを返します。なお、本ルーチンではパラメータチェックは行っていません。

非同期版では、本ルーチンの呼び出し後に返ってくるコールバック関数の引数である構造体 CTRDGTaskInfo のメンバ result を参照することで、 Verify 処理に成功したのかを知ることができます。

u32 **CTRDG_WriteAndVerifyAgbFlash** (u16 sec_num, u8 *src,u32 verifysize)

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

＜引数＞	u16 sec_num	対象セクタ№	
	u8 *src	書込み元アドレス（メモリマップ上のアドレス）	
	u32 verifysize	ベリファイサイズ（byte）	
	CTRDG_TASK_FUNC callback	WriteAndVerify 処理終了時に呼び出されるコールバック関数 (非同期関数の場合のみ)	
＜戻り値＞	u16 result (※1) (同期関数の場合のみ)	正常終了	⇒ 0
		パラメータエラー(secNo>0x0f)	⇒ 0x80ff
		セクタイレースベリファイエラー	⇒ 0x8004（三洋製 FLASH のみ）
		セクタイレースタイムアウトエラー	⇒ 0xc002
		イレース時デバイス内部エラー	⇒ 0xa002（1M FLASH のみ）
		プログラムタイムアウトエラー	⇒ 0xc001
		プログラム時デバイス内部エラー	⇒ 0xa001（1M FLASH のみ）
		ベリファイエラー	⇒ デバイス側エラーアドレス

本関数は、内部で CTRDG_WriteAgbFlashSector で書き込みを行った後 CTRDG_VerifyAgbFlash で verifysize バイト分ベリファイを行います。つまり、書き込みは 1 セクタ分のみ行われますが、ベリファイサイズはセクタサイズよりも小さくしたり、セクタ境界をまたいで指定することも可能です。

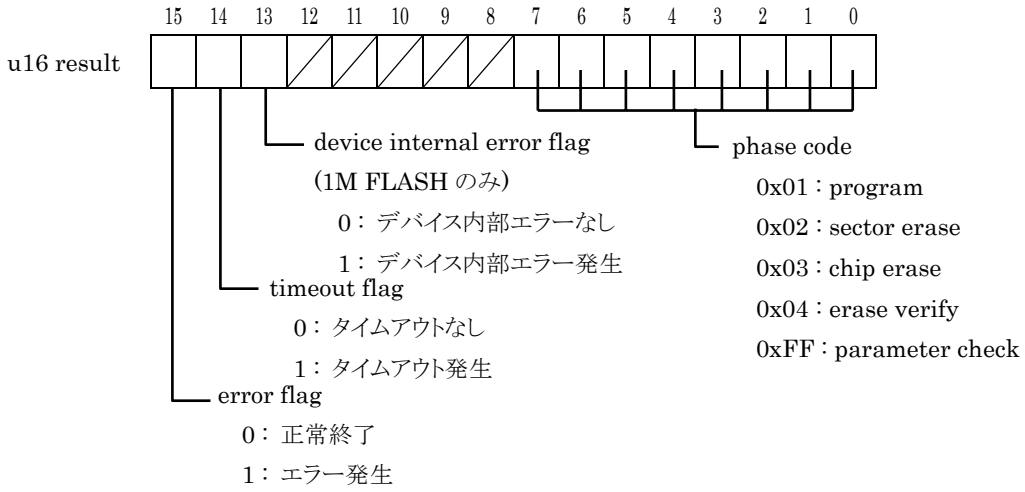
エラーの場合は最大で CTRDG_AGB_FLASH_RETRY_MAX（ctrdg_flash.h にて定義）回リトライを行います。

本関数では、ライトエラー時は 32bit 中の 16bit で上記のエラーコードを返しますが、ベリファイエラー時は 32bit のデバイス側エラーアドレスを返しますので、エラーコード確認の際はご注意下さい。

非同期版では、本ルーチンの呼び出し後に返ってくるコールバック関数の引数である構造体 CTRDGTaskInfo のメンバ result を参照することで、 WriteAndVerify 処理に成功したのかを知ることができます。

※1 エラーコードの詳細

エラー発生時は以下で構成されるエラーコードを返します。



5 各デバイスへのアクセス時のフローチャート

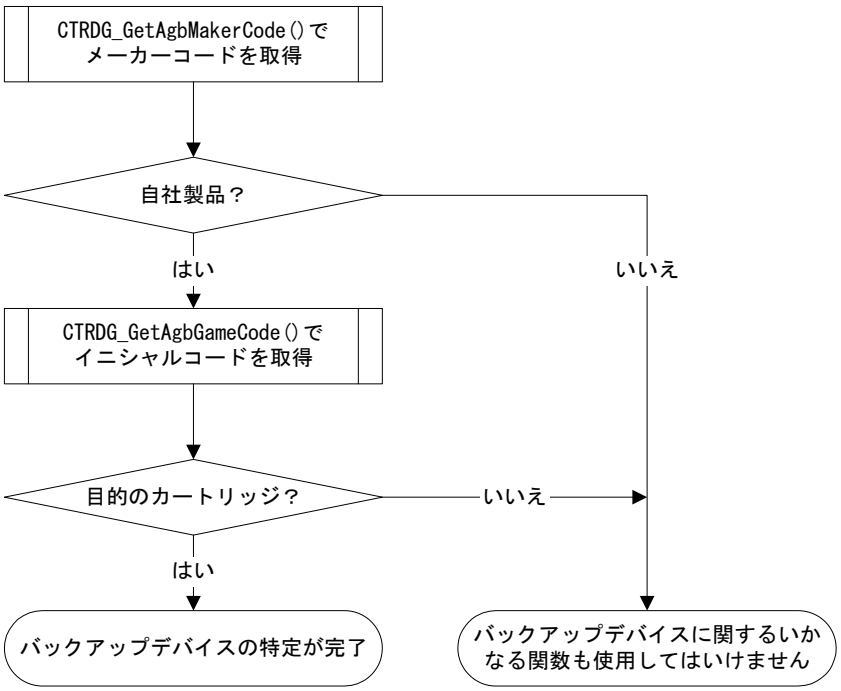
各デバイスにアクセスする際の大まかなフローを以下に記します。

5.1 全デバイス共通

本アクセス関数は、バックアップデバイスの種別毎に関数が分かれており、バックアップメモリに対して不適切な組み合わせで関数を使用する(例:SRAM に対して FLASH 用の関数を使用する等)と不具合の原因になります。

このため、バックアップメモリの種別と容量を特定するまでは、バックアップメモリに関するいかなる関数も使用してはいけません(CTRDG_IdentifyAgbBackup 関数も例外ではありません)。

バックアップメモリの種別と容量は、下記フロー図のように特定してください。

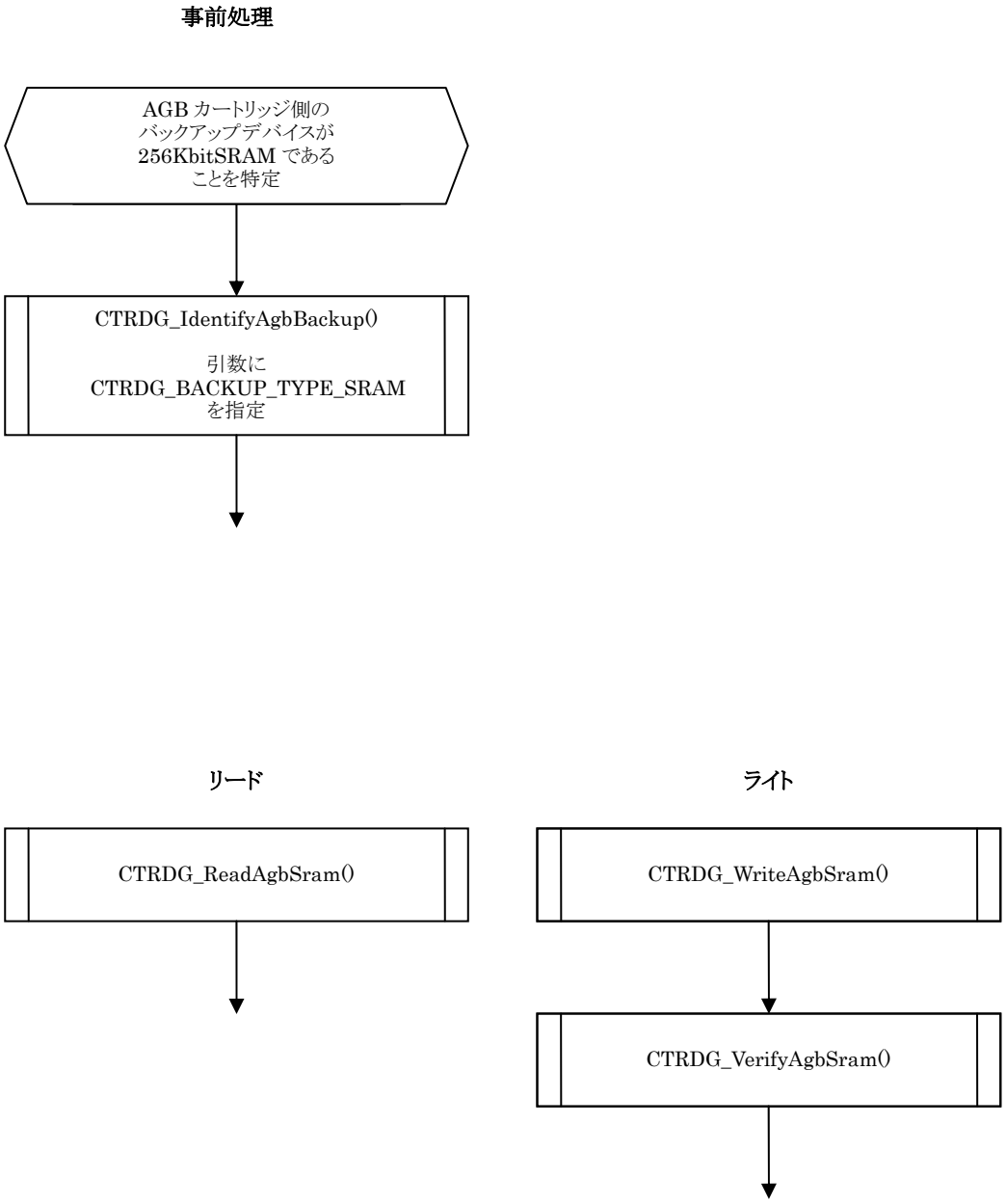


5.2 256Kbit SRAM

本アクセス関数の SRAM 関数を使用する前に、事前処理として AGB カートリッジ側のバックアップデバイスが SRAM であることを特定してください。(事前処理の詳細は前述のフロー図を参照してください。)

特定が完了してから、下記フロー図のように CTRDG_IdentifyAgbBackup 関数を呼んでください。

これらの処理を終えると、目的の SRAM に対して正しくリードやライトを行うことが可能になります。



5.3 512Kbit, 1Mbit FLASH

本アクセス関数の FLASH 関数を使用する前に、事前処理として AGB カートリッジ側のバックアップデバイスが FLASH であることと、容量が 512Kbit か 1Mbit のどちらであるかを特定してください。(事前処理の詳細は前述のフロー図を参照してください。)

特定が完了してから、下記フロー図のように CTRDG_IdentifyAgbBackup 関数を呼んでください。

これらの処理を終えると、目的のフラッシュメモリに対して正しくリードやライトを行うことが可能になります。