(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4537482号
(P4537482)
(45)発行日　平成22年9月1日(2010.9.1)　(24)登録日　平成22年6月25日(2010.6.25)

(51)Int.Cl.　F I
G06F 12/00 (2006.01)　G06F 12/00 542J
G06F 12/00 597U

請求項の数 11　(全 65 頁)

(21)出願番号　特願2008-525181 (P2008-525181)
(86)(22)出願日　平成18年8月1日(2006.8.1)
(65)公表番号　特表2009-503746 (P2009-503746A)
(43)公表日　平成21年1月29日(2009.1.29)
(86)国際出願番号　PCT/US2006/030242
(87)国際公開番号　WO2007/019220
(87)国際公開日　平成19年2月15日(2007.2.15)
審査請求日　平成21年8月3日(2009.8.3)
(31)優先権主張番号　60/705,388
(32)優先日　平成17年8月3日(2005.8.3)
(33)優先権主張国　米国(US)

早期審査対象出願

(73)特許権者　506197901
サンディスク　コーポレイション
アメリカ合衆国、９５０３５、カリフォルニア州、ミルピタス、マッカシー　ブルバード　６０１
(74)代理人　100075144
弁理士　井ノ口　壽
(72)発明者　シンクレア，アラン　ダブリュー.
イギリス連邦共和国、ＦＫ２　ＯＢＵ、フォールカーク、マディストン、キャンディ、ブロードヘッド、ザ　コテージース
(72)発明者　ライト，バリー
イギリス連邦共和国、スコットランド、ＥＨ６　６ＤＥ、エディンバラ、ヘンダーソン　ストリート、４０　フラット　４
最終頁に続く

(54)【発明の名称】直接データファイル記憶メモリにおけるデータ統合およびガベージコレクション

(57)【特許請求の範囲】
【請求項１】
データがその中に書き込まれる前に個別に消去されるブロックであるメモリセルの複数のブロックを有し、かつデータを効率的に格納するために最小数の消去済ブロックのインベントリを必要とする再プログラム可能な不揮発性メモリシステムのための操作方法であって、
前記メモリシステムが、ファイルをアドレス指定するために前記メモリシステムで用いられるファイル内のオフセットと一意のファイル識別子とで識別されたデータを受信し、
前記メモリシステムが、消去済ブロック内の第１の消去済ブロックを部分的にしか満たさない１つ以上の消去済ブロック内にページとして第１のファイルの受信されたデータを格納することで部分的に満たされた第１のブロック内に消去済データ記憶容量を残し、かつ消去済ブロック内の第２の消去済ブロックを部分的にしか満たさない１つ以上の消去済ブロック内にページとして第２のファイルの受信されたデータを格納することで部分的に満たされた第２のブロック内に消去済データ記憶容量を残し、
インベントリ内の消去済ブロックの数が少なくとも最小数を下回る場合には、部分的に満たされた第１および第２のブロック内のデータが無効になる前に、前記メモリシステムが、部分的に満たされた第１および第２のブロックからのデータを消去済ブロック内の別のブロックに統合する方法。
【請求項２】
請求項１記載の方法において、

インベントリ内の消去済ブロックの数を増やす必要性に応じて、部分的に満たされた第１および第２のブロックからのデータを消去済ブロック内の別のブロックに統合することによって、前記メモリシステムが、部分的に満たされた第１および第２のブロックのうちの少なくとも１つをその後消去することでインベントリに消去済ブロックを加える方法。

【請求項３】

請求項１記載の方法において、

第１のファイルを削除するコマンドの受信に応じて、前記メモリシステムが、少なくとも部分的に満たされた第１のブロック内の第１のファイルのすべてのデータを古いものとしてマークするので、少なくとも部分的に満たされた第１のブロック内に第１のファイルからの有効なデータをまったく格納せず、それによって、部分的に満たされた第１のブロック内の第１のファイルのデータの統合がまったく必要ではなくなる方法。

【請求項４】

請求項１記載の方法において、

前記メモリシステムが第１のファイルを構成する可変のデータ量のグループを識別する複数の記録を維持し、個別のグループがグループ内のデータの隣接する論理オフセットアドレスおよび隣接する物理アドレス双方を有する方法。

【請求項５】

請求項１記載の方法において、

ファイル内のオフセットと一意のファイル識別子とをメモリシステム内に格納する方法。

【請求項６】

請求項５記載の方法において、

前記メモリシステム用のいかなる中間的論理アドレスまたは仮想アドレス空間も使用せずに、第１のファイル内のオフセットと一意のファイル識別子とを前記メモリシステム内に維持する方法。

【請求項７】

データがその中に書き込まれる前に個別に消去されるブロックであるメモリセルの複数のブロックを有する再プログラム可能な不揮発性メモリシステムであって、

最小数の消去済ブロックのインベントリがデータを効率的に格納するために維持され、

消去済ブロック内の第１の消去済ブロックを部分的にしか満たさない１つ以上の消去済ブロック内にページとして第１のファイルの受信されたデータを格納することで部分的に満たされた第１のブロック内に消去済データ記憶容量を残し、かつ消去済ブロック内の第２の消去済ブロックを部分的にしか満たさない１つ以上の消去済ブロック内にページとして第２のファイルの受信されたデータを格納することで部分的に満たされた第２のブロック内に消去済データ記憶容量を残すことにより、前記メモリシステムによってファイル内のオフセットと一意のファイル識別子とで識別されたファイルのデータをメモリブロック内に格納し、

インベントリ内の消去済ブロックの数が少なくとも最小数を下回る場合には、部分的に満たされた第１および第２のブロック内のデータが無効になる前に、前記メモリシステムが、部分的に満たされた第１および第２のブロックからのデータを消去済ブロック内の別のブロックに統合するメモリシステム。

【請求項８】

請求項７記載のメモリシステムにおいて、

ブロックが消去された後で、かつそのブロック内の有効なデータが消去済ブロック内の別のブロック内に統合された後で、インベントリ内の消去済ブロックの数を増やす必要性に応じて、前記インベントリが、第１および第２のブロックのうちの少なくとも１つを含むメモリシステム。

【請求項９】

請求項７記載のメモリシステムにおいて、

第１のファイルを削除するコマンドの受信に応じて、少なくとも部分的に満たされた第

１のブロック内の第１のファイルのすべてのデータを古いものとしてマークするので、少なくとも部分的に満たされた第１のブロック内に第１のファイルからの有効なデータをまったく格納せず、それによって、部分的に満たされた第１のブロック内の第１のファイルのデータの統合がまったく必要ではなくなるメモリシステム。

【請求項１０】

請求項７記載のメモリシステムにおいて、

ファイル内のオフセットと一意のファイル識別子とをメモリシステム内に格納するメモリシステム。

【請求項１１】

請求項１０記載のメモリシステムにおいて、

前記メモリシステムがシステムコントローラをさらに備え、メモリシステム用のいかなる中間的論理アドレスまたは仮想アドレス空間も使用せずに、そのコントローラによって、第１のファイル内のオフセットと一意のファイル識別子とを前記メモリシステム内に維持するメモリシステム。

【発明の詳細な説明】

【技術分野】

【０００１】

本願は、一般的にはホストデバイスとメモリシステムとの間のインターフェイスの管理を含む、半導体フラッシュメモリのような再プログラム可能な不揮発性メモリシステムの操作に関し、より具体的には、共通の大容量メモリ論理アドレス空間（ＬＢＡ）インターフェイスではなくデータファイルインターフェイスの効率的な使用に関する。

【背景技術】

【０００２】

本願明細書中には、すべて２００５年２月１６日にAlan W. Sinclair単独またはPeter J. Smithと共同で出願された同時継続中の米国特許出願第１１／０６０，１７４号（特許文献１）、第１１／０６０，２４８号（特許文献２）および第１１／０６０，２４９号（特許文献３）（以下、「先行出願」と称する）に記載されるフラッシュメモリの様々な操作における開発が記載されている。

【０００３】

さらなる開発が、Alan W. SinclairおよびBarry Wrightの米国特許出願、すなわち、すべて２００６年５月８日に出願された米国特許出願第１１／３８２，２２４号（特許文献４）および第１１／３８２，２３２号（特許文献５）、ならびに仮特許出願第６０／７４６，７４０号（特許文献６）および第６０／７４６，７４２号（特許文献７）、さらにはすべて２００６年７月２１日出願された「Indexing Of File Data In Reprogrammable Non-Volatile Memories That Directly Store Data Files」（特許文献８）、「Reprogrammable Non-Volatile Memory Systems With Indexing Directly Stored Data Files 」（特許文献９）、「Methods of Managing Blocks in NonVolatile Memory」（特許文献１０）および「NonVolatile Memory With Block Management」（特許文献１１）という米国特許出願に記載されている。

【０００４】

すべての特許、特許出願、記事およびその他の刊行物、文書ならびに本願明細書中で参照される事物は、すべての目的のために、その全体が本願明細書において参照により援用されている。含まれる刊行物、文書または事物のいずれかと本願との間の用語の定義または使用におけるいずれかの不一致または矛盾の範囲まで、本願の定義または使用がまさるものとする。

【特許文献１】米国特許出願第１１／０６０，１７４号

【特許文献２】米国特許出願第１１／０６０，２４８号

【特許文献３】米国特許出願第１１／０６０，２４９号

【特許文献４】米国特許出願第１１／３８２，２２４号

【特許文献５】米国特許出願第１１／３８２，２３２号

【特許文献６】米国仮特許出願第６０／７４６，７４０号
【特許文献７】米国仮特許出願第６０／７４６，７４２号
【特許文献８】２００６年７月２１日出願された「Indexing Of File Data In Reprogrammable Non-Volatile Memories That Directly Store Data Files」という米国特許出願
【特許文献９】２００６年７月２１日出願された「Reprogrammable Non-Volatile Memory Systems With Indexing Directly Stored Data Files 」という米国特許出願
【特許文献１０】２００６年７月２１日出願された「Methods of Managing Blocks in NonVolatile Memory」という米国特許出願
【特許文献１１】２００６年７月２１日出願された「NonVolatile Memory With Block Management」という米国特許出願
【特許文献１２】本願と同時に出願されたSergey Anatolievich Gorobetsによる「Interfacing Systems Operating Through A Logical Address Space and on a Direct Date File Basis 」
【特許文献１３】米国特許出願第１０／７４９，８３１号
【特許文献１４】米国特許出願第１０／７５０，１５５号
【特許文献１５】米国特許出願第１０／９１７，８８８号
【特許文献１６】米国特許出願第１０／９１７，８６７号
【特許文献１７】米国特許出願第１０／９１７，８８９号
【特許文献１８】米国特許出願第１０／９１７，７２５号
【特許文献１９】米国特許出願第１１／１９２，２００号
【特許文献２０】米国特許出願第１１／１９２，３８６号
【特許文献２１】米国特許出願第１１／１９１，６８６号
【発明の開示】
【０００５】
データ統合(data consolidation)は本願明細書中で、ガベージコレクションとは別に扱われ、これら２つのプロセスは、異なるアルゴリズムにより少なくとも部分的に実施される。ファイルまたはメモリブロックが古い(obsolete)データを含む場合、ファイルまたはブロックの有効なデータを１つ以上の他のブロックへ移動するためにガベージコレクション操作が用いられる。ガベージコレクション操作は、有効なデータをより少数のブロックへ集め、こうして当初のソースブロックがひとたび消去されると古いデータにより占められる容量を解放する。データ統合において、通常、新しいファイルの書き込みの結果として作られるような、１つの部分的に満たされたブロックの有効なデータは、別の部分的に満たされたブロックの有効なデータと組み合わされる。古い重複データを現在含んでいる、データのソースであった当初のブロックの一方または両方は、ガベージコレクションを予定する。古いデータにより占められたメモリ記憶容量を回復するために個別のガベージコレクション操作を予定するためにキュー(queue) が提供されるが、データ統合は好ましくは、ガベージコレクションが全く予定されておらず条件がその他の点で統合について十分である場合に行われる。
【０００６】
新しく書き込まれたファイルのデータ統合をそのファイルが閉じられたすぐ後に予定するのではなく、新しく書き込まれたファイルのデータは、ホストからの受信後にそれらがプログラムされた当初のブロック中で維持される。これは最も一般的には、新しいデータで部分的に満たされたブロックを含んでいる。データファイルが、古いデータを作成するやり方で削除または更新されることは珍しくないので、部分的に満たされたブロックのデータの統合は、ファイルが閉じられた後できるだけ長く延期される。ファイルは、どのデータも再配置する必要なく削除できる。従って、そのような統合が必要になる前にファイルが削除または更新されれば、統合は回避される。そのような統合は、メモリが一杯になると、新しいデータのさらにプログラミングのための十分な消去済ブロックがあるように、必要になり得る。しかし、論理インターフェイスが用いられる場合とは反対に、ファイ

ルベースのメモリは、ホストにより削除されたデータファイルを維持しないので、たとえ統合が遅延されたとしても、メモリは通常、十分な数の消去済ブロックを有する。従って、省略された統合により取られる時間が節約され、結果としてメモリの性能が改善される。

【０００７】

以下に記載される、直接ファイル記憶におけるいくつかの他の展開があり、それらは、以下のように要約できる。

１．ひとたびファイルが閉じられると、そのファイルは、古いデータを含んでいない限り、ファイルガベージコレクションキューに加えられない。

２．ファイルのガベージコレクションは、別のファイルからのデータを含む共通ブロックを作成しない。ガベージコレクションの間にコピーされなければならないファイルのためのデータは、そのファイルのための現在のプログラムブロックにプログラムされる。プログラムブロックは、ガベージコレクションの最後に部分的にプログラムされて残る。

３．ファイルの削除によって古いものにされるブロック中のファイルグループの結果として、データが共通ブロックから再配置されなければならない場合、残る有効なデータはプログラムブロックの利用可能な空間に再配置される。

４．プログラムブロックの全部または一部に直接書き込まれたホストデータの移動は、回避される。

５．ファイルのガベージコレクションの間に、データは、そのファイルのためのプログラムブロックに再配置される。専用の中間コピーブロックは皆無である。

６．データは、メモリシステムのコントローラバッファメモリとデータの完全なセクタ中のメモリセルアレイとの間で転送される。これにより、プログラミングの間のＥＣＣの発生およびデータを読み出す間のＥＣＣのチェックが可能になる。

７．共通ブロック中のデータグループまたはファイルグループの開始は、メタページの開始に整合される。オンチップコピーはその結果、ブロック統合のために用いられ得る。プログラムブロック中のデータグループは、物理的構造への特定の整合を全く有さない。

８．フラッシュメモリ内のスワップブロックは、アクティブではない、すなわち、最近のwrite コマンドが別のファイルに関する場合の、開いているファイルのための揮発性コントローラバッファメモリ中に維持されたデータの保全コピーを行うために用いられる。スワップブロックは、利用可能なバッファメモリ容量が、より多くの開いているファイルを、それらの間でのスワップ操作の使用によってサポートできるようにするため、仮想バッファ構造の一部として用いられることもある。

９．ＦＩＴファイルが、その現在の範囲があふれるので、別のＦＩＴ範囲に移動される場合、ディレクトリ中のファイルデータポインタは、新しいＦＩＴ範囲を反映するように更新される。

１０．ＦＩＴ範囲についてのＦＩＴ更新ブロック中のデータは、ＦＩＴ更新ブロック中の範囲についてのデータ量がしきい値を越える場合に、範囲についてのＦＩＴブロック中のデータと統合される。これにより、新しいファイルについてのデータが、ＦＩＴブロックに統合されることが可能になる。

１１．ＦＩＴ更新ブロックの圧縮の間、十分な消去済空間が存在すれば、閉じられたファイルのためのＦＩＴファイルは、その範囲のためのＦＩＴブロックに再配置される。さもなければ、そのＦＩＴファイルは、圧縮されたＦＩＴ更新ブロックに再配置される。

１２．ホストは、メタブロックのサイズと同じ、等しいサイズをセット中のすべてのファイルが有するようにwrite_pointer およびread_pointer コマンドを用いることができ、セット中の１つのファイルが、このファイルが閉じられた直後に単一のメタブロックに統合されるようにするために、close およびidleコマンドを用い得る。

１３．ホストコマンドのセットは、命令されたデータ書き込みまたは読み出しが始まるメモリドレスを与えるwrite_pointer およびread_pointer の値についてのcompanion コマンドを含む指定されたfileIDについてのreadコマンドおよびwrite コマンドを含む。

【０００８】

添付図面は、本願の一部として含まれ、以下の説明において参照される。
【発明を実施するための最良の形態】
【０００９】
１．直接データファイル・プラットフォーム
１．１概要
直接データファイル・プラットフォームを有するメモリカードが図１－１に例示してある。直接データファイル・プラットフォームは、データがファイル名およびファイルオフセットアドレスにより識別される、ファイル組織化データ記憶デバイスである。これは、データ記憶以外の機能を含み得るメモリカードにおける記憶プラットフォームとして働く。ファイルデータは、外部ファイルインターフェイスチャネルによりこのプラットフォーム中でアクセスされる。
【００１０】
記憶デバイスは、論理アドレスを全く持たない。独立したアドレス空間が各ファイルについて存在し、メモリ管理アルゴリズムが、データのファイル構造に従ってデータ記憶を組織する。直接データファイル・プラットフォームにおいて用いられるデータ記憶の組織化は、従来の論理ブロック化メモリ管理を有する従来のファイルシステムを組み込むファイル記憶デバイスの操作特性と比べて操作特性にかなりの向上をもたらす。
【００１１】
１．２プラットフォームの構成要素
直接データファイル・プラットフォームは、図１－２に例示されるような機能性層に構造化された以下の構成要素を有する。
直接データファイルインターフェイス：カード中の他の機能ブロックから、ファイル名およびファイルオフセットアドレスにより識別されたデータへのアクセスを提供するファイルＡＰＩ。
ファイル－ツー－フラッシュマッピングアルゴリズム：ファイル断片化を排除し最高の性能および耐久性を提供するファイル組織化データ記憶のための方式。
ファイルデータのプログラミング：ファイル－ツー－フラッシュマッピングアルゴリズムに従うファイルデータをプログラムすること。
ファイルデータの読み出し：フラッシュメモリからの、ファイルオフセットアドレスにより指定されたデータを読み出すこと。
ファイル削除：削除されたファイルについてのデータを含むブロックを識別し、それらのブロックをガベージコレクションキューに加えること。
ガベージコレクション：古いデータにより占められたメモリ容量を回復するために実行される操作。これらの操作は、ブロックを消去するために、有効なデータを別の位置へコピーすることを伴うことがある。
ファイルインデックス化：ファイルインデックス化により、あるファイルについての有効なデータグループの位置を、オフセットアドレス順に、識別することが可能になる。
データバッファリングおよびプログラミング：プログラムされるべきデータについてのバッファメモリの使用、およびプログラムブロック中のファイルデータをプログラムするシーケンス。
消去ブロック管理：ファイルデータまたは制御情報の記憶のための割り当てに利用可能なデバイス中の消去ブロックのプールの管理。
ブロック状態管理：ファイルデータ記憶のためのブロックが分類できる８つの状態間の移行。
制御データ構造：目的専用のフラッシュブロック中に格納された制御データ構造。
【００１２】
２．直接データファイルインターフェイス
直接データファイルインターフェイスは、直接データファイル・プラットフォームへのＡＰＩであり、フラッシュ大容量データ記憶を含むデバイス内のフラッシュメモリ管理のためのバックエンドシステムを形成する。

【００１３】
２．１コマンドセット
以下のセクションは、複数のソースとのファイルベースのインターフェイスをサポートする一般的なコマンドを定義する。コマンドは、６つのクラスにおいて定義される。
１．ファイルコマンド
２．データコマンド
３．ファイル情報コマンド
４．ストリームコマンド（モデリングについてのみ）
５．状態コマンド
６．デバイスコマンド
【００１４】
２．１．１ファイルコマンド（図２－１参照）
ファイルは、デバイス内でfileIDにより独立して識別されるオブジェクトである。ファイルは、ホストにより作成されたデータのセットを含むか、またはデータを全く持たないことがあり、その場合、ファイルは、ディレクトリまたはフォルダを表す。
【００１５】
２．１．１．１create（作成）
createコマンドは、デバイス中のディレクトリ内の＜fileID＞により識別されたエントリを作成する。＜fileID＞パラメータが省略されれば、デバイスは、利用可能な値をファイルに割り当て、それをホストに返す。これは、ファイルを作成する通常の方法である。
【００１６】
ホストは代わりに、＜fileID＞値をファイルに割り当てることができる。fileIDの特定の値がホストインターフェイスプロトコル内のファイルの特定タイプを示せば、この方法が用いられ得る。例えば、ルートディレクトリは、ホストにより特定のfileIDを割り当てられ得る。
【００１７】
２．１．１．２open（開く）
このコマンドは、＜fileID＞により指定されたファイルのための後のデータコマンドの実行を可能にする。ファイルが存在しなければ、エラーメッセージが返される。ファイルのためのwrite_pointer がファイルの終わりに設定され、ファイルのためのread_pointer がファイルの最初に設定される。file_infoのためのinfo_write_pointer は、file_infoの終わりに設定され、ファイルのためのinfo_read_pointer は、file_infoの最初に設定される。同時に開いていることができるファイルの最大数がある。この数が超過されれば、コマンドは実行されず、エラーメッセージが返される。同時に開いているファイルの最大数は、例えば、８であり得る。
指定されたファイルに書き込むためのデバイス内の資源は、その後のwrite, insert またはdeleteコマンドの受信後にのみ利用可能にされる。
【００１８】
２．１．１．３close （閉じる）
このコマンドは、指定されたファイルのためのその後のデータコマンドの実行をできなくする。ファイルについてのwrite_pointer, read_pointer, info_write_pointer およびinfo_read_pointer 値は、無効になる。
【００１９】
２．１．１．４delete（削除）
deleteコマンドは、＜fileID＞により指定されたファイルについてのディレクトリ、ファイルインデックステーブルおよび情報テーブルエントリが削除されるべきであることを示す。ファイルのためのデータが消去され得る。削除されたファイルは、その後アクセスできない。
【００２０】
２．１．１．５erase （消去）

eraseコマンドは、＜fileID＞により指定されたファイルについてのディレクトリ、ファイルインデックステーブルおよび情報テーブルエントリが削除されるべきであることを示す。何か他のコマンドが実行される前に、ファイルデータは消去されなければならない。消去済ファイルは、その後アクセスできない。

【００２１】

２．１．１．６list_files

ディレクトリ中のすべてのファイルについてfileID値は、list_filesコマンドの受信に続いて、デバイスから番号順にストリームされ得る。最後のファイルに達すると、fileIDストリーミングは終了され、この条件は、statusコマンドによりホストによって識別され得る。list_filesコマンドは、何か他のコマンドの受信により終了される。

【００２２】

２．１．２データコマンド（図２－２参照）

データコマンドは、指定されたファイルのためのデータ入出力操作を開始し、ファイル内のオフセットアドレス値を定義するために用いられる。指定されたファイルは、ホストにより開かれていなければならない。そうでなければ、エラーが返される。は、ファイルが最後に開かれたときにホストへ返されたファイルハンドルである。

【００２３】

２．１．２．１write（書き込み）

writeコマンドの受信に続いてデバイスにストリームされたデータは、write_pointerの現在値により定義されたオフセットアドレスにおいて、指定されたファイルに上書きされる。writeコマンドは、ファイルについての新しいデータを書き込み、データをファイルに付加し、ファイル内のデータを更新するために用いられる。writeコマンドは、何か他のコマンドの受信によって終了される。

【００２４】

２．１．２．２insert（挿入）

insertコマンドの受信に続いてデバイスにストリームされたデータは、write_pointerの現在値により定義されたオフセットアドレスにおいて、指定されたファイルに挿入される。ファイルサイズは、挿入されたデータの長さだけ増大される。insertコマンドは、何か他のコマンドの受信によって終了される。

【００２５】

２．１．２．３remove（除去）

removeコマンドは、write_pointerの現在値により定義されたオフセットアドレスにおいて、指定されたファイルから、＜length＞により定義された連続データを削除する。ファイルサイズは＜length＞だけ減少させられる。

【００２６】

２．１．２．４read（読み出し）

read_pointerの現在値により定義されたオフセットアドレスにおける指定されたファイル中のデータは、readコマンドの受信に続いてデバイスからストリームされ得る。

ファイルの終わりに達すると、データストリーミングは終了され、この条件は、statusコマンドによってホストにより識別され得る。readコマンドは、何か他のコマンドの受信によって終了される。

【００２７】

２．１．２．５save_buffer

デバイスバッファ中に含まれ、フラッシュメモリにまだプログラムされていない指定されたファイルについてのデータは、フラッシュメモリ中の一時的な位置に保存される。

その後のwrite またはinsertコマンドが受信されると、そのデータは、バッファに返され、そのコマンドに関するデータと共にフラッシュするようにプログラムされる。

【００２８】

２．１．２．６write_pointer

write_pointerコマンドは、指定されたファイルについてのwrite_pointerを、指定さ

れたオフセットアドレスに設定する。write またはinsertコマンドに続いてデータがデバイスへストリームされるにつれ、write_pointer はデバイスにより増分される。

【００２９】

２．１．２．７read_pointer

read_pointer コマンドは、指定されたファイルについてのread_pointer を、指定されたオフセットアドレスに設定する。readコマンドに続いてデータがデバイスからストリームされるにつれ、read_pointer はデバイスにより増分される。

【００３０】

２．１．３infoコマンド（図２－３参照）

file_infoは、ファイルに関連付けられているホストにより生成された情報である。file_infoの性質と内容は、ホストにより決定され、それはデバイスによって解釈されない。infoコマンドは、指定されたファイルについてのfile_info入出力操作を開始し、file_info内のオフセットアドレス値を定義するために用いられる。

【００３１】

２．１．３．１write_info

デバイスへストリームされたfile_infoは、write_infoコマンドの受信に続いて、info_write_pointer の現在値により定義されたオフセットアドレスにおいて、指定されたファイルについてのfile_infoを上書きする。指定されたファイルについてのfile_infoの内容および長さは、ホストにより決定される。write_infoコマンドは、何か他のコマンドの受信により終了される。

【００３２】

２．１．３．２read_info

info_read_pointer の現在値により定義されたオフセットアドレスにおける指定されたファイルについてのfile_infoは、read_infoコマンドの受信に続いてデバイスへストリームされ得る。file_infoの終わりに達すると、file_infoストリーミングは終了され、この条件はstatusコマンドによりホストによって識別され得る。read_infoコマンドは、何か他のコマンドの受信により終了される。

【００３３】

２．１．３．３info_write_pointer

info_write_pointer コマンドは、指定されたファイルについてのinfo_write_pointer を、指定されたオフセットアドレスに設定する。info_write_pointer は、write_infoコマンドに続いてfile_infoがデバイスにストリームされるにつれ、デバイスにより増分される。

【００３４】

２．１．３．４info_read_pointer

info_read_pointer コマンドは、指定されたファイルについてのinfo_read_pointer を、指定されたオフセットアドレスに設定する。info_read_pointer は、read_infoコマンドに続いてfile_infoがデバイスからストリームされるにつれ、デバイスにより増分される。

【００３５】

２．１．４ストリームコマンド（図２－４参照）

ストリームコマンドは、直接データファイル・プラットフォームの行動モデルと共にのみ用いられる。それらの目的は、データコマンドに関連して、ホストへまたはホストからのデータのストリーミングをエミュレートすることである。

【００３６】

２．１．４．１ストリーム

ストリームコマンドは、ホストによりプラットフォームへまたはプラットフォームから転送されるべき＜length＞により定義されたデータの中断されないデータのストリームをエミュレートする。データがバッファメモリに追加またはバッファメモリから除去されるにつれ、ストリームの残りの長さを表す変数が、プラットフォームのモデルにより減分さ

れる。
【００３７】
２．１．４．２pause （休止）
pause コマンドは、直接データファイルモデルの操作を制御しているコマンドリスト中の後続コマンドの実行の前に挿入される長さ＜time＞の遅延を挿入する。＜time＞は、マイクロ秒で定義される。
【００３８】
２．１．５状態コマンド（図２－６参照）
状態コマンドは、デバイスの状態を制御する。
【００３９】
２．１．５．１idle（アイドル）
idleコマンドは、ホストが、直接データファイルデバイスをアイドル状態にすることを示し、このアイドル状態の間、デバイスは、内部ハウスキーピング操作を実行し得る。ホストは、アイドル状態のデバイスからわざと電力を取り去らない。デバイスが内部操作でビジーであるかどうかにかかわらず、アイドル状態は、ホストによる何か他のコマンドの伝送によって終了され得る。そのような他のコマンドが受信されると、デバイス中で進行中のどのような内部操作も、指定された時間以内に中断または終了されなければならない。この時間の例は、１０ミリ秒以下である。
【００４０】
２．１．５．２standby （待機）
standby コマンドは、ホストが、直接データファイルデバイスを待機状態にすることを示し、この待機状態の間、デバイスは、内部ハウスキーピング操作を実行できない。ホストは、待機状態のデバイスからわざと電力を取り去らない。待機状態は、ホストによる何か他のコマンドの伝送によって終了され得る。
【００４１】
２．１．５．３shutdown（シャットダウン）
shutdownコマンドは、デバイスが次にビジー状態にならない場合、ホストにより電力がデバイスから取り去られることを示す。すべての開いているファイルは、shutdownコマンドに応答してデバイスにより閉じられる。
【００４２】
２．１．６デバイスコマンド（図２－６参照）
デバイスコマンドは、ホストがデバイスに問い合わせできるようにする。
【００４３】
２．１．６．１capacity（容量）
capacityコマンドに応答して、デバイスは、デバイス中に格納されたファイルデータの容量、および新しいファイルデータ用に利用可能な容量を報告する。
【００４４】
２．１．６．２status（ステータス）
statusコマンドに応答して、デバイスは、その現在のステータスを報告する。
ステータスは、以下の３つのタイプのビジーステータスを含む。
１．デバイスは、データの書き込みまたは読み出しのためのフォアグラウンド操作の実行でビジーである。
２．デバイスは、デバイスがアイドル状態にあった間に開始されたバックグラウンド操作の実行でビジーである。
３．バッファメモリはビジーであり、データの書き込みまたは読み出しのためにホストが利用できない。
【００４５】
２．１．７コマンドパラメータ
以下のパラメータは、以下で定義されるようなコマンドと共に用いられる。
【００４６】

２．１．７．１FileID
これは、デバイスのディレクトリ内のファイルを識別するために用いられるファイル識別子である。
【００４７】
２．１．７．２オフセット
オフセットは、ファイルまたはfile_infoの開始に関する、バイトで表した、ファイルまたはfile_info内の論理アドレスである。
【００４８】
２．１．７．３長さ
これは、連続オフセットアドレスを有するファイルについてのデータランの、バイトで表した長さである。
【００４９】
２．１．７．４時間
これはマイクロ秒で表した時間である。
【００５０】
３．ファイル－ツー－フラッシュマッピングアルゴリズム
直接データファイル・プラットフォームにより採用されたファイル－ツー－フラッシュマッピングアルゴリズムは、ホストが、ファイルベースのインターフェイスを介してファイルデータ書き込みおよびファイル削除操作を実行する際に、最大システム性能および最大メモリ耐久性を提供するために定義されたファイル組織化データ記憶のための新しい方式である。マッピングアルゴリズムは、フラッシュメモリ中のブロック間のファイルデータのコピーを最小限にするように設計されている。これは、１つ以上のファイルについてのデータを含むブロックの発生を最も低くするやり方でファイルデータをフラッシュブロックにマッピングすることにより達成される。
【００５１】
３．１ファイル－ツー－フラッシュマッピング原理
３．１．１ファイル
ファイルは、ホストにより作成および維持されるデータのセットである。データは、ファイル名によってホストにより識別され、ファイルの冒頭からのそのオフセット位置によってアクセスされ得る。ファイルオフセットアドレスは、ホストにより設定されることができ、書き込みポインタとしてデバイスにより増分され得る。
【００５２】
３．１．２物理メモリ構造
直接データファイル・プラットフォームは、固定サイズのメタブロック中のファイルについてのすべてのデータを格納する。メタブロックを含むフラッシュ消去ブロックの実際の数、すなわち、消去ブロックの並列性(parallelism) は、製品間で変わり得る。本願明細書全体を通して、用語「ブロック」は、「メタブロック」を示すために用いられる。
【００５３】
用語「メタページ」は、メタブロックの完全な並列性を有するページを示すために用いられる。メタページは、プログラミングの最大単位である。
用語「ページ」は、メモリのプレーン内、すなわち、フラッシュ消去ブロック内のページを示すために用いられる。ページは、プログラミングの最小単位である。
用語「セクタ」は、ＥＣＣが関連付けられた格納されたデータの単位を示すために用いられる。セクタは、フラッシュメモリとの間のデータ転送の最小単位である。
ファイルについてのオフセットアドレスと物理フラッシュメモリ構造との間で維持される指定された整合は皆無である。
【００５４】
３．１．３データグループ
データグループは、単一のメモリブロック中の隣接する物理アドレスにおいてプログラムされた、ファイル内の隣接するオフセットアドレスを持つファイルデータのセットであ

る。ファイルは通常、多数のデータグループとしてプログラムされる。データグループは、１バイトと１つのブロックとの間の任意の長さを有し得る。各データグループは、相互参照目的のためのファイル識別子情報を含むヘッダを用いてプログラムされる。ファイルのためのデータは、それが含むデータグループに従って物理メモリ中でインデックスを付けられる。ファイルインデックステーブルは、ファイルの各データグループについてのファイルオフセットアドレスおよび物理アドレス情報を提供する。

【００５５】

３．１．４プログラムブロック

ファイルは、ファイルデータがプログラムされるようにするために、ホストにより開かれなければならない。各開いているファイルは、プログラムブロックとして割り当てられた専用ブロックを有し、そのファイルのためのデータは、プログラムブロック内のプログラムポインタにより定義された位置でプログラムされる。ファイルがホストにより開かれるとき、プログラムブロックがすでに存在していなければ、ファイルのためのプログラムブロックが開かれる。プログラムポインタは、プログラムブロックの冒頭に設定される。ホストにより開かれるファイルのためにプログラムブロックがすでに存在していれば、そのプログラムブロックは、ファイルのためのデータをプログラムするために使われ続ける。

【００５６】

ファイルデータは、ファイル内のそのオフセットアドレスまたはそのオフセットアドレスのためのデータが前もってプログラムされているかどうかにかかわらず、ファイルデータがホストから受信される順序でプログラムブロック中でプログラムされる。プログラムブロックが一杯になると、そのプログラムブロックはファイルブロックとして知られ、消去済ブロックプールからの消去済ブロックは、新しいプログラムブロックとして開かれる。ファイルについてのデータを格納しているブロック間の物理アドレス関係は皆無である。

【００５７】

３．１．５共通ブロック

共通ブロックは、１つ以上のファイルについてのデータグループを含んでいる。同じファイルについての複数のデータグループが共通ブロック中に存在すれば、それらのデータグループは、隣接して配置され、隣接している単位は、ファイルグループとして知られる。データは、ブロック統合操作または共通ブロックのガベージコレクション操作の間にのみ、共通ブロックにプログラムされる。

【００５８】

共通ブロック内の個別のデータグループまたはファイルグループの開始は、メタページの開始と整合しなければならない。

ファイルグループ内のデータグループには、介入スペースがない。そのようなデータグループ間の境界は、ページ内で生じ得る。ファイルは、単一の共通ブロック中でのみデータを有するべきである（これの例外については、８．３．４参照）。

【００５９】

３．２ファイルタイプ

３．２．１プレーンファイル

プレーンファイルは、任意数の完全なファイルブロックおよび１つの部分的にプログラムされたプログラムブロックを含む。プレーンファイルは、通常逐次オフセットアドレス順で、ホストからのファイルのためのデータのプログラミングにより、または編集済ファイルのガベージコレクションにより作成され得る。プレーンファイルの例が図３－１に示してある。プレーンファイルは、開いているファイルまたは閉じられたファイルであり得る。

【００６０】

ファイルについてのさらなるデータが、プログラムブロック中のプログラムポインタにおいてプログラムされ得る。ファイルがホストにより削除されれば、そのデータを含むブ

ロックは、そのようなブロックからのデータをフラッシュメモリ中の別の位置にコピーする必要なしに、直ちに消去され得る。従って、プレーンファイル形式は非常に効率的であり、ファイルをできるだけ長くこの形式で維持することには利点がある。

【００６１】

３．２．２共通ファイル

共通ファイルは、任意数の完全なファイルブロック、および他の無関係なファイルのためのデータと共にそのファイルのためのデータを含む１つの共通ブロックを含む。例が図３－２に示してある。共通ファイルは、ガベージコレクション操作の間、またはプログラムブロックの統合によりプレーンファイルから作成され得る。

【００６２】

共通ファイルは通常、閉じたファイルであり、関連付けられた書き込みポインタを持たない。ホストが共通ファイルを開けば、プログラムブロックが開かれ、プログラムポインタは、プログラムブロックの冒頭に設定される。ファイルがホストにより削除されれば、そのファイルブロックは直ちに消去されるであろうが、無関係なファイルまたは無関係なファイルのためのデータは、共通ブロックが消去される前に、ガベージコレクション操作において共通ブロックからフラッシュメモリ中の別の位置にコピーされなければならない。

【００６３】

３．２．３編集済プレーンファイル

プレーンファイルは、ホストによりいつでも編集でき、ホストは、そのファイルのための前にプログラムされたオフセットアドレスのための更新されたデータを書き込む。例が図３－３に示してある。そのような更新されたデータは、プログラムブロック中のプログラムポインタにおいて通常のやり方でプログラムされ、結果として生じる編集済プレーンファイルは、１つ以上の古いファイルブロック中、またはプログラムブロックそれ自体の中に古いデータを含む。

【００６４】

編集済プレーンファイルは、ファイルのためのガベージコレクション操作においてプレーンファイルに復元され得る。そのようなガベージコレクションの間、任意の有効なファイルデータが、各古いファイルブロックからファイルのためのプログラムポインタにコピーされ、結果として生じる完全に古いブロックが消去される。ガベージコレクションは、可能であれば、ファイルがホストにより閉じられるまで実行されない。

【００６５】

３．２．４編集済共通ファイル

開いた共通ファイルは、ホストによりいつでも編集でき、ホストは、そのファイルのための前にプログラムされたオフセットアドレスのための更新されたデータを書き込む。例が図３－４に示してある。そのような更新されたデータは、プログラムブロック中のプログラムポインタにおいて通常のやり方でプログラムされ、結果として生じる編集済共通ファイルは、１つ以上の古いファイルブロック中、共通ブロック中、またはプログラムブロックそれ自体の中に古いデータを含む。

【００６６】

編集済共通ファイルは、ファイルのためのガベージコレクション操作においてプレーンファイル形式に復元され得る。そのようなガベージコレクションの間、任意の有効なファイルデータが、各古いファイルブロックおよび共通ブロックからファイルのためのプログラムポインタにコピーされる。結果として生じる完全に古いファイルブロックが消去され、古い共通ブロックは、別個のその後のガベージコレクション操作のために記録される。

【００６７】

３．３ガベージコレクションおよびブロック統合

３．３．１ガベージコレクション

ガベージコレクション操作は、古いデータにより占められたメモリ容量を回復するために実行される。これらのガベージコレクション操作は、１つのブロックを消去するために

、有効なデータを別の位置へコピーすることを伴うことがある。ガベージコレクションは、古いデータの作成に応答して直ちに実行される必要はない。保留中のガベージコレクション操作は、ガベージコレクションキューに記録され、その後スケジューリング・アルゴリズムに従って最適レートで実行される。

【００６８】

直接データファイル・プラットフォームは、ホストコマンドにより開始されることがあるバックグラウンドガベージコレクション操作をサポートする。これにより、ホストは、内部ハウスキーピング操作のために休止時間をデバイスに割り当てることが可能になり、これにより、ファイルがその後ホストにより書き込まれるときにより高い性能が可能になる。

【００６９】

十分なバックグラウンド時間がホストにより利用可能にされなければ、デバイスはガベージコレクションをフォアグラウンド操作として実行する。ガベージコレクション操作のバーストは、ホストからのファイルデータのプログラミングのバーストとインターリーブされる。インターリーブ動作周期は、未処理分が蓄積されないことを保証しつつ、ガベージコレクションレートを最小に維持するために適応的に制御され得る。

【００７０】

３．３．２ブロック統合

デバイス中の各プレーンファイルは、不完全に満たされたプログラムブロックを含み、かなりのボリュームの消去済容量がそのようなプログラムブロック中にロックアップされることがある。共通ブロックも、消去済容量を含み得る。従って、閉じられたファイルおよび共通ブロックのための統合プログラムブロックの進行中のプロセスは、ロックされた消去済容量を制御するために実施される。ブロック統合は、ガベージコレクション機能の一部として扱われ、同じスケジューリング・アルゴリズムにより管理される。

【００７１】

プログラムブロックまたは共通ブロック中のデータは、１つ以上の無関係なファイルのためのデータと、そのような無関係なデータを別の共通ブロックまたはプログラムブロックからコピーすることによって統合される。当初のブロックがプログラムブロックであったならば、それは共通ブロックになる。プログラムブロックを、別のプログラムブロックとではなく、古い共通ブロックと統合することが望ましい。古い共通ブロックは、古いデータを含み、従って、有効なデータをそのブロックから別の位置に移さなければならないことは避けがたい。しかし、プログラムブロックは古いデータを含まず、そのブロックからデータを別の位置へコピーすることは望ましくないオーバーヘッドである。

【００７２】

３．３．３平衡状態

ファイルデータがデバイス容量の高い百分率を占める場合、新しいファイルを書き込むための容量を作成するために、ホストは、ファイルに関する削除操作を実行しなければならない。この状態では、デバイス中のほとんどのファイルは、共通ファイル形式を有する。なぜならば、プレーンファイル形式のファイルのためのプログラムブロック中の消去済空間のための利用可能な容量がほとんどないからである。

【００７３】

共通ファイルの削除は、ガベージコレクションの間に無関係なファイルがその共通ブロックから再配置されるための有効なデータを必要とする。そのようなファイルグループのためのデータは最も一般的には、閉じられたファイルのための１つ以上のプログラムブロック中の利用可能な容量に再配置される。ホストにより最近書き込まれ次に削除されたファイルのためのプログラムブロック中の利用可能な未使用容量と、ホストにより削除されるファイルの結果として共通ブロックからファイルデータを再配置するための必要容量との間に、平衡が頻繁にある。この一般的な平衡状態は、ファイルデータをプログラムブロックから再配置する必要を低減し、ファイル－ツー－フラッシュマッピングアルゴリズムの効率に寄与する。

【００７４】
４．デバイス操作
４．１デバイス操作の実行
デバイスの操作シーケンスは、ホストから供給されたコマンドの流れにより決定される。ホストコマンドが受信されると、現在のデバイス操作が中断され、コマンドは正常に解釈される。一定のコマンドは、以下の４つの主要なデバイス操作のうちの１つの実行を引き起こす。
１．データの読み出し
２．データのプログラミング
３．ファイルの削除、または
４．ガベージコレクション
デバイス操作は、以下の条件の１つに到達するまで続く。
１．操作が完了され、
２．別のホストコマンドが受信されるか、または
３．フォアグラウンドのガベージコレクションモードにおいて、インターリーブされたバーストの最後に到達する。
【００７５】
プライオリティのガベージコレクション操作が実行のためにキューに入れられると、これらの操作は、いずれかの新しいコマンドが解釈される前に、完了される。
デバイス操作を示す全体的な流れ図が、図４－１として示してある。
【００７６】
５．ファイルデータのプログラミング
５．１ファイルデータのプログラミングのための原理
ファイルのためのデータは、ホストからのwrite またはinsertコマンドに続いてそのデータがホストからデバイスへストリームされる際にフラッシュメモリにプログラムされる。次のプログラム操作のためにバッファメモリ中に十分なデータが蓄積されたとき、データは、ファイルのためのプログラムブロックにおいてプログラムされる。この操作の説明については、第９章を参照のこと。
【００７７】
プログラムブロックが一杯になると、そのブロックは、ファイルブロックと呼ばれ、消去済ブロックプールからの消去済ブロックが、プログラムブロックとして割り当てられる。加えて、共通ブロックおよび古いブロックのためのファイルインデックステーブルおよびガベージコレクションキューが更新される。
【００７８】
ファイルデータのプログラミング手順は、ガベージコレクションのスケジューリング・アルゴリズムにより設定されるインターリーブなパラメータＮ１に従って、フォアグラウンドのガベージコレクションのバーストを開始する（セクション８．４参照）。メタページプログラム操作がフラッシュメモリにおいて開始されるときはいつでも、インターリーブなプログラムカウンタが増分され、このカウンタが値Ｎ１を超えると、フォアグラウンドモードでのガベージコレクション操作が開始される。
【００７９】
ファイルデータのプログラミングは、ホストが別のコマンドを伝送するまで、１つのメタページのユニット中に留まる。
ファイルデータのプログラミングの例を例示する流れ図が、図５－１として示してある。
【００８０】
６．ファイルデータの読み出し
６．１ファイルデータの読み出しのための原理
ホストからのreadコマンドに応答して、read_pointer により指定されたところから始まるファイルオフセットアドレスのためのデータが、フラッシュメモリから読み出され、

ファイルの終わりに達するまで、ホストに逐次返される。ファイルインデックステーブル（ＦＩＴ）が読み出され、read_pointer に対応する位置を識別するためにファイルのためのＦＩＴエントリが評価される。それに続くＦＩＴエントリは、ファイルのためのデータグループの位置を指定する。

【００８１】

ファイルデータは、ファイルの終わりに達するまで、または、ホストが別のコマンドを伝送するまで、１つのメタページの単位で読み出される。

ファイルデータ読み出しプロセスの例が、図６－１に示してある。

【００８２】

７．ファイルの削除

７．１ファイル削除のための原理

ホストからのファイルのためのdeleteコマンドに応答して、そのファイルのためのデータを含むブロックが識別され、その後のガベージコレクション操作のためにガベージコレクションキューに加えられる。ファイルを削除するための手順は、これらのガベージコレクション操作を開始せず、従って、ファイルのためのデータは、直ちに消去されない。

【００８３】

ファイルのためのＦＩＴエントリは、当初ファイルのためのデータを含み得る共通ブロックを識別するために評価される。その後、ファイルのためのＦＩＴエントリがオフセットアドレス順に評価されて、データグループが位置するデータブロックが、その後のガベージコレクションのために、共通ブロックキューまたは古いブロックキューのいずれかに加えられる。ファイルディレクトリおよびファイルインデックステーブルは次に、ファイルのためのエントリを除去するために更新される。

【００８４】

７．２ファイルの消去

ホストからのファイルのためのerase コマンドに応答して、deleteコマンドについてと同じ手順がたどられるべきであるが、ブロックは、ガベージコレクションのためのプライオリティの共通ブロックキューおよびプライオリティの古いブロックキューに加えられるファイルのためのデータを含んでいる。

【００８５】

主要なデバイス操作シーケンスは次に、何か他のホストコマンドが実行される前に、これらのブロックのためのガベージコレクション操作が実行されることを保証する。これは、erase コマンドにより識別されたファイルのためのデータが直ちに消去されることを保証する。

ファイル削除プロセスの流れ図が、図７－１として示してある。

【００８６】

８．ガベージコレクション

８．１ガベージコレクションのための原理

ガベージコレクションは、古いファイルデータにより占められたフラッシュメモリ容量を回復するために実行されなければならない操作である。ガベージコレクションは、ファイルの削除またはファイルのデータの編集の結果として必要になることがある。

【００８７】

ブロック統合は、ガベージコレクションの一形態であり、ファイルデータで不完全に満たされたブロックの消去済容量を回復し、無関係なファイルを格納するためにそれらのブロックの使用を可能にするために実行される。統合は、プレーンファイル中のプログラムブロックを共通ブロックへ変換するためにプレーンファイル中のプログラムブロックに関して、または共通ブロックの数を低減するために共通ブロックに関して実行され得る。

【００８８】

ガベージコレクションおよびブロック統合のプロセスは、ソースブロックが消去され得るために、ファイルーツーフラッシュマッピングアルゴリズムの要求に合わせて、有効なファイルデータをソースフラッシュブロックから１つ以上の宛先ブロックへ再配置した

。
【００８９】
保留中のガベージコレクション操作は直ちには実行されないが、段階的実行のためのスケジューリング・アルゴリズムに従って実行される。ガベージコレクションを必要とするオブジェクトのためのエントリが、デバイスの操作の間に３つのガベージコレクションキューにその時々に加えられる。別個のキューが、ファイル、古いブロック、および共通ブロックについて存在する。オブジェクトが、あらかじめ決められたプライオリティの順番で次のガベージコレクション操作のためのキューから選ばれる。そのキューが空であれば、ブロック統合は実行され得る。
【００９０】
ガベージコレクション操作は、２つのやり方でスケジュールを組むことができる。バックグラウンド操作は、ホストがデバイスへの読み出しまたは書き込みアクセスを行っていない場合に、ホストにより開始することができ、そのホストが別のアクセスを行うまで連続的にデバイスによって実行される。フォアグラウンド操作は、デバイスがホストによりアクセスされている間に、そのデバイスによりスケジュールを組むことができ、ホストから受信されたファイルデータのためのプログラム操作のバーストとインターリーブされたバーストにおいて実行される。インターリーブされたバーストの長さは、ガベージコレクションレートを要求される最小値に常に維持するために適応的に制御され得る。
【００９１】
８．２ガベージコレクションキュー
ガベージコレクションキューは、保留中のガベージコレクション操作がそのためにあるオブジェクトのためのエントリを含む。３つのキューは各々、古いブロック、共通ブロックおよびファイルのためのエントリをそれぞれ含む。２つの付加的なキューは、これら３つのキューよりも高いプライオリティを与えられ、古いブロックおよび共通ブロックそれぞれのためのエントリを含む。これら５つのガベージコレクションキューは、フラッシュメモリ中の制御ブロック中の制御ログに格納される。
【００９２】
８．２．１プライオリティの古いブロックキュー
このキューは、ホストからのerase コマンドの結果として完全に古いものにされたブロックのためのエントリを含む。これは、最も高いプライオリティのガベージコレクションキューである。このキューの中で識別されたすべてのブロックに関するガベージコレクション操作は、何か他のコマンドがホストから受け取られる前に、または何か他のキューからのオブジェクトに関してガベージコレクション操作が開始される前に、完了されなければならない。
【００９３】
８．２．２プライオリティの共通ブロックキュー
このキューは、ホストからのerase コマンドの結果として部分的に古くされた共通ブロックのためのエントリを含む。これは、２番目に高いプライオリティのガベージコレクションキューである。このキューの中で識別されたすべてのブロックに関するガベージコレクション操作は、何か他のコマンドがホストから受け取られる前に、またはより低いプライオリティのキューからのオブジェクトに関してガベージコレクション操作が開始される前に、完了されなければならない。
【００９４】
８．２．３古いブロックキュー
このキューは、ホストからのdeleteコマンド、またはファイルのデータの編集の結果として完全に古いものにされたブロックのためのエントリを含む。これは、３番目に高いプライオリティのガベージコレクションキューである。このキューの中で識別されたすべてのブロックに関するガベージコレクション操作は、より低いプライオリティのキューからのオブジェクトに関して操作が開始される前に、完了されるべきである。
【００９５】

８．２．４共通ブロックキュー
このキューは、ホストからのdeleteコマンド、またはファイルのデータの編集の結果として部分的に古くされた共通ブロックのためのエントリを含む。これは、４番目に高いプライオリティのガベージコレクションキューである。このキューの中で識別されたすべてのブロックに関するガベージコレクション操作は、より低いプライオリティのキューからのオブジェクトに関して操作が開始される前に、完了されるべきである。
【００９６】
８．２．５ファイルキュー
このキューは、ファイルのデータの編集の結果として古いデータを有するファイルのためのエントリを含む。これは、最も低いプライオリティのガベージコレクションキューである。ファイルがホストにより閉じられる場合、そのファイルがプレーンファイルでない限り、そのファイルのためのエントリがファイルキューに加えられる。従って、あるファイルがホストにより閉じられるときに、そのファイルがプレーンファイルであるのか、または編集済ファイル（プレーンまたは共通）であるのかを決定するために、そのファイルのためのＦＩＴエントリに関する分析を実行することが必要である。
【００９７】
この分析のための手順は以下の通りである。
１．関連のＦＩＴファイル中のＦＩＴエントリは、オフセットアドレス順に評価される。
２．データグループおよびデータグループヘッダの累積容量が決定される。
３．ファイルデータにより占められた物理容量が決定される。これは、そのファイルのためのデータを含むプログラムブロック以外のブロックの数に、プログラムブロック中の使用容量を加えたものである。
４．データグループ容量が物理容量のＸ％を超えれば、そのファイルは、プレーンファイルであると決定される。Ｘの値の例は、９８％である。Ｘ％は、バッファ消去操作の結果生じるプログラムされていない空間がプレーンファイル中に残ることを見越して、１００％未満である。
【００９８】
８．３ガベージコレクション操作
８．３．１古いブロック
古いブロックは、古いデータのみを含み、データを別のブロックへ再配置する必要なく消去できる。
【００９９】
８．３．２共通ブロック
共通ブロックは、ソースブロックであり、１つ以上のファイルのための１つ以上の部分的または完全に古いデータグループを含む。有効なデータは、このソースブロックから１つ以上の宛先ブロックに再配置されなければならない。
【０１００】
データは、完全ファイルグループの単位で再配置され、その場合、１つのファイルグループは、共通ブロック内の同じファイルのための１つ以上のデータグループを含む。各ファイルグループは、そのままの状態で宛先ブロックに再配置されるが、異なるファイルグループは異なるブロックに再配置され得る。
【０１０１】
共通ブロックは、宛先ブロックとして好ましい選択であり、適切な共通ブロックが全く利用できなければプログラムブロックが続き、適切なプログラムブロックが全く利用できなければ消去済ブロックが続く。
【０１０２】
ガベージコレクション操作は、以下の条件、すなわち、操作が完了するか、ホストがコマンドを送るか、またはフォアグラウンドモードにおいてインターリーブされたバーストの終わりに達するかの１つが発生するまで継続し得る。

共通ブロックのガベージコレクション操作についての流れ図が図８－５に示してある。

【０１０３】

８．３．３ファイルガベージコレクション

ファイルガベージコレクションは、ファイルのための古いデータにより占められた容量を回復するために実行される。ファイルガベージコレクションは、編集済プレーンファイル状態または編集済共通ファイル状態のファイルをプレーンファイル状態に復元する。第１のステップは、ガベージコレクションの間にデータグループがそこからコピーされなければならない古いファイルブロックおよび共通ブロックを識別するために、そのＦＩＴファイル中のＦＩＴエントリに関する分析を実行することである。この分析のための手順は以下の通りである。

１．関連のＦＩＴファイル中のＦＩＴエントリが、オフセットアドレス順に評価される。

２．データグループを物理ブロックと関係付けるために、データグループリストが構築される。プログラムブロック中のデータグループは、このリストから除外される。

３．リスト中で参照される各ブロック中のデータグループおよびデータグループヘッダにより占められる物理容量が決定される。

４．データグループ容量がブロックの容量のＸ％を超えれば、そのブロックは、ファイルブロックであると決定される。Ｘの値の例は、９８％である。Ｘ％は、バッファ消去操作の結果生じるプログラムされていない空間がファイルブロック中に残ることを見越して、１００％未満である。

５．ファイルブロックであると決定されるブロック中のデータグループが、前に構築されたデータグループリスト構成から取り去られる。

【０１０４】

改訂されたデータグループリスト中で参照されるデータグループは、古いファイルブロックまたは共通ブロック中に含まれ、ファイルガベージコレクション操作の間に、プログラムブロックにコピーされる。ファイルガベージコレクション操作の結果、ファイルのデータグループ構造は修正され得る。すなわち、再配置されたデータグループが、ブロック境界により２分されたり、または隣接するデータグループと併合されたりし得る。ファイルのためのプログラムブロックは、宛先ブロックとして用いられる。これが満たされると、別のプログラムブロックが開かれる。

【０１０５】

ガベージコレクション操作は、以下の条件、すなわち、操作が完了するか、ホストがコマンドを送るか、またはフォアグラウンドモードにおいてインターリーブされたバーストの終わりに達するかの１つが発生するまで継続し得る。

ファイルガベージコレクション操作についての流れ図が図８－４に示してある。

【０１０６】

８．３．４ブロック統合

ブロック統合は、不完全にプログラムされたプログラムブロックおよび共通ブロック中の消去済容量を回復するため、およびその容量を他のファイルのためのデータを格納するために利用可能にするため、実行される。

【０１０７】

統合のためのソースブロックが、最小限の可能なデータ再配置後にブロックが消去され得るようにするために、最も低いプログラムされた容量を有するプログラムブロックログまたは共通ブロックログ中のブロックとして選択される。データは、完全ファイルグループの単位で再配置され、その場合、１つのファイルグループは、プログラムブロックまたは共通ブロック内の同じファイルのための１つ以上のデータグループを含む。各ファイルグループは、そのままの状態で宛先ブロックに再配置されるが、異なるファイルグループは異なるブロックに再配置され得る。

【０１０８】

共通ブロックは、宛先ブロックとして好ましい選択であり、適切な共通ブロックが全く

利用できなければ、プログラムブロックが続く。まれに、宛先ブロックが全く利用できない場合、２つ以上の単一の宛先ブロックに再配置されるように、１つのファイルグループが分割され得る。

【０１０９】

ブロック統合操作は、以下の条件、すなわち、操作が完了するか、ホストがコマンドを送るか、またはフォアグラウンドモードにおいてインターリーブされたバーストの終わりに達するかの１つが発生するまで継続し得る。

ブロック統合操作についての流れ図が図８－６に示してある。

【０１１０】

８．４ガベージコレクション操作のスケジューリング

ガベージコレクションは、好ましくは、ホストデバイスがカードにアクセスしている期間の間にバックグラウンドタスクとして実行される。ホストにより開始されたバックグラウンドガベージコレクションは、直接データファイル・プラットフォームにおいてサポートされる。

【０１１１】

しかし、ホストがデータをデバイスに書き込んでいる間にフォアグラウンドタスクとしてガベージコレクションを実行することが必要なこともあり得る。このモードでは、完全なガベージコレクション操作は、単一の事象として完了される必要はない。ガベージコレクションのバーストは、ガベージコレクション操作が複数の別個の段階で完了され、ホストへのデバイスの利用可能性の中断が限定されるように、ホストからのデータのプログラミングのバーストとインターリーブされ得る。

【０１１２】

８．４．１バックグラウンド操作

バックグラウンドガベージコレクションは、ホストがidleコマンドをデバイスに送るときに開始される。これは、ホストが電力をデバイスから故意に除去せず、デバイスに直ちにアクセスすることを意図しないことを示す。しかし、ホストは、別のコマンドを伝送することにより、アイドル状態をいつでも終了し得る。

【０１１３】

アイドル状態において、デバイスは、以下の条件、すなわち、ホストが別のコマンドを送る、または、すべてのガベージコレクションキューが空でありどのようなブロック統合操作も可能ではない、の１つが発生するまで連続的なガベージコレクション操作を実行する。

【０１１４】

８．４．２インターリーブされた操作

インターリーブされたガベージコレクション操作は、ファイルデータをプログラムするための直接データファイルプロセスによって開始される。インターリーブされた操作は、図８－１に例示されている。ホストインターフェイスがアクティブでありかつＮ１のプログラム操作がフラッシュメモリに対してなされるホストデータ書き込み段階の後、デバイスは、ガベージコレクション段階に切り替わる。この段階において、フラッシュメモリに対するＮ２のプログラム操作が完了されるまで、１つ以上のガベージコレクション操作の一部が実行される。

【０１１５】

進行中のガベージコレクション操作は、ガベージコレクション段階の最後に中断され、次のそのような段階において再開され得る。Ｎ１およびＮ２の値は、適応スケジューリング・アルゴリズムにより決定される。

【０１１６】

８．４．３適応スケジューリング

適応スケジューリング方法は、ホストデータプログラミングおよびガベージコレクションのインターリーブされたバーストの相対長さを制御するために用いられ、その結果、ガベージコレクション操作によるホストデータ書き込み操作の中断は最小限に維持され得る

。これは、その後の性能低下を引き起こし得る保留中のガベージコレクションの未処理分が蓄積されないことをさらに保証することによっても達成される。

【０１１７】

８．４．３．１操作の原理

いつでも、デバイス状態は、前に書き込まれたホストデータにより占められる容量、消去済ブロックプール中のブロックの中の消去済容量、およびガベージコレクション操作により、さらなるホストデータを書き込むために利用可能にされ得る回復可能容量を含む。この回復可能容量は、プログラムブロック、共通ブロック、または前に書き込まれたホストデータと共有される古いファイルブロック中、あるいは完全に古いブロック中にあり得る。これらのタイプの容量利用が、図８－２に示してある。

【０１１８】

ガベージコレクションの適応スケジューリングは、増分ホストデータのプログラミングと前に書き込まれたホストデータの再配置とのインターリーブ比率を制御し、その比率が、すべての回復可能容量がホストデータ用に利用可能にされ得る適応期間にわたって一定なままであり得るようにする。ホストがファイルを削除し、これが、前に書き込まれたホストデータが回復可能容量に変換されれば、インターリーブ比率はそれに応じて変更され、新しい適応期間が開始される。

【０１１９】

最適なインターリーブ比率は、以下のように決定できる。

消去済ブロックプール中の消去済ブロックの数＝erased_blocks、

プログラムブロックおよび共通ブロックを合わせた数＝data_blocks、

プログラムブロックおよび共通ブロック中の有効なデータページの総数＝data_pages、

古いブロックの数＝obsolete_blocks、

ブロック中のページ数＝pages_per_block であるならば、

ガベージコレクションにより作成され得る消去済ブロックの数は、data_blocks-(data_pages/pages_per_block)により与えられ、

ガベージコレクション後の消去済ブロックの総数は、erased_blocks+obsolete_blocks+data_blocks-(data_pages/pages_per_block)により与えられ、

書き込まれ得る増分データメタページの数は、pages_per_block*(erased_blocks+obsolete_blocks+data_blocks)-data_pageにより与えられる。

有効なデータが、プログラムブロックおよび共通ブロック全体にわたって均等に分布される（データページカウントが低いブロックは、ブロック統合操作のためのソースブロックとして選択されるので、悲観的な仮定である）と仮定すれば、

ガベージコレクションの間に再配置されるメタページの数は、data_pages*(data_blocks-data_pages/pages_per_block)/data_blocksにより与えられる。

最適なインターリーブ比率Ｎ１：Ｎ２は、書き込まれ得る増分データメタページの数とガベージコレクションの間に再配置されなければならないメタページの数との比率である。従って、

N1:N2=(pages_per_block*(erased_blocks+obsolete_blocks+data_blocks)-data_pages)/(data_pages*(data_blocks-data_pages/pages_per_block)/data_blocks) となる。

【０１２０】

古いファイルブロック中の古い容量の回復が適応スケジューリング・アルゴリズムに含まれていないことに留意されたい。そのような容量は、ファイルの編集のみに起因し、共通した発生ではない。かなりの容量が古いファイルブロック中に存在すれば、適応的に決定されたインターリーブ比率は最適条件ではないことがあるが、最小比率を有する操作（８．４．３．２において説明される）に切り替えることにより、そのようなブロックの効率的なガベージコレクションが保証される。

【０１２１】

８．４．３．２インターリーブ制御

インターリーブ比率Ｎ１：Ｎ２は、以下のように３つのバンドにおいて定義される。
１）最大：ガベージコレクションが完全に阻止されることが決してないように保証するため、インターリーブ比率の最大限度が設定される。この最大限度の例は、１０対１である。
２）適応：適応バンドにおいて、インターリーブ比率は、共通ブロックおよび古いブロックの保留中のガベージコレクション、ならびにプログラムブロックと共通ブロックとの統合にとって最適であるように制御される。インターリーブ比率は、
N1:N2=(pages_per_block*(erased_blocks+obsolete_blocks+data_blocks)-data_pages)/(data_pages*(data_blocks-data_pages/pages_per_block)/data_blocks) の関係から定義される。
ここで、Ｎ１およびＮ２は、ページプログラム操作の数として定義される。ガベージコレクションのバーストの好ましい期間を表すＮ２の値が定義される。この値の例は、１６である。
３）最小：消去済ブロックプール中のブロックの数が、定義された最小値を下回れば、インターリーブ比率は、適応的に定義されないが、固定された最小限度に設定される。この最小限度の例は、１対１０である。

【０１２２】

８．４．３．３制御パラメータ

以下のパラメータは、適応スケジューリングの制御のために、フラッシュメモリ中の制御ブロック中の制御ログ中に維持される。
消去済ブロックカウント：消去済ブロックプール中のブロック数のカウントが維持される。消去済ブロックプールにブロックが付加され、かつこれから除去されるときに、このカウントが更新される。
プログラムおよび共通ブロックカウント：プログラムブロックおよび共通ブロックを合わせた数のカウントが維持される。共通ブロックは、古いデータを含み得る。プログラムブロックログおよび共通ブロックログにブロックが付加され、かつこれらから除去されるときに、このカウントが更新される。
プログラムおよび共通ブロックページカウント：プログラムブロックおよび共通ブロック中の有効なデータページの数のカウントが維持される。プログラムブロックログおよび共通ブロックログにブロックが付加され、かつこれらから除去されるときに、このカウントが更新される。
古いブロックカウント：ガベージコレクションを待っている完全に古くなったブロックの数のカウントが維持される。古いブロックのガベージコレクションキューにブロックが付加され、かつこれから除去されるときに、このカウントが更新される。

【０１２３】

８．５ガベージコレクションのための流れ図

いくつかの特定のガベージコレクション操作の１つを選ぶための特定のアルゴリズムの流れ図が図８－３に示してある。図８－４は、図８－３の「ファイルガベージコレクション」ブロックについての流れ図である。図８－３の「共通ブロックのガベージコレクション」ブロックについての流れ図が、図８－５の主題である。図８－３の「ブロック統合」機能は、図８－６の流れ図により示される。

【０１２４】

図８－７Ａ～図８－７Ｄの４つの図は、図８－５のプロセスに起因し得る共通ブロックのガベージコレクションの例を示している。図８－７Ａが初期条件を示しているのに対し、図８－７Ｂ～図８－７Ｃは、ガベージコレクションプロセスにおける３つのステップを例示している。矢印は、古いブロックから一杯ではないファイルブロックへの有効なデータの転送を示し、これらの宛先ファイルブロックは共通ブロックになる。

【０１２５】

９．データバッファリングおよびプログラミング

このセクションにおいて説明されるデータバッファリングおよびプログラミング方法は

、現行製品において用いられる同じフラッシュインターフェイスおよび誤り訂正符号（ＥＣＣ）構造の使用を余儀なくされる。新しいフラッシュインターフェイスおよびＥＣＣ構造が導入されれば、代わりの最適化された方法が将来採用される。

【０１２６】

９．１データバッファ

フラッシュメモリにプログラムし、あるいはフラッシュメモリから読み出されるデータの一時的格納のために、コントローラ（先行出願のＲＡＭ３１）中のＳＲＡＭ中にバッファメモリが存在する。バッファメモリの割り当てられた領域は、完全なメタページが単一操作でフラッシュメモリにプログラムされるようにするために、ファイルについての十分なデータを蓄積するために使用される。バッファメモリ中のファイルについてのデータのオフセットアドレスは重要ではない。バッファメモリは、複数のファイルについてのデータを格納し得る。

【０１２７】

ホストとの書き込み操作および読み出し操作双方のパイプライニングを可能にするために、２つのメタページの容量を有するバッファメモリ空間が、バッファされる各ファイルについて利用可能であるべきである。バッファメモリは１組のセクタバッファを含む。個別のセクタバッファは、単一のファイルのためのデータの一時的格納のために割り当てられ、データがその最終的な宛先に転送されたときに割り当て解除され得る。セクタバッファは、０～Ｎ－１のセクタバッファ番号により識別される。セクタバッファの数（Ｎ）の例は、６４である。

【０１２８】

利用可能なセクタバッファは、それらのセクタバッファ番号順に周期的に割り当てられる。各セクタバッファは、１つのファイルラベルと、その中に含まれるデータの開始および終わりを定義する２つの関連したポインタとを有する。セクタバッファ中のデータ内のファイルのオフセットアドレス範囲も記録される。セクタバッファおよびこれらと関連した制御情報双方は、コントローラ中の揮発性メモリの内部にのみ存在する。

【０１２９】

９．２データプログラミング

９．２．１メタページ

メタページは、フラッシュメモリにおけるプログラミングの最大単位である。データは、最大性能のために、可能な限りいかなる場合でもメタページ単位でプログラムされるべきである。

【０１３０】

９．２．２ページ

ページは、メタページのサブセットであり、フラッシュメモリにおけるプログラミングの最小単位である。

【０１３１】

９．２．３セクタ

セクタは、ページのサブセットであり、コントローラとフラッシュメモリとの間のデータ転送の最小単位である。１つのセクタは通常、５１２バイトのファイルデータを含む。ＥＣＣは、各セクタについてコントローラにより生成されて（先行出願の図２のコントローラＥＣＣ回路３３によるように）、セクタの終わりに付加されたフラッシュに転送される。データがフラッシュから読み出されるとき、そのデータは、ＥＣＣがチェックされるようにするために、多数の完全なセクタでコントローラに転送されなければならない。

【０１３２】

９．３バッファ消去

ファイルのためのデータは通常、フラッシュメモリにおいて完全なメタページをプログラムするために十分なデータが利用可能になるまで、セクタバッファ中に蓄積される。ホストが、あるファイルのためのデータのストリーミングを停止すると、１つ以上のセクタバッファが、１つのメタページの一部のためのファイルデータと共にとどまる。このデー

タは、ホストがそのファイルのためのさらなるデータを書き込めるようにするために、バッファメモリ中にとどまる。しかし、一定の状況の下では、バッファメモリ中のデータは、バッファ消去として知られる操作においてフラッシュメモリに投入されなければならない。バッファ消去操作は、セクタバッファ中に保持されるファイルのためのすべてのデータが、メタページ内の１つ以上のペー中でプログラムされるようにする。

【０１３３】

バッファ消去操作は、以下の２つの事象において実行される。

１）ファイルがホストにより閉じられるか、または

２）shutdownコマンドがホストにより受信される。

【０１３４】

ホストにより閉じられるファイルのためのデータが、スワップブロックへスワップアウトされていれば、そのデータは、バッファメモリに返されるべきであり、バッファ消去が実行されるべきである。電力の除去に続くデバイスの初期化の間スワップブロック中にあるデータは、バッファメモリに返されるべきであり、バッファ消去が実行されるべきである。

【０１３５】

９．４バッファスワップ

バッファスワップは、１つ以上のセクタバッファ中のファイルのためのデータが、スワップブロックとして知られる一時的位置でプログラムされ、ホストがそのファイルのためのデータを書き込み続けるときに、後にバッファメモリに返される、操作である。

【０１３６】

９．４．１スワップブロックの形式

スワップブロックは、セクタバッファからスワップアウトされたファイルのためのデータを格納する専用ブロックである。ファイルのためのデータは、スワップブロック中のそのファイル専用の１つ以上のページ中で隣接して格納される。データがその後バッファメモリへスワップインにより戻される場合、そのデータは、そのスワップブロック中で古くなる。

【０１３７】

スワップブロックが一杯になると、その内部の有効なデータは、圧縮形態で消去済ブロックに書き込まれ、このブロックがスワップブロックになる。この圧縮形態は、異なるファイルのためのデータが同じページ内に存在できるようにする。好ましくは、単一のスワップブロックのみが存在する。

【０１３８】

９．４．２スワップブロック中に格納されるデータのインデックス化

スワップブロック中の各ファイルについて、バッファメモリ中のファイルについて前に記録された情報のコピーを含むスワップブロックインデックスがフラッシュメモリ中で維持される（９．１参照）。

【０１３９】

９．４．３スワップブロックへのデータの移動（スワップアウト）

ホストにより開かれたファイル、またはホストからのファイルについてのwriteコマンドの結果としてスワップブロックからスワップインされなければならいファイルに割り当てられる利用可能なセクタバッファが不十分な場合、スワップアウト操作が生じる。スワップアウト用に選択されるファイルは、バッファメモリ中にそのためのバッファが存在するファイルのうちの最近書き込まれていないファイルであるべきである。

【０１４０】

必要に応じ、予定されていない電力の除去が万一発生した場合のデータの保全性を向上するため、ホストからの最近のwrite コマンドに関連しないバッファメモリ中の任意のファイルについてスワップアウトが実行され得る。この場合、ファイルのためのデータは、バッファメモリ中に残ることができ、電力の除去がなかったのであれば、その後のスワップイン操作は必要とされない。

【０１４１】
９．４．４スワップブロックからのデータの復元（スワップイン）
ファイルのための完全なデータは、スワップブロックから１つ以上のセクタバッファに読み出される。ファイルデータは、そのスワップアウト前と全く同じ整合をセクタバッファと有する必要がない。整合は、スワップブロックの圧縮の結果、変更されている可能性がある。スワップブロック中のファイルのためのデータは、古くなる。
【０１４２】
９．５ホストからのファイルデータのプログラミング
このセクションで示される例は、１メタページあたり２ページ、および１ページあたり２セクタを有するフラッシュメモリ構成に関する。
【０１４３】
９．５．１ホストからの連続データのプログラミング
ファイルのためのデータは、ホストからストリームされ、連続して割り当てられたセクタバッファ中に蓄積される。十分なセクタバッファが満たされると、それらのセクタバッファのデータは、各セクタについての１ＥＣＣと共にフラッシュメモリに転送され、ファイルのためのプログラムブロック中の宛先メタページがプログラムされる。連続的なホストデータプログラミングの例が、図９－１に示してある。
【０１４４】
９．５．２ホストからの中断データのプログラミング
ファイルのためのデータは、ホストからストリームされ、連続して割り当てられたセクタバッファ中に蓄積される。図９－２は、中断されたホストデータのプログラミングの例を示す。ストリームは、データセグメント２Ａの後に中断されると同時に、異なるファイルのための書き込み操作が実行される。ファイルのためにさらにwrite コマンドが受信されると、ホストからストリームされたデータが、前と同じセクタバッファ中に蓄積され、データセグメント２Ｂにおいて開始する。十分なセクタバッファが満たされると、それらのセクタバッファのデータは、各セクタについてのＥＣＣと共にフラッシュメモリに転送され、ファイルのためのプログラムブロック中の宛先メタページがプログラムされる。
【０１４５】
９．５．３バッファから消去されるデータのプログラミング
ファイルのためのデータは、ホストからストリームされ、連続して割り当てられたセクタバッファ中に蓄積される。しかし、完全なメタページにおいてプログラムされる不十分なデータが存在する。１つの例が図９－３に示してある。データセグメント１および２Ａは、パディングセグメント２Ｂと共に、フラッシュメモリへ各セクタについてのＥＣＣと共に転送され、ファイルのためのプログラムブロック中の宛先ページがプログラムされる。
【０１４６】
９．５．４バッファからスワップアウトされるデータのプログラミング
この操作は、宛先ページが、ファイルのためのプログラムブロック中ではなく、スワップブロック中で次に利用可能なものであること以外は、バッファ消去プログラミングのための操作と同一である。図９－４は、これを例示している。
【０１４７】
９．５．５バッファからの消去に続くホストからのデータのプログラミング
ファイルのためのバッファ消去操作の後にホストにより供給されたファイルデータは、バッファメモリから消去済データとは別個にプログラムされる。従って、プログラミングは、ファイルのためのプログラムブロック中の次に利用可能なページにおいて開始しなければならない。現在のメタページを完成するために十分なデータが蓄積され、そのデータは、図９－５のセクタ３および４について示されるようにＥＣＣと共に転送されプログラムされる。
【０１４８】
９．５．６バッファへのスワップインに続くホストからのデータのプログラミング

バッファメモリからスワップアウトされたファイルのためのさらなるデータが受信されると、そのデータは、バッファメモリ中に割り当てられたセクタバッファ中に蓄積される。スワップアウトされたデータも、スワップブロックからバッファメモリに返される。十分なデータが蓄積されると、完全なメタページが単一操作でプログラムされる。

【０１４９】

図９－６に例示されるように、データセグメント１および２Ａは、パディングセグメント２Ｂと共に、スワップブロックから読み出され、２つのセクタバッファ中で復元される。ＥＣＣは、両方のセクタでチェックされる。

図９－７の例に示されるように、ホストからのファイルデータは、バッファセクタ中で蓄積される。データセグメント１、２Ａ／２Ｂ、３Ａ／３Ｂおよび４Ａ／４Ｂは、各セクタのためのＥＣＣと共にフラッシュメモリに転送され、セクタ１、２、３および４としてプログラムされる。

【０１５０】

９．６フラッシュからコピーされたデータのプログラミング

９．６．１整合されたメタページからのデータのコピー

図９－８に例示されるように、一杯の宛先メタページにコピーされるべきデータが単一の一杯のソースメタページを占める場合、ソースおよび宛先メタページは、整合されていると言われる。データセクタ１、２、３および４は、ソースメタページから４つのセクタバッファに読み出され、ＥＣＣは、各セクタでチェックされる。

【０１５１】

９．６．１．１バッファからの書き戻し

図９－９に示されるように、データは、４つのセクタバッファから宛先メタページにプログラムされる。ＥＣＣが生成され、セクタ１、２、３および４について格納される。

【０１５２】

９．６．１．２オンチップコピー

コピーされるべきデータのメタページ整合がソースおよび宛先メタページにおいて同じ場合、フラッシュチップ内のオンチップコピーは、コピー操作の速度を増大させるために用いられ得る。ＥＣＣチェックがエラーを全く示さなければ、データは、宛先メタページにプログラムされる。

【０１５３】

９．６．１．３共通ブロック内のメタページ整合

共通ブロック内の各ファイルグループの開始は、メタページの開始と整合することを強制されるべきである。プログラムブロック中のデータグループも、ブロック中の第１のメタページの開始と整合する。従って、共通ブロックへのプログラムブロックの統合および１つの共通ブロックからプログラムブロックまたは別の共通ブロックへのファイルグループのコピーのような、共通ブロックについてのすべてのデータコピー操作は、整合されたメタページ間のデータコピーで動作する。フラッシュチップ内のオンチップコピーは、共通ブロックへまたは共通ブロックからデータをコピーする場合にコピー操作の速度を増大させるために用いられるべきである。

【０１５４】

９．６．２不整合な連続したメタページからのデータコピー

完全な宛先メタページにコピーされるべきデータが隣接しているが、２つの連続したソースメタページを占める場合、ソースおよび宛先メタページは、不整合であると言われる。ソースメタページの読み出しの例が、図９～１０に示してある。データセクタ１Ａ／１Ｂ、２、および３は、第１のソースメタページから３つのセクタバッファに読み出され、データセクタ４および５Ａ／５Ｂは、第２のソースメタページからさらなる２つのセクタバッファに読み出される。ＥＣＣは、各セクタでチェックされる。

【０１５５】

データ部分１Ａ／１Ｂ、２Ａ／２Ｂ、３Ａ／３Ｂ、および４Ａ／４Ｂは、図９～１１に示されるように、セクタバッファから、宛先メタページ中のセクタ１、２、３および４に

プログラムされる。ＥＣＣが生成され、セクタ１、２、３および４のために格納される。
【０１５６】
コピーされているメタページデータが連続データのより大きいランの一部である場合、コピーは部分的にパイプラインされ得る。データは、ソース位置から十分なメタページ中のバッファメモリに読み出される。Ｎ＋１のソースメタページは、Ｎの宛先メタページをプログラムするために読み出されなければならない。
【０１５７】
９．６．３不整合かつ非連続なメタページからのデータのコピー
完全な宛先メタページにコピーされるべきデータが隣接しておらず、かつ２つ以上の非連続なソースメタページを占める場合、ソースおよび宛先メタページは、不整合かつ非連続と言われる。この場合は、ファイル内の２つ以上の隣接していないデータグループの一部を単一の宛先メタページへコピーすることを表す。図９～１２に示されるように、データセクタ１Ａ／１Ｂ、２、および３Ａ／３Ｂは、第１のソースメタページから３つのセクタバッファに読み出され、データセクタ４Ａ／４Ｂおよび５Ａ／５Ｂは、第２のソースメタページからさらなる２つのセクタバッファに読み出される。ＥＣＣは、各セクタでチェックされる。
【０１５８】
次にデータ部分１Ａ／１Ｂ、２Ａ／２Ｂ、３Ａ／３Ｂ、および４Ａ／４Ｂは、図９～１３に示されるように、セクタバッファから宛先メタページ中のセクタ１、２、３および４にプログラムされる。ＥＣＣが生成され、セクタ１、２、３および４のために格納される。
【０１５９】
１０．ファイルインデックス化
１０．１ファイルインデックス化の原理
図１０－１にファイルインデックス化が一般的に示してある。ファイルのためのデータは、データグループのセットとして格納され、各データグループは、ファイルオフセットアドレス空間および物理アドレス空間双方の中の隣接するアドレスのランにわたっている。ファイルのためのセット内のデータグループは、互いにどのような特定の物理アドレス関係も有する必要がない。ファイルインデックステーブル（ＦＩＴ）により、ファイルのための有効なデータグループの位置を、オフセットアドレス順で識別することができる。ファイルのためのＦＩＴエントリのセットが、ファイルデータポインタにより識別される。
【０１６０】
ホストにより生成されるファイルに関連した情報は、情報テーブル（ＩＴ）中でfile_infoとして格納される。file_infoの性質および内容はホストにより決定され、その性質および内容は、デバイスにより解釈されない。file_infoは、ファイル名、親ディレクトリ、子ディレクトリ、属性、権利情報、およびファイルについてのファイル関連付けを含み得る。ＩＴ中のファイルのためのfile_infoは、ファイル情報ポインタにより識別される。
【０１６１】
ディレクトリは、デバイス中のすべての有効なファイルのためのファイルデータポインタおよびファイル情報ポインタを含む。ファイルのためのこれらのディレクトリエントリは、数値であるfileIDにより識別さる。
【０１６２】
１０．２ファイルインデックス化構造
図１０－２は、ファイルインデックス化構造の例を示す。
【０１６３】
１０．３ディレクトリ
１０．３．１FileID
fileIDは、直接データファイル・プラットフォーム中に存在するファイルのための数値

識別子である。これは、createコマンドに応答して直接データファイル・プラットフォームにより割り当てられるか、またはcreateコマンドを有するパラメータとして指定され得る。

fileID値がデバイスにより割り当てられると、ディレクトリ中のエントリへの周期的ポインタが、次の利用可能なfileIDの位置を特定するために用いられる。ファイルが削除または消去されると、ファイルのfileIDにより識別されたディレクトリエントリは、利用可能としてマークされる。

fileID値がディレクトリ中のエントリを定義し、このエントリは、ファイルのためのファイルデータポインタおよびファイル情報ポインタのためのフィールドを含む。

デバイス中に格納され得るファイルの最大数は、fileIDに割り当てられるビット数により決定される。

【０１６４】

１０．３．２ファイルデータポインタ

ファイルデータポインタは、ＦＩＴブロックリスト中のファイルのためのエントリへの論理ポインタであり、あるいは制御ログ内のＦＩＴ更新ブロックリストでもある。

ファイルデータポインタは、以下の２つのフィールドを有する。

１）ＦＩＴ範囲

２）ＦＩＴファイル番号

ファイルが０の長さを有するときでさえ、ファイルのためのファイルデータポインタが存在する。

【０１６５】

１０．３．２．１ＦＩＴ範囲

ＦＩＴ範囲は、ＦＩＴのサブセットである。各ＦＩＴ範囲は、別個の物理ＦＩＴブロックにマッピングされる。１つのＦＩＴ範囲は、１つのＦＩＴファイルと、例えば、５１２であり得る最大数のＦＩＴファイルとの間を含むことができる。

【０１６６】

１０．３．２．２ＦＩＴファイル番号

ＦＩＴファイル番号は、ＦＩＴ内のＦＩＴファイルを識別するために用いられる論理番号である。

【０１６７】

１０．３．３ファイル情報ポインタ

ファイル情報ポインタは、情報ブロックリスト中のファイルのためのエントリへの論理ポインタであり、あるいは、また制御ログ内の情報更新ブロックリストでもある。

ファイル情報ポインタは、以下の２つのフィールドを有する。

１）情報範囲

２）情報番号

【０１６８】

１０．３．３．１情報範囲

情報範囲は、情報テーブルのサブセットである。各情報範囲は、別個の物理情報ブロックにマッピングされる。１つの情報範囲は、１つのfile_infoのセットと、例えば、５１２であり得る最大数のfile_infoのセットとの間を含むことができる。

【０１６９】

１０．３．３．２情報番号

情報番号は、情報ブロック内のfile_infoのセットを識別するために用いられる論理番号である。

【０１７０】

１０．３．４ディレクトリ構造

ディレクトリは、目的専用のフラッシュブロック中に格納される。図１０－３は、ディレクトリブロック形式の例を示す。ディレクトリは、ページのセットとして構造化され、各ページ内には、連続したfileID値を有するファイルのためのエントリのセットが存在す

る。このエントリのセットは、ディレクトリ範囲と呼ばれる。

【０１７１】

ディレクトリは、制御ポインタにより定義された次の消去済ページ位置にディレクトリページの改訂バージョンを書き込むことにより更新される。必要であれば、メタページ中の異なるページにそれらをプログラムすることによって、複数のページが同時に更新され得る。

ディレクトリ範囲のための現在のページ位置は、ディレクトリブロック中の最後に書き込まれたページ中の範囲ポインタにより識別される。

【０１７２】

１０．４ブロックリスト

ファイルインデックステーブルおよび情報テーブルは両方とも、一連の論理範囲を含み、１つの範囲は、物理フラッシュブロックとの相関関係を有する。ブロックリストは、ファイルデータポインタまたはファイル情報ポインタ中で定義された範囲と物理ブロックとの間、ならびにファイルデータポインタまたはファイル情報ポインタ中で定義された論理数とファイルインデックステーブルおよび情報テーブル内の物理ブロックにおいて用いられる論理数との間の相関関係を記録するために、制御ログ中に維持される。

【０１７３】

１０．４．１ＦＩＴブロックリスト

ＦＩＴブロックリストは、ファイルのためのＦＩＴ中のエントリにＦＩＴファイルポインタを割り当てる制御ログ中のリストである。ＦＩＴファイルポインタは、ファイルデータポインタ中で定義された範囲に割り当てられる物理フラッシュブロックのアドレス、およびファイルデータポインタ中で定義される同じＦＩＴファイル番号を含む。ＦＩＴブロックリスト中のエントリは、単一のフィールド、ブロック物理アドレスを含む。

【０１７４】

ＦＩＴ更新ブロックリストは、更新されるＦＩＴ中のファイルのためのエントリにＦＩＴファイルポインタを割り当てる制御ログ中のリストである。ＦＩＴファイルポインタは、ＦＩＴ更新ブロックエントリとして現在割り当てられる物理フラッシュブロックのアドレス、およびＦＩＴ更新ブロックにおいて更新されるＦＩＴファイルＦＩＴに割り当てられるＦＩＴ更新ファイル番号を含む。ＦＩＴ更新ブロックリスト中のエントリは、以下の３つのフィールドを含む。

１）ＦＩＴ範囲

２）ＦＩＴファイル番号

３）ＦＩＴ更新ファイル番号

【０１７５】

ファイルデータポインタに対応するＦＩＴファイルポインタがＦＩＴブロックリストから決定されるべきである場合、ファイルデータポインタに関するエントリが存在するかどうかを決定するためにＦＩＴ更新ブロックリストが検索される。なにも存在しなければ、ＦＩＴブロックリスト中のファイルデータポインタに関するエントリが有効である。

【０１７６】

１０．４．２情報ブロックリスト

ホストにより書き込まれたfile_infoは、ファイル情報ポインタにより識別された情報テーブル中に直接格納される。情報ブロックリストは、情報ポインタを情報テーブル中のfile_infoに割り当てるために存在する。これらの情報ブロックリストのためのインデックス化機構は、ＦＩＴブロックリストについて記載された機構と完全に類似する。

情報ブロックリスト中のエントリは、単一のフィールド、ブロック物理アドレスを含む。

情報更新ブロックリスト中のエントリは、以下の３つのフィールドを含む。

１）情報範囲

２）情報番号

３）更新情報数

【０１７７】
１０．５ファイルインデックステーブル
ファイルインデックステーブル（ＦＩＴ）は、ＦＩＴエントリのストリングを含み、各ＦＩＴエントリは、データグループのフラッシュメモリ中のファイルオフセットアドレスおよび物理位置を識別する。ＦＩＴは、デバイス中に格納されたファイルのためのすべての有効なデータグループのためのエントリを含む。古いデータグループは、ＦＩＴによりインデックスされない。ＦＩＴ論理的構造の例が、図１０－４に示してある。
【０１７８】
ファイル中のデータグループのためのＦＩＴエントリのセットが、連続したエントリとして、ファイルオフセットアドレス順に維持される。エントリのセットは、ＦＩＴファイルとして知られる。ＦＩＴは、一連のＦＩＴ範囲として維持され、各ＦＩＴ範囲は、物理フラッシュブロックとの相関関係を有する。ＦＩＴ範囲の数は、デバイス中のデータグループの数に応じて変わる。新しいＦＩＴ範囲が作成され、ＦＩＴ範囲は、デバイスの操作の間に除去される。ＦＩＴブロックリストは、ファイルデータポインタからＦＩＴファイルポインタを作成するために用いられ、それによってＦＩＴ中の位置が識別され得る。
【０１７９】
１０．５．１ＦＩＴファイル
ＦＩＴファイルは、ファイル内のデータグループのための隣接するＦＩＴエントリのセットである。このセット中のエントリは、ファイルオフセットアドレス順である。ＦＩＴファイル中のＦＩＴエントリは連続し、単一のＦＩＴ範囲内に含まれるか、または１つのＦＩＴ範囲から次の連続したＦＩＴ範囲にオーバーフローする。
【０１８０】
１０．５．２ＦＩＴヘッダ
ＦＩＴファイル中の第１のエントリは、ＦＩＴヘッダである。ＦＩＴヘッダは、以下の３つのフィールドを有する。
１）fileID
２）プログラムブロック
３）プログラムポインタ
ＦＩＴヘッダは、ＦＩＴエントリの整数に等しい固定長さを有する。この数は、１であり得る。
【０１８１】
１０．５．２．１FileID
fileIDは、ディレクトリ中のファイルのためのエントリを識別する。
【０１８２】
１０．５．２．２プログラムブロック
ＦＩＴファイルの更新されたバージョンがＦＩＴに書き込まれるときはいつも、ファイルのためのプログラムブロックの現在の物理位置がＦＩＴヘッダに記録される。ファイルがホストにより再び開かれるときに、この物理位置は、ファイルのためのプログラムブロックの位置を特定するために用いられる。この物理位置は、ＦＩＴファイルと、プログラムブロック統合のために選択されたファイルのためのプログラムブロックとの間の対応関係を有効化するためにも用い得る。
【０１８３】
１０．５．２．３プログラムポインタ
ＦＩＴファイルの更新されたバージョンがＦＩＴに書き込まれるときはいつも、ファイルのためのプログラムブロック内のプログラムポインタの現在値がＦＩＴヘッダに記録される。この現在値は、ファイルがホストにより再び開かれるとき、またはプログラムブロック統合のためにプログラムブロックが選択されるときに、ファイルのためのプログラムブロック内のデータをプログラムするための位置を定義するために使用される。
【０１８４】
１０．５．３ＦＩＴエントリ

ＦＩＴエントリは、データグループを指定する。ＦＩＴエントリは、以下の４つのフィールドを有する。
１）オフセットアドレス
２）長さ
３）ポインタ
４）ＥＯＦフラグ
【０１８５】
１０．５．３．１オフセットアドレス
オフセットアドレスは、データグループの最初のバイトに関するファイル内のバイトで表したオフセットである。
【０１８６】
１０．５．３．２長さ
これは、データグループ内のファイルデータの長さをバイトで定義する。完全なデータグループの長さは、この値よりデータグループヘッダの長さだけ長い。
【０１８７】
１０．５．３．３ポインタ
これは、データグループの開始のフラッシュブロック中の位置へのポインタである。ポインタは、以下の２つのフィールドを有する。
１）データグループを含む物理ブロックを定義する、ブロックアドレス
２）データグループの開始のブロック内のバイトオフセットを定義する、バイトアドレス（このアドレスは、データグループヘッダを含む。）
【０１８８】
１０．５．３．４ＥＯＦフラグ
ＥＯＦフラグは、あるデータグループを、ファイルの終わりであると識別する単一のビットである。
【０１８９】
１０．５．４ＦＩＴブロック形式
ＦＩＴ範囲は、ＦＩＴブロックとして知られる単一の物理ブロックにマッピングされる。これらのブロック中のデータの更新バージョンは、ＦＩＴ更新ブロックとして知られる、共通更新ブロック中にプログラムされる。データは、１ページ単位で更新される。メタページ内の複数のページは、必要であれば、並列に更新され得る。
【０１９０】
１０．５．４．１間接アドレス指定
ＦＩＴファイルは、ＦＩＴファイルポインタにより識別される。このポインタ内のＦＩＴファイル番号フィールドは、論理ポインタであり、これは、インデックス化に用いられる物理構造内でＦＩＴファイルのためのデータが移動される際に一定のままである。物理ページ構造内のポインタフィールドは、論理ポインタから物理ポインタへの変換を提供する。
【０１９１】
１０．５．４．２ページ形式
ＦＩＴブロックおよびＦＩＴ更新ブロックにおいて用いられるページ形式は、同一である。
１つのページが、ＦＩＴエントリのための第１の領域およびファイルポインタのための第２の領域の２つの領域にさらに分割される。図１０－５に、例が示してある。
第１の領域は、各々がデータグループを指定するか、またはＦＩＴファイルのためのＦＩＴヘッダを含むＦＩＴエントリを含む。１つのＦＩＴページ中のＦＩＴエントリ数の例は、５１２である。１つのＦＩＴページまたは重なっている２つ以上のＦＩＴページ内で、ＦＩＴエントリの隣接するセットによりＦＩＴファイルが指定される。ＦＩＴヘッダを含む、ＦＩＴファイルの第１のエントリは、第２の領域中のファイルポインタにより識別される。

【０１９２】
第２の領域は、最近プログラムされたＦＩＴページ中のみの有効なファイルポインタを含む。すべての他のページ中の第２の領域は古く、用いられない。ファイルポインタ領域は、ＦＩＴブロックに含まれ得る各ＦＩＴファイルのための１つのエントリを含み、すなわち、ファイルポインタのエントリの数は、ＦＩＴブロック中に存在し得るＦＩＴファイルの最大数に等しい。ファイルポインタのエントリは、ＦＩＴファイル番号に従って逐次的に格納される。Ｎ番目のファイルポインタのエントリは、ＦＩＴブロック内のＦＩＴファイルＮへのポインタを含む。ファイルポインタのエントリは、以下の２つのフィールドを有する。
１）ＦＩＴブロック内の物理ページを指定するページ番号
２）物理ページ内のＦＩＴエントリを指定するエントリ番号
【０１９３】
ファイルポインタのエントリは、ＦＩＴブロック内の論理ＦＩＴファイル番号をブロック内の物理位置に変換するための機構を提供する。ファイルポインタの完全セットは、すべてのＦＩＴページがプログラムされるときに更新されるが、最近プログラムされたページにおいてのみ有効である。ＦＩＴファイルがＦＩＴ更新ブロックにおいて更新される場合、ＦＩＴブロック中またはＦＩＴ更新ブロック中のその以前の位置は古くなり、ファイルポインタによりもはや参照されない。
【０１９４】
１０．５．５ＦＩＴ更新ブロック
ＦＩＴブロック中のＦＩＴファイルの変化は、すべてのＦＩＴブロックにより共有される単一のＦＩＴ更新ブロックにおいてなされる。物理ＦＩＴブロックの例が図１０－６に示してある。
【０１９５】
ファイルデータポインタは、ＦＩＴファイルへの論理ポインタである。そのＦＩＴ範囲フィールドは、このＦＩＴ範囲にマッピングされるＦＩＴブロックの物理ブロックアドレスを識別するためにＦＩＴブロックリストにアドレス指定するために用いられる。ＦＩＴファイルポインタのＦＩＴファイル番号フィールドは次に、ＦＩＴブロック中の目標ＦＩＴファイルのための正しいファイルポインタを選択する。
【０１９６】
ファイルデータポインタのＦＩＴ範囲フィールドおよびＦＩＴファイル番号フィールド双方は、目標ＦＩＴファイルが更新されたかどうかを識別するため、ＦＩＴ更新ブロックリストにアドレス指定するために用いられる。このリスト中にエントリが発見されれば、ＦＩＴ更新ブロックの物理ブロックアドレス、およびＦＩＴファイルの更新されたバージョンの更新ブロック内のＦＩＴファイル番号を提供する。これは、ＦＩＴブロック中のＦＩＴファイルのために用いられるＦＩＴファイル番号と異なり得る。ＦＩＴ更新ブロックは、ＦＩＴファイルの有効なバージョンを含み、ＦＩＴブロック中のバージョンは古い。
【０１９７】
１０．５．６更新操作
ＦＩＴブロックは、統合操作の間のみプログラムされる。これは、ブロック内で密に詰められるＦＩＴファイルという結果になる。ＦＩＴ更新ブロックは、ＦＩＴエントリが修正、追加、または除去されるとき、および圧縮操作の間に更新される。図１０－７は、ＦＩＴファイルに関する更新操作の例を示す。
【０１９８】
ＦＩＴファイルは、統合操作の結果として、ＦＩＴブロック中で密に詰められる。ＦＩＴブロックは、その中に存在できるＦＩＴファイルの最大数があるため、完全に充填されないことがある。ＦＩＴファイルは、１つのページから次のページにオーバーフローすることがある。ＦＩＴブロック中のＦＩＴファイルは、そのＦＩＴファイルがＦＩＴ更新ブロック中で更新および再書き込みされると、古くなる。
【０１９９】

ＦＩＴファイルが更新されると、そのＦＩＴファイルは、ＦＩＴ更新ブロック中の次の利用可能なページ中でその全体が再書き込みされる。ＦＩＴファイルの更新は、既存のＦＩＴエントリの内容の変更またはＦＩＴエントリの数の変更を含み得る。ＦＩＴファイルは、１つのページから次のページにオーバーフローすることがある。ＦＩＴ更新ブロック内のＦＩＴファイルは、同じＦＩＴ範囲とすべて関連する必要はない。

【０２００】

１０．５．７ＦＩＴ範囲の作成

ＦＩＴファイルのための付加的な記憶空間に対応するために新しいＦＩＴ範囲が作成されなければならない場合、ＦＩＴブロックは、直ちに作成されない。この範囲内の新しいデータは当初、ＦＩＴ更新ブロックに書き込まれる。その範囲のための統合操作が実行されるとき、ＦＩＴブロックがその後で作成される。

【０２０１】

１０．５．８圧縮および統合

１０．５．８．１ディレクトリ更新ブロックまたはＦＩＴ更新ブロックの圧縮

ＦＩＴ更新ブロックが一杯になると、その有効なＦＩＴファイルデータが、消去済ブロックに圧縮された形でプログラムされることができ、このブロックが次に更新ブロックになる。更新が少数のファイルにのみ関連すれば、プログラムされるべき圧縮された有効なデータの１つのページしかないこともある。

圧縮操作において再配置されるべきＦＩＴファイルが、閉じられたファイルに関連し、範囲のためのＦＩＴブロックが、十分にプログラムされていないページを含んでいれば、ＦＩＴファイルは、圧縮された更新ブロックではなく、ＦＩＴブロックに再配置され得る。

【０２０２】

１０．５．８．２ディレクトリブロックまたはＦＥＴブロックの統合

ＦＩＴエントリが更新されると、ＦＩＴブロック中の元のＦＩＴファイルは古くなる。そのようなＦＩＴブロックは、古い空間を回復するために、ガベージコレクションを周期的にかけられるべきである。これは、統合操作によって達成される。加えて、新しいファイルが、範囲内で作成され、更新ブロック中のエントリを有することがあるが、ＦＩＴブロック中の対応する古いエントリは全く存在し得ない。そのようなＦＩＴファイルは、ＦＩＴブロックに周期的に再配置されるべきである。

【０２０３】

更新ブロックのＦＩＴファイルは適切な範囲のためのＦＩＴブロックに統合され、従って、更新ブロックから取り除かれうる一方で、他のＦＩＴファイルは更新ブロックにとどまる。

ＦＩＴファイル中のＦＩＴエントリの数が更新プロセスの間に増大し、かつＦＩＴ範囲のための有効なデータを単一の消去済ブロックに統合できなければ、そのＦＩＴ範囲に本来割り当てられたいくつかのＦＩＴファイルが、別のＦＩＴ範囲に割り当てられ、統合が、別個の操作において２つのブロックへ実行され得る。ＦＩＴファイルのそのような再割り当ての場合、ディレクトリ中のファイルデータポインタは、新しいＦＩＴ範囲を反映するために更新されなければならない。

【０２０４】

ある範囲のための統合操作は、ＦＩＴ更新ブロック中のその範囲のための有効なデータの容量が定義されたしきい値に達したときに、実行されるべきである。このしきい値の例は、５０％である。

圧縮は、いまだ開き、ホストがアクセスを続け得るファイルに関するアクティブなＦＩＴファイルについての統合に優先して実行されるべきである。

【０２０５】

１０．６情報テーブル

情報テーブルは、セクション１０．５においてファイルインデックステーブルについて定義された同じ構造、インデックス化機構および更新手法を用いる。しかし、ファイルの

ためのfile_infoは、直接データファイル・プラットフォーム内では解釈されない単一の情報ストリングを含む。

【０２０６】

１０．７データグループ

データグループは、単一のメモリブロック中の隣接する物理アドレスにおいてプログラムされた、ファイルのための隣接するオフセットアドレスを有するファイルデータのセットである。ファイルは通常、いくつかのデータグループとしてプログラムされる。データグループは、１バイトと１ブロックとの間のどのような長さでも有し得る。

【０２０７】

１０．７．１データグループヘッダ

各データグループは、相互参照を目的としてファイル識別子情報を含むヘッダと共にプログラムされる。ヘッダは、データグループがその一部を形成するファイルのためのＦＩＴファイルポインタを含む。

【０２０８】

１１．ブロック状態管理

１１．１ブロック状態

ファイルデータ記憶のためのブロックは、図１１－１の状態図に示されるように、以下の８つの状態に分類できる。

【０２０９】

１１．１．１消去済ブロック

消去済ブロックは、消去済ブロックプール中で消去済状態にある。この状態から可能な移行は以下の通りである。

（ａ）消去済ブロックからプログラムブロック

単一のファイルのためのデータは、そのデータがホストから供給される場合、またはそのデータがファイルのためのガベージコレクションの間にコピーされる場合に、消去済ブロックにプログラムされる。

【０２１０】

１１．１．２プログラムブロック

プログラムブロックは、単一のファイルのための有効なデータで部分的にプログラムされ、いくらかの消去済容量を含む。ファイルは、開いているか、または閉じられている。ファイルのためのさらなるデータは、ホストにより供給される場合、またはファイルのガベージコレクションの間にコピーされる場合に、ブロックにプログラムされるべきである。

【０２１１】

この状態からの可能な移行は、以下の通りである。

（ｂ）プログラムブロックからプログラムブロック

単一のファイルのためのデータは、そのデータがホストから供給される場合、またはそのデータがファイルのためのガベージコレクションの間にコピーされる場合に、そのファイルのためのプログラムブロックにプログラムされる。

（ｃ）プログラムブロックからファイルブロック

ホストからの単一のファイルのためのデータは、そのファイルのためにプログラムブロックを満たすようにプログラムされる。

（ｆ）プログラムブロックから古いブロック

プログラムブロック中のファイルのためのすべてのデータは、有効なデータがガベージコレクションの間に別のブロックにコピーされる結果、あるいはファイルのすべてまたは一部がホストにより削除される結果として、古くなる。

（ｈ）プログラムブロックから古いプログラムブロック

プログラムブロック中のデータの一部は、データの更新されたバージョンがホストにより同じプログラムブロックに書き込まれる結果、またはファイルの一部がホストにより削除される結果として、古くなる。

（ｌ）プログラムブロックから共通ブロック
あるファイルのための残りのデータは、そのファイルまたは共通ブロックのガベージコレクションの間、あるいはプログラムブロックの統合の間に、異なる閉じられたファイルのためのプログラムブロックにプログラムされる。
【０２１２】
１１．１．３ファイルブロック
ファイルブロックは、単一のファイルのための完全に有効なデータで満たされる。
この状態からの可能な移行は、以下の通りである。
（ｄ）ファイルブロックから古いファイルブロック
ファイルブロック中のデータの一部は、データの更新されたバージョンがホストによりファイルのためのプログラムブロックにプログラムされる結果として、古くなる。
（ｇ）ファイルブロックから古いブロック
ファイルブロック中のすべてのデータは、ブロック中のデータの更新されたバージョンがホストによりファイルのためのプログラムブロックにプログラムされる結果、あるいはファイルのすべてまたは一部がホストにより削除される結果として、古くなる。
【０２１３】
１１．１．４古いファイルブロック
古いファイルブロックは、単一のファイルのための有効なデータと古いデータとの任意の組み合わせで満たされる。
この状態からの可能な移行は、以下の通りである。
（ｅ）古いファイルブロックから古いブロック
古いファイルブロック中のすべてのデータは、ブロック中の有効なデータの更新されたバージョンがホストによりファイルのためのプログラムブロックにプログラムされる結果、有効なデータがガベージコレクションの間に別のブロックにコピーされる結果、あるいはファイルのすべてまたは一部がホストにより削除される結果として、古くなる。
【０２１４】
１１．１．５古いプログラムブロック
古いプログラムブロックは、単一のファイルのための有効なデータおよび古いデータの任意の組み合わせで部分的にプログラムされ、いくらかの消去済容量を含む。ファイルのためのさらなるデータは、ホストにより供給されるときに、そのブロックにプログラムされるべきである。しかし、ガベージコレクションの間、ファイルのためのデータは、ブロックにコピーされるべきでなく、新しいプログラムブロックが開かれるべきである。
【０２１５】
この状態からの可能な移行は、以下の通りである。
（ｉ）古いプログラムブロックから古いプログラムブロック
単一のファイルのためのデータは、ホストから供給されるときに、そのファイルのための古いプログラムブロックにプログラムされる。
（ｊ）古いプログラムブロックから古いブロック
古いプログラムブロック中のファイルのためのすべてのデータは、有効なデータがガベージコレクションの間に別のブロックにコピーされる結果、あるいはファイルのすべてまたは一部がホストにより削除される結果として、古くなる。
（ｋ）古いプログラムブロックから古いファイルブロック
単一のファイルのためのデータは、ホストから供給されるときに、そのファイルのための古いプログラムブロックを満たすためにプログラムされる。
【０２１６】
１１．１．６共通ブロック
共通ブロックは、２つ以上のファイルのための有効なデータでプログラムされ、通常いくらかの消去済容量を含む。どのようなファイルのための残りのデータも、ガベージコレクションまたはプログラムブロックの統合の間にプログラムされ得る。
この状態からの可能な移行は、以下の通りである。

（ｍ）共通ブロックから共通ブロック
ファイルのガベージコレクションの間の共通ブロックまたは共通ブロック、またはプログラムブロックの統合の間に、ファイルのための残りのデータがプログラムされる。
（ｎ）共通ブロックから古い共通ブロック
共通ブロック中の１つのファイルのためのデータの一部またはすべては、データの更新されたバージョンがホストによりそのファイルのためのプログラムブロックにおいてプログラムされる結果、ファイルのガベージコレクションの間にデータが別のブロックにコピーさる結果、あるいはファイルのすべてまたは一部がホストにより削除される結果として、古くなる。
【０２１７】
１１．１．７古い共通ブロック
古い共通ブロックは、２つ以上のファイルのための有効なデータおよび古いデータの任意の組み合わせでプログラムされ、通常いくらかの消去済容量を含む。さらなるデータが、そのブロックにプログラムされるべきではない。
この状態からの可能な移行は、以下の通りである。
（ｏ）古い共通ブロックから古いブロック
古い共通ブロック中のすべてのファイルのためのデータは、１つのファイルのためのデータの更新されたバージョンがホストによりそのファイルのためのプログラムブロックにおいてプログラムされる結果、ファイルのガベージコレクションの間に１つのファイルのためのデータが別のブロックにコピーされる結果、あるいは１つのファイルのすべてまたは一部がホストにより削除される結果、古くなる。
【０２１８】
１１．１．８古いブロック
古いブロックは古いデータのみを含んでいるが、まだ消去されていない。
この状態からの可能な移行は、以下の通りである。
（ｐ）古いブロックから消去済ブロック
古いブロックは、ガベージコレクションの間に消去され、消去済ブロックプールに戻して付加される。
【０２１９】
１２．消去済ブロック管理
１２．１メタブロックリンキング
消去ブロックをメタブロックにリンクする方法は、以前の第３世代のＬＢＡシステムのために定義された方法と変わっていない。
【０２２０】
１２．２消去済ブロックプール
消去済ブロックプールは、ファイルデータまたは制御情報の記憶のための割り当てに利用可能なデバイス中の消去済ブロックのプールである。プール中の各消去済ブロックはメタブロックであり、すべてのメタブロックは、同じ固定された並列性を有する。
プール中の消去済ブロックは、制御ブロック中の消去済ブロックログ中のエントリとして記録される。エントリは、ブロックの消去順序に従ってログ中で並べられる。割り当てのための消去済ブロックが、ログの頭におけるエントリとして選択される。ブロックが消去されるときに、エントリがログの末尾に加えられる。
【０２２１】
１３．制御データ構造
制御データ構造は、目的専用のフラッシュブロック中に格納される。ブロックの３つのクラスが、以下の通り定義される。
１）ファイルディレクトリブロック
２）ファイルインデックステーブルブロック
３）制御ブロック
【０２２２】

１３．１ファイルディレクトリブロック
ファイルディレクトリブロックの構造は、前に説明されている。
【０２２３】
１３．２ファイルインデックステーブルブロック
ファイルインデックステーブルブロックの構造は、前に説明されている。
【０２２４】
１３．３制御ブロック
制御ブロックは、制御情報を４つの独立ログに格納する。ログの各々に、別個のページが割り当てられる。必要であれば、１つのログあたり複数のページに拡張できる。制御ブロックの形式の例が、図１３－１に示してある。
【０２２５】
ログは、制御ポインタにより定義された次の消去済ページ位置に、完全ログの改訂バージョンを書き込むことにより更新される。複数のログは、必要であれば、それらをメタページ中の異なるページにプログラムすることにより、同時に更新され得る。４つのログの各々の有効なバージョンのページ位置は、制御ブロック中の最近書き込まれたページ中のログポインタにより識別される。
【０２２６】
１３．３．１共通ブロックログ
共通ブロックログは、デバイス中に存在するすべての共通ブロックについての情報を記録する。共通ブロックログ中のログエントリは、図１３－２において説明されるように、ブロックエントリのための第１の領域およびデータグループエントリのための第２の領域の２つの領域に再分割される。各ブロックエントリは、共通ブロックの物理位置を記録する。エントリは、固定サイズであり、決まった数が共通ブロックログ中に存在する。各エントリは、以下のフィールドを有する。
１）ブロック物理アドレス
２）プログラミングのための共通ブロック中の次の利用可能なページへのポインタ
３）ブロックのためのデータグループエントリの最初のものへのポインタ
４）データグループエントリの数
【０２２７】
データグループエントリは、共通ブロック中のデータグループについての情報を記録する。隣接しているデータグループエントリのセットは、共通ブロック中のすべてのデータグループを定義する。共通ブロック中には、可変数のデータグループがある。各エントリは、好ましくは、以下のフィールドを有する。
１）共通ブロック内のバイトアドレス
２）ＦＩＴファイルポインタ
【０２２８】
１３．３．２プログラムブロックログ
プログラムブロックログは、閉じられたファイルのためのデバイス中に存在するすべてのプログラムブロックについての情報を記録する。各プログラムブロックについて、１つのエントリが存在し、以下のフィールドを有する。
１）物理アドレスブロック
２）プログラミングのためのプログラムブロック中の次の利用可能なページへのポインタ
３）ＦＩＴファイルポインタ
【０２２９】
１３．３．３消去済ブロックログ
消去済ブロックログは、デバイス中に存在するすべての消去済ブロックの同一性を記録する。各消去済ブロックについて１つのエントリが存在する。エントリは、ブロックの消去順に従ってログ中で並べられる。割り当てのための消去済ブロックが、ログの頭におけるエントリとして選択される。ブロックが消去されるときに、エントリがログの末尾に加

えられる。エントリは、単一のフィールド、すなわち、ブロック物理アドレスを有する。
【０２３０】
１３．３．４制御ログ
制御ログは、様々な制御情報を以下のフィールドに記録する。
【０２３１】
１３．３．４．１開いているファイルリスト
このフィールドは、以下のような、現在開いているファイルの各々についての情報を含む。
１）パス名
２）ファイル名
３）ＦＩＴファイルポインタ
４）プログラムブロック物理アドレス
開いているファイルのためのプログラムブロックは、プログラムブロックログに含まれない。
【０２３２】
１３．３．４．２共通ブロックカウント
このフィールドは、共通ブロックログ中に記録された共通ブロックの総数を含む。
【０２３３】
１３．３．４．３プログラムブロックカウント
このフィールドは、プログラムブロックログ中に記録されたプログラムブロックの総数を含む。ブロックがプログラムブロックログに付加され、かつこれから除去されるときに、カウントが更新される。
【０２３４】
１３．３．４．４消去済ブロックカウント
このフィールドは、消去済ブロックログ中に記録された消去済ブロックの総数を含む。ブロックが消去済ブロックログに付加され、かつこれから除去されるときに、カウントが更新される。
【０２３５】
１３．３．４．５プログラム／共通ブロックページカウント
このフィールドは、プログラムブロックおよび共通ブロック中の有効なデータページの数のカウントを含む。ブロックがプログラムブロックログおよび共通ブロックログに付加され、かつこれらから除去されるときに、カウントが更新される。
【０２３６】
１３．３．４．６古いブロック計算
このフィールドは、ガベージコレクションを待つ完全に古いブロックの数のカウントを含む。ブロックが古いブロックのガベージコレクションキューに付加され、かつこれから除去されるときに、カウントが更新される。
【０２３７】
１３．３．４．７ＦＩＴブロックリスト
このフィールドは、ＦＩＴ範囲をＦＩＴブロックへマッピングするための情報を含む。このフィールドは、各ＦＩＴ範囲についてのＦＩＴブロック物理アドレスを定義するエントリを含む。
【０２３８】
１３．３．４．８ＦＩＴ更新ブロックリスト
このフィールドは、ＦＩＴ範囲およびＦＩＴファイル番号をＦＩＴ更新ファイル番号にマッピングするための情報を含む。このフィールドは、更新ブロック中に存在する各有効なＦＩＴファイルについてのエントリを含む。エントリは、以下の３つのフィールドを有する。
１）ＦＩＴ範囲
２）ＦＩＴファイル番号

３）ＦＩＴ更新ファイル番号
【０２３９】
１３．３．４．９ディレクトリブロックリスト
このフィールドは、ディレクトリ範囲をディレクトリブロックへマッピングするための情報を含む。このフィールドは、各ディレクトリ範囲についてのディレクトリブロック物理アドレスを定義するエントリを含む。
【０２４０】
１３．３．４．１０ディレクトリ更新ブロックリスト
このフィールドは、サブディレクトリ番号を更新するためにディレクトリ範囲およびサブディレクトリ番号をマッピングするための情報を含む。このフィールドは、更新ブロック中に存在する有効なサブディレクトリの各々についてのエントリを含む。エントリは、以下の３つのフィールドを有する。
１）ディレクトリ範囲
２）サブディレクトリ番号
３）更新サブディレクトリ番号
【０２４１】
１３．３．４．１１バッファスワップブロックインデックス
このフィールドは、スワップブロック中の有効なデータグループのインデックスを含む。各データグループについてのインデックスは、以下のフィールドを含む。
１）ＦＩＴファイルポインタ
２）スワップブロック内のバイトアドレス
３）長さ
【０２４２】
１３．３．４．１２プライオリティの古いブロックキュー
このフィールドは、ガベージコレクションのためのプライオリティの古いブロックキュー中のすべてのブロックのブロックアドレスを含む。
【０２４３】
１３．３．４．１３プライオリティの共通ブロックキュー
このフィールドは、ガベージコレクションのためのプライオリティの共通ブロックキュー中のすべてのブロックのブロックアドレスを含む。
【０２４４】
１３．３．４．１４古いブロックキュー
このフィールドは、ガベージコレクションのための古いブロックキュー中のすべてのブロックのブロックアドレスを含む。
【０２４５】
１３．３．４．１５共通ブロックキュー
このフィールドは、ガベージコレクションのための共通ブロックキュー中のすべてのブロックのブロックアドレスを含む。
【０２４６】
１３．３．４．１６ファイルキュー
このフィールドは、ガベージコレクションのためのファイルキュー中のすべてのファイルのＦＩＴファイルポインタを含む。
【０２４７】
１４．静的ファイル
１４．１静的ファイル
いくつかのホストは、同一サイズを有するフィルのセットを作成し、そのセット中のファイル内でデータを周期的に更新することにより、直接データファイルデバイス中にデータを格納し得る。そのようなセットの一部であるファイルは、静的ファイルと呼ばれる。ホストは、メモリカードの外部にあってもよく、またはオンカードアプリケーションを実行しているメモリカード内のプロセッサであってもよい。

【０２４８】
静的ファイルの使用の応用例が、本願と同時に出願されたSergey Anatolievich Gorobetsによる「Interfacing Systems Operating Through A Logical Address Space and on a Direct Date File Basis 」（特許文献１２）という米国特許出願に記載されている。その出願において、ホストの論理アドレス空間は、メモリコントローラによりそのような静的ファイルに分割される。
【０２４９】
直接データファイルデバイスは、何か他のファイルについてと全く同じ方法で静的ファイル記憶を管理する。しかし、ホストは、静的ファイルによってデバイスの挙動および性能を最適化するやり方で直接データファイルコマンドセット中のコマンドを用い得る。
【０２５０】
１４．１．１静的ファイルパーティション
静的ファイルは、デバイス中の専用パーティション中のセットとして格納される。１つのパーティション中のすべての静的ファイルは、同一のファイルサイズを有する。
【０２５１】
１４．１．２静的ファイルサイズ
ファイルサイズは、ファイルに書き込まれたオフセットアドレスの範囲を介して、ホストにより定義される。静的ファイルは、メタブロックのサイズに等しいサイズを有する。
ホストは、write_pointer およびread_pointer により表されるファイルオフセット値を管理し、それらの値を、静的ファイルに許された値の範囲内に常に維持する。
【０２５２】
１４．１．３静的ファイルの削除
直接データファイルデバイス中の他のファイルとは異なり、ホストは、通常操作の間に、静的ファイルを削除しない。静的ファイルは、ホストにより作成され、次にデバイス中に連続的に存在する。ファイルにいつ書き込まれたデータも、既存のファイルデータを上書きする。
しかし、ホストは常に、例えば、デバイスを再フォーマットするためまたはデバイス中の静的ファイルのためにパーティションのサイズを減少させるためのホストによる操作間で、静的ファイルを削除する能力を有する。
【０２５３】
１４．２静的ファイルと共に用いられるコマンドセット
図１４－１は、静的ファイルと共に用いるコマンドセットを示し、そのサブセットが図２－１～２－６に示してあり、これらは、静的ファイルのとって必要とされるすべての操作をサポートする。
【０２５４】
１４．３静的ファイルの作成
静的ファイルは、ホストからのcreateコマンドの使用によってデバイス中で作成される。ホストは通常、ホストがそれを用いてファイルを識別することを望むfileIDを指定する。
ホストは、デバイス中でホストがどのファイルを作成したかを追跡するか、あるいはそのfileIDがすでにデバイス中に存在しないファイルを開くことをホストが試みた後のデバイスからのエラーメッセージに応答してファイルを作成することができる。
【０２５５】
１４．４静的ファイルを開く
ホストは、パラメータとしてファイルのためのfileIDを用いるopenコマンドを送ることにより、静的ファイルを開く。
ホストは、デバイス中で同時に開いているファイルの数、またはデバイス中で同時に開いているホストにより定義された特定タイプのファイルの数をホストが制御するように、静的ファイルのセットを用いてデバイス中で動作し得る。従って、ホストは、別の静的ファイルを開く前に、１つ以上の静的ファイルを閉じることができる。

【０２５６】
１４．５静的ファイルへの書き込み
静的ファイルが最初に書き込まれるとき、その静止ファイルは、デバイス中の単一の完全ファイルブロックを占める。なぜならば、ファイルサイズは、ホストにより、フラッシュメモリ中のメタブロックのサイズと正確に等しいと定義されるからである。従って、ファイルのためのオフセットアドレス範囲は、フラッシュメモリ中のメタブロックのサイズと正確に等しい。
【０２５７】
静的ファイルへのその後の書き込みにより、このオフセットアドレス範囲内でデータが更新される。ホストは、write_pointer コマンドを用いてファイルのためのwrite_pointer 値を制御することによってデータが更新されるファイルオフセットアドレスを制御する。ホストは、静的ファイルのサイズに関するオフセットアドレス範囲の終わりをwrite_pointer 値が超えることを許さない。同様に、ホストは、read_pointer 値をこの範囲内に拘束する。
【０２５８】
ファイルが開かれた後に静的ファイル中の既存データが更新されると、更新されたデータがプログラムされるプログラムブロックが開かれる。ファイルブロック中の対応するオフセットアドレスを有するデータは古くなる。完全な静的ファイルが更新されれば、プログラムブロック中のすべてのデータは有効であり、プログラムブロックはファイルのためのファイルブロックになる。ファイルのための以前のファイルブロック中のすべてのデータは古くなり、ブロックは、古いブロックのガベージコレクションキューに加えられる。ファイルのためにさらなる更新データが受信されれば、消去済ブロックは、プログラムブロックとして割り当てられる。
【０２５９】
静的ファイルのためのプログラムブロックが一杯になるが、そのファイルのためのすべての有効なデータを含んでいなければ、プログラムブロック中のデータのうちのいくつかは古い。なぜならば、複数の更新が同じオフセットアドレスになされているからである。この場合、プログラムブロックは、ファイルブロックになれず、ファイルのためのさらなるデータが受信されるときに、別の空のプログラムブロックは開かれない。プログラムブロック（プログラムブロックは圧縮されている）からの有効なデータがコピーされる消去済ブロックが割り当てられ、この部分的に満たされたブロックは次に、ファイルのためのプログラムブロックになる。ファイルのための以前のプログラムブロック中のすべてのデータは今では古く、ブロックは、古いブロックのガベージコレクションキューに加えられる。
【０２６０】
ホストが、以下のセクション１４．６において記載されるようにファイルを閉じることにより、ファイルのためのいくつかの有効なデータを各々含んでいる、ファイルブロックとプログラムブロックとの統合とを強制できることに留意されたい。ホストは、ファイルのためのさらなるデータが受信されたときに、直接データファイルデバイスがプログラムブロックを圧縮できるようにするのではなく、部分的に古いプログラムブロックが一杯になったときに、一時的にファイルを閉じることを選ぶことができる。
【０２６１】
１４．６静的ファイルの閉鎖
ホストは、パラメータとしてファイルのためのfileIDを用いるcloseコマンドを送ることにより、静的ファイルを閉じる。
【０２６２】
ファイルのためのデータの一部のみが更新されていれば、静的ファイルの閉鎖は、ファイルをファイルガベージコレクションキューに入れる。これは、以下のセクション１４．７に記載されるように、ファイルのためのその後のガベージコレクション操作を可能にする。しかし、ホストは、同じくセクション１４．７に記載されるように、ファイルに関す

る即時のガベージコレクション操作を強制することができる。

【０２６３】

１４．７静的ファイルのガベージコレクション

ファイル中のデータの一部の更新に続いて、ファイルガベージコレクションキュー中のエントリを有する静的ファイルが閉じられている。ファイルのためのファイルブロックは、いくつかの有効なデータおよびいくつかの古いデータを含み、プログラムブロックは、いくつかの有効なデータ、ことによるといくつかの古いデータ、およびことによるといくらかの消去済容量を含む。

【０２６４】

ファイルガベージコレクション操作は、ファイルのためのすべての有効なデータを単一のブロックに統合する。プログラムブロックが古いデータをまったく含まなければ、有効なデータは、ファイルブロックからプログラムブロックにコピーされ、ファイルブロックは、消去される。プログラムブロックが古いデータを含んでいれば、ファイルブロックおよびプログラムブロック双方からのすべての有効なデータが、消去済ブロックにコピーされ、ファイルブロックおよびプログラムブロック双方が消去される。

【０２６５】

ファイルガベージコレクションは、ガベージコレクションのスケジューリング・アルゴリズムにより決定されるときに、エントリがキューの頭に達する場合に、実行される。しかし、ホストがファイルを閉じるとき、ホストはファイルに関する即時のガベージコレクション操作を強制することができる。ホストは、ファイルのためのclose コマンドの直後にidleコマンドを送ることによりこれを行い、これにより、別のコマンドが受信されるまで、デバイスにガベージコレクションまたはブロック統合操作を連続的に実行させる。ホストは、別のコマンドを送る前に、デバイスが内部操作の実行でもはやビジーではないことをホストが検出するまで、デバイスの内部ビジーステータスを監視する。この機構により、ファイルが閉じられてすぐのファイルについてのファイルブロックおよびプログラムブロックの統合が、ホストにより保証され得る。

【０２６６】

前述した例としてのメモリシステムの概要

直接データファイル・プラットフォーム

直接データファイル・プラットフォームは、フラッシュメモリにおいてデータ記憶を管理するための全般的なバックエンドシステムとして働く。

直接データファイルインターフェイスは、複数のデータのソースをサポートする内部ファイル記憶インターフェイスである。

あらかじめ定められた長さを持たないファイルデータのランダムな読み出し／書き込みアクセスを有するファイルインターフェイスである。

あらかじめ定められた長さを有する完全なファイルオブジェクトの転送を有するオブジェクトインターフェイスである。

ファイルシステムを組み込んだ従来のホストへのＬＢＡインターフェイスである。論理ブロックは、論理ファイルとして格納される。

ファイル内のデータへのランダムアクセスを有する埋め込みアプリケーションプログラムである。

直接データファイル記憶は、ファイルごとのデータ記憶を組織するバックエンドシステムである。

記憶デバイスのための論理アドレス空間はない。

ファイルシステムはない。

【０２６７】

直接データファイル対先行システム

直接データファイル・プラットフォームは、先行システムに対して以下の利点を提供する。

高いデータ書き込み速度：

ファイル断片化による徐々に進行する性能低下が排除され、
最高データ書き込み速度は、ホストにより削除されるファイルがバックグラウンド操作において消去されるときに、増大され得る。
データ書き込み速度の均一性：
ストリーミングデータのための維持される書き込み速度は、ガベージコレクションが、バックグラウンドにおいて、またはホストデータの書き込みとインターリーブされたバーストにおいて実行されるときに、改善され得る。

【０２６８】

以下の利点は、直接データファイル・プラットフォームにおいて用いられるアルゴリズムの特性の結果である。
限定されたファイル断片化
限定されたファイルおよびブロック統合
真のファイル削除
最適なファイルデータインデックス化
効率的なガベージコレクション

【０２６９】

直接データファイルインターフェイス－望ましい特徴

直接データファイルインターフェイスは、ホスト中の動作システムから独立しているべきである。
数値識別子を有するファイルは、フラットな階層において管理され、
ファイルと関連付けられたデータは、インターフェイスより上のレベルにおける階層ディレクトリの構築および維持を可能にするために、格納され得る。

【０２７０】

直接データファイルインターフェイスは好ましくは、ファイルデータ転送の以下の種々の形式をサポートする。
サイズが未定義であり、データをそれにストリームできるファイル、
書き込まれる前にサイズが定義されるファイル、
サイズが固定され、永続的に存在するファイル。

【０２７１】

直接データファイルインターフェイス－実施

ファイル内のデータは、１バイトの粒度のランダムな書き込みおよび読み出しアクセスを有する。
データは、ファイルのためのデータに、追加、上書き、または内部に挿入され得る。
書き込みまたは読み出されるファイルデータは、長さをあらかじめ定義せずに、デバイスへまたはデバイスからストリームされる。
現在の操作は、別のコマンドの受信によって終了される。
ファイルは、データを書き込むために開かれ、ファイルの終わりにおいて、またはファイルが非活動状態にあるときに閉じられる。
ファイルハンドルが、ホストにより指定されたファイルのためのデバイスにより返される。
階層ディレクトリがサポートされるが、維持されない。
ファイルのための関連付けられた情報が、格納され得る。
その中においてデバイスがバックグラウンドで内部操作を実行できる状態が、ホストにより開始され得る。

【０２７２】

直接データファイルインターフェイス－コマンドセット

ファイルコマンド：
ファイルオブジェクトを制御するためのコマンド、create，open，close，delete，erase，List_files
データコマンド：

ファイルデータを書き込みおよび読み出すためのコマンド、write，insert，remove，read，save_buffer，write_pointer，read_pointer
情報コマンド：
ファイルに関連付けられた情報を書き込みおよび読み出すためのコマンド、write_info、read_info，info_write_pointer，info_read_pointer
状態コマンド：
デバイスの状態を制御するためのコマンド、idle，standby，shutdown
デバイスコマンド：
デバイスに問い合わせるためのコマンド、capacity，status

【０２７３】

ファイル－ツー－フラッシュマッピングアルゴリズム

データ構造：
ファイル
データグループ
ブロックタイプ：
プログラムブロック
ファイルブロック
共通ブロック
ファイルタイプ：
プレーンファイル
共通ファイル
編集済ファイル
メモリ回復：
ガベージコレクション
ブロック統合

【０２７４】

ファイル－ツー－フラッシュマッピングアルゴリズム－データ構造

ファイル：
ファイルは、ホストにより作成および維持されるデータのセットであり、
ホストは、外部ホストであってもよく、またはメモリカード内のアプリケーションプログラムであってもよく、
ファイルは、ホストによって作成されたファイル名により、または直接データファイル・プラットフォームによって作成されたファイルハンドルにより識別され、
ファイル内のデータは、ファイルオフセットアドレスにより識別され、
異なるファイルのためのオフセットアドレスのセットは、デバイス内の独立した論理アドレス空間として働く。デバイス自体のための論理アドレス空間は皆無である。

【０２７５】

データグループ：
データグループは、ファイル内の隣接したオフセットアドレスを有する単一のファイルのためのデータのセットであり、
データグループは、単一のブロック中の隣接した物理アドレスに格納され、
データグループは、１バイトから１ブロックまでの間のどのような長さでも持つことができ、
データグループは、物理フラッシュアドレスに論理ファイルアドレスをマッピングするための基本単位である。

【０２７６】

ファイル－ツー－フラッシュマッピングアルゴリズム－ブロックタイプ

プログラムブロック：
ホストにより書き込まれたすべてのデータは、プログラムブロックにおいてプログラムされ、

プログラムブロックは、単一のファイルのためのデータ専用であり、
プログラムブロック中のファイルデータは、ファイルオフセットアドレスのどのような順序であってもよく、プログラムブロックは、ファイルのための複数のデータグループを含むことができ、
各開いているファイルのため、および指定されない数の閉じられたファイルのために別個のプログラムブロックが存在する。

【０２７７】

ファイルブロック：
プログラムブロックは、その最後の位置がプログラムされたときに、ファイルブロックになる。
共通ブロック：
共通ブロックは、１つ以上のファイルのためのデータグループを含み、
共通ブロックは、共通ブロックのガベージコレクションの間またはブロック統合操作の間に、無関係なファイルのためのデータグループをプログラムブロックにプログラムすることによって作成され、
データグループは、別の共通ブロックのガベージコレクションの間またはブロック統合操作の間に、共通ブロックに書き込まれ得る。

【０２７８】

ファイル－ツー－フラッシュマッピングアルゴリズム－ファイルタイプ
プレーンファイル（図３－１参照）：
プレーンファイルは、任意の数の完全なファイルブロックおよび１つの部分的に書き込まれたプログラムブロックを含む。
プレーンファイルは、その消去前に、どのブロックからもデータを再配置する必要なく、削除され得る。

【０２７９】

共通ファイル（図３－２参照）：
共通ファイルは、任意の数の完全なファイルブロック、および１つの共通ブロックを含み、この共通ブロックは、ファイルのためのデータを、他の無関係なファイルのためのデータと共に含む。
共通ブロックのみに関するガベージコレクション操作は、削除されるファイルに続いて実行されなければならない。

【０２８０】

編集済ファイル（図３－３および３－４参照）
編集済ファイルは、上書きされた既存のオフセットアドレスにおけるデータの結果として、そのブロックの１つ以上に、古いデータを含む。
古いデータにより占められたメモリ容量は、ファイルガベージコレクション操作によって回復され得る。
ファイルガベージコレクション操作は、編集済ファイルをプレーンファイル形式に復元する。

【０２８１】

ファイル－ツー－フラッシュマッピングアルゴリズム－メモリ回復
ガベージコレクション：
ガベージコレクション操作は、古いデータにより占められたメモリ容量を回復するために実行される。
保留中の操作は、ガベージコレクションキューに記録され、スケジューリング・アルゴリズムに従って最適なレートで後に実行される。
ガベージコレクションは、ホストコマンドにより開始され、ホストインターフェイスが動作休止している間に、バックグラウンドで実行され得る。操作は、何か他のホストコマンドの受信と同時に一時停止される。
ガベージコレクションは、フォアグラウンド操作として、ホストデータ書き込み操作と

インターリーブされたバーストにおいても実行され得る。

【０２８２】

ブロック統合：

進行中のブロック統合プロセスは、プログラムブロックおよび共通ブロック中にロックアップされた消去済容量を回復するために実施され得る。

プログラムブロック中のファイルデータの容量および共通ブロックの削除済ファイルのための古いデータの容量の分布が不均衡である場合にのみ、必要である。

複数のプログラムブロックまたは共通ブロック中のデータは、１つ以上のブロックの消去を可能にするために統合される。

【０２８３】

ファイルデータのプログラミング

ファイルハンドルにより識別されたファイルのためのデータは、そのデータがwrite コマンドまたはinsertコマンドに続いてホストからストリームされるにつれて、フラッシュメモリにプログラムされる。

データの当初のファイルオフセットアドレスは、その値がホストにより設定され得る書き込みポインタにより定義される。

バッファメモリ中に十分なデータが蓄積されたら、メタページが、ファイルのためのプログラムブロック中にプログラムされる。

プログラムブロックが一杯になると、そのプログラムブロックはファイルブロックと呼ばれ、消去済ブロックが、ファイルのための新しいプログラムブロックとして割り当てられる。

プログラムブロックが一杯になるときはいつも、または別のホストコマンドが受信されたときはいつも、データグループのインデックス化構造が、フラッシュメモリ中で更新される。

ファイルデータのプログラミング手順は、適応スケジューリング・アルゴリズムにより決定されるホストデータストリームにおける間隔で、フォアグラウンドガベージコレクションのバーストを開始する。

別のホストコマンドが受信されると、ファイルデータのプログラミング手順が終了される。

【０２８４】

ファイルデータの読み出し

ファイルハンドルにより識別されたファイルのためのデータは、フラッシュメモリから読み出され、readコマンドに続いてホストにストリームされる。

データの当初のファイルオフセットアドレスは、その値がホストにより設定され得る読み出しポインタにより定義される。

ファイルデータは、ファイルの終わりに達するまで、または別のホストコマンドが受信されるまで、１メタページ単位で読み出される。

データは、ファイルオフセットアドレス順にホストへ転送される。

ファイルのために読み出されるデータグループの位置は、ファイルのインデックス化構造により定義される。

別のホストコマンドが受信されると、ファイルデータ読み出し手順が終了される。

【０２８５】

ファイルの削除

ファイルのためのdeleteコマンドに応答して、そのファイルのためのデータを含むブロックが識別され、その後のガベージコレクション操作のためのガベージコレクションキューに加えられる。

ファイルディレクトリおよびファイルインデックステーブルは、ファイルのためのエントリを除去するために更新される。

ファイルを削除するための手順は、ガベージコレクション操作を開始せず、ファイルのためのデータは、直ちには消去されない。

ファイルのためのerase コマンドに応答して、deleteコマンドについてと同じ手順がたどられるが、ガベージコレクション操作が開始され、何か他のホストコマンドが実行される前に、完了される。

【０２８６】

ガベージコレクション

ガベージコレクションは、古いデータにより占められたフラッシュ容量を回復する操作である。

その後のガベージコレクション操作を定義するために、デバイスの操作の間に、オブジェクトが３つのガベージコレクションキューにその時々に加えられる。

古いブロックキュー－ファイルデータの更新またはファイルの削除の結果としてブロックが完全に古くなると、そのブロックは、このキューに加えられる。

共通ブロックキュー－複数のファイルのためのデータを含むブロックの一部の中のデータが、ファイルデータ更新、ファイルの削除、またはファイルのガベージコレクションの結果として古くなると、そのデータは、このキューに加えられる。

ファイルキュー－ファイルがホストにより閉じられると、そのファイルは、このキューに加えられる。オブジェクトは、プライオリティのガベージコレクションのために指定され得る。

ガベージコレクション操作は、以下の２つの方法でスケジュールを組みこまれ得る。

バックグラウンド操作は、ホストがデバイスへの読み出しまたは書き込み操作を行っていないときに、ホストにより開始され得る。

フォアグラウンド操作は、直接データファイル・プラットフォームがホストによりアクセスされているのと同時に、直接データファイル・プラットフォームにより開始され得る。

【０２８７】

ガベージコレクション－スケジューリング

バックグラウンドガベージコレクションは、ホストにより開始される。デバイスが内部操作を実行することが許されるアイドル状態は、直接データファイルインターフェイスにおいて特定のコマンドを介してホストにより開始される。アイドル状態が持続している間、ガベージコレクションキューからのオブジェクトのガベージコレクションが続く。いずれかのコマンドがホストから受信されると、ガベージコレクションは保留される。ホストは、ガベージコレクション操作が、次のコマンドを送る前に完了できるようにするために、デバイスのビジー状態を任意に監視できる。

【０２８８】

フォアグラウンドガベージコレクションは、ホストがバックグラウンド操作を開始していない場合に、直接データファイル・プラットフォームにより開始される。ガベージコレクションは、適応アルゴリズムに従ってスケジュールを組み込まれる。現在のガベージコレクション操作のためのプログラムのバーストおよび消去操作は、ホストから受信されたファイルデータのためのプログラム操作のバーストとインターリーブされる。バーストの長さは、インターリーブされたガベージコレクションの動作周期を定義するために適応的に制御され得る。

【０２８９】

ガベージコレクション－適応スケジューリング（図８－２参照）

フラッシュメモリは通常、プログラムブロック、共通ブロック、および古いファイルブロックに含まれるさらなるホストデータを書き込むために必要とされる回復可能な容量を有する。

適応ガベージコレクションは、さらなるホストデータのプログラミングおよび前に書き込まれたホストデータの再配置のインターリーブ比率を制御する。回復可能な容量は、これを消去済容量に変換することにより、新しいホストデータ用に利用可能にされる。ガベージコレクションレートは、適応期間にわたって一定のままである。

【０２９０】

ガベージコレクション－操作のプライオリティ

スケジュールが組み込まれたガベージコレクションのための操作は、以下のプライオリティの順序を有するガベージコレクションキューから選ばれる。

１．古いブロックのプライオリティのガベージコレクション：

ファイル消去コマンドの結果として作成された古いブロックのための次のエントリが選択される。

２．共通ブロックのプライオリティのガベージコレクション：

ファイル消去コマンドの結果として作成された部分的に古い共通ブロックのための次のエントリが選択される。

３．古いブロックのガベージコレクション：

古いブロックのための次のエントリが選択される。

４．共通ブロックのガベージコレクション：

部分的に古い共通ブロックのための次のエントリが選択される。

５．ファイルガベージコレクション：

部分的に古いファイルのための次のエントリが選択される。

６．ブロック統合：

ガベージコレクションキュー中にエントリがまったく存在しない場合、ソースブロックおよび宛先ブロックが、ブロック統合操作のために選択される。

【０２９１】

ガベージコレクション－共通ブロックのガベージコレクション

有効なファイルは、プログラムブロックまたは共通ブロックのいずれかの中にいくつかのデータを含む。

ファイルが削除されるとき、ファイルのための古いデータを含むいずれかの共通ブロックが、共通ブロックのガベージコレクション操作を経験する。

無関係なファイルのためのデータグループが、別の共通ブロックまたはプログラムブロックに再配置される（図８－７Ａ～図８－７Ｄ参照）。

共通ブロックのガベージコレクション操作の間、有効なファイルグループが、ソース共通ブロックから、１つ以上の選択された宛先ブロックに再配置される。

宛先ブロックは、各ファイルグループについて個別に選択される。

宛先ブロックの選択のためのプライオリティは、以下の通りである。

１．再配置されるソースファイルグループにとって最適である利用可能な消去済容量を有する共通ブロック

２．再配置されるソースファイルグループにとって最適である利用可能な消去済容量を有するプログラムブロック

３．次にプログラムブロックと呼ばれる消去済ブロック

【０２９２】

ガベージコレクション－ファイルガベージコレクション

ファイルのための古いデータにより占められた容量を回復するため、ファイルガベージコレクションは、ファイルが閉じられた後、実行され得る。これは、編集の間にファイルのためのデータが上書きされていれば、必要になるだけである。

編集済プレーンファイル状態または編集済共通ファイル状態のファイルは、プレーンファイル状態（単一のプログラムブロックを含み、共通ブロックをまったく含まない）に復元される。

ファイルガベージコレクションは、有効なデータグループを、古いデータを含んでいるブロックからファイルのためのプログラムブロックへコピーすることにより実行される。

データグループは、当初のプログラムポインタに続いてオフセットアドレスから順次コピーされ、ファイルの終わりにラップアラウンドを有する。

【０２９３】

ガベージコレクション－ブロック統合

ブロック統合操作の間に、有効なファイルグループが、１つの選択されたソースブロッ

クから、１つ以上の選択された宛先ブロックに再配置される。
ソースブロックは、データの容量が最も低い共通ブロックまたはプログラムブロックとして選択される。
宛先ブロックは、各ファイルグループについて個別に選択される。
宛先ブロックの選択のためのプライオリティは、以下の通りである。
１．再配置されるソースのファイルグループにとって最適である利用可能な消去済容量を有する共通ブロック
２．再配置されるソースのファイルグループにとって最適である利用可能な消去済容量を有するプログラムブロック
３．ファイルグループの一部が書き込まれる、最高の利用可能な消去済容量を有するプログラムブロックまたは共通ブロック（この状況において、ファイルが、２つのブロックを他の無関係なファイルと共有することが許される。）
４．ファイルグループの残りが書き込まれる、ソースのファイルグループの残りにとって最適である利用可能な消去済容量を有する第２のプログラムブロックまたは共通ブロック
５．次にプログラムブロックと呼ばれ、ファイルグループの残りが書き込まれる消去済ブロック
【０２９４】
ファイルインデックス化（図１０－１参照）
ファイルがホストにより作成されるときに直接データファイルデバイスにより割り当てられるfileIDにより、ファイルが識別される。
フラットディレクトリが、各fileIDについてファイルデータポインタおよびファイル情報ポインタを指定する。
ファイルデータポインタは、ファイルインデックステーブル中のエントリのセットを識別し、各エントリは、そのセットが関連するファイルのためのデータグループを指定する。
ファイル情報ポインタは、情報テーブル中のファイルのための情報ストリングを識別する：
file_infoは、ホストにより書き込まれ、直接データファイルデバイスによって解釈されない。
file__infoは、ファイル名、親ディレクトリ、子ディレクトリ、属性、権利情報、およびファイルのためのファイル関連付けを含み得る。
【０２９５】
ファイルインデックス化－インデックス化構造
図１０－２参照。
【０２９６】
ファイルインデックス化－ファイルインデックステーブル（ＦＩＴ）－図１０－４参照
ＦＩＴは、フラッシュメモリ中のファイルのためのすべての有効なデータグループのためのエントリを含む。古いデータグループは、ＦＩＴによりインデックスを付けられない。
ＦＩＴは、各々が物理ブロックにマッピングされる論理範囲に分割される。
ＦＩＴファイルは、ファイルオフセットアドレス順の、ファイルのための連続したエントリのセットである。
ＦＩＴファイルは、物理ブロックおよび論理ファイル数を定義する、ＦＩＴファイルポインタにより識別される。
【０２９７】
ファイルインデックス化－ファイルインデックスの更新（図１０－６および図１０－７参照）
ファイルインデックステーブルおよび情報テーブルについて同じ構造が用いられる。
ブロックリストは、論理ファイルデータポインタを、物理ＦＩＴブロック内またはＦＩ

Ｔ更新ブロック内をＦＩＴファイルと関係付けるために用いられる。
ＦＩＴファイルは、圧縮形式でＦＩＴブロックに格納される。
ＦＩＴファイルの更新されたバージョンは、共有されたＦＩＴ更新ブロック中に、１ページ中に単一のＦＩＴファイルで、格納される。
ＦＩＴ更新ブロックの圧縮およびＦＩＴブロック中のＦＩＴファイルの統合は、その時々に実行される。

【０２９８】

ファイルインデックス化－インデックスページ形式（図１０－５参照）

ＦＩＴブロック、ＦＩＴ更新ブロック、情報ブロック、および情報更新ブロックのために、同じ構造が用いられる。
情報は、１ページ単位でプログラムされる。
１つのページが、ＦＩＴエントリおよびファイルポインタのために２つの領域に再分割される。
ファイルポインタは、範囲内の論理ファイル番号を、対応するＦＩＴファイルの開始のためのページ番号およびエントリ番号に変換する。
ＦＩＴファイルは、物理的に連続したＦＩＴエントリを含む。

【０２９９】

データバッファリングおよびプログラミング

ホストによりフラッシュメモリ内に書き込みまたは再配置されるデータは、セクタバッファのセット中にバッファされる。
データグループ境界の分解能は１バイトであるが、データは、ＥＣＣ生成およびチェックのために、フラッシュへまたはフラッシュから１セクタの整数倍で転送される。
バッファからのデータは、可能であれば、メタページ単位でフラッシュにおいてプログラムされる。
バッファ消去操作は、ファイルが閉じられているまたはシャットダウンが保留中である場合に、１つのページの一部のみをプログラムする。ファイルインデックス化技術は、ページのプログラムされていない部分が残ることを可能にする。
バッファスワップアウト操作は、バッファ空間の管理およびバッファ中のデータのバックアップのために、バッファ中のファイルデータが、共通スワップブロック中に一時的に格納されることを可能にする。
プログラムブロックまたは共通ブロック中のファイルグループの開始は、メタページの開始と整合される。
オンチップコピーは、フラッシュにおけるほとんどのデータ再配置に用い得る。

【０３００】

ブロック状態管理

直接データファイルシステムは、データ記憶と関連付けられたブロックのための８つの状態を維持する（図１１－１参照）。

【０３０１】

消去済ブロック管理

直接データファイルは、ファイルのためのすべてのデータおよびすべての制御情報を、固定サイズのメタブロック中に格納する。（用語「ブロック」は、「メタブロック」を指定するためによく用いられる）。
消去ブロックをブロックにリンクする方法は、以下の継続中の米国特許出願において記載される論理アドレス空間（ＬＢＡ）インターフェイスを有するシステムにおいて用いられる方法から変わっていない。継続中の米国特許出願とは、２００３年１２月３０日に出願された「Management of Non-Volatile Memory Systems Having Large Erase Blocks 」という米国特許出願第１０／７４９，８３１号（特許文献１３）、２００３年１２月３０日に出願された「Non-Volatile Memory and Method with Block Management System 」という米国特許出願第１０／７５０，１５５号（特許文献１４）、２００４年８月１３日に出願された「Non-Volatile Memory and Method with Memory Planes Alignment 」という

米国特許出願第１０／９１７，８８８号（特許文献１５）、２００４年8月１３日に出願された米国特許出願第１０／９１７，８６７号（特許文献１６）、２００４年8月１３日に出願された「Non-Volatile Memory and Method with Phased Program Failure Handling 」という米国特許出願第１０／９１７，８８９号（特許文献１７）、２００４年8月１３日に出願された「Non-Volatile Memory and Method with Control Data Management 」という米国特許出願第１０／９１７，７２５号（特許文献１８）、２００５年7月２７日に出願された「Non-Volatile Memory and Method with Multi-Stream Update Tracking」という米国特許出願第１１／１９２，２００号（特許文献１９）、２００５年7月２７日に出願された「Non-Volatile Memory and Method with Improved Indexing for Scratch Pad and Update Blocks 」という米国特許出願第１１／１９２，３８６号（特許文献２０）、および２００５年7月２７日に出願された「Non-Volatile Memory and Method with Multi-Stream Updating 」という米国特許出願第１１／１９１，６８６号（特許文献２１）である。

データまたは制御情報を格納するための割り当てに利用可能な消去済ブロックは、消去済ブロックプール中に維持される。

消去済ブロックは、消去済ブロックログ中のエントリとして記録される。

割り当てのための消去済ブロックは、ログの頭におけるエントリとして選択される。

ブロックが消去されるときに、ログの末尾にエントリが加えられる。

【０３０２】

制御データ構造

制御データ構造は、専用制御ブロック中に格納される。

制御情報は、４つの独立したログ中に格納される。各ログは、制御ブロック中の１ページ以上を占める。有効なログページは、書き込まれた最後のページのログポインタにより追跡される。

共通ブロックログは、フラッシュメモリ中に存在するすべての共通ブロックのためのエントリを、共通ブロックが含む利用可能な消去済容量順で含む。

プログラムブロックログは、フラッシュメモリ中に存在するすべてのプログラムブロックのためのエントリを、プログラムブロックが含む利用可能な消去済容量順で含む。

消去済ブロックログは、フラッシュメモリ中に存在するすべての消去済ブロックのためのエントリを、消去済ブロックの消去順で含む。

制御ログは、制御パラメータ、カウント、およびリストのためのあらかじめ定義されたフィールドを含む。

ログは、制御ブロック中の次の消去済ページ位置に完全なログの改訂バージョンを書き込むことにより更新される。

【図面の簡単な説明】

【０３０３】

【図１－１】直接データファイル・プラットフォームを有するメモリカードを示す。

【図１－２】直接データファイル・プラットフォームの構成要素を示す。

【図２－１】ファイルコマンドを示す。

【図２－２】データコマンドを示す。

【図２－３】情報コマンドを示す。

【図２－４】ストリームコマンドを示す。

【図２－５】状態コマンドを示す。

【図２－６】デバイスコマンドを示す。

【図３－１】プレーンファイルの形式を示す。

【図３－２】共通ファイルの形式を示す。

【図３－３】編集済プレーンファイルの形式を示す。

【図３－４】編集済共通ファイルの形式を示す。

【図４－１】デバイス操作のための流れ図を示す。

【図５－１】ファイルデータのプログラミングのための流れ図を示す。

【図６－１】ファイルデータの読み出しのための流れ図を示す。
【図７－１】ファイル削除のための流れ図を示す。
【図８－１】フォアグラウンドガベージコレクションのためのインターリーブされた操作を示す。
【図８－２】ガベージコレクションの適応スケジューリングのための操作原理を示す。
【図８－３】ガベージコレクション選択のための流れ図を示す。
【図８－４】ファイルガベージコレクションのための流れ図を示す。
【図８－５】共通ブロックのガベージコレクションのための流れ図を示す。
【図８－６】ブロック統合のための流れ図を示す。
【図８－７Ａ】４つの時系列段階を示す、共通ブロックのガベージコレクションの例を示す。
【図８－７Ｂ】４つの時系列段階を示す、共通ブロックのガベージコレクションの例を示す。
【図８－７Ｃ】４つの時系列段階を示す、共通ブロックのガベージコレクションの例を示す。
【図８－７Ｄ】４つの時系列段階を示す、共通ブロックのガベージコレクションの例を示す。
【図９－１】連続したホストデータプログラミングを示す。
【図９－２】中断されたホストデータプログラミングを示す。
【図９－３】バッファ消去プログラミングを示す。
【図９－４】バッファスワップアウトプログラミングを示す。
【図９－５】バッファ消去後のホストデータプログラミングを示す。
【図９－６】スワップインデータ読み出しを示す。
【図９－７】バッファスワップイン後のホストデータプログラミングを示す。
【図９－８】バッファへの整合データ読み取出しを示す。
【図９－９】バッファからの整合データプログラミングを示す。
【図９－１０】バッファへの不整合データ読み出しを示す。
【図９－１１】バッファからの不整合データプログラミングを示す。
【図９－１２】バッファへの不整合非逐次データ読み出しを示す。
【図９－１３】バッファからの不整合非逐次データプログラミングを示す。
【図１０－１】ファイルインデックス化を示す。
【図１０－２】ファイルインデックス化構造を示す。
【図１０－３】ディレクトリブロック形式を示す。
【図１０－４】ファイルインデックステーブル（ＦＩＴ）論理構造を示す。
【図１０－５】ＦＩＴページ形式を示す。
【図１０－６】物理ＦＩＴブロックを示す。
【図１０－７】ＦＩＴファイル更新操作の例を示す。
【図１１－１】ブロック状態図を示す。
【図１２－１】制御ブロック形式を示す。
【図１２－２】共通ブロックログ形式を示す。
【図１３－１】（一緒にされるべきパートＡおよびパートＢにおいて）静的ファイルと共に用いられるコマンドセットを示す。

【図１－１】

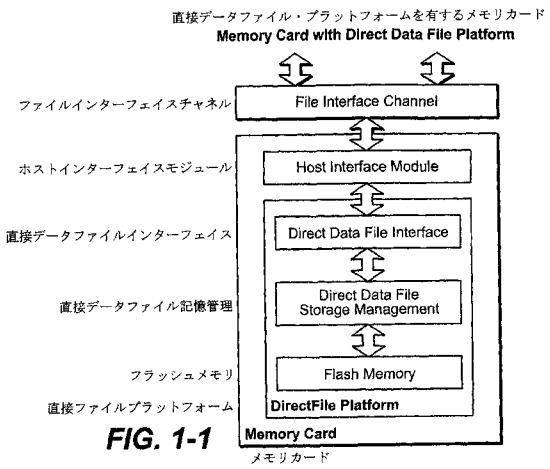


FIG. 1-1

【図１－２】

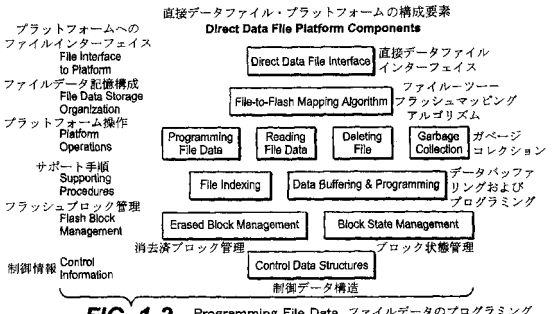


FIG. 1-2

Programming File Data ファイルデータのプログラミング
Reading File Data ファイルデータの読み出し
Deleting File ファイルの削除
File Indexing ファイルインデックス化

【図２－１】

ファイルコマンド

コマンド	パラメータ	説明
Create	＜fileID＞	デバイス中のディレクトリ中に、＜fileID＞により識別されたエントリを作成する。＜fileID＞がコマンドで指定されなければ、このパラメータはデバイスにより割り当てられ、ホストに返される。
Open	＜fileID＞	指定されたファイルのためのその後のデータコマンドの実行を可能にする。
Close	＜fileID＞	指定されたファイルのためのその後のデータコマンドの実行をできなくする。
Delete	＜fileID＞	指定されたファイルについてのディレクトリ、ファイルインデックス、およびファイル情報エントリが削除されるべきであることを示す。ファイルデータは、消去され得る。
Erase	＜fileID＞	指定されたファイルについてのディレクトリ、ファイルインデックス、およびファイル情報エントリが削除されるべきであり、ファイルデータが直ちに消去されるべきであることを示す。
List_files		すべての有効な＜fileID＞値をデバイスから、番号順に読み出す。

File Commands

Command	Parameters	Description
Create	<fileID>	Creates an entry identified by <fileID> in the directory in the device. If <fileID> is not specified with the command, it is assigned by the device and is returned to the host.
Open	<fileID>	Enables execution of subsequent data commands for the specified file.
Close	<fileID>	Disables execution of subsequent data commands for the specified file.
Delete	<fileID>	Indicates that directory, file index, and file info entries for the specified file should be deleted. File data may be erased.
Erase	<fileID>	Indicates that directory, file index, and file info entries for the specified file should be deleted and that file data should be erased immediately.
List_files		Reads all valid <fileID> values from the device, in numerical order.

FIG. 2-1

【図２－２】

データコマンド

コマンド	パラメータ	説明
Write	＜fileID＞	データを、write_pointer の現在値により定義されたオフセットアドレスにおいて上書きする。
Insert	＜fileID＞	データを、write_pointer の現在値により定義されたオフセットアドレスにおいて、指定されたファイルに挿入する。
Remove	＜fileID＞＜length＞	write_pointer の現在値により定義されたオフセットアドレスにおいて、指定されたファイルから、サイズ＜length＞のデータを削除する。
Read	＜fileID＞	read_pointer の現在値により定義されたオフセットアドレスにおいて指定されたファイルからデータを読み出す。
Save_buffer	＜fileID＞	指定されたファイルについてのバッファ中のデータを、フラッシュメモリ中の一時的な位置に保存する。
Write_pointer	＜fileID＞＜offset＞	指定されたファイルについての write_pointer のための新しい現在値を定義する。
Read_pointer	＜fileID＞＜offset＞	指定されたファイルについての read_pointer のための新しい現在値を定義する。

Data Commands

Command	Parameters	Description
Write	<fileID>	Overwrites data in specified file at offset address defined by current value of write_pointer.
Insert	<fileID>	Inserts data in specified file at offset address defined by current value of write_pointer.
Remove	<fileID> <length>	Deletes data of size <length> from specified file at offset address defined by current value of write_pointer.
Read	<fileID>	Reads data from specified file at offset address defined by current value of read_pointer.
Save_buffer	<fileID>	Saves data in buffer for specified file to a temporary location in flash memory.
Write_pointer	<fileID> <offset>	Defines new current value for write_pointer for specified file.
Read_pointer	<fileID> <offset>	Defines new current value for read_pointer for specified file.

FIG. 2-2

【図２－３】

情報コマンド

コマンド	パラメータ	説明
Write_info	<fileID>	info_write_pointer の現在値により定義されたオフセットアドレスにおいて、指定されたファイルについての file_info を書き込む。
Read_info	<fileID>	info_read_pointer の現在値により定義されたオフセットアドレスにおいて、指定されたファイルについての file_info を読み出す。
Info_write_pointer	<fileID><offset>	指定されたファイルについての info_write_pointer のための新たな現在値を定義する。
Info_read_pointer	<fileID><offset>	指定されたファイルについての info_read_pointer のための新たな現在値を定義する。

Info Commands

Command	Parameters	Description
Write_info	<fileID>	Writes file_info for specified file at offset address defined by current value of info_write_pointer.
Read_info	<fileID>	Reads file_info for specified file at offset address defined by current value of info_read_pointer.
Info_write_pointer	<fileID> <offset>	Defines new current value for info_write_pointer for specified file.
Info_read_pointer	<fileID> <offset>	Defines new current value for info_read_pointer for specified file.

FIG. 2-3

【図２－４】

ストリームコマンド

コマンド	パラメータ	説明
Stream	<length>	デバイスへまたはデバイスから転送されるデータの中断されないストリームを定義する。
Pause	<time>	以降のコマンドの実行前に挿入される遅延を定義する。

Stream Commands

Command	Parameters	Description
Stream	<length>	Defines an uninterrupted stream of data that will be transferred to or from the device.
Pause	<time>	Defines a delay that is inserted before execution of the following command.

FIG. 2-4

【図２－５】

状態コマンド

コマンド	パラメータ	説明
Idle		デバイスはアイドル状態に入り、この状態の間、デバイスは内部操作を実行し得る。
Standby		デバイスは待機状態に入り、この状態の間、デバイスは内部操作を実行することができない。
Shut-down		デバイスはシャットダウンされ、デバイスが次にビジー状態にならなければ、電力が取り去られる。

State Commands

Command	Parameters	Description
Idle		The device should enter an idle state, within which it may perform internal operations.
Standby		The device should enter a standby state, within which it may not perform internal operations.
Shut-down		The device will be shut down and power will be removed when it is next not in a busy state.

FIG. 2-5

【図２－６】

デバイスコマンド

コマンド	パラメータ	説明
Capacity		ファイルデータにより占められる容量および新しいファイルデータが利用可能な容量の報告をデバイスに求めるホストからの要求。
Status		デバイスの現在のステータスの報告をデバイスに求めるホストからの要求。

Device Commands

Command	Parameters	Description
Capacity		Request from host for the device to report capacities occupied by file data and available for new file data.
Status		Request from host for the device to report its current status.

FIG. 2-6

【図３－１】

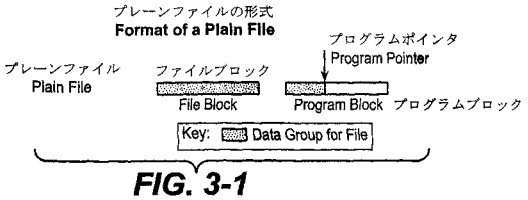


FIG. 3-1

凡例：ファイルのためのデータグループ

【図３－２】

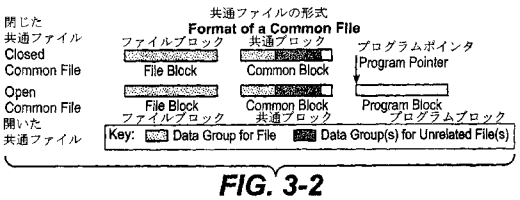


FIG. 3-2

凡例：ファイルのためのデータグループ　無関係なファイルのためのデータグループ

【図３－３】

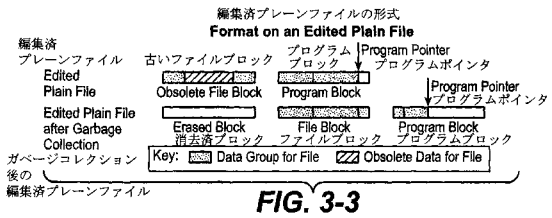


FIG. 3-3

凡例：ファイルのためのデータグループ　ファイルのための古いデータ

【図３－４】

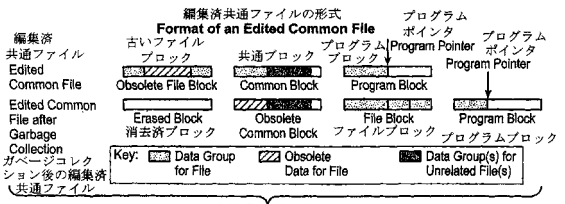


FIG. 3-4

凡例：ファイルのためのデータグループ　ファイルのための古いデータ　無関係なファイルのためのデータグループ

Obsolete Common Block　古い共通ブロック

【図４－１】

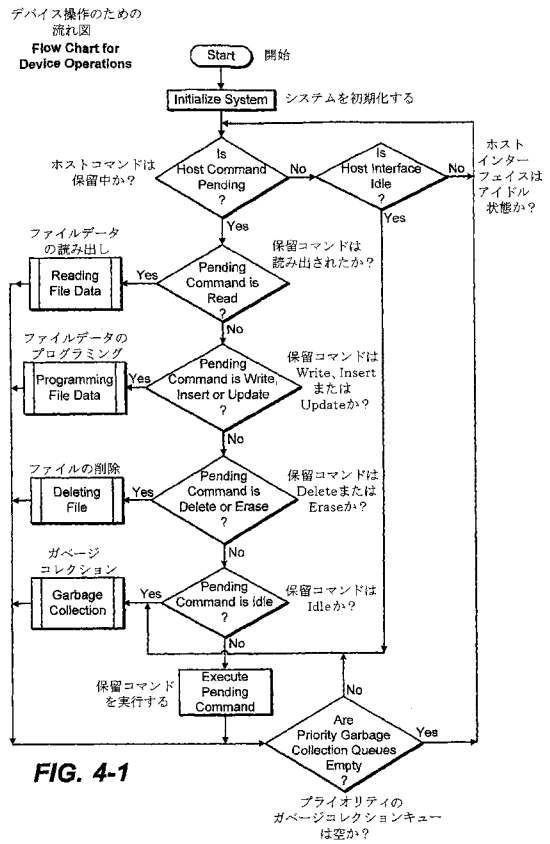


FIG. 4-1

【図５－１】

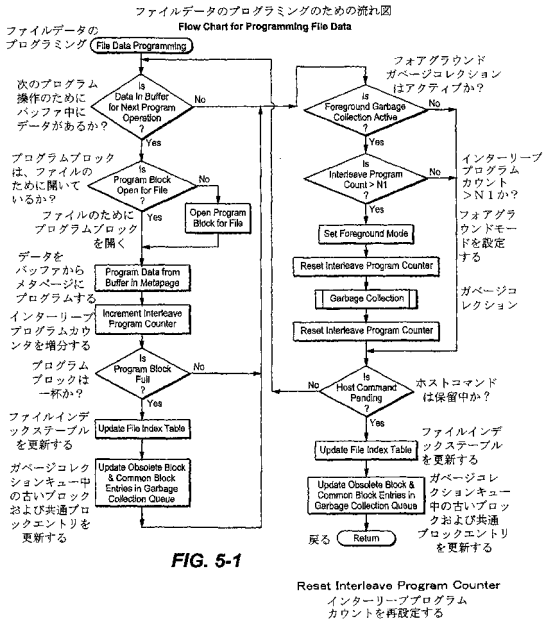


FIG. 5-1

Reset Interleave Program Counter
インターリーブプログラム
カウントを再設定する

【図６－１】

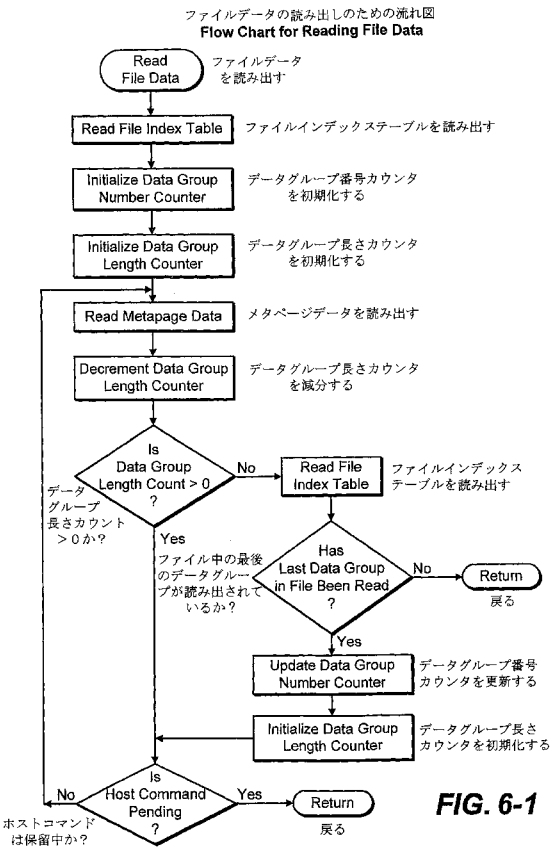


FIG. 6-1

【図７－１】

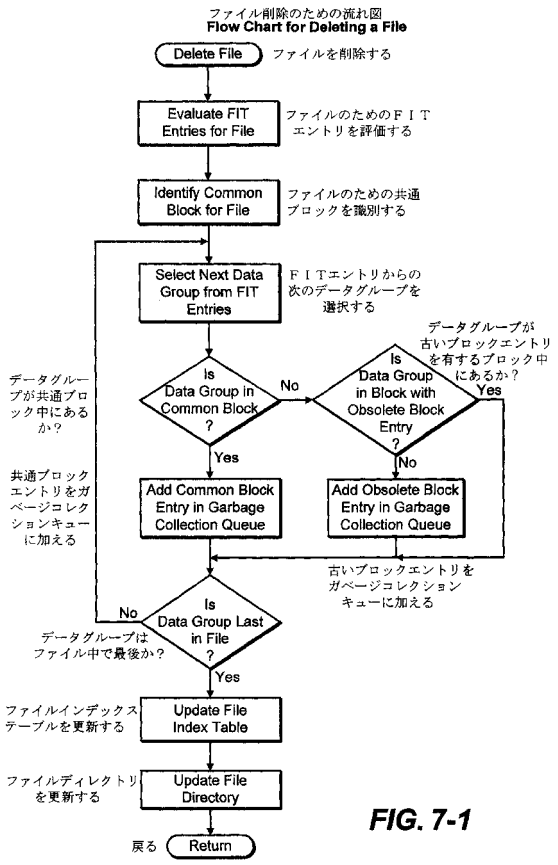


FIG. 7-1

【図８－１】

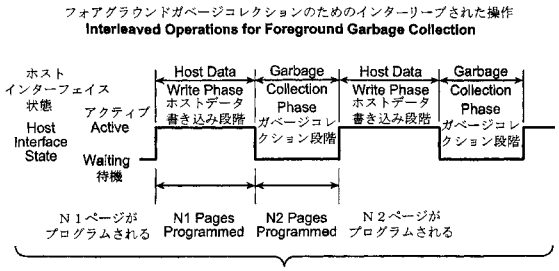


FIG. 8-1

【図８－２】

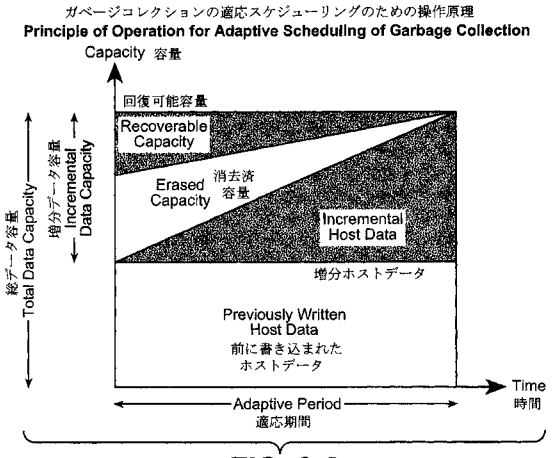


FIG. 8-2

【図８－３】

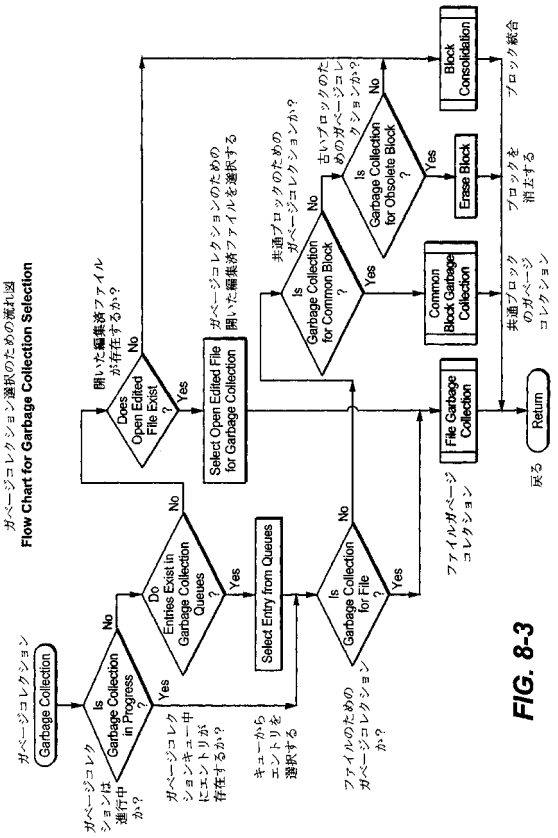


FIG. 8-3

【図８－４】

FIG. 8-4

FIG. 8-4A FIG. 8-4B

【図８－４Ａ】

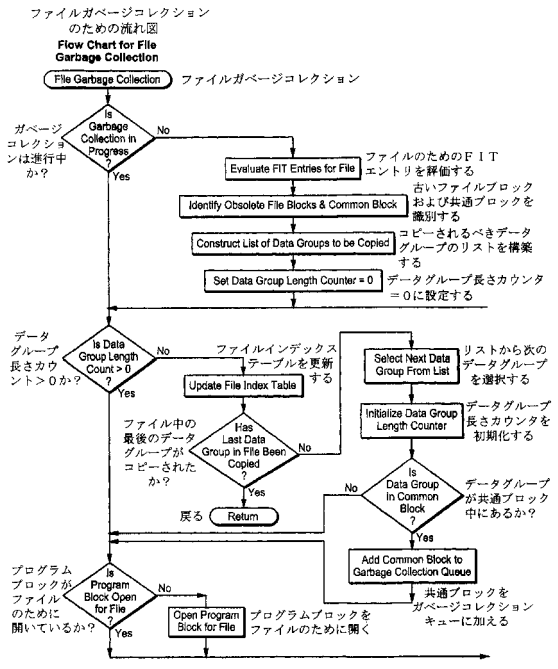


FIG. 8-4A

【図８－４Ｂ】

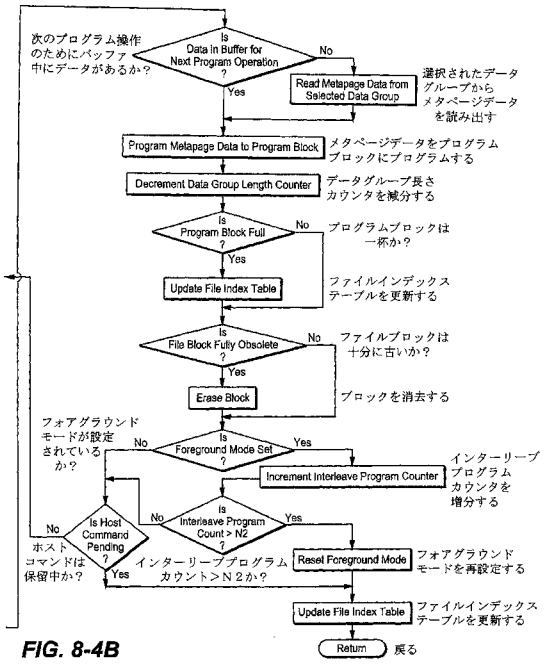


FIG. 8-4B

【図８－５】

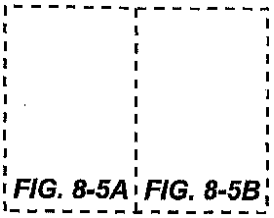


FIG. 8-5

【図８－５Ａ】

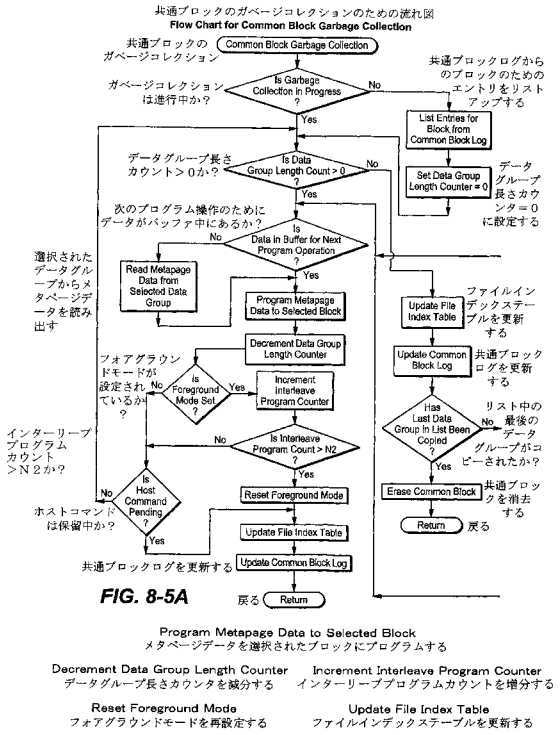


FIG. 8-5A

Program Metapage Data to Selected Block
メタページデータを選択されたブロックにプログラムする

Decrement Data Group Length Counter
データグループ長さカウンタを減分する

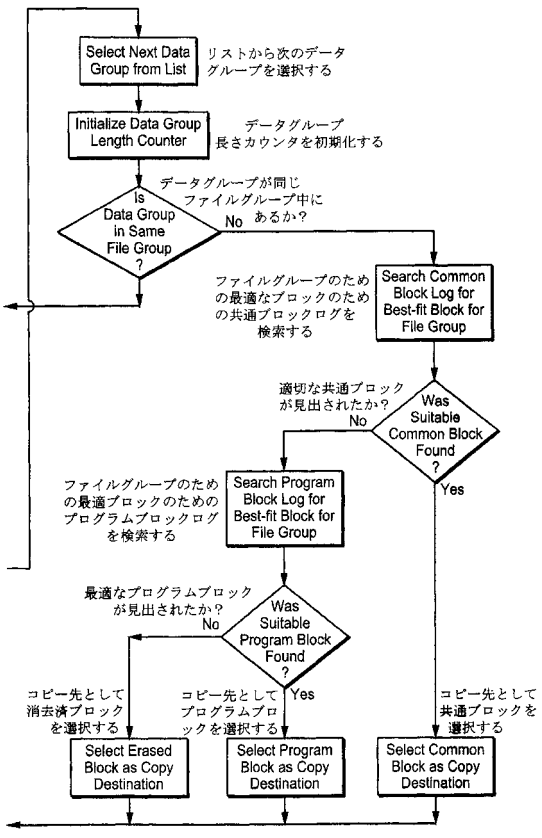
Increment Interleave Program Counter
インターリーブプログラムカウントを増分する

Reset Foreground Mode
フォアグラウンドモードを再設定する

Update File Index Table
ファイルインデックステーブルを更新する

【図８－５Ｂ】

FIG. 8-5B



【図８－６】

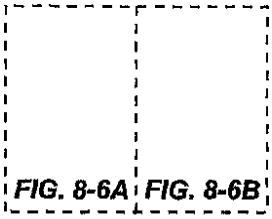


FIG. 8-6

【図８－６Ａ】

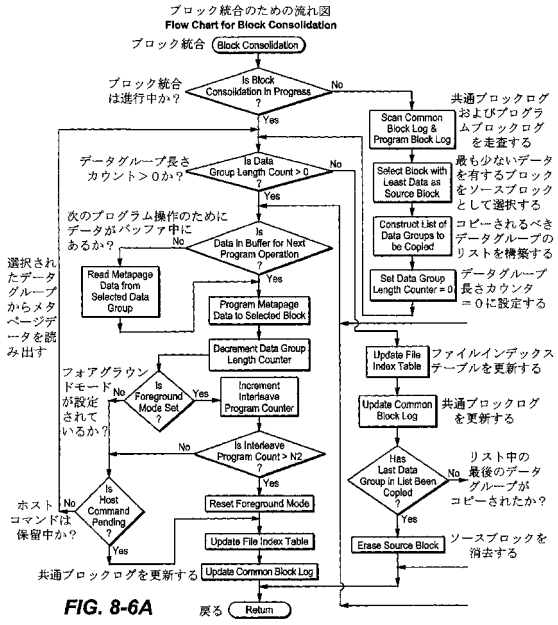


FIG. 8-6A

Program Metapage Data to Selected Block
選択されたブロックにメタページデータをプログラムする

Decrement Data Group Length Counter
データグループ長さカウンタを減分する

Increment Interleave Program Counter
インターリーブプログラムカウンタを増分する

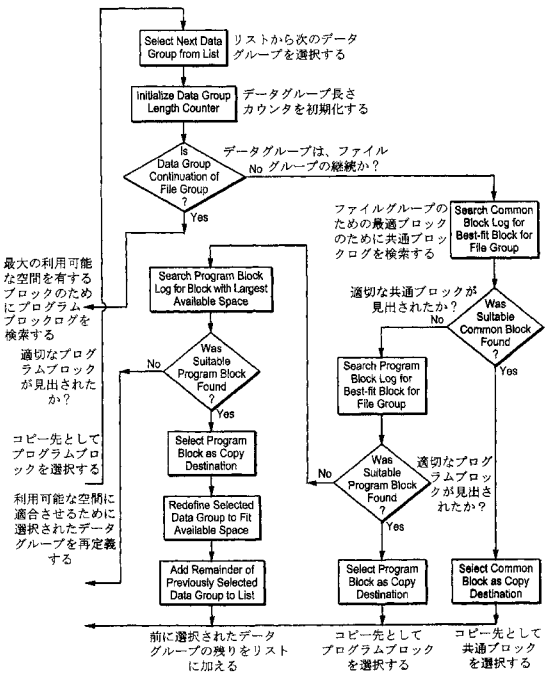
Is Interleave Program Count>N2?
インターリーブプログラムカウント＞Ｎ２か？

Reset Foreground Mode
フォアグラウンドモードを再設定する

Update File Index Table
ファイルインデックステーブルを更新する

【図８－６Ｂ】

FIG. 8-6B



Search Program Block Log for Best-fit Block for File Group
ファイルグループのための最適ブロックのためにプログラムブロックログを検索する

【図８－７Ａ】

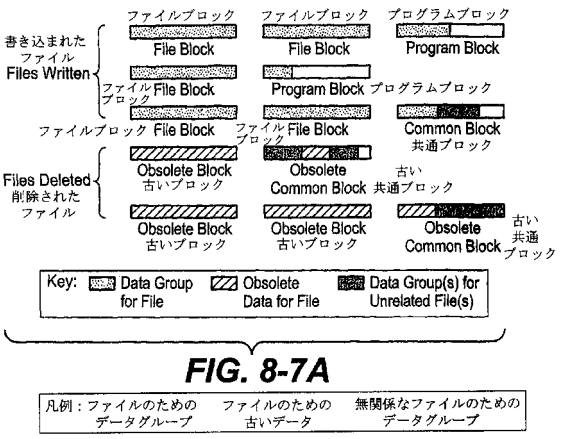


FIG. 8-7A

凡例：ファイルのためのデータグループ	ファイルのための古いデータ	無関係なファイルのためのデータグループ

【図８－７Ｂ】

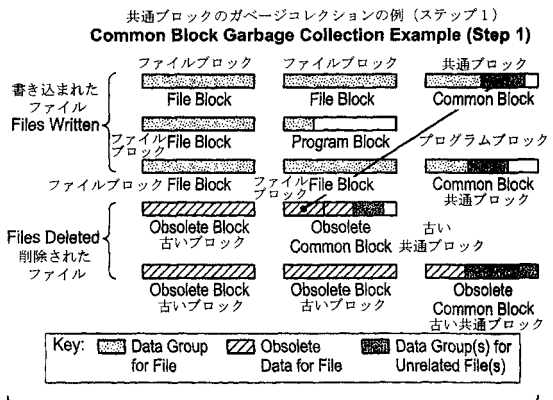


FIG. 8-7B

凡例：ファイルのためのデータグループ　ファイルのための古いデータ　無関係なファイルのためのデータグループ

【図８－７Ｃ】

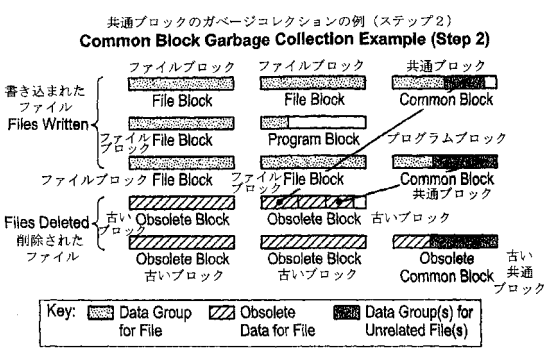


FIG. 8-7C

凡例：ファイルのためのデータグループ　ファイルのための古いデータ　無関係なファイルのためのデータグループ

【図８－７Ｄ】

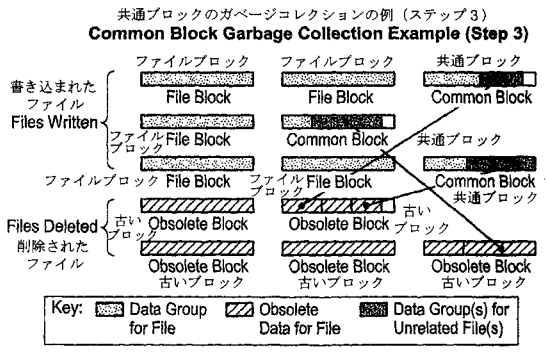


FIG. 8-7D

凡例：ファイルのためのデータグループ　ファイルのための古いデータ　無関係なファイルのためのデータグループ

【図９－１】

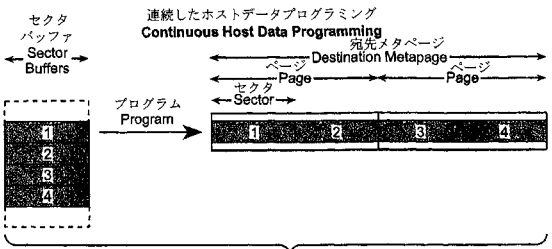


FIG. 9-1

【図９－２】

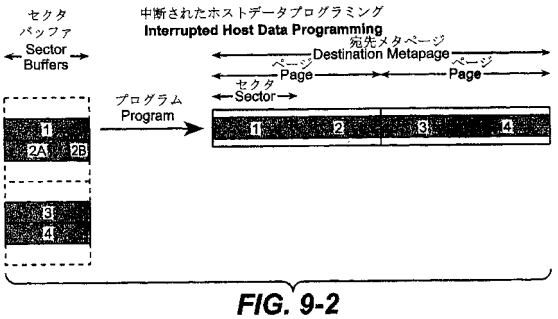


FIG. 9-2

【図９－３】

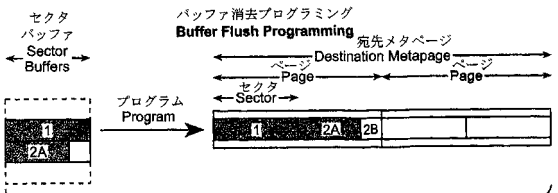


FIG. 9-3

【図９－４】

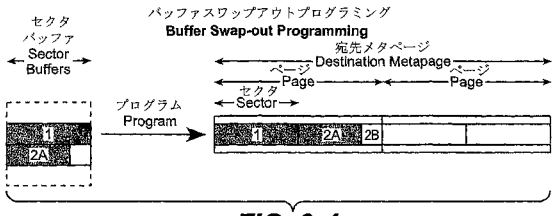


FIG. 9-4

【図９－５】

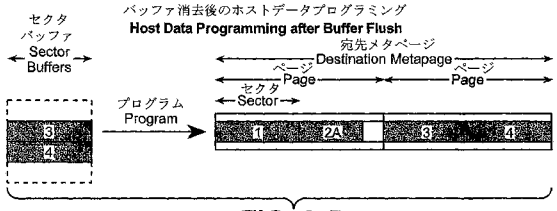


FIG. 9-5

【図９－６】

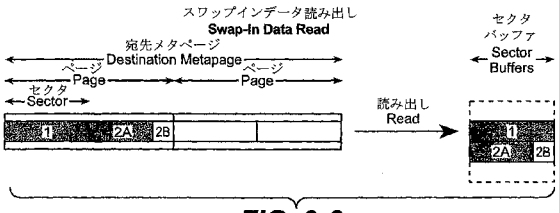


FIG. 9-6

【図９－７】

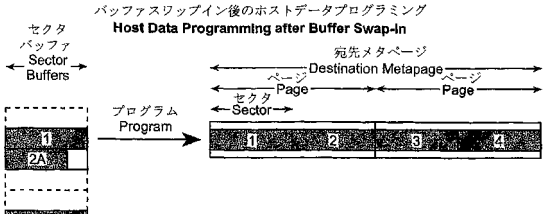


FIG. 9-7

【図９－８】

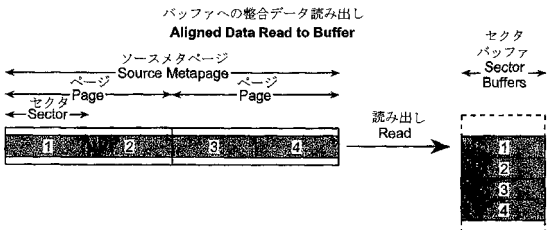


FIG. 9-8

【図９－９】

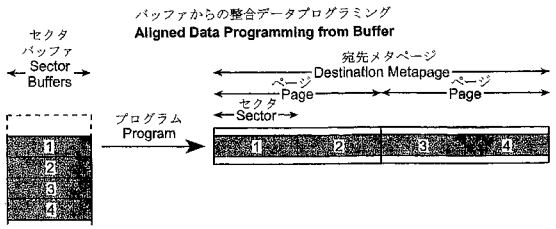


FIG. 9-9

【図９－１０】

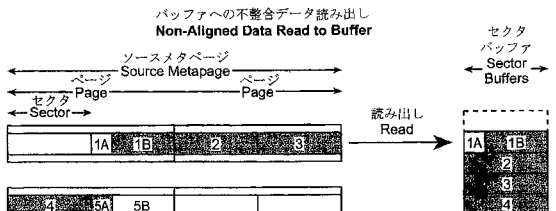


FIG. 9-10

【図９－１１】

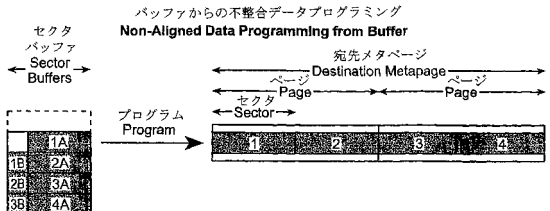


FIG. 9-11

【図９－１２】

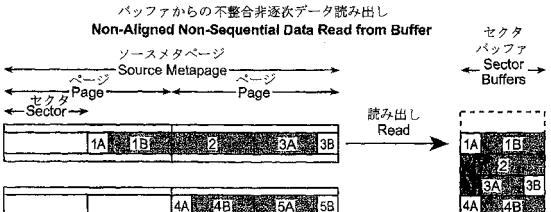


FIG. 9-12

【図９－１３】

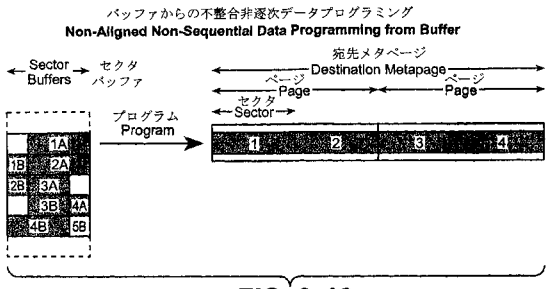


FIG. 9-13

【図１０－１】

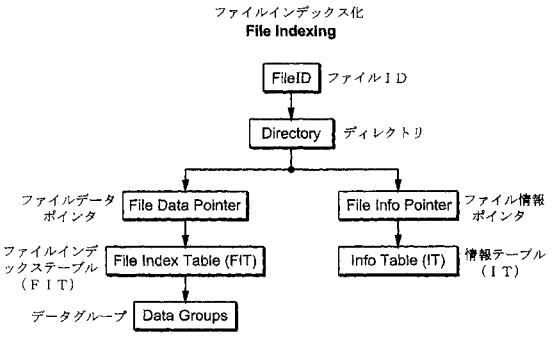


FIG. 10-1

【図１０－２】

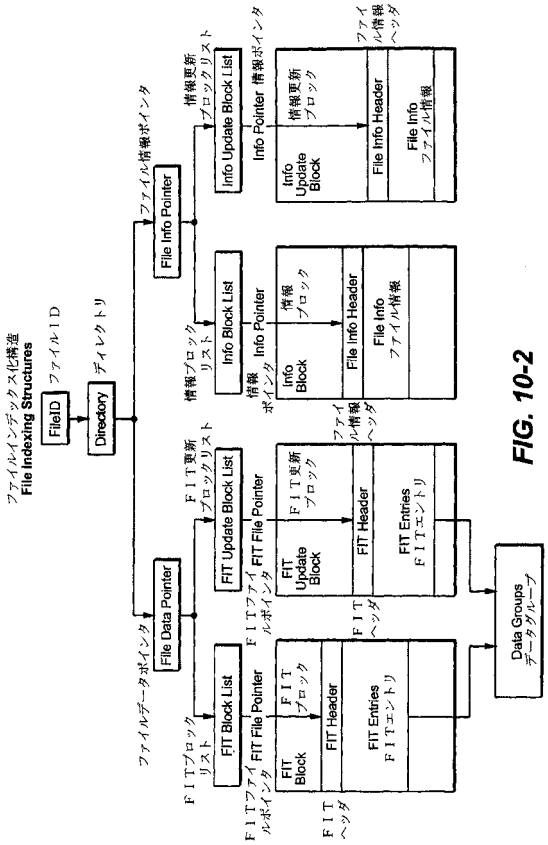


FIG. 10-2

【図１０－３】

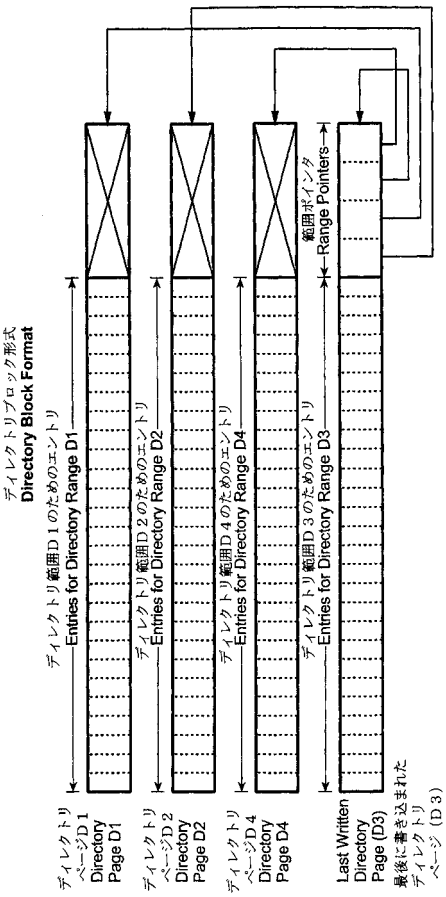


FIG. 10-3

【図１０－４】

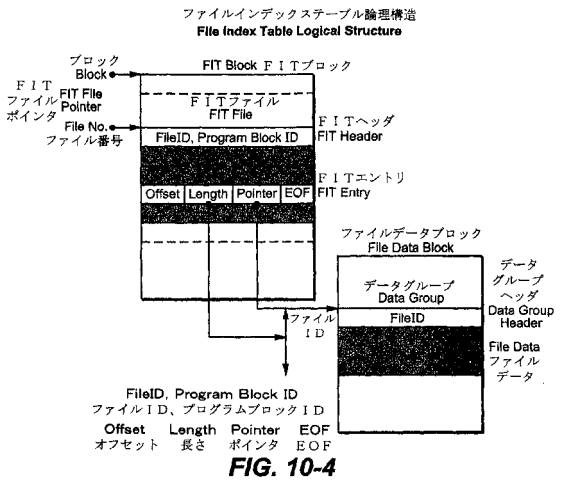


FIG. 10-4

【図１０－５】

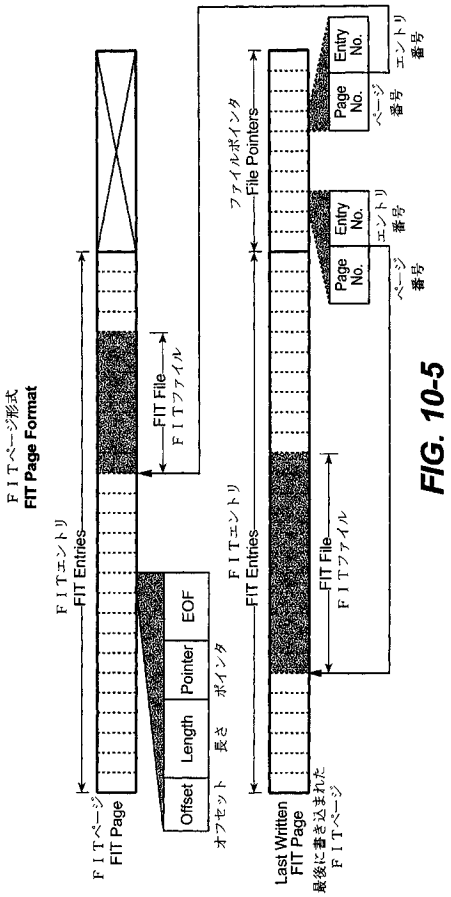


FIG. 10-5

【図１０－６】

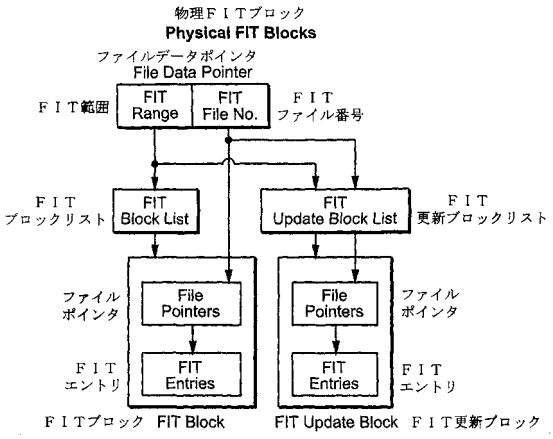


FIG. 10-6

【図１０－７】

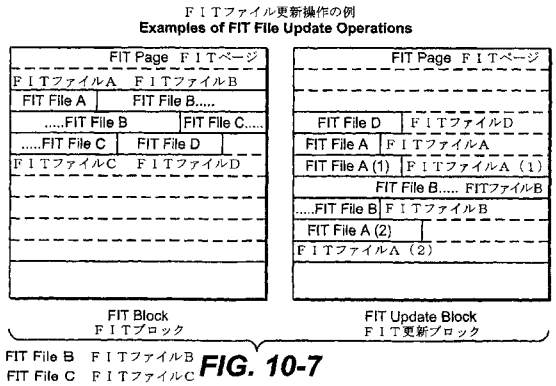


FIG. 10-7

【図１１－１】

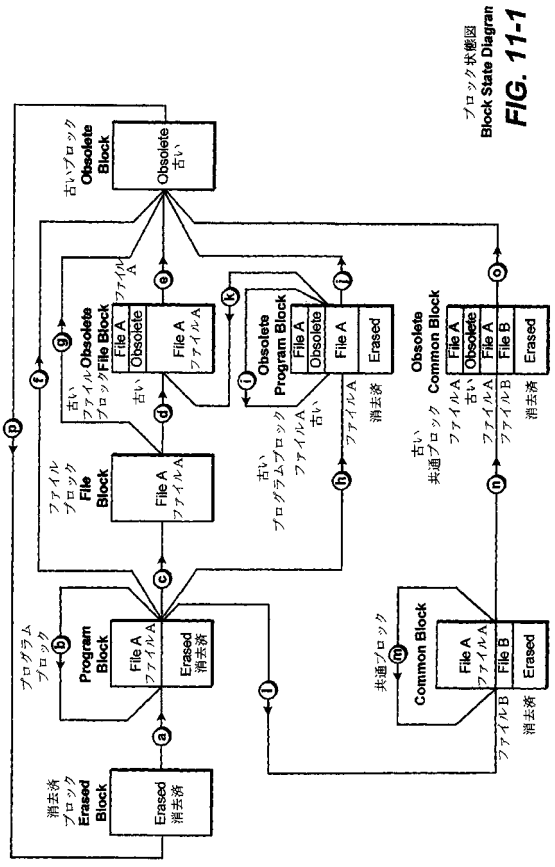


FIG. 11-1

【図１２－１】

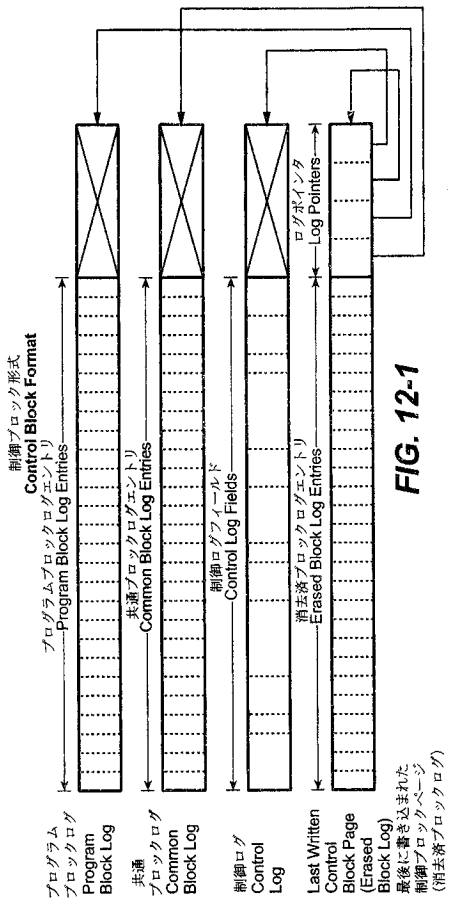


FIG. 12-1

【図１２－２】

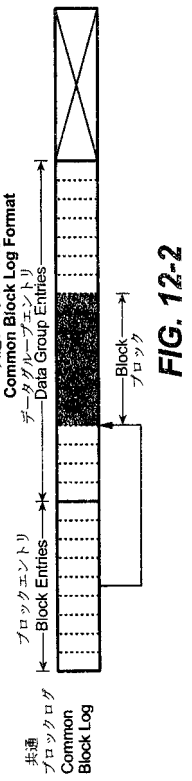


FIG. 12-2

【図１３－１】

FIG. 13-1

FIG. 13-1A

FIG. 13-1B

【図１３－１Ａ】

FIG. 13-1A

Command Set Used with Static Files (Part A)

Command	Parameters	Description
Create	<fileID>	Creates an entry identified by <fileID> in the directory in the device. <fileID> should be specified with the command, to denote an appropriate numerical identifier for the static file. The Create command may be used after the direct data file has been deleted; an error status is returned by a Create command for a file that does not exist in the directory in the direct data file device.
Open	<fileID>	Enables execution of subsequent data commands for the specified file. The write_pointer is set to the end of the file and the read_pointer is set to the beginning of the file. If the file does not exist, an error status is returned.
Close	<fileID>	Disables execution of subsequent data commands for the specified file. The Close command for a static file causes an entry to be inserted into either the obsolete block garbage collection queue or the file garbage collection queue.
Delete	<fileID>	The Delete command is not normally used with static files. However, it may be used if the size of the device partition assigned for static files is reduced.
List_files		Reads all valid <fileID> values from the device, in numerical order.
Write	<fileID>	Overwrites data in specified static file at offset address defined by *current value of write_pointer*. The offset address should not be outside the range corresponding to the file size used by the host for all static files.
Read	<fileID>	Reads data from specified file at offset address defined by *current value of read_pointer*. The offset address should not be outside the range corresponding to the file size used by the host for all static files.

【図１３－１Ａ】

静的ファイルと共に用いられるコマンドセット（パート A）

コマンド	パラメータ	説明
Create	<fileID>	デバイス中のディレクトリ内に、<fileID>により識別されたエントリを作成する。 静的ファイルのための適切な数値識別子をセットするために、<fileID>は、このコマンドで指定されるべきである。 create コマンドは、直接データファイルデバイス中のディレクトリ中に存在しないファイルのための open コマンドに続いて直接データファイルデバイスにエラーステータスを返すために用いられる。
Open	<fileID>	指定されたファイルのためのその後のデータコマンドの実行を可能にする。 write_pointer は、ファイルの終わりに設定され、read_pointer は、ファイルの最初に設定される。 ファイルが存在しなければ、エラーステータスが返される。
Close	<fileID>	指定されたファイルのためのその後のデータコマンドの実行をなくする。 静的ファイルのための close コマンドは、エントリが、古いオブソレッセンスキューまたはファイルガベージコレクションキューのいずれかに挿入されるようにする。
Delete	<fileID>	delete コマンドは通常、静的ファイルと共に用いられることはない。しかし、静的ファイルのために割り当てられたデバイスのパーティションサイズが減りさられれば、このコマンドが用いられることもある。
List_files		デバイスからのすべての有効な<fileID>値を、番号順に読み出す。
Write	<fileID>	指定された静止ファイル中のデータを、write_pointer の *現在値* により定義されたオフセットアドレスに書き込む。 オフセットアドレスは、すべての静的ファイルのために本ホストにより用いられるファイルサイズに対応する範囲内にあってはならない。
Read	<fileID>	指定されたファイルから、指定されたオフセットアドレスにおいて指定されたファイルからデータを読み出す。 read_pointer の現在値により定義されたオフセットアドレスは、すべての静的ファイルのために本ホストにより用いられるファイルサイズに対応する範囲内にあってはならない。

【図１３－１Ｂ】

静的ファイルと共に用いられるコマンドセット（パートB）

コマンド	パラメータ	説明
Save_buffer	<fileID>	指定された静的ファイルについてのバッファ中のデータを、フラッシュメモリ中の一時的な位置に保存する。 save_buffer は、静的ファイルに用いられるプロトコルが、ファイルのために供給されたデータがフラッシュメモリにコミットされなければならないことを要求する場合に、用いられる。
write_pointer	<fileID><offset>	指定されたファイルについての write_pointer のための新しい現在値を定義する。 write_pointer は、ファイルが開かれた後の最初のデータのため、または前に書き込まれたデータと連続していないデータのために、書き込まれるべきデータの静的ファイル内のオフセットを定義するために用いられる。 オフセットは、すべての静的ファイルのためにホストにより用いられるファイルサイズに対応する範囲外にあってはならない。
Read_pointer	<fileID><offset>	指定されたファイルについての read_pointer のための新しい現在値を定義する。 read_pointer は、ファイルが開かれた後の最初のデータのため、または前に書き込まれたデータと連続していないデータのために、読み出されるべきデータの静的ファイル内のオフセットを定義するために用いられる。 オフセットは、すべての静的ファイルのためにホストにより用いられるファイルサイズに対応する範囲外にあってはならない。
Stream	<length>	デバイスへまたはデバイスから転送されるデータの中断されないストリームを定義する。 モデリングのためのみに用いられる。
Pause	<time>	以降のコマンドの実行前に挿入される遅延を定義する。 モデリングのためのみに用いられる。
Idle		デバイスはアイドル状態に入り、この状態の間、デバイスは内部操作を実行し得る。 idle コマンドは、どのような保留中のガベージコレクション操作も強制的に実行されるようにするため、静止ファイルのための close コマンドの直後に用いられる。 デバイスがバックグラウンド操作について非ビジーステータスを示すまで、その他のコマンドはデバイスに一切送られるべきではない。
Standby		デバイスは待機状態に入り、この状態の間、デバイスは内部操作を実行することができない。
Shut-down		デバイスはシャットダウンされ、デバイスが次にビジー状態にならなければ、電力が取り去られる。
Capacity		ファイルデータにより占められる容量および新しいファイルデータが利用可能な容量の報告をデバイスに求めるホストからの要求。
Status		デバイスの現在のステータスの報告をデバイスに求めるホストからの要求。 status コマンドは、実行されているコマンドを終了させない。

Command Set Used with Static Files (Part B)

Command	Parameters	Description
Save_buffer	<fileID>	Saves data in buffer for specified static file to a temporary location in flash memory. Save_buffer is used when the protocol in use for static files requires that data that has been supplied for the file must be commited to flash memory.
Write_pointer	<fileID> <offset>	Defines new current value for write_pointer for specified file. Write_pointer is used to define the offset within the static file of data to be written, for the first data after the file is opened, or for data that is not sequential to previously written data. The offset should not be outside the range corresponding to the file size used by the host for all static files.
Read_pointer	<fileID> <offset>	Defines new current value for read_pointer for specified file. Read_pointer is used to define the offset within the static file of data to be read, for the first data after the file is opened, or for data that is not sequential to previously read data. The offset should not be outside the range corresponding to the file size used by the host for all static files.
Stream	<length>	Defines an uninterrupted stream of data that will be transferred to or from the device. Used for modeling only.
Pause	<time>	Defines a delay that is inserted before execution of the following command. Used for modeling only.
Idle		The device should enter an idle state, within which it may perform internal operations. The idle command is used immediately after a Close command for a static file, to force any pending garbage collection operations to be performed. No other command should be sent to the device until the device shows not-busy staus for background operations.
Standby		The device should enter a standby state, within which it may not perform internal operations.
Shut-down		The device will be shut down and power will be removed when it is next not in a busy state.
Capacity		Request from host for the device to report capacities occupied by file data and available for new file data.
Status		Request from host for the device to report its current status. The status command does not terminate a command being executed.

FIG. 13-1B

フロントページの続き

審査官　上嶋　裕樹

(56)参考文献　特開２００３－２０８３５２（ＪＰ，Ａ）
特開２００４－３１０５７３（ＪＰ，Ａ）
特開平１０－０６９４２０（ＪＰ，Ａ）
特開２００５－１２２４３９（ＪＰ，Ａ）
国際公開第２００４／０４０４５３（WO，Ａ２）
特開２００２－３６６４２３（ＪＰ，Ａ）
国際公開第２００５／０６６７９３（WO，Ａ２）
特表２００４－５２６２３３（ＪＰ，Ａ）
特開平１０－３２６２２７（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 12/00