(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2011년06월20일
(11) 등록번호 10-1042588
(24) 등록일자 2011년06월13일

(51) Int. Cl.
G06F 12/00 (2006.01) *G06F 12/02* (2006.01)
(21) 출원번호 10-2007-7018754
(22) 출원일자(국제출원일자) 2006년02월08일
심사청구일자 2011년02월08일
(85) 번역문제출일자 2007년08월16일
(65) 공개번호 10-2007-0116792
(43) 공개일자 2007년12월11일
(86) 국제출원번호 PCT/US2006/004658
(87) 국제공개번호 WO 2006/088727
국제공개일자 2006년08월24일
(30) 우선권주장
11/060,249 2005년02월16일 미국(US)
(56) 선행기술조사문헌
US20040073727 A1
US6763424 B2
WO2004046937 A1
EP1571557 A
전체 청구항 수 : 총 25 항

(73) 특허권자
샌디스크 코포레이션
미합중국, 캘리포니아주 95035, 밀피타스, 맥카시 볼레바드 601
(72) 발명자
신클레어, 알란, 웰시
영국, 매디스톤 에프케이2 0비유, 팔키르크, 캔디, 브로드헤드, 더 코테지스
스미스, 피터, 존
영국, 스코트랜드 이에이치22 3에이치에이, 미들로시안, 이스크뱅크, 보니리그 로드 21
(74) 대리인
송범엽, 박경재

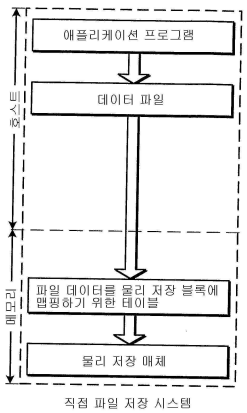
심사관 : 권오성

(54) 플래시 메모리들에 직접 데이터 파일 저장

(57) 요 약

호스트 시스템 데이터 파일들은 각 파일의 고유 신원 및 이 파일 내 데이터의 오프셋들로, 그러나 메모리에 대한 어떠한 중간 논리 어드레스들 혹은 가장 어드레스 공간의 사용없이 큰 소거 블록 플래시 메모리 시스템에 직접 기입된다. 파일들이 메모리에 저장되는 디렉토리 정보는 호스트가 아니라 그의 제어기에 의해 메모리 시스템 내에 유지된다. 호스트와 메모리 시스템들간의 파일 기반의 인터페이스는 메모리 시스템 제어기가 증가된 효율로 메모리 내에 데이터 저장 블록들을 이용할 수 있게 한다.

대 표 도 - 도10



직접 파일 저장 시스템

특허청구의 범위

청구항 1

호스트 시스템과, 함께 소거 가능하고 복수의 데이터 단위를 다수의 페이지로 개별 저장하는 메모리 셀의 블록으로 구성된 메모리 셀을 구비한 재프로그램 가능한 비휘발성 반도체 대량 저장 시스템 사이에 데이터를 전송하는 방법으로서,

개별 블록은 새로운 데이터가 이에 기입되기 전에 소거되는, 데이터를 전송하는 방법에 있어서,

다양한 양의 데이터를 포함하는 복수의 파일은 상기 호스트 시스템에 의해 생성되고,

상기 호스트 시스템은 고유한 파일 식별자 및 개별 파일 내 데이터의 오프셋에 의해 생성된 개별 데이터 파일을 확인하고, 상기 개별 파일의 데이터 및 상기 파일 식별자와 오프셋을 상기 대량 저장 시스템에 전송하며,

상기 대량 저장 시스템은, 상기 호스트 시스템으로부터 수신된 데이터가 기입될 때까지, 상기 호스트 시스템으로부터 수신된 데이터의 파일 중 개별 파일의 적어도 하나의 단위를 적어도 한 번에, 적어도 하나의 이전에 소거된 메모리 셀의 블록에 기입하고, 데이터의 개별 단위는 512보다 많은 바이트를 포함하고,

상기 대량 저장 시스템은 상기 호스트로부터 수신된 파일 식별자와 오프셋을, 상기 확인된 파일의 데이터가 임의의 중간 논리 어드레스 변환을 사용하지 않고 기입되는 메모리 셀의 페이지와 블록의 물리 어드레스로 직접 변환하는, 데이터 전송 방법.

청구항 2

제 1항에 있어서, 상기 개별 데이터 단위는 적어도 1024 바이트를 포함하는, 데이터 전송 방법.

청구항 3

제 2항에 있어서, 적어도 8개 단위의 데이터가 상기 대량 저장 시스템의 개별 블록에 기입되는, 데이터 전송 방법.

청구항 4

제 1항에 있어서, 파일 대량 저장 시스템의 상기 블록 내 상기 메모리 셀은 복수의 행으로 배열되고, 데이터 기입은 상기 적어도 하나의 이전에 소거된 메모리 셀의 블록 내에 행당 적어도 두 단위의 데이터를 기입하는 것을 포함하는, 데이터 전송 방법.

청구항 5

제 4항에 있어서, 데이터 기입은 상기 블록의 일 단부로부터 타 단부까지 순서대로 한 번에 상기 적어도 하나의 이전에 소거된 메모리 셀의 블록 내에 상기 메모리 셀의 하나의 행에 상기 데이터 단위를 기입하는 것을 포함하는, 데이터 전송 방법.

청구항 6

제 1항에 있어서, 상기 대량 저장 시스템은, 대량 저장 시스템 블록의 대응하는 어드레스의 디렉토리 및 상기 고유한 파일 식별자와 오프셋을 유지함으로써, 상기 호스트로부터 수신된 파일 식별자와 오프셋을 메모리 셀의 블록의 물리 어드레스로 변환하는, 데이터 전송 방법.

청구항 7

제 1항에 있어서, 상기 수신된 데이터를 기입하는 것은, 상기 수신된 호스트 파일 데이터의 적어도 하나의 단위를 적어도 두 개의 이전에 소거된 메모리 셀의 블록의 각 블록에 동시에 기입하는 것을 포함하는, 데이터 전송 방법.

청구항 8

제 7항에 있어서, 상기 수신된 호스트 파일 데이터가 기입되는 메타블록으로서 함께 블록을 링크하는 것을 더 포함하는, 데이터 전송 방법.

청구항 9

제 1항에 있어서, 상기 메모리 셀은 NAND 구조에 따라 구성되는, 데이터 전송 방법.

청구항 10

제 1항에 있어서, 상기 대량 저장 시스템은 상기 호스트 데이터 파일을 생성하는 호스트와 접속하도록 구성된 외부 콘택을 갖한 엔클로저(enclosure) 내에 내장된, 데이터 전송 방법.

청구항 11

제 1항에 있어서, 상기 대량 저장 시스템은 적어도 256 메가바이트의 데이터를 저장하기에 충분한 다수의 메모리 셀을 포함하는, 데이터 전송 방법.

청구항 12

제 1항에 있어서, 상기 복수의 파일은 상기 호스트 시스템에서 작동하는 애플리케이션 프로그램으로부터 상기 호스트 시스템에 의해 생성되는, 데이터 전송 방법.

청구항 13

제 1항에 있어서, 추가적으로,

상기 호스트는 대량 저장장치 시스템에 의해 허용된 미리 설정된 수 N개 미만인 한 번에 열린 다수의 파일의 데이터를 생성하고,

상기 대량 저장장치 시스템은, 한 번에 열린 파일의 수 중 하나의 파일만의 데이터를 수신하고 저장하도록 개별 제공된 대응하는 개수의 메모리 셀의 열린 블록으로, 상기 호스트로부터 수신된 열린 파일의 수의 데이터를 기입하는, 데이터 전송 방법.

청구항 14

제 13항에 있어서, 추가적으로, 상기 호스트는 열린 파일의 수 중 하나를 닫기 위한 명령을 생성하고, 상기 대량 저장장치 시스템은 닫힐 파일의 데이터를 수신하고 저장하도록 제공된 블록 중 적어도 하나를 가비지 수거(garbage collecting)하여 열린 파일의 수 중 하나를 닫기 위한 명령에 대응하는, 데이터 전송 방법.

청구항 15

제 14항에 있어서, 상기 대량 저장장치 시스템은, 상기 블록 중 상기 적어도 하나의 블록이 닫힐 파일의 유효 데이터로 부분적으로만 채워지면, 닫힌 파일의 데이터를 수신하고 저장하도록 제공된 블록 중 적어도 하나를 가비지 수거하는, 데이터 전송 방법.

청구항 16

제 1항에 있어서, 추가적으로, 상기 호스트로부터 수신된 파일 식별자와 오프셋을, 상기 식별된 파일의 데이터가 기입된 메모리 셀의 블록 및 페이지의 물리 어드레스로 변환하는 것은, 메모리 셀의 특정 블록으로 기입된 연속적인 오프셋 어드레스를 갖는 파일의 데이터 그룹의 적어도 하나의 기록(record)을 포함하는 각각의 파일에 대한 개별 인덱스(separate index)로 상기 대량 저장장치 시스템에 유지되고, 상기 기록은, 메모리 셀의 상기 특정 블록 내에 데이터 그룹의 어드레스와, 상기 특정 블록 어드레스에 저장된 파일 데이터의 대응하는 오프셋 어드레스를 포함하는, 데이터 전송 방법.

청구항 17

제 16항에 있어서, 상기 오프셋 및 블록 어드레스는 일 바이트의 입도(granularity)로 유지되는, 데이터 전송 방법.

청구항 18

적어도 256 메가바이트의 데이터 저장 용량을 구비한 메모리와 상기 메모리의 제어기를 갖는 플래시 메모리 시스템에서, 상기 메모리는 이의 저장 요소가 함께 소거 가능한 요소의 블록으로 그룹화되어 구성되고, 상기 제어

기는,

상기 메모리 시스템의 외부로부터 상기 메모리에 저장될 다양한 양의 데이터의 호스트에 의해 생성된 데이터 파일을 수신하는 단계로서, 개별 수신된 데이터 파일의 데이터는 고유한 파일 식별자와 상기 파일 내 데이터의 오프셋에 의해 확인되는, 상기 단계와,

소거된 저장 요소 블록을 확인하고, 상기 확인된 블록 내 상기 수신된 데이터 파일의 하나 이상의 데이터 단위를 한 번에 저장하는 단계로서, 데이터 단위는 개별적으로 512 바이트보다 많은 수신된 데이터를 포함하는, 상기 단계와,

상기 블록, 및 상기 개별 데이터 파일의 데이터가 고유한 파일 식별자와 상기 파일 내 데이터의 오프셋에 저장되는 상기 블록 내 저장 요소를 직접 맵핑하는 어드레스 변환 기록을, 상기 메모리 내에서, 임의의 중간 논리 어드레스 변환을 사용하지 않고, 유지하는 단계를

포함하는 방법에 따라 동작하는, 방법.

청구항 19

제 18항에 있어서, 상기 데이터 단위는 개별적으로 적어도 1024 바이트의 수신된 호스트 파일 데이터를 포함하는, 방법.

청구항 20

제 18항에 있어서, 상기 수신된 데이터 파일을 저장하는 것은, 상기 수신된 호스트 파일 데이터의 일 단위를 적어도 두 개의 소거된 메모리 셀 블록 각각에 한 번에 기입하는 단계를 포함하는, 방법.

청구항 21

제 20항에 있어서, 상기 수신된 호스트 파일 데이터가 저장되는 메타블록으로서 함께 블록을 논리적으로 링크하는 단계를 더 포함하는, 방법.

청구항 22

제 18항에 있어서, 상기 제어기 동작 방법은, 상기 호스트 파일을 생성하는 호스트와 접속하도록 구성된 외부 콘택을 구비하는 엔클로저 내에 내장된 상기 플래시 메모리 시스템에서 행해지는, 방법.

청구항 23

대량 저장 메모리 시스템에 있어서,

적어도 하나의 반도체 기판에 형성된 재프로그램 가능한 비휘발성 메모리 셀로서, 상기 메모리 셀은, 함께 소거될 수 있고 복수의 직렬 스트링의 메모리 셀을 가로질러 확장하는 도전성 워드 라인을 공유하는 이들 메모리 셀에서 함께 접속되어 메모리 셀의 행을 한정하는 복수의 메모리 셀 블록에 배열되고, 개별 행의 상기 메모리 셀은 적어도 1024 바이트의 데이터를 저장하며, 상기 블록은 적어도 8 행의 메모리 셀을 개별적으로 포함하는, 상기 재프로그램 가능한 비휘발성 메모리 셀과,

고유한 파일 식별자 및 파일 내 데이터의 오프셋에 의해 개별적으로 식별된 다양한 양의 데이터의 파일을 수신하도록 구성되고 수신된 데이터 파일이 하나 이상의 메모리 셀 블록에 저장되도록 하는 마이크로프로세서를 포함하는 제어기와,

상기 제어기에 의해 유지되는 것으로서, 상기 블록과, 상기 수신된 데이터가 고유 파일 신원 및 상기 파일 내 데이터의 오프셋과 직접 대응에 의해 저장되는 메모리 셀의 행을 확인하는 어드레스 변환 인덱스로서, 상기 확인된 파일의 데이터는 임의의 중간 논리 어드레스 변환을 사용하지 않고 기입되는, 상기 어드레스 변환 인덱스를

포함하는, 대량 저장 메모리 시스템.

청구항 24

제 23항에 있어서, 상기 제어기는 상기 메모리 셀 블록의 적어도 일부를 함께 링크하여 2개 이상의 블록을 개별 포함하는 복수의 메타블록을 형성하고, 상기 제어기는 수신된 데이터 파일의 데이터가, 적어도 하나의 메타블록

의 블록의 행에 병렬로 기입되도록 하는, 대량 저장 메모리 시스템.

청구항 25

제 23항에 있어서, 상기 메모리 시스템은 상기 데이터 파일을 생성하는 호스트와 착탈 가능하게 접속되도록 구성된 외부 콘택을 구비한 휴대용 엔클로저 내에 포함되는, 대량 저장 메모리 시스템.

명 세 서

배 경 기 술

[0001] 이 출원은 반도체 플래시 메모리와 같은 재-프로그램가능한 비휘발성 메모리 시스템들의 동작에 관한 것으로, 특히, 호스트 디바이스와 메모리간 인터페이스의 관리에 관한 것이다. 여기에서 참조되는 모든 특허들, 특허출원들, 논문들 및 그외 공보들, 문헌들 및 등은 모든 목적들을 위해 이들 전체를 참조문헌으로 여기 포함시킨다.

[0002] 상용 플래시 메모리 시스템들의 초기 세대에서, 사각형상 어레이의 메모리 셀들은 각각이 표준 디스크 드라이브 섹터의 데이터량, 즉 512바이트를 저장한 많은 그룹들의 셀들로 분할되었다. 이를테면 16바이트의 추가된 량의 데이터는 통상적으로 오류정정 코드(ECC) 및 아마도 사용자 데이터 및/또는 이것이 저장된 메모리 셀 그룹에 관계된 그외 오버헤드 데이터를 저장하기 위해 각 그룹 내에 또한 포함된다. 이러한 각 그룹 내 메모리 셀들은 함께 소거될 수 있는 최소 수의 메모리 셀들이다. 즉, 소거단위는 실제로, 하나의 데이터 섹터와 포함되는 임의의 오버헤드 데이터를 저장하는 메모리 셀들의 수이다. 이러한 유형의 메모리 시스템의 예들은 미국특허 5,602,987 및 6,426,893에 기술되어 있다. 메모리 셀들을 데이터로 재-프로그래밍하기에 앞서 이들 메모리 셀들이 소거될 필요가 있는 것이 플래시 메모리의 특징이다.

[0003] 플래시 메모리 시스템들은 가장 공통적으로 메모리 카드 형태로, 혹은 개인용 컴퓨터, 카메라 등과 같은 다양한 호스트들에 착탈가능하게 접속되는, 그러나 이러한 호스트 시스템들 내 내장될 수도 있는 플래시 드라이브 형태로 제공된다. 데이터를 메모리에 기입할 때, 통상적으로 호스트는 고유 논리 어드레스들을 메모리 시스템의 연속된 가상 어드레스 공간 내에 섹터들, 혹은 클러스터들 혹은 이외의 단위들의 데이터에 할당한다. 디스크 운영 시스템(DOS)처럼, 호스트는 메모리 시스템의 논리 어드레스 공간 내 어드레스들에 데이터를 기입하고 이들 어드레스로부터 데이터를 독출한다. 메모리 시스템 내 제어기는 호스트로부터 수신된 논리 어드레스들을 메모리 어레이 내, 데이터가 실제로 저장되는 물리 어드레스들로 변환하고, 이어서 이들 어드레스 변환들을 주시한다. 메모리 시스템의 데이터 저장용량은 적어도 메모리 시스템에 대해 정의된 전체 논리 어드레스 공간에 대해 어드레싱이 가능한 데이터량 만큼 크다.

[0004] 플래시 메모리 시스템들의 나중 세대에서, 소거단위의 크기는 복수 섹터들의 데이터를 저장하기에 충분한 한 블록의 메모리 셀들로 증가되었다. 메모리 시스템들이 접속되는 호스트 시스템들이 섹터들과 같은 작은 최소단위들로 데이터를 프로그램하고 독출할 수 있을지라도, 많은 수의 섹터들이 플래시 메모리의 단일 소거단위로 저장된다. 호스트가 논리 섹터들의 데이터를 업데이트하거나 교체할 때 블록 내 어떤 섹터들의 데이터가 폐용(obsolete)되는 것이 일반적이다. 블록 내 저장된 임의의 데이터가 덮어씌여질 수 있기 전에 전체 블록이 소거되어야 하기 때문에, 새로운 혹은 업데이트된 데이터는 통상적으로, 소거되어 있고 데이터를 위한 남은 용량을 갖는 또 다른 블록에 저장된다. 이 프로세스는 원 블록을 메모리 내에 사용가능 공간을 취하는 폐용 데이터를 갖게 한다. 그러나, 이 블록은 이 블록 내 남아있는 어떤 유효한 데이터가 있다면 소거될 수 없다.

[0005] 그러므로, 메모리의 저장용량을 더 잘 이용하기 위해서, 유효 부분 블록 분량들의 데이터를 소거된 블록에 카피함으로써 이들 데이터를 통합 또는 수거하고 이에 따라 이들 데이터를 카피한 블록(들)이 소거되어 이들 전체 저장용량이 재사용될 수 있게 하는 것이 일반적이다. 블록 내 데이터 섹터들의 논리 어드레스들의 순서로 이들 데이터 섹터들을 그룹화하는 것이 데이터를 독출하여 독출된 데이터를 호스트에 전송하는 속도를 증가시키기 때문에 이를 행하기 위해서 데이터를 카피하는 것이 또한 바람직하다. 이러한 데이터 카피가 너무 빈번하게 일어난다면, 메모리 시스템의 동작 수행이 저하될 수 있다. 이것은, 전형적인 경우로서 메모리의 저장용량이 시스템의 논리 어드레스 공간을 통해 호스트에 의해 어드레스가능한 데이터량 정도인 메모리 시스템들의 동작에 특히 영향을 미친다. 이 경우, 호스트 프로그래밍 명령이 실행될 수 있기 전에 데이터 통합 또는 수거가 요구될 수 있다. 그러면 프로그래밍 시간이 증가된다.

[0006] 블록들의 크기들은 주어진 반도체 영역에 저장될 수 있는 데이터의 비트들의 수를 증가시키기 위해서 메모리 시스템들의 연속적인 세대들에서 증가하고 있다. 256 데이터 섹터들 및 그 이상을 저장하는 블록들이 일반적인 것이 되고 있다. 또한, 2, 4 혹은 그 이상의 블록들의 서로 다른 어레이들 혹은 서브-어레이들은 데이터 프로그래

밍 및 독출에서 병행도를 증가시키기 위해서 흔히 메타블록들에 함께 논리적으로 링크된다. 이러한 큰 용량과 함께 동작 단위들은 이들을 효율적으로 동작시키는데 있어 도전이 되고 있다.

[0007] <발명의 요약>

[0008] 이러한 큰 소거 블록 플래시 메모리 시스템들을 효율적으로 동작시킴에 있어 부닥치게 되는 어떤 문제들을 다양한 정도로 극복하는 많은 기술들이 개발되었다. 한편, 본 발명은 메모리와 호스트 시스템간의 데이터 전송 인터페이스를 변경함으로써 보다 근본적인 접근을 취한다. 현재 행해지는 것으로서, 가상 어드레스 공간 내 논리 어드레스들의 사용에 의해 메모리와 호스트 시스템간에 데이터를 통신하기보다는, 데이터 파일은 호스트에 의해 할당된 파일명 및 파일 내 오프셋 어드레스에 의해 확인된다. 이때 메모리 시스템은 각 섹터 혹은 데이터의 다른 단위가 속하는 호스트 파일을 안다. 여기에서 논의되는 파일단위는 이를테면 순차적 오프셋 어드레스들을 갖춤으로써, 순서로 되어 있고 또한 호스트 계산 시스템에서 동작하는 애플리케이션 프로그램에 의해 생성되고 고유하게 식별되는 한 세트의 데이터이다.

[0009] 이것은 호스트들이 파일들을 식별함이 없이 공통의 한 세트의 논리 어드레스들에 의해 모든 파일들 내에 메모리 시스템에의 데이터를 확인하기 때문에 현 상용 메모리 시스템들에 의해 채용되지 않는다. 논리 어드레스들을 사용하는 대신에 파일 객체들에 의해 호스트 데이터를 확인함으로써, 메모리 시스템 제어기는 이러한 빈번한 데이터 통합 및 수거의 필요성을 감소시키도록 데이터를 저장할 수 있다. 데이터 카피 동작들의 빈도 및 카피되는 데이터량은 현저하게 감소되고, 그럼으로써 메모리 시스템의 데이터 프로그래밍 독출 성능을 증가시킨다. 또한, 메모리 시스템 제어기는 디렉토리 및 호스트 파일들이 저장되는 메모리 블록들의 인덱스 테이블 정보를 유지한다. 그러면 현재 논리 어드레스 인터페이스를 관리하는데 필요한 파일 할당 테이블(FAT)을 호스트가 유지하는 것은 불필요하다.

[0010] 본 발명의 다른 면들, 이점들, 특징들 및 상세는 다음의 예들의 설명에 포함되고 이 설명은 첨부한 도면들과 함께 취해질 것이다.

실 시 예

[0083] 파일 기반 인터페이스 실시예들의 설명

[0084] 대량의 데이터의 저장을 위한 호스트와 메모리 시스템간 개선된 인터페이스는 논리 어드레스 공간의 사용을 제거한다. 대신에 호스트는 고유 fileID(혹은 이외 고유 참조) 및 파일 내 데이터의 단위들(이를테면 바이트들)의 오프셋 어드레스들에 의해 각 파일을 논리적으로 어드레스한다. 이 파일 어드레스는 직접 메모리 시스템 제어기에 주어지고, 그러면 이 제어기는 각 호스트 파일의 데이터가 물리적으로 저장된 곳에 대한 자신의 테이블을 유지한다. 이 새로운 인터페이스는 도 2-6을 참조로 위에 기술된 바와 동일한 메모리 시스템에 구현될 수 있다. 위에 기술된 것과의 주된 차이는 이 메모리 시스템이 호스트 시스템과 통신하는 방식이다.

[0085] 이 파일 기반의 인터페이스는 도 9에 도시되었고, 이는 도 7의 논리 어드레스 인터페이스와 비교된다. 파일 1, 파일 2, 파일 3의 각각의 신원과 도 9의 파일들 내 데이터의 오프셋들은 메모리 제어기에 직접 보내진다. 그러면 이 논리 어드레스 정보는 메모리 제어기 기능(173)에 의해 메타블록들 및 메모리(165)의 메타페이지들의 물리 어드레스들로 변환된다.

[0086] 파일 기반의 인터페이스가 도 10에 의해 도시되었고, 이는 도 8의 논리 어드레스 인터페이스와 비교된다. 논리 어드레스 공간 및 호스트에 의해 유지되는 도 8의 FAT 테이블은 도 10에는 없다. 그보다는, 호스트에 의해 생성된 데이터 파일들은 파일 번호 및 파일 내 데이터의 오프셋들에 의해 메모리 시스템에 확인된다. 이어서 메모리 시스템은 파일들을 메모리 셀 어레이의 물리 블록들에 직접 맵핑한다.

[0087] 도 11을 참조하면, 여기에서 기술된 예로서의 대량 저장 시스템의 기능층이 도시되었다. 주로 이 설명의 주제는 "직접 파일 저장 백-엔드 시스템" 이다. 이것은 메모리 시스템의 동작에 본질이며, "직접 파일 인터페이스" 및 "파일 기반 프론트-엔드 시스템"을 통해 파일 기반 인터페이스 채널로 호스트 시스템과 통신한다. 각 호스트 파일은 이를테면 파일 이름에 의해 고유하게 식별된다. 파일 내 데이터는 파일에 고유한 선형 어드레스 공간 내 오프셋 어드레스에 의해 식별된다. 논리 어드레스 공간이 메모리 시스템에 대해 정의될 필요성은 없다.

[0088] 명령들

[0089] 호스트 시스템으로부터 한 세트의 직접 파일 인터페이스 명령들은 메모리 시스템의 동작을 지원한다. 예로서의

한 세트의 이러한 명령들이 도 12a-12e에 주어져 있다. 이들은 이 설명의 나머지 부분을 전체를 통해 참조를 위해 단지 간략하게만 요약된다. 도 12a는 정의된 프로토콜에 따라, 호스트와 메모리 시스템들간에 데이터가 전송되게 하는데 사용되는 호스트 명령들을 리스트한다. 파일 내 특정 오프셋 특정 오프셋 (<offset>)에 지정된 파일 (<fileID>) 내 데이터가 메모리 시스템에 기입되거나 이로부터 독출된다. Write, Insert 혹은 Update 명령의 전송에 이어 호스트로부터 데이터가 메모리 시스템에 전송되고, 메모리 시스템은 이의 메모리 어레이에 데이터를 기입함으로써 응답한다. 호스트에 의한 Read 명령의 전송은 지정된 파일의 데이터를 호스트에 보냄으로써 메모리 시스템이 응답하게 한다. 데이터 오프셋은 메모리 시스템이 파일의 추가의 데이터가 저장될 수 있는 다음 저장 위치를 확인하는 포인터를 유지한다면 Write 명령과 함께 보내질 필요는 없다. 그러나, Write 명령이 이미 기입된 파일 내 오프셋 어드레스를 포함한다면, 메모리 디바이스는 이를 오프셋 어드레스에서 시작하는 파일 데이터를 업데이트하라는 명령인 것으로 해석하고 그럼으로써 별도의 Update 명령에 대한 필요성을 제거할 수 있다. Read 명령에 대해서, 데이터 오프셋은 전체 파일이 독출되어야 한다면 호스트에 의해 명시될 필요는 없다. 이들 도 12a의 데이터 명령들 중 하나의 실행은 호스트 시스템에 의한 어떤 다른 명령의 전송에 응하여 종료된다.

[0090] 또 다른 데이터 명령은 Remove 명령이다. 도 12a의 다른 데이터 명령들과는 달리, Remove 명령은 이에 이어 데이터의 전송이 행해지지 않는다. 이의 효과는 메모리 시스템이 명시된 offset1과 offset2간에 데이터를 폐용으로서 마크하게 하는 것이다. 그러면 이들 데이터는 폐용 데이터가 존재하는 파일 혹은 블록의 다음 데이터 콤팩트 또는 가비지 수거 동안 제거된다.

[0091] 도 12b는 메모리 시스템 내 파일들을 관리하는 호스트 명령들을 리스트한다. 호스트가 메모리 시스템 내 새로운 파일의 데이터를 기입하려고 할 때, 호스트는 먼저 Open 명령을 발행하고 메모리 시스템은 새로운 파일을 여는 것에 의해 응답한다. 한번에 열려 있을 수 있는 파일 수는 통상적으로 명시될 것이다. 호스트가 파일을 닫을 때, Close 명령은 열린 파일을 유지하는데 사용되는 자원들이 리다이렉트(redirect)될 수 있음을 메모리 시스템에 알린다. 통상적으로 메모리 시스템은 가비지 수거를 위해 즉시 이러한 파일을 스케쥴링할 것이다. 기술된 직접 파일 인터페이스로, 가비지 수거는 논리적으로 관리되고, 물리적으로 개개의 메모리 셀 블록들에가 아니라, 주로 파일들에 대해 수행된다. Close_after 명령은 파일이 닫혀질 것이라는 앞선 통보를 메모리 시스템에 준다. 파일 Delete 명령은 명시된 우선도 규칙들에 따라, 삭제된 파일로부터의 데이터를 내포하는 메모리 셀 블록들을 메모리 시스템이 소거되게 즉시 스케쥴링하게 한다. Erase 명령은 메모리로부터 즉시 소거될 명시된 파일의 데이터를 명시한다.

[0092] Delete 명령과 Erase 명령간에 주된 차이는 지정된 파일 데이터를 소거하는데 있어 주어진 우선도이다. 호스트는 가장 이른 실제적인 시간에 메모리로부터 보안적 혹은 민감 데이터를 제거하기 위해 Erase 명령을 사용할 수 있는 반면, Delete 명령은 이러한 데이터가 낮은 우선도로 소거되게 한다. 메모리 시스템에 전원을 내렸을 때 Erase 명령의 사용은 메모리 디바이스가 호스트로부터 제거되기 전에 민감 데이터를 제거하고 이에 따라 메모리 디바이스의 후속 사용동안 다른 사용자들 혹은 호스트 시스템들에 그 데이터의 배포를 방지한다. 이들 명령들 모두는 바람직하게는 백그라운드에서, 즉 주 데이터 명령들(도 12a)의 실행을 느려지게 함이 없이 실행된다. 어쨌든, 호스트로부터 또 다른 명령의 수신은 통상적으로, 메모리 제어기가 어떤 백그라운드 동작이든 이를 종료하게 할 것이다.

[0093] 메모리 시스템 내에 디렉토리들에 관계된 호스트 명령들이 도 12c에 리스트되어 있다. 각각의 디렉토리 명령은 명령이 속한 디렉토리의 신원 (<directoryID>)을 포함한다. 메모리 시스템 제어기가 디렉토리들을 유지할지라도, 디렉토리들에 관한 명령들 및 디렉토리들의 지정들은 호스트 시스템에 의해 제공된다. 메모리 제어기는 메모리 시스템에 저장된 펌웨어에 따라, 호스트에 의해 공급된 디렉토리 지정들로, 이들 명령들을 실행한다.

[0094] <fileID> 파라미터는 파일에 대한 완전한 경로명이거나, 혹은 여기에서는 file_handle로 언급되는, 파일에 대한 어떤 단축 식별자일 수 있다. 파일 경로명은 어떤 명령들에 관련하여 도 11의 직접-파일 인터페이스에 제공된다. 이것은 파일이 처음으로 열렸을 때 완전히 명백한 엔트리가 파일 내 생성되게 하며, 현존하는 파일이 열렸을 때 파일 디렉토리 내 정확한 현존 엔트리가 액세스될 수 있게 한다. 파일 경로명 신택스는 DOS 파일 시스템에 의해 사용되는 표준에 따를 수 있다. 경로명은 디렉토리들의 계층과 디렉토리의 최하위의 레벨 내의 파일을 기술한다. 경로 세그먼트들은 "\"에 의해 경계가 정해질 수 있다. "\"가 앞에 놓여지는 경로는 루트 디렉토리에 관련된다. "\"가 앞에 놓여지지 않은 경로는 현재의 디렉토리에 관련된다. ".."의 세그먼트 경로는 현재 디렉토리의 모(parent) 디렉토리를 나타낸다.

[0095] 대안적으로, 열린 파일들은 파일이 처음 생성될 때 저장 디바이스에 의해 할당되는 file_handle 파라미터에 의해 식별될 수도 있다. 이어서, 저장 디바이스는 호스트가 파일을 열 때마다 단축 파일 지정을 호스트에 알린다. 이어서 호스트는 열린 파일의 Write, Insert, Update, Read, Close 및 Close_after 명령들에 file_handle을 사용할 수 있다. 통상적으로 호스트에 의한 파일에의 액세스는 파일 디렉토리의 계층이 내비게이트될 필요가 없기 때문에 완전한 경로명이 사용되는 경우보다 더 빠를 것이다. Open 명령의 사용에 의해 파일이 처음 열렸을 때, file_handle이 메모리 시스템에 의해 이 파일에 아직 할당되지 않을 것이기 때문에 통상적으로 완전한 경로명이 사용된다. 그러나, file_handle은 이미 가용하다면 사용될 수 있다. fileID를 이용하는 도 12a 및 도 12b의 나머지 Delete 및 Erase 명령들에 대해서, 호스트에 의해 공급되는 부정확한 file_handle에 대한 방호로서 완전한 파일 경로명의 사용이 바람직하다. 호스트가 현존하나 의도되지 않은 파일 중 하나와 일치하는 부정확한 경로명을 부주의하게 생성하는 것은 더 어렵다.

[0096] 유사하게 도 12c의 디렉토리 명령들은 이들이 속한 디렉토리의 <directoryID> 신원으로 도 11의 직접-파일 인터페이스에 의해 수신된다. 완전한 경로명은 디렉토리 명령에 의해 수신되는 바람직한 directoryID이다.

[0097] file_handle은 Open 명령에 응하여 직접-파일 인터페이스에서 대량 저장 디바이스에 의해 호스트에 리턴되는 단축형 식별자이다. file_handle을 파일에 대한 디렉토리 엔트리에 존재하는 FIT에의 포인터인 것으로서 정의하는 것이 편리하다. 이 포인터는 파일에 대한 논리 FIT 블록 번호 및 이 블록 내 논리 파일 번호를 정의한다. 이것을 file_handle로서 사용하는 것은 먼저 파일 디렉토리에서 파일을 탐색해야 할 필요없이 파일 FIT 엔트리들이 액세스될 수 있게 한다. 예를 들면, 메모리 디바이스가 64 FIT 블록들까지를 가질 수 있고, 각 FIT 블록이 64개까지의 파일들을 인덱스할 수 있다면, file_handle 1107을 가진 파일은 FIT 블록 11에 논리 파일 7로 설정된 FIT 내 데이터 그룹 엔트리들에의 포인터를 갖는다. 이 file_handle은 파일에 대한 디렉토리 및 FIT 엔트리들이 Open 명령에 응하여 생성될 때 메모리 시스템 제어기에 의해 생성되고 Close 명령에 응하여 무효하게 된다.

[0098] 도 12d는 호스트와 메모리 시스템들간 인터페이스의 상태를 관리하는 호스트 명령들을 도시한 것이다. Idle 명령은 이전에 스케쥴링되어 있는 데이터 소거 및 가비지 수거와 같은 내부 동작들을 수행할 수 있음을 메모리 시스템에 알린다. Standby 명령을 수신한 것에 응하여, 메모리 시스템은 가비지 수거 및 데이터 소거와 같은 백그라운드 동작들을 수행하는 것을 중지할 것이다. Shut-down 명령은 파워 상실이 임박함을 미리 메모리 제어기에 경고하는 것으로, 이것은 휘발성 제어기 버퍼들로부터 비휘발성 플래시 메모리에 데이터를 기입하는 것을 포함한 진행중의 메모리 동작들을 완료할 수 있게 한다.

[0099] 도 12e에 도시된 Size 명령은 통상적으로 Write 명령 전에 호스트에 의해 발행될 것이다. 이에 응하여, 메모리 시스템은 기입될 또 다른 파일 데이터에 대한 가용 용량을 호스트에 보고한다. 이것은 가용한 프로그램되지 않은 물리적 용량에서 정의된 파일 데이터 용량의 저장을 관리하는데 요구되는 물리적 용량을 감한 것에 기초하여 계산될 수 있다.

[0100] 호스트가 Status 명령(도 12e)을 발행할 때, 메모리 디바이스는 이의 현 상태로 응답할 것이다. 이 응답은 호스트에 메모리 디바이스에 관한 여러 가지 특정한 아이템들의 정보를 제공하는 서로 다른 필드들의 비트들을 가진 바이너리 워드 혹은 워드들 형태일 수 있다. 예를 들면, 하나의 2비트 필드는 디바이스 비지(busy)인지를 보고할 수 있고, 그러하다면, 메모리 디바이스가 어떤 것을 행하는 데에 비지인지에 따라 2이상의 비지 상태를 제공한다. 한 비지 상태는 메모리 디바이스가 포그라운드(foreground) 동작으로서, 데이터를 전송하기 위해 호스트의 Write 혹은 Read 명령을 실행하는 것을 처리하고 있음을 나타낼 수 있다. 제2 비지 상태 표시는 이를테면 데이터 콤팩트 혹은 가비지 수거와 같은 백그라운드 하우스키핑 동작을 메모리 시스템이 수행하고 있는 때를 호스트에 알리는데 사용될 수 있다. 호스트는 메모리 디바이스에 또 다른 명령을 보내기 전에 이 제2 비지의 종료까지 기다려야 할지를 결정할 수 있다. 하우스키핑 동작이 완료되기 전에 또 다른 명령이 보내진다면, 메모리 디바이스는 하우스키핑 동작을 종료하고 그 명령을 실행할 것이다.

[0101] 호스트는 하우스키핑 동작들이 메모리 디바이스 내에서 행해질 수 있게 하기 위해서 제2 디바이스 비지를 Idle 명령과 공동하여 사용할 수 있다. 디바이스가 하우스키핑 동작을 행할 필요성을 아마도 야기할 명령 혹은 일련의 명령을 호스트가 보낸 후에, 호스트는 Idle 명령을 보낼 수 있다. 후술하는 바와 같이, 메모리 디바이스는 하우스키핑 동작을 개시함으로써 Idle 명령에 응답하고 동시에 위에 기술된 제2 비지를 시작하게 프로그램될 수 있다. 예를 들면, Delete 명령은 이하 기술되는 알고리즘들에 따라, 가비지 수거를 수행할 필요성을 야기한다. 일련의 Delete 명령들을 발행한 후에 호스트로부터의 Idle 명령은 메모리 디바이스가 후속의 호스트 Write 명령에 응답할 수 있게 하는데 필요할 수 있는 가비지 수거를 수행할 디바이스 시간을 갖게 한다. 그렇지 않다면, 가비지 수거는 다음 Write 명령을 수신한 후에 그러나 이것이 실행될 수 있기 전에 수행될 필요가 있어, 이 명

령의 실행을 현저하게 느려지게 한다.

[0102] 데이터 기입

[0103] 새로운 데이터 파일이 메모리에 프로그램될 때, 데이터는 소거된 한 블록의 메모리 셀들에 블록 내 제1 물리적 위치부터 시작하여 순차적으로 순서대로 블록의 위치들로 진행하여 기입된다. 데이터는 파일 내 이 데이터의 오프셋들의 순서에 관계없이, 호스트로부터 수신된 순서로 프로그램된다. 프로그래밍은 파일의 모든 데이터가 메모리에 기입될 때까지 계속된다. 파일 내 데이터량이 단일 메모리 블록의 용량을 초과한다면, 제1 블록이 충만되었을 때, 프로그래밍은 제2 소거된 블록에서 계속된다. 제2 메모리 블록은 제1 메모리 블록과 동일한 방식으로, 파일의 모든 데이터가 저장되거나 제2 블록이 충만될 때까지 제1 위치로부터 순서대로 프로그램된다. 제3 혹은 추가의 블록들은 파일의 어떤 나머지 데이터로 프로그램될 수 있다. 단일 파일의 데이터를 저장하는 복수의 블록들 혹은 메타블록들은 물리적으로 혹은 논리적으로 인접할 필요는 없다. 쉽게 설명하기 위해서, 다른 것이 명시되지 않는 한, 여기에서 사용되는 "블록"이라는 용어는 메타블록들이 특정 시스템에서 사용되고 있는지에 따라, 블록 단위의 소거 혹은 복수 블록의 "메타블록"을 지칭한다.

[0104] 도 13a를 참조하면, 메모리 시스템에 데이터 파일의 기입이 도시되었다. 이 예에서, 데이터 파일(181)은 메모리 시스템의 한 블록 혹은 메타블록(183)의 저장용량보다 크며, 수직 실선들 사이에 연장한 것으로 도시되었다. 그러므로 데이터 파일(181)의 부분(184)은 제2 블록(185)에 기입된다. 이들 메모리 셀 블록들은 물리적으로 인접한 것으로 도시되었으나 그럴 필요는 없다. 파일(181)로부터의 데이터는 파일의 모든 데이터가 메모리에 기입될 때까지 이들이 호스트로부터 스트림으로 수신될 때 기입된다. 도 13a의 예에서, 데이터(181)는 도 12a의 Write 명령 후에 호스트로부터 수신되는, 파일의 처음 데이터이다.

[0105] 메모리 시스템이 저장된 데이터를 관리하고 주시하기 위한 바람직한 방법은 가변 크기의 데이터 그룹들의 사용에 의한 것이다. 즉, 파일의 데이터는 완전한 파일을 형성하기 위해 정의된 순서로 함께 연쇄(chain)될 수 있는 복수 그룹들의 데이터로서 저장된다. 그러나, 바람직하게, 파일 내 데이터 그룹들의 순서는 파일 인덱스 테이블(FIT)을 사용하여 메모리 시스템 제어기에 의해 유지된다. 호스트로부터 데이터의 스트림이 기입되고 있을 때, 불연속이 파일 데이터의 논리 오프셋 어드레스들에 혹은 데이터가 저장되고 있는 물리 공간에 있을 때는 언제나 새로운 데이터 그룹이 시작된다. 이러한 물리적 불연속의 예는 파일의 데이터가 한 블록을 채우고 또 다른 블록에 기입되기 시작할 때이다. 이것이 도 13a에 도시되었으며, 여기서 제1 데이터 그룹은 제1 블록(183)을 채우고 파일의 나머지 부분(184)은 제2 데이터 그룹으로서 제2 블록(185)에 저장된다. 제1 데이터 그룹은 (F0,D0)에 의해 나타낼 수 있고, 여기서 F0은 데이터 파일의 시작부분의 논리 오프셋이고 D0은 파일이 시작하는 메모리 내 물리 위치이다. 제2 데이터 그룹은 (F1,D1)으로 나타내며, F1은 제2 블록(185)의 시작부분에 저장되는 데이터의 논리 파일 오프셋이고, D1은 그 데이터가 저장되는 물리 위치이다.

[0106] 호스트-메모리 인터페이스를 통해 전송되는 데이터량은 데이터의 바이트 수, 데이터의 섹터들의 수, 혹은 어떤 다른 입도(granularity)로 표현될 수 있다. 호스트는 대부분 흔히 이의 파일들의 데이터를 바이트 입도로 정의하나 현 논리 어드레스 인터페이스를 통해 대용량 메모리 시스템과 통신할 때, 바이트들을 각각 512바이트들의 섹터들로, 혹은 각각 복수의 섹터들의 클러스터들로 그룹화한다. 통상적으로 이것은 메모리 시스템의 동작을 단순화하기 위해 행해진다. 여기에서 기술되는 파일 기반 호스트-메모리 인터페이스가 어떤 다른 데이터 단위를 사용할 수 있을지라도, 원 호스트 파일 바이트 입도가 일반적으로 바람직하다. 즉, 데이터 오프셋들, 길이들 등은 바이트들(들), 클러스터(들) 등에 의한 것보다는 바이트(들), 최소의 적합한 데이터 단위로 표현되는 것이 바람직하다. 이것은 여기에 기술된 기술들로 플래시 메모리 저장용량을 보다 효율적으로 사용할 수 있게 한다.

[0107] 일반적인 현존의 논리 어드레스 인터페이스들에서, 호스트는 또한 기입되는 데이터의 길이를 명시한다. 이것은 여기에서 기술된 파일 기반의 인터페이스로 행해질 수 있으나 Write 명령(도 12a)의 실행엔 필요하지 않기 때문에, 호스트는 기입되는 데이터의 길이를 제공하지 않는 것이 바람직하다.

[0108] 이때 도 13a에 도시된 방식으로 메모리에 기입된 새로운 파일은 FIT에 데이터 그룹들에 대한 인덱스 엔트리들 (F0,D0), (F1,D1)의 시퀀스로서 이 순서로 나타내어진다. 즉, 호스트 시스템이 특정 파일에 액세스하기를 원할 때는 언제나, 호스트는 이의 fileID 혹은 다른 신원을 메모리 시스템에 보내고, 그러면 메모리 시스템은 이의 FIT에 액세스하여 그 파일을 구성하는 데이터 그룹들을 확인한다. 개개의 데이터 그룹들의 길이 <length>는 메모리 시스템의 동작의 편의를 위해, 그들의 개개의 엔트리들에 포함될 수도 있다. 사용될 때, 메모리 제어기는 데이터 그룹들의 길이들을 계산하여 저장한다.

[0109] 호스트가 도 13a의 파일을 열린 상태로 유지하는 한, 물리적 기입 포인터 P는 또한 바람직하게는 그 파일에 대해 호스트로부터 수신된 어떤 또 다른 데이터를 기입하기 위한 위치를 정의하기 위해 유지된다. 파일에 대한 어떤 새로운 데이터는 파일 내 새로운 데이터의 논리 위치에 무관하게 물리 메모리 내 파일의 끝에 기입된다. 메모리 시스템은 복수의 파일들, 이를테면 4 혹은 5개의 이러한 파일들이 한번에 열려있게 할 수 있고, 이들 각각에 대해 기입 포인터 P를 유지한다. 서로 다른 파일들에 대한 기입 포인터들은 서로 다른 메모리 블록들 내 위치들을 가리킨다. 다수의 열린 파일들의 메모리 시스템 한계가 이미 존재할 때 호스트 시스템이 새로운 파일을 열기를 원한다면, 열려진 파일들 중 하나가 먼저 닫혀지고 새로운 파일이 열린다. 파일이 닫힌 후에, 그 파일에 대한 기입 포인터 P를 더 이상 유지할 필요성은 없다.

[0110] 도 13b는 도 13a의 이전에 기입되었으나 아직 열려있는 파일의 끝에 호스트에 의해서, 또한 Write 명령(도 12a)의 사용에 의해 데이터의 첨부를 도시한 것이다. 데이터(187)는 파일의 끝에 호스트 시스템에 의해 부가된 것으로 도시되었고, 이 데이터는 또한 그 파일에 대한 데이터의 끝에서 제2 블록(185) 내에 기입된다. 첨부된 데이터는 데이터 그룹(F1,D1)의 일부가 되며, 따라서 이제는 기존 데이터 그룹(184)과 첨부된 데이터(189)간에 논리적으로나 물리적 어드레스 불연속은 없기 때문에, 더 많은 데이터를 내포한다. 이에 따라 전체 파일은 여전히 FIT 내에 인덱스 엔트리들의 시퀀스 (F0,D0), (F1,D1)로서 나타내어진다. 포인터 P의 어드레스는 또한 저장된 첨부된 데이터의 끝의 포인터로 변경된다.

[0111] 도 13a의 전에 기입된 파일에 한 블록의 데이터(191)의 삽입의 예를 도 13c에 도시하였다. 호스트가 데이터(191)를 파일에 삽입하고 있을지라도, 메모리 시스템은 삽입되는 데이터를 전에 기입된 파일 데이터의 끝에 위치(193)에 첨부한다. 데이터가 열린 파일에 삽입되고 있을 때 파일의 데이터를 이들의 논리 순서로 재기입할 필요가 없지만, 그러나 이것은 호스트가 파일을 닫은 후에 백그라운드에서 나중에 행해질 수도 있다. 삽입된 데이터가 전체적으로 제2 메모리 블록(185) 내에 저장되기 때문에, 이것은 단일의 새로운 그룹(F1,D3)을 형성한다. 그러나, 이와 같이 삽입함으로써 도 13a의 이전의 데이터 그룹(F0, D0)은 2개의 그룹들로서, 하나는 삽입 전의 그룹(F0,D0)와 삽입 후의 그룹(F2,D1)로 분할된다. 이것은 이를테면 삽입의 시작 F1에서 그리고 삽입의 끝 F2에서 일어나는 것과 같이 데이터의 논리적 불연속이 있을 때는 언제나 새로운 데이터 그룹이 형성될 필요가 있기 때문이다. 그룹(F3,D2)은 물리 어드레스(D2)가 제2 블록(185)의 시작인 결과이다. 그룹들 (F1,D3) 및 (F3,D2)은, 이들에 저장된 데이터의 오프셋들에서 불연속이 있기 때문에, 이들이 동일 메모리 블록에 저장될지라도, 개별적으로 유지된다. 그러면, 삽입을 가진 원 파일은 데이터 그룹 인덱스 엔트리들 (F0,D0), (F1,D3), (F2,D1), (F3,D2)에 의해, 이 순서로 메모리 시스템 FIT에 나타내어진다. 도 13a, 도 13b, 도 13c의 예들로부터, 새로운 혹은 기존 파일에 대한 새로운 데이터가 메모리에 어떤 데이터도 폐용되게 함이 없이 기입될 수 있는 것에 유념한다. 즉, Write 및 Insert 명령들(도 12a)의 실행은 어떠한 다른 데이터도 무효가 되게 혹은 폐용되지 않게 한다.

[0112] 도 13d는 도 13a에 도시한 방식으로 원래 기입된 데이터의 어떤 부분이 Update 명령(도 12a)을 사용하여 업데이트되는 또 다른 예를 도시한 것이다. 데이터 파일의 부분(195)이 업데이트되는 것으로 도시되었다. 메모리 시스템 내 전체 파일을 업데이트로 재기입하기 보다는, 파일의 업데이트된 부분(197)이 전에 기입된 데이터에 첨부된다. 전에 기입된 데이터의 부분(199)이 이제 폐용된다. 통상적으로 폐용 데이터에 의해 취해진 공간을 제거하기 위해서 업데이트된 파일을 통합하는 것이 바람직하지만, 이것은 통상적으로 호스트가 열려진 파일을 유지하는 동안 행해지는 것이 아니라 그보다는 파일이 닫혀진 후에 백그라운드에서 행해질 수 있다. 업데이트 한 후에, 파일은 데이터 그룹 인덱스 엔트리들 (F0,D0), (F1,03), (F2,D1), (F3,D2)에 의해 이 순서로 메모리 시스템 FIT에 나타내어진다. 도 13a의 단일 데이터 그룹(F0,D0)은 다시 도 13d에서 업데이트되는 부분 앞에 하나와, 업데이트된 부분과 업데이트된 부분 뒤에 하나로, 여러 개로 분할된다.

[0113] 가변 길이 데이터 그룹들의 사용을 더 예시하기 위해서, 동일 파일을 수반하는 일련의 몇 개의 기입동작들이 도 14a-14e에 이 순서로 도시되었다. 원 파일 데이터(W1)가 먼저 Write 명령(도 12a)을 사용하여 도 14a에 도시된 바와 같이 메모리 시스템의 2개의 블록들에 기입된다. 이어서 파일은 물리 메모리 블록의 시작부분에서 시작하는 제1 그룹과 물리 메모리 블록 경계 다음에 요구되는 제2 그룹으로서, 2개의 데이터 그룹들에 의해 정의된다. 이어서 도 14a의 파일은 데이터 그룹들에 대한 인덱스 엔트리들의 시퀀스 (F0,D0), (F1,D1)에 의해 기술된다.

[0114] 도 14b에서, 도 14a에서 기입된 파일 데이터는 Update 명령(도 12a)의 사용에 의해 업데이트된다. 업데이트된 파일 데이터(U1)는 업데이트된 데이터의 이전 버전을 폐용되게 하여, 이전 그룹(F1,D1) 바로 다음에 기입된다. 도 14a의 이전 그룹(F0,D0)은 도 14b의 수정된 그룹(F0,D0)으로 단축되고, 이전 그룹(F1,D1)은 그룹(F4,D2)으로 단축된다. 업데이트된 데이터는 이들이 메모리 블록들의 경계에 겹치기 때문에 2개의 그룹들(F2,D3) 및 (F3,D4)에 기입된다. 일부 데이터는 제3 메모리 블록에 저장된다. 파일은 이제 데이터 그룹들의 인덱스 엔트리

들의 시퀀스 (F0,D0), (F2,D3), (F3,D4), (F4,D2)에 의해 기술된다.

[0115] 도 14b의 파일은 Insert 명령(도 12a)을 사용하여, 새로운 데이터 파일 I1의 삽입에 의해 도 14c에서 더욱 수정된다. 새로운 데이터 I1은, 삽입된 데이터가 메모리 블록들의 경계와 겹치기 때문에 도 14c의 새로운 그룹들 (F5,D6) 및 (F6,D7)으로서, 도 14b의 이전 그룹 (F4, D2) 바로 다음에 메모리에 기입된다. 제4 메모리 블록이 사용된다. 도 14b의 이전 그룹 (F0,D0)은 새로운 데이터 I1의 삽입 때문에, 도 14c에 단축된 그룹들 (F0,D0) 및 (F7,D5)으로 분할된다. 파일은 데이터 그룹들에 대한 인덱스 엔트리들의 시퀀스 (F0,D0), (F5,D6), (F6,D7), (F7,D5), (F8,D3), (F9,D4), (F10,D2)에 의해 기술된다.

[0116] 도 14d는 Write 명령(도 12a)을 사용하여, 파일의 끝에 새로운 데이터 W2를 첨부하는 도 14c의 데이터 파일의 또 다른 수정을 도시한 것이다. 새로운 데이터 W2는 도 14d의 새로운 그룹(F11,D8)으로서, 도 14c의 이전 그룹 (F10,D2) 바로 다음에 기입된다. 파일은 이제 데이터 그룹들에 대한 인덱스 엔트리들의 시퀀스 (F0,D0), (F5,D6), (F6,D7), (F7,D5), (F8,D3), (F9,D4), (F10,D2), (F11,D8)에 의해 기술된다.

[0117] 열린 파일에 대한 제2 업데이트가 도 14e에 도시되었고, 업데이트된 파일 데이터 U2는 호스트가 Update 명령(도 12a)을 발행함으로써 도 14d의 파일에 기입된다. 업데이트된 데이터 U2는 이 데이터의 이전 버전을 폐용되게 하고, 도 14d의 이전 그룹(F11,D8) 바로 다음에 도 14e에 기입된다. 도 14d의 이전 그룹 (F9,D4)은 도 14e에서 수정된 그룹 (F9,D4)으로 단축되고, 이전 그룹 (F10,D2)은 완전히 폐용이 되고, 이전 그룹 (F11,D8)은 새로운 그룹(F14,D9)을 형성하게 단축된다. 업데이트된 데이터는 도 14e의 새로운 그룹들 (F12,D10) 및 (F13,D11)에 기입되어 블록경계에 중첩한다. 제5 블록은 파일을 저장하는데 필요하게 된다. 파일은 이제 데이터 그룹들의 인덱스 엔트리들의 시퀀스 (F0,D0), (F5,D6), (F6,D7), (F7,D5), (F8,D3), (F9,D4), (F12,D10), (F13,D11), (F14,D9)에 의해 기술된다.

[0118] 각 파일의 데이터의 오프셋들은 바람직하게는 앞의 설명에 따라 파일의 생성 혹은 수정 후에 정확한 논리적 순서로 연속적이게 유지되는 것이 바람직하다. 그러므로, Insert 명령을 실행하는 부분으로서, 예를 들면, 호스트에 의해 제공된 삽입된 데이터의 오프셋들은 삽입 바로 전의 오프셋부터 연속적이고 삽입 후에 파일 내 현존 데이터는 삽입된 데이터 량만큼 증분된다. Update 명령은 가장 일반적으로 기존 파일의 주어진 어드레스 범위 내 데이터가, 업데이트된 데이터의 유사 량만큼 대치되는 결과를 가져오므로, 파일의 다른 데이터의 오프셋들은 통상적으로 대치될 필요가 없다. 별도의 Update 명령 대신에, 대안적으로 파일의 데이터는 저장된 파일 데이터에 이미 존재하는 한 범위의 오프셋들과 함께 호스트로부터의 데이터의 수신은 메모리 시스템에 의해서 그 오프셋 범위 내 데이터를 업데이트하라는 지시로서 해석될 수 있기 때문에 Write 명령의 사용에 의해 업데이트될 수 있다.

[0119] 위에 기술되고 도 13 및 도 14에 의해 도시된 데이터 할당 및 인덱싱 기능들 모두는 메모리 시스템의 제어기에 의해 수행됨에 유의한다. 도 12a의 Write, Insert 혹은 Update 명령들 중 하나와 함께, 호스트는 메모리 시스템에 보내지고 있는 fileID 및 파일 내 데이터의 오프셋들만을 통보한다. 메모리 시스템은 나머지를 행한다.

[0120] 지금 기술한 방식으로 호스트로부터의 파일 데이터를 플래시 메모리에 직접 기입하는 이점은 이렇게 하여 저장된 데이터의 입도 혹은 분해능이 호스트의 것과 동일하게 유지될 수 있다는 것이다. 예를 들면, 호스트 애플리케이션이 1바이트 입도로 파일 데이터를 기입한다면, 이 데이터는 1바이트 입도로 플래시 메모리에 기입될 수도 있다. 개개의 데이터 그룹 내 데이터의 량 및 위치는 바이트 수로 측정된다. 즉, 호스트 애플리케이션 파일 내 개별적으로 어드레싱이 가능한 동일 오프셋 단위의 데이터는 플래시 메모리에 저장될 때 그 파일 내 개별적으로 어드레싱이 가능하다. 블록 내 동일 파일의 데이터 그룹들 간 어떤 경계들은 가장 가까운 바이트 혹은 다른 호스트 오프셋 단위로 인덱스 테이블에 명시된다. 유사하게, 블록 내 서로 다른 파일들의 데이터 그룹들간의 경계들은 호스트 오프셋의 단위로 정의된다.

[0121] 데이터를 기입하는데 사용되지 않을 지라도, 파일의 일부를 삭제하기 위한 Delete 명령의 동작은 Insert 명령의 반대이기 때문에 이 Delete 명령의 사용을 고려하는 것이 적절하다. Delete 명령은 Delete <fileID> <offset> <length>의 포맷으로, 한 범위의 파일 오프셋 어드레스들에 적용될 수 있다. 삭제의 <length>에 대해 <offset>부터 시작하여 그 어드레스부터 계속되는 파일의 부분 내 데이터가 삭제된다. 이어서, 삭제 후 파일의 나머지 데이터의 오프셋 어드레스들은 파일 전체를 통해 인접 오프셋 어드레스들을 유지하기 위해 호스트에 의해 감분된다. 이것은 Insert 명령의 역이며, 호스트는 데이터를 파일의 중간에 부가한 후에 수정된 파일의 오프셋들이 연속적이게 되도록 삽입 후 나머지 파일 데이터의 오프셋들을 증분한다.

[0122] 가비지 수거

[0123] 도 14b 및 도 14e로부터, Update 명령에 의해, 파일을 저장하는데 필요한 물리적 공간은 파일 내 데이터량보다 커지게 되는 것에 유의한다. 이것은 업데이트들에 의해 대치된 데이터가 메모리에 저장된 채로 있게 되기 때문이다. 그러므로, 폐용된 무효한 데이터를 소거함으로써 보다 적은 물리적 저장공간에 파일의 데이터를 통합(가비지 수거)하는 것이 매우 바람직하다. 그러므로 더 많은 저장공간은 다른 데이터에 대해 사용될 수 있게 된다.

[0124] 도 14b 및 도 14e의 파일 데이터 업데이트들 외에도, 도 14c의 데이터 삽입으로 파일 데이터는 순서에서 벗어나 저장되는 결과로 되는 것에 유의한다. 즉, 업데이트들 및 삽입들은 이들이 행해질 때 메모리에 저장된 파일의 끝에 부가되고, 이들은 파일 내 어떤 곳에 거의 항시 논리적으로 위치하게 된다. 이것이 도 14b, 도 14c 및 도 14e의 예들의 경우이다. 그러므로 메모리에 저장된 파일의 데이터를 파일 내 오프셋들의 순서와 일치하게 순서를 재조정하는 것이 바람직할 수 있다. 그러면 이것은 페이지들 및 블록들을 차례차례 독출하는 것이 파일의 데이터를 이들의 오프셋 순서로 제공할 것이기 때문에 저장된 데이터를 독출하는 속도를 개선한다. 또한, 이것은 파일의 최대 가능한 조각모으기(defragmentation)를 제공한다. 그러나, 독출을 보다 효율적으로 하기 위해 파일 데이터의 순서를 재조정하는 것은 메모리 시스템의 수행에 대해선, 다른 데이터를 저장하는데 사용하기 위한 하나 이상의 메모리 블록들을 잠재적으로 일소하는 파일 데이터 통합만큼은 중요하지 않다. 그러므로 파일 내 데이터의 순서를 재조정하는 것은 통상적으로, 잇점이 동작 오버헤드를 추가할 만큼 가치가 없는 것인 단독으로 행해지는 것이 아니라, 동작 오버헤드를 거의 혹은 전혀 추가하지 않는 많은 가비지 수거 동작들의 일부로서 행해질 수 있다.

[0125] 도 14e의 파일은 두 개의 데이터 업데이트들 U1 및 U2가 행해졌기 때문에 메모리에 저장된 폐용 데이터 그룹들(회색 부분들)을 포함한다. 결국, 파일을 저장하는데 사용되는 메모리 용량은 도 14e로부터 명백한 바와 같이 파일의 크기보다 실질적으로 크다. 그러므로 가비지 수거가 적합하다. 도 15는 도 14e의 데이터 파일을 가비지 수거한 결과를 도시한 것이다. 이 파일은 가비지 수거 전에, 거의 5개의 블록들의 저장용량(도 14e)을 취하나, 가비지 수거 후에 같은 파일은 약간 3개 이상의 메모리 셀 블록들(도 15) 이내로 맞게 된다. 가비지 수거 동작의 일부로서, 데이터는 이들이 처음에는 다른 소거된 블록들에 기입되고 이어서 원 블록들은 소거되는 블록들로부터 카피된다. 전체 파일이, 수거된 데이터이라면, 이의 데이터는 파일 내 데이터 논리 오프셋 순서와 동일한 물리적 순서로 새로운 블록들에 카피될 수 있다. 예를 들면, 업데이트들 U1 및 U2, 및 삽입 I1은 이들이 호스트 파일 내 나타나는 순서와 동일한 순서로 가비지 수거(도 15) 후에 저장된다.

[0126] 또한, 가비지 수거에 의해서 정상적으로는 통합된 파일 내 새로운 서로 다른 데이터 그룹들이 형성된다. 도 15의 경우에, 파일은 새로운 데이터 그룹들에 대해 인덱스 엔트리들의 새로운 시퀀스 (F0,D0), (F1,D1), (F2,D2), (F3,D3)에 의해 기술된다. 이것은 도 14e에 도시된 파일의 상태로 존재하는 것보다 훨씬 더 적은 수의 데이터 그룹들이다. 이제 파일의 데이터가 카피된 메모리 셀 블록들 각각에 대해 하나의 데이터 그룹이 있다. 가비지 수거 동작의 일부로서, 파일 인덱스 테이블(FIT)은 파일을 형성하는 새로운 데이터 그룹들을 반영하게 업데이트된다.

[0127] 파일이 가비지 수거를 위한 후보일 때, 이 파일에 대한 FIT 데이터 그룹 엔트리들은 파일이 가비지 수거를 위한 설정된 기준을 충족하는지를 판정하기 위해 검사된다. 가비지 수거는 파일의 데이터를 내포하는 메모리 셀 블록들 중 어느 것이 폐용 데이터를 내포할 때 진행될 것이다. 그렇지 않다면, 가비지 수거는 필요하지 않다. 도 14e의 파일이 폐용 데이터를 내포하는 것은 위에 주어진 데이터 그룹 인덱스 엔트리들의 시퀀스로부터 명백하다. 예를 들면, 2개의 연속한 데이터 그룹들 (F7,D5) 및 (F8,D3)로부터, 파일 데이터가 저장되는 제1 두 개의 블록들에 폐용 데이터가 존재함을 알 수 있다. 물리 어드레스 위치들(D5,D3)에서 차이는 논리 오프셋들(F7,F8)에서의 차이보다 훨씬 크다. 또한, 파일 인덱스 데이터 그룹 엔트리들로부터 데이터는 논리 오프셋 순서로 저장되지 않았음이 명백하다. 대안적으로, 파일에 대한 개개의 FIT 엔트리들은 폐용로서 참조된 이 파일의 폐용 데이터 그룹들의 기록들을 보존할 수도 있다. 그러면 제어기는 간단히, 어떤 폐용 데이터 그룹들 및 이들이 저장된 물리 블록들을 식별하기 위해 각 파일에 대해 FIT 엔티리들을 스캔한다.

[0128] 가비지 수거는 통상적으로 호스트가 또 다른 폐용 데이터를 생성하고 및/또는 논리 오프셋 순서에서 벗어나 데이터를 저장하는 업데이트들 혹은 삽입들을 파일에 행하기를 계속할 수 있기 때문에 열려진 파일에 대해 수행되지 않아야 한다. 그러면 많은 이러한 가비지 수거 동작들이 결과로 나타날 수도 있을 것이다. 그러나, 소거된 풀 블록들의 수가 설정 레벨 미만인 경우에, 열린 파일들의 가비지 수거는 새로운 데이터 혹은 다른 동작들을 저장하는데 충분한 소거된 블록들을 제공하기 위해 필요할 수도 있다.

[0129] 호스트에 의해 파일을 닫음으로써 정규로는 이 파일에 대해 가비지 수거를 고려하게 된다. 가비지 수거는 바람

직하게는 파일이 닫힌 직후에 수행되지 않고, 현 메모리 동작들에 간섭하지 않게 될 때 백그라운드에서 가비지 수거를 위해, 닫힌 파일이 메모리 제어기에 의해 스케쥴링되는 것이 바람직하다. 가비지 수거 대기열이 유지될 수도 있는데, 여기서 파일은 폐쇄후에 대기열에 추가되며, 이어서, 수행될 보다 높은 우선도의 그외 다른 메모리 시스템 동작들이 없을 때, 대기열에 가장 오랜 동안 있었던 파일이 메모리 제어기에 의해 선택되고 가비지 수거가 필요하다면 가비지 수거된다. 이러한 선택 및 가비지 수거 동작은 예를 들면 호스트로부터 Idle 명령(도 12d)의 수신에 응하여 행해질 수 있다.

[0130] 특히 큰 파일들에 있어서, 데이터를 카피하는 것이 가비지 수거의 가장 시간 소비적인 부분이기 때문에, 타스크는 잠깐동안 임의의 한 시간에 파일의 데이터의 일부만을 카피함으로써 성분들로 분할될 수 있다. 이러한 부분적 파일 카피는 다른 메모리 동작들에 인터리브될 수 있고, 호스트가 메모리 시스템에 혹은 이로부터 데이터를 전송하는 중에도 행해질 수 있다. 개개의 카피 데이터 버스트들의 길이는 소거된 블록들의 수가 어떤 지정된 수 미만으로 되는 것에 응하여 증가될 수 있다.

[0131] 목적은 호스트 시스템에서 데이터를 전송하는 주 동작들에 간섭하거나 느려지게 함이 없이 가비지 수거를 완전히 백그라운드에서 수행하는 것이다. 그러나, 이것은 특히 새로운 데이터를 프로그래밍하기 위해 소거된 블록 풀 내 소거된 가용 블록들의 수가 어떤 기설정된 최소수 미만이 된다면, 항시 가능하지 않다. 이때 가비지 수거에 우선이 부여되고 대기열 내 임의의 파일들은 추가의 소거된 블록들이 호스트 시스템으로부터 새로운 데이터를 수신하는데 사용될 수 있도록 데이터를 통합하기 위해서 우선으로서 포그라운드에서 가비지 수거될 수 있다. 대기열에 파일들이 없다면, 열린 파일들은 가비지 수거되어야 한다. 가비지 수거가 우선이 될 때, 호스트는 통상적으로 메모리 시스템으로부터 비지 상태 신호를 수신할 것이며 따라서 소거된 블록들의 적절한 수가 다시 존재할 때까지 새로운 데이터의 어떤 프로그래밍이든 중지할 것이다. 한편, 충분한 수 이상의 소거된 풀 블록들이 있다면, 가비지 수거 동작들의 빈도는 감소될 수 있고, 메모리 시스템의 수행에 영향을 미칠 수도 있을 가비지 수거는 연기된다.

[0132] 가비지 수거가 호스트 데이터 파일들에 관해 수행되는 것에 유의할 것이다. 가비지 수거는 파일의 상태에 응하여 파일에 대해 개시된다. 개시되었을 때, FIT에 파일의 모든 데이터 그룹들의 인덱스 엔트리들은 프로세스의 일부로서 검사된다. 파일이 가비지 수거될 때, 이의 데이터는 파일에 대한 데이터 그룹 인덱스 엔트리들에 명시된 순서로, 이들의 기존 데이터로부터 새로이 열린 카피 블록에 한번에 한 데이터 그룹으로 카피된다. 이것은 전적으로 개개의 메모리 셀 블록들의 상태에 기초하는 현존 플래시 메모리 가비지 수거와는 반대이다.

[0133] 그러나, 폐용 데이터를 내포하는 파일의 데이터를 저장하는 개개의 블록들만이 가비지 수거될 것이라는 것이 일반적인 규칙이다. 따라서 파일의 데이터를 저장하는 모든 블록들이 가비지 수거되지는 않을 것이다. 파일이 예를 들면 단지 두 블록들에 저장되고, 제1 블록이 폐용 데이터를 내포하고 제2 블록이 내포하지 않는다면, 제1 블록은 가비지 수거될 것이며 제2 블록의 데이터는 그대로 놔둔다. 유효 데이터는 제1 블록으로부터, 소거된 카피 블록으로 카피되고, 그러면 카피 블록은 어떤 소거된 페이지들을 갖게 될 것이며 혹은 이에 남겨진 다른 용량은 거의 폐용 데이터량이다. 한 블록 미만의 소거된 저장용량의 사용을 이하 기술한다. 도 14e의 예에서, 가비지 소거 전에 메모리 공간은 파일의 데이터를 내포하는 5개의 블록들 중 4개의 각각에 폐용 데이터(회색 영역들)를 갖는다.

[0134] 폐용 데이터를 내포하는 블록들만이 가비지 수거된다는 일반적인 규칙의 수정으로서, 일단 주어진 파일이 가비지 수거될 것으로 결정되면, 부분적으로 채워진 블록 내 파일의 임의의 데이터는 가비지 수거 동작에 포함된다. 그러므로, 도 14e의 제5 블록 내 U2 데이터는 그 블록 내에 폐용 데이터가 없을지라도 가비지 수거 동작에 포함된다. 모든 5개의 블록들 내 데이터는 제5 블록의 데이터를 포함시키지 않음으로서, 결과적인 4개의 카피 블록들 중 두 개가 데이터로 단지 부분적으로 채워질 것이기 때문에 4개의 소거된 블록들에 카피됨으로써 함께 통합된다. 도 15의 경우에, 단지 하나의 카피 블록만이 부분적으로 채워진 상태에 있게 된다. 메모리 수행은 소수의 부분적으로 사용된 블록들을 구비함으로써 개선된다.

[0135] 호스트 시스템에 의해 발행된 도 12b의 어떤 파일 명령들은 가비지 수거를 개시할 수 있다. 파일에 대한 Close 명령의 수신은 이미 기술되었다. 닫혀진 파일은 가비지 수거될 대기열에 놓여진다. Delete 및 Erase 명령들은 가비지 수거를 야기할 수 있다. 파일의 삭제는 폐용 파일 데이터를 내포하는 블록들이 가비지 수거를 위해 대기열에 놓여지게 할 수 있다. 삭제된 파일을 가비지 수거하는 효과는 삭제된 파일의 유효한 데이터가 전혀 없고, 따라서 다른 블록들에 데이터 카피는 일어나지 않는다는 것이다. 삭제된 파일의 데이터만을 내포하는 모든 블록들은 가비지 수거 프로세스의 일부로서 간단히 소거된다. Erase 명령은 소거된 파일의 폐용 데이터만을 내포하는 블록들의 가비지 수거가 즉시 혹은 이를테면 가비지 수거 대기열의 헤드에 블록들을 놓아둠으로써 우선도에

기초하여 행해질 수 있는 것을 제외하고 유사한 효과를 갖는다.

[0136] 논의된 직접 파일 저장 시스템의 현저한 이점은 호스트가 파일을 삭제할 때 이를 행하는 명령이 직접 메모리 시스템에 가기 때문에 그 때를 즉시 메모리가 안다는 것이다. 그러면 메모리 제어기는 메모리의 다른 동작들에 악영향을 미치지 않고 행해질 수 있는 한 곧 삭제된 파일의 데이터를 저장하는 블록들을 소거할 수 있다. 그러면, 소거된 블록들은 새로운 데이터의 저장에 사용될 수 있게 된다.

[0137] 기술된 구현에서, 파일을 삭제 혹은 소거하는 명령들 중 어느 하나는 그 파일의 데이터가 직접적인 응답으로서 소거되게 한다. 한편, 전형적인 논리 기반의 인터페이스에서, 이러한 명령은 직접 메모리 제어기에 도달하지 않는다. 그보다는, 호스트는 삭제된 혹은 소거된 파일에 의해 점유되었던 메모리 논리 어드레스 공간의 어떤 세그먼트들만을 일소한다. 나중에 호스트가 이들 동일 논리 어드레스들 중 하나 이상에 데이터를 기입할 때만 메모리 시스템은 재사용된 논리 어드레스 세그먼트들에 데이터가 폐용되었음을 안다. 메모리는 여전히 파일에 의해 점유된 다른 논리 어드레스 세그먼트들의 데이터가 삭제 혹은 소거되었음을 알지 못한다. 메모리는 단지, 이들 논리 세그먼트들에 전에 기입된 데이터가 폐용된 이들 다른 논리 어드레스들에 호스트가 언제 데이터를 기입하는지를 안다.

[0138] 공통 블록

[0139] 연속한 블록들을 단일 파일에 대한 데이터로 채우는 결과는 파일이 닫혀지고 가비지 수거될 때, 파일 데이터 중 일부가 메모리 셀 블록의 일부만을 점유할 수 있다는 것이다. 또한, 작은 파일의 데이터는 전체 블록을 채울 수도 없다. 이것은 한 파일의 데이터만이 블록에 저장될 수 있다면 개개의 블록들의 공간의 상당량이 미사용으로 되는 결과가 될 것이다. 개개의 블록들 내 페이지들의 저장용량은 소거된 상태에 있게 될 것이고 데이터를 저장하는데 사용될 수 있게 되나 이용되지는 않을 것이다.

[0140] 그러므로, 일부 메모리 셀 블록들은 바람직하게는 전체 작은 파일들 및/이거나 다른 블록들이 채워진 후에 남겨진 잔여 파일 데이터로서 2개 이상의 파일들 각각으로부터 데이터 중 더 적은 량들을 저장한다. 잔여 파일 데이터는 전체 블록을 점유하지 않는 닫혀진 파일의 데이터이며, 하나 이상의 데이터 그룹들을 포함할 수 있다. 파일 맵은 블록 내 모든 데이터 그룹들이 한 파일의 전부인처럼하는 것과 동일 방식으로 단일 메모리 셀 블록 내 1이상의 파일의 데이터 그룹들을 주시할 수 있다. 개개의 데이터 그룹들의 맵은 데이터 그룹이 일부인 fileID를 포함한다. 공통 블록들을 사용하는 주된 목적은 위에 기술된 바와 같이 메모리에 파일들의 데이터를 기입하는 것에 기인할 수 있는 미사용되는 물리 메모리 저장 용량을 최소화하는 것이다. 가장 공통적으로, 파일의 가비지 수거에 기인한 하나 이상의 데이터 그룹들의 잔여 데이터는 가비지 수거의 일부로서 공통 블록에 기입될 것이다. 예는 도 15의 가비지 수거된 파일이며, 여기서 마지막 데이터 그룹 (F3,D3)은 마지막 블록의 작은 부분만을 점유한다. 이러한 식으로 된 상태에 있다면, 이 마지막 블록의 대부분은 미사용으로 될 것이다. 도 15의 데이터 그룹 (F3,D3)은 또 다른 하나 이상의 파일들로부터 하나 이상의 데이터 그룹들을 내포하는 공통 블록에 기입될 수 있고, 이 마지막 메모리 블록은 공통 블록으로서 지정될 수 있다. 나중의 경우에, 하나 이상의 다른 파일들로부터 데이터 그룹들은 후속하여 이 동일 블록에 기입될 것이다.

[0141] 대안적으로, 잔여 파일 데이터는 잔여 데이터량이 지정된 공통 블록들 중 한 블록 내 소거된 공간에 들어맞게 될 량인 것으로 알려졌을 때 가비지 수거 동안 호스트로부터 직접 공통 블록에 기입될 수 있다. Close_after 명령(도 12b)은 잔여 파일 데이터를 식별하고 완전히 채워지지 않게 될 소거 풀 블록에 잔여 파일 데이터를 기입하기보다는, 명령의 일부로서 명시된 데이터량을 저장하는데 충분한 공간을 갖는 열린 공통 블록에 상기 잔여 파일 데이터가 기입되게 하는데 사용될 수 있다. 이것은 후속의 가비지 수거 동작 동안에 잔여 데이터를 카피할 필요성을 제거할 수 있다.

[0142] 새로운 데이터 파일이 메모리에 프로그램될 때, 데이터는 전에 기술되고 도 13a에 도시된 데이터 기입방법에 대한 대안으로서, 블록 내 제1 가용 물리적 위치에서 시작하여 블록의 위치들을 순차로 순서대로 가면서, 하나 이상의 다른 파일들에 대한 잔여 데이터를 내포하는 열린 공통 블록에 기입될 수 있다. 호스트가 파일에 대한 Open 명령을 보내고 바로 이어 파일에 대한 데이터를 보내기 전에 Close_after 명령이 보내진다면, 파일에 대한 전체 데이터는 열린 공통 블록 내에 들어맞을 수 있고 열린 공통 블록의 끝에 도달되기 전에 파일이 닫혀질 것으로 판정될 수 있다. 새로운 파일에 대한 데이터는 파일 데이터의 길이를 모를지라도 열린 공통 블록에 기입될 수도 있다.

[0143] 그러므로, 다수의 공통 블록들은 바람직하게는 2이상의 서로 다른 파일들의 공통 데이터 그룹들을 저장하기 위

한 메모리 시스템에 유지된다. 공통 데이터 그룹은 한 바이트의 입도를 갖고, 블록의 저장용량의 크기까지의 크기를 가질 수 있다. 공통 블록들 중 단지 하나만이 바람직하게는 주어진 파일의 데이터를 내포하나 파일의 2이상의 데이터 그룹을 내포할 수 있다. 또한, 공통 데이터 그룹은 단지 하나의 공통 블록에 저장되는 것이 바람직하며, 복수 공통 블록들 내 저장을 위해 분할되지 않는다. 이것은 데이터 그룹이 폐용으로 될 때 복수의 공통 블록들에 가비지 수거를 회피한다. 공통 블록은 데이터 그룹이 폐용으로 될 때 가비지 수거를 받는다. 이러한 공통 블록 내 어떤 남은 유효 데이터 그룹(들)은 또 다른 공통 블록 내 소거된 가용 공간에 기입되거나 소거된 풀 블록에 기입되며, 이어서 공통 블록이 소거된다. 가비지 소거된 공통 블록이 서로 다른 파일들로부터 2이상의 유효 데이터 그룹들을 내포한다면, 이들은 함께 유지될 필요는 없고 그보다는 서로 다른 블록들에 카피될 수 있다.

[0144] 공통 블록의 예는 도 16에 도시되었고, 3개의 서로 다른 데이터 파일들 각각으로부터 공통 데이터 그룹으로 프로그램되어 있다. 데이터를 기입하는 것이 NAND 어레이들에서는 블록의 한 끝에 페이지들부터 다른 끝으로 진행하도록 되어 있기 때문에, 데이터 그룹들은 서로 인접하게 저장된다. 블록의 끝에 백색의 공간은 이들에 어떤 데이터도 기입되지 않은 블록의 페이지들을 나타낸다. 사용가능한 데이터 그룹들을 전체 공통 블록들에 완벽하게 들어맞게 하는 것은 어렵다.

[0145] 1 이상 파일의 잔여 데이터가 기입될 수 있는 열린 공통 블록들로서 메모리 시스템에 의해 지정된 블록들의 수는 정상적으로는 단지 소수일 것이지만 가비지 수거된 많은 수의 파일들이 어떤 현존의 열린 공통 블록 내 가용 공간에 들어맞지 않는 잔여 파일 데이터를 갖는다면 많게 될 수 있다. 잔여 파일 데이터는 전체 메모리 용량을 최상으로 이용하는 열린 공통 블록들 중 한 블록에 기입된다. 열린 공통 블록들 중 한 블록에 충분한 소거된 공간이 없을 때, 현존하는 공통 블록들 중 하나는 닫혀지고 또 다른 공통블록이 그 대신에 열린다. 대안적으로, 현존하는 열린 공통블록은 닫혀질 필요는 없고, 열린 공통 블록들의 수가 증가되게 할 수 있다. 새로운 공통 블록은 소거된 풀 블록들 중 하나로서 지정될 수 있으나 바람직하게는, 잔여 파일 데이터 그룹 (F3,D3)만을 내포하는 도 15에서 마지막 블록과 같이, 사용할 수 있게 되었을 때, 이미 어떤 잔여 파일 데이터를 내포하는 불완전하게 기입된 블록이다. 그러나, 공통 블록의 가비지 수거에 있어서, 소거된 풀 블록은 바람직하게는 필요시 새로운 공통 블록으로서 지정된다.

[0146] 도 17은 또 다른 파일로부터 하나 이상의 데이터 그룹들의 잔여 데이터를 이미 내포하는 5개의 잔여 혹은 공통의 단위들(각각 상이한 블록 혹은 메타블록)이 있는 경우에 잔여 파일 데이터를 기입하는 프로세스의 예를 도시한 것이다. 도 15의 가비지 수거된 파일에 기인한 마지막 데이터 그룹 (F3,D3)은, 단일 파일로부터 1 이상 데이터 그룹을 포함할 수도 있을지라도, 이러한 잔여 파일 데이터의 예이다. 도시된 3가지 가능성들이 있다. 제1 가능성 (A)는 가용한 가장 많은 소거된 공간을 갖고 있기 때문에 잔여 단위 2에 잔여 데이터를 기입한다. 제2 가능성 (B)는 잔여 파일 데이터에 대해 잔여 단위 5가 5개의 잔여 단위들 중에 최상으로 들어맞기 때문에 이 잔여 단위 5를 선택한다. 잔여 파일 데이터는 단위 5를 거의 채우며 그럼으로써 추후에 더 많은 량의 데이터를 수신하기 위해 단위 2에 가용한 더 큰 공간을 남긴다. 단위 1, 단위 3 혹은 단위 4 중 어느 것에도 잔여 데이터를 취할 충분한 공간이 없고 따라서 이들 단위들은 바로 제외된다.

[0147] 제3 가능성 (C)은 잔여 파일 데이터를 소거 풀로부터 블록 혹은 메타블록에 기입한다. 일단 잔여 파일 데이터가 완전히 소거된 단위에 기입되었으면, 이것은 잔여 단위가 되며, 나중에 공통블록으로서 메모리 시스템 제어기에 의해 열릴 수 있다. 도시된 예에서, 잔여 데이터 파일에 대한 잔여 단위들 2 혹은 5 중 어느 하나에 여유가 있기 때문에 통상적으로 가능성 (C)는 사용되지 않을 것이다.

[0148] 대안적으로, 파일에 대한 잔여 데이터는 서로 다른 공통블록들 내 저장을 위해 2이상 부분들로 분할되게 할 수 있다. 예를 들면, 어떠한 현존의 열린 공통블록도 특정 파일에 대한 잔여 데이터의 저장을 위한 충분한 가용 공간을 갖고 있지 않을 수 있고 어떠한 소거된 블록도 잔여 파일 데이터에 대한 새로운 공통 블록으로서 열려지게 하는데 사용될 수 없다. 이 경우, 잔여 파일 데이터는 2이상의 열려진 공통 블록들간에 분할될 수 있다.

[0149] 공통블록은 이것이 데이터로 충만되기 전에 닫혀질 수 있다. 위에 언급된 바와 같이, 최소량의 가용한 소거된 공간을 가진 열린 공통블록은 일반적으로 파일의 주어진 량의 잔여 데이터를 저장하기 위해서 또 다른 공통 블록을 여는 것이 필요할 때 닫혀질 것이다. 이것은 부분적으로는 파일에 대한 잔여 데이터가 서로 다른 공통 블록들에 저장을 위해 분할되지 않는 바람직한 동작의 결과이다. 메모리 시스템이 새로운 데이터에 대한 가용 메모리 저장 공간의 량을 호스트의 Size 명령(도 12e)에 응답함으로써 호스트에 보고할 때, 닫혀진 공통 블록들 내 미사용된 소량의 이들 용량은 이들이 즉시로 사용될 수 없기 때문에 포함되지 않는다.

[0150] 일부가 삭제 혹은 소거되는 파일들에 기인하여 잔여 파일 데이터가 폐용이 되기 때문에, 공통블록들 내 데이터

는 또한 통합된다. 예를 들면, 공통 블록 내 잔여 파일 데이터가 폐용으로서 지정될 때마다, 이유가 무엇이든 간에, 이 블록은 위에서 논한 폐용 파일 데이터 블록 대기열 혹은 또 다른 대기열에 추가된다. 파일에 대한 호스트의 Delete 명령 때문에 데이터가 폐용된다면, 공통블록은 대기열의 끝에 놓여진다. 그러나, 이것이 Erase 명령에 기인한다면, 공통블록은 대기열의 헤드로 가고 그렇지 않다면 가비지 소거에 대한 우선이 부여된다.

[0151] 단일 대기열 대신에, 메모리 시스템의 동작동안 제어기에 의해 유지되는 5개의 서로 다른 가비지 수거 대기열들이 있을 수 있고, 첫 번째 2개의 대기열들 내 엔트리들에 우선이 부여된다.

[0152] (1) 폐용 블록들의 우선 대기열, 즉 파일에 대한 Erase 명령(도 12b)의 결과로서 폐용 데이터만을 내포하는 블록들;

[0153] (2) 파일에 대한 Erase 명령에 의해 폐용으로 된 데이터를 내포하나 일부 유효 데이터도 내포하는 공통 블록들의 우선 대기열;

[0154] (3) Update 혹은 Delete 명령들(도 12a 및 도 12b)의 실행에서 비롯되거나 가비지 수거동안 모든 유효 데이터가 또 다른 블록에 카피되었기 때문에 비롯되는 폐용 블록들(폐용 데이터만을 내포하는 블록들)의 대기열;

[0155] (4) Delete 명령에 응하여 혹은 가비지 수거동안 공통 블록에 폐용 데이터가 존재한다는 발견에 응하여, 어떤 폐용 데이터를 내포하나 유효 데이터도 내포하는 공통 블록들의 대기열;

[0156] (5) Close 명령이나 Close_after 명령이 수신된 파일들의 대기열.

[0157] 이들 5개의 큐들에는 위에 열거된 순서로 우선이 부여될 수 있고, 대기열 (1) 내 모든 아이템들은 대기열 (2)의 것들 전에 가비지 수거되고, 등등이 행해진다. 블록들은 이들의 데이터의 전부 혹은 일부가, 각각, Delete 명령에 의한 것이 아니라 Erase 명령에 의해 폐용으로 되었기 때문에, 우선 대기열들 (1) 및 (2)에 리스트된다. 블록들 및 파일들은 메모리 시스템의 동작동안 확인된 순서로 대기열들의 각각에 부가되는데, 각 대기열 내 가장 오래된 것이 먼저 가비지 수거된다(선입선출, 또는 FIFO).

[0158] 대기열들 (4) 및 (5)에 리스트된 항목들의 우선은 거의 같고, 따라서 역으로 될 수도 있을 것이다. 대안적으로, 파일들 및 공통 블록들이 아마도 하나 이상의 더 높은 우선 대기열들이 비어있기도 전에, 설정된 기준에 따라 대기열들 (4) 및 (5)로부터 선택되는 보다 복잡한 우선 시스템일 수 있다. 공통 블록들의 가비지 수거를 관리하는 2가지 목적들이 있다. 하나는 공통 블록 내 데이터 그룹이 폐용이 될 때 공통 블록의 가비지 수거에 기인할 수 있는 메모리 시스템의 정규 동작에 중단을 최소화하는 것이다. 다른 하나는 공통 블록으로부터 데이터 그룹이 블록의 가비지 수거동안 재배치될 때 인덱싱 업데이트들을 최소화하는 것이다.

[0159] 공통 블록의 가비지 수거 동안에, 블록 내 남은 유효 데이터의 모든 공통 그룹들은 공간을 갖는 공통 블록들 중 하나에 한번에 하나가 카피된다. 카피되는 공통 그룹을 위한 열려진 공통 블록에 여유가 없다면, 공통 그룹은 소거 풀로부터 블록에 기입되고 이 블록은 이때 공통 블록으로서 지정될 수 있다. 그러면 통상적으로 열려진 공통 블록들 중 하나는 프로세스의 일부로서 닫혀진다. 모든 가비지 수거 동작들에서와 같이, 파일 인덱스 테이블(FIT)은 카피된 데이터 그룹들의 새로운 저장 위치들을 반영하게 업데이트된다.

[0160] 가비지 수거 대기열들은 비휘발성 메모리에 기록될 것이다. 대기열 정보를 내포하는 페이지 혹은 메타페이지에 대한 독출-수정-기입(read-modify-write) 동작은 엔트리를 추가하거나 제거하기 위해 수행되어야 한다. 가비지 수거 대기열 정보를 내포하는 페이지들은 전용 블록 혹은 메타블록에 있을 수 있고, 혹은 스왑 블록과 같은 다른 유형들의 페이지들과 블록 혹은 메타블록을 공유할 수도 있다.

[0161] 프로그래밍 동안에 메타페이지들의 버퍼링 및 사용

[0162] 전술한 바는 메모리 셀 블록들이 함께 메타블록들에 링크되어 있는지 여부에 특별히 관계없이 메모리 시스템의 동작을 기술한 것이다. 호스트와 메모리 시스템간의 파일 기반의 인터페이스, 및 위에 기술된 관계된 메모리 시스템 동작은 메타블록들을 사용하는 메모리 시스템, 및 이를 사용하지 않는 메모리 시스템에서도 동작한다. 경향은 한번에(병행도) 기입 및 독출되는 이것이 곧바로 메모리 수행을 향상시키기 때문에 한번에 독출 및 기입되는 데이터량을 확실히 증가시키는 것이다. 다수의 데이터 비트들에 대해서 개개의 메모리 셀 블록들의 유효 폭은 2이상의 이러한 블록들로 형성된 메타블록들의 사용에 의해 증가된다.

[0163] 도 3을 참조하면, 예를 들어, 단일 블록 내 워드라인들 각각을 따른 모든 메모리 셀들은 페이지로서 함께 프로그램 및 독출될 수 있고, 잠재적으로 각 행에 몇몇의 1, 2, 4 혹은 그 이상의 섹터들의 512 바이트의 사용자 데

이터를 저장할 수 있다. 또한, 2이상의 블록들이 논리적으로 메타블록에 링크될 때, 2이상 블록들 각각의 한 행에 모든 메모리 셀들은 함께 프로그램 및 독출될 수 있다. 함께 프로그램 및 독출되는 2이상 블록들로부터 2이상 페이지들은 메타페이지를 형성한다. 한번에 훨씬 많은 데이터를 프로그램하는 능력에 있어, 메타페이지 내 데이터의 어떤 스테이징(staging)은 이러한 가능한 병행도를 완전히 이용하는데 도움을 줄 수 있다.

[0164] 3개의 호스트 파일들 각각에 대한 한 메타페이지의 데이터가 도 18에 도시되었고, 각 메타페이지는 간략성을 위해 단지 두 페이지들만을 포함하는 것으로 도시되었다. 메타페이지의 각 페이지의 데이터는 상이한 블록의 메타블록에 프로그램된다.

[0165] 호스트 파일 1에 대해서, 메타페이지는 도 18a에는 단일 데이터 그룹 0의 일부로서, 인접한 오프셋들(논리 어드레스들)을 갖는 데이터로 채워진 것으로 도시되었다. 이들 데이터는 파일 1에 데이터를 수신하는 메타블록 내 순서적으로 다음 메타페이지에 병렬로 프로그램된다. 도 18b에서, 이 예의 호스트 파일 2는 이의 제1 페이지의 부분이 연속된 오프셋들을 가진 데이터 그룹 1의 부분을 포함하고 메타페이지의 나머지가 연속된 오프셋들을 갖는 또 다른 데이터 그룹 2의 부분을 내포하는 점에서 다른 것으로 도시되었다. 2개의 데이터 그룹들이 파일 2 메타페이지 내에서 결합하는 데이터의 오프셋들에서 불연속이 있을 수 있을지라도, 이들 데이터는 단일 메타페이지에서 함께 프로그램된다. 앞에서 기술된 바와 같이, 직접 파일 저장 인터페이스는 파일 내 데이터의 오프셋들의 순서에 관계없이, 호스트 파일의 데이터가 호스트로부터 수신될 때 이를 기입한다.

[0166] 어떤 플래시 메모리들은 소거된 페이지에 데이터의 프로그래밍을 1번 이상 허용하지 않는다. 이 경우에, 도 18b의 데이터 그룹 1과 데이터 그룹 2 둘 다는 동시에 프로그램된다. 그러나, 메모리가 부분적 페이지 프로그래밍을 허용한다면, 파일 2의 2개의 데이터 그룹들은 서로 다른 시간들에서 비휘발성 메모리의 단일 메타페이지에 프로그램될 수 있다. 이 경우, 페이지 0은 서로 다른 시간들에서 프로그램될 것이며, 우선은 데이터 그룹 1에 대해 프로그램되고 이어서 데이터 그룹 2가 페이지 0의 나머지에 그리고 페이지 1의 전부에 프로그램된다. 그러나, 통상적으로는 시스템 성능을 증가시키기 위해서 두 데이터 그룹들을 병렬로 프로그램하는 바람직하다.

[0167] 도 18c에서, 단일 데이터 그룹 3의 끝은 파일 3에 대한 메타페이지에 기입된 것으로 도시되었다. 이러한 소량의 데이터가 한 메타페이지의 비휘발성 메모리에 프로그램되고 이 동안 이 메타페이지의 나머지 부분들이 소거된 상태에 있을 때 어떤 상태들이 있다. 하나는 데이터 그룹 3이 잔여 파일 데이터인 파일에 대해 호스트가 Close 명령(도 12b)을 발행할 때이다. 파일 3은 호스트가 허용된 수를 초과하여 다수의 파일들을 열려고 시도하고 또 다른 보다 활성적인 파일이 대신 열려지게 하기 위해서 파일 3이 닫기 위해 선택된다면 메모리 제어기에 의해 닫혀질 수도 있을 것이다. 메모리 제어기는 열려져 있는 모든 파일 메타페이지 버퍼들을 위한 버퍼 메모리에 불충분한 용량이 있다면 파일 3을 닫을 수도 있을 것이다. 이들 경우들 중 어느 것이든, 파일 버퍼의 부분적 데이터 콘텐츠를 비휘발성 메모리에 곧 기입하는 것이 바람직하다(파일 버퍼를 "플러시(flush)"한다). 호스트는 Shut-down 명령(도 12d)을 보낼 수 있는데, 이것은 파워가 메모리에 대해 상실될 수도 있을 것이고 이의 비휘발성 버퍼 메모리 내의 모든 데이터가 비휘발성 메모리에 곧 프로그램되지 않는다면 유실될 수도 있을 것임을 의미한다.

[0168] 도 18c의 메모리 메타페이지에 파일 3의 또 다른 데이터 그룹 4의 시작부분을 기입하는 것이 나중에 요구되고 메모리가 부분적 페이지 프로그래밍을 허용하지 않는다면, 데이터 그룹 4의 제1 페이지는 도 18d에 도시된 바와 같이, 파일 3 메타페이지의 제2 페이지에 기입된다. 이것은 비휘발성 메모리 메타페이지 내에, 도 18d의 데이터 그룹 3과 데이터 그룹 4 사이에 공백 부분으로 나타낸 바와 같이, 어떤 미사용된 메모리 용량이 생기게 할 수 있다. 이 미사용된 용량은 파일 3이 닫혀진 후에 수행되는 가비지 수거 동작에 의해 복구될 수 있고, 혹은 이 상황이 드물게 일어날 것이기 때문에 존속되게 할 수도 있다.

[0169] 본 발명의 직접 파일 저장 기술이 비휘발성 메모리의 블록들 내에 이러한 채워지지 않은 갭들을 묵인할 수 있다는 것에 유의하는 것이 중요하다. 이러한 갭들은 현 시스템들에 의해 쉽게 묵인될 수 없고, 여기서 메모리 시스템은 도 7 및 도 8에 도시된 바와 같이, 논리 어드레스 공간을 통해 호스트와 인터페이스한다. 논리 데이터는 메모리 블록들과 동일 크기를 그룹화하거나 메타블록들은 이러한 블록들 혹은 메타블록들에 맵핑된다. 갭이 현존의 메모리 시스템의 블록 혹은 메타블록 내에 저장된 데이터에 있었다면, 갭에 맵핑되는 논리 데이터 어드레스들은 호스트에 의해 효과적으로 사용될 수 없을 것이다. 호스트는 이러한 현존의 인터페이스에서 이것이 사용할 수 있는 전체 논리 어드레스 공간을 갖는 것으로 가정하나, 갭을 지정하는 논리 어드레스들에 새로운 데이터가 기입되는 것은, 가능할지라도, 매우 어렵다.

[0170] 도 2의 메모리(31)와 같은, 시스템 제어기 내 버퍼 메모리의 부분은 통상적으로 프로그래밍 데이터 버퍼들로서 이용된다. 이 메모리 내 버퍼 용량은 호스트에 의해 기입되는 각각의 활성 파일을 위해 존재해야 한다. 각각의

이러한 "파일 버퍼"는 플래시 메모리 내 적어도 한 메타페이지와 동일한 용량을 가져야 한다. 활성 파일은 데이터가 최근에 호스트에 의해 기입이 된 열린 파일이다.

[0171] 메모리 시스템이 어떤 한 시간에 취급하는 활성 호스트 파일들의 수는 현재 열린 메모리 시스템 파일들의 수와 같을 수도 있고, 혹은 그 미만의 수일 수도 있다. 활성 파일들의 허용된 최대 수가 열린 파일들의 허용된 최대 수 미만이면, 파일의 상태를 활성과 열림간에 및 그 반대로의 변경을 수 있을 것이 제공되어야 한다. 이것을 허용하기 위해서, 플래시 메모리 내 임시 저장 블록이 스왑 블록으로서 지정되고, 길이가 한 메타페이지 미만인 데이터는 파일 버퍼로부터 스왑 버퍼 내 메타페이지에 기입될 수 있고 그 반대로도 기입될 수 있다. 유효한 스왑된 파일 버퍼들의 인덱스는 스왑 블록에 유지된다. 파일 버퍼 데이터는 정수개의 페이지들로서 스왑 블록에 카피되고, 이어서 단일 페이지가 각각의 카피된 파일 버퍼의 길이 및 위치의 인덱스, 및 이것이 관계된 파일을 제공한다. 스왑 블록은 주기적으로 콤팩트되고 충만되었을 때 소거된 블록에 기입된다.

[0172] 예를 들면, 열린 파일 A가 이에 호스트 기입 동작의 결과로서 활성이 되게 해야 할 때, 최소한으로 최근에 기입된 활성 파일 B가 확인되고, 이의 파일 버퍼 데이터는 제어기 버퍼 메모리로부터 스왑 블록 내 다음 가용 페이지들에 카피된다. 그러면 파일 A에 대한 파일 버퍼 데이터가 스왑 블록 내에서 확인되고 파일 B에 이전에 할당된 제어기 버퍼 메모리 내 가용 공간에 카피된다.

[0173] 파일 버퍼로부터 데이터는 바람직하게는 도 18에 도시된 경우들 중 한 경우에 따라 플래시 메모리에 그 파일에 대한 열린 기입 블록 내 다음 가용 메타페이지에 기입된다. 도 18a에서, 파일 1 내 단일 데이터 그룹 0의 부분을 형성하는 충분한 데이터가 파일 1 버퍼에 있을 때, 단일 동작으로 완전한 메타페이지에 프로그램된다. 도 18b에서, 데이터 그룹 1의 끝을 형성하는 충분한 데이터와 파일 2 내에 데이터 그룹 2의 시작이 파일 2 버퍼에 있을 때, 이들은 단일 동작으로 완전한 메타페이지에 프로그램된다. 플래시 메모리 디바이스가 단일 페이지 상에 복수의 프로그램 동작들(부분적 페이지 프로그래밍)을 지원한다면, 파일 2 내 데이터 그룹 1의 끝을 형성하는 충분한 데이터가 파일 2 버퍼에 있을 때, 이들은 페이지 0의 부분에 프로그램될 수 있다. 파일 2 내에 데이터 그룹 2의 시작을 형성하는 충분한 데이터가 파일 2 버퍼 내에서 가용할 때, 이들은 후속하여 별도의 프로그래밍 동작에서 동일 메타페이지의 나머지에 프로그램될 수 있다.

[0174] 도 18c에 도시된 바와 같이, 파일 3 내 데이터 그룹 3의 끝을 형성하는 데이터가 파일 3 버퍼에 있고, 버퍼 플러시 동작이 수행되어야 할 때, 이들은 페이지 0의 부분에 프로그램된다. 플래시 메모리 디바이스가 단일 페이지 상에 복수의 프로그래밍 동작들을 지원하지 않는다면(부분적 페이지 프로그래밍), 파일 3 내 데이터 그룹 4의 시작을 형성하는 충분한 데이터가 파일 3 버퍼 내에서 가용할 때, 후속하여 별도의 프로그램 동작에서 동일 메타페이지 내 페이지 1에 프로그램된다.

[0175] 메타페이지 파일 버퍼들은 또한 가비지 수거 동안 사용된다. 유효 데이터 그룹들의 파일이 비휘발성 메모리로부터 독출되고 이 파일에 대해 제어기 버퍼 메타페이지에 기입될 수 있다. 파일이 동시에 재순서화되고 있다면, 데이터 그룹들은 이들의 호스트 데이터 오프셋들의 순서로 버퍼에 기입된다. 물론, 각 데이터 그룹 내 데이터는 인접 논리 오프셋들을 갖는다. 일단 메타페이지 버퍼가 충만되면, 이의 데이터는 병렬로 비휘발성 메모리의 새로운 블록에 프로그램된다.

[0176] 파일 인덱싱

[0177] 메모리 시스템에 저장된 각 파일은 특히 도 13-16에 관하여, 위에 기술된 바와 같이 인덱스 엔트리들의 그 자신의 시퀀스에 의해 정의된다. 이들 엔트리들은 파일에 대한 데이터가 첨부되거나, 삽입되거나 업데이트될 때, 그리고 파일이 가비지 수거될 때 시간에 따라 변한다. 도 19는 몇몇의 서로 다른 시간들 0, 2, 4, 5에서 한 파일에 대한 파일 인덱스 테이블(FIT) 내 인덱스 엔트리들의 시퀀스를 도시한 것이다. 이들은 각각 도 14a, 14c, 14e, 15에 관하여 위에 기술된 시퀀스들이다. FIT 내 데이터는 바람직하게는 호스트 시스템으로부터 도움없이 메모리 제어기에 의해 기입되고 현재로 유지된다. 호스트 시스템은 메모리 시스템에 데이터를 기입할 때 경로명들, 파일명들 및 파일 내 데이터의 오프셋들을 공급하나 호스트는 데이터 그룹들을 정의하거나 이들이 메모리 셀 어레이에 저장되는 경우 관여하지 않는다. 도 19의 엔트리들에서, 도 14 및 도 15의 메모리 셀 블록들은 1부터 시작하여 좌측부터 숫자가 매겨진다. 따라서, 예를 들면 도 14c에 도시된 상태에 파일에 대해서, 이의 제3 데이터 그룹(F6,D7)은 도 19에서 좌측에서 네 번째 블록인 블록 004에 이 블록의 최초 어드레스로부터 D7 바이트가 저장되는 것에 유의한다. 각 데이터 그룹의 길이는 테이블의 각 엔트리와 함께 포함되는 것이 바람직하다.

[0178] 한 파일에 대해 도 19에 도시된 시퀀스 인덱스 엔트리들은 파일의 변경으로 파일의 데이터 그룹들이 수정되었을

때, 혹은 이외 덜 빈번한 간격들로 메모리 제어기에 의해 재기입된다. 제어기는 이의 메모리에 이러한 변경들을 저장할 수 있고, 이어서 이들 중 대부분을 한번에 플래시 메모리에 기입할 수 있다. 임의의 한 시간에 파일에 대해 단지 하나의 유효한 한 세트의 인덱스들이 존재하며, 서로 다른 시간들에서 파일을 정의하는 4개의 이러한 세트들의 인덱스들이 도 19에 도시되었다. 그러면 메모리 시스템 제어기는 파일에 추가의 데이터를 프로그램하고, 파일로부터 데이터를 독출하고, 파일의 데이터를 가비지 수거하고 잠재적으로 그외 다른 동작들을 위해서 필요한 파일의 현 한 세트의 인덱스들을 사용한다. 그러므로 FIT는 개개의 파일 인덱스 엔트리들이 이들의 파일 오프셋들(Fx)의 순서로 저장된다면 사용하기가 더 쉬우나, 그렇지 않다면, 제어기는 그 논리적 순서로 엔트리들을 확실하게 읽을 수 있다. 특정 파일에 대해 인덱스 엔트리들을 읽는 메모리 제어기의 가장 일반적인 원인은 호스트 명령의 실행하는 과정에 있다.

[0179] 도 20은 파일 맵의 일부로서 파일 인덱스 테이블(FIT)을 유지 및 사용하는 바람직한 기술을 도시한 것이다. 파일 인덱싱 구조들의 연쇄는 명시된 경로명의 파일 내 명시된 오프셋 어드레스의 데이터의 물리적 위치를 확인하는데 사용된다. 파일 맵은 통상적인 논리 어드레스 인터페이스에 사용되는 표준 디스크 운영 시스템(DOS) 루트 디렉토리 및 서브-디렉토리 구조들과 실질적으로 동일한 논리 구조를 갖는 디렉토리(201)를 포함한다. 파일 디렉토리(201)는 메모리 시스템 내에 하나 이상의 전용 플래시 블록들에 저장될 수 있다. 그러나, 파일 디렉토리(201)는 바람직하게는 호스트 대신에 저장 시스템에 의해 관리된다. 여기에서 도 12c의 호스트 디렉토리 명령들은 메모리 시스템 제어기에 의해 실행되나 그러나 제어기에 의해 결정된 방식으로 한번에 실행된다. 파일 디렉토리에 직접 기입하는 것이 호스트에 허용되지 않는 것이 바람직하다.

[0180] 각각이 디렉토리나 파일을 확인하는 것인 균일 크기의 엔트리들은 파일 디렉토리(201)에 유지된다. 한 세트의 인접한 엔트리들은 특정 디렉토리 내 요소들을 명시하는데 사용된다. 디렉토리를 명시하는 엔트리 내 포인터 필드는 이 디렉토리 내 요소들을 명시하는 다른 인접 엔트리들의 시작을 확인한다. 파일을 명시하는 엔트리 내 포인터 필드는 연관된 파일 인덱스 테이블(FIT)(203) 내에 블록 번호 및 파일 번호를 정의한다.

[0181] FIT(203)는 데이터 그룹들의 균일 크기의 엔트리들을 내포하며, 각각의 엔트리는 데이터 그룹의 시작의 오프셋과 데이터 그룹의 메모리 시스템 내 물리적 위치를 확인한다. 논리 어드레스들이 바이트 입도로 유지된다면, 각 FIT 엔트리의 오프셋은 데이터 그룹의 시작을 지정하는 파일의 시작부터 명시된 수의 바이트들을 내포한다. 마찬가지로, 데이터 그룹의 시작의 물리적 위치는 바이트 입도로 명시될 수 있다. 서로 다른 시간들에서 전형적인 파일에 대한 FIT 엔트리들의 내용들이 도 19에 관련하여 위에 기술되었다. 각 데이터 그룹마다 별도의 엔트리가 유지된다. 각 파일은 파일 내 데이터 그룹들의 한 세트의 인접한 인덱스 엔트리들에 의해 정의된다. 파일 당 이러한 엔트리들의 수는 통상적으로 달라질 것이다.

[0182] 파일에 대해 호스트에 의해 공급되는 경로명은 FIT 내에서 블록 번호 및 파일 번호를 얻기 위해서 파일 디렉토리의 계층을 조사하는데 사용된다. 통상적으로 경로명은 하나, 둘, 혹은 그 이상의 디렉토리들 및 서브-디렉토리들을 포함한다. 이들은 액세스할 파일을 내포하는 파일 디렉토리(201) 내의 디렉토리에 액세스하는데 사용된다. 그러면 이 디렉토리 내 파일명들은 호스트에 의해 공급된 파일명에 대한 엔트리(205)를 찾기 위해 탐색된다. 발견되었을 때, 엔트리(205)는 이 파일에 대한 FIT(203) 내 한 그룹의 인접한 엔트리들에의 포인터를 제공한다. 이들 엔트리들은 도 19에 관하여 기술된 것들과 유사하다. 이어서 이들 엔트리들의 오프셋들은 정확한 엔트리(207), 및 이에 따라 정확한 데이터 그룹을 확인하기 위해서 호스트에 의해 공급된 오프셋 어드레스와 비교된다. 오프셋들의 동일한 일치가 전혀 없다면, 이것은 흔히 엔트리들 내 포함된 오프셋들이 데이터 그룹들의 시작 어드레스들일 뿐이기 때문에 그럴 수 있는 것으로, 오프셋이 내포된 데이터 그룹을 확인하는 엔트리가 선택된다. 이 예에서, FIT(203)의 선택된 엔트리(207)는 호스트가 액세스하고자 하는 데이터를 내포하는 물리 블록 및 한 바이트의 메모리 위치를 내포한다.

[0183] 파일에 대한 도 20의 디렉토리 엔트리(205)의 포인터는 파일이 처음 생성되었을 때 혹은 FIT(203) 내 파일에 대한 데이터 그룹 엔트리들의 위치가 변경되었을 때 메모리 시스템 제어기에 의해 할당된다. 이 포인터는 앞서 논의된 file_handle의 예이다. 파일이 액세스될 때마다 파일에 대한 FIT 엔트리들을 찾기 위해서 호스트가 파일 디렉토리(201)의 계층을 조사해야 하는 것을 피하기 위해서, 메모리 시스템은 file_handle을 호스트에 보냄으로써 그후에, 열린 파이들의 FIT 엔트리들에 직접 액세스할 수 있게 된다.

[0184] 도 21은 파일 디렉토리(201) 내 개개의 엔트리들을 저장하는 메모리 시스템 내 페이지들을 도시한 것이다. 엔트리들의 일부는 파일 디렉토리(201) 내 디렉토리들에의 포인터들을, FIT(203)내 데이터에 다른 것들을 제공한다. 단일 메모리 페이지에 저장된 많은 것들 중 하나인, 예로서의 엔트리(209)는 각 엔트리가 4개의 필드들을 갖는 것을 예시하고 있다. 첫 번째는 호스트에 의해 할당되는 디렉토리 혹은 파일의 이름을 내포한다. 제2 필드는 호

스트에 의해 정의되는 디렉토리 혹은 파일의 속성들을 내포한다. 제3 필드에 포인터는, 디렉토리에 대한 엔트리의 경우에, 파일 디렉토리 내 또 다른 엔트리를 가리킨다. 파일에 대한 엔트리의 경우에, 제3 필드는 엔트리(207)와 같은 FIT(203) 내 파일 엔트리에의 포인터를 내포한다. "데이터 그룹들"로서 확인되는 제4 필드는 디렉토리를 위한 엔트리에선 비어있으나, 파일에 대한 엔트리의 경우에, 이 필드는 파일에 대한 데이터를 내포하는 데이터 그룹들의 수 및 따라서 파일에 대한 FIT(203) 내 엔트리들의 수를 명시한다.

[0185] 도 21에 예시된 파일 디렉토리 페이지들은 두 부분들을 내포하는 것에 유의한다. 가장 최근에 기입된 페이지(211)에는 디렉토리 엔트리들(페이지(213) 내 209로 표시된 것) 및 페이지 포인터들이 다 존재한다. 페이지(213)와 같은 다른 디렉토리 페이지들에는 디렉토리 엔트리들이 존재하나 페이지의 영역은 폐용 페이지 포인터들을 내포한다. 현 페이지 포인터들은 가장 최근에 기입된 페이지로서, 이 경우엔 페이지(211)의 동일 부분에 유지된다. 디렉토리 블록 내 각 논리 페이지에 대해 한 페이지 포인터가 존재하고 이 포인터는 메모리 제어기가 액세스하고 있는 논리 페이지에 대응하는 물리 페이지에 메모리 제어기를 안내한다. 가장 최근에 기입된 디렉토리 페이지 내 페이지 포인터들을 유지함으로써, 이들은 디렉토리 엔트리와 동시에 그리고 또 다른 페이지를 업데이트할 필요없이 업데이트된다. 이 기술은 2004년 8월 13일 Gorobets 등에 의해 출원된 미국특허출원 10/917,725에 논리 어드레스 유형의 파일 시스템에 대해 상세히 기술되어 있다.

[0186] 구체적인 구현에서, 디렉토리 블록은 메모리 셀 블록 내 총 페이지들 중 지정된 부분(예를 들면, 50%)인 고정된 수의 논리 페이지들을 내포한다. 디렉토리 블록은 바람직하게는 디렉토리 페이지들의 저장에 전용되는 메타블록이지만 단일 소거 블록일 수도 있다. 이것은 데이터 그룹들의 저장을 위해 사용되는 메타블록들과 동일한 병행도를 가질 수도 있고, 혹은 감소된 병행도를 가질 수도 있다. 파일 디렉토리 블록이 충만되었을 때, 원 블록을 소거하기 전에, 각 논리 페이지의 유효 버전을 새로운 블록에 카피함으로써 콤팩트된다. 디렉토리 블록이 컴팩트된 직후에, 새로운 카피 블록 내 페이지들의 정의된 부분만이(이를테면 50%) 디렉토리 엔트리들에 기입된다. 나머지 페이지들은 업데이트된 혹은 새로운 엔트리들이 기입될 수 있게 소거된 상태에 있다.

[0187] 디렉토리 혹은 파일에 대한 엔트리가 생성되거나, 수정되거나, 혹은 삭제될 때, 이 디렉토리 혹은 파일을 내포하는 디렉토리에 대한 한 세트의 엔트리들은 디렉토리 블록 내 다음 가용한 소거된 페이지에 재기입되고, 이에 따라 디렉토리에 대한 엔트리들을 인접하게 유지한다. 이것은 그 디렉토리에 대한 엔트리들을 전에 내포하는 페이지에 대한 독출/수정/기입 동작에 의해 행해진다. 이어서 이전 페이지는 폐용이 된다. 이 논리 페이지에 대한 페이지 포인터 엔트리는 이의 새로운 물리 페이지를 확인하기 위해 업데이트된다. 이것은 디렉토리에 대한 엔트리들을 기입하는데 사용되는 것과 동일한 페이지 프로그래밍 동작에서 행해질 수 있다.

[0188] 엔트리의 생성으로 디렉토리에 대한 한 세트의 엔트리들이 페이지를 오버플로하게 된다면, 대신에 또 다른 페이지 내 제1 엔트리로서 기입될 수도 있다. 기존 페이지는 디렉토리에 대한 한 세트의 엔트리들을, 필요하다면, 페이지의 끝으로 이동시키기 위해 재기입될 수도 있다. 디렉토리 블록 내 논리 페이지 번호들은 디렉토리에 대한 엔트리들을 인접하게 유지하기 위해서 페이지 포인터들에서 재할당될 수 있다.

[0189] 도 20의 파일 인덱스 테이블(FIT)(203)은 디바이스 내 모든 파일들을 구성하는 데이터 그룹들에 대한 인덱스들을 저장한다. FIT는 메모리 시스템의 하나 이상의 전용 FIT 블록들에 저장된다. 각 FIT 블록은 최대 수까지의 파일들에 대한 인덱스들을 저장할 수 있다. FIT는 테이블에 의해 인덱스된 파일들 중 하나를 명시하는, 파일 디렉토리로부터 논리 포인터에 의해 액세스된다. 포인터는 가장 최근에 기입된 FIT 페이지(217)의 일부로서 제공된 파일 포인터들에 액세스함으로써 간접적 어드레싱을 사용하고, 따라서 인덱스들이 업데이트되고 FIT 블록 내 재기입될 때 변하지 않는다.

[0190] 엔트리(205)와 같은, 파일 디렉토리(201) 내 파일 엔트리들의 FIT 포인터는 2개의 주요 필드들을 갖는다. 첫 번째는 FIT 블록 번호로서, FIT를 구성하는 논리 FIT 블록들 중 하나는 확인한다. FIT 블록 번호에 할당된 실제 물리적 블록 어드레스는 별도로 관리된다. FIT 포인터의 제2 필드는 FIT 파일 번호로서, 확인된 FIT 블록 내 논리 파일 번호를 확인한다. 논리 파일 번호는 FIT 블록의 가장 최근에 기입된 페이지 내 저장된 특정의 파일 포인터에 의해 물리 파일 엔트리 위치로 변환된다.

[0191] FIT는 기정의된 크기를 갖지 않으며, FIT 블록들의 수는 파일들의 수 및 데이터가 구성되는 데이터 그룹들의 수의 함수이다. 새로운 FIT 블록들이 생성될 것이며 FIT 블록들은 디바이스의 동작동안 소거될 것이다.

[0192] 도 22를 참조하면, 개개의 FIT 페이지들은 도 19에 관하여 전에 기술된 바와 같이, 각 데이터 그룹마다 하나의 엔트리(219)로, 다수의 데이터 그룹 엔트리들을 구비한다. 이 특정의 예에서, 각 엔트리(219)는 4개의 필드들을 내포한다. 파일 오프셋 필드는 엔트리에 의해 확인되는 데이터 그룹의 시작의 파일 내 오프셋 어드레스를 명시

한다. 블록 어드레스 필드는 데이터 그룹을 내포하는 메모리 시스템 내 블록 혹은 메타블록의 물리 어드레스를 명시한다. 바이트 어드레스 필드는 블록 혹은 메타블록 내 페이지의 어드레스, 및 데이터 그룹이 시작하는 페이지 내 바이트를 명시한다. 제4 필드는 데이터 그룹의 길이이다. 데이터 그룹 길이는 데이터 그룹의 길이가 인접 엔트리들의 데이터로부터 계산될 수 있기 때문에 모든 경우들에 있어서 필요하지 않을 수 있다. 그러나, 길이를 유지함으로써, 데이터 그룹의 길이가 요구될 때마다 이러한 계산을 할 필요는 없다. 이것은 데이터 그룹 엔트리들이 이들의 논리 어드레스들의 순서로 FIT에 기입되지 않는다면 -이때 계산은 더욱 어려워진다- 특히 가치있는 정보이다.

[0193] 호스트 파일은 바람직하게는, 파일을 형성하는, 순서로 된 데이터 그룹들을 함께 정의하는 한 세트의 인접한 데이터 그룹 엔트리들에 의해 정의된다. 도 22의 페이지(215)에 나타낸 흐리게 표시한 인접한 그룹 엔트리들은 하나의 파일을 정의하고, 페이지(217) 내 것들은 또 다른 파일을 정의한다. FIT 내 개개의 파일들 각각에 대해, 그 블록 내 유지된 개개의 파일들에 대해 마지막 기입된 FIT 페이지(217) 내 한 파일 포인터(221)가 있다. 파일 포인터(221)는 FIT 블록 내 특정의 FIT 파일 번호로 파일에 대한 파일 엔트리의 시작을 정의한다. 이것은 2개의 필드들을 내포한다. 한 필드는 데이터 그룹 인덱스 엔트리들이 상주하는 페이지의 물리 번호이고, 다른 필드는 그 페이지 내 제1 그룹 엔트리의 번호이다. 파일 포인터는 FIT 블록 내 각각의 가능한 파일 번호마다 존재한다. 즉, 파일 포인터들의 수는 FIT 블록 내 파일들의 최대 수와 같다. 파일 포인터 엔트리 내 유보된 코드(도시생략)는 특정 파일 번호가 FIT 블록에서 미사용된 것을 나타낸다. 파일 포인터들이 다른 FIT 페이지들에 존재할지라도, 이들은 FIT 블록의 가장 최근에 기입된 페이지에서만 유효하다.

[0194] FIT 블록은 바람직하게는 FIT 페이지들의 저장에 전용되는 메타블록이다. 데이터 그룹들에 대한 저장을 위해 사용되는 메타블록들과 동일한 병행도를 가질 수도 있고, 혹은 감소된 병행도를 가질 수도 있다. 단일 소거 블록이 대신에 사용될 수 있다. FIT 블록이 컴팩트된 직후에, 블록 내 페이지들의 정의된 부분만이(이를테면 50%) 엔트리들을 내포할 것이다. 나머지 페이지들은 업데이트된 혹은 새로운 엔트리들이 기입되게 하기 위해 소거된 상태에 있어야 한다. 1이상 FIT 블록은 많은 수의 파일들이 유지되고 있을 때 필요할 수 있다.

[0195] FIT는 FIT 블록의 프로그램되지 않은 다음 가용 페이지에 하나 이상의 완전한 파일들의 데이터 그룹 엔트리들을 기입함으로써 업데이트된다. 또한, 파일들에 대한 파일 포인터 엔트리들은 파일 엔트리들의 새로운 위치들을 확인하기 위해 업데이트된다. 파일 데이터 그룹 엔트리들 및 파일 포인터들은 동일 페이지 프로그램 동작에 기입된다. 이어서 파일의 이전 위치는 FIT 블록 내에서 폐용이 된다. 대안적으로, 파일 엔트리는 이를 다른 FIT 블록 혹은 새로운 FIT 블록에 기입함으로써 업데이트될 수 있다. 이 경우, 두 블록들 내 파일 포인터는 업데이트되어야 하고, 파일 디렉토리 내 파일에 대한 논리 포인터는 수정되어야 한다.

[0196] FIT 블록이 충만되었을 때, 유효한 그룹 엔트리들은 컴팩트된 형태로 새로운 소거된 블록에 카피되고 이전 FIT 블록은 소거된다. 논리 FIT 블록 번호 및 논리 파일 번호들은 이 동작에 의해 변경되지 않는다. 데이터 그룹 엔트리들의 수는 페이지들의 정의된 부분만이(이를테면 50%) 컴팩트된 FIT 블록에 프로그램되도록 제약될 것이다. 파일 엔트리들은 필요하다면 다른 FIT 블록들에 이전될 것이며, 파일 디렉토리 내 이들의 논리 포인터들이 수정될 것이다.

[0197] 개개의 파일의 모든 FIT 엔트리들을 한 페이지 혹은 메타페이지에 유지하는 것이 바람직하다. 이것은 파일에 대한 모든 엔트리들을 읽는 것을 비교적 용이하게 한다. 파일의 FIT 엔트리들이 각각에 다음 물리 어드레스를 각각에 포함함으로써 함께 연쇄될 수도 있을지라도, 이것은 1 이상 FIT 페이지 혹은 메타페이지를 독출할 것을 요구하게 될 것이다. 또한, FIT 엔트리들을 페이지 내에 인접되게 유지하는 것이 바람직하다. 이것은 FIT 엔트리들의 수가 증가할 때, 특히 닫혀진 파일이 열리고 데이터 그룹들이 그에 추가되었을 때, 새로운 페이지들을 빈번하게 기입하는 결과를 가져올 수 있다. 새로운 FIT 엔트리들이 새로운 페이지에 기입될 때, 현존의 엔트리들은 또 다른 페이지로부터 카피되고 새로운 페이지에 새로운 엔트리들과 함께 기입된다. 현존의 엔트리들의 이전 페이지 내 폐용 데이터에 의해 취해진 공간은 FIT 블록이 콤팩트되었을 때 사용하기 위해 검색된다.

[0198] 위에 논한 바와 같이, 개개의 데이터 그룹들은 단일 블록 혹은 메타블록 내에 내포되고, 블록 내 임의의 바이트 경계에서 시작할 수 있고, 임의의 정수개의 바이트들의 길이를 가질 수 있다. 헤더는 메모리에 기입될 때 제어기에 의해 각 데이터 그룹에 부가될 수 있다. 이러한 헤더는 2개의 필드들을 포함할 수 있다. 제1 필드는 데이터 그룹의 시작을 확인하기 위한 코드를 내포한다. 이 코드는 모든 데이터 그룹들에 대해 동일할 수 있고 그룹들의 일부로서 저장된 데이터에 드물게 존재하는 비트 패턴으로서 선택되나, 코드가 결코 이러한 데이터에 나타나지 않는 것이 반드시 필요한 것은 아니다. 데이터 그룹의 시작은 이 코드에 대해 저장된 데이터를 스캐닝하고 코드의 비트 패턴을 발견하였을 때 이것이 실제로 데이터 그룹의 시작이고 그룹 내의 데이터가 아님을 확인하는

메모리 제어기에 의해 발견될 수 있다. 헤더의 제2 필드는 이러한 확인이 행해질 수 있게 한다. 이 제2 필드는 데이터 그룹을 내포하는 파일에 대한 FIT(203) 내 파일 엔트리에의 포인터(도 20 및 도 22)이다. 제2 필드는 FIT 블록 번호 및 FIT 파일 엔트리를 정의한다. 이어서 제어기는 코드가 독출되었던 데이터 그룹을 다시 가리키는지를 알기 위해서 FIT 파일 엔트리를 읽는다. 그러하다면, 코드는 이것이 데이터 그룹의 헤드 내에 있다는 표시인 것으로 확인된다. 그렇지 않다면, 코드의 비트 패턴은 다른 데이터로부터 독출된 것임을 알게 되고 따라서 데이터 그룹 헤더로서 무시된다.

[0199] 개개의 데이터 그룹들 내 저장된 데이터의 부분으로서 이러한 헤더를 포함함으로써, 메모리 내 임의의 데이터 그룹이 속하는 파일은 이의 헤더를 읽고 헤더가 가리키는 FIT 엔트리에 액세스함으로써 결정될 수 있다. FIT 엔트리(219)(도 22)는 fileID를 포함할 수도 있을 것이다. 이 용량은 여기에서는 예를 들면 공통 블록의 가비지 수거동안 공통 블록 내 데이터 그룹들을 확인하는데 사용된다. 공통 블록 내 하나 이상의 다른 데이터 그룹들이 폐용이 되기 때문에 가비지 수거가 시작된다면, 각각의 나머지 유효 데이터 그룹이 속하는 파일을 확인할 수 있는 것이 바람직하다.

[0200] 헤더를 각 데이터 그룹의 일부로 하는 대신에, 파일마다 한 헤더가 제공될 수 있다. 파일 내 데이터 그룹에 대한 FIT 엔트리의 위치가 알려졌을 때 파일에 대한 완전한 경로명을 확인할 수 있는 것이 이점이 있고, 파일 헤더는 이것을 가능하게 한다. 이러한 헤더는 파일에 대한 FIT 내 한 세트의 엔트리들 중 첫 번째로서, 혹은 파일 디렉토리 내 디렉토리에 대한 한 세트의 엔트리들 중 첫 번째로서 제공될 수 있다.

[0201] 이 파일 헤더는 이를 FIT 파일 헤더로서 확인하기 위한 코드, 및 이를 가리키고 있는 디렉토리 엔트리에의 역 포인터를 내포한다. FIT 파일 헤더는 파일의 길이 및 데이터 그룹들의 수와 같은, 추가의 정보를 내포할 수도 있다. 대안적으로, 헤더 엔트리는 하나 이상의 정규 엔트리들의 공간을 점유할 수도 있다.

[0202] 유사하게, 디렉토리 헤더 엔트리는 이를 디렉토리 헤더로서 확인할 코드, 및 이를 가리키고 있는 디렉토리 엔트리에의 역 포인터를 내포한다. 루트 디렉토리는 이의 헤더에서 명확하게 확인된다. 디렉토리 헤더는 디렉토리 내 엔트리들의 수와 같은, 추가의 정보를 내포할 수도 있다.

[0203] 이러한 데이터 그룹 혹은 파일 헤더들의 사용에 대한 대안으로서, 공통 블록 내 각 데이터 그룹이 관계된 파일을 확인하기 위해 각 공통 블록의 각각의 끝에 인덱스가 기입될 수도 있다. 공통 블록이 닫혔을 때, 이러한 인덱스가 기입될 수 있다. 인덱스는 바람직하게는 공통 블록 내 각 데이터 그룹이 시작하는 이 공통 블록 내 물리 바이트 어드레스, 및 이 데이터 그룹에 대한 FIT 엔트리에의 포인터를 내포한다. 그러면 각 공통블록 데이터 그룹이 일부인 파일은 데이터 그룹의 인덱스 엔트리에 제공된 FIT 엔트리에의 참조에 의해 결정될 수 있다.

[0204] 위에 기술된 데이터 그룹 헤더는, 이들 공통 블록 인덱스들이 채용될지라도, 메모리에 저장된 데이터 그룹들에 대한 파일 신원의 어떤 용장성을 갖추는 것이 요망된다면, 여전히 보존될 수 있다.

[0205] 메모리 디바이스가 파일 디렉토리 및 파일 인덱싱을 관리하기 때문에, 호스트에 보고되는 디바이스 구성 파라미터들을 제어할 수 있다. 디바이스에 대한 파일 시스템이 현 상용 시스템들에서처럼 호스트에 의해 관리될 때는 통상적으로 가능하지 않다. 메모리 디바이스는 예를 들면 전체 디바이스의 용량 및 기입되지 않은 가용 용량 둘 다인 보고된 저장용량을 변경할 수도 있다.

[0206] 직접-파일 인터페이스에서 파일들에 대해 사용되는 경로명은 저장 디바이스 자체의 ID, 및 저장 디바이스 내 분할을 확인일 수 있다. 메모리 디바이스가 분할들의 크기 및 개수를 수정할 수 있게 하는 것이 이점일 수 있다. 분할이 충만되면, 하나 이상의 다른 분할들로부터의 미사용된 용량은 전체 분할에 재할당될 수 있다. 유사하게, 다른 분할들로부터의 미사용된 용량은 새로운 분할이 생성된다면 할당될 수 있다. 디바이스는 위에서처럼, 여러 분할들에 대해 보고된 용량을 수정할 수 있다.

[0207] 도 20 및 도 21에 도시된 디렉토리 구조는 이 디렉토리 내 요소들에 대한 한 세트의 다른 인접한 엔트리들의 시작을 확인하기 위해 디렉토리 엔트리 내 포인터를 사용한다. 이 포인터는 디렉토리 블록 내 논리 페이지를 가리키며, 이 논리 페이지에 대응하는 물리 페이지가 변경될 때 변경되지 않은 채로 있다. 그러나, 디렉토리 내 요소들의 수는 자주 변경되며, 특정 디렉토리에 대한 한 세트의 엔트리들은 한 논리 페이지에서 다른 논리 페이지로 이동되거나 이의 현 논리 페이지 내에서 이동될 필요가 자주 있을 것이다. 이것은 디렉토리 블록들 내 포인터 참조들을 업데이터할 빈번한 필요성을 초래할 수 있다. 도 21에 대해서 디렉토리 구조에 대한 대안적인 엔트리 인덱싱 기술이 도 23에 도시되었고 여기서 이의 대응하는 요소들은 프라임(')이 추가된 동일 참조부호로 확인될 수 있다.

[0208] 도 23의 구조는 도 22의 FIT와 동일하나 디렉토리들에 관계된 술어는 다르다. 특정 엔트리들은 도 21에 행해진

페이지만을 가리킨다. 이것은 도 21에 관하여 위에 기술된 것보다는 명시된 디렉토리에 대한 한 세트의 엔트리들에 대한 디렉토리 블록 내 간접 어드레싱의 보다 효율적 방법을 제공한다. 디렉토리(도 23) 및 FIT(도 22)에 대한 같은 인덱싱 방식 둘 다는 매우 유사한 특징 및 요건을 갖기 때문에, 이들을 사용하는 것이 적합하다. 이들은 각각이 디렉토리들이나 파일들에 관계된 복수 세트들의 인접한 엔트리들을 저장한다. 한 세트의 엔트리들의 업데이트는 현존 엔트리들의 내용을 변경하거나 엔트리들의 수를 변경하는 것으로 구성될 수 있다.

[0209] FIT 엔트리들을 업데이트하기 위해 도 22에 관하여 기술된 방법은 블록의 콤팩트후에 FIT 엔트리들을 내포하는 물리적 블록에 페이지들의 부분(예를 들면 50%)을 소거된 상태로 남겨놓는다. 그러면 이것은 업데이트된 FIT가 컴팩트된 블록의 나머지에 기입되게 한다. 모든 FIT 블록들은 블록에 의해 인덱스된 어떠한 파일들도 실제로 열리지 않았을 때에도, 이 용량 오버헤드를 포함시킨다. 그러므로 FIT는 요망되는 것보다 많은 메모리 용량을 점유한다.

[0210] 대안으로서의 FIT 업데이트 방법은 업데이트된 복수 세트들의 FIT 엔트리들이 원래 위치하였던 블록들 내 소거된 가용 용량에가 아니라, 별도의 FIT 업데이트 블록에 기입되게 한다. 도 24 및 도 25는 업데이트 블록들을 이용하는 파일 디렉토리 및 FIT에 대한 인덱싱 기술들을 각각 개괄한다. 기술들은 디렉토리 및 FIT에 대한 술어만이 상이하고, 동일하다.

[0211] 이하 설명은 도 25에 도시한 FIT 블록 내 그룹 엔트리들을 업데이트하는 것에 관한 것이지만, 파일 디렉토리 내 그룹 엔트리들을 업데이트하는 것(도 24)에 똑같이 적용할 수 있다.

[0212] FIT 블록으로부터 그룹 엔트리들을 읽는 것은 위에 기술된 바와 같다. FIT 포인터의 FIT 블록 번호 필드는 논리 FIT 블록을 정의한다. FIT 블록 리스트는 플래시에 데이터 구조에 내포되고, FIT 블록 번호를 이것이 놓인 블록의 물리 어드레스로 변환하는 정보를 제공한다. FIT 블록 리스트 내 블록 어드레스는 FIT 블록이 컴팩트 또는 통합 동작동안 이동될 때마다 업데이트된다.

[0213] 이 블록에 가장 최근에 기입된 페이지의 파일 포인터들 영역은 FIT 포인터 내 FIT 파일 번호 값을 명시된 파일에 대한 한 세트의 파일 그룹 엔트리들의 시작을 지정하는 파일 포인터로 변환될 수 있게 한다. 그러면 파일 그룹 엔트리들은 FIT 블록으로부터 독출될 수 있다.

[0214] 그룹 엔트리의 내용, 혹은 파일에 대한 세트 내 그룹 엔트리들의 수가 업데이트되었을 때, 완전한 한 세트의 엔트리들이 소거된 페이지에 재기입된다(이것은 동일 페이지에 복수의 기입 동작들이 소거 동작들 사이에선 금지되면서, 페이지가 플래시 메모리에서 최소 프로그래밍 단위라는 가정에 기초한다). 세트는 필요하다면 복수의 페이지들을 점유할 수도 있다.

[0215] 현재 기술되는 기술에서, 이 페이지는 FIT 블록 내 다음 가용 페이지이다. 수정된 방식에서, 이 페이지는 별도의 FIT 업데이트 블록 내 다음 가용 페이지이다. FIT 업데이트 블록은 도 23에 도시된 바와 같이, FIT 블록과 동일한 페이지 구조를 갖는다. 이의 존재는, 업데이트 블록에 대한 물리 블록 어드레스 및 원 FIT 파일 번호에 대응하는 업데이트 파일 번호도 내포하는, FIT 업데이트 블록 리스트 내 타겟 FIT 블록 번호의 존재에 의해 확인된다. 업데이트되는 각 FIT 블록에 대한 별도의 FIT 업데이트 블록이 있을 수 있으며, 혹은 바람직하게는 FIT 업데이트 블록이 복수의 FIT 블록들에 관계될 수도 있다. 또한, 단일 FIT 블록이 복수의 FIT 업데이트 블록들에 관계될 수도 있다.

[0216] FIT 업데이트 블록이 채워졌을 때, 이의 유효한 데이터는 컴팩트된 형태로 소거된 블록에 기입될 수 있고, 이것은 새로운 FIT 업데이트 블록이 된다. 업데이트들이 단지 소수의 파일들에 관계되었다면, 단일 페이지의 유효 데이터만큼 적을 수도 있다. 복수의 FIT 업데이트 블록들은 단일의 블록으로 함께 컴팩트될 수도 있다. 블록 컴팩트는 업데이트 블록이 파일, 혹은 계속 업데이트될 수 있는 여전히 열려 있는 파일들에 관계된다면 블록 통합보다 바람직하다.

[0217] 파일에 대한 그룹 엔트리들이 FIT 업데이트 블록에서 업데이트될 때, 원 FIT 블록 내 이 파일에 대한 엔트리들은 폐용으로 된다. 어떤 단계에서, 원 FIT 블록은 이를 일소하기 위해 이에 가비지 수거 동작이 행해져야 한다. 이것은 FIT 블록 및 FIR 업데이트 블록내 유효 데이터를 소거된 블록에 통합함으로써 행해질 수 있다.

[0218] 엔트리들의 수가 업데이트 프로세스 동안 증가하였고, 유효 데이터가 단일 소거된 블록에 통합될 수 없다면, 원래 이 FIT 블록에 할당된 파일들은 2이상의 FIT 블록들에 재할당될 수 있고, 통합이 2이상의 블록들에 수행될 수 있다.

[0219] FIT 업데이트 블록으로부터 엔트리들은 FIT 블록으로부터의 엔트리들에 통합될 수 있고, 그러므로 FIT 업데이트

블록으로부터 제거되고, 반면 다른 파일들에 대한 엔트리들은 FIT 업데이트 블록 내 남아있다.

[0220] 또 다른 대안으로서, 디렉토리 블록 및 FIT 블록 구조들은 단일 블록 구조로 합쳐질 수 있고, 이것은 디렉토리 엔트리들 및 파일 그룹 엔트리들 둘 다를 내포할 수 있다. 인덱스 업데이트 블록은 개별적인 디렉토리 블록 구조 및 FIT 블록 구조가 있을 때, 혹은 결합된 인덱스 블록 구조가 있을 때, 디렉토리 엔트리들 및 파일 그룹 엔트리들 둘 다에 대한 업데이트 블록으로서 작용할 수 있다.

[0221] 도 20-25에 관하여 위에 기술된 파일 디렉토리 및 FIT는 DOS 시스템들에 맞추어짐을 알 것이다. 대안적으로, 이들은 리눅스, 유닛스, NT 파일 시스템(NTFS) 혹은 이외 어떤 공지된 운영 시스템에 맞추어질 수 있다.

[0222] 구체적 메모리 시스템 동작의 예

[0223] 도 26-32의 동작 흐름도들은 도 9 및 도 10-22에 관하여 기술된 직접 파일 인터페이스 기술들의 특정한 조합의 사용과 더불어 도 2-6에 관하여 위에 기술된 바와 같이 구성된 메모리 시스템의 동작의 예를 제공한다. 도 26-32의 흐름도들에 포함된 기능들은 주로 제어기(11)의 저장된 펌웨어를 실행하는 제어기(11)(도 2)에 의해 수행된다.

[0224] 먼저 도 26을 참조하면, 전체 시스템 동작이 도시되었다. 제1 단계 251에서, 메모리 시스템이 초기화되는데, 이것은 프로세서(27)(도 2)가 ROM(29) 내 부트 코드를 실행하여 비휘발성 메모리로부터 펌웨어를 RAM(31)에 로딩하는 것을 포함한다. 초기화 후에, 이 펌웨어의 제어 하에, 메모리 시스템은 단계 253에 나타낸 바와 같이, 호스트 시스템으로부터 명령을 찾는다. 호스트 명령이 미완이면, 명령은 도 12에 열거된 것들 중에서 확인된다. Read 명령(도 12a)이라면, 이것은 단계 255에서 확인되어 도 27에 관하여 후술하는 257에 나타낸 프로세스에 의해 실행된다. 독출 명령이 아니면, 도 12a의 Write, Insert 혹은 Update 프로그래밍 명령들 중 어느 하나가 도 26의 단계 259에서 확인되고 도 28의 요지인 프로세스(261)에 의해 실행된다. Delete 혹은 Erase 명령(도 12b)이라면, 이것은 도 30에 상세히 기술된 바와 같이, 단계 262에서 확인되고 단계 263에서 실행된다. 호스트로부터 Idle 명령(도 12d)은 도 26의 264에서 확인되고 도 31의 흐름도에 도시된 가비지 수거 동작(265)으로 된다. 도 26-32의 이 예는 앞서 기술된 바와 같이, 메타페이지들 및 메타블록들로 동작하는 메모리 시스템에 대해 기술되었지만 페이지들 및/또는 블록들로 구성된 메모리 시스템에 의해 실행될 수도 있다.

[0225] 호스트 명령이 Read, Write, Insert, Update, Delete, Erase 혹은 Idle 이외의 것이라면, 이 특정의 예에서, 이러한 또 다른 명령은 도 26의 단계 267에 의해 실행된다. 이들 다른 명령들 중에는 도 12b에 열거되고 위에 기술된 Close 및 Close_after 명령들과 같이, 가비지 수거 동작이 대기열에 추가되게 하는 것들이 있다. 단계들 257, 261, 263, 265 혹은 267 중 어느 하나에 의한 수신된 명령을 실행한 후에, 다음 단계 268은 우선 가비지 수거 대기열들이 비어있는지 여부를 문의한다. 그러하다면, 처리는 단계 253으로 되돌아가서, 미완 호스트 명령이 존재한다면 이를 실행한다. 그렇지 않다면, 처리는 단계 265로 되돌아가서 또 다른 호스트 명령의 가능성이 실행되게 하기보다는 가비지 수거를 계속한다. 도 26-32의 흐름도들에 관하여 기술된 구체적 예에 있어서, 위에 기술된 5개의 서로 다른 가비지 수거 대기열들로서: 폐용 메타블록들(폐용 데이터만을 가진 메타블록들) 및 폐용 데이터를 가진 공통의 메타블록들 -폐용 데이터는 Erase 명령에 기인한다- 에 대한 2개의 우선 대기열들; 폐용 데이터 메타블록들 및 폐용 데이터를 가진 공통의 메타블록들 -폐용 데이터는 Erase 명령들의 실행보다는 다른 것으로부터 비롯된다- 에 대한 2개의 다른 대기열들; 및 가비지 수거될 파일들에 대한 하나의 대기열이 있다. 3개의 비-우선 대기열들에 열거된 가비지 수거에 있어서, 또 다른 미완 호스트 명령에는 열거된 가비지 수거 동작들을 수행하는 것에 대한 우선이 부여된다. 그러나, 우선 대기열들 내 것들에 대해서, 가비지 수거에는 새로운 호스트 명령의 실행에 대한 우선이 부여된다. 즉, 새로운 호스트 명령이 실행될 수 있기 전에 어떤 우선 가비지 수거 동작들이든 완료될 때까지 호스트를 기다리게 할 수 있다. 이것은 Erase 명령의 사용에 의해 우선 대기열들에 메타블록들 내 데이터의 소거에 대해 호스트에 우선이 사전에 부여되었기 때문이다. 또한, 우선 가비지 수거에 의해 추가의 소거된 메타블록들이 비교적 짧은 처리시간에 생성된다. 그러나, 우선이 없다면, 가비지 수거는 호스트가 아이들 상태에 있을 때 백그라운드에서 수행되거나, 소거된 블록 풀을 유지할 필요가 있을 때 다른 동작들에 개재된다. 단계 268에 의해서, 미완의 우선 가비지 수거 동작들이 없는 것으로 판정되면, 다음 단계는 단계 253이고 여기서 새로운 호스트 명령이 있다면 이 명령이 실행된다.

[0226] 도 26의 단계 253으로 가서, 어떠한 호스트 명령도 미완이 아니라면, 단계 269에 의해서 호스트가 소정 길이의 시간동안 비활성 혹은 아이들 상태였거나, 가장 최근에 수신된 호스트 명령이 Idle 명령인 것으로 판정된 경우에, 혹은 단계 264에 의해서 미완의 호스트 명령이 Idle 명령인 것으로 판정된 경우에, 비-우선 가비지 수거를

포함한, 가비지 수거(265)가 수행된다. 이들 상황들 하에서, 가비지 수거는 백그라운드에서 거의 전적으로 수행될 것이다.

[0227] 도 26의 단계 257에서 Read 명령의 실행이 도 27의 흐름도의 요체이다. 제1 단계 271은 도 19 및 도 22에 관하여 위에 기술된 바와 같이, 파일 인덱스 테이블(FIT)을 독출할 것이다. 호스트의 Read 명령은 fileID 및 독출이 시작하는 파일 내 오프셋(도 12a 참조)을 포함한다. 독출할 파일에 대한 모든 FIT 엔트리들, 혹은 이의 일부는 바람직하게는 어떤 데이터가 필요할 때마다 비휘발성 메모리로부터 FIT를 독출할 필요성을 피하기 위해서 비휘발성 메모리로부터 제어기 메모리(31)(도 2)로 독출된다. 단계 273에서, 데이터 그룹 수 카운터는 시작 오프셋이 놓인 요청된 파일을 구성하는 데이터 그룹의 수로 초기화된다. 이것은 호스트에서 명시된 시작 오프셋을 호스트에서 지정된 파일에 대한 FIT 엔트리들 내 데이터 그룹의 오프셋과 비교함으로써 행해진다. 다음에, 단계 275에서, 데이터 그룹 길이 카운터는 호스트에서 공급된 오프셋부터 데이터 그룹의 끝까지의 처음 데이터 그룹 내 데이터량으로 초기화된다. 한 데이터 그룹이 한번에 독출되고, 단계 273 및 단계 275는 제1 데이터 그룹의 독출을 관리하는 2개의 카운터들을 셋업한다. 독출할 데이터의 비휘발성 메모리 내 시작하는 물리적 바이트 어드레스는 처음 데이터 그룹에 대한 FIT 엔트리로부터 결정된다. 그룹 내 데이터의 논리 오프셋 및 물리 바이트 어드레스는 선형으로 관계되므로 시작 바이트 어드레스는 데이터 그룹의 시작부분에서 독출이 시작하지 않는다면, 파일 내 호스트에서 공급된 시작 오프셋으로부터 계산된다. 대안적으로, 데이터 그룹 길이 카운터의 초기화를 간이화하기 위해서, 각 데이터 그룹의 길이는 FIT의 그의 기록(219)에 추가될 수 있다(도 22).

[0228] 흐름도들에 대한 이 설명의 나머지에 있어서는 메모리 시스템이 메타블록들로 동작하고 있고, 앞서 기술된 바와 같이, 데이터가 메타페이지 단위들로 독출 및 프로그램되는 것으로 가정한다. 데이터 메타페이지는 시작 바이트 어드레스를 포함하는 비휘발성 메모리에 저장된 초기 데이터 그룹으로부터 단계 277에서 독출된다. 이어서, 독출된 데이터는 통상적으로 호스트에 전송을 위해 제어기 버퍼 메모리(예를 들면, 도 2의 RAM(31))에 기입된다. 이어서, 단계 279에서 데이터 그룹 길이 카운터는 한 메타페이지만큼 감분된다. 이어서, 이 카운터는 단계 281에서 독출되어 이것이 제로에 도달하였는지 판정한다. 아니라면, 독출할 초기 데이터 그룹의 더 많은 데이터가 있다. 그러나, 데이터의 다음 메타페이지를 순서대로 독출하기 위해 단계 277로 돌아가기 전에, 단계 283는 호스트가 또 다른 명령을 발행하였는지를 체크한다. 그러하다면, 독출 동작은 종료되고 프로세스는 도 26의 단계 253으로 돌아가서 수신된 명령을 확인하고 이를 실행한다. 그렇지 않다면, 초기 데이터 그룹의 독출은 도 27의 단계 277 및 단계 279에서 그의 다음 메타페이지를 독출함으로써 계속된다. 이것은 데이터 그룹 길이 카운터가 단계 281에 의해서 제로에 도달한 것으로 판정될 때까지 계속된다.

[0229] 이것이 일어났을 때, 파일에 대한 FIT 엔트리들이 단계 285에서 다시 읽혀져, 단계 287에서 독출한 현 파일의 더 이상의 데이터 그룹들이 있는지 판정한다. 그러하다면, 데이터 그룹 수 카운터는 다음 데이터 그룹을 확인하기 위해 단계 289에서 업데이트되고, 데이터 그룹 길이 카운터는 새로운 그룹 내 데이터의 길이로 단계 291에서 초기화된다. 이어서 단계 283에 의해 어떠한 다른 호스트 명령도 미완이 아닌 것으로 판정되면, 새로운 데이터 그룹의 제1 메타페이지가 단계 277에서 독출된다. 이어서 단계 277 및 단계 279는 새로운 데이터 그룹의 각 메타페이지에 대해 이의 모든 데이터가 독출될 때까지, 이 때 단계 285 및 단계 287은 아직 또 다른 데이터 그룹이 존재하는지 판정하고, 등등을 행하여 반복된다. 단계 287에 의해서 호스트에서 공급된 오프셋 후에 파일의 모든 데이터 그룹들이 독출된 것으로 단계 287에 의해 판정되었을 때, 처리는 도 26의 단계 253으로 돌아가서 또 다른 호스트 명령을 실행한다.

[0230] 파일이 원형 버퍼로서 사용되고 있는 특별한 경우에, 파일의 독출은 도 26의 단계 253으로 되돌아가기보다는 도 27의 단계 287 후에 반복될 수 있다. 이것은 단계 283에서 데이터를 동일 파일에 기입하라는 호스트 명령에 응답함으로써 독출할 동안 호스트에 의해 현 파일의 데이터가 프로그램되고 있는 경우에 일어날 수 있다.

[0231] 도 26의 데이터 프로그래밍 동작(261)의 예가 도 28의 흐름도에 주어진다. 도 12a의 데이터 프로그래밍 명령들 중 하나가 호스트로부터 수신될 때, 이 명령은 데이터가 기입될 파일의 fileID를 포함한다. 제1 단계 295는 지정된 파일이 현재 프로그래밍을 위해 열려있는지를 판정한다. 그러하다면, 다음 단계 297은 이 파일에 대한 제어기 버퍼(이를테면 도 2의 RAM(31))에 데이터가 있는지를 판정한다. 데이터는 호스트 시스템에 의해 제어기 버퍼 메모리에 전송되고, 이어서 메모리 시스템 제어기에 의해 플래시 메모리에 전송된다.

[0232] 그러나 호스트에서 지정된 파일이 단계 295에 의해 열리지 않은 것으로 판정되면, 다음 단계 299는 프로그래밍을 위해 메모리 시스템에 의해 현재 열린 파일들의 수가 메모리 시스템에 의해 동시에 열려지게 허용되는 최대 수 N1 이상인지를 묻는다. 수 N1은 메모리 시스템 내에 기설정되며, 파일들의 5, 8, 혹은 어떤 다른 수일 수 있다. 열린 파일들의 수가 N1 미만이면, 다음 단계 301은 새로운 파일에 데이터를 프로그램하는데 필요한 시스템

자원들을 제공함으로써 이 새로운 파일이 열려지게 하고, 처리는 단계 297로 진행한다. 그러나, 열린 파일들의 수가 단계 299에서 N1 이상인 것으로 판정되면, 새로운 파일이 단계 301에서 열려질 수 있기 전에, 단계 303에 나타낸 바와 같이, 현재 열려진 파일이 먼저 닫혀질 필요가 있다. 열린 파일이 단계 303에서 닫혀지게 선택되는 근거는 다양할 수 있으나 가장 일반적으로는 호스트에 의해 데이터가 최소로 최근에 기입된 열린 파일일 것이다. 이로부터, 호스트는 가까운 미래에 이 파일에 데이터를 기입하지 않을 것이라고 가정한다. 그러나 그렇게 된다면, 이 파일은 그 때에 필요한 경우 또 다른 열린 파일이 닫혀진 후에 다시 열린다.

[0233] 단계 297에서 현재 파일의 적어도 메타페이지의 데이터가 제어기 버퍼에 있는 것으로 판정될 때, 다음 단계 305는 메모리 어레이 내 메타블록이 프로그래밍을 위해 열렸는지를 판정한다. 그러하다면, 데이터는 단계 307에서, 제어기 버퍼 메모리로부터 열린 메타블록으로 프로그램된다. 그렇지 않다면, 단계 308에서 메타블록에 데이터를 프로그램하는데 필요한 시스템 자원들을 제공함으로써 이 메타블록이 먼저 열린다. 데이터는, 이 예에서는, 한 번에 한 메타페이지 단위들로, 열린 메모리 메타블록에 기입된다. 일단 이 데이터 단위가 기입되면, 다음 단계 309는 열린 기입 메타블록이 데이터로 충만되었는지를 판정한다. 그렇지 않다면, 프로세스는 정규로 단계 311 및 단계 313을 진행하고 단계 297로 돌아가서 현재 열린 파일에 다음 메타페이지의 데이터를 프로그래밍하기 위한 프로세스를 반복한다.

[0234] 그러나, 기입 메타블록이 단계 309에 의해 충만된 것으로 판정되면, 이 메타블록은 단계 315에서 닫혀지고, 메모리 메타블록 경계에 도달하였기 때문에 현 데이터 그룹을 닫는 것을 포함하여, FIT는 단계 317에서 업데이트된다. 이어서, 낮은 우선 폐용 메타블록 가비지 수거 대기열의 메타블록 엔트리들이 단계 318에서 업데이트된다. 단계 317에서 FIT 업데이트 동안에, 기입 메타블록을 채우는 것이 현재 파일의 모든 폐용 데이터를 내포하는 또 다른 메타블록을 생성하였는지, 아니면 현재 파일의 폐용 데이터를 내포하는 공통의 메타블록인지가 판정된다. 그러하다면, 이 메타블록은 가비지 수거를 위해 단계 318에 의해 적합한 낮은 우선 메타블록 대기열에 추가된다. 이어서 처리는 단계 311로 되돌아가서 단계 313을 통해 다시 단계 297로 간다. 단계 305 및 단계 307을 거치는 시간에, 단계 308은 이전 메타블록이 단계 315에 의해 방금 닫혀졌기 때문에 새로운 기입 메타블록을 열 것이다.

[0235] 단계 311은 데이터의 각 메타페이지가 단계 307을 포함한 경로에 의해 기입된 후에, 소거된 메타블록 풀에 현재 존재하는 메타블록들의 수가 메모리 시스템을 효율적으로 동작시키는데 필요한 것으로 판정된 최소 수 N2를 초과하였는지를 묻는다. 그러하다면, 단계 313은 또 다른 호스트 명령이 수신되었는지를 묻는다. 어떤 다른 호스트 명령도 미완이 아니라면, 단계 297은 메모리에 프로그램될 다음 메타페이지의 데이터에 대해 반복된다. 그러나, 호스트 명령이 수신되었다면, FIT는 단계 319에서 업데이트되어 기입된 데이터 그룹을 닫는다. 단계 320에서 낮은 우선 폐용 메타블록 가비지 수거 대기열 내 메타블록 엔트리들을 업데이트한 후에(위에 기술된 단계 318과 유사함), 프로세스는 도 26의 단계 253으로 되돌아간다.

[0236] 그러나, 도 28의 단계 311에서 데이터의 저장을 위한 소거된 메타블록들의 부족(기설정된 수 N2 이하)이 있는 것으로 판정되면, 소거된 메타블록들의 수를 증가시키기 위해서, 포그라운드에서 파일 혹은 공통 메타블록을 가비지 수거하기 위한 서브루틴이 실행된다. 이러한 가비지 수거는 바람직하게는 데이터의 각 메타페이지가 메모리에 기입된 후에 수행되지 않고 그보다는 각 N3 메타페이지들이 기입된 후에만 수행된다. 기입 메타블록 메타페이지 프로그램은 프로그래밍간에 어떠한 가비지 수거도 없이 연속하여 프로그램된 다수의 메타페이지들의 호스트 데이터의 카운트를 유지한다. 이 카운터는 가비지 수거가 수행될 때마다 제로로 리셋되고 이어서 한 메타페이지의 데이터가 프로그램될 때마다 1만큼 증분된다. 단계 321에서, 이 카운트가 소정의 수 N3를 초과하는지가 판정된다. 그렇지 않다면, 카운터는 단계 307에서 메모리에 메타페이지가 기입되었다는 기록에, 단계 323에서, 1만큼 증분된다. 그러나, 프로그램 카운터의 카운트가 수 N3를 초과한다면, 가비지 수거는 단계들 325, 327, 329에 의해 행해지고, 이어서 프로그램 카운터는 단계 331에서 제로로 리셋된다.

[0237] 프로세스에서 이 지점에서 가비지 수거의 목적은 소거된 메타블록 풀을 위한 추가의 소거된 메타블록들을 생성하는 것이다. 그러나, 가비지 수거 동작의 부분만이 단계들 325, 327, 329의 실행에 의해 매번 수행되는 것이 바람직하다. 카피 동작은 호스트가 메모리를 사용할 수 없는 간격들의 기간을 최소화하여 데이터 프로그래밍 수행에 악영향을 최소화하기 위해서 시간에 따라 연장되는 보다 작은 동작들로 분할된다. 그리고 부분적 가비지 수거 동작은 같은 이유로 매 N3 프로그래밍 사이클들에만 수행된다. 수 N2 및 N3가 시스템에서 기설정될 수 있을지라도, 이에 대안적으로 이들은 부닥칠 수 있는 특정한 상태들에 적응하게 메모리 시스템의 동작동안 결정될 수도 있다.

[0238] 하나 이상의 추가의 소거된 메타블록들을 제공하기 위해 파일 혹은 공통 메타블록에 대해 완전한 가비지 수거

동작에 의해 요구되는 대부분의 시간은 유효 데이터를 하나 이상의 카피 메타블록들에 카피함으로써 취해지기 때문에, 도 28에서 데이터 프로그래밍과 주로 인터리브되는 것이 이 카피하는 것이다. 단계 325는 가비지 수거(329)를 위해 포그라운드(foreground) 모드를 선택하며, 이것은 도 31의 흐름도에 관하여 기술된다. 이어서 데이터 그룹의 어떤 기설정된 수의 메타페이지들은 가비지 수거되는 메타블록으로부터 이전에 소거된 카피 메타블록에 연속하여 카피된다. 단계 327은 이러한 인터리브 식으로 카피되는 메타페이지들의 카운터를 리셋하며, 따라서 단계 329에서 기설정된 개수들의 메타페이지들은 동작이 가비지 수거 단계로부터 데이터 프로그래밍 루프로 되돌아가지 전에 카피된다. 단계 321에 의해 참조되는 기입 메타블록 메타페이지 프로그램 카운터는 각각의 가비지 수거 동작(329) 후에 단계 331에 의해 리셋된다. 이것은 또 다른 가비지 수거 동작이 포그라운드에서 행해지기 전에 N3개의 메타페이지들의 데이터가 메모리에 기입되게 하며, 그럼으로써 가비지 수거에 의해 야기되는 데이터 프로그래밍에 있어서의 어떤 지연들을 연장시킨다.

[0239] 이러한 가비지-프로그래밍 알고리즘의 인터리브 특징이 도 29의 시간선(timeline)에 의해 도시되었다. 호스트로부터 N3 메타페이지들의 데이터는 가비지 수거 동작들 사이에 메모리에 연속적으로 기입된다. 각각의 가비지 수거 동작은 N4개의 메타페이지들의 데이터를 카피하는 것으로 제한되고, 그후에 또 다른 N3개의 메타페이지들의 데이터의 프로그래밍이 행해진다. 가비지 수거 단계 중에, 호스트는 추가의 메타페이지들의 데이터를 메모리 시스템 제어기 버퍼에 전송하기 전에 메모리 시스템이 가비지 수거를 완료하기를 기다리게 된다.

[0240] 도 26의 파일 삭제 루틴(263)의 실행이 도 30의 흐름도에 의해 도시되었다. 이의 주요 목적은 삭제되는 파일의 폐용 블록들 및 폐용 데이터를 가진 공통 블록들을 확인하고 이어서 이들 블록들을 적합한 가비지 수거 대기열들에 두는 것이다. Delete 혹은 Erase 명령(도 12b)이 삭제될 파일의 이름 혹은 이외 다른 신원과 함께 메모리 시스템에 의해 수신될 때, 단계 335는 데이터 그룹 수 카운터를 제로로 초기화한다. 단계 337에서, 지정된 파일을 구성하고 있는 메모리 내 저장된 데이터 그룹들의 신원들을 얻기 위해서 파일 디렉토리 및 파일 인덱스 테이블(FIT)이 독출된다. 이어서 파일의 각각의 데이터 그룹은 논리 순서로 각 데이터 그룹을 가리키는 데이터 그룹 수 카운터를 증분시킴으로써 한번에 하나씩 검사된다.

[0241] 데이터 그룹에 대한 제1 문의(339)는 이 데이터 그룹이 공통 메타블록 내에 놓여 있는지 여부이다. 그러하다면, 단계 341은 이 메타블록을 공통 블록 가비지 수거 대기열들 중 하나에 추가한다. 메타블록은 파일이 Erase 명령에 의해 삭제되는 것이면 우선 공통 메타블록 대기열에 두어지고 Delete 명령에 의해 파일일 삭제되는 것이면 다른 공통 메타블록에 두어진다. 삭제될 파일의 데이터 그룹을 가진 임의의 공통 메타블록은 가비지 수거에 스케쥴링된다. 단계 339에 의해 데이터 그룹이 공통 메타블록에 있지 않은 것으로 판정되면, 다음 문의(343)는 폐용 메타블록 가비지 수거 대기열에 있게 함으로써 가비지 수거에 이미 스케쥴링된 메타블록 내에 데이터 그룹이 있는지를 판정한다. 메타블록이 가비지 수거에 이미 스케쥴링되어 있으면, 다시 이 동일 대기열에 추가되지 않을 것이다. 그러나, 그와 같이 스케쥴링되지 않았으면, 단계 345에 의해 폐용 메타블록 가비지 수거 대기열들 중 하나에 추가된다. 메타블록은 파일이 Erase 명령에 의해 삭제되는 것이면 우선 폐용 메타블록에 두어지고 Delete 명령에 의해 삭제되는 것이면 다른 폐용 메타블록 대기열에 두어진다.

[0242] 단계 341 혹은 단계 345 중 어느 하나가 실행되었거나 단계 343의 문의가 긍정이면, 한 데이터 그룹에 대한 프로세스가 완료된다. 다음 단계 347은 데이터 그룹 수 카운터를 증분한다. 이어서 파일 내 또 다른 데이터 그룹이 있는지에 대해 문의(349)가 행해진다. 그러하다면, 처리는 단계 339로 되돌아가서 그 다음 데이터 그룹에 대해 반복된다. 그렇지 않다면, 삭제할 파일의 데이터를 내포하는 모든 메타블록들은 폐용 및 공통 메타블록 가비지 수거 대기열들에 넣어졌음을 알게 된다. 이어서 단계 351에서 삭제할 파일의 데이터 그룹 기록들을 폐용하기 위해 FIT가 업데이트된다. 이어서 폐용 데이터 그룹들의 FIT 내 기록들은 통사적으로 FIT가 콤팩트될 때 소거된다. 마지막 단계 353에서, 파일 디렉토리(201)(도 20)는 삭제된 파일을 그로부터 제거하기 위해 업데이트된다.

[0243] 도 31의 흐름도는 도 26의 가비지 수거 동작(265)과 도 28의 가비지 수거 동작(329)의 구체적인 예를 도시한 것이다. 이 알고리즘은 호스트가 Idle 명령을 보낼 때(도 26의 단계 264), 혹은 호스트가 잠시 아이들 상태에 있을 때(도 26의 단계 265) 백그라운드에서 진입되고, 혹은 N2 소거된 메타블록들보다 적은 수의 메타블록이 소거된 메타블록 풀 내에 남아있을 때 프로그래밍 동작(도 28의 단계 329) 동안에 포그라운드에서 진입된다. 제1 문의(355)는 아직 완료되지 않은 진행중의 가비지 수거 동작이 있는지 여부이다. 도 31의 이 설명으로부터 이를테면 호스트가 또 다른 명령을 발행할 때 혹은 데이터 카피가 다른 동작들에 인터리브될 때와 같은 어떤 상황들 하에서 처리가 가비지 수거로부터 벗어남을 알 것이다. 그러므로 미완의 미완료된 가비지 수거동작이 있다면, 미완 가비지 수거에 우선이 부여되기 때문에 단계 357은 다음이다. 그러나, 단계 355에 의해서 미완 가비지 수거 동작이 없는 것으로 판정되면, 다음 단계 356은 여러 가비지 수거 대기열들 내 적어도 하나가 있는지를 알기 위해서 이들 대기열들을 조사한다. 그러하다면 다음 단계 358은 위에 논의된 우선들에 따라, 1이상이 있다면,

엔트리들 중 하나를 선택한다. 폐용 메타블록 블록 대기열 내에 순서대로 다음 메타블록의 가비지 수거는 일반적으로, 공통 메타블록 대기열 상의 메타블록 혹은 가비지 수거될 파일들의 대기열 상의 파일보다 높은 우선이 부여될 것이다. 이것은 소거된 메타블록 풀의 크기가, 어떤 데이터도 카피될 필요가 없어 폐용 메타블록들을 가비지 수거하는 것보다 빠르게 증가될 수 있기 때문이다.

[0244] 다음 한 세트의 단계들 360, 362, 366은 파일들, 공통 메타블록들 및 폐용 메타블록들에 대해 프로세스가 다르기 때문에, 선택된 대기열 엔트리에 대해서, 가비지 수거의 대상을 결정한다. 문의(360)는 가비지 수거되고 있는 파일이 있는지를 묻는다. 그러하다면, 어떤 카운터들 및 카운트를 설정하기 위해 파일에 대한 FIT 엔트리들이 단계 370 및 단계 372에서 순서대로 독출된다. 단계 370에서, 제1 카운터는 파일 내 데이터 그룹들의 수로 설정되고 제2 카운터는 파일의 제1 데이터 그룹 내 데이터의 메타페이지들(길이)의 수로 설정된다. 이 길이는 FIT 데이터 그룹 엔트리들의 길이들이 이들의 엔트리들의 일부가 아니라면 이들 엔트리들로부터 계산될 수 있다. 대안적으로, 필요로 될 때마다 데이터 그룹 길이를 계산할 필요성을 제거하기 위해서, 도 20 및 도 22에 도시된 바와 같이, 각 데이터 그룹의 길이는 FIT 내 엔트리의 일부로서 포함될 수도 있다. 단계 372에서, 제3 카운터는 폐용 데이터 혹은 다른 파일들로부터 데이터를 내포하는 메모리 블록들에 저장된 현 파일 데이터의 메타페이지들의 수로 설정된다. 이들은 옮겨질 필요가 있는 현 파일의 데이터이다. 이 제3 카운트는 바람직하게는 파일 내 모든 폐용 및 무효 데이터 그룹들을 무시한다. 마지막으로, 파일의 유효 데이터가 정수개의 메타블록들에 컴팩트되어 채워졌을 경우 나타날 데이터의 잔여 메타페이지들의 수가 카운트된다. 즉, 잔여 메타페이지 카운트는, 파일의 데이터를 내포하는 모든 메타블록들이 가비지 수거 내 포함된다면 공통의 혹은 부분적으로 채워진 메타블록을 점유하게 될, 메타블록 미만인, 파일의 데이터량이다. 이들 3개의 카운터들이 설정되고 잔여 메타페이지 카운트가 행해져 저장된 후에, 다음 단계 359는 파일의 데이터를 가비지 수거하는 것을 시작한다.

[0245] 단계 360으로 되돌아가서, 단계 358에 의해 선택된 엔트리가 파일이 아니라면, 다음 문의(362)는 이것이 가비지 수거할 공통 메타블록인지를 판정한다. 그러하다면, 도 32의 공통 메타블록 가비지 수거(364)가 수행된다. 그렇지 않다면, 문의(366)는 선택된 엔트리가 도 30의 단계 345에 의해 폐용 메타블록 대기열에 추가된 것과 같은 폐용 메타블록에 대한 것인지를 판정한다. 그러하다면, 선택된 메타블록은 단계 368에서 소거된다. 공통 메타블록 가비지 수거(364) 후에, 메타블록 소거(368) 혹은 문의(366)가 부정 응답으로 된 경우, 처리는 도 26의 단계 253으로 되돌아간다. 가비지 수거 동작이 미완료된 상태에 있다면, 가비지 수거 알고리즘에 진입되는 다음 번에 개재될 것이다.

[0246] 단계 355로 가서, 미완 가비지 수거가 있다면, 이것이 또한 처리될 것이다. 도 31의 단계 357는 가비지 수거 대기열로부터 선택된 새로운 아이템에 대한 단계 360처럼, 미안 동작이 파일에 대한 것인지 아닌지를 판정한다. 파일에 대한 것이 아니라면, 새로운 아이템에 대해 위에 기술된 단계들 362, 364, 366, 368의 프로세스가 실행되고 도 26의 단계 253으로 되돌아감으로써 가비지 수거가 종료된다.

[0247] 도 31의 단계 357에 의해서, 재개된 가비지 수거가 파일에 대한 것으로 판정된다면, 문의(359)가 다음에 행해진다. 이어지는 처리는 가비지 수거가 재개되는(단계 357을 통해) 파일에 대해서, 혹은 카운터들 및 잔여 메타페이지 카운트가 단계 370 및 단계 372에 의해 설정되는 대기열(단계 360을 통해)로부터 선택된 새로운 파일에 대해서 실질적으로 동일하다. 파일의 가비지 수거가 진행중이고 단계 357을 통해 재개되고 있을 때, 데이터 그룹 수 및 파일 메타페이지 수 카운터들과 잔여 메타페이지 카운트는 파일의 가비지 수거가 처음 시작되었을 때 설정되었다. 데이터 그룹 및 메타페이지 카운터들은 현재 파일의 초기의 부분적 가비지 수거에 의해 원래 계산된 것과는 다른 값들로 감분되어 있을 것이다. 모든 4개는 파일의 가비지 수거가 중지되고 이어서 동일 파일의 재개된 처리동안 액세스될 때 저장된다.

[0248] 공통 단계 359는 아직 카피되지 않은 현재 데이터 그룹 내 하나 이상의 메타페이지들이 있는지 문의한다. 이것은 데이터 그룹 길이 카운터에 참조에 의해 판정된다. 유효 데이터 그룹의 한 메타페이지가 검사되고 후속되는 단계들에 의해 한번에 카피된다. 데이터가 단계 359에 의해 미완 데이터 그룹 내 잔류된 것으로 판정되면, 다음 문의(361)는 카피 메타블록이 열려있는지 판정한다. 그러하다면, 단계 363에 의해, 가비지 수거되는 파일의 현 데이터 그룹의 한 메타페이지의 데이터가 이것이 저장된 메타블록으로부터 독출되고, 단계 365에서, 카피 메타블록에 기입된다. 데이터 그룹 기리 카운터는 한 메타페이지만큼 단계 367에서 감분되고 파일 메타페이지 카운터는 단계 369에서 1만큼 감분된다.

[0249] 단계 361에 의해서 카피 메타블록이 열리지 않은 것으로 판정되면, 다음 단계 371은 가비지 수거된 파일이 열린 파일인지를 판정한다. 그러하다면, 다음 단계 373은 카피 메타블록을 열고, 이어서 프로세스는 앞에서 기술된 단계 363으로 진행한다. 그러나, 열린 파일이 아니라면, 다음 단계 375는 파일 메타페이지 카운터의 감분된 카

운트가 잔여 메타페이지 카운트와 동일한지를 문의한다. 그렇지 않다면, 카피 메타블록은 단계 373에 의해 열린다. 그러나 이들이 동일하다면, 이것은 카피할 현 파일에 남아있는 메타블록 데이터가 없음을 의미한다. 그러므로, 단계 377에서, 열린 공통 메타블록은 잔여 데이터를 그에 카피하기 위해 확인된다. 단계 363 및 이후의 단계들의 처리는 현 파일의 현 데이터 그룹의 남은 메타페이지들을 공통 메타블록에 카피한다. 이 경로는 또 다른 메타블록을 채울 현 파일에 다른 데이터 그룹들이 기입될 수도 있을 것이기 때문에, 단계 371에 의해 결정된 바와 같이, 가비지 수거되고 있는 열린 파일일 때는 취해지지 않는다. 열린 파일이 닫힌 후에, 이것은 가비지 수거를 위한 대기열에 두어지고, 이 때 어떤 잔여 데이터든지 단계 377에 의해 공통 메타블록에 카피된다.

[0250] 도 31의 문의(379)는 이 추가의 메타페이지의 데이터를 기입한 후에, 카피 메타블록이 이제 충만되었는지를 판정한다. 그러하다면, 카피 메타블록은 단계 381에서 닫혀지고 FIT는 이 사실을 반영하기 위해 단계 383에 의해 업데이트된다. 카피 메타블록이 충만되지 않았거나 단계 383 이후에, 문의(385)는 가비지 수거되는 메타블록 내 모든 데이터가 이제 폐용으로 되었는지를 판정한다. 그러하다면, 메타블록은 단계 387에서 소거된다. 그렇지 않다면, 혹은 메타블록 소거 후에, 문의(389)는 프로그래밍 동작 동안 도 28의 단계 325에 의해 포그라운드 모드가 설정되는지를 묻는다. 그렇지 않다면, 도 31의 단계 391은 호스트 명령이 미완인지를 판정한다. 그러하다면, 그리고 단계 392에 의해 판정된 바와 같이, 진행중의 가비지 수거가 우선이 아니라면, 가비지 수거는 단계 393에서 FIT를 업데이트함으로써 중지되고 이어서 도 26의 단계 253으로 돌아간다. 그러나, 단계 391에 의해 판정된 바와 같이, 호스트 명령이 미완이 아니라면, 혹은 단계 392에 의해 판정된 바와 같이, 미완 호스트 명령이 있고 진행중의 가비지 수거가 우선이 아니라면, 처리는 문의(359)로 돌아가서 현 파일 메타블록으로부터 다음 메타페이지의 데이터를 카피 메타블록에 카피한다.

[0251] 도 31의 문의(389)에서 포그라운드 모드가 설정된 것으로 판정하면, 다음 단계 395는 도 28의 단계 327에 의해 리셋된 카피 메타블록 메타페이지를 증분한다. 단계 397에서 기설정된 수 N4의 메타페이지들이 연속하여 카피된 것으로 판정하면, 다음 단계 399는 포그라운드 모드를 리셋하고 FIT는 단계 393에 의해 업데이트된다. 그러나, 수 N4에 도달되기 전에 카피할 더 많은 메타페이지들이 있다면, 프로세스는 문의(391)에서 미완 호스트 명령이 있다고 판정하지 않으면 단계 359로 계속된다. 미완 호스트 명령이 있다면, 카피 동작은 중단되고 프로세스는 도 26의 단계 253으로 돌아간다.

[0252] 문의(359)로 돌아가서, 데이터 그룹 길이 카운트가 제로이면, 이것은 파일 메타블록으로부터 카피 메타블록으로 한 데이터 그룹의 카피가 완료되었음을 의미한다. 이 상황에서, 단계 401은 이 상태를 반영하기 위해 FIT를 업데이트한다. 다음에, 단계 403에서, 데이터 그룹 수 카운터에 참조함으로써, 가비지 수거되는 현 파일이 또 다른 데이터 그룹을 포함하는지가 판정된다. 그렇지 않다면, 파일의 가비지 수거는 완료되었다. 그러면 프로세스는 대기열로부터 순서대로 다음 파일 혹은 메타블록을 가비지 수거하기 위해 단계 356으로 되돌아간다. 현 파일의 가비지 수거의 완료는 진행중이고 재개될 수도 있을 파일의 가비지 수거는 더 이상 없었을 것임을 의미하기 때문에 단계 355로 돌아가지 않을 것이다.

[0253] 단계 403으로부터 카피할 현 파일의 적어도 하나 이상의 데이터 그룹이 있는 것으로 알려진다면, 단계 405에서 이 다음 데이터 그룹을 내포하는 메타블록이 폐용된 다른 데이터로 포함하는지가 판정된다. 앞에서 기술된 바와 같이, 파일의 폐용 데이터를 내포하는 메타블록들이 가비지 수거되나 파일의 폐용 데이터를 내포하지 않는 것들은 메타블록이 공통 메타블록 혹은 파일의 잔여 데이터를 내포하는 불완전한 메타블록이 아니라면 가비지 수거되지 않는 것이 바람직하다. 그러므로, 단계 405에 의해서 다음 데이터 그룹의 메타블록 내에 폐용 데이터가 있는 것으로 판정되면, 데이터 그룹 수 카운터는 단계 407에 의해서 순서대로 다음 데이터 그룹의 수로 업데이트되고, 데이터 그룹 길이 카운터는 새로운 데이터 그룹 내 데이터량으로 단계 409에서 초기화된다. 이어서 처리는 단계 361로 가서 앞에서 기술된 방식으로, 파일의 새로운 데이터 그룹으로부터 제1 메타블록의 데이터를 카피 메타블록 내 다음 소거된 메타블록에 카피한다.

[0254] 그러나, 단계 405에 의해서 다음 데이터 그룹이 존재하는 메타블록 내에 폐용 데이터가 없는 것으로 판정될지라도, 이 메타블록 내 데이터의 가비지 수거는 (1) 다음 데이터 그룹이 공통 메타블록에 있거나 (2) 완전하지 않은 메타블록에 있고 파일이 닫힌 파일이라면 여전히 행해질 수 있다. 현 데이터 그룹이 공통 메타블록에 있는지 여부에 대해 먼저 문의(411)가 행해진다. 그러하다면, 다음 단계 413은 공통 메타블록을 차후 가비지 수거를 위해 공통 메타블록 가비지 수거 대기열에 추가하나, 처리는 데이터 그룹 수 카운터 및 데이터 그룹 길이 카운터를 단계 361 및 이하 단계들에 의해 카피를 위해 다음 메타블록의 것으로 업데이트하기 위해 단계들 407 및 409로 진행한다. 그러나, 현 데이터 그룹이 공통 메타블록에 있지 않다면, 단계 415는 현 데이터 그룹이 완전하지 않은 메타블록에 있는지를 문의한다. 즉, 단계 415는 적어도 최소량의 소거된 용량(이를테면 도 15의 데이터 그룹 F3,D3)을 여전히 갖는 메타블록에 현 데이터 그룹이 존재하는지를 판정한다. 그렇지 않다면, 현 데이터 그룹

은 카피되지 않고 처리는 단계 403으로 가서 파일의 다음 데이터 그룹이 존재한다면, 이를 처리한다. 그러나, 현 데이터 그룹이 완전하지 않은 메타블록에 있다면, 다음 문의(417)는 완전하지 않은 메타블록이 열린 파일의 데이터를 내포하는지를 묻는다. 그러하다면, 현 데이터 그룹의 카피는 파일의 어떤 다른 데이터 그룹으로 진행하기 위해 문의(403)로 되돌아감으로써 스킵된다. 그러나, 다음 데이터 그룹이 존재하는 메타블록이 열린 파일의 데이터를 내포하지 않는다면, 단계들 407, 409, 361, 및 이하의 단계들이 이 다음 데이터 그룹에 대해 수행된다.

[0255] 도 31의 단계 356로 돌아가서, 가비지 수거는 가비지 수거 대기열에 파일 혹은 메타블록이 전혀 없는 것으로 판정될지라도 행해진다. 없다면, 프로세스를 종료하지 않고, 단계 421은 소거된 메타블록 풀에 N2개보다 많은 소거된 메타블록들이 있는지 체크한다. 있다면, 가비지 수거 프로세스는 종료하고 도 26의 단계 253으로 되돌아간다. 그러나, 시스템에 N2보다 많은 소거된 메타블록들이 없다면, 단계 421로부터 가비지 수거를 수행할 기회가 취해진다. 이러한 가비지 수거는, 단계 423에 나타낸 바와 같이, 열린 파일이 존재한다면 이 파일에 대해 행해질 수 있다. 대기열엔 아무 것도 없기 때문에, 풀 내에 소거된 메타블록들의 수가 N2 미만일 때 더 많은 소거된 메타블록들의 다른 잠재적 소스는 없을 것이다. 2이상의 열린 파일이 있다면, 이들 중 하나가 단계 425에서 선택된다. 이어서 처리는 앞에서 기술된 바와 같이, 단계들 370, 372, 359을 통해 열린 파일에 대해 진행한다.

[0256] 도 31의 문의(362)에 의해서 현 가비지 수거 동작이 공통 메타블록 내 데이터 그룹들에 대해 수행되고 있는 것으로 판정될 때, 가비지 수거(364)는 다른 메타블록들에 대해 위에 기술된 것과는 다소 다르다. 도 32의 흐름도는 공통 메타블록의 가비지 수거를 개괄한다. 단계 431는 가비지 수거가 진행중에 있고 이에 따라 재개되고 있는지를 묻는다. 그러하다면, 데이터 그룹 길이 카운터는 가비지 수거가 중지되었을 때 마지막 값을 보존한다. 그렇지 않다면, 데이터 그룹 수 카운터가 먼저 처리의 요체인 제1 데이터 그룹에 대한 FIT를 참조하여 단계 435에서 초기화되고 이어서 이 데이터 그룹의 길이는 단계 433에 의해 결정된다.

[0257] 데이터 그룹 길이 카운터가 단계 433에 의해 제로보다 큰 것으로 판정될 때, 현 데이터 그룹의 카피가 시작된다. 단계 437에 의해서, 현 데이터 그룹의 메타페이지는 공통 메타블록으로부터 독출되고, 단계 439에 의해서, 열린 공통 메타블록에 프로그램된다. 단계 441에서, 데이터 그룹 길이 카운터는 한 메타페이지만큼 감분된다. 단계 443에 나타낸 바와 같이, 포그라운드에서 가비지 수거를 행한다면, 단계 445는 카피 메타블록 메타페이지 카운터가 1만큼 증분되게 한다. 이 카운터는 현 순서로 카피된 데이터의 메타페이지들의 수를 주시한다. 이 카운트가 단계 447에 의해서 소정의 수 N4를 초과한 것으로 판정되면, 프로세스는 단계 449에서 포그라운드 모드에서 나간다. 이에 이어서 단계 451에서 FIT가 업데이트되고, 처리는 도 26의 단계 253으로 돌아간다.

[0258] 카피 메타블록 메타페이지 카운터가 도 32의 단계 447에 의해서 N4 미만의 값을 갖는 것으로 판정되면, 다음 단계는 호스트 명령이 미완인지를 문의한다. 그러하다면, FIT는 단계 451에서 업데이트되고 처리는 미완 호스트 명령이 실행될 수 있게, 도 26의 단계 253으로 돌아간다. 그렇지 않다면, 처리는 도 32의 단계 433으로 돌아가서 현 데이터 그룹에 또 다른 데이터 메타페이지가 존재하는지를 문의하고, 그러하다면, 이를 폐용 명령 메타블록으로부터 열린 공통 메타블록으로 카피하고, 등등을 행한다. 유사하게, 단계 443에서, 가비지 수거가 포그라운드에서 수행되고 있지 않은 것으로 판정되면, 단계 453이 다음에 실행된다. 단계 453에 의해 판정된 바와 같이, 호스트 명령이 미완이 아니면, 처리는 단계 433으로 돌아가서 단계 447이 우회되기 때문에, 카피 메타블록 메타페이지 카운터가 N4를 초과하는지에 관계없이, 현 데이터 그룹 내 다음 메타페이지의 데이터를 잠재적으로 카피한다. 이 경우 가비지 수거는 백그라운드에서 수행되고 있다.

[0259] 도 32의 단계 433으로 돌아가서, 데이터 그룹 길이 카운트가 제로보다 크지 않다면, 현 데이터 그룹의 모든 메타페이지들이 카피되었음을 안다. 이어서, FIT는 단계 455에서 업데이트되고, 단계 457에 의해서 가비지 수거되는 공통 메타블록 내에 또 다른 데이터 그룹이 있는지 판정된다. 그렇지 않다면, 단계 459에서, 폐용 공통 메타블록이 소거되고, 처리는 도 26의 단계 253으로 돌아간다. 그러나, 공통 메타블록에 또 다른 데이터 그룹이 있다면, 데이터 그룹 수 카운터는 단계 461에 의해 업데이트된다. 이어서 공통 메타블록 인덱스로부터 FIT 포인트가 단계 463에서 독출되고, 이 포인터에 의해 정의된 FIT 엔트리가 단계 465에서 독출된다. 단계 467에 의해 판정된 바와 같은 FIT 엔트리가 현 데이터 그룹과 부합한다면, 문의(468)는 현 데이터 그룹이 이미 확인된 잔여 데이터의 일부인지 판정한다. 그러하다면, 단계 469는 데이터 그룹 길이 카운터가 FIT로부터 독출된 데이터 그룹의 길이로 초기화되게 한다.

[0260] 그러나, 단계 468에서 현 데이터 그룹이 현존 잔여 데이터에 있지 않은 것으로 판정되면, 이것이 일부인 새로운 잔여 데이터가 단계 470에서 확인된다. 단계 471에서, 새로운 잔여 데이터의 모든 데이터를 저장하는데 충분한 공간을 가진 열린 공통 블록이 확인된다. 이것은 잔여 데이터가 2이상의 데이터 그룹들을 내포할 때 2이상의 서

로 다른 메타블록들간에 파일의 잔여 데이터를 분할하는 것을 방지한다. 그렇지 않다면 이것은 2이상의 데이터 그룹들이 독립적으로 카피될 경우 일어날 수도 있을 것이다. 이 경우, 제1 데이터 그룹은 이 데이터 그룹을 저장하는 충분한 소거된 공간을 가지지만 제2 데이터 그룹도 저장하기에 충분한 공간은 없는 공통 블록에 카피될 수도 있을 것이다. 이어서 제2 데이터 그룹은 상이한 공통 블록에 카피될 것이다. 이것은 파일의 잔여 데이터가 2개의 서로 다른 메타블록들간에 분할되지 않아야 하기 때문에 바람직하지 못할 것이다. 이들이 이와 같이 분할된다면, 파일 혹은 공통 블록들의 가비지 수거는 더 많은 시간이 걸릴 것이다.

[0261] 이어서 현 데이터 그룹에 대한 데이터 그룹 길이 카운터는 단계 469에 의해 초기화된다. 현 데이터 그룹의 제1 메타페이지의 데이터는 단계 437 및 단계 439에 의해, 공통 메타블록으로부터, 확인된 열린 공통 메타블록에 카피된다. 그러나, FIT 엔트리가 단계 467에 의해 현 데이터 그룹과 부합하지 않은 것으로 판정되면, 처리는 또 다른 데이터 그룹이 공통 메타블록에 존재하는지 판정하기 위해 단계 457로 되돌아간다.

[0262] 도 32의 흐름도에 도시된 가비지 수거는 현 공통 메타블록의 모든 유효 데이터 그룹들이 전에 소거된 메타블록-이것은 새로운 공통 메타블록이 된다- 에, 혹은 하나 이상의 열린 공통 메타블록들에 카피될 때까지 계속된다. 서로 다른 파일들의 데이터 그룹들은 서로 다른 열린 공통 메타블록들에 카피될 수 있다. 공통 메타블록은 이것이 폐용 데이터 그룹을 내포하였기 때문에 전에 가비지 수거 대기열에 놓여졌었다. 공통 메타블록 내 각각의 유효한 데이터 그룹의 모든 메타페이지들의 데이터의 완전한 전송은 각각의 이러한 메타페이지에 대한 도 32의 단계들을 거치는 1패스(pass)를 취한다. 이러한 카피는 가비지 수거가 포그라운드에서 행해지고 있다면, 혹은 포그라운드 혹은 백그라운드에서 동작하고 있을 때에 새로운 호스트 명령이 수신될 때 모든 N4 메타페이지들에 중단될 수 있다.

[0263] 수 N3 및 N4를 영구적으로 기설정하는 대안으로서, 이들은 균일한 데이터 프로그래밍 속도를 지속하기 위해서 호스트의 데이터 프로그래밍 패턴들에 응하여 메모리 시스템 제어기에 의해 변경될 수 있다.

[0264] 동작동안에 메모리 블록들의 여러 상태들

[0265] 도 33의 도면은 위에 기술된 유형의 직접 파일 저장 메모리 시스템 내에서, 시스템의 메모리 블록들 혹은 메타블록들의 다양한 개별적 상태들, 및 이들 상태들간 천이들을 도시한 것이다.

[0266] 좌측의 열에서, 블록(501)은 소거된 블록 풀 내에, 소거된 상태에 있는 것으로 도시된 것이다.

[0267] 다음 열에서, 블록들(503, 505, 507) 각각은 호스트로부터의 데이터가 기입될 수 있는 소거된 용량을 갖는다. 기입 블록(503)은 단일 파일에 대한 유효 데이터로 부분적으로 기입되고, 파일에 대한 또 다른 데이터는 호스트에 의해 공급될 때 이 블록에 기입될 것이다. 카피 블록(505)은 단일 파일에 대한 유효 데이터로 부분적으로 기입되고, 파일에 대한 또 다른 데이터는 파일의 가비지 수거동안 카피될 때 그에 기입될 것이다. 열린 공통 블록(507)은 2이상의 파일들에 대한 유효 데이터와 함께 부분적으로 기입되고, 임의의 파일에 대한 잔여 데이터 그룹은 가비지 수거 동안 그에 기입될 수 있다.

[0268] 다음 열의 블록들(509, 511)은 파일 데이터로 충만된다. 파일 블록(509)은 단일 파일에 대한 유효 데이터로 채워진다. 공통 블록(511)은 2이상의 파일들에 대한 유효 데이터로 채워진다.

[0269] 도 33의 우측 열 다음은 각각이 어떤 폐용 데이터를 내포하는 블록들(513, 515, 517, 519, 521)을 포함한다. 폐용 파일 블록(513)은 단일 파일에 대한 유효 데이터 및 폐용 데이터의 어떤 조합으로 채워진다. 폐용 기입 블록(515)은 단일 파일에 대한 유효 데이터 및 폐용 데이터의 어떤 조합으로 부분적으로 기입되고, 파일에 대한 또 다른 데이터는 호스트에 의해 공급될 때 그에 기입될 것이다. 폐용 카피 블록(517)은 단일 파일에 대한 유효 데이터 및 폐용 데이터의 어떤 조합으로 부분적으로 기입된다. 폐용의 열린 공통 블록(519)은 2이상의 파일들에 대한 유효 데이터 및 폐용 데이터의 어떤 조합으로 부분적으로 기입된다. 폐용 공통 블록(521)은 2이상의 파일들에 대한 유효 데이터 및 폐용 데이터의 어떤 조합으로 채워진다.

[0270] 도 33의 우측열 내 폐용 블록(523)은 폐용 데이터만을 내포한다.

[0271] 도 33에 도시된 블록 상태들간에 개개의 블록들의 천이들은 소문자들로 라벨된 선들에 의해 도시되었다. 이들 천이들은 다음과 같다.

[0272] a- 소거된 블록(501)에서 기입 블록(503)으로: 단일 호스트 파일이 데이터가 소거된 블록에 기입된다.

[0273] b- 기입 블록(503)에서 기입 블록(503)으로: 호스트로부터 단일 파일에 대한 데이터가 이 파일에 대한 기입 블

록에 기입된다.

[0274] c- 기입 블록(503)에서 파일 블록(509)으로: 호스트로부터 단일 파일에 대한 데이터가 이 파일에 대한 기입 블록을 채우기 위해 기입된다.

[0275] d- 파일 블록(509)에서 폐용 파일 블록(513)으로: 파일 블록 내 데이터의 부분은 데이터의 업데이트된 버전이 파일에 대한 기입 블록에 호스트에 의해 기입되는 결과로서 폐용된다.

[0276] e- 폐용 파일블록(513)에서 폐용 블록(523)으로: 폐용 파일블록 내 모든 유효 데이터는 데이터가 가비지 수거동안 또 다른 블록에 카피되거나 파일이 호스트에 의해 삭제되는 결과로서 폐용된다.

[0277] f- 기입블록(503)에서 폐용 기입블록(515)으로: 기입블록 내 데이터의 부분은 데이터의 업데이트된 버전이 동일 기입 블록에 호스트에 의해 기입되거나, 데이터가 가비지 수거동안 또 다른 블록에 카피되는 결과로서 폐용된다.

[0278] g- 폐용 기입 블록(515)에서 폐용 기입블록(515)으로: 호스트로부터 단일 파일에 대한 데이터가 이 파일에 대한 폐용 기입블록에 기입된다.

[0279] h- 폐용 기입 블록(515)에서 폐용 파일블록(513)으로: 호스트로부터 단일 파일에 대한 데이터는 이 파일에 대한 폐용 기입블록을 채우기 위해 기입된다.

[0280] i- 폐용 기입블록(515)에서 폐용 블록(523)으로: 폐용 기입블록 내 모든 유효 데이터는 데이터가 가비지 수거동안 또 다른 블록에 카피되거나 파일이 호스트에 의해 삭제되는 결과로서 폐용된다.

[0281] j- 소거된 블록(501)에서 카피 블록(505)으로: 단일 파일에 대한 데이터가 가비지 수거동안 또 다른 블록으로부터 소거된 블록으로 카피된다.

[0282] k- 기입 블록(503)에서 카피 블록(505)으로: 단일 파일에 대한 데이터가 가비지 수거 동안 이 파일에 대해 또 다른 블록으로부터 기입 블록에 기입된다.

[0283] l- 카피 블록(505)에서 카피 블록(505)으로: 단일 파일에 대한 데이터가 가비지 수거동안 이 파일에 대해 또 다른 블록으로부터 카피 블록으로 카피된다.

[0284] m- 카피 블록(505)에서 파일 블록(509)으로: 단일 파일에 대한 데이터가 가비지 수거동안 이 파일에 대해 또 다른 블록으로부터 카피 블록으로 카피된다.

[0285] n- 카피 블록(505)에서 기입 블록(503)으로: 호스트로부터 단일 파일에 대한 데이터가 가비지 수거동안 파일이 다시 열렸을 때 이 파일에 대해 카피 블록에 기입된다.

[0286] o- 카피 블록(505)에서 폐용 카피 블록(517)으로: 카피 블록 내 일부 혹은 모든 데이터는 데이터의 업데이트된 버전이 호스트에 의해 파일에 대해 기입 블록에 기입되거나 파일이 호스트에 의해 삭제되는 결과로서 폐용된다.

[0287] p- 폐용 카피 블록(517)에서 폐용 블록(523)으로: 폐용 카피 블록 내 모든 유효 데이터는 데이터가 가비지 수거동안 또 다른 블록에 카피되거나 파일이 호스트에 의해 삭제되는 결과로서 폐용된다.

[0288] q- 기입 블록(503)에서 열린 공통 블록(507)으로: 파일에 대한 잔여 데이터는 가비지 수거동안 다른 닫혀진 파일에 대한 기입 블록에 기입된다.

[0289] r- 카피 블록(505)에서 열린 공통 블록(507)으로: 파일에 대한 잔여 데이터가 가비지 수거 동안 다른 파일에 대한 잔여 데이터를 내포하는 카피 블록에 기입된다.

[0290] s- 열린 공통 블록(507)에서 열린 공통 블록(507)으로: 파일에 대한 잔여 데이터는 가비지 수거 동안 다른 블록에서 열린 공통 블록으로 카피된다.

[0291] t- 열린 공통 블록(507)에서 폐용 열린 공통 블록(519)으로: 열린 공통 블록 내 한 파일에 대한 일부 혹은 모든 데이터는 데이터의 업데이트된 버전이 파일에 대한 기입 블록에 호스트에 의해 기입되거나, 데이터가 가비지 수거 동안 또 다른 블록에 카피되거나, 파일이 호스트에 의해 삭제되는 결과로서 폐용된다.

[0292] u- 폐용 열린 공통 블록(519)에서 폐용 블록(523)으로: 폐용 열린 공통 블록 내 모든 유효 데이터는 데이터가 가비지 수거동안 또 다른 블록에 카피되거나 파일이 호스트에 의해 삭제되는 결과로서 폐용된다.

[0293] v- 열린 공통블록(507)에서 공통 블록(511)으로: 파일에 대한 잔여 데이터 그룹은 가비지 수거 동안 그 파일에

대한 열린 공통 블록을 채우기 위해 또 다른 블록으로부터 카피된다.

[0294] w- 공통 블록(511)에서 폐용 공통 블록(521)에서: 공통 블록 내 한 파일에 대한 일부 혹은 모든 데이터는 데이터의 업데이트된 버전이 파일에 대한 기입 블록에 호스트에 의해 기입되거나, 데이터가 가비지 수거동안 또 다른 블록에 카피되거나, 파일이 호스트에 의해 삭제되는 결과로서 폐용된다.

[0295] x- 폐용 공통 블록(521)에서 폐용 블록(523)으로: 폐용 공통 블록 내 모든 유효 데이터는 데이터가 가비지 수거동안 또 다른 블록에 카피되거나 파일이 호스트에 의해 삭제되는 결과로서 폐용된다.

[0296] y- 기입 블록(503)에서 폐용 블록(523)으로: 기입 블록 내 단일 파일에 대한 모든 유효한 데이터는 파일이 호스트에 의해 삭제된 결과로서 폐용된다.

[0297] z- 카피 블록(505)에서 폐용 블록(523)으로: 카피 블록 내 모든 유효한 데이터는 데이터가 가비지 수거 동안 또 다른 블록에 카피되거나 파일이 호스트에 의해 삭제되는 결과로서 폐용된다.

[0298] aa- 파일 블록(509)에서 폐용 블록(523)으로: 파일 블록 내 모든 데이터는 파일이 호스트에 의해 삭제된 결과로서 폐용된다.

[0299] ab- 폐용 블록(523)에서 소거된 블록(501)로: 폐용 블록은 가비지 수거 동안 소거된다.

[0300] 결론

[0301] 본 발명의 여러 가지 면들이 이의 실시예들에 관하여 기술되었을지라도, 본 발명은 첨부된 청구항들의 전체 범위 내에서 보호됨을 알 것이다.

도면의 간단한 설명

[0011] 도 1은 현재 구현된 호스트 및 접속된 비휘발성 메모리 시스템을 개략적으로 도시한 것이다.

[0012] 도 2는 도 1의 비휘발성 메모리로서 사용하기 위한 플래시 메모리 시스템 예의 블록도이다.

[0013] 도 3은 도 2의 시스템에서 사용될 수 있는 메모리 셀 어레이의 대표적 회로도이다.

[0014] 도 4는 도 2의 시스템의 물리적 메모리 구성의 예를 도시한 것이다.

[0015] 도 5는 도 4의 물리적 메모리의 부분을 확대한 도면이다.

[0016] 도 6은 도 4 및 도 5의 물리 메모리의 부분을 더욱 확대한 도면이다.

[0017] 도 7은 호스트와 재프로그램 가능한 메모리 시스템간에 공통의 종래 기술의 어드레스 인터페이스를 도시한 것이다.

[0018] 도 8은 호스트와 재프로그램가능한 메모리 시스템간의 공통의 종래 기술의 논리 어드레스 인터페이스를 도 7과는 다른 방식으로 도시한 것이다.

[0019] 도 9는 본 발명에 따라, 호스트와 재프로그램가능한 메모리 시스템간의 직접 파일 저장 인터페이스를 도시한 것이다.

[0020] 도 10은 본 발명에 따라, 호스트와 재프로그램가능한 메모리 시스템간의 직접 파일 저장 인터페이스를 도 9와는 다른 방식으로 도시한 것이다.

[0021] 도 11은 예로서의 메모리 시스템의 기능 계층도이다.

[0022] 도 12a-12e는 예로서의 한 세트의 직접 파일 인터페이스 명령들을 제공한다.

[0023] 도 13a-13d는 메모리에 직접 데이터 파일들을 기입하는 4개의 서로 다른 예들을 도시한 것이다.

[0024] 도 14a-14e는 메모리에 직접 단일 데이터 파일을 기입하는 순서를 도시한 것이다.

[0025] 도 15는 도 14e에 도시한 데이터 파일을 가비지 수거한 결과를 도시한 것이다.

[0026] 도 16은 공통 블록의 예를 제공한다.

[0027] 도 17은 몇 개의 열린 공통 블록들 중 한 블록에 공통 데이터 그룹을 프로그램하는 것을 도시한 것이다.

[0028] 도 18은 서로 다른 파일들에 비휘발성 메모리에 프로그램된 메타페이지들의 데이터의 몇가지 예들을 도시한 것이다.

[0029] 도 19는 파일 인덱스 테이블(FIT)의 구조 및 도 14a, 14c, 14e, 15의 예들로부터 엔트리들을 도시한 것이다.

[0030] 도 20은 예로서의 파일 인덱싱 데이터 구조를 개념적으로 도시한 것이다.

[0031] 도 21은 도 20의 파일 디렉토리의 페이지들의 구조를 도시한 것이다.

[0032] 도 22는 도 20의 파일 인덱스 테이블의 페이지들의 구조를 도시한 것이다.

[0033] 도 23은 도 21의 구조에 대한 대안으로서, 도 20의 파일 디렉토리의 페이지들에 대한 구조를 도시한 것이다.

[0034] 도 24는 도 21 및 도 23의 파일 디렉토리들의 운영을 도시한 것이다.

[0035] 도 25는 도 22의 파일 인덱스 테이블의 운영을 도시한 것이다.

[0036] 도 26은 여기에 기술된 메모리 시스템의 동작들의 전체 순서를 도시한 흐름도이다.

[0037] 도 27은 도 26의 "파일 데이터 독출" 블록의 흐름도이다.

[0038] 도 28은 도 26의 "파일 데이터 프로그램" 블록의 흐름도이다.

[0039] 도 29는 도 28의 흐름도에 포함된 2개의 동작들의 상대적 타이밍을 도시한 것이다.

[0040] 도 30은 도 26의 "파일 삭제" 블록의 흐름도이다.

[0041] 도 31은 도 26의 "가비지 수거" 블록의 흐름도이다.

[0042] 도 32는 도 31의 "공통 블록 가비지 수거" 블록의 흐름도이다.

[0043] 도 33은 여기에서의 예로서의 메모리 시스템의 기술된 동작 동안 메모리 셀 블록들의 상태도이다.

[0044] 플래시 메모리 시스템의 전반적인 설명

[0045] 현재의 플래시 메모리 시스템, 및 호스트 디바이스들의 전형적인 동작을 도 1-8에 관하여 기술한다. 이러한 시스템에서 본 발명의 여러 가지 면들이 구현될 수 있다. 도 1의 호스트 시스템(1)은 데이터를 플래시 메모리(2)에 저장하고 이로부터 데이터를 가져온다. 플래시 메모리가 호스트 내 내장될 수 있을지라도, 메모리(2)는 기구적 및 전기적 커넥터의 서로 짝이되는 부분들(3, 4)을 통해 호스트에 착탈가능하게 연결되는 보다 보급된 형태의 카드인 것으로 도시되었다. 시판되는 현재 많은 서로 다른 플래시 메모리 카드들이 있고, 예들로서는 CompactFlash(CF), MultiMediaCard (MMC), SD(Secure Digital), miniSD, 메모리 스틱, SmartMedia 및 TransFlash 카드들이다. 이들 카드들 각각이 이의 표준화된 명세들에 따른 고유한 기구적 및/또는 전기적 인터페이스를 구비할지라도, 각각에 포함된 플래시 메모리는 매우 유사하다. 이들 카드들은 모두 본 출원의 양수인인 샌디스크 사로부터 입수될 수 있다. 또한, 샌디스크는 호스트의 USB(Universal Serial Bus) 리셉터클에 끼움으로써 호스트에 접속하기 위한 USB 플러그를 구비하는 소형 패키지들의 휴대 메모리 시스템들인 크루저 등록상표의 일련의 플래시 드라이브들을 제공한다. 이들 메모리 카드들 및 플래시 드라이브들 각각은 호스트와 인터페이스하고 이들 내 플래시 메모리의 동작을 제어하는 제어기들을 포함한다.

[0046] 이러한 메모리 카드들 및 플래시 드라이브들을 사용하는 호스트 시스템들은 많고 다양하다. 이들은 개인용 컴퓨터들(PC), 랩탑 및 그외 휴대 컴퓨터들, 셀룰라 전화들, PDA(personal digital assistants), 디지털 스틸 카메라들, 디지털 무비 카메라들 및 휴대 오디오 플레이어들을 포함한다. 통상적으로 호스트는 하나 이상의 유형들의 메모리 카드들 혹은 플래시 드라이브들을 위한 내장 리셉터클을 포함하나 어떤 것들은 메모리 카드를 끼워넣는 어댑터들을 필요로 한다.

[0047] 도 1의 호스트 시스템(1)은 메모리(2)에 관한 한, 회로와 소프트웨어의 조합으로 구성되는 2개의 주요 부분들을 갖는 것으로 간주될 수 있다. 이들은 애플리케이션부(5), 및 메모리(2)와 인터페이스하는 드라이버부(6)이다. 예를 들면, 개인용 컴퓨터에서, 애플리케이션부(5)는 워드 프로세싱, 그래픽스, 제어 혹은 이외 인기있는 애플리케이션 소프트웨어를 실행시키는 프로세서를 포함할 수 있다. 카메라, 셀룰라 전화 혹은 이외 주로 단일의 한 세트의 기능들을 수행하는데 전용인 호스트 시스템에서, 애플리케이션부(5)는 사진들을 촬영하여 저장하는 카메라, 전화를 걸고 받는 셀룰라 전화, 등을 동작시키는 소프트웨어를 포함한다.

[0048] 도 1의 메모리 시스템(2)은 플래시 메모리(7), 및 데이터를 주고받기 위해 카드가 접속되는 호스트와 인터페이

스하고 메모리(7)를 제어하는 회로들(8)을 포함한다. 통상적으로 제어기(8)는 프로그래밍 및 독출시 호스트(1)에 의해 사용되는 데이터의 논리 어드레스들과 메모리(7)의 물리 어드레스들간을 변환한다.

[0049] 도 2를 참조하여, 도 1의 비휘발성 메모리(2)로서 사용될 수 있는 전형적인 플래시 메모리 시스템의 회로를 기술한다. 시스템 제어기는 통상적으로, 시스템 버스(13)로 하나 이상의 집적회로 메모리 칩들과 병렬로 접속되는 단일의 집적회로(11)에 구현되고, 단일의 이러한 메모리 칩(15)이 도 2에 도시되었다. 도시된 특정의 버스(13)는 데이터를 전달하는 개별적인 한 세트의 도체들(17), 메모리 어드레스들용의 세트(19), 제어 및 상태 신호들용의 세트(21)를 포함한다. 대안적으로, 단일의 한 세트의 도체들은 이들 3개의 기능들간에 시-분할된다. 또한, 이를테면 "Ring Bus Structure and It's Use in Flash Memory Systems" 명칭으로 2004년 8월 9일 출원된 미국특허출원 번호 10/915,039에 기술된 링 버스와 같은 시스템 버스들의 그외 다른 구성들이 채용될 수도 있다.

[0050] 전형적인 제어기 칩(11)은 인터페이스 회로들(25)을 통해 시스템 버스(13)와 인터페이스하는 그 자신의 내부 버스(23)를 구비한다. 버스에 정규로 접속되는 주 기능들은 프로세서(27)(이를테면 마이크로프로세서 혹은 마이크로제어기), 시스템을 초기화("부팅")하기 위한 코드를 내장한 독출전용 메모리(ROM)(29), 주로 메모리와 호스트간에 전송되는 데이터를 버퍼링하는데 사용되는 랜덤 액세스 메모리(RAM)(31), 및 메모리와 호스트간의 제어기를 통해 전달되는 데이터에 대한 오류정정 코드(ECC)를 계산하고 체크하는 회로들(33)이다. 제어기 버스(23)는 회로들(35)을 통해 호스트 시스템과 인터페이스하는데, 이것은 도 2의 시스템이 메모리 카드 내 내장되는 경우, 커넥터(4)의 일부인 카드의 외부 콘택들(37)을 통해 행해진다. 클럭(39)은 제어기(11)의 다른 구성성분들 각각에 접속되고 이에 의해 이용된다.

[0051] 시스템 버스(13)에 접속되는 어떤 다른 것뿐만 아니라 메모리 칩(15)은 통상적으로, 복수의 서브-어레이들 또는 플레인들로 구성되는 한 어레이의 메모리 셀들을 내장하는데, 간이성을 위해서 2개의 이러한 플레인들(41, 43)이 도시되었지만 그러나 그 이상의, 이를테면 4 혹은 8개의 이러한 플레인들이 사용될 수도 있다. 대안적으로, 칩(15)의 메모리 셀 어레이는 플레인들로 분할되지 않을 수도 있다. 그러나 그와 같이 분할되었을 때, 각 플레인은 서로 독립적으로 동작할 수 있는 그 자신의 열(column) 제어 회로들(45, 47)을 구비한다. 회로들(45, 47)은 시스템 버스(13)의 어드레스부(19)로부터 그들 각각의 메모리 셀 어레이의 어드레스들을 수신하고, 이들을 디코딩하여 특정의 하나 이상의 각각의 비트라인들(49, 51)을 어드레싱한다. 워드라인들(53)은 어드레스 버스(19) 상에 수신된 어드레스들에 응하여 행 제어 회로들(55)을 통해 어드레스된다. 소스 전압 제어회로들(57, 59)은 p-웰 전압 제어회로들(61, 63)처럼 각각의 플레인들에 접속된다. 메모리 칩(15)이 단일의 한 어레이의 메모리 셀들을 구비하고 2 이상의 이러한 칩들이 시스템에 존재한다면, 각 칩의 어레이는 위에 기술된 복수-플레인 칩 내 플레인 혹은 서브-어레이와 유사하게 동작될 수 있다.

[0052] 데이터는 시스템 버스(13)의 데이터부(17)에 접속되는 각각의 데이터 입력/출력 회로들(65, 67)을 통해 플레인들(41, 43)에 및 이들로부터 전송된다. 회로들(65, 67)은 각각의 열 제어회로들(45, 47)을 통해 플레인들에 접속된 라인들(69, 71)을 통해, 데이터를 메모리 셀들에 프로그램하는 것과 이들의 각각의 플레인들의 메모리 셀들로부터 데이터를 독출하는 것 둘 다를 제공한다.

[0053] 제어기(11)가 데이터를 프로그램하고, 데이터를 독출하고 소거하고 각종 하우스키핑(housekeeping) 사안들을 수행하게 메모리 칩(15)의 동작을 제어할지라도, 각 메모리 칩은 이러한 기능들을 수행하기 위해 제어기(11)로부터 명령들을 실행하는 어떤 제어회로를 또한 내장한다. 인터페이스 회로들(73)은 시스템 버스(13)의 제어 및 상태부(21)에 접속된다. 제어기로부터의 명령들은 상태 머신(75)에 제공되고 이 상태 머신(75)은 이어서 이들 명령들을 실행하기 위해 다른 회로들의 특정의 제어를 제공한다. 제어라인들(77-81)은 상태 머신(75)을 도 2에 도시된 바와 같은 이들 다른 회로들에 접속한다. 상태 머신(75)으로부터의 상태 정보는 버스부(21)로 제어기(11)에의 전송을 위해 라인들(83)을 통해 인터페이스(73)에 보내진다.

[0054] 메모리 셀 어레이들(41, 43)의 NAND 구조가 현재 바람직하나, NOR와 같은 다른 구조들도 대신에 사용될 수 있다. NAND 플래시 메모리들 및 메모리 시스템의 이들 메모리들의 동작의 예들은 미국특허 5,570,315, 5,774,397, 6,046,935, 6,373,746, 6,456,528, 6,522,580, 6,771,536, 6,781,877 및 미국특허출원 공개 번호 2003/0147278에의 참조에 의해 취해질 수 있다.

[0055] 예로서의 NAND 어레이가 도 2의 메모리 시스템의 메모리 셀 어레이(41)의 부분인 도 3의 회로도에 의해 예시되어 있다. 많은 수의 전역 비트라인들이 제공되지만, 단지 4개의 이러한 라인들(91 - 94)이 설명의 간략성을 위해 도 2에 도시되었다. 다수의 직렬 연결된 메모리 셀 스트링들(97 - 104)은 이들 비트라인들 중 하나와 기준전위간에 접속된다. 대표로서 메모리 셀 스트링(99)을 사용하면, 복수의 전하 저장 메모리 셀들(107-110)은 스트링의 양 단부에 선택 트랜지스터들(111, 112)에 직렬로 접속된다. 스트링의 선택 트랜지스터들이 도통하게 되었

을 때, 스트링은 이의 비트라인과 기준전위간에 접속된다. 그러면 이 스트링 내 한 메모리 셀이 한번에 프로그램되거나 독출된다.

[0056] 도 3의 워드라인들(115-118)은 개별적으로 다수의 스트링들의 메모리 셀들의 각 스트링 내 한 메모리 셀의 전하 저장 요소를 가로질러 연장되고, 게이트들(119, 120)은 스트링들의 각 단부의 선택 트랜지스터들의 상태들을 제어한다. 공통 워드 및 제어 게이트 라인들(115-120)을 공유하는 메모리 셀 스트링들은 함께 소거되는 한 블록(123)의 메모리 셀들을 형성하게 구성된다. 이 블록의 셀들은 한번에 물리적으로 소거될 수 있는 최소 수의 셀들을 내포한다. 워드라인들(115-118) 중 하나를 따라 있는 것들인 한 행의 메모리 셀들은 한번에 프로그램된다. 통상적으로, NAND 어레이의 행들은 규정된 순서로 프로그램되는데, 이 경우엔 접지 혹은 또 다른 공통 전위에 연결된 스트링들의 단부에 가장 가까운 워드라인(118)을 따른 행부터 시작한다. 워드라인(117)을 따른 한 행의 메모리 셀들이 다음에 프로그램되고, 등등이 블록(123) 전체에 걸쳐 행해진다. 워드라인(115)을 따른 행은 마지막에 프로그램된다.

[0057] 제2 블록(125)도 유사한데, 이의 스트링들의 메모리 셀들은 제1 블록(123) 내 스트링들과 동일한 전역 비트라인들에 접속되나 다른 한 세트의 워드 및 제어 게이트 라인들을 갖는다. 워드 및 제어 게이트 라인들은 이들의 적합한 동작 전압들로 행 제어회로들(55)에 의해 구동된다. 도 2의 플레인 1 및 플레인 2와 같이 시스템에 1 이상 플레인 혹은 서브-어레이가 있다면, 한 메모리 구조들은 이들간에 연장하는 공통 워드라인들을 사용한다. 대안적으로 공통의 워드라인들을 공유하는 3이상 플레이들 혹은 서브-어레이들이 있을 수 있다. 다른 메모리 구조들에서, 개개의 플레인들 혹은 서브-어레이의 워드라인들은 개별적으로 구동된다.

[0058] 위에서 참조된 NAND 특허들 및 공개된 출원 중 몇몇에 기술된 바와 같이, 메모리 시스템은 각 전하 저장요소 혹은 영역 내 2 이상의 검출가능한 레벨들의 전하를 저장하고, 그럼으로써 각각에서 1비트 이상의 데이터를 저장하게 동작될 수 있다. 메모리 셀들의 전하 저장요소들은 가장 공통적으로 도전성 플로팅 게이트들이지만 대안적으로 미국특허출원 공개 2003/0109093에 기술된 바와 같이, 비도전성 유전체 전하 트랩 물질일 수도 있다.

[0059] 도 4는 이하 설명에서 예로서 사용되는 플래시 메모리 셀 어레이(7)(도 1)의 구성을 개념적으로 도시한 것이다. 메모리 셀들의 4개의 플레인들 혹은 서브-어레이들(131-134)은 단일 집적 메모리 셀 칩 상에, 혹은 2개의 칩들(각 칩 상에 2개의 플레인들), 혹은 4개의 개별적 칩들 상에 있을 수 있다. 특정한 배열은 이하 논의에 중요하지 않다. 물론, 다른 개수의 플레인들, 이를테면 1, 2, 8, 16 혹은 그 이상이 시스템에 존재할 수 있다. 플레인들은 각각의 플레인들(131-134)에 놓여지는, 이를테면 블록들(137, 138, 139, 140)과 같은, 사각형들로 도 4에 도시된 다수 블록들의 메모리 셀들로 개개가 분할된다. 각 플레인에 수십 혹은 수백 개의 블록들이 있을 수 있다. 위에 언급된 바와 같이, 한 블록의 메모리 셀들은 물리적으로 함께 소거될 수 있는 최소 수의 메모리 셀들인 소거단위이다. 그러나, 증가된 병행도를 위해서, 블록들은 더 큰 메타블록 단위들로 동작된다. 각 플레인으로부터 한 블록은 논리적으로 함께 링크되어 메타블록을 형성한다. 4개의 블록들(137-140)이 하나의 메타블록(141)을 형성하는 것으로 도시되었다. 메타블록 내 모든 셀들은 통상적으로 함께 소거된다. 블록들(145-148)로 구성된 제2 메타블록(143)에 보인 바와 같이, 메타블록을 형성하는데 사용되는 블록들은 이들의 각각의 플레인들 내에 동일 상대적 위치들로 제약될 필요는 없다. 통상적으로 모든 플레인들에 걸쳐 메타블록들을 연장하는 것이 바람직할지라도, 높은 시스템 수행을 위해서, 메모리 시스템은 서로 다른 플레인들 내 하나, 둘 혹은 세 개의 블록들 중 하나 혹은 전부의 메타블록들을 동적으로 형성하는 능력을 갖고 동작될 수 있다. 이것은 메타블록의 크기가 한 프로그래밍 동작에서 저장을 위해 사용될 수 있는 데이터량에 더 가깝게 부합되게 한다.

[0060] 개개의 블록들은 이번에는 사용 목적을 위해 도 5에 도시된 바와 같이, 다수 페이지들의 메모리 셀들로 분할된다. 예를 들면, 블록들(131-134)의 각각의 메모리 셀들은 각각 8개의 페이지들(P0-P7)로 분할된다. 대안적으로, 각 블록 내에 16, 32, 혹은 그 이상의 페이지들의 메모리 셀들이 있을 수 있다. 페이지는 한번에 프로그램되는 최소 데이터량을 내포하는, 블록 내 데이터 프로그래밍 및 독출 단위이다. 도 3의 NAND 구조에서, 페이지는 블록 내 워드라인을 따른 메모리 셀들로 형성된다. 그러나, 메모리 시스템 동작의 병행도를 증가시키기 위해서, 2 이상 블록들 내 이러한 페이지들은 논리적으로 메타페이지들에 링크될 수 있다. 메타페이지(151)는 도 5에 도시되었고, 4개의 블록들(131-134) 각각으로부터 한 물리적 페이지로 형성된다. 예를 들면, 메타페이지(151)는 4개의 블록들 각각의 블록 내에 페이지(P2)를 포함하는데, 메타페이지의 페이지들은 반드시 블록들의 각 블록 내에 동일한 상대적 위치를 가질 필요는 없다. 높은 시스템 수행을 위해서, 모든 4개의 플레인들에 걸쳐 병렬로 최대량의 데이터를 프로그램하고 독출하는 것이 바람직할지라도, 메모리 시스템은 서로 다른 플레인들에서 개별적 블록들 내 하나, 둘 혹은 세 개의 페이지들 중 하나 혹은 전부의 메타페이지들을 형성하게 동작될 수도 있다. 이것은 프로그래밍 및 독출 동작들이 편리하게 병렬로 취급될 수 있는 데이터량에 적응형으로 부합되게 하며,

메타페이지의 일부가 데이터로 프로그램되지 않은 채로 있게 되는 경우들을 감소시킨다.

[0061] 도 5에 도시된 바와 같이, 복수의 플레인들의 물리적 페이지들로 형성된 메타페이지는 이들 복수의 플레인들의 워드라인 행들을 따라 메모리 셀들을 내포한다. 동시에 한 워드라인 행 내 모든 셀들을 프로그램하기보다는, 보다 일반적으로 이들은 2이상의 인터리브된 그룹들로 번갈아 프로그램되며, 각 그룹은 한 페이지의 데이터(단일 블록에) 혹은 메타페이지의 데이터(복수 블록들에 걸쳐)를 저장한다. 한번에 교번적 메모리 셀들을 프로그래밍함으로써, 데이터 레지스터들 및 센스 증폭기들을 포함하는 한 단위의 주변회로들은 각 비트라인마다 제공될 필요는 없고 그보다는 이웃한 비트라인들간에 시-분할된다. 이것은 주변회로들에 대해 요구되는 기판 공간의 량을 절약하며 메모리 셀들이 행들을 따라 증가된 밀도로 패킹될 수 있게 한다. 그렇지 않다면, 주어진 메모리 시스템으로부터 가능한 병행도를 최대화하기 위해서 행을 따른 모든 셀을 동시에 프로그램하는 것이 바람직하다.

[0062] 도 3을 참조하면, 행을 따른 모든 다른 메모리 셀에 데이터의 동시적 프로그래밍은 도시된 단일 행 대신, NAND 스트링들의 적어도 일 단부를 따른 두 행들의 선택 트랜지스터들(도시생략)에 의해 가장 편리하게 달성된다. 이때 한 행의 선택 트랜지스터들은 한 제어 신호에 응하여 블록 내 모든 다른 스트링을 이들의 각각의 비트라인들에 연결하고, 다른 행의 선택 트랜지스터들은 개재된 모든 다른 스트링을 또 다른 제어신호에 응하여 그들의 각각의 비트라인들에 연결한다. 그러므로 두 페이지들의 데이터가 각 행의 메모리 셀들에 기입된다.

[0063] 각 논리 페이지 내 데이터량은 통상적으로 정수개의 하나 이상의 섹터들의 데이터이고, 각 섹터는 관례로 512바이트의 데이터를 내포한다. 도 6은 페이지 혹은 메타페이지의 논리 데이터 페이지의 두 섹터들(153, 155)의 데이터를 도시한 것이다. 통상적으로 각 섹터는 사용자 혹은 시스템 데이터가 저장되는 512 바이트의 부분(157)과, 부분(157) 내 데이터에 관계되거나 이 데이터가 저장된 물리적 페이지 혹은 블록에 관계된 오버헤드 데이터를 위한 또 다른 수의 바이트(159)를 내포한다. 오버헤드 데이터의 바이트 수는 통상적으로 16바이트이어서, 섹터들(153, 155)의 각각에 대해 총 528 바이트가 된다. 오버헤드 부분(159)은 프로그래밍동안 데이터 부분(157)으로부터 계산된 ECC, 이의 논리 어드레스, 블록이 소거되어 재-프로그램된 회수의 경험 카운트, 하나 이상의 플래그들, 동작전압 레벨들, 및/또는 등과, 이에 더하여 이러한 오버헤드 데이터(159)로부터 계산된 ECC를 내포할 수 있다. 대안적으로, 오버헤드 데이터(159), 혹은 이의 일부는 다른 블록들 내 다른 페이지들에 저장될 수도 있다.

[0064] 메모리들의 병행도가 증가함에 따라, 메타블록의 데이터 저장용량이 증가하고 데이터 페이지 및 메타페이지의 크기도 결과로서 증가한다. 그러면 데이터 페이지는 2섹터 이상의 데이터를 내포할 수 있다. 데이터 페이지 내 두 섹터들, 및 메타페이지 당 두 데이터 페이지들로, 메타페이지엔 4개의 섹터들이 있다. 이에 따라 각 메타페이지는 2048 바이트의 데이터를 저장한다. 이것은 고도의 병행도이고, 행들 내 메모리 셀들의 수가 증가됨에 따라 더욱 증가될 수 있다. 이러한 이유로, 플래시 메모리들의 폭은 페이지 및 메타페이지 내 데이터량을 증가시키기 위해 확장되고 있다.

[0065] 위에서 확인된 물리적으로 소형의 재프로그램가능 비휘발성 메모리 카드들 및 플래시 드라이브들은 512 메가바이트(MB), 1 기가바이트(GB), 2GB 및 4GB의 데이터 저장용량으로 시판되고 있고, 그 이상일 수도 있다. 도 7은 호스트와 이러한 대량 메모리 시스템간의 가장 일반적인 인터페이스를 도시한 것이다. 호스트는 애플리케이션 소프트웨어 혹은 호스트에 의해 실행되는 펌웨어 프로그램들에 의해 생성 혹은 사용되는 데이터 파일들을 처리한다. 워드 프로세싱 데이터 파일이 예이며, CAD(computer aided design) 소프트웨어의 도면 파일이 또 다른 예이고, PC들, 랩탑 컴퓨터들 등과 같은 일반적인 컴퓨터 호스트들에서 주로 볼 수 있다. pdf 포맷의 문서 또한 이러한 파일이다. 스틸 디지털 비디오 카메라는 메모리 카드에 저장되는 각 사진에 대한 데이터 파일을 생성한다. 셀룰라 전화는 전화번호와 같은, 내부 메모리 카드 상의 파일들로부터 데이터를 이용한다. PDA는 어드레스 파일, 달력 파일, 등과 같은 몇몇의 서로 다른 파일들을 저장하고 사용한다. 임의의 이러한 애플리케이션에서, 메모리 카드는 호스트를 동작시키는 소프트웨어를 내장할 수도 있다.

[0066] 호스트와 메모리 시스템간의 공통의 논리 인터페이스가 도 7에 도시되었다. 연속된 논리 어드레스 공간(161)은 메모리 시스템에 저장될 수 있는 모든 데이터에 대해 어드레스들을 제공할 만큼 충분히 크다. 호스트 어드레스 공간은 통상적으로 데이터 클러스터들의 증분들로 분할된다. 각각의 클러스터는 주어진 호스트 시스템에서 다수의 섹터들의 데이터 -4 내지 64 섹터들이 전형적이다- 를 내포하게 설계될 수 있다. 표준 섹터는 512 바이트의 데이터를 내포한다.

[0067] 3개의 파일들로서 파일 1, 파일 2, 파일 3이 도 7의 예에서 생성된 것으로 도시되었다. 호스트 시스템 상에서 실행되는 애플리케이션 프로그램은 순서로 된 한 세트의 데이터로서 각 파일을 생성하고 이를 고유의 이름 혹은 다른 기준에 의해 식별한다. 다른 파일들에 이미 할당되지 않은 충분히 가용한 논리 어드레스 공간은 호스트에

의해 파일 1에 할당된다. 파일 1은 인접한 범위의 가용 논리 어드레스들이 할당된 것으로 도시되었다. 일반적으로 어드레스들의 범위들은 호스트 동작 소프트에어를 위한 특정한 범위와 같은, 특정의 목적들을 위해 할당되는데, 이때 이들 어드레스들이 호스트가 데이터에 논리 어드레스들을 할당하고 있을 때 이용되지 않았을지라도 데이터를 저장하기 위해 이들 범위의 어드레스들을 피한다.

[0068] 파일 2가 호스트에 의해 나중에 생성될 때, 호스트는 유사하게 도 7에 도시된 바와 같이 논리 어드레스 공간(161) 내에 2개의 서로 다른 범위들의 인접하는 어드레스들을 할당한다. 파일은 인접한 논리 어드레스가 할당될 필요는 없고 그보다는 다른 파일들에 이미 할당된 어드레스 범위들 사이의 어드레스들의 단편들일 수 있다. 이때 이 예는 호스트에 의해 생성되는 또 다른 파일 3이 파일 1 및 파일 2에 전에 할당되지 않은 호스트 어드레스 공간의 다른 부분들 및 다른 데이터가 할당됨을 보이고 있다.

[0069] 호스트는 파일 할당 테이블(FAT)을 유지함으로써 메모리 논리 어드레스 공간을 주시하며, 여기서 호스트가 여러 호스트 파일들에 할당하는 논리 어드레스들이 유지된다. FAT 테이블은 통상적으로, 호스트 메모리에 뿐만 아니라 비휘발성 메모리에 저장되고, 새로운 파일들이 저장되고, 다른 파일들이 삭제되고, 파일들이 수정되는 등이 행해질 때 호스트에 의해 빈번히 업데이트된다. 예를 들면, 호스트 파일이 삭제될 때, 호스트는 삭제된 파일에 전에 할당된 논리 어드레스들이 다른 데이터 파일들에 이제 사용이 가능함을 나타내게 FAT 테이블을 업데이트함으로써, 이들 논리 어드레스들을 할당해제(deallocate)한다.

[0070] 호스트는 메모리 시스템 제어기가 파일들을 저장하기로 선택한 물리적 위치들에 관하여 관심을 갖지 않는다. 전형적인 호스트는 이의 논리 어드레스 공간과, 여러 파일들에 할당한 논리 어드레스들만을 안다. 한편, 메모리 시스템은 전형적인 호스트/카드 인터페이스를 통해서, 데이터가 기입된 논리 어드레스 공간의 부분들만을 알고 특정 호스트 파일들에 할당된 논리 어드레스들, 혹은 호스트 파일들의 수조차도 모른다. 메모리 시스템 제어기는 데이터의 저장 혹은 가져오기를 위해 호스트에 의해 제공되는 논리 어드레스들을 호스트 데이터가 저장되는 플래시 메모리 셀 어레이 내 고유 물리 어드레스들로 변환한다. 블록(163)은 이들 논리-물리 어드레스 변환들의 작업 테이블을 나타내고, 이는 메모리 시스템 제어기에 의해 유지된다.

[0071] 메모리 시스템 제어기는 시스템의 수행을 고 레벨로 유지하도록 메모리 어레이(165)의 블록들 및 메타블록들 내에 데이터 파일들을 저장하게 프로그램된다. 4개의 플레인들 혹은 서브-어레이들이 이 예시에서 사용된다. 데이터는 플레인들의 각각으로부터 블록으로 형성된 전체 메타블록에 걸쳐, 시스템이 허용하는 최대 병행도로 프로그램 및 독출되는 것이 바람직하다. 적어도 한 메타블록(167)은 늘, 메모리 제어기에 의해 사용되는 동작 펌웨어 및 데이터를 저장하기 위한 유보된 블록으로서 할당된다. 또 다른 메타블록(169), 혹은 복수의 메타블록들은 호스트 동작 소프트웨어, 호스트 FAT 테이블 등의 저장을 위해 할당될 수 있다. 물리적 저장공간 대부분은 데이터 파일들의 저장을 위해 남아있다. 그러나, 메모리 제어기는 수신된 데이터가 호스트의 여러 파일 객체들간 호스트에 의해 어떻게 할당되었는지를 모른다. 호스트와의 인터페이싱으로부터 메모리 제어기가 통상적으로 아는 모든 것은 특정의 논리 어드레스들에 호스트에 의해 기입되는 데이터가 제어기의 논리-물리 어드레스 테이블(163)에 의해 유지되는 대응 물리 어드레스들에 저장된다는 것이다.

[0072] 전형적인 메모리 시스템에서, 어드레스 공간(161) 내 데이터량을 저장하는데 필요한 것보다 몇 개의 여분의 블록들의 저장용량이 제공된다. 하나 이상의 이들 여분의 블록들은 메모리의 수명 동안 결함이 될 수도 있을 다른 블록들에 대한 대치를 위해 용장성 블록들로서 제공될 수도 있다. 개개의 메타블록들 내 내포된 블록들의 논리적 그룹화는, 원래 메타블록에 할당된 결함이 난 블록에 대한 용장성 블록의 대치를 포함한, 여러 이유들로 늘 변경될 수 있다. 메타블록(171)과 같은 하나 이상의 추가의 블록들은 통상적으로 소거된 블록 풀(pool) 내에 유지된다. 호스트가 메모리 시스템에 데이터를 기입할 때, 제어기는 소거된 블록 풀 내 메타블록 내에서 호스트-물리 어드레스들에 의해 할당된 논리 어드레스들을 변환한다. 이어서 논리 어드레스 공간(161) 내에 데이터를 저장하는데 사용되는 다른 메타블록들은 소거되고 후속 데이터 기입 동작동안 사용하기 위해 소거된 풀 블록들로서 지정된다.

[0073] 특정 호스트 논리 어드레스들에 저장된 데이터는 원래 저장된 데이터가 폐용될 때 새로운 데이터에 의해 자주 덮어씌어진다. 이에 응하여, 메모리 시스템 제어기는 새로운 데이터를 소거된 블록에 기입하고 이어서 이들 논리 어드레스들의 데이터가 저장되는 새로운 물리 블록을 확인하기 위해 이들 논리 어드레스들을 위해 논리-물리 어드레스 테이블을 변경한다. 이어서 이들 논리 어드레스들의 원 데이터를 내포하는 블록들이 소거되고 새로운 데이터 저장에 사용할 수 있게 된다. 이러한 소거는 빈번히, 기입 시작시 소거 블록 풀로부터 전에 소거된 블록들에 충분한 저장용량이 없다면 현 데이터 기입 동작이 완료될 수 있기 전에 행해져야 한다. 이것은 시스템 데이터 프로그래밍 속도에 악영향을 미칠 수 있다. 통상적으로 메모리 제어기는, 주어진 논리 어드레스에 데이터

는 호스트가 이들의 동일 논리 어드레스에 새로운 데이터를 기입할 때만 호스트에 의해 폐용하게 된 것을 안다. 그러므로 메모리의 많은 블록들은 이러한 무효한 데이터를 한동안 저장하고 있을 수 있다.

[0074] 블록들 및 메타블록들의 크기들은 집적회로 메모리 칩의 면적을 효율적으로 이용하기 위해서 증가하고 있다. 이에 따라 대부분의 개개의 데이터 기입들은 메타블록의 저장용량미만, 많은 경우들에 있어서 블록의 저장용량보다 훨씬 미만인 데이터량을 저장하게 된다. 메모리 시스템 제어기는 통상적으로 새로운 데이터를 소거된 풀 메타블록에 보내기 때문에, 이것은 얼마간의 메타블록들을 채워지지 않게 하는 결과를 가져올 수 있다. 새로운 데이터가 다른 메타블록에 저장된 어떤 데이터의 업데이트들이라면, 새로운 데이터 메타페이지들의 어드레스들에 인접한 논리 어드레스들을 갖는 그 다른 메타블록으로부터의 데이터의 나머지 유효한 메타페이지들은 새로운 메타블록에 논리적 어드레스 순서로 카피되는 것이 바람직하다. 이전의 메타블록은 다른 유효 데이터 메타페이지들을 보존할 수 있다. 이것은 시간에 따라 개개의 메타블록의 어떤 메타페이지들의 데이터가 폐용으로 되어 무효하게 되고, 다른 메타블록에 기입되는 동일 논리 어드레스를 가진 새로운 데이터에 의해 대치되는 결과로 된다.

[0075] 전체 논리 어드레스 공간(161)에 대해 데이터를 저장할 충분한 물리 메모리 공간을 유지하기 위해서, 이러한 데이터는 주기적으로 콤팩트 혹은 통합된다(가비지 수거(garbage collection)). 또한, 메타블록들 내 데이터의 섹터들을 가능한 한 실제적으로 이들의 논리 어드레스들과 동일 순서로 유지하는 것이 인접한 논리 어드레스들 내 데이터를 독출하는 것을 보다 효율적이게 하므로 바람직하다. 따라서 데이터 콤팩트 및 가비지 수거는 통상적으로 이러한 추가의 목적으로 수행된다. 부분적인 블록 데이터 업데이트들을 수신할 때 메모리를 관리하고 메타블록들의 사용의 어떤 면들이 미국특허 6,763,424에 기술되어 있다.

[0076] 통상적으로 데이터 콤팩트는 메타블록으로부터 모든 유효한 데이터 메타페이지들을 독출하고, 프로세스에서 무효한 데이터를 가진 메타페이지들을 무시하고, 이들 독출된 것들을 새로운 블록에 기입하는 것을 수반한다. 또한, 유효 데이터를 가진 메타페이지들은 이들에 저장된 데이터의 논리 어드레스 순서와 맞는 물리 어드레스 순서로 배열되는 것이 바람직하다. 새로운 메타블록에 점유되는 메타페이지들의 수는, 무효 데이터를 내포하는 메타페이지들이 새로운 메타블록에 카피되지 않기 때문에 이전 메타블록에 점유된 것들보다 적을 것이다. 이어서 이전 블록은 소거되고 새로운 데이터를 저장하는데 사용할 수 있게 된다. 이때 통합에 의해 얻어진 추가의 메타페이지들의 용량은 다른 데이터를 저장하는데 사용될 수 있다.

[0077] 가비지 수거동안에, 인접한 혹은 거의 인접한 논리 어드레스들을 가진 메타페이지들의 유효 데이터는 2이상의 메타블록들로부터 수거되어 통상 소거된 블록 풀 내의 메타블록인 또 다른 메타블록에 재기입된다. 모든 유효 데이터 메타페이지들이 원래의 2이상의 메타블록들로부터 카피될 때, 이들은 향후 사용을 위해 소거될 수 있다.

[0078] 데이터 통합 및 가비지 수거는 시간이 걸리고, 특히 데이터 통합 혹은 가비지 수거가 호스트로부터의 명령이 실행될 수 있기 전에 행해질 필요가 있을 경우, 메모리 시스템의 수행에 영향을 미칠 수 있다. 이러한 동작들은 정규적으로는 가능한 한 백그라운드(background)에서 행하기 위해 메모리 시스템 제어기에 의해 스케쥴링되나, 이들 동작들을 수행할 필요성은 이러한 동작이 완료될 때까지 제어기가 호스트에 비지(busy) 상태 신호를 주어야 하는 것을 야기할 수 있다. 호스트 명령의 실행이 지연될 수 있는 예는 호스트가 메모리에 기입하기를 원하는 모든 데이터를 저장할 소거된 블록 풀 내 사전에 소거된 충분한 메타블록들이 없고 먼저 하나 이상의 메타블록들의 유효 데이터를 소거하기 위해 데이터 통합 혹은 가비지 수거가 필요하고 이어 이들 메타블록들이 소거될 수 있는 경우이다. 그러므로, 이러한 중단을 최소화하기 위해서 메모리의 제어를 관리하는 것이 연구되었다. 많은 이러한 기술들은 다음의 미국특허출원: "Management of Non-Volatile Memory Systems Having Large Erase Blocks" 명칭으로 2003년 12월 30일에 출원된 출원번호 10/749,831; "Non- Volatile Memory and Method with Block Management System" 명칭으로 2003년 12월 30일에 출원된 출원번호 10/750,155; "Non-Volatile Memory and Method with Memory Planes Alignment" 명칭으로 2004년 8월 13일에 출원된 출원번호 10/917,888; 2004년 8월 13일에 출원된 출원번호 10/917,867; "Non- Volatile Memory and Method with Phased Program Failure Handling" 명칭으로 2004년 8월 13일에 출원된 출원번호 10/917,889; "Non-Volatile Memory and Method with Control Data Management." 명칭으로 2004년 8월 13일에 출원된 출원번호 10/917,725에 기술되어 있다.

[0079] 매우 큰 소거 블록들을 가진 메모리 어레이들의 동작을 효율적으로 제어하기 위한 한 해결할 문제는 주어진 기입동작 동안 저장되는 데이터 섹터들의 수를 메모리의 블록들의 용량에 맞추고 이들 블록들의 경계들에 정렬시키는 것이다. 한 방법은 호스트로부터 새로운 데이터를 저장하는데 사용되는 메타블록을, 전체 메타블록을 채우는 량 미만의 데이터량을 저장하는데 필요한 최대 수의 블록들 미만으로 구성하는 것이다. 적응형 메타블록들의 사용은 "Adaptive Metablocks" 명칭으로 2003년 12월 30일 출원된 미국특허출원 10/749,189에 기술되어 있다.

데이터의 블록들간 경계들과 메타블록들간 물리적 경계들을 맞추는 것은 2004년 5월 7일에 출원된 미국특허출원 10/841,118 및 "Data Run Programming" 명칭으로 2004년 12월 16일 출원된 미국특허출원 11/016,271에 기술되어 있다.

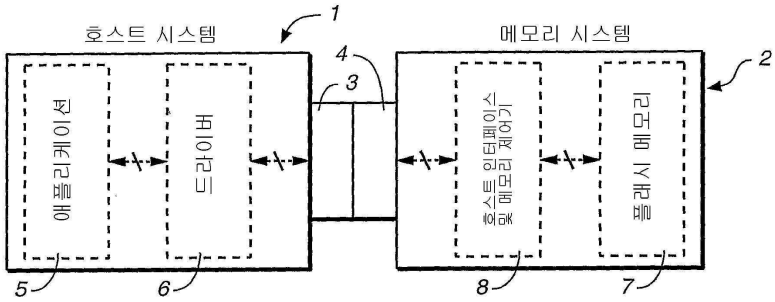
[0080] 메모리 제어기는 FAT 테이블로부터의 데이터를 사용할 수도 있고, 이 데이터는 메모리 시스템을 더 효율적으로 동작시키기 위해서 비휘발성 메모리에 호스트에 의해 저장된다. 한 이러한 사용은 논리 어드레스들을 할당해제함으로써 데이터가 호스트에 의해 폐용될 것으로 확인되었을 때를 아는 것이다. 이것을 앎으로써 메모리 제어기는 정상적으로는 호스트가 이들 논리 어드레스들에 새로운 데이터를 기입함으로써 그에 대해 알게 되기 전에 이러한 무효 데이터를 내포하는 블록들의 소거를 스케쥴링할 수 있게 된다. 이것은 "Method and Apparatus for Maintaining Data on Non- Volatile Memory Systems" 명칭으로 2004년 7월 21일 출원된 미국특허출원 10/897,049에 기술되어 있다. 이외 기술들은 주어진 기입동작이 단일 파일인지를 혹은 복수 파일들이라면 파일들간 경계들이 어디에 놓여 있는지를 추론하기 위해서 새로운 데이터를 메모리에 기입하는 호스트 패턴들을 모니터하는 것을 포함한다. "FAT Analysis for Optimized Sequential Cluster Management" 명칭의 2004년 12월 23일 출원된 미국특허출원 11/022,369은 이러한 유형의 기술들의 사용을 기술한다.

[0081] 메모리 시스템을 효율적으로 동작시키기 위해서, 제어기가 개개의 파일들의 데이터에 호스트에 의해 할당된 논리 어드레스들에 관하여 할 수 있는 한 많은 것을 아는 것이 바람직하다. 데이터 파일들은 파일 경계들을 모를 때 더 많은 수의 메타블록들간에 산재되기보다는 단일 메타블록 혹은 일 그룹의 메타블록들 내에 제어기에 의해 저장될 수 있다. 결과는 데이터 통합 및 가비지 수거 동작들의 수 및 복잡성이 감소된다는 것이다. 메모리 시스템의 수행은 결과적으로 향상된다. 그러나, 위에 기술된 바와 같이, 호스트/메모리 인터페이스가 논리 어드레스 공간(161)(도 7)을 포함할 때 메모리 제어기가 호스트 데이터 파일 구조에 관한 많은 것을 아는 것은 어렵다.

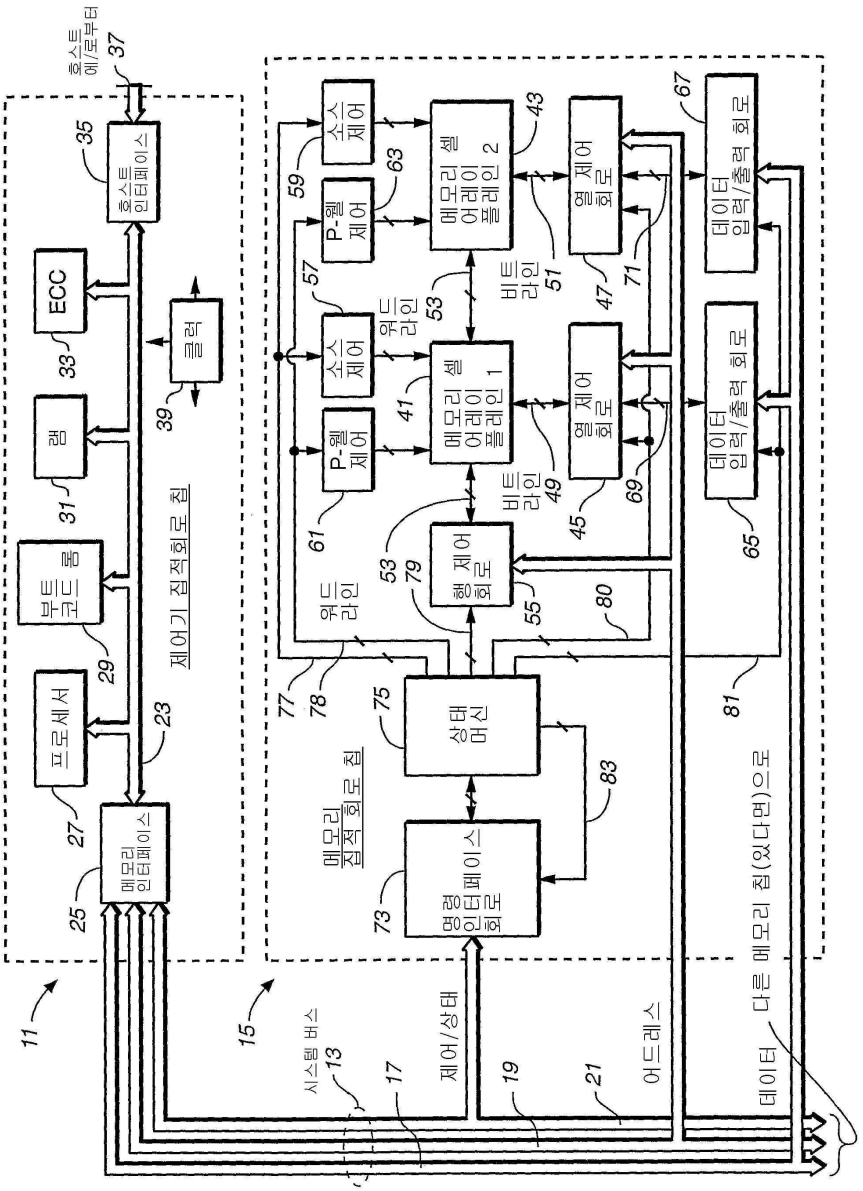
[0082] 도 8을 참조하면, 도 7에 이미 도시된 바와 같은 전형적인 논리 어드레스 호스트/메모리 인터페이스는 다르게 도시되었다. 호스트에 의해 생성된 데이터 파일들에는 호스트에 의해 논리 어드레스들이 할당된다. 그러면 메모리 시스템은 이들 논리 어드레스들을 보고 이들을 데이터가 실제로 저장되는 메모리 셀들의 블록들의 물리 어드레스들에 맵핑한다.

도면

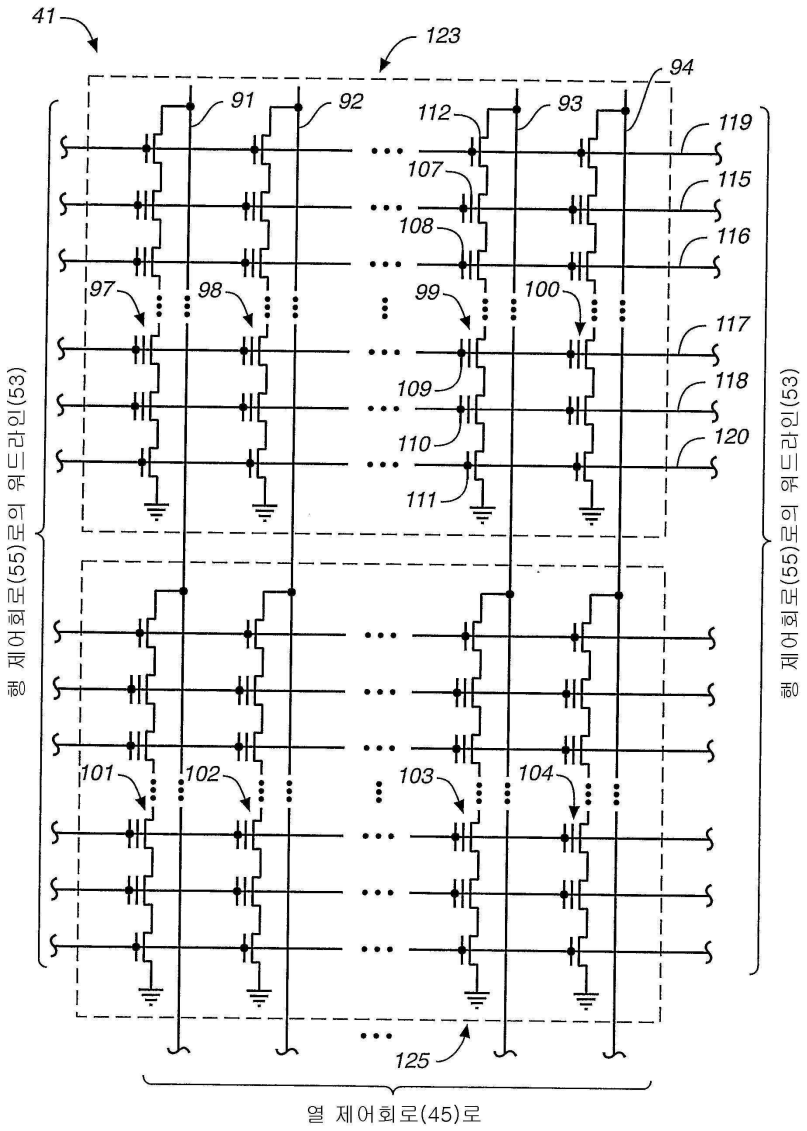
도면1



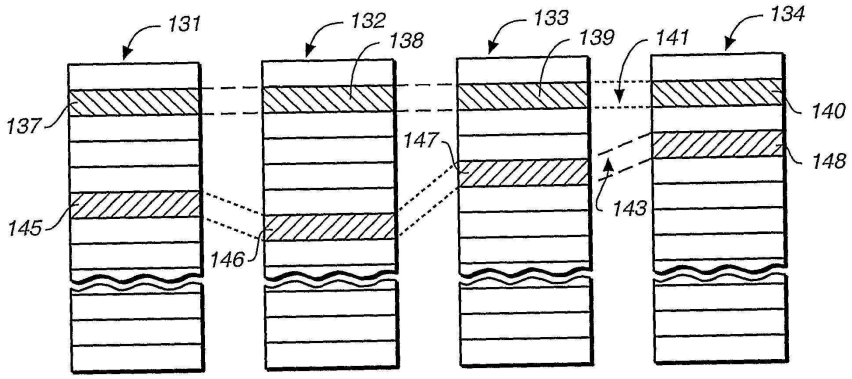
도면2


호스트 에/로부터
37
35
호스트 인터페이스
ECC
33
램
31
부트 코드 롬
29
프로세서
27
메모리 인터페이스
25
클럭
39
제어기 집적회로 칩
23
11
15
시스템 버스
13
17
제어/상태
19
21
어드레스
데이터
다른 메모리 칩(있다면)으로
메모리 집적 회로 칩
소스 제어
59
P-웰 제어
63
메모리 셀 어레이 플레인 2
43
열 제어 회로
47
데이터 입력/출력 회로
67
비트 라인
51
워드 라인
53
57
소스 제어
P-웰 제어
61
메모리 셀 어레이 플레인 1
41
열 제어 회로
45
데이터 입력/출력 회로
65
비트 라인
49
69
71
워드 라인
77
78
행 제어 회로
79
55
80
상태 머신
75
명령 인터페이스 회로
73
83
81

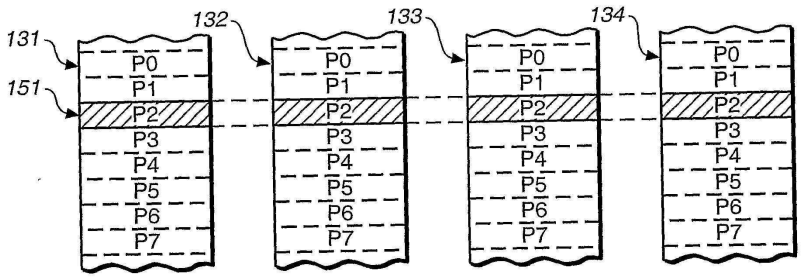
도면3


41
123
94
91
92
112
93
119
107
115
108
116
97
98
99
100
117
109
118
110
120
111
행 제어회로(55)로의 워드라인(53)
행 제어회로(55)로의 워드라인(53)
101
102
103
104
125
열 제어회로(45)로

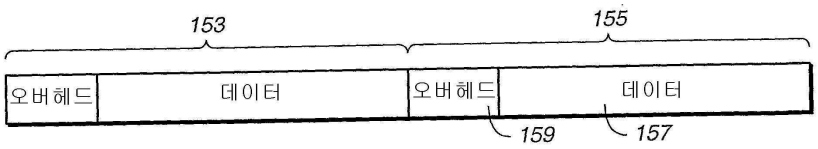
도면4


131
132
138
133
139
141
134
137
140
147
148
145
143
146

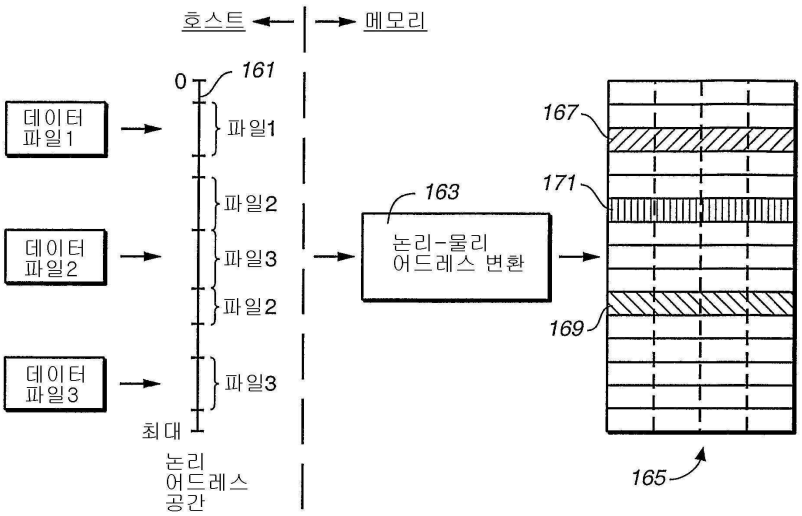
도면5


131
132
133
134
151
P0
P1
P2
P3
P4
P5
P6
P7

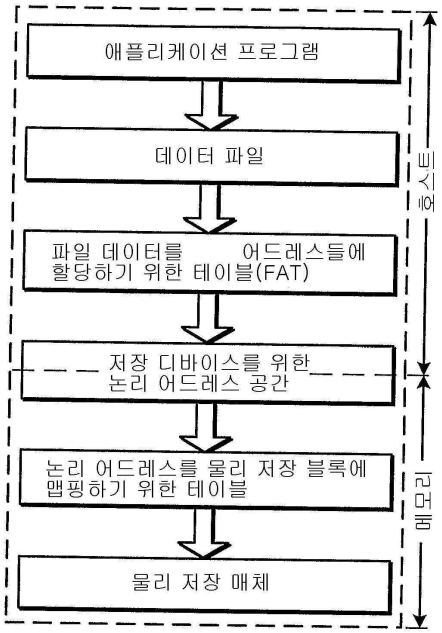
도면6


153
155
오버헤드
데이터
오버헤드
데이터
159
157

도면7

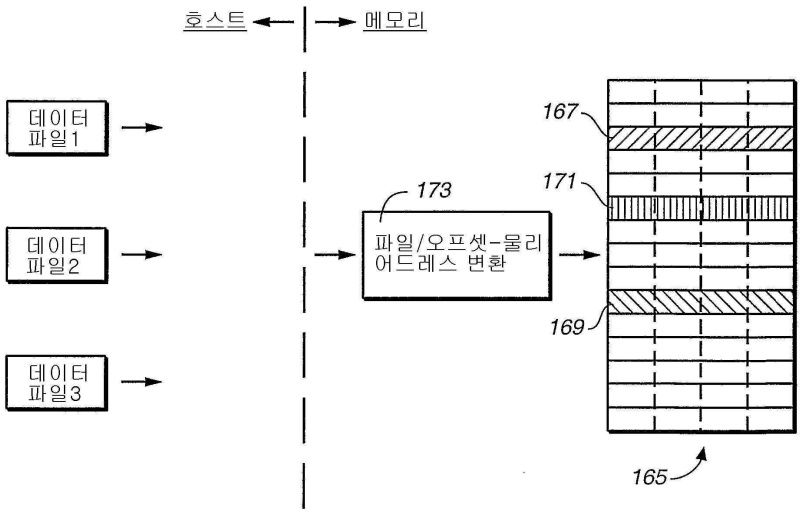

호스트
메모리
0
161
데이터 파일1
데이터 파일2
데이터 파일3
파일1
파일2
파일3
파일2
파일3
최대
논리 어드레스 공간
163
논리-물리 어드레스 변환
167
171
169
165

도면8

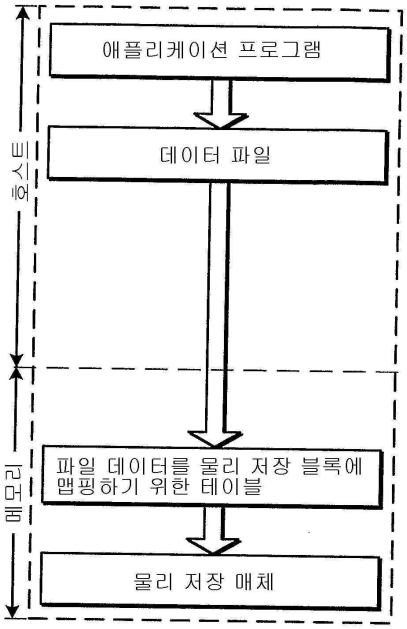


종래 기술 시스템

도면9

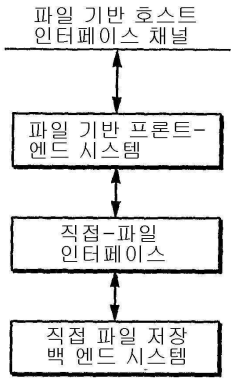


도면10



직접 파일 저장 시스템

도면11



도면12A

데이터 명령들

명령	파라미터	설명
Write	< fileID >	명시된 파일의 끝에 데이터를 첨부
Insert	< fileID > < offset >	시작부터 <offset>에, 명시된 파일 내에 데이터를 삽입
Update	< fileID > < offset >	시작부터 <offset>에서, 명시된 파일에 데이터를 덮어쓰기한다.
Read	< fileID > < offset >	시작부터 <offset>에서, 명시된 파일로부터 데이터를 독출
Remove	< fileID > < offset 1 > < offset 2 >	시작부터 <offset1>와, <offset2> 사이에 명시된 파일 내 데이터를 제거

도면12B

파일 명령들

명령	파라미터	설명
Open	< fileID >	명시된 파일에 기입하기 위해 자원들을 사용할 수 있게 되도록 요청
Close	< fileID >	자원들은 명시된 파일로부터 즉시 제거될 수 있고, 파일은 가비지 수거에 스케쥴링될 수 있다.
Close_after	< fileID > < length >	자원들은 호스트로부터 파일 데이터의 <length>의 전송 후에, 명시된 파일로 부터 제거될 수 있고, 이때 파일은 수거에 스케쥴링될 수 있다.
Delete	< fileID >	명시된 파일에 대한 테이블 엔트리들 및 데이터가 삭제될 것임을 지시한다. 파일 데이터는 소거될 수 있다.
Erase	< fileID >	명시된 파일에 대한 테이블 엔트리들 및 데이터가 삭제되고 파일 데이터가 즉시로 소거될 것임을 지시한다.

도면12C

디렉토리 명령들

명령	파라미터	설명
List	< directoryID >	디바이스가 디렉토리 및 파일정보를 보고할 것을 호스트로부터 요청
Current	< directoryID >	새로운 파일이 열리거나, 새로운 디렉토리가 생성되거나, 디렉토리가 삭제되는 현 디렉토리에 대해 호스트에 의한 정의.
Create	< directoryID >	새로운 디렉토리가 생성될 것을 호스트로부터 요청
Delete	< directoryID >	디렉토리가 삭제될 것을 호스트로부터 요청. 서브-디렉토리들 및 디렉토리 내 파일들도 삭제된다.
Erase	< directoryID >	디렉토리가 삭제될 것을 호스트로부터 요청. 서브-디렉토리들 및 디렉토리 내 파일들도 삭제된다. 명시된 파일들에 대한 데이터는 즉시 소거될 것이다.

도면12D

상태 명령들

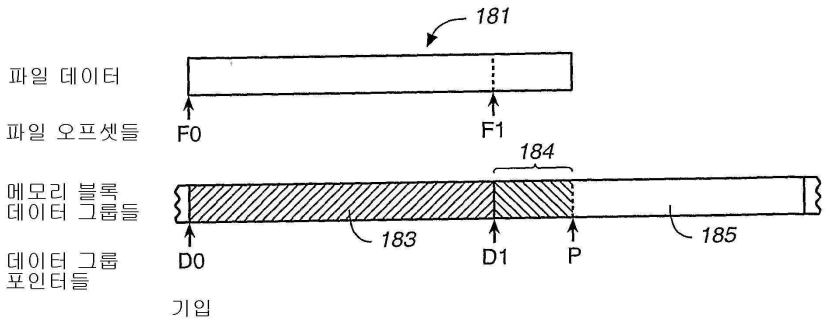
명령	파라미터	설명
Idle		호스트 인터페이스는 아이들 상태에 들어가고, 이 상태 내에서 디바이스는 내부 동작들을 수행할 수 있다. 아이들 상태는 디바이스가 내부 동작으로 비지인지 아니든 간에, 호스트에 의해 또 다른 명령의 전송에 의해 종료될 수 있다. 이러한 다른 명령의 수신시, 디바이스 내 진행중의 어떤 내부동작이든 명시된 시간 내에 종료되어야 한다.
Standby		호스트 인터페이스는 대기상태에 들어가고, 이 상태 내에서 디바이스는 내부 동작들을 수행하지 않을 수 있다. 대기상태는 호스트에 의한 또 다른 명령의 전송에 의해 종료될 수 있다.
Shut-down		호스트 인터페이스는 차단될 것이며 파워는 디바이스가 다음에 비지 상태로 있지 않을 때 카드로부터 제거될 것이다.

도면12E

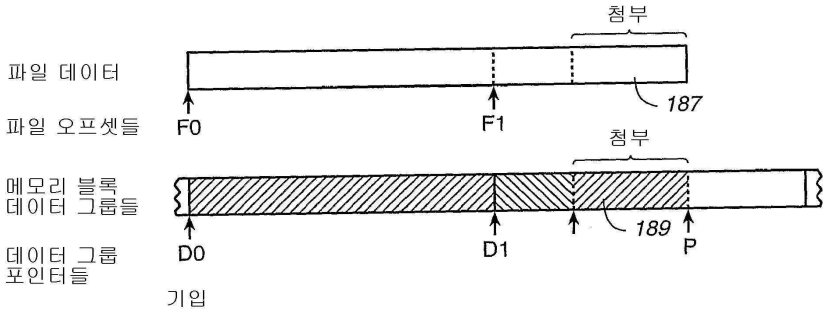
메모리 시스템 명령

명령	파라미터	상태
Size		디바이스가 새로운 파일 데이터에 대해 가용 용량을 보고할 것을 호스트로부터 요청
Status		디바이스가 현 상태를 보고할 것으로 호스트로부터 요청

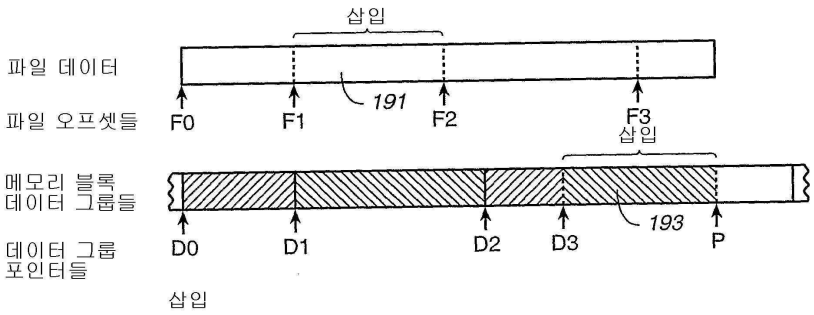
도면13A



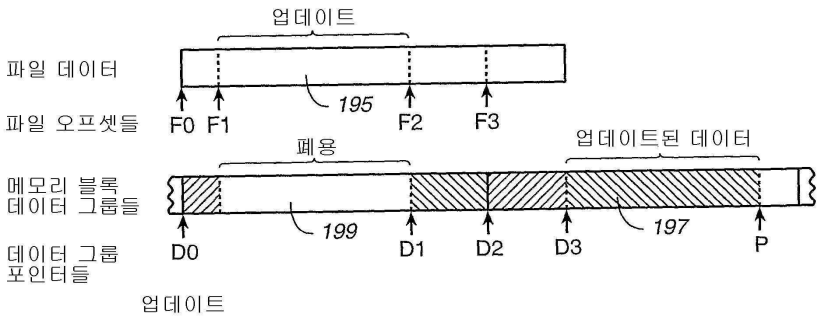
도면13B



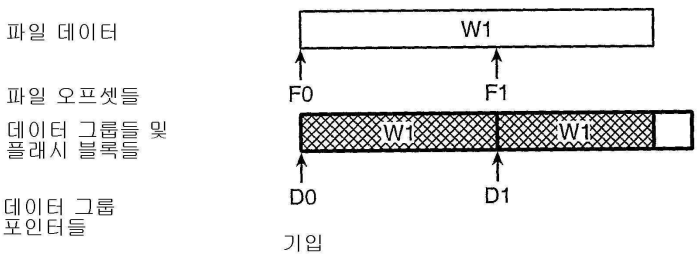
도면13C



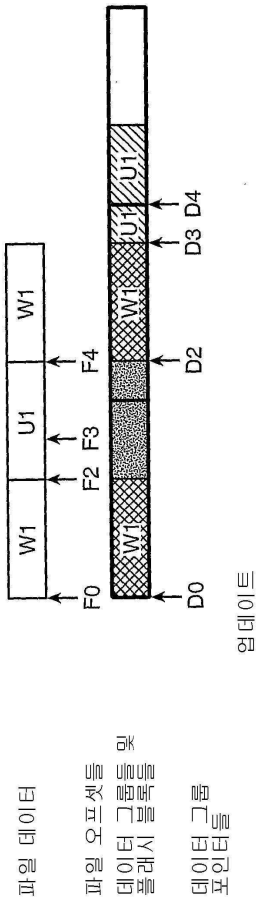
도면13D



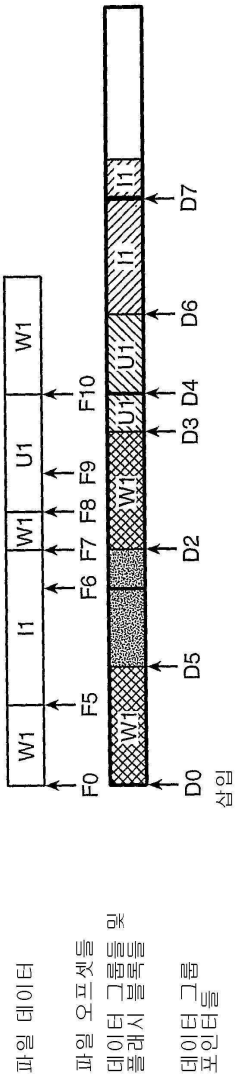
도면14A



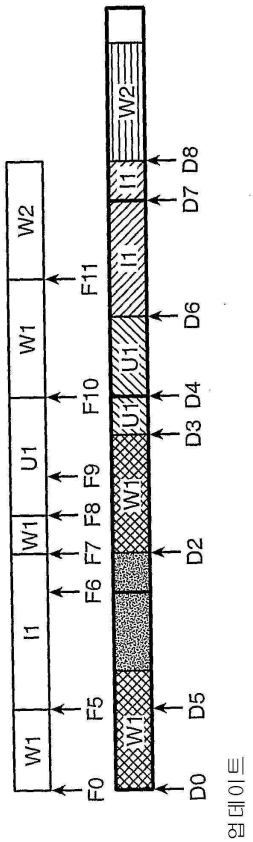
도면14B


W1
U1
W1
F0
F2
F3
F4
W1
W1
U1
U1
D0
D2
D3
D4
업데이트
파일 데이터
파일 오프셋들
데이터 그룹들을 및
플래시 블록들
데이터 그룹
포인터들

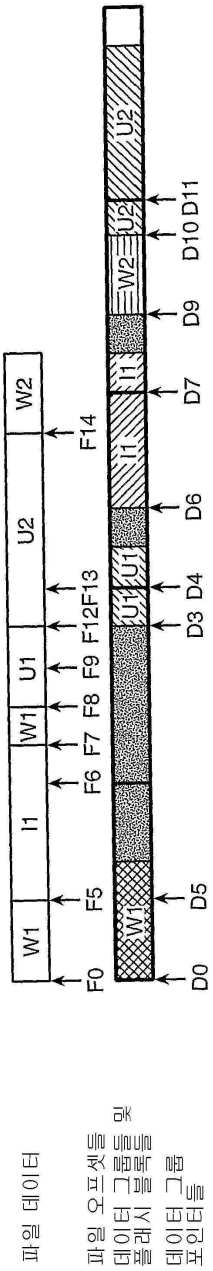
도면14C


파일 데이터
파일 오프셋들
데이터 그룹들 및
플래시 블록들
데이터 그룹
포인터들
W1
I1
W1
U1
W1
U1
I1
I1
F0
F5
F6
F7
F8
F9
F10
D0
D5
D2
D3
D4
D6
D7
삽입

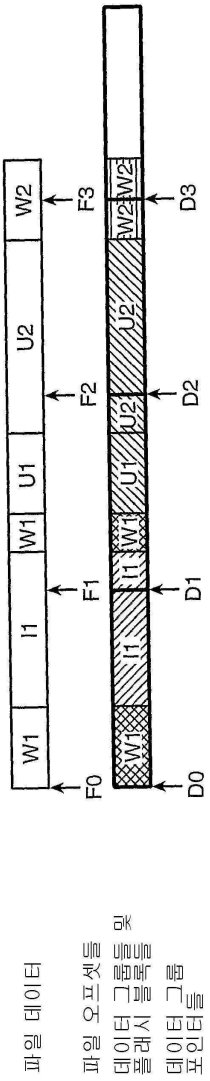
도면14D


W1
I1
W1
U1
W1
W2
F0
F5
F6
F7
F8
F9
F10
F11
W1
W1
U1
U1
I1
I1
W2
D0
D5
D2
D3
D4
D6
D7
D8
업데이트
파일 데이터
파일 오프셋들
데이터 그룹들 및
블래시 블록들
데이터 그룹
포인터들

도면14E


파일 데이터
파일 오프셋들 및
데이터 그룹들을
룰레시 블록들
데이터 그룹
포인터들
F0
F5
F6
F7
F8
F9
F12
F13
F14
D0
D3
D4
D5
D6
D7
D9
D10
D11
W1
I1
W1
U1
U2
W2
W1
U1
U1
I1
I1
W2
U2
U2

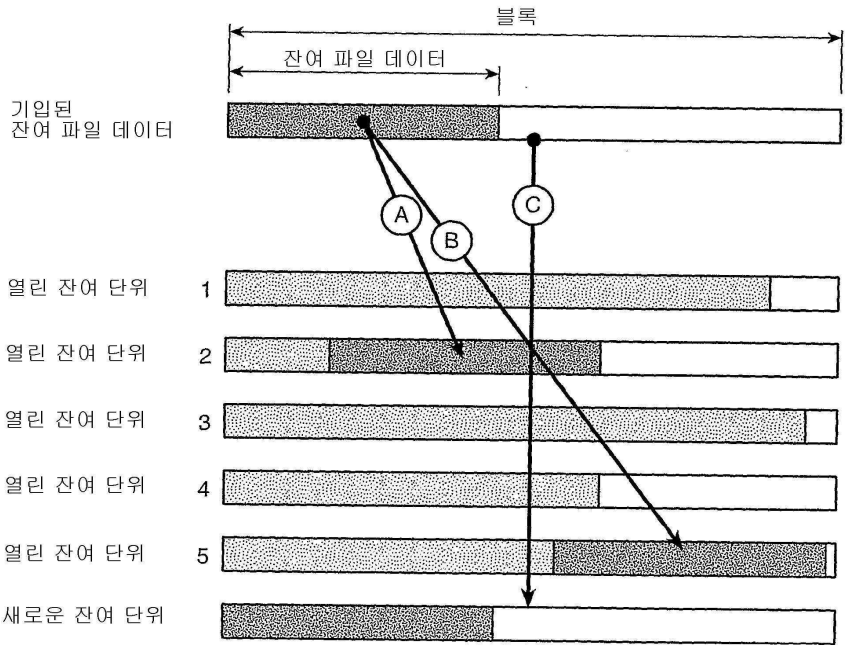
도면15


W2
U2
U1
W1
I1
W1
F3
F2
F1
F0
W2
U2
U1
W1
I1
W1
D3
D2
D1
D0
파일 데이터
파일 오프셋들
데이터 그룹들 및
플래시 블록들
데이터 그룹
포인터들

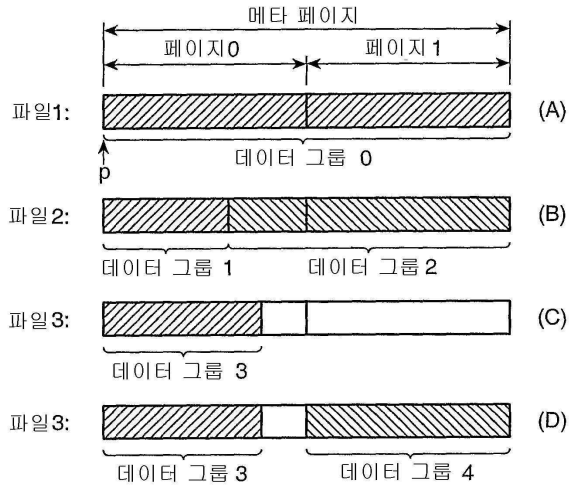
도면16


파일명1
(Fx1, Dy2)
파일명2
(Fx2, Dy2)
파일명3
(Fx3, Dy3)
미사용된
소거된
공간

도면17


블록
잔여 파일 데이터
기입된
잔여 파일 데이터
A
B
C
열린 잔여 단위 1
열린 잔여 단위 2
열린 잔여 단위 3
열린 잔여 단위 4
열린 잔여 단위 5
새로운 잔여 단위

도면18

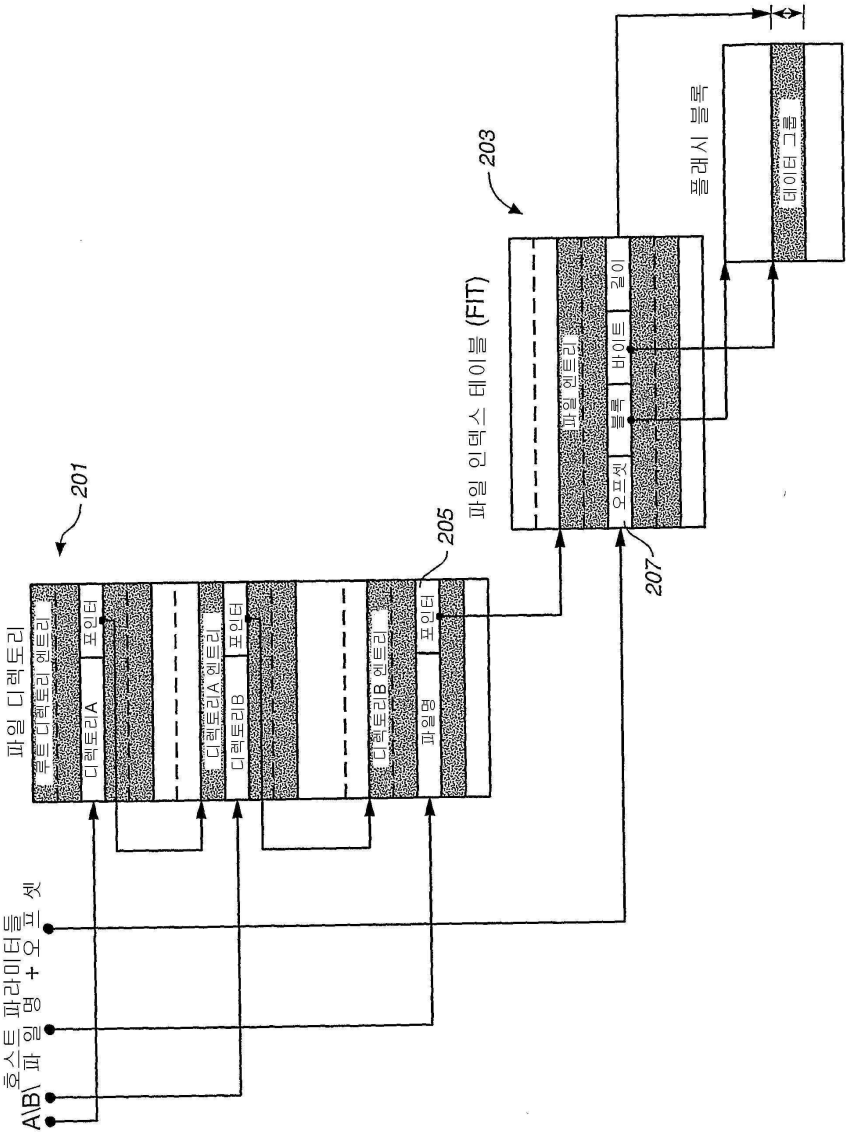

메타 페이지
페이지0
페이지1
파일1:
(A)
p
데이터 그룹 0
파일2:
(B)
데이터 그룹 1
데이터 그룹2
파일3:
(C)
데이터 그룹 3
파일3:
(D)
데이터 그룹 3
데이터 그룹 4

도면19

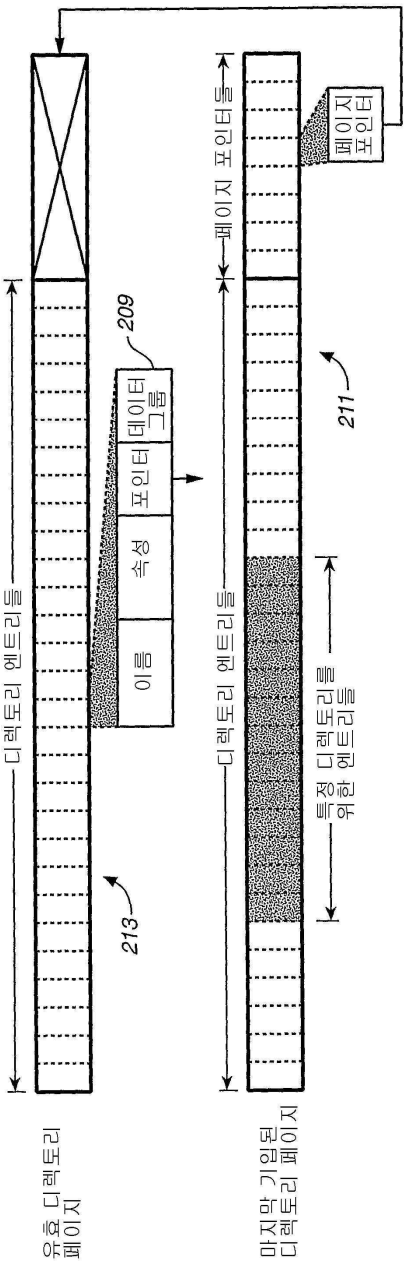
파일 인덱스 테이블 (FIT)

	오프셋	블록	바이트	길이
시간0 (도14A)	F0	001	D0	——
	F1	002	D1	——
시간2 (도14C)	F0	001	D0	——
	F5	003	D6	——
	F6	004	D7	——
	F7	001	D5	——
	F8	002	D3	——
	F9	003	D4	——
	F10	002	D2	——
시간4 (도14E)	F0	001	D0	——
	F5	003	D6	——
	F6	004	D7	——
	F7	001	D5	——
	F8	002	D3	——
	F9	003	D4	——
	F12	004	D10	——
	F13	005	D11	——
	F14	004	D9	——
시간5 (도15)	F0	001	D0	——
	F1	002	D1	——
	F2	003	D2	——
	F3	004	D3	——

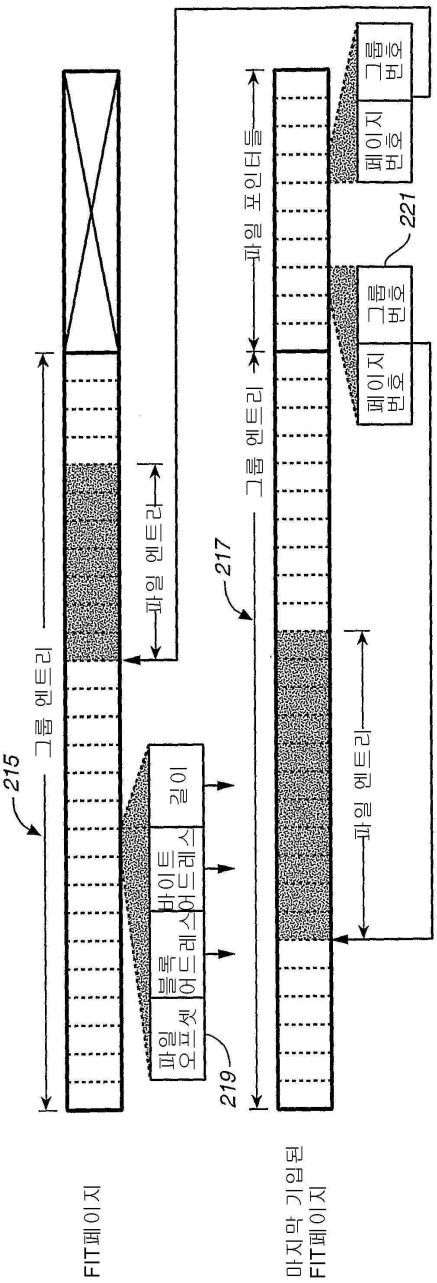
도면20


201
파일 디렉토리
루트 디렉토리 엔트리
디렉토리A
포인터
디렉토리A 엔트리
디렉토리B
포인터
디렉토리B 엔트리
파일용
포인터
205
203
파일 인덱스 테이블 (FIT)
파일 엔트리
오프셋
클러스터
바이트
길이
207
플래시 블록
데이터 그룹
호스트 파라미터들
A\B\ 파일명 + 오프셋

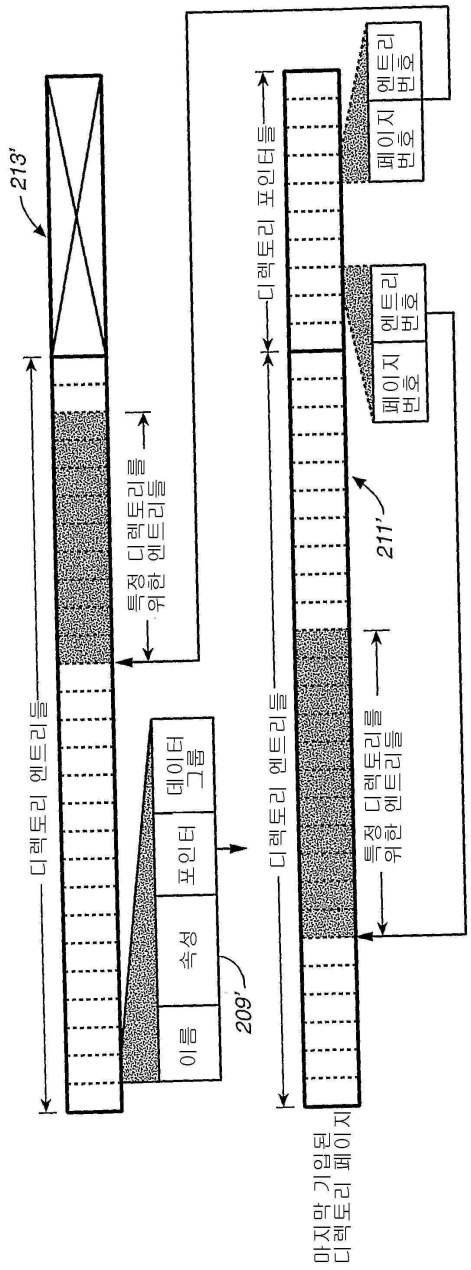
도면21


디렉토리 엔트리들
209
데이터 그룹
포인터
속성
이름
213
유효 디렉토리 페이지
페이지 포인터들
디렉토리 엔트리들
211
페이지 포인터
특정 디렉토리를 위한 엔트리들
마지막 기입된 디렉토리 페이지

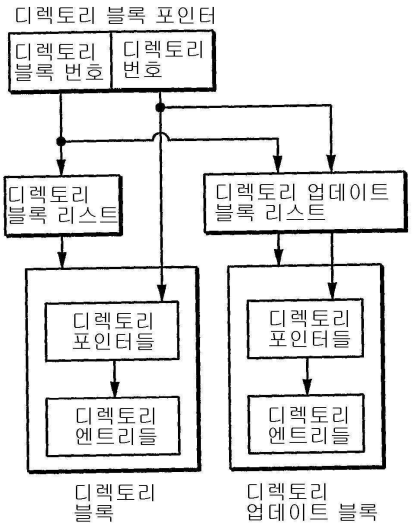
도면22


215
그룹 엔트리
파일 엔트리
파일 오프셋
블록 어드레스
바이트 어드레스
길이
219
217
그룹 엔트리
파일 포인터들
221
그룹 번호
페이지 번호
그룹 번호
페이지 번호
파일 엔트리
FIT 페이지
마지막 기입된 FIT 페이지

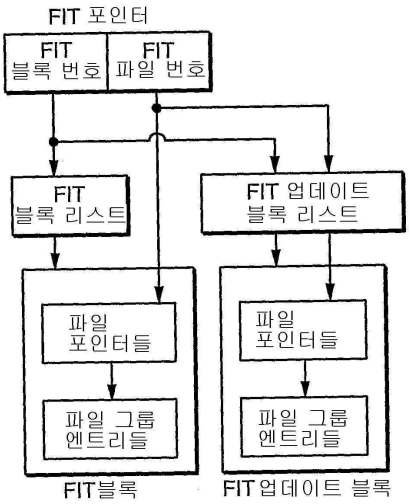
도면23


213'
디렉토리 엔트리들
특정 디렉토리를 위한 엔트리들
데이터 그룹
포인터
속성
이름
209'
디렉토리 엔트리들
디렉토리 포인터들
211'
페이지 번호
엔트리 번호
페이지 번호
엔트리 번호
특정 디렉토리를 위한 엔트리들
마지막 기입된 디렉토리 페이지

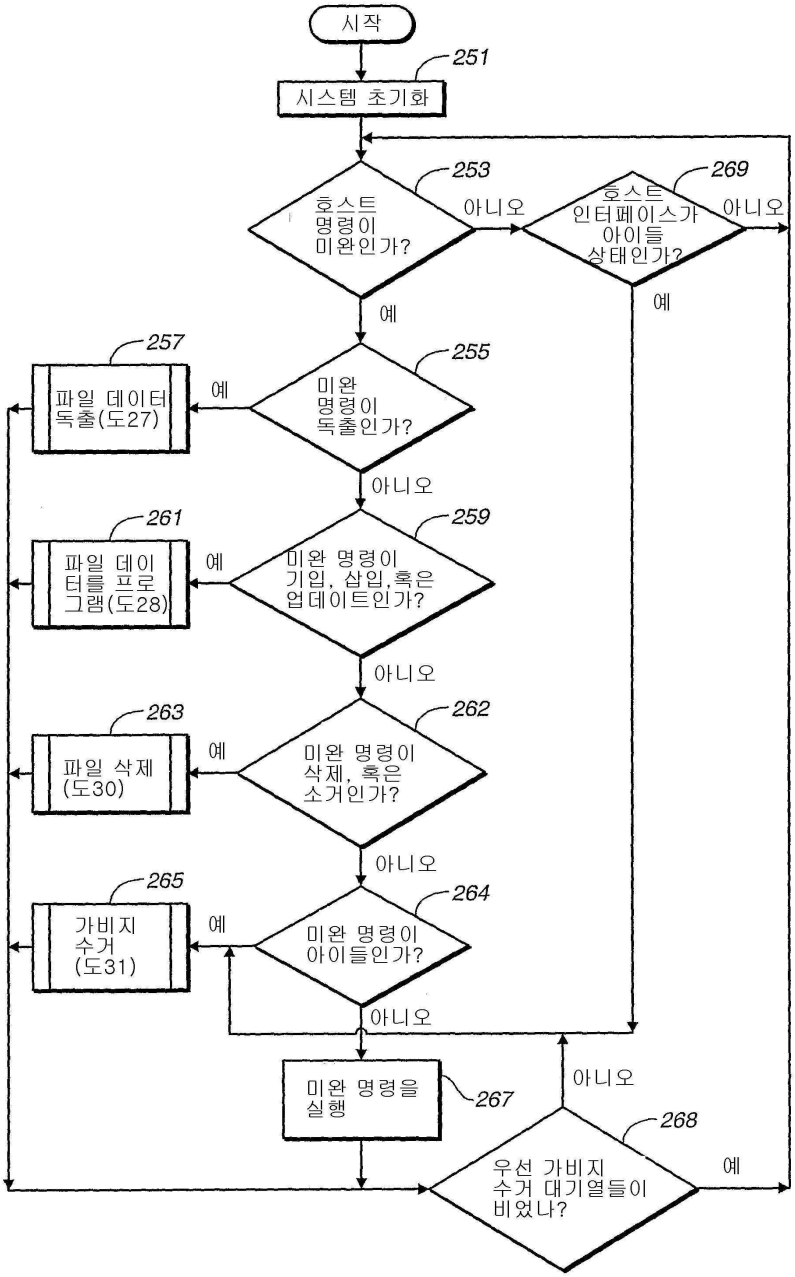
도면24


디렉토리 블록 포인터
디렉토리 블록 번호
디렉토리 번호
디렉토리 블록 리스트
디렉토리 업데이트 블록 리스트
디렉토리 포인터들
디렉토리 엔트리들
디렉토리 포인터들
디렉토리 엔트리들
디렉토리 블록
디렉토리 업데이트 블록

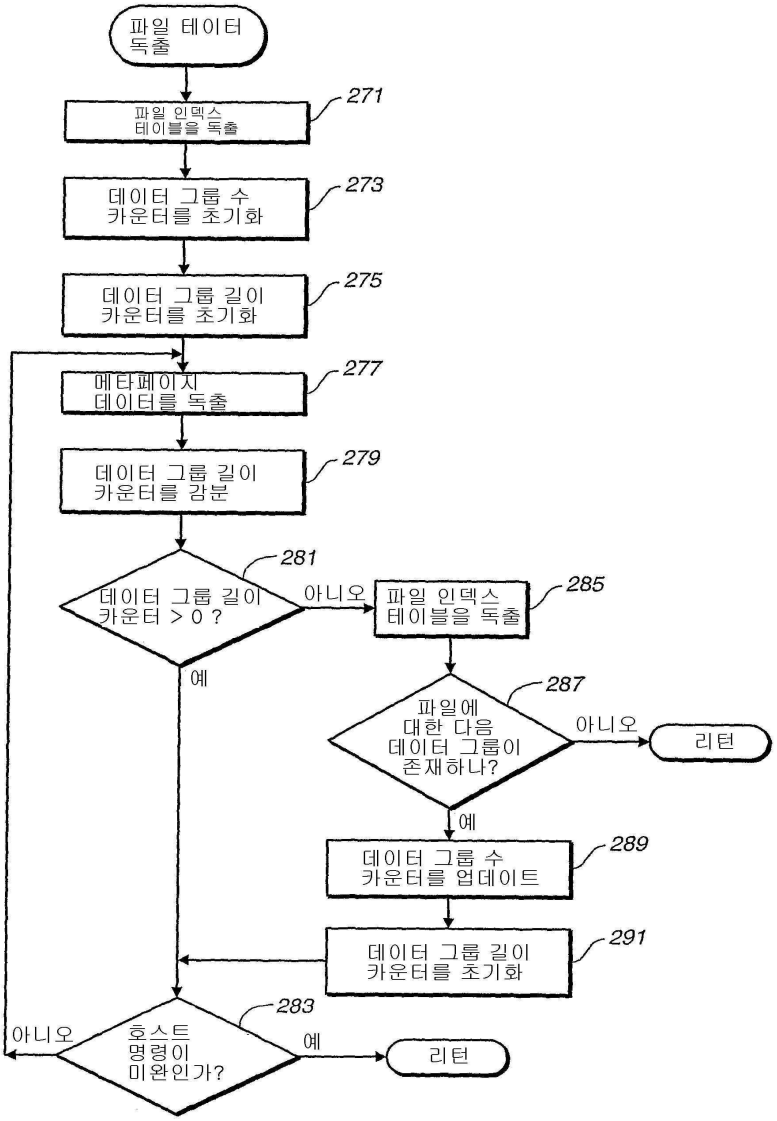
도면25


FIT 포인터
FIT 블록 번호
FIT 파일 번호
FIT 블록 리스트
FIT 업데이트 블록 리스트
파일 포인터들
파일 그룹 엔트리들
파일 포인터들
파일 그룹 엔트리들
FIT블록
FIT업데이트 블록

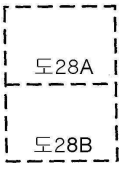
도면26


시작
251
시스템 초기화
253
호스트 명령이 미완인가?
아니오
269
호스트 인터페이스가 아이들 상태인가?
아니오
예
예
255
미완 명령이 독출인가?
257
파일 데이터 독출(도27)
예
아니오
259
미완 명령이 기입, 삽입, 혹은 업데이트인가?
261
파일 데이터를 프로그램(도28)
예
아니오
262
미완 명령이 삭제, 혹은 소거인가?
263
파일 삭제 (도30)
예
아니오
264
미완 명령이 아이들인가?
265
가비지 수거 (도31)
예
아니오
267
미완 명령을 실행
아니오
268
우선 가비지 수거 대기열들이 비었나?
예

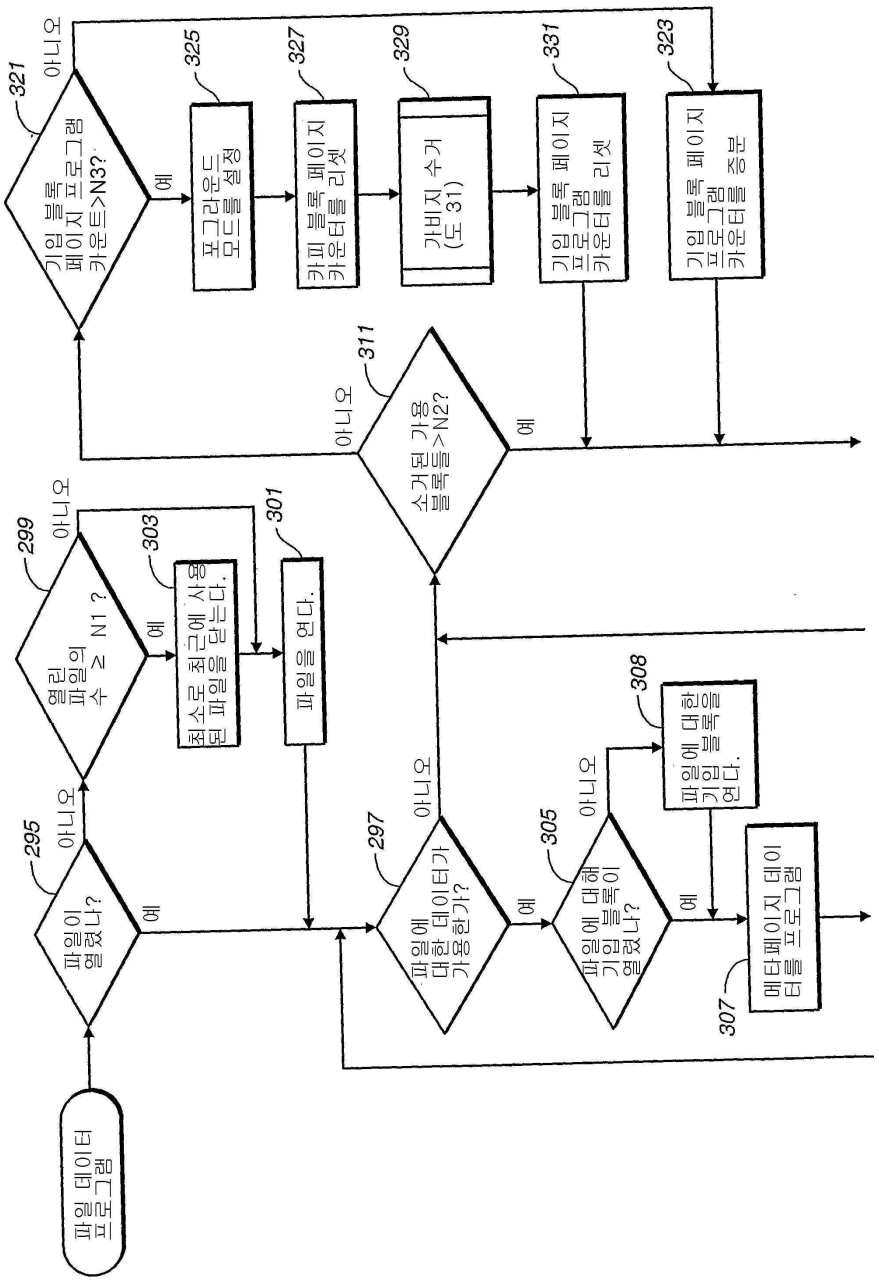
도면27


파일 데이터 독출
271
파일 인덱스 테이블을 독출
273
데이터 그룹 수 카운터를 초기화
275
데이터 그룹 길이 카운터를 초기화
277
메타페이지 데이터를 독출
279
데이터 그룹 길이 카운터를 감분
281
데이터 그룹 길이 카운터 > 0 ?
아니오
285
파일 인덱스 테이블을 독출
예
287
파일에 대한 다음 데이터 그룹이 존재하나?
아니오
리턴
예
289
데이터 그룹 수 카운터를 업데이트
291
데이터 그룹 길이 카운터를 초기화
283
호스트 명령이 미완인가?
아니오
예
리턴

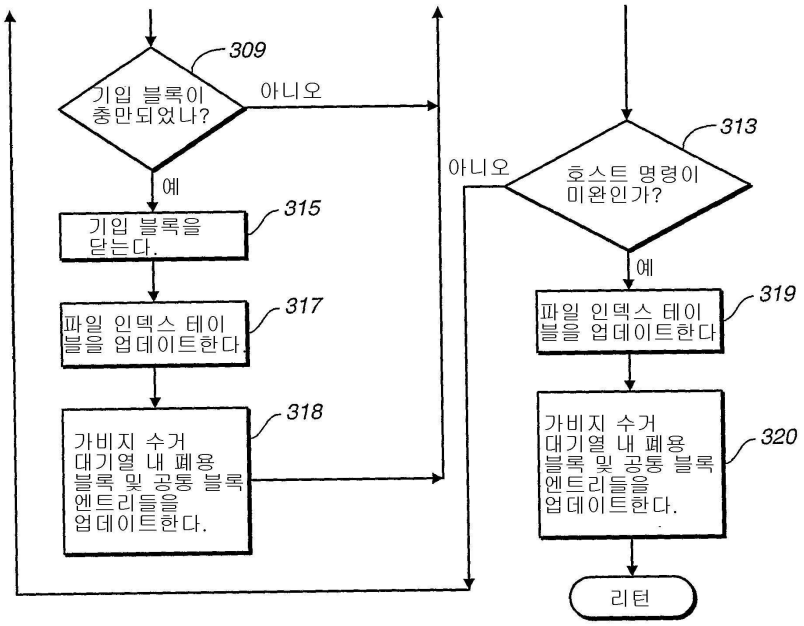
도면28


도28A
도28B

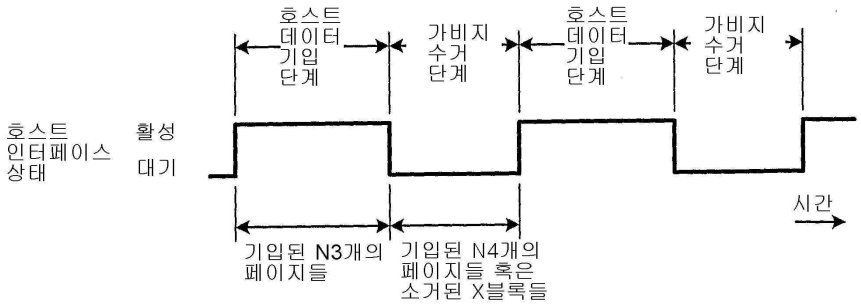
도면28A


파일 데이터 프로그램
295
파일이 열렸나?
아니오
예
299
열린 파일의 수 ≥ N1 ?
아니오
예
303
최소로 최근에 사용된 파일을 닫는다.
301
파일을 연다.
297
파일에 대한 데이터가 가용한가?
아니오
예
305
파일에 대해 기입블록이 열렸나?
아니오
예
308
파일에 대한 기입블록을 연다.
307
메타페이지 데이터 테이블 프로그램
311
소거된 가용 블록들>N2?
아니오
예
321
기입 블록 프로그램 페이지 카운트>N3?
아니오
예
325
포그라운드 모드를 설정
327
카피 블록 페이지 카운터를 리셋
329
가비지 수거 (도 31)
331
기입 블록 페이지 프로그램 카운터를 리셋
323
기입 블록 페이지 프로그램 카운터를 증분

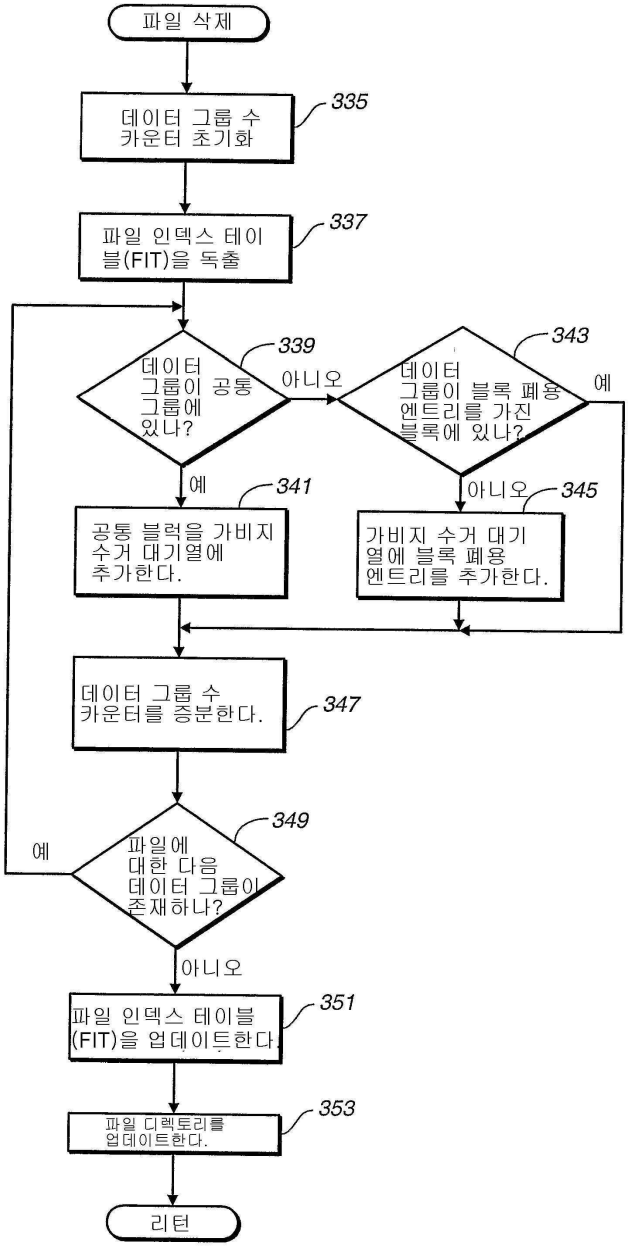
도면28B


309
기입 블록이 충만되었나?
아니오
예
315
기입 블록을 닫는다.
317
파일 인덱스 테이블을 업데이트한다.
318
가비지 수거 대기열 내 폐용 블록 및 공통 블록 엔트리들을 업데이트한다.
313
호스트 명령이 미완인가?
아니오
예
319
파일 인덱스 테이블을 업데이트한다
320
가비지 수거 대기열 내 폐용 블록 및 공통 블록 엔트리들을 업데이트한다.
리턴

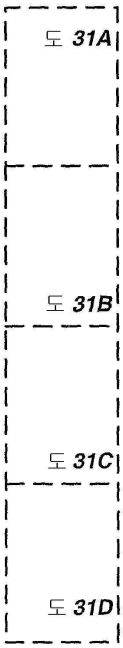
도면29


호스트 데이터 기입 단계
가비지 수거 단계
호스트 데이터 기입 단계
가비지 수거 단계
호스트 인터페이스 상태
활성
대기
시간
기입된 N3개의 페이지들
기입된 N4개의 페이지들 혹은 소거된 X블록들

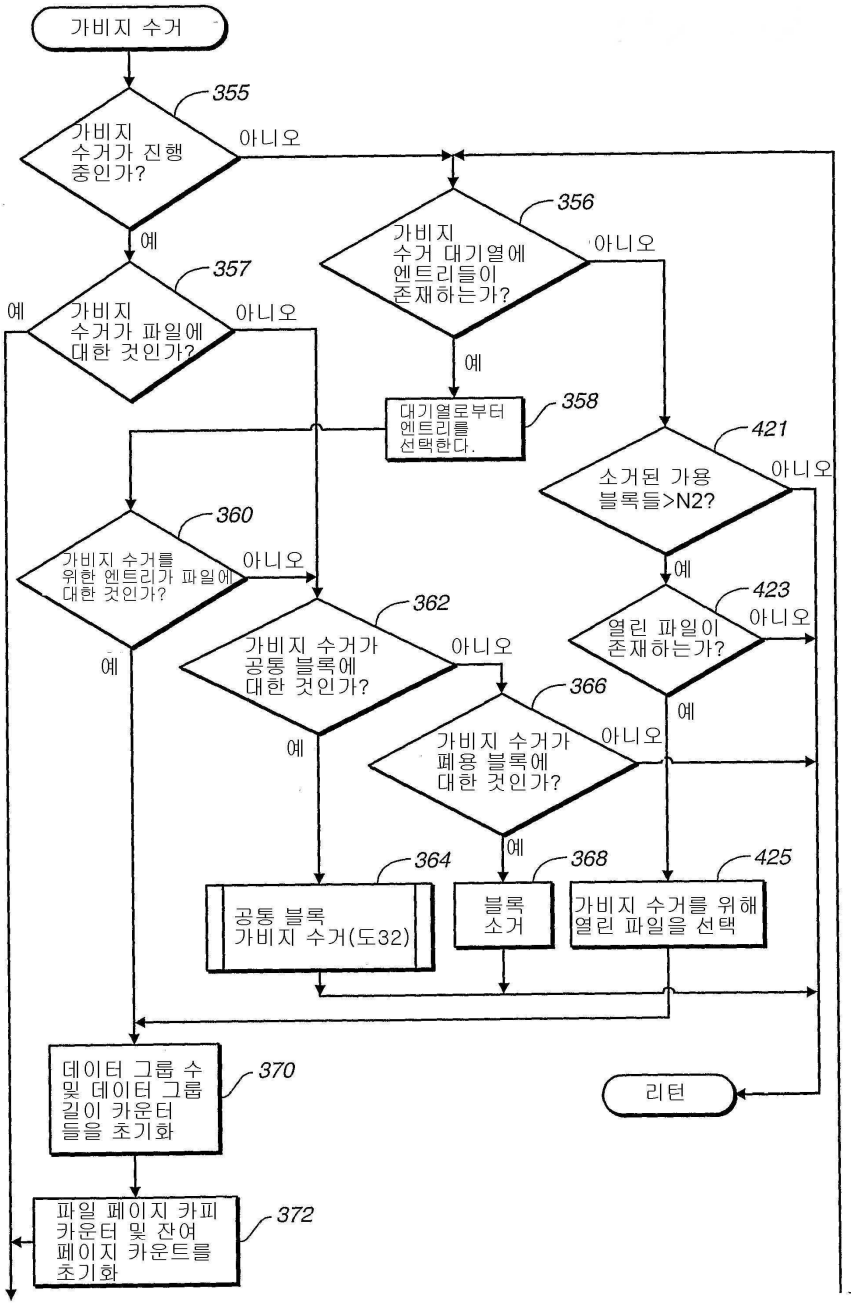
도면30


파일 삭제
데이터 그룹 수 카운터 초기화
335
파일 인덱스 테이블(FIT)을 독출
337
데이터 그룹이 공통 그룹에 있나?
339
아니오
데이터 그룹이 블록 폐용 엔트리를 가진 블록에 있나?
343
예
예
341
공통 블럭을 가비지 수거 대기열에 추가한다.
아니오
345
가비지 수거 대기열에 블록 폐용 엔트리를 추가한다.
데이터 그룹 수 카운터를 증분한다.
347
예
349
파일에 대한 다음 데이터 그룹이 존재하나?
아니오
파일 인덱스 테이블(FIT)을 업데이트한다.
351
파일 디렉토리를 업데이트한다.
353
리턴

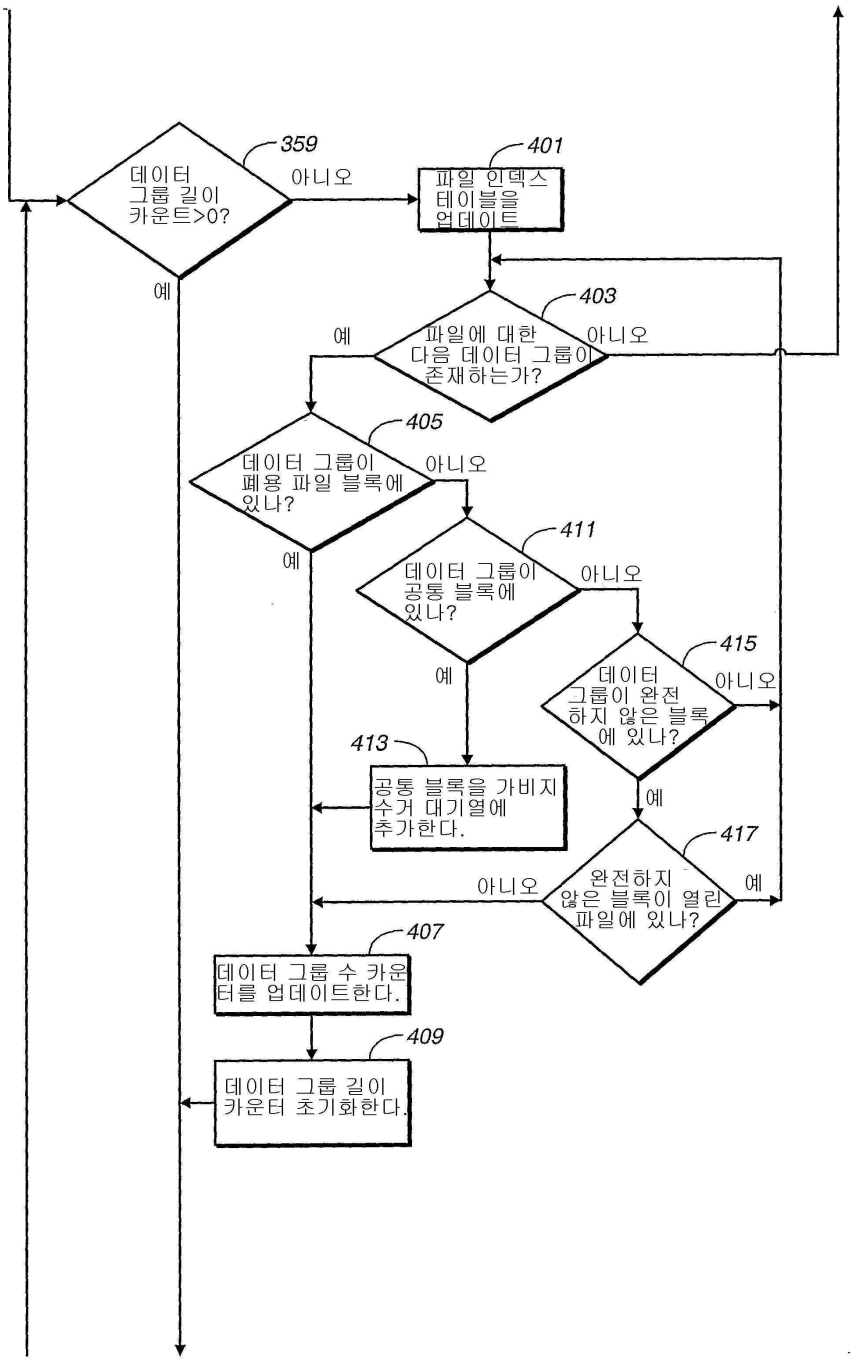
도면31



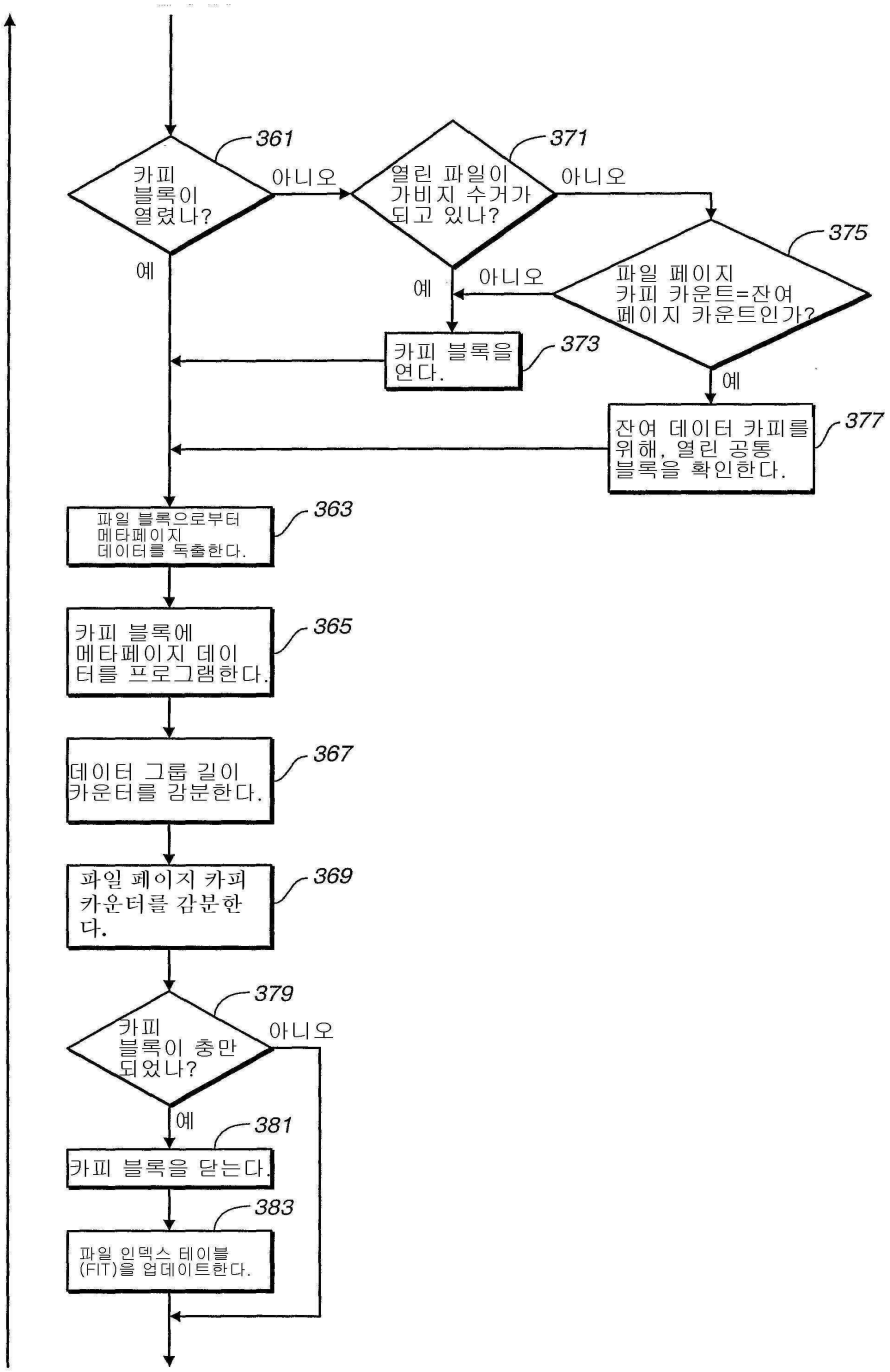
도면31A


가비지 수거
355
가비지 수거가 진행 중인가?
아니오
예
356
가비지 수거 대기열에 엔트리들이 존재하는가?
아니오
예
357
가비지 수거가 파일에 대한 것인가?
예
아니오
358
대기열로부터 엔트리를 선택한다.
421
소거된 가용 블록들>N2?
아니오
예
360
가비지 수거를 위한 엔트리가 파일에 대한 것인가?
아니오
예
362
가비지 수거가 공통 블록에 대한 것인가?
아니오
예
423
열린 파일이 존재하는가?
아니오
예
366
가비지 수거가 폐용 블록에 대한 것인가?
아니오
예
364
공통 블록 가비지 수거(도32)
368
블록 소거
425
가비지 수거를 위해 열린 파일을 선택
370
데이터 그룹 수 및 데이터 그룹 길이 카운터들을 초기화
372
파일 페이지 카피 카운터 및 잔여 페이지 카운트를 초기화
리턴

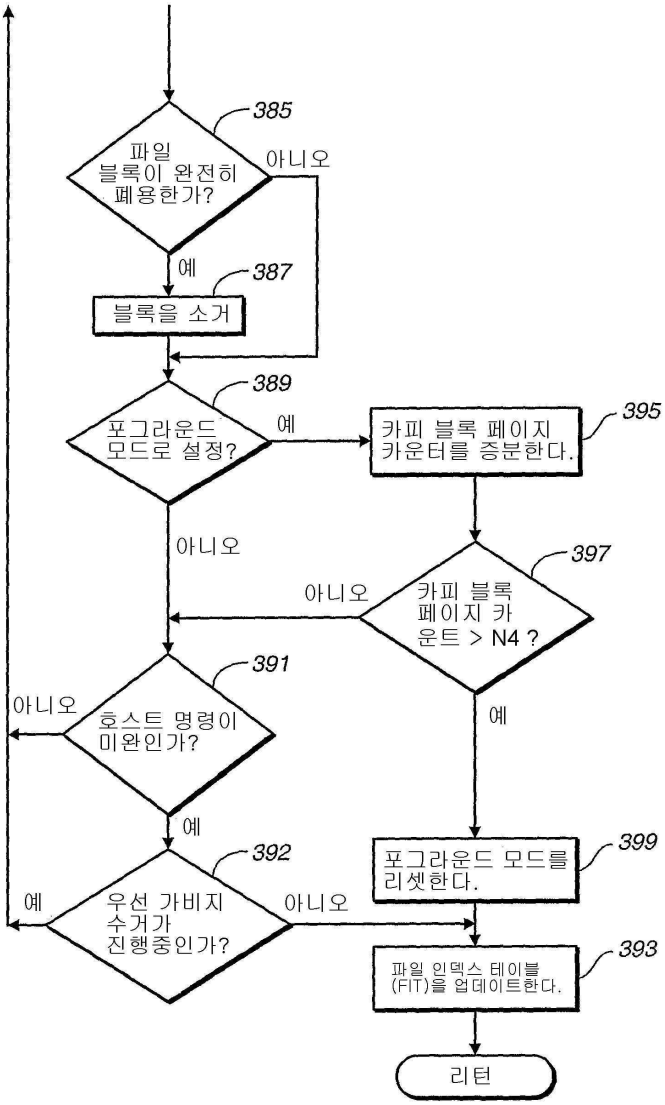
도면31B


359
데이터 그룹 길이 카운트>0?
아니오
401
파일 인덱스 테이블을 업데이트
예
403
파일에 대한 다음 데이터 그룹이 존재하는가?
예
아니오
405
데이터 그룹이 폐용 파일 블록에 있나?
아니오
예
411
데이터 그룹이 공통 블록에 있나?
아니오
예
415
데이터 그룹이 완전하지 않은 블록에 있나?
아니오
예
413
공통 블록을 가비지 수거 대기열에 추가한다.
417
완전하지 않은 블록이 열린 파일에 있나?
아니오
예
407
데이터 그룹 수 카운터를 업데이트한다.
409
데이터 그룹 길이 카운터 초기화한다.

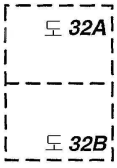
도면31C


361
카피 블록이 열렸나?
아니오
371
열린 파일이 가비지 수거가 되고 있나?
아니오
375
파일 페이지 카피 카운트=잔여 페이지 카운트인가?
예
아니오
예
373
카피 블록을 연다.
377
잔여 데이터 카피를 위해, 열린 공통 블록을 확인한다.
363
파일 블록으로부터 메타페이지 데이터를 독출한다.
365
카피 블록에 메타페이지 데이터를 프로그램한다.
367
데이터 그룹 길이 카운터를 감분한다.
369
파일 페이지 카피 카운터를 감분한다.
379
카피 블록이 충만되었나?
아니오
예
381
카피 블록을 닫는다.
383
파일 인덱스 테이블(FIT)을 업데이트한다.

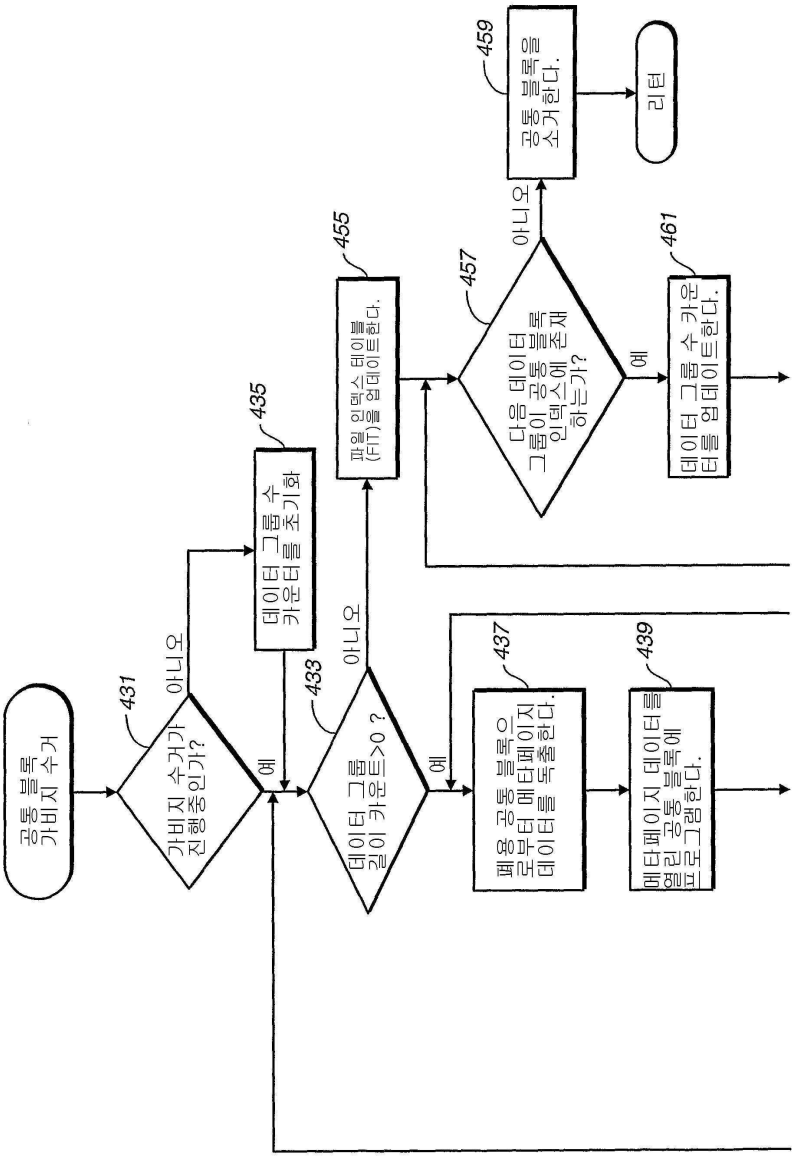
도면31D


385
파일 블록이 완전히 폐용한가?
아니오
예
387
블록을 소거
389
포그라운드 모드로 설정?
예
395
카피 블록 페이지 카운터를 증분한다.
아니오
397
아니오
카피 블록 페이지 카운트 > N4 ?
391
아니오
호스트 명령이 미완인가?
예
392
예
우선 가비지 수거가 진행중인가?
아니오
399
포그라운드 모드를 리셋한다.
393
파일 인덱스 테이블 (FIT)을 업데이트한다.
리턴

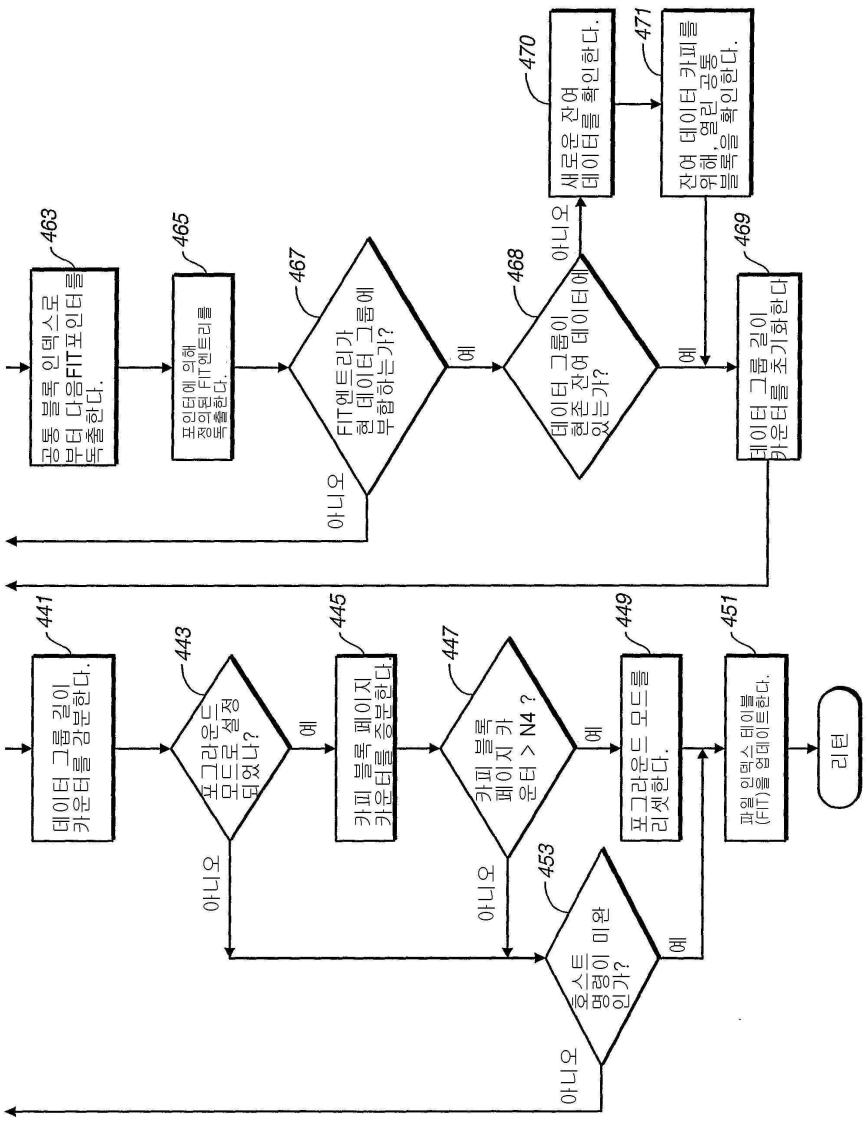
도면32


도 32A
도 32B

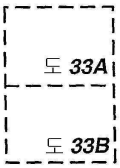
도면32A


공통 블록 가비지 수거
431
가비지 수거가 진행중인가?
아니오
예
435
데이터 그룹 수 카운터를 초기화
433
데이터 그룹 길이 카운트>0 ?
아니오
예
455
파일 입/출력 테이블(FIT)을 업데이트한다.
457
다음 데이터 그룹이 공통 블록 인덱스에 존재하는가?
아니오
예
459
공통 블록을 소거한다.
리턴
461
데이터 그룹 수 카운터를 업데이트한다.
437
폐용 공통 블록으로부터 메타페이지 데이터를 독출한다.
439
메타페이지 데이터를 열린 공통 블록에 프로그램한다.

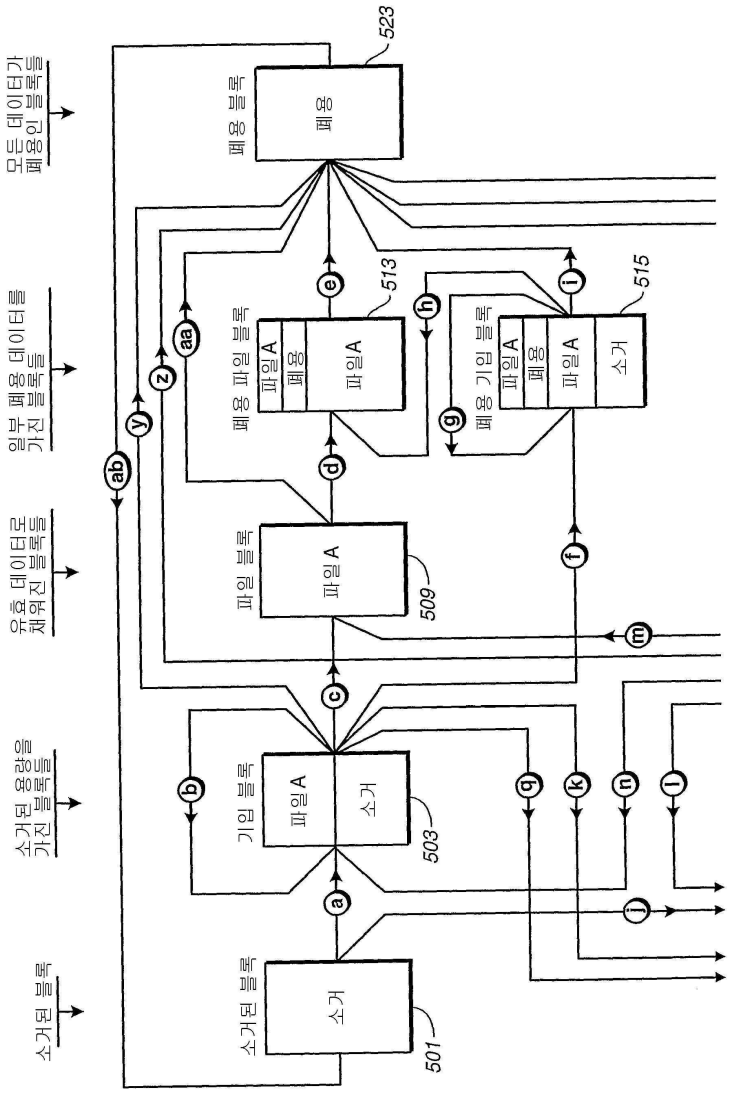
도면32B


441
데이터 그룹 길이 카운터를 감분한다.
443
포그라운드 모드로 설정 되었나?
예
아니오
445
카피 블록 페이지 카운터를 증분한다.
447
카피 블록 페이지 카운터 > N4 ?
예
아니오
449
포그라운드 모드를 리셋한다.
451
파일 인덱스 테이블(FIT)을 업데이트한다.
453
호스트 명령이 미완인가?
예
아니오
리턴
463
공통 블록 인덱스로부터 다음 FIT 포인터를 독출한다.
465
포인터에 의해 정의된 FIT 엔트리를 독출한다.
467
FIT 엔트리가 현 데이터 그룹에 부합하는가?
예
아니오
468
데이터 그룹이 현존 잔여 데이터에 있는가?
예
아니오
469
데이터 그룹 카운터 길이를 초기화한다
470
새로운 잔여 데이터를 확인한다.
471
잔여 데이터 카피를 위해, 열린 공통 블록을 확인한다.

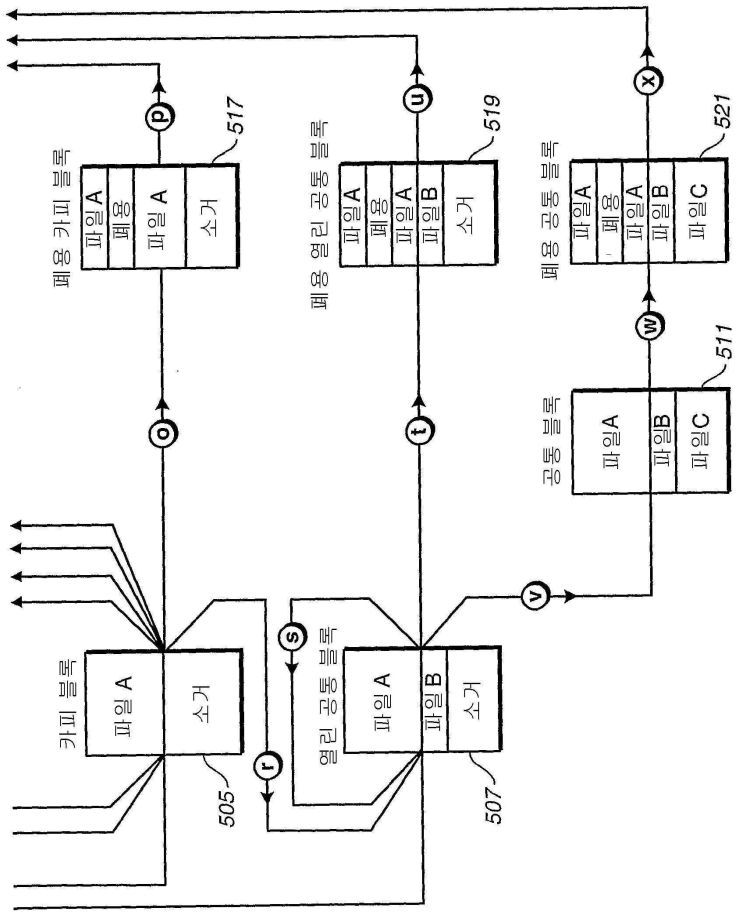
도면33


33A
33B

도면33A


모든 데이터가 폐용인 블록들
일부 폐용 데이터를 가진 블록들
유효 데이터로 채워진 블록들
소거된 용량을 가진 블록들
소거된 블록
폐용 블록
523
폐용
폐용 파일 블록
파일A
폐용
파일A
513
폐용 기입 블록
파일A
폐용
파일A
소거
515
파일 블록
파일A
509
기입 블록
파일A
소거
503
소거된 블록
소거
501
a
b
c
d
e
f
g
h
i
j
k
l
m
n
y
z
aa
ab

도면33B


폐용 카피 블록
파일A
폐용
파일A
소거
517
폐용 열린 공통 블록
파일A
폐용
파일A
파일B
소거
519
폐용 공통 블록
파일A
폐용
파일A
파일B
파일C
521
공통 블록
파일A
파일B
파일C
511
카피 블록
파일A
소거
505
열린 공통 블록
파일A
파일B
소거
507
o
p
q
r
s
t
u
v
w
x