(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2008-0038364
(43) 공개일자 2008년05월06일

(51) Int. Cl.
G06F 12/00 (2006.01)
(21) 출원번호 10-2008-7004671
(22) 출원일자 2008년02월27일
심사청구일자 없음
번역문제출일자 2008년02월27일
(86) 국제출원번호 PCT/US2006/030093
국제출원일자 2006년08월01일
(87) 국제공개번호 WO 2007/019174
국제공개일자 2007년02월15일
(30) 우선권주장
60/705,388 2005년08월03일 미국(US)

(71) 출원인
샌디스크 코포레이션
미합중국, 캘리포니아주 95035, 밀피타스, 맥카시 볼레바드 601
(72) 발명자
신클레어, 알란, 더블유.
영국, 매디스톤 팔키르크 에프케이2 0비유, 캔디,브로드헤드, 더 코테지스
라이트, 배리
영국, 이에이치6 6디이 스코트랜드, 에딘버르, 헨더슨스트리트, 플랫 4 40
(74) 대리인
박경재, 송범엽

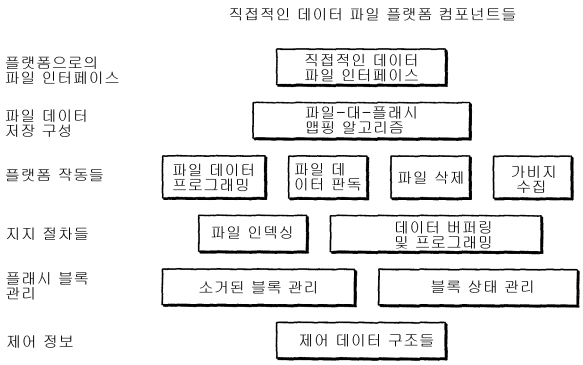
전체 청구항 수 : 총 14 항

(54) 직접적인 데이터 파일 저장소를 이용하는 플래시 메모리들에서의 데이터 작동들

(57) 요 약

호스트 시스템 데이터 파일들은 파일 내의 데이터의 오프셋들 및 각각의 파일의 특정 아이덴티피케이션을 갖지만, 메모리에 대한 가상 어드레스 공간 또는 임의의 중간 논리적 어드레스를 사용하지 않는 큰 소거 블록 플래시 메모리 시스템으로 직접 기록된다. 파일들이 메모리에 저장되는 디렉토리 정보는 호스트에 의해서라기보다는 오히려, 메모리 시스템의 제어기에 의해 메모리 시스템 내에서 유지된다.

대표도 - 도1B



특허청구의 범위

청구항 1

데이터가 내부에 기록되기 전에 개별적으로 소거되며 개별적인 파일들 내의 오프셋들 및 특정 파일 식별자들의 논리적 어드레스들을 갖는 데이터를 수신하는 메모리 셀들의 다수의 블록들을 갖는 재프로그래밍 가능한 비휘발성 메모리 시스템을 작동시키는 방법에 있어서,

제1 파일의 데이터가 적어도 제1 블록 및 제2 블록들에 저장될 때, 상기 제2 블록은 제1 파일의 데이터로 단지 부분적으로 채워지며 제2 블록의 나머지 부분은 소거되며,

제1 블록으로부터 재배치되는 제1 파일의 데이터에 응답하여, 제1 파일의 유효 데이터를 제1 블록으로부터 제2 블록의 소거된 부분 내로 카피하는 단계를 포함하며, 상기 유효 데이터는 완전히 소거된 블록 내로는 카피되지 않는 재프로그래밍 가능한 비휘발성 메모리 시스템을 작동시키는 방법.

청구항 2

제1항에 있어서, 제2 파일의 데이터는 또한 제1 블록으로부터의 제1 파일의 데이터를 카피하기 전에 제1 블록에 저장되는 재프로그래밍 가능한 비휘발성 메모리 시스템을 작동시키는 방법.

청구항 3

제1항에 있어서, 제1 파일의 데이터는 그룹 내에서 데이터의 인접한 논리적 오프셋 어드레스들 및 인접한 물리적 어드레스들 둘 모두를 개별적으로 갖는 다수의 데이터 그룹들로서 저장되는 재프로그래밍 가능한 비휘발성 메모리 시스템을 작동시키는 방법.

청구항 4

데이터가 내부에 기록되기 전에 개별적으로 소거되며 파일 내의 데이터의 오프셋들 및 특정 파일 식별자의 논리적 어드레스들을 갖는 데이터를 파일들로서 수신하는 메모리 셀들의 다수의 블록들을 갖는 재프로그래밍 가능한 비휘발성 메모리 시스템의 작동 방법에 있어서,

하나 이상의 소거된 블록들의 연속적인 페이지들로 파일의 데이터를 기록하는 단계, 및

파일의 데이터가 하나 이상의 소거된 블록들 중 하나의 페이지들 중 일부에만 기록되고 나서, 클로우즈되는 경우에, 또한 하나의 블록에 저장된 파일의 데이터 중 적어도 일부가 쓸모없는 경우에만, 파일을 클로우즈하도록 하는 명령을 수신하는 것에 응답하여 하나의 블록에 대해 가비지 수집 작동이 수행되는 단계를 포함하는 재프로그래밍 가능한 비휘발성 메모리 시스템의 작동 방법.

청구항 5

제4항에 있어서, 파일의 데이터를 기록하는 상기 단계는 그룹 내에서 데이터의 인접한 논리적 오프셋 어드레스들 및 인접한 물리적 어드레스들 둘 모두를 개별적으로 갖는 다수의 데이터 그룹들을 기록하는 단계를 포함하는 재프로그래밍 가능한 비휘발성 메모리 시스템의 작동 방법.

청구항 6

데이터가 내부에 기록되기 전에 개별적으로 소거되며 개별적인 파일들 내의 오프셋들 및 특정 파일 식별자들의 논리적 어드레스들을 갖는 데이터를 수신하는 메모리 셀들의 다수의 블록들을 갖는 재프로그래밍 가능한 비휘발성 메모리 시스템을 작동시키는 방법에 있어서,

소정 파일의 데이터를 제1 블록에 저장하는 단계,

소정 파일의 새롭게 수신된 데이터를 제2 블록에 기록하는 단계, 및

소정 파일의 데이터를 제1 블록으로부터 제2 블록 내로 카피함으로써 소정 파일의 가비지 수집을 수행하는 단계를 포함하는 재프로그래밍 가능한 비휘발성 메모리 시스템을 작동시키는 방법.

청구항 7

제6항에 있어서, 소정 파일의 저장 위치들의 레코드를 그룹 내에서 데이터의 인접한 논리적 오프셋 어드레스들 및 인접한 물리적 어드레스 둘 모두를 개별적으로 갖는 다수의 데이터 그룹들로서 메모리 시스템에서 유지하는 단계를 더 포함하는 프로그래밍 가능한 비휘발성 메모리 시스템을 작동시키는 방법.

청구항 8

데이터가 내부에 기록되기 전에 개별적으로 소거되는 메모리 셀들의 다수의 블록들을 갖는 재프로그래밍 가능한 비휘발성 메모리 시스템에 있어서:

개별적인 파일들 내의 오프셋들 및 특정 파일 식별자들의 논리적 어드레스들을 갖는 데이터가 수용되고,

제1 파일의 데이터가 적어도 제1 및 제2 블록에 저장되는데, 상기 제2 블록은 제1 파일의 데이터로 단지 부분적으로 채워지며 제2 블록의 나머지 부분은 소거되며,

제1 블록으로부터 재배치되는 제1 파일의 데이터에 응답하여, 제1 파일의 유효 데이터가 제1 블록으로부터 제2 블록의 소거된 부분 내로 카피되며, 상기 유효 데이터는 완전히 소거된 블록 내로는 카피되지 않는 재프로그래밍 가능한 비휘발성 메모리 시스템.

청구항 9

제8항에 있어서, 제2 파일의 데이터는 또한 제1 블록으로부터 제1 파일의 데이터를 카피하기 전에 제1 블록에 저장되는 재프로그래밍 가능한 비휘발성 메모리 시스템.

청구항 10

제8항에 있어서, 제1 파일의 데이터는 그룹 내에서 데이터의 인접한 논리적 오프셋 어드레스들 및 인접한 물리적 어드레스 둘 모두를 개별적으로 갖는 다수의 데이터 그룹들로서 저장되는 재프로그래밍 가능한 비휘발성 메모리 시스템.

청구항 11

파일 내의 데이터의 오프셋들 및 특정 파일 식별자의 논리적 어드레스들을 갖는 데이터를 파일들로서 수신할 수 있는 재프로그래밍 가능한 비휘발성 메모리 시스템에 있어서:

상기 메모리 시스템은 데이터가 내부에 기록되기 전에 개별적으로 소거되는 메모리 셀들의 다수의 블록들을 포함하고,

파일의 데이터가 소거 상태에 있을 때 하나 이상의 블록들의 연속적인 페이지들로 기록되며,

파일의 데이터가 하나의 블록 중 하나의 페이지들 중 일부만에 기록되고 나서, 클로우즈되는 경우, 및 하나의 블록에 저장된 파일의 데이터 중 적어도 일부가 쓸모없게 된 경우에만, 파일을 클로우즈하도록 하는 명령을 수신하는 것에 응답하여 하나 이상의 블록들 중 적어도 하나에 기록된 데이터에 대해 가비지 수집 작동이 수행되는 재프로그래밍 가능한 비휘발성 메모리 시스템.

청구항 12

제11항에 있어서, 파일의 데이터는 그룹 내에서 인접한 논리적 오프셋 어드레스들 및 인접한 물리적 어드레스들 둘 모두를 개별적으로 갖는 다수의 데이터 그룹들의 형태로 기록되는 재프로그래밍 가능한 비휘발성 메모리 시스템.

청구항 13

개별적인 파일들 내의 오프셋들 및 특정 파일 식별자들의 논리적 어드레스들을 갖는 데이터를 수신하는 재프로그래밍 가능한 비휘발성 메모리 시스템에 있어서:

상기 메모리 시스템은 데이터가 내부에 기록되기 전에 개별적으로 소거되는 메모리 셀들의 다수의 블록들을 포함하고,

소정 파일의 데이터가 제1 블록에 저장되며,

소정 파일의 새롭게 수신된 데이터가 제2 블록에 기록되고,

소정 파일의 데이터를 제1 블록으로부터 제2 블록 내로 카피함으로써 소정 파일의 가비지 수집이 수행되는 재프로그래밍 가능한 비휘발성 메모리 시스템.

청구항 14

제13항에 있어서, 소정 파일의 데이터는 그룹 내에서 인접한 논리적 오프셋 어드레스들 및 인접한 물리적 어드레스들 둘 모두를 개별적으로 갖는 다수의 데이터 그룹들의 형태로 기록되는 재프로그래밍 가능한 비휘발성 메모리 시스템.

명 세 서

기 술 분 야

<1> 본 출원은 일반적으로 호스트 장치 및 메모리 시스템 사이의 인터페이스의 관리를 포함한, 반도체 플래시 메모리와 같은 재프로그래밍 가능한 비휘발성 메모리 시스템들의 작동에 관한 것이며, 특히 공통 대형 메모리 논리적 어드레스 공간(LBA) 인터페이스보다는 오히려, 데이터 파일 인터페이스의 효율적인 사용에 관한 것이다.

배 경 기 술

<2> Alan W. Sinclair 단독 명의, 또는 Peter J. Smith와 공동 명의의 모두 2005년 2월 16일자로 출원된 계류중인 미국 특허 출원 번호 11/060,174, 11/060,248 및 11/060,249(이하에서, "종래의 출원들"로서 칭해짐)에서 설명되는 플래시 메모리의 다양한 작동들에서의 개선점들이 본원에 설명되어 있다.

<3> 추가적인 개선점들은 Alan W. Sinclair 및 Barry Wright의 관련된 미국 특허 출원들, 즉 모두 2006년 5월 8일자로 출원된 비-가출원 번호 11/382,224와 11/382,232, 및 가출원 번호 60/746,740과 60/746,742, 및 모두 2006년 7월 21일자로 출원된 명칭이 "Indexing Of File Data In Reprogrammable Non-Volatile Memories That Directly Store Data Files", "Reprogrammable Non-Volatile Memory Systems With Indexing of Directly Stored Data Files", "Methods of Managing Blocks in NonVolatile Memory" 및 "NonVolatile Memory With Block Management"인 비-가출원들에서 설명되어 있다.

발명의 상세한 설명

<4> 본원에 인용된 모든 특허들, 특허 출원들, 아티클들 및 다른 간행물들, 문서들 및 사물들(things)은 모든 목적들을 위해 전체적으로 본원에 참조되어 있다. 포함된 간행물들, 문서들 또는 사물들 중 어느 하나 및 본 출원 사이에서 용어들의 정의 또는 이용에서 임의의 불일치 또는 상충이 존재하면, 본 출원의 용어들의 정의 또는 이용이 지배적이다.

<5> 데이터 통합은 본원에서 가비지 수집과 상이하게 취급되며, 적어도 부분적으로 상이한 알고리즘들에 의해 2개의 프로세스들이 구현된다. 파일 또는 메모리 블록이 쓸모없는 데이터를 포함할 때, 파일 또는 블록의 유효 데이터를 하나 이상의 다른 블록들로 이동시키기 위하여 가비지 수집 작동이 이용된다. 이것은 유효 데이터를 더 적은 수의 블록들 내로 모으므로, 일단 원래 소스 블록(들)이 소거되면 쓸모없는 데이터에 의해 차지된 용량을 자유롭게 한다. 데이터 통합에서, 새로운 파일을 기록하는 결과로서 통상적으로 생성되는 바와 같은, 하나의 부분적으로 채워진 블록의 유효 데이터는 또 다른 부분적으로 채워진 블록의 유효 데이터와 결합된다. 그 후, 지금 쓸모없는 복제 데이터를 포함하는, 데이터의 소스였던 원래 블록들 중 하나 또는 둘 모두는 가비지 수집이 스케줄링(sheduling)된다. 쓸모없는 데이터에 의해 차지된 메모리 저장 용량을 회복하기 위하여 개별적인 가비지 수집 작동들을 스케줄링하기 위해 큐(queue)들이 제공될지라도, 가비지 수집이 스케줄링되지 않고 조건들이 통합에 만족스러울 때, 바람직하게는 데이터 통합이 발생한다.

<6> 파일이 클로우즈(close)된 직후에 새롭게 기록된 파일의 데이터 통합을 스케줄링하기보다는 오히려, 새롭게 기록된 파일의 데이터는 자신들이 호스트로부터의 수신 이후에 프로그래밍되었던 원래 블록들에서 유지된다. 이것은 가장 통상적으로 새로운 데이터로 부분적으로 채워지는 블록을 포함한다. 데이터 파일이 쓸모없는 데이터를 생성하는 방식으로 삭제(delete)되거나 갱신되는 것이 통상적이기 때문에, 부분적으로 채워진 블록의 데이터의 통합은 파일이 클로우즈된 후에 가능한 한 길게 연기된다. 파일은 임의의 데이터를 재배치할 필요 없이 삭제될 수 있다. 따라서, 이와 같은 통합이 필요로 되기 전에 파일이 삭제되거나 갱신되는 경우, 통합은 피해진다. 이와 같은 통합은 메모리가 가득 찰 때, 메모리가 새로운 데이터의 추가적인 프로그래밍을 위해 충분히 소거된 블록들이 되도록 하기 위하여 필요로 될 수 있다. 그러나, 논리적 인터페이스가 사용되는 경우와 대조적으로, 파

일 기반으로 한 메모리가 호스트에 의해 삭제되었던 데이터 파일을 보유하지 않기 때문에, 메모리는 통상적으로 통합이 지연될지라도 충분한 수의 소거된 블록들을 가질 것이다. 따라서, 생략된 통합에 의해 드는 시간이 절약되며, 결과적으로 메모리의 성능이 개선된다.

<7> 후술되는, 직접적인 파일 저장에서의 여러 다른 개선점들이 존재하며, 다음과 같이 요약될 수 있다:

<8> 1. 일단 파일이 클로우즈되면, 파일은 자신이 쓸모없는 데이터를 포함하지 않는다면 파일 가비지 수집 큐에 추가되지 않는다.

<9> 2. 파일의 가비지 수집은 또 다른 파일로부터의 데이터를 포함하는 공통 블록을 생성하지 않는다. 가비지 수집 동안 카피되어야 하는 파일에 대한 데이터는 파일에 대한 현재 프로그램 블록으로 프로그래밍된다. 프로그램 블록은 가비지 수집의 종점에서 부분적으로 프로그래밍된 채로 유지된다.

<10> 3. 파일의 삭제에 의해 쓸모없게 되는 블록 내의 파일 그룹의 결과로서 데이터가 공통 블록으로부터 재배치되어야 할 때, 나머지 유효 데이터는 프로그램 블록의 이용 가능한 공간 내로 재배치된다.

<11> 4. 프로그램 블록의 전체 또는 일부 내로 직접 기록된 호스트 데이터의 이동이 피해진다.

<12> 5. 파일의 가비지 수집 동안, 데이터는 파일에 대한 프로그램 블록에 재배치된다. 전용된 중간 카피 블록은 존재하지 않는다.

<13> 6. 데이터의 완전한 섹터들 내의 메모리 셀 어레이 및 메모리 시스템 제어기 버퍼 메모리 사이에서 데이터가 전달된다. 이것은 프로그래밍 동안 ECC의 발생 및 데이터 판독 동안 ECC의 검사를 허용한다.

<14> 7. 공통 블록 내의 데이터 그룹 또는 파일 그룹의 스타트(start)는 메타페이지의 스타트와 정렬된다. 결과적으로, 블록 통합을 위해 온-칩 카피가 사용될 수 있다. 프로그램 블록들 내의 데이터 그룹들은 물리적 구조들에 대한 특정한 정렬을 갖지 않는다.

<15> 8. 플래시 메모리 내의 스왑 블록(swap block)은 활성이 아닌 오픈 파일(open file)에 대한 휘발성 제어기 버퍼 메모리에 수용되는 데이터의 보안 카피(security copy)를 만드는데; 즉, 가장 최근의 기록 명령이 상이한 파일과 관련될 때, 사용된다. 이것은 또한 이용 가능한 버퍼 메모리 용량이 더 많은 수의 오픈 파일들을 이들 사이의 스왑 작동들의 사용을 통하여 지원하도록 하기 위하여 가상 버퍼 구조의 부분으로서 사용될 수 있다.

<16> 9. FIT 파일이 자신의 현재 범위가 가득 차서 또 다른 FIT 범위로 이동할 때, 디렉토리 내의 파일 데이터 포인터는 새로운 FIT 범위를 반영하도록 갱신된다.

<17> 10. FIT 범위에 대한 FIT 갱신 블록 내의 데이터는 FIT 갱신 블록 내에서의 범위에 대한 데이터의 량이 임계값을 초과할 때 그 범위에 대한 FIT 블록 내의 데이터와 통합된다. 이는 새로운 파일에 대한 데이터가 FIT 블록에 통합되도록 한다.

<18> 11. FIT 갱신 블록의 압축 동안, 클로우즈된 파일에 대한 FIT 파일은 충분한 소거된 공간이 존재하는 경우, 자신의 범위에 대해 FIT 블록에 재배치된다. 그렇지 않은 경우, 상기 파일은 압축된 FIT 갱신 블록에 재배치된다.

<19> 12. 호스트는 세트 내의 모든 파일들이 동일한 크기를 갖고 메타블록의 크기와 동일하도록 제어하기 위하여 기록_포인터 및 판독-포인터 명령들을 사용할 수 있고, 파일이 클로우즈된 직후에 세트 내의 파일이 단일 메타블록내로 통합되도록 하기 위하여 클로우즈 및 유휴(idle) 명령들을 사용할 수 있다.

<20> 13. 호스트 명령들의 세트는 명령된 데이터 기록 또는 판독이 시작되어야 하는 메모리 어드레스들을 제공하는 기록_포인터 및 판독-포인터의 값들에 대한 짝 명령(companion command)들을 포함하는 지정된 파일ID에 대한 판독 및 기록 명령들을 포함한다.

실 시 예

<69> 1. 직접적인 데이터 파일 플랫폼

<70> 1.1 요약

<71> 직접적인 데이터 파일 플랫폼을 갖는 메모리 카드가 도1A에 도시되어 있다. 직접적인 데이터 파일 플랫폼은 데이터가 파일이름 및 파일 오프셋 어드레스에 의해 식별되는 파일-유기적 데이터 저장 장치이다. 상기 직접적인 데이터 파일 플랫폼은 데이터 저장 이외의 기능들을 포함할 수 있는 메모리 카드에서 저장 플랫폼의 역할을 한다. 파일 데이터는 외부 파일 인터페이스 채널에 의해 플랫폼에서 액세스된다.

<72> 저장 장치는 논리적 어드레스들을 갖지 않는다. 각각의 파일에 대하여 독립적인 어드레스 공간들이 존재하고, 메모리 관리 알고리즘들이 데이터의 파일 구조에 따라 데이터 저장소를 구성한다. 직접적인 데이터 파일 플랫폼에서 사용되는 데이터 저장소 구성은 종래의 논리적으로 차단되는 메모리 관리를 갖는 종래의 파일 시스템을 포함하는 파일 저장 장치에 비하여, 작동 특성의 상당한 개선점을 발생시킨다.

<73> 1.2 플랫폼 컴포넌트들

<74> 직접적인 데이터 파일 플랫폼은 도1B에 도시된 바와 같이 기능의 층들로 구성된 다음의 컴포넌트들을 갖는다.

<75> 직접적인 데이터 파일 인터페이스: 카드 내의 다른 기능적 블록들로부터 파일이름 및 파일 오프셋 어드레스에 의해 식별되는 데이터로의 액세스를 제공하는 파일 API.

<76> 파일-대-플래시 맵핑 알고리즘: 파일 단편화(file fragmentation)를 제거하고 최대 성능 및 인듀어런스(endurance)을 제공하는 파일-유기적 데이터 저장을 위한 방식.

<77> 파일 데이터 프로그래밍: 파일-대-플래시 맵핑 알고리즘에 따라 파일 데이터 프로그래밍.

<78> 파일 데이터 판독: 플래시 메모리로부터의 오프셋 어드레스에 의해 지정된 데이터 판독.

<79> 파일 삭제: 삭제된 파일에 대한 데이터를 포함하는 블록들을 식별하고 이를 가비지 수집 큐들에 추가.

<80> 가비지 수집: 쓸모없는 데이터에 의해 차지된 메모리 용량을 복구하기 위하여 수행된 작동들. 이들은 블록을 소거하기 위하여 유효 데이터를 또 다른 위치에 카피하는 것을 수반할 수 있다.

<81> 파일 인덱싱: 파일 인덱싱은 파일에 대한 유효 데이터 그룹들의 위치들이 오프셋 어드레스 순서로 식별되도록 한다.

<82> 데이터 버퍼링 및 프로그래밍: 프로그래밍될 데이터에 대한 버퍼 메모리의 사용, 및 프로그램 블록들에서 파일 데이터를 프로그래밍하는 시퀀스.

<83> 소거된 블록 관리: 파일 데이터 또는 제어 정보의 저장을 위한 할당에 이용 가능한 장치 내의 소거된 블록들의 풀(pool)의 관리.

<84> 블록 상태 관리: 파일 데이터의 저장을 위한 블록들이 분류될 수 있는 8개의 상태들 사이의 트랜지션(transition)들.

<85> 제어 데이터 구조들: 그 목적에 전용된 플래시 블록들에 저장된 제어 데이터 구조들.

<86> 2. 직접적인 데이터 파일 인터페이스

<87> 직접적인 데이터 파일 인터페이스는 플래시 대량 데이터 저장소를 포함하는 장치 내에서 플래시 메모리 관리를 위한 백-엔드 시스템(back-end system)을 형성하는, 직접적인 데이터 파일 플랫폼으로의 API이다.

<88> 2.1 명령 세트

<89> 다음 단락은 다수의 소스들과 파일-기반으로 한 인터페이싱을 지원하기 위한 일반적인 명령 세트를 규정한다. 명령들은 6개의 등급들로 규정된다.

<90> 1. 파일 명령들

<91> 2. 데이터 명령들

<92> 3. 파일 정보 명령들

<93> 4. (모델링만을 위한) 스트림 명령들

<94> 5. 상태 명령들

<95> 6. 장치 명령들

<96> 2.1.1 파일 명령들(도2A 참조)

<97> 파일은 파일ID에 의해 장치 내에서 독립적으로 식별되는 오브젝트이다. 파일은 호스트에 의해 생성된 데이터의 세트를 포함하거나, 데이터를 갖지 않을 수 있고, 이 경우에, 파일은 디렉토리 또는 폴더를 나타낸다.

<98> 2.1.1.1 생성(Create)

<99> 생성 명령은 장치의 디렉토리 내에서 <파일ID>에 의해 식별되는 엔트리를 생성한다. <파일ID> 파라미터가 생략되면, 장치는 이용 가능한 값을 파일에 할당하고, 이를 호스트에 리턴시킨다. 이것은 파일을 생성하는 통상적인 방법이다.

<100> 호스트는 대안적으로 <파일ID> 값을 파일에 할당할 수 있다. 이 방법은 파일ID의 특정 값이 호스트 인터페이스 프로토콜 내에서 특정 유형의 파일을 나타내는 경우에 사용될 수 있다. 예를 들어, 루트 디렉토리는 호스트에 의해 특정 파일ID를 할당받을 수 있다.

<101> 2.1.1.2 오픈(Open)

<102> 이 명령은 <파일ID>에 의해 지정된 파일에 대한 후속 데이터 명령들의 실행을 인에이블시킨다. 파일이 존재하지 않는 경우, 에러 메시지가 리턴된다. 파일에 대한 기록_포인터가 파일의 종점으로 설정되고, 파일에 대한 판독_포인터가 파일의 시작점으로 설정된다. 파일_정보에 대한 정보_기록 포인터가 파일_정보의 종점으로 설정되고, 파일에 대한 정보_판독_포인터가 파일_정보의 시작점으로 설정된다. 동시에 오픈될 수 있는 최대 수의 파일들이 존재한다. 이 수가 초과되는 경우, 명령이 실행되지 않고 에러 메시지가 리턴된다. 동시 오픈 파일들의 최대 수는 예를 들어, 8일 수 있다.

<103> 지정된 파일로의 기록을 위한 장치 내의 자원들은 단지 후속 기록, 삽입 또는 제거(remove) 명령의 수신 이후에만 이용 가능해진다.

<104> 2.1.1.3 클로우즈(Close)

<105> 이 명령은 지정된 파일에 대한 후속 데이터 명령들의 실행을 디스에이블시킨다. 파일에 대한 기록_포인터, 판독_포인터, 정보_기록_포인터 및 정보_판독_포인터 값들이 무효해진다.

<106> 2.1.1.4 삭제(Delete)

<107> 삭제 명령은 <파일ID>에 의해 지정되는 파일에 대한 정보 테이블 엔트리들과 파일 인덱스 테이블, 디렉토리가 삭제되어야 한다는 것을 나타낸다. 파일들에 대한 데이터가 소거될 수 있다. 삭제된 파일은 나중에 액세스될 수 없다.

<108> 2.1.1.5. 소거(Erase)

<109> 소거 명령은 <파일ID>에 의해 지정되는 파일에 대한 정보 테이블 엔트리들과 파일 인덱스 테이블, 디렉토리가 삭제되어야 한다는 것을 나타낸다. 파일 데이터는 임의의 다른 명령이 실행되기 전에 소거되어야 한다. 소거된 파일은 나중에 액세스될 수 없다.

<110> 2.1.1.6 리스트_파일(List_files)

<111> 디렉토리 내의 모든 파일들에 대한 파일ID 값들은 리스트-파일 명령의 수신 다음에 수치적인 순서로 장치로부터 스트리밍될 수 있다. 파일ID 스트리밍은 최종 파일이 도달될 때 종료되고, 이 조건은 상태 명령에 의하여 호스트에 의해 식별된다. 리스트_파일 명령은 임의의 다른 명령의 수신에 의해 종료된다.

<112> 2.1.2. 데이터 명령들(도2B 참조)

<113> 데이터 명령들은 지정된 파일에 대한 데이터 입력 및 출력 작동들을 개시하고 파일 내에서 오프셋 어드레스 값을 규정하는데 사용된다. 지정된 파일은 호스트에 의해 오픈되어야 한다. 지정된 파일이 호스트에 의해 오픈되지 않는다면, 에러가 리턴된다. <파일ID>는 파일이 최종 오픈될 때 호스트에 리턴되는 파일 핸들(file handle)이다.

<114> 2.1.2.1 기록(Write)

<115> 기록 명령의 수신 다음에 장치로 스트리밍되는 데이터는 기록_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일에서 겹쳐쓰기된다. 기록 명령은 파일에 대한 새로운 데이터를 기록하고, 데이터를 파일에 추가하고, 파일 내에서 데이터를 갱신하는데 사용된다. 기록 명령은 다른 임의의 명령의 수신에 의해 종료된다.

<116> 2.1.2.2 삽입(Insert)

<117> 삽입 명령의 수신 다음에 장치에 스트리밍되는 데이터는 기록_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일에서 삽입된다. 파일 크기는 삽입된 데이터의 길이만큼 증가된다. 삽입 명령은 임의의 다른 명

령의 수신에 의해 종료된다.

<118> 2.1.2.3 제거(Remove)

<119> 제거 명령은 기록_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일로부터 <길이>에 의해 규정된 순차적인 데이터를 삭제한다. 파일 크기는 <길이>만큼 감소된다.

<120> 2.1.2.4 판독(Read)

<121> 판독_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일 내의 데이터는 판독 명령의 수신 다음에 장치로부터 스트리밍될 수 있다.

<122> 데이터 스트리밍은 파일의 종점에 도달할 때 종료되고, 이 조건은 상태 명령에 의하여 호스트에 의해 식별될 수 있다. 판독 명령은 임의의 다른 명령의 수신에 의해 종료된다.

<123> 2.1.2.5 저장_버퍼(Save_buffer)

<124> 장치 버퍼에 포함되고 플래시 메모리로 아직 프로그래밍되지 않은 지정된 파일에 대한 데이터는 플래시 메모리 내의 일시적인 위치에 저장된다.

<125> 데이터는 후속 기록 또는 삽입 명령이 수신될 때 버퍼로 복귀되며, 명령과 관련된 데이터와 함께 플래시에 프로그래밍된다.

<126> 2.1.2.6 기록_포인터(Write_pointer)

<127> 기록_포인터 명령은 지정된 파일에 대한 기록_포인터를 지정된 오프셋 어드레스로 설정한다. 기록_포인터는 기록 또는 삽입 명령 다음에 데이터가 장치에 스트리밍될 때 장치에 의해 증분된다.

<128> 2.1.2.7 판독_포인터(Read_pointer)

<129> 판독_포인터 명령은 지정된 파일에 대한 판독_포인터를 지정된 오프셋 어드레스로 설정한다. 판독_포인터는 판독 명령 다음에 데이터가 장치로부터 스트리밍될 때 장치에 의해 증분된다.

<130> 2.1.3 정보 명령들(도2C 참조)

<131> 파일_정보는 파일과 관련되는 호스트에 의해 발생된 정보이다. 파일_정보의 특성 및 내용은 호스트에 의해 결정되며, 장치에 의해 해석되지 않는다. 정보 명령들은 지정된 파일에 대한 파일_정보 입력 및 출력 작동들을 개시하고 파일_정보 내의 오프셋 어드레스 값들을 규정하는데 사용된다.

<132> 2.1.3.1 기록_정보(Write_info)

<133> 기록_정보 명령의 수신 다음에 장치에 스트리밍된 파일_정보는 정보_기록_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일에 대해 파일_정보를 겹쳐쓰기한다. 지정된 파일에 대한 파일_정보의 내용 및 길이는 호스트에 의해 결정된다. 기록_정보 명령은 임의의 다른 명령의 수신에 의해 종료된다.

<134> 2.1.3.2 판독_정보(Read_info)

<135> 정보_판독_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일에 대한 파일_정보는 판독_정보 명령의 수신 다음에 장치로부터 스트리밍될 수 있다. 파일_정보 스트리밍은 파일_정보의 종점에 도달할 때 종료되며, 이 조건은 상태 명령에 의하여 호스트에 의해 식별된다. 판독_정보 명령은 임의의 다른 명령의 수신에 의해 종료된다.

<136> 2.1.3.3 정보_기록_포인터(Info-write_pointer)

<137> 정보_기록_포인터 명령은 지정된 파일에 대한 정보_기록_포인터를 지정된 오프셋 어드레스로 설정한다. 정보_기록_포인터는 기록_정보 명령 다음에 파일_정보가 장치로 스트리밍될 때 장치에 의해 증분된다.

<138> 2.1.3.4 정보_판독_포인터(Info_read_pointer)

<139> 정보_판독_포인터 명령은 지정된 파일에 대한 정보_판독_포인터를 지정된 오프셋 어드레스로 설정한다. 정보_판독_포인터는 판독_정보 명령 다음에 파일_정보가 장치로부터 스트리밍될 때 장치에 의해 증분된다.

<140> 2.1.4 스트림 명령들(도2D 참조)

<141> 스트림 명령들은 직접적인 데이터 파일 플랫폼의 작동 모델(behavioural model)로만 사용된다. 이들의 목적은

데이터 명령들과 관련하여, 호스트로 그리고 호스트로부터 스트리밍 데이터를 에뮬리이팅하는 것이다.

<142> 2.1.4.1 스트림(Stream)

<143> 스트림 명령은 플랫폼으로 또는 상기 플랫폼으로부터 호스트에 의하여 전달되어야 하는 <길이>에 의해 규정된 데이터의 인터럽트되지 않은 스트림을 에뮬레이팅한다. 스트림의 나머지 길이를 나타내는 변수는 데이터가 버퍼 메모리로부터 추가되거나 제거될 때 플랫폼의 모델만큼 감소된다.

<144> 2.1.4.2 중단(Pause)

<145> 중단 명령은 직접적인 데이터 파일 모델의 작동을 제어하고 있는 명령 리스트 내의 다음 명령의 실행 이전에 삽입되는 길이 <시간>의 지연을 삽입한다. <시간>은 마이크로초로 규정된다.

<146> 2.1.5 상태 명령들(도 2F 참조)

<147> 상태 명령들은 장치의 상태를 제어한다.

<148> 2.1.5.1 유휴(Idle)

<149> 유휴 명령은 호스트가 직접적인 데이터 파일 장치를 상기 장치가 내부 하우스키핑 작동들을 수행할 수 있는 유휴 상태로 이르게 한다는 것을 나타낸다. 호스트는 유휴 상태에서 상기 장치로부터 전력을 고의로 제거하지는 않을 것이다. 유휴 상태는 상기 장치가 내부 작동으로 작동중(busy)이든지 아니든지 간에, 호스트에 의한 임의의 다른 명령의 전송에 의해 종료될 수 있다. 이와 같은 다른 명령의 수신 시에, 상기 장치에서 진행중인 임의의 내부 작동은 지정된 시간 내에 중단되거나 종료되어야 한다. 이 시간의 일례는 10 밀리초 또는 그 이하이다.

<150> 2.1.5.2 대기(Standby)

<151> 대기 명령은 호스트가 직접적인 데이터 파일 장치를 상기 장치가 내부 하우스키핑 작동들을 수행할 수 없는 대기 상태에 이르게 한다는 것을 나타낸다. 호스트는 대기 상태에서 상기 장치로부터 전력을 고의로 제거하지는 않을 것이다. 대기 상태는 호스트에 의한 임의의 다른 명령의 전송의 의해 종료될 수 있다.

<152> 2.1.5.3 셧다운(Shutdown)

<153> 셧다운 명령은 장치가 다음에 작동중인 상태가 아닐 때 전력이 호스트에 의해 장치로부터 제거될 것이라는 것을 나타낸다. 모든 오픈 파일들은 셧다운 명령에 응답하여 장치에 의해 클로우즈된다.

<154> 2.1.6 장치 명령들(도2F 참조)

<155> 장치 명령들은 호스트가 장치에 질문하도록 한다.

<156> 2.1.6.1 용량(Capacity)

<157> 용량 명령에 응답하여, 장치는 장치에 저장된 파일 데이터의 용량, 및 새로운 파일 데이터에 이용 가능한 용량을 보고한다.

<158> 2.1.6.2 상태(Status)

<159> 상태 명령에 응답하여, 장치는 장치의 현재 상태를 보고한다.

<160> 상태는 3가지 유형의 작동중인 상태를 포함한다:

<161> 1. 장치는 데이터를 기록 또는 판독하기 위하여 포어그라운드(foreground) 작동을 수행하면서 작동중이다.

<162> 2. 장치는 자신이 유휴 상태에 있었던 동안에 개시된 백그라운드 작동을 수행하면서 작동중이다.

<163> 3. 버퍼 메모리가 작동중이고, 데이터를 기록 또는 판독하기 위하여 호스트에 이용 가능하지 않다.

<164> 2.1.7 명령 파라미터들

<165> 다음 파라미터들은 아래에 규정된 바와 같은 명령들과 함께 사용된다.

<166> 2.1.7.1 파일 ID

<167> 이것은 장치의 디렉토리 내에서 파일을 식별하는데 사용되는 파일 식별자이다.

<168> 2.1.7.2 오프셋

<169> 오프셋은 파일 또는 파일_정보의 시작과 관련된, 바이트 단위의, 파일 또는 파일_정보 내의 논리적 어드레스이다.

<170> 2.1.7.3 길이

<171> 이것은 순차적인 오프셋 어드레스들을 갖는 파일에 대한 데이터의 실행의 바이트 단위의 길이이다.

<172> 2.1.7.4 시간

<173> 이것은 마이크로초 단위의 시간이다.

<174> 3. 파일-대-플래시 맵핑 알고리즘

<175> 직접적인 데이터 파일 플랫폼에 의해 채용된 파일-대-플래시 맵핑 알고리즘은 호스트가 파일-기반으로 한 인터페이스를 통하여 파일 데이터 기록 및 파일 삭제 작동들을 수행할 때, 최대 시스템 성능 및 최대 메모리 인듀어런스를 제공하기 위하여 규정되었던 파일-유기적 데이터 저장을 위한 새로운 방식이다. 상기 맵핑 알고리즘은 플래시 메모리 내의 블록들 사이에서 파일 데이터를 카피하는 것을 최소화하도록 디자인되었다. 이것은 하나 이상의 파일에 대한 데이터를 포함하는 블록들의 가능한 최소 인시던스(incidence)를 성취하는 방식으로 플래시 블록들에 파일 데이터를 맵핑함으로써 성취된다.

<176> 3.1 파일-대-플래시 맵핑 원리들

<177> 3.1.1 파일들

<178> 파일은 호스트에 의해 생성되고 유지되는 데이터의 세트이다. 데이터는 파일이름에 의해 호스트에 의해 식별되고, 파일의 시작점으로부터 자신의 오프셋 위치에 의해 액세스될 수 있다. 파일 오프셋 어드레스는 호스트에 의해 설정될 수 있고, 장치에 의해 기록 포인터로서 증분될 수 있다.

<179> 3.1.2 물리적 메모리 구조들

<180> 직접적인 데이터 파일 플랫폼은 파일들에 대한 모든 데이터를 고정된-크기의 메타블록들에 저장한다. 메타블록을 포함하는 플래시 소거 블록들의 실제 수, 즉, 소거 블록 병렬성(erase block parallelism)은 제품들 사이에서 가변될 수 있다. 본 명세서 전체에 걸쳐서, 용어 "블록"은 "메타블록"을 나타내는데 사용된다.

<181> 용어 "메타블록"은 메타블록의 전체 병렬성으로 페이지를 나타내는데 사용된다. 메타블록은 프로그래밍의 최대 단위이다.

<182> 용어 "페이지"는 메모리의 플레인 내에서, 즉 플래시 소거 블록 내에서 페이지를 나타내는데 사용된다. 페이지는 프로그래밍의 최소 단위이다.

<183> 용어 "섹터"는 ECC가 관련되는 저장된 데이터의 단위를 나타내는데 사용된다. 섹터는 플래시 메모리로의 그리고 플래시 메모리로부터의 데이터 전달의 최소 단위이다.

<184> 파일에 대한 오프셋 어드레스들 및 물리적 플래시 메모리 구조들 사이에서 유지되는 지정된 정렬이 존재하지 않는다.

<185> 3.1.3 데이터 그룹들

<186> 데이터 그룹은 단일 메모리 블록 내의 인접한 물리적 어드레스들에서 프로그래밍된, 파일 내의 인접한 오프셋 어드레스들을 갖는 파일 데이터의 세트이다. 파일은 통상적으로 다수의 데이터 그룹들로서 프로그래밍될 것이다. 데이터 그룹은 한 바이트 및 한 블록 사이에 임의의 길이를 가질 수 있다. 각각의 데이터 그룹은 교차상관 목적들을 위한 파일 식별자 정보를 포함한 헤더와 함께 프로그래밍된다. 파일에 대한 데이터는 자신이 포함되는 데이터 그룹에 따라 물리적 메모리에서 인덱싱된다. 파일 인덱스 테이블은 파일의 각각의 데이터 그룹에 대한 파일 오프셋 어드레스 및 물리적 어드레스 정보를 제공한다.

<187> 3.1.4 프로그램 블록들

<188> 파일은 파일 데이터가 프로그래밍되도록 하기 위하여 호스트에 의해 오픈되어야 한다. 각각의 오픈 파일은 프로그램 블록으로서 할당되는 전용 블록을 가지며, 그 파일에 대한 데이터는 프로그램 블록 내에서 프로그램 포인터에 의해 규정된 위치에서 프로그래밍된다. 파일이 호스트에 의해 오픈될 때, 파일에 대한 프로그램 블록은 이미 존재하지 않는다면 오픈된다. 프로그램 포인터는 프로그램 블록의 시작점으로 설정된다. 프로그램 블록이 호스트에 의해 오픈되는 파일에 대해 이미 존재한다면, 상기 프로그램 블록은 파일에 대한 데이터를 프로그래밍하

는데 지속적으로 사용된다.

<189> 파일 데이터는 파일 내의 자신의 오프셋 어드레스 또는 그 오프셋 어드레스에 대한 데이터가 이전에 프로그래밍되었었는지 여부에 관계없이, 자신이 호스트로부터 수신되는 순서로 프로그램 블록에서 프로그래밍된다. 프로그램 블록이 가득 찰 때, 상기 프로그램 블록은 파일 블록으로서 인식되고, 소거된 블록 풀(pool)로부터의 소거된 블록이 새로운 프로그램 블록으로서 오픈된다. 파일에 대한 데이터를 저장하는 블록들 사이에 물리적 어드레스 관계가 존재하지 않는다.

<190> 3.1.5 공통 블록들

<191> 공통 블록은 하나 이상의 파일에 대한 데이터 그룹들을 포함한다. 동일한 파일에 대한 다수의 데이터 그룹들이 공통 블록에서 존재한다면, 상기 다수의 데이터 그룹들은 인접하여 위치되고, 인접한 유닛이 파일 그룹으로서 인식된다. 데이터는 블록 통합 작동 또는 공통 블록 가비지 수집 작동 동안에만 공통 블록으로 프로그래밍된다.

<192> 공통 블록 내에서의 개별적인 데이터 그룹 또는 파일 그룹의 시작점은 메타페이지의 시작점과 정렬되어야 한다.

<193> 파일 그룹 내의 데이터 그룹들은 개재 공간(intervening space)을 갖지 않는다. 이와 같은 데이터 그룹들 사이의 경계는 페이지 내에서 발생할 수 있다. 파일은 단지 단일 공통 블록 내의 데이터를 가져야 한다(이에 대한 예외에 대해선 8.3.4 참조).

<194> 3.2 파일 유형들

<195> 3.2.1 플레인 파일(Plain File)

<196> 플레인 파일은 임의의 수의 완전한 파일 블록들 및 하나의 부분적으로 프로그래밍된 프로그램 블록을 포함한다. 플레인 파일은 통상적으로 순차적인 오프셋 어드레스 순서로, 호스트로부터의 파일에 대한 데이터의 프로그래밍에 의하여, 또는 편집된 파일의 가비지 수집에 의하여 생성될 수 있다. 예시적인 플레인 파일이 도3A에 도시되어 있다. 플레인 파일은 오픈 파일 또는 클로우즈 파일 중 하나일 수 있다.

<197> 파일에 대한 추가적인 데이터는 프로그램 블록 내의 프로그램 포인터에서 프로그래밍될 수 있다. 파일이 호스트에 의해 삭제된 경우, 파일의 데이터를 포함하는 블록들은 데이터를 이와 같은 블록들로부터 플래시 메모리 내의 또 다른 위치로 카피할 필요 없이 즉시 소거될 수 있다. 따라서, 플레인 파일 포맷은 매우 유효하며, 파일들을 가능한 한 길게 이 포맷으로 유지하는데 있어서 장점이 존재한다.

<198> 3.2.2 공통 파일

<199> 공통 파일은 임의의 수의 완전한 파일 블록들 및 다른 관련되지 않은 파일들에 대한 데이터와 함께 그 파일에 대한 데이터를 포함하는 하나의 공통 블록을 포함한다. 예들이 도3B에 도시되어 있다. 공통 파일은 가비지 수집 작동 동안 또는 프로그램 블록들의 통합에 의하여 플레인 파일로부터 생성될 수 있다.

<200> 공통 파일은 통상적으로 클로우즈된 파일이며, 관련된 기록 포인터를 갖지 않는다. 호스트가 공통 파일을 오픈하는 경우, 프로그램 블록이 오픈되고 프로그램 포인터는 프로그램 블록의 시작점으로 설정된다. 파일이 호스트에 의해 삭제되는 경우, 이의 파일 블록들은 즉시 소거될 수 있지만, 관련되지 않은 파일 또는 관련되지 않은 파일들에 대한 데이터는 공통 블록이 소거되기 전에, 가비지 수집 작동에서 공통 블록으로부터 플래시 메모리 내의 또 다른 위치로 카피되어야 한다.

<201> 3.2.3 편집된 플레인 파일

<202> 플레인 파일은 그 파일에 대한 이전에-프로그래밍된 오프셋 어드레스들에 대한 갱신된 데이터를 기록하는 호스트에 의해 언제라도 편집될 수 있다. 예들이 도3C에서 제공된다. 이와 같은 갱신된 데이터는 프로그램 블록 내의 프로그램 포인터에서 통상적인 방식으로 프로그래밍되며, 결과적인 편집된 플레인 파일은 하나 이상의 쓸모없는 파일 블록들 또는 프로그램 블록들 자체에서 쓸모없는 데이터를 포함할 것이다.

<203> 편집된 플레인 파일은 그 파일에 대한 가비지 수집 작동에서 플레인 파일로 복구될 수 있다. 이와 같은 가비지 수집 동안, 임의의 유효 파일 데이터가 각각의 쓸모없는 파일 블록으로부터 그 파일에 대한 프로그램 포인터로 카피되고, 결과적인 완전히-쓸모없는 블록들은 소거된다. 가능하다면, 가비지 수집은 파일이 호스트에 의해 클로우즈될 때까지 수행되지 않는다.

<204> 3.2.4 편집된 공통 파일

<205> 오픈 공통 파일은 그 파일에 대한 이전에-프로그래밍된 오프셋 어드레스들에 대한 갱신된 데이터를 기록하는 호

스트에 의해 언제라도 편집될 수 있다. 예들이 도3D에 도시되어 있다. 이와 같은 갱신된 데이터는 프로그램 블록 내의 프로그램 포인터에서 통상적인 방식으로 프로그래밍되며, 결과적인 편집된 공통 파일은 하나 이상의 쓸모없는 파일 블록들, 공통 블록 또는 프로그램 블록들 자체에서 쓸모없는 데이터를 포함할 것이다.

<206> 편집된 공통 파일은 그 파일에 대한 가비지 수집 작동에서 플레인 파일 포맷으로 복구될 수 있다. 이와 같은 가비지 수집 동안, 임의의 유효 파일 데이터가 각각의 쓸모없는 파일 블록 및 공통 블록으로부터 그 파일에 대한 프로그램 포인터로 카피된다. 결과적인 완전히-쓸모없는 블록들은 소거되고, 쓸모없는 공통 블록은 별도의 후속 가비지 수집 작동을 위해 로깅(logging)된다.

<207> 3.3 가비지 수집 및 블록 통합

<208> 3.3.1 가비지 수집

<209> 가비지 수집 작동들은 쓸모없는 데이터에 의해 차지된 메모리 용량을 복구하기 위하여 수행된다. 이 작동들은 블록을 소거하기 위하여 유효 데이터를 또 다른 위치에 카피하는 것을 수반할 수 있다. 가비지 수집은 쓸모없는 데이터의 생성에 응답하여 즉시 수행될 필요가 없다. 진행중인 가비지 수집 작동들은 가비지 수집 큐들에서 로깅되고 나서, 스케줄링 알고리즘들에 따라 최적의 레이트로 수행된다.

<210> 직접적인 데이터 파일 플랫폼들은 호스트 명령에 의해 개시될 수 있는 백그라운드 가비지 수집 작동들을 지원한다. 이것은 호스트가 파일들이 나중에 호스트에 의해 기록될 때 더 높은 성능을 인에이블시키는 내부 하우스키핑 작동들에 대한 휴지 시간(quiescent time)을 장치에 할당하도록 한다.

<211> 충분한 백그라운드 시간이 호스트에 의해 이용 가능하지 않게 되는 경우, 장치는 포어그라운드 작동으로서 가비지 수집을 수행한다. 가비지 수집 작동들의 버스트들(bursts)은 호스트로부터의 파일 데이터를 프로그래밍하는 버스트들과 인터리빙된다. 인터리브 듀티 사이클은 백로그(backlog)가 구성되지 않도록 하면서, 최소의 가비지 수집 레이트를 유지하도록 적응형으로 제어될 수 있다.

<212> 3.3.2 블록 통합

<213> 장치 내의 각각의 플레인 파일은 불완전하게 채워진 프로그램 블록을 포함하며, 소거된 용량의 상당한 볼륨이 이와 같은 프로그램 블록들에서 록킹 업(locking up)될 수 있다. 공통 블록들은 또한 소거된 용량을 포함할 수 있다. 따라서, 클로우즈된 파일에 대한 프로그램 블록들 및 공통 블록들을 통합하는 진행중인 프로세스는 록킹된 소거된 용량을 제어하도록 구현된다. 블록 통합은 가비지 수집 기능의 부분으로서 취급되며, 동일한 스케줄링 알고리즘에 의해 관리된다.

<214> 프로그램 블록 또는 공통 블록 내의 데이터는 또 다른 공통 블록 또는 프로그램 블록으로부터의 이와 같은 관련되지 않은 데이터를 카피함으로써 하나 이상의 관련되지 않은 파일들에 대한 데이터와 통합된다. 원래 블록이 프로그램 블록이었다면, 상기 원래 블록이 공통 블록이 된다. 프로그램 블록을 또 다른 프로그램 블록보다는 오히려, 쓸모없는 공통 블록과 통합하는 것이 바람직하다. 쓸모없는 공통 블록은 쓸모없는 데이터를 포함하므로, 유효 데이터를 상기 블록으로부터 또 다른 위치로 재배치시키는 것을 불가피하다. 그러나, 프로그램 블록은 쓸모없는 데이터를 포함하지 않고, 데이터를 상기 블록으로부터 또 다른 위치로 카피하는 것은 바람직하지 않은 오버헤드(overhead)이다.

<215> 3.3.3 평형 상태

<216> 파일 데이터가 장치 용량의 높은 퍼센티지를 차지할 때, 호스트는 새로운 파일들을 기록하기 위한 용량을 생성하기 위하여 파일들에 대한 삭제 작동을 수행해야 한다. 이 상태에서, 장치 내의 대부분의 파일들은 플레인 파일 포맷의 파일들에 대한 프로그램 블록들에서 소거된 공간에 이용 가능한 용량이 거의 없기 때문에, 공통 파일 포맷을 가질 것이다.

<217> 공통 파일의 삭제는 관련되지 않은 파일들에 대한 유효 데이터가 가비지 수집 동안 자신의 공통 블록으로부터 재배치되는 것을 필요로 한다. 이와 같은 파일 그룹들에 대한 데이터는 클로우즈된 파일에 대한 하나 이상의 프로그램 블록들 내의 이용 가능한 용량에 가장 통상적으로 재배치된다. 호스트에 의해 최근에 기록되고 나서 클로우즈된 파일들에 대한 프로그램 블록들 내의 이용 가능한 사용하지 않은 용량 및 파일들이 호스트에 의해 삭제되는 결과로서 공통 블록들로부터 파일 데이터를 재배치하는데 필요로 되는 용량 사이에 자주 평형이 존재한다. 이 일반적인 평형 상태는 프로그램 블록으로부터 파일 데이터를 재배치할 필요성을 감소시키며, 파일-대-플래시 맵핑 알고리즘의 효율에 기여한다.

<218> 4. 장치 작동

<219> 4.1 장치 작동들의 실행

<220> 장치의 작동 시퀀스는 호스트에 의해 공급되는 명령들의 흐름에 의해 결정된다. 호스트 명령이 수신될 때, 현재 장치 작동은 인터럽트되고, 상기 명령이 통상적으로 해석된다. 일부 명령들은 다음과 같이 4개의 주요 장치 작동들 중 하나를 실행하도록 할 것이다:

<221> 1. 데이터 판독;

<222> 2. 데이터 프로그래밍;

<223> 3. 파일 삭제; 또는

<224> 4. 가비지 수집.

<225> 장치 작동은 다음 조건들 중 하나에 도달할 때까지 지속된다:

<226> 1. 작동이 완료된다;

<227> 2. 또 다른 호스트 명령이 수신된다; 또는

<228> 3. 포어그라운드 가비지 수집 모드에서 인터리빙된 버스트의 종점에 도달한다.

<229> 우선순위 가비지 수집 작동(priority garbage collection operation)들이 실행을 위해 큐잉되는 경우, 이 작동들은 임의의 새로운 명령이 해석되기 전에 완료된다.

<230> 장치 작동들을 나타낸 전체 흐름도가 도4A에 도시되어 있다.

<231> 5. 파일 데이터 프로그래밍

<232> 5.1 파일 데이터를 프로그래밍하는 원리들

<233> 파일에 대한 데이터는 호스트로부터의 기록 또는 삽입 명령 다음에 호스트로부터 장치에 스트리밍될 때 플래시 메모리로 프로그래밍된다. 충분한 데이터가 다음 프로그램 작동을 위해 버퍼 메모리에 축적될 때, 상기 데이터는 파일에 대한 프로그램 블록에서 프로그래밍된다. 이 작동의 설명에 대해서는 9장을 참조하라.

<234> 프로그램 블록이 가득 찰 때, 상기 프로그램 블록은 파일 블록으로서 지정되고, 소거된 블록 풀로부터의 소거된 블록이 프로그램 블록으로서 할당된다. 게다가, 쓸모없는 블록들 및 공통 블록들에 대한 파일 인덱스 테이블 및 가비지 수집 큐들이 갱신된다.

<235> 파일 데이터 프로그래밍 절차는 가비지 수집 스케줄링 알고리즘(8.4절 참조)에 의해 설정되는 인터리브 파라미터(N1)에 따라, 포어그라운드 가비지 수집의 버스트들을 개시한다. 인터리브 프로그램 카운터는 메타페이지 프로그램 작동이 플래시 메모리에서 개시될 때마다 증분되고, 포어그라운드 모드에서의 가비지 수집 작동은 이 카운터가 값(N1)을 초과할 때 개시된다.

<236> 파일 데이터 프로그래밍은 호스트가 또 다른 명령을 전송할 때까지 하나의 메타페이지 단위들로 지속된다.

<237> 파일 데이터를 프로그래밍하는 일례를 나타낸 흐름도가 도5A에 도시되어 있다.

<238> 6. 파일 데이터 판독

<239> 6.1 파일 데이터를 판독하는 원리들

<240> 호스트로부터의 판독 명령에 응답하여, 판독_포인터에 의해 지정되는 것에서 시작하는 파일 오프셋 어드레스들에 대한 데이터는 파일의 종점에 도달할 때까지, 플래시 메모리로부터 판독되고 순차적으로 호스트에 리턴된다. 파일 인덱스 테이블(FIT)이 판독되고, 파일에 대한 FIT 엔트리들이 판독_포인터에 대응하는 위치를 식별하기 위하여 평가된다. 후속 FIT 엔트리들은 파일에 대한 데이터 그룹들의 위치들을 지정한다.

<241> 파일 데이터는 파일의 종점에 도달할 때까지 또는 호스트가 또 다른 명령을 전송할 때까지, 하나의 메타페이지 단위들로 판독된다.

<242> 파일 데이터를 판독하는 예시적인 프로세스가 도6A에서 제공된다.

<243> 7. 파일 삭제

<244> 7.1 파일을 삭제하는 원리들

<245> 호스트로부터의 파일에 대한 삭제 명령에 응답하여, 파일에 대한 데이터를 포함하는 블록들이 식별되고, 후속 가비지 수집 작동들을 위해 가비지 수집 큐들에 추가된다. 파일을 삭제하는 절차는 이러한 가비지 수집 작동들을 개시하지 않으므로, 파일에 대한 데이터는 즉시 소거되지 않는다.

<246> 파일에 대한 FIT 엔트리들은 최초에 파일에 대한 데이터를 포함할 수 있는 공통 블록을 식별하기 위하여 평가된다. 그 후, 파일에 대한 FIT 엔트리들은 오프셋 어드레스 순서로 평가되고, 데이터 그룹들이 위치되는 데이터 블록들이 후속 가비지 수집을 위해, 공통 블록 큐 또는 쓸모없는 블록 큐 중 하나에 추가된다. 그 후, 파일에 대한 엔트리들을 제거하기 위하여, 파일 디렉토리 및 파일 인덱스 테이블이 갱신된다.

<247> 7.2 파일 소거

<248> 호스트로부터의 파일에 대한 소거 명령에 응답하여, 삭제 명령에 대한 것과 동일한 절차가 뒤따라야 하지만, 파일에 대한 데이터를 포함하는 블록들이 가비지 수집을 위해 우선순위 공통 블록 큐(priority common block queue) 및 우선순위 쓸모없는 블록 큐(priority obsolete block queue)에 추가된다.

<249> 그 후, 주요 장치 작동 시퀀스는 이러한 블록들에 대한 가비지 수집 작동들이 임의의 다른 호스트 명령이 실행되기 전에 수행되는 것을 보장한다. 이것은 소거 명령에 의해 식별된 파일에 대한 데이터가 즉시 소거되는 것을 보장한다.

<250> 파일 삭제 프로세스의 흐름도가 도7A에 도시되어 있다.

<251> 8. 가비지 수집

<252> 8.1 가비지 수집에 대한 원리들

<253> 가비지 수집은 쓸모없는 파일 데이터에 의해 차지된 플래시 메모리 용량을 복구하기 위하여 수행되어야 하는 작동이다. 가비지 수집은 파일의 삭제 또는 파일의 데이터에 대한 편집들의 결과로서 필요할 수 있다.

<254> 블록 통합은 관련되지 않은 파일들을 저장하는데 이들이 사용될 수 있도록 하기 위하여, 파일 데이터로 불완전하게 채워지는 블록들에서 소거된 용량을 복구하기 위하여 수행되는 가비지 수집의 형태이다. 통합은 프로그램 블록들을 공통 블록들로 변환하기 위하여 플레인 파일들에서의 프로그램 블록들 상에서 수행되거나, 이들의 수를 감소시키기 위하여 공통 블록들 상에서 수행될 수 있다.

<255> 가비지 수집 및 블록 통합의 프로세스들은 소스 블록이 소거되도록 하기 위하여, 파일-대-플래시 맵핑 알고리즘에 의해 언급된 바와 같이, 유효 파일 데이터를 소스 플래시 블록으로부터 하나 이상의 목적지 블록들로 재배치시킨다.

<256> 진행중인 가비지 수집 작동들은 즉시 수행되는 것이 아니라, 단계적 실행을 위한 스케줄링 알고리즘에 따라 수행된다. 가비지 수집을 필요로 하는 오브젝트들에 대한 엔트리들은 장치의 작동 동안 때때로 3개의 가비지 수집 큐들에 추가된다. 파일들, 쓸모없는 블록들, 및 공통 블록들을 위한 별도의 큐들이 존재한다. 오브젝트들은 미리 규정된 우선순위 순서로 다음 가비지 수집 작동을 위해 큐들로부터 선택된다. 큐들이 비어 있는 경우, 블록 통합이 수행될 수 있다.

<257> 가비지 수집 작동들은 2개의 방식으로 스케줄링될 수 있다. 백그라운드 작동들은 호스트가 장치로의 판독 또는 기록 액세스를 행하고 있지 않을 때 호스트에 의해 개시될 수 있고, 호스트가 또 다른 액세스를 행할 때까지 장치에 의해 지속적으로 실행된다. 포어그라운드 작동들은 장치가 호스트에 의해 액세스되고 있는 동안 장치에 의해 스케줄링될 수 있고, 호스트로부터 수신된 파일 데이터에 대한 프로그램 작동들의 버스트들과 인터리빙된 버스트들에서 실행된다. 인터리빙된 버스트들의 길이들은 가비지 수집 레이트를 항상 필요로 되는 최소값으로 유지하도록 적응형으로 제어될 수 있다.

<258> 8.2 가비지 수집 큐들

<259> 가비지 수집 큐들은 진행중인 가비지 수집 작동이 존재하는 오브젝트들에 대한 엔트리들을 포함한다. 3개의 큐들 각각은 쓸모없는 블록들, 공통 블록들 및 파일들 각각에 대한 엔트리들을 포함한다. 2개의 추가적인 큐들은 이러한 3개의 큐들보다 더 높은 우선순위를 제공받고, 쓸모없는 블록들 및 공통 블록들 각각에 대한 엔트리들을 포함한다. 5개의 가비지 수집 큐들이 플래시 메모리 내의 제어 블록에서 제어 로그에 저장된다.

<260> 8.2.1 우선순위 쓸모없는 블록 큐

<261> 이 큐는 호스트로부터의 소거 명령의 결과로서 완전히 쓸모없게 된 블록들에 대한 엔트리들을 포함한다. 이 큐는 가장 높은 우선순위 가비지 수집 큐이다. 큐에서 식별된 모든 블록들에 대한 가비지 수집 작동들은 임의의 다른 명령이 호스트로부터 수용되기 전에, 또는 가비지 수집 작동들이 임의의 다른 큐로부터의 오프젝트들에 대해 개시되기 전에 완료되어야 한다.

<262> 8.2.2 우선순위 공통 블록 큐

<263> 이 큐는 포스트로부터의 소거 명령의 결과로서 부분적으로 쓸모없게 된 공통 블록들에 대한 엔트리들을 포함한다. 이 큐는 두 번째로 높은 우선순위 가비지 수집 큐이다. 큐에서 식별된 모든 공통 블록들에 대한 가비지 수집 작동들은 임의의 다른 명령이 호스트로부터 수용되기 전에, 또는 가비지 수집 작동들이 더 낮은 우선순위 큐로부터의 오브젝트들에 대해 개시되기 전에 완료되어야 한다.

<264> 8.2.3 쓸모없는 블록 큐

<265> 이 큐는 호스트로부터의 삭제 명령, 또는 파일의 데이터에 대한 편집들의 결과로서 완전히 쓸모없게 된 블록들에 대한 엔트리들을 포함한다. 이 큐는 세번째로 높은 우선순위 가비지 수집 큐이다. 큐에서 식별된 모든 블록들에 대한 가비지 수집 작동들은 더 낮은 우선순위 큐로부터의 오브젝트들에 대해 작동들이 개시되기 전에 완료되어야 한다.

<266> 8.2.4 공통 블록 큐

<267> 이 큐는 호스트로부터의 삭제 명령, 또는 파일의 데이터에 대한 편집들의 결과로서 부분적으로 쓸모없게 된 공통 블록들에 대한 엔트리들을 포함한다. 이 큐는 네 번째로 높은 우선순위 가비지 수집 큐이다. 큐에서 식별된 모든 블록들에 대한 가비지 수집 작동들은 더 낮은 우선순위 큐로부터의 오브젝트들에 대해 작동들이 개시되기 전에 완료되어야 한다.

<268> 8.2.5 파일 큐

<269> 이 큐는 파일의 데이터의 편집들의 결과로서 쓸모없는 데이터를 갖는 파일들에 대한 엔트리들을 포함한다. 이 큐는 가장 낮은 우선순위 가비지 수집 큐이다. 파일이 호스트에 의해 클로우즈될 때, 이에 대한 엔트리는 파일이 플레인 파일이 아니라면, 파일 큐에 추가된다. 따라서, 파일이 플레인 파일인지 또는 편집된 파일(플레인 또는 공통)인지를 결정하기 위하여, 파일이 호스트에 의해 클로우즈되는 시간에 파일에 대한 FIT 엔트리들에 대한 분석을 수행하는 것이 필요하다.

<270> 이 분석에 대한 절차는 다음과 같다:

<271> 1. 관련된 FIT 파일에서의 FIT 엔트리들이 오프셋 어드레스 순서로 평가된다.

<272> 2. 데이터 그룹들 및 데이터 그룹 헤더들에 대한 누적 용량이 결정된다.

<273> 3. 파일 데이터에 의해 차지된 물리적 용량이 결정된다. 이것은 파일에 대한 데이터를 포함하는 프로그램 블록 이외의 블록들의 수, 더하기 프로그램 블록에서의 사용된 용량이다.

<274> 4. 데이터 그룹 용량이 물리적 용량의 X%를 초과하는 경우, 파일은 플레인 파일이다라고 결정된다. X의 값의 일례는 98%이다. X%는 버퍼 플러시 작동(buffer flush operation)에 기인한 프로그래밍되지 않은 공간이 플레인 파일에서 존속하도록 하기 위하여 100%보다 적다.

<275> 8.3 가비지 수집 작동들

<276> 8.3.1 쓸모없는 블록

<277> 쓸모없는 블록은 쓸모없는 데이터만을 포함하며, 데이터를 또 다른 블록으로 재배치할 필요 없이 소거될 수 있다.

<278> 8.3.2 공통 블록

<279> 공통 블록은 소스 블록이며, 하나 이상의 파일들에 대한 하나 이상의 부분적이거나 완전히 쓸모없는 데이터 그룹들을 포함한다. 유효 데이터는 이 소스 블록으로부터 하나 이상의 목적지 블록들로 재배치되어야 한다.

<280> 데이터는 완전한 파일 그룹의 단위들로 재배치되며, 여기서 파일 그룹은 공통 블록 내에서 동일한 파일들에 대한 하나 이상의 데이터 그룹들을 포함한다. 각각의 파일 그룹은 목적지 블록으로 그대로 재배치되지만, 상이한

파일 그룹들은 상이한 블록들에 재배치될 수 있다.

<281> 목적지 블록으로서 공통 블록이 우선적으로 선택되고, 적절한 공통 블록이 이용 가능하지 않은 경우 프로그램 블록이 그 다음에 선택되며, 적절한 프로그램 블록이 이용 가능하지 않은 경우 소거된 블록이 그 다음에 선택된다.

<282> 가비지 수집 작동은 다음 조건들: 작동이 종료되는 것, 호스트가 명령을 전송하는 것, 또는 포어그라운드 모드에서 인터리빙된 버스트의 종점에 도달하는 것 중 하나의 발생까지 지속될 수 있다.

<283> 공통 블록 가비지 수집 작동에 대한 흐름도가 도8E에 도시되어 있다.

<284> 8.3.3 파일 가비지 수집

<285> 파일 가비지 수집은 파일에 대한 쓸모없는 데이터에 의해 차지된 용량을 복구하기 위하여 수행된다. 파일 가비지 수집은 편집된 플레인 파일 상태 또는 편집된 공통 파일 상태의 파일을 플레인 파일 상태로 복구한다. 제1 단계는 쓸모없는 파일 블록들 및 데이터 그룹들이 가비지 수집 동안 카피되어야 하는 공통 블록을 식별하기 위하여, 이의 FIT 파일 내의 FIT 엔트리들에 대한 분석을 수행하는 것이다. 이 분석에 대한 절차는 다음과 같다:

<286> 1. 관련된 FIT 파일 내의 FIT 엔트리들은 오프셋 어드레스 순서로 평가된다.

<287> 2. 데이터 그룹 리스트는 데이터 그룹들을 물리적 블록들과 관련시키도록 구성된다. 프로그램 블록 내의 데이터 그룹들은 이 리스트로부터 배제된다.

<288> 3. 리스트에서 참조된 각각의 블록들에서 데이터 그룹들 및 데이터 그룹 헤더들에 의해 차지된 물리적 용량이 결정된다.

<289> 4. 데이터 그룹 용량이 블록의 용량의 X%를 초과하는 경우, 블록은 파일 블록이다라고 결정된다. X의 값의 일례는 98%이다. X%는 버퍼 플러시 작동에 기인한 프로그래밍되지 않은 공간이 파일 블록에서 존속하도록 하기 위하여 100%보다 적다.

<290> 5. 파일 블록들인 것으로 결정되는 블록들 내의 데이터 그룹들이 상기 구성된 데이터 그룹 리스트로부터 제거된다.

<291> 개정된 데이터 그룹 리스트에서 참조된 데이터 그룹들은 쓸모없는 파일 블록들 또는 공통 블록에 포함되고, 파일 가비지 수집 작동 동안 프로그램 블록으로 카피된다. 파일의 데이터 그룹 구조는 파일 가비지 수집 작동의 결과로서 변경될 수 있는데, 즉, 재배치된 데이터 그룹이 블록 경계에 의해 둘로 나누어지거나, 인접한 데이터 그룹과 병합될 수 있다. 파일에 대한 프로그램 블록은 목적지 블록으로서 사용된다. 이 프로그램 블록이 채워질 때, 또 다른 프로그램 블록이 오픈된다.

<292> 가비지 수집 작동은 다음 조건들: 작동이 종료되는 것, 호스트가 명령을 전송하는 것, 또는 포어그라운드 모드에서 인터리빙된 버스트의 종점에 도달하는 것 중 하나의 발생까지 지속될 수 있다.

<293> 파일 가비지 수집 작동에 대한 흐름도가 도8D에 도시되어 있다.

<294> 8.3.4 블록 통합

<295> 블록 통합은 불완전하게 프로그래밍되었던 공통 블록들 및 프로그램 블록들에서 소거된 용량을 복구하고 상기 용량을 다른 파일들에 대한 데이터를 저장하는데 이용 가능하도록 하기 위하여 수행된다.

<296> 통합에 대한 소스 블록은 블록이 가능한 최소의 데이터 재배치 이후에 소거되도록 하기 위하여, 최저 프로그래밍된 용량을 갖는 공통 블록 로그 또는 프로그램 블록 로그 내의 블록으로서 선택된다. 데이터는 완전한 파일 그룹의 단위들로 재배치되고, 여기서 파일 그룹은 프로그램 블록 또는 공통 블록 내에서 동일한 파일들에 대한 하나 이상의 데이터 그룹들을 포함한다. 각각의 파일 그룹은 목적지 블록에 그대로 재배치되지만, 상이한 파일 그룹들은 상이한 블록들에 재배치될 수 있다.

<297> 목적지 블록으로서 공통 블록이 우선적으로 선택되며, 적절한 공통 블록이 이용 가능하지 않은 경우 프로그램 블록이 그 다음에 선택된다. 목적지 블록이 이용 가능하지 않은 드문 경우에, 파일 그룹은 단일 이상의 목적지 블록에 재배치되도록 분할될 수 있다.

<298> 블록 통합 작동은 다음 조건들: 작동이 종료되는 것, 호스트가 명령을 전송하는 것, 또는 포어그라운드 모드에서 인터리빙된 버스트의 종점에 도달하는 것 중 하나의 발생까지 지속될 수 있다.

<299> 블록 통합 작동에 대한 흐름도가 도8F에 도시되어 있다.

<300> 8.4 가비지 수집 작동들의 스케줄링

<301> 가비지 수집은 바람직하게는 호스트 장치가 카드에 액세스하고 있는 기간들 동안 백그라운드 태스크로서 수행된다. 호스트에 의해 개시되는 백그라운드 가비지 수집은 직접적인 데이터 파일 플랫폼에서 지원된다.

<302> 그러나, 호스트가 장치에 데이터를 기록하고 있는 동안, 포어그라운드 태스크로서 가비지 수집을 수행하는 것이 또한 필요할 수 있다. 이 모드에서, 완전한 가비지 수집 작동은 단일 이벤트로서 완료될 필요가 없다. 가비지 수집의 버스트들은 호스트로부터의 데이터를 프로그래밍하는 버스트들과 인터리빙되어, 가비지 수집 작동이 다수의 별도의 단계들에서 종료될 수 있도록 할 수 있고, 호스트에 대한 장치의 유용성에 대해 제한된 인터럽션(interruption)이 존재하도록 한다.

<303> 8.4.1 백그라운드 작동

<304> 백그라운드 가비지 수집은 호스트가 장치에 유휴 명령을 전송할 때 개시된다. 이것은 호스트가 장치로부터 고의로 전력을 제거하지 않을 것이며, 즉시 장치에 액세스하고자 하지는 않는다는 것을 의미한다. 그러나, 호스트는 또 다른 명령을 전송함으로써 언제라도 유휴 상태를 종료시킬 수 있다.

<305> 유휴 상태에서, 장치는 다음 조건들: 호스트가 또 다른 명령을 전송하는 것, 또는 모든 가비지 수집 큐들이 비어 있는 것 및 블록 통합 작동들이 불가능한 것 중 하나의 발생까지 연속적인 가비지 수집 작동들을 수행한다.

<306> 8.4.2 인터리빙된 작동

<307> 인터리빙된 가비지 수집 작동들은 파일 데이터를 프로그래밍하는 직접적인 데이터 파일 프로세스에 의해 개시된다. 인터리빙된 작동은 도8A에 도시되어 있다. 호스트 인터페이스가 활성이고 플래시 메모리에 대해 N1 프로그램 작동들이 행해지는 호스트 데이터 기록 단계 이후에, 장치는 가비지 수집 단계로 전환한다. 이 단계에서, 하나 이상의 가비지 수집 작동들 중 일부가 플래시 메모리에 대한 N2 프로그램 작동이 완료될 때까지 수행된다.

<308> 진행중인 가비지 수집 작동은 가비지 수집 단계의 끝까지 중지되고, 다음의 이와 같은 단계에서 재개될 수 있다. N1 및 N2의 값들은 적응형 스케줄링 알고리즘에 의해 결정된다.

<309> 8.4.3 적응형 스케줄링

<310> 적응형 스케줄링 방법은 가비지 수집 및 호스트 데이터 프로그래밍의 인터리빙된 버스트들의 상대적인 길이들을 제어하여, 가비지 수집 작동들에 의한 호스트 데이터 기록 작동들에 대한 인터럽션이 최소로 유지될 수 있도록 하는데 사용된다. 이것은 나중에 성능의 감소를 초래할 수 있는 진행중인 가비지 수집의 백로그가 구성되지 않는다는 것을 보장하면서 성취된다.

<311> 8.4.3.1 작동의 원리

<312> 언제라도, 장치 상태는 이전에 기록된 호스트 데이터에 의해 차지된 용량, 소거된 블록 풀 내의 블록들에서의 소거된 용량, 및 가비지 수집 작동들에 의해 추가적인 호스트 데이터를 기록하는데 이용 가능해질 수 있는 복구 가능한 용량을 포함한다. 이 복구 가능한 용량은 이전에 기록된 호스트 데이터와 공유된 프로그램 블록들, 공통 블록들, 또는 쓸모없는 파일 블록들, 또는 완전히 쓸모없는 블록들에서 존재할 수 있다. 용량 이용의 이러한 유형들이 도8B에 도시되어 있다.

<313> 가비지 수집의 적응형 스케줄링은 증분 호스트 데이터를 프로그래밍하고 이전에 기록된 호스트 데이터를 재배치하는 인터리브 비율을 제어하여, 상기 비율이 모든 복구 가능한 용량이 호스트 데이터에 이용 가능해질 수 있는 적응형 기간에 걸쳐 일정하게 유지되도록 할 수 있다. 호스트가 파일들을 삭제하는 경우(이는 이전에 기록된 호스트 데이터를 복구 가능한 용량으로 변환한다), 인터리브 비율은 이에 따라 변화되고, 새로운 적응형 기간이 시작된다.

<314> 최적의 인터리브 비율은 다음과 같이 결정될 수 있다:

<315> If

<316> 소거된 블록 풀 내의 소거된 블록의 수 = *소거된_블록들*;

<317> 프로그램 및 공통 블록들의 결합된 수 = *데이터_블록들*;

<318> 프로그램 및 공통 블록들 내의 유효 데이터 페이지들의 총 수 = *데이터_페이지들*;

<319> 쓸모없는 블록들의 수 = *쓸모없는_블록들*; 및

<320> 블록 내의 페이지들의 수 = *블록_당_페이지들*,

<321> Then

<322> 가비지 수집에 의해 생성될 수 있는 소거된 블록들의 수가 *데이터_블록들* - (*데이터_페이지들*/*블록_당_페이지들*)에 의해 제공된다;

<323> 가비지 수집 이후에 소거된 블록의 총수는 *소거된_블록들* + *쓸모없는_블록들* + *데이터_블록들* - (*데이터_페이지들*/*블록_당_페이지들*)에 의해 제공된다;

<324> 기록될 수 있는 증분 데이터 메타페이지들의 수는 *블록_당_페이지들* * (*소거된_블록들* + *쓸모없는_블록들* + *데이터_블록들*) - *데이터_페이지들*에 의해 제공된다.

<325> If

<326> 유효 데이터가 프로그램 및 공통 블록들에 걸쳐 고르게 분포된다라고 가정된다(낮은 데이터 페이지 카운트들을 갖는 블록들이 블록 통합 작동들에 대한 소스 블록으로서 선택되기 때문에, 비관적인 가정)

<327> Then

<328> 가비지 수집 동안 재배치된 메타페이지들의 수는 *데이터_페이지들* * (*데이터_블록들* - *데이터_페이지들*/*블록_당_페이지들*)/*데이터_블록들*에 의해 제공된다.

<329> 최적의 인터리브 비율(N1:N2)은 가비지 수집 동안 재배치되어야 하는 메타페이지들의 수에 기록될 수 있는 증분 데이터 메타페이지들의 수의 비율이다. 따라서,

<330> N1:N2 = (*블록_당_페이지들* * (*소거된_블록들* + *쓸모없는_블록들* + *데이터_블록들*) - *데이터_페이지들*)/(*데이터_페이지들* * (*데이터_블록들* - *데이터_페이지*들/*블록_당_페이지들*)/*데이터_블록들*)

<331> 쓸모없는 파일 블록들에서의 쓸모없는 용량의 복구가 적응형 스케줄링 알고리즘에 포함되지 않았다는 점에 주의하라. 이와 같은 용량은 단지 파일들의 편집에 기인하며, 통상적인 사항이 아니다. 쓸모없는 파일 블록들에서 상당한 용량이 존재하는 경우, 적응형으로 결정된 인터리브 비율은 최적이 아닐 수 있지만, (8.4.3.2에 설명된) 최소 비율을 갖는 작동으로의 스위칭이 이와 같은 블록들의 효율적인 가비지 수집을 보장할 것이다.

<332> 8.4.3.2 인터리브 제어

<333> 인터리브 비율(N1:N2)은 다음과 같이 3개의 대역들에서 규정된다.

<334> 1) 최대: 인터리브 비율에 대한 최대 한도는 가비지 수집이 결코 완전히 금지되지는 않도록 하기 위하여 설정된다. 이 최대 한도에 대한 일례는 10:1이다.

<335> 2) 적응형: 적응형 대역에서, 인터리브 비율은 공통 블록들 및 쓸모없는 블록들의 진행중인 가비지 수집, 및 프로그램 블록들 및 공통 블록들의 통합에 대해 최적이도록 제어된다. 이것은 다음 관계에 의해 규정되며,

<336> N1:N2 = (*블록_당_페이지들* * (*소거된_블록들* + *쓸모없는_블록들* + *데이터_블록들*) - *데이터_페이지들*)/(*데이터_페이지들* * (*데이터_블록들* - *데이터_페이지*들/*블록_당_페이지들*)/*데이터_블록들*)

<337> 여기서 N1 및 N2는 페이지 프로그램 작동들의 수로서 규정된다. 가비지 수집의 버스트의 바람직한 지속기간을 나타내는 N2의 값이 규정된다. 이 값의 일례는 16이다.

<338> 3) 최소: 소거된 블록 풀 내의 블록들의 수가 규정된 최소값 아래로 떨어지는 경우, 인터리브 비율은 적응형으로 규정되는 것이 아니라, 고정된 최소 한도로 설정된다. 이 최소 한도의 일례는 1:10이다.

<339> 8.4.3.3 제어 파라미터들

<340> 다음 파라미터들은 적응형 스케줄링의 제어를 위하여, 플래시 메모리 내의 제어 블록에서의 제어 로그에서 유지된다.

<341> *소거된 블록 카운트*: 소거된 블록 풀 내의 블록들의 수의 카운트가 유지된다. 이것은 블록들이 소거된 블록 풀로 추가되고 상기 소거된 블록 풀로부터 제거될 때 갱신된다.

<342> *프로그램 및 공통 블록 카운트*: 프로그램 블록들 및 공통 블록들의 결합된 수의 카운트가 유지된다. 공통 블록들은 쓸모없는 데이터를 포함할 수 있다. 상기 카운트는 블록들이 프로그램 블록 로그 및 공통 블록 로그에 추가되고 상기 프로그램 블록 로그 및 공통 블록 로그로부터 제거될 때 갱신된다.

<343> *프로그램 및 공통 블록 페이지 카운트*: 프로그램 블록들 및 공통 블록들 내의 유효 데이터 페이지들의 수의 카운트가 유지된다. 상기 카운트는 블록들이 프로그램 블록 로그 및 공통 블록 로그에 추가되고 상기 프로그램 블록 로그 및 공통 블록 로그로부터 제거될 때 갱신된다.

<344> *쓸모없는 블록 카운트*: 가비지 수집을 대기하는 완전히 쓸모없는 블록들의 수의 카운트가 유지된다. 상기 카운트는 블록들이 쓸모없는 블록 가비지 수집 큐에 추가되고 상기 쓸모없는 블록 가비지 큐로부터 제거될 때 갱신된다.

<345> 8.5 가비지 수집에 대한 흐름도들

<346> 여러 특정 가비지 수집 작동들 중 하나를 선택하는 특정 알고리즘의 흐름도가 도8C에 제공되어 있다. 도8D는 도8C의 "파일 가비지 수집" 블록에 대한 흐름도이다. 도8C의 "공통 블록 가비지 수집" 블록에 대한 흐름도는 도8E의 주제이다. 도8C의 "블록 통합" 기능은 도8F의 흐름도에 의해 도시되어 있다.

<347> 4개의 도면들(8G1 내지 8G4)은 도8E의 프로세스에 기인할 수 있는 공통 블록의 예시적인 가비지 수집을 도시한다. 도8G1은 최초 조건을 도시하는 반면, 도8G2 내지 8G3는 가비지 수집 프로세스에서의 3개의 단계들을 도시한다. 화살표들은 쓸모없는 블록들로부터 가득 차지 않은 파일 블록들 내로의 유효 데이터의 전달을 도시하며, 이러한 목적지 파일 블록들은 그 후에 공통 블록들이 된다.

<348> 9. 데이터 버퍼링 및 프로그래밍

<349> 이 단락에서 설명된 데이터 버퍼링 및 프로그래밍 방법은 현재 제품들에서 사용되는 동일한 플래시 인터페이스 및 에러 수정 코드(ECC) 구조들을 사용하도록 억제된다. 새로운 플래시 인터페이스 및 ECC 구조들이 소개되면, 대안적인 최적화된 방법들이 미래에 채용될 수 있다.

<350> 9.1 데이터 버퍼들

<351> 버퍼 메모리는 플래시 메모리로 프로그래밍되고 플래시 메모리로부터 판독되는 데이터의 일시적인 저장을 위해, 제어기 내의 SRAM(종래 애플리케이션들의 RAM 31)에 존재한다. 버퍼 메모리의 할당된 영역은 전체 메타페이지가 플래시 메모리에서의 단일 작동으로 프로그래밍되도록 파일에 대한 충분한 데이터를 축적하는데 사용된다. 버퍼 메모리에서의 파일에 대한 데이터의 오프셋 어드레스는 중요하지 않다. 버퍼 메모리는 다수의 파일들에 대한 데이터를 저장할 수 있다.

<352> 호스트에 의한 기록 및 판독 작동들 둘 모두의 파이프라이닝(pipelining)을 허용하기 위하여, 2개의 메타페이지들의 용량을 갖는 버퍼 메모리 공간은 버퍼링되는 각각의 파일에 대해 이용 가능해야 한다. 버퍼 메모리는 한 세트의 섹터 버퍼들을 포함한다. 개별적인 섹터 버퍼들은 단일 파일에 대한 데이터의 일시적인 저장을 위해 할당될 수 있고, 데이터가 자신의 최종 목적지에 전달될 때, 할당해제될 수 있다. 섹터 버퍼들은 섹터 버퍼 번호(0 내지 N-1)에 의해 식별된다. 섹터 버퍼들의 수(N)의 일례는 64이다.

<353> 이용 가능한 섹터 버퍼들은 자신의 섹터 버퍼 번호 순으로 순환적으로 할당된다. 각각의 섹터 버퍼는 파일 라벨 및 자신의 포함된 데이터의 시작점 및 종점을 규정하는 2개의 관련 포인터들을 갖는다. 섹터 버퍼 내의 데이터에서의 파일 오프셋 어드레스 범위들이 또한 기록된다. 섹터 버퍼들 및 이들과 관련된 제어 정보 둘 모두는 제어기 내의 휘발성 메모리 내에서만 존재한다.

<354> 9.2 데이터 프로세싱

<355> 9.2.1 메타페이지

<356> 메타페이지는 플래시 메모리에서 프로그래밍의 최대 단위이다. 데이터는 최대 성능을 위하여, 가능할 때마다 메타페이지의 단위들로 프로그래밍되어야 한다.

<357> 9.2.2 페이지

<358> 페이지는 메타페이지의 서브셋이며, 플래시 메모리에서 프로그래밍의 최소 단위이다.

<359> 9.2.3 섹터

<360> 섹터는 제어기 및 플래시 메모리 사이에서 데이터 전달의 최소 단위이다. 섹터는 통상적으로 512 바이트의 파일 데이터를 포함한다. ECC는 (종래 애플리케이션들의 도2의 제어기 ECC 회로(33)와 같은) 각각의 섹터에 대한 제어기에 의해 발생되고, 섹터의 종점에 첨부된 플래시에 전달된다. 데이터가 플래시로부터 판독될 때, 데이터는 ECC가 검사되도록 하기 위하여, 다수의 완전한 섹터로 제어기에 전달되어야 한다.

<361> 9.3 버퍼 플러시

<362> 파일에 대한 데이터는 통상적으로 충분한 데이터가 플래시 메모리에서 완전한 메타페이지를 프로그래밍하는데 이용 가능할 때까지 섹터 버퍼들에서 축적된다. 호스트가 파일에 대한 데이터를 스트리밍하는 것을 중단할 때, 하나 이상의 섹터 버퍼들은 메타페이지의 부분에 대한 파일 데이터와 함께 유지된다. 이 데이터는 호스트가 파일에 대한 추가적인 데이터를 기록하도록 하기 위하여 버퍼 메모리에서 유지된다. 그러나, 어떤 상황들 하에서, 버퍼 메모리 내의 데이터는 버퍼 플러시로서 공지된 작동에서 플래시 메모리로 커미팅(committing)되어야 한다. 버퍼 플러시 작동은 섹터 버퍼들에 수용되는 파일에 대한 모든 데이터가 메타페이지 내의 하나 이상의 페이지들에서 프로그래밍되도록 한다.

<363> 버퍼 플러시 작동은 다음 2가지 이벤트들에서 수행된다:

<364> 1) 파일이 호스트에 의해 클로우즈되는 이벤트, 또는

<365> 2) 셧-다운 명령이 호스트에 의해 수신되는 이벤트

<366> 호스트에 의해 클로우즈되는 파일에 대한 데이터가 스왑 블록으로 스왑-아웃되었다면, 상기 데이터는 버퍼 메모리로 복구되어야 하고, 버퍼 플러시가 수행되어야 한다. 전력 제거 다음에 장치의 초기화 동안 스왑 블록 내에 있는 데이터는 버퍼 메모리로 복구되어야 하고, 버퍼 플러시가 수행되어야 한다.

<367> 9.4 버퍼 스왑

<368> 버퍼 스왑은 호스트가 파일에 대한 데이터를 지속적으로 기록할 때 하나 이상의 섹터 버퍼들 내의 파일에 대한 데이터가 나중에 버퍼 메모리로 복구되도록 하기 위하여 스왑 블록으로서 인식된 일시적인 위치에서 프로그래밍되는 작동이다.

<369> 9.4.1 스왑 블록의 포맷

<370> 스왑 블록은 섹터 버퍼들로부터 스왑-아웃되었던 파일들에 대한 데이터를 저장하는 전용 블록이다. 파일에 대한 데이터는 스왑 블록 내의 그 파일에 전용된 하나 이상의 페이지들에서 인접하게 저장된다. 데이터가 나중에 다시 버퍼 메모리로 스왑-인될 때, 상기 데이터는 스왑 블록에서 쓸모없어진다.

<371> 스왑 블록이 가득 차게 될 때, 상기 블록 내의 유효 데이터는 나중에 스왑 블록이 되는 소거된 블록으로 압축된 형태로 기록된다. 이 압축된 형태는 상이한 파일들에 대한 데이터가 동일한 페이지 내에서 존재하도록 한다. 바람직하게는, 단일 스왑 블록만이 존재한다.

<372> 9.4.2 스왑 블록에서 저장된 데이터 인덱싱

<373> 스왑 블록 내의 각각의 파일에 대해 버퍼 메모리 내의 파일에 대해 이전에 기록된 정보의 카피를 포함하는 스왑 블록 인덱스가 플래시 메모리에서 유지된다(9.1 참조).

<374> 9.4.3 스왑 블록으로 데이터 이동(스왑-아웃)

<375> 스왑-아웃 작동은 불충분한 섹터 버퍼들이 호스트에 의해 오픈되었던 파일, 또는 호스트로부터의 파일에 대한 기록 명령의 결과로서 스왑 블록으로부터 스왑-인되어야 하는 파일에 할당되는데 이용 가능할 때, 발생한다. 스왑-아웃을 위해 선택된 파일은 버퍼들이 버퍼 메모리에 존재하는 것들 중 가장 적은 최근에 기록된 파일이어야 한다.

<376> 선택적으로, 전력의 스케줄링되지 않은 제거의 이벤트에서 데이터의 안전성을 개선시키기 위하여, 호스트로부터의 가장 최근의 기록 명령과 관련되지 않은 버퍼 메모리 내의 임의의 파일에 대하여 스왑-아웃이 수행될 수 있다. 이 경우에, 파일에 대한 데이터는 버퍼 메모리에서 유지될 수 있고, 후속 스왑-인 작동은 전력의 제거가 존재하지 않는다면 필요로 되지 않는다.

<377> 9.4.4 스왑 블록으로부터의 데이터 복구(스왑-인)

<378> 파일에 대한 완전한 데이터는 스왑 블록으로부터 하나 이상의 섹터 버퍼들로 판독된다. 파일 데이터는 자신이

스왑-아웃되기 이전과 같이, 섹터 버퍼와 정확히 동일하게 정렬될 필요가 없다. 정렬은 스왑 블록의 압축(compaction)의 결과로서 변화될 수 있다. 스왑 블록에 있는 파일에 대한 데이터는 쓸모없어진다.

<379> 9.5 호스트로부터의 파일 데이터 프로그래밍

<380> 이 단락에서 제공된 예들은 메타페이지 당 2개의 페이지들, 및 페이지당 2개의 섹터들을 갖는 플래시 메모리 구성과 관련된다.

<381> 9.5.1 호스트로부터의 연속적인 데이터 프로그래밍

<382> 파일에 대한 데이터는 호스트로부터 스트리밍되고, 연속적으로 할당된 섹터 버퍼들에서 축적된다. 충분한 섹터 버퍼들이 채워졌을 때, 이들의 데이터는 각각의 섹터에 대한 ECC와 함께 플래시 메모리에 전달되고, 파일에 대한 프로그램 블록 내의 목적지 메타페이지가 프로그래밍된다. 연속적인 호스트 데이터 프로그래밍의 일례가 도9A에 도시되어 있다.

<383> 9.5.2 호스트로부터의 인터럽트된 데이터 프로그래밍

<384> 파일에 대한 데이터는 호스트로부터 스트리밍되고, 연속적으로 할당된 섹터 버퍼들에서 축적된다. 도9B는 인터럽트되었던 호스트 데이터 프로그래밍의 일례를 도시한다. 스트림은 데이터 세그먼트(2A) 후에 인터럽트되는 반면, 상이한 파일에 대한 기록 작동이 실행된다. 파일에 대한 추가적인 기록 명령이 수신될 때, 데이터 세그먼트(2B)에서 시작하는, 호스트로부터 스트리밍된 데이터는 이전과 동일한 섹터 버퍼에서 축적된다. 충분한 섹터 버퍼들이 채워졌을 때, 이들의 데이터는 각각의 섹터에 대한 ECC와 함께 플래시 메모리에 전달되고, 파일에 대한 프로그램 블록 내의 목적지 메타페이지가 프로그래밍된다.

<385> 9.5.3 버퍼로부터 플러시되는 데이터 프로그래밍

<386> 파일에 대한 데이터는 호스트로부터 스트리밍되고, 연속적으로 할당된 섹터 버퍼들에서 축적된다. 그러나, 완전한 메타페이지에서 프로그래밍되는데 충분하지 않은 데이터가 존재한다. 일례가 도9C에서 제공된다. 패딩 세그먼트(2B)와 함께, 데이터 세그먼트들(1 및 2A)은 각각의 섹터에 대한 ECC와 함께 플래시 메모리에 전달되며, 파일에 대한 프로그램 블록 내의 목적지 페이지가 프로그래밍된다.

<387> 9.5.4 버퍼로부터 스왑-아웃되는 데이터 프로그래밍

<388> 이 작동은 목적지 페이지가 파일에 대한 프로그램 블록 대신에, 스왑 블록에서 다음에 이용 가능하다는 것을 제외하면, 버퍼 플러시 프로그래밍에 대한 것과 동일하다. 도9D는 이것을 도시한다.

<389> 9.5.5 버퍼로부터의 플러시 다음의 호스트로부터의 데이터 프로그래밍

<390> 파일에 대한 버퍼 플러시 작동 다음에 호스트에 의해 공급되는 파일 데이터는 버퍼 메모리로부터 플러시되는 데이터와 별도로 프로그래밍된다. 따라서, 프로그래밍은 파일에 대한 프로그램 블록 내의 다음에 이용 가능한 페이지에서 시작되어야 한다. 현재 메타페이지를 완성시키는데 충분한 데이터가 축적되고, 이 데이터는 ECC와 함께 전달되며, 도9E의 섹터들(3 및 4)에 대해 도시된 바와 같이 프로그래밍된다.

<391> 9.5.6 버퍼로의 스왑-인 다음의 호스트로부터의 데이터 프로그래밍

<392> 버퍼 메모리로부터 스왑-아웃되었던 파일에 대한 추가적인 데이터가 수신될 때, 이 추가적인 데이터는 버퍼 메모리에서 할당된 섹터 버퍼들에 축적된다. 스왑-아웃된 데이터는 또한 스왑 블록으로부터 버퍼 메모리로 복구된다. 충분한 데이터가 축적될 때, 전체 메타페이지가 단일 작동으로 프로그래밍된다.

<393> 도9F에 도시된 바와 같이, 패딩 세그먼트(2B)와 함께, 데이터 세그먼트들(1 및 2A)는 스왑 블록으로부터 판독되고, 2개의 섹터 버퍼들에서 복구된다. ECC가 섹터들 둘 모두에 대해 검사된다.

<394> 도9G의 예에서 도시된 바와 같이, 호스트로부터의 파일 데이터는 버퍼 섹터들에서 축적된다. 데이터 세그먼트들(1, 2A/2B, 3A/3B 및 4A/4B)은 각 섹터에 대한 ECC와 함께 플래시 메모리에 전달되고, 섹터들(1, 2, 3 및 4)로서 프로그래밍된다.

<395> 9.6 플래시로부터 카피된 데이터 프로그래밍

<396> 9.6.1 정렬된 메타페이지로부터의 데이터 카피

<397> 소스 및 목적지 메타페이지들은 도9H에 도시된 바와 같이, 전체 목적지 메테페이지에 카피될 데이터가 단일의 전체 소스 메타페이지를 차지할 때 정렬된다고 한다. 데이터 섹터들(1, 2, 3 및 4)은 소스 메타페이지로부터 4

개의 섹터 퍼버들로 판독되고, 각각의 섹터에 대하여 ECC가 검사된다.

<398> 9.6.1.1 버퍼로부터의 라이트-백(Write-back)

<399> 데이터는 도9I에 도시된 바와 같이, 4개의 섹터 버퍼로부터 목적지 메타페이지로 프로그래밍된다. 섹터들(1, 2, 3 및 4)에 대하여 ECC가 발생되고 저장된다.

<400> 9.6.1.2 온-칩 카피

<401> 카피될 데이터의 메타페이지 정렬이 소스 및 목적지 메타페이지들에서 동일할 때, 카피 작동의 속도를 증가시키기 위하여 플래시 칩 내의 온-칩 카피가 사용될 수 있다. 데이터는 ECC 검사가 에러가 없다는 것을 나타내는 경우에 목적지 메타페이지로 프로그래밍된다.

<402> 9.6.1.3 공통 블록 내에서의 메타페이지 정렬

<403> 공통 블록 내에서의 각각의 파일 그룹의 시작점은 메타페이지의 시작점과 정렬되어야 한다. 프로그램 블록 내의 데이터 그룹들은 또한 블록 내에 제1 메타페이지의 시작점과 정렬된다. 따라서, 프로그램 블록들을 공통 블록 내로 통합하는 것 및 파일 그룹들을 하나의 공통 블록으로부터 프로그램 블록 또는 또 다른 공통 블록으로 카피하는 것과 같은 공통 블록에 대한 모든 데이터 카피 작동들은 정렬된 메타페이지들 사이의 데이터 카피와 함께 작동할 것이다. 플래시 칩 내의 온-칩 카피는 카피 작동의 속도를 증가시키기 위하여 데이터를 공통 블록에 또는 공통 블록으로부터 카피할 때 사용되어야 한다.

<404> 9.6.2 비-정렬된 순차적인 메타페이지들로부터의 데이터 카피

<405> 소스 및 목적지 메타페이지들은 전체 목적 메타페이지에 카피될 데이터가 인접하지만, 2개의 순차적인 소스 메타페이지들을 차지할 때 비-정렬된다고 한다. 소스 메타페이지를 판독하는 일례가 도9J에 도시되어 있다. 데이터 섹터들(1A/1B, 2, 및 3)은 제1 소스 메타페이지로부터 3개의 섹터 버퍼들로 판독되고, 데이터 섹터들(4 및 5A/5B)은 제2 소스 메타페이지로부터 추가적인 2개의 섹터 버퍼들로 판독된다. 각각의 섹터에 대하여 ECC가 검사된다.

<406> 데이터 부분들(1A/1B, 2A/2B, 3A/3B, 및 4A/4B)은 도9K에 도시된 바와 같이 섹터 버퍼들로부터 목적지 메타페이지 내의 섹터들(1, 2, 3 및 4)로 프로그래밍된다. 섹터들(1, 2, 3 및 4)에 대하여 ECC가 발생되고 저장된다.

<407> 카피되는 메타페이지 데이터가 연속적인 데이터의 보다 큰 런(run)의 부분일 때, 카피는 부분적으로 파이프라이닝될 수 있다. 데이터는 소스 위치로부터 전체 메타페이지들 내의 버퍼 메모리로 판독된다. N개의 목적지 메타페이지들을 프로그래밍하기 위하여 N+1개의 소스 메타페이지들이 판독되어야 한다.

<408> 9.6.3 비-정렬된 비-순차적인 메타페이지들로부터의 데이터 카피

<409> 소스 및 목적지 메타페이지들은 전체 목적지 메타페이지에 카피될 데이터가 인접하지 않고, 2개 이상의 비-순차적인 소스 메타페이지들을 차지할 때 비-정렬되고 비-순차적이라고 한다. 이 경우는 파일 내의 2개 이상의 비-순차적인 데이터 그룹들 중 일부를 단일 목적지 메타페이지에 카피하는 것을 나타낸다. 도9L에 도시된 바와 같이, 데이터 섹터들(1A/1B, 2, 및 3A/3B)은 제1 소스 메타페이지로부터 3개의 섹터 버퍼들로 판독되고, 데이터 섹터들(4A/4B 및 5A/5B)은 제2 소스 메타페이지로부터 추가적인 2개의 섹터 버퍼들로 판독된다. 각각의 섹터에 대해 ECC가 검사된다.

<410> 그 후, 데이터 부분들(1A/1B, 2A/2B, 3A/3B, 및 4A/4B)이 도9M에 도시된 바와 같이, 섹터 버퍼들로부터 목적지 메타페이지 내의 섹터들(1, 2, 3 및 4)로 프로그래밍된다. 섹터들(1, 2, 3 및 4)에 대해 ECC가 발생되고 저장된다.

<411> 10. 파일 인덱싱

<412> 10.1 파일 인덱싱의 원리들

<413> 파일 인덱싱은 도10A에 일반적으로 도시되어 있다. 파일에 대한 데이터는 파일 오프셋 어드레스 공간 및 물리적 어드레스 공간 둘 모두에서 인접한 어드레스들의 런을 각각 스패닝(spanning)하는, 데이터 그룹들의 세트로서 저장된다. 파일에 대한 세트 내의 데이터 그룹들은 서로 임의의 특정한 물리적 어드레스 관계를 가질 필요가 없다. 파일 인덱스 테이블(FIT)은 파일에 대한 유효 데이터 그룹들의 위치들이 오프셋 어드레스 순서로 식별되도록 한다. 파일에 대한 FIT 엔트리들의 세트는 파일 데이터 포인터에 의해 식별된다.

<414> 호스트에 의해 발생되는 파일과 관련된 정보는 정보 테이블(IT) 내에 파일_정보로서 저장된다. 파일_정보의 특

성 및 내용은 호스트에 의해 결정되며, 장치에 의해 해석되지 않는다. 파일_정보는 파일이름, 페어런트 디렉토리, 차일드 드렉토리들, 속성들, 권리 정보, 및 파일에 대한 파일 어소시에이션(file association)을 포함할 수 있다. IT 내의 파일에 대한 파일_정보는 파일 정보 포인터에 의해 식별된다.

<415> 디렉토리는 장치 내의 모든 유효 파일에 대한 파일 데이터 포인터 및 파일 정보 포인터를 포함한다. 파일에 대한 이러한 디렉토리 엔트리들은 수치적인 값이 파일ID에 의해 식별된다.

<416> 10.2 파일 인덱싱 구조들

<417> 도10B는 파일 인덱싱 구조들의 일례를 도시한다.

<418> 10.3 디렉토리

<419> 10.3.1 파일ID

<420> 파일ID는 직접적인 데이터 파일 플랫폼 내에 존재하는 파일에 대한 수치적인 식별자이다. 이 파일 ID는 생성 명령에 응답하여 직접적인 데이터 파일 플랫폼에 의해 할당되고, 생성 명령을 갖는 파라미터로서 지정될 수 있다.

<421> 파일ID 값이 장치에 의해 할당될 때, 디렉토리 내의 엔트리들에 대한 순환적인 포인터가 다음에 이용 가능한 파일ID를 위치시키기 위해 사용된다. 파일이 삭제되거나 소거될 때, 파일의 파일ID에 의해 식별된 디렉토리 엔트리는 이용 가능한 것으로 마킹된다.

<422> 파일ID 값은 파일에 대한 파일 데이터 포인터 및 파일 정보 포인터를 위한 필드들을 포함하는 디렉토리 내의 엔트리를 규정한다.

<423> 장치에 저장될 수 있는 파일들의 최대 수는 파일ID에 대해 할당된 비트들의 수에 의해 결정된다.

<424> 10.3.2 파일 데이터 포인터

<425> 파일 데이터 포인터는 제어 로그 내에서, FIT 블록 리스트, 및 아마도 또한 FIT 갱신 블록 리스트 내의 파일에 대한 엔트리에 대한 논리적 포인터이다.

<426> 파일 데이터 포인터는 2개의 필드들을 갖는다:

<427> 1) FIT 범위, 및

<428> 2) FIT 파일 번호

<429> 파일에 대한 파일 데이터 포인터는 파일이 제로의 길이를 가질 때에도 존재한다.

<430> 10.3.2.1 FIT 범위

<431> FIT 범위는 FIT의 서브셋이다. 각각의 FIT 범위는 별도의 물리적 FIT 블록에 맵핑된다. FIT 범위는 하나의 FIT 파일 및 예를 들어, 512일 수 있는 FIT 파일들의 최대 수 사이를 포함할 수 있다.

<432> 10.3.2.2 FIT 파일 번호

<433> FIT 파일 번호는 FIT 내에서 FIT 파일을 식별하는데 사용되는 논리적 수이다.

<434> 10.3.3 파일 정보 포인터

<435> 파일 정보 포인터는 제어 로그 내에서, 정보 블록 리스트, 및 아마도 또한 정보 갱신 블록 리스트 내의 파일에 대한 엔트리에 대한 논리적 포인터이다.

<436> 파일 정보 포인터는 2개의 필드들을 갖는다:

<437> 1) 정보 범위; 및

<438> 2) 정보 번호

<439> 10.3.3.1 정보 범위

<440> 정보 범위는 정보 테이블의 서브셋이다. 각각의 정보 범위는 별도의 물리적 정보 블록에 맵핑된다. 정보 범위는 하나의 파일_정보 세트 및 예를 들어, 512일 수 있는 최대 수의 파일-정보 세트들 사이를 포함할 수 있다.

<441> 10.3.3.2 파일 번호

<442> 파일 번호는 정보 블록 내에서 파일_정보 세트를 식별하는데 사용되는 논리적인 수이다.

<443> 10.3.4 디렉토리 구조

<444> 디렉토리는 그 목적에 전용된 플래시 블록에 저장된다. 도10C은 예시적인 디렉토리 블록 포맷을 도시한다. 디렉토리는 페이지들의 세트로서 구성되며, 이들 각각 내에는 연속적인 파일ID 값들을 갖는 파일들에 대하여 엔트리들의 세트가 존재한다. 엔트리들의 이 세트는 디렉토리 범위라 칭해진다.

<445> 디렉토리는 제어 포인터에 의해 규정된 다음의 소거된 페이지 위치에서 디렉토리 페이지의 개정된 버전을 기록함으로써 갱신된다. 다수의 페이지들은 자신들을 메타페이지 내의 상이한 페이지들에 프로그래밍함으로써, 필요하다면 동시에 갱신될 수 있다.

<446> 디렉토리 범위에 대한 현재 페이지 위치들은 디렉토리 블록 내의 최종적인 기록된 페이지에서의 범위 포인터들에 의해 식별된다.

<447> 10.4 블록 리스트들

<448> 파일 인덱스 테이블 및 정보 테이블 둘 모두는 일련의 논리적 범위들을 포함하며, 여기서 범위는 물리적 플래시 블록과의 상관성을 갖는다. 블록 리스트들은 파일 데이터 포인터 또는 파일 정보 포인터에서 규정된 범위 및 물리적 블록 사이, 및 파일 데이터 포인터 또는 파일 정보 포인터에서 규정된 논리적 번호 및 파일 인덱스 테이블 및 정보 테이블 내의 물리적 블록들에서 사용되는 논리적 번호 사이의 상관성들을 기록하기 위하여 제어 로그에서 유지된다.

<449> 10.4.1 FIT 블록 리스트들

<450> FIT 블록 리스트는 파일에 대한 FIT 내의 엔트리들에 대해 FIT 파일 포인터를 할당하는 제어 로그 내의 리스트이다. FIT 파일 포인터는 파일 데이터 포인터에서 규정되는 동일한 FIT 파일 번호, 및 파일 데이터 포인터에서 규정된 범위에 할당되는 물리적 플래시 블록의 어드레스를 포함한다. FIT 블록 리스트 내의 엔트리는 단일 필드, 블록 물리적 어드레스를 포함한다.

<451> FIT 갱신 블록 리스트는 갱신되고 있는 FIT 내의 파일에 대한 엔트리들에 대해 FIT 파일 포인터를 할당하는 제어 로그 내의 리스트이다. FIT 파일 포인터는 FIT 갱신 블록 내에서 갱신되는 FIT 파일에 할당되는 FIT 갱신 파일 번호, 및 현재 FIT 갱신 블록 엔트리로서 할당되는 물리적 플래시 블록의 어드레스를 포함한다. FIT 갱신 블록 리스트 내의 엔트리는 3개의 필드들을 포함한다:

<452> 1) FIT 범위,

<453> 2) FIT 파일 번호, 및

<454> 3) FIT 갱신 파일 번호.

<455> 파일 데이터 포인터에 대응하는 FIT 파일 포인터가 FIT 블록 리스트로부터 결정되어야 할 때, 파일 데이터와 관련된 엔트리가 존재하는지를 결정하기 위하여 FIT 갱신 블록 리스트가 탐색된다. 존재하지 않는 경우, FIT 블록 리스트 내의 파일 데이터 포인터와 관련된 엔트리는 유효하다.

<456> 10.4.2 정보 블록 리스트들

<457> 호스트에 의해 기록된 파일_정보는 정보 테이블 내에 직접 저장되고 파일 정보 포인터에 의해 식별된다. 정보 블록 리스트는 정보 포인터를 정보 테에블 내의 파일_정보에 할당하기 위하여 존재한다. 이러한 정보 블록 리스트들에 대한 인덱싱 메커니즘들은 FIT 블록 리스트들에 대해 상술된 것과 완전히 유사하다.

<458> 정보 블록 리스트 내의 엔트리는 단일 필드: 블록 물리적 어드레스를 포함한다.

<459> 정보 갱신 블록 리스트 내의 엔트리는 3개의 필드들을 포함한다:

<460> 1) 정보 범위,

<461> 2) 정보 번호, 및

<462> 3) 갱신 정보 번호.

<463> 10.5 파일 인덱스 테이블

<464> 파일 인덱스 테이블(FIT)은 FIT 엔트리들의 스트링을 포함하며, 여기서 각각의 FIT 엔트리는 데이터 그룹의 플래시 메모리에서 파일 오프셋 어드레스 및 물리적 위치를 식별한다. FIT는 장치에 저장된 파일들에 대한 모든 유효 데이터 그룹들에 대한 엔트리들을 포함한다. 쓸모없는 데이터 그룹들은 FIT에 의해 인덱싱되지 않는다. 예시적인 FIT 논리적 구조가 도10D에 제공된다.

<465> 파일 내의 데이터 그룹들에 대한 FIT 엔트리들의 세트는 파일 오프셋 어드레스 순서로 연속적인 엔트리들로서 유지된다. 엔트리들의 세트는 FIT 파일로서 인식된다. FIT는 일련의 FIT 범위들로서 유지되고, 각각의 FIT 범위는 물리적 플래시 블록과의 상관성을 갖는다. FIT 범위들의 수는 장치 내의 데이터 그룹들의 수에 따라 가변될 것이다. 새로운 FIT 범위들이 생성될 것이며, 장치의 작동 동안 FIT 범위들이 제거될 것이다. FIT 블록 리스트들은 FIT 내의 위치가 식별될 수 있는 FIT 파일 포인터를 파일 데이터 포인터로부터 생성하는데 사용된다.

<466> 10.5.1 FIT 파일

<467> FIT 파일은 파일 내의 데이터 그룹들에 대한 인접한 FIT 엔트리들의 세트이다. 상기 세트의 엔트리들은 파일 오프셋 어드레스들의 순서이다. FIT 파일 내의 FIT 엔트리들은 연속적이며, 단일 FIT 범위 내에서 포함되거나, 하나의 FIT 범위로부터 다른 연속적인 FIT 범위로 오버플로우(overflow)된다.

<468> 10.5.2 FIT 헤더

<469> FIT 파일 내의 제1 엔트리는 FIT 헤더이다. 이것은 3개의 필드들을 포함한다:

<470> 1) 파일ID,

<471> 2) 프로그램 블록, 및

<472> 3) 프로그램 포인터.

<473> FIT 헤더는 FIT 엔트리의 정수(integral number)와 동일한 고정된 길이를 갖는다. 이 수는 1일수 있다.

<474> 10.5.2.1 파일ID

<475> 파일ID는 디렉토리 내의 파일에 대한 엔트리를 식별한다.

<476> 10.5.2.2 프로그램 블록

<477> 파일에 대한 프로그램 블록의 현재 물리적 위치는 FIT 파일의 갱신된 버전이 FIT에서 기록될 때마다 FIT 헤더에 레코딩된다. 이것은 파일이 호스트에 의해 재오픈될 때, 파일에 대한 프로그램 블록을 위치시키는데 사용된다. 이것은 또한 프로그램 블록 통합을 위해 선택되었던 파일에 대한 프로그램 블록 및 FIT 파일 사이의 대응성(correspondence)을 검증하는데 사용될 수 있다.

<478> 10.5.2.3 프로그램 포인터

<479> 파일에 대한 프로그램 블록 내의 프로그램 포인터의 현재 값은 FIT 파일의 갱신된 버전이 FIT에서 기록될 때마다 FIT 헤더에 레코딩된다. 이것은 파일이 호스트에 의해 재오픈될 때, 또는 프로그램 블록이 프로그램 블록 통합을 위해 선택되었을 때, 파일에 대한 프로그램 블록 내에서 데이터를 프로그래밍하는 위치를 규정하는데 사용된다.

<480> 10.5.3 FIT 엔트리

<481> FIT 엔트리는 데이터 그룹을 지정한다. 이것은 4개의 필드들을 갖는다:

<482> 1) 오프셋 어드레스,

<483> 2) 길이,

<484> 3) 포인터, 및

<485> 4) EOF 플래그.

<486> 10.5.3.1 오프셋 어드레스

<487> 오프셋 어드레스는 데이터 그룹의 제1 바이트와 관련된 파일 내에서의 바이트 단위의 오프셋이다.

<488> 10.5.3.2 길이

<489> 이것은 데이터 그룹 내의 파일 데이터의 바이트 단위의 길이이다. 완전한 데이터 그룹의 길이는 이 값보다 데이터 그룹 헤더의 길이만큼 더 길다.

<490> 10.5.3.3 포인터

<491> 이것은 데이터 그룹의 시작점의 플래시 블록 내의 위치에 대한 포인터이다. 이 포인터는 2개의 필드들을 갖는다:

<492> 1) 데이터 그룹을 포함하는 물리적 블록을 규정하는 블록 어드레스, 및

<493> 2) 데이터 그룹의 시작점의 블록 내에서의 바이트 오프셋을 규정하는 바이트 어드레스. 이 어드레스는 데이터 그룹 헤더를 포함한다.

<494> 10.5.3.4 EOF 플래그

<495> EOF 플래그는 파일의 종점인 것으로 데이터 그룹을 식별하는 단일 비트이다.

<496> 10.5.4 FIT 블록 포맷

<497> FIT 범위는 FIT 블록으로서 인식되는 단일 물리적 블록에 맵핑된다. 이러한 블록들 내의 데이터의 갱신된 버전들은 FIT 갱신 블록으로서 인식되는 공통 갱신 블록에서 프로그래밍된다. 데이터는 한 페이지의 단위로 갱신된다. 메타페이지 내의 다수의 페이지들은 필요하다면, 병렬로 갱신될 수 있다.

<498> 10.5.4.1 간접적인 어드레싱

<499> FIT 파일은 FIT 파일 포인터에 의해 식별된다. 이 포인터 내의 FIT 파일 번호 필드는 FIT 파일에 대한 데이터가 인덱싱을 위해 사용되는 물리적 구조들 내에서 이동될 때 일정하게 유지되는 논리적 포인터이다. 물리적 페이지 구조들 내의 포인터 필드들은 논리적 대 물리적 포인터 번역을 제공한다.

<500> 10.5.4.2 페이지 포맷

<501> FIT 블록들 및 FIT 갱신 블록들에서 사용된 페이지 포맷들은 동일하다.

<502> 페이지는 2개의 에어리어들, 즉, FIT 엔트리들에 대한 제1 에어리어 및 파일 포인터들에 대한 제2 에어리어로 세분된다. 일례가 도10E에 제공된다.

<503> 제1 에어리어는 데이터 그룹을 각각 지정하거나 FIT 파일에 대한 FIT 헤더를 각각 포함하는 FIT 엔터리들을 포함한다. FIT 페이지 내의 FIT 엔트리들의 수의 일례는 512이다. FIT 파일은 하나의 FIT 페이지 또는 중첩하는 2개 이상의 FIT 페이지들 내에서, FIT 엔트리들의 인접한 세트에 의해 지정된다. FIT 헤더를 포함하는, FIT 파일의 제1 엔트리는 제2 에어리어 내의 파일 포인터에 의해 식별된다.

<504> 제2 에어리어는 단지 가장 최근에 프로그래밍되었던 FIT 페이지 내의 유효 파일 포인터들을 포함한다. 모든 다른 페이지들의 제2 에어리어는 쓸모없고, 사용되지 않는다. 파일 포인터 에어리어는 FIT 블록에 포함될 수 있는 각각의 FIT 파일에 대해 하나의 엔트리를 포함하는데, 즉, 파일 포인터 엔트리들의 수는 FIT 블록에 존재할 수 있는 FIT 파일들의 최대 수와 동일하다. 파일 포인터 엔트리들은 FIT 파일 번호에 따라 순차적으로 저장된다. N번째 파일 포인터 엔트리는 FIT 블록 내의 FIT 파일(N)에 대한 포인터를 포함한다. 이것은 2개의 필드들을 갖는다:

<505> 1) FIT 블록 내에서의 물리적 페이지를 지정하는 페이지 번호, 및

<506> 2) 물리적 페이지 내에서의 FIT 엔트리를 지정하는 엔트리 번호.

<507> 파일 포인터 엔트리는 FIT 블록 내의 논리적 FIT 파일 번호를 블록 내의 물리적 위치로 번역하는 메커니즘을 제공한다. 파일 포인터들의 완전한 세트는 모든 FIT 페이지가 프로그래밍될 때 갱신되지만, 단지 가장 최근에 프로그래밍된 페이지에서만 유효하다. FIT 파일이 FIT 갱신 블록에서 갱신될 때, FIT 블록 또는 FIT 갱신 블록 중 하나에서의 FIT 파일의 이전 위치는 쓸모없게 되고, 더 이상 파일 포인터에 의해 참조되지 않는다.

<508> 10.5.5 FIT 갱신 블록들

<509> FIT 블록 내의 FIT 파일들에 대한 변화는 모든 FIT 블록들 사이에서 공유되는 단일 FIT 갱신 블록에서 행해진다. 물리적 FIT 블록들의 일례가 도10F에 도시되어 있다.

<510> 파일 데이터 포인터는 FIT 파일에 대한 논리적 포인터이다. 이것의 FIT 범위 필드는 이 FIT 범위에 맵핑되는

FIT 블록의 물리적 어드레스를 식별하기 위하여 FIT 블록 리스트에 어드레스하는데 사용된다. 그 후, FIT 파일 포인터의 FIT 파일 번호 필드는 FIT 블록에서 타겟 FIT 파일에 대한 정확한 파일 포인터를 선택한다.

<511> 파일 데이터 포인터의 FIT 범위 필드 및 FIT 파일 번호 필드 둘 모두는 타겟 FIT 파일이 갱신되었는지를 식별하기 위하여 FIT 갱신 블록 리스트에 어드레스하는데 사용된다. 엔트리가 이 리스트에서 발견되는 경우, 이 엔트리는 FIT 파일의 갱신된 버전의 갱신 블록 내에서 FIT 파일 번호 및 FIT 갱신 블록의 물리적 블록 어드레스를 제공한다. 이것은 FIT 블록에서 FIT 파일에 대해 사용된 FIT 파일 번호와 상이할 수 있다. FIT 갱신 블록은 FIT 파일의 유효 버전을 포함하며, FIT 블록 내의 버전은 쓸모없다.

<512> 10.5.6 갱신 작동들

<513> FIT 블록은 통합 작동 동안에만 프로그래밍된다. 이로 인해, FIT 파일들이 블록 내에서 밀접하게 패킹된다. FIT 갱신 블록은 FIT 엔트리들이 변경되거나, 추가되거나 제거될 때, 및 압축 작동 동안 갱신된다. 도10G은 FIT 파일들에 대한 갱신 작동의 예들을 도시한다.

<514> FIT 파일들은 통합 작동의 결과로서 FIT 블록에서 밀접하게 패킹된다. FIT 블록은 자신 내에 존재할 수 있는 FIT 파일들의 최대 수가 존재하기 때문에, 완전히 채워지지 않을 수 있다. FIT 파일들은 하나의 페이지로부터 다음 페이지로 오버플로우될 수 있다. FIT 블록 내의 FIT 파일은 자신이 갱신되고 FIT 갱신 블록에서 재기록될 때 쓸모없게 된다.

<515> FIT 파일은 갱신될 때, FIT 갱신 블록 내의 다음의 이용 가능한 페이지에서 완전히 재기록된다. FIT 파일을 갱신하는 것은 기존의 FIT 엔트리들의 내용을 변화시키는 것, 또는 FIT 엔트리들의 수를 변화시키는 것 중 하나로 이루어질 수 있다. FIT 파일들은 하나의 페이지로부터 다음 페이지로 오버플로우될 수 있다. FIT 갱신 블록 내의 FIT 파일들은 모두 동일한 FIT 범위와 관련될 필요는 없다.

<516> 10.5.7 FIT 범위의 생성

<517> FIT 파일들에 대한 추가적인 저장 공간을 수용하기 위하여 새로운 FIT 범위가 생성되어야 할 때, FIT 블록은 즉시 생성되지는 않는다. 이 범위 내의 새로운 데이터는 최초에 FIT 갱신 블록에 기록된다. 그 후에 , FIT 블록은 범위에 대해 통합 작동이 수행될 때 생성된다.

<518> 10.5.8 압축 및 통합

<519> 10.5.8.1 디렉토리 갱신 블록 또는 FIT 갱신 블록의 압축

<520> FIT 갱신 블록이 채워질 때, 이의 유효 FIT 파일 데이터는 나중에 갱신 블록이 되는 소거된 블록에 압축된 형태로 프로그래밍될 수 있다. 갱신이 몇 개의 파일들에만 관련되는 경우, 프로그래밍될 적은 한 페이지의 압축된 유효 데이터가 존재할 수 있다.

<521> 압축 작동에서 재배치될 FIT 파일이 클로우즈된 파일과 관련되고, 그 범위에 대한 FIT 블록이 충분한 프로그래밍되지 않은 페이지들을 포함하는 경우, FIT 파일은 압축된 갱신 블록보다는 오히려, FIT 블록에 재배치될 수 있다.

<522> 10.5.8.2 디렉토리 블록 또는 FIT 블록의 통합

<523> FIT 엔트리들이 갱신될 대, FIT 블록 내의 원래 FIT 파일은 쓸모없게 된다. 이와 같은 FIT 블록들은 쓸모없는 공간을 복구하기 위하여 주기적으로 가비지 수집을 겪어야 한다. 이것은 통합 작동에 의해 성취된다. 게다가, 새로운 파일들이 범위내에서 생성될 수 있고, 갱신 블록 내의 엔트리를 가질 수 있지만, FIT 블록 내의 대응하는 쓸모없는 엔트리들은 존재하지 않을 수 있다. 이와 같은 FIT 파일들은 FIT 블록에 주기적으로 재배치되어야 한다.

<524> 갱신 블록 내의 FIT 파일들은 관련 범위에 대한 FIT 블록 내로 통합될 수 있으므로, 갱신 블록으로부터 제거될 수 있는 반면, 다른 FIT 파일들은 갱신 블록에서 유지된다.

<525> FIT 파일 내의 FIT 엔트리들의 수가 갱신 프로세스 동안 증가하고, FIT 범위에 대한 유효 데이터가 단일 소거된 블록 내로 통합될 수 없는 경우, 그 FIT 범위에 원래 할당된 일부 FIT 파일들은 또 다른 FIT 범위에 할당될 수 있고, 통합은 별도의 작동들로 2개의 블록들 내로 수행될 수 있다. FIT 파일의 이와 같은 재배치의 경우에, 디렉토리 내의 파일 데이터 포인터는 새로운 FIT 범위를 반영하기 위하여 갱신되어야 한다.

<526> 범위에 대한 통합 작동은 FIT 갱신 블록 내의 그 범위에 대한 유효 데이터의 용량이 규정된 임계값에 도달할 때

수행되어야 한다. 이 임계값의 일례는 50%이다.

<527> 압축은 여전히 오픈되어 있고 호스트가 지속적으로 액세스할 수 있는 파일들과 관련된 활성 FIT 파일들에 대해 통합에 우선하여 수행되어야 한다.

<528> 10.6 정보 테이블

<529> 정보 테이블은 10.5 장의 파일 인덱스 테이블에 대해 규정되는 것과 동일한 구조들, 인덱싱 메커니즘들 및 갱신 기술들을 사용한다. 그러나, 파일에 대한 파일_정보는 직접적인 데이터 파일 플랫폼 내에서 해석되지 않는 정보의 단일 스트링을 포함한다.

<530> 10.7 데이터 그룹들

<531> 데이터 그룹은 단일 메모리 블록 내의 인접한 물리적 어드레스들에서 프로그래밍되는, 파일에 대한 인접한 오프셋 어드레스들을 갖는 파일 데이터의 세트이다. 파일은 통상적으로 다수의 데이터 그룹들로서 프로그래밍될 것이다. 데이터 그룹은 하나의 바이트 및 하나의 블록 사이의 임의의 길이를 가질 수 있다.

<532> 10.7.1 데이터 그룹 헤더

<533> 각각의 데이터 그룹은 교차 상관 목적들을 위한 파일 식별자 정보를 포함하는 헤더와 함께 프로그래밍된다. 헤더는 이의 데이터 그룹이 부분을 형성하는 파일에 대한 FIT 파일 포인터를 포함한다.

<534> 11. 블록 상태 관리

<535> 11.1 블록 상태들

<536> 파일 데이터의 저장에 대한 블록들은 도11A의 상태도에 도시된 바와 같이, 다음의 8개의 상태들로 분류될 수 있다.

<537> 11.1.1 소거된 블록

<538> 소거된 블록은 소거된 블록 풀 내의 소거된 상태에 있다. 이 상태로부터의 가능한 트랜지션은 다음과 같다:

<539> (a) 소거된 블록 대 프로그램 블록

<540> 단일 파일에 대한 데이터는 자신이 호스트로부터 공급될 때, 또는 자신이 파일에 대한 가비지 수집 동안 카피될 때, 소거된 블록으로 프로그래밍된다.

<541> 11.1.2 프로그램 블록

<542> 프로그램 블록은 부분적으로 단일 파일에 대한 유효 데이터로 프로그래밍되며, 일부 소거된 용량을 포함한다. 파일은 오픈되거나 클로우즈될 수 있다. 파일에 대한 추가적인 데이터는 호스트에 의해 공급될 때, 또는 파일의 가비지 수집 동안 카피될 때, 블록으로 프로그래밍되어야 한다.

<543> 이 상태로부터의 가능한 트랜지션들은 다음과 같다:

<544> (b) 프로그램 블록 대 프로그램 블록

<545> 단일 파일에 대한 데이터는 자신이 호스트로부터 공급될 때, 또는 자신의 파일에 대한 가비지 수집 동안 카피될 때 그 파일에 대한 프로그램 블록에 프로그래밍된다.

<546> (c) 프로그램 블록 대 파일 블록

<547> 호스트로부터의 단일 파일에 대한 데이터는 그 파일에 대한 프로그램 블록을 채우기 위하여 프로그래밍된다.

<548> (f) 프로그램 블록 대 쓸모없는 블록

<549> 프로그램 블록 내의 파일에 대한 모든 데이터는 가비지 수집 동안 또 다른 블록으로 카피되는 유효 데이터, 또는 호스트에 의해 삭제되는 파일의 모두 또는 일부의 결과로서, 쓸모없게 된다.

<550> (h) 프로그램 블록 대 쓸모없는 프로그램 블록

<551> 프로그램 블록 내의 데이터의 일부는 동일한 프로그램 블록에서 호스트에 의해 기록되는 데이터의 갱신된 버전, 또는 호스트에 의해 삭제되는 파일의 일부의 결과로서 쓸모없게 된다.

<552> (l) 프로그램 블록 대 공통 블록

<553> 파일에 대한 나머지 데이터는 파일 또는 공통 블록의 가비지 수집 동안, 또는 프로그램 블록들의 통합 동안 상이한 클로우즈된 파일에 대한 프로그램 블록으로 프로그래밍된다.

<554> 11.1.3 파일 블록

<555> 파일 블록은 단일 파일에 대한 완전히 유효한 데이터로 채워진다.

<556> 이 상태로부터의 가능한 트랜지션들은 다음과 같다:

<557> (d) 파일 블록 대 쓸모없는 파일 블록

<558> 파일 블록 내의 데이터의 일부는 파일에 대한 프로그램 블록에서 호스트에 의해 프로그래밍되는 데이터의 갱신된 버전의 결과로서 쓸모없게 된다.

<559> (g) 파일 블록 대 쓸모없는 블록

<560> 파일 블록 내의 모든 데이터는 파일에 대한 프로그램 블록에서 호스트에 의해 프로그래밍되는 블록 내의 데이터의 갱신된 버전, 또는 호스트에 의해 삭제되는 파일의 모두 또는 일부의 결과로서 쓸모없게 된다.

<561> 11.1.4 쓸모없는 파일 블록

<562> 쓸모없는 파일 블록은 단일 파일에 대한 유효 데이터 및 쓸모없는 데이터의 임의의 조합으로 채워진다.

<563> 이 상태로부터의 가능한 트랜지션들은 다음과 같다:

<564> (e) 쓸모없는 파일 블록 대 쓸모없는 블록

<565> 쓸모없는 파일 블록 내의 모든 데이터는 파일에 대한 프로그램 블록에서 호스트에 의해 프로그래밍되는 블록 내의 유효 데이터의 갱신된 버전, 가비지 수집 동안 또 다른 블록으로 카피되는 유효 데이터, 또는 호스트에 의해 삭제되는 파일의 모두 또는 일부의 결과로서 쓸모없게 된다.

<566> 11.1.5 쓸모없는 프로그램 블록

<567> 쓸모없는 프로그램 블록은 부분적으로 단일 파일에 대한 유효 데이터 및 쓸모없는 데이터의 임의의 조합으로 프로그래밍되며, 일부의 소거된 용량을 포함한다. 파일에 대한 추가적인 데이터는 호스트에 의해 공급될 때 블록으로 프로그래밍되어야 한다. 그러나, 가비지 수집 동안, 파일에 대한 데이터는 블록으로 카피되지 않아야 하며, 새로운 프로그램 블록이 오픈되어야 한다.

<568> 이 상태로부터의 가능한 트랜지션들은 다음과 같다:

<569> (i) 쓸모없는 프로그램 블록 대 쓸모없는 프로그램 블록

<570> 단일 파일에 대한 데이터는 자신이 호스트로부터 공급될 때, 그 파일에 대한 쓸모없는 프로그램 블록에 프로그래밍된다.

<571> (j) 쓸모없는 프로그램 블록 대 쓸모없는 블록

<572> 쓸모없는 프로그램 블록 내의 파일에 대한 모든 데이터는 가비지 수집 동안 또 다른 블록으로 카피되는 유효 데이터, 또는 호스트에 의해 삭제되는 파일의 모두 또는 일부의 결과로서 쓸모없게 된다.

<573> (k) 쓸모없는 프로그램 블록 대 쓸모없는 파일 블록

<574> 단일 파일에 대한 데이터는 자신이 호스트로부터 공급될 때, 그 파일에 대한 쓸모없는 프로그램 블록을 채우기 위하여 프로그래밍된다.

<575> 11.1.6 공통 블록

<576> 공통 블록은 2개 이상의 파일들에 대한 유효 데이터로 프로그래밍되고, 통상적으로 일부의 소거된 용량을 포함한다. 임의의 파일에 대한 나머지 데이터는 가비지 수집 또는 프로그램 블록들의 통합 동안 이 블록에 프로그래밍될 수 있다.

<577> 이 상태로부터의 가능한 트랜지션들은 다음과 같다:

<578> (m) 공통 블록 대 공통 블록

<579> 파일에 대한 나머지 데이터는 파일 또는 공통 블록의 가비지 수집 동안, 또는 프로그램 블록들의 통합 동안 공

통 블록에 프로그래밍된다.

<580> (n) 공통 블록 대 쓸모없는 공통 블록

<581> 공통 블록 내의 하나의 파일에 대한 데이터의 일부 또는 모두는 파일에 대한 프로그램 블록에서 호스트에 의해 프로그래밍되는 데이터의 갱신된 버전, 파일의 가비지 수집 동안 또 다른 블록에 카피되는 데이터, 또는 호스트에 의해 삭제되는 파일의 모두 또는 일부의 결과로서 쓸모없게 된다.

<582> 11.1.7 쓸모없는 공통 블록

<583> 쓸모없는 공통 블록은 2개 이상의 파일들에 대한 유효 데이터 및 쓸모없는 데이터의 임의의 조합으로 프로그래밍되며, 통상적으로 일부의 소거된 용량을 포함한다. 추가적인 데이터는 블록으로 프로그래밍되지 않아도 된다.

<584> 이 상태로부터의 가능한 트랜지션들은 다음과 같다:

<585> (o) 쓸모없는 공통 블록 대 쓸모없는 블록

<586> 쓸모없는 공통 블록 내의 모든 파일들에 대한 데이터는 파일에 대한 프로그램 블록에서 호스트에 의해 프로그래밍되는 하나의 파일에 대한 데이터의 갱신된 버전, 파일의 가비지 수집 동안 또 다른 블록으로 카피되는 하나의 파일에 대한 데이터, 또는 호스트에 의해 삭제되는 하나의 파일의 모두 또는 일부의 결과로서 쓸모없게 된다.

<587> 11.1.8 쓸모없는 블록

<588> 쓸모없는 블록은 쓸모없는 데이터만을 포함하지만, 아직 소거되지 않았다.

<589> 이 상태로부터의 가능한 트랜지션은 다음과 같다:

<590> (p) 쓸모없는 블록 대 소거된 블록

<591> 쓸모없는 블록은 가비지 수집 동안 소거되며, 다시 소거된 블록 풀에 추가된다.

<592> 12. 소거된 블록 관리

<593> 12.1 메타블록 링킹(Metablock Linking)

<594> 소거 블록들을 메타블록들 내로 링킹하는 방법은 이전의 제3세대 LBA 시스템에 대해 규정된 것으로부터 변화되지 않는다.

<595> 12.2 소거된 블록 풀

<596> 소거된 블록 풀은 파일 데이터 또는 제어 정보의 저장을 위한 할당에 이용 가능한 장치 내의 소거된 블록들의 풀이다. 풀 내의 각각의 소거된 블록은 메타블록이며, 모든 메타블록들은 동일한 고정된 병렬성을 갖는다.

<597> 풀 내의 소거된 블록들은 제어 블록 내의 소거된 블록 로그 내의 엔트리들로서 기록된다. 엔트리들은 블록들의 소거 순서에 따라 로그 내에서 순서가 매겨진다. 할당을 위한 소거된 블록은 로그의 헤드에서 엔트리로서 선택된다. 엔트리는 블록이 소거될 때 로그의 테일(tail)에 추가된다.

<598> 13. 제어 데이터 구조들

<599> 제어 데이터 구조들은 그 목적에 전용된 플래시 블록들에 저장된다. 블록들의 3개의 등급들은 다음과 같이 규정된다:

<600> 1) 파일 디렉토리 블록,

<601> 2) 파일 인덱스 테이블 블록, 및

<602> 3) 제어 블록.

<603> 13.1 파일 디렉토리 블록

<604> 파일 디렉토리 블록의 구조는 상술되었다.

<605> 13.2 파일 인덱스 테이블 블록

<606> 파일 인덱스 테이블 블록들의 구조는 상술되었다.

<607> 13.3 제어 블록

<608> 제어 블록은 4개의 독립적인 로그들에 제어 정보를 저장한다. 로그들 각각에 대해 별도의 페이지가 할당된다. 이것은 필요하다면, 로그 당 다수의 페이지들로 확장될 수 있다. 제어 블록의 예시적인 포맷이 도13A에 도시되어 있다.

<609> 로그는 제어 포인터에 의해 규정된 다음의 소거된 페이지 위치에서 완전한 로그의 개정된 버전을 기록함으로써 갱신된다. 다수의 로그들은 상기 로그들을 메타페이지 내의 다른 페이지들로 프로그래밍함으로써, 필요하다면, 동시에 갱신될 수 있다. 4개의 로그들 각각의 유효 버전들의 페이지 위치들은 제어 블록 내의 최종 기록된 페이지에서의 로그 포인터들에 의해 식별된다.

<610> 13.3.1 공통 블록 로그

<611> 공통 블록 로그는 장치에 존재하는 모든 공통 블록에 관한 정보를 레코딩한다. 공통 블록 로그 내의 로그 엔트리들은 도12B에 도시된 바와 같이, 2개의 에어리어들, 즉 블록 엔트리들에 대한 제1 에어리어 및 데이터 그룹 엔트리들에 대한 제2 에어리어로 세분된다. 각각의 블록 엔트리는 공통 블록의 물리적 위치를 레코딩한다. 엔트리들은 고정된 크기를 가지며, 공통 블록 로그에서 고정된 수가 존재한다. 각각의 엔트리는 다음 필드들을 갖는다:

<612> 1) 블록 물리적 어드레스,

<613> 2) 프로그래밍을 위한 공통 블록 내의 다음의 이용 가능한 페이지에 대한 포인터,

<614> 3) 블록에 대한 데이터 그룹 엔트리들 중 제1 데이터 그룹 엔트리에 대한 포인터, 및

<615> 4) 데이터 그룹 엔트리들의 수

<616> 데이터 그룹 엔트리는 공통 블록 내의 데이터 그룹에 관한 정보를 기록한다. 인접한 데이터 그룹 엔트리들의 세트는 공통 블록 내의 모든 데이터 그룹들을 규정한다. 공통 블록에는 가변하는 수의 데이터 그룹들이 존재한다. 각각의 엔트리는 바람직하게는 다음 필드들을 갖는다:

<617> 1) 공통 블록 내의 바이트 어드레스, 및

<618> 2) FIT 파일 포인터.

<619> 13.3.2 프로그램 블록 로그

<620> 프로그램 블록 로그는 클로우즈된 파일들에 대해 장치 내에 존재하는 모든 프로그램 블록에 관한 정보를 레코딩한다. 각각의 프로그램 블록에 대해 하나의 엔트리가 존재하며, 상기 하나의 엔트리는 다음의 필드들을 갖는다:

<621> 1) 블록 물리적 어드레스,

<622> 2) 프로그래밍을 위한 프로그램 블록 내의 다음의 이용 가능한 페이지에 대한 포인터, 및

<623> 3) FIT 파일 포인터.

<624> 13.3.3 소거된 블록 로그

<625> 소거된 블록 로그는 장치에 존재하는 모든 소거된 블록의 아이덴티티를 레코딩한다. 각각의 소거된 블록에 대해 하나의 엔트리가 존재한다. 엔트리들은 블록들의 소거 순서에 따라 로그에서 순서가 매겨진다. 할당을 위한 소거된 블록은 로그의 헤드에서 엔트리로서 선택된다. 엔트리는 블록이 소거될 때 로그의 테일에 추가된다. 엔트리는 단일 필드: 블록 물리적 어드레스를 갖는다.

<626> 13.3.4 제어 로그

<627> 제어 로그는 다음 필드들에서 다양한 제어 정보를 레코딩한다:

<628> 13.3.4.1 오픈 파일 리스트

<629> 이 필드는 다음과 같은 현재 오픈 파일들 각각에 관한 정보를 포함한다:

<630> 1) 경로이름,

<631> 2) 파일이름,

<632> 3) FIT 파일 포인터, 및

<633> 4) 프로그램 블록 물리적 어드레스.

<634> 오픈 파일들에 대한 프로그램 블록들은 프로그램 블록 로그에 포함되지 않는다.

<635> 13.3.4.2 공통 블록 카운트

<636> 이 필드는 공통 블록 로그에서 레코딩된 공통 블록들의 총 수를 포함한다.

<637> 13.3.4.3 프로그램 블록 카운트

<638> 이 필드는 프로그램 블록 로그에서 레코딩된 프로그램 블록들의 총수를 포함한다. 카운트는 블록들이 프로그램 블록 로그에 추가되고 상기 프로그램 블록 로그로부터 제거될 때 갱신된다.

<639> 13.3.4.4 소거된 블록 카운트

<640> 이 필드는 소거된 블록 로그에서 레코딩된 소거된 블록의 총수를 포함한다. 카운트는 블록들이 소거된 블록 로그에 추가되고 상기 소거된 블록 로그로부터 제거될 때 갱신된다.

<641> 13.3.4.5 프로그램/공통 블록 페이지 카운트

<642> 이 필드는 프로그램 블록들 및 공통 블록들 내의 유효 데이터 페이지들의 수의 카운트를 포함한다. 카운트는 블록들이 프로그램 블록 로그 및 공통 블록 로그에 추가되고 상기 프로그램 블록 로그 및 공통 블록 로그로부터 제거될 때 갱신된다.

<643> 13.3.4.6 쓸모없는 블록 카운트

<644> 이 필드는 가비지 수집을 대기하는 완전히 쓸모없는 블록들의 수의 카운트를 포함한다. 카운트는 블록들이 쓸모없는 블록 가비지 수집 큐에 추가되고 상기 쓸모없는 블록 가비지 수집 큐로부터 제거될 때 갱신된다.

<645> 13.3.4.7 FIT 블록 리스트

<646> 이 필드는 FIT 범위를 FIT 블록에 맵핑하기 위한 정보를 포함한다. 이 필드는 각각의 FIT 범위에 대한 FIT 블록 물리적 어드레스를 규정하는 엔트리를 포함한다.

<647> 13.3.4.8 FIT 갱신 블록 리스트

<648> 이 필드는 FIT 범위 및 FIT 파일 번호를 FIT 갱신 파일 번호에 맵핑하기 위한 정보를 포함한다. 이 필드는 갱신 블록에 존재하는 각각의 유효 FIT 파일에 대한 엔트리를 포함한다. 엔트리는 다음의 3개의 필드들을 갖는다:

<649> 1) FIT 범위,

<650> 2) FIT 파일 번호, 및

<651> 3) FIT 갱신 파일 번호.

<652> 13.3.4.9 디렉토리 블록 리스트

<653> 이 필드는 디렉토리 범위를 디렉토리 블록에 맵핑하기 위한 정보를 포함한다. 이 필드는 각각의 디렉토리 범위에 대한 디렉토리 블록 물리적 어드레스를 규정하는 엔트리를 포함한다.

<654> 13.3.4.10. 디렉토리 갱신 블록 리스트

<655> 이 필드는 디렉토리 범위 및 서브디렉토리 번호를 갱신 서브디렉토리 번호에 맵핑하기 위한 정보를 포함한다. 이 필드는 갱신 블록에 존재하는 각각의 유효 서브디렉토리에 대한 엔트리를 포함한다. 엔트리는 다음의 3개의 필드를 갖는다:

<656> 1) 디렉토리 범위,

<657> 2) 서브디렉토리 번호, 및

<658> 3) 갱신 서브디렉토리 번호.

<659> 13.3.4.11 버퍼 스왑 블록 인덱스

<660> 이 필드는 스왑 블록 내의 유효 데이터 그룹들에 대한 인덱스를 포함한다. 각각의 데이터 그룹에 대한 인덱스는 다음 필드들을 포함한다:

<661> 1) FIT 파일 포인터,

<662> 2) 스왑 블록 내의 바이트 어드레스, 및

<663> 3) 길이.

<664> 13.3.4.12 우선순위 쓸모없는 블록 큐

<665> 이 필드는 가비지 수집에 대한 우선순위 쓸모없는 블록 큐 내의 모든 블록들의 블록 어드레스들을 포함한다.

<666> 13.3.4.13 우선순위 공통 블록 큐

<667> 이 필드는 가비지 수집에 대한 우선순위 공통 블록 큐 내의 모든 블록들의 블록 어드레스들을 포함한다.

<668> 13.3.4.14 쓸모없는 블록 큐

<669> 이 필드는 가비지 수집에 대한 쓸모없는 블록 큐 내의 모든 블록들의 블록 어드레스들을 포함한다.

<670> 13.3.4.15 공통 블록 큐

<671> 이 필드는 가비지 수집에 대한 공통 블록 큐 내의 모든 블록들의 블록 어드레스들을 포함한다.

<672> 13.3.4.16 파일 큐

<673> 이 필드는 가비지 수집에 대한 파일 큐 내의 모든 파일들의 FIT 파일 포인터들을 포함한다.

<674> 14. 정적 파일들

<675> 14.1 정적 파일들

<676> 일부의 호스트들은 동일한 크기들을 갖는 파일들의 세트를 생성하고 상기 세트의 파일들 내에서 데이터를 주기적으로 갱신함으로써 직접적인 데이터 파일 장치에 데이터를 저장할 수 있다. 이와 같은 세트의 부분인 파일은 정적 파일이라 칭해진다. 호스트는 메모리 카드와 무관하거나, 온-카드 애플리케이션(on-card application)을 실행시키고 있는 메모리 카드 내의 프로세서일 수 있다.

<677> 정적 파일들의 사용의 예시적인 애플리케이션은 본원과 동시에 출원된 명칭이 "Interfacing systems Operating Through A Logical Address Space and on a Direct Data File Basis"인 Sergey Anatolievich Gorobets의 특허 출원에 설명되어 있다. 그 출원에서, 호스트의 논리적 어드레스 공간은 메모리 제어기에 의하여 이와 같은 정적 파일들로 분할된다.

<678> 직접적인 데이터 파일 장치는 임의의 다른 파일에 대한 것과 정확히 동일한 방식으로 정적 파일의 저장을 관리한다. 그러나, 호스트는 정적 파일들로 장치의 성능 및 작동을 최적화하는 방식으로 직접적인 데이터 파일 명령 세트 내의 명령들을 사용할 수 있다.

<679> 14.1.1 정적 파일 구획(Static File Partition)

<680> 정적 파일들은 장치 내의 전용된 구획에 세트로서 저장된다. 구획 내의 모든 정적 파일들은 동일한 파일 크기를 갖는다.

<681> 14.1.2 정적 파일 크기

<682> 파일 크기는 파일에 기록된 오프셋 어드레스들의 범위를 통하여, 호스트에 의해 규정된다. 정적 파일들은 메타블록의 크기와 동일한 크기를 갖는다.

<683> 호스트는 기록_포인터 및 판독_포인터에 의해 표현된 파일 오프셋 값들을 관리하여, 상기 값들을 항상 정적 파일에 대해 허용되는 값들의 범위 내에서 유지하도록 한다.

<684> 14.1.3 정적 파일들 삭제

<685> 직접적인 데이터 파일 장치에서의 다른 파일들과 달리, 호스트는 정상적인 작동 동안 정적 파일을 삭제하지 않는다. 정적 파일은 호스트에 의해 생성되고 나서, 장치에서 지속적으로 존재한다. 임의의 시간에 파일에 기록된 데이터는 기존 파일 데이터에 겹쳐쓰기된다.

<686> 그러나, 호스트는 예를 들어, 장치를 리포맷시키거나 장치에서 정적 파일들에 대한 구획의 크기를 감소시키기 위한 호스트에 의한 작동 동안, 항상 정적 파일을 삭제하는 능력을 갖는다.

<687> 14.2 정적 파일들과 함께 사용된 명령 세트

<688> 도13A는 정적 파일들에 필요한 모든 작동들을 지원하는 정적 파일들과 함께 사용하기 위한 명령 세트, 도2A 내지 2F에 도시된 이의 서브세트를 제공한다.

<689> 14.3 정적 파일들 생성(Creating Static Files)

<690> 정적 파일은 호스트로부터의 생성 명령의 사용에 의하여 장치에서 생성된다. 호스트는 통상적으로 자신이 파일을 식별하기를 희망하는 파일ID를 지정할 것이다.

<691> 호스트는 자신이 장치에서 어느 파일들을 생성하였는지를 추적하거나, 자신이 그 파일ID가 장치에 이미 존재하지 않는 파일을 오픈하도록 시도한 이후에 장치로부터 에러 메시지에 응답하여 파일을 생성할 수 있다.

<692> 14.4 정적 파일들 오픈(Opening Static Files)

<693> 호스트는 파일에 대한 파일ID를 파라미터로서 사용하여 오픈 명령을 전송함으로써 정적 파일을 오픈한다.

<694> 호스트는 자신이 장치에서 동시에 오픈되는 파일들의 수 또는 장치에서 동시에 오픈되는 호스트에 의해 규정된 특정 유형의 파일들의 수를 제어하는 방식으로 장치 내의 정적 파일들의 세트와 함께 작동할 수 있다. 따라서, 호스트는 또 다른 정적 파일을 오픈하기 전에 하나 이상의 정적 파일들을 클로우즈할 수 있다.

<695> 14.5 정적 파일들로의 기록(Writing to Static Files)

<696> 정적 파일이 처음 기록될 때, 상기 정적 파일은 파일 크기가 플래시 메모리 내의 메타블록의 크기와 정확히 동일한 것으로 호스트에 의해 규정되기 때문에, 장치에서 단일의 완전한 파일 블록을 차지한다. 따라서, 파일에 대한 오프셋 어드레스 범위는 플래시 메모리 내의 메타블록의 크기와 정확히 동일하다.

<697> 정적 파일로의 후속 기록들은 데이터가 이 오프셋 어드레스 범위 내에서 갱신되도록 한다. 호스트는 기록_포인터 명령에 의하여 파일에 대한 기록-포인터 값을 제어함으로써 데이터가 갱신되고 있는 파일 오프셋 어드레스를 제어한다. 호스트는 기록_포인터 값이 정적 파일의 크기와 관련된 오프셋 어드레스 범위의 끝을 초과하지 않도록 한다. 유사하게, 호스트는 기록_포인터 값을 이 범위 내에서 억제한다.

<698> 파일이 오픈된 이후에 정적 필드 내의 기존 데이터가 갱신될 때, 갱신된 데이터가 프로그래밍되는 프로그램 블록이 오픈된다. 파일 블록 내의 대응하는 오프셋 어드레스를 갖는 데이터는 쓸모없어진다. 완전한 정적 파일이 갱신되는 경우, 프로그램 블록 내의 모든 데이터는 유효하고, 프로그램 블록은 파일에 대한 파일 블록이 된다. 파일에 대한 이전 파일 블록 내의 모든 데이터는 쓸모없어지고, 이 블록은 쓸모없는 블록 가비지 수집 큐에 추가된다. 소거된 블록은 파일에 대한 추가적인 갱신된 데이터가 수신되는 경우, 프로그램 블록으로서 할당된다.

<699> 정적 파일에 대한 프로그램 블록이 가득 차지만, 상기 블록이 파일에 대한 모든 유효 데이터를 포함하지 않는 경우에, 프로그램 블록 내의 데이터의 일부는 동일한 오프셋 어드레스에 대해 다수의 갱신들이 행해지기 때문에, 쓸모없다. 이 경우에, 프로그램 블록은 파일 블록이 될 수 없고, 또 다른 빈 프로그램 블록은 파일에 대한 추가적인 데이터가 수신될 때, 오픈되지 않는다. 프로그램 블록으로부터의 유효 데이터가 카피되는 소거된 블록이 할당되고(프로그램 블록이 압축됨) 나서, 부분적으로 채워진 블록이 파일에 대한 프로그램 블록이 된다. 파일에 대한 이전의 프로그램 블록 내의 모든 데이터는 이제 쓸모없고, 블록은 쓸모없는 블록 가비지 수집 큐에 추가된다.

<700> 호스트가 다음 단락 14.6에서 설명된 바와 같이, 파일을 클로우즈함으로써, 파일에 대한 어떤 유효 데이터를 각각 포함하는 파일 블록 및 프로그램 블록의 통합을 강요할 수 있다. 호스트는 파일에 대한 추가적인 데이터가 수신될 때 직접적인 데이터 파일 장치가 프로그램 블록을 압축하도록 하기보다는 오히려, 부분적으로 쓸모없는 프로그램 블록이 가득 찰 때, 파일을 일시적으로 클로우즈하는 것을 택할 수 있다.

<701> 14.6 정적 파일들 클로우즈(Closing Static Files)

<702> 호스트는 파일에 대한 파일ID를 파라미터로서 사용하여 클로우즈 명령을 전송함으로써 정적 파일을 클로우즈한다.

<703> 정적 파일의 클로우저(closure)는 파일에 대한 데이터의 일부만이 갱신되었던 경우에, 파일이 파일 가비지 수집 큐에 놓이도록 한다. 이것은 다음 단락 14.7에서 설명된 바와 같이, 파일에 대한 후속 가비지 수집 작동을 허용한다. 그러나, 호스트는 또한 단락 14.7에 설명된 바와 같이, 파일에 대한 즉각적인 가비지 수집 작동을 강요할 수 있다.

<704> 14.7 정적 파일들의 가비지 수집(Garbage Collection of Static Files)

<705> 파일 가비지 수집 큐 내의 엔트리를 갖는 정적 파일은 파일 내의 데이터의 일부의 갱신 다음에 클로우즈된다. 파일에 대한 파일 블록은 일부의 유효 데이터 및 일부의 쓸모없는 데이터를 포함하며, 프로그램 블록은 일부의 유효 데이터, 아마도 일부의 쓸모없는 데이터, 및 아마도 일부의 소거된 용량을 포함한다.

<706> 파일 가비지 수집 작동은 파일에 대한 모든 유효 데이터를 단일 블록에 통합시킨다. 프로그램 블록이 쓸모없는 데이터를 포함하지 않는 경우, 유효 데이터는 파일 블록으로부터 프로그램 블록으로 카피되고, 파일 블록은 소거된다. 프로그램 블록이 쓸모없는 데이터를 포함하는 경우, 파일 블록 및 프로그램 블록 둘 모두로부터의 모든 유효 데이터가 소거된 블록으로 카피되고, 파일 블록 및 프로그램 블록 둘 모두는 소거된다.

<707> 파일 가비지 수집은 가비지 수집-스케줄링 알고리즘에 의해 결정된 시간에서, 엔트리가 큐의 헤드에 도달할 때 수행된다. 그러나, 호스트는 자신이 파일을 클로우즈할 때 파일에 대한 즉각적인 가비지 수집 작동을 강요할 수 있다. 호스트는 파일에 대한 클로우즈 명령 직후에 유휴 명령을 전송함으로써 이를 행하며, 이 명령은 또 다른 명령이 수신될 때까지, 장치가 가비지 수집 또는 블록 통합 작동들을 연속적으로 수행하도록 한다. 호스트는 또 다른 명령을 전송하기 이전에, 장치가 내부 작동들을 수행하는데 있어서 더 이상 작동중이지 않다는 것을 자신이 검출할 때까지, 장치의 내부 작동중인 상태들을 모니터한다. 이 메커니즘에 의하여, 파일이 클로우즈된 직후의 파일에 대한 프로그램 블록들 및 파일의 통합이 호스트에 의해 보장될 수 있다.

<708> 상술된 설명에 따른 예시적인 메모리 시스템의 개요

<709> 직접적인 데이터 파일 플랫폼

<710> 직접적인 데이터 파일 플랫폼은 플래시 메모리에서 데이터 저장소를 관리하기 위한 보편적인 백-엔드 시스템으로서 작동한다.

<711> 직접적인 데이터 파일 인터페이스는 데이터의 다수의 소스들을 지원하는 내부 파일 저장 인터페이스이다.

<712> 미리 규정된 길이 없이 파일 데이터의 랜덤 판독/기록 액세스를 갖는 파일 액세스 인터페이스.

<713> 미리 규정된 길이를 갖는 완전한 파일 오브젝트들의 전달을 갖는 오브젝트 인터페이스.

<714> 파일 시스템을 통합한 종래의 호스트들로의 LBA 인터페이스. 논리적 블록들은 논리적 파일들로서 저장된다.

<715> 파일 내의 데이터로의 랜덤 액세스를 갖는 내장된 응용 프로그램.

<716> 직접적인 데이터 파일 저장소는 파일마다 데이터 저장소를 구성하는 백-엔드 시스템이다.

<717> 저장 장치에 대한 논리적 어드레스 공간이 없음.

<718> 파일 시스템이 없음.

<719> 직접적인 데이터 파일 대 종래의 시스템들

<720> 직접적인 데이터 파일 플랫폼은 종래 시스템들에 비하여 이점들을 제공한다:

<721> 높은 데이터 기록 속도:

<722> 파일 단편화로 인한 점진적인 속도 감소가 제거된다;

<723> 호스트에 의해 삭제된 파일들이 백그라운드 작동에서 소거될 때 피크 데이터 기록 속도가 증가될 수 있다.

<724> 데이터 기록 속도의 균일성:

<725> 가비지 수집이 백그라운드 또는 호스트 데이터의 기록과 인터리빙된 버스트들에서 수행될 때, 데이터를 스트리밍하는 일관된 기록 속도가 개선될 수 있다.

<726> 이점들(benefits)은 직접적인 데이터 파일 플랫폼에서 사용된 알고리즘들의 특성들의 결과이다:

<727> 제한된 파일 단편화

<728> 제한된 파일 및 블록 통합

<729> 정확한 파일 삭제

<730> 최적의 파일 데이터 인덱싱

<731> 효율적인 가비지 수집

<732> 직접적인 데이터 파일 인터페이스-바람직한 특성들

<733> 직접적인 데이터 파일 인터페이스는 호스트 내의 운영 시스템과 무관해야 한다:

<734> 수치적인 식별자를 갖는 파일들은 플랫 계층구조(flat hierachy)에서 관리된다;

<735> 파일과 관련된 데이터는 인터페이스 위의 레벨에서 계층적 디렉토리의 구성 및 유지보수를 허용하기 위하여 저장될 수 있다.

<736> 직접적인 데이터 파일 인터페이스는 바람직하게는 파일 데이터 전달의 다양한 포맷들을 지원한다:

<737> 데이터가 스트리밍될 수 있는 자신의 크기가 규정되지 않은 파일들;

<738> 기록되기 전에 자신의 크기가 규정되는 파일들;

<739> 자신의 크기가 고정되고 영구적으로 존재하는 파일들.

<740> 직접적인 데이터 파일 인터페이스 - 구현예

<741> 파일 내의 데이터는 1 바이트의 단위로, 랜덤 기록 및 판독 액세스를 갖는다.

<742> 데이터는 파일에 대한 기존 데이터에 겹쳐쓰기되거나 상기 기존 데이터 내에 삽입되도록 추가될 수 있다.

<743> 기록되거나 판독되는 파일 데이터는 미리 규정된 길이 없이 장치로 또는 상기 장치로부터 스트리밍된다.

<744> 현재 작동은 또 다른 명령의 수신에 의하여 종료된다.

<745> 파일들은 데이터 기록을 위해 오픈되고 파일의 종점에서, 또는 파일이 비활성일 때 클로우즈된다.

<746> 파일 핸들은 호스트에 의해 지정된 파일들에 대해 장치에 의하여 리턴된다.

<747> 계층적 디렉토리가 지원되지만 유지되지 않는다.

<748> 파일에 대한 관련된 정보가 저장될 수 있다.

<749> 장치가 백그라운드에서 내부 작동들을 수행할 수 있는 상태는 호스트에 의해 개시될 수 있다.

<750> 직접적인 데이터 파일 인터페이스 - 명령 세트

<751> 파일 명령들:

<752> 파일 오브젝트들을 제어하기 위한 명령들,

<753> 생성, 오픈, 클로우즈, 삭제, 소거, 리스트_파일

<754> 데이터 명령들:

<755> 파일 데이터를 기록 및 판독하기 위한 명령들,

<756> 기록, 삽입, 제거, 판독, 저장_버퍼, 기록_포인터, 판독_포인터.

<757> 정보 명령들:

<758> 파일과 관련된 정보를 기록 및 판독하기 위한 명령들,

<759> 기록_정보, 판독_정보, 정보_기록_포인터, 정보_판독_포인터.

<760> 상태 명령들:

<761> 장치의 상태를 제어하기 위한 명령들,

<762> 유휴, 대기, 셧-다운.

<763> 장치 명령들:

<764> 장치에 질문하기 위한 명령들,

<765> 용량, 상태.

<766> 파일-대-플래시 맵핑 알고리즘

<767> 데이터 구조들:

<768> 파일들

<769> 데이터 그룹들

<770> 블록 유형들:

<771> 프로그램 블록들

<772> 파일 블록들

<773> 공통 블록들

<774> 파일 유형들:

<775> 플레인 파일

<776> 공통 파일

<777> 편집된 파일

<778> 메모리 복구:

<779> 가비지 수집

<780> 블록 통합

<781> 파일-대-플래시 맵핑 알고리즘 - 데이터 구조들

<782> 파일들:

<783> 파일은 호스트에 의해 생성되고 유지되는 데이터의 세트이다;

<784> 호스트는 외부 호스트이거나 메모리 카드 내의 응용 프로그램일 수 있다;

<785> 파일은 호스트에 의해 생성된 파일이름, 또는 직접적인 데이터 파일 플랫폼에 의해 생성된 파일 핸들에 의해 식별된다;

<786> 파일 내의 데이터는 파일 오프셋 어드레스들에 의해 식별된다;

<787> 상이한 파일들에 대한 오프셋 어드레스들의 세트들은 장치 내에서 독립적인 논리적 어드레스 공간들로서 작동한다. 장치 자체에 대한 논리적 어드레스 공간은 존재하지 않는다.

<788> 데이터 그룹들;

<789> 데이터 그룹은 파일 내의 인접한 오프셋 어드레스들을 갖는 단일 파일에 대한 데이터의 세트이다;

<790> 데이터 그룹은 단일 블록 내의 인접한 물리적 어드레스들에서 저장된다;

<791> 데이터 그룹은 한 바이트 및 한 블록 사이의 임의의 길이를 가질 수 있다:

<792> 데이터 그룹은 논리적 파일 어드레스를 물리적 플래시 어드레스로 맵핑하기 위한 기본 단위이다.

<793> 파일-대-플래시 맵핑 알고리즘 - 블록 유형들

<794> 프로그램 블록들:

<795> 호스트에 의해 기록된 모든 데이터는 프로그램 블록에서 프로그래밍된다;

<796> 프로그램 블록은 단일 파일에 대한 데이터에 전용된다;

<797> 프로그램 블록 내의 파일 데이터는 파일 오프셋 어드레스의 임의의 순서일 수 있고, 프로그램 블록은 파일에 대한 다수의 데이터 그룹들을 포함할 수 있다;

<798> 각각의 오픈 파일, 및 지정되지 않은 수의 클로우즈된 파일들에 대하여 별도의 프로그램 블록들이 존재한다.

<799> 파일 블록들:

<800> 프로그램 블록은 자신의 최종 위치가 프로그래밍될 때 파일 블록이 된다.

<801> 공통 블록들:

<802> 공통 블록은 하나 이상의 파일에 대한 데이터 그룹들을 포함한다.

<803> 공통 블록은 공통 블록의 가비지 수집 동안, 또는 블록 통합 작동 동안 관련되지 않은 파일들에 대한 데이터 그룹들을 프로그램 블록에 프로그래밍함으로써 생성된다;

<804> 데이터 그룹들은 또 다른 공통 블록의 가비지 수집 동안, 또는 블록 통합 작동 동안 공통 블록에 기록될 수 있다.

<805> 파일-대-플래시 맵핑 알고리즘 - 파일 유형들

<806> 플레인 파일(도3A 참조):

<807> 플레인 파일은 임의의 수의 완전한 파일 블록들 및 하나의 부분적으로 기록된 프로그램 블록을 포함한다.

<808> 플레인 파일은 자신의 소거 이전에 임의의 블록으로부터 데이터를 재배치할 필요 없이 삭제될 수 있다.

<809> 공통 파일(도3B 참조):

<810> 공통 파일은 임의의 수의 완전한 파일 블록들 및 다른 관련되지 않은 파일들에 대한 데이터와 함께 그 파일에 대한 데이터를 포함하는 하나의 공통 블록을 포함한다.

<811> 파일이 삭제된 후에, 공통 블록에 대해서만 가비지 수집 작동이 수행되어야 한다.

<812> 편집된 파일(도3C 및 3D 참조)

<813> 편집된 파일은 겹쳐쓰기된 기존의 오프셋 어드레스에서의 데이터의 결과로서, 자신의 블록들 중 하나 이상에서 쓸모없는 데이터를 포함한다.

<814> 쓸모없는 데이터에 의해 차지된 메모리 용량은 파일 가비지 수집 작동에 의해 복구될 수 있다.

<815> 파일 가비지 수집 작동은 편집된 파일을 플레인 파일 포맷으로 복구시킨다.

<816> 파일-대-플래시 맵핑 알고리즘 - 메모리 복구

<817> 가비지 수집:

<818> 가비지 수집 작동은 쓸모없는 데이터에 의해 차지된 메모리 용량을 복구하기 위하여 수행된다.

<819> 진행중인 작동들은 가비지 수집 큐들에서 로깅되고, 그 후에 스케줄링 알고리즘들에 따라 최적의 레이트로 수행된다.

<820> 가비지 수집은 호스트 인터페이스가 휴지인 동안, 호스트 명령에 의해 개시되고 백그라운드에서 수행될 수 있다. 작동들은 임의의 다른 호스트 명령의 수신 시에 중지된다.

<821> 가비지 수집은 또한 호스트 데이터 기록 작동들과 인터리빙된 버스트들에서, 포어그라운드 작동들로서 수행될 수 있다.

<822> 블록 통합:

<823> 블록 통합의 진행중인 프로세스는 프로그램 블록들 및 공통 블록들에서 록업되는 소거된 용량을 복구하기 위하여 구현될 수 있다.

<824> 프로그램 블록들 내의 파일 데이터의 용량들 및 공통 블록 내의 삭제된 파일들에 대한 쓸모없는 데이터의 용량들의 분포들이 불균형인 경우에만 필요로 된다.

<825> 다수의 프로그램 또는 공통 블록들 내의 데이터는 하나 이상의 블록들을 소거하도록 하기 위하여 통합된다.

<826> 파일 데이터 프로그래밍

<827> 파일 핸들러에 의해 식별된 파일에 대한 데이터는 삽입 또는 기록 명령 다음에 자신이 호스트로부터 스트리밍될 때, 플래시 메모리에 프로그래밍된다.

<828> 데이터의 최초 파일 오프셋 어드레스는 그 값이 호스트에 의해 설정될 수 있는 기록 포인터에 의해 규정된다.

<829> 충분한 데이터가 버퍼 메모리에서 축적되었을 때, 메타페이지가 파일에 대한 프로그램 블록에서 프로그래밍된다.

<830> 프로그램 블록은 가득 찰 때, 파일 블록으로서 지정되고, 소거된 블록은 파일에 대한 새로운 프로그램 블록으로서 할당된다.

<831> 데이터 그룹 인덱싱 구조들은 프로그램 블록이 가득 찰 때마다, 또는 또 다른 호스트 명령이 수신될 때마다 플래시 메모리에서 갱신된다.

<832> 파일 데이터 프로그래밍 절차는 적응형 스케줄링 알고리즘에 의해 결정되는 호스트 데이터 스트림에서 간격을 두고 포어그라운드 가비지 수집의 버스트들을 개시한다.

<833> 파일 데이터 프로그래밍 절차는 또 다른 호스트 명령이 수신될 때 종료된다.

<834> 파일 데이터 판독

<835> 파일 핸들에 의해 식별된 파일에 대한 데이터는 판독 명령 다음에, 플래시 메모리로부터 판독되고 호스트로 스트리밍된다.

<836> 데이터의 최초 파일 오프셋 어드레스는 그 값이 호스트에 의해 설정될 수 있는 판독 포인터에 의해 규정된다.

<837> 파일 데이터는 파일의 종점에 도달될 때까지, 또는 또 다른 호스트 명령이 수신될 때까지 하나의 메타페이지의 단위들로 판독된다.

<838> 데이터는 파일 오프셋 어드레스 순서로 호스트에 전달된다.

<839> 파일에 대해 판독될 데이터 그룹의 위치는 파일 인덱싱 구조들에 의해 규정된다.

<840> 파일 데이터 판독 절차는 또 다른 호스트 명령이 수신될 때 종료된다.

<841> 파일 삭제

<842> 파일에 대한 삭제 명령에 응답하여, 파일에 대한 데이터를 포함하는 블록들이 식별되고 후속 가비지 수집 작동들을 위해 가비지 수집 큐들에 추가된다.

<843> 파일에 대한 엔트리들을 제거하기 위하여, 파일 디렉토리 및 파일 인덱스 테이블이 갱신된다.

<844> 파일을 삭제하기 위한 절차는 가비지 수집 작동들을 개시하지 않고, 파일에 대한 데이터는 즉시 소거되지 않는다.

<845> 파일에 대한 소거 명령에 응답하여, 삭제 명령에 대해 동일한 절차가 뒤따르지만, 임의의 다른 호스트 명령이 실행되기 전에 가비지 수집 작동들이 개시되고 완료된다.

<846> 가비지 수집

<847> 가비지 수집은 쓸모없는 데이터에 의해 차지된 플래시 용량을 복구하기 위한 작동이다.

<848> 오브젝트들은 후속 가비지 수집 작동들을 규정하기 위하여 장치의 작동 동안 때때로 3개의 가비지 수집 큐들에 추가된다:

<849> *쓸모없는 블록 큐* - 블록이 파일 데이터의 갱신 또는 파일의 삭제의 결과로서 완전히 쓸모없게 될 때, 상기 블록은 이 큐에 추가된다.

<850> *공통 블록 큐* - 다수의 파일들에 대한 데이터를 포함하는 블록의 일부의 데이터가 파일 데이터 갱신, 파일의 삭제, 또는 파일의 가비지 수집의 결과로서 쓸모없게 될 때, 상기 데이터는 이 큐에 추가된다.

<851> *파일 큐* - 파일이 호스트에 의해 클로우즈될 때, 상기 파일은 이 큐에 추가된다. 오브젝트들은 우선순위 가비지 수집에 대해 지정될 수 있다.

<852> 가비지 수집 작동들은 2개의 방식들로 스케줄링될 수 있다:

<853> 백그라운드 작동들은 호스트가 장치로의 판독 또는 기록 액세스를 행하고 있지 않을 때, 호스트에 의해 개시될 수 있다.

<854> 포어그라운드 작동들은 직접적인 데이터 파일 플랫폼이 호스트에 의해 액세스되고 있을 때 직접적인 데이터 파일 플랫폼에 의해 개시될 수 있다.

<855> 가비지 수집 - 스케줄링

<856> 백그라운드 가비지 수집은 호스트에 의해 개시된다. 장치가 내부 작동들을 수행하도록 허용받는 유휴 상태는 직접적인 데이터 파일 인터페이스에서 특정 명령을 통하여 호스트에 의해 개시된다. 가비지 수집 큐들로부터의 오브젝트들의 가비지 수집은 유휴 상태가 지속되는 동안 계속된다. 가비지 수집은 임의의 명령이 호스트로부터 수신될 때 중지된다. 호스트는 다음 명령을 전송하기 전에 가비지 수집 작동들이 완료되도록 하기 위하여 선택적으로 장치의 작동중인 상태를 모니터할 수 있다.

<857> 포어그라운드 가비지 수집은 호스트가 백그라운드 작동들을 개시하지 않았을 때, 직접적인 데이터 파일 플랫폼에 의해 개시된다. 가비지 수집은 적응형 알고리즘에 따라 스케줄링된다. 현재 가비지 수집 작동을 위한 프로그램 및 소거 작동들의 버스트들은 호스트로부터 수신된 파일 데이터에 대한 프로그램 작동들의 버스트들과 인터리빙된다. 버스트들의 길이들은 인터리빙된 가비지 수집의 듀티 사이클을 규정하기 위하여 적응형으로 제어될 수 있다.

<858> 가비지 수집 - 적응형 스케줄링(도8B참조)

<859> 플래시 메모리는 통상적으로 프로그램 블록들, 공통 블록들 및 쓸모없는 파일 블록들에 포함된 추가적인 호스트 데이터를 기록하는데 필요로 되는 복구 가능한 용량을 갖는다.

<860> 적응형 가비지 수집은 추가적인 호스트 데이터를 프로그래밍하고 이전에 기록된 호스트 데이터를 재배치하는 인터리브 비율을 제어한다. 복구 가능한 용량은 이를 소거된 용량으로 변환함으로써 새로운 호스트 데이터에 이용 가능하게 된다. 가비지 수집 레이트는 적응형 기간에 걸쳐 일정하게 유지된다.

<861> 가비지 수집 - 작동들의 우선순위

<862> 스케줄링된 가비지 수집을 위한 작동은 다음의 우선순위 순서에 따라 가비지 수집 큐들로부터 선택된다:

<863> 1. 쓸모없는 블록 우선순위 가비지 수집:

<864> 파일 소거 명령의 결과로서 생성된 쓸모없는 블록에 대한 다음 엔트리가 선택된다.

<865> 2. 공통 블록 우선순위 가비지 수집:

<866> 파일 소거 명령의 결과로서 생성된 부분적으로 쓸모없는 공통 블록에 대한 다음 엔트리가 선택된다.

<867> 3. 쓸모없는 블록 가비지 수집:

<868> 쓸모없는 블록에 대한 다음 엔트리가 선택된다.

<869> 4. 공통 블록 가비지 수집:

<870> 부분적으로-쓸모없는 공통 블록에 대한 다음 엔트리가 선택된다.

<871> 5. 파일 가비지 수집:

<872> 부분적으로 쓸모없는 파일에 대한 다음 엔트리가 선택된다.

<873> 6. 블록 통합:

<874> 가비지 수집 큐들에서 엔트리들이 존재하지 않을 때, 블록 통합 작동을 위해 소스 블록 및 목적지 블록이 선택된다.

<875> 가비지 수집 - 공통 블록 가비지 수집

<876> 유효 파일들은 프로그램 블록 또는 공통 블록 중 하나에서 어떤 데이터를 포함한다.

<877> 파일이 삭제될 때, 파일에 대한 쓸모없는 데이터를 포함하는 임의의 공통 블록은 공통 블록 가비지 수집 작동을 겪는다.

<878> 관련되지 않은 파일들에 대한 데이터 그룹은 또 다른 공통 블록 또는 프로그램 블록에 재배치된다(도8G1 내지 8G4 참조).

<879> 공통 블록 가비지 수집 작동 동안, 유효 파일 그룹들이 소스 공통 블록으로부터 하나 이상의 선택된 목적지 블록들로 재배치된다.

<880> 목적지 블록은 각각의 파일 그룹에 대해 개별적으로 선택된다.

<881> 목적지 블록의 선택에 대한 우선순위들은 다음과 같다:

<882> 1. 소스 파일 그룹이 재배치되는데 가장 적합한 이용 가능한 소거된 용량을 갖는 공통 블록;

<883> 2. 소스 파일 그룹이 재배치되는데 가장 적합한 이용 가능한 소거된 용량을 갖는 프로그램 블록; 및

<884> 3. 나중에 프로그램 블록으로 지정되는 소거된 블록.

<885> 가비지 수집 - 파일 가비지 수집

<886> 파일 가비지 수집은 파일에 대한 쓸모없는 데이터에 의해 차지된 용량을 복구하기 위하여, 파일이 클로우즈된 후에 수행된다. 이것은 파일에 대한 데이터가 편집 동안 겹쳐쓰기되었던 경우에만 필요하다.

<887> 편집된 플레인 파일 상태 또는 편집된 공통 파일 상태의 파일은 (단일 프로그램 블록을 포함하고 공통 블록을 포함하지 않는) 플레인 파일 상태로 복구된다.

<888> 파일 가비지 수집은 유효 데이터 그룹들을 쓸모없는 데이터를 포함하는 블록으로부터 파일에 대한 프로그램 블록으로 카피함으로써 수행된다.

<889> 데이터 그룹들은 파일의 종점에서의 랩-어라운드(wrap-around)와 함께, 최초 프로그램 포인터 다음에 오프셋 어드레스로부터 순차적인 순서로 카피된다.

<890> 가비지 수집 - 블록 통합

<891> 블록 통합 작동 동안, 유효 파일 그룹들은 선택된 소스 블록으로부터 하나 이상의 선택된 목적지 블록들로 재배치된다.

<892> 소스 블록은 데이터의 최저 용량을 갖는 공통 블록 또는 프로그램 블록으로서 선택된다.

<893> 목적지 블록은 각각의 파일 그룹에 대해 개별적으로 선택된다.

<894> 목적지 블록의 선택에 대한 우선순위들은 다음과 같다:

<895> 1. 소스 파일 그룹이 재배치되는데 가장 적합한 이용 가능한 소거된 용량을 갖는 공통 블록.

<896> 2. 소스 파일 그룹이 재배치되는데 가장 적합한 이용 가능한 소거된 용량을 갖는 프로그램 블록.

<897> 3. 파일 그룹의 일부가 기록되는 최고의 이용 가능한 소거된 용량을 갖는 프로그램 블록 또는 공통 블록. 이 상황에서, 파일이 다른 관련되지 않은 파일들과 2개의 블록들을 공유하는 것이 허용 가능하다.

<898> 4. 파일 그룹의 나머지가 기록되는, 소스 파일 그룹의 나머지에 가장 적합한 이용 가능한 소거된 용량을 갖는 제2 프로그램 블록 또는 공통 블록.

<899> 5. 파일 그룹의 나머지가 기록되는, 나중에 프로그램 블록으로 지정되는 소거된 블록.

<900> 파일 인덱싱(도10A 참조)

<901> 파일은 자신이 호스트에 의해 생성될 때, 직접적인 데이터 파일 장치에 의해 할당되는 파일ID에 의해 식별된다.

<902> 플랫 디렉토리는 각각의 파일ID에 대한 파일 데이터 포인터 및 파일 정보 포인터를 지정한다.

<903> 파일 데이터 포인터는 파일 인덱스 테이블에서 엔트리들의 세트를 식별하며, 상기 각각의 엔트리는 세트가 관련되는 파일에 대한 데이터 그룹을 지정한다.

<904> 파일 정보 포인터는 정보 테이블에서 파일에 대한 정보의 스트링을 식별한다:

<905> 파일_정보는 호스트에 의해 기록되고 직접적인 데이터 파일 장치에 의해 해석되지 않는다.

<906> 파일_정보는 파일이름, 페어런트 디렉토리, 차일드 디렉토리들, 속성들, 권리 정보, 및 파일에 대한 파일 어소

시에이션을 포함할 수 있다.

<907> 파일 인덱싱 - 인덱싱 구조들

<908> 도10B 참조

<909> 파일 인덱싱 - 파인 인덱스 테이블(FIT) - 도10D 참조

<910> FIT는 플래시 메모리 내의 파일들에 대한 모든 유효 데이터 그룹들에 대한 엔트리들을 포함한다. 쓸모없는 데이터 그룹들은 FIT에 의해 인덱싱되지 않는다.

<911> FIT는 각각이 물리적 블록으로 맵핑되는 논리적 범위들로 분할된다.

<912> FIT 파일은 파일 오프셋 어드레스 순서의, 파일에 대한 연속적인 엔트리들의 세트이다.

<913> FIT 파일은 물리적 블록 및 논리적 파일 번호를 규정하는 FIT 파일 포인터에 의해 식별된다.

<914> 파일 인덱싱 - 파일 인덱스들 갱신(도10F 및 10G 참조)

<915> 파일 인덱스 테이블 및 정보 테이블에 대해 동일한 구조가 사용된다.

<916> 블록 리스트는 논리적 파일 데이터 포인터를 물리적 FIT 블록 또는 FIT 갱신 블록 내의 FIT 파일들과 관련시키는데 사용된다.

<917> FIT 파일들은 압축된 포맷으로 FIT 블록에 저장된다.

<918> FIT 파일들의 갱신된 버전들은 페이지 내의 단일 FIT 파일과 함께, 공유된 FIT 갱신 블록에 저장된다.

<919> FIT 갱신 블록의 압축 및 FIT 블록 내의 FIT 파일들의 압축은 때때로 수행된다.

<920> 파일 인덱싱 - 인덱스 페이지 포맷(도10E 참조)

<921> FIT 블록, FIT 갱신 블록, 정보 블록, 및 정보 갱신 블록에 대해 동일한 구조가 사용된다.

<922> 정보는 한 페이지의 단위들로 프로그래밍된다.

<923> 페이지는 2개의 에어리어들, 즉, FIT 엔트리들에 대한 에어리어 및 파일 포인터에 대한 에어리어로 세분된다.

<924> 파일 포인터는 범위 내의 논리적 파일 번호를 대응하는 FIT 파일의 시작점에 대한 엔트리 번호 및 페이지 번호로 번역한다.

<925> FIT 파일은 물리적으로 연속적인 FIT 엔트리들을 포함한다.

<926> 데이터 버퍼링 및 프로그래밍

<927> 호스트에 의해 기록되거나 플래시 메모리에 재배치되는 데이터는 한 세트의 섹터 버퍼들에서 버퍼링된다.

<928> 데이터 그룹 경계들의 레졸루션(resolution)은 1 바이트이지만, 데이터는 ECC 발생 및 검사를 위해, 한 섹터의 집합들로서 플래시로 그리고 상기 플래시로부터 전달된다.

<929> 버퍼로부터의 데이터는 가능한 경우, 메타페이지의 단위들로 플래시에서 프로그래밍된다.

<930> 버퍼 플러시 작동은 파일이 클로우즈되거나 셧다운이 진행중일 때, 페이지의 일부만을 프로그래밍한다. 파일 인덱싱 기술들은 페이지의 프로그래밍되지 않은 부분이 존속하도록 한다.

<931> 버퍼 스왑-아웃 작동은 버퍼 내의 파일 데이터가 버퍼 내의 데이터의 백-업 및 버퍼 공간의 관리를 위해, 일시적으로 공통 스왑 블록에 저장되도록 한다.

<932> 프로그램 블록 또는 공통 블록 내의 파일 그룹의 시작점은 메타페이지의 시작점과 정렬된다.

<933> 플래시에서 대부분의 데이터 재배치를 위해 온-칩 카피가 사용될 수 있다.

<934> 블록 상태 관리

<935> 직접적인 데이터 파일 시스템은 데이터의 저장과 관련된 블록들에 대해 8개의 상태들을 유지한다(도11A 참조).

<936> 소거된 블록 관리

<937> 직접적인 데이터 파일은 파일들에 대한 모든 데이터 및 모든 제어 정보를 고정된 크기의 메타블록들에

저장한다. (용어 "블록"은 종종 "메타블록"을 나타내는데 사용된다.).

<938> 소거된 블록들을 블록들 내로 링킹하는 방법은 다음의 계류중인 미국 특허 출원들: 2003년 12월 30일자로 출원되고 명칭이 "Management of Non-Volatile Memory Systems Having Large Erase Blocks"인 일련 번호 10/749,831; 2003년 12월 30일자로 출원되고 명칭이 "Non-Volatile Memory and Method with Block Management System"인 일련 번호 10/750,155; 2004년 8월 13일자로 출원되고 명칭이 "Non-Volatile Memory and Method with Memory Planes Alignment"인 일련 번호 10/917,888; 2004년 8월 13일자로 출원된 일련 번호 10/917,867; 2004년 8월 13일자로 출원되고 명칭이 "Non-Volatile Memory and Method with Phased Program Failure Handling"인 일련 번호 10/917,889; 2004년 8월 13일자로 출원되고 명칭이 "Non-Volatile Memory and Method with Control Data Management"인 일련 번호 10/917,725; 2005년 7월 27일자로 출원되고 명칭이 "Non-Volatile Memory and Method with Multi-Stream Update Tracking"인 일련 번호 11/192,200; 2005년 7월 27일자로 출원되고 명칭이 "Non-Volatile Memory and Method with Improved Indexing for Scratch Pad and Update Blocks"인 일련 번호 11/192,386; 및 2005년 7월 27일자로 출원되고 명칭이 "Non-Volatile Memory and Method with Multi-Stream Updating"인 일련 번호 11/191,686에 설명되어 있는 논리적 어드레스 공간(LBA)을 갖는 시스템에서 사용되는 것으로부터 변화되지 않는다.

<939> 데이터 및 제어 정보를 저장하기 위한 할당에 이용 가능한 소거된 블록들은 소거된 블록 풀에 수용된다.

<940> 소거된 블록들은 소거된 블록 로그 내의 엔트리들로서 레코딩된다.

<941> 할당을 위한 소거된 블록은 로그의 헤드에서 엔트리로서 선택된다.

<942> 엔트리는 블록이 소거될 때, 로그의 테일에서 추가된다.

<943> 제어 데이터 구조들

<944> 제어 데이터 구조들은 전용된 제어 블록에 저장된다.

<945> 제어 정보는 4개의 독립적인 로그들에 저장된다. 각각의 로그는 제어 블록에서 하나 이상의 페이지들을 차지한다. 유효 로그 페이지들은 기록된 최종 페이지에서 로그 포인터들에 의해 추적된다.

<946> 공통 블록 로그는 이들이 포함하는 이용 가능한 소거된 용량의 순서로, 플래시 메모리에 존재하는 모든 공통 블록들에 대한 엔트리들을 포함한다.

<947> 프로그램 블록 로그는 이들이 포함하는 이용 가능한 소거된 용량의 순서로, 플래시 메모리에 존재하는 모든 프로그램 블록들에 대한 엔트리들을 포함한다.

<948> 소거된 블록 로그는 이들의 소거의 시퀀스의 순서로, 플래시 메모리에 존재하는 모든 소거된 블록들에 대한 엔트리들을 포함한다.

<949> 제어 로그는 제어 파라미터들, 카운트들 및 리스트들에 대한 미리 규정된 필드들을 포함한다.

<950> 로그는 제어 블록 내의 다음의 소거된 페이지 위치에서 완전한 로그의 갱신된 버전을 기록함으로써 갱신된다.

도면의 간단한 설명

<21> 다음의 목록화된 도면들은 본 출원의 부분으로서 포함되며 이하의 설명들에서 참조된다.

<22> 도1A는 직접적인 데이터 파일 플랫폼을 갖는 메모리 카드를 도시한 도면.

<23> 도1B는 직접적인 데이터 파일 플랫폼 컴포넌트들을 도시한 도면.

<24> 도2A는 파일 명령들을 도시한 도면.

<25> 도2B는 데이터 명령들을 도시한 도면.

<26> 도2C는 정보 명령들을 도시한 도면.

<27> 도2D는 스트림 명령들을 도시한 도면.

<28> 도2E는 상태 명령들을 도시한 도면.

<29> 도2F는 장치 명령들을 도시한 도면.

<30> 도3A는 플레인 파일(plain file)의 포맷을 도시한 도면.

<31> 도3B는 공통 파일의 포맷을 도시한 도면.

<32> 도3C는 편집된 플레인 파일에 대한 포맷을 도시한 도면.

<33> 도3D는 편집된 공통 파일의 포맷을 도시한 도면.

<34> 도4A는 장치 작동들에 대한 흐름도.

<35> 도5A는 파일 데이터를 프로그래밍하는 것에 대한 흐름도.

<36> 도6A는 파일 데이터를 판독하는 것에 대한 흐름도.

<37> 도7A는 파일을 삭제하는 것에 대한 흐름도.

<38> 도8A는 포어그라운드 가비지 수집(foreground garbage collection)에 대한 인터리빙된 작동들을 도시한 도면.

<39> 도8B는 가비지 수집의 적응형 스케줄링에 대한 작동의 원리를 도시한 도면.

<40> 도8C는 가비지 수집 선택에 대한 흐름도.

<41> 도8D는 파일 가비지 수집에 대한 흐름도.

<42> 도8E는 공통 블록 가비지 수집에 대한 흐름도.

<43> 도8F는 블록 통합에 대한 흐름도.

<44> 도8G1 내지 8G4는 4개의 시간 순차적인 단계들을 나타낸, 공통 블록 가비지 수집 예를 도시한 도면.

<45> 도9A는 연속적인 호스트 데이터 프로그래밍을 도시한 도면.

<46> 도9B는 중단된 호스트 데이터 프로그래밍을 도시한 도면.

<47> 도9C는 버퍼 플러시 프로그래밍을 도시한 도면.

<48> 도9D는 버퍼 스왑-아웃 프로그래밍을 도시한 도면.

<49> 도9E는 버퍼 플러시 이후의 호스트 데이터 프로그래밍을 도시한 도면.

<50> 도9F는 스왑-인 데이터 판독을 도시한 도면.

<51> 도9G는 버퍼 스왑-인 이후의 호스트 데이터 프로그래밍을 도시한 도면.

<52> 도9H는 버퍼에 대한 정렬된 데이터 판독을 도시한 도면.

<53> 도9I는 버퍼로부터의 정렬된 데이터 프로그래밍을 도시한 도면.

<54> 도9J는 버퍼에 대한 비-정렬된 데이터 판독을 도시한 도면.

<55> 도9K는 버퍼로부터의 비-정렬된 데이터 프로그래밍을 도시한 도면.

<56> 도9L은 버퍼에 대한 비-정렬된 비-순차적인 데이터 판독을 도시한 도면.

<57> 도9M은 버퍼로부터의 비정렬된 비-순차적인 데이터 프로그래밍을 도시한 도면.

<58> 도10A은 파일 인덱싱을 도시한 도면.

<59> 도10B는 파일 인덱싱 구조들을 도시한 도면.

<60> 도10C은 디렉토리 블록 포맷을 도시한 도면.

<61> 도10D는 파일 인덱스 테이블(FIT) 논리적 구조를 도시한 도면.

<62> 도10E는 FIT 페이지 포맷을 도시한 도면.

<63> 도10F은 물리적 FIT 블록들을 도시한 도면.

<64> 도10G은 FIT 파일 갱신 작동들의 예들을 도시한 도면.

<65> 도11A는 블록 상태도.

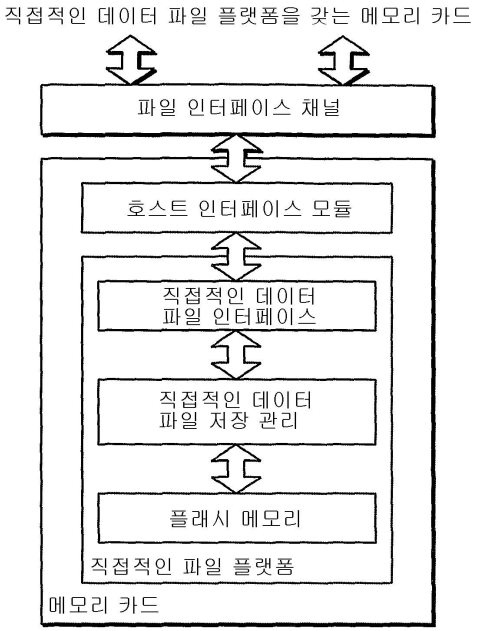
<66> 도12A는 제어 블록 포맷을 도시한 도면.

<67> 도12B는 공통 블록 로그 포맷을 도시한 도면.

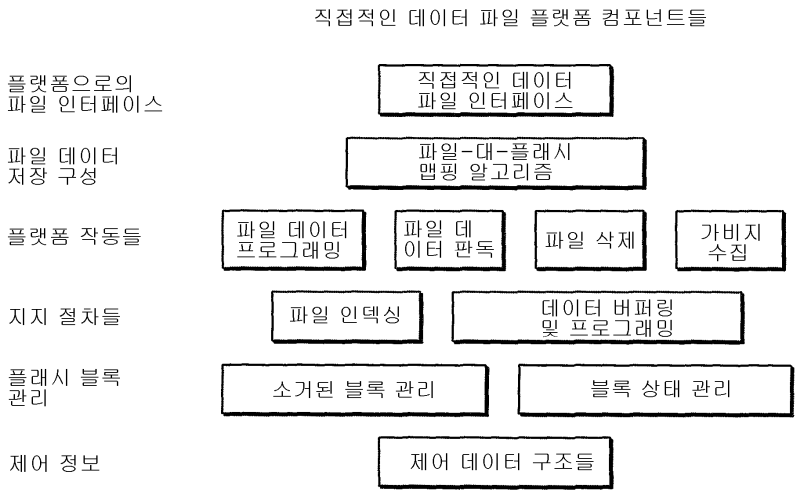
<68> 도13A는 (함께 도시되어야 하는 부분들(13A1 및 13A2)에서) 정적 파일과 함께 사용되는 명령 세트를 도시한 도면.

도면

도면1A



도면1B



도면2A

파일 명령들

명령	파라미터들	설명
생성	<파일 ID>	<파일 ID>장치 내의 디렉토리에서 <파일 ID>에 의해 식별되는 엔트리 생성 <파일 ID>가 명령에 의해 지정되지 않는 경우, 상기 <파일 ID>는 장치에 의해 할당되고 호스트로 리턴됨.
오픈	<파일 ID>	지정된 파일에 대한 후속 데이터 명령들의 실행을 인에이블시킴
클로우즈	<파일 ID>	지정된 파일에 대한 후속 데이터 명령들의 실행을 디스에이블시킴
삭제	<파일 ID>	지정된 파일에 대한 파일 정보 엔트리들과 파일 인덱스, 디렉토리가 삭제되어야 한다는 것을 나타냄. 파일 데이터는 소거될 수 있음.
소거	<파일 ID>	지정된 파일에 대한 파일 정보 엔트리들과 파일 인덱스, 디렉토리가 삭제되어야 하고 파일 데이터가 즉시 소거되어야 한다는 것을 나타냄.
리스트_파일		수치적인 순서로, 장치로부터 모든 유효한 <파일 ID> 값들 판독

도면2B

데이터 명령들

명령	파라미터들	설명
기록	<파일 ID>	기록_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일에서 데이터 겹쳐쓰기
삽입	<파일 ID>	기록_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일에서 데이터 삽입
제거	<파일 ID><길이>	기록_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일로부터 크기<길이>의 데이터 삭제
판독	<파일 ID>	판독_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일로부터 데이터 판독
저장_버퍼	<파일 ID>	플래시 메모리 내의 일시적 위치로 지정된 파일에 대한 버퍼 내의 데이터 저장
기록_포인터	<파일 ID> <오프셋>	<오프셋>지정된 파일에 대한 기록_포인터에 대한 새로운 현재 값 규정
판독_포인터	<파일 ID> <오프셋>	지정된 파일에 대한 판독_포인터에 대한 새로운 현재 값 규정

도면2C

정보 명령들

명령	파라미터	설명
기록_정보	<파일 ID>	정보_기록_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일에 대한 파일_정보 기록
판독_정보	<파일 ID>	정보_판독_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서 지정된 파일에 대한 파일_정보 판독
정보_기록_포인터	<파일 ID> <오프셋>	규정된 파일에 대한 정보_기록_포인터에 대한 새로운 현재 값 규정
정보_판독_포인터	<파일 ID> <오프셋>	규정된 파일에 대한 정보_판독_포인터에 대한 새로운 현재 값 규정

도면2D

스트림 명령들

명령	파라미터	설명
스트림	<길이>	장치로 또는 장치로부터 전달될 데이터의 인터럽트되지 않은 스트림 규정
중단	<시간>	다음 명령의 실행 이전에 삽입되는 지연 규정

도면2E

상태 명령들

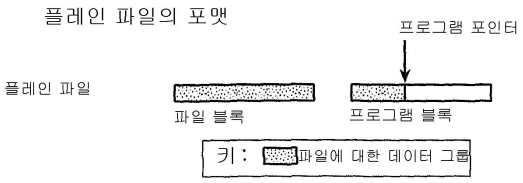
명령	파라미터	설명
유휴		장치는 유휴 상태에 진입해야 하고, 이 상태에서 상기 장치는 내부 작동들을 수행할 수 있다.
대기		장치는 대기 상태에 진입해야 하고, 이 상태에서 상기 장치는 내부 작동들을 수행할 수 없다.
셧다운		장치는 셧다운될 것이며, 장치가 다음에 작동중인 상태가 아닐 때 전력이 제거될 것이다.

도면2F

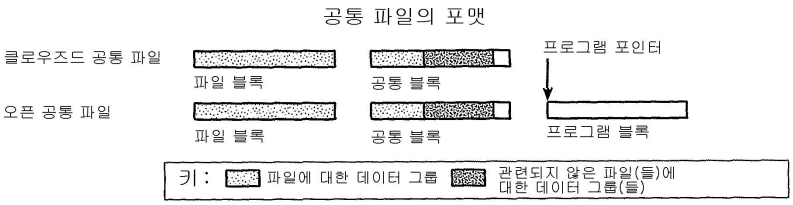
장치 명령들

명령	파라미터	설명
용량		장치가 파일 데이터에 의해 차지되고 새로운 파일 데이터에 이용 가능한 용량들을 보고하도록 호스트로부터 요청
상태		장치가 자신의 현재 상태를 보고하도록 호스트로부터 요청

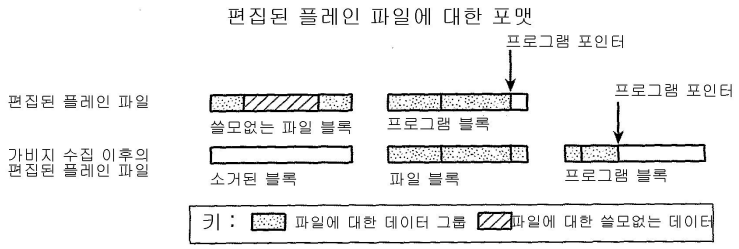
도면3A



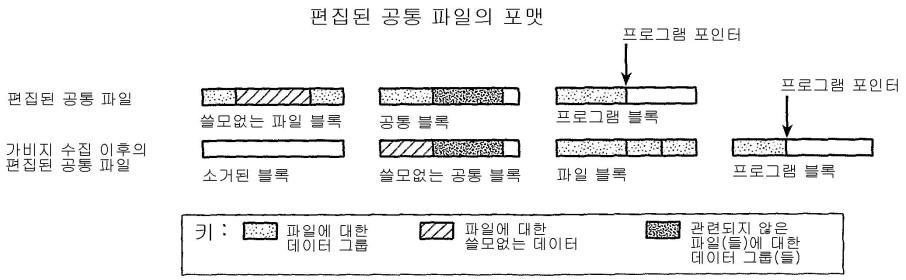
도면3B



도면3C

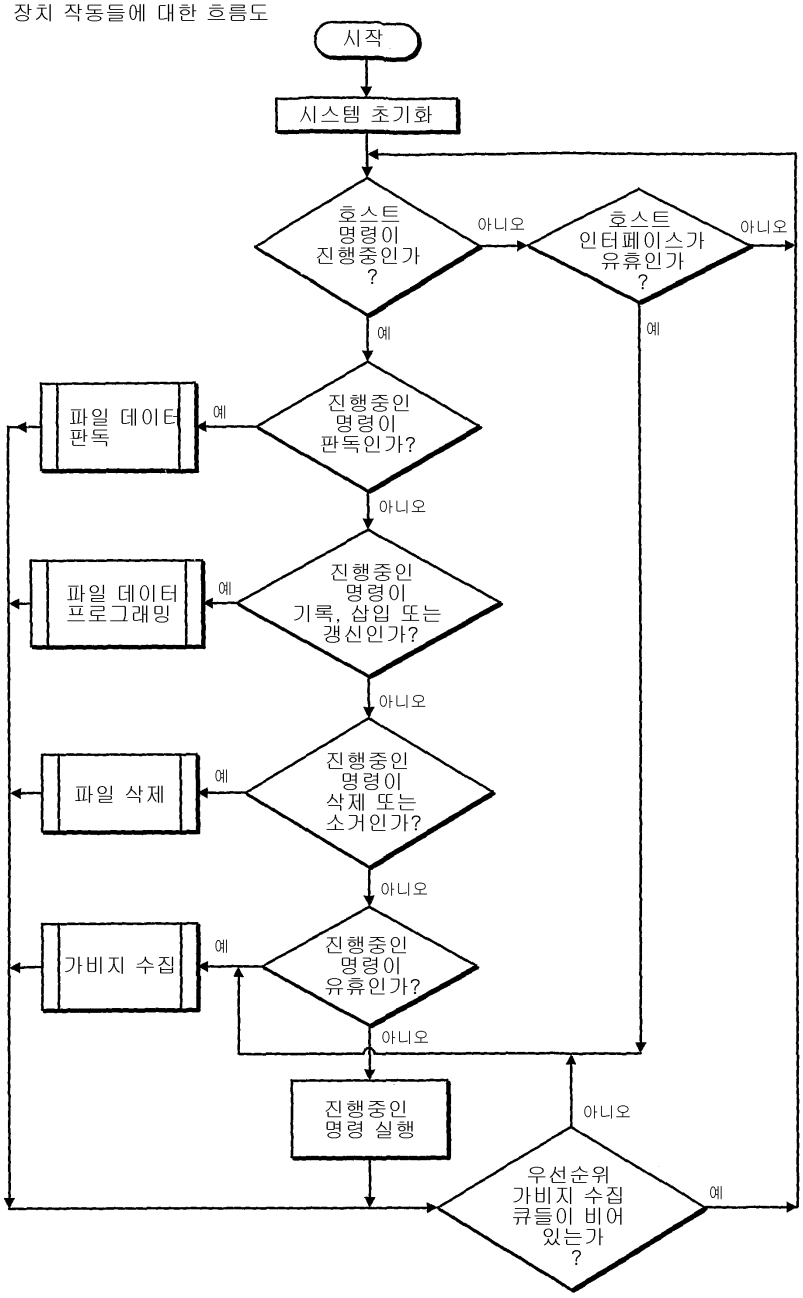


도면3D



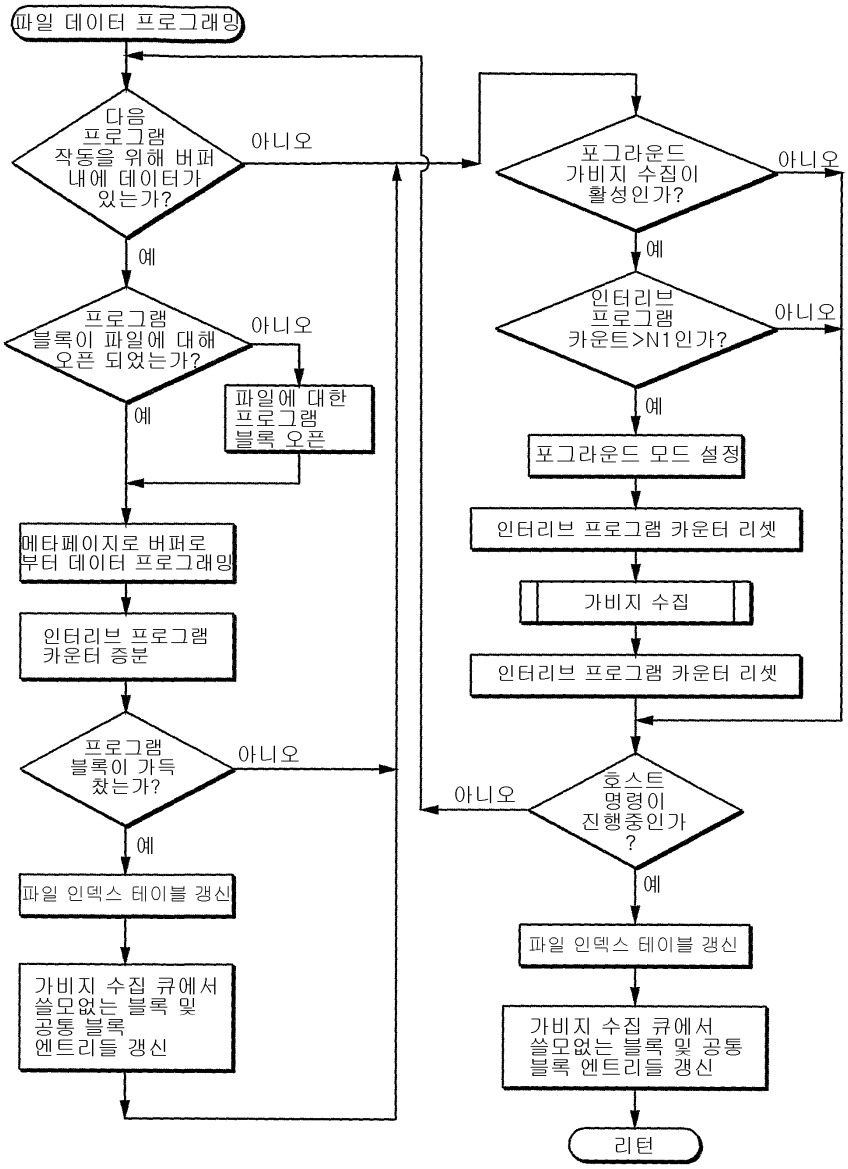
도면4A

장치 작동들에 대한 흐름도



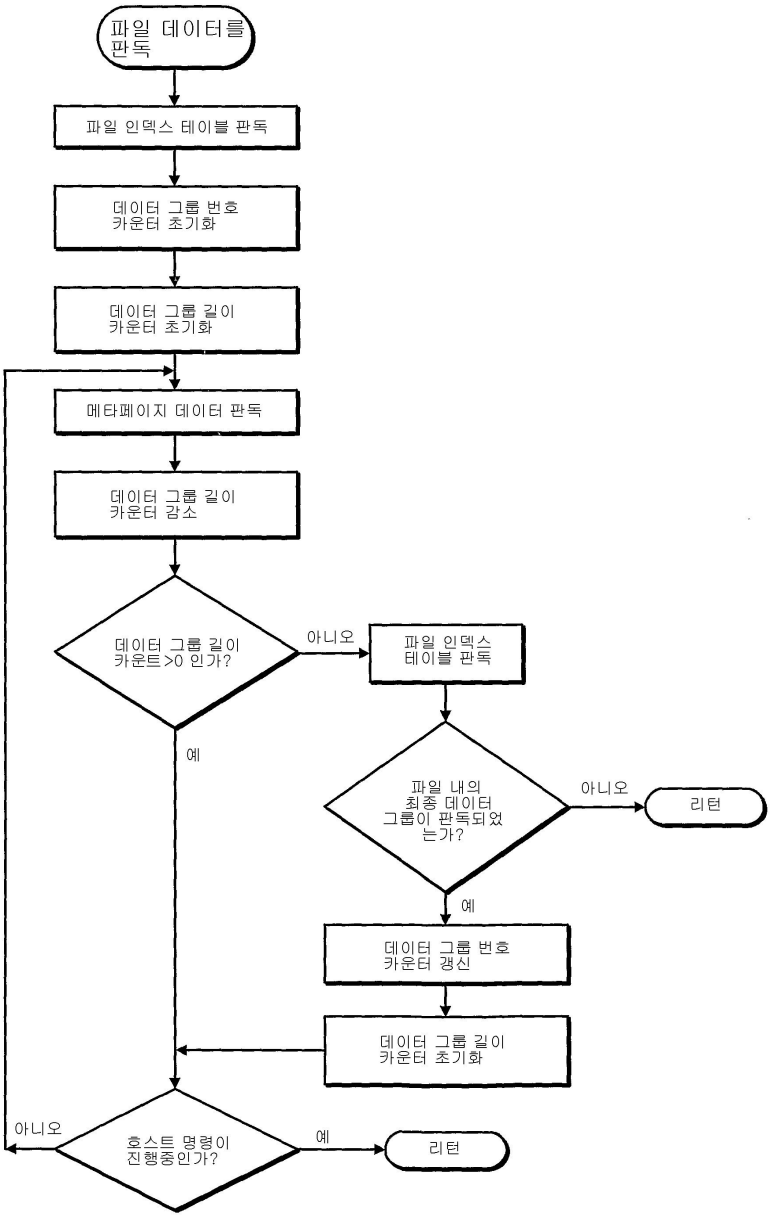
도면5A

파일 데이터를 프로그래밍하는 흐름도



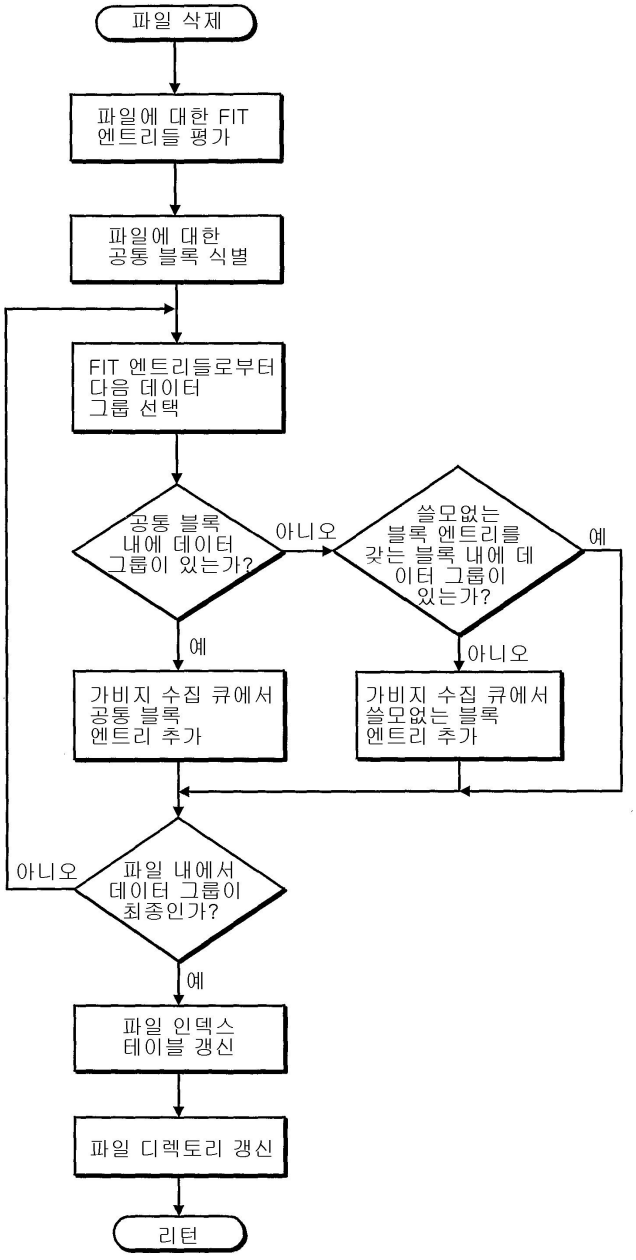
도면6A

파일 데이터를 판독하는 흐름도



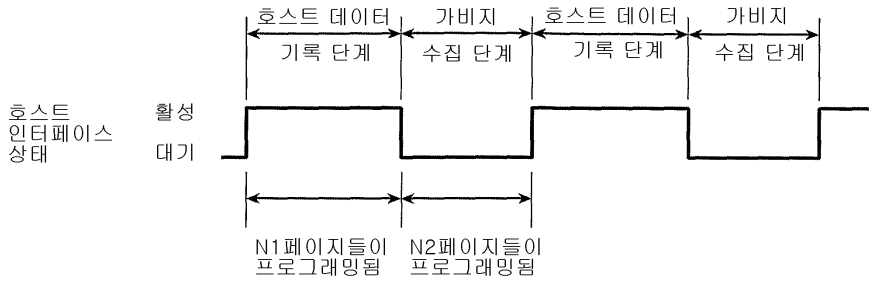
도면7A

파일을 삭제하는 흐름도



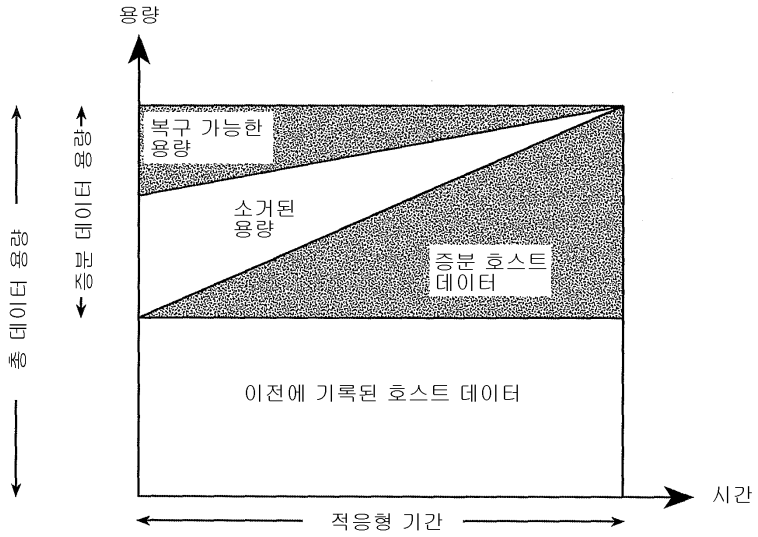
도면8A

포그라운드 가비지 수집에 대한 인터리빙된 작동들

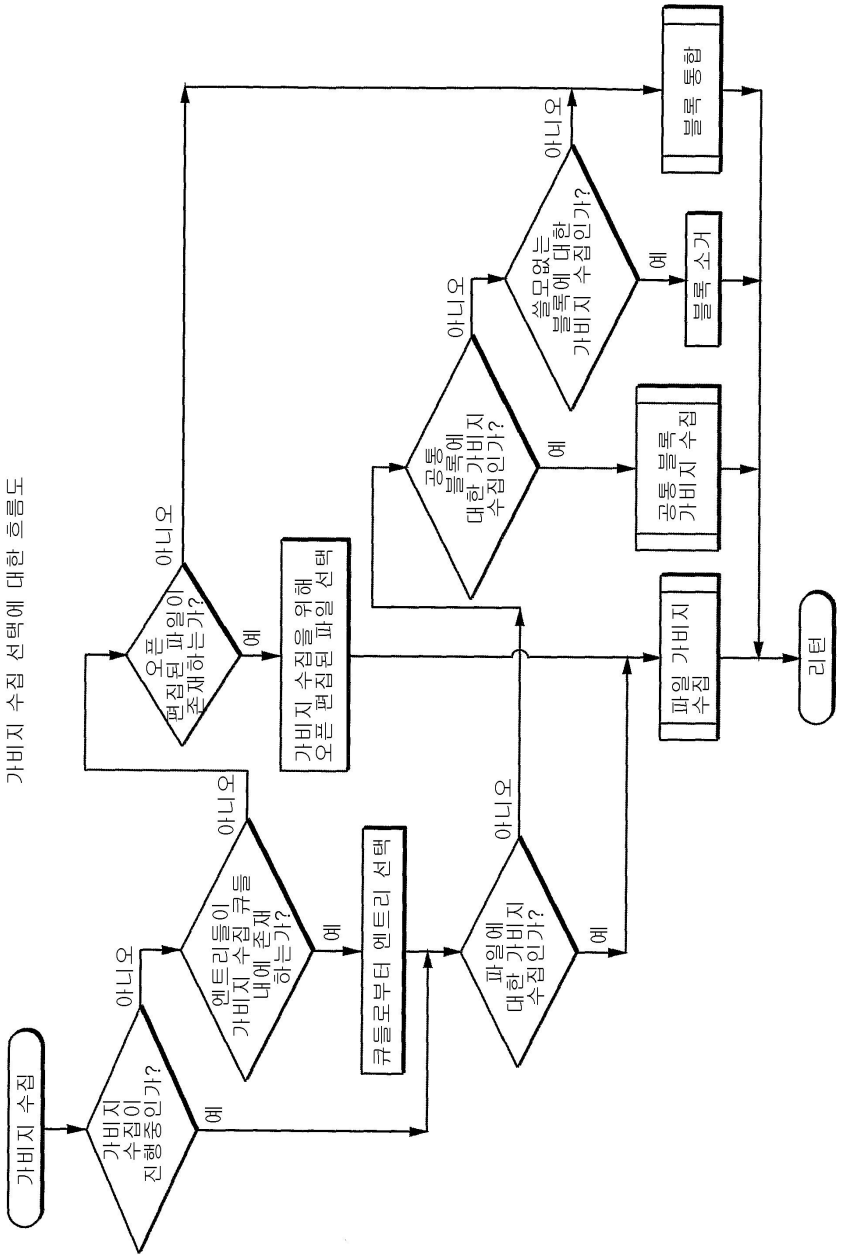


도면8B

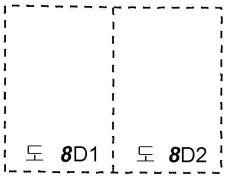
가비지 수집의 적응형 스케줄링에 대한 작동의 원리



도면8C

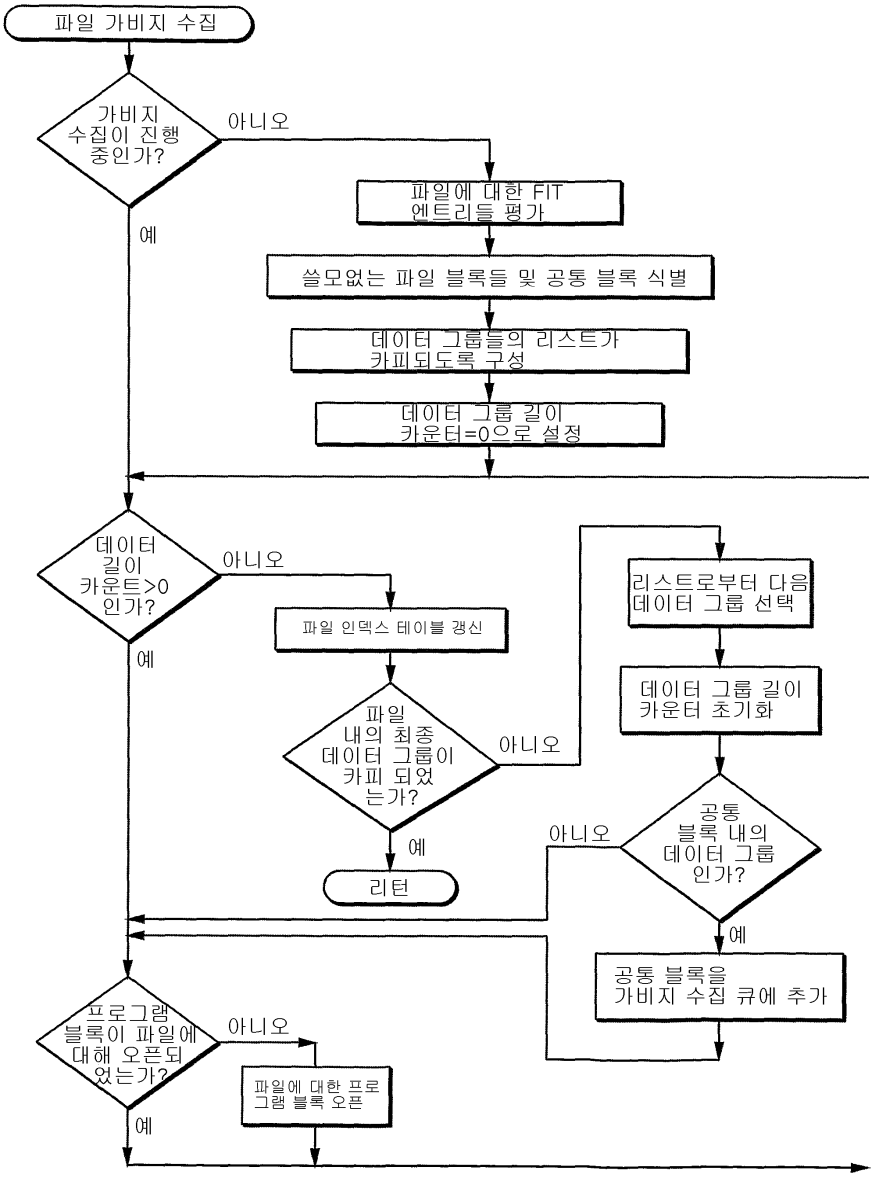

가비지 수집 선택에 대한 흐름도
가비지 수집
가비지 수집이 진행중인가?
아니오
예
엔트리들이 가비지 수집 큐들 내에 존재하는가?
아니오
예
큐들로부터 엔트리 선택
파일에 대한 가비지 수집인가?
아니오
예
오픈 편집된 파일이 존재하는가?
아니오
예
가비지 수집을 위해 오픈 편집된 파일 선택
공통 블록에 대한 수집인가?
아니오
예
공통 블록 가비지 수집
슬모없는 블록에 대한 가비지 수집인가?
아니오
예
블록 소거
블록 통합
파일 가비지 수집
리턴

도면8D

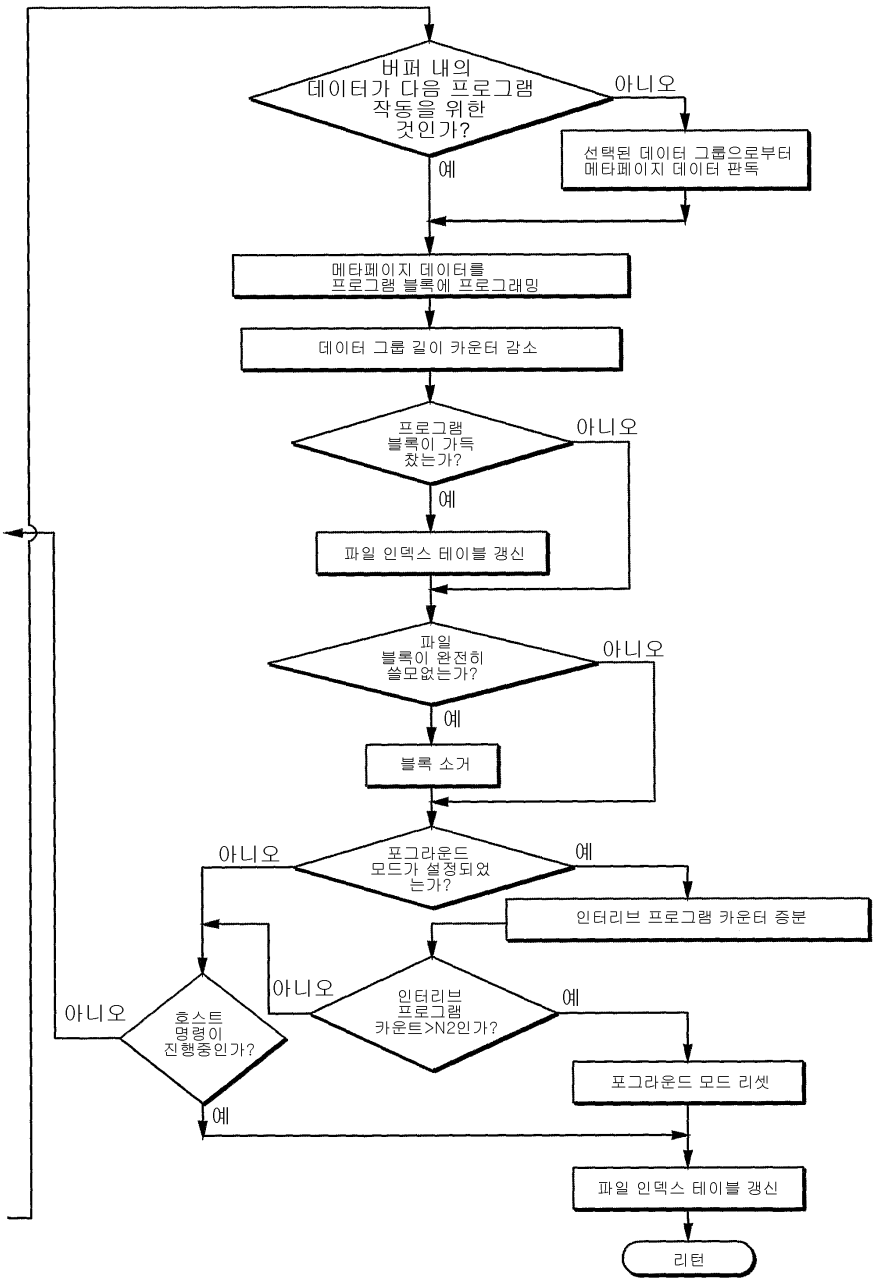

도 8D1
도 8D2

도면8D1

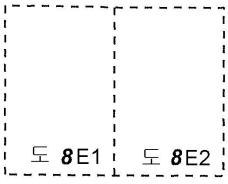
파일 가비지 수집에 대한 흐름도



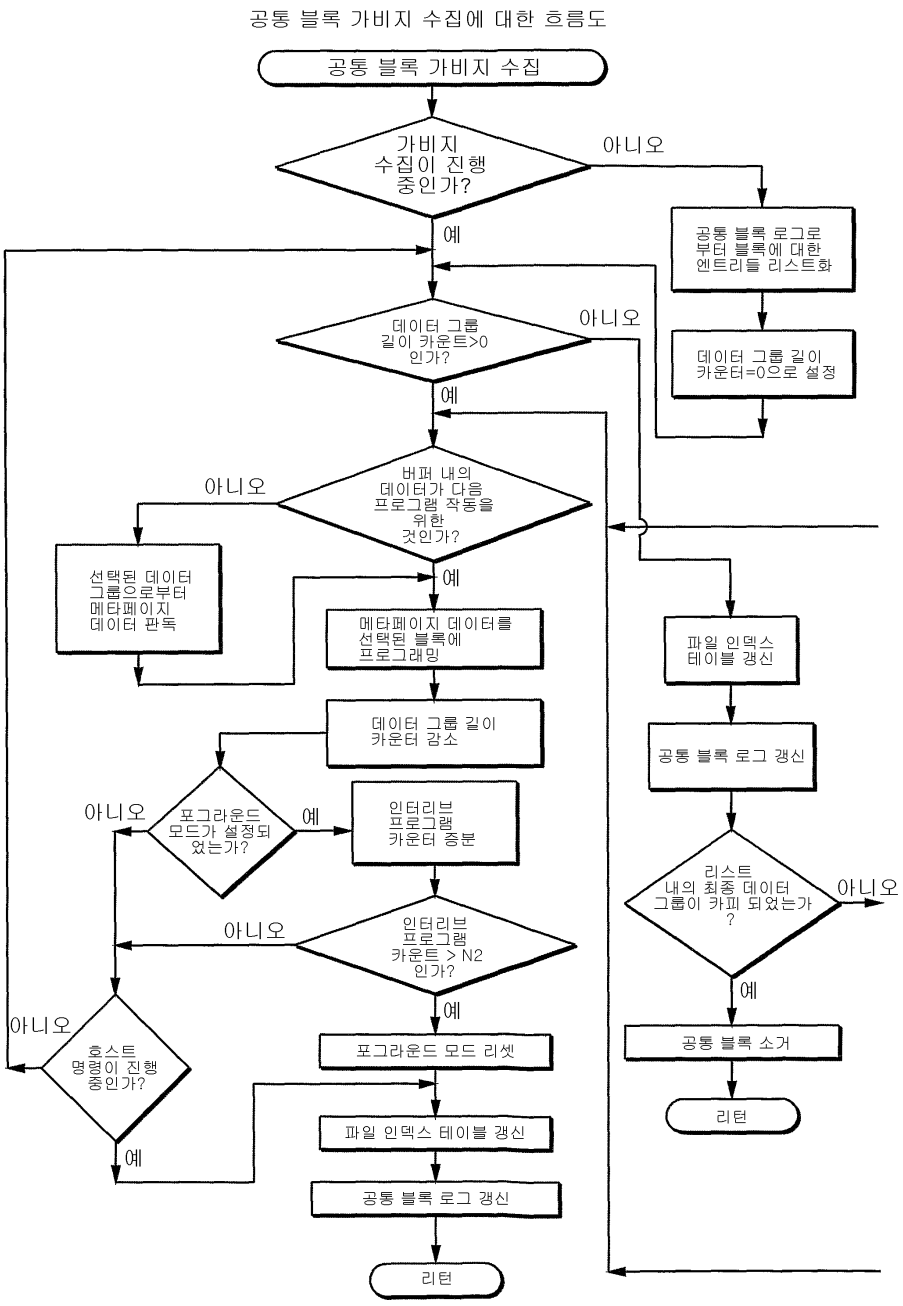
도면8D2


버퍼 내의 데이터가 다음 프로그램 작동을 위한 것인가?
아니오
예
선택된 데이터 그룹으로부터 메타페이지 데이터 판독
메타페이지 데이터를 프로그램 블록에 프로그래밍
데이터 그룹 길이 카운터 감소
프로그램 블록이 가득 찼는가?
아니오
예
파일 인덱스 테이블 갱신
파일 블록이 완전히 쓸모없는가?
아니오
예
블록 소거
포그라운드 모드가 설정되었는가?
아니오
예
인터리브 프로그램 카운터 증분
인터리브 프로그램 카운트>N2인가?
아니오
예
호스트 명령이 진행중인가?
아니오
예
포그라운드 모드 리셋
파일 인덱스 테이블 갱신
리턴

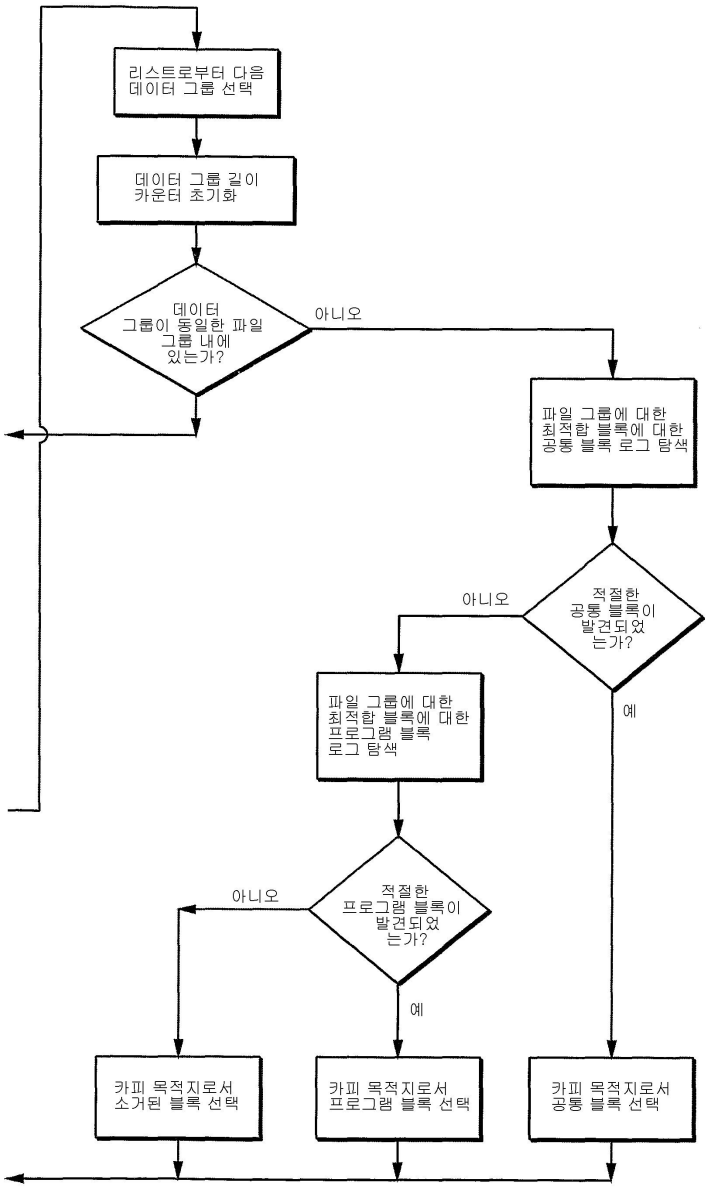
도면8E


도 8E1
도 8E2

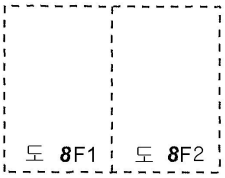
도면8E1


공통 블록 가비지 수집에 대한 흐름도
공통 블록 가비지 수집
가비지 수집이 진행 중인가?
아니오
예
공통 블록 로그로부터 블록에 대한 엔트리들 리스트화
데이터 그룹 길이 카운트>0 인가?
아니오
데이터 그룹 길이 카운터=0으로 설정
예
버퍼 내의 데이터가 다음 프로그램 작동을 위한 것인가?
아니오
선택된 데이터 그룹으로부터 메타페이지 데이터 판독
예
메타페이지 데이터를 선택된 블록에 프로그래밍
파일 인덱스 테이블 갱신
데이터 그룹 길이 카운터 감소
공통 블록 로그 갱신
포그라운드 모드가 설정되었는가?
아니오
예
인터리브 프로그램 카운터 증분
리스트 내의 최종 데이터 그룹이 카피 되었는가 ?
아니오
예
인터리브 프로그램 카운트 > N2 인가?
아니오
예
공통 블록 소거
리턴
호스트 명령이 진행 중인가?
아니오
예
포그라운드 모드 리셋
파일 인덱스 테이블 갱신
공통 블록 로그 갱신
리턴

도면8E2

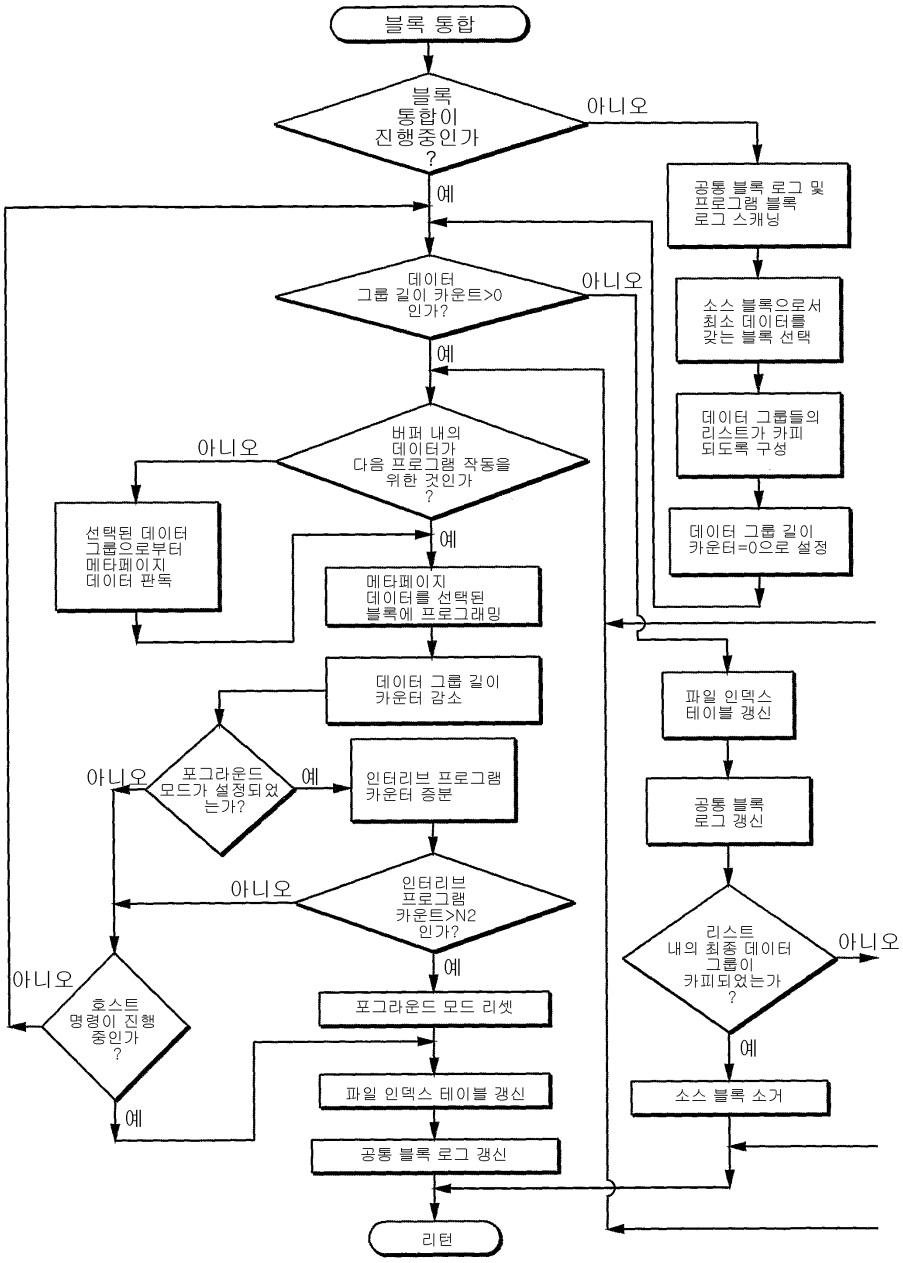

리스트로부터 다음 데이터 그룹 선택
데이터 그룹 길이 카운터 초기화
데이터 그룹이 동일한 파일 그룹 내에 있는가?
아니오
파일 그룹에 대한 최적합 블록에 대한 공통 블록 로그 탐색
적절한 공통 블록이 발견되었는가?
아니오
예
파일 그룹에 대한 최적합 블록에 대한 프로그램 블록 로그 탐색
적절한 프로그램 블록이 발견되었는가?
아니오
예
카피 목적지로서 소거된 블록 선택
카피 목적지로서 프로그램 블록 선택
카피 목적지로서 공통 블록 선택

도면8F

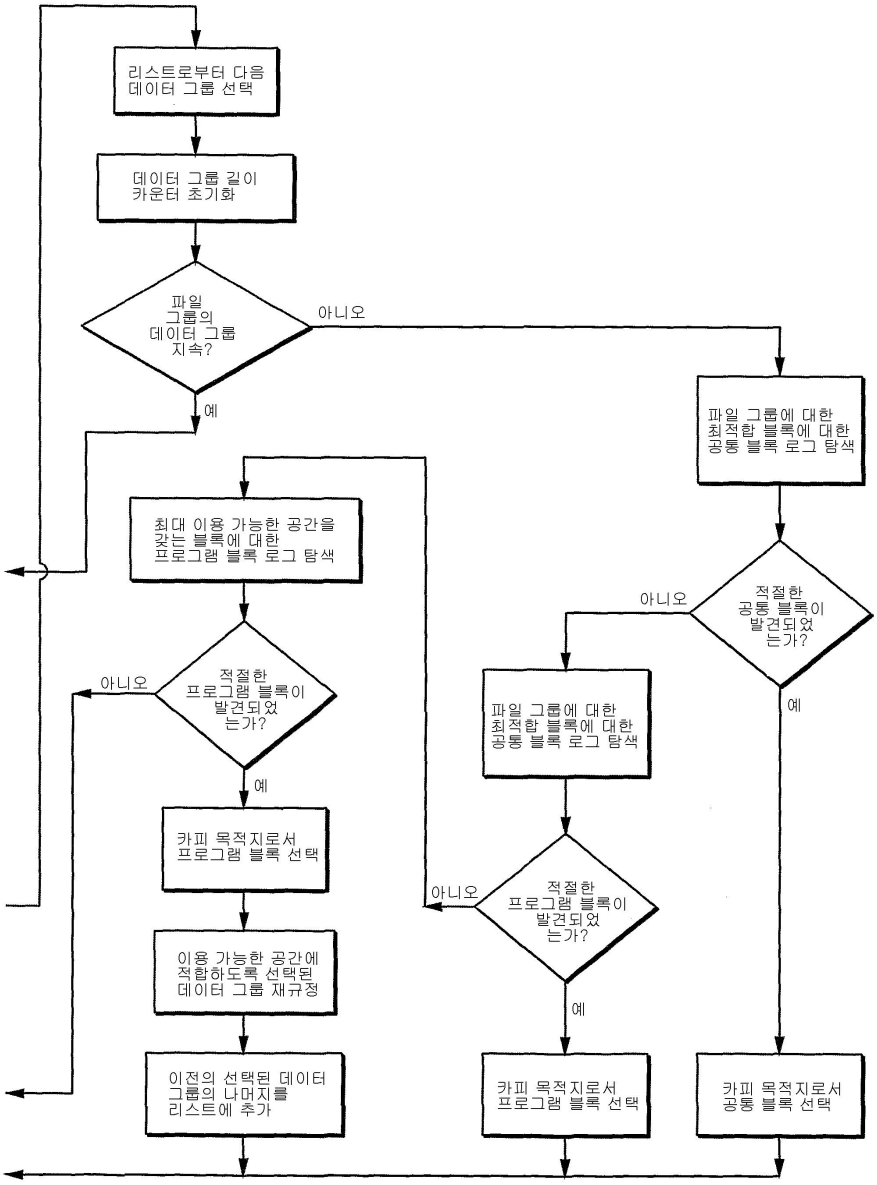

도 8F1
도 8F2

도면8F1

블록 통합에 대한 흐름도

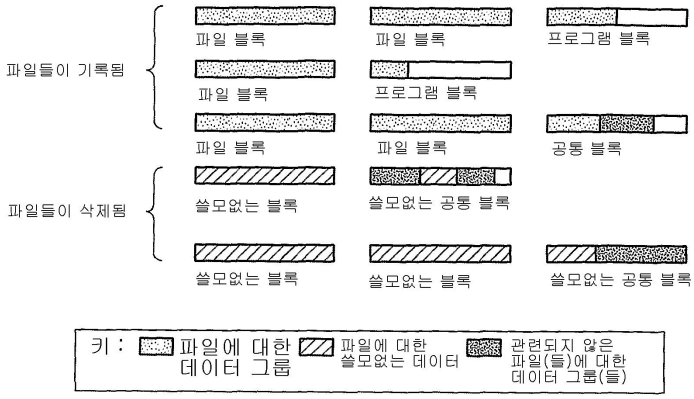


도면8F2


리스트로부터 다음 데이터 그룹 선택
데이터 그룹 길이 카운터 초기화
파일 그룹의 데이터 그룹 지속?
아니오
예
파일 그룹에 대한 최적합 블록에 대한 공통 블록 로그 탐색
최대 이용 가능한 공간을 갖는 블록에 대한 프로그램 블록 로그 탐색
적절한 공통 블록이 발견되었는가?
아니오
예
적절한 프로그램 블록이 발견되었는가?
아니오
예
파일 그룹에 대한 최적합 블록에 대한 공통 블록 로그 탐색
카피 목적지로서 프로그램 블록 선택
적절한 프로그램 블록이 발견되었는가?
아니오
예
이용 가능한 공간에 적합하도록 선택된 데이터 그룹 재규정
이전의 선택된 데이터 그룹의 나머지를 리스트에 추가
카피 목적지로서 프로그램 블록 선택
카피 목적지로서 공통 블록 선택

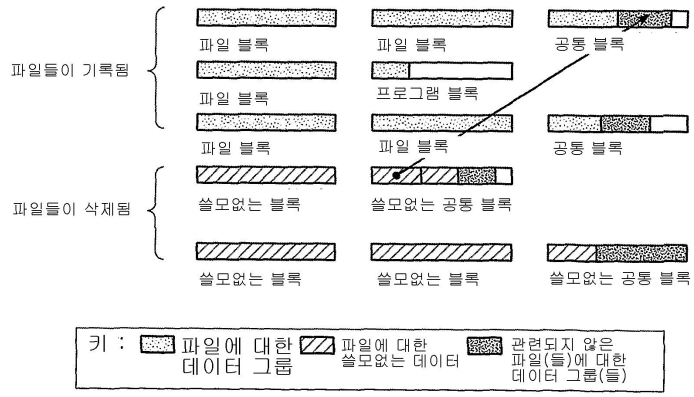
도면8G1

공통 블록 가비지 수집 예(최초 조건)



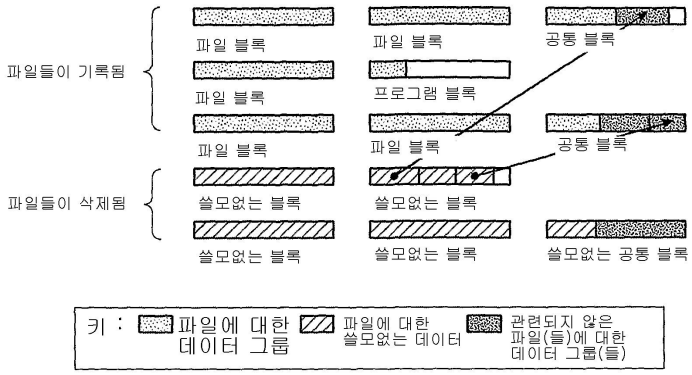
도면8G2

공통 블록 가비지 수집 예(단계 1)



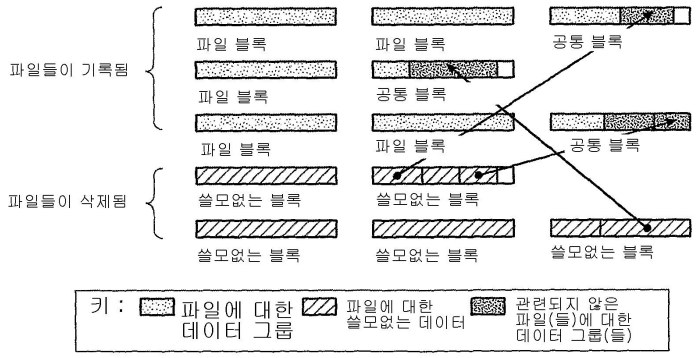
도면8G3

공통 블록 가비지 수집 예(단계 2)

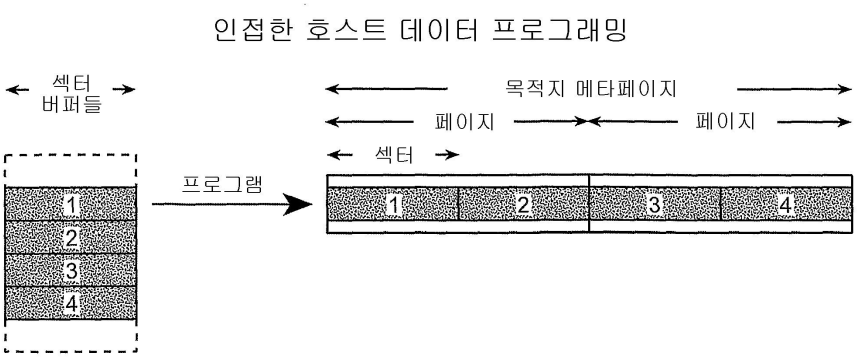


도면8G4

공통 블록 가비지 수집 예(단계 3)

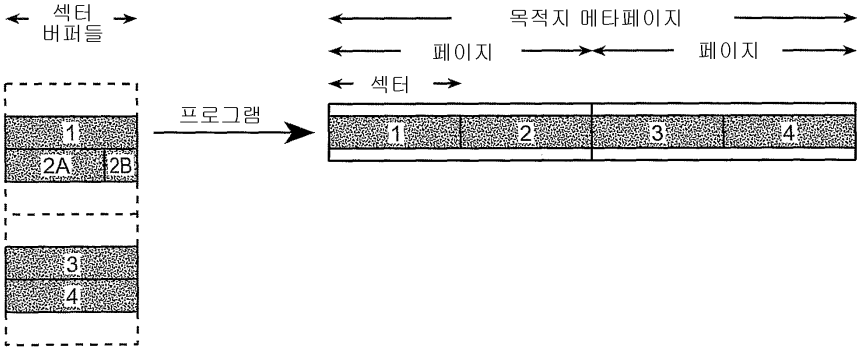


도면9A



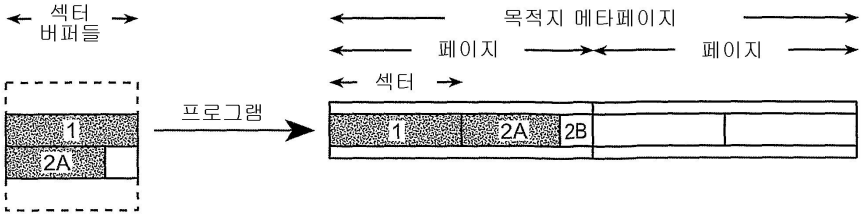
도면9B

인터럽트된 호스트 데이터 프로그래밍



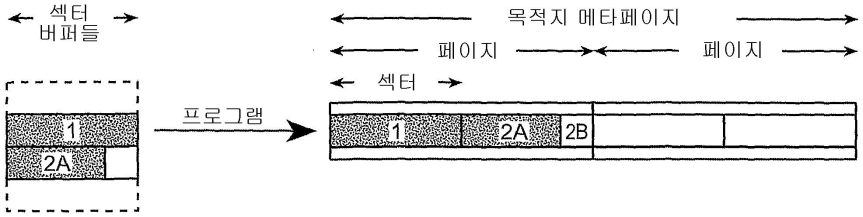
도면9C

버퍼 플러시 프로그래밍



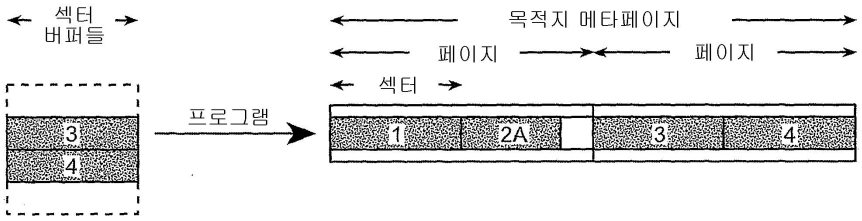
도면9D

버퍼 스왑-아웃 프로그래밍



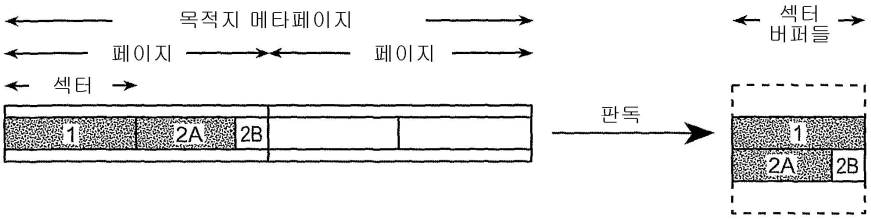
도면9E

버퍼 플러시 이후의 호스트 데이터 프로그래밍



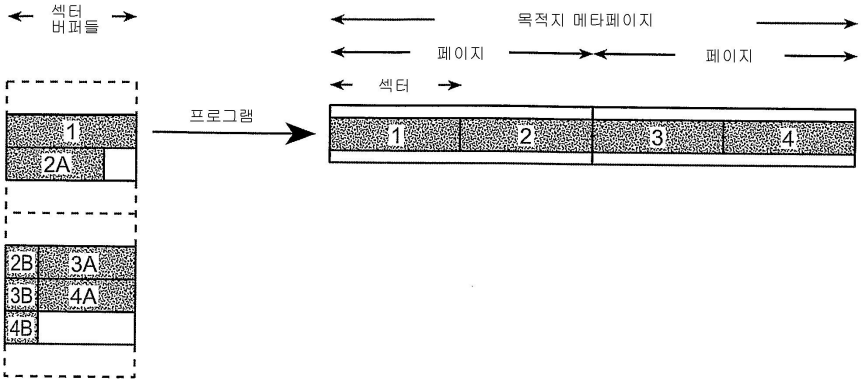
도면9F

스왑-인 데이터 판독



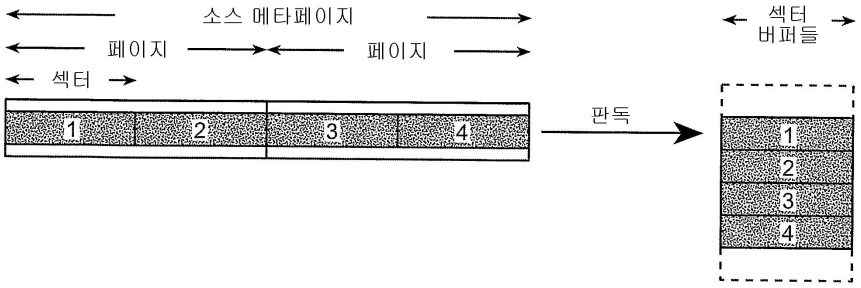
도면9G

버퍼 스왑-인 이후의 호스트 데이터 프로그래밍



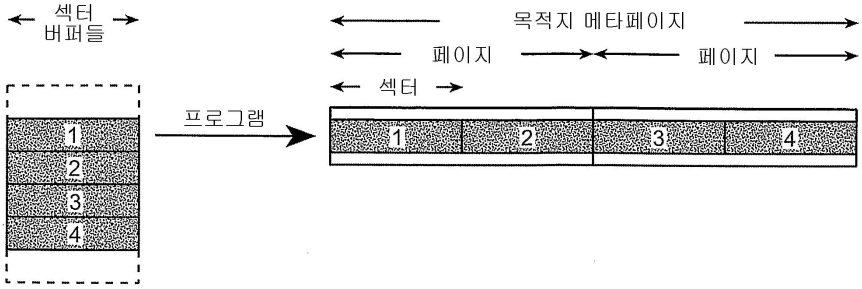
도면9H

버퍼로부터의 정렬된 데이터 판독



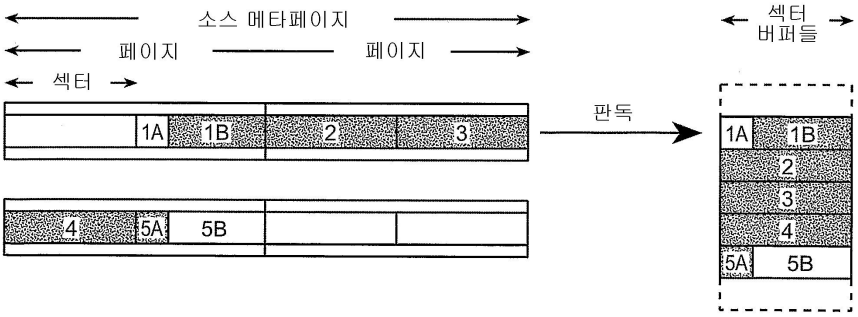
도면9I

버퍼로부터의 정렬된 데이터 프로그래밍



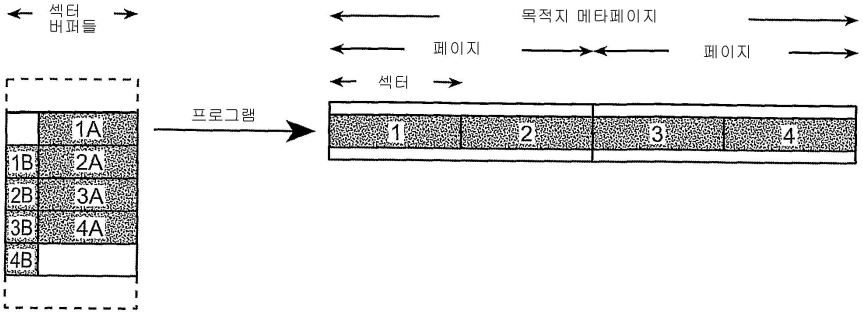
도면9J

버퍼로의 비-정렬된 데이터 판독



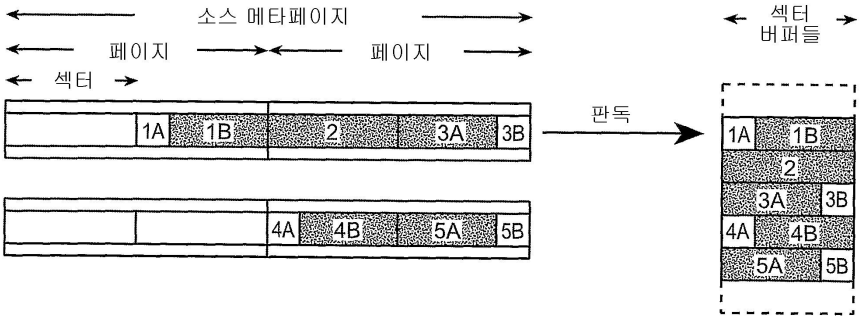
도면9K

버퍼로부터의 비-정렬된 데이터 프로그래밍



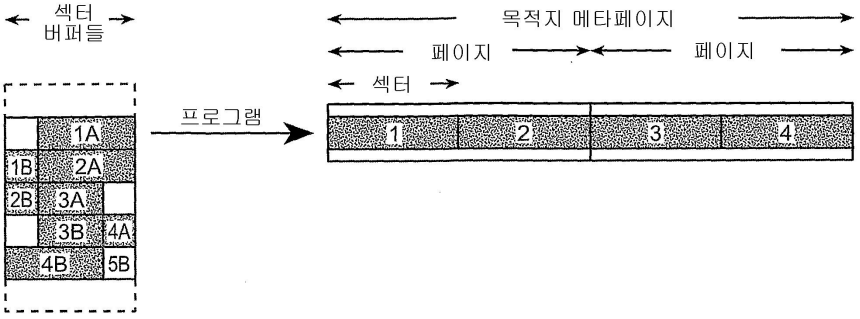
도면9L

버퍼로부터의 비-정렬된 비-순차적인 데이터 판독

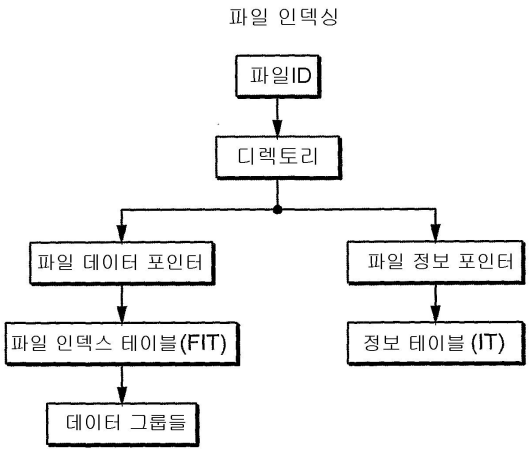


도면9M

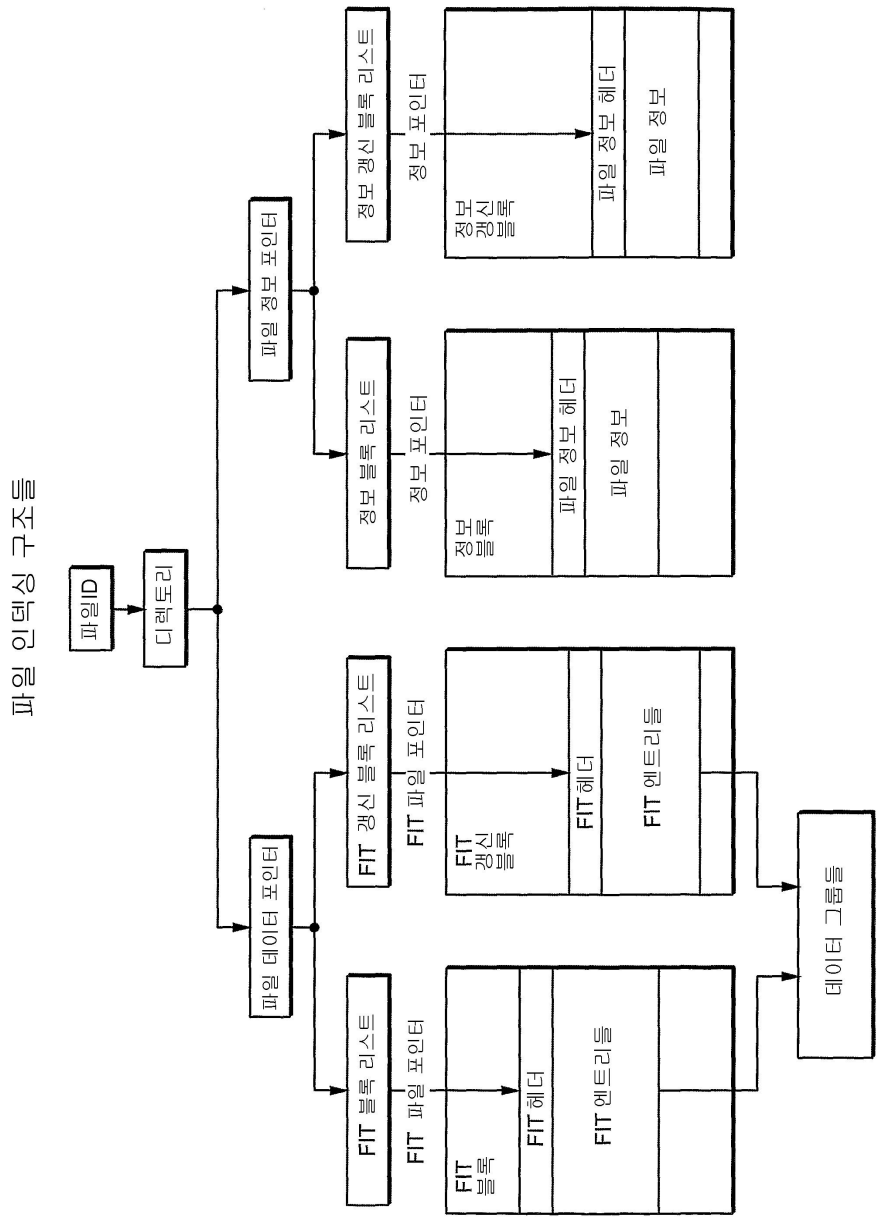
버퍼로부터의 비-정렬된 비-순차적인 데이터 프로그래밍



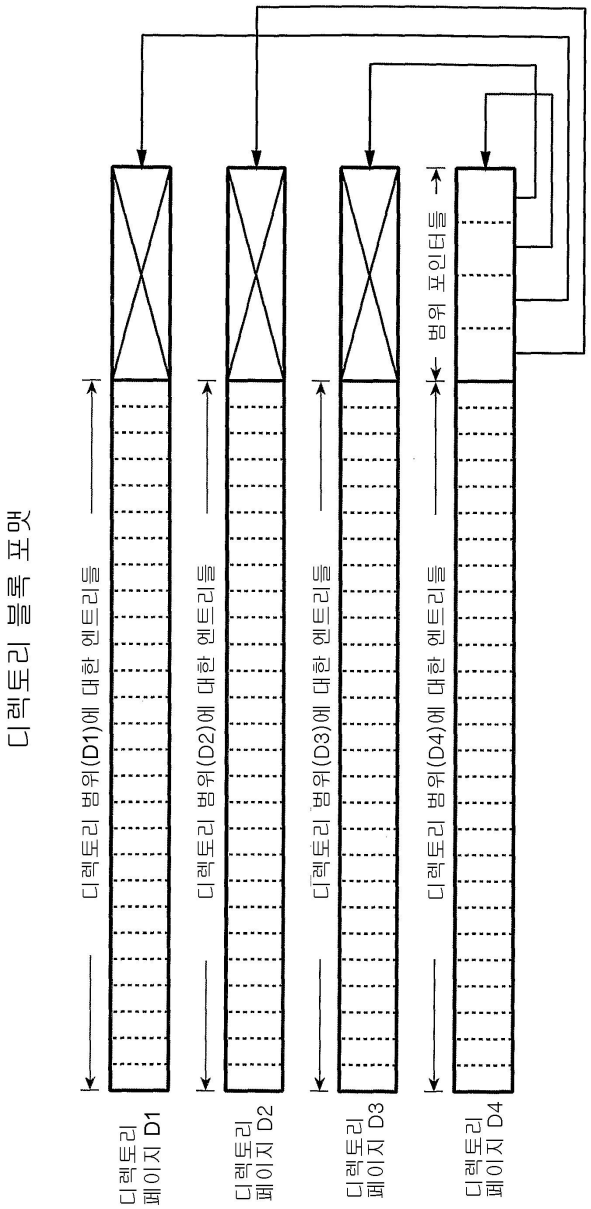
도면10A


파일 인덱싱
파일ID
디렉토리
파일 데이터 포인터
파일 정보 포인터
파일 인덱스 테이블(FIT)
정보 테이블 (IT)
데이터 그룹들

도면10B

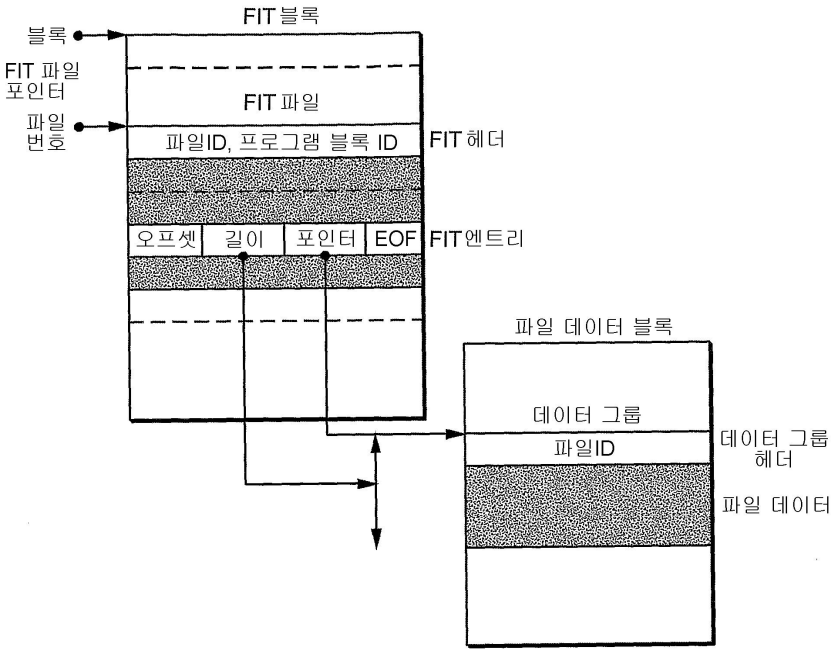

파일 인덱싱 구조들
파일ID
디렉토리
파일 정보 포인터
파일 데이터 포인터
정보 블록 리스트
정보 포인터
정보 블록
파일 정보 헤더
파일 정보
정보 갱신 블록 리스트
정보 포인터
정보 갱신 블록
파일 정보 헤더
파일 정보
FIT 갱신 블록 리스트
FIT 파일 포인터
FIT 갱신 블록
FIT 헤더
FIT 엔트리들
FIT 블록 리스트
FIT 파일 포인터
FIT 블록
FIT 헤더
FIT 엔트리들
데이터 그룹들

도면10C

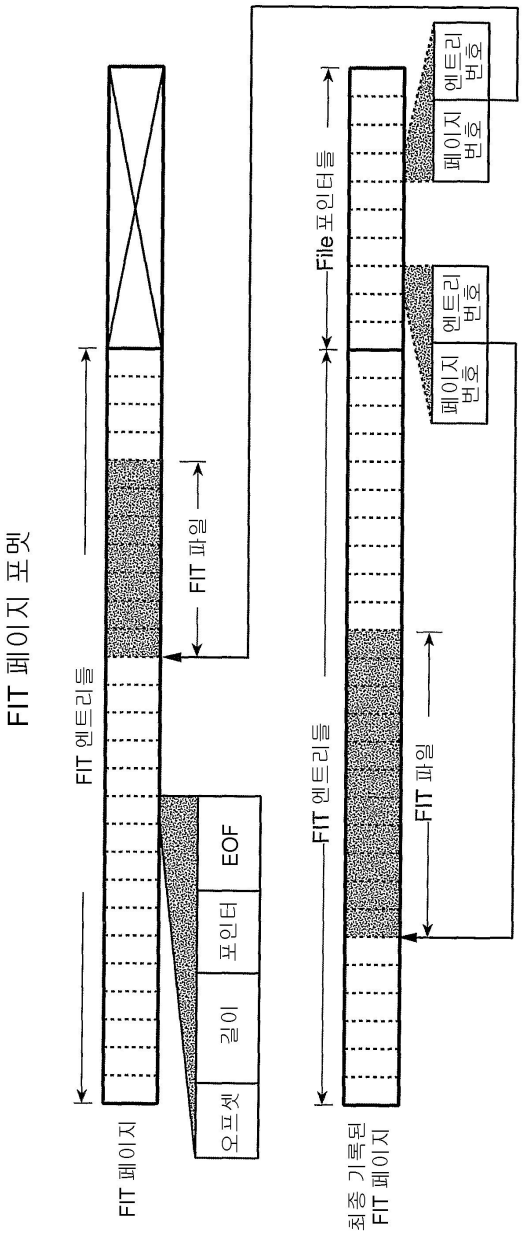

디렉토리 블록 포맷
디렉토리 범위(D1)에 대한 엔트리들
디렉토리 범위(D2)에 대한 엔트리들
디렉토리 범위(D3)에 대한 엔트리들
디렉토리 범위(D4)에 대한 엔트리들
범위 포인터들
디렉토리 페이지 D1
디렉토리 페이지 D2
디렉토리 페이지 D3
디렉토리 페이지 D4

도면10D

파일 인덱스 테이블 논리적 구조

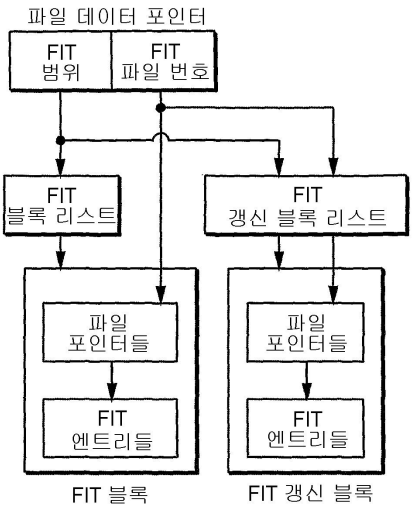


도면10E


FIT 페이지 포멧
FIT 엔트리들
FIT 파일
오프셋
길이
포인터
EOF
FIT 페이지
File 포인터들
FIT 엔트리들
FIT 파일
페이지 번호
엔트리 번호
페이지 번호
엔트리 번호
최종 기록된 FIT 페이지

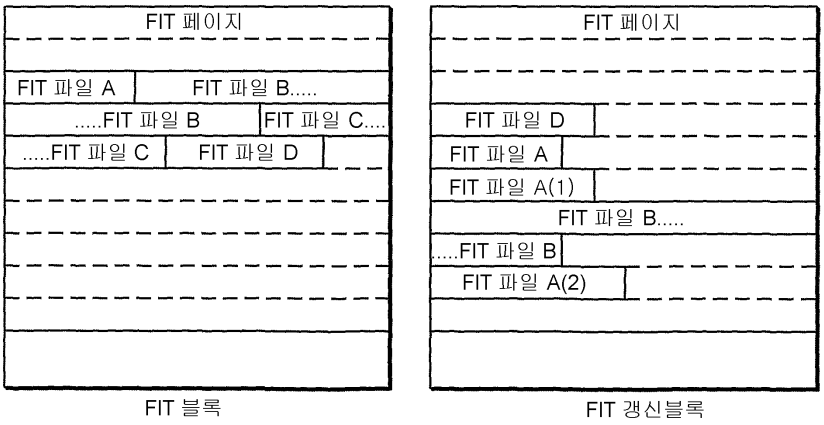
도면10F

물리적 FIT 블록들

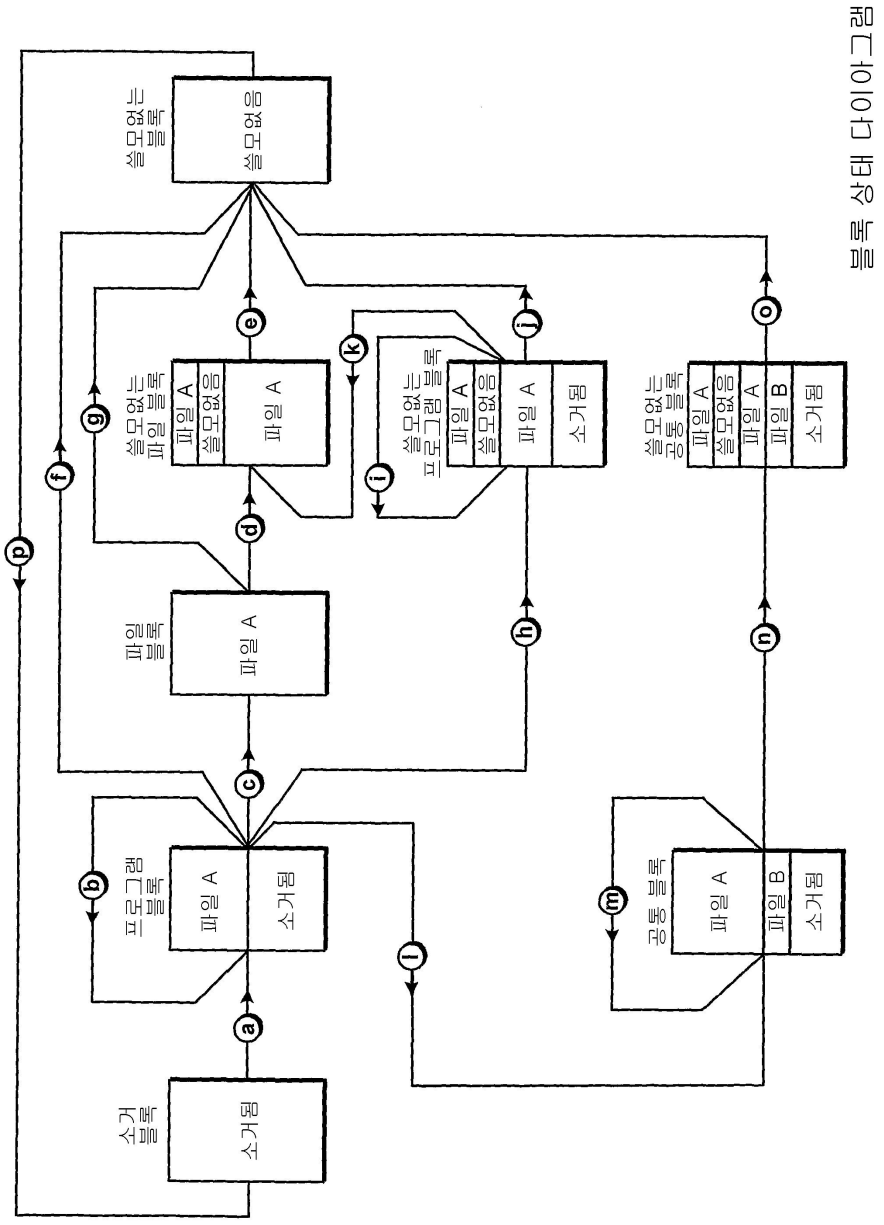


도면10G

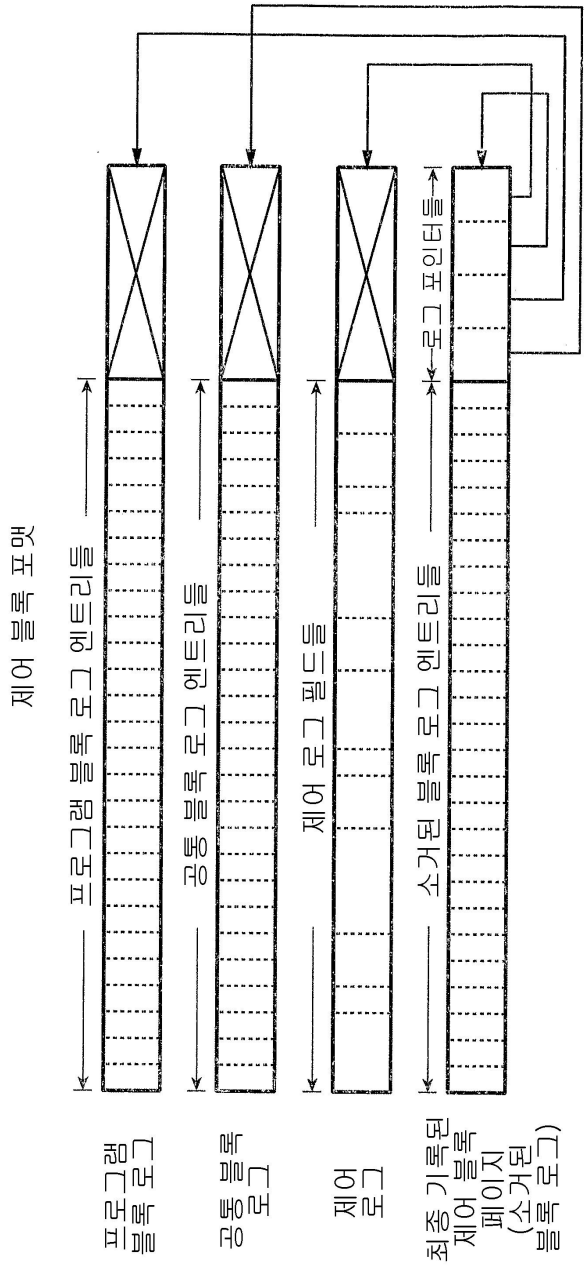
FIT 파일 갱신 작동들의 예들



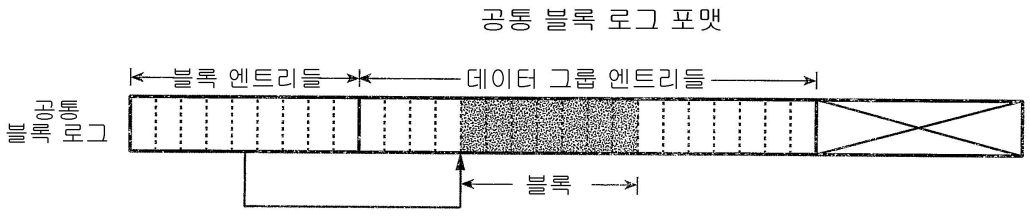
도면11A


블록 상태 다이아그램
슬모없는 블록
슬모없음
슬모없는 파일블록
파일 A
슬모없음
파일 A
슬모없는 프로그램블록
파일 A
슬모없음
파일 A
소거됨
슬모없는 공통블록
파일 A
슬모없음
파일 A
파일 B
소거됨
파일 블록
파일 A
프로그램 블록
파일 A
소거됨
공통 블록
파일 A
파일 B
소거됨
소거 블록
소거됨
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p

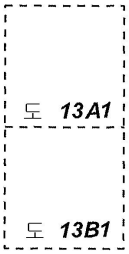
도면12A


제어 블록 포맷
프로그램 블록 로그 엔트리들
공통 블록 로그 엔트리들
제어 로그 필드들
로그 포인터들
소거된 블록 로그 엔트리들
프로그램 블록 로그
공통 블록 로그
제어 로그
최종 기록된 제어 블록 페이지 (소거된 블록 로그)

도면12B


공통 블록 로그 포맷
블록 엔트리들
데이터 그룹 엔트리들
공통 블록 로그
블록

도면13A



도면13A1

정적 파일들과 함께 사용된 명령 세트(부분 A)

명령	파라미터	설명
생성	<파일ID>	장치에서의 디렉토리 내의 <파일ID>에 의해 식별된 엔트리를 생성한다. <파일ID>는 정적 파일에 대한 적절한 수치적인 식별자를 표시하기 위하여 명령으로 지정되어야 한다. 생성 명령은 직접적인 데이터 파일 장치에서의 디렉토리 내에 존재하지 않는 파일에 대한 오픈 명령 다음에 직접적인 데이터 파일 장치가 에러 상태를 리턴시킨 후에 사용될 수 있다.
오픈	<파일ID>	지정된 파일에 대한 후속 데이터 명령들의 실행을 인에블시킨다. 기록_포인터는 파일이 종점으로 설정되고 판독_포인터는 파일의 시작점으로 설정된다. 파일이 존재하지 않는 경우, 에러 상태가 리턴된다.
클로우즈	<파일ID>	지정된 파일에 대한 후속 데이터 명령들의 실행을 디스에이블시킨다. 정적 파일에 대한 클로우즈 명령은 엔트리가 쓸모없는 블록 가비지 수집 큐 또는 파일 가비지 수집 큐 내로 삽입되도록 한다.
삭제	<파일ID>	삭제 명령은 통상적으로 정적 파일들을 위해 할당된 장치 구획의 크기가 감소되는 경우에 사용될 수 있다.
리스트_파일		수치적인 순서로, 장치로부터의 모든 유효 <파일ID>값들을 판독한다.
기록	<파일ID>	기록_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서의 지정된 정적 파일에서 데이터를 겹쳐쓰기한다. 오프셋 어드레스는 모든 정적 파일들에 대해 호스트에 의해 사용된 파일 크기에 대응하는 범위 밖에 있지 않아야 한다.
판독	<파일ID>	판독_포인터의 현재 값에 의해 규정된 오프셋 어드레스에서의 지정된 파일로부터 데이터를 판독한다. 오프셋 어드레스는 모든 정적 파일들에 대해 호스트에 의해 사용된 파일 크기에 대응하는 범위 밖에 있지 않아야 한다.

도면13A2

정적 파일들과 함께 사용된 명령 세트(부분 B)

명령	파라미터들	설명
저장_버퍼	<파일ID>	플래시 메모리 내의 일시적인 위치에 지정된 정적 파일에 대한 버퍼 내의 데이터를 저장. 저장_버퍼는 정적 파일들을 위한 사용중인 프로토콜이 상기 파일에 대해 공급되었던 데이터가 플래시 메모리로 커미팅 되어야 할 때 사용된다.
기록_포인터	<파일ID> <오프셋>	지정된 파일에 대한 기록_포인터에 대한 새로운 현재 값을 규정한다. 기록_포인터는 파일이 오픈된 후에 제1 데이터에 대하여, 또는 이전에 기록된 데이터에 순차적이지 않은 데이터에 대하여, 기록될 데이터의 정적 파일 내의 오프셋을 규정하는데 사용된다. 오프셋은 모든 정적 파일들에 대하여 호스트에 의해 사용된 파일 크기에 대응하는 범위 밖에 있지 않아야 한다.
판독_포인터	<파일ID> <오프셋>	지정된 파일에 대한 판독_포인터에 대한 새로운 현재 값을 규정한다. 판독_포인터는 파일이 오픈된 후에 제1 데이터에 대하여, 또는 이전에 판독된 데이터에 순차적이지 않은 데이터에 대하여, 판독될 데이터의 정적 파일 내의 오프셋을 규정하는데 사용된다. 오프셋은 모든 정적 파일들에 대하여 호스트에 의해 사용된 파일 크기에 대응하는 범위 밖에 있지 않아야 한다.
스트림	<길이>	장치로 또는 장치로부터 전달될 데이터의 인터럽트되지 않은 스트림을 규정한다. 모델링을 위해서만 사용된다.
중단	<시간>	다음 명령의 실행 이전에 삽입되는 지연을 규정한다. 모델링을 위해서만 사용된다.
유휴		장치는 자신이 내부 작동들을 수행할 수 있는 유휴 상태에 진입해야 한다. 유휴 명령은 임의의 진행중인 가비지 수집 작동들이 수행되도록 하기 위하여, 정적 파일에 대한 클로우즈 명령 직후에 사용된다. 장치가 백그라운드 작동들에 대해 작동중이지 않은 상태를 나타낼 때까지 다른 명령을 장치에 전송하지 않아야 한다.
대기		장치는 자신이 내부 작동들을 수행할 수 없는 대기 상태에 진입해야 한다.
셧-다운		장치는 자신이 다음에 작동중인 상태가 아닐 때 셧다운될 것이고, 전력이 제거될 것이다.
용량		장치가 파일 데이터에 의해 차지되고 새로운 파일 데이터에 이용가능한 용량들을 보고하도록 하는 호스트로부터의 요청.
요청		장치가 자신의 현재 상태를 보고하도록 하는 호스트로부터의 요청. 상태 명령은 실행중인 명령을 종료시키지 않는다.