USB 저장장치의 효율적인 통합을 위한 시그마 허브

(Sigma Hub for Efficiently Integrating USB Storages)

최 오 훈 † 임 정 은 † 나 홍 석 †† 백 두 권 †††
(O-Hoon Choi) (Jung-Eun Lim) (Hong-Seok Na) (Doo-Kwon Baik)

요 약 USB 저장장치는 반도체 메모리 저장 용량에 대한 기술 발전에 따라, 대용량 저장공간을 제공하는 제품으로 생산되고 있다. 따라서 소비자는 저장공간이 상대적으로 작은 기존 USB 저장장치를 폐기하거나, 효율적으로 사용하지 않는다. 본 논문에서는 메모리 저장 용량이 다른, 다수의 USB 저장장치를 USB 허브를 통하여 그룹화 하여 사용자가 컴퓨터에서 논리적 단일 저장공간으로 사용할 수 있는 개선된 USB 허브인 시그마 허브(Sigma Hub)를 제안한다. 제안된 시그마 허브는 다수의 USB 저장장치를 단일화하기 위한 관리 모듈로서 시그마 콘트롤러(Sigma Controller)를 포함한다. 시그마 콘트롤러는 USB 저장장치 통합 알고리즘을 통하여 트랜잭션을 제어하여 논리적 단일 저장장치 사용을 가능하게 한다.

키워드 : 메모리 관리, USB 저장장치, 가상 저장장치, 프로토콜, USB 허브

Abstract With technological advances for storage volume size of a semiconductor memory, USB storage is made as products to support a high capacity storage. Hereby, consumers discard pint-sized USB storages which they already had, or do not use them efficiently. To integrate and unify these pint-sized USB storages as one big USB storage, we proposed Sigma Hub. It can be grouping multiple USB storages, which have each different volume size of memory storage, as logical unity Storage through USB Hub. The proposed Sigma Hub includes Sigma Controller as a core management module to unify the multiple USB storages in transaction level layer. Sigma controller can efficiently control transaction packet in Sigma Hub through a USB Storage-Integration algorithms which ensure an integrity for data read and write processes. Consequently, Sigma Hub enables the use of USB storage that is logical unity.

Key words : Memory Management, USB Storage, Virtual Driver, Protocol, USB Hub

1. 서 론

USB(Universal Serial Bus: 범용직렬버스, 이하 "USB"라 함)는 컴퓨터 외부에 장착된 포트로서, 주변기기와 컴퓨터 본체를 연결시켜주는 장치이다. 이는 최대 127개까지 주변 기기를 연결할 수 있으며, 컴퓨터를 사용하는 도중에 연결하여도 곧바로 인식이 가능해 간편하게 사용할 수 있다. 따라서 키보드, 마우스 및 전자제품들과의 데이타 전송을 위한 입출력 장치로 사용되고 있다[1,2]. 이러한 주변 USB 기기 중 소용량 데이타를 저장하기 위하여 사용되는 것은 USB 저장장치이다[3,4]. USB 저장장치는 작은 크기에 무게도 가벼우며, 용량도 수백 메가바이트(MB) 급~수 기가바이트(Giga) 급으로 기존의 저장 장치(플로피, 시디)보다 저장 용량이 크고 인식 및 휴대가 간편하다. 현재 상용화되어 있는 USB 메모리 용량은 16메가바이트에서 6기가바이트까지 다양하여

· 이 연구에 참여한 연구자의 일부는 '2단계 BK21사업'의 지원비를 받았음

† 학생회원 : 고려대학교 컴퓨터학과
ohchoi@gmail.com
limjungeun@gmail.com
†† 정 회 원 : 한국디지털대학교 디지털정보학과 교수
hsna99@kdu.edu
††† 종신회원 : 고려대학교 컴퓨터학과 교수
baikdk@gmail.com
(Corresponding author임)

논문접수 : 2008년 2월 18일
심사완료 : 2008년 9월 22일

정보과학회논문지: 데이타베이스 제35권 제6호(2008.12)

그 저장용량은 반도체 기술의 발달로 인하여 계속 늘어나고 있다. 사용자는 용량이 큰 저장장치를 사용함에 따라 기존에 사용하던 비교적 저장용량이 작은 저장장치를 폐기하거나 사용하지 않는 경우가 많다. 또한, 기존의 저용량 저장장치는 수 기가바이트 급 콘텐츠 데이타를 저장할 수 없으므로, 대용량 저장장치에 대한 요구가 커지고 있다. 독립된 장치로써만 사용할 수 있는 현재의 USB 저장장치는 다음과 같은 한계점이 있으며, 이를 위하여 논리적으로 단일화된 대용량 저장장치를 제공할 필요가 있다.

- 데이타 전송 후 저장 취소: USB 저장장치의 저장공간보다 큰 단일 파일 데이타에 대한 저장 요구 시, 저장공간 비교를 위한 전처리 과정을 거치지 않는다. 따라서 데이타 전송 트랜잭션을 완료한 후 (전송 시간 소모), 저장공간 부족에 의한 전송실패(NAK) 패킷을 수신하여 저장을 취소하게 된다.
- 자동 파일 분류 기능 부족: 복수개의 USB 저장장치는 컴퓨터에 각각 연결되었을 때, 모두 다른 저장공간으로 인식된다. 따라서 다수의 파일을 포함하는 대용량 데이타의 저장 시, 각 USB 저장장치의 저장용량을 사용자가 파악하여 그 용량에 따라 사용자 스스로 파일을 구분하여 데이타 저장을 실행해야 한다.

본 논문은 기존에 사용하던 저 용량 USB 저장장치들을 효율적으로 사용할 수 있는 개선된 USB 허브를 제안하며 그 이름을 시그마 허브(Sigma Hub)라 명명한다. 제안된 시그마 허브는 최대 127개의 USB 저장장치의 저장용량을 모두 합쳐, 단일한 대용량 저장장치로 인식할 수 있도록 한다[5,6]. 시그마 허브는 시그마 콘트롤러(Sigma Controller)라 명명된 모듈을 통하여 USB 저장장치에서 사용되는 트랜잭션을 본 논문에서 제공하는 USB 저장장치 통합 알고리즘을 통하여, 데이타 입출력을 단일 저장장치에서 이루어지듯이 제어한다[7,8]. 또한 시그마 콘트롤러는 시그마 허브 내부의 주소 로그 메모리(Address Log Memory)를 통하여 USB 저장장치 제어시 필요한 주소 값 및 저장파일에 대한 정보를 파일 로그(File Log)로서 저장한다.

시그마 허브에 대한 개념도는 그림 1과 같다. 기존 USB 허브는 호스트 플랫폼(Host Platform)의 USB 포트의 수를 늘리기 위한 다수의 포트 장치 이외의 기능은 없다. 그러나 시그마 허브는 트랜잭션 제어를 위한 USB 저장장치 통합 알고리즘이 포함된 시그마 콘트롤러를 내부 모듈로 갖고 있기 때문에, 다수 포트에 연결된 USB 저장장치는 호스트 플랫폼에서 하나의 대용량 USB 저장장치로 인식된다.

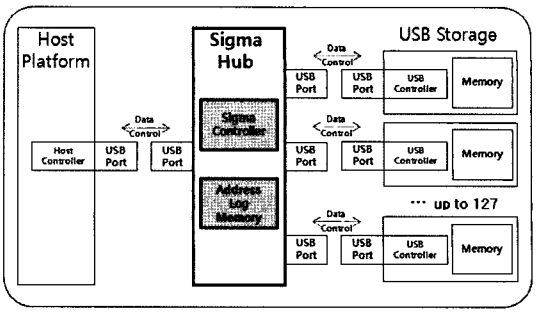


그림 1 제안하는 시그마 허브 개요도

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 언급하고 있는 USB 저장장치의 구조 및 트랜잭션을 알아본다. 3장에서는 본 논문에서 제안하는 시그마 허브의 구조 및 핵심 컴포넌트에 대하여 설명하고, 해당 컴포넌트가 사용하는 USB 저장장치 통합 알고리즘을 설명한다. 4장에서는 구현 및 실험을 통하여 제안된 시그마 허브의 유효성을 살펴본다. 끝으로 5장에서 본 논문에 대한 결론을 내린다.

2. 관련 연구

2.1 USB 저장장치 구조

USB는 호스트 컨트롤러를 통하여 디바이스 장치와 연결된다. 호스트 컨트롤러는 루트 허브라고도 불리며, 대부분 컴퓨터 USB 포트에 내장되어 있다. 디바이스에는 호스트 컨트롤러와 통신하기 위한 USB 포트와 USB 컨트롤러가 있으며, USB 저장장치인 경우 플래시 메모리가 내장되어 있다. USB 저장장치가 컴퓨터의 USB 포트와 연결되면 호스트 컨트롤러는 디바이스에 대한 정보를 요청하며 이를 디스크립트 정보를 통하여 응답한다. 디스크립트 정보는 장치의 제조사, 저장용량 및 저장 파일에 대한 메타데이타가 저장되어 있다.

호스트 컨트롤러는 디스크립트 정보를 획득한 후 각 USB 저장장치를 구분하기 위하여 인식된 순서대로 고유 주소를 지정한다. USB 트랜잭션에서 주소는 7비트(bit)를 사용하므로 총 128개의 주소를 사용할 수 있으나, 0번지는 초기 인식 주소로 사용되기 때문에 USB에 할당되는 주소는 127개 이다. 따라서 호스트 콘트롤러에서 관리하는 USB 저장장치는 최대 127개까지 확장 가능하다[1,2].

그림 2는 호스트 콘트롤러가 버스(BUS)를 통해 127개의 주소로 구분된 USB 저장장치를 찾아 데이타를 메모리에 읽기·쓰기를 할 수 있는 구조도를 보여주고 있다. USB는 호스트 중심의 버스이기 때문에 USB 저장장치 내부에서 발생하는 저장에 관한 제어는 호스트 컨트롤러에서 담당할 수 없다. 따라서 엔드 포인트(End

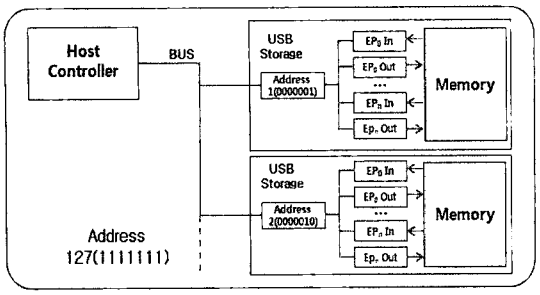


그림 2 호스트 콘트롤러와 USB 저장장치의 연결

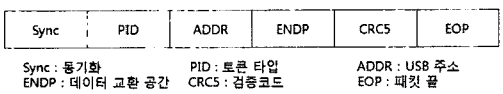
Point)라고 하는 저장공간에 버스에서 전송된 데이타를 임시 보관한 후 USB 저장장치의 제어를 통해 읽기·쓰기가 이루어진다.

2.2 USB 저장장치 트랜잭션

USB 저장장치는 데이타를 읽기·쓰기 위한 패킷 트랜잭션이 존재한다. 호스트 컨트롤러와 USB 저장장치 사이의 데이타 전송 및 상태 파악을 위하여 다음과 같은 패킷을 사용한다.

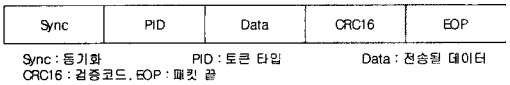
• 토큰 패킷(Token Packet)

호스트 컨트롤로가 초기화 된 후 전송되는 패킷이다. 토큰 패킷 이후에 어떤 패킷이 전송될지 알려준다. 셋업(Setup) 토큰 패킷인 경우, 제어전송을 시작하기 위하여 사용된다. 데이타 읽기 전송인 경우에 인(In) 패킷이 사용되며 데이타 쓰기 전송인 경우에 아웃(Out) 토큰이 사용된다. 인 토큰 패킷이 전송되는 경우, 수신측은 호스트 콘트롤러가 읽기(Read) 작업을 수행할 USB 저장장치 주소를 전송한다. 아웃 토큰 패킷일 경우, 호스트 콘트롤러가 전송(Send) 작업을 원하는 USB 저장장치 주소를 알려준다. 또한, 호스트 컨트롤러가 제어하는 버스와 저장장치 내부 메모리간의 데이타 교환을 위한 공간인 엔드 포인트가 어디인지를 알려준다. 토큰 패킷의 형식은 다음과 같다.



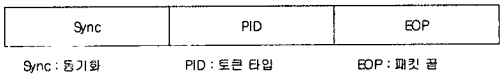
• 데이타 패킷(Data Packet)

실제 데이타 전송에 사용되는 패킷이다. 최대 데이타 전송 크기는 낮은 전송률에서는 8byte, 최대 전송률에서는 1024byte이다. 데이타 패킷은 다음 형식을 갖는다.



• 상태 패킷(Status Packet)

전송이 성공하였는지, 실패하였는지를 알려주기 위하여 사용된다. 아크(ACK) 패킷은 패킷이 성공적으로 전송되었음을 알려준다. 낙(NAK) 패킷은 데이타가 준비되지 않아, 장치가 데이타를 전송하거나 받을 수 없음을 알려준다. 스톨(STALL) 패킷은 장치가 호스트의 중재를 요청하는 상태로 엔드 포인트에서 내부적으로 오류가 발생하여 올바른 동작을 하지 못할 때 사용된다. 상태 패킷은 다음 형식을 갖는다.



본 논문은 토큰 패킷과 상태 패킷을 사용하여 최대 127개에 해당하는 USB 저장장치를 통합하여 단일 저장장치로 사용하기 위한 USB 저장장치 통합 알고리즘을 제안한다.

3. 시그마 허브 아키텍처

3.1 전체 아키텍처와 구성 컴포넌트

시그마 허브는 기본적으로 다수의 포트를 제공하여 USB 허브와 같이 다수의 USB 저장장치들을 하나의 호스트 콘트롤러와 연결한다. 따라서 다수의 USB 저장장치가 독립적으로 인식되지 않고 단일한 저장장치로 운영하기 위한 시그마 콘트롤러가 있다. 시그마 콘트롤러는 호스트 콘트롤러에서 전송된 패킷을 시그마 콘트롤러 내부 모듈을 통하여 적정한 USB 저장장치에 전송한다. 또한 USB 저장장치에서 전송된 상태 패킷을 호스트 콘트롤러에게 전송하지 않고 시그마 콘트롤러 내부 모듈을 통하여 처리한 후, 해당 USB 저장장치로 전송하거나, 호스트 콘트롤러에게 전송한다. 이를 위하여 사용되는 모듈은 디스크립트 통합기 (Descript Integrator: DI), 접근 제어 관리기(Access Control Manager: ACM),

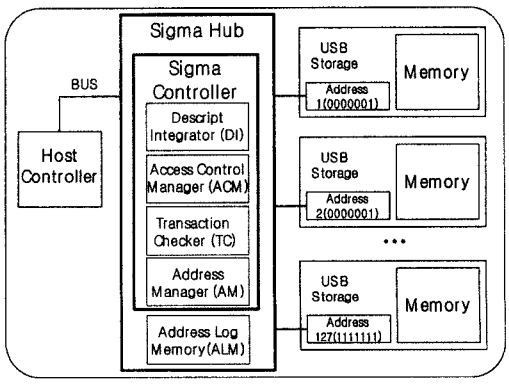


그림 3 시그마 허브의 핵심 모듈의 구조도

트랜잭션 검사기(Transaction Checker: TC), 주소 관리기(Address Manager: AM)로 구성된다. 또한 시그마 콘트롤러가 필요로 하는 데이타를 임시 저장하기 위한 플래시 메모리인 주소 로그 메모리(Address Log Memory: ALM)가 있다. 각 모듈에 대한 설명은 다음과 같다.

• 디스크립트 통합기(Descript Integrator: DI)

디스크립트 통합기의 기능은 통합 디스크립트 정보 작성, 전송, 저장 및 각 디스크립트 정보 비교이다. 가장 중요한 기능은 시그마 허브에 접속된 USB 저장장치들이 각기 다른 저장장치로 인식되지 않고 단일한 저장 장치로 인식 될 수 있게 하는 것이다. 이를 위하여 호스트 콘트롤러로 직접 전송되는 USB 저장장치 디스크립트 정보를 접근 제어 관리기를 통해 차단한다. 차단된 디스크립트 정보를 바탕으로 통합 디스크립트 정보를 작성하여 호스트 콘트롤러에게 전송하여 단일 장치로 인식시킨다. 또한, USB 저장장치 제거시 통합 디스크립트 정보를 재작성하여 호스트 콘트롤러로 전송한다. 통합 디스크립트 정보 및 각 USB 저장장치의 디스크립트 정보들은 주소 로그 메모리에 저장된다. 시그마 허브에 저장된 디스크립트 정보와 USB 저장장치에 저장된 디스크립트 정보는 버전 비교를 통하여 USB 저장장치 탈착시 기접근 유무를 확인한다. 그림 4는 디스크립트 정보의 통합 및 비교를 도식화하여 나타낸다. 버전이 다르면 새로운 USB 저장장치로 인식하며, 버전이 동일하면 이전 삭제된 USB 저장장치의 재삽입으로 인식한다. 그림 5는 디스크립트 통합기에서 사용되는 알고리즘을 나타내고 있다.

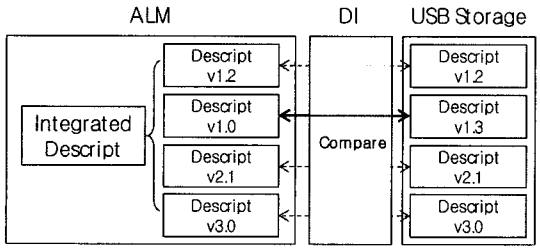


그림 4 디스크립트 통합기를 통한 디스크립트 정보 비교

```
Function Descript_Integrator (Request_Signal, USB_Storage[]) {
 If (Request_Signal = Get_Integrated_Descript) then
   Call Get_Descript (USB_Storage[]);
   Call Add_Descript (USB_Storage[]);
   Call Address_Manager (New_USB_Storage);
 Else If (Request_Signal = Compare_Descript) then
    Call Compare_Descript( );
    IF (Compare_Descript ( ) == 0) then
       Call Address_Manager (OLD_USB_Storage);
       Call Address_Control_Manager(Update_Tuple_in_FILELOG);
       USB_Descript = New_USB_Descript;
    End If
End If
    Store in Address_Log_Memory(USB_Descript);
    Store in Address_Log_Memory(Integrated_Descript);
    Send_Result_to_Host_Controller(Integrated_Descript);
}
Function Compare_Descript ( ) {
 IF (USB_Storage[] =! USB_Storage_ALM[]) then
   Access_Value = 0
 Else  Access_Value = 1
}
```

그림 5 디스크립트 통합기에서 사용되는 알고리즘

• 주소 관리기(Address Manager: AM)

주소 관리기는 시그마 허브에 접근한 USB 저장장치를 구별하기 위한 로컬 주소를 부여 및 관리한다. 주소는 7bit 사용으로 최대 128개를 사용할 수 있으나 0은 디폴트 주소로 할당하지 않는다. 시그마 허브에서 USB 저장장치 제거시 부여된 지역 주소는 삭제되며, 새로운 USB 저장장치 삽입시 디스크립트 통합기의 디스크립트 정보 비교 결과에 따라 동일한 USB 저장장치인 경우 제거 전 사용했던 지역 주소를 할당하며, 동일하지 않을 경우 최상위 주소보다 하나 큰 주소를 부여한다. 또한, 최상위 주소가 126인 경우 지역주소는 1부터 시작하며 빈 주소를 할당한다.

주소 관리기에서 사용하는 주소는 튜플 형태를 구성한다. 튜플은 그림 6과 같이 시그마 허브에 접근한 USB 저장장치를 구분하기 위해 시그마 허브의 포트 번호를 지역 주소와 접근 순서의 쌍으로 표현한 것이다. 포트번호는 시그마 허브의 하드웨어 포트 번호이며 본 논문에서는 4개의 포트를 갖는 프로토타입을 제작하였기 때문에 4개의 포트 번호를 사용한다. 그림 7은 주소 관리기의 튜플 및 제어 순서를 관리하기 위한 알고리즘을 나타내고 있다.

Tuple: Port No (address, sequence)	Port No.	Local Address	Access Order
P1($0000010_{(2)}$,2)	Port 1	$0000010_{(2)}$	2^{nd}
P2($0000001_{(2)}$,1)	Port 2	$0000001_{(2)}$	1^{st}
P3($0000100_{(2)}$,4)	Port 3	$0000100_{(2)}$	4^{th}
P4($0000011_{(2)}$,3)	Port 4	$0000011_{(2)}$	3^{rd}
...		...	...
Px($1111111_{(2)}$,126)	Port x	$1111111_{(2)}$	126^{th}

그림 6 USB 저장장치 제어를 위한 주소 할당

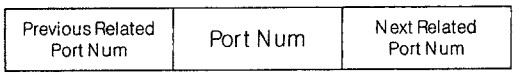
```
Function Address_Manager (USB_Storage) {
    Call Transaction_Checker (Port_Number[]);
    If (USB_Storage == New_USB_Storage) &&
      (Transaction_Checker == USB_INSERT) then
      Call Allocate_Tuple (Port_Number, New_Tuple);
      Call Access_Control_Manager (Insert_Tuple_in_FILELOG);

        Else if (USB_Storage == OLD_USB_Storage) &&
        (Transaction_Checker == USB_INSERT) then
        Call Allocate_Tuple (Port_Number, OLD_Tuple);
        Call Access_Control_Manager (Insert_Tuple_in_FILELOG);

          Else If (USB_Storage == OLD_USB_Storage) &&
          (Transaction_Checker == USB_EXTRACT) then
          Call Free_Tuple(Port_Number);
          Call Access_Control_Manager (Delete_Tuple_in_FILELOG);
        Else Nothing
 End If
}
Function Allocate_Tuple (Port_Number, Tuple_Value) {
  If (Tuple_Value == OLD) then
  Recover Local Address, Access Order in Temporary Memory
  Set Tuple(Port_Number, Local_Address, Access_Order);
 Else
  Get Empty_Local_Address( );
  Get Access_Order( );
  Set Tuple(Port_Number, Local_Address, Access_Order);
}
Function Free_Tuple (Port_Number) {
  Store element in Temporary Memory and Delete in Tuple;
}
```

그림 7 주소 관리기에서 사용되는 알고리즘

• 접근 제어 관리기(Access Control Manager: ACM) 접근 제어 관리기는 호스트 콘트롤러와 패킷 교환을 위한 USB 저장장치의 제어 순서를 관리하는 모듈이다. 제어 순서는 포트 2 ➔ 포트 1 ➔ 포트 4 ➔ 포트 3 ➔ … ➔ 포트 X와 같이 순차적인 형태로 제어 순서를 위한 저장공간인 파일 로그에 저장된다. 순차적인 형태는 그림 6의 튜플을 참고로 작성된다. 시그마 허브가 컴퓨터와 접속 중 새로운 USB 저장장치가 삽입되거나, 접속된 USB 저장장치가 제거될 경우 호스트 컨트롤러로 전송되는 상태 패킷을 차단하고, 통합 디스크립트 정보를 갱신하기 위한 신호를 디스크립트 통합기에 전송한다. 이후 호스트 콘트롤러를 통해 전송되는 데이타 패킷을 순차적으로 저장하기 위한 USB 저장장치에 대한 제어 순서를 관리한다. 제어 순서 변경은 트랜잭션 검사기의 접근 주소 변경 요청 시 일어나며 접근 포트를 변경한다. 또한, USB 저장장치 인식 및 제거 시 재구성 된다 .제어 순서는 파일 로그로 작성하여 주소 로그 메모리에 저장된다. 파일 로그는 USB 저장장치의 제어 순서 및 파일의 분리 저장 유무와 그 연관관계를 표현한다. 분리 저



Related : Port Num, Not Related : 0

그림 8 연관 포트 기록을 위한 제어 순서 저장 구조

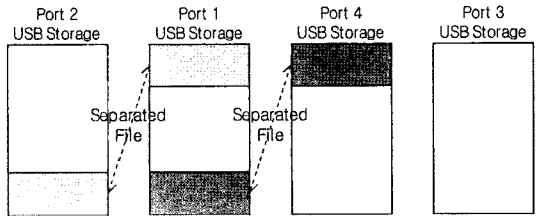


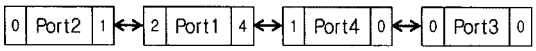
그림 9 USB 저장장치에 저장된 분할된 파일

장된 데이타를 확인 할 수 있도록 그림 8과 같은 구조를 갖는다. 그림 8은 해당 포트와 데이타를 분리 저장한 연관 포트의 번호를 저장할 수 있는 링크구조를 나타내고 있다. 데이타를 분리 저장한 포트가 존재할 경우 해당 분리 저장된 포트번호를 제어 순서 전후 관계에 따라 이전 연관 포트 혹은 다음 연관 포트에 기록한다. 연관된 포트가 없을 경우 0으로 나타내고, 있을 경우에 해당 포트 번호를 적는다.

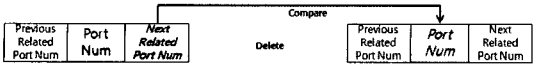
그림 9는 포트 2 ➔ 포트 1 ➔ 포트 4 ➔ 포트 3 순서로 파일이 저장될 경우, 하나의 파일이 분리되어 다른 포트의 USB 저장장치에 저장되는 경우를 나타내고 있다. 포트 2와 포트 1에 하나의 파일이 분할되어 저장되었으며, 포트 1과 포트 4의 USB 저장장치의 경우도, 단일 파일이 저장공간 부족으로 인하여 분할됨을 나타내고 있다.

따라서 그림 10의 (a)와 같이 파일 로그는 포트 2와 포트 1이 연관되었으며, 포트 1과 포트 4가 연관되었음을 표현한다. 만약 포트에 삽입된 USB 저장장치가 삭제되는 경우 해당 링크구조는 삭제되며 제어 순서가 변경된다.

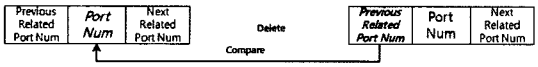
USB 저장장치 제거로 인하여 분리 저장된 파일에 대한 잘못된 읽기 작업이 발생할 수 있다. 일부 내용이 삭제된 단일 파일은 가치가 없으므로 읽기 방지를 통해 오류를 방지한다. 이를 위하여 파일 로그는 링크구조를 통하여 다음과 같은 두 가지 경우로 단일 파일의 분리 저장 여부를 확인한다. 첫 번째 경우는 삭제된 링크구조 이전의 다음 연관 포트 번호와 다음 링크구조의 포트 번호가 다를 경우이다. 그림 10의 (b)는 이러한 경우를 나타내고 있다. 두 번째는 삭제된 링크구조 이후의 이전 연관 포트 번호와 이전 링크구조 포트 번호가 다른 그림 10의 (c)와 같은 경우이다. 두 경우 모두 분리 저장



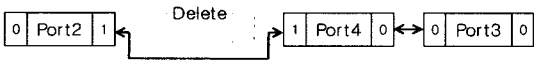
(a) 제어 순서 및 파일 연관을 표현한 파일 로그



(b) 삭제된 링크구조 이전의 다음 연관 포트 번호와 다음 링크구조의 포트 번호가 다를 경우



(c) 삭제된 링크구조 이후의 이전 연관 포트 번호와 이전 링크구조 포트 번호가 다른



(d) USB 저장장치 삭제시 파일 로그 재설정

그림 10 연관포트 및 제어 순서

```
Function Address_Control_Manager (Value) {
If (value==USB_OVER) then
   Set Link_Structure
   Set Previous_Related_Port_Number = Port_Number
   Set Next_Related_Port_Number = Next_Port_Number
If (value==USB_IN) then
  if ( Previous_Related_Port_Number == Port_Number) &&
    Set Next_Related_Port_Number == Next_Port_Number) then
    Do read separated file;
  else
   Do Not read a separated file;
If (value==Insert_tuple_in_FILELOG) then
  Set Link_Structure;
If (value==Update_tuple_in_FILELOG) then
   Update( );
If (value==Delete_tuple_in_FILELOG) then
   Delete Link_Structure; // Linker mismatched
}
```

그림 11 주소 제어 관리기에서 사용하는 알고리즘

된 데이타가 있으므로 해당 데이타에 대한 읽기를 금지한다.

그림 10의 (d)는 그림 10의 (a)에서 포트 1의 USB 저장장치가 제거된 경우 파일 로그의 제어 순서를 나타내고 있다. 포트 2의 이전 연관 포트 및 다음 연관 포트를 확인하여 이전 연관 포트가 0이므로 포트 2 이전에 제어할 USB 저장장치가 없음을 확인한다. 또한, 포트 2의 다음 연관 포트가 1이었으나 제어 순서가 포트 4로 재설정되어 연관 정보가 다르므로 포트 2의 마지막 파일에 대한 읽기 작업을 금지한다. 또한 포트 4의 이전 연관 포트가 포트 1로 되어 있으나 재설정된 포트가 2번

```
Function Transaction_Checker (USB_Storage) {
   Get Packet ( );
   If (Packet == SETUP) then
   Call Descript_Integrator;
   Return USB_INSERT
   If (Packet == NAK) then
   Call Descript_Integrator;
   Return USB_Extract
   If (Packet == IN) then
   Call Address_Control_Manager(USB_IN);
   If (Packet == OUT) then
   Call Address_Control_Manager(USB_OUT);
   If (Packet == STALL) then
   Call Address_Control_Manager(USB_OVER);
}
```

그림 12 트랜잭션 검사기에서 사용하는 알고리즘

으로 연관 정보가 다르므로 포트 4의 처음 저장된 파일의 읽기 작업을 금지한다. 이로서 분리된 파일에 대한 읽기 작업 오류를 막을 수 있다.

• 트랜잭션 검사기(Transaction Checker: TC)
토큰 패킷 및 상태 패킷을 제어한다. 인·아웃 패킷 수신시 접근 제어 관리기에게 신호를 전송하여 로컬 주소에 알맞은 포트로 변경이 일어날 수 있도록 한다. 또한 USB 저장장치의 공간이 없을 경우 발생하는 낙 혹은 스톨 패킷이 호스트 콘트롤러로 전송되어 전송 취소가 발생하지 않도록 한다. 이를 위하여 접근 제어 관리기를 통하여 제어 순서에 맞는 USB 저장장치로 데이타 전송을 실행할 수 있도록 한다. 또한, 저장공간이 부족하여 분리되어 저장된 데이타가 있을 경우, 파일 로그에 대한 연관관계를 표시하여 읽기 과정 중 오류가 발생되지 않도록 한다.

• 주소 로그 메모리(Address Log Memory: ALM)
주소 로그 메모리는 시그마 허브의 동작에 필요한 정보를 저장하는 플래시 메모리이다. 통합 디스크립트 정보, 각 USB 저장장치들의 디스크립트 정보, 튜플, 파일 로그를 저장하며 삭제된 링크구조를 임시 저장한다.

3.2 USB 저장장치 통합 알고리즘

시그마 허브를 통해 다수의 USB 저장장치를 통합하기 위해서 단일 저장장치처럼 사용할 수 있는 제어 알고리즘이 필요하다. 본 장에서는 시그마 허브를 사용 시 발생할 수 있는 상황들을 분류하여 이를 해결하는 알고리즘을 제시한다.

• 컴퓨터에 시그마 허브를 처음 인식하는 경우
컴퓨터가 시그마 허브를 처음 인식하는 경우는 두 가

지로 구분할 수 있다. 시그마 허브에 USB 저장장치가 연결된 후 컴퓨터에 인식하는 경우와 아무것도 연결되지 않은 상태에서 컴퓨터에 인식되는 경우이다. 두 경우 모두 컴퓨터의 호스트 콘트롤러는 시그마 허브로 상태 패킷을 전송하여, 시그마 허브에 저장된 통합 디스크립트 정보를 요구한다. USB 저장장치가 연결되지 않은 초기 인식인 경우 디폴트값으로 구성된 통합 디스크립트 정보가 전송된다. 시그마 허브에 USB 저장장치가 미리 연결된 경우에는 USB 저장장치 디스크립트 정보가 호스트 콘트롤러로 전송되어 개별 장치로 인식되지 않기 위하여, 디스크립트 통합기는 디스크립트 정보를 차단, 시그마 허브를 위한 통합 디스크립트 정보로 가공하여 호스트 콘트롤러에게 전송한다. 디스크립트 전송 요구 시, 디스크립트 통합기는 자신의 포트에 접근한 모든 USB 저장장치의 디스크립트 정보를 수집하여 저장장치 용량 정보를 획득하여 그 전체 용량을 합산한다. 합산된 전체 용량은 통합 디스크립트 정보로 가공되어, 호스트 콘트롤러에게 단일 USB 저장장치가 접근한 것처럼 하나의 디스크립트 정보를 전송하여 인식된다. 접속된 USB 저장장치를 구분하기 위하여 주소 관리기를 통해 지역 주소를 순차적으로 지정한다. 지역주소 및 포트번호와 접근 순서는 튜플로 구성되어 접근제어를 위한 파일 로그로 사용된다. 해당 알고리즘은 그림 13의 (a)이다.

- 시그마 허브에 USB 저장장치를 제거 하는 경우

컴퓨터에 인식되어 동작중인 시그마 허브에 USB 저장치가 제거될 경우, 통합 디스크립트 정보 갱신을 통한 총 저장 용량에 대한 변경 및 지역 주소의 해지가 이루어진다. 또한, 분리된 USB 저장장치에 기록된 데이타가 있을 시, 해당 데이타와 분리 저장된 데이타의 사용을 금지해야 한다. 이를 위하여 다음과 같은 과정을 거친다.

USB 저장장치가 시그마 허브와 분리될 경우, 트랜잭션 검사기는 해당 분리 신호를 감지하여, 디스크립트 통합기를 통해 저장 용량을 갱신하고, 주소 관리기를 통해 해당 주소를 폐기한다. 또한 USB 저장장치 제거로 인한 접근 제어 순서 변경은 접근 제어 관리기를 통하여 이루어진다. 파일 로그에서 해당 링크구조를 제거하여 제어 순서를 재구성한다.

USB 저장장치 제거로 인하여 분리 저장된 파일에 대한 잘못된 읽기 작업이 발생할 수 있다. 이로 인한 오류를 방지하기 위하여, 접근 제어 관리기에서는 포트번호 비교를 통한 읽기 작업을 통제한다. 자세한 설명은 3.1 접근 제어 관리기에 설명되어 있으며, 해당 알고리즘은 그림 13의 (b)이다.

- 시그마 허브에 USB 저장장치를 삽입하는 경우

컴퓨터에 인식되어 동작중인 시그마 허브에 USB 저장장치가 삽입될 경우는 두 가지로 구분한다. 새로운 USB 저장장치가 삽입되는 경우와 이전에 제거되었던 USB 저장장치가 다시 삽입되는 경우이다. USB 저장장치의 삽입시 디스크립트 통합기로 전송된 디스크립트 정보와 주소 로그 메모리에 저장된 해당 포트의 디스크립트 정보를 비교하여 버전이 동일한 경우 이전에 제거된 USB 저장장치로 판단하며 동일하지 않은 경우 새로운 USB 저장장치로 판단한다. 디스크립트 정보는 메타데이타를 포함하고 있어 디스크립트 정보가 동일하면 삽입된 USB 저장장치의 데이타에 변경 일어나지 않았으므로 동일한 USB 저장장치가 삽입되었음을 판단할 수 있다. 따라서 USB 저장장치 제거시 삭제되었던 지역 주소 및 접근 순서, 파일 로그의 링크 구조는 임시 저장공간에서 복원되어 사용된다. 새로운 USB 저장장치일 경우 저장 용량을 갱신하고, 새로운 지역 주소를 할당받는 첫 번째 경우와 동일하다. 해당 알고리즘은 그림 13의 (c)이다.

- 데이타 쓰기 작업 시 USB 저장장치의 저장 용량을 넘어서는 경우

호스트 콘트롤러에서 전송한 데이타가 USB 저장장치 저장 용량을 넘어서는 경우, USB 콘트롤러는 낙 패킷 혹은 스톨 패킷을 호스트 콘트롤러에 전송하여 해당 파일의 전송을 취소한다. 저장용량 부족에 인한 전송취소를 방지하기 위하여 시그마 허브는 낙 패킷 혹은 스톨 패킷을 트랜잭션 검사기를 통해 인지하여 호스트 콘트롤러로 전송하지 않는다. 트랜잭션 검사기를 통하여 해당 USB 저장장치의 저장공간이 없어 전송취소를 원하는 신호를 받으면, 접근 제어 관리기는 바로 다음 USB 저장장치의 주소로 데이타 패킷을 전송할 수 있도록 파일 로그를 참고하여 접근 포트를 변경하고, 링크 구조에 해당 포트들에 대한 연관관계를 설정한다. 해당 알고리즘은 그림 13의 (d)이다.

- USB 저장장치에서 데이타 읽기 작업을 할 경우

호스트 콘트롤러에서 데이타의 읽기를 진행하는 중 발생된 낙 패킷 혹은 스톨 패킷은 호스트 콘트롤러로 전송되어 읽기 실패가 되며 해당 프로세스가 취소된다. 시그마 허브는 다수의 USB 저장장치에 파일을 분산 저장했기 때문에 호스트 콘트롤러가 원하는 데이타를 모두 읽을 때까지 작업을 중단해서는 안 된다. 따라서 트랜잭션 검사기는 USB 저장장치에 저장된 모든 파일에 대한 읽기 작업 후 발생되는 낙 패킷 및 스톨 패킷을 인지하여 호스트 콘트롤러로 전송되지

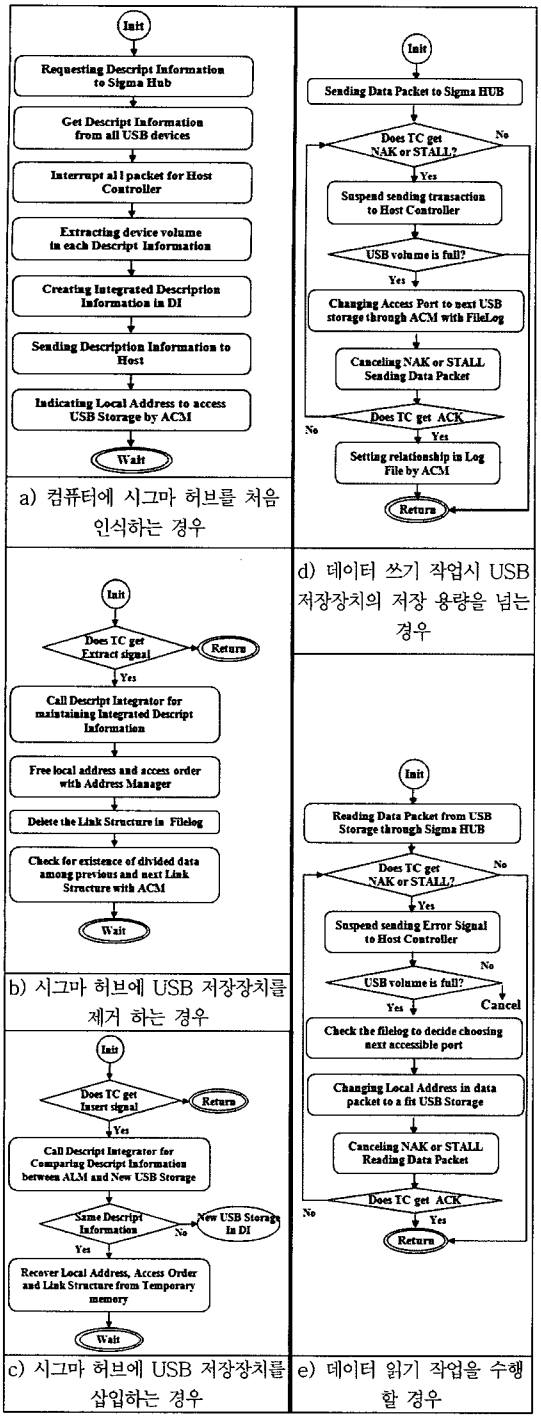


그림 13 시그마 컨트롤러에서 수행하는 USB 저장장치 통합 알고리즘

않게 한다. 접속 제어 관리기는 파일 로그의 접속 제어 순서에 따라 읽기 작업을 수행하여 하나의 대용량 저장장치를 사용하는 효과를 나타낸다. USB 저장장치 제거를 통한 접근 제어 순서의 변경이 읽어난 경우 접근 제어 장치는 분산된 데이타 파일에 대한 읽기 작업을 수행하지 않고 다음 데이타 파일에 대한 읽기 작업을 수행한다. 해당 알고리즘은 그림 13의 (e)이다.

4. 구현 및 평가

이 장에서는 소용량 USB 저장장치를 단일한 논리적 저장장치로 사용하기 위한 시그마 허브를 구현하였다. 시그마 허브는 4개의 포트로 구성되어 있으며 트랜잭션을 제어하기 위한 시그마 콘트롤러가 내장되어 있다. 시뮬레이션을 위하여 시그마 허브에 표 1과 같이 4가지 USB 저장장치를 접속하여 총 1297 MB의 파일을 저장한다. 저장에 사용된 파일은 총 410개이며 폴더는 35개이다. 시그마 허브는 포트에 삽입된 USB 저장장치 순으로 지역 주소를 할당하며, 데이타 저장 시 해당 주소의 USB 저장장치를 접근한다. 따라서 본 실험에서는 128MB(포트 1), 32MB(포트 3), 2GB(포트 4), 512MB (포트 2)의 순서로 USB 저장장치를 삽입하여 읽기, 쓰기 및 USB 저장장치의 탈착 시 저장된 파일에 대한 제어 여부를 실험한다. 일반적인 USB 허브에 연결된 저장 장치에 1297MB의 데이타를 저장할 경우, 사용자는 각 저장장치에 알맞은 크기로 파일을 분리하여 저장해야 한다. 그러나 본 논문에서 제안한 시그마 허브는 단일 저장장치를 이용하듯 데이타 용량을 고려하지 않고 파일을 저장할 수 있다.

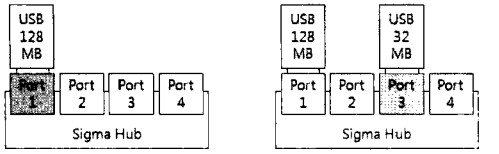
자세한 평가를 위하여 그림 14와 같이 6가지 상황을 시나리오로 작성하여 시뮬레이션을 실행한다.

• 컴퓨터에 시그마 허브를 인식하는 경우

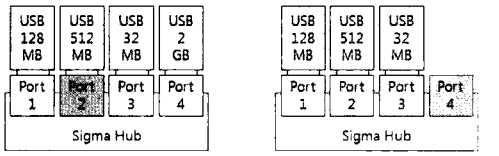
그림 14의 (a)와 같이 시그마 허브는 포트 1에 삽입된 128MB 용량의 USB 저장장치의 디스크립트 정보를 통합하여 컴퓨터에게 121MB 용량의 시그마 허브 저장장치로 인식된다. 또한 그림 14의 (b)와 같이 시그마 허브 3에 새로운 USB 저장장치가 삽입될 경우 통합 디스크립트 정보를 갱신하여 150MB의 단일 저장장치로 인식된다. 시그마 허브는 2개의 USB 저장

표 1 시그마 허브 프로토타입에 접속한 USB 저장장치

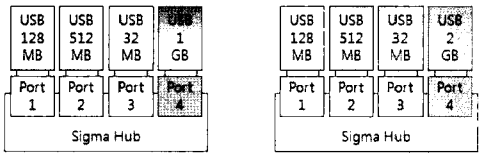
USB 저장장치 종류	사용 가능한 저장공간
32 MB	29 MB
128 MB	121 MB
512 MB	495 MB
2 GB (2048MB)	1956 MB
Total: 2720 MB	Total: 2601 MB



a) USB 저장장치 인식 b) 새로운 포트에 추가 인식



c) 데이터 쓰기 및 읽기 d) 사용 중인 저장장치 제거



e) 새로운 저장장치 인식 f) 기존 저장장치 재인식

그림 14 시뮬레이션을 위한 6가지 상황

Tuple: Port No (address, sequence)	Port No.	Local Address	Access Order
P1(0000001$_{(2)}$,1)	Port 1	0000001$_{(2)}$	1st
…	Port 2	…	…
P3(0000010$_{(2)}$,2)	Port 3	0000010$_{(2)}$	2nd
…	Port 4	…	…

그림 15 2개의 USB 저장장치가 인식 된 경우 튜플

저장장치의 지역 주소를 할당하기 위하여 그림 15와 같은 튜플을 작성한다.

작성된 튜플을 통하여 그림 16과 같은 파일 로그가 작성된다. 파일 로그의 원소인 링크구조는 데이타의 저장이 실행되지 않았기 때문에 데이타의 분산 저장 위치를 알려주는 연관 포트 번호가 모두 0으로 설정된다. 파일 로그의 순서는 시그마 허브가 읽기 및 쓰기를 수행할 때 접근할 USB 저장장치의 순서를 나타낸다.

포트 4와 포트 2에 차례대로 USB 저장장치를 삽입하면, 삽입 순서에 따라 그림 16과 같은 파일 로그가 생성된다. 그림 17은 4개의 USB 저장장치가 시그마 허브를 통하여 모두 인식된 화면을 나타내고 있다. 그림 17의 (a)와 같이 단일한 장치로 인식되며, 그 저장용량은 그림 17의 (b)와 같이 시그마 허브에 접속된 USB 저장장치 저장용량 총 합과 같다.

- 데이타 쓰기 작업시 USB 저장장치의 저장 용량을 넘어서는 경우 및 USB 저장장치에서 데이타 읽기 작업을 할 경우

	접근 제어 순서
포트 1삽입	0 Port1 0
포트 3삽입	0 Port1 0 ↔ 0 Port3 0
포트 4삽입	0 Port1 0 ↔ 0 Port3 0 ↔ 0 Port4 0
포트 2삽입	0 Port1 0 ↔ 0 Port3 0 ↔ 0 Port4 0 ↔ 0 Port2 0

그림 16 USB 저장장치 삽입시 생성되는 접근 제어

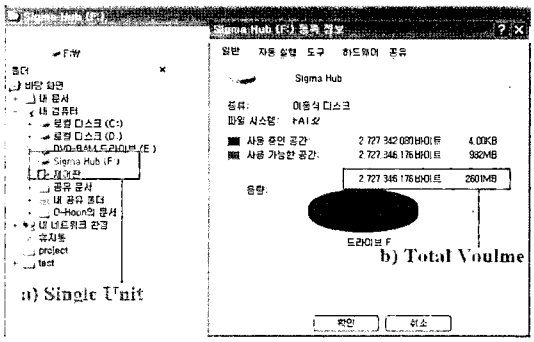


그림 17 시그마 허브를 통한 USB 저장장치 인식

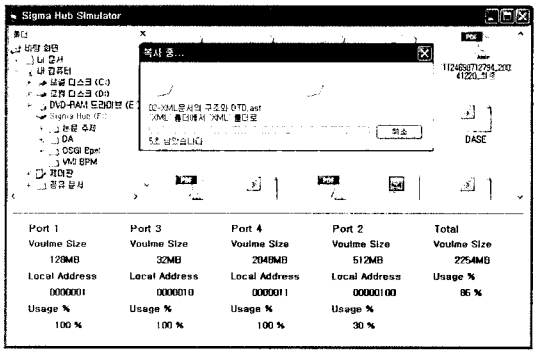


그림 18 데이타 읽기 및 쓰기를 위한 시뮬레이터 GUI

시그마 허브를 통해 데이타를 읽거나 쓰는 경우, USB 저장장치 내부적인 변화를 살펴보기 힘들다. 따라서 본 논문에서는 GUI를 통하여 그림 14의 (c) 상황인 파일의 쓰기 및 읽기를 탐색 창을 통하여 실행할 때, USB 저장장치의 내부적인 변화를 살펴볼 수 있는 그림 18과 같은 시그마 허브 시뮬레이터를 구현하였다. 읽기는 쓰기 동작과 유사하므로 쓰기에 대한 시뮬레이션을 통하여 읽기 동작에 대한 결과를 유추한다.

그림 18은 컴퓨터에 저장된 파일을 시그마 허브로 저장하고 있는 화면이다. 탐색 창 하부는 시그마 허브에 접근한 USB 저장장치의 상태를 차례대로 나타내고 있다. 파일 로그를 참고하여 왼쪽부터 차례로 물리적 저장공간에 데이타를 저장하고 필요시 다음 주소에

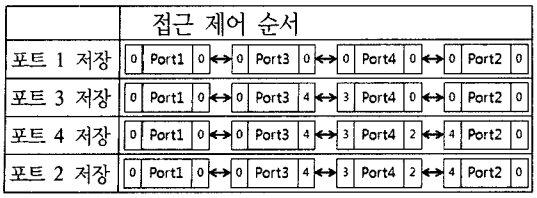


그림 19 파일 분할 저장에 의한 연관관계

해당하는 저장공간을 사용한다. 저장하려는 데이타의 총 용량은 2254 MB이며 포트 3에 저장되던 마지막 데이타는 분할되어 포트 4의 처음 파일로 저장되었다. 또한 포트 4에 저장되던 마지막 데이타도 분할되어 포트 2에 분할 저장되었다. 따라서 파일 로그는 그림 19와 같이 작성된다.

그림 19의 접근 제어 순서 및 파일 분리 관계를 통하여 읽기 작업시 분리된 파일에 대한 읽기 작업을 실행 할 수 있다.

- 시그마 허브에 USB 저장장치를 제거하는 경우

시그마 허브에 접근한 4개의 USB 저장장치 중 본 시뮬레이션을 위하여 그림 14의 (d)와 같이 포트 4의 2GB USB 저장장치를 제거한다. 제거한 저장장치에 저장된 데이타는 접근이 불가능 하며, 제거한 저장장치에 분산 저장된 파일은 읽기 작업이 이루어질 수 없다. 이는 파일 로그에 저장된 링크구조가 나타내는 연관 포트 번호를 참고하여 판단할 수 있다. 그림 20을 보면 포트 4가 삭제되면서 포트 3과 포트 2가 연속적으로 연결되었다. 그러나 포트 3의 다음 연관 포트 번호인 4는 포트 2와 같지 않기 때문에 포트 3에 저장된 마지막 파일에 대한 읽기 작업은 수행되지 않는다. 포트 2의 이전 연관 포트 번호인 4인데 반해 이전 포트 번호는 3번이므로 포트 2번에 분할되어 저장된 데이타는 읽기 작업을 수행하지 않는다. 따라서 분할 된 파일에 대한 잘못된 접근에 의한 오류를 방지할 수 있다.

- 시그마 허브에 새로운 USB 저장장치를 삽입하는 경우

위의 USB 저장장치 제거 실험을 통해 비어있는 포트 4에 새로운 USB 저장장치를 인식한다. 본 실험에서는 1GB USB 저장장치를 삽입한다. 새로운 장치 인식시 시그마 콘트롤러는 디스크립트 정보를 갱신하여 호스트 콘트롤러로 전송한다. 또한, 새로 인식된 저장장치는 최상위 주소보다 상위 지역 주소를 할당 받는다. 작성된 튜플을 통하여 그림 22와 같은 파일 로그가 작성된다.

- 시그마 허브에 기존 USB 저장장치를 재인식하는 경우

위의 새로운 USB 저장장치 삽입 실험 후 포트 4의

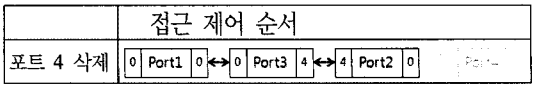


그림 20 재설정된 접근 제어 순서

Tuple: Port No (address, sequence)	Port No.	Local Address	Access Order
P1($0000001_{(2)}$,1)	Port 1	$0000001_{(2)}$	1^{st}
P2($0000100_{(2)}$,4)	Port 2	$0000100_{(2)}$	4^{th}
P3($0000010_{(2)}$,2)	Port 3	$0000010_{(2)}$	2^{nd}
P4($0000101_{(2)}$,5)	Port 4	$0000101_{(2)}$	5^{th}

그림 21 최상의 지역주소를 사용한 튜플

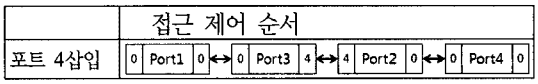


그림 22 재설정된 접근 제어 순서

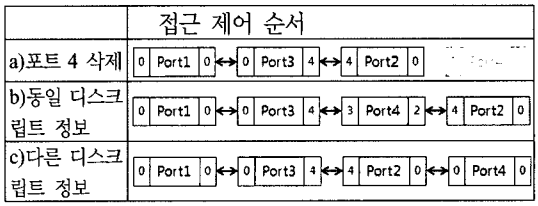


그림 23 디스크립트 정보 비교 결과에 따른 접근 제어 순서

1GB USB 저장장치를 삭제한다. 포트 4에 이전에 삭제한 사용하던 2GB USB 저장장치를 인식한다. USB 저장장치 인식 시 시그마 콘트롤러는 디스크립트 정보를 갱신하면서, 주소 로그 메모리에 저장된 인식했던 USB 저장장치에 대한 디스크립트 정보와 버전을 비교한다. 비교한 디스크립트 정보가 동일시, 기존 USB 저장장치 저장공간의 변형없이 재인식된 것으로 간주하여 파일 로그를 변경한다. 그림 23은 변경된 파일 로그를 보여주고 있다. 만일 디스크립트 정보가 다를 경우, 새로운 USB 저장장치로 인식하여 그림 23의 (c)와 같이 제어 순서가 변경되어 실험 3)의 경우와 동일한 제어 순서를 유지한다.

5. 결 론

반도체 기술의 발전으로 메모리에 대한 집적도가 향상되고 있으며, 이에 따라 많은 저장 용량을 갖는 휴대용 저장장치가 개발되고 있다. 이에 따라, 기존에 생산된 USB 저장장치는 장치가 갖는 저장용량의 협소함 때문에 사용되지 않거나, 폐기되고 있다. 따라서 본 논문은 현재 상용화되어 있는 다양한 용량의 USB 저장장치

들 효율적으로 사용하기 위한 여러 가지 방안 중, 스토리지 통합을 위한 방안을 제안한다. 본 논문은 USB 저장장치들을 USB 허브에 연결하여 통합된 단일 대용량 저장창치로 사용할 수 있는 시그마 허브를 디자인하고 프로토타입으로 구현하였다. 또한, 복수개의 USB 저장장치를 제어하기 위한 트랜잭션과 이를 위한 알고리즘을 제안하여 구현하였으며, 시뮬레이션을 통하여 그 실용성을 검증하였다. 시그마 허브를 통해, 사용되지 않는 저용량 USB 저장장치를 통합하여 휴대용 대용량 저장장치로 이용할 수 있을 것이다.

참 고 문 헌

[1] Universal Serial Bus Specification, Revision 2.0, April 27, 2000.

[2] Inter-Chip USB Supplement to the USB 2.0 Specification, Revision 1.0, March 13th, 2006.

[3] Ravi, N.; Narayanaswami, C.; Rosu, M.-C.; Raghunath, M., "Securing Pocket Hard Drives," Pervasive Computing, IEEE Volume 6, Issue 4, Oct.-Dec. 2007 Page(s):18 23.

[4] Heikkila, Faith M., "Encryption: Security Considerations for Portable Media Devices," Security & Privacy Magazine, IEEE Volume 5, Issue 4, July-Aug. 2007 Page(s):22-27.

[5] Min-An Song, M.-A.; Sy-Yen Kuo, S.-Y.; I-Feng Lan, I.-F, "A Low Complexity Design of Reed Solomon Code Algorithm for Advanced RAID System," Consumer Electronics, IEEE Transactions on Volume 53, Issue 2, May 2007 Page(s):467-473.

[6] Soyoon Lee; Hyokyung Bahn, "Data allocation in MEMS-based mobile Storage devices," Consumer Electronics, IEEE Transactions on Volume 52, Issue 2, May 2006 Page(s):472-476.

[7] Jungong Han; Farin, D.; de With, P.H.N.; Weilun Lao, "Real-time video content analysis tool for consumer media Storage system," Consumer Electronics, IEEE Transactions on Volume 52, Issue 3, Aug. 2006 Page(s):870-878.

[8] Sunhwa Park; Seong-Young Ohm, "New techniques for real-time FAT file system in mobile multimedia devices," Consumer Electronics, IEEE Transactions on Volume 52, Issue 1, Feb. 2006 Page(s):1-9.

최 오 훈

2000년 고려대학교 컴퓨터학과 학사. 2002년 고려대학교 컴퓨터학과 석사. 2009년 고려대학교 컴퓨터학과 박사. 관심분야는 메타데이타, 데이타 품질, 상황인지, 온톨로지 엔지니어링

임 정 은

2000년 관동대학교 컴퓨터학과 학사. 2002년 숙명여자대학교 컴퓨터학과 석사. 2004년 고려대학교 컴퓨터학과 박사수료. 관심분야는 시멘틱웹, 메타데이타, 온톨로지, 유비쿼터스 헬스케어 (U-Healthcare)

나 홍 석

1994년 고려대학교 전산과학 학사. 1996년 고려대학교 컴퓨터학과 석사. 2004년 고려대학교 컴퓨터학과 박사. 2002년~현재 한국디지털대학교 디지털정보학과 교수. 관심분야는 소프트웨어공학, 데이타아키텍처, 메타데이타, 온톨로지, 디지털콘텐츠, 데이타 품질, 이러닝(e-learning)

백 두 권

정보과학회논문지 : 데이타베이스
제 35 권 제 3 호 참조