역방향 레딕스 방식에 의한 고속 하드웨어 정렬기의 설계 및 구현

박 희 순[†] · 전 종 연[††] · 김 희 숙[†††]

요 약

레딕스 정렬 방식은 알고리즘이 단순하며 하드웨어 구현이 비교적 용이하다는 장점이 있으나, 정렬할 데이터를 2회의 탐색과정을 통해 논리 정보 0과 1을 구분 저장한다는 단점이 있다. 본 논문은 레딕스 정렬에 있어 2회의 탐색을 1회로 줄이고 논리 0의 정보는 하위 주소로 부터 오름차순으로, 논리 1의 정보는 하위 주소로부터 내림차순으로 저장하도록 하여 분류 속도를 높이는 새로운 알고리즘을 개발하고 이를 하드웨어로 구현 한 후 그 실험 결과에 대하여 설명한다. 구현된 하드웨어는 별도의 메모리, 레지스터, 카운터, 비교기 등으로 구성된다. 본 논문의 시뮬레이션에서 소프트웨어 방법은 8비트 데이터 만개를 정렬하는데 54.9ms가 소모되고, 하드웨어 방법은 5.3ms의 시간이 소모되었다.

A Design and Implementation of High Speed Hardware Sorter with Reverse Radix Method

Hee Soon Park[†] · Jong Yun Chun[††] · Hee Sook Kim[†††]

ABSTRACT

Radix sort scans the data twice in a pass, to search bit 0s of the items being sorted and store them into the lowest address, and to search bit 1s and store them into the following addresses. This doubles the sorting time. In this paper, we introduce Reverse Radix Sort Algorithm, in which the data being sorted are sacnned just once and write upward from the lowest address if it is 0 and downward from the highest address if it is 1. The algorithm is simple and the hardware sorter implemented by this method shows very high sorting speed. Hardware implemantation requires two separate pocket memories, register, an upward increasing address counter, a downward decreasing address counter, and comparator. The software simulation of Reverse Radix Sort Algorithm performs sorting in the speed of 54.9ms per 10 thousand of 8 bit digit data, but the hardware sorter spends 5.3 ms to sort the same number of data.

1. 서 론

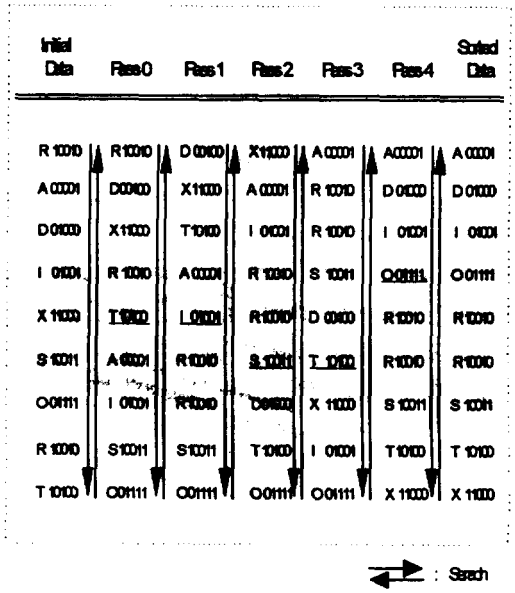
데이터 정렬 작업은 컴퓨터에 의한 정보처리 분야에서 가장 중요한 비중을 차지하고 또한 가장 많은 시간을 소모하는 작업의 하나이다. IBM사의 고객중 97%의 고객이 데이터 정렬 소프트웨어를 장착하였으며, Knuth는 컴퓨터 가동 시간중 25%가 데이터 정렬 작업에 소모된다고 지적하였다[1].

지금까지 여러종류의 소프트웨어[1] 또는 하드웨어[8,9,10] 정렬기가 개발되어 왔으며 지금도 그 연구는 진행 중이다. 하드웨어 정렬기에 관한 연구는 동경

• 본 논문은 '95학년도 원광대학교 교내연구비의 지원에 의해 연구됨.
† 정 회 원:원광대학교 컴퓨터공학과 교수
†† 준 회 원:원광대학교 대학원 컴퓨터공학과 석사
††† 준 회 원:원광대학교 교육대학원 컴퓨터공학과 박사과정
논문접수:1995년 11월 13일, 심사완료:1996년 5월 17일

대학의 T.Kisuregawa교수팀에 의해서 활발히 진행되고 있으며, TTL을 이용하여 구현하였던 정렬기들을 최근에는 VLSI기법으로 다시 설계한 논문을 발표하고 있다[7, 9, 10]. 하드웨어 정렬기는 버블과 합병의 알고리즘에 근거하여 구현 되었지만[2..6] 본 논문은 레딕스 교환 방식[14]에서 그 처리 과정과 저장 방식을 달리한 새로운 역방향 레딕스 정렬 알고리즘(Reverse Radix Sort Algorithm)을 소개하고 이 방식으로 동작하는 하드웨어 정렬기를 설계하고 구현한다. 본 정렬기는 2개의 메모리와 카운터, 쉬프트 레지스터, 논리 게이트 및 제어부로 구성된다. 최초의 정렬할 데이터의 입력과 정렬된 데이터의 출력시에만 주 컴퓨터와 교신하고 정렬 과정의 모든 작업은 독립적, 하드웨어적으로 동작된다.

동일 처리 방식으로 수행한 소프트웨어 시뮬레이션에서 정렬 속도가 범용 데이터베이스 소프트웨어(FoxBase)보다 10분의 1이하로 고속 처리됨을 보여주었고, 하드웨어 정렬기는 다시 소프트웨어보다 10분의 1이상 빠른 속도를 보였다.

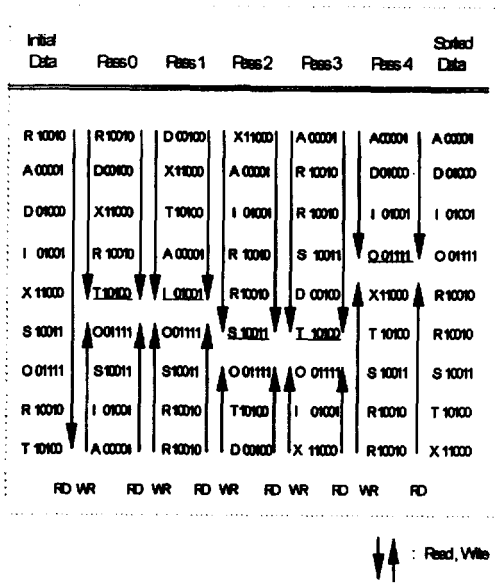


(그림 1) 레딕스 교환 정렬 과정
(Fig. 1) Processing radix exchange sort

2. 역방향 레딕스 정렬 알고리즘

정렬 방식은 비교(comparison)와 배분(distribution)으로 나눌 수 있다. 비교법은 두 자료의 대소를 비교하여 정렬하는 방법으로 대부분의 정렬 방식이 이에 속하며, 배분법은 각 자리수의 기수의 양을 배분하는 것으로써 레딕스 교환 정렬이 이에 속한다.

레딕스 교환 정렬은 각 데이터들을 구성하고 있는 비트 상태에 따라 정렬하는 방식으로써, 그 처리 과정이 단순하고 발생되는 중간 정보의 수가 비교적 적으며 속도가 빠르다는 장점이 있다. (그림 1)은 레딕스 교환 정렬 과정이며 각 알파벳의 아스키 코드 값을 입력으로 받아서 최하위 비트(LSB)부터 시작하여 0이면 하위주소에, 1이면 상위주소에 보관한다. 최초 분류할 데이터중 처음 데이터("10010")의 최하위 비트부터 검색하여 1회째 탐색시 0인 것만 별도 메모리의 하위주소부터 기록하고, 다음 2회째 탐색시 1인 것만 메모리의 다음 번지부터 차례로 기록한다. 이러한 과정으로 한번의 패스가 끝나고, 두 번째 패스에서 이전 패스의 결과를 같은 방식으로 반복하며 이를 비트 수 (5회)만큼 반복하면 데이터는 정렬된다.



(그림 2) 역방향 레딕스 정렬 과정
(Fig. 2) Processing reverse radix sort

위의 방식에서 각 패스는 이전 패스의 결과를 두 번 탐색하여야 한다는 단점이 있다.

본 논문에서는 이를 1회의 탐색으로 처리될 수 있도록 각 패스의 정렬된 데이터의 보관방식을 (그림 2)와 같이 변경하였다. 첫 번째 데이터부터 시작하여 n번째 최하위 비트(패스 0일 경우 0번째 비트)가 0이면 최하위 주소부터, 1이면 최상위 주소부터 저장한다. 다음 데이터의 최하위 비트가 0이면 최하위 주소를 증가하여 저장하고, 1이면 최상위 주소를 감소하여 저장한다. 입력 데이터에 대한 1번의 탐색으로 분류 저장한 후 1번 패스가 끝나고 이를 n번 반복하면 모든 데이터 정렬이 끝난다.

메모리에 위치한 최종 결과 데이터는 레딕스 교환 정렬에서는 순차적으로 저장되지만, 역방향 레딕스 정렬은 최상위 비트(MSB)의 0과 1에 따라 양분된다. 0인 비트들은 순차적으로 저장되고, 1인 비트들은 메모리의 최상위 주소로부터 역방향으로 저장된 최종 결과를 얻을 수 있다.

역방향 레딕스 정렬 알고리즘은 다음과 같으며, 알고리즘의 상위 부분과 하위 부분은 기능상 거의 동일하며 각각 짝수 번째와 홀수 번째 패스에 해당한다.

【 알고리즘 】 Reverse Radix Sort Algorithm

```
procedure reveseradix(N, b : integer);
begin
 for pass:=0 to dataFields do
  begin { pass n }
   if (slr and $AA)=1 then
   begin { MA --> MB }
     Tcua:=Dcua; Tcda:=Dcda;
     Dcua:=0;  Dcub:=0;  Dcda:=N-1;  Dcdb:=N-1;

     for Dcua=0 to Dcua<Tcua do Dcua:=Dcua+1;
     { upper address, or 0 bits }
     if (MA[Dcua] and slr)=0
     then MB[Dcub]:=MA[Dcua], Dcub:=Dcub+1;
     else MB[Dcdb]:=MA[Dcua], Dcdb:=Dcdb-1;

     for Dcda=0 to Dcda>Tcda do Dcda:=Dcda-1;
     { under address, or 1 bits }
     if (MA[Dcda] and slr)=0
     then MB[Dcub]:=MA[Dcda], Dcub:=Dcub+1;
     else MB[Dcub]:=MA[Dcda], Dcdb:=Dcdb-1;
    slr shl 1
   end; { MA --> MB }

   else
   begin { MB --> MA }
     Tcub:=Dcub; Tcdb:=Dcdb;
     Dcua:=0;  Dcub:=0;  Dcda:=N-1;  Dcdb:=N-1;

     for Dcub=0 to Dcub<Tcub do Dcub:=Dcub+1;
     { upper address, or 0 bits }
     if (MB[Dcub] and slr)=0
     then MA[Dcua]:=MB[Dcub], Dcua:=Dcua+1;
     else MA[Dcda]:=MB[Dcub], Dcda:=Dcda-1;

     for Dcdb=0 to Dcdb>Tcdb do Dcdb:=Dcdb-1;
     { under address, or 1 bits }
     if (MA[Dcdb] and slr)=0
     then MA[Dcua]:=MB[Dcdb], Dcua:=Dcua+1;
     else MB[Dcda]:=MA[Dcdb], Dcda:=Dcda-1;
     slr shl 1
    end; { MB --> MA }
  end; { pass n }
end. { reverse radix }
```

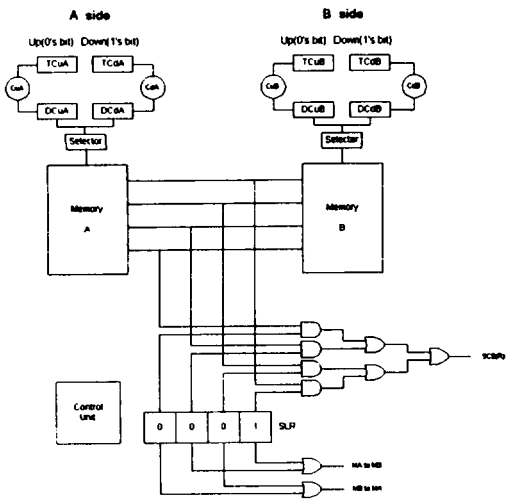
3. 하드웨어 구성 및 기능

2장에서 제시한 알고리즘으로 동작할 수 있는 하드웨어를 구성하기 위하여 하위주소와 상위주소의 현재 값을 보관할 수 있는 증가 및 감소 카운터를 설치하고, 각 패스마다의 n번째 비트 상태(0과 1)를 검색할 수 있는 비교장치(comparator)와 0 데이터와 1 데이터의 총수를 보관할 수 있는 레지스터를 설치한다.

3.1 역방향 레딕스 하드웨어 정렬기의 구성

(그림 3)은 설계된 하드웨어 구성도이며, 상위주소와 하위주소의 상태를 보관하는 4개의 Load/Count 레지스터와 정렬할 데이터를 보관하는 2개의 메모리와 현재 진행되는 정렬 비트를 알리고 종료 여부를 판단할 수 있는 쉬프트 레지스터와 제어부 및 논리게이트들로 구성된다.

역방향 레딕스 하드웨어 정렬기는 4비트 데이터를 처리할 수 있게 구성되어 있다. 최초 분류할 데이터의 입력과 분류된 데이터의 출력시에만 주 컴퓨터와 교신하며, 정렬 과정은 독립적으로 동작한다.



(그림 3) 하드웨어 구성도
(Fig. 3) Configuration of hardware

3.2 Address 회로의 레지스터 기호 및 기능

(1)최종 상위 주소 보관 레지스터(Terminal Count Register up A/B:TCuA/TCuB)

TCuA/TCuB는 각 패스마다 한번의 패스가 끝난 후 카운트 된 0 비트를 키로 갖는 데이터의 총수를 보관하는 레지스터이다. 첫 번째 실행되는 패스(Pass=0)일 경우 TCuA/TCuB는 주 컴퓨터로부터 메모리에 로드 되어질 입력 데이터의 총수를 보관한다. 패스가 진행되는 동안 TCuA/TCuB에 보관된 데이터의 총수는 변경되지 않는다.

TCuA/TCuB에 보관되어 있는 정렬될 데이터의 총수는 패스 과정 중에 메모리에 저장되어 있는 데이터를 Read-Write하면서 증가, 감소된 DCuA/DCuB와 비교하여 0비트를 키로 갖는 데이터의 정렬이 모두 끝났는지를 알 수 있다. TCuA/TCuB는 한번의 패스 수행이 끝나면 0비트를 키로 갖는 데이터의 총수를 보관한다.

(2)상위 주소 보관 레지스터(Data Count Register up A/B:DCuA/DCuB)

정렬할 데이터가 저장된 쪽에서는 0 비트인 데이터의 현재 read 주소를 카운트하고, 정렬된 데이터가 저장되는 쪽에서 0 비트인 데이터의 보관 주소를 카운트한다.

각 패스의 초기에 0번지로 클리어되고 해당 비트가 0인 데이터가 저장될 때마다 1씩 증가된다. 임의의 패스가 끝나고 다음 패스 시작전에 최후의 DCuA/DCuB 값은 0인 비트의 총수이므로 이 값을 TCuA/TCuB로 복사하고, 다음 패스의 최종 데이터 수 비교 값으로 사용된다.

(3)최종 하위 주소 보관 레지스터(Terminal Count Register down A/B:TCdA/TCdB)

TCdA/TCdB는 TCuA/TCuB와 유사한 기능을 하는 레지스터로써 TCdA/TCdB는 한번의 패스가 끝난 후 카운트 된 1비트 데이터의 총수를 보관하는 레지스터이다. TCdA/TCdB에 저장된 데이터의 수는 패스 과정 중 DCdA/DCdB와 비교되어 두 값이 동일할 때 비트가 1인 데이터의 분류(Read)를 끝내고 다음 패스로 진행한다.

(4)하위 주소 보관 레지스터(Data Count Register down A/B:DCdA/DCdB)

위 (2)의 기능과 유사하고, 1인 비트의 데이터가 보관될 주소를 카운터하는 레지스터이다. 패스 초기에 모든 비트를 1로 초기화하고 분류된 데이터가 저장될 때마다 DCdA/DCdB는 1씩 감소한다. 패스가 끝나면 최후의 DCdA/DCdB에 저장되어 있는 값은 TCdA/TCdB로 복사되고 다음 패스의 데이터 read시 최종 값으로 사용된다.

(5)상위 주소 비교기(Comparator up A/B:CuA/CuB)

메모리 A에서 B로 진행되는 패스일 때 현재 read 되는 데이터의 주소 DCuA와 최종 주소 TCuA와 동일한가를 판단하여 동일하면 CuA는 1로 설정된다. CuA가 1로 설정되면 0인 데이터의 정렬이 끝났음을 알리고, 1인 데이터의 정렬이 시작된다.

CuB는 DCuB와 TCuB의 상태를 비교하여 CuA와 동일한 동작을 하며 메모리 A에서 B로 진행되는 패스일 때 유효하다.

(6)하위 주소 비교기(Comparator down A/B:CdA/CdB)

현재 진행되는 비트가 1인 데이터에 적용되며, CuA/CuB와 유사한 기능을 한다. CdA/CdB가 1로 설정되면 현 패스의 정렬이 끝난다.

3.3 메모리 회로의 레지스터 기호 및 기능

(1)메모리(Memory A/B:MA/MB)

분류할 데이터와 분류된 데이터를 보관하는 메모리로써 비트 수와 주소의 크기는 저장될 데이터의 비트수, 크기에 관계된다.

(2)정렬 제어 신호(Sort Control Signal:SCS)

정렬 제어 신호는 현재 진행 중인 데이터의 비트가 0인지 1인지 판단하고, 현재의 데이터를 저장할 메모리의 하위 주소(0비트 데이터)에 보관할 것인지, 상위 주소(1비트 데이터)에 보관할 것인지를 제어한다. 정렬 제어 신호는 현재 진행중인 데이터의 비트와 현재 진행중인 패스 신호를 입력 받아 결정된다. 패스 신호는 SLR 레지스터에서 입력 받는다.

(3)쉬프트 레프트 레지스터(Shift Left Register:SLR)

데이터 비트 수와 동일한 비트 수를 갖는 SLR은 현재 진행되고 있는 패스가 몇 번째 패스인가를 알 수 있다. 초기의 SLR는 1로 설정한 후 패스 0을 시작하고, 한번의 패스가 끝날 때마다 왼쪽으로 1비트씩 이동한다. SLR의 비트 중 1의 위치는 데이터의 read/write의 방향을 표시한다. SLR의 1의 위치가 짝수(0, 2,4…) 위치에 있을 때는 MA에서 데이터를 read하여 MB에 저장하고(MA MB), SLR의 1의 위치가 홀수(1,3,5…) 위치에 있을 때는 MB에서 데이터를 Read하여 MB에 저장한다(MB MA).

4. 역방향 레딕스 하드웨어 정렬기의 동작 과정

4.1 초기화 과정

주 컴퓨터와 하드웨어 정렬기의 데이터 교환은 다음과 같은 초기화 단계를 걸친다.

단계 1. 최초 분류할 데이터들은 주 컴퓨터로부터 memory A에 load한다.

단계 2. memory A의 데이터의 총 개수는 DCuA에 저장된다.

단계 3. SLR의 최하위 비트를 1로 설정한다.

단계 1.2.3의 초기화 과정이 이루어진 후, 주 컴퓨터에서 정렬 시작 신호(b)가 1이면 정렬기는 (그림 4)와 같은 흐름도에 따라 독립적인 동작을 하게 된다. 정렬 과정이 완료되면 정렬 완료 신호(w)는 1로 설정되며, 주 컴퓨터로 최종 결과 데이터가 전송된다. 그림의 상위부분은 정렬할 비트 번호가 짝수(n=0, 2, …)일 때이고, 하위 부분은 홀수(n=1, 3, 5,…)일 때의 과정이다(비트의 번호와 패스의 번호는 동일). 각각의 패스 과정은 다음과 같다.

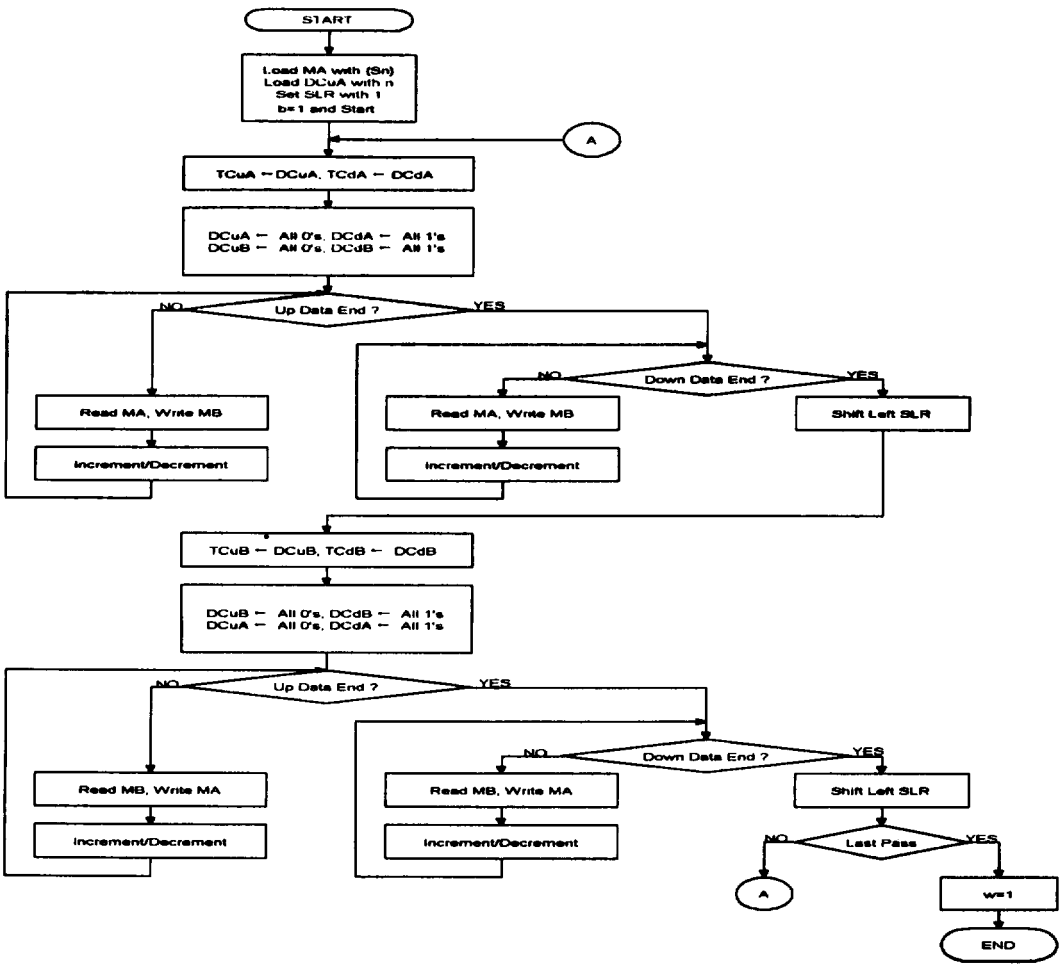
4.2 동작 과정

4.2.1 패스 0

MA에서 데이터를 읽어서 MB에 정렬, 저장하는 과정으로 각 레지스터들은 다음과 같이 초기화와 마이크로동작을 한다.

TCuA ← DCuA, TCdA ← DCdA
DCuA ← 0, DCdA ← 1, DCuB ← 0, DCdB ← 1
if (DCuA = TCuA)′ and SCS′
then read MA[DCuA], write MB[DCuB]
DCuA = DCuA + 1, DCuB = DCuB + 1
if (DCuA = TCuA)′ and SCS
then read MA[DCuA], write MB[DCdB]
DCuA = DCuA + 1, DCdB = DCdB-1
if (DCdA = TCdA)′ and SCS′
then read MA[DCdA], write MB[DCuB]
DCdA = DCdA-1, DCuB = DCuB + 1
if (DCdA = TCdA)′ and SCS
then read MA[DCdA], write MB[DCdB]
DCdA = DCdA-1, DCdB = DCdB-1
if (DCuA = TCuA) then shr SLR

DCuA 내용을 주소로 MA에서 데이터를 읽고 정렬할 비트는 최하위 비트가 된다. 이 비트가 0이면 DCuB의 값을 주소로 하여 MB에 저장하고, 1이면 DCdB의 값을 주소로 하여 MB에 저장한다. 이때 DCuB 주소에 저장하였으면 DCuB ← DCuB + 1, DCdB에 저장하였으면 DCdB ← DCdB − 11을 수행한다. 다음 정렬할 데이터의 주소(DCuA)를 1 증가한



(그림 4) 역방향 레딕스 하드웨어 정렬기의 전체 흐름도
(Fig. 4) Flowchart of Reverse Radix Sort

다. 정렬할 데이터의 끝을 판단하기 위하여 DCuA = TCuA 확인한다. 두 값이 같지 않으면 다음 번지의 데이터를 읽어 정렬을 반복하고, 같으면 DCdA 내용을 주소로 하여 MA에서 데이터를 읽어 위의 과정과 같이 MB에 데이터를 정렬한다. DCdA = TCdA이면 한 패스가 완료된다. 패스 0의 경우는 DCdA와 TCdA가 같기 때문에 SLR을 한비트 왼쪽으로 이동시키고, 패스 1 동작을 한다.

4.2.2 패스 1

MB에서 데이터를 읽어서 MA에 정렬, 저장하는 과정으로 각 레지스터들은 다음과 같이 초기화와 마이크로동작을 한다.

$TCuA \leftarrow DCuB,\ TCdA \leftarrow DCdB$
$DCuB \leftarrow 0,\ DCdB \leftarrow 1,\ DCuA \leftarrow 0,\ DCdA \leftarrow 1$
$if\ (DCuB = TCuB)'\ and\ SCS'$
$\quad then\ read\ MB[DCuB],\ write\ MA[DCuA]$
$\quad\quad DCuB = DCuB + 1,\ DCuA = DCuA + 1$
$if\ (DCuB = TCuB)'\ and\ SCS$
$\quad\quad then\ read\ MB[DCuB],\ write\ MA[DCdA]$
$\quad\quad\quad DCuB = DCuB + 1,\ DCdA = DCdA\text{-}1$

if (DCdB=TCdB)' and SCS'
then read MB[DCdB], write MA[DCuA]
DCdB=DCdB-1, DCuA=DCuA+1
if (DCdB=TCdB)' and SCS
then read MB[DCdB], write MA[DCdA]
DCdB=DCdB-1, DCdA=DCdA-1
if (DCuB=TCuB) then shr SLR

DCuB 내용을 주소로 MB의 데이터를 읽고 정렬할 비트는 1번 비트이다. 이 비트가 0이면 MA의 DCuA 주소에, 1이면 MA의 DCdA 주소에 저장한다. 이 과정은 DCuB=TCuB일 때까지 반복된 후 DCdB 주소의 데이터 정렬을 시작한다. 이후 DCdB=TCdB일 때 패스 1을 끝내고 SLR 내용을 1비트 왼쪽으로 이동시킨다.

4.2.3 패스 n

n이 짝수일 때 위 패스 0과 동일하고, 홀수일 때 1과 동일하다. SLR 비트가 n-1 패스일 때 종료한다.

$if\ SLR_{n-1}=1\ then\ w=1$

정렬 완료 신호(w)가 1로 설정되며, 주 컴퓨터로 최종 결과 데이터가 전송된다.

5. 하드웨어 설계 및 구현

제어회로에서 발생되는 모든 상태들의 변화 시각은 클럭의 부논리에서 작동하며, 모든 레지스터의 동작 클럭은 칩의 특성표에 준한다.

5.1 제어 회로 설계

제어 장치는 선택 변수를 제어하고 레지스터들의 입력을 인에이블시키는 2진 변수를 만들어 낸다. 제어 장치의 출력은 시스템 내에서 데이터 프로세서를 선택하고 인에이블시킬 뿐만 아니라 제어 장치 자체의 다음 상태를 결정하기도 한다.

정렬기의 제어 회로의 설계는 순차 레지스터와 디코우더 방법(sequence register and decoder method)을 이용하여 회로를 구현한다. 회로 구현을 위한 상태도를 그리고, 상태에 따른 마이크로 작동을 기술한다.

5.1.1 상태도(state diagram)

(그림 5)의 상태도는 21개의 상태와 7개의 외부 입력 조건을 갖고, 마이크로 작동은 〈표 1〉과 같다. 제어 논리부는 순차 레지스터와 디코우더 방법을 이용하므로 5개의 플립플롭(21상태〈2^5)과 2개의 디코우더(4 x 16)가 필요하다. 5개의 플립플롭의 입력항들은 여기표와 상태도에서 쉽게 구할 수 있다.

여기표에서 플립플롭 D_5의 입력은 디코우터의 현재 출력이 S15, S16, S18, S19·w일 경우만 1이 되고 다음과 같이 부울 대수의 형태로 표시할 수 있다.

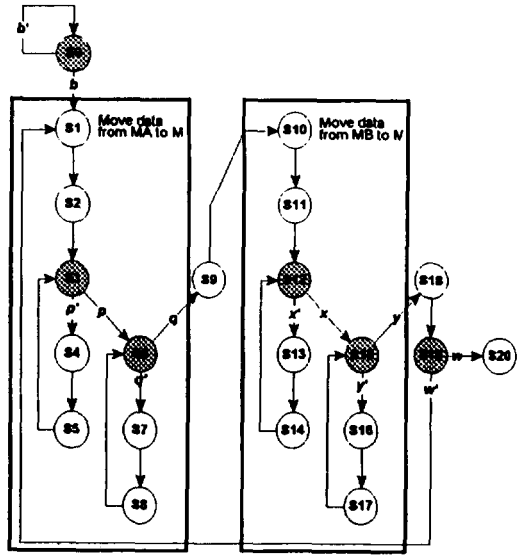
$$D_5=S15+S16+S18+S19\cdot w$$

같은 방법으로 나머지 D플립플롭의 입력 함수를 구할 수 있다.

$$D_1=S0\cdot b+S2+S4+S6+S10+S12+S16+S17+S18+S19\cdot \bar{w}$$

$$D_2=S1+S2+S3\cdot p+S5+S6\cdot \bar{q}+S8+S10+S12\cdot x+S13+S15\cdot y+S17+S18$$

$$D_3=S3+S4+S6\cdot \bar{q}+S8+S11+S12+S13+S14+S17+S19\cdot w$$



(그림 5) 상태도
(Fig. 5) State diagram.

〈표 1〉 순차 마이크로 작동
〈Table 1〉 Sequence of microoperation

상 태	Microoperation
S0	초기화
S1	TCuA ← DCuA, TCdA ← DCdA
S2	DCuA = 0, DCdA = 1, DCuB = 0, DCdB = 1
S3	Nothing
S4	Read MA, Write MB
S5	Increment/Decrement address
S6	Nothing
S7	Read MA, Write MB
S8	Increment/Decrement address
S9	Shift Left SLR
S10	TCuB ← DCuB, TCdB ← DCdB
S11	DCuA = 0, DCdA = 1, DCuB = 0, DCdB = 1
S12	Nothing
S13	Read MB, Write MA
S14	Increment/Decrement address
S15	Nothing
S16	Read MB, Write MA
S17	Increment/Decrement address
S18	Shift Left SLR
S19	Nothing
S20	END

$$D_4 = S6 \cdot q + S7 + S9 + S10 + S11 + S12 + S13 + S14 + S17$$

디코우터의 출력은 회로의 다음 상태를 결정 짓고 제어기의 출력 S0…S20은 하드웨어 정렬기의 Address 회로와 Memory회로를 제어하는 신호가 된다.

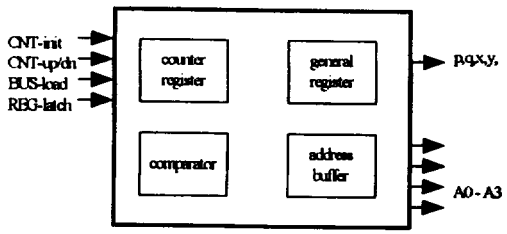
5.2 Address 회로 설계

하드웨어 정렬기의 주소부는 MA의 상위주소, 하위주소와 MB의 상위주소, 하위주소 총 4개의 주소부로 나누어진다. 주소부의 입출력과 레지스터 형태는 (그림 6)과 같다. 최초의 데이터가 들어 와서 회로 전체를 초기화하는 동작과 각 패스가 시작될 때 부분적인 레지스터의 초기화 동작을 수행한다. 또한 SCS의 상태에 따라 주소를 증가시키고 감소시키는 동작과 주소를 메모리로 보내는 마이크로 동작을 한다. 제어기의 상태에 따라 각 레지스터의 카운트 증가단자와 감소단자, 메모리로 주소를 보내는 마이크로 작동은 다음과 같이 상태 입력 값에 의해 동작한다.

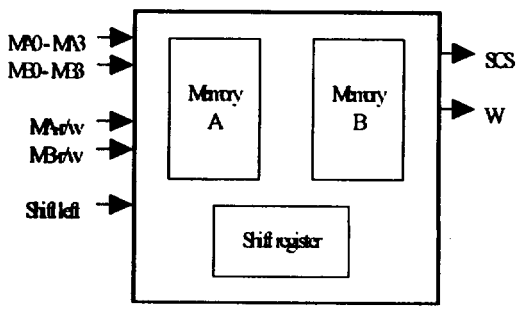
$$Initial = \overline{S0} \cdot \overline{S2} \cdot \overline{S11}$$
$$AU-CNT = \overline{SCS} \cdot (\overline{S13} + \overline{S17}) + \overline{S5}$$
$$AD-CNT = \overline{SCS} \cdot (\overline{S14} + \overline{S17}) + \overline{S8}$$
$$BU-CNT = \overline{SCS} \cdot (\overline{S5} + \overline{S8}) + \overline{S14}$$
$$BD-CNT = \overline{SCS} \cdot (\overline{S5} + \overline{S8}) + \overline{S17}$$
$$\overline{AU-BUS} = \overline{SCS} \cdot (\overline{S13} + \overline{S16}) + \overline{S4}$$
$$\overline{AD-BUS} = \overline{SCS} \cdot (\overline{S13} + \overline{S16}) + \overline{S7}$$
$$\overline{BU-BUS} = \overline{SCS} \cdot (\overline{S4} + \overline{S7}) + \overline{S13}$$
$$\overline{BD-BUS} = \overline{SCS} \cdot (\overline{S4} + \overline{S7}) + \overline{S16}$$

5.3 메모리 회로 설계

메모리 회로는 MA, MB의 두개의 메모리, 쉬프트 레지스터, 현재 정렬 비트가 1인지 0인지를 판단하는 조합회로로 구성된다. (그림 7)의 메모리 회로부는 메모리의 데이터를 읽고, 저장하는 동작을 한다. 패스가 끝나면 다음 패스로의 진행을 위해 쉬프트 레지스터를 왼쪽으로 1비트 이동시키며, 현재 정렬 중인 데이터 비트가 1인지 0인지를 알리는 신호 SCS 발생한다. 모든 패스가 종료되면 w신호를 설정한다.



(그림 6) 어드레스부의 레지스터 형태
(Fig. 6) Register configuration for address circuit



(그림 7) 메모리부의 레지스터 형태
(Fig. 7) Register configuration for memory circuit

메모리 회로의 정확한 마이크로 작동은 상태에 따라 다음과 같은 상태 입력 값에 의해 결정된다.

$$\overline{MB-cs}=\overline{S4}+\overline{S7}+\overline{S13}+\overline{S16}$$
$$\overline{MA-we}=\overline{S4}+\overline{S7}$$
$$SLR=\overline{S9}+\overline{S18}$$
$$\overline{MA-cs}=\overline{S4}+\overline{S7}+\overline{S13}+\overline{S16}$$
$$\overline{MA-we}=\overline{S13}+\overline{S16}$$

6. 시뮬레이션 및 하드웨어 검증

시뮬레이션은 설계된 역방향 레딕스 정렬의 소프트웨어적 방법과 하드웨어 방법으로 기존의 레딕스 교환 정렬과 비교하였다.

구현된 하드웨어 정렬기는 정렬할 수 있는 데이터 수를 제한하였기 때문에 수행 시간을 클럭 단위로 측정하고, 메인 클럭을 조정하여 정확히 동작할 수 있는 시간을 체크한다. 검출된 자료는 다음과 같다.

1. 전체 시스템 초기화 :1 clock
2. 각 패스의 초기화 :3 clock
3. 데이터 이동/정렬 :3 clock
4. 종료/반복 :1 clock

정렬에 소요되는 클럭의 전체 수는 (n+1)3b+2이다. n은 총 데이터 개수이고 b는 데이터의 비트수가 된다. 하드웨어 정렬기에서 메인 클럭은 10M이다.

하드웨어 구현 방식과 동일한 조건으로 동작하는 소프트웨어 방식은 알고리즘 1을 C언어로 구현하여 실행했다. 데이터는 random으로 발생한 8 비트 정수를 대상으로 실행했다.

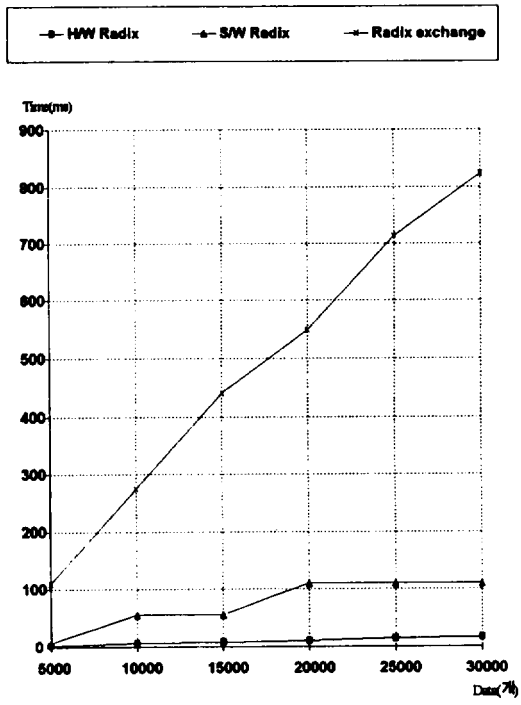
시뮬레이션 결과는 〈표 2〉과 (그림 8)이다. 이 결과 순수 하드웨어 정렬이 소프트웨어 정렬보다 10배 이상의 수행 속도를 나타냄을 알 수 있다. 하드웨어 정렬기는 데이터의 수가 증가하면 그에 비례적으로 처리 시간이 증가된다. 〈표 2〉에서 처럼 10000개의 데이터 정렬에서 소프트웨어 방법은 54.9ms가 소모되고, 하드웨어 방법은 5.3ms 시간이 소모되었다.

중앙처리장치의 제어 및 연산 속도는 프로세서마다 차이가 나지만, 하드웨어 정렬기는 자체 메인 클럭을 이용하기 때문에 처리 속도의 변화는 없다.

〈표 1〉 정렬 알고리즘의 데이터 정렬 수행 시간
〈Table 1〉 Execution time of sort algorithms

데이터수 / 방법	5000	10000	15000	20000	25000	30000
역방향 레딕스 (H/W)	2.65	5.3	7.95	10.6	13.2	15.9
역방향 레딕스 (S/W)	5	54.9	54.9	109.9	109.9	109.9
Radix exchange	109.9	247.7	439.6	549.5	714.3	824,.2

(단위 : ms)



(그림 8) 데이터 정렬 수행 시간
(Fig. 8) Sorting performance.

7. 결 론

본 역방향 레딕스 방식은 레딕스 교환 정렬의 2회 탐색 과정을 1회의 탐색으로 정렬, 저장이 가능하도록 하였다. 데이터 수가 메모리 크기 이내의 범위일 경우 현재 사용되고 있는 범용 소프트웨어 정렬 속도보다 10분의 1 이상으로 고속 처리됨을 확인하였다.

메모리 범위를 초과하는 수의 데이터일 경우 부분 정렬후 합병이 가능하다. 이의 연구는 앞으로의 과제이다.

본 하드웨어 정렬기는 컴퓨터 본체와는 별도로 장치하여야 하는 단점이 있고, 메모리 용량보다 큰 레코드들의 정렬 작업에 대한 연구도 계속되어야 한다.

참 고 문 헌

[1] Aki, "The Design and Analysis of Parallel Algorithms", Prentice Hall, pp. 85-107.

[2] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley, 1974.

[3] Clark D. Thompson, "The VLSI Complexity of Sorting", IEEE Transaction on Computers, VOL C-32, NO 12, December 1983, pp. 1171-1182.

[4] Clyde P. Kruskal, "Searching, Merging and Sorting in Parallel Comptation", IEEE Transaction on Computers, October 1983, pp. 942-946.

[5] Edgene E. Lindstrom & Jeffrey Scott Vitter, "The Design and Analysis of Bucketsort for Bubble Memory Secondary Storage", IEEE Transaction on Computers, March 1985, pp. 218-233.

[6] Isaac D. Scherson & Sandeep Sen, "Parallel Sorting in Two-Dimensional VLSI Models of Computation", IEEE Transaction on Computers, February 1989.

[7] J. Ja'Ja' and M. Owens, "VLSI Sorting with reduced hardware", IEEE Transaction on Computers, VOL C-33, NO 7, July 1984.

[8] Leon E. Winslow, Yuan-Chieh Chow, "The Analysis and Design of Some New Sorting Machines", IEEE Transaction on Computers, July 1983, pp. 677-683.

[9] M. Kitsuregawa, W. Yang, T. Suzuki, M. Takagi, "Design and Implementation of High-Speed Pipeline Merge Sorter with Run Length Tuning Mechanism", In Proc. of IWDM-87, pp. 89-102.

[10] M. Kitsuregawa et al. "GERO:A Commerical Database Processor Based on A Pipelined Hardware Sorter", SIGMOD RECORD VOL 22, NO 2, June 1993.

[11] Michael J. Quinn, "Parallel Computing Theory and Practice", Mcgraw-Hill, 1994.

[12] Michel Dubois, Christoph Scheurichm, Faye A. Briggs, "Synchronization, Coherence and Event Ordering in Multiprocessor", IEEE Computer, Vol. 21, No. 2, Feb. 1988, pp. 9-21.

[13] S. Devadas and K. Keutzer, "Synthesis of Delay-Fault-Testable Circuit:Theory", IEEE Transaction on Computer-Aided Design, Vol. 11, no. 1, January, 1992, pp. 87-101.

[14] Sedgewick, "Algorithms", Addison Wesley, 1983, pp. 117-123

박 희 순

1972년 연세대학교 전기공학과(학사)
1973년 (주)성남전자 전기제어
1977년 (주)현대양행 콘트롤시스템부
1983년 전남대학교 대학원 전기공학과(석사)
1986년 전북대학교 대학원 전기공학과(박사)
1983년~현재 원광대학교 공과대학 컴퓨터공학과 교수
관심분야:컴퓨터구조, 병렬처리, 컴퓨터 제어

전 종 연

1994년 원광대학교 공과대학 컴퓨터공학과(학사)
1996년 원광대학교 대학원 컴퓨터공학과(석사)
관심분야:컴퓨터구조, 병렬처리, VLSI설계

김 희 숙

1991년 원광대학교 공과대학 전자계산공학과(학사)
1995년 원광대학교 교육대학원 전자계산교육전공(석사)
1996년 원광대학교 컴퓨터공학과 박사과정
관심분야:컴퓨터구조, 병렬처리, 병렬알고리즘