확장형 VLSI 리바운드 정렬기의 설계

윤 지 헌† 안 병 철††

요 약

시간 복잡도가 O(*N*)인 고집적 회로(VLSI)의 병렬 정렬기 설계에 관한 논문이다. 발표된 빠른 VLSI 정렬 알고리즘은 *N*개의 데이타를 정렬하기 위해 O(log *N*) 시간 복잡도를 가지고 있다. 그러나 이러한 알고리즘은 입출력 시간을 고려하지 않고, 복잡한 네트워크 구조를 가지므로 확장이나 실용화하기 힘들다. 입출력 시간이 포함된 병렬 정렬 알고리즘들의 칩면적과 시간 복잡도를 분석한 후 가장 효과적인 rebound sort이론을 확장하여 VLSI로 구현한다. 이 리바운드 정렬기는 파이프라인으로 구성하여 O(*N*)의 시간 복잡도를 가지며 한 개의 칩에 8개의 16비트 레코드를 정렬할 수 있다. 그리고 이 정렬 칩은 확장성을 가지고 있어 수직으로 연결할 경우 8개 이상의 레코드를 정렬할 수 있다.

Design of an Expandable VLSI Rebound Sorter

Ji Heon Yun† and Byoung Chul Ahn††

ABSTRACT

This paper presents an improved VLSI implementation of a parallel sorter to achieve O(*N*) time complexity. Many fast VLSI sort algorithms have been proposed for sorting N elements in O(log *N*) time. However, most such algorithms proposed have complex network structure without considering data input and output time. They are also very difficult to expand or to use in real applications. After analyzing the chip area and time complexity of several parallel sort algorithms with overlapping data input and output time, the most effective algorithm, the rebound sort algorithm, is implemented in VLSI with some improvements. To achieve O(*N*) time complexity, an improved rebound sorter is able to sort 8 16-bits records on a chip. And it is possible to sort more than 8 records by connecting chips in a chain vertically.

1. 서 론

상당히 오래 전부터 정렬에 대한 이론과 실제 자료 처리를 위해 많은 연구가 있었다. 1973년 Knuth[1]는 대부분의 전자 계산 시스템 상에서 전체 실행시간의 약 25% 정도가 정렬에 소비된다고 추산했다. 정렬은 사용하는 알고리즘에 따라 많은 시간 차이가 있으며 정렬에 관한 많은 종류의 알고리즘이 있다. 이들 정렬 알고리즘의 장단점과 하드웨어의 구현, 외부 병합에 대한 개념은 1984년에 Bitton[3]에 의해 정리되었다. 하나의 프로세서를 사용한 내부 순차 정렬 알고리즘에 대한 처리 시간의 하한은 *N*개의 레코드를 정렬하기 위해 O(*N* log *N*)이나 대부분의 병렬 정렬 알고리즘은 *N*개의 프로세서를 사용한다고 가정할 경우 O(*N*)과 O(log *N*)사이의 시간 복잡도를 가지고 있다.

1985년에 Akl[2]이 대부분의 병렬 정렬 이론을 소개하였으며 1980년부터 병렬 정렬 알고리즘에 대한 실제 하드웨어 구현에 관한 연구가 시작되었다. 이들 알고리즘 중 몇 가지는 병렬 컴퓨터로 구현이 되었고 정렬 알고리즘의 구현 가능성이 입증되었다[7, 8, 9]. 병렬 정렬 알고리즘은 1) 네트워크 정렬 알고리즘, 2) 공유 메모리 정렬 알고리즘, 3) 병렬 화일 정렬 알고리즘 및 4) 병합 알고리즘의 4가지 범주로 분류할 수 있다. 대부분의 이들 알고리즘은 많은 수의 프로세서를 필요로 하며, 정렬될 레코드 수가 *N*개 일 때 프로세서의 수는 *N*개 혹은 최소 log*N* 개에

† 정 회 원 : 영남전문대학 전자계산기과 교수
†† 정 회 원 : 영남대학교 전산공학과 조교수
논문접수 : 1995년 2월 6일, 심사완료 : 1995년 4월 21일

비례하여야 한다. N개의 범용 프로세서를 사용할 경우 정렬의 성능과 가격은 성능 대 가격비가 아주 낮으나 hypercube 구조의 컴퓨터나 massively parallel 컴퓨터를 이용할 경우 성능 대 가격비를 개선할 수 있다. 이 경우는 정렬에 필요한 데이타의 분배와 정렬된 데이타의 병합을 위한 시간이 필요하다. 이것을 효과적으로 구현하는 것이 많은 개수의 병렬 정렬기를 VLSI 칩으로 구현하는 것인데 대부분의 VLSI 구현 가능한 이론은 네트워크를 사용하여 O(log N)정렬 하는 것으로 $O(N^2)$의 면적 복잡성을 가지고 있으며 cube-connected cycles이나 tree-of-combiners와 같은 망구조로 구현하기가 상당히 복잡하다[4, 5, 6].

본 논문에서는 몇 가지 알고리즘에 대한 분석을 VLSI 구현시 칩의 면적과 시간 복잡도에 대해 분석하며 가장 성능이 우수한 Chen의 리바운드 정렬(rebound sort) 알고리즘의 설계에 대해 기술하고자 한다.

2. 병렬 VLSI 정렬기

자기 버블 기억소자(magnetic bubble memory)를 사용한 정렬 알고리즘 이론이 1980년 Chung [11], 1981년 Lee[12]에 의해 각각 제안된 후, 1982년 Carey[10], 1982년 Yasuura[14], 1983년 Miranker[15], 1985년 Rajgopal[13]과, Leighton과 Bilardi, 1993년 Alnuweiri[6]등에 의해 개발 및 구현되었다. 그 중에서 Carey는 실리콘 상에 스택을 기초로한 마이크로구조를 사용하여 버블 정렬 알고리즘을 병렬처리 하도록 구현하였다. 이 정렬기는 O(N)의 시간 복잡도와 O(N)의 칩면적을 사용하며 8비트 정수 key와 8비트 레코드 포인터로 구성된 16비트 레코드를 단일 스택으로 push하여 정렬한다. N개의 데이타를 정렬하는 전체 소요 시간은 $2N$ 사이클이며, 레코드가 스택으로부터 pop될 때 오름차순으로 출력된다.

전원과 I/O 패드부를 제외한 핵심부의 칩면적은 7000㎛ x 7000㎛(280mils x 280mils)이다. 각 칩은 출력으로 정렬된 32개의 16비트 레코드를 생성할 수 있는 능력을 가지며, 최대 처리 속도는 0.84MHz이며, 칩을 cascade로 접속하면 데이타베이스 기계를 위한 디스크 블록 버퍼로도 사용할 수 있다.

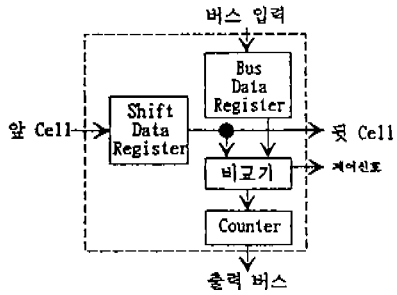
Rajgopal은 Yasuura의 병렬계수 정렬 알고리즘을 구현하였으며 시간 복잡도는 Carey의 정렬기와 같이 N개의 데이타를 정렬하기 위해 $2N$ 사이클이 필요하며, 한 개의 칩은 8개의 16비트 key를 정렬할 수 있으며, 정렬 동작 후 정렬된 key 값의 인덱스가 되돌려 진다. 각 칩은 8개의 key로 제한되어 있으나 32개의 칩이 직접 cascade로 접속되면 256개의 key들을 정렬할 수 있다. 칩의 전체 면적은 4㎛ NMOS기술로 9200㎛ x 7900㎛ (368mils x 316mils)이며 이 칩의 처리 속도에 대한 정보는 없다.

위의 칩에서 정렬될 레코드들은 디스크나 테이프 같은 보조기억장치에 기억되어 있으며 정렬이 수행될 때 기억장치에 적재되고, 레코드가 정렬된 후 다시 보조기억장치에 되돌려진다. 실제적인 정렬과정은 데이타 입력, 정렬, 데이타 출력의 3단계로 분리할 수 있다. 입출력 정렬시간을 고려한 O(N) 시간 복잡도의 알고리즘은 N개의 레코드를 정렬하기 위해 N개의 처리기를 사용하고, 데이타는 순차적인 방법으로 정렬기에 공급한다. 대부분의 병렬 정렬 알고리즘은 입출력 시간을 무시하며 프로세서에 의해 처리되는 정렬 시간만의 시간 복잡도를 계산하고 있다. 그러나 이 입출력 시간은 무시될 수 없으며, 입출력 시간과 실제 정렬되는 시간이 중첩되는 병렬 정렬 알고리즘으로 1) Yasuura[14]가 제안한 계수형 정렬기, 2) Miranker[15]가 제안한 VLSI 정렬기, 3) Chen[16]이 제안한 리바운드 정렬기, 4) Todd[17]가 제안한 파이프라인 병합 정렬, 5) Takagi[18]가 제안한 하드웨어 정렬 병합 시스템이 있다. 이들 알고리즘 중에서 정렬에만 관련된 계수형 정렬기, VLSI 정렬기와 리바운드 정렬기에 대한 알고리즘을 분석한다.

2.1 계수형 정렬기

계수형 정렬은 순서처리와 배열처리의 두 과정으로 분리할 수 있다. 순서처리는 key의 집합 $K = \{k_1, k_2, \cdots\cdots, k_n\}$ 내에서 각 key의 크기를 비교하여 순서를 결정하는 과정이다. N개의 정렬 셀 프로세서 중에서 j번째 정렬 셀을 J라 할 때,

k_j는 J셀이 가지고 있는 key이다. 순서 c_j는 J셀의 값 k_j와 K내의 모든 key k_i들을 비교하여 그 크기의 결과를 계수하여 얻어진다. 계수형 정렬기의 기본 셀은 (그림 1)과 같다.



(그림 1) 계수형 정렬기의 기본셀
(Fig. 1) Basic Cell of Enumeration Sorter

배열 처리는 얻어진 크기의 순서를 가지고 호스트 컴퓨터나 특수 기구로된 메모리 장치를 통해 재배열 처리하는 과정이다. 이것을 알고리즘으로 표현하면 아래와 같다.

```
begin
for j = 1 to N do in parallel
   cj = 0
   for i = 1 to N do
      Input ki
      if kj ≥ ki and ki not eof then
         cj = cj + 1
      end if
      if i = j then
         save kj
      else
         shift ki to right cell
      end if
   end for

   for i = 1 to N do
      if kj ≥ ki and ki not eof then
         cj = cj + 1
      end if
      if i = j then
         Output cj
      else
         shift ki to right cell
      end if
   end for
end for
end
```

위의 알고리즘은 2개 이상의 key들이 같은 기억 장소에 기억되지 않으며 $k_i = k_j$ 이고 $i < j$ 이면 $c_i < c_j$ 임이 보장된다. N개의 정렬 처리 소자들이 위의 알고리즘과 같이 병렬로 순서 계산을 한다. 모든 key는 메모리 장치와 순서처리를 수행하는 정렬회로간에 순차적으로 전송된다. 입출력 시간 복잡도는 정렬 처리 시간과 완전히 중첩된다. 위의 알고리즘에서 프로세서의 수 $p(N)$ 은 N이고 요구되는 전체 시간은 $O(N^2)$ 이다.

N개의 프로세서에 대한 시간 복잡도와 전체 가격은 다음과 같다.

$$p(N) = N, \quad (1)$$
$$t(N) = O(N^2)/O(N) = O(N), \quad (2)$$
$$c(N) = p(N) \times t(N) = O(N^2) \quad (3)$$

순서처리는 $2N$ 사이클이 소요되고 순서처리 완료 후 호스트 컴퓨터의 재배열 처리가 필요하며 재배열은 $O(N)$ 시간 복잡도로 처리된다.

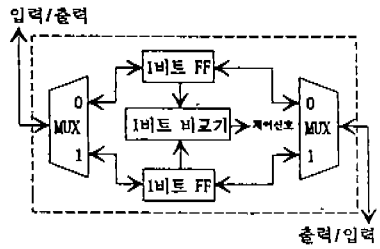
2.2 VLSI 정렬기

Miranker[15]에 의해 제안된 VLSI 정렬기는 Lee[12]에 의해 제안된 버블정렬기의 기본 동작과 유사하다. 그러나 셀의 마이크로구조는 상당한 차이가 있다. 이 정렬기는 (그림 2)와 같이 $\lceil N/2 \rceil$개의 셀 즉, 각각 1 비트 비교기와 메모리 소자의 배열이 선형 배열로 구성되어 있다. 예를 들어 n개의 기본셀(dibit cell)이 n비트 key 필드를 정렬하기 위해 서로 연결되어 있다. 각 셀은 정렬 중 적절한 데이타 이동을 위해 상하의 셀들과 연결되어 있다.

VLSI 정렬기는 데이타 흐름 방향을 수정하여 오름차순 혹은 내림차순으로 key들을 정렬할 수 있다. 각 정렬 단계는 다음 두 단계를 거친다.

(1) 비교단계 : 각 셀 내의 2개의 key가 서로 비교된다.

(2) 전송단계 : 비교의 결과에 따라 2개의 key 중 하나가 인접 셀로 전송되고, 원래의 셀은 다른 인접 셀로부터 item을 받아들인다.



(그림 2) VLSI 정렬기의 기본셀
(Fig. 2) Basic Cell of VLSI Sorter

VLSI 정렬기 셀은 입력단계와 출력단계를 포함하고 있다. 입력단계에서 각 셀 내에서 2개의 key중 큰 key값이 아래 셀로 전송되므로 가장 작은 최소 key는 가장 위의 셀에 위치하고 가장 큰 key는 가장 아래 셀에 위치한다. 일반적으로 i번째 최소 key는 top의 i셀들 중의 하나에 있어야 한다. 출력 단계에서 각 셀의 2개의 key중 작은 쪽이 상향 전송된다.

```
begin
for j = 1 to N/2 in parallel
   for i = 1 to N
      /*Input stage*/
      Input data
      /*larger data transferred*/
      if a_j ≤ b_j then
         a_{j+1} = b_j or b_{j+1} = b_j
         b_j = max(a_{j-1}, b_{j-1})
      else
         a_{j+1} = a_j or b_{j+1} = a_j
         a_j = max(a_{j-1}, b_{j-1})
      end if
   end for

   for i = N+1 to 2N
      /*smaller data transferred*/
      if a_j < b_j then
         b_{j-1} = a_j or a_{j-1} = a_j
         a_j = min(a_{j+1}, b_{j+1})
      else
         a_{j-1} = b_j or b_{j-1} = b_j
         b_j = min(a_{j+1}, b_{j+1})
      end if
      /*Output stage*/
      Output data
   end for
end for
end
```

위의 알고리즘에서 요구되는 시간은 $O(N^2)$이고 $N/2$개의 프로세서가 사용되므로 시간 복잡도는 $O(N)$이다.

$$p(N) = N/2 = O(N), \quad (4)$$
$$t(N) = O(N^2)/O(N) = O(N), \quad (5)$$
$$c(N) = p(N) \times t(N) = O(N^2) \quad (6)$$

이 알고리즘은 오름차순과 내림차순으로 정렬할 수 있다. N개의 key를 $N/2$개의 프로세서로 정렬하기 위해 $4N$ 사이클이 요구된다.

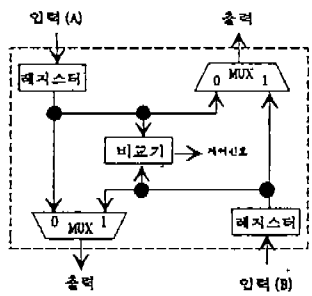
2.3 리바운드 정렬기

리바운드 정렬기는 루프(loop)당 1개의 key씩 N개의 key를 수용할 수 있는 N개 루프 자리 이동이 가능한 사다리형 정렬기(ladder sorter)[19]를 기초로한 정렬 방법이다[20]. 입력과 출력시간이 정렬시간과 완전히 중첩되며 리바운드 정렬기의 기본셀은 (그림 3)과 같이 기억소자와 데이타 조향장치로 구성된다. 기억 소자는 수평으로 2개의 key를 저장하며 key의 입출력은 최상위 셀로만 가능하며 중간층과 가장 밑의 셀은 데이타의 이동만 가능하게 한다. 그러므로 정렬될 key는 왼쪽 상위 기억소자로 입력되고 정렬된 데이타는 오른쪽 상위 기억 소자로 출력된다. N개의 key열을 정렬하기 위해 $4N$ 사이클이 필요하다.

리바운드 정렬 알고리즘은 $O(N^2)$ 시간을 필요로 하며 N개의 프로세서에 대한 시간 복잡도는 아래와 같다.

$$p(N) = N-1 = O(N), \quad (7)$$
$$t(N) = O(N^2)/O(N) = O(N), \quad (8)$$
$$c(N) = p(N) \times t(N) = O(N^2) \quad (9)$$

```
begin
for j = 1 to N in parallel
   for i = 0 to 2N-1 do
      Input data into L_j
         and Output data from U_j
      if mod(i,2) = 0 then
         /*Even processors start*/
         if L_j ≥ U_j then
            /*vertical data movement*/
            L_{j+1} = L_j
            U_{j-1} = U_j
         else if L_j < U_j then
            /*horizontal data movement*/
            L_{j+1} = U_j
            U_{j-1} = L_j
         end if
      else
         /*Odd processors start*/
         if L_j < U_j then
            /*L_j = U_j at cycle (i-1)*/
```



(그림 3) 리바운드 정렬기의 기본셀
(Fig. 3) Rebound Sorter Cell

```
                L_{j+1} = U_j
                U_{j-1} = L_j
            else
                /*L_j > U_j at cycle (i-1)*/
                L_{j+1} = L_j
                U_{j-1} = U_j
                or
                /*L_j < U_j at cycle (i-1)*/
                L_{j+1} = U_j
                U_{j-1} = L_j
            end if
        end if
    end for
end for
end
```

위의 알고리즘에서 비교한 결과에 따라 수평 혹은 수직으로 데이타를 이동한다. 각 비교기는 N개의 key를 정렬하기 위해 N회의 비교를 필요로 하나, 같은 데이타를 비교할 경우는 $2N$회의 비교를 필요로 한다.

2.4 병렬 VLSI 정렬기의 비교

앞의 세 가지 병렬 VLSI 정렬 알고리즘은 같은 시간 복잡도를 가지고 있으나 실제 처리시간은 다르다. Yasuura의 계수형 정렬기는 $2N$ 사이클이 소요되며 Miranker의 VLSI 정렬기와 Chen의 리바운드 정렬기는 $4N$ 사이클이 소요된다. 그러나 리바운드 정렬기는 파이프라인을 적용할 경우 $2N$ 사이클로 줄일 수 있으나 VLSI 정렬기는 프로세서를 반만 사용하므로 클럭 사이클 수를 줄일 수 없다. 〈표 1〉의 실제 구현시 최대 클럭 주파수는 리바운드 정렬기가 다른 정렬기보다 약 3배가 빠르다. 계수형 정렬기는 각 셀에 대한 카운터의 내부지연과 재배열 처리시간에 어려움이 있는데, 프로세서의 수가 증가되면 카운터의 지연시간이 증대되고 지연시간을 줄이기 위해서는 카운터 면적이 커져야 한다. Miranker[15]에 의해 제안된 VLSI 정렬기는 $N/2$개의 적은 수의 프로세서를 요구하기 때문에 VLSI 구현을 위한 적절한 알고리즘이나 VLSI 정렬기의 처리 속도는 리바운드 정렬기의 비교기의 속도의 1/2이다.

Berkeley의 Espresso와 Genesil Silicon Compiler를 사용하여 16비트 key를 정렬하기 위해 세 가지 알고리즘의 기본셀을 시뮬레이션을 통해 최대 처리 속도와 면적과의 관계를 조사해 본 결과를 〈표 1〉에 도시하였다. 이때 각 셀을 제어하는 제어기의 면적은 세 가지 알고리즘 모두다 같다고 가정했다. 〈표 1〉에서 셀의 면적 면에서는 VLSI 정렬기가 $N/2$의 셀이 필요하므로 가장 작은 면적이 요구된다. 그러나 이 면적은 리바운드 정렬기와 비교하면 큰 차이가 없다. 그 이유는 리바운드 정렬기는 한번에 1/2레코드만 입력하므로 다른 정렬기에 비해 레지스터, 비교기, 멀티플렉서의 크기가 반만 소요되기 때문이다. 적은 비트를 사용하는 것이 칩면적을 줄일 뿐만 아니라 지연 시간을 줄일 수 있으므로 리바운드 정렬기의 최대 클럭 주파수가 다른 정렬기에 비해 가장 빠른 이유이다. 그러므로 리바운드 정렬기는 처리 속도와 칩면적에 대해서 가장 효율적이다.

〈표 1〉 VLSI 정렬기의 비교
〈Table 1〉 VLSI Sorter Comparison

	정렬 기본셀의 기능적 분석	기본셀의 면적 (mils x mils)	N 셀의 전체면적 (mils x mils)	클럭 사이클 수	최대 클럭 주파수
계수형 정렬기	2개의 16비트 레지스터 1개의 16비트 비교기 1개의 8비트 카운터	2x 3.60x115.19 52.34x 11.81 16.67x 27.98	Nx1903.31	2N	6MHz
VLSI 정렬기	2개의 1비트 레지스터 1개의 1비트비교기 2개의 1비트 멀티플렉서	2x 3.60x 6.86 3.46x11.61 2x 3.96x 2.84	16xN/2x112.74	4N	6MHz
리바운드 정렬기	2개의 8비트 레지스터 1개의 8비트 비교기 2개의 8비트 멀티플렉서	2x 3.60x 62.75 30.89x 11.61 2x 3.96x 24.83	Nx1013.27	2N (파이프라인 사용)	17MHz

3. 확장형 리바운드 정렬기

Chen에 의해 제안된 본래의 리바운드 정렬기는 버블 기억 장치로 구현하기 위해 고안되었다. 레코드의 크기와 한번에 정렬할 수 있는 레코드 수를 사용자에 따라 확장할 수 있도록 하였으며 파이프라인 방법을 도입하여 N개의 레코드를 정렬하는데 $2N$ 사이클이 소요되도록 성능 면에서 2배로 개선시켰다.

리바운드 정렬기를 구현하기 위해 다음과 같은 가정을 하였다.

(1) 정렬 방법은 오름차순으로 한다. 값이 큰 레코드들은 입력 포트를 통해 정렬기의 상위 셀에서 하위 셀로 내려지며, 값이 작은 레코드들은 상위 셀로 올라가게 되며 마지막에는 출력 포트로 출력된다.

(2) 각 레코드의 크기는 16비트이며, 부가적인 하드웨어를 사용하면 가변의 레코드를 정렬할 수 있다.

(3) 정렬할 레코드 수가 정렬기의 용량을 초과

할 때 부가적인 정렬장치를 수직으로 연결하여 사용할 수 있다.

한 개의 정렬 기본셀은 (그림 3)과 같은 N개의 동일한 구조의 셀로 구성된다.

3.1 데이타 조향 장치

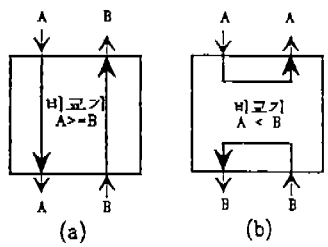
조향 장치는 한번에 한 레코드를 1/2 레코드씩 두번 비교하며 (그림 3)과 같이 1개의 비교기와 2개의 멀티플렉서로 구성된다. (그림 3)에서 각 셀의 상부 좌측으로부터 특정레코드(A)의 1/2레코드 데이타가 입력되며 동시에 하부 우측으로부터 다른 레코드(B)의 1/2레코드 데이타가 입력된다. 조향 장치는 입력된 1/2레코드 데이타들을 (그림 4)에서와 같이 상대적인 크기에 따라 수직 혹은 수평으로 셀을 통해 이동하게 한다. 즉, 레코드 A와 레코드 B가 비교기에 의해 비교된 후 레코드 A가 레코드 B 보다 클 경우는 (그림 4 (a))와 같이 레코드 A는 수직 아래로 레코드 B는 수직 위로 움직인다. 레코드 A가 레코드 B보다 작을 경우는 (그림 4(b))와 같이 레코드 A는 수평 위로 레코드 B는 수평 아래로 움직인다. 멀티플렉서는 비교기의 결과에 따라 레코드의 이동 방향을 변경시키기 위해 사용된다.

3.2 제어장치

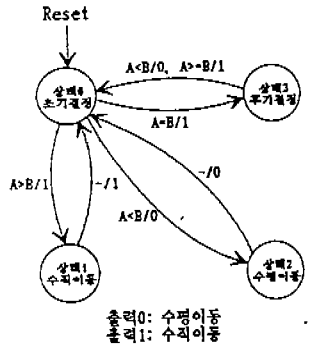
레코드를 비교한 후 레코드 조향 제어는 (그림 5)와 같은 상태도로 도시할 수 있다. 이 상태도는 비교기로부터 입력을 받아 다음 상태를 결정하고 비교 결과에 따른 레코드를 조향하기 위해 멀티플렉서의 선택 신호를 출력한다. (그림 5)에서 제어기는 4가지 상태를 가진다. 초기 결정 단계(상태-0)에서 비교할 2개의 레코드의 상위 1/2 레코드를 비교한다. 이때 데이타 이동은 (그림 4)의 데이타 조향 방법에 따른다. 상부 좌측 입력 레코드가 하부 우측 입력 레코드 보다 클 경우 데이타들은 수직으로 이동되며 상태 기계는 수직 이동 상태(상태-1)로 들어가며, 수직 이동 상태는 초기 결정 단계에서 행해진 비교 결과가 이미 결정되었으므로 나머지 레코드 크기에 대한 추가의 판단이 필요치 않으므로 초기 결정에 따라 수평으로 이동한다. 만약 상부 좌측에 입력 레코드가 하부 우측 입력 레코드 보다 작으면 상태 기계는 수평 이동 상태(상태-2)가 되며 레코드는 수평 이동하고 남은 레코드도 비교 없이 수평 이동한다. 그러나 상부 좌측 입력 레코드가 하부 우측 입력 레코드와 같으면 레코드의 비교가 하위 레코드들에 대한 비교가 더 필요하다. 남은 레코드를 비교하기 위해 후기 결정 상태(상태-3)가 필요하며 이 비교 결과에 따라 레코드는 수평 혹은 수직으로 데이타의 흐름 방향을 결정한다.

3.3 파이프라인 처리

리바운드 정렬기는 한번에 N개의 레코드를 유지하고 처리할 수 있는 능력을 가지고 있으며 출력부에 동일한 길이의 정렬된 레코드를 생성할 수 있다. 최대의 성능을 얻기 위해서는 입력과 출력이 동시에 수행되는 것이 필요하다. 이것은 이미 입력된 레코드들이 정렬되어 정렬기의



(그림 4) 데이타 조향 방향
(Fig. 4) Data Steering Direction



(그림 5) 정렬기 제어 상태도
(Fig. 5) State Diagram of Sorter Controller

출력으로 나타날 때 정렬기 내부에서 레코드를 계속해서 정렬할 수 있는 파이프라인 처리가 필요하다. 파이프라인 정렬기는 처음 N 사이클 동안 1/2레코드가 입력되는데 비교없이 하향으로 이동한다. 그 다음 N 사이클 동안 남은 1/2레코드가 비교되어 정렬된다.

$2N+1$ 사이클부터는 정렬된 데이타가 비교없이 출력된다. 그러므로 첫 번째 N 사이클과 세 번째 N 사이클은 레코드의 비교없이 데이타만 상위 혹은 하위로 움직이도록 한다. 이 새로운 이동시 레코드를 입력시킬 수 있으므로 파이프라인 처리가 가능하다.

이 파이프라인 처리는 상태 기계로 구현할 수 있으며 상태도는 (그림 6)에 도시하였으며 각각의 정렬기 셀 제어기에 필요한 제어신호를 제공하도록 한다.

4. VLSI 구현

이 장에서는 리바운드 정렬기의 VLSI 구현과 관련된 사항에 대해 기술한다. 칩면적과 floorplan 때문에 N을 8로 하였다.

4.1 확장성

정렬기 칩은 내부적으로 단지 8개 레코드만 정렬할 수 있으므로 큰 화일을 정렬하기 위해서는 여러 개의 정렬 칩을 사용하여야 한다. 리바운드 정렬기 칩들을 수직으로 연결하여 사용자가 응용에 따라 확장할 수 있도록 하기 위해 다음의 사항을 고려하였다.

(1) 파이프라인 제어신호들은 체인 내의 여러 칩을 연결하였을 경우에도 동작해야 한다.
(2) 연결된 모든 칩들은 체인 내에서의 그들의 위치를 알고 있어야 한다. 즉 연결의 제일 위에 있는 칩은 적절한 I/O동작을 수행해야 하고, 제일 밑에 있는 칩은 하부 좌측 출력을 하부 우측 입력으로 연결할 수 있어야 한다.

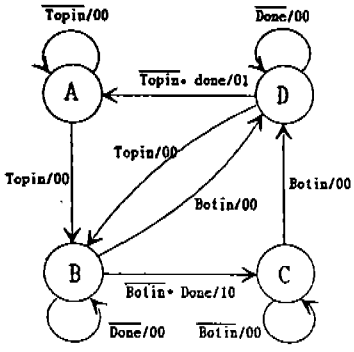
확장 능력을 가진 완전한 VLSI 리바운드 정렬기 칩의 블록도를 (그림 7)에 나타내었다.

이 확장 능력을 위해 요구되는 제어신호들을 아래에 기술한다.

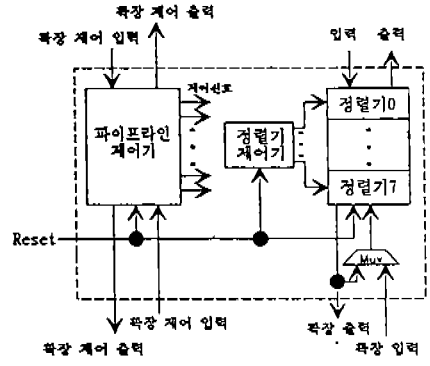
(1) Reset 신호 : 이 신호는 파이프라인 제어 유한 상태 기계를 포함한 칩상의 모든 제어기를 동시에 초기화한다.
(2) 확장 입출력 신호 : 이 신호는 정렬 칩의 하부 처리소자 혹은 셀에 인가된 데이타의 방향을 결정하기 위해 사용된다. 칩에 대한 데이타 입력은 다른 칩으로부터 또는 이 칩의 하부 좌측 입력 포트로 부터 입력된다.
(3) 확장 제어 신호 : 이들 신호는 체인내의 칩들에 대한 파이프라인 패턴 생성을 위해 사용된다.

4.2 논리 설계

리바운드 정렬기는 (그림 7)과 같이 기본적인



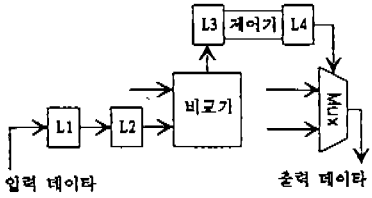
(그림 6) 파이프라인 제어 상태도
(Fig. 6) State Diagram of Pipeline Controller



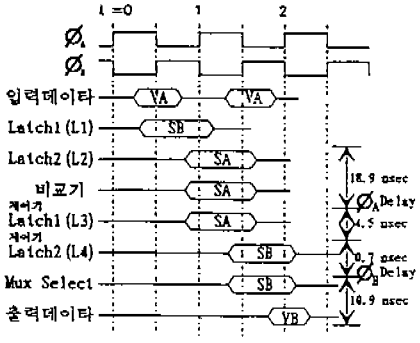
(그림 7) 확장형 리바운드 칩의 블록도
(Fig. 7) Block Diagram of a Expandable Rebound Chip

블록들로 구성되어 있다. 각 블록은 먼저 상위 레벨의 시뮬레이터로 모의 실험을 한 후 설계를 시작하였으며 제어기는 Berkeley의 Espresso를 이용하여 PLA를 구현하였으며 나머지 각 블록 및 칩 설계는 Genesil Silicon Compiler를 사용하여 구현하였다. 정렬기 내에서 레코드를 저장하기 위해 래치를 사용하였다. 정렬기 셀 제어기와 파이프라인 제어기, 비교기, 래치는 칩의 코아를 구성하며 코아 주위에 전원, 접지, 클럭, 입력과 출력 패드를 배열하였다.

설계에서 사용된 각 회로의 일관성 검사를 위해 타이밍 분석을 하였다. 2상의 비중첩(non-overlapping) 클럭 방법을 사용하여 논리적으로 연관된 클럭이 인가되는 모든 회로에 대한 타이밍 기준을 검사하였다. 이 검증 도구를 사용하여 모든 타이밍 경로를 통한 최악의 경우에 대한 신호 전파 지연과 입력 설정 유지 시간을 계산하였다. 그리고 시스템의 최대 클럭 주파수를 결정하였는데 확장형 리바운드 정렬기는 17MB/s의 성능을 나타내었다. 리바운드 정렬기의 임계 경로와 중요한 블록의 지연 시간을 (그림 8)에 도시하였다.



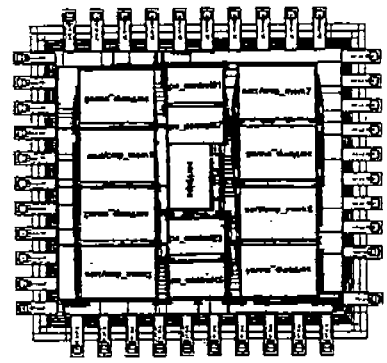
(a) Critical Path



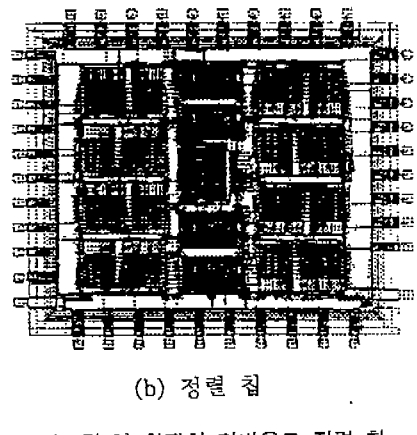
(b) Timing Analysis of Critical Path

(그림 8) 타이밍 분석
(Fig. 8) Timing Analysis

Floorplan은 칩 상에 설계 대상의 배치, routing 채널의 할당과 외부와의 적절한 연결로 구성된다. Floorplan은 placement, fusion, pinout의 3단계로 나누어 설계하였으며 (그림 9(a))에 도시하였다. 리바운드 정렬기는 8개의 기본셀, 1개의 제어기 및 1개의 파이프라인 제어기로 구성되며, 칩면적과 전파 지연을 줄이고 배선의 복잡도를 최소화하기 위해 이 10개의 모듈들을 인접 배치하였다. 최종 배치를 위해 여러 번의 시행 착오를 시도하였으며, 완료 후 코아의 크기는 159mils x 146mils이다. 입력과 출력 패드를 포함한 전체 칩 크기는 2㎛ CMOS 기술로 5,801㎛ x 5,283㎛(228.4mils x 208.0mils)이며, 이 칩은 48핀 핀-그리드 패키지를 사용하였으며 (그림 9(b))에 도시하였다.



(a) Floorplan and Routing



(b) 정렬 칩

(그림 9) 확장형 리바운드 정렬 칩
(Fig. 9) Expandable Rebound Sorting Chip

5. 결 론

입출력 시간이 정렬하는 시간과 중첩되는 세 가지 병렬 VLSI 정렬 알고리즘을 분석하였다. 정렬 시간은 $2N$과 $4N$의 범위이지만 같은 $O(N)$ 시간 복잡도를 가진다. Yasuura의 계수형 정렬기는 각 셀에 대한 카운터의 내부지연과 재배열 처리 시간 때문에 좋은 처리 시간을 얻을 수 없다. 즉, 프로세서의 수가 증가되면 카운터의 비트 수가 많아지고 이로 인한 지연시간이 증대되므로 지연 시간을 줄이기 위해서는 카운터 면적이 커져야 한다. Miranker VLSI 정렬기는 $N/2$의 적은 수의 프로세서를 요구하므로 VLSI 구현을 위한 적절한 알고리즘이다. 그러나 처리 속도가 리바운드 정렬기에 비해 2배정도 느린 것이 단점으로 나타났다. 그리고 한 레코드를 구성하기 위해 dibit 셀을 서로 연결하여야 하므로 비교기의 지연 시간이 문제가 된다. 리바운드 정렬기는 파이프라인을 사용하여 Yasuura의 정렬기와 같은 $2N$ 사이클의 처리 속도를 가지며, Miranker의 VLSI 정렬기보다 2배가 빠르다. 그러나 실제 설계하여 비교한 최대 클럭 주파수는 리바운드 정렬기가 약 3배 빠르다. 칩면적은 Yasuura의 정렬기의 1/2정도이며 Miranker의 VLSI 정렬기보다 조금 작으므로 리바운드 정렬기가 가장 효율적이다.

세 가지 병렬 정렬기를 비교한 결과에서 리바운드 정렬기를 VLSI로 구현하기 위해 비교기, 레코드의 조향 방법에 대해 고찰하였으며 한 개의 칩은 8개의 16비트 레코드를 정렬할 수 있으며 많은 양의 레코드 데이타를 처리하기 위해 수직으로 확장 가능하도록 하였다.

이 구현된 칩은 시뮬레이션과 검증 과정을 통해 동작이 확인되었으며 이 칩을 제작하여 실제적인 테스트가 필요하다. 이 칩을 full custom IC 설계 방법으로 설계한다면 10%~20% 정도의 속도 향상과 칩면적을 줄일 수 있으나 설계하는데 상당한 시간이 걸린다. 그리고 사용한 CMOS 기술을 1㎛ CMOS기술로 바꾼다면 현재의 칩보다 더 큰 비트의 레코드를 정렬할 수 있으며 동시에 50MHz 이상의 처리속도로 32개 이상의 레코드를 정렬할 수 있는 칩으로 설계할 수 있다.

참 고 문 헌

[1] D. E. Knuth, 'The Art of Computer Programming : Sorting and Searching,' Addision-Wesley, Vol. 3, 1973.

[2] S. G. Akl, 'Parallel Sorting Algorithms,' Academic Press, 1985.

[3] D. Bitton, D. J. Dewitt, D. K. Hsiao and J. Menon, "A Taxonomy of Parallel Sorting," ACM Comput. Surveys, Vol. 16, pp. 287-318, Sep. 1984.

[4] G. Bilardi and F. P. Preparata, "A minimum area VLSI network for O(log n) time sorting," IEEE Trans. Comput., Vol. C-34, No. 4, pp. 336-343, Apr. 1985.

[5] F. T. Leighton, "Tight bounds on the complexity of parallel sorting," IEEE Trans. Comput., Vol. C-34, No. 4, pp. 344-354, Apr. 1985.

[6] H. M. Alnuweiri, "A New Class of Optimal Bounded-Degree VLSI Sorting Networks," IEEE Trans. Comput., Vol. 42, No. 6, pp. 746-752, Jun. 1993.

[7] M. Beck, D. Bitton and W. K. Wilkinson, "Sorting Large Files on a Backend Multiprocessor," IEEE Trans. on Comput. Vol. 37, No. 7, pp. 769-778, July 1988.

[8] R. Francis and L. D. Mathieson, "A Bench Mark Parallel Sort for Shared Memory Multiprocessors," IEEE Trans. on Comput. Vol. 37, No. 12, pp. 1619-1626, Dec. 1988.

[9] P. H. Singgih, H. B. Demuth, M. T. Hagan and R. L. Wainwrite, "Parallel Merge-Sort Algorithms on the HEP," ACM Fourteenth Annual Comput. Science Conf.:CSC '86 Proceedings, pp. 237-244, Feb. 1986.

[10] M. J. Carey, P. M. Hansen and C. D. Thompson, "RESST : A VLSI Implementation of a Record-Sorting Stack," Report No. UCB/CSD 82/101, Univ. of California, Berkeley, Apr. 1982.

[11] K. M. Chung, F. Luccio and C. K. Wong,

"On the complexity of sorting in Magnetic Bubble Memory Systems," IEEE Trans. on Comput. Vol. C-29, No-7, pp. 553-563, July 1980.

[12] D. T. Lee, H. Chang and C. K. Wong, "An On-chip Compare/Steer Bubble Sorter," IEEE Trans. Comput. Vol. C-30, pp. 396-405, June 1981.

[13] S. Rajgopal, S. Ghatak, J. McNair, S. Kumar and D. Bouldin, "A VLSI Implementation of the Parallel Enumeration Sort Technique," IEEE VLSI Tech. Bulletin, Vol. 1, No. 3, pp. 35-42, Dec. 1986.

[14] H. Yasuura, N. Takagi and A. Yajima, "The Parallel Enumeration Sorting Scheme for VLSI," IEEE Trans. Comput. Vol. C-31., pp. 1192-1201, Dec. 1982.

[15] G. Miranker, L. Tang and C. K. Wong, "A 'Zero-Time' VLSI Sorter," IBM J. of Res. Develop. Vol. 27, pp. 140-148, Mar. 1983.

[16] T. C. Chen, V. Y. Lum and C. Tung, "The Rebound Sorter : An Efficient Sort Engine for Large Files," IEEE Proc., pp. 312-318, June 1978.

[17] S. Todd, "Algorithm and Hardware for a Merge Sort using Multiple processors," IBM J. of Res. Develop. Vol. 22, No. 5, pp. 509-517, Sep. 1978.

[18] N. Takagi and C. K. Wong, "A Hardware Sort-Merge System," IBM J. of Res. Develop. Vol. 29, pp. 49-67, Jan. 1985.

[19] C. Tung, T. C. Chen and H. Chang, "Bubble Ladder for Information Processing," IEEE Trans. on Magn., Vol. Mag-11, No. 5, pp. 1163-1165, Sep. 1975.

[20] T. C. Chen and C. Tung, "Storage Management Operations in Linked Uniform Shift Register Loop," IBM J. of Res. Develop., Vol. 20, pp. 123-131, Mar. 1976.

윤 지 헌

1974년 영남대학교 공과대학 전자공학과 졸업(공학사)
1979년 영남대학교 대학원 전자공학과 졸업(공학석사)
1990년 영남대학교 대학원 전자공학과 박사과정(전자계산기 전공) 수료
1979년~현재 영남전문대학 전자계산기과 교수
관심분야 : 병렬처리, 알고리즘, 프로그래밍언어 등.

안 병 철

1976년 영남대학교 공과대학 전자공학과 졸업(공학사)
1978년 영남대학교 대학원 전자공학과 졸업(공학석사)
1986년 미국 오레곤 주립대학 전기 및 전산공학과(공학석사)
1989년 미국 오레곤 주립대학 전기 및 전산공학과(공학박사)
1978~84년 국방과학연구소 연구원
1989~92년 삼성전자 컴퓨터부분 수석연구원
1992년~현재 영남대학교 공과대학 전산공학과 조교수
관심분야 : VLSI설계, 컴퓨터구조, 멀티컴퓨터, 그래픽스, 네트워크.