A Pipelined Parallel Hardware Sorter

Wen-Lung Shu
CSIE Department, Chung-Hua University
No:30, Tung-Shiao Rd., Hsin-Chu, Taiwan
wlshu@chu.edu.tw

Abstract

A new hardware sorter which combines both Batcher's parallel merge sort [1] and Stodd's pipelined two way merge sort algorithm [2] is proposed in the paper. This hardware contains one k-sorter and log(n/k) k-to-k mergers, and can sort n records in O(n/k) assuming data is retrieved through k parallel data paths. The internal processing algorithm and control unit of this pipelined parallel device have been completely designed. This sorter is readily suitable for VLSI Implementation, and can be used to process very large databases efficiently.

Key words: hardware sorter, parallel, pipeline.

1. Introduction

Many sorting approach was presented in the past. Part of these sorting algorithms operate on a single processor with the best performance of order n*log n cycles to sort n records. They are the quicksort, heap sort, and merge sort [3]. Some sorting algorithm may complete sort with time proportional to n, but only in certain circumstances. Address sorting [3] requires the spread of sort key values to be known and fairly random. Digital sorting [3, 6] is performed by using short keys. Several multiple processor sorts are proposed also. Such as Batcher's merge exchange sort [3,7], Thompson and Kung's mesh sort [4], and Chen's parallel bubble sort [5]. The odd-even transposition sort has been extended to the multiprocessor case in [8]. Mesh sort algorithm was processed among the multiple mesh networks in [9, 10]. Multi-way merge algorithm was applied on multiple mesh networks in [11, 12].

In pratical condition, n records usually are distributed to k locations. A new hardware sorter is designed in the paper by combining both Batcher's parallel merge sort and Stodd's pipelined two-way merge algorithm. This hardware contains one k-sorter and log(n/k) k-to-k-mergers, and can sort n records in O(n/k) assuming data can be retrieved through k parallel data paths. Basic parallel merge unit of this sorter includes data flow control, state sequencer, micro-code generator and internal processing algorithm are designed. This sorter is readily suitable for VLSI Implementation. Let data transfer rate be matched with data processing rate. Then this type sorter can reach to the highest degree of efficiency, and can meet the emerging need of processing very large databases.

The overview of Todd's algorithm and overall architecture of the proposed sorter are given in Section 2. Data flow control, internal processing unit, state sequence and micro-code generator are developed in Section 3.

2. The Design Concept of the Proposed Sorter

Todd's two way merge sort algorithm and the proposed sorter are discussed in Section 2.1 and Section 2.2.

2.1 The overview of Todd's two way merge sort algorithm

This algorithm is a variation of a straight two-way merge sort. A serial two-way merge sort operates in several passes, with each pass creating sorted sequences of records. The first pass creates strings of two records; the second pass merges each pair into four-record strings. After i passes, the strings have length 2^i. After log n passes, all n records are in one sorted string. The passes of this algorithm are run overlappedly rather than serially. Each pass is supported by a separate processor. The passes are run overlappedly using multiple processors which merge each pair of strings into a sorted sequence.

Assume n is the number of records, and n is equal to 2^r where r is integer. There are r+1 processors, 0 through r. The output from the ith processor consists of sorted sequences of 2^i records, created by merging two output strings from the (i-l)th processor. The example of sorting 4 elements is shown in Table 1.

2.2 The overall architecture of the proposed sorter

Table 1. An example of Todd's algorithm for sorting 4 elements.

Cycles	Input	P_0	P_1	P_2	Output	Comments
0	d b c a	→	→	→		Input consists of length 1 strings
		→	→	→		
1	d b c	→ a	→	→		P_0 switches input to alternate queues.
		→	→	→		
2	d b	→ a	→	→		Shift a to upper queue, b to lower queue.
		→ c	→	→		
3	d	→ b	→ a	→		P_1 begin to merge strings a and c.
		→ c	→	→		
4		→ d	→ b a	→		String b a is finished.
		→ c	→	→		P_0 has passed the last record.
5		→ d	→ b a	→		P_1 begins to merge strings c and d.
		→	→ c	→		
6		→	→ b	→	a	P_2 now starts on b a and d c.
		→	→ d c	→		
7		→	→	→	b a	P_2 continues processing.
		→	→ d c	→		
8		→	→	→	c b a	P_2 passes d c to output.
		→	→ d	→		
9		→		→	d c b a	P_2 completes d c b a.
		→		→		

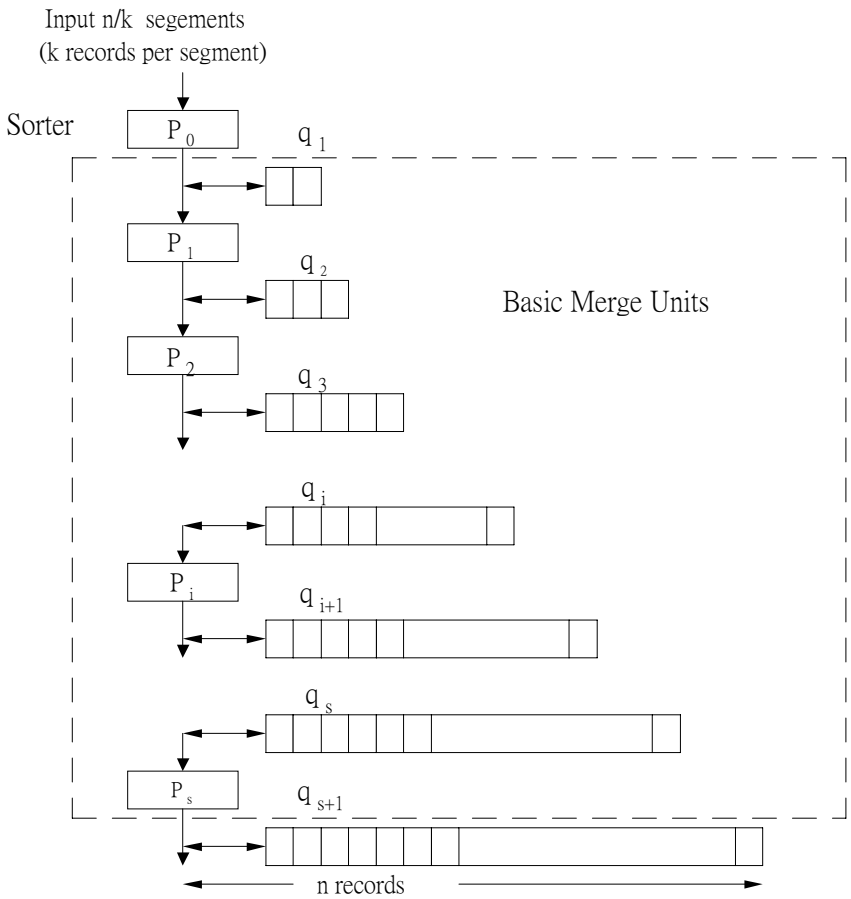


Figure 1. The overall architecture of the proposed sorter.

A hardware sorter that combines Batcher's bitonic merger and two way merge algorithm is shown in Figure 1. Assume that bitonic sorter and mergers are consist of bit-slice comparators. The k bit strings in a segment can enter the hardware simultaneously. This sorter is developed by using one k-bitonic sorter in P_0, and s bitonic *kxk* mergers in P_1~P_s where s=log(n/k). A k-sorter needs (k/2)[log k(1+log k)/2] comparators and the *kxk* merger needs k(1+log k) comparators. P_0 sorts k records in each segment, then transfer results to any available space in q_1. P_1~*Ps* process internal processing algorithm, then data can be sorted in 0(n/k).

Processor P_i can merge 2^{i-1}segments (k parallel records in each segment) with another 2^{i-1} segments in q_i and transfer the resulting data into q_{i+1}. Using Stodd's two way merge algorithm, it only requires the space of 2^{i-1}+l segments in q_i to process 2^{i-1} to 2^{i-1}merge. Hence,

Total space requirement = 2^s+ s -1 segments
= n/k+log(n/k)-l segments.

But, in order to make the developing work easy, a simple model is adopted in the following section. Three assumptions are made for this model:

1. Bitonic merger can completely merge data in a cycle.

2. Queue q_i is divided into two queues A_i and *Bi* which contain 2^{i-1} segments.

3. Using the semaphore signals, the odd merge units of P_1,…,P_s are processed first, and even merge units then processed in next turn, and so on.

It is noted that the first assumption can be improved by changing delay cycles for different record size in merge related states. The architecture of merge unit and internal processing algorithm can be further improved, such that pipeline process in Table 1 and smaller storage requirement can also be achieved in our design.

3. Design the Basic Parallel Merge Unit of Proposed Sorter

The proposed hardware sorter has been designed in this section. Figure 2 shows the block diagram of a basic parallel merging unit. Each parallel merging unit includes data flow control, hardware merger, state sequencer and micro-code generator.

3.1 The design of data flow control

The design of data flow control is shown in Figure 3. It includes four counters (countA_i, countB_i, lastA_i, and lastB_i,), three flags (top, E_i, and semaphore[i] and output signals (Z_{1i},…,Z_{7i}). The main functions of these 4 counters and three flags are described below :

1. The countA_i, and countB_i are utilized to indicate the number of data segments expected to be processed in the current string, when 2^{i-1} data segments are merged with 2^{i-1} data segments . The binary value of 2^{i-1} is loaded into countA_i and countB_i at initial. The number of countA_i (or countB_i) will be decreased by 1 when A_i (or B_i) is shift right one data segment. CountA_i, (or countB_i) is zero means that the queue A_i (or queue B_i) already processes 2^{i-1} data segments in this string. Output line Z_{1i} become 1 in this case. Count A_i have two input lines : C_{1i} loads 2^{i-1} value, C_{2i} is "DCR" will decrease counter by 1. It is similar to countB_i.

2. The lastA_i (or lastB_i) can tells the number of data segments remaining in the queue A_i (or B_i). A_i, or B_i may contains data belonging to current and next strings. When last string is found, lastA_i or lastB_i can be treated also as the final number of segments waiting to be processed. P_i activates when both lastA_i, and lastB_i are not zero, or when flag E_i =1 (It tells that $P_{i-1}$is processing the last string). The lastA_{i+1} or lastB_{i+1} must be incremented by 1 when P_i transfers a data segment to P_{i+1}, and lastA_i, and lastB_i must be decreased by 1 after right shifting. Counter lastA_i have three input lines: C_{5i} (CLR) clear lastA_i, C_{6i} (DCR) decrements lastA_i by 1, and C_{7i} (INC) will increments lastA_i, by 1. In lastA_i, Z_{3i} is zero detecting signal. Similar to C_{8i} (CLR), C_{9i} (DCR), C_{10i} (INC) and Z_{4i} in lastB_i counter.

3. The one bit flag "top" is used to indicate which queue is outputting. The top is initialized to 1, and is set to zero if lastA_i is zero. If top=l then P_i outputs to A_{i+1}, else to B_{i+1}. Input line C_{11i} sets flag "top" to 1, C_{16i} complements this flag, and output line Z_{5i} is the complement of flag "top".

4. E_i is a flag associated with every data segment. E_i = *0* represents this data segment is not the last one, otherwise Ei=1. Output $Z_{6\,i}$ must be set to 1.

5. The semaphore SM[i] can be updated by P_{i-1} and P_i. It is used to lock data in the pipelining system. This flag will effects the output Z_{7i}.

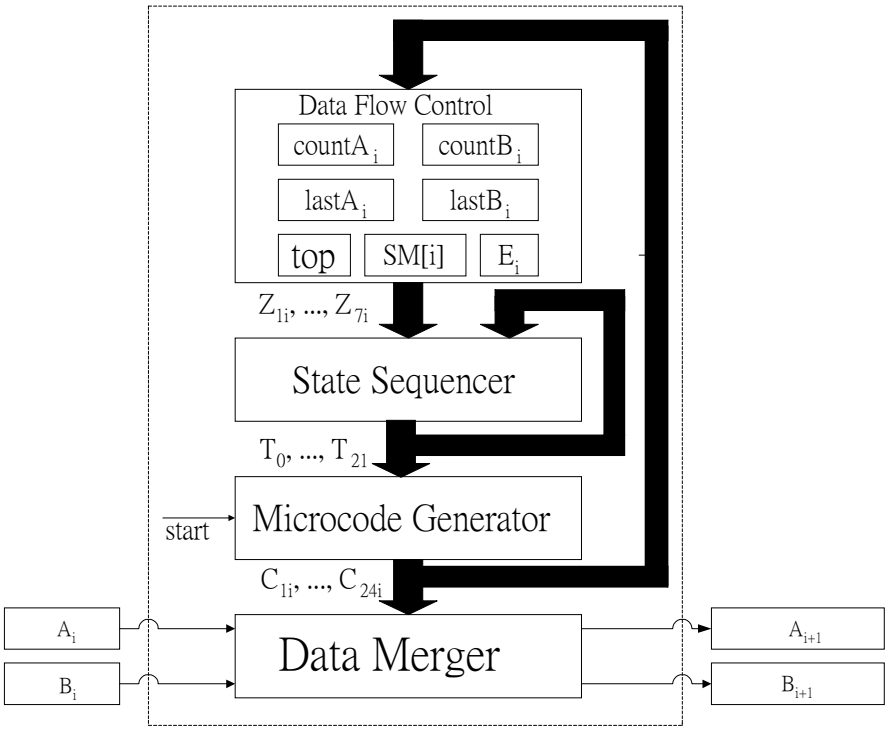
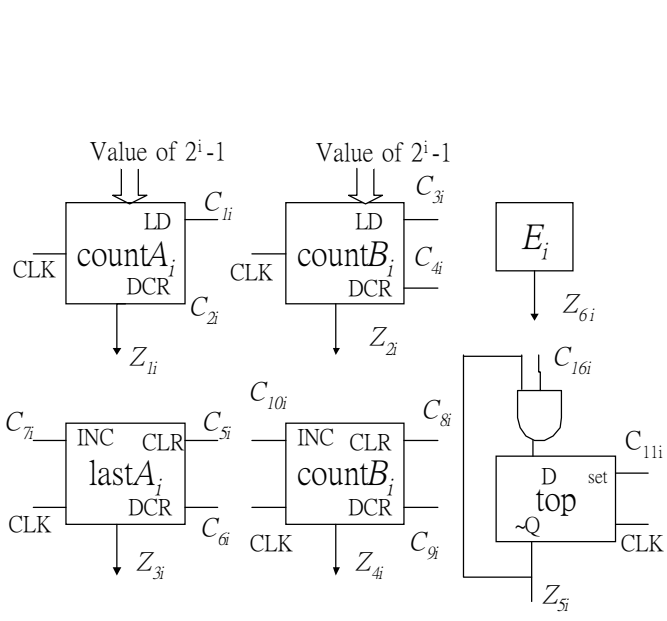


Figure 2. The Merge Unit P_i of the proposed Sorter.



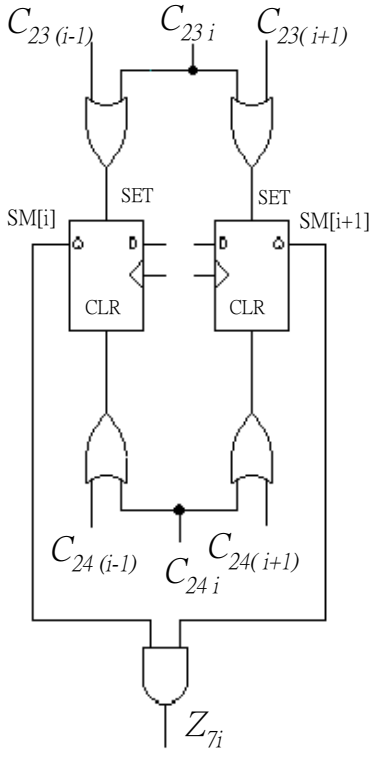


Figure 3. Control Registers in Data Merger.

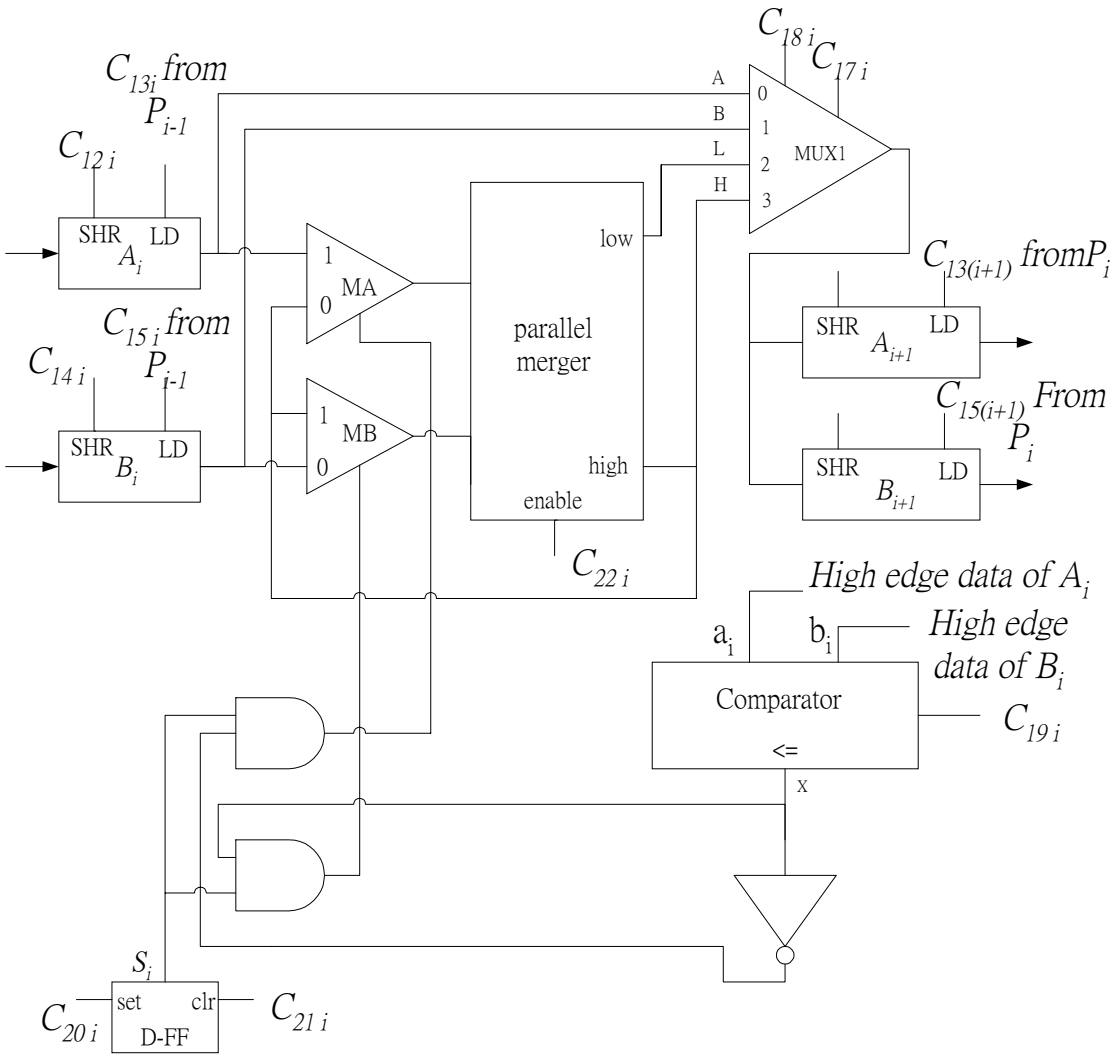


Figure 4. The Hardware Design for Merger P_i.

3.2 The design of data merger

The hardware of data meger is given in Figure 4. A_i and B_i, are shift registers used as two queues of processor P_i. When P_i processes the first data segment in the queue, the processing data segments are retrieved from queues by shifting registers to the right. The "SHR" control line C_{12i} or $C_{14\,i}$ is used to shift right for A_i or B_i. When a data segment is transferred from P_{i-1} to P_i the data is moved to the location pointed by lastA_i (or lastB_i). lastA_i (or lastB_i,) will be incremented by one thereafter.

The internal parallel merger can be designed by using odd-even merger, bitonic merger. The enable line (C_{22i}) will enable the parallel merger. After input segments are merged, low output segment will be transferred to P_{i+1} and high output segment will be returned back for next merge.

The register S_i has two control lines : clear (C_{21i}) and set (C_{22i}), S_i is cleared at initial, such that data segments from A_i and B_i will be merged first. Input data are passed through multiplexer MA and MB. Later S_i is set to 1. Signal "x" is the result of comparing two edge records (denoted as a_i and b_i) of two input segments from *Ai*, and *Bi*,. If x = 0 ($a_i > b_i$) then data segment from *Bi* is chosen to merge with feedback segment through MA and MB, otherwise data segment of *Ai* is chosen. Control line C_{19i} is used to initialize edge record comparator.

There is a Multiplexer MUX1 uses control lines (C_{17i}, and $C_{18\,i}$) to select the output data

segment from A, B, L or H. When input data are merged , data segment L will be transferred to P_{i+1}. After merger is no longer needed, segment H, segment A or segment B can be passed to P_{i+1} in the next cycle. The function of this multiplexer is described on detail in the internal processing algorithm

3.3 The design of internal processing algorithm and state sequencer

In order to design the state sequencer, the internal processing algorithm are developed by using output signals of data flow control (Z_{1i}, …,Z_{7i}) and micro-codes (C_{1i}, …,C_{24i}). Micro-codes will be set or reset according to the current conditions indicated by Z_{1i}, …,Z_{7i}. This modified internal processing algorithm is given below :

Registers:
A_i[2^{i-1}:1], B_i[2^{i-1}:1], top[0], S_i[0], countA_i[2^{i-1}:1], countB_i[2^{i-1}:1], lastA_i[2^{i-1}:1], lastB_i[2^{i-1}:1], SM=array[l...log(n)] of semaphore initialized to up, P_i the ith processing unit.

```
Begin
  while ( ~ start ) do NOP;
  set C5i, C8i, C11i, C23 i;  /* initialize values */
  repeat
    while ( Z3i or Z4i ) and(~ Z6 i) do NOP;
   /* wait until both queues have data or
   last string is indicated '"/
    if ~Z3i and ~Z4i then
      begin /* begin merging */
        set C1i, C3i, C19i, C21i;
       /* initialize values for new string
   */
          repeat
            call MERGE
     /* merge data and send data P,^ */
          until Z1i or Z2i or Z3i or Z4i;
/* If a queue is empty in the string then
feedback data and the remaining data must
be moved */
          if  ~Z2i, and ~Z4i then
            begin
              MOVEH(A);
              while ~Z1i, and ~Z3i
                call MOVE(A);
            end;
          if ~Z2i and ~Z4i then
            begin
              MOVEH(B);
               while ~Z2i and ~Z4i
                 call MOVE(B);
            end;
          set C16i;
/* After complete a string , data will be moved to
another queue */
          end
          else /* processing for last string */
            DELAY_A_CYCLE;
          repeat
            call MOVE(A);
          until Z3i;
  until Z3i and Z4i;
end;

subroutine MERGE
Wait (Z7i, Z7(i+1) );
enable merger;
if Z5i then
  set ~C17i, C18i, C22i, C24i, C13(i+1), C7(i+1)
else
  set ~C17i, C18i, C22i, C24i, C15(i+1), C10(i+1)
if (ai ≦ bi ) then
  set C7(i+1), C13(i+1), C2 i, C6 i, C12 i
else
  set C10(i+1), C15(i+1), C4i, C9 i, C14i
set C20i;
set C23i;

subroutine MOVEH
parameter X
Wait (Z7i, Z7(i+1) );
if ~Z5i then
  set C17i, C18i, C24i, C13(i+1), C7(i+1)
else
  set C17i, C18i, C24i, C15(i+1), C10(i+1);
if   X = A
  set C2i, C6i, C12 i
else
   set C4i, C9i, C14 i;
set C23i;

subroutine MOVE
parameter X
Wait ( Z7i, , Z7(i+1) );
If  X  =
A
  begin
    if ~Z5i then
      set C17i, C18i, C24i, C13(i+1), C7(i+1)
    else
      set C17i, C18i, C24i, C15(i+1), C10(i+1);
    set C2i, C6i, C12i;
  end;
else
   begin
      if ~Z5i then
```

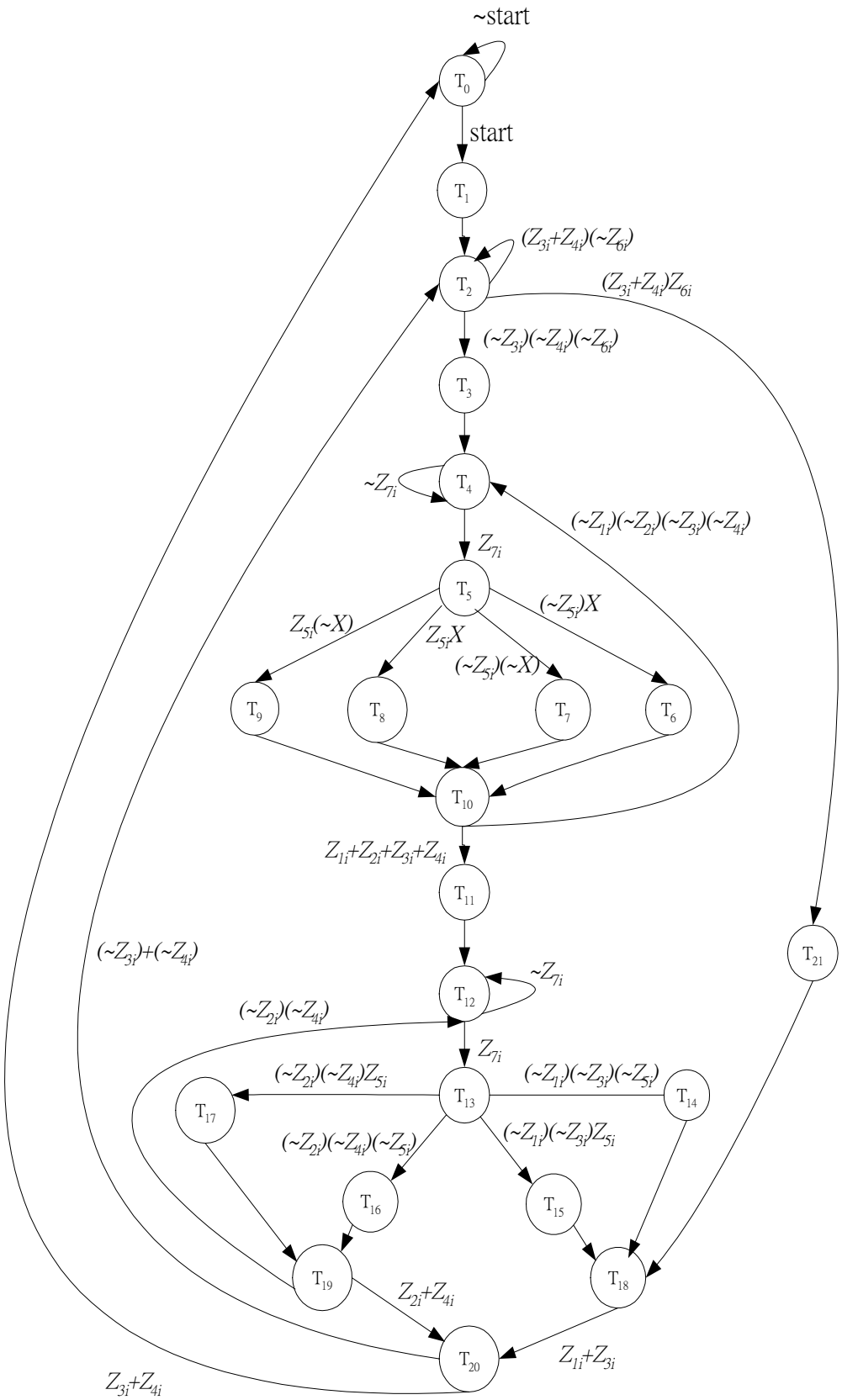


Figure 5. The State Diagram for State Sequencer.

set C_{17i}, C_{18i}, C_{24i}, $C_{13(i+1)}$, $C_{7(i+1)}$
else
set C_{17i}, C_{18i}, C_{24i}, $C_{15(i+1)}$, $C_{10(i+1)}$**;**
set C_{4i}, C_{9i}, C_{14i}**;**
end;

From the previous internal processing algorithm, a state diagram in Figure 5 can be derived. The state sequencer can be designed thereafter. Within several wait states, system will wait until the wanted conditions was found. However in the states T_6 ~T_9, T_{14}, T_{15} and T_{21}, system has to give enough delay time for merging input records with different data size.

3.4 Designing micro-code generator

Micro-code generator itself is a combination circuits, T_0,…,T_{20} are inputs and C_{1i},…,$C_{24\ i}$ are outputs of this circuits. The boolean expressions for this combination circuits can be derived from the above state diagram. They are:

$C_{1i} = T_3$
$C_{2i} = T_6 + T_8 + T_{14} + T_{15}$
$C_{3i} = T_3$
$C_{5i} = T_1$
$C_{6i} = T_6+T_8+T_{14}+T_{15}$
$C_{7(i+1)} = T_6+T_7+T_{14}+T_{16}$
$C_{8i} = T_1$
$C_{9i} = T_7+T_9+T_{16}+T_{17}$
$C_{10(i+1)} = T_8+T_9+T_{15}+T_{17}$
$C_{11i} = T_1$
$C_{12i} = T_6+T_8+T_{14}+T_{15}$
$C_{13(i+1)} = T_6+T_7+T_{14}+T_{16}$
$C_{14i} = T_7+T_9+T_{16}+T_{17}$
$C_{15(i+1)} = T_8+T_9+T_{15}+T_{17}$
$C_{16i} = T_{20}$
$C_{17i} = (\sim T_5)+T_{11}+ (\sim T_{18})+T_{19}+(\sim T_{21})$
$C_{18i} = T_5+T_{11}+ (\sim T_{18})+(\sim T_{19})+(\sim T_{21})$
$C_{19i} = T_3$
$C_{20i} = T_{10}$
$C_{21i} = T_3$
$C_{22i} = T_6+T_7+T_8+T_9$
$C_{23i} = -T_1+T_{10}$
$C_{24i} = T_5+T_{13}$

4 Conclusion

A pipelined parallel hardware sorter has been developed in the paper. The parallel and pipelining sort algorithm can be performed on the hardware when each basic merge unit implements internal processing algorithm. In this system, data transfer rate can be matched with data processing rate. Hence a large amount of data can be sorted efficiently.

5. Reference

[1] Michael J. Quinn, " Designing Efficient Algorithms for Parallel Computers ", pp. 84 – 100, McGraw Hill.

[2] S.todd, "Algorithm and Hardware for a Merge Sort Using Multiple Processors", IBM J. Res. Develop. Vol. 22, NO. 5, September 1978.

[3] D.E. Knuth," Sorting and Searching", The Art of Computer Programming. Vol.3 , Addison Wesley Publishing Company, Reading, MA, 1973.

[4] C. D. Thompson and H. T. Kung, "Sorting on a Mesh-Connected Parallel Computer", Commun. ACM 20, 263 (1977).

[5] T. C. Chen, K. P. Eswaran, V. Y. Lum, and C. Tung, "Apparatus for Transposition Sorting of Equal-Length Records in Overlap Relation with Records Loading and Extraction ", U.S. Patent Application Serial Number 685,859,1977.

[6] S. Even, "Parallelism in Tape Sifting ", Commun, ACM 17,202(1974).

[7] K.. E. Batcher, " Sorting networks and their applications, " Proc. AFIPS Spring Joint Computer Conf. pp. 307-314, 1968.

[8] G. Baudet and D. Stevenson, "Optimal sorting algorithms for parallel computers, " IEEE Trans. Comput., Vol. 27, No. 1, pp. 657-661. July 1978,

[9] M. De, D. Das, M. Ghosh and B. P. Sinha,"An efficient sorting algorithm on the multi-mesh network" IEEE Trans. Comput., pp. 1132 – 1137. Oct. 1997.

[10] D. Das, M. De, and B. P. Sinha, "A new network topology with multiple meshes ", IEEE Trans. Comput., pp. 536 – 551, May 1999, Vol: 48. No: 5.

[11] P .F. Corbett and I. D. Scherson, "Sorting in mesh connected multiprocessors" , IEEE Trans. on Parallel Distrib. Systems. pp. 626 – 632, Sept. 1992, Vol: 3, No: 5.

[12] Bhabani P. Sinha, Amar Mukherjee, "Parallel Sorting Algorithm Using Multiway Merge and Its Implementation on a Multi-Mesh Network", Journal of Parallel and Distributed Computing, Vol:60, No:7, July 1, 2000.