ALGORITHMICALLY SPECIALIZED PARALLEL COMPUTERS

EDITED BY

LAWRENCE SNYDER
Department of Computer Science
University of Washington
Seattle, Washington

LEAH H. JAMIESON
School of Electrical Engineering
Purdue University
West Lafayette, Indiana

DENNIS B. GANNON
Department of Computer Sciences
Purdue University
West Lafayette, Indiana

HOWARD JAY SIEGEL
School of Electrical Engineering
Purdue University
West Lafayette, Indiana

1985

ACADEMIC PRESS, INC.
(Harcourt Brace Jovanovich, Publishers)
Orlando San Diego New York London
Toronto Montreal Sydney Tokyo

Sorting Records in VLSI

Michael J. Carey
Paul M. Hansen
Clark D. Thompson

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720

ABSTRACT

This report discusses a VLSI implementation of a record-sorting stack. Records are represented on the stack as (key, record-pointer) pairs, and the operations supported are PUSH, POP, and CLEAR. When records are POPped, they are returned in smallest-first order. The implementation allows the sorting of n records in $O(n)$ time, and the design is cascadable so that the capacity of a single VLSI chip does not limit the amount of data which may be sorted.

This report describes a paper design and evaluation, and thus serves two purposes: It describes one particular VLSI sorting circuit, and it also serves as a case study in VLSI design methodology. The algorithm is described, the overall chip organization and data flow are presented, and detailed circuits, layouts, and timing analyses are given.

1. Introduction

This report describes RESST, a VLSI REcord Sorting STack. The chip operates in a stack-like manner, allowing (key, record-pointer) pair representations of records to be pushed and popped. Each key is an 8-bit integer, and each record-pointer is an 8-bit pointer value. When a (key, record-pointer) pair is popped from the RESST chip, the pair with the smallest key value is returned. Hence, RESST may be used to sort a group of records by pushing them and then popping them. We envision RESST chips being utilized as a form of hardware support for database systems, perhaps as disk buffer storage for providing automatic sorting of secondary indices for multi-relation query processing in a relational database system [Ull80].

This work was supported by the National Science Foundation Grant ECS-8110684, a Chevron U.S.A. Career Development Grant, a California MICRO Fellowship, the Air Force Office of Scientific Research Grant AFOSR-78-3596, and the Naval Electronic Systems Command Contract NESC-N00039-81-C-0569.

ISBN 0-12-654130-2

The algorithm chosen for the VLSI RESST implementation, a parallel version of the classic bubblesort algorithm, is quite similar to other recent work in the area of hardware sorting devices. Armstrong and Rem [Arm82], Chung, Luccio, and Wong [Chu80], Leiserson [Lei79], Miranker, Tang, and Wong [Mir82], and Mukhopadhyay and Ichikawa [Muk72, Muk81] have all published papers on similar $O(n)$ time hardware sorting algorithms. The particular algorithm chosen for the RESST implementation is basically the Weavesort algorithm of Mukhopadhyay [Muk81], though it was independently developed when RESST was designed.

In the remainder of this report, we will describe the hardware sorting algorithm and a 32 data item VLSI implementation. The algorithm and the overall chip organization will be presented in section 2. Circuit schematics and cell layouts will be presented in section 3. Performance estimates for RESST, based on simple SPICE models for timing and power consumption, will be discussed in section 4. Section 5 contains a discussion of some possible enhancements for RESST. Finally, section 6 contains a summary of the main things that we have learned from our RESST experience.

2. High-Level Design Description

This section of our report discusses the high-level design issues involved in the RESST project. The RESST sorting algorithm is presented, and the overall structure and data flow of the RESST chip are described.

2.1. A Hardware Sorting Algorithm

The algorithm that we chose to use in the design of RESST is a parallel bubblesort algorithm. Let n be the number of items to be sorted, where each item has a fixed-length key (that is, key length is independent of n). Let N be the number of items that the chip can hold. This algorithm allows n items to be sorted in $O(n)$ time using $O(N)$ chip area. We chose this algorithm from the many possible VLSI sorting algorithms [Tho82] for several reasons, including simplicity, regularity, and extensibility.

The parallel bubblesort algorithm is two-phase in nature. That is, each step of the sorting process consists of two substeps. Let $key[i]\langle j\rangle$ and $recPtr[i]\langle j\rangle$, where $i = 0,1,\ldots,31$ and $j = 0,1,\ldots,7$, denote the jth bit of the ith word of the RESST key and record storage, respectively. (For the ith element, $key[i]\langle 0\rangle$ and $recPtr[i]\langle 0\rangle$ represent the least significant bits.) Let $key[-1]$ and $recPtr[-1]$ represent the values presented to/from the I/O pins of the chip. The two phases of the algorithm are given in terms of this notation in Figure 2.1.

The first substep of the algorithm (phase 1) involves shifting data in or out, clearing the chip's storage cells, or refreshing the chip's storage cells. The PUSH, POP, and REFRESH operations have the obvious meanings, while the CLEAR operation is somewhat more subtle. Associated with each 8-bit key is a 9th, hidden bit. This bit serves to distinguish real key values from empty cells, with a one in this most significant bit


```
Phase 1:    { I/O Phase }

    case operation of
      PUSH:
          forall i in 0..31 pardo
            key[i] := key[i-1];
            recPtr[i] := recPtr[i-1];
          od;
      POP:
          forall i in 0..31 pardo
            key[i-1] := key[i];
            recPtr[i-1] := recPtr[i];
          od;
      CLEAR:
          forall i in 0..31 pardo
            key[i]<8> := 1;
          od;
      REFRESH:
          forall i in 0..31 pardo
            key[i] := key[i];
            recPtr[i] := recPtr[i];
          od;
    end;

Phase 2:    { Compare/Exchange Phase }

    forall i in 0..31 by 2 pardo
      if key[i] > key[i+1] then
        Exchange(key[i], key[i+1]);
        Exchange(recPtr[i], recPtr[i+1]);
      fi;
    od;
```

Figure 2.1: Hardware Sorting Algorithm.

position representing an empty cell. This way, nonempty cells are always kept towards the left (I/O) side of the chip.

The second substep of the algorithm (phase 2) does the actual sorting, and involves comparing and conditionally exchanging the (key, record-pointer) pairs associated with keys 0 and 1, 2 and 3, and so on, up to $N-2$ and $N-1$. As denoted by the **forall...pardo...od** notation, these pairwise comparisons take place in parallel. An example of the operation of this algorithm is depicted in Figure 2.2.

The advantages of this algorithm for VLSI implementation should be immediately obvious. Because data only moves between adjacent storage cells, and because comparisons take place only between every other pair of adjacent cells, there is no need for global communication. As a result, the cell layout can be organized as a simple linear array of storage cells with a compare/exchange cell between every other pair of adjacent storage cells. Also, it is quite easy to accommodate the sorting of more than N items by making RESST chips cascadable. Providing cascadability just involves buffering the right-hand outputs and inputs of

operation	item	phase	REGISTER PAIRS 0	1	2	3	4	5	
PUSH	1	1	1						...
		2	1						
PUSH	3	1	3	1					...
		2	1	3					
PUSH	9	1	9	1	3				...
		2	1	9	3				
PUSH	5	1	5	1	9	3			...
		2	1	5	3	9			
PUSH	2	1	2	1	5	3	9		...
		2	1	2	3	5	9		
PUSH	8	1	8	1	2	3	5	9	...
		2	1	8	2	3	5	9	
POP	1	1	8	2	3	5	9		...
		2	2	8	3	5	9		
POP	2	1	8	3	5	9			...
		2	3	8	5	9			
POP	3	1	8	5	9				...
		2	5	8	9				
POP	5	1	8	9					...
		2	8	9					
POP	8	1	9						...
		2	9						
POP	9	1							...
		2							

Figure 2.2. Example of Parallel Bubblesort Operation.

the last storage cell in the array and providing off-chip connections for them.

2.2. Chip Structure and Data Flow

The actual structure and data flow for RESST follow naturally from the preceding discussion of the algorithm and its advantages. As described in the discussion, the algorithm is two-phase, so our design utilizes a two-phase clocking scheme. The chip is organized as a linear array of storage and compare/exchange cells. In our physical design, we chose to group each pair of storage cells and their associated compare/exchange unit into a single cell, called a COL cell. The overall RESST structure, shown in terms of this type of cell, is shown in Figure 2.3. As shown, data flows horizontally between adjacent COL cells.

Each COL cell contains a pair of 8-bit keys, a pair of 8-bit record-pointers, and logic for comparing keys and exchanging (key, record-pointer) pairs. There are 16 COL cells in our RESST implementation, so

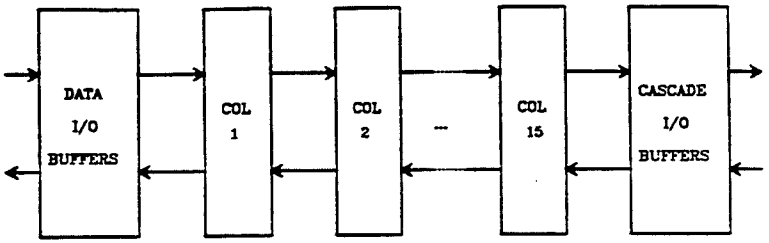

Figure 2.3. RESST Chip Structure.

the kth COL cell, $k = 0,1, \ldots, 15$, contains $key[2k]$<0:8>, $key[2k+1]$<0:8>, $recPtr[2k]$<0:7>, and $recPtr[2k+1]$<0:7>.

Five types of cells are used for building a COL cell: a CE cell, which contains storage and compare/exchange logic for a pair of key bits, a TOPCE cell, which is a CE cell with preset capability (to support the CLEAR function), an RP cell, which contains storage and exchange logic for a pair of record-pointer bits, a CBUF cell, which buffers a Manchester-type carry chain used for word-parallel comparisons, and a PHI2SIG cell, which generates clocked exchange signals from the end of the carry chain for controlling the CE and RP cell exchange logic. The structure of a COL cell in terms of these five subcell types is depicted in Figure 2.4.

As shown in the figure, data flows horizontally, the carry chain signal flows vertically towards the PHI2SIG cell, the phase 2 clock signal flows horizontally through the PHI2SIG cell, and the clocked exchange signals flow vertically out from the PHI2SIG cell. There are also clocked PUSH, POP, CLEAR, and HOLD signals which flow vertically through the COL cell, and power and ground which flow horizontally through each subcell of the COL cell.

3. Low-level Design Description

This section of our report discusses the design of the main circuits of the RESST chip. The COL cell is described in terms of its component subcells: the CE, RP, and TOPCE cells. Further details, including control logic and cell layouts for the RESST chip, are given in [Car82].

3.1. Compare-Exchange Circuit (CE)

Figure 3.1 shows a block diagram representation of the basic compare/exchange circuit (CE) showing control and data signals. Functionally, it can be simply described as a pair of semi-static registers with some additional circuitry to provide a comparison of the information contained in the two cells and to either pass the carry-like exchange chain signal EXCHIN to the next most significant bit or to assert EXCHOUT high if an exchange is called for by a mismatch in the cell. Figure 3.2 shows the circuitry in mixed notation. The italicized signal names refer to signals which stay within the bounds of a single CE cell.

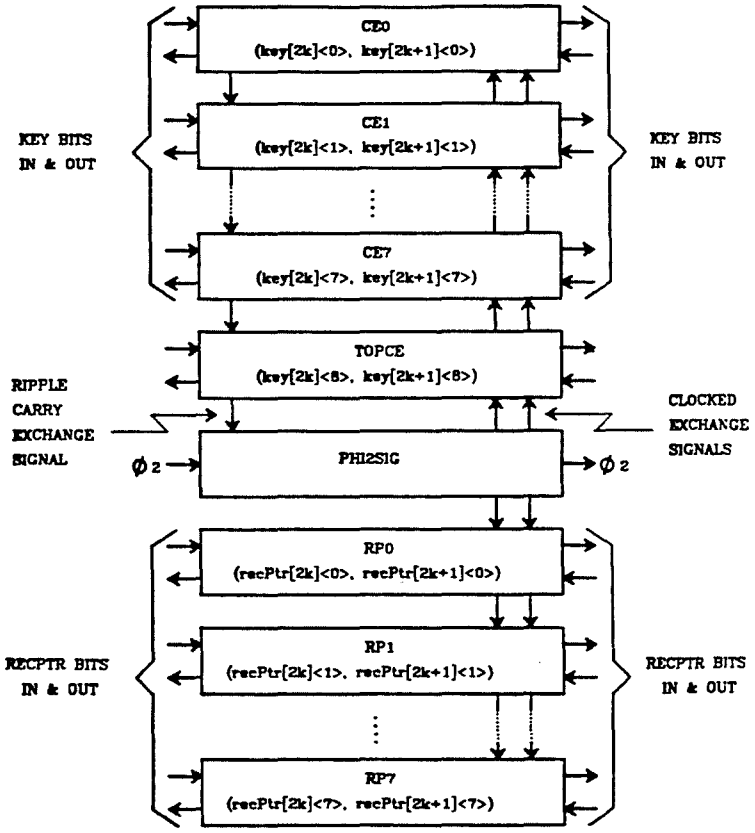

Figure 2.4. COL Cell Structure.

The controlling signals PUSH1 and POP1 function as you would expect. The user asserts either PUSH or POP in proper phasing with clock phase 1 (PHI1). PHI1 is ANDed to produce PUSH1 or POP1 which are then used to control writing into and reading from the RESST. Data present on BDIN will be passed to the gate of the first inverter of the cell during PUSH1. Data in the cell may also be passed to the left (read) via BDOUT by controlling POP1 in similar fashion. Obviously, POP1 and PUSH1 are mutually exclusive for proper circuit operation. During a PHI1 clock cycle in which neither reading nor writing of the RESST is desired, the signal HOLD1 guarantees that the information which is stored in the semi-static cell is refreshed.

Adjacent cells function in parallel, allowing true stack operations. Data is allowed to flow between the ith and i+1st CE cells by connecting UBDIN of the ith cell to BDOUT of the i+1st cell and, similarly, connecting UBDOUT of the ith cell to BDIN of the i+1st cell. Hence, data flows to the "right" during PUSH operations, and to the "left" during POP operations.

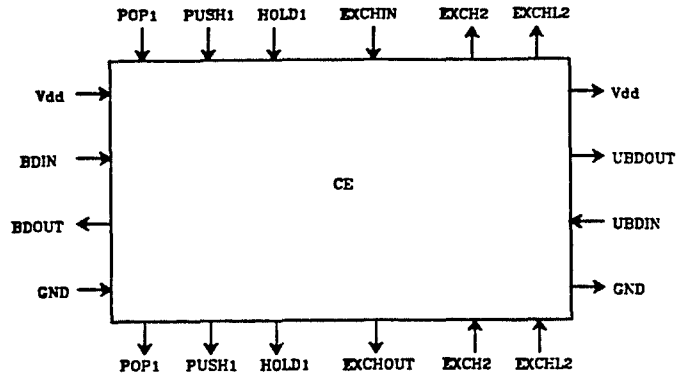

Figure 3.1. COMPARE/EXCHANGE Cell Block Diagram.

As mentioned above, the contents of the two cells are compared during clock phase 2 (PHI2). If a match occurs, the EXCHIN signal is propagated. If a mismatch occurs, EXCHOUT will be the inverted value of the rightmost bit (signaling that an exchange is needed if the rightmost bit is "0"). An exchange is required if EXCH2 goes high after the carry-like EXCHIN/EXCHOUT signal has been fully propagated through all the CE cells in a word (going from the least to the most significant bit position). If an exchange is indeed called for, the two EXCH2-controlled pass transistors in the circuit serve to exchange the inverted data of the two storage cells. Thus, data is always applied to the leftmost inverter of a pair on PHI1 and to the rightmost inverter on PHI2. This is consistent throughout the design.

3.2. Record Pointer Circuit (RP)

As mentioned in the functional description of RESST, record pointers are entered simultaneously with keys to form (key, record-pointer) pairs. The record-pointer cell (RP) is identical to the CE cell, except that the compare circuitry is not necessary.

3.3. Top COMPARE/EXCHANGE Circuit (TOPCE)

The TOPCE circuit is functionally identical to the CE circuit, with the minor addition of a *clear* capability. Following assertion of the clear signal, the entire array is supposed to contain null values. To achieve this, the CLEAR1 signal must cause the most significant bit of all keys to be set to "1" (by definition). Thus, the TOPCE circuit is simply a CE circuit with a pass transistor (controlled by CLEAR1) which gates Vdd into each bit.

3.4. Other Circuitry

There are a number of other circuits involved in the RESST implementation. These include circuitry for buffering the EXCHIN/EXCHOUT signals, generating various control signals, and buffering data on and off

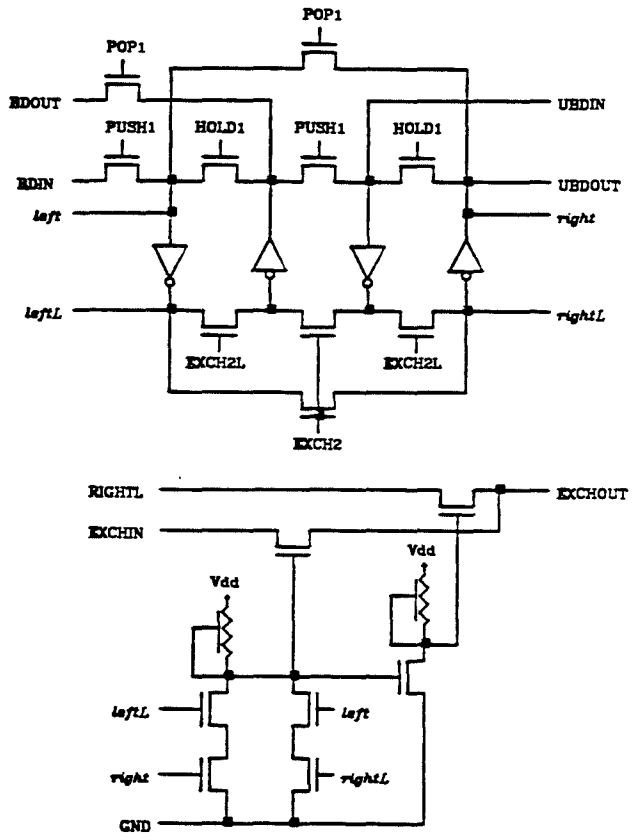

Figure 3.2. COMPARE/EXCHANGE Cell Schematic Diagram.

the chip. Circuit details and layouts for the complete RESST chip may be found in [Car82].

4. Performance Estimates

This section of our report discusses the performance of the RESST chip. We examined the timing of each of the clock phases using simple SPICE models of the salient portions of our circuit. (The MOS parameters used in these SPICE models were taken from page 51 of Mead and Conway [Mea80], with the polysilicon resistivity assumed to be 50 ohms per square.) We also briefly considered the power consumption of the design. The results of these performance estimates will be summarized here, with further details available in [Car82].

4.1. Timing Estimates

During phase 1, data is PUSHed or POPped from the RESST chip. This involves getting data on and off the chip, getting phase 1 clocked signals distributed throughout the chip, shifting data from cell to cell, and allowing the carry-chain-like EXCHIN/EXCHOUT signal to ripple from the least to the most significant bit (worst case). Our SPICE simulations indicated that the time required for these events is approximately 705 nanoseconds.

During phase 2, keys are compared and exchanges are conditionally made. This involves time to get the phase 2 signal to the PHI2SIG cell, time for the PHI2SIG cell to generate and distribute clocked EXCH2 and EXCHL2 signals, and time to actually perform the exchanges. Our SPICE simulations indicated that the time required for these events is approximately 480 nanoseconds.

Using pessimistic approximations and SPICE for circuit simulation, we obtained a width of 705 nanoseconds for phase 1 and a width of 480 nanoseconds for phase 2. The total cycle time for our present RESST design, then, is 1185 nanoseconds, or about 1.2 microseconds. With 8-bit keys and 8-bit record pointers, this is equivalent to a processing rate of about 1.67 megabytes per second. As discussed in [Car82], this speed could be enhanced by maximizing the use of metal in long signal runs or by utilizing the Mead and Conway scheme for optimal buffering [Mea80].

4.2. Power Estimates

We used Mextra and Powest [Ber82] to extract and estimate power consumption for our various cells. According to Powest, our circuitry (array plus clocked signal logic), which occupies an area of 3358 microns by 4466 microns, requires a worst-case DC power of 0.182 watts. Similarly, our pads, laid out around a 7000 micron by 7000 micron square chip perimeter, require a worst-case DC power of 0.633 watts. Thus, the worst-case DC power consumption of a RESST chip should be about 0.815 watts.

5. Other Enhancements

In reflecting on our work, several possible enhancements come to mind. First, rather than have separate CE and RP cell types, we could have stored record-pointer bits in CE cells. If they were stored in the least significant bit positions, they would appear as "insignificant bits" in sorting, coming into play only with duplicate keys. The advantages of doing this would be one less cell type and variability in the boundary between where keys end and record-pointers begin. The disadvantages would be increased RP cell size, increased EXCHIN/EXCHOUT signal delays, and somewhat increased overall power consumption.

Another possible enhancement involves the external RESST control circuitry, whereby an intelligent RESST controller could monitor the FULL pin on the leftmost RESST chip in a cascaded collection, and vary the clock cycle speed based on whether or not the leftmost chip is full [Car82].

6. Conclusions

This report discussed a VLSI implementation of a record-sorting stack. The implementation allows the sorting of n records, represented as (key, record-pointer) pairs, to be accomplished in $O(n)$ time. The design is cascadable so that the capacity of a single VLSI chip does not limit the amount of data which may be sorted.

The algorithm, a parallel version of the classic bubblesort algorithm, was described, the overall chip organization and data flow were presented, and detailed circuits, layouts, and timing analyses were given. It was shown that a RESST implementation can perform at disk transfer rates, making feasible its use as an enhancement to a database machine.

References

[Arm82] Armstrong, P., and Rem, M., "A Serial Sorting Machine", Computers and Electrical Engineering, Vol. 9, No. 1, Permagon Press, March 1982.

[Ber82] "Berkeley VLSI Tools", R. Mayo (ed.), Computer Science Division, University of California, Berkeley, 1982.

[Car82] Carey, M., Hansen, P., and Thompson, C., "RESST: A VLSI Implementation of a Record-Sorting Stack", Report No. UCB/CSD 82/102, Computer Science Division (EECS), University of California, Berkeley, April 1982.

[Chu80] Chung, K., Luccio, F., and Wong, C., "On the Complexity of Sorting in Magnetic Bubble Memory Systems", IEEE Transactions on Computers, Vol. C-29, No. 7, July 1980.

[Lei79] Leiserson, C., "Systolic Priority Queues", CMU Technical Report No. CMU-CS-79-115, Department of Computer Science, Carnegie-Mellon University, 1979.

[Mea80] Mead, C., and Conway, L., "Introduction to VLSI Systems", Addison-Wesley Publishing Company, 1980.

[Mir82] Miranker, G., Tang, L., and Wong, C., "A 'Zero-Time' VLSI Sorter", IBM Research Report, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1982.

[Muk72] Mukhopadhyay, A., and Ichikawa, T., "An n-Step Parallel Sorting Machine", Technical Report No. 72-03, University of Iowa, Iowa City, 1972.

[Muk81] Mukhopadhyay, A., "WEAVESORT - A New Sorting Algorithm for VLSI", Technical Report No. TR-53-81, University of Central Florida, Orlando, 1981.

[Tho82] Thompson, C., "The VLSI Complexity of Sorting", ERL Memo No. UCB/ERL M82/5, University of California, Berkeley, 1982.

[Ull80] Ullman, J., "Principles of Database Systems", Computer Science Press, 1980.