A VIRTUAL ZERO–TIME, MONOLITHIC SYSTOLIC SORTING ARRAY

C.L. Britton, Jr., M.N. Ericson*, and D.W. Bouldin***

Oak Ridge National Laboratory* and*

***The University of Tennessee*

ABSTRACT

A virtual zero–time monolithic sorting chip is described. The chip has a systolic array architecture and implements the "sinking sort" algorithm. The basic functional module of the systolic array is detailed and development techniques employed as well as functional simulation and results are presented. Lessons learned and educational significance of the development of this chip at a university are discussed.

INTRODUCTION

With the advent of the personal computer and low–priced, easy to use data bases, sorting has become a major function in computing and one of the most dominant operations in computer data processing. Banks and other businesses sort great volumes of transactions every day. As files lengthen and bit–fields widen, more time is required for sorting and time requirement is a major problem. One source estimates that over 25% of computer time is spent sorting[1].

****Operated by Martin Marietta Energy Systems, Inc. for the U. S. Department of Energy under Contract No. DE–AC05–84OR21400.*

Conventional methods of performing this function can generally be divided into two parts, internal and external sorting. Internal sorting requires dedicated processor time and is feasible only for small files. For larger files, special–purpose hardware such as external disk drives and tapes are needed to handle the increased data volume. Because sorting is usually only one part of a given program, its time–consuming execution can severely limit the information throughput of the entire program.

The alternative to internal sorting is external sorting, which requires the use of sort–specific external hardware that can perform sorting for the processor with minimum processor intervention. This procedure is sometimes referred to as coprocessing. Hardware specifically designed for external sorting performs two useful functions. First, it frees the processor for other program tasks. Second, if the hardware is properly designed, sorting can be done in a shorter time than if only the processor is sorting. This second point is very important, since hardware dedicated to a specific function and designed around an efficient algorithm will generally be faster than general–purpose hardware executing an algorithm in software. A monolithic implementation of an effective sorting algorithm will greatly improve processor efficiency in sorting. With the feature size of VLSI continually shrinking, it will be possible to get more and more complex designs onto a given area of silicon. This development means that, while large files can be sorted now, ever larger files can be sorted in the future.

This paper describes the design, fabrication, and testing of a monolithic chip that can greatly reduce processor time dedicated to sorting. The chip implements an algorithm that allows a virtual "zero–time" sorting environment; the only apparent time spent sorting is that spent writing the data to be sorted to the chip and reading the sorted data from the chip. To minimize the necessary area for a VLSI design, architectures composed of highly arrayable modules with minimized intercommunications are desired. The algorithm chosen for this implementation, the sinking sort algorithm[2], lends itself to this objective. It can be implemented as a systolic array of cells, each capable of performing the operations necessary for sorting.

Two primary areas were directly affected as a result of this development's taking place in an educational environment. First, the design methodology was directly influenced by that proposed by Mead and Conway[3]. Second, design efficiency in the classroom was maximized, as the University maintains current versions of all applicable VLSI design software tools.

SINKING SORT ALGORITHM

The sinking sort algorithm implements a "greatest in, first out" type of sorting in which input values are compared against previous inputs. Smaller inputs are allowed to "sink" through the sorting array so that values are stored in descending order.

The sinking sort algorithm can be implemented as a systolic array of identical cells in which each cell contains three registers, A, B, and C. The sorting process results in the larger of A and B being stored in B and the lesser stored in C. A two-phase, nonoverlapping clock controls the process. Figure 1 illustrates the operation of the algorithm when it is implemented as an array of these cells.

For illustration purposes, suppose the computer writes unsorted data to the cells in the order I = 1,4,7,6. The two basic operations necessary for the algorithm are to compare and to shift. The compare-and-shift process continues after the data have been entered into the array. If the B registers in the first column of cells are read, the largest entered value will be retrieved. Shifting and comparing must continue for the sorting to continue, even while data are being read out. If the column of B registers is read consecutively from first to last, the data will be retrieved with the greatest values first. The data must be read after Phase 1 (P1) of each consecutive clock cycle.

DESCRIPTION OF THE CHIP

As previously mentioned, the algorithm was implemented as a systolic array of identical cells on the chip. These are actually three distinct types of cells. The one–bit cell is the compare–shift entity. The chip also contains a set of edge–triggered flip–flops used for writing the data into the one–bit cells and another set for reading the data from the one–bit cells. A third set of cells, also composed of flip–flops, is used to implement scan–path testing. These sets will be discussed separately.

The one–bit cell, shown in Figs. 2 and 3, is the heart of the monolithic implementation of the sinking sort algorithm. The sorting process, illustrated in Fig. 1, is controlled by a two–phase, nonoverlapping clock. P1 loads register A, after which the comparison (CMP) of A and B takes place. Phase 2 (P2) loads the output register C through the multiplexer (MUX) with the lesser of the compared values. The carry–in, carry–out lines send the result of the comparison of a lesser bit cell up to the next most–significant–bit cell. As a result, the COUT of the most–significant–bit cell becomes the signal "A greater than B". AGB controls the function of the multiplexer, which allows the lesser value to be passed on to the next cell slice in the array. The sorting process must be initialized before data are entered by loading 0s into all B and C registers. Initialization will also load 0s into all A registers on the first P1 cycle, thus precluding the need to explicitly reset the A registers.

Data are read out by consecutively reading each B register slice on succeeding clock cycles. Buffered transmission gate R1 is used for this purpose. The gates are driven by a flip–flop shift register. A 1 is loaded into the input of the shift register and propagated through the flip–flops on each succeeding clock cycle. This process is started on the clock cycle following the last data byte written to the array allowing the reading to take place while sorting continues. Data may still be sinking through the array during this reading, but the reading stays one slice behind the last shifted data byte.

Because sorting continues as reading takes place, data must be continuously written to the array. The only data, however, that would not affect the results of the sorting are 0s. We therefore continue to write 0s into the array by resetting the input registers on each read pulse. This forces the data to appear to be 0s when in fact no data are being written to the chip. The 0s sink harmlessly to the end of the array without being read.

Buffered transmission gates S1 and S2 are driven by scan–path registers to allow the output of A and the input of C to be observed. The scan–path registers together with the read registers allow most of the chip to be accessible to the outside world, thereby increasing the observability of the internal array. The scan–path implementation allows a user to statically observe any of the test points or to follow a data byte down through the array. The read bus (KBUS) can be configured to do the same operation, because its structure allows part of the architecture to be used for scan–path testing.

A photograph of the chip is shown in Fig. 4. The read registers are shown along the top of the one–bit cell core, the scan–path registers along the bottom, and the input registers along the left side. The chip was a proof–of–principle implementation with a four–bit wide "nibble" I/O path and a depth of five nibbles. The I/O structure was designed to allow chips to be cascaded for increased depth.

FEATURES OF SORTPIPE II DEVELOPMENT

SORTPIPE II was designed with several goals in mind:

1. No minimum–size devices were used, allowing for improved speed and drive capability.

2. Transistor gates were wide on dynamic memory nodes to increase input capacitance. This design improves the charge holding capability of these nodes.

3. Transistor outputs used for driving buses were wide to ensure adequate drive and speed. The output KBUS has five gated buffers connected in parallel while the Scanbus (SCBUS) has ten. These buffers were designed to ensure adequate drive capability.

4. The signal paths were designed for a high degree of observability. The systolic structure of this chip naturally lends itself to scanpath testing, while the KBUS combined with the SCBUS allows most of the chip to be observed.

5. The one–bit cells were designed to be arrayed in all directions. This design allows shorter interconnection length.

6. The lowest level of leaf cells was designed with plugs (biased substrate ohmic contacts) in place so that plugs would be replicated along with the cells. This design makes plugging the entire circuit much easier.

SIMULATION AND RESULTS

Different simulations were performed on the chip at all stages of development. Using ESIM, each leaf cell was simulated at the lowest level of functionality to ensure proper operation. The one-bit cell and D flip-flops were the lowest-level collections of leaf cells simulated as a group. The cells were not used until fully functional at each level of simulation. The final simulations performed were on the core and on the chip with pads. Simulations on the core revealed undefined states resulting from certain input vectors. The problem cells were SPICE'd and found to be operating properly. The problem was finally traced to a "problem" with ESIM; the simulator was not recovering from a state that tied Vdd and GND together through the transistors.

The chip was fabricated in 3 micron, p-well CMOS and worked as desired. No problems with input vectors were encountered. The chip functioned properly down to a clock cycle time of 300 ns.

LESSONS LEARNED

Inevitably, a designer learns certain lessons during a complicated development, especially if the development is in an area new to him. SORTPIPE II was no exception. Of all the lessons learned, the following three stand out as most important:

1. Custom cells allow small size, but they do not allow many changes far into the design phase. SORTPIPE II was designed mainly with small leaf cells, which perform such general functions as flip-flops, inverters, and registers. The comparator was the only "customized" cell in the development. The comparator was smaller than if it had been made up of small leaf cells, but changes were harder to make.

2. It is easy to plug leaf cells because they usually contain few transistors. This condition allows plugs to be easily inserted into the design. Plugs then get replicated as often as the leaf cell.

3. Simulators are to be taken with a grain of SPICE. High–level simulators are useful for testing functionality of leaf cells and, to a great extent, the functionality of the entire circuit. If different simulators do not agree, the final authority, short of waiting for the chip to be returned, is SPICE or RELAX. These programs will allow accurate functional verification of the circuit.

4. Because development was carried out within an educational environment, conditions were well suited for the design of this innovative architecture. As in industry, there was indeed pressure to finish within an allotted time period. Unlike industry, there was little if any financial pressure so that risky design decisions could be justified. Also, a wider variety of design tools was available from the University than probably would have been available at a given company.

CONCLUSION

A zero–time monolithic sorting chip has been described, and the algorithm implemented on the chip and its systolic architecture have been presented. The techniques used for development as well as test structures implemented on the chip were discussed. Lessons learned and educational significance of the development of this chip at a university were discussed.

ACKNOWLEDGMENTS

The authors would like to extend their appreciation to the following for their work on the first SORTPIPE architecture: M. A. Abidi, G. A. Armstrong, P. L. Butler, A. Perez, R. J. Sharp, M. A. Shell, and R. A. Todd.

REFERENCES

1. D. E. Knuth, *The Art of Programming, Vol. 3: Sorting and Searching*, Addison–Wesley, Reading, Mass., 1973.

2. D. W. Bouldin and M. A. Abidi, *A Zero–Time VLSI Sorter*, The University of Tennessee internal report.

3. Carver Mead and Lynn Conway, *Introduction to VLSI Systems*, Addison Wesley, Reading, Mass., 1980.

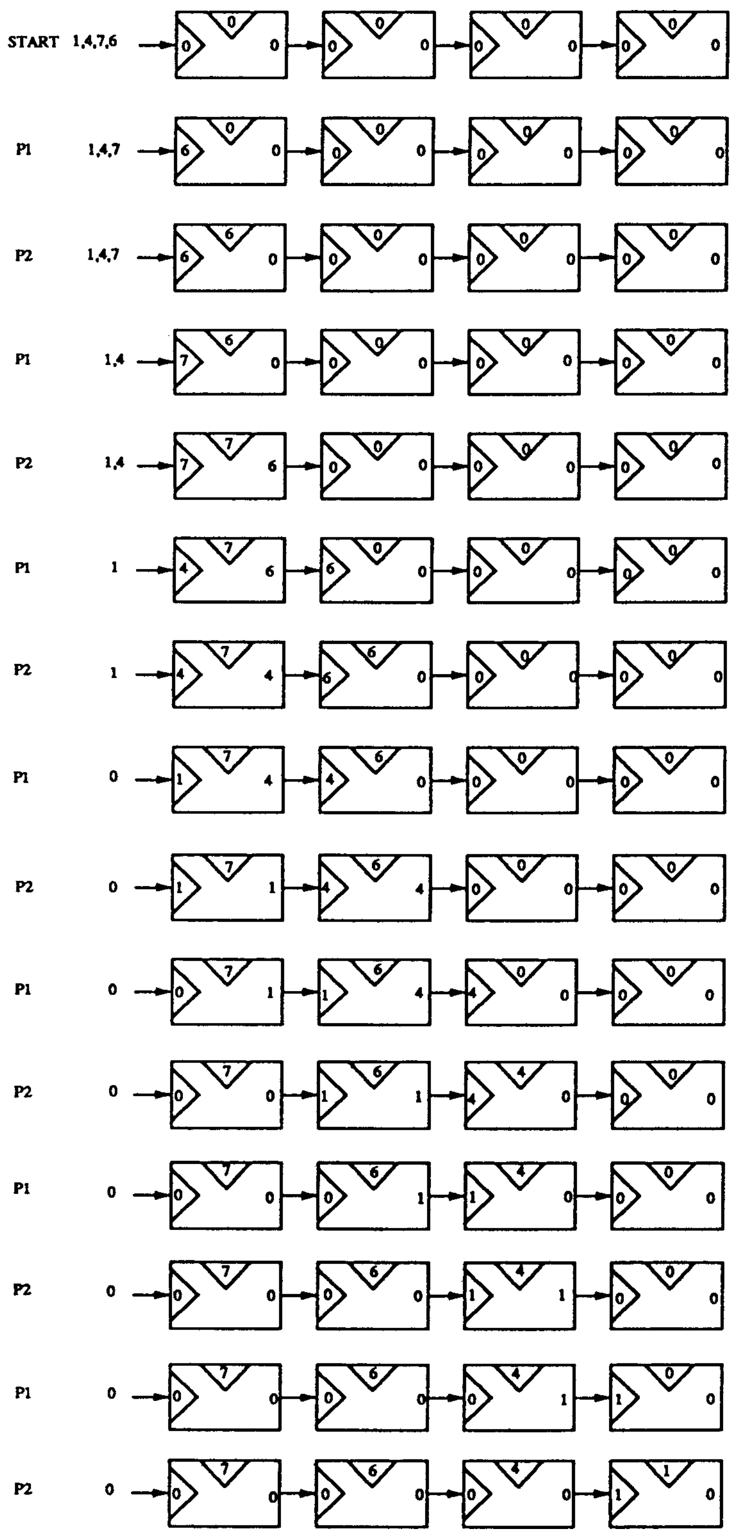

FIG. 1. Sinking sort algorithm.

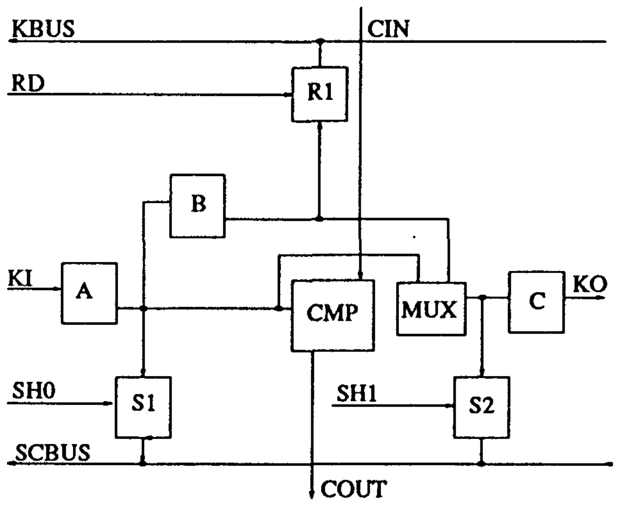

FIG. 2. Block diagram of a One-bit Cell.

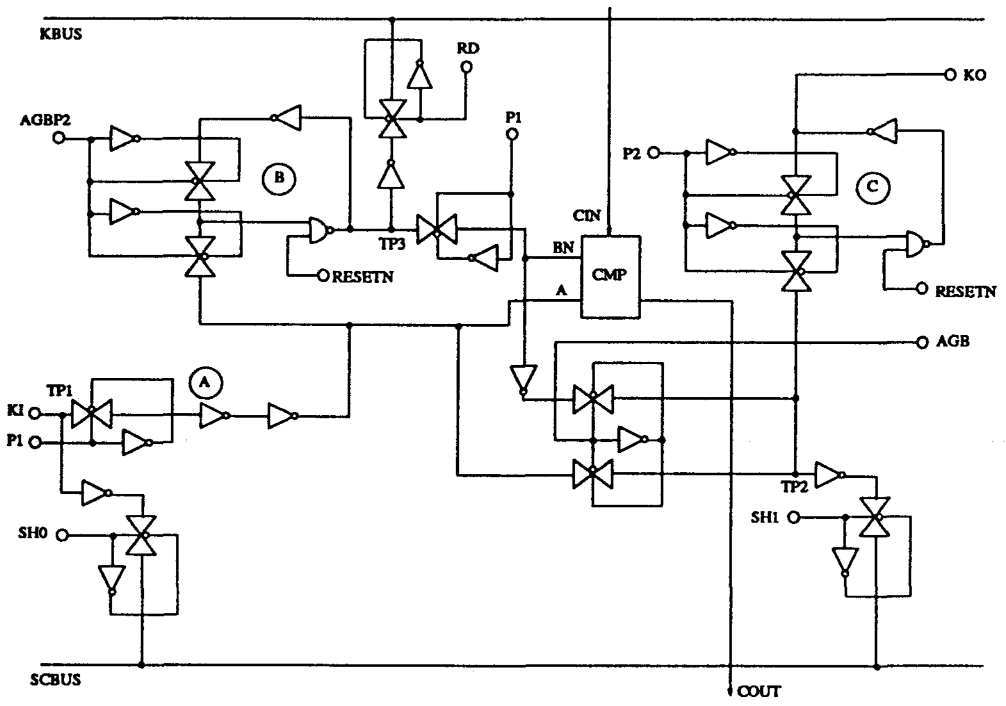

FIG. 3. Schematic diagram of a One-bit Cell.

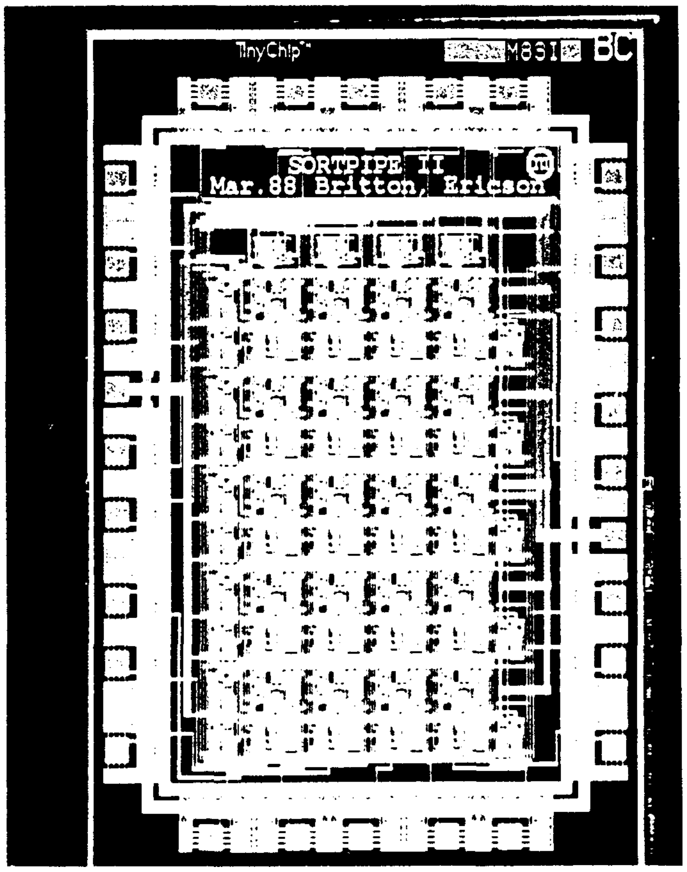

Figure 4. Photograph of SORTPIPE II