VLSI Design
1994, Vol. 2, No. 3, pp. 259–265
Reprints available directly from the publisher
Photocopying permitted by license only

An Improved Data Flow Architecture for Logic Simulation Acceleration

A. MAHMOOD
Washington State University at Tri-Cities, Richland, WA 99352

J. HERATH
Drexel University, Philadelphia, PA 19104

A. JAYASUMANA
Colorado State University, Fort Collins, CO 80523

(*Received August 31, 1991, Revised April 11, 1992*)

The high degree of parallelism in the simulation of digital VLSI systems can be utilized by a data flow architecture to reduce the enormous simulation times. The existing logic simulation accelerators based on the data flow principle use a static data flow architecture along with a timing wheel mechanism to implement the event driven simulation algorithm. The drawback in this approach is that the timing wheel becomes a bottleneck to high simulation throughput. Other shortcomings of the existing architecture are the high communication overhead in the arbitration and distribution networks, and reduced pipelining due to a static data flow architecture. To overcome these, three major improvements are made to the design of a classical data flow based logic simulation accelerator. These include:

1) A novel and efficient technique for implementing a pseudo-dynamic data flow architecture to increase pipelining.
2) Implementation of a modified distributed event driven simulation algorithm.
3) Localized processors for fast evaluation of small primitives.

Key Words: *Simulation accelerator, Logic simulation, Event driven simulation, Distributed simulation, Data flow*

1 INTRODUCTION

The parallelism in logic simulation is irregular in nature (especially in the case of event driven simulation), thus a data flow architecture is a good candidate for the simulation accelerator. In a data flow computer [1], there is no single control of execution of a program like a program counter in a Von Neumann architecture. Instead, the instructions are enabled when they receive their required data operands. The data flow programs thus implicitly draw out the concurrent operations according to the availability of hardware resources. For logic simulation, the programming on a data flow computer is further simplified when implementing an event driven simulation, since the implementation of event driven simulation already requires a data flow graph to be generated for the circuit under simulation.

Some of the popular commercial logic simulation accelerators based on the data flow architecture include Megalogician [2] from Daisy Systems (now Dazix) and the ZSE [3, 4] by Zycad. The Megalogician uses a classical static data flow architecture for event driven logic simulation. The drawbacks of this architecture are elaborated in the next section. The ZSE (Zycad Simulation Engine—LE series) by Zycad uses parallel busses for communication, instead of multi-stage arbitration and distribution networks, in a data flow design. These busses impose an upper limit for achieving a maximum throughput.

The purpose of this paper is to identify the inherent problems in the architecture of a classical static data

flow based logic simulation accelerator and to improve upon these. The classical design will be explained in the next section and the throughputs of the improvements made will be compared to this classical design.

2 CRITIQUE OF THE CLASSICAL DATA FLOW ARCHITECTURE FOR LOGIC SIMULATION ACCELERATION

The architecture of a classical static data flow based event driven logic simulation accelerator is shown in Figure 1. The host computer initializes the memory cells according to the elements in the circuit under simulation. It also downloads the input test vectors and receives output signals after processing. The arbitration and distribution networks are self-routing buffered delta networks. The timing wheel block implements the functions of a timing wheel as in a conventional event driven software simulator [5]. It contains sequential slots each of which emulates a simulation time step. The parallel events in a given time step are indicated by placing the events in the partitions of a time slot. The timing wheel increments to the next time slot (simulation time step) when all partitions have been handled in the current (active) slot.

In the conventional data flow based hardware accelerator, the timing wheel is the foremost bottleneck to the system throughput as the scheduling of every circuit element has to be controlled by it. A multiple bank implementation of the timing wheel can improve the performance, however, this gain may be limited due to the unbalanced distribution of events in the different timing wheel banks. Another shortcoming of the static data flow architecture is the lack of pipelining for sequential stream of instructions. This is because in a static data flow machine, only one token is allowed to exist on the input arc of an instruction. Hence simulation performance can be severely affected if a circuit contains many sequential

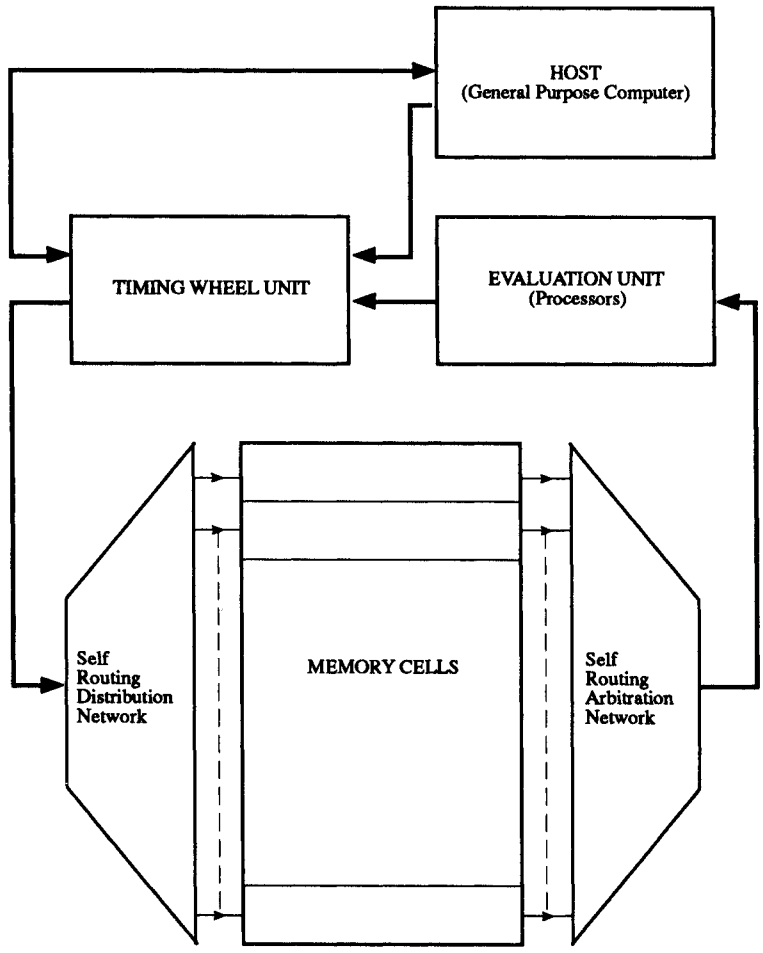

FIGURE 1 Architecture of a Data Flow Accelerator for Logic Simulation.

paths relative to the number of parallel paths. One more drawback of the data flow based hardware accelerator is the overhead in the evaluation of primitives. Two passes through the routing networks are required for elements with a change in output; i.e., one pass for evaluation of output and the other for distribution of result packets to the fanout elements.

The static data flow architecture for logic simulation acceleration is modified to improve upon these drawbacks in Section 3. In order to compare the improvements in the architecture to the conventional design, a detailed static data flow accelerator model is simulated which has a total of 64K memory cells, each of which can contain a logic element with a maximum of four inputs and a maximum fanout of three. There are a total of 256 processors connected to the memory cells by a $4^8 \times 2^8$ buffered delta arbitration network. In order to improve parallelism, the timing wheel is implemented as multiple banks. A 256×256 cross-bar switch connects the processors to 256 timing wheel banks which are in turn connected to the memory cells by a $2^8 \times 4^8$ buffered delta distribution network.

3 IMPROVED DATA FLOW ARCHITECTURE FOR LOGIC SIMULATION ACCELERATOR

Three major improvements have been made to the design of a conventional data flow based simulation accelerator. These include a novel pseudo-dynamic data flow architecture for improved pipelining, implementation of the modified distributed simulation algorithm of Chandy and Misra [6–7], and the design of localized processors in the memory cells for fast evaluation of small primitives. The design of each of these modifications is explained in the following subsections.

3.1 Pseudo-Dynamic Data Flow Architecture

A dynamic data flow architecture [1] can improve the pipelining of data by allowing multiple data tokens to exist on the inputs of an instruction. The dynamic architecture utilizes a matching unit to forward the appropriate input tokens for the evaluation of a primitive. The drawback of this dynamic architecture is that the matching unit requires the use of an associative memory which is very expensive. Further, the associative memory introduces its own bottleneck as all packets have to pass through it. Hence the dynamic data flow architecture is not practical for logic simulation acceleration.

In order to improve the pipelining of data in logic simulation, an efficient pseudo-dynamic data flow architecture is implemented. This pseudo-dynamic architecture does not utilize an associative matching unit but instead, it provides buffers (queues) at the inputs of each memory cell. Thus, multiple tokens (upto the buffer depth) can be allowed to exist on an input. The buffer depth in the current design is five. The self routing networks in the pseudo-dynamic architecture are designed such that there is always a predetermined path from a source memory cell to a destination cell. This feature guarantees that the tokens generated by a source will always maintain proper order. Hence, an associative match is not required in the input buffers for correct ordering of inputs.

A demand driven scheme is implemented to avoid deadlock and the overflow of input buffers. A memory cell will transmit an output to its fanout list only if a demand exists from each of the elements in the fanout list. Every cell maintains a demand count for the elements in the fanout list. The initial demand count is five (maximum buffer depth) for the elements in the fanout list. A demand is sent to the fanin element when a data value is consumed in the input buffer of an element. The demand count for the element in the fanout list is decremented whenever a value is sent to it. To further improve the event driven simulation scheme, the input buffers check for redundant data values i.e., identical logic values on an input are absorbed. Thus, these buffers are referred to as RCIBs (Redundancy Check Input Buffer).

3.2 Distributed Event Driven Simulation Algorithm

A distributed simulation algorithm has been developed by Chandy and Misra [6–7]. In this algorithm, each token carries a time stamp, known as the time tag, which indicates the time up to which the logical value is valid. Hence by using this algorithm, the timing wheel is no longer needed as the data values carry the time information themselves. The distributed simulation algorithm can run into deadlock situations for circuits containing feedback loops. There are two popular approaches to deadlock avoidance: one is a conservative scheme proposed by Chandy and Misra and uses NULL messages; the other is an optimistic scheme, known as Time Warp [8]. For

hardware accelerators, the conservative approach to deadlock avoidance is more economical.

A modified version of the distributed algorithm of Chandy and Misra using NULL messages to avoid the deadlock has been implemented. This algorithm is modified to include event driven simulation both on the inputs and the outputs, and improve pipelining by using RCIBs. Whenever an element has at least one token on each of its inputs, it is enabled for execution and uses the data on each input port to evaluate the output. The input tokens with the lowest time tag are absorbed while the other input tokens remain on their input port. If after absorbing the lowest time tagged inputs, an input buffer becomes empty, the output signal should be passed to the elements in the fanout list regardless of a change in its output. This is necessary to avoid deadlock in the simulation. If the input buffers are not empty, then the output is propagated only if there is a change in its value implementing the event driven scheme.

Two processes are being executed concurrently in each memory cell. One is the RCIB process which handles receiving of the input tokens in the buffer and performs redundancy checks. If an input is discarded or absorbed, a demand signal is sent to the corresponding fanin element. The second process is the evaluate process which computes the output tokens and implements the event driven scheme on the output. The pseudo code for the modified distributed algorithm using RCIBs is described below. The code is hard-wired into each memory cell controller in the accelerator.

RCIB PROCESS:

```
If a token arrives
  then
    If logical value of incoming token = previous token
      then
        .Previous token = incoming token
        .Discard the previous token
        .Send demand signal to the source element
      else
        .Insert the incoming token in the buffer queue
```

EVALUATE PROCESS:

```
If all input buffers have at least one token
  then {
    .Compute the logical output based on the inputs in the front of the queue.
    .Absorb the input(s) with the lowest time tag.
     (The computed output known as the new output has
     a time tag = time tag of absorbed input(s) + delay of the element)
    .Send demand signal(s) to the input(s) absorbed

    If (logical value of new output ! = present output) & (transmit_flag = FALSE)
      then {
        .Transmit present token if demand > 0 for all the fanout elements and decrement the demand
         count.
        .Present token = new token
        if (at least one input buffer is empty)
          then {
            .Transmit present token if demand > 0 for fanout elements and decrement the demand
             count.
            .Transmit_flag = true }
          else
            .Transmit_flag = FALSE }
            else {
```

```
        If at least one input buffer is empty
          then {
            .Present token = new token
            .Transmit present token to the fanout list if demand > 0 for fanout elements and decrement
            the demand count.
            .Transmit_flag = TRUE }
          else {
            .Present token = new token
            .Transmit flag = false }
    }
}
```

3.3 Localized Processors

In the conventional design, the evaluation of a circuit element requires the transmission of an operation packet to the processing unit. This extra pass through the routing networks can reduce the throughput considerably. However, by providing each basic memory cell with a small lookup table, the evaluation of output can be done locally without having to go through any routing network. For a four input gate with four state simulation (high, low, undefined, or high impedance), a total of $4^4 = 256$ entries are required in the lookup table. Using two bits to represent the four states, it requires 64 bytes for the lookup table in each cell. The lookup table in each cell is initialized by the host according to the type of gate it will contain during a simulation. Since memory availability is improving in size and cost each year, the design of localized processors with a memory lookup table is practically feasible.

The overall design of the improved data flow based simulation accelerator is shown in figure 2. In the improved design, out of the total 64K memory cells, 4K are of complex type such as flip flops, multiplexers etc. These cells are connected to 64 external processors through a $4^6 \times 2^6$ buffered delta arbitration network and a $2^6 \times 4^6$ buffered delta distribution network. The remaining 60K cells are basic cells which have localized processors in the form of a lookup table. Since the output from a basic cell (e.g., a gate) may need to be routed to a complex cell (a function level primitive) or vice versa, the basic cells and the complex cells can communicate to each other through a $4^8 \times 2^8$ buffered delta arbitration network and a $2^8 \times 4^8$ buffered delta distribution network.

4 RESULTS

Five versions of the accelerator were simulated to study the effects of different modifications. These include:

- model A: The static data flow accelerator based on a timing wheel as shown in figure 1.
- model B: The improved data flow architecture implementing the distributed simulation algorithm but without using any buffers on the inputs.
- model C: The improved data flow architecture implementing the distributed simulation algorithm using RCIB's, but not the event driven output implementation.
- model D: The improved data flow architecture implementing the distributed simulation algorithm with five deep input buffers without redundancy checking capability and event driven output implementation.
- model E: The improved data flow architecture implementing the distributed simulation algorithm with RCIBs and event driven output implementation.

The design of similar units is kept identical for the different models. As an example, all models use exactly the same kind of routing networks i.e., a $4^8 \times 2^8$ size buffered delta arbitration network and a $2^8 \times 4^8$ size distribution network. All of the five architecture models were simulated in detail at the behavioral level using the "C" language. A basic gate delay of 10 nanosecond was assumed in the modelling of all accelerator designs. Table I gives the performance of different circuits on these five models. All circuits were simulated at the gate level (except the 4 bit binary counter and the 8 bit microprocessor employed behavioral flip flop models) in the simulations in Table I using the localized processors within the cells in models B, C, D and E. Figure 3 shows the average simulation times of the six different circuits from Table I on the five accelerator models. It can be seen that the model E which uses

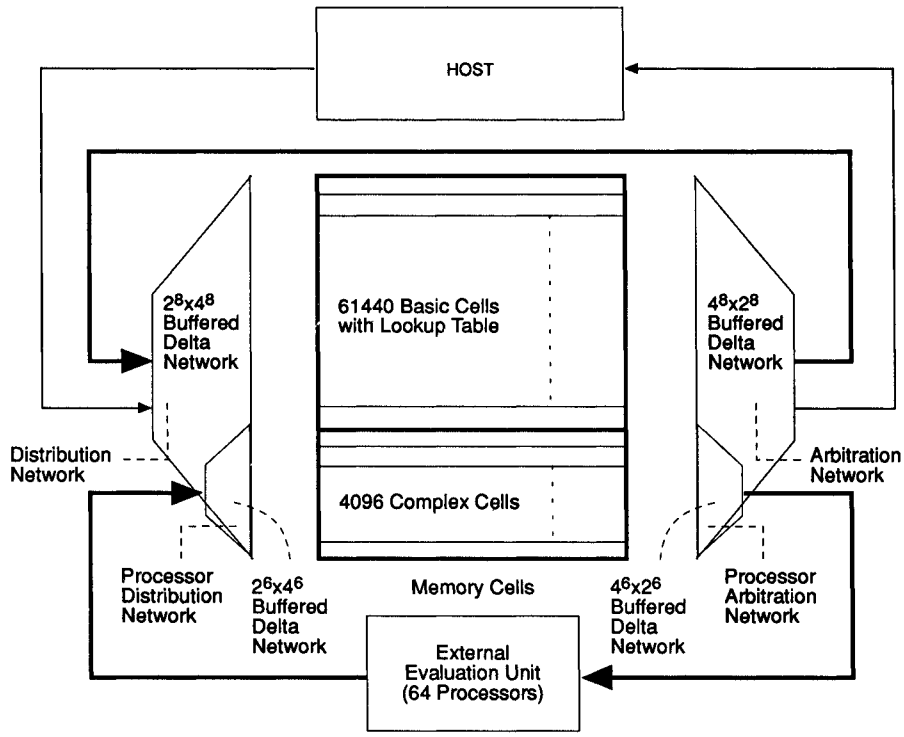

FIGURE 2 Design of the Improved Data Flow Accelerator.

distributed simulation algorithm with RCIBs on the inputs and event driven output, has the best overall performance. An improvement of more than 300% is obtained over the conventional timing wheel based static data flow model (model A). An improvement of about 35% comes from the use of localized processors in the execution time.

5 CONCLUSIONS

Three main improvements were made to a conventional implementation of a data flow based accelerator for logic simulation. These include a novel pseudo-dynamic data flow architecture for improving the pipelining, the implementation of the modified

TABLE I
Comparison of Different Accelerator Models (Times are in micro-seconds)

CIRCUIT	NUMBER OF GATES	MODEL A	B	C	D	E
Chain of 64 Inverters	64	305	285	185	185	185
4 Bit Binary Counter	53	370	198	162	132	127
4x4 Array Multiplier	192	527	238	176	155	149
8x8 Array Multiplier	768	868	340	265	223	211
8 Bit Serial Multiplier	642	785	315	208	176	172
8 Bit Microprocessor	3266	1621	705	564	450	434

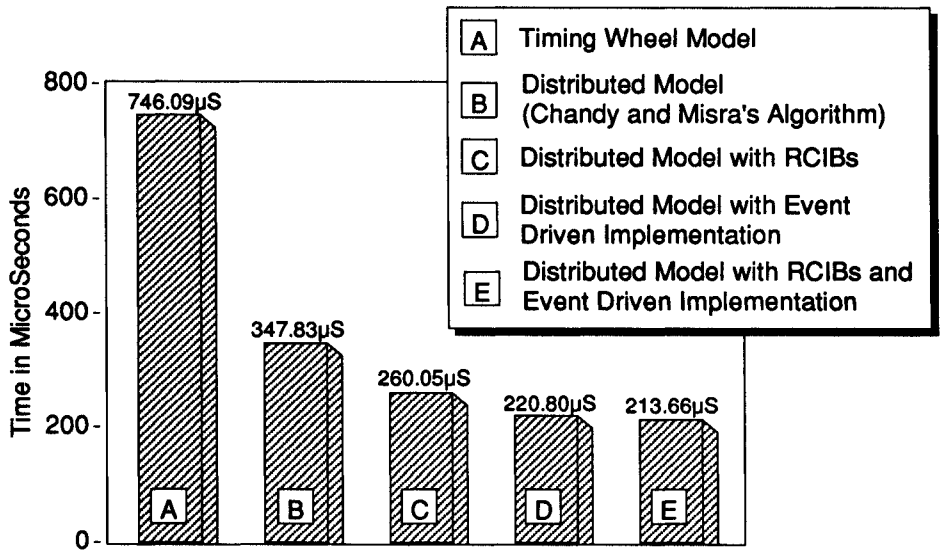

FIGURE 3 Comparison of Different Accelerator Designs.

distributed event driven simulation algorithm, and the design of localized processors for fast evaluation of small primitives. Variations of the accelerator model were simulated to study the design improvements. The simulation results on several different circuits indicate that the modified design is on the average three times more time efficient than the conventional timing wheel based design. The scheme with RCIBs and event driven implementation on the output yields the highest performance. The RCIBs provide absorption of redundant input tokens and also improve the pipelining of data thereby giving the most efficient simulation.

References

[1] Hwang, K., and F.A. Briggs, "Computer Architecture and Parallel Processing," NY: McGraw-Hill, 1984, pp. 732–763.

[2] Siegel, S., and M.E. Kaszynski, "The Design of a Logic Simulation Accelerator," *VLSI Systems Design,* Oct. 1985, pp. 76–80.

[3] Paseman, W.G., "Data Flow Concepts Speed Simulation in CAE Systems," *Computer Design,* Jan. 1985, pp. 131–140.

[4] Catlin, G. and B. Paseman, "Hardware Acceleration of Logic Simulation Using a Data Flow Architecture," Proceedings of the 1985 IEEE International Conference on Computer Design (ICCD), 1985, pp. 130–132.

[5] Mahmood, A., "An Extensible Multilevel Logic Simulator with Model Abstraction Capabilities," *Progress in Computer Aided VLSI Design,* (editor G. Zobrist), Norwood, NJ: ABLEX Publishing Corp., 1989, pp. 137–190.

[6] Chandy, K.M., and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering,* vol. SE-5, no. 5, September, 1979, pp. 440–452.

[7] Chandy, K.M., and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Communications of the ACM,* vol. 24, no. 11, April, 1981, pp. 198–206.

[8] Jefferson, D.R., "Virtual Time," *ACM Transactions on Programming Languages and Systems,* vol. 7, no. 3, July 1985, pp. 404–425.

Biographies

AUSIF MAHMOOD received the B.Sc. degree in Electrical Engineering from the University of Engineering and Technology, Lahore, Pakistan in 1979, and the M.S. and Ph.D. degrees in Electrical Engineering from Washington State University, Pullman, in 1982 and 1985 respectively.

He is currently an Assistant Professor in the Department of Electrical Engineering, Washington State University at Tri-Cities. His research interests include simulation of VLSI systems, parallel computer architectures, and digital signal processing.

J. HERATH has been a Japanese government researcher from 1981–1987. From 1987 to 1989, he was with George Mason university. He, is currently working as an assistant professor at Drexel University. His research interests include application-specific high performance dependable computing systems, and parallel and distributed processing.

A. JAYASUMANA is currently an associate professor at Colorado State University, Fort Collins. He holds a B.Sc. in Electronics from University of Sri-Lanka, M.S. and PH.D. degrees from Michigan State University, East Lansing in Electrical Engineering. His research interests are in computer-aided design of VLSI systems.