TECHNIQUES TO REDUCE SYNCHRONIZATION IN DISTRIBUTED PARALLEL LOGIC SIMULATION

Kai-Hui Chang, Wei-Ting Tu, Yi-Jong Yeh, and Sy-Yen Kuo
Dept. of Electrical Engineering, National Taiwan University, Taipei, Taiwan

ABSTRACT

As the complexity of chip designs increase, simulation time also elongates. Unit and variable delay simulation takes the most simulation time in IC design process; however, parallel processing performs inefficiently due to large amount of synchronization. In this paper, techniques to reduce the number of synchronization points in synchronous designs are proposed, and a partitioner to partition designs along flip-flop boundaries is also proposed so that these techniques can be employed on real designs.

1. INTRODUCTION

As the complexity of chips increase, longer and longer time is spent on simulation. Therefore, parallel logic simulation has been proposed to solve the problem. There are two kinds of overheads in parallel processing: synchronization overhead and communication overhead. Partitioning plays an important role to reduce these overheads, and several algorithms have been proposed to solve this problem. For example, K-L [1] and F-M [2] have been used to partition gate-level designs. However, they mainly focused on reducing communication overhead in gate-level designs. Recent research like Manjikian and Loucks [3] also took other issues like network performance into consideration, but several issues are still not discussed, such as methods to reduce number of synchronization in practical designs under various kinds of clocks.

Synchronization problem is most serious in variable delay simulation, which usually occurs in post-synthesis simulation with Standard Delay Format (SDF) file annotated back. It takes the most time to simulate in the Integrated Circuit (IC) design flow. However, parallel simulation performs the worst in this level. It is because there are far more synchronization points than other levels of abstraction due to delays in gates and wires, and there is much less computation between synchronization points.

The most widely used and cheapest parallel processing environment is computers connected by Ethernet. Therefore, this paper focuses on characteristic of Ethernet. Two ways to reduce number of synchronization are proposed. One way is to synchronize at clock edges; the other way is to partition along flip-flop boundaries.

2. BACKGROUND

2.1 Levels of Abstraction

Top-down design flow is the prevalent methodology used in current IC design process. The abstraction levels for logic design can be illustrated in Figure 1 [4].

Abstraction	Primitives	Simulators
Instruction level	Machine instructions, chips	Verilog, VHDL
Functional level	Functional units (ALU, CPU)	Verilog, VHDL
Register transfer level (RTL)	Register, counter, MUX	Verilog, VHDL, HILO, THOR
Gate level	Gate, flip-flop, memory cell	Verilog, VHDL, HILO, THOR
Switch level	Ideal transistor (switch)	Verilog, VHDL, ESIM, COSMOS
	Transistor, resistor, capacitor	Verilog, VHDL, RSIM, RNL
Circuit level	Resistors, capacitor, current, voltage	SPICE, CAZM

Figure 1. Abstraction levels for logic design.

From empirical studies on real designs, it is found that for each level of detail the total execution time grow by a factor of ten [5][6].

There are different timing models to model the temporal behavior of circuits:

Zero-delay assumes that every change of a signal's value will affect its output immediately.

Unit delay assumes that every change of a signal's value needs exactly one time unit to become available.

Variable delay provides the most flexible way to simulate elements.

2.2 Parallel Simulation Scheme

Methods of parallelizing logic simulation can be categorized into two major approaches: synchronous and asynchronous. In this paper, Avery Design System's SimCluster [7], a synchronous parallel processing environment, is used.

2.3 Synchronization Models

There are at least three kinds of synchronization models:

Simulation backplane: It allows models to be accurate to basic time unit level. This provides the time-unit to time-unit accuracy but with very heavy overhead. This model

uses delta-cycle synchronization and guarantees the most detailed timing accuracy.

Bus transaction model: It allows models to be accurate at each clock cycle level. This is consistent with the RTL description. What happens at time-unit level is not important at this level as long as there is no ordering dependency to affect results at cycle-level. This model uses cycle-based synchronization.

Data transaction model: At this level, it will synchronize at a transaction level. For example, in a computer model, the CPU and DMA work in parallel. When a DMA is instructed to transfer a block of data to memory, there is no more interaction with the CPU. Each transaction may last from a few instructions to thousands of instructions. This model uses transaction-based synchronization.

3. TECHNIQUES TO REDUCE SIMULATION OVERHEAD

3.1 Characteristics of Ethernet

Communication architecture has a significant impact on the performance of distributed parallel processing. In Ethernet, it takes much longer time to send a one-byte packet than to send an extra byte in a packet. It means that transmitting the same amount of data with a large number of small-sized packets will substantially degrade performance.

3.2 Principles to Reduce Overheads on Ethernet

The following principles are used as guidelines to design synchronization protocols and partition algorithms for an Ethernet distributed simulation environment:

1. All port changes should be accumulated and be written in a socket write.
2. Number of synchronization should be reduced as many as possible, while number of ports between partitions is not so important. Therefore the partitioner has more freedom to duplicate or arrange instances in order to reduce synchronization and balance workload without concerning too much on connection between partitions. It is different from traditional partitioning algorithms which minimize communication between partitions.
3. Synchronization using higher level of abstraction should be used whenever applicable to reduce number of synchronization and increase concurrency.

3.3 Synchronize at Clock Edges

Determining the next synchronization point in advance is called “conservative lookahead.” Its theory and impact on performance have been studied by several researchers such as Nicol [8], Peterson and Chamberlain [9].

Most of current designs are synchronous, and it provides good hints to determine the next synchronization point in advance. In synchronous designs, there are flip-flops between two blocks of circuits, and values propagate to another block only at clock edges. Therefore in synchronous design, we can use cycle-based synchronization instead of delta-cycle synchronization to avoid redundant synchronization points. To achieve this, we must determine the time of the next clock edge in advance. The two approaches to predict the next clock are described below.

3.3.1 Presimulation

Presimulation requires simulating the design in single process or multiple-processes with delta-cycle accuracy first and write clock changes to a Value Change Dump (VCD) file. Then this information can be used to determine the next clock time for cycle-based simulation. This approach is effective in real designs because only the clock generator needs to be presimulated. Futhermore, for clocks that are not periodic, like spread-spectrum clocks, this approach is the only way to know the next clock time in advance.

3.3.2 Dynamic Phase Locker

Dynamic phase locker tries to lock periodic clocks during simulation. Once a clock pattern repeats more than three times, the clock is locked and the same pattern is used to determine the next clock time. And then cycle-based synchronization will be started. If the period of a clock changes, it will be unlocked, and the simulation will return to delta-cycle synchronization immediately. This approach is more flexible and is adaptive to clock changes. However, it only works if the clock is periodic.

4. CLOCK PARTITIONER

Gate level simulation with variable delay is the most time-consuming one in the IC design process. However, the performance of distributed simulation is usually poor in these designs because complicated timing will result in lots of synchronization points, and the workload between synchronization points will be low. In this situation, the synchronization overhead is too much and will slow simulation down. But if the design is partitioned at flip-flop boundaries, only synchronization points at clock edges are necessary. The number of synchronization points will then be reduced significantly.

In order to have correct simulation results, the simulation should synchronize at the following time points.

1. When clock changes, for clock signal propogation.
2. The time that the output of the flip-flop changes.

If more than one type of flip-flops is used at flip-flop boundary, there may be more than one propagation delay and the ports may change at different times. In this case, synchronization points should be added whenever the port values change. However, using the largest delay is enough for most designs.

Mueller-Thuns et al [10] proposed an approach specifically for sequential logic circuits which produces

partitions such that only latch outputs cross partition boundaries. Manjikian and Loucks further address issues in workload balance and communication. However, their approaches require that the storage devices on the partition boundary be known in advance. In flattened gate-level designs, such information may not be available. Assuming that all clocked storage devices are at partition boundary is also incorrect because storage devices are also used for other purposes. Therefore an algorithm to find flip-flop boundaries is proposed in this paper, called clock partitioner.

4.1 Algorithm

i, *j* are instances in the design. Three functions are used to return the ports of an instance: input(*i*) returns its inputs, output(*i*) returns its outputs, and ports(*i*) returns its ports.

U is the set of instances not partitioned. Initially, it contains all the instances in the design.

P contains instances added to the partition and is initially empty.

S is the set of storage devices in the design.

PORT is the set of instances that will become ports at partition boundary. It is initially empty.

Q is a queue used during processing. Initially, it contains an instance selected randomly from *U*.

"Connect" is a function which returns true if the two instances in its arguments are connected.

"Mark" marks a flag on the port.

"Dequeue" returns the first instance in the queue and remove it from the queue.

The algorithm is given in Figure 2.

```
while Q ≠ null do
  i ← dequeue(Q)
  if i ∈ S then
    if i ∉ PORT then
      PORT ← PORT ∪ {i}
      if !marked(output(i )) then
        Q ←Q ∪ {j ; connect(input(i ), j ) && j ∈ U
                && j ∉ Q}
      else if i ∈ PORT && marked(port(i )) then
        PORT ←PORT–{i}
        P ←P ∪ {i}
        U ←U - {i}
        Q ←Q ∪ {j ; connect(port(i ), j ) && j ∈ U
                && j ∉ Q}
  else
    P ←P ∪ {i}
    Q ←Q ∪ {j; connect(port(i), j)&& j ∈ UP && j ∉ Q}
```

Figure 2. Clock Partitioner Algorithm

A partition bounded by flip-flops will be produced by this algorithm and is stored in *P*. To partition the whole design, this algorithm can be repeated until *U* is empty. Flip-flops on the partition boundary will be added to the partition that connects to their inputs.

In this algorithm, each instance is processed once or twice so its time complexity is O(n).

4.2 SDF Partitioner

In IC design flow, delays between gates and wires are stored in SDF files and are annotated back to the design before simulation. A partitioner for the SDF files is implemented so that each partition still has correct delays.

5. EXPERIMENTAL RESULTS

5.1 Synchronize at Clock Edges

The experimental design is a 128 * 128 gate-level multiplier. It is composed of four Wallace tree multipliers and three ripple-carry adders. The design is three-stage pipelined as Figure 3 shows.

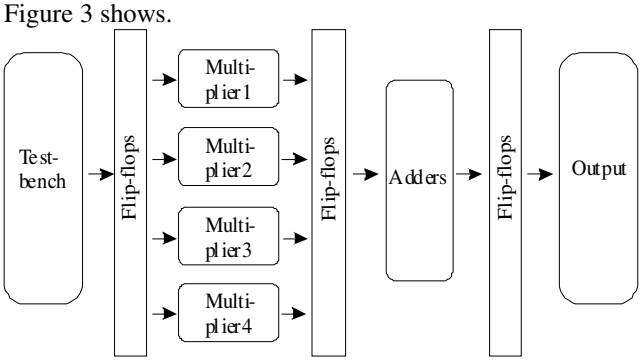

Figure 3. Design of Benchmark.

Simulation environment:
Simulator: VCK
Platform: Redhat Linux 8.0
Simulation is distributed to five AMD XP 1.8GHz computers with 512M RAM connected by local area network (100Mbps Ethernet).

Result:
Delta cycle synchronization result is given in Figure 4, and synchronization at clock edge result is given in Figure 5.
In the figures, Var-delay means variable delay.

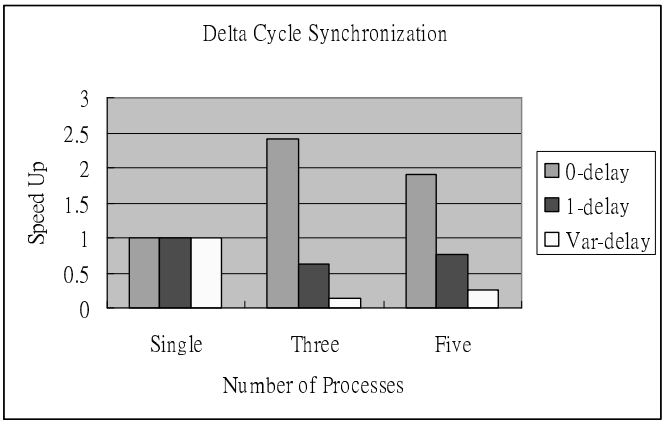

Figure 4. Delta Cycle Synchronization Results.

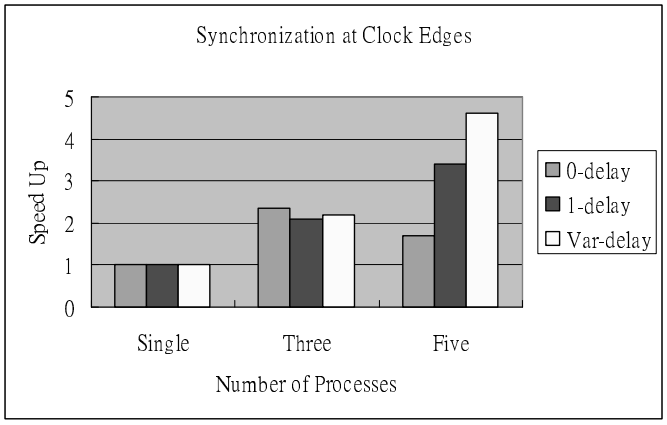

Figure 5. Cycle Based Synchronization Results.

From Figure 4, we can find that the speed up of three or five partitions is better than that of a single process. However, the speed up of five partitions is less than that of three partitions because the communication overhead acts against the parallel processing speed up. Next, both unit-delay and variable-delay simulations with more than one partitions do not have any speed up due to large amount of communication overhead. Third, variable delay simulation is slower than unit delay simulation because of extra synchronization produced by the complex timing in variable delay simulation.

From Figure 5, we can find that both unit-delay and variable delay simulations are benefited from parallel processing and the speed up is almost proportional to the number of processes.

From the experimental results, we can see that communication overhead plays an important role in the performance of parallel processing. Complex timing models introduce too much synchronization overhead and make delta-cycle synchronization impractical to simulate designs with unit or variable-delay. Furthermore, synchronization at clock edges reduces communication overhead significantly and the speed up is almost linear to the number of processors used, which makes parallel processing applicable to real designs.

5.2 Clock Partitioner

Clock partitioner has been used on the design in Figure 3 after its design hierarchy is flattened. A total of 8832 instances were partitioned and five partitions with flip-flops as their boundaries were correctly found. The partition time is 2.9 seconds.

6. CONCLUSION

In this paper, techniques to determine lookahead synchronization time by exploiting clocks in synchronous designs are described. A partition algorithm with time complexity O(n) is also proposed to partition the design at flip-flop boundaries so that these techniques can be used. From the experimental results, it can be concluded that for unit and variable delay models, parallel distributed simulation synchronized at delta-cycle accuracy is impractical because of large amount of communication overhead. But our techniques can improve performance and make speed up almost linear to the number of processes. Techniques proposed in this paper can reduce simulation time significantly, especially in post-synthesis designs with variable delay. They are innovative ideas and will greatly save the simulation time.

7. REFERENCES

[1] W. Kernighan and S. Lin., An Efficient Heuristic Procedure for Partitioning Graphs, Bell System Technical Journal, 1970

[2] C. M. Fiduccia and R. M. Mattheyses, A Linear-Time Heuristics for Improving Network Partitions, *Proceedings of the 19th Design Automation Conference*, 1982

[3] Naraig Manjikian and Wayne M. Loucks, "High Performance Parallel Logic Simulation on a Network of Workstations,' *Proceedings of the Seventh Workshop on Parallel and Distributed Simulation*, 1993

[4] Gerd Meister, "A Survey on Parallel Logic Simulation," Dept. of Computer Science, University of Saarland, Germany, Sep. 1993

[5] L. Soulé, and T. Blank, "Statistics for Parallelism and Abstraction in Digital Simulation," *Proc. 24th Design Automation Conference*, 1987

[6] K. Wong, M. Franklin, R. Chamberlain, and B. Shing, "Statistics on Logic Simulation," *Proc. 23th Design Automation Conference*, 1986

[7] Avery Design Systems, *VCK/Simlib User's Guide, Rev. 1.1.0*, May 2002

[8] D. M. Nicol, "High Performance Parallelized Discrete-Event Simulation of Stochastic Queuing Networks," *Proceedings of Winter Simulation Conference*, 1988

[9] George D. Peterson and Roger D. Chamberlain, "Exploiting Lookahead in Synchronous Parallel Simulation," *Proceedings of Winter Simulation Conference*, 1993

[10] Mueller-Thuns, R. B. Saab, D. G., Damiano, R. F., and Abraham, J. A., "Portable Parallel Logic and Fault Simulation," *Proceedings of the IEEE International Conference on Computer-Aided Design*, 1989