D. Maclise, R.A. E. P. Becker

# MDCSIM: A COMPILED EVENT-DRIVEN MULTI-DELAY SIMULATOR


## YUN SIK LEE

Technical Report DA-6, VCAPP Laboratory
Dept of Computer Sci. & Eng., University of South Florida
Tampa, Florida 33620

# MDCSIM: A COMPILED EVENT-DRIVEN MULTI-DELAY SIMULATOR

**Yun-Sik Lee**
**Peter M. Maurer**
**Department of Computer Science and Engineering**
**University of South Florida**
**Tampa, FL 33620**

# MDCSIM: A COMPILED EVENT-DRIVEN MULTI-DELAY SIMULATOR

## ABSTRACT

This paper describes a complied event driven logic simulator which allows gates to have delays that are integral multiples of some basic time unit. The nets and gates of a circuit are compiled into a routines that perform the evaluation of gates and process events. These routines also manage the current timing wheel slot and insert events into the appropriate future time slots. A threaded code implementation is used to reduce execution time and space. Experimental results have shown a 26% improvement in execution time for compiled simulation over a standard event driven simulator.

# MDCSIM: A COMPILED EVENT-DRIVEN MULTI-DELAY SIMULATOR

## Introduction.

As the design of a circuit proceeds, it is necessary to simulate circuit's behavior more and more accurately. In particular, more and more accurate timing models are needed. During the final phases of the design it is usually necessary to deal with the delays of the individual elements more accurately than is possible with a unit-delay or zero-delay simulator. Recently there has been much renewed interest in compiled simulation, particularly because it promises to provide better performance than is normally provided by interpreted simulators[1-9]. Although there are many well-known compiled simulation algorithms, these are based on the zero delay or the unit delay timing models. These timing models do not provide an accurate model of the circuit's timing behavior. For some circuit elements, such as delay lines, multivibrators and inverters, delay is the essential nature of their function, and a reasonably accurate timing model is necessary to model their behavior.

This paper focuses on the multi-delay timing model, in which the delay of each gate is modeled as an integral multiple of some basic time unit. Delays may be the same for each instance of a particular gate type or different delays can be assigned to two gates of the same type. This model permits a more accurate circuit analysis than is possible with the unit-delay or zero-delay models. The algorithms used by MDCSIM are based on the threaded code model used by Lewis[4], while the internal structures are based on the the work of Wang[3]. The timing algorithm is the traditional timing-wheel algorithm originally described by Szygenda et. al.[10].

Szygenda et. al.[10] recognize several types of delay that could be modeled in a multi-delay simulation, among these are transport delay, which is the amount of time taken for changes in a gate's inputs to reach the gate's outputs, ambiguous delay, which are short intervals in which the gate's outputs are undefined, and rise-fall delay, which is the amount of time a signal takes to change from low to high and vice-versa.

At the present time MDCSIM models only transport delay. MDCSIM is a three valued simulator, so we could easily model ambiguous delay , and, to a certain extent, rise-fall delay as well. In our logic description language[11], the delay of each gate is provided by the circuit description, as illustrated in Figure 1.

```
abc:    circuit
        inputs      a,b,c
        and         (a,b),i1,delay=2
        or          (i1,c),delay=5
        endcircuit
```

Figure 1. A Circuit Description with Delays.

## 2. The Multi-delay model and Compiled Event Driven Simualtion.

Although the principles of multi-delay event-driven simulation are well known, we present them here for completeness. In general a gate will not be simulated unless one of its inputs changes value. For example, consider the circuit pictured in Figure 2, and suppose that the input A changes at time k. Gates G1 and G0 will be simulated at time k and will generate events that contain the new values of D and C at times k+1 and k+2 respectively. It is necessary to simulate both gates at time k so that the simulation of these gates will use the proper values of A, B, and C. The event containing the new value of D is processed at time k+1 and if the value of D changes due to this event, the gate G3 is

simulated at time k+1. However if the value of D does *not* change, G3 is *not* simulated at time k+1.

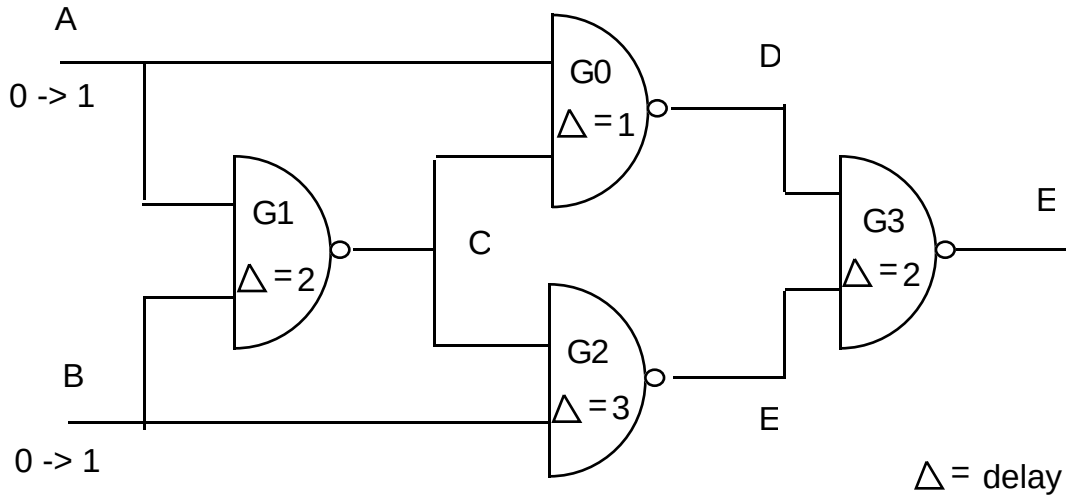


Figure 2. XOR logic with NAND gates

Our implementation contains a gate queue, and several event queues which are organized as a timing wheel. The timing wheel is compiled to size, so no overflow queue is required. Two types of routines are generated by the compiler, gate simulation routines and event handling routines. There is one event routine for each net and one gate routine for each gate. When a vector is read, events for all changed inputs will be queued at time 0 as event elements in the timing wheel. The current time is set to 0, and the first event routine is executed. Event queue elements contain the new net value and the address of the event processing routine. Each event processing routine assigns its net value to its net, and if the net value differs with the previous one it also places gates that use the net into the gate queue. Finally, it jumps to the next event processing routine in the current time slot of timing wheel. The last event element in each time slot is a queue terminator routine, which jumps to the first gate handling routine in the gate queue. The gate processing routine simulates a gate with current net values, and adds one or more events to the timing wheel at appropriate future locations. The last element in the gate queue is a gate terminator routine. This routine checks the count of queued events and terminates if count is zero. If the count is not zero, then it advances the current time by one and branches to the first element in the event queue for the current time. When events are added or deleted from any event queue, a count of queued element is updated for termination purposes.

## 3. Implementation

In the event driven compiled simulation, the evaluation of gates and updating net values requires the execution of pre-generated routines for each gate and each net. These routines could be processed by a central scheduler which is responsible for performing sequencing, but such an implementation would require the routines to be accessed via some sort of subroutine call. However in gate level simulation, the overhead of tens of thousands stack operations and execution time to process the subroutine calls would consume an enormous amount of time, not to mention the space required. Therefore, we have chosen to use the threaded code model in much the same manner as Lewis [4] and Wang [3].

The output of our compiler is primarily C code with a few line of assembler code to implement routine addressing. A sample of the generated code (for the circuit pictured in Figure 2) is shown in Figure 3.

```
int ad[6], fg[5], eqitems;
struct event *twheel[maxdelay+1];

SCH:
        if (eqitems)
        {
                ptr_event = twheel[current_time];
                addr = ptr_event->proc;
                net_value = ptr_event->net;
                eqitems--;
                asm("movl    _addr,a0");
                asm("jra         a0@");
        }
        else
                return;
BLK0:
        fg[0] = 0;
        value = ~(A&C);
        ptr_event->net = value;
        ptr_event->proc = nad[3];
        index = ( current + 1) % (maxdelay+1);
        twheel[index] = ptr_event;
        eqitems++;
        goto gate_scheduler;

NBLK2:
        if ( C != net_value)
        {
                if ( fg[0] == 0)
                {
                        fg[0] = 1;
                        ptr_gate->addr = ad[0];
                        qhp = ptr_gate;
                }
                if ( fg[2] == 0 )
                {
                        fg[2] = 1;
                        ptr_gate->addr = ad[2];
                        qhp = ptr_gate;
                }
        }
        else
                goto SCH;
```

Figure 3. Compiled Code for Figure 2.

The generated code contains three major functions. The initialization procedure loads the net and gate handling routine addresses into the "ad" and "nad" arrays, and allocates a free pool of event and gate queue elements (all queues are implemented as singly linked

lists). The scheduler is implemented in MC68020 assmebly code, which is included into the C program with the built-in function "asm". The scheduler scans the timing wheel slot for the current time. If there is any event element, it fetches the address of event processing routine and jumps to its address. The new value of the net is placed in the global variable "net_value." The net handling routine, as shown in figure 3 for net C, checks the previous net value with new net value. If the values differ, the routine puts the fanout gates in the gate queue.

The gate routine, as shown in figure 3 for gate G0, contains the evaluation code and gate delay function code. It first removes the gate element from the queue and sets its flag to 0. (This flag is used to prevent putting any gate on the gate-queue twice.) It evaluates the gate and puts an event at the appropriate place in the timing wheel. Finally, the last element of gate queue advances the time to access the next element of timing wheel.

## 4. Experimental Results

Figure 4 summarizes some of the results we have detailed from comparative studies of various simulation techniques. The circuits listed are the ISCAS85 benchmarks which have become a standard for measuring the performance of logic simuators. Delays were added to each gate by randomly selecting a delay from 1 to 8 for each gate. Each circuit was simulated on 5,000 randomly generated vectors using the techniques described above, and a conventional multi-delay event-driven interpretive simulator of our own design. Since the three valued logic model is the more natural for event-driven simulation than the two-valued model, especially when evaluating the first input vector, both simulators use the three valued logic model. The values reported in Figure 4 do not include the time required to read vectors or print output. These figures were obtained by running the two simulators on a SUN model 3/280.

| Circuit | Compiled Simulat. | | Interpretive | |
|---|---|---|---|---|
| | Eval | Time | Eval | Time |
| C432 | 1276897 | 58.6 | same | 87.5 |
| C499 | 2766217 | 117.0 | same | 171.3 |
| C880 | 2009913 | 108.8 | same | 134.9 |
| C1355 | 7659897 | 317.6 | same | 460.6 |
| C1908 | 8451813 | 367.1 | same | 517.3 |
| C2670 | 9389592 | 490.7 | same | 606.3 |
| C3540 | 18443225 | 820.7 | same | 1035.9 |

Eval : # of gates being simulated
Time: CPU seconds to simulate 5,000 vectors

Figure 4. Experimental Results.

Figure 4 shows us that there is about 26% performance improvement in compiled event driven multi-delay simulation over interpreted simulation.

## 5. Conclusion

Compiled Multi-Delay event-driven simulation has been shown to provide a significant improvement in performance over conventional interpreted multi-delay simulation. The performance improvement has not been as spectacular as the improvements reported for the zero-delay timing models[9], however this is due to the nature of the timing-wheel algorithm. In this algorithm a significant amount of time is spent in manipulating queue elements. The compiled technique used here does not reduce this time. There is time saved in searching lists, decoding gate types, and in table look-ups, but comparatively less time is spent on these activities than on queue manipulation. We are still in the process of tuning the algorithms used by MDCSIM and expect to see further improvements in the future.

# REFERENCES


1. Peter M. Maurer and Z. Wang, " Techniques for unit-delay compiled simulation", 27th Design Automation Conference, 1990, pp. 480-484
2. Melvin A. Breuer and Arther D. Friedman, Diagnosis & Reliable Design of Digital Systems, Computer Science Press, Woodland Hills, CA, 1976
3. Z. Wang and Peter M. Maurer, " LECSIM : A Levelized Event Driven Compiled Logic Simulator", 27th Design Automation Conf., 1990
4. D. M. Lewis, " Hierarchical Compiled Event-Driven Logic Simulation," Proceeding of ICCAD-89.
5. Wang, L., N. Hoover, E. Porter and J. Zasio, "SSIM: A Software Levelized Compiled-Code Simulator," Proceedings of the 24th Design Automation Conference, 1984, pp. 473-478.
6. Bryant, R. E., D. Beatty, K. Brace, K. Cho and T. Sheffler, "COSMOS: A Compiled Simulator for MOS Circuits," Proceedings of the 24th Design Automation Conference, 1987, pp. 9-16.
7. Hansen, C., "Hardware Logic Simulation by Compilation," Proceedings of the 25th Design Automation Conference, 1988, pp. 712-715
8. Barzilai, Z., J. L. Carter, B. K. Rosen and J. D. Rutledge, "HSS -- A High Speed Simulator," IEEE Transactions on Computer-Aided Design, Vol. CAD-6, No. 4. July 1987, pp. 601-617.
9. Chiang, M., and R. Palkovic, "LCC Simulators Speed Development of Synchronous Hardware," Computer Design, Mar. 1, 1986, pp. 87-91.
10. Stephen A. Szygenda, David M. Rouse and Edward W. Thomson, " A model and implementation of a universal time delay simulator for large digital nets", in Spring Joint Computer Conference, 1970, pp. 207-216
11. P. Maurer, Z. Wang, C. Morency, A. Tokuta and N. Bhate, "The Florida Hardware Design Language," *Proceedings Southeastcon-90*, pp. 430-434.