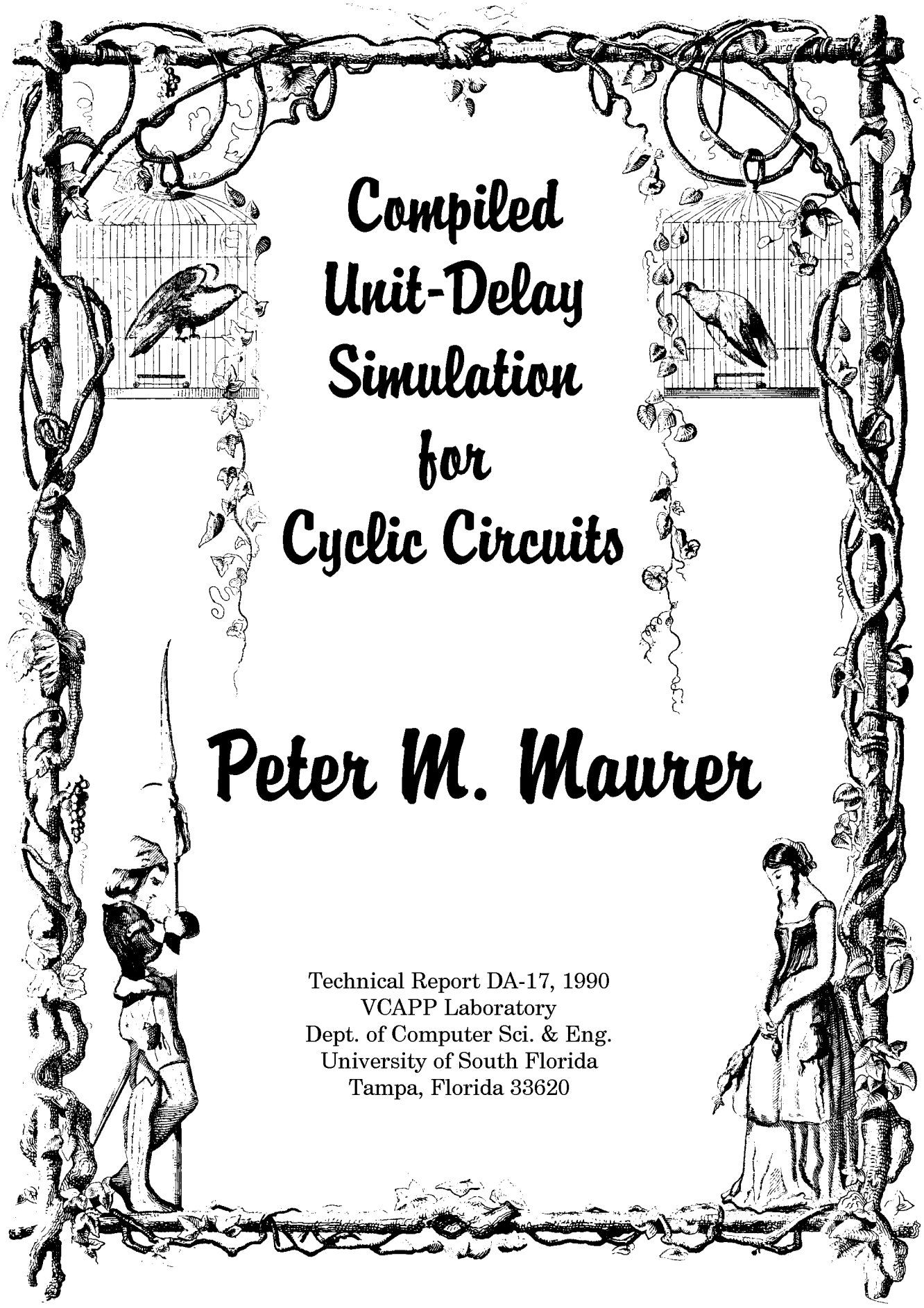


Compiled Unit-Delay Simulation for Cyclic Circuits

Peter M. Maurer

Technical Report DA-17, 1990

VCAPP Laboratory
Dept. of Computer Sci. & Eng.
University of South Florida
Tampa, Florida 33620

COMPILED UNIT-DELAY SIMULATION FOR CYCLIC CIRCUITS

Peter M. Maurer
Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620

ABSTRACT

The convergence algorithm is capable of generating a compiled unit-delay simulation for any cyclic circuit. The algorithm analyzes the circuit structure to created a sequence of subsets T_0, T_1, ... , T_n. The subset T_k contains the gates that will be executed at time *k*. Because the number of gates in the circuit is finite, this sequence will eventually converge. When the convergence point is reached, the sequence is used to generate code for the circuit. Code is generated in two segments called the prefix section and the iterative section. Bounds on the lengths of the two segments are given in terms of the structure of the circuit.

COMPILED UNIT-DELAY SIMULATION FOR CYCLIC CIRCUITS

Peter M. Maurer
Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620

1. Introduction.

Compiled simulation of circuits at various levels has been the subject of much recent research[1-7]. Although compiled simulation tends to be much faster than interpreted event driven simulation[5], the traditional methods for generating compiled simulators tend to be less accurate than the techniques used by interpreted simulators[8]. For various reasons, the most important of which is performance, the loss in accuracy for compiled simulation has become an acceptable trade-off for many of today's VLSI circuits [5]. However, this is not to say that the more accurate techniques used by interpreted simulators are not useful for debugging VLSI circuits. If it were possible to use a more accurate simulation without sacrificing performance, the more accurate technique would certainly be preferred.

Traditionally, compiled simulators have been based on a zero-delay simulation model, while interpreted simulations have been based on a unit-delay model. Because the unit-delay model provides for a more accurate simulation than a zero-delay simulation, there has been much research focused on the problem of generating compiled unit-delay simulators[2][7][10]. These simulators are modeled after interpreted event-driven simulators, with much of the work being performed at compile time. However, other research has shown that it is possible to extend concept of levelized compiled simulation to handle unit-delay simulation[9]. In this paper, these techniques will be called *purely compiled* techniques to distinguish them from those based on event-driven simulation. (This term is admittedly somewhat inaccurate.)

In a levelized compiled simulation, every net in the circuit is assigned a level which corresponds to the maximum-length path between the net and the primary inputs of the circuit. Because levelization cannot be used on cyclic circuits without special precautions, existing purely compiled techniques are restricted to acyclic circuits. This restriction is not as severe as it first seems, because *synchronous* sequential circuits can generally be simulated using techniques similar to those used for acyclic circuits[8]. Nevertheless, if purely compiled unit-delay simulation is to be considered a serious competitor with event-

driven simulation, the restriction to synchronous circuits must be removed. The purpose of this paper is to present a method for generating a compiled unit-delay simulation for *any* circuit, regardless of whether it is cyclic, acyclic, synchronous or asynchronous.

2. Theoretical Background.

Conceptually, a unit-delay simulation can be viewed as a sequence of state transformations, where each state contains the value of all nets at a particular simulated time. The simulated times are consecutive integers, and the state transformations are logic equations that simulate the logical behavior of one or more gates. Figure 1 illustrates this concept.

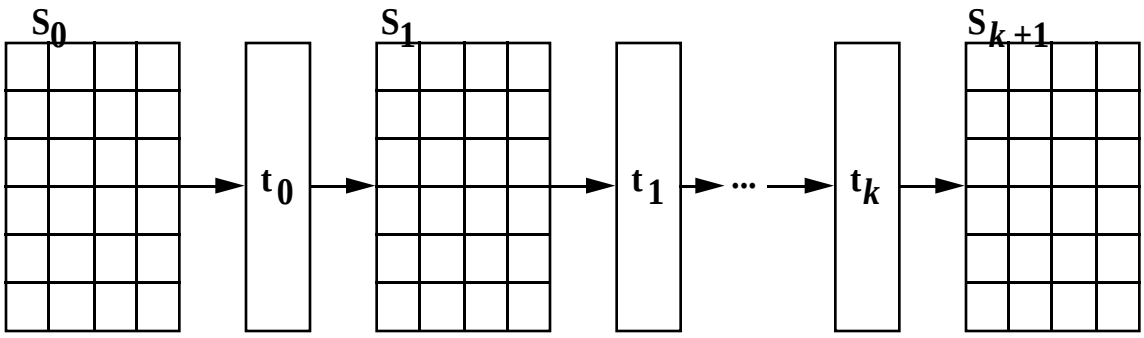


Figure 1. The State-Transition Model of Unit-Delay Simulation.

For each state transition, t_k illustrated in Figure 1, there is a minimal set of gates that must be simulated to correctly compute the new state S_{k+1}. Although there is a performance penalty for simulating too many gates in a transition, the correctness of the simulation will not be affected by simulating too many gates. In fact, it is possible to simulate *every* gate in the circuit for each transition, without affecting the correctness of the simulation.

There are two ways to improve the performance of a unit-delay simulation. One can reduce the number of gates simulated for each state transition, or one can reduce the amount of time required to simulate each gate. Interpreted event-driven simulation is quite successful at reducing the the number of gates simulated, but the simulation time per gate is extremely high. On the other hand, the PC-set method of compiled unit-delay simulation[9] reduces the amount of time necessary to simulate a gate at the expense of simulating more gates than necessary in each transition.

In the PC-set method, a set of potential-change times is calculated for each gate and each net in the circuit. This set, called the *PC-set*, contains the set of times at which a net

or the output of a gate can potentially change value. For example if a net has the PC-set {2,3,7}, the net may change value at times 2, 3, and 7, but its value at times 4, 5, and 6 will be identical to its value at time 3. The PC-set is calculated by first assigning the set {0} to each primary input of the circuit, and then traversing the network in levelized order. Once all the inputs of a gate have been assigned PC-sets, the PC-set of the gate is calculated by obtaining the union of the PC-sets of the gate's inputs and incrementing each element of the new set by 1. The PC-set of a net is equal to the union of the PC-sets of all gates that could potentially drive the net. Once all PC-sets have been computed, code is generated by generating one gate simulation for each element of the PC-set of a gate. Because generation of the PC-set depends on levelization, the PC-set method is restricted to acyclic circuits.

Code can be generated for the PC-set method in two ways which are called the gate-first method and the time-first method. Regardless of which method is used, one variable is generated for each element of each net's PC-set. These variables allow one to reconstruct the state-sequence pictured in Figure 1. In practice, only the portion of the state-sequence that corresponds to the set of monitored variables is reconstructed. In the gate-first method of code generation, the circuit is traversed in levelized order, and all code for a particular gate is generated at once. The code for a particular state transition is usually distributed over a wide area. In the time-first method, the code for a particular state transition is all generated at once, and state transitions are generated in sequential order. Existing implementations of the PC-set method use the gate-first method, because the circuit has already been levelized during the creation of the PC-sets, and generating code in time-first order would require an additional sort operation.

Although techniques for generating a compiled simulation for a cyclic circuit are well known[8], these techniques are based on a zero-delay simulation model. Briefly, the method for handling cyclic circuits is to identify the strongly connected components of the circuit, and implement the strongly connected component as the body of a "while" loop. A depth-first search algorithm is used to identify back-arcs, the back-arcs are broken, and the result is treated as an acyclic circuit. Within the body of the loop, code is generated in levelized order. The loop causes the simulation to be repeated until no back-arc changes value or until some predetermined limit is reached.

This simplistic approach does not work for the unit-delay model, as the circuit of Figure 2 illustrates. In a unit-delay simulation, the loop consisting of the gates B and E must be simulated twice for every simulation of the loop consisting of gates A, C, D, and E

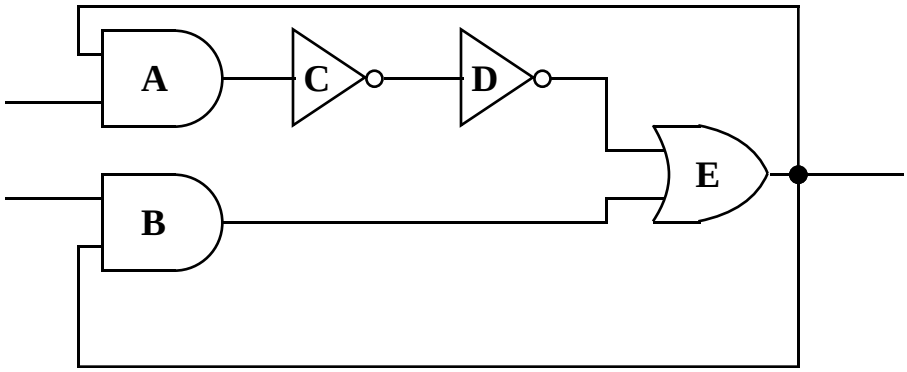


Figure 2. A Sample Cyclic Circuit.

The next section describes an algorithm that was developed to deal with the problem of varying length cycles.

3. The Convergence Algorithm.

The convergence algorithm is a time-first variation of the PC-set method, and is applicable to all circuits both cyclic and acyclic. The objective of the convergence algorithm is to generate the state transitions $t_0, t_1, \ldots, t_k \ldots$. This done by generating a sequence of subsets of gates, $T_0, T_1, \ldots, T_k, \ldots$. Each subset T_k contains all gates whose inputs could potentially change value at time *k*. The following is a formal description of the convergence algorithm.

1. Place any gate that is directly connected to a primary input into T_0.
2. Set *n* equal to 1.
3. Place any gate which is a direct successor of a gate in T_{n-1} into T_n.
4. If T_n is empty or equal to T_k for any $k<n$, then stop,
   otherwise, increment *n* by 1 and return to step 3.

If the circuit in question is acyclic, then T_n will be empty for some $n>0$, otherwise the algorithm will terminate when $T_n=T_k$ for some $0\le k<n$. Termination is guaranteed for cyclic circuits, because the number of gates in the circuit is finite which implies that the number of different sets, T_k, must also be finite. Note that if $T_n=T_k$ for some $0\le k<n$, then $T_{n+i}=T_{k+i}$ for all $i>0$. More precisely, the sequence of sets from T_k through T_{n-1} will repeat indefinitely. The sequence of sets T_0 through T_{n-1} is called the *convergence sequence* of the circuit.

The code generated for the circuit will consist of a prefix portion followed by an iterative portion, as illustrated in Figure 3. The prefix portion will be generated from sets

T_0 through T_{k-1}, while the iterative portion will be generated from sets T_k through T_{n-1}. At execution time, the iterative portion will be repeated until there is no change in the outputs of the gates contained in T_{n-1}, or until a predetermined limit is reached. When T_{n-1} contains a small number of gates, *m*, then the predetermined limit will be equal to 2^m+1 (assuming that each gate has only one output), otherwise an arbitrary value of 20 is used.

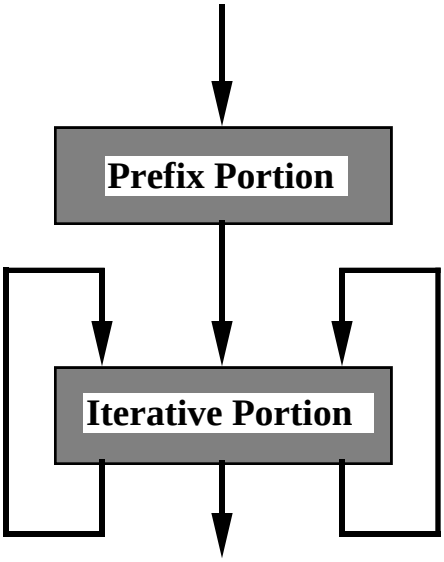
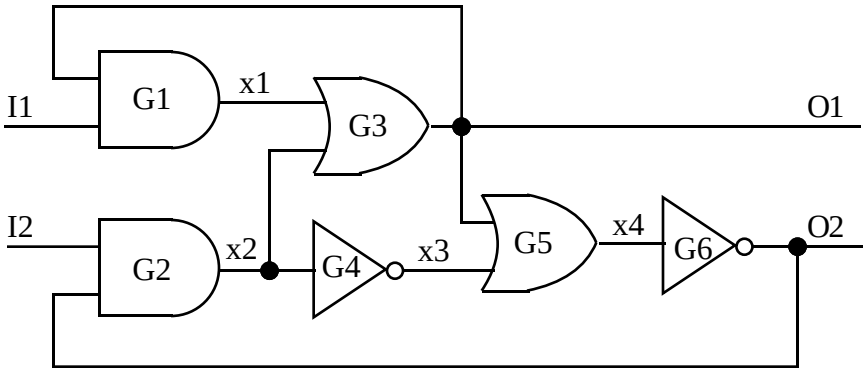


Figure 3. The Structure of the Convergence Algorithm's Generated Code.

Although it is possible to generate code in such a way as to maintain a complete history of each net for each input vector, the time-first method of code-generation allows a more space-efficient approach to be used. Since there is a clear division between gates simulated at different times, it is possible to print an output vector after each state transition is executed. If necessary, a one-vector history can be maintained for the purpose of hazard detection. When generating the code for a particular transition, it is necessary to avoid using net values that are computed within the transition. In other words, if Tj contains two gates G1 and G2, such that the signal x1 is an output of G1 and an input of G2, it is necessary to guarantee that the value computed for G1 is *not* used by G2 *in this transition*. This is necessary because the output of G1 must be delayed by one time unit and will not be available until the *next* transition. In some cases it is necessary to store the output of a gate in a temporary variable until the transition has been completely computed. Figure 4 illustrates a cyclic circuit and its associated convergence sequence.



Convergence sequence:

T_0 = {G1,G2}
T_1 = {G3,G4}
T_2 = {G1,G5}
T_3 = {G3,G6}
T_4 = {G1,G5,G2}
T_5 = {G3,G6,G4}
(T_6 = T_4)

Figure 4. A Circuit and Its Convergence Sequence.

The convergence sequence illustrated in Figure 4 can be used to generate code for the circuit. The generated code for the circuit of Figure 4 is illustrated in Figure 5. The code of Figure 5 is written in the C language. For those not acquainted with this language, the operators &, | and ~ are used to perform bit-level AND, OR, and NOT operations, while the operators && and || are the AND and OR connectives for conditions. The operator != is read "not equal to."

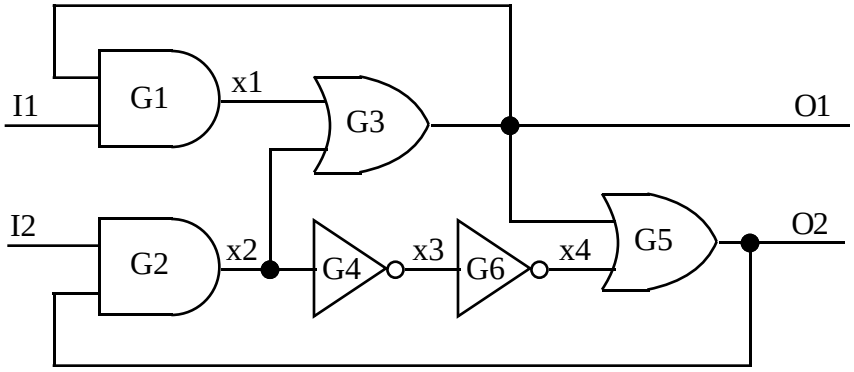
```
I1 = vector input;
I2 = vector input;

/* time 0 */
x1 = I1 & O1;
x2 = I2 & O2;

/* time 1 */
O1 = x1 | x2;
x3 = ~ x2;

/* time 2 */
x1 = I1 & O1;
x4 = O1 | x3;

/* time 3 */
O1 = x1 | x2;
O2 = ~ x4;

t1 = O1; t2= O2 ; t3 = x3; i = 0;
while ((t1 != O1 || t2 != O2 || t3 != x3) && i<9)
{
    t1 = O1; t2= O2 ; t3 = x3; i = i+1;

    /* time 3+i */
    x1 = I1 & O1;
    x2 = I2 & O2;
    x4 = O1 | x3;

    /* time 3+i+1 */
    O1 = x1 | x2;
    x3 = ~ x2;
    O2 = ~ x4;
}
```

Figure 5. Code Generated by the Convergence Algorithm.

4. The Length of the Convergence Sequence.

A this time we are primarily concerned with the behavior of the convergence algorithm on the strongly connected components of a circuit. Although a considerable amount of work remains to characterize the behavior of the convergence algorithm on various types of circuits, it appears that a more efficient simulation can be obtained by treating the strongly connected components of a circuit as independent components. Therefore, the analysis performed in this section will concentrate on circuits that consist of a single strongly connected component, such as that pictured in Figure 4.

A little experimentation with the convergence algorithm will show that a small change in the structure of a circuit can cause a big change in the convergence sequence. To illustrate this point, consider the circuit pictured in Figure 6. This circuit differs from that of Figure 4 in that the NOT gate G6 has been moved from the output of G5 to one of its inputs.



Convergence Sequence:

T_0 = {G1,G2}
T_1 = {G3,G4}
T_2 = {G1,G5,G6}
T_3 = {G2,G3,G5}
T_4 = {G1,G2,G3,G4,G5}
T_5 = {G1,G2,G3,G4,G5,G6}
(T_6 = T_5)

Figure 6. The Effect of Circuit Modifications on the Convergence Sequence.

Note that in Figure 6, the change in the circuit has resulted in lengthening the prefix section from four states to five, and shortening the iterative section from two states to one. To understand why this has occurred, it is necessary to examine the length of the cycles in

the two circuits. The circuit of Figure 4 the three cycles G1-G3, G2-G4-G5-G6, and G2-G3-G5-G6, while the circuit of Figure 6 has the three cycles G1-G3, G2-G4-G6-G5, and G2-G3-G5. In both cases, the length of the iterative section is equal to the greatest common divisor of the lengths of the cycles. In general, if a strongly connected circuit has cycles C_1, C_2, ... ,C_i of lengths l_1, l_2, ..., l_i, then the length of the iterative section will be no longer than the greatest common divisor of l_1, l_2, ... , l_i, and the length of the prefix section will be no longer than $ic+L$ where c is the longest simple path between any two gates, and L is the least common multiple of l_1, l_2, ... , l_i.

To see why this is true, it is helpful to think in terms of events arriving at various gates. Since the convergence algorithm runs at compile time, it is not possible to test for changes in a gate's output, so every gate simulation is assumed to generate an event. The initial set of events for the primary inputs are assumed to arrive at time zero. Note that if an event arrives at a gate G at time t, then the convergence algorithm will place G into T_t.

Now, suppose we are given a strongly connected circuit C containing two cycles C_1 and C_2 of lengths l_1 and l_2. Let G_1 be a gate contained in C_1 and G_2 be a gate contained in C_2. It is obvious that if an event reaches G_1 at time t then a new event will arrive at G_1 every l_1 time units thereafter. The first event will reach G_1 no later than time c. Since the circuit C is assumed to be strongly connected, every event generated by G_2 will eventually reach G_1, and vice-versa. The first event from G_2 will arrive at G_1 no later than time $2c$, and another will arrive every l_2 time units thereafter. Therefore, events will arrive at G_1 at times $a+il_1+jl_2$ for all $i,j \geq 0$ and some $a \leq 2c$, and let us assume for the moment that *all* event-times for gate G_1 are of this form.

The length of the iterative portion of the code generated for C is equal to the difference in time between successive events. For events described above, the difference will be of the form il_1+jl_2. Let g be the greatest common divisor of l_1 and l_2. It is obvious that g divides all integers of the form il_1+jl_2. The length of the iterative section will be equal to g if and only if there exists an integer p such that $il_1+jl_2=qg$ has a solution (i,j) in non-negative integers whenever $q>p$. Let $m_1=l_1/g$ and $m_2=l_2/g$. Dividing the equation by g gives the equation $im_1+jm_2=q$. Equations of this form are known to have an infinite number of integral solutions, all of which lie on the real line given by the equation $y=-(m_1/m_2)x+(q/m_2)$. Since the slope of this line is negative, only a finite segment of it will cross the first quadrant of the x-y plane. The segment crossing the first quadrant is of length L which is given by the following equation.

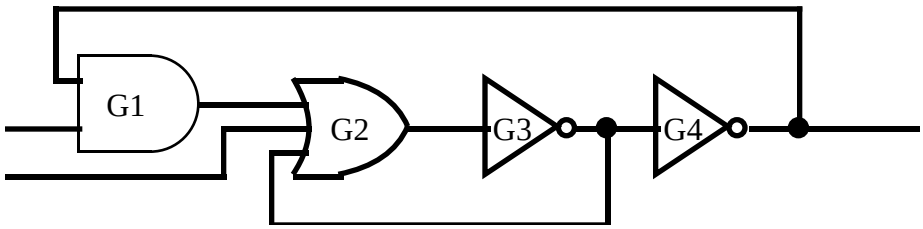
$$L = \sqrt{\frac{q^2}{m_1^2} + \frac{q^2}{m_2^2}}$$

Any equation of the form $im_1+jm_2=q$ is known to have at least one integral solution in every segment of length K given by the following equation.

$$K = \sqrt{m_1^2 + m_2^2}$$

If $q>m_1m_2$ then L>K and the equation will have at least one solution in non-negative integers. Therefore, after a sufficiently long time, a new event will arrive at G_1 every g time units. The behavior of the circuit will converge no later than the point where q becomes equal to m_1m_2. Setting il_1+jl_2 equal to m_1m_2g and substituting back into the original equation, the behavior of the circuit will converge no later than $a+m_1m_2g$. The least common multiple of l_1 and l_2 is equal to m_1m_2g, and as stated above, $a\leq 2c$. This argument can be extended to circuits with three or more cycles.

As the examples show, the bound on the length of the prefix section tends to be pessimistic. The bound on the length of the iterative section also tends to be pessimistic, as the circuit of Figure 7 shows.The circuit pictured in Figure 7 has two cycles, one of length 2 and one of length 4. The greatest common divisor is 2, but the length of the iterative section is 1.

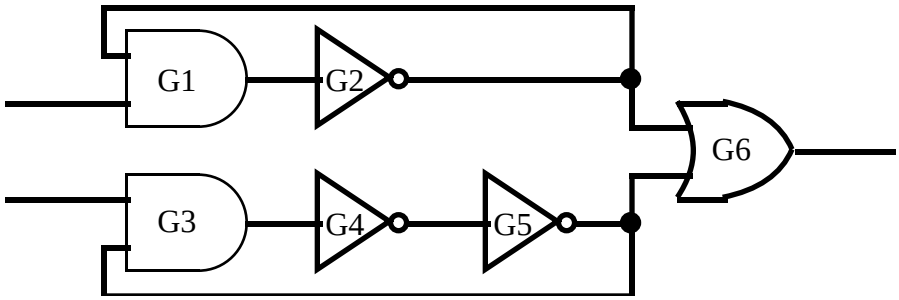


Convergence Sequence:

T_0 = {G1,G2}
T_1 = {G2,G3}
T_2 = {G2,G3,G4}
T_3 = {G1,G2,G3,G4}
($T_4 = T_3$)

Figure 7. An Iterative Section Smaller than the GCD.

It is important to note that the bounds derived above apply only to strongly connected circuits, as Figure 8 illustrates.



Convergence Sequence:

T_0 = {G1,G3}
T_1 = {G2,G4}
T_2 = {G1,G5,G6}
T_3 = {G2,G3,G6}
T_4 = {G1,G4,G6}
T_5 = {G2,G5}
T_6 = {G1,G3,G6}
T_7 = {G2,G4,G6}
($T_8 = T_2$)

Figure 8. A Non-Strongly Connected Circuit.

The circuit of Figure 8, which is not strongly connected, contains two cycles, one of length 3 and one of length 2. The greatest common divisor is 1, but the length of the iterative section is 6. Intuition suggests that it would be more efficient to treat the two cycles of Figure 8 independently rather than processing the circuit as a whole. More research is needed in this area.

5. Conclusion.

Although the convergence algorithm is capable of generating a compiled unit-delay simulator for any circuit, more research is needed to determine how well the simulators perform in relationship to interpreted event-driven simulations. Work is currently under way to measure the performance of the convergence algorithm on the ISCAS-89 sequential benchmarks[11]. When the circuit being processed is not strongly connected, there are many opportunities for optimizing the generated code. For example, when the circuit has the form pictured in Figure 9, the iterative section generated by the convergence algorithm will execute both strongly connected components repeatedly until they both converge.

Since SCC1 can be expected to converge before SCC2, there is almost certainly a more efficient method for organizing the simulation code.

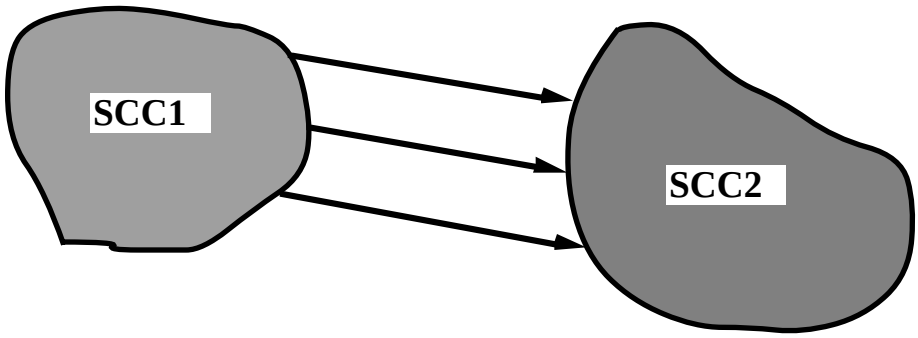


Figure 9. Two Consecutive Strongly-Connected Components.

Although much research remains, the convergence algorithm demonstrates that purely compiled unit-delay simulation is feasible for cyclic circuits, and provides a solid basis for much new research.

REFERENCES

1. 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.

2. 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.

3. Hansen, C., "Hardware Logic Simulation by Compilation," Proceedings of the 25th Design Automation Conference, 1988, pp. 712-715

4. 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.

5. Chiang, M., and R. Palkovic, "LCC Simulators Speed Development of Synchronous Hardware," Computer Design, Mar. 1, 1986, pp. 87-91.

6. Wang, Z. and Maurer, P. M., "Scheduling High-Level Blocks for Functional Simulation," Proceedings of the 26th Design Automation Conference, 1989, pp. 87-90.

7. D. M. Lewis, "Hierarchical Compiled Event-Driven Logic Simulation," *proceeding of ICCAD-89.*

8. M. A. Breuer, A. D. Friedman, "Diagnosis and Reliable Design of Digital Systems," Computer Science Press Inc., Rockville MD, 1976.

9. P. Maurer, Z. Wang, "Techniques for Unit-Delay Compiled Simulation,", *Proceedings of the 27th Design Automation Conference*, to appear June 1990.

10. Z. Wang, P. Maurer, LECSIM: A Levelized Event Driven Compiled Logic Simulator,", submitted for publication.

11. F. Brglez, D. Bryan, K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proceedings of ISCAS-89*, pp. 1929-1934.