A Parallel Switch-Level Simulator for Mixed Analog-Digital Circuit Simulation

B.-A. Molin and S. Mattisson
Dept. of Applied Electronices
Lund University
Lund, Sweden

Abstract

This paper presents a parallel event-driven switch-level simulator intended for medium-grain multicomputers. The chosen algorithm ensures that node voltages are always defined. This avoids problems with undefined node states and makes the simulator suitable for mixed analog-digital simulation. The active devices are modeled as piecewise-constant current sources, which gives high timing accuracy and enables simulation of other circuit technologies than MOS logic. Experimental results from simulation of a CMOS synchronous counter run on parallel workstations are presented.

1 Introduction

The advent of switch-level simulators for digital CMOS circuits in the beginning of the 80's greatly improved the capabilities of the computer tools available for design of VLSI circuits. Circuit simulators were too slow and gate level simulators did not give enough accuracy for dynamic logic and bidirectional circuit designs. The complexity of the circuits designed today, however, is so high that the response time for switch-level simulators is too long. Furthermore, the mix of analog and digital circuitry on the same chip accentuates the need for mixed methods for simulation of VLSI circuits.

In order to speed up simulations, faster computers are needed. Because of technology constraints and the high costs for ultimate improvements, traditional sequential computers will not be sufficient in the long run. Neither will vector processors, since circuit simulation does not vectorize well [1]. Instead, we direct our attention towards multicomputers, i.e. general-purpose computers connected by a message-passing network. Today, medium-grain multicomputers, consisting of up to 1024 nodes of typically a 32-bit microprocessor and some mega-bytes of memory, are commercially available [2]. The challenge of using multicomputers efficiently is to choose algorithms that inherently have high parallelism and that are suitable for a message-passing programming environment.

Parallelism may be exploited at several levels. On the system level, the waveform relaxation method provides concurrency. Different subsystems, or blocks, can be simulated separately with the resulting waveforms used for comparisons. On the block level, a parallel algorithm can utilize the inherent parallel activity in a circuit. In this paper, we are concerned with the solution of a digital subsystem by event-driven simulation. Here, the parallelism is limited by the sequential nature of the scheduler. The amount of concurrent activities in the simulator varies between different time points and reflects the amount of concurrency taking place in the circuit itself. However, the number of simultaneous events depends on the time resolution in the simulation. Hence, the choice of time resolution can be used as a means for speeding up simulation runs.

The simulator presented here will be used as a stand-alone parallel switch-level simulator. However, in a hierarchy of parallelism, the simulator could perform on a block that is part of a large digital circuit, or part of a mixed analog-digital circuitry. In the latter case, the simulator is aimed to be an extension to a concurrent circuit simulator [3].

2 The algorithm

Our main objectives for the choice of a simulator algorithm are: 1) the algorithm should be robust, i.e. convergence should be reasonably fast for all circuits of practical interest; 2) the simulator should be easy to adapt to a circuit simulator. The robustness is crucial for the reliability of the simulator. Furthermore, the convergence is primarily of greater importance than
the speed, since the parallelism will provide speed on large problems. Recently, a switch-level simulator based on current-limited switches was presented [4]. Here, conducting transistors are replaced by piecewise-constant current sources. The only other components allowed are capacitors to ground. Instead of dealing with node states, node voltages which are known to be piecewise-linear segments are calculated as the integral of the piecewise-constant node currents. Since the node voltages are always defined, no unknown states are propagated. Thus, this algorithm seems to meet our requirements.

Numerical integration

With this choice of algorithm, a breakpoint is defined as a change in device current. Between two breakpoints all currents in the simulated circuit are constant. This means that the state of the simulated circuit, i.e. the node voltages, is known. The voltage in a node is given by

\[ v_i = v_i^0 + \frac{1}{C_i} \int_{t_0}^{t_1} I_i \, dt = v_i^0 + \frac{(t_1 - t_0)I_{\text{int}}}{C_i} \]  

(1)

where \( I_{\text{int}} \) is the constant net current flow into node \( i \) between the two breakpoints \( t_0 \) and \( t_1 \), and \( C_i \) is the capacitance to ground at node \( i \).

If the integration is performed each time an active component changes current, equation 1 could be used, i.e. the integral is evaluated through a multiplication and a division. The information available for breakpoint calculations is for each of the component's connected nodes: a time point, the voltage at the time point, and the time derivative of the voltage at the time point. Let \( \dot{v}_i \) denote the time derivative of the voltage at node \( i \). Since the current into a node is constant between two breakpoints, \( \dot{v}_i \) is also constant and given by:

\[ \dot{v}_i = \frac{I_i}{C_i} \]  

(2)

Device Modeling

The current characteristics for an n-channel MOS-device in our model is shown in figure 1. The device is off for \( V_{GS} < V_T \). The on current is constant, and positive for \( V_D > V_S \) and negative for \( V_D < V_S \). The value of the current is determined by

\[ I_o = \frac{1}{2} k \frac{W}{L} (\alpha V_{DD} - V_T)^2 \]  

(3)

where \( k \) and \( V_T \) are process parameters, \( W \) and \( L \) are device geometry, and \( \alpha \) is a fitting parameter. Breakpoints are determined by the conditions \( V_{GS} = V_T \) and \( V_D = V_S \). Details about breakpoint calculations can be found in [5].

Since a device turns off at \( V_{DS} = 0 \), in contrast to the method presented in [4], we do not have any problems with oscillations due to ambiguity in the device current at \( V_{DS} = 0 \). On the other hand, the time step needs to be infinitesimally small when we leave this device state, which will cause generation of unnecessary events. To solve this problem we have introduced a hysteresis, shown as shaded areas in the current characteristics in figure 2. An on device is turned off when \( V_{DS} = 0 \). An off device is turned on when \( V_{DS} \) reaches one of the edges of the hysteresis region (depending on the sign of \( \dot{v}_i \)). Similarly, a hysteresis is introduced for the \( V_{GS} \) breakpoint. In this case, an on device turns off at the left edge of the hysteresis region, and an off device turns on at the right edge.
read_circuit_description;
partition_data_base;
send (data_base) to all processes;
while (not done) {
    receive_message (m);
    switch message (m) {
        case node_state:
            report_node_state;
            done;
            break;
        case scheduler_done:
            send_message (data_base) from host;
            unpack_data_base;
            while ((time < stop_time) and
                   (events_in_queue)) {
                get_next_events;
                send_message (events) to all processing
                nodes;
                sort_event_queue;
                for each processing node n {
                    receive_message (breakpoints) from n;
                    store_events;
                }
            }
            send_message (done) to host;
    }
}

Figure 3: Pseudo code for the host process.

Figure 4: Pseudo code for the scheduler process.

Program structure

The program consists of three pieces: a host process, a scheduler process, and a set of node processes. The host reads the input file, builds a data base, and distributes it to the scheduler and the node processes. Then the host deals with the user interaction. Figure 3 shows the pseudo code for the host process.

The scheduler handles the event queue. We have chosen a conservative algorithm, i.e. all generated events are collected before any computations on further time points are started. When new components are to be processed, or when any input signal changes, the scheduler activates the node processes where the affected components or nodes are located. Then it collects all generated events at the time point and stores them in the event queue. When the event queue is empty, or when the desired time window is solved, the scheduler sends an acknowledge to the host. The pseudo code for the scheduler process is shown in figure 4.

In sequential event-driven simulators, the event queue is organized with time wheels in order to minimize the queuing operations. In a parallel simulator where the scheduler is running on its own computing node, the most important thing is to deliver new events to the node processes as fast as possible. When waiting for incoming events, the scheduler has time to sort the event queue without much loss in speed. Hence, a linked list that is sorted during event calculations is used.

A node process is responsible for calculations on the part of the network that is distributed to the processing node. It receives messages either from the scheduler or from other nodes. When invoked from
the scheduler, the node handles the events to be processed at the current time, and returns the next event for each circuit node that is affected at the time.

The device current is a function of the component's node voltages at the current time. Since these are history, i.e. they can be calculated from $v$ and $\dot{v}$ at the last time point, a node process can calculate a component's current independently of other computing nodes activities if the old values are available locally. The next breakpoint, however, depends on $v$ in all neighbor nodes. Hence, breakpoint calculations cannot start until all currently active neighbor nodes are updated, which causes a synchronization between the computing nodes.

The node process performs as follows. First all input node voltages and changed device currents are calculated. The new currents determine $\dot{v}$, and consequently, neighbor node processes must exchange information about changed $\dot{v}$ values before any breakpoint calculations can be performed. When information from neighbor processing nodes is received, all breakpoints at the node are calculated. Then, the nearest breakpoint is reported to the scheduler. Figure 5 shows the pseudo code for the node process.

![Figure 5: Pseudo code for the node process.](image)

4 Results

A prototype version of the simulator is running on a multicomputer consisting of Sun4 workstations connected in a network. The program has been run on up to 8 parallel nodes.

The test circuit is a synchronous counter (figure 6), which consists of two T-flip-flops. The T-flip-flop (figure 7) has been chosen because it contains transmission gates and feedback loops, both known to cause problems for switch-level simulators. The flip-flop is run for five clock cycles, with a time resolution of 0.1ns and a hysteresis $\Delta V$ of 0.1V.

Figure 8 shows the clock and the output diagrams. With a proper choice of parameters, the timing accuracy can be within 10% of that of SPICE2[7]. In the diagrams, we notice that the output spikes are registered by the algorithm. However, the high accuracy requires a large number of events, which significantly slows down the simulation speed.

The test results show that the algorithm has successfully been implemented on a multicomputer. Since
Figure 8: Input and output waveforms of 2-bit counter.

the test circuit is small, and because of the nature of the chosen multicomputer, speed-up figures are not presented. However, the cosmic environment [6] enables us to recompile the program for some other multicomputer than parallel workstations. Further runs on large circuits will be performed on an Intel iPSC/860 hyper-cube.

5 Conclusions

In this paper a parallel switch-level simulator has been presented. The chosen switch-level algorithm works with continuous node voltages, which has the benefits that it avoids propagation of unknown node states, and is compatible with circuit simulators. This can be exploited for parallelism on a higher level of abstraction through waveform relaxation: for simulation of large digital circuits or mixed analog-digital simulators. The algorithm also has the advantage that the simulator can be extended to other types of circuits than CMOS logic, e.g. BiCMOS, and ECL.

Experiments on a simple test circuit show that the algorithm has successfully been implemented on a multicomputer. For speed-up measures and quantitative performance measures, simulation of large circuits run on a true multicomputer must be performed. The main challenge for achieving large speed-up, is to keep down the number of generated events.

References


