Section 9

Scheduling semantics

This section details the simulation cycles for analog simulation and mixed A/D simulations.

9.1 Analog simulation cycle

Simulation of a network, or system, starts with an analysis of each node to develop equations which define the complete set of values and flows in a network. Through transient analysis, the value and flow equations are solved incrementally with respect to time. At each time increment, equations for each signal are iteratively solved until they converge on a final solution.

9.1.1 Nodal analysis

To describe a network, simulators combine constitutive relationships with Kirchhoff's Laws in *nodal analysis* to form a system of differential-algebraic equations of the form

$$f(v, t) = \frac{dq(v, t)}{dt} + i(v, t) = 0$$
$$v(0) = v_0$$

These equations are a restatement of Kirchhoff's Flow Law (KFL).

v is a vector containing all node values

t is time

q and i are the dynamic and static portions of the flow

$f()$ is a vector containing the total flow out of each node

v_0 is the vector of initial conditions

This equation was formulated by treating all nodes as being conservative (even signal flow nodes). In this way, signal-flow and conservative terminals can be connected naturally. However, this results in unnecessary KFL equations for those nodes with only signal-flow terminals attached. This situation is easily recognized and those unnecessary equations are eliminated along with the associated flow unknowns, which shall be zero (`0`) by definition.

9.1.2 Transient analysis

The equation describing the network is differential and non-linear, which makes it impossible to solve directly. There are a number of different approaches to solving this problem numerically. However, all approaches discretize time and solve the nonlinear equations iteratively, as shown in Figure 9-1.

The simulator replaces the time derivative operator (*dq/dt*) with a discrete-time finite difference approximation. The simulation time interval is discretized and solved at individual time points along the interval. The simulator controls the interval between the time points to ensure the accuracy of the finite difference approximation. At each time point, a system of nonlinear algebraic equations is solved iteratively. Most circuit simulators use the Newton-Raphson (NR) method to solve this system.

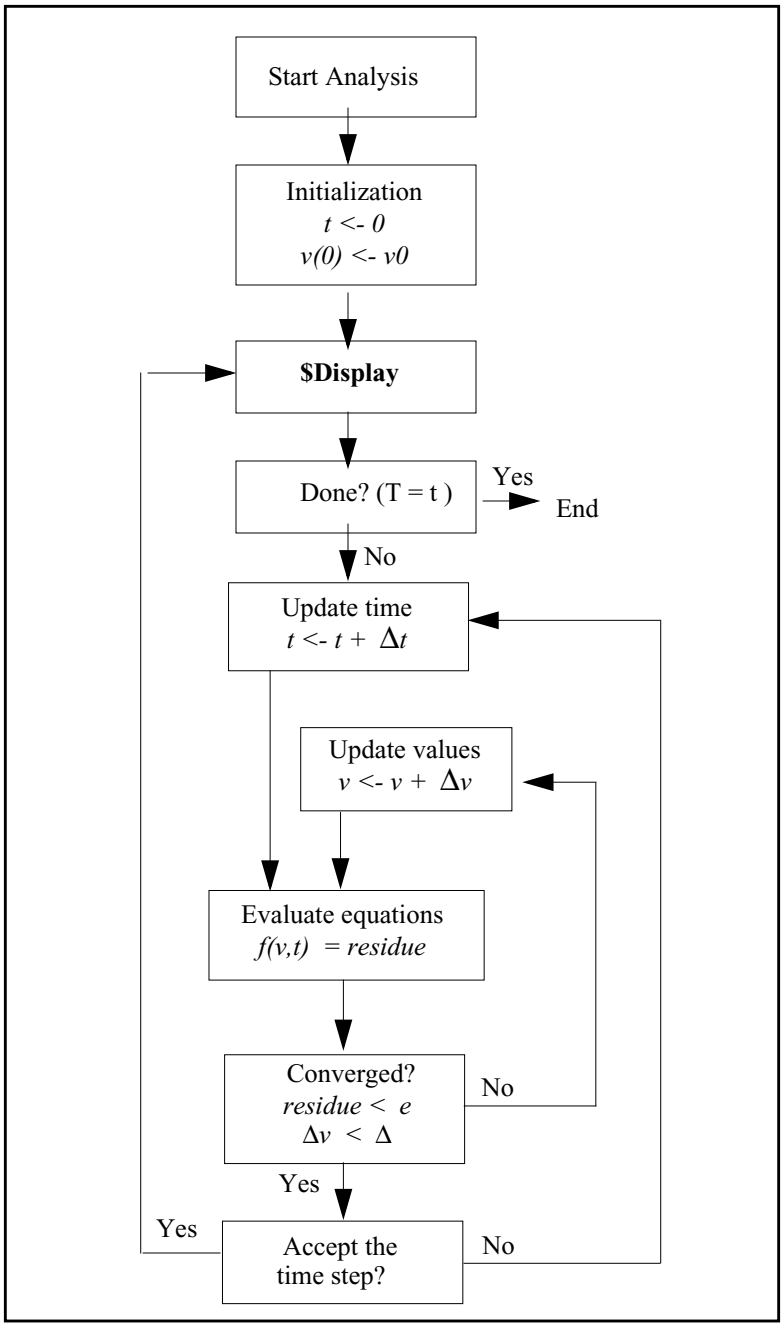

Figure 9-1 Simulation flowchart (transient analysis)

9.1.3 Convergence

In the analog kernel, the behavioral description is evaluated iteratively until the NR method converges. On the first iteration, the signal values used in expressions are approximate and do not satisfy Kirchhoff's Laws.

In fact, the initial values might not be reasonable, so models need to be written so they do something reasonable even when given unreasonable signal values.

For example, the log or square root of a signal value is being computed, some signal values cause the arguments to these functions to become negative, even though a real-world system never exhibits negative values.

As the iteration progresses, the signal values approach the solution. Iteration continues until two convergence criteria are satisfied. The first criterion is the proposed solution on this iteration, $v^{(j)}(t)$, shall be close to the proposed solution on the previous iteration, $v^{(j-1)}(t)$, and

$$| v_n^{(j)} - v_n^{(j-1)} | < reltol\ (max(| v_n^{(j)}| , |v_n^{(j-1)}|)) + abstol$$

where *reltol* is the relative tolerance and *abstol* is the absolute tolerance.

reltol is set as a simulator option and typically has a value of `0.001`. There can be many absolute tolerances, which one is used depends on the quantity the signal represents (volts, amps, etc.). The absolute tolerance is important when v_n is converging to zero (`0`). Without *abstol*, the iteration never converges.

The second criterion ensures Kirchhoff's Flow Law is satisfied:

$$\left|\sum_n f_n(v^{(j)})\right| < reltol(max(\left|f^i{}_n(v^{(j)})\right|)) + abstol$$

where $f_n^i(v^{(j)})$ is the flow exiting node *n* from branch *i*.

Both of these criteria specify the absolute tolerance to ensure convergence is not precluded when v_n or $f_n(v)$ go to zero (`0`). The relative tolerance can be set once in an options statement to work effectively on any node in the circuit, but the absolute tolerance shall be scaled appropriately for its associated signal. The absolute tolerance shall be the largest signal value which is considered negligible on all the signals where it is associated.

The simulator uses absolute tolerance to get an idea of the scale of signals. Absolute tolerances are typically 1,000 to 1,000,000 times smaller than the largest typical value for signals of a particular quantity. For example, in a typical integrated circuit, the largest potential is about 5 volts, so the default absolute tolerance for voltage is 1µV. The largest current is about 1mA, so the default absolute tolerance for current is 1pA.

9.2 Mixed-signal simulation cycle

This section describes the semantics of the initialization and time-sweep phases of a transient analysis in a mixed-signal simulation cycle.

9.2.1 Circuit initialization

The initialization phase of a transient analysis is the process of initializing the circuit state before advancing time.

9.2.2 Synchronization of Analog and Digital in Transient Analysis

A Verilog-AMS simulation consists of a number of (analog and digital) processes communicating via events, shared memory and conservative nodes. Analog processes which share conservative nodes are “solved” jointly and can be viewed as a “macro” process, there may be any number “macro” processes, and it is left up to the implementation whether it solves them in a single matrix, multiple matrices or uses other techniques but it should abide by the accuracy stipulated in the disciplines and analog functions.

9.2.2.1 Concurrency

Most (current) simulators are single-threaded in execution, meaning that although the semantics of Verilog-AMS imply processes are active concurrently, the reality is that they are not. If an implementation is genuinely multithreaded, it should not evaluate processes that directly share memory concurrently as there are no data locking semantics in Verilog-AMS.

9.2.2.2 Analog Macro Process Semantics

An analog “macro” process interacts with other processes through events and shared variables. When it is initially activated, it will attempt to predicate a potential “solution” at a a future time (the “acceptance time”) and will store (but not communicate) values[1] for all nodes at that time, and will schedule a “wake up” event for the acceptance time, the process is then inactive until woken up or it receives an event from another process. If it is woken up by its own “wake up” event it calculates a new solution point, acceptance time etc. and deactivates. If it is woken up prior to acceptance time by an event that disturbs its current solution it will cancel its own “wake up” event, accept at the wake-up time, recalculate its solution and schedule a new “wake up” event for the new acceptance time.

If the analog process identifies potential “crossings” then it will schedule it’s wake-up event for the time of the first such event rather than the acceptance time. If the analog process is woken by such a crossing event it will communicate any related events at that time and de-activate, rescheduling it’s wake-up for the next crossing or acceptance; events to external processes generated from analog events are not communicated until the global simulation time reaches the time of the analog event.

If the time to acceptance is infinite then no “wake up” event needs to be scheduled[2].

1. Or derivatives w.r.t. time used to calculate the values.
2. The case when all derivatives are zero - the circuit is stable.

As with digital processes, analog processes are insensitive to changes in variables i.e. a change in a variable does not force re-evaluation of the process, neither are they implicitly sensitive to digital signals used in procedural code, only changes in signals in event expressions will trigger re-evaluation prior to scheduled wake-up.

9.2.2.3 A/D Boundary Timing

In the analog kernel, time is a floating point value. In the digital kernel time is an integer value. Hence, A2D events generally do not occur exactly at digital integer clock ticks.

For the purpose of reporting results and scheduling delayed future events, the digital kernel rounds A2D events to the nearest tick so that error is limited to half a tick when swapping an analog device for its digital equivalent. A2D statements that do not include a scheduling delay are processed immediately in a new digital simulation cycle such that dependent non-blocking assigns are executed before control returns to the analog domain.

Consequently an A2D event which results in a D2A event being scheduled with zero (`0`) delay, shall have its effect propagated back to the analog kernel with zero (`0`) delay.

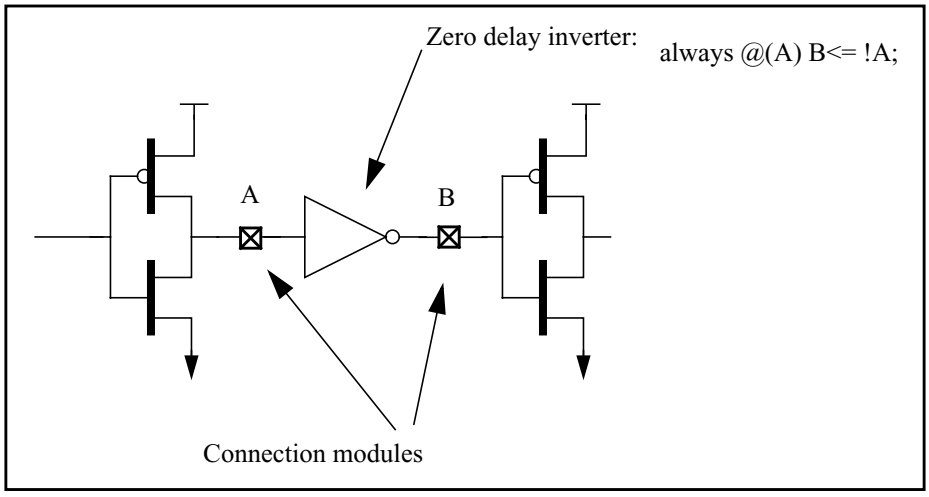

Figure 9-2 A zero delay inverter

If the circuit shown in Figure 9-2 is being simulated with a digital time resolution of `1e-9` (one (`1`) nanosecond) then all digital events shall be reported by the digital kernel as having occurred at an integer multiple of `1e-9`.

If connector `A` detects a positive threshold crossing the resulting falling edge at connector `B`, this shall be reported to the analog kernel with no further advance of analog time. However, the digital kernel will round the time of these events to the nearest nanosecond.

Example:

If `A` detects a positive crossing as a result of a transient solution at time `5.27e-9`, the digital kernel shall report a rising edge at `A` at time `5.0e-9` and falling edge at `B` at time `5.0e-9`, but the analog kernel shall see the transition at `B` begin at time `5.27e-9`, as

shown in Figure 9-3. D2As fed with zero delay events cannot be preemptive, so the crossover on the return is delayed from the digital event; zero-delay inverters are not physicaly realizable devices.

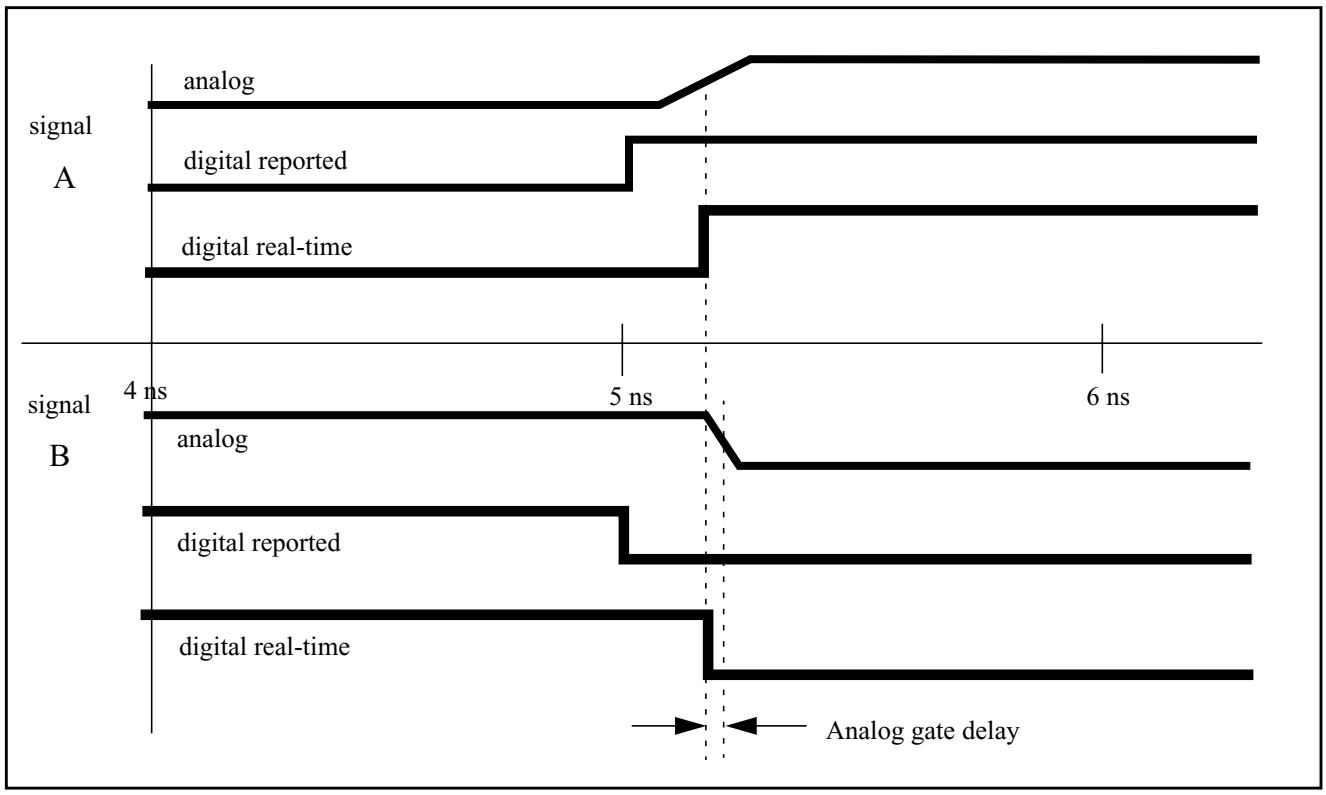

Figure 9-3 Transient solution times