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 8-1.

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.

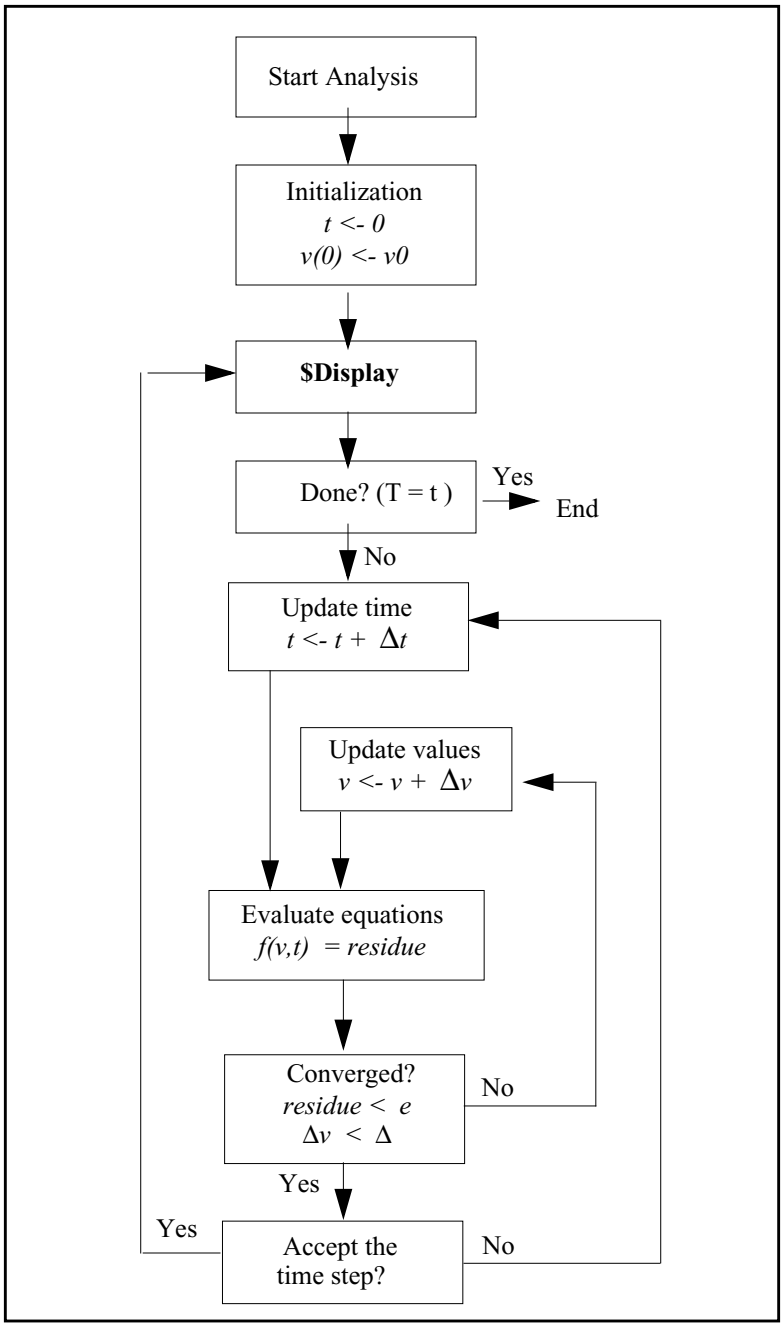

Figure 8-1 Simulation flowchart (transient analysis)

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.

This section addresses Issue 5. >>

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 predict a potential “solution” at 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. The process may also wake itself up early for reevaluation by use of a timer (which can be viewed as just another process).

If the analog process identifies future analog events such as “crossings” or timer events (see Monitored Events 6.7.5 ??) 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 an analog event it will communicate any related events at that time and de-activate, rescheduling it’s wake-up for the next analog event or acceptance; events to external

1. Or derivatives w.r.t. time used to calculate the values.

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[1].

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 all digital signals used in procedural code, only changes in signals used in event expressions or as arguments to filter functions will trigger re-evaluation prior to scheduled wake-up.

<< Being sensitive to all digital signals automatically creates problems with digital behavioral code which produces valid values only when required (e.g. data around a clock edge, test code often drives Xs deliberately), the analog process should be able to sample data the same way digital code does so that it can ignore spurious data and not do unnecessary evaluation. Variables are local to a module and can only be changed by external processes if accessed through hierarchical reference (we may want to add a hierarchical mechanism to force acceptance and recalculation for testbenches probing into analog modules). SystemVerilog has added the syntax "@(*)" to catch all dependencies in digital verilog, we could adopt it too.

Macro processes can be evaluated seperately, but may be evaluated together[2] in which case the wake up event for one process will cause the re-evaluation of all or some of the processes. Users should bear this in mind when writing mixed signal code, as it will mean that the code should be able to handle re-evaluation at any time (not just at its own event times).

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.

Addressing Issue 1 >>

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 zero-delay non-blocking assigns are executed before control returns to the analog domain. Rounding of time to the digital clock tick on A2D events is required to support implementations using seperate event queues (See section 9.2.4).

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.

---

1. The case when all derivatives are zero - the circuit is stable.
2. This is implementation dependent.

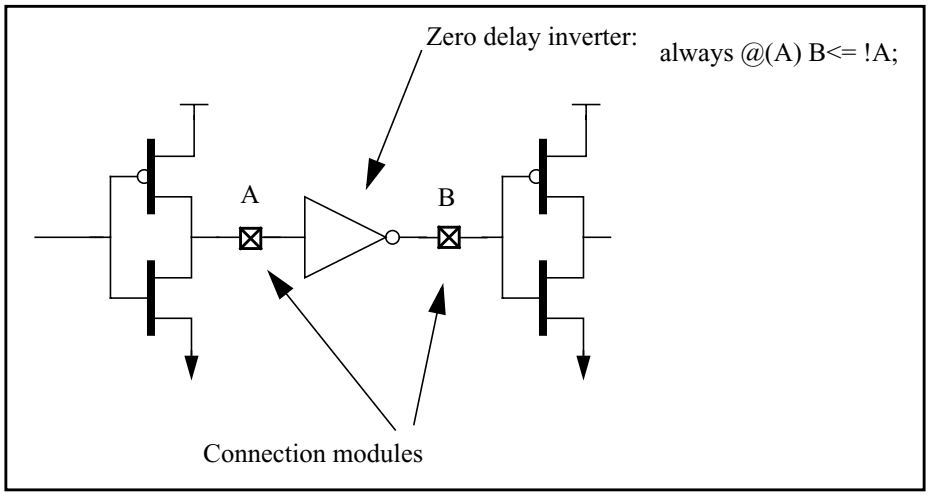

Figure 8-2 A zero delay inverter

If the circuit shown in Figure 8-2 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`. The A2D and D2A modules inserted are a simple level detector and a voltage ramp generator:

```
connectmodule a2d(i,o);
   parameter VDD = 1.0;
   input       i;
   output      o;
   reg         o;
   electrical  i;
   analog begin   @(cross(V(i) - VDD/2,+1)o <= 1;
                  @(cross(V(i) - VDD/2,-1)o <= 0; end
endconnectmodule
connectmodule d2a(i, o);
   parameter   VDD   =  1.0;
   parameter   slew  =  2.0/1e9; // V/s
   input       i;
   output      o;
   electrical  o;
   reg         qd_val,  // queued value
               nw_val;
   real        et;      // delay to event
   always @(driver_update i) begin
```

```
        nw_val = $driver_next_state(i,0); // assume one driver
        if (nw_val == qd_val) begin
           // no change (assume delay constant)
        else
           et       = $driver_delay(i,0) * 1e-9; // real delay
           qd_val   = nw_val;
        end
     end
     real  start_delay;   // .. to ramp start
     analog begin
        @(qd_val)   start_dly = et - (VDD/2)/slew;
        V(o) <+ VDD * transition(qd_val,start_dly,VDD/slew);
     end
  endmodule
```

If connector `A` detects a positive threshold crossing the resulting falling edge at connector `B` generated by the propagation of the signal through verilog inverter model shall be reported to the analog kernel with no further advance of analog time. The digital kernel will treat these events as if they occurred at the nearest nanosecond.

Example:

If `A` detects a positive crossing as a result of a transient solution at time `5.2e-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 `5.2e-9`, as shown in Figure 8-3. 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.

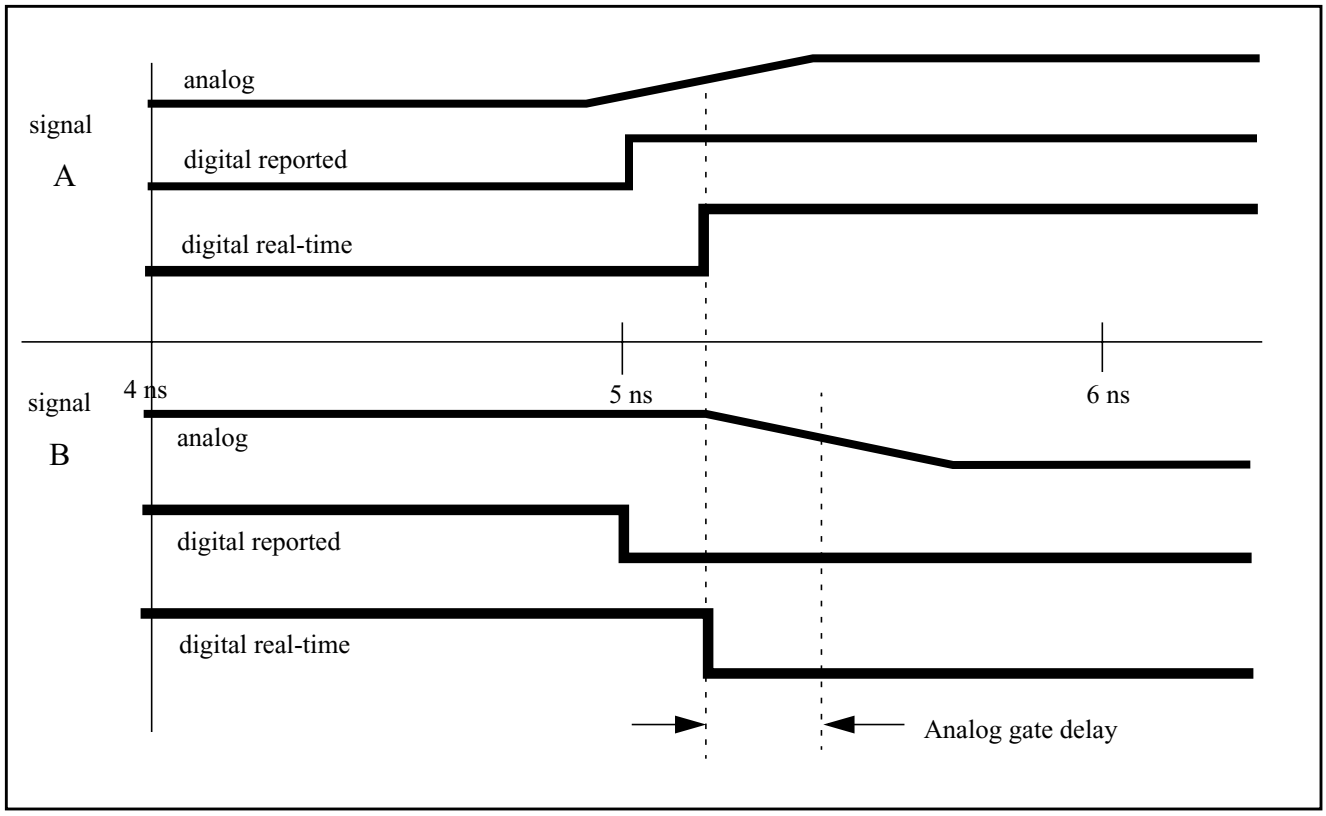

Figure 8-3 Zero delay transient solution times

If the inverter equation is changed to use a one unit delay (*always @(A) B<= #1 !A*), then the timing is as in Figure 8-4.

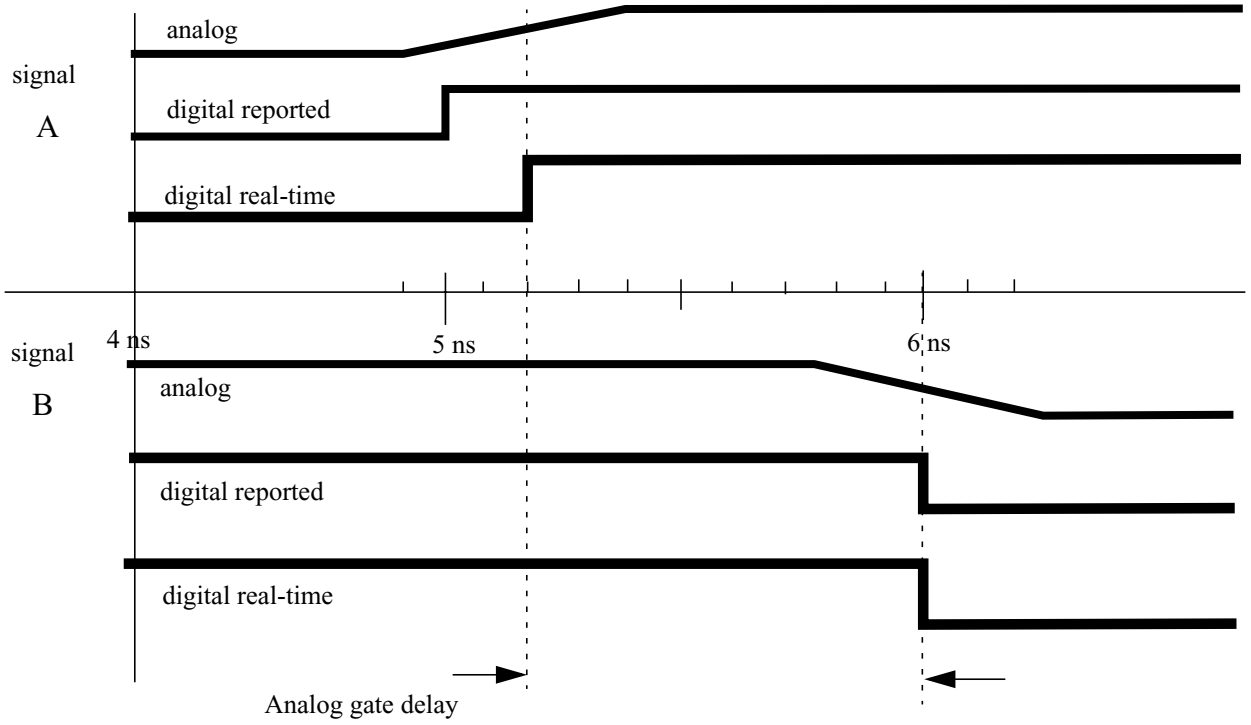

Figure 8-4 Unit delay transient solution times

9.2.3 The synchronization loop

Verilog-AMS uses a “conservative” simulation algorithm, the analog and digital processes which are managed by the simulation kernel are synchronized such that neither computes results which will invalidate signal values that have already been assigned; time never goes backwards. While the implementation of the simulator may have seperate event queues for analog and digital events (See section 9.2.4), it can be viewed as a single event queue logically with a common global time. Analog processes are similar to Verilog *initial* statements in that they start automatically at time zero. The event sequence for the transient simulation shown in Figure 8-4 would be as follows:

Time	**Event Queue**
4.9ns	Evaluate the first analog inverter
	Evaluate acceptance at 5.4ns, but schedule wake-up for 5.2 for crossing.
5.2ns	Evaluate crossing event
	The A2D logic sets the digital signal A which triggers the evaluation of the non-blocking assign to B which

	schedules the actual assignment for 6ns (rounded 1ns delay).
	D2A notices queued event and schedules wake-up for 5.75 via rampgen module.
	Schedule wake-up at 5.4ns (as previously calculated).
5.4ns	Evaluate acceptance
	Circuit evaluates stable, nothing scheduled.
5.75ns	D2A/rampgen process wake-up
	Start ramp in analog domain.
6.0ns	Non blocking assign performed (digital event).
	D2A may be sensitive, but doesn't need to do anything.
6.25ns	D2A/rampgen process wake-up
	Drive 0V to complete ramp. Nothing more to schedule.

Any events queued ahead of the current global event time may be cancelled. For instance if the sequence above is interrupted by an a change on the primary input before digital assignment takes place as shown in Figure 8-5.

Time	**Event Queue**
4.9ns	Evaluating the first analog inverter
	Evaluate acceptance at 5.4ns, but schedule wake-up
	for 5.2 for crossing.
5.2ns	Evaluate crossing event
	The A2D logic sets the digital signal A which triggers the evaluation of the non-blocking assign to B which schedules the actual assignment for 6ns (rounded 1ns delay).
	D2A notices queued event and schedules wake-up for 5.75.
	Schedule wake-up at 5.4ns (as previously calculated).
5.3ns	Analog event disturbs the solution
	Accept at 5.3ns.
	Cancel 5.4ns wake-up.
	New acceptance is 5.45ns, but schedule wake-up for crossing at 5.4ns.
5.4ns	Evaluate crossing event

The A2D logic sets the digital signal A which triggers the evaluation of the non-blocking assign to B which schedules the actual assignment for 6ns (rounded 1ns delay) cancelling previous event.

D2A notices queued event is going to drive the current value and deschedules the wake-up for 5.75.

Schedule wake-up at 5.45ns (as previously calculated).

5.45ns Evaluate acceptance

Circuit evaluates stable, nothing scheduled.

6.00ns Non blocking assign performed (digital event).

Value of B doesn't change.

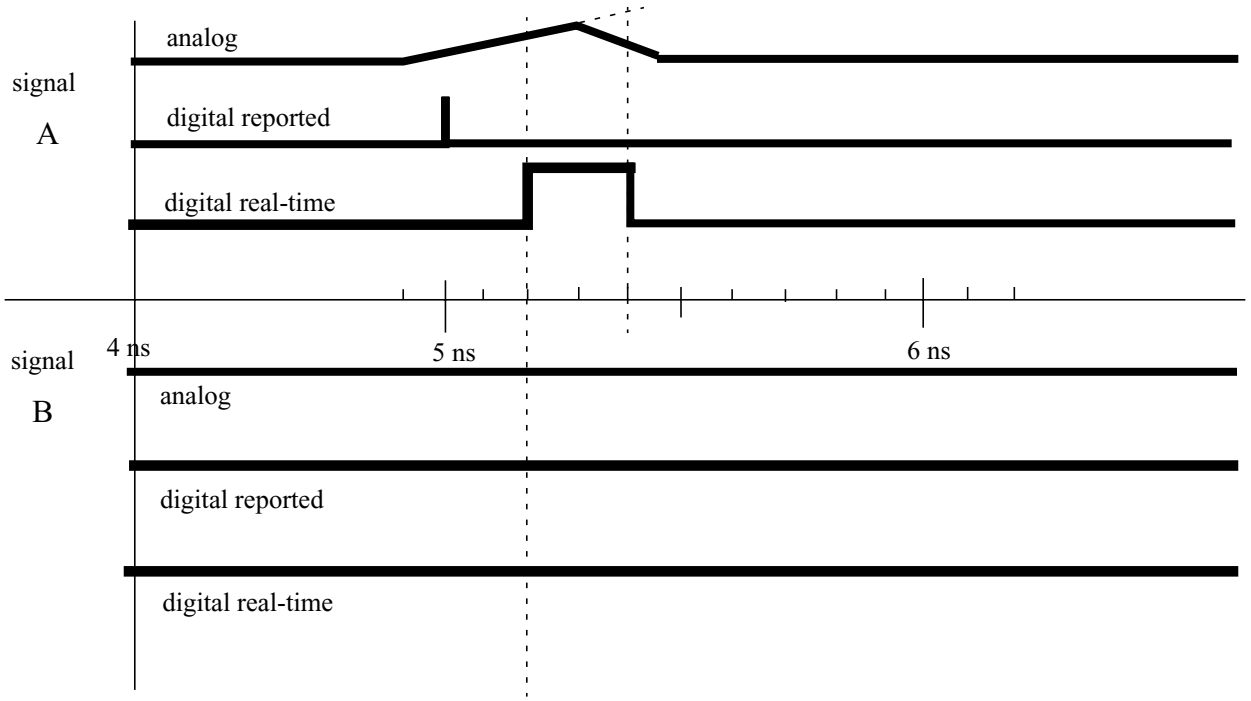

Figure 8-5 Transient solution times with glitch

If the cancelling event arrived after the ramp on B had started but before the assignment to the digital B it is possible to see the glitch propagate back into the analog domain without an event appearing on B.

9.2.4 Dual Kernel Algorithm

Mixed-signal simulators are often the combination of two seperate analog and digital simulators. The algorithm above is an abstract description of how digital and analog processes communicate and synchronize. Dual kernel simulators will have two seperate event queues, one for analog events and one for digital, and control passes between the kernels to keep them synchronized.

The synchronization algorithm can exploit characteristics of the analog and digital kernels described in the next section.

A sample run is shown in Figure 8-6.

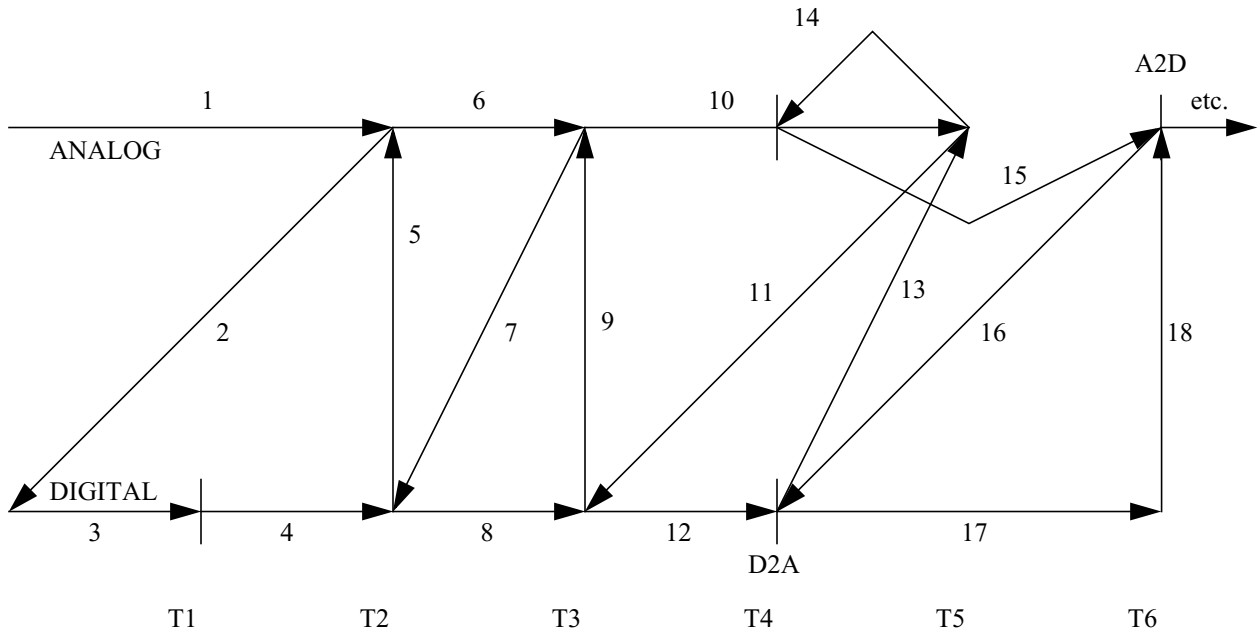

Figure 8-6 Sample Run

1. The Analog engine begins transient analysis and sends state information (that it is good up to T2) to the Digital engine (1, 2).
2. The Digital engine begins to run using its own time steps (3); however, if there is no D2A event, the Analog engine is not notified and the digital engine continues to simulate until it can not advance its time without surpassing the time of the analog solution (4). Control of the simulation is then returned to the analog engine (5) which accepts at T2. This process is repeated (7, 8, 9, 10, and 11).
3. If the Digital engine produces a D2A event (12), control of the simulation is returned to the Analog engine (13). The analog engine accepts at the time of the

D2A event (14, which may involve recalculating from T3). The Analog engine then calculates the next time step (15).

4. If the Analog engine produces an A2D event, it returns control to the Digital engine (16), which simulates up to the time of the A2D event and then surrenders control (17 and 18).

5. This process continues until transient analysis is complete.

9.2.5 Assumptions about the analog and digital algorithms

1. Advance of time in a digital algorithm

   A. The digital simulation has some minimum time granularity and all digital events occur at a time which is some integer multiple of that granularity.

   B. The digital simulator can always accept events for a given simulation time provided it has not yet executed events for a later time. Once it executes events for a given time, it can not accept events for an earlier time.

   C. The digital simulator can always report the time of the most recently executed event and the time of the next pending event.

2. Advance of time in an analog algorithm

   A. The analog simulator advances time by calculating a sequence of solutions. Each solution has an associated time which, unlike the digital time, is not constrained to a particular minimum granularity.

   B. The analog simulator can not tell for certain the time when the next solution converges. Thus, it can tell the time of the most recently calculated solution, but not the time of the next solution.

   C. In general, the analog solution is a function of one or more previous solutions. Having calculated the solution for a given time, the analog simulator can either accept or reject that solution; it cannot calculate a solution for a future time until it has accepted the solution for the current time.

3. Analog to digital events

   A. Analog to digital events are generated by conversion elements (which are analog/digital behavioral models) when evaluated by the analog simulator.

   B. Analog events (e.g., `cross`, `initial_step`, and `final_step`) cause an analog solution of the time where they occur.

   C. Thus, any analog to digital event is generated as the result of a particular transient solution. This means events can stay associated with the solution which produced them until they are passed to the digital simulator, until then they can be rejected along with the solution if it is rejected.

4. Digital to analog events shall cause an analog solution of the time where they occur.