Section 9

Scheduling semantics

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

AMS-LRM Issue 25: MS synchronization mechanism not clearly defined - Martin O'Leary

9.1 Introduction

A mixed signal simulator shall contain an analog solver that follows the analog simulation cycle described in 9.2. This component of the mixed-signal simulator is termed the analog engine. It shall also contain a discrete event simulator that is described in 9.4. This component is termed the digital engine.

In a mixed signal circuit, an "analog macro-process" is a set of continuous nodes that must be solved together because they are joined by analog blocks or analog primitives. A mixed-signal circuit can comprise one or more of these analog macro-processes separated by digital processes. The analog engine may solve the analog macro-processes together or separately.

9.2 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.

9.2.1 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.

9.2.2 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.

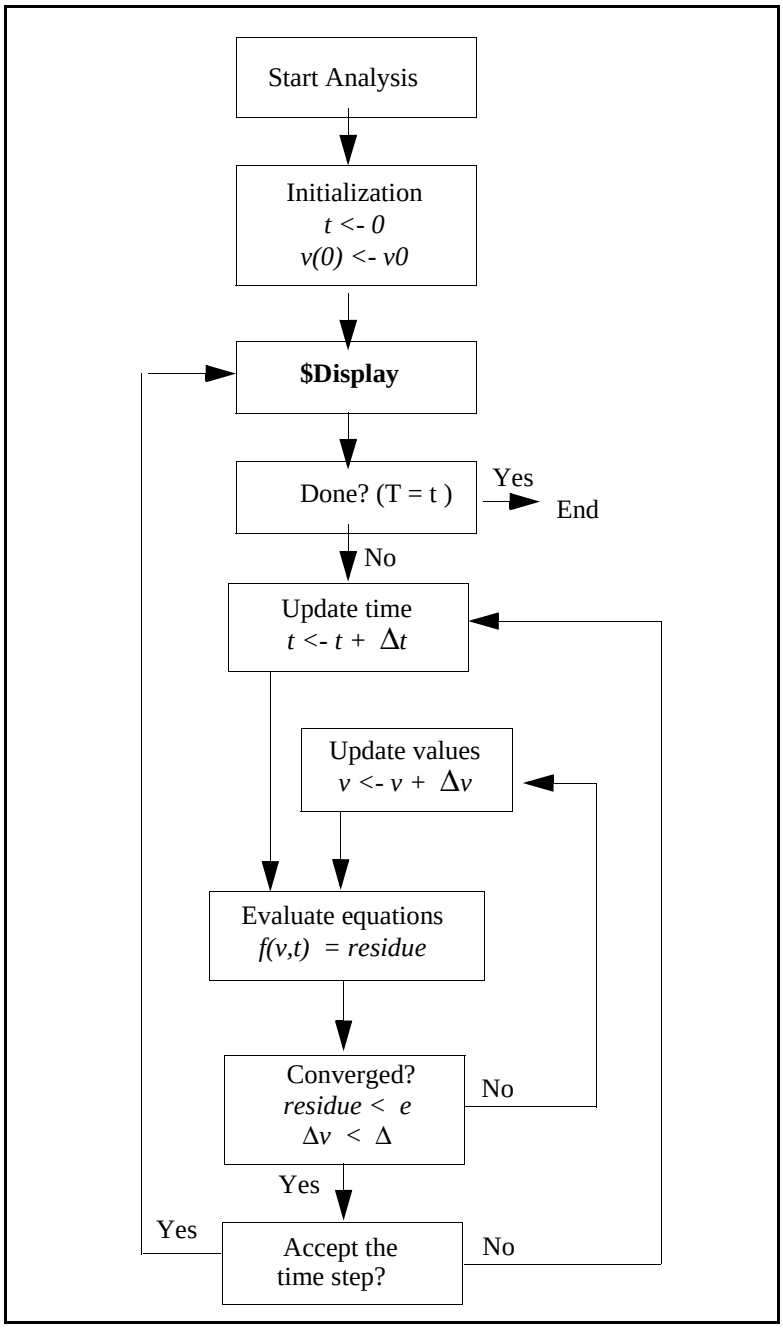

Figure 9-1 Simulation flowchart (transient analysis)

9.2.3 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:

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

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.

9.3 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.

9.3.1 Circuit initialization

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

9.3.2 Transient analysis & A/D algorithm synchronization

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 truncates A2D events down to the earlier tick. Any events which are scheduled with zero (`0`) delay (as a result of the A2D) are not snapped down. Instead they are processed immediately.

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.

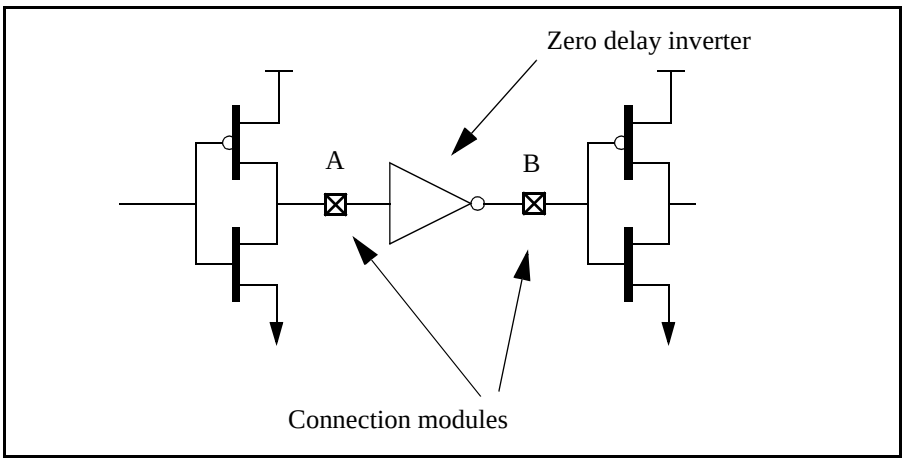

Figure 9-2 A zero delay inverter

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 could to 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.

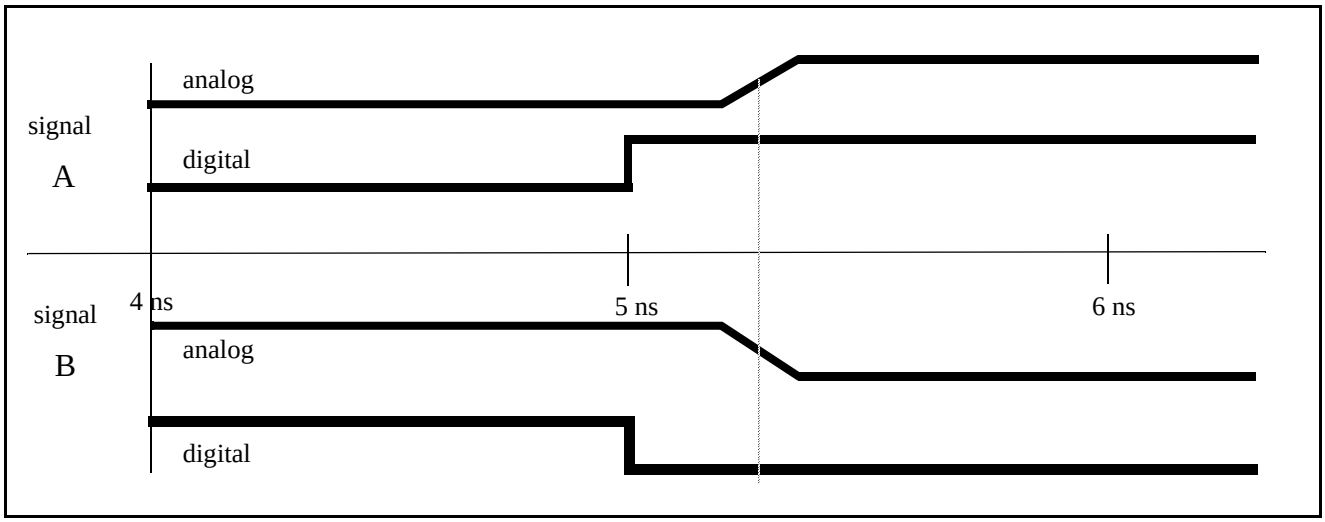

Figure 9-3 Transient solution times

9.3.3 The synchronization loop

The digital and analog kernels shall be synchronized so neither computes results which the other is ineligible to accept. The synchronization algorithm can exploit characteristics of the analog and digital kernels described in the next section. A sample run is shown in Figure 9-4.

1. The Analog engine begins transient analysis and sends state information 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 to 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). 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 returns to the point at which the digital engine last surrendered control (14). The Analog engine recalculates the analog solution up to the time when the D2A event occurred (15). The Analog engine then takes the next time step (16).
4. If the Analog engine produces an A2D event, it returns control to the Digital engine (17), which simulates up to the time of the A2D event and then surrenders control (18 and 19).
5. This process continues until transient analysis is complete.

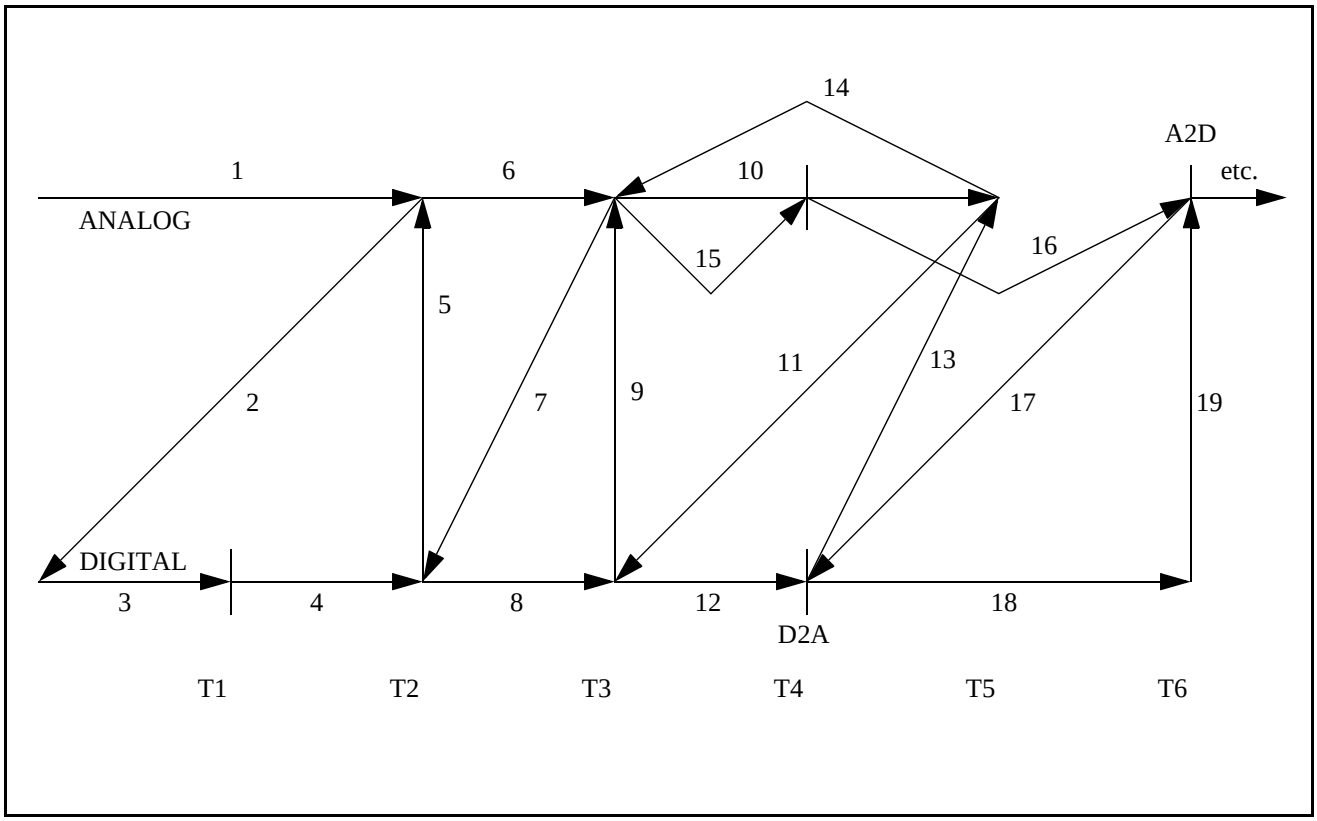

Figure 9-4 Sample run

9.3.4 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 can not 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, 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.

9.4 Scheduling Semantics for the Digital Engine

The scheduling semantics for Verilog-HDL simulation are outlined in Section 5 of 1364-2001.

The digital engine of a Verilog-AMS mixed-signal simulator will comply with that section except for the changes outlined in this section.

For mixed-signal simulation the major change from Section 5 of 1364-2001 is that two new type of event must be supported by the event queue called the *explicit D2A* (digital-to-analog) *event,* and the *analog macro-process event.*

Explicit D2A events are created when a digital event which an analog block is *explicity* sensitive to, occurs. An analog block is explicity sensitive to event expressions mentioned in an event control statement in that analog block.

Similarly there is also the concept of the *implicit D2A event* which is created when a digital variable which analog block is *implicity sensitive* to, changes. An analog block is implicity sensitive to all digital variable references that are not guarded event control statement in that analog block.

An analog macro-process event is also created when either type of D2A event occurs. The analog macro-process event is associated with the analog macro-process that is sensitive to the D2A event. An analog macro-process event is evaluated by calling the

analog engine to solve the associated analog macro-process. Note that implicit D2A events are not added to the statified event queue but as they directly cause an analog macro-process event, they effectively force a digital-analog synchonization.

9.4.1 The stratified event queue

The Verilog event queue is logically segmented into six different regions. Events are added to any of the six regions but are only removed from the *active* region.

1. Events that occur at the current simulation time and can be processed in any order. These are the *active* events. Explicit D2A events, an addition for mixed signal simulation, are a type of active event.
2. Events that occur at the current simulation time,but that shall be processed after all the active events are processed. These are the *inactive* events.
3. Events that have been evaluated during some previous simulation time, but that shall be assigned at this simulation time after all the active and inactive events are processed. These are the non blocking assign update events.
4. Events that shall be processed after all the active, inactive, and non blocking assign update events are processed. These are the *monitor* events.
5. Events that occur at some future simulation time. These are the *future* events.Future events are divided into *future inactive events*, and *future non blocking assignment update events*.
6. Another mixed signal addition to the scheduling semantics for is that D2A events cause an analog macro-process event to be scheduled for the analog macro-process which is to process the D2A event.

The processing of all the active events is called a *simulation cycle*.

The freedom to choose any active event for immediate processing is an essential source of nondeterminism in the Verilog HDL.

An *explicit zero delay* (#0) requires that the process be suspended and added as an inactive event for the current time so that the process is resumed in the next simulation cycle in the current time.

A non blocking assignment (see 9.2.2 of 1364-2001) creates a non blocking assign update event, scheduled for current or a later simulation time.

The **$monitor** and **$strobe** system tasks (see 17.1 of 1364-2001) create monitor events for their arguments. These events are continuously re-enabled in every successive time step. The monitor events are unique in that they cannot create any other events.

The call back procedures scheduled with PLI routines such as tf_synchronize() (see 25.58 of 1364-2001) or vpi_register_cb(cb_readwrite) (see 27.33 of 1364-2001) shall be treated as inactive events.

Note that A2D events must be analog event controlled statements (e.g. @cross, @timer) - these are schedule just like other event controlled statements in Verilog-HDL (e.g. @posedge).

9.4.2 The Verilog-AMS digital engine reference model

In all the examples that follow, T refers to the current simulation time of the digital engine, and all events are held in the event queue, ordered by simulation time.

```
while (there are events){
   if (no active events){
      if (there are inactive events){
         activate all inactive events;
      }else if (there are non blocking assign update events){
         activate all non blocking assign update events;
      }else if (there are analog macro-process events) {
         activate the analog macro-process events;
      }else if (there are monitor events){
         activate all monitor events;
      }else {
         advance T to the next event time;
         activate all inactive events for time T;
      }
   }
   E =any active event;
   if (E is an update event){
      update the modified object;
      add evaluation events for sensitive processes to event queue;
   }else if (E is an analog macro-process event) {
      evaluate the analog macro-process
      modify the analog values
      add A2D events to event queue, if any
   }else {/*shall be an evaluation event */
```

```
            evaluate the process;
            add update events to the event queue;
        }
    }
```

9.4.3 Scheduling implication of assignments

Assignments are translated into processes and events as follows.

9.4.3.1 Continuous assignment

A continuous assignment statement (Section 6 of 1364-2001) corresponds to a process, sensitive to the source elements in the expression.When the value of the expression changes, it causes an active update event to be added to the event queue, using current values to determine the target.

9.4.3.2 Procedural continuous assignment

A procedural continuous assignment (which are the **assign** or **force** statement; see 9.3 of 1364-2001) corresponds to a process that is sensitive to the source elements in the expression. When the value of the expression changes, it causes an active update event to be added to the event queue,using current values to determine the target.

A **deassign** or a **release** statement deactivates any corresponding **assign** or **force** statement(s).

9.4.3.3 Blocking assignment

A blocking assignment statement (see 9.2.1 of 1364-2001) with a delay computes the right-hand side value using the current values,then causes the executing process to be suspended and scheduled as a future event. If the delay is 0, the process is scheduled as an inactive event for the current time.

When the process is returned (or if it returns immediately if no delay is specified), the process performs the assignment to the left-hand side and enables any events based upon the update of the left-hand side. The values at the time the process resumes are used to determine the target(s). Execution may then continue with the next sequential statement or with other active events.

9.4.3.4 Non blocking assignment

A non blocking assignment statement (see 9.2.2 of 1364-2001) always computes the updated value and schedules the update as a non blocking assign update event, either in this time step if the delay is zero or as a future event if the delay is nonzero. The values in effect when the update is placed on the event queue are used to compute both the right-hand value and the left-hand target.

9.4.3.5 Switch (transistor) processing

The event-driven simulation algorithm described in 5.4 of 1364-2001 depends on unidirectional signal flow and can process each event independently. The inputs are read,the result is computed, and the update is scheduled. The Verilog HDL provides switch-level modeling in addition to behavioral and gate-level modeling. Switches provide bi-directional signal flow and require coordinated processing of nodes connected by switches.

The Verilog HDL source elements that model switches are various forms of transistors, called **tran**, **tranif0**, **tranif1**, **rtran**, **rtranif0**, and **rtranif1**.

Switch processing shall consider all the devices in a bidirectional switch-connected net before it can determine the appropriate value for any node on the net, because the inputs and outputs interact. A simulator can do this using a relaxation technique. The simulator can process tran at any time. It can process a subset of tran-connected events at a particular time, intermingled with the execution of other active events. Further refinement is required when some transistors have gate value x. A conceptually simple technique is to solve the network repeatedly with these transistors set to all possible combinations of fully conducting and nonconducting transistors. Any node that has a unique logic level in all cases has steady-state response equal to this level. All other nodes have steady-state response.

9.4.3.6 Explicit D2A event processing

When an explicit D2A event is processed, the analog block that is sensitive to this event is processed before all the active events are exhausted so that the values for the digital variables referenced inside the sensitive analog event control statement are the values of those variables before the inactive events are activated.

9.4.3.7 analog macro-process event scheduling/processing

When a D2A event occurs, the analog macro-process which is sensitive to the D2A event is scheduled for evaluation on the digital engine event queue. Note that if multiple analog macro-process events are scheduled for a particular analog macro-process for a particular time, then a single evaluation of the analog macro-process should consume all of these events from the queue.

The reason for processing analog macro-processes after other digital events is in order to minimize the number of times analog macro-processes are evaluated because such evaluations tend to be expensive.