Time Driven

In this type, time is advanced in predetermined uniform increments. As time is advanced, any events in the next time step are simulated. There is a computational overhead because many time increments do not contain any changes in output values. A time-driven simulator still must execute a simulation cycle for each time step. In a time-driven simulation there is a variable recording the current time, which is incremented in fixed steps. After each increment the simulator checks to see which events may happen at the current time point and it handles those that do.

For example, suppose ones want to simulate an 8-bit counter. At time zero we assign the counter an initial value. At each time step we can calculate a new value that is a function of the time step. Time-driven simulation is suitable here because there is an event (incrementation) happening at each time step. A time-driven simulation may be terminated either at a pre-determined time or when the system reaches a certain state.

Event-Driven

Event-driven simulators are driven based on events. Outputs are a function of the inputs: the outputs change only when the inputs change. The event then is the input changing. Each time such an I/O event occurs, the simulator must reevaluate and calculate the new output. Note that outputs of one block may be used as inputs for another block. In this way events can cascade. These cascaded reevaluations continue until there are no more changes to propagate.

Delta time is a concept that must be addressed, therefore. In the above cascade examples, evaluation cycles occurring at the time step occur in delta time. Delta time is non-real time that allows for a sequence of evaluation to occur at each time step for all eligible devices receiving and input change.

The event driven simulator maintains many lists. It maintains a list of all atomic executable blocks and maintains fanout lists which are a data structure that represent the interconnect of the blocks via signals. A time queue records the points in time when events happen. The event queue holds one queue pair for each entry in the time queue. The simulator then processes all these queues at event time.

Event-driven simulators advance time in non-uniform steps which have a size that depends on when the event occurs. The event-driven simulator responds to each I/O event by executing a sequence of simulation cycles which determine when and to what values the simulated system's signals change. Some of advantages of event-driven simulators include the elimination of the need for the simulator to evaluate the model at every time step like the time-driven simulator does. This results in faster and more precise simulations.

There are three main steps used to accomplish an event-driven simulation.

- Elaboration
- Initialization
- Execution/Update

Elaboration

Elaboration is the creation of a simulation model for a design entity from its VHDL description. This simulation model is stored in the memory of the host computer. During elaboration all concurrent statements are converted into equivalent simulation processes. During elaboration a signal driver is created for each assigned signal value. This driver is associated with the simulation process containing the signal assignment statement.

Initialization

Once elaboration has completed, simulation is the next step, which consists of an initialization phase followed by the repetitive execution of simulation cycles. At the beginning of the initialization phase, the current time is set to 0. The kernel then places all the simulation processes in the active processes queue. Each simulation process is then taken from this queue and executed until it suspends. A simulation process is suspended either implicitly or explicitly. A process with a sensitivity list is suspended implicitly after its sequential statements have been executed at the end of the process. A process with one or more wait statements is suspended explicitly when its first wait statement is executed.

After the initialization, all simulation processes are in their suspended states. The first simulation cycle is then executed. A simulation cycle consists of two phases.

Execution/Update

During the execution phase, each simulation process in the active processes queue is taken from that queue and executed until it suspends. The update phase first determines the next value for the current simulation time and advances the simulator clock to this value. Based on this new simulation time, a determination is made as to whether the simulation is complete.

Cycle based

Cycle based simulations are faster than event-driven simulation. This is only applicable to functional simulation of synchronous sequential systems that have a single clock. A cycle-based simulator collapses logic which determines the flip-flop I/O values in a sequential design into equations based on the present input values and present state of the system's flip-flops. The resultant model is evaluated only at the triggering edge of each clock cycle. Because the execution of a model only at each clock triggering edge, this reduces the simulation time. However, there is a loss of delay information and in addition only signal values at clock triggering edges are available.

In a cycled-based simulation the stead-state response of the circuit is calculated at each clock cycle and at each boundary node. Cycle based simulation is a technique for simulating circuits that do not consider the detailed circuit timing. Cycle-based simulation computes the steady-state response of the circuit at each clock cycle boundary. Another tool is required for timing analysis.

In cycle-based simulation all scheduled events will take place at the active edge of the clock. All events are put in an event order which eliminates unnecessary evaluations. Cycle-based simulation allows for 10 to 100 times performance over an event-driven simulator. This is because timing is ignored which allows for efficient evaluation and faster simulation. In addition, a simpler data structure is used for the simulation. Cycle-based simulation finds its best use cases when there are large synthesizable or logic optimizable designs. In addition, designs using this simulation type should be mostly synchronous. In addition, the design should fall into either RTL regression or gate-level verification.