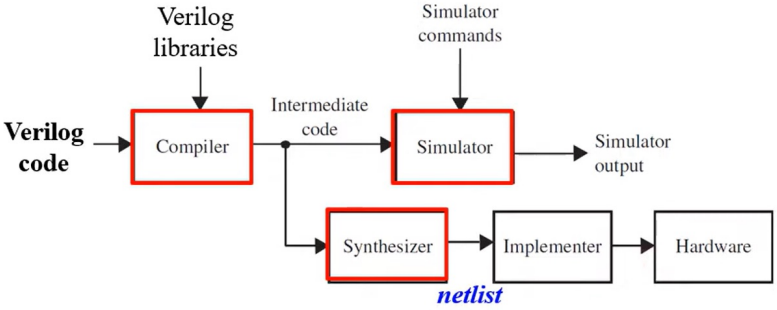
2.10 Compilation, Simulation, and Synthesis of Verilog Code

1 Introduction

Compilation, Simulation, and Synthesis of Verilog Code

- Compilation, Simulation, and Synthesis of Verilog Code:

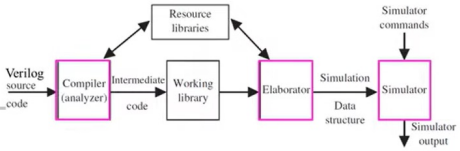


  - ***netlist*** : a list of required ***components*** and their ***interconnections***

2-102

2 Simulation

Simulation Phases



- Analysis (compilation):
  - Checks source code; if a syntax or semantic error occurs, then the compiler gives error message.
  - Checks references to libraries.
- Elaboration:
  - creates a hierarchy of module instances, propagates parameters among modules, makes drivers for signals, and constructs a design hierarchy.
- Simulation: ***discrete event simulation***
  - consists of an initialization phase and actual simulation.
  - ***Initialization phase***: gives an initial value to the signal
  - ***Simulation***: Events are kept on an event queue, ordered by simulation time.

Verilog Event Queue

- Five regions of the Verilog event queue:
  1. ***Active event region***
  2. ***Inactive event region***
  3. ***Non-blocking assign update region***
  4. ***Monitor event region***
  5. ***Future event region***

current simulation time

future simulation time

Scheduling Convention

Regions of Verilog event queue:
- Active event region
- Inactive event region
- Non-blocking assign update region
- Monitor event region
- Future event region

- Adding events to various queue regions for each type of statements:

1. ***Continuous assignment***: evaluate RHS and add to active region as an ***active update event***
2. ***Procedural continuous assign***: evaluate RHS and add to active region as an ***active update event***
3. ***Blocking assignment with no delay***: compute RHS and put into inactive region as an ***inactive event*** for current time
4. ***Blocking assignment with delay***: compute RHS and put into future event region for time after delay
5. ***Non-blocking assignment with no delay***: compute RHS and schedule as ***non-blocking assign update event*** for current time
6. **Non-blocking assignment with delay**: compute RHS and schedule as ***non-blocking assign update event*** for future time
7. **$monitor** / **$strobe** system tasks: create ***monitor events*** which are continuously reenabled in every successive time step.

- Actions for each ***simulation cycle***:

1. Process all active update events.
2. Activate all inactive events for that time.
3. Activate all non-blocking assign update events and process them.
4. Activate all monitor events and process them.
5. Advance time to the next event time and repeat from step 1.

* All of these 5 steps happen at the same time (one simulation cycle), but the events occur in the order *active*, *inactive*, *non-blocking update*, and *monitor* events.

Scheduling of Nonblocking (<=) Assignment

- Scheduling of simulator for "<=":
  - Whenever a component input changes, the output is scheduled to change after ***the specified delay*** or after ***an infinitesimal delay*** (***Δ delay***) if no delay is specified.
  - If two non-blocking updates are made to the same variable in the same time step, the 2nd one dominates by the end of the time step.

Ex 1.

Examples: p.90, Fig 2-26

- (a) determinate

```
module determinate;
   reg a;
   initial a = 0;
   always begin
      a <= #5 0;
      a <= #5 1;
   end
endmodule
```

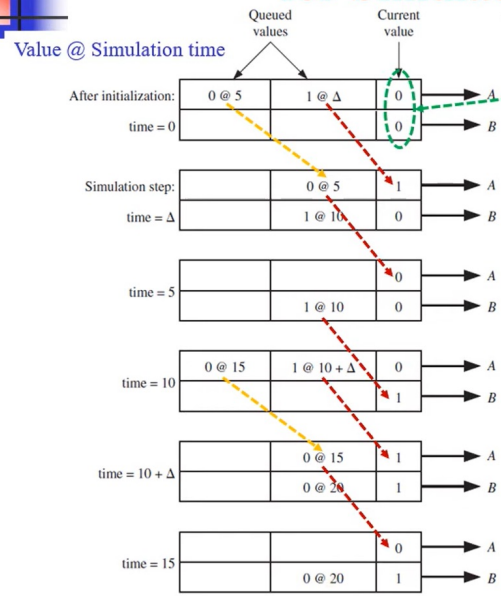
* The assigned value of *a* is deterministic because of ordering from **begin** to **end**.

- (b) nondetrminate

```
module nondeterminate;
   reg a;
   initial a = 0;
   always a <= #5 0;
   always a <= #5 1;
endmodule
```

* The assigned value of *a* is non-deterministic.

Example: Signal Drivers for Simulation




```
module twoprocess;
   reg A, B;

   initial
   begin
      A = 0;
      B = 0;
   end

   // process P1
   always @(B)
   begin
      A <= 1;
      A <= #5 0;
   end

   // process P2
   always @(A)
   begin
      if (A)
         B <= #10 ~B;
   end
endmodule
```

Simulation of Inertial Delays

* ***Inertial delay***, *T*:

Delay the input signals by time *T*, and reject any pulse w/ a width less than *T*.

- Simulation of ***inertial delays***:
  - Each input change causes the simulator to schedule a change, which is scheduled to occur after the ***specified delay.***
  - If another input change happens before the specified delay has elapsed, the 1st change is ***dequeued*** from the simulation driver queue.

    ⇒ Only pulses wider than the specified inertial delay appear at the output.