the usual way of doing assignments is

LHS = RHS;

there are however procedural continuous assignments

they override existing assignments

two types

- assign/deassign
- force/release

The **assign** and **deassign** constructs are now considered to be a bad coding style and it is recommended that alternative styles be used in Verilog HDL code.

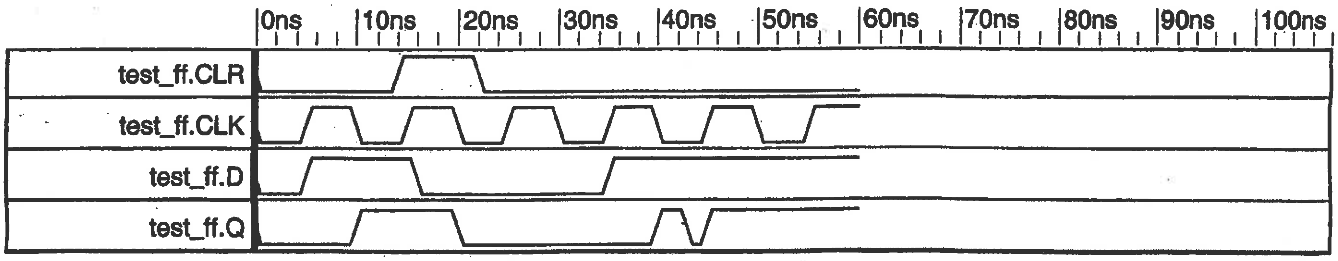
9.1.2 force and release

```
module stimulus;
...
...
//instantiate the d-flipflop
edge_dff dff(Q, Qbar, D, CLK, RESET);
...
...
initial
begin
    //these statements force value of 1 on dff.q between time 50 and
    //100, regardless of the actual output of the edge_dff.
    #50 force dff.q = 1'b1; //force value of q to 1 at time 50.
    #50 release dff.q;  //release the value of q at time 100.
end
...
...
endmodule
```

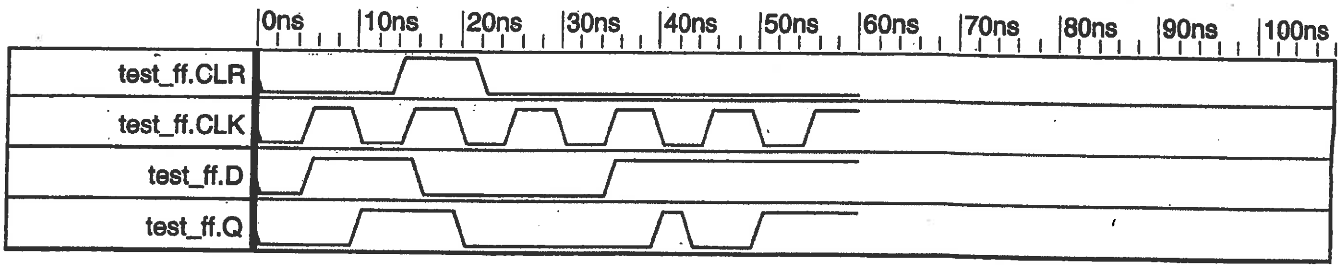
you can force/release nets

```
 ⋮
assign out = a&b&c;   // normal assignment

initial
   begin
     #50 force out = a|b&c;
     #20 release out;
   end
```

```
module test_ff;  // testbench
reg CLR,CLK,D;
wire Q;

// instantiate device

dff my_FF(.clr(CLR), .q(Q), .clock(CLK), .d(D));


initial
       CLR=0;

initial  // define stimuli
    begin
       D=0;
       wait(CLK);
       D=1;  // input for 1st clock
       wait(!CLK);
       #4 CLR = 1;
       wait(CLK);
       #1 D=0;  // input for 2nd clock
       wait(!CLK)
       #2 CLR=0;
       wait(CLK);
       #1 D=0;
       wait(!CLK);
       wait(CLK);
       D=1;
       wait(!CLK);
       wait(CLK);
       D=1;
    end

    initial
      begin
      #43 force my_FF.Q = 1'b0;   // "Q" declared as net in this file
       #2 release my_FF.Q;        // "Q" is the flip-flop output
      end

initial
       CLK=0;

always   // generate clock signal
       #5 CLK= ~CLK;

initial
      #60 $finish;

endmodule
```



```
module test_ff;  // testbench
reg CLR,CLK,D;
wire Q;


// instantiate device

dff my_FF(.clr(CLR), .q(Q), .clock(CLK), .d(D));


initial
      CLR=0;

initial  // define stimuli
    begin
      D=0;
      wait(CLK);
      D=1;  // input for 1st clock
      wait(!CLK);
      #4 CLR = 1;
      wait(CLK);
      #1 D=0;  // input for 2nd clock
      wait(!CLK)
      #2 CLR=0;
      wait(CLK);
      #1 D=0;
      wait(!CLK);
      wait(CLK);
      D=1;
      wait(!CLK);
      wait(CLK);
      D=1;
    end

    initial
      begin
      #43 force my_FF.q = 1'b0;  // "q" declared as reg in D FF template
       #2 release my_FF.q;
      end

initial
      CLK=0;

always   // generate clock signal
      #5 CLK= ~CLK;

initial
      #60 $finish;

endmodule
```



Module instance parameter values

the great thing about HDLS is
IP (i.e. module) reuse

```
module  Hello;
 parameter  id_num = 22;
        initial
            $display("id_num = %d", id_num);

 endmodule
```

```
 module top;
    Hello #(49) w1 ;  // pass by position
    Hello #(.id_num(49)) w2; // connect by name
  endmodule
```

************** EXAMPLE 1 ************

```
// define module with delays
module ALU;
          parameter delay1 =2;
          parameter delay2 = 3;
          parameter delay3 = 7;
          ....
          <module internals>
          .....
endmodule

// parameter assignment by position
ALU #(4, 5,6) b1( );  // delay1= 4, delay2 =5, delay3=6

// parameter assignment by name (no worrying about order)

ALU #(.delay2(5),  .delay3(6)) b2( );  // only some values changed
```

************** EXAMPLE 2 ************

```
module bus_master;

          parameter width=8;
          parameter speed=10;
          ....
          <module internals>
          .....
endmodule

bus_master #(.width(32)) bus_master( );
```

timescales in simulation

default time unit is (usually) 1 ns

Syntax: `timescale time-unit/time_precision

Notes: 1) Values must be 1, 10, 100

2) units can

s, ms, us, ns, ps, fs
(μs)

3) time-unit sets the delay units
time-precision sets the round off
units during simulation

e.g.

```
`timescale 1ns/100ps
    assign #5 A=B; // delay by 5ns

`timescale 100ns/1ns
    assign #5 A=B; // delay by 0.5μs
```

$display , $monitor, $strobe

suppose I want

temp = 3 @ t=10ns
temp = 7 @ t=20ns
temp = 9 @ t=30ns

$dislay in verilog ⇒ printf in C

$display prints to std out once

$monitor only has to in the code once

$strobe executes only after all assignments in the same time unit have executed

Example 9-11 Strobing

```
//Strobing
always @(posedge clock)
begin
   a = b;
   c = d;
end

always @(posedge clock)
   $strobe("Displaying a = %b, c = %b", a, c); // display values at
posedge
```

opening a file

syntax : `$fopen("file_name");`

returns an integer pointer to file_name

e.g.

```
integer ptr;
ptr = $fopen("my_file");
```

closing a file

syntax : `$fclose(file_ptr);`

e.g.

```
$fclose(ptr);
```

you write to files

```
$fdisplay(file-ptr, p1, p2, ....);
      or                          parameters
$fmonitor(file-ptr, p1, p2, ....);
```

reading from a file

```
$readmemb    // binary
$readmemh    // hex
```

Example 9-14 Initializing Memory

```
module test;

reg [7:0] memory[0:7]; //declare an 8-byte memory
integer i;

initial
begin
  //read memory file init.dat. address locations given in memory
  $readmemb("init.dat", memory);
  //display contents of initialized memory
  for(i=0; i < 8; i = i + 1)
    $display("Memory [%0d] = %b", i, memory[i]);
end

endmodule
```

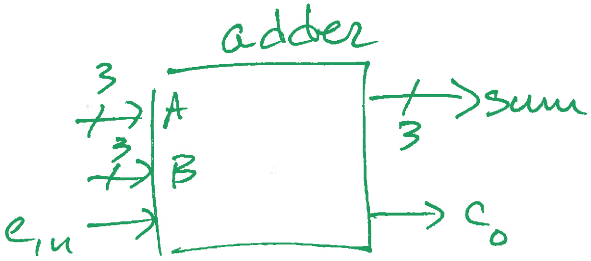
init.dat

```
@002
11111111 01010101
00000000 10101010

@006
1111zzzz 00001111
```

When the test module is simulated, we will get the following output:

```
Memory [0] = xxxxxxxx
Memory [1] = xxxxxxxx
Memory [2] = 11111111
Memory [3] = 01010101
Memory [4] = 00000000
Memory [5] = 10101010
Memory [6] = 1111zzzz
Memory [7] = 00001111
```

e.g. (3-bit adder)



assume "test.vec" contains

$$\underbrace{\begin{matrix}0\,1\,0\\0\,1\,0\end{matrix}}_{A}\;\vdots\;\underbrace{\begin{matrix}0\,1\,0\\0\,1\,1\end{matrix}}_{B}\;\vdots\;\underbrace{\begin{matrix}0\\1\end{matrix}}_{C_{in}}\;\vdots\;\underbrace{\begin{matrix}1\,0\,0\\1\,1\,0\end{matrix}}_{\text{expected sum}}\;\vdots\;\begin{matrix}0\\0\end{matrix}\xleftarrow{}\text{expected } C_o$$

```
reg [11:1] Vmem[2:1];

// instantiate Adder

   adder_3bit F1 (A, B, CIN, SUM, cout);

initial
   begin
      $readmemb ("test.vec", Vmem);

      for (j=1 ; j<=2 ; j=j+1)
        begin
          {A, B, CIN, SUM_EX, cout_EX} = Vmem[j];

if ((sum !== sum_EX) || (cout !== cout_EX))
                  $display ("mismatch vector %d",
                                                   VMEM[J]);

        end
```

Value Change Dump (VCD) files

Verilog equivalent of a source level debugger

Defn (VCD file)

an ASCII file that stores timestamped signal changes. It is used by postprocessing design automation tools

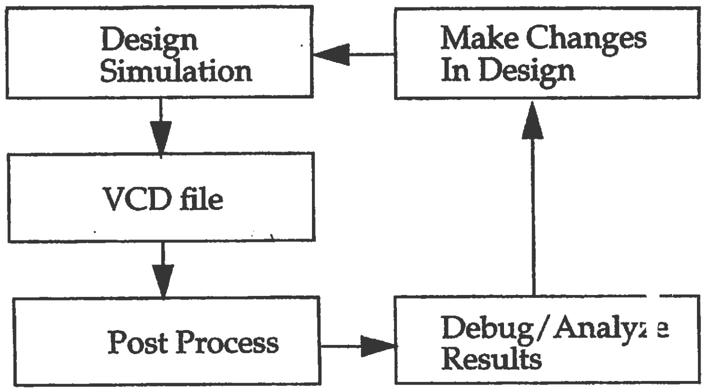


Figure 9-1 *Debugging and Analysis of Simulation with VCD file*

```
//specify name of VCD file. Otherwise,default name is
//assigned by the simulator.
initial
      $dumpfile("myfile.dmp"); //Simulation info dumped to myfile.dmp

//Dump signals in a module
initial
   $dumpvars; //no arguments, dump all signals in the design
initial
   $dumpvars(1, top); //dump variables in module instance top.
                 //Number 1 indicates levels of hierarchy. Dump one
                 //hierarchy level below top,i.e.,dump variables in top,
                 //but not signals in modules instantiated by top.
initial
   $dumpvars(2, top.m1);//dump up to 2 levels of hierarchy below top.m1
initial
   $dumpvars(0, top.m1);//Number 0 means dump the entire hierarchy
                       // below top.m1

//Start and stop dump process
initial
begin
   $dumpon;              //start the dump process.
   #100000 $dumpoff;   //stop the dump process after 100,000 time units
end

//Create a checkpoint. Dump current value of all VCD variables
initial
   $dumpall;
```

example 9-15

other VCD File system tasks

1) $dumplimit(size);

↑ decimal # in bytes

2) $dumpvars(0, sig1, sig2, . . . , sigK);

reg'd

which signals get stored in VCD file

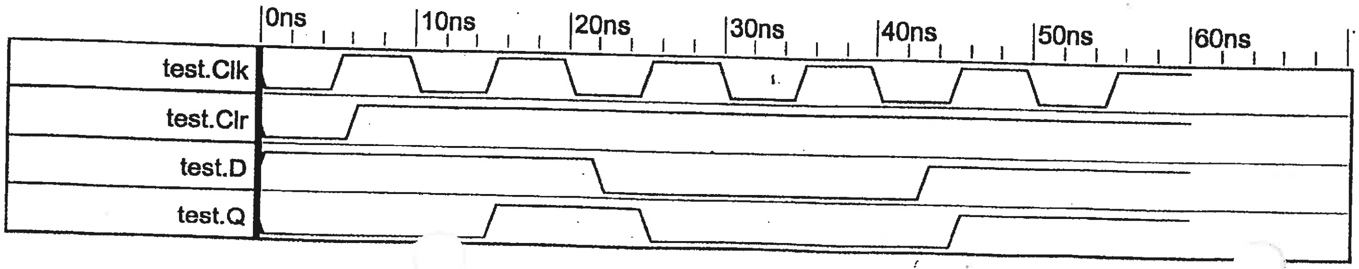
e.g.

$dumpvars(0, div.clk);

store variable (or signal) "clk" from module "div" data_test.vcd

```
$date
      Sat May 10 16:06:36 2003

$end
$version
      VeriLogger 9.0g
$end
$timescale
      1 ps
$end


$scope module test $end
$var reg       1 !    Clk  $end
$var reg       1 "    Clr  $end
$var reg       1 #    D  $end
$var wire      1 $    Q  $end
$upscope $end

$enddefinitions      $end
$dumpvars
0$
1#
0"
0!
$end
#5000
1!
#6000
1"
#10000
0!
#15000
1!
1$
#20000
0!
#22000
0#
#25000
1!
0$
#30000
0!
#35000
1!
#40000
0!
#43000
1#
#45000
1!
1$
#50000
0!
#55000
1!
```

$dumpvars(1,test);



```
$date
      Sat May 10 16:02:30 2003

$end
$version
      VeriLogger 9.0g
$end
$timescale
      1 ps
$end


$scope module test $end
$var reg       1 !    Clk  $end
$var reg       1 "    Clr  $end
$var reg       1 #    D  $end
$var wire      1 $    Q  $end

$scope module gg $end
$var wire      1 %    d  $end
$var wire      1 &    clr  $end
$var wire      1 '    clk  $end
$var reg       1 (    q  $end
$upscope $end

$upscope $end

$enddefinitions      $end
$dumpvars
0(
0'
0&
1%
0$
1#
0"
0!
$end
#5000
1!
1'
#6000
1"
1&
#10000
0!
0'
#15000
1!
1'
1(
1$
#20000
0!
0'
#22000
0#
0%
#25000
1!
1'
0(
0$
#30000
0!
```

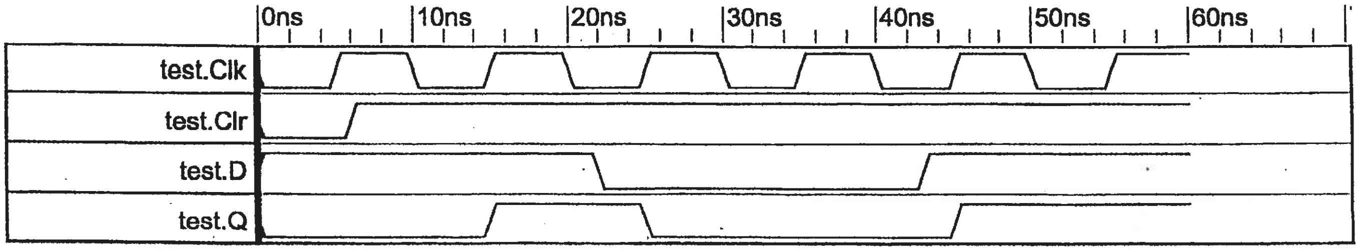
testbench

instantiated D-FF

— 5 ns
— CLK=1 (module test)
CLK=1 (module gg)

```
0'
#35000
1!
1'
#40000
0!
0'
#43000
1#
1%
#45000
1!
1'
1(
1$
#50000
0!
0'
#55000
1!
1'
```

$dumpvars(0, test);


0ns
10ns
20ns
30ns
40ns
50ns
60ns
test.Clk
test.Clr
test.D
test.Q