module hdl1;

integer A, B, C;
initial begin
  A = 3;
  B = 10;
  $display( A, B, C );
  C = A+B;
  $display( A, B, C );
  for ( A = 3 ; A > 0 ; A = A-1 )
    begin
      C = C*B;
      $display( A, B, C );
    end
end
endmodule
SystemVerilog HDL - a programming language

Output

Compiling source file "hdl1.sv"

Highest level modules:
hd1

3 10 x
3 10 13
3 10 130
2 10 1300
1 10 13000

Notes

• Operators include
  + - / * != == > < >= <= && || !

• Constructs include
  if else while repeat for case

• C is initially undefined: x
module hdl2;

timeunit 1ns; timelprecision 10ps;

logic [7:0] A, B, C, E; logic D;

initial
begin
  A = 10;
  #2.5ns B = 3;
  #2.5ns C = A + B;
  #4 D = A + B;
  #333ps E = {A[3:0],A[7:4]} + (B << 1);
end

initial
$monitor( A,"",B,"",C,"",D,"",E,"" @time ",,$realtime," ns");
endmodule
SystemVerilog HDL - for hardware description

Output

10  x  x  x  x  @time 0 ns
10  3  x  x  x  @time 2.5 ns
10  3  13  x  x  @time 5 ns
10  3  13  1  x  @time 9 ns
10  3  13  1  166  @time 9.33 ns

Notes

• Time is cumulative.

• `timeunit 1ns  #4` is interpreted as `#4ns`

• `timeprecision 10ps  #333ps` is rounded to `#0.33ns`

• `$monitor` monitors all changes in specified variables

• `logic [n-1:0]` declares an n-bit variable

  when A+B is assigned to D only the least significant bit is stored.

• Bit Manipulation Operators include: `[:]  {}  <<  <<<  >>  >>>  &  |  ^  ~ `
SystemVerilog HDL - for hardware description

Waveforms for hdl2

A: 10 10 10
B: 3 3
C: 13 13
D: 10
E: 166
module hdl3;
timeunit 1ns; timeprecision 1ns; logic [7:0] A, B, C;

initial
  begin
    A = 1; B = 5;
    $display("%d %d %d @ %.2f", A, B, C, $realtime);
    #100 C = A * B;
    $display("%d %d %d @ %.2f", A, B, C, $realtime);
  end

initial
  #30 A = 3;

always
  #15 B = B+1;
endmodule
SystemVerilog HDL - concurrency

Output

1  5  x  @ 0.00
3  11  33  @ 100.00

Notes

• The three procedural blocks operate concurrently.
• begin and end are optional where only one statement exists within a block.
• This example doesn’t terminate.
• $display supports formatted output.
  %b  %d  %x  %s  %f
module hdl4;
timeunit 1ns; timeprecision 1ns; logic [7:0] A, B, C;

initial
    begin
        A = 1; B = 5;
        #100 C = A * B;
    end
initial #30 A = 3;
always #15 B = B+1;

initial
    begin
        $monitor("%d %d %d @ %.2f", A, B, C, $realtime);
        #115 $finish;
    end
endmodule
SystemVerilog HDL - concurrency

Output

1 5 x @ 0.00
1 6 x @ 15.00
3 7 x @ 30.00
3 8 x @ 45.00
3 9 x @ 60.00
3 10 x @ 75.00
3 11 x @ 90.00
3 11 33 @ 100.00
3 12 33 @ 105.00

Notes

- `$monitor` allows us to track all variable changes.
- `$finish` is used to force the simulation to end.
SystemVerilog HDL - concurrency

Waveforms for hdl3/hdl4
module hdl5;
timeunit 1ns; timeprecision 1ns;
logic Clear; logic [2:0] Count;

initial begin Clear = 1; #25 Clear = 0; end

always
    #10
    if (Clear == 1) Count = 0; else Count = Count + 1;

initial
begin
    $display("Clear Count @time");
    $monitor(" %b %b (%d) %5.1f",Clear,Count,Count,$realtime);
    #115 $finish;
end
endmodule
SystemVerilog HDL - modelling

Output

<table>
<thead>
<tr>
<th>Clear</th>
<th>Count</th>
<th>@time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>xxx</td>
<td>(x)</td>
</tr>
<tr>
<td>1</td>
<td>000</td>
<td>(0)</td>
</tr>
<tr>
<td>0</td>
<td>000</td>
<td>(0)</td>
</tr>
<tr>
<td>0</td>
<td>001</td>
<td>(1)</td>
</tr>
<tr>
<td>0</td>
<td>010</td>
<td>(2)</td>
</tr>
<tr>
<td>0</td>
<td>011</td>
<td>(3)</td>
</tr>
<tr>
<td>0</td>
<td>100</td>
<td>(4)</td>
</tr>
<tr>
<td>0</td>
<td>101</td>
<td>(5)</td>
</tr>
<tr>
<td>0</td>
<td>110</td>
<td>(6)</td>
</tr>
<tr>
<td>0</td>
<td>111</td>
<td>(7)</td>
</tr>
<tr>
<td>0</td>
<td>000</td>
<td>(0)</td>
</tr>
<tr>
<td>0</td>
<td>001</td>
<td>(1)</td>
</tr>
</tbody>
</table>

Notes
Here a simple counter is modelled. The counter increments when Clear is not asserted. The three bit count rolls over from 7 to 0 since 8 is not valid.
SystemVerilog HDL - modelling

Waveforms for hdl5
module hd16;

timeunit 1ns; timeprecision 1ns;

logic Clear, Clock; logic [2:0] Count;

initial begin
    Clear = 0; #17 Clear = 1; #10 Clear = 0; end
always begin
    Clock = 1; #5 Clock = 0; #5 Clock = 1; end

always @(posedge Clock)
    if (Clear == 1) Count = 0; else Count = Count + 1;

initial begin
    $display("Clear Count @time");
    $monitor(" %b %b (%d) %5.1f",Clear,Count,Count,$realtime);
    #115 $finish;
end
endmodule
SystemVerilog HDL - modelling

Output

<table>
<thead>
<tr>
<th>Clear</th>
<th>Count</th>
<th>@time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xxx</td>
<td>(x)</td>
</tr>
<tr>
<td>1</td>
<td>xxx</td>
<td>(x)</td>
</tr>
<tr>
<td>1 000</td>
<td>(0)</td>
<td>20.0</td>
</tr>
<tr>
<td>0 000</td>
<td>(0)</td>
<td>27.0</td>
</tr>
<tr>
<td>0 001</td>
<td>(1)</td>
<td>30.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 111</td>
<td>(7)</td>
<td>90.0</td>
</tr>
<tr>
<td>0 000</td>
<td>(0)</td>
<td>100.0</td>
</tr>
<tr>
<td>0 001</td>
<td>(1)</td>
<td>110.0</td>
</tr>
</tbody>
</table>

Notes

We can wait on a signal rather than waiting for a time.

- @(signal)
- @(negedge signal)
- @(signal1, signal2)
Since Clear is synchronous, Count goes to zero on the rising edge of Clock once Clear is high.
module hdl7;

timeunit 1ns; timeprecision 1ns;
logic nReset, Clock; logic [2:0] Count;

initial begin
    nReset = 1; #17 nReset = 0; #10 nReset = 1;
end

always begin
    Clock = 1; #5 Clock = 0; #5 Clock = 1;
end

always @(posedge Clock, negedge nReset)
    if (nReset == 0) Count = 0; else Count = Count + 1;

initial
    begin
        $display("nReset Count @time");
        $monitor(" %b %b (%d) %5.1f",nReset,Count,Count,$realtime);
        #115 $finish;
    end
endmodule

1017
## SystemVerilog HDL - modelling

### Output

<table>
<thead>
<tr>
<th>nReset</th>
<th>Count</th>
<th>@time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>xxx (x)</td>
<td>0.0</td>
</tr>
<tr>
<td>0</td>
<td>000 (0)</td>
<td>17.0</td>
</tr>
<tr>
<td>1</td>
<td>000 (0)</td>
<td>27.0</td>
</tr>
<tr>
<td>1</td>
<td>001 (1)</td>
<td>30.0</td>
</tr>
<tr>
<td>1</td>
<td>010 (2)</td>
<td>40.0</td>
</tr>
<tr>
<td>1</td>
<td>011 (3)</td>
<td>50.0</td>
</tr>
<tr>
<td>1</td>
<td>100 (4)</td>
<td>60.0</td>
</tr>
<tr>
<td>1</td>
<td>101 (5)</td>
<td>70.0</td>
</tr>
<tr>
<td>1</td>
<td>110 (6)</td>
<td>80.0</td>
</tr>
<tr>
<td>1</td>
<td>111 (7)</td>
<td>90.0</td>
</tr>
<tr>
<td>1</td>
<td>000 (0)</td>
<td>100.0</td>
</tr>
<tr>
<td>1</td>
<td>001 (1)</td>
<td>110.0</td>
</tr>
</tbody>
</table>

### Notes

- includes active low asynchronous nReset
Since nReset is asynchronous, Count goes to zero on the falling edge of nReset.
module hdl8_stim;

timeunit 1ns; timeprecision 1ns;

logic nReset, Clock;
wire [2:0] Count;

initial begin
nReset = 1; #17 nReset = 0; #10 nReset = 1; end
always begin
Clock = 1; #5 Clock = 0; #5 Clock = 1; end

hdl8 unit1(.Count, .Clock, .notReset(nReset));

initial
begin
    $display("nReset Count @time");
    $monitor("%b %b (%d) %5.1f",nReset,Count,Count,$realtime);
    #115ns $finish;
end
endmodule
SystemVerilog HDL - hierarchy

Notes

- **hd18_stim** will be the top level module in the hierarchy.
- **hd18_stim** contains the stimulus and monitoring information.
- This module calls an instance of the counter module, **hd18**. The name of this instance is **unit1**.
- **nReset** and **Clock** are generated here and passed to the counter module. They are declared to be of type **var (logic ≡ var logic)**.
- **Count** is generated elsewhere (in the counter module) and is declared to be of type **wire (wire ≡ wire logic)**.
- Port connection is by name, indicated by this format: 
  .<signal_name>
- Since the names of the two reset signals do not match, we use a different format for the connection: 
  .<sub_module_port_name>(<our_name>)
module hdl8(
    output logic [2:0] Count,
    input Clock,
    input notReset
);

timeunit 1ns; timeprecision 1ns;

always_ff @(posedge Clock, negedge notReset)
    if (notReset == 0)
        Count = 0;
    else
        Count = Count + 1;

endmodule
module hdl8(
    output logic [2:0] Count,
    input Clock,
    input notReset
);

timeunit 1ns; timeprecision 1ns;

always_ff @(posedge Clock, negedge notReset)
    if (notReset == 0)
        Count <= 0;
    else
        Count <= Count + 1;
endmodule
Notes

- `hdl8` contains the model of the counter.
- `hdl8` is synthesizable.
- `notReset` and `Clock` are generated elsewhere and are declared to be of type `wire (input ≡ input wire logic)`.
- `Count` is generated here and passed to the parent module `module`. It is declared to be of type `variable (output logic ≡ output var logic)`.
- `always_ff` simulates the same as `always` but hints to the synthesis program that synchronous sequential logic is intended.
- `<=` indicates a *non-blocking assignment* and is used by default within `always_ff` blocks.
SystemVerilog HDL - hierarchy

Waveforms for hdl8

```
0 1 2 3 4 5 6 7 0 10
unit1.Count
unit1.Clock
unit1.notReset

0 1 2 3 4 5 6 7 0 10
100ns 50ns 0
```

1024
module hdl9_stim;

timeunit 1ns; timeprecision 1ns;
logic nReset, Clock;
wire Max; wire [2:0] Count;

initial begin nReset = 1; #17 nReset = 0; #10 nReset = 1; end
always begin Clock = 1; #5 Clock = 0; #5 Clock = 1; end

hdl9 unit1(Max, Count, Clock, nReset);

initial
begin
    $display(" Count Max @time");
    $monitor("%b (%d) %b %5.1f",Count,Count,Max,$realtime);
    #115 $finish;
end
endmodule
module hdl9(
    output logic Max,
    output logic [2:0] Count,
    input Clock,
    input notReset
);

timeunit 1ns; timeprecision 1ns;


always_ff @(posedge Clock, negedge notReset)
    if (notReset == 0)
        Count <= 0;
    else
        Count <= Count + 1;
endmodule
### Output

<table>
<thead>
<tr>
<th>Count</th>
<th>Max</th>
<th>@time</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxx (x)</td>
<td>x</td>
<td>0.0</td>
</tr>
<tr>
<td>000 (0)</td>
<td>0</td>
<td>17.0</td>
</tr>
<tr>
<td>001 (1)</td>
<td>0</td>
<td>30.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>110 (6)</td>
<td>0</td>
<td>80.0</td>
</tr>
<tr>
<td>111 (7)</td>
<td>1</td>
<td>90.0</td>
</tr>
<tr>
<td>000 (0)</td>
<td>0</td>
<td>100.0</td>
</tr>
<tr>
<td>001 (1)</td>
<td>0</td>
<td>110.0</td>
</tr>
</tbody>
</table>

### Notes

- **Continuous Assignment**
  The `assign` statement offers one way to model combinational logic.

- **Port connection by position in ordered list**
  An alternative to connection by name – should be used with care.
Waveforms for hdl9

- nReset
- Clock
- Count
- Max
- unit1.notReset
- unit1.Clock
- unit1.Count
- unit1.Max

Time: 0 50ns 100ns