

SystemVerilog HDL - a programming language

```
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:

hdl1

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

SystemVerilog HDL - for hardware description

```
module hdl2;

timeunit 1ns; timeprecision 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

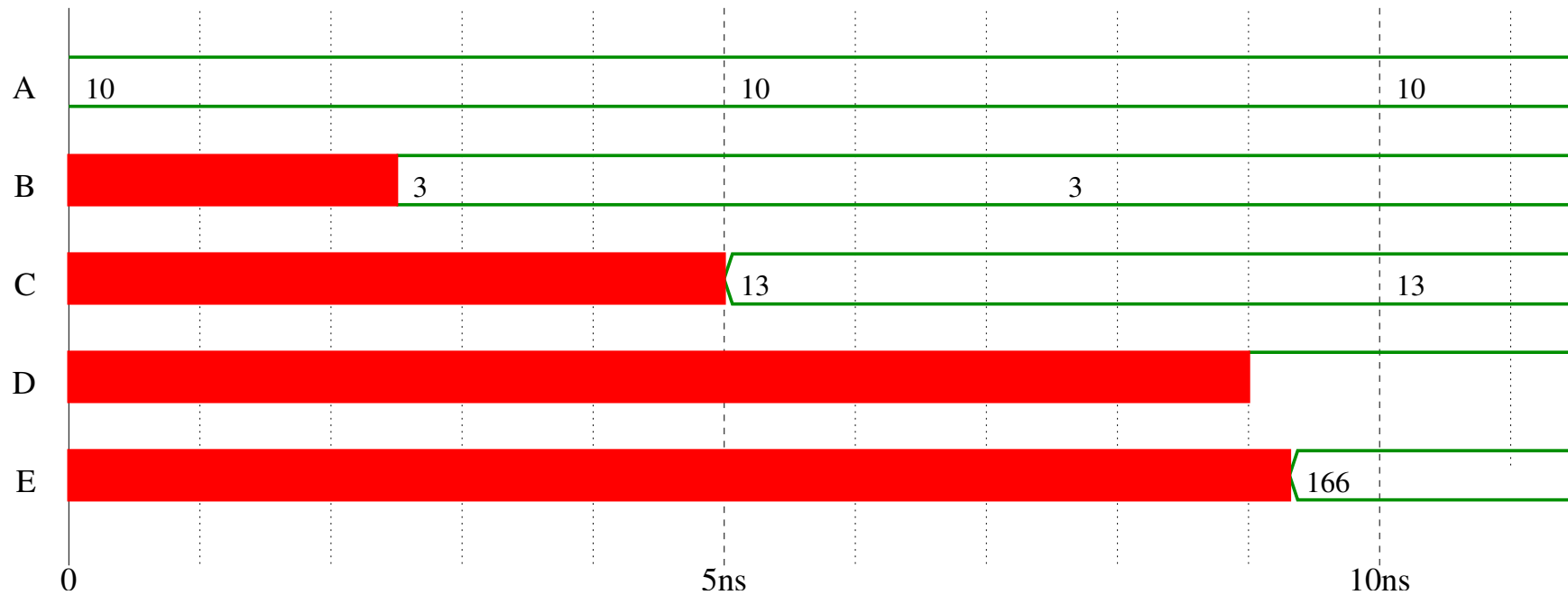
```
10      x      x x      x      @time 0 ns
10      3      x x      x      @time 2.5 ns
10      3    13 x      x      @time 5 ns
10      3    13 1      x      @time 9 ns
10      3    13 1 166      @time 9.33 ns
```

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



SystemVerilog HDL - concurrency

```
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
```

SystemVerilog HDL - concurrency

```
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

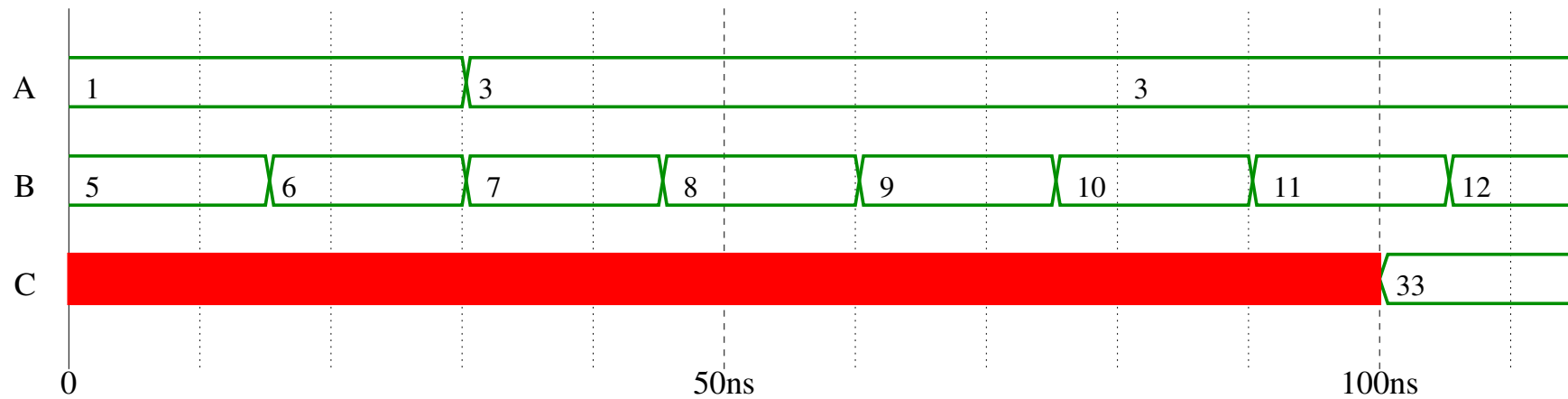
```
1      5      x  @  0.00
1      6      x  @ 15.00
3      7      x  @ 30.00
3      8      x  @ 45.00
3      9      x  @ 60.00
3     10      x  @ 75.00
3     11      x  @ 90.00
3     11     33  @ 100.00
3     12     33  @ 105.00
```

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



SystemVerilog HDL - modelling

```
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

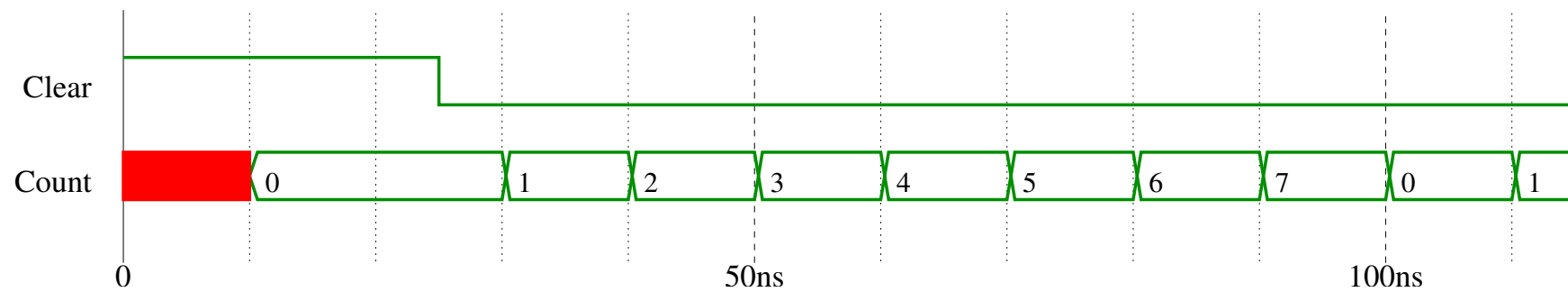
Clear	Count	@time
1	xxx (x)	0.0
1	000 (0)	10.0
0	000 (0)	25.0
0	001 (1)	30.0
0	010 (2)	40.0
0	011 (3)	50.0
0	100 (4)	60.0
0	101 (5)	70.0
0	110 (6)	80.0
0	111 (7)	90.0
0	000 (0)	100.0
0	001 (1)	110.0

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



SystemVerilog HDL - modelling

```
module hdl6;
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

```
Clear Count  @time
0   xxx  (x)   0.0
1   xxx  (x)  17.0
1   000  (0)  20.0
0   000  (0)  27.0
0   001  (1)  30.0
    ...
0   111  (7)  90.0
0   000  (0) 100.0
0   001  (1) 110.0
```

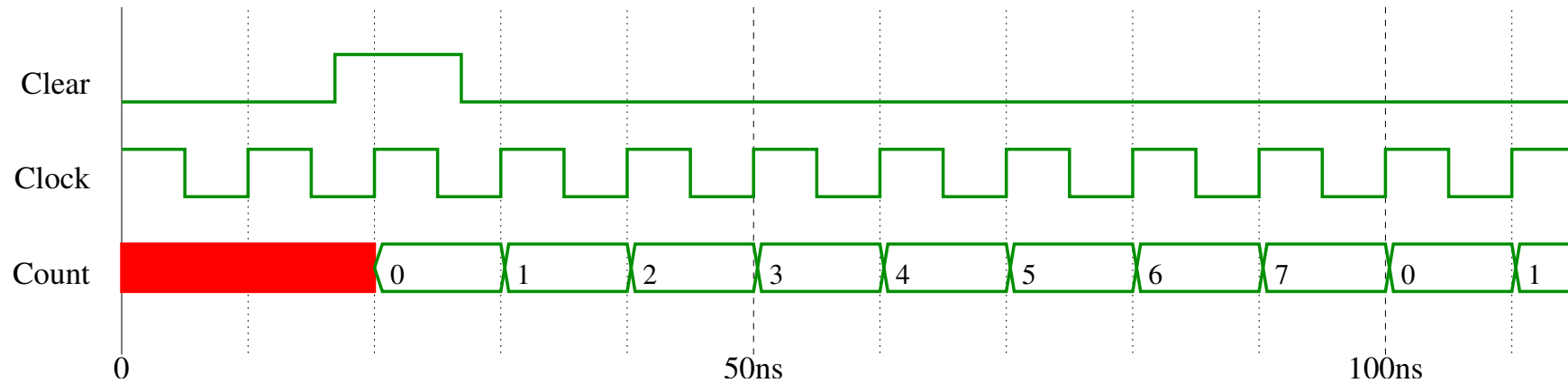
Notes

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

- @(signal)
- @(negedge signal)
- @(signal1, signal2)

SystemVerilog HDL - modelling

Waveforms for hdl6



Since Clear is synchronous, Count goes to zero on the rising edge of Clock once Clear is high.

SystemVerilog HDL - modelling

```
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
```

SystemVerilog HDL - modelling

Output

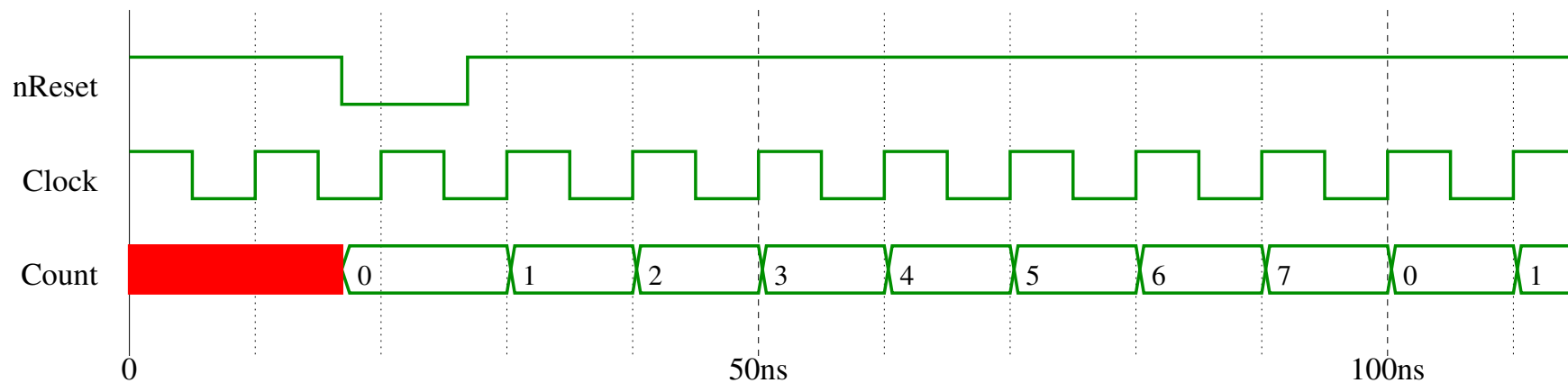
nReset	Count	@time
1	xxx (x)	0.0
0	000 (0)	17.0
1	000 (0)	27.0
1	001 (1)	30.0
1	010 (2)	40.0
1	011 (3)	50.0
1	100 (4)	60.0
1	101 (5)	70.0
1	110 (6)	80.0
1	111 (7)	90.0
1	000 (0)	100.0
1	001 (1)	110.0

Notes

- includes active low asynchronous nReset

SystemVerilog HDL - modelling

Waveforms for hdl7



Since nReset is asynchronous, Count goes to zero on the falling edge of nReset

SystemVerilog HDL - hierarchy

```
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(count), .Clock(clock), .nReset(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

- `hdl8_stim` will be the top level module in the hierarchy.
- `hdl8_stim` contains the stimulus and monitoring information.
- This module calls an instance of the counter module, `hdl8`. The name of this instance is `unit1`.
- `nreset` and `clock` are generated here and passed to `nReset` and `Clock` in the counter module. They are declared to be of type `var` (`logic ≡ var logic`).
- `count` is generated elsewhere (as `Count` in the counter module) and is declared to be of type `wire` (`wire ≡ wire logic`).

SystemVerilog HDL - hierarchy

```
module hdl8(  
    output logic [2:0] Count,  
    input Clock,  
    input nReset  
);  
  
timeunit 1ns; timeprecision 1ns;  
  
    always_ff @(posedge Clock, negedge nReset)  
        if (nReset == 0)  
            Count = 0;  
        else  
            Count = Count + 1;  
  
endmodule
```

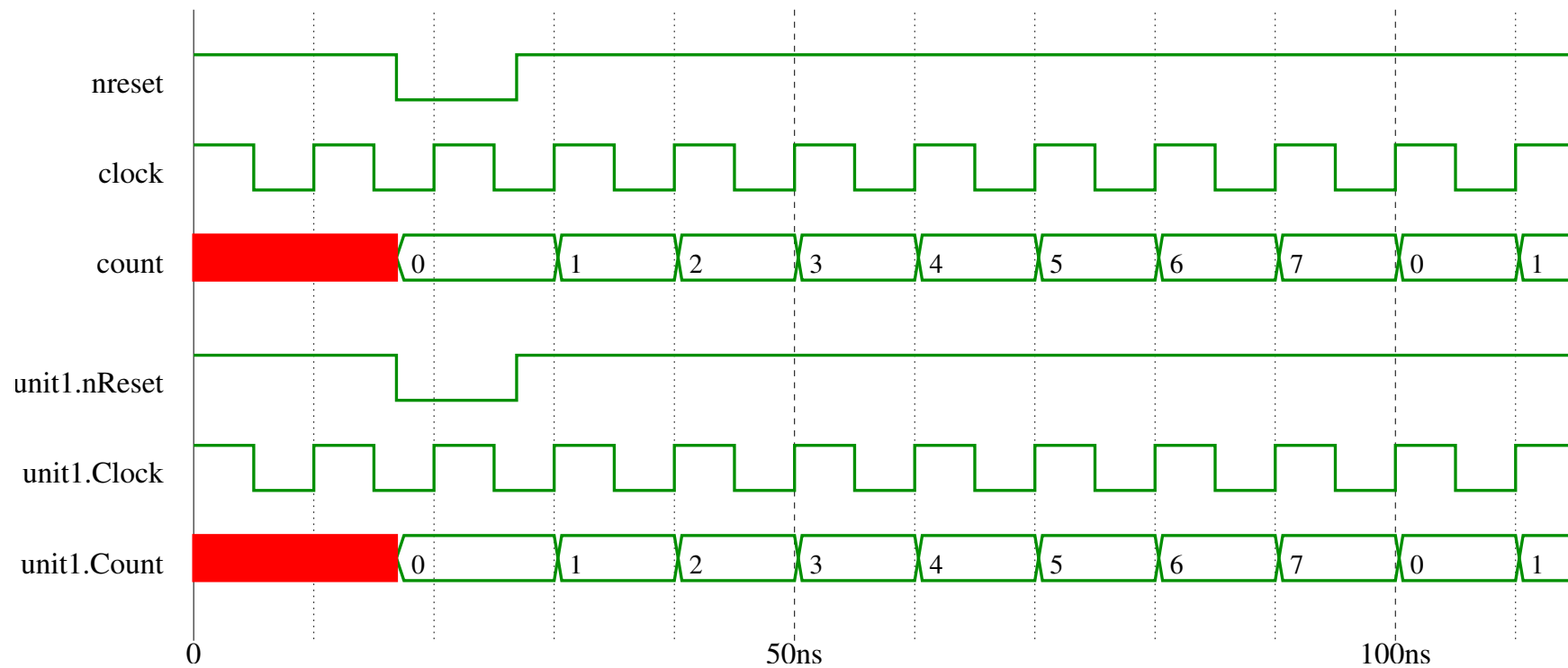
SystemVerilog HDL - hierarchy

Notes

- `hdl8` contains the model of the counter.
- `hdl8` is synthesizable.
- `nReset` and `Clock` are generated elsewhere and are declared to be of type `wire` (`input  $\equiv$  input wire logic`).
- `Count` is generated here and passed to the parent module module. It is declared to be of type `variable` (`output logic  $\equiv$  output var logic`).
- `always_ff` simulates the same as `always` but hints to the synthesis program that synchronous sequential logic is intended.

SystemVerilog HDL - hierarchy

Waveforms for hdl8



SystemVerilog HDL - hierarchy

```
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
```

SystemVerilog HDL - hierarchy

```
module hdl9(  
    output logic Max,  
    output logic [2:0] Count,  
    input Clock,  
    input nReset  
);  
timeunit 1ns; timeprecision 1ns;  
  
    assign Max = Count[2] && Count[1] && Count[0];  
  
    always_ff @(posedge Clock, negedge nReset)  
        if (nReset == 0)  
            Count = 0;  
        else  
            Count = Count + 1;  
  
endmodule
```

SystemVerilog HDL - hierarchy

Output

count	max	@time
xxx (x)	x	0.0
000 (0)	0	17.0
001 (1)	0	30.0
.....		
110 (6)	0	80.0
111 (7)	1	90.0
000 (0)	0	100.0
001 (1)	0	110.0

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.

SystemVerilog HDL - hierarchy

Waveforms for hdl9

