Verilog 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
```

Verilog HDL - a programming language

Output

```
Compiling source file "hdl1.v"
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`

Verilog HDL - for hardware description

```
`timescale 1ns / 10ps
module hdl2;

reg [7:0] A, B, C;
reg D;

initial
  begin
    A = 3;    B = 10;
    $display( A, B, C, D, "   @time ", $realtime, " ns");
    #50
    C = A+B;
    $display( A, B, C, D, "   @time ", $realtime, " ns" );
    #25.495
    D = A+B;
    $display( A, B, C, D, "   @time ", $realtime, " ns" );
  end
endmodule
```

Output

```
3 10  x x   @time 0 ns
3 10 13 x   @time 50 ns
3 10 13 1   @time 75.5 ns
```

Notes

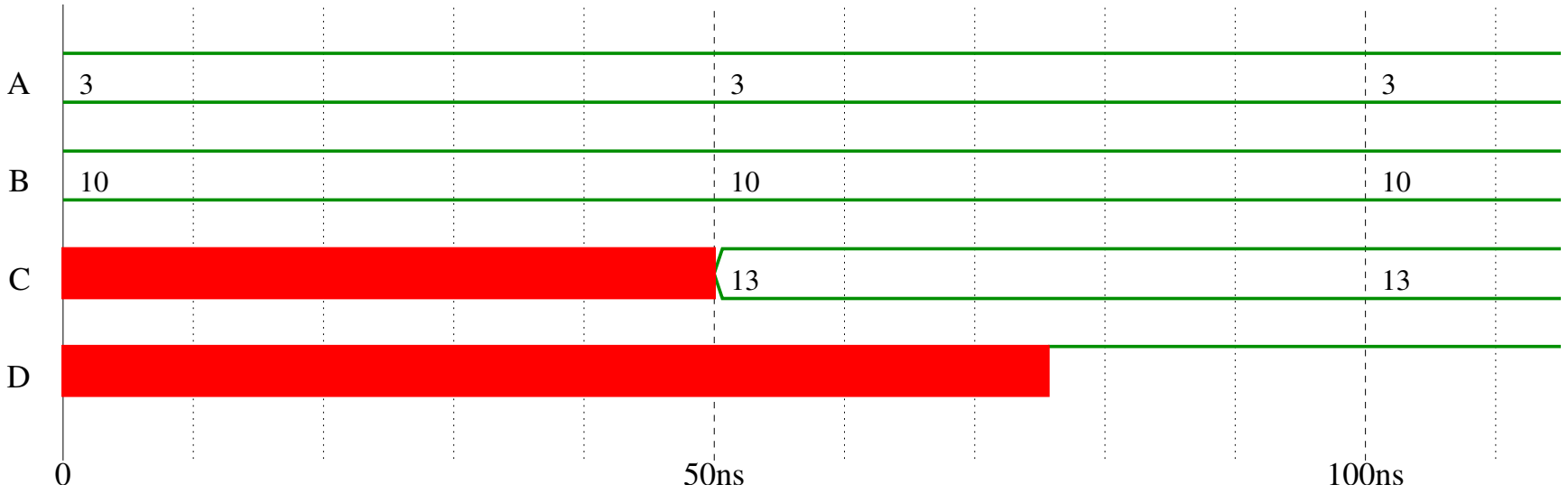
- Time is cumulative.
- `timescale 1ns / 10ps`

  The time units are nanoseconds with a resultion of 10ps hence 25.495 ns is rounded to 25.50 ns.
- `reg`isters

  D is a single bit register while the rest are 8 bit registers.

  When 13 is assigned to D only the least significant bit is stored.

Waveforms for hdl2



Verilog HDL - concurrency

```
`timescale 1ns / 1ns

module hdl3;
reg [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
  begin
    #15 B = B+1;
  end
endmodule
```

Verilog 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`

Verilog HDL - concurrency

```
`timescale 1ns / 1ns

module hdl4;
reg [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
```

Verilog HDL - concurrency

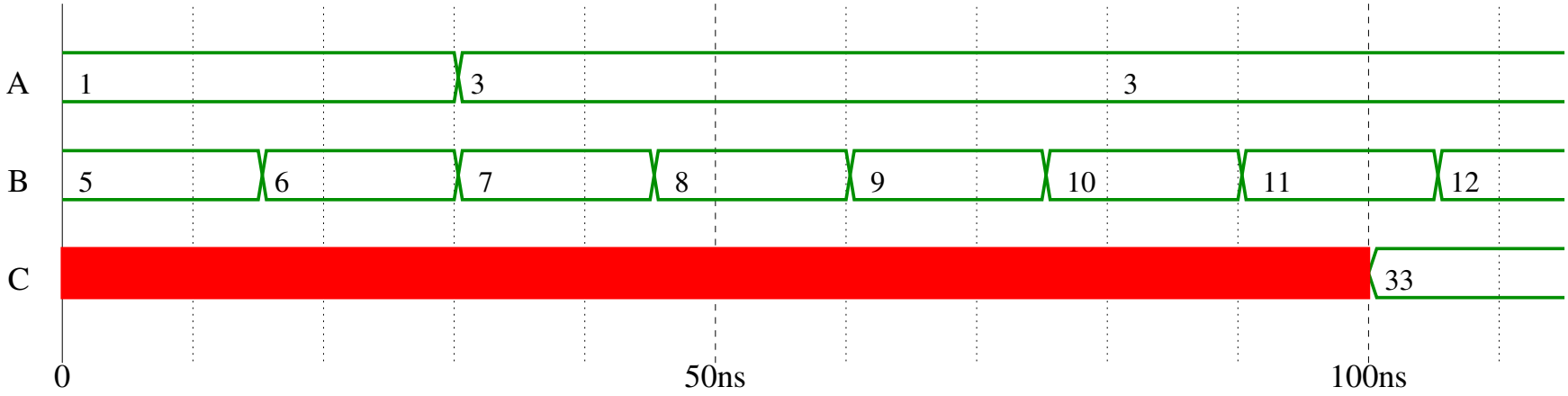
Output

```
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.

Waveforms for hdl3/hdl4



Verilog HDL - modelling

```
`timescale 1ns / 1ns

module hdl5;
reg Clear; reg [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
```

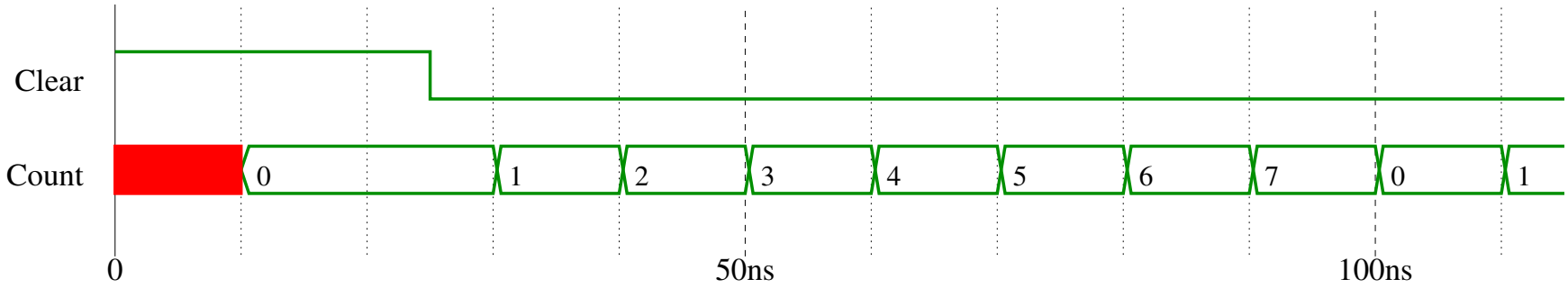
Output

```
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.

Waveforms for hdl5



Verilog HDL - modelling

```
`timescale 1ns / 1ns

module hdl6;
reg Clear, Clock; reg [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
```

Verilog HDL - modelling

Output

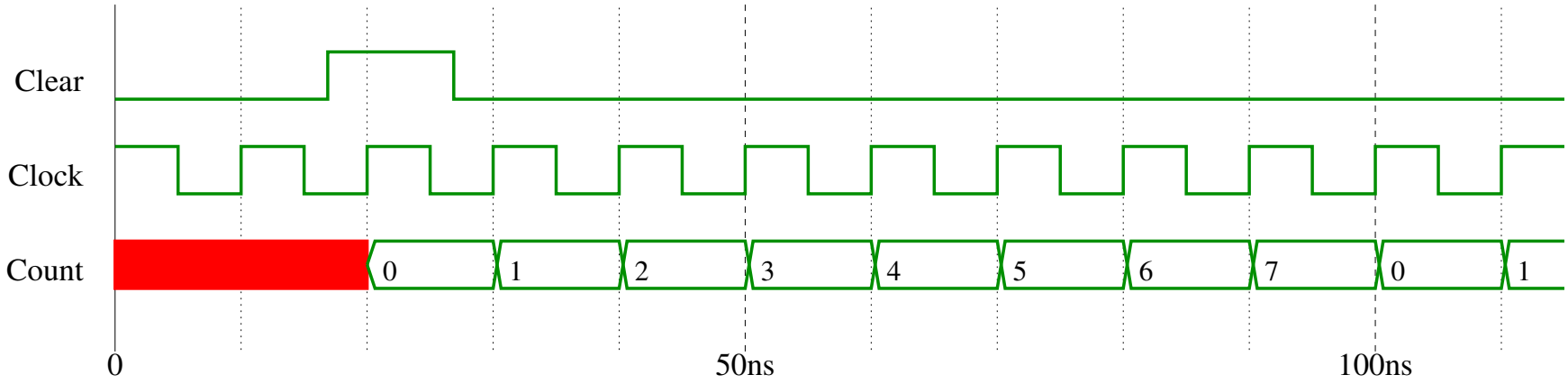
```
Clear Count  @time
  0  xxx (x)   0.0
  1  xxx (x)  17.0
  1  000 (0)  20.0
  0  000 (0)  27.0
  0  001 (1)  30.0
       ...
  0  111 (7)  90.0
  0  000 (0) 100.0
  0  001 (1) 110.0
```

Notes

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

- `@(signal)`
- `@(negedge signal)`
- `@(signal1 or signal2)` – n.b. `or` is not the same as `||`.

Waveforms for hdl6



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

Verilog HDL - modelling

```
`timescale 1ns / 1ns

module hdl7;
reg nReset, Clock; reg [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 or 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
```

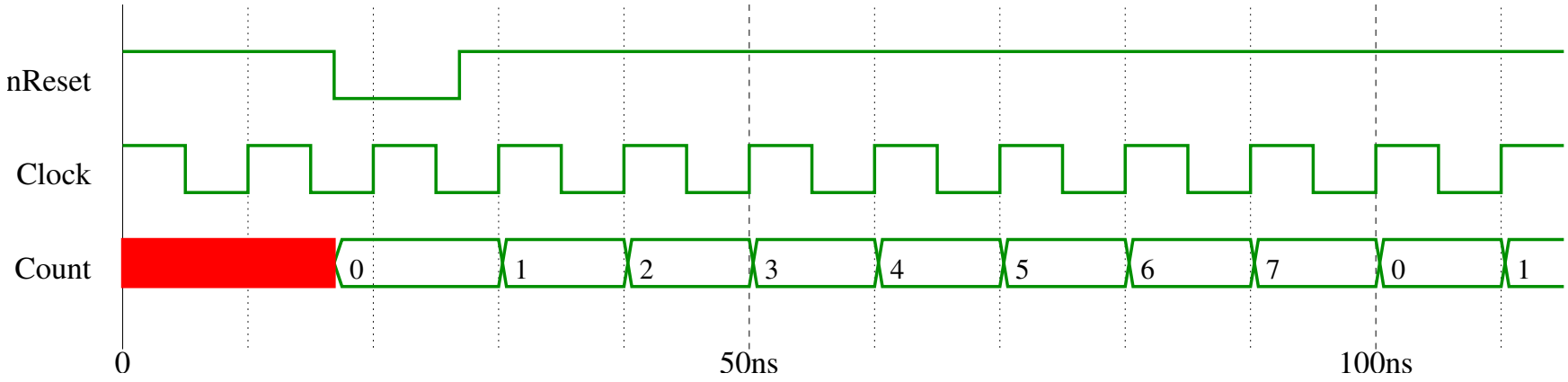
Output

```
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

Waveforms for hdl7



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

Verilog HDL - hierarchy

```
`timescale 1ns / 1ns
module hdl8_stim;

reg 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);
    #115 $finish;
  end
endmodule
```

Verilog 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 here as `reg`isters.
- `count` is generated elsewhere (as `Count` in the counter module) and so must be declared here as `wire`.

  Each signal may be passed as a wire through many modules but must only exist as a register in its source module.

Verilog HDL - hierarchy

```
`timescale 1ns / 1ns
module hdl8(Count, Clock, nReset);

input Clock, nReset;
output [2:0] Count;

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

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

endmodule
```

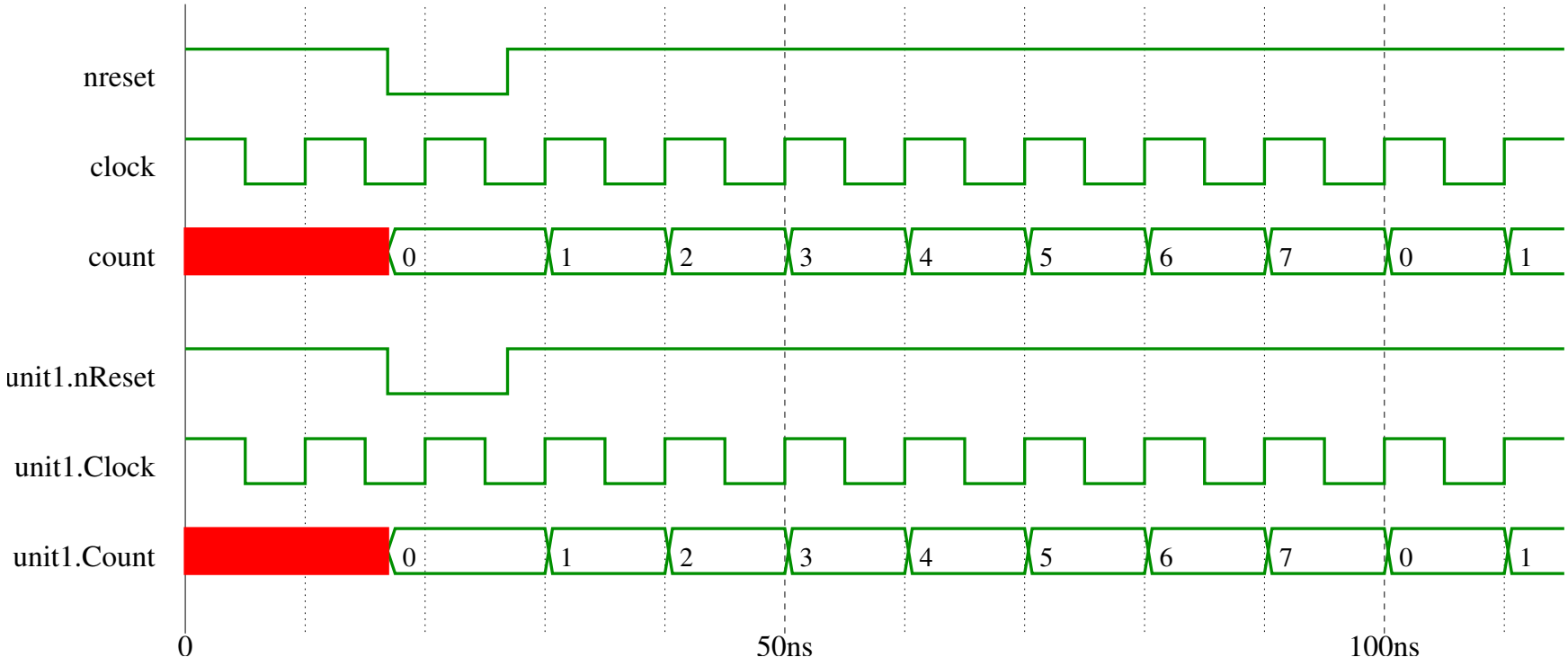
Verilog HDL - hierarchy

Notes

- `hdl8` contains the model of the counter.
- `hdl8` is synthesizable.
- `nReset` and `Clock` are generated elsewhere and so must be declared here as `wire`.
- `Count` is generated here and passed to the parent module module. It is declared here as `reg`ister.

Verilog HDL - hierarchy

Waveforms for hdl8



Verilog HDL - hierarchy

```
`timescale 1ns / 1ns
module hdl9_stim;

reg 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
```

Verilog HDL - hierarchy

```
`timescale 1ns / 1ns
module hdl9(Max, Count, Clock, nReset);

input Clock, nReset;
output Max; output [2:0] Count;

wire Clock, nReset, Max;
reg [2:0] Count;

assign Max = Count[2] && Count[1] && Count[0];

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

endmodule
```

Verilog 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

  Where signals are continuously assigned no register is needed.

- Port connection by position in ordered list

  An alternative to connection by name – should be used with care.

Verilog HDL - hierarchy