UNIVERSITY OF LIVERPOOL

Digital Systems Design ELEC373/473

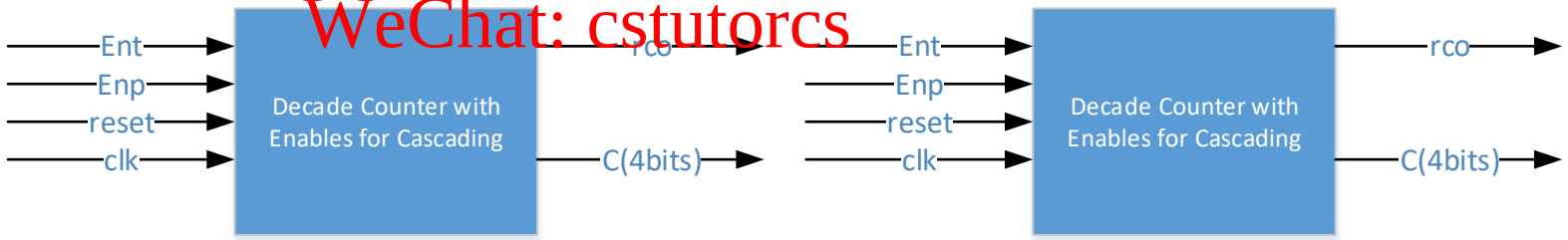
Counters Again

Prof J.S. Smith
Room A515;
E-mail: j.s.smith@liv.ac.uk

Decade Counter

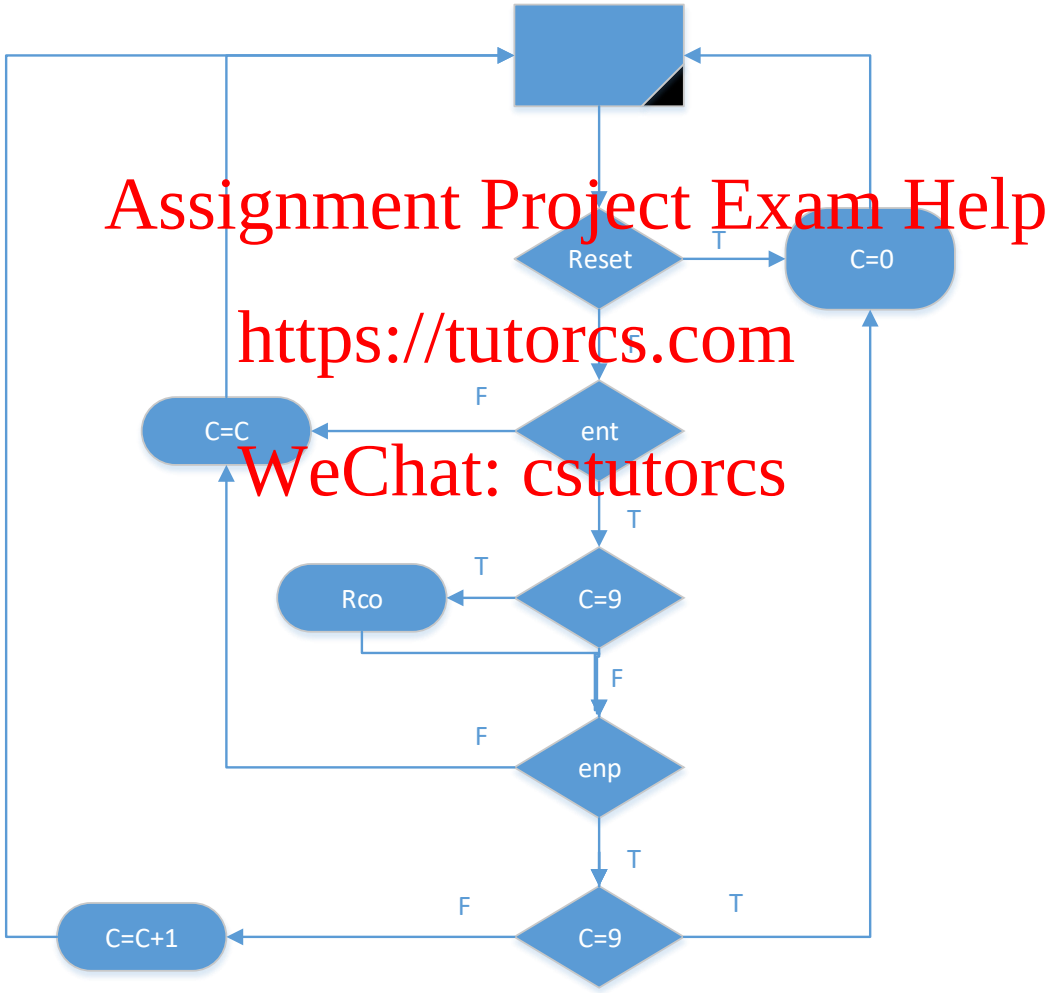
Rco = ripple carry out
True when count = max and ent is valid

Counter increments on clock active edge when ent & enp = True



Cascade counter by connecting rco of lsb to ent of msb
Use common enp to enable counters

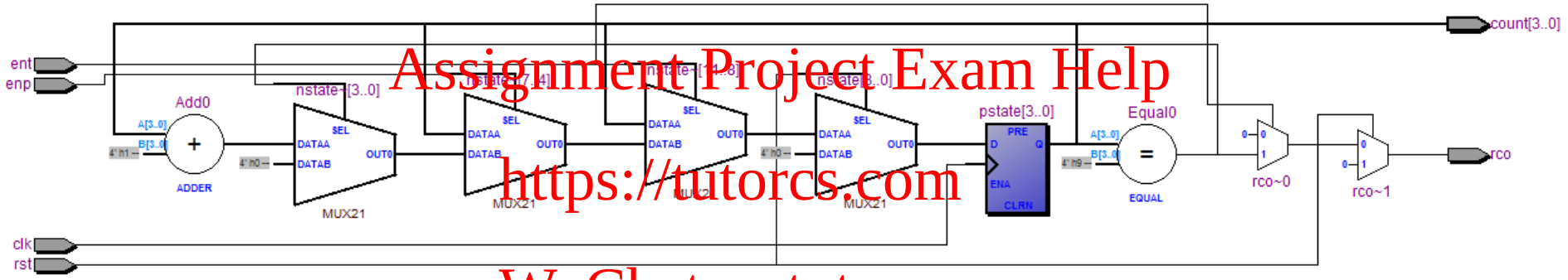
Decade counter ASM



Decade Counter

```
1   module count10 (input clk, input rst, input enp, input ent, output [3:0] count, output
    reg rco);
2       // standard decade counter with rco
3       // and two enables ent & enp.
4       // ent also enables rco
5       reg [3:0] pstate, nstate;  // Present state and Next State
6       assign count = pstate;
7
8       always @(posedge clk)
9       begin
10          if (clk) begin        // technically  "if (clk)" not needed as only clk in list
11              pstate <= nstate; // load next state into present state at next clock edge
12          end
13      end
14
15      always @(pstate, rst, enp, ent) // could be "," or "or"
16      begin
17          rco = 1'b0;                 // default value to stop latch synthesis
18          nstate = pstate;            // default assignment to stop latch synthesis
19          if (rst) begin              // synchronous active high reset
20              nstate = 4'b0;
21          end
22          else
23              begin
24                 if (ent)             // enable rco if count = 9 and ent valid
25                    begin
26                       if (pstate == 4'b1001)
27                          rco = 1'b1;
28                       if (enp)       // count if ent and enp valid
29                          if (pstate == 4'b1001) // if 9 go to 0
30                             nstate = 4'b0000;
31                          else
32                             nstate = pstate + 4'b0001; // otherwise add 1.
33                    end
34              end
35      end
36  endmodule // count10
37
```

Synthesised design of decade counter

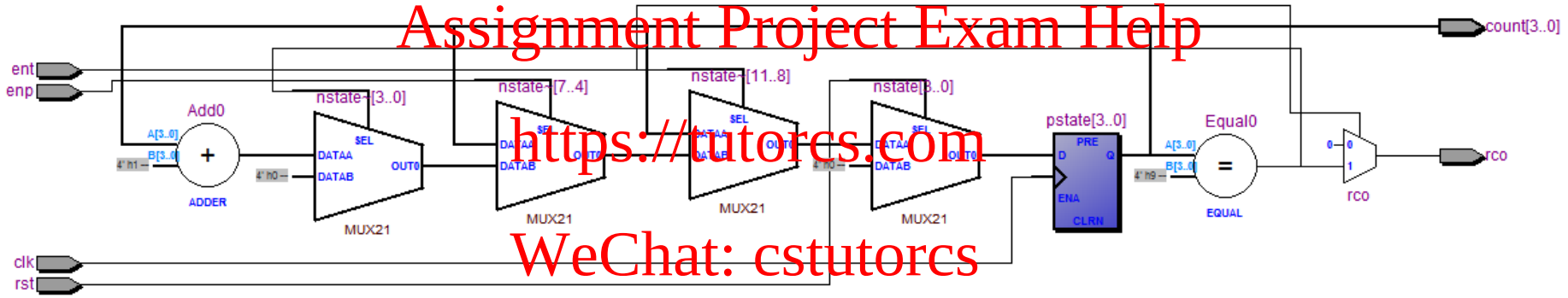


Note that rco is cleared by the reset, is that needed?

Decade Counter - 2

```
1  module count10 (input clk, input rst, input enp, input ent, output [3:0] count, output reg rco);
2  // standard decade counter with 4 bits output
3  // can be cascaded to build larger sequential counters
4  // only counts if ent and enp are valid
5  // produce rco output which should be connected to
6  // ent of the next stage
7
8      reg [3:0] pstate, nstate;
9      assign count = pstate;
10
11     always @(posedge clk)
12     begin
13         if (clk) begin
14             pstate <= nstate;
15         end
16     end
17
18     always @(pstate, rst, enp, ent)
19     begin
20         rco = 1'b0;          //default to prevent latch assigment
21         nstate = pstate;     //default to prevent latch assigment
22         if (rst)
23             begin
24                nstate = 4'b0;
25                rco = 1'bx;
26             end
27         else
28             begin
29                if (ent)
30                   begin
31                      if (pstate == 4'b1001)
32                         rco = 1'b1;
33                      if (enp)
34                         if (pstate == 4'b1001)
35                            nstate = 4'b0000;
36                         else
37                            nstate = pstate + 4'b0001;
38                   end
39             end
40     end
41 endmodule
```

Synthesised decade counter (2)

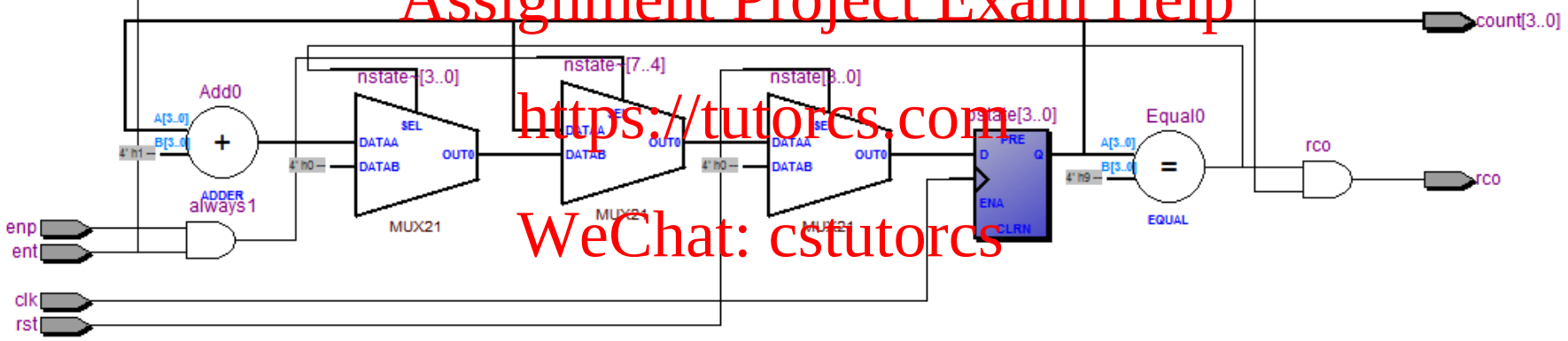


Decade Counter - 3

```
1  module count10 (input clk, input rst, input enp, input ent, output [3:0] count, output rco);
2  // standard decade counter with 4 bits output
3  // can be cascaded to build larger sequential counters
4  // only counts if ent and enp are valid
5  // produce rco output which should be connected to
6  // ent of the next stage
7
8      reg [3:0] pstate, nstate;
9
10     assign count = pstate;
11     assign rco = (pstate==4'b1001) & ent;
12
13     always @(posedge clk)
14     begin
15         if (clk) begin
16             pstate <= nstate;
17         end
18     end
19
20     always @(pstate, rst, enp, ent)
21     begin
22         nstate = pstate;      //default to prevent latches
23         if (rst)
24             begin
25                nstate = 4'b0;
26             end
27         else
28             begin
29                if (ent & enp)
30                   if (pstate == 4'b1001)
31                         nstate = 4'b0000;
32                   else
33                         nstate = pstate + 4'b0001;
34             end
35     end
36 endmodule
```

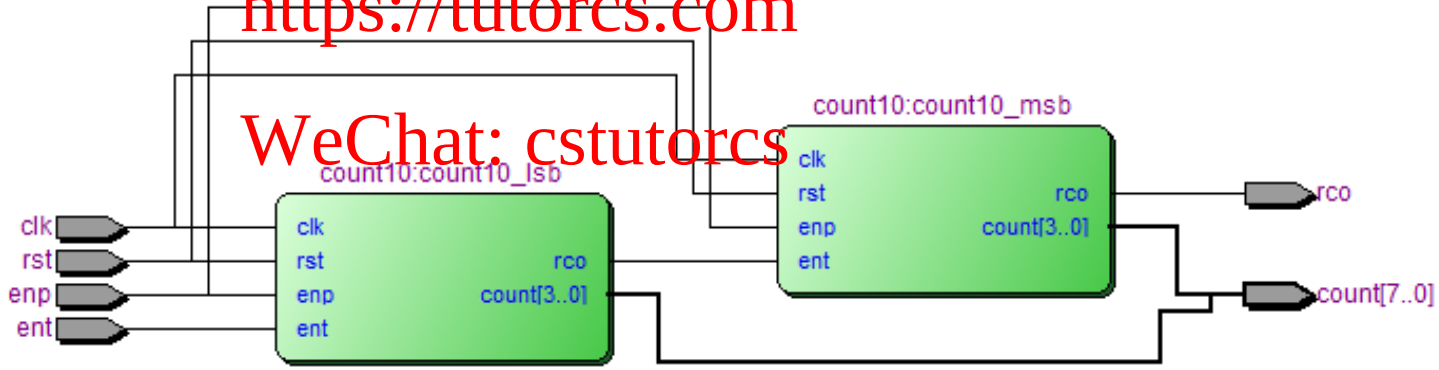

Use assign to generate rco
Note rco now not declared as ‘reg’ as it is always being assigned

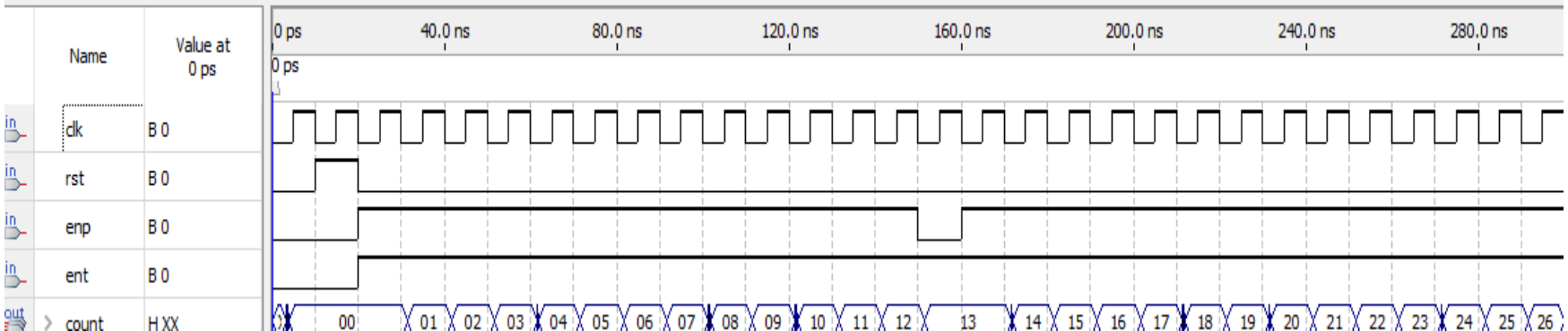
Synthesised decade counter (3)



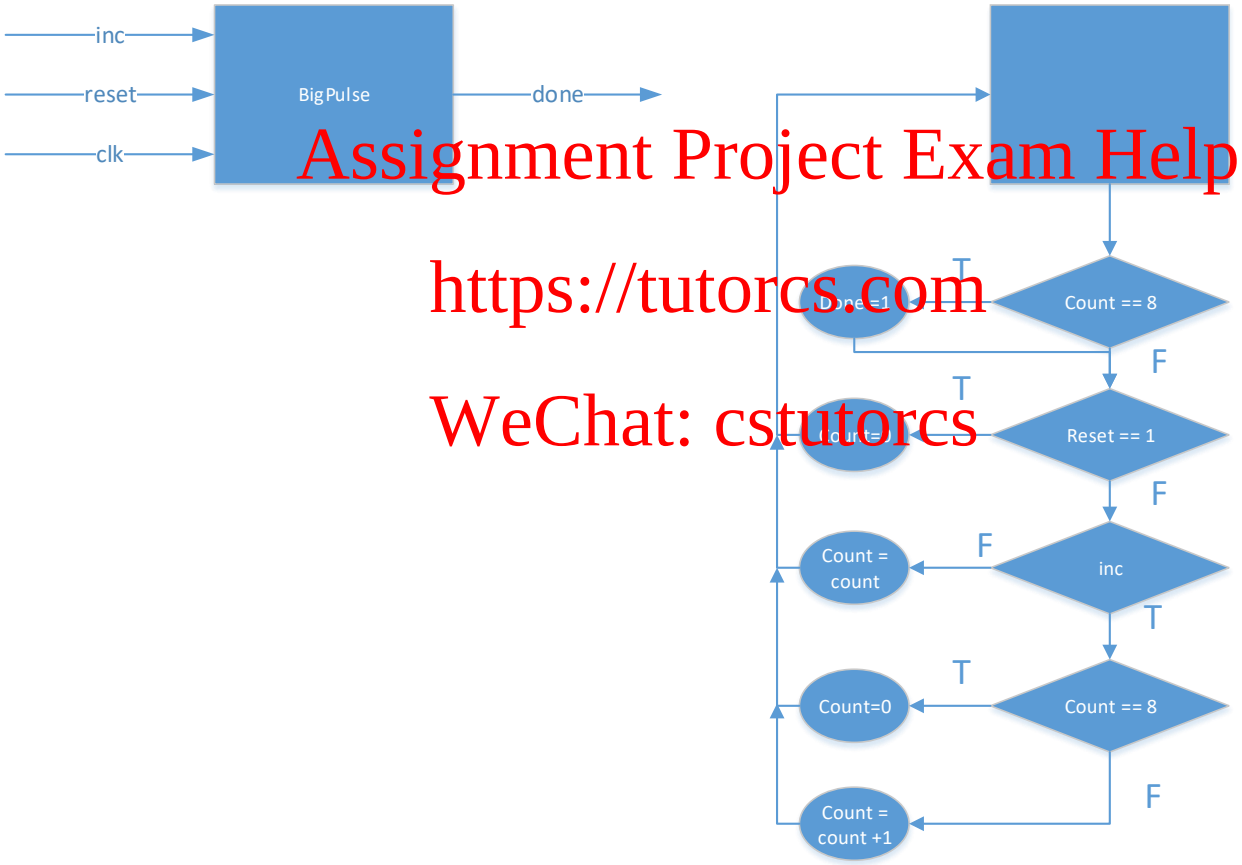
Hundred Counter

```
1  module count100 (input clk, input rst, input enp, input ent, output[7:0] count, output rco);
2
3     wire rco_int;
4
5     count10 count10_lsb(clk, rst, enp, ent, count[3:0], rco_int);
6     count10 count10_msb(clk, rst, enp, rco_int, count[7:4], rco);
7
8  endmodule // count100
9
```



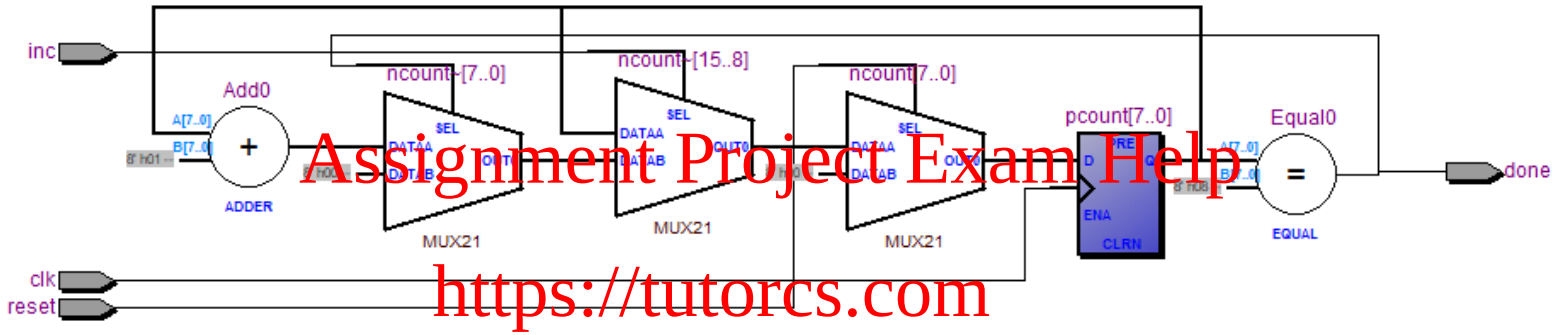
Large Counter (bigpulse)

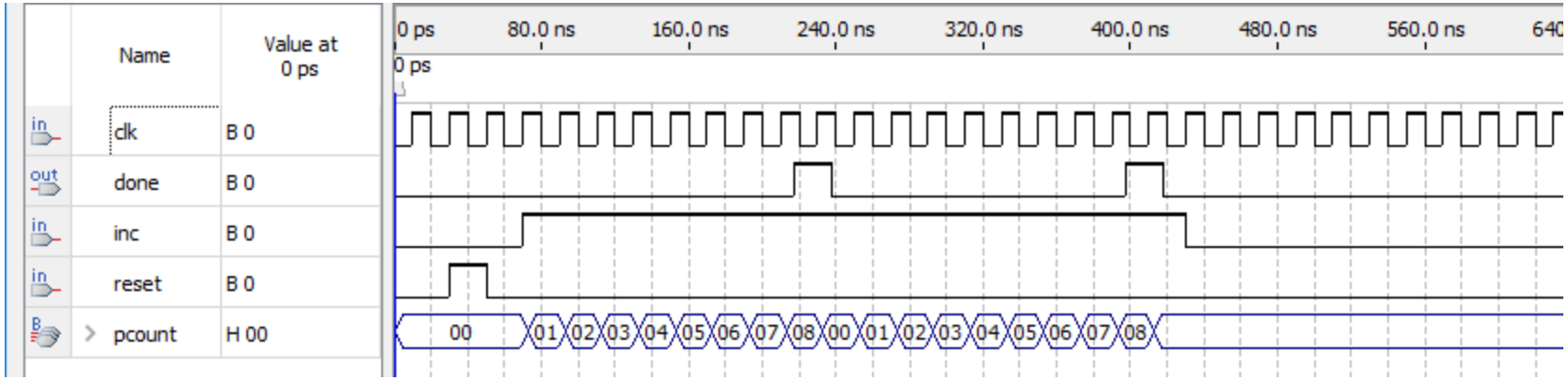


Verilog for bigpulse

```
1  module bigpulse (input clk, input reset, input inc, output reg done);
2  // a counter that counts up to a big number and generates a pulse when the
3  // count is reached and then resetting to 0
4
5  // assign the variables to hold the present count
6  // and the next count (8 bits for this example)
7      reg [7:0] pcount, ncount;
8
9  // the sequential code to synthesise the latches
10    always @(posedge clk)
11       pcount <= ncount; // load next count into present count
12
13 // the combinational code to increment the count if required
14 // for simulation tests just count to 8
15
16     always @(pcount, reset, inc)
17     begin
18          done = 1'b0;          // specify the defaults
19          ncount = pcount;  // stops latch synthesis
20          if (pcount == 8'd8)  // if c = max
21             done = 1'b1;
22         if (reset)             // check for syhncronous reset check
23             begin
24                ncount = 8'd0; // if reset clear next count
25                done = 1'bx;
26             end
27         else
28             if (inc)        // if enabled
29                if (pcount == 8'd8)  // if enabled and c = 8
30                   ncount = 8'd0; // set count to 0
31                else
32                   ncount = pcount + 8'd1; // else add 1
33     end
34 endmodule // bigcount
```

RTL View for bigpulse





Alternate reset implementation

```
1   module bigpulse (input clk, input reset, input inc, output reg done);
2   // a counter that counts up to a big number and generates a pulse when the
3   // count is reached and then resetting to 0
4
5   // assign the variables to hold the present count
6   // and the next count (8 bits for this example)
7       reg [7:0] pcount, ncount;
8
9   // the sequential code to synthesise the latches
10     always @(posedge clk)
11        if (reset)
12           pcount <= 8'd0;
13        else
14           pcount <= ncount; // load next count into present count
15
16  // the combinational code to increment the count if required
17  // for simulation tests just count to 8
18
19      always @(pcount, reset, inc)
20      begin
21           done = 1'b0;         // specify the defaults
22           ncount = pcount;  // stops latch synthesis
23           if (pcount == 8'd8)  // if c = max
24              done = 1'b1;
25           if (inc)         // if enabled
26              if (pcount == 8'd8)  // if enabled and c = 8
27                 ncount = 8'd0; // set count to 0
28              else
29                 ncount = pcount + 8'd1; // else add 1
30      end
31  endmodule // bigcount
32
```

New RTL View for bigpulse