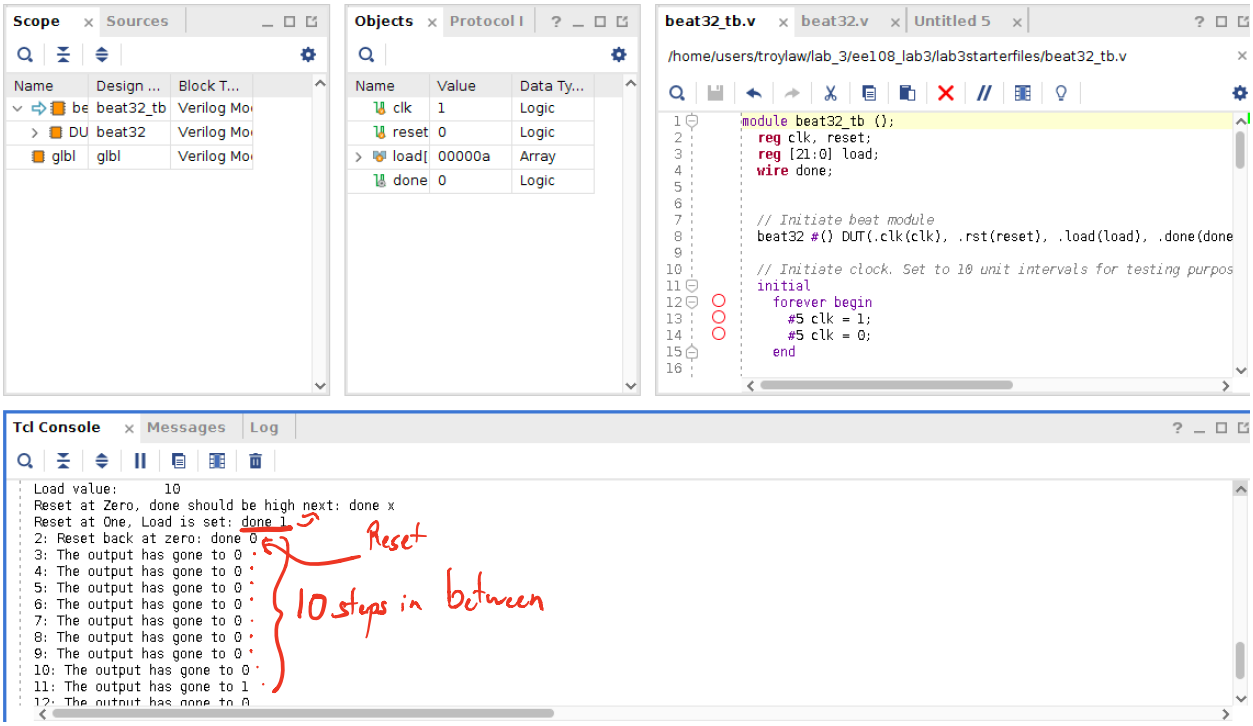
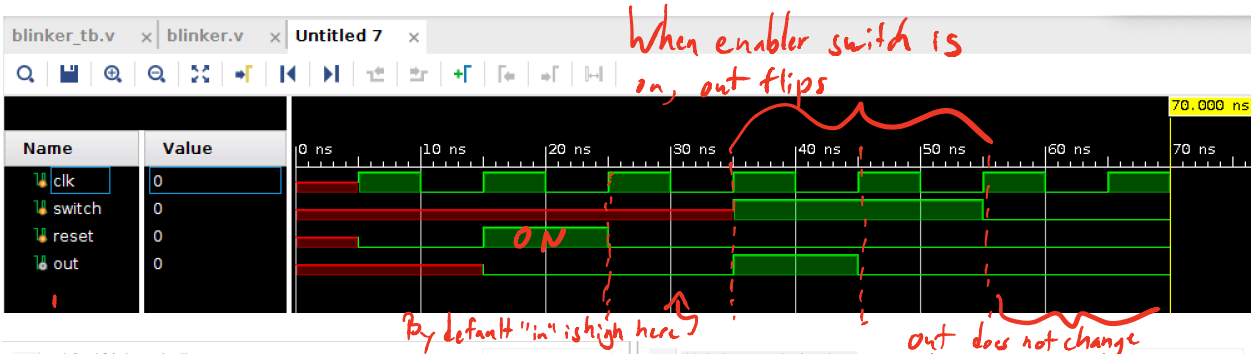
Beat 32



Inputs: load here is initialized at 10 instead of 3,125,000 for the sake of testing.

Expect: The loop to decrement 10 times before outputting high. Should be high immediately after reset as well since done == 1 when the state is all 0s, otherwise it resets

Blinker.v ✓




```
1  module blinker_tb ();
2
3    reg clk, switch, reset;
4    wire [1:0]out;
5
6    blinker #() DUT(.clk(clk), .switch(switch), .reset(reset), .out(out));
7
8      initial
9      forever
10     begin
11         #5 clk = 1 ;
12         #5 clk = 0 ;
13     end
14
15
16 initial begin
17
18   #5 reset = 0; // start without reset (x state)
19   #5 $display("First Reset: out %b", out);
20   #5 reset = 1;
21   #5 $display("Second Reset: out %b", out);
22   #5 reset = 0; // remove reset
23   #5 $display("Third Reset: out %b", out);
24
25   // switch on, input 1
26   #5 switch = 1'b1;
27   //in = 1'b1;
28   #5 $display("Switch is %b, output is %b", switch, out);
29
30   // switch on, input 0
```

SV/Verilog Testbench

```
1  // Code your design here
2  module blinker (
3      input clk, switch, reset, // Clock, he enabler is the switch output from timer.v
4    output wire[1:0] out  // Blinker is on if out is hot.
5  );
6
7    wire in;
8    wire out1;
9
10   // Switch is fed from timer, in should be intialized at one to start the blinker on.
11   dffre #(1) blink(clk, reset, switch, in, out1);
12
13   assign in = (out1 == 1) ? 0 : 1;
14
15   assign out = {in, out1};
16 endmodule
17
18 // dffre: D flip-flop with active high enable and reset
19 // Parametrized width; default of 1
20 module dffre #(parameter WIDTH = 1) (
21     input clk,
22     input r,
23     input en,
24     input [WIDTH-1:0] d,
25     output reg [WIDTH-1:0] q
26 );
27
28     always @(posedge clk)
```

SV/Verilog Design

```
Log   Share
[2020-10-15 00:46:59 EDT] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
First Reset: out xx
Second Reset: out 10
Third Reset: out 10
Switch is 1, output is 01
Switch is 1, output is 10
Switch is 0, output is 10
Switch is 0, output is 10
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 70 ticks.
```

in out

Switch is 1, output is 01

Switch is 1, output is 10

no change

Inputs: First have the module reset to default settings. Then set enabler (switch).

Expect: When enabler is on the output should show to different values, and flop after a cycle. With the enabler off, the module should hold state.

N.B. This output concatenated {in, out} for the purpose of analysis. The actual verilog module only outputs one "out" bit.

Timer.v

testbench.sv

```
// Code your testbench here
// or browse Examples
module timer_tb ();
  reg clk;
  reg rst;
  reg[3:0] load_value;
  reg count_en;
  reg fast;
  wire q;
  timer #() DUT(.clk(clk), .rst(rst), .load_value(load_value),
.count_en(count_en), .fast(fast), .q(q));


    initial
      forever begin
        #5 clk = 1;
```

SV/Verilog Testbench

design.sv

```
module timer (
  input clk,
  input rst,
  input wire [3:0] load_value,
  input wire count_en,
  input wire fast,
  output wire q
);

  reg [8:0] counter;

  always @(*) begin
    if (fast) begin
      if(load_value[0] == 1) begin
        counter = 9'd4;
      end
```

Log | Share

clock cycle : #10

```
1) Load Value: 0001, Fast: 1, count_en: 1, OUT: 0
The output has gone to 0
The output has gone to 0
The output has gone to 0
The output has gone to 1
The output has gone to 0
2) Load Value: 1000, Fast: 1, count_en: 1, OUT: 0
The output has gone to 0
The output has gone to 0
The output has gone to 0
The output has gone to 1
The output has gone to 0
3) Load Value: 0001, Fast: 0, count_en: 1, OUT: 0
The output has gone to 0
The output has gone to 0
The output has gone to 0
The output has gone to 1
The output has gone to 0
4) Load Value: 1000, Fast: 0, count_en: 1, OUT: 0
The output has gone to 0
The output has gone to 0
The output has gone to 0
The output has gone to 1
The output has gone to 0
```

1) 4 displays at #10 each = 40 sec
2) 4 displays at #80 each = 1 sec
3) 4 displays at #80 each = 1 sec
4) 4 displays at #640 each = 8 sec

Input: load_value: shifted number for speed
fast: on faster instantiation or slower
count_en: simulating on from beat32.

Expect: (Details in code comments). We have a full cycle clock of #10. We manipulated the waits before each display for each test to output 1 in four displays. Therefore the waits were:

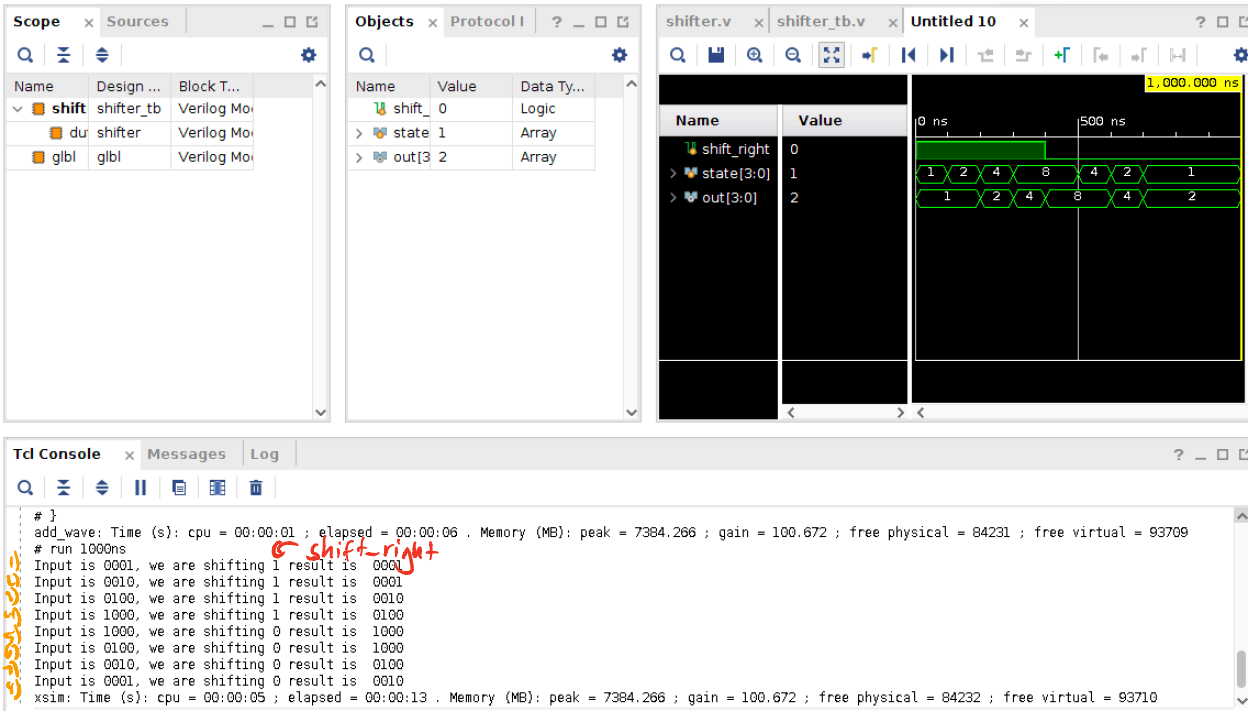
Test1: 32/8 at #10 ← fastest fast

Test2: 32/1 at #10x8 ← slowest fast

Test3: 32/1 at #10x8 ← fastest slow

Test4: 32/(1/8) at #10x8x8 ← slowest slow

Shifter.v



Input: shift number, shift right (on ? shift right : shift left)

Expected: 1) 1 shifted right still = 1 2) 2 shifted right = 1

3) 4 shifted right = 2 4) 8 shifted right = 4

5) 8 shifted left still = 8 6) 4 shifted left = 8

7) 2 shifted left = 4 8) 1 shifted left = 2