

```
1  `timescale 1ns / 1ps
2
3  module gcd_top(
4      input logic clk,
5      input logic [1:0] btn,
6      input logic [3:0] sw,
7      output logic[3:0] led
8  );
9
10 logic rst, load, rst_deb, load_deb;
11
12 // synthesis only
13 //assign rst = rst_deb;
14 //assign load = load_deb;
15
16 // sim only
17 assign rst = btn[0];
18 assign load = btn[1];
19
20 // internal signals
21
22 // fsm signals
23
24 // instantiate debounce
25 debounce debounce_inst (
26     .clk      (clk),
27     .rst_btn   (btn[0]),
28     .load_btn  (btn[1]),
29     .rst_deb   (rst_deb),
30     .load_deb  (load_deb)
31 );
32
33 // instantiate the GCD core
34 gcd_core core0(
35     .clk(clk),
36     .rst(rst),
37     .load(load_core),
38     .din(din),
39     .gcd_rslt(result),
40     .done(done)
41 );
42
43 // wrapper logic
44
45
46 endmodule
47
```

*Don't sim the debounce circuit!
Bypass instead.*

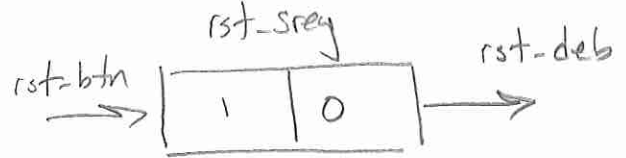
```

1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3
4  module debounce(
5      input logic clk, ← 125MHz
6      input logic rst_btn,
7      input logic load_btn,
8      output logic load_deb, rst_deb
9  );
10
11
12  logic [1:9] lfsr = 0;
13  logic [9:0] load_sreg = 0;
14  logic [1:0] rst_sreg = 0; // 2FF synchronizer
15
16  assign rst_deb = rst_sreg[0];
17
18  // lfsr and reset sync
19  always_ff@(posedge clk)
20  begin
21      // 2FF reset sync
22      rst_sreg <= {rst_btn, rst_sreg[1]};
23
24      // LFSR and load debounce
25      if (rst_deb)
26          begin
27              lfsr <= 0; load_sreg <= 0; load_deb <= 0;
28          end
29      else
30          begin
31              lfsr <= {(lfsr[5] ^ lfsr[9]), lfsr[1:8]};
32              if (lfsr == 0)
33                  begin
34                      load_sreg <= {load_btn, load_sreg[9:1]};
35                      load_deb <= (load_sreg[8:0] == 9'b100000000);
36                  end
37              else load_deb <= 0;
38          end // else
39  end // always
40
41  endmodule
42

```

(Only 1 clock domain - no "derived" clocks!)

2-bit shift reg



load_sreg collects 10 samples of load_btn, but at a reduced frequency due to LFSR, i.e. spread out over longer time span.

