

```

1  ==> async_up.v <==
2  module async_up (
3      input clk,
4      input rst,
5      output [1:0] q
6  );
7
8      dff U0 (
9          .rst(rst),
10         .clk(clk),
11         .d (~q[0]),
12         .q (q[0])
13     );
14
15     dff U1 (
16         .rst(rst),
17         .clk(~q[0]),
18         .d (~q[1]),
19         .q (q[1])
20     );
21 endmodule
22
23 ==> dff.v <==
24 module dff #(
25     parameter bool INIT = 1'b0
26 ) (
27     input clk,
28     input rst,
29     input d,
30     output reg q
31 );
32     always @(posedge clk) begin
33         if (rst) begin
34             q = INIT;
35         end else begin
36             q = d;
37         end
38     end
39 endmodule
40
41 ==> jkff.v <==
42 module jkff (
43     input clk,
44     input rst,
45     input j,
46     input k,
47     output reg q
48 );
49     always @(posedge clk) begin
50         if (rst) begin
51             q = 1'b0;
52         end else
53             case ({
54                 j, k
55             })
56                 2'b00: q = q;
57                 2'b01: q = 0;
58                 2'b10: q = 1;
59                 2'b11: q = ~q;
60             endcase
61     end
62 endmodule
63
64 ==> johnson.v <==
65 module johnson (
66     input rst,
67     input clk,
68     output [2:0] q
69 );

```

```

69     ,
70
71     dff U0 (
72         .rst(rst),
73         .clk(clk),
74         .d (~q[2]),
75         .q (q[0])
76     );
77
78     dff U1 (
79         .rst(rst),
80         .clk(clk),
81         .d (q[0]),
82         .q (q[1])
83     );
84
85     dff U2 (
86         .rst(rst),
87         .clk(clk),
88         .d (q[1]),
89         .q (q[2])
90     );
91
92     endmodule
93
94     ==> ring.v <==
95     module ring (
96         input rst,
97         input clk,
98         output [2:0] q
99     );
100
101     dff #(
102         .INIT(1'b1)
103     ) U0 (
104         .rst(rst),
105         .clk(clk),
106         .d (q[2]),
107         .q (q[0])
108     );
109
110     dff #(
111         .INIT(1'b0)
112     ) U1 (
113         .rst(rst),
114         .clk(clk),
115         .d (q[0]),
116         .q (q[1])
117     );
118
119     dff #(
120         .INIT(1'b0)
121     ) U2 (
122         .rst(rst),
123         .clk(clk),
124         .d (q[1]),
125         .q (q[2])
126     );
127
128     endmodule
129
130     ==> sync_down.v <==
131     module sync_down (
132         input clk,
133         input rst,
134         output [1:0] q
135     );
136     jkff U1 (
137         .clk(clk),
138         .rst(rst),

```

```

139     .j  (~q[0]),
140     .k  (~q[0]),
141     .q  (q[1])
142 );
143
144 jkff U0 (
145     .clk(clk),
146     .rst(rst),
147     .j  (1'b1),
148     .k  (1'b1),
149     .q  (q[0])
150 );
151 endmodule
152
153 ==> sync_updown.v <==
154 module sync_updown (
155     input clk,
156     input rst,
157     input M,
158     output [1:0] q
159 );
160     jkff U1 (
161         .clk(clk),
162         .rst(rst),
163         .j  ((~M & ~q[0]) | (M & q[0])),
164         .k  ((~M & ~q[0]) | (M & q[0])),
165         .q  (q[1])
166     );
167
168     jkff U0 (
169         .clk(clk),
170         .rst(rst),
171         .j  (1'b1),
172         .k  (1'b1),
173         .q  (q[0])
174     );
175 endmodule
176
177 ==> sync_up.v <==
178 module sync_up (
179     input clk,
180     input rst,
181     output [1:0] q
182 );
183     jkff U1 (
184         .clk(clk),
185         .rst(rst),
186         .j  (q[0]),
187         .k  (q[0]),
188         .q  (q[1])
189     );
190
191     jkff U0 (
192         .clk(clk),
193         .rst(rst),
194         .j  (1'b1),
195         .k  (1'b1),
196         .q  (q[0])
197     );
198 endmodule
199
200 ==> tb_ripple.v <==
201 // tb_ripple_up_counter_2bit.v
202 `timescale 1ns / 1ps
203 module tb_ripple_up_counter_2bit;
204     reg clk = 0, rst = 1;
205     wire [1:0] q;
206
207     async_up dut (
208         .clk(clk),

```

```

209         .rst(rst),
210         .q (q)
211     );
212
213     // 10 ns clock
214     always #5 clk = ~clk;
215
216     // Print one line per *count* (wait a tiny time so ripple completes)
217     always @(posedge clk) begin
218         #1 $display("%0t  clk=%0b rst=%0b Q=%b%b", $time, clk, rst, q[1], q[0]);
219     end
220
221     initial begin
222         $dumpfile("ripple_up_2bit.vcd");
223         $dumpvars(0, tb_ripple_up_counter_2bit);
224
225         // Hold reset for two edges, then release
226         repeat (2) @(posedge clk);
227         rst <= 0;
228
229         repeat (12) @(posedge clk);
230         $finish;
231     end
232 endmodule
233
234 ==> tb_sync_down.v <==
235 // Testbench
236 module tb_down_counter_2bit;
237     reg clk, reset;
238     wire [1:0] q;
239
240     // Instantiate counter
241     sync_down uut (
242         .clk(clk),
243         .rst(reset),
244         .q (q)
245     );
246
247     // Clock generation
248     initial begin
249         clk = 0;
250         forever #5 clk = ~clk; // Clock period = 10
251     end
252
253     // Stimulus
254     initial begin
255         $dumpfile("down_counter_jk.vcd");
256         $dumpvars(0, tb_down_counter_2bit);
257
258         $display("Time\ttClk\ttReset\ttQ1Q0");
259         $monitor("%0t\t%b\t%b\t%b%b", $time, clk, reset, q[1], q[0]);
260
261         reset = 1;
262         #10; // Apply reset
263         reset = 0;
264
265         #80; // Run counter for a while
266         reset = 1;
267         #10; // Reset again
268
269         reset = 0;
270
271         #40;
272         $finish;
273     end
274 endmodule
275
276 ==> tb_sync_updown.v <==
277 `timescale 1ns / 1ps
278 module tb_updown_counter_2bit;
279     reg clk, reset;
280     wire [1:0] q;

```

```

278 reg clk = 0;
279 reg rst = 1;
280 reg up_down = 1;
281 wire [1:0] q;
282
283 // DUT (your module from before)
284 sync_updown dut (
285     .clk(clk),
286     .rst(rst),
287     .M (up_down),
288     .q (q)
289 );
290
291 // 10 ns period clock
292 always #5 clk = ~clk;
293
294 // Print ONLY at posedge so we see state after each synchronous update
295 always @(posedge clk) begin
296     $display("%0t  clk=%0b rst=%0b upDn=%0b  Q=%0b%0b", $time, clk, rst, up_down, q[1], q[0]);
297 end
298
299 initial begin
300     $dumpfile("updown_2bit_jk.vcd");
301     $dumpvars(0, tb_updown_counter_2bit);
302
303     // Keep reset high for one edge -> Q becomes 00 at first posedge
304     @(negedge clk);
305     rst = 1;
306     @(negedge clk);
307     rst = 0; // release reset just before a posedge
308
309     // Count UP to reach 11: (00)->01->10->11 (3 transitions)
310     // Change 'up_down' only on negedge so the next posedge samples it cleanly
311     up_down = 1;
312     repeat (3) @(negedge clk); // settle to 11
313
314     // Now switch to DOWN **before** the next posedge so next state is 10
315     up_down = 0;
316     repeat (3) @(negedge clk); // 11->10->01->00
317
318     $finish;
319 end
320 endmodule
321
322 ==> tb_sync_up.v <==
323 // Testbench
324 module tb_up_counter_2bit;
325     reg clk, reset;
326     wire [1:0] q;
327
328     // Instantiate counter
329     sync_up uut (
330         .clk(clk),
331         .rst(reset),
332         .q (q)
333     );
334
335     // Clock generation
336     initial begin
337         clk = 0;
338         forever #5 clk = ~clk; // Clock period = 10
339     end
340
341     // Stimulus
342     initial begin
343         $dumpfile("up_counter_jk.vcd");
344         $dumpvars(0, tb_up_counter_2bit);
345
346         // Print header
347         $display("Time\tClk\tReset\tQ1Q0");

```

```

348     $monitor("%0t\t%b\t%b\t%b%", $time, clk, reset, q[1], q[0]);
349
350     reset = 1;
351     #10; // Apply reset
352     reset = 0;
353
354     #60; // Run counter for a while
355     reset = 1;
356     #10; // Reset again
357     reset = 0;
358
359     #40;
360     $finish;
361 end
362 endmodule
363 ==> async_down.output <==
364
365 ==> async_up.output <==
366 VCD info: dumpfile ripple_up_2bit.vcd opened for output.
367 6000   clk=1 rst=1  Q=00
368 16000  clk=1 rst=0  Q=00
369 26000  clk=1 rst=0  Q=01
370 36000  clk=1 rst=0  Q=10
371 46000  clk=1 rst=0  Q=11
372 56000  clk=1 rst=0  Q=00
373 66000  clk=1 rst=0  Q=01
374 76000  clk=1 rst=0  Q=10
375 86000  clk=1 rst=0  Q=11
376 96000  clk=1 rst=0  Q=00
377 106000 clk=1 rst=0  Q=01
378 116000 clk=1 rst=0  Q=10
379 126000 clk=1 rst=0  Q=11
380 tb_ripple.v:30: $finish called at 135000 (1ps)
381
382 ==> sync_down.output <==
383 VCD info: dumpfile down_counter_jk.vcd opened for output.
384 Time Clk Reset Q1Q0
385 0 0 1 xx
386 5 1 1 00
387 10 0 0 00
388 15 1 0 11
389 20 0 0 11
390 25 1 0 10
391 30 0 0 10
392 35 1 0 01
393 40 0 0 01
394 45 1 0 00
395 50 0 0 00
396 55 1 0 11
397 60 0 0 11
398 65 1 0 10
399 70 0 0 10
400 75 1 0 01
401 80 0 0 01
402 85 1 0 00
403 90 0 1 00
404 95 1 1 00
405 100 0 0 00
406 105 1 0 11
407 110 0 0 11
408 115 1 0 10
409 120 0 0 10
410 125 1 0 01
411 130 0 0 01
412 135 1 0 00
413 tb_sync_down.v:37: $finish called at 140 (1s)
414 140 0 0 00
415
416 ==> sync_updown.output <==
417 VCD info: dumpfile updown_2bit.vcd opened for output

```

```
417 VCD info: dumptfile updown_zbit_jk.vcd opened for output.
418 5000   clk=1 rst=1 upDn=1   Q=xx
419 15000  clk=1 rst=1 upDn=1   Q=00
420 25000  clk=1 rst=0 upDn=1   Q=00
421 35000  clk=1 rst=0 upDn=1   Q=10
422 45000  clk=1 rst=0 upDn=1   Q=10
423 55000  clk=1 rst=0 upDn=0   Q=10
424 65000  clk=1 rst=0 upDn=0   Q=10
425 75000  clk=1 rst=0 upDn=0   Q=00
426 tb_sync_updown.v:43: $finish called at 80000 (1ps)
427
428 ==> sync_up.output <==
429 VCD info: dumptfile up_counter_jk.vcd opened for output.
430 Time Clk Reset Q1Q0
431 0 0 1 xx
432 5 1 1 00
433 10 0 0 00
434 15 1 0 01
435 20 0 0 01
436 25 1 0 10
437 30 0 0 10
438 35 1 0 11
439 40 0 0 11
440 45 1 0 00
441 50 0 0 00
442 55 1 0 01
443 60 0 0 01
444 65 1 0 10
445 70 0 1 10
446 75 1 1 00
447 80 0 0 00
448 85 1 0 01
449 90 0 0 01
450 95 1 0 10
451 100 0 0 10
452 105 1 0 11
453 110 0 0 11
454 115 1 0 00
455 tb_sync_up.v:38: $finish called at 120 (1s)
456 120 0 0 00
```