```
==> async_up.v <==
module async_up (
    input clk,
    input rst,
    output [1:0] q
);

  dff U0 (
      .rst(rst),
      .clk(clk),
      .d  (~q[0]),
      .q  (q[0])
  );

  dff U1 (
      .rst(rst),
      .clk(~q[0]),
      .d  (~q[1]),
      .q  (q[1])
  );
endmodule

==> dff.v <==
module dff #(
    parameter bool INIT = 1'b0
) (
    input clk,
    input rst,
    input d,
    output reg q
);
  always @(posedge clk) begin
    if (rst) begin
      q = INIT;
    end else begin
      q = d;
    end
  end
endmodule

==> jkff.v <==
module jkff (
    input clk,
    input rst,
    input j,
    input k,
    output reg q
);
  always @(posedge clk) begin
    if (rst) begin
      q = 1'b0;
    end else
      case ({
        j, k
      })
        2'b00: q = q;
        2'b01: q = 0;
        2'b10: q = 1;
        2'b11: q = ~q;
      endcase
  end
endmodule

==> johnson.v <==
module johnson (
    input rst,
    input clk,
    output [2:0] q
);
```

```
);

  dff U0 (
      .rst(rst),
      .clk(clk),
      .d  (~q[2]),
      .q  (q[0])
  );

  dff U1 (
      .rst(rst),
      .clk(clk),
      .d  (q[0]),
      .q  (q[1])
  );

  dff U2 (
      .rst(rst),
      .clk(clk),
      .d  (q[1]),
      .q  (q[2])
  );

endmodule

==> ring.v <==
module ring (
    input rst,
    input clk,
    output [2:0] q
);

  dff #(
      .INIT(1'b1)
  ) U0 (
      .rst(rst),
      .clk(clk),
      .d  (q[2]),
      .q  (q[0])
  );

  dff #(
      .INIT(1'b0)
  ) U1 (
      .rst(rst),
      .clk(clk),
      .d  (q[0]),
      .q  (q[1])
  );

  dff #(
      .INIT(1'b0)
  ) U2 (
      .rst(rst),
      .clk(clk),
      .d  (q[1]),
      .q  (q[2])
  );

endmodule

==> sync_down.v <==
module sync_down (
    input clk,
    input rst,
    output [1:0] q
);
  jkff U1 (
      .clk(clk),
      .rst(rst),
      .j  (~q[0]),
      .k  (~q[0]),
      .q  (q[1])
  );

  jkff U0 (
      .clk(clk),
      .rst(rst),
      .j  (1'b1),
      .k  (1'b1),
      .q  (q[0])
  );
endmodule

==> sync_updown.v <==
module sync_updown (
    input clk,
    input rst,
    input M,
    output [1:0] q
);
  jkff U1 (
      .clk(clk),
      .rst(rst),
      .j  ((~M & ~q[0]) | (M & q[0])),
      .k  ((~M & ~q[0]) | (M & q[0])),
      .q  (q[1])
  );

  jkff U0 (
      .clk(clk),
      .rst(rst),
      .j  (1'b1),
      .k  (1'b1),
      .q  (q[0])
  );
endmodule

==> sync_up.v <==
module sync_up (
    input clk,
    input rst,
    output [1:0] q
);
  jkff U1 (
      .clk(clk),
      .rst(rst),
      .j  (q[0]),
      .k  (q[0]),
      .q  (q[1])
  );

  jkff U0 (
      .clk(clk),
      .rst(rst),
      .j  (1'b1),
      .k  (1'b1),
      .q  (q[0])
  );
endmodule

==> tb_ripple.v <==
// tb_ripple_up_counter_2bit.v
`timescale 1ns / 1ps
module tb_ripple_up_counter_2bit;
  reg clk = 0, rst = 1;
  wire [1:0] q;

  async_up dut (
      .clk(clk),
      .rst(rst),
      .q   (q)
  );

  // 10 ns clock
  always #5 clk = ~clk;

  // Print one line per *count* (wait a tiny time so ripple completes)
  always @(posedge clk) begin
    #1 $display("%0t  clk=%0b rst=%0b  Q=%b%b", $time, clk, rst, q[1], q[0]);
  end

  initial begin
    $dumpfile("ripple_up_2bit.vcd");
    $dumpvars(0, tb_ripple_up_counter_2bit);

    // Hold reset for two edges, then release
    repeat (2) @(posedge clk);
    rst <= 0;

    repeat (12) @(posedge clk);
    $finish;
  end
endmodule

==> tb_sync_down.v <==
// Testbench
module tb_down_counter_2bit;
  reg clk, reset;
  wire [1:0] q;

  // Instantiate counter
  sync_down uut (
      .clk(clk),
      .rst(reset),
      .q   (q)
  );

  // Clock generation
  initial begin
    clk = 0;
    forever #5 clk = ~clk;  // Clock period = 10
  end

  // Stimulus
  initial begin
    $dumpfile("down_counter_jk.vcd");
    $dumpvars(0, tb_down_counter_2bit);

    $display("Time\tClk\tReset\tQ1Q0");
    $monitor("%0t\t%b\t%b\t%b%b", $time, clk, reset, q[1], q[0]);

    reset = 1;
    #10;  // Apply reset
    reset = 0;

    #80;  // Run counter for a while
    reset = 1;
    #10;  // Reset again
    reset = 0;

    #40;
    $finish;
  end
endmodule

==> tb_sync_updown.v <==
`timescale 1ns / 1ps
module tb_updown_counter_2bit;
```

```
  reg clk = 0;
  reg rst = 1;
  reg up_down = 1;
  wire [1:0] q;

  // DUT (your module from before)
  sync_updown dut (
      .clk(clk),
      .rst(rst),
      .M  (up_down),
      .q  (q)
  );

  // 10 ns period clock
  always #5 clk = ~clk;

  // Print ONLY at posedge so we see state after each synchronous update
  always @(posedge clk) begin
    $display("%0t  clk=%0b rst=%0b upDn=%0b  Q=%0b%0b", $time, clk, rst, up_down, q[1], q[0]);
  end

  initial begin
    $dumpfile("updown_2bit_jk.vcd");
    $dumpvars(0, tb_updown_counter_2bit);

    // Keep reset high for one edge -> Q becomes 00 at first posedge
    @(negedge clk);
    rst = 1;
    @(negedge clk);
    rst = 0;     // release reset just before a posedge

    // Count UP to reach 11: (00)->01->10->11  (3 transitions)
    // Change 'up_down' only on negedge so the next posedge samples it cleanly
    up_down = 1;
    repeat (3) @(negedge clk);  // settle to 11

    // Now switch to DOWN **before** the next posedge so next state is 10
    up_down = 0;
    repeat (3) @(negedge clk);  // 11->10->01->00

    $finish;
  end
endmodule

==> tb_sync_up.v <==
// Testbench
module tb_up_counter_2bit;
  reg clk, reset;
  wire [1:0] q;

  // Instantiate counter
  sync_up uut (
      .clk(clk),
      .rst(reset),
      .q  (q)
  );

  // Clock generation
  initial begin
    clk = 0;
    forever #5 clk = ~clk;  // Clock period = 10
  end

  // Stimulus
  initial begin
    $dumpfile("up_counter_jk.vcd");
    $dumpvars(0, tb_up_counter_2bit);

    // Print header
    $display("Time\tClk\tReset\tQ1Q0");
    $monitor("%0t\t%b\t%b\t%b%b", $time, clk, reset, q[1], q[0]);

    reset = 1;
    #10;  // Apply reset
    reset = 0;

    #60;  // Run counter for a while
    reset = 1;
    #10;  // Reset again
    reset = 0;

    #40;
    $finish;
  end
endmodule
==> async_down.output <==

==> async_up.output <==
VCD info: dumpfile ripple_up_2bit.vcd opened for output.
6000  clk=1 rst=1  Q=00
16000  clk=1 rst=0  Q=00
26000  clk=1 rst=0  Q=01
36000  clk=1 rst=0  Q=10
46000  clk=1 rst=0  Q=11
56000  clk=1 rst=0  Q=00
66000  clk=1 rst=0  Q=01
76000  clk=1 rst=0  Q=10
86000  clk=1 rst=0  Q=11
96000  clk=1 rst=0  Q=00
106000  clk=1 rst=0  Q=01
116000  clk=1 rst=0  Q=10
126000  clk=1 rst=0  Q=11
tb_ripple.v:30: $finish called at 135000 (1ps)

==> sync_down.output <==
VCD info: dumpfile down_counter_jk.vcd opened for output.
Time Clk Reset Q1Q0
0 0 1 xx
5 1 1 00
10 0 0 00
15 1 0 11
20 0 0 11
25 1 0 10
30 0 0 10
35 1 0 01
40 0 0 01
45 1 0 00
50 0 0 00
55 1 0 11
60 0 0 11
65 1 0 10
70 0 0 10
75 1 0 01
80 0 0 01
85 1 0 00
90 0 1 00
95 1 1 00
100 0 0 00
105 1 0 11
110 0 0 11
115 1 0 10
120 0 0 10
125 1 0 01
130 0 0 01
135 1 0 00
tb_sync_down.v:37: $finish called at 140 (1s)
140 0 0 00

==> sync_updown.output <==
VCD info: dumpfile updown_2bit_jk.vcd opened for output.
5000  clk=1 rst=1 upDn=1  Q=xx
15000  clk=1 rst=1 upDn=1  Q=00
25000  clk=1 rst=0 upDn=1  Q=00
35000  clk=1 rst=0 upDn=1  Q=10
45000  clk=1 rst=0 upDn=1  Q=10
55000  clk=1 rst=0 upDn=0  Q=10
65000  clk=1 rst=0 upDn=0  Q=10
75000  clk=1 rst=0 upDn=0  Q=00
tb_sync_updown.v:43: $finish called at 80000 (1ps)

==> sync_up.output <==
VCD info: dumpfile up_counter_jk.vcd opened for output.
Time Clk Reset Q1Q0
0 0 1 xx
5 1 1 00
10 0 0 00
15 1 0 01
20 0 0 01
25 1 0 10
30 0 0 10
35 1 0 11
40 0 0 11
45 1 0 00
50 0 0 00
55 1 0 01
60 0 0 01
65 1 0 10
70 0 1 10
75 1 1 00
80 0 0 00
85 1 0 01
90 0 0 01
95 1 0 10
100 0 0 10
105 1 0 11
110 0 0 11
115 1 0 00
tb_sync_up.v:38: $finish called at 120 (1s)
120 0 0 00
```