ripple.sv
4-bit Ripple Carry Counter Example

```
// ripple.sv - 4-bit Ripple Carry counter example
//
// Created By:       Roy Kravitz
// Last Modified:    01-April-2020
//
// Revision History:
// -----------------
// 26-Jun-2015   RK      Created this module as an overview to Verilog
// 01-April-2020 RK      Converted to SystemVerilog
//
//
// Description:
// ------------
// Stimulus block and implementation block for a 4-bit Ripple Carry counter.
// Original Source:  Samir Palnitkar, Verilog HDL 2nd Edition, Chapter 2
////////////////////////////////////////////////////////////////////////////
```

```
// D flip-flop with asynchronous reset
// RTL model
module DFF(
    output logic q,
    input logic d, clk, reset
);

    always_ff @(negedge clk or posedge reset) begin
        if (reset)
            q <= 1'b0;
        else
            q <= d;
    end
endmodule : DFF
```

```
// Toggle Flip-flop
module TFF (
    output logic q,
    input logic clk, reset
);
    logic d;

    // instantiate a D flip-flop
    DFF dff0(.q, .d, .clk, .reset);

    // create the toggle flip-flop by driving the input with the inverted output
    not n1(d, q); // not is a Verilog provided primitive.
endmodule : TFF
```

```
// Top level module for 4-bit Ripple Carry Counter
module ripple_carry_counter (
    output [3:0] count,
    input clk, reset,
);
    // internal signals
    logic q0, q1, q2, q3;

    //instantiate 4 toggle flip-flops
    TFF tff0(q0,clk, reset);
    TFF tff1(q1,q0, reset);
    TFF tff2(q2,q1, reset);
    TFF tff3(q3,q2, reset);

    // assign the output
    assign count = {q3, q2, q1, q0};
endmodule : ripple_carry_counter
```

```
// Stimulus module to test the ripple carry counter
// Toggle reset and watch it count
// NOTE: There are no external ports - typical for test benches
module stimulus;
    logic clk;
    logic reset;
    logic[3:0] q;

    // instantiate the design under test (DUT)
    ripple_carry_counter DUT(.*, .count(q));

    // Create the clk signal that drives the design block.
    initial begin
        clk = 1'b0;
    end // initial block

    always begin
        #5 clk = ~clk;
    end // always block

    // Monitor the outputs
    initial begin
        $monitor($time, " Output q = %d",  q);
    end

    // Control the reset signal that drives the design block
```

```
    initial begin : test_vectors
        #0  reset = 1'b1;
        #15 reset = 1'b0;
        #180 reset = 1'b1;
        #10 reset = 1'b0;
        #20 $stop;
    end : test_vectors

endmodule
```