Verilog by Example, errata

The body of the always-block has now become more complicated as we introduce if/else conditional statements to accommodate the reset. Any time "reset" ~~has~~ is high, "out_1" is forced to zero. Since this happens as soon as reset goes active (reset is part of the sensitivity list), and at every rising clock edge, you can see that this effects an asynchronous clear. When reset is

page 20

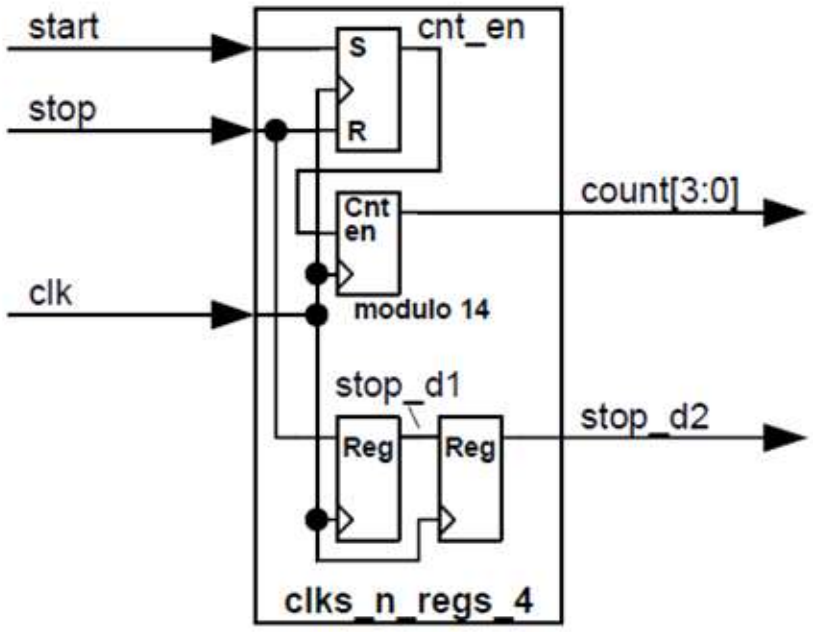
Verilog by Example, errata

Verilog by Example

We now introduce a few common state-type operations to show how increasingly sophisticated register-based functions are implemented in always-blocks. A four-bit counter is enabled by a "start" event, and stopped by a "stop" event. The SR flop allows the start and stop events to be short, e.g. one-clock pulses, rather than a continuously enabling flag. Additionally, for further illustration, we delay the start signal two clocks and send it out.

stop

You'll notice that we have not shown the asynchronous reset. This is done for clarity; from this point forward it is assumed. It is implemented in the code, and always will be (in this book).



SR flop and counter

```
////////////////////////////////////////////
// SR flop and counter
////////////////////////////////////////////

module srflop_n_cntr (  clk,
                        reset,
                        start,
                        stop,
                        count
                      );

    input           clk;
    input           reset;
    input           start;
    input           stop;
    output [3:0]    count;
    output          stop_d2;
    reg             cnt_en;
    reg [3:0]       count;
    reg             stop_d1;
    reg             stop_d2;

    // ------ Design implementation -----

    // SR flop
    always @( posedge clk or posedge reset )
      begin
        if ( reset )
          cnt_en <= 1'b0;
        else if ( start )
          cnt_en <= 1'b1;
        else if ( stop )
          cnt_en <= 1'b0;
      end
```

Verilog by Example, errata

```
////////////////////////////////////////////////
// Clock Buffer

module clock_buffer
          ( reset,
            clk_in,
            dat_in,
            dat_out
          );

    input          reset;
    input          clk_in;
    input          dat_in;
    output         dat_out;

    wire           clk;
    reg            dat_out;
```

60

page 60