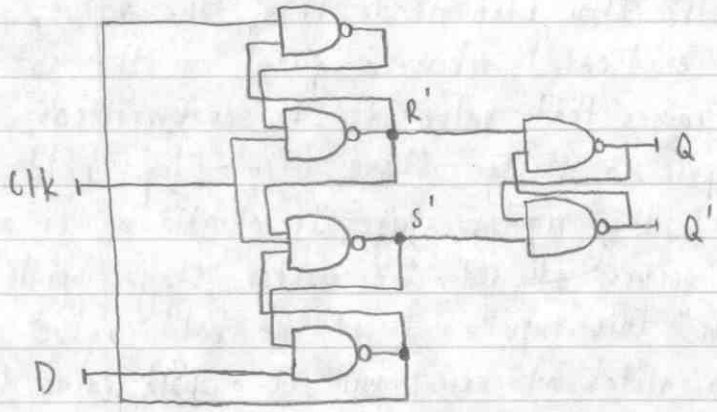
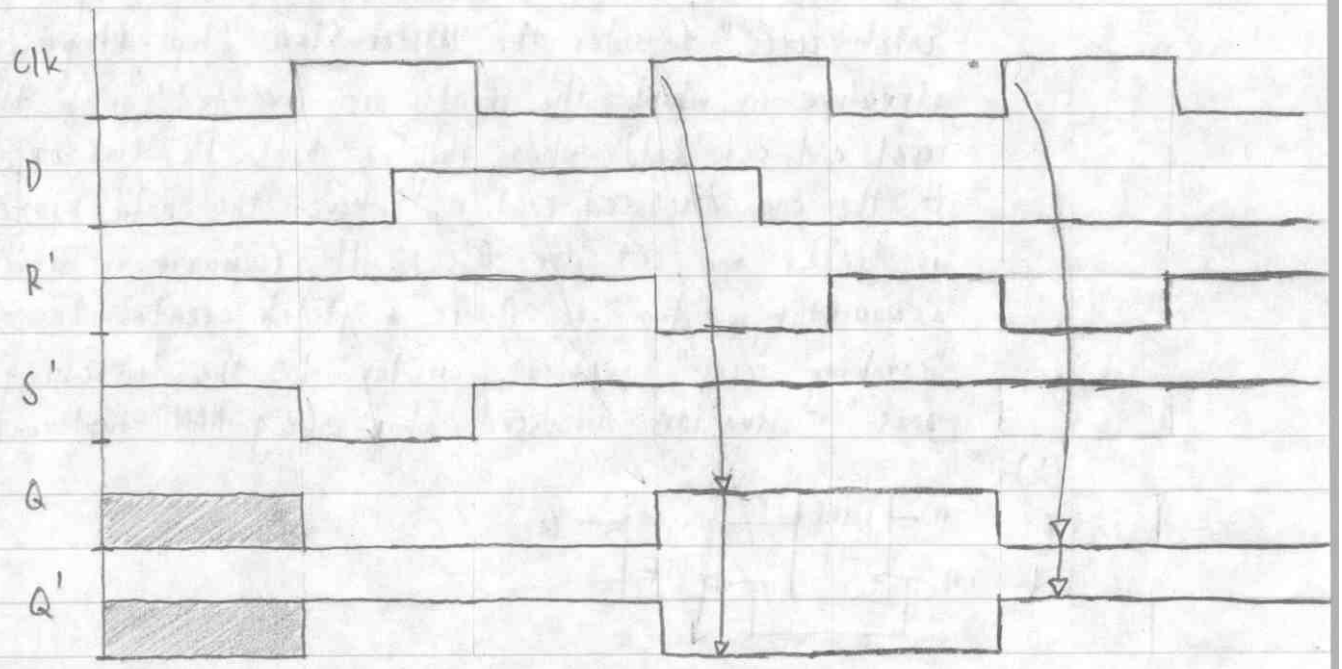
CS150 Spring 2004
Problem Set #4 Solutions

1. (a)



Positive-Edge Triggered Flip-Flop implemented entirely by NAND gates

(b)

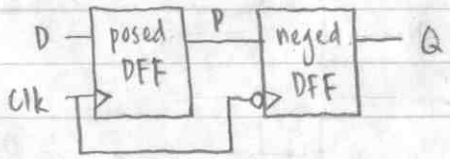


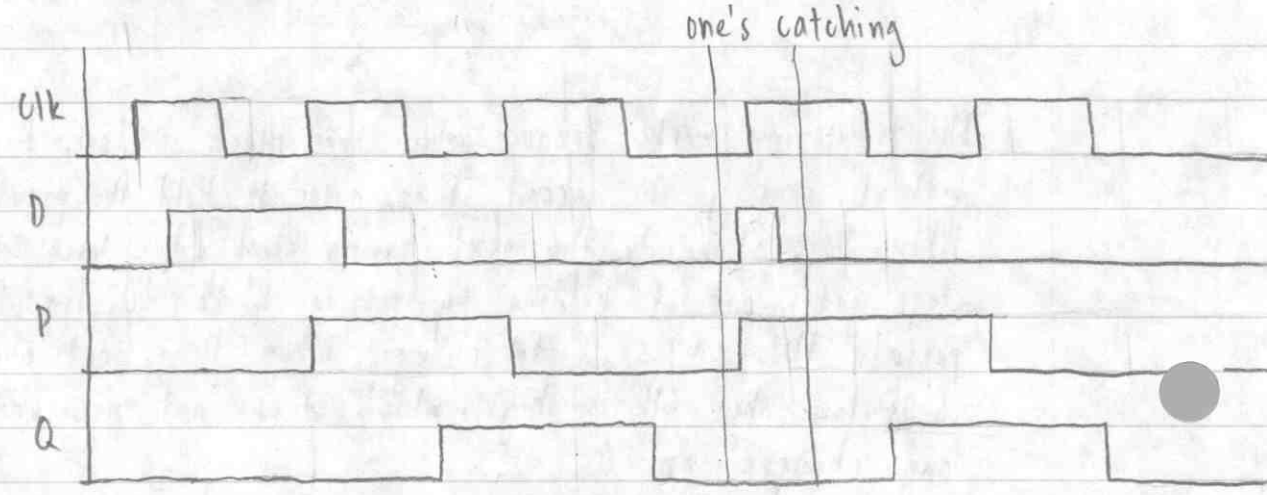
The triggering works because when the clock is low, R' and S' are both 1, causing the second stage gates to hold the previous values. When D is changed, the next rising clock edge forces the second stage gates out of holding by setting R' to 0 for that half period. This causes Q to change. When D is not changed, S' is low when Clk is high, which does not necessarily reflect any changes on Q.

2. (a) - The "ones catching" phenomena occurs in a Master - Slave flip-flop when the flip-flop is to hold some value, only to have it eradicated when a glitch in the set or reset input causes that value to be overwritten. For example, suppose a Master - Slave flip-flop holds the value 0 at the master stage. If a glitch, or a momentary spike, occurs at set, the master stage will read, or "catch" this input and set the hold value to 1. Similarly, a glitch at reset will set a hold value to 0 when the intention is to hold a 1. (Error is unrecoverable.)

- Yes, it is possible for a Master - Slave flip-flop to "catch zeros." Consider the Master-Slave flip-flop structure in which the inputs are inverted, such that reset and set hold when they're both 1's (as opposed to the one discussed earlier, where the hold state is reset and set are 0's). The scenario in which a momentary "dip" to 0 is a glitch creates the "catching zeros" problem, similar to the "catching ones" situation discussed above (e.g. NAND implemented DFF).

(b)





(continued on the next page)

2. (c) No, this system is not "one's catching" since the system corrects itself from the glitch, as shown with the system output Q. The glitch is magnified in Q, but the erroneous value is eventually corrected, as it samples P once it has stabilized. The posedge-negedge DFF's negate the glitch.

3.

```
module nonseq_counter (CLK, RST, state);

    input CLK, RST;
    output [2:0] state;

    reg [2:0] state, nextState;

// state transition
    always @ (posedge CLK) begin
        if (RST)   state <= 3'b000;
        else       state <= nextState;
    end // always @ (posedge CLK)

// next state logic
    always @ (state) begin
        case (state)
            3'b000:    nextState = 3'b111;
            3'b111:    nextState = 3'b010;
            3'b010:    nextState = 3'b101;
            3'b101:    nextState = 3'b001;
            3'b001:    nextState = 3'b110;
            3'b110:    nextState = 3'b100;
            3'b100:    nextState = 3'b011;
            3'b011:    nextState = 3'b000:
            default:   nextState = 3'bxxx;
        endcase;
    end // always @ (state)

endmodule // nonseq_counter
```

```
// traffic light controller

module traffic_light_ctrl (CLK, RST,
                           counter08, counter56,
                           counter08_rst, counter56_rst,
                           ns_tlight_mode, ew_tlight_mode,
                           ns_plight_mode, ew_plight_mode);

    input CLK, RST;
    input[2:0] counter08;
    input[6:0] counter56;

    output [2:0] ns_tlight_mode, ew_tlight_mode;
    output [2:0] ns_plight_mode, ew_plight_mode;

    wire counted08, counter28, counted56;

    reg [2:0] ns_tlight_mode, ew_tlight_mode;
    reg [2:0] ns_plight_mode, ew_plight_mode;
    reg [2:0] crntState, nextState;

    parameter  st_RESTART = 3'b000,
               st_NSG1    = 3'b001,
               st_NSG2    = 3'b010,
               st_NSY     = 3'b011,
               st_EWG1    = 3'b100,
               st_EWG2    = 3'b101,
               st_EWY     = 3'b110;

    parameter  GREEN  = 3'b100,
               YELLOW = 3'b010,
               RED    = 3'b001;

    parameter  WALK = 3'b100,     // walk = lit, don't walk = dim
               WARN = 3'b010,     // walk = dim, don't walk = blink
               STAY = 3'b001;     // walk = dim, don't walk = lit

    assign counted08 = (counter08 == 3'd7);
    assign counted28 = (counter56 == 6'd27);
    assign counted56 = (counter56 == 6'd55);

    always @ (posedge CLK) begin
       if (RST)   crntState <= st_RESTART;
       else       crntState <= st_nextState;
    end // always @ (posedge CLK)

    always @ (*) begin

       nextState = crntState;

       case (crntState)
          st_RESTART:   nextState = st_NSG1;

          st_NSG1: begin
             counter08_rst = 1'b1;
             counter56_rst = 1'b0;
             ns_tlight_mode = GREEN;
```

```
    ns_plight_mode = WALK;
    ew_tlight_mode = RED;
    ew_plight_mode = STAY;

    if (counted28) nextState = st_NSG2;
end // case st_NSG1

st_NSG2: begin
    counter08_rst = 1'b1;
    counter56_rst = 1'b0;
    ns_tlight_mode = GREEN;
    ns_plight_mode = WARN;
    ew_tlight_mode = RED;
    ew_plight_mode = STAY;

    if (counted56) nextState = st_NSY;
end // case st_NSG2

st_NSY: begin
    counter08_rst = 1'b0;
    counter56_rst = 1'b1;
    ns_tlight_mode = YELLOW;
    ns_plight_mode = STAY;
    ew_tlight_mode = RED;
    ew_plight_mode = STAY;

    if (counted08) nextState = st_EWG1;
end // case st_NSY

st_EWG1: begin
    counter08_rst = 1'b1;
    counter56_rst = 1'b0;
    ns_tlight_mode = RED;
    ns_plight_mode = STAY;
    ew_tlight_mode = GREEN;
    ew_plight_mode = WALK;

    if (counted28) nextState = st_EWG2;
end // case st_EWG1

st_EWG2: begin
    counter08_rst = 1'b1;
    counter56_rst = 1'b0;
    ns_tlight_mode = RED;
    ns_plight_mode = STAY;
    ew_tlight_mode = GREEN;
    ew_plight_mode = WARN;

    if (counted56) nextState = st_EWY;
end // case st_EWG2

st_EWY: begin
    counter08_rst = 1'b0;
    counter56_rst = 1'b1;
    ns_tlight_mode = RED;
    ns_plight_mode = STAY;
    ew_tlight_mode = YELLOW;
```

```
            ew_plight_mode = STAY;

            if (counted08) nextState = st_NSG1;
        end // case st_EWY

        default:   nextState = st_RESTART;

    endcase // crntState

end // always @ (*)

endmodule // traffic_light_ctrl
```