Hw 4

Question1:

HW4 Alejandro Sherman

#1 For the RC circuit shown below, the required arrival times at node 1 and 2 are respectively 2ns and 1.2ns.

a) Compute the required arrival time at the source node 0.

$T_{source} = \min(T_i - D_i) = \min\{T_1 - D_1, T_2 - D_2\}$

$D_1 = R_1(C_1+C_2+C_3+C_4+C_5+C_6+(C_7+C_8) + R_2(C_2+C_3+C_4+C_5) + R_3(C_3+C_4+C_5) + R_4(C_4+C_5) + R_5C_5 = 10\times19 + 60\times12 + 60\times9 + 60\times6 + 60\times3 = 1990ps$

$D_2 = R_1(C_1+C_2+C_3+C_4+C_5+C_6+C_7+C_8) + R_6(C_6+C_7+C_8) + R_7(C_7+C_8) + R_8C_8$

$= 10\times19 + 80\times6 + 80\times4 + 80\times2$

$= 1150ps$

$1n = 1000p$

$T_{source} = \min(2{,}000 - 1990,\ 1{,}200 - 1150)$

10ps 50ps

$T_{source} = 10ps$

b) If we insert one buffer with $C_{buf} = 2pF$, $R_{buf} = 10\Omega$ and $D_{buf} = 5ps$ at node x, what is the new required arrival time at node 0?

Updated $D_1 = R_1(C_1 + C_{buf} + C_6 + C_7 + C_8) + R_2(C_{buf}) + D_{buf} + R_{buf}(C_2+C_3+C_4+C_5) + R_3(C_3+C_4+C_5) + R_4(C_4+C_5) + R_5C_5$

$= 10\times9 + 60\times2 + 5 + 10\times12 + 60\times9 + 60\times6 + 60\times3$

$= 1415ps$

Updated $D_2 = R_1(C_1+C_{buf}+C_6+C_7+C_8) + R_6(C_6+C_7+C_8) + R_7(C_7+C_8) + R_8C_8$

$= 10\times9 + 80\times6 + 80\times4 + 80\times2$

$= 1050ps$

$T_{source} = \min(2{,}000 - 1415,\ 1200 - 1050)$

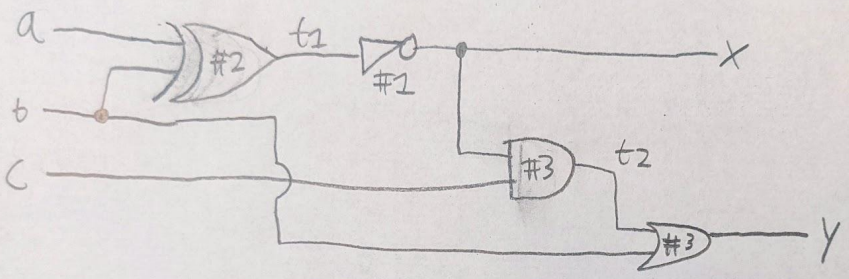
585 150

$T_{source} = 150ps$

Question2:

Question #2

a) Draw the schematic of the logic circuit described by the given verilog:



b) What is the critical path of this design from primary inputs to outputs and what is the delay of the critical path in terms of unit delay?

Critical path:

a → xor → not → and → or → y Delay = 2+1+3+3 = 9

b → xor → not → and → or → y Delay = 2+1+3+3 = 9

Question 3:
Notice that the logic below consists of three repeated parts.
(a) Write a Verilog explicit structural description of the logic which consists of two modules, one module, name it part-logic, will be for the part that's repeated, the other, name it whole-logic, will instantiate part-logic three times and interconnect them appropriately. Write the part-logic using the Verilog primitives. Choose appropriate inputs and outputs for the two modules based on the diagram.

a) part-logic

```
module part-logic( x, y, a, b, c);
        input a, b, c;
        wire t1;
        output x, y;
        not n1(t1, b);
        and a1(x, a, t1);
        or o1(y, c, x);
endmodule
```

whole-logic

```
module whole-logic( x, y, a, b, c);
        input [2:0] a;
        input [2:0] b;
        input c;
        wire t1, t2;
        output [2:0] x;
        output y;
        part-logic pl1(x[0], t1, a[0], b[0], c);
        part-logic pl2(x[1], t2, a[1], b[1], t1);
        part-logic pl3(x[2], y, a[2], b[2], t2);
endmodule
```

(b) Repeat (a) by writing the part-logic using the continuous assignment statement.
b) part-logic

```
module part-logic( x, y, a, b, c);
        input a, b, c;
        output x, y;
        assign x = a & (~b);
        assign y = c | x;
endmodule
```

Question 4: Modified code from lecture slides

```
module xor8(xout, xin1, xin2);
output   [1:8]   xout;
input    [1:8]   xin1, xin2;
        xor  (xout[8], xin1[8], xin2[8]),
             (xout[7], xin1[7], xin2[7]),
             (xout[6], xin1[6], xin2[6]),
             (xout[5], xin1[5], xin2[5]),
             (xout[4], xin1[4], xin2[4]),
             (xout[3], xin1[3], xin2[3]),
             (xout[2], xin1[2], xin2[2]),
             (xout[1], xin1[1], xin2[1]);
endmodule

module deMux(outVector, a, b, c, d, enable);
output   [1:8]   outVector;
input            a, b, c, d enable;
       and   (m12, d, c, ~b, ~a, enable),
             (m11, d, ~c, b, a, enable),
             (m10, d, ~c, b, ~a, enable),
             (m9, d, ~c, ~b, a, enable),
             (m7, ~d, c, b, a, enable),
             (m6, ~d, c, b, ~a, enable),
             (m5, ~d, c, ~b, a, enable),
             (m3, ~d, ~c, b, a, enable);

      assign outVector = {m3, m5, m6, m7, m9, m10, m11, m12};
endmodule

module hamEncode(vIn, valueOut);
input    [1:8]     vIn;
output   [1:12]    valueOut;
wire [1:12]        swap;
wire h1, h2, h4, h8, hold;

     xor   (h1, vIn[1], vIn[2], vIn[4], vIn[5], vIN[7]),
           (h2, vIn[1], vIn[3], vIn[4], vIn[6], vIN[7]),
           (h4, vIn[2], vIn[3], vIn[4], vIn[8]),
           (h8, vIn[5], vIn[6], vIn[7], vIn[8]);

     assign swap = {h1, h2, vIn[1], h4, vIn[2:4], h8, vIn[5:8]};
     assign hold = ^swap;
     assign valueOut = {swap, hold};
endmodule

module hamDecode(vIn, valueOut, doubleerror);
input    [1:12]    vIn;
output   [1:8]     valueOut;
wire   c1, c2, c4, c8;
wire   [1:8]   bitflippers
        xor   (c1, vIn[1], vIn[3], vIn[5], vIn[7], vIn[9], vIn[11]),
              (c2, vIn[2], vIn[3], vIn[6], vIn[7], vIn[10], vIn[11]),
               (c4, vIn[4], vIn[5], vIn[6], vIn[7], vIn[12]),
```

```
            (c8, vIn[8], vIn[9], vIn[6], vIn[7], vIn[12]);

        deMux mux1 (bitFlippers, c1, c2, c4, c8, 1'b1);
        xor8 x1 (valueOut, bitFlippers, {vIn[3], vIn[5], vIn[6], vIn[7], vIn[9], vIn[10], vIn[11], vIn[12]});
        assign doubleerror = ~(^(vIn & (c1 | c2 | c4 | c8)));
endmodule

module testHam();
reg   [1:8] original;
wire  [1:8] regenerated;
wire  [1:12] encoded;
             messedUp;
wire  [1:13] thirteenth;
integer      seed;

      initial begin
            seed = 1;
            forever begin
                  original = $random (seed);
                  thirteenth = $random (seed) & 13'b 0000_0000_0000_0000_1;
                  #1
                  $display ("original=%h, encoded=%h, thirteenth=%h, doubleerror=%h,
                  messed=%h, regen=%h", original, encoded, thirteenth, doubleerror,
                  messedUp, regenerated);
            end
     end
     hamEncode hIn(original, encoded);
     hamDecode hOut(messedUp, regenerated, doubleerror);

     assign messedUp = encoded ^ 12'b 0000_0010_0000 ^ thirteenth;
endmodule
```