ECSE 281 HW

Tyler Yoak

1. Implement the following function using only 74x138 Binary decoders & NAND Gates

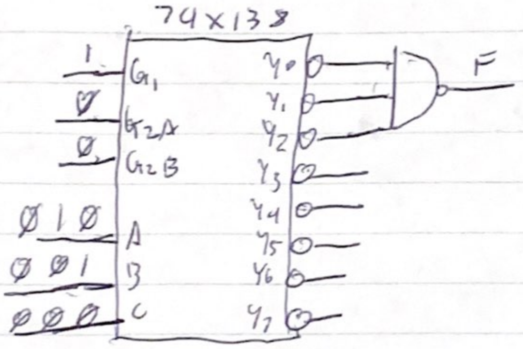
$$F = \prod_{ABC} (3, 4, 5, 6, 7)$$

$$F = \sum_{w,x,y,z} (2, 3, 4, 5, 8, 10, 12, 14)$$

74x138 Binary Decoder → Active Low & NAND gates

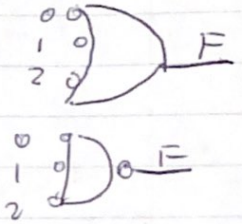
a.

Maxterms: offsets → Product of sums



$F = \prod_{ABC} (3, 4, 5, 6, 7)$

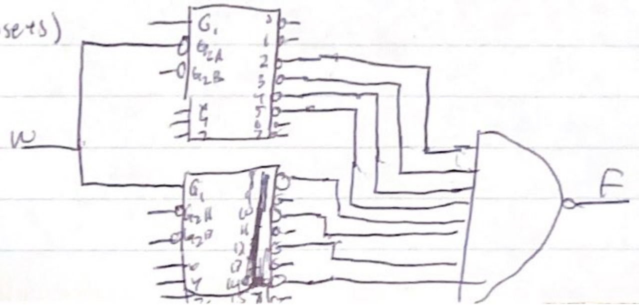
$= F = \sum_{ABC} (1, 2, 3)$



Active Low   NAND F=1   AND F=0

b. $F = \sum_{w,x,y,z} (2, 3, 4, 5, 8, 10, 12, 14)$

Sum of Products = Minterms (Onsets)



Tyler
Youk

2. Design a 10 to 4 encoder w/ the inputs 1 out of 10 code & outputs coded normally for 0-7 [binary 0000-0111]
& 8 coded as E [1110]
& 9 coded as F [1111]

Show the internal circuit

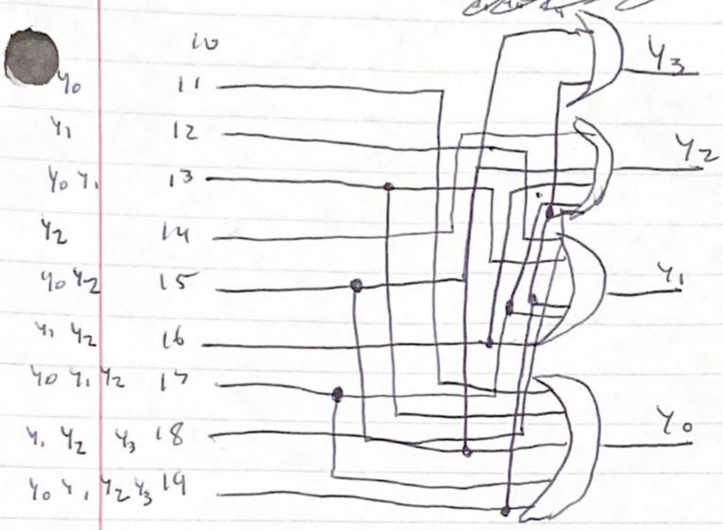
10 to 4 encoder

$Y3: I8 + I9$

$Y2: I4 + I5 + I6 + I7 + I8 + I9$

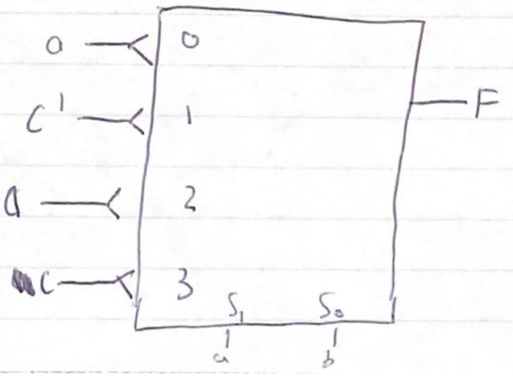
$Y1: I2 + I3 + I6 + I7 + I8 + I9$

$Y0: I_1 + I_3 + I_5 + I_7 + I_9$



3. Implement the following only using a single 4x1 multiplexer & inverters

a	b	c	F	
0	0	0	0	> a
0	0	1	0	
0	1	0	1	> c'
0	1	1	0	
1	0	0	1	> a
1	0	1	1	
1	1	0	0	> c
1	1	1	1	



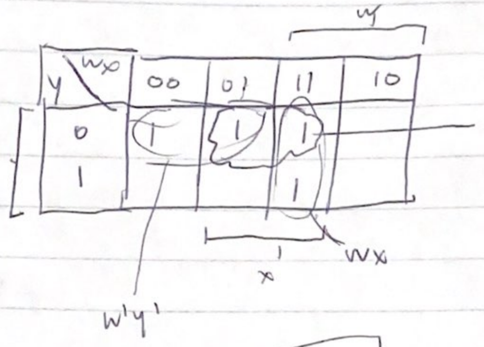
Tyler
Youk

9. For the logic expression below, find all of the static hazards and design a hazard-free circuit that realizes the same logic function.

~~F = w~~ w

$F = w \cdot x + w' \cdot y'$

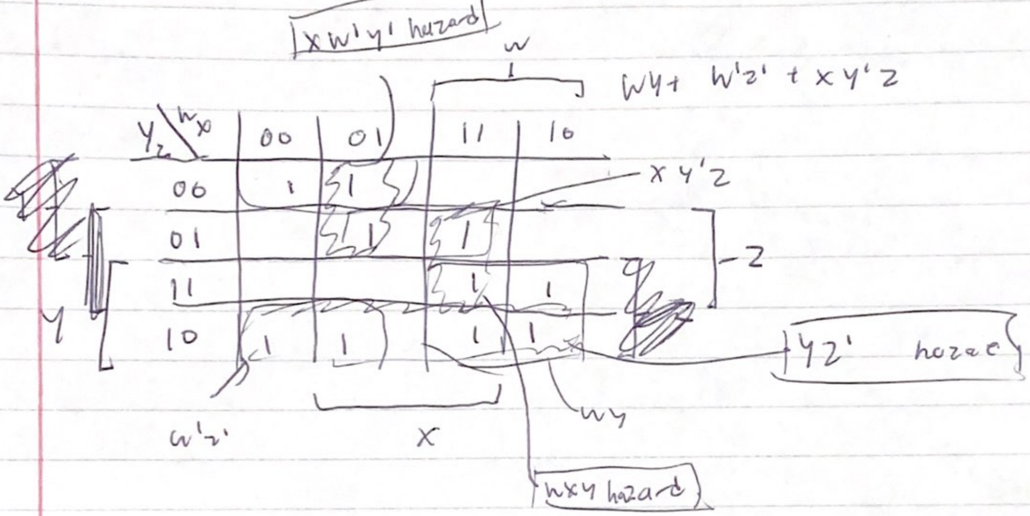
$F = w \cdot y + w' \cdot z' + x \cdot y' \cdot z$



y \ wx	00	01	11	10
0	1	1	1	
1			1	

Static hazard
x . y'

$F = w'y' + wx + xy'$



wy + w'z' + xy'z

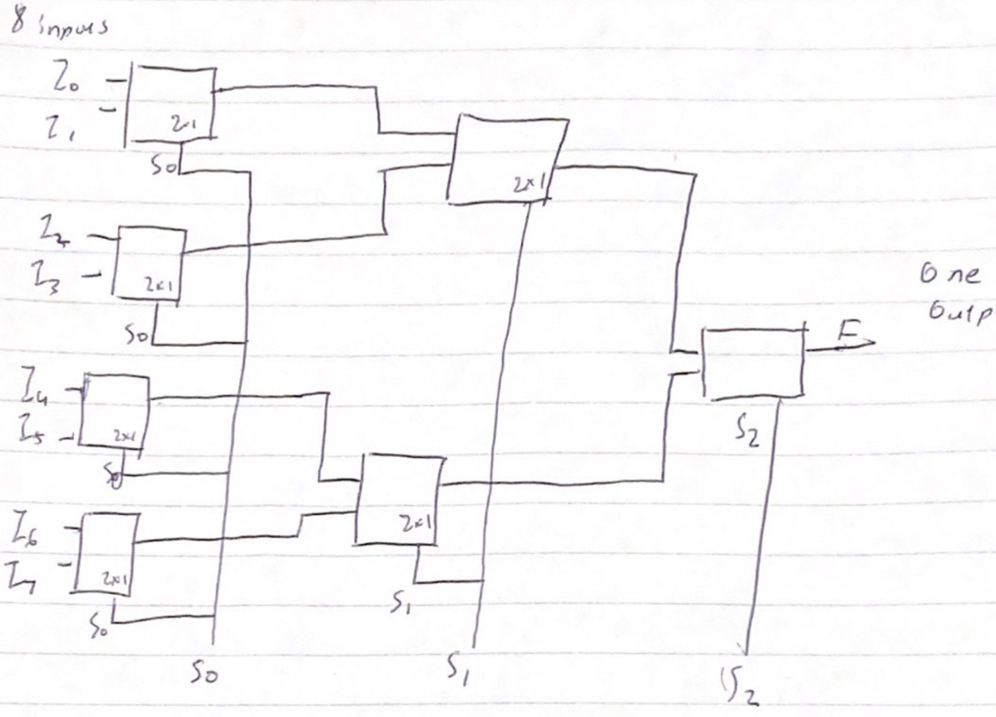
yz \ wx	00	01	11	10
00	1	1		
01		1	1	
11			1	1
10	1	1	1	1

$F = \underbrace{wy + w'z' + xy'z}_{\text{original}} + \underbrace{xw'y' + yz' + wxy}_{\text{hazards}}$

Tyler York

5. Desisn an 8×1 multiplexer (8 data sources/ 1 bit data from each sources)

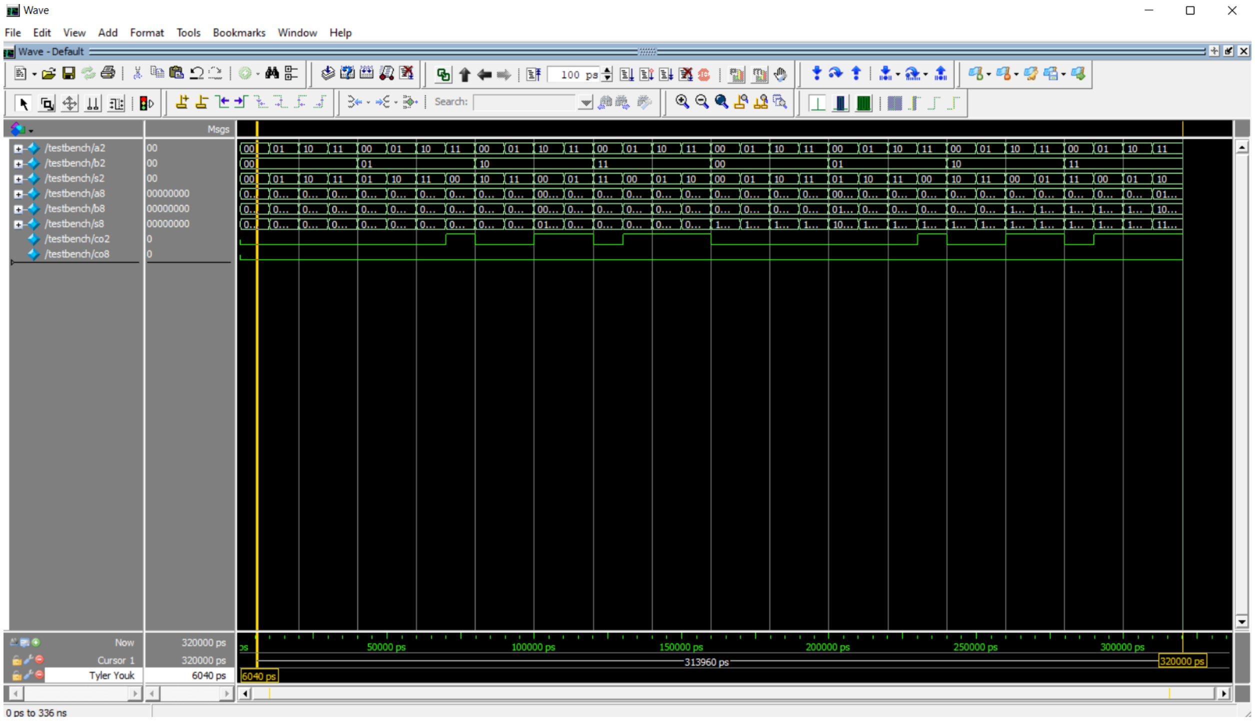
Using 2x1 multiplexes only. You can use as many 2x1 multiplexes as needed. clearly label all inputs and outputs.



```
// Tyler Youk Source Code
// HW Assignment #8

//cla_adder_homework5.sv source code
module cla_adder #(
    parameter N = 8
) (
    input logic [N-1:0] a, b,
    input logic c_in,
    output logic [N-1:0] s,
    output logic c_out

);

    logic [N-1:0] p, g;
    logic [N:0]   c;

    assign p = a ^ b;
    assign g = a & b;

    for (genvar i = 0; i <= N; i++) begin
        if (i == 0)
            assign c[i] = c_in;
        else
            assign c[i] = g[i-1] | p[i-1] & c[i-1];
    end

    assign s = c^p; //COMPLETE
    assign c_out = c[N]; //COMPLETE
endmodule




//Testbench_homework5.sv source code
`timescale 1ns/10ps
module testbench ();

    logic [1:0] a2, b2, s2;
    logic [7:0] a8, b8, s8;
    logic       co2, co8;


//2 bit cla adder
```

```
    cla_adder #(
        .N(2)
    ) UUT2 (
            .a(a2),
            .b(b2),
            .c_in(1'b0),
            .s(s2),
            .c_out(co2)
    );

//8 bit cla adder
    cla_adder #(
        .N(8)
    ) UUT8 (
        .a(a8),
            .b(b8),
            .c_in(1'b0),
            .s(s8),
            .c_out(co8)
    );

    initial begin
        a2 = 0;
        forever
            #10 a2++;
    end

    initial begin
        b2 = 0;
            forever
                        #40 b2++;
    end

// COMPLETE for a8 that will increase by 3 every 10 time units
            initial begin

                        a8 = 0;
                        forever
                                    #10 a8+=3;
            end

// COMPLETE for b8 that will increase by 5 every 10 time units
            initial begin
            b8 = 0;
            forever
                  #10 b8+=5;
      end



  initial begin

    #320 $finish();
end

endmodule
```

//Waveform screenshot with Tyler Youk tag



```
/testbench/a2     00
/testbench/b2     00
/testbench/s2     00
/testbench/a8     00000000
/testbench/b8     00000000
/testbench/s8     00000000
/testbench/co2    0
/testbench/co8    0

Now               320000 ps
Cursor 1          320000 ps
Tyler Youk        6040 ps
```

0 ps to 336 ns