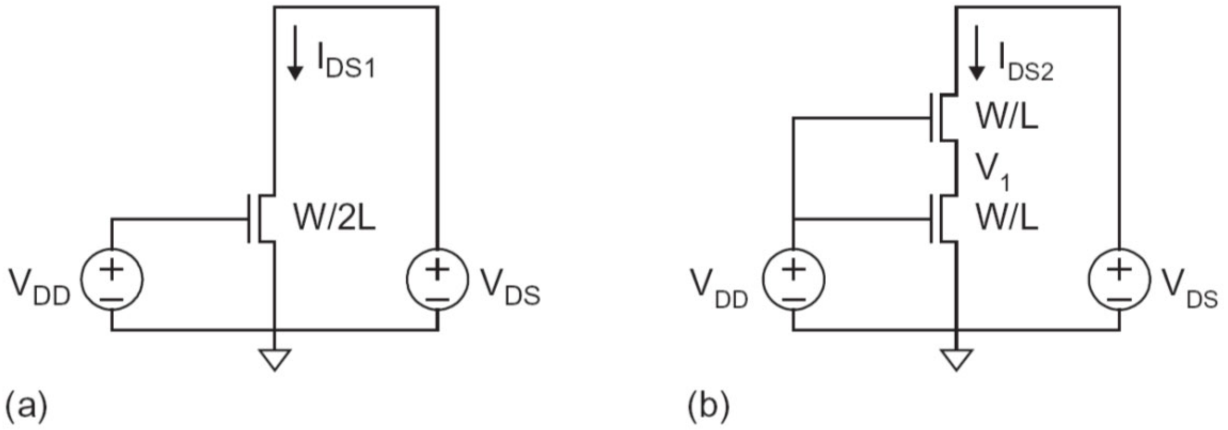
1. (10) Problem 2.2



2.2 In (a), the transistor sees V_{gs} = V_{DD} and V_{ds} = V_{DS}. The current is

$$I_{DS1}=\frac{\beta}{2}\left(V_{DD}-V_t-\frac{V_{DS}}{2}\right)V_{DS}$$

(a)s transistor is 2X length of (b)s. Thus β(a) is ½ β(b)

In (b), the bottom transistor sees V_{gs} = V_{DD} and V_{ds} = V_1. The top transistor sees V_{gs} = V_{DD} - V_1 and V_{ds} = V_{DS} - V_1. The currents are

$$I_{DS2}=\underbrace{\beta\left(V_{DD}-V_t-\frac{V_1}{2}\right)V_1}_{\text{Bottom Transistor}}=\underbrace{\beta\left((V_{DD}-V_1)-V_t-\frac{(V_{DS}-V_1)}{2}\right)(V_{DS}-V_1)}_{\text{Top Transistor}}$$

Solving for V_1, we find

$$V_1=(V_{DD}-V_t)-\sqrt{(V_{DD}-V_t)^2-\left(V_{DD}-V_t-\frac{V_{DS}}{2}\right)V_{DS}}$$

Substituting V_1 indo the I_{DS2} equation and simplifying gives I_{DS1} = I_{DS2}.

Note: V1 != VDS/2, even when VDD = VDS (KEY QUESTION – WHY????)

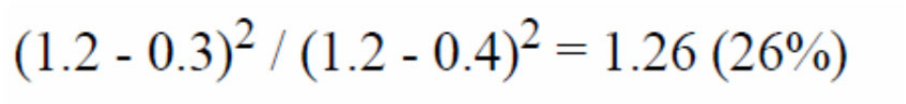
Examples for 65nm tech:

VDD	1
VDS	1
Vt	0.3
V1	0.161484
IDS1/Beta	0.1
IDS2Lower/Beta	0.1
IDSUpper/Beta	0.1

VDD	1.3
VDS	1
Vt	0.3
V1	0.292893
IDS1/Beta	0.25
IDS2Lower/Beta	0.25
IDSUpper/Beta	0.25

2. (10) W&H problem 2.10a.

$I_{dsat} = \beta V_{GT}^{2}/2$. In this case $V_{GT} = V_{gs} – V_{T} = V_{dd} - V_{T}$

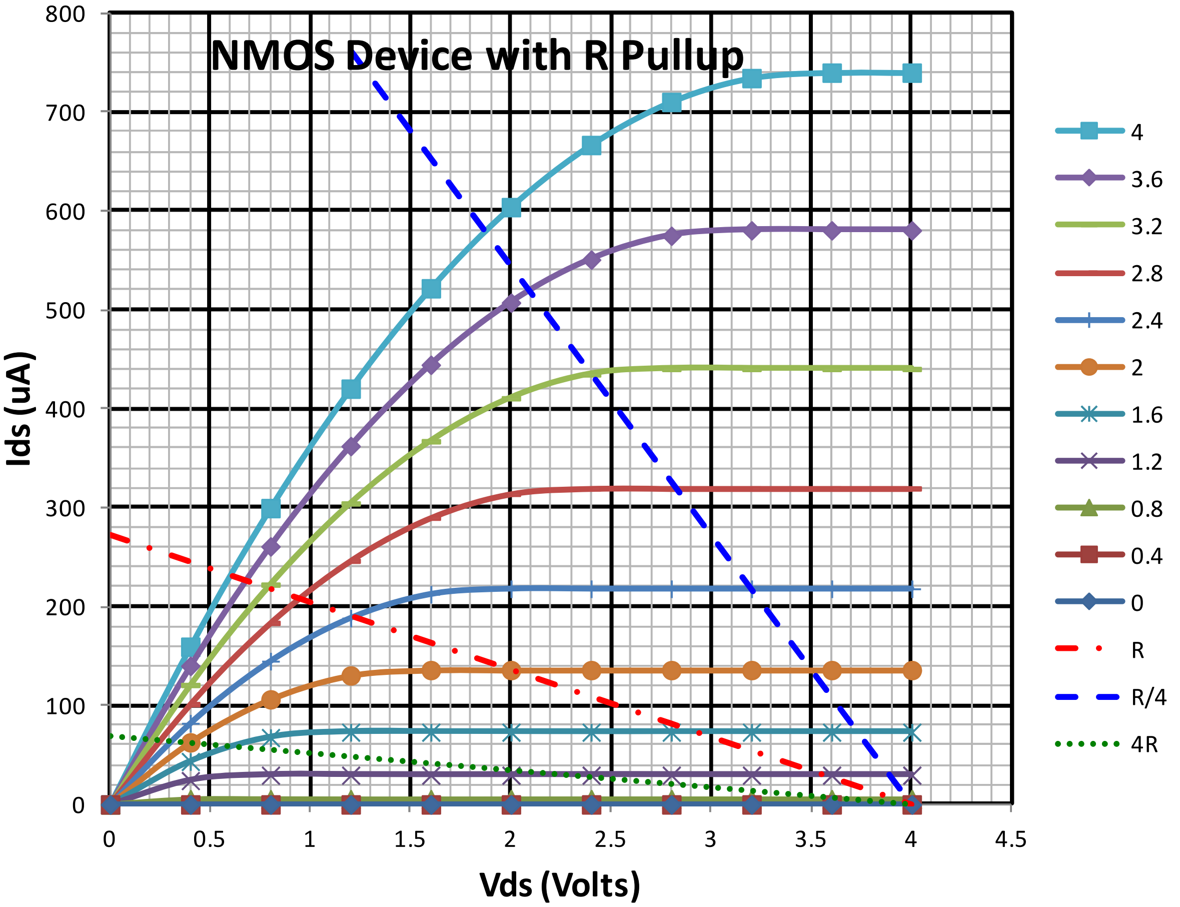


$$(1.2 - 0.3)^2 / (1.2 - 0.4)^2 = 1.26\ (26\%)$$

3. (10) W&H 2.20a. – i.e. develop using the Long Channel Model the IV equation for two identical NMOS transistors in series.

both transistors are on. 0V

4. (10)Using the Excel spreadsheet on the class website, the ND technology, an NMOS invertor where the width is 4λ, and a Vdd of 4V, what is the resistor value you would need to have a Vgs of 2V give a Vds of 2V as output. What is the maximum on current when Vgs=4V.



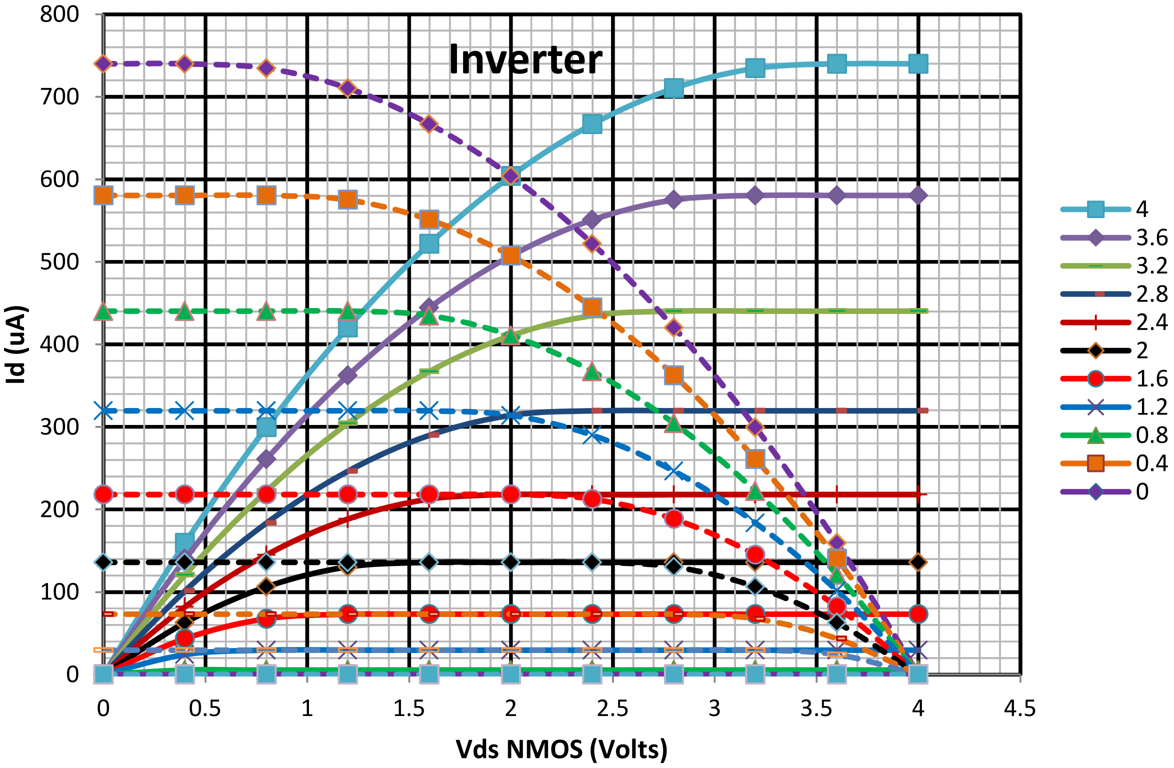
For Vds=2V when Vgs=2V, load line must go thru (4,0) and cross Vds=2 when Vgs=2, at about 135uA, or Resistance of 2V/135uA = 14.71K – the dotted red line. The max current is then when Vgs=4V (blue line) and crosses the red line at about 230uA.

Note here the width is **4λ, not 4L**. 4λ is 2L. If you did 4L you would get 7.14K and 280uA.

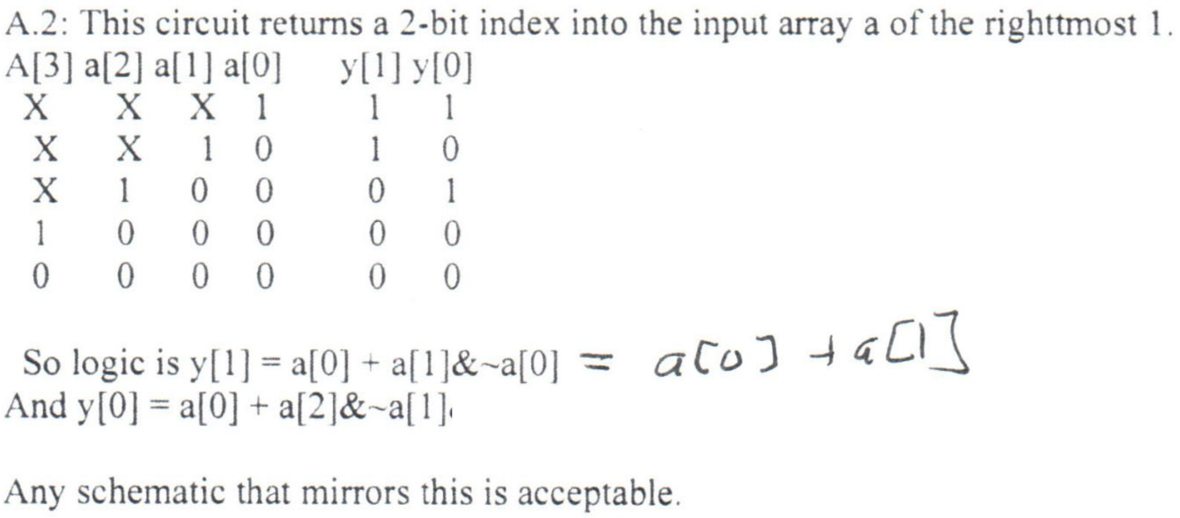
5. (10)Again using the Excel spreadsheet for a CMOS invertor where the NMOS is as in the above problem, what is the PMOS width you would need so that Ain = 2V gives Aout of 2V? What is the max current for an input of 4V; What is it for an input of 0V?

Note: the parameter set in the spreadsheet has the mobility of the ptype twice, not ½ of the n type. My fault. I left it as is below, and simply adjusted width to get the 2,2 point, at a width of 1 and current of about 140uA.

Max current at 4V VGgs is 0, at 0V is 0 (Its where the N and P cross)



6. (10) Book A2



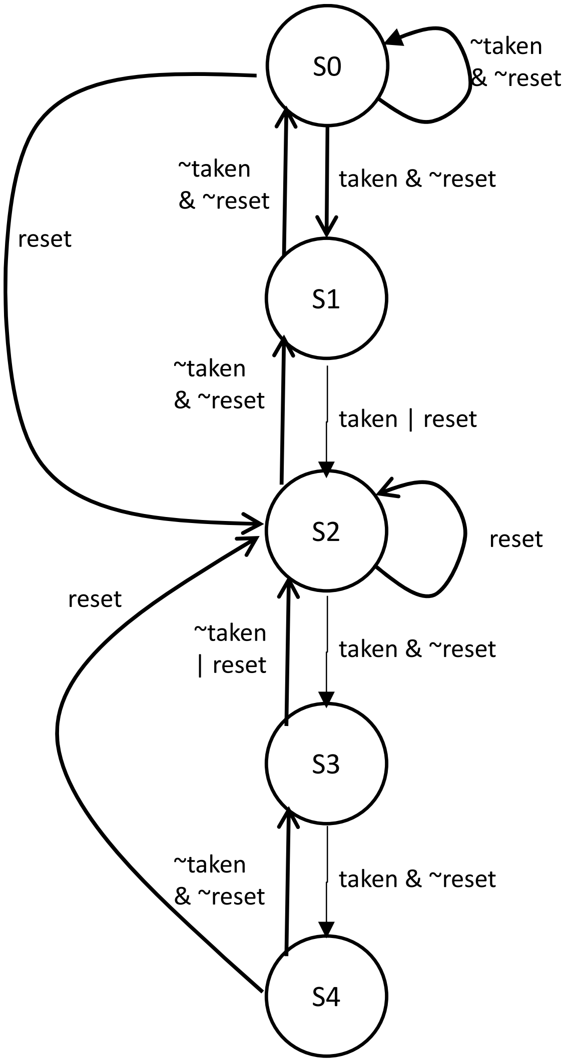
A.2: This circuit returns a 2-bit index into the input array a of the righttmost 1.

A[3]	a[2]	a[1]	a[0]	y[1]	y[0]
X	X	X	1	1	1
X	X	1	0	1	0
X	1	0	0	0	1
1	0	0	0	0	0
0	0	0	0	0	0

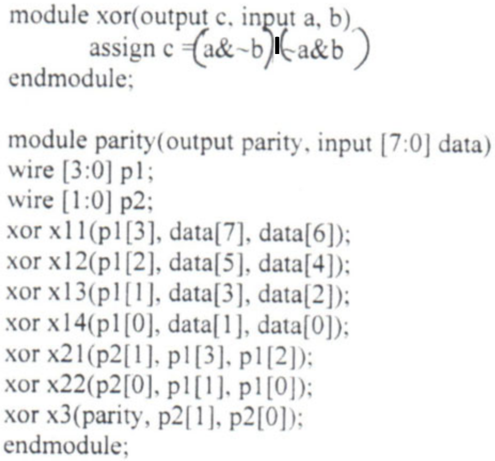
So logic is y[1] = a[0] + a[1]&~a[0] = a[0] + a[1]
And y[0] = a[0] + a[2]&~a[1]

Any schematic that mirrors this is acceptable.

7. (10) A16. Took off -1 if ~reset not included on taken paths, took off 2 if reset to S2 not included



8. (10) Write down Verilog code for a behavioral model for a 2-input exclusive or gate using just "&", "|", and "~". Then using this module, develop a structural module for an 8 bit parity generator – 1.e. there are 8 dAta inputs which should be combined by 3 l3v3ls of xors to compute the odd parity of the data bits. This parity should be output.



```
module xor(output c, input a, b)
        assign c =(a&~b)|(~a&b )
endmodule;

module parity(output parity, input [7:0] data)
wire [3:0] p1;
wire [1:0] p2;
xor x11(p1[3], data[7], data[6]);
xor x12(p1[2], data[5], data[4]);
xor x13(p1[1], data[3], data[2]);
xor x14(p1[0], data[1], data[0]);
xor x21(p2[1], p1[3], p1[2]);
xor x22(p2[0], p1[1], p1[0]);
xor x3(parity, p2[1], p2[0]);
endmodule;
```

9. (20) Write the behavioral code for a module named “grey” that implements a 4-bit “Grey code” (see https://en.wikipedia.org/wiki/Gray_code) counter with the following characteristics:

1. An input CLk that is the clock
2. An input Reset that when high resets the counter to 0000 when the Clk goes from low to high
3. An input Count that when high, and when the Clk goes from low to high (while Reset is low) advances the counter to the next Grey code value. When Count is low, the outputs stay unchanged.

Gray code table from Wikipedia (several are possible)

Decimal	**Binary**	**Gray**
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Note for the above code (other codes have different patterns):

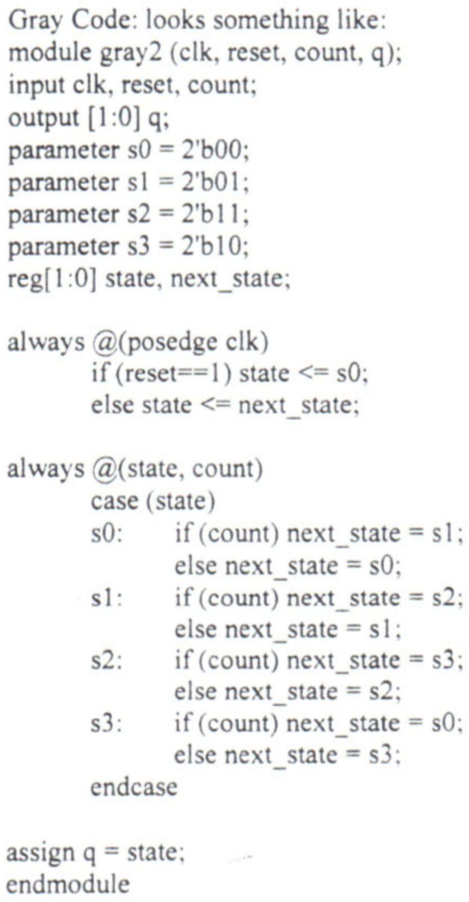
- greycode[3] always same as the binary[3].
- greycode[0] = binary[0] xor binary[1],
- greycode[1] = binary[2] xor binary[1],
- greycode [2] = binary[2] xor binary[3]

Several ways to generate this: simplest is like HW5.16 that steps thru 16 states in a big case statement.

A bit shorter (again for this code):

```
module grey(clk,reset,count,code);
        input clk,reset,count;
        output [3:0] code;
        reg [3:0] counter;
        always@(posedge clk)
        begin
                if (reset == 1) begin
                        counter <= 4’b0000;
                end
                else if (count == 1) begin
                        code[3] <= counter[3];
                        code[2] <= counter[3] ^ counter[2];
                        code[1] <= counter[2] ^ counter[1];
                        code[0] <= counter[1] ^ counter[0];
                end
        end
endmodule
```

A simple 2-bit gray code



Gray Code: looks something like:

```
module gray2 (clk, reset, count, q);
input clk, reset, count;
output [1:0] q;
parameter s0 = 2'b00;
parameter s1 = 2'b01;
parameter s2 = 2'b11;
parameter s3 = 2'b10;
reg[1:0] state, next_state;

always @(posedge clk)
        if (reset==1) state <= s0;
        else state <= next_state;

always @(state, count)
        case (state)
        s0:     if (count) next_state = s1;
                else next_state = s0;
        s1:     if (count) next_state = s2;
                else next_state = s1;
        s2:     if (count) next_state = s3;
                else next_state = s2;
        s3:     if (count) next_state = s0;
                else next_state = s3;
        endcase

assign q = state;
endmodule
```