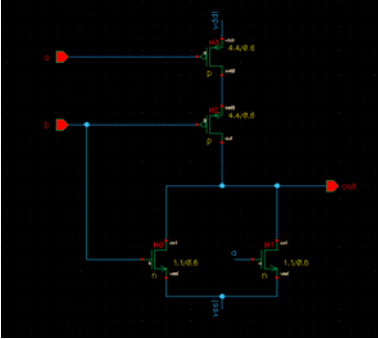


CS/EE 5710/6710

Layout  
Basic Transistor Sizing  
Intro to Verilog

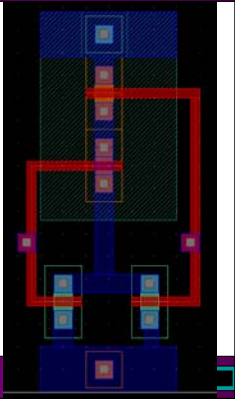
An Example: NOR



► NOR schematic in Composer

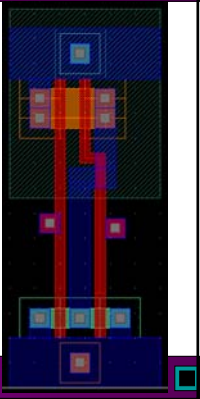
First Layout: Follow Schematic

- Note that layout of transistors follows the schematic
  - Two P-types in series pulling up
  - Two N-types in parallel pulling down

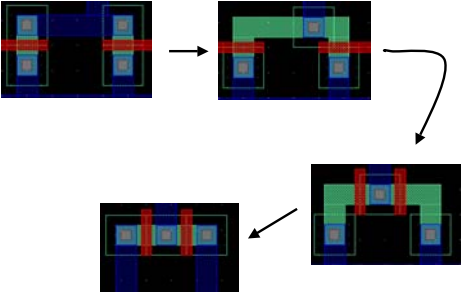


Another Layout: Better?

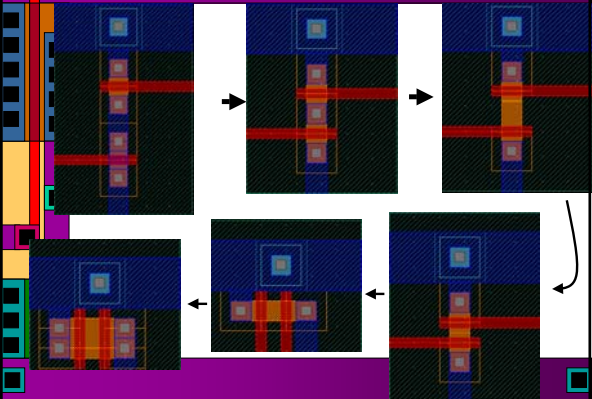
- Same four transistors
  - But, organized a little differently
  - And sized a little differently

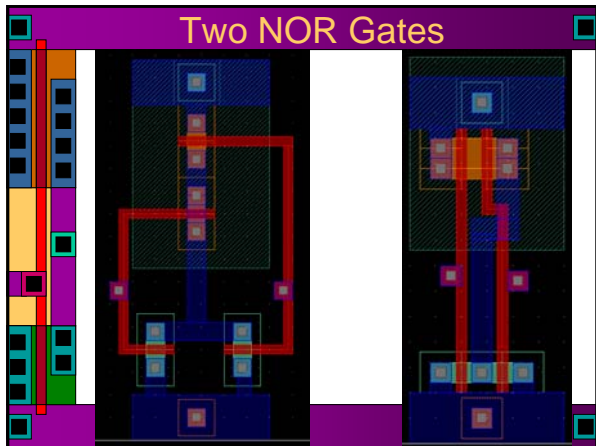


Use Shared Source/Drain



Another Shared S/D





### Transistor Sizing

- ▶ We'll get into the details later...
- ▶ Consider a transistor's Width and Length
  - ▶ Current capability is proportional to  $W/L$
  - ▶ Length is almost always minimum allowed
  - ▶ Change width to change current capability

### Sizing Rule of Thumb

- ▶ Also, P-type is about twice as bad as N-type
  - ▶ Has to do with hole mobility vs. electron mobility
- ▶ So, make P-types twice as wide as N-types to start with
- ▶ Unit size for transistors this semester
  - ▶ N-type 1.2u (contact pitch)
  - ▶ P-type 2.4u

### Sizing Rule of Thumb

- ▶ Now multiply each width by  $n$  for a series stack of  $n$  transistors.
  - ▶ Stack of 2, each transistor should be 2x unit size
  - ▶ Stack of 3, each transistor should be 3x unit size
- ▶ This is because series connections are like increasing the  $L$  of the device...
  - ▶ Current is proportional to  $W/L$

### For example:

- ▶ Notice the difference in width...
- ▶ This roughly equalizes the current sourcing capability of pull-up and pull-down stacks in this gate

### And now for something completely different...

- ▶ A little Verilog...
- ▶ Big picture: Two main Hardware Description Languages (HDL) out there
  - ▶ VHDL
    - ▶ Designed by committee on request of the Department of Defense
    - ▶ Based on Ada
  - ▶ Verilog
    - ▶ Designed by a company for their own use
    - ▶ Based on C
- ▶ Both now have IEEE standards
- ▶ Both are in wide use

## Data Types

- ▶ Possible Values:
  - ▶ 0: logic 0, false
  - ▶ 1: logic 1, true
  - ▶ X: unknown logic value
  - ▶ Z: High impedance state
- ▶ Registers and Nets are the main data types
- ▶ Integer, time, and real are used in behavioral modeling, and in simulation

## Registers

- ▶ Abstract model of a data storage element
- ▶ A reg holds its value from one assignment to the next
  - ▶ The value “sticks”
- ▶ Register type declarations
  - ▶ `reg a;` // a scalar register
  - ▶ `reg [3:0] b;` // a 4-bit vector register

## Nets

- ▶ Nets (wires) model physical connections
- ▶ They don't hold their value
  - ▶ They must be driven by a “driver” (i.e. a gate output or a continuous assignment)
  - ▶ Their value is Z if not driven
- ▶ Wire declarations
  - ▶ `wire d;` // a scalar wire
  - ▶ `wire [3:0] e;` // a 4-bit vector wire
- ▶ There are lots of types of regs and wires, but these are the basics...

## Memories

- ▶ Verilog memory models are arrays of regs
- ▶ Each element in the memory is addressed by a single array index
- ▶ Memory declarations:
  - ▶ `reg [7:0] imem[0:255];` // a 256 word 8-bit memory
  - ▶ `reg [31:0] dmem[0:1023];` // a 1k word memory with 32-bit words

## Other types

- ▶ Integers:
  - ▶ `integer i, j;` // declare two scalar ints
  - ▶ `integer k[7:0];` // an array of 8 ints
- ▶ `$time` - returns simulation time
  - ▶ Useful inside `$display` and `$monitor` commands...

## Number Representations

- ▶ Constant numbers can be decimal, hex, octal, or binary
- ▶ Two forms are available:
  - ▶ Simple decimal numbers: 45, 123, 49039...
  - ▶ `<size>'<base><number>`
  - ▶ base is d, h, o, or b
  - ▶ `4'b1001` // a 4-bit binary number
  - ▶ `8'h2fe4` // an 8-bit hex number

## Relational Operators

- ▶  $A < B$ ,  $A > B$ ,  $A <= B$ ,  $A >= B$ ,  $A == B$ ,  $A != B$ 
  - ▶ The result is 0 if the relation is false, 1 if the relation is true, X if either of the operands has any X's in the number
- ▶  $A === B$ ,  $A !== B$ 
  - ▶ These require an exact match of numbers, X's and Z's included
- ▶  $!$ ,  $\&\&$ ,  $||$ 
  - ▶ Logical not, and, or of expressions
- ▶  $\{a, b[3:0]\}$  - example of concatenation

## Block Structures

- ▶ Two types:
  - ▶ **always** // repeats until simulation is done  
begin  
...  
end
  - ▶ **initial** // executed once at beginning of sim  
begin  
...  
end

## Example

- ▶ Reg [1:0] a,b;
  - initial begin** // only executed once
  - a = 2'b01; // initialize a
  - b = 2'b10; // initialize b
  - end**
  - always begin** // repeated until simulation done
  - #50 a = ~a; // a inverts every 50 time units
  - end**
  - always begin** // repeated until simulation done
  - #100 b = ~b; // b inverts every 100 time units
  - end**
- ▶ Note timing control: #50 = delay for 50 time units

## Conditional, For

- ▶ If ( $\langle \text{expr} \rangle$ )  $\langle \text{statement} \rangle$  else  $\langle \text{statement} \rangle$ 
  - ▶ else is optional and binds with closest previous if that lacks an else
  - ▶ if (index > 0)
    - if (rega > regb)
      - result = rega;
    - else
      - result = regb;
- ▶ For is like C
  - ▶ for (initial; condition; step)
  - ▶ for (k=0; k<10; k=k+1)
    - statement;

## Basic Testbench

```

initial
begin
a[1:0] = 2'b00;
b[1:0] = 2'b00;
cin = 1'b0;
$display("Starting...");
#20
$display("A = %b, B = %b, c = %b, Sum = %b, Cout = %b", a, b, cin, sum, cout);
if (sum != 00) $display("ERROR: Sum should be 00, is %b", sum);
if (cout != 0) $display("ERROR: cout should be 0, is %b", cout);
a = 2'b01;
#20
$display("A = %b, B = %b, c = %b, Sum = %b, Cout = %b", a, b, cin, sum, cout);
if (sum != 00) $display("ERROR: Sum should be 01, is %b", sum);
if (cout != 0) $display("ERROR: cout should be 0, is %b", cout);
b = 2'b01;
#20
$display("A = %b, B = %b, c = %b, Sum = %b, Cout = %b", a, b, cin, sum, cout);
if (sum != 00) $display("ERROR: Sum should be 10, is %b", sum);
if (cout != 0) $display("ERROR: cout should be 0, is %b", cout);
$display("...Done");
$finish;
end
  
```

## Nifty Testbench

```

reg [1:0] ainarray [0:4]; // define memory arrays to hold input and result
reg [1:0] binarray [0:4];
reg [2:0] resultsarray [0:4];
integer i;
initial begin
$readmemb("ain.txt", ainarray); // read values into arrays from files
$readmemb("bin.txt", binarray);
$readmemb("results.txt", resultsarray);
a[1:0] = 2'b00; // initialize inputs
b[1:0] = 2'b00;
cin = 1'b0;
$display("Starting...");
#10 $display("A = %b, B = %b, c = %b, Sum = %b, Cout = %b", a, b, cin, sum, cout);
for (i=0; i<=4; i=i+1) // loop through all values in the memories
begin
a = ainarray[i]; // set the inputs from the memory arrays
b = binarray[i];
#10 $display("A = %b, B = %b, c = %b, Sum = %b, Cout = %b", a, b, cin, sum, cout);
if ((cout,sum) != resultsarray[i])
$display("Error: Sum should be %b, is %b instead", resultsarray[i],sum); // check results array
end
$display("...Done");
$finish;
end
  
```