Netlisting, schematic capture, cell libraries

Jennifer Volk[1], Alexander Wynn[2], Andreas Olofsson[3]

February 23, 2026

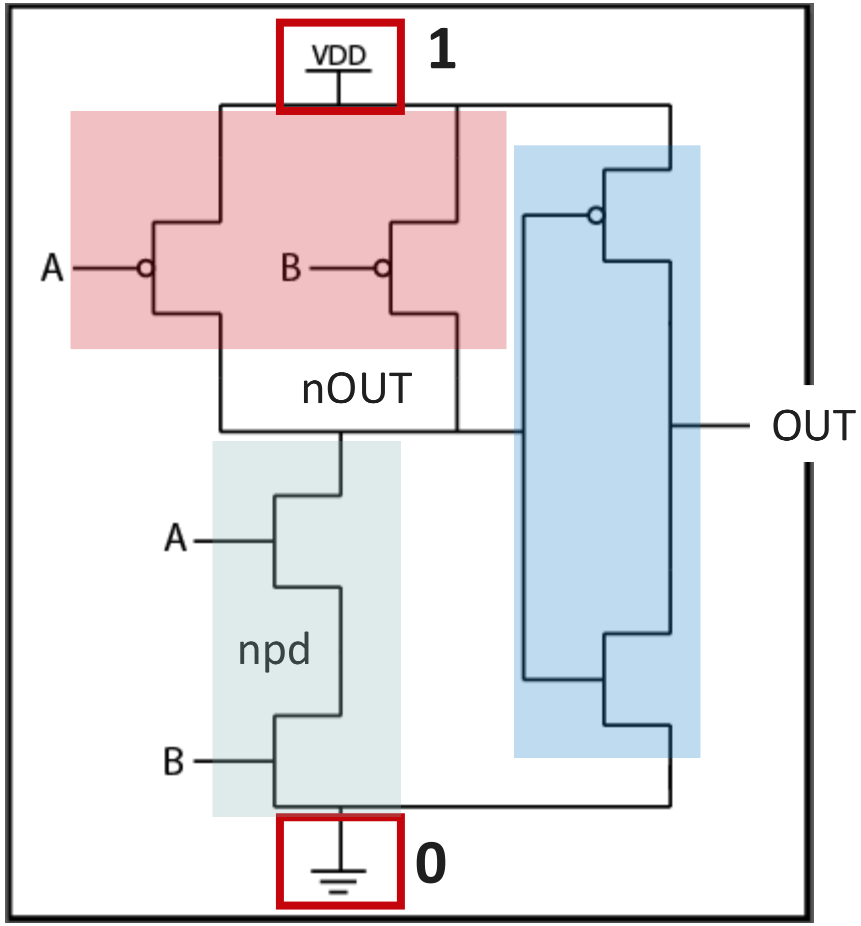
[1] University of Wisconsin-Madison
[2] Adiabatic Machines and MIT
[3] Zero ASIC

In today's class…

1. Review
2. Schematic capture & translation to netlist
3. Further netlisting breakdown
4. Common SPICE errors & commands
5. How to save results and plot them in Python
6. Propagation delays
7. Corner cases
8. Homework assignment

Example netlist

Let's look at the AND gate again…



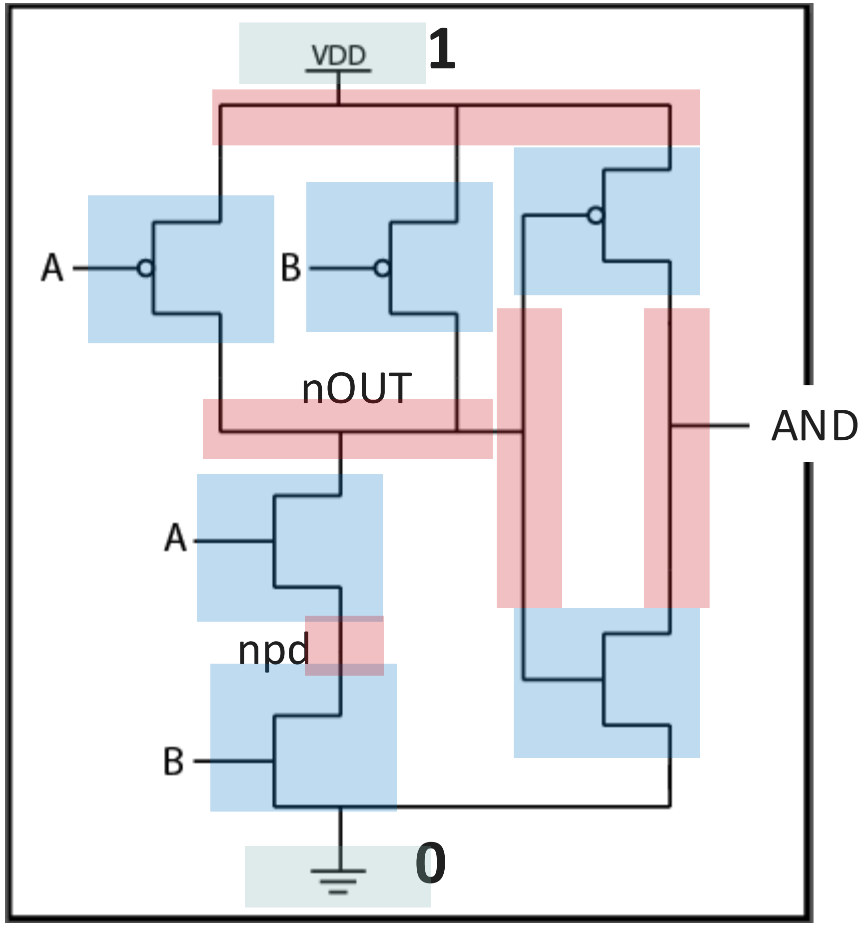

```
* 1 V power supply
vsup VDD 0 1
```

```
* AND pull-up network -- two pmos in parallel
Mp1 nOUT A VDD VDD mosp L=0.35u W=2u
Mp2 nOUT B VDD VDD mosp L=0.35u W=2u
```

```
* Pull-down network -- two nmos in series
Mn1 nOUT A npd 0 mosn L=0.35u W=2u
Mn2 npd B 0 0 mosn L=0.35u W=2u
```

```
* Inverter, or a logical NOT
Mp3 AND nOUT VDD VDD mosp L=0.35u W=2u
Mn3 AND nOUT 0 0 mosn L=0.35u W=2u
```

Schematic capture



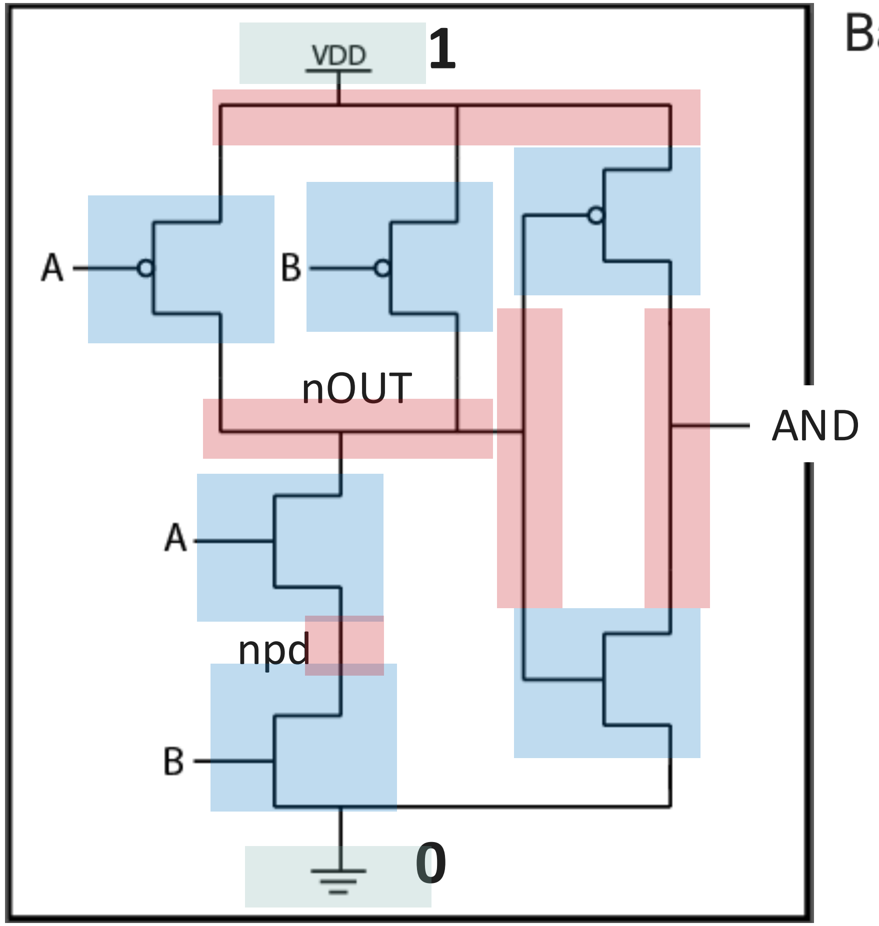
Wires (typically there's a dot at intersections)

Devices

Power/voltage references

Text = net labels (define label for a connected node)

Schematic/netlist elements



Basic Schematic Elements

Element	Symbol	Purpose
Wire	Line	Electrical connection
Net label	Text	Names a wire (same name = connected)
Instance	Symbol	A component (resistor, transistor, etc.)
Port	I/O marker	Input/output of the circuit
Ground	GND symbol	Reference voltage (0V)

Netlisting instance examples

Netlist Elements

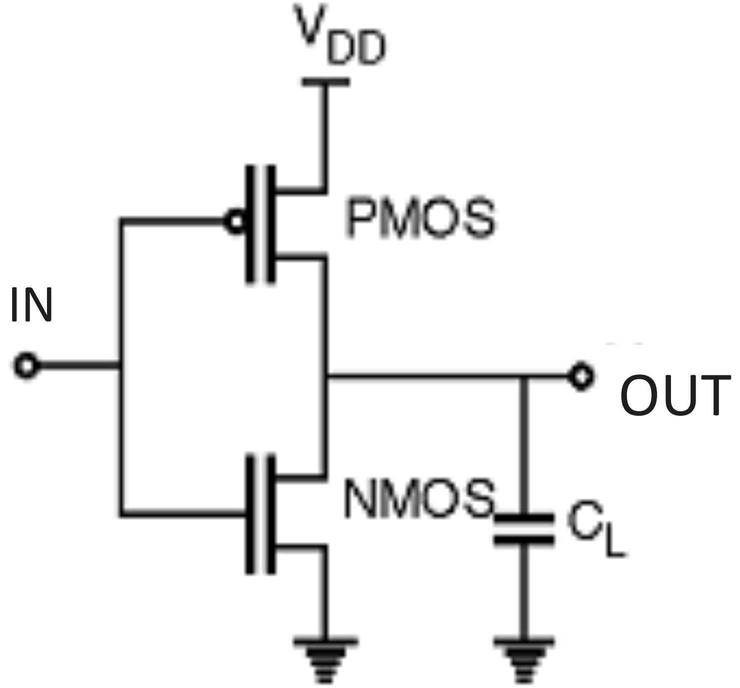
First Letter	Component
R	Resistor
C	Capacitor
L	Inductor
V	Voltage source
I	Current source
M	MOSFET
X	Subcircuit instance

Format

New library—closer-to-industry models

```
* Inverter circuit
.include "sky130.lib"

* Power supply
Vdd vdd gnd 1.8

* Input signal (rise/fall times realistic for 130nm)
Vin in gnd PULSE(0 1.8 0 100p 100p 1n 2n)

* PMOS: drain gate source body
Mp out in vdd vdd sky130_fd_pr__pfet_01v8 W=1u L=150n

* NMOS: drain gate source body
Mn out in gnd gnd sky130_fd_pr__nfet_01v8 W=0.5u L=150n

* Simulation commands
.tran 10p 10n
.end
```

A simple simulation (NOT gate)



A simple simulation (NOT gate)

Step 1: Write your netlist file. Save as inverter.spice.

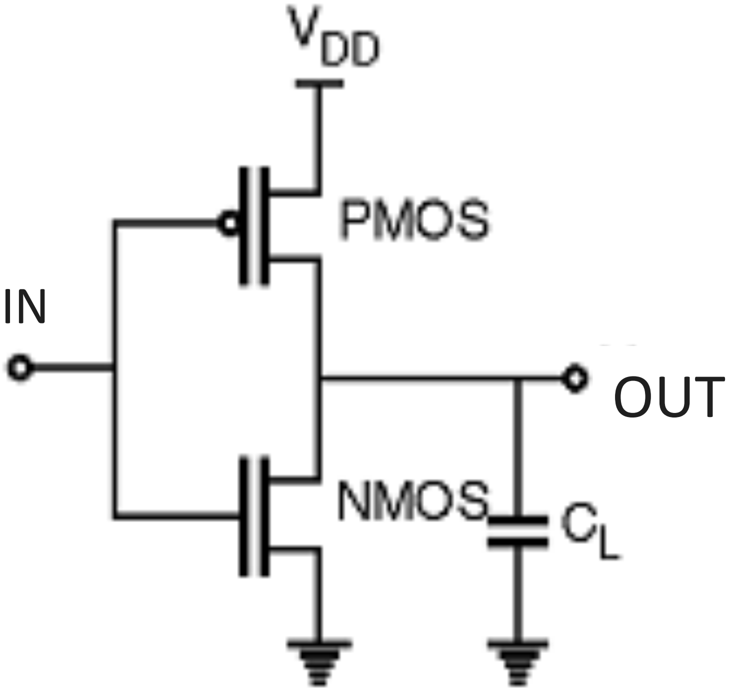
Other common SPICE analyses

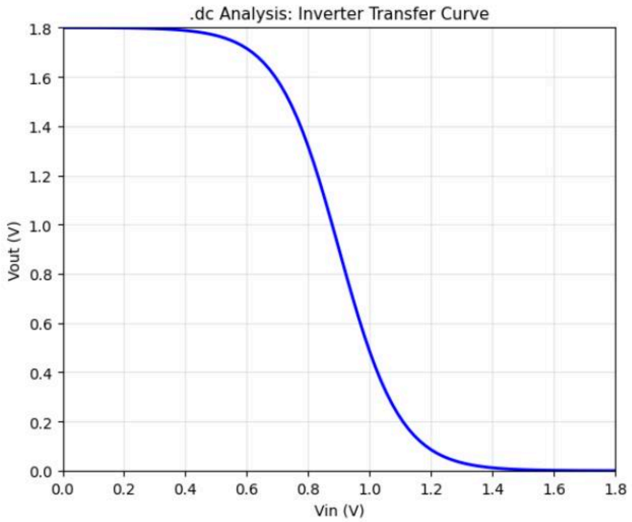
Analysis	Command	Purpose
DC	`.dc`	Sweep DC voltage/current, find operating point
AC	`.ac`	Frequency response (gain, phase)
Transient	`.tran`	Time-domain response
Operating Point	`.op`	DC node voltages and currents

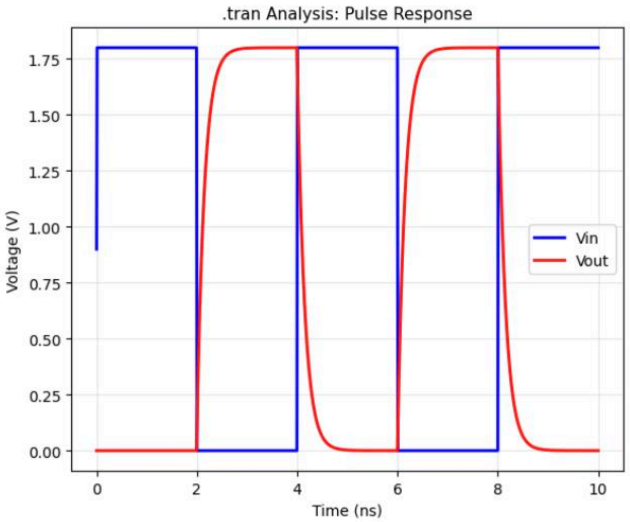
```
* CMOS Inverter Simulation
* For Fab Futures - Week 2

* Include the Sky130 device models
.lib "sky130A/libs.tech/ngspice/sky130.lib.spice" tt

* Power supply: 1.8V
Vdd vdd gnd 1.8

* Input: pulse from 0V to 1.8V
* PULSE(initial final delay rise fall width period)
Vin in gnd PULSE(0 1.8 1n 100p 100p 2n 4n)

* PMOS transistor (W=1u, L=150n)
* Format: Mname drain gate source body model W=... L=...
Xp out in vdd vdd sky130_fd_pr__pfet_01v8 W=1u L=150n

* NMOS transistor (W=0.5u, L=150n)
Xn out in gnd gnd sky130_fd_pr__nfet_01v8 W=0.5u L=150n

* Output load capacitor (typical gate load)
Cload out gnd 10f

* Simulation: transient analysis, 10ps step, 20ns duration
.tran 10p 20n

* Save node voltages for plotting
.save v(in) v(out)

* Control block for ngspice
.control
run
plot v(in) v(out)
meas tran tpd_hl TRIG v(in) VAL=0.9 RISE=1 TARG v(out) VAL=0.9 FALL=1
meas tran tpd_lh TRIG v(in) VAL=0.9 FALL=1 TARG v(out) VAL=0.9 RISE=1
.endc

.end
```

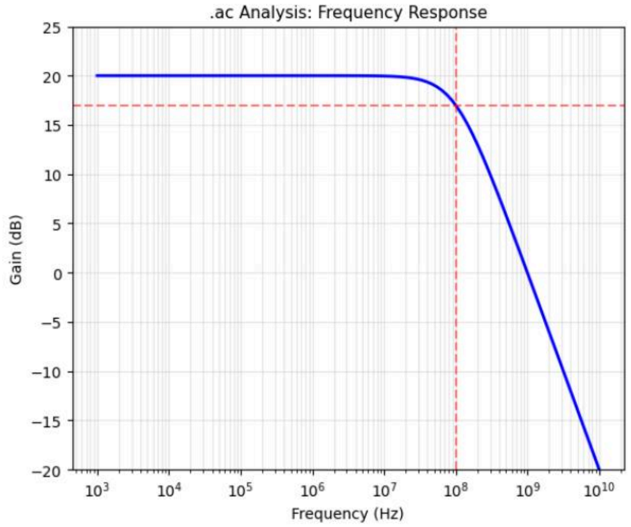
Example .dc, .tran, and .ac analysis

.tran ”curving” is an artifact of output capacitance!









Operating Point (.op)

```
Node Voltages:
 V(vdd)  = 1.800 V
V(in)   = 0.900 V
 V(out)  = 0.892 V

Device Currents:
I(Mp)   = -15.2 µA
I(Mn)   =  15.2 µA
```

A simple simulation (NOT gate)

Step 1: Write your netlist file. Save as inverter.spice.

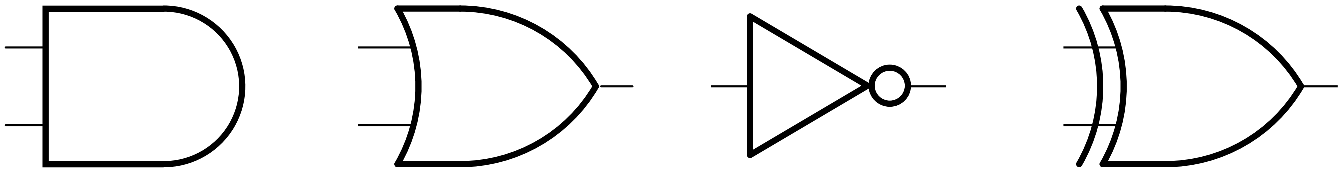
Other common SPICE analyses

Analysis	Command	Purpose
DC	`.dc`	Sweep DC voltage/current, find operating point
AC	`.ac`	Frequency response (gain, phase)
Transient	`.tran`	Time-domain response
Operating Point	`.op`	DC node voltages and currents

These .meas statements give you propagation delays

```
* CMOS Inverter Simulation
* For Fab Futures - Week 2

* Include the Sky130 device models
.lib "sky130A/libs.tech/ngspice/sky130.lib.spice" tt

* Power supply: 1.8V
Vdd vdd gnd 1.8

* Input: pulse from 0V to 1.8V
* PULSE(initial final delay rise fall width period)
Vin in gnd PULSE(0 1.8 1n 100p 100p 2n 4n)

* PMOS transistor (W=1u, L=150n)
* Format: Mname drain gate source body model W=... L=...
Xp out in vdd vdd sky130_fd_pr__pfet_01v8 W=1u L=150n

* NMOS transistor (W=0.5u, L=150n)
Xn out in gnd gnd sky130_fd_pr__nfet_01v8 W=0.5u L=150n

* Output load capacitor (typical gate load)
Cload out gnd 10f

* Simulation: transient analysis, 10ps step, 20ns duration
.tran 10p 20n

* Save node voltages for plotting
.save v(in) v(out)

* Control block for ngspice
.control
run
plot v(in) v(out)
meas tran tpd_hl TRIG v(in) VAL=0.9 RISE=1 TARG v(out) VAL=0.9 FALL=1
meas tran tpd_lh TRIG v(in) VAL=0.9 FALL=1 TARG v(out) VAL=0.9 RISE=1
.endc

.end
```

What are propagation delays?



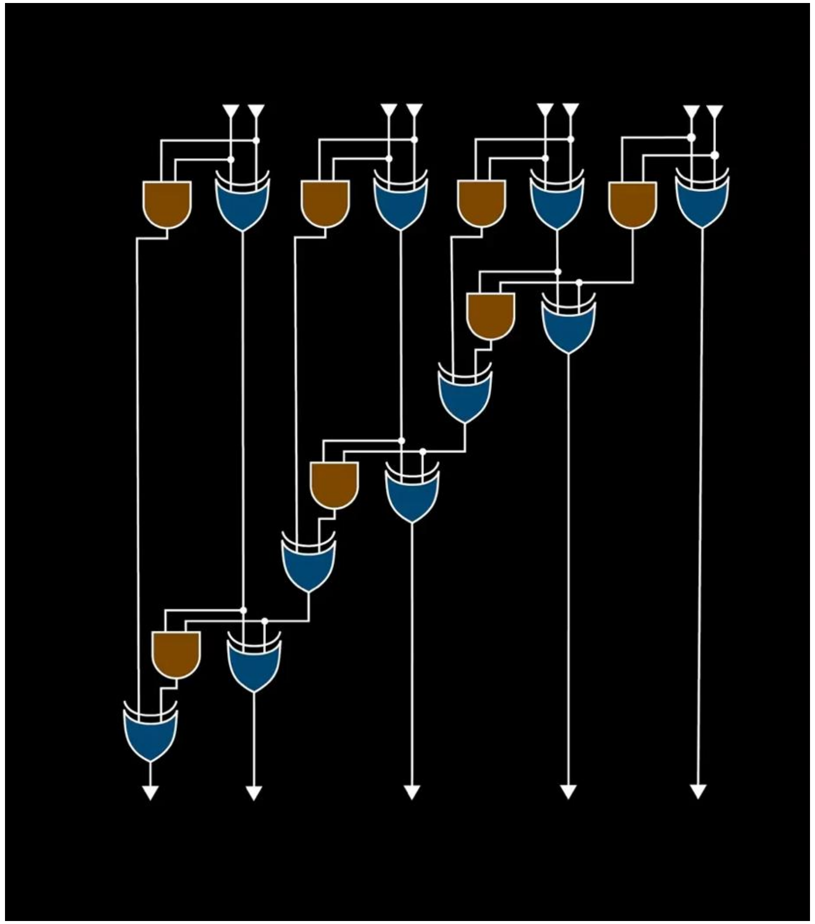
AND OR NOT XOR

^^It takes time for inputs to “pass” to the outputs of each gate

(Called “propagation delay”)

Chains of logic exhibit delays serially (e.g., total delay = n*(AND delay)+y*(XOR delay)

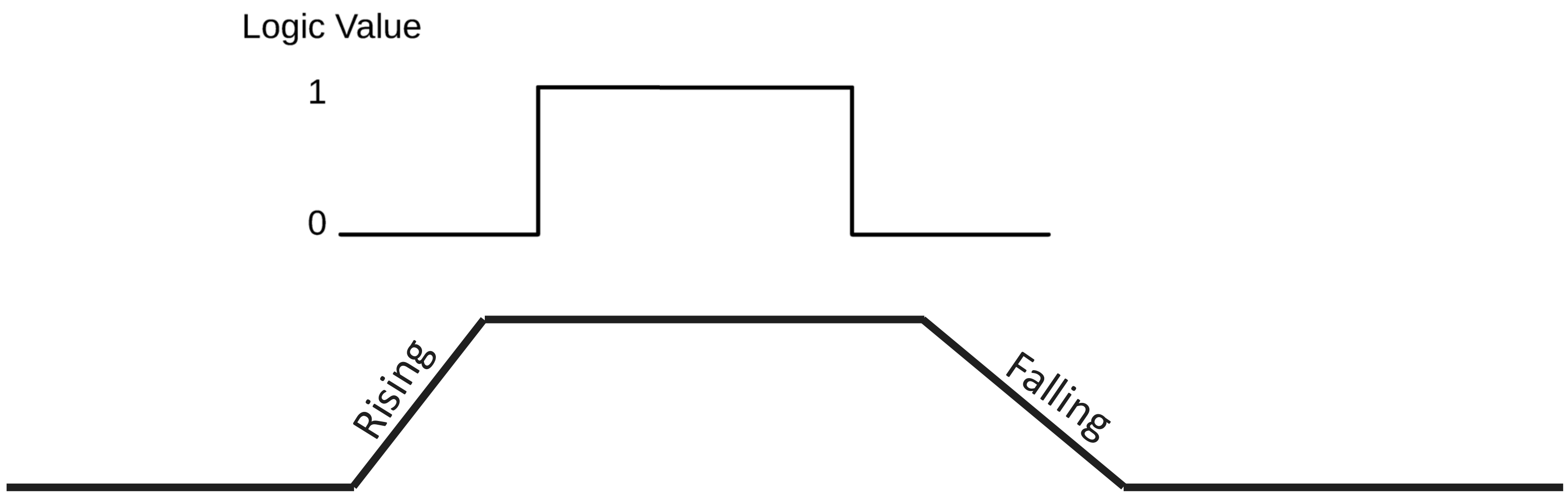
for n AND gates and y XOR gates



Long wires + slow logic gates make overall compute speeds slower

Zooming in to a signal

We also see that these transition edges are not sharp—they have some slanting



The amount of slant varies on the rising and falling edges, dependent on the transistor-level implementation (PMOS controls the rising edges, NMOS controls the falling edges)

```
meas tran tpd_hl TRIG v(in) VAL=0.9 RISE=1 TARG v(out) VAL=0.9 FALL=1
meas tran tpd_lh TRIG v(in) VAL=0.9 FALL=1 TARG v(out) VAL=0.9 RISE=1
```

^^these measure the 10%-90% difference in timing on each edge

A simple simulation (NOT gate)

Step 1: Write your netlist file. Save as inverter.spice.

Step 2: Run the simulation.

```
# Run ngspice in batch mode
ngspice -b inverter.spice
```

(All at once)

```
# Or run interactively
ngspice inverter.spice
```

(Run and keep ngspice waiting for new commands e.g., plot more signals)

A simple simulation (NOT gate)

Step 1: Write your netlist file. Save as inverter.spice.

Step 2: Run the simulation.

Successful simulation output looks like:

```
Circuit: * cmos inverter simulation

Doing analysis at TEMP = 27.000000 and target temp = 27.000000

Initial Transient Solution
--------------------------
Node                                       Voltage
----                                       -------
vdd                                            1.8
in                                               0
out                                        1.79998

...

tpd_hl                    =  2.34567e-11  targ=  3.234e-09   trig=  3.000e-09
tpd_lh                    =  3.12345e-11  targ=  5.312e-09   trig=  5.000e-09
```

Step 3: Understand the output.

Help! My simulation didn't run! Common error messages & fixes

Error	Meaning	Fix
`model xxx not found`	Missing PDK library	Check `.lib` path
`singular matrix`	Floating node or short	Check connections
`timestep too small`	Convergence failure	Try `.option reltol=1e-3`
`node xxx is floating`	Unconnected node	Add connection or resistor
`out of region`	Transistor bias issue	Check power supplies

Interactive commands

Things you can do when ngspice is in interactive mode (e.g., you'll see this:

```
ngspice 1 ->
```

)

```
ngspice 1 -> source inverter.spice    # Load the netlist
ngspice 1 -> run                       # Run simulation
ngspice 1 -> plot v(in) v(out)        # Plot waveforms
ngspice 1 -> print v(out)             # Print values
ngspice 1 -> wrdata output.csv v(out) # Export data
ngspice 1 -> quit                      # Exit
```

Saving your results

Export to CSV for external plotting:

```
.control
run
wrdata results.csv v(in) v(out)
.endc
```

Then in Python:

```python
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt('results.csv')
time = data[:, 0]
v_out = data[:, 2]  # Column 0=time, 1=v(in), 2=v(out)
plt.plot(time * 1e9, v_out)
plt.xlabel('Time (ns)')
plt.ylabel('Vout (V)')
```

Device library (comes with every Process Design Kit)

SkyWater 130nm examples:

Device	SPICE Model Name	Description
1.8V NFET	`sky130_fd_pr__nfet_01v8`	Standard NMOS
1.8V PFET	`sky130_fd_pr__pfet_01v8`	Standard PMOS
3.3V NFET	`sky130_fd_pr__nfet_03v3_nvt`	I/O transistor
Poly resistor	`sky130_fd_pr__res_high_po`	High-R poly
MIM capacitor	`sky130_fd_pr__cap_mim_m3_1`	Metal capacitor

<<Resistor that is close to transistors in layout

Manufacturing of these transistors introduces variation; affects timing

To account for this variable (and somewhat random) changes in timing (e.g., low-to-high and high-to-low), we simulate over various process “corners” (worst cases)

Corner	NMOS	PMOS	Meaning
TT	Typical	Typical	Nominal process
FF	Fast	Fast	Best-case speed
SS	Slow	Slow	Worst-case speed
SF	Slow	Fast	Skewed
FS	Fast	Slow	Skewed

To run your own corner simulations

Change your netlist:

```
* Multi-corner simulation
* Run the same circuit at TT, FF, and SS corners

.param corner_select = 0

.if (corner_select == 0)
.lib "sky130A/libs.tech/ngspice/sky130.lib.spice" tt
.elseif (corner_select == 1)
.lib "sky130A/libs.tech/ngspice/sky130.lib.spice" ff
.elseif (corner_select == 2)
.lib "sky130A/libs.tech/ngspice/sky130.lib.spice" ss
.endif

* ... rest of your circuit ...
```

Or, run multiple simulations (e.g., loop the simulation, changing the parameters as you iterate)—see markdown for this code

Expected behavior

Expected corner behavior:

Corner	Delay	Power	Notes
TT	Nominal	Nominal	Design target
FF	Fastest	Highest	Best-case timing, worst-case power
SS	Slowest	Lowest	Worst-case timing
SF/FS	Mixed	Mixed	Check for skew-related issues

Rule: Your design must meet timing at **SS corner** (slow-slow) and not exceed power budget at **FF corner** (fast-fast).

In digital design, you don’t need all these details

You can abstract away the transistor-level implementation into a behavioral one:

Verilog-A Example

```
// Ideal voltage-controlled switch
module vcswitch(p, n, ctrl);
    inout p, n;
    input ctrl;
    electrical p, n, ctrl;

    parameter real ron = 1;       // On resistance
    parameter real roff = 1e9;    // Off resistance
    parameter real vth = 0.5;     // Threshold

    analog begin
        if (V(ctrl) > vth)
            I(p, n) <+ V(p, n) / ron;
        else
            I(p, n) <+ V(p, n) / roff;
    end
endmodule
```

This part says that the transistor is not just a wire “short” or ”open” but it has resistance (low resistance when “ON” and high resistance when “OFF”)

This part says that if the transistor is ON, use Ron for the calculation; if it is OFF, use Roff

The ”<+” operator is a “contribution operator”, which assigns an expression to an electrical parameter (in this case, it assigns V/R to current, a common relationship in circuits)

Homework for next time

1. Reuse and.sp netlist, which has a 2-input AND gate to make a 2-input NAND gate (e.g., remove the output inverter) and change the models to refer to the PDK models

2. Simulate it in SPICE, verify truth table (it will look something like the right table), and measure propagation delays (low-to-high and high-to-low)

A	B	OUT
0	0	1
0	1	1
1	0	1
1	1	0

3. Write an initial analog block that you can use in your chip project (e.g., an adder, counter, etc.)

Here's a good starting point: https://analoghub.ie/category/verilogModels/article/counter

Additional Resources

- https://www.tinkercad.com – TinkerCAD, a free online app for breadboard simulation…without a breadboard!
- https://youtu.be/y5I76fLlupI?si=2fpVd0IcPH9cpvfJ – TinkerCAD breadboarding tutorial
- https://youtu.be/eLSoJn__gAc?si=4noB48_DYluNywS2 – A more in-depth circuits introduction
- http://instrumentacion.qi.fcen.uba.ar/libro/Scherz.pdf - "Practical Electronics for Inventors" by Paul Scherz
- https://www.analog.com/en/resources/design-tools-and-calculators/ltspice-simulator.html - LTSPICE, a free simulator with schematic capture built-in

Questions?