

# Custom async circuit design from production rules to netlist

Benjamin Hill

[benjamin.hill@intel.com](mailto:benjamin.hill@intel.com)

ASYNC Summer School 2024

# Legal Information

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Your costs and results may vary.

Results have been estimated or simulated

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

# Implementing custom circuit designs

## **Our task:**

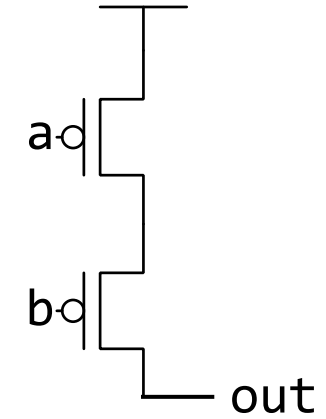
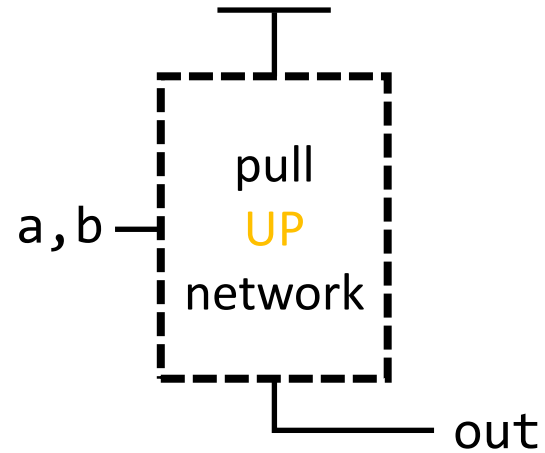
Given a gate-level system expressed as production rule set (PRS), generate netlist (SPICE) and physical implementation (layout)

Do as little full custom design as possible!

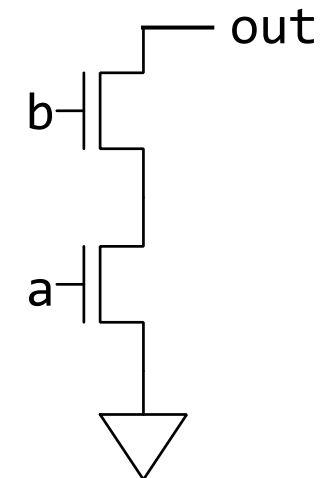
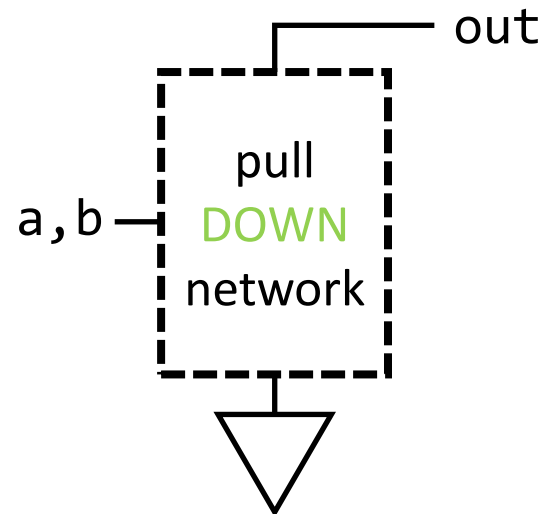
Refer to 2022 week 3 sessions describing Yale's open-source automated physical implementation flow

# Production rule basics

$\sim a \ \& \ \sim b \ \rightarrow \text{out}+$

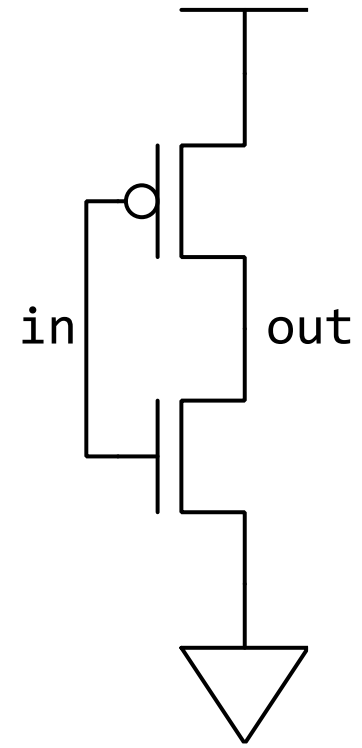
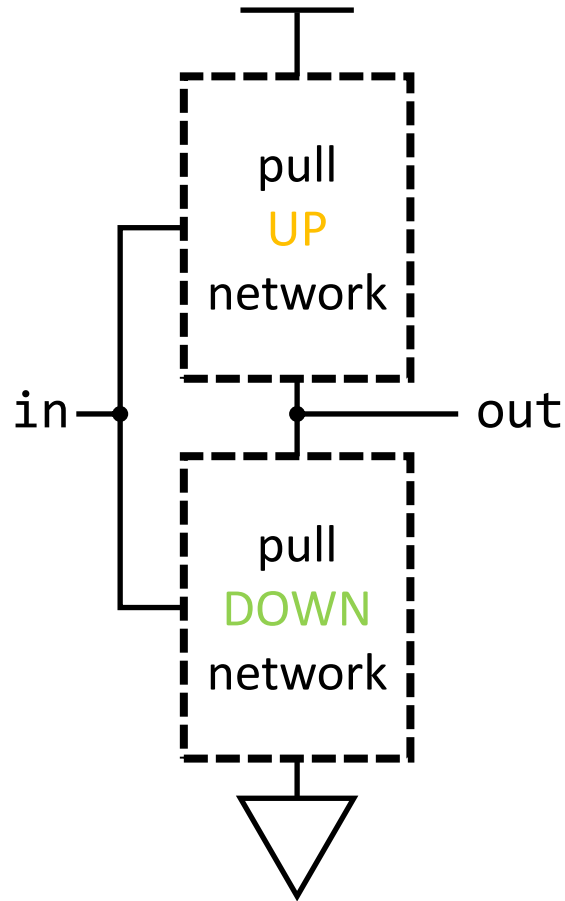


$a \ \& \ b \ \rightarrow \text{out}-$

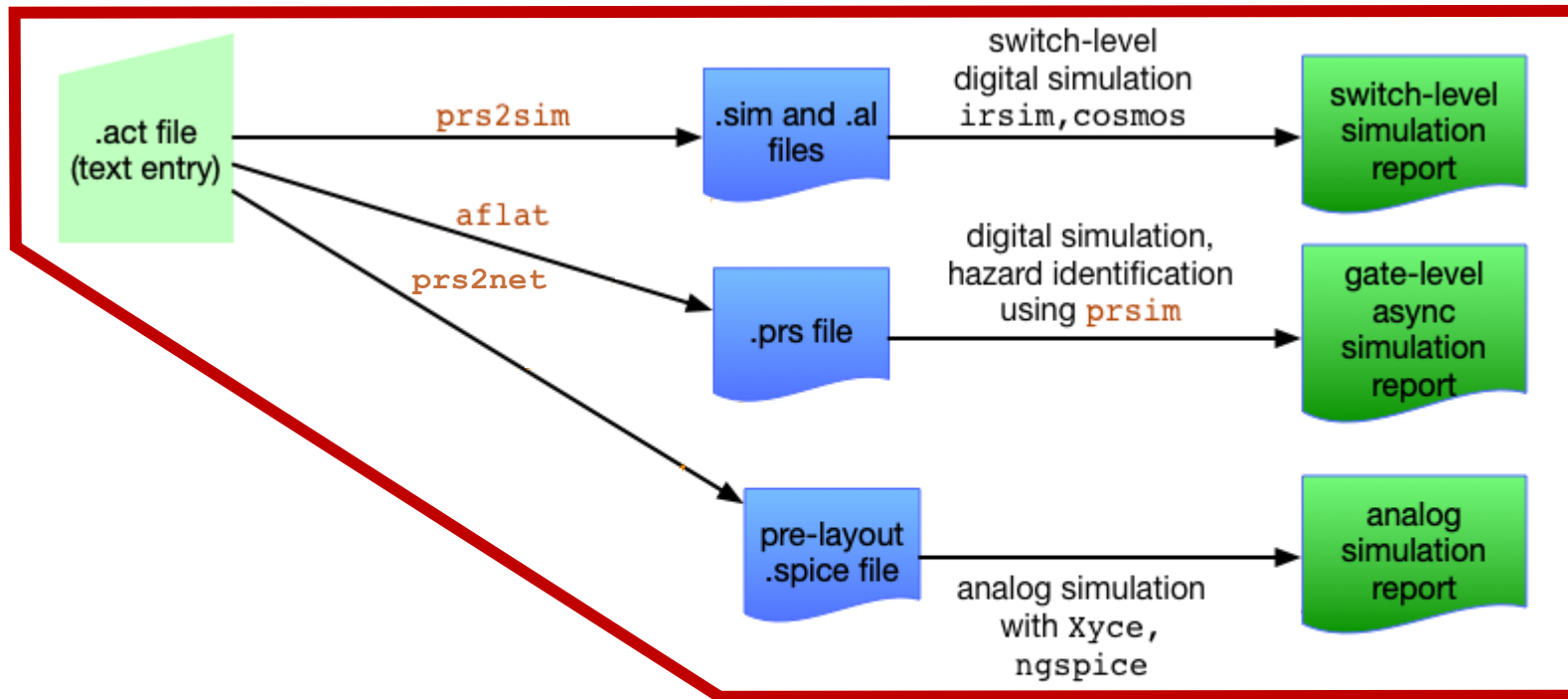


# Production rule sets form gates

$\sim in \rightarrow out+$   
 $in \rightarrow out-$



# Custom design flow



toolname: *existing open-source tools*  
toolname: *ACT tools*

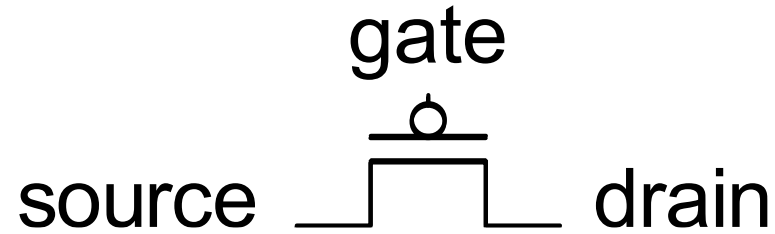
# Simulation options

	<b>Gate level</b>	<b>Switch level</b>	<b>Analog</b>
<b>simulator</b>	prsim, actsim	irsim	Xyce
<b>input</b>	ACT PRS	.sim	SPICE
<b>to generate</b>	write directly or use e.g. chp2prs	prs2sim	prs2net
<b>model</b>	unit delay	RC delay	full analog
<b>fidelity</b>	lowest	medium	highest
<b>speed</b>	fastest	fast	slow

CMOS transistors



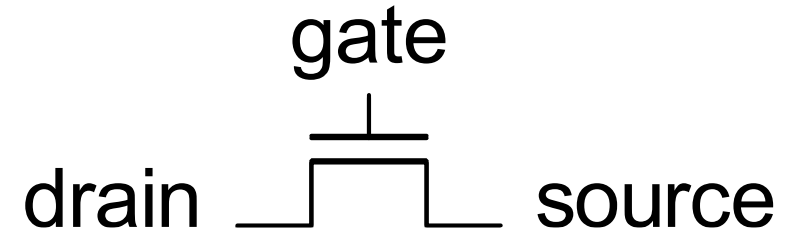
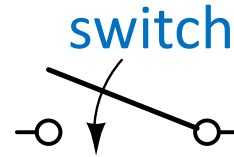
# CMOS transistors



PMOS

conducts current  
between source and drain  
(through P-type channel)  
when gate is **LOW**

voltage-controlled



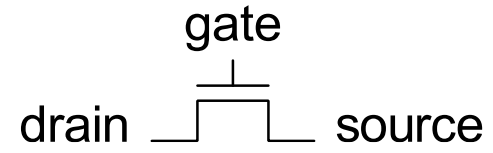
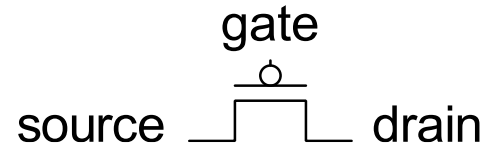
NMOS

conducts current  
between source and drain  
(through N-type channel)  
when gate is **HIGH**

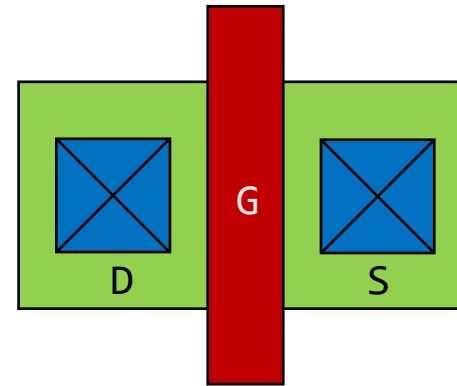
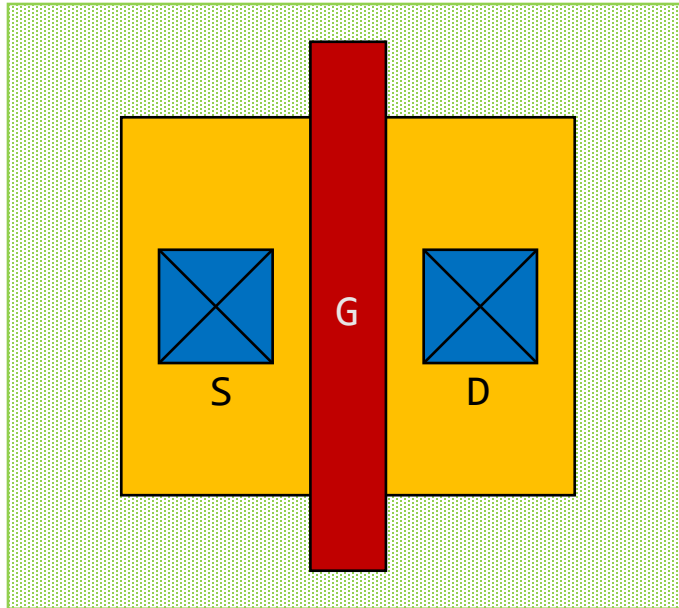
# PMOS

# NMOS

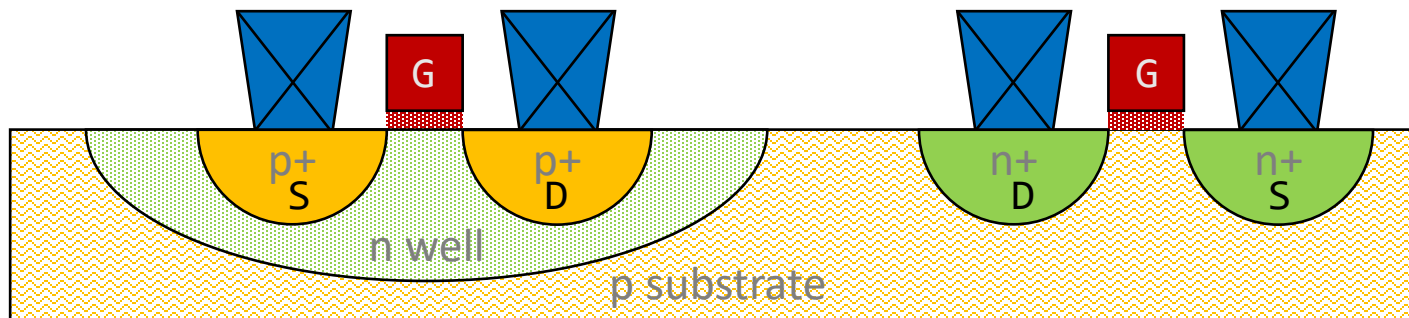
schematic  
symbol



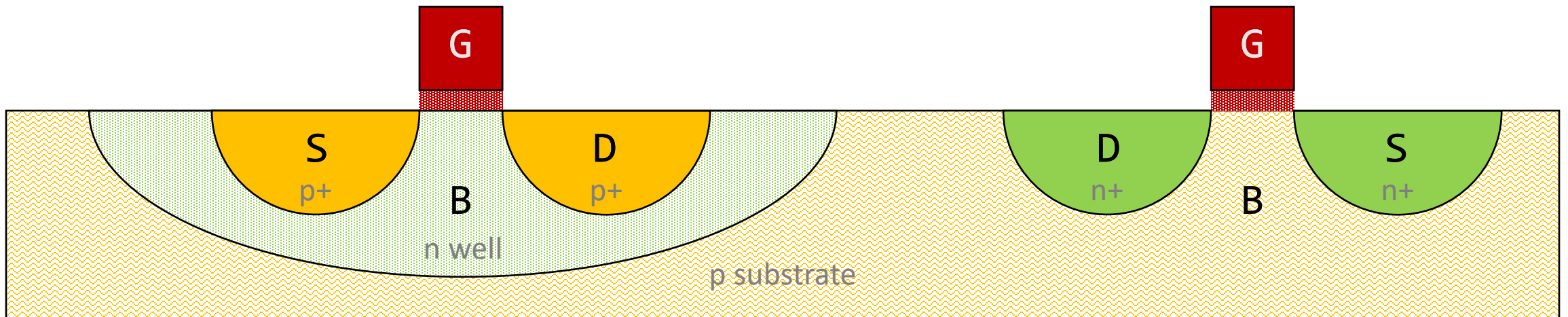
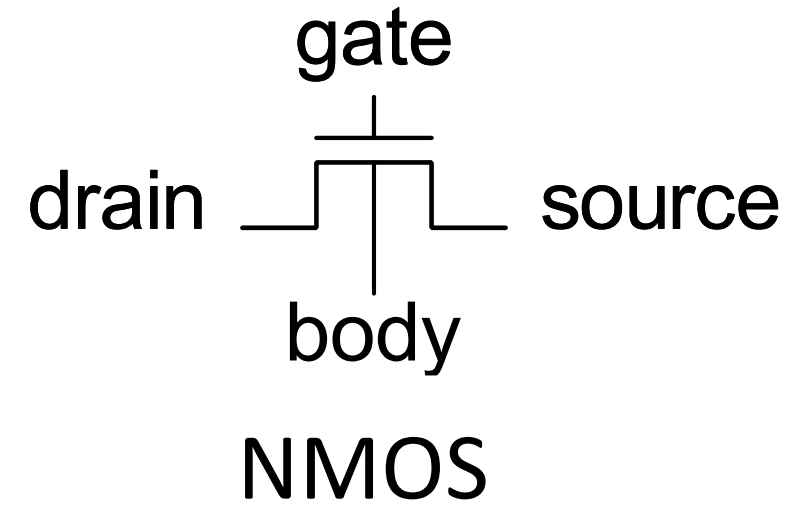
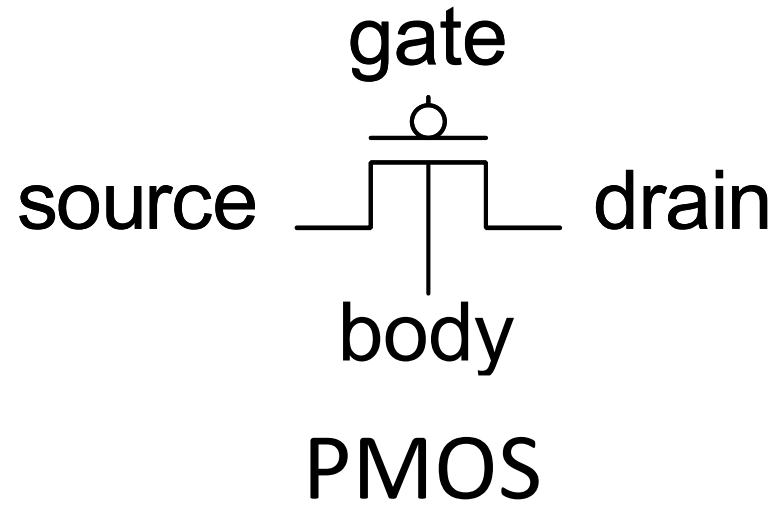
layout  
(top down view)



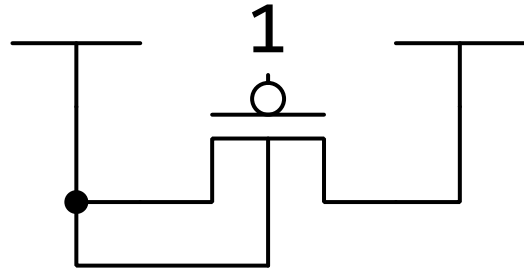
cross section



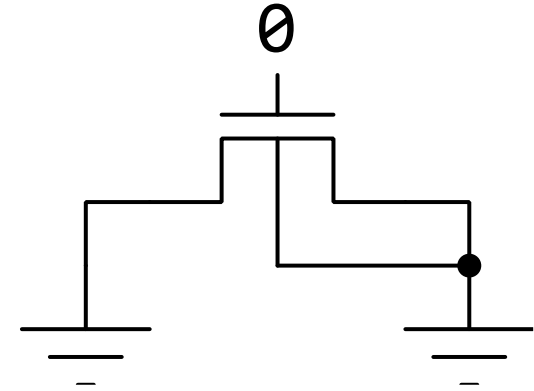
# CMOS transistors



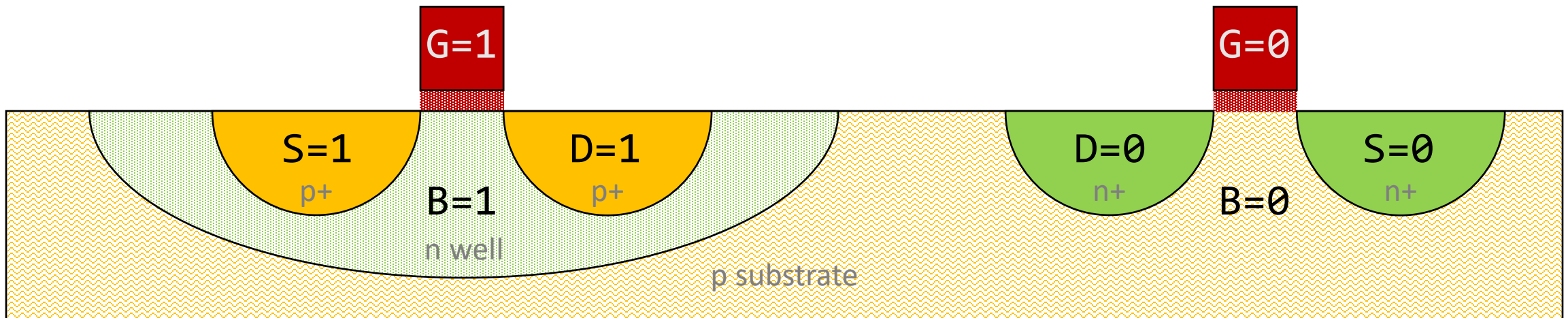
# Transistor operation: cutoff



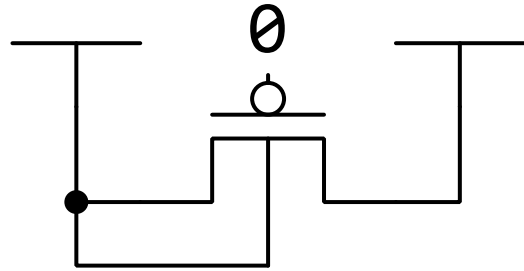
PMOS



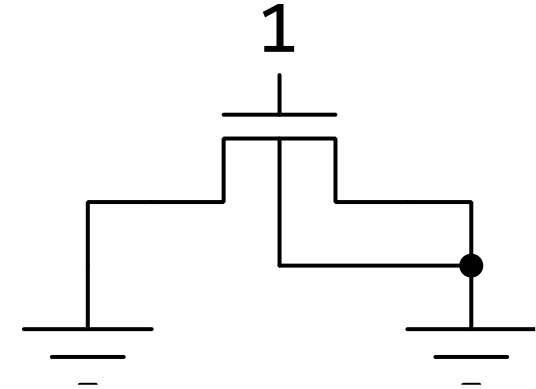
NMOS



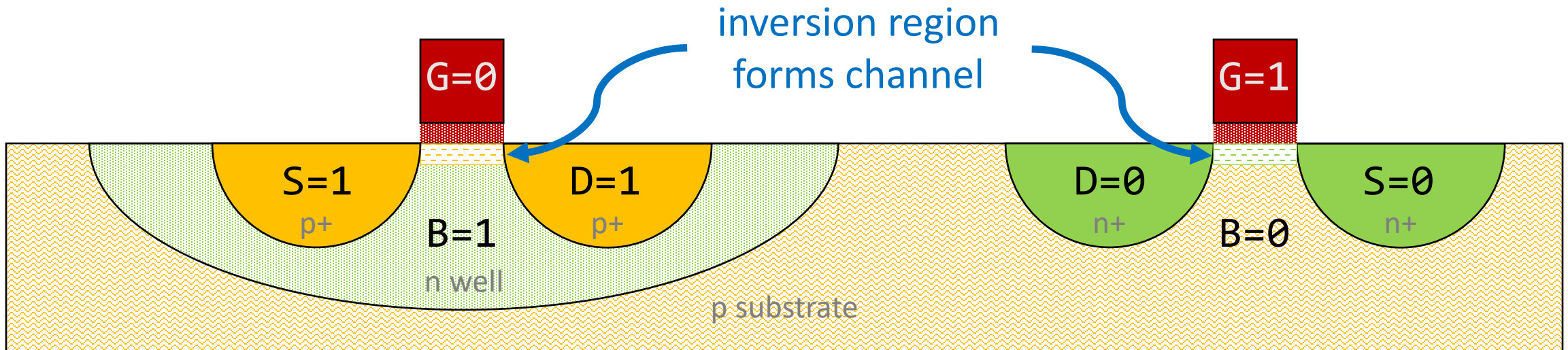
# Transistor operation: channel formation



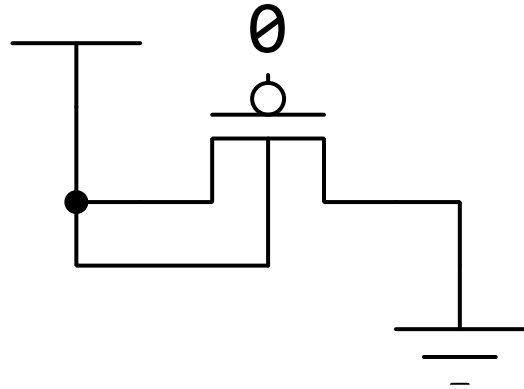
PMOS



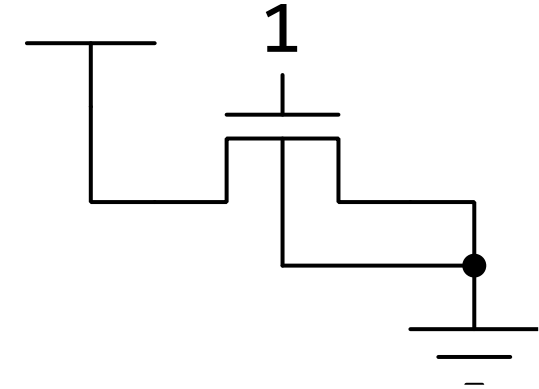
NMOS



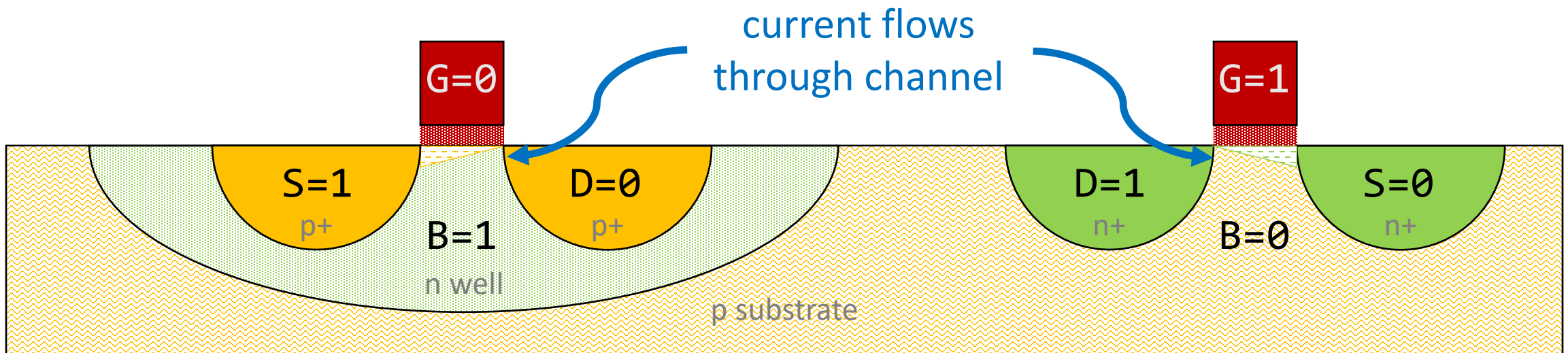
# Transistor operation: saturation



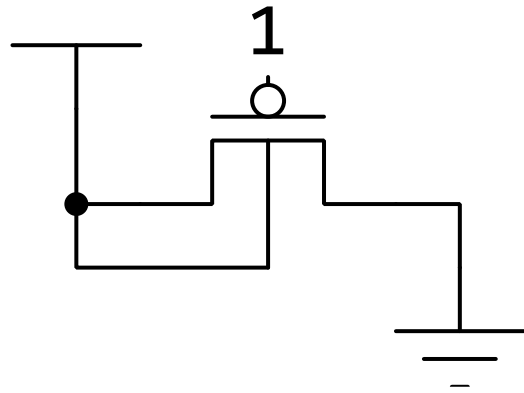
PMOS



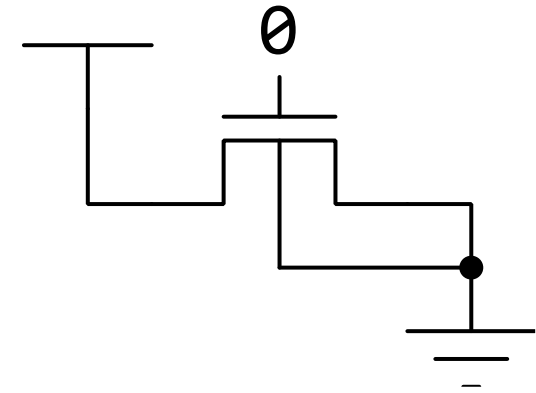
NMOS



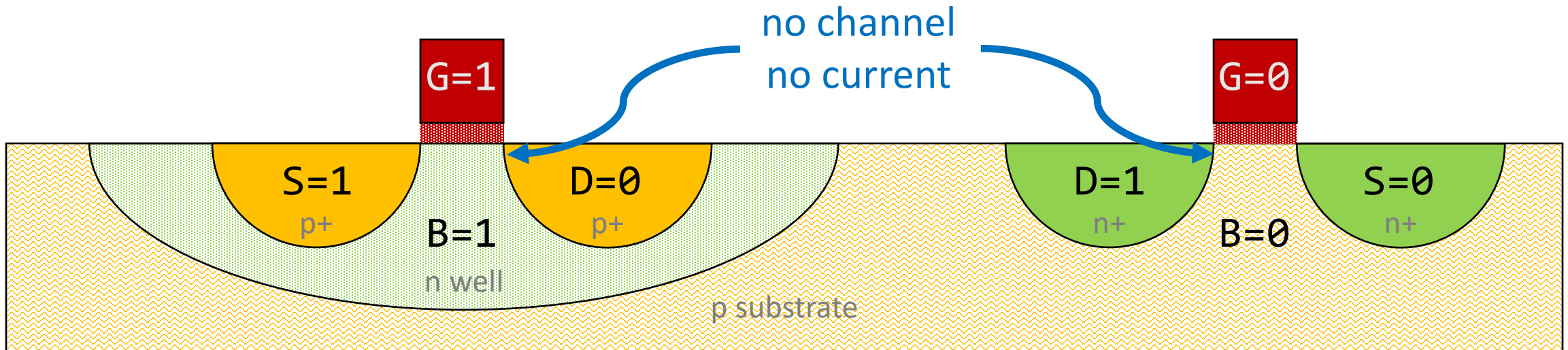
# Transistor operation: cutoff



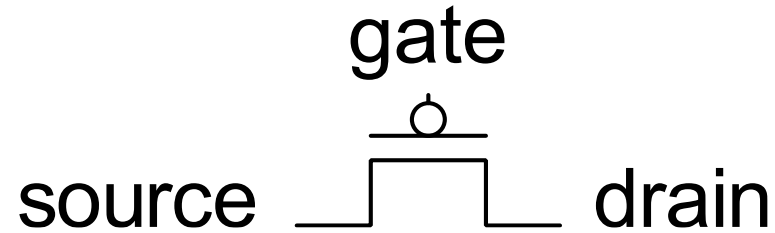
PMOS



NMOS



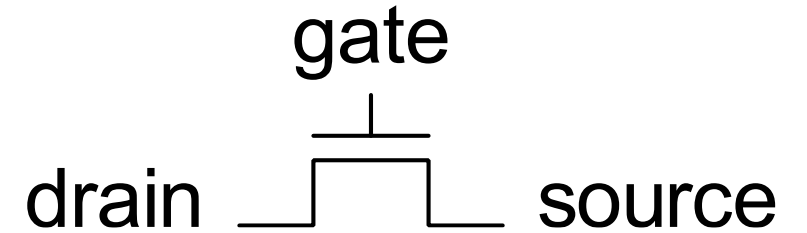
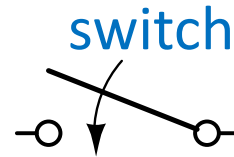
# CMOS transistors



PMOS

conducts current  
between source and drain  
(through P-type channel)  
when gate is **LOW**

voltage-controlled



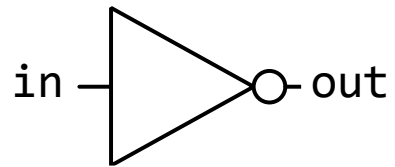
NMOS

conducts current  
between source and drain  
(through N-type channel)  
when gate is **HIGH**

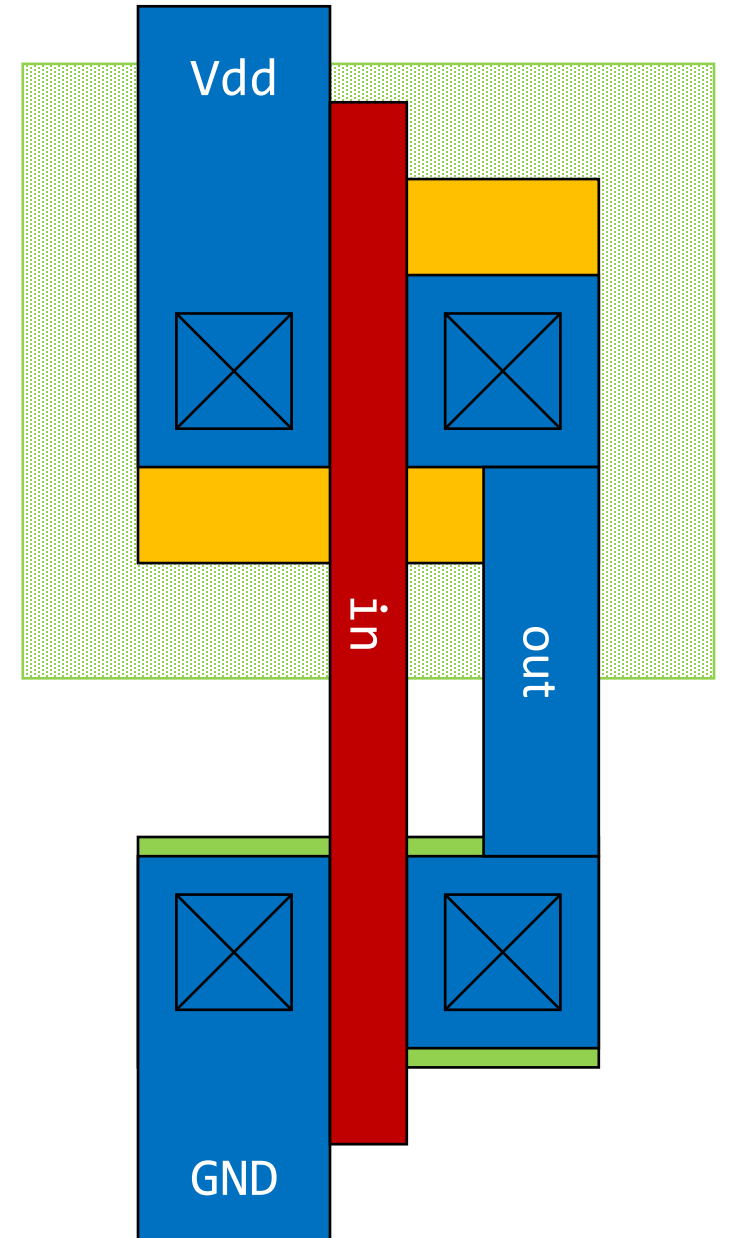
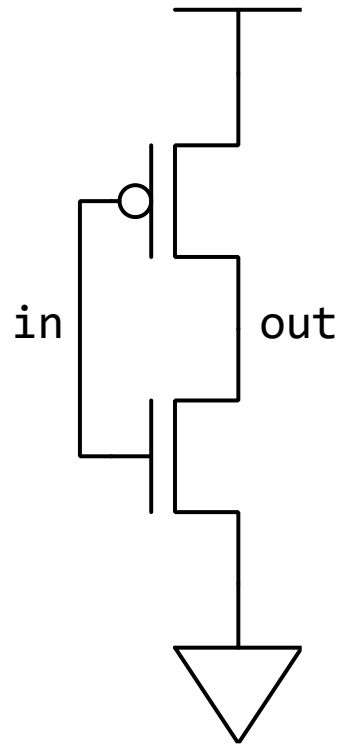


Layout and sizing

# Layout example: inverter



in	out
0	1
1	0

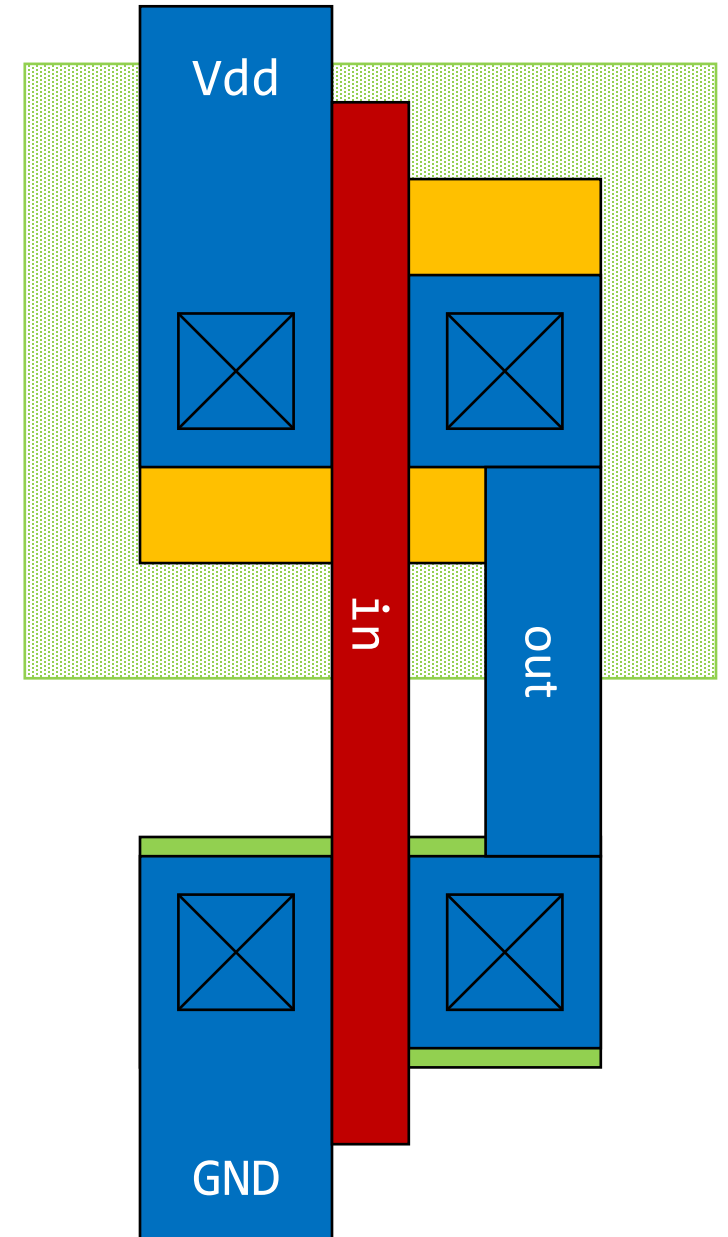


# Design rules

set of geometric restrictions intended to yield high probability of correct fabrication, operation, and lifetime

Table 3 Table 2 - Minimum CDs in Design or on Wafer, required by Technology (Core or Periphery)

Layer Name	Feature Size	Space Size	Feature Name	Space Name
Field Oxide	0.14	0.27	FOMCD	FOMCDSP
Deep N-Well	3	6.3	DNMCD	DNMCDSP
P-Well Block Mask	0.84	1.27	PWBMCD	PWBMCDSP
P-Well Drain Extended	0.84	1.27	PWDEMCD	PWDEMCDSP
N-Well	0.84	1.27	NWMCD	NWMCDSP
Metal 1	0.14	0.14	MM1CD	MM1CDSP
Metal 1 - Cu	0.14	0.14	MM1_CuCD	MM1_CuCDSP
Via	0.15	0.17	VIMCD	VIMCDSP
Via - Cu	0.18	0.13	VIM_CuCD	VIM_CuCDSP
Capacitor MIM	2	0.84	CAPMCD	CAPMCDSP
Metal 2	0.14	0.14	MM2CD	MM2CDSP
Metal 2 - Cu	0.14	0.14	MM2_CuCD	MM2_CuCDSP
Via 2-TNV	0.28	0.28	VIM2CD	VIM2CDSP
Via 2-S8TM	0.8	0.8	VIM2CD	VIM2CDSP



# Transistor sizing

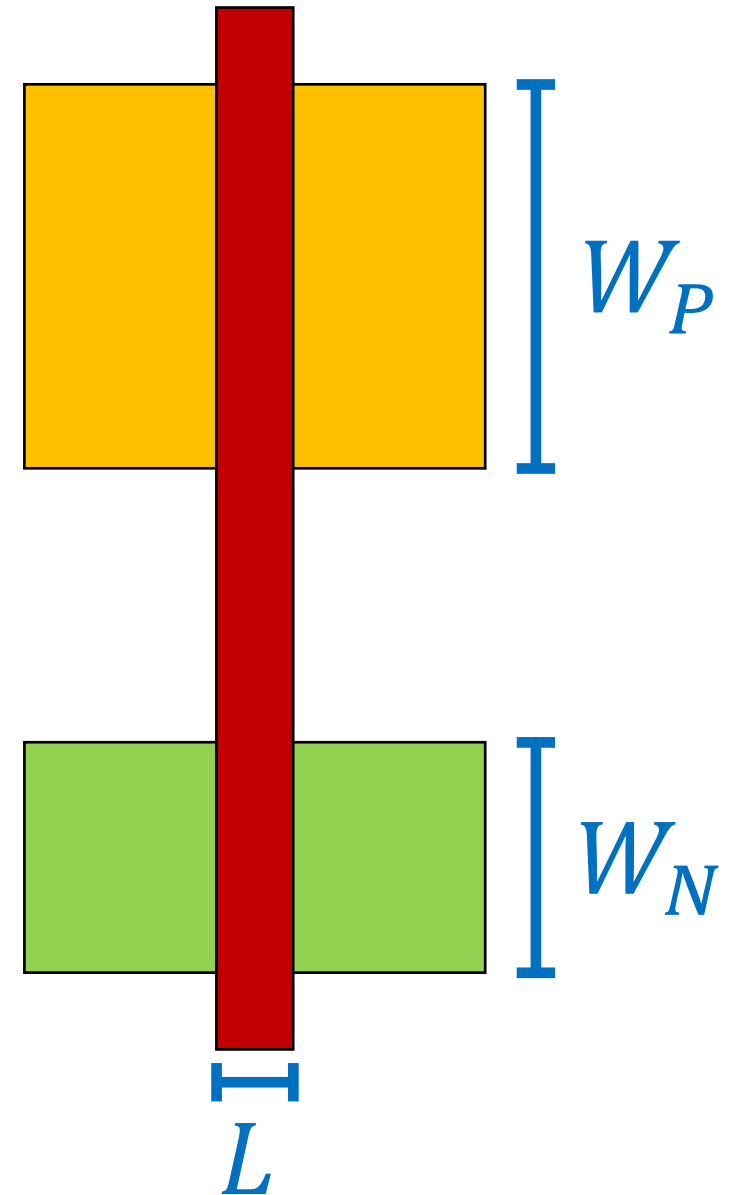
## Skywater 130 simplified design rules

$\lambda$	75 nm*
$L$	$2\lambda$
$W_N$	$6\lambda$
$W_P$	$10\lambda$

```
# sky1301 prs2net.conf
...
int std_p_width 10
int std_p_length 2

int std_n_width 6
int std_n_length 2
...
real p_n_ratio 1.512
real weak_to_strong_ratio 0.1

real lambda 7.5e-8
```

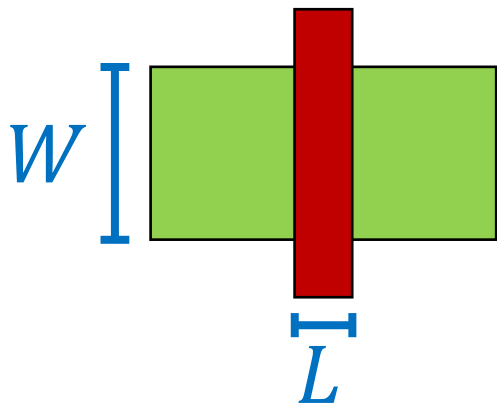


\* minimum feature size for Skywater 130  
is 150nm transistor gate length

# Transistor performance scaling intuition

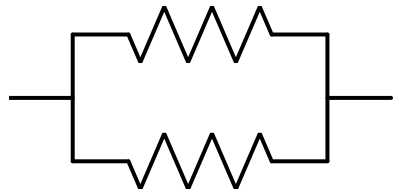
Transistor device with  
width and length parameters

What happens as we vary them?

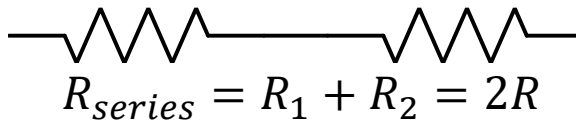


# Analogy: resistors

Decreased resistance,  
increased current



$$R_{parallel} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}} = \frac{R}{2}$$



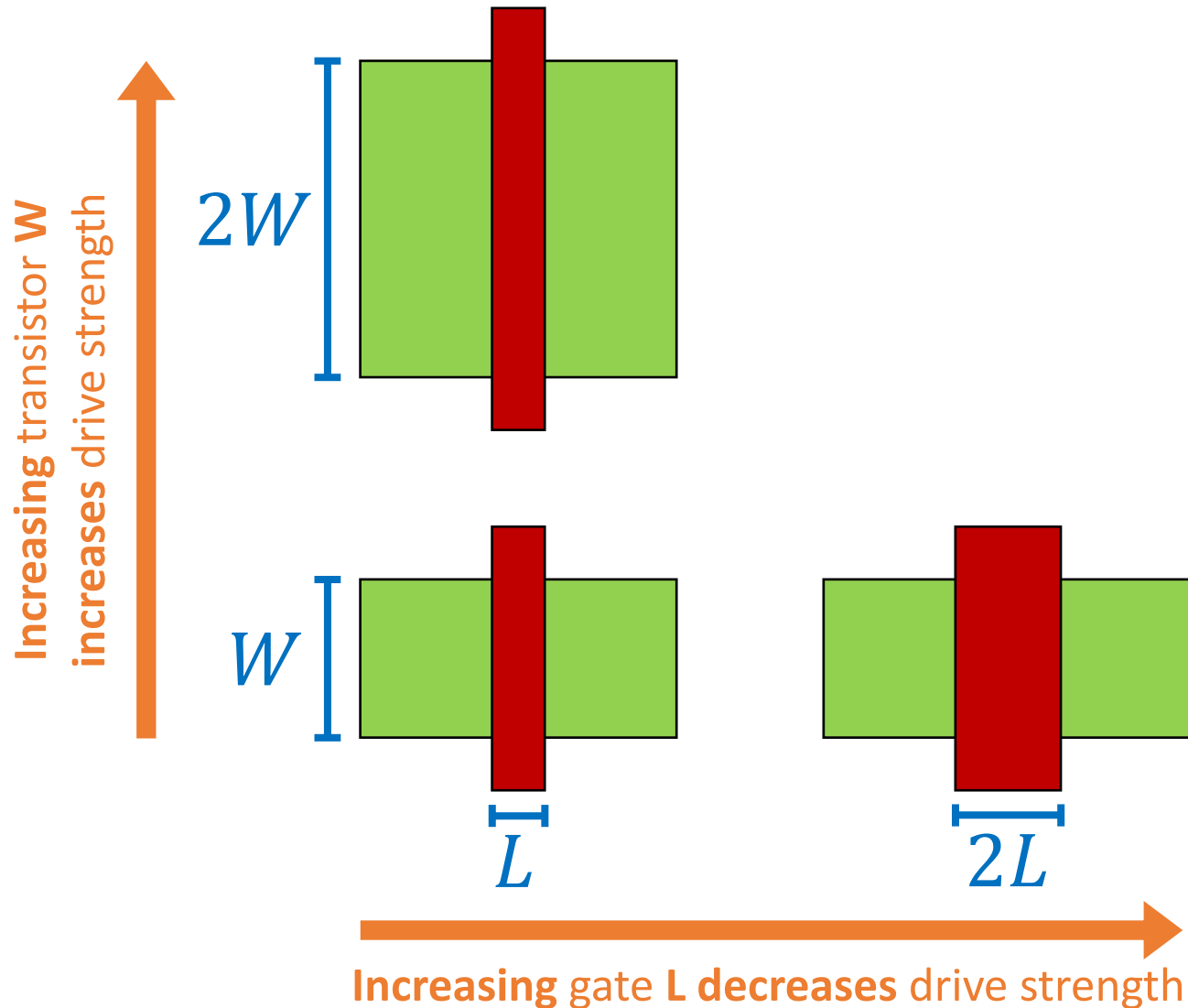
$$R_{series} = R_1 + R_2 = 2R$$

current through resistor  
(aka “drive strength”)  
is inversely proportional to  
effective resistance

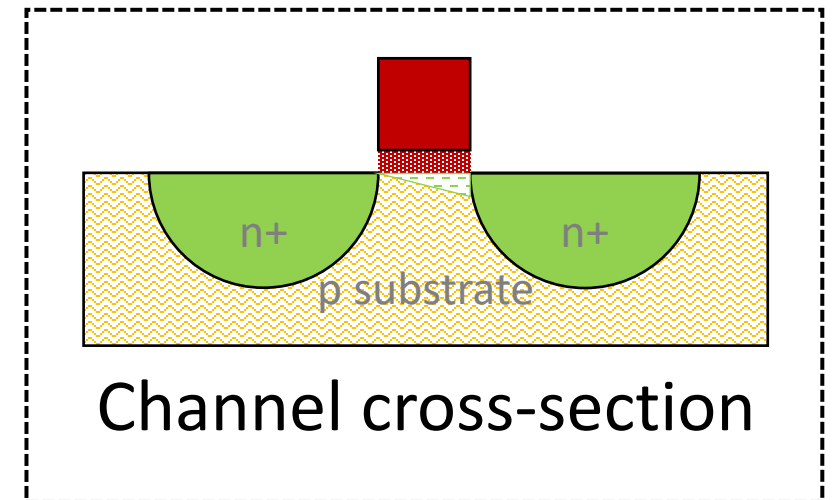
$$I = \frac{V}{R}$$

Increased resistance, decreased current

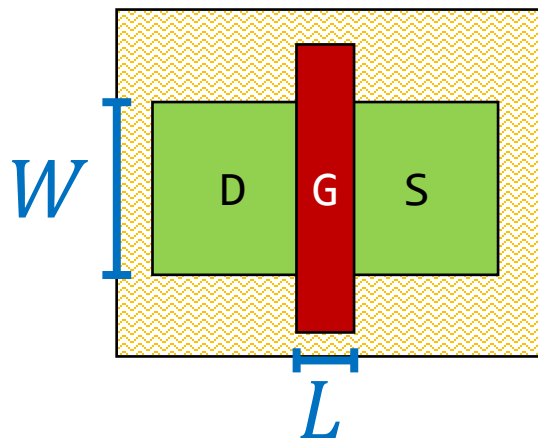
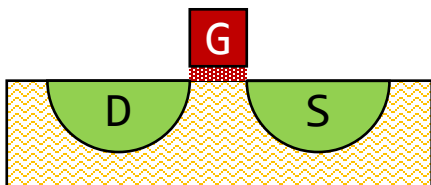
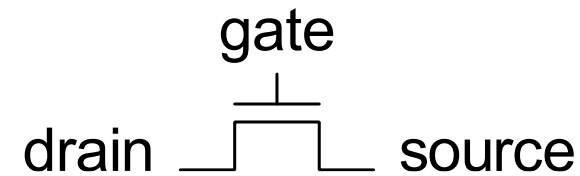
# Transistor performance scaling intuition



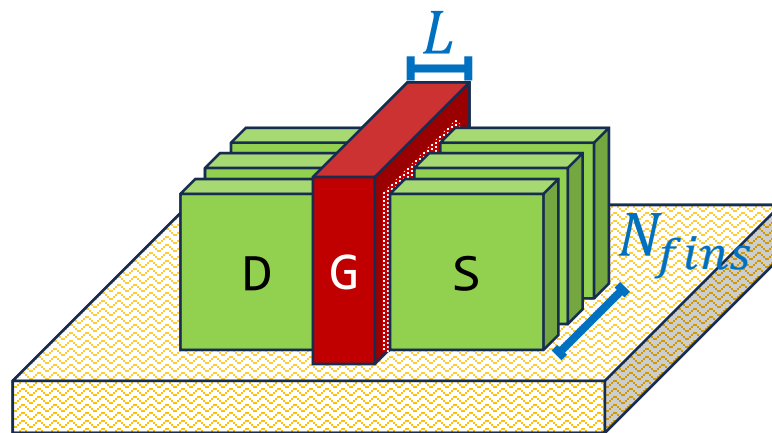
**Key performance metric:**  
current through transistor  
effective resistance  
“drive strength”



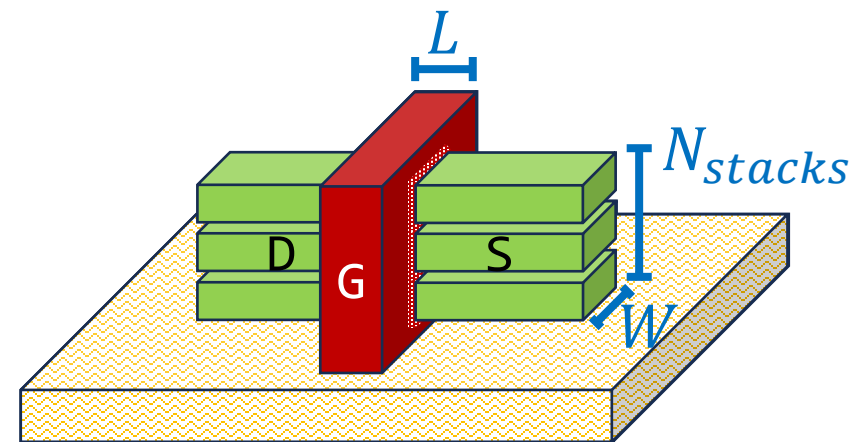
# Modern transistor architectures



Planar



FinFET



Gate-All-Around



# Digital designer summary

Use NMOS in pull-down network, PMOS in pull-up

⇒ single stage logic is always inverting

Transistor drive strength  $\propto \frac{\textit{Width}}{\textit{Length}}$

⇒ use minimum gate length for digital logic (usually)

State-holding gates

# Combinational vs state-holding gates

## Combinational

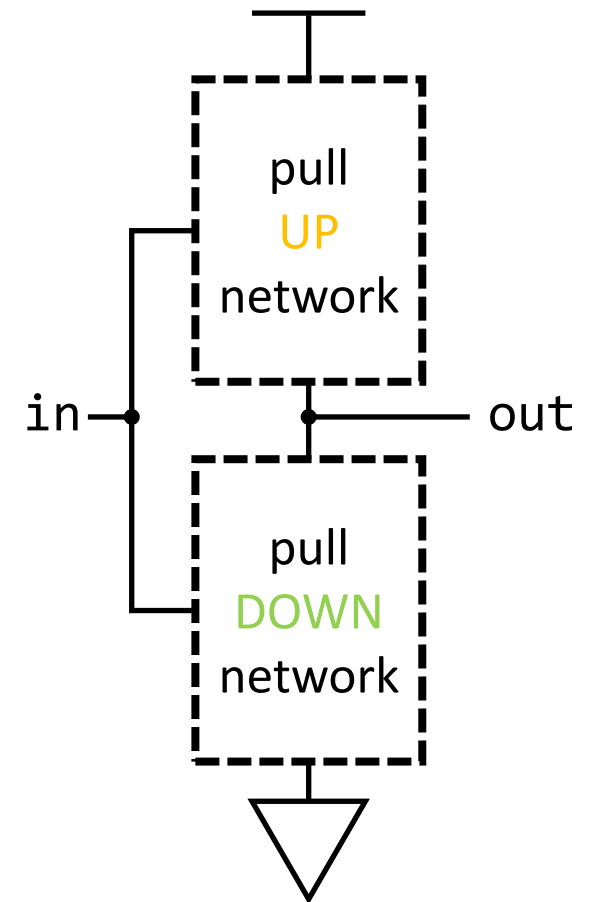
- Either **UP** or **DOWN** (but not both) is always conducting

## State-holding

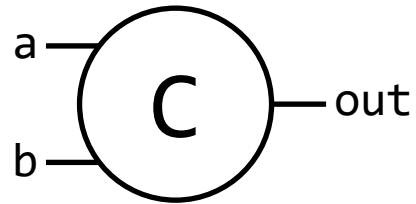
- At times neither **UP** nor **DOWN** is conducting
- out is undriven and maintains its previous value

## Interfering

- Both **UP** and **DOWN** conducting simultaneously
- Causes short-circuit/crowbar current through gate, should not be more than transient

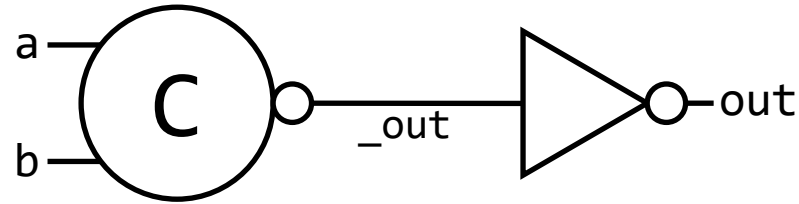


# Example state-holding gate: Muller C-element



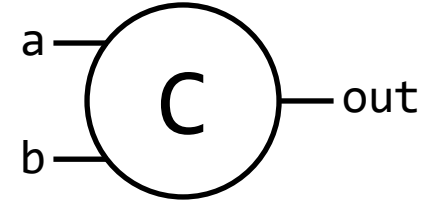
$a \ \& \ b \ \rightarrow \text{out}+$   
 $\sim a \ \& \ \sim b \ \rightarrow \text{out}-$

not CMOS implementable!



$a \ \& \ b \ \rightarrow \text{\_out}-$   
 $\sim a \ \& \ \sim b \ \rightarrow \text{\_out}+$   
 $\text{\_out} \ \rightarrow \text{out}-$   
 $\sim \text{\_out} \ \rightarrow \text{out}+$

inverting C-element plus inverter

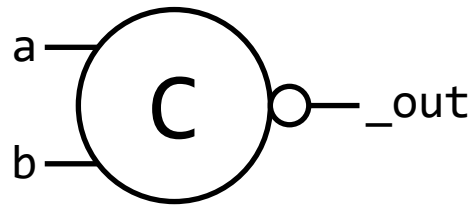


$a \ \& \ b \ \# \rightarrow \text{\_out}-$   
 $\text{\_out} \ \Rightarrow \text{out}-$

shorthand syntax

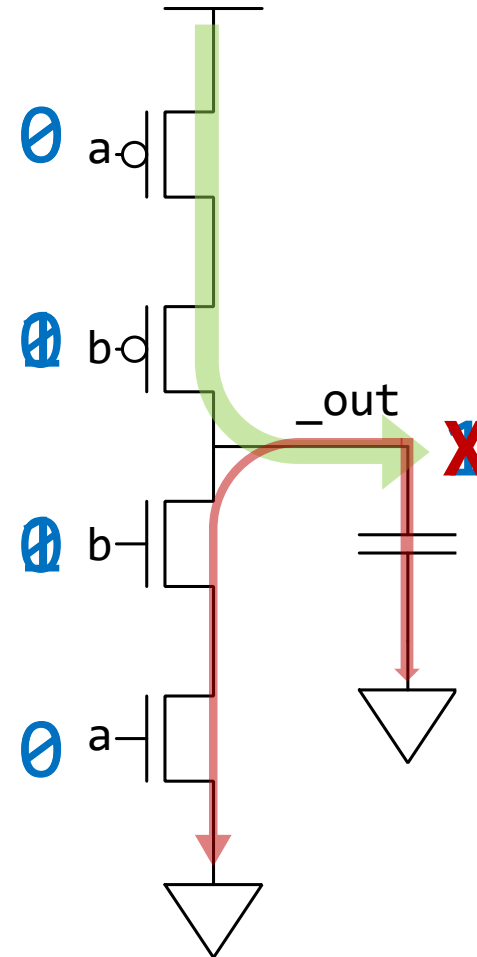
a	b	out
0	0	0
0	1	hold previous state
1	0	hold previous state
1	1	1

# Problem: undriven dynamic nodes



`a & b -> _out-`  
`~a & ~b -> _out+`

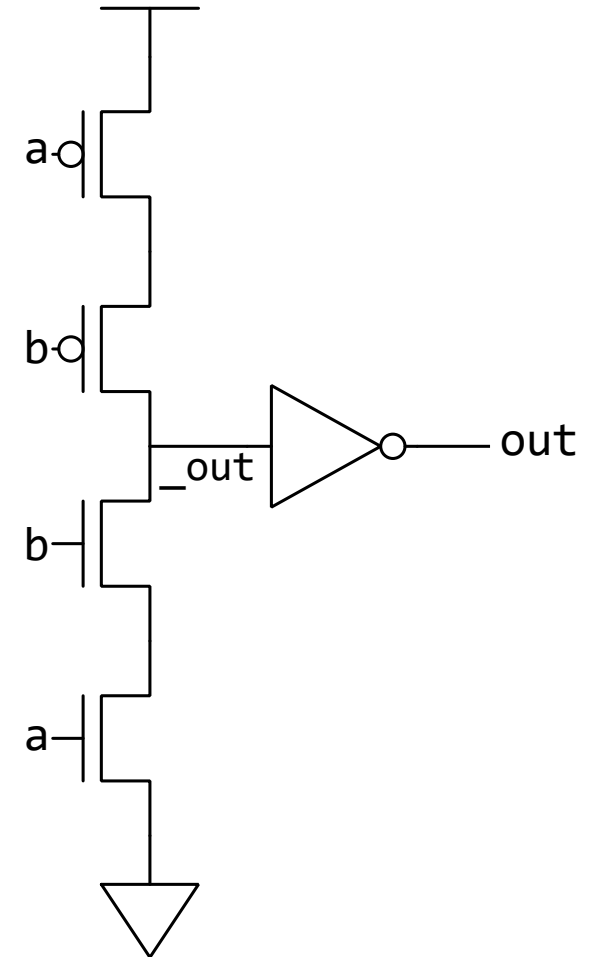
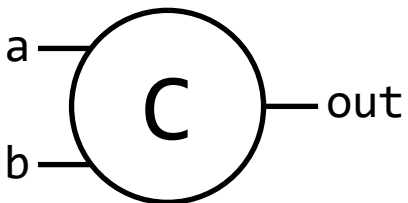
a	b	_out
0	0	1
0	1	hold previous state
1	0	hold previous state
1	1	0



Solution: staticizers

# C-element PRS to SPICE example

```
defproc celem (bool? a,b; bool! out)
{
  bool _out;
  prs {
    a & b #> _out-
    _out => out-
  }
}
```



# SPICE basics

Define new subcircuit (cell):  
`.subckt name ports`

Comments begin with \*  
Metadata generated by prs2net

MOSFET instances:  
`Mname D G S B type <param=val>`

End of inv subcircuit:

Instantiate subcircuits hierarchically:  
`xname ports cellname`

```
*----- act defproc: inv<> -----
* raw ports:  in out
.subckt inv in out
*.PININFO in:I out:O
*.POWER VDD Vdd
*.POWER GND GND
*.POWER NSUB GND
*.POWER PSUB Vdd
* --- node flags ---
* out (combinational)
* --- end node flags ---
M0_ Vdd in out Vdd p W=1.5U L=0.6U
M1_ GND in out GND n W=0.9U L=0.6U

.ends
*----- end of process: inv<> -----

*----- act defproc: buf<> -----
* raw ports:  in out
.subckt buf in out
xstage1 in __out inv
xstage2 __out out inv
.ends
*----- end of process: buf<> -----
```

# C-element PRS to SPICE example

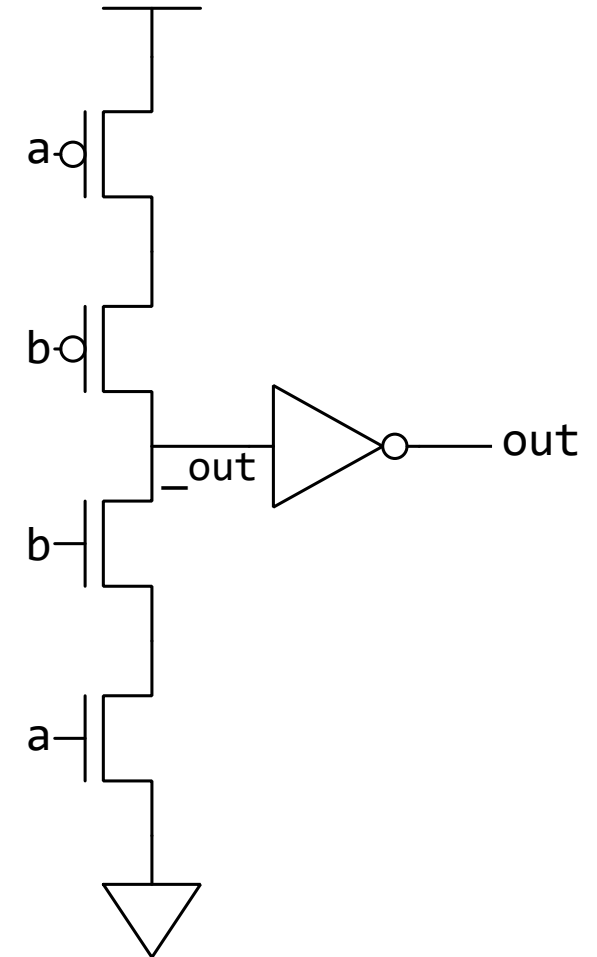
```
ben@summerschool:week3$ █
```

```
I
```



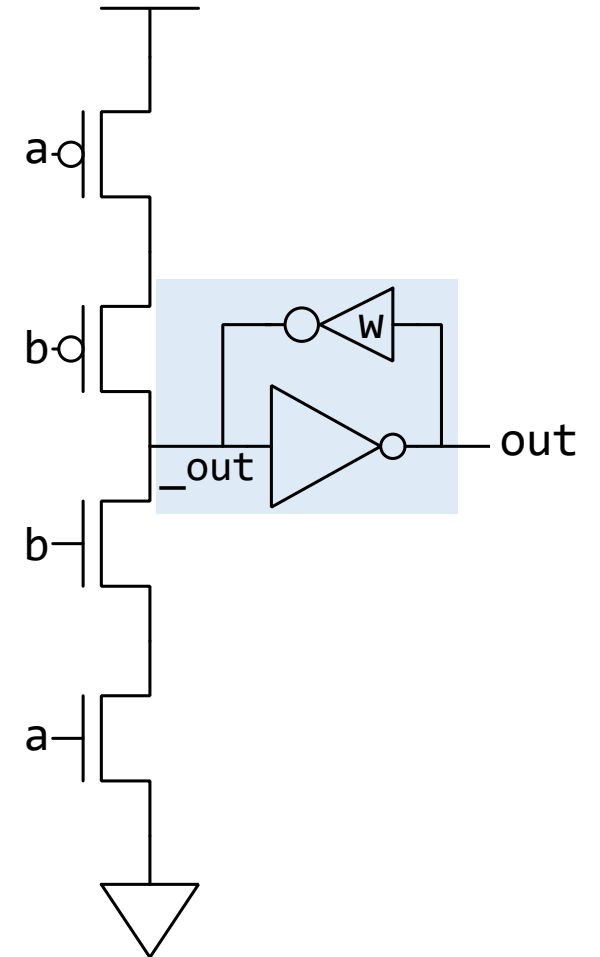
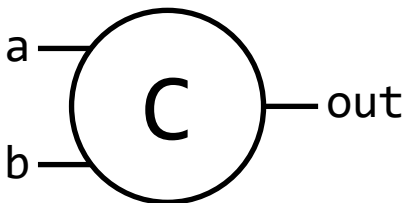
# C-element PRS to SPICE example

```
[ben@summerschool:week3$ vi celem.act ]
[ben@summerschool:week3$ prs2net celem.act -p celem ]
*
*---- act defproc: celem<> -----
* raw ports:  a b out
*
*.subckt celem a b out
*.PININFO a:I b:I out:O
*.POWER VDD Vdd
*.POWER GND GND
*.POWER NSUB GND
*.POWER PSUB Vdd
*
* --- node flags ---
*
* __out (state-holding): pup_reff=0.8; pdn_reff=1.33333
* out (combinational)
*
* --- end node flags ---
*
M0_ Vdd a #6 Vdd p W=1.5U L=0.6U
M1_ Vdd __out out Vdd p W=1.5U L=0.6U
M2_keeper Vdd GND #8 Vdd p W=0.9U L=5.4U
M3_ GND a #3 GND n W=0.9U L=0.6U
M4_ GND __out out GND n W=0.9U L=0.6U
M5_keeper GND Vdd #9 GND n W=0.9U L=13.8U
M6_ #3 b __out GND n W=0.9U L=0.6U
M7_ #6 b __out Vdd p W=1.5U L=0.6U
M8_keeper #8 out __out Vdd p W=0.9U L=0.6U
M9_keeper #9 out __out GND n W=0.9U L=0.6U
.ends
*---- end of process: celem<> -----
ben@summerschool:week3$ █
```



# C-element with weak keeper staticizer

```
defproc celem (bool? a,b; bool! out)
{
  bool _out;
  prs {
    a & b #> _out-
    _out => out-
  }
}
```



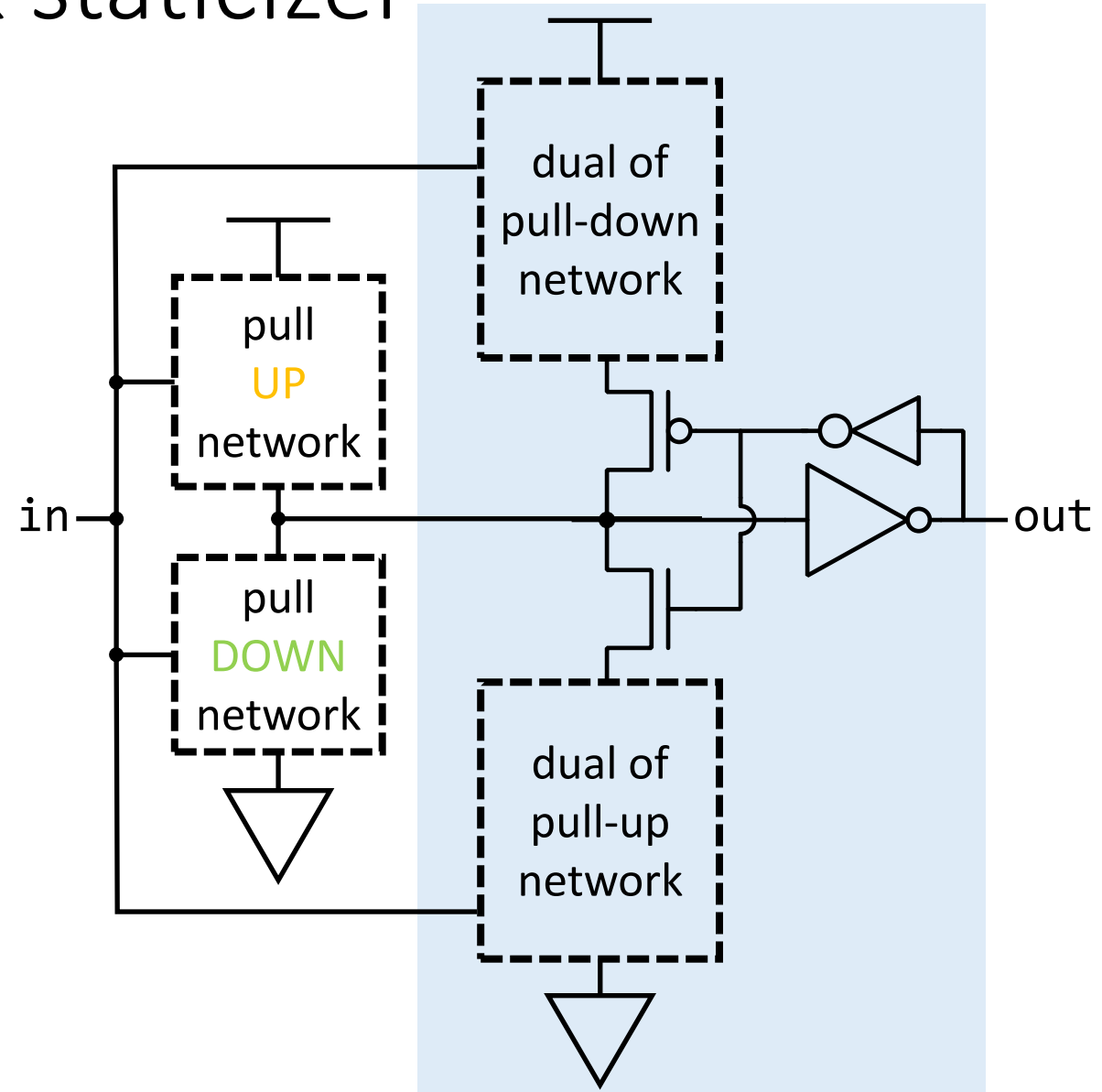
# Combinational feedback staticizer

Converts **state-holding** gate to be **combinational**

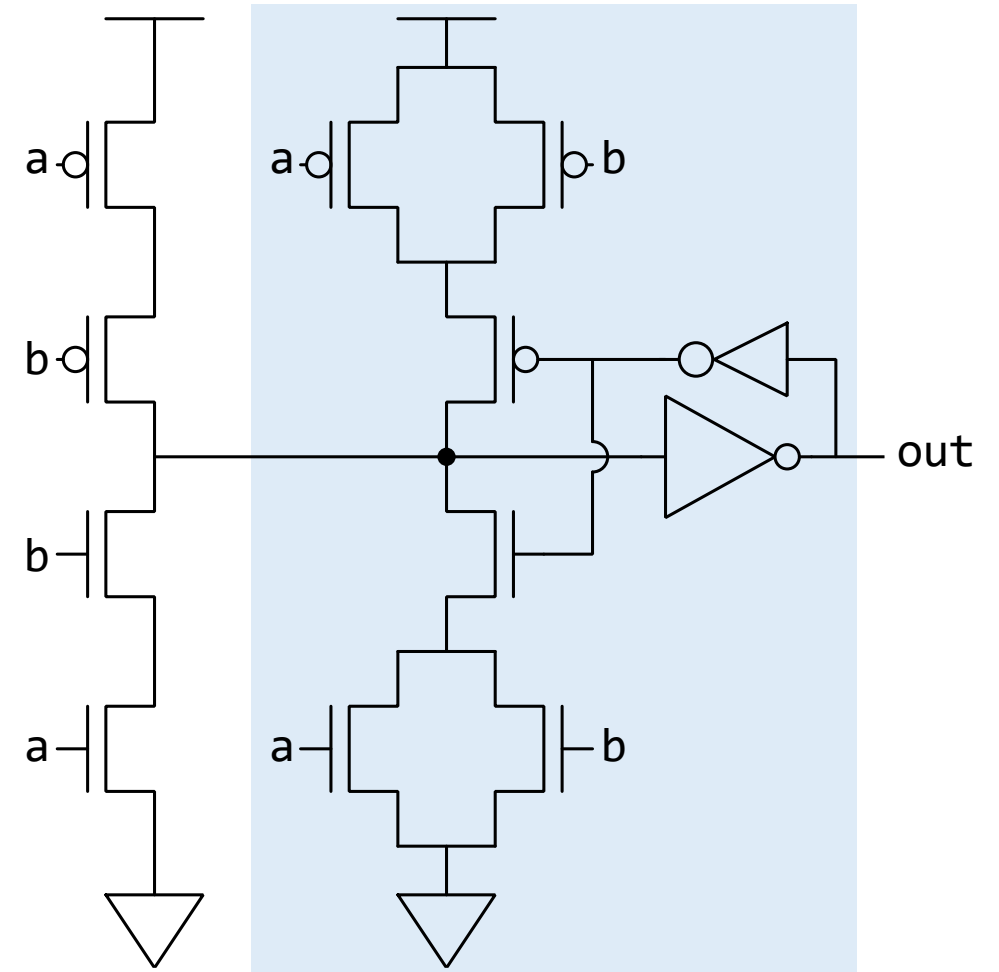
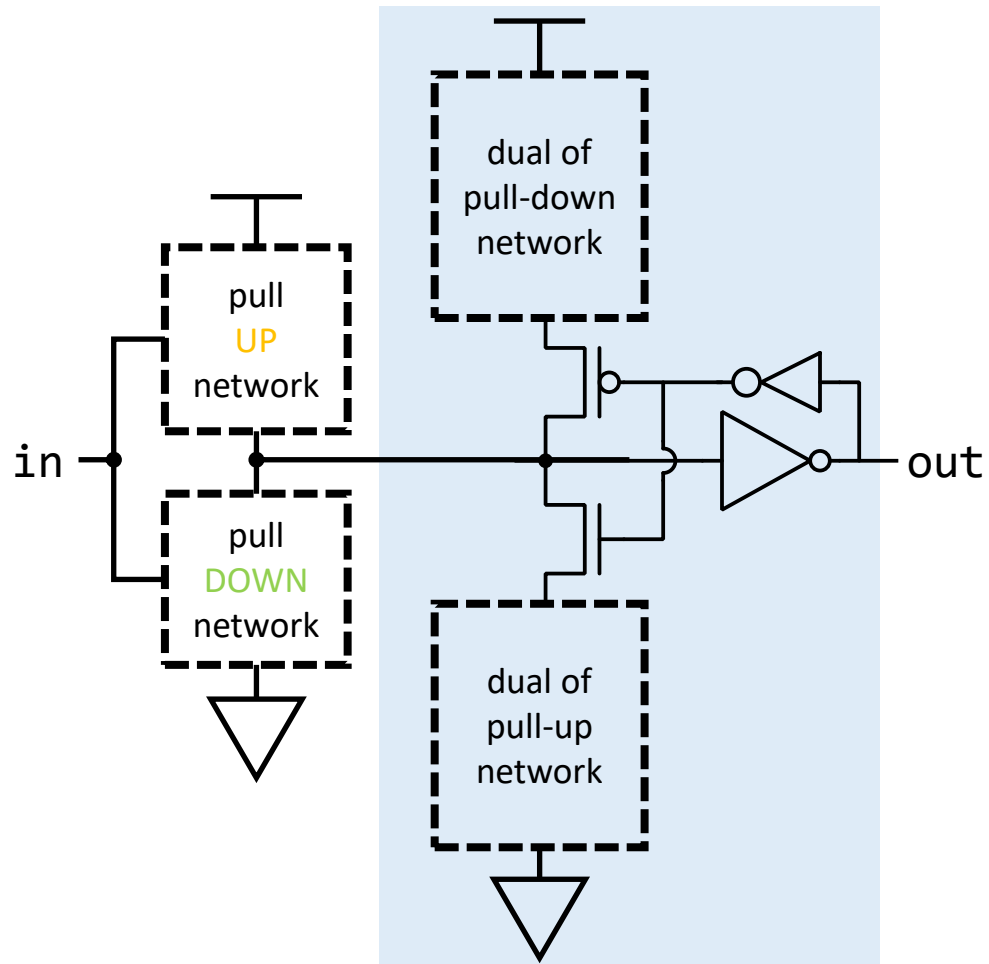
**Dual network** = inverse logic function

- Between a network and its dual, exactly one is conducting – never **interfering**

If neither **UP** nor **DOWN** is conducting (original **state-holding** case), *both* dual networks will conduct and feedback from out selects which is enabled



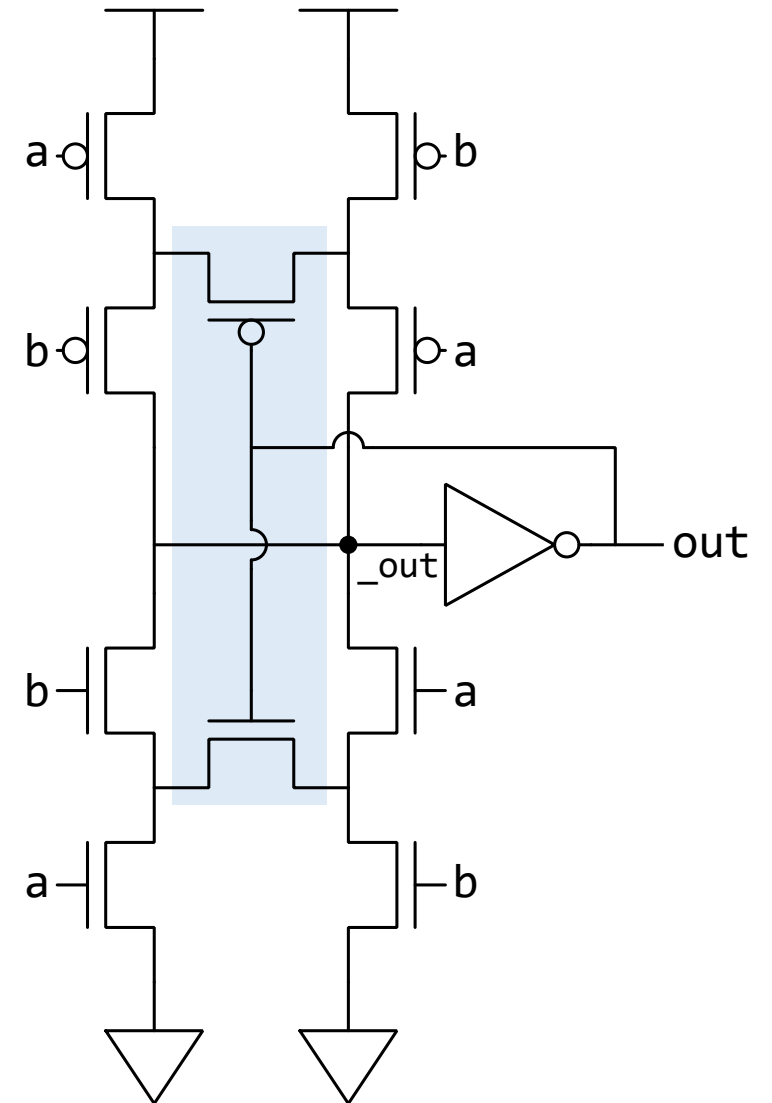
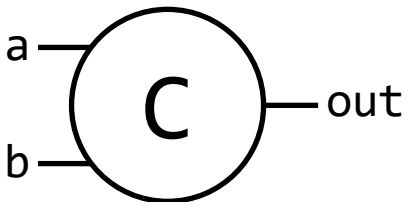
# C-element with combinational feedback



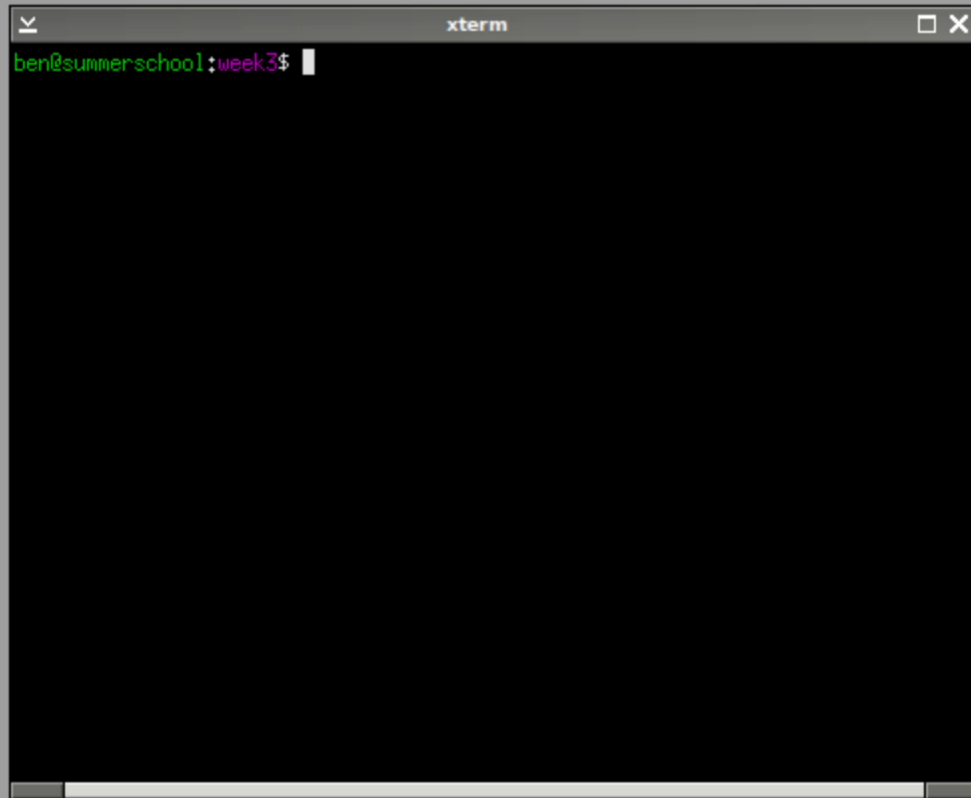
# van Berkel C-element

```
defproc celem_H (bool? a,b; bool! out)
{
  bool _out;
  bool nmid[2], pmid[2];
  prs {
    // N-stack
    [keeper=0] a -> nmid[0]-
    [keeper=0] b -> nmid[1]-
    passn (out, nmid[0], nmid[0])
    passn (b, nmid[0], _out)
    passn (a, nmid[1], _out)

    // Symmetric P-stack, out inverter
    ...
  }
}
```



Layout generation

A terminal window titled "xterm" with a black background. The prompt "ben@summerschool:~:week3\$" is displayed in green and purple text at the top left, followed by a white cursor. The window has a standard Linux-style title bar with a maximize button, a close button, and a small icon on the left.

```
ben@summerschool:~:week3$
```

# Generating transistor stacks

```
# get_rect.ia: Generate rect layout files for sized cells
# Load design and generate transistor netlist
act:read "sized.act"
act:expand
act:top all_cells
ckt:cell-map
ckt:map
ckt:save-sp "all_cells.sp"

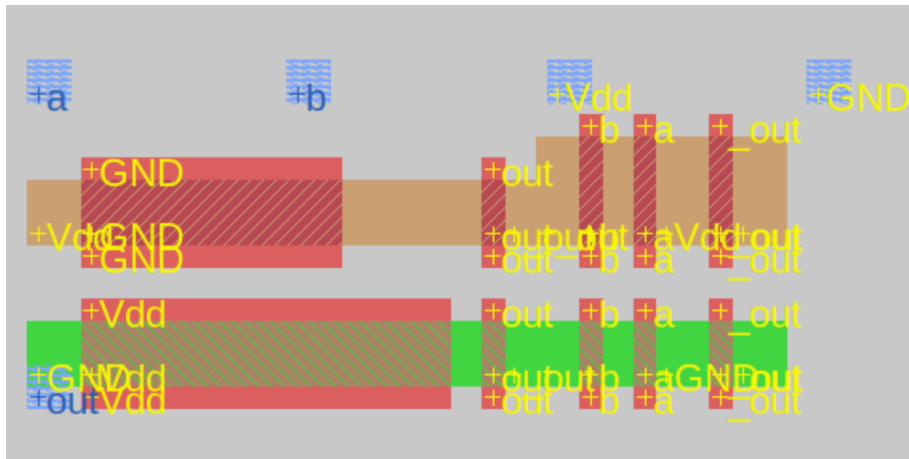
# Generate layout
load-scm "phydb.scm"
# (area multiplier and aspect ratio arbitrary,
#  since we're only creating cells)
phydb:create 2 1 "output.lef"
act:layout:rect
```

```
> interact -Tsky1301 < get_rect.ia
> mag.pl *.rect > magic_cells.tcl
> magic -Tsky1301 [source magic_cells.tcl]
```

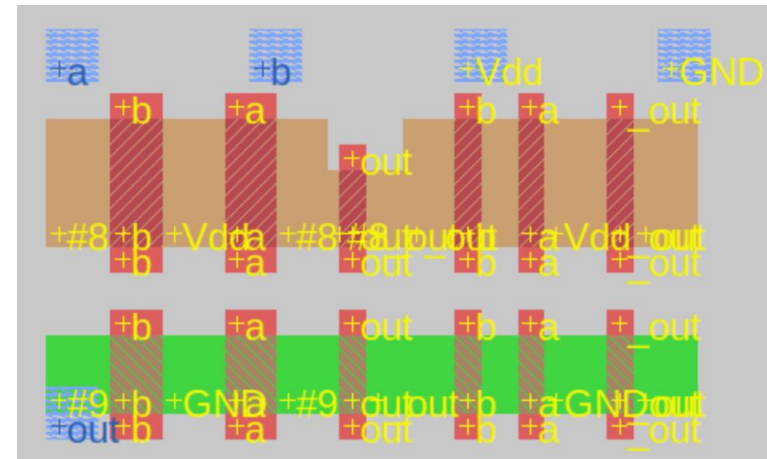


# Auto-generated transistor stacks

## Weak keeper

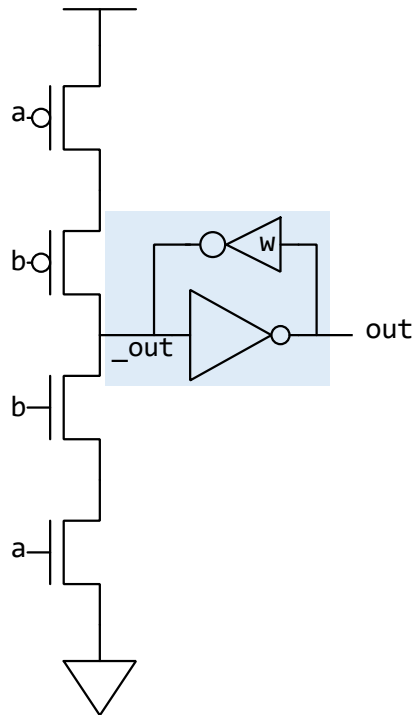


## Combinational feedback

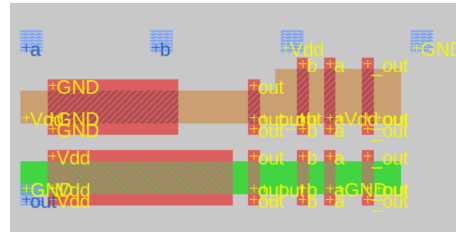


Complete cell wiring for use with place and route flow  
(e.g. gridded cell or standard cell)

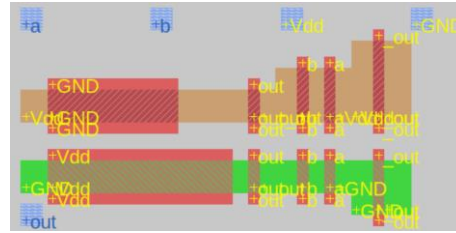
# Gate sizing



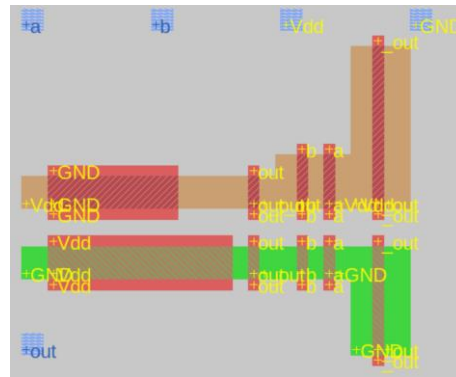
```
import "celem.act";
```



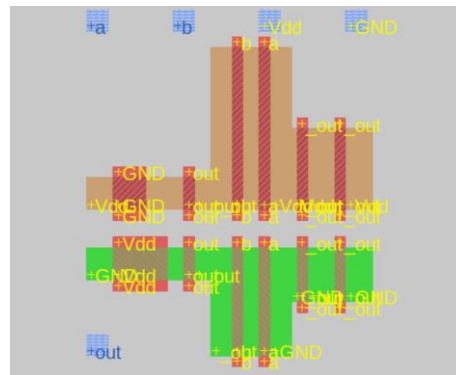
```
// Baseline x1 uses default sizes
defcell C2x1 <: celem() {}
```



```
// Double size output inverter
defcell C2x2 <: celem() {sizing{out{-2}}}
```



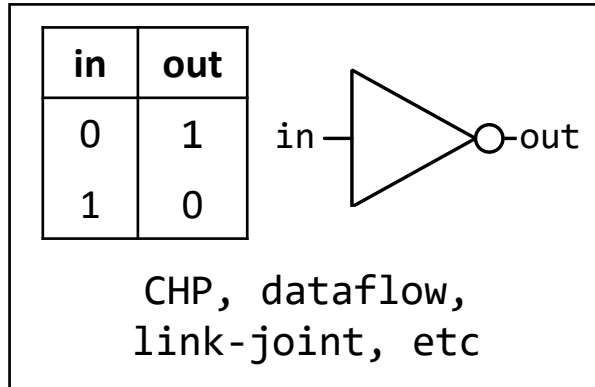
```
// 2x logic stacks, 4x output inverter
defcell C2x4 <: celem()
{sizing{_out{-2}; out{-4}}}
```



```
// That output inverter was a bit tall; fold in half
defcell C2x4F <: celem()
{sizing{_out{-2}; out{-2,svt,2}}}
```

# Full custom flow example

## Design specification



write or  
compile

## Production rules

```
defproc inv (bool? in; bool! out)
{
  prs { in => out- }
}
```

prs2net  
prs2sim

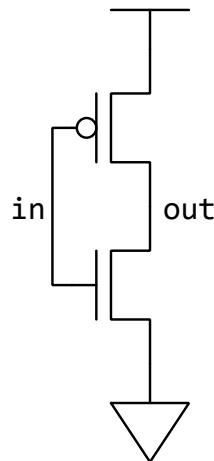
## Netlist (SPICE)

```
*---- act defproc: inv<> ----
.subckt inv in out
*.PININFO in:I out:O
*.POWER VDD Vdd
*.POWER GND GND
*.POWER NSUB GND
*.POWER PSUB Vdd
* --- node flags ---
* out (combinational)

M0_ Vdd in out Vdd p W=1.5U L=0.6U
M1_ GND in out GND n W=0.9U L=0.6U

.ends
*---- end of process: inv<> ----
```

## Schematic



full custom or  
semi-automated

extracted  
parasitics

## Layout

