Custom async circuit design from production rules to netlist

Benjamin Hill

benjamin.hill@intel.com

ASYNC Summer School 2022

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.

Implementing custom circuit designs

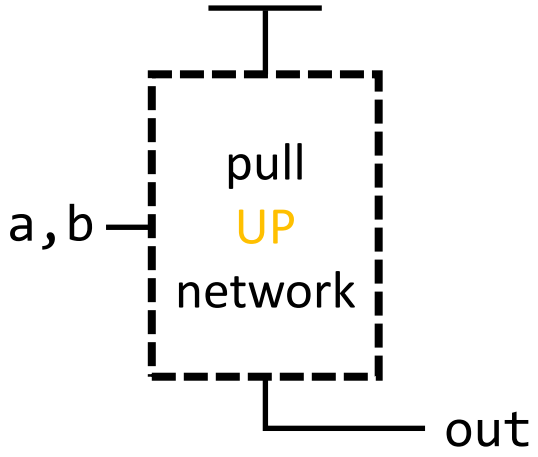
Our task:
Given a gate-level system expressed as production rule set (PRS), generate netlist (SPICE) and physical implementation (layout)

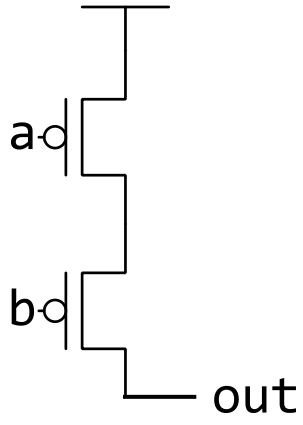
Do as little full custom design as possible!

Stay tuned for later sessions describing more automated parts of the Yale physical implementation flow

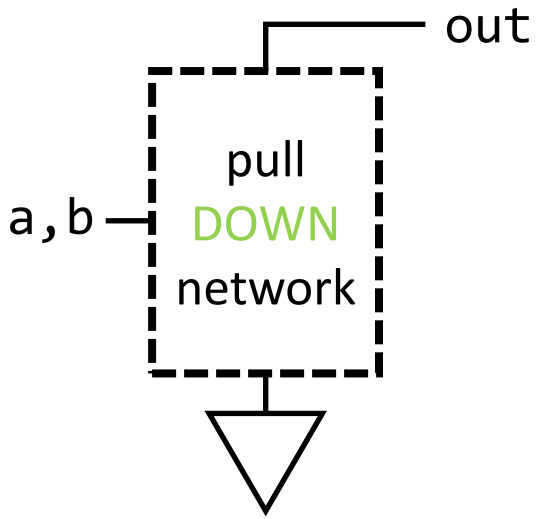
Production rule basics

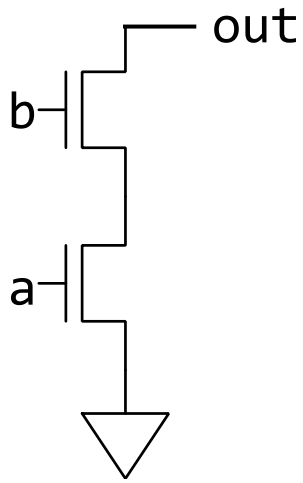
```
~a & ~b -> out+
```





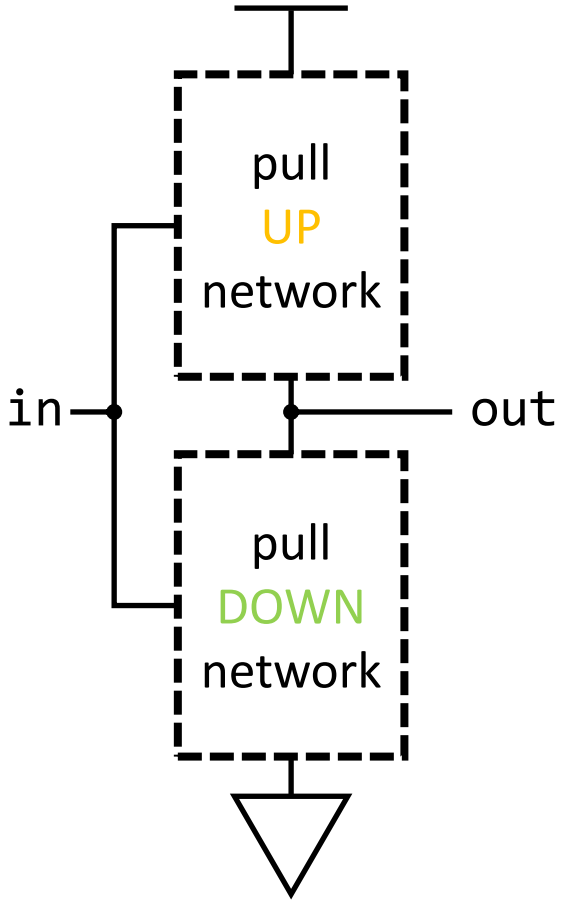

```
a & b -> out-
```

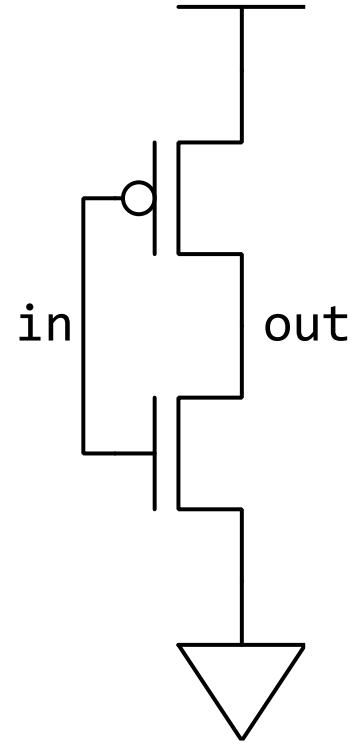




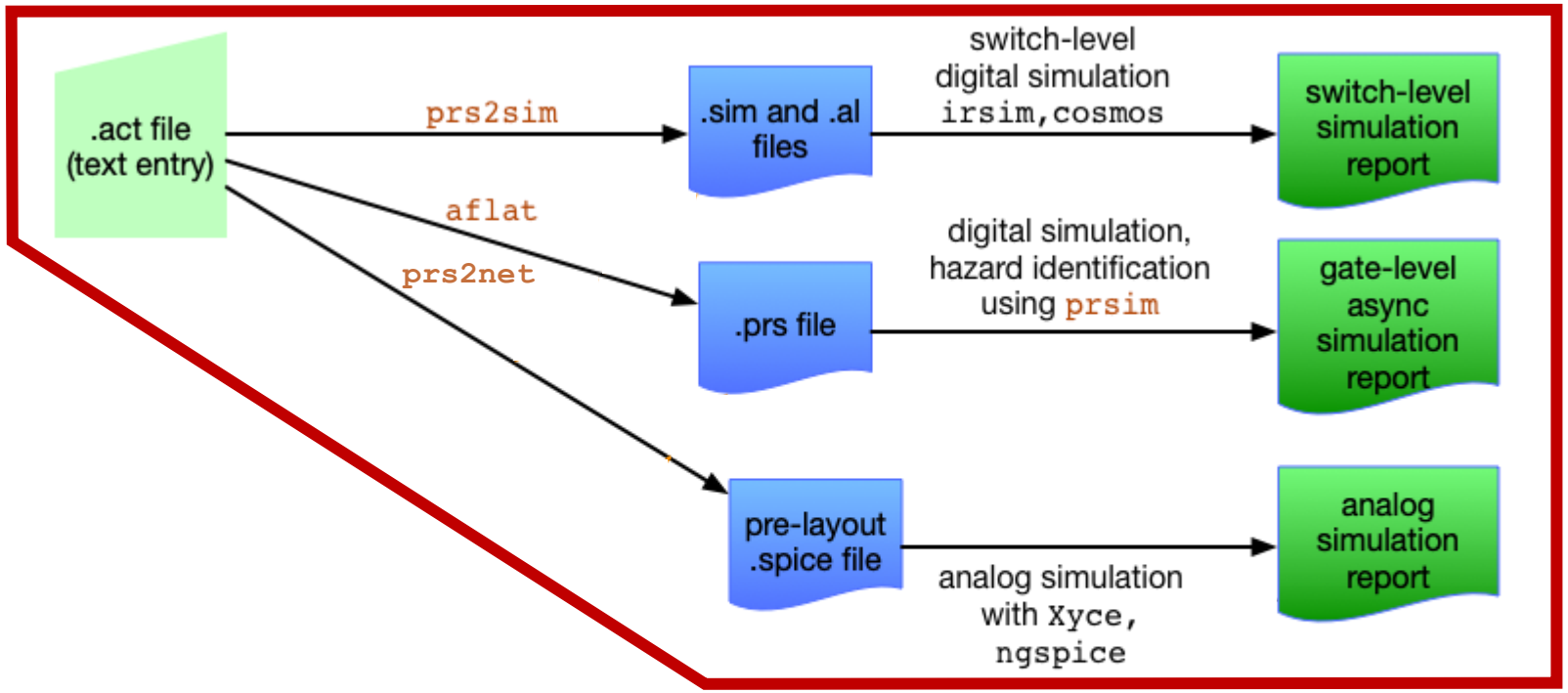
Production rule sets form gates

```
~in -> out+
 in -> out-
```





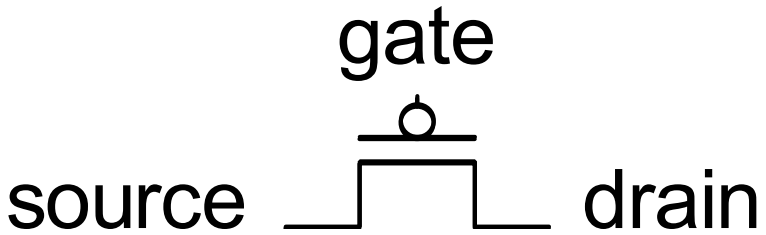
Custom design flow

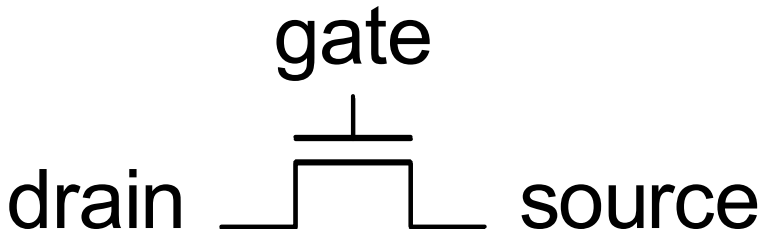


https://avlsi.csl.yale.edu/act/doku.php?id=custom:start

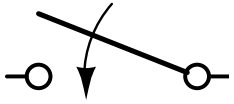
CMOS transistors

CMOS transistors





voltage-controlled switch



PMOS

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

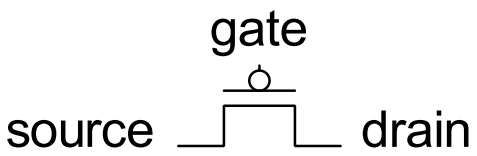
NMOS

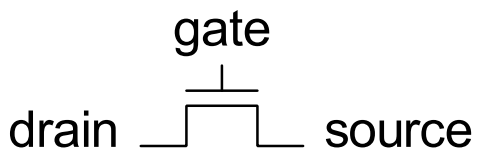
conducts current
between source and drain
(through N-type channel)
when gate is **HIGH**

PMOS

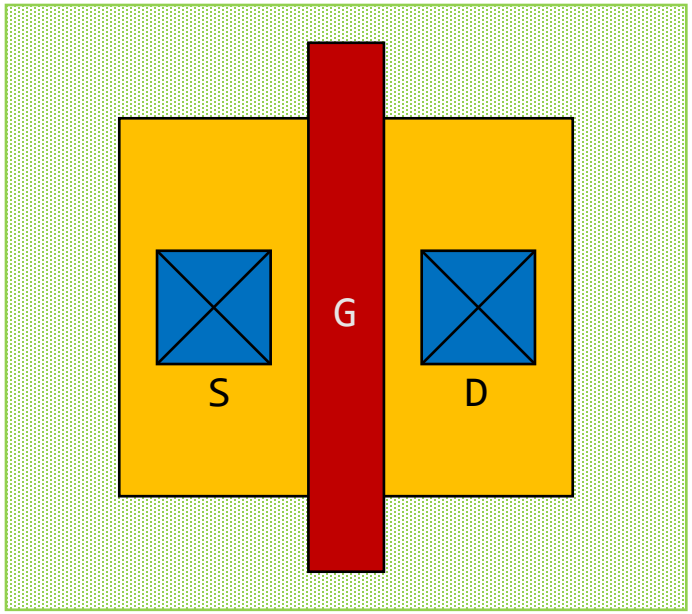
NMOS

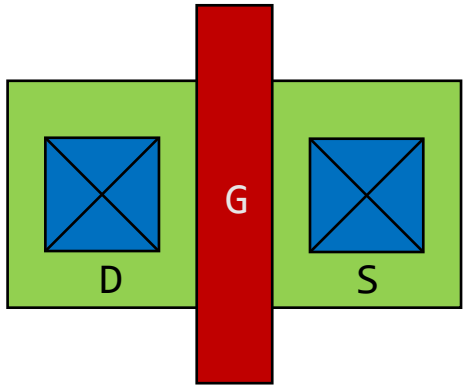
schematic symbol



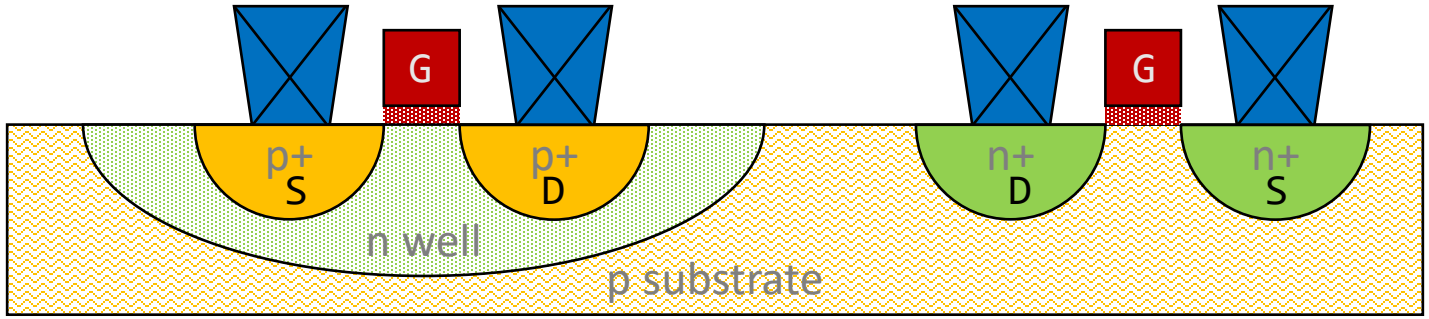


layout (top down view)

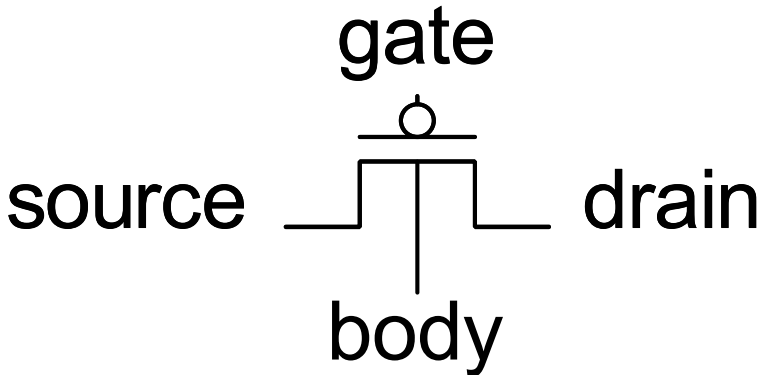




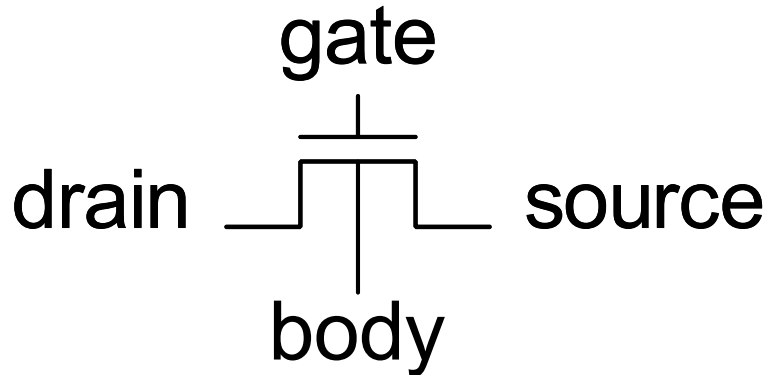
cross section



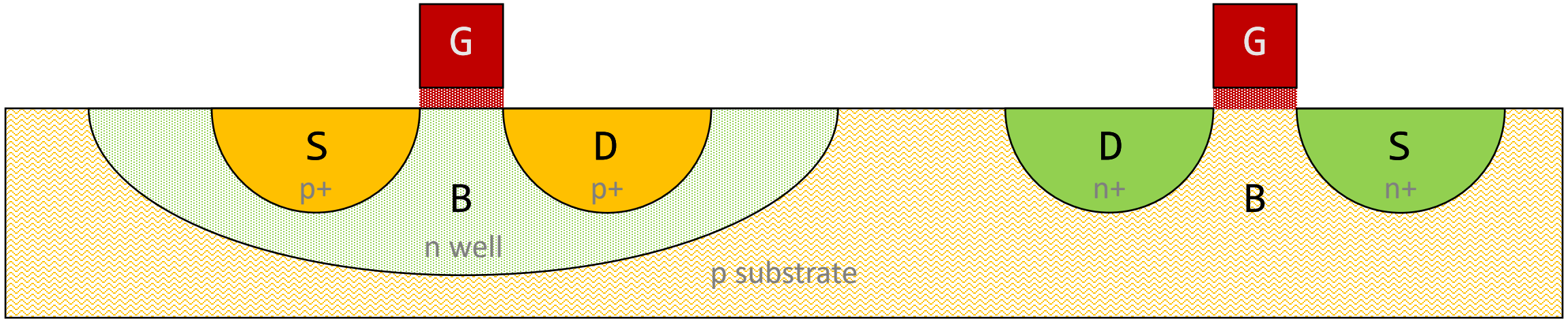
CMOS transistors



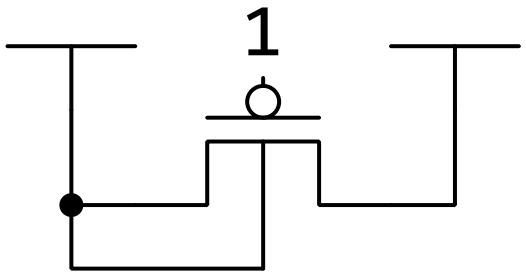
PMOS

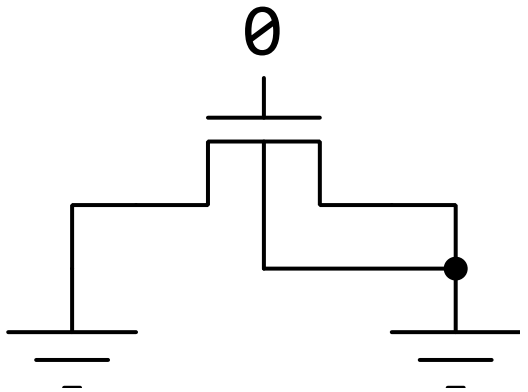


NMOS



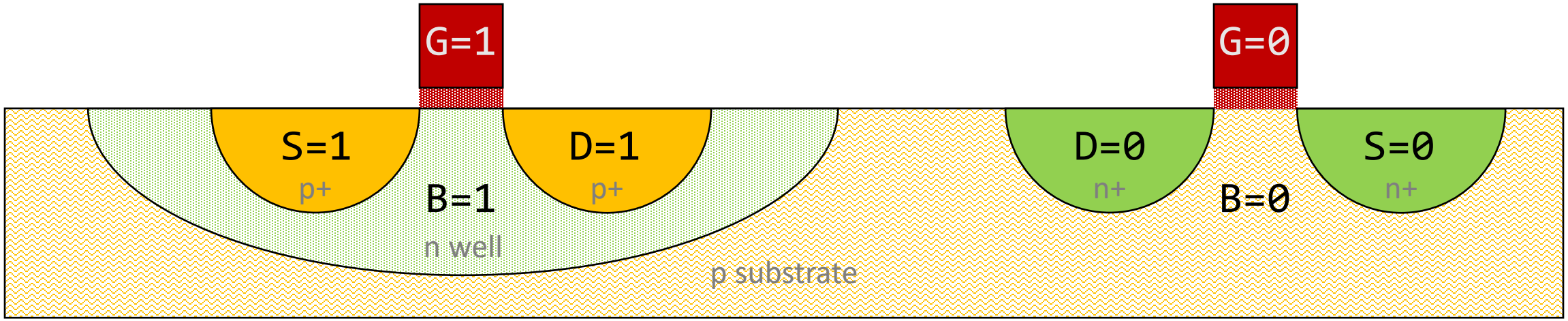
Transistor operation: cutoff





PMOS

NMOS

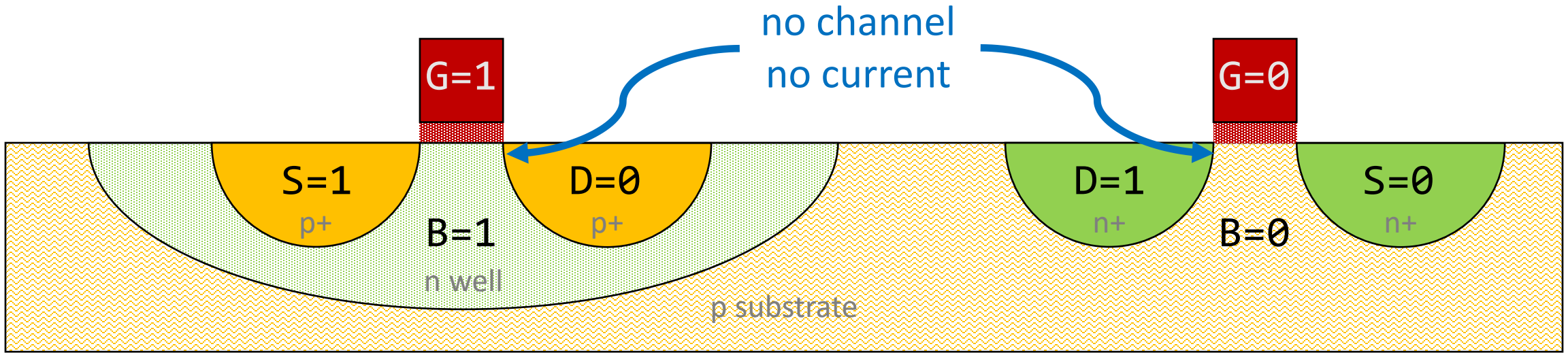


Transistor operation: channel formation

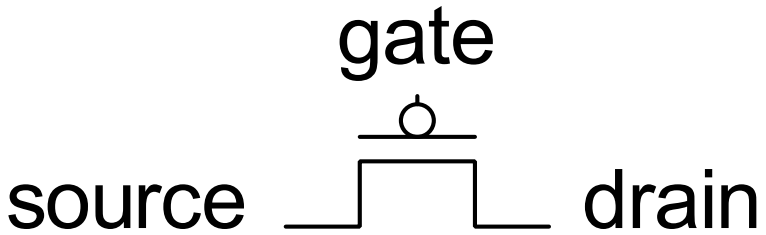
Transistor operation: saturation

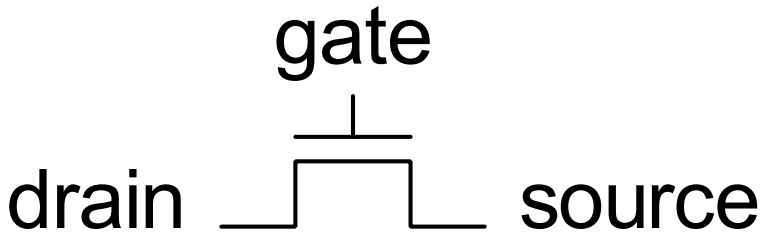
Transistor operation: cutoff

PMOS NMOS

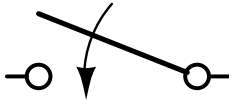


CMOS transistors





voltage-controlled switch



PMOS

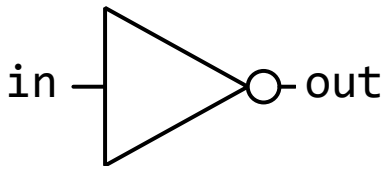
conducts current
between source and drain
(through P-type channel)
when gate is **LOW**

NMOS

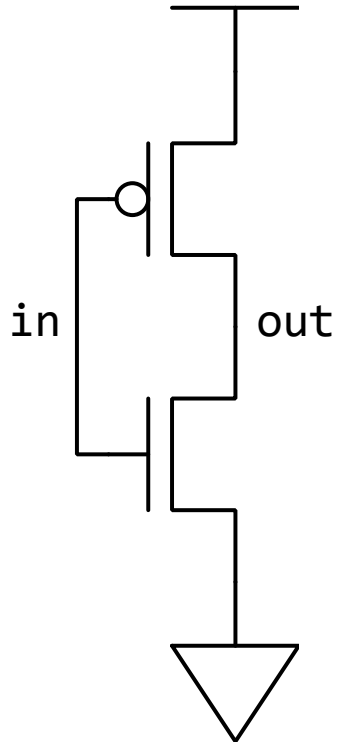
conducts current
between source and drain
(through N-type channel)
when gate is **HIGH**

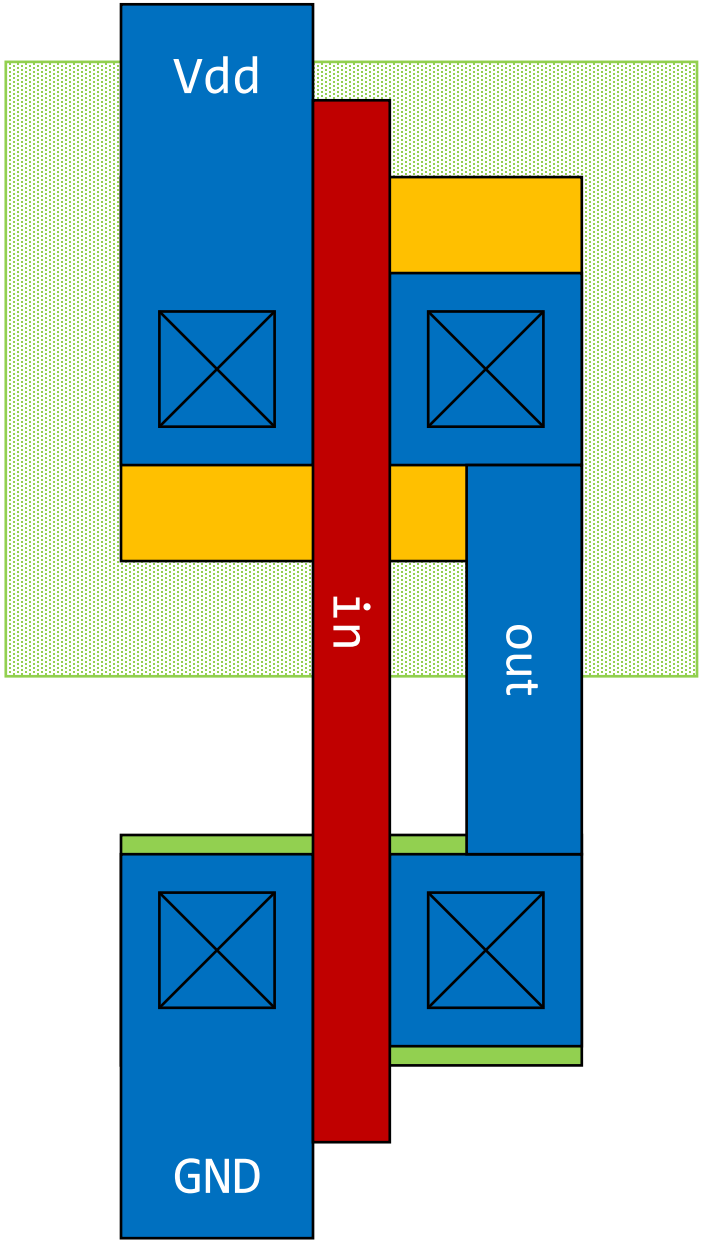
Layout and sizing

Layout example: inverter



in	out
0	1
1	0



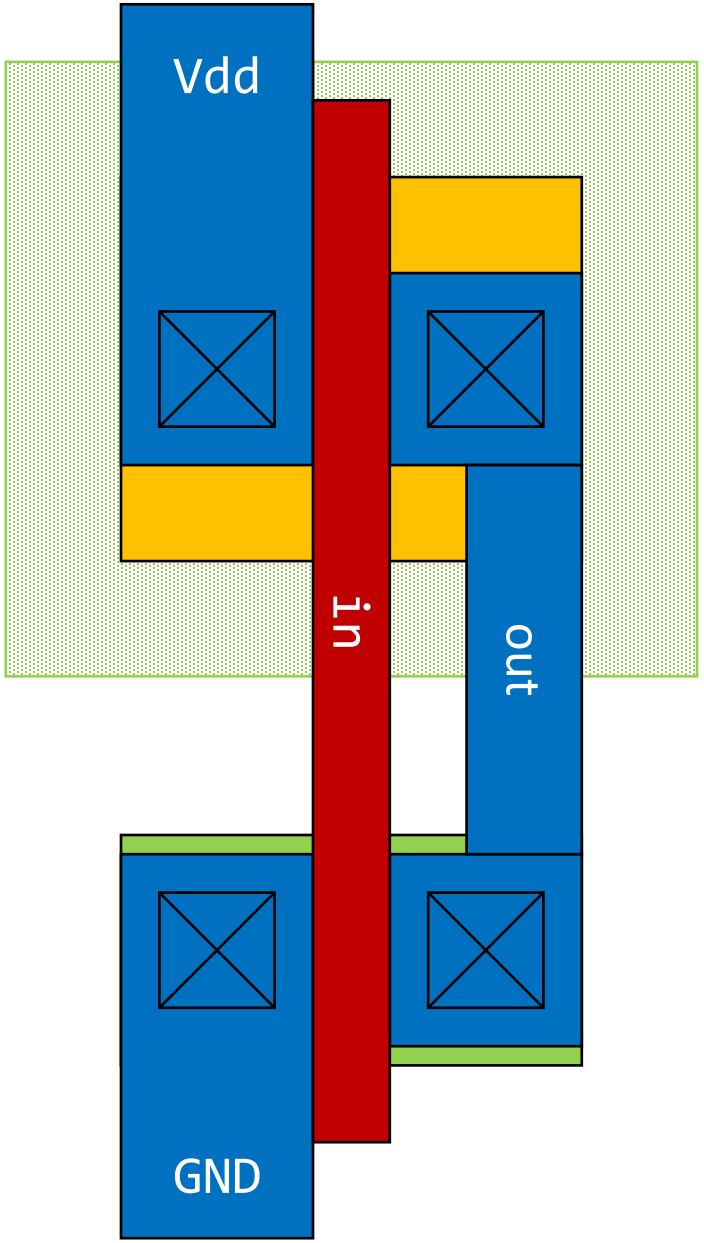


Design rules

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

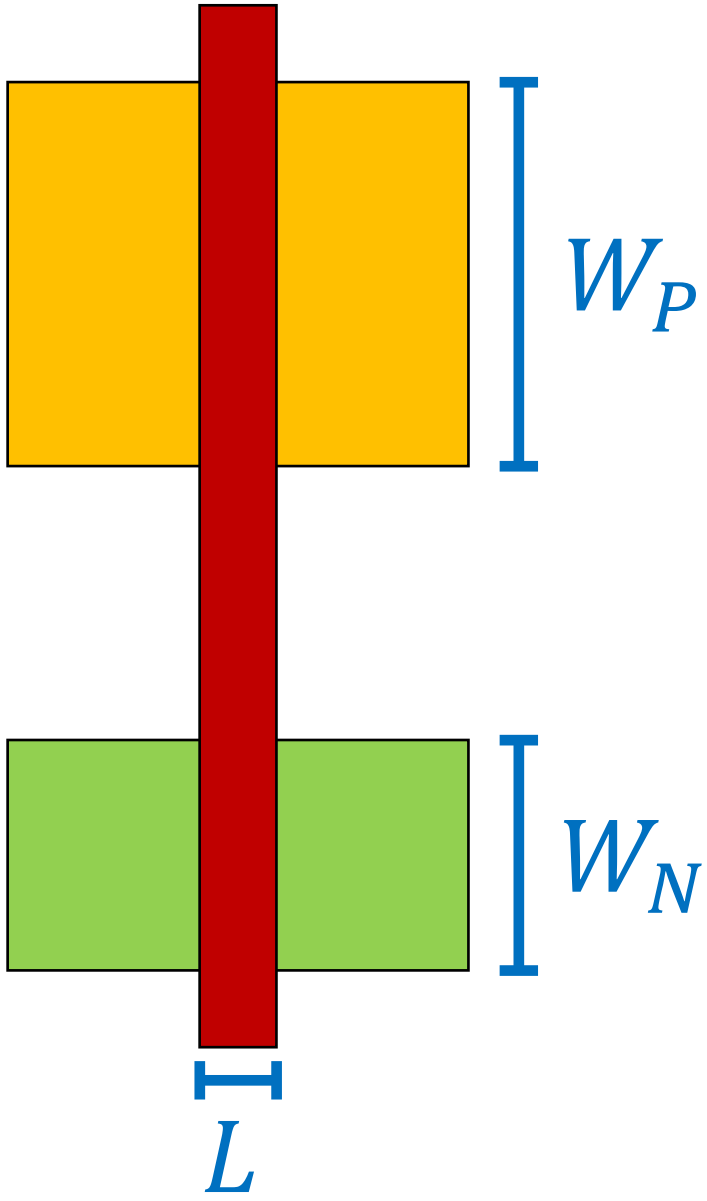


https://skywater-pdk.readthedocs.io/en/main/rules

Transistor sizing

Skywater 130 simplified design rules

λ	75 nm*
L	2 λ
W_N	6 λ
W_P	10 λ



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

https://github.com/asyncvlsi/sky130l

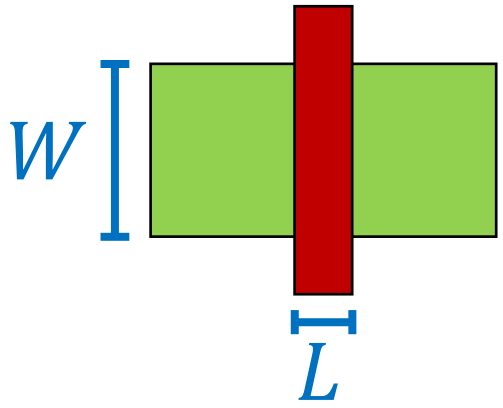
Sizing – how and why

- W, L, lambda
- ACT sizing body, prs body, default config
- Gates drive capacitance: think of simple R-C circuit
- Switching burns energy: charge and discharge capacitor
- Always trading off energy/area/performance

Transistor performance scaling intuition

Transistor device with
width and length parameters

What happens as we vary them?

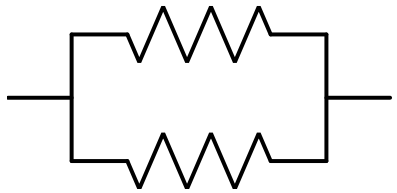


Analogy: resistors

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

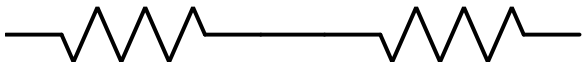
$$I = \frac{V}{R}$$

Decreased resistance,
increased current



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

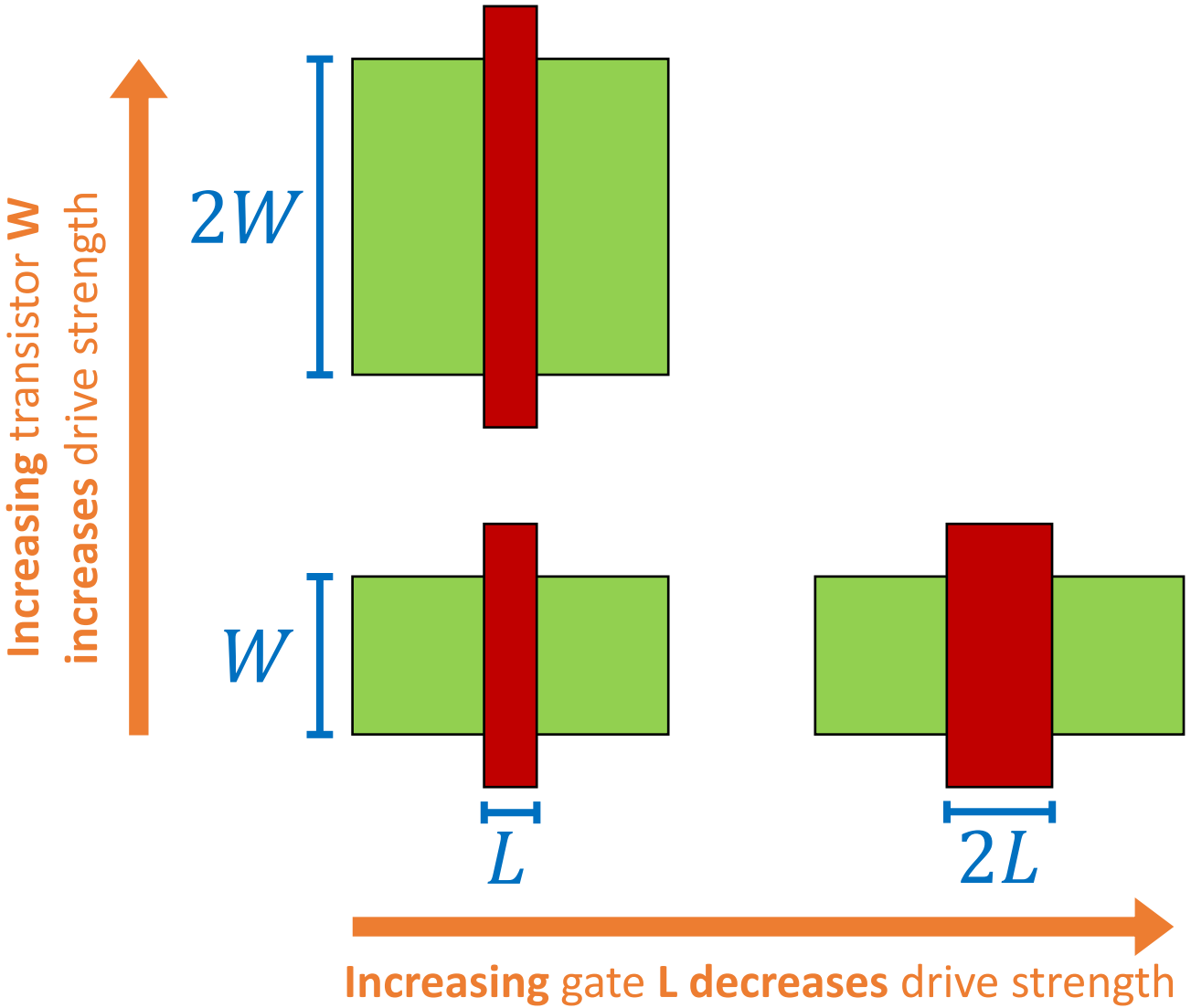
R



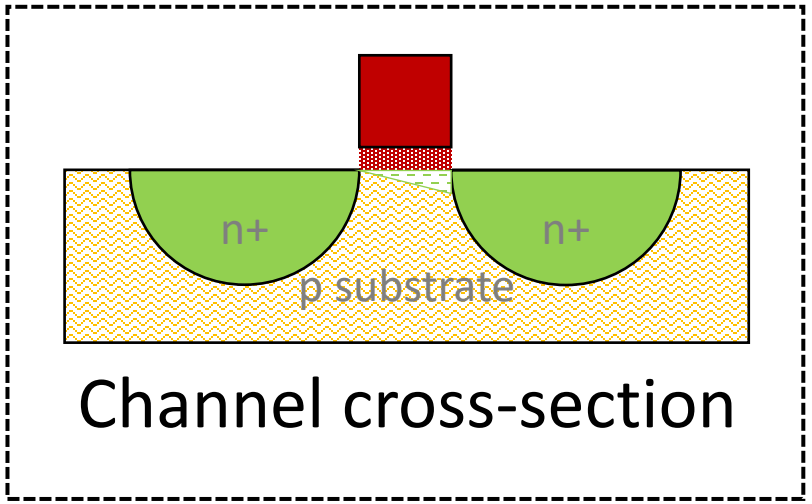
$$R_{series} = R_1 + R_2 = 2R$$

Increased resistance, **decreased** current

Transistor performance scaling intuition



Key performance metric:
current through transistor
effective resistance
"drive strength"



Digital designer summary

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

⇒ single stage logic is always inverting

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

⇒ use minimum gate length for digital logic (usually)

State-holding gates

Combinational vs state-holding gates

Combinational

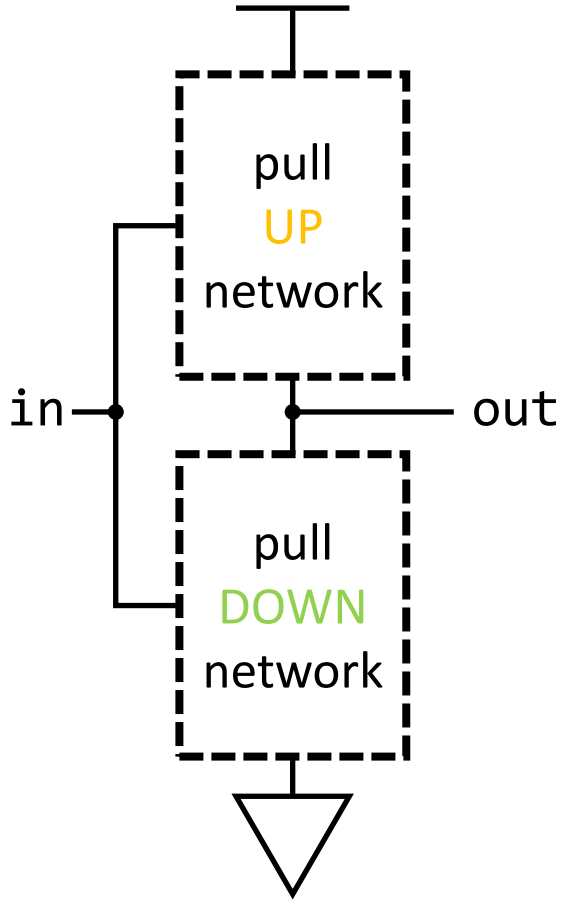
- Either UP or DOWN (but not both) is always conducting

State-holding

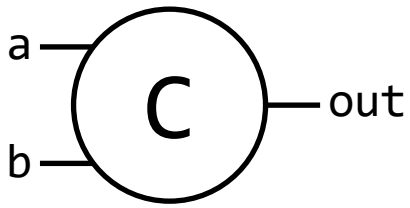
- At times neither UP nor DOWN is conducting
- `out` is undriven and maintains its previous value

Interfering

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

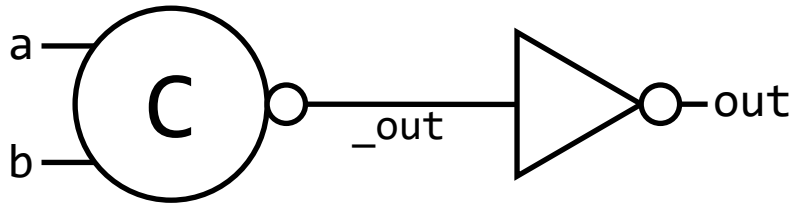


Example state-holding gate: Muller C-element



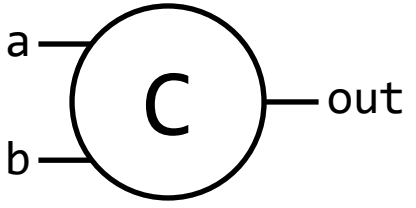

```
a &  b -> out+
~a & ~b -> out-
```

not CMOS implementable!




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

inverting C-element plus inverter

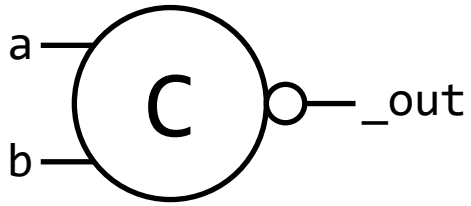



```
a & b #> _out-
 _out =>  out-
```

shorthand syntax

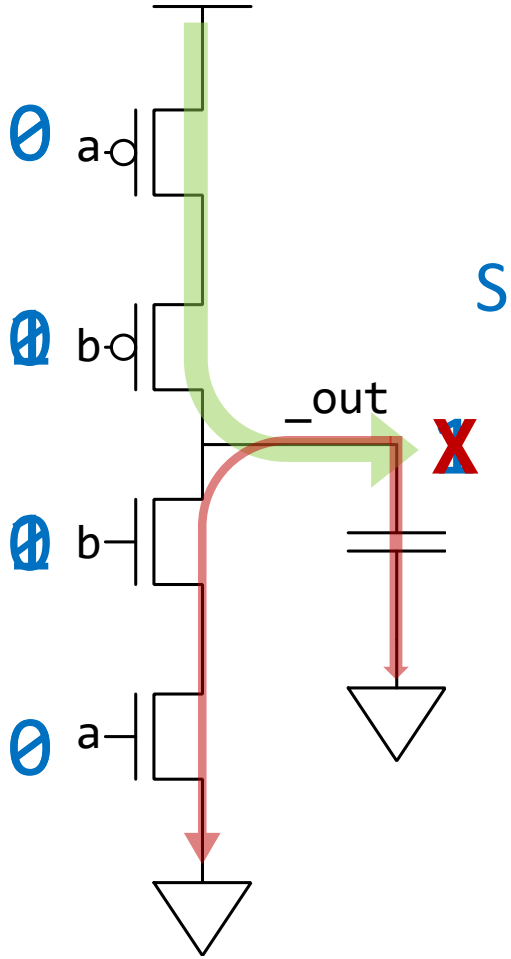
a	b	out
0	0	0
0	1	hold previous state
1	0	hold previous state
1	1	1

Problem: undriven dynamic nodes




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

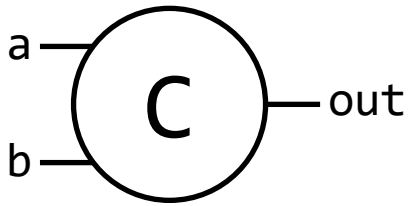
a	b	_out
0	0	1
0	1	hold previous state
1	0	hold previous state
1	1	0

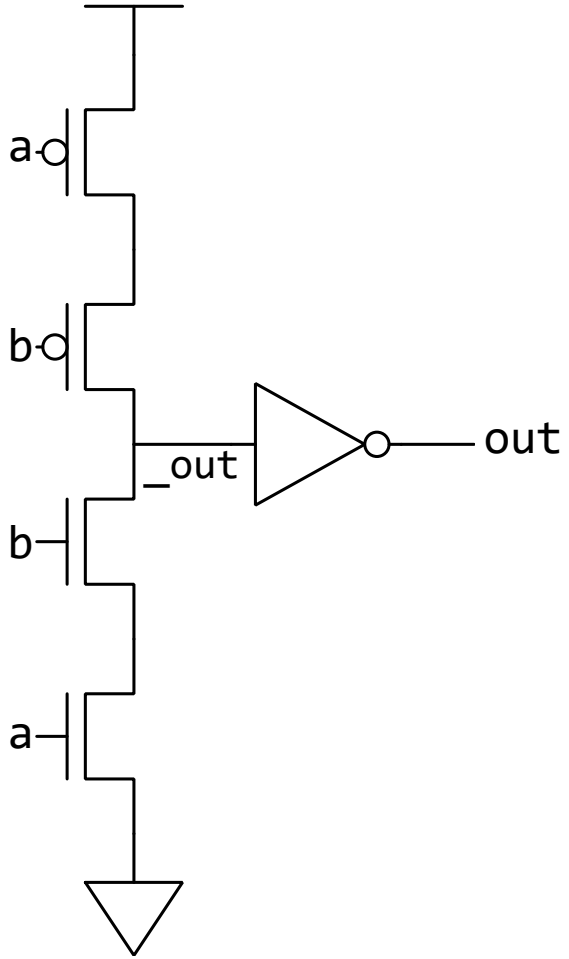


Solution: staticizers

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:
M*name* D G S B type <param=val>

End of inv subcircuit:

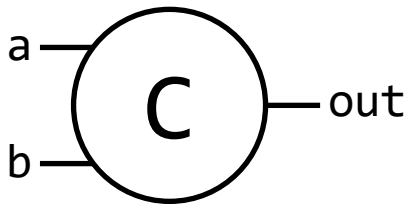
Instantiate subcircuits hierarchically:
x*name* ports cellname

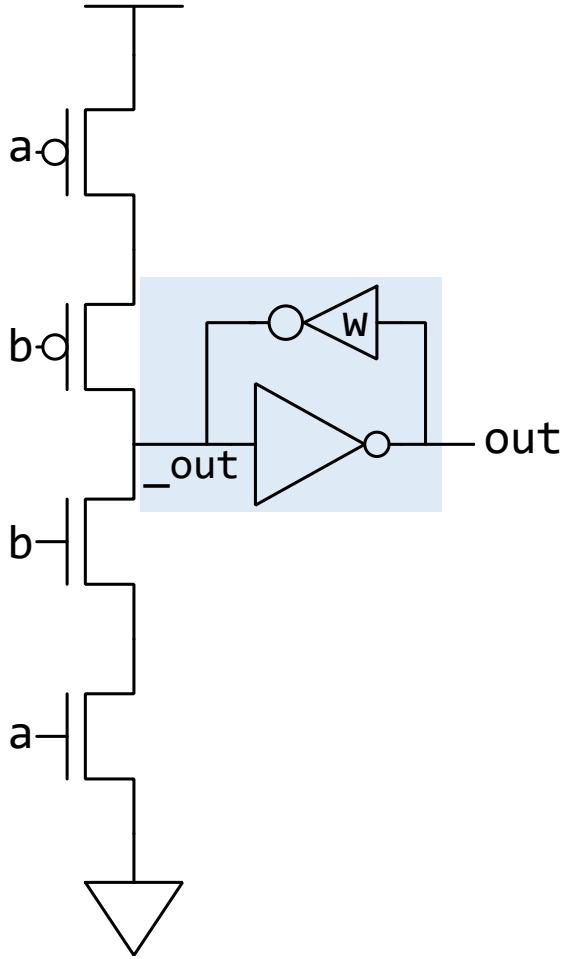
```
*---- act defproc: inv<> -----
* raw ports:  in out

.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

C-element with weak keeper staticizer

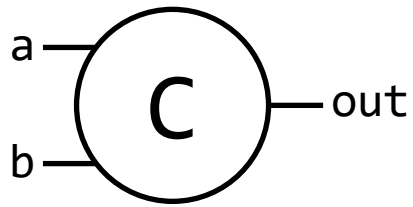
```
defproc celem (bool? a,b; bool! out)
{
  bool _out;
  prs {
    a & b #> _out-
    _out => out-
  }
}
```

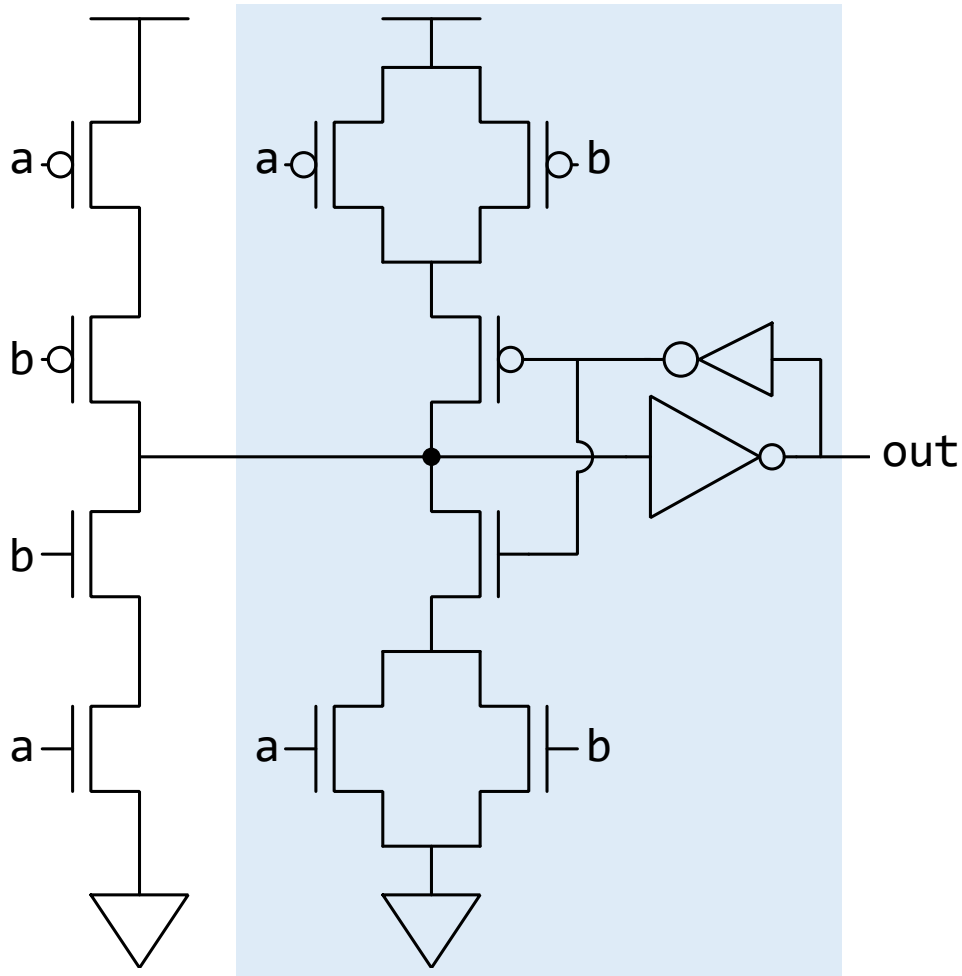




C-element with combinational feedback

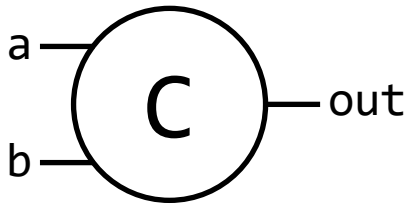
```
defproc celem_comb (bool? a,b; bool! out)
{
  bool _out;
  prs {
    [comb=1] a & b #> _out-
    _out => out-
  }
}
```

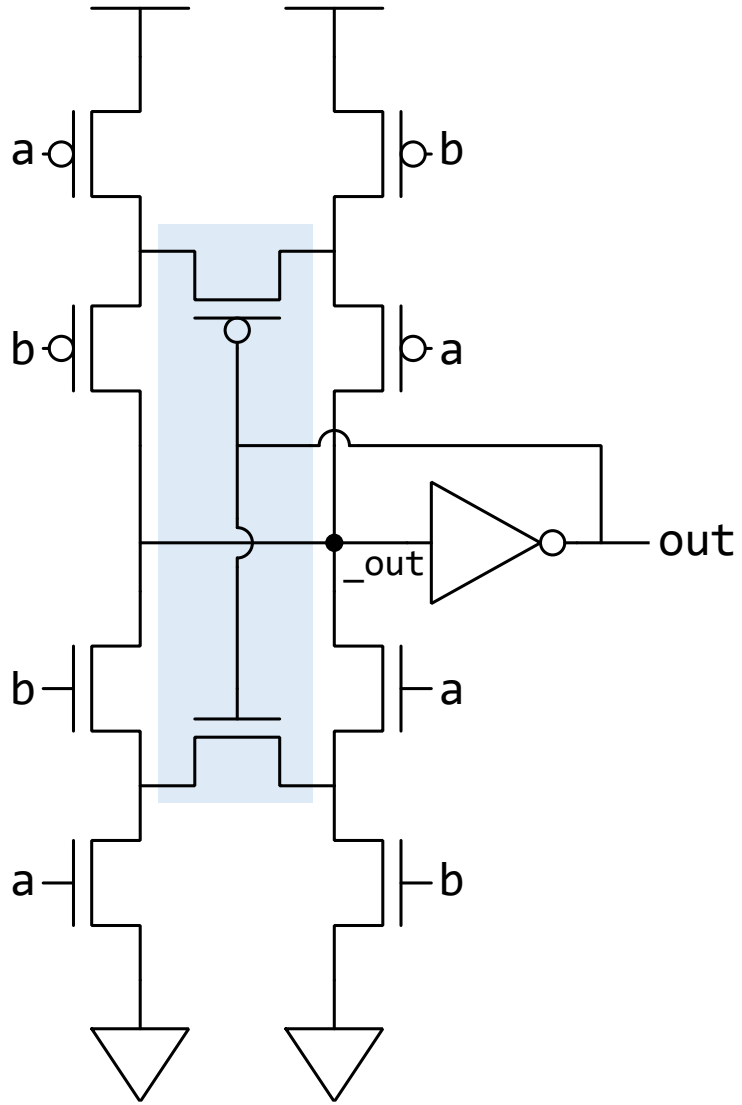




van Berkel C-element

```
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
    …
  }
}
```





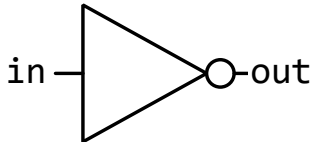
Simulation options

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

Full custom flow example

Design specification

in	out
0	1
1	0



CHP, dataflow,
link-joint, etc

write or
compile

Production rules

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

prs2net
prs2sim

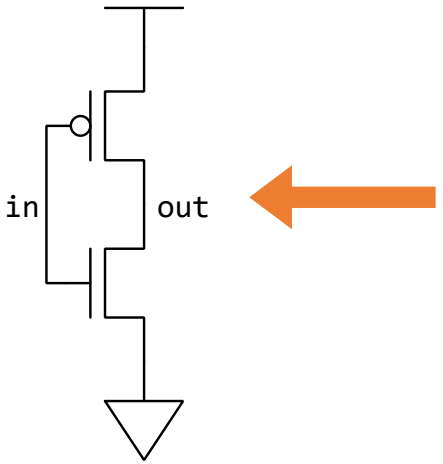
Netlist (SPICE)

```
*---- 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<> -----
```

full custom or
semi-automated

extracted
parasitics

Schematic



Layout