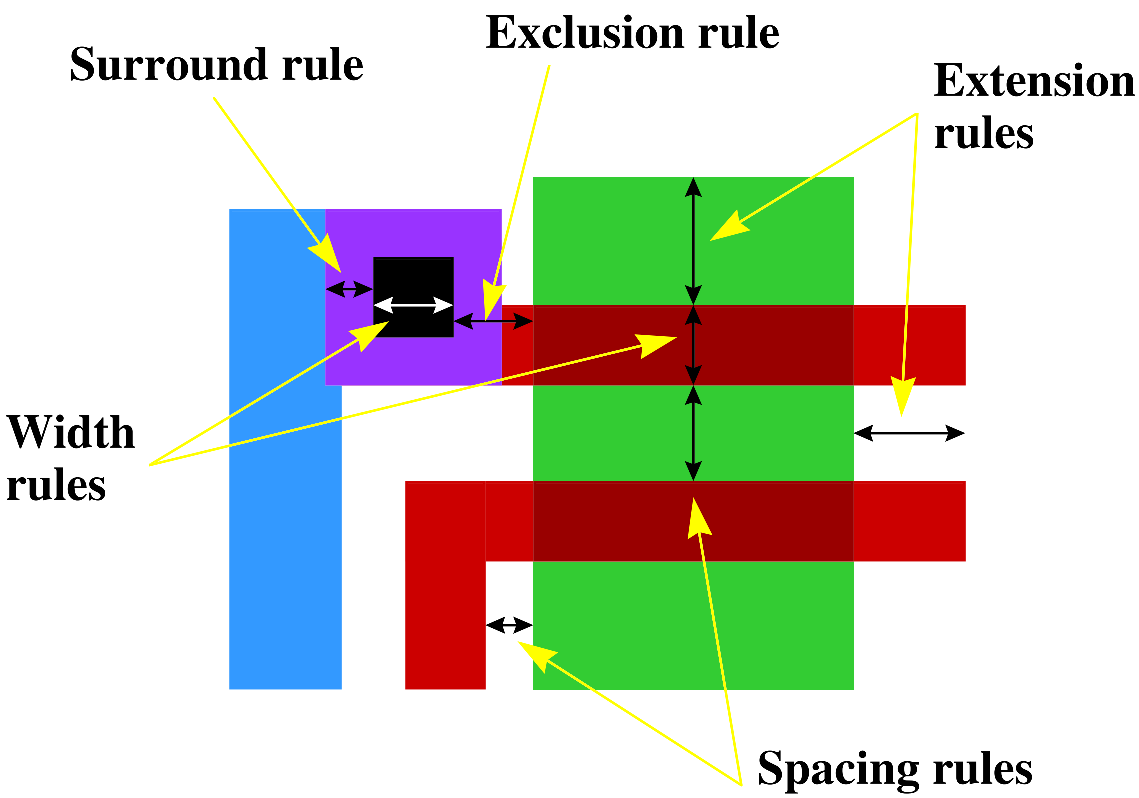
Design Rules



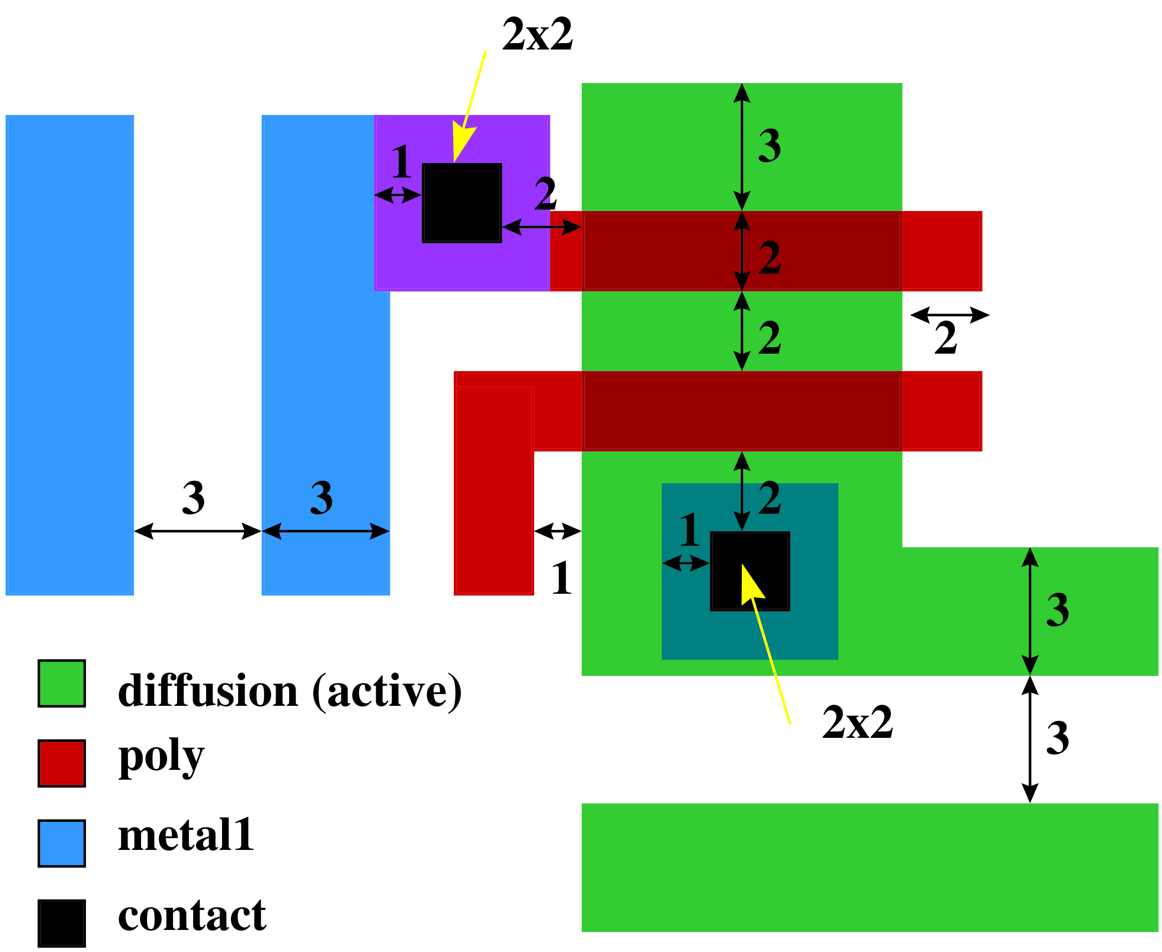
We can specify the design rules using some convenient units, e.g., microns but what happens if we want to manufacture the chip using different manufacturers?

One suggestion: use an abstract unit, the *lambda*, and scale the design to the appropriate actual dimensions when the chip is to be manufactured.

Lambda-based Rules

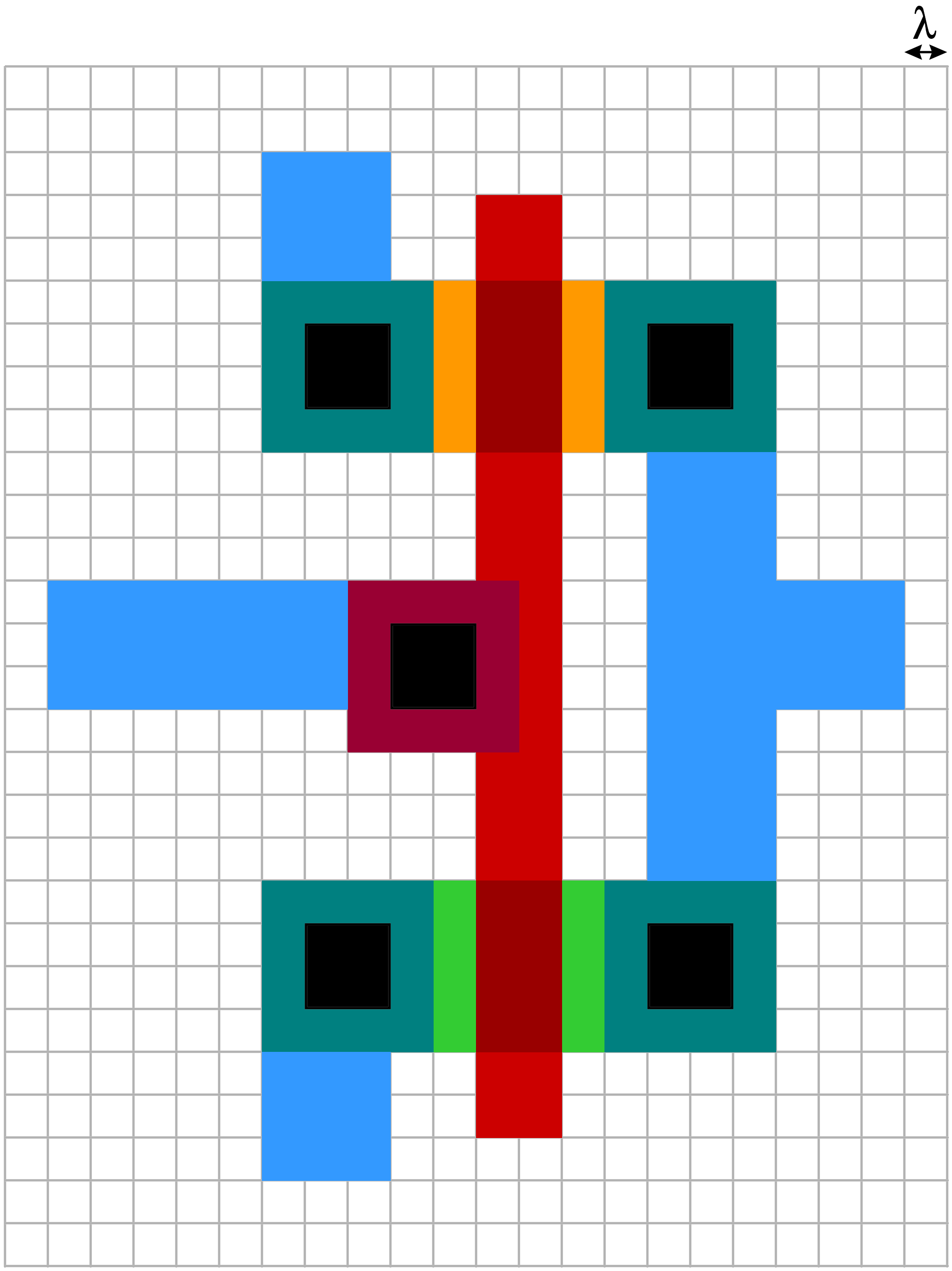
One lambda = one half of the “minimum” mask dimension, typically the length of a transistor channel.

Usually all edges must be “on grid”



For mosis2n, lambda = 1µm

Sample "Lambda" Layout

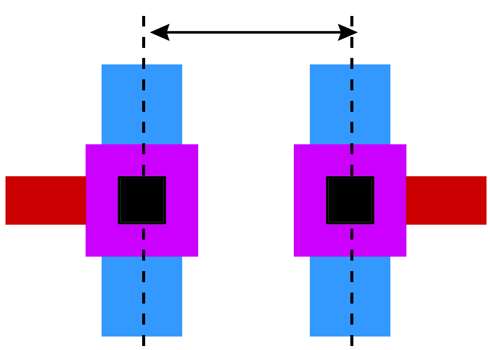


Lambda vs. Micron Rules

Lambda-based design rules are based on the assumption that one can scale a design to the appropriate size before manufacture.

Consider the following data from *Weste*, Table 3.2:

contacted metal pitch	***lambda rule***	***lambda = 0.5u***	***micron rule***
1/2 * contact size	**1λ**	**0.5μ**	**0.375μ**
contact surround	**1λ**	**0.5μ**	**0.5μ**
metal-to-metal spacing	**3λ**	**1.5μ**	**1.0μ**
contact surround	**1λ**	**0.5μ**	**0.5μ**
1/2 * contact size	**1λ**	**0.5μ**	**0.375μ**
	8λ	**4μ**	**2.75μ**



Scaled design is legal but much larger than it needs to be!

Retargetable Layouts?

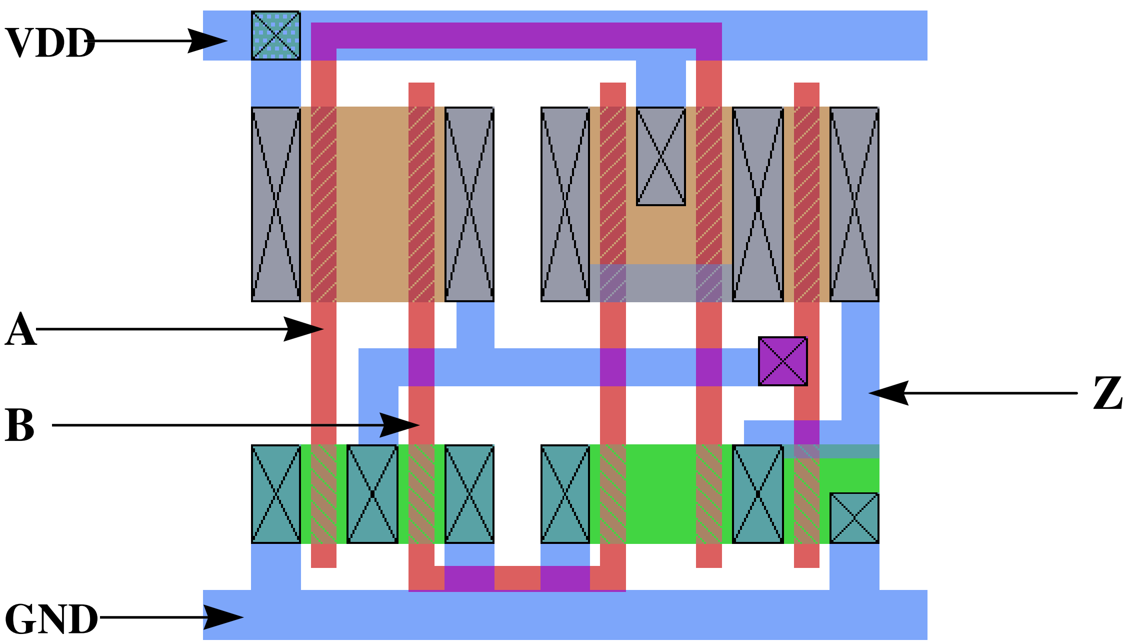
So, should one use lambda rules, or not?

- **probably okay for retargetting between "similar" processes, e.g., when later process is a simple "shrink" of the earlier process. Some 0.35u processes are shrinks of an earlier 0.5u process.**

- **most industrial designs use micron rules to get the extra space efficiency. Cost of retargeting by hand is acceptable for a successful product, but usually it's time for a redesign anyway.**

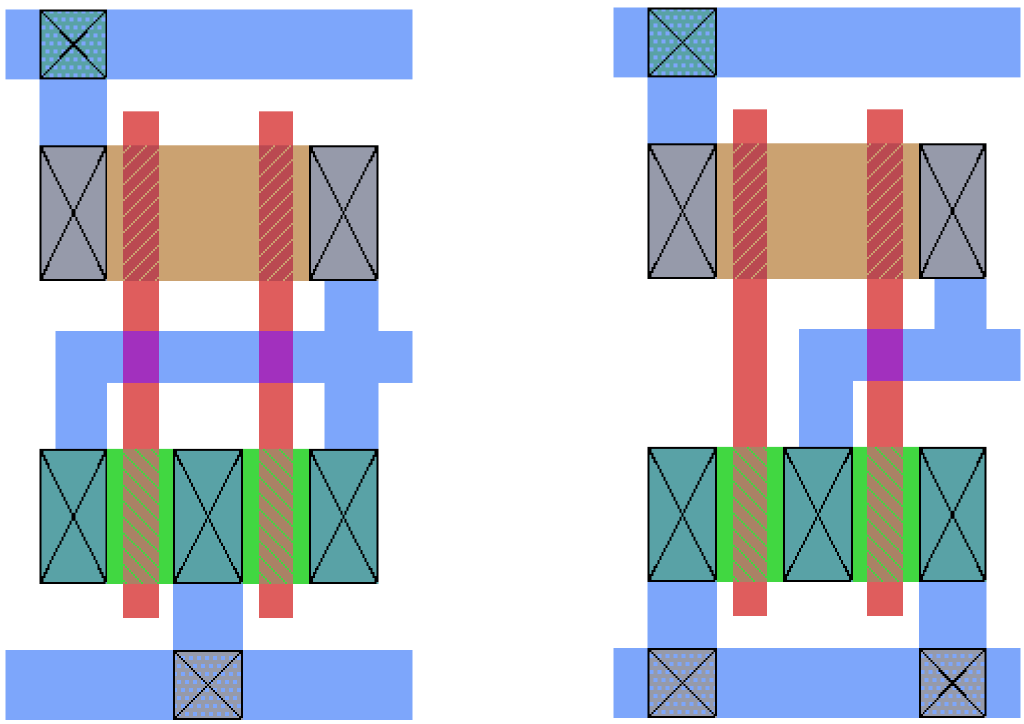
- **invent some way of entering a design symbolically but use a more sophisticated technique for producing the masks for a particular process. Insight: relative sizes may change but topological relationship between components does not. So, instead of shrinking a design, *compact* it!**

CMOS Logic Gate



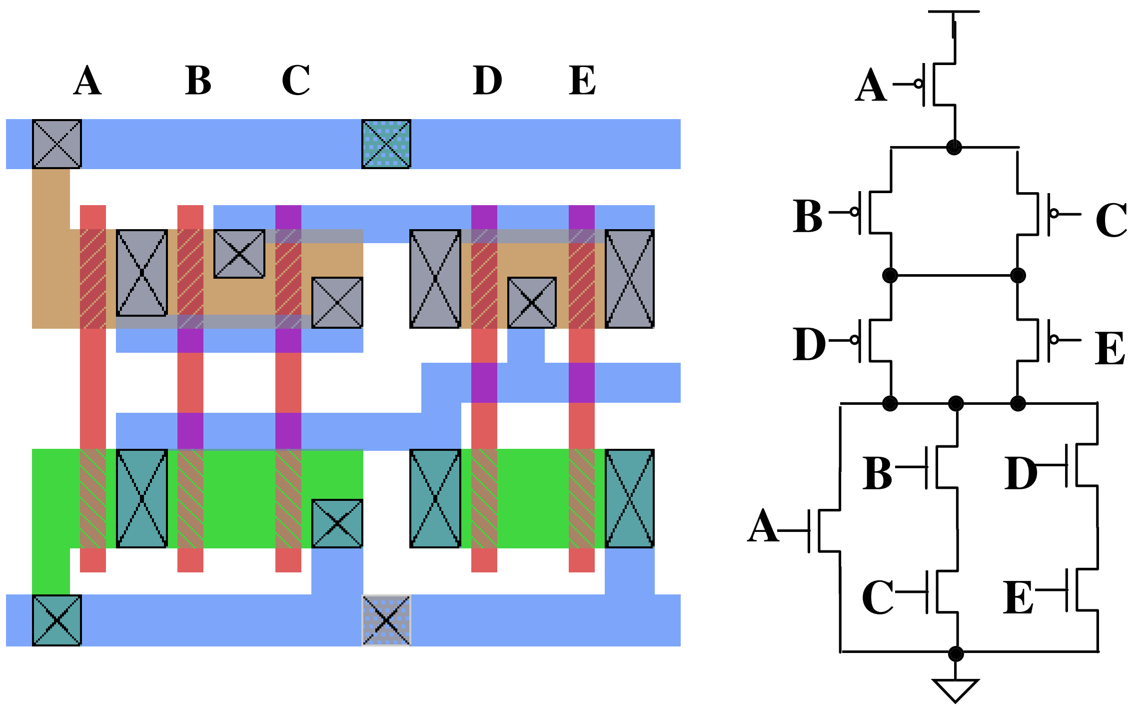
Schematic:

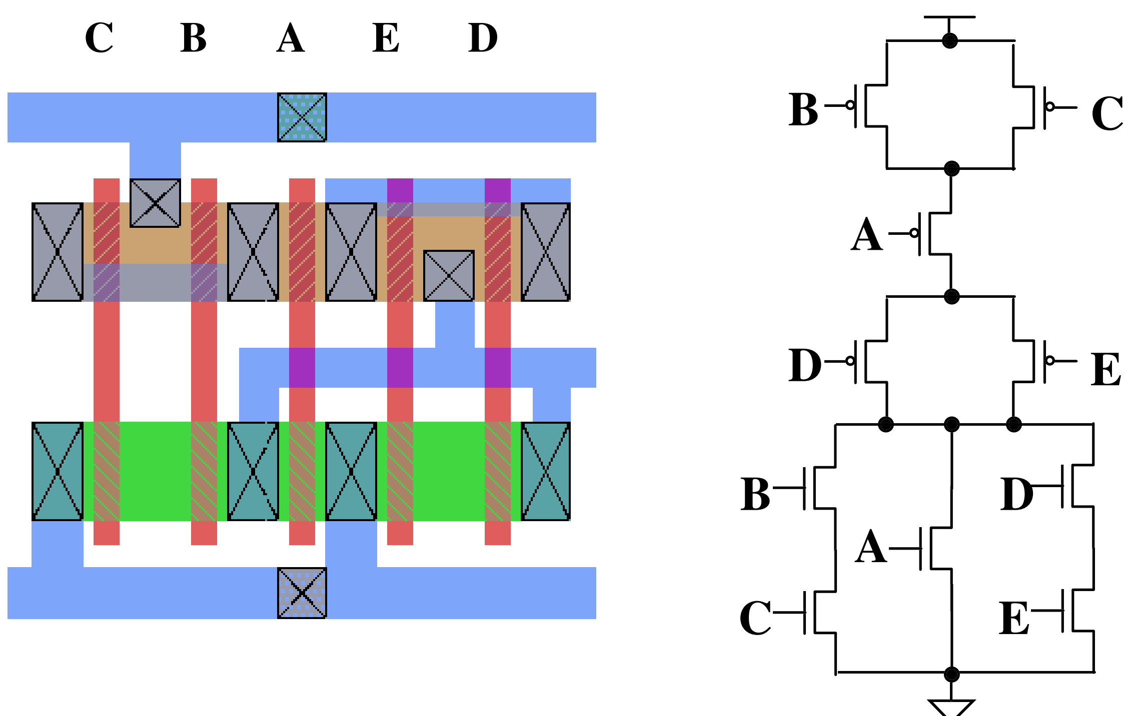
Optimizing connections



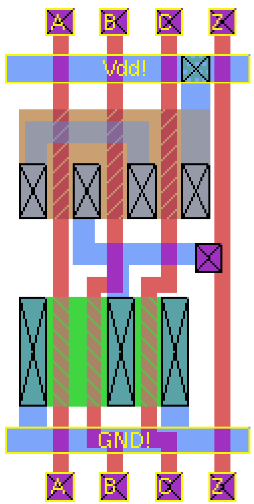
Which is the better gate layout?

Eliminating Gaps

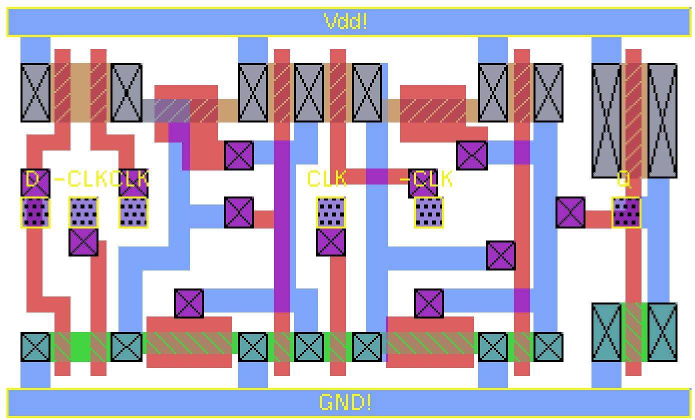




Standard Cells Routing

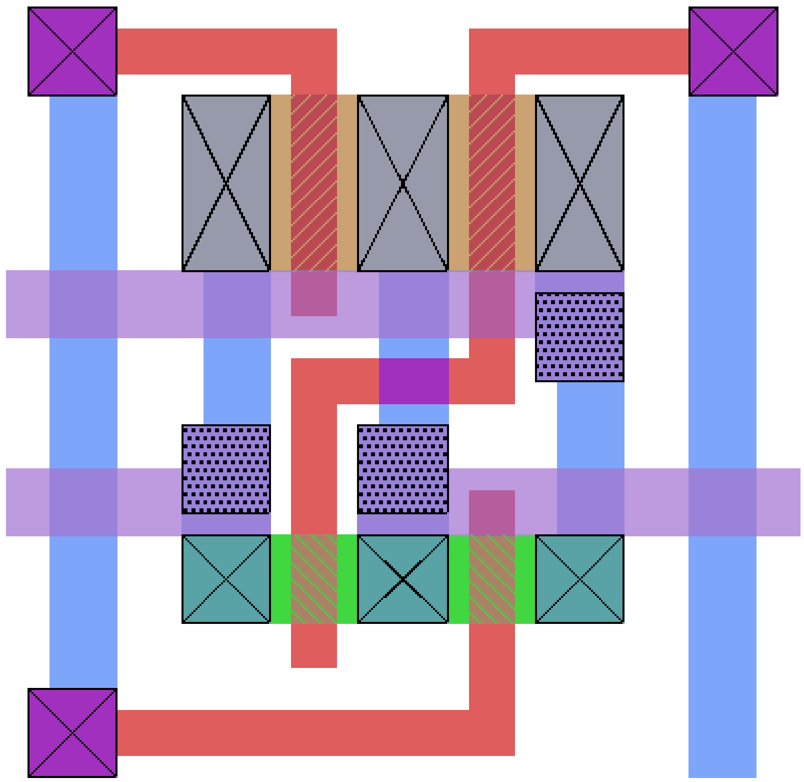


routing outside cell



routing over cell

Replicating Cells

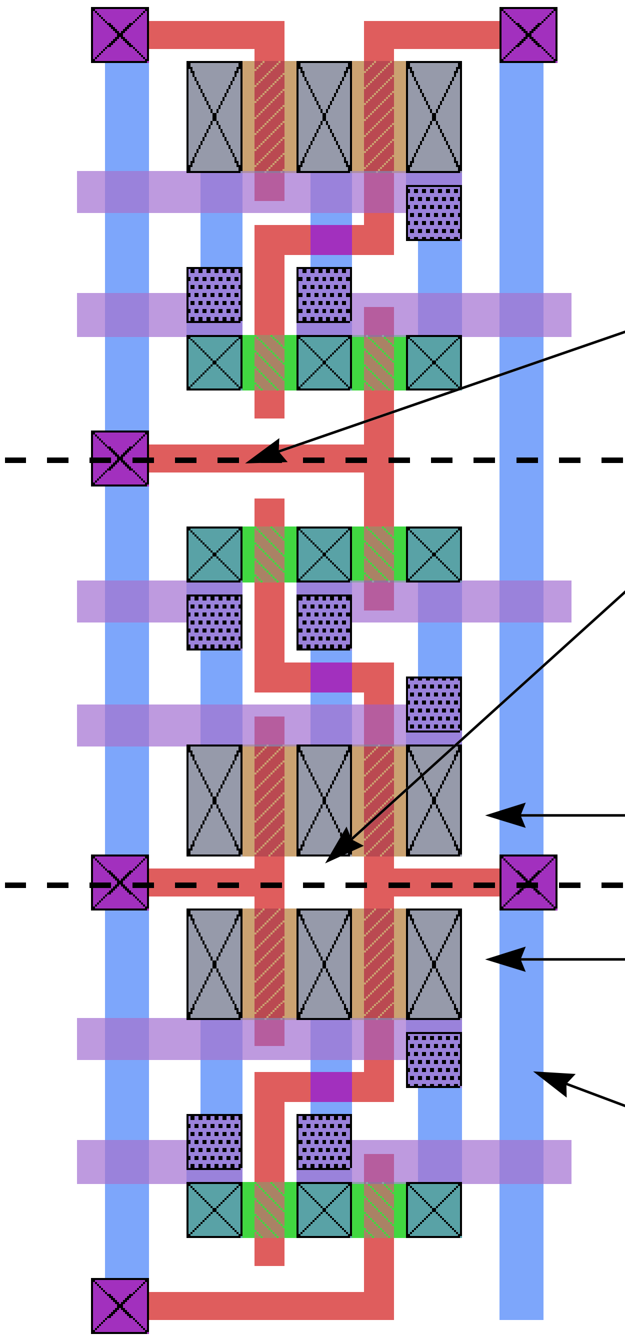


What does this cell do?

What if we want to replicate this cell vertically, i.e., make a stack of the cells, to process many bits in parallel?

- **what nodes are shared among the cells?**
- **what nodes aren't shared?**
- **how should we arrange the cells vertically?**

Vertical Replication



Place shared geometry symmetrically about shared boundary.

Place items that aren't to be shared 1/2 min spacing rule from shared boundary.

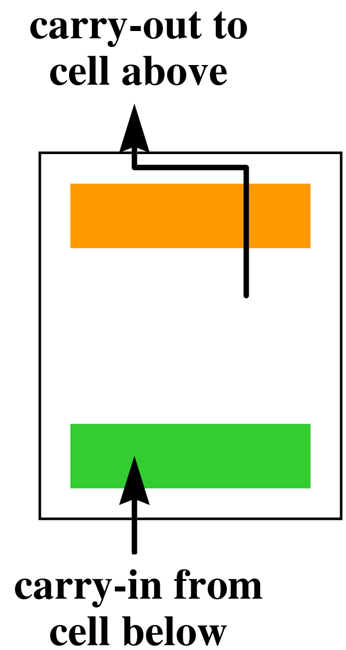
Reflect cell about X axis so that Pfets are next to each other: this avoids large ndiff/pdiff spacing.

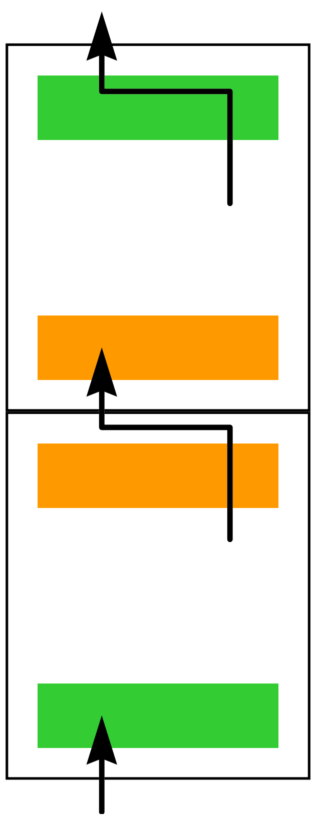
Run shared control signals vertically -- they'll wire themselves up automatically?

Vertical Intercell Routing

Suppose we have a signal that will run vertically from one cell to the next, e.g., the carry-out from one cell becomes the carry-in for the cell above.

Looks okay until we reflect the cell when we do the vertical replication!

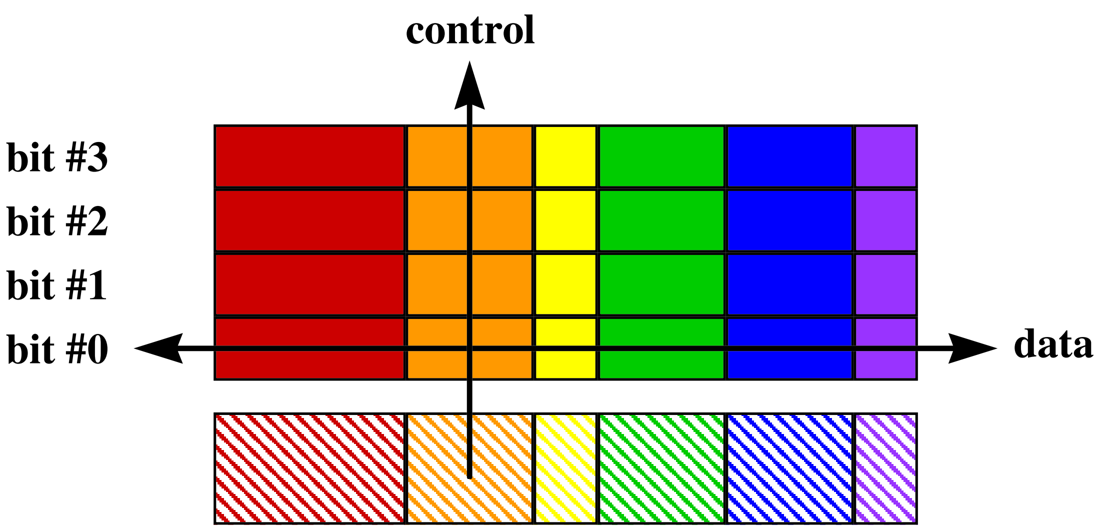




Solution: we have to do the routing for vertical intercell signals for a pair of cells, then replicate the pair (complete with routing) vertically.

Building a Datapath

It's often the case that we want to operate on many bits in parallel. A sensible way to arrange the layout of this sort of logic is as a datapath where data signals run horizontally between functional units and control signals run vertically to all the bits of a particular functional unit:



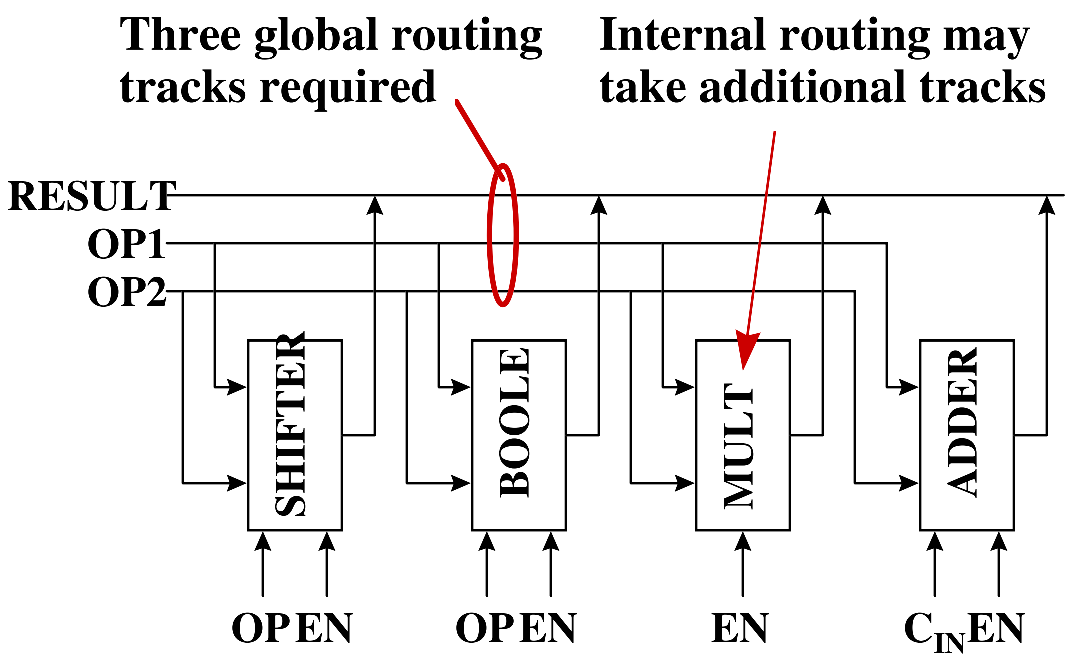
Logic that generates the control signals can be placed at the bottom of the datapath. If control logic is complicated or irregular, it might be placed in a separate standard cell block and only the control signal buffers placed placed just below the datapath. Although it's tempting to run control signals in poly (so they can control fets) this is unwise for tall datapaths because of poly resistance (e.g., 32 bits x 20u/bit = 640u = ~1000 squares = ~20k ohms!)

Datapath Bit Pitch

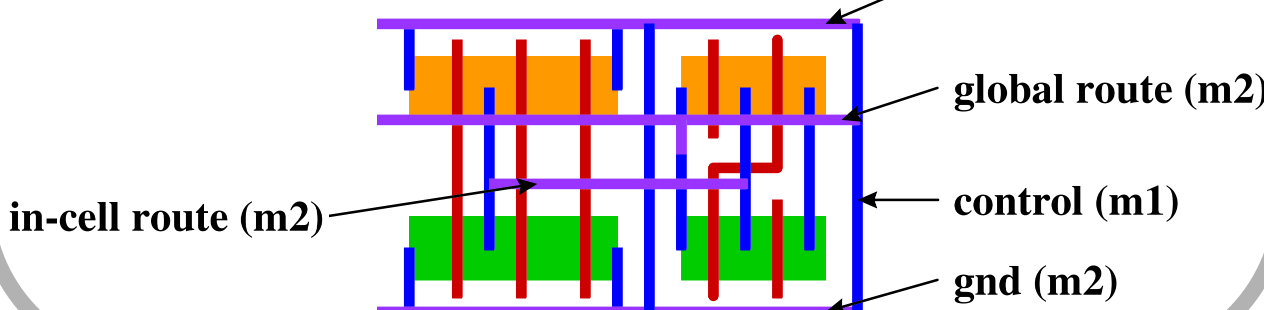
How tall should we make each bit of the datapath?
That depends on

- the width of the nfets and pfets
- how much in-cell routing there is
- how much over-the-cell global routing there is

Global routes can be determined from datapath schematic:



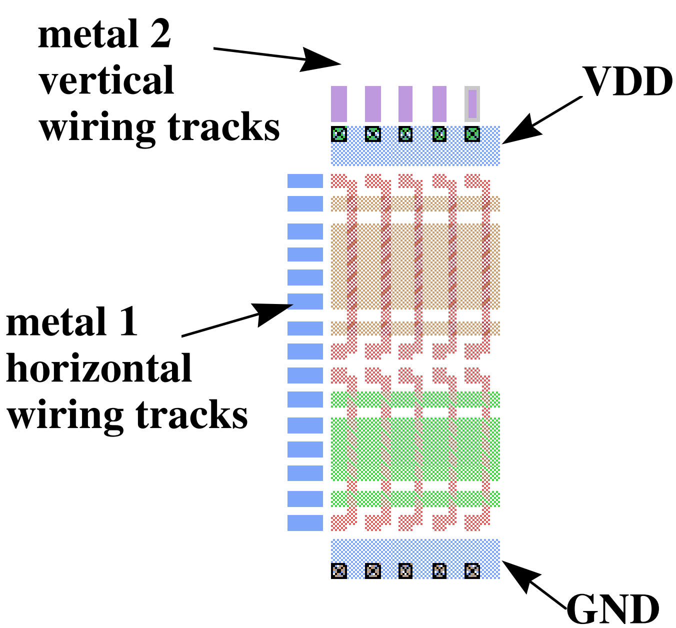
Cell routing plan:



Gate Arrays

A gate array uses a fixed array of transistors which is "personalized" by adding metal wires and contacts. Most popular architecture is Sea-of-Gates where the core of the chip is a continuous array of fets. Routing occurs over the top of unused transistors. Pros:

- **base wafers are common and can be made in volume**
- **only 2-5 masks needed (<$)**
- **standard packaging**
- **design can be highly automated**
- **short processing time**

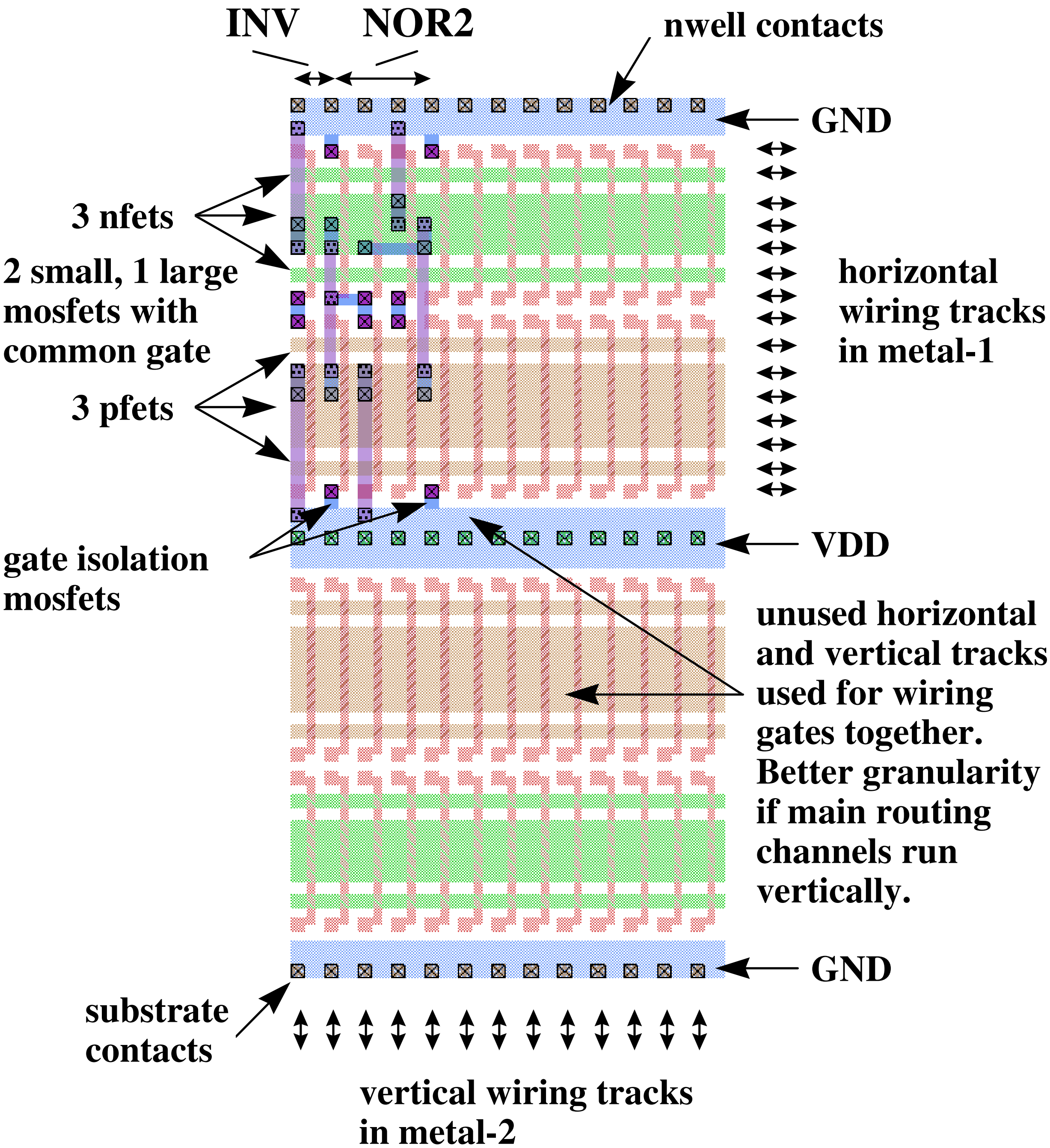


Sea-of-gates layout using "gate-isolation" where nfets with gates tied to GND and pfets with gates tied to VDD isolate adjacent logic gates.

This layout includes both small and large nfets/pfets so designer has more flexibility in PLA, ROM and RAM structures, ...

from IEEE JSSC, V25, No5, Oct 1990

SOG Example


INV
NOR2
nwell contacts
GND
3 nfets
2 small, 1 large mosfets with common gate
horizontal wiring tracks in metal-1
3 pfets
gate isolation mosfets
VDD
unused horizontal and vertical tracks used for wiring gates together. Better granularity if main routing channels run vertically.
GND
substrate contacts
vertical wiring tracks in metal-2

Checking Layouts

Design Rule Checker (DRC). This is a program that checks each piece of the layout against the process design rules. This is a slow process:

- **canonicalize layout into a set of leading and trailing non-overlapping mask edges. Some boolean mask operations may be needed.**
- **determine electrical connectivity and label each edge with the node it belongs to.**
- **test each edge end point against neighboring edges to check for spacing (leading edges) and width (trailing edges) violations.**

Layout vs. Schematic (LVS). First a netlist is extracted from the layout. Use the electrical info generated by the DRC and then recognize transistors are juxtapositions of channel with diffusion. Then see if extracted netlist is isomorphic to the schematic netlist. This is done by a coloring algorithm:

- **initialize all nodes to the same color**
- **compute a new color for each node as some hashing function involving the colors of connected (ie, thru a fet) nodes.**
- **nodes that have a unique color are isomorphic to similarly colored node in other network**
- **worry about parallel fets, ambiguous nodes**