MOS: Metal-Oxide-Silicon

Metal gate has been replaced by **polysilicon** or **poly** in today's processes.

Poly is used as a mask to allow precise definition of the source and drain regions.

Minimizes gate-to-source/drain overlap which is good for performance.

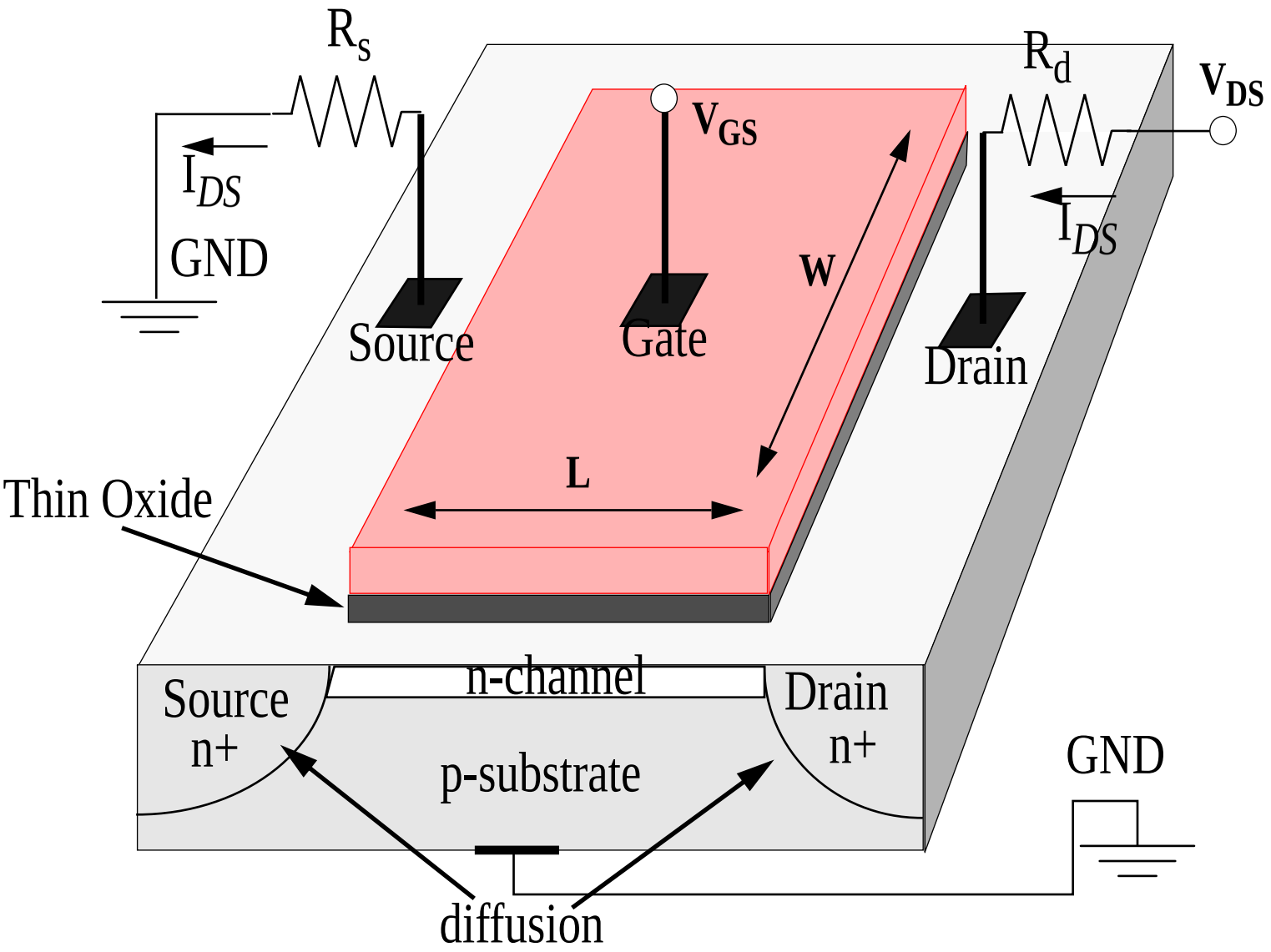
MOS structure created by superimposing several layers of conducting, insulating and transistor-forming materials.

Construction process is carried out on a *SINGLE* crystal of silicon.
Wafers are 15-20 cm in diameter (6-8 inches).

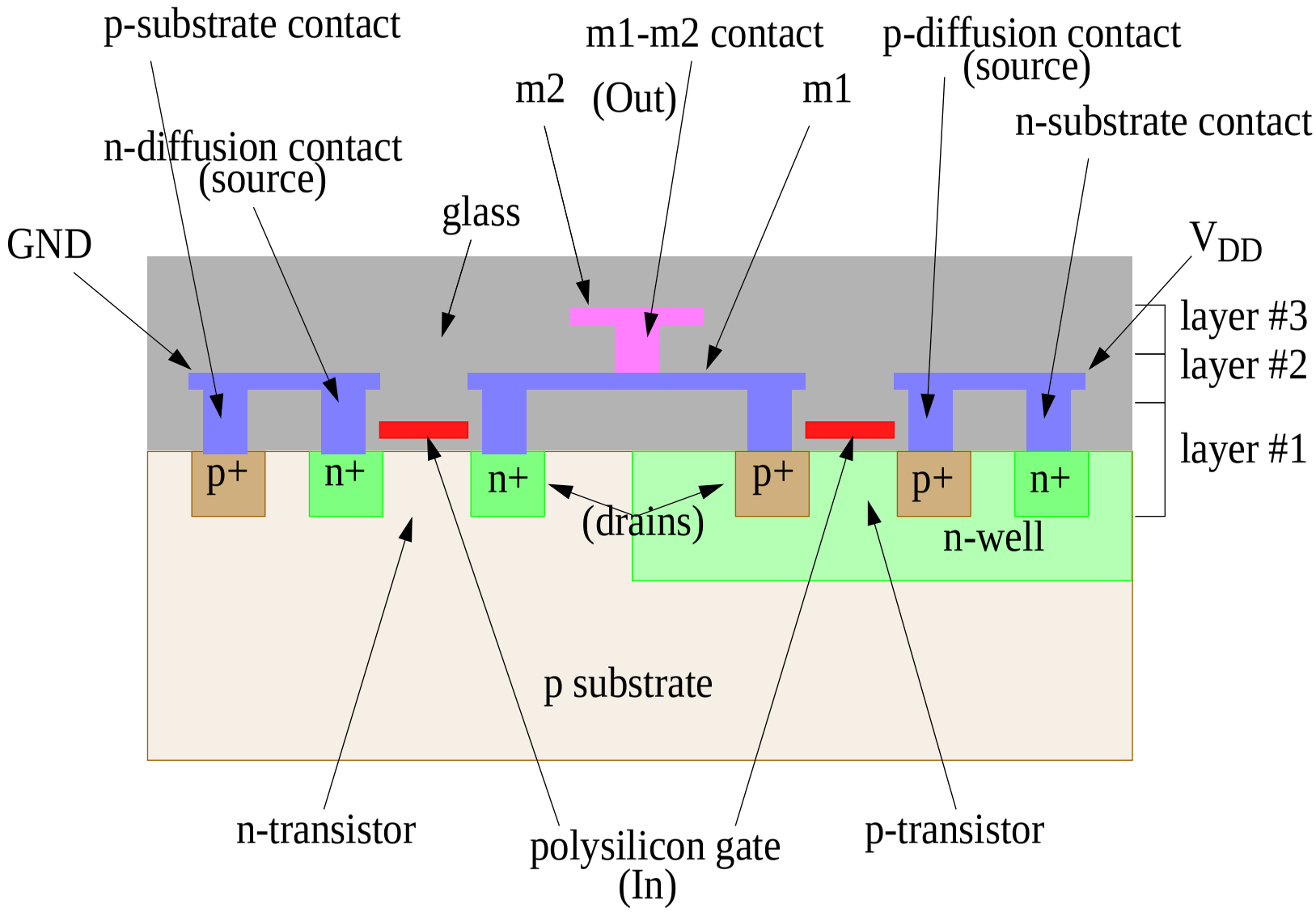
CMOS: Two types of transistors are used, *pMOS* and *nMOS*.

nMOS: negatively doped silicon, rich in electrons.
pMOS: positively doped silicon, rich in holes (the DUAL of electrons).

An nMOS transistor



Inverter Cross-section



MOS Transistors as Switches

We can treat MOS transistors as simple on-off switches with a source (S), gate (G) (controls the state of the switch) and drain (D).

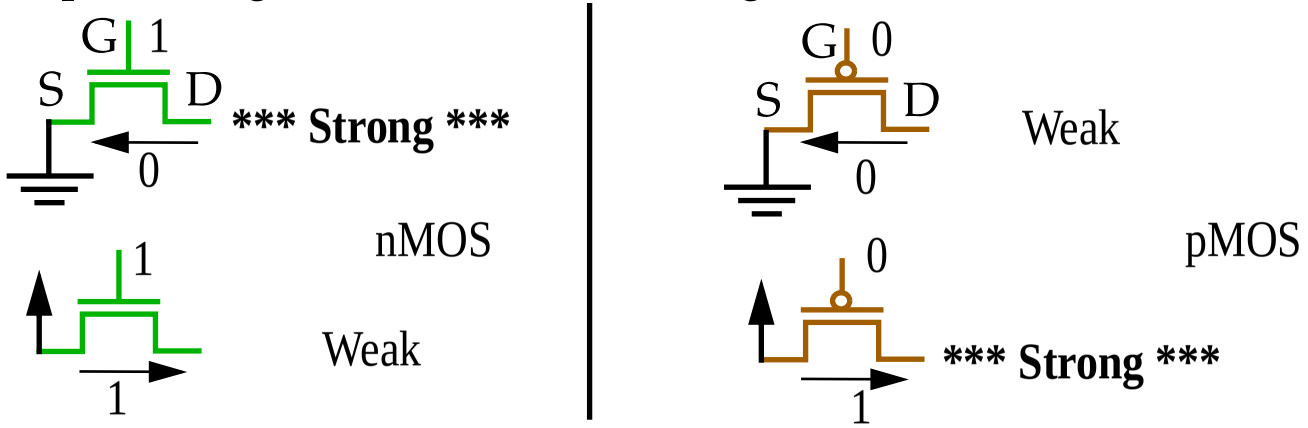
Let '1' represent high voltage: 1.5V to 15V (V_{DD} =< 3.3V today).

Let '0' represent low voltage: GND or V_{SS}.

Signals such as '1' and '0' have **strength**, measures their ability to:

- Sink (to lower voltage, e.g. GND) or
- Source (from higher voltage, e.g. V_{DD}) current.

nMOS and pMOS signal transmission strength:



MOS Transistor Switches

The reason p-transistors are *poor* transmitters of logic 0 and n-transistors are *poor* transmitters of logic 1 is related to threshold voltage (V_t ~= 700mV).

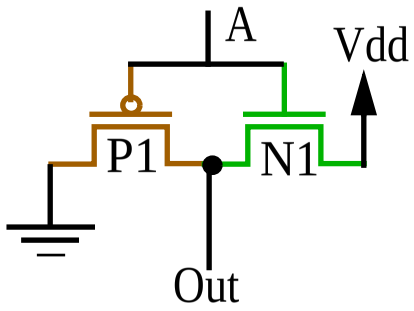
Threshold voltage will be discussed in detail soon.

Under the "switch" abstraction, G has complete control and S and D have no effect.

In reality, the gate can turn the switch on only if a potential difference of at least V_t exists between the G and S.

This is clearly not the case for the "weak" bias configurations and "weak" 0s (~700mV) and "weak" 1s (~4.3V) result.

Therefore, the following (buffer) implementation is a bad idea.

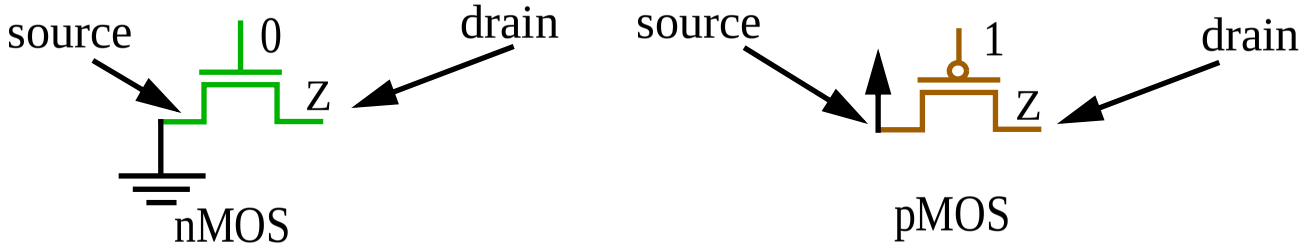


BAD IDEA

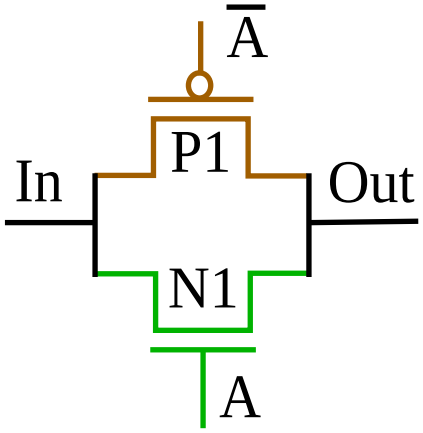
MOS Transistor Switches

The off state of a transistor creates a high impedance condition Z at the drain.

No current flows from source to drain:



Complementary Switch or Transmission gate or Pass Gate:



One pMOS and one nMOS in parallel.

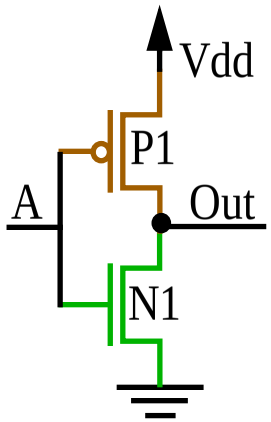
Note that neither transistor is connected to V_{DD} or GND.

A and $\overline{A}$ control the transmission of a signal on *In* to *Out*.

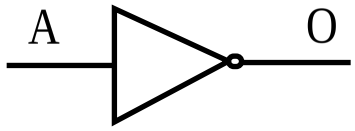
This configuration allows '1's and '0's to be passed in an acceptable fashion.

When A = '0', *Out* is in a high impedance state (not driven by *In*).

The CMOS Inverter



CMOS Inverter



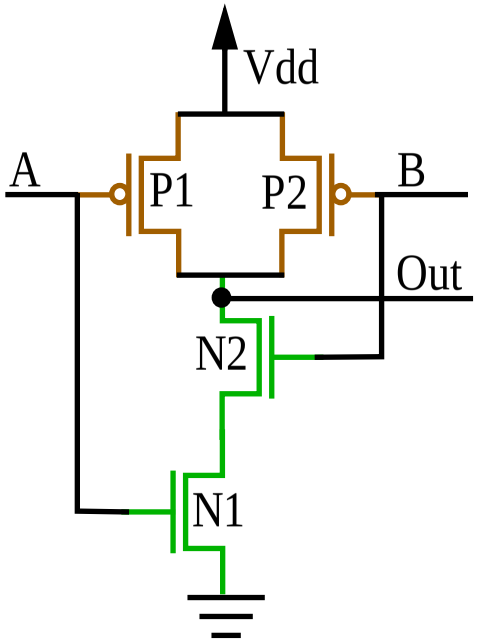
A	O
0	1
1	0

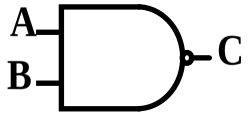
If the gates of transistors P1 and N1 are not connected, then 4 possible output states are possible.

What are the two additional states?

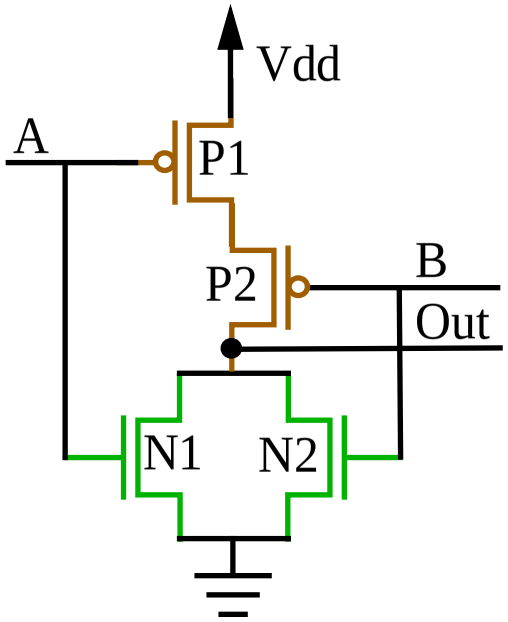
Are any of these states undesirable? Why?

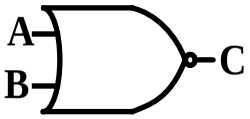
NAND and NOR CMOS Gates





A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

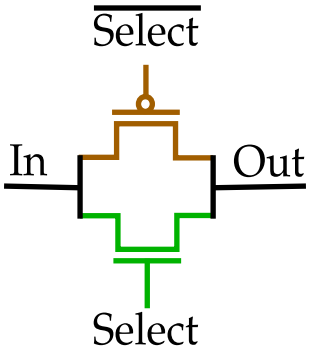




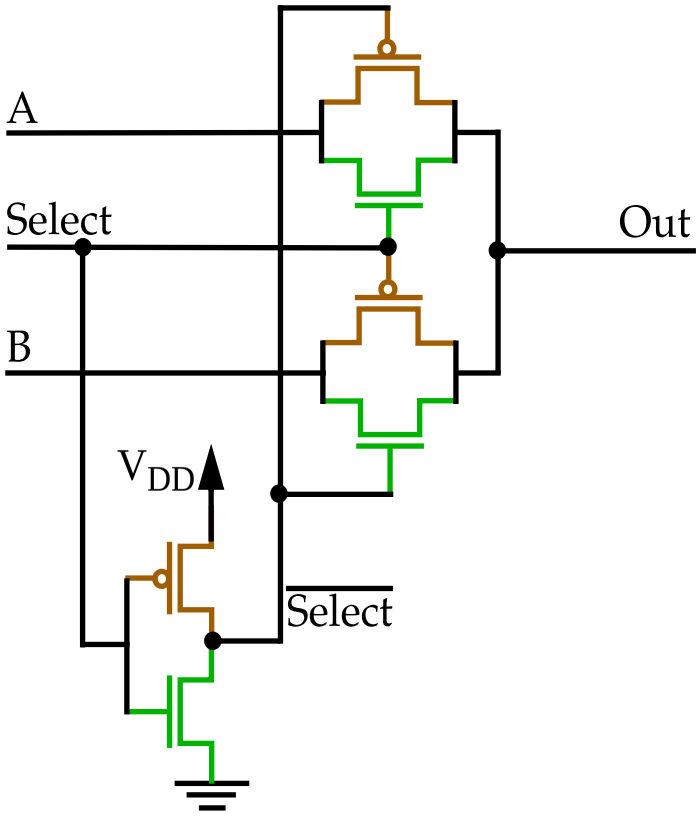
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Pass Gates Applications: Select Mux

Transmission Gate



2-to-1 MUX



Truth Table for 2-to-1 MUX

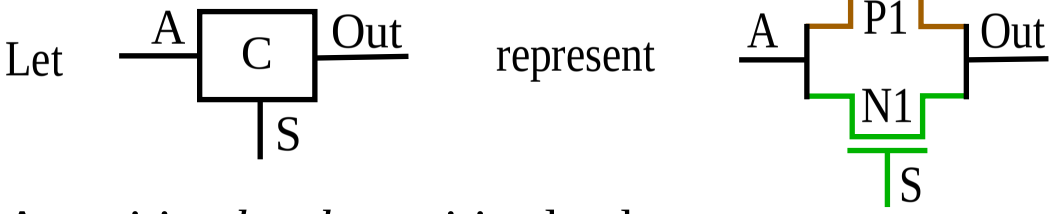
Select	Out
0	B
1	A

$Out = A.S + B.\overline{S}$

How would you implement this function using logic gates instead of CMOS switches?

Pass Gates Applications: Latches and Registers

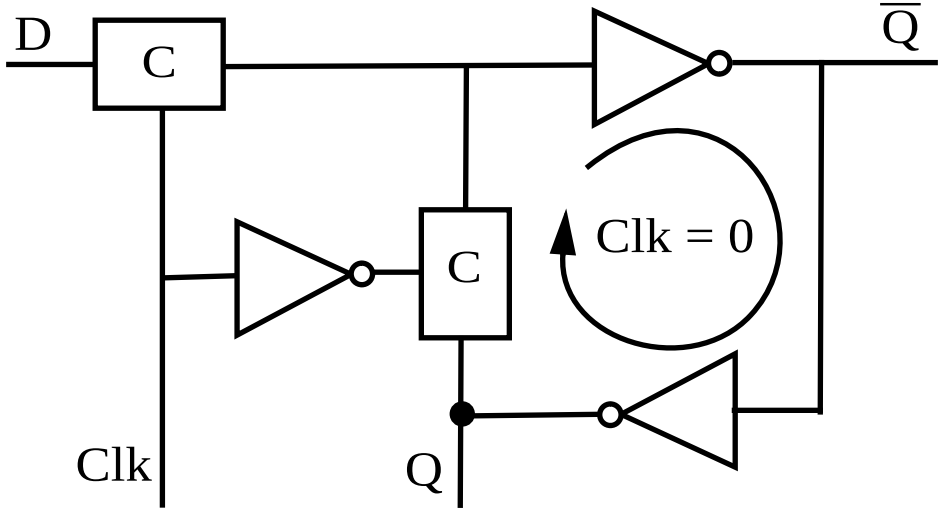
The D latch:



Although $\overline{S}$ is not given in the "black box" abstraction, it must be routed to the pass gate.

The S "black box" terminal connects to the n-channel device.

A positive *level-sensitive* latch:



When Clk = '1', Q set to D and $\overline{Q}$ set to $\overline{D}$

When Clk = '0', D is ignored, Feedback path is established.

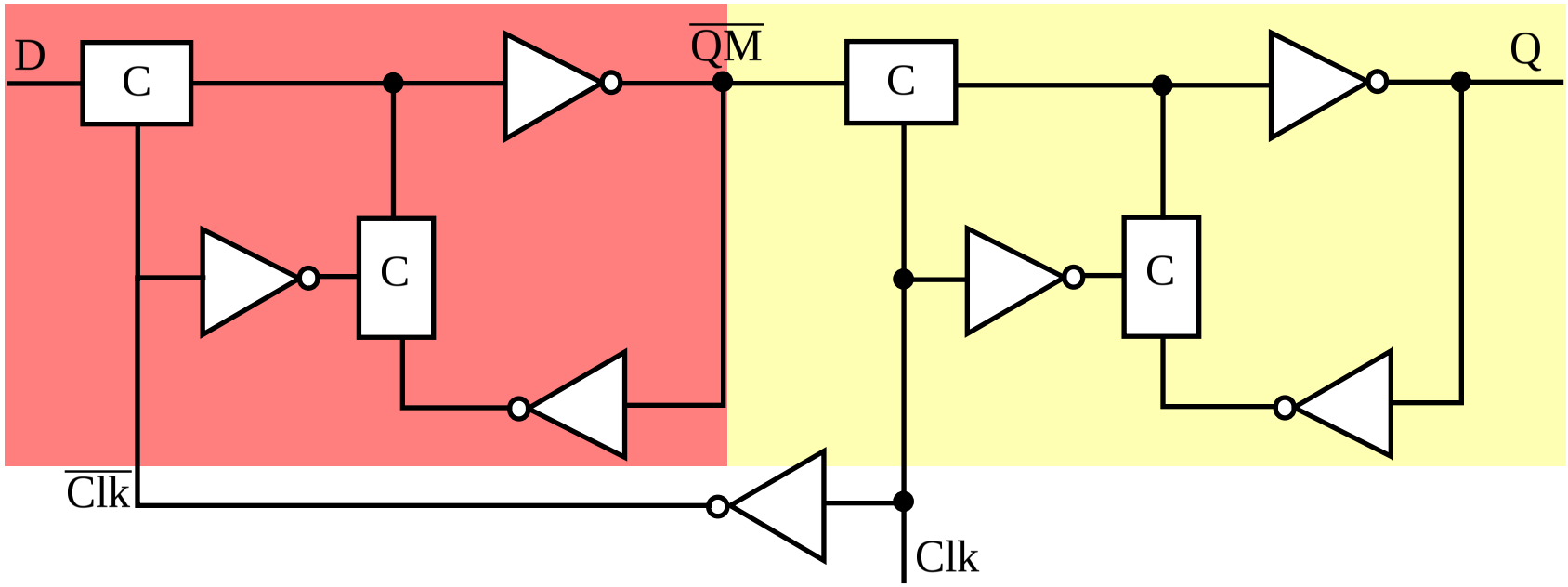
State of the output is dependent on the level of the clock.

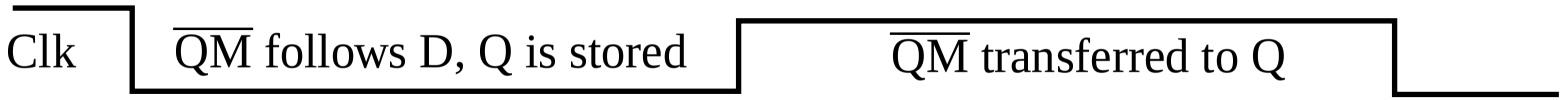
Note: Other notations for $\overline{D}$: -D or DN or D.L.

Pass Gates Applications: Latches and Registers

Master-Slave D Flip-Flop:

Combine one negative (master) and one positive (slave) *level-sensitive* latch.

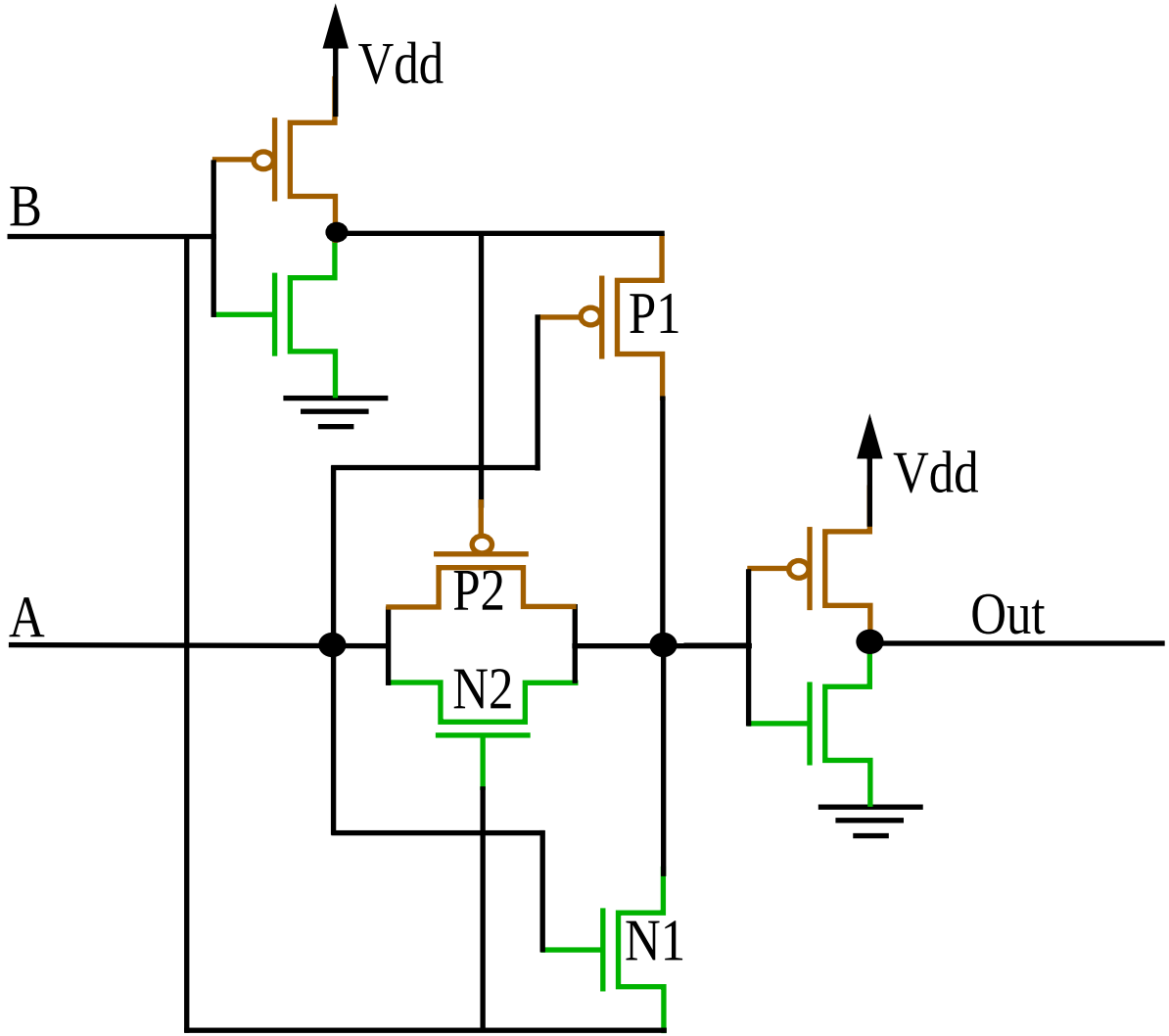




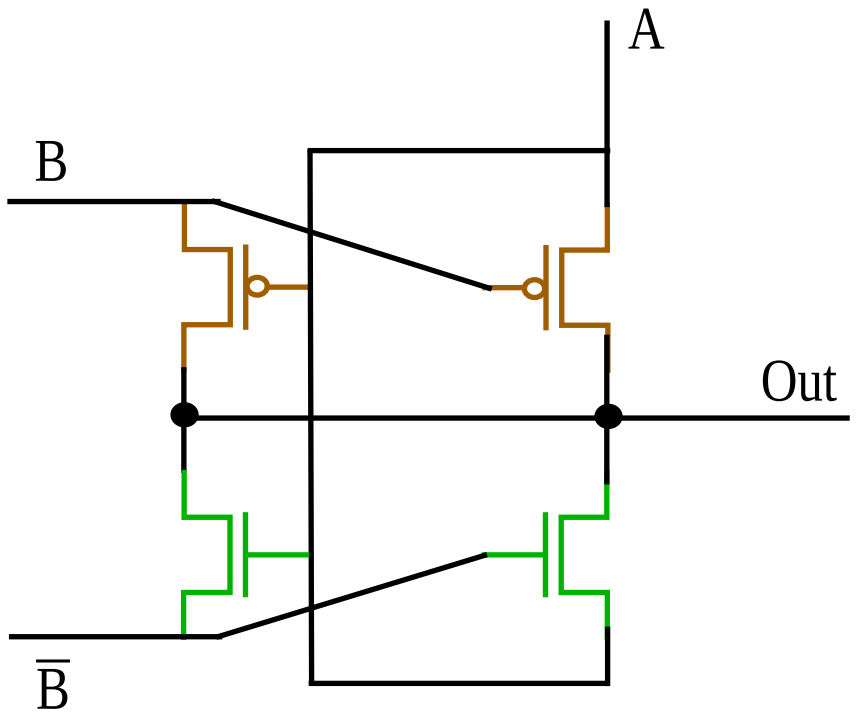
Forms the basis of most CMOS storage elements (EXCEPTIONS: RAM and ROM).

We will look at memory elements in more detail later.

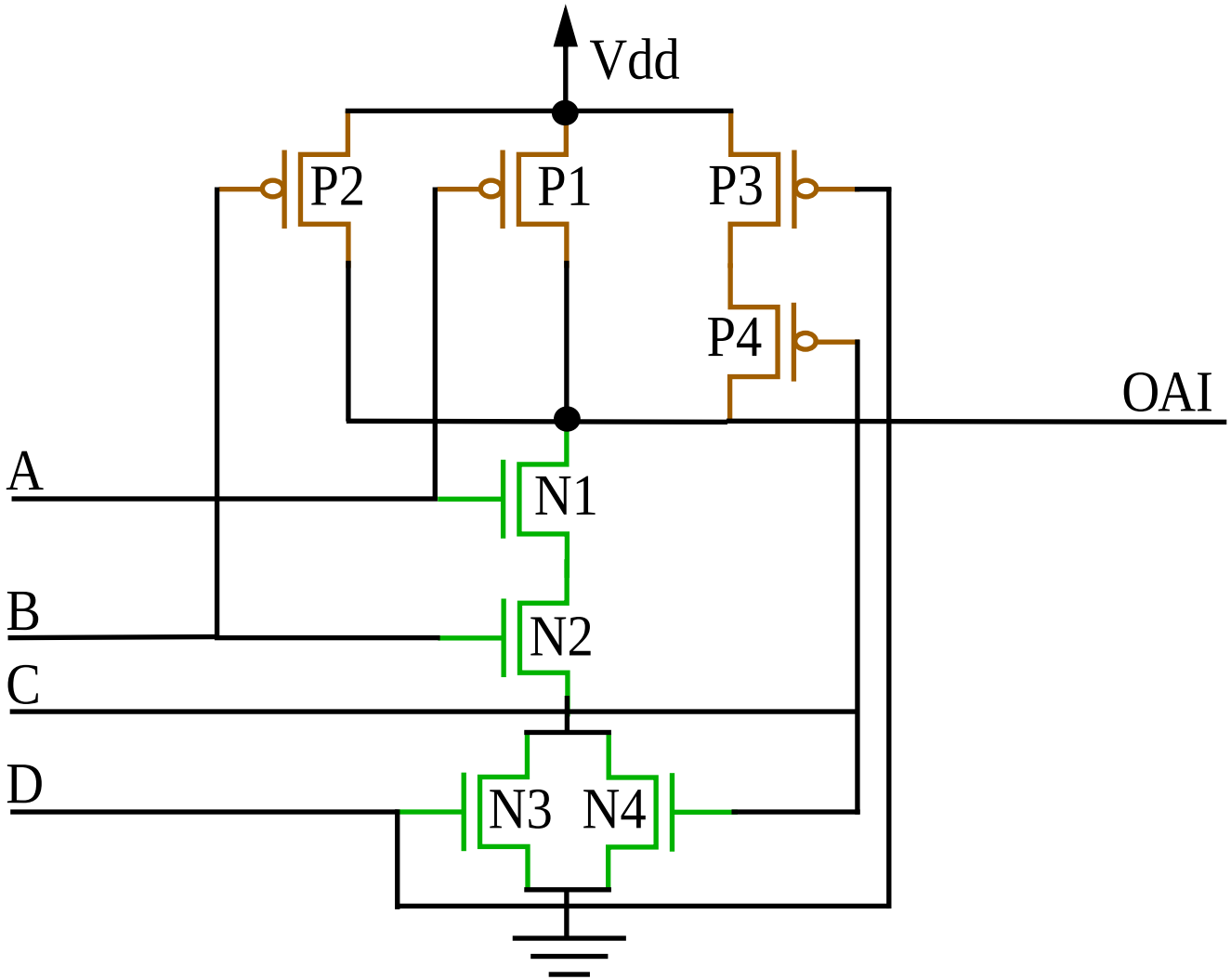
More CMOS Gates



And More CMOS Gates



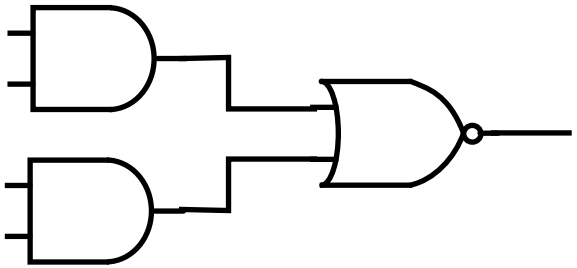
And More CMOS Gates



AOI and OAI

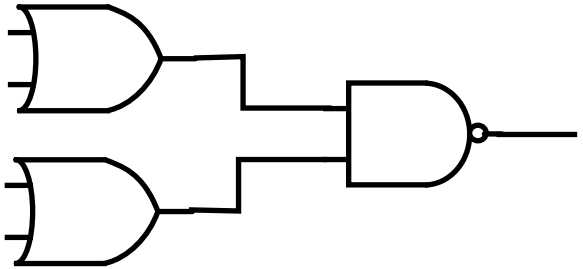
$AOI = \overline{(\ .\ ) + (\ .\ )}$ Sum of Products

Disjunctive Normal Form



$OAI = \overline{(\ +\ ) . (\ +\ )}$ Product of Sums

Conjunctive Normal Form



Building CMOS logic gates from expressions:

How do we build ?

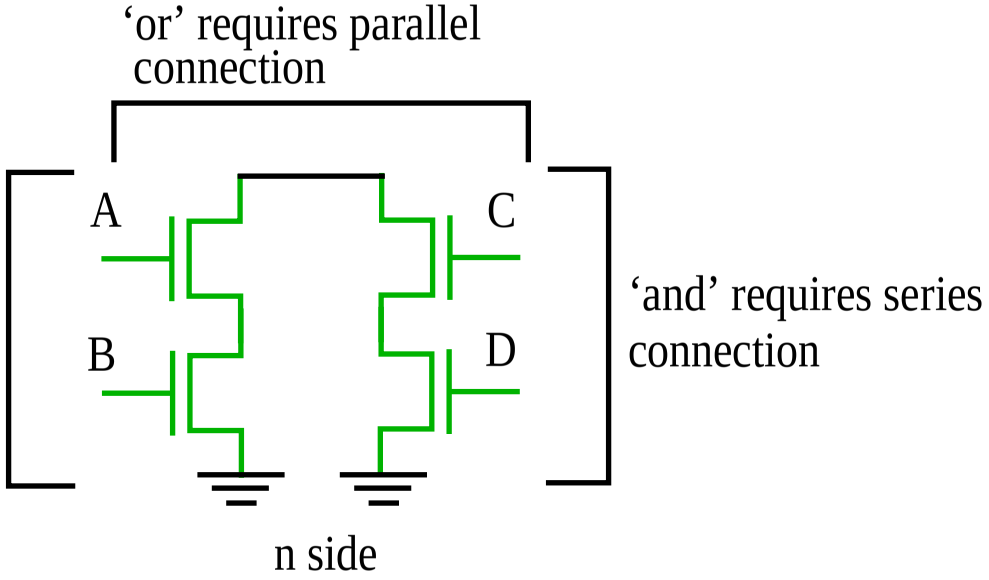
$F = \overline{((A.B)+(C.D))}$

For the n-side, take the uninverted expression (the complement of F, e.g., $\overline{F}$):

$F = ((A.B)+(C.D))$

'and' expressions are implemented using series connections of n transistors.

'or' expressions are implemented using parallel connections of n transistors.



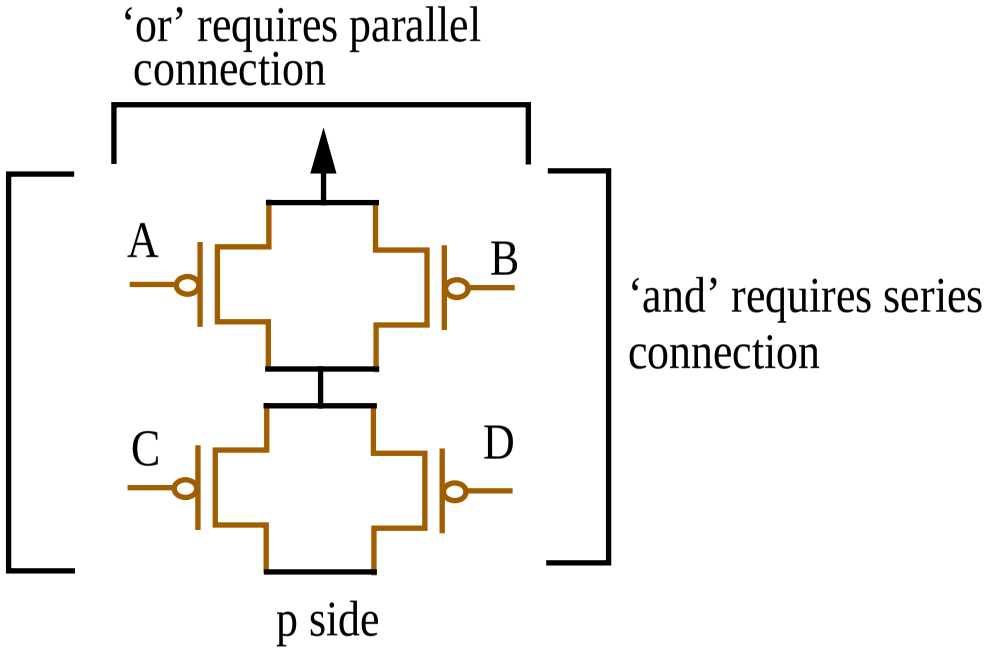
Building CMOS logic gates from expressions:

For the p-side, invert expression used for n-expansion:

$((\overline{A}+\overline{B}).(\overline{C}+\overline{D}))$

'and' expressions are implemented using series connections of p transistors.

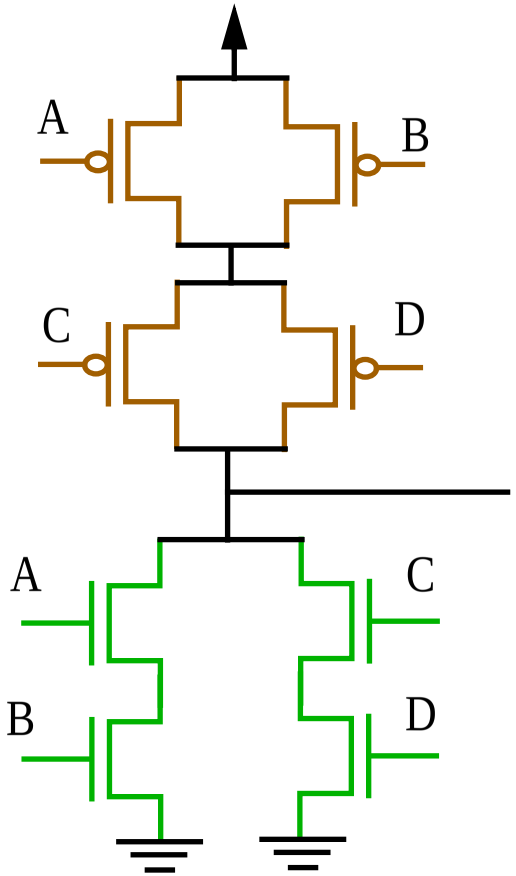
'or' expressions are implemented using parallel connections of p transistors.



Building CMOS logic gates from expressions:

Combine to build function:

$$F = \overline{((A.B)+(C.D))}$$



Try building

$$F = \overline{((A+B+C).D}$$

Other Transformations

You must master all of the following transformations between levels of abstractions:

1) The previous analysis shows how to take a Boolean expression and create a transistor-level schematic diagram.

However, it assumes the Boolean expression is already in the appropriate form, which may not always be the case.

Boolean expression reduction:

You should already know how to manipulate boolean expressions, e.g., using De Morgan's Laws, from exercises in other courses.

The objective is to reduce a boolean expression so that it can be realized in full-complementary CMOS using the minimum number of transistors.

Other Transformations

In general, I am **not** expecting you to realize CMOS gates using *pass structures* in which the inputs are used to drive the output of the gate.

The XOR/XNOR implementations we saw earlier are examples of this.

The following heuristics can be applied as target reductions that will help you to obtain minimum realizations:

- Since CMOS is naturally inverting, you'll want to target a final expression of the form:

$$F = \overline{(\text{expression})}$$

- Many times only **un**complemented literals are available as signals in your circuit. Therefore, the reductions should attempt to **remove** the complemented literals in the Boolean expression.

  Application of De Morgan's Laws can be used to transform complemented literals to NANDs and NORs.

- You should analyze each transformation to learn the trade-offs.

Other Transformations

For example:

$F = (\bar{A}B) + (C + D)E$ Build inverse? 14 transistors

The following reduction sequence can be applied that targets NANDs and removes the complemented literals:

$\bar{F} = \overline{(\bar{A}B) + (C + D)E}$ Invert both sides.

$\bar{F} = \overline{\bar{A}B} \bullet \overline{(C + D)E}$ How many transistors are needed here?

$\bar{F} = (A + \bar{B}) \bullet \overline{(C + D)E}$ Build here?:

$F = \overline{(A + \bar{B}) \bullet \overline{(C + D)E}}$

transistors: 6 for OAI, 2 for inverter for B, 6 for final OAI.

$\bar{F} = A\overline{(C + D)E} + \bar{B}\overline{(C + D)E}$ Multiply.

$\bar{F} = A\overline{(C + D)E} + \overline{B + (C + D)E}$

$F = \overline{A\overline{(C + D)E} + \overline{B + (C + D)E}}$ Or Build here?: 6 for OAI, 8 for B AOI, 6 for final AOI.

Other Transformations

Note that further reductions to NANDs and NORs may not pay off, as in the previous case.

In the next case, it is possible to get rid of an complemented literal without increasing the size of the OAI:

$$F = (\bar{A}B) + (C + D)\bar{E}$$

$$\bar{F} = \overline{(\bar{A}B) + (C + D)\bar{E}}$$ Invert both sides.

$$\bar{F} = \overline{\bar{A}B} \bullet \overline{(C + D)\bar{E}}$$ Apply DeMorgan's Laws.

$$\bar{F} = (A + \bar{B}) \bullet \overline{(C + D)\bar{E}}$$ Build here?: $\overline{F = (A + \bar{B}) \bullet \overline{(C + D)\bar{E}}}$

transistors: 6 for OAI, 4 for inverter for B, 6 for final OAI.

$$\bar{F} = (A + \bar{B})(\overline{C + D} + E)$$ Or Build here?: 4 for NOR, 2 for inverter, 8 for final OAI.

Further transformations are not useful -- convince yourself.

Other Transformations

Expressions with repeated variables may be simplified to save a couple transistors, in some cases:

$$F = A\overline{BC} + \bar{A}CD$$

$$\bar{F} = \overline{A\overline{BC} + \bar{A}CD} \qquad 4 + 2 + 10 + 2$$

$$\bar{F} = (\bar{A} + BC)(A + \overline{CD}) \qquad 2 + 4 + 10$$

$$\bar{F} = \bar{A}A + \bar{A}\overline{CD} + ABC + BC\overline{CD}$$

$BC\overline{CD}$ is redundant (covered) by the other terms, e.g,

$$\bar{F} = \bar{A}\overline{CD} + BC(A + \overline{CD}) = \bar{A}\overline{CD} + BC(A + \bar{A}\cancel{\overline{CD}})$$

$$\bar{F} = \bar{A}\overline{CD} + ABC \qquad 2 + 4 + 10$$

$$\bar{F} = (\overline{A + CD}) + ABC$$

$$F = \overline{(\overline{A + CD}) + ABC} \qquad 6 + 8 \text{ (*14*)}$$

Other Transformations

In contrast to:

$$F = (\bar{A}B) + (A + C)\bar{D}$$

$$\bar{F} = \overline{(\bar{A}B) + (A + C)\bar{D}}$$

$$F = \overline{\overline{(\bar{A}B)}\,\overline{(A + C)\bar{D}}}$$

6 for AB NAND, 8 for OAI, 4 for final NAND.

$$\bar{F} = (A + \bar{B})\overline{(A + C)\bar{D}}$$

$$F = \overline{(A + \bar{B})\overline{(A + C)\bar{D}}}$$

8 for OAI, 2 for inverted B, 6 for final OAI.

$$\bar{F} = (A + \bar{B})\overline{(A\bar{D} + C\bar{D})}$$

$$\bar{F} = (A + \bar{B})(\bar{A} + D)(\bar{C} + D)$$

$$\bar{F} = (A\bar{A} + AD + \bar{A}\bar{B} + \bar{B}D)(\bar{C} + D)$$

$$\bar{F} = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}D + AD\bar{C} + ADD + \bar{B}D\bar{C} + \bar{B}DD$$

$$\bar{F} = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}D + AD + \bar{B}D$$

$$\bar{F} = \bar{A}\bar{B}\bar{C} + D(A + \bar{B})$$

$$F = \overline{\overline{(A + B + C)} + D(A + \bar{B})}$$

6 for NOR, 2 for inverter, 8 for final OAI -- no better than the earlier expression.

Other Transformations

Sometimes it is best to implement the *inverse* function and add an inverter.

For example, Carry, which has all uncomplemented inputs.

$$\text{Carry} = AB + C_{in}(A + B)$$

$$\overline{\text{Carry}} = \overline{AB + C_{in}(A + B)}$$

What about XOR and XNOR?

$$F = A\bar{B} + \bar{A}B$$

$$\bar{F} = \overline{A\bar{B} + \bar{A}B}$$

$$F = \overline{(\bar{A} + B)(A + \bar{B})}$$

$$F = \overline{AB + \bar{A}\bar{B}}$$

$$F = \overline{AB + \overline{(A + B)}}$$

How many transistors are needed here?

The best way to learn this is through practice.

Simply make up an expression of multiple variables and invert a couple of the literals and/or subexpressions.

Other Transformations

2) Translating from **transistor-level schematics** to **Boolean expressions** is straightforward.

Simply write the *n-tree* expression using the rules for series and parallel transistors given earlier.

Invert the final expression.

3) Translating from **transistor-level schematic** diagrams to **layout** is covered in the laboratories.

4) Translating from **layout** to **transistor-level schematic** diagrams is also covered in the laboratories.

- In general, start by identifying the transistor sources connected to V_{DD} or GND nodes.
- Add series transistors in the schematic for transistors whose sources are connected to drains of the previously identified transistors.
- Add parallel transistors at fan-out points.
- Label the transistors so it possible to connect the gates properly by tracing the poly connections.