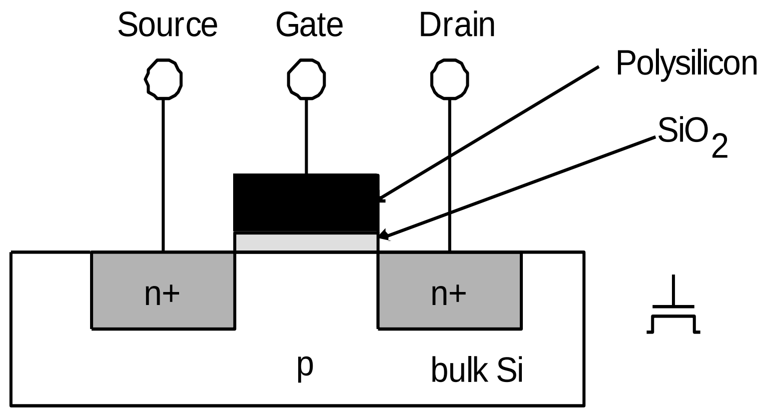
High Performance Computing Platform:
#5 Design of High Performance Calculation Units

Renyuan ZHANG

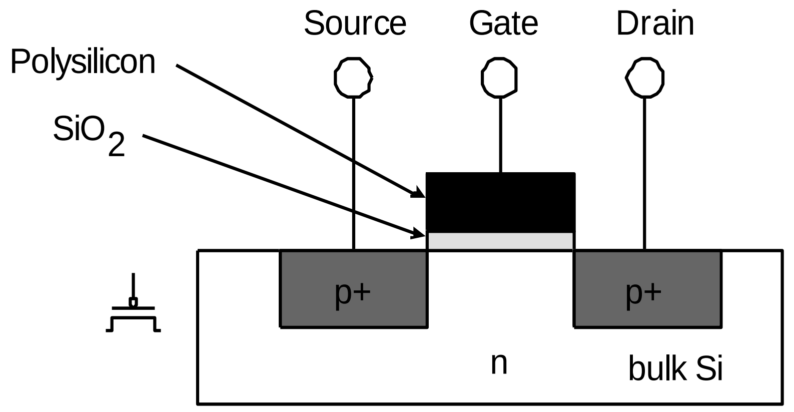
Comp-Arch-Lab, Division of Info. Sci., NAIST

MOS Transistors

- Four terminal device: gate, source, drain, body
- Gate – oxide – body stack looks like a capacitor
  - Gate and body are conductors (body is also called the substrate)
  - SiO_2 (oxide) is a “good” insulator (separates the gate from the body
  - Called metal–oxide–semiconductor (MOS) capacitor, even though gate is mostly made of poly-crystalline silicon (polysilicon)



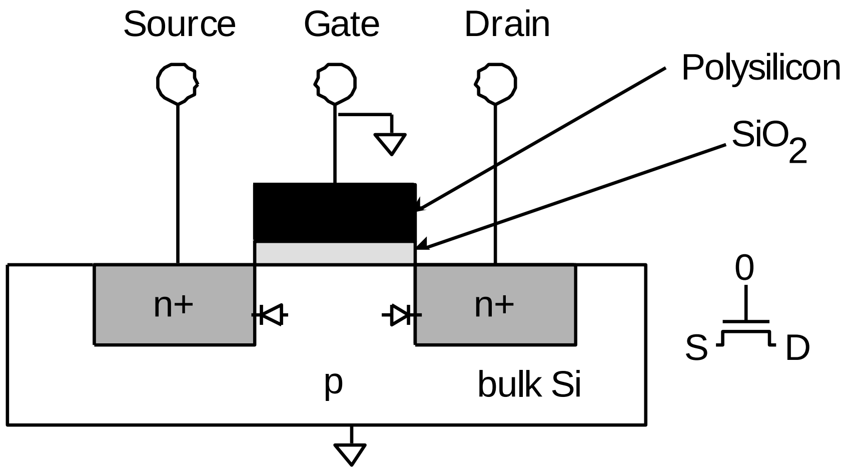
➢**NMOS**



➢**PMOS**

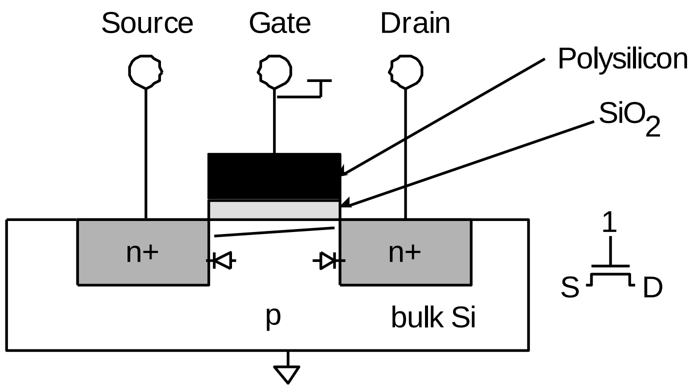
NMOS Operation

- Body is commonly tied to ground (0 V)
- Drain is at a higher voltage than Source
- When the gate is at a low voltage:
  - P-type body is at low voltage
  - Source-body and drain-body “diodes” are OFF
  - No current flows, transistor is OFF



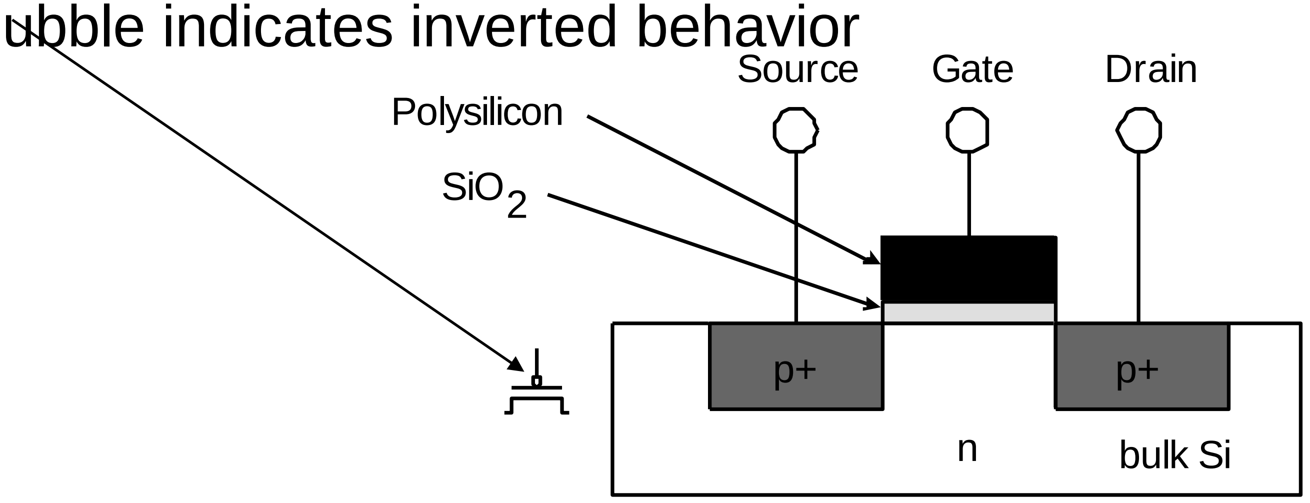
NMOS Operation Cont.

- When the gate is at a high voltage: Positive charge on gate of MOS capacitor
  - Negative charge is attracted to body under the gate
  - Inverts a channel under gate to “n-type” (N-channel, hence called the NMOS) if the gate voltage is above a threshold voltage (VT)
  - Now current can flow through “n-type” silicon from source through channel to drain, transistor is ON



PMOS Transistor

- Similar, but doping and voltages reversed
  - Body tied to high voltage (V_{DD})
  - Drain is at a lower voltage than the Source
  - Gate low: transistor ON
  - Gate high: transistor OFF
  - Bubble indicates inverted behavior

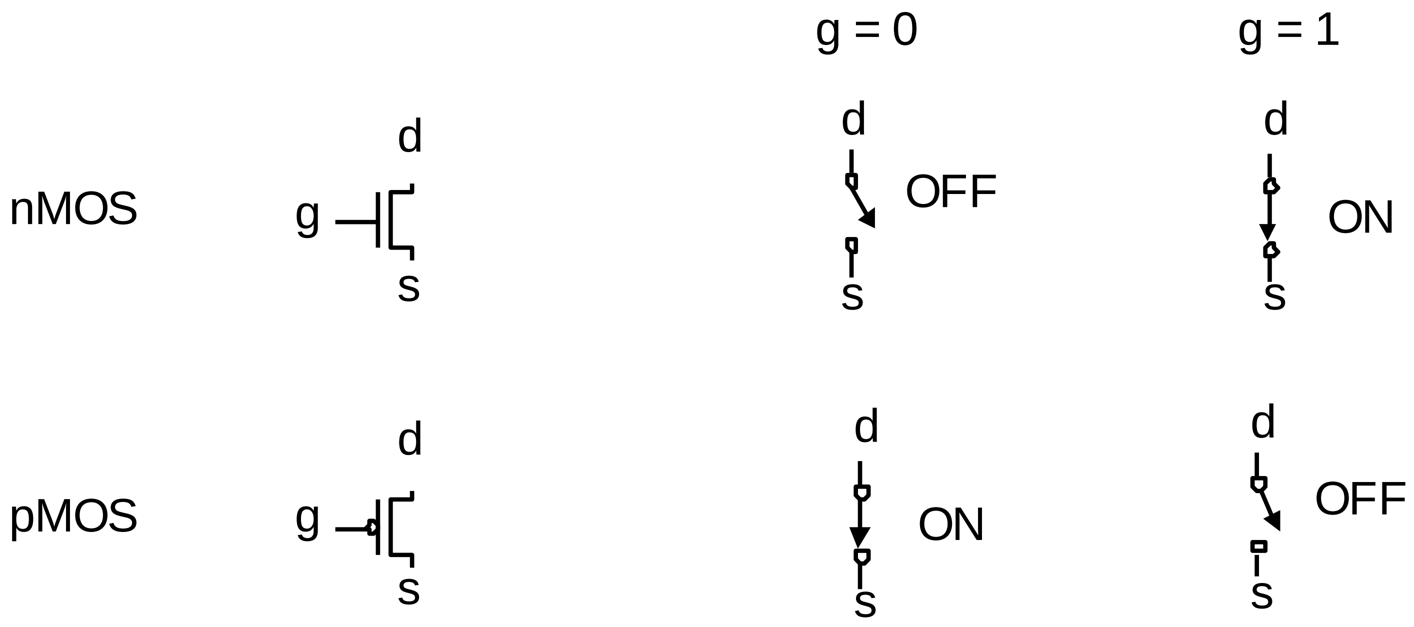


Power Supply Voltage

- GND = 0 V
- In 1980's, V_{DD} = 5V
- V_{DD} has decreased in modern processes
  - High V_{DD} would damage modern tiny transistors
  - Lower V_{DD} saves power
- V_{DD} = 3.3, 2.5, 1.8, 1.5, 1.2, 1.0,
- Effective power supply voltage can be lower due to IR drop across the power grid.

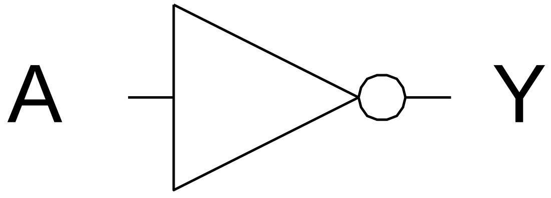
Transistors as Switches

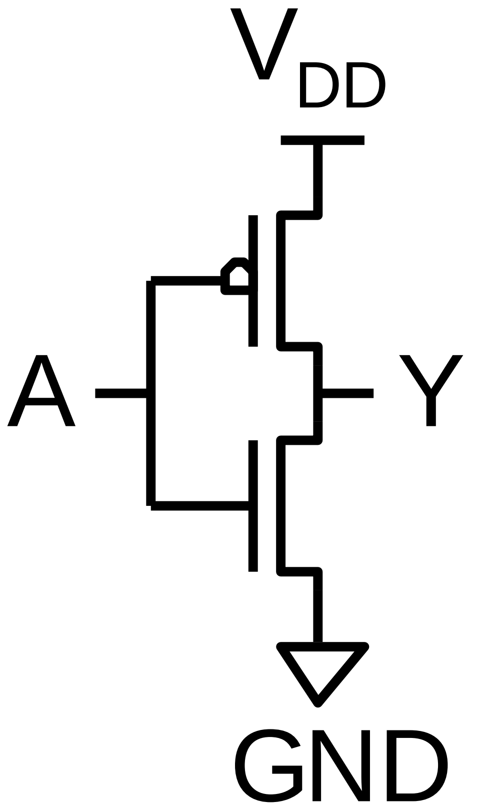
- In Digital circuits, MOS transistors are electrically controlled switches
- Voltage at gate controls path from source to drain



CMOS Inverter

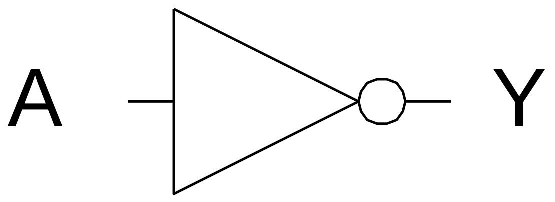
A	**Y**
0	
1	

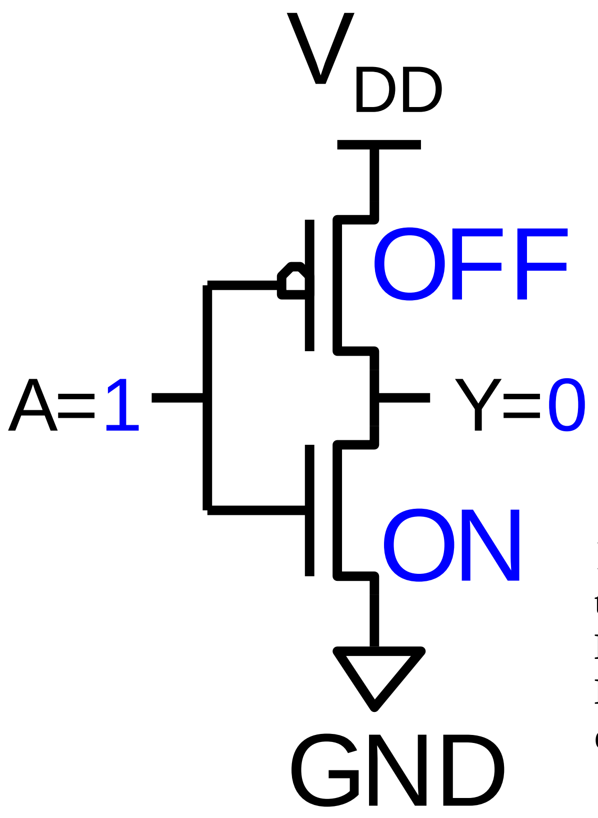




CMOS Inverter

A	Y
0	
1	0

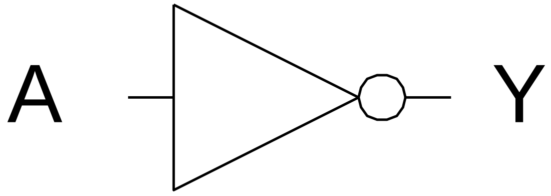


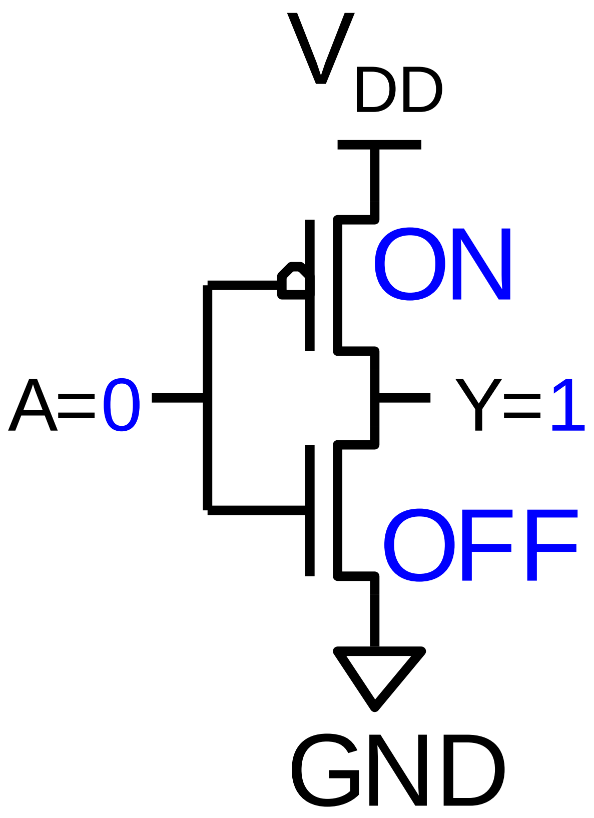


➢Y is pulled low by the turned on NMOS Device. Hence NMOS is the pull-down device.

CMOS Inverter

A	Y
0	1
1	0





➢Y is pulled high by the turned on PMOS Device. Hence PMOS is the pull-up device.

Power & energy

- Source of power disspation
    - $P = P_{switching} + P_{short\text{-}circuit} + P_{leakage} + P_{static}$
  - Definitions:
    - Switching power $P = CV^2f\alpha$
    - Short circuit power $P = I_{sc}V$
    - Leakage power $P = I_{leakage}V$
    - Static power $P = I_{static}V$
    - α : switching activity factor
  - Low power design would look at the trade-offs of the above issues

Power & energy

- Warning! In everyday language, the term "power" is used incorrectly in place of "energy"
- Power is not energy
- Power is not something you can run out of
- Power can not be lost or used up
- It is not a thing, it is merely a rate
- It can not be put into a battery any more than velocity can be put in the gas tank of a car

Power & energy

- Power supply provides energy for charging and discharging wires and transistor gates. The energy supplied is stored & then dissipated as heat.

$$\boxed{P = dw/dt}$$

Power: Rate of work being done w.r.t time
Rate of energy being used

$$P = E/\Delta t$$

Unit: *Watts = Joules/seconds*

- If a differential amount of charge dq is given a differential increase in energy dw, the potential of the charge is increased by:
- By definition of current: $I = dq/dt$ $\qquad V = dw/dq$

$$dw/dt = \frac{dw}{dq} \times \frac{dq}{dt} = P = V \times I$$

A very practical formulation!

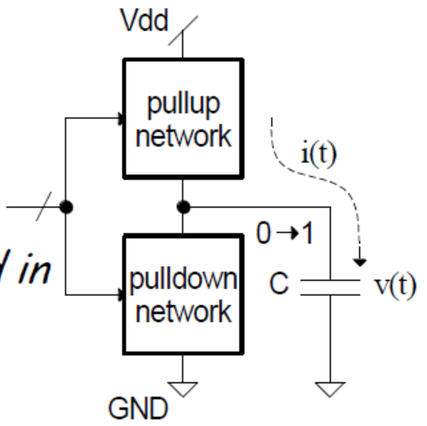
$$w = \int_{-\infty}^{t} P\,dt$$

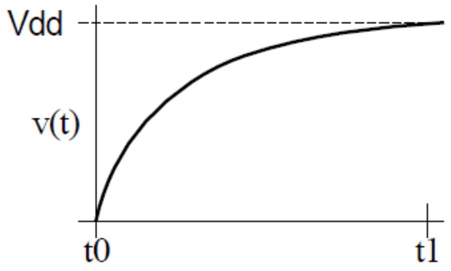
Total energy

Power & energy

Switching Energy:
energy used to
switch a node

Calculate energy dissipated in pullup:





$$E_{sw} = \int_{t_0}^{t_1} P(t)dt = \int_{t_0}^{t_1} (V_{dd} - v) \cdot i(t)dt = \int_{t_0}^{t_1} (V_{dd} - v) \cdot c(dv/dt)\,dt =$$

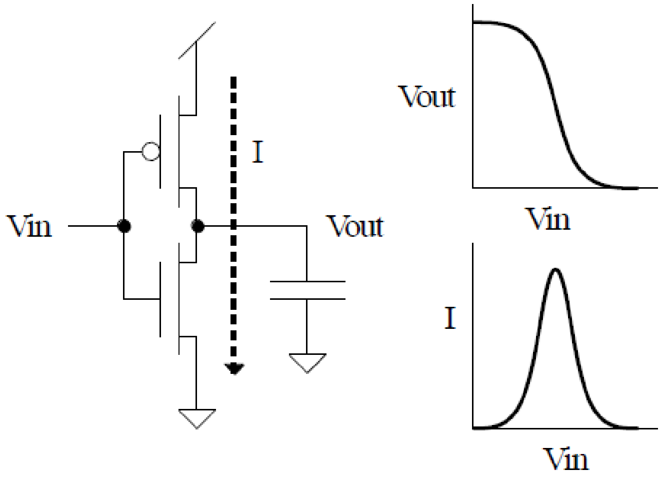
$$= \underbrace{cV_{dd}\int_{t_0}^{t_1} dv - c\int_{t_0}^{t_1} v \cdot dv = cV_{dd}^2}_{\text{Energy supplied}} - \underbrace{1/2cV_{dd}^2}_{\text{Energy stored}} = \underbrace{\boxed{1/2\,cV_{dd}^2}}_{\text{Energy dissipated}}$$

Energy supplied *Energy stored* *Energy dissipated*

An equal amount of energy is dissipated on pulldown

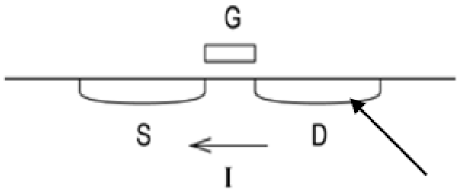
Power & energy

- "Short Circuit" Current:

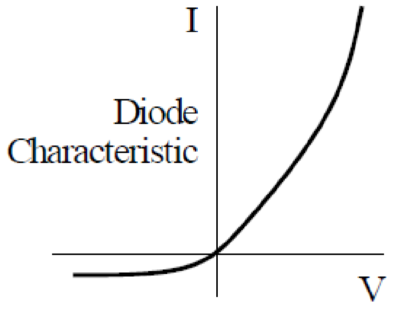


10-20% of total chip power

⌘ Junction Diode Leakage :



Transistor drain regions "leak" charge to substrate.



~1nWatt/gate
few mWatts/chip

DVS

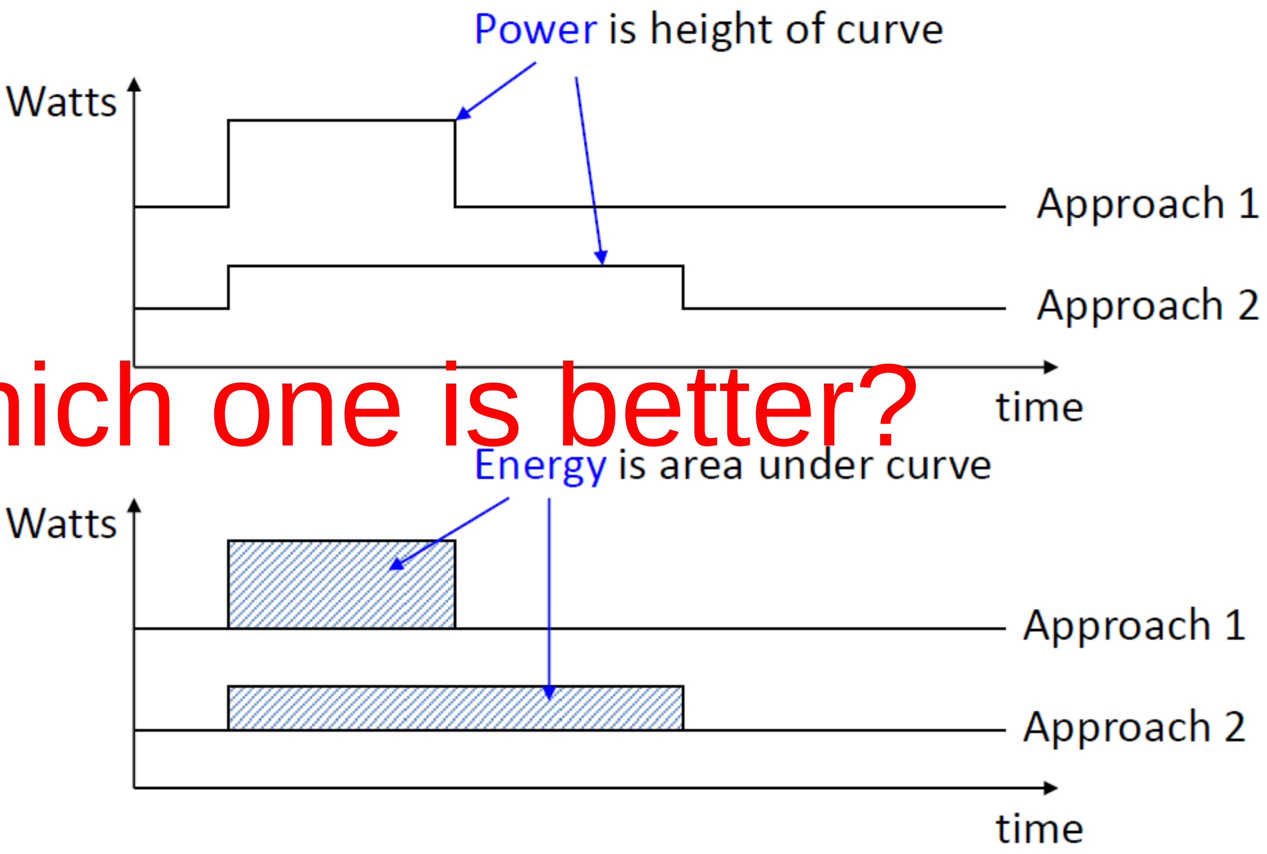
Dynamic Voltage Scaling must be along with Vth scaling!

DVS

Dynamic Voltage Scaling must be made with trade-off between ENERGY and speed!

NOT power v.s. Speed

DVS



Which one is better?

Energy = Power * time for calculation = Power * Delay

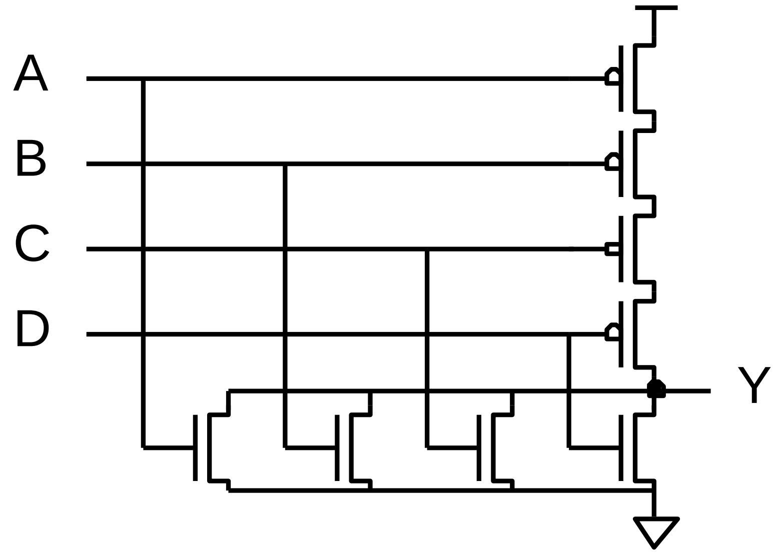
DVS

$$P = \alpha f C_L V_{DD}^2 + V_{DD} I_{peak} (P_{0\rightarrow1} + P_{1\rightarrow0}) + V_{DD} I_{leak}$$

Dynamic power (≈ 40 - 70% today and decreasing relatively)	Short-circuit power (≈ 10 % today and decreasing absolutely)	Leakage power (≈ 20 – 50 % today and increasing)

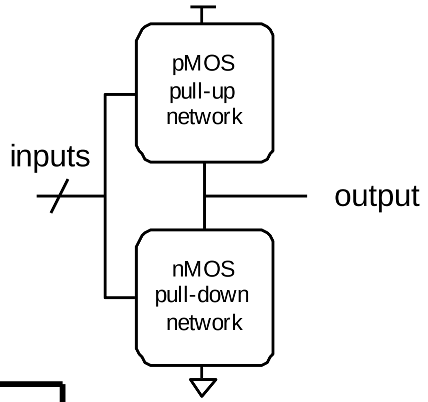
CMOS Gate Design

- A 4-input CMOS NOR gate



Complementary CMOS

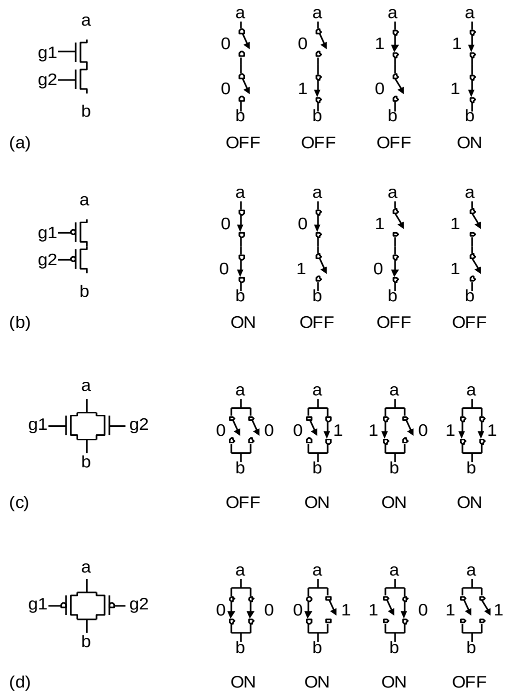
- Complementary CMOS logic gates
  - nMOS *pull-down network*
  - pMOS *pull-up network*
  - a.k.a. static CMOS



	Pull-up OFF	Pull-up ON
Pull-down OFF	Z (float)	1
Pull-down ON	0	X (crowbar)

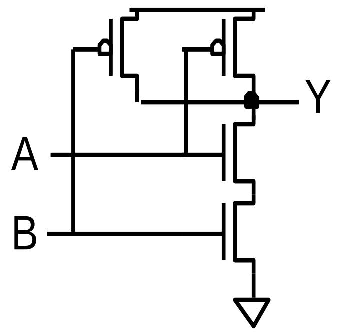
Series and Parallel

- nMOS: 1 = ON
- pMOS: 0 = ON
- *Series*: both must be ON
- *Parallel*: either can be ON



Conduction Complement

- Complementary CMOS gates always produce 0 or 1
- Ex: NAND gate
  - Series nMOS: Y=0 when both inputs are 1
  - Thus Y=1 when either input is 0
  - Requires parallel pMOS

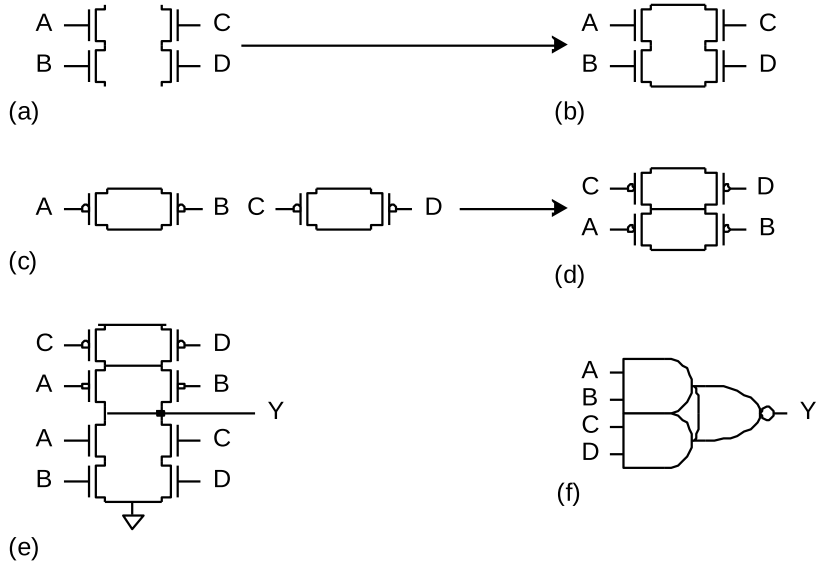


- Rule of *Conduction Complements*
  - Pull-up network is complement of pull-down
  - Parallel -> series, series -> parallel

Compound Gates

- *Compound gates* can do any inverting function
- Ex: AND-AND-OR-INV（AOI22）

$$Y = \overline{(A \bullet B) + (C \bullet D)}$$



Example: O3AI

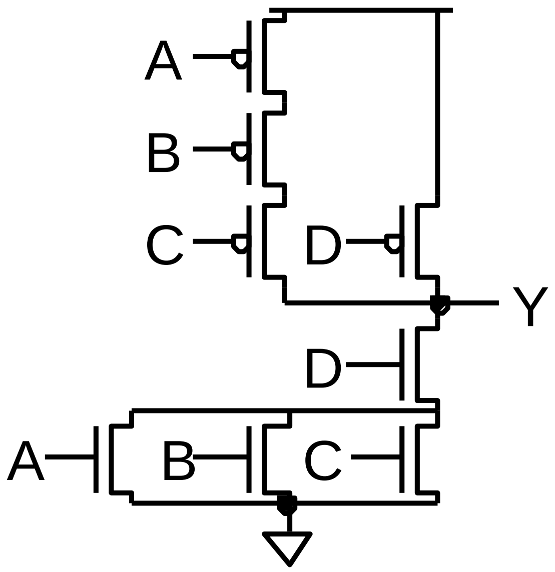
•

$$Y = \overline{(A + B + C) \bullet D}$$

Example: O3AI

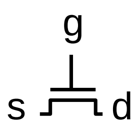
-

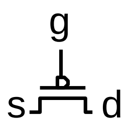
$$Y = \overline{(A+B+C)\bullet D}$$



Pass Transistors

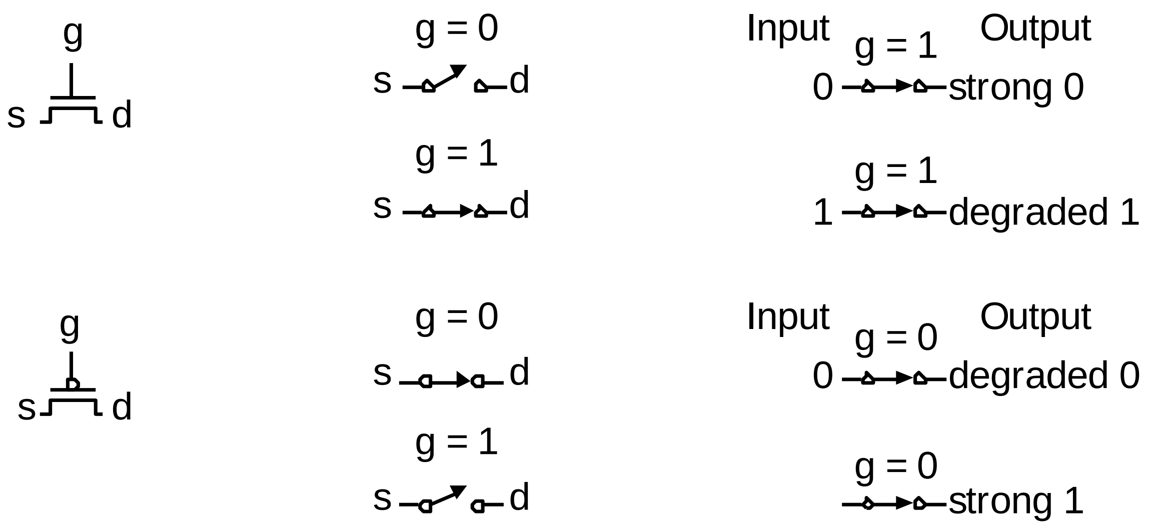
- Transistors can be used as switches





Pass Transistors

- Transistors can be used as switches



Signal Strength

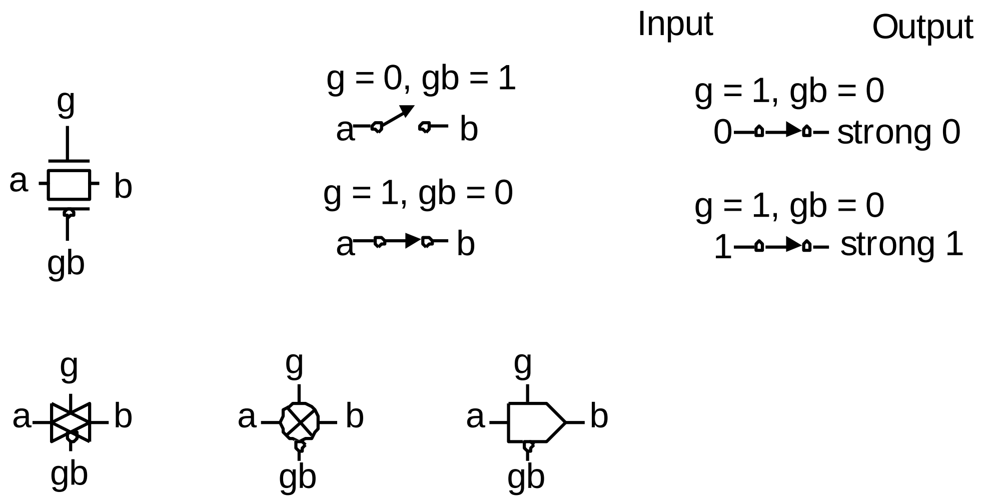
- *Strength* of signal
  - How close it approximates ideal voltage source
- V_{DD} and GND rails are strongest 1 and 0
- nMOS pass strong 0
  - But degraded or weak 1
- pMOS pass strong 1
  - But degraded or weak 0
- Thus NMOS are best for pull-down network
- Thus PMOS are best for pull-up network

Transmission Gates

- Pass transistors produce degraded outputs
- *Transmission gates* pass both 0 and 1 well

Transmission Gates

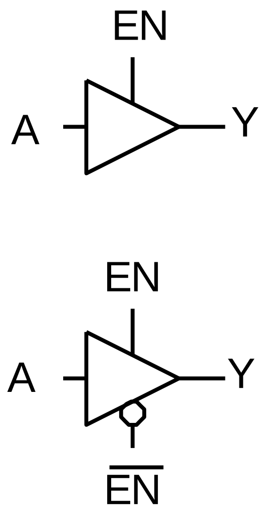
- Pass transistors produce degraded outputs
- *Transmission gates* pass both 0 and 1 well



Tristates

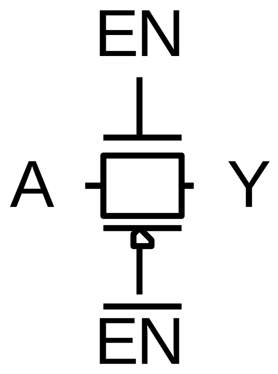
- *Tristate buffer* produces Z when not enabled

EN	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1



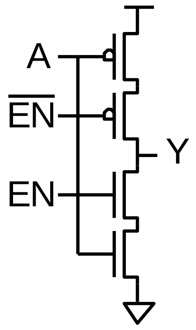
Nonrestoring Tristate

- Transmission gate acts as tristate buffer
  - Only two transistors
  - But *nonrestoring*
    - Noise on A is passed on to Y (after several stages, the noise may degrade the signal beyond recognition)



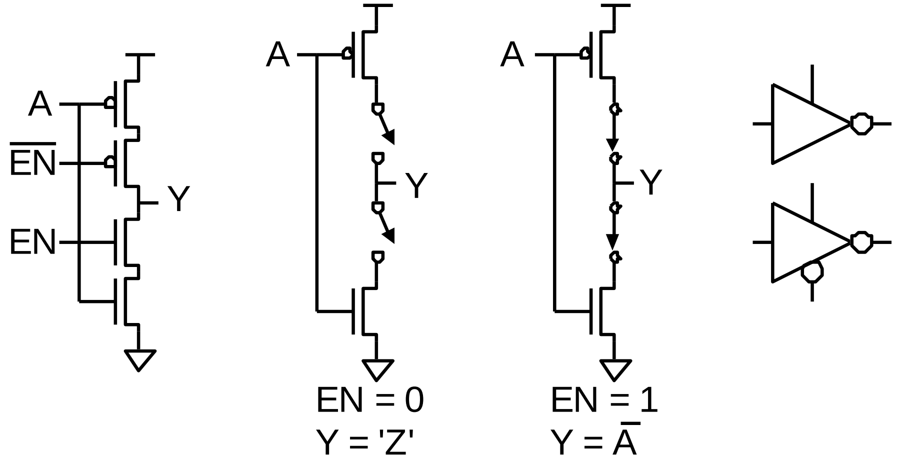
Tristate Inverter

- Tristate inverter produces restored output
- Note however that the Tristate buffer
  - ignores the conduction complement rule because we want a Z output



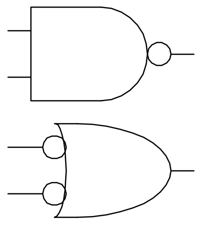
Tristate Inverter

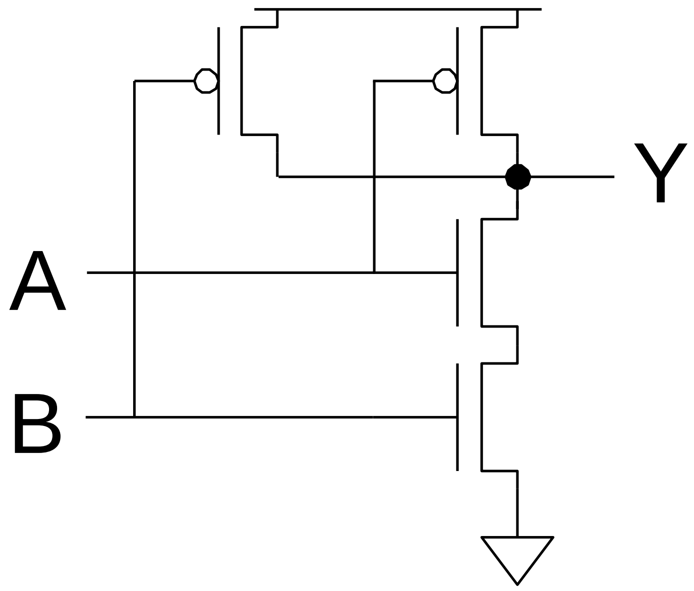
- Tristate inverter produces restored output
- Note however that the Tristate buffer
  - ignores the conduction complement rule because we want a Z output



CMOS NAND Gate

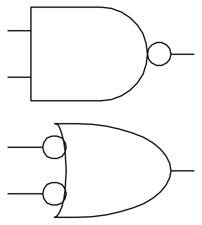
A	B	Y
0	0	
0	1	
1	0	
1	1	

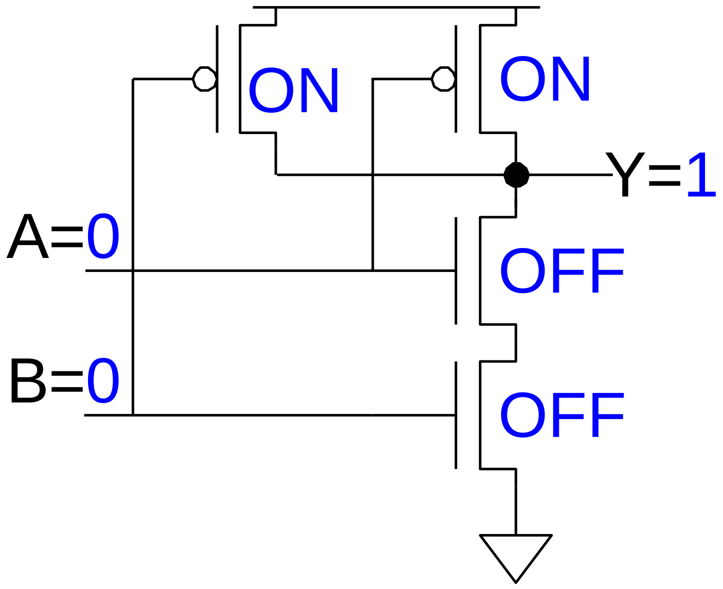




CMOS NAND Gate

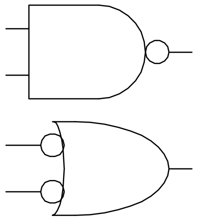
A	B	Y
0	**0**	**1**
0	1	
1	0	
1	1	

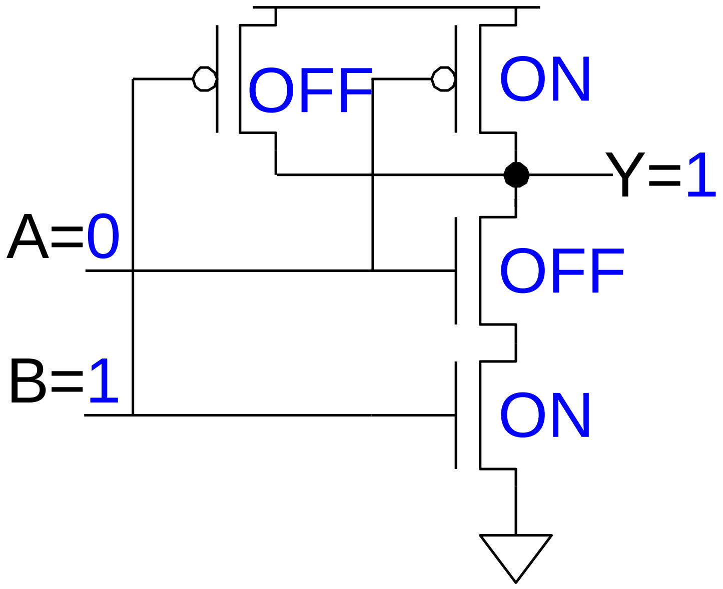




CMOS NAND Gate

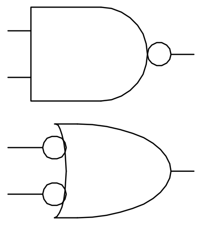
A	B	Y
0	0	1
0	**1**	**1**
1	0	
1	1	

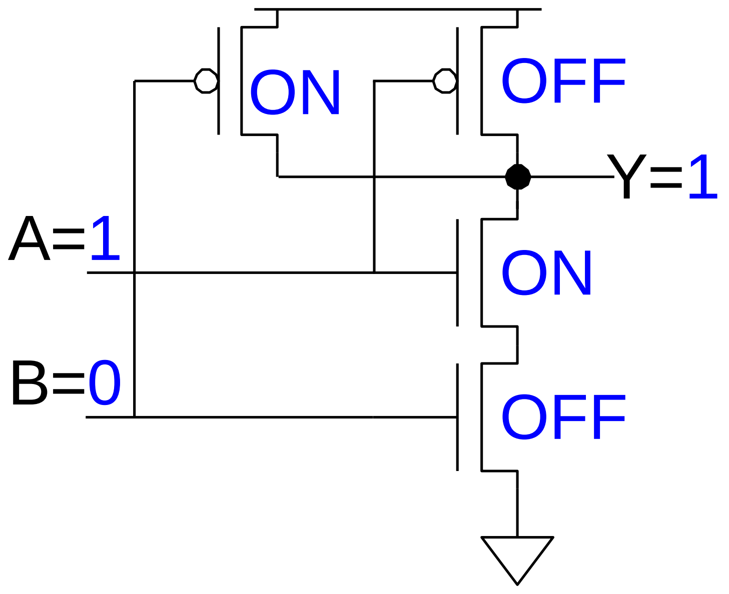




CMOS NAND Gate

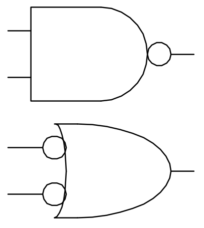
A	B	Y
0	0	1
0	1	1
1	**0**	**1**
1	1	

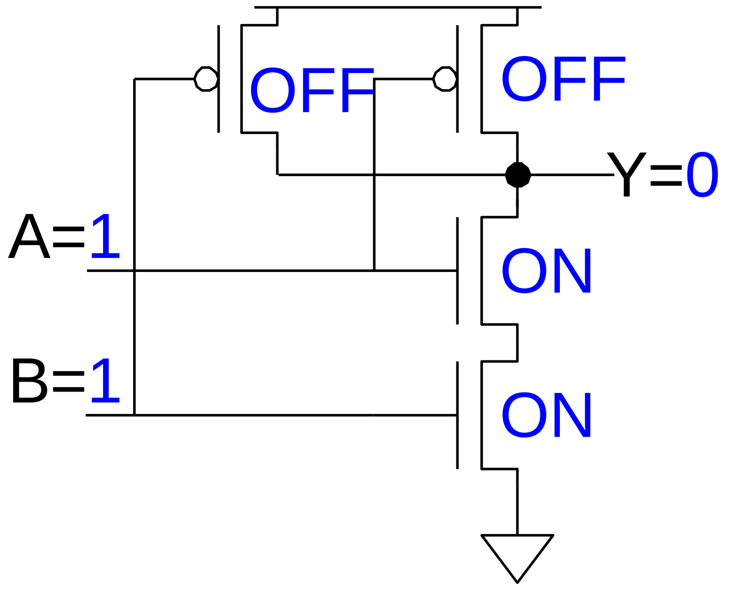




CMOS NAND Gate

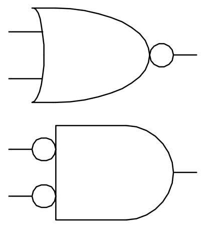
A	B	Y
0	0	1
0	1	1
1	0	1
1	**1**	**0**

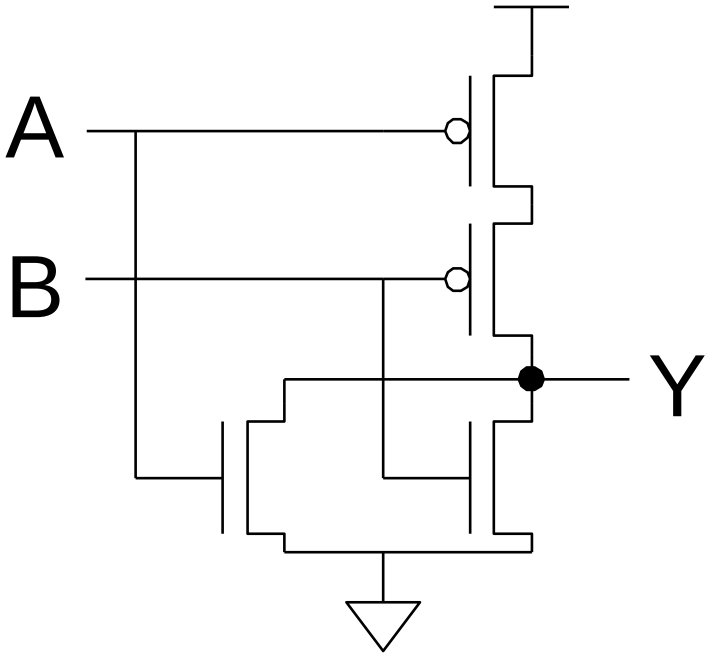




CMOS NOR Gate

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



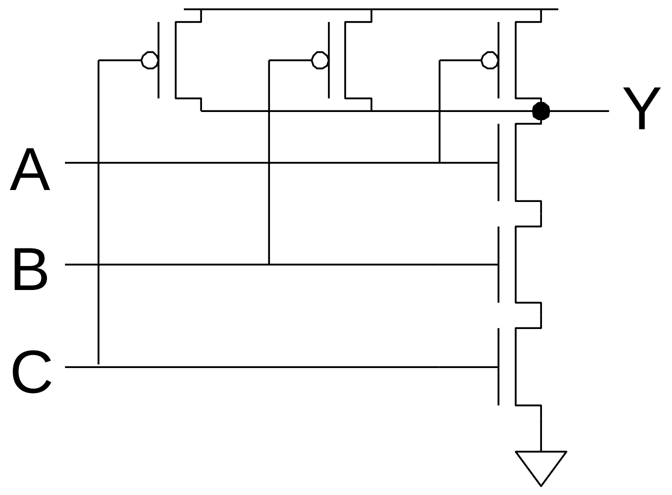


3-input NAND Gate

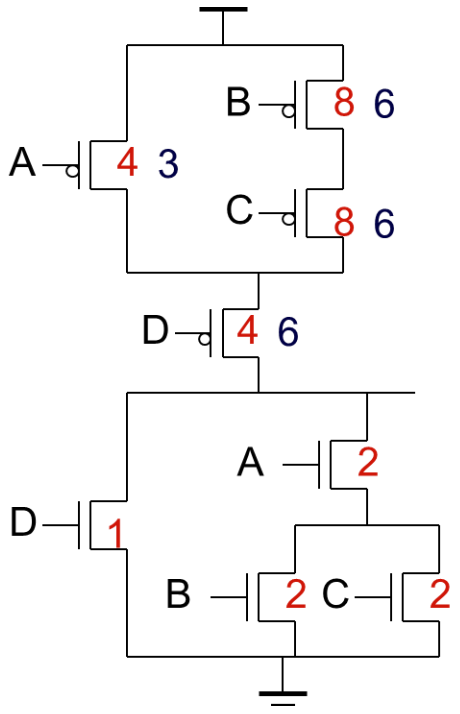
- Y pulls low if ALL inputs are 1
- Y pulls high if ANY input is 0

3-input NAND Gate

- Y pulls low if ALL inputs are 1
- Y pulls high if ANY input is 0



Complex Gate



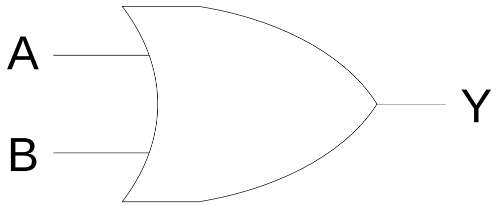
$OUT = \overline{D + A \cdot (B + C)}$

Karnaugh maps

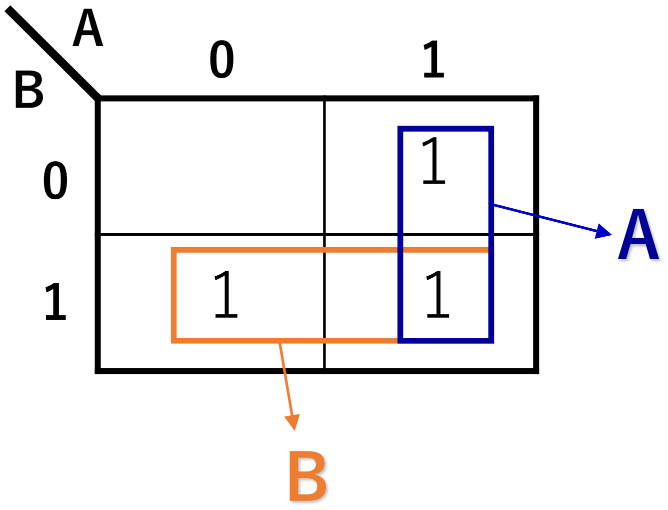
- The Karnaugh map is completed by entering a '1‘(or ‘0’) in each of the appropriate cells.
- Within the map, adjacent cells containing 1's (or 0’s) are grouped together in twos, fours, or eights.

Example

2-variable Karnaugh maps are trivial but can be used to introduce the methods you need to learn. The map for a 2-input OR gate looks like this:



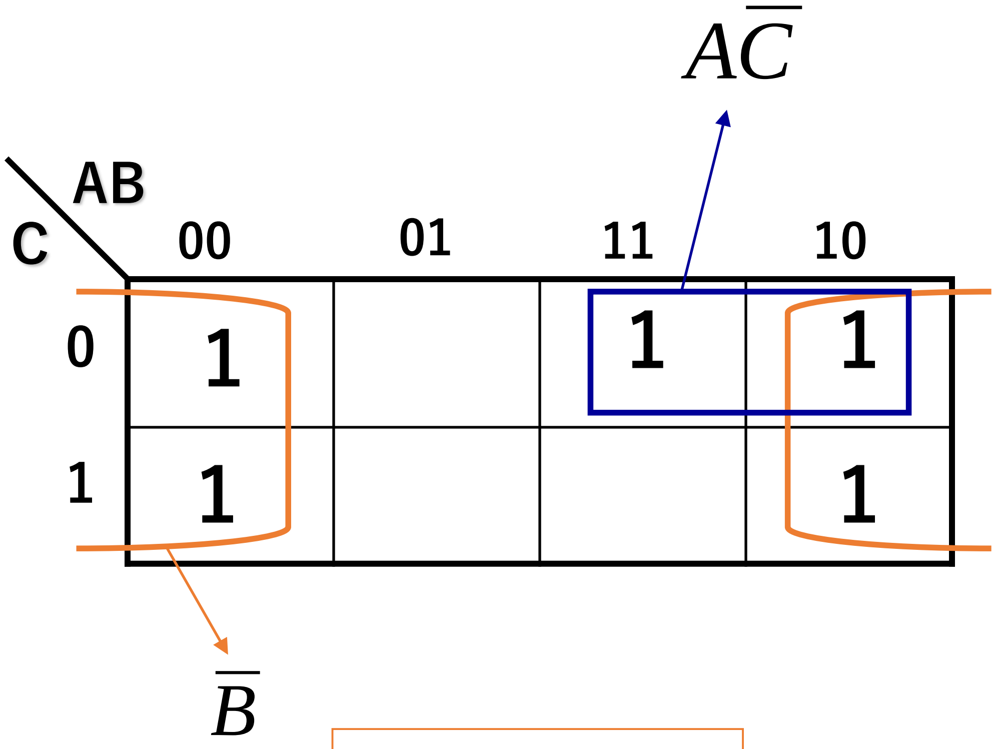
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



A+B

Example

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



$$\boxed{\overline{B} + A\overline{C}}$$

Binary addition by hand

- You can add two binary numbers one column at a time starting from the right, just like you add two decimal numbers.
- But remember it's binary. For example, 1 + 1 = 10 and you have to carry!

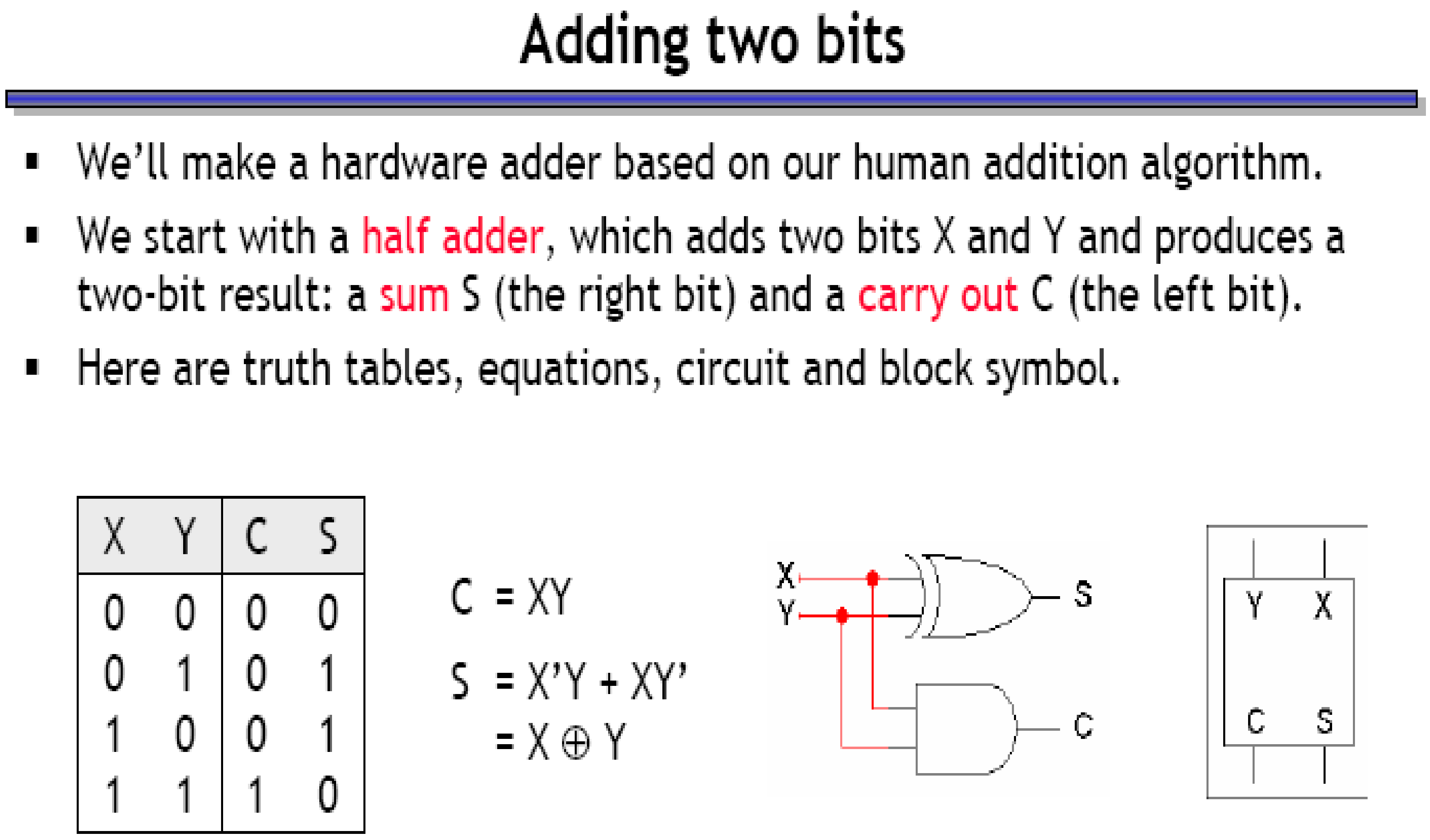
The initial carry in is implicitly 0

$$\begin{array}{cccccc l}
 & 1 & 1 & 1 & 0 & \downarrow & \text{Carry in}\\
 & & 1 & 0 & 1 & 1 & \text{Augend}\\
+ & & 1 & 1 & 1 & 0 & \text{Addend}\\ \cline{1-6}
 & 1 & 1 & 0 & 0 & 1 & \text{Sum}\\
 & \uparrow & & & & \uparrow &
\end{array}$$

most significant bit (MSB)

least significant bit (LSB)

HALF ADDER



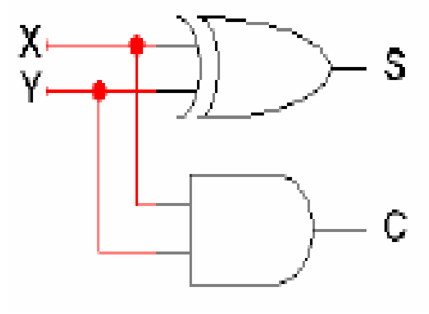
Adding two bits

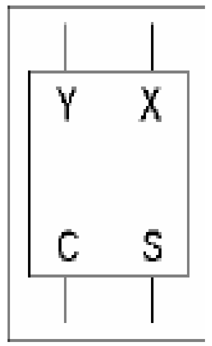
- We'll make a hardware adder based on our human addition algorithm.
- We start with a half adder, which adds two bits X and Y and produces a two-bit result: a sum S (the right bit) and a carry out C (the left bit).
- Here are truth tables, equations, circuit and block symbol.

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

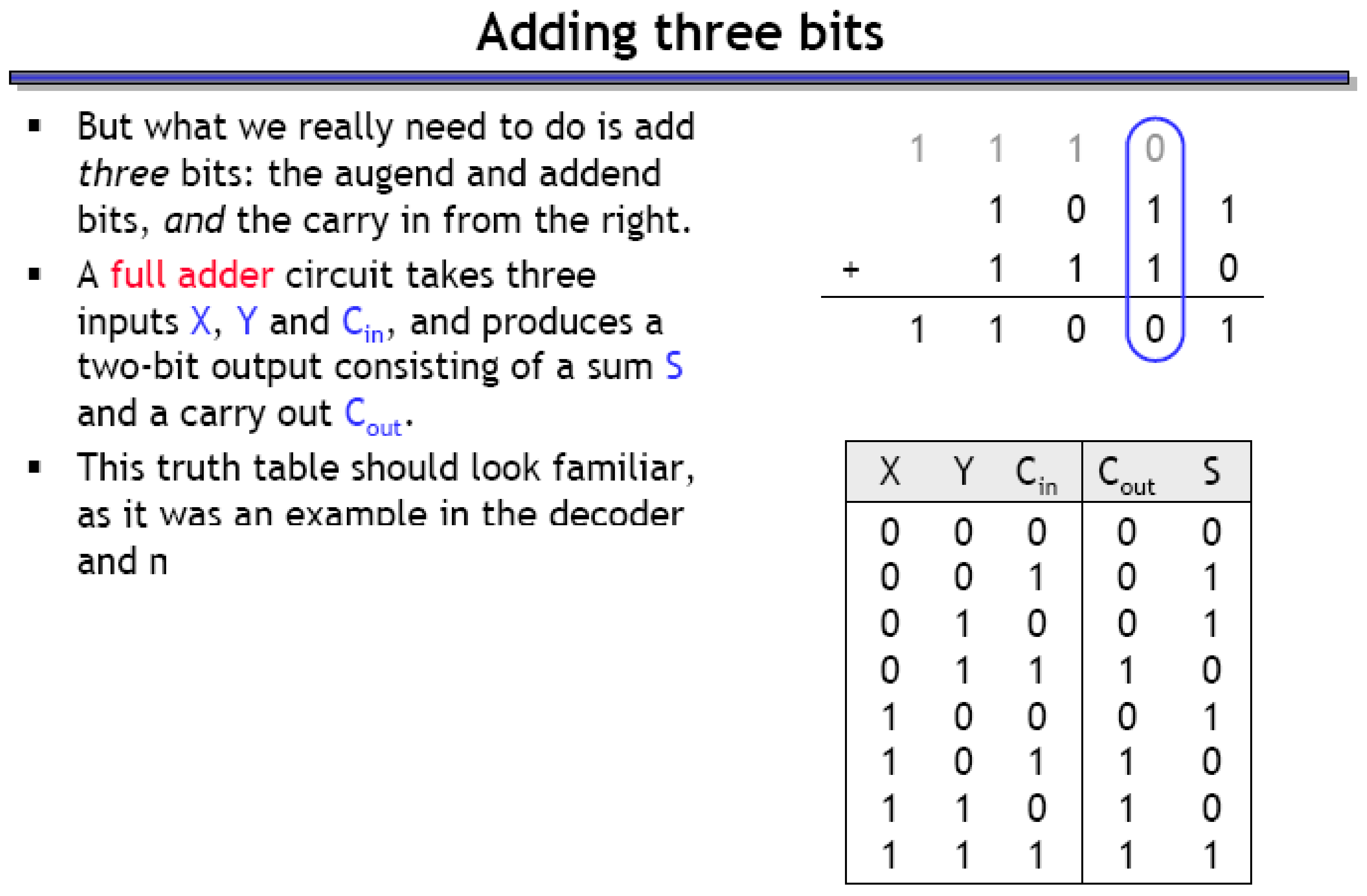
$$C = XY$$

$$S = X'Y + XY' = X \oplus Y$$





FULL ADDER



Adding three bits

- But what we really need to do is add *three* bits: the augend and addend bits, *and* the carry in from the right.
- A full adder circuit takes three inputs X, Y and C_{in}, and produces a two-bit output consisting of a sum S and a carry out C_{out}.
- This truth table should look familiar, as it was an example in the decoder and n

$$\begin{array}{cccccc} & 1 & 1 & 1 & 0 & \\ & & 1 & 0 & 1 & 1 \\ + & & 1 & 1 & 1 & 0 \\ \hline & 1 & 1 & 0 & 0 & 1 \end{array}$$

X	Y	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full adder equations

- Using Boolean algebra, we can simplify S and C_{out} as shown here.

X	Y	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$\begin{aligned}
S &= \Sigma m(1,2,4,7) \\
&= X'Y'C_{in} + X'YC_{in}' + XY'C_{in}' + XYC_{in} \\
&= X'(Y'C_{in} + YC_{in}') + X(Y'C_{in}' + YC_{in}) \\
&= X'(Y \oplus C_{in}) + X(Y \oplus C_{in})' \\
&= X \oplus Y \oplus C_{in}
\end{aligned}$$

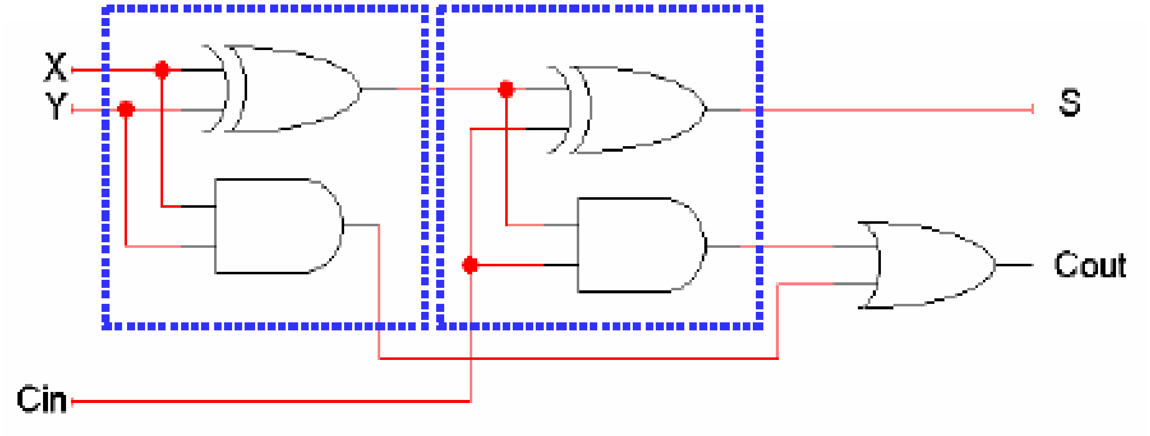
$$\begin{aligned}
C_{out} &= \Sigma m(3,5,6,7) \\
&= X'YC_{in} + XY'C_{in} + XYC_{in}' + XYC_{in} \\
&= (X'Y + XY')C_{in} + XY(C_{in}' + C_{in})
\end{aligned}$$

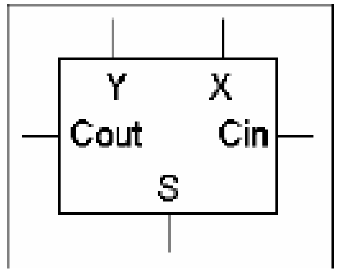
- Notice that XOR operations simplify things a bit, but we had to resort to using algebra since it's hard to find XOR-based expressions with K-maps.

Full adder circuit

- We write the equations this way to highlight the hierarchical nature of adder circuits—you can build a full adder by combining two half adders!

$$S = X \oplus Y \oplus C_{in}$$
$$C_{out} = (X \oplus Y)\, C_{in} + XY$$



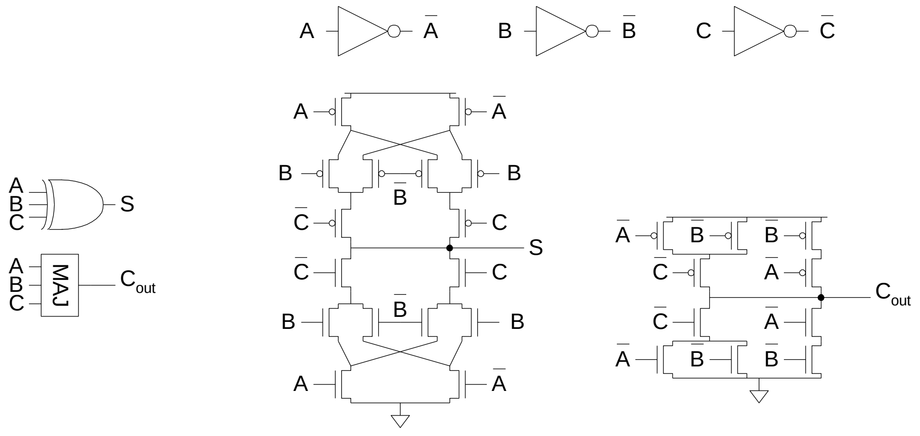


Full Adder Design I

- Brute force implementation from eqns

$$S = A \oplus B \oplus C$$

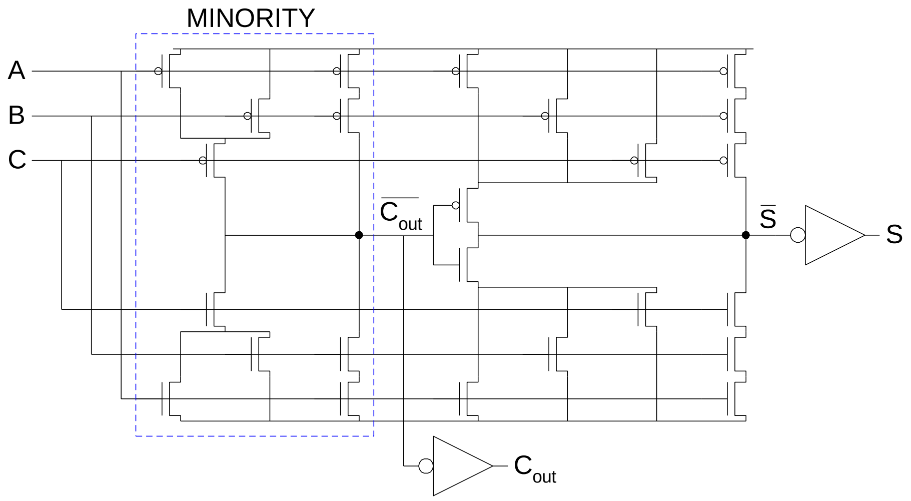
$$C_{\text{out}} = MAJ(A, B, C)$$



Full Adder Design II

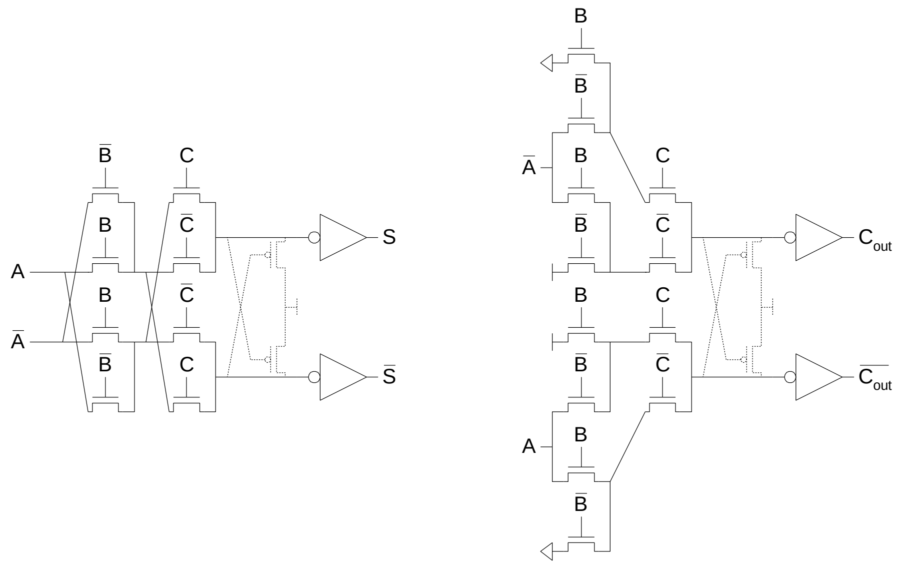
- Factor S in terms of C_{out}

  $S = ABC + (A + B + C)(\sim C_{out})$
- Critical path is usually C to C_{out} in ripple adder

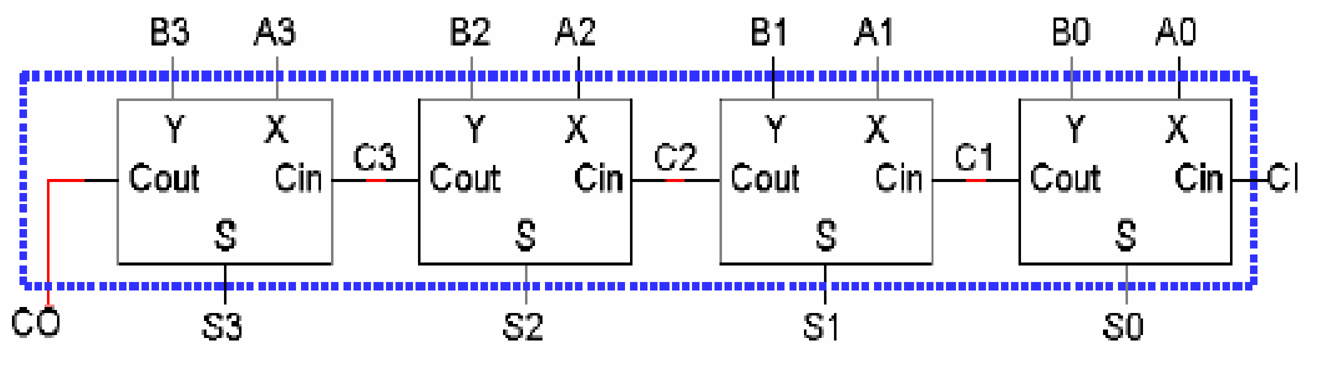


Full Adder Design III

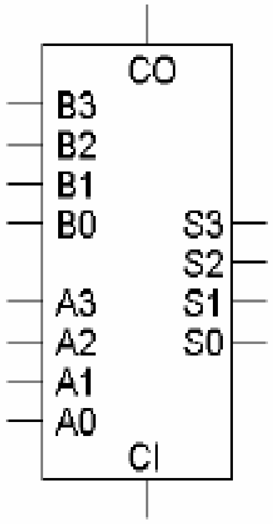
- Complementary Pass Transistor Logic (CPL)
  - Slightly faster, but more area



A four-bit adder

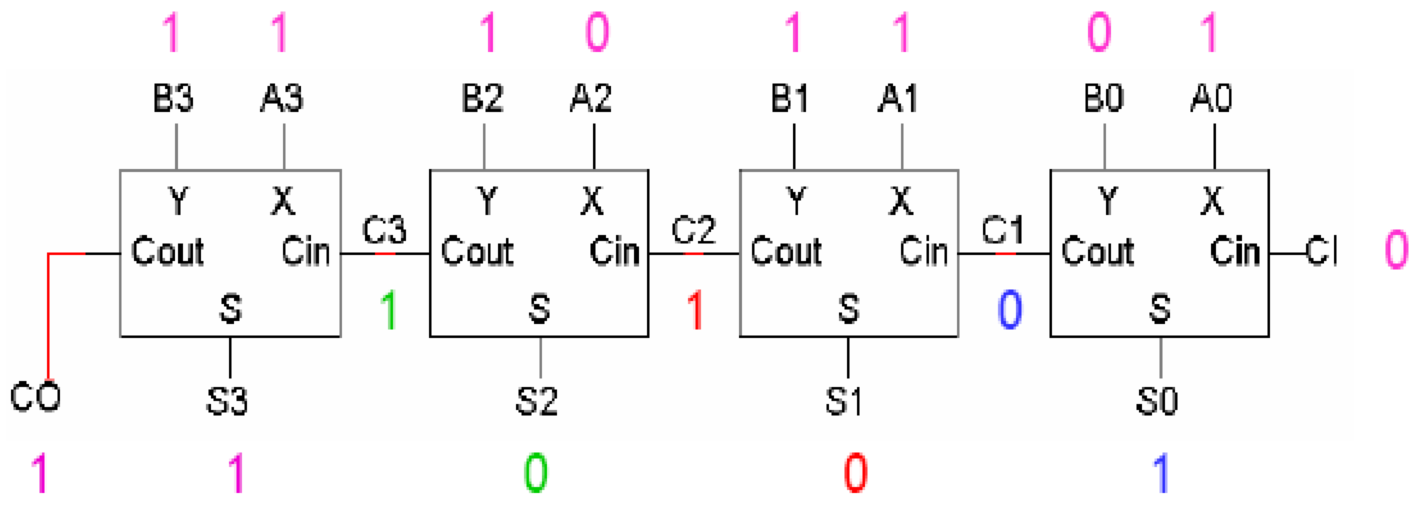


- Similarly, we can cascade four full adders to build a four-bit adder.
  - The inputs are two four-bit numbers (A3A2A1A0 and B3B2B1B0) and a carry in CI.
  - The two outputs are a four-bit sum S3S2S1S0 and the carry out CO.
- If you designed this adder without taking advantage of the hierarchical structure, you'd end up with a 512-row truth table with five outputs!



An example of 4-bit addition

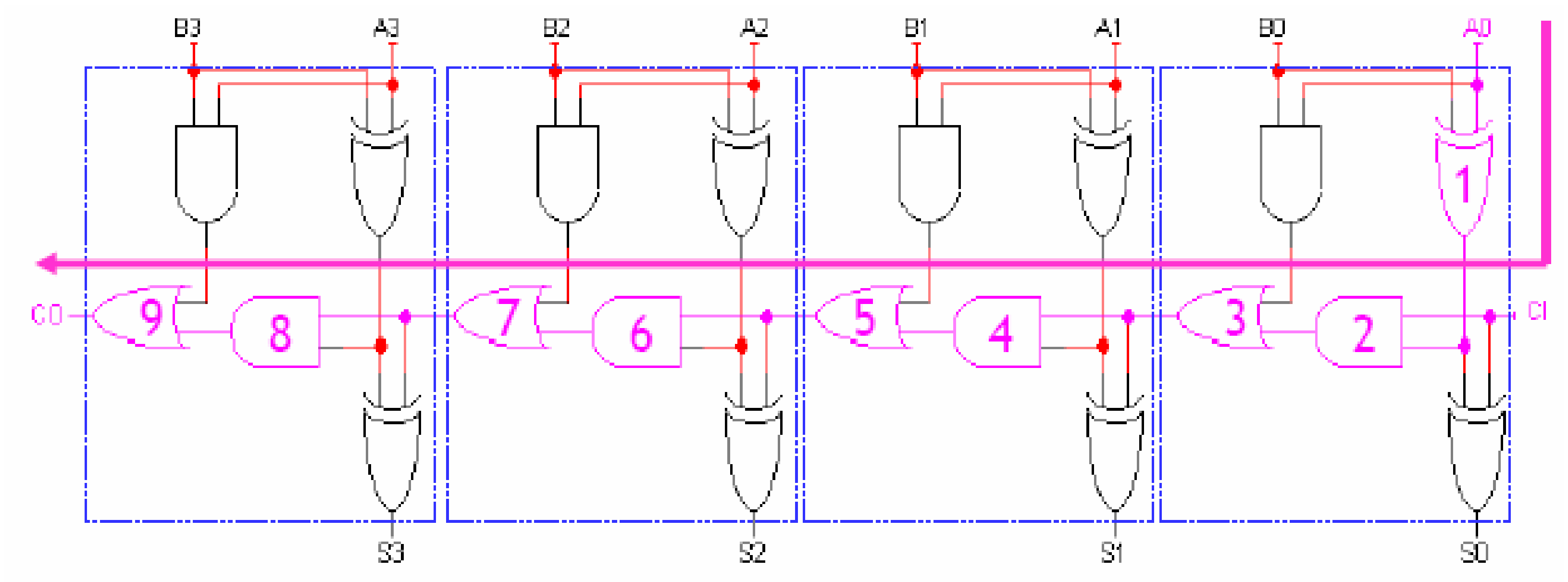
- Let's put our initial example into this circuit, with A=1011 and B=1110.



1. Fill in all the inputs, including CI=0
2. The circuit produces C1 and S0 (1 + 0 + 0 = 01)
3. Use C1 to find C2 and S1 (1 + 1 + 0 = 10)
4. Use C2 to compute C3 and S2 (0 + 1 + 1 = 10)
5. Use C3 to compute CO and S3 (1 + 1 + 1 = 11)

Ripple carry delays

- The diagram below shows our four-bit adder completely drawn out.
- This is called a ripple carry adder, because the inputs A0, B0 and CI "ripple" leftwards until CO and S3 are produced.
- Ripple carry adders are slow!
  - There is a very long path from A0, B0 and CI to CO and S3.
  - For an *n*-bit ripple carry adder, the longest path has 2*n*+1 gates.
  - The longest path in a 64-bit adder would include 129 gates!



PGK

- For a full adder, define what happens to carries
  - Generate: $C_{out} = 1$ independent of C
    - G =
  - Propagate: $C_{out} = C$
    - P =
  - Kill: $C_{out} = 0$ independent of C
    - K =

PGK

- For a full adder, define what happens to carries
  - Generate: $C_{out} = 1$ independent of C
    - $G = A \bullet B$
  - Propagate: $C_{out} = C$
    - $P = A \oplus B$
  - Kill: $C_{out} = 0$ independent of C
    - $K = {\sim}A \bullet {\sim}B$

A faster way to compute carry outs

- Instead of waiting for the carry out from each previous stage, we can minimize the delay by computing it directly with a two-level circuit.
- First we'll define two functions.
  - The "generate" function G_i produces 1 when there *must* be a carry out from position i (i.e., when A_i and B_i are both 1).

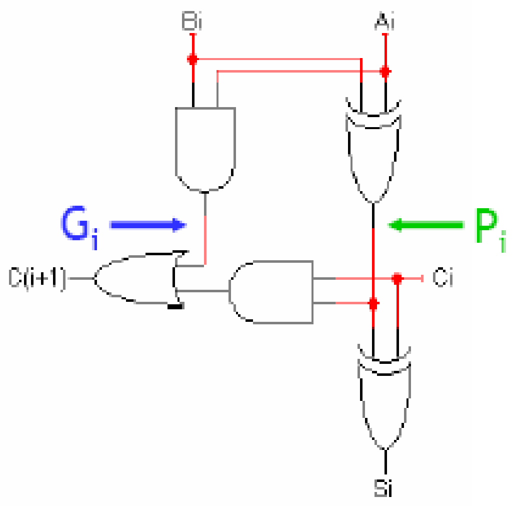
$$G_i = A_iB_i$$

  - The "propagate" function P_i is true when an incoming carry is propagated (i.e, when A_i=1 or B_i=1, but not both).

$$P_i = A_i \oplus B_i$$

- Then we can rewrite the carry out function.

$$C_{i+1} = G_i + P_iC_i$$

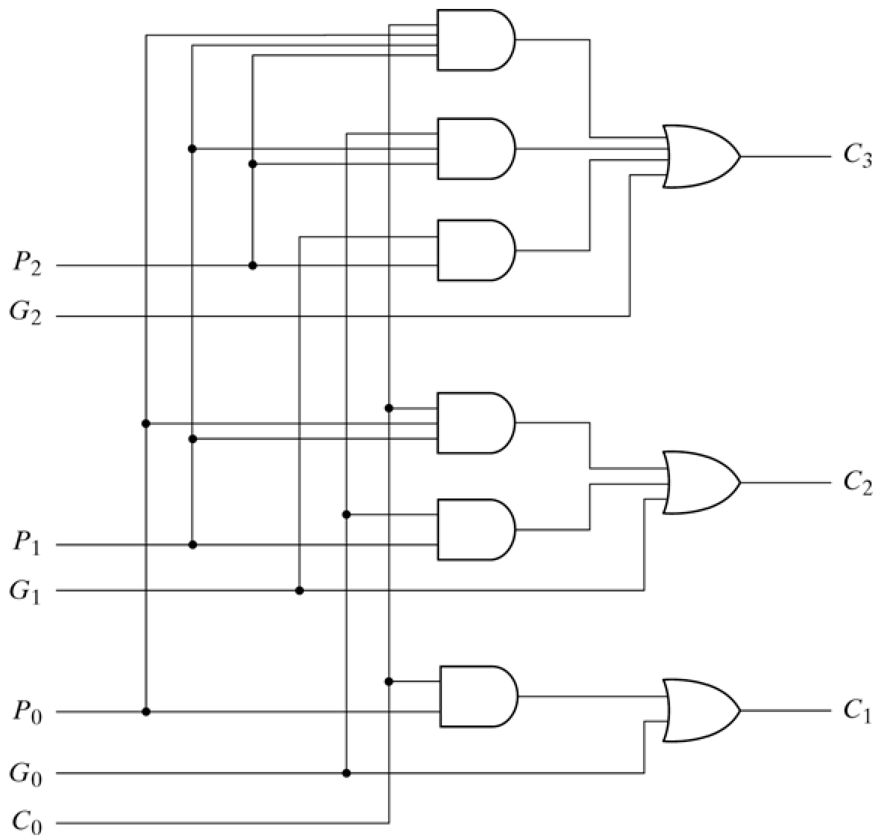


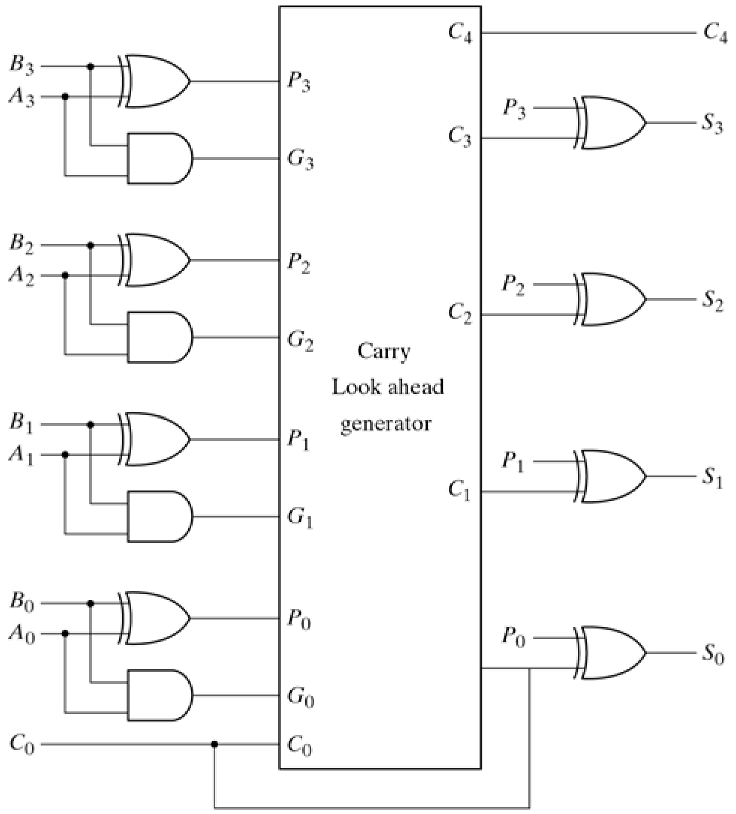
A_i	B_i	C_i	C_{i+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- Let's look at the carry out equations for specific bits, using the general equation from the previous page $C_{i+1} = G_i + P_iC_i$.

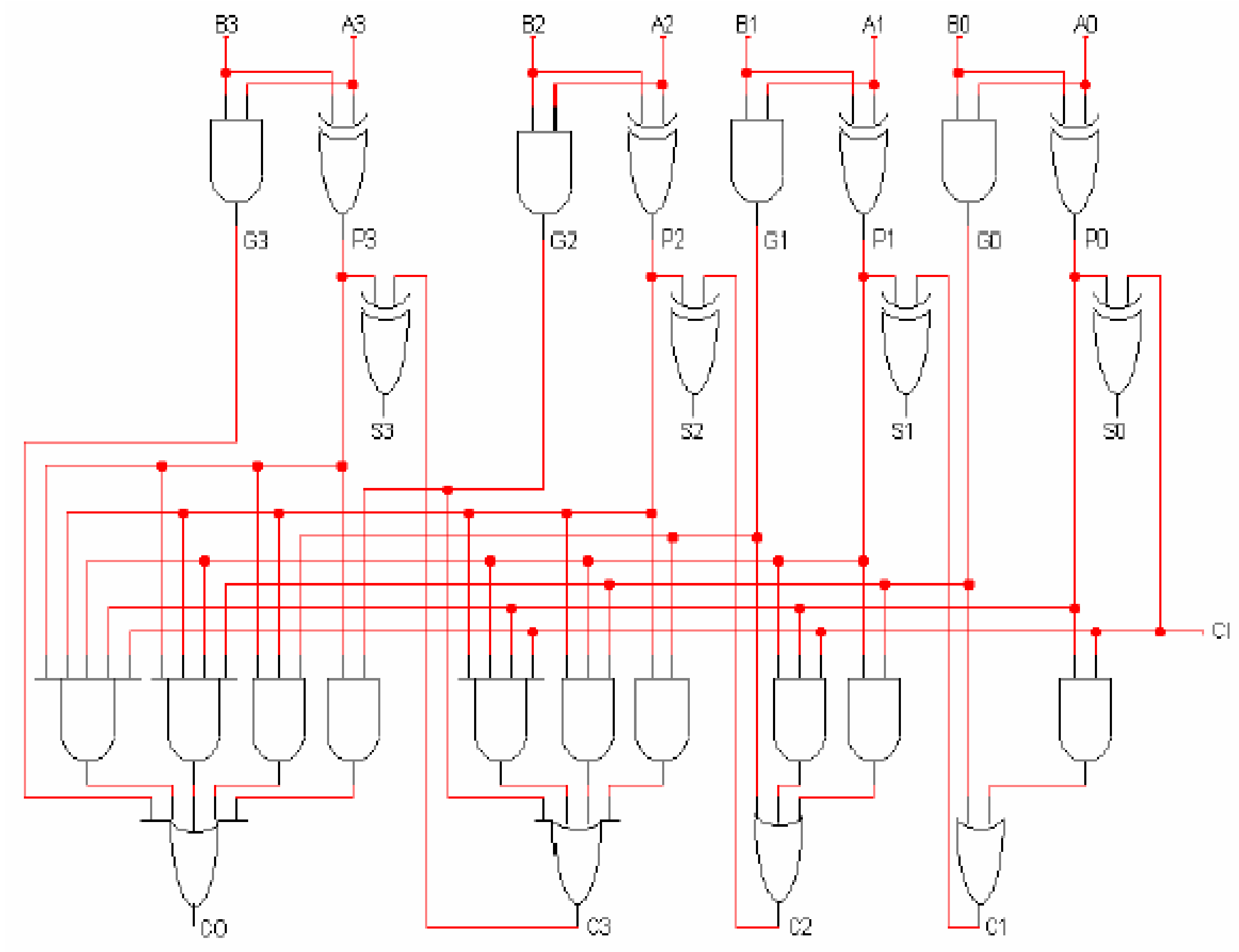
$$
\begin{aligned}
C_1 &= G_0 + P_0C_0 \\[1em]
C_2 &= G_1 + P_1C_1 \\
&= G_1 + P_1(G_0 + P_0C_0) \\
&= G_1 + P_1G_0 + P_1P_0C_0 \\[1em]
C_3 &= G_2 + P_2C_2 \\
&= G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0) \\
&= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0 \\[1em]
C_4 &= G_3 + P_3C_3 \\
&= G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0) \\
&= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0
\end{aligned}
$$

- These expressions are all sums of products, so we can use them to make a circuit with only a two-level delay.

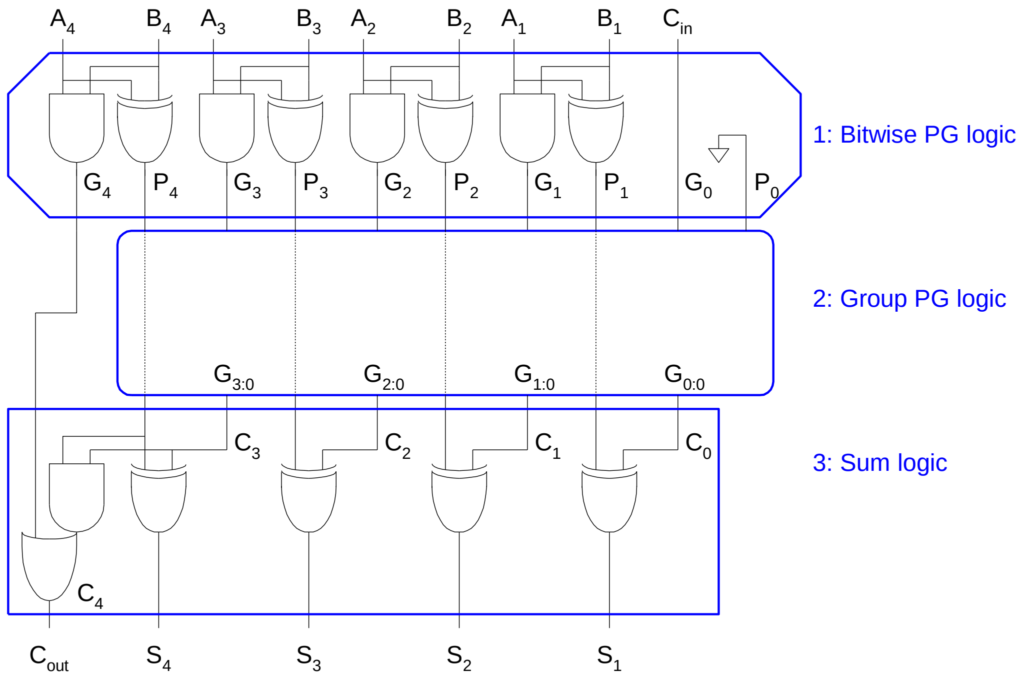

P_2
G_2
P_1
G_1
P_0
G_0
C_0
C_3
C_2
C_1


B_3
A_3
P_3
G_3
B_2
A_2
P_2
G_2
B_1
A_1
P_1
G_1
B_0
A_0
P_0
G_0
C_0
C_0
Carry
Look ahead
generator
C_4
C_4
P_3
C_3
S_3
P_2
C_2
S_2
P_1
C_1
S_1
P_0
S_0

A faster four-bit adder



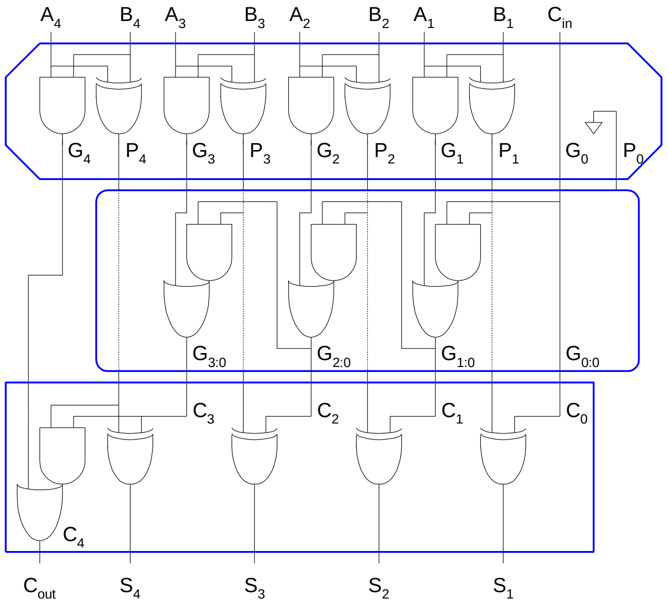
PG Logic



A_4 B_4 A_3 B_3 A_2 B_2 A_1 B_1 C_{in}
G_4 P_4 G_3 P_3 G_2 P_2 G_1 P_1 G_0 P_0
1: Bitwise PG logic
2: Group PG logic
$G_{3:0}$ $G_{2:0}$ $G_{1:0}$ $G_{0:0}$
C_3 C_2 C_1 C_0
3: Sum logic
C_4
C_{out} S_4 S_3 S_2 S_1

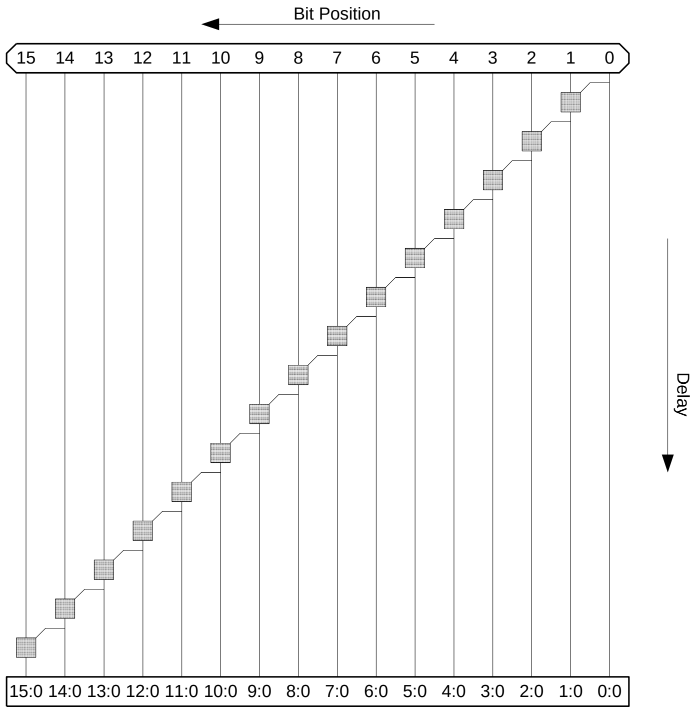
Carry-Ripple Revisited

$$G_{i:0} = G_i + P_i \bullet G_{i-1:0}$$



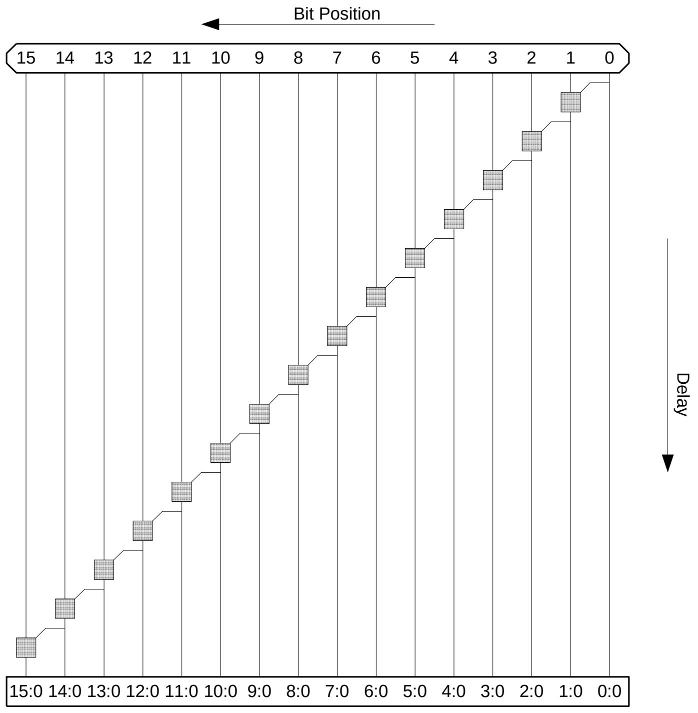
Carry-Ripple PG Diagram

$t_{\text{ripple}} =$

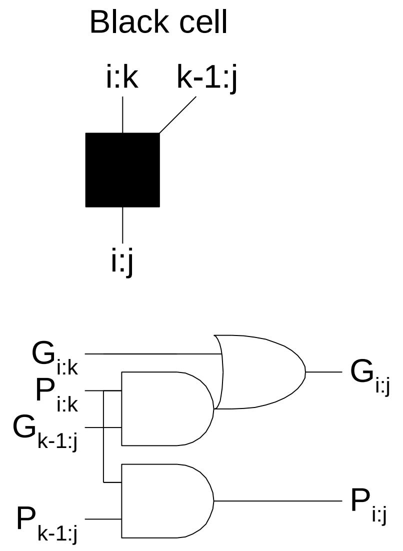


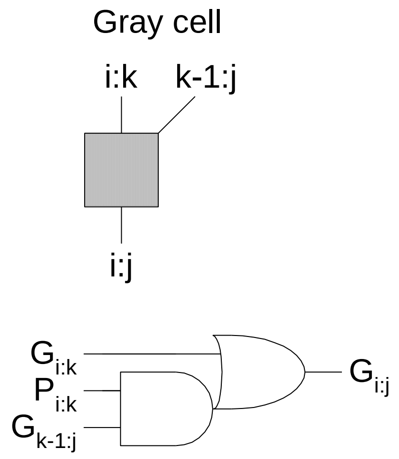
Carry-Ripple PG Diagram

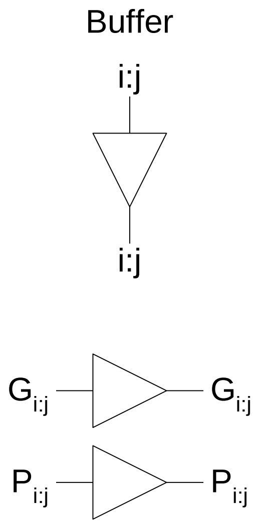
$$t_{\text{ripple}} = t_{pg} + (N-1)t_{AO} + t_{\text{xor}}$$



PG Diagram Notation

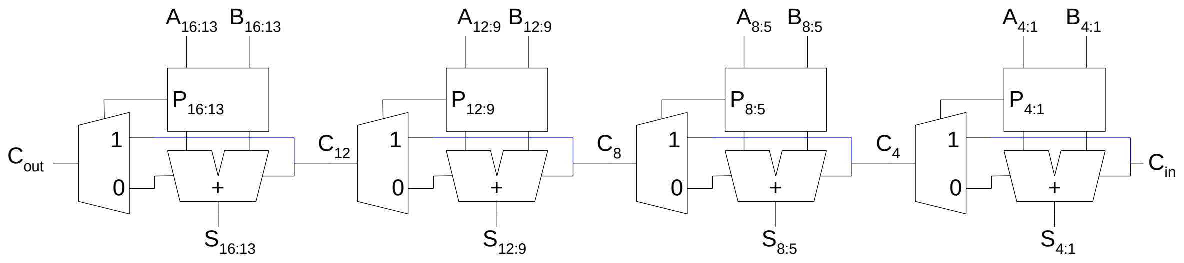




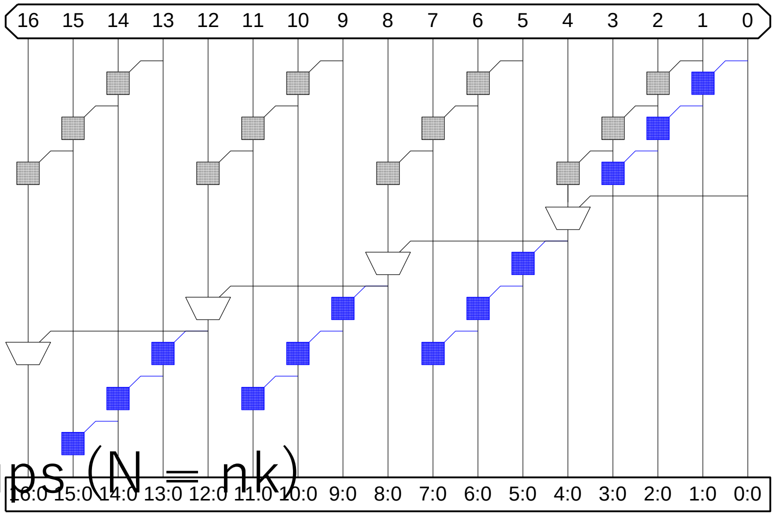


Carry-Skip Adder

- Carry-ripple is slow through all N stages
- Carry-skip allows carry to skip over groups of n bits
  - Decision based on n-bit propagate signal



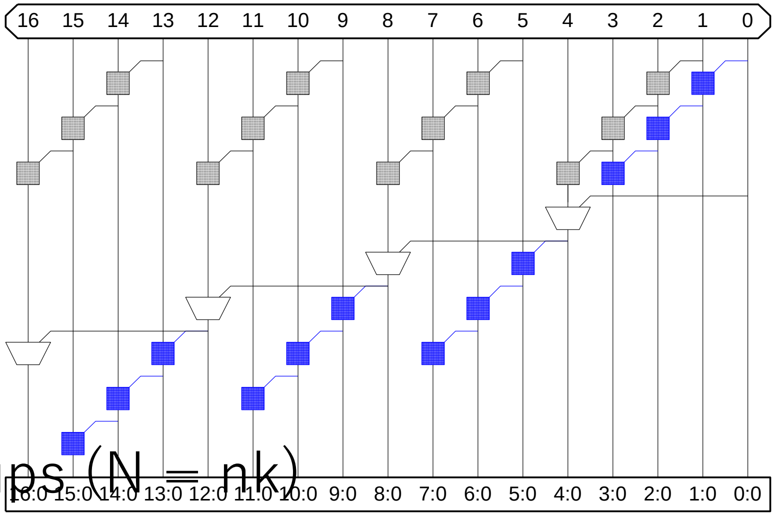
Carry-Skip PG Diagram



For k n-bit groups (N = nk)

$t_{\text{skip}} =$

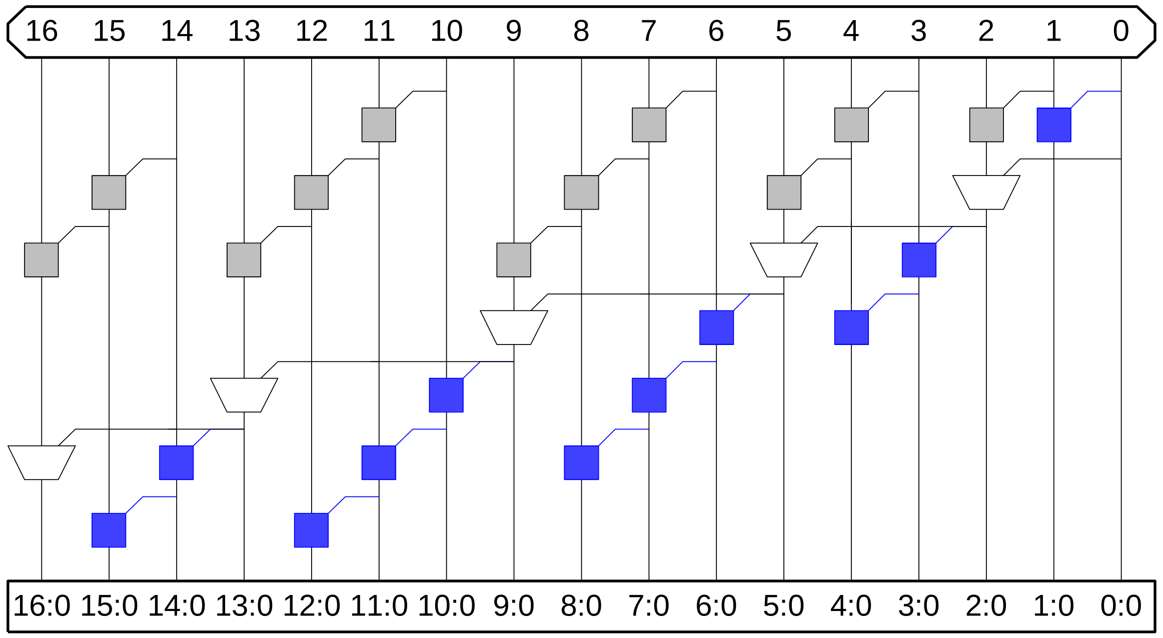
Carry-Skip PG Diagram



For k n-bit groups (N = nk)

$$t_{\text{skip}} = t_{pg} + \left[2(n-1) + (k-1)\right] t_{AO} + t_{\text{xor}}$$

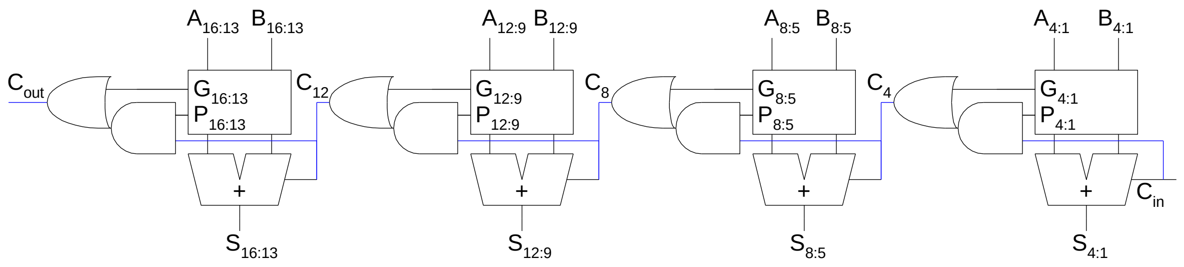
Variable Group Size



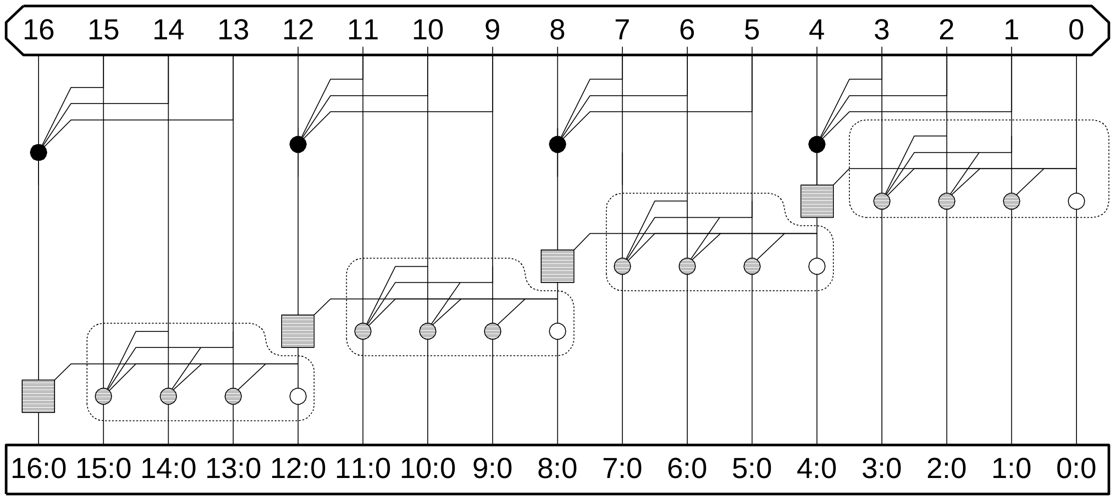
Delay grows as O(sqrt(N))

Carry-Lookahead Adder

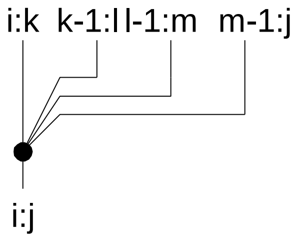
- Carry-lookahead adder computes $G_{i:0}$ for many bits in parallel.
- Uses higher-valency cells with more than two inputs.

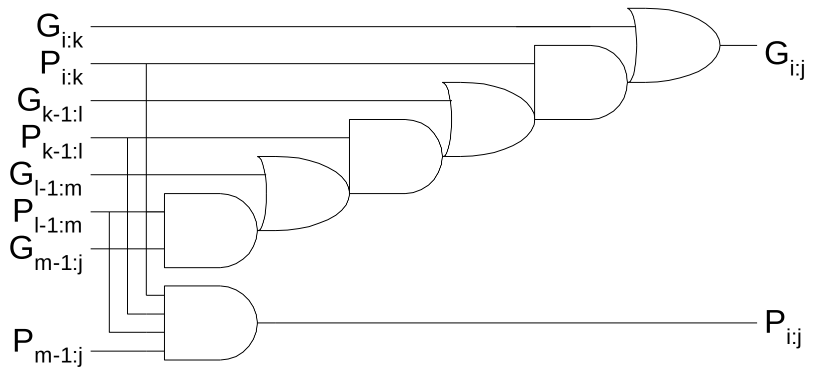


CLA PG Diagram

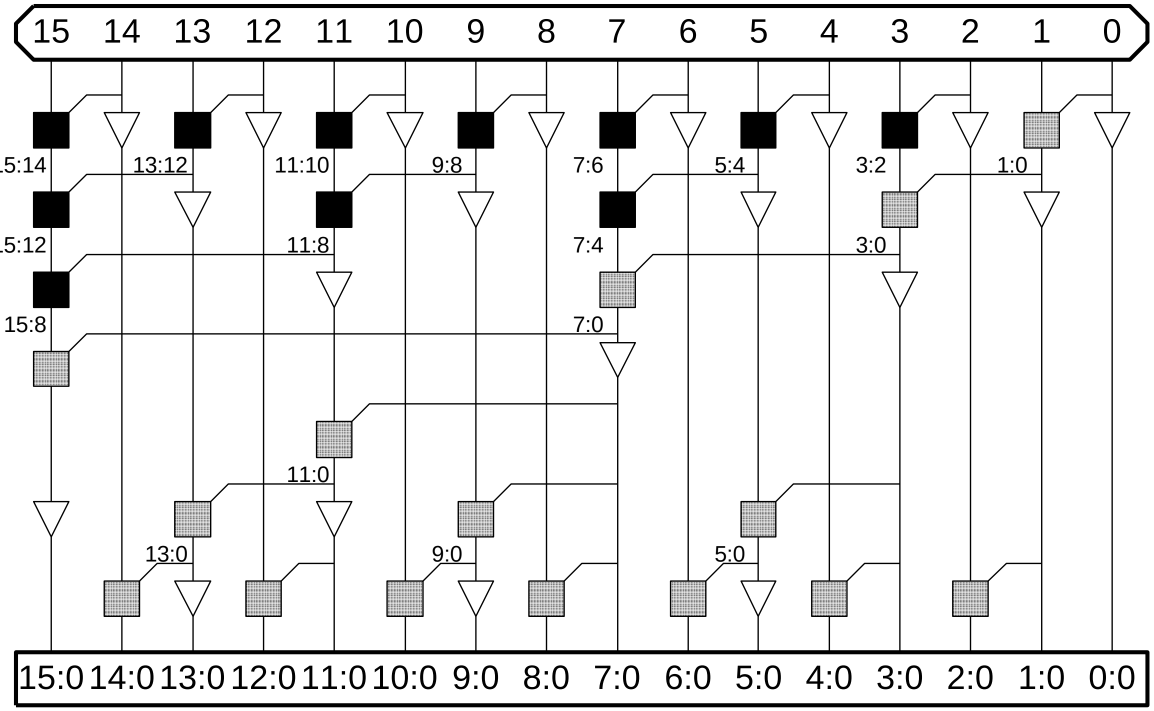


Higher-Valency Cells

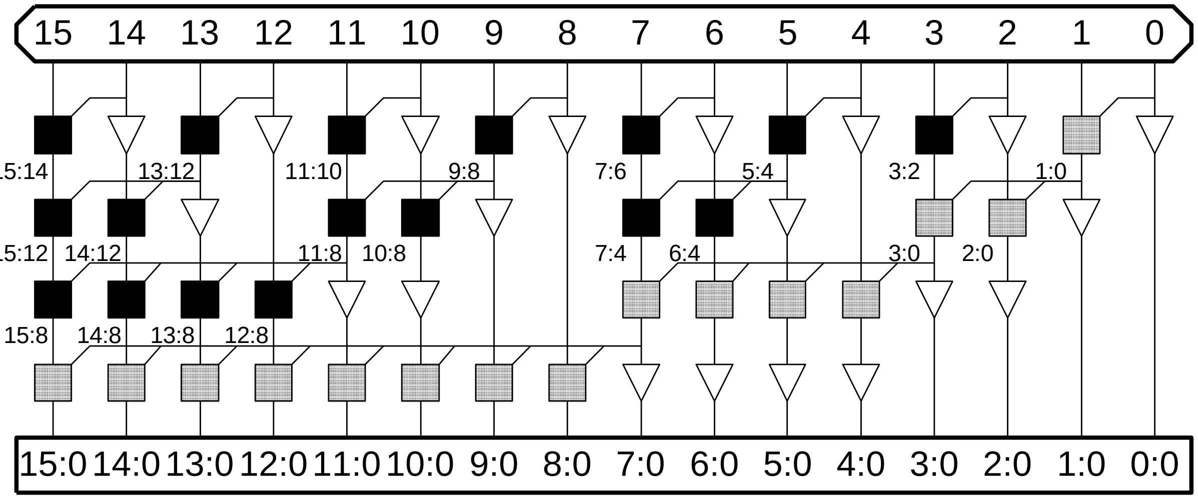




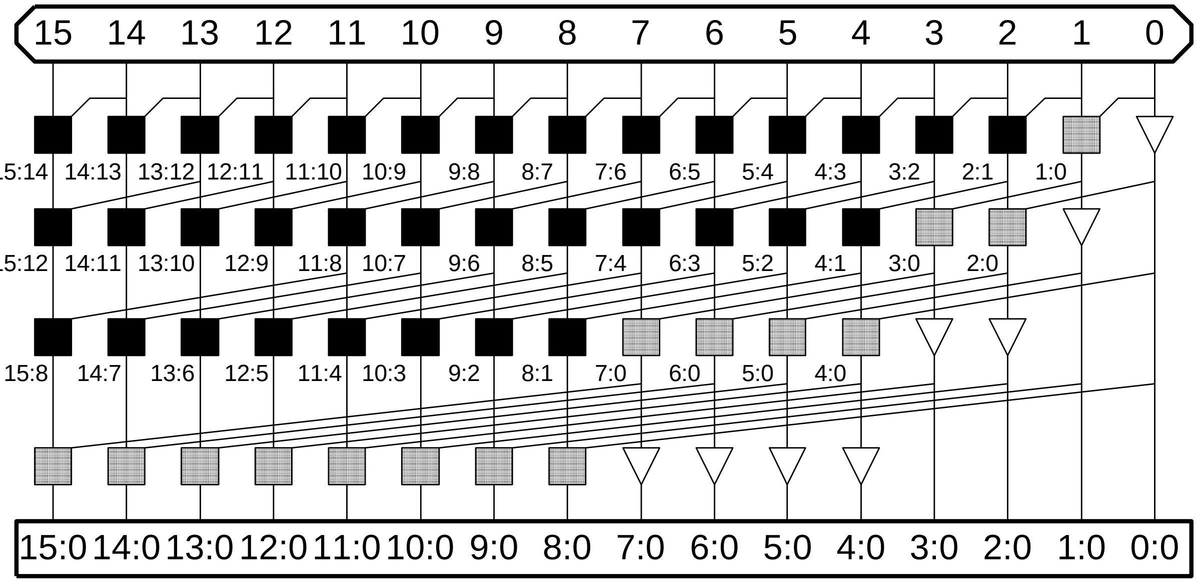
Brent-Kung


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14
13:12
11:10
9:8
7:6
5:4
3:2
1:0
15:12
11:8
7:4
3:0
15:8
7:0
11:0
13:0
9:0
5:0
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

Sklansky


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14
13:12
11:10
9:8
7:6
5:4
3:2
1:0
15:12
14:12
11:8
10:8
7:4
6:4
3:0
2:0
15:8
14:8
13:8
12:8
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

Kogge-Stone


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14 14:13 13:12 12:11 11:10 10:9 9:8 8:7 7:6 6:5 5:4 4:3 3:2 2:1 1:0
15:12 14:11 13:10 12:9 11:8 10:7 9:6 8:5 7:4 6:3 5:2 4:1 3:0 2:0
15:8 14:7 13:6 12:5 11:4 10:3 9:2 8:1 7:0 6:0 5:0 4:0
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

Summary

Adder architectures offer area / power / delay tradeoffs.

Choose the best one for your application.

Architecture	**Logic Levels**	**Max Fanout**	**Cells**
Carry-Ripple	N-1	1	N
Carry-Skip n=4	N/4 + 5	2	1.25N
Carry-Inc. n=4	N/4 + 2	4	2N
Brent-Kung	$2\log_2 N - 1$	2	2N
Sklansky	$\log_2 N$	N/2 + 1	$0.5\, N\log_2 N$
Kogge-Stone	$\log_2 N$	2	$N\log_2 N$

End

Thank you very much.