Where are we?

- Subsystem Design
  - Registers and Register Files
  - Adders and ALUs
  - Simple ripple carry addition
  - Transistor schematics
  - Faster addition
  - Logic generation
  - How it fits into the datapath

Data Path Design

- Block-diagram style data path description

Bit Slice Design

Tile identical processing elements

- Layout Reality

Bit Slice Design

Tile identical processing elements

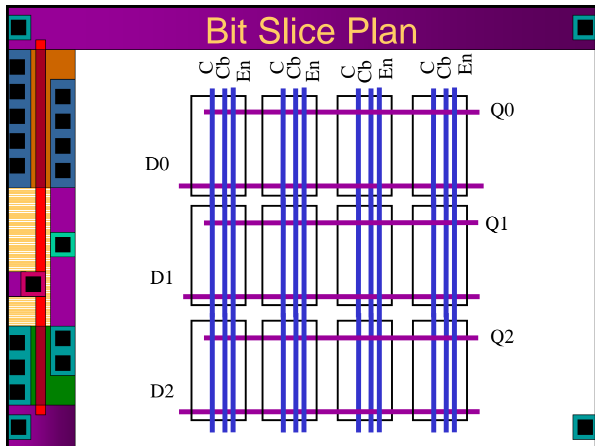
- Layout Reality

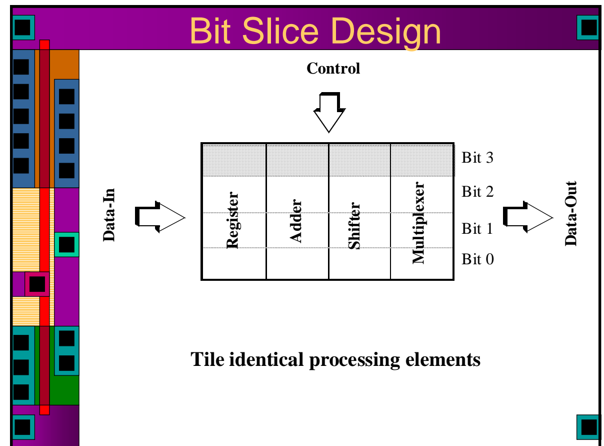
Bit Slice Plan

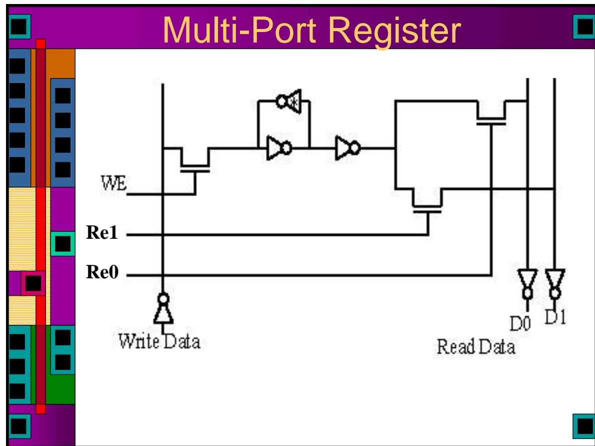
- Recall planning a DFF to make a register
  - Inputs on top in M2
  - Outputs on bottom in M2
  - Clock and Clock-bar routed horizontally in M1

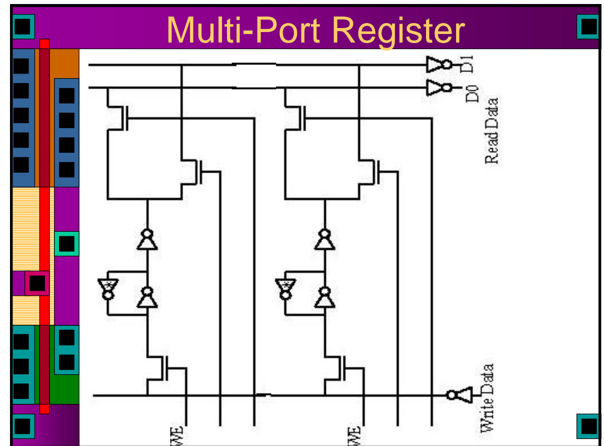
Bit Slice Plan

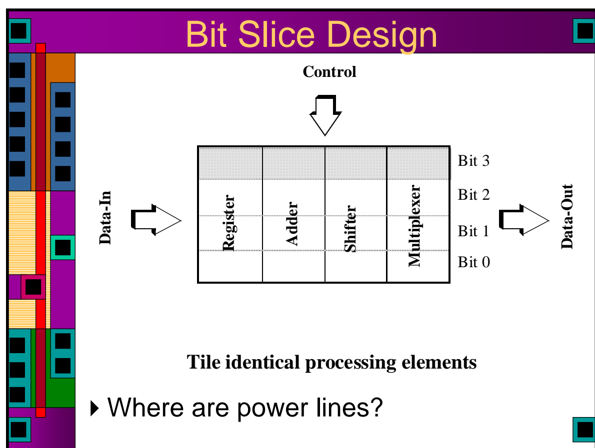
- Now extend this to a register file
  - D inputs go to all cells
    - Can select one register for writing by controlling the clock
  - Q outputs go all the way through the register file
  - Each cell can drive Q from enabled inverter
    - Now you can select one register for reading by selecting which cell is driving its output

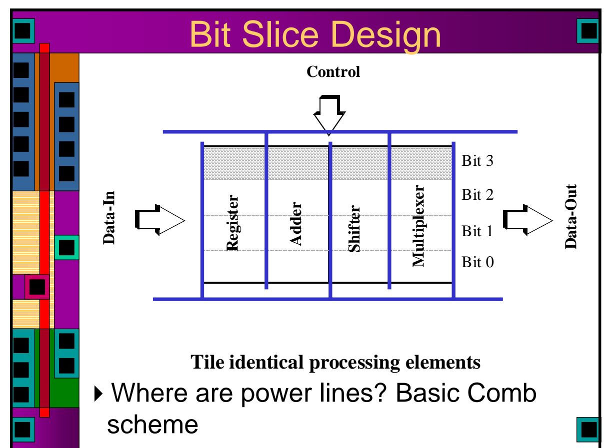

Bit Slice Plan
C
Cb
En
Q0
D0
Q1
D1
Q2
D2


Bit Slice Design
Control
Data-In
Register
Adder
Shifter
Multiplexer
Bit 3
Bit 2
Bit 1
Bit 0
Data-Out
Tile identical processing elements


Multi-Port Register
WE
Re1
Re0
Write Data
D0
D1
Read Data


Multi-Port Register
D1
D0
Read Data
WE
WE
Write Data


Bit Slice Design
Control
Data-In
Register
Adder
Shifter
Multiplexer
Bit 3
Bit 2
Bit 1
Bit 0
Data-Out
Tile identical processing elements
Where are power lines?


Bit Slice Design
Control
Data-In
Register
Adder
Shifter
Multiplexer
Bit 3
Bit 2
Bit 1
Bit 0
Data-Out
Tile identical processing elements
Where are power lines? Basic Comb scheme

Chip-Wide View of Power

- Power Routing is a global chip-wide issue
- Here's another approach
- Note the Vdd and Gnd pads
- Global rings with combs for regions of the chip

Chip-Wide View of Power

- Power Routing is a global chip-wide issue
- Here's another approach
- Note the Vdd and Gnd pads
- Global rings with combs for regions of the chip

Core power routing

Core power routing

Chip-Wide View of Power

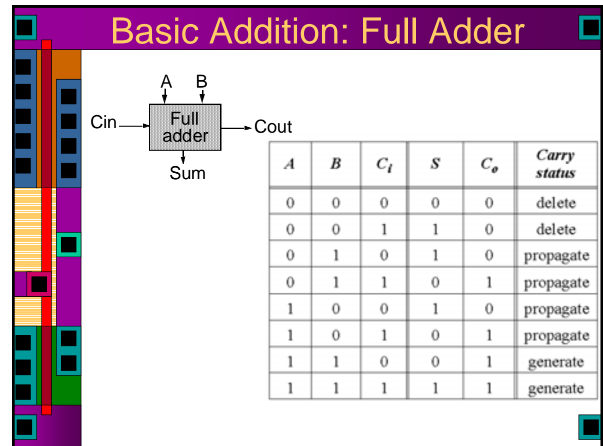
- Another view of the same issue
- Watch out for routing blockages!

A Tweak on the Scheme

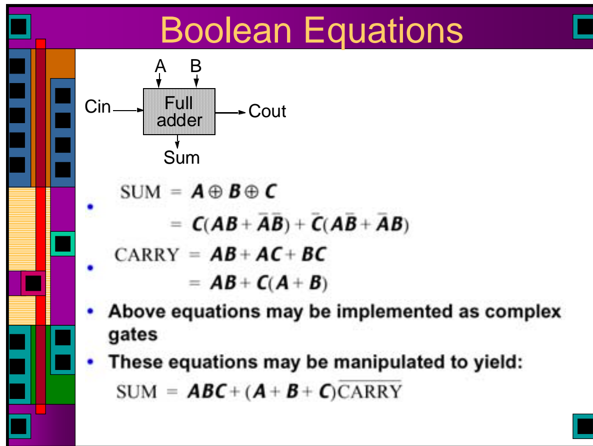
- Same basic scheme
- But with no internal jumpers
- Jumpers are restricted to outer loops

Adders Etc.

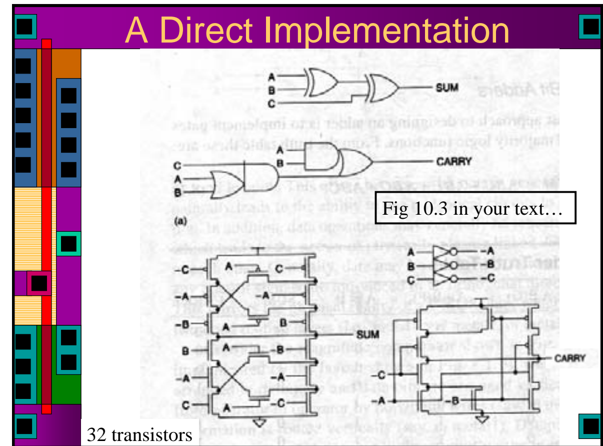
- Check out Chapter 10 in your text

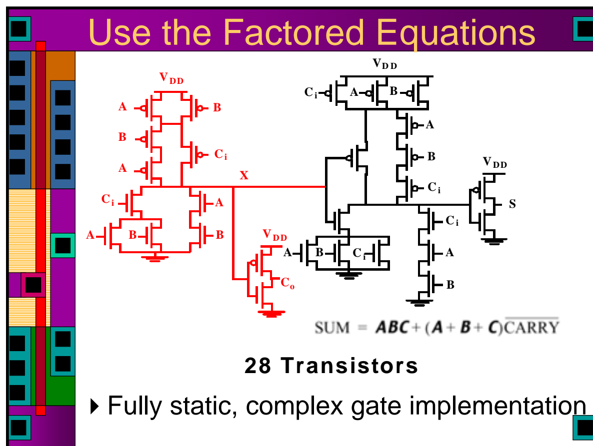


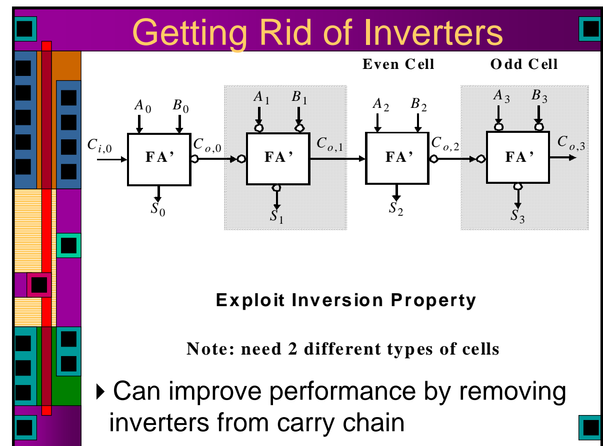
A	B	C_i	S	C_o	Carry status
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

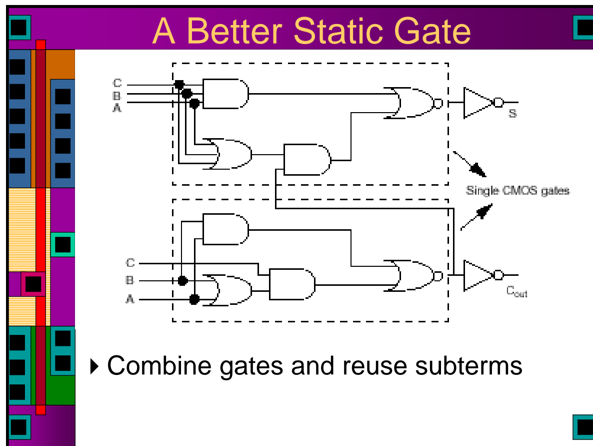


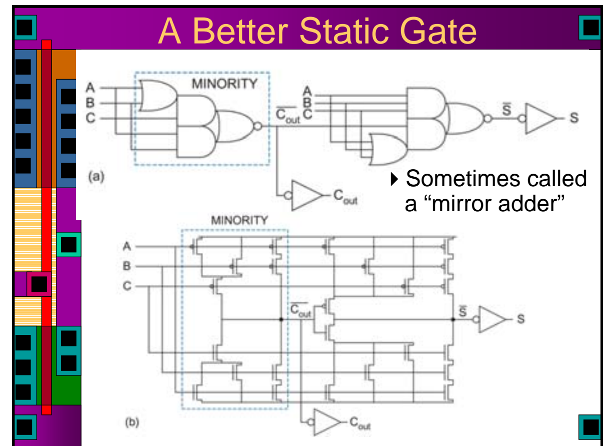
- $\text{SUM} = A \oplus B \oplus C$
  $= C(AB + \overline{A}\overline{B}) + \overline{C}(A\overline{B} + \overline{A}B)$
- $\text{CARRY} = AB + AC + BC$
  $= AB + C(A + B)$
- Above equations may be implemented as complex gates
- These equations may be manipulated to yield:
  $\text{SUM} = ABC + (A + B + C)\overline{\text{CARRY}}$





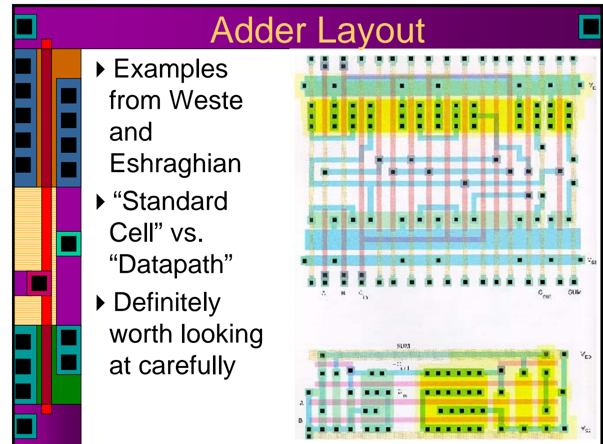


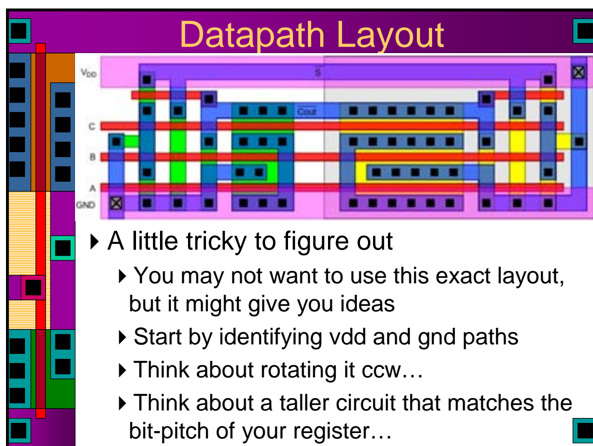


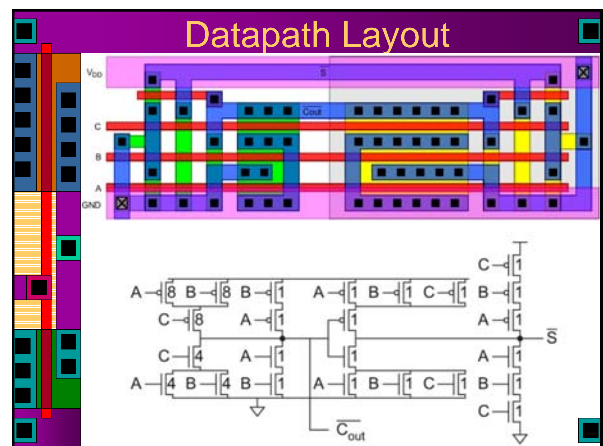


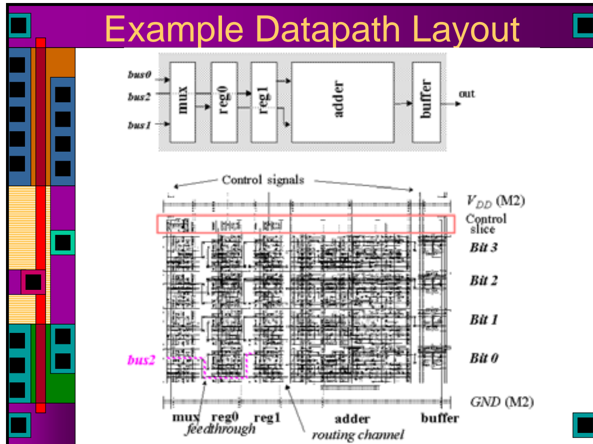
Mirror Adder Considerations

- Feed the Carry-In to the inner inputs so the internal capacitance is already discharged
- Make all transistors whose gates are connected to Cin and carry logic minimum size – minimizes branching effort on critical path (carry out)
- Determine gate widths by Logical Effort – reduce effort from C to CoutB at the expense of Sum
- Use relatively large transistors on critical path so that stray wiring cap is a small fraction of overall cap





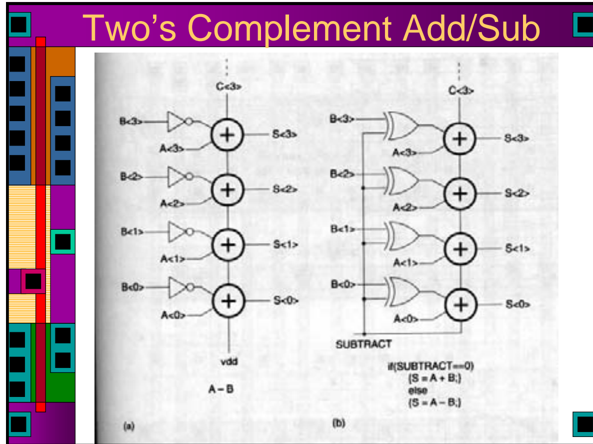




Addition and Subtraction

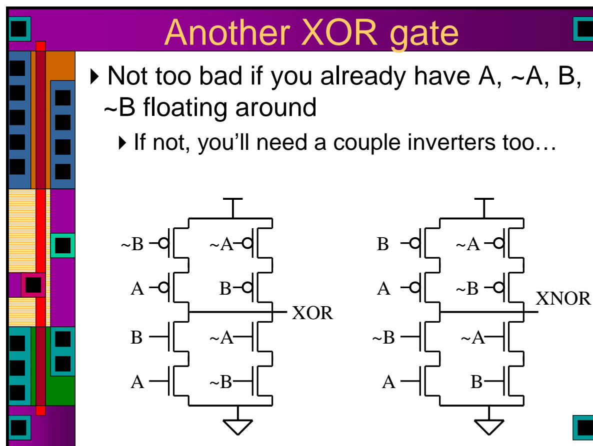
- Remember back to your logic design class
  - Add the two's complement to subtract
  - Take two's complement by inverting all the bits and adding one
  - Use the carry-in to add one
  - Use an XOR to invert or not

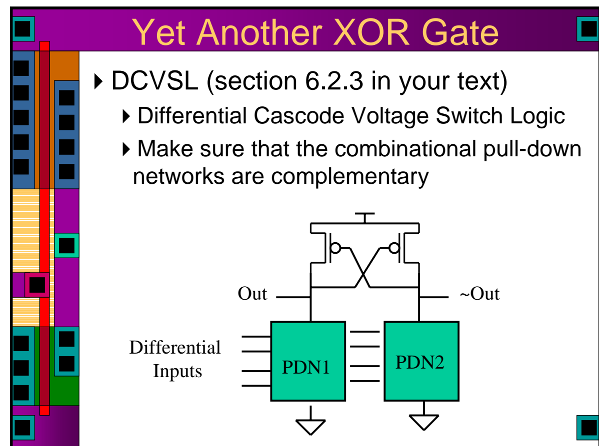
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0



Aside: XOR Gates

- Slightly tricky gate, ~AB + A~B
- Lots of different schematics…





DCVSL XOR/XNOR

Out ~Out B ~B B ~B A ~A

- Generates both XOR/XNOR
- Still static, but might be slower than others

Another DCVSL Example

Out ~Out D A E B C ~D ~E ~B ~A ~C

- Pull-down stacks must be complementary

DCVSL Large XOR

Four-input XOR aka odd parity

Out ~Out ~D D ~D D C ~C C ~C B ~B B ~B A ~A

DCVSL Large XOR

Four-input XOR aka odd parity

Out ~Out ~D D ~D D C ~C C ~C B ~B B ~B A ~A

DCVSL Large XOR

Four-input XOR aka odd parity

Out ~Out ~D D ~D D C ~C C ~C B ~B B ~B A ~A

Transmission Gate XOR

A P_1 P_2 B N_1 N_2 ~A A⊕B

- Tiny, clever circuit
  - If A is high, N1, P1 act like inverter
  - If A is low, B is passed to the output through transmission gate

Transmission Gate Adder

- **Truth Table**

A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- **When $A \oplus B = 0$, SUM $= C$, and Carry $= B$.**
- **When $A \oplus B = 1$, SUM $= \bar{C}$, and Carry $= C$.**
- **Using the 6T XOR, this full adder uses 18T.**

Another Version

Setup

Sum Generation

Carry Generation

Yet Another Version

An Example Layout…

- Not the same style we're used to seeing…

More Pass Transistors

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

Speeding Up Addition

- It all comes back to the carry circuit
  - Ripple carry delay goes from low-order to high-order bit
  - This determines the speed of the addition

Delay is proportional to *n*

- Many many ways to speed up the carry calculation

Section 10.2.2 in your text

Carry Lookahead

- A carry out C_i is generated from bit position i, when both A_i and B_i are '1' i.e. $G_i = A_i B_i$
- A carry in is propagated to the carry out at bit position i when either A_i or B_i is '1' (if both are '1' G_i will cover) e.g. $P_i = A_i \oplus B_i$

Sum = P ⊕ Ci

complex gate (6T)

- Thus the carryout, $C_i = G_i + P_i C_{i-1}$
- Key is that the carry depends ONLY on A and B, not the carry-in
  - Catch is that the gates have large fan-in

Carry Lookahead

- Restated: Ci = Gi + Pi C(i-1)
- C0 = G0 + P0 Cin
- C1 = G1 + P1 C0
  = G1 + P1(G0 + P0 Cin)
  = G1 + P1 G0 + P1 P0 Cin
- C2 = G2 + P2G2 + P2P1G0 + P2P1P0Cin
- C3 = G1 + P3G2 + P3P2G1 + P3P2P1G0
  + P3P2P1P0Cin
- Or C3 = G3 + P3(G2 +P2(G1 + P1(G0 + P0 Cin)))

Carry Lookahead

A_0,B_0 A_1,B_1 ... A_{N-1},B_{N-1}

$C_{i,0}$ P_0 $C_{i,1}$ P_1 $C_{i,N-1}$ P_{N-1}

- The C equations get larger with each stage
  - Usually do lookahead in small blocks (I.e. 4) and the combine in a tree

Carry Lookahead Logic

Carry Generation

Sum Generator

PG Generator

Fast Carry Lookahead Logic

Pseudo-nMOS

Uses lots of current!

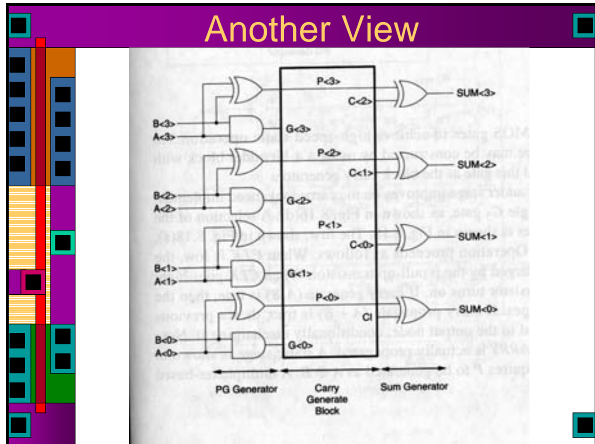
Another Version

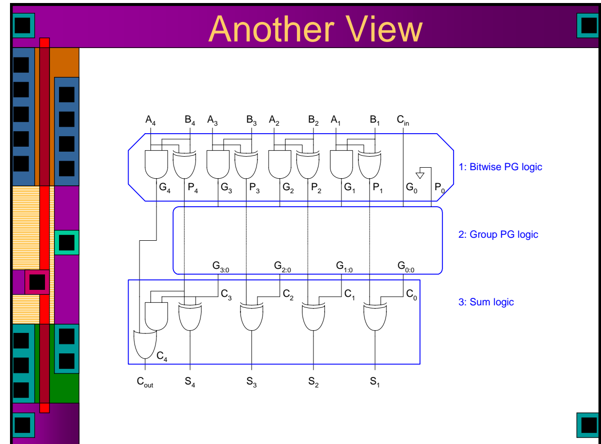
V_{DD}

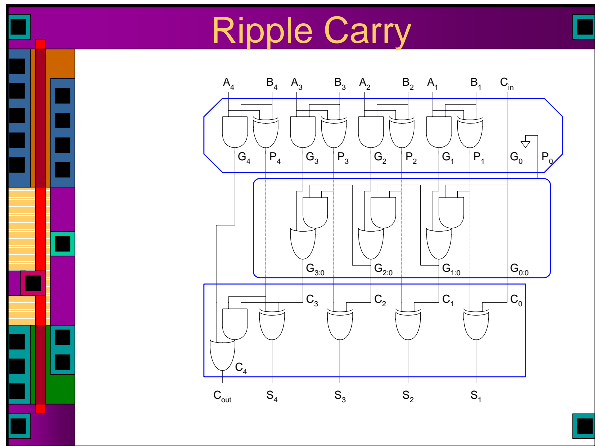
G_3 G_2 G_1 G_0

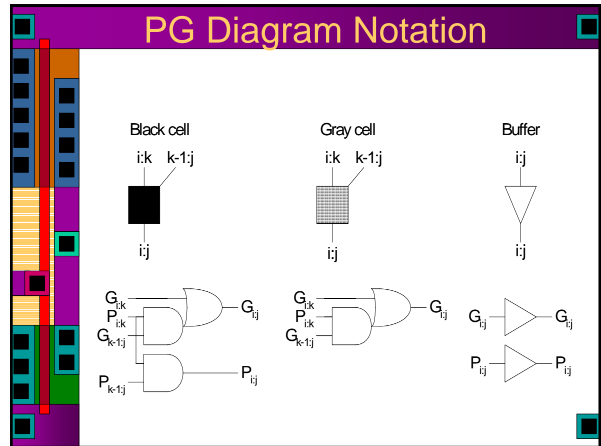
$C_{i,0}$ $C_{o,3}$

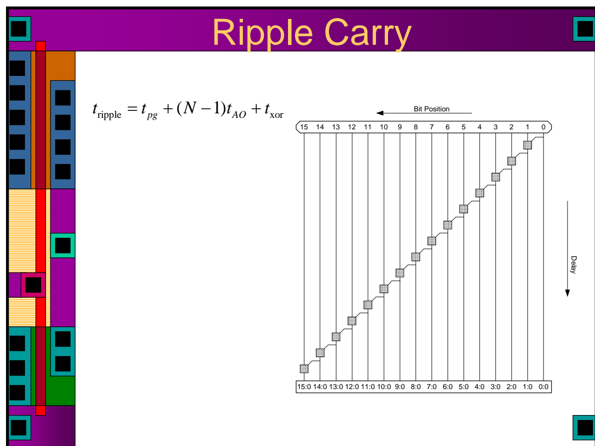
P_0 P_1 P_2 P_3

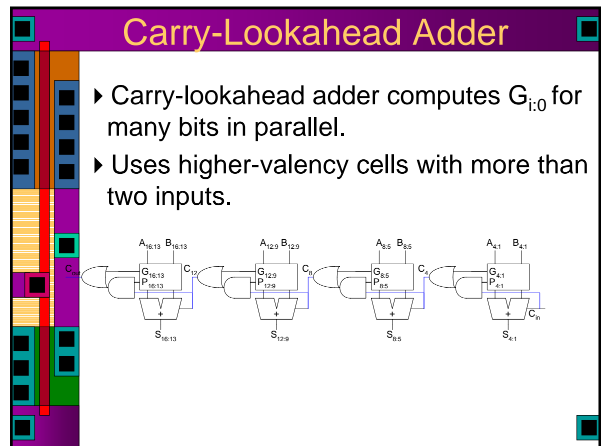

Another View
P<3>
C<2>
SUM<3>
B<3>
A<3>
G<3>
P<2>
C<1>
SUM<2>
B<2>
A<2>
G<2>
P<1>
C<0>
SUM<1>
B<1>
A<1>
G<1>
P<0>
CI
SUM<0>
B<0>
A<0>
G<0>
PG Generator
Carry Generate Block
Sum Generator

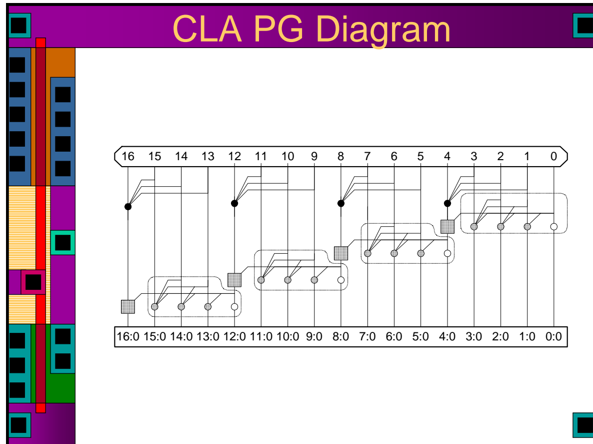

Another View
1: Bitwise PG logic
2: Group PG logic
3: Sum logic

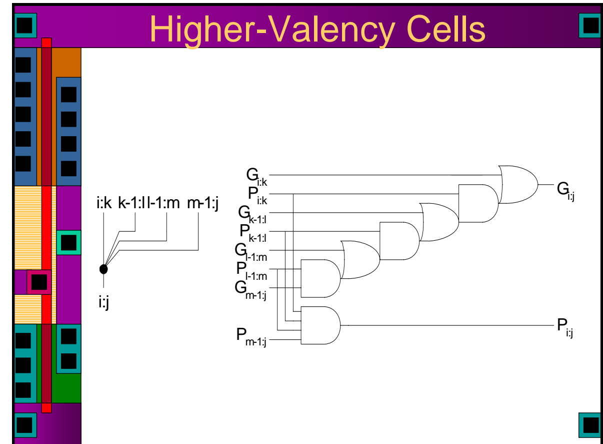

Ripple Carry

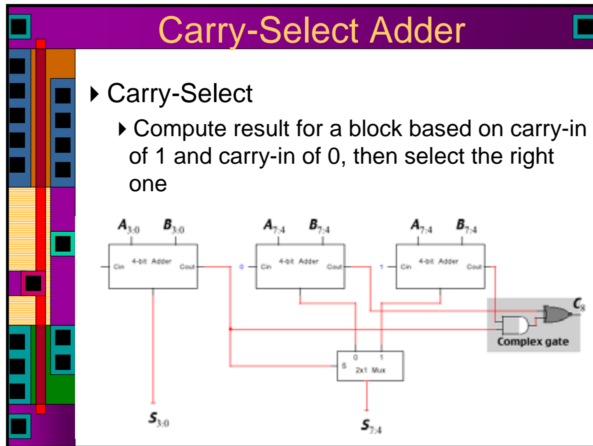

PG Diagram Notation
Black cell
Gray cell
Buffer

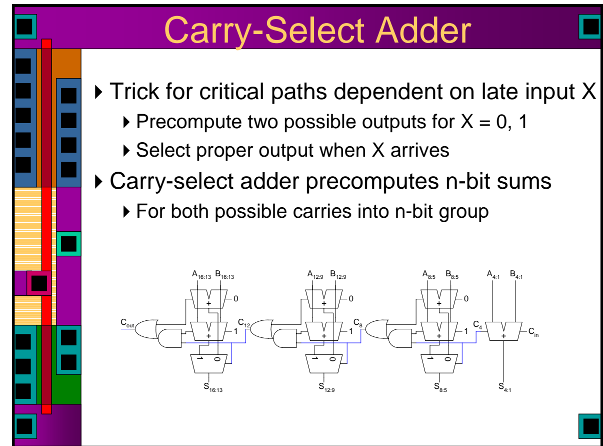

Ripple Carry
$t_{ripple} = t_{pg} + (N-1)t_{AO} + t_{xor}$
Bit Position
Delay

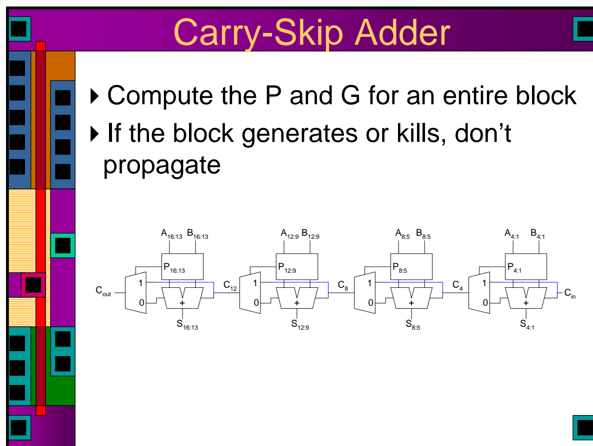

Carry-Lookahead Adder
Carry-lookahead adder computes $G_{i:0}$ for many bits in parallel.
Uses higher-valency cells with more than two inputs.

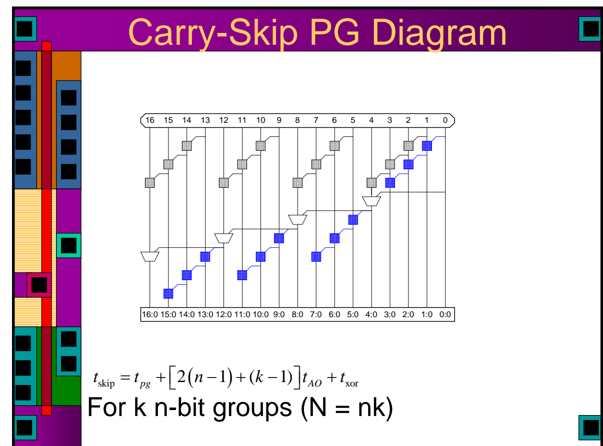

CLA PG Diagram
16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
16:0 15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0


Higher-Valency Cells
i:k k-1:l l-1:m m-1:j
i:j
$G_{i:k}$
$P_{i:k}$
$G_{k-1:l}$
$P_{k-1:l}$
$G_{l-1:m}$
$P_{l-1:m}$
$G_{m-1:j}$
$P_{m-1:j}$
$G_{i:j}$
$P_{i:j}$


Carry-Select Adder
Carry-Select
Compute result for a block based on carry-in of 1 and carry-in of 0, then select the right one
$A_{3:0}$ $B_{3:0}$
$A_{7:4}$ $B_{7:4}$
$A_{7:4}$ $B_{7:4}$
4-bit Adder
Cin
Cout
2x1 Mux
Complex gate
C_8
$S_{3:0}$
$S_{7:4}$


Carry-Select Adder
Trick for critical paths dependent on late input X
Precompute two possible outputs for X = 0, 1
Select proper output when X arrives
Carry-select adder precomputes n-bit sums
For both possible carries into n-bit group
$A_{16:13}$ $B_{16:13}$
$A_{12:9}$ $B_{12:9}$
$A_{8:5}$ $B_{8:5}$
$A_{4:1}$ $B_{4:1}$
C_{out}
C_{12}
C_8
C_4
C_{in}
$S_{16:13}$
$S_{12:9}$
$S_{8:5}$
$S_{4:1}$

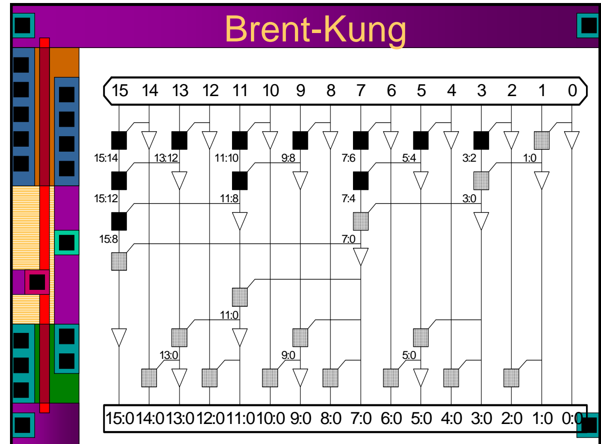

Carry-Skip Adder
Compute the P and G for an entire block
If the block generates or kills, don't propagate
$A_{16:13}$ $B_{16:13}$
$A_{12:9}$ $B_{12:9}$
$A_{8:5}$ $B_{8:5}$
$A_{4:1}$ $B_{4:1}$
$P_{16:13}$
$P_{12:9}$
$P_{8:5}$
$P_{4:1}$
C_{out}
C_{12}
C_8
C_4
C_{in}
$S_{16:13}$
$S_{12:9}$
$S_{8:5}$
$S_{4:1}$


Carry-Skip PG Diagram
16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
16:0 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
$t_{skip} = t_{pg} + \left[2(n-1) + (k-1)\right] t_{AO} + t_{xor}$
For k n-bit groups (N = nk)

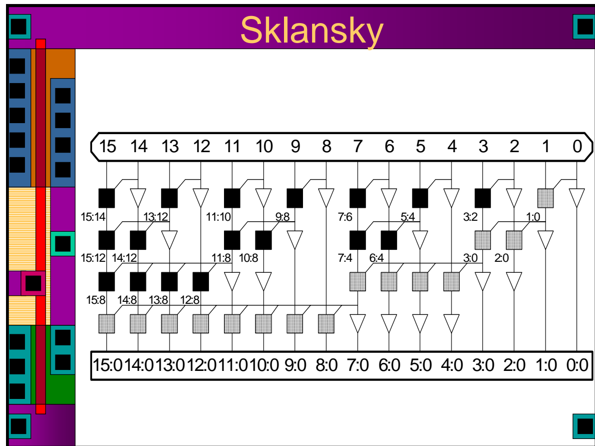
Tree Adder

- If lookahead is good, lookahead across lookahead!
  - Recursive lookahead gives O(log N) delay
- Many variations on tree adders

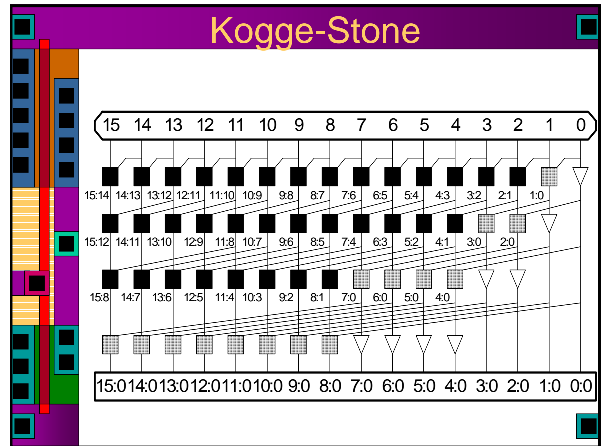
Brent-Kung



Sklansky



Kogge-Stone



Manchester Carry Chain

- Instead of changing the architecture of the adder, use a clever circuit to ripple the carry more effectively

- **Propagate and generate signals computed in about two gate delays**
- **Active low carry is propagated through a chain of transmission gates**
- **The three shaded areas of the circuit are mutually exclusive, and represent P_i, G_i, and $\overline{P_i G_i}$.**

Alternate Implementation

- **The Manchester carry chain computation may also be implemented with a 2x1 mux.**

Four Bit Block

- Signal propagation through a chain of transmission gates must be restored after about 4 gates

G_0 P_0 G_1 P_1 G_2 P_2 G_3 P_3 C_{-1} C_4 $\overline{P_0}$ $\overline{P_1}$ $\overline{P_2}$ $\overline{P_3}$ $\overline{C_0}$ $\overline{C_1}$ $\overline{C_2}$ $\overline{C_3}$

- A block propagate (bypass) circuit may be added to further improve performance on wide adders

Summary

Adder architectures offer area / power / delay tradeoffs.

Choose the best one for your application.

Architecture	Classification	Logic Levels	Max Fanout	Tracks	Cells
Carry-Ripple		N-1	1	1	N
Carry-Skip n=4		N/4 + 5	2	1	1.25N
Carry-Inc. n=4		N/4 + 2	4	1	2N
Brent-Kung	(L-1, 0, 0)	$2\log_2 N - 1$	2	1	2N
Sklansky	(0, L-1, 0)	$\log_2 N$	N/2 + 1	1	$0.5 N\log_2 N$
Kogge-Stone	(0, 0, L-1)	$\log_2 N$	2	N/2	$N\log_2 N$

Design as Trade-Off

- Do you want speed or size?
  - There's always power to consider too…

How well does Synopsys do?

FIG 10.47 Area vs. delay of synthesized adders

- Design compiler using a 180nm library

What should you use?

- Ripple if timing allows
  - Compact, easy
- CLA or carry-skip work well for 8-16 bits
- For 32, and especially 64 bits tree adders are faster
- Adders designed and tiled by hand will be much smaller (and probably faster) than synthesized adders

Logic Functions

- Use the features of the full adder cell to generate logic functions
- Lots of other ideas in your text…

General Logic Generator

- 4x1 multiplexor can implement any function of two variables

4 S A B F

- Simply place the truth table for F on the inputs of the mux.
- The operands A and B will select the correct value of the function

One Possible MUX Version

I0 I1 I2 I3 Out
$\overline{S1}$ S1 $\overline{S0}$ S0

- Note: Two t-gates in series do not need the internal connection between p-fet and n-fet

Remember the Big Picture

Control
Data-In | Register | Adder | Shifter | Multiplexer | Data-Out
Bit 3, Bit 2, Bit 1, Bit 0

- We want things to stack up nicely in the datapath

Shifters

Right nop Left
A_i B_i
A_{i-1} B_{i-1}
Bit-Slice i

- Essentially a muxing operation… select the shift you want (section 10.8)

Barrel Shifter

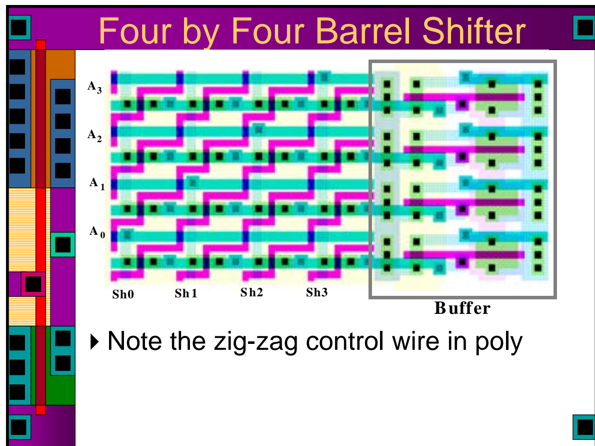
A_3 A_2 A_1 A_0 B_3 B_2 B_1 B_0
Sh0 Sh1 Sh2 Sh3
: Data Wire
: Control Wire

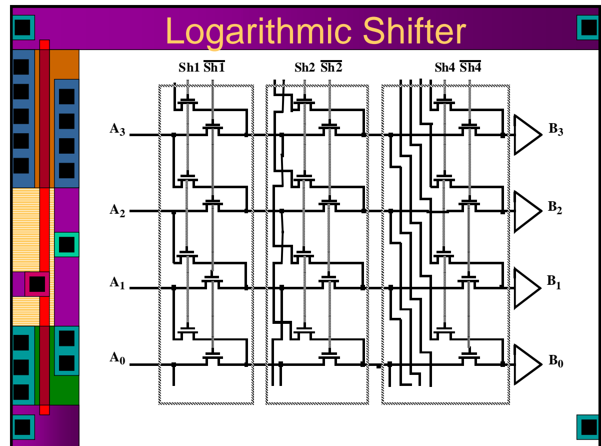
- Shift any number of bits in one shot
  - Clever layout is possible…
  - Lots of wiring…

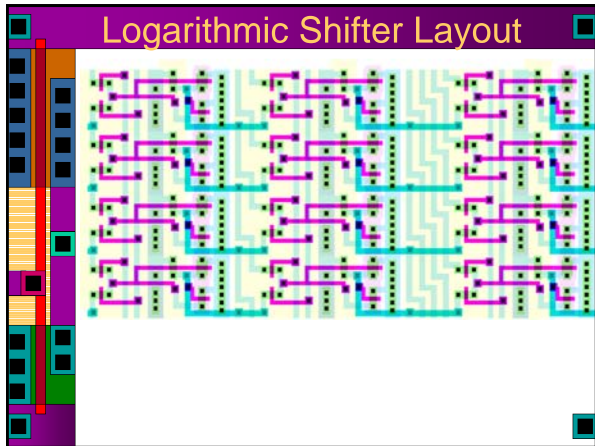
Barrel Shifter

A_3 A_2 A_1 A_0 B_3 A3 B_2 A3 B_1 A3 B_0 A2
Sh0 Sh1 Sh2 Sh3
: Data Wire
: Control Wire

- Shift any number of bits in one shot
  - Clever layout is possible…
  - Lots of wiring…


Four by Four Barrel Shifter
A3
A2
A1
A0
Sh0
Sh1
Sh2
Sh3
Buffer
Note the zig-zag control wire in poly


Logarithmic Shifter
Sh1 Sh1
Sh2 Sh2
Sh4 Sh4
A3
A2
A1
A0
B3
B2
B1
B0


Logarithmic Shifter Layout