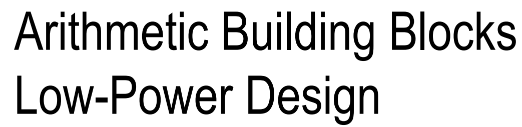


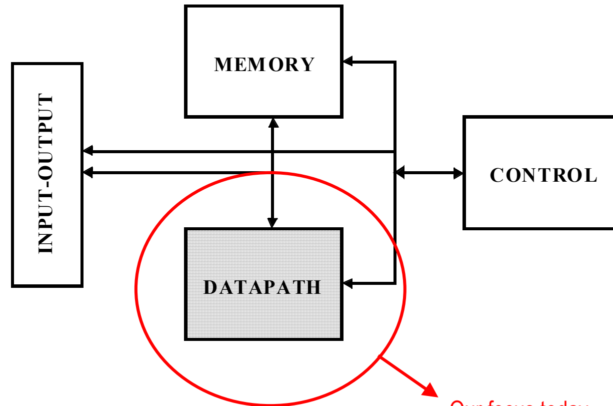
Lecture 12

18-322 Fall 2002

Textbook: [Chapter 7, Section 4.4]

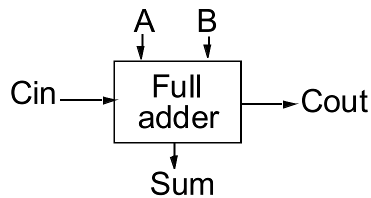
- Arithmetic building blocks
  - Adders
  - Multipliers
- Low-power design
  - Reducing power consumption
  - Data-path/Control circuitry

A Generic Digital Processor



Our focus today.
Also, main emphasis in the project!

The Binary Adder

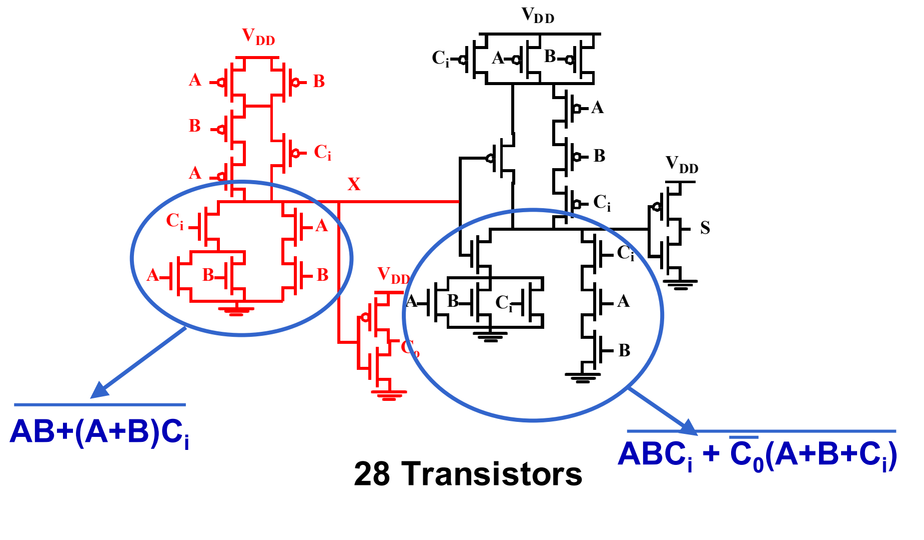


$$S = A \oplus B \oplus C_i$$

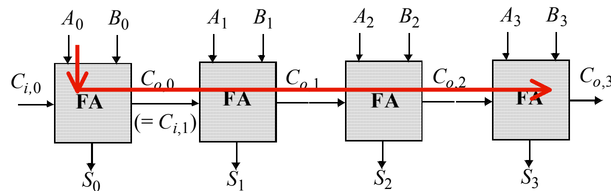
$$= A\overline{B}\overline{C}_i + \overline{A}B\overline{C}_i + \overline{A}\overline{B}C_i + ABC_i$$

$$C_o = AB + BC_i + AC_i$$

Static CMOS Full Adder: Is this a great implementation?



The Ripple-Carry Adder



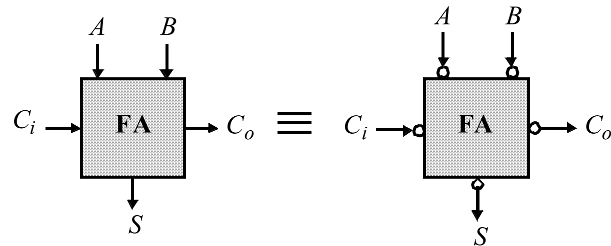
Worst case delay linear with the number of bits

$t_d = O(N)$

$$t_{adder} \approx (N-1)t_{carry} + t_{sum}$$

Goal: Make the fastest possible carry path circuit

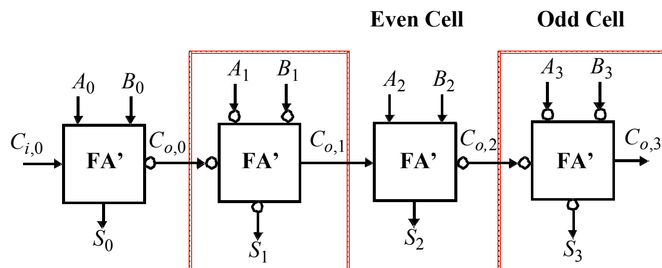
Inversion Property



$$\bar{S}(A, B, C_i) = S(\bar{A}, \bar{B}, \bar{C_i})$$

$$\bar{C_o}(A, B, C_i) = C_o(\bar{A}, \bar{B}, \bar{C_i})$$

Minimize Critical Path by Reducing the Number of Inverting Stages



Exploit Inversion Property

Note: Needs 2 different types of cells

Express Sum and Carry using P, G, D

Define 3 new variable which ONLY depend on A, B

Generate (G) = AB

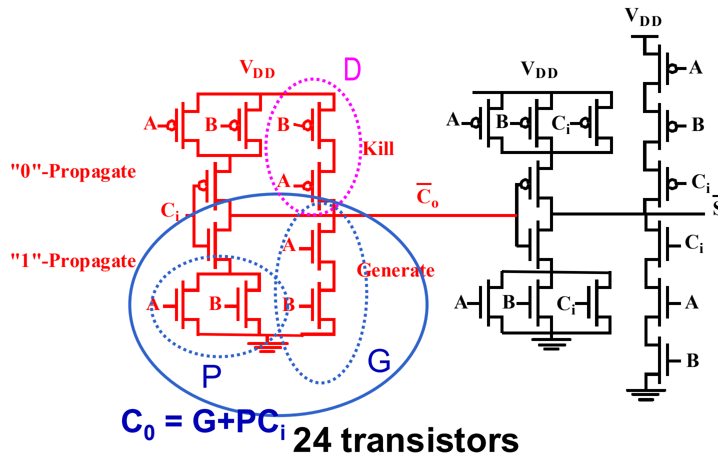
Propagate (P) = A ⊕ B

Delete = $\overline{A}\,\overline{B}$

$$C_o(G, P) = G + PC_i$$
$$S(G, P) = P \oplus C_i$$

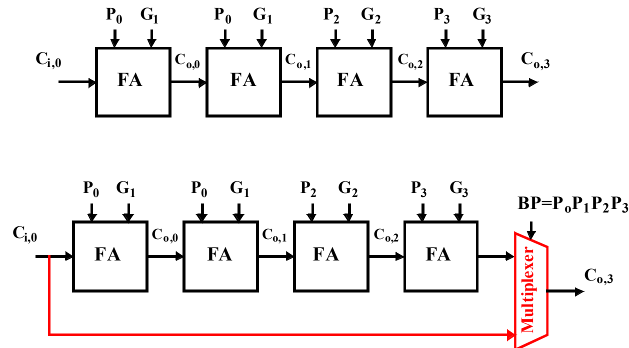
Can also derive expressions for *S* and C_o based on *D* and *P*

A better structure: the Mirror Adder



C_0 = G+PC_i **24 transistors**

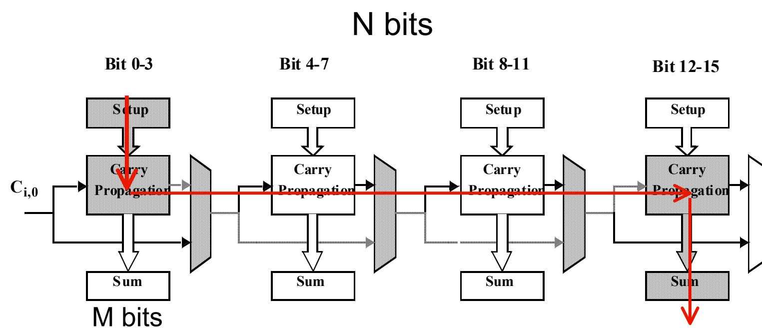
Carry-Bypass Adder



Idea: If (P0 and P1 and P2 and P3 = 1)
then $C_{o3} = C_0$, else "kill" or "generate".

Carry-Bypass Adder (cont.)

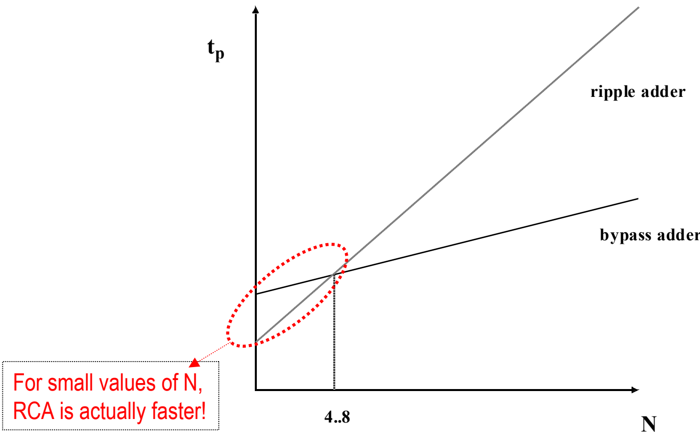
N bits



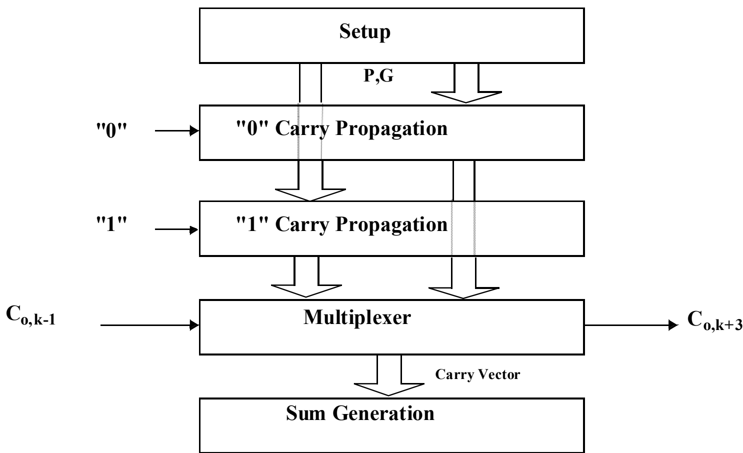
M bits

Note: the topological path worst-case delay is much higher than the true critical path!

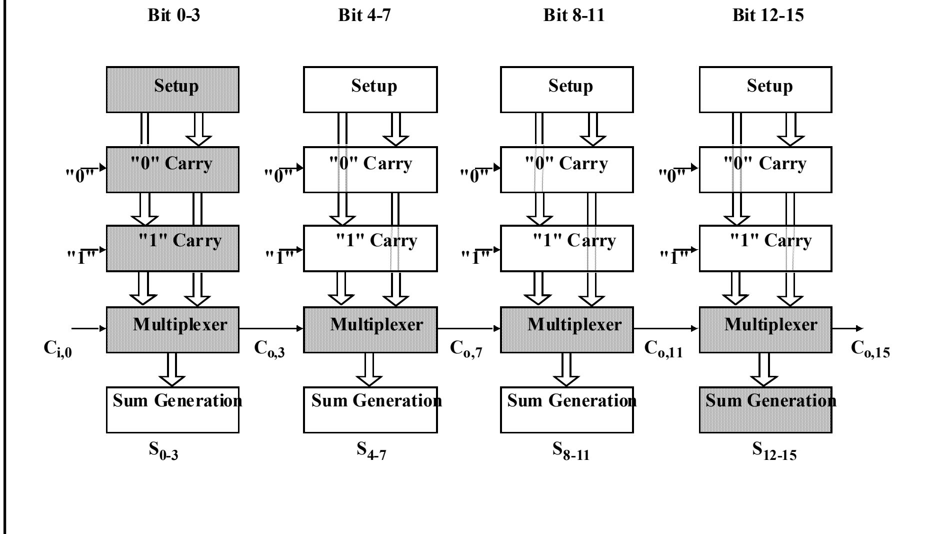
Carry Ripple vs. Carry Bypass



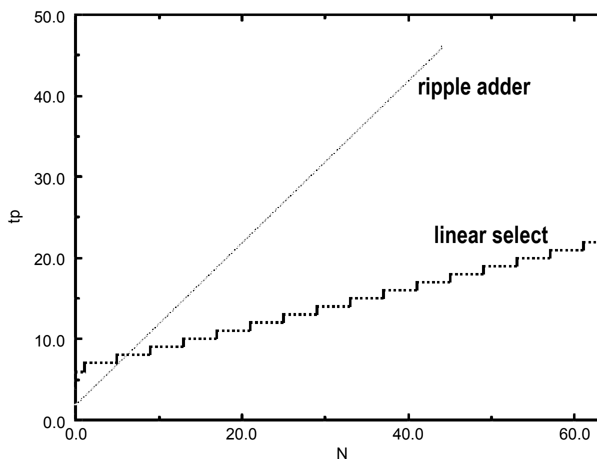
Carry-Select Adder



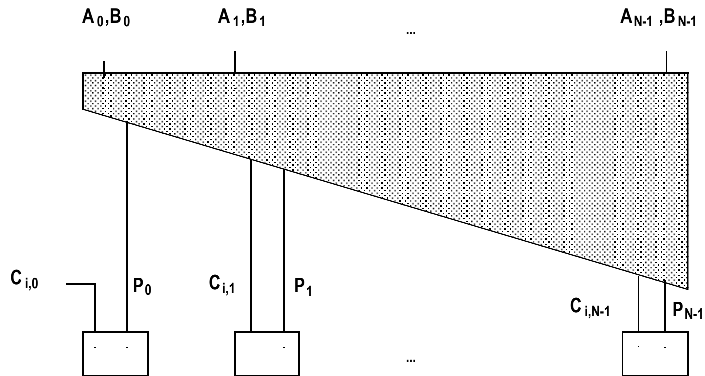
Carry Select Adder: Critical Path



Adder Delays - Comparison



Look Ahead - Basic Idea

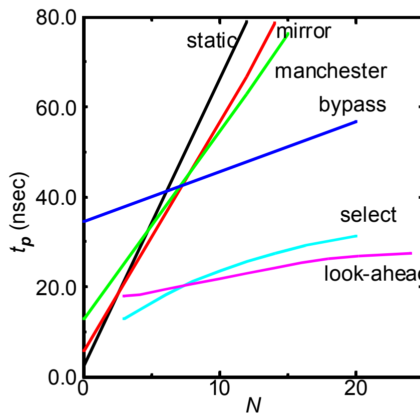


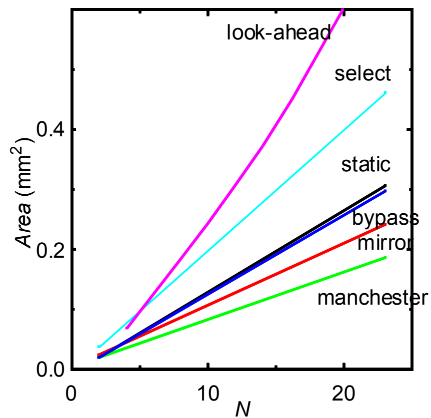
$C_{0,k} = G_k + P_k C_{0,k-1}$

$C_{0,k} = G_k + P_k (G_{k-1} + P_{k-1}(\ldots + P_1 (G_0 + P_0 C_{i,0})))$

The look ahead structure is useful for small values of N (N<4)

Design as a Trade-Off



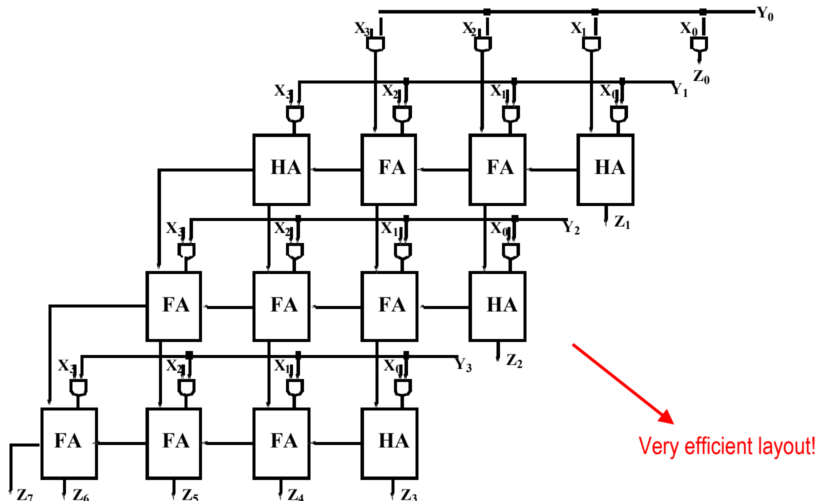


The Binary Multiplication

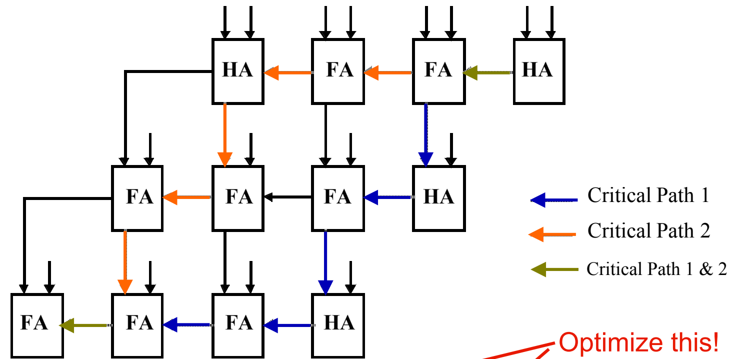
- Multiplications are *expensive* and *slow* operations
- Performance is often dominated by the speed at which multiplications can be executed

```
          1 0 1 0 1 0      multiplicand
×             1 0 1 1      multiplier
-------------------------
          1 0 1 0 1 0
        1 0 1 0 1 0        partial product
      0 0 0 0 0 0          (this corresponds to
+   1 0 1 0 1 0            a AND operation)
-------------------------
    1 1 1 0 0 1 1 1 0
```

The Array Multiplier



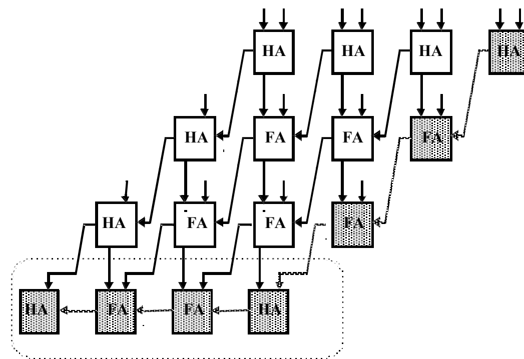
The MxN Array Multiplier — Critical Path



$$t_{mult} \approx [(M-1)+(N-2)]t_{carry} + (N-1)t_{sum} + (N-1)t_{and}$$

Optimization is very difficult since there exists several critical paths!

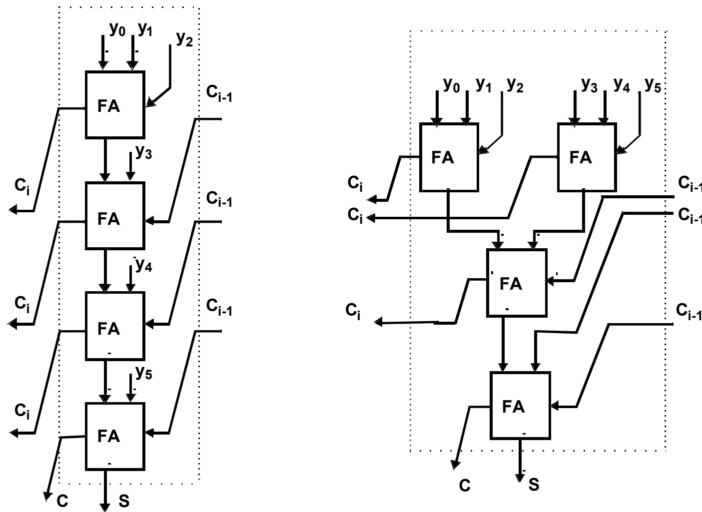
Carry-Save Multiplier



$$t_{mult} = (N-1)t_{carry} + (N-1)t_{and} + t_{merge}$$

Optimization becomes easier (unique critical path)!

Wallace-Tree Multiplier



Multipliers —Summary

- Optimization Goals Different compared to the Binary Adder
- Once Again: Identify the Critical Path
- Other possible techniques
  - Logarithmic versus Linear (Wallace Tree Multiplier)
  - Data encoding (Booth)
  - Pipelining

Outline

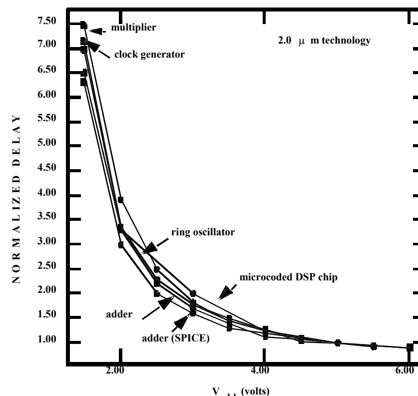
✓ Arithmetic building blocks
- Adders
- Multipliers

- Low-power design
  - Reducing power consumption
  - Data-path/Control circuitry

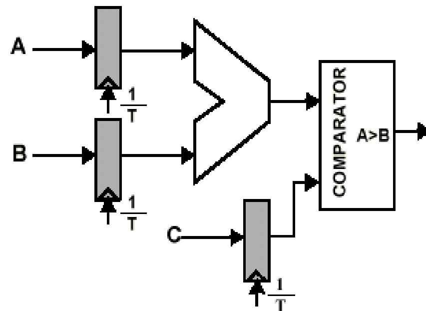
How about POWER? Ways to reducing power consumption

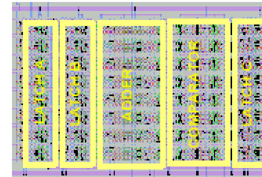
- Load capacitance (C_L)
  - Roughly proportional to the chip area
- Switching activity (avg. number of transitions/cycle)
  - Very data dependent
  - A big portion due to glitches (real-delay)
- Clock frequency (f)
  - Lowering only f decreases average power, but total energy is the same and throughput is worse

- Voltage supply (V_{DD})
  - Biggest impact



Using parallelism (1)



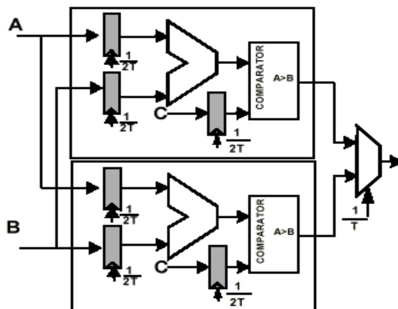


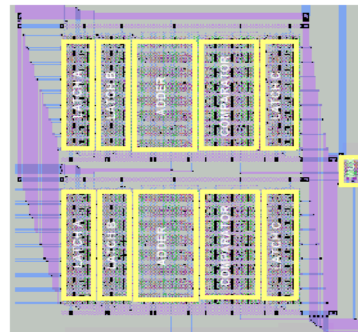
Area = 636 x 833 μ^2

$$P_{ref} = C_{ref} V_{DD}^2 f_{ref}$$

Assume: t_p = 25ns (worst-case, *all* modules) at V_{DD} = 5V

Using parallelism (2)

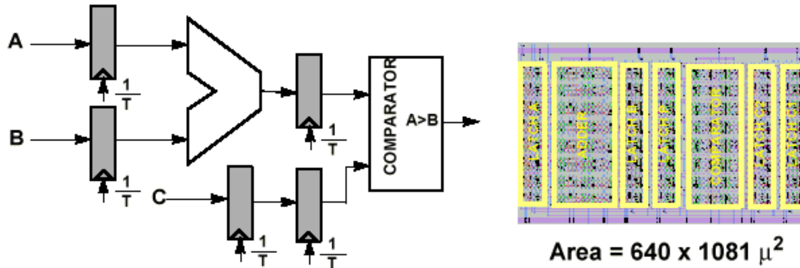




Area = 1476 x 1219 μ^2

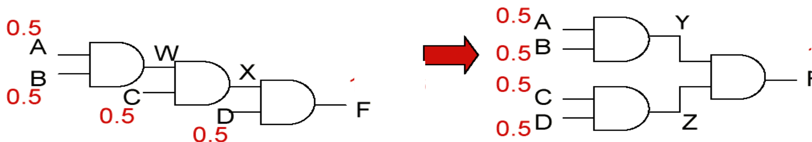
- C_{par} = 2.15C (extra-routing needed)
- f_{par} = f/2 ($t_{p,new}$ = ½ (50)ns => V_{DD} ~ 2.9V; $V_{DD,par}$ = 0.58 V_{DD})
- $P_{par} = C_{par} V_{DD}^2 f_{par}$ = 0.36 P_{ref}

Using pipelining



- C_{pipe} = 1.15C
- Delay decreases 2 times ($V_{DD,pipe}$ = 0.58 V_{DD})
- P_{pipe} = 0.39 P

Chain vs. balanced design

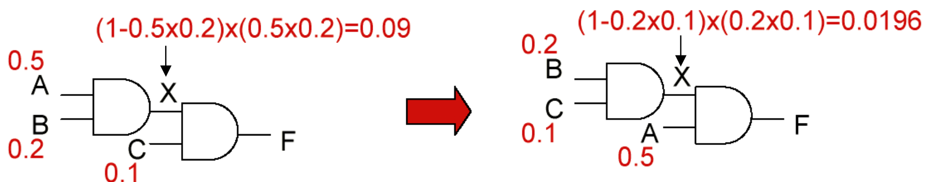


- *Question for you*:
  - Which of the two designs is more energy efficient?
    - Assume:
      - Zero-delay model
      - All inputs have a signal probability of 0.5
    - Hint: Calculate $p_{0\rightarrow1}$ for W, X and F

Low energy gates – transistor sizing

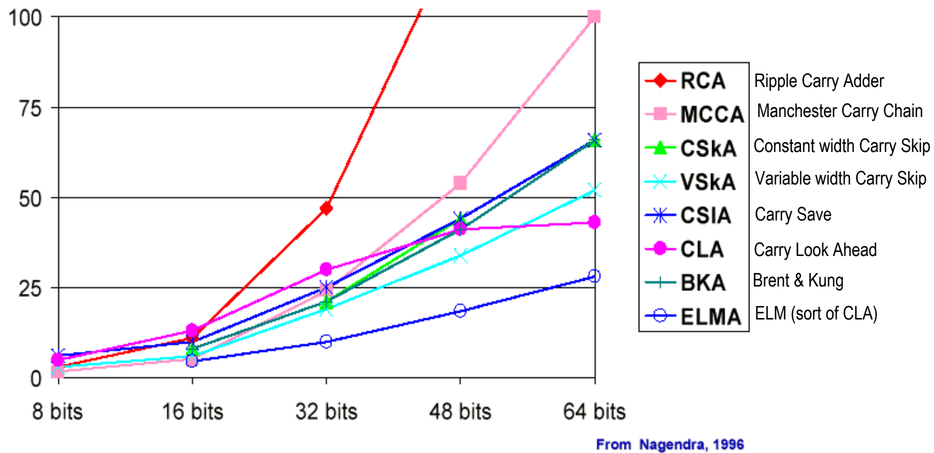
- Use the *smallest transistors* that satisfy the delay constraints
  - Increasing transistor size improves the speed but it also increases power dissipation (since the load capacitances increases)
    - Slack time - difference between required time and arrival time of a signal at a gate output
      - Positive slack - size down
      - Negative slack - size up
- Make gates that toggle more frequently smaller
- Slope engineering to reduce short circuit currents

Low energy gate netlists – pin ordering



- Better to postpone the introduction of signals with a high transition rate (signals with signal probability close to 0.5)

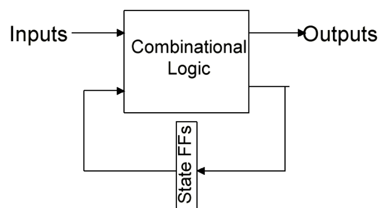
Power×Delay trade-off for various adders

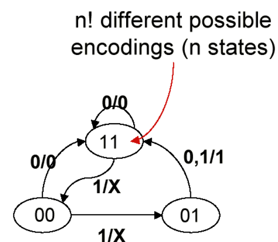


From Nagendra, 1996

Use a design that is fast enough and consumes the least power!

Control circuits

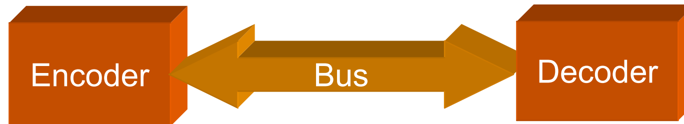




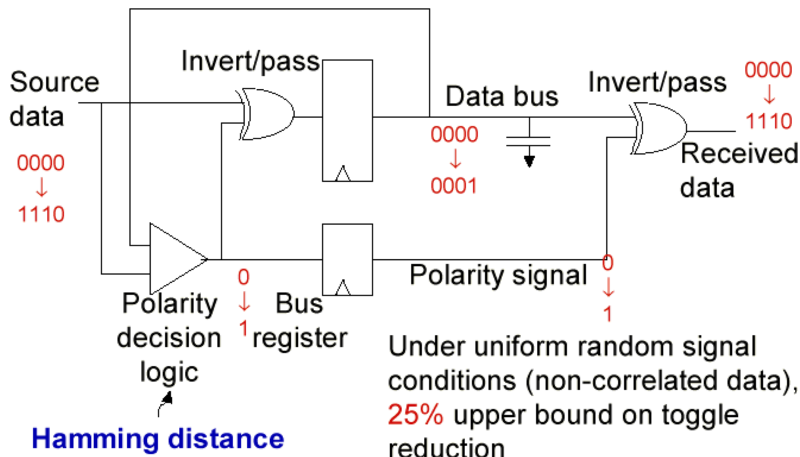
- State encoding has a big impact on the power efficiency
- Energy driven -> try to minimize number of bit transitions in the state register
  - Fewer transitions in state register
  - Fewer transitions propagated to combinational logic

Bus encoding

- Reduces number of bit toggles on the bus
- Different flavors
  - Bus-invert coding
    - Uses an extra bus line *invert*:
      - if the number of transitions is < $K/2$, invert = 0 and the symbol is transmitted as is
      - if the number of transitions is > $K/2$, invert = 1 and the symbol is transmitted in a complemented form
  - Low-weight coding
    - Uses *transition* signaling instead of *level* signaling



Bus invert coding



Source: M.Stan et al., 1994