MODULE 7

MODULARITY

Outline

- Background on Modular Design
  - Hierarchy, reuse, regularity
  - Architecture, bit-slicing
- Adder Design
- (Multiplier Design)
- Shifter Design
- Layout Strategies (regularity)
- Design as a Trade-Off

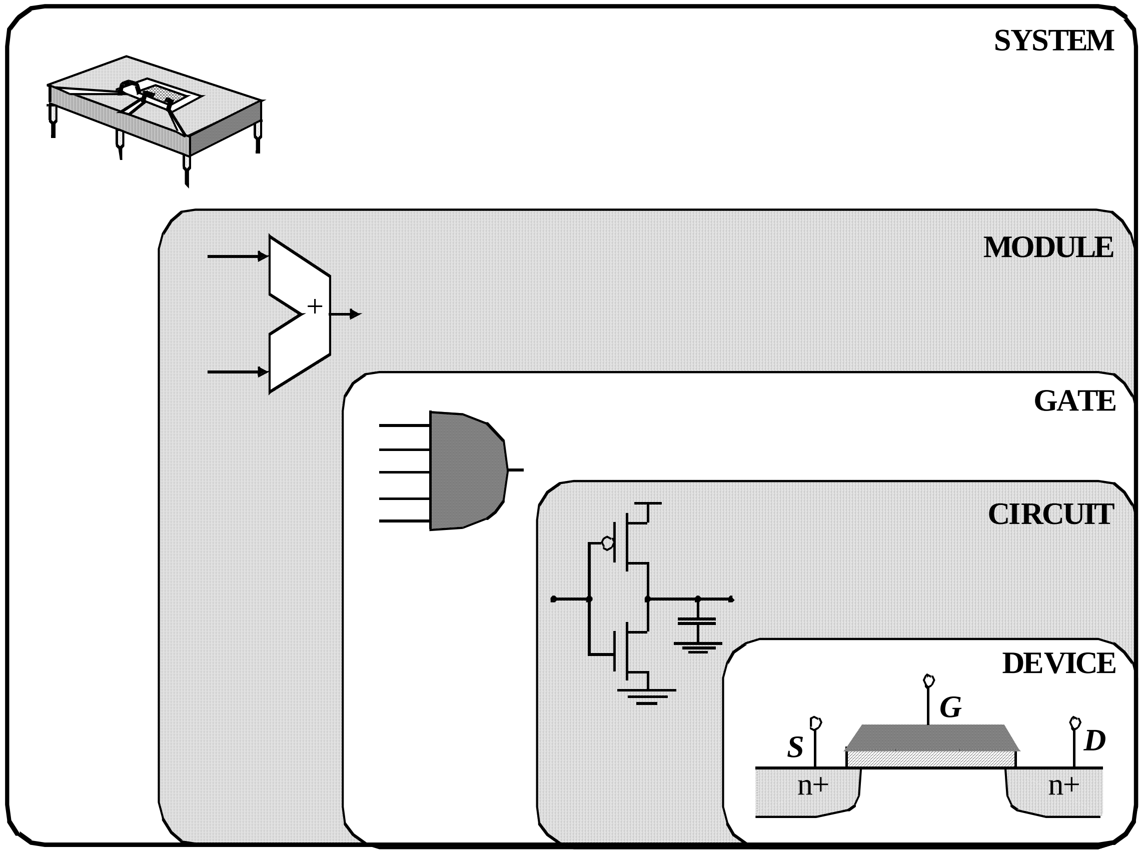
contains a lot of reminders

Get further appreciation of some system level design issues

Anonymous Transistors

- Large Chips have > 10^8 transistors
- Cannot consider each transistor separately during design
- Large chips have many "anonymous transistors" [Chris Verhoeven]
- Assembled from pre-designed building blocks
  - Such as on chip memory blocks
  - Or other blocks ranging from adders to complete sub systems [e.g. MP3 decoder, memory, …]
- Blocks are assembled from sub blocks, and so on
- Block types depend on architecture

Design Levels



7 - timing design

8 - modularity

6 - sequential

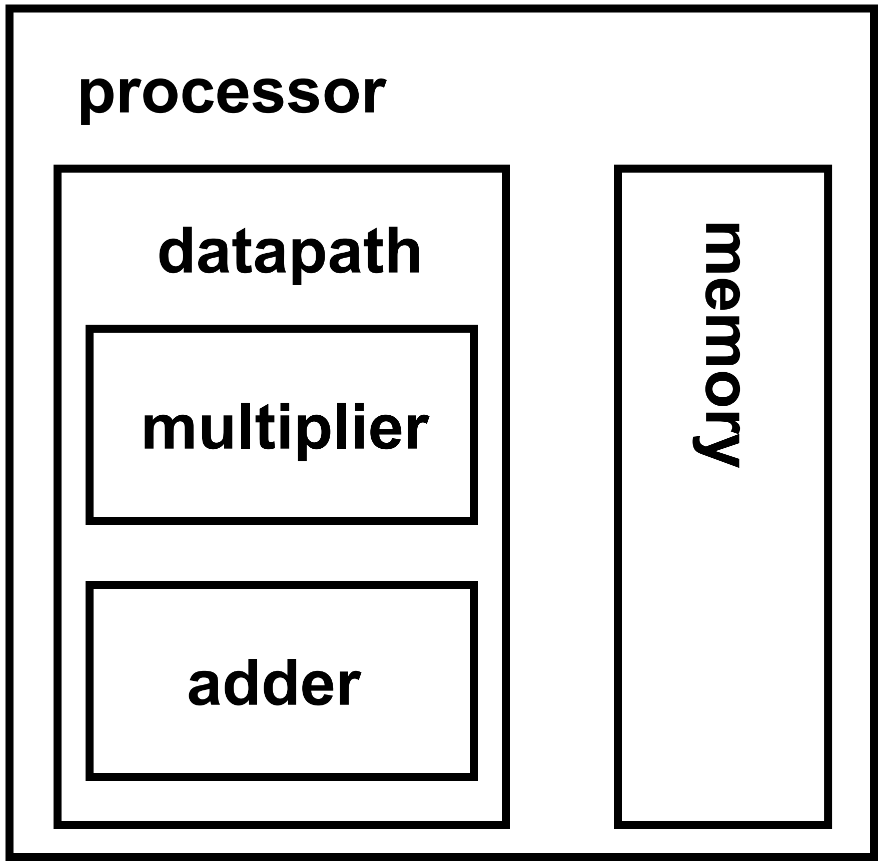
5 - combinational

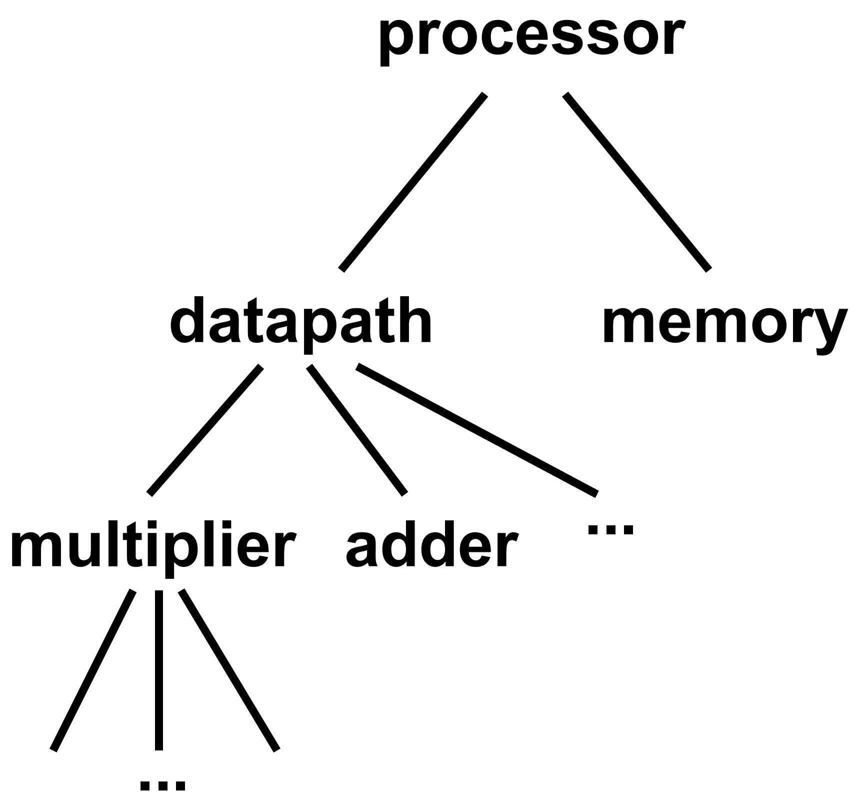
4 - inverter

2 - devices

3 - process

Hierarchy - Modularity



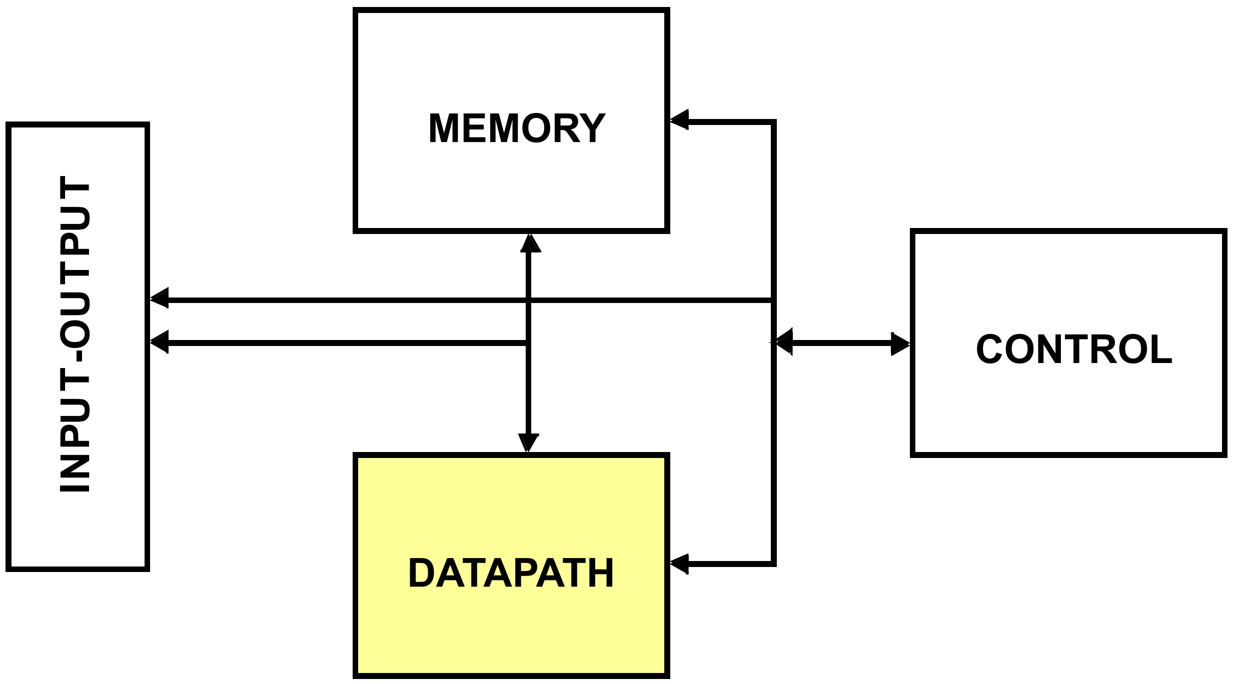


Modularity - Regularity

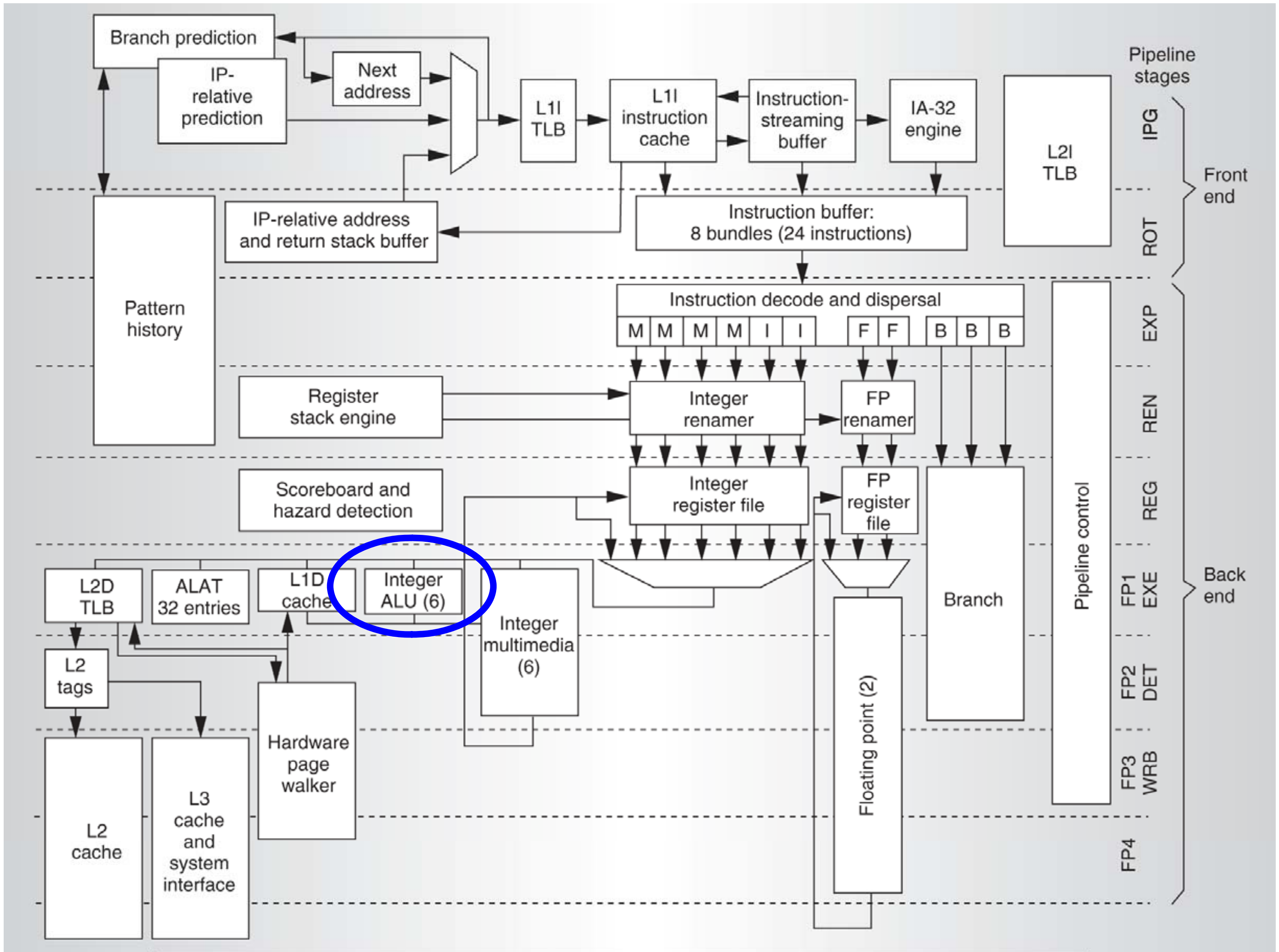
- Regularity at architecture level
- At logic level
- At transistor level
- At layout level

  - Bit-slices, abutment, array-structures,

A Generic Digital Processor



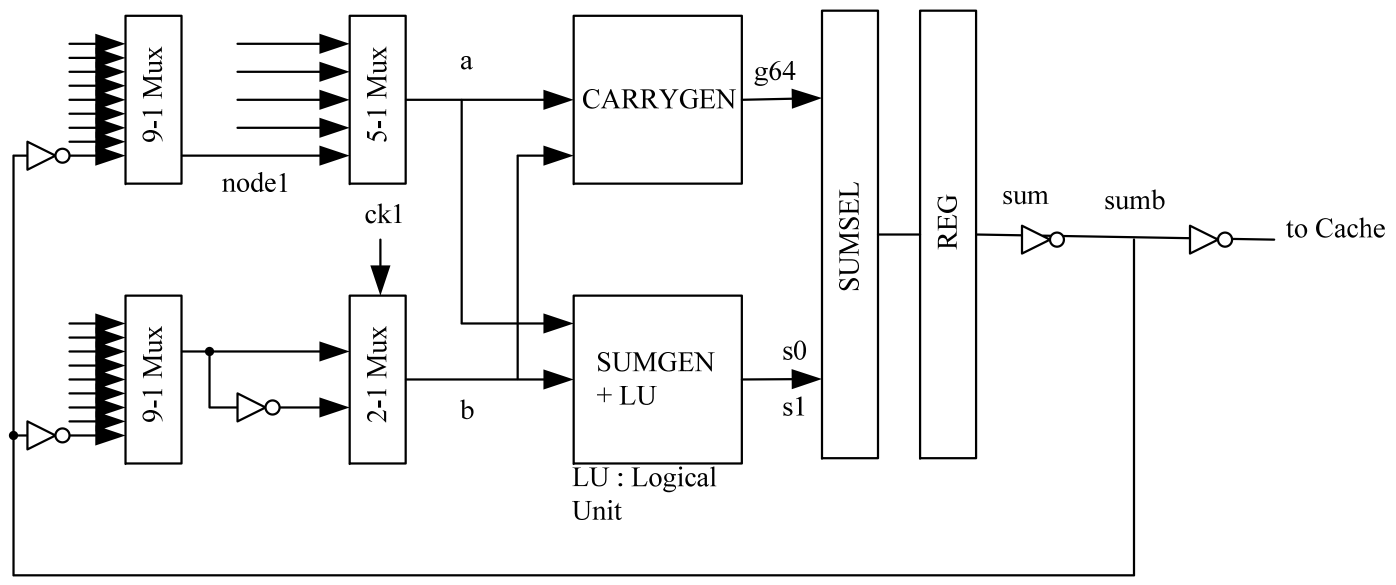
Itanium Datapath Pipeline



ALAT	Advanced-load address table	REG	Register file read
TLB	Translation look-aside buffer	EXE	ALU execution
IPG	Instruction pointer generation and fetch	DET	Exception detection
ROT	Instruction rotation	WRB	Write back
EXP	Instruction template decode, expand, and disperse	FP*x*	Floating-point pipe stage
REN	Rename (for register stack and rotating registers) and decode		

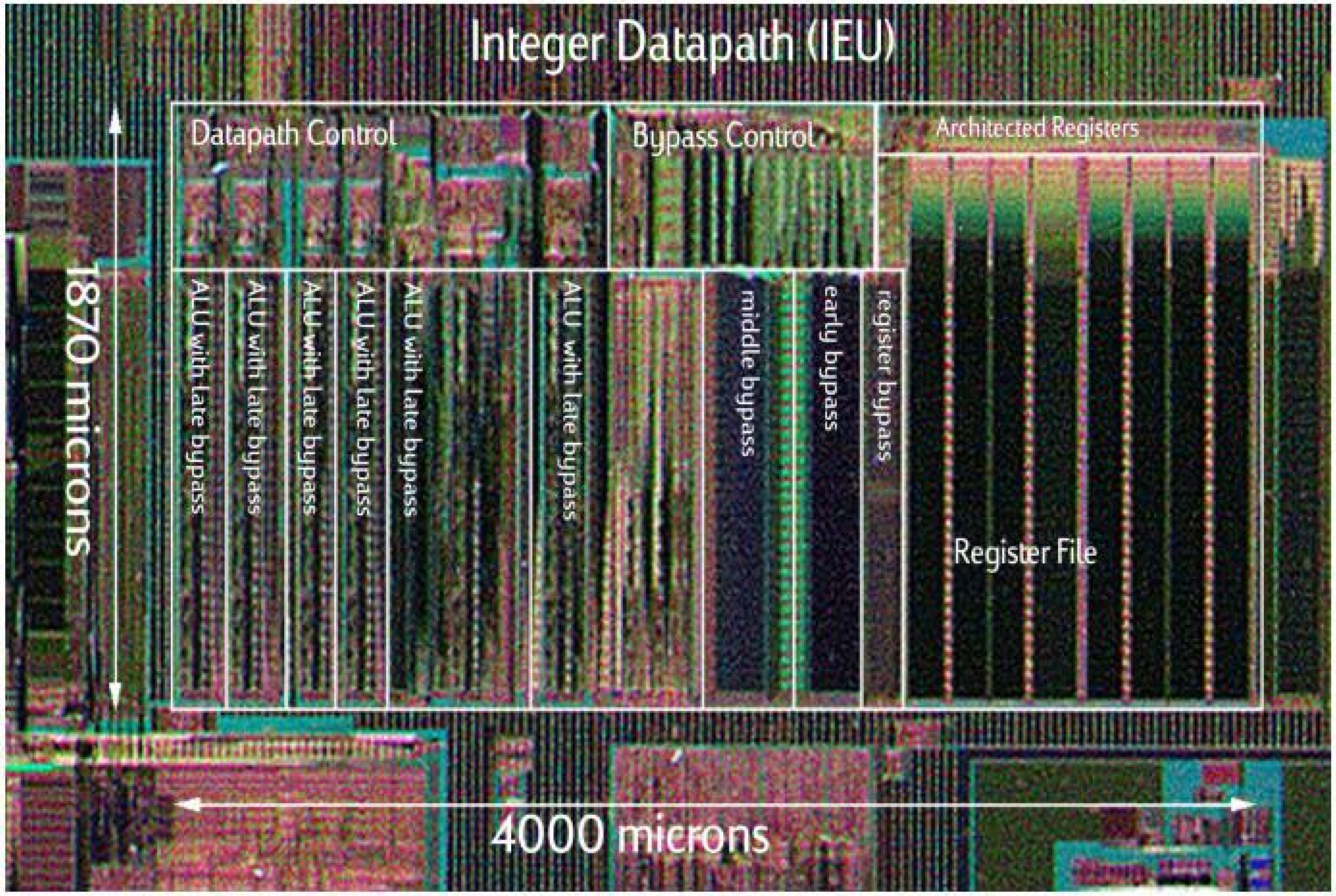
McNairy
IEEE Micro 2003

Itanium Integer Unit



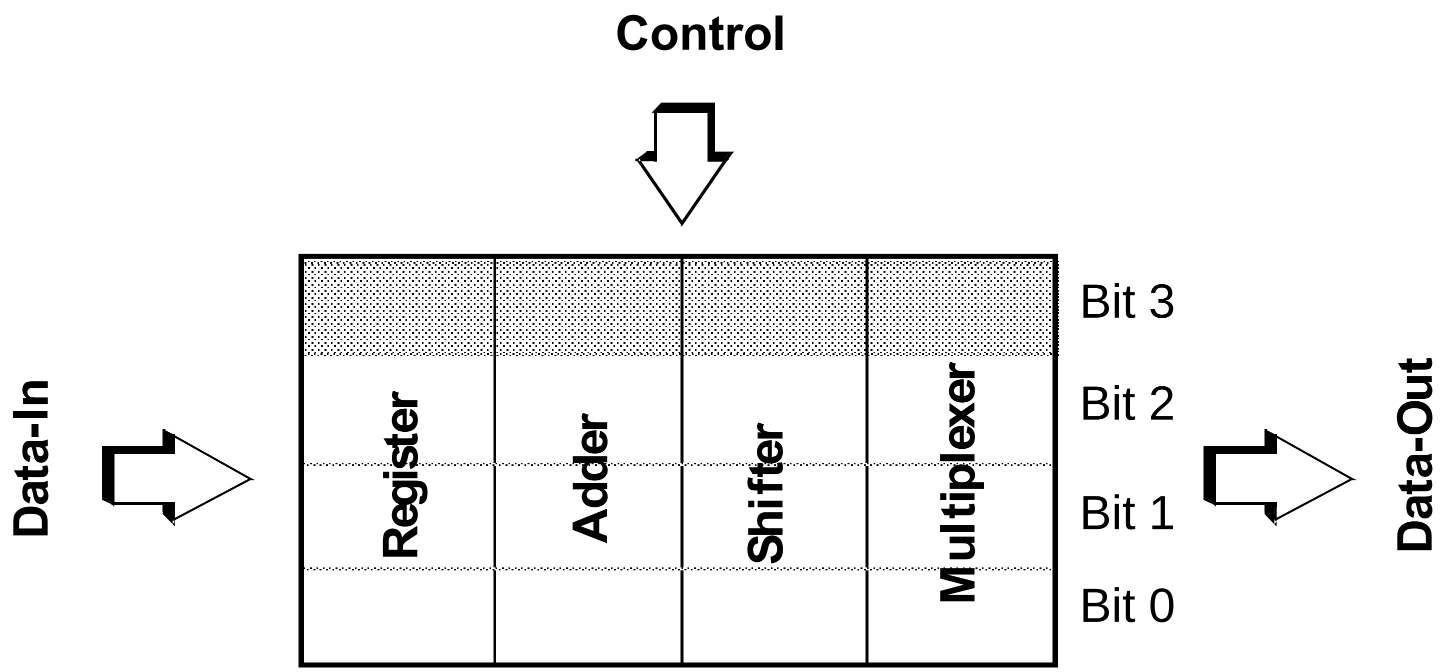
Itanium has 6 integer execution units like this

Itanium Integer Datapath



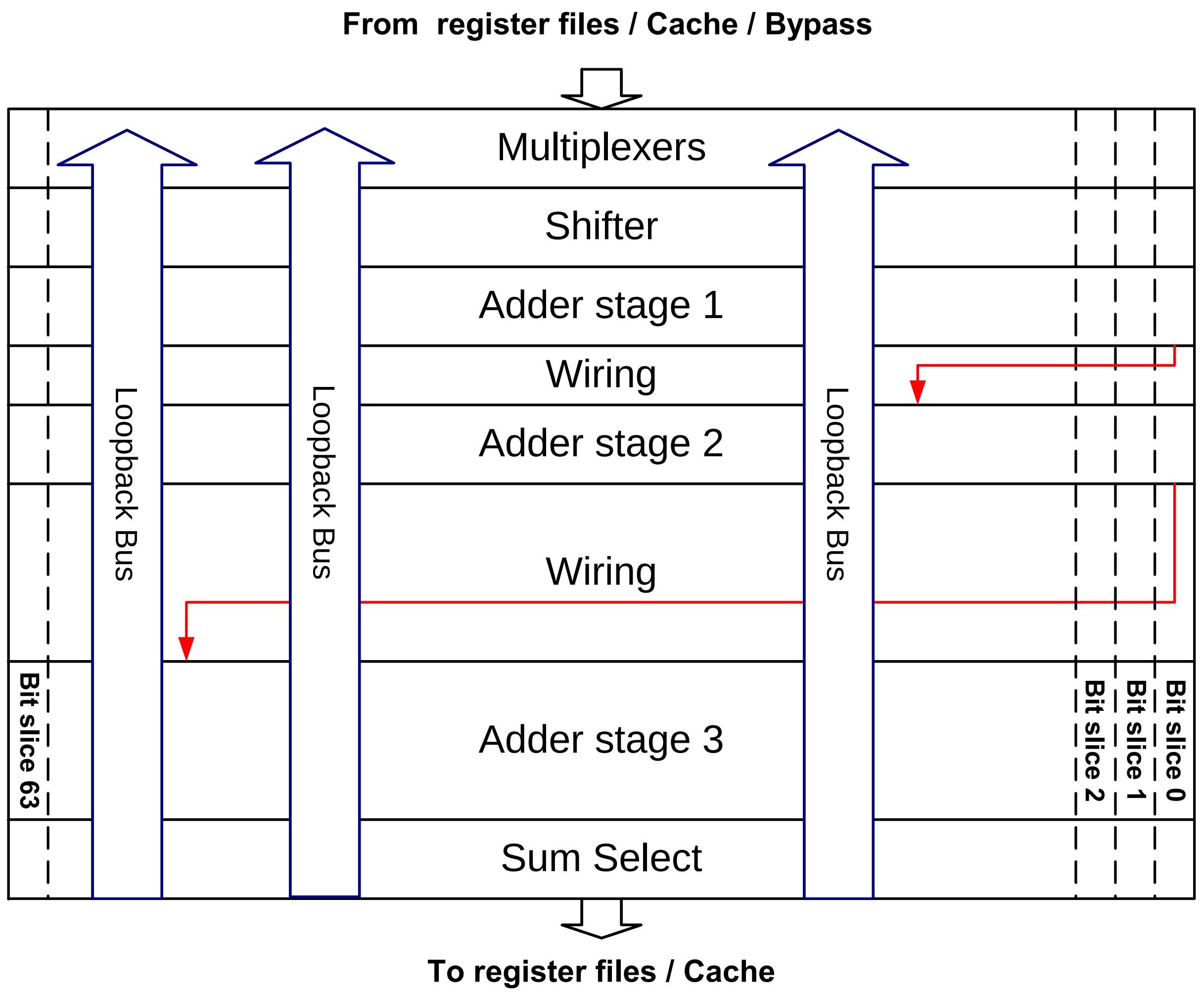
Fetzer, Orton, ISSCC'02

Bit-Sliced Design



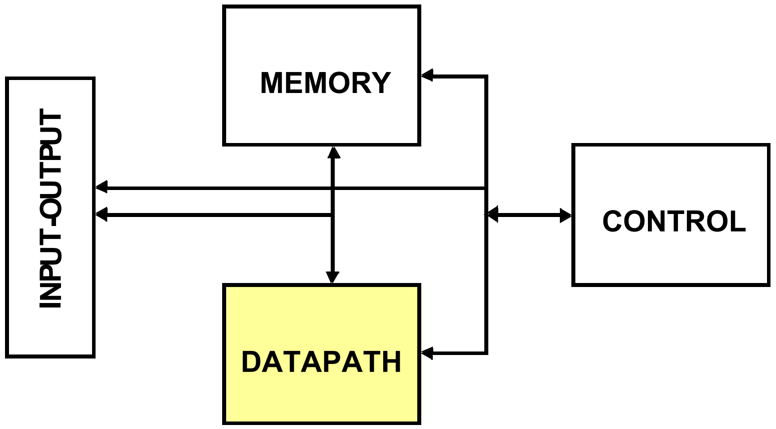
Tile identical processing elements

Bit-Sliced Datapath



Building Blocks for Digital Architectures

- **Arithmetic unit**
  - **Bit-sliced datapath (adder, multiplier, shifter, comparator, etc.)**
- **Memory**
  - **RAM, ROM, Buffers, Shift registers**
- **Control**
  - **Finite state machine (PLA, random logic.)**
  - **Counters**
- **Input-Output**
  - **Off-chip drivers, receivers**
- **Interconnect**
  - **Switches**
  - **Arbiters**
  - **Bus**



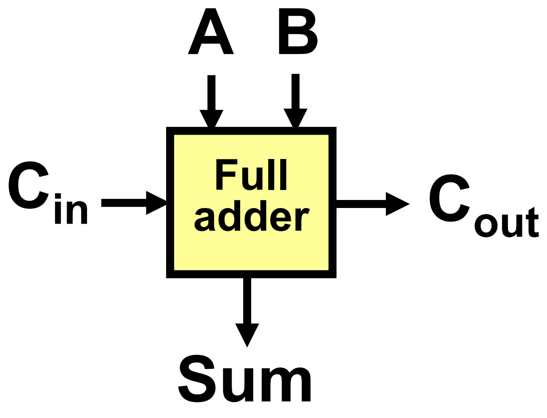
Adder Design

- Adders are fundamental building blocks
  - Digital filtering (DSP): MP3 en/decoder, GSM, GPS, …
  - Data processing
  - Multiplication
  - Address arithmetic
  - …
- Good performance is key
- Many architectures
  - Static adder
  - Dynamic adder (Manchester Carry Chain)
  - Pipelined Adder
  - Carry-Bypass, Carry Lookahead, Carry Select
  - …
- Design trade-offs, optimization
  - Architecture level
  - Logic level
  - Circuit level
  - Layout level

↑ Most effective

↓ Least effective

Full-Adder



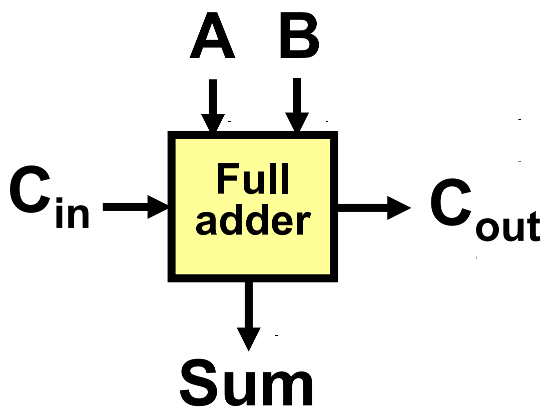
Add three one-bit numbers

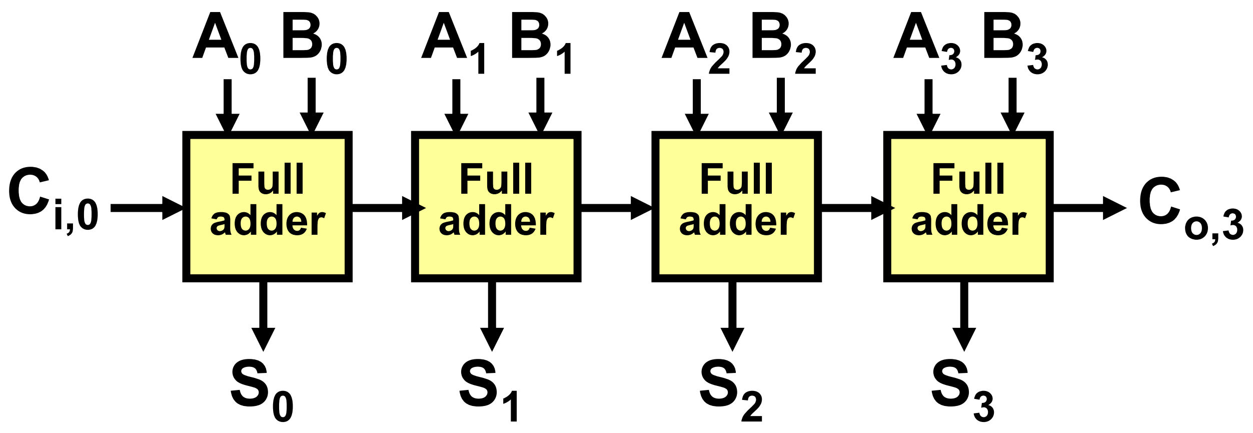
Equivalently: count # 1's in A, B, C_i

Output as 2-bit number <C_{out}S>

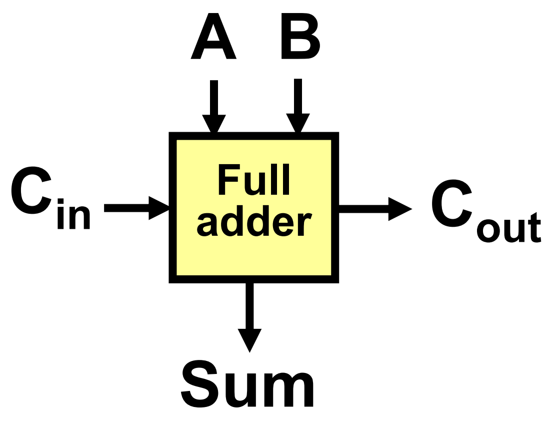
C_{in}	B	A	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

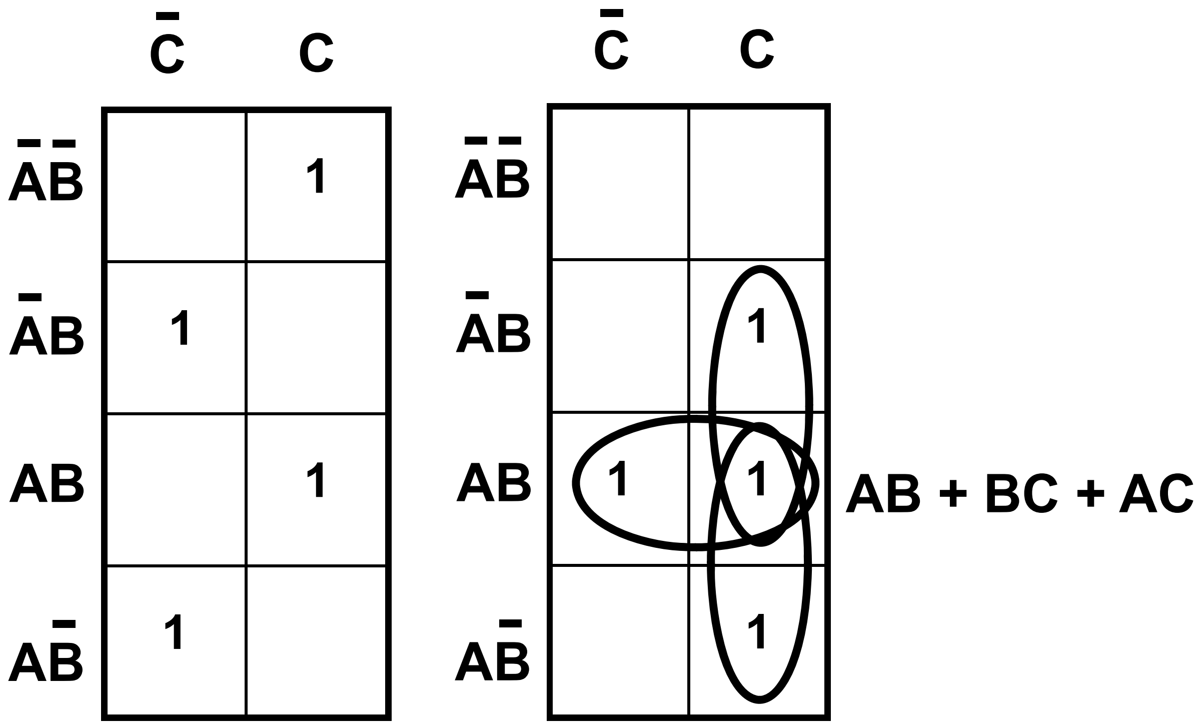
The Ripple-Carry Adder





The Binary Adder



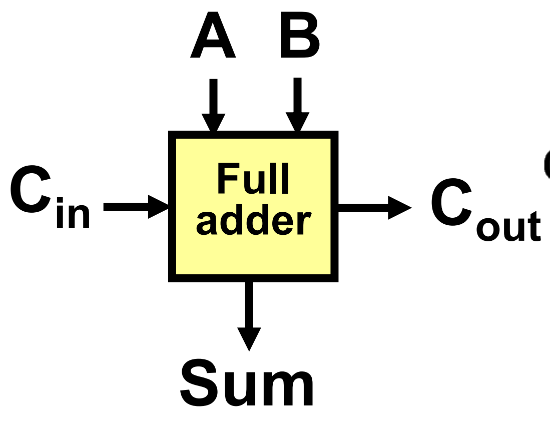


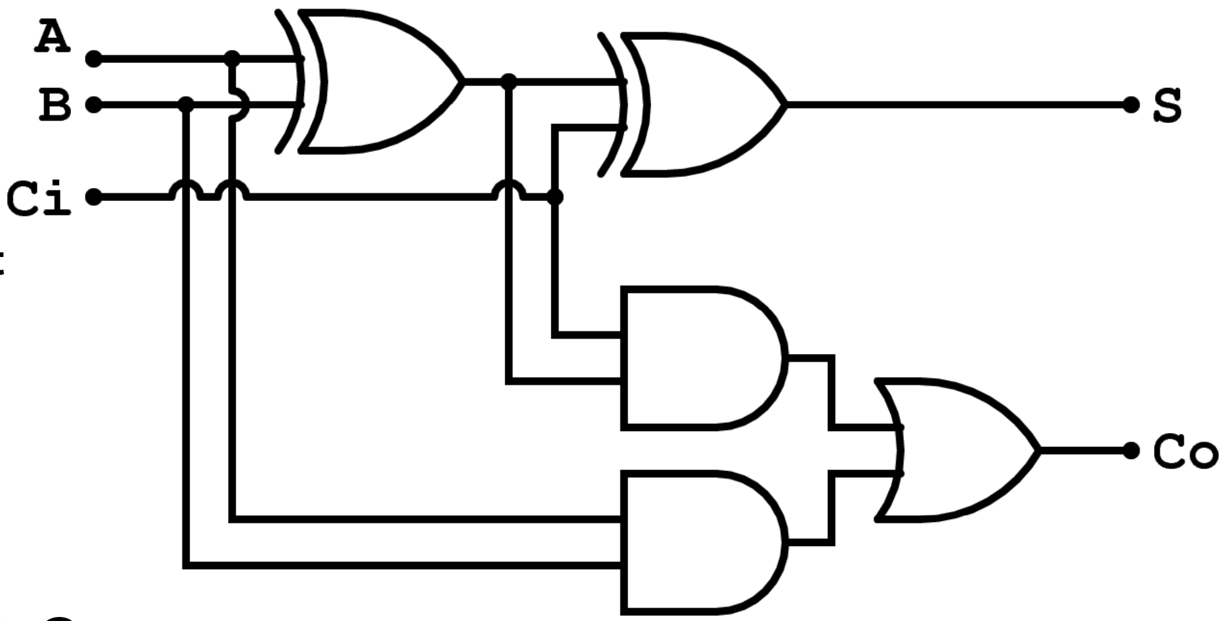
(a) SUM (b) CARRY

$$S = A \oplus B \oplus C_i$$
$$= \bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + ABC_i$$
$$C_0 = AB + BC_i + AC_i$$

AND-OR expressions for sum and carry

Full Adder Logic





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

$$C_0 = AB + BC_i + AC_i = AB + (A \oplus B)\, C_i$$

Why is this not so good in CMOS?

Naïve Complementary CMOS Implementation

- Use DeMorgan to convert AND-OR expressions for *SUM* and *CARRY* to NAND-NAND
- $PQ + RS = \overline{\overline{PQ}\ \overline{RS}}$ (example)

Q: What is advantage of NAND-NAND over NOR-NOR? Consider drive strength vs. area

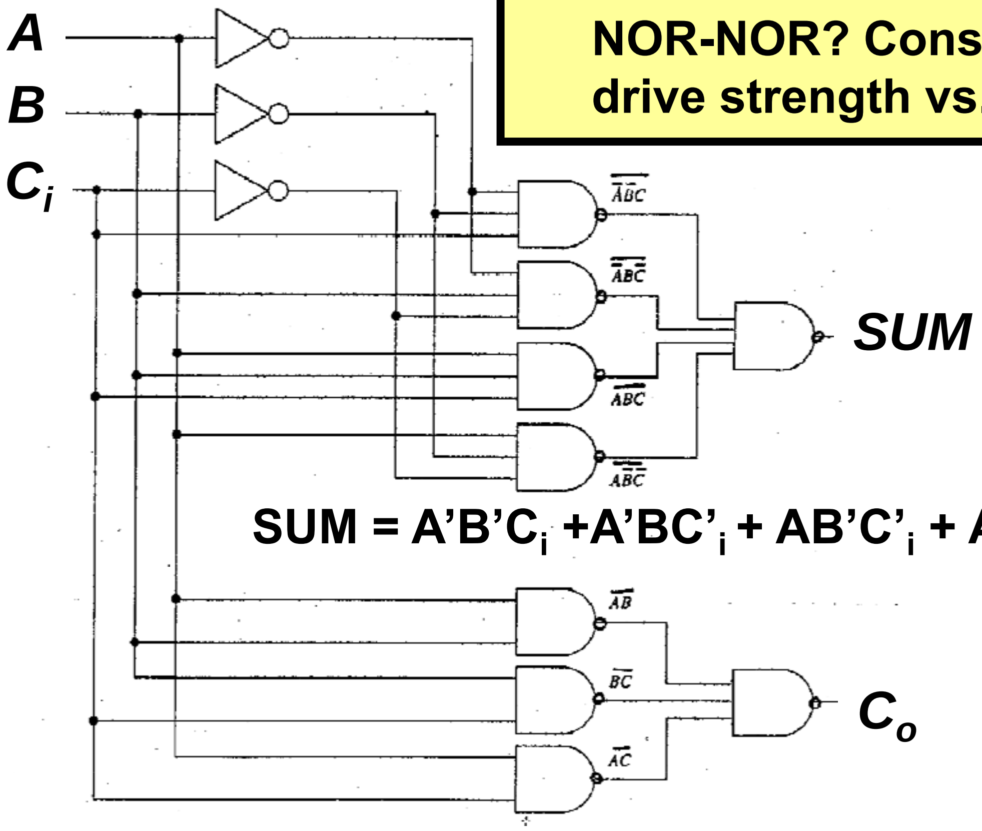
Transistor Count

3 × INVERT

3 × NAND-2

5 × NAND-3

1 × NAND-4

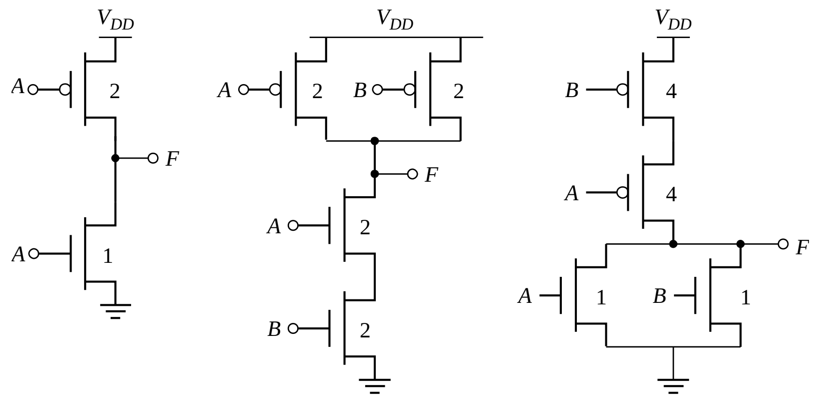


SUM = $A'B'C_i + A'BC'_i + AB'C'_i + ABC_i$

Can do better using more clever boolean factoring, but…

Nand/Nand vs Nor/Nor

Logical Effort



Inverter	2-input NAND	2-input NOR
p=1, g=1	**p=2, g=4/3**	**p=2, g=5/3**

p **ratio of intrinsic delay compared to inverter**

g **logical effort – ratio of inp. cap for same strength**

p, g independent of sizing, only topology of gate

TUD/EE ET4293 digic - 1011 - © NvdM - 04 Combinational *3/22/2011* *34*

NAND Logical Effort: (n+2)/3

NOR Logical Effort: (2n+1)/3

Full-Adder Boolean Factoring

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

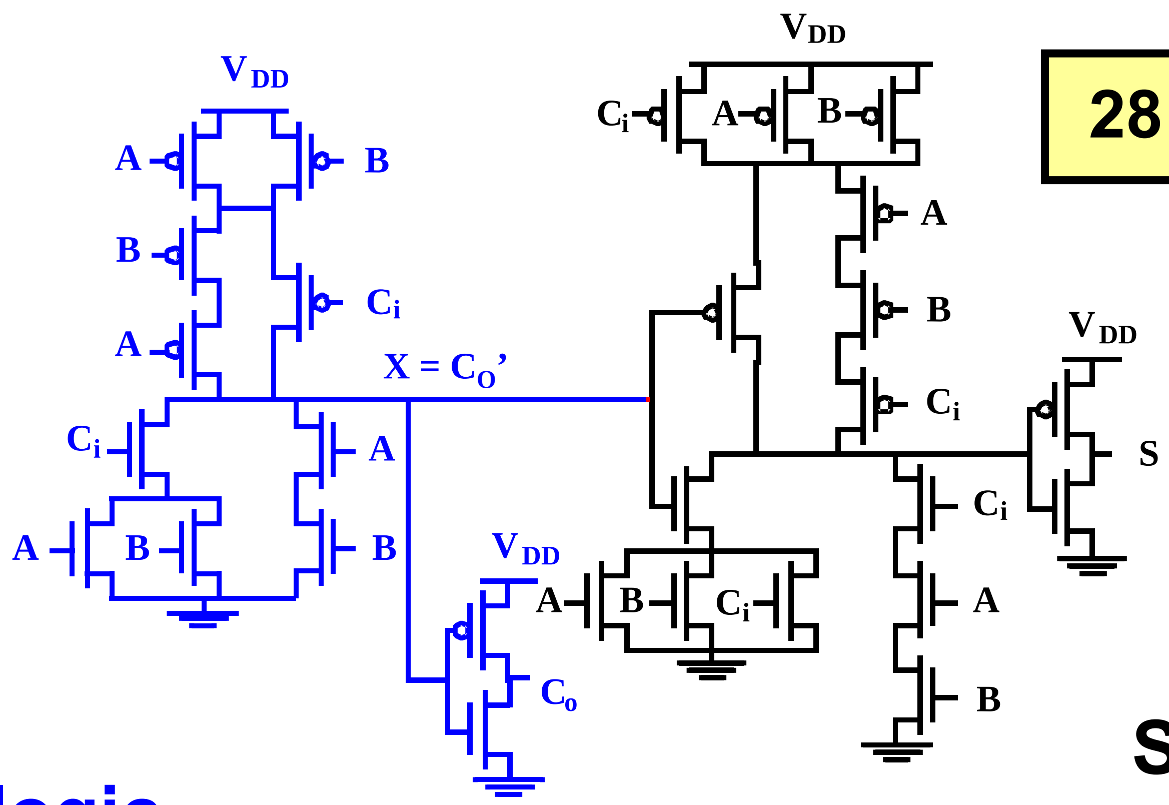
$$= ABC_i + \bar{C}_0(A + B + C_i)$$

$$C_0 = AB + BC_i + AC_i$$

$$= AB + (A + B)C_i$$

C_{in}	B	A	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

Improved Complementary Static Full Adder



28 Transistors

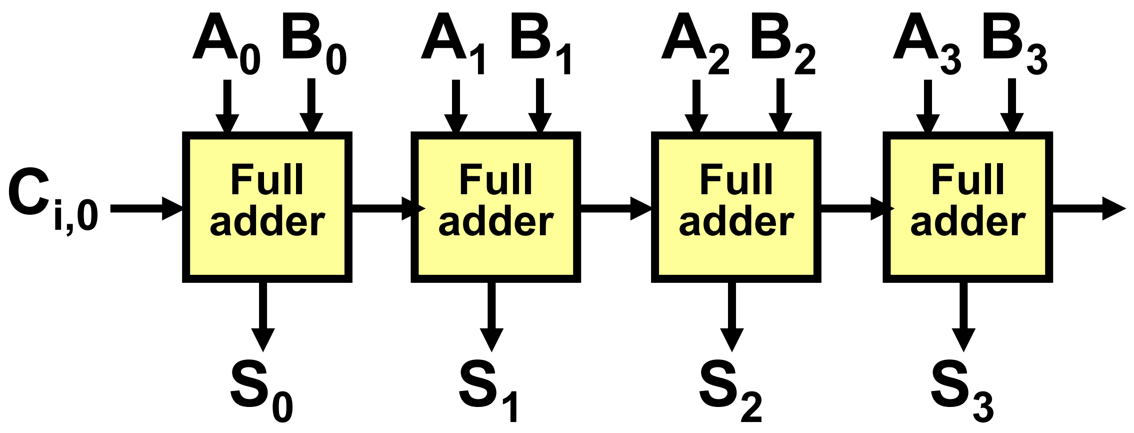
Sum logic

Carry logic

$C_0 = AB + (A + B)C_i$

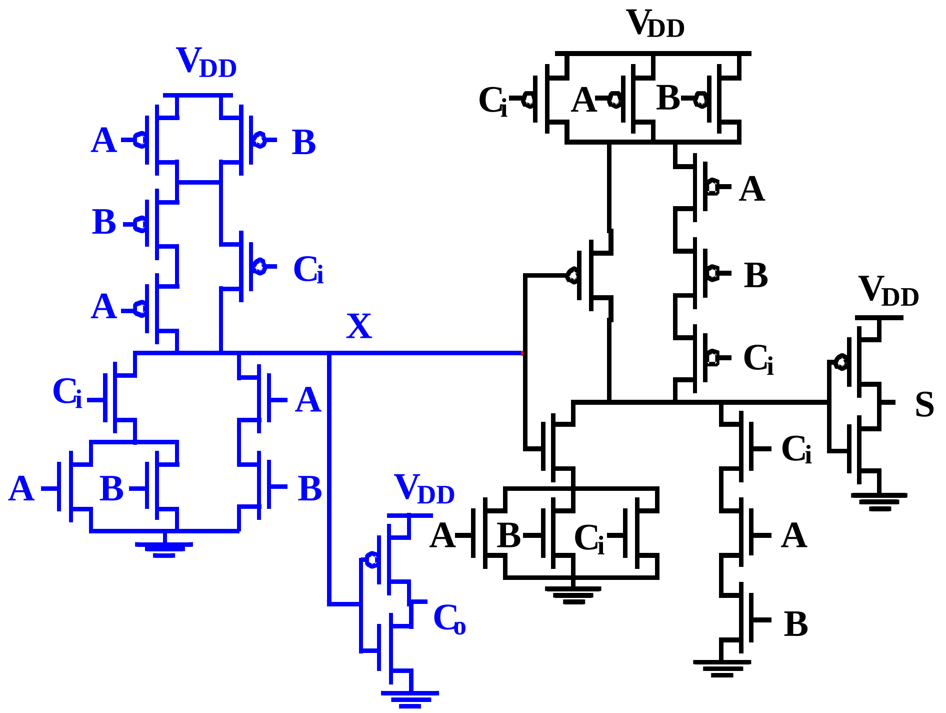
$S = ABC_i + \overline{C}_0(A + B + C_i)$

Ripple-Carry Adder Delay



- **Worst case delay through full carry path (ripple carry)**
- **Linear with the number of bits (N)**
- **$T_{adder} = (N\text{-}1)\ T_{carry} + \text{Max}\ (T_{carry}, T_{sum})$**
- **$T_{adder} = O(N)$ "T_{adder} is of Order N"** ***means linear with N***
- **Goal: Make the fastest possible carry path circuit**

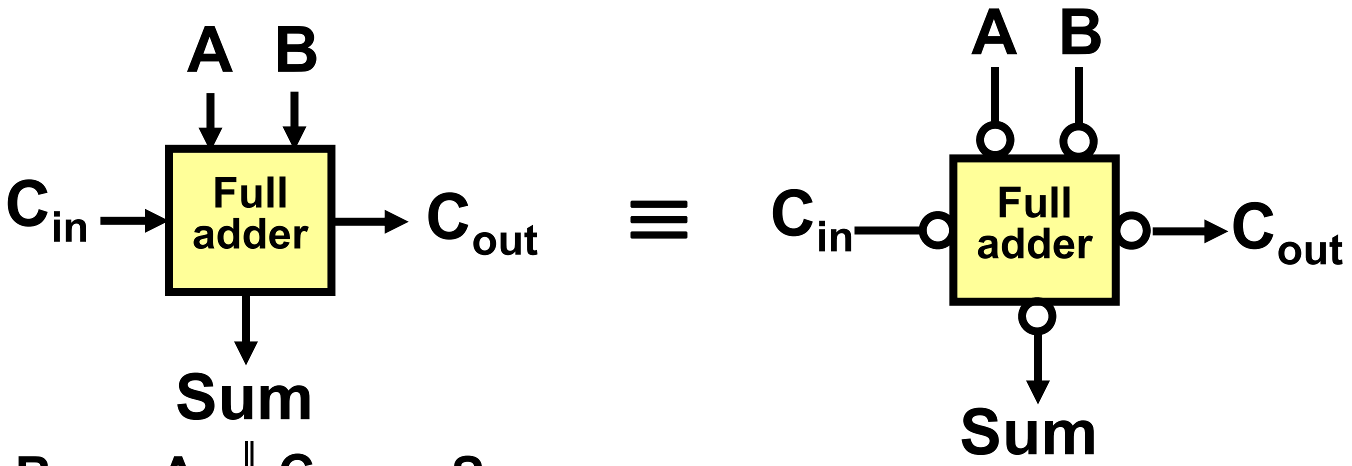
Adder Evaluation



Carry Chain:

- Long PMOS chains
- High C at X
- 2 (inverting) stages

Inversion Property



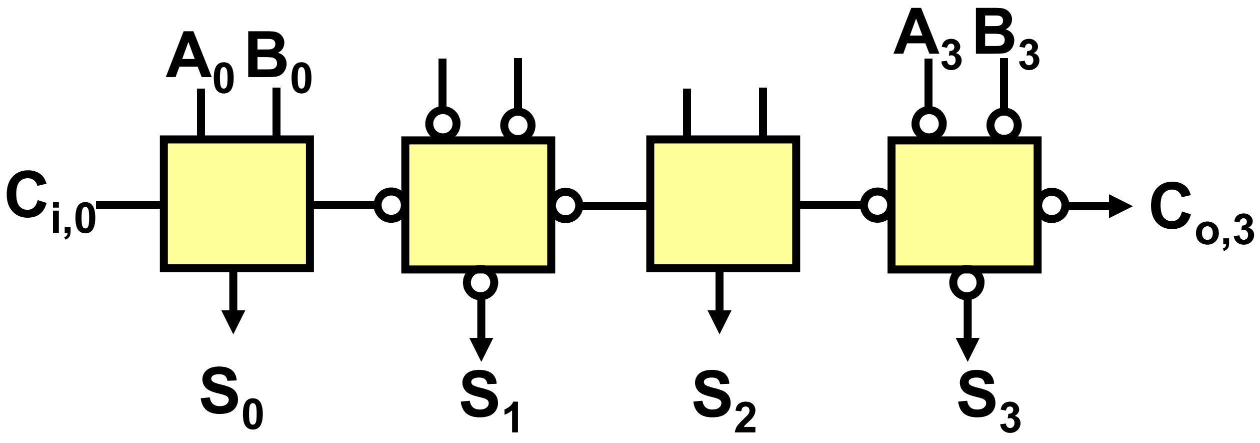
C_{in}	B	A	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

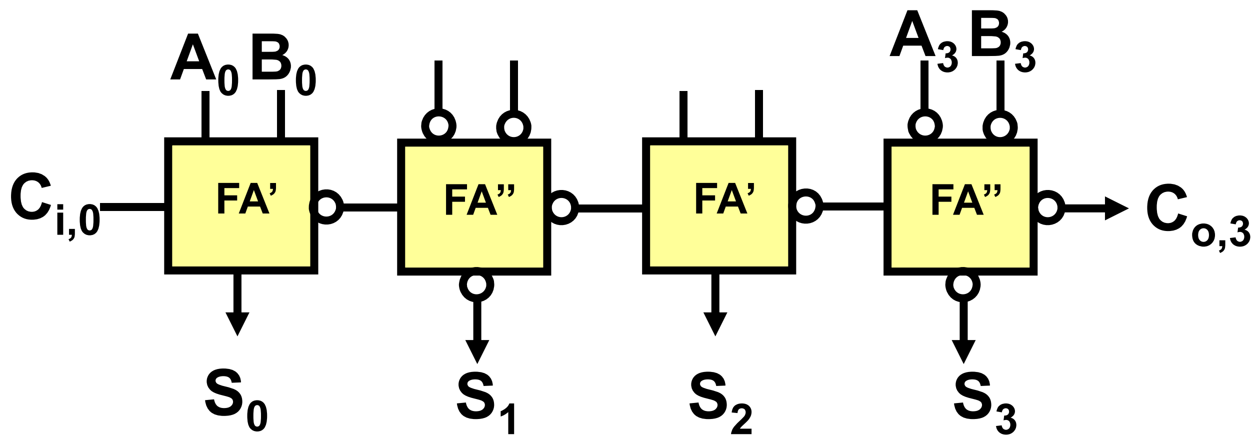
Mirror Symmetry

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

$$\overline{C_0}(A,B,C_i) = C_0(\overline{A},\overline{B},\overline{C_i})$$

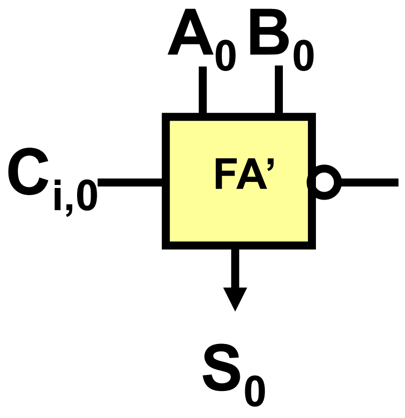
Minimize Critical Path by Reducing Inverting Stages

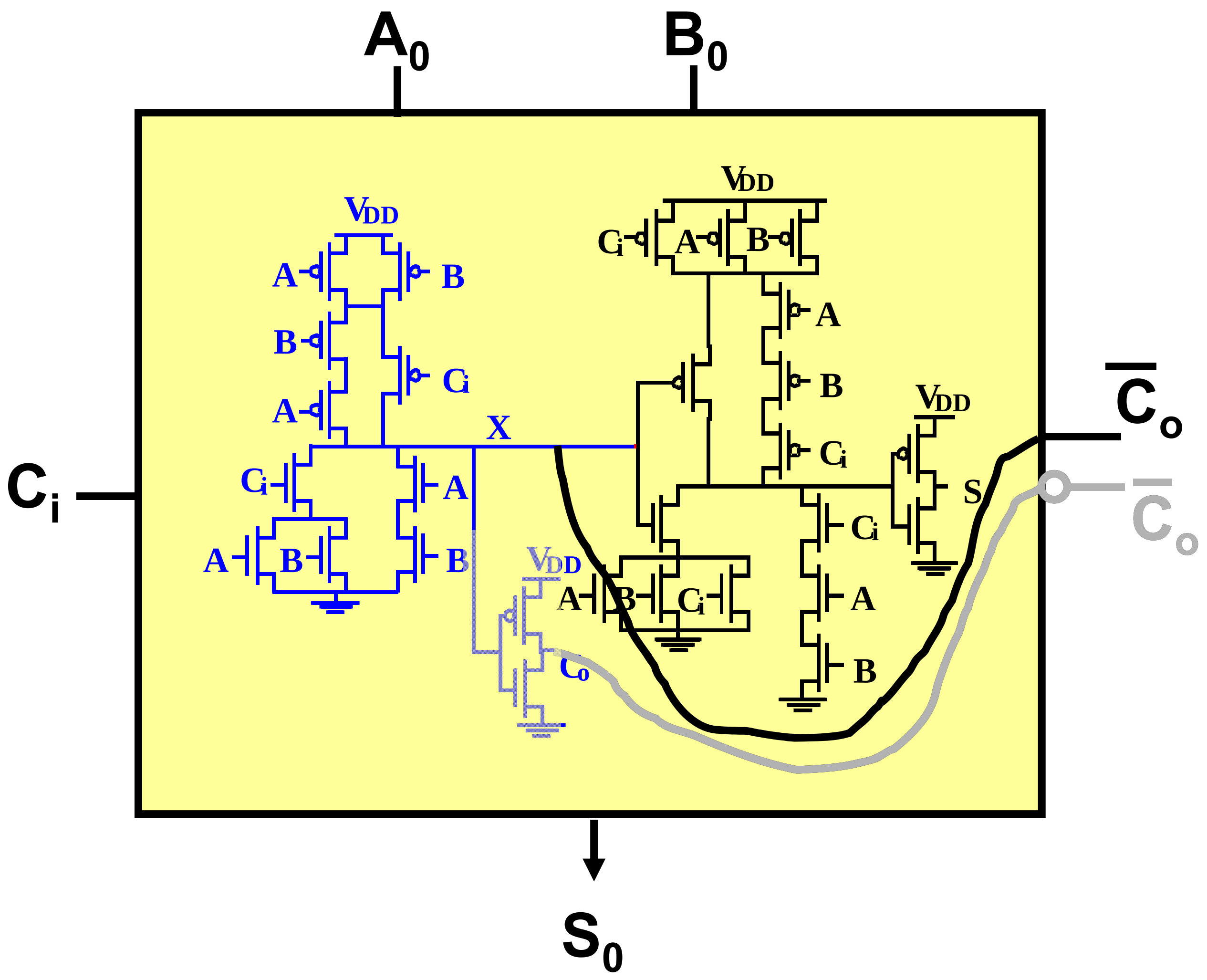




- Can eliminate inverter in carry from each FA
- Need 2 different types of cells, but both with inverting carry – will require only one stage per bit

Eliminate Inverter In Carry.





Further Boolean Factoring

- **Goal: fastest carry computation possible**
- **Pre-compute intermediate variables based on A, B, and when C_i arrives, C_o is almost ready**
- **The intermediate variables are called**
  P (Propagate)
  G (Generate)
  K (Kill) aka D (Delete)
- **Only one or two of these are needed**

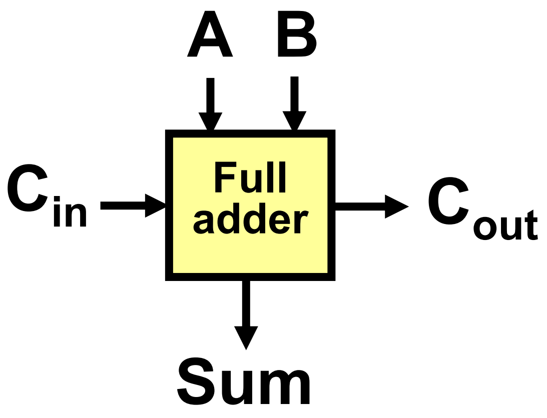
Carry Status Intermediate Variables

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

$\bar{A}\bar{B}$

$A \oplus B$

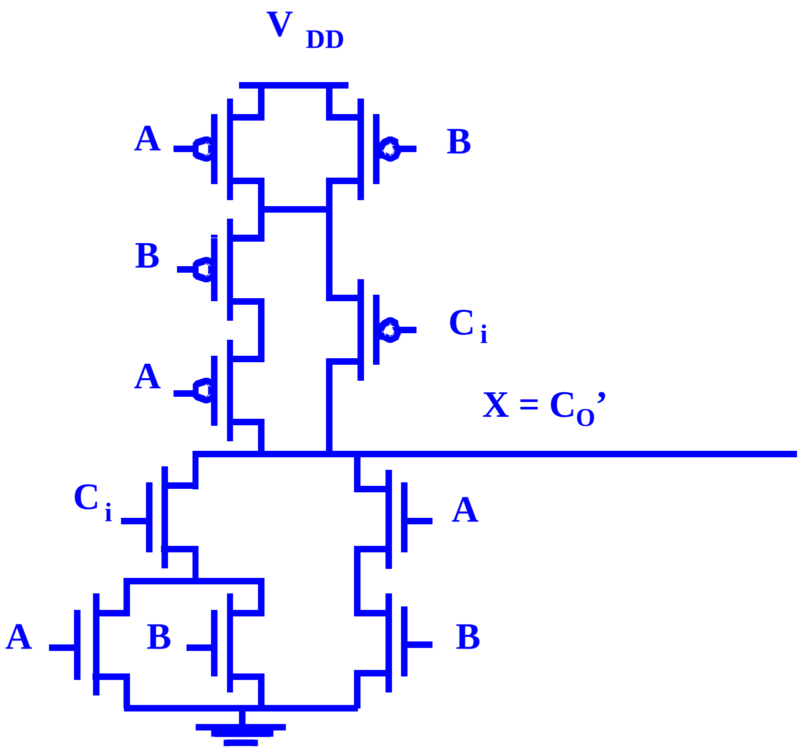
AB

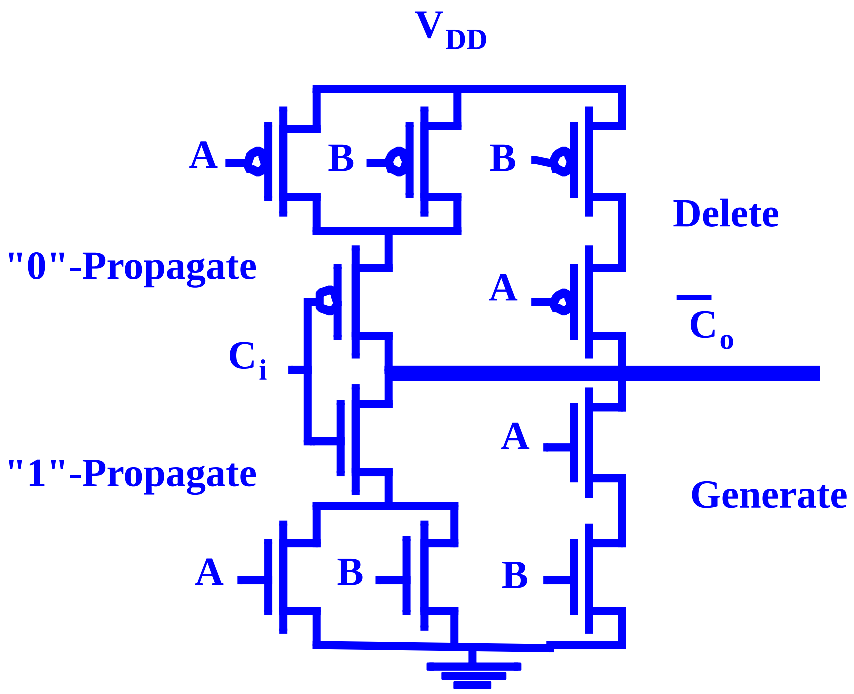


- *P, G, D* **ONLY** **depend on** ***A, B***
- **Simple and fast expressions for *S* and C_o based on *G, P* and C_i**
- **Or D instead of G**

$$C_0\,(G, P) = G + PC_i$$
$$S\,(G, P) = P \oplus C_i$$

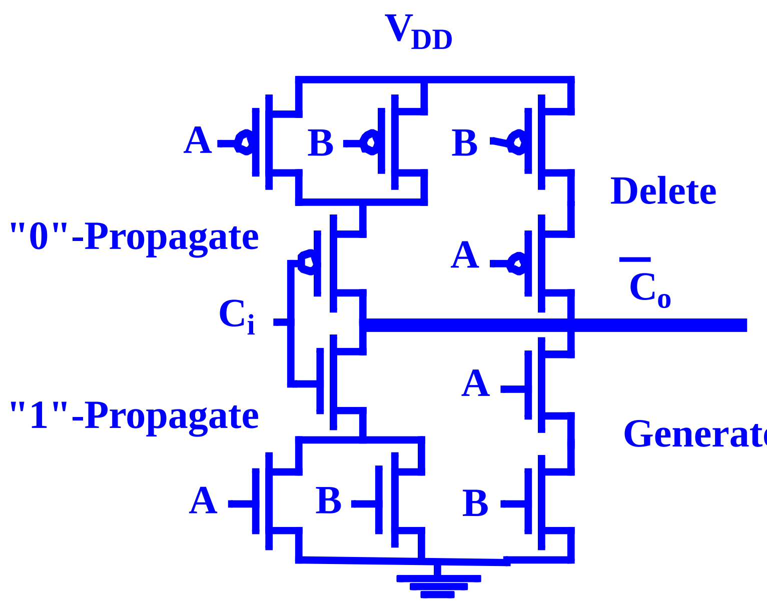
Carry – Dual PU/PD vs Mirror Structure





Mirror Adder - Carry Logic

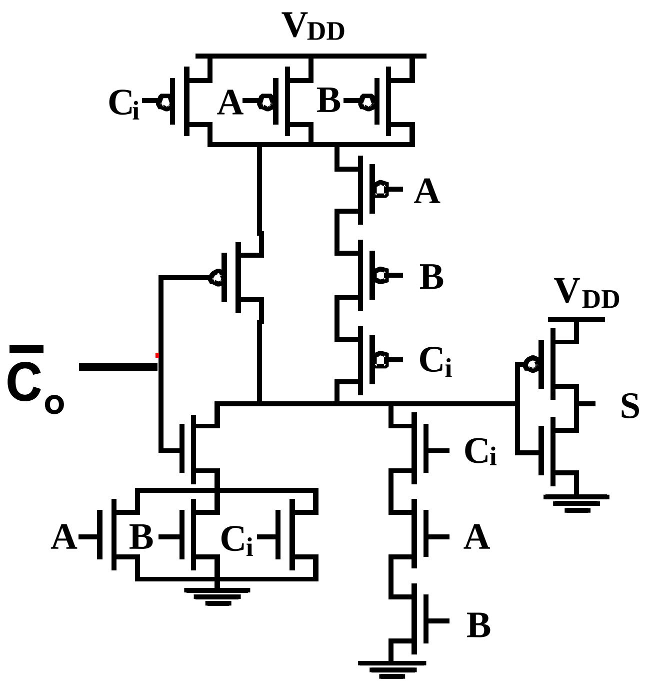
A	*B*	C_i	*S*	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate



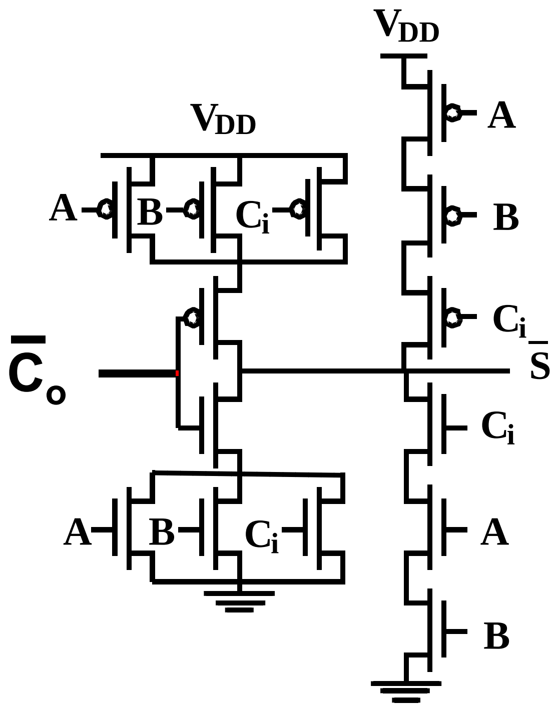
$$C_0\,(G, P) = G + PC_i$$

$$S\,(G, P) = P \oplus C_i$$

Mirror Adder - Symmetrical Sum Logic



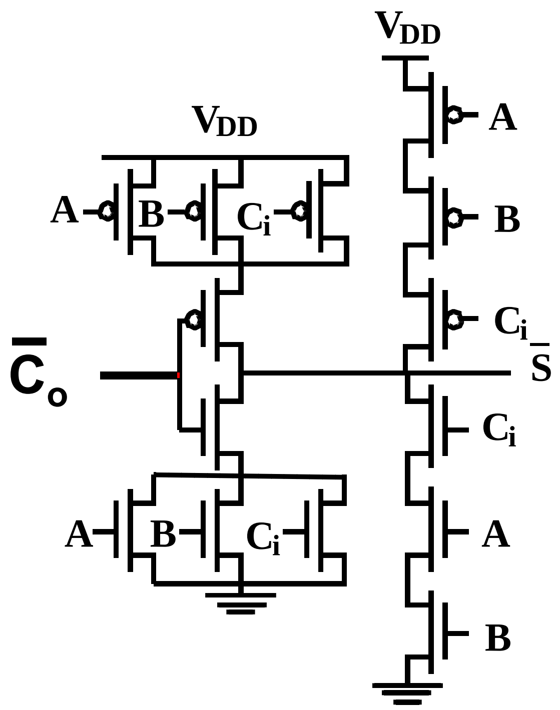
Dual Structure



Symmetrical Structure

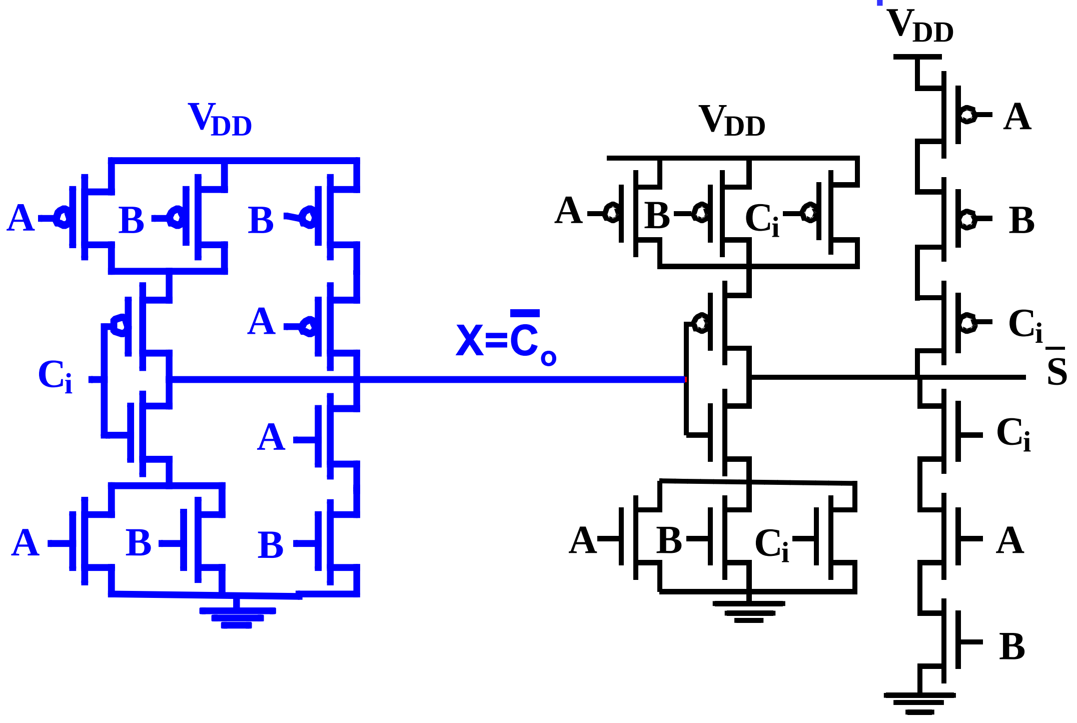
Mirror Adder - Symmetrical Sum Logic

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



- No need to restrict topologies to strictly dual
- But must always obey **Mutual Exclusion Principle** (and in static CMOS, output must always be driven)

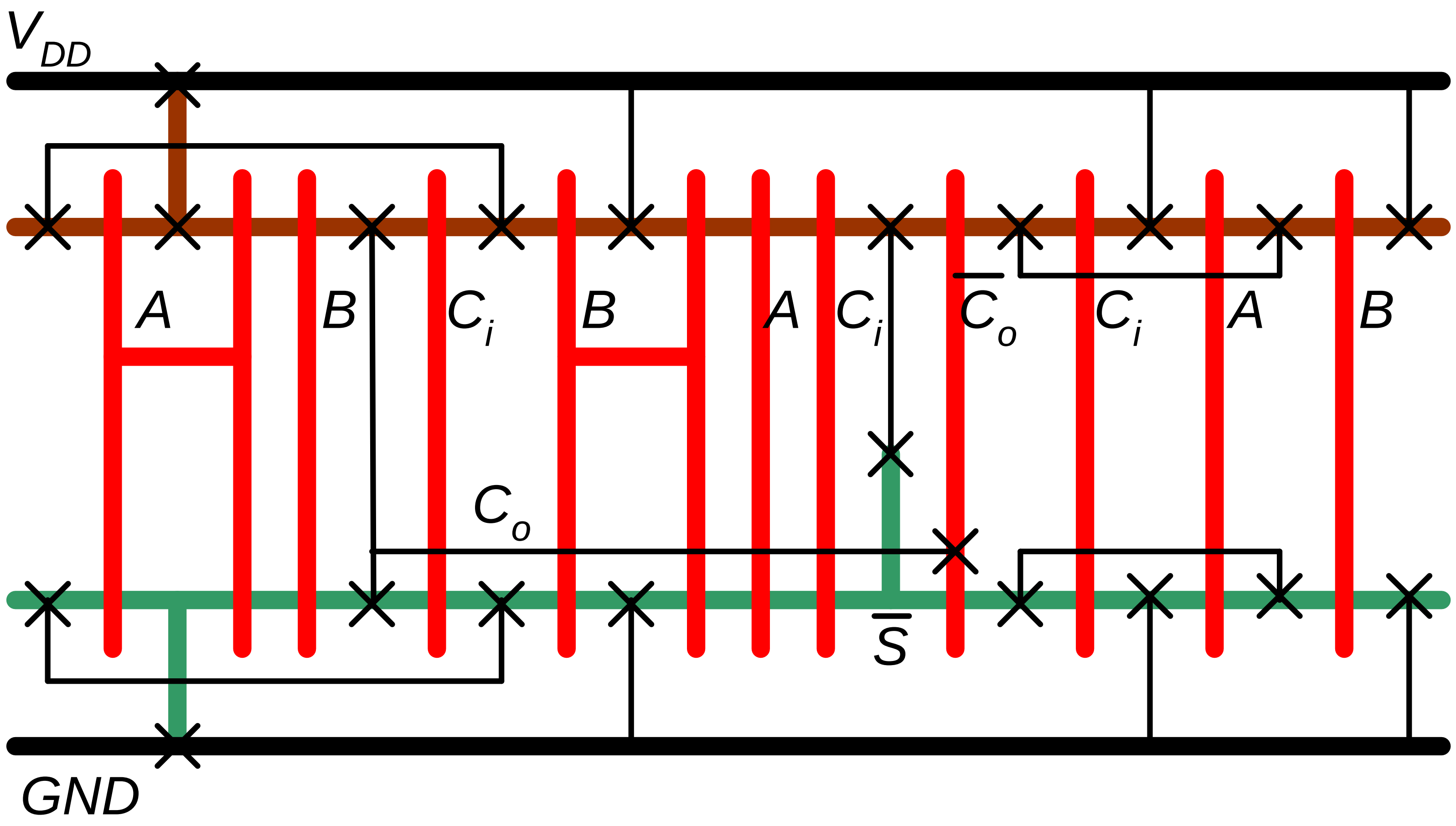
Mirror Adder – Complete.



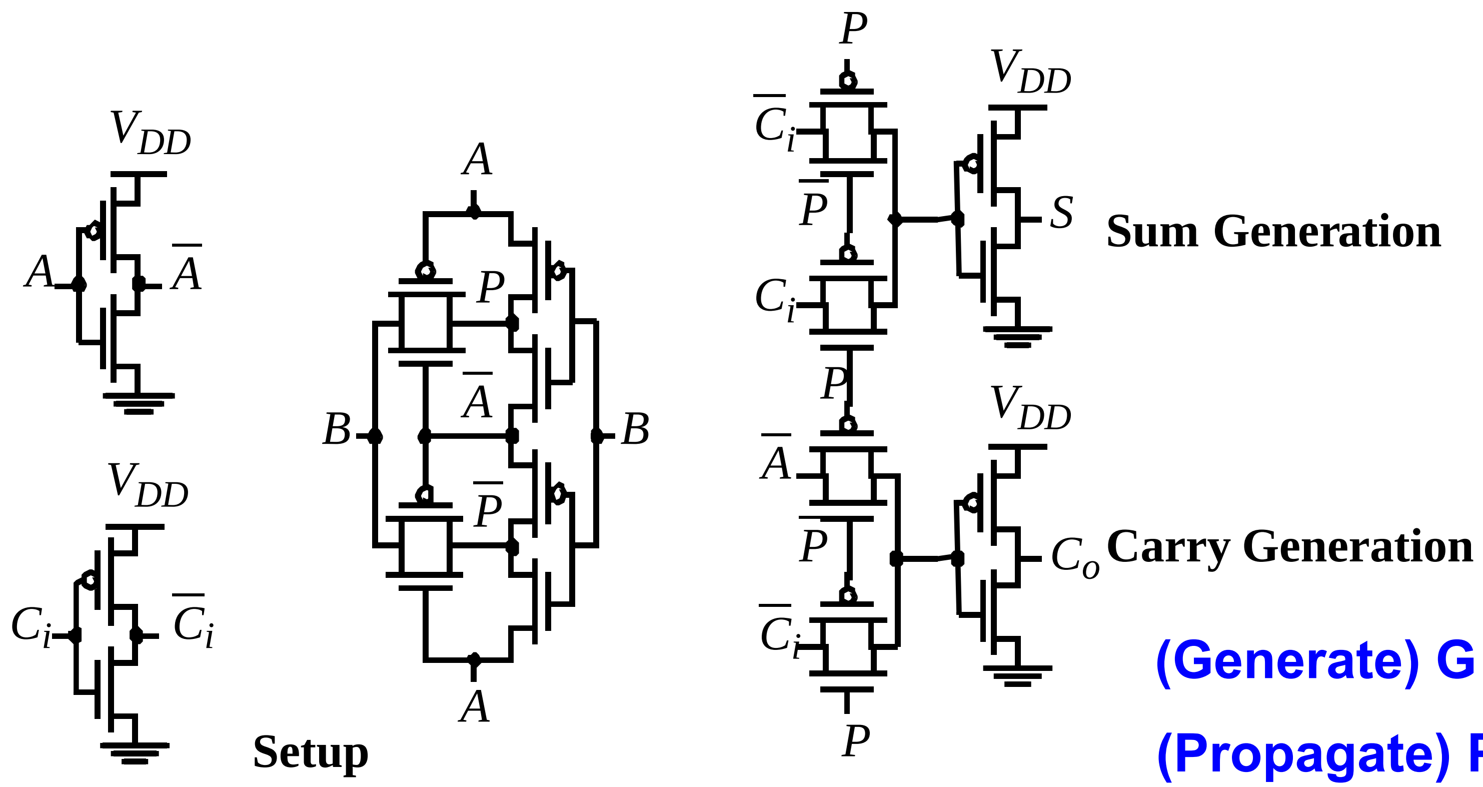
- Symmetrical NMOS/PMOS chains ==> symmetrical timing
- Max 2 PMOS in series in carry chain
- High capacitance at X, but only one stage in carry
  … and can optimize layout to reduce (diffusion) cap at X
- Transistors connected to C_i closest to output
- Only transistors in carry chain need size optimization for speed
  rest can be minimum size

Mirror Adder

Stick Diagram



Transmission Gate Full Adder



(Generate) $G = AB$

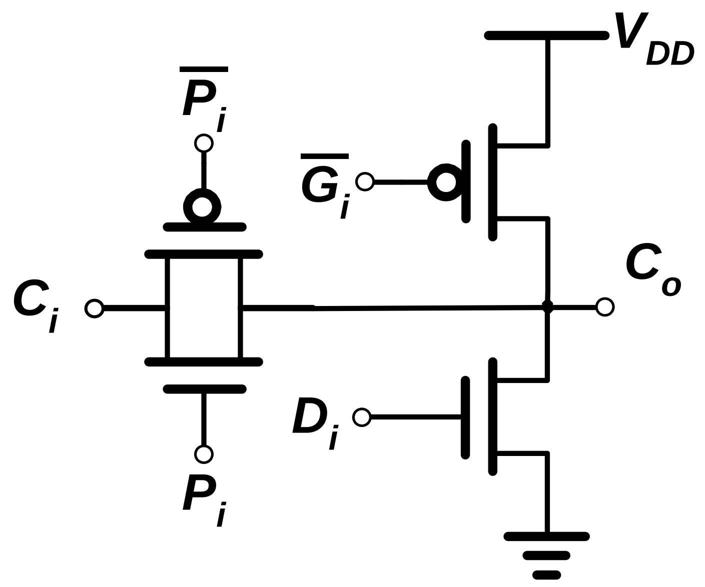
(Propagate) $P = A \oplus B$

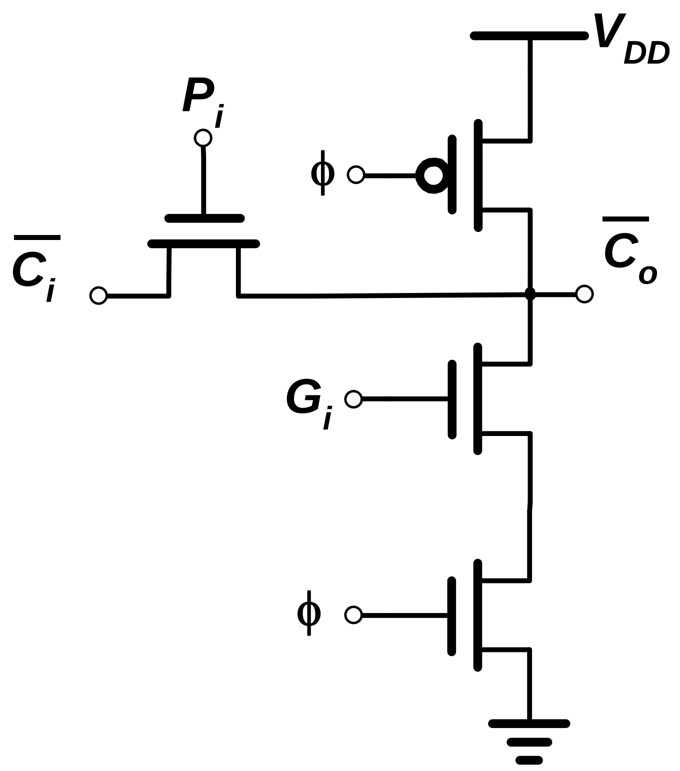
(Delete) $D = \overline{A}\,\overline{B}$

(Propagate) $P = A \oplus B$

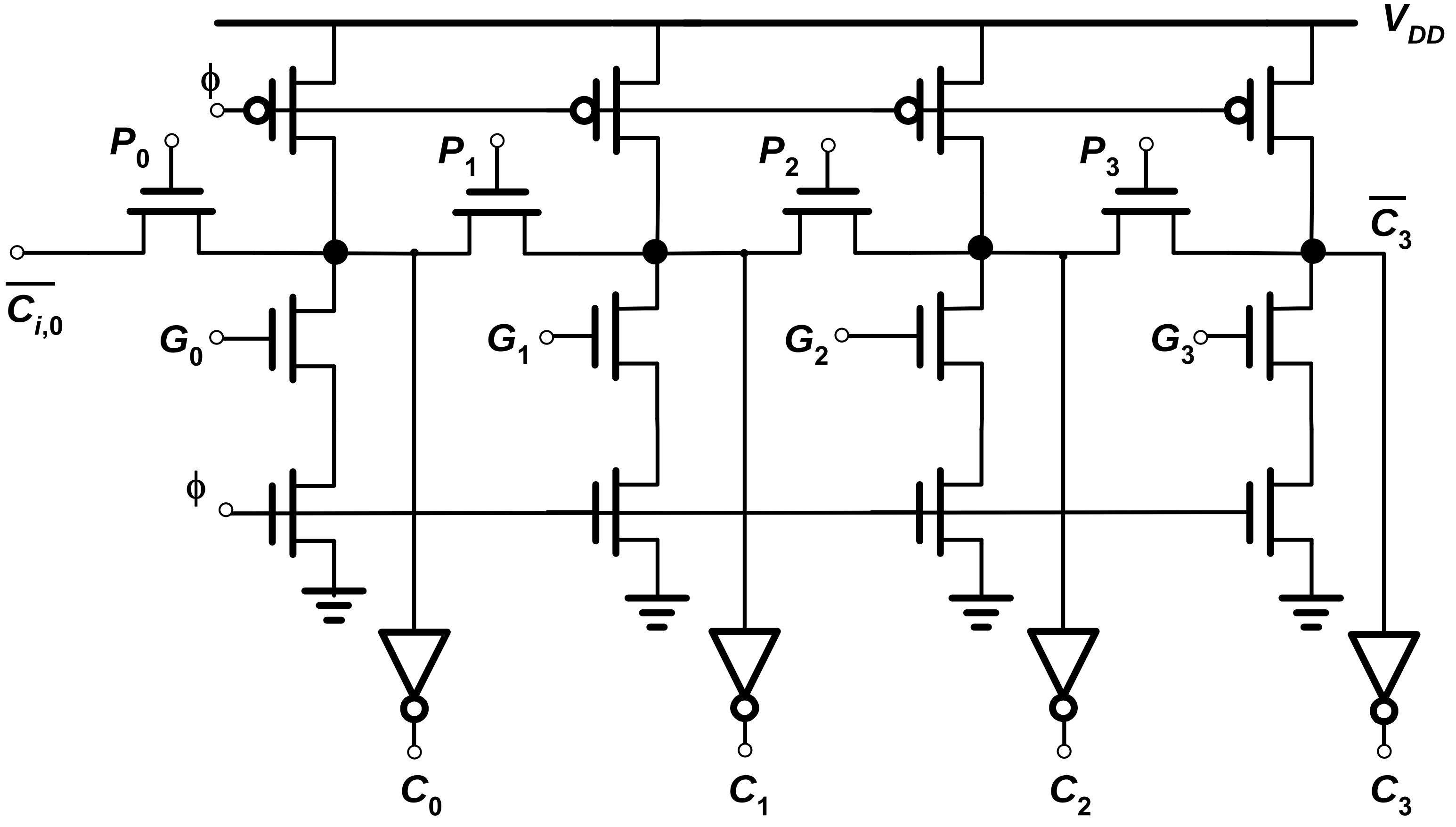
$$\boxed{\begin{aligned} C_0(G, P) &= G + PC_i \\ S(G, P) &= P \oplus C_i \end{aligned}}$$

Manchester Carry Chain

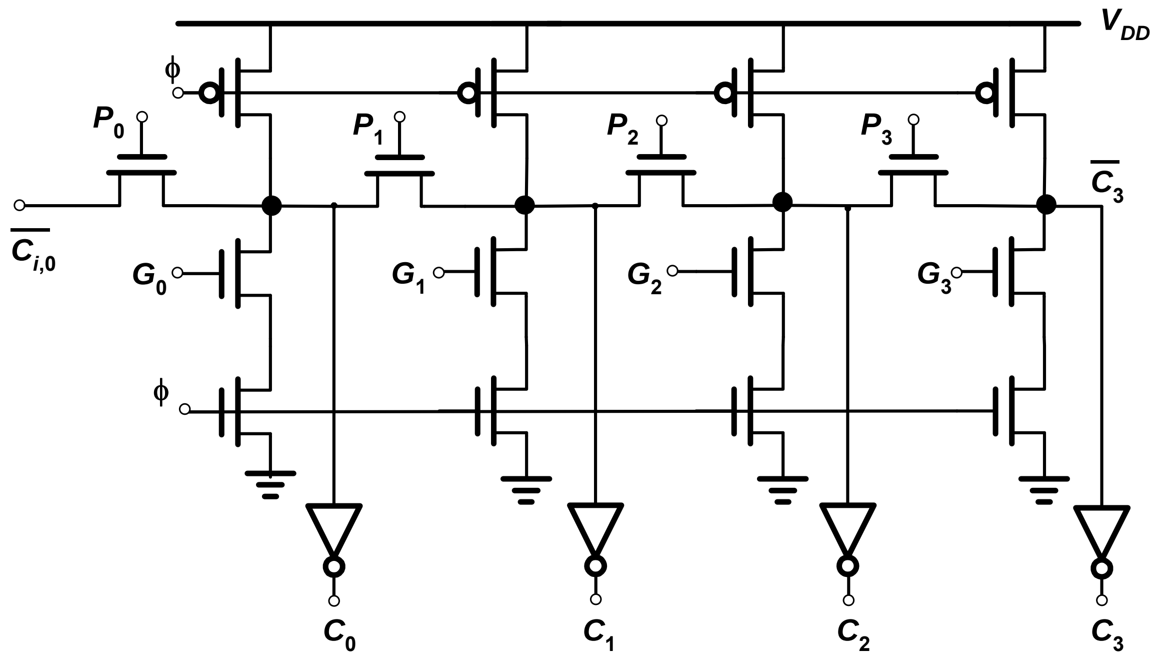




Manchester Carry Chain



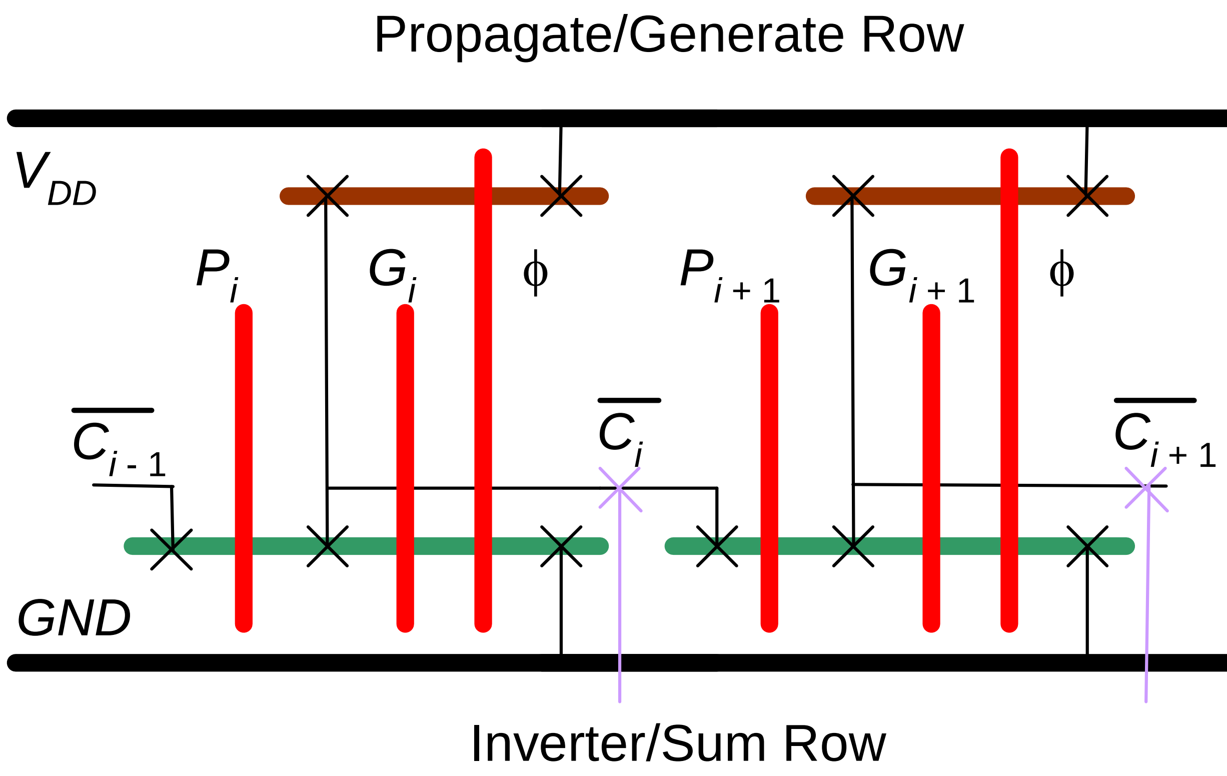
Manchester Carry Chain Delay



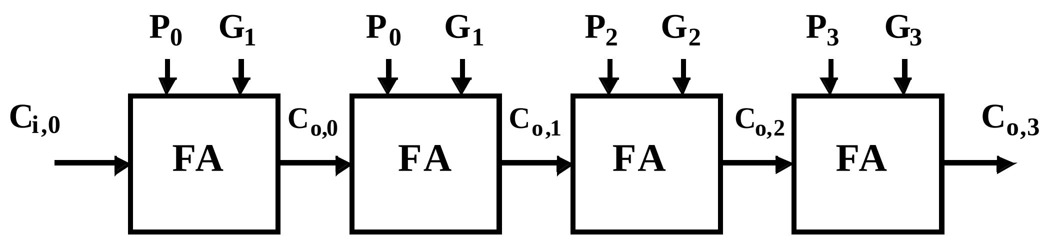
Given an expression of delay (symbols, not numbers) as a function of the number of bits

Manchester Carry Chain

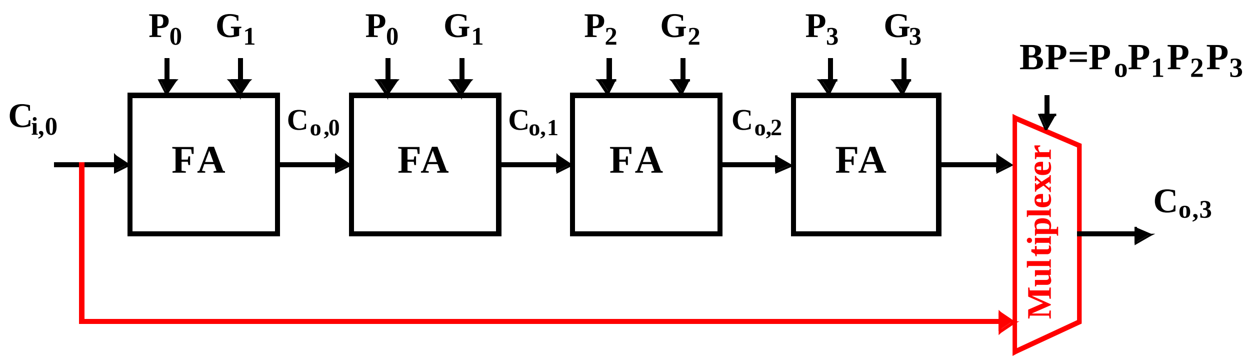
Stick Diagram



Carry-Bypass Adder

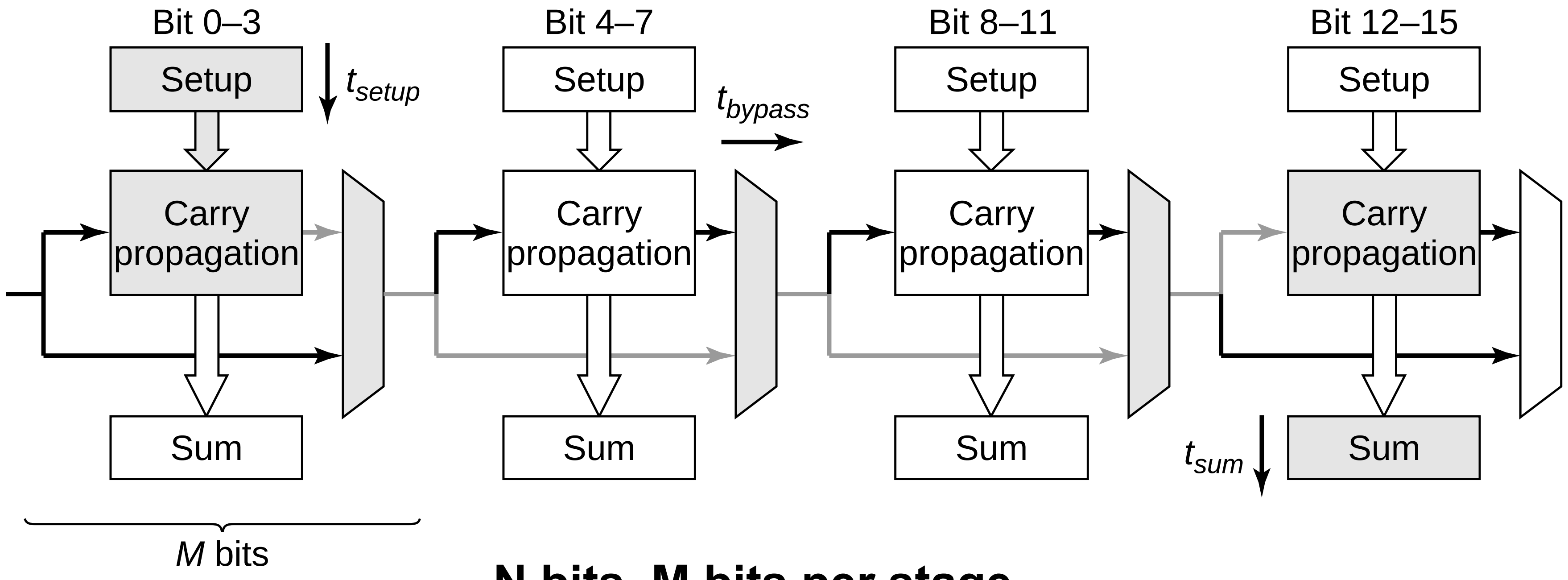


Also called Carry-Skip



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

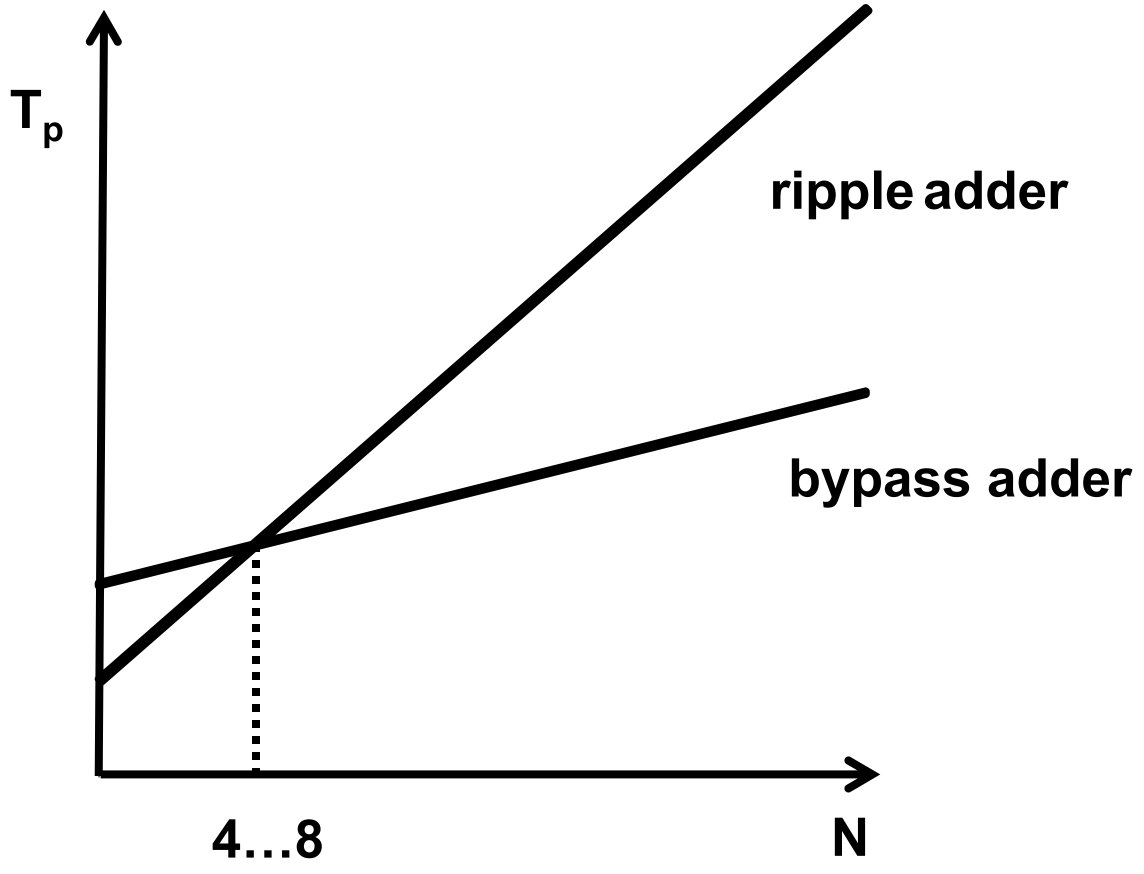
Carry-Bypass Adder (cont.)



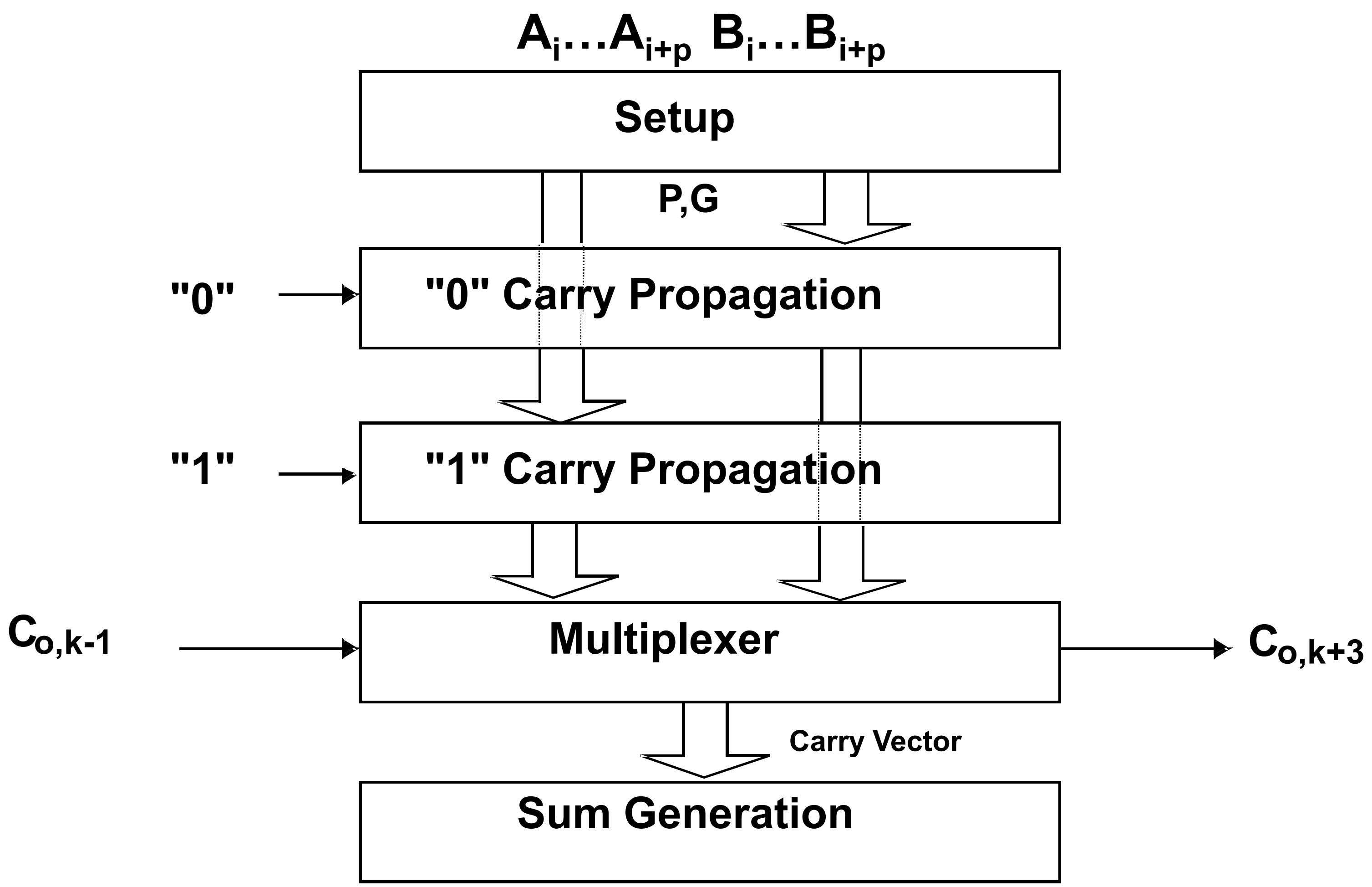
N bits, M bits per stage

$$t_{adder} = t_{setup} + M_{tcarry} + (N/M-1)t_{bypass} + (M-1)t_{carry} + t_{sum}$$

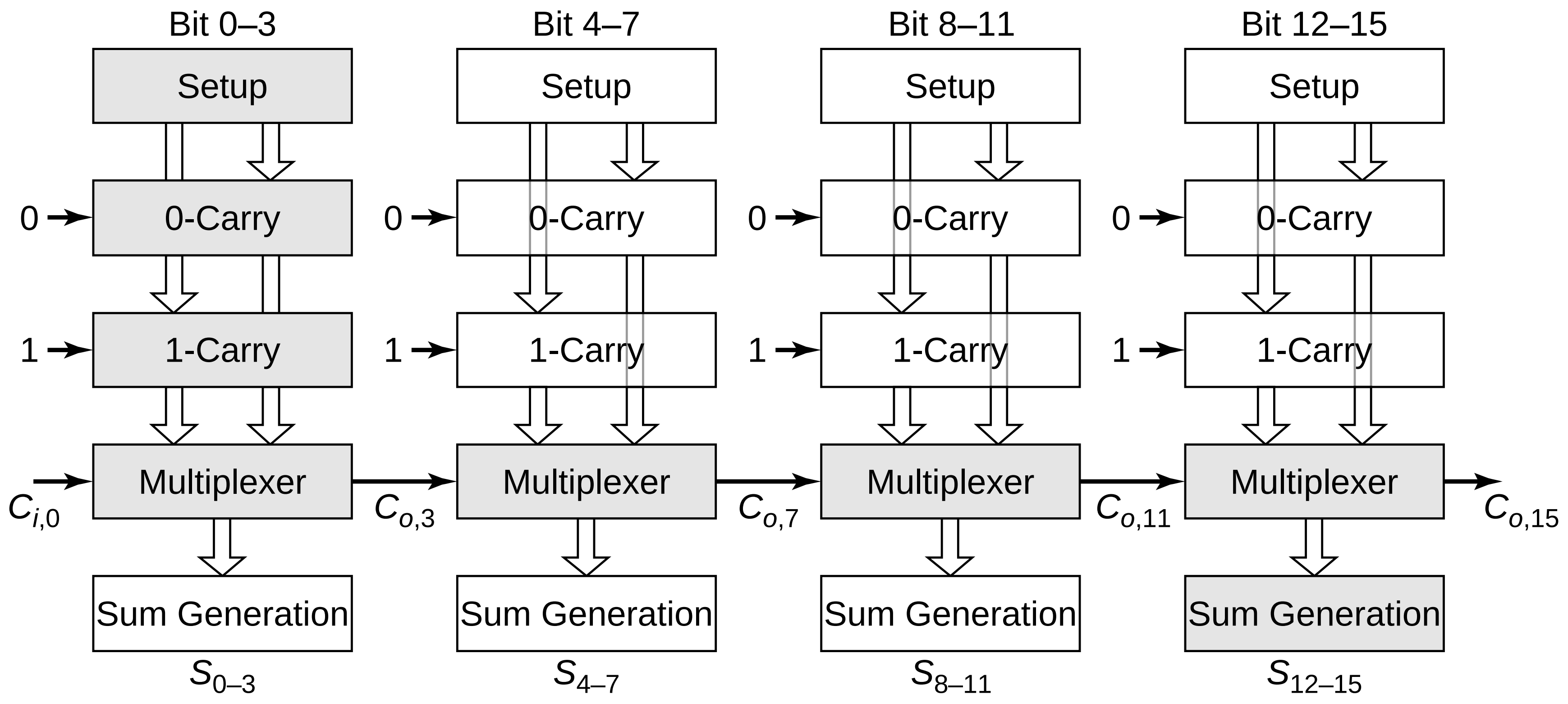
Carry Ripple versus Carry Bypass



Carry-Select Adder



Carry Select Adder: Critical Path

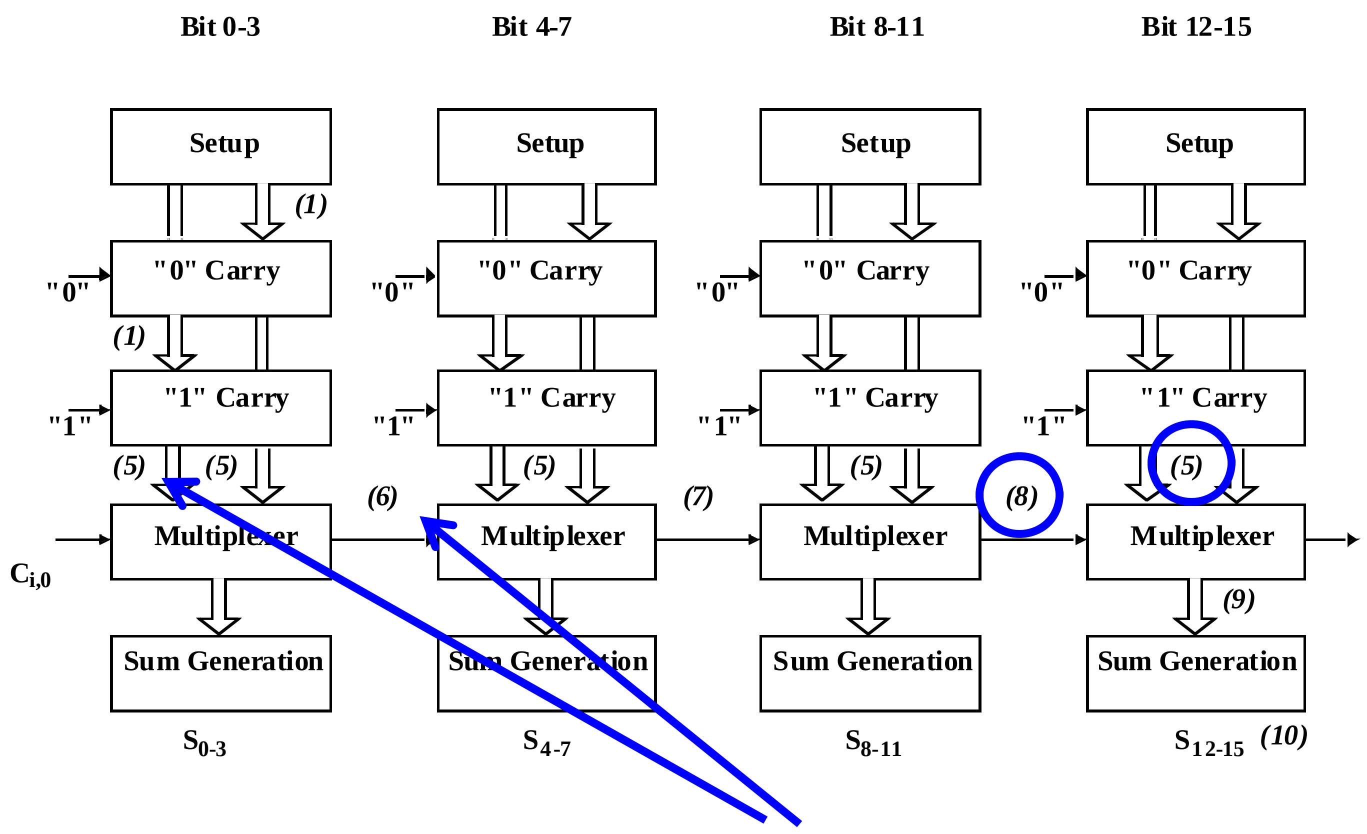


N bits, M bits/stage

t_{carry} is delay per bit

$$t_{add} = t_{setup} + Mt_{carry} + \left(\frac{N}{M}\right)t_{mux} + t_{sum}$$

Linear Carry Select

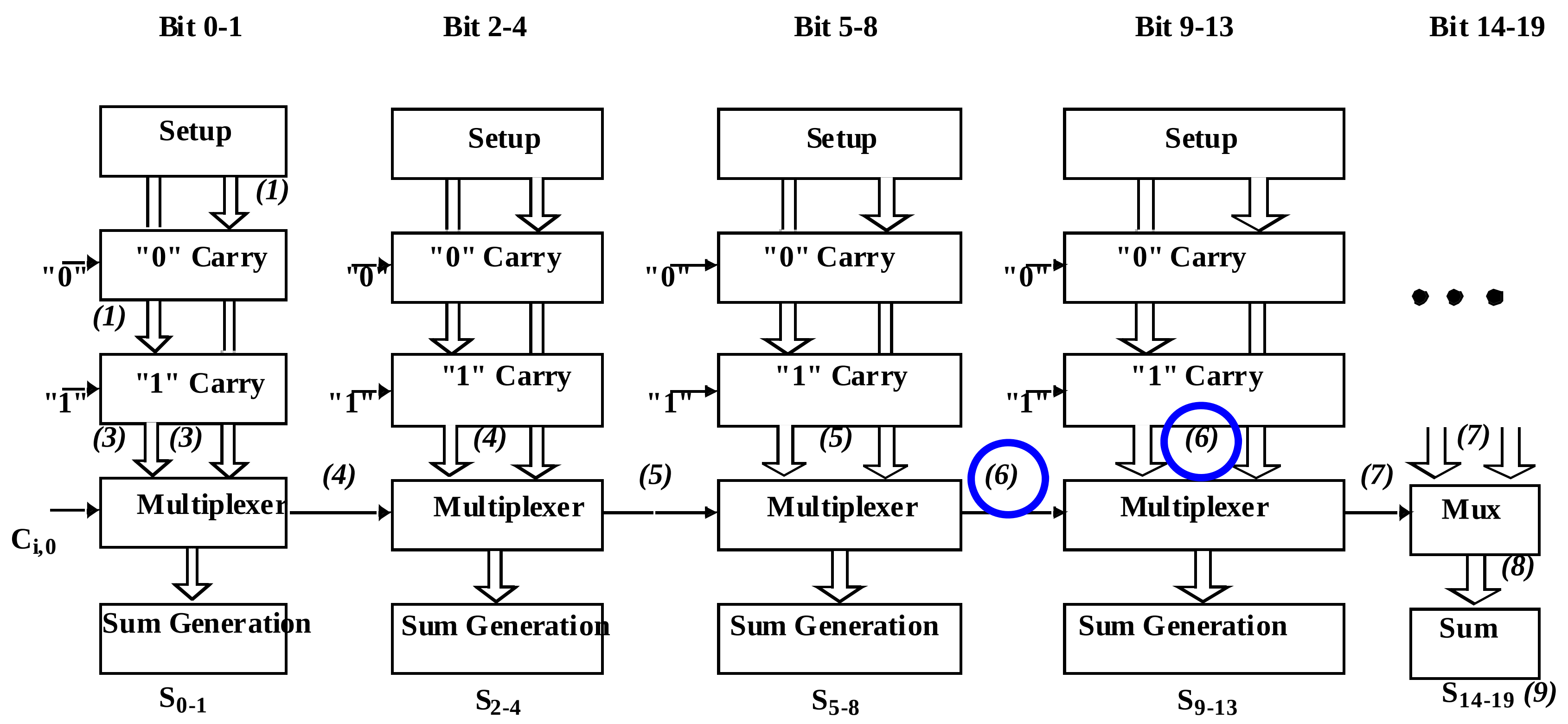


Assume unit delays per block, delays annoted as such

There is some slack

Square Root Carry Select

P **stages of increasing width**

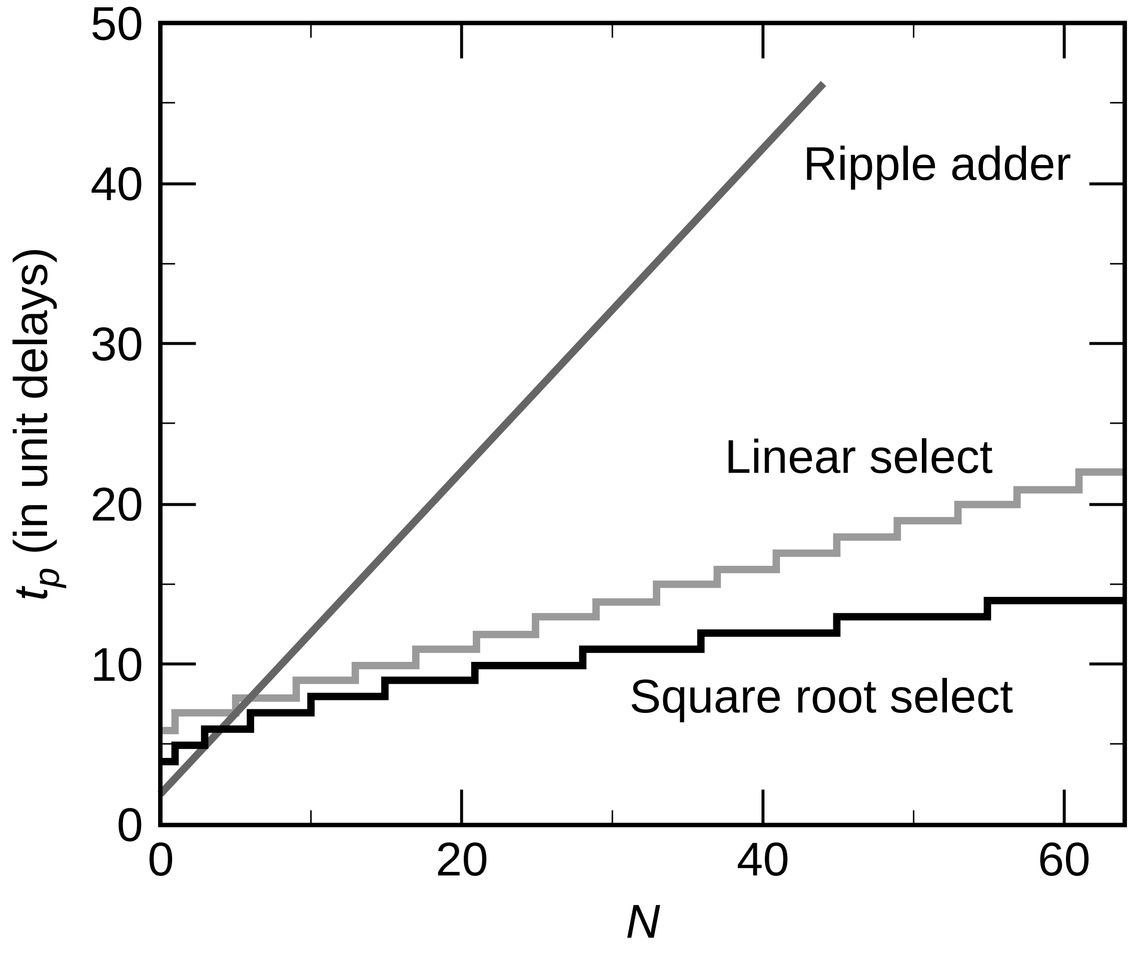


$$N = M + (M+1) + (M+2) + \ldots + (M+P-1)$$

$$= MP + \frac{P(P-1)}{2} = \frac{P^2}{2} + P\left(M - \tfrac{1}{2}\right) \approx \frac{P^2}{2} \quad if \quad N >> M$$

$$P = \sqrt{2N} \Rightarrow \qquad \boxed{t_{add} = t_{setup} + P \cdot t_{carry} + (\sqrt{2N})t_{mux} + t_{sum}}$$

Adder Delays - Comparison



Sub linear

Multiplier Design

- Multipliers are fundamental building blocks too
  - Digital Signal Processing (DSP): MP3 en/decoder, GSM, GPS, …
  - Data processing
  - Address arithmetic
  - …
- Good performance is key, often they are the performance bottleneck
- Multipliers are complex arrays of adders
- Many architectures
  - Basic Array Multiplier
  - Bit-serial
  - Booth-encoding multiplier
  - Baugh-Wooley multiplier
  - Wallace tree multiplier
  - …
- Design trade-offs, optimization
  - Architecture level, Logic level, Circuit level, Layout level

The Binary Multiplication

$$X = \sum_{i=0}^{M-1} X_i 2^i \qquad Y = \sum_{j=0}^{N-1} Y_j 2^j$$

$$Z = X \times Y = \sum_{k=0}^{M+N-1} Z_k 2^k$$

$$= \left(\sum_{i=0}^{M-1} X_i 2^i\right)\left(\sum_{j=0}^{N-1} Y_j 2^j\right)$$

$$= \underbrace{\sum_{i=0}^{M-1} \Bigl(\sum_{j=0}^{N-1}}_{\text{ADD}} \underbrace{X_i Y_j}_{\text{AND}} \underbrace{2^{i+j}}_{\text{SHIFT}} \Bigr)$$

Example: 42 x 11 = 462

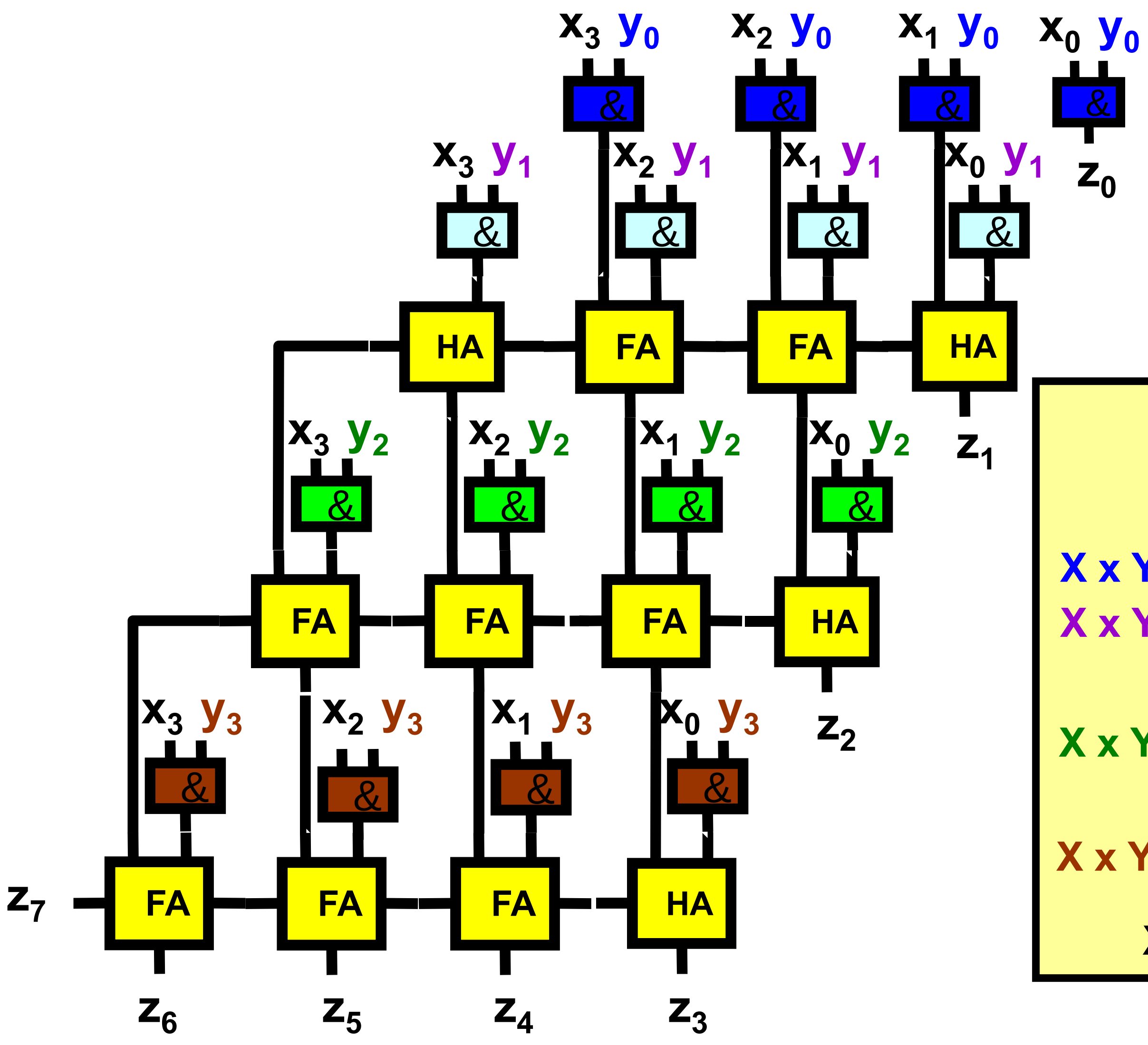
```
      1 0 1 0 1 0
          1 0 1 1  x
      -----------
      1 0 1 0 1 0
    1 0 1 0 1 0
  0 0 0 0 0 0
1 0 1 0 1 0
-----------------  +
1 1 1 0 0 1 1 1 0
```

Each partial product formed by bitwise AND operation

Partial products are shifted before being added

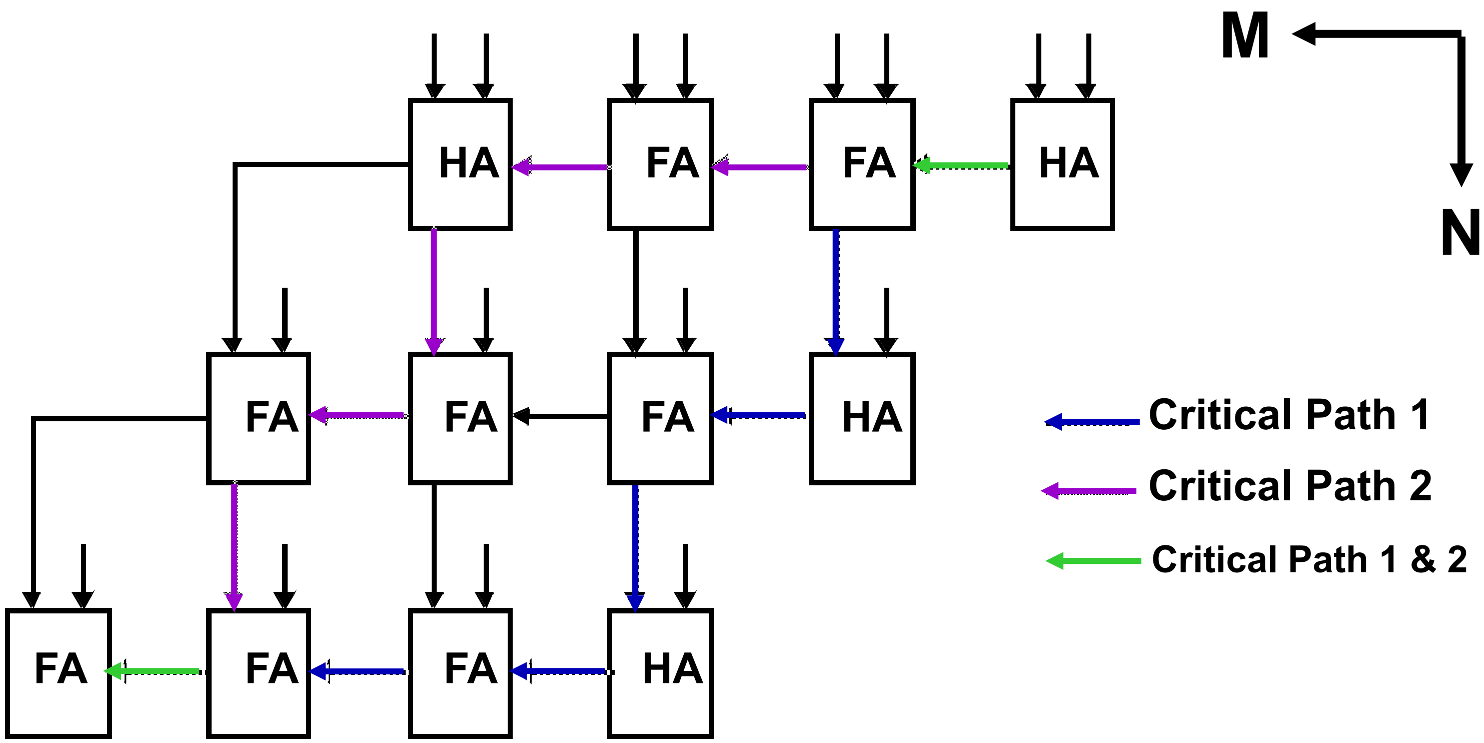
- **Conclusion:** similar to decimal multiplication
- $$X_{10} \times Y_{10} = \sum_{i=0}^{M-1}\left(\sum_{j=0}^{N-1} X_i Y_j 10^{i+j}\right)$$

The Array Multiplier



X				1	0	1	0	
Y				1	0	1	1	x
X x Y_0 x 2^0				1	0	1	0	
X x Y_1 x 2^1			1	0	1	0		+
			0	1	1	1	0	
X x Y_2 x 2^2		0	0	0	0			+
		1	0	1	1	1	0	
X x Y_3 x 2^3	1	0	1	0				+
X x Y	1	0	0	1	1	1	0	

The MxN Array Multiplier — Critical Path



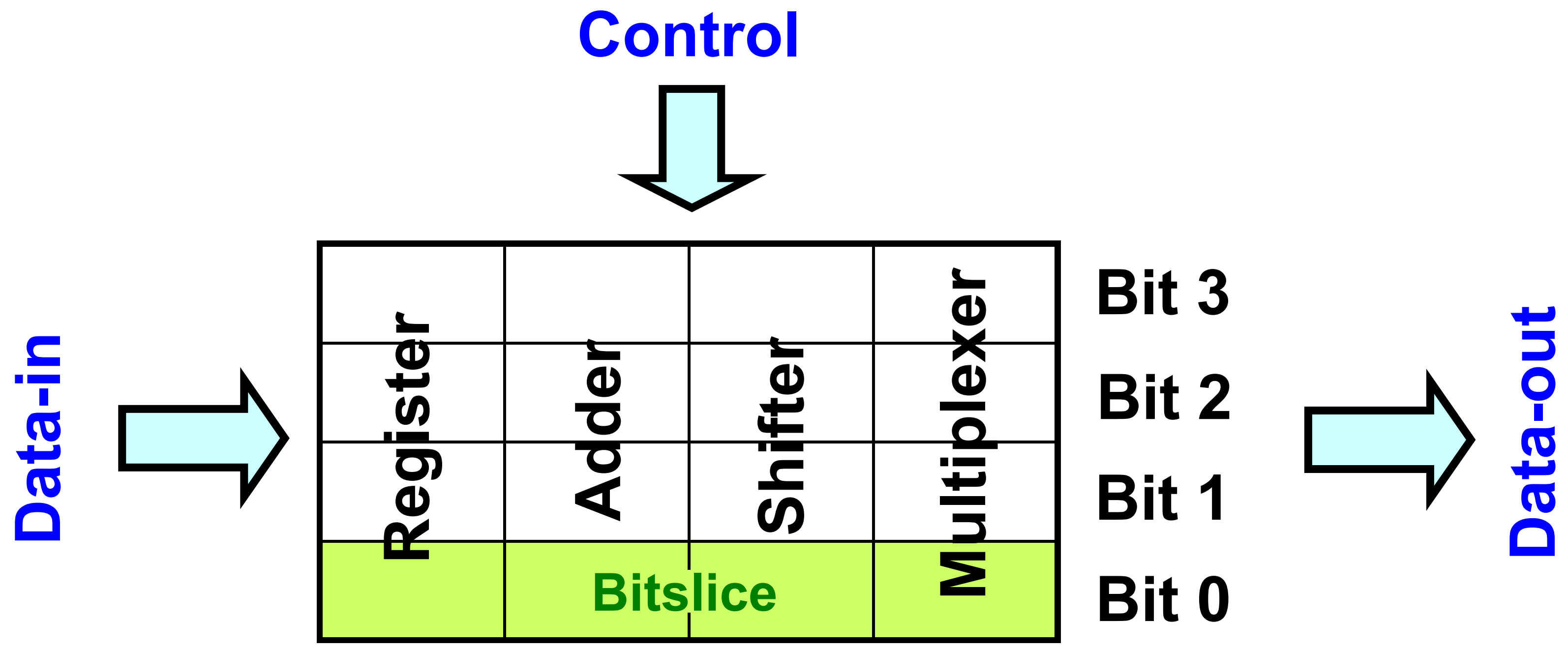
- $t_{mult} \approx [(M - 1) + (N - 2)]t_{carry} + (N - 1)t_{sum} + t_{and}$

Requires comparable carry and sum delays

➔ Different adder architectures

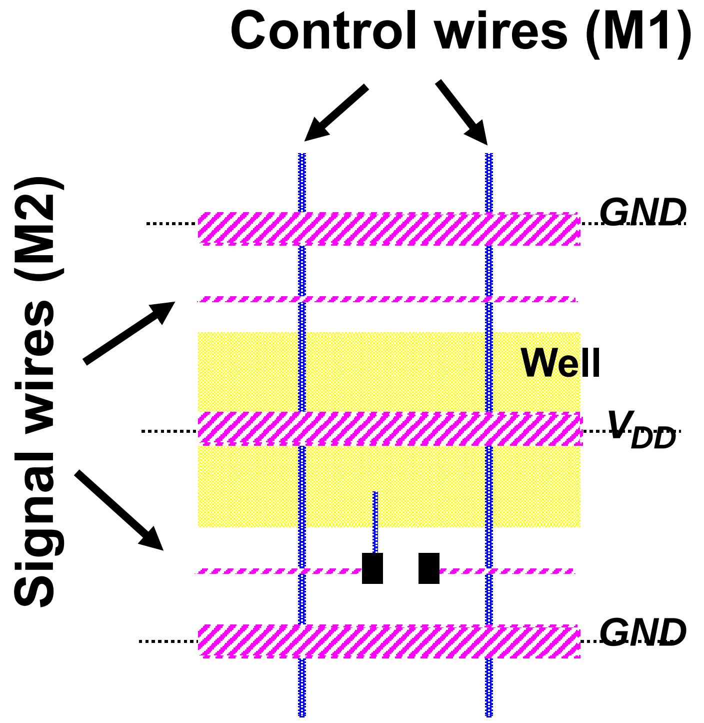
Layout Strategies (regularity)

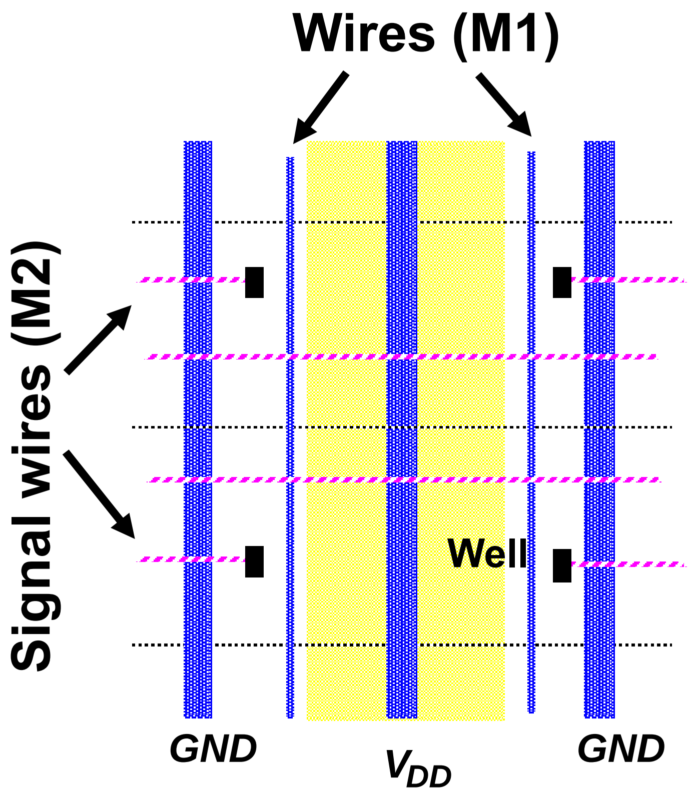
Bit-Sliced Design



- **Tile** identical processing elements
- **Rows for each bit**
- **Columns for each function**
- **Control from top (often with *control-slice*)**
- **(Example orientation)**

Layout Strategies for Bit-Sliced Datapaths





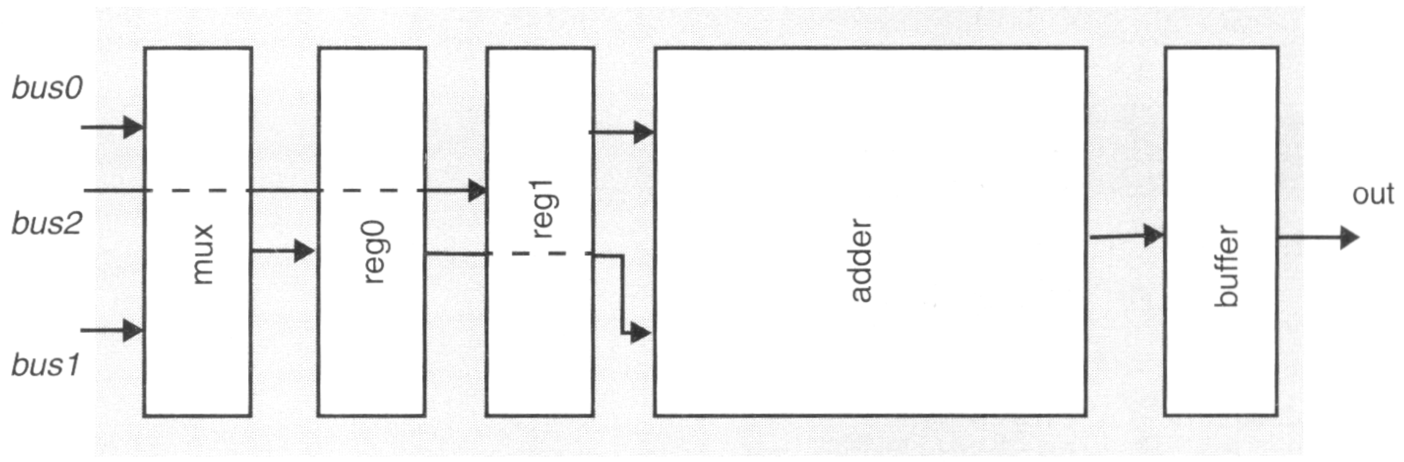
Approach I —

Signal and power lines parallel

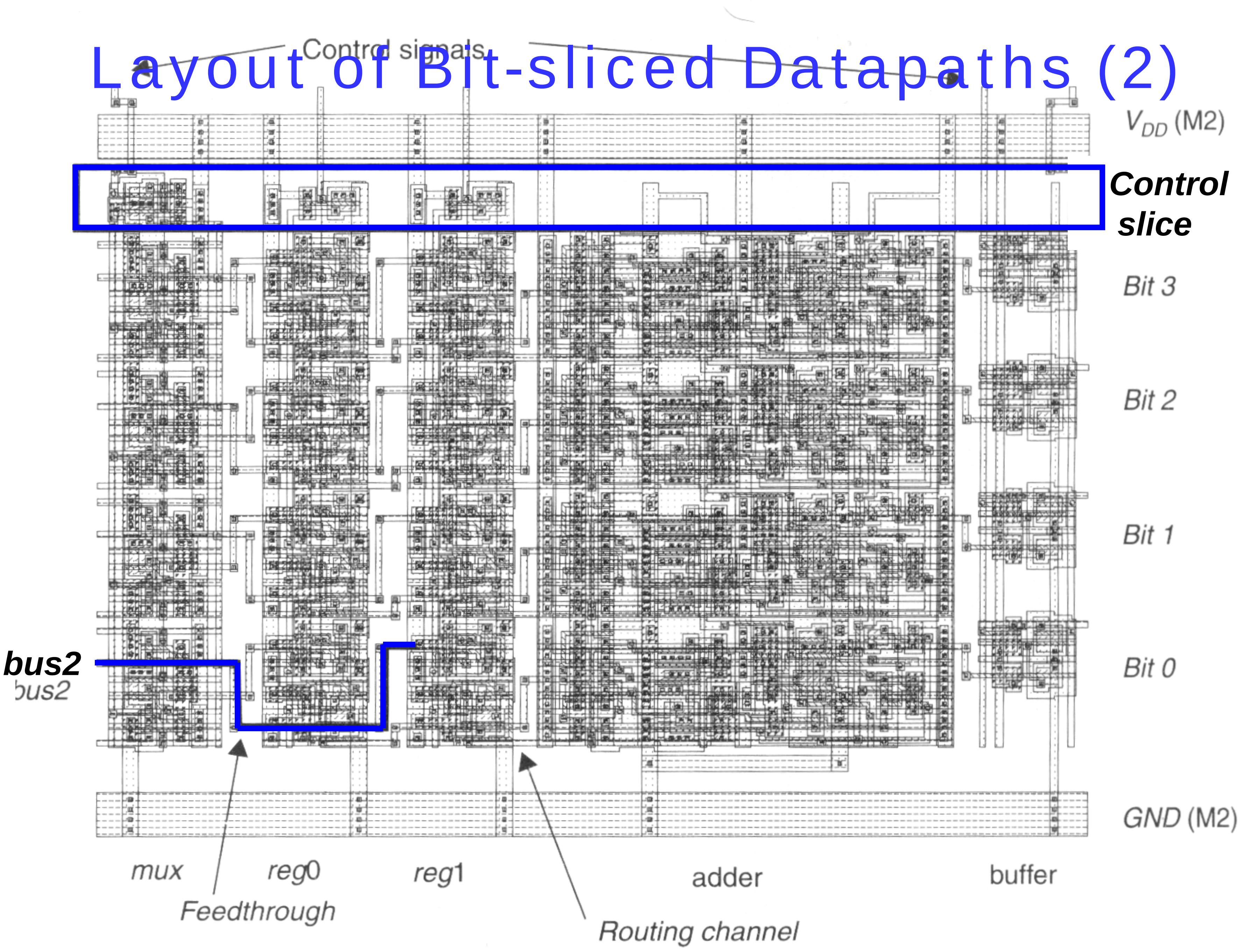
Approach II —

Signal and power lines perpendicular

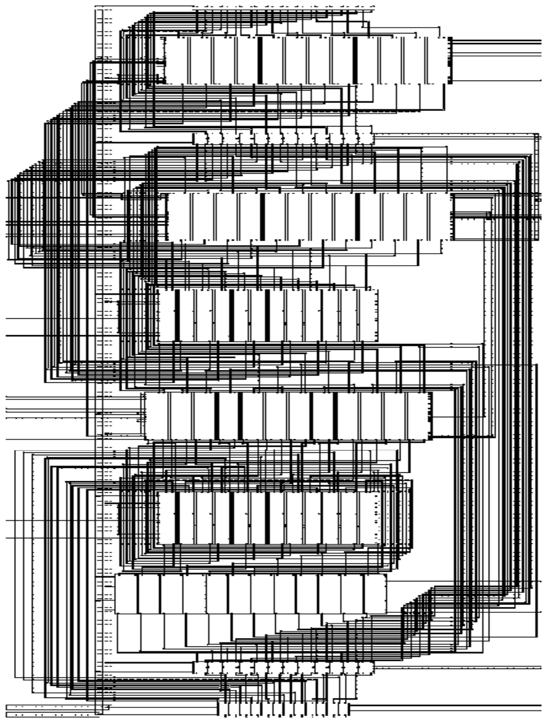
Layout of Bit-sliced Datapaths



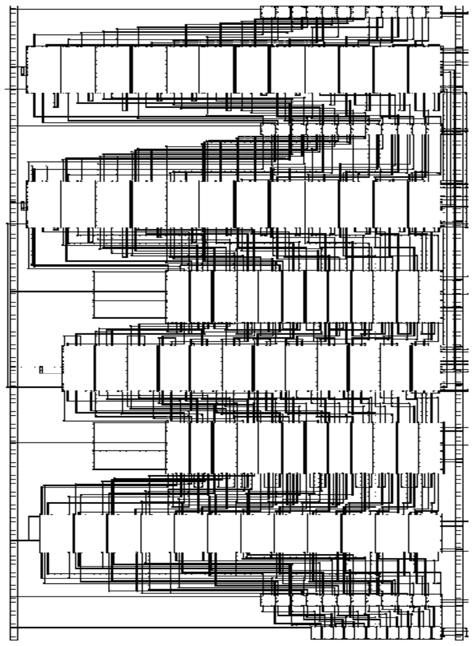
Layout of Bit-sliced Datapaths (2)



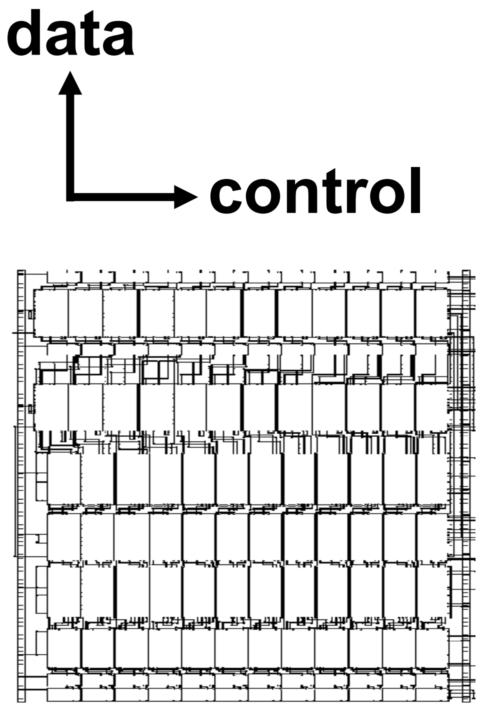
Layout of Bit-sliced Datapaths (3)



Unoptimized
Area: 4.2mm^2



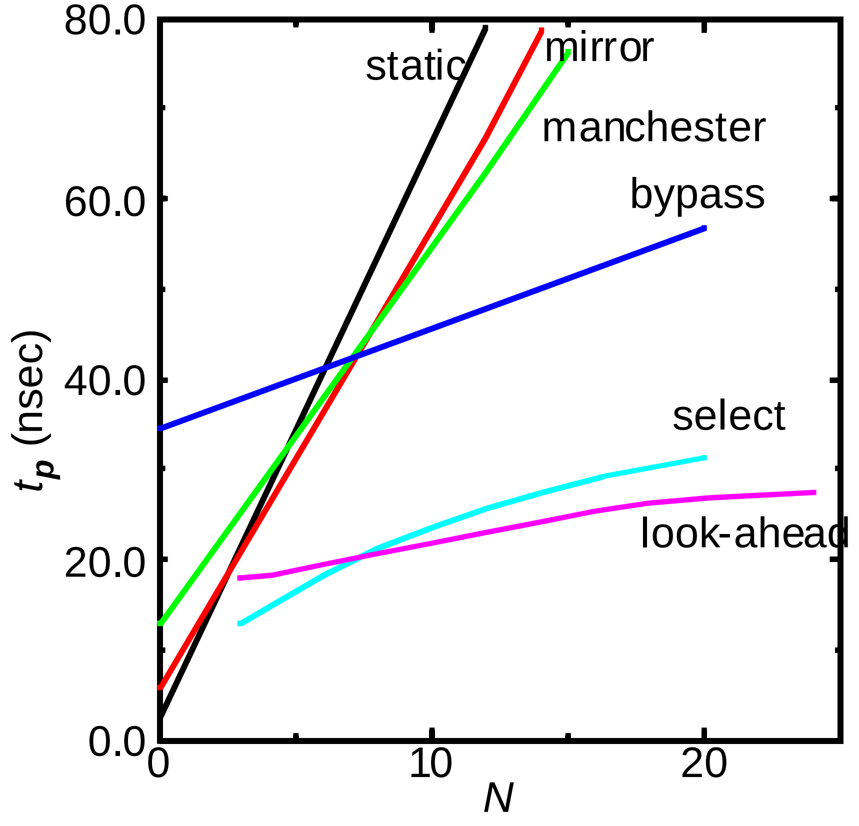
With feedthroughs
Area: 3.2mm^2

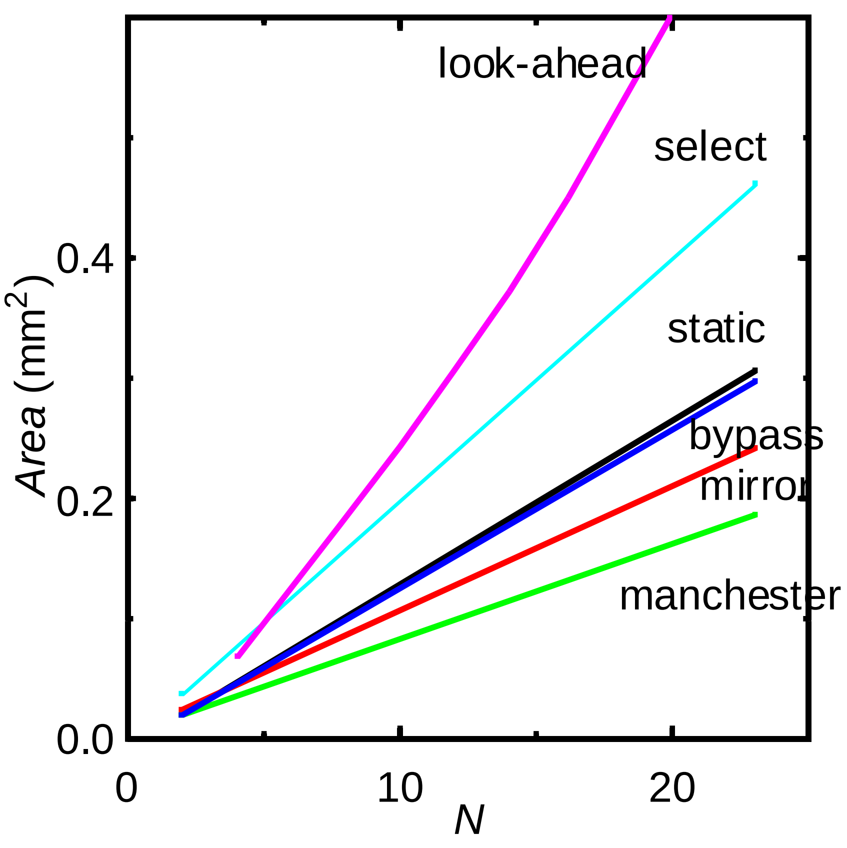


+ Equalized cell height
Area: 2.2mm^2

- **Good layout really counts!**
- **Feedthroughs less (but still) useful with multiple metal layers**

Design as a Trade-Off





VLSI Design.

- Select right structure
- Determine and optimize critical timing path for speed
- Optimize rest for area (cost) and/or power and/or design time
- Consider layout aspects

Regularity and modularity are a VLSI designer's best friends

Summary.

- Background on Modular Design
  - Hierarchy, reuse, regularity
  - Architecture, bit-slicing
- Adder Design
- Multiplier Design
- Shifter Design
- Layout Strategies (regularity)
- Design as a Trade-Off

Got further appreciation of some system level design issues?