MODULE 8

MODULARITY

Course Material for Modularity

Chapter 11

P = primair, I = Illustratie, O = overslaan

P	11.1	Introduction	560
P	11.2	Datapaths in Digital Processor Architectures	560-561
P	11.3	The Adder	561
P	11.3.1	The Binary Adder: Definitions	561-564
P	11.3.2	The Full Adder: Circuit Design Consideration	564-578
I	11.3.2	Manchester Carry Chain Adder	568-570
O	11.3.3	The Binary Adder: Logic Design Considerations	571-586
P	11.4	The Multiplier	586
P	11.4.1	The Multiplier: Definitions	586-587
P	11.4.2	Partial-Product Generation	587-589
P	11.4.3	Partial-Product Accumulation	589-592
O		The Tree Multiplier	592-593
I	11.4.4	Final Addition	593-594
P	11.5	The Shifter	595
P	11.5.1	Barrel Shifter	595-596
P	11.5.2	Logarithmic Shifter	596
O	11.6	Other Arithmetic Operators	596-600
O	11.7	Power and speed trade-offs	600-618
P	11.8	Perspective: Design as a Trade-off	618-619
P	11.9	Summary	619-620

2008-2009: the multiplier and shifter (§11.4-5) will be skipped

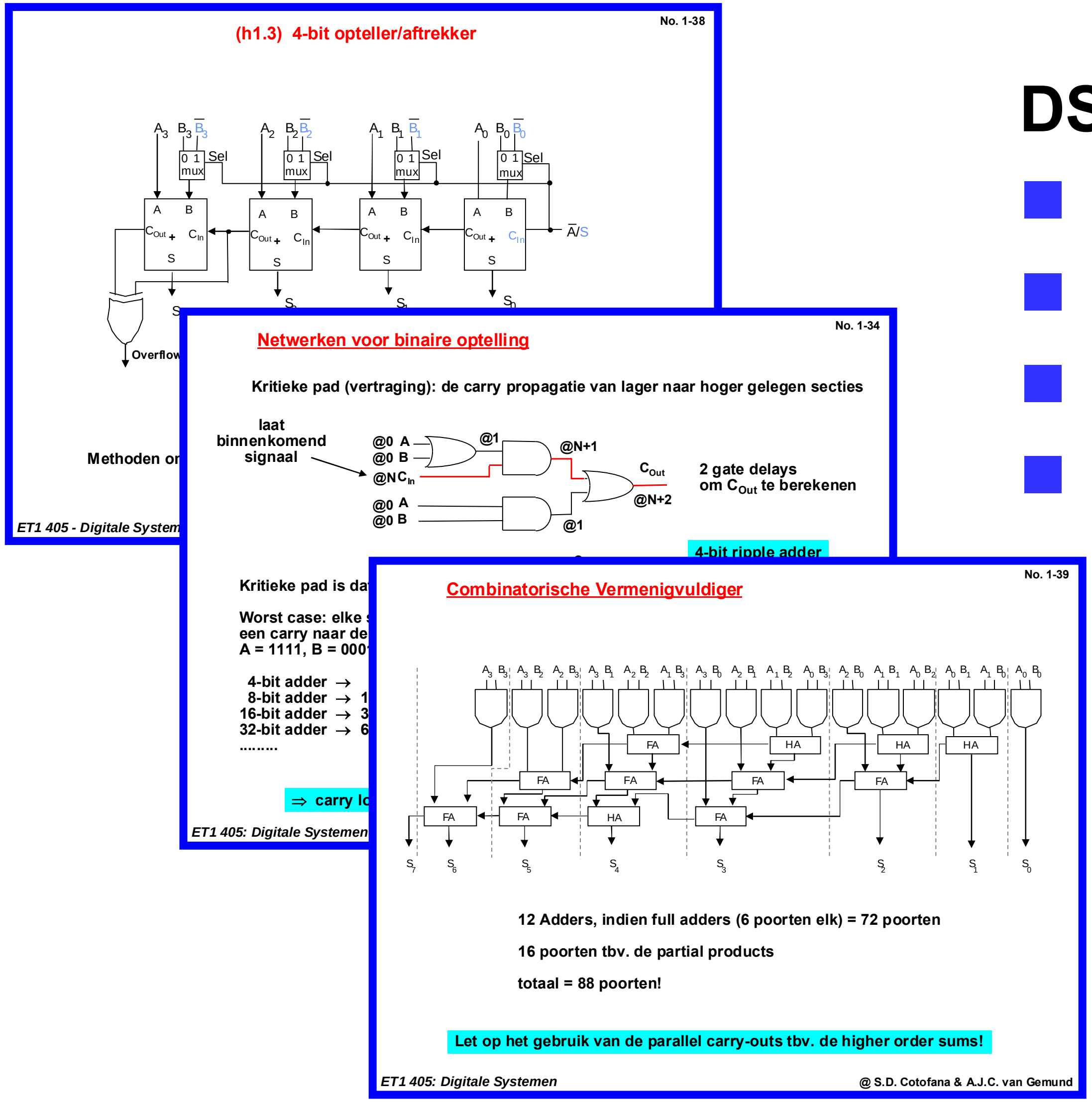
Outline

- 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

Arithmetic Circuits



No. 1-38

(h1.3) 4-bit opteller/aftrekker

Methoden or

ET1 405 - Digitale System

No. 1-34

Netwerken voor binaire optelling

Kritieke pad (vertraging): de carry propagatie van lager naar hoger gelegen secties

2 gate delays om C_{Out} te berekenen

4-bit ripple adder

Kritieke pad is da

Worst case: elke
een carry naar de
A = 1111, B = 000

4-bit adder →
8-bit adder → 1
16-bit adder → 3
32-bit adder → 6
………

⇒ carry lo

ET1 405: Digitale Systemen

No. 1-39

Combinatorische Vermenigvuldiger

12 Adders, indien full adders (6 poorten elk) = 72 poorten

16 poorten tbv. de partial products

totaal = 88 poorten!

Let op het gebruik van de parallel carry-outs tbv. de higher order sums!

ET1 405: Digitale Systemen

@ S.D. Cotofana & A.J.C. van Gemund

DS:

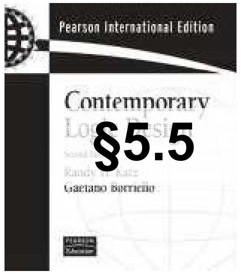
- Number systems
- Intro of full-adders
- Critical paths
- Intro comb. multiplier

Adder Design

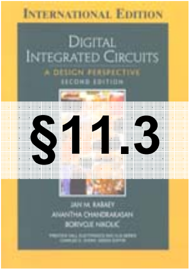
- Adders are fundamental building blocks
  - Digital filtering (DSP): MP3 en/decoder, GSM, GPS, …
  - Data processing
  - Multiplication
  - Address arithmetic
  - …
- Good performance is key
- Many architectures
  - ✓ Static adder
  - × Dynamic adder (Manchester Carry Chain)
  - × Pipelined Adder
 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

↑ Most effective

↓ Least effective

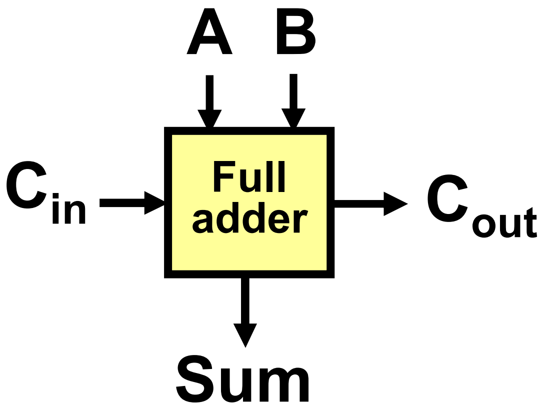


§5.5



§11.3

Full-Adder



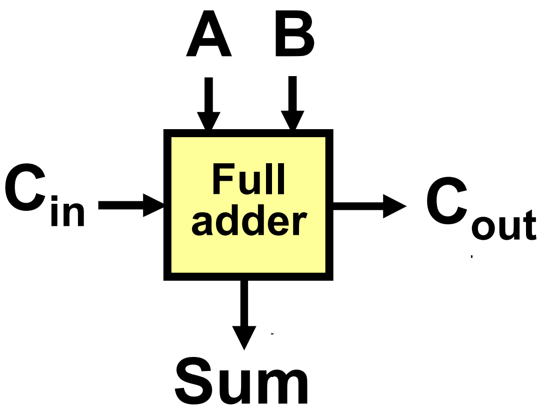
Add three one-bit numbers

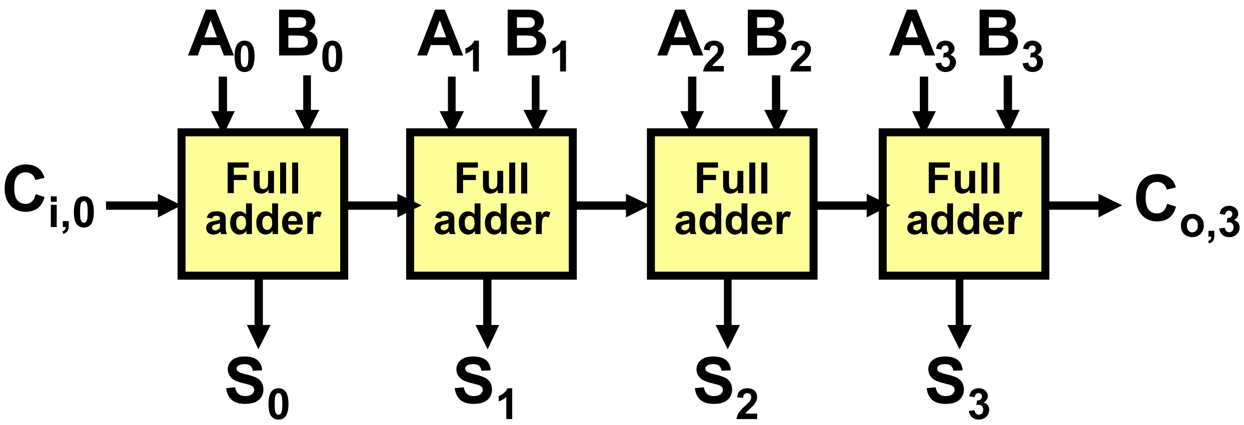
Equivalently: count # 1's in A, B, C_i

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

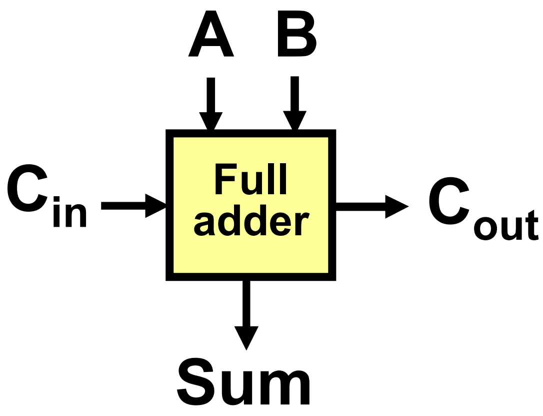
C_{in}	B	A	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The Ripple-Carry Adder





The Binary Adder



	$\bar{C}$	C
$\bar{A}\bar{B}$		1
$\bar{A}B$	1	
AB		1
$A\bar{B}$	1	

(a) SUM

	$\bar{C}$	C
$\bar{A}\bar{B}$		
$\bar{A}B$		1
AB	1	1
$A\bar{B}$		1

AB + BC + AC

(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

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

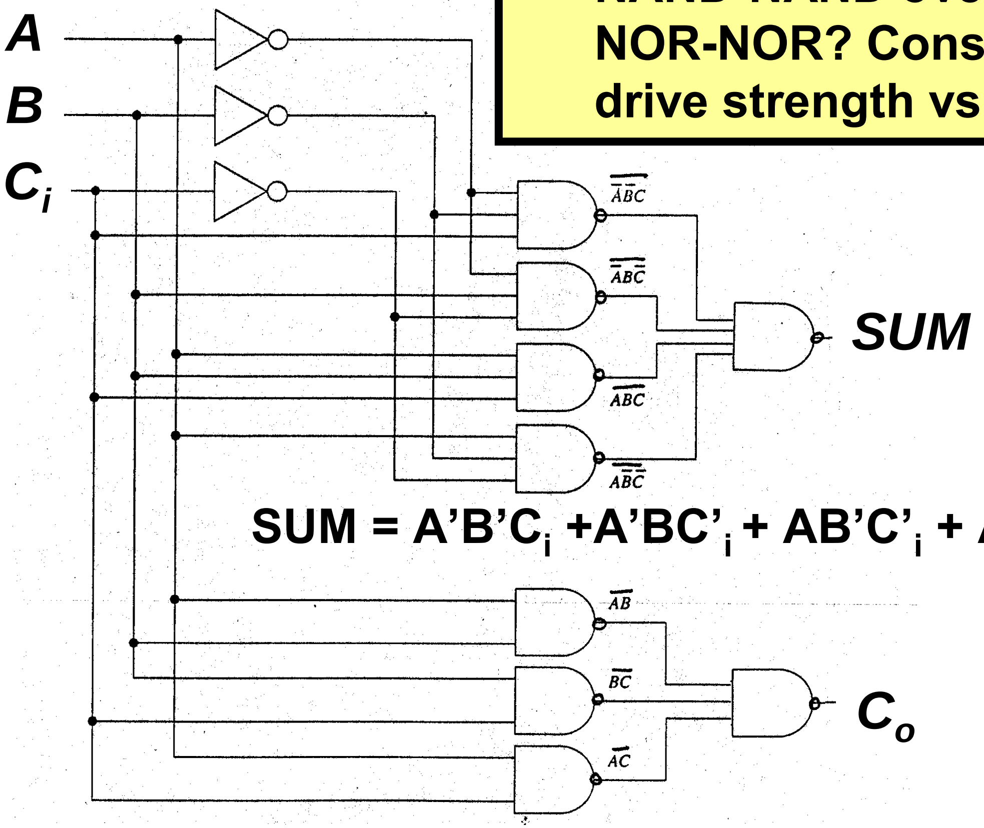
Transistor Count

3 × INVERT

3 × NAND-2

5 × NAND-3

1 × NAND-4



A
B
C_i
$\overline{\bar{A}\bar{B}C}$
$\overline{\bar{A}B\bar{C}}$
$\overline{A\bar{B}\bar{C}}$
$\overline{ABC}$
SUM
$\overline{AB}$
$\overline{BC}$
$\overline{AC}$
C_o

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

Can do better using more clever boolean factoring, but…

Full-Adder Boolean Factoring

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

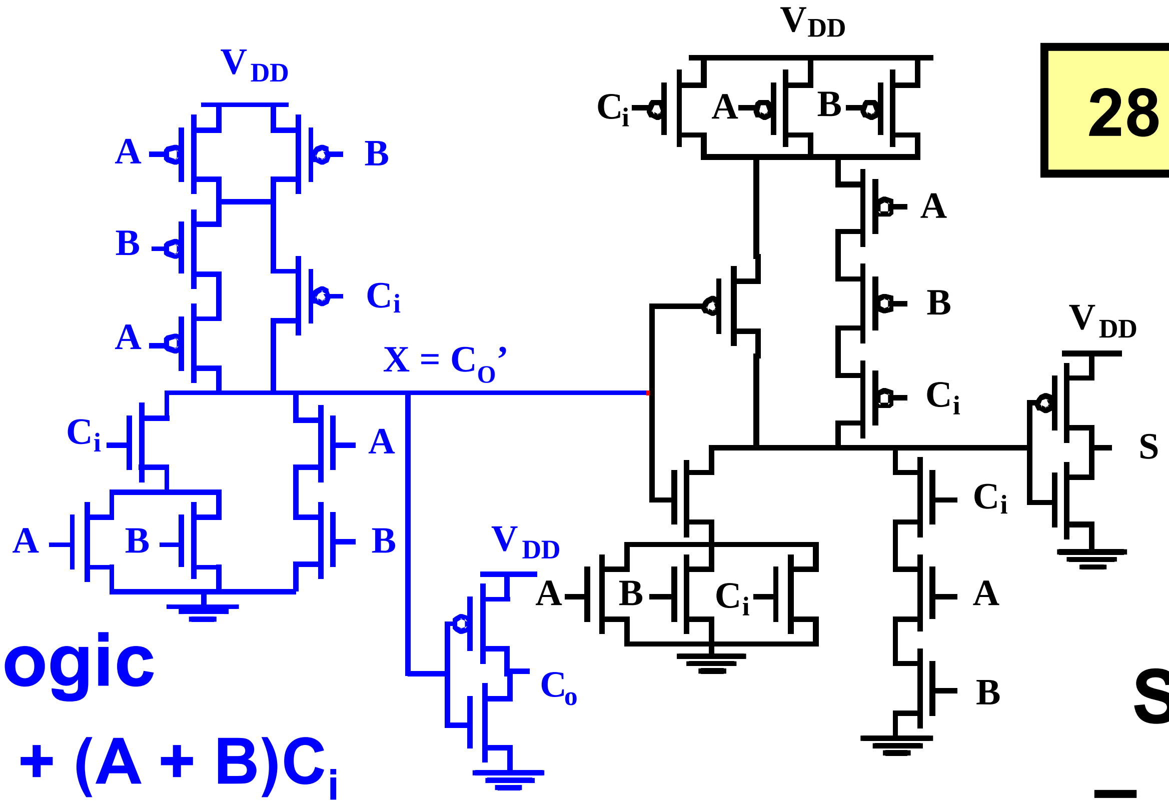
$$= ABC_i + \overline{C}_0(A + B + C_i)$$

$$C_0 = AB + BC_i + AC_i$$

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

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

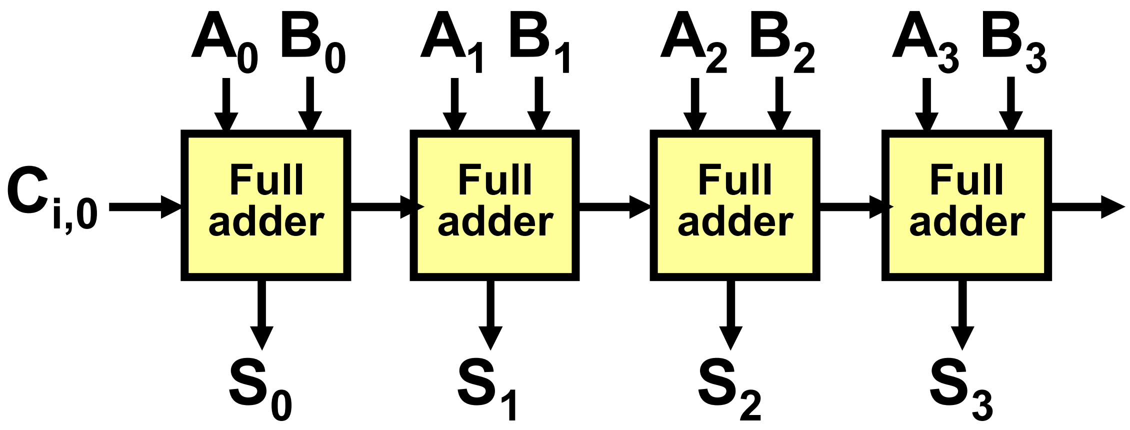
Carry logic

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

Sum logic

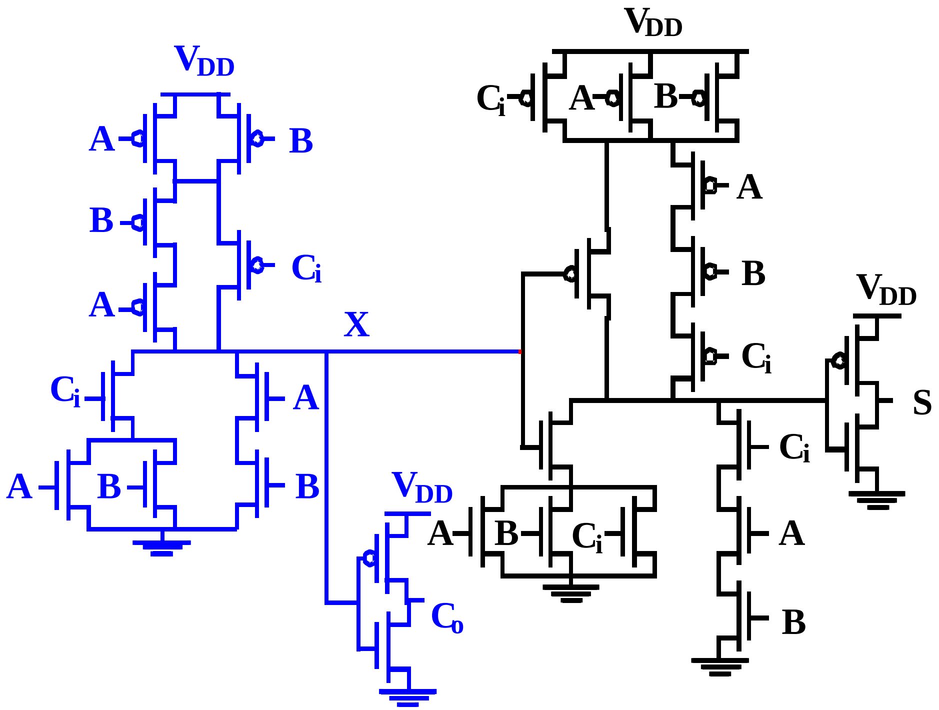
$$S = ABC_i + \overline{C}_0(A + B + C_i)$$

Ripple-Carry Adder Delay



- Worst case delay through full carry path (ripple carry)
- Linear with the number of bits (N)
- T_{adder} = (N-1) T_{carry} + 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

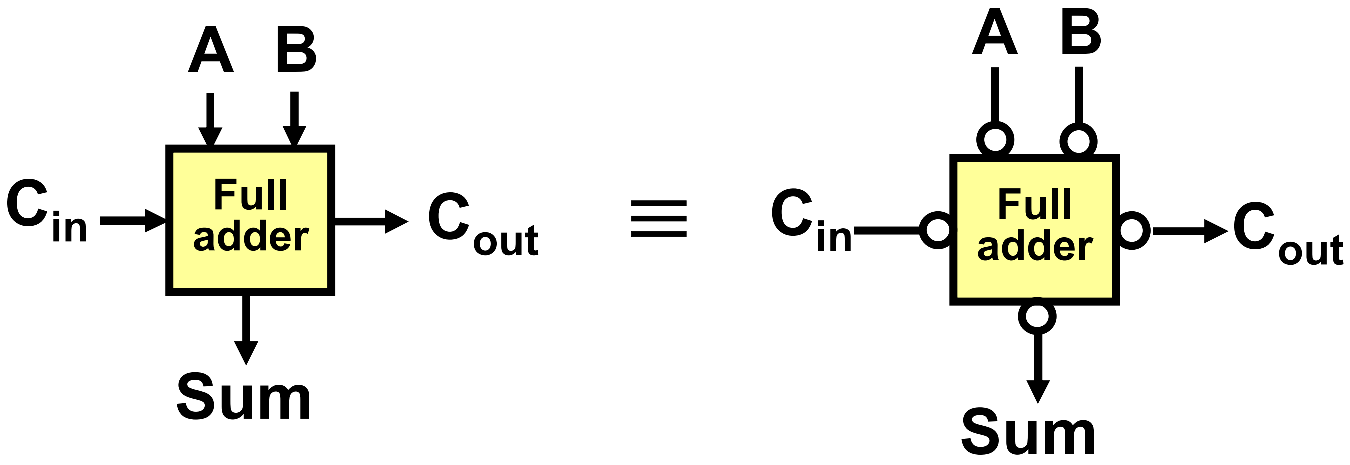
Adder Evaluation



Carry Chain:

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

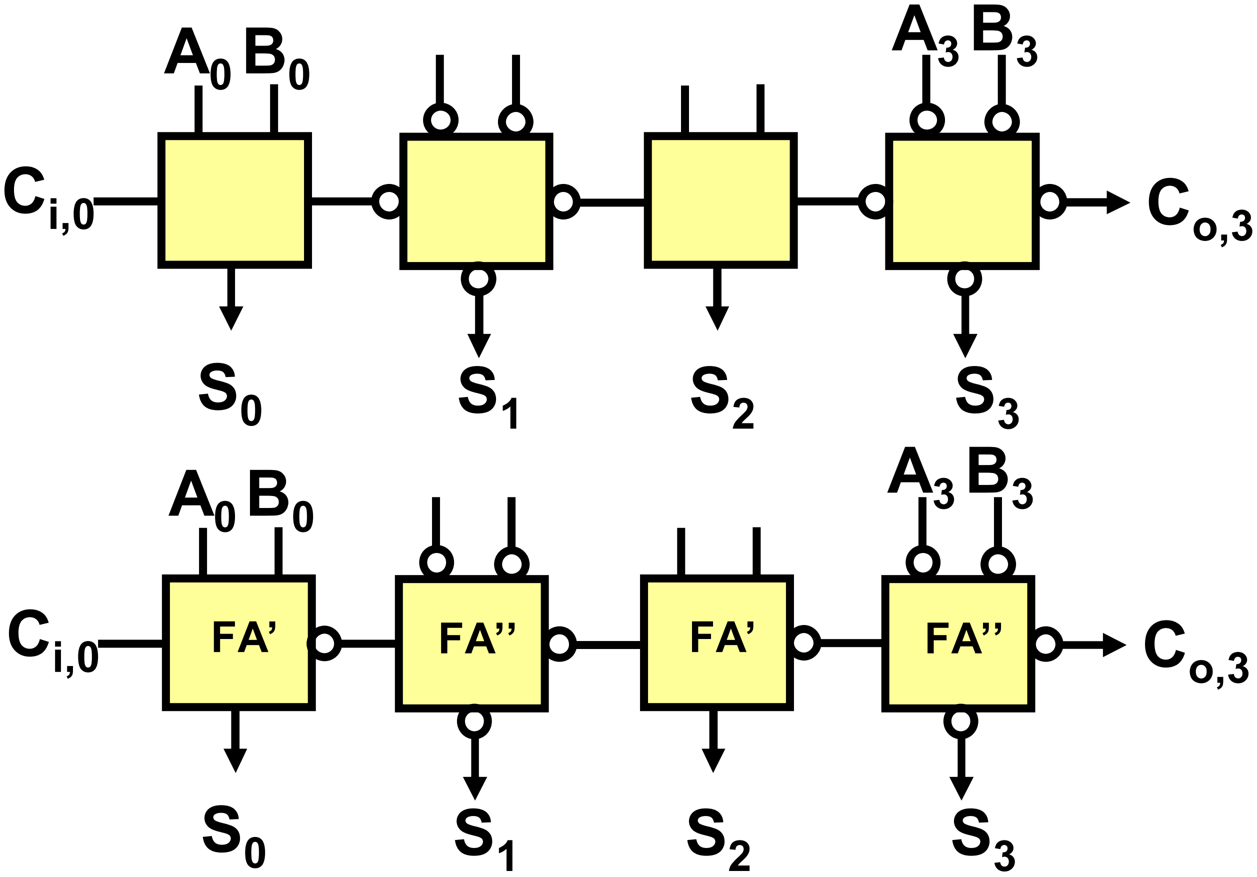
Inversion Property



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

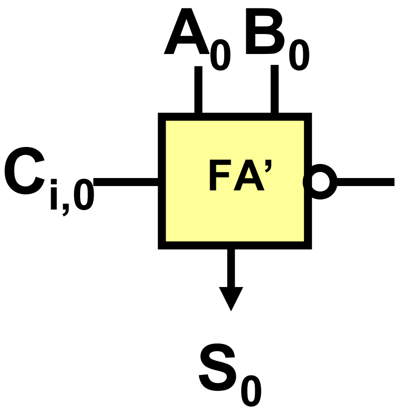
$$\overline{C_0}(A,B,C_i) = C_0(\overline{A},\overline{B},\overline{C_i})$$

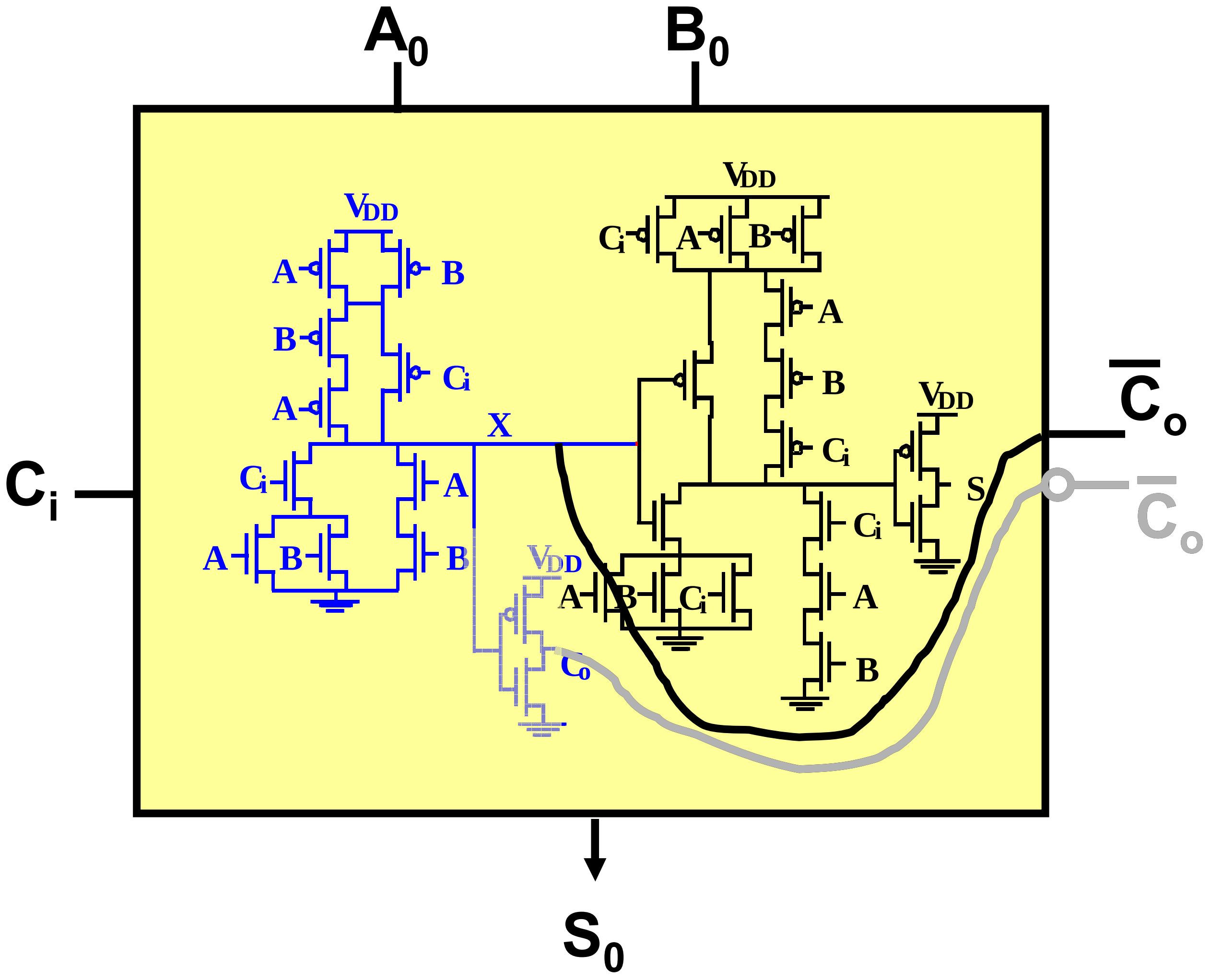
Minimize Critical Path by Reducing Inverting Stages



- **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.





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

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

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

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

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

Example: 42 x 11 = 462

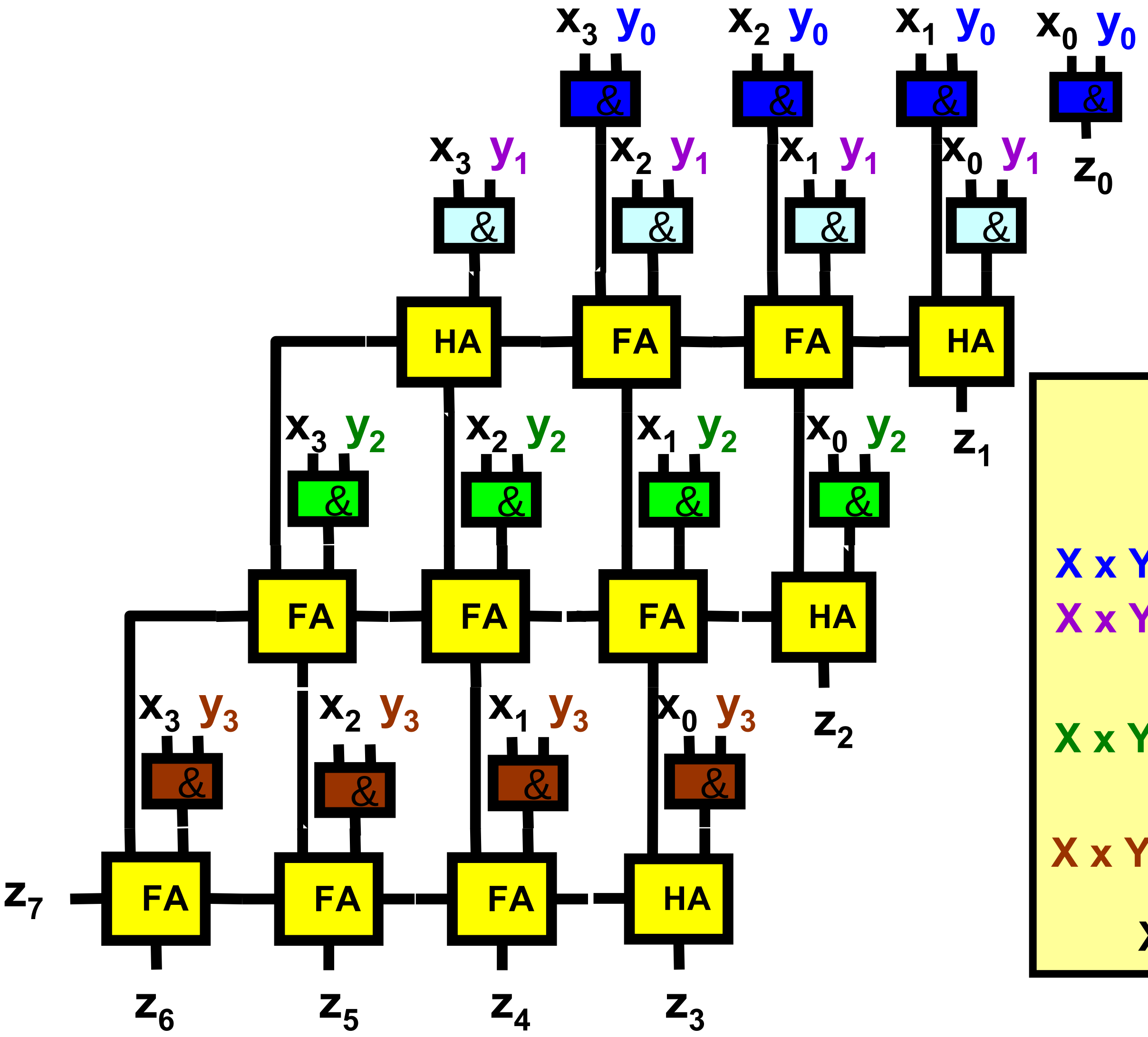
```
      1 0 1 0 1 0
          1 0 1 1  x
      -----------
      1 0 1 0 1 0
    1 0 1 0 1 0
  0 0 0 0 0 0
1 0 1 0 1 0
-----------------  +
1 1 1 0 0 1 1 1 0
```

Each partial product formed by bitwise AND operation

Partial products are shifted before being added

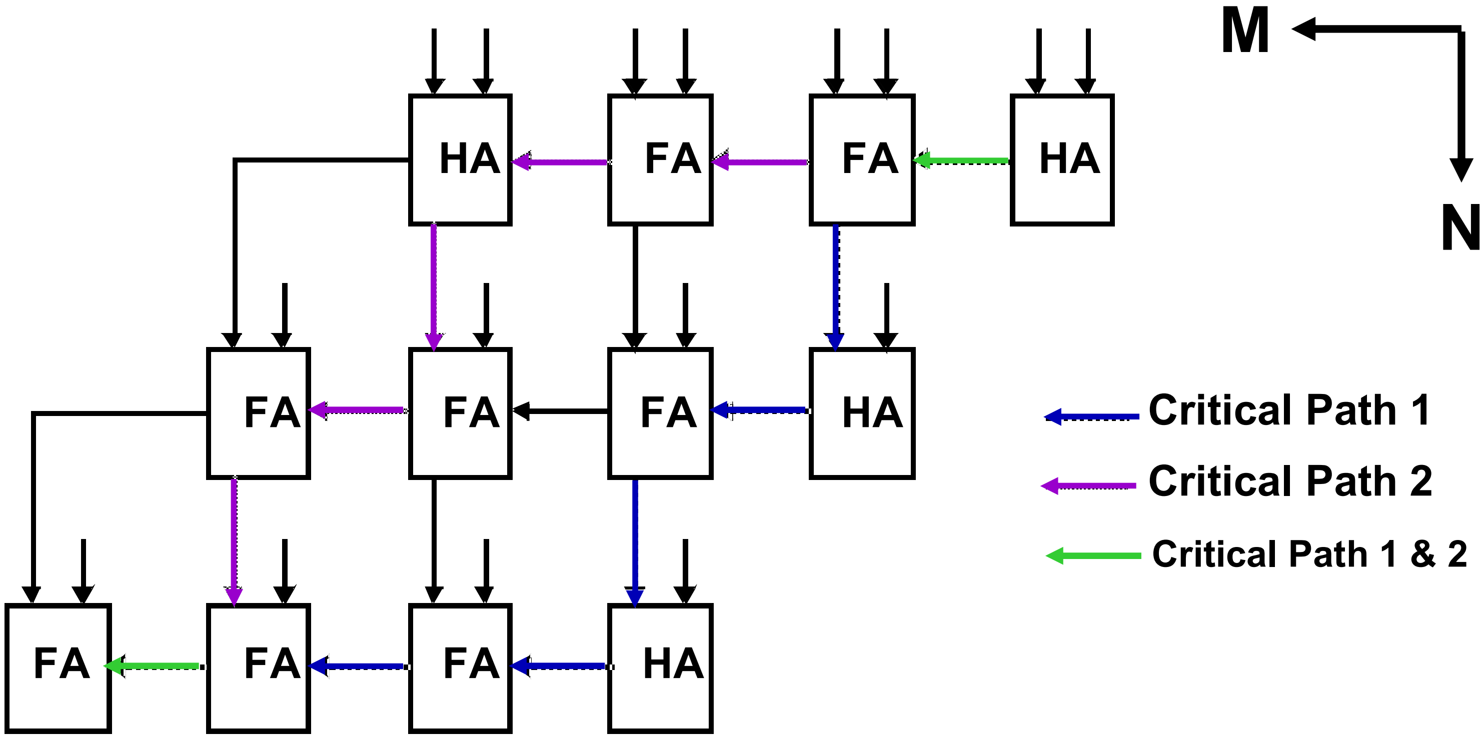
- **Conclusion: similar to decimal multiplication**
- $$X_{10} \times Y_{10} = \sum_{i=0}^{M-1} \left(\sum_{j=0}^{N-1} X_i Y_j 10^{i+j} \right)$$

The Array Multiplier



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

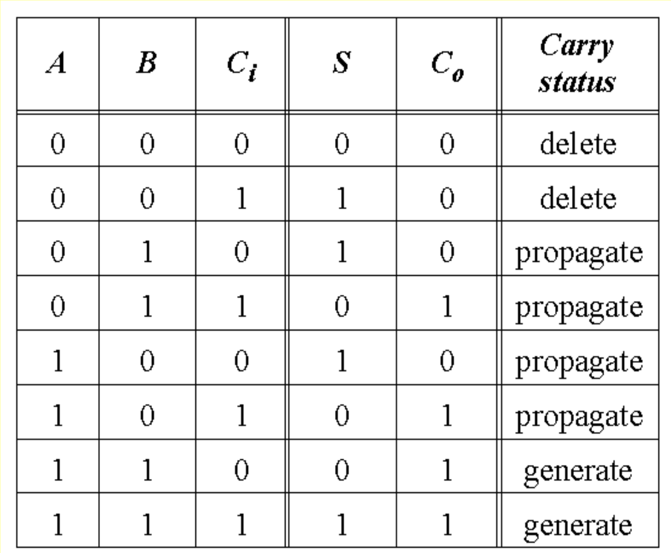
The MxN Array Multiplier — Critical Path



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

Adder Cells in Array Multiplier

Identical Delays for Carry and Sum

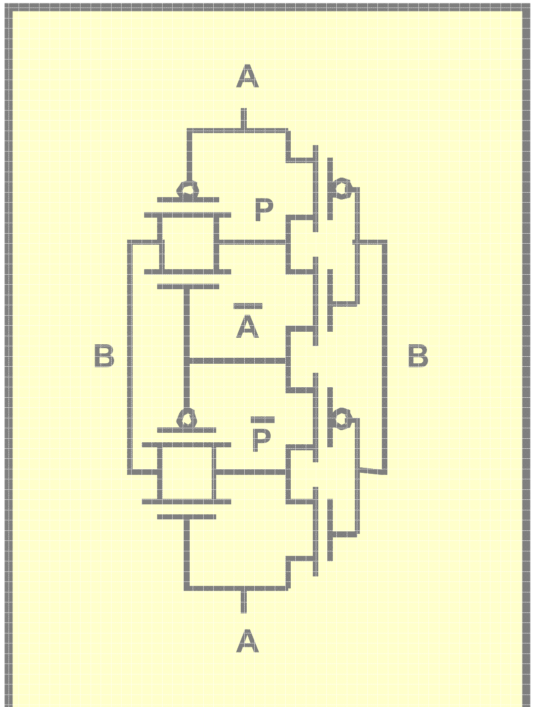


A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

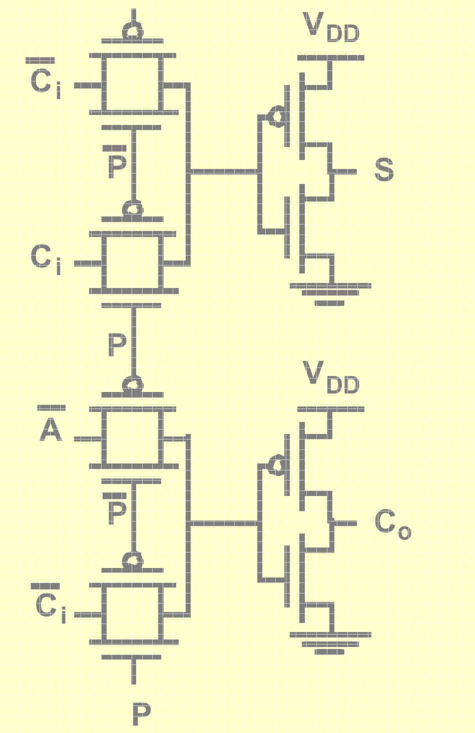
$P = A \oplus B$

If $P = 1$ then $S = \overline{C}_i$, $C_o = C_i$

If $P = 0$ then $S = C_i$, $C_o = A$

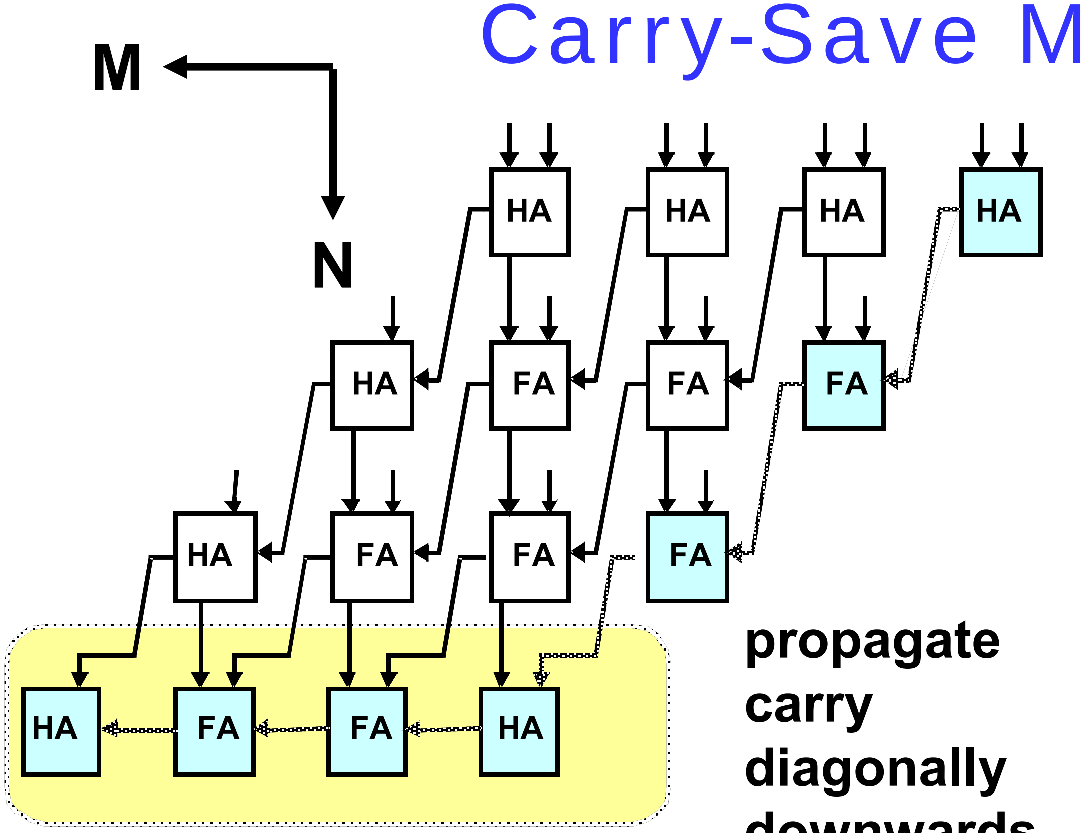


2 x transmission gate XOR for P, $\overline{P}$ (Fig 11.17)



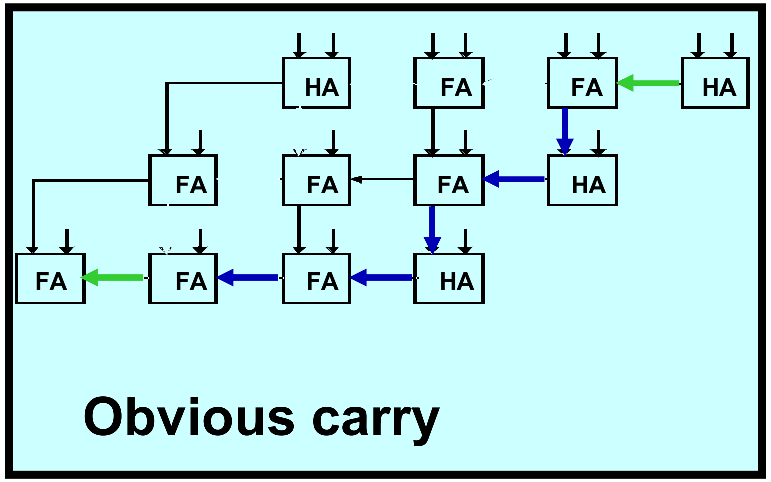
Multiplexers for S, C_o

Carry-Save Multiplier



Vector Merging Adder

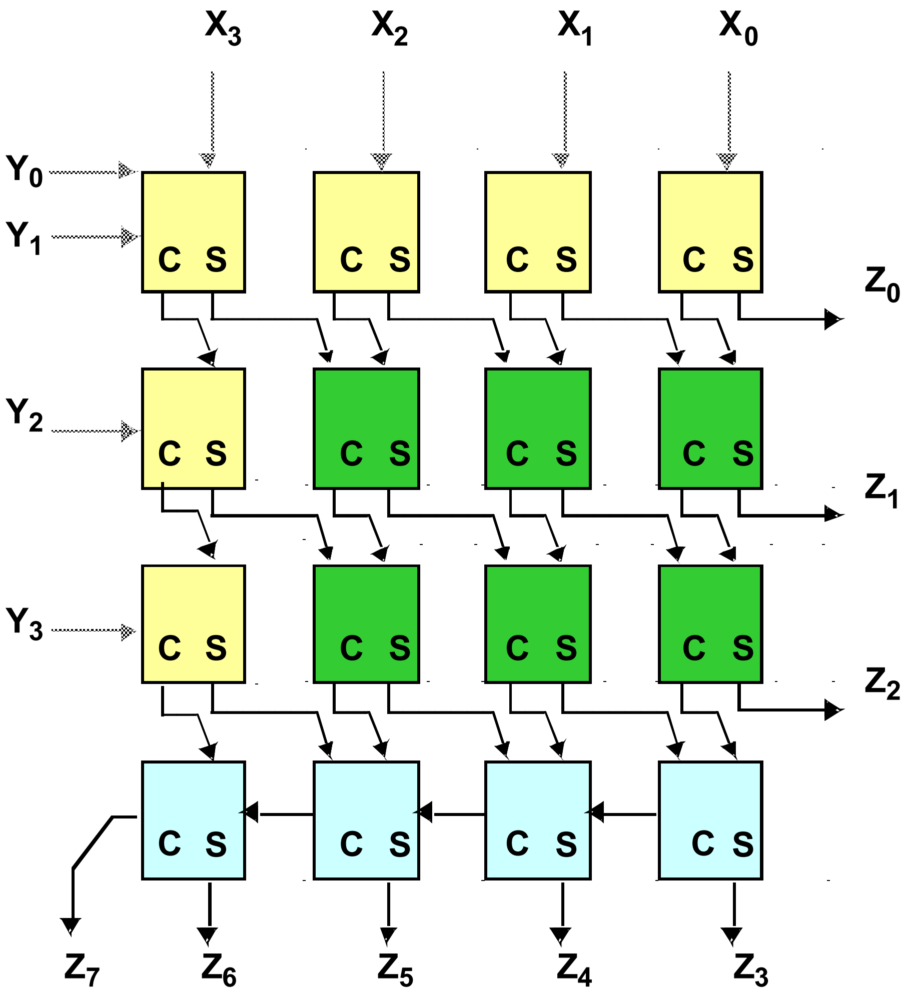
propagate carry diagonally downwards



Obvious carry

- $t_{mult} \approx (N - 1)t_{carry} + t_{and} + t_{merge}$ (assuming $t_{add} \approx t_{carry}$)
- Use **fastest possible adder** for final vector merging
- Will be larger, use more power, etc, **but need only one row!**

Multiplier Floorplan



🟨 HA Multiplier Cell

🟩 FA Multiplier Cell

🟦 Vector Merging Cell

X signals are routed vertically across each column (“broadcast”)

Y signal are broadcasted horizontally across rows

Regularity!

Multipliers — Summary.

- Optimization Goals Different Vs Binary Adder
- Once Again: Identify Critical Path
- Other possible techniques

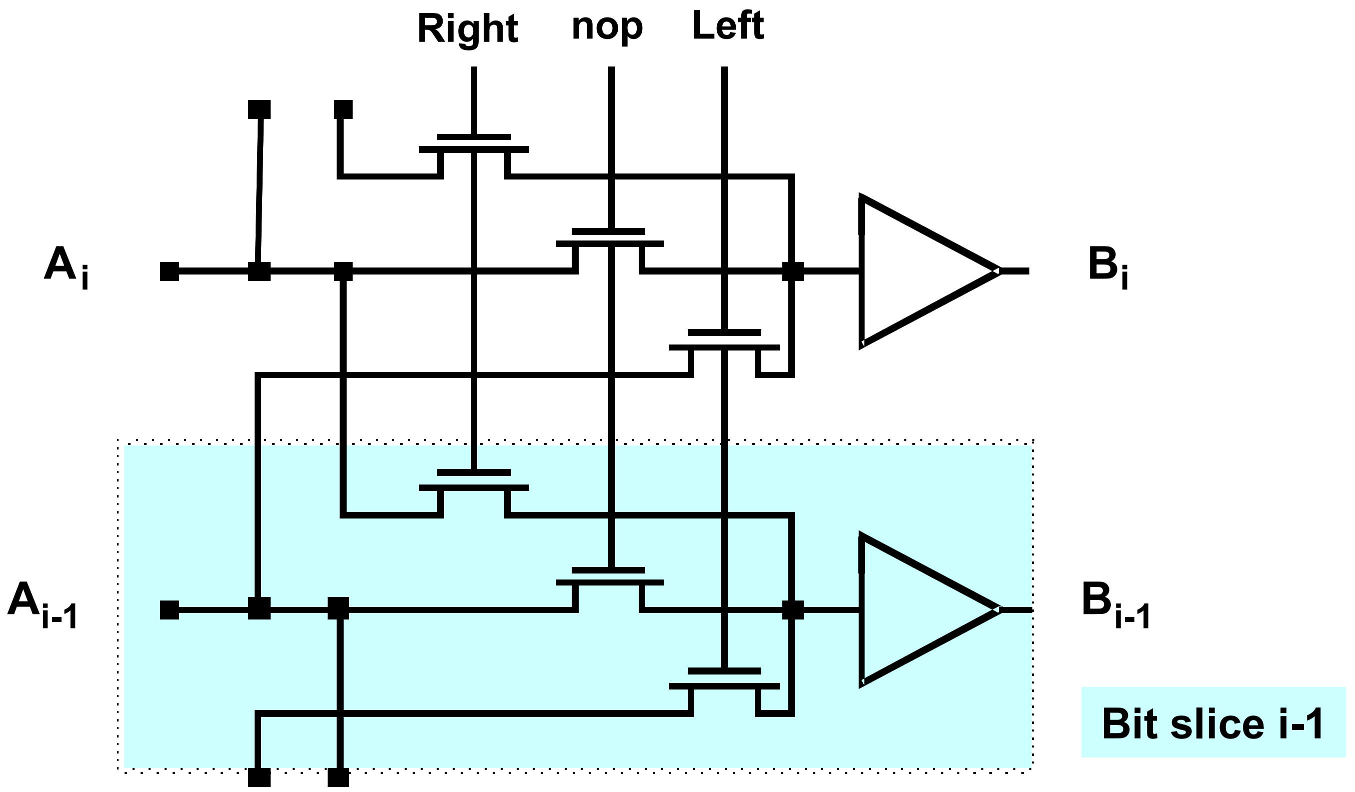
  - Logarithmic versus Linear (Wallace Tree Mult)
  - Data encoding (Booth)
  - Pipelining

GLIMPSE AT SYSTEM LEVEL OPTIMIZATION

Shifter Design

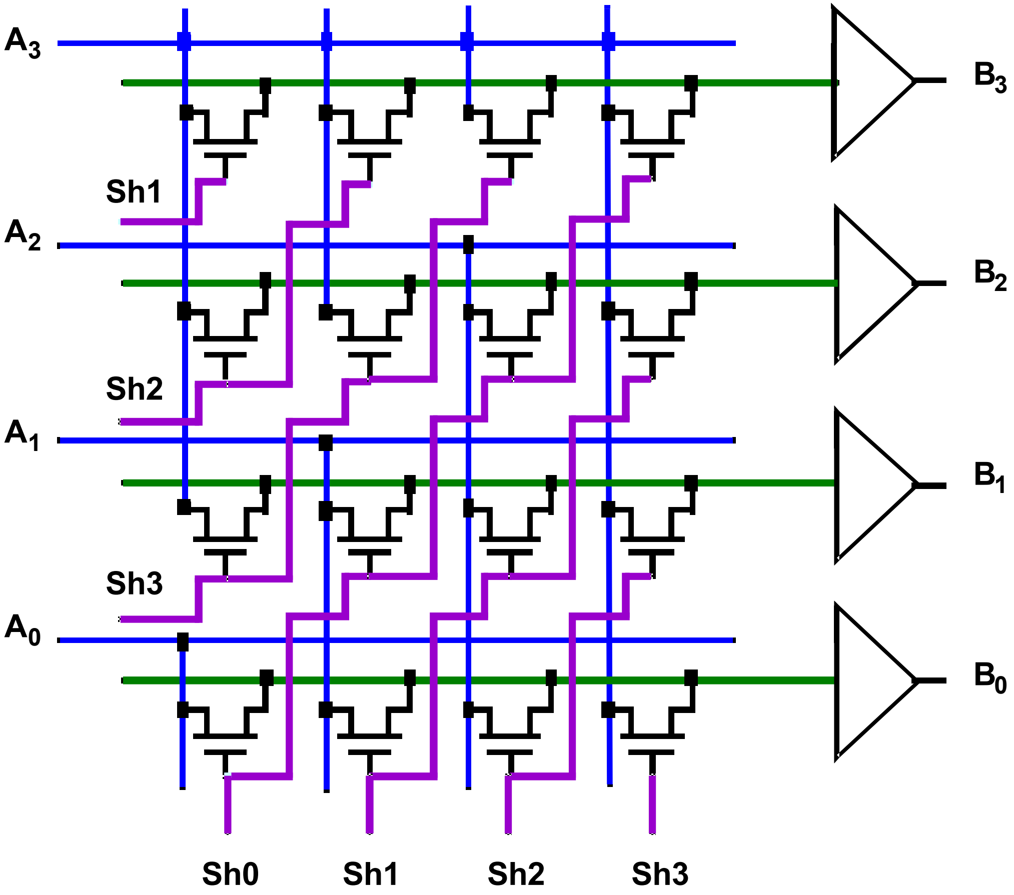
- Shifters are fundamental building blocks too
  - Floating point units
  - Scalers
  - Multiplication by constant numbers (add and shift)
  - …
- Constant shifting is only interconnect
- Programmable shifting requires active circuitry
- Usually dominated by interconnect
- Architectures
  - ✓ Barrel Shifter
  - ✓ Logarithmic Shifter
  - …
- Design trade-offs, optimization
  - Architecture level, Logic level, Circuit level, Layout level
  - Simpler compared to Adder, Multiplier, hence less rewarding
- Good example of pay-off of structural design

The Binary Shifter



- Multibit shifters by cascading
- M stages for M-bit shift
- Complex and slow for larger M
- More structured approach needed

The Barrel Shifter



Input data wire

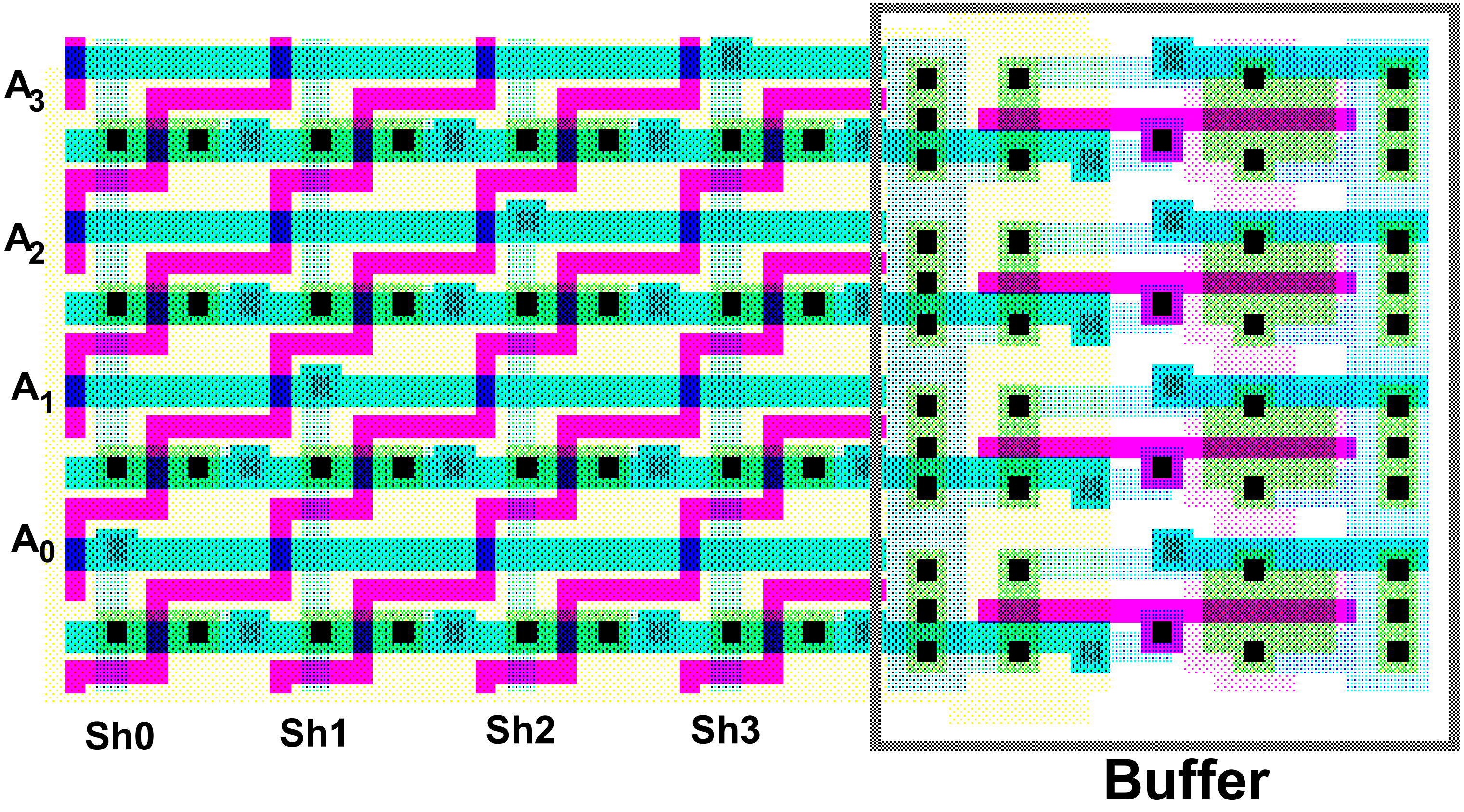
Output data wire

Control wire

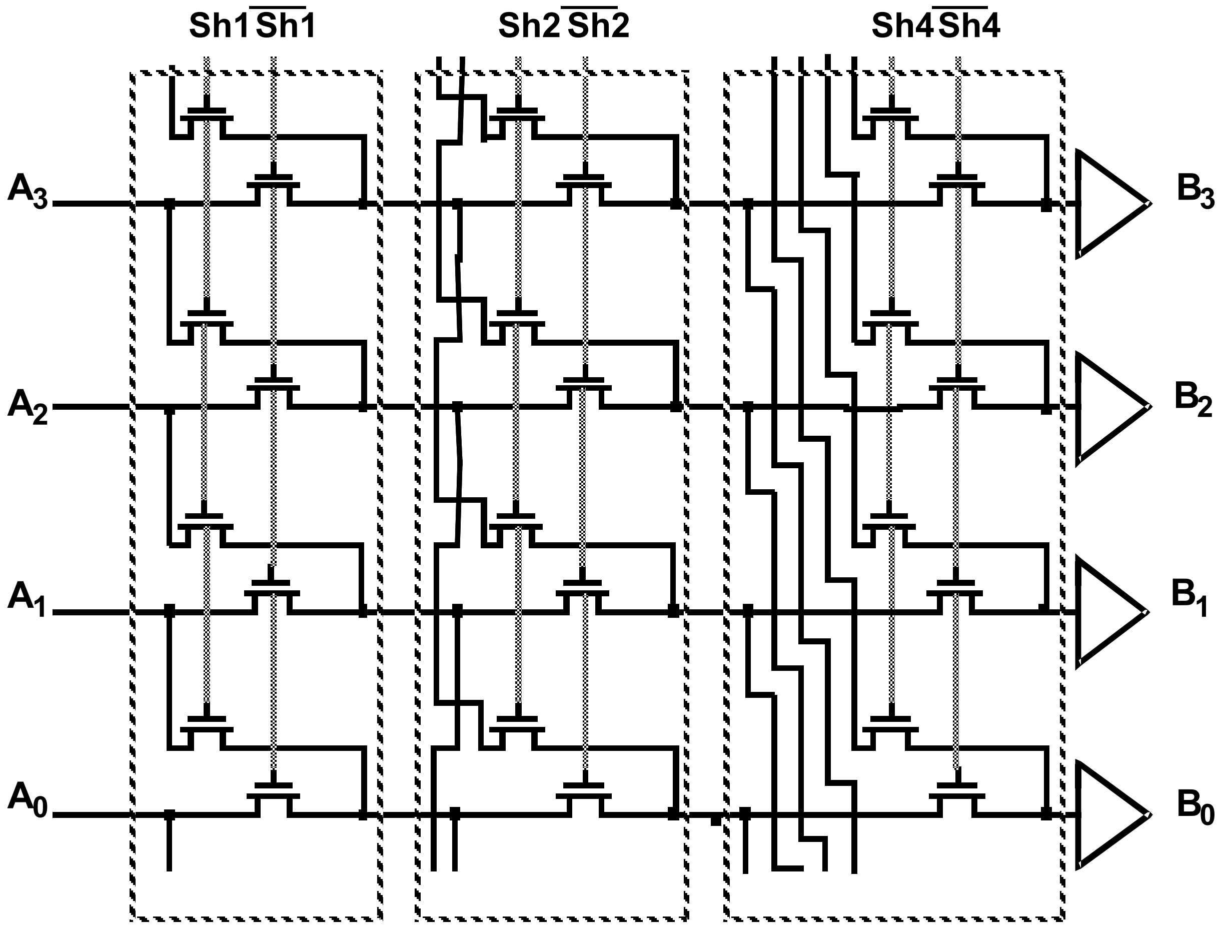
- Shift 1-N positions
- One control bit high

- Need M stages for M-bit shift
- Signal passes only one pass-transistor => delay?
- Area Dominated by Wiring, not (always) by # transistors

4x4 Barrel Shifter

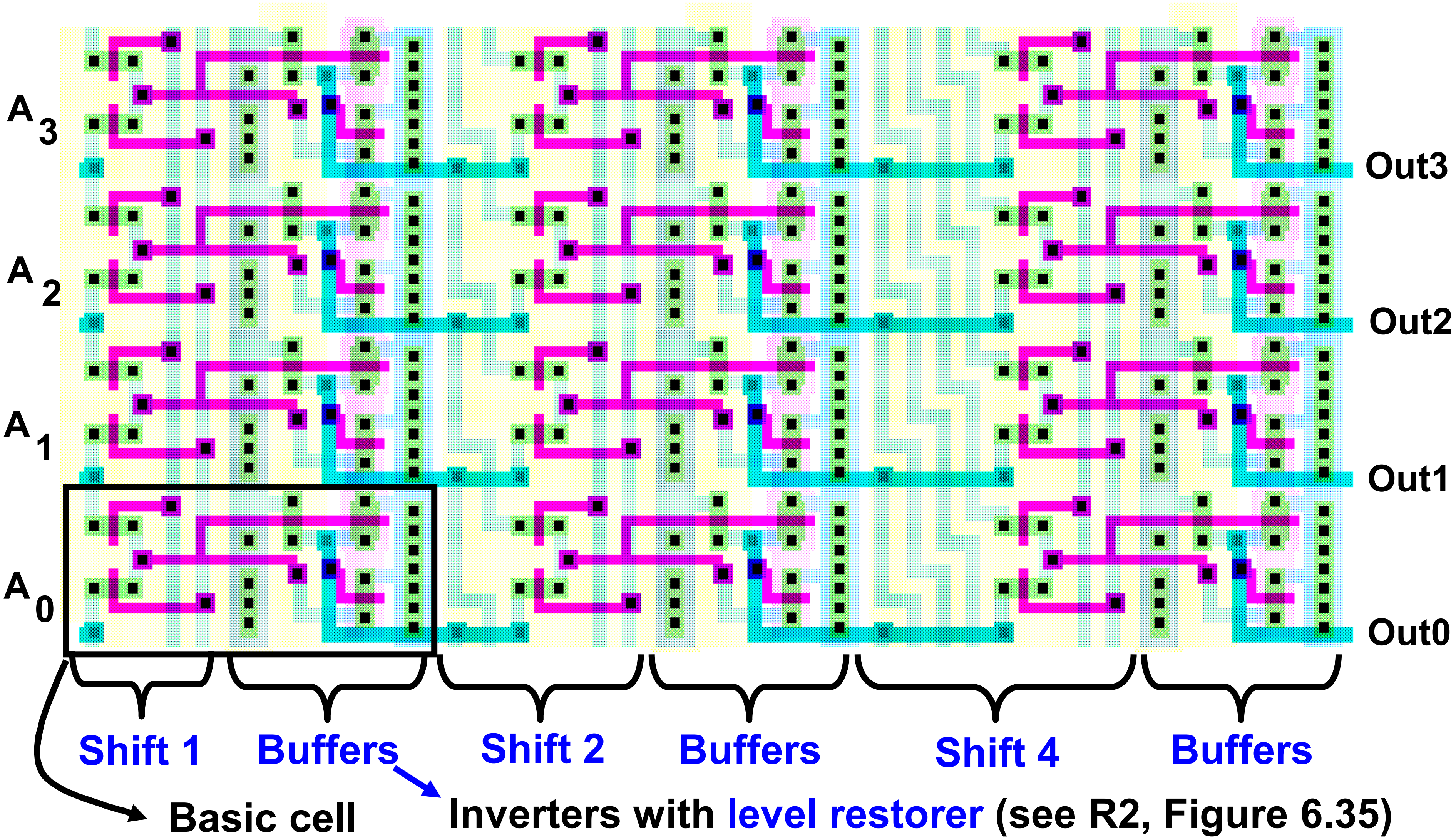


Logarithmic Shifter



- **Section i shifts $2^{(i-1)}$ bits**
- **Need only *log_2M stages* for M-bit shift**

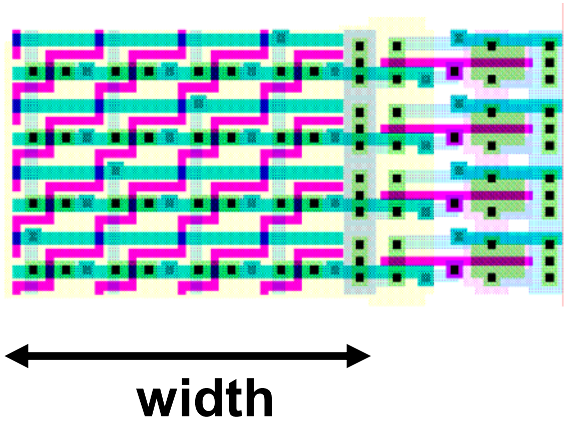
0-7 bit Logarithmic Shifter

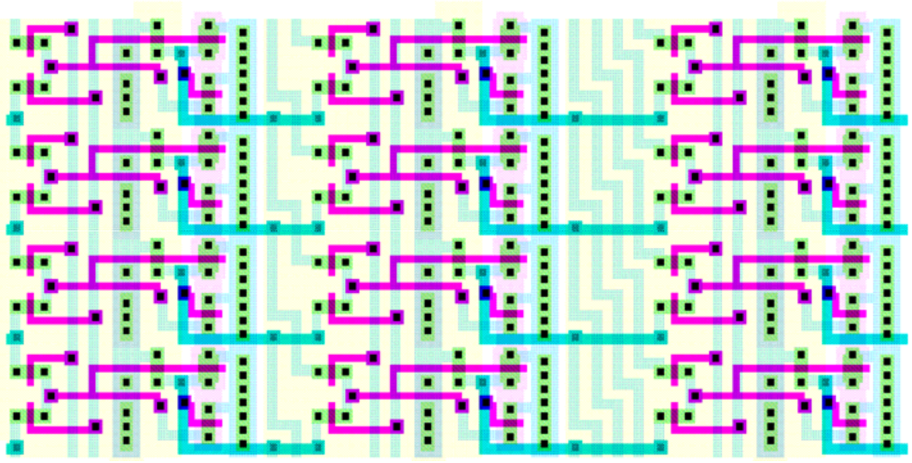


Inverters with **level restorer** (see R2, Figure 6.35)

Exercise: decipher layout of basic cell and draw transistor circuit

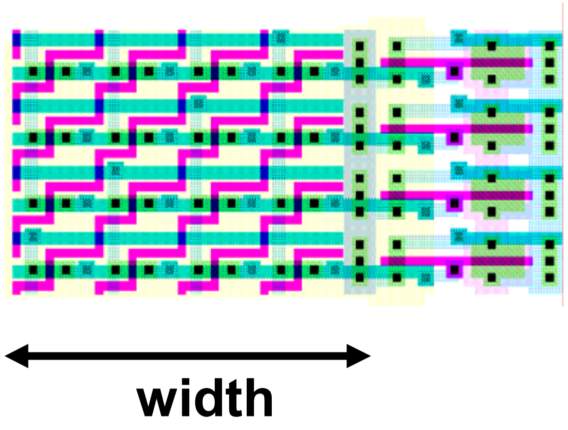
Size Comparison

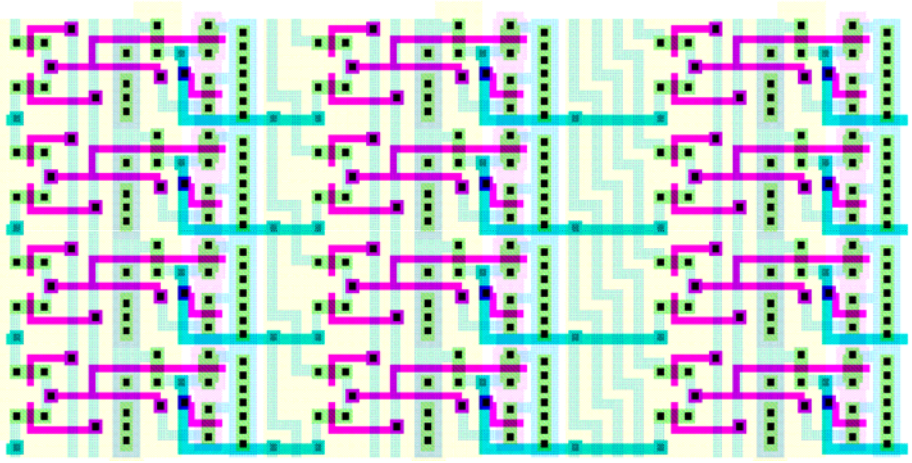




- *M* is maximum bit displacement, $K = \log_2 M$
- For large *M*, width is dominated by vertical metal wires
- Disregard buffer size, only count vertical wires
- Barrel shifter needs 1 control and 1 data wire per stage
- # Wires: *2M*
- Log shifter needs 2 control + 2^{i-1} data wires for stage *i*
- # Wires: $2K+(1+2+4+\ldots+2^{K-1}) = 2K+2^K-1 = 2\log_2 M + M - 1$
- Log shifter will have smaller area for larger *M*!

Speed Comparison.



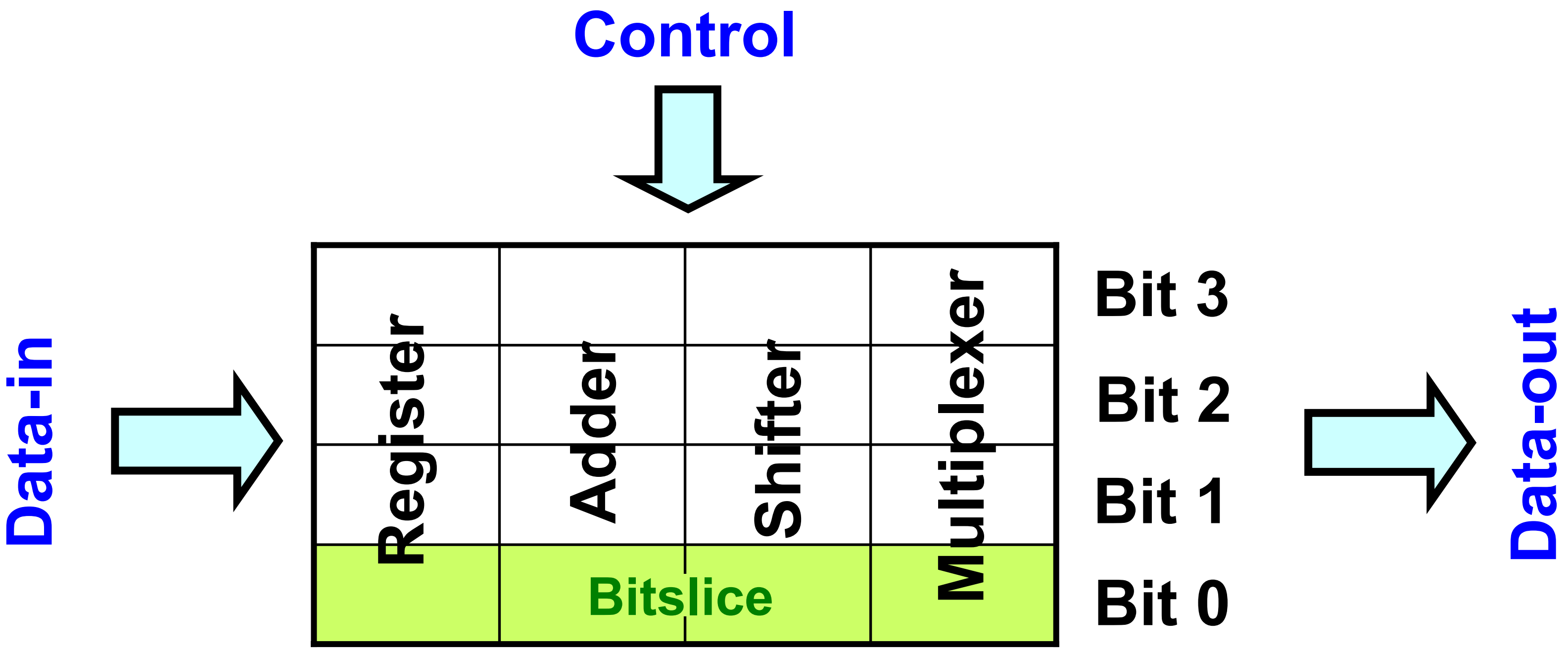


Exercise

- **Discuss the relative speeds of both shifters, as a function of *M* (see discussion in book). Consider:**
  - **Number of sections**
  - **Input capacitance at the buffers (including diffusion areas of the driving pass-transistors)**
  - **Number of buffers (necessity of buffers)**
  - **The number of pass-transistors the signal has to pass**
  - **…**

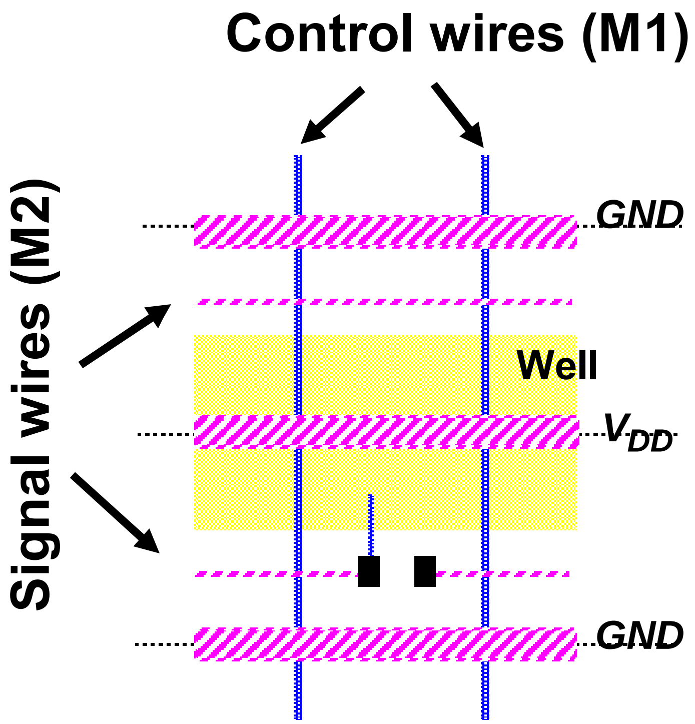
Layout Strategies (regularity)

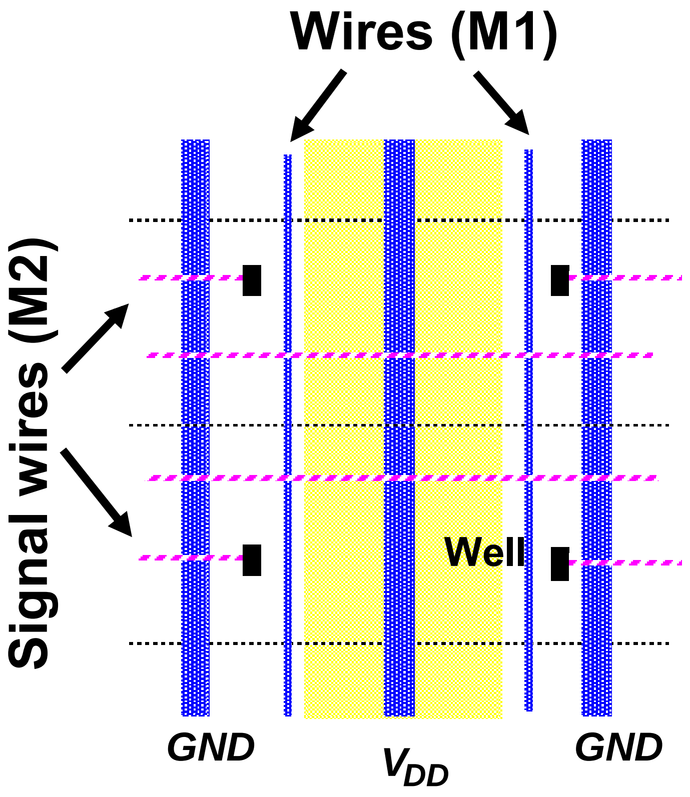
Bit-Sliced Design



- **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





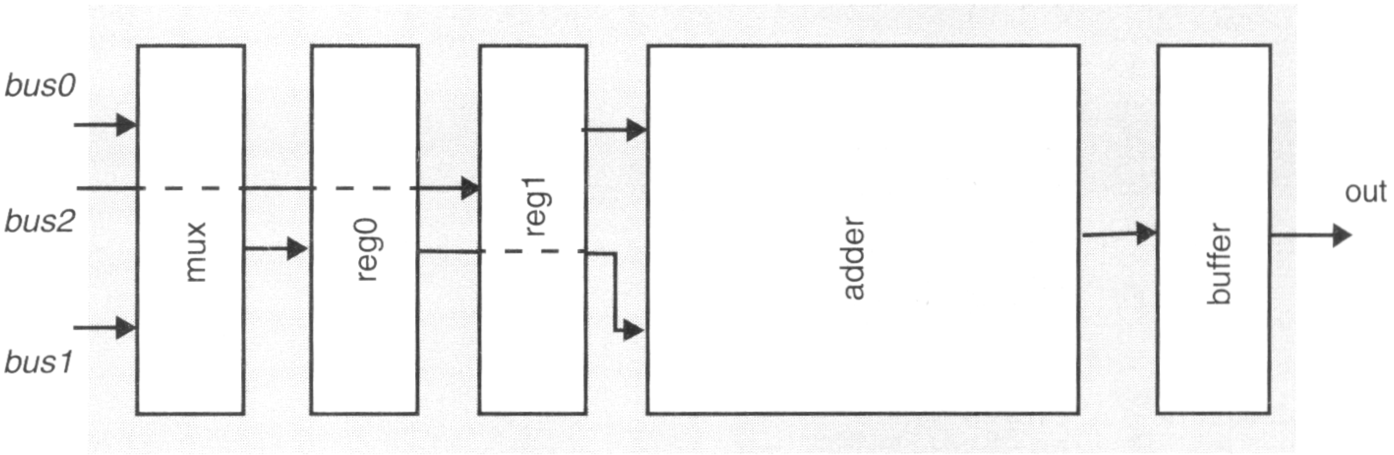
Approach I —

Signal and power lines parallel

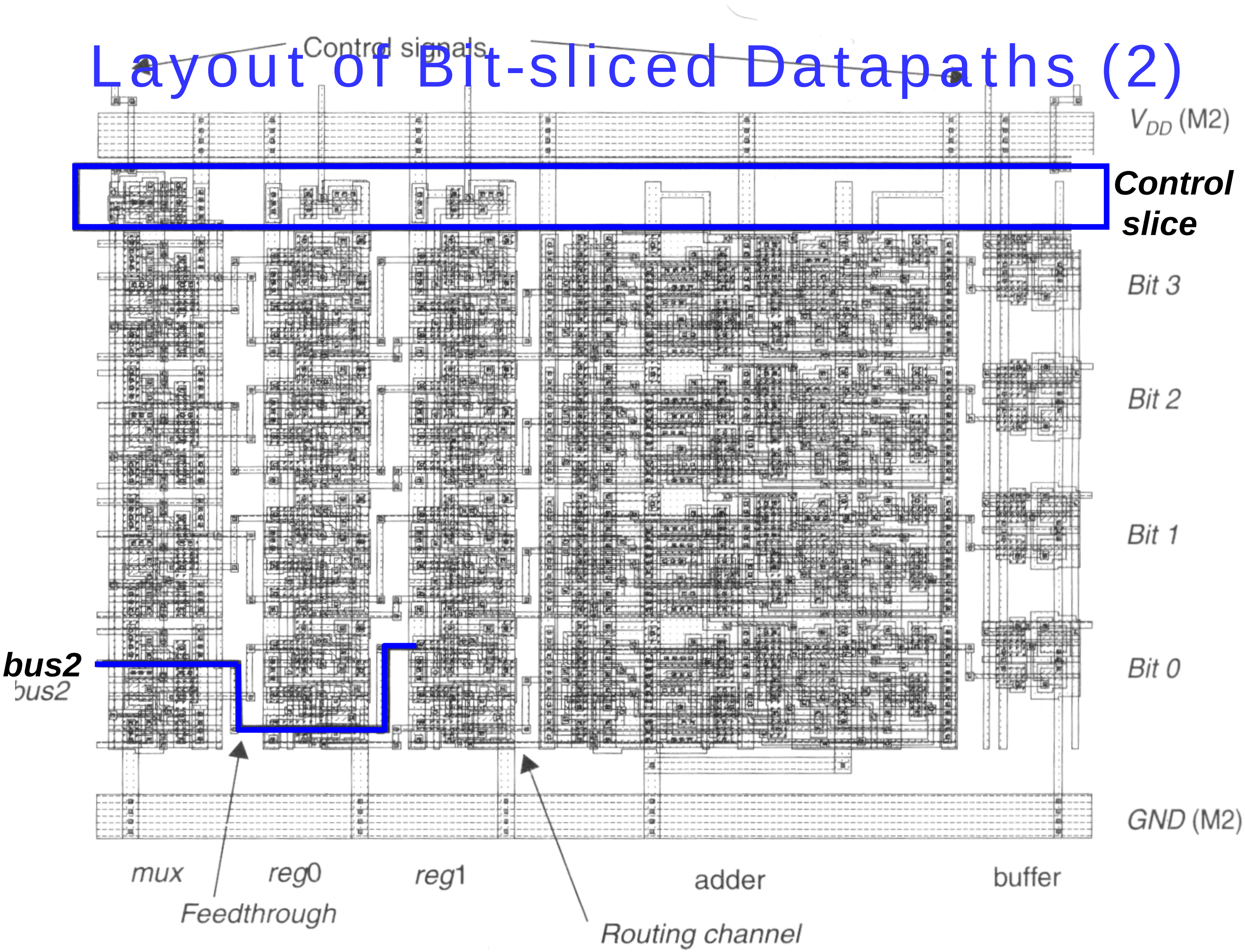
Approach II —

Signal and power lines perpendicular

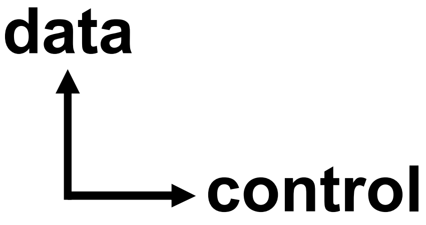
Layout of Bit-sliced Datapaths

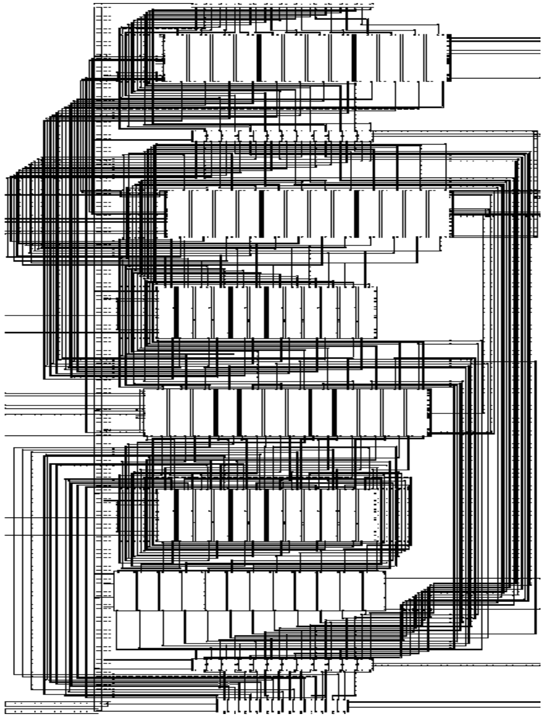


Layout of Bit-sliced Datapaths (2)



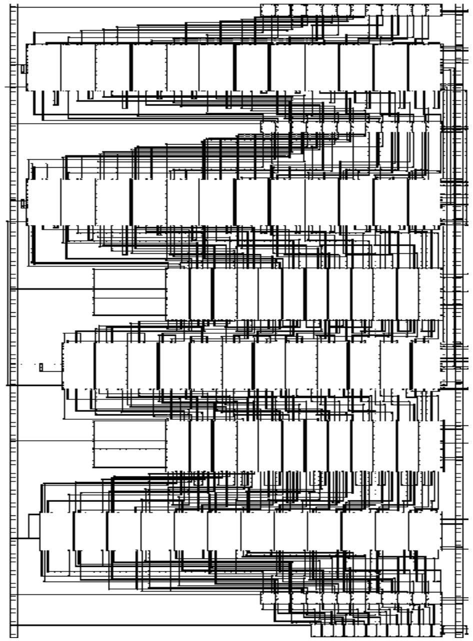
Layout of Bit-sliced Datapaths (3)





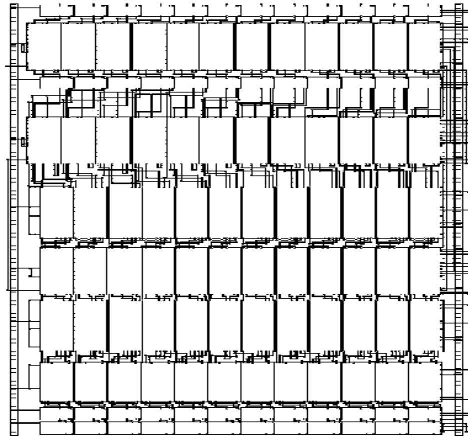
Unoptimized

Area: 4.2mm^2



With feedthroughs

Area: 3.2mm^2

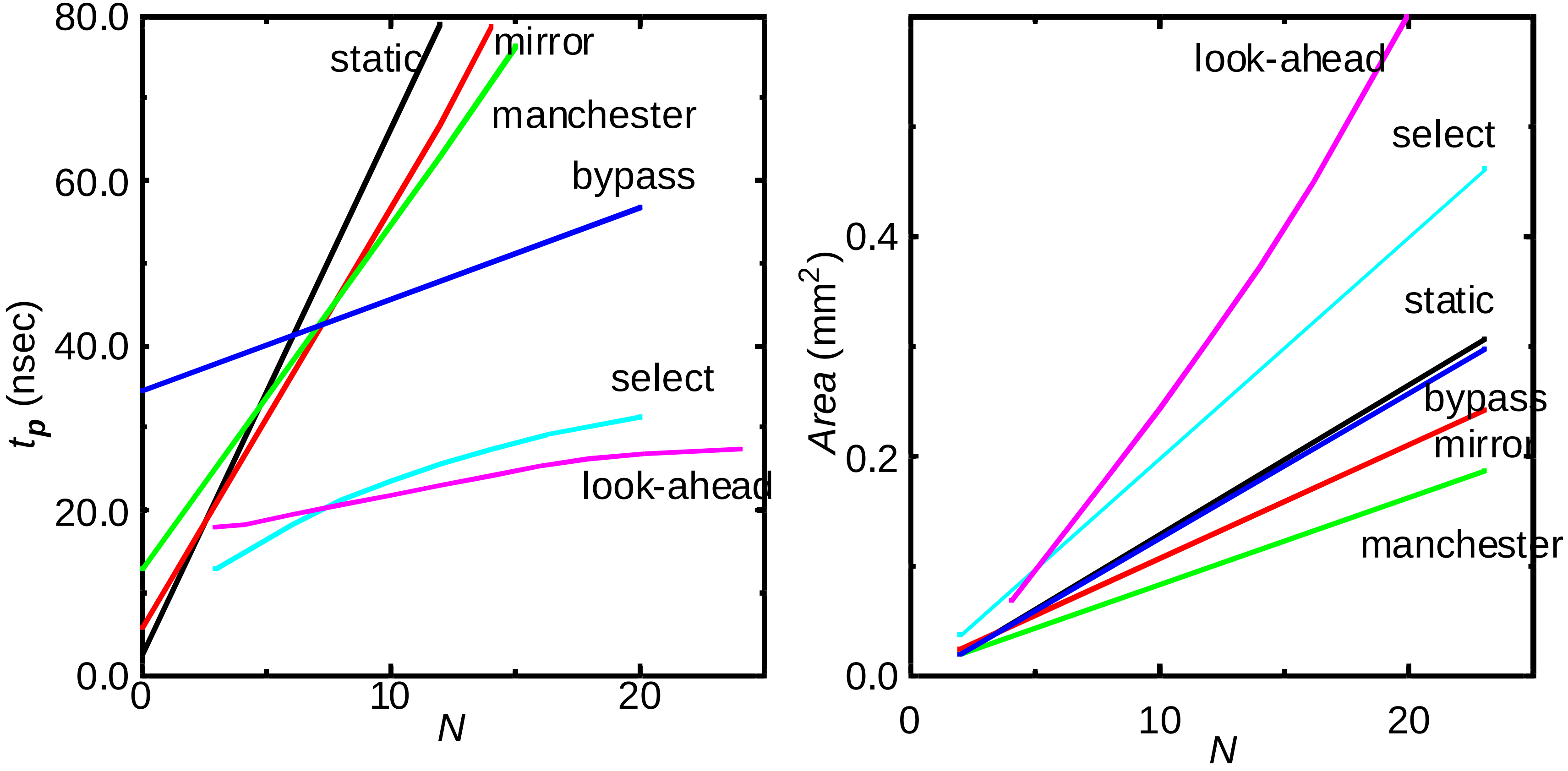


+ Equalized cell height

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?