MODULE 8

MODULARITY

Course Material for Modularity

Chapter 11 P = primair, I = Illustratie, O = overslaan

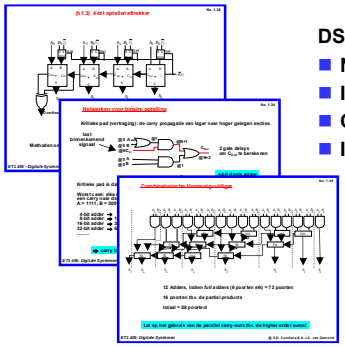
Outline

- 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



DS:

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

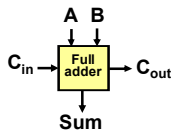
Adder Design

- 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

Brown §5.2-5.4

Full-Adder



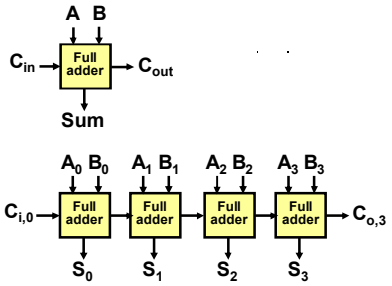
C_{in}	B	A	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Add three one-bit numbers

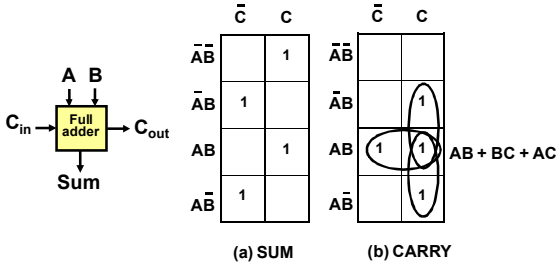
Equivalently: count # 1's in A, B, C_i

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

The Ripple-Carry Adder



The Binary Adder



$S = A \oplus B \oplus C_i$

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

$C_0 = AB + BC_i + AC_i$

AND-OR expressions for sum and carry

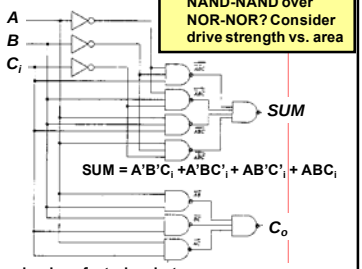
Naïve Complementary CMOS Implementation

- 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



Can do better using more clever boolean factoring, but…

Full-Adder Boolean Factoring

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

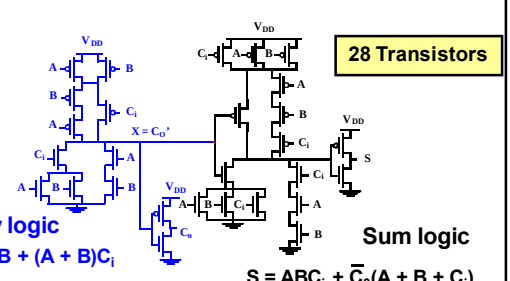
$= ABC_i + \bar{C}_0(A + B + C_i)$

$C_0 = AB + BC_i + AC_i$

$= AB + (A + B)C_i$

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

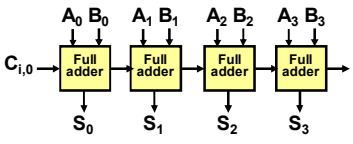
Carry logic

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

Sum logic

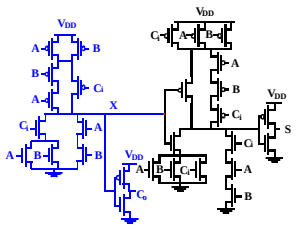
$S = ABC_i + \bar{C}_0(A + B + C_i)$

Ripple-Carry Adder Delay



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

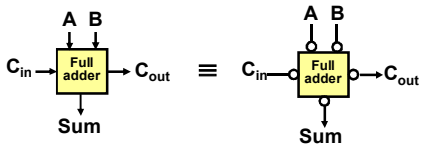
Adder Evaluation



Carry Chain:

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

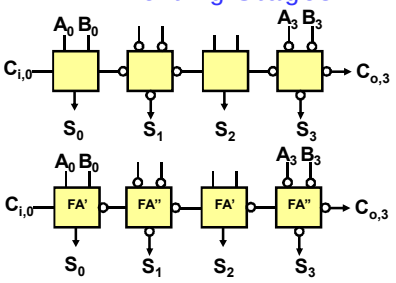
Inversion Property



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

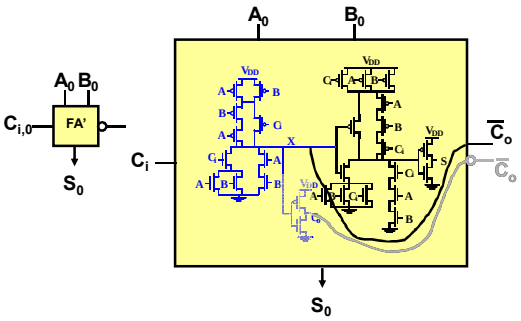
$$\overline{C_0}(A,B,C_i) = C_0(\overline{A},\overline{B},\overline{C_i})$$

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

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

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

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

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

ADD AND SHIFT

Example: 42 x 11 = 462

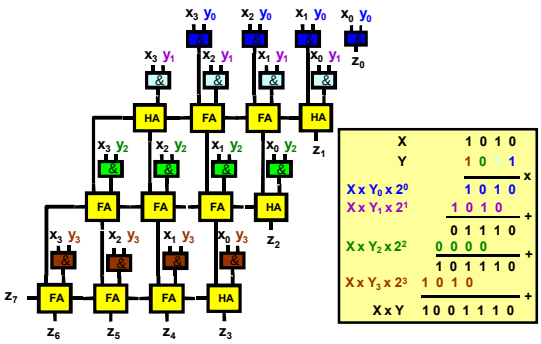
```
  1 0 1 0 1 0
    1 0 1 1  x
  -------------
  1 0 1 0 1 0
+ 1 0 1 0 1 0
  0 0 0 0 0 0
+ 1 0 1 0 1 0
  -------------
  1 1 1 0 0 1 1 1 0
```

Each partial product formed by bitwise AND operation

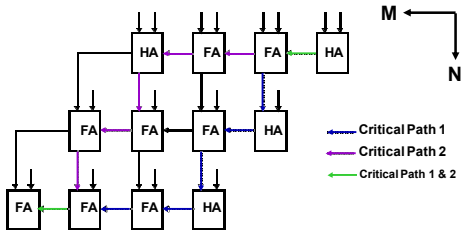
Partial products are shifted before being added

- **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



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

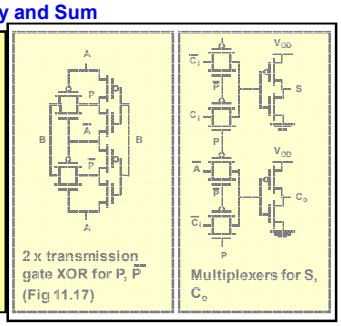
Identical Delays for Carry and Sum

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

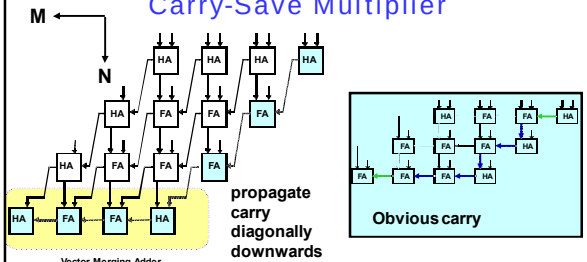
P = A ⊕ B

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

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

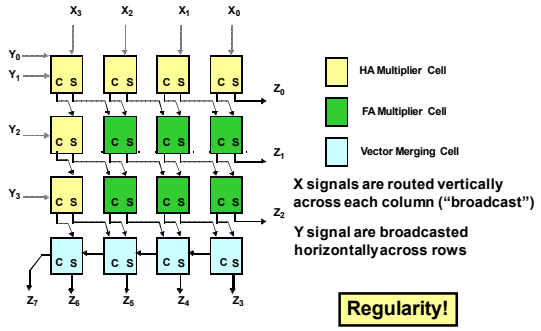


Carry-Save Multiplier



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



X signals are routed vertically across each column ("broadcast")

Y signal are broadcasted horizontally across rows

Regularity!

Multipliers — Summary.

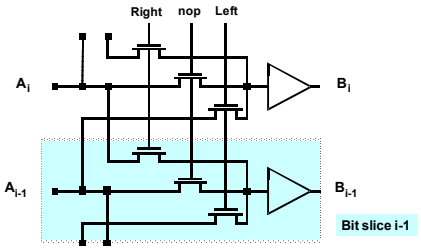
- Optimization Goals Different Vs Binary Adder
- Once Again: Identify Critical Path
- Other possible techniques
  - Logarithmic versus Linear (Wallace Tree Mult)
  - Data encoding (Booth)
  - Pipelining

GLIMPSE AT SYSTEM LEVEL OPTIMIZATION

Shifter Design

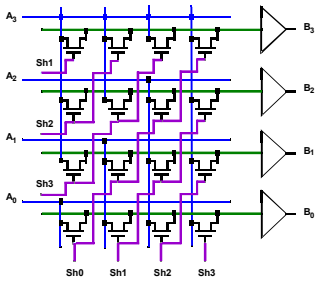
- Shifters are fundamental building blocks too
  - Floating point units
  - Scalers
  - Multiplication by constant numbers (add and shift)
  - …
- Constant shifting is only interconnect
- Programmable shifting requires active circuitry
- Usually dominated by interconnect
- Architectures
  - Barrel Shifter
  - 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

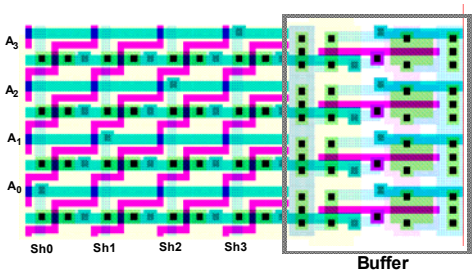


Input data wire
Output data wire
Control wire

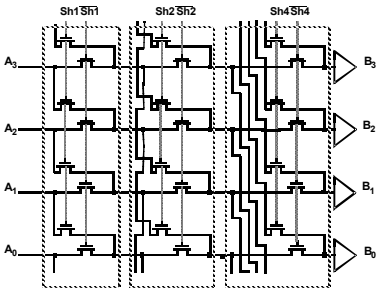
- Shift 1-N positions
- One control bit high

- 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

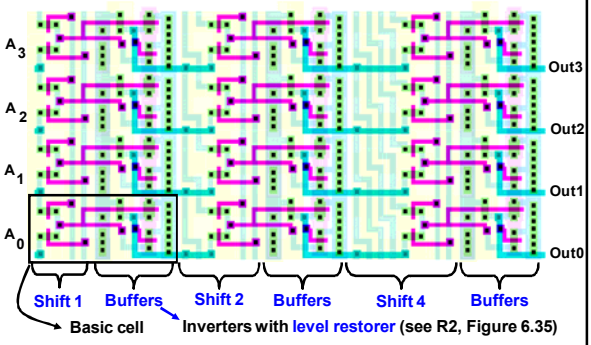


Logarithmic Shifter



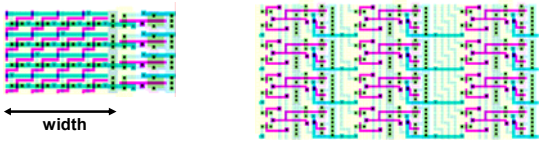
- Section i shifts $2^{(i-1)}$ bits
- Need only $log_2 M$ stages for M-bit shift

0-7 bit Logarithmic Shifter



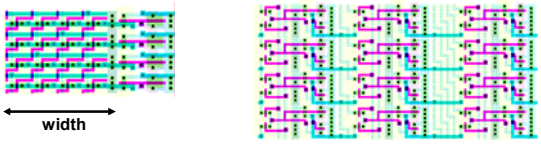
Exercise: decipher layout of basic cell and draw transistor circuit

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.

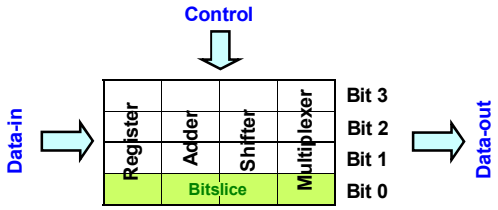


Exercise

- 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
  - …

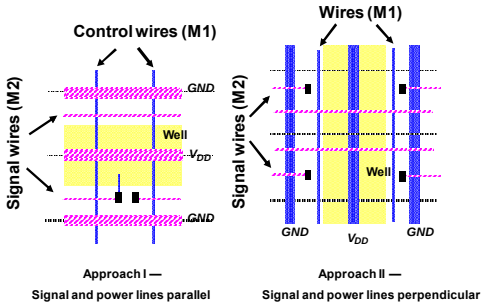
Layout Strategies (regularity)

Bit-Sliced Design



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

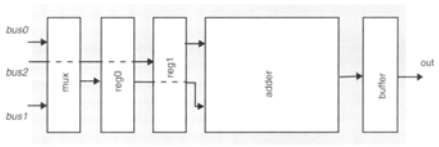
Layout Strategies for Bit-Sliced Datapaths



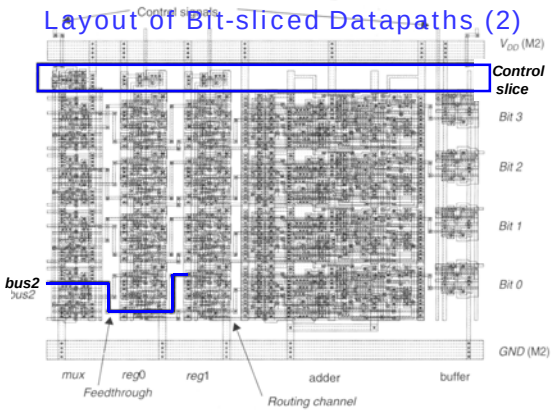
Approach I — Signal and power lines parallel

Approach II — Signal and power lines perpendicular

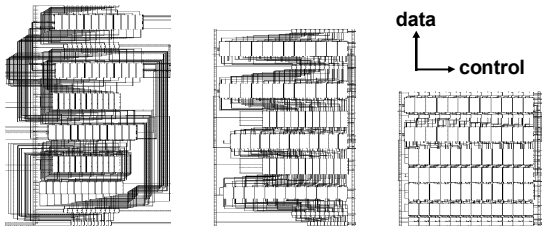
Layout of Bit-sliced Datapaths



Layout of Bit-sliced Datapaths (2)



Layout of Bit-sliced Datapaths (3)



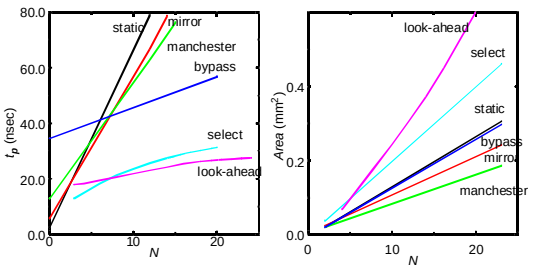
Unoptimized
Area: 4.2mm²

With feedthroughs
Area: 3.2mm²

+ Equalized cell height
Area: 2.2mm²

- 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?

The End