CPE 110408423
VLSI Design
Chapter 0: Fundamentals of VLSI Design

Bassam Jamil
[Computer Engineering Department,
Hashemite University]

Course Administration

- Instructor: Bassam Jamil
- Instructor's e-mail: bassam@hu.edu.jo
- Office Hours: Posted on my office door

- Textbook:

Required: *CMOS VLSI Design: A Circuits and Systems Perspective, 4th Edition, Neil H. E. Weste, and David Money Harris, Pearson Publication, Inc., 2011.*

Optional:Digital Integrated Circuits: A Design Perspective, Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic, Prentice-Hall Publication, Inc., 2002.

Sedra & Smith, Microelectronic Circuits Oxford University Press, 5th Edition.

Slides : pdf on the course web page (Moodle System)

Course Content

- Course goals
  - The ability to analyze and design of digital integrated circuits families.
  - The ability to model datapath subsystems.
  - Acquire a basic knowledge to analyze design and circuit optimization of digital building blocks with respect to different quality metrics: cost, speed, power dissipation, and reliability.
  - The ability to learn fabrication and layout process, invertors design, clocks and power distribution and memory design.
  - The ability to work in groups to do design problems in the assignments and the projects.

- Course prerequisites

  Digital Integrated Circuits (0408422) Digital Electronics and Integrated Circuits (110408327).

Course Structure

- Design focused class:
  - Various homework assignments throughout the semester
- Lectures:

Ch0. Introduction

Ch11. Datapath Subsystems

Ch12. Array Subsystems

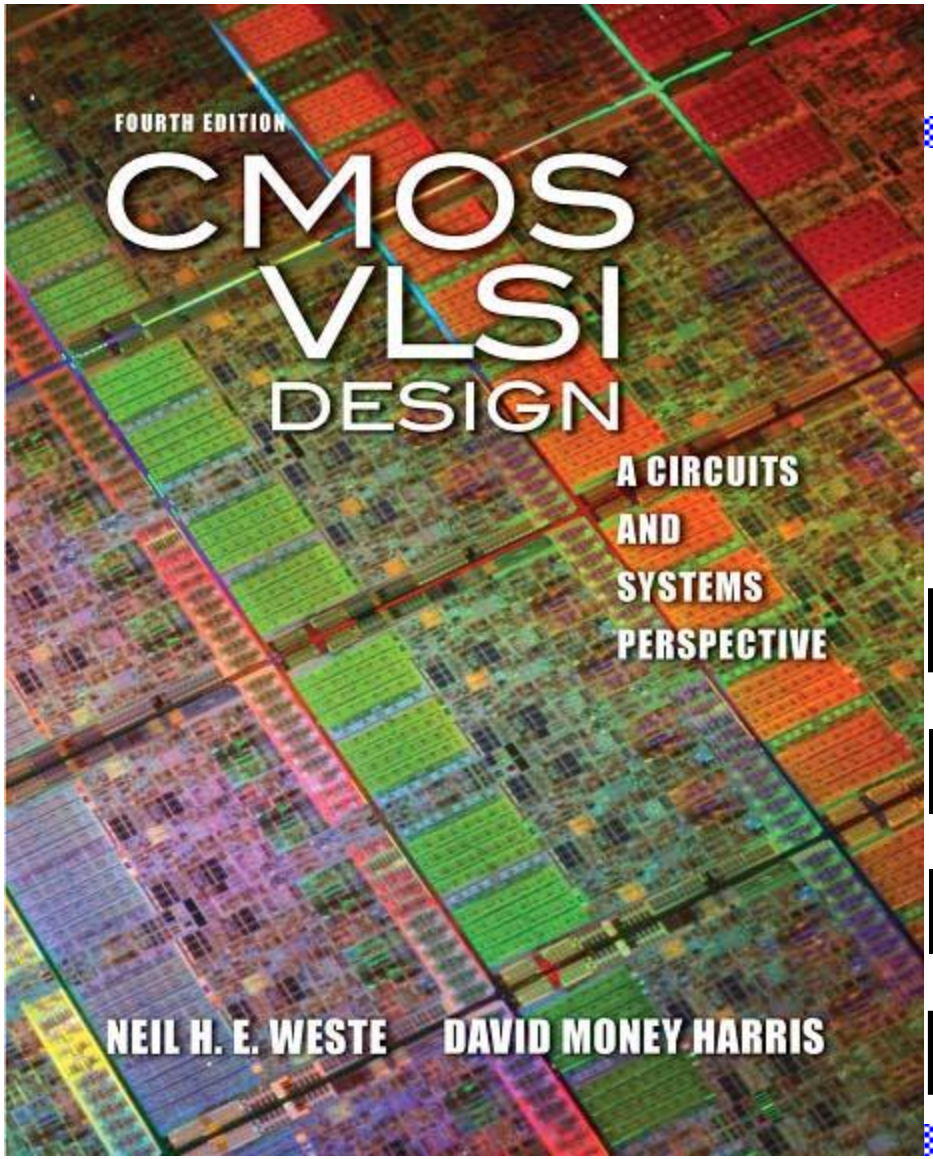
Ch13. Special-Purpose Subsystems **(optional)**

Ch14. Design Methodology Tools

Ch15. Testing, Debugging, and Verification

Grading Information

- Grade determinates
  - Midterm Exam ~30
  - Project ~20%
  - Final Exam ~50%
- Let me know about any exam conflicts ASAP



Lecture 0: Introduction: Intel Processors

Introduction

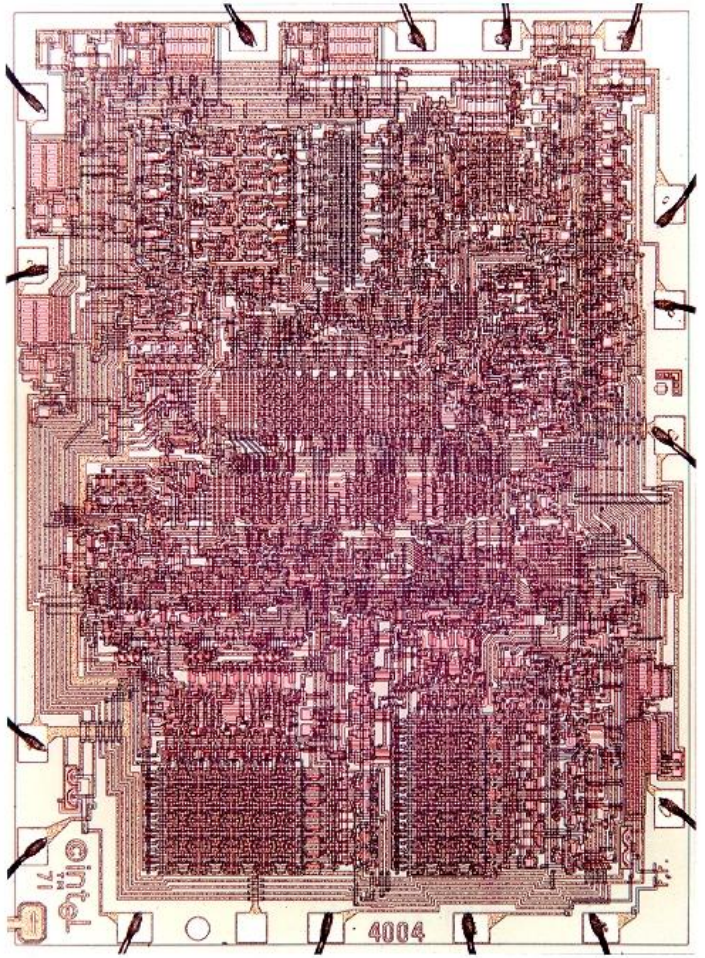
- Integrated circuits: many transistors on one chip.
- *Very Large Scale Integration* (VLSI): bucketloads!
- *Complementary Metal Oxide Semiconductor*
  - Fast, cheap, low power transistors
- Today: How to build your own simple CMOS chip
  - CMOS transistors
  - Building logic gates from transistors
  - Transistor layout and fabrication
- Rest of the course: How to build a good CMOS chip

Outline

- Evolution of Intel Microprocessors
  - Scaling from 4004 to Core i7
  - Courtesy of Intel Museum

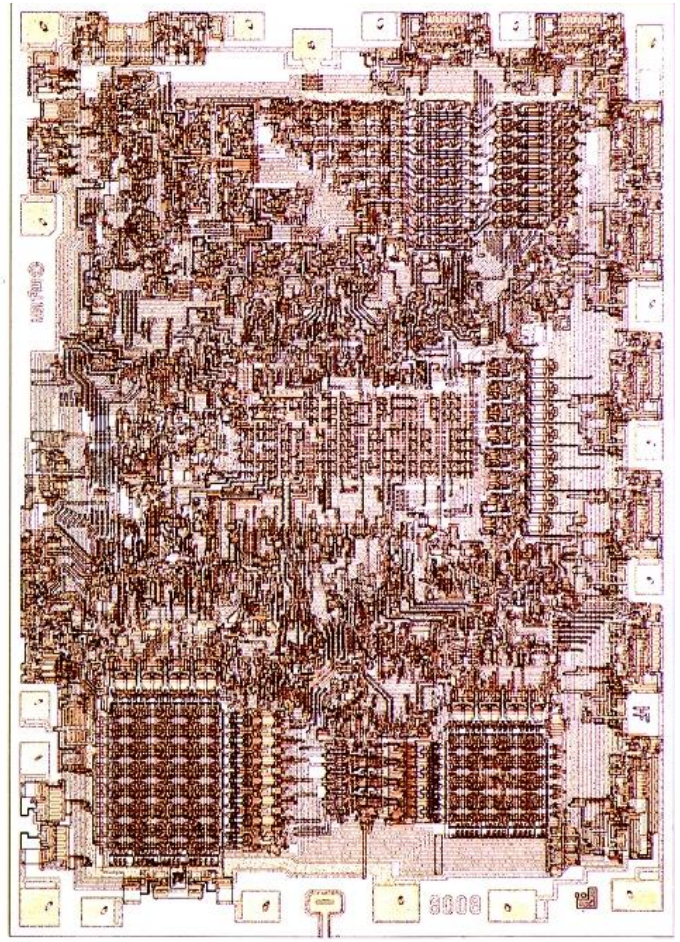
4004

- **First microprocessor** (1971)
  - For Busicom calculator
- Characteristics
  - 10 μm process
  - 2300 transistors
  - 400 – 800 kHz
  - 4-bit word size
  - 16-pin DIP package
- Masks hand cut from Rubylith
  - Drawn with color pencils
  - 1 metal, 1 poly (jumpers)
  - Diagonal lines (!)



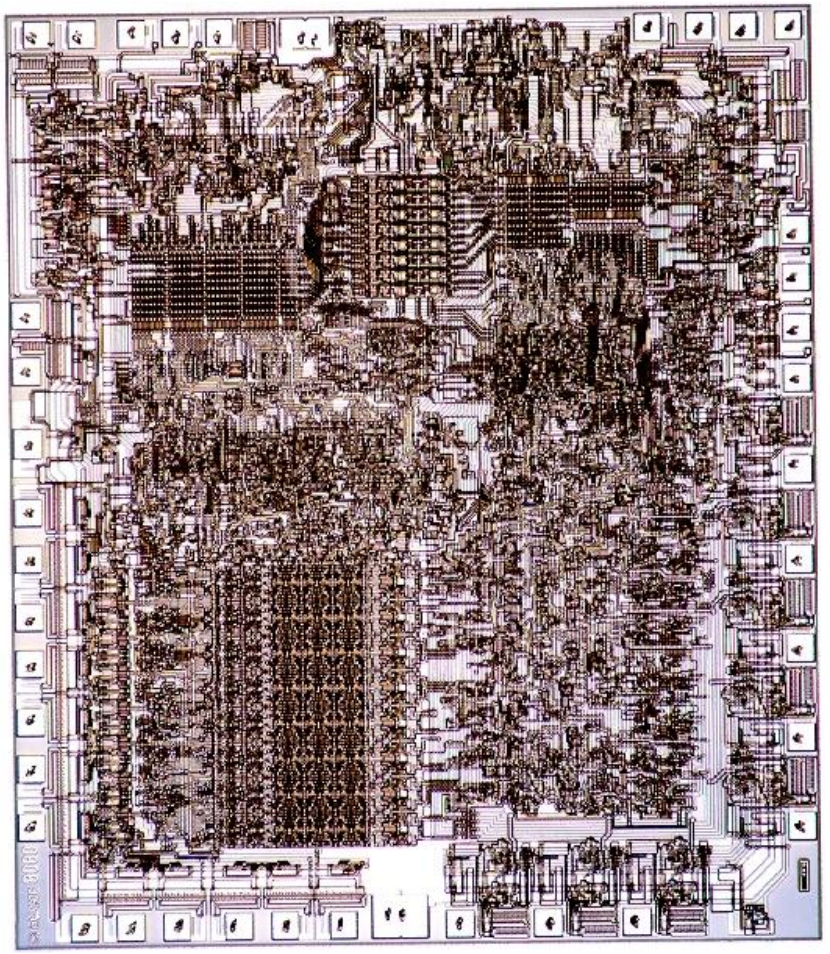
8008

- **8-bit follow-on** (1972)
  - Dumb terminals
- Characteristics
  - 10 μm process
  - 3500 transistors
  - 500 – 800 kHz
  - 8-bit word size
  - 18-pin DIP package
- Note 8-bit datapaths
  - Individual transistors visible



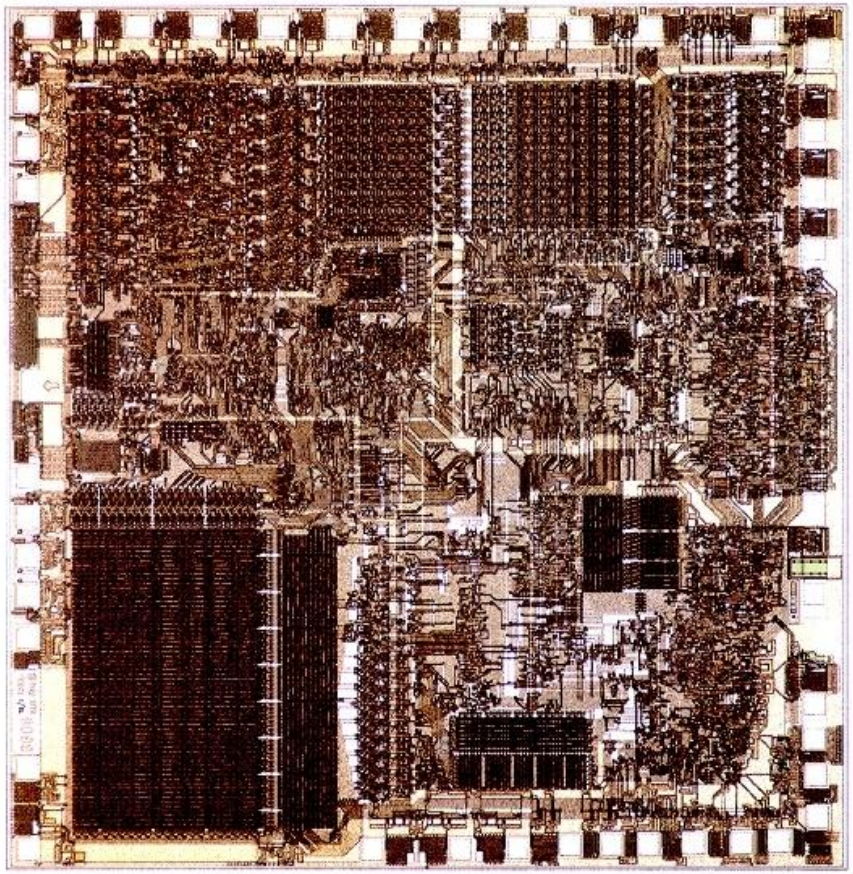
8080

- **16-bit address bus** (1974)
  - Used in Altair computer
    - (early hobbyist PC)
- Characteristics
  - 6 μm process
  - 4500 transistors
  - 2 MHz
  - 8-bit word size
  - 40-pin DIP package



8086 / 8088

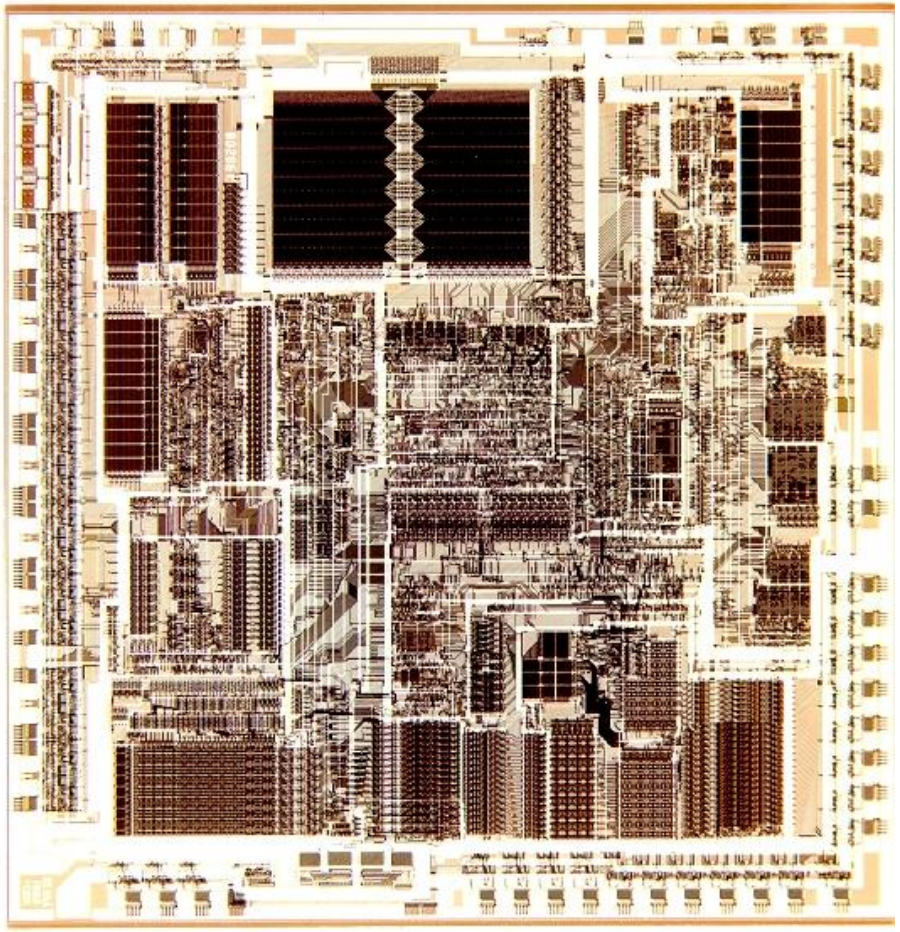
- **16-bit processor** (1978-9)
  - IBM PC and PC XT
  - Revolutionary products
  - Introduced x86 ISA
- Characteristics
  - 3 μm process
  - 29k transistors
  - 5-10 MHz
  - 16-bit word size
  - 40-pin DIP package
- Microcode ROM



80286

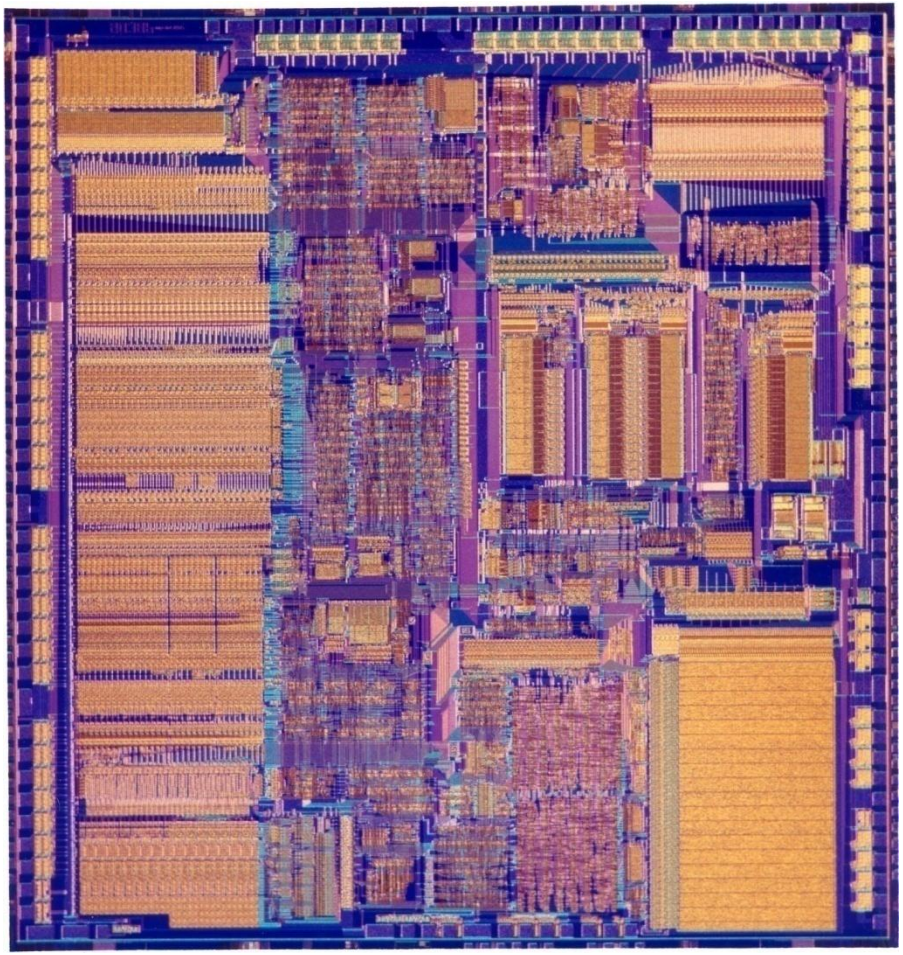
- **Virtual memory** (1982)
  - IBM PC AT
- Characteristics
  - 1.5 μm process
  - 134k transistors
  - 6-12 MHz
  - 16-bit word size
  - 68-pin PGA
- Regular datapaths and ROMs

  Bitslices clearly visible



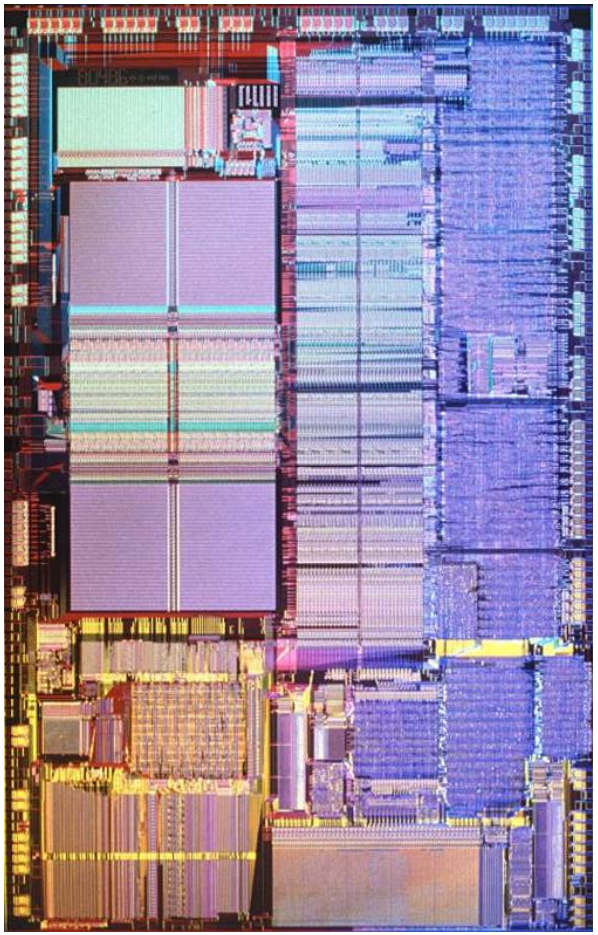
80386

- **32-bit processor (1985)**
  - **Modern x86 ISA**
- Characteristics
  - 1.5-1 μm process
  - 275k transistors
  - 16-33 MHz
  - 32-bit word size
  - 100-pin PGA
- 32-bit datapath, microcode ROM, synthesized control



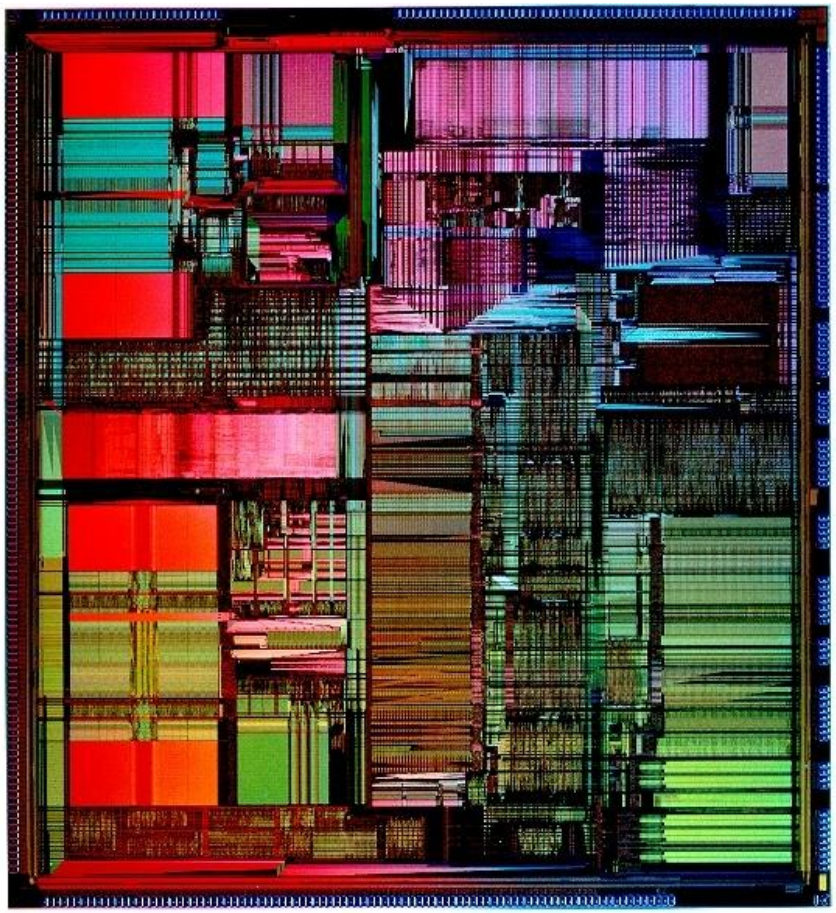
80486

- **Pipelining (1989)**
  - **Floating point unit**
  - 8 KB cache
- Characteristics
  - 1-0.6 μm process
  - 1.2M transistors
  - 25-100 MHz
  - 32-bit word size
  - 168-pin PGA
- Cache, Integer datapath, FPU, microcode, synthesized control



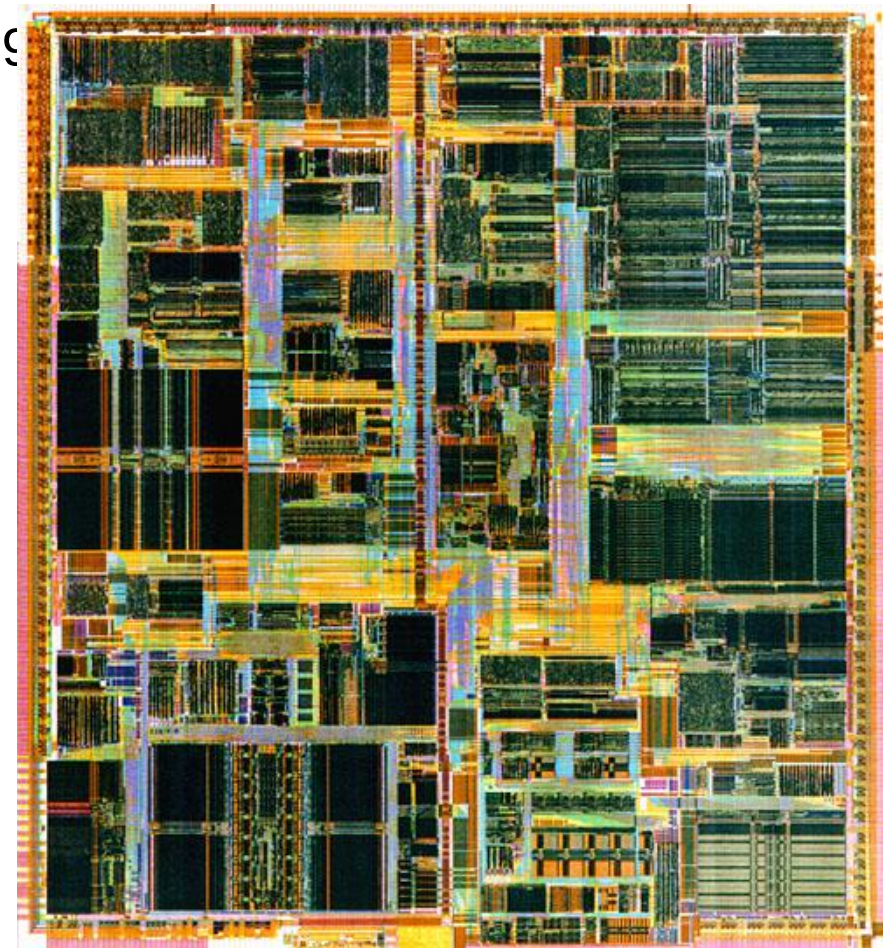
Pentium

- **Superscalar** (1993)
  - 2 instructions per cycle
  - Separate 8KB I$ & D$
- Characteristics
  - 0.8-0.35 μm process
  - 3.2M transistors
  - 60-300 MHz
  - 32-bit word size
  - 296-pin PGA
- Caches, datapath, FPU, control



Pentium Pro / II / III

- **Dynamic execution** (1995-9
  - 3 micro-ops / cycle
  - Out of order execution
  - 16-32 KB I$ & D$
  - Multimedia instructions
  - PIII adds 256+ KB L2$
- Characteristics
  - 0.6-0.18 μm process
  - 5.5M-28M transistors
  - 166-1000 MHz
  - 32-bit word size
  - MCM / SECC

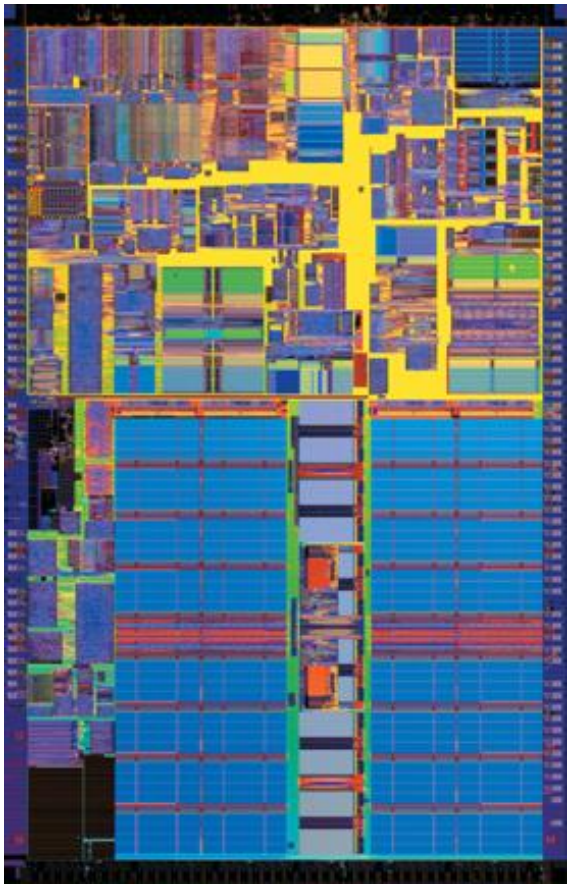


Pentium 4

- **Deep pipeline** (2001)
  - Very fast clock
  - 256-1024 KB L2$
- Characteristics
  - 180 – 65 nm process
  - 42-125M transistors
  - 1.4-3.4 GHz
  - Up to 160 W
  - 32/64-bit word size
  - 478-pin PGA
- Units start to become invisible on this scale

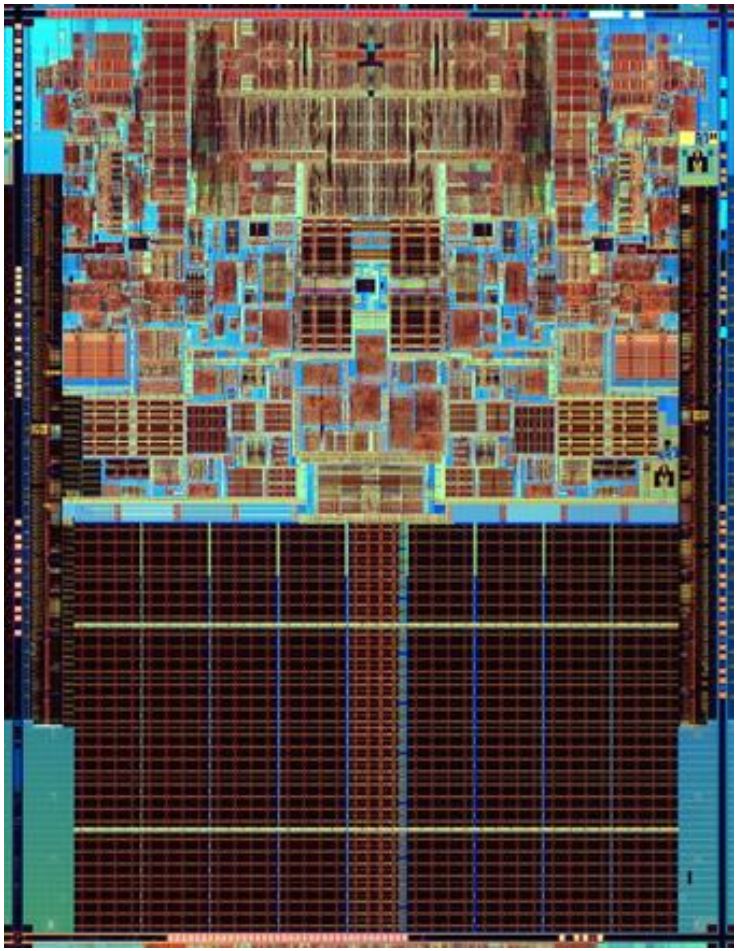
Pentium M

- Pentium III derivative
  - **Better power efficiency**
  - 1-2 MB L2$
- Characteristics
  - 130 – 90 nm process
  - 140M transistors
  - 0.9-2.3 GHz
  - 6-25 W
  - 32-bit word size
  - 478-pin PGA
- **Cache dominates chip area**



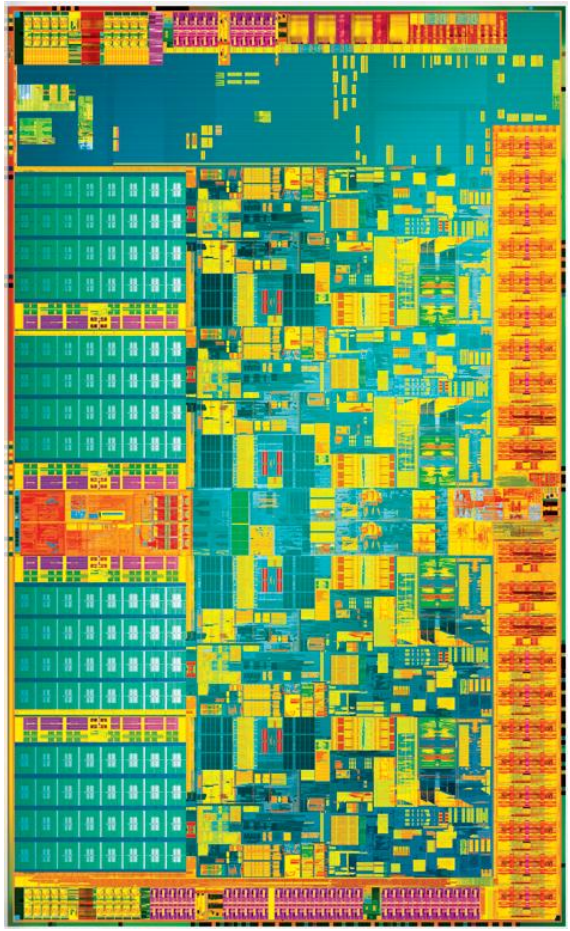
Core2 Duo

- **Dual core** (2006)
  - **1-2 MB L2$ / core**
- Characteristics
  - 65-45 nm process
  - 291M transistors
  - 1.6-3+ GHz
  - 65 W
  - 32/64 bit word size
  - 775 pin LGA
- Much better performance/power efficiency



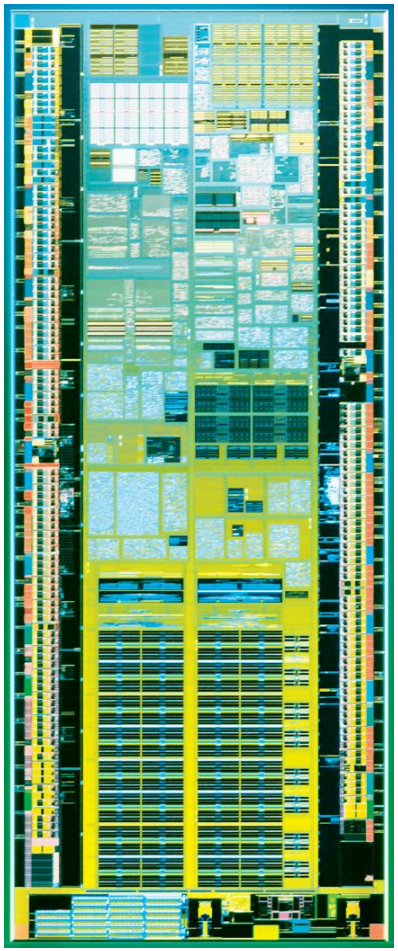
Core i7

- **Quad core (& more)**
  - Refinement of Core architecture
  - 2 MB L3$ / core
- Characteristics
  - 45-32 nm process
  - 731M transistors
  - 2.66-3.33+ GHz
  - Up to 130 W
  - 32/64 bit word size
  - 1366-pin LGA
  - Multithreading
- On-die memory controller



Atom

- **Low power CPU for netbooks**
  - Pentium-style architecture
  - 512KB+ L2$
- Characteristics
  - 45-32 nm process
  - 47M transistors
  - 0.8-1.8+ GHz
  - 1.4-13 W
  - 32/64-bit word size
  - 441-pin FCBGA
- Low voltage (0.7 – 1.1 V) operation
  - Excellent performance/power



Summary

- 10^4 increase in transistor count, clock frequency over 3 decades!

Processor	Year	Feature Size (μm)	Transistors	Frequency (MHz)	Word Size	Power (W)	Cache (L1 / L2 / L3)	Package
4004	1971	10	2.3k	0.75	4	0.5	none	16-pin DIP
8008	1972	10	3.5k	0.5–0.8	8	0.5	none	18-pin DIP
8080	1974	6	6k	2	8	0.5	none	40-pin DIP
8086	1978	3	29k	5–10	16	2	none	40-pin DIP
80286	1982	1.5	134k	6–12	16	3	none	68-pin PGA
Intel386	1985	1.5–1.0	275k	16–25	32	1–1.5	none	100-pin PGA
Intel486	1989	1–0.6	1.2M	25–100	32	0.3–2.5	8K	168-pin PGA
Pentium	1993	0.8–0.35	3.2–4.5M	60–300	32	8–17	16K	296-pin PGA
Pentium Pro	1995	0.6–0.35	5.5M	166–200	32	29–47	16K / 256K+	387-pin MCM PGA
Pentium II	1997	0.35–0.25	7.5M	233–450	32	17–43	32K / 256K+	242-pin SECC
Pentium III	1999	0.25–0.18	9.5–28M	450–1000	32	14–44	32K / 512K	330-pin SECC2
Pentium 4	2000	180–65 nm	42–178M	1400–3800	32/64	21–115	20K+ / 256K+	478-pin PGA
Pentium M	2003	130–90 nm	77–140M	1300–2130	32	5–27	64K / 1M	479-pin FCBGA
Core	2006	65 nm	152M	1000–1860	32	6–31	64K / 2M	479-pin FCBGA
Core 2 Duo	2006	65–45 nm	167–410M	1060–3160	32/64	10–65	64K / 4M+	775-pin LGA
Core i7	2008	45 nm	731M	2660–3330	32/64	45–130	64K / 256K / 8M	1366-pin LGA
Atom	2008	45 nm	47M	800–1860	32/64	1.4–13	56K / 512K+	441-pin FCBGA

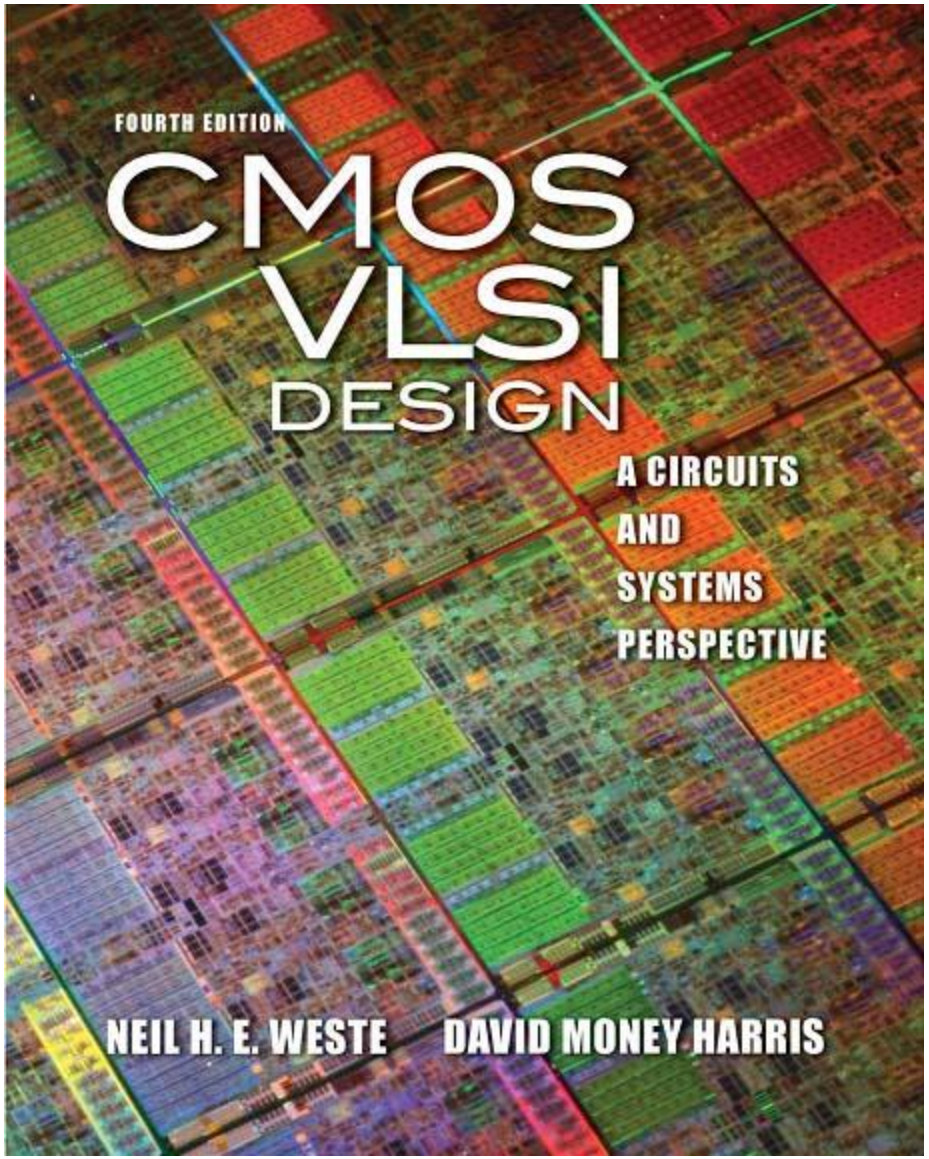
CPE 110408423
VLSI Design
Chapter 11: Datapath Subsystems

Bassam Jamil

[Computer Engineering Department,
Hashemite University]

Chip Functions:

- Datapath Operators
- Memory Elements
- Control Structure
- Special-Purpose Cells:
  - I/O
  - Power Distribution
  - Clock Generation and Distribution
  - Analog and RF



Lecture 1: Adders / Subtractors

Outline

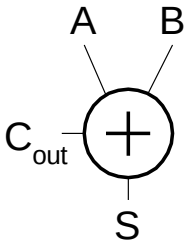
- Single-bit Addition
- Carry-Ripple Adder
- Propagate and Generate (P/G) logic
- Carry-Lookahead Adder
- Tree Adder

11.2.1 Single-Bit Addition

Half Adder

$S = A \oplus B$

$C_{out} = A.B$

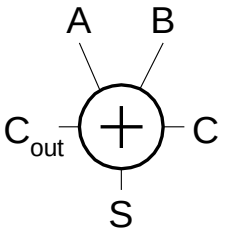


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full Adder

$S = A \oplus B \oplus C$

$C_{out} = MAJ(A, B, C)$



A	B	C	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 carry gate is also called a *majority gate because it produces a 1 if at least two of* the three inputs are 1.

Cout = A B + B C + A C

= [A' B' + C' (A' + B')]'

PGK

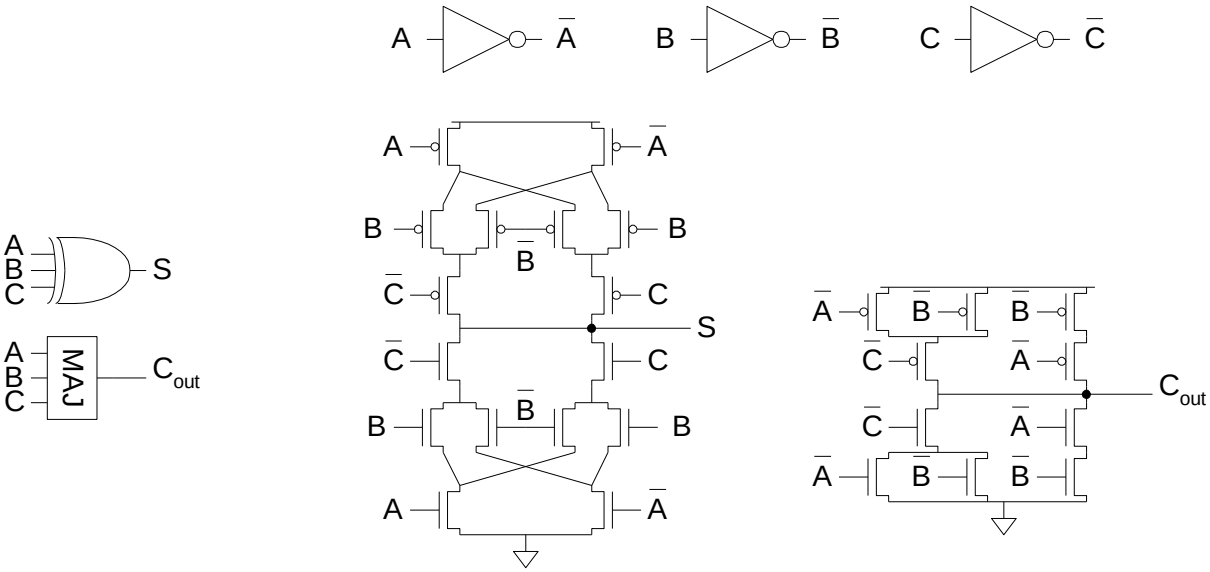
- For a full adder, define what happens to carries (in terms of A and B)
  - Generate: $C_{out} = 1$ independent of C
    - $G = A \bullet B$
  - Propagate: $C_{out} = C$
    - $P = A \oplus B$
  - Kill: $C_{out} = 0$ independent of C
    - $K = \sim A \bullet \sim B$

Full Adder Design I: 32 Transistors

- Brute force implementation from eqns:32 transistors

$$S = A \oplus B \oplus C$$

$$C_{out} = MAJ(A, B, C)$$



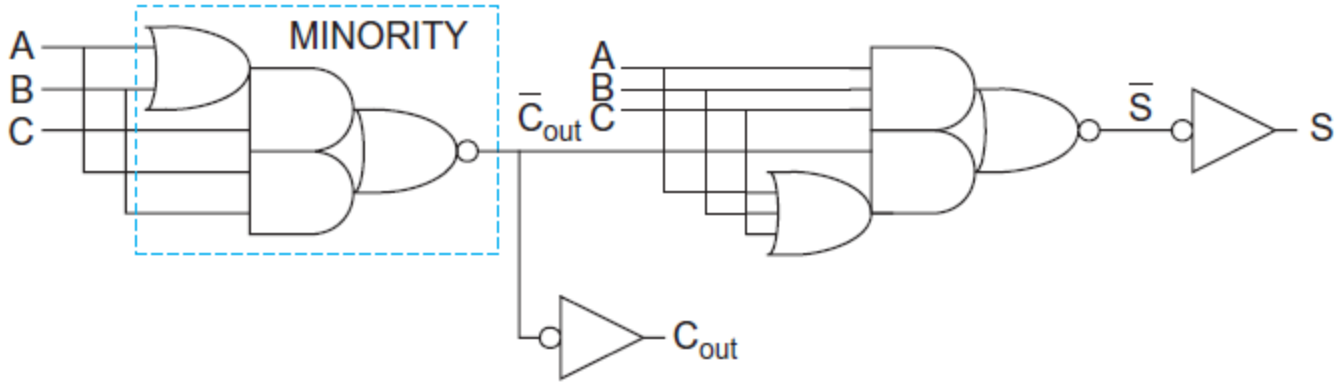
S = [A xor B xnor C]' Cout = [A' B' + C' (A' + B')]'

Full Adder Design II: 28 transistors

- Factor S in terms of C_{out}

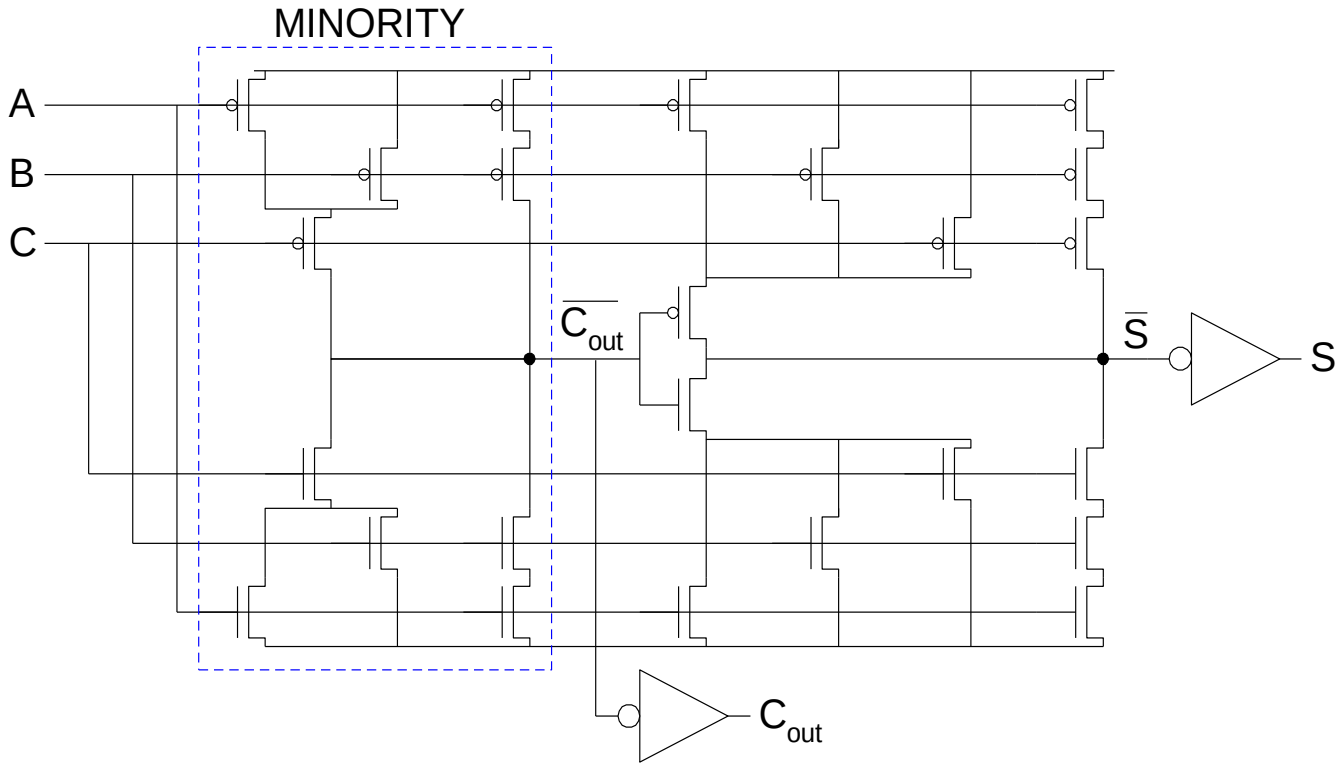
  $S = ABC + (A + B + C)(\sim C_{out})$

- Critical path is usually C to C_{out} in ripple adder, **28 transistors**, extra delay, good in CRA.

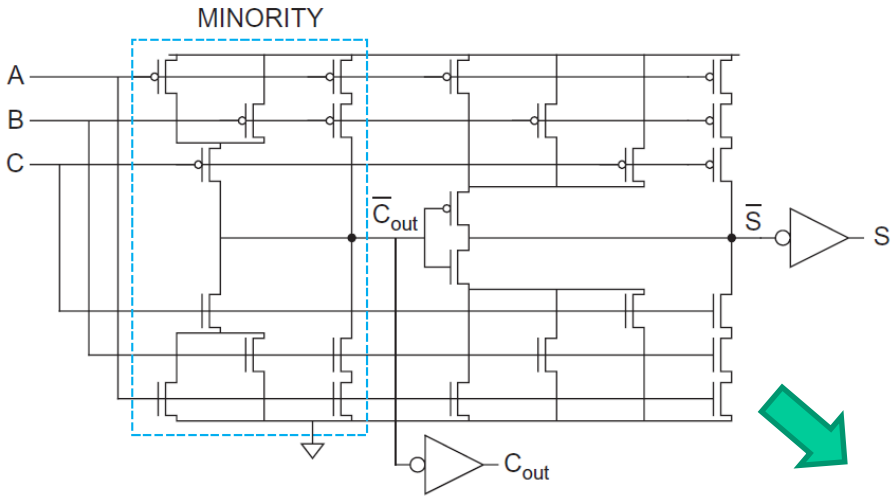


Full Adder Design II: 28 transistors

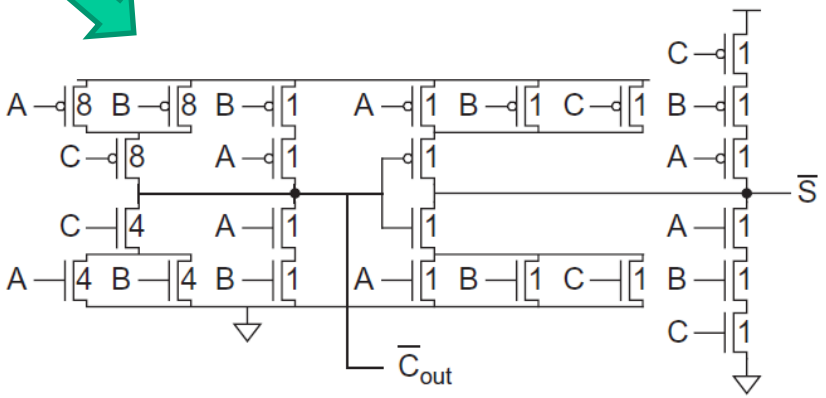
- Critical path is usually C to C_{out} in ripple adder
- Mirror adder: pMOS network is identical to nMOS network
  - Reduces number of transistors and simplify layout



Full Adder Design II: 28 -> 24 trans.

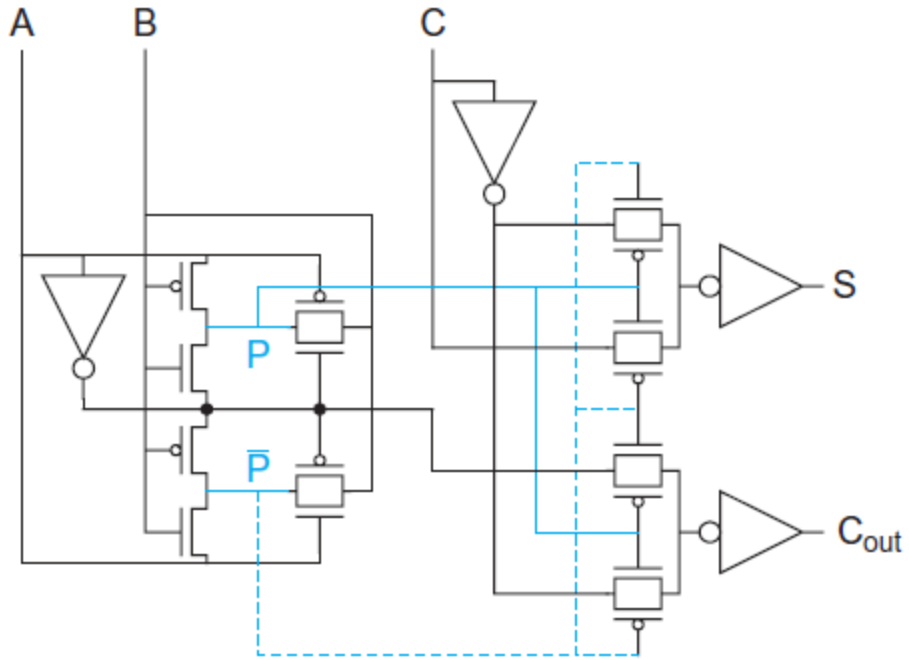


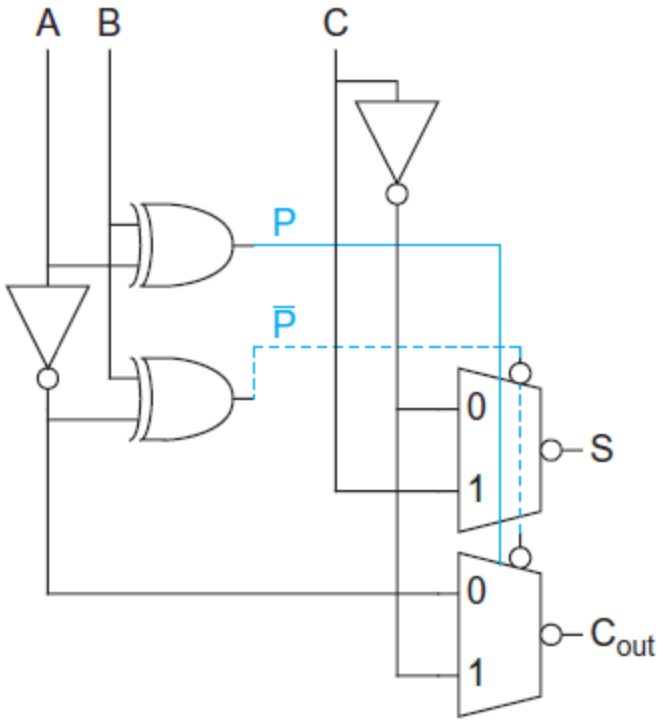
Hint: remove invertors, useful for some adder such as the one in Fig 11.11.



Full Adder Design III: 24 transistors

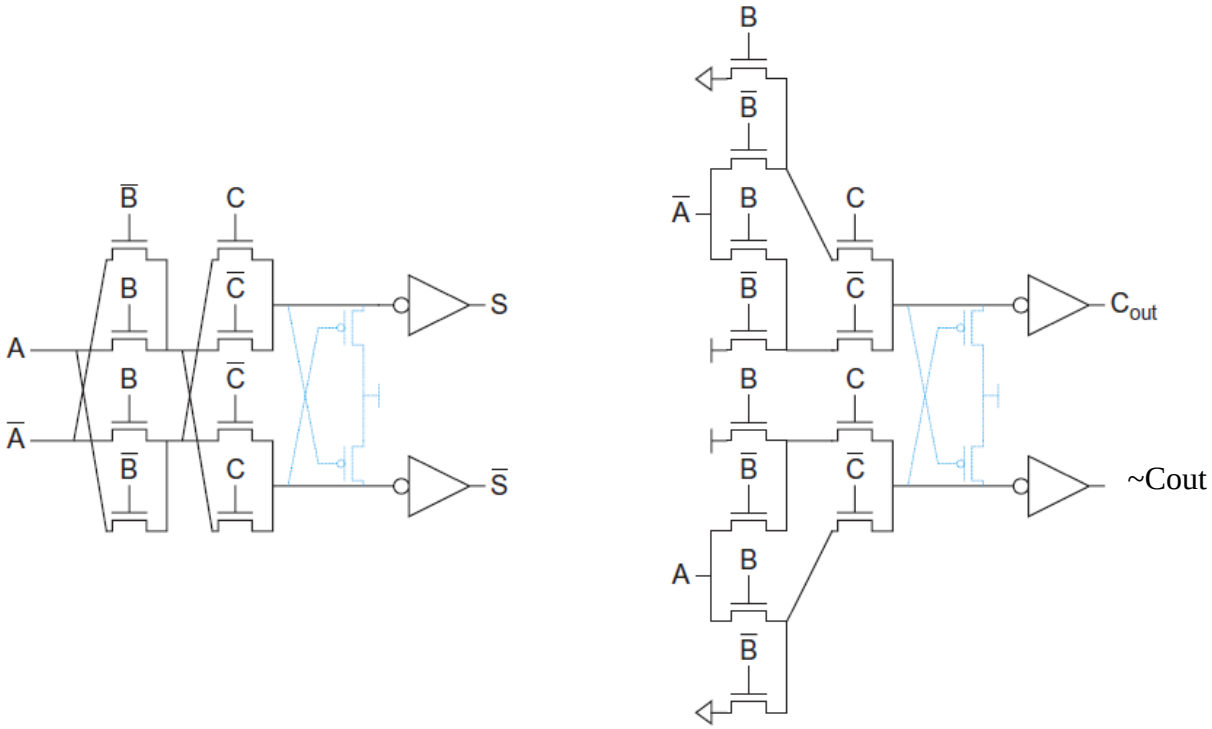
- Transmission gate to form multiplexers and XORs.
  - 24 transistors, equally delay (S and Cout).





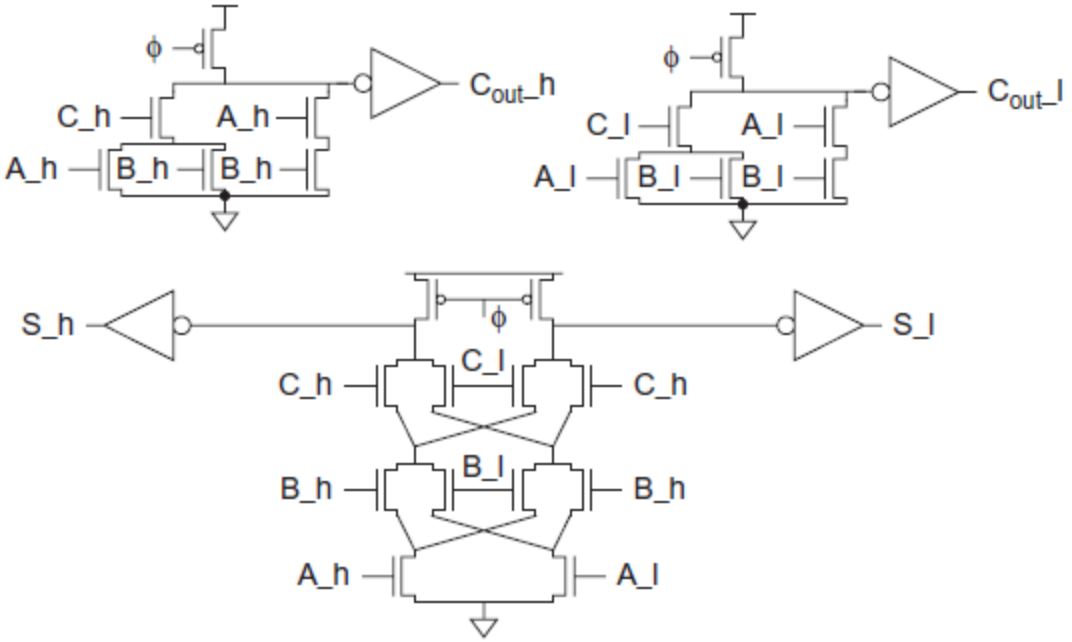
Full Adder Design IV: 40 trans.

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



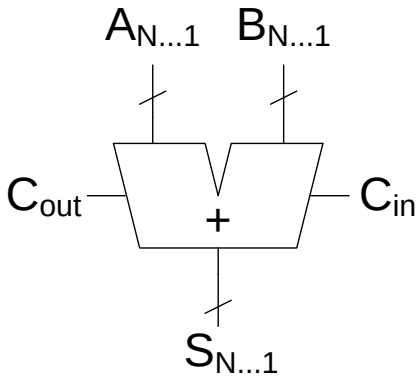
Full Adder Design V: 32 Trans.

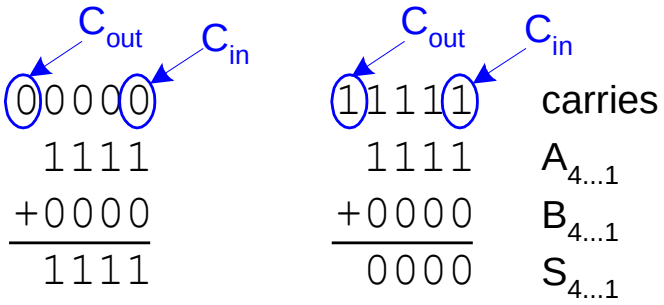
- Dual-rail domino (XOR/XNOR, MAJORITY/MINORITY)
  - Very fast, but large and power hungry
  - Used in very fast multipliers



11.2.2 Carry Propagate Adders

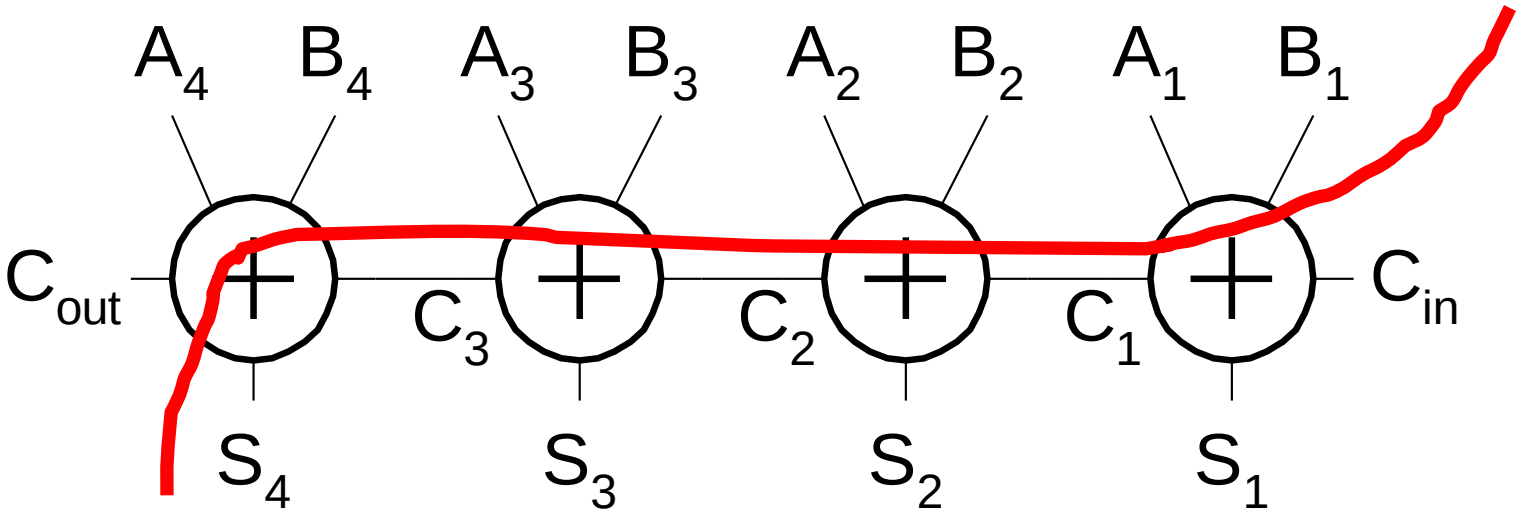
- N-bit adder called CPA
  - Each sum bit depends on all previous carries
  - How do we compute all these carries quickly?





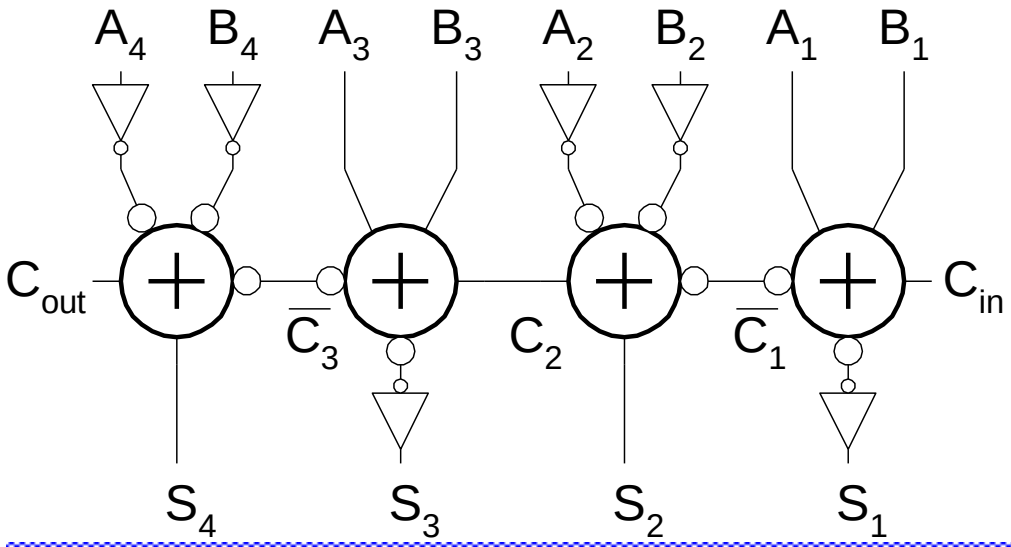
11.2.2.1 Carry-Ripple Adder

- Simplest design: cascade full adders
  - Critical path goes from C_{in} to C_{out}
  - Design full adder to have fast carry delay
- The delay of the adder is set by the time for the carries to ripple through the *N stages, so the* $t_{C->Cout}$ *delay should be* minimized.

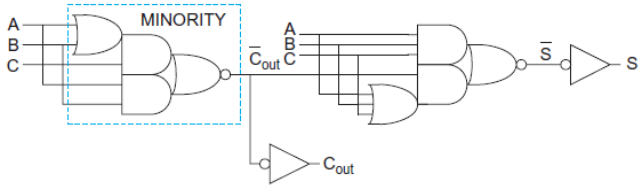


Omitting Inversions

- Critical path passes through majority gate.
- This delay can be reduced by omitting the inverters on the outputs, as shown below.
- Because addition is a *self-dual function* (i.e., the function of complementary inputs is the complement of the function), an inverting full adder receiving complementary inputs produces true outputs.



Recall:



Self-Dual Function

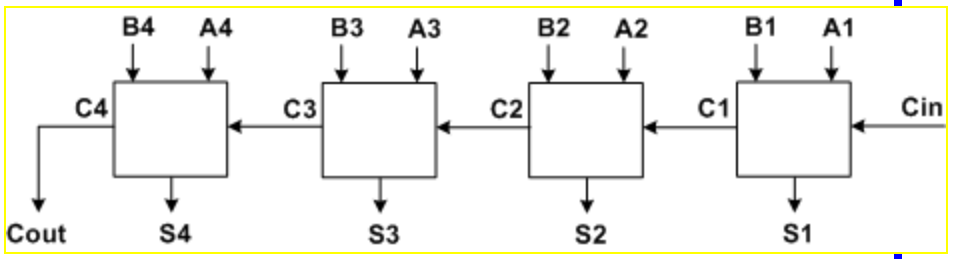
- The function of complementary inputs is the complement of the function),
- An inverting full adder receiving complementary inputs produces true outputs.

A	B	Cin	Cout	S	A'	B'	Cin'	Cout'	S'
0	0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	0	1	0
0	1	0	0	1	1	0	1	1	0
0	1	1	1	0	1	0	0	0	1
1	0	0	0	1	0	1	1	1	0
1	0	1	1	0	0	1	0	0	1
1	1	0	1	0	0	0	1	0	1
1	1	1	1	1	0	0	0	0	0

1.2.2.2 Carry Generation and Propagation

- We introduce the group Propagate (P) and group generate (G).
- We can generalize G and P signals to describe whether a group spanning bits *i…j, inclusive, generate a carry (G=1) or propagate a carry (P=1).*
- A group of bits:
  - generates a carry if its carry-out is true independent of the carry-in;
  - propagates a carry if its carry-out is true when there is a carry-in.

Recall Ps&Gs: 4-Bit Adder Example



- $C_{out} = G_{4:0}$
- $C_{in} = C_0 = G_0 = G_{0:0}$
- $C_1 = G_{1:0} = G_1 + P_1 C_0$
- $C_2 = G_{2:0} = G_2 + P_2 C_1$
  $= G_2 + P_2 G_1 + P_2 P_1 C_0$
- $C_3 = G_{3:0} = G_3 + P_3 C_2$
  $= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0$
- $C_4 = G_{4:0} = G_4 + P_4 C_3$
  $= G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_2 P_1 G_1 + P_4 P_3 P_2 P_1 C_0$

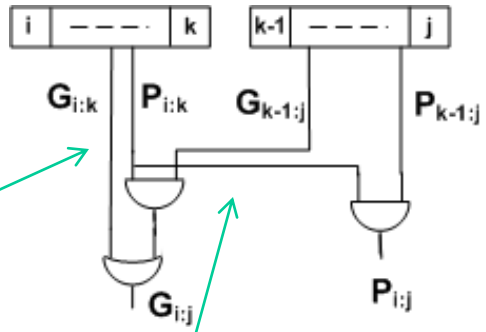
We need to simply this MESS!!

Carry Generation and Propagation

- For group of signals spanning i..j, and *i≥k>j. The* Generate and propagate signals:

$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j}$$

$$P_{i:j} = P_{i:k} \cdot P_{k-1:j}$$



  - A group generates a carry if
    - the upper (more significant) generates a carry **OR**
    - the lower portion generates and the upper portion propagates that carry.
  - The group propagates a carry if both the upper and lower portions propagate the carry

- Base case: group size = 1, which means i==j:

$$G_{i:i} \equiv G_i = A_i \cdot B_i$$
$$P_{i:i} \equiv P_i = A_i \oplus B_i$$

$$G_{0:0} \equiv G_0 = C_{in}$$
$$P_{0:0} \equiv P_0 = 0$$

- Sum/Carry: $S_i = P_i \oplus G_{i-1:0}$ $C_{i-1} = G_{i-1:0}$

Steps of Addition Using PG Logic

- Addition can be reduced to a three-step process:
  1. Computing bitwise generate and propagate signals using Eqs:

$$G_{i:i} \equiv G_i = A_i \cdot B_i \qquad G_{0:0} \equiv G_0 = C_{in}$$
$$P_{i:i} \equiv P_i = A_i \oplus B_i \qquad P_{0:0} \equiv P_0 = 0$$

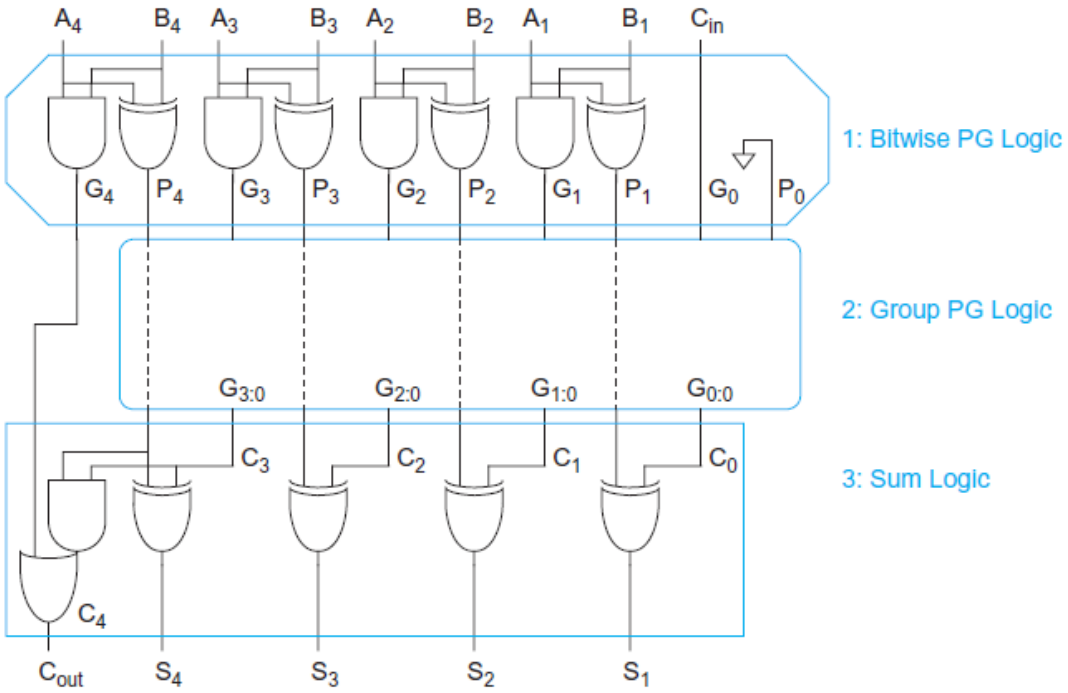
  2. Combining PG signals to determine group generates G_{i-1}*:0 for all* $N \geq i \geq 1$ *using* EQs:

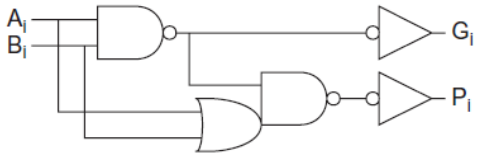
$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j}$$
$$P_{i:j} = P_{i:k} \cdot P_{k-1:j}$$

  3. Calculating the sums using EQ:

$$S_i = P_i \oplus G_{i-1:0}$$

Steps of Addition Using PG Logic



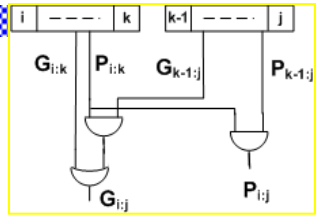


The first and third steps are routine, so most of the attention in the remainder of this section is devoted to alternatives for the group PG logic with different trade-offs between speed, area, and complexity. Some of the hardware can be shared in the bitwise PG logic, as shown

Higher Valency

- We discussed the equations for $G_{i:j}$ and $P_{i:j}$

$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j}$$
$$P_{i:j} = P_{i:k} \cdot P_{k-1:j}$$



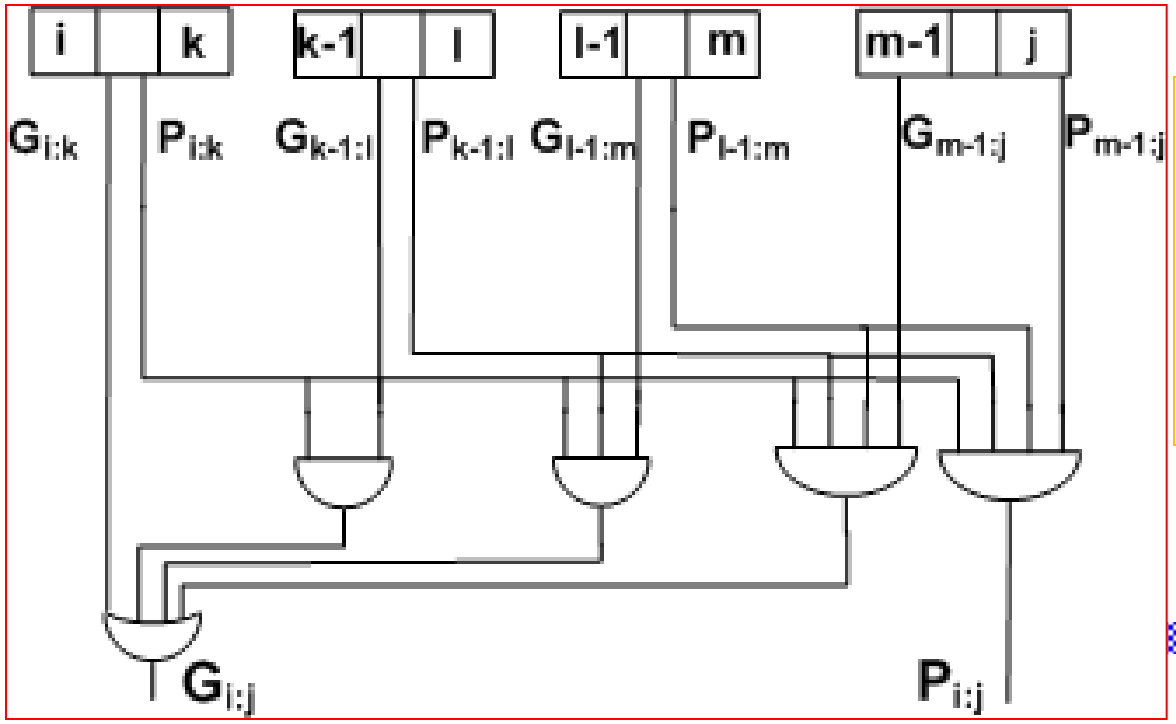
- The above equations define *valency-2 (also called radix-2) group PG logic because it combines* pairs of smaller groups.
- It is also possible to define higher-valency group logic to use fewer stages of more complex gates, as shown in the following equations:

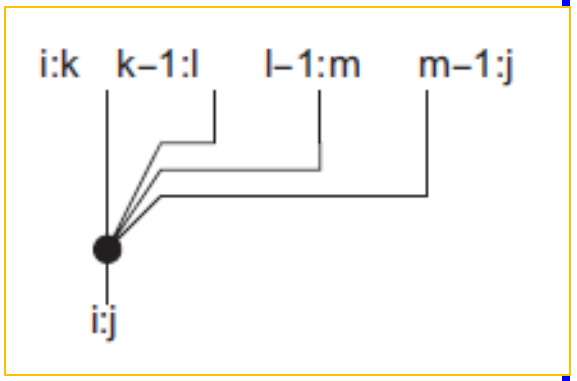
$$\left.\begin{aligned} G_{i:j} &= G_{i:k} + P_{i:k} \cdot G_{k-1:l} + P_{i:k} \cdot P_{k-1:l} \cdot G_{l-1:m} + P_{i:k} \cdot P_{k-1:l} \cdot P_{l-1:m} \cdot G_{m-1:j} \\ &= G_{i:k} + P_{i:k}\left(G_{k-1:l} + P_{k-1:l}\left(G_{l-1:m} + P_{l-1:m} G_{m-1:j}\right)\right) \\ P_{i:j} &= P_{i:k} \cdot P_{k-1:l} \cdot P_{l-1:m} \cdot P_{m-1:j} \end{aligned}\right\} \quad (i \geq k > l > m > j)$$

- For example, in valency-4 group logic, a group propagates the carry if all four portions propagate. A group generates a carry if the upper portion generates, the second portion generates and the upper propagates, the third generates and the upper two propagate, or the lower generates and the upper three propagate.

Valency-4

$$
\begin{aligned}
G_{i:j} &= G_{i:k} + P_{i:k} \cdot G_{k-1:l} + P_{i:k} \cdot P_{k-1:l} \cdot G_{l-1:m} + P_{i:k} \cdot P_{k-1:l} \cdot P_{l-1:m} \cdot G_{m-1:j} \\
&= G_{i:k} + P_{i:k} \left(G_{k-1:l} + P_{k-1:l} \left(G_{l-1:m} + P_{l-1:m} G_{m-1:j} \right) \right) \\
P_{i:j} &= P_{i:k} \cdot P_{k-1:l} \cdot P_{l-1:m} \cdot P_{m-1:j}
\end{aligned}
\qquad (i \geq k > l > m > j)
$$





11.2.2.3 PG Carry Ripple Addition

- The critical path of the carry-ripple adder passes from carry-in to carry-out along the carry chain majority gates.
- As the *P and G signals will have* already stabilized by the time the carry arrives, we can use them to simplify the majority function into an AND-OR gate.

$$C_i = A_i B_i + (A_i + B_i) C_{i-1}$$
$$= A_i B_i + (A_i \oplus B_i) C_{i-1}$$
$$= G_i + P_i C_{i-1}$$

- Because $C_i = G_{i:0}$, *carry-ripple addition can now be viewed as the extreme case of* group PG logic in which a 1-bit group is combined with an *i-bit group to form an (i+1)*-bit group.

$$G_{i:0} = G_i + P_i \cdot G_{i-1:0}$$

- In this extreme, the group propagate signals are never used and need not be computed.

4-bit Carry-Ripple Using PG logic

$$t_{\text{ripple}} = t_{pg} + (N-1)t_{AO} + t_{\text{xor}}$$

where t_{pg} *is the delay of the 1-bit propagate/generate gates,*
t_{AO} *is the delay of the ANDOR* gate in the gray cell,
and t_{xor} *is the delay of the final sum XOR.*

For N=4

$t_{ripple}=t_{pg}+ 3\times t_{AO}+t_{xor}$

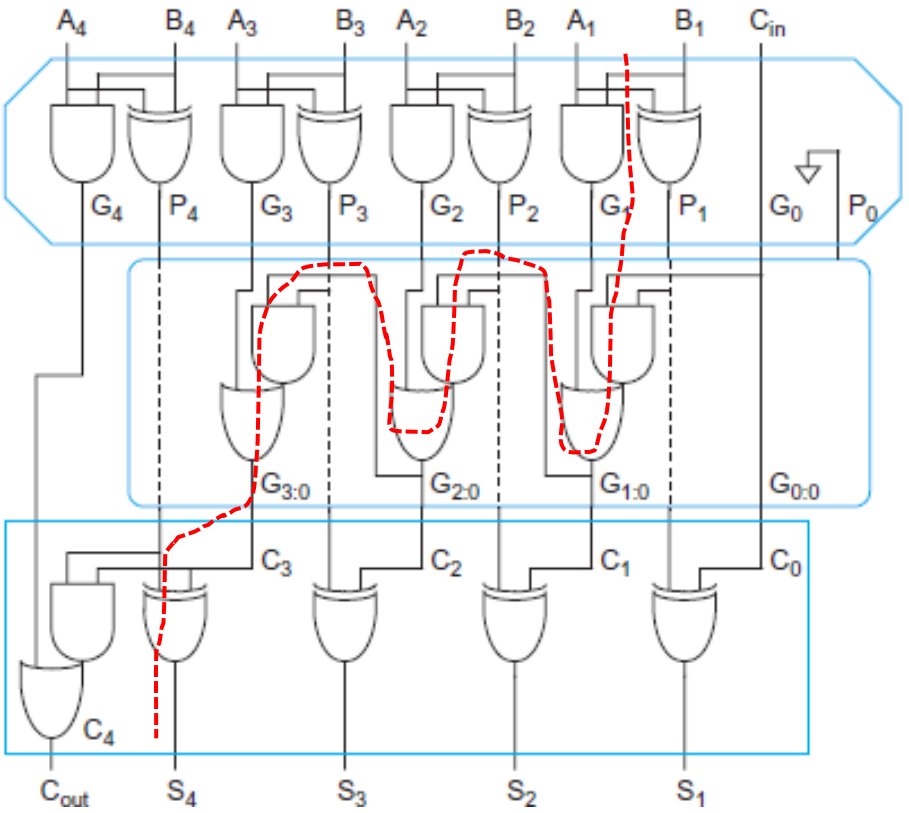
The ripple counter is a special case of CLA adder (covered next). The CLA adder delay:

$t_{cla}=t_{pg}+t_{pg(n)}+[(n-1)+(k-1)]t_{AO}+t_{xor}$

For ripple adder k=1,

*N=4, k=1, N= n*k, so: n=4*

$t_{pg(k=1)}$ *is not used and never computed.*

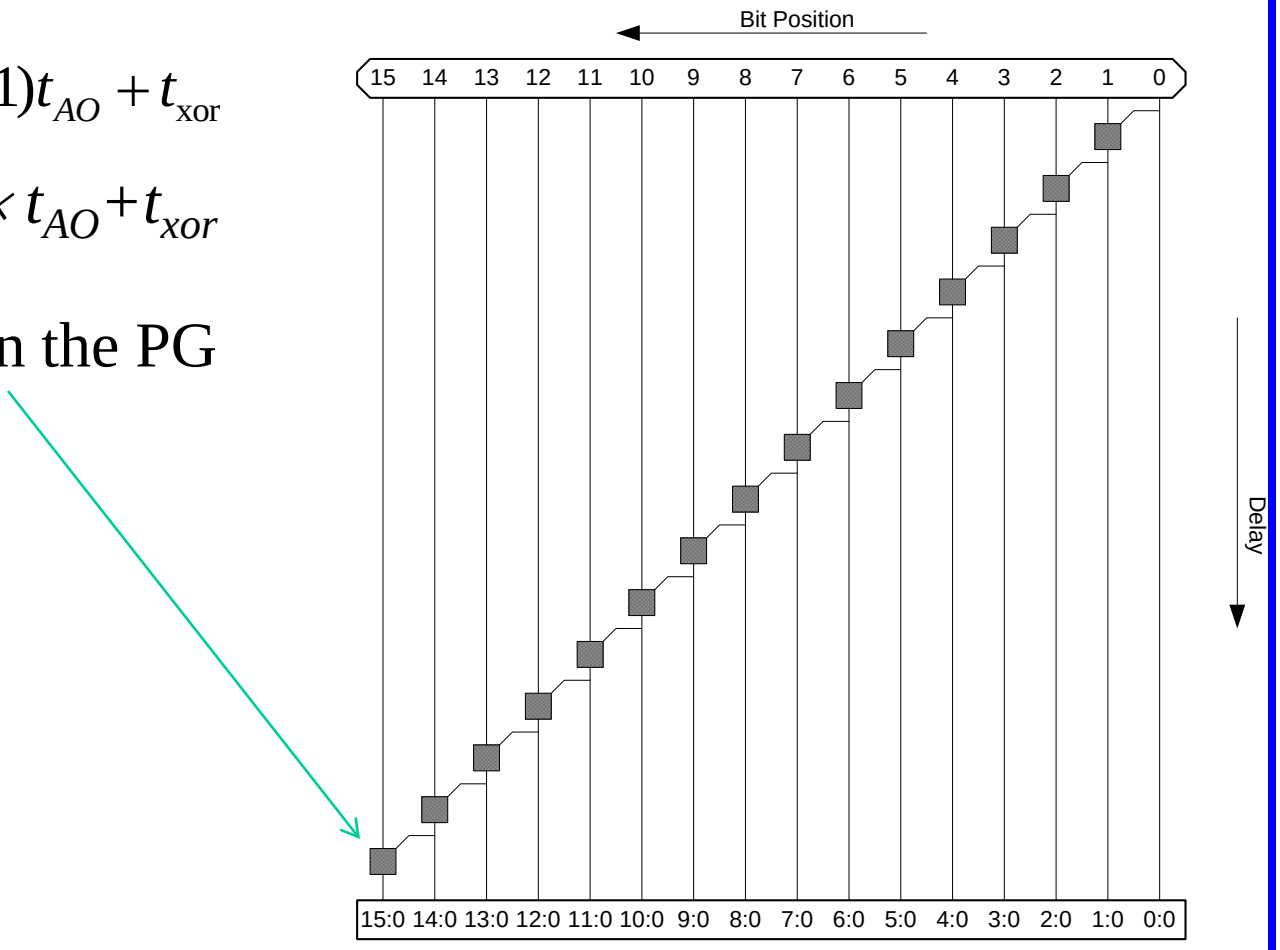


16-bit Carry-Ripple using PG Logic

$$t_{\text{ripple}} = t_{pg} + (N-1)t_{AO} + t_{\text{xor}}$$

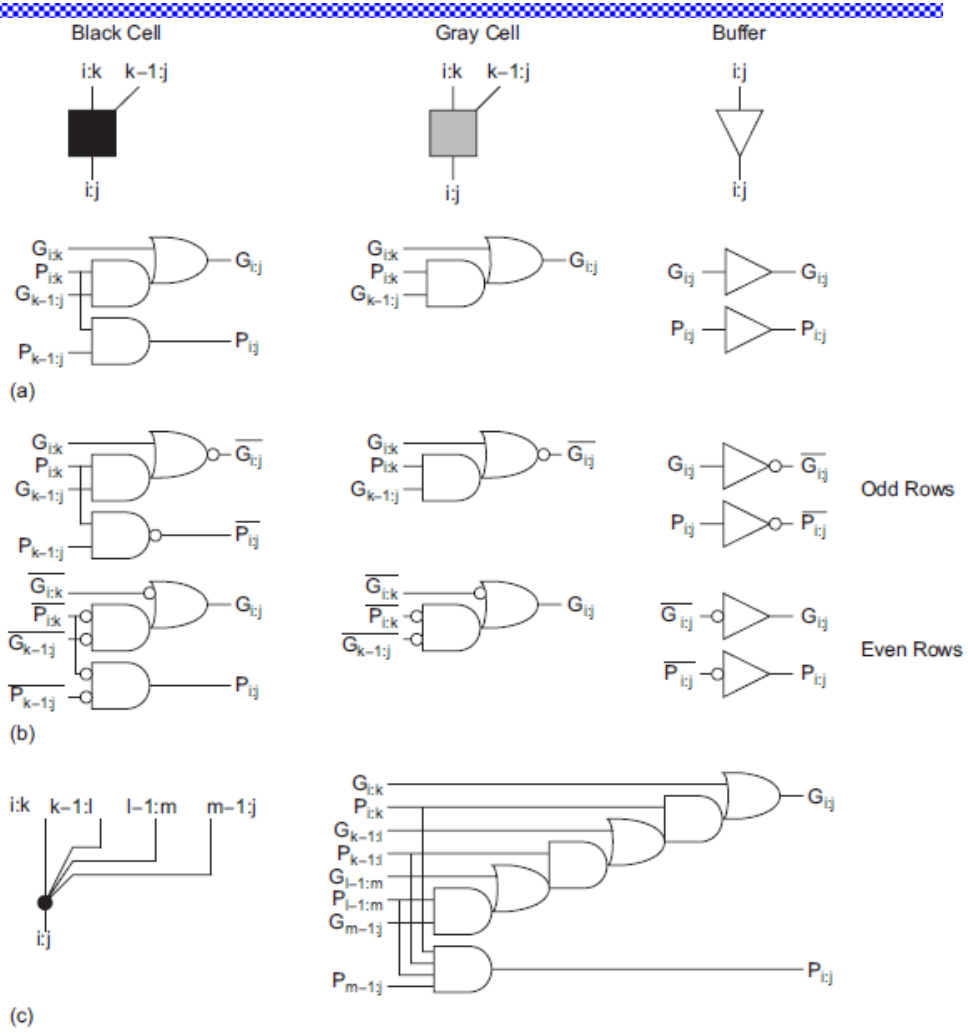
$$t_{ripple} = t_{pg} + 15 \times t_{AO} + t_{xor}$$

AND-OR gates in the PG network.



Groups PG Cells

- **Black cells** contain the group generate and propagate logic (an AND-OR gate and an AND gate) defined in EQ (11.4).
- **Gray cells** containing only the group generate logic are used at the final cell position in each column because only the group generate signal is required to compute the sums.
- **Buffers** can be used to minimize the load on critical paths.
- Each line represents a *bundle of the group generate* and propagate signals (propagate signals are omitted after gray cells).

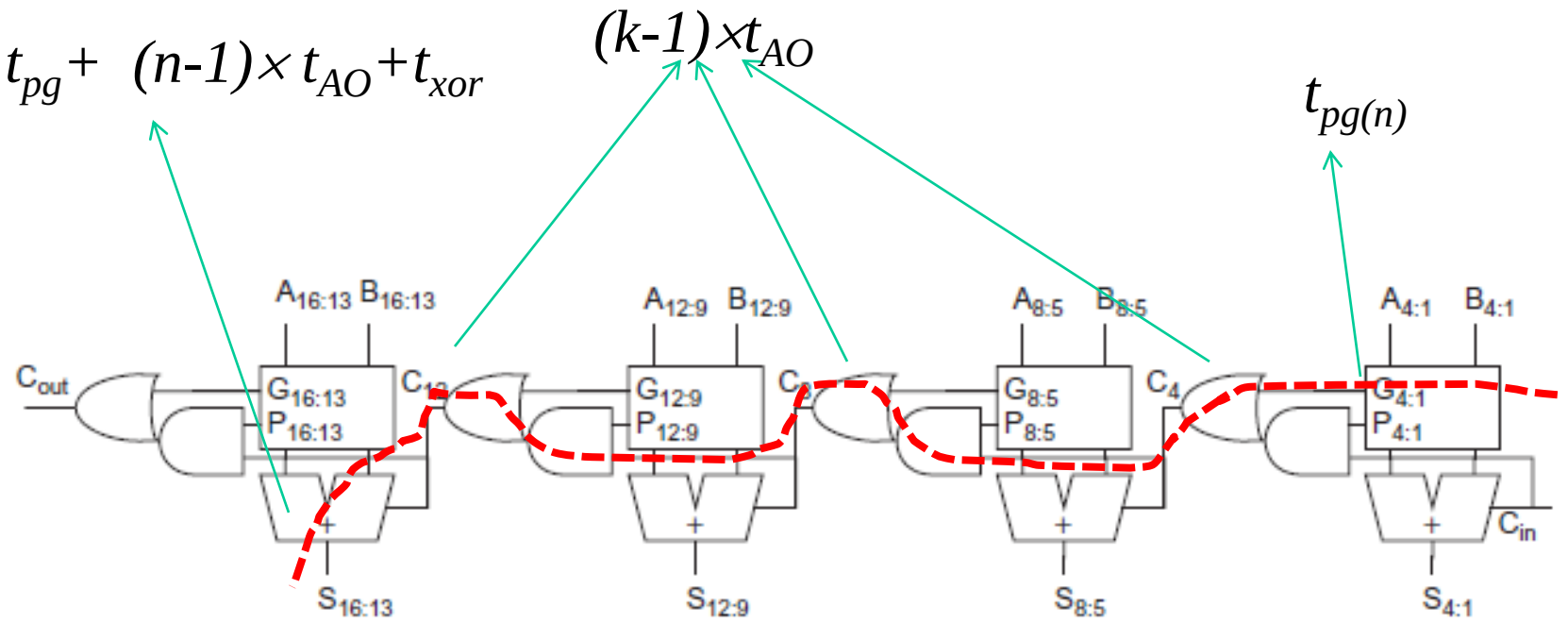


11.2.2.6 Carry-Lookahead Adder

- Computes group generate signals and group propagate signals to avoid waiting for a ripple
- Uses valency-4 black cells to compute 4-bit group PG signals.
- In general the delay for CLA using ***k*** *groups of* ***n*** *bits:*

  $t_{cla} = t_{pg} + t_{pg(n)} + [(n-1)+(k-1)]t_{AO} + t_{xor}$

  where $t_{pg(n)}$ *is the delay of the AND-OR-AND-OR-…-AND-OR gate computing the* valency-*n generate signal.*



CLA: PG logic

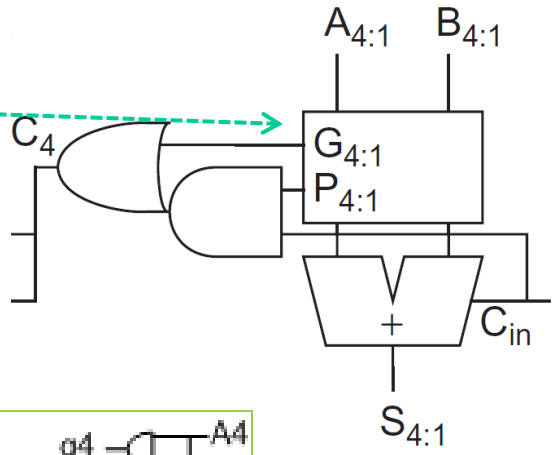
- The PG logic is shown below:

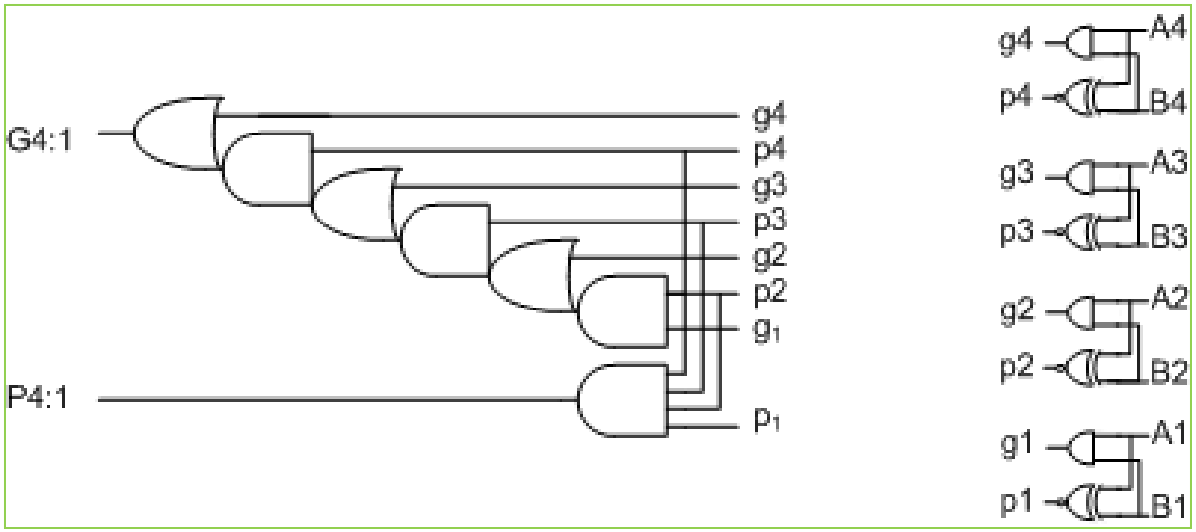
$$P4 = p4.\ p3\ .\ p2.\ p1$$

$$G4 = g4 + p4\ (\ g3 + p3\ (\ g2 + p2\ g1))$$

$$= g4 + p4\ .\ g3\ \ + p4\ p3\ g2$$

$$+ p4\ .\ p3\ .\ \ p2\ .\ g1$$





11.2.2.8 Tree Adders

- For wide adders (roughly, *N > 16 bits), the delay of carry-lookahead* adders becomes dominated by the delay of passing the carry through the lookahead stages. This delay can be reduced by looking ahead across the lookahead blocks.
- In general, you can construct a multilevel tree of look-ahead structures to achieve delay that grows with **log *N***.
- Such adders are variously referred to as *tree adders, logarithmic adders, multilevel-lookahead adders, parallel-prefix adders, or simply lookahead adders.*

11.2.2.8 Tree Adders

- There are many ways to build the lookahead tree that offer trade-offs among
  - the number of stages of logic,
  - the number of logic gates,
  - the maximum fanout on each gate,
  - the amount of wiring between stages.
- The delay of tree adders is approximately

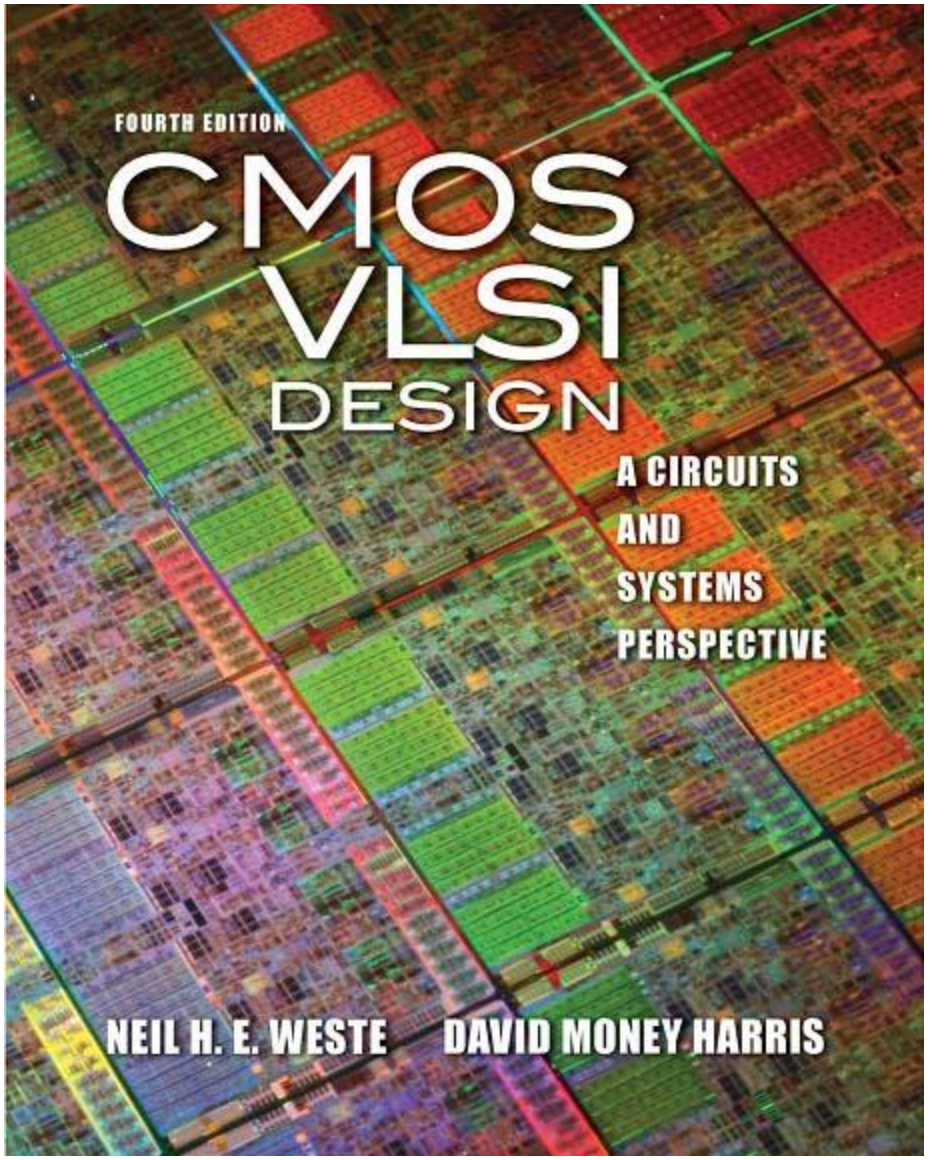
$$t_{\text{tree}} \approx t_{pg} + \lceil \log_2 N \rceil t_{AO} + t_{\text{xor}}$$

Tree Adder Examples

- Reading the examples in the textbook.
  - Brent-Kung
  - Sklansky
  - Kogge-Stone
  - Han-Carlson
  - Knowles [2,1,1,1]
  - Ladner-Fischer

CPE 110408423
VLSI Design
Chapter 11: Datapath Subsystems

Bassam Jammil
[Computer Engineering Department,
Hashemite University]



Lecture 2: Datapath Functional Units

Outline

- What is datapath
- Multi-input Adders
- 1's & 0's Detectors
- Comparators
- Shifters
- Multipliers

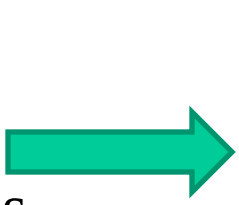
Datapath

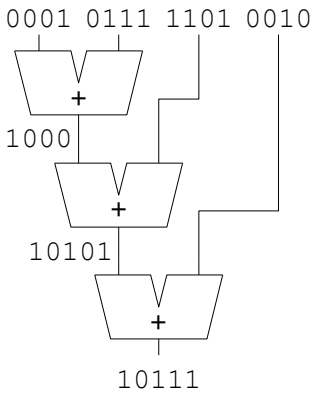
- CPU consists mainly of:
  - Datapath: functional units which perform operations on data
  - Control: sequences datapath, memory, ...
  - Memory elements
  - Special purpose cells
    - I/O
    - Power distribution
    - Clock generation and distribution
    - Analog and RF

11.2.4 Multi-input Adders

- Suppose we want to add ***k*** N-bit words
  - Ex: 0001 + 0111 + 1101 + 0010 = 10111
- Straightforward solution: ***k-1*** N-input cascaded carry-propagation adders (CPAs).
  - The main problems: Large and slow

Adding 4 4-bit numbers using three cascaded CPAs.
Note: ***k***=4

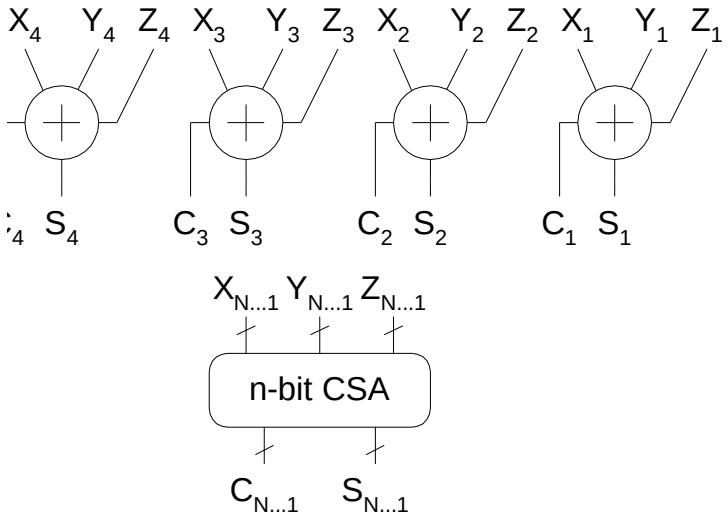




Carry-Save Adder (CSA)

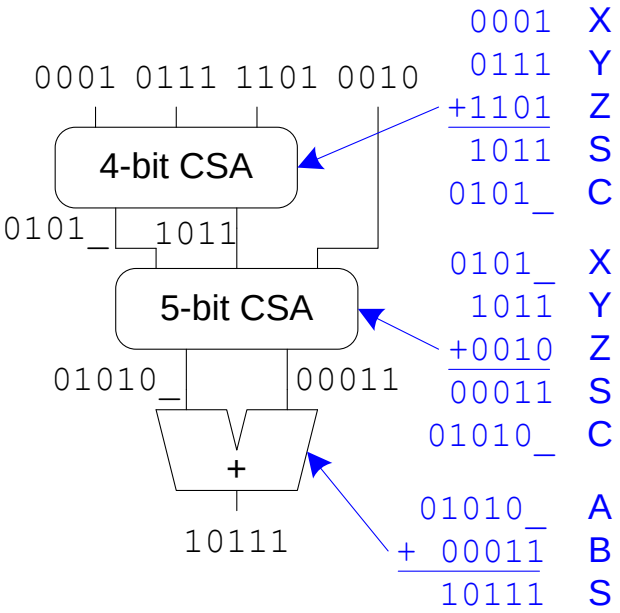
- CSA is a full adder sums 3 inputs and produces 2 outputs
  - Carry output has twice *weight* of sum output
- N full adders in parallel are called *carry save adder*
  - Produce N sums and N carry outs

The weight of X_i , Y_i , Z_i and S_i is 2^{i-1}.
The weight of Ci is 2^i.
$X+Y+Z = S + 2C$
So, the carry is not propagated, rather it is saved. Hence, the name carry save adder.
When one of the inputs to a CSA is a constant, the hardware can be reduced further. If a bit of the input is 0, the CSA column reduces to a half-adder. If the bit is 1, the CSA column simplifies to $S=A \oplus B$ *and* $C = A+B$.



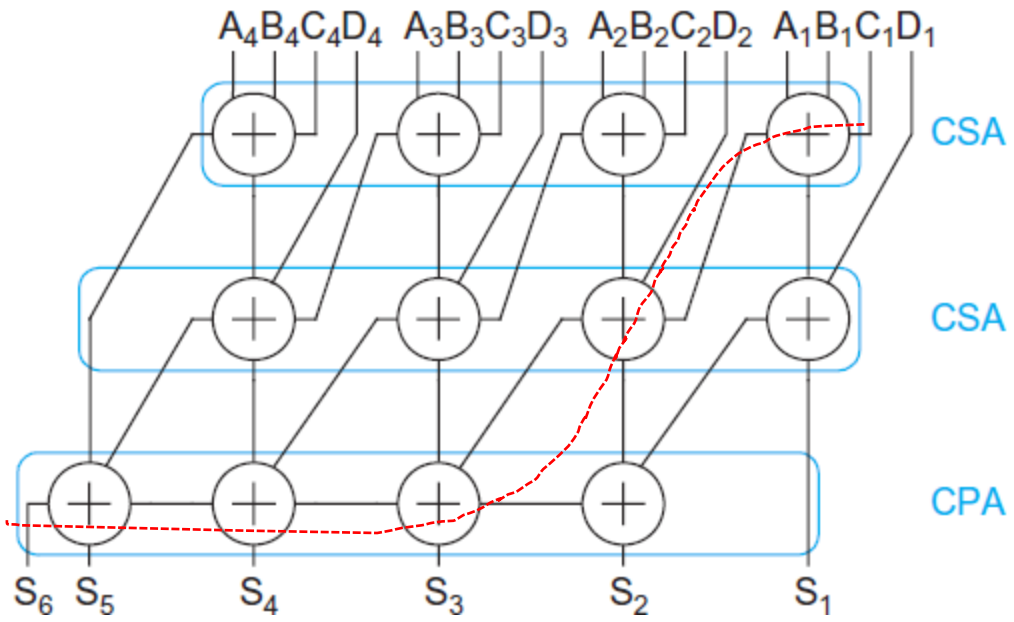
CSA Application

- Use k-2 stages of CSAs
  - Keep result in carry-save redundant form
- Final CPA computes actual result



The critical path of adding k numbers

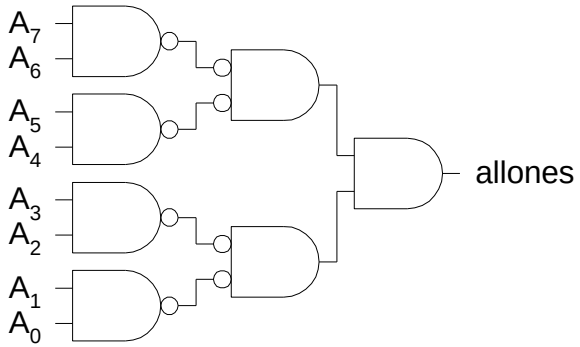
- In general *k* numbers can be summed with *k – 2 [3:2] CSAs and only one CPA.*
- This is much faster compared with *k-1* CPAs.

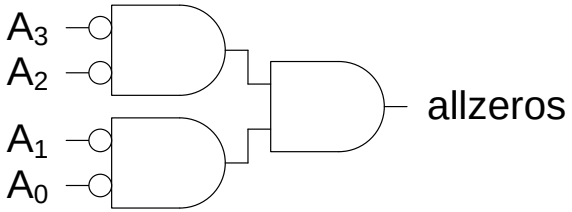


11.3 1's & 0's Detectors

- 0's detector: A = 00…000
  - It is a function which outputs 1 when all N inputs are zeros.
  - Designed using N-input AND gate
- 1's detector: A = 11…111
  - It is a function which outputs 1 when all N inputs are ones.
  - Designed with
    - NOR gate
    - NOTs + 1's detector

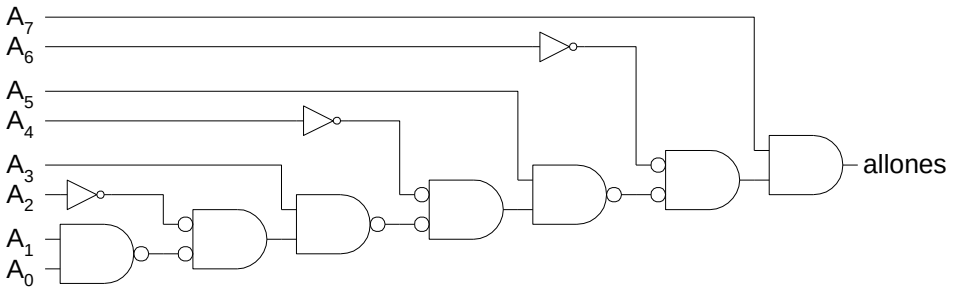
11.3 1's & 0's Detectors





The gates can be built in a binary tree style. The timing path has log *N* stages.

If inputs arrives in different times, then build the design in a sequential /cascaded style. The timing path is *N-1* stages.

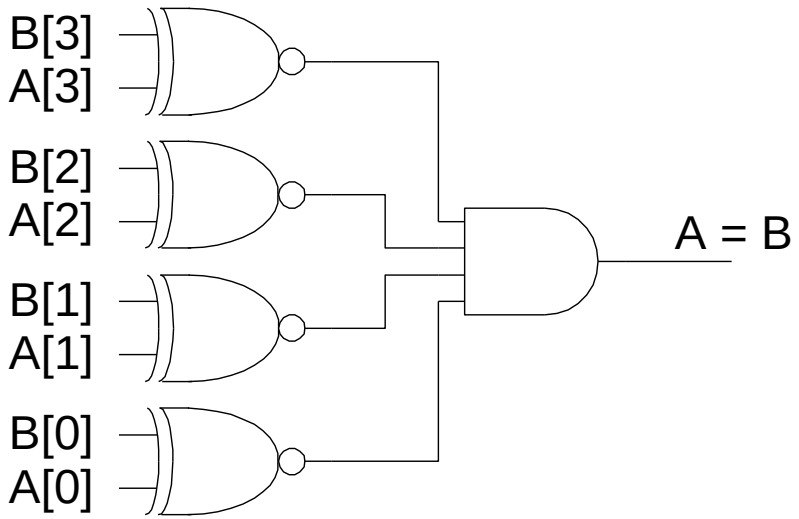


11.4 Comparators: unsigned numbers

- Naming convention
  - 0's detector: A = 00…000
  - 1's detector: A = 11…111
  - Equality comparator: A = B
  - Magnitude comparator: A < B
- A *magnitude comparator* determines the larger of two binary numbers.
- *To compare two* **unsigned** numbers *A and B, compute* $B - A = B + \sim A + 1$.
  - *If there is a carry-out,* $A \leq B$; otherwise, $A > B$.
  - *A zero detector indicates that the numbers are equal.*

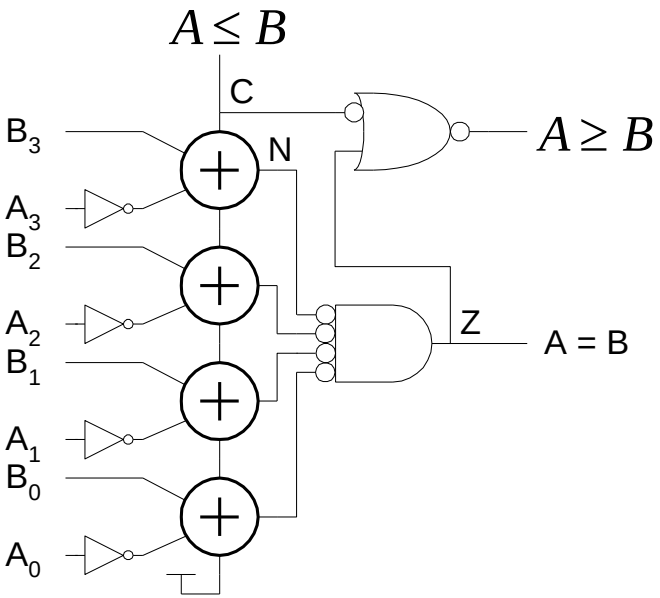
Equality Comparator

- Check if each bit is equal (XNOR, aka equality gate)
- 1's detect on bitwise equality



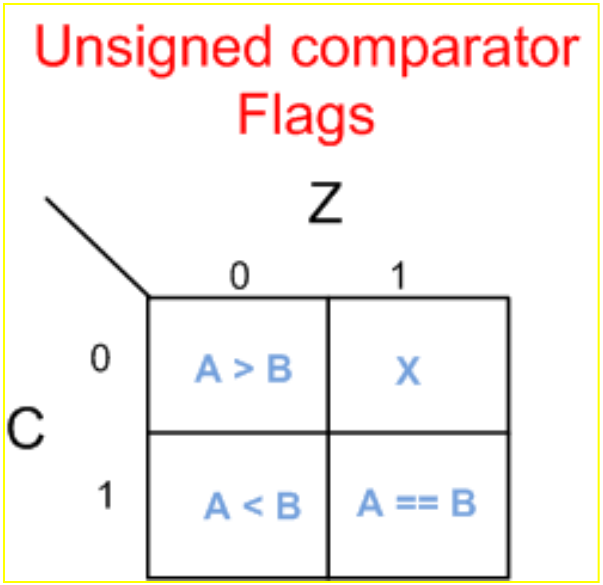
Magnitude Comparator

- Compute B – A and look at sign
- B – A = B + ~A + 1
- For unsigned numbers, carry out is sign bit



Unsigned Magnitute Comparator

Relation	Unsigned Comparison
$A = B$	Z
$A \neq B$	$\overline{Z}$
$A < B$	$C \cdot \overline{Z}$
$A > B$	$\overline{C}$
$A \leq B$	C
$A \geq B$	$\overline{C} + Z$



Comparators: signed vs. unsigned

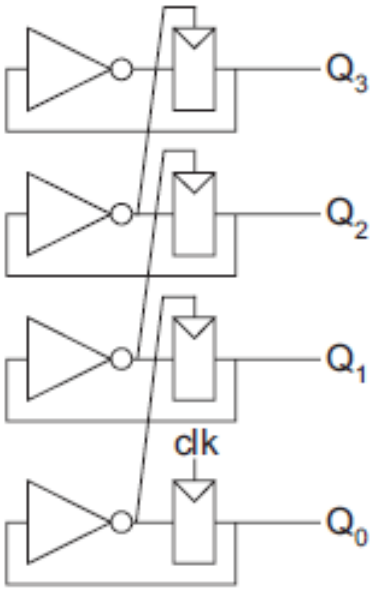
- Reading only
- For signed numbers, comparison is harder and depends on the following flags.
  - C: carry out
  - Z: zero (all bits of B – A are 0)
  - N: negative (MSB of result)
  - V: overflow (inputs had different signs, output sign ≠ B)
  - S: N xor V (sign of result)

11.5 Counters

- Two commonly used types of counters are binary counters and **linear-feedback shift registers**.
- An N-bit binary counter sequences through 2^N outputs in binary order.
- An N-bit linear-feedback shift register sequences through up to $2^N - 1$ outputs in pseudo-random order
- Some of the common features of counters include the following: *Resettable, Loadable, Enabled Reversible(UP/DOWN input), Terminal Count (TC output asserted when counter overflows* or underflows).

11.5.1 Binary Counters

- Asynchronous Counter
  - It is composed of *N registers connected in toggle configuration, where the falling* transition of each register clocks the subsequent register.
  - Therefore, the delay can be quite long.
  - It has no reset signal, making it difficult to test.
  - It is good frequency divider.



Binary Counters

- Shown below:
  - (a) up/down counter
  - (b) up counter (incrementer)
  - (c) Fully featured counter with reset/load/ up/dn.

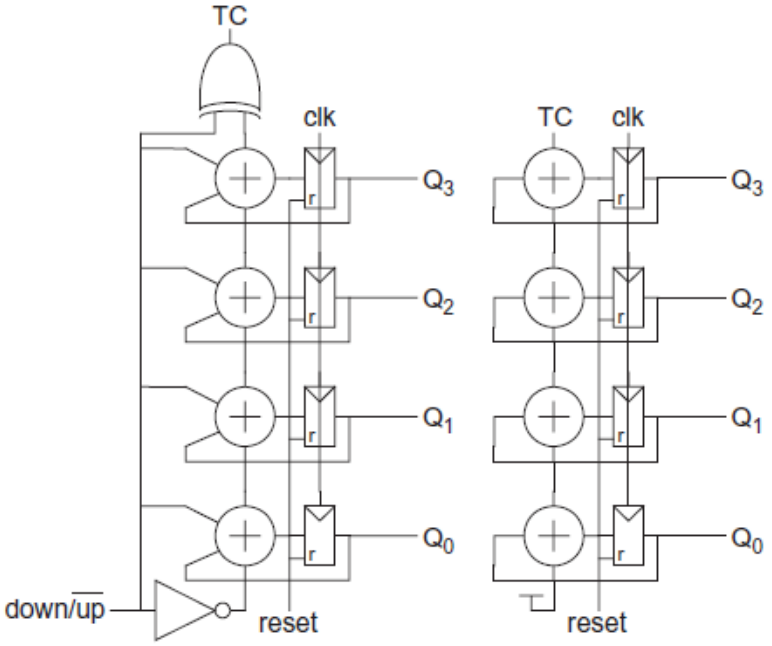


FIGURE 11.49 Synchronous counters

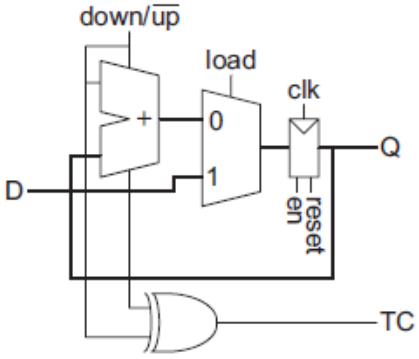


FIGURE 11.50 Synchronous up/down counter with reset, load, and enable

11.5.4 Linear-Feedback Shift Register

- A linear-feedback shift register (LFSR) consists of *N registers configured as a shift register.* The input to the shift register comes from the XOR of particular bits of the register, as shown in the Figure for a 3-bit LFSR. On reset, the registers must be initialized to a nonzero value (e.g., all 1s). The pattern of outputs for the LFSR is shown in the Table.
- The output *Y follows the 7-bit sequence [1110010].* This is an example of a *pseudorandom bit sequence (PRBS).*

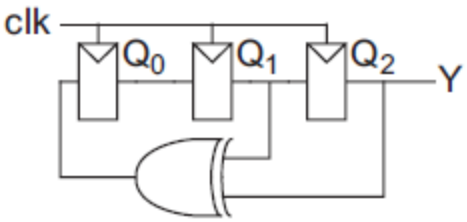


FIGURE 11.54 3-bit LFSR

TABLE 11.7 LFSR sequence

Cycle	Q_0	Q_1	Q_2 / Y
0	1	1	1
1	0	1	1
2	0	0	1
3	1	0	0
4	0	1	0
5	1	0	1
6	1	1	0
7	1	1	1
Repeats forever			

Linear-Feedback Shift Register

- **Maximal-length shift register** outputs sequences through all $2^n - 1$ combinations (excluding all 0s).
  - Such as the example in the previous slide.
- The inputs fed to the XOR are called the tap sequence and are often specified with a **characteristic polynomial**.
- For example, this 3-bit LFSR has the characteristic polynomial $1 + x^2 + x^3$ because the taps come after the second and third registers.

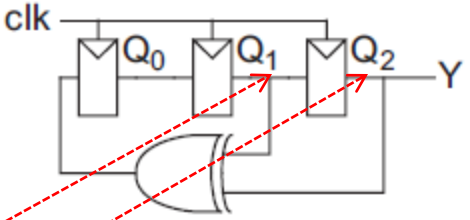


FIGURE 11.54 3-bit LFSR

LFSR Example

Example 11.1

Sketch an 8-bit linear-feedback shift register. How long is the pseudo-random bit sequence that it produces?

SOLUTION: Figure 11.55 shows an 8-bit LFSR using the four taps after the 1st, 6th, 7th, and 8th bits, as given in Table 11.7. It produces a sequence of $2^8 - 1 = 255$ bits before repeating.

Characteristic Polynomial is:

$1 + x^1 + x^6 + x^7 + x^8$

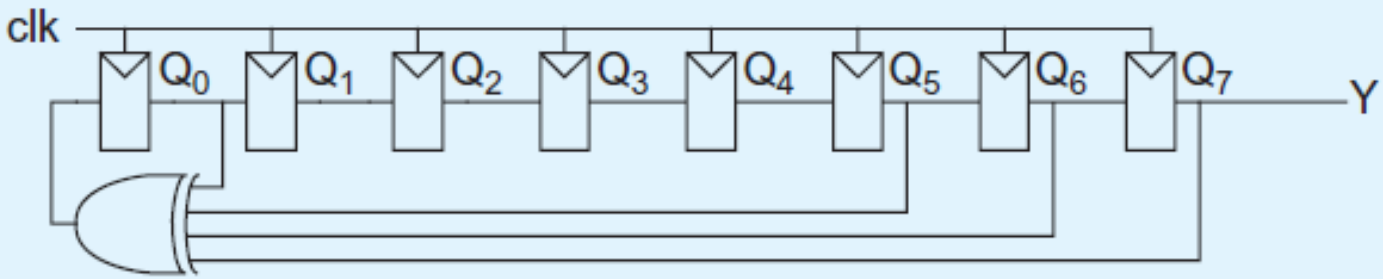


FIGURE 11.55 8-bit LFSR

11.8 Shifters

- Shifts can either be performed by a constant or variable amount.
  - Constant shifts are trivial in hardware, requiring only wires. They are also an efficient way to perform multiplication or division by powers of two.
  - A variable shifter takes an *N-bit input, A, a shift amount, k,* and control signals indicating the shift type and direction. It produces an *N-bit output, Y.*

Shifters

- There are three common types of variable shifts, each of which can be to the left or right:

1. Logical Shift:

- Shifts number left or right and fills with 0's
  - 1011 LSR 1 = 0101 1011 LSL1 = 0110

2. Arithmetic Shift:

- Shifts number left or right. Right shift sign extends
  - 1011 ASR1 = 1101 1011 ASL1 = 0110

3. Rotate:

- Shifts number left or right and fills with lost bits
  - 1011 ROR1 = 1101 1011 ROL1 = 0111

Implementing Rotate Operation

- Array shifter
  - Involves an array of N N-input multiplexers to select each of the outputs from each of the possible input positions.
  - It requires a decoder to produce the 1-of-N-hot shift amount.
  - In practice, multiplexers with more than 4–8 inputs have excessive parasitic capacitance
  - , so they are faster
- Logarithmic shifter
  - Uses $\log_v N$ levels of v-input multiplexers
  - For example, in a radix-2 logarithmic shifter, the first level shifts by N/2, the second by N/4, and so forth until the final level shifts by 1.
  - In a logarithmic shifter, no decoder is necessary.

Implementing Rotate Operation

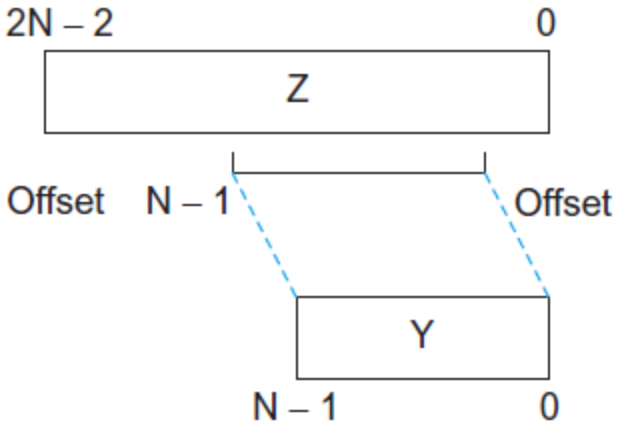
- A left rotate by *k bits is equivalent to a right rotate by* ***N – k*** *bits.*
  - *Computing* ***N – k*** requires a subtracter in the critical path.
  - ***N – k*** *= N + k + 1 = k + 1. Thus, the left* rotate can be performed by preshifting right by 1, then doing a right rotate by the complemented shift amount.
- Logical and arithmetic shifts are similar to rotates, but must replace bits at one end or the other with a *kill value (either 0 or the sign bit).*

Shift Architectures

- The two major shifter architectures are **funnel shifters** and **barrel shifters**.
- Funnel shifter
  - the kill values are incorporated at the beginning,
- Barrel shifter
  - the kill values are chosen at the end
- Comparison:
  - Both barrel and funnel shifters can use array or logarithmic implementations.
  - For general-purpose shifting, both architectures are comparable in energy and delay.
  - If only shift operations (but not rotates) are required, the funnel architecture is simpler, while if only rotates (but not shifts) are required, the barrel is simpler.

11.8.1 Funnel Shifter

- The funnel shifter creates a 2N – 1-bit input word ***Z*** from ***A*** and/or the kill values, then selects an N-bit field from this input word
- A funnel shifter can do all six types of shifts
- Selects N-bit field Y from 2N–1-bit input
  - Shift by k bits ($0 \leq k < N$)
  - Logically involves N N:1 multiplexers

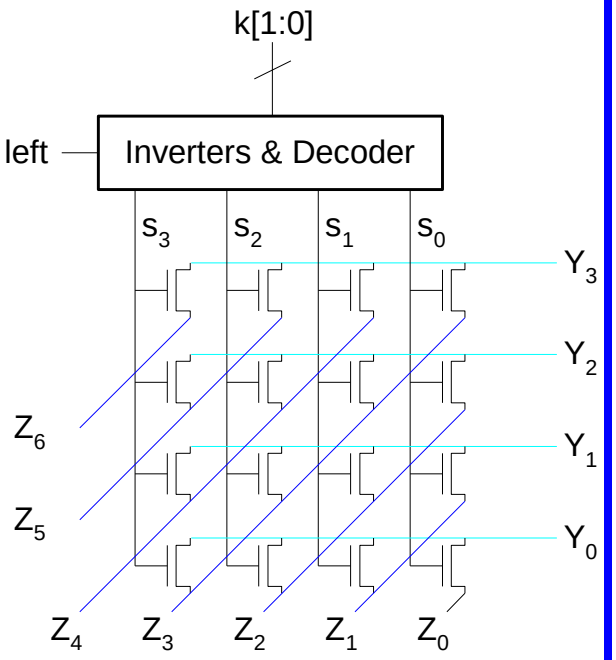


Funnel Source Generator

Shift Type	$Z_{2N-2:N}$	Z_{N-1}	$Z_{N-2:0}$	Offset
Rotate Right	$A_{N-2:0}$	A_{N-1}	$A_{N-2:0}$	k
Logical Right	0	A_{N-1}	$A_{N-2:0}$	k
Arithmetic Right	sign	A_{N-1}	$A_{N-2:0}$	k
Rotate Left	$A_{N-1:1}$	A_0	$A_{N-1:1}$	$\bar{k}$
Logical/Arithmetic Left	$A_{N-1:1}$	A_0	0	$\bar{k}$

Array Funnel Shifter

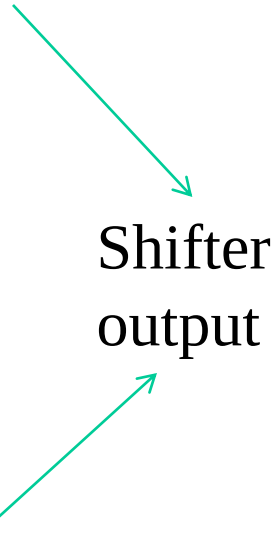
- N N-input multiplexers
  - Use 1-of-N hot (one multiplexer for each output bit) select signals for shift amount
  - nMOS pass transistor design (V_t drops!)
- The shift amount is conditionally inverted (for left shifts) and decoded into select signals that are fed vertically across the array. The outputs are taken horizontally.
- Each row of transistors attached to an output forms one of the multiplexers. The 2N – 1 inputs run diagonally to the appropriate mux inputs.



Array Funnel Shifter

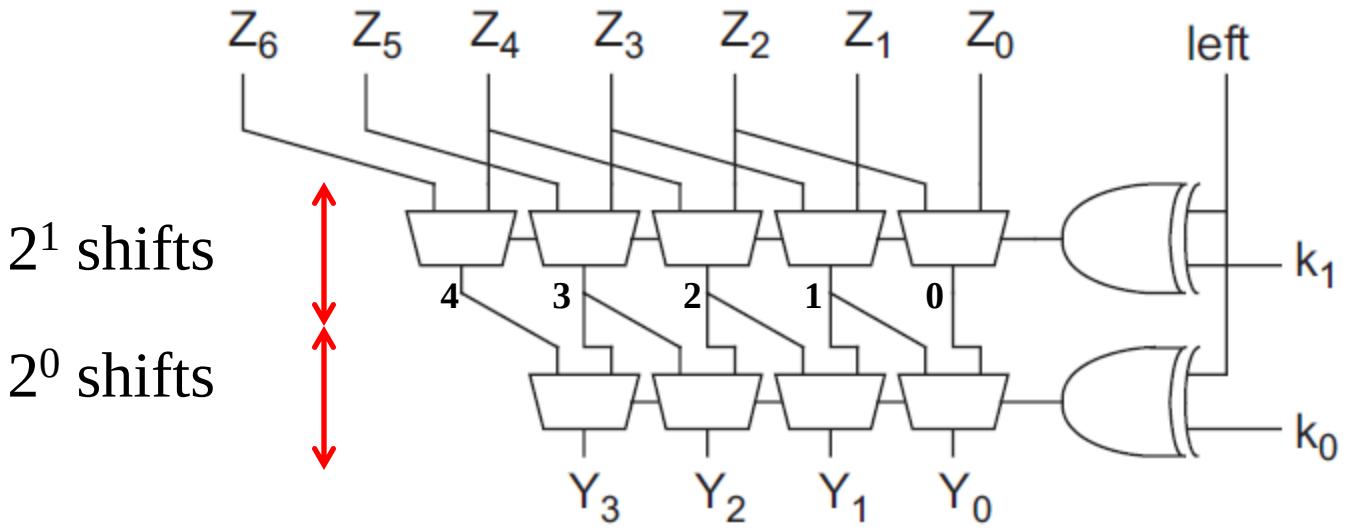
Shift Right	k	Z6	Z5	Z4	Z3	Z2	Z1	Z0
0	0	0	0	0	A3	A2	A1	A0
1	1	0	0	0	A3	A2	A1	A0
2	2	0	0	0	A3	A2	A1	A0
3	3	0	0	0	A3	A2	A1	A0

Shift Left	k	Z6	Z5	Z4	Z3	Z2	Z1	Z0
0	3	A3	A2	A1	A0	0	0	0
1	2	A3	A2	A1	A0	0	0	0
2	1	A3	A2	A1	A0	0	0	0
3	0	A3	A2	A1	A0	0	0	0



Logarithmic Funnel Shifter

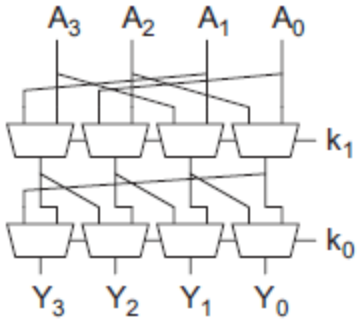
- Log N stages of 2-input muxes
  - No select decoding needed
  - The XOR gates on the control inputs conditionally invert the shift amount for left shifts.



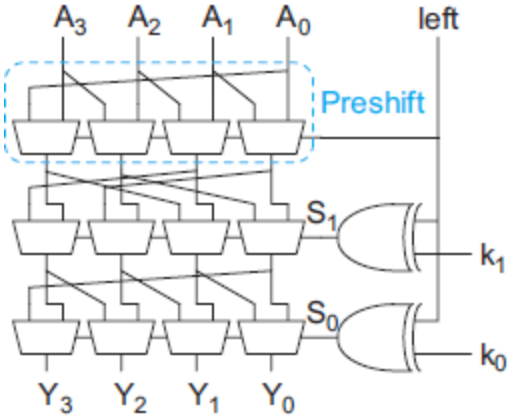
11.8.2 Barrel Shifter

- Barrel shifters perform right rotations using wrap-around wires.
  - Unlike funnel shifters, barrel shifters contain long wrap-around wires. In a large shifter, it is beneficial to upsize or buffer the drivers for these wires.
- Left rotations are right rotations by $N - k = \overline{k} + 1$ bits.
- Shifts are rotations with the end bits masked off.
- Barrel shifters come in array and logarithmic forms; we focus on logarithmic barrel.

Logarithmic Barrel Shifter

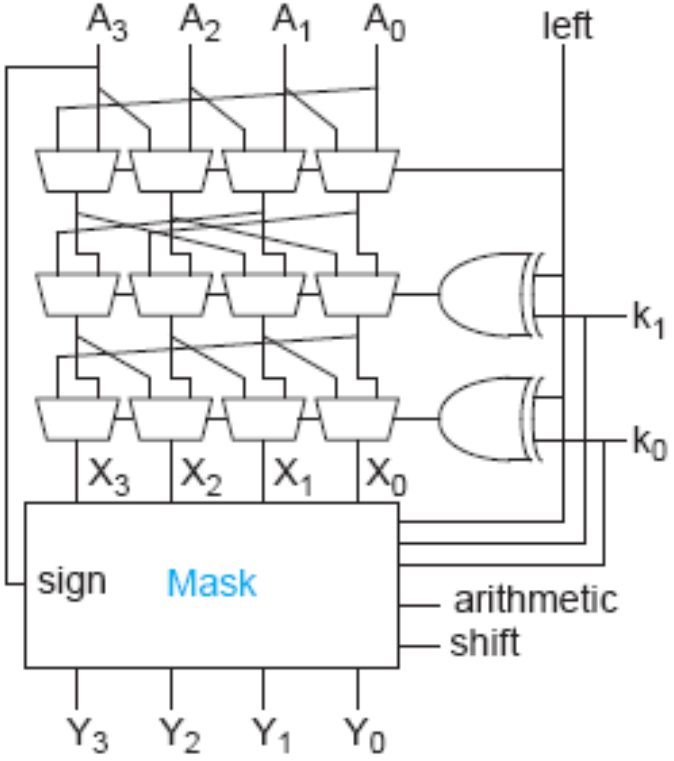


Right shift only



Right/Left shift

Rotate left by prerotating right by 1, then rotating right by ~*k*.

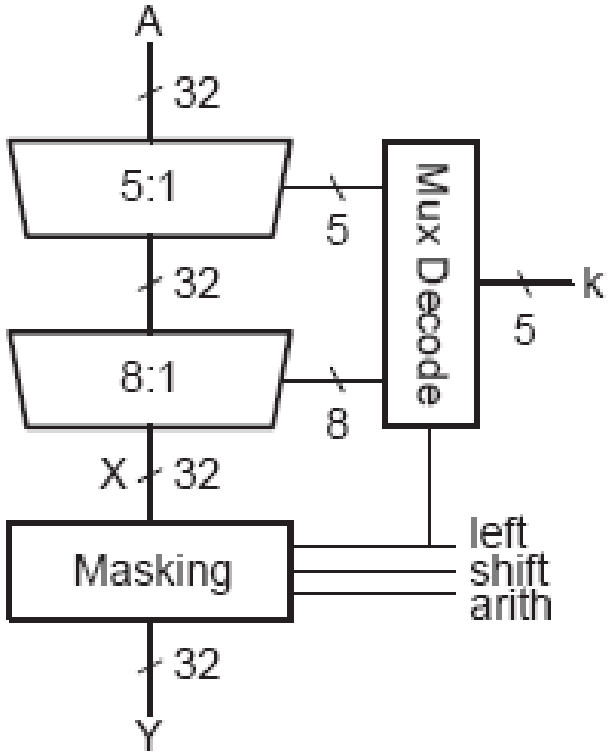


Right/Left Shift & Rotate

Performing logical or arithmetic shifts on a barrel shifter requires a way to mask out the bits that are rotated off the end of the shifter

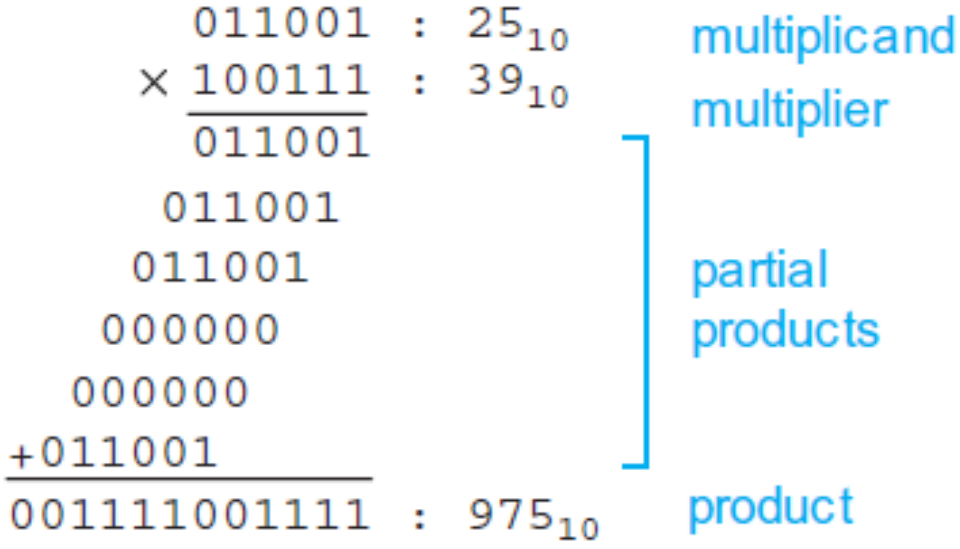
32-bit Logarithmic Barrel (read)

- Datapath never wider than 32 bits
- First stage preshifts by 1 to handle left shifts and for rotate rights by 0, 1, 2,3 or 4 bits.
- Second stage rotate rights by by 0, 4, 8, 12 16, 20, 24 or 28 bits
- The masking unit generates an N-bit mask with ones.



11.9 Multiplication

- Example:

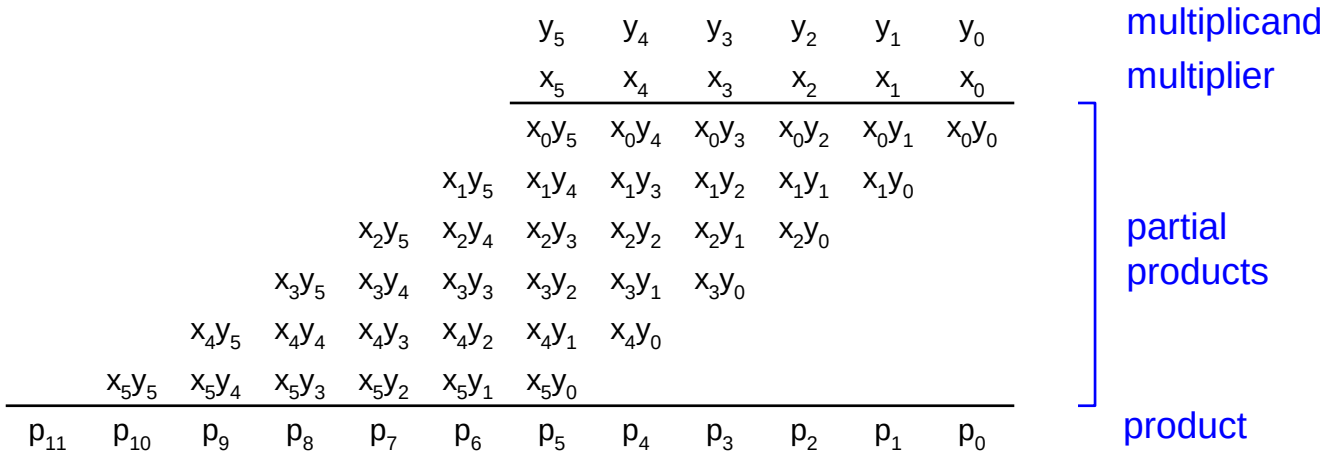


- M x N-bit multiplication
  - Produce N partial products, each partial product is M-bit .
  - Sum these to produce (M+N)-bit product

General Form

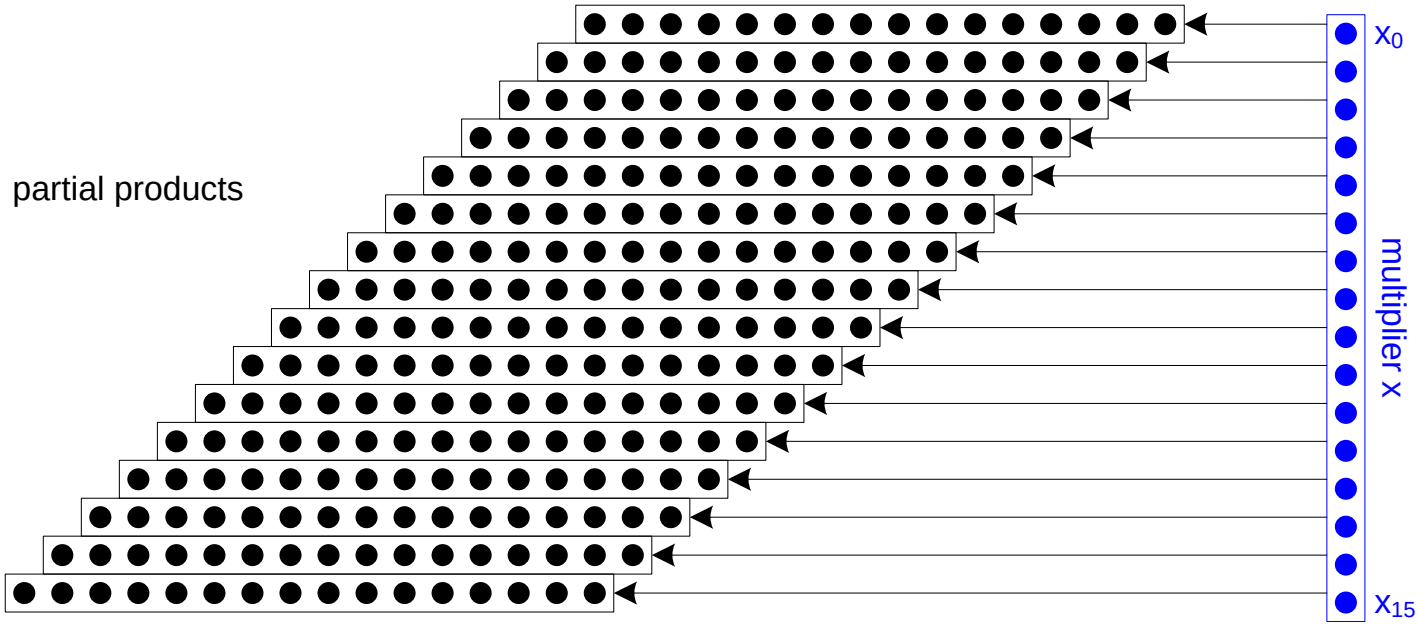
- Multiplicand: $Y = (y_{M-1}, y_{M-2}, \ldots, y_1, y_0)$
- Multiplier: $X = (x_{N-1}, x_{N-2}, \ldots, x_1, x_0)$
- Product:

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



Dot Diagram

- Large multiplications can be more conveniently illustrated using *dot diagrams.*
- Each dot represents a bit
- The partial products are represented by a horizontal boxed row of dots, shifted according to their weight.
- The multiplier bits used to generate the partial products are shown on the right.



Multiplication Techniques

- There are a number of techniques that can be used to perform multiplication. The choice is based upon factors such as latency, throughput, energy, area, and design complexity.
- One approach is to use an *M + 1-bit* carry-propagate adder (CPA) :
  - Add the first two partial products, then another CPA to add the third partial product to the running sum, and so forth.
  - Such an approach requires *N – 1 CPAs and is slow, even if a* fast CPA is employed.
- More efficient **parallel** approaches use some sort of **array or tree** of full adders to sum the partial products.

11.9.1 Array Multiplier

•The first row converts the first partial product into carry-save redundant form.

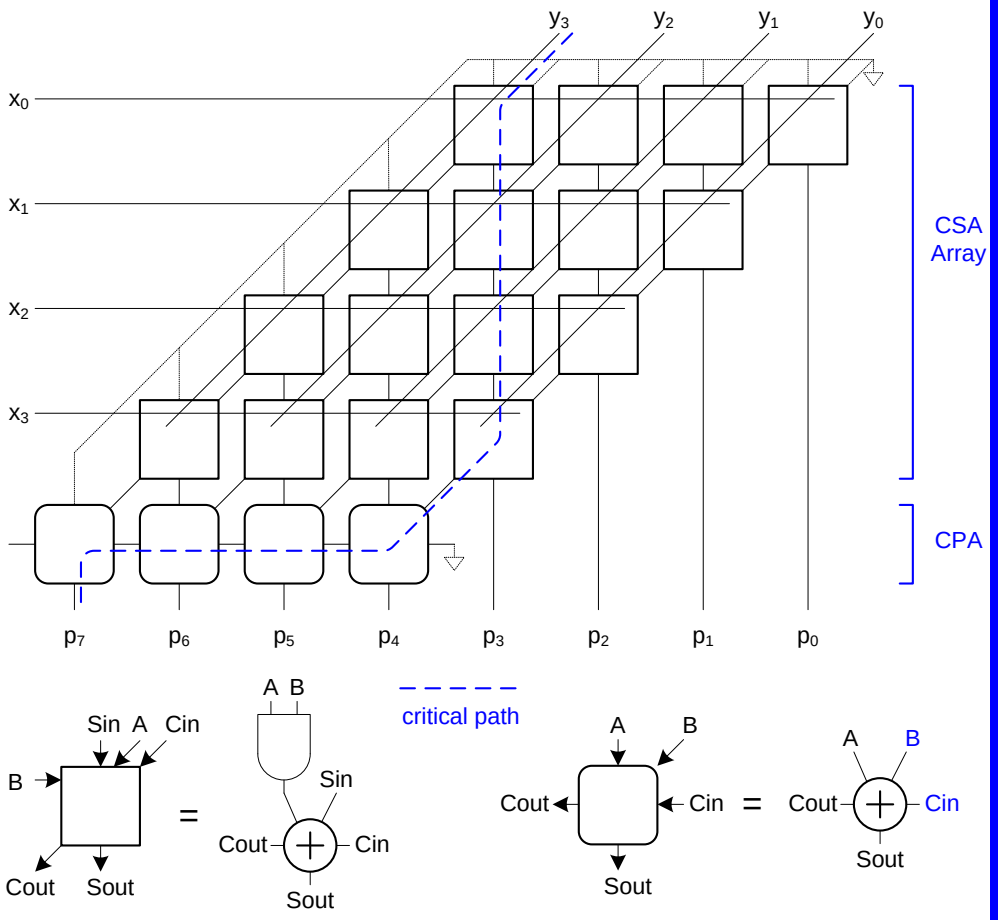
• Each later row uses the CSA to add the corresponding partial product to the carry-save redundant result of the previous row and generate a carry-save redundant result.

• Note that the first row of CSAs adds the first partial product to a pair of 0s.

• The least significant N output bits are available as sum outputs directly from CSAs. The most significant output bits arrive in carry-save redundant form and require an M-bit carry-propagate adder to convert into regular binary form.

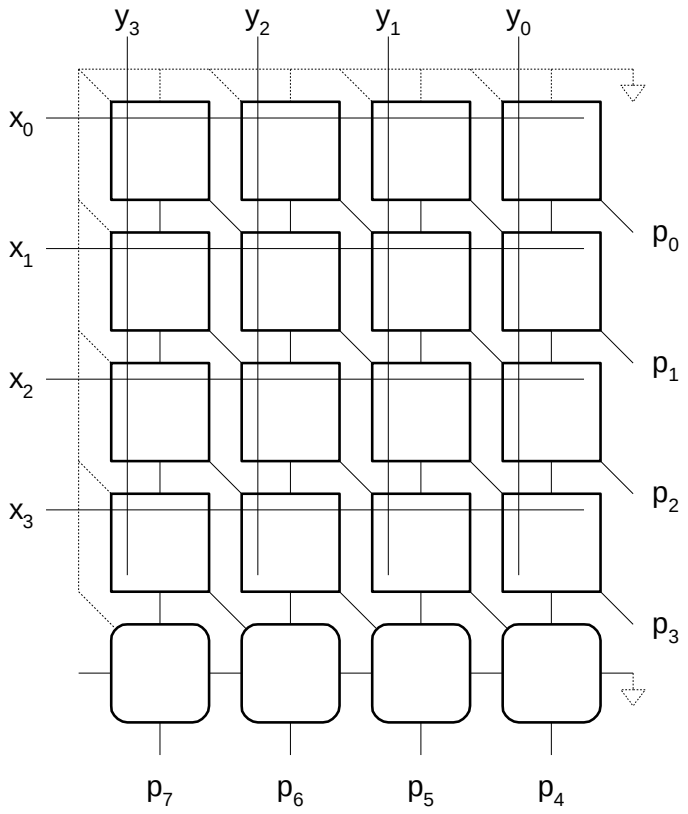
• The first row of CSAs can be used to add the first three partial products together. **Critical path:** ***N–2 CSAs and a CPA.***

•To improve speed: replace the bottom row with a faster CPA such as a lookahead or tree adder.



Rectangular Array

- Squash array to fit rectangular floorplan
- circuits are assigned rectangular blocks in the floorplan so the parallelogram shape wastes space.



11.9.3 Fewer Partial Products

- Array multiplier requires N partial products
- If we looked at groups of r bits, we could form N/r partial products.
  - Faster and smaller?
  - Called radix-2^r encoding
- Ex: r = 2: look at pairs of bits
  - Form partial products of 0, Y, 2Y, 3Y
  - First three are easy, but 3Y requires adder ☹

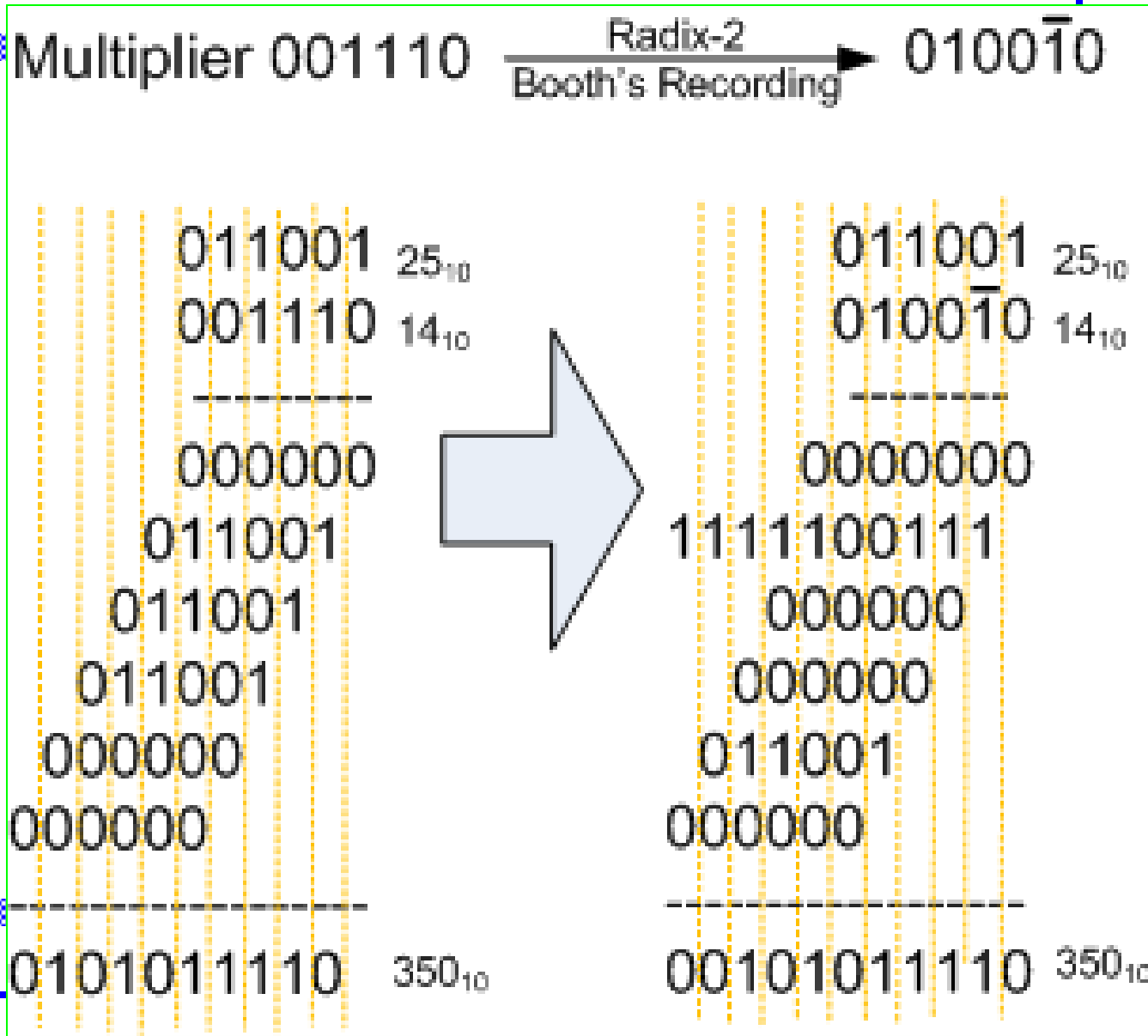
Radix-2 Booth's Recording

- In old (**serial**)multipliers, addition operations were very slow.
- Booth attempted to reduce the addition operations, because addition is slow.
  - He observed that when there are a large number of consecutive 1's in multiplier, the multiplication can be speeded up.
  - $2^j + 2^{j-1} + \ldots + 2^i = 2^{j+1} - 2^i$
  - Example:
    - 01110 = 10000 - 000010 = $2^6 - 2^1$
    - 14 = 16 - 2
  - The larger the sequence of 1's, the larger the savings
- Booth Recording converts the binary number represented by the set [0,1] to binary signed-digit number using the digit set [-1, 1, **0**].
- Radix-2 booth encoding examines x_i and x_{i-1} to generate the encoding y_i.

Xi	Xi-1	Yi	Explanation
0	0	0	No string of one's insight
0	1	1	End of string of ones
1	0	-1	Beginning of string of ones
1	1	0	Continuation of string of ones

Radix-2 Booth's Recording Example

- Regular multiplication: 3 additions
- Booth: one addition and one subtraction



11.9.3 Booth Encoding: Radix-4

- Radix-4 multiplier produces N/2 partial products.
- Each partial product is 0, Y, 2Y, or 3Y, depending on a pair of bits of X.
  - Computing 2Y is a simple shift,
  - but 3Y is a hard multiple requiring a slow carry propagate addition of Y + 2Y before partial product generation begins.
- Booth encoding was originally proposed to accelerate **serial multiplication**.
- Modified Booth encoding allows higher radix parallel operation without generating the hard 3Y multiple by instead using negative partial products.
- Observe that
  - 3Y = 4Y – Y
  - 2Y = 4Y – 2Y
  - 4Y in a radix-4 multiplier array is equivalent to Y in the next row of the array that carries four times the weight.

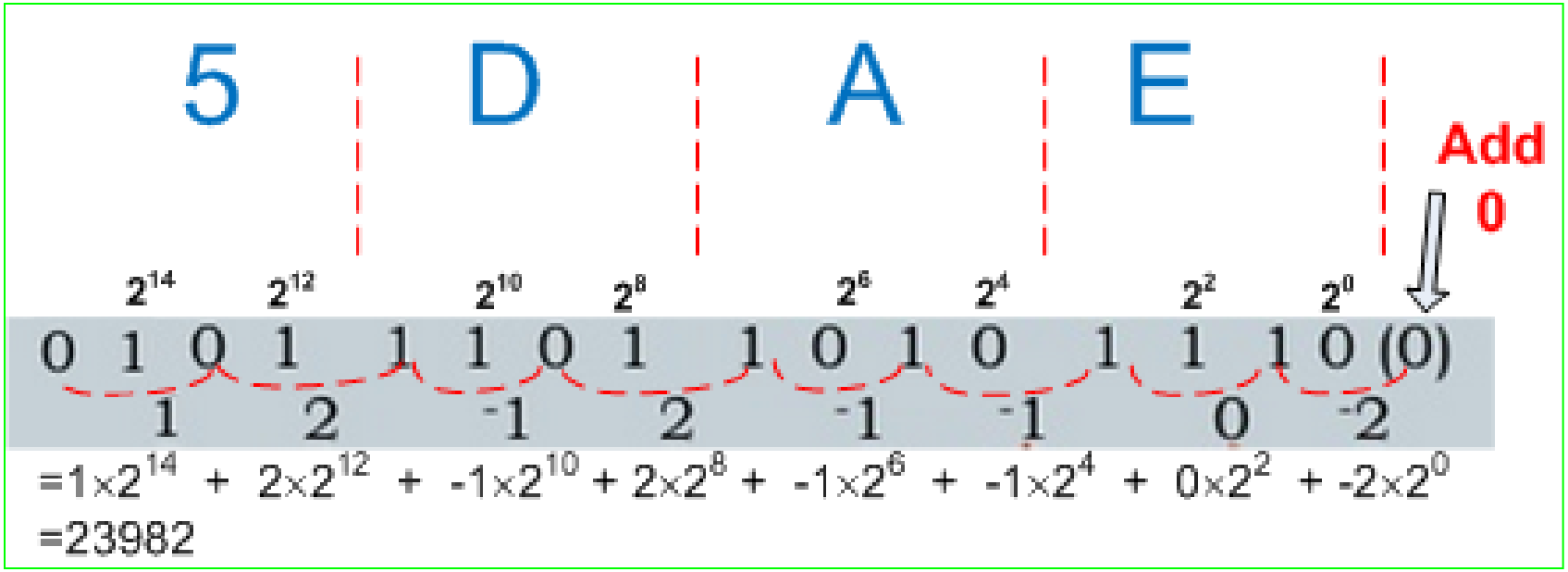
Radix-4 Modified Booth

Inputs			Partial Product	
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	**Explanation**
0	0	0	0	No string of 1's
0	0	1	Y	End of string of 1's
0	1	0	Y	Isolated 1
0	1	1	$2Y$	End of string of 1's
1	0	0	$-2Y$	Beginning of string of 1's
1	0	1	$-Y$	End string, begin new string
1	1	0	$-Y$	Beginning of string of 1's
1	1	1	$-0\ (=0)$	Continuation of string of 1's

Radix-4 Booth Encoding Example

- Compute the Booth's Encoding for the 16-bit unsigned number: 5DAE

Inputs			Partial Product
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i
0	0	0	0
0	0	1	Y
0	1	0	Y
0	1	1	$2Y$
1	0	0	$-2Y$
1	0	1	$-Y$
1	1	0	$-Y$
1	1	1	$-0\ (=0)$

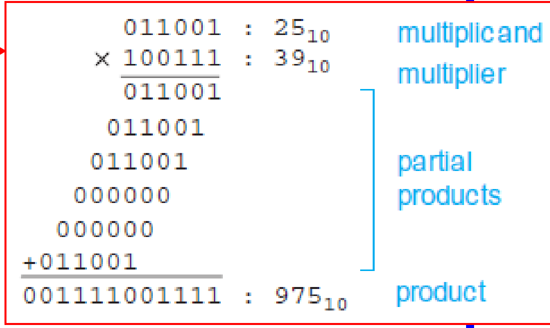


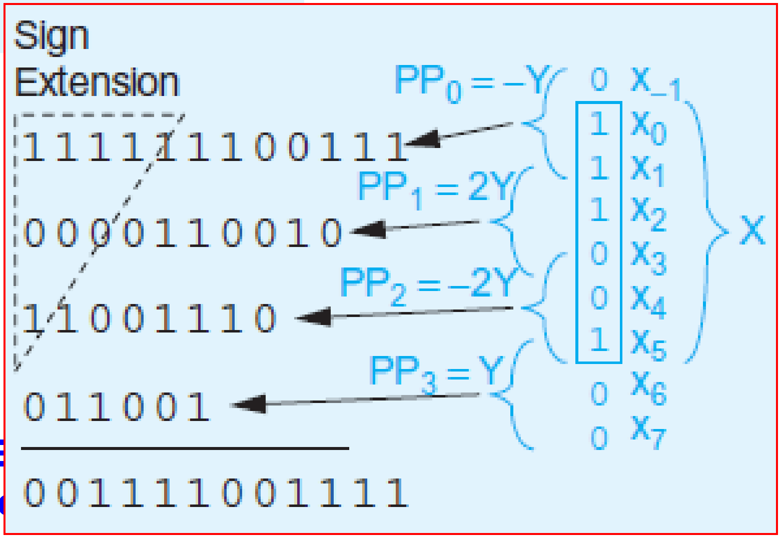
Radix-4 Booth Encoding Example

Example 11.3

Repeat the multiplication of $P = Y \times X = 011001_2 \times 100111_2$ from Figure 11.71, applying Booth encoding to reduce the number of partial products.

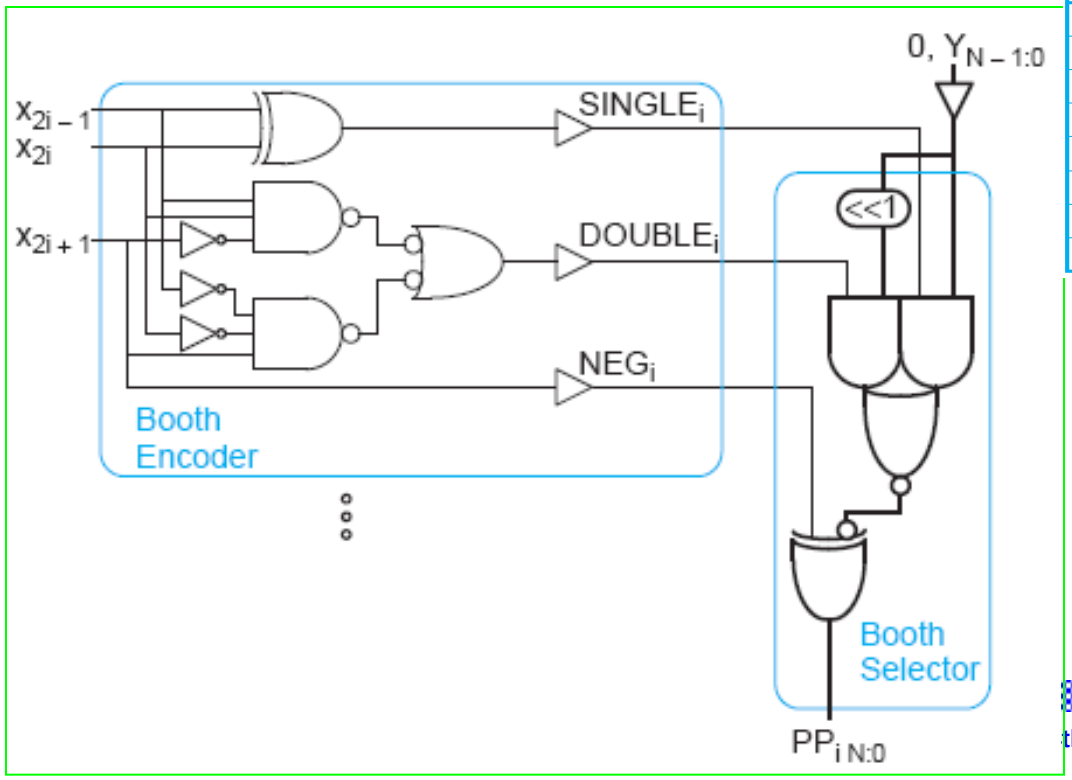
SOLUTION: Figure 11.79 shows the multiplication. X is written vertically and the bits are used to select the four partial products. Each partial product is shifted two columns left of the previous one because it has four times the weight. The upper bits are sign-extended with 1s for negative partial products and 0s for positive partial products. The partial products are added to obtain the result.





Booth Hardware

- Booth encoder generates control lines for each PP
  - Booth selectors choose PP bits

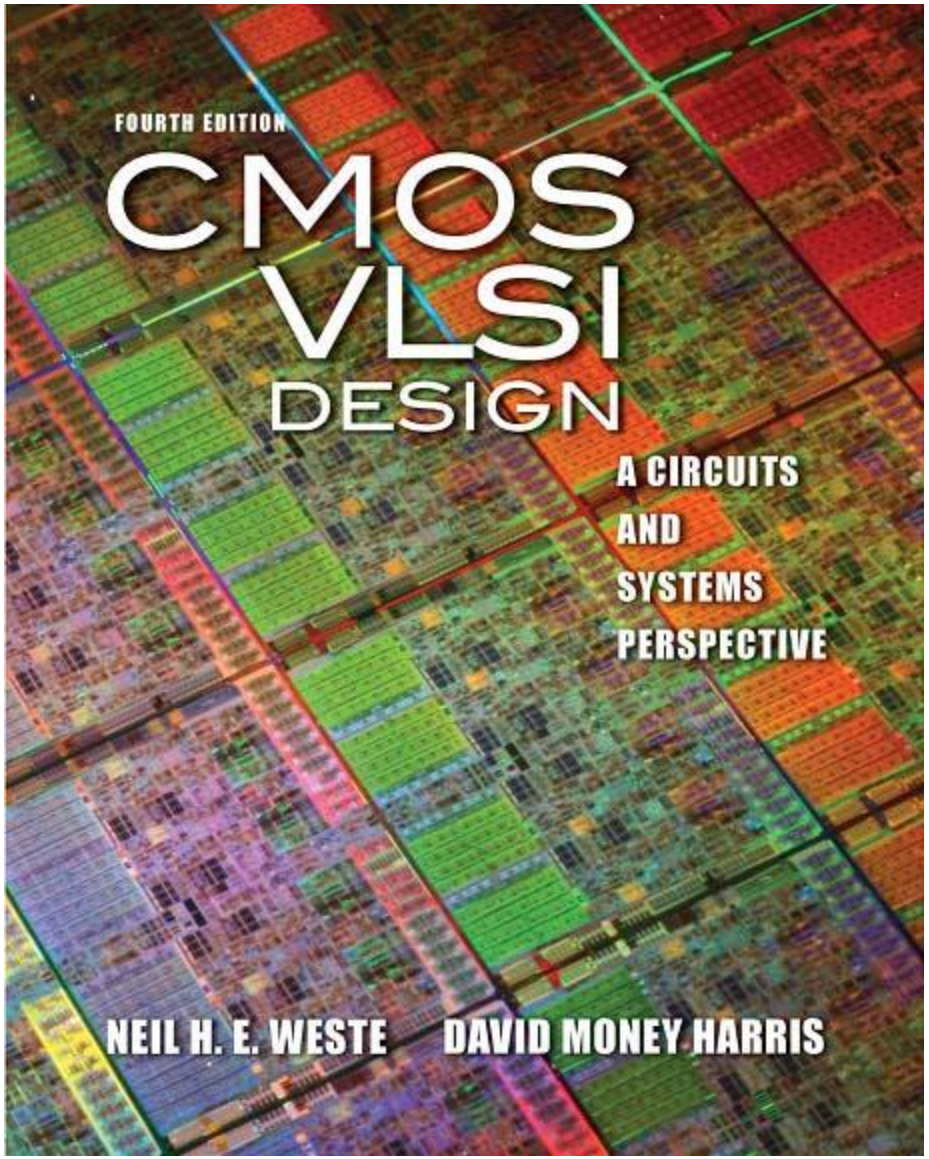


Inputs			Partial Product	Booth Selects		
x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	$SINGLE_i$	$DOUBLE_i$	NEG_i
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0	Y	1	0	0
0	1	1	$2Y$	0	1	0
1	0	0	$-2Y$	0	1	1
1	0	1	$-Y$	1	0	1
1	1	0	$-Y$	1	0	1
1	1	1	$-0\ (=0)$	0	0	1

CPE 110408423
VLSI Design
Chapter 12: Array Subsystems

Bassam Jamil

[Computer Engineering Department,
Hashemite University]

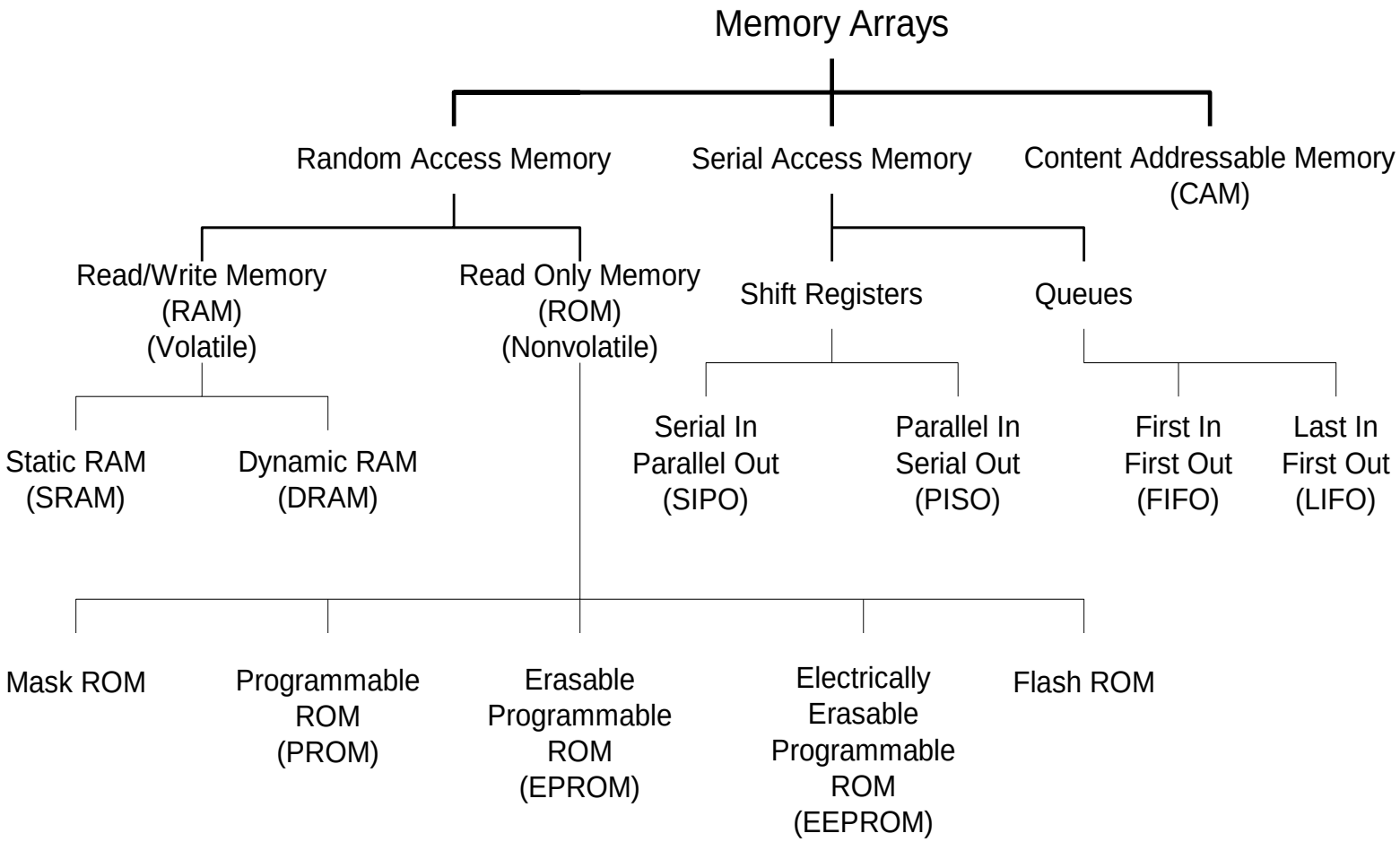


Lecture 3: SRAM

Outline

- Memory Arrays
- SRAM Architecture
  - SRAM Cell
  - Decoders
  - Column Circuitry
  - Multiple Ports

12.1 Memory Arrays



Memory Arrays

- *Random access memory (RAM)* is accessed with an address and has a latency independent of the address.
- *Serial access memories (SAM)* are accessed sequentially so no address is necessary.
- *Content addressable memories (CAM)* determine which address(es) contain data that matches a specified key.
- Volatile vs. nonvolatile memory.
  - Volatile memory retains its data as long as power is applied,
  - nonvolatile memory will hold data indefinitely.
  - RAM is synonymous with volatile memory, while ROM is synonymous with nonvolatile memory.

Volatile Memory

- The memory cells used in volatile memories can be divided into **static** structures and **dynamic** structures.
- **Static cells** use some form of feedback to maintain their state,
- **Dynamic cells** use charge stored on a floating capacitor through an access transistor.
  - Charge will leak away through the access transistor even while the transistor is OFF,
  - So dynamic cells must be periodically read and rewritten to refresh their state.
- Static RAMs (SRAMs) are **faster** and **less troublesome**, but require **more area per bit** than their dynamic counterparts (DRAMs).

Non-Volatile Memory

- **Volatility:**
  - Some nonvolatile memories are read-only (e.g. mask ROM), but many nonvolatile memories can be written, albeit more slowly than their volatile memories.
- **Types of non-volatile memories:**

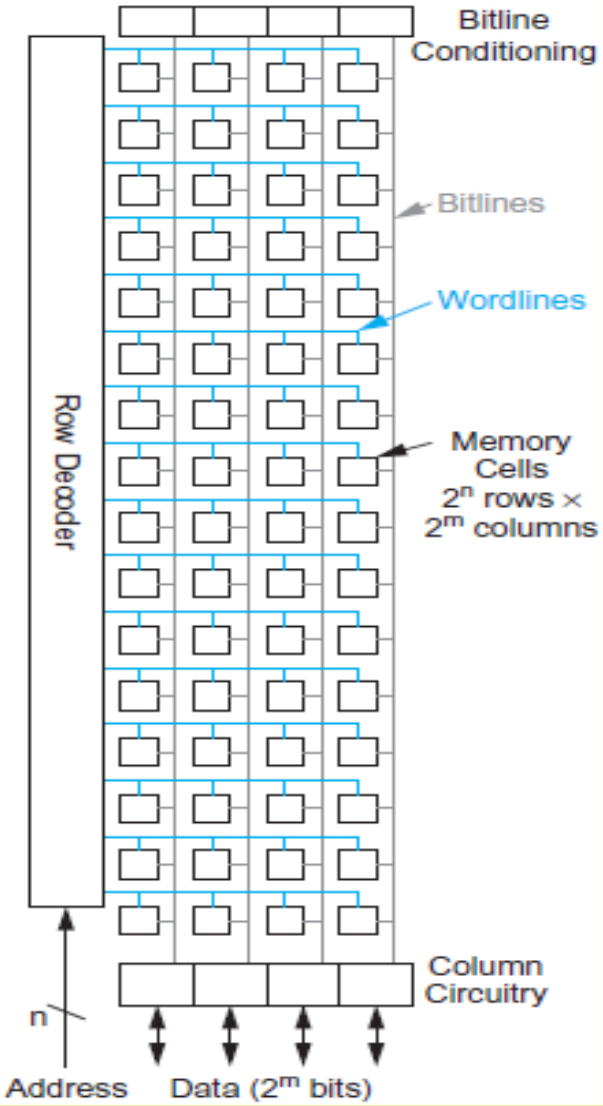
1. **Mask ROM**: the contents of a mask ROM are **hardwired** during fabrication and cannot be changed.
2. **Programmable ROM (PROM)** is **programmed** once after fabrication by blowing on-chip fuses with a special high programming voltage.
3. **Erasable programmable ROM (EPROM)** is programmed by storing charge on a floating gate.
   - It can be erased by exposure to ultraviolet (UV) light for several minutes to knock the charge off the gate. Then the EPROM can be reprogrammed.
4. **Electrically erasable programmable ROMs (EEPROMs)** are erased in microseconds with on-chip circuitry.
5. **Flash** memories are a variant of EEPROM that erases entire blocks rather than individual bits.
   - Sharing the erase circuitry across larger blocks reduces the area per bit.
   - Because of their good density and easy in-system reprogrammability, Flash memories have replaced other nonvolatile memories in most modern CMOS systems.

Memory Array Architecture

- Memory cells can have one or more ports for access.
- On a read/write memory, each port can be read-only, write-only, or both.
- A memory array contains 2^n words of 2^m bits each.
  - Total number of bits = $2^n \times 2^m$ bits
- memory array containing 16 4-bit words (n = 4, m = 2).
  - Total number of bits = 64 bits

Memory Array :One row per word and One column per bit

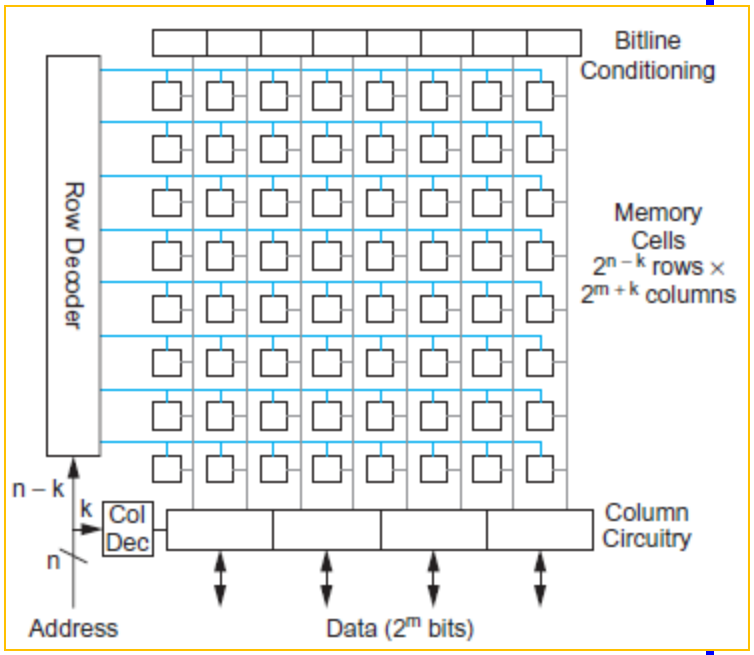
- The simplest design with one row per word and one column per bit.
- The **row decoder** uses the address to activate one of the rows by asserting the **wordline**.
- During a read operation, the cells on this wordline drive the **bitlines**.
- The **column** circuitry may contain amplifiers or buffers to sense the data.
- A typical memory array may have thousands or millions of words of only 8–64 bits each, which would lead to a tall, skinny layout that is hard to fit in the chip floorplan and slow because of the long vertical wires.
- Therefore, the array is often folded into fewer rows of more columns. After folding, each row of the memory contains 2^k words, so the array is physically organized as 2^{n-k} rows of 2^{m+k} columns or bits.



16 4-bit words:
(n = 4, m = 2, k=0)

Memory Array : Two-way Fold

- The array is physically organized as 2^{n-k} rows of 2^{m+k} columns or bits.
- The Figure shows a two-way fold (k = 1) with eight rows and eight columns.
- The column decoder controls a multiplexer in the column circuitry to select 2^m bits from the row as the data to access.
- Larger memories are generally built from multiple smaller sub-arrays so that the wordlines and bitlines remain reasonably short, fast, and low in power dissipation.



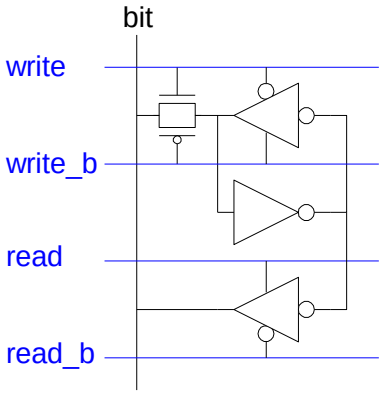
16 4-bit words:
(n = 4, m = 2, k=1)

12.2 SRAM

- Static RAMs use a memory cell with internal feedback that retains its value as long as power is applied. It has the following attractive properties:
  - Denser than flip-flops
  - Compatible with standard CMOS processes
  - Faster than DRAM
  - Easier to use than DRAM
- SRAMs are widely used i: caches, register files, tables and buffers.

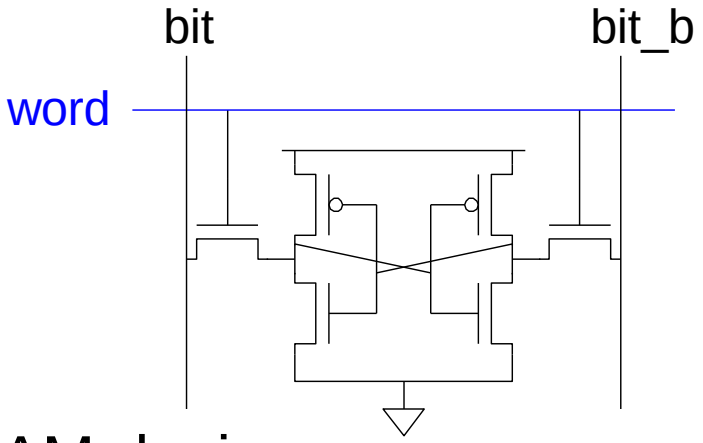
SRAM Cell: 12T SRAM Cell

- Basic building block: SRAM Cell
  - **Holds** one bit of information, like a latch
  - Must be **read and written**
- 12-transistor (12T) SRAM cell
  - Use a simple latch connected to bitline
  - Large area, so it is not used.
- Cell size accounts for most of array size
  - Reduce cell size at expense of complexity
  - The small cell size also offers shorter wires and hence lower dynamic power consumption.



SRAM Cell: 6T SRAM Cell

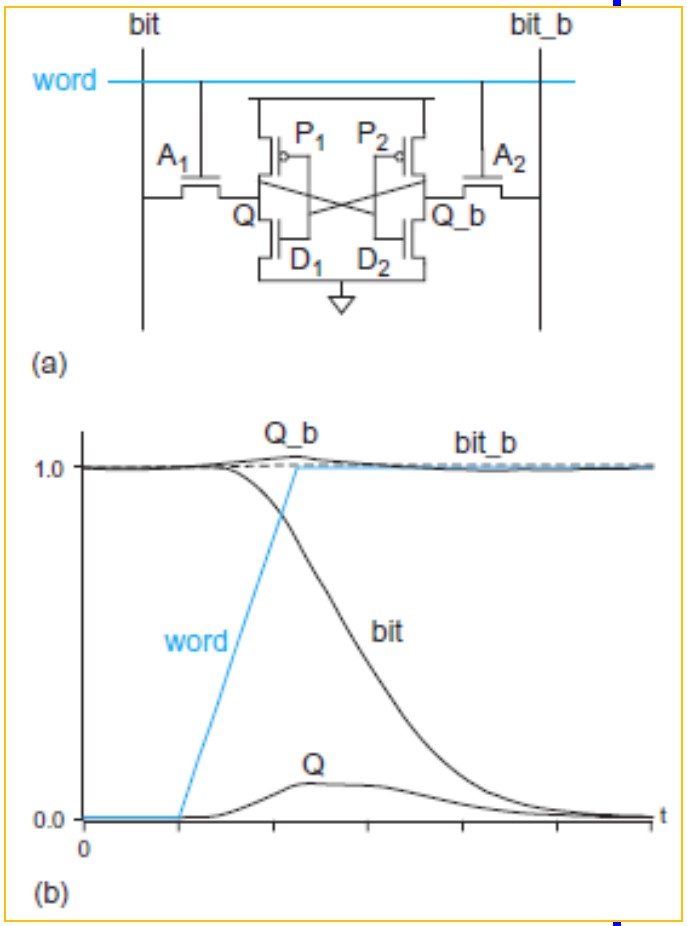
- 6T SRAM Cell
  - Used in most commercial chips
  - Data stored in cross-coupled inverters
- Read:
  - Precharge bit, bit_b
  - Raise wordline
- Write:
  - Drive data onto bit, bit_b
  - Raise wordline



- The central challenges in SRAM design are
  - minimizing its size
  - ensuring that the circuitry holding the state is weak enough to be overpowered during a write, yet strong enough not to be disturbed during a read.

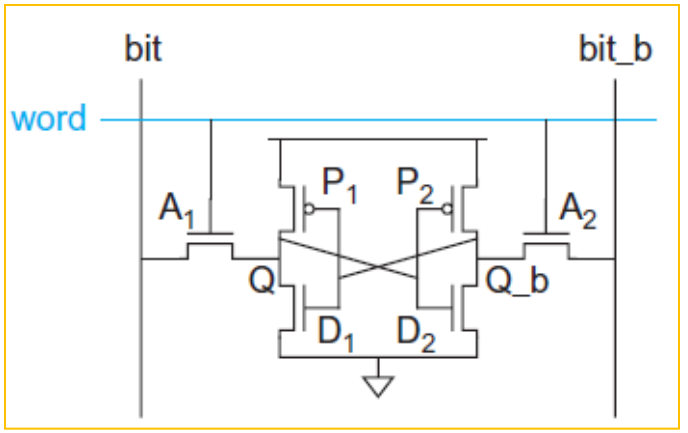
12.2.1.1 Read Operation

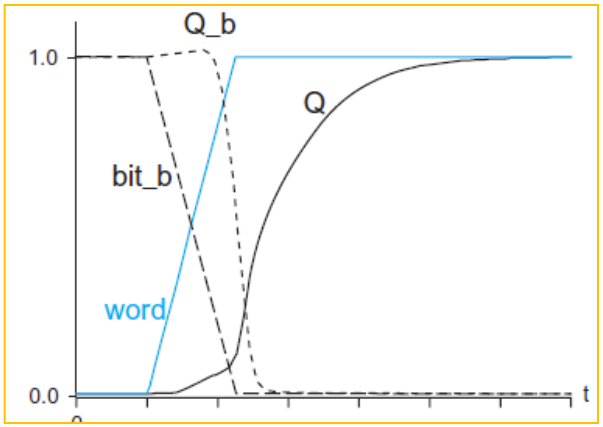
- Read operation:
  - Precharge both bitlines high
  - Then turn on wordline
  - One of the two bitlines will be pulled down by the cell
- Ex: Q= 0, Q_b = 1
  - bit discharges, bit_b stays high
  - But Q bumps up slightly
- *Read stability*
  - P2/D2 must not flip
  - D1 >> A1



12.2.1.2 Write Operation

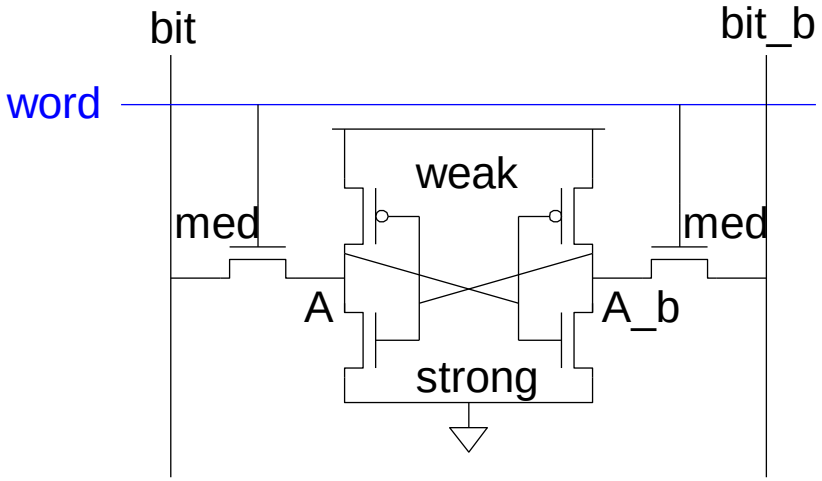
- Write operation:
  - Drive one bitline high, the other low
  - Then turn on wordline
  - Bitlines overpower cell with new value
- Ex: Q = 0, Q_b = 1, bit = 1, bit_b = 0
  - Force Q_b low, then Q rises high
- *Writability*
  - Must overpower feedback inverter
  - A2 >> P2





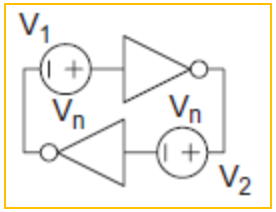
12.2.1.3 Cell Stability

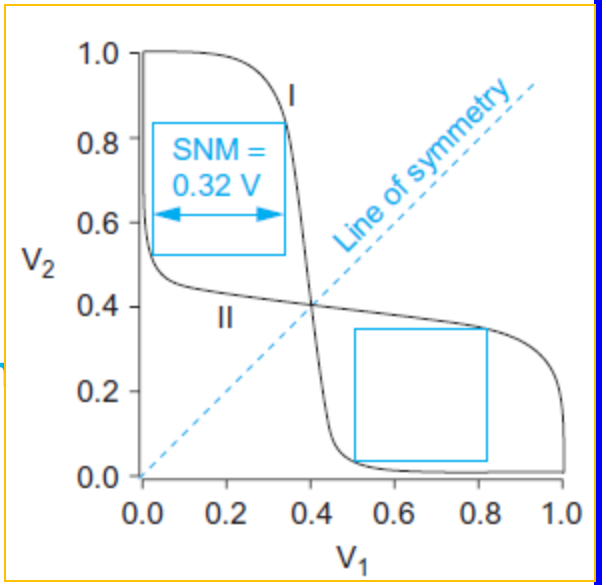
- **Read stability:**
  - The nMOS pulldown transistor in the cross-coupled inverters must be strongest.
- **Writibility:**
  - the access transistors are of intermediate strength, and the pMOS pullup transistors must be weak.



Static Noise Margin

- The *static noise margin (SNM)* measures how much noise can be applied to the inputs of the two cross-coupled inverters before a stable state is lost (during hold or read) or a second stable state is created (during write).
- The static noise margin can be determined graphically from a butterfly diagram shown in Figure.
- The butterfly plot shows two stable states (with one output low and the other high) and one metastable state (with V1 = V2). A positive value of noise shifts curve I left and curve II up. Excessive noise eliminates the stable state of V1 = 0 and V2 = VDD, forcing the cell into the opposite state.
- **The static noise margin is determined by the length of the side of the largest square that can be inscribed between the curves.**

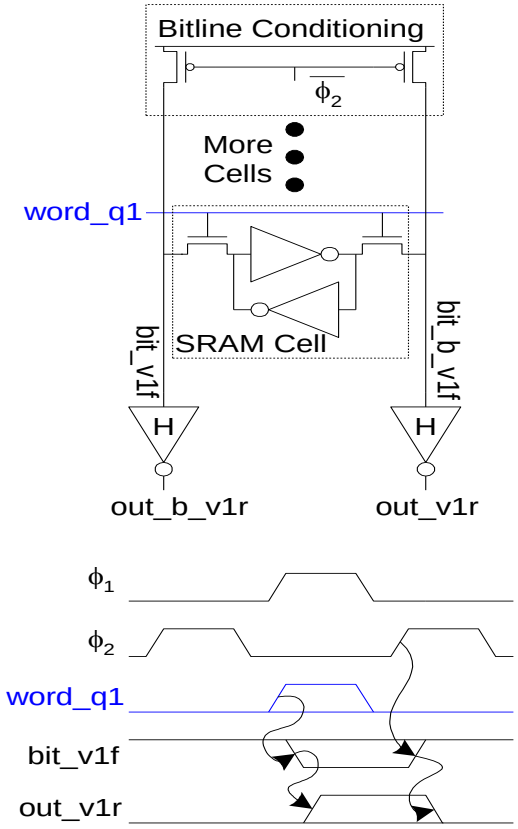




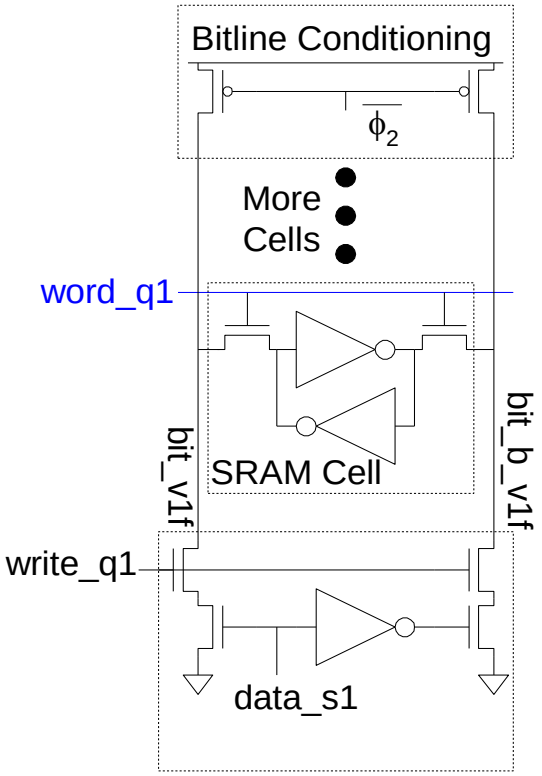
REST of the SLIDES Are reading material

SRAM Column Example

Read

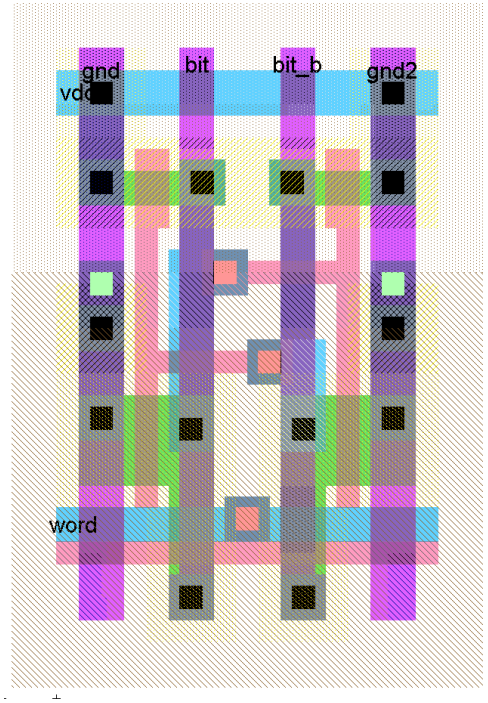


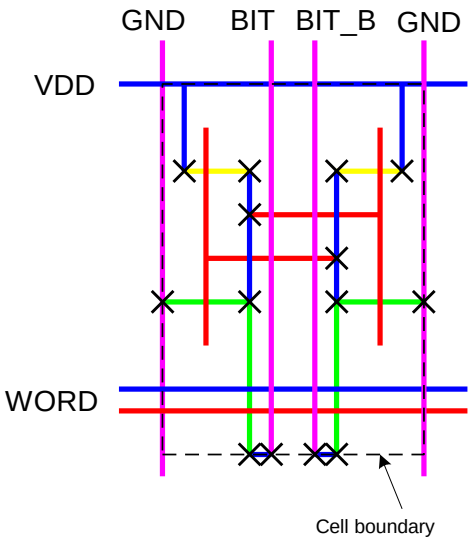
Write

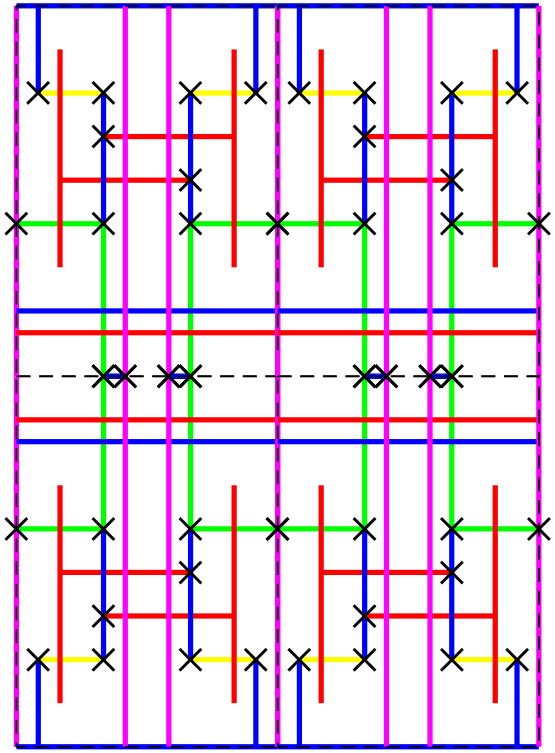


SRAM Layout: Physical Design

- Cell size is critical (left): 26 x 45 λ (smaller in industry)
- Tile cells sharing V_{DD}, GND, bitline contacts (6T:right).

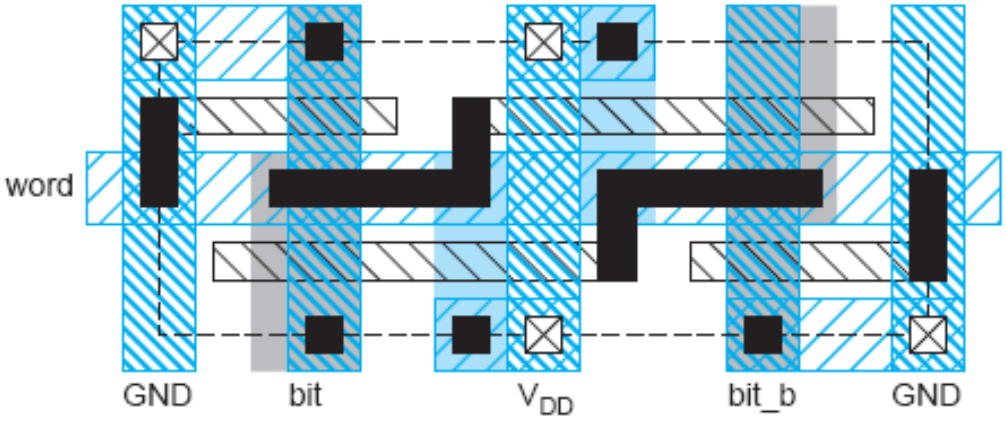






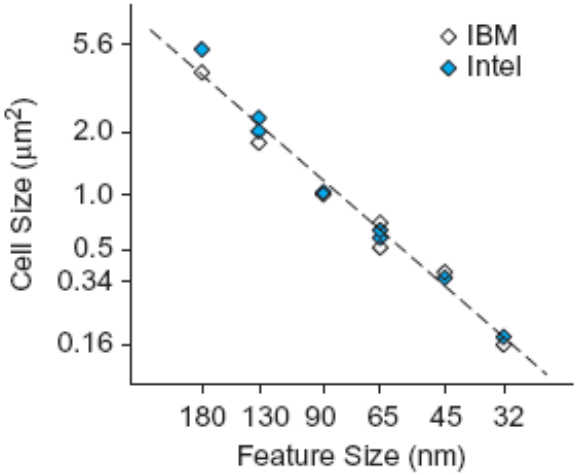
Thin Cell

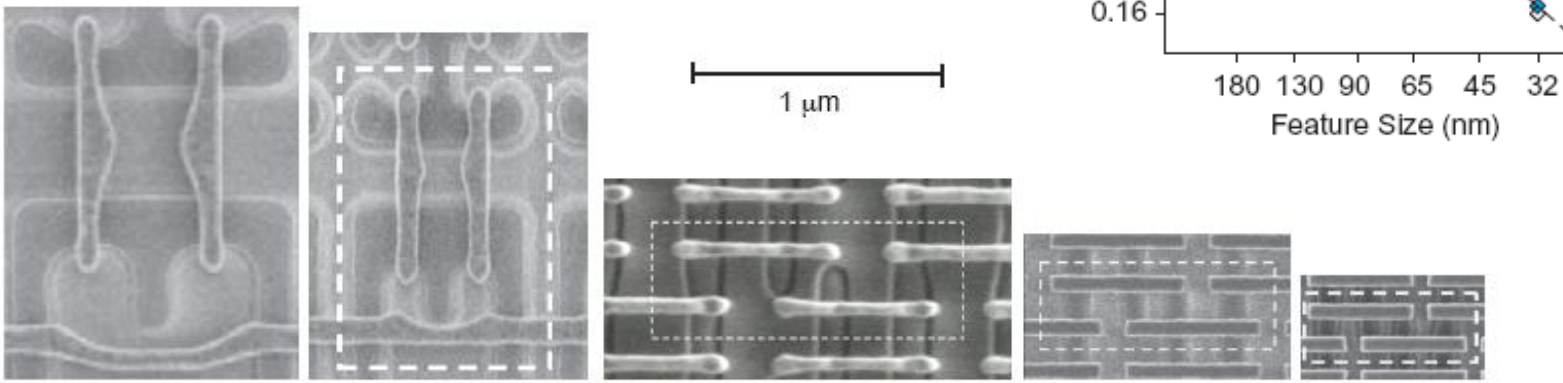
- In nanometer CMOS
  - Avoid bends in polysilicon and diffusion
  - Orient all transistors in one direction
- *Lithographically friendly* or *thin cell* layout fixes this
  - Also reduces length and capacitance of bitlines



Commercial SRAMs

- Five generations of Intel SRAM cell micrographs.
  - Transition to thin cell at 65 nm
  - Steady scaling of cell area



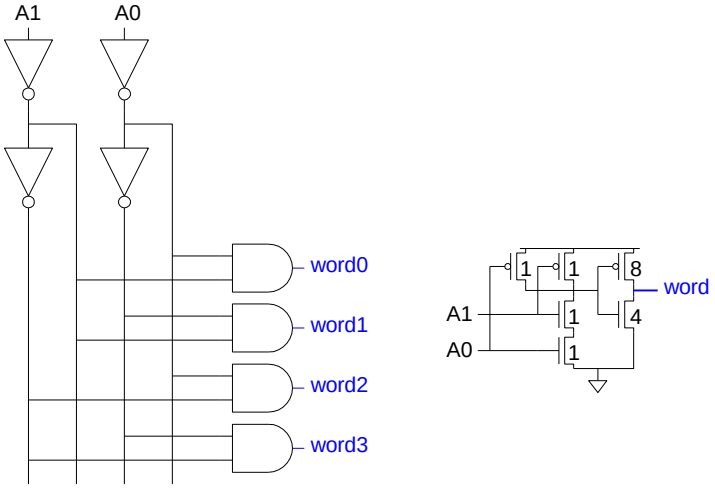


130 nm [Tyagi00] 90 nm [Thompson02] 65 nm [Bai04] 45 nm [Mistry07] 32 nm [Natarajan08]

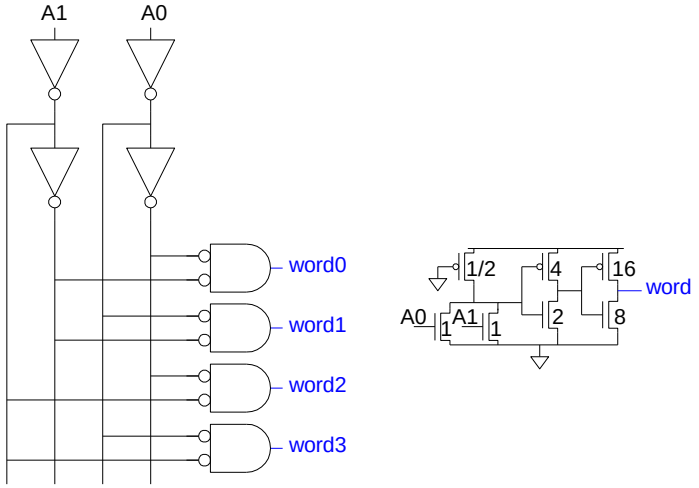
Row Circuitry: Decoders

- Row circuitry consists of decoder & worldline drivers
- n:2^n decoder consists of 2^n n-input AND gates
  - One needed for each row of memory
  - Build AND from NAND or NOR gates

Static CMOS

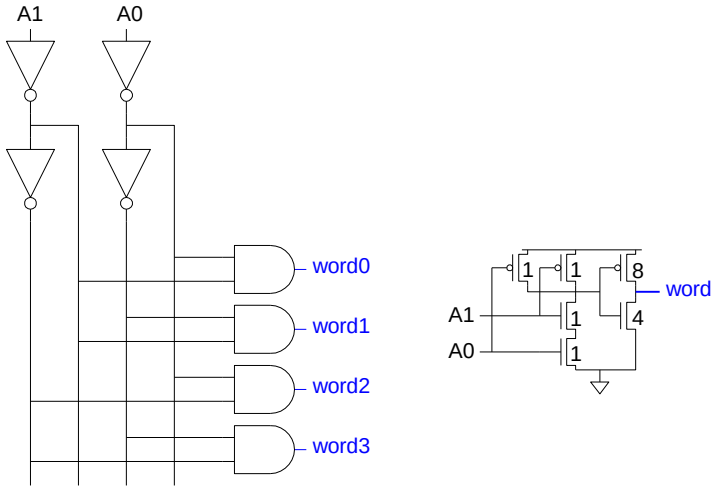


Pseudo-nMOS

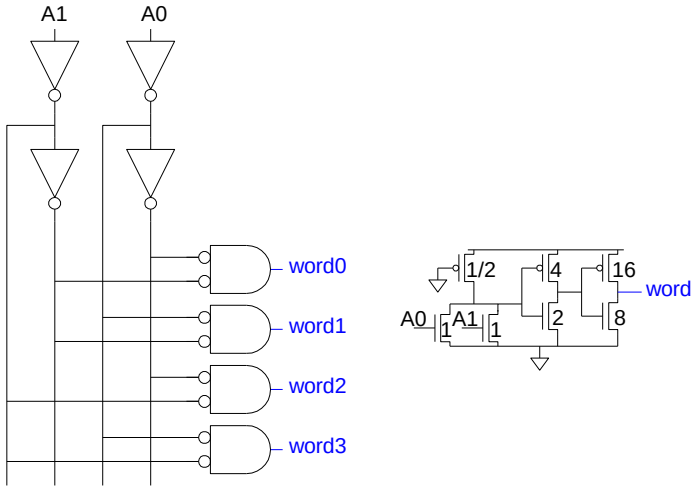


Row Circuitry: Decoders

Static CMOS



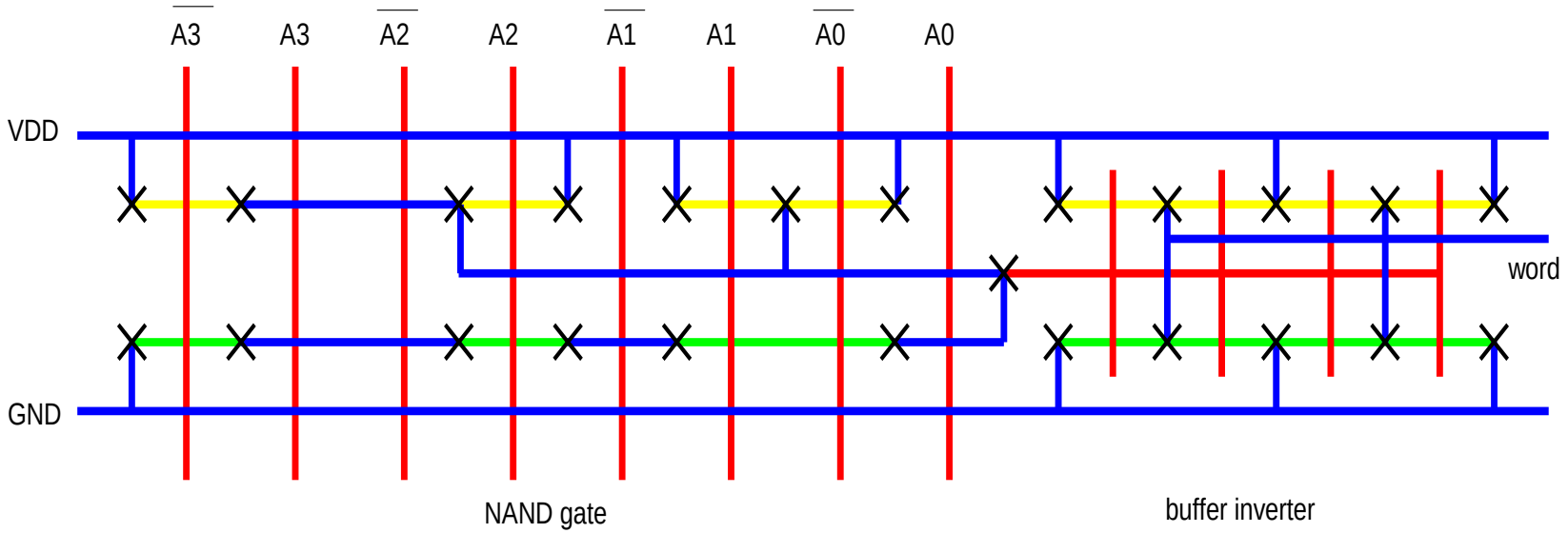
Pseudo-nMOS



- Wordline must be qualified with clock (shared clock and transistors).

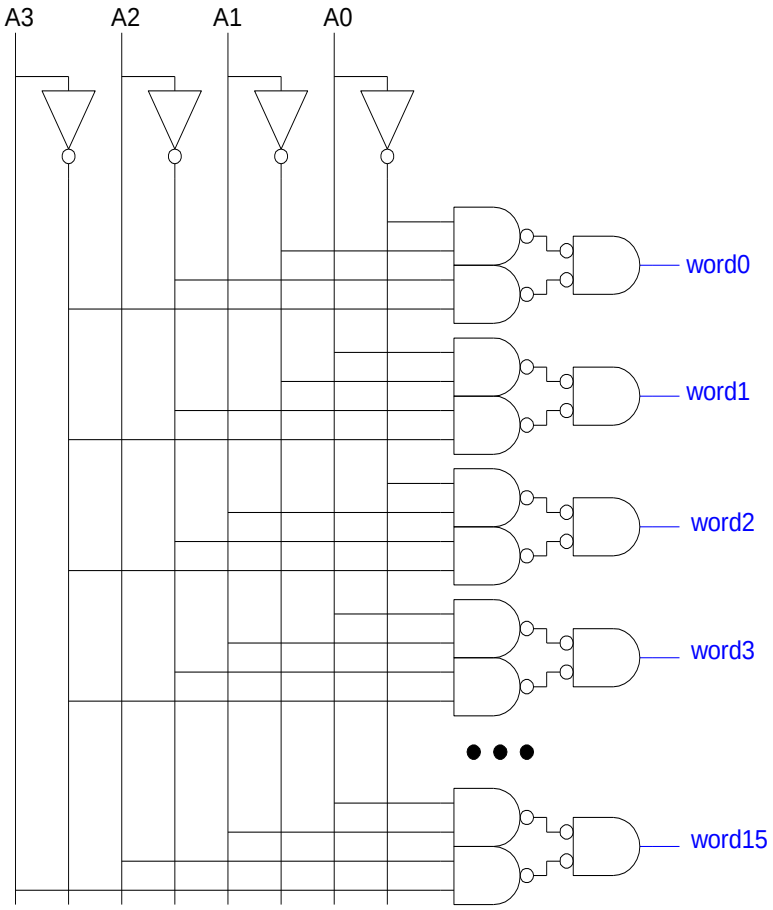
Decoder Layout

- Decoders must be pitch-matched to SRAM cell: height of decoder gate must match height of the row it drives.
  - Requires very skinny gates



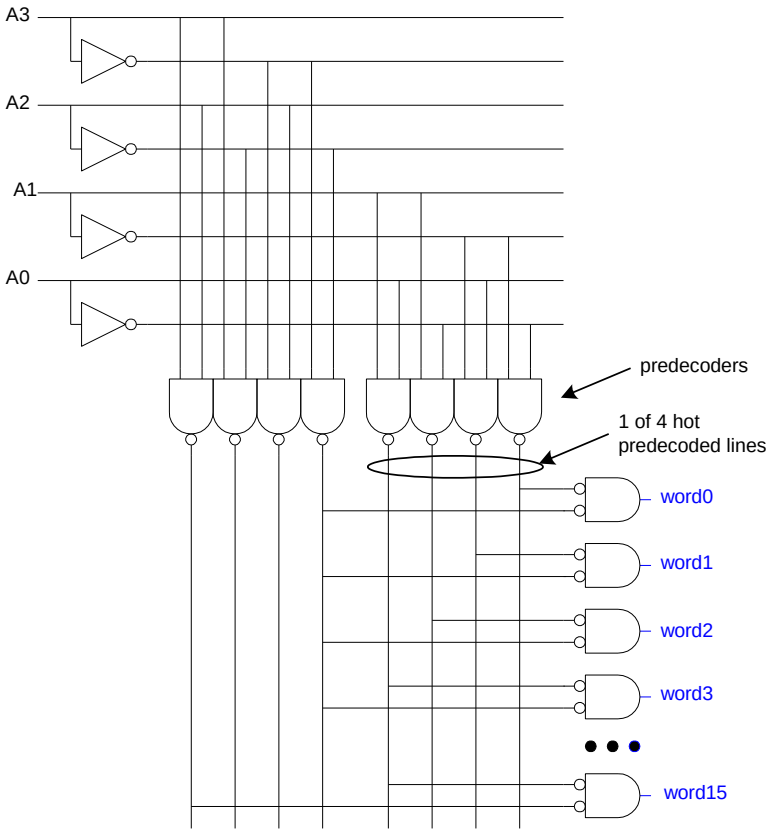
Large Decoders

- For n > 4, NAND gates become slow.
  - Break large gates into multiple smaller gates



Predecoding

- Many of these gates are redundant
  - Factor out common gates into predecoder
  - Saves area
  - Same path effort

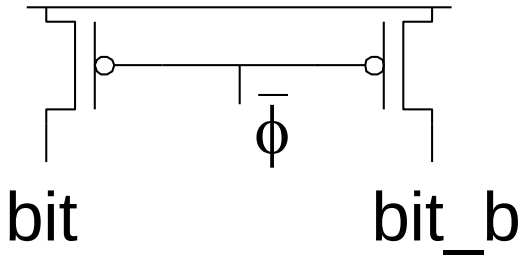


Column Circuitry

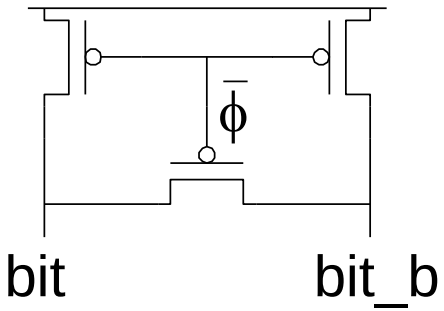
- Some circuitry is required for each column
  - Bitline conditioning
  - Write driver
  - Bitline sense amplifiers
  - Column multiplexing

Bitline Conditioning

- Precharge bitlines high before reads or writes



- Equalize bitlines to minimize voltage difference when using sense amplifiers



Bitline Sense Amplifiers

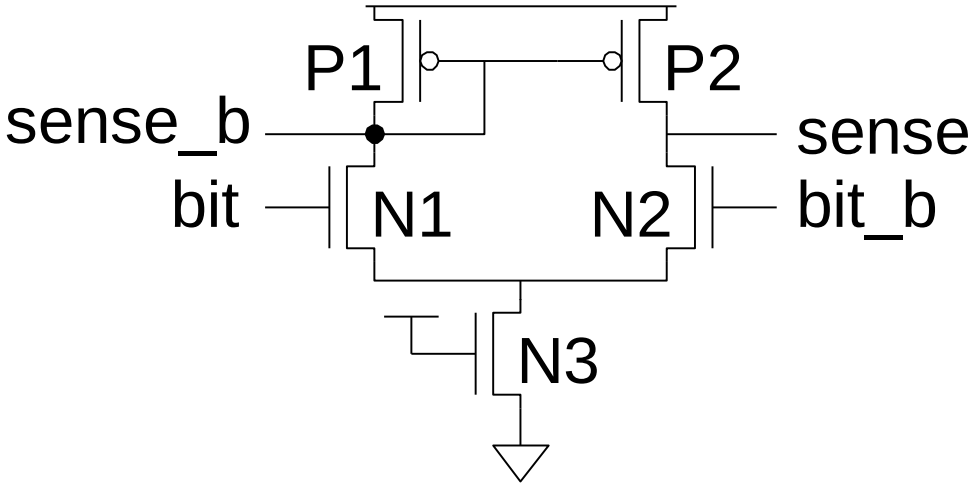
- Bitlines is classified: large-signal (single-ended sensing) or small-signal. (differential sensing).
- Large-signal: bitline swings between Vdd and GND.
- Small-signal: one of the two bitlines changes by a small amount to save delay and reduce energy consumption.

Bitline Sense Amplifiers

- Bitlines have many cells attached
  - Ex: 32-kbit SRAM has 128 rows x 256 cols
  - 128 cells on each bitline
- $t_{pd} \propto (C/I)\ \Delta V$
  - Even with shared diffusion contacts, 64C of diffusion capacitance (big C)
  - Discharged slowly through small transistors (small I)
- *Sense amplifiers* are triggered on small voltage swing (reduce ΔV)

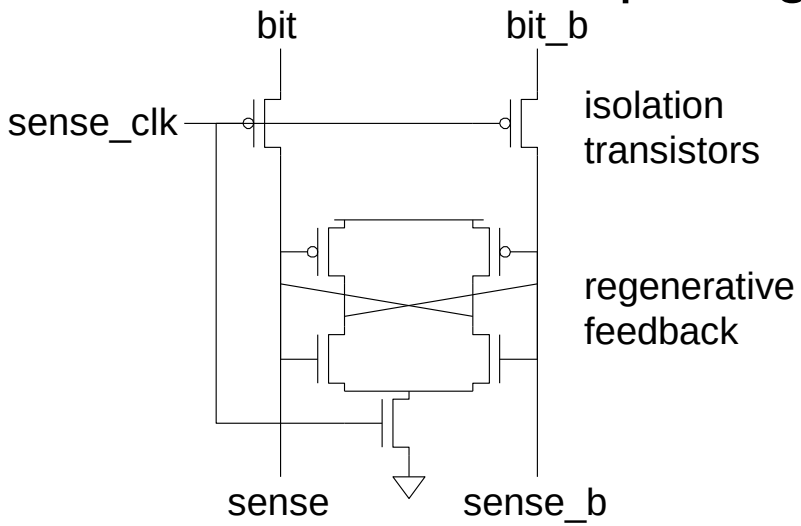
Differential Pair Amp

- Differential pair requires no clock
- But always dissipates static power
- Consumes a significant amount of DC power.



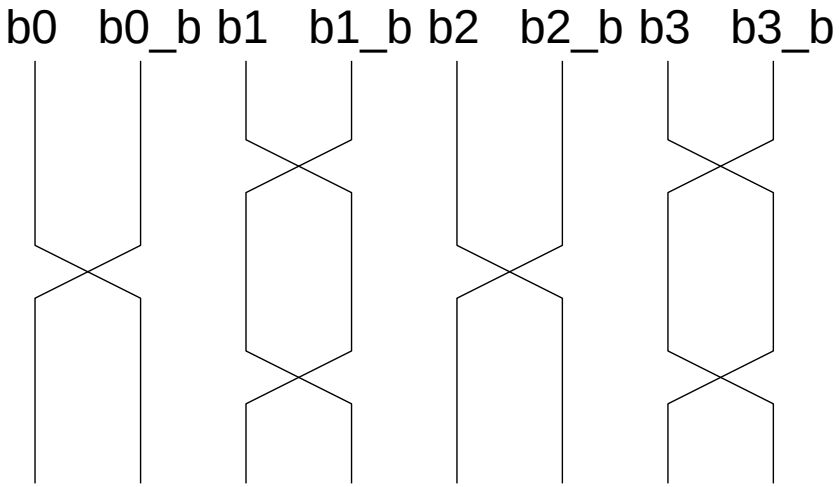
Clocked Sense Amp

- Clocked sense amp saves power (consumes power only while activated).
- Requires sense_clk after enough bitline swing
- Isolation transistors cut off large bitline capacitance, regenerative feedback to make one output high and the other low.



Twisted Bitlines

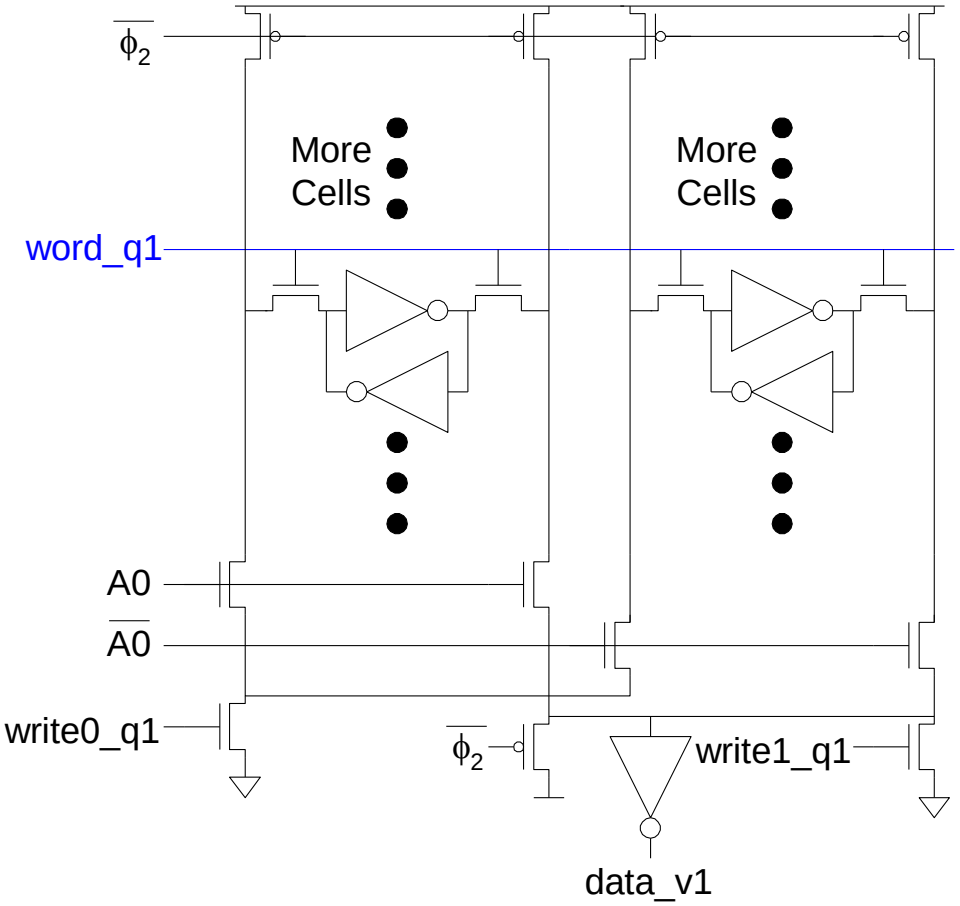
- Sense amplifiers also amplify noise
  - Coupling noise is severe in modern processes
  - Try to couple equally onto bit and bit_b
  - Done by *twisting or transposing* bitlines



Column Multiplexing

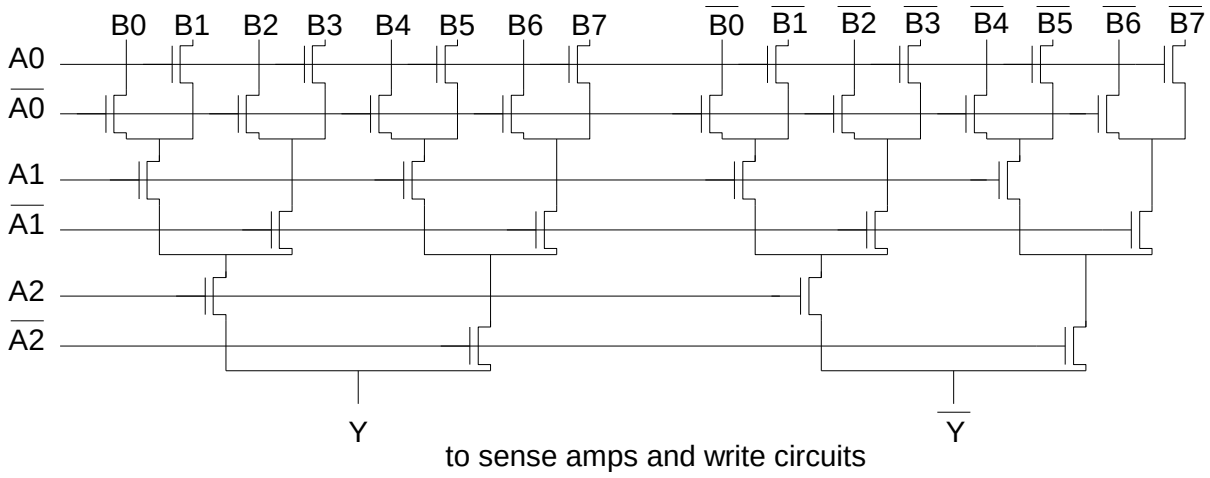
- Recall that array may be folded for good aspect ratio
- Ex: 2 kword x 16 folded into 256 rows x 128 columns
  - Must select 16 output bits from the 128 columns
  - Requires 16 8:1 column multiplexers

Ex: 2-way Muxed SRAM



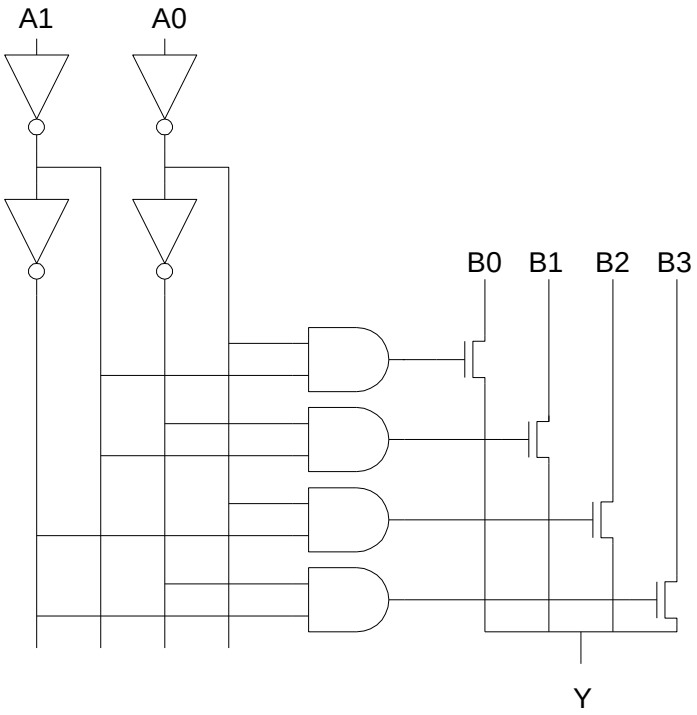
Tree Decoder Mux

- Column mux can use pass transistors
  - Use nMOS only, precharge outputs
- One design is to use k series transistors for 2^k:1 mux
  - No external decoder logic needed



Single Pass-Gate Mux

- Or eliminate series transistors with separate decoder

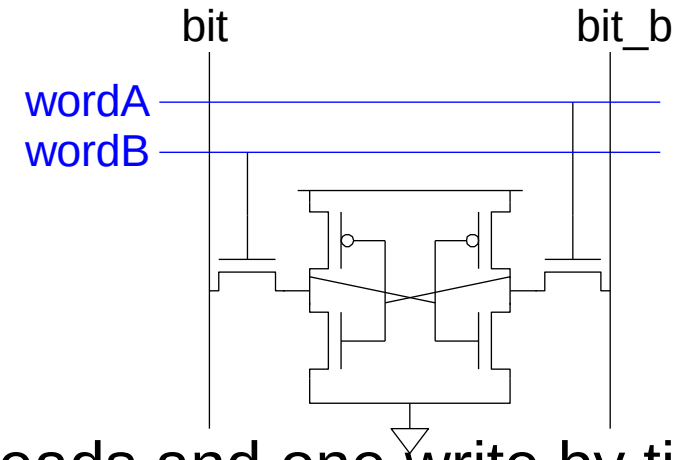


Multiple Ports

- We have considered single-ported SRAM
  - One read or one write on each cycle
- *Multiported* SRAM are needed for register files
- Examples:
  - Multicycle MIPS must read two sources or write a result on some cycles
  - Pipelined MIPS must read two sources and write a third result each cycle
  - Superscalar MIPS must read and write many sources and results each cycle

Dual-Ported SRAM

- Simple dual-ported SRAM
  - Two independent single-ended reads
  - Or one differential write



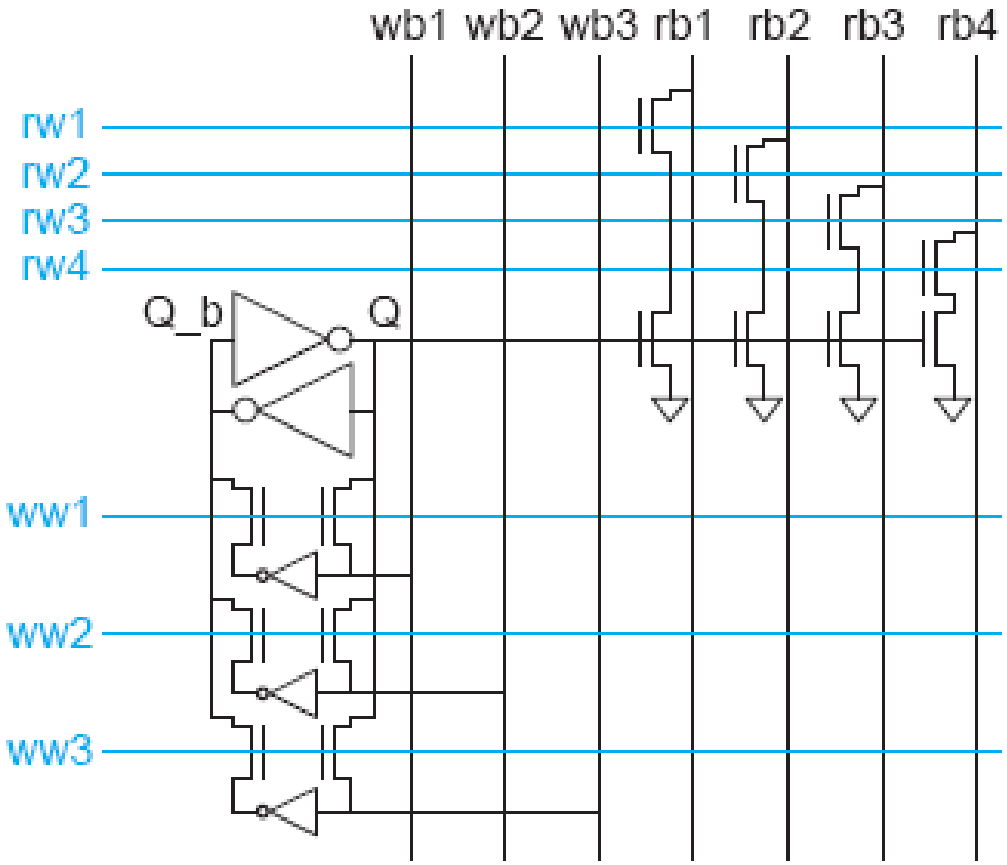
- Do two reads and one write by time multiplexing
  - Read during ph1, write during ph2

Multi-Ported SRAM

- Adding more access transistors hurts read stability
- Multiported SRAM isolates reads from state node
- Differential read ports double the number of read bitlines and transistors.
- Single-ended bitlines save area
- Multiple write ports simply attach the ports to the state node.

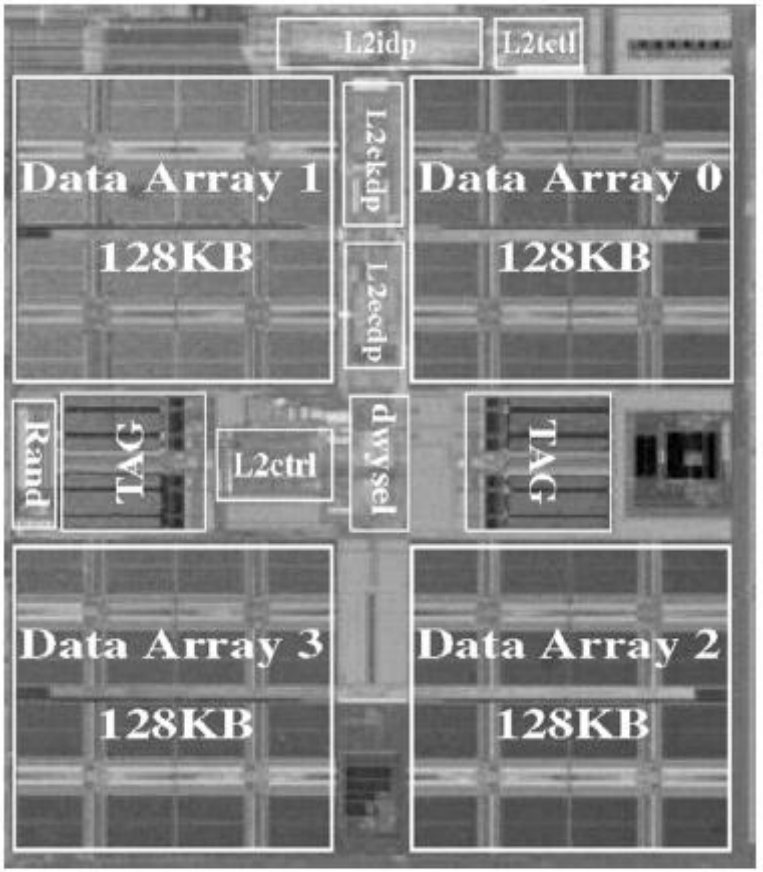
Multi-Ported SRAM

- 3 write ports and 4 read ports:



Large SRAMs

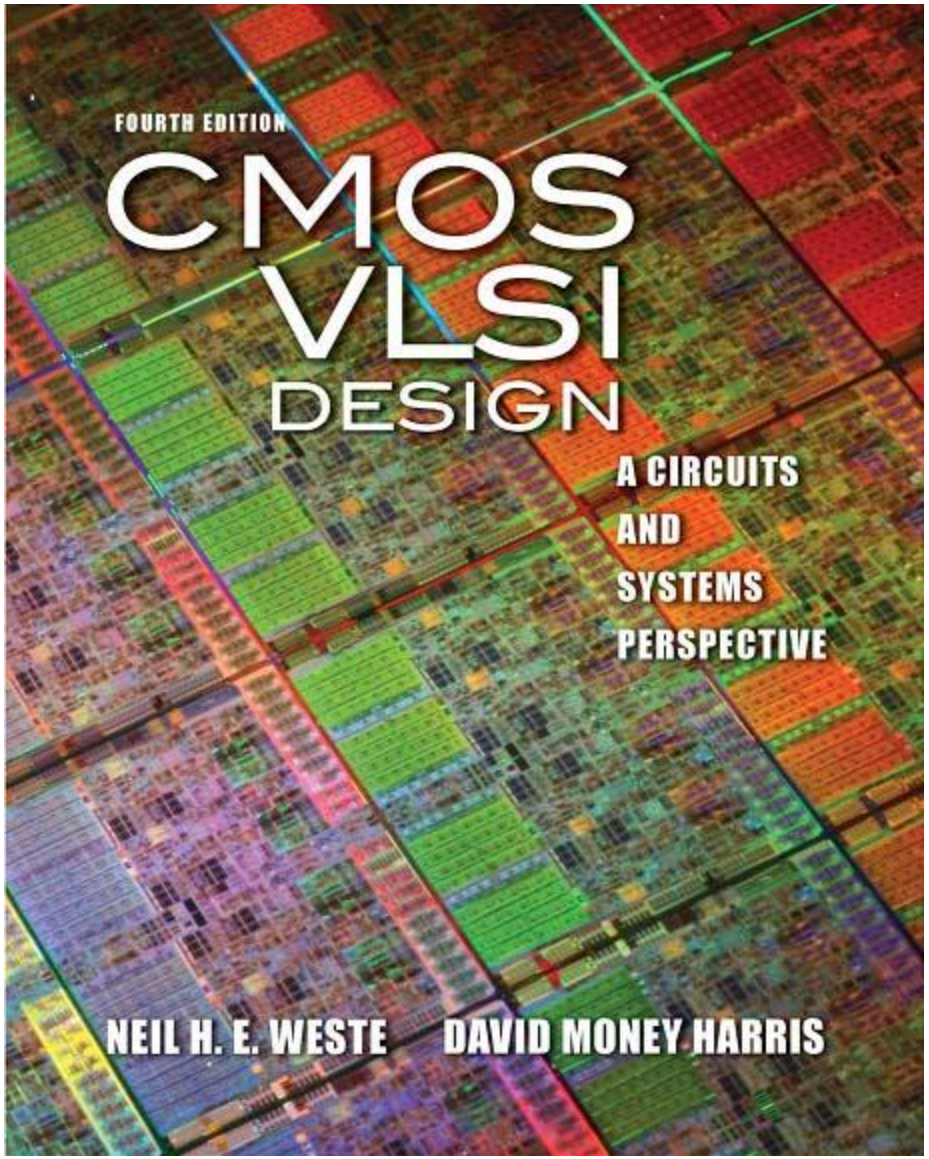
- Large SRAMs are split into banks or subarrays for speed and reduce power
- Ex: UltraSparc 512KB cache
  - 4 128 KB subarrays
  - Each have 16 8KB banks
  - 256 rows x 256 cols / bank
  - 60% subarray area efficiency
  - Also space for tags & control



CPE 110408423
VLSI Design
Chapter 12: Array Subsystems

Bassam Jamil

[Computer Engineering Department,
Hashemite University]



Lecture 4: ROMs, SAMs, CAMs, PLAs

Outline

- Read-Only Memories
- Serial Access Memories
- Content-Addressable Memories
- Programmable Logic Arrays

12.4 Read-Only Memories

- Read-Only Memory (ROM) cells can be built with only **one transistor per bit** of storage.
- A ROM is a nonvolatile memory structure in that the state is **retained** indefinitely—even **without power**.
- A ROM array is commonly implemented:
  - As a **single-ended NOR array**.
  - Commercial ROMs are normally dynamic, although pseudo-nMOS is simple and suffices for small structures.
- As in SRAM cells and other footless dynamic gates, the wordline input must be low during precharge on dynamic NOR gates.
- In situations where DC power dissipation is acceptable and the speed is sufficient, the **pseudo-nMOS ROM is the easiest to design**, requiring no timing.

ROM Example

- Below figure shows **4-word x 6-bit Pseudo-nMOS ROM**
  - Represented with dot diagram
  - Dots indicate 1's in ROM
- The dots correspond to nMOS pulldown transistors connected to the bitlines, but the outputs are inverted.

```
Word 0: 010101
Word 1: 011001
Word 2: 100101
Word 3: 101010
```

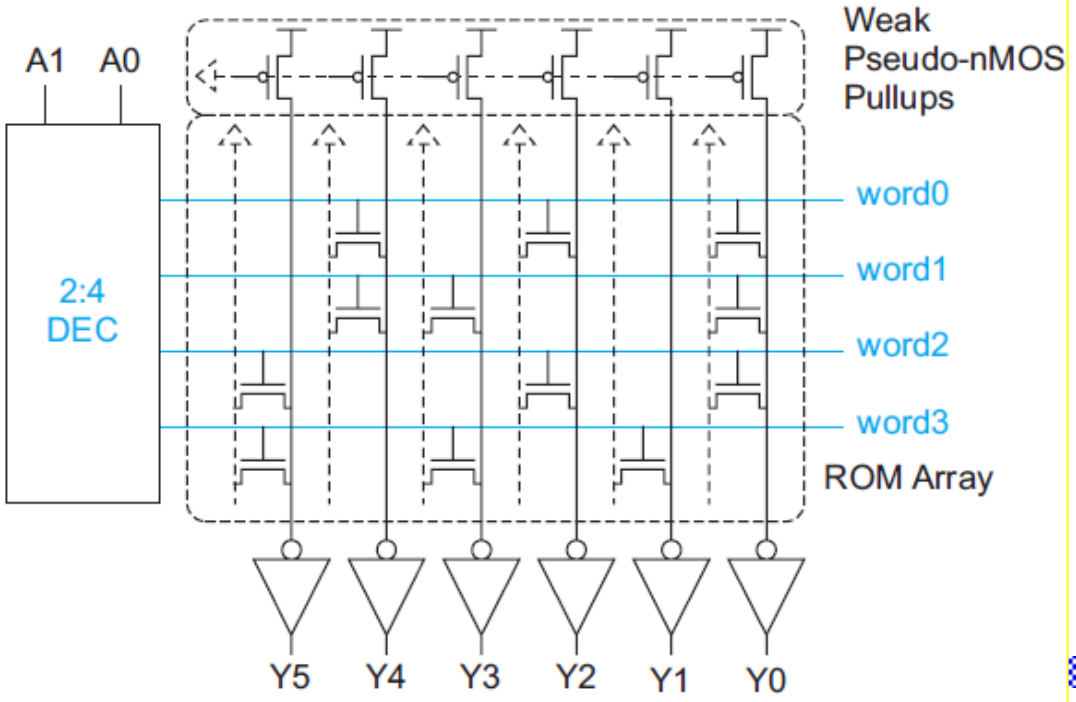
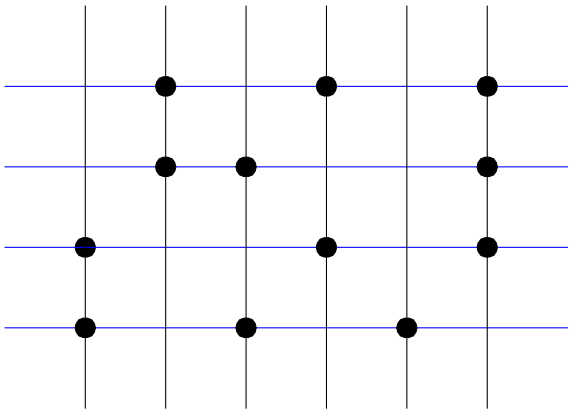


FIGURE 12.52 Pseudo-nMOS ROM



Configuring Mask-Programmed ROMs

- Mask-programmed ROMs can be **configured** by:
  - The **presence** or **absence** of a **transistor** or **contact**, OR
  - **Threshold implant** that turns a transistor permanently OFF where it is not needed.
- Omitting transistors has the advantage of reducing capacitance on the wordlines and power consumption.
- Programming with metal contacts was once popular because such ROMs could be completely manufactured except for the metal layer, and then programmed according to customer requirements through a metallization step.
- The advent of EEPROM and Flash memory chips has reduced demand for such mask programmed ROMs.

14.4.1 Programmable ROM

- Programming/writing speeds are generally slower than read speeds for ROMs.
- Four types of nonvolatile memories :
  - Programmable ROMs (PROMs),
  - Erasable Programmable ROMs (EPROMs),
  - Electrically Erasable Programmable ROMs (EEPROMs),
  - Flash memories.
- All of these memories require some enhancements to a standard CMOS process:
  - PROMs use fuses
  - EPROMs, EEPROMs, and Flash use charge stored on a floating gate.

PROMs

- Programmable ROMs can be fabricated as ordinary **ROMs** fully populated with pulldown transistors in every position.
  - Each transistor is placed in series with a **fuse** that can be **burned** out by applying **a high current**.
- The user typically configures the ROM in a specialized PROM programmer.
- As there is no way to repair a blown fuse, PROMs are also referred to as **one-time programmable memories**.
- As technology has improved, **reprogrammable nonvolatile memory** has largely displaced PROMs. These memories:
  - include: EPROM, EEPROM, Flash.
  - use a second layer of polysilicon to form a floating gate between the control gate and the channel

Floating Gate Transistor

- The floating gate is a good conductor, but it is **not attached** to anything.
- Applying a high voltage to the control gate causes **electrons to jump through the thin oxide onto the floating gate** through the processes called Fowler-Nordheim (FN) tunneling.
- Injecting the electrons **induces a negative voltage on the floating gate**, effectively **increasing the threshold** voltage of the transistor to the point that it is **always OFF**.

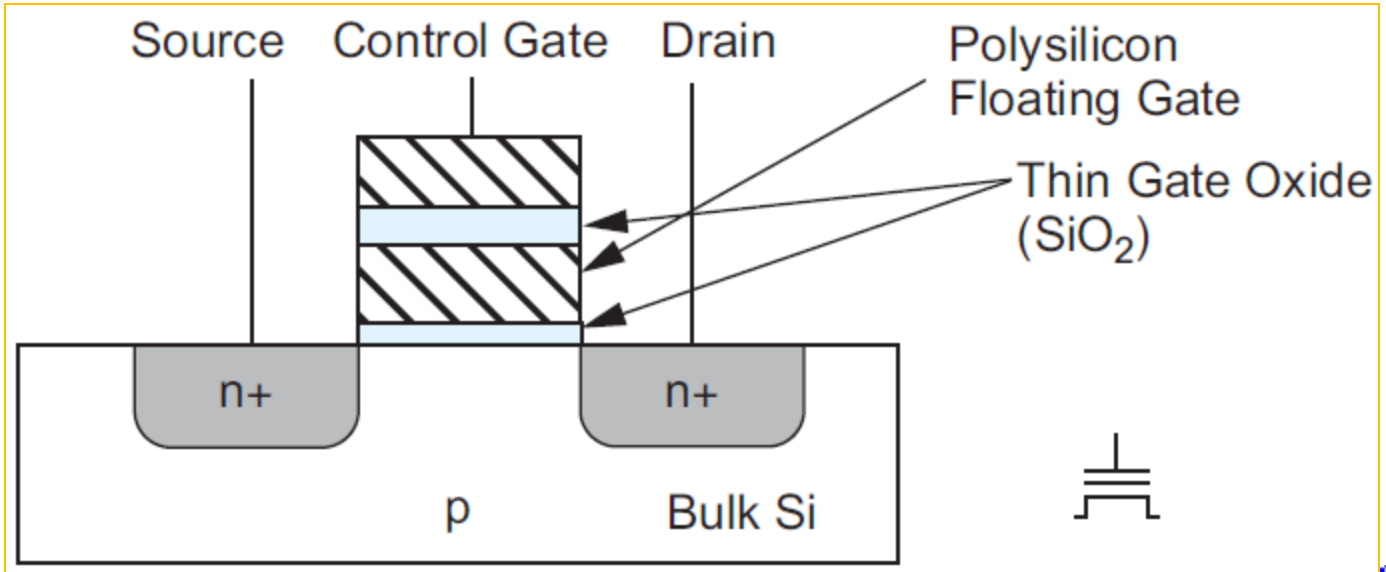


FIGURE 12.57 Cross-section of floating gate nMOS transistor

EPROM, EEPROM and Flash

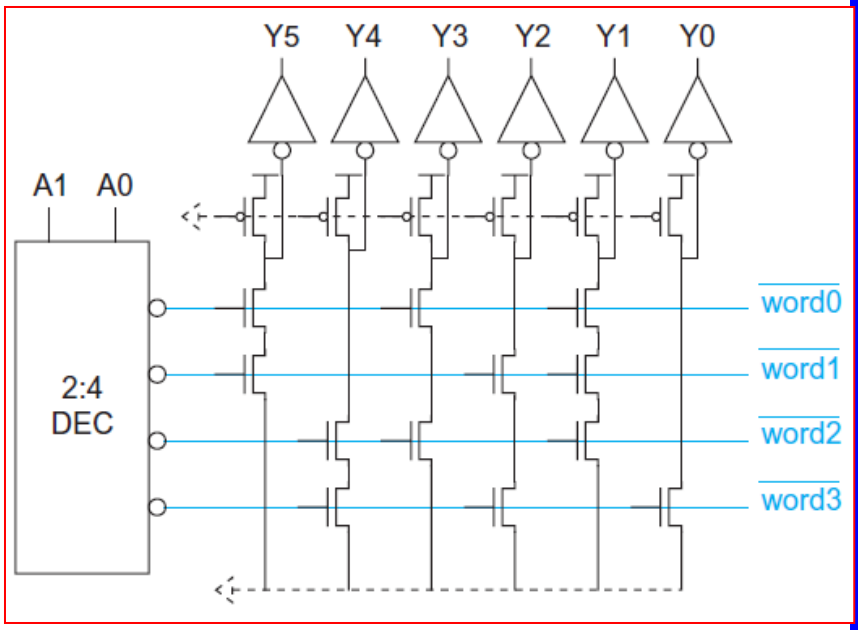
- The difference between EPROM and EEPROM lies in the way that the memory programs and erases.
  - EPROMs:
    - EPROM is programmed electrically, but it is erased through exposure to ultraviolet light that knocks the electrons off the floating gate.
    - EPROMs are encapsulated in plastic that is opaque to UV light, making them "one-time programmable".
    - It offers a dense cell.
  - EEPROM can be programmed and erased electrically.

EEPROM and Flash

- Both EEPROM and Flash can be erased electrically without being removed from the system.
  - EEPROM offers fine-grained control over which bits are erased, while Flash is erased in bulk.
  - EEPROM cells are larger, so Flash has become the most economical form of convenient nonvolatile storage.
- Most NOR Flash memory is a hybrid style—programming is through hot carrier injection and erase is through Fowler–Nordheim tunneling.

NAND ROMs

- The main issue of NOR ROM:
  - The size of the cell is limited by the ground line.
- NAND ROM
  - Does not require the ground line .
    - → Smaller area
  - Uses active-low wordlines.
  - Transistors are placed in series and the transistors on the nonselected rows are ON.
  - If no transistor is associated with the selected word, the bitline will pull down. If a transistor is present, the bitline will remain high.



- Disadvantage of the NAND ROM is that the delay grows quadratically with the number of series transistors discharging the bitline. NAND structures with more than 8–16 series transistors become extremely slow.
- NAND structures are attractive for Flash memories in which density and cost are more important than access time.

12.4.3 Flash

- **Blocks** of memory were erased all at once.
- Nonvolatile, high density and low cost per bit.
- Most stand-alone Flash memory uses the NAND architecture to minimize bit cell size and cost.
- NAND Flash memories are divided into blocks, which in turn are made of pages.
- The memory is written one **page** at a time and erased one **block** at a time.
- The charge on the floating gate determines the threshold of the transistor and indicates the state of the cell. A negative threshold represents a logic '1' and a positive threshold represents a logic 0.'.

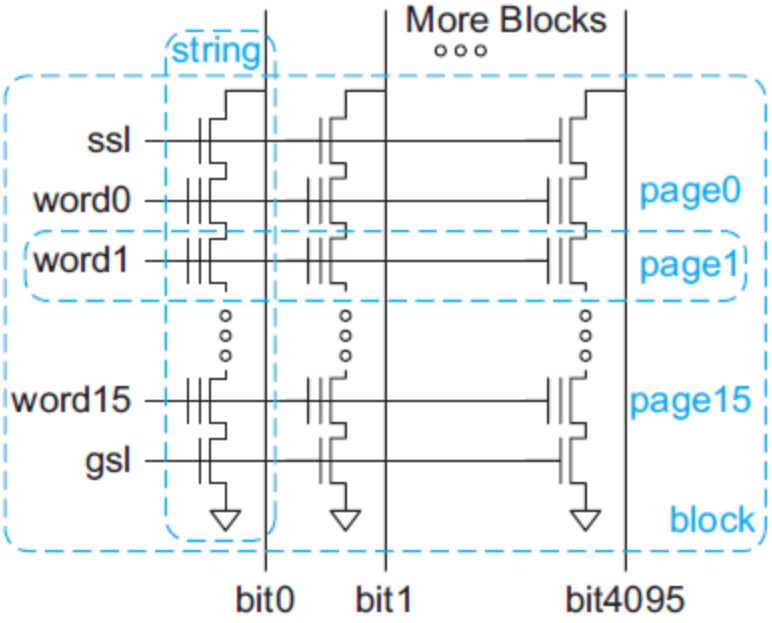


FIGURE 12.60 NAND Flash string

NAND Flash

- Floating gate transistors are connected in series to form strings.
- The Figure shows the organization of a string, page, and block.
- Each string consists of :**16 cells**, **a string select** transistor, and a **ground select** transistor all connected in series and attached to the bitline.
- The control gate of each cell is connected to a wordline.

- The array contains one column for each bit in a page.
- Each column contains one string per block.
- The number of cells in the string determines the number of pages per block.

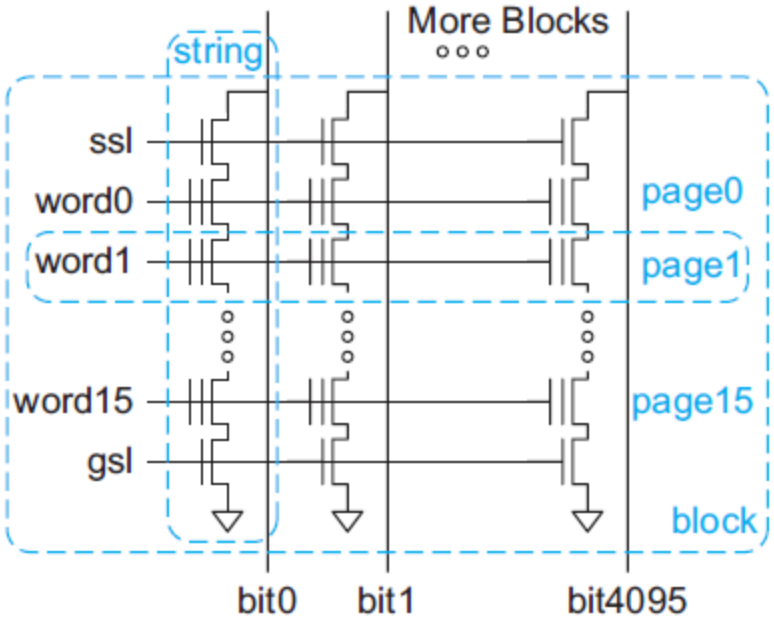


FIGURE 12.60 NAND Flash string

Flash Size

- Flash Size = Number of Blocks × Block Size
- Block Size = Page Size × String Size × Bits/Trans
- Page Size = # of columns
- String Size= # of pages
- Bits/Trans:
  - Default = 1
  - Multi-level: 2 or 4 ….
- Example (1):
  - Flash has 8 blocks, 16 pages, each page is 512 B (4 Kb).
  - Flash size = Number of Blocks × Page Size × String Size × Bits/Trans
    = 8 × 512B × 16 × 1 = 64KB
- Example (2): flash has
  - 2 blocks, Number of columns =64K, String size = 64 transistors
  - Flash uses 16 levels to store 4 bits per transistor.
  - Flash size = Number of Blocks × Page Size × String Size × Bits/Trans
    = 2 × 64K × 64 × 4
    = 32 Gb = 4 GB

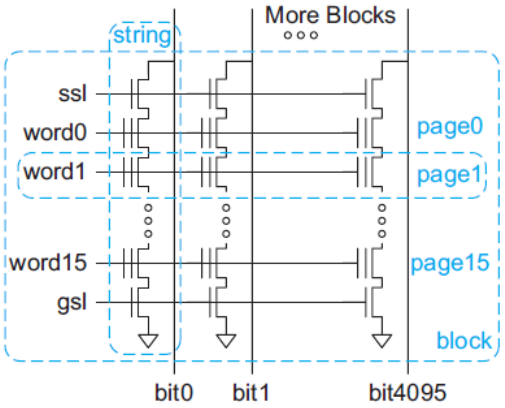
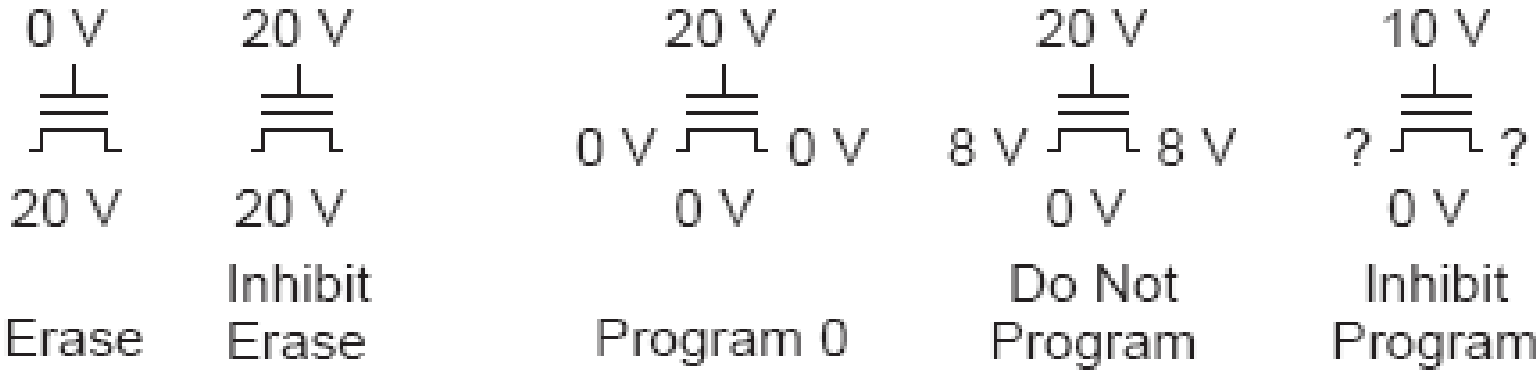


FIGURE 12.60 NAND Flash string

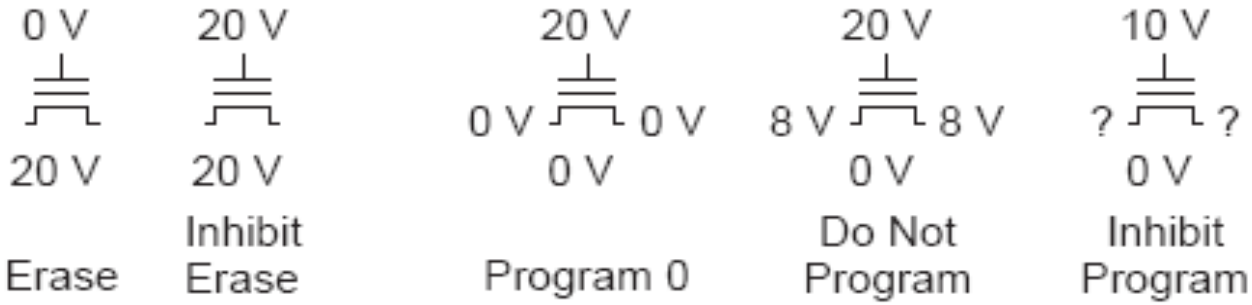
Flash Programming

- Charge on floating gate determines Vt
- Logic 1: negative Vt
- Logic 0: positive Vt
- Cells erased to 1 by applying a high body voltage so that electrons tunnel off floating gate into substrate
- Programmed to 0 by applying high gate voltage



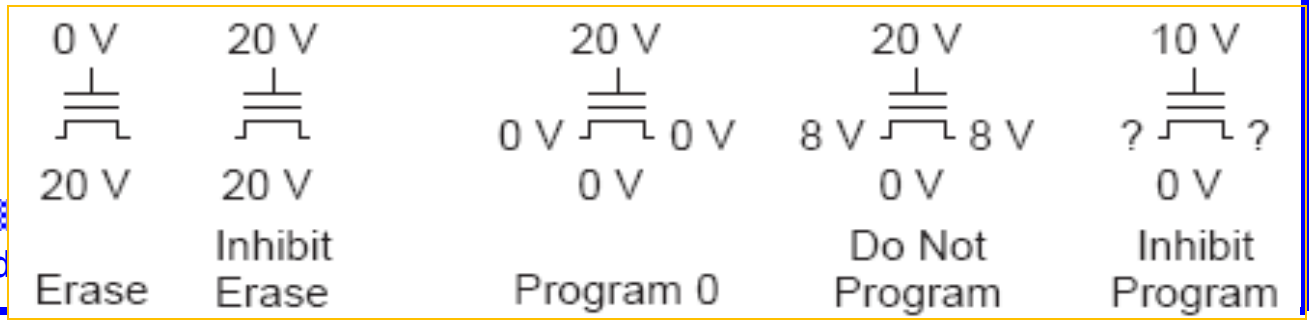
Flash Programming: Block Erase

- Block is erased by:
  - Setting all of the control **gates to GND and raising the substrate to 20 V**.
  - The high voltage across the gate oxide **induces FN tunneling**, causing the electrons to flow from the floating gate to the substrate.
  - At the end of the erase step, all the floating gate transistors have a negative Vt and thus represent 1.
- Tunneling is a **slow** process, so block erase takes on the order of a millisecond.
- The wordlines for other blocks on the chip are set to the same voltage as the substrate to inhibit erasing.
- An on-chip charge pump is used to generate the high voltages.



Flash Programming: page programming

- A cell is programmed (written) to 0 by tunneling electrons onto the floating gate.
- The programming cannot restore 1 values, so the block must be erased before any cell is reprogrammed.
- An entire page is programmed at once.
- To program a page:
  - bitlines are driven with the data values: 0 V for a logic 0 and 8 V for a logic 1.
  - substrate is held at ground.
  - wordline is set to 20 V for the page being programmed and 10 V for the other pages in the block.
  - ground select line (gsl) is left OFF but the string select line (ssl) for the block is turned ON, passing the voltage on the bitline to the channels of all the transistors being programmed.
  - Thus, cells being programmed to 0 see 20 V on the control gate and 0 V on the channel.
- This high voltage difference induces FN tunneling that drives electrons onto the floating gate, raising Vt to a positive voltage. The other cells see a smaller voltage that is insufficient to cause tunneling.

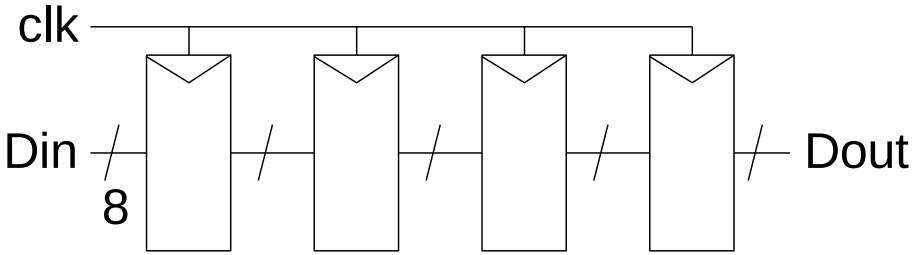


12.5 Serial Access Memories

- Serial access memories do not use an address
  - Shift Registers
  - Tapped Delay Lines
  - Serial In Parallel Out (SIPO)
  - Parallel In Serial Out (PISO)
  - Queues (FIFO, LIFO)

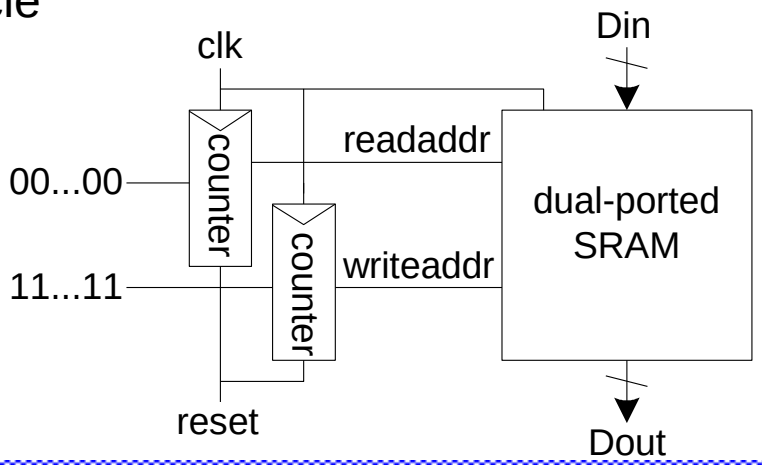
12.5.1 Shift Register

- *Shift registers* store and delay data
  - commonly used in signal-processing applications to store and delay data.
- Simple design:
  - a simple 4-stage 8-bit shift register constructed from 32 flip-flops.
  - No logic between the registers Watch your hold times!



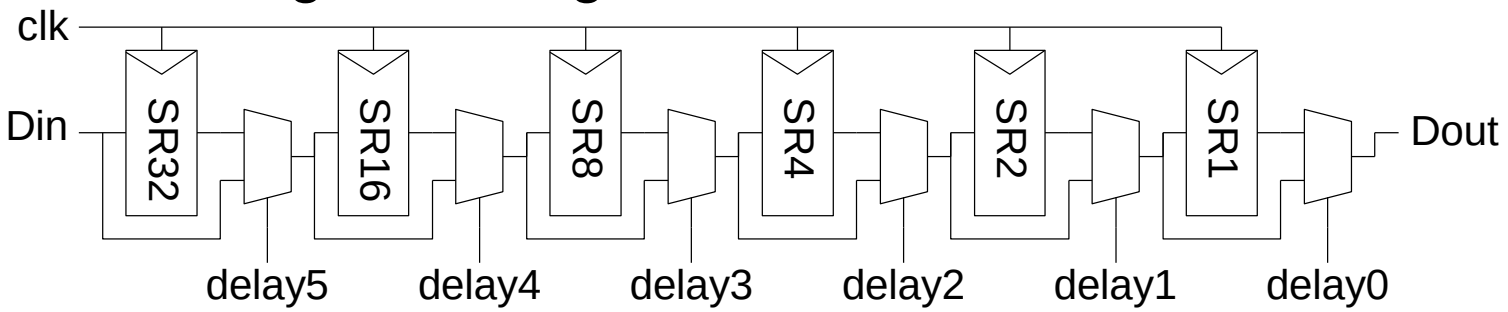
Denser Shift Registers

- Flip-flops aren't very area-efficient
- For large shift registers, keep data in SRAM instead
- The RAM is configured as a circular buffer with a pair of counters specifying where the data is read and written.
- Move read/write pointers to RAM rather than data
  - The read counter is initialized to the first entry and the write counter to the last entry on reset
  - Increment address on each cycle



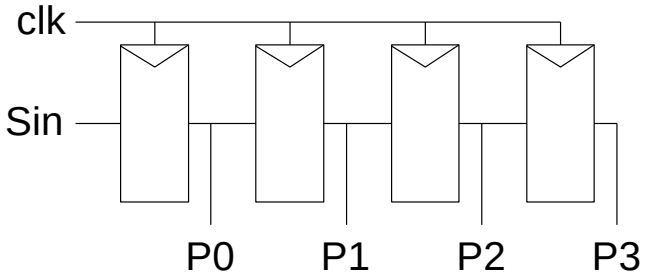
Tapped Delay Line

- A *tapped delay line* is a shift register with a programmable number of delay stages
- Multiplexers control pass-around of the delay blocks to provide the appropriate total delay.
- The Figure shows a 64-stage tapped delay line .
  - Delay blocks are built from 32-, 16-, 8-, 4-, 2-, and 1-stage shift registers.

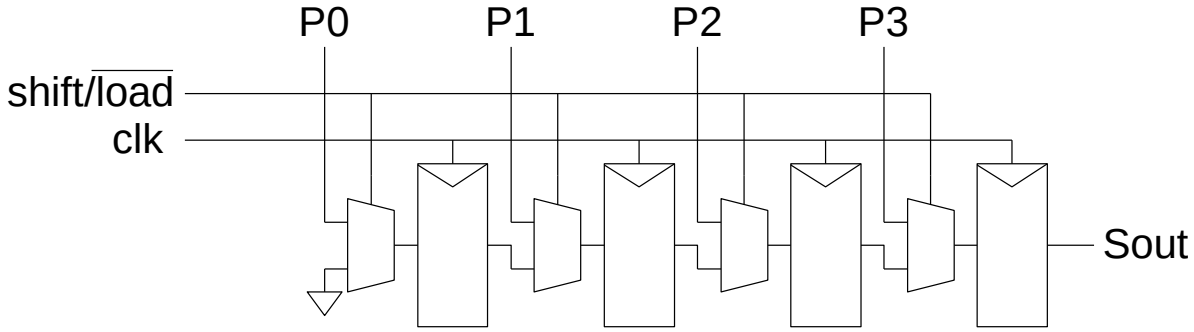


Serial In Parallel Out & Parallel In Serial Out

- **Serial In Parallel Out**: 1-bit shift register reads in serial data
  - After N steps, presents N-bit parallel output

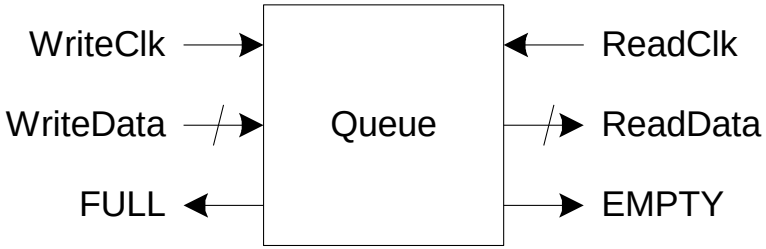


- **Parallel In Serial Out**: Load all N bits in parallel when shift = 0
  - Then shift one bit out per cycle



12.5.2 Queues

- *Queues* allow data to be read and written at different rates.
- Below figure shows an interface to a queue:
  - Read and write operations each are controlled by their own clocks that may be **asynchronous**
  - **FULL flag** means there is no room remaining to write data
  - **EMPTY flag** means there is no data to read.
- Build with SRAM and read/write counters (pointers)
  - The queue internally maintains read and write pointers indicating which data should be accessed next.
  - The queue internally maintains read and write pointers indicating which data should be accessed next.
- Some queues also provide **ALMOST-FULL** and **ALMOST-EMPTY** flags to communicate the impending state and halt write or read requests.



FIFO, LIFO Queues

First In First Out **(FIFO)**

Used to buffer data between two asynchronous streams.
Organized as a circular buffer.

On reset,

the read and write pointers are both initialized to the first element
FIFO is EMPTY.

On a write,

write pointer advances to the next element.
If it is about to catch the read pointer, the FIFO is FULL.

On a read,

the read pointer advances to the next element.
If it catches the write pointer, the FIFO is EMPTY again.

Last In First Out **(LIFO)**

- Known as stacks, are used in applications such as subroutine or interrupt stacks in microcontrollers.
- Uses a single pointer for both read and write.

On reset,

the pointer is initialized to the first element
LIFO is EMPTY.

On a write,

the pointer is incremented.
If it reaches the last element, the LIFO is FULL.

On a read,

the pointer is decremented.
If it reaches the first element, the LIFO is EMPTY again.

12.6 Content-Addressable Memory (CAM)

- The CAM is an extention of SRAM
  - Can be read or written given *adr and data*
  - But also performs matching operations.
- Matching asserts a matchline output for each word of the CAM that contains a specified key.
- A common application of CAMs is translation lookaside buffers (TLBs)
  - The virtual address is given as the key to the TLB CAM.
  - If this address is in the CAM, the corresponding matchline is asserted.
  - This matchline can serve as the wordline to access a RAM containing the associated physical address, as shown in Figure 12.68.
  - A NOR gate processing all of the matchlines generates a miss signal for the CAM.

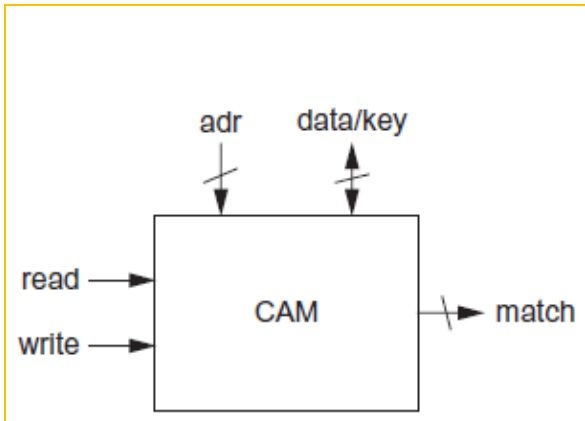


FIGURE 12.67 Content-addressable memory

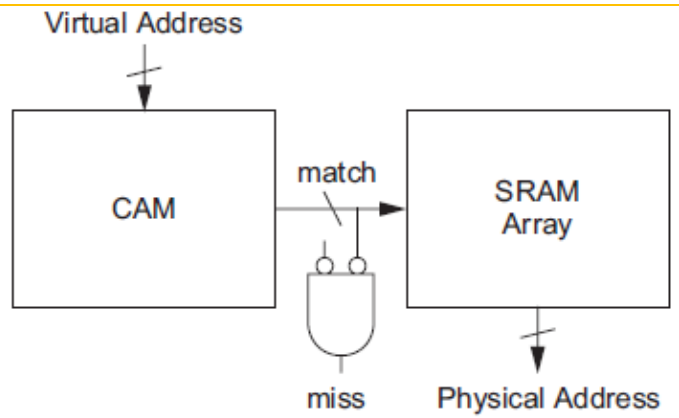
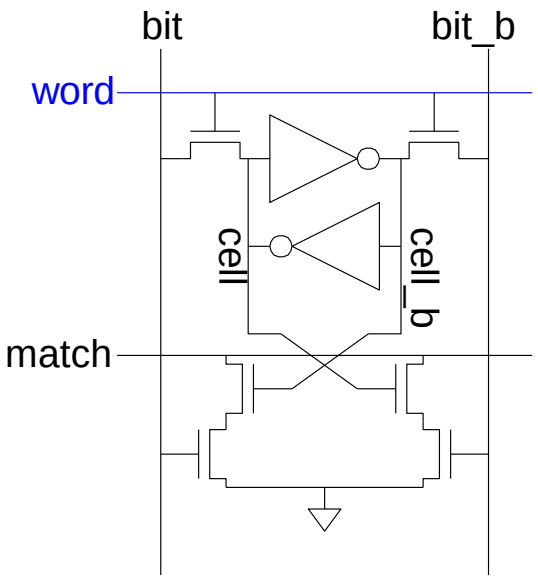
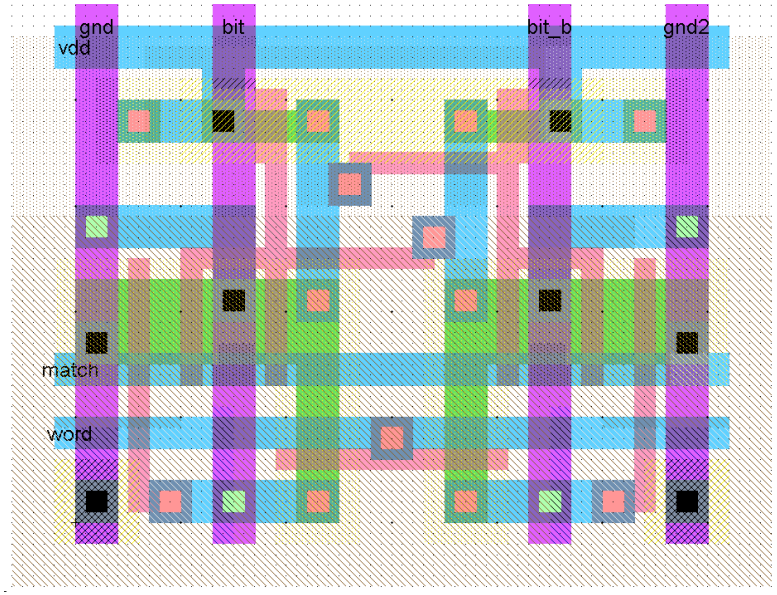


FIGURE 12.68 Translation Lookaside Buffer (TLB)

10T CAM Cell (read)

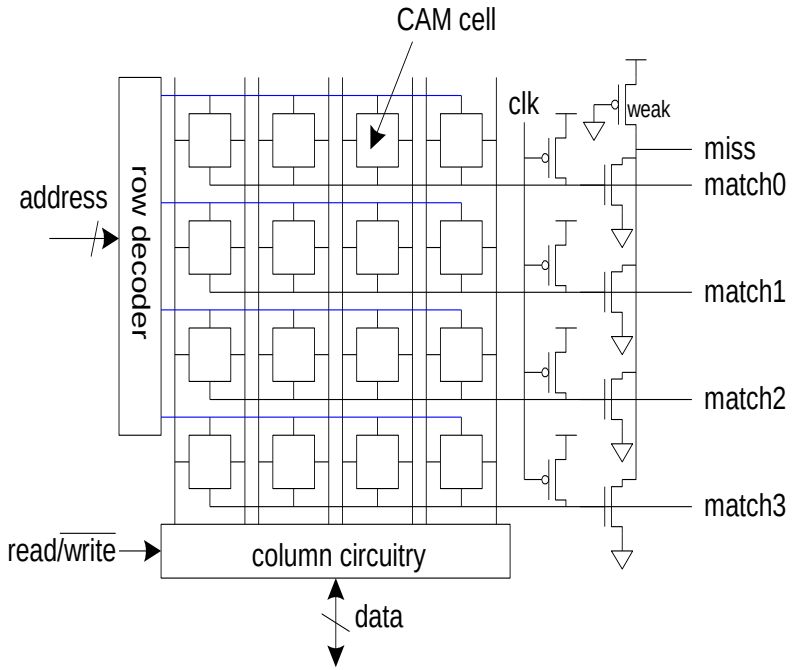
- Add four match transistors to 10T SRAM
  - 56 x 43 λ unit cell





CAM Cell Operation (read)

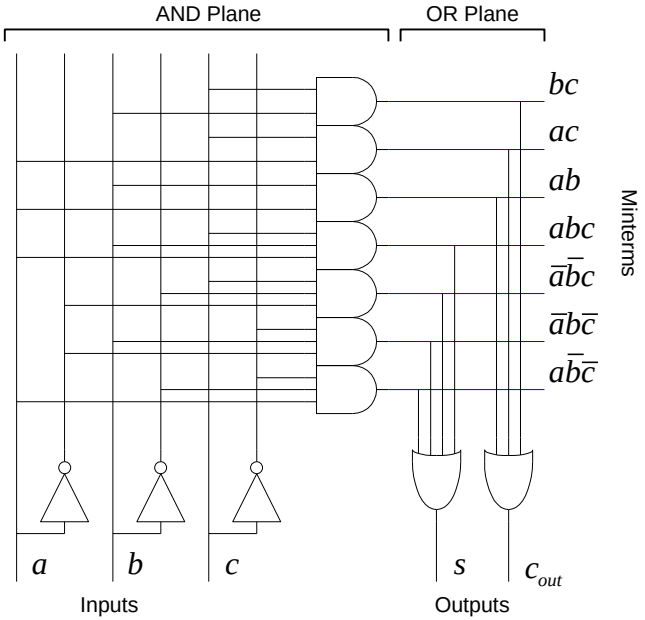
- Read and write like ordinary SRAM
- For matching:
  - Leave wordline low
  - Precharge matchlines
  - Place key on bitlines
  - Matchlines evaluate



- Miss line
  - Pseudo-nMOS NOR of match lines
  - Goes high if no words match

12.7 Programmable Logic Array

- A *PLA* performs any function in sum-of-products form (combinational logic circuit).
- *Literals*: inputs & complements
- *Products / Minterms*: AND of literals
- *Outputs*: OR of Minterms



- Example: Full Adder

$$s = a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c + abc$$

$$c_{\text{out}} = ab + bc + ac$$

NOR-NOR PLAs

- ANDs and ORs are not very efficient in CMOS
- Dynamic or Pseudo-nMOS NORs are very efficient
- Use DeMorgan's Law to convert to all NORs

PLA Example

Example 12.5

Write the equations for a full adder in sum-of-products form. Sketch a 3-input, 2-output PLA implementing this logic.

SOLUTION: Figure 12.75 shows the PLA. The logic equations are

$$s = a\bar{b}\bar{c} + \bar{a}b\bar{c} + \bar{a}\bar{b}c + abc \qquad (12.9)$$
$$c_{out} = ab + bc + ac$$

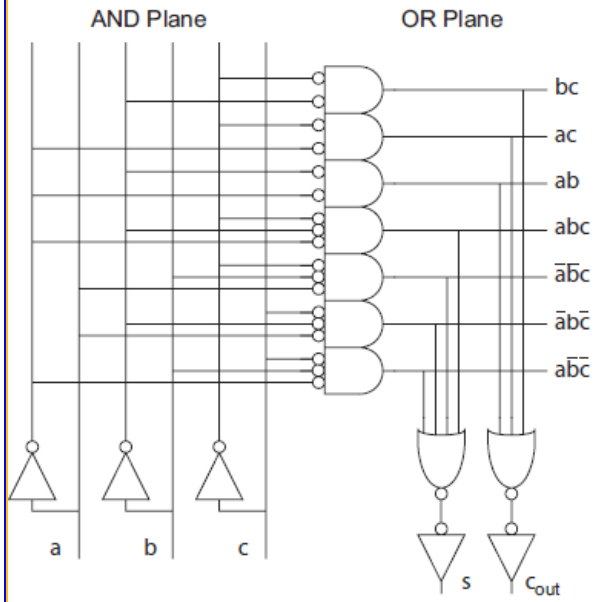


FIGURE 12.75 AND/OR representation of PLA

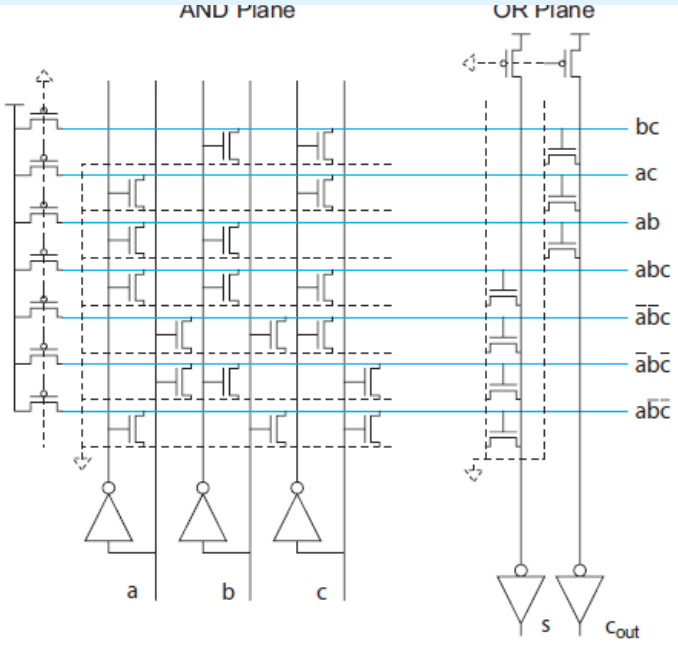


FIGURE 12.76 Pseudo-nMOS PLA schematic

PLA Example: Dot Diagram

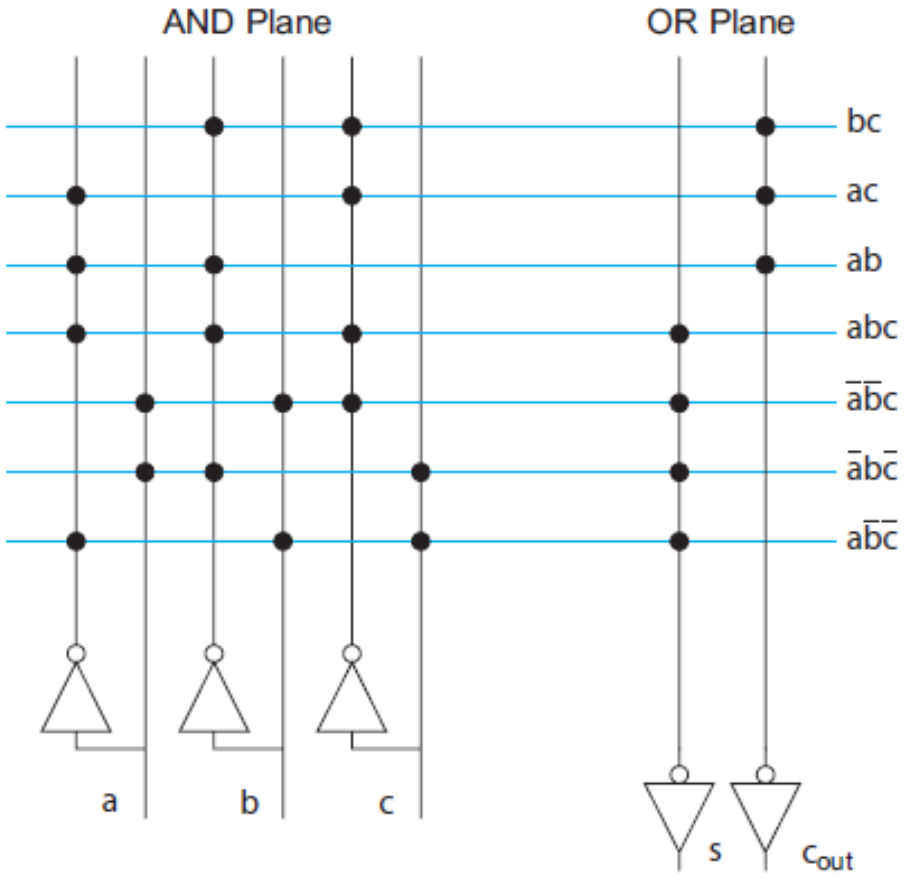


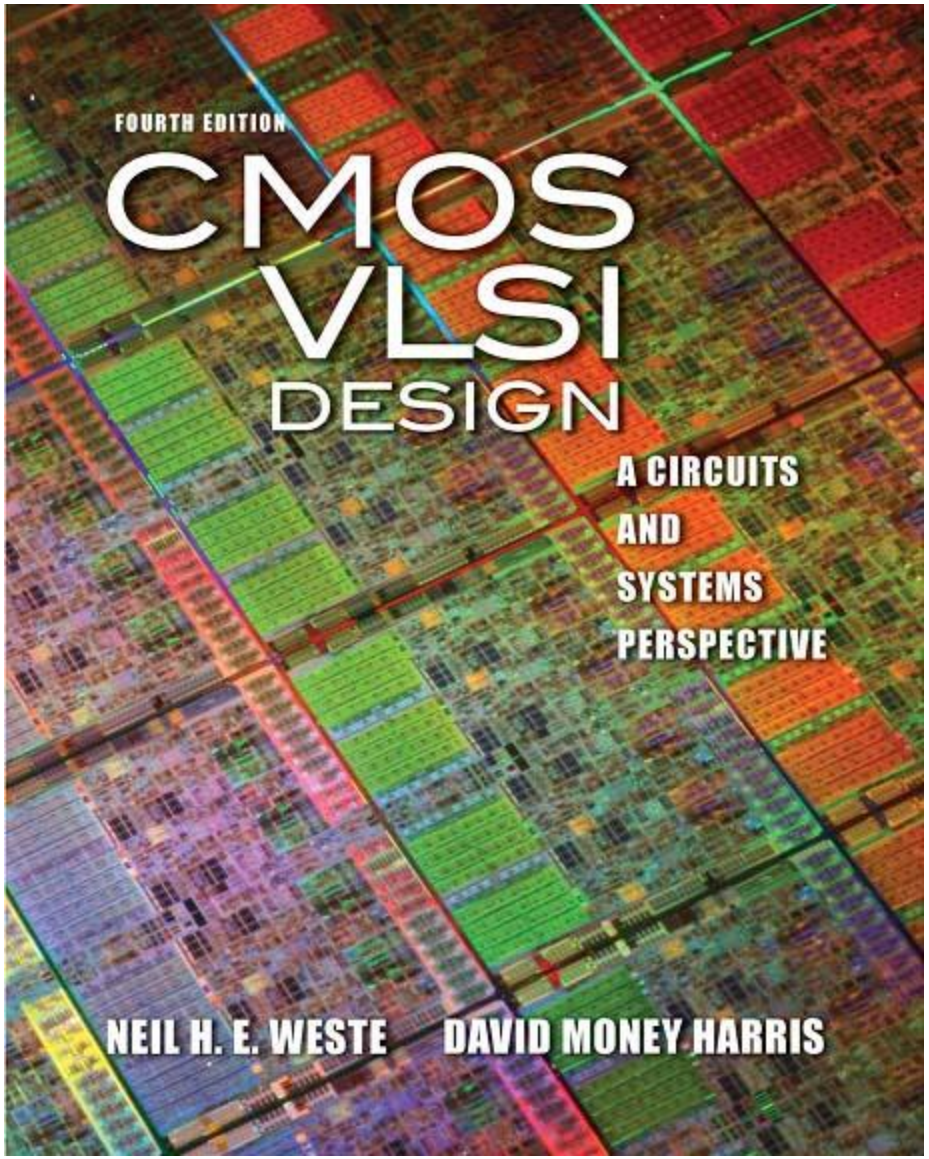
FIGURE 12.74 Dot diagram representation of PLA

PLAs vs. ROMs

- The OR plane of the PLA is like the ROM array
- The AND plane of the PLA is like the ROM decoder
- PLAs are more flexible than ROMs
  - No need to have 2^n rows for n inputs
  - Only generate the minterms that are needed
  - Take advantage of logic simplification

CPE 110408423
VLSI Design
Chapter 14: Design Methodology Tools

Bassam Jamil
[Computer Engineering Department,
Hashemite University]



Lecture 8: Design Methodology Tools

Outline

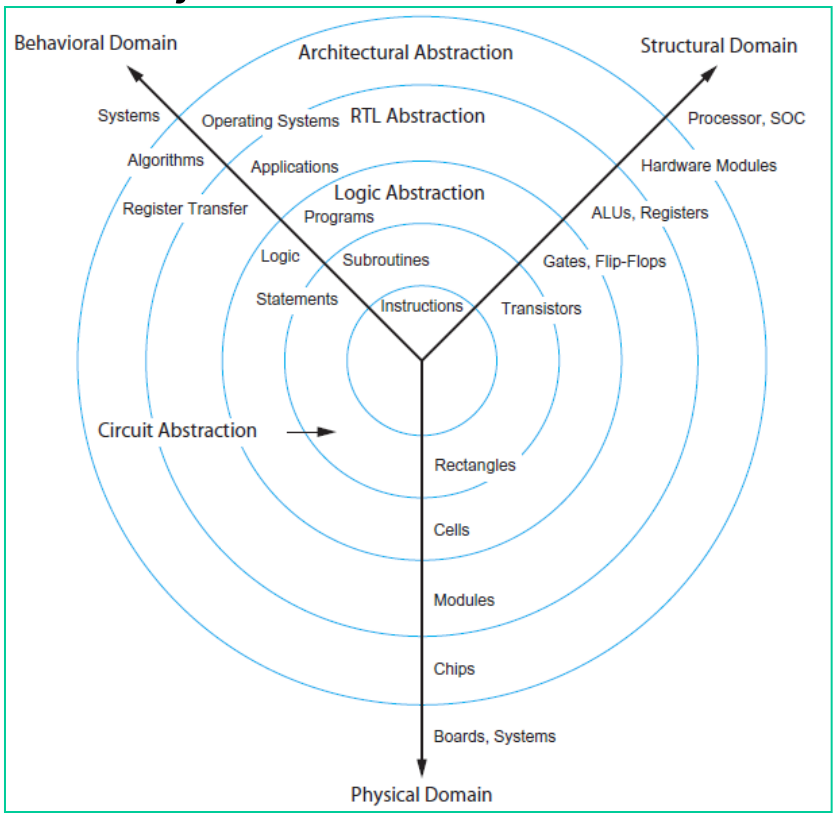
- Introduction
- Structure Design Strategies
- Design Methods
- Design Flows
- Design Economics

14.1 Introduction

- **Design Methodologies** are: developed and adapted strategies to form a cohesive set of principles to increase the likelihood of timely, successful designs.
- Integrated circuit can be described in terms of three domains:
  - **The behavioural domain** specifies what we wish to accomplish with a system.
  - **The structural domain** specifies the interconnection of components required to achieve the behavior we desire.
  - **The physical domain** specifies how to arrange the components in order to connect them, which in turn allows the required behavior.
- Design flows from behaviour to structure and to a physical implementation via transformations where the correctness is tested by comparing the pre- and post-transformation design.
- These domains can further be hierarchically divided into different levels of *design abstraction:*
  - Architectural or functional level
  - Logic or Register Transfer Level (RTL)
  - Circuit level

14.1 Introduction: Y-chart

- The relationship between description domains and levels of abstraction is elegantly shown by the *Gajski-Kuhn Y chart.*
  - The three radial lines represent the behavioral, structural, and physical domains.
  - As we move out along any of the radial axes, the increasing level of design abstraction is able to represent greater complexity.
  - Circles represent levels of similar design abstraction: the architectural, RTL, logic, and circuit levels.



14.2 Structured Design Strategies

- A good VLSI design system should provide for consistent descriptions in all three description domains (behavioural, structural, and physical) and at all relevant levels of abstraction (e.g., architecture, RTL/block, logic, and circuit).
- The performance can be measured by the following design parameters:
  - Performance—speed, power, function, flexibility
  - Size of die (hence, cost of die)
  - Time to design (hence, cost of engineering and schedule)
  - Ease of verification, test generation, and testability (hence, cost of engineering and schedule).
- Design is a continuous trade-off to achieve adequate results for all of the above parameters.
- A good VLSI-design aids is to reduce complexity, increase productivity, and assure the designer of a working product.

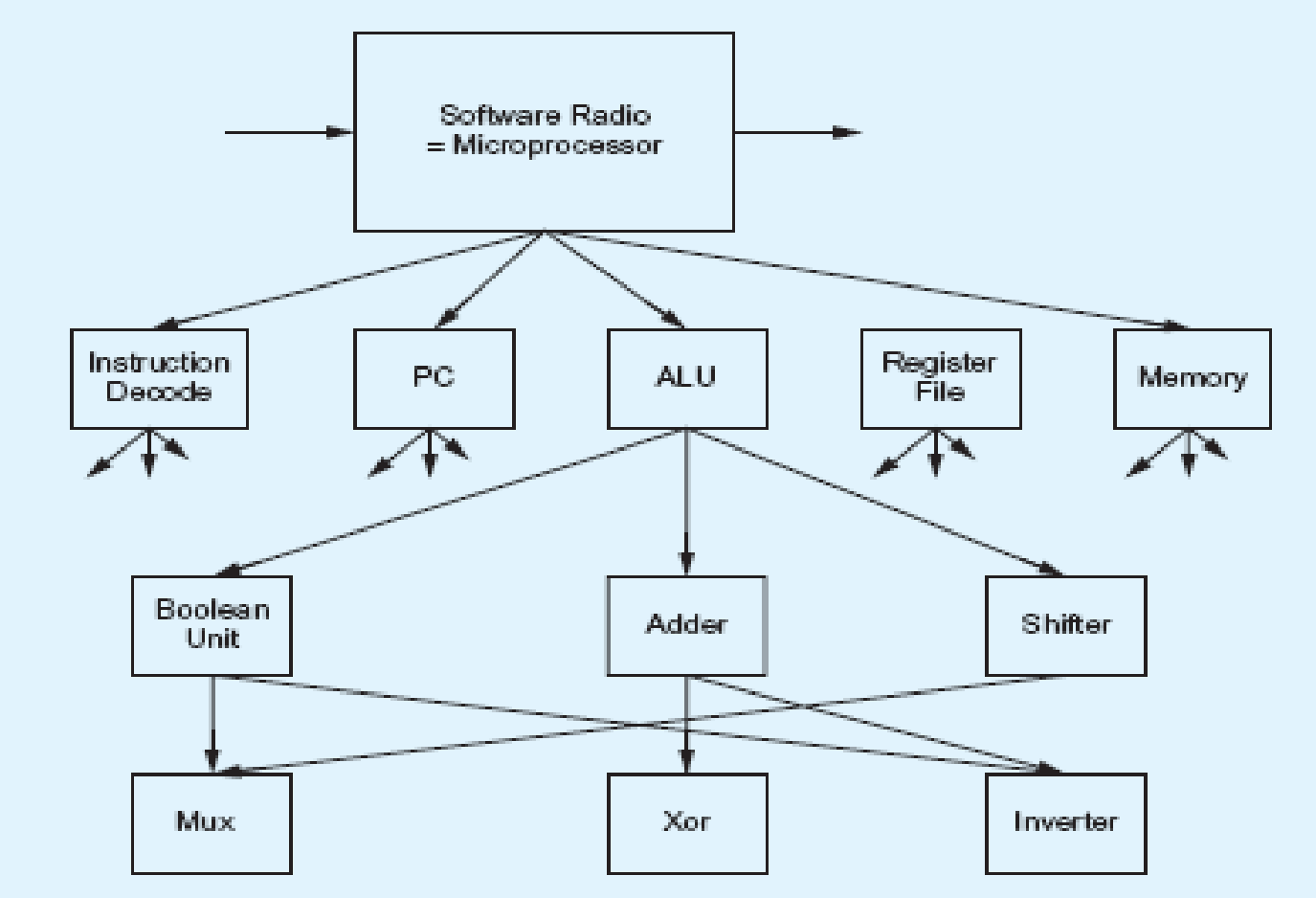
Design Principles:

- hierarchy,
- regularity,
- modularity,
- and locality

14.2.2 Hierarchy : “divide and conquer”

- Dividing a system into modules, then repeating this process until complexity of the sub modules is at an appropriately comprehensible level of detail.
- One can progress in verification from the bottom to the top of a design, checking each level of hierarchy where domains are intended to correspond.
- Hierarchy allows the use of *virtual components, soft versions of the more conventional* packaged IC. They are placed into a chip design as pieces of code. They can be supplied by an independent *intellectual property (IP) provider or can be reused from a previous product developed* in your organization.

hierarchy of software radio using a single microprocessor



14.2.3 Regularity

- The designer attempts to divide the hierarchy into a set of similar building blocks. Regularity can exist at all levels of the design hierarchy.
- Regularity aids in verification efforts by reducing the number of subcomponents to validate and by allowing formal verification programs to operate more efficiently.
- Design reuse depends on the principle of regularity to use the same virtual component in multiple places or products.

14.2.4 Modularity

- Modules have well-defined functions and interfaces.
- In the IC case, this corresponds to a clearly defined behavioural, structural, and physical **interface** that indicates the function, the name, signal type, electrical and timing constraints of the ports on the design.
- The ability to divide the task into a set of well-defined modules also aids in System-On-Chip (SOC) designs where a number of IP sources have to be interfaced to complete a design.

14.2.5 Locality

- The internals of the module are unimportant to other modules.
  - In this way we are performing a form of “information hiding” that reduces the apparent complexity of the module.
- Increasingly, locality often means temporal locality or adherence to a clock or timing protocol.
  - Reference all signals to a clock
  - Input signals are specified with required setup and hold times relative to the clock, and outputs have delays related to the edges of the clock.

14.2.6 Design Principles Summary

- There are strong parallels between the methods of design for software and hardware systems. The table summarizes some of these parallels for the principles outlined above.

TABLE 14.1 Structure software and VLSI hardware design

Design Principle	Software	Hardware
Hierarchy	Subroutines, libraries	Modules
Regularity	Iteration, code sharing, object-oriented procedures	Datapaths, module reuse, regular arrays, gate arrays, standard cells
Modularity	Well-defined subroutine interfaces	Well-defined module interfaces, timing and loading data for modules, registered inputs and outputs
Locality	Local scoping, no global variables	Local connections through floorplanning

Design Methods

- The base design methods are arranged roughly in order of "increased investment," which loosely relates to the time and cost it takes to design and implement the system.
- It is important to understand the costs, capabilities, and limitations of a given implementation technology to select the right solution.

Design Method	Non-Recurring Engineering	Unit Cost	Power Dissipation	Complexity of Implementa-tion	Time to Market	Perfor-mance	Flexibility
Microprocessor/DSP	Low	Medium	High	Low	Low	Low	High
PLD	Low	Medium	Medium	Low	Low	Medium	Low
FPGA	Low	Medium	Medium	Medium	Low	High	High
Cell-Based	High	Low	Low	High	High	High	Low
Custom Design	High	Low	Low	High	High	Very High	Low
Platform-Based	High	Low	Low	High	High	High	Medium

14.3.1 Microprocessor/DSP

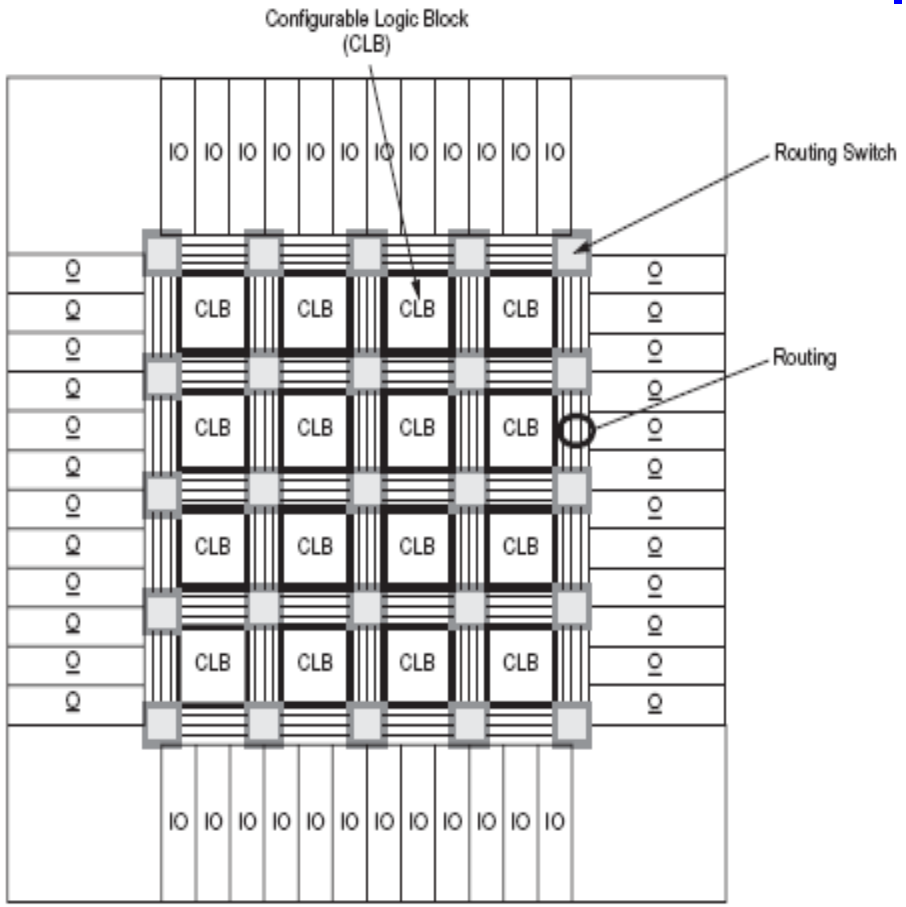
- **Microprocessor/DSP** The practical method to solve a system design problem: use microprocessor or digital signal processor (DSP).
  - Example: PIC family
  - Examples of embedded commercial processor cores include ARM, MIPS, and IBM's PowerPC.

14.3.2 Programmable Logic

- **Programmable Logic:** a variety of programmable chips are efficient than MPs yet faster to develop than dedicated chips:
  - Chips with programmable logic arrays
  - Chips with programmable interconnect
  - Chips with reprogrammable logic and interconnect
- The designer should be familiar with these options for two reasons:
  - Allows the designer to competently assess a particular system requirement for an IC and recommend a solution.
  - Familiarizes the IC designer with methods of making any chip reprogrammable.

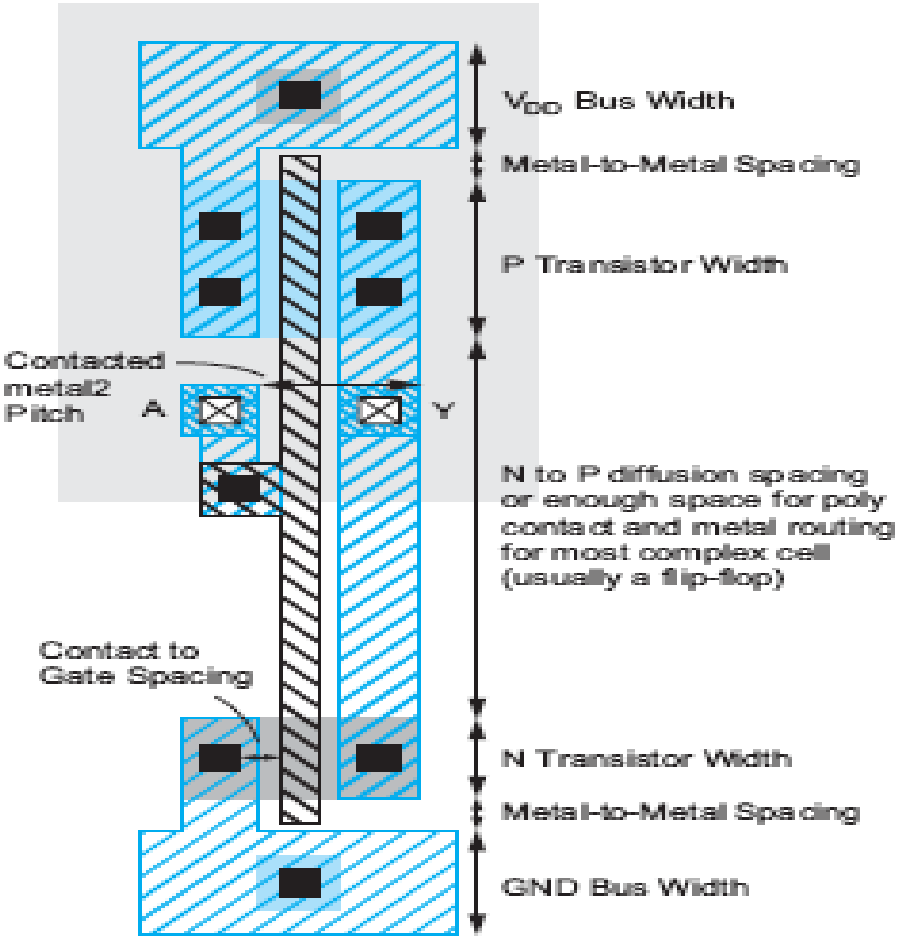
14.3.2 Programmable Logic

- **Programmable Logic Devices:** The devices are descended from chips that implement two-level sum-of-product (AND plane and OR plane to compute any function) programmable logic arrays (PLAs).
- **Field-Programmable Gate Arrays (FPGAs):** use the high circuit densities in modern processes to construct ICs that are completely programmable.



14.3.2 Programmable Logic

- **Cell-Based Design:** uses a standard cell library as the basic building blocks of a chip. Cells are placed in appropriate positions, then their interconnections are routed. It can deliver smaller, faster, and lower-power chips than FPGAs.



14.3.2 Programmable Logic

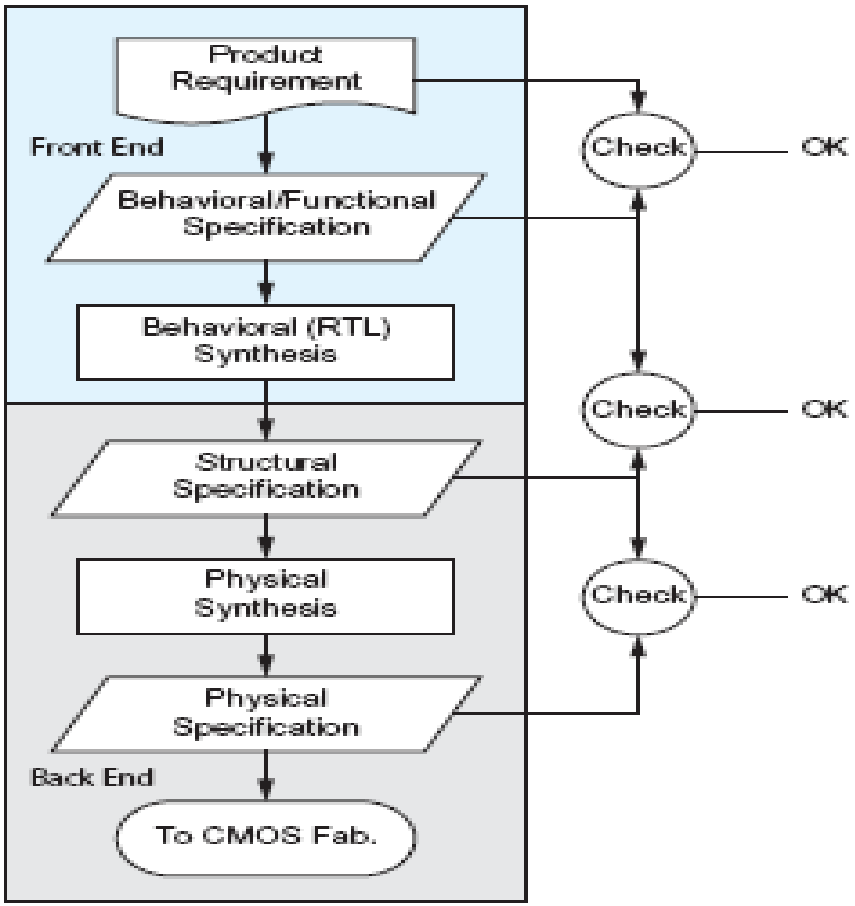
- **Full Custom Design**
- **Platform-based Design- A System on a Chip:** Implement a design by using common structures such as busses and common high-level languages (such as C) to program the processors.

TABLE 14.4 Comparison of CMOS design methods

Design Method	Non-Recurring Engineering	Unit Cost	Power Dissipation	Complexity of Implementation	Time to Market	Performance	Flexibility
Microprocessor/DSP	Low	Medium	High	Low	Low	Low	High
PLD	Low	Medium	Medium	Low	Low	Medium	Low
FPGA	Low	Medium	Medium	Medium	Low	High	High
Cell-Based	High	Low	Low	High	High	High	Low
Custom Design	High	Low	Low	High	High	Very High	Low
Platform-Based	High	Low	Low	High	High	High	Medium

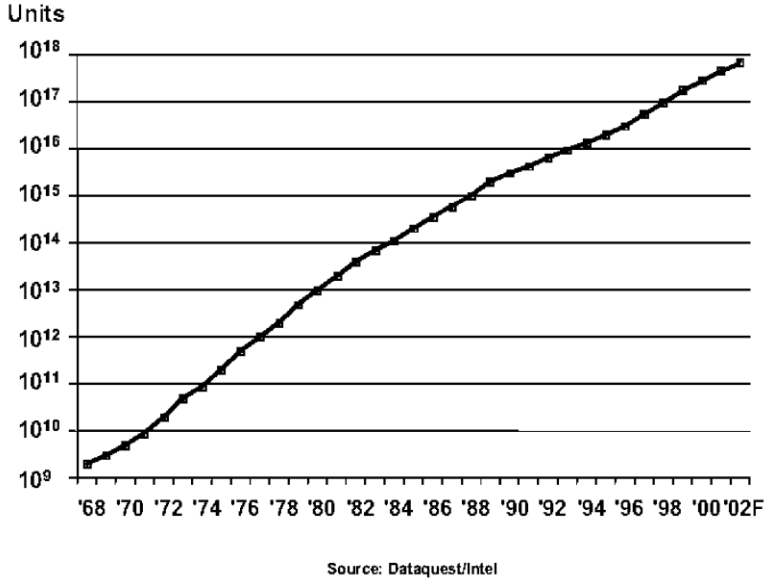
14.4 Design Flow

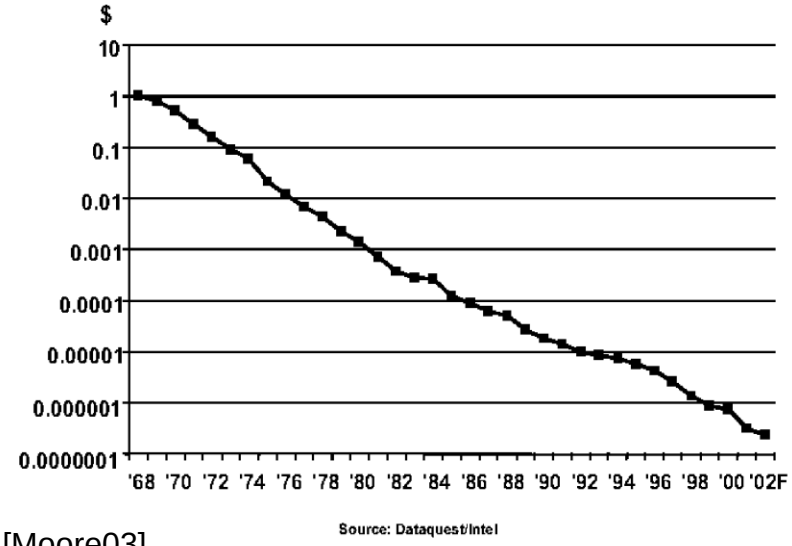
- **Design flow** is a set of procedures that allows designers to progress from a specification for a chip to the final chip implementation in an error-free way.



14.5 Design Economics

- It is important for the IC designer to be able to predict the cost and the time to design a particular IC or sets of ICs. This can guide the choice of an implementation strategy.
- In 2003, $0.01 bought you 100,000 transistors
  - Moore's Law is still going strong





[Moore03]

Physical Limits

- Will Moore's Law run out of steam?
  - Can't build transistors smaller than an atom…
- Many reasons have been predicted for end of scaling
  - Dynamic power
  - Subthreshold leakage, tunneling
  - Short channel effects
  - Fabrication costs
  - Electromigration
  - Interconnect delay
- Rumors of demise have been exaggerated

VLSI Economics

- Selling price S_{total}
  - $S_{total} = C_{total} / (1-m)$
- m = profit margin
- C_{total} = total cost
  - Nonrecurring engineering cost (NRE)
  - Recurring cost
  - Fixed cost

14.5.1 NRE

- Engineering cost
  - Depends on size of design team
  - Include benefits, training, computers
  - CAD tools:
    - Digital front end: $10K
    - Analog front end: $100K
    - Digital back end: $1M
- Prototype manufacturing
  - Mask costs: $5M in 45 nm process
  - Test fixture and package tooling

14.5.2 Recurring Costs

- Once the cost of an IC has been determined, the IC manufacturer will arrive at a price for the specific IC.
- the cost to fabricate an IC is as follows:

$$R_{total} = R_{process} + R_{package} + R_{test}$$

where

- $R_{package}$ = *package cost*
- R_{test} = *test cost—the cost to test an IC is usually proportional to number of vectors* and time to test.
- $R_{process} = W/(N \times Y_w \times Y_{pa})$

where

14.5.2 Recurring Costs

- *W = wafer cost ($500–$5000 depending on process and wafer size)*
- *N = gross die per wafer (the number of complete die on a wafer)*
- *Y_w = die yield per wafer (should be ~70–90+% for moderate-sized dice in a mature* process)
- *Y_{pa} = packaging yield (should be ~95–99%)*
- Note that:
  - Wafer cost / (Dice per wafer * Yield)
  - Wafer cost: $500 - $3000

14.5.2 Recurring Costs

- Dice per wafer: $N = \pi\left[\frac{r^2}{A} - \frac{2r}{\sqrt{2A}}\right]$

Where die area *A and is fabricated on a wafer with radius r.*

- Yield: $Y = e^{-AD}$
  - *A is* the size of the die *and D is the average number of defects per unit area.*
  - For small A, $Y \approx 1$, cost proportional to area
  - For large A, $Y \rightarrow 0$, cost increases exponentially

14.5.3 Fixed Costs

- Once a chip has been designed and put into manufacture, the cost to support that chip from an engineering viewpoint may have a few sources:
  - Data sheets
  - Application notes
  - Marketing and advertising
  - Yield analysis

Example

- You want to start a company to build a wireless communications chip. How much venture capital must you raise?

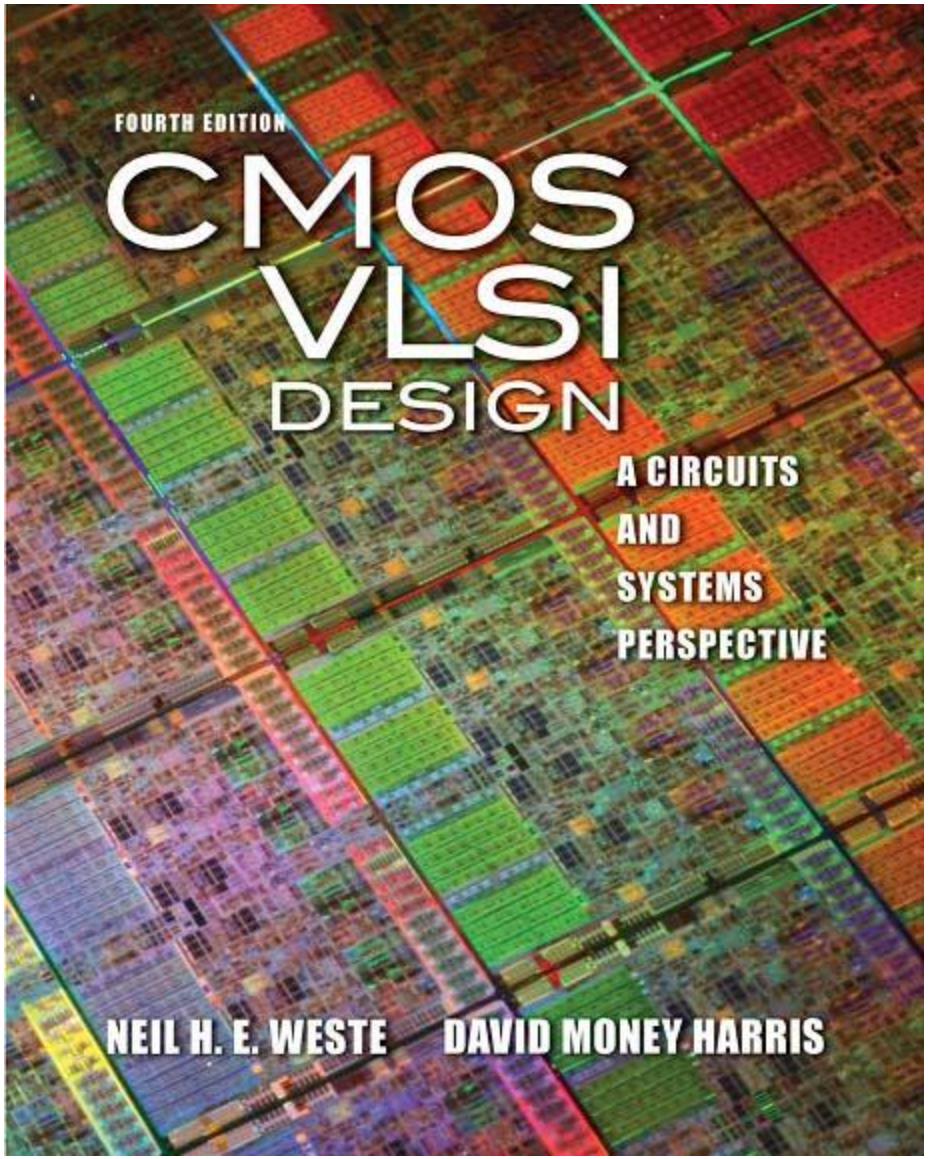
- Because you are smarter than everyone else, you can get away with a small team in just two years:
  - Seven digital designers
  - Three analog designers
  - Five support personnel

Solution

- Digital designers:
  - $70k salary
  - $30k overhead
  - $10k computer
  - $10k CAD tools
  - Total: $120k * 7 = $840k
- Analog designers
  - $100k salary
  - $30k overhead
  - $10k computer
  - $100k CAD tools
  - Total: $240k * 3 = $720k
- Support staff
  - $45k salary
  - $20k overhead
  - $5k computer
  - Total: $70k * 5 = $350k
- Fabrication
  - Back-end tools: $1M
  - Masks: $5M
  - Total: $6M / year
- Summary
  - 2 years @ $7.91M / year
  - $16M design & prototype

CPE 110408423
VLSI Design
Chapter 13: Special-Purpose Subsystems

Bassam Jamil
[Computer Engineering Department,
Hashemite University]



Lecture 5: Packaging, Power, & Clock

13.1 Outline

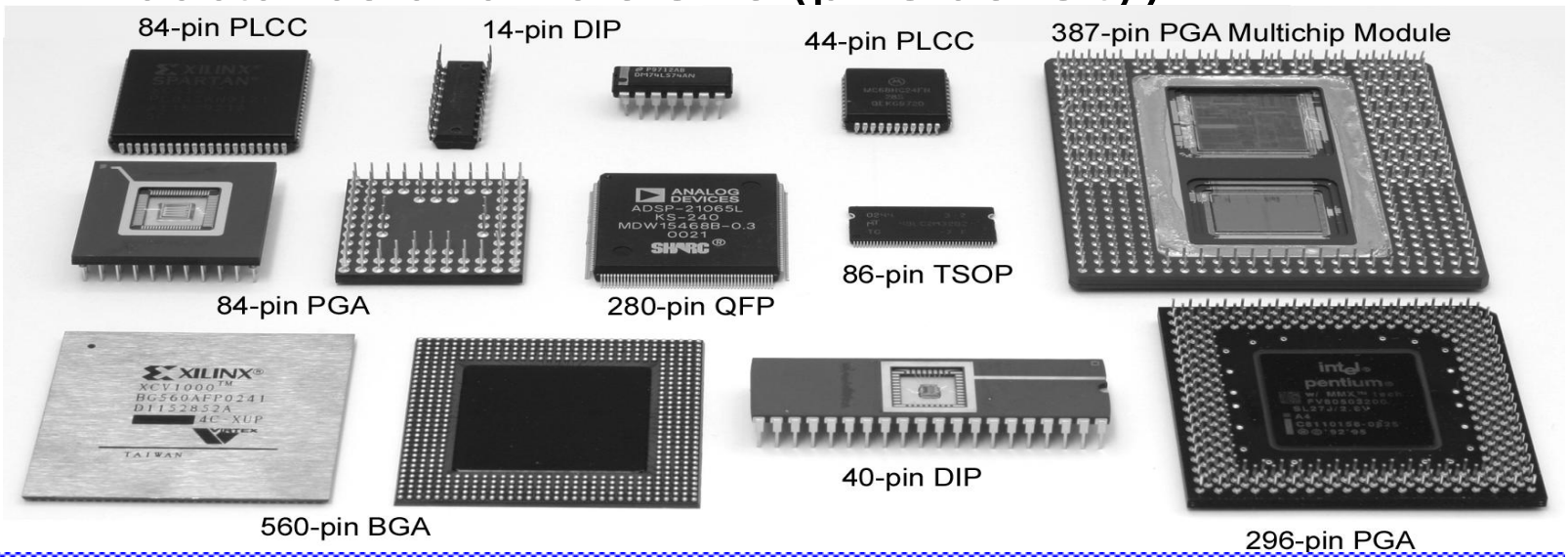
- Packaging and cooling
- Power Distribution
- Clock Distribution
- We will limit our discussion to:
  - Package types
  - Heat
  - Power Distribution:
  - 1. Power Supply drop:
    - IR drop
    - I di/dt
  - 2. Bypass capacitor
    - I = c dv/dt

13.2 Packaging and Cooling

- Package functions
  - Electrical connection of signals and power from chip to board
    - Little delay or distortion
  - Mechanical connection of chip to board
    - Removes heat produced on chip
    - Protects chip from mechanical damage
    - Compatible with thermal expansion
  - Inexpensive to manufacture and test

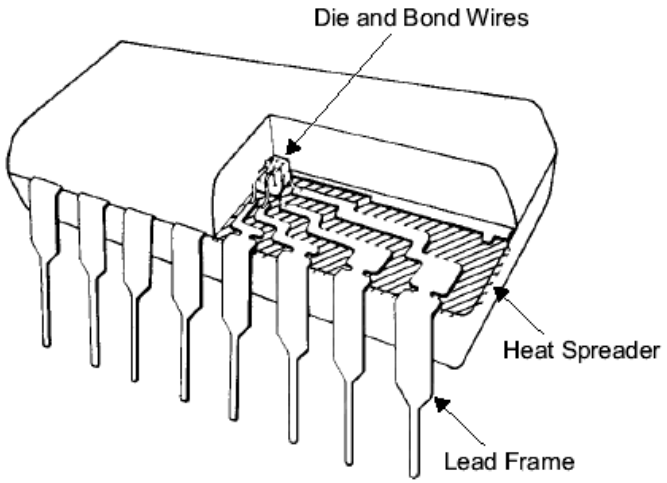
Package Types

- *Through-hole* pass through holes in a printed circuit board and are soldered from below.
- *Surface mount (SMT) packages* are soldered to the surface of a printed circuit board to alleviate pin inductance and hole size (pins density).



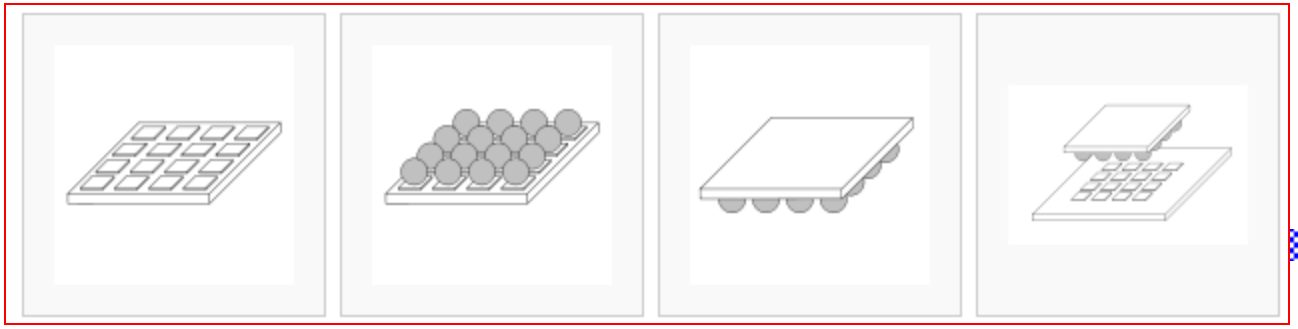
Chip-to-Package Bonding

- Traditionally, chip is surrounded by *pad frame*
  - Metal pads on 100 – 200 μm pitch (hundreds I/O)
  - Gold *bond wires* attach pads to package
  - *Lead frame* distributes signals in package
  - Metal *heat spreader* helps with cooling



Advanced Packages

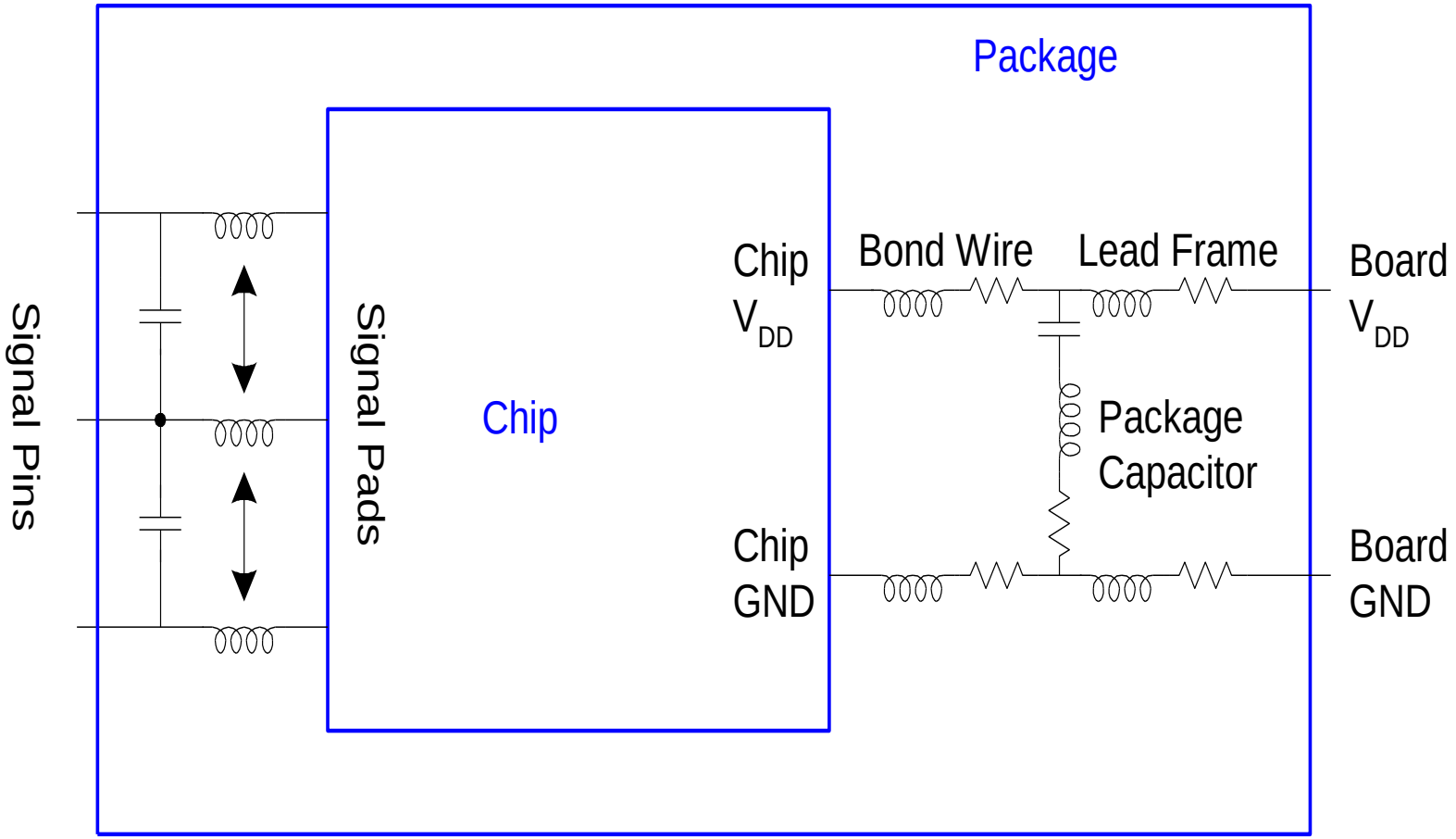
- Bond wires contribute parasitic inductance
- Fancy packages have many signal, power layers
  - Like tiny printed circuit boards
- *Flip-chip* places connections across surface of die rather than around periphery
  - Top level metal pads covered with solder balls
  - Chip flips upside down
  - Carefully aligned to package (done blind!)
  - Heated to melt balls
  - Also called *C4* (Controlled Collapse Chip Connection)



Package Parasitics

- Bond wires and lead frame contribute parasitic inductance to the signal traces.
- Inductive, capacitive coupling.
- Use many V_{DD}, GND in parallel
  - Inductance, I_{DD}

Package Parasitics



Heat Dissipation

- 60 W light bulb has surface area of 120 cm^2
- Itanium 2 die dissipates 130 W over 4 cm^2
  - Chips have **enormous power densities**
  - Cooling is a serious challenge
- Package spreads heat to larger surface area
  - Heat sinks may increase surface area further
  - Fans increase airflow rate over surface area
  - Liquid cooling used in extreme cases ($$$)

Thermal Resistance

- $\Delta T = \theta_{ja} P$
  - ΔT: temperature rise on chip
  - θ_{ja}: thermal resistance of chip junction to ambient (C/W)
  - P: power dissipation on chip
- Thermal resistances combine like resistors
  - Series and parallel
- $\theta_{ja} = \theta_{jp} + \theta_{pa}$
  - Series combination from the die to the package θ_{jp} and from package to the air θ_{pa}.

Heat Example

- Your chip has a heat sink with a thermal resistance to the package of 4.0° C/W.
- The resistance from chip to package is 1° C/W.
- The system box ambient temperature may reach 55° C.
- The chip temperature must not exceed 100° C.
- What is the maximum chip power dissipation?
- Solution:
  - $\theta_{ja} = \theta_{jp} + \theta_{pa} = (1 + 4) = 5$ C/W
  - $\Delta T = \theta_{ja} P$
  - (100-55 C) = 5 P
  - ➔ P=9W

Temperature Sensor

- Monitor die temperature and throttle performance if it gets too hot
  - **T**: absolute temp, **I_c** collector current, base emitter voltage **V_{BE}**

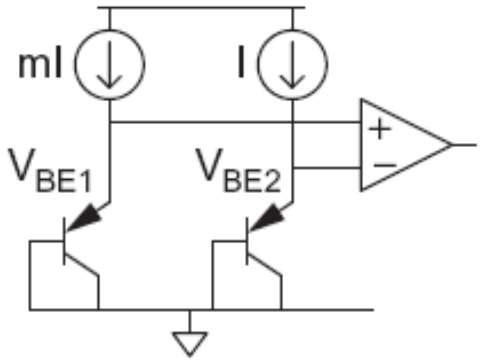
$$I_c = I_s e^{\frac{qV_{BE}}{kT}} \rightarrow V_{BE} = \frac{kT}{q} \ln \frac{I_c}{I_s}$$

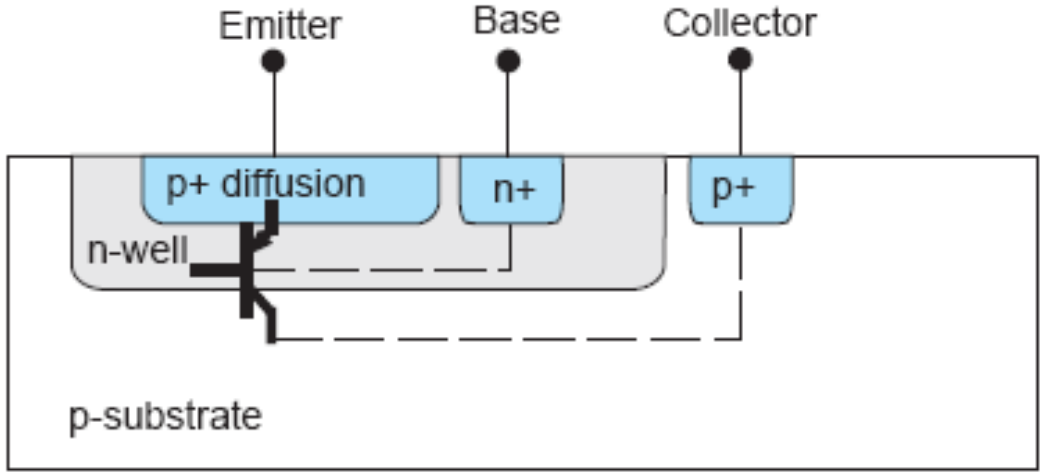
$$\Delta V_{BE} = V_{BE1} - V_{BE2} = \frac{kT}{q}\left(\ln \frac{I_{c1}}{I_s} - \ln \frac{I_{c2}}{I_s} \right) = \frac{kT}{q}\left(\ln \frac{I_{c1}}{I_{c2}} \right) = \frac{kT}{q} \ln m$$

  - q: charge on electron, k: Boltzmann constant, Is: function of transistor geometry and processing.

Temperature Sensor

- Use a pair of pnp bipolar transistors
  - Vertical pnp available in CMOS





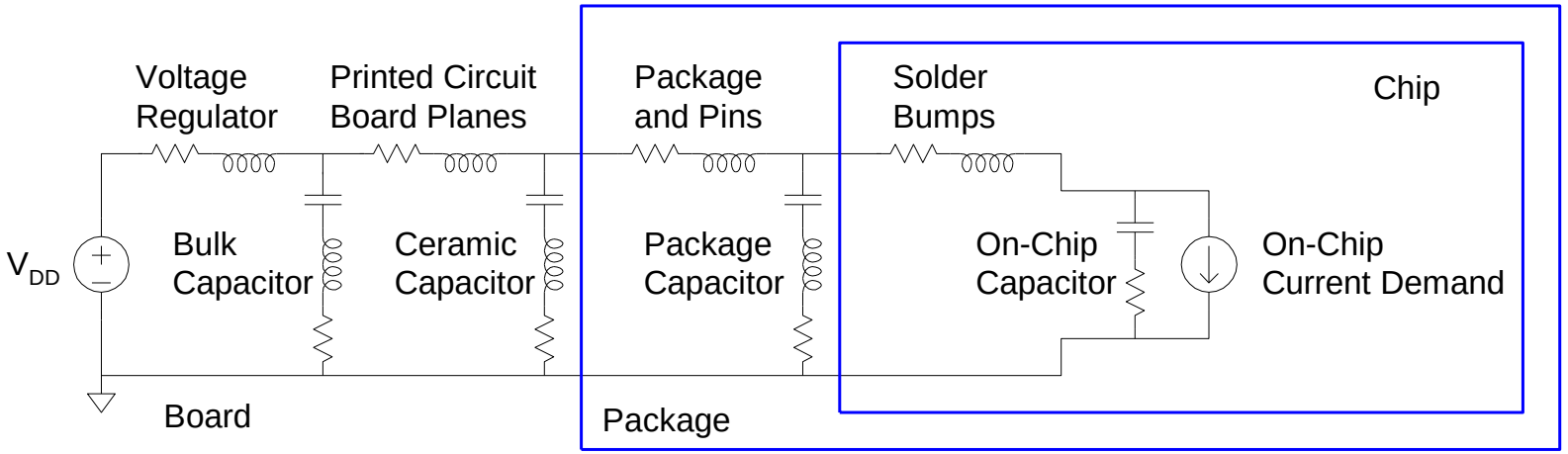
- **Voltage difference is proportional to absolute temp**
  - Measure with on-chip A/D converter

13.3 Power Distribution

- Power Distribution Network functions
  - Carry current from pads to transistors on chip
  - Maintain stable voltage with low noise
  - Provide average and peak power demands
  - Provide current return paths for signals
  - Avoid electromigration & self-heating wearout
  - Consume little chip area and wire
  - Easy to lay out

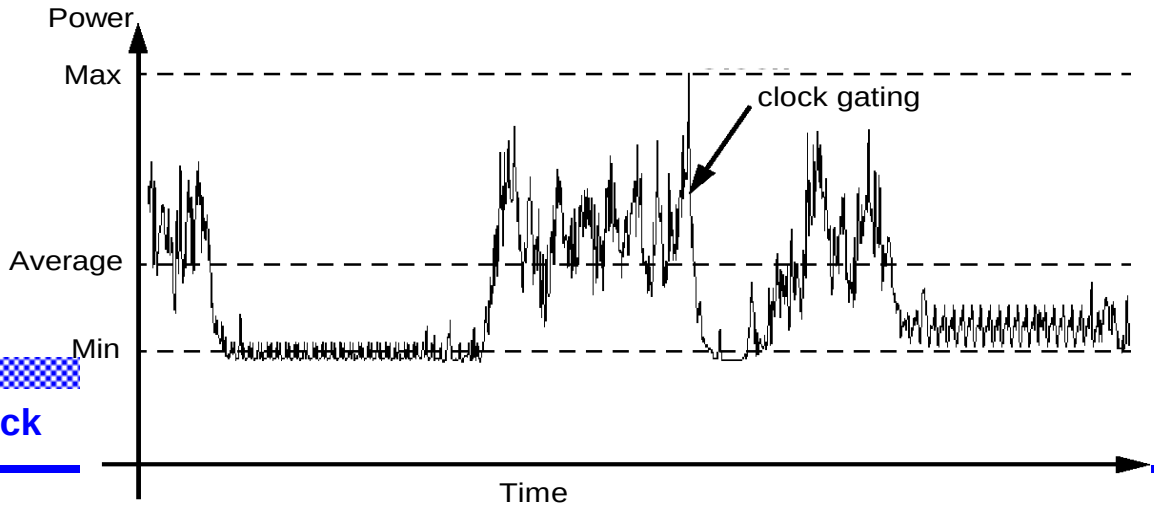
Power System Model

- Power comes from regulator on system board
  - Board and package add parasitic R and L
  - Bypass capacitors help stabilize supply voltage
  - But capacitors also have parasitic R and L
- Simulate system for time and frequency responses



Power Requirements

- $V_{DD} = V_{DDnominal} - V_{drop}$
- Want V_{droop} < +/- 10% of V_{DD}
- Sources of V_{drop}
  - IR drops (Caused by the resistance of distribution network)
  - L di/dt noise: caused by the inductance of the distribution network)
- I_{DD} changes on many time scales



IR Drop

- IR drop:
  - R: resistance of the power distribution network
  - I : is the draw current
  - Instantaneous drawing of current I causes voltage drop in the power supply equals to IR drop.
- IR Drop = I_{DD} $_{\text{Power Supply}}$ × Impedance $_{\text{Power Supply}}$
- A chip draws 24 W from a 1.2 V supply. The power supply impedance is 5 mΩ. What is the IR drop?
  - **I_{DD} = 24 W / 1.2 V = 20 A**
  - **IR drop = (20 A)(5 mΩ) = 100 mV**

L di/dt Noise

- The change in current flows through the inductance of the printed circuit board and package causes L di/dt
- The drop in the power supply due to drawing current
  - $V(t)_{\text{drop in power supply}} = L \times di / dt$
- A 1.2 V chip switches from an idle mode consuming 5W to a full-power mode consuming 53 W. The transition takes 10 clock cycles at 1 GHz. The supply inductance is 0.1 nH. What is the L di/dt droop?
- **$\Delta I = \Delta W/V = (53\ W - 5\ W)/(1.2\ V) = 40\ A$**
- **$\Delta t = 10$ cycles * (1 ns / cycle) = 10 ns**
- **L di/dt drop = (0.1 nH) * (40 A / 10 ns) = 0.4 V**

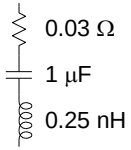
Bypass Capacitors

- Need low supply impedance at all frequencies
- Ideal capacitors have impedance decreasing with ω
- Real capacitors have parasitic R and L
  - Leads to resonant frequency of capacitor

$$Z = \frac{1}{j\omega C} + R + j\omega L$$

This impedance has a minimum of $Z = R$ at the self-resonant frequency of

$$f_{\text{resonant}} = \frac{\omega_{\text{resonant}}}{2\pi} = \frac{1}{2\pi\sqrt{LC}}$$



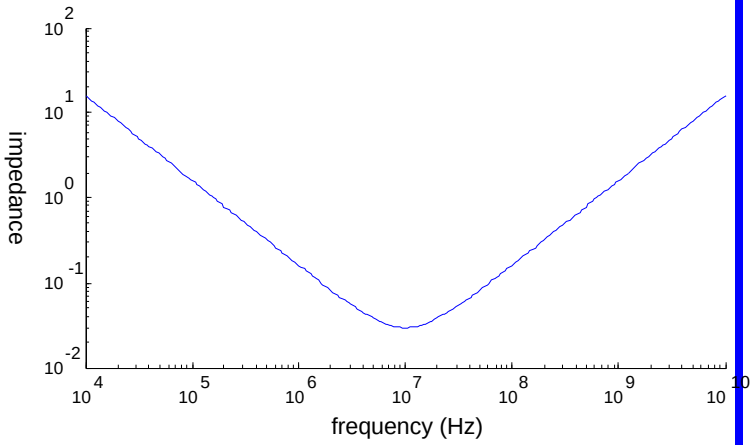


Figure 13.11 plots the magnitude of the impedance of a 1 μF capacitor with 0.25 nH of series inductance and 0.03 Ω of series resistance. The capacitor has low impedance near its resonant frequency of 10 MHz, but higher impedance elsewhere.

Bypass Capacitors

Example:

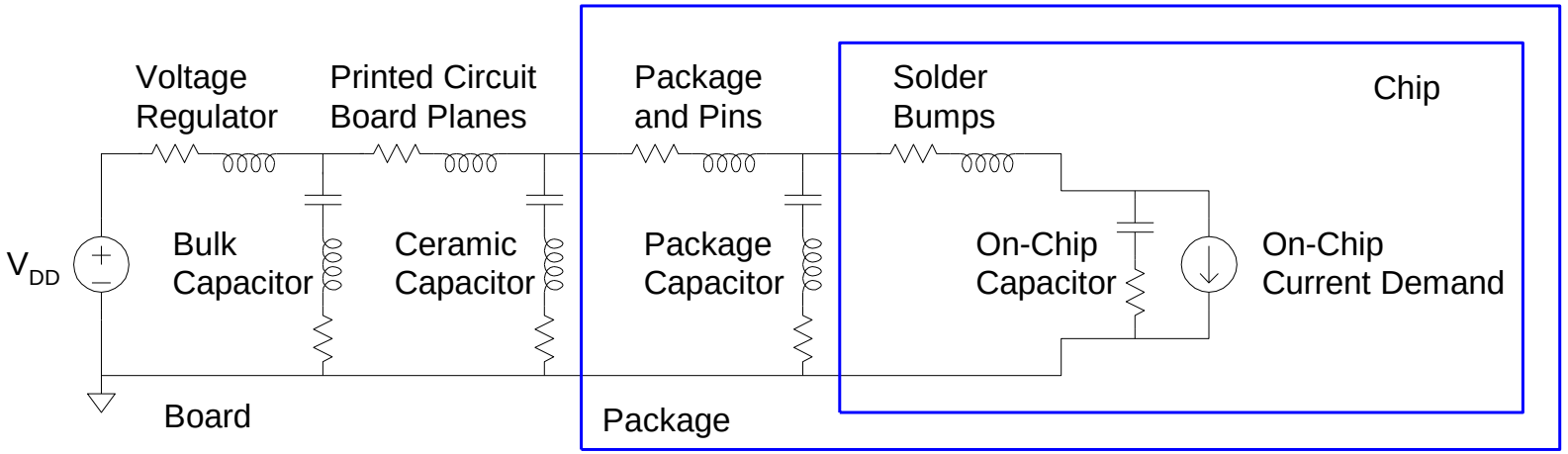
How much bypass capacitance is needed to supply a sudden current spike of 40 A for 1 ns with no more than a 200 mV supply droop?

SOLUTION: We solve

$$I = C\frac{\Delta V}{\Delta t} \Rightarrow C = \frac{(40\text{ A})(1\text{ ns})}{0.2\text{ V}} = 200\text{ nF}$$

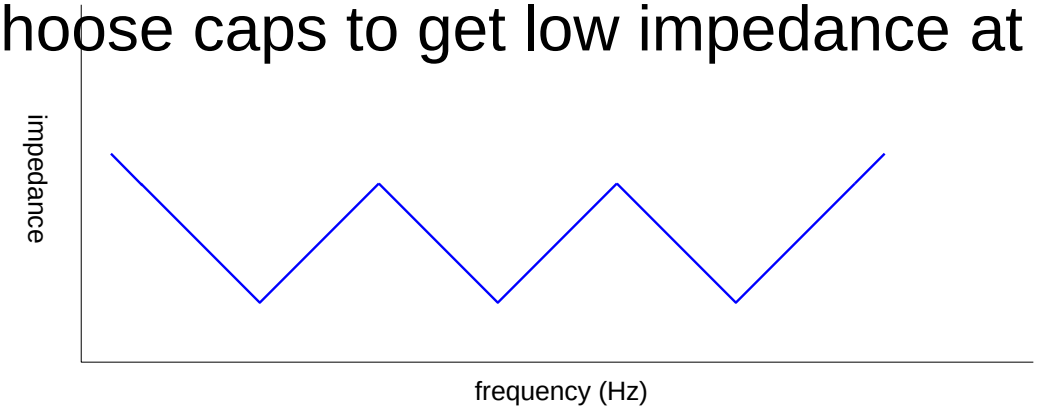
Power System Model

- Power comes from regulator on system board
  - Board and package add parasitic R and L
  - Bypass capacitors help stabilize supply voltage
  - But capacitors also have parasitic R and L
- Simulate system for time and frequency responses



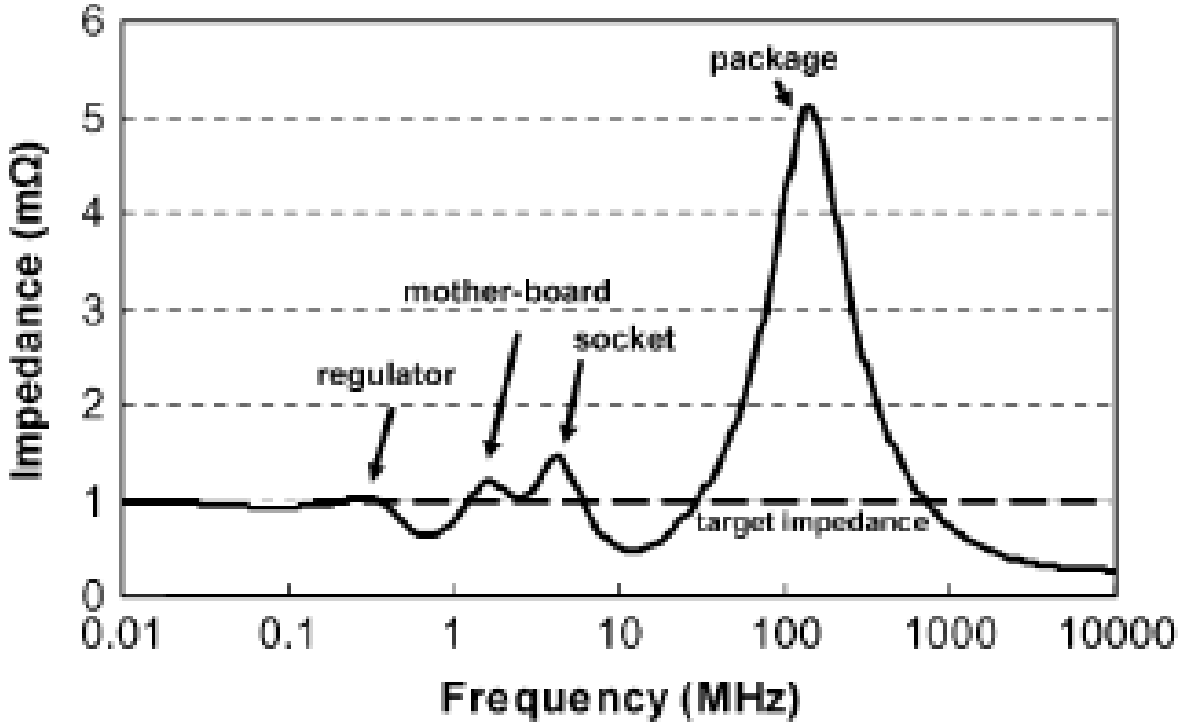
Frequency Response

- Multiple capacitors in parallel
  - Large capacitor near regulator has low impedance at low frequencies
  - But also has a low self-resonant frequency
  - Small capacitors near chip and on chip have low impedance at high frequencies
- Choose caps to get low impedance at all frequencies



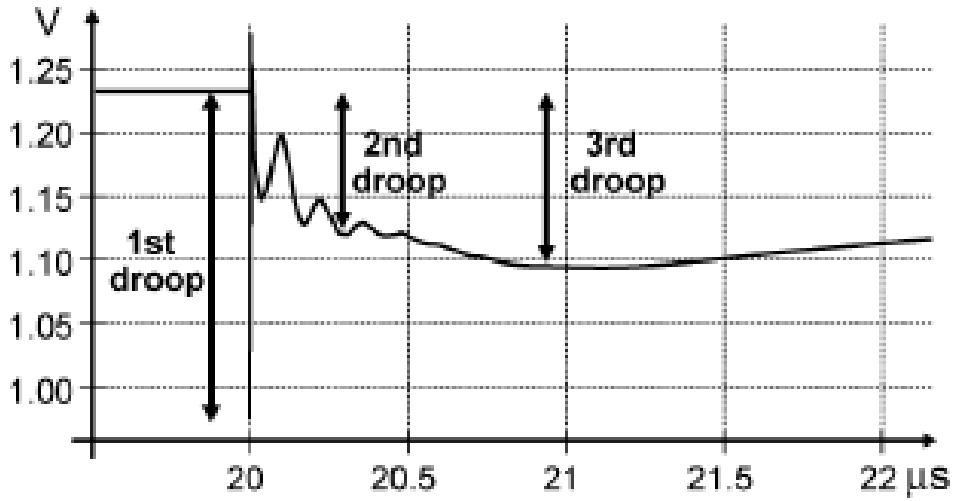
Example: Pentium 4

- Power supply impedance for Pentium 4
  - Spike near 100 MHz caused by package L



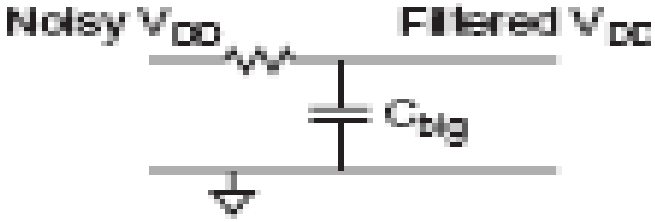
Example: Pentium 4

- Step response to sudden supply current chain
  - 1st droop: on-chip bypass caps
  - 2nd droop: package capacitance
  - 3rd droop: board capacitance



Power Supply Filtering

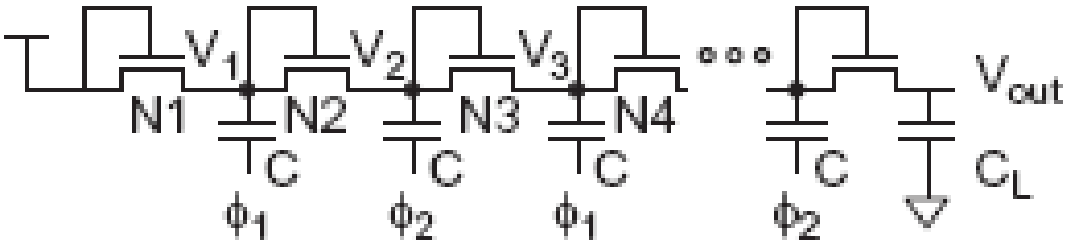
- Certain structures such as the phase-locked loop (PLL), clock buffers, and analog circuits are particularly sensitive to power supply noise.
- An RC power supply filter circuit that eliminates the high-frequency noise on the local supply.



- The resistance of this wire must be low enough to carry the current demand of the local circuitry without excessive IR drop, yet low enough to produce an RC time constant that will filter noise at frequencies of interest.

Charge Pumps

- Sometimes a different supply voltage is needed but little current is required
  - 20 V for Flash memory programming
  - Negative body bias for leakage control during sleep
- Generate the voltage on-chip with a charge pump



$$V_{out} = N\left[\frac{CV_{DD} - \frac{I_{out}}{f}}{C + C_s} - V_t\right]$$

Energy Scavenging

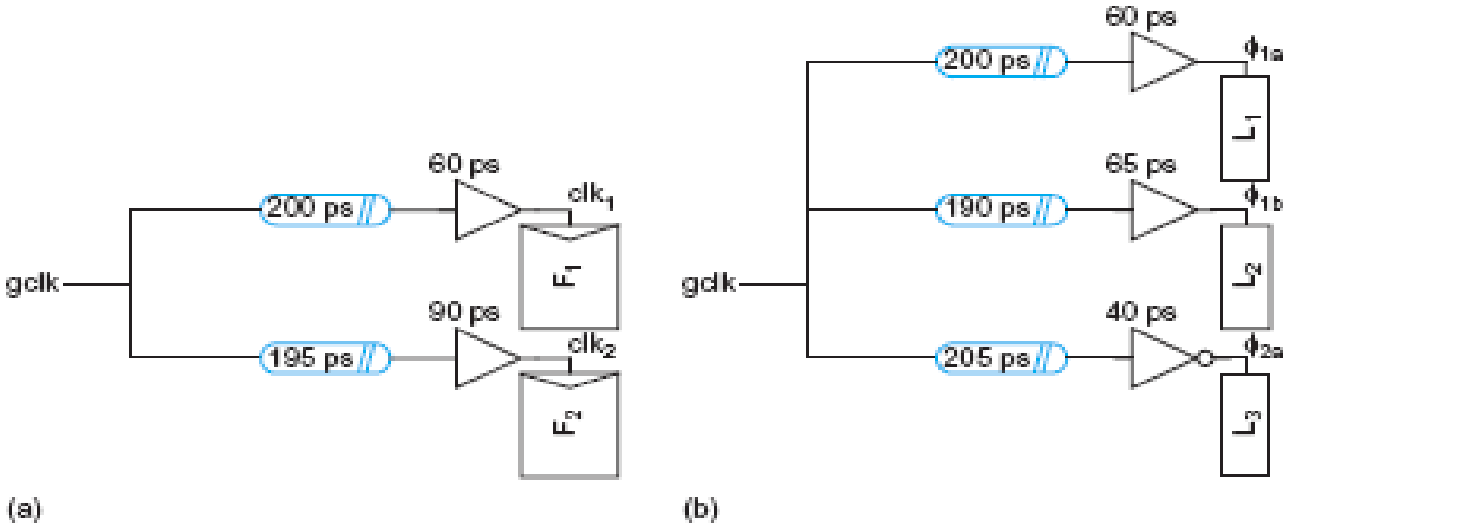
- Ultra-low power systems can scavenge their energy from the environment rather than needing batteries
  - Solar calculator (solar cells)
  - RFID tags (antenna)
  - Tire pressure monitors powered by vibrational energy of tires (piezoelectric generator)
- Thin film microbatteries deposited on the chip can store energy for times of peak demand

13.4 Clock Distribution

- On a small chip, the clock distribution network is just a wire
  - And possibly an inverter for clkb
- On practical chips, the RC delay of the wire resistance and gate load is very long
  - Variations in this delay cause clock to get to different elements at different times
  - This is called *clock skew*
- Most chips use repeaters to buffer the clock and equalize the delay
  - Reduces but doesn't eliminate skew

Clock Skew

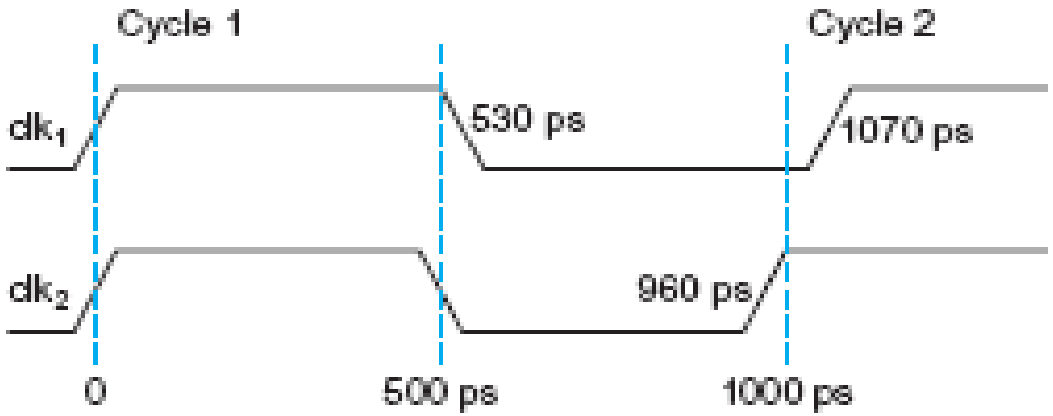
- Skew is the *difference between the nominal and actual inter-arrival time of a pair of physical clocks.*



  - (a) clock skew is 25 ps
  - (b) $t_{skew}^{\phi_{1a}\phi_{1b}} = 5$ ps; $t_{skew}^{\phi_{1a}\phi_{2a}} = 15$ ps; $t_{skew}^{\phi_{1b}\phi_{2a}} = 10$ ps

Clock Skew

- Clock skew can also be measured between different edges of the clock or between different cycles.

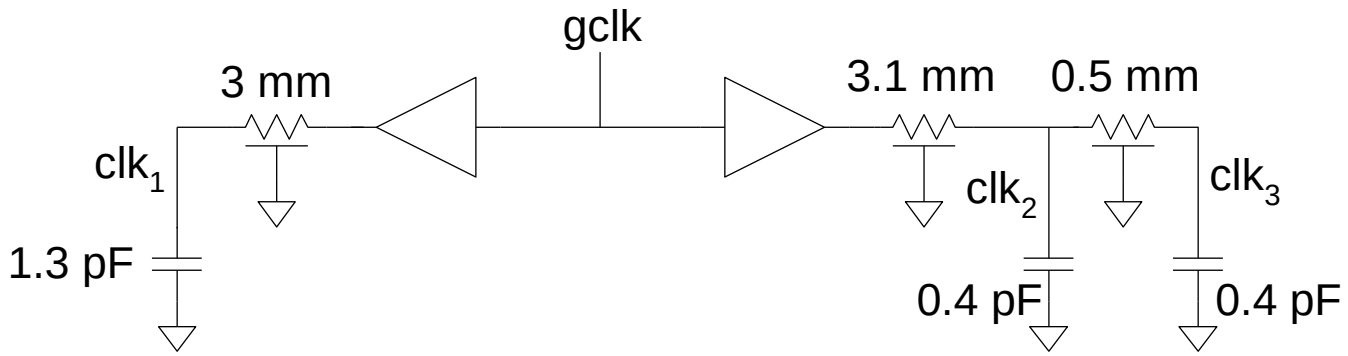


$$t_{skew}^{clk_1,clk_1}(r,r,0) = 0 \text{ ps}; \quad t_{skew}^{clk_1,clk_1}(r,f,0) = 30 \text{ ps}; \quad t_{skew}^{clk_1,clk_1}(r,r,1) = 70 \text{ ps}$$

$$t_{skew}^{clk_1,clk_2}(r,r,0) = 0 \text{ ps}; \quad t_{skew}^{clk_1,clk_2}(r,f,0) = 0 \text{ ps}; \quad t_{skew}^{clk_1,clk_2}(r,r,1) = 40 \text{ ps}$$

Example

- Skew comes from differences in gate and wire delay
  - With right buffer sizing, clk_1 and clk_2 could ideally arrive at the same time.
  - But power supply noise changes buffer delays
  - clk_2 and clk_3 will always see RC skew



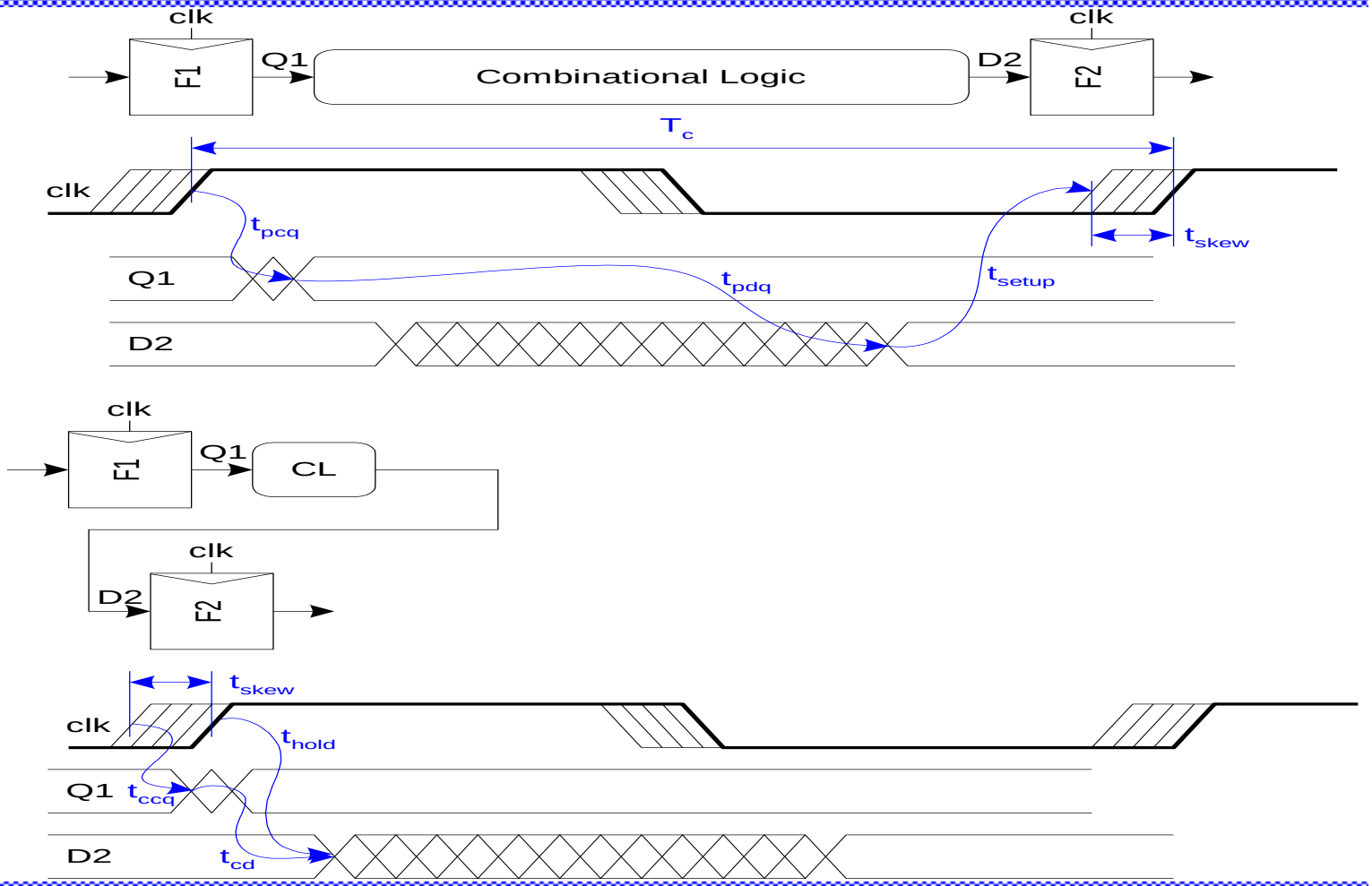
Review: Skew Impact (1)

- Ideally full cycle is available for work
- Skew adds sequencing overhead
- Increases hold time too

$$t_{pd} \leq T_c - \underbrace{\left(t_{pcq} + t_{\text{setup}} + t_{\text{skew}} \right)}_{\text{sequencing overhead}}$$

$$t_{cd} \geq t_{\text{hold}} - t_{ccq} + t_{\text{skew}}$$

Review: Skew Impact (2)



Solutions

- Reduce clock skew
  - Careful clock distribution network design
  - Plenty of metal wiring resources
- Analyze clock skew
  - Only budget actual, not worst case skews
  - Local vs. global skew budgets
- Tolerate clock skew
  - Choose circuit structures insensitive to skew

Global Clock Dist. Networks

- *Ad hoc*
- Grids
- H-tree
- Hybrid

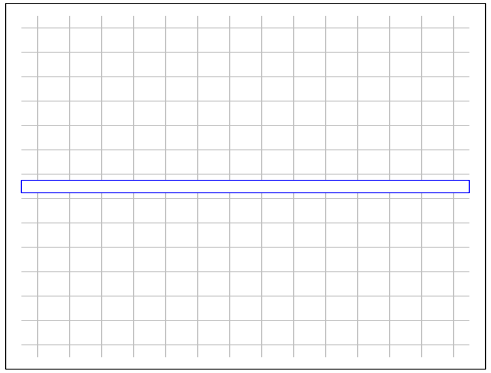
- Random skew (manufacture), drift (time-dependent environment variation : temperature), and jitter (high-frequency environment variation: power supply noise) from the clock distribution network are proportional to the delay through network because they are caused by process or environmental variations in the distribution elements. Designer should try to keep this distribution delay low.

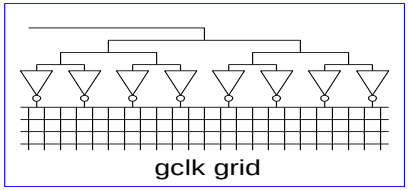
Clock Grids

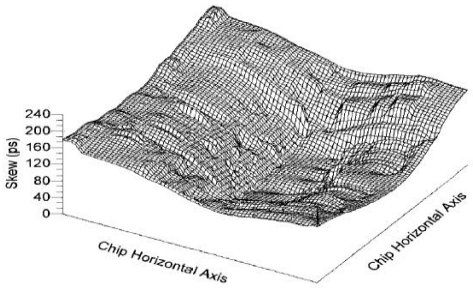
- Use grid on two or more levels to carry clock (mesh)
- Make wires wide to reduce RC delay
- Ensures low skew between nearby points
- But possibly large skew across die
- Grids can be routed early in the design without detailed knowledge of latch placement.
- However, grids do have significant systematic skew between the points closest to the drivers and the points farthest away. They also consume a large amount of metal resources and hence have a high switching capacitance and power consumption.

Alpha Clock Grids

Alpha 21064

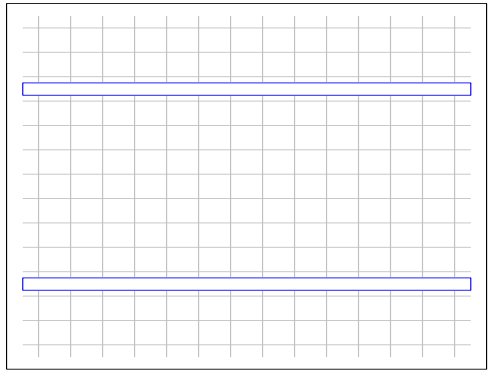


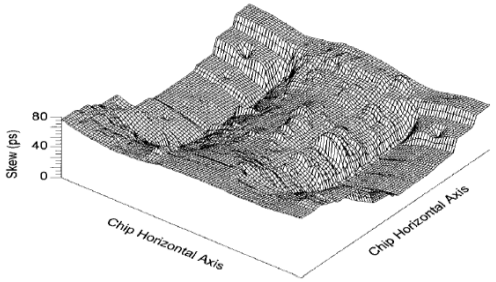




Alpha 21064

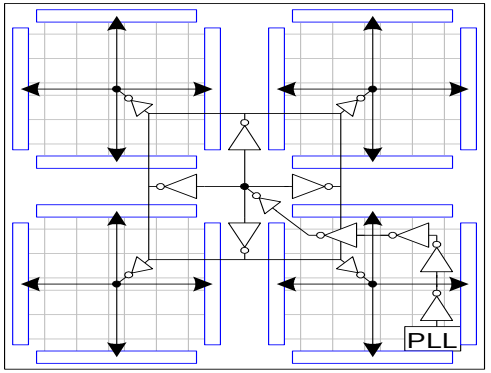
Alpha 21164

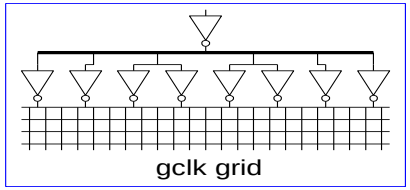


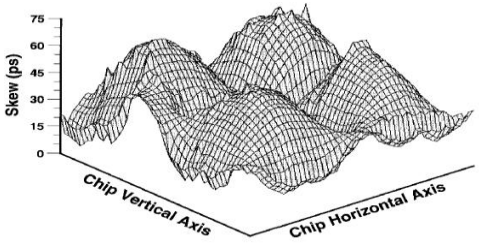


Alpha 21164

Alpha 21264



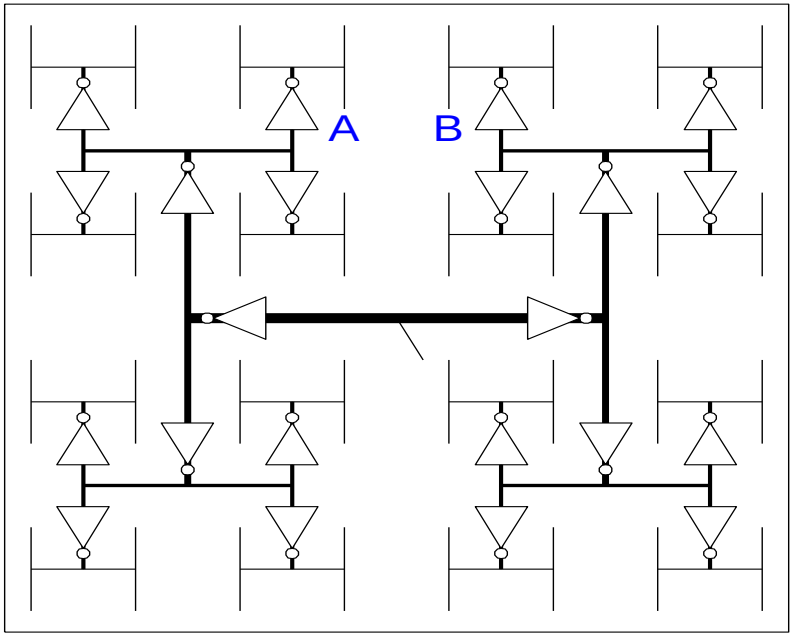




Alpha 21264

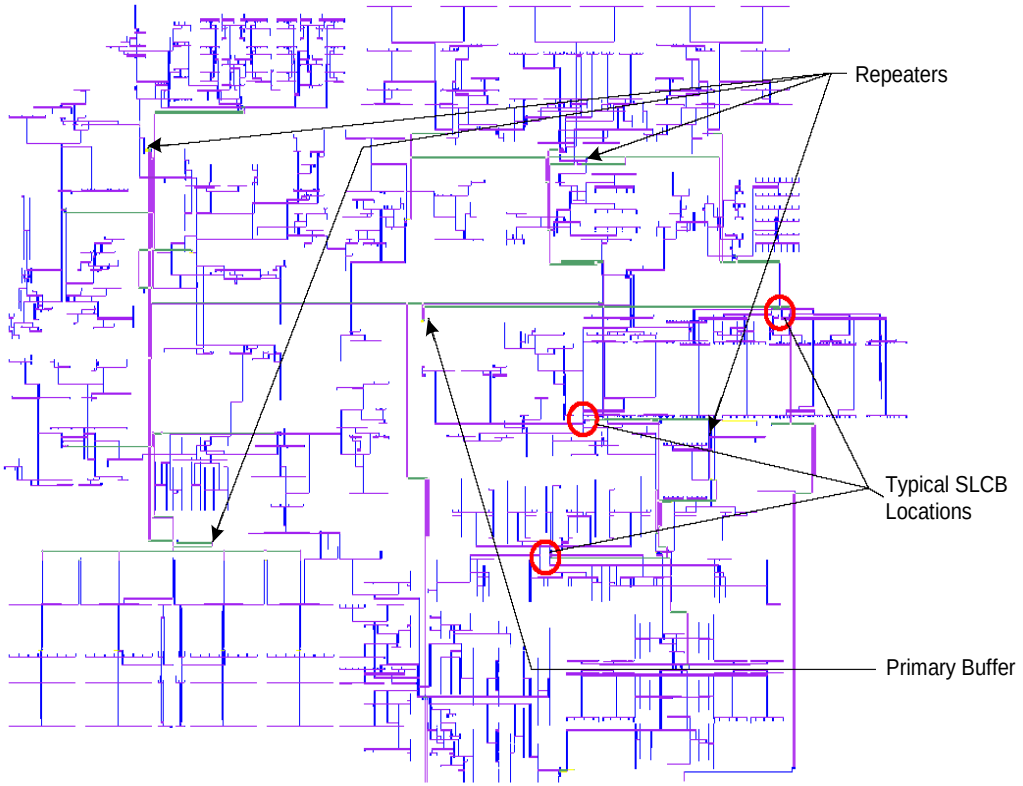
H-Trees

- Fractal structure has H-shape on each vertices
  - Gets clock arbitrarily close to any point
  - Matched delay along all paths
  - Buffers are added as repeaters
- Delay variations cause skew because load are not uniform.
- A and B might see big skew
- A drawback of H-trees is that they may have high random skew, drift, and jitter between two nearby points that are leaves of different legs of the tree.



Itanium 2 H-Tree

- Four levels of buffering:
  - Primary driver
  - Repeater
  - Second-level clock buffer
  - Gater
- Route around obstacles



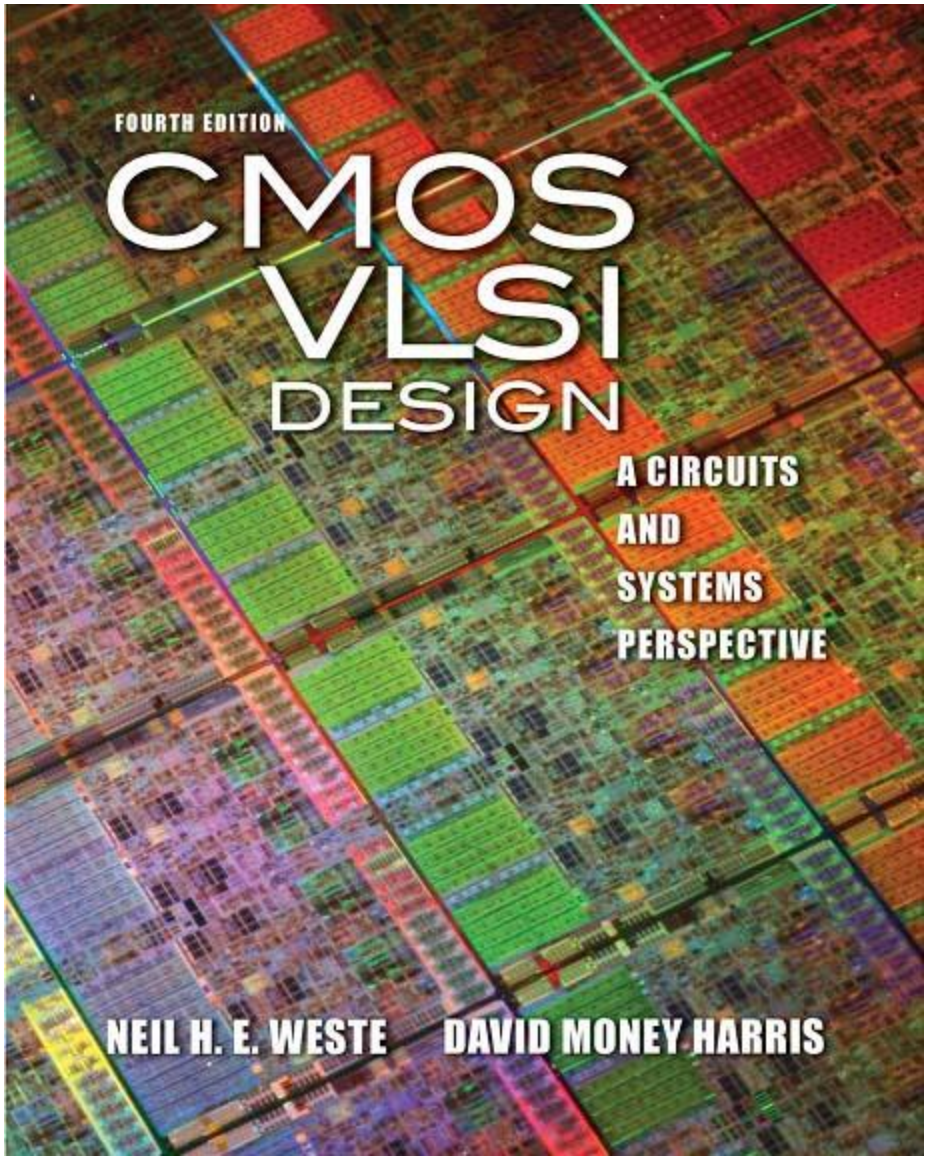
Hybrid Networks

- Use H-tree to distribute clock to many points
- Tie these points together with a grid

- Ex: IBM Power4, PowerPC
  - H-tree drives 16-64 sector buffers
  - Buffers drive total of 1024 points
  - All points shorted together with grid

CPE 110408423
VLSI Design
Chapter 15: Testing, Debugging, and Verification

Bassam Jamil
[Computer Engineering Department,
Hashemite University]



Lecture 9: Testing, Debugging, and Verification

Outline

- Introduction
  - Logic Verification
  - Silicon Debug
  - Manufacturing Test
- Fault Models
- Observability and Controllability
- Design for Test
  - Scan
  - BIST
- Boundary Scan

15.1 Introduction: Testing

- Testing is one of the most expensive parts of chips:
  - *Logic verification* accounts for > 50% of design effort for many chips (verify functionality)
  - *Debug* time after fabrication has enormous opportunity cost
  - *manufacturing tests: s*hipping defective parts can sink a company
- Example: Intel FDIV bug (1994)
  - Logic error not caught until > 1M units shipped
  - Recall cost $450M (!!!)

15.1 Introduction: Testing

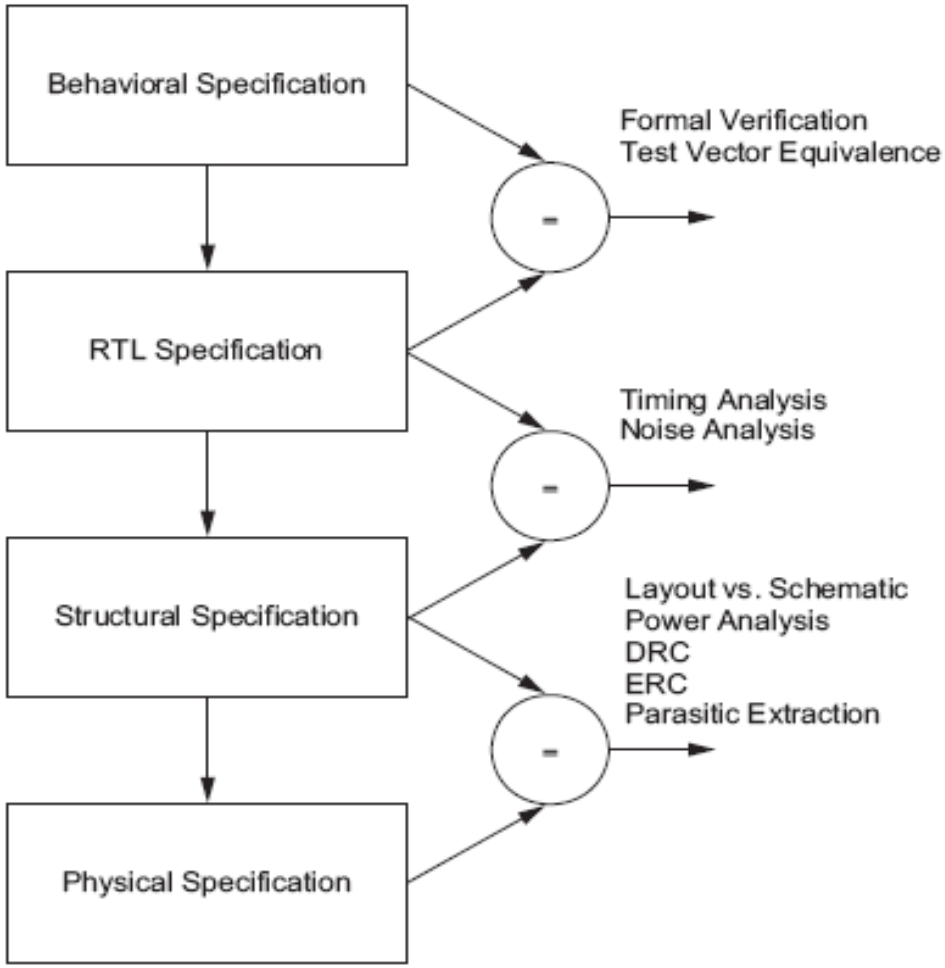
- Testing a die (chip) can occur at the following levels:
  - Wafer level
  - Packaged chip level
  - Board level
  - System level
  - Field level
- By detecting a malfunctioning chip early, the manufacturing cost can be kept low.
- Most failures of first-time silicon result from problems with the functionality of the design; i.e., the chip does exactly what the simulator said it would do, but for some reason (almost always human error) this functionality is not what the rest of the system expects.

Logic Verification

- Does the chip simulate correctly?
  - Usually done at HDL level
  - Verification engineers write test bench for HDL
    - Can't test all cases
    - Look for corner cases
    - Try to break logic design
- Ex: 32-bit adder
  - Test all combinations of corner cases as inputs:
    - 0, 1, 2, 2^{31}-1, -1, -2^{31}, a few random numbers
- Good tests require ingenuity

Logic Verification

- The amount of verification effort greatly exceeds the design effort.
- Whenever you are tempted to minimize verification effort to meet tight schedules: “If you don’t test it, it won’t work! (guaranteed).”



Silicon Debug

- Test the first chips back from fabrication
  - If you are lucky, they work the first time
  - If not… ????
- Power for the IC with ability to vary *VDD and measure power dissipation*
- Real-world signal connections (i.e., analog and digital inputs and outputs as required)
- Clock inputs as required (it is helpful to have a stable variable-frequency clock generator)
- A digital interface to a PC (either serial or parallel ports for slow data or PCI bus for fast data interchanges)

Silicon Debug

- An alternate or complementary approach is to provide interfaces for a logic analyzer.
- These are easily added to a PCB design in the form of **multi-pinned headers**.
- For the most part, if a digital chip simulates at the gate level and passes timing analysis checks during design, it will do exactly the same in silicon.

Manufacturing Test

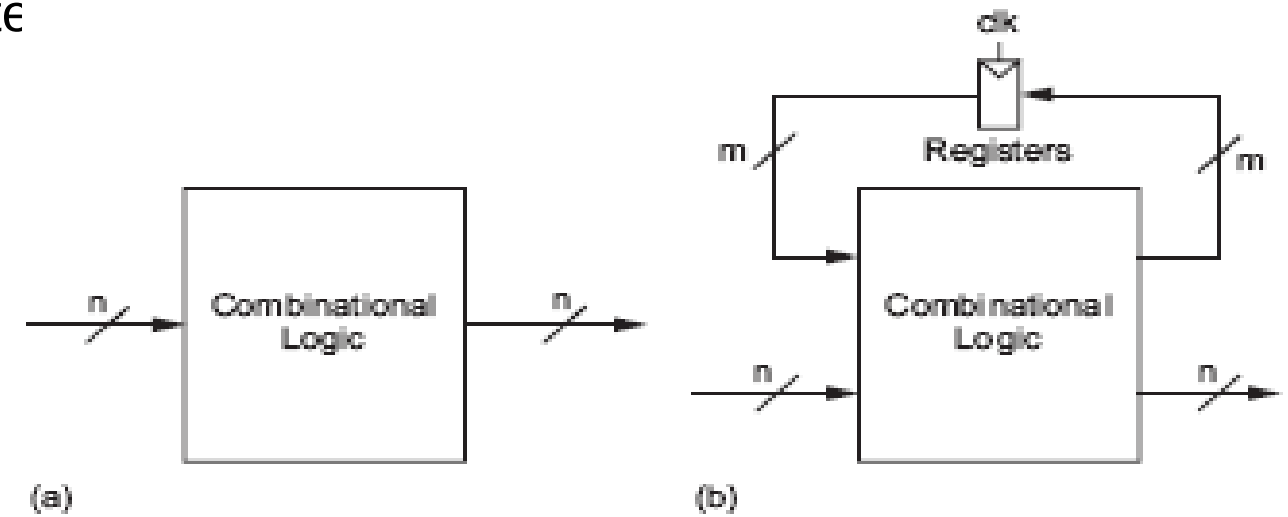
- A speck of dust on a wafer is sufficient to kill chip
- *Yield* of any chip is < 100%
  - Must test chips after manufacturing before delivery to customers to only ship good parts

Manufacturing Test

- Typical defects include the following:
  - Layer-to-layer shorts (e.g., metal-to-metal)
  - Discontinuous wires (e.g., metal thins when crossing vertical topology jumps)
  - Missing or damaged vias
  - Shorts through the thin gate oxide to the substrate or well
- Manufacturing testers are very expensive
  - Minimize **time** on tester
  - Careful selection of **test vectors**

15.3 Logic Verification Principles

- *With LSI, N = 25 and M = 50, or 2^{75} patterns, which is 3.8×10^{22}. With an application rate of 1 miros per pattern, the test time would be over a billion years (10^9).*
- **Exhaustive testing is infeasible** for most systems. The verification engineer must cleverly devise test vectors that detect any (or nearly any) defective node without requiring so many patte



15.3 Logic Verification Principles

- **Test Vectors:** are a set of patterns applied to inputs and a set of expected outputs. The set should be large enough to catch all the logic errors and manufacturing defects, yet small enough to keep test time (and cost) reasonable.
  - *Directed and random vectors are the most common types. Directed vectors are selected* by an engineer who is knowledgeable about the system.
- **Testbenches and Harnesses:** is a piece of HDL code that is placed as a wrapper around a core piece of HDL to apply and check test vectors.
- **Regression Testing:** High-level language scripts are frequently used when running large testbenches which involves performing a suite of simulations automatically every night.
- **Bug Tracking:** allow the management of a wide variety of bugs.

15.4 Silicon Debug

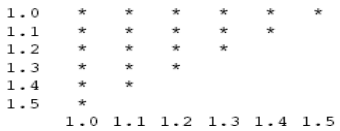
- Logic bugs vs. electrical failures
  - Most chip failures are logic bugs from inadequate simulation
  - Some are electrical failures
    - Crosstalk
    - Dynamic nodes: leakage, charge sharing
    - Ratio failures
  - A few are tool or methodology failures (e.g. DRC)
- Fix the bugs and fabricate a corrected chip

Shmoo Plots

- How to diagnose failures?
  - Hard to access chips
    - Picoprobes
    - Electron beam
    - Laser voltage probing
    - Built-in self-test
- Shmoo plots
  - Vary voltage, frequency
  - Look for cause of electrical failures

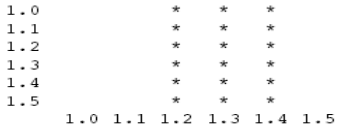
Clock period in ns on the left, frequency increases going up
Voltage on the bottom, increase left to right

** indicates a failure*

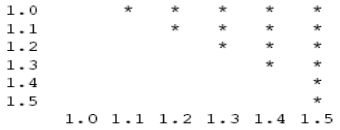


Normal
Well-behaved shmoo
Typical Speedpath

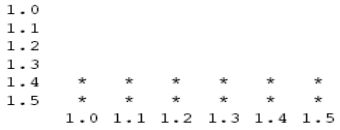
"Brick Wall"
Bistable
Initialization



"Wall"
Fails at a certain voltage
Coupling, charge share, races



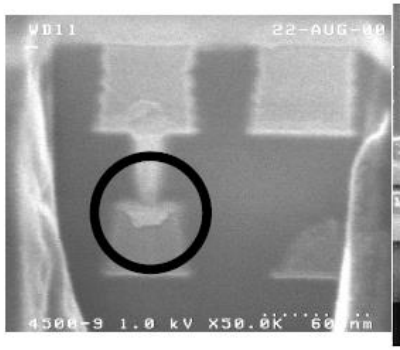
"Reverse speedpath"
Increase in voltage reduces frequency
Speedpath, leakage



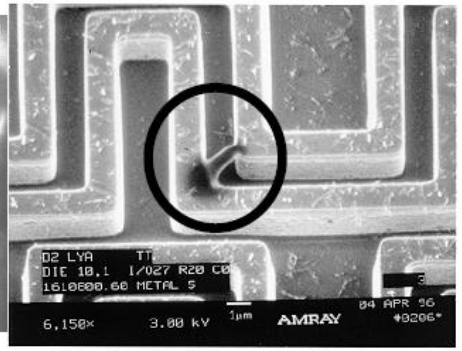
"Floor"
Works at high but not low frequency
Leakage

"Finger"
Fails at a specific point in the shmoo
Coupling

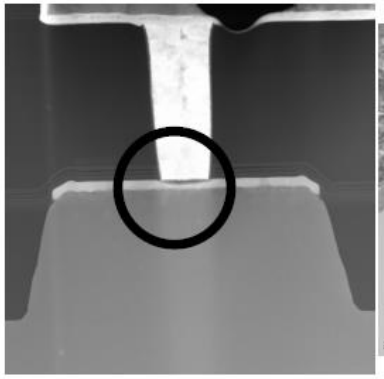
15.5 Manufacturing Test



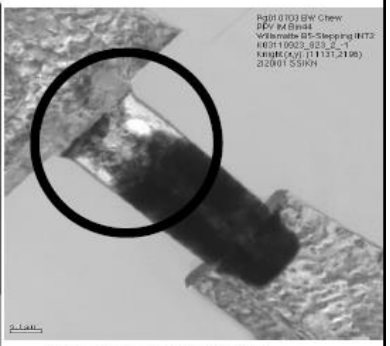
Metal 1 Shelving



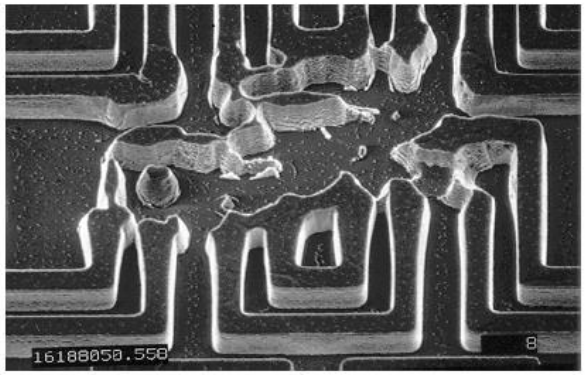
Metal 5 film particle
(bridging defect)



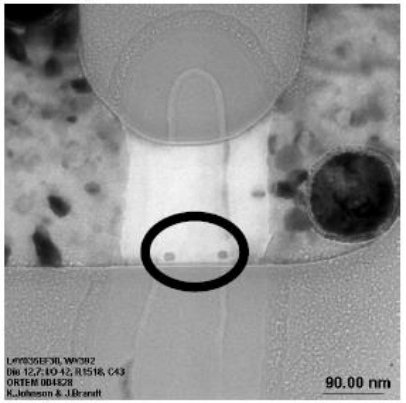
Open defect



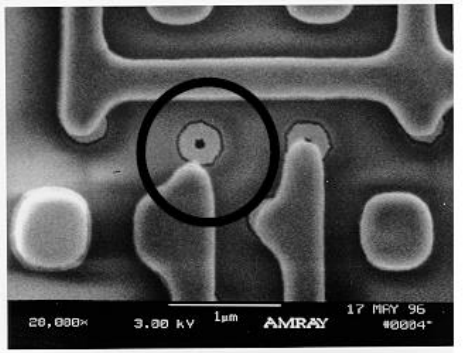
Spongy Via2
(Infant mortality)



Metal 5 blocked etch
(patterning defect)



Spot defects
"Co" Defect under Gate



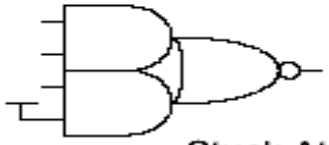
Metal 1 missing pattern
(open at contact)

SEM images courtesy Intel Corporation

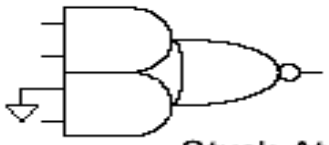
Fault Models

- A *fault model is a model for how faults* occur and their impact on circuits
- How does a chip fail?
  - Usually failures are shorts between two conductors or opens in a conductor
  - This can cause very complicated behavior
- A simpler model: ***Stuck-At***
  - Assume all failures cause nodes to be “stuck-at” 0 or 1, i.e. shorted to GND or V_{DD}
  - Not quite true, but works well in practice

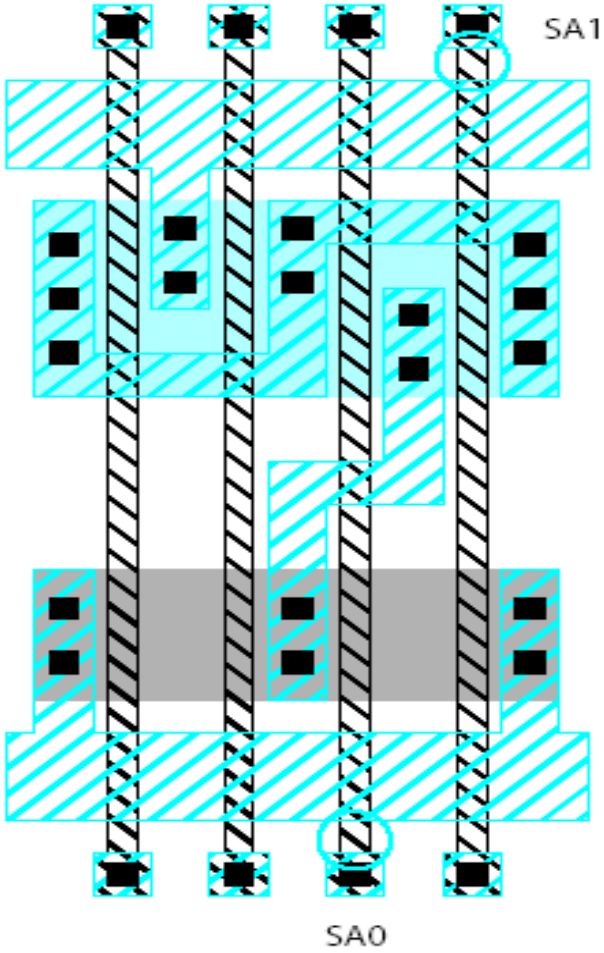
Examples



Stuck-At-1
SA1 Fault



Stuck-At-0
SA0 Fault



Observability & Controllability

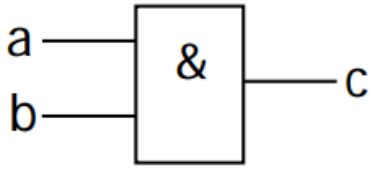
- *Observability*: ease of observing a node by watching external output pins of the chip
- *Controllability*: ease of forcing a node to 0 or 1 by driving input pins of the chip
- Combinational logic is usually easy to observe and control
- Finite state machines can be very difficult, requiring many cycles to enter desired state
  - Especially if state transition diagram is not known to the test engineer

Test Pattern Generation

- Manufacturing test ideally would check every node in the circuit to prove it is not stuck.
- Apply the smallest sequence of test vectors necessary to prove each node is not stuck.

- **Good observability and controllability reduces number of test vectors** required for manufacturing test.
  - Reduces the cost of testing
  - Motivates design-for-test

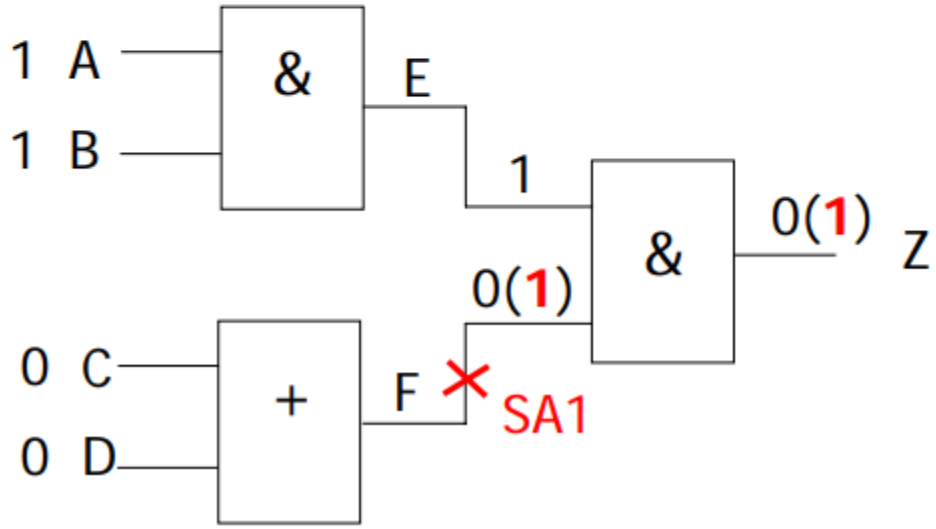
Stuck-at fault Table for AND gate



Input	Fault-free	Output Value with Stuck-at Fault					
a b	Output	a/0	a/1	b/0	b/1	c/0	c/1
0 0	0	0	0	0	0	0	1
0 1	0	0	1	0	0	0	1
1 1	1	0	1	0	1	0	1
1 0	0	0	0	0	1	0	1

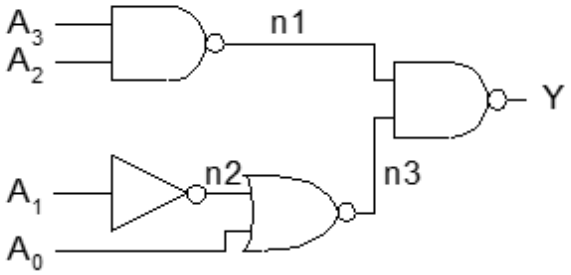
Single Stuck-At

- 14 faults
  - 2 faults (SA0, SA1) per each line
- ABCD = 1100 detects F/1
  - Faulty and fault-free outputs different
- ABCD = 1101 does NOT detect F/1
  - Faulty and fault-free outputs are the same



Test Example (read)

	SA1	SA0
A_3	{0110}	{1110}
A_2	{1010}	{1110}
A_1	{0100}	{0110}
A_0	{0110}	{0111}
n1	{1110}	{0110}
n2	{0110}	{0100}
n3	{0101}	{0110}
Y	{0110}	{1110}



- Minimum set: {0100, 0101, 0110, 0111, 1010, 1110}

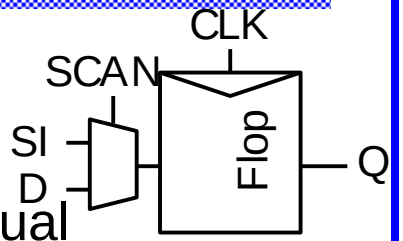
15.6 Design for Testability

- Design the chip to increase **observability** and **controllability**.
- If each register could be observed and controlled, test problem reduces to testing combinational logic between registers.
- Better yet, logic blocks could enter test mode where they generate test patterns and report the results automatically.
- Approaches of *Design for Testability* (DFT):
  - *Ad hoc testing*
  - Scan-based approaches
  - Built-in self-test (BIST)

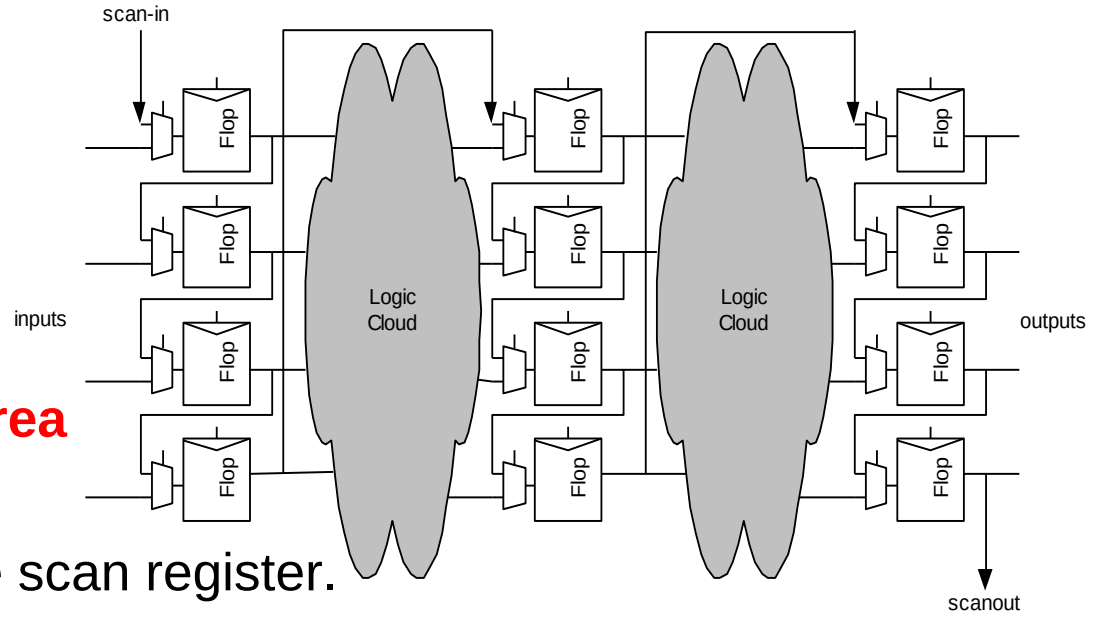
15.6.1 Ad Hoc Testing

- Are collections of ideas aimed at reducing the combinational explosion of testing.
- They are only useful for small designs where scan, ATPG, and BIST are not available.
- Common techniques for ad hoc testing:
  - Partitioning large sequential circuits
  - Adding test points
  - Adding multiplexers
  - Providing for easy state reset
- Multiplexers are used to provide alternative signal paths during testing.

15.6.2 Scan Design



- Convert each flip-flop to a scan register
  - Only costs one extra multiplexer
- Two modes: Normal mode: flip-flops behave as usual
- Scan mode: flip-flops behave as shift register (scan chain)
- Contents of flops can be scanned out and new values scanned in.
- Disadvantage is **area** and **delay** impact of extra multiplexer in the scan register.



ATPG

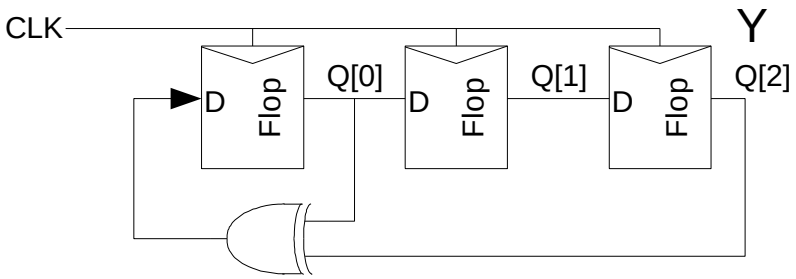
- Test pattern generation is tedious
- Automatic Test Pattern Generation (ATPG) tools produce a good set of vectors for each block of combinational logic
- Scan chains are used to control and observe the blocks
- Complete coverage requires a large number of vectors, raising the cost of test
- Most products settle for covering 90+% of potential stuck-at faults

15.6.3 Built-in Self-test

- Built-in self-test lets blocks test themselves. It adds area to the chip for the test logic, but reduces the test time required and thus can lower the overall system cost.
  - Signature Analysis: Generate pseudo-random inputs to comb. logic. And *analyzer to observe the output signals.*
  - Combine outputs into a *syndrome*
  - With high probability, block is fault-free if it produces the expected syndrome

PRSG

- *Linear Feedback Shift Register*
  - Shift register with input taken from XOR of state
  - *Pseudo-Random Sequence Generator*

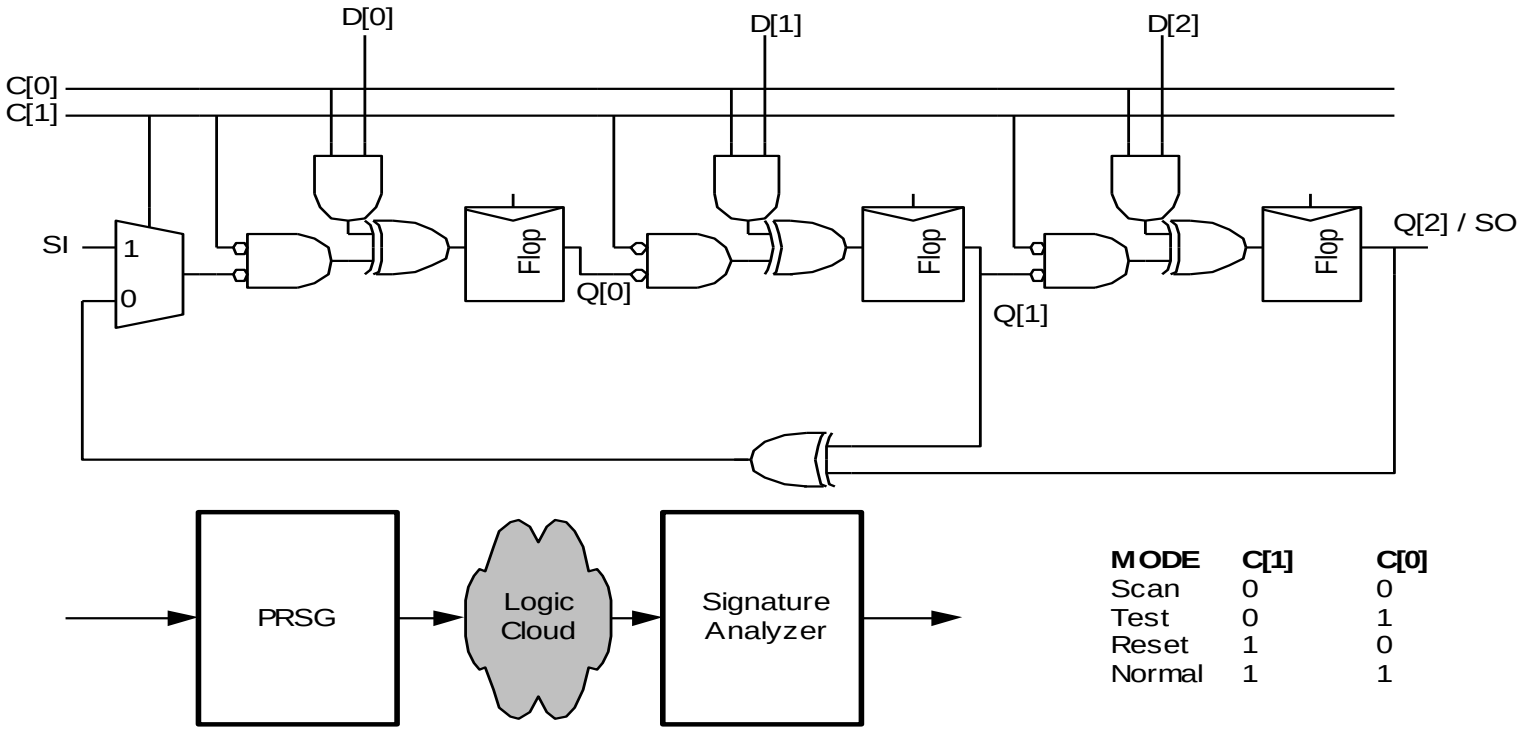


Flops reset to 111

Step	Y
0	
1	
2	
3	
4	
5	
6	
7	

BILBO

- Built-in Logic Block Observer
  - Combine scan with PRSG & signature analysis

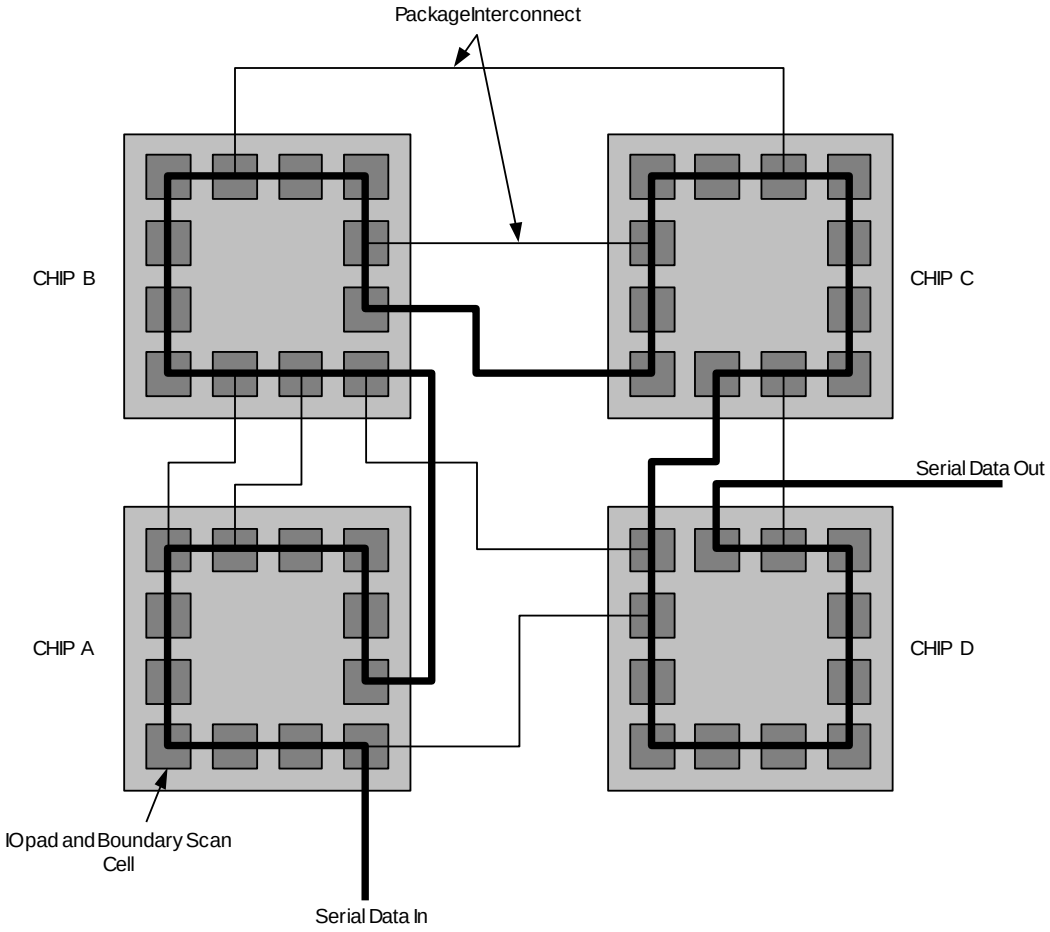


MODE	C[1]	C[0]
Scan	0	0
Test	0	1
Reset	1	0
Normal	1	1

15.7 Boundary Scan

- Testing boards is also difficult
  - Need to verify solder joints are good
    - Drive a pin to 0, then to 1
    - Check that all connected pins get the values
- Through-hold boards used “bed of nails”
- SMT and BGA boards cannot easily contact pins
- Build capability of observing and controlling pins into each chip to make board test easier

Boundary Scan Example



Boundary Scan Interface

- Boundary scan is accessed through five pins
  - TCK: test clock
  - TMS: test mode select
  - TDI: test data in
  - TDO: test data out
  - TRST*: test reset (optional)

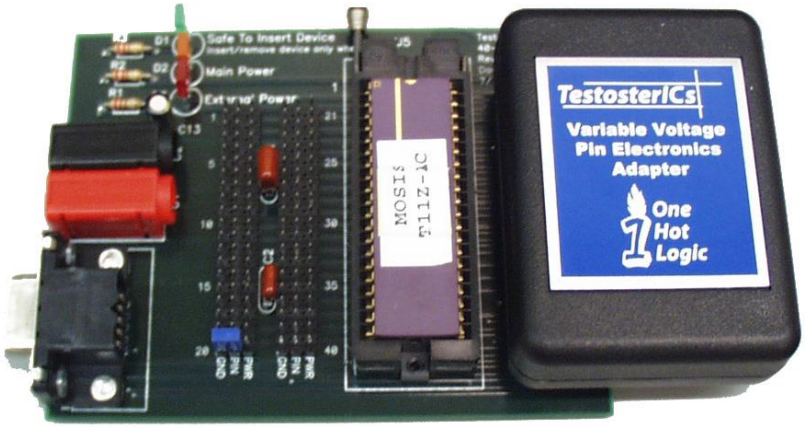
- Chips with internal scan chains can access the chains through boundary scan for unified test strategy.

Testing Your Class Project

- Presilicon Verification
  - Test vectors: corner cases and random vectors
  - HDL simulation of schematics for functionality
  - Use 2-phase clocking to avoid races
  - Use static CMOS gates to avoid electrical failures
  - Use LVS to ensure layout matches schematic
  - Don’t worry about timing
- Postsilicon Verification
  - Run your test vectors on the fabricated chip
  - Use a functional chip tester
  - Potentially use breadboard or PCB for full system

TestosterICs

- TestosterICs functional chip tester
  - Designed by clinic teams and David Diaz at HMC
  - Reads your test vectors, applies them to your chip, and reports assertion failures



Summary

- Think about testing from the beginning
  - Simulate as you go
  - Plan for test after fabrication

- "If you don't test it, it won't work! (Guaranteed)"