EECS151/251A Spring 2021 Digital Design and Integrated Circuits

Instructor:
John Wawrzynek

Lecture 27:
Wrap-up

Outline

- *Important takeaways*
- *Exam Topics*

Why Study and Learn Digital Design?

- We expect that many of you will eventually be employed as designers.
  - ***Digital design is not a spectator sport.*** The only way to learn it, and to appreciate the issues, is to do it.
  - To a large extent, it comes with practice/experience (this course is just the beginning).
  - Another way to get better is to study other designs. Not time to do much of this during the semester, but a good practice for later.
- However, a significant percentage of our graduates will not be digital designers. What's in it for them?
  - Better manager of designers, marketers, field engineers, etc.
  - Better researcher/scientist/designer in related areas
    - – Software engineers, fabrication process development, etc.
  - Even to become a better user of electronic systems.

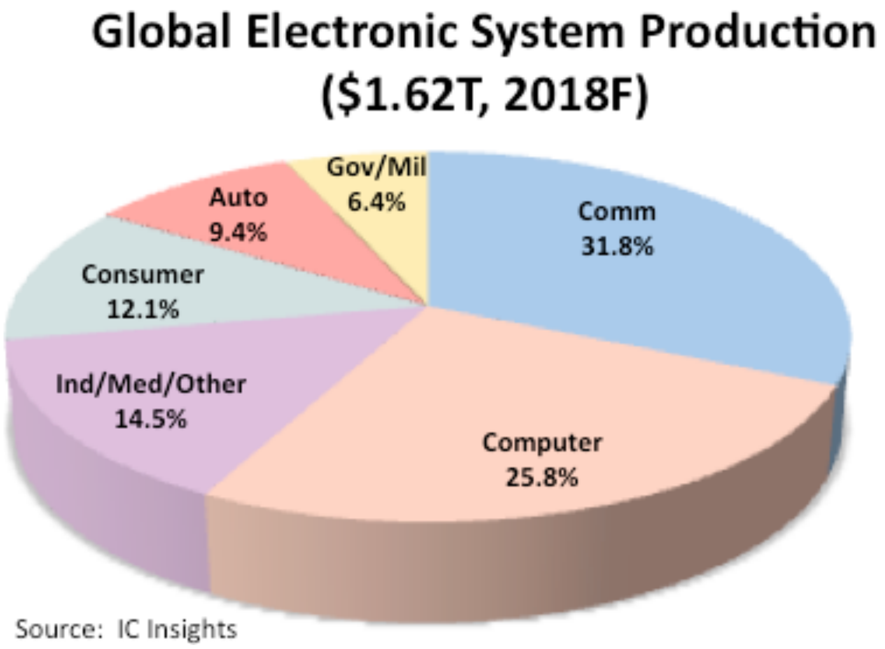
In What Context Will You be Designing?

Engineers learn so that they can build.
Scientists build so that they can learn.

- Electronic design is a critical tool for most areas of pure science:
  - Astrophysics – special electronics used for processing radio antenna signals.
  - Genomics – special processing architectures for DNA string matching.
  - In general - sensor processing, control, and number crunching.
  - Machine Learning now relies heavily on special hardware.
  - In some fields, computation has replaced experimentation – particle physics, world weather prediction (fluid dynamics).
- In computer engineering, prototypes often designed, implemented, and studied to "prove out" an idea. Common within universities and industrial research labs. Lessons learned and proven ideas often transferred to industry through licensing, technical communications, or startup companies.
  - RISC processors were first proved out at Berkeley and IBM Research

Designs in Industry

- Of course, companies are the primary employer of designers. Provide some useful products to society or government and make a profit for the shareholders.
- Interesting recent shift
  - All software giants now have hardware design teams (embedded and chips)
  - Google, Amazon, Facebook, Microsoft, …



Ten Big Ideas from EECS151

1. **Modularity and Hierarchy** is an important way to describe and think about digital systems.
2. **Parallelism** is a key property of hardware systems and distinguishes them from serial software execution.
3. **Clocking** and the use of state elements (latches, flip-flops, and memories) control the flow of data.
4. **Cost/Performance/Power tradeoffs** are possible at all levels of the system design.
5. **Boolean Algebra** and other logic representations.
6. Hardware Description Languages **(HDLs) and Logic Synthesis** are a central tool for digital design.
7. **Datapath + Controller** is a effective design pattern.
8. **Finite State Machines** abstraction gives us a way to model any digital system – used for designing controllers.
9. **Arithmetic circuits** are often based on "long-hand" arithmetic techniques.
10. **FPGAs + ASICs** give us a convenient and flexible implementation technology.

What We Didn't Cover

- Design Verification and Testing
  - Industrial designers spend more than half their time testing and verifying correctness of their designs.
    - Some of this covered in the lab and a bit in lecture. Didn't cover rigorous testing procedures.
  - Most industrial products are designed from the start for testability. Important for design verification and later for manufacturing test.
  - Related: Fault modeling and fault tolerant design.

- Other High-level Optimization Techniques
  - High-level Synthesis - now starting to catch on

- Other High-level Architectures: GPUs, video processing, network routers, …

- Asynchronous Design

Most Closely Related Courses

- CS152 Computer Architecture and Engineering
  - Design and Analysis of Microprocessors
  - Applies basic design concepts from EECS151
- EE241B Digital Integrated Circuits
  - Transistor-level design of ICs
  - More on Advanced ASIC Tool use
- CS250 VLSI Systems Design
  - Advanced-undergrad/grad course
  - Design tradeoffs at the chip design level

Future Design Issues

- Automatic High-level synthesis (HLS) and optimization (with micro-architecture synthesis) and hardware/software co-design.
- Current practice is "system on a chip" (SOC) design methodology:
  - Pre-designed subsystems (processor cores, bus controllers, memory systems, network interfaces, etc.) connected with standard on-chip interconnect or bus.
  - Strong emphasis on "accelerators" for energy efficiency and performance.
- A number of alternatives to silicon VLSI have been proposed, including techniques based on:
  - Carbon nanotubes*, molecular electronics, quantum mechanics, and biological processes.
  - How will these change the way we design systems?

**In 2012, IBM produced a sub-10 nm carbon nanotube transistor that outperformed silicon on speed and power. "The superior low-voltage performance of the sub-10 nm CNT transistor proves the viability of nanotubes for consideration in future aggressively scaled transistor technologies", according to the abstract of the paper in Nano Letters.*

Final Exam and Project Info

- **Exam held May 14th Friday 7-10PM.**
  - "Comprehensive" Final Exam
  - Emphasis on second half, but some coverage of first half
  - Same format as Exam 1. Open-book and notes. Zoom proctoring.
- FPGA project interviews: Wednesday of RRR week, 5/5.
- FPGA project final reports: Friday 5/7.

The exam will be a “take home exam” and take place Thursday April 30, 7-10PM. The exam comprises a set of questions with 1 point per expected minute of completion with a total of approximately 90 points. 251A students will be asked to complete extra questions. All students are allowed to refer to your notes, the class lecture notes, and any other reference materials that you have available. However, the problems are challenging and if you are not suitable familiar with the course topics, you may not have much time to look at notes. You are not allowed to speak with anyone on any topic related to the course during the exam period. After completing the exam, you will be asked to sign a statement attesting that you did not discuss the exam problems with anyone else. You will turn in your answers with Gradescope as you do your homework.

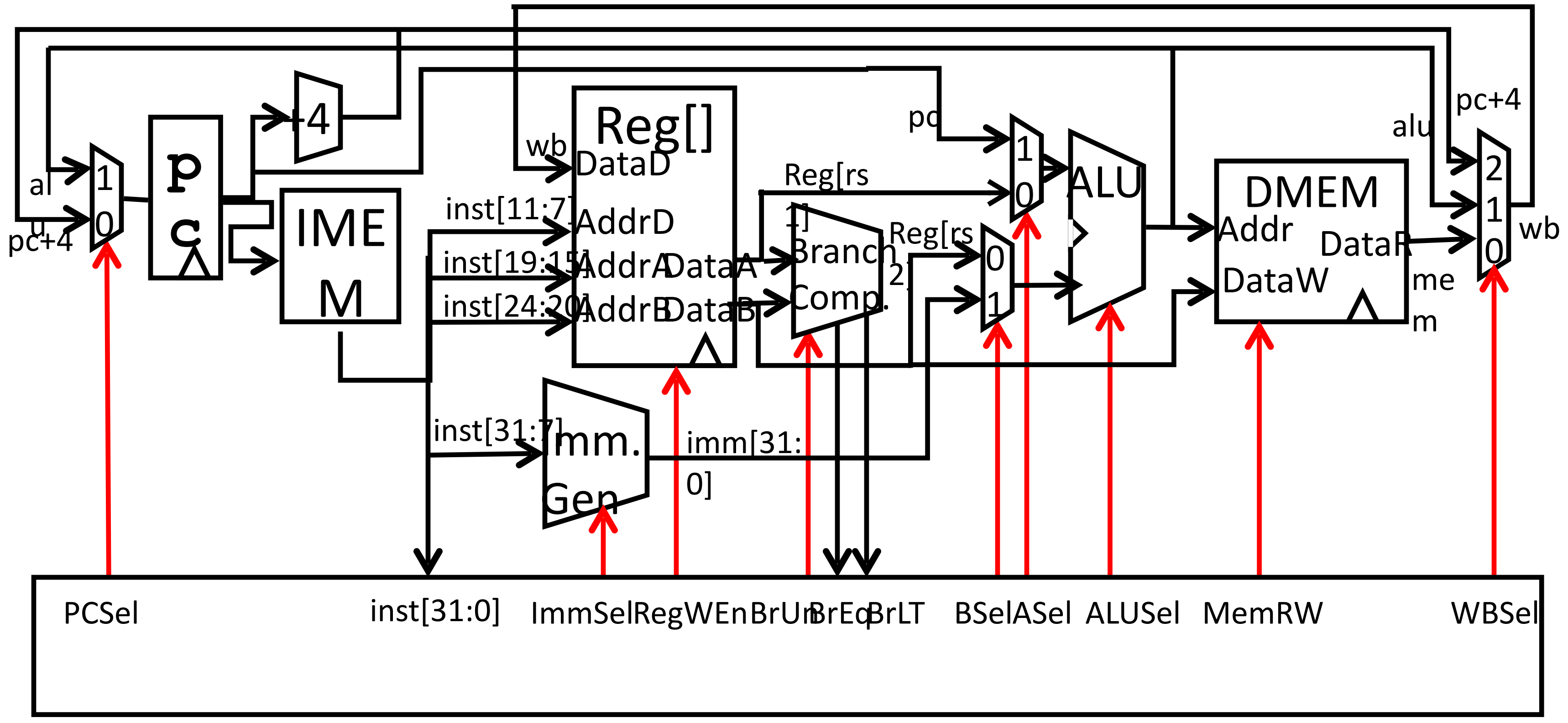
As with the first exam, clarification questions can be directed to the teaching staff via piazza.

Topics:
The final exam will be comprehensive and test all topics covered this semester. However, emphasis will be placed on topics covered after the midterm exam—those listed below.

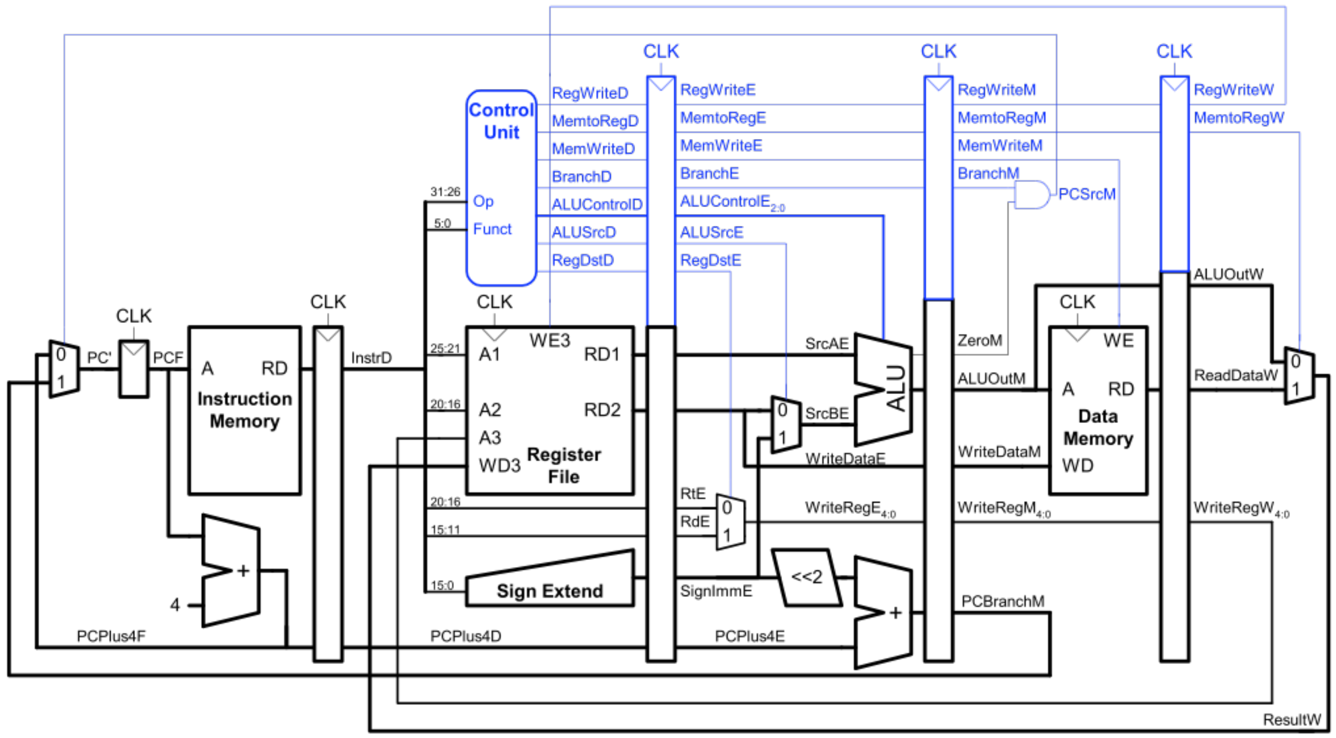
1. **How to Design a RISC-V Single-Cycle Processor from the ISA**
2. **Processor Pipelining Hazards and Mechanisms**
3. **Sources of Power and Energy consumption in Digital ICs**
4. **Principles Behind Five Low-power Design Techniques**
5. **How to Improve Energy Efficiency through Parallelism and Pipelining**
6. **Memory Block Internal Architecture**
7. **SRAM Cell and Read/Write Operation**
8. **Memory Block Periphery Circuits**
9. **Memory Decoder Design**
10. **DRAM Cell and Read/Write Operation**
11. **Dual-port Memory Architecture**
12. **Cascading Memory blocks for More Width, Depth, and Ports**
13. **FIFO Implementation**

14. **Serialization versus Parallelization in Iterative Computations**
15. **Principles of Pipelining and Restrictions of Loops**
16. **C-Slow Technique for Pipelining Loops**
17. **List Processor Design and Optimizations**
18. **Modulo Scheduling**
19. **Carry Select Adder Design**
20. **Carry Lookahead and Parallel Prefix Adders**
21. **Bit-Serial Addition**
22. **Array Multiplier Design**
23. **Carry Save Addition**
24. **Signed Multiplication**
25. **Bit-Serial Multiplication**
26. **CSD Multiplier Design**
27. **Booth Encoding Multiplication**
28. **Log and Barrel Shifters Design and Analysis**
29. **Use of Counters in Controller Design**
30. **Effect of Clock Uncertainties on Maximum Clock Frequency**
31. **Source of Clock Uncertainties**
32. **Principles of Good Clock Distribution**
33. **IR and dI/dt effects in Power distribution**
34. **Types and Sources of Faults in ICs**
35. **Hamming Codes**

Single-Cycle RISC-V RV32I Datapath

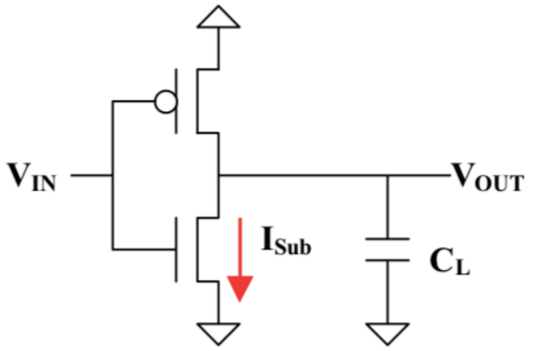


Pipelined Processor

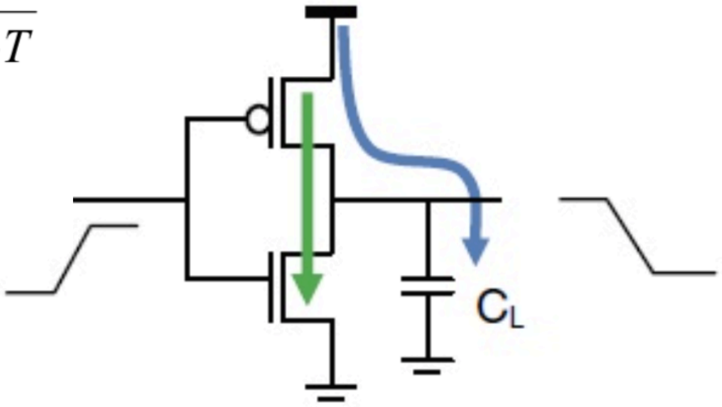


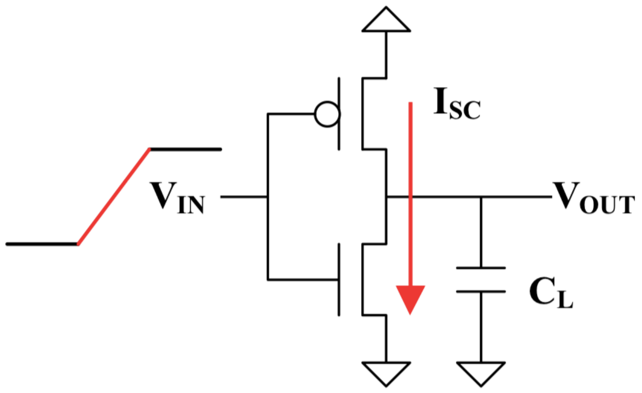
Sources of Power and Energy consumption in Digital ICs

Total Power = $P_{\text{switching}} + P_{\text{short-circuit}} + P_{\text{leakage}}$



$$I_{DSub} = k \cdot e^{\frac{-q \cdot V_T}{a \cdot k_a \cdot T}}$$

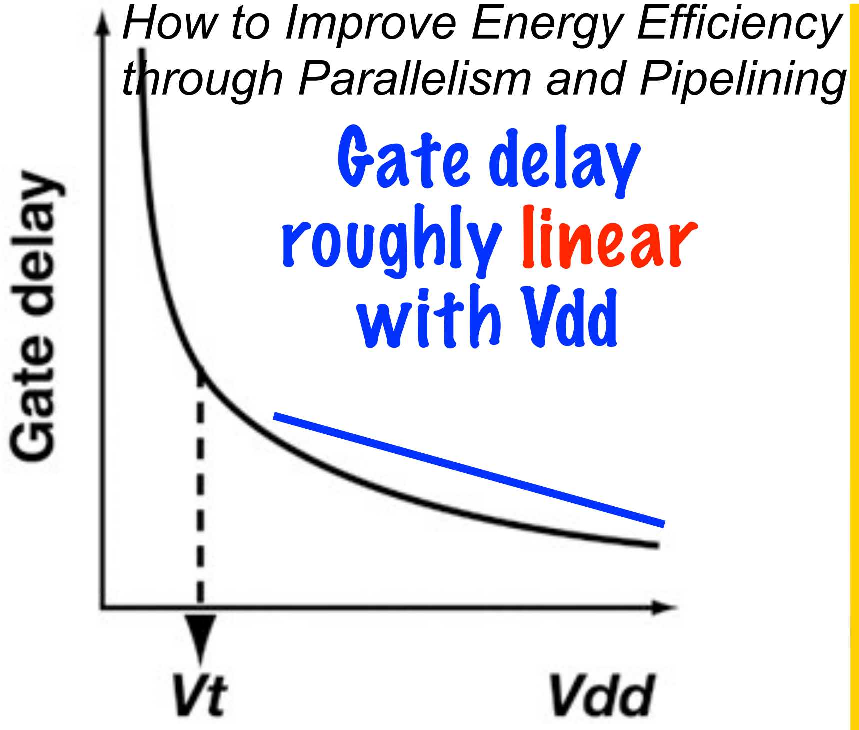




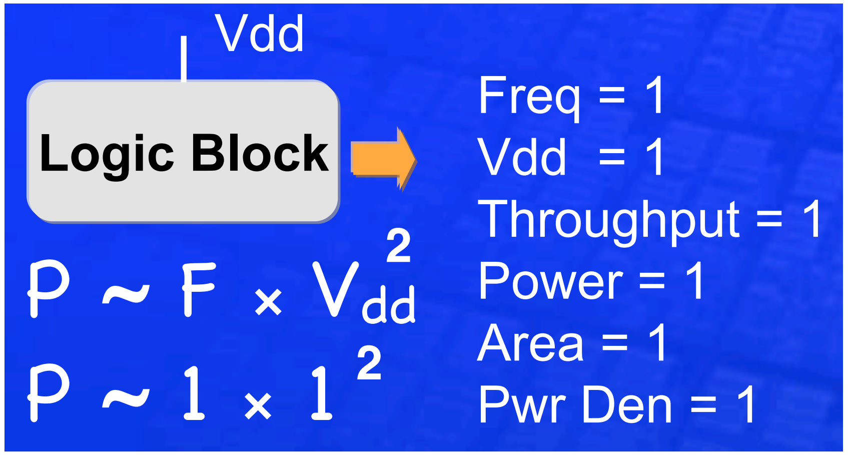
Some low-power design techniques

- Parallelism and pipelining
- Power-down idle transistors
- Slow down non-critical paths
- Clock gating
- Thermal management

Principles Behind Five Low-power Design Techniques



And so, we can transform this:



$P \sim F \times V_{dd}^2$

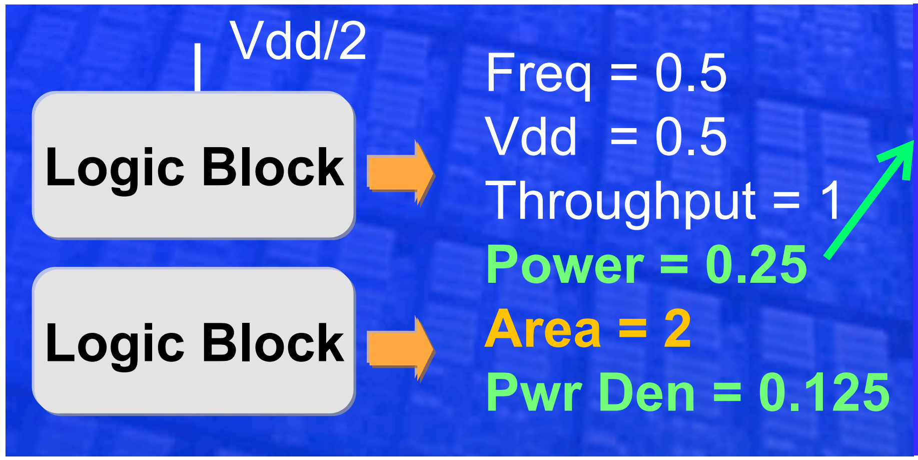
$P \sim 1 \times 1^2$

Freq = 1
Vdd = 1
Throughput = 1
Power = 1
Area = 1
Pwr Den = 1

Block processes stereo audio. 1/2 of clocks for "left", 1/2 for "right".

Into this:

Top block processes "left", bottom "right".



Freq = 0.5
Vdd = 0.5
Throughput = 1
Power = 0.25
Area = 2
Pwr Den = 0.125

$P \sim \#blks \times F \times V_{dd}^2$

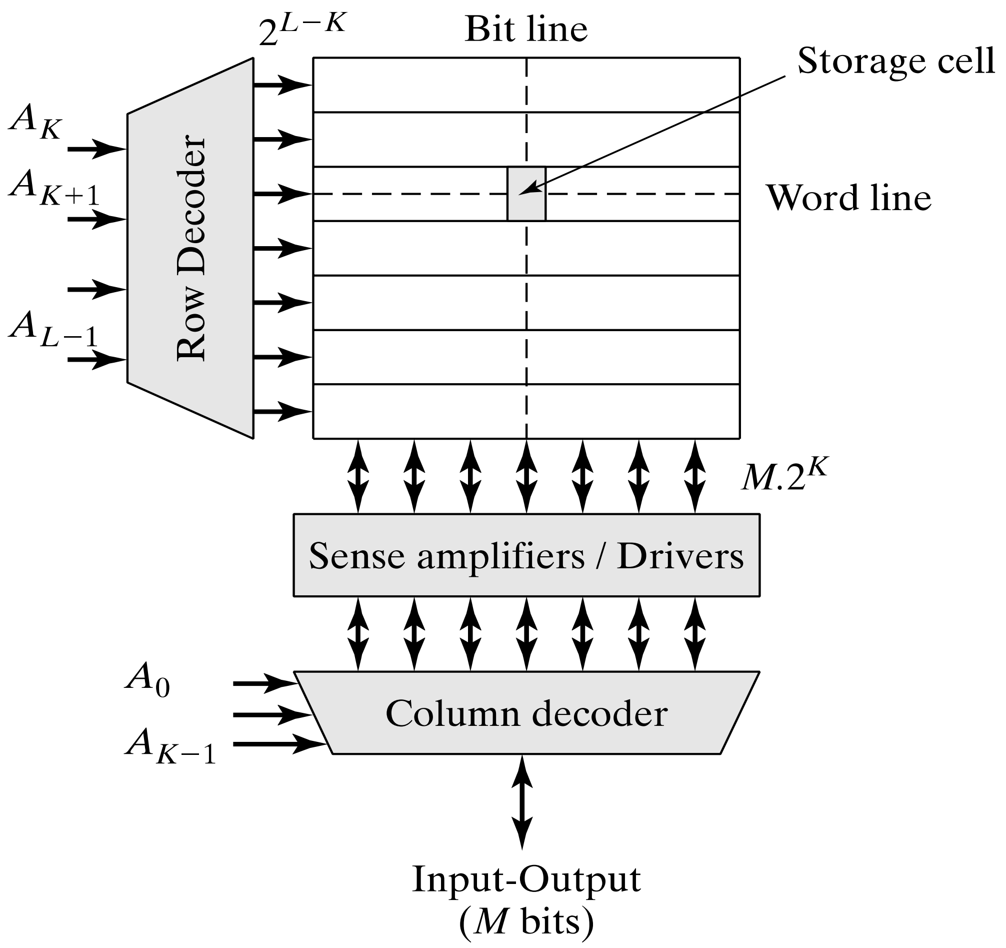
$P \sim 2 \times 1/2 \times 1/4 = 1/4$

CV^2 power only

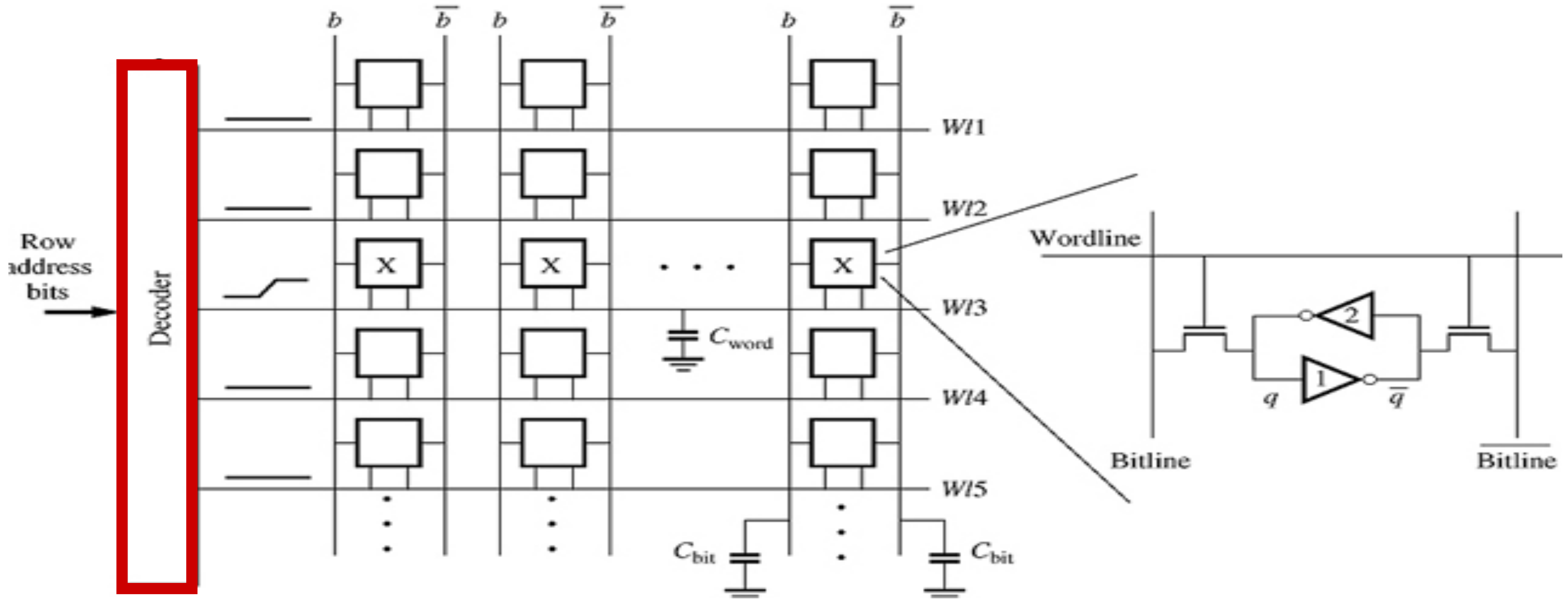
THIS MAGIC TRICK BROUGHT TO YOU BY CORY HALL ...

Memory Architecture Overview

- **Word lines** used to select a row for reading or writing
- **Bit lines** carry data to/from periphery
- **Core** *aspect ratio* keep close to 1 to help balance delay on word line versus bit line
- **Address bits** are divided between the two decoders
- **Row decoder** used to select word line
- **Column decoder** used to select one or more columns for input/output of data



SRAM read/write operations

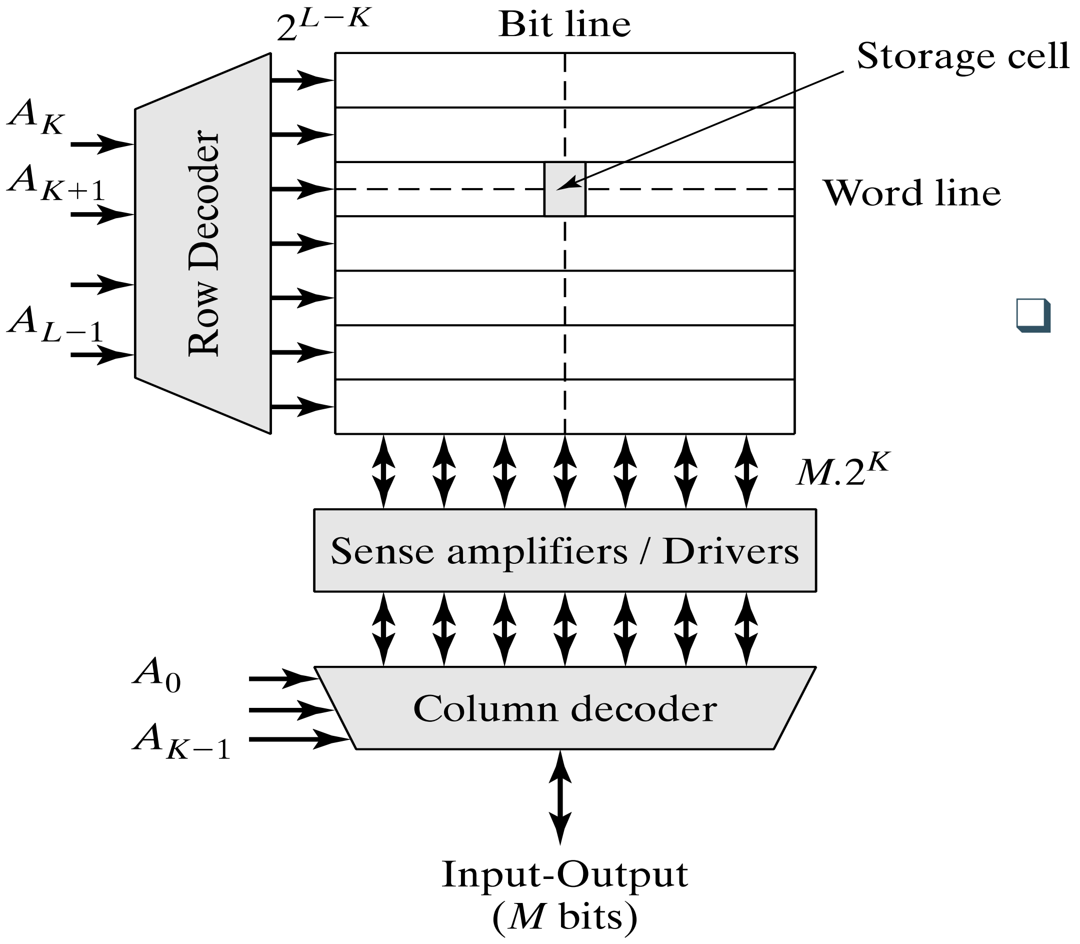


Periphery

- ❑ Decoders
- ❑ Sense Amplifiers
- ❑ Input/Output Buffers
- ❑ Control / Timing Circuitry

Row Decoder

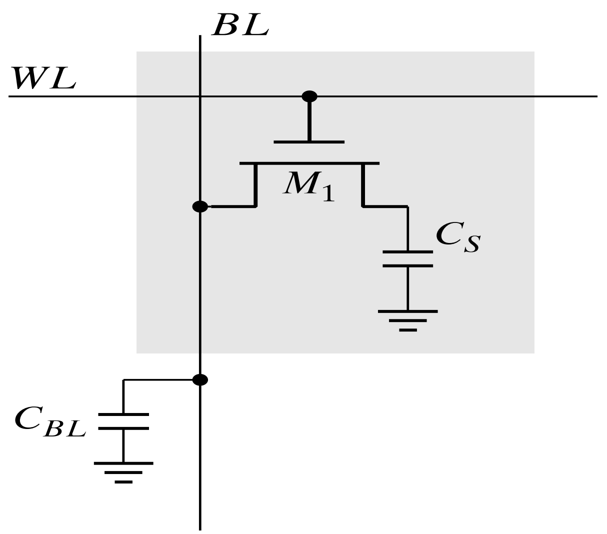
- Expands L-K address lines into 2^{L-K} word lines

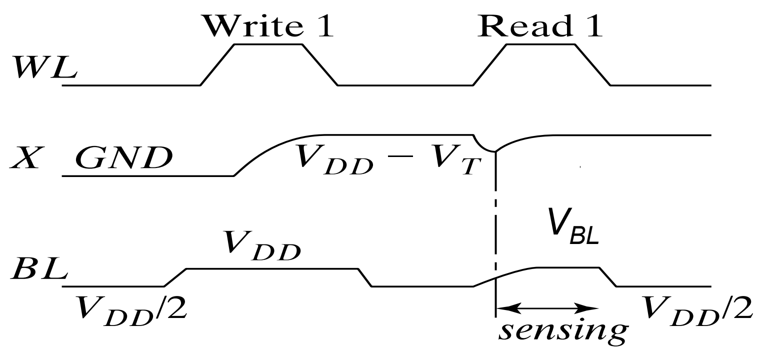


- ❑ Example: decoder for 8Kx8 memory block
  - ❑ core arranged as 256x256 cells
  - ❑ Need 256 AND gates, each driving one word line

DRAM Cell and Read/Write Operation

1-Transistor DRAM Cell





$V_{BIT} = 0$ or $(V_{DD} - V_T)$

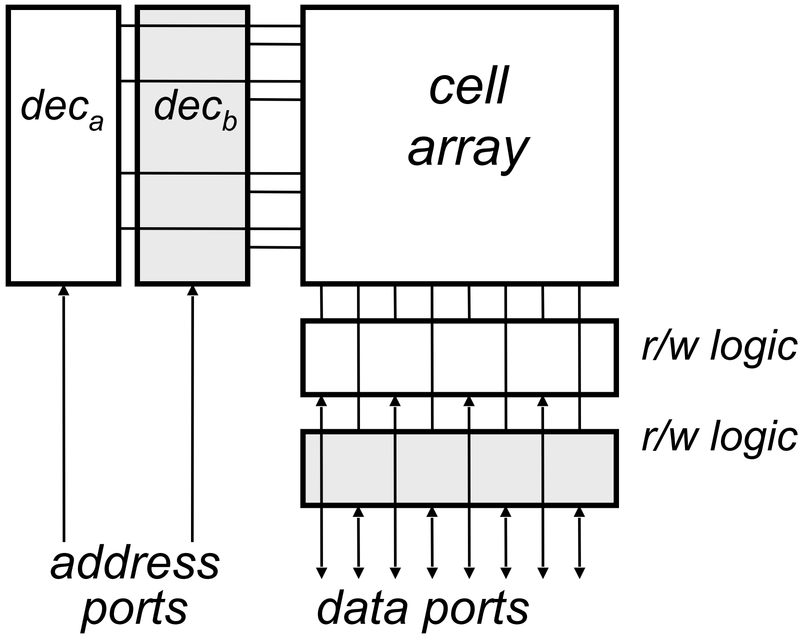
Write: C_S is charged or discharged by asserting WL and BL.
Read: Charge redistribution takes places between bit line and storage capacitance

$C_S << C_{BL}$ Voltage swing is small; typically around 250 mV.

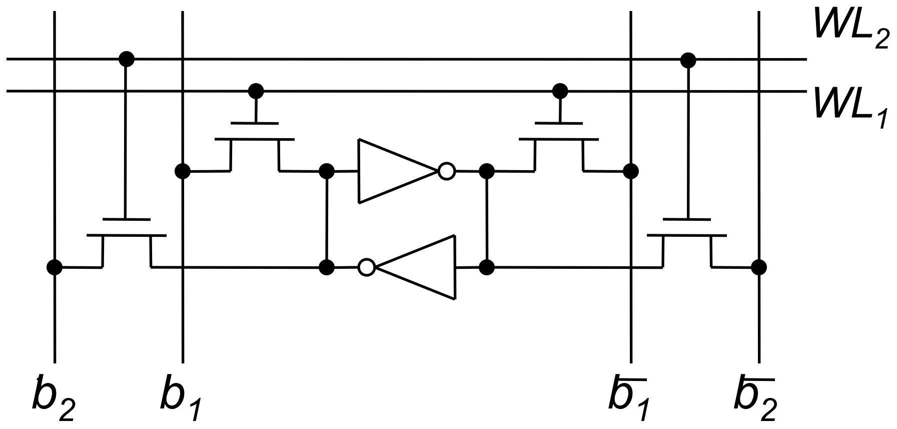
- To get sufficient C_s, special IC process is used
- Cell reading is destructive, therefore read operation always is followed by a write-back
- Cell looses charge (leaks away in ms - highly temperature dependent), therefore cells occasionally need to be "refreshed" - read/write cycle

Dual-ported Memory Internals

- Add decoder, another set of read/write logic, bits lines, word lines:



- *Example cell: SRAM*

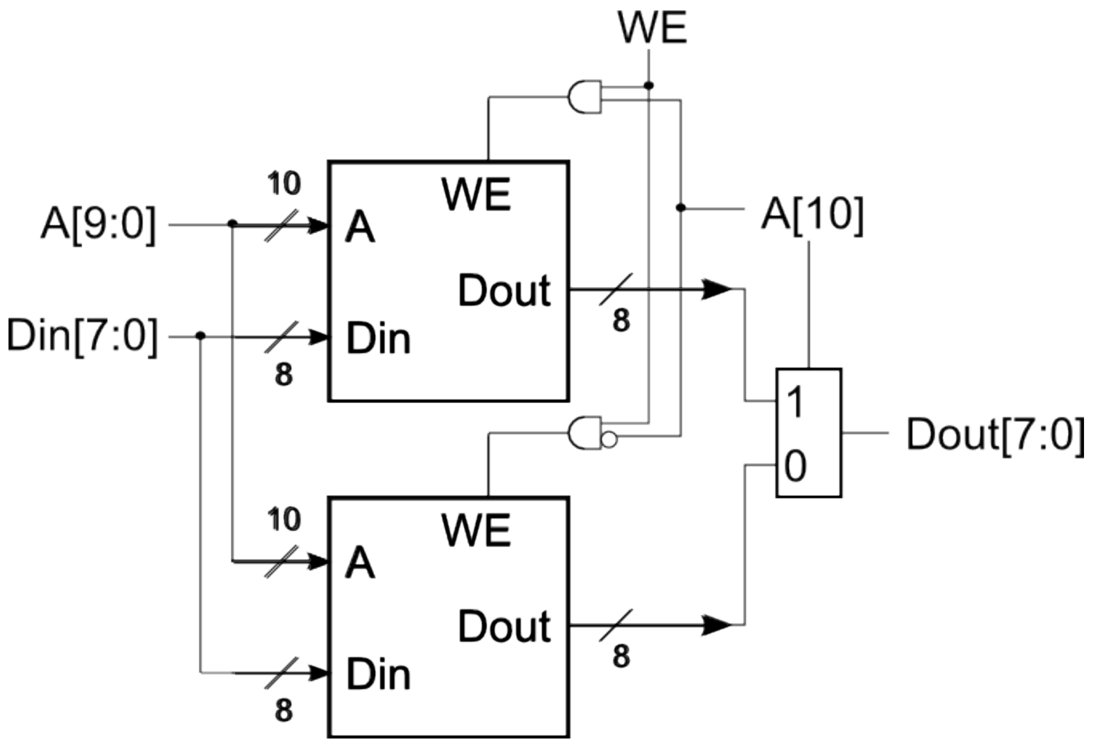


- *Repeat everything but cross-coupled inverters.*
- *This scheme extends up to a couple more ports, then need to add additional transistors.*

Cascading Memory-Blocks

How to make larger memory blocks out of smaller ones.

Increasing the depth. Example: given 1Kx8, want 2Kx8



FIFO Implementation Details

- *Assume, dual-port memory with asynchronous read, synchronous write.*
- *Binary counter for each of read and write address. CEs (count enable) controlled by WE and RE.*
- *Equal comparator to see when pointers match.*
- *State elements for FULL and EMPTY flags:*

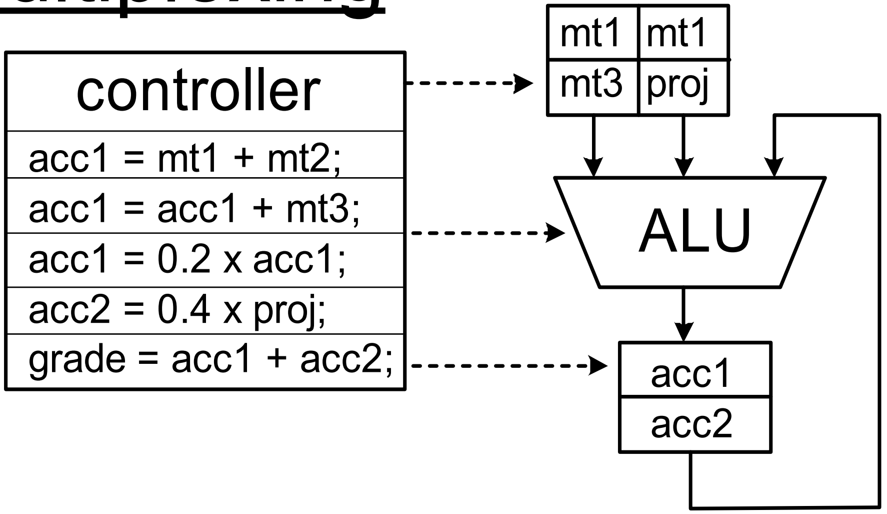
WE	RE	equal*	$EMPTY_i$	$FULL_i$
0	0	0	0	0
0	0	1	$EMPTY_{i-1}$	$FULL_{i-1}$
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	0	0
1	1	1	$EMPTY_{i-1}$	$FULL_{i-1}$

- *Control logic (FSM) with truth-table (draft) shown to left.*

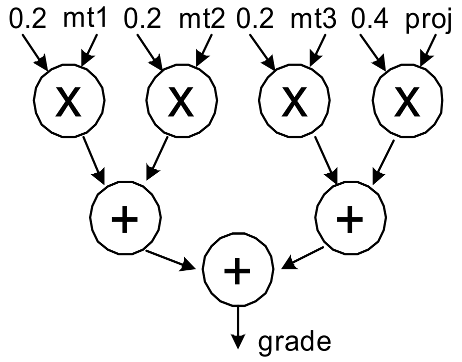
** Actually need 2 signals: "will be equal after read" and "will be equal after write"*

Time-Multiplexing

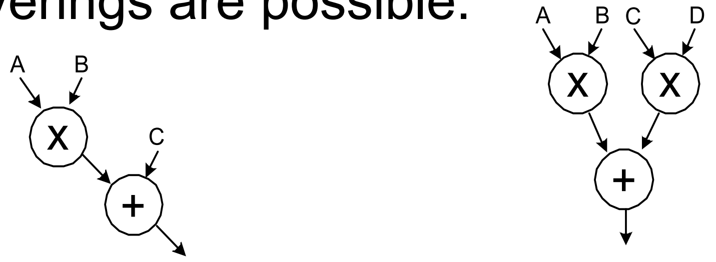
- *Time multiplex* single ALU for all adds and multiplies:
- Attempts to minimize cost at the expense of time.
  - Need to add extra register, muxes, control.



- If we adopt above approach, we can then consider the combinational hardware circuit diagram as an *abstract computation-graph*.



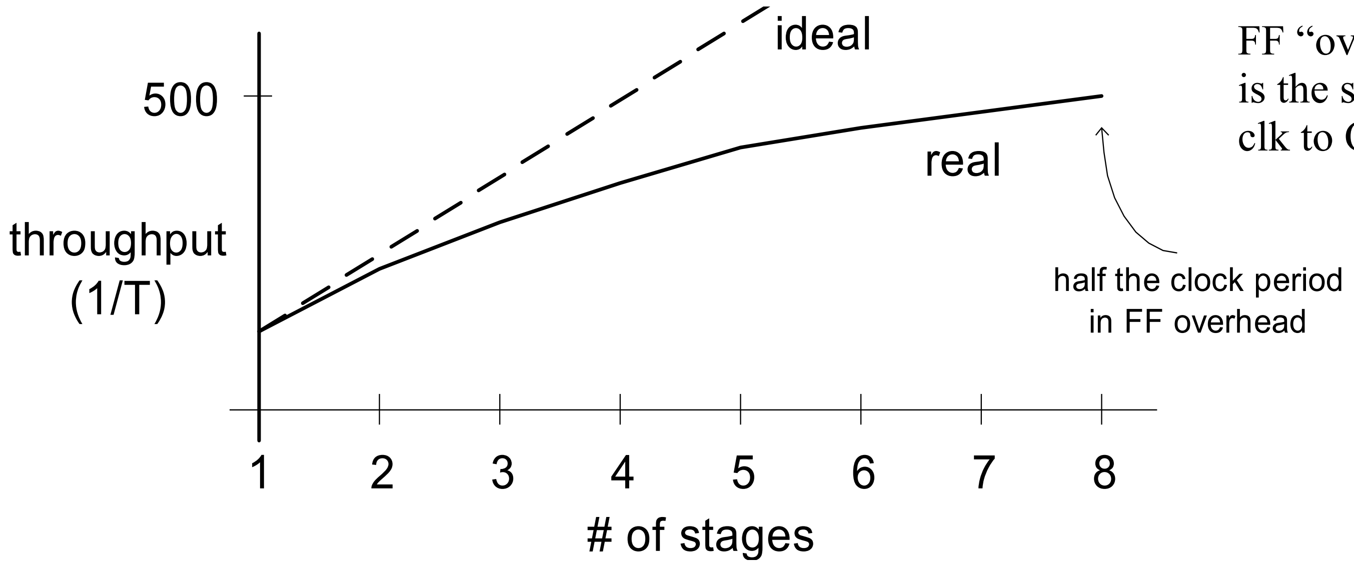
Using other primitives, other coverings are possible.



- This time-multiplexing "covers" the computation graph by performing the action of each node one at a time. (Sort of *emulates* it.)

Limits on Pipelining

- Without FF overhead, throughput improvement α # of stages.
- After many stages are added FF overhead begins to dominate:



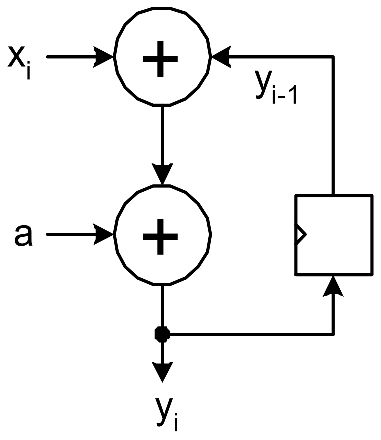
FF "overhead" is the setup and clk to Q times.

- Other limiters to effective pipelining:
  - clock skew contributes to clock overhead
  - unequal stages
  - FFs dominate *cost*
  - clock distribution power consumption
  - feedback (dependencies between loop iterations)

Pipelining Loops with Feedback

"Loop carry dependency"

However, we can overlap the "non-feedback" part of the iterations:

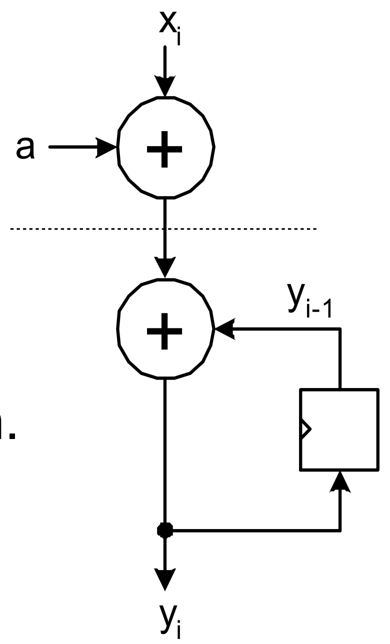


Add is associative and communitive. Therefore we can reorder the computation to shorten the delay of the feedback path:

$$\mathbf{y_i = (y_{i-1} + x_i) + a = (a + x_i) + y_{i-1}}$$

add_1	x_i+a	$x_{i+1}+a$	$x_{i+2}+a$	
add_2		y_i	y_{i+1}	y_{i+2}

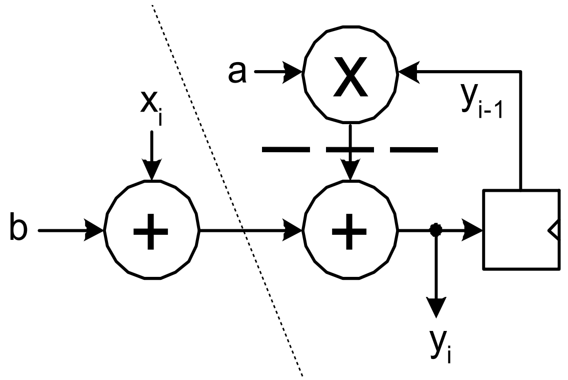
"Shorten" the feedback path.



- Pipelining is limited to 2 stages.

“C-slow” Technique

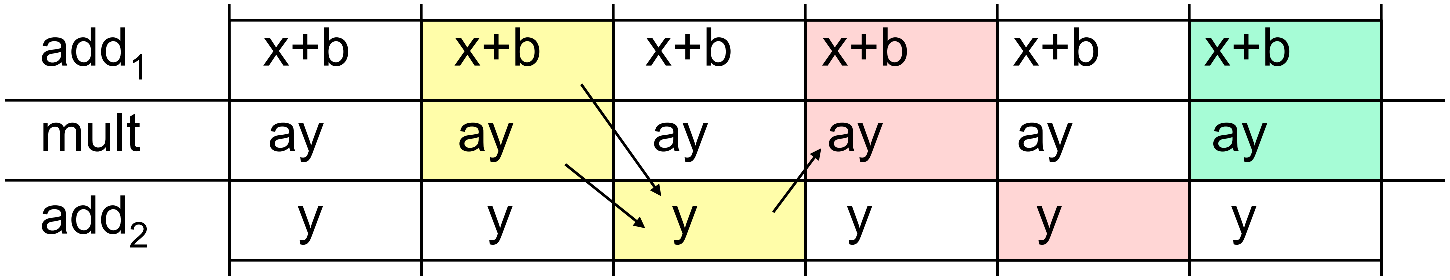
- Essentially this means we go ahead and cut feedback path:



- This makes operations in adjacent pipeline stages independent and allows full cycle for each:

- C computations (in this case C=2) can use the pipeline simultaneously.
- Must be independent.
- Input MUX interleaves input streams.
- Each stream runs at half the pipeline frequency.
- Pipeline achieves full throughput.

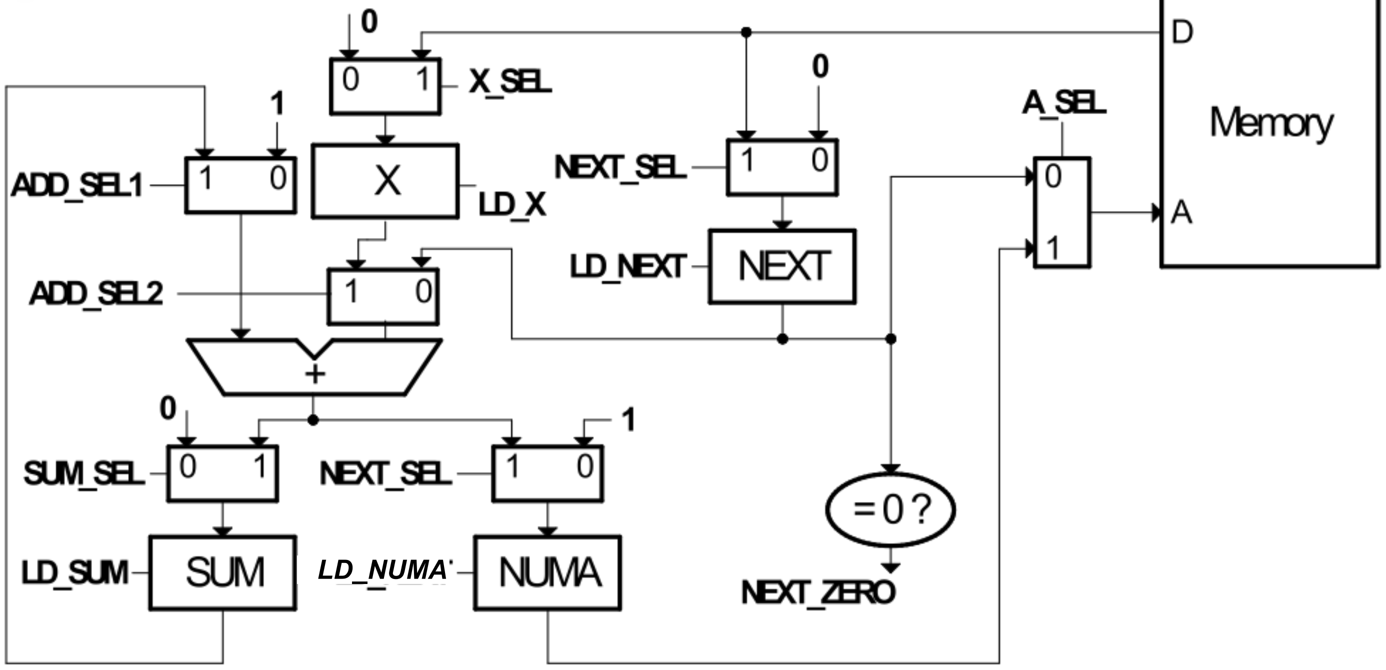
Multithreaded Processors use this.



add_1	x+b	x+b	x+b	x+b	x+b	x+b
mult	ay	ay	ay	ay	ay	ay
add_2	y	y	y	y	y	y

5. Optimization, Architecture #4

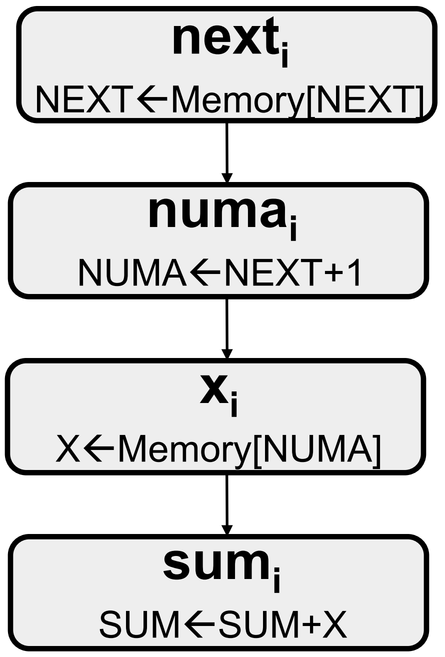
- Datapath:



- Incremental cost:
  - Addition of another register & mux, adder mux, and control.
- Performance: find max time of the four actions

1. X←Memory[NUMA], NUMA←NEXT+1; 0.5+1+10+1+0.5 = 13ns
   same for all ⇒ $\boxed{\text{T>13ns, F<77MHz}}$
2. NEXT←Memory[NEXT], SUM←SUM+X;

Modulo Scheduling List Processor



- Assuming a single adder and a single ported memory. Minimal schedule section length = 2.

  Because both memory and adder are used for 2 cycles during one iteration.

memory	$next_i$	
adder		$numa_i$

memory	$next_i$	X_{i-1}
adder		$numa_i$

wrap-around, decrease subscript

memory	$next_i$	X_{i-1}
adder	sum_{i-2}	$numa_i$

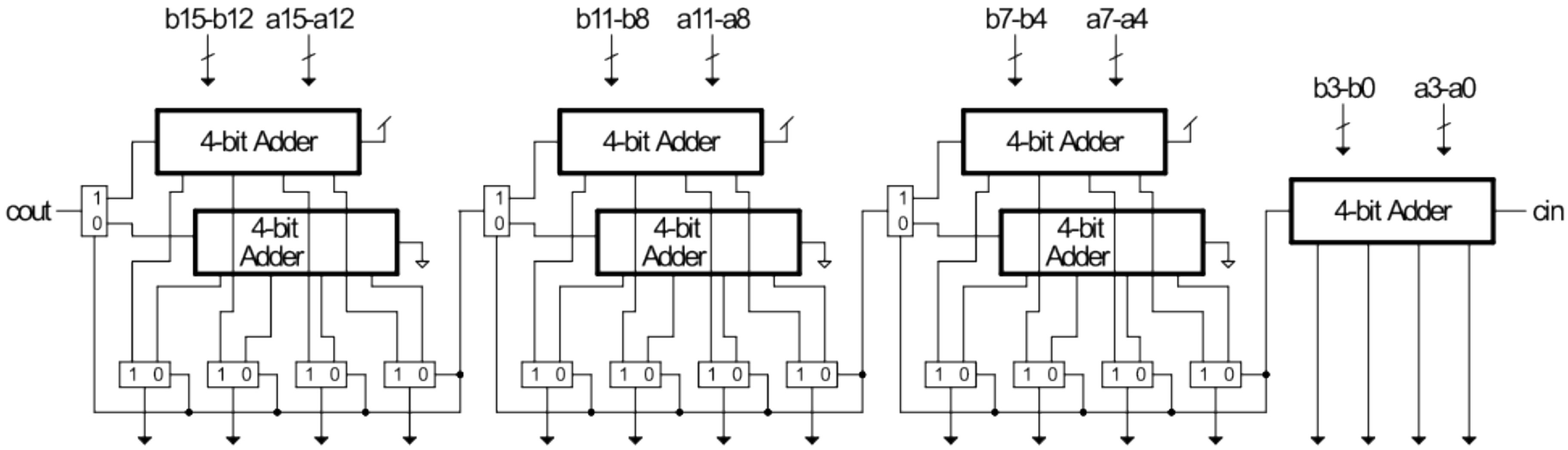
wrap-around, decrease subscript

- Finished schedule for 4 iterations:

Memory	$next_1$		$next_2$	x_1	$next_3$	x_2	$next_4$	x_3	
adder		$numa_1$		$numa_2$	sum_1	$numa_3$	sum_2	$numa_4$	sum_3

Carry Select Adder

- Extending Carry-select to multiple blocks

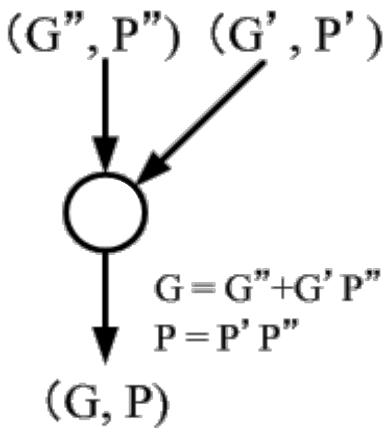


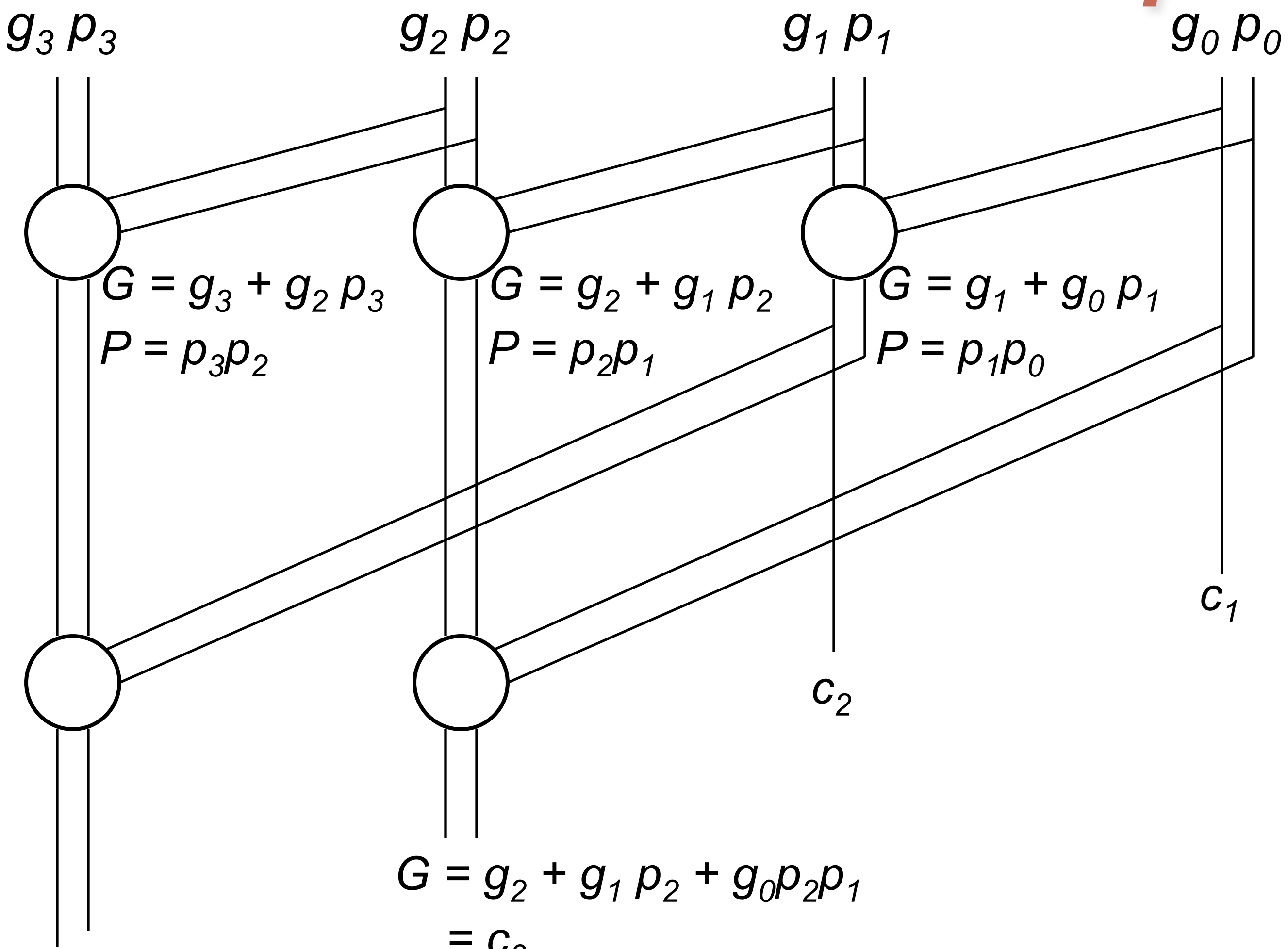
- What is the optimal # of blocks and # of bits/block?
  - If blocks too small delay dominated by total mux delay
  - If blocks too large delay dominated by adder ripple delay

$$\boxed{\sqrt{N} \text{ stages of } \sqrt{N} \text{ bits}}$$

T α sqrt(N),
*Cost ≈2*ripple + muxes*

Parallel Prefix Adder Example





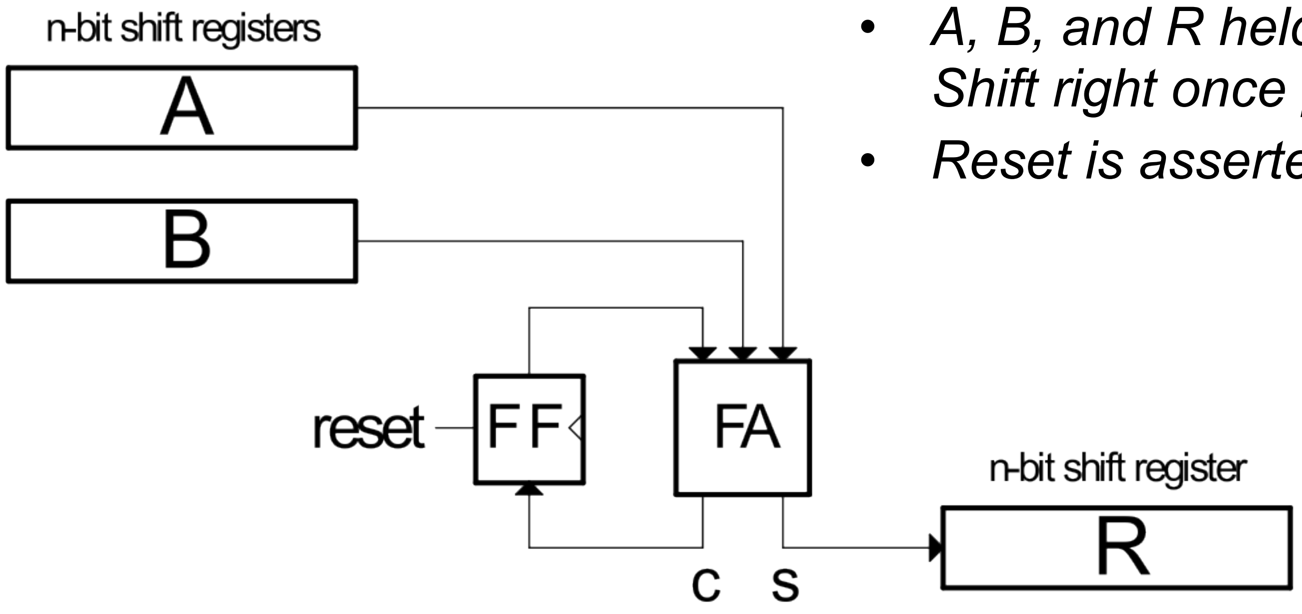
$G = g_2 + g_1\ p_2 + g_0p_2p_1$
$= c_3$

$G = g_3 + g_2\ p_3 + (g_1 + g_0p_1)p_3p_2$
$= g_3 + g_2p_3 + g_1p_3p_2 + g_0p_3p_2p_1$
$= c_4$

$\boxed{s_i = a_i \oplus b_i \oplus c_i = p_i \oplus c_i}$

Bit-serial Adder

Bit-Serial Addition



- *A, B, and R held in shift-registers. Shift right once per clock cycle.*
- *Reset is asserted by controller.*

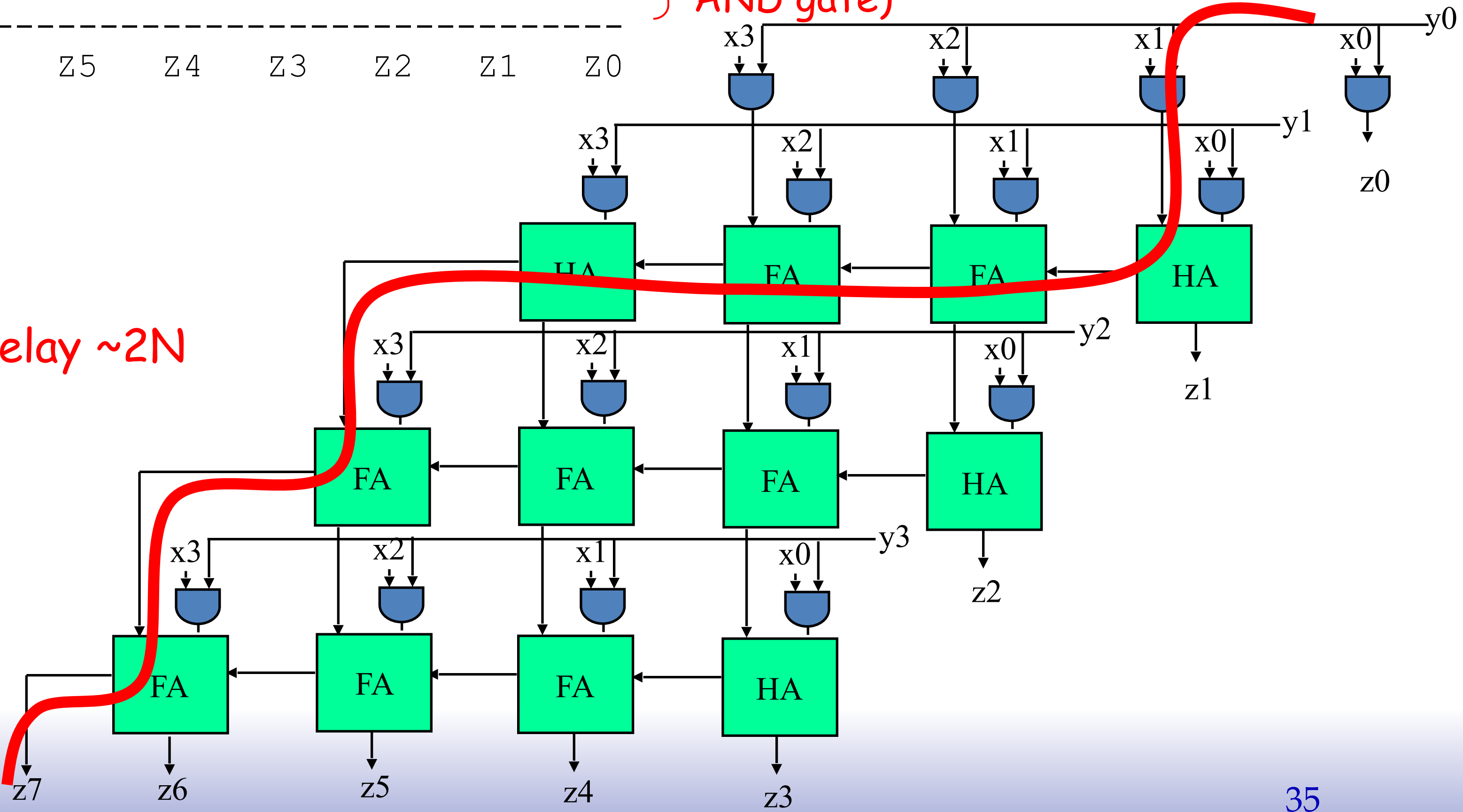
- Addition of 2 n-bit numbers:
  - takes n clock cycles,
  - uses 1 FF, 1 FA cell, plus registers
  - the bit streams may come from or go to other circuits, therefore the registers might not be needed.

Combinational Multiplier (unsigned)

```
                              X3    X2    X1    X0   ← multiplicand
                           *  Y3    Y2    Y1    Y0   ← multiplier
                           --------------------------
                             X3Y0  X2Y0  X1Y0  X0Y0
+                      X3Y1  X2Y1  X1Y1  X0Y1
+                X3Y2  X2Y2  X1Y2  X0Y2
+          X3Y3  X2Y3  X1Y3  X0Y3
---------------------------------------------------
     Z7    Z6    Z5    Z4    Z3    Z2    Z1    Z0
```

Partial products, one for each bit in multiplier (each bit needs just one AND gate)

Propagation delay ~2N



Carry-Save Addition

- Speeding up multiplication is a matter of speeding up the summing of the partial products.
- “Carry-save” addition can help.
- Carry-save addition passes (saves) the carries to the output, rather than propagating them.

- Example: sum three numbers, $3_{10} = 0011$, $2_{10} = 0010$, $3_{10} = 0011$

$$\begin{array}{rrl} & 3_{10} & 0011 \\ + & 2_{10} & \underline{0010} \\ & c & 0100 = 4_{10} \\ & s & 0001 = 1_{10} \end{array} \left.\vphantom{\begin{array}{l}a\\a\\a\\a\end{array}}\right\} \text{carry-save add}$$

$$\text{carry-save add} \left\{\vphantom{\begin{array}{l}a\\a\\a\\a\end{array}}\right. \begin{array}{rrl} & c & 0100 = 4_{10} \\ & s & 0001 = 1_{10} \\ & 3_{10} & \underline{0011} \\ & c & 0010 = 2_{10} \\ & s & 0110 = 6_{10} \end{array}$$

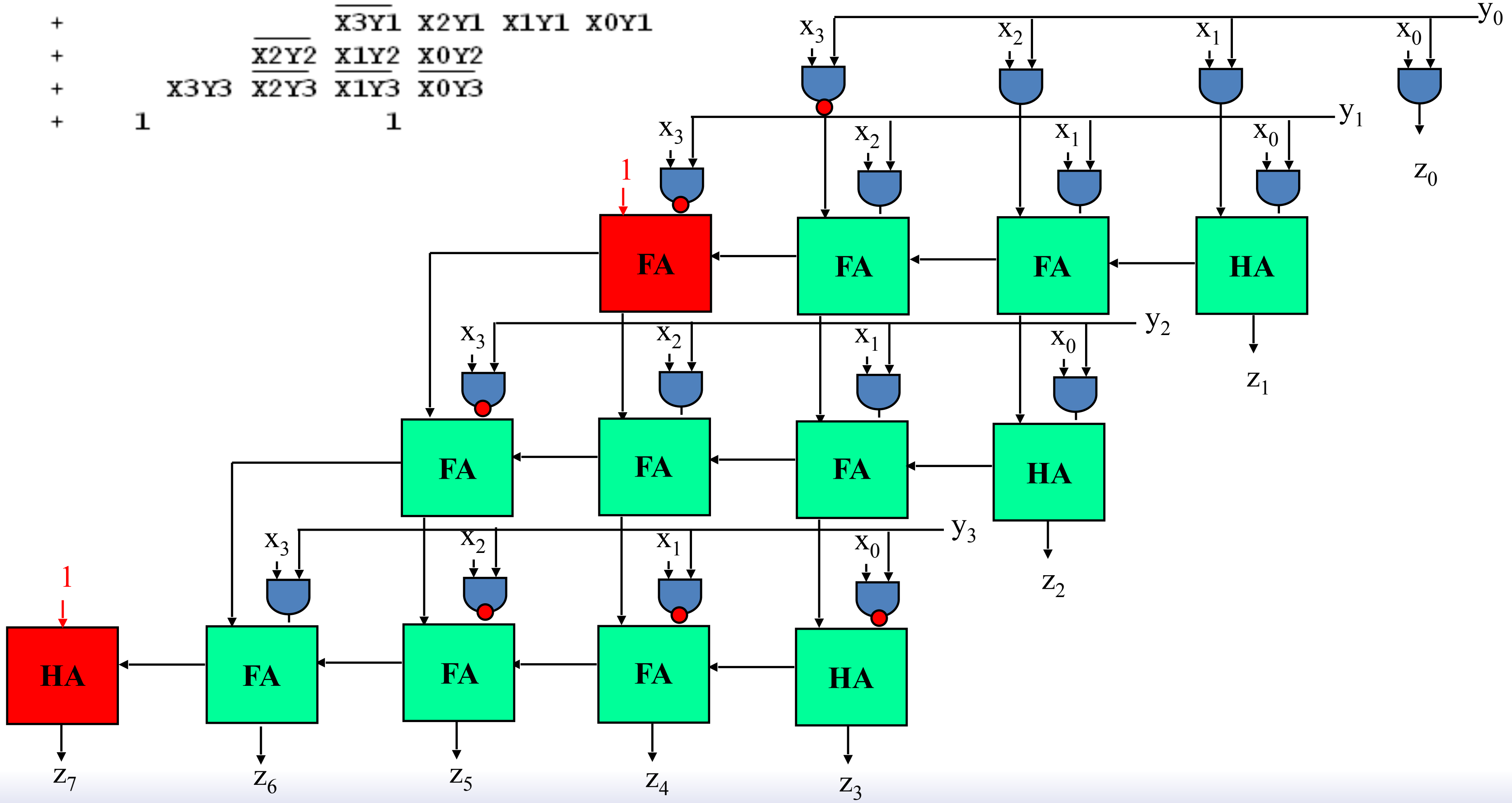
$$\text{carry-propagate add} \left\{\vphantom{\begin{array}{l}a\\a\\a\end{array}}\right. \begin{array}{rrl} & c & 0010 = 2_{10} \\ & s & \underline{0110} = 6_{10} \\ & & 1000 = 8_{10} \end{array}$$

- In general, *carry-save* addition takes in 3 numbers and produces 2.
  - Sometimes called a “3:2 compressor”: 3 input signals into 2 in a potentially lossy operation
- Whereas, *carry-propagate* takes 2 and produces 1.
- With this technique, we can avoid carry propagation until final addition

Signed Multiplication

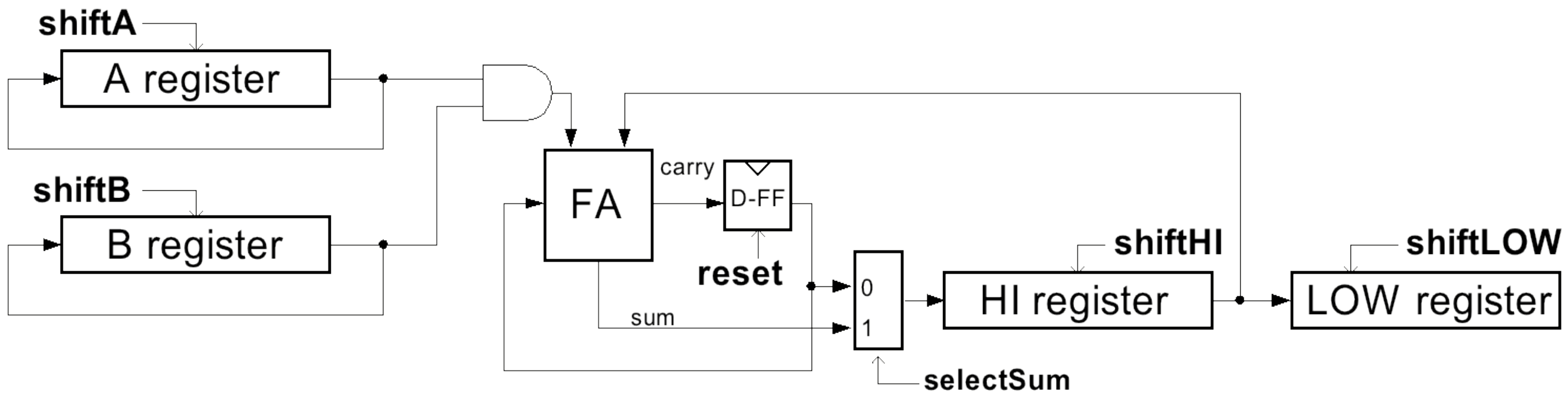
2's Complement Multiplication

$$
\begin{array}{ccccccccc}
 & & & & & \overline{X3Y0} & X2Y0 & X1Y0 & X0Y0 \\
+ & & & & \overline{X3Y1} & X2Y1 & X1Y1 & X0Y1 & \\
+ & & & \overline{X2Y2} & X1Y2 & X0Y2 & & & \\
+ & & X3Y3 & \overline{X2Y3} & \overline{X1Y3} & \overline{X0Y3} & & & \\
+ & 1 & & & 1 & & & &
\end{array}
$$



Bit-serial Multiplier

- Bit-serial multiplier (n^2 cycles, one bit of result per n cycles):



- Control Algorithm:

```
repeat n cycles {   // outer (i) loop
      repeat n cycles{    // inner (j) loop
            shiftA, selectSum, shiftHI
      }
      shiftB, shiftHI, shiftLOW, reset
}
```

Note: The occurrence of a control signal x means x=1. The absence of x means x=0.

Booth recoding

(On-the-fly canonical signed digit encoding!)

current bit pair — from previous bit pair

B_{K+1}	B_K	B_{K-1}	action		
0	0	0	add 0		
0	0	1	add A		
0	1	0	add A		
0	1	1	add 2*A		
1	0	0	sub 2*A		
1	0	1	sub A	$\leftarrow$	-2*A+A
1	1	0	sub A		
1	1	1	add 0	$\leftarrow$	-A+A

$$
\begin{aligned}
B_{K+1,K}*A &= 0*A \rightarrow 0\\
&= 1*A \rightarrow A\\
&= 2*A \rightarrow 4A - 2A\\
&= 3*A \rightarrow 4A - A
\end{aligned}
$$

A "1" in this bit means the previous stage needed to add 4*A. Since this stage is shifted by 2 bits with respect to the previous stage, adding 4*A in the previous stage is like adding A in this stage!

Canonic Signed Digit Representation

- CSD represents numbers using 1, $\bar{1}$, & 0 with the least possible number of non-zero digits.
  - Strings of 2 or more non-zero digits are replaced.
  - Leads to a unique representation.
- To form CSD representation might take 2 passes:
  - First pass: replace all occurrences of 2 or more 1’s:

    01..10 by 10..$\bar{1}$0
  - Second pass: same as above, plus replace 01$\bar{1}$0 by 0010 and 0$\bar{1}$10 by 00$\bar{1}$0
- Examples:

011101 = 29
100$\bar{1}$01 = 32 - 4 + 1

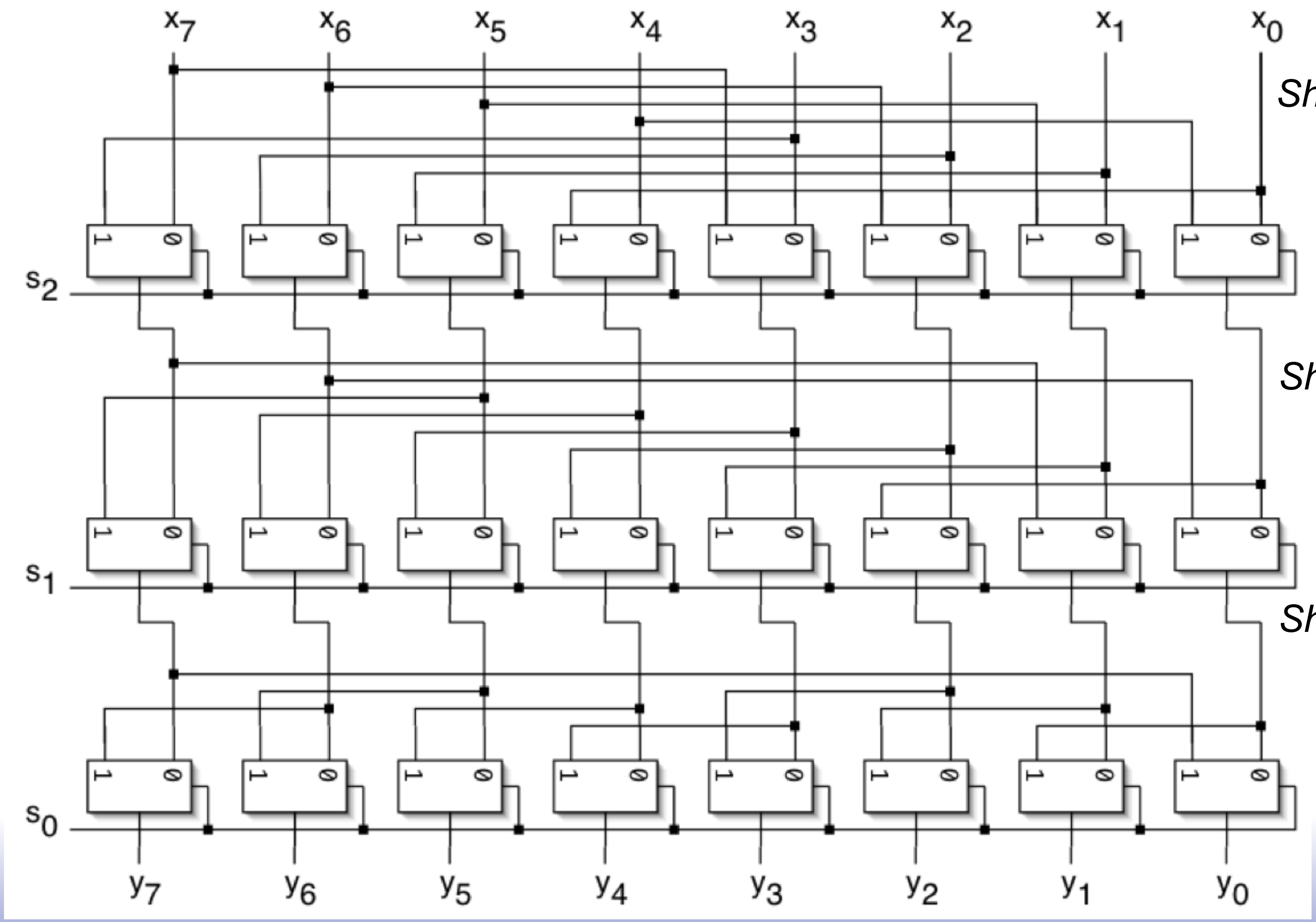
0010111 = 23
001100$\bar{1}$
010$\bar{1}$00$\bar{1}$ = 32 - 8 - 1

0110110 = 54
10$\bar{1}$10$\bar{1}$0
100$\bar{1}$0$\bar{1}$0 = 64 - 8 - 2

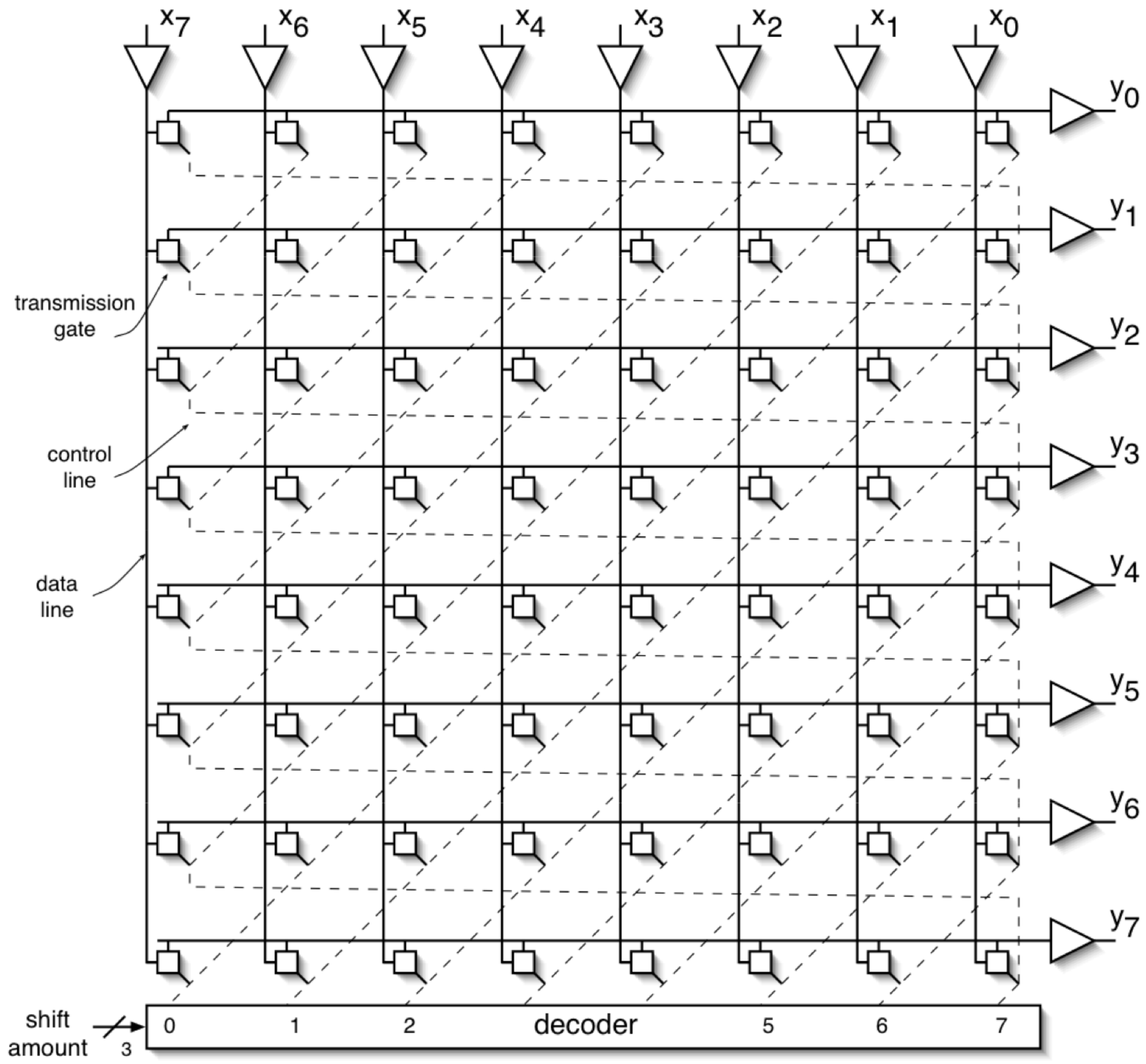
- Can we further simplify the multiplier circuits?

Log Shifter / Rotator

- Log(N) stages, each shifts (or not) by a power of 2 places, $S=[s_2;s_1;s_0]$:



Barrel Shifter



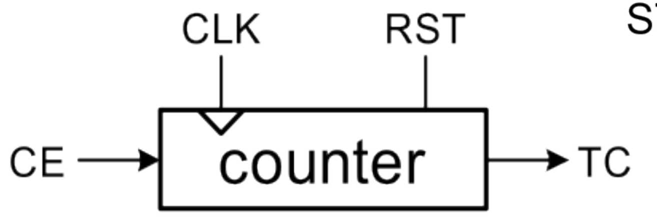
x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7
transmission gate
control line
data line
shift amount
3
0 1 2 decoder 5 6 7

Cost/delay?

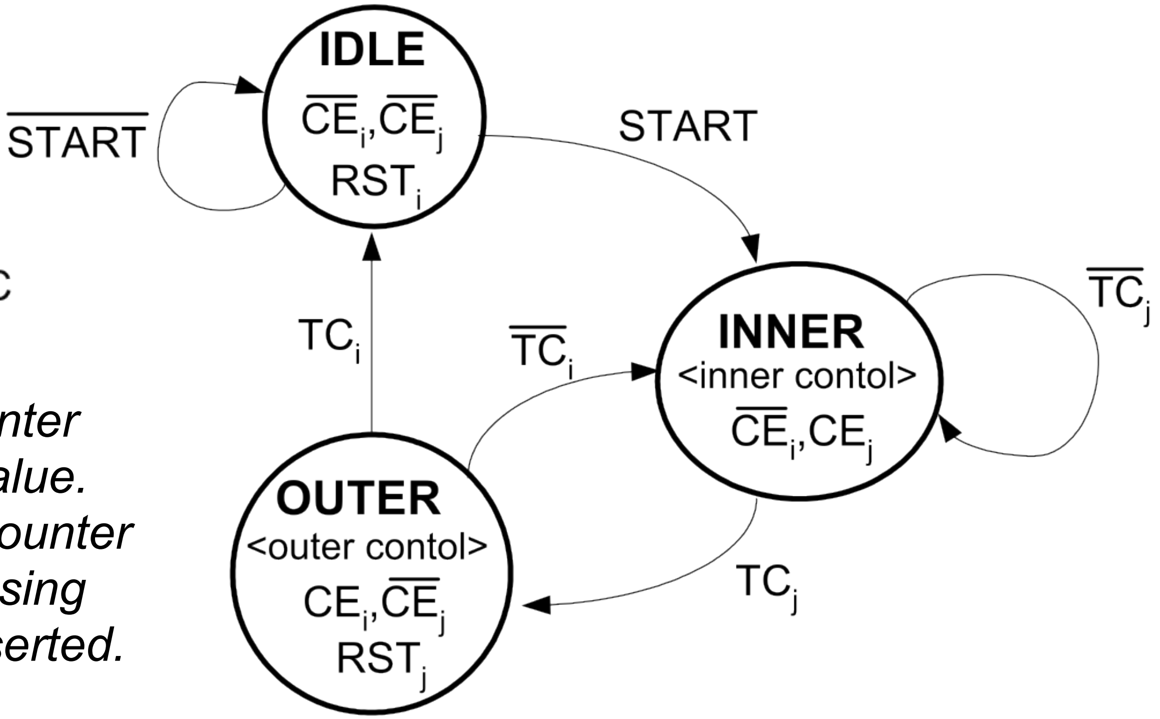
- (don't forget the decoder)

Controller using Counters

- **State Transition Diagram:**
  - Assume presence of two binary counters. An "i" counter for the outer loop and "j" counter for inner loop.



TC *is asserted when the counter reaches it maximum count value.* ***CE*** *is "count enable". The counter increments its value on the rising edge of the clock if CE is asserted.*



Effect of Clock Uncertainties on Maximum Clock Frequency

Clock Constraints in Edge-Triggered Systems

If launching edge is late and receiving edge is early, the data will not be too late if:

$$t_{clk\text{-}q,max} + t_{logic,max} + t_{setup} < T_{CLK} - t_{JS,1} - t_{JS,2} + \delta$$

Minimum cycle time is determined by the maximum delays through the logic

$$t_{clk\text{-}q,max} + t_{logic,max} + t_{setup} - \delta + 2t_{JS} < T_{CLK}$$

Skew can be either positive or negative

Jitter t_{JS} usually expressed as peak-to-peak or n x RMS value

Clock Constraints in Edge-Triggered Systems

If launching edge is early and receiving edge is late:

$$t_{clk\text{-}q,min} + t_{logic,min} - t_{JS,1} > t_{hold} + t_{JS,2} + \delta$$

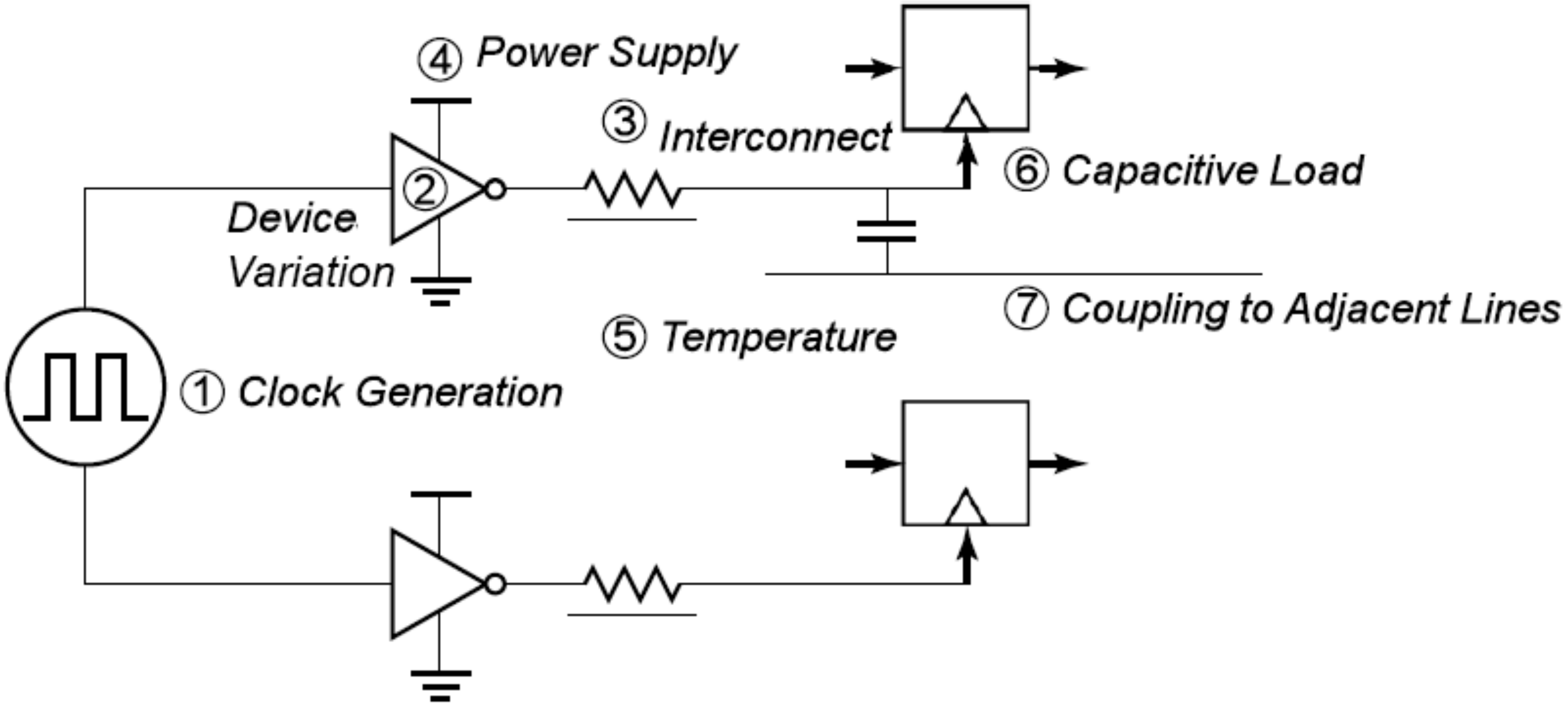
Minimum logic delay

$$t_{clk\text{-}q,min} + t_{logic,min} > t_{hold} + 2t_{JS} + \delta$$

(This assumes jitter at launching and receiving clocks are independent – which usually is not true)

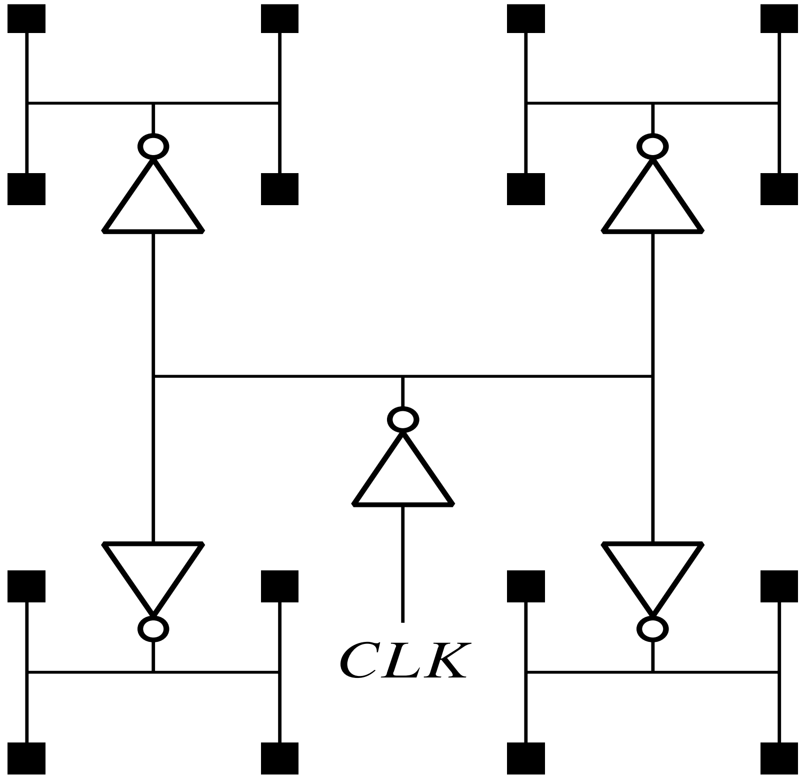
Clock Uncertainties

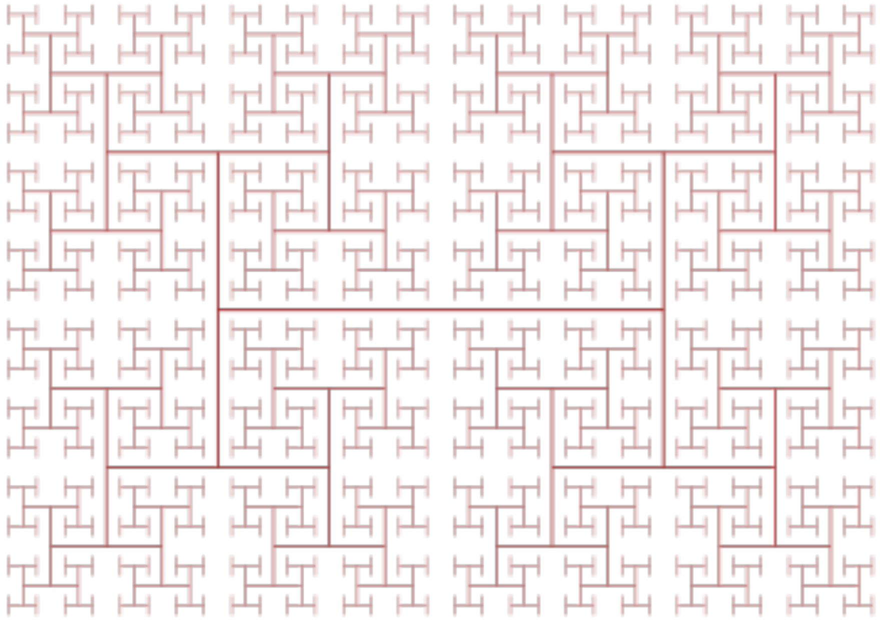
Sources of clock uncertainty



H-Tree

Principles of Good Clock Distribution



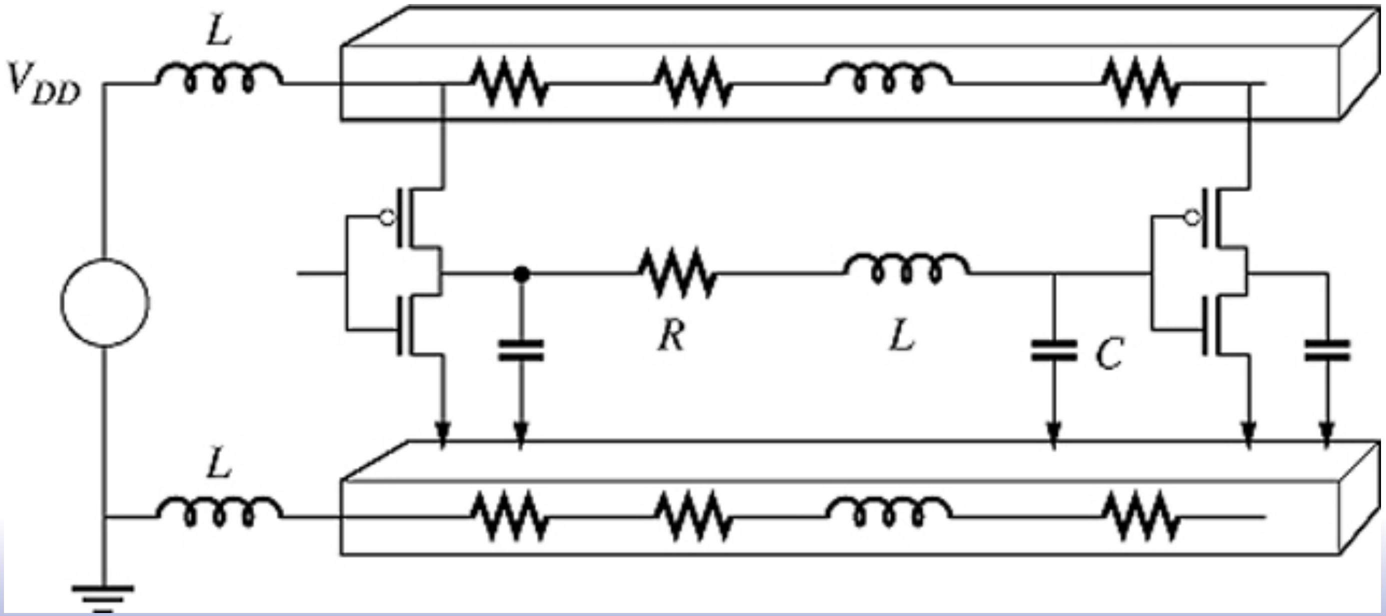


Equal wire length/number of buffers to get to every location

IR and dI/dt effects in Power distribution

Power Supply Impedance

- No voltage source is ideal - ||Z|| > 0
- Two principal elements increase Z:
  - Resistance of supply lines (IR drop)
  - Inductance of supply lines (L·di/dt drop)



Types of Faults in Digital Designs

- Design Bugs (function, timing, power draw)
  - detected and corrected at design time through testing and verification (simulation, static checks)
- Manufacturing Defects (violation of design rules, impurities in processing, statistical variations)
  - post production testing for sorting
  - spare on-chip resources for repair
- Runtime Failures (physical effects and environmental conditions)
  - assuming design is correct and no manufacturing defects

Hamming Error Correcting Code

- Use more parity bits to pinpoint bit(s) in error, so they can be corrected.
- Example: Single error correction (SEC) on 4-bit data
  - use 3 parity bits, with 4-data bits results in 7-bit code word
  - 3 parity bits sufficient to identify any one of 7 code word bits
  - overlap the assignment of parity bits so that a single error in the 7-bit word can be corrected
- **Procedure**: group parity bits so they correspond to subsets of the 7 bits:
  - p_1 protects bits 1,3,5,7
  - p_2 protects bits 2,3,6,7
  - p_3 protects bits 4,5,6,7

1	2	3	4	5	6	7
p_1	p_2	d_1	p_3	d_2	d_3	d_4

Note: number bits from left to right.

Bit position number

$$\left.\begin{array}{l} 001 = 1_{10} \\ 011 = 3_{10} \\ 101 = 5_{10} \\ 111 = 7_{10} \end{array}\right\} p_1$$

$$\left.\begin{array}{l} 010 = 2_{10} \\ 011 = 3_{10} \\ 110 = 6_{10} \\ 111 = 7_{10} \end{array}\right\} p_2$$

$$\left.\begin{array}{l} 100 = 4_{10} \\ 101 = 5_{10} \\ 110 = 6_{10} \\ 111 = 7_{10} \end{array}\right\} p_3$$

The End.

- Special thanks to our GSIs: Sean and Tan. And to Charles (reader).
- Good luck on the final.
- Thanks for a great semester!