EECS151/251A
Spring 2019
Digital Design and Integrated Circuits

Instructor:
John Wawrzynek

Lecture 26:
Wrap-up

Outline

- *Important takeaways*
- *Exam Topics*

Why Study and Learn Digital Design?

- We expect that many of our graduates 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.
  - To become a better user of electronic systems.

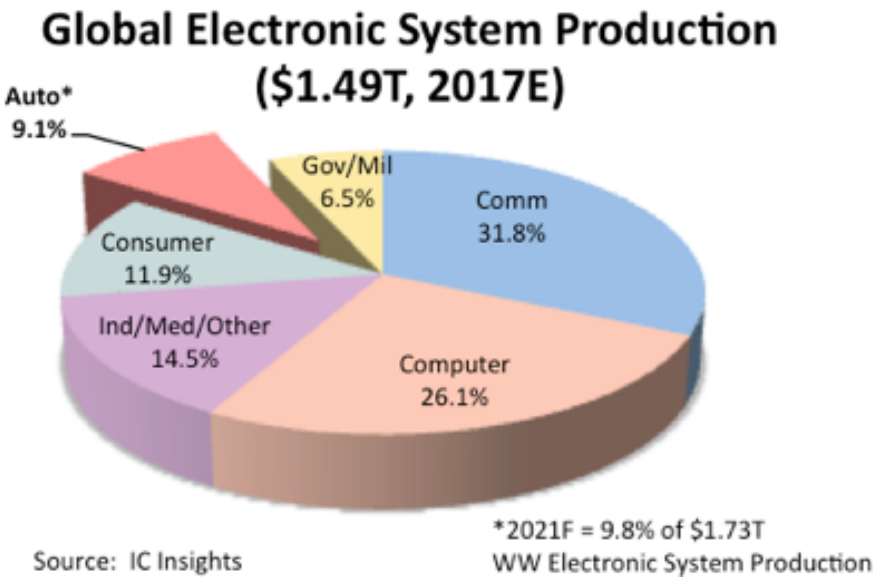
In What Context Will You be Designing?

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 trend is towards “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”.
- 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 in scheduled final exam slot**: Friday May 17, 7-10PM, Hearst Gym 251
  - "Comprehensive" Final Exam
  - Emphasis on second half
- Project interviews
- Project final reports

The exam will take place Friday May 17, 7–10PM in Hearst Gym 251. 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 one 2-sided 8.5 × 11 inch sheet of notes. No calculators, phones, or other electronic devices will be allowed. Slide-rules will be permitted.

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. Sources of Power and Energy consumption in Digital ICs
2. Principles Behind Six Low-power Design Techniques
3. How to Improve Energy Efficiency through Parallelism and Pipelining

4. How to Design a RISC-V Single-Cycle Processor from the ISA
5. Processor Pipelining Hazards and Mechanisms
6. Principle behind and motivations for hardware acceleration
7. Line Drawing Accelerator Design Details
8. Memory Block Internal Architecture
9. SRAM Cell and Read/Write Operation

10. Memory Block Periphery Circuits
11. Memory Decoder Design
12. DRAM Cell and Read/Write Operation
13. Dual-port Memory Architecture
14. Effect of Clock Uncertainties on Maximum Clock Frequency 15. Source of Clock Uncertainties
16. Principle of Good Clock Distribution
17. IR and dI/dt effects in Power distribution

18. Cascading Memory blocks for More Width, Depth, and Ports

19. FIFO Implementation
20. Memory Block Specification in Verilog
21. Serialization versus Parallelization in Iterative Computations

22. Principles of Pipelining and Restrictions of Loops

23. C-Slow Technique for Pipelining Loops
24. Carry Select Adder Design
25. Carry Lookahead and Parallel Prefix Adders

26. Bit-Serial Addition
27. Array Multiplier Design
28. Carry Save Addition
29. Signed Multiplication
30. Booth Encoding
31. Bit-Serial Multiplication
32. CSD Multiplier Design
33. Log and Barrel Shifters Design and Analysis

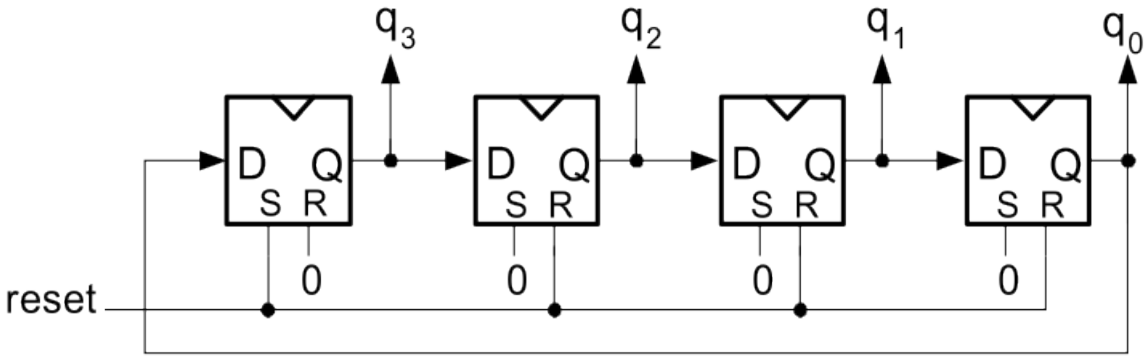
34. Use of Counters in Controller Design
35. Binary Counter Design and Optimization

36. Ring Counter Design
37. LFSR Implementation
38. List Processor Design and Optimizations
39. Modulo Scheduling
40. Types and Sources of Faults in ICs
41. Hamming Codes

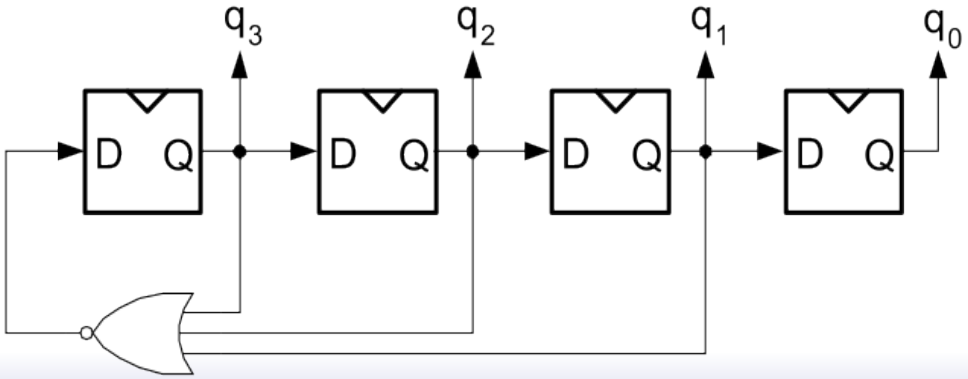
Ring Counters

- "one-hot" counters

0001, 0010, 0100, 1000, 0001, …

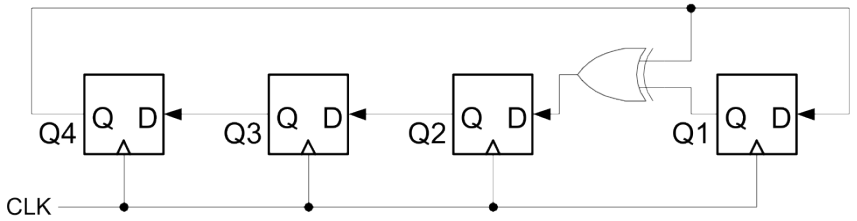


"Self-starting" version:

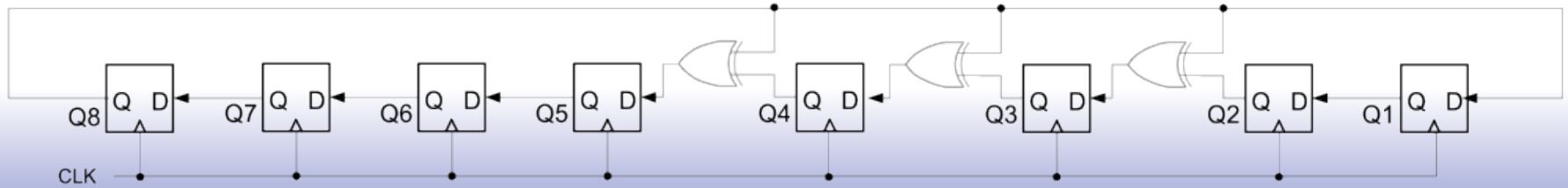


Building an LFSR from a Primitive Polynomial

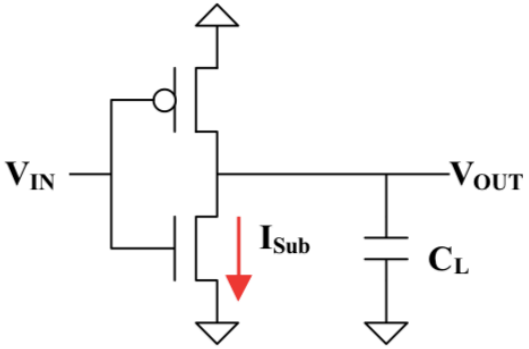
- For *k-bit* LFSR number the flip-flops with FF1 on the right.
- Find the primitive polynomial of the form $x^k + \ldots + 1$.
- The feedback path comes from the Q output of the leftmost FF, corresponding to the x^k term.
- The $x^0 = 1$ term corresponds to connecting the feedback directly to the D input of FF 1.
- Each term of the form x^n corresponds to connecting an xor between FF n and $n+1$.
- 4-bit example, uses $x^4 + x + 1$
  - $x^4 \Leftrightarrow$ FF4's Q output
  - $x \Leftrightarrow$ xor between FF1 and FF2
  - $1 \Leftrightarrow$ FF1's D input



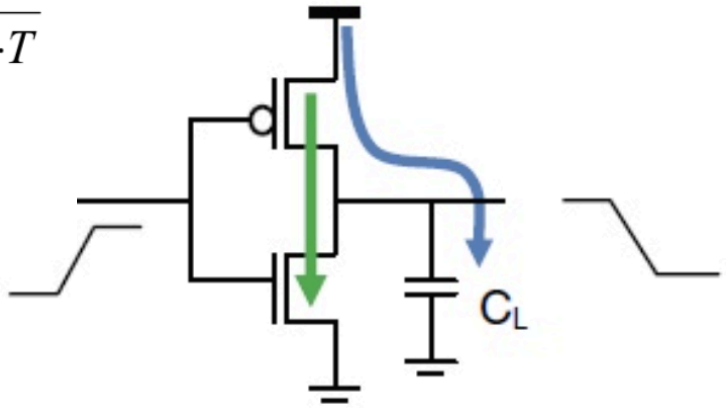
- To build an 8-bit LFSR, use the primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$ and connect xors between FF2 and FF3, FF3 and FF4, and FF4 and FF5.

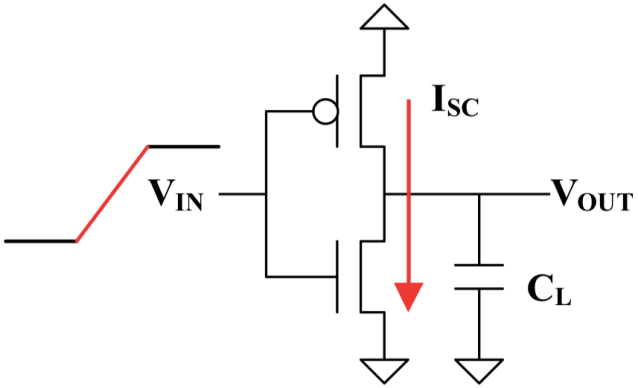


Total Power = $P_{switching} + P_{short\text{-}circuit} + P_{leakage}$



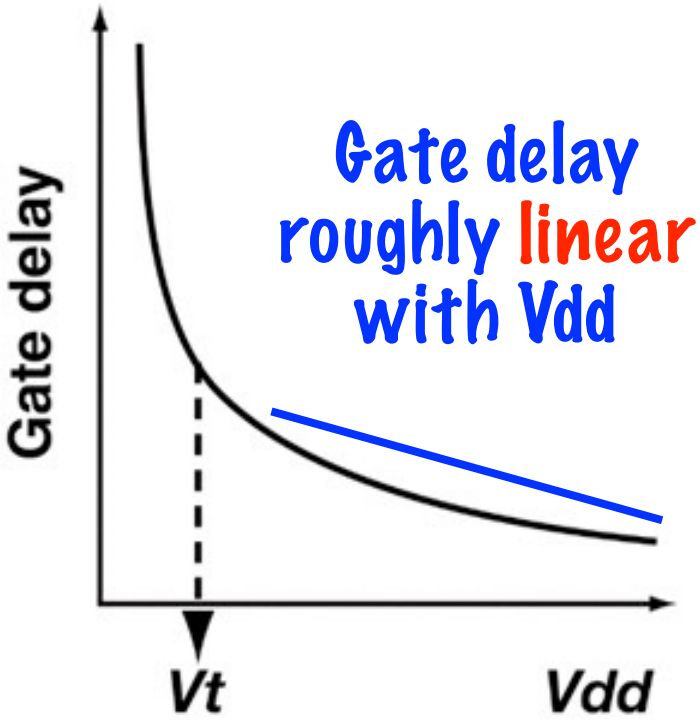
$$I_{DSub} = k \cdot e^{\frac{-q \cdot V_T}{a \cdot k_a \cdot T}}$$





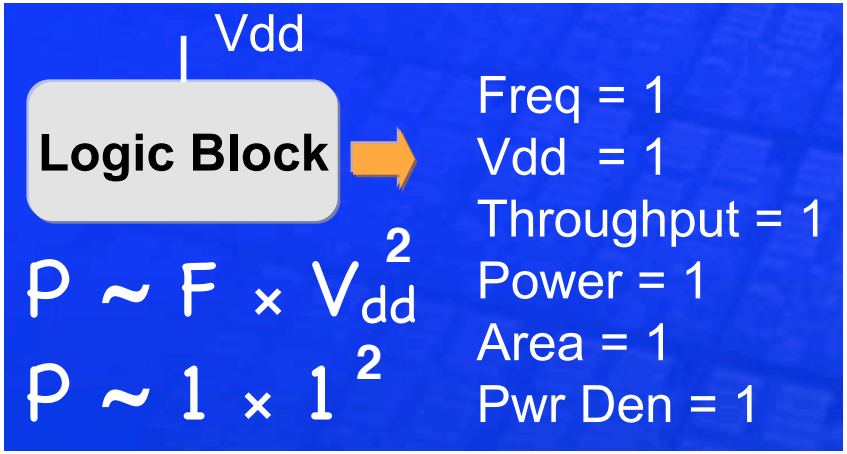
Six low-power design techniques

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



Gate delay roughly linear with Vdd

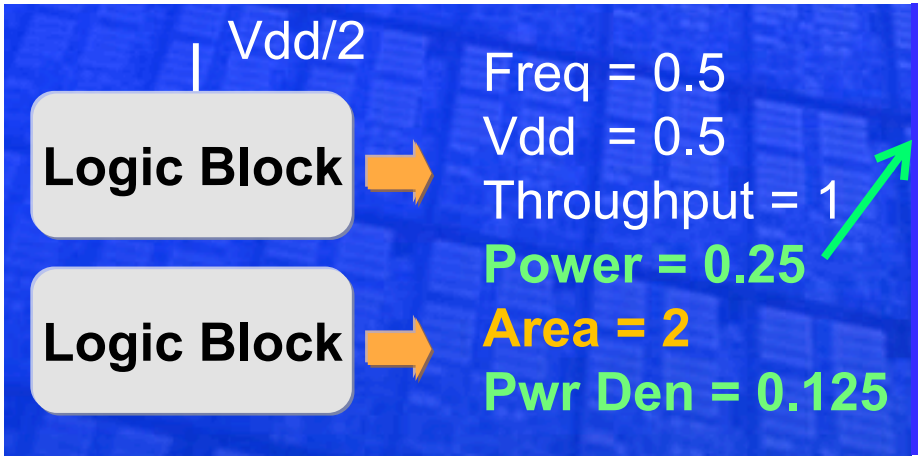
And so, we can transform this:



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

Into this:

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



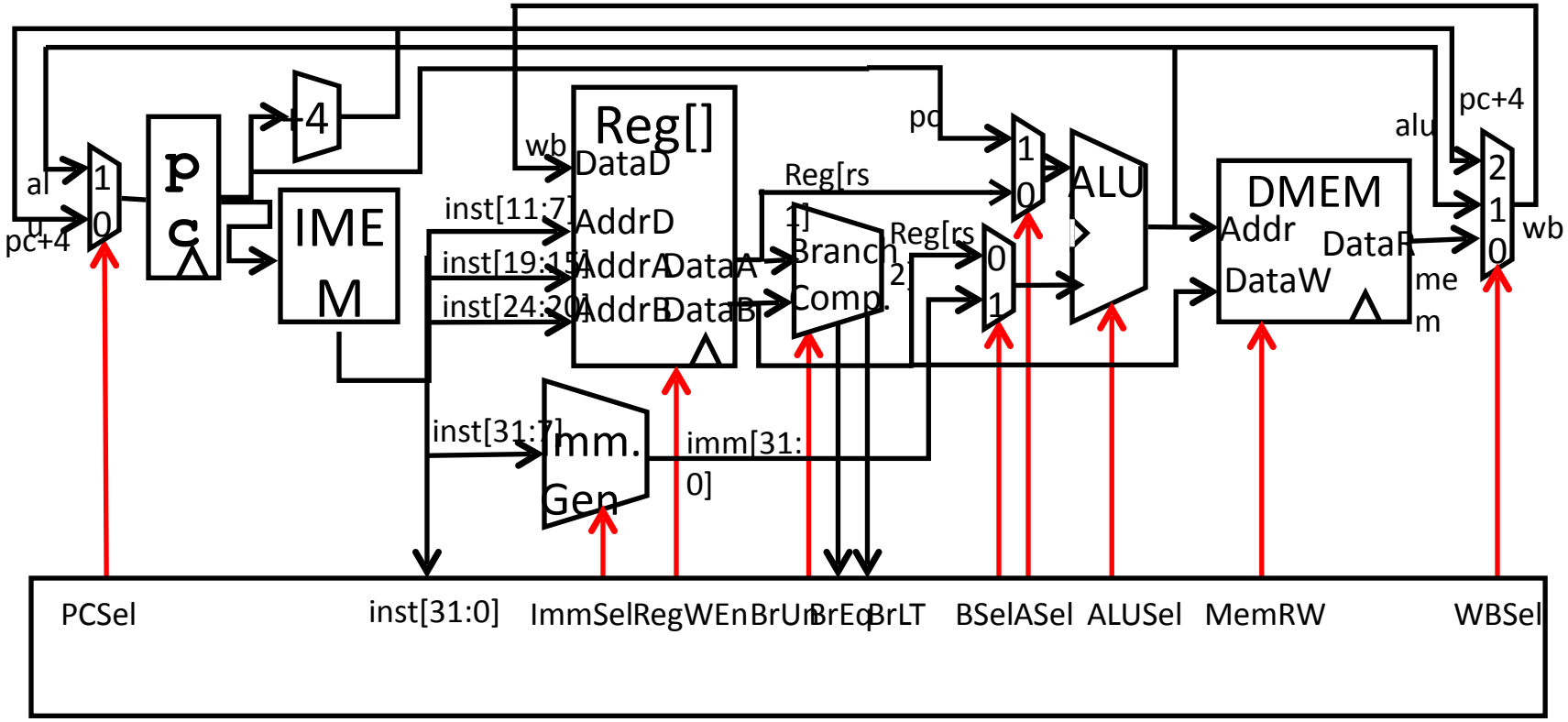
$P \sim \#blks \times F \times V_{dd}^2$

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

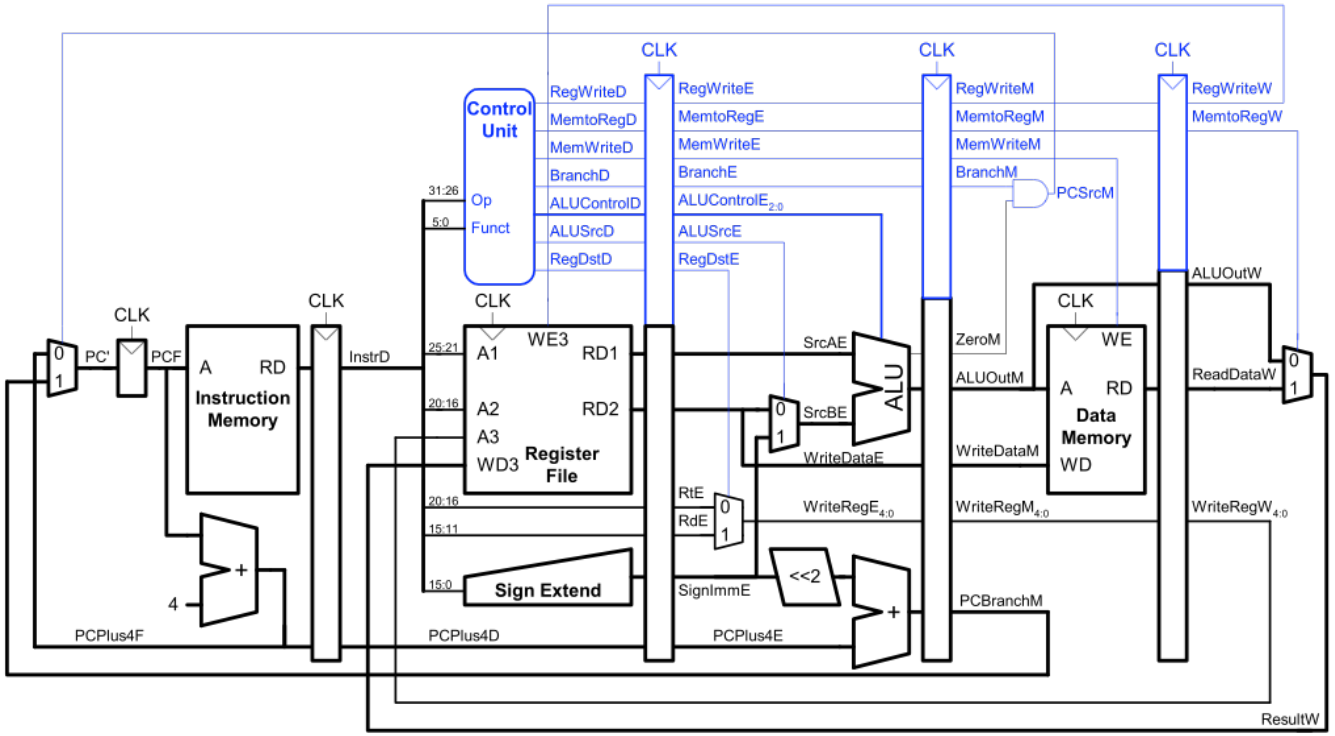
CV^2 power only

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

Single-Cycle RISC-V RV32I Datapath

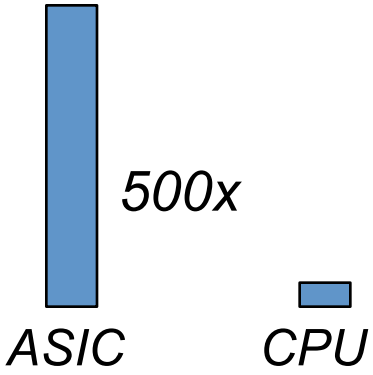


Pipelined Processor

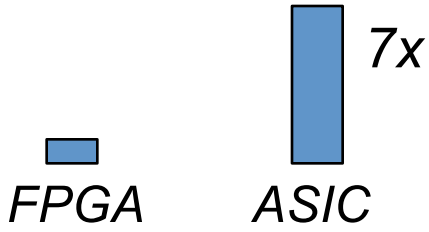


Energy Efficiency of CPU versus ASIC versus FPGA

Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. SIGARCH Comput. Archit. News, 38:37–47, June 2010.



Ian Kuon and Jonathan Rose. Measuring the gap between fpgas and asics. In Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays, FPGA '06, pages 21–30, New York, NY, USA, 2006. ACM

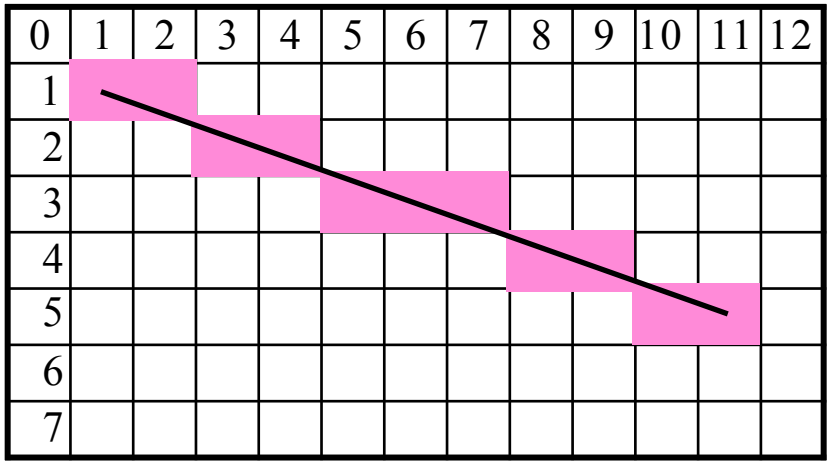


$\therefore$ *FPGA : CPU = 70x*

Similar story for performance efficiency

Line Drawing Algorithm

This version assumes: $x_0 < x_1,\ y_0 < y_1,$ slope =< 45 degrees

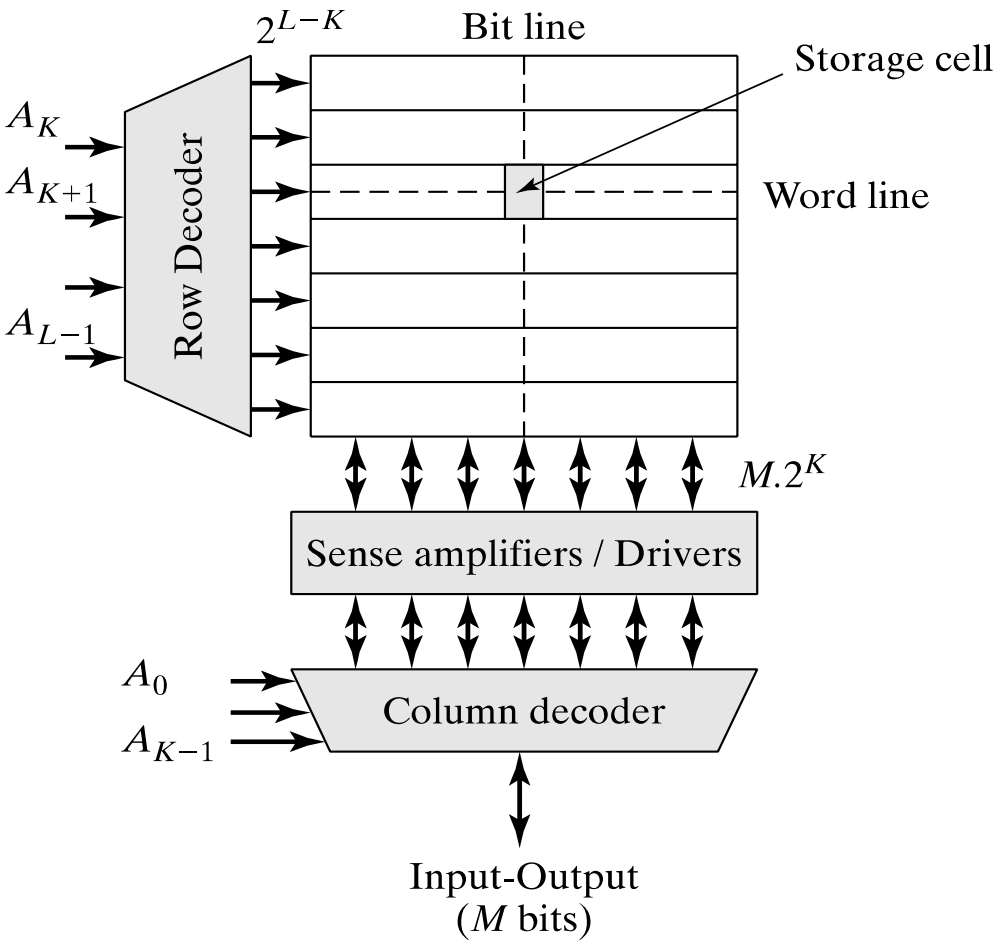



```
function line(x0, x1, y0, y1)
   int deltax := x1 - x0
   int deltay := y1 - y0
   int error := deltax / 2
   int y := y0
   for x from x0 to x1
      plot(x,y)
      error := error - deltay
      if error < 0 then
         y := y + 1
         error := error + deltax
```

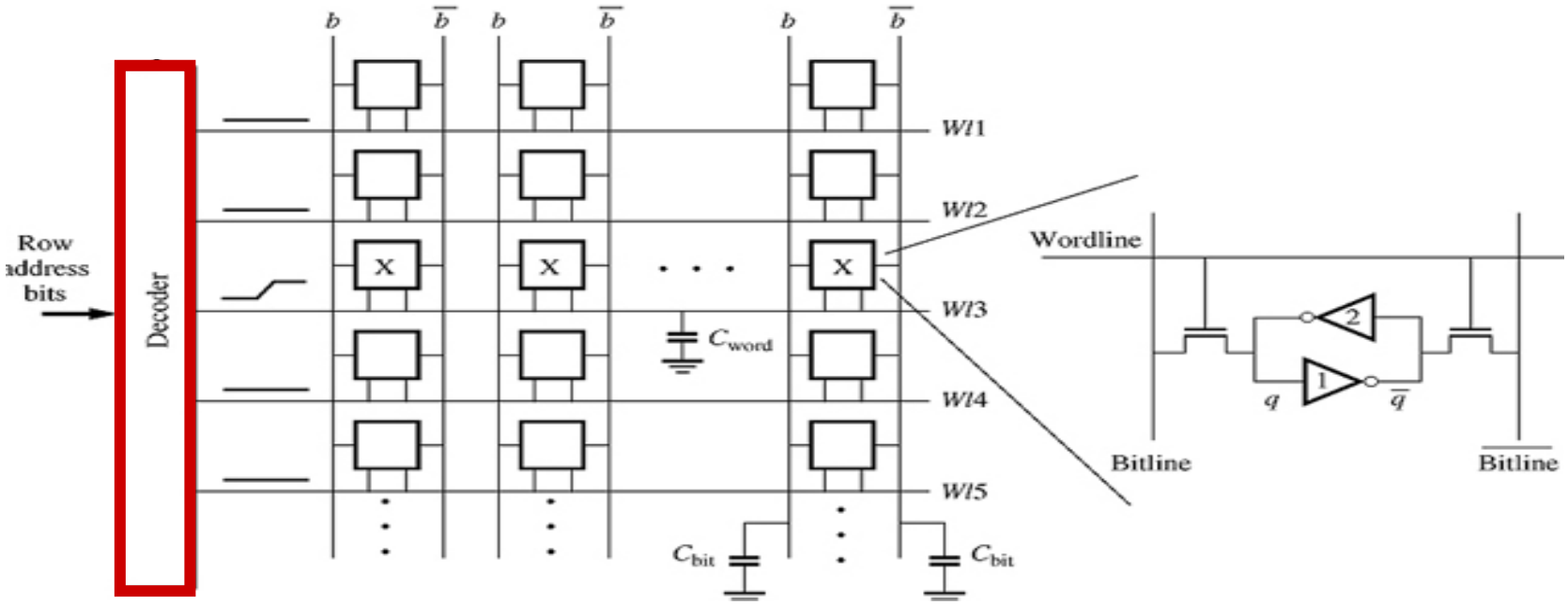
Note: error starts at deltax/2 and gets decremented by deltay for each x. y gets incremented when error goes negative, therefore y gets incremented at a rate proportional to deltax/deltay.

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

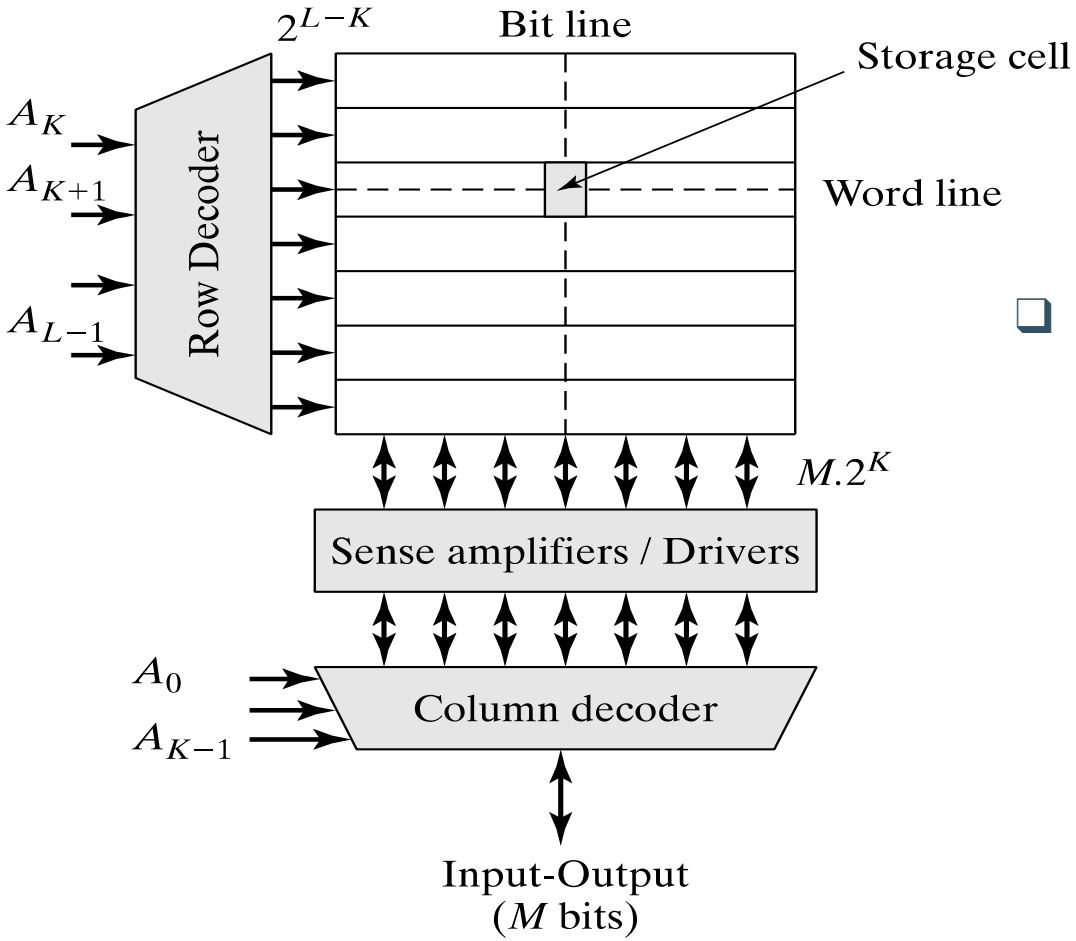


Periphery

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

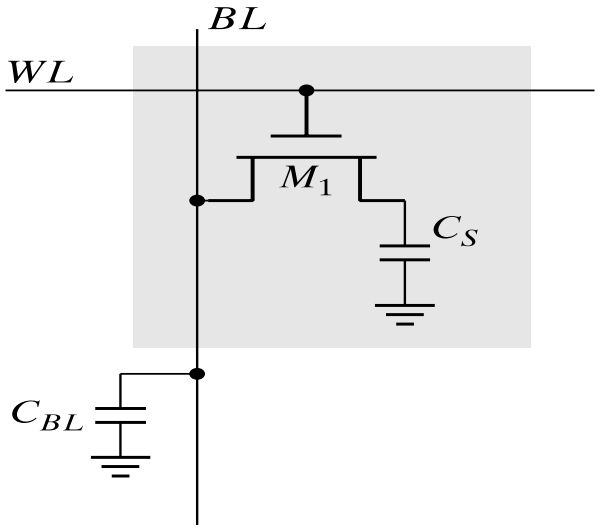
Row Decoder

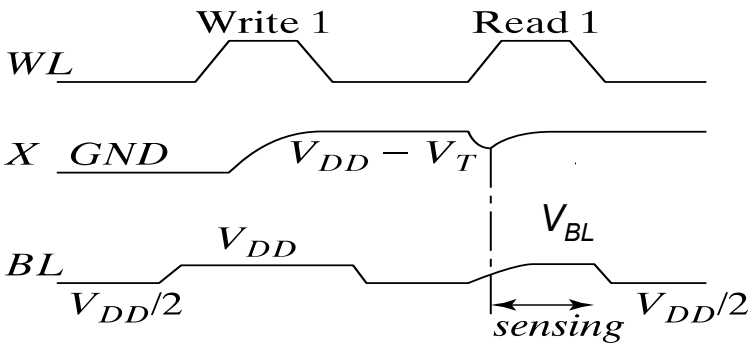
- Expands L-K address lines into 2^{L-K} word lines



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

1-Transistor DRAM Cell





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

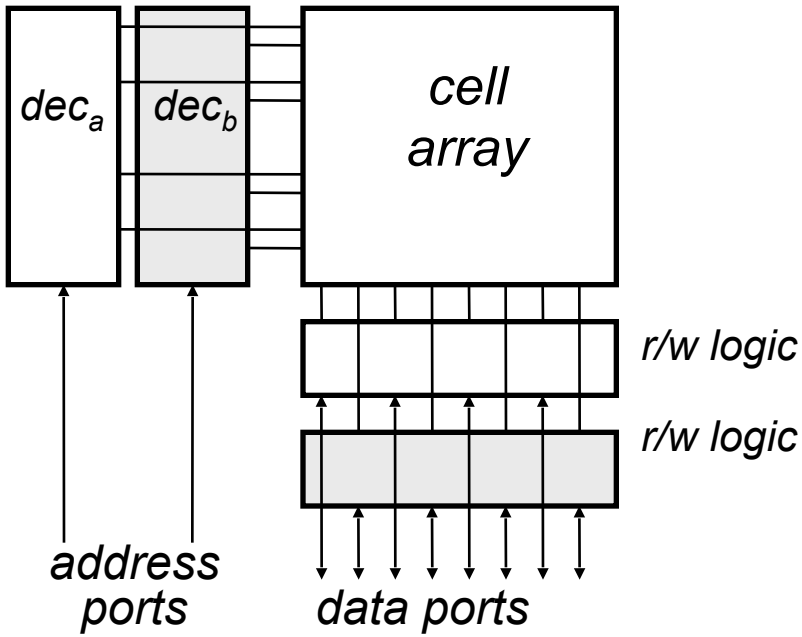
Write: C_S is charged or discharged by asserting WL and BL.
Read: Charge redistribution takes places between bit line and storage capacitance

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

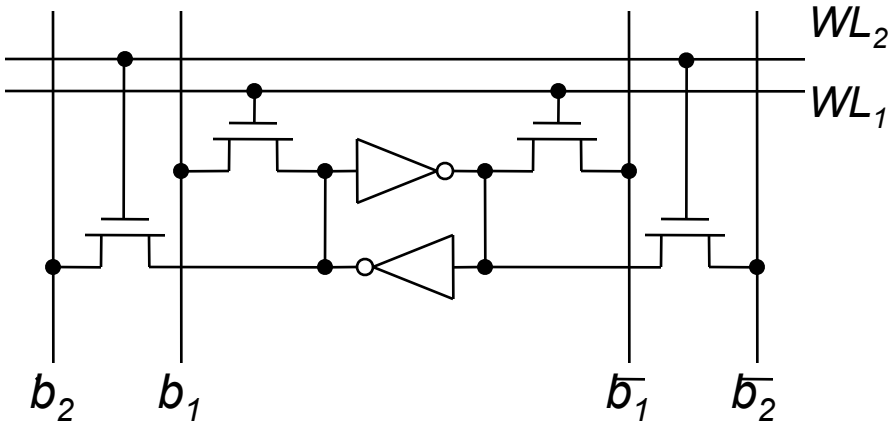
- To get sufficient C_s, special IC process is used
- Cell reading is destructive, therefore read operation always is followed by a write-back
- Cell looses charge (leaks away in ms - highly temperature dependent), therefore cells occasionally need to be “refreshed” - read/write cycle

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.*

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

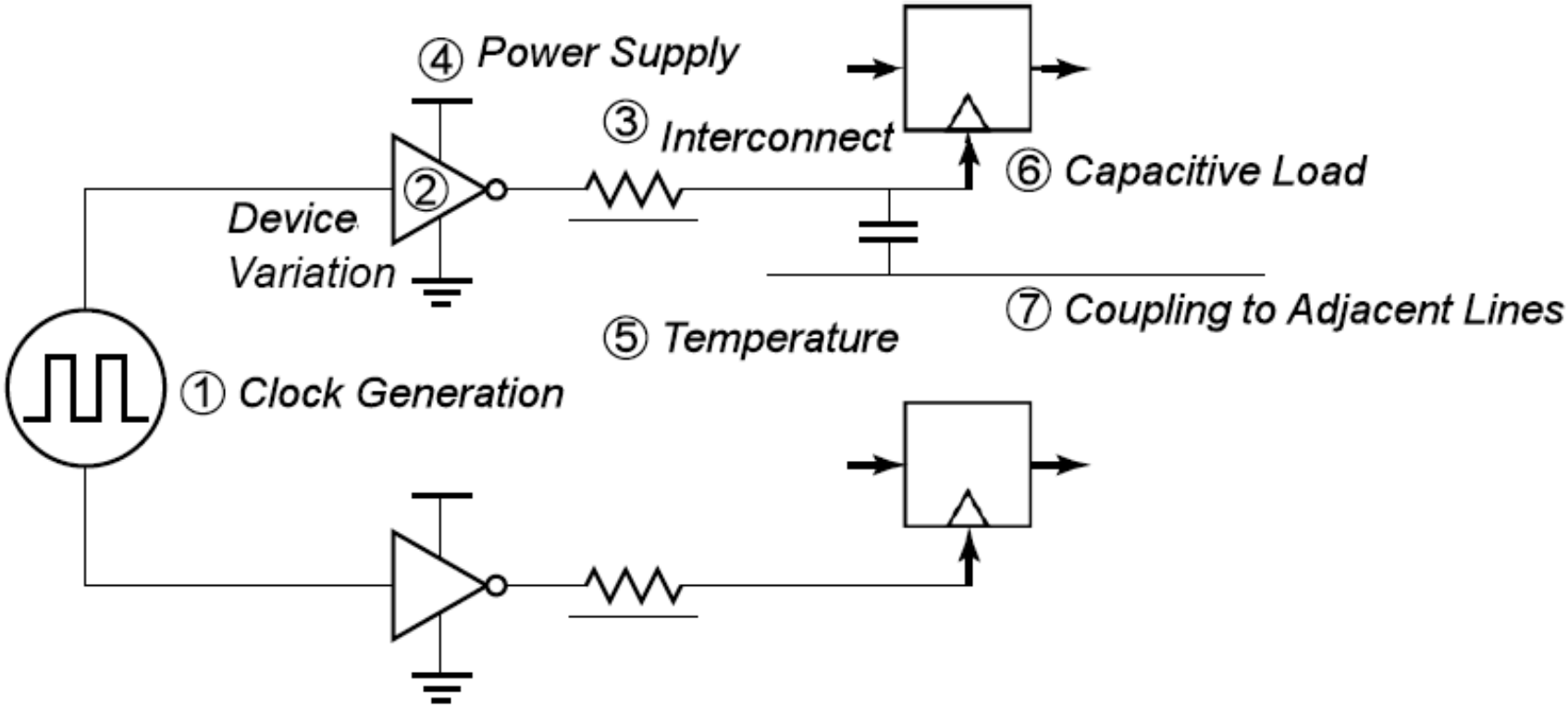
$$t_{clk\text{-}q,max} + t_{logic,max} + t_{setup} - \delta + 2t_{JS} < T_{CLK}$$

Skew can be either positive or negative

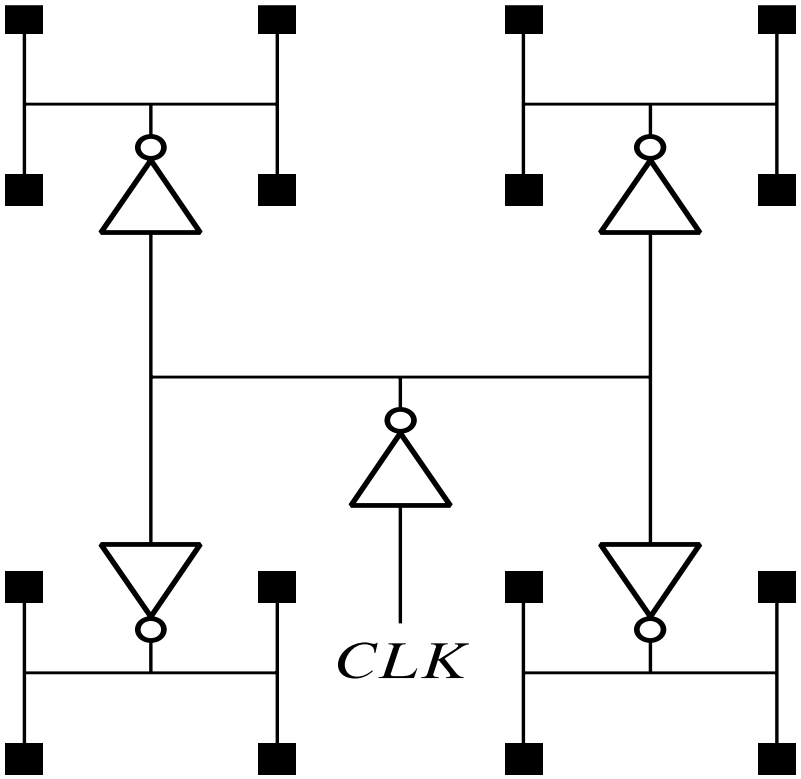
Jitter t_{JS} usually expressed as peak-to-peak or n x RMS value

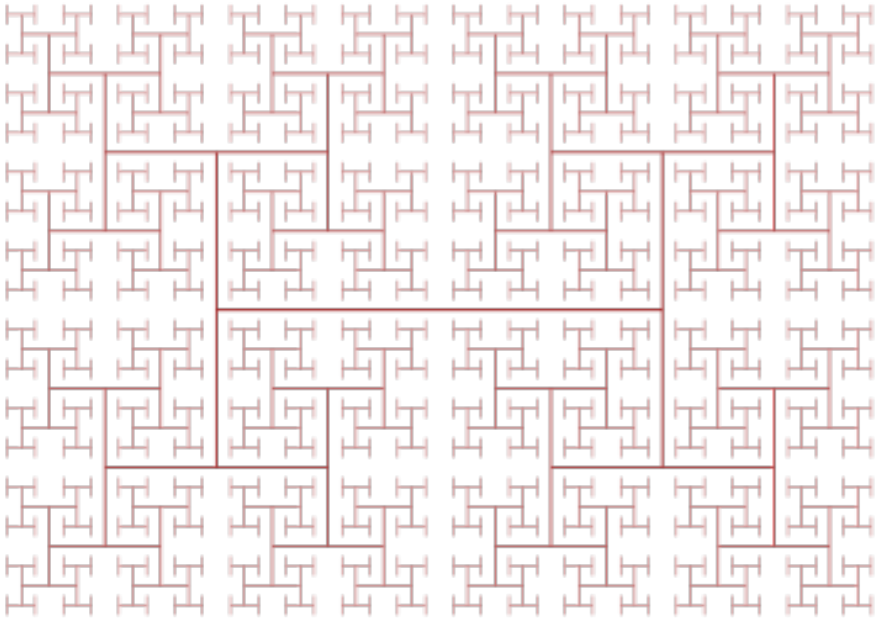
Clock Uncertainties

Sources of clock uncertainty



H-Tree

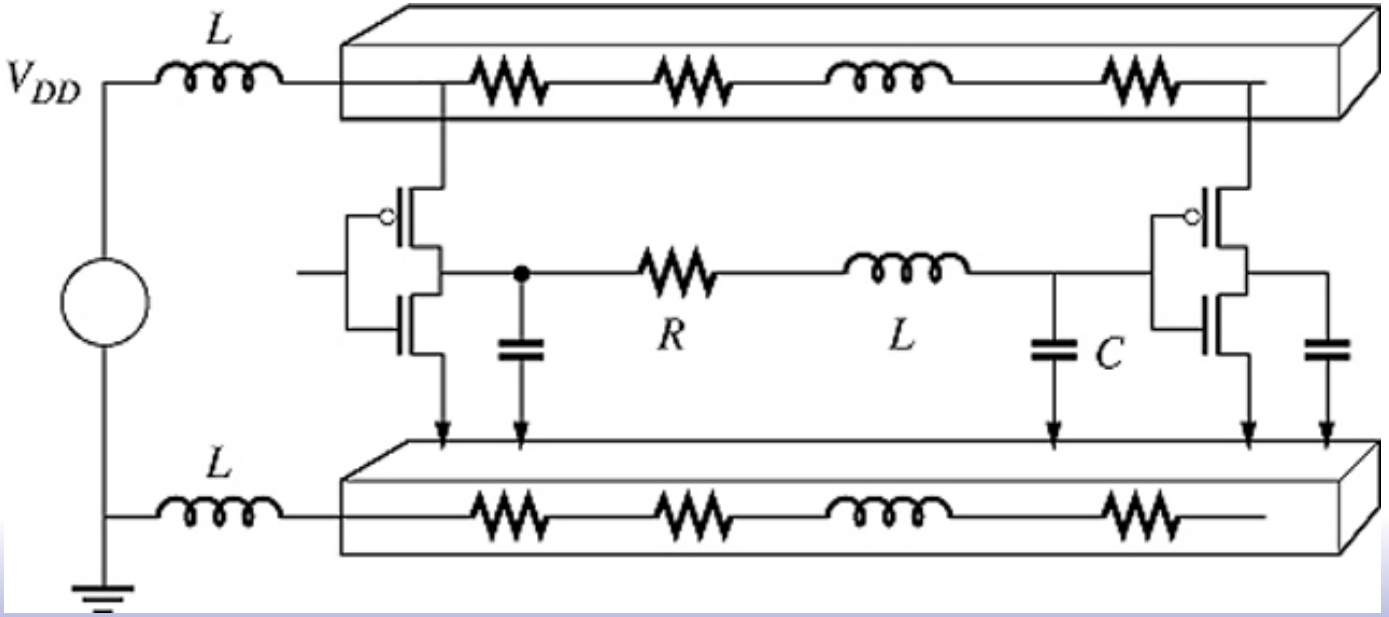




Equal wire length/number of buffers to get to every location

Power Supply Impedance

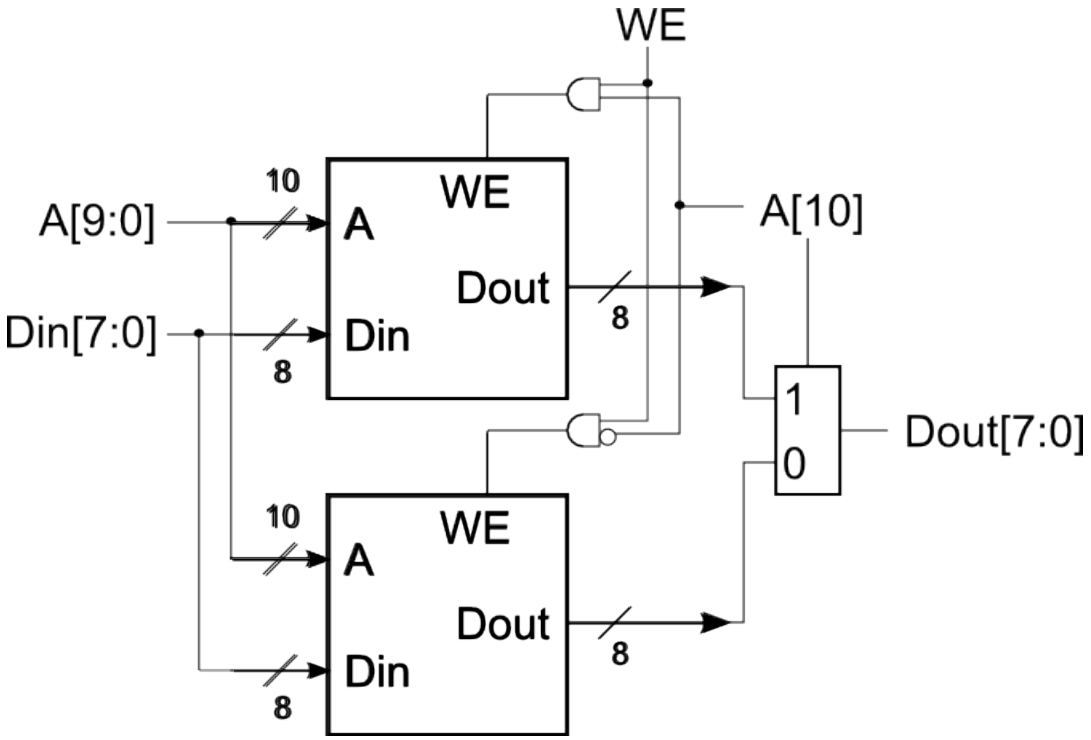
- No voltage source is ideal - ||Z|| > 0
- Two principal elements increase Z:
  - Resistance of supply lines (IR drop)
  - Inductance of supply lines (L·di/dt drop)



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"*

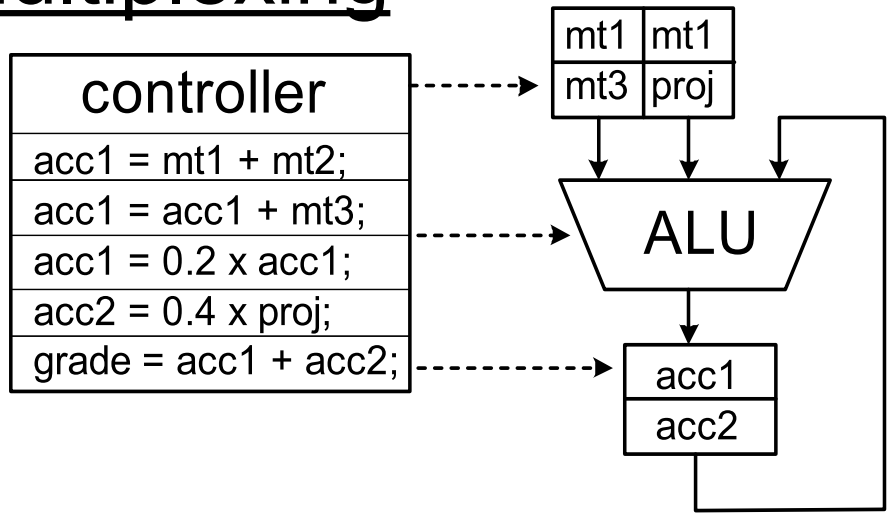
Verilog RAM Specification

```
//
// Single-Port RAM with Asynchronous Read
//
module ramBlock (clk, we, a, di, do);
    input  clk;
    input  we;             // write enable
    input  [19:0] a;       // address
    input  [7:0] di;     // data in
    output [7:0] do;     // data out
    reg    [7:0] ram [1048575:0];  // 8x1Meg
    always @(posedge clk) begin    // Synch write
        if (we)
            ram[a] <= di;
    assign do = ram[a];            // Asynch read
endmodule
```

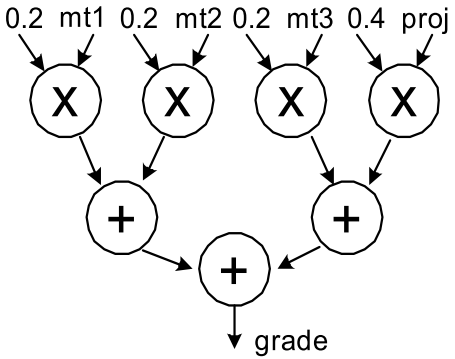
What do the synthesis tools do with this?

Time-Multiplexing

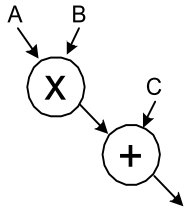
- *Time multiplex* single ALU for all adds and multiplies:
- Attempts to minimize cost at the expense of time.
  - Need to add extra register, muxes, control.

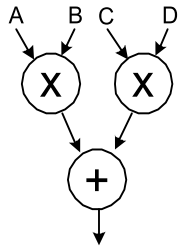


- If we adopt above approach, we can then consider the combinational hardware circuit diagram as an *abstract computation-graph*.



Using other primitives, other coverings are possible.

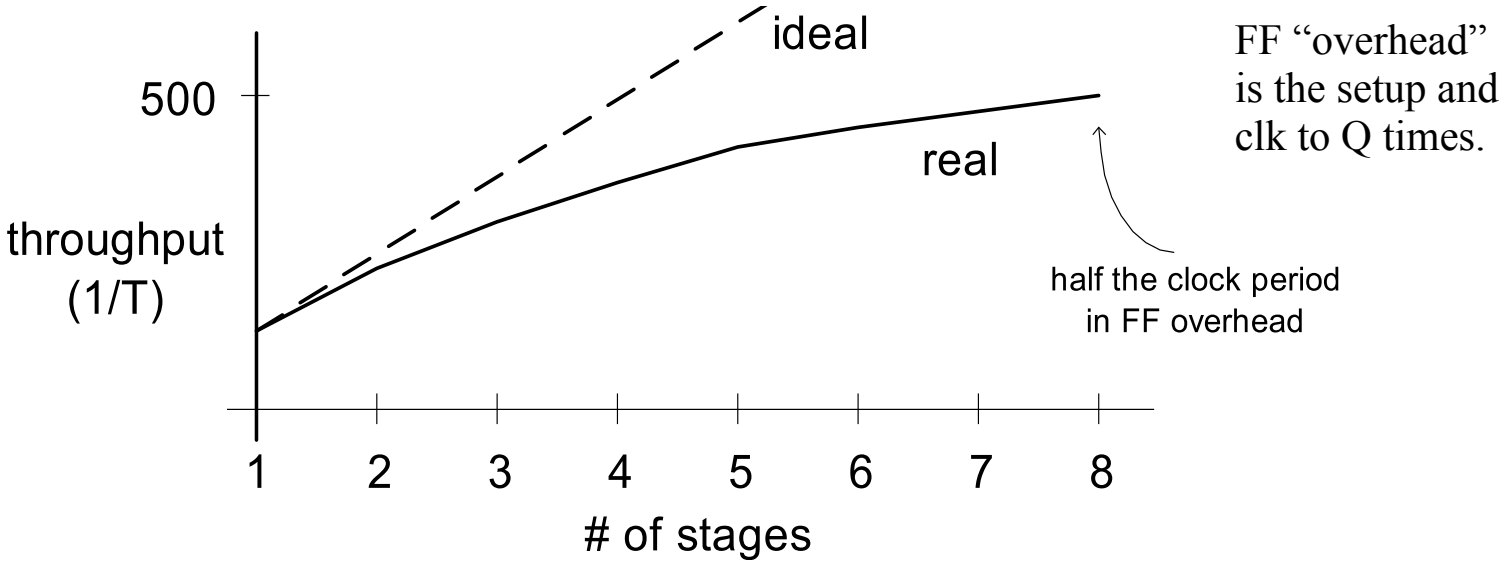




- This time-multiplexing "covers" the computation graph by performing the action of each node one at a time. (Sort of *emulates* it.)

Limits on Pipelining

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



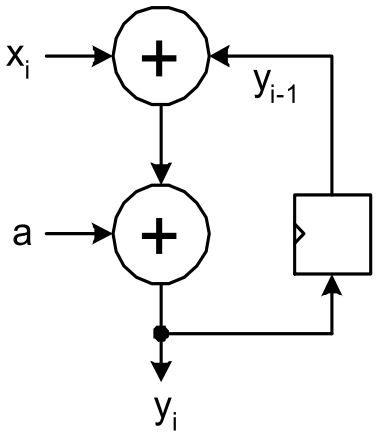
FF "overhead" is the setup and clk to Q times.

- 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

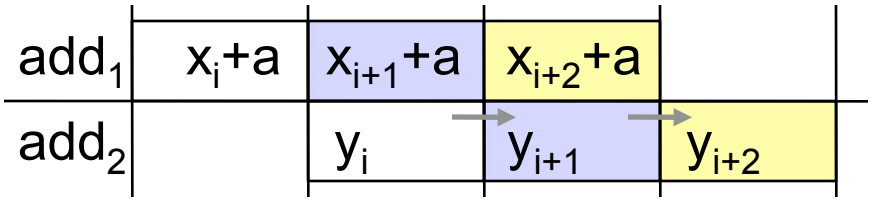
"Loop carry dependency"

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

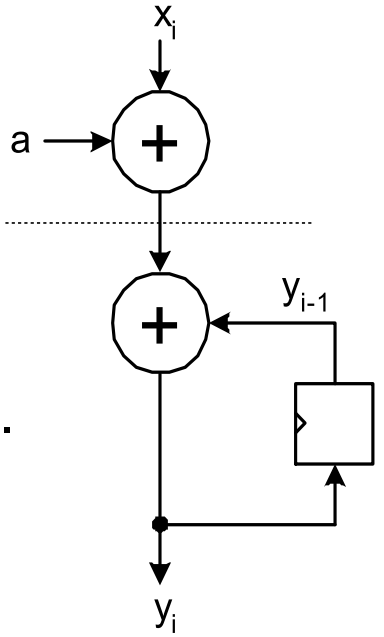


Add is associative and communitive. Therefore we can reorder the computation to shorten the delay of the feedback path:

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



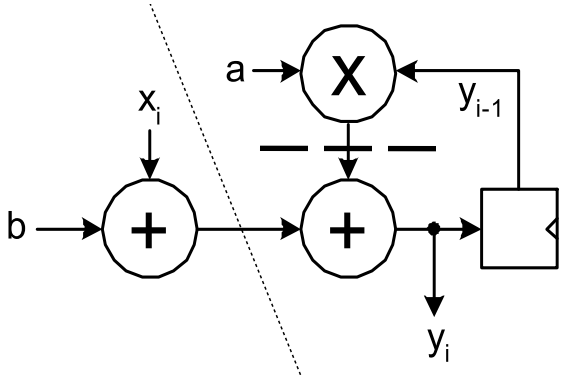
"Shorten" the feedback path.



- 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:

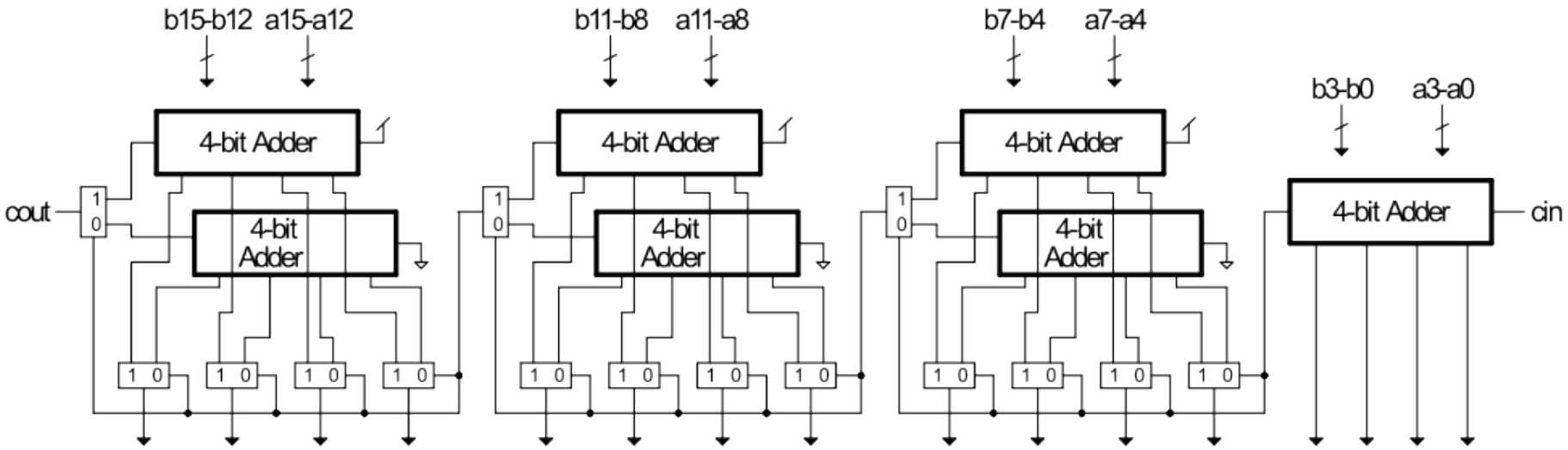
- C computations (in this case C=2) can use the pipeline simultaneously.
- Must be independent.
- Input MUX interleaves input streams.
- Each stream runs at half the pipeline frequency.
- Pipeline achieves full throughput.

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

Carry Select Adder

- Extending Carry-select to multiple blocks

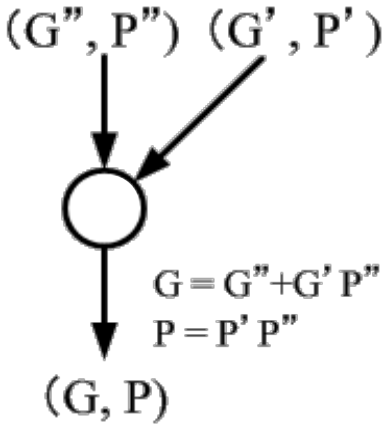


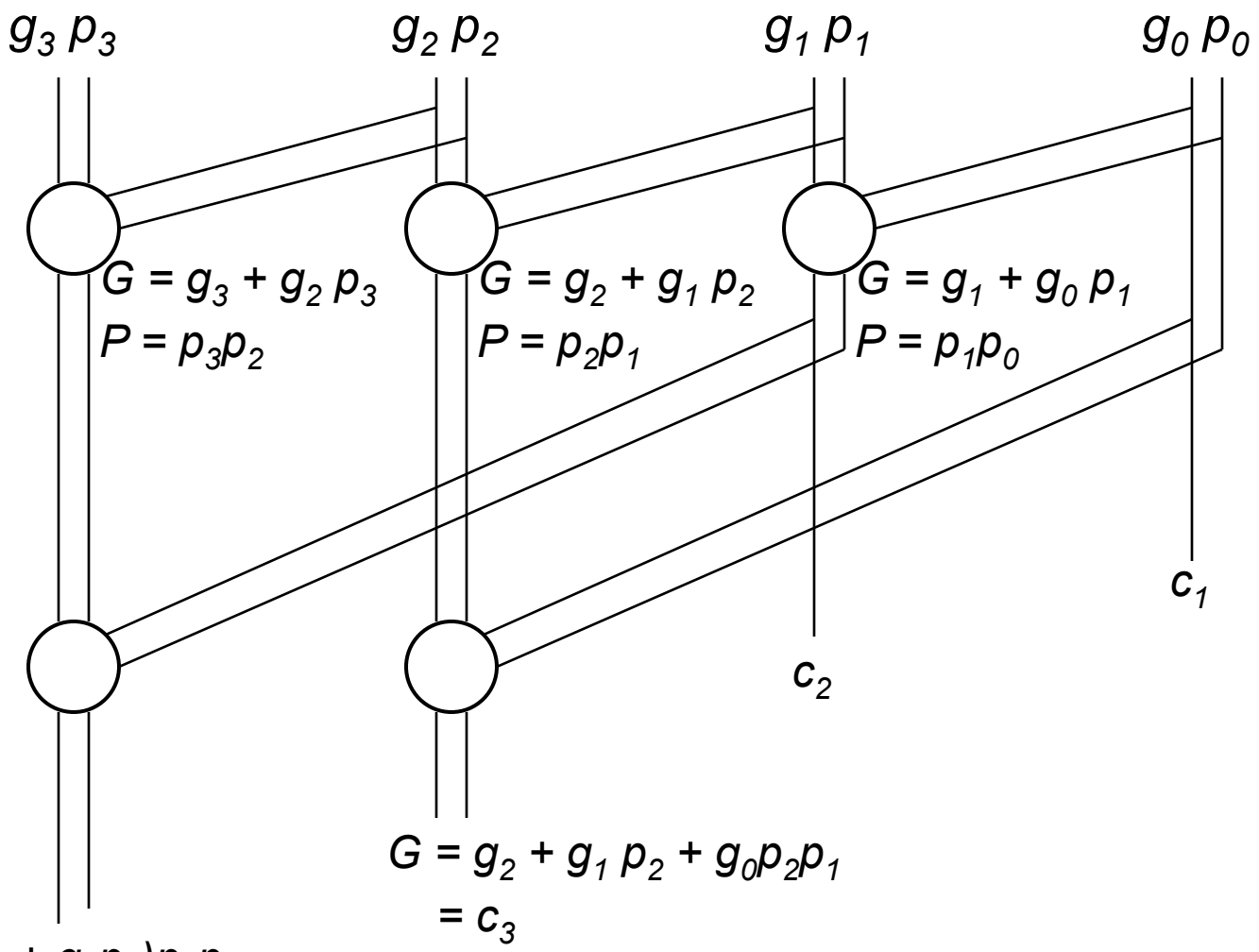
- What is the optimal # of blocks and # of bits/block?
  - If blocks too small delay dominated by total mux delay
  - If blocks too large delay dominated by adder ripple delay

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

T α sqrt(N),
*Cost ≈2*ripple + muxes*

Parallel Prefix Adder Example





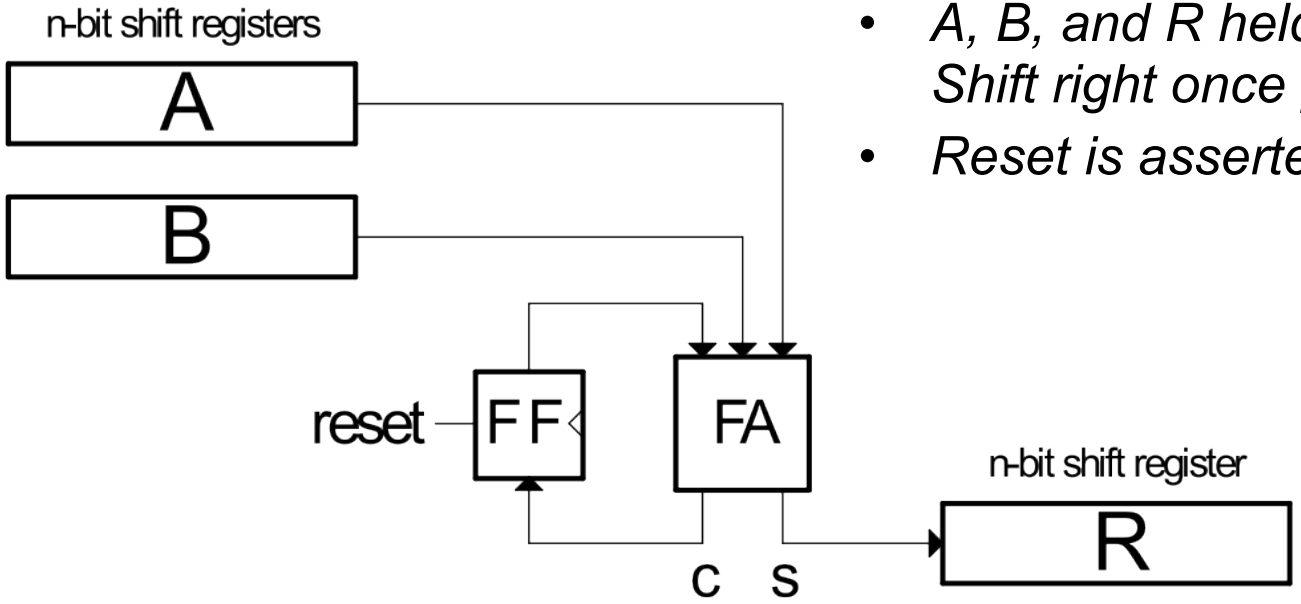
$G = g_3 + g_2 p_3 + (g_1 + g_0 p_1) p_3 p_2$

$= g_3 + g_2 p_3 + g_1 p_3 p_2 + g_0 p_3 p_2 p_1$

$= c_4$

$$s_i = a_i \oplus b_i \oplus c_i = p_i \oplus c_i$$

Bit-serial Adder



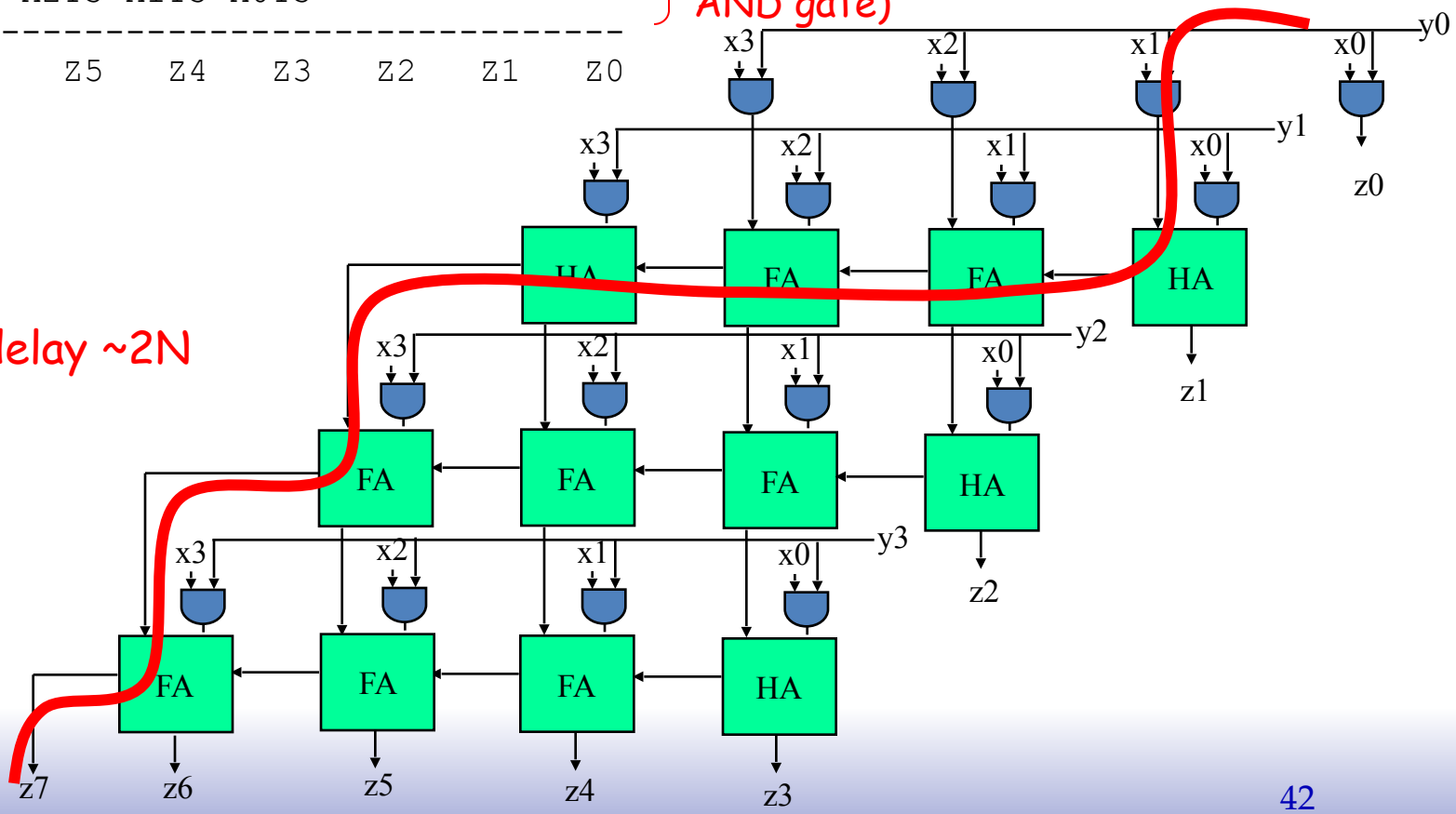
- *A, B, and R held in shift-registers. Shift right once per clock cycle.*
- *Reset is asserted by controller.*

- 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
```

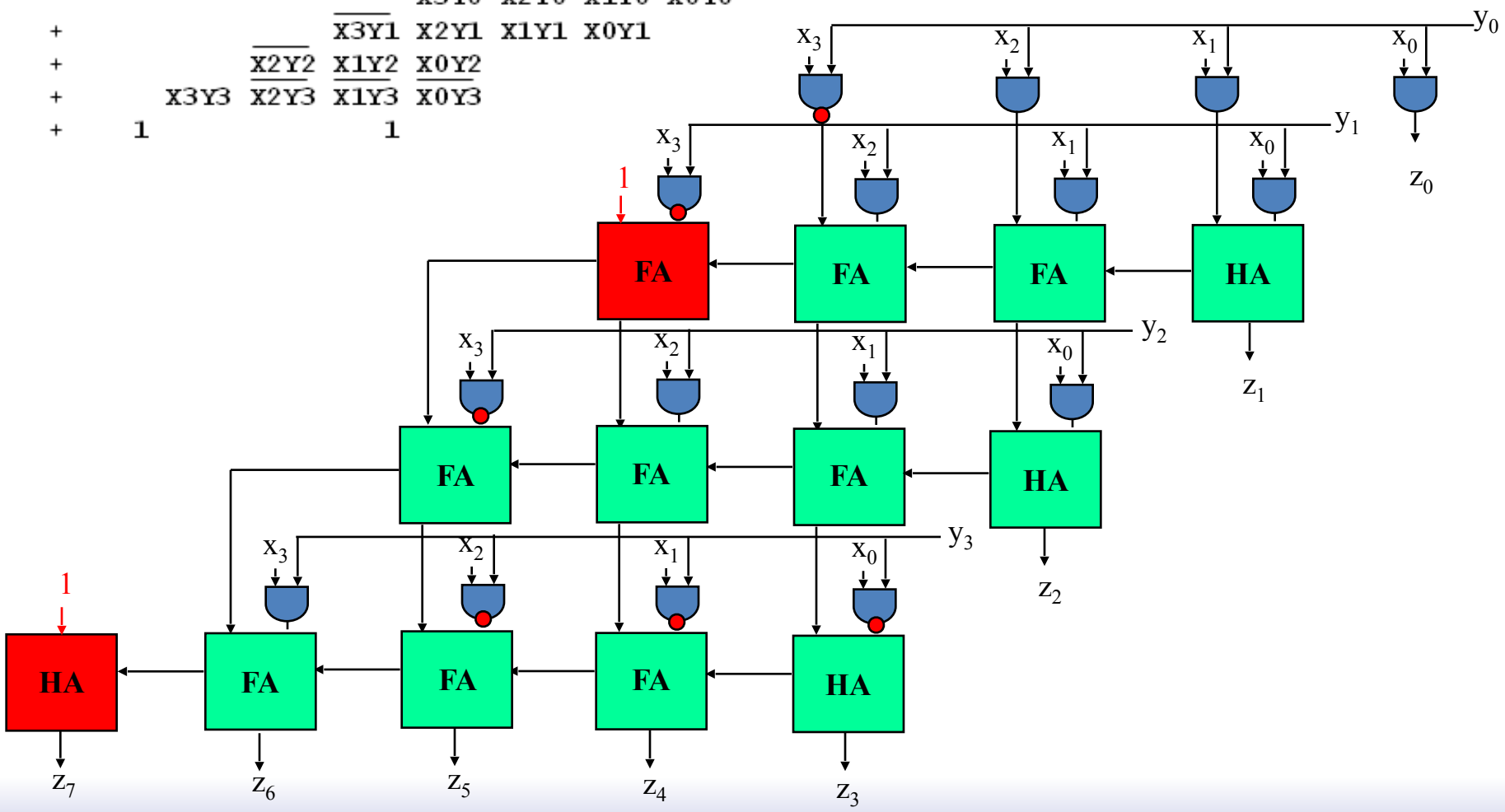
Partial products, one for each bit in multiplier (each bit needs just one AND gate)



Propagation delay ~2N

2's Complement Multiplication

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



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, $3_{10} = 0011$, $2_{10} = 0010$, $3_{10} = 0011$

$$
\begin{array}{lll}
 & 3_{10} & 0011 \\
+ & 2_{10} & 0010 \\
\hline
 & c & 0100 = 4_{10} \\
 & s & 0001 = 1_{10}
\end{array}
$$

carry-save add

$$
\begin{array}{lll}
 & 3_{10} & 0011 \\
\hline
 & c & 0010 = 2_{10} \\
 & s & 0110 = 6_{10} \\
\hline
 & & 1000 = 8_{10}
\end{array}
$$

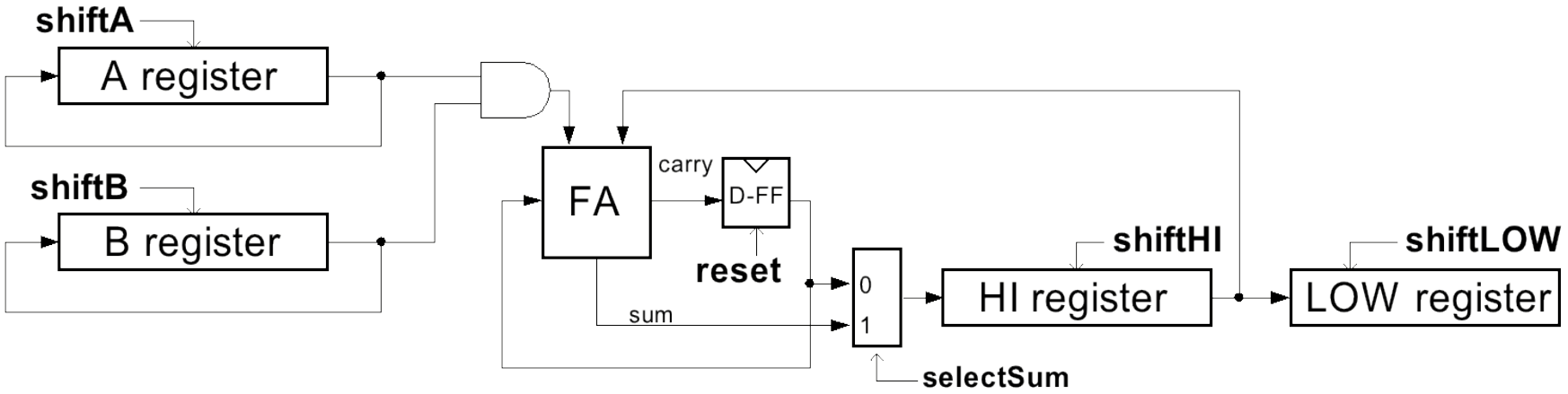
carry-save add

carry-propagate add

- 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

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	← -2*A+A
1	1	0	sub A	
1	1	1	add 0	← -A+A

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

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

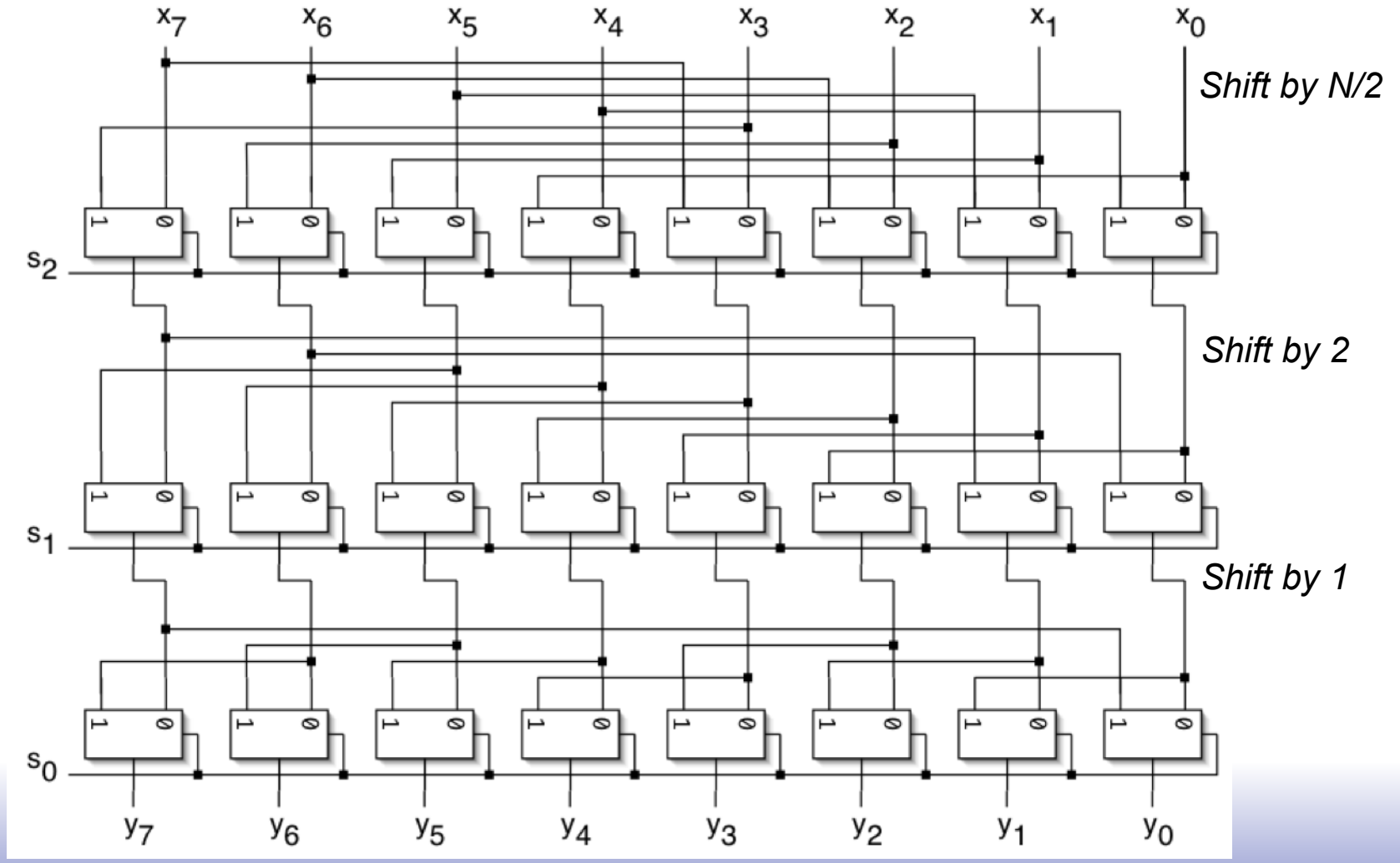
0010111 = 23
0011001$\bar{1}$
0101001$\bar{1}$ = 32 - 8 - 1

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

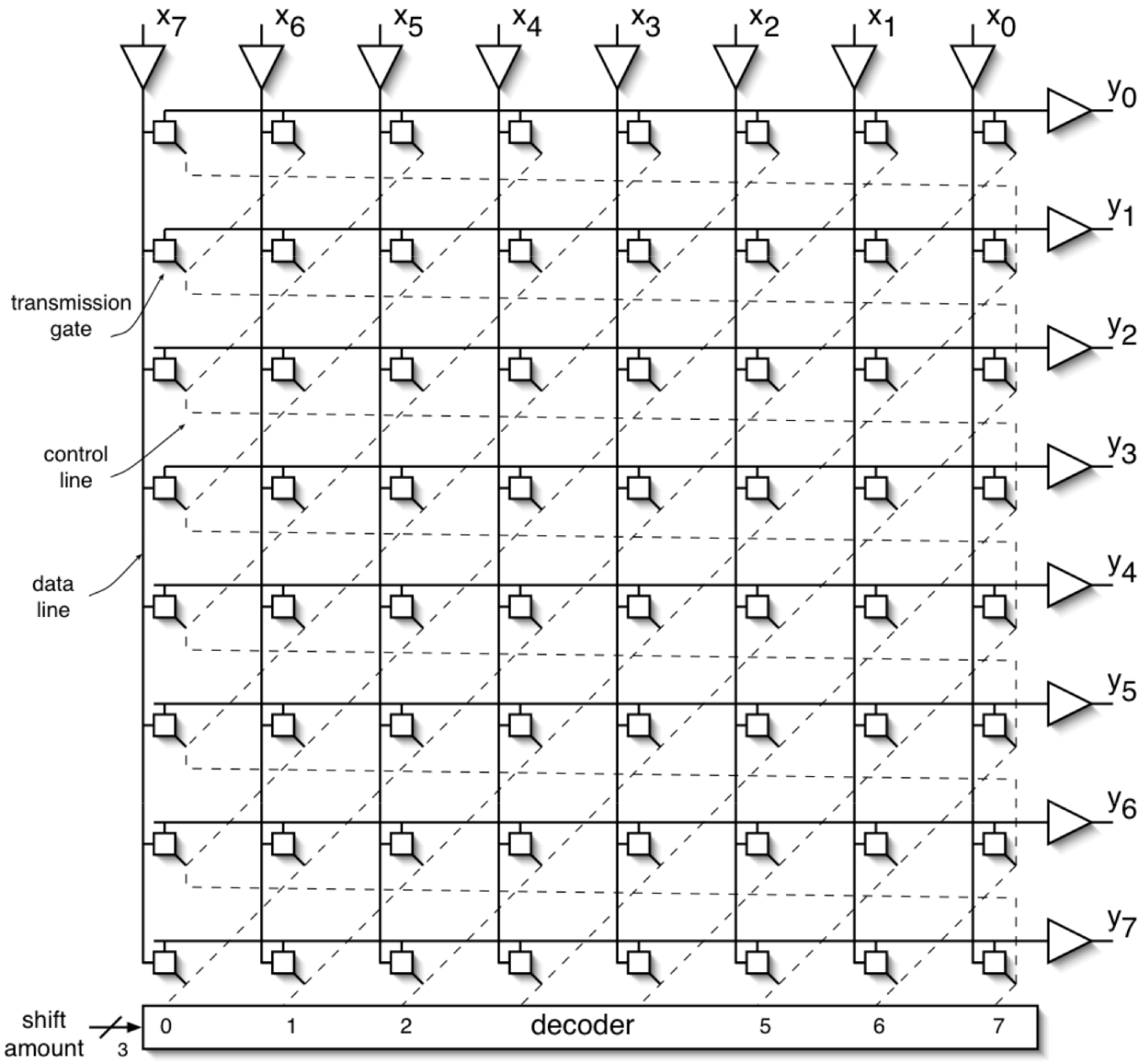
- Can we further simplify the multiplier circuits?

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

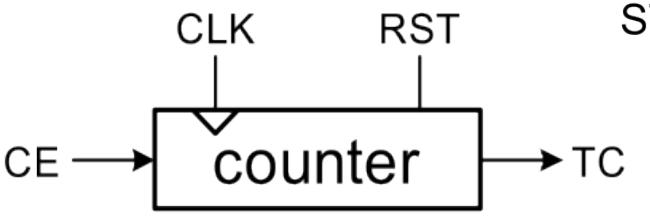


Cost/delay?

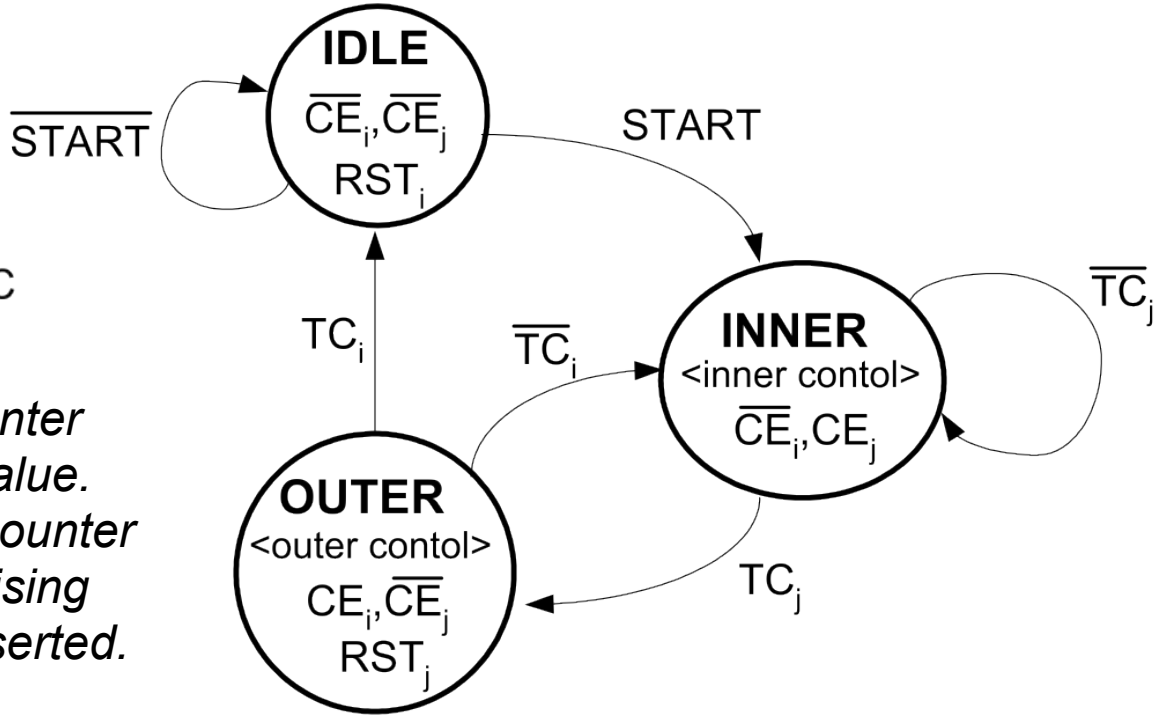
- (don't forget the decoder)

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.

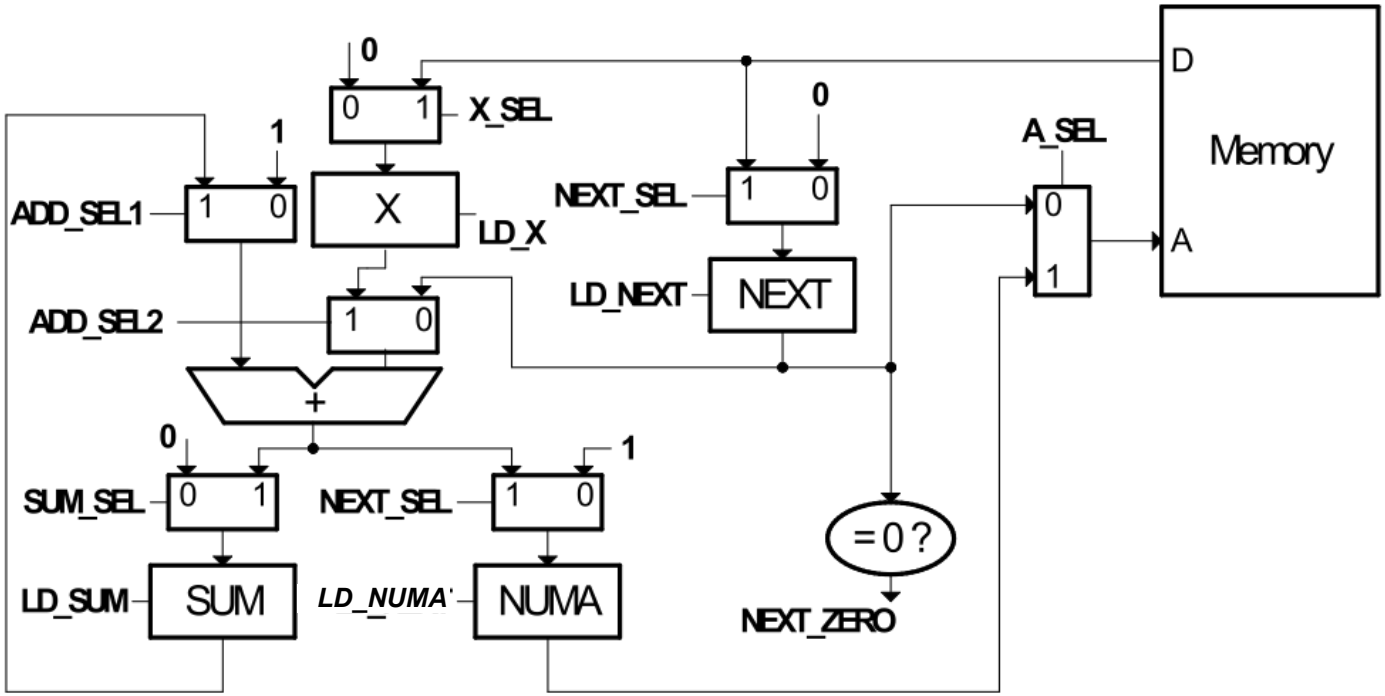


***TC** is asserted when the counter reaches it maximum count value. **CE** is “count enable”. The counter increments its value on the rising edge of the clock if CE is asserted.*



5. Optimization, Architecture #4

- Datapath:

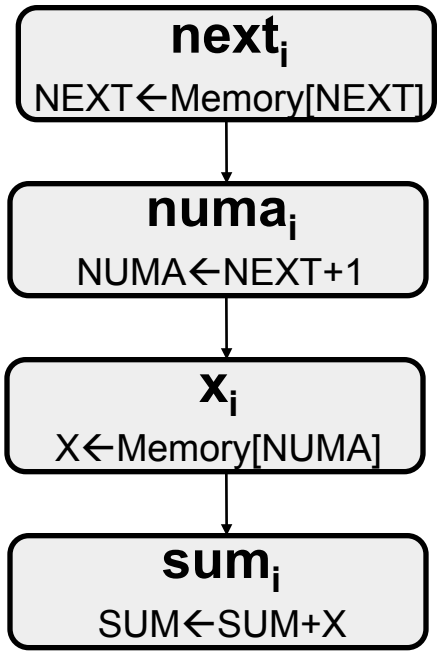


- Incremental cost:
  - Addition of another register & mux, adder mux, and control.
- Performance: find max time of the four actions

1. X←Memory[NUMA], NUMA←NEXT+1; 0.5+1+10+1+0.5 = 13ns
2. NEXT←Memory[NEXT], SUM←SUM+X;

same for all ⇒ T>13ns, F<77MHz

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$_i$

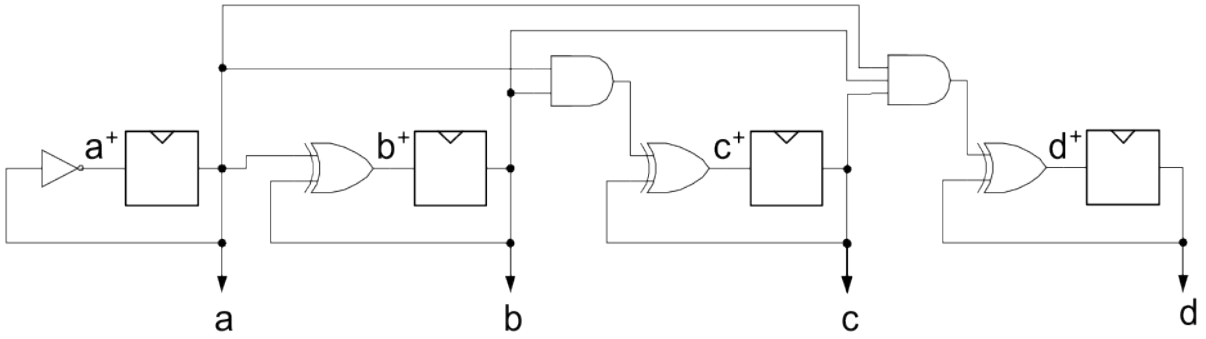
wrap-around, decrease subscript

- Finished schedule for 4 iterations:

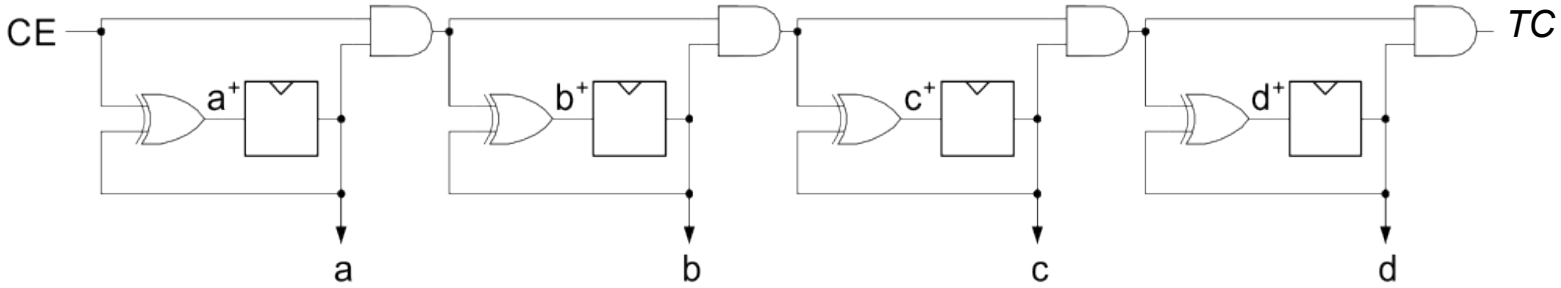
Memory	next$_1$		next$_2$	x$_1$	next$_3$	x$_2$	next$_4$	x$_3$	
adder		numa$_1$		numa$_2$	sum$_1$	numa$_3$	sum$_2$	numa$_4$	sum$_3$

Synchronous Counters

- How do we extend to n-bits?
- Extrapolate c^+: $d^+ = d \oplus abc$, $e^+ = e \oplus abcd$



- Has difficulty scaling (AND gate inputs grow with n)



- CE is “count enable”, allows external control of counting,
- TC is “terminal count”, is asserted on highest value, allows cascading, external sensing of occurrence of max value.

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

$001 = 1_{10}$
$011 = 3_{10}$
$101 = 5_{10}$
$111 = 7_{10}$
} p_1

$010 = 2_{10}$
$011 = 3_{10}$
$110 = 6_{10}$
$111 = 7_{10}$
} p_2

$100 = 4_{10}$
$101 = 5_{10}$
$110 = 6_{10}$
$111 = 7_{10}$
} p_3

The End.

- Special thanks to our GSIs: Chris and Arya.
- Good luck on the final.
- Thanks for a great semester!