CS61C : Machine Structures

Lecture 4.1.2

State and FSMs

2004-07-13

Kurt Meinz

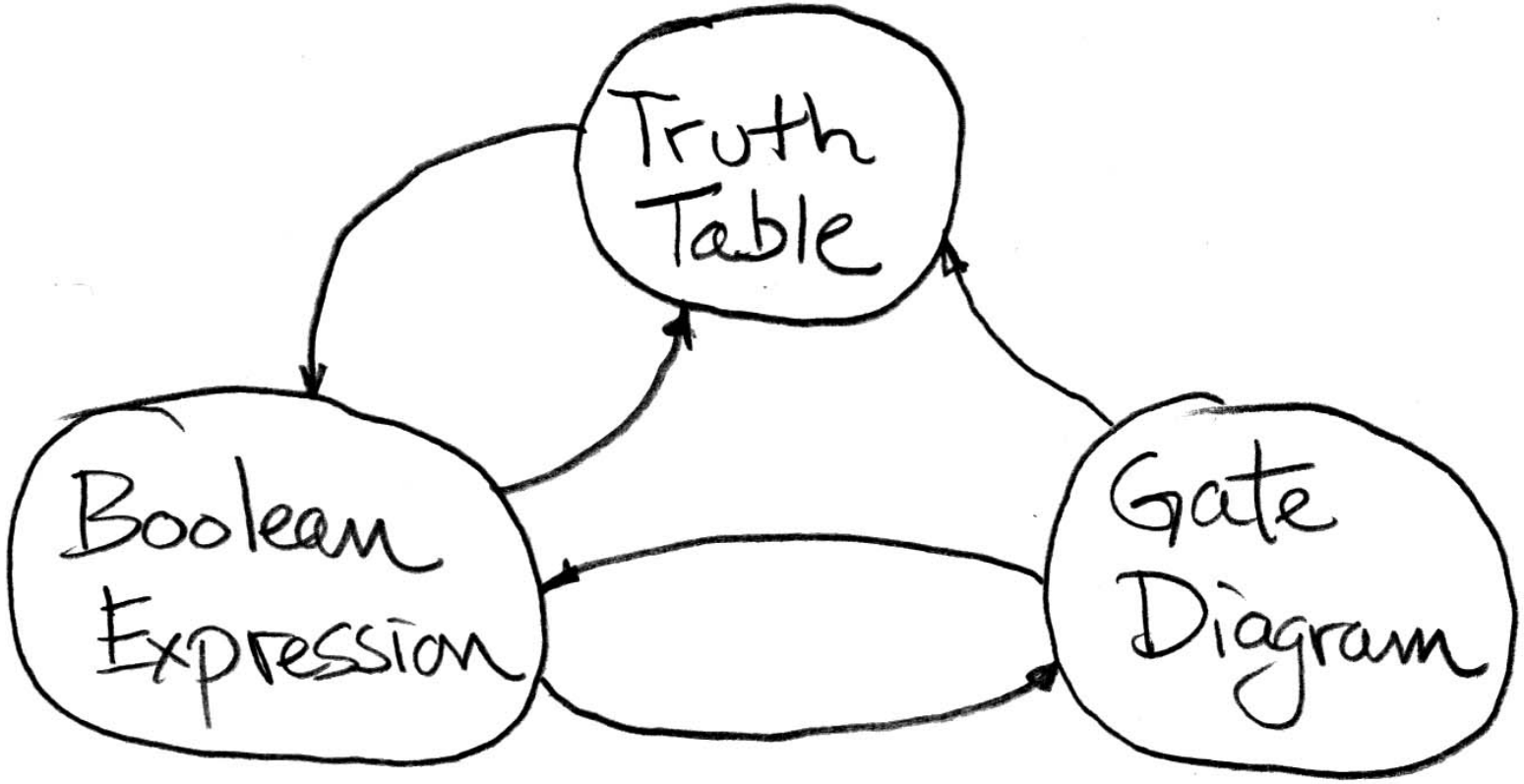
`inst.eecs.berkeley.edu/~cs61c`

Outline

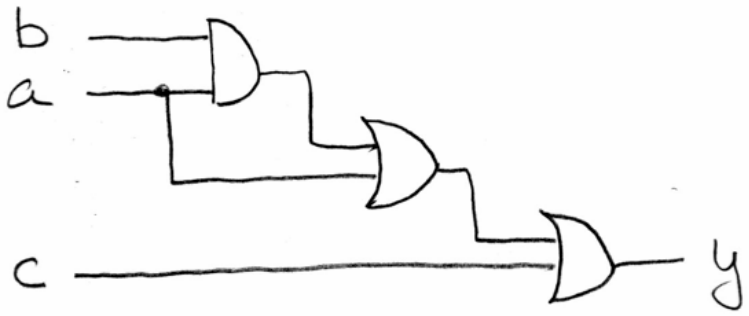
- CL Blocks
- Waveforms
- State
- Clocks
- FSMs

Review (1/3)

- **Use this table and techniques we learned to transform from 1 to another**



(2/3): Circuit & Algebraic Simplification



original circuit

$$\downarrow$$

$$y = ((ab) + a) + c$$

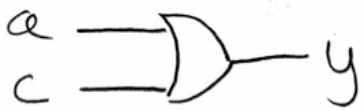
equation derived from original circuit

$$\downarrow$$

$$\begin{aligned} &= ab + a + c \\ &= a(b + 1) + c \\ &= a(1) + c \\ &= a + c \end{aligned}$$

algebraic simplification

$$\downarrow$$



simplified circuit

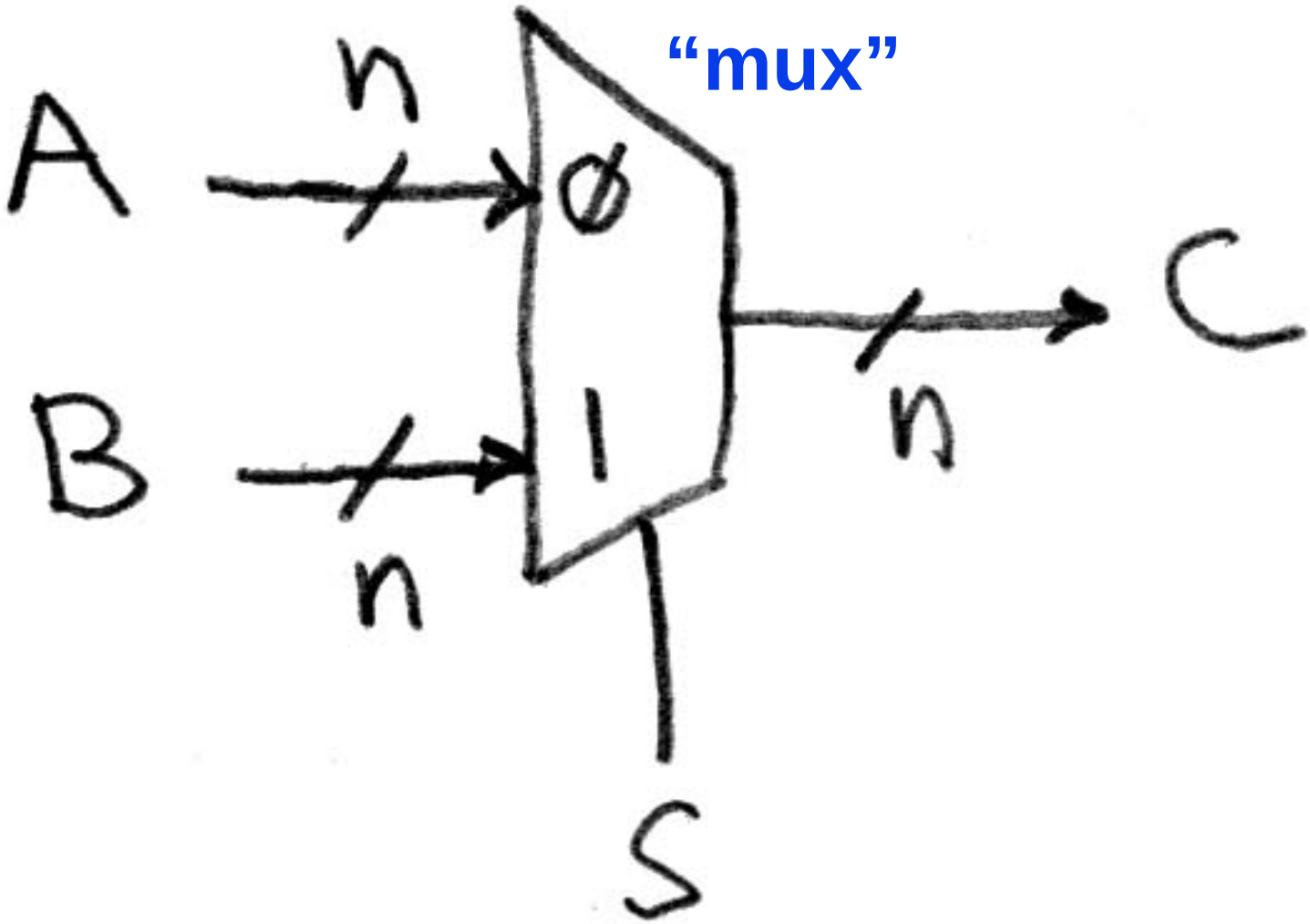
(3/3):Laws of Boolean Algebra

$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$	complementarity
$x \cdot 0 = 0$	$x + 1 = 1$	laws of 0's and 1's
$x \cdot 1 = x$	$x + 0 = x$	identities
$x \cdot x = x$	$x + x = x$	idempotent law
$x \cdot y = y \cdot x$	$x + y = y + x$	commutativity
$(xy)z = x(yz)$	$(x + y) + z = x + (y + z)$	associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	distribution
$xy + x = x$	$(x + y)x = x$	uniting theorem
$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{(x + y)} = \overline{x} \cdot \overline{y}$	DeMorgan's Law

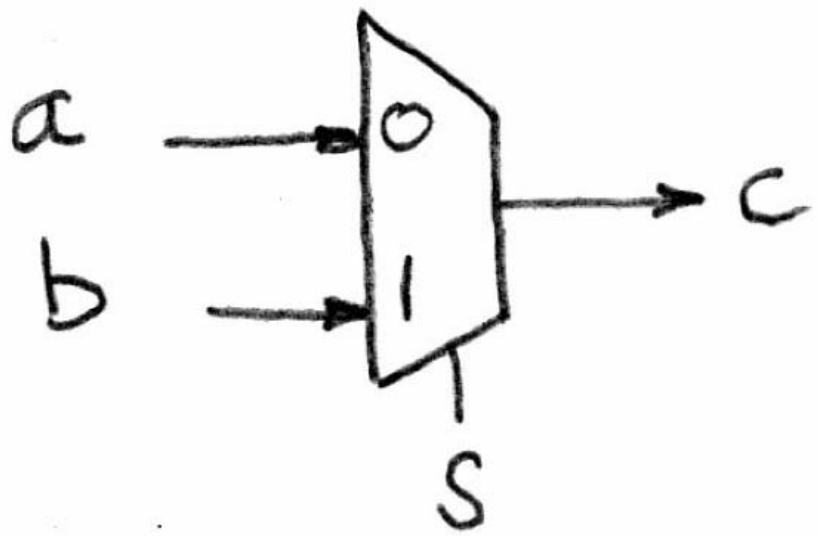
CL Blocks

- Let's use our skills to build some CL blocks:
  - Multiplexer (mux)
  - Adder
  - ALU

Data Multiplexor (here 2-to-1, n-bit-wide)



N instances of 1-bit-wide mux

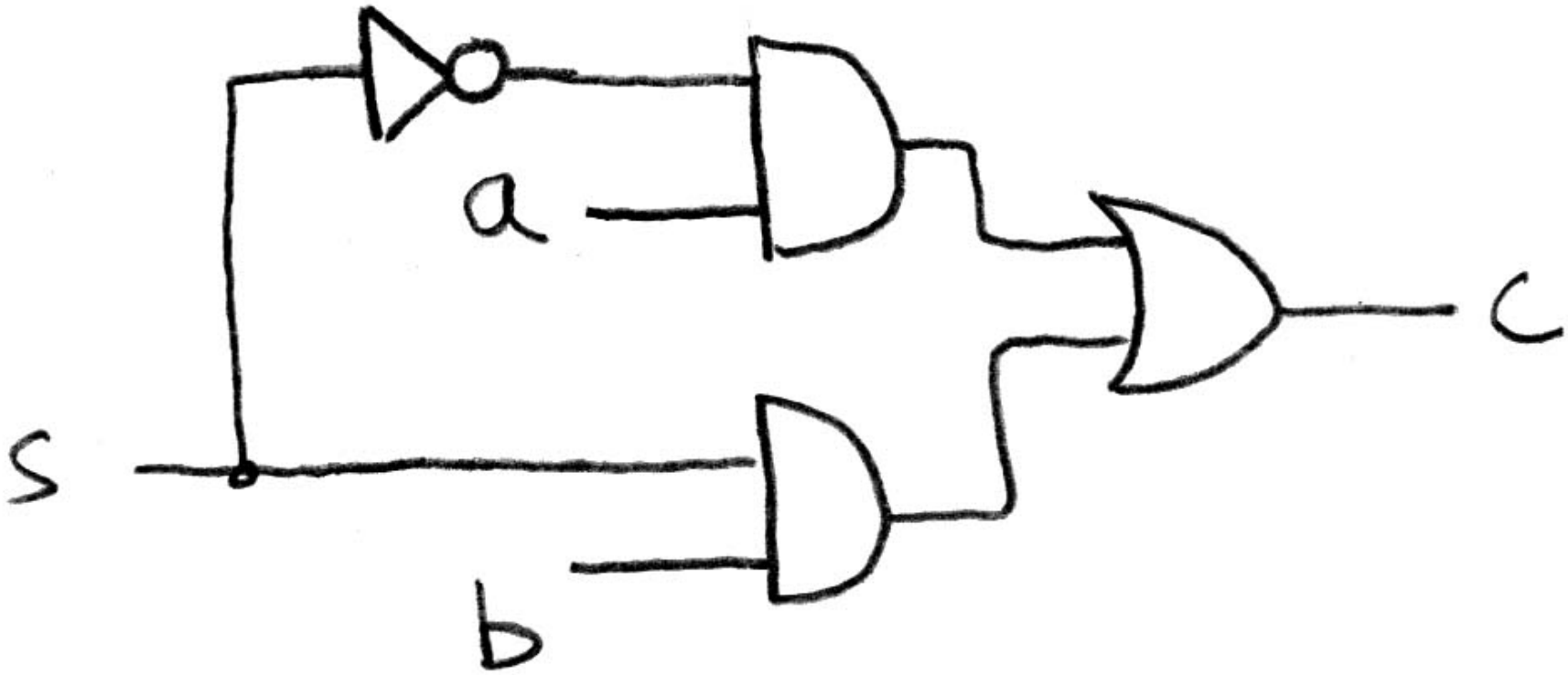


s	ab	c
0	00	0
0	01	0
0	10	1
0	11	1
1	00	0
1	01	1
1	10	0
1	11	1

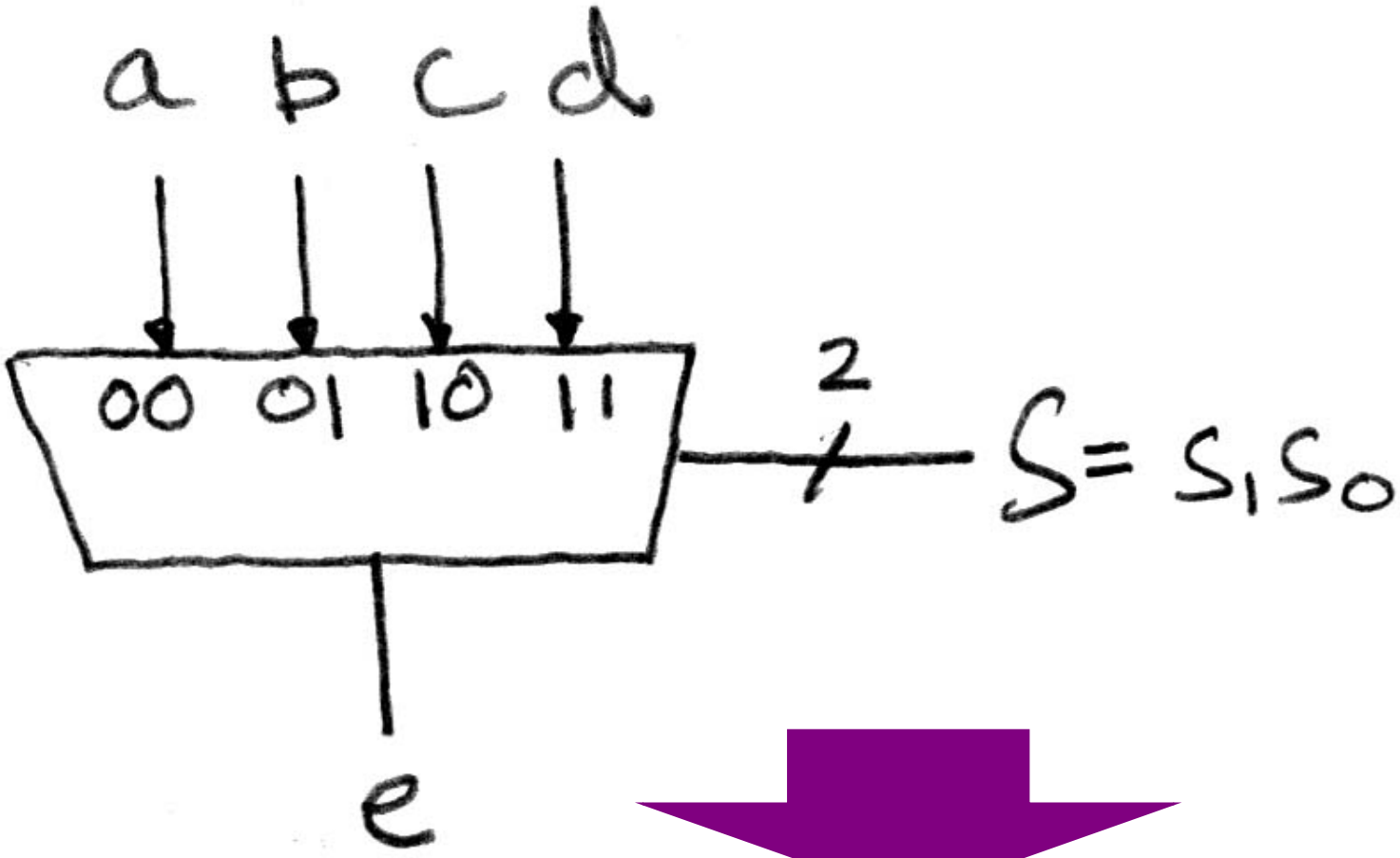
$$
\begin{aligned}
c &= \overline{s}a\overline{b} + \overline{s}ab + s\overline{a}b + sab \\
&= \overline{s}(a\overline{b} + ab) + s(\overline{a}b + ab) \\
&= \overline{s}(a(\overline{b} + b)) + s((\overline{a} + a)b) \\
&= \overline{s}(a(1) + s((1)b) \\
&= \overline{s}a + sb
\end{aligned}
$$

How do we build a 1-bit-wide mux?

$$\overline{s}a + sb$$

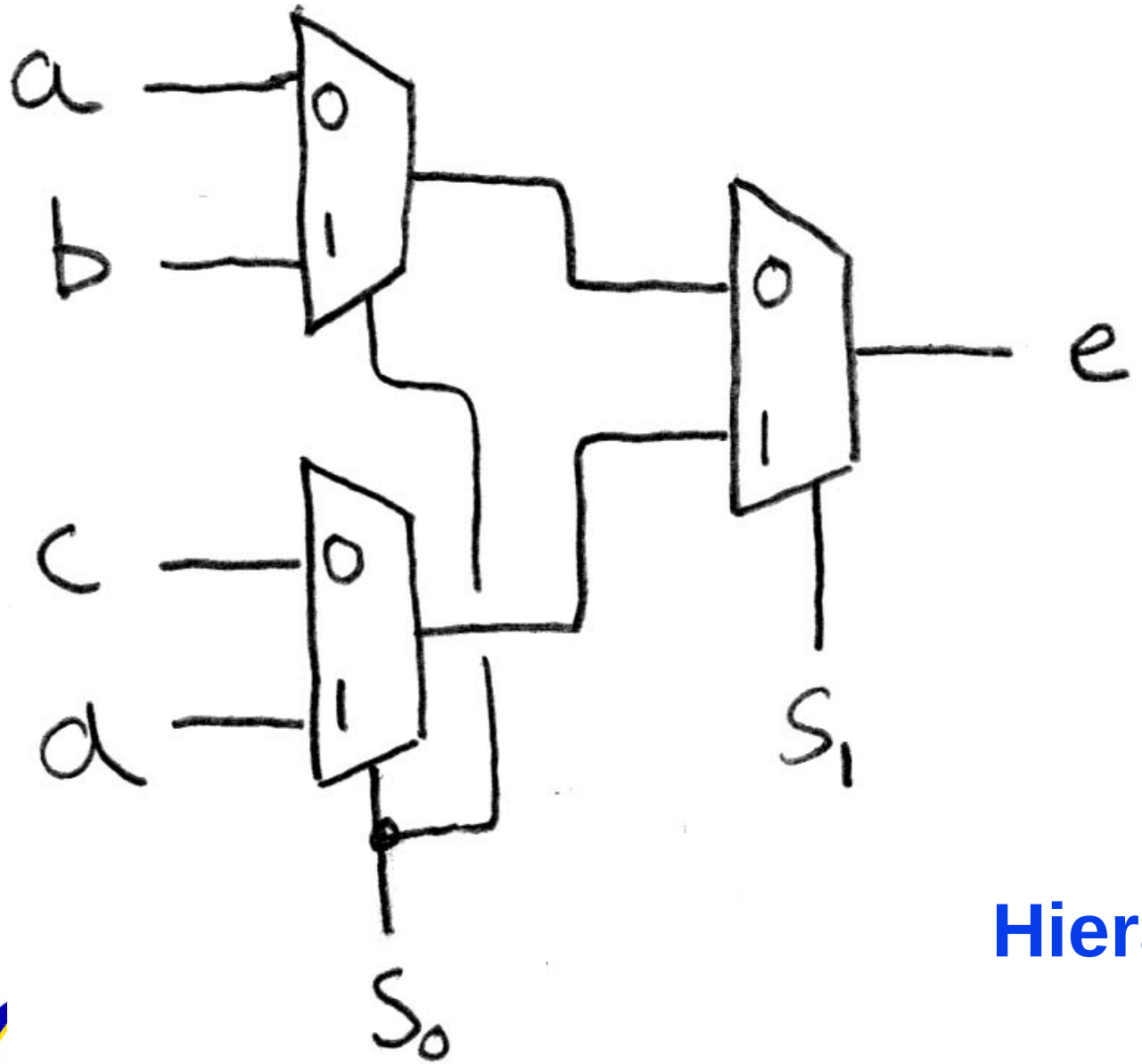


4-to-1 Multiplexor?



$$e = \overline{s_1 s_0} a + \overline{s_1} s_0 b + s_1 \overline{s_0} c + s_1 s_0 d$$

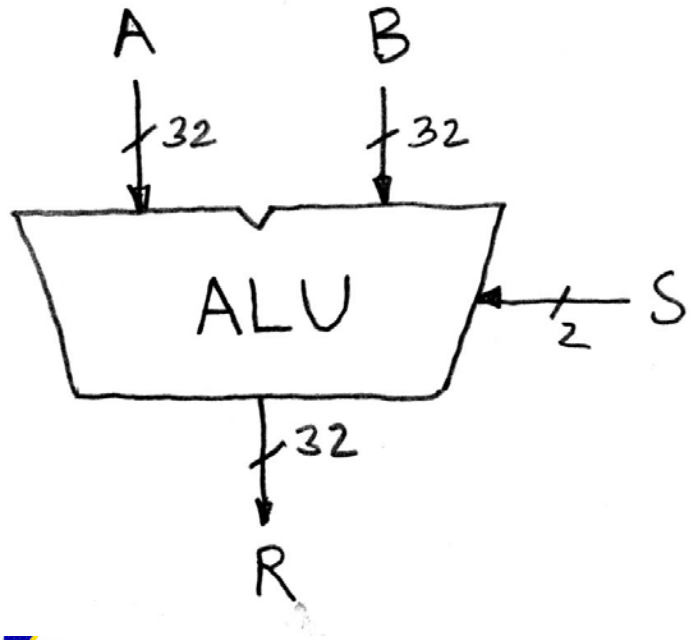
An Alternative Approach



Hierarchically!

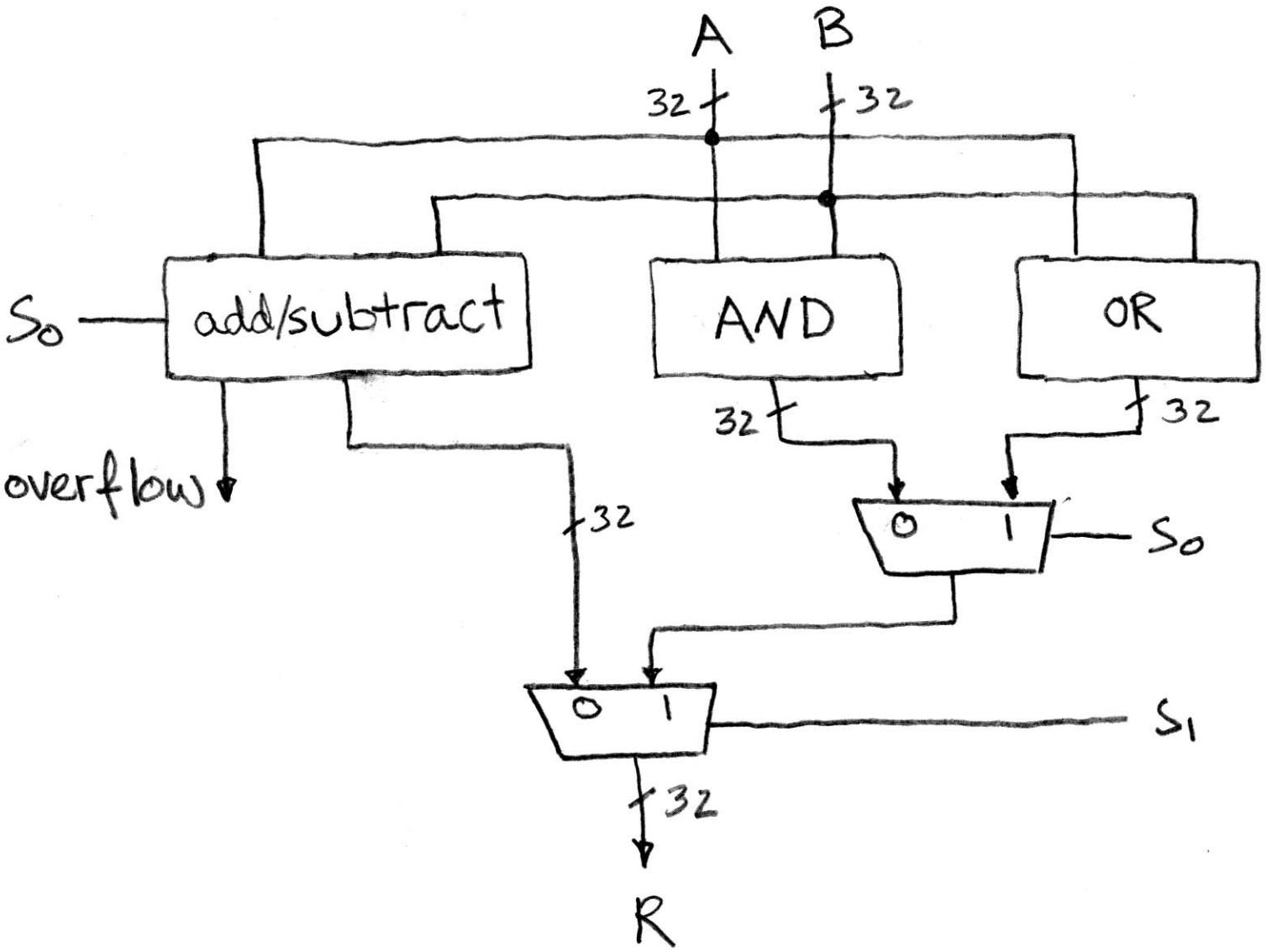
Arithmetic and Logic Unit

- Most processors contain a logic block called “Arithmetic/Logic Unit” (ALU)
- We’ll show you an easy one that does ADD, SUB, bitwise AND, bitwise OR



when S=00, R=A+B
when S=01, R=A-B
when S=10, R=A AND B
when S=11, R=A OR B

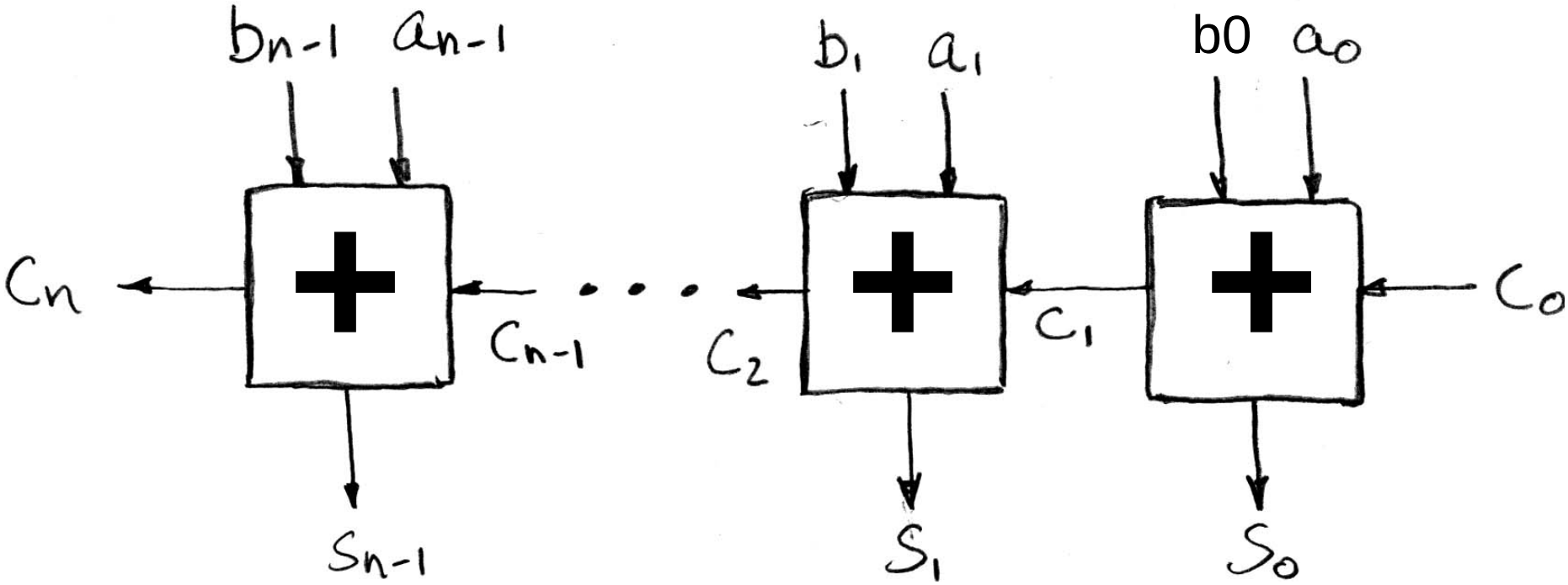
Our simple ALU



Adder/Subtracter Design -- how?

- Truth-table, then determine canonical form, then minimize and implement as we've seen before
- Look at breaking the problem down into smaller pieces that we can cascade or hierarchically layer

N 1-bit adders ⇒ 1 N-bit adder



Adder/Subtracter – One-bit adder LSB…

	a_3	a_2	a_1	a_0
+	b_3	b_2	b_1	b_0
	s_3	s_2	s_1	s_0

a_0	b_0	s_0	c_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$s_0 = a_0 \text{ XOR } b_0$$
$$c_1 = a_0 \text{ AND } b_0$$

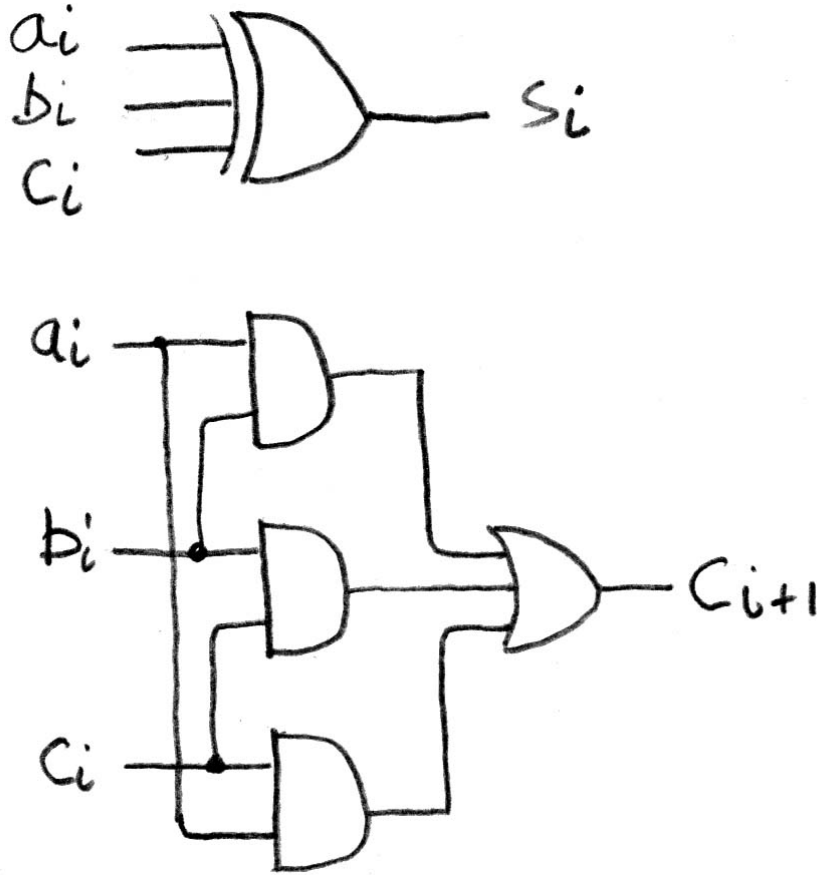
Adder/Subtracter – One-bit adder (1/2)…

$$
\begin{array}{cccc|c|c}
 & a_3 & a_2 & a_1 & a_0 \\
+ & b_3 & b_2 & b_1 & b_0 \\
\hline
 & s_3 & s_2 & s_1 & s_0
\end{array}
$$

a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

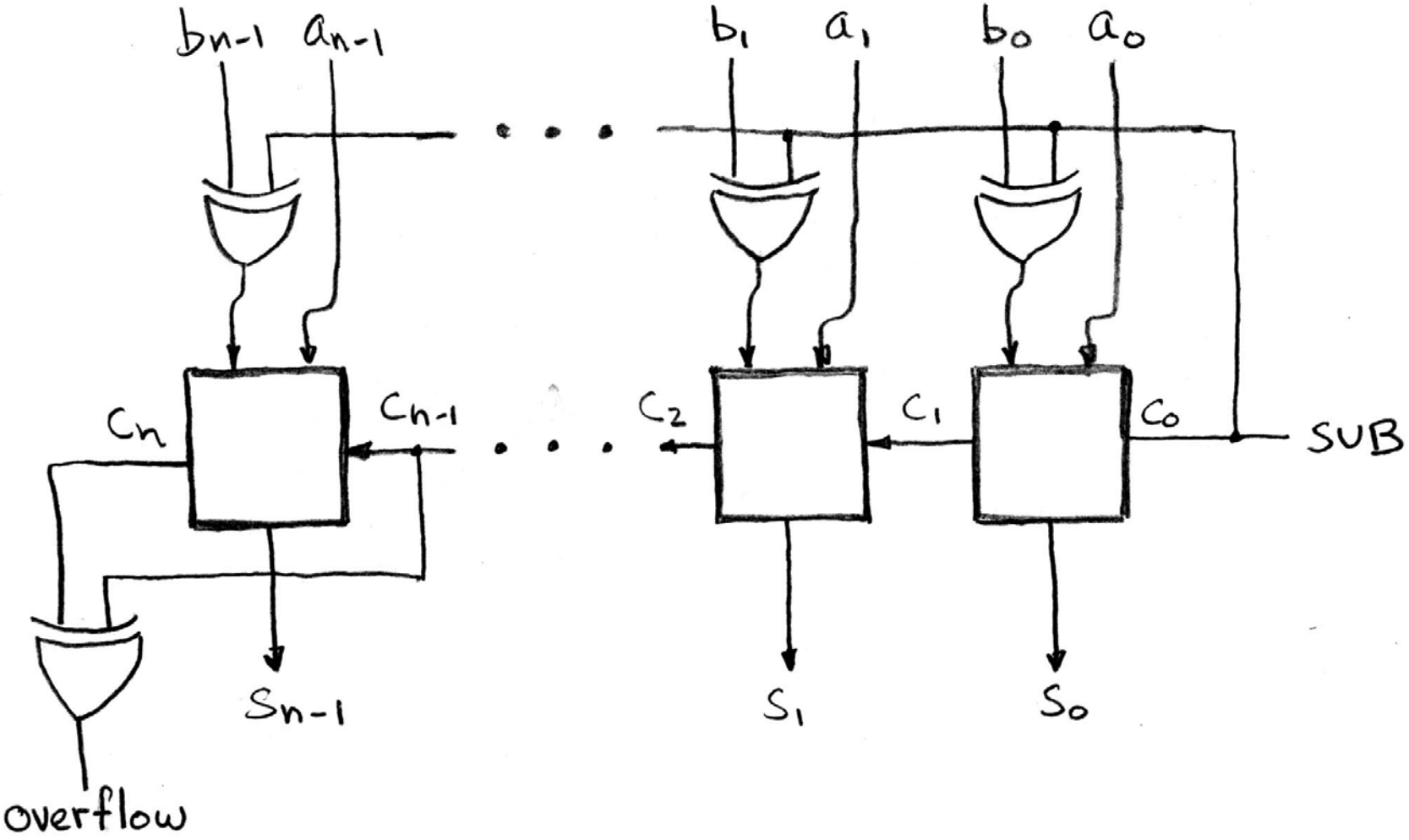
$$
\begin{aligned}
s_i &= \mathrm{XOR}(a_i, b_i, c_i) \\
c_{i+1} &= \mathrm{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i
\end{aligned}
$$

Adder/Subtracter – One-bit adder (2/2)…



$$s_i = \mathrm{XOR}(a_i, b_i, c_i)$$
$$c_{i+1} = \mathrm{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i$$

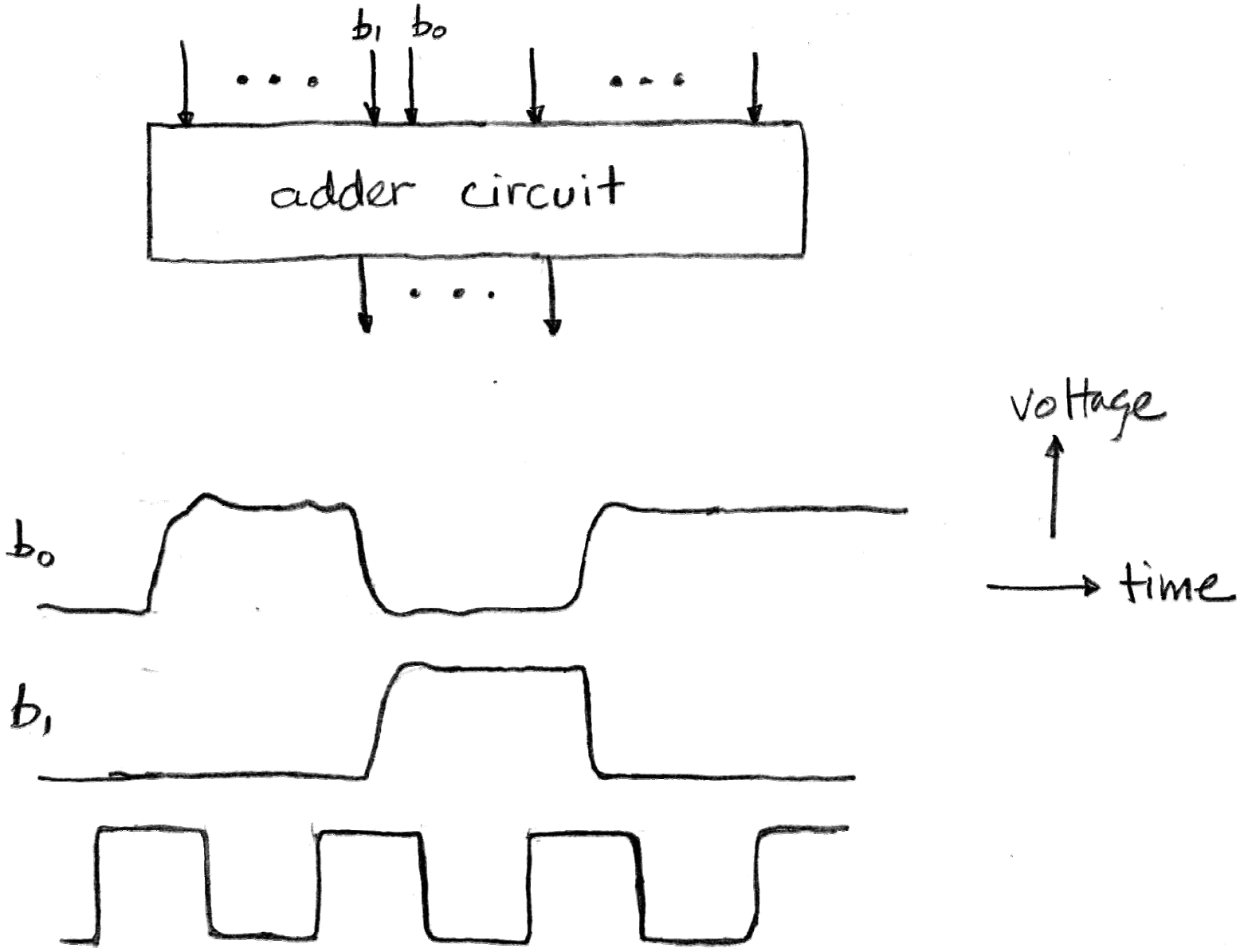
Extremely Clever Subtractor



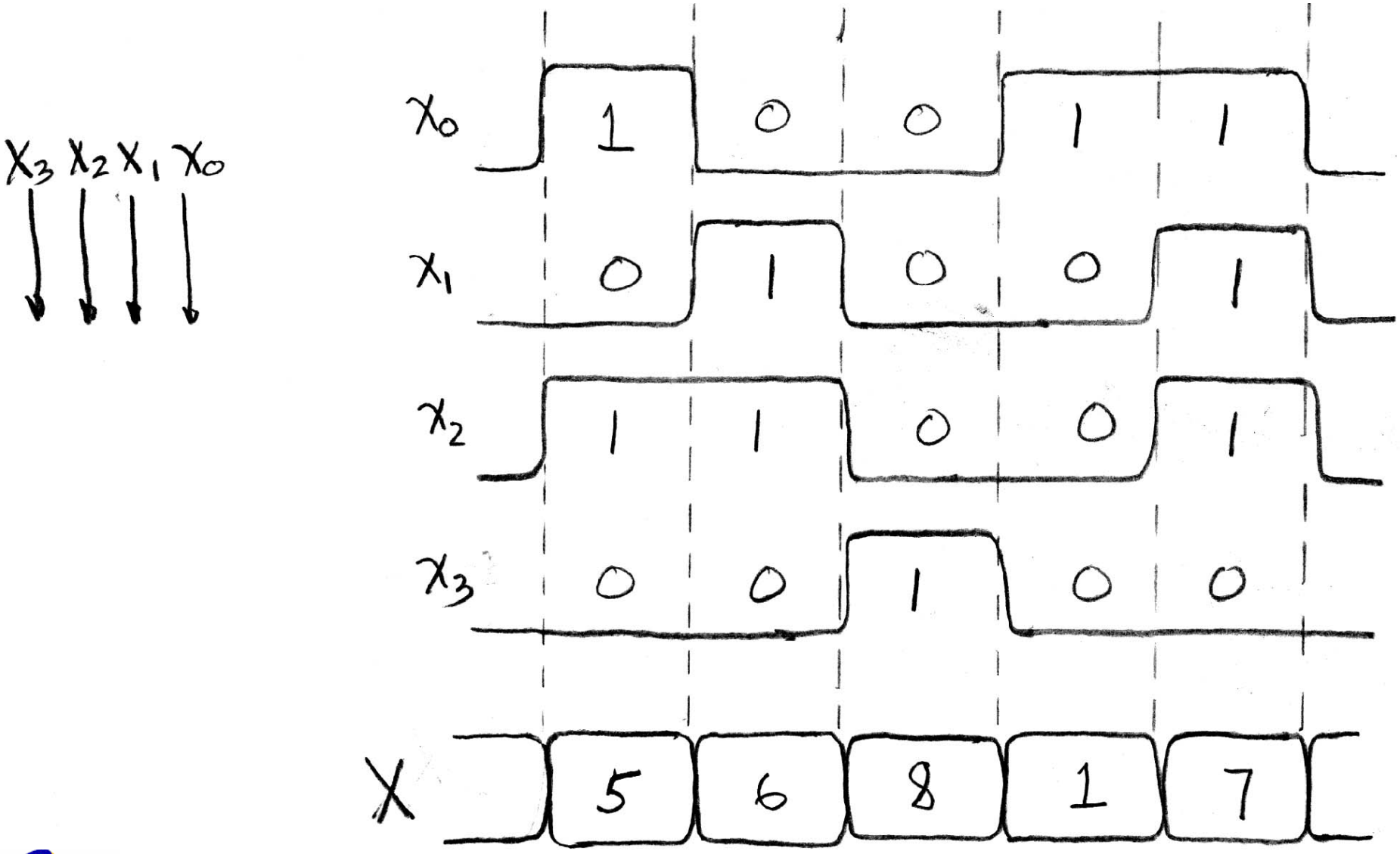
Signals and Waveforms (1/4)

- Outputs of CL change over time
  - With what? → Change in inputs
  - Can graph changes with waveforms …

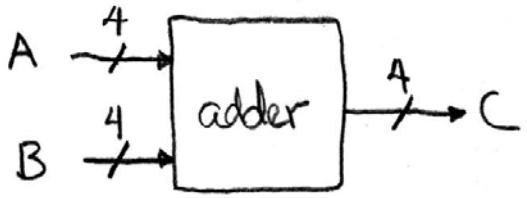
Signals and Waveforms (2/4): Adders



Signals and Waveforms (3/4): Grouping

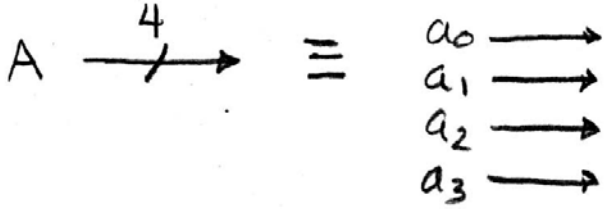


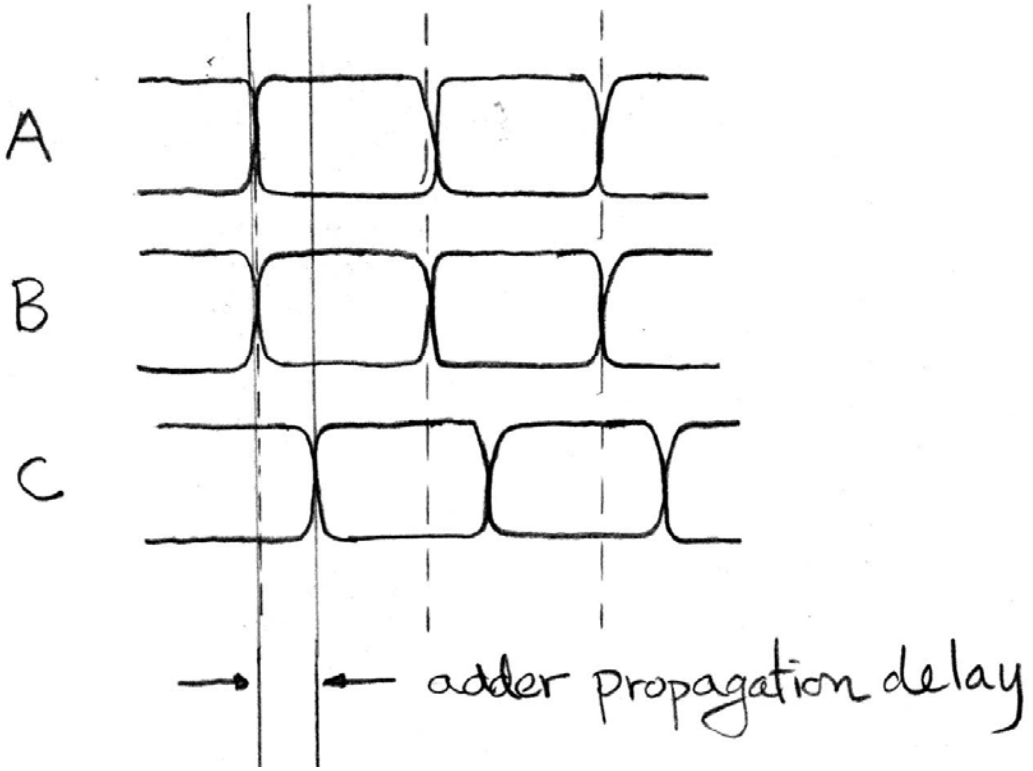
Signals and Waveforms (4/4): Circuit Delay



$A = [a_3, a_2, a_1, a_0]$

$B = [b_3, b_2, b_1, b_0]$

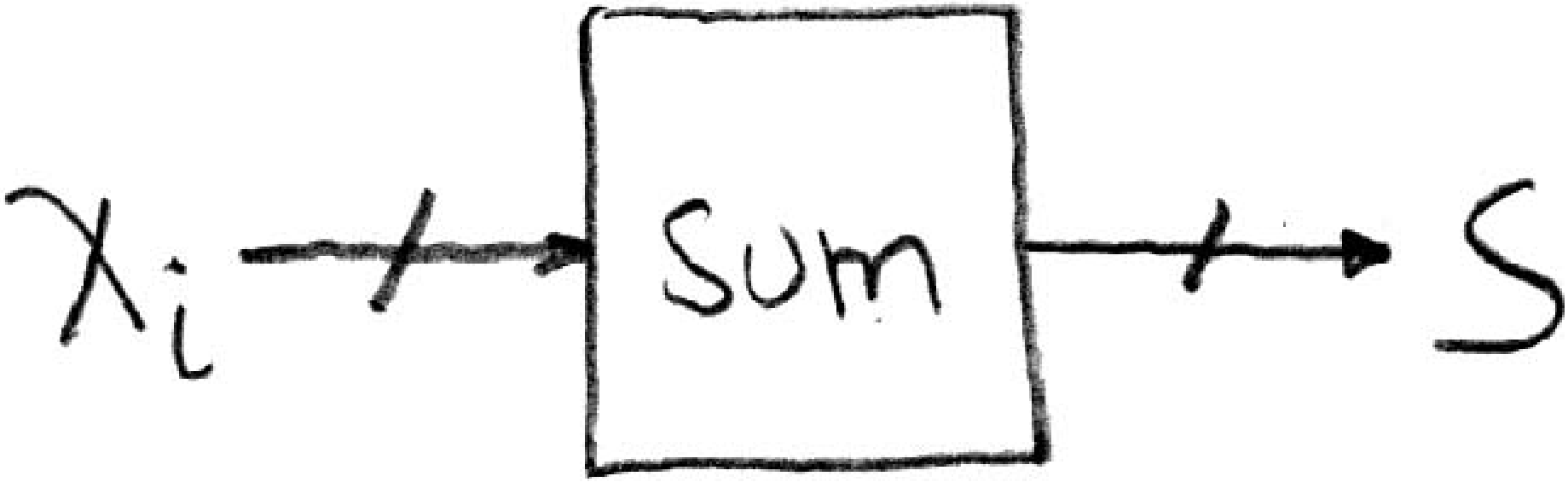




State

- With CL, output is always a function of CURRENT input
  - With some (variable) propagation delay
- Clearly, we need a way to introduce state into computation

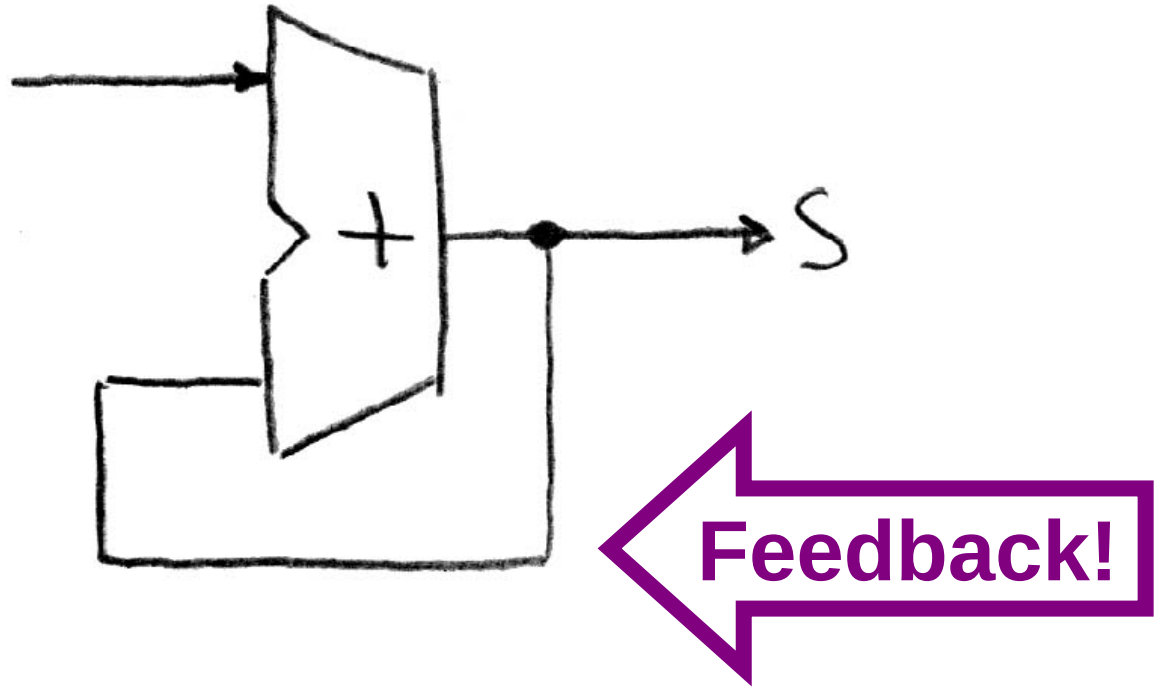
Accumulator Example



Want:

```
S=0; for i from 0 to n-1
S = S + X_i
```

First try…Does this work?



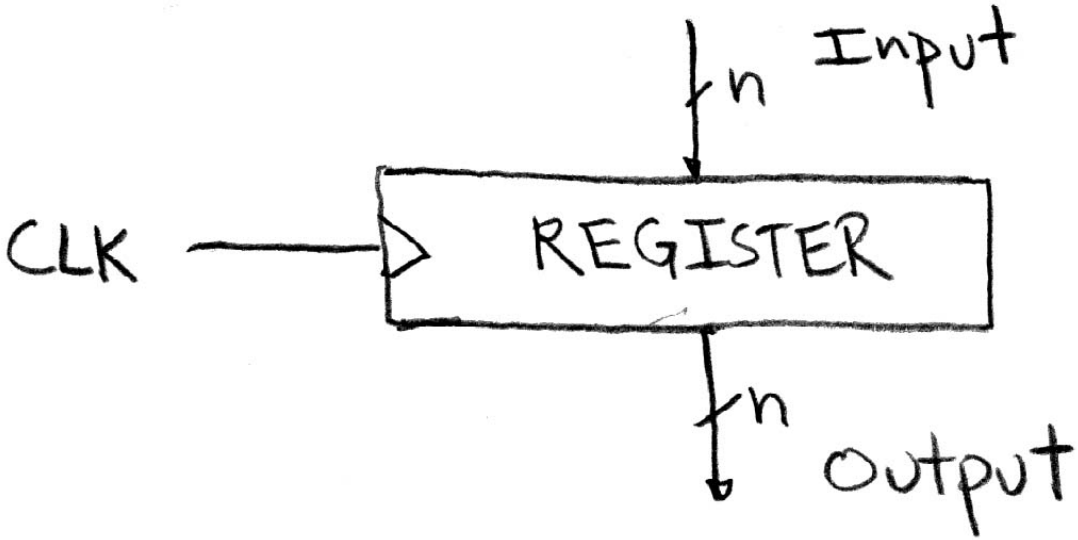
Nope!

Reason #1… What is there to control the next iteration of the ‘`for`’ loop?

Reason #2… How do we say: ‘`S=0`’?

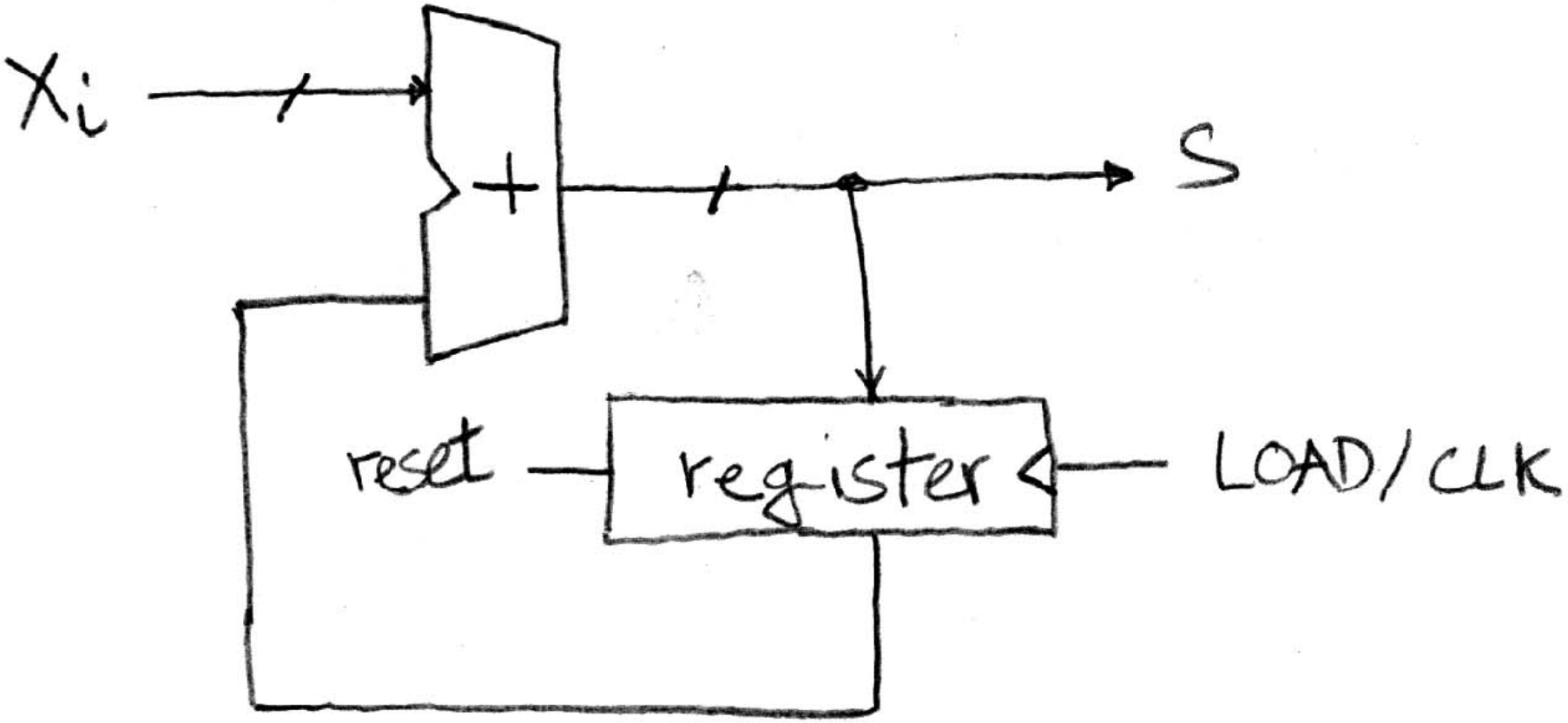
Need a way to store partial sums! …

Circuits with STATE (e.g., register)

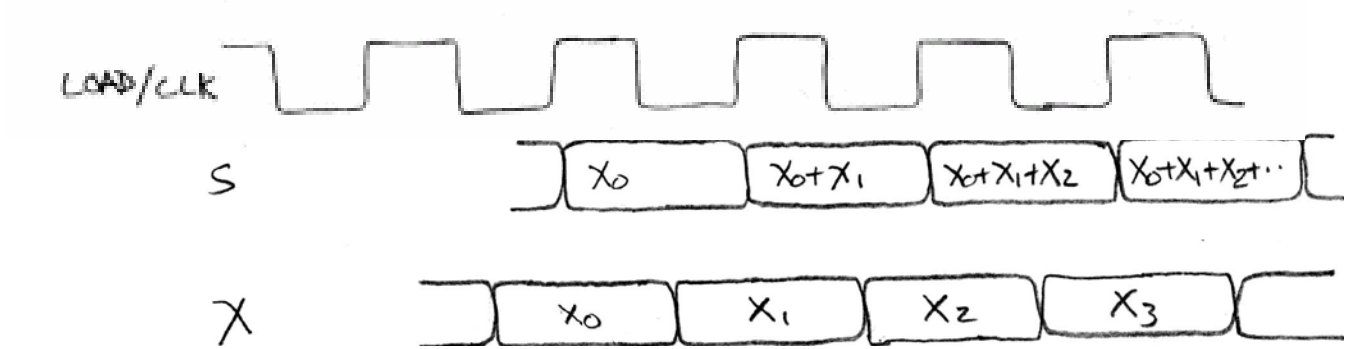


Need a Logic Block that will:
1. store output (partial sum) for a while,
2. until we tell it to update with a new value.

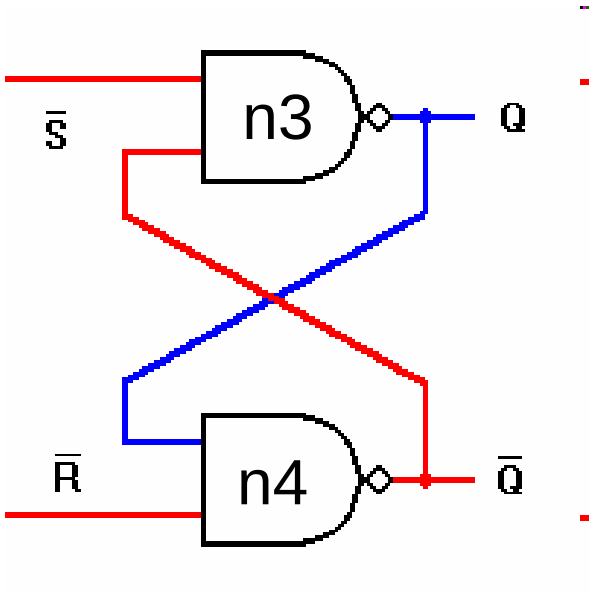
Second try…How about this? Yep!



Rough timing…



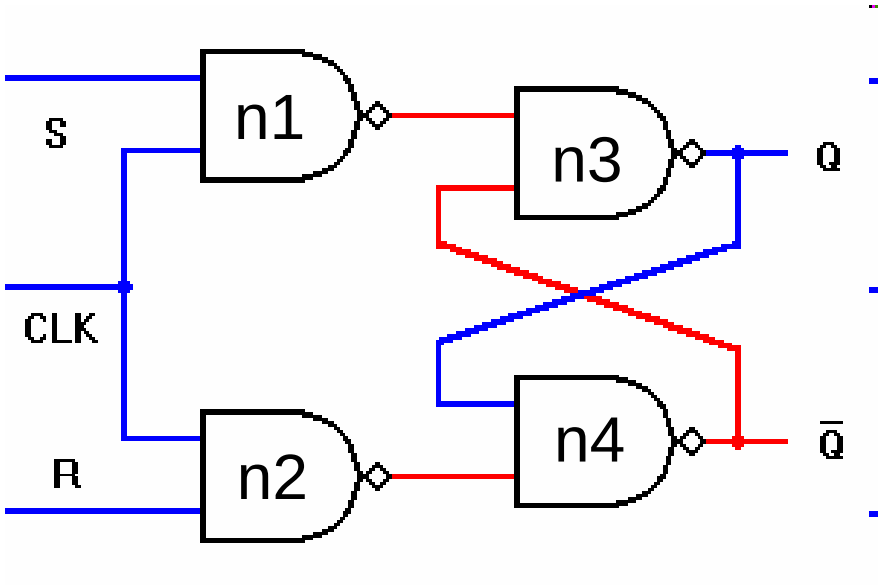
State



A	B	O
0	0	1
0	1	1
1	0	1
1	1	1

- S=1, R=0 → Sbar=0, Rbar=1 → Q=1 → Qbar = 0
- S=0, R=1 → Sbar=1, Rbar=0 → Qbar=1 → Q = 0
- S=0, R=0 → Sbar=1, Rbar=1 → Qbar <= not Q
  Q <= not Qbar

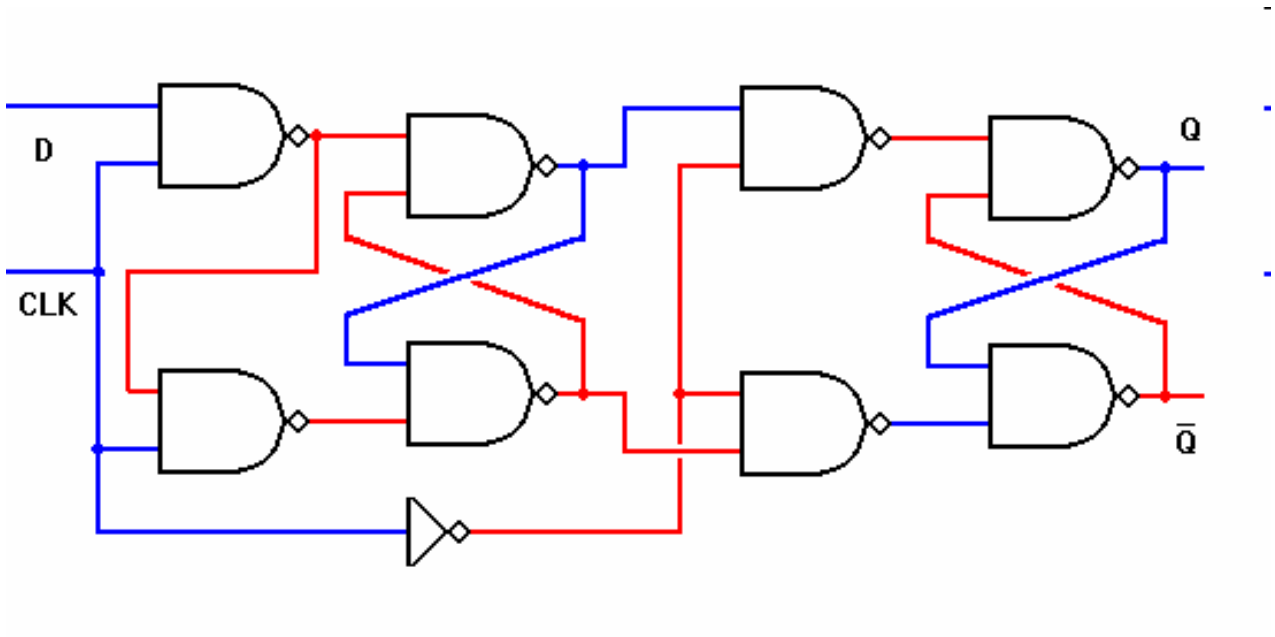
State



A	B	O
0	0	1
0	1	1
1	0	1
1	1	1

- **When CLK is low:**
  - **n1 and n2 = 1 ➔ no change**
- **When CLK is high:**
  - **S, R = 0 ➔ n1, n2 = 1 ➔ no change**
  - **S=1, R=0 ➔ n1=0, n1=1 ➔ Q = 1, Qbar = 0**

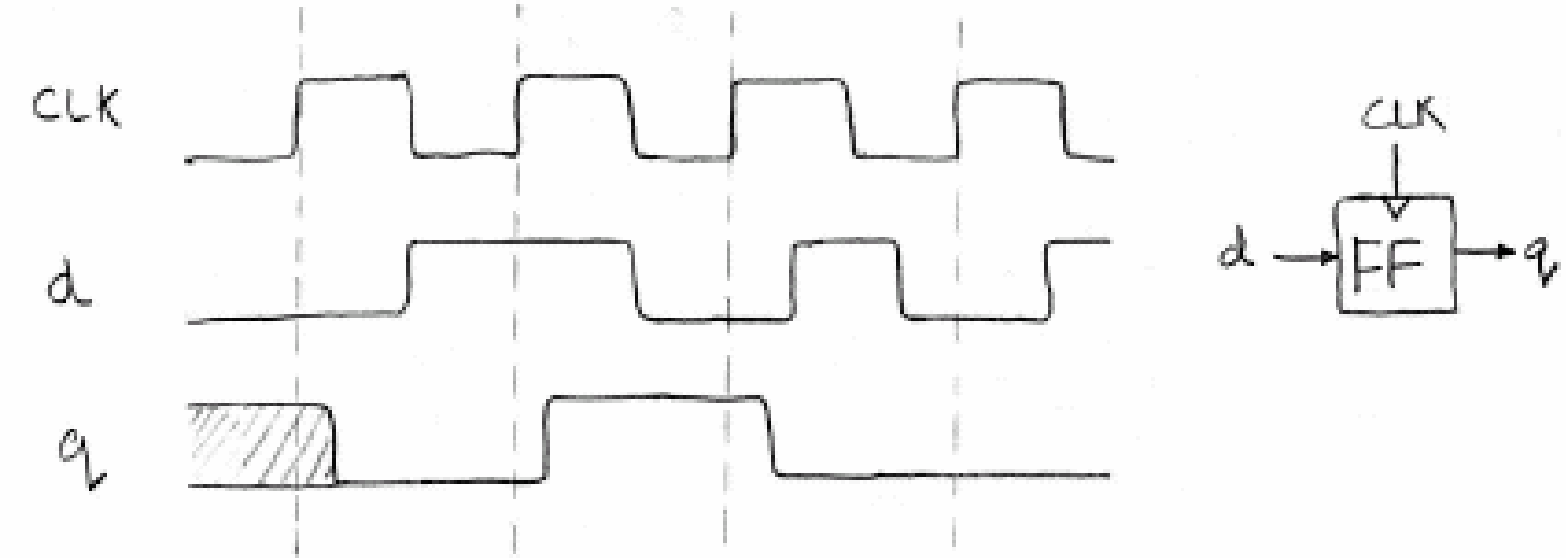
State



A	B	O
0	0	1
0	1	1
1	0	1
1	1	1

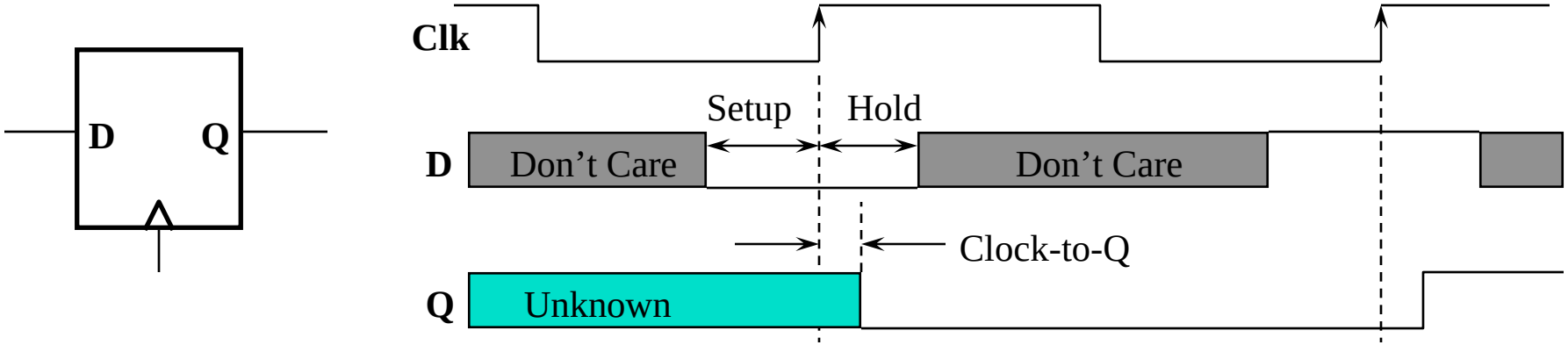
- **This is a "rising-edge D Flip-Flop"**
  - **When the CLK transitions from 0 to 1 (rising edge) …**
    - **Q ← D; Qbar ← not D**
  - **All other times: Q ← Q; Qbar ← Qbar**

D Flip Flop



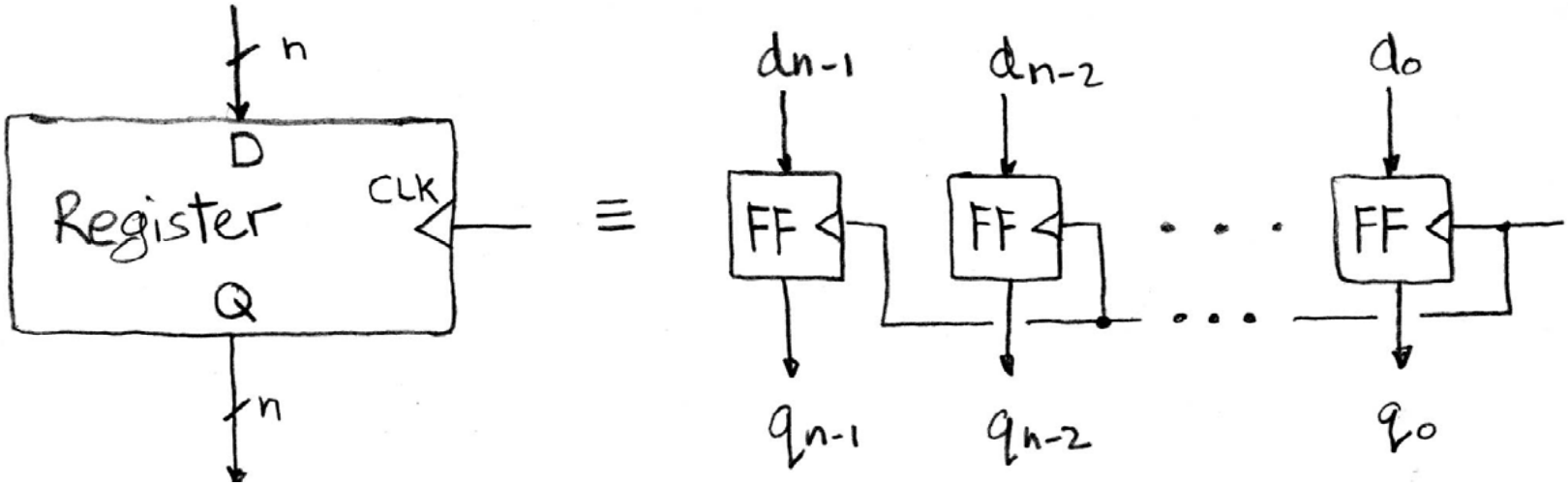
- Called “edge-sensitive”
- RS latches: “level sensitive”

Storage Element's Timing Model



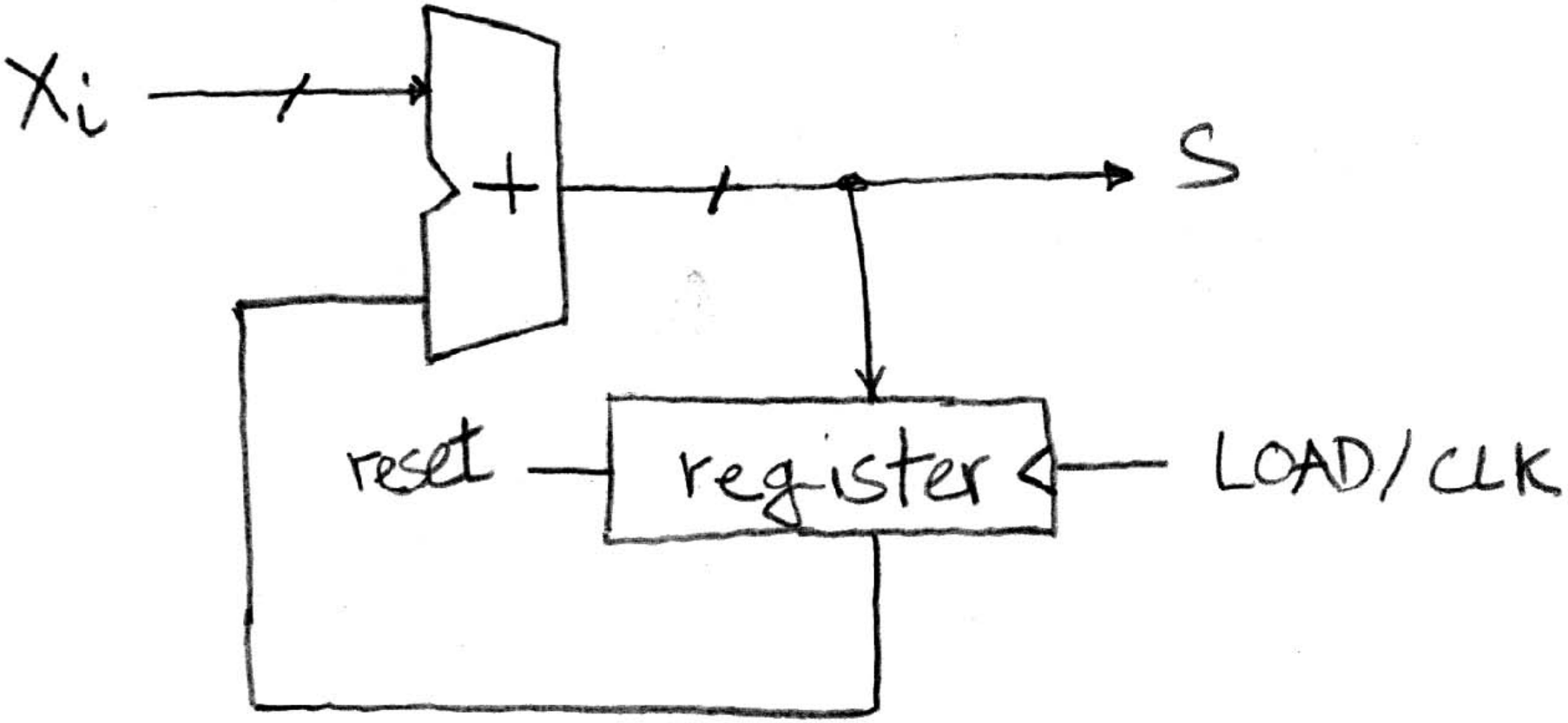
- **Setup Time: Input must be stable BEFORE trigger clock edge**
- **Hold Time: Input must REMAIN stable after trigger clock edge**
- **Clock-to-Q time:**
  - **Output cannot change instantaneously at the trigger clock edge**
  - **Similar to delay in logic gates**

Bus a bunch of D FFs together …

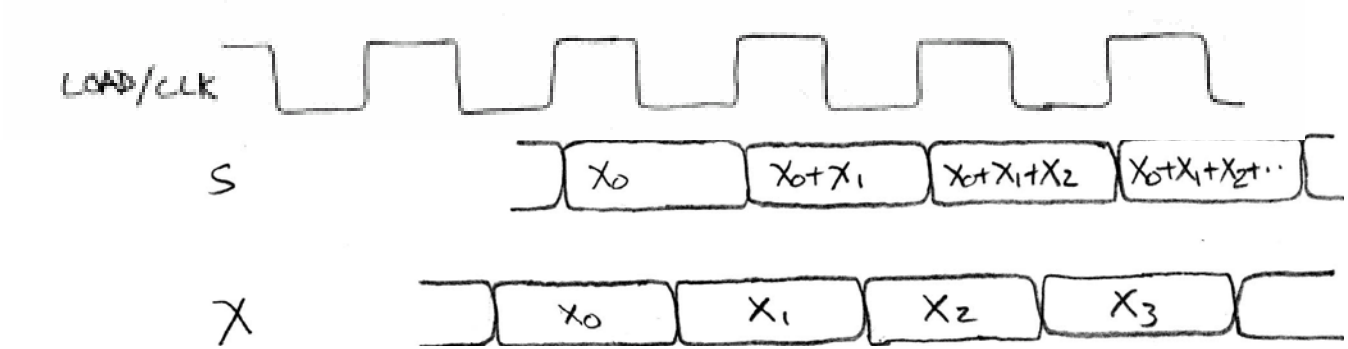


- **Register of size N:**
  - **n instances of D Flip-Flop**

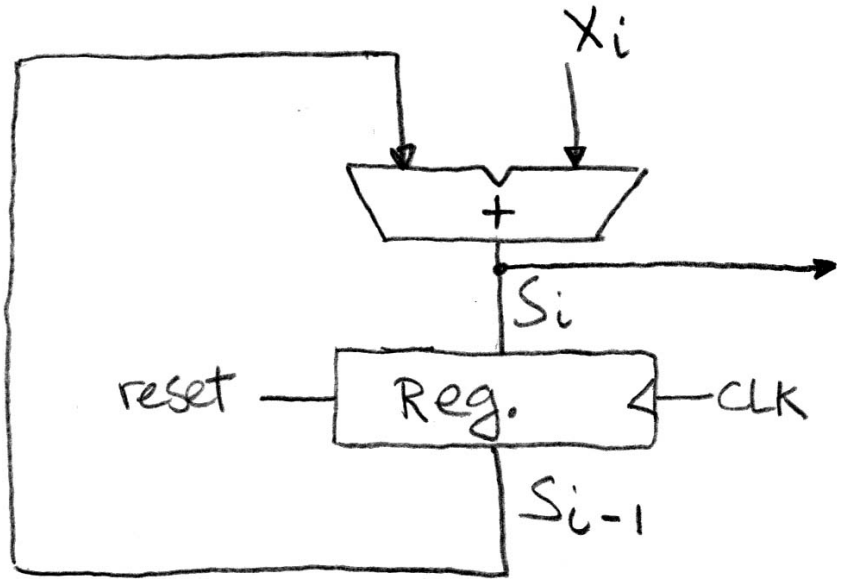
Second try…How about this? Yep!

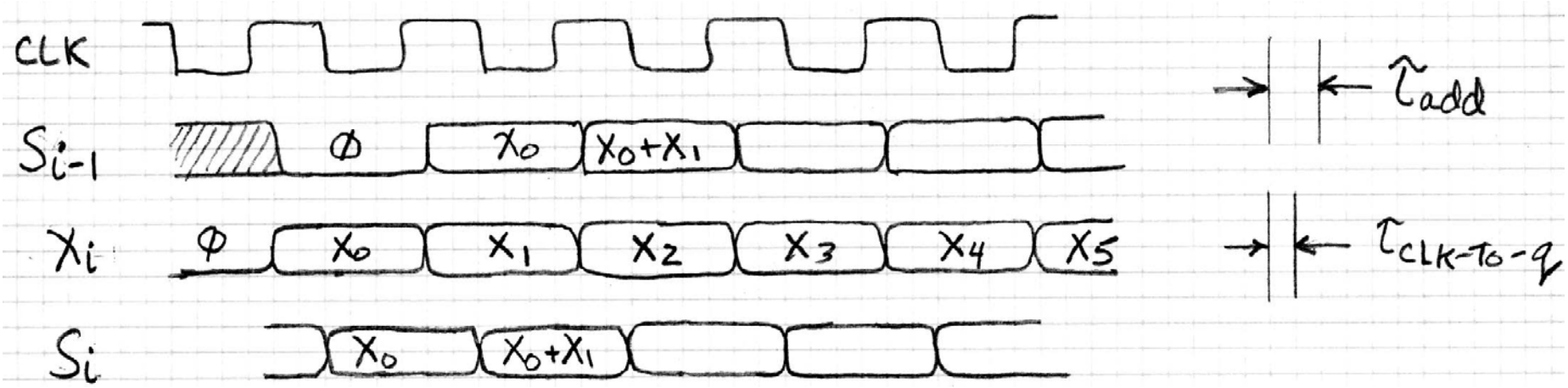


Rough timing…

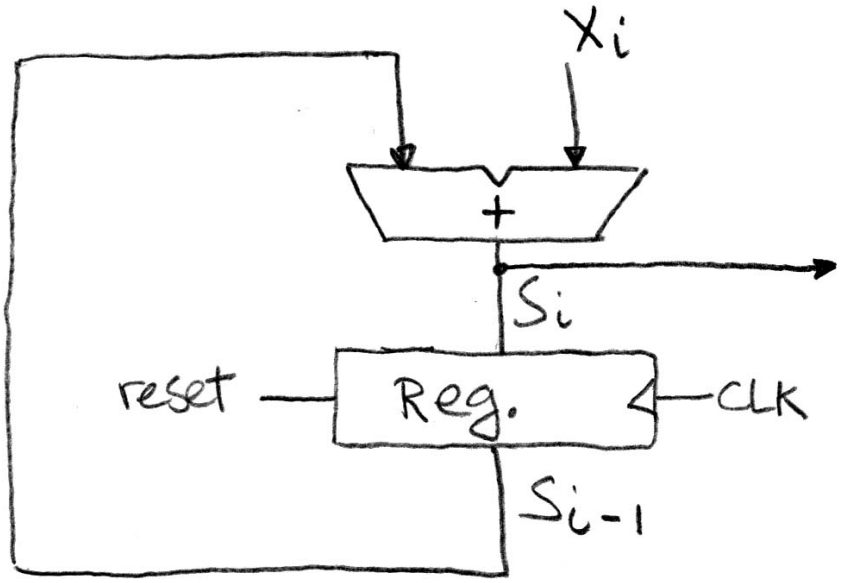


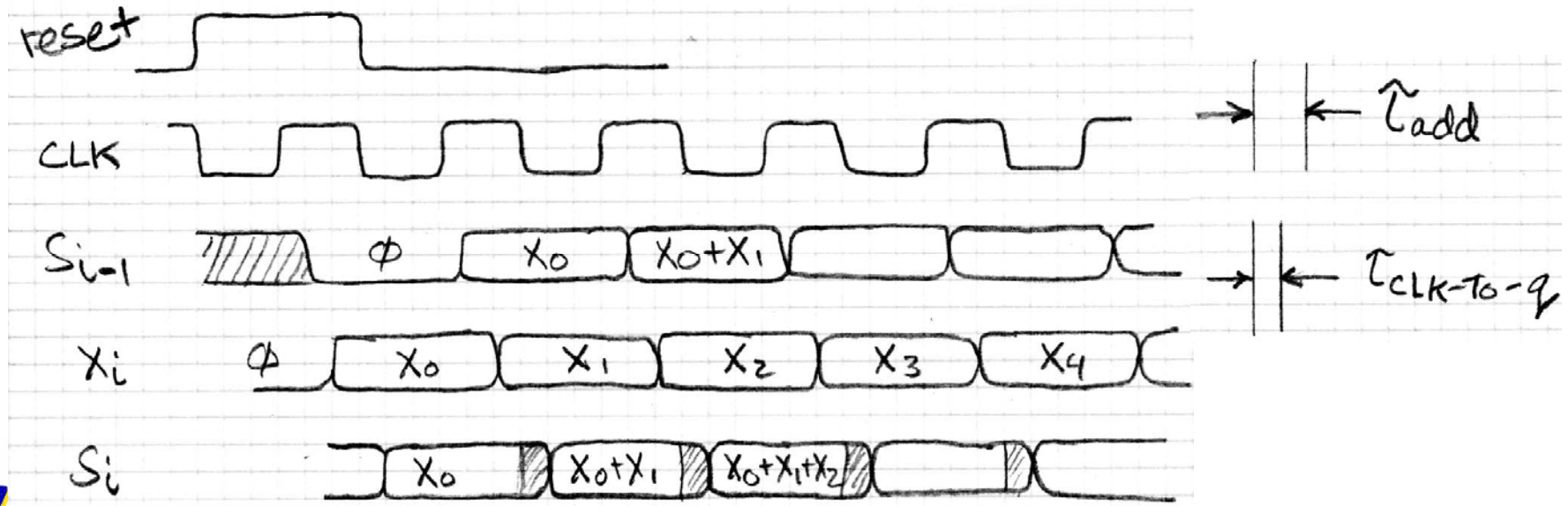
Accumulator Revisited (proper timing 1/2)





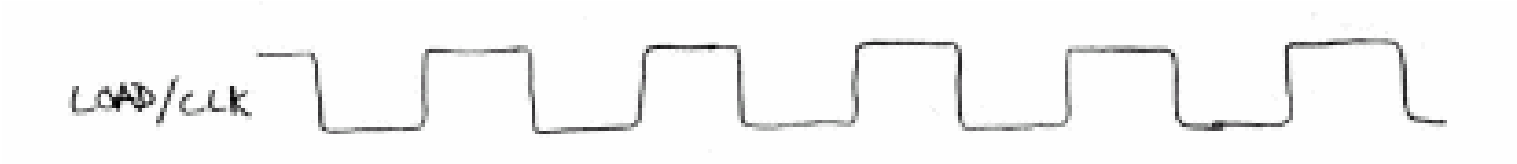
Accumulator Revisited (proper timing 2/2)





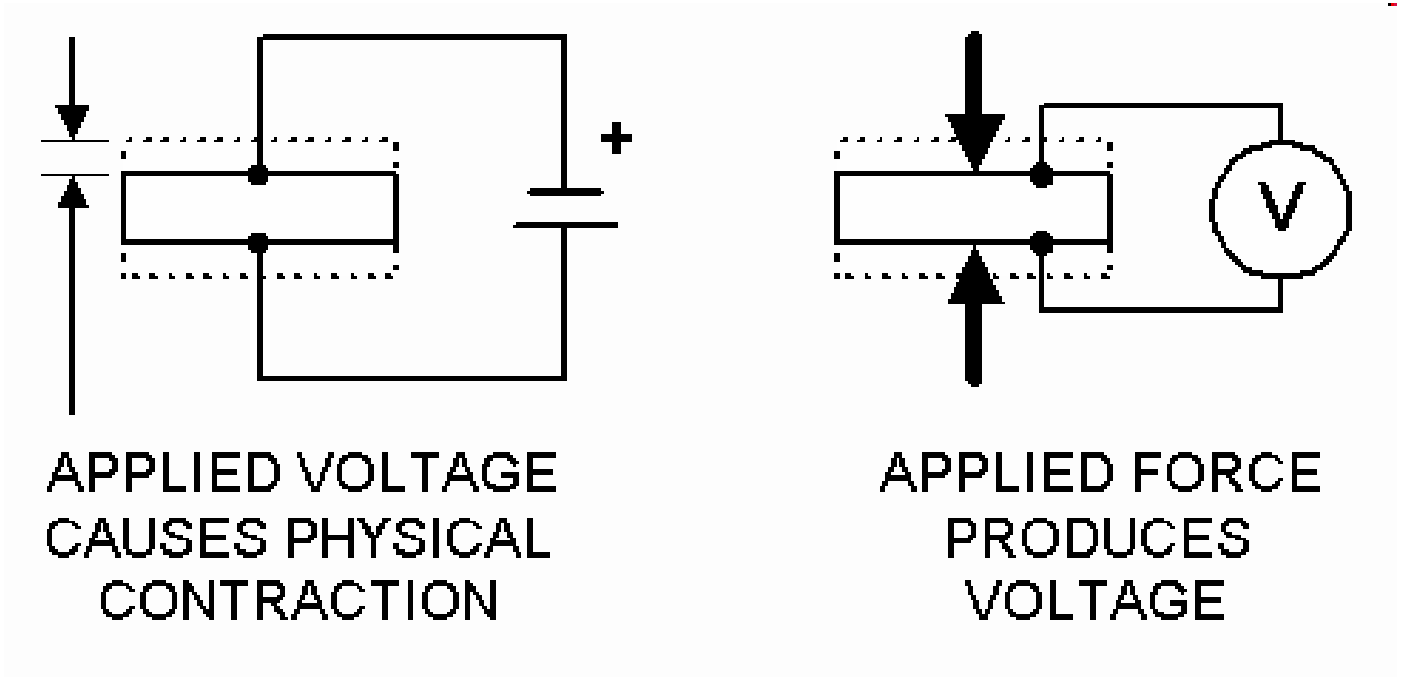
Clocks

- Need a regular oscillator:



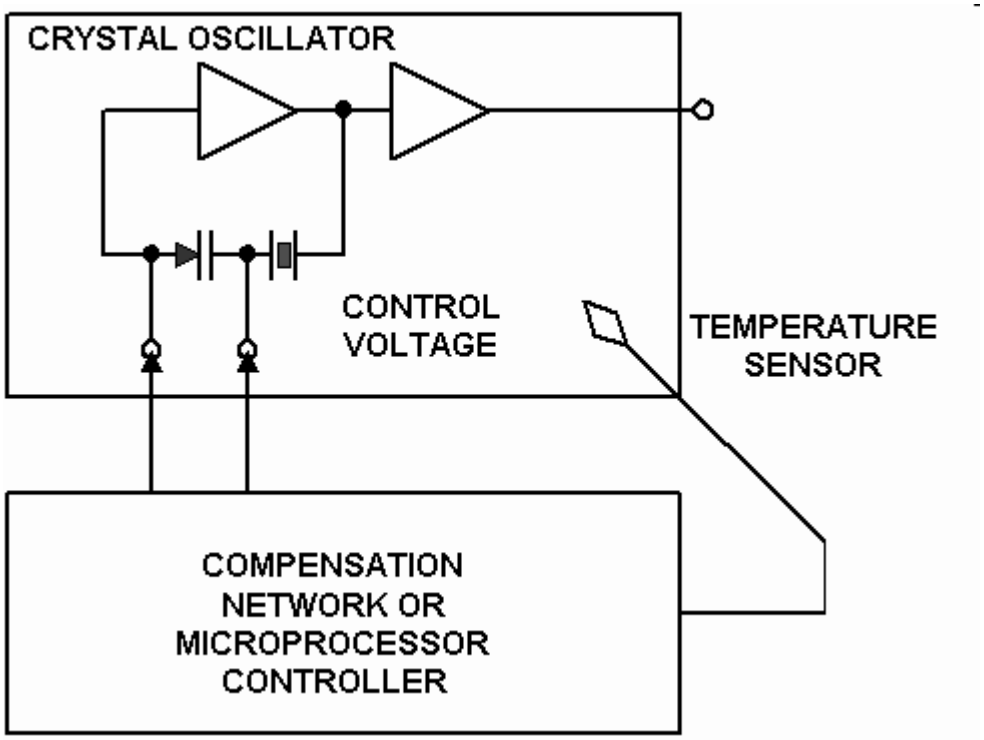
- Wire up three inverters in feedback?…
  - Not stable enough
  - 1->0 and 0->1 transitions not symmetric.
- Solution: Base oscillation on a natural resonance. But of what?

Clocks



- **Crystals and the Piezoelectric effect:**
  - **Voltage → deformation → voltage → …**
  - **Deformations have a resonant freq.**
    - **Function of crystal cut**

Clocks



- Controller puts AC across crystal:
  - At anything but resonant freqs → destructive interference
  - Resonant freq → CONSTRUCTIVE!

Signals and Waveforms: Clocks