

CS61C : Machine Structures

Lecture 4.1.2

State and FSMs

2004-07-13

Kurt Meinz

inst.eecs.berkeley.edu/~cs61c



CS 61C L4.1.2 State (1)

K. Meinz, Summer 2004 © UCB

Outline

- CL Blocks
- Waveforms
- State
- Clocks
- FSMs

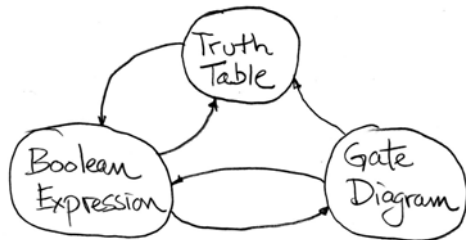


CS 61C L4.1.2 State (2)

K. Meinz, Summer 2004 © UCB

Review (1/3)

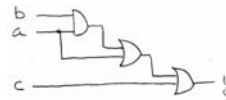
- Use this table and techniques we learned to transform from 1 to another



CS 61C L4.1.2 State (3)

K. Meinz, Summer 2004 © UCB

(2/3): Circuit & Algebraic Simplification



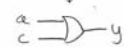
original circuit

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

equation derived from original circuit

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

algebraic simplification



simplified circuit



CS 61C L4.1.2 State (4)

K. Meinz, Summer 2004 © UCB

(3/3): Laws of Boolean Algebra

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



CS 61C L4.1.2 State (5)

K. Meinz, Summer 2004 © UCB

CL Blocks

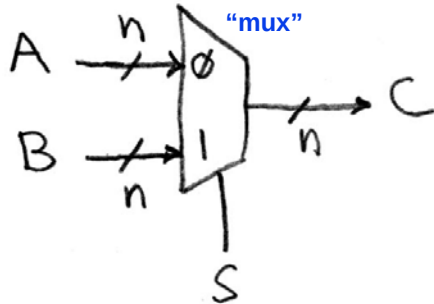
- Let's use our skills to build some CL blocks:
 - Multiplexer (mux)
 - Adder
 - ALU



CS 61C L4.1.2 State (6)

K. Meinz, Summer 2004 © UCB

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

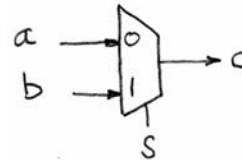


Cal

CS 61C L4.1.2 State (7)

K. Mehta, Summer 2004 © UCB

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

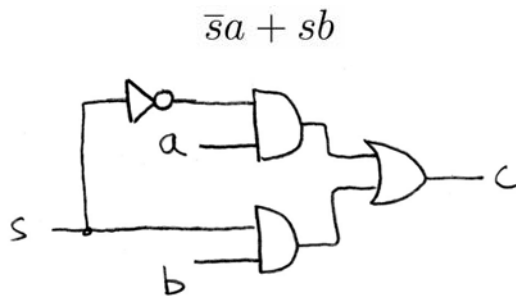
$$\begin{aligned}
 c &= \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab \\
 &= \bar{s}(a\bar{b} + ab) + s(\bar{a}b + ab) \\
 &= \bar{s}(a(\bar{b} + b)) + s((\bar{a} + a)b) \\
 &= \bar{s}(a(1)) + s((1)b) \\
 &= \bar{s}a + sb
 \end{aligned}$$

Cal

CS 61C L4.1.2 State (8)

K. Mehta, Summer 2004 © UCB

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

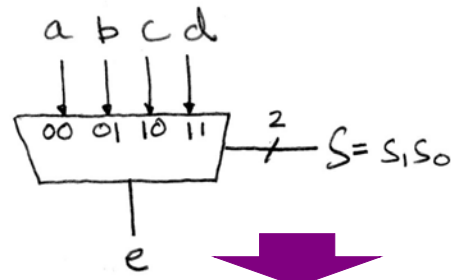


Cal

CS 61C L4.1.2 State (9)

K. Mehta, Summer 2004 © UCB

4-to-1 Multiplexor?



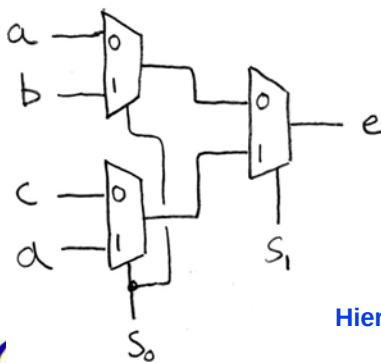
$$e = \bar{s}_1\bar{s}_0a + \bar{s}_1s_0b + s_1\bar{s}_0c + s_1s_0d$$

Cal

CS 61C L4.1.2 State (10)

K. Mehta, Summer 2004 © UCB

An Alternative Approach



Hierarchically!

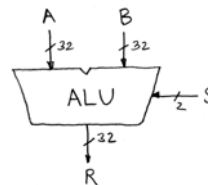
Cal

CS 61C L4.1.2 State (11)

K. Mehta, Summer 2004 © UCB

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



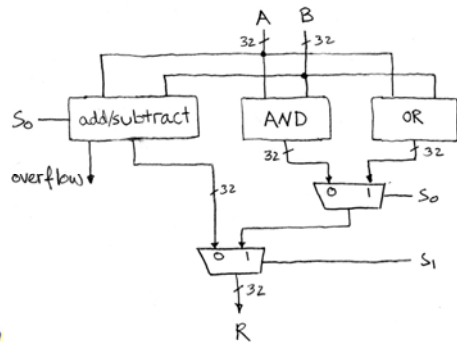
when $S=00$, $R=A+B$
 when $S=01$, $R=A-B$
 when $S=10$, $R=A \text{ AND } B$
 when $S=11$, $R=A \text{ OR } B$

Cal

CS 61C L4.1.2 State (12)

K. Mehta, Summer 2004 © UCB

Our simple ALU



CS 61C L4.1.2 Slide (13)

K. Mehta, Summer 2004 © UCB

Adder/Subtractor Design -- how?

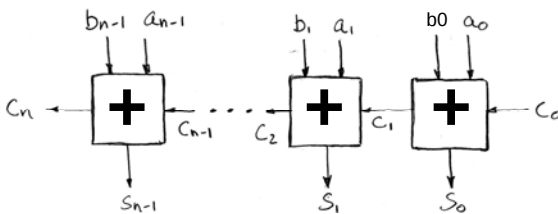
- Truth-table, then determine canonical form, then minimize and implement as we've seen before
- Look at breaking the problem down into smaller pieces that we can cascade or hierarchically layer



CS 61C L4.1.2 Slide (14)

K. Mehta, Summer 2004 © UCB

N 1-bit adders $\Rightarrow$ 1 N-bit adder



CS 61C L4.1.2 Slide (15)

K. Mehta, Summer 2004 © UCB

Adder/Subtractor – One-bit adder LSB...

	a ₃	a ₂	a ₁	a ₀		a ₀	b ₀	s ₀	c ₁
+	b ₃	b ₂	b ₁	b ₀		0	0	0	0
	s ₃	s ₂	s ₁	s ₀		0	1	1	0
						1	0	1	0
						1	1	0	1

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

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



CS 61C L4.1.2 Slide (16)

K. Mehta, Summer 2004 © UCB

Adder/Subtractor – One-bit adder (1/2)...

	a ₃	a ₂	a ₁	a ₀		a _i	b _i	c _i	s _i	c _{i+1}
+	b ₃	b ₂	b ₁	b ₀		0	0	0	0	0
	s ₃	s ₂	s ₁	s 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

$$s_i = \text{XOR}(a_i, b_i, c_i)$$

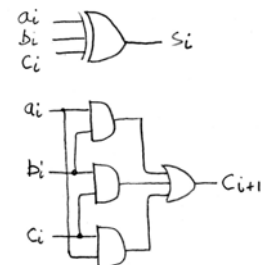
$$c_{i+1} = \text{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i$$



CS 61C L4.1.2 Slide (17)

K. Mehta, Summer 2004 © UCB

Adder/Subtractor – One-bit adder (2/2)...



$$s_i = \text{XOR}(a_i, b_i, c_i)$$

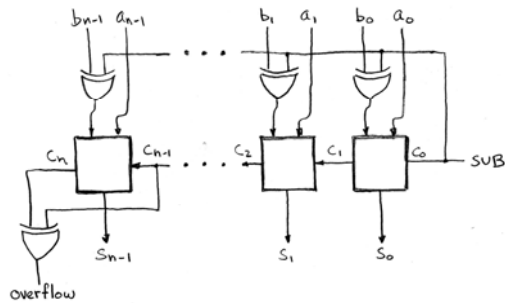
$$c_{i+1} = \text{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i$$



CS 61C L4.1.2 Slide (18)

K. Mehta, Summer 2004 © UCB

Extremely Clever Subtractor



Cal

CS 61C L4.1.2 State (19)

K. Meina, Summer 2004 © UCB

Signals and Waveforms (1/4)

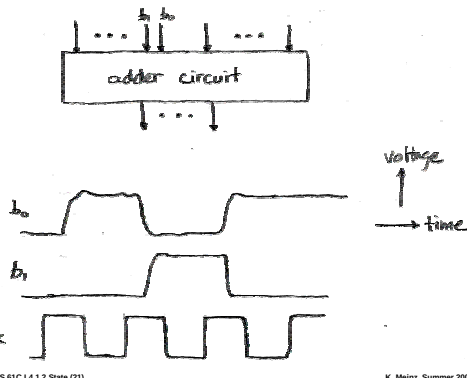
- Outputs of CL change over time
 - With what? → Change in inputs
- Can graph changes with waveforms ...

Cal

CS 61C L4.1.2 State (20)

K. Meina, Summer 2004 © UCB

Signals and Waveforms (2/4): Adders

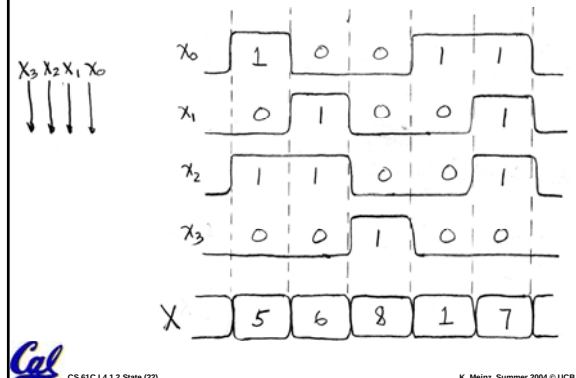


Cal

CS 61C L4.1.2 State (21)

K. Meina, Summer 2004 © UCB

Signals and Waveforms (3/4): Grouping

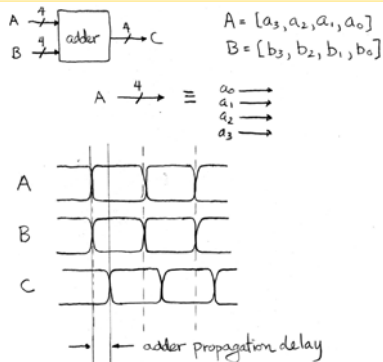


Cal

CS 61C L4.1.2 State (22)

K. Meina, Summer 2004 © UCB

Signals and Waveforms (4/4): Circuit Delay



Cal

CS 61C L4.1.2 State (23)

K. Meina, Summer 2004 © UCB

State

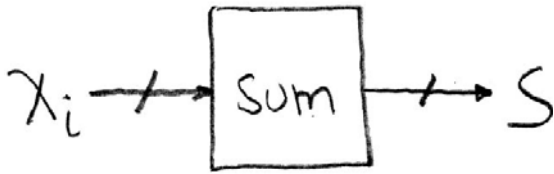
- With CL, output is always a function of **CURRENT** input
 - With some (variable) propagation delay
- Clearly, we need a way to introduce state into computation

Cal

CS 61C L4.1.2 State (24)

K. Meina, Summer 2004 © UCB

Accumulator Example



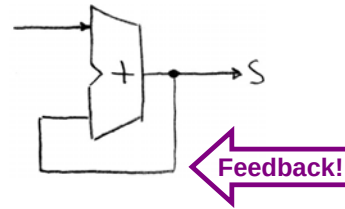
Want: $S=0$; for i from 0 to $n-1$
 $S = S + X_i$



CS 61C L4.1.2 State (25)

K. Mehta, Summer 2004 © UCB

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: ' $S=0$ '?

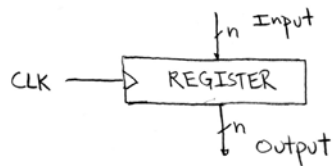
Need a way to store partial sums! ...



CS 61C L4.1.2 State (26)

K. Mehta, Summer 2004 © UCB

Circuits with STATE (e.g., register)



Need a Logic Block that will:

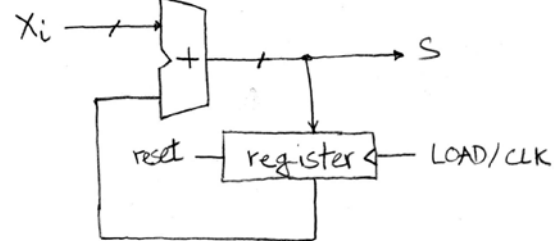
1. store output (partial sum) for a while,
2. until we tell it to update with a new value.



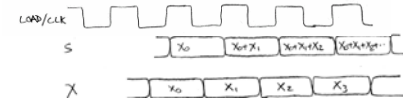
CS 61C L4.1.2 State (27)

K. Mehta, Summer 2004 © UCB

Second try...How about this? Yep!



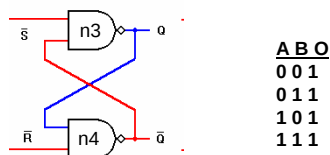
Rough timing...



CS 61C L4.1.2 State (28)

K. Mehta, Summer 2004 © UCB

State



A	B	Q
0	0	1
0	1	1
1	0	1
1	1	1

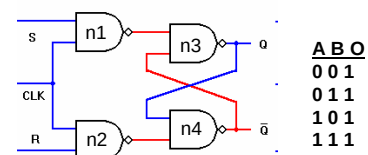
- $S=1, R=0 \Rightarrow Sbar=0, Rbar=1 \Rightarrow Q=1 \Rightarrow Qbar=0$
- $S=0, R=1 \Rightarrow Sbar=1, Rbar=0 \Rightarrow Qbar=1 \Rightarrow Q=0$
- $S=0, R=0 \Rightarrow Sbar=1, Rbar=1 \Rightarrow Qbar \leq \text{not } Q$
 $Q \leq \text{not } Qbar$



CS 61C L4.1.2 State (29)

K. Mehta, Summer 2004 © UCB

State



A	B	Q
0	0	1
0	1	1
1	0	1
1	1	1

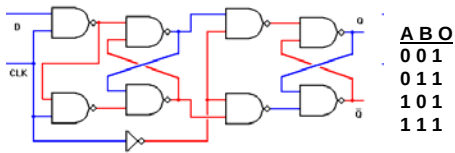
- When CLK is low:
 - $n1$ and $n2 = 1 \Rightarrow$ no change
- When CLK is high:
 - $S, R = 0 \Rightarrow n1, n2 = 1 \Rightarrow$ no change
 - $S=1, R=0 \Rightarrow n1=0, n2=1 \Rightarrow Q=1, Qbar=0$



CS 61C L4.1.2 State (30)

K. Mehta, Summer 2004 © UCB

State



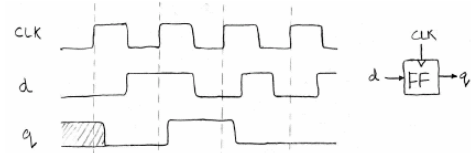
- This is a “rising-edge D Flip-Flop”
 - When the CLK transitions from 0 to 1 (rising edge) ...
 - $Q \leftarrow D$; $Qbar \leftarrow \text{not } D$
 - All other times: $Q \leftarrow Q$; $Qbar \leftarrow Qbar$



CS 61C L4.1.2 State (31)

K. Mehta, Summer 2004 © UCB

D Flip Flop



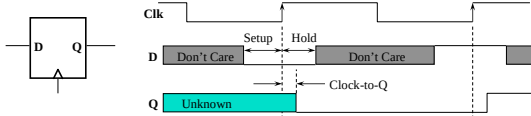
- Called “edge-sensitive”
- RS latches: “level sensitive”



CS 61C L4.1.2 State (32)

K. Mehta, Summer 2004 © UCB

Storage Element's Timing Model



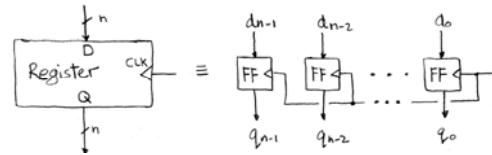
- Setup Time: Input must be stable BEFORE trigger clock edge
- Hold Time: Input must REMAIN stable after trigger clock edge
- Clock-to-Q time:
 - Output cannot change instantaneously at the trigger clock edge
 - Similar to delay in logic gates



CS 61C L4.1.2 State (33)

K. Mehta, Summer 2004 © UCB

Bus a bunch of D FFs together ...



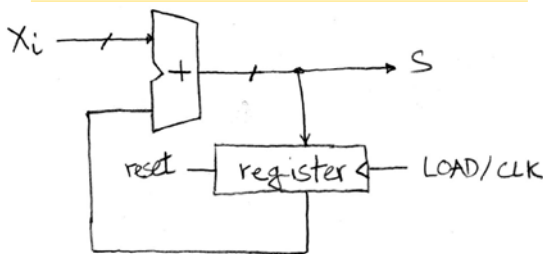
- Register of size N:
 - n instances of D Flip-Flop



CS 61C L4.1.2 State (34)

K. Mehta, Summer 2004 © UCB

Second try...How about this? Yep!



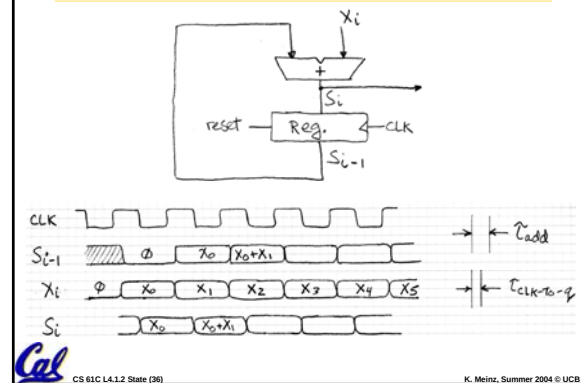
Rough timing...



CS 61C L4.1.2 State (35)

K. Mehta, Summer 2004 © UCB

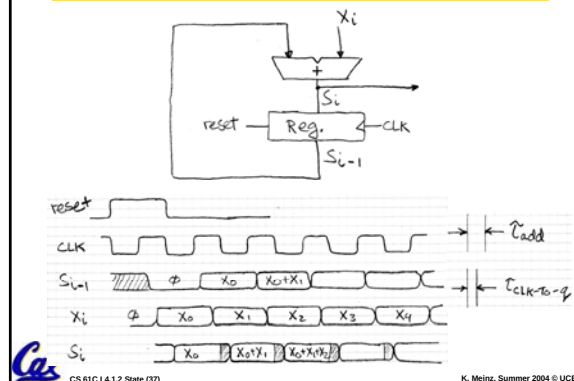
Accumulator Revisited (proper timing 1/2)



CS 61C L4.1.2 State (36)

K. Mehta, Summer 2004 © UCB

Accumulator Revisited (proper timing 2/2)



Clocks

- Need a regular oscillator:

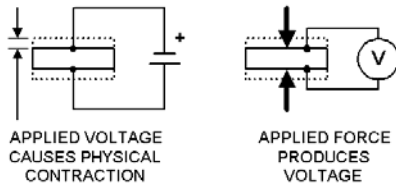


- Wire up three inverters in feedback?...
 - Not stable enough
 - 1->0 and 0->1 transitions not symmetric.

- Solution: Base oscillation on a natural resonance. But of what?

CS 61C L4.1.2 State (38) K. Meina, Summer 2004 © UCB

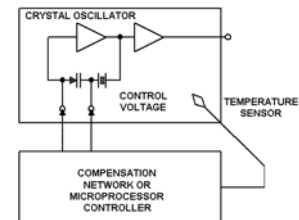
Clocks



- Crystals and the Piezoelectric effect:
 - Voltage $\rightarrow$ deformation $\rightarrow$ voltage $\rightarrow$...
 - Deformations have a resonant freq.
 - Function of crystal cut

CS 61C L4.1.2 State (39) K. Meina, Summer 2004 © UCB

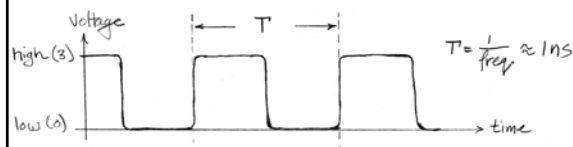
Clocks



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

CS 61C L4.1.2 State (40) K. Meina, Summer 2004 © UCB

Signals and Waveforms: Clocks



CS 61C L4.1.2 State (41) K. Meina, Summer 2004 © UCB