B38DB: Digital Design and Programming
Datapath Components – Adders and Comparators

Mustafa Suphi Erden

Heriot-Watt University
School of Engineering & Physical Sciences
Electrical, Electronic and Computer Engineering
Room: EM 2.01
Phone: 0131-4514159
E-mail: m.s.erden@hw.ac.uk

Adders

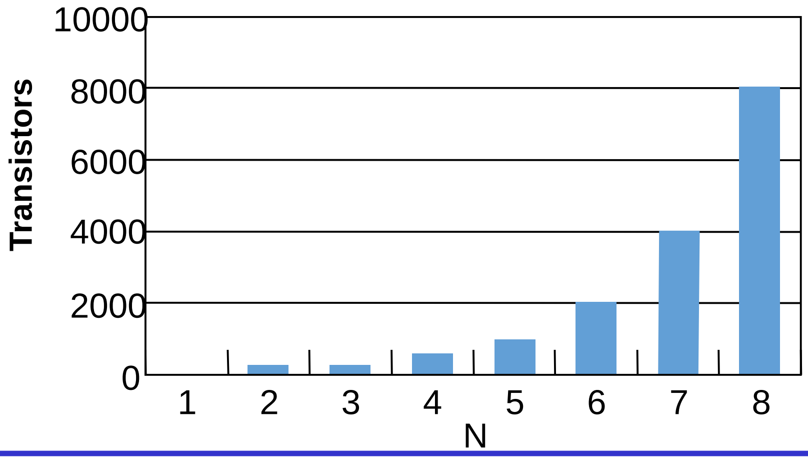
- Adds two N-bit binary numbers
  - 2-bit adder: adds two 2-bit numbers, outputs 3-bit result
  - e.g., 01 + 11 = 100   (1 + 3 = 4)
- We can design using combinational design process, but doesn't work well for reasonable-size N.
  - Why not?

Inputs				Outputs		
a1	a0	b1	b0	c	s1	s0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Why Adders Aren't Built Using Standard Combinational Design Process

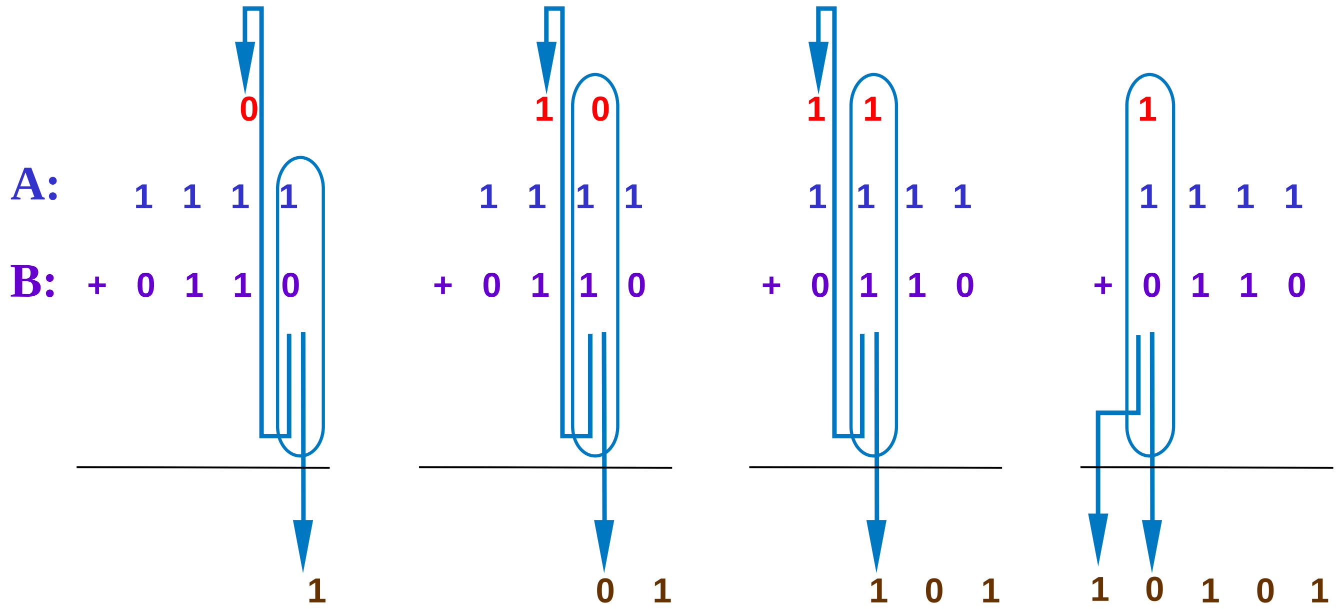
- Truth table too big
  - 2-bit adder's truth table shown
    - Has $2^{(2+2)}$ = 16 rows
  - 8-bit adder: $2^{(8+8)}$ = 65,536 rows
  - 16-bit adder: $2^{(16+16)}$ = ~4 billion rows
  - 32-bit adder: ...
  - Plot shows number of transistors for N-bit adders, using state-of-the-art automated combinational design tool

Inputs				Outputs		
a1	a0	b1	b0	c	s1	s0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0



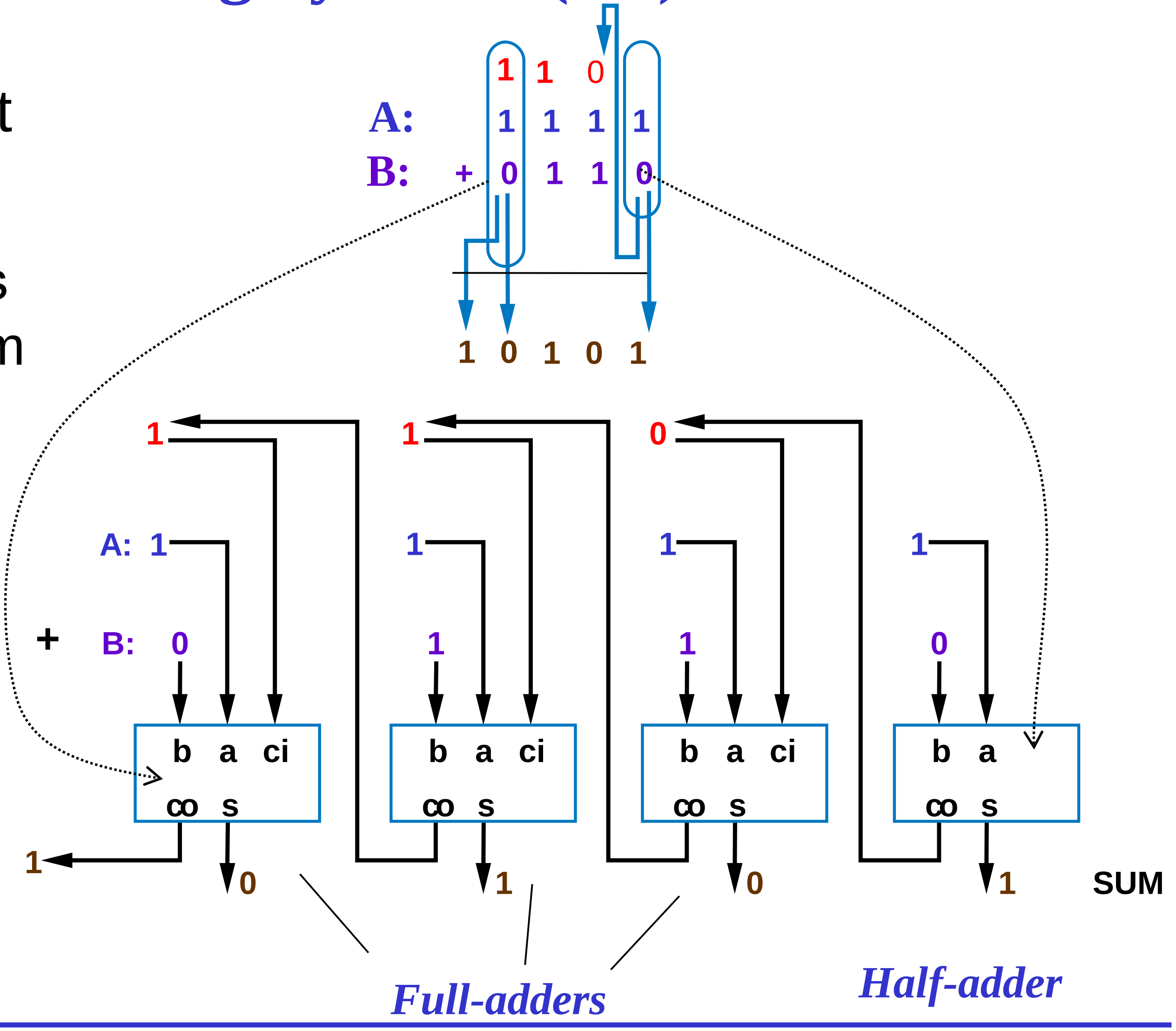
Alternative Method to Design an Adder: Imitate Adding by Hand (1/2)

- Alternative adder design: mimic how people do addition by hand
- One column at a time
  - Compute sum, add carry to next column



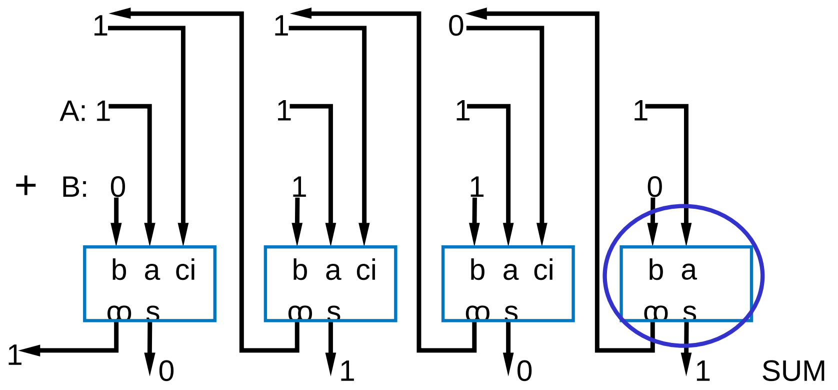
Alternative Method to Design an Adder: Imitate Adding by Hand (2/2)

- Create a component for each column
  - Add that column's bits, generate sum and carry bits



Half-Adder

- ***Half-adder***: Adds 2 bits, generates sum and carry
- Design using combinational design process as we did before



<u>Step 1: Capture the function</u>

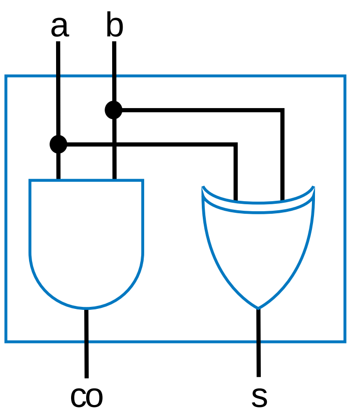
Inputs		Outputs	
a	b	co	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

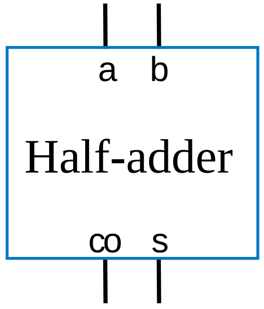
<u>Step 2: Convert to equations</u>

co = ab

s = a'b + ab' (same as s = a xor b; a ⊕ b)

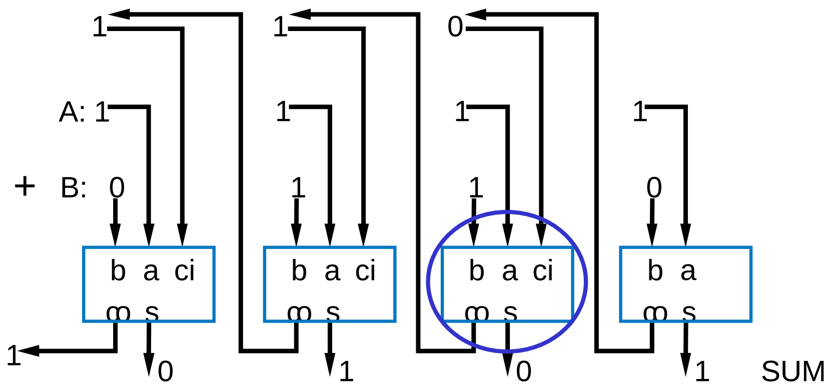
<u>Step 3: Create the circuit</u>





Full-Adder

- ***Full-adder:*** Adds 3 bits, generates sum and carry
- Design using combinational design process



Step 1: Capture the function

Inputs			Outputs	
a	b	ci	co	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step 2: Convert to equations

co = a’bc + ab’c + abc’ + abc

co = a’bc +abc +ab’c +abc +abc’ +abc

co = (a’+a)bc + (b’+b)ac + (c’+c)ab

co = bc + ac + ab

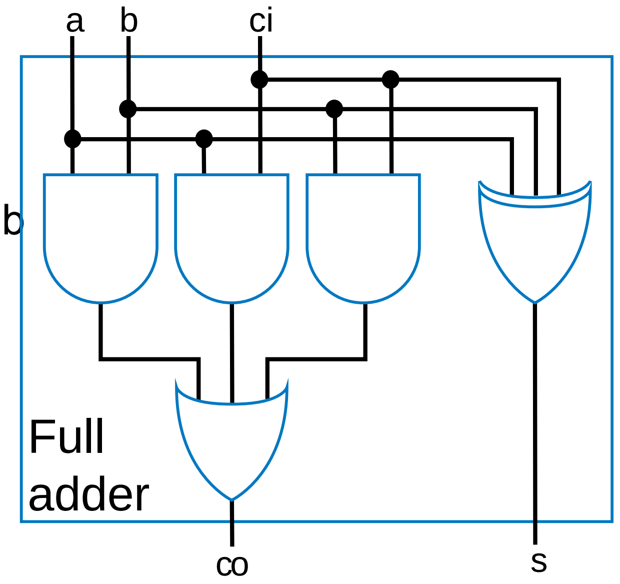
s = a’b’c + a’bc’ + ab’c’ + abc

s = a’(b’c + bc’) + a(b’c’ + bc)

s = a’(b xor c)’ + a(b xor c)

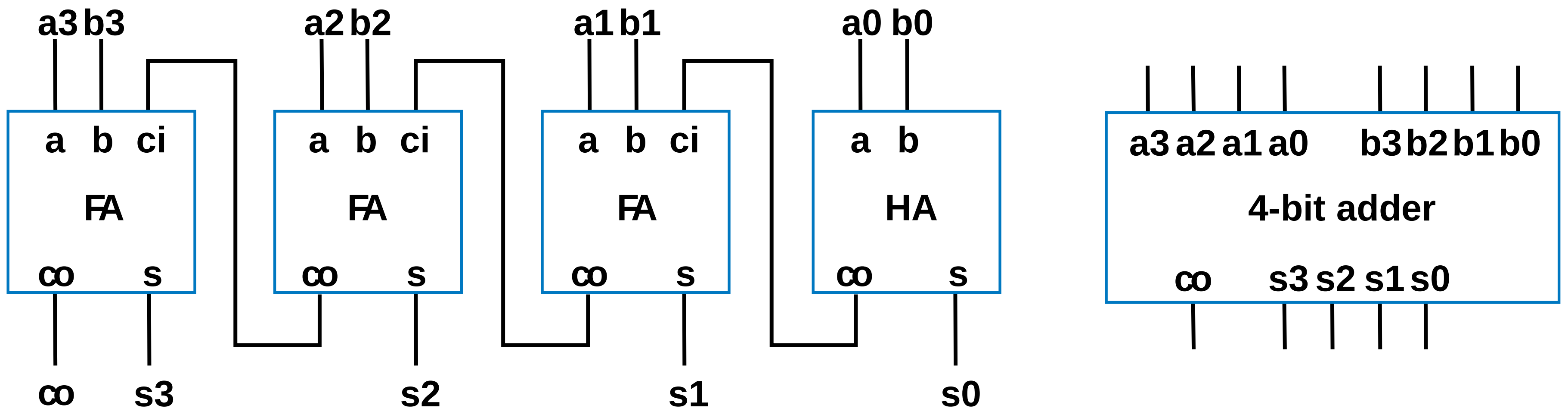
s = a xor b xor c

Step 3: Create the circuit



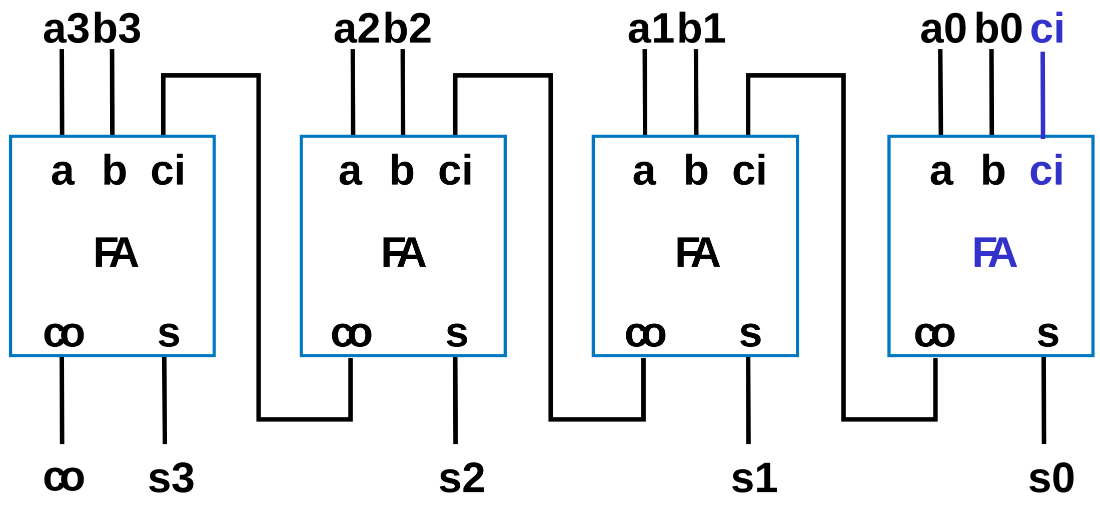
Carry-Ripple Adder (1/2)

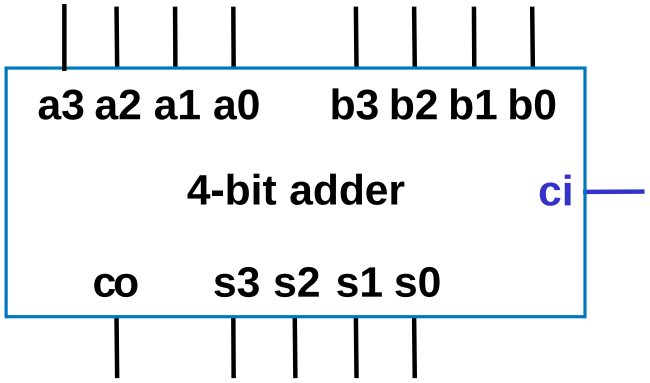
- Using half-adder and full-adders, we can build an adder that adds like we would do by hand.
- Called a ***carry-ripple adder***
  - 4-bit adder shown: Adds two 4-bit numbers, generates 5-bit output
    - 5-bit output can be considered 4-bit "sum" plus 1-bit "carry out"
  - Can easily build any size adder



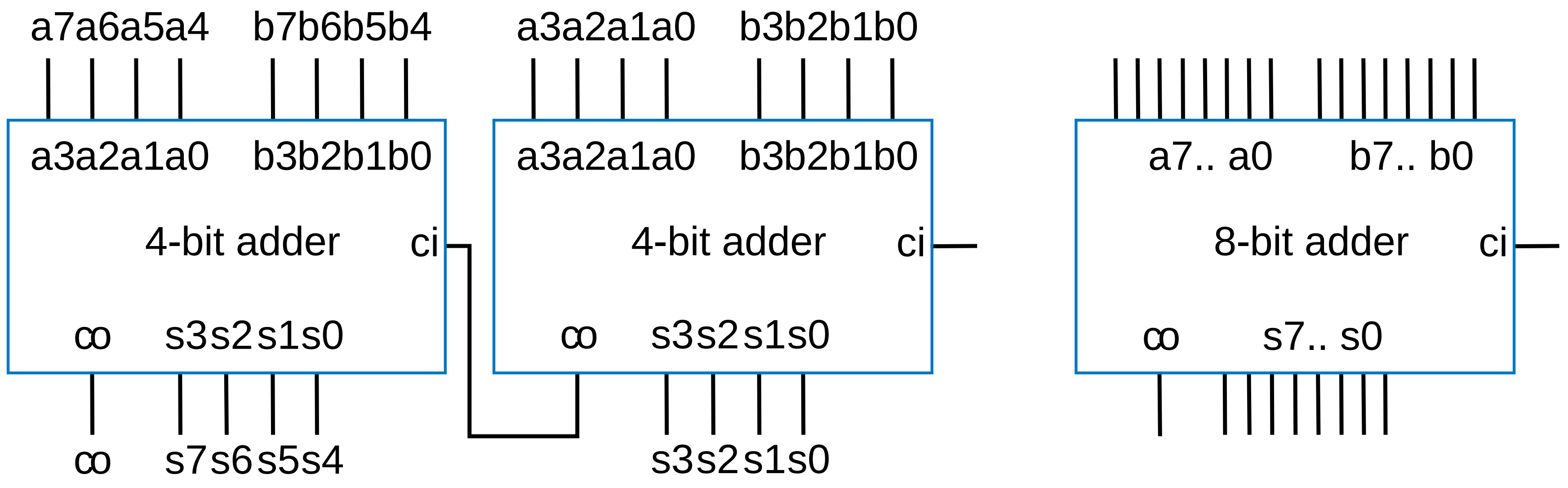
Carry-Ripple Adder (2/2)

- Using a full-adder instead of a half-adder for the first bit, we can include a “carry in” bit in the addition
    - Useful to connect smaller adders to form bigger adders



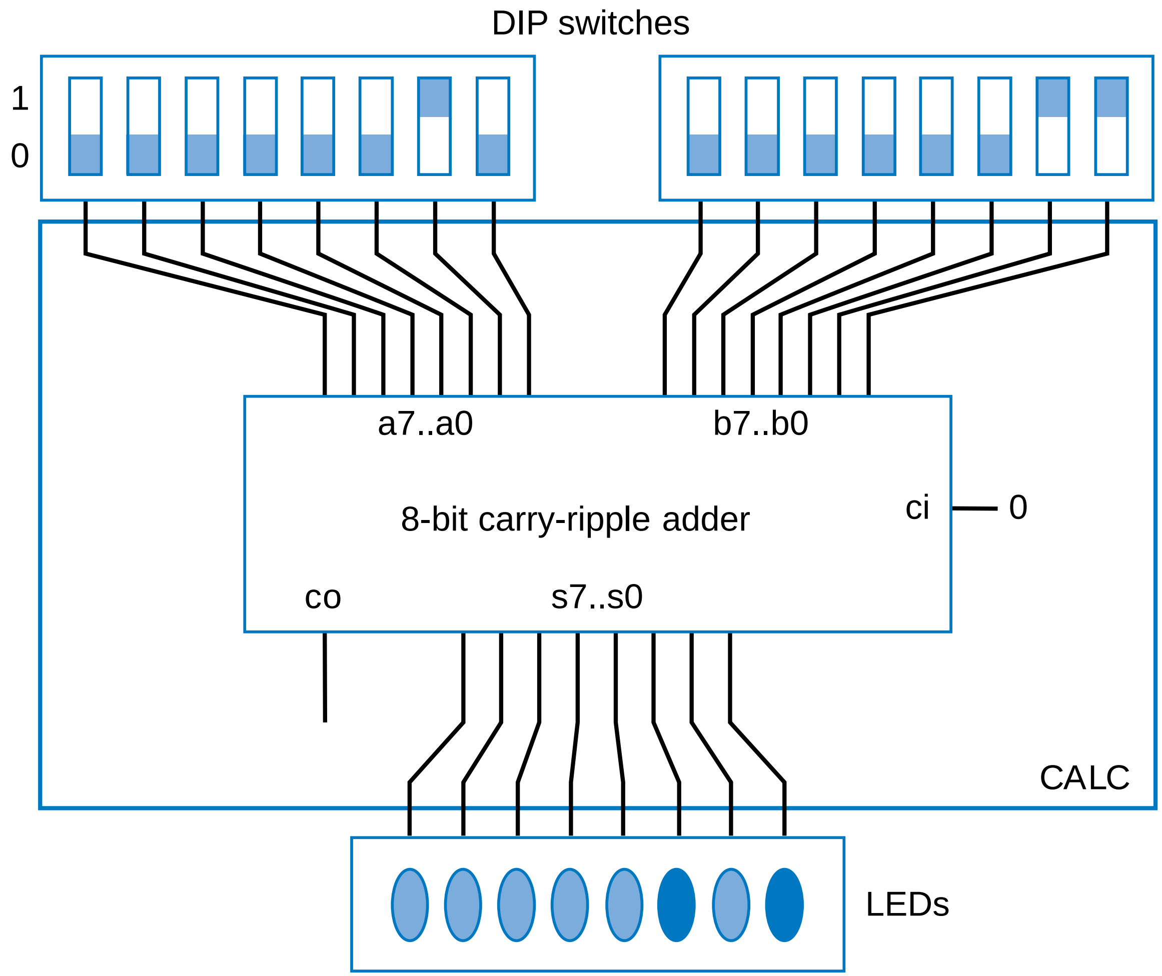


Cascading Adders



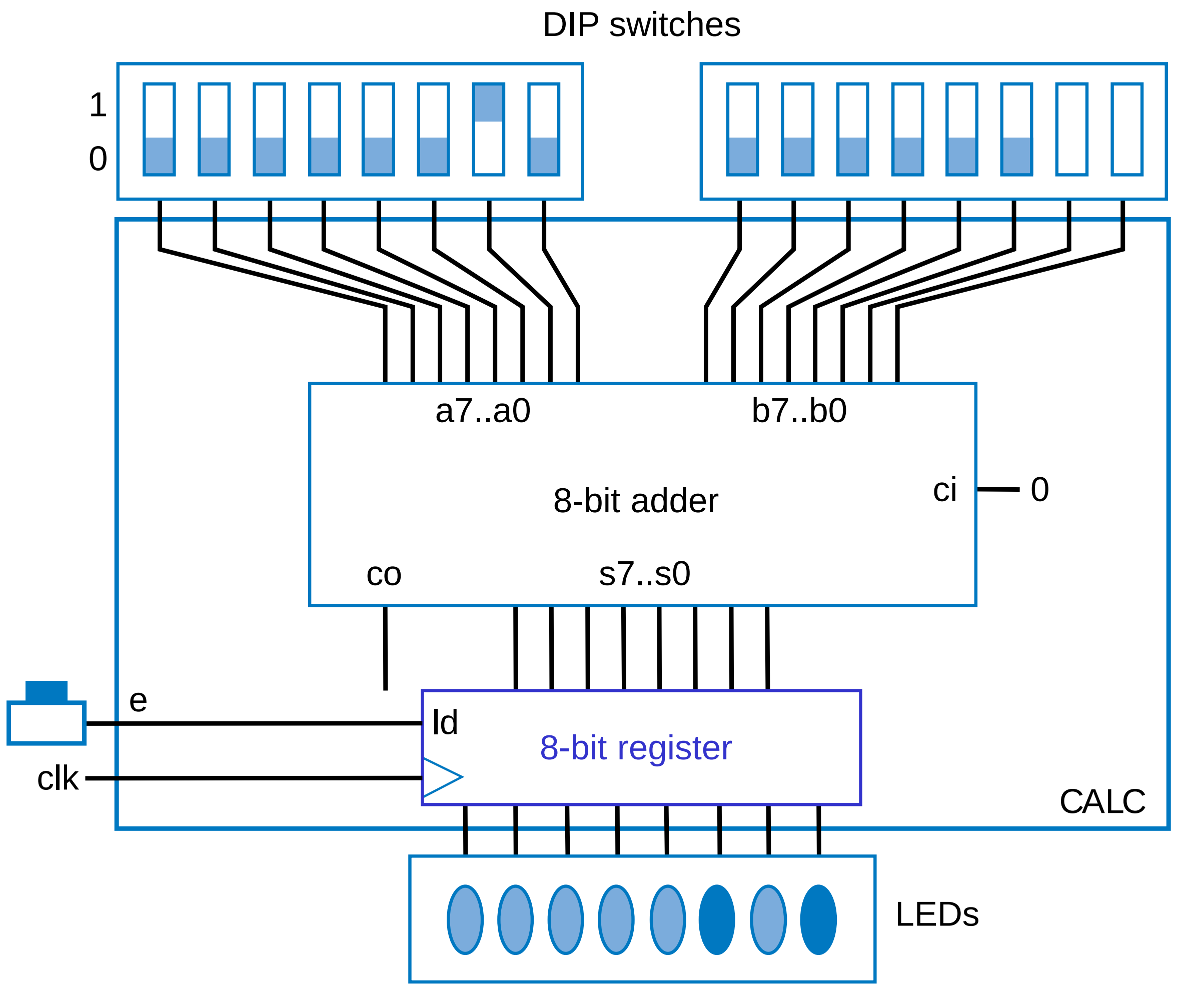
Adder Example: DIP-Switch-Based Adding Calculator (1/2)

- Calculator that adds two 8-bit binary numbers, specified using DIP switches



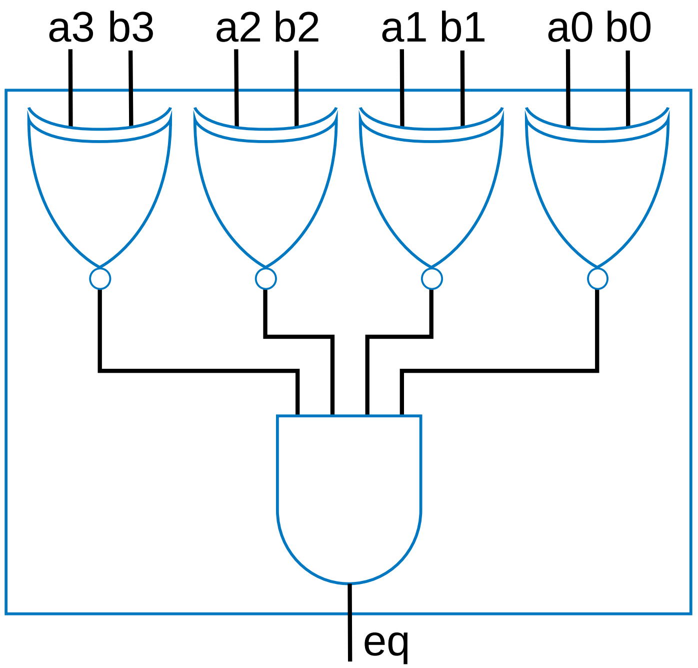
Adder Example: DIP-Switch-Based Adding Calculator (2/2)

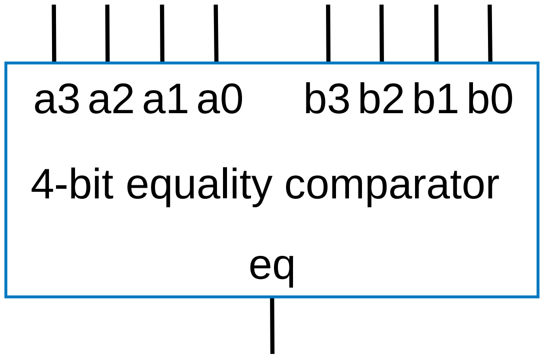
- To prevent spurious values from appearing at the output while data propagation through the carry-ripple adder's gates, we can place a register at the output.



Comparators

- ***N-bit equality comparator***: Outputs 1 if two N-bit numbers are equal
  - 4-bit equality comparator with inputs A and B
    - Recall that XNOR outputs 1 if its two input bits are the same
      - eq = (a3 ⊗ b3) * (a2 ⊗ b2) * (a1 ⊗ b1) * (a0 ⊗ b0)





Magnitude Comparator (1/3)

- ***N-bit magnitude comparator***: Indicates whether

  A>B, A=B, or A<B, for its two N-bit inputs A and B

  - How to design?

A=1011 B=1001

1011 **1**001 Equal

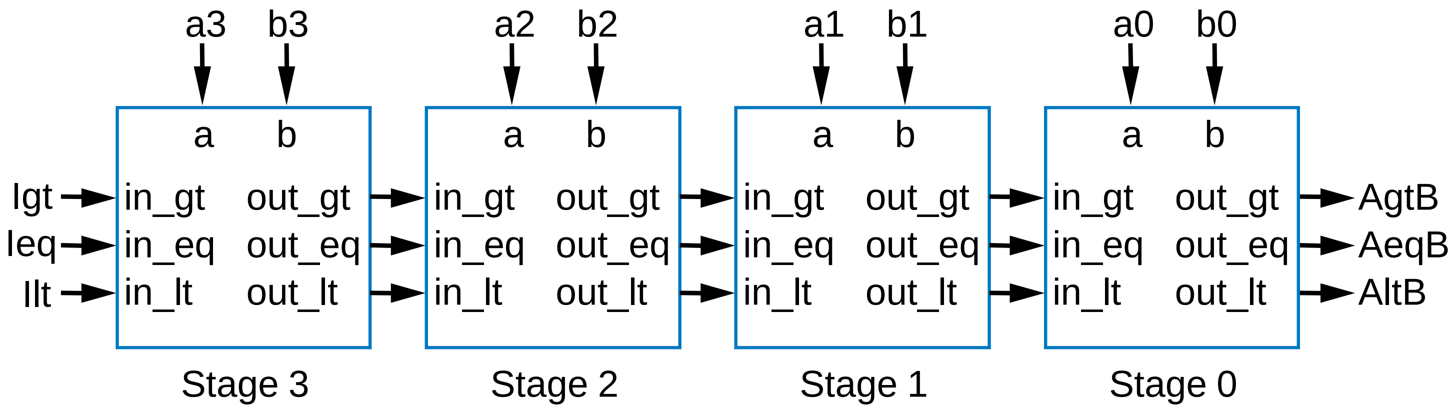
1**0**11 1**0**01 Equal

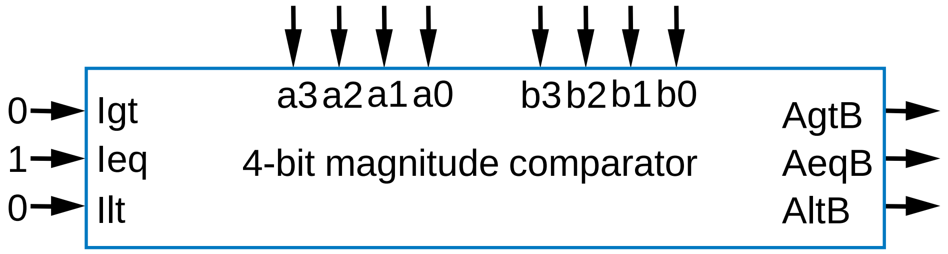
10**1**1 10**0**1 **Unequal**

So A > B

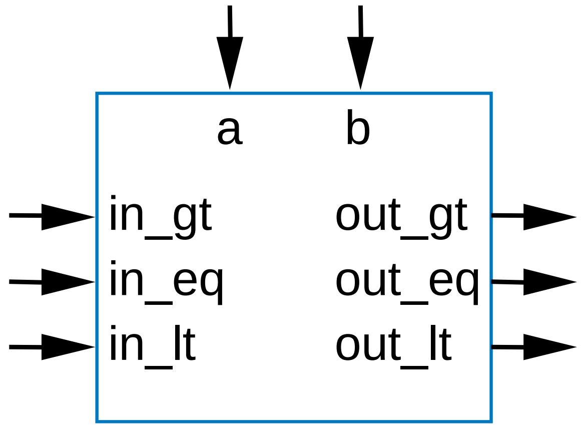
Magnitude Comparator (2/3)

- Start at left, compare each bit pair, pass results to the right
- Each stage has 3 inputs indicating results of higher stage, passes results to lower stage





Magnitude Comparator (3/3)



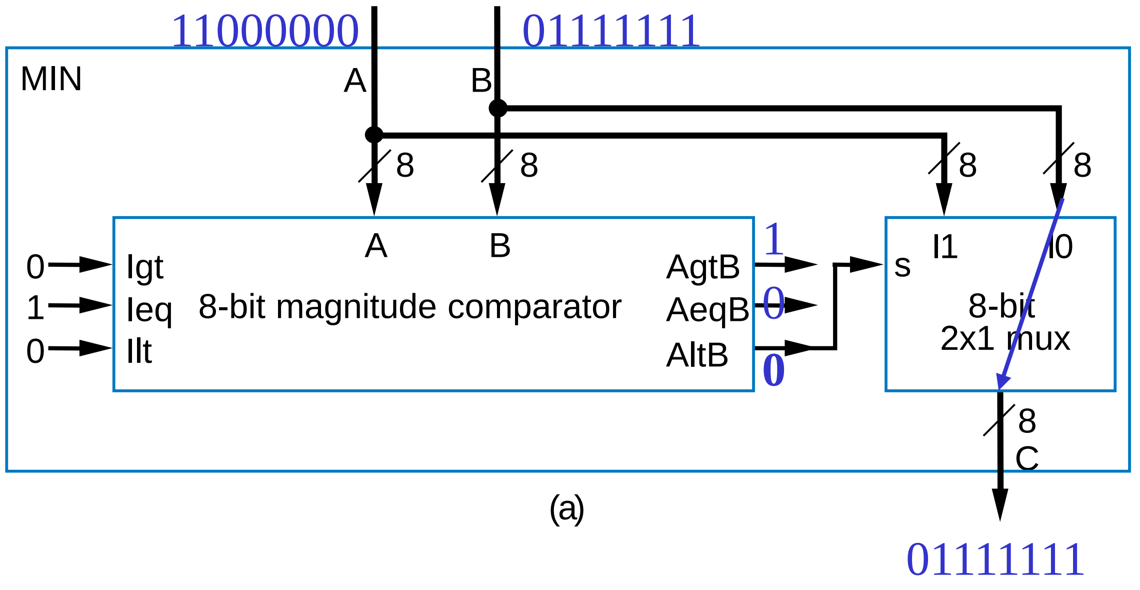
- Each stage:

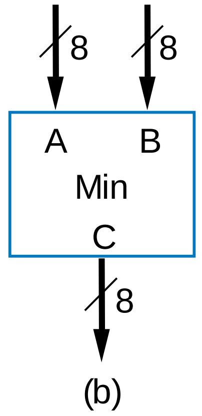
  - out_gt = in_gt **OR** (in_eq **AND** a **AND** b')
  - out_lt = in_lt **OR** (in_eq **AND** a' **AND** b)
  - out_eq = in_eq **AND** (a **XNOR** b)

  - Simple circuit inside each stage, just a few gates (not shown)

Magnitude Comparator Example: Minimum of Two Numbers

- Design a combinational component that computes the minimum of two 8-bit numbers
  - Solution: Use 8-bit magnitude comparator and 8-bit 2x1 mux
    - If A<B, pass A through mux. Else, pass B.


11000000
01111111
MIN
A
B
8
8
8
8
0
1
0
Igt
Ieq
Ilt
A
B
8-bit magnitude comparator
AgtB
AeqB
AltB
1
0
0
s
I1
I0
8-bit
2x1 mux
8
C
01111111
(a)


8
8
A
B
Min
C
8
(b)